People Counting System – User's Manual

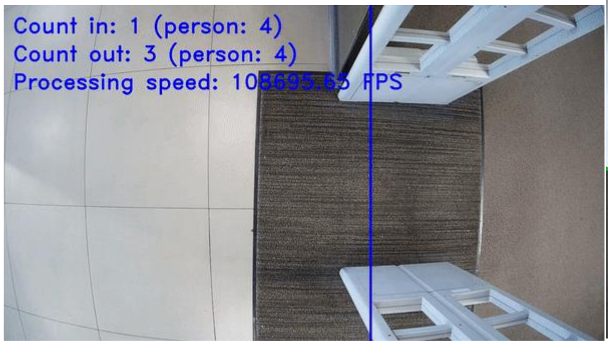


Figure 1 - Central library video (clickable)

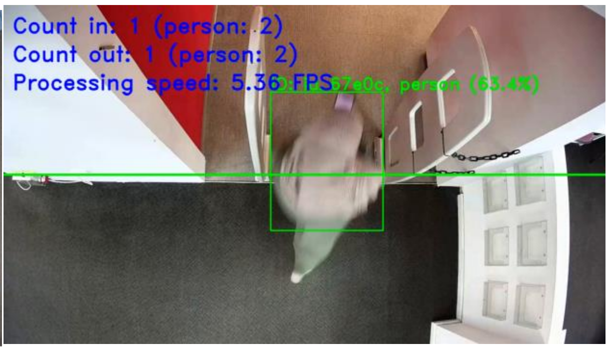


Figure 2 - Architecture library counting

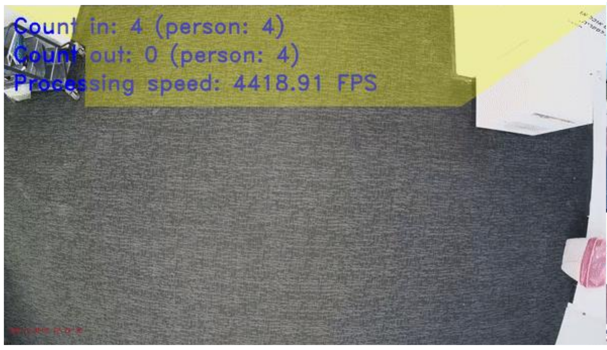


Figure 3 - Medical library video (clickable)

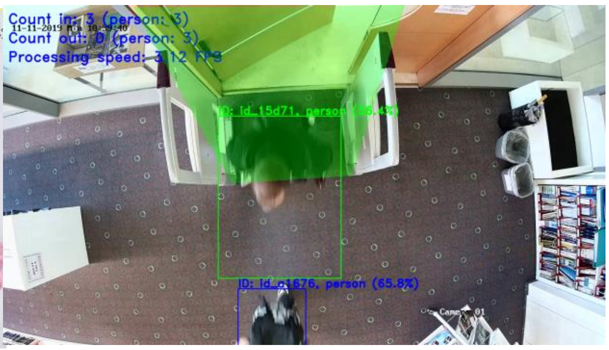


Figure 4 - Mechanical library counting with ROI

(The photos on the left side are clickable for a short video)

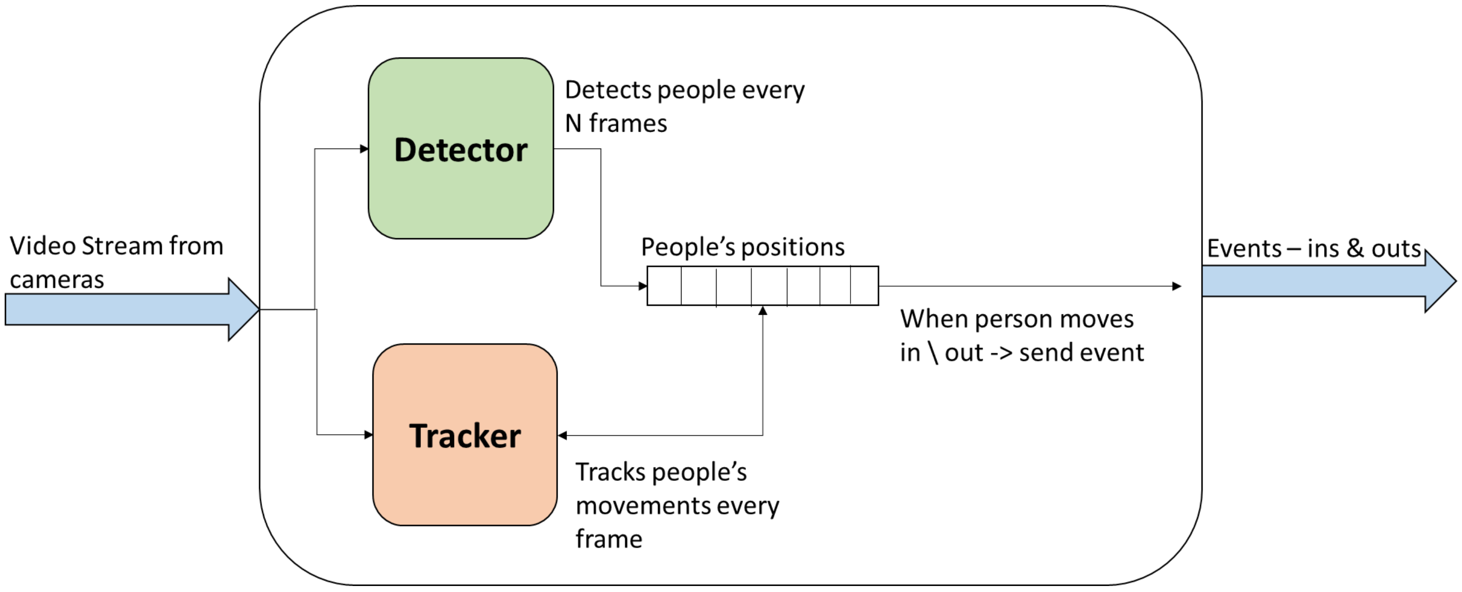
Authors: Ido Galil, Or Farfara

Table of Contents

System Implementation

System Design

In this chapter we will view in depth the code flow of the counting system.
First, a short overview of the system:


Video Stream from cameras
Detector
Detects people every N frames
People's positions
Tracker
Tracks people's movements every frame
When person moves in \ out -> send event
Events – ins & outs
Figure 5 - System's general scheme

Input & Output: The system receives a video stream from the cameras, and outputs events whenever a person was detected entering or exiting the entrance (with the corresponding "in" or "out" event).

