Social Distancing Detection

Hello there! My name is Ori Zur and I am a rising junior at Northwestern University studying computer science and music composition. For the past 6 months, the world has been enduring a historic pandemic due to the COVID-19 virus. As society attempts to adjust to the new lifestyle of mask wearing, virtual education, and toilet-paper shortages, one term that constantly gets brought up is “social distancing.” Social distancing is the action of keeping a distance of at least six feet from others in a public environment in order to reduce the spread of the Coronavirus disease. With governments and businesses across the country taking measures to aid people in following these new distancing guidelines, an important question arises: what percentage of people are actually following these guidelines, specifically in outdoor urban environments?

This summer at Argonne, I created a means to answer this question by designing and coding a social distancing detector using Python and OpenCV. The program takes a video of pedestrians, typically from surveillance camera footage, and analyzes each frame by detecting the people, calculating the distance between each person, and indicating if any two people are standing less than six feet apart. OpenCV, a computer vision function library, was used because it greatly simplifies the process of loading in a video, separating it into individual frames for analysis and editing, and outputting the final results.

The program has three major components. The first is human detection. This was accomplished using a real-time object detection system called YOLO (You Only Look Once). This system is able to recognize a wide variety of objects including people, animals, vehicles, furniture, and many other common objects. My program included a filter so that only person detections were kept. After person detection occurs, each person is represented by a “bounding box,” a rectangle whose coordinates surround the person.

The second step is image transformation, whose purpose is to solve the problem that when a video is taken from a certain angle, the conversion from distance on the ground to pixels on the screen is not constant. In other words, the number of pixels that make up six feet on the ground depends on the distance from the camera. In order to solve this problem, help from the user is required. When the program begins running, the first window that is outputted is the first frame of the video being used. The user then inputs six points on the image using their mouse. The first four represent a rectangle on the ground plane and are used to warp the frame using an OpenCV function called Perspective Transform. This creates a pseudo “bird’s-eye-view” image where the pixel to real-life distance conversion is constant. The last two points approximate a six-foot distance on the ground. These two points, when passed through the same warping function, yield a length in pixels that is equivalent to six-feet on the ground.

The final step is coordinate mapping and distance calculations. In order to calculate the distance between the people, a single point is taken from each person’s bounding box (in my case the bottom-center point), and is warped using the same function used in the image transformation step. This maps the coordinate of each person from the original image to the warped bird’s-eye-view image. Using these new warped coordinates, the distance formula can be used to calculate the distance between each pair of points, since pixel to feet conversion is now constant. The resulting distances are compared to the minimum safe distance calculated in step two, and the pairs of bounding boxes are sorted into safe and unsafe categories. Finally, red and green rectangles are drawn around the unsafe and safe people respectively and the frame is returned. The final output is the video being used with the red and green rectangles drawn around the people detected, but I also included a second output: a blank white screen containing red and green dots that act as a simulated “bird’s-eye-view” perspective of the video as it plays.