SSD Tracker Pipeline

Prepared by
Muhammad Abdullah Sumbal

Summer SURE Project
2018-07-04

Table of Contents

Introduction:

SSD tracker is a program that can count and track detected objects using SSD algorithm in a video stream/file. This document explains the pipeline process of SSD Tracker program. The pipeline gives an overview of detection, tracking and counting along with other components used in this program. Input parameters and outputs are also listed with expected values for each component.

Abbreviations and Definitions:

SSD: Single Shot Multibox Detector. This is a detection method that uses deep neural networks.

Detection Classify an object.

Tracking Track the detected objects in consecutive video frames.

Counting Counting the number of tracked objects crossing the counting line.

Counting line Imaginary line drawn by the user on the frame to count objects.

Dataset Set of images used to train the model.

Model Complex mathematical functions for detecting objects

Weight Parameters and coefficient for variables used in the model.

Path Trajectory points of the detected object on the frame.

Caffe Deep learning framework which is used for running and training SSD.

OpenCV Computer vision library with deep neural network support.

Tensorflow Computer vision library with deep neural network support.

Objective and Background:

The main purpose of the SDD tracker is to count the number of people, dogs and bicycles passing in a video file. There are multiple ways to achieve this task. A lot of work has been done by using background and foreground subtraction. However, SSD Tracker program requirement stated the program must only count people, dog and bicycles. Therefore, It is necessary to use a detection algorithm. There are many detection algorithms. However, SSD Tracker program requirement stated that detection should be achieved by SSD[1]. SSD is a detection method that uses deep neural network for multiple categories. SSD implementation can run on different deep neural network including tensorflow, caffe and opencv. SSD Tracker program uses caffe to run SSD models which in our test proved to be the fastest.

SSD is run on every single video frame because the input to the SSD Tracker program is a video file. A tracking algorithms is used to track the detected objects in consecutive frames. SSD Tracker program uses a tracking implementation of Multitarget-tracker project[2]. The Multitarget-tracker project uses Hungarian algorithm and Kalman filter to track objects. More details about detection and tracking are discussed under the pipeline heading. The image below shows the detection, tracking and counting working together.

[1] "SSD: Single Shot MultiBox Detector." Accessed July 3, 2018. https://arxiv.org/abs/1512.02325.
[2] "Smorodov/Multitarget-tracker - GitHub." Accessed July 3, 2018. https://github.com/Smorodov/Multitarget-tracker.