Internally, the system has two main components:

1) Detector: a component detecting objects it classifies as human (implemented with YOLOv3), and either identifies them as objected already detected in the past (in which case their positions will be updated) or as new objects (and then adds new objects to the people's positions array).
   The detector attempts to make detections only every 'N' frames in order to lower the computational cost of the system.
2) Tracker: this component tracks the locations of already existing people (previously detected by the detector) and updates their positions in the array. Implemented by Open-CV's CSRT tracker algorithm.
   In our implementation, each person has his own tracker object.

Every time a person has changed his position from the entrance exterior region to the interior region (which is marked by a line or polygon), a counting event occurs and an "in" or "out" event is sent from the system.

We will now explain the system's code flows (mentioning its files and most important functions).

Full flow

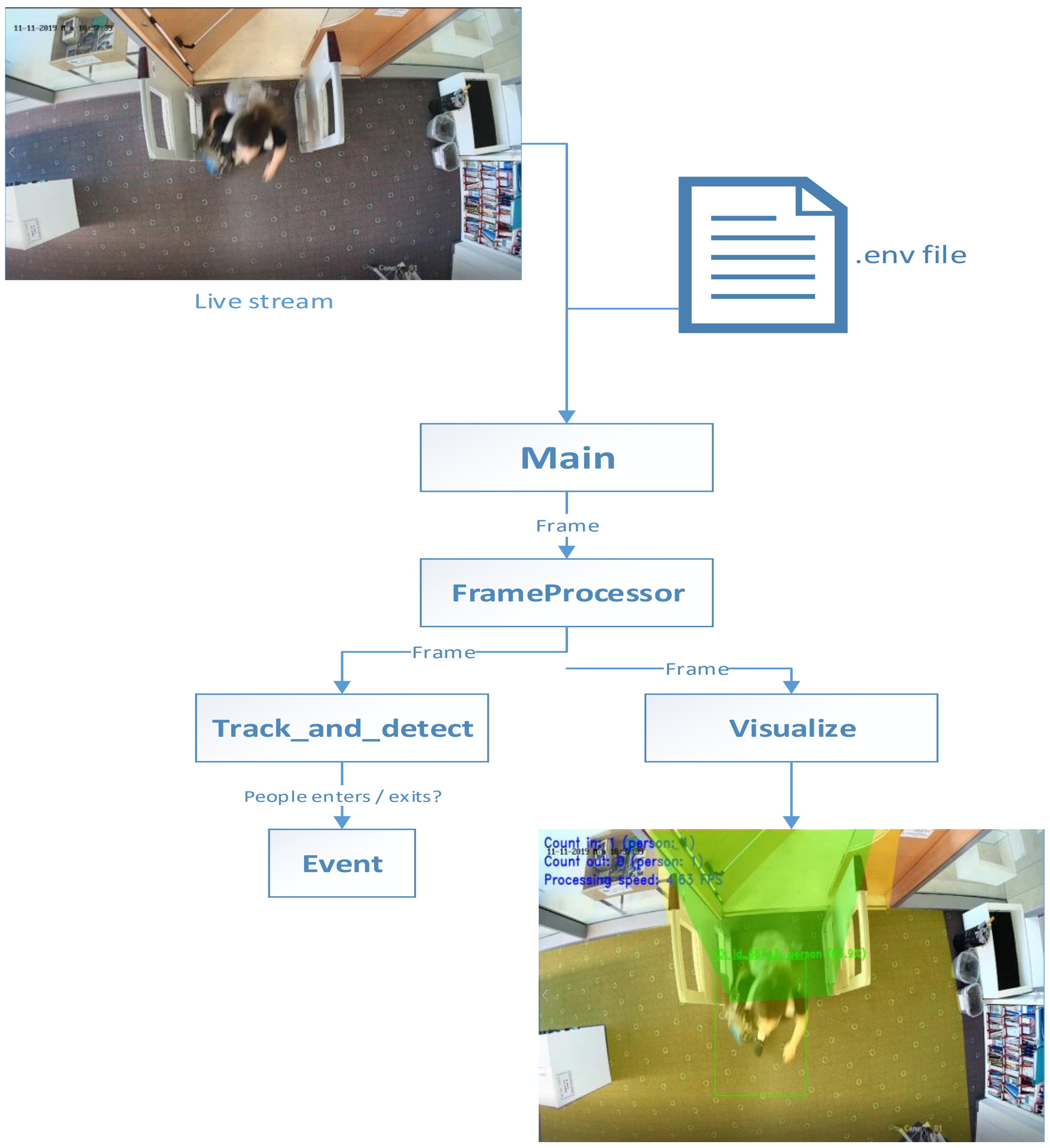


Figure 6 - Full flow of the counting system

The counting system starts in the “main” file.
It receives an “.env” file that contains all the configurations needed for the system (the configuration parameters will be explained in the “System parameters” section).
The main file is in charge of parsing all the parameters from the configuration file and validating them.
After parsing the parameters, “main” then starts a live connection to the required camera.
From this point, live stream enters the system and are sent by “main” to the **FrameProcessor** component, frame by frame.

