Social Distancing Detection

Abstract

Introduction

Code Components

There are two main components to the program: the setup, which only occurs once in the beginning, and the operation, which is a loop that occurs once for each frame of the input video.

When the program begins running, the first frame of the input video is shown to the user. The user then inputs six points with their mouse. The first four points make up a quadrilateral that approximates a rectangle on the ground plane, which will be referred to as the “region of interest” or ROI. The last two points approximate a six-foot distance on the ground. If this program were to be used in a real-world setting, there would ideally be distance markers on the ground, objects of known dimensions, or other features in the image such as road lines and sidewalks to aid the user in accurately plotting the six points.

The purpose of creating a region of interest with the first four mouse points is to solve the issue of camera distortion. Because the camera is filming from an angle, the conversion rate between physical distance on the ground and pixel distance in the image is not constant. In other words, the number of pixels that make up a six-foot distance on the ground changes depending on the distance from the camera. The solution to this problem is to use the OpenCV function “cv2.getPerspectiveTransform.” The function takes as arguments the coordinates of the first four mouse points as the source and the dimensions of the frame as the destination and outputs a 3x3 transformation matrix. This matrix is then used to warp, or change the perspective of the last two mouse points using the function “cv2.perspectiveTransform,” giving the two points new (x, y) coordinates. The distance formula is then used to calculate the Euclidian distance between these two new coordinates, and the result, known as the minimum safe distance, is a float value that represents the number of pixels that make up a six-foot distance on the ground plane. Because the two points are warped using the transformation matrix before calculating the distance between them, this pixel value is constant for any six-foot distance, regardless of the distance from the camera.

As a proof of concept, a small-scale experiment was performed using LEGO figures as human substitutes. Four figures were placed on a piece of graph paper and a rectangle was drawn around them. On the sides of the rectangle, equidistant tick marks were drawn. Images of the figures were taken from angles approximating those of common surveillance footage. This angle caused the tick marks on the side of the rectangle to distort, meaning that the pixel distance between each tick mark decreased as the distance from the camera increased. The pictures were run through the same perspective transformation function described above in order to create the transformation matrix. The drawn rectangle was used as a guide to input the mouse points. The matrix was then used to run the entire image through the function “cv2.warpPerspective,” which warps the entire image rather than specific points. The result of this program was an image in which the LEGO figures looked misshapen and distorted, but the pixel distances between each tick mark on the sides of the rectangle appeared to be equal. This result indicated that a transformation matrix could be used to make the conversion rate between the physical distance on the ground to pixel distance in the image constant. This method will allow for more accurate distance calculations and detections of social distancing violations.

The second component of the code is the operation, which is a while loop that occurs once for each frame of the input video until either the video ends or the user ends the program manually. The first step in the operation loop is person detection, which is accomplished using a real-time object detection program called You Only Look Once, or YOLOv3. YOLO recognizes a wide variety of objects including people, animals, furniture, vehicles, and many other common items. This specific program included a filter so that only human detections were kept. In addition, one of the arguments passed into the program by the user is a confidence value between zero and one that filters out weak and uncertain detections by YOLO. The default value for this argument is 0.5 but it can be changed using the “--confidence” flag when running the program. Once detection occurs, the results are stored as a list of bounding boxes, or rectangles whose coordinates surround the person being detected. The final part of this step is to apply an algorithm called “non-maximum suppression,” which removes all extraneous boxes to ensure that there is only one box for each person. This algorithm takes in a threshold argument between zero and one that is defaulted to 0.3 but can be changed by the user using the “--threshold” flag when running the program.

The next step in the operation loop is to assign unique object IDs to each detected person. In order to do so, the program uses a class called the Centroid Tracker. This class assigns unique object IDs to each bounding box and keeps a register of people that have been detected in previous frames. The class contains an update function which is called every frame. The function takes in a list of bounding boxes that were outputted by YOLO in the current frame and compares these input boxes with the boxes currently in the register. The Euclidian distance is calculated between each pair of input box and registered box in order to associate each input box with an object that has already been detected. The assumption is that the input box that is closest to an already registered object must be the same object, and so that object’s location is updated. If there are more input boxes than boxes in the register, it means that a new detection has occurred, and so the program assigns it a new ID and the object is added to the register. If there are fewer input boxes than boxes in the register, it means that an object from the register was not detected in the most recent frame, and so it is marked “disappeared.” If an object is marked as disappeared for too many consecutive frames, that object is removed from the register. After the update function is called, the results are organized into a list of tuples where the first element is the object ID and the second element is the coordinates and dimensions of that object’s bounding box. Using this method, people detected in the video can be assigned unique object IDs, allowing the user to keep track of specific individuals across multiple frames, rather than analyzing each frame independently.

The third step of the operation loop is calculating the distance between each pair of detected people and marking those who are standing less than six feet from another person. This part of the program employs the aid of a series of helper functions. The first helper function, called “transform\_box\_points,” takes as input the list of object ID – bounding box tuples as well as the transformation matrix from the setup phase. It first calculates the coordinate of the bottom-center point of each bounding box, and then applies the transformation matrix to warp each of these bottom-center points. The function returns a list of tuples where the first element is the object ID and the second element is the new warped coordinate associated with that object’s box.

The second helper function is called “violation\_detection,” and it takes as input the list of object ID – bounding box tuples, the list of object ID – warped coordinate tuples generated from the first helper function, and the minimum safe distance calculated in the setup phase. The function uses the warped coordinates list to calculate the distance between each pair of bounding boxes and compares the distances to the minimum safe distance. Each pair of bounding boxes is then tagged with a Boolean called “safe” which is true if the distance between the two boxes is greater than the minimum safe distance and false if the distance between the two boxes is less than the minimum safe distance. The function returns two outputs: the first is a list of lists where each inner list contains two bottom-point coordinates and the safe Boolean associated with them. The second output is also a list of lists, but here every inner list contains two bounding box coordinates and dimensions and the safe Boolean associated with them. Finally, the first output of this function is then passed through the third helper function, called “get\_violation\_count,” which counts and returns the number of people who are currently violating social distancing guidelines and the number of people who are safe.

PARAGRAPH ABOUT CREATING DISPLAYS

The following is a flowchart that summarizes the steps of the code. The orange boxes represent the setup component and the green boxes represent the operation loop.

Program Usage

The python code for this program can be found here: <https://github.com/ozur1/Social_Distancing_Detector>

How to use the program:

1. Download the YOLO object detector: ./download\_model.sh
2. Install all dependencies: pip install -r requirements.txt
3. Run the program:
   1. Using computer’s webcam: python sdd.py
   2. Using a pre-recorded video: python sdd.py --input (path to video)
4. Other arguments that can be adjusted:
   1. --confidence: float, confidence threshold for yolo detection (default=0.5)
   2. --threshold: float, non-maximum suppression algorithm threshold (default=0.3)
   3. --disappeared: int, number of frames that a registered object ID is not detected before it is de-registered (default=10)
   4. --frames: int, number of frames between data outputs to text file (default=20)
5. Follow the instructions on the image that pops up to input the six mouse points for the region of interest and the six-foot approximation

Next Steps and Conclusion