Tracking is shown by trace line/path

Detection is shown by label and box

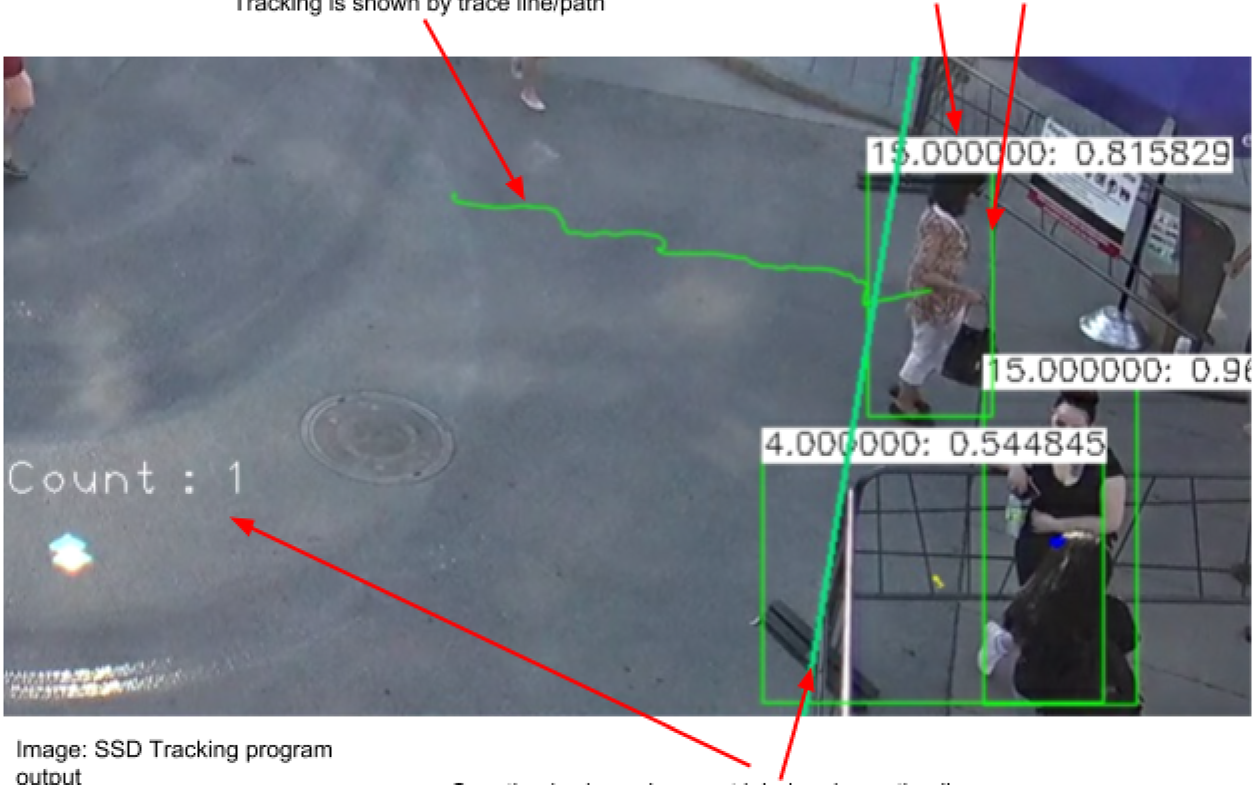


Image: SSD Tracking program output

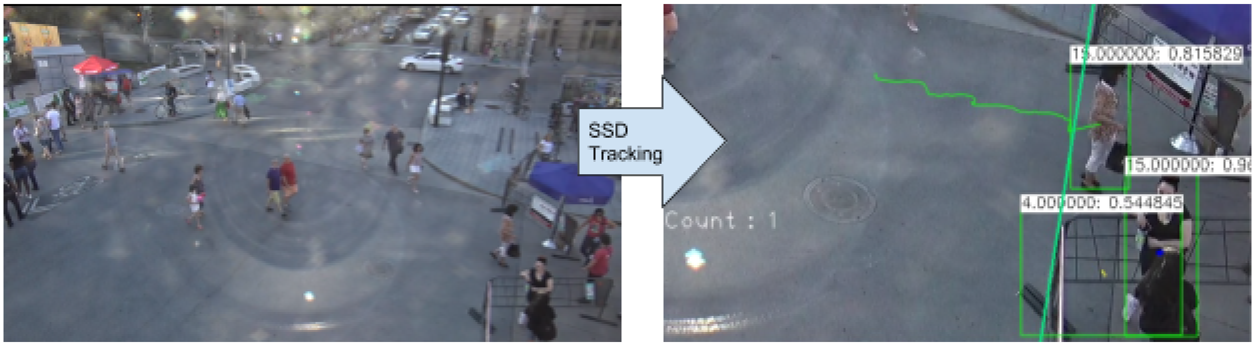
Counting is shown by count label and counting line

Pipeline Process:

This section explains functionality and interconnectivity of each component. It also list the input parameter and outputs with there expected values.

Overview:

The main input to the SSD Tracker program is a video file or stream and the output is the same video with modified frames. SSD Tracker draws boxes, lines and labels on the output video. The image below illustrate the modified frame.



Input Video

Output Video

The program reads one video frame at a time and that frame is processed and stitched together into a video output file. This cycle continues until all frames are read in the input video file. The program has three main components detection, tracking and counting. The components are executed in the following order

1. Detection
2. Tracking
3. Counting

The image below shows how three main components are connected along with other components.

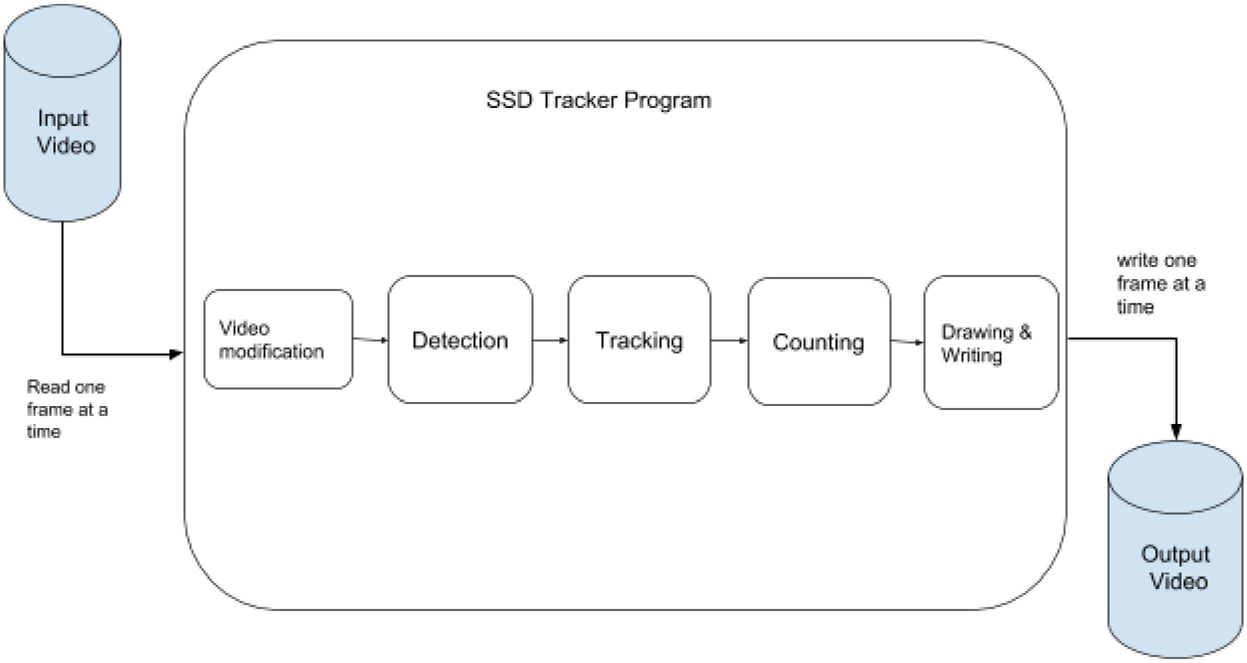


Figure: Block diagram for SSD Tracker program

All components are explained in the following headings. Each heading describes the function of the component along with there input parameters and outputs.