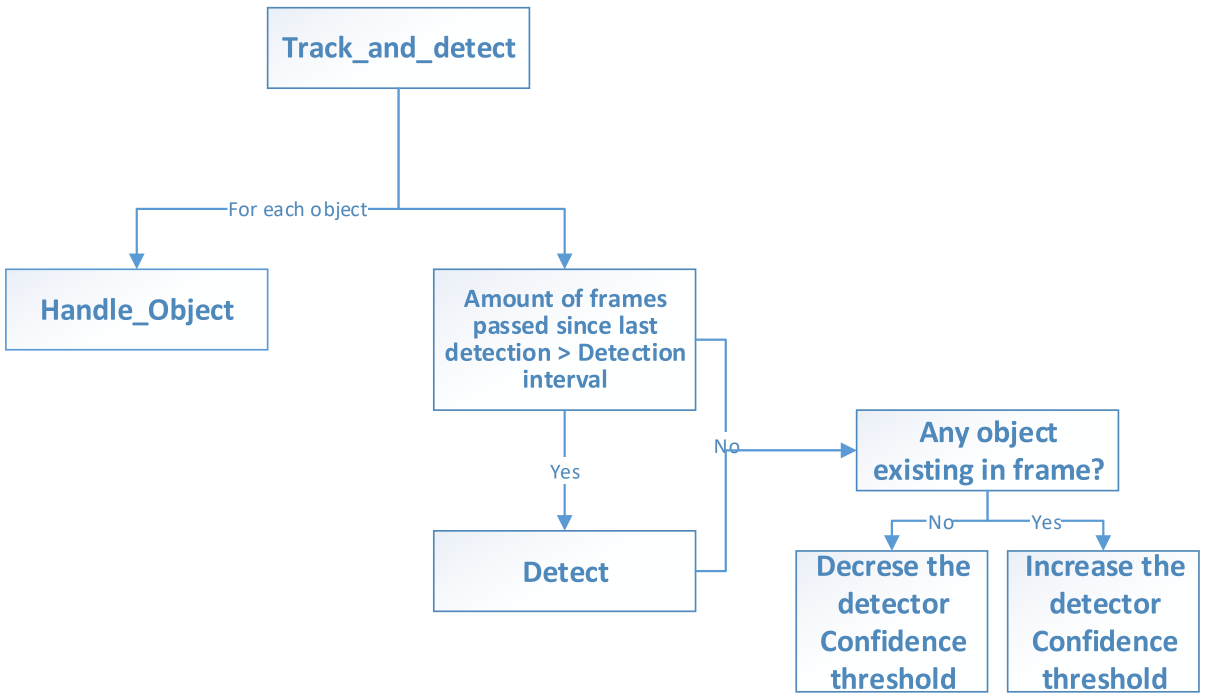
The FrameProcessor component will send each frame to 2 main functions:

1. **Track_and_detect** – will detect objects each X frames (configured parameter that called “DI” [Interval Detection]), track existing objects and finally will count the objects that entered and exited from the library.

2. **Visualize** – will create visualizations on the frame (in accordance with the configurations made):
    a. Object bounding boxes.
    b. Line/ROI that separates the “In” zone and the “Out” zone.
    c. Detection area (“DROI”).
    d. Objects “Liveness area”.
    e. Amount of people that enter and exit the library.
    f. Frame per seconds (FPS) rate.

In-depth Flow

Track_and_detect


Track_and_detect
For each object
Handle_Object
Amount of frames passed since last detection > Detection interval
Yes
Detect
No
Any object existing in frame?
No
Yes
Decrese the detector Confidence threshold
Increase the detector Confidence threshold

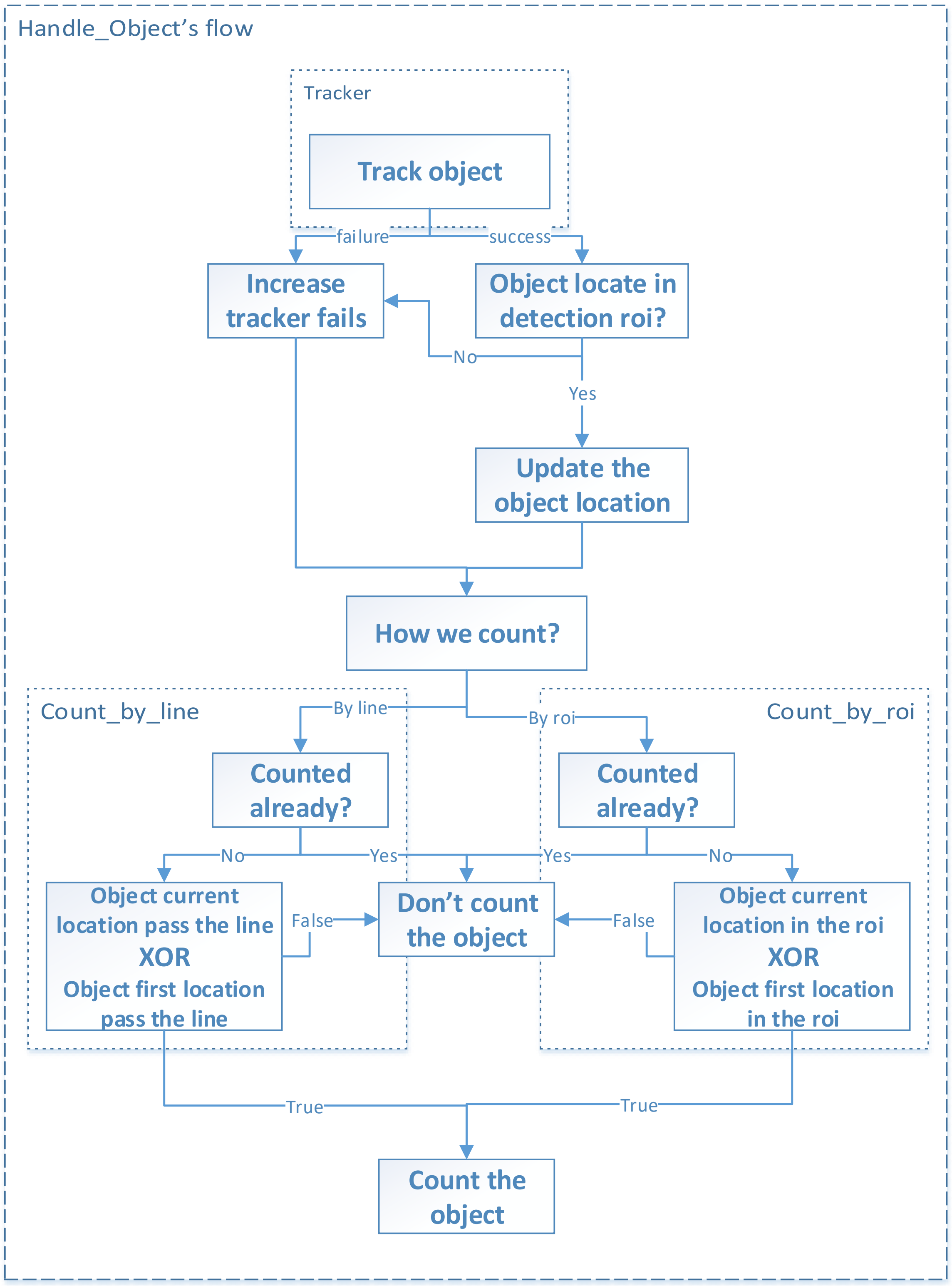
Track_and_detect function purposes are:
1. Track existing objects and update their positions.
2. Detect new objects.
3. Count object if it enters / exits the library.
4. Send IN / OUT event.

The FrameProcessor component holds a list of all the objects that exists.
First, Track_and_detect will perform the Handle_Object function for each existing object.
Handle_Object will track the object, update the object location and count it if needed.
Second, Track_and_detect will check if the amount of frames since the last detection has passed the Detection Interval (DI) limit, and if so will perform the Detect function.

The **Detect** function will run the detector model and detect the objects on the frame, will add new objects \ update existing ones.

Finally, if no object exists in the frame, the detector's confidence threshold will be decrease in order to be more sensitive to new objects emerging in the next frames (as part of the "Detection Slowdown" mode explained later).

Handle_Object



Handle_Object does the following:

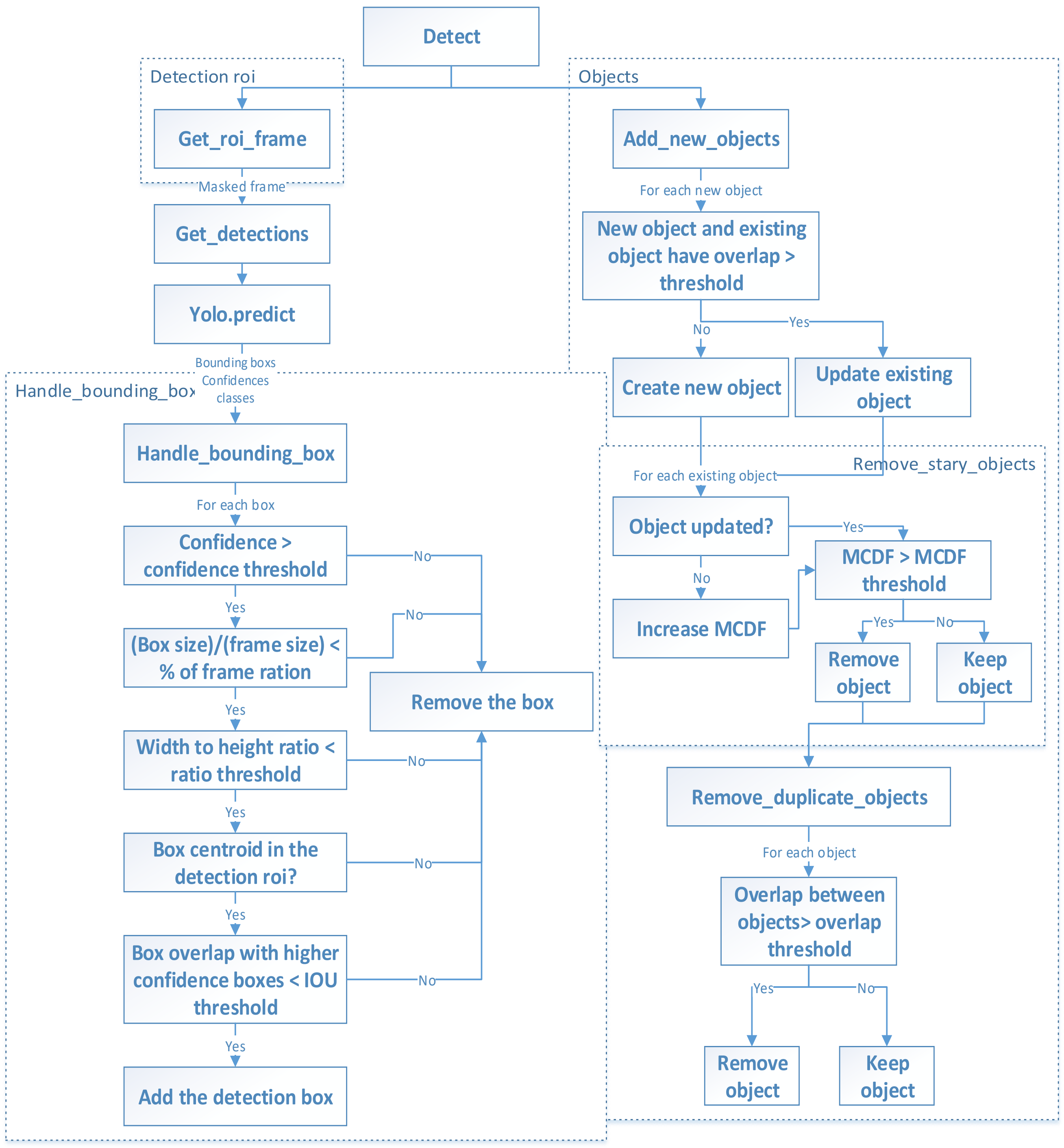
1. Runs the object's tracker.