Fame modification:

This component is responsible for preparing the input video frame before applying detection. The frame is cropped into a desired area of interest. This step is necessary because it helps eliminate the extra objects in the frame. For example, the frame below is an unmodified frame of a video.

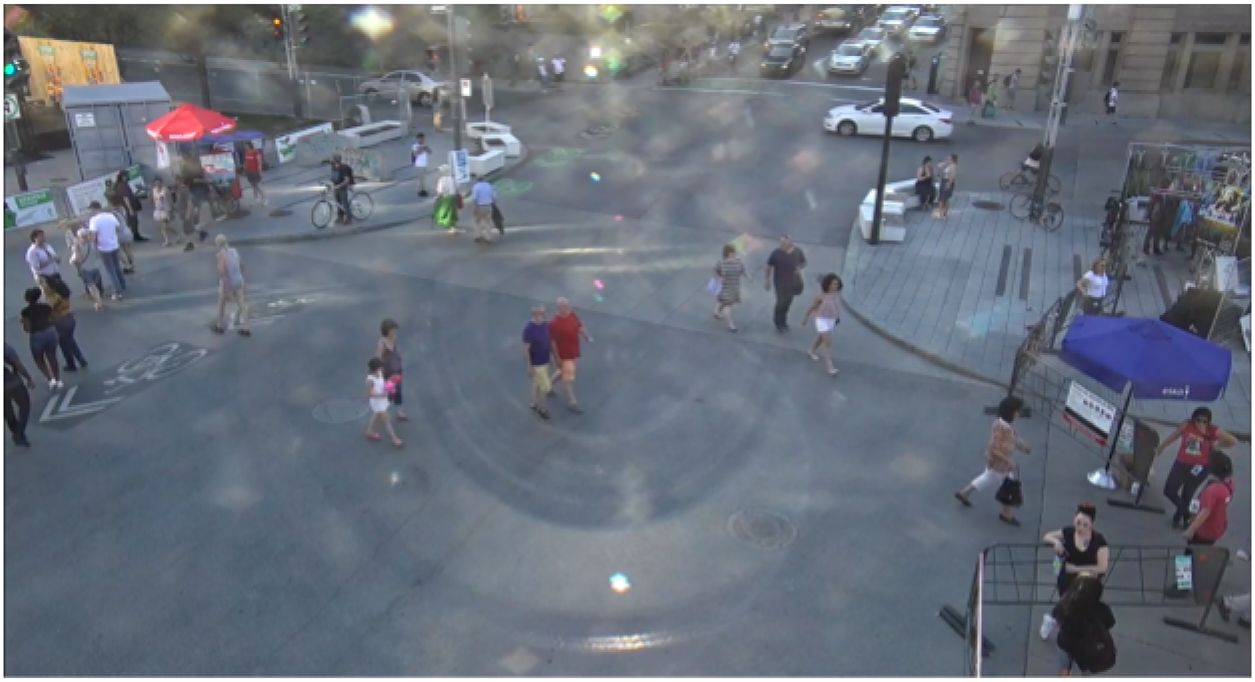


Image: input video frame (unmodified)

If we are only interested in the entrance area on the right side of the frame, we need to crop the following frame.

Image: input video frame after cropping

This cropped frame above is the output of Frame modification component. This way the program can ignore the rest of the frame. This also helps improve the processing time for the rest of the program because the program has to run other components on fewer pixels after cropping.

Input parameters:

1	Input frame from the video file:	Raw frame from the input video file.
2	Location of the area of interest:	This parameters needs three values. 1. Point coordinate 2. Width

3. Height

The point coordinate will be the top left corner of a rectangle. The rectangle shape of the rectangle is defined by the width and height. This rectangle is used to crop the frame.

Output:

1	Copped frame:	Interested area cropped into a new frame.

Expected values for input parameters and output:

1	Input frame from the video file:	Any video frame stored by Mat (an OpenCV object) which represents a n-dimensional dense numerical single-channel or multi-channel array.
2	Location of the area of interest:	1. Coordinate point: This point should be within the frame rows and columns. For example if the frame is 300 x 200, then the point coordinate x-axis value should range from 0 to 300 and y-axis value should range from 0 to 200. 2. Width: Once the coordinate point is set, width represent the wide of the rectangle on the x-axis 3. Height: height is the length of the rectangle on the y axis.
3	Cropped Frame	A frame stored by Mat (an OpenCV object) which represents a n-dimensional dense numerical single-channel or multi-channel array.

Detection:

SSD algorithm is pre-trained in advance on a dataset with different classes to detect objects. For example SSD can be trained on a large number of dog images. In this example large number of images is a dataset and dog is a class. The result of training a SSD algorithm produces model and weight files. SSD Tracker program uses model file and weight file to run the detection to identify object class name, object confidence, object location in a frame. There can be multiple objects in a frame so all detected objects are returned in an list. For example a person is detected in the image below.