2. If the tracker succeeded, it will check if the object is located in the detection ROI (DROI).
   if so, the location of the object will be updated. otherwise, it will increase the tracker failure (points that the object might have left the frame).
3. Will check if the object has transitioned from the area it was first detected to the other one (e.g. In -> Out), and wasn't counted yet. If so, it will count the object and send an event.
   The separation between the areas can be by a line or ROI (area of the inside / outside of the entrance).

Detect

The **Detect** function has two main goals:

1. Detect objects on the frame and validate them.
2. Add the detected objects to the objects list (as new objects or update existing ones).



<u>The detection procedure</u>:

The detection itself is performed by a YOLO model's predict function that was implemented by Keras (with a Tensorflow backend).

The predict function gets as input an image and an area on it that it needs to detect from (DROI) and outputs the detected objects.

It will choose the objects that:

- Have higher confidence than the confidence threshold.
  Purpose: choose only the object that we are sure about.
- Smaller than the maximum object size threshold (measured as a percentage by Box size / frame size)
  Purpose: Avoid objects too big (unlikely to be human)
- Ratio between the height and the width smaller than the maximum object ratio.
  Purpose: Avoid objects with abnormal shapes (unlikely to be human)
- The center of the object is in the detection ROI (DROI)
  Purpose: Detect only objects where they should be
- Smaller overlap with objects with higher confidence than the IOU threshold
  Purpose: Avoid object duplication.

The inserting of the detected object procedure:
In this stage we want to decide for each detected object if it’s a new object in the image or an old object that moved and needs to be updated.
In addition, we want to remove the objects that have left the frame.
How do we do that?
In order to decide if this object already exists we will check if the object has high overlap with one of the existing objects. If so, we will update the existing object.
In order to remove objects that left the frame, we will hold a counter of the detection failures for each object. We update the counter every time the object wasn’t updated by the detection procedure. When the amount of detection failures become higher than the threshold (MCDF), the object will be removed because it has probably left the frame.
Lastly, existing objects having high overlap can be deduced as the same object, so we will remove them (leaving only one) to avoid duplicates.

System's parameters

The system's parameters purpose is to configure the system to the relevant camera, and the way the counting system makes its detections and tracking of the objects.
The system's parameters are divided into 5 categories:

Input

Configures the input of the system.
We have 2 options for video input:

A. Local video file – file that is located in the computer.
B. Live stream – Internet connection to the camera that gets live frames.

<u>Parameters:</u>

- **Is_cam** – Flag that specifies if we want to set as input live stream or a video file.
- **Video** – the path of the video file.
- **Connection** – live connection to the camera.
  Example:
  "rtsp://<username_of_camera>:<password_of_camera>@<ip>:<port>/h264_stream"

Counting

Configures the counting method, the counting area / line and the liveness area of objects.
Counting methods: we have 2 counting methods:

A. **Count by ROI** - (Region of Interest). Count all the objects that passed from / to the ROI.
   Example - How is it done? (Assuming that the ROI is the area outside the library)
   <u>The object enters the library</u>: if the object's first detection was in the ROI and the current location is not in the ROI.
   <u>The object exits from the library</u>: if the object's first detection wasn't in the ROI and the current location is in the ROI.

<u>Count by ROI Parameters:</u>

- **Counting_roi**- Specifies a counting region of interest. (A set of vertices that represent the area where you want counting to be made).

[1]

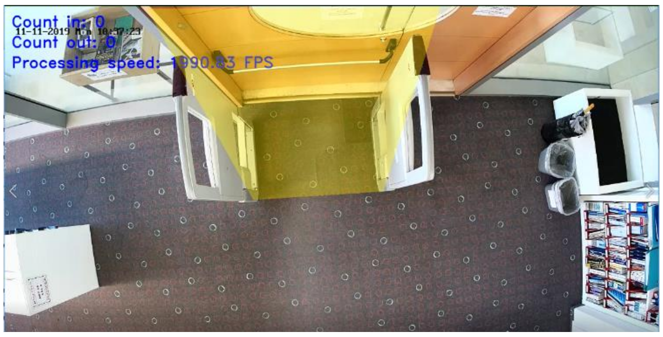


Figure 7 - Example of using counting ROI – use it when the entrance of the library is not straight.

- **Use_count_roi** –flag for enable counting by roi and not by line. If False- will count by line.
- **Show_count_roi** – Disable/overlay the counting roi on the video.
- **Counting_roi_outside** – orientation of counting region (True – the region is the outside area, False – the region is the inside area).

B. **Count by line** – count all the objects when they pass a line.

Note that the line is only for display, and marks a position on the X \ Y axis. The object doesn't have to be tracked or detected directly on the line, just on the two sides of it.

Example - How is it done? (Assuming that the right side of the line is the outside of the library)

<u>The object enters the library</u>: if the object's first detection's x coordinate was right of the line and the current location's x coordinate is left to the line.

<u>The object exits the library</u>: if the object first detection's x coordinate was left to the line and the current location's x coordinate is right of the line.

<u>Count by line Parameters:</u>

- **Counting_line_orientation** – orientation of the counting line (top/bottom/left/right). If right – all object that are right of the line are considered outside the area.
- **Counting_line_position** – position of counting line, needs to be a float number between 0 to 1. The line will be in the position*width of the frame.

[1] Recommendations for using counting ROI.

Liveness area of objects:

An object “liveness” ROI.

This simple ROI should cover most of the frame, except for a narrow strip next each of the exits from the frame. Whenever an object that was already counted leaves the liveness zone, it is discarded.

You should configure this area as far as you can from the counting line in order to avoid counting object that was discarded but by a small chance detected again in the area outside the liveness area.

Parameters:

- **Object_liveness_roi** – set of vertices that represent the area which within counted objects will not be discarded.
- **Enable_object_liveness** – enable the liveness feature.
- **Show_object_liveness** – draw the liveness ROI on the video.

Detection

DROI:

DROI (Detection Region of Interest) – a polygon that includes the only areas detections are plausible from.

For each frame extracted from the video, before being sent to the detector, the area outside the DROI polygon get masked to ensure detections will not be made there. Additionally, if the tracker returns that an object has entered a zone outside of the DROI, it will be considered as a failure.

The DROI is intended for zones that are impossible for human movement (like book-shelves, walls or fixed inanimate objects).

Parameters:

- **Use_roi** – Enable / Disable detection by region.
- **DROI** – set of vertices that represent the area (Polygon) where you want the detection to be made.
- **Show_droi** – Display / overlay the detection ROI on the video.

DI

DI (Detection Interval) – This is a number that dictates the amount of frames between every 2 detections to be made.

The lowest possible DI is 1. Generally, lower DI will result in more accurate detections but will slow down the system more.

The DI has a correlation with FPS. If the FPS is lower, then the DI should be lower.

(Our tests and parameters were adjusted for 12 FPS)

Parameter: **DI**

MCDF

MCDF (Maximum Continuous Detector Failure) - the amount of detection failures until the object will be removed.

In order to remove objects that have left the frame, the system holds a counter of the detection failures for each object. We update the counter every time the object wasn’t updated by the detection (the detection procedure as explained earlier didn’t detect this object). When the amount

of detection failures becomes higher than MCDF, the system will remove the object because it has probably left the frame.
Higher MCDF values allow the tracker to significantly support the detector.

The smaller the camera zoom is, the higher MCDF should be – because it will be longer until the object will leave the frame (increasing the probability for higher streaks of detection failures).

If the line is positioned far from the center of the frame, MCDF should be higher – because it will take longer for the object to reach the line (and if the object was only detected once when it entered the far side of the frame it will need MCDF high enough for the tracker to support it all the way to the line).

Note that high MCDF values may result in a phenomenon we call a "bouncing box". It happens when an object leaves the frame, and the tracker makes bad attempts to locate where the object has gone to next. It will continue to bounce the box locations around until the MCDF count is depleted.
Parameter: **MCDF**

<u>IOU threshould:</u>
The Threshold above which 2 objects detected will be considered as the same object and the one with the lower confidence will be removed.
Parameter: **IOU_threshould**

<u>Percentage of frame threshold:</u>
The threshold of bounding box size (measured as a percentage of the frame's size) above which a detected object is removed.
Recommended values depend on the camera's zoom. The larger the zoom is, the higher the threshold should be.
Parameter: **Percentage_of_frame_threshold**

<u>Maximum width to high ratio allowed:</u>
The maximum ratio allowed between width and height of a box, ensuring only proportionate bounding boxes. Value mast be larger than 1.
Parameter: **Maximum_width_to_high_ratio_allowed**

<u>Detection slowdown:</u>
This mode, when turned on, has two phases:

1) <u>Phase 1</u>: When no people were detected on screen, detections would behave as usual in terms of timings (detect every DI frames), but detection confidence is doubly more sensitive than usual (so that the detector would more easily detect objects).
2) <u>Phase 2</u>: While at least one person is detected on the frame, the DI is decreased to "1", and the confidence threshold is returned to normal.

Parameters:

- **Detection_slowdown** – enable/disable the feature.
- **Sensitive_confidence_threshold** – A sensitive confidence threshold meant to be lower until the first detection occurs. It relevant only when the Detection_slowdown is enable.

YOLO_model_path:
The path of the yolo h5 model file. This path can be replaced by tiny-yolo / other yolo versions easily (or other detectors inputting and outputting the same format). It has to be an h5 file.
Parameter: **YOLO_model_path.**

Confidence_threshold:
All the detected object with lower confidence than this threshold will be removed.
Parameter: **Confidence_threshold.**

Tracker

Tracker:
Configure the algorithm to use for people tracking.
There are a lot of trackers implemented in Open-CV. We have used only CSRT.
The supported trackers (**not fully tested**) – kcf, csrt, medianflow, mosse, tld, goturn.
Parameter: **Tracker.**

MCTF:
"Maximum Consecutive Tracking Failures", the number of tracking failures before the tracker concludes the tracked object has left the frame.
The tracker can fail after the track function (if the tracker itself failed) or if the tracker returns an object not in the detection ROI.
Parameter: **MCTF.**

Overlap threshold:
When a new detection is made overlapping with an existing person, the minimum overlap threshold to correlate the existing tracked person to the newly detected one (deducing they're the same person).
Parameter: **Overlap threshold.**

Output

Record:
Enable / disable the recording of the video with the people counter output.
Parameter: **Record.**

Output_video_path:
The path where the video record will be stored.
Parameter: **Output_video_path.**

UI:
Run the system with UI display.
Parameter: **UI.**

Logging:

The logger of the system creates messages that contain the events occurring during the running of the system. i.e.: object creation / deletion.

Parameters:

- **Enable_console_logger –** Enable / disable the logger messages into the console.
- **Enable_file_logger –** Enable / disable the logger messages into file.
- **Log_file_directory -** The path where the log file will be stored.

Debug_window_size:

Size of window UI.

Parameter: **Debug_window_size.**

Color_change_interval_for_counting_line:

The amount of frame that the color of the object and the counting roi/line will be different from the moment the object was counter.

Parameter: **Color_change_interval_for_counting_line.**

DEBUG-

Enable/disable debug mode. Debug mode has the ability to control the video procession (pause/stop), get pixel positions and display the UI.

Parameter: **DEBUG.**

Manual: How to?

How to set a new "ROI" polygon?