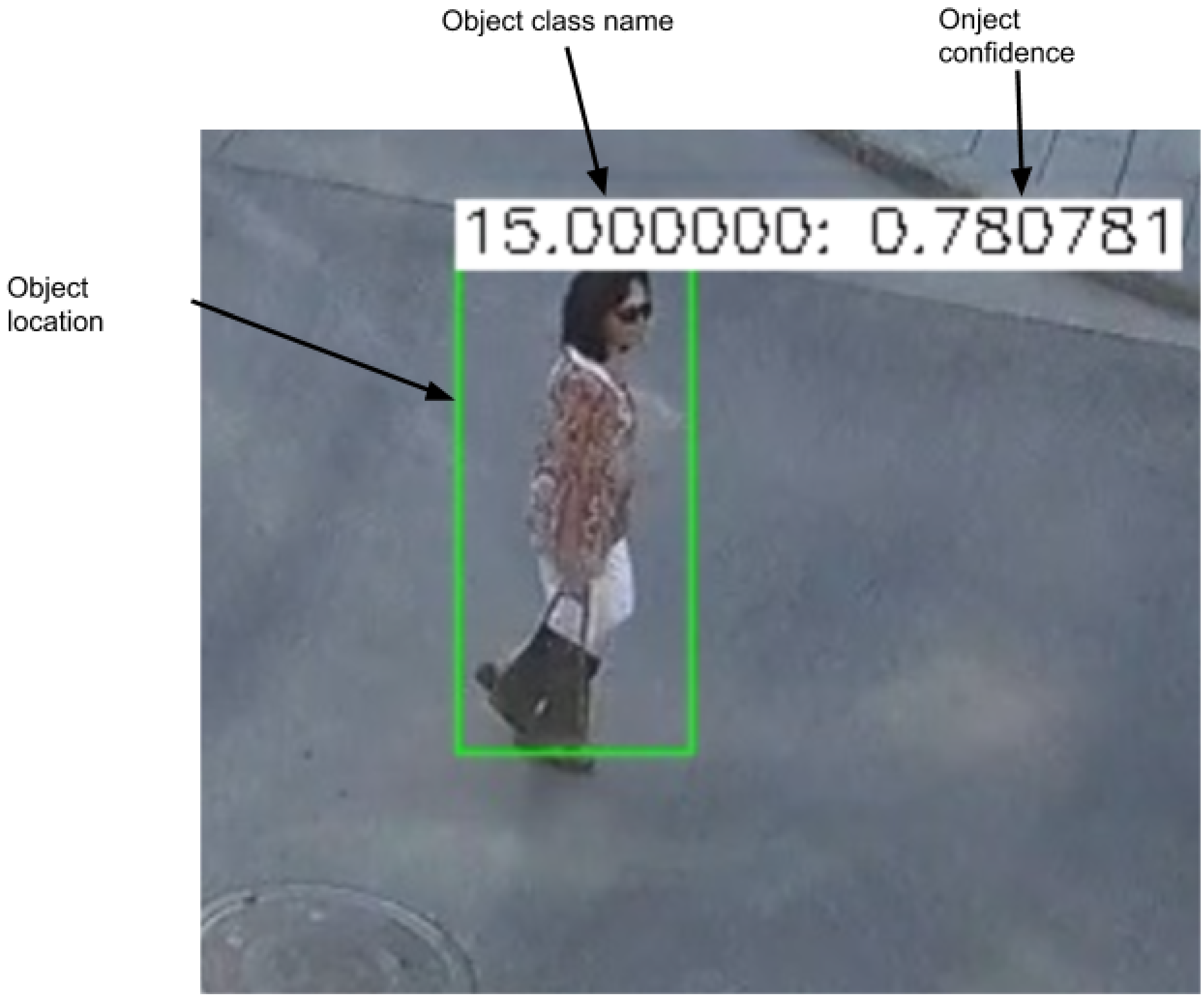


Image: Detected object in a frame

Object confidence is the probability of an object belonging to a certain class determined by SSD. In this example, SSD is 78.0781 % sure that the the detected object is from a class category of 15.0. 15.0 is the label name which corresponds to person in the label map file. Label map file contains label names for all the classes that the SSD is trained on. Object Location is the pixel position of the rectangle box (shown in a green box in the image above) around the detected object in a frame.

This component can limit the number of object classes detected. For example, the program can be configured to only detect dogs. This component can also ignore detected objects if the confidence percentages are below the threshold percentage input parameter. These two configurations might help reduce load on the program because fewer object are detected and less information is transferred to following components to be processed.

Input parameters:

1 Model file A trained deep neural network file for SSD.

2 Weight file A file containing variable coefficient and parameters for model file.

3 Label map file A file containing object class label names along with corresponding real names.

4 Threshold value: Confidence percentage of detected objects must exceed this value to be reported as a detected object.

5 Desired objects: List of object classes to be detected.

6 Cropped frame: Interested area cropped into a new frame (output of the Frame modification component).

Output:

1 Confidence: The probability of an object belonging to a certain class determined by SSD

2 Location: Rectangle boxes pixel position on the frame where the object is detected.

3 Label: Name of the class in which the object belongs.

Expected values for input parameters and output:

1 Model file A .prototxt file trained on caffe.

2 Weight file A .caffemodel file trained on caffe.

3 Label map file A file containing object class label names along with corresponding real names.

4 Threshold value: This is a percentage and it can range from 0 to 1. By default it is set to 0.5 (50 percent). At value close to 0 percent will detect most of the objects and may include many false negative. At values close to 1 (100 percent), only a few elements will be detected and may include a lot of false negatives.