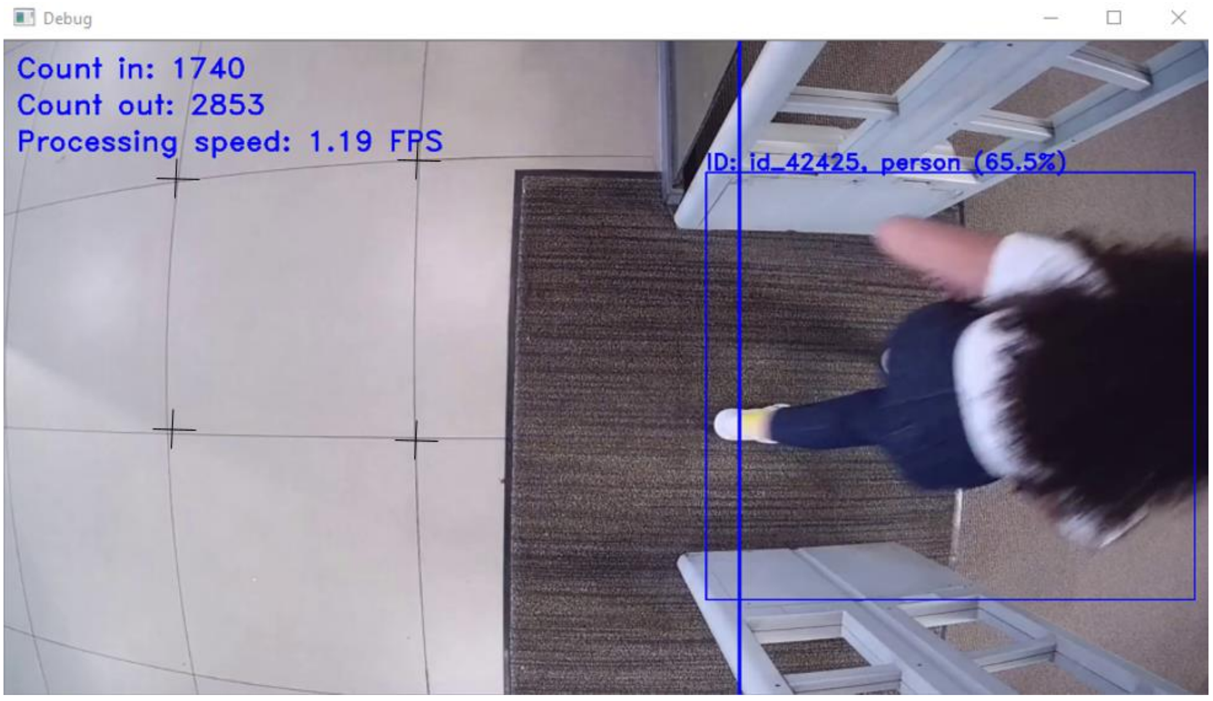
Do the following steps:

1. Run the system on "debug" (Enable the DEBUG parameter). It is recommended to pause the video.
2. Mouse click on the polygon's vertices you want to set.
3. The pixel positions for those vertices will be shown in the console log.
4. Order the pixel position from left to right into a list.
   For example: [(the top left point), (the bottom left point, (the bottom right point), (the top right point)].
5. Add the pixel list to the env file.
6. Enable the show flag for that specific ROI in the env file in order to verify the polygon.

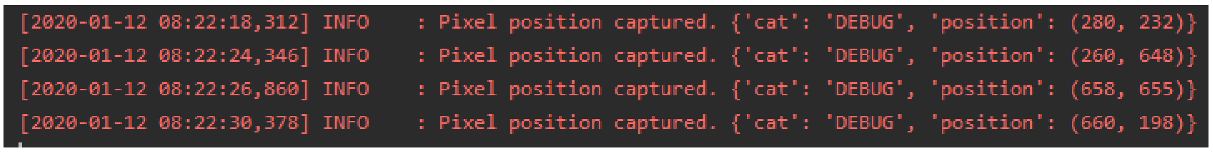
Example

We want to capture the tile from the floor as a polygon.

We first click on the mouse click 4 times (where the cursors are marked in the photo):



We got the following positions:



```
[2020-01-12 08:22:18,312] INFO    : Pixel position captured. {'cat': 'DEBUG', 'position': (280, 232)}
[2020-01-12 08:22:24,346] INFO    : Pixel position captured. {'cat': 'DEBUG', 'position': (260, 648)}
[2020-01-12 08:22:26,860] INFO    : Pixel position captured. {'cat': 'DEBUG', 'position': (658, 655)}
[2020-01-12 08:22:30,378] INFO    : Pixel position captured. {'cat': 'DEBUG', 'position': (660, 198)}
```