5 Desired objects: An integer value, which has a range equivalent to the number of class the model can detect. For example if the model can detect 21 classes then range is from 0 to 20.

6 Cropped frame: A frame stored by Mat (an OpenCV object) which represents a n-dimensional dense numerical single-channel or multi-channel array.

7 Confidence: This is a percentage and it ranges from 0 to 1.

8 Location: Location is given by 4 values which corresponds to frame row and column numbers.

1. Left bottom column number (on x-axis)
2. Left bottom row number (on y-axis)
3. Right top column number (on x-axis)
4. Right top row number (on y-axis)

9 Label: Depending on the number of classes the model can detect. For example if the can detect 21 classes, then the label can range from 0 to 20.

Tracking:

SSD tracking uses a tracking implementation from Multitarget-Tracking[3] project. The main function of tracking is to figure out if detected objects in two consecutive frame are same or different objects. First the detected object path is predicted using Kalman filter[4]. Then predicted path is compared with the actual path (which is the location output from the detection component) in the next frame. If the distance between them is less than *max distance* input parameter than it is declared as the same object as in the previous frame otherwise it is declared as a new object. This object assignment is done by the Hungarian algorithm[5]. The tracking also has to resolve collisions problems. Collision happens when a previously detected object is not detection in the current frame and it is considered lost. Tracking component keeps predicting the path of the lost object until a certain number of frames defined by the *maximum allowed skipped frames* input parameter or the object is found again. The image below shows a path of two tracked object.

[3] "Smorodov/Multitarget-tracker - GitHub." https://github.com/Smorodov/Multitarget-tracker. Accessed 4 Jul. 2018.

[4] "Kalman filter - Wikipedia." https://en.wikipedia.org/wiki/Kalman_filter. Accessed 4 Jul. 2018.

[5] "Hungarian algorithm - Wikipedia." https://en.wikipedia.org/wiki/Hungarian_algorithm. Accessed 4 Jul. 2018.

Lost object path

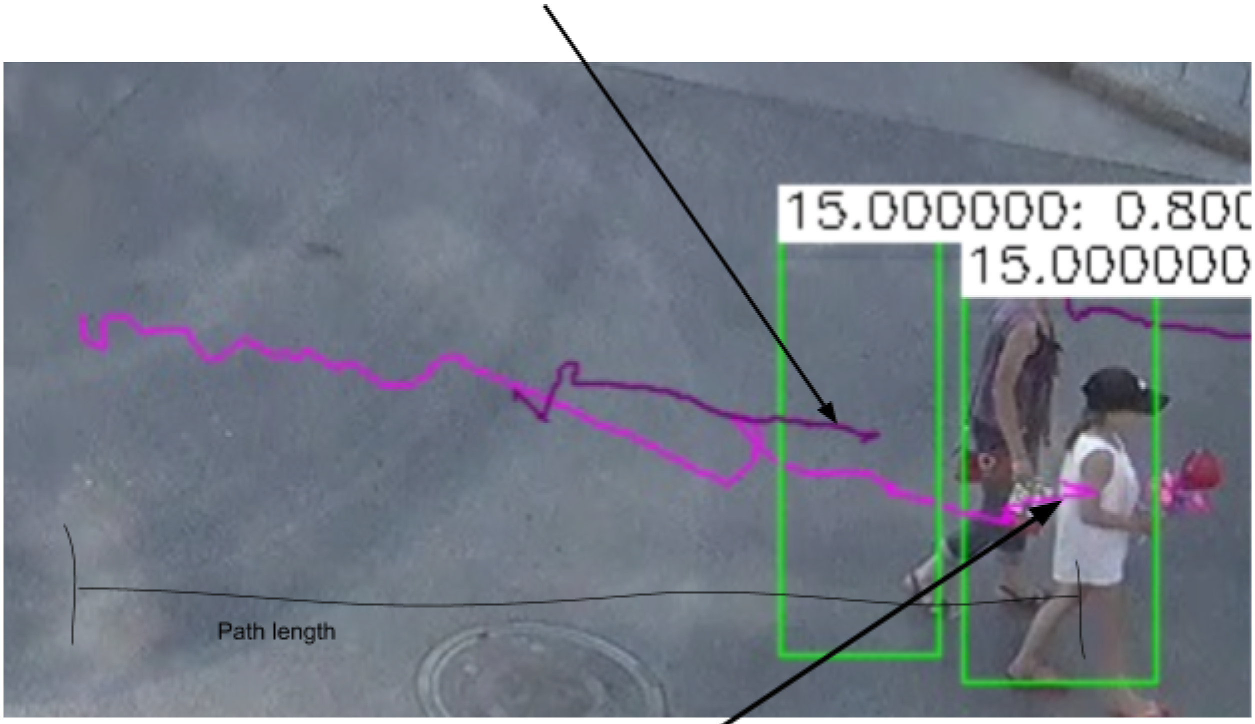


Image: After processed by the tracking component

Detected object path

In the example above, there is a occlusion and the person behind the little girl is not detected. The tracking algorithm is predicting the path for the lost person and It will keep predicting path for maximum allowed skipped frames input parameter before removing the path history. Tracking component makes sure that only the recent number of trajectory points in a path are saved which is defined by the maximum trace length input parameter. Path is shown is by the lines leading to the boxes centre in the example shown above.

Input parameters:

1 Location: Detected object rectangle boxes pixel position (output from detection component).

2 Delta time for Kalman filter:

3 Acceleration noise magnitude for Kalman filter:

4 Max distance: Distance threshold between predicted and actual path

5 Maximum allowed skipped frames: Number of frames to keep predicting path of the lost objects.

6 Maximum trace length: Number of frames to keep track of the previous trajectory points in a path.

Output:

1 Predicted path: Path predicated by the tracking algorithm based on the detected object’s location information.

Expected values for input parameters and output:

1 Location: Location is given by values which corresponds to frame row and column numbers.

1. Left bottom column number (on x-axis)
2. Left bottom row number (on y-axis)
3. Right top column number (on x-axis)
4. Right top row number (on y-axis)

2 Delta time for Kalman filter:

3 Acceleration noise magnitude for Kalman filter:

4 Max distance: This input depends on the frame size and frame rate of the video. The input also should not exceed the frame width or height. 0 value means that the predicted path and actual matches exactly on the dot for an object to be tracked. 100 value means that if the predicted trajectory point in a path is with in 100 pixels away from actual position, then it is the same object in the two consecutive frames.

5 Maximum allowed skipped frames: This variable depends on the how many collision happen in the video. This can range from 0 to total number of frame in video.

6 Maximum trace length: This can range from 0 to total number of frame in video.

7	Predicted path:	A point on a frame. For example (100, 200). This point is always in the frame window. It can not be more than the width or height of the frame.

Counting:

The main function of this component is to count the object passing a line defined by the user on the frame. There is a counter that increments every time an object crosses the counting line. Counting can be restricted to only one direction when objects cross the line (for example, only count the objects coming from from left to right). Counting can also be configured to count in both directions. Direction input parameter can be configured to set the counting direction in the program. Counting component looks at the predicted path from the tracking component output and checks if the object has crossed the line. The image shows that the person is about to cross the line.

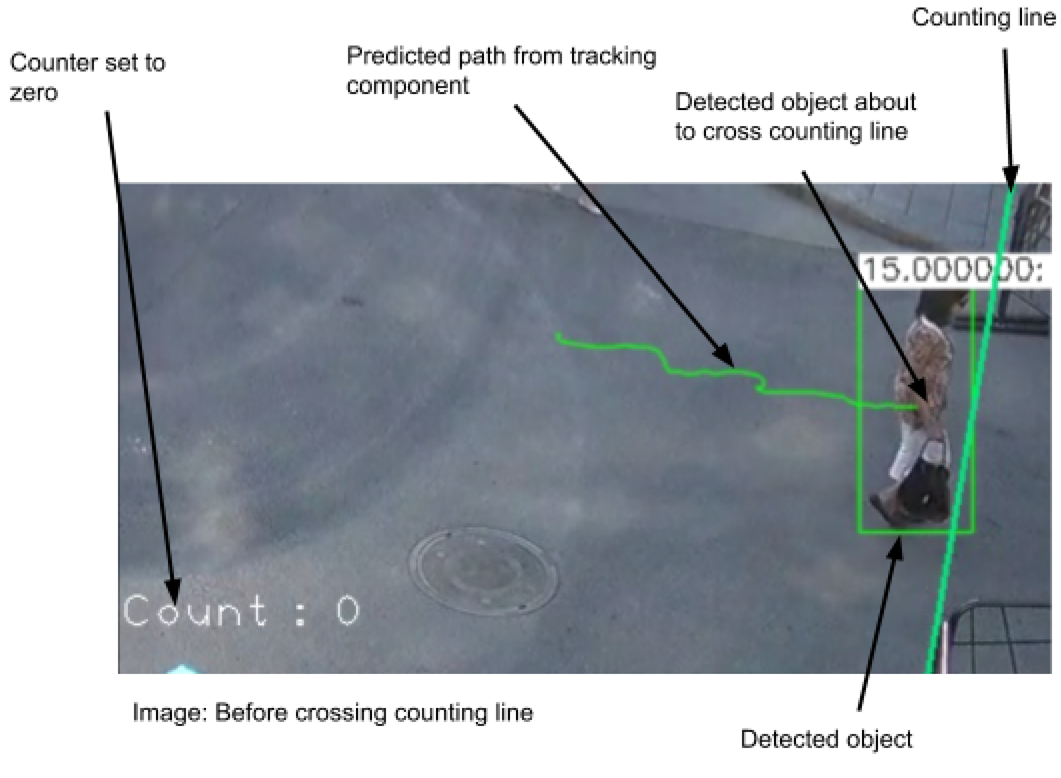

Counter set to zero
Predicted path from tracking component
Counting line
Detected object about to cross counting line
15.000000:
Count : 0
Detected object

Image: Before crossing counting line

The direction in this example is set from left to right. The counting component is processing every frame to check if the object’s path has crossed the counting line. The image below shows the counter increments when the object (person) crosses the counting line.

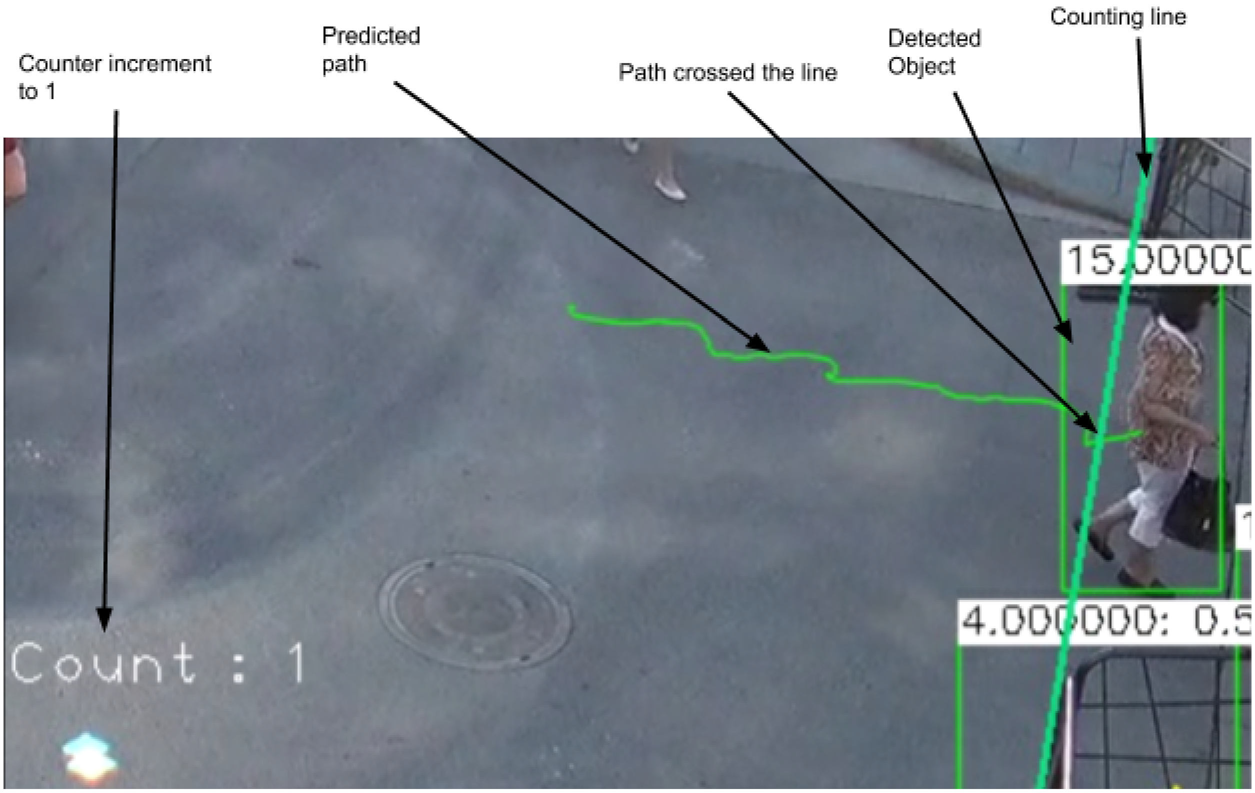


Image: After crossing counting line.

Input parameter:

1 Predicted path: Path predicated by the tracking algorithm based on the detected object location information (Output from the tracking component).

2 Direction: Variable to allow counting in a certain direction.

3 Line: Coordinates of the counting line on the frame.

Output:

1 Counter: A variable that increments every time an object passes the counting line.

Expected values for input parameters and output:

1 Predicted path: A point on a frame. For example (100, 200). This point is always in the frame window. It can not be more than the width or height of the frame.

2 Direction: Range from 0 to 2
0: count object moving from left to right (or top to down)
1: count object moving from right to left (or down to top)
2: count object moving from both direction.

3 Line: 4 points on the frame. The object has to cross two line to be sure it has passed. The first two points are the point for line 1 and the last two points are for line 2.

4 Counter: An integer number that is equivalent to the number of objects crossing the counting line. For example if the 4 detected and tracked objects are crosses the line, the counter value is 4.

Drawing & Writing:

This component is responsible for displaying information on the cropped frame outputted by the detection, tracking and counting components and it is also combines the new frames into a video file. It draws the detected objects' location rectangle boxes, detected objects' label names, detected objects' confidences, tracked objects' paths, counting line and counter incrementer on the frame of the video. The image below shows the output out the drawing component.

Detection is shown by label, box and confidence

Tracking is shown by trace line/path

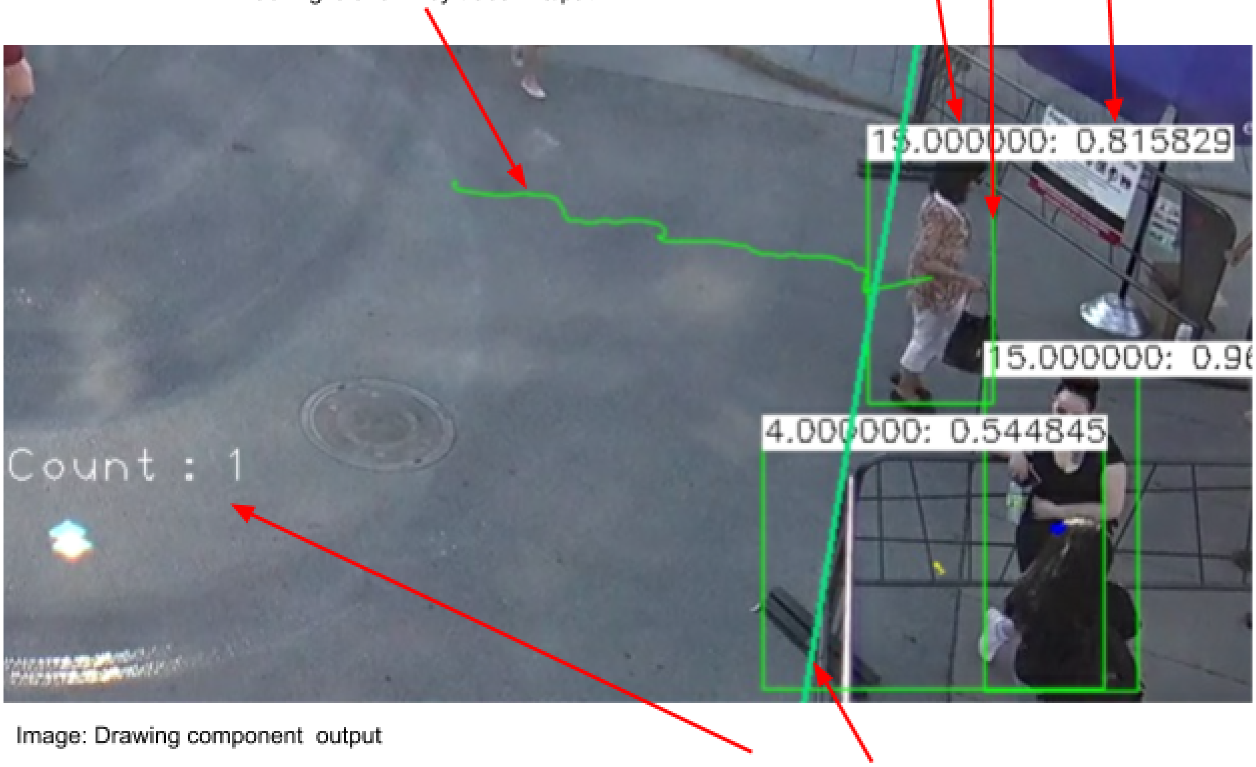


Image: Drawing component output

Counting is shown by count label and counting line

After the drawing step, the frame are combined together at frame rate defined by the FPS input parameter. The writing step also follows an encoding scheme provided by the fourcc parameter to write the combined frames to disk as a video file. This fourcc values affects the formatting of output file in terms of size and time it takes to write video to disk.

Input parameter:

1 Predicted path: Path predicated by the tracking algorithm based on the detected object location information (Output from the tracking component).

2 Counter: A variable that increments every line the object crosses the counting line (Output from the counting component).

3 Confidence: The probability that the object's class is identified correctly by SSD (Output from the detection component).

4 Location: Rectangle boxes pixel position on the frame where the object is detected (Output from detection component).

5 Label: Name of the class in which the object belongs (Output from the detection component).

6 Cropped Frame: Interested area cropped into a new frame (Output from the Frame modification component).

7 FPS Frame rate for output video

8 Fourcc Encoding scheme for output video.

Output:

1 Drawn Frame: Information displayed on the cropped frame outputted by the detection, tracking and counting component.

2 Output video file Output video files after merging the Drawn Frames.

Expected values for input parameters and output:

1 Predicted path: A point on a frame. For example (100, 200). This point is always in the frame window. It can not be more than width or height of the frame.

2	Counter:	An integer number that is equivalent to the number of objects
3	Confidence:	This is a percentage and it ranges from 0 to 1.
4	Location:	Location is given by values which corresponds to frame row and column numbers. 1. Left bottom column number (on x-axis) 2. Left bottom row number (on y-axis) 3. Right top column number (on x-axis) 4. Right top row number (on y-axis)
5	Label:	Depending on the number of classes the model can detect. For example if the can detect 21 classes, then the label can range from 0 to 20.
6	Cropped Frame:	A frame stored by cv::Mat which represents a n-dimensional dense numerical single-channel or multi-channel array.
7	FPS	An integer value and it can range from 0 to any number. For example, if the actual input video has 30 fps, then setting FPS to Less than 30: slower moving object compare to actual video More than 30: faster moving objects compare to actual video Equal to 30: same speed moving objects compare to actual video
8	Fourcc	A four character code which is an identifier for a video codec, compression format, color or pixel format used in media files For example fourcc('M', 'J', 'P', 'G'). For more examples check here.