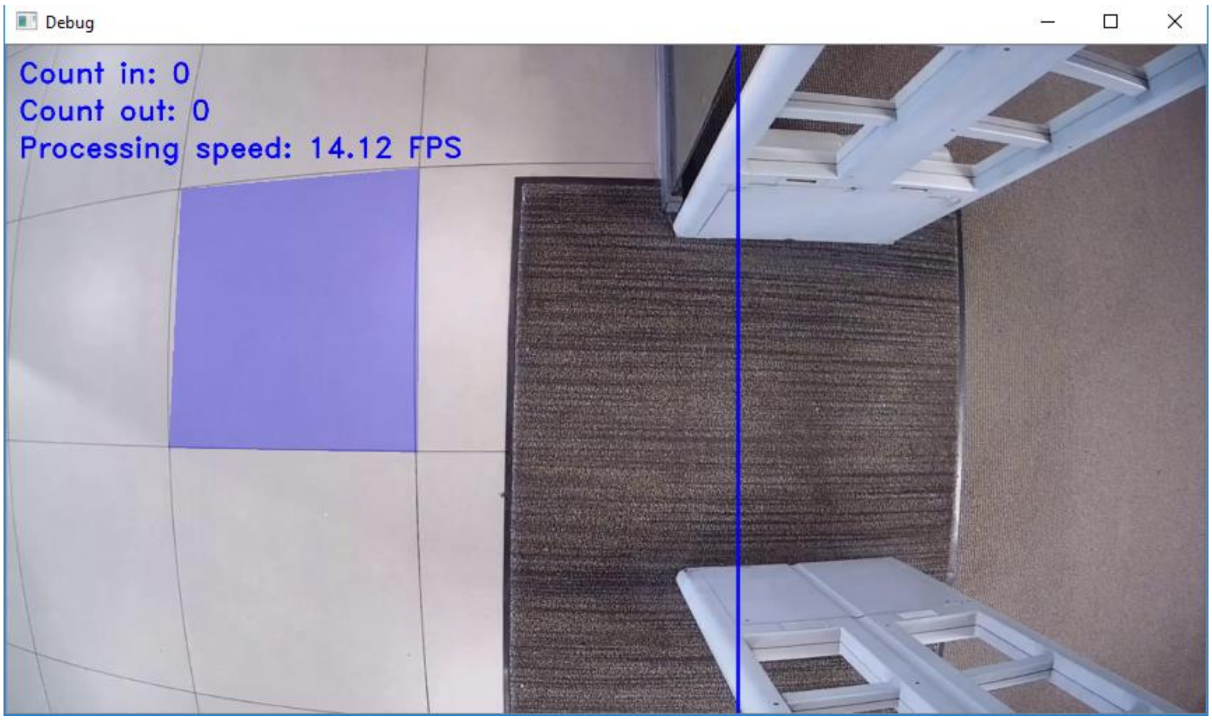
We order the pixels:

[(280,232), (260,648), (658,655), (660,198)]

We added the pixel list to the env file and enable the show polygon flag:

```
OBJECT_LIVENESS_ROI = [(280,232), (260,648), (658,655), (660,198)]
ENABLE_OBJECT_LIVENESS = True
SHOW_OBJECT_LIVENESS = True
```

The polygon now looks like this:



```
Count in: 0
Count out: 0
Processing speed: 14.12 FPS
```

How to set up a new camera?

1. Run the system on “debug” (Enable the DEBUG parameter).
2. Configure the camera connection in the env file. (change the IP / port / protocol)

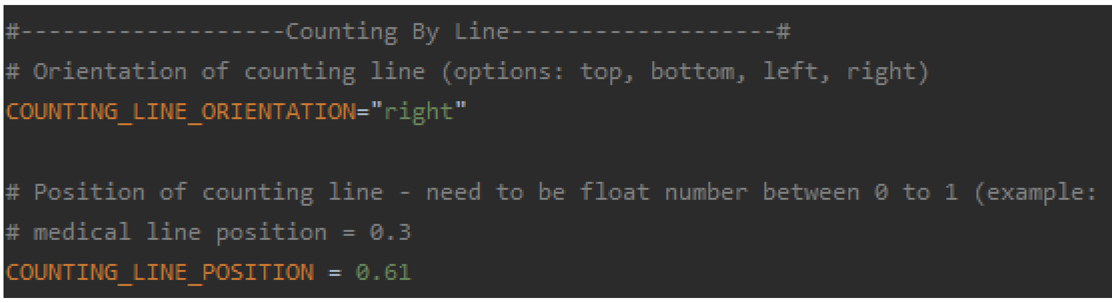
```
#--------------------Live Camera----------------------#
# Specify if video capture is from a live camera
IS_CAM=True
```

```
CONNECTION= "rtsp://[redacted]"
```

3. Decide how to count
    a. Count by Line –
        i. Disable the counting by ROI flag in the env file –

```
# flag for enable counting by roi and not line, if it False ,we will count by l
USE_COUNT_ROI = False
```

        ii. Configure the counting position-

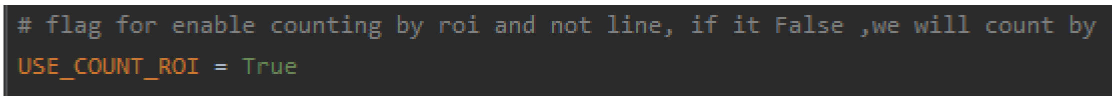


```
#-------------------Counting By Line-------------------#
# Orientation of counting line (options: top, bottom, left, right)
COUNTING_LINE_ORIENTATION="right"

# Position of counting line - need to be float number between 0 to 1 (example:
# medical line position = 0.3
COUNTING_LINE_POSITION = 0.61
```

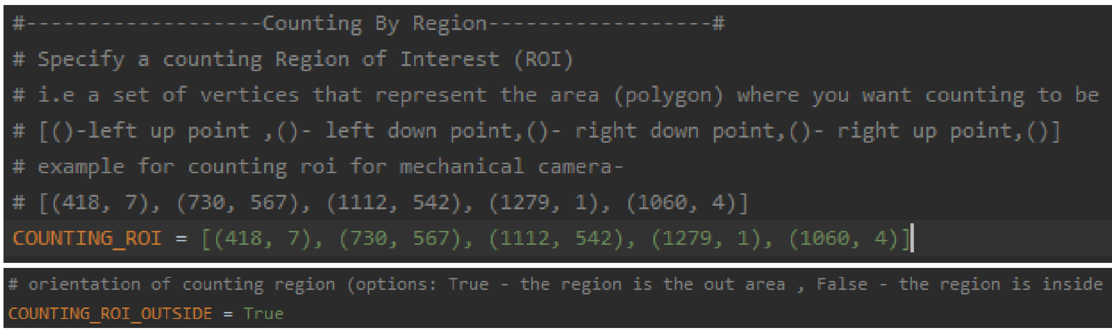
Try to locate the line in the center of the frame (closer to 0.5), and in the middle of a physical bottleneck if one is possible.
The line orientation should be the “out” area by the line.

b. Count by ROI-
   i. Enable the counting ROI flag:



```
# flag for enable counting by roi and not line, if it False ,we will count by
USE_COUNT_ROI = True
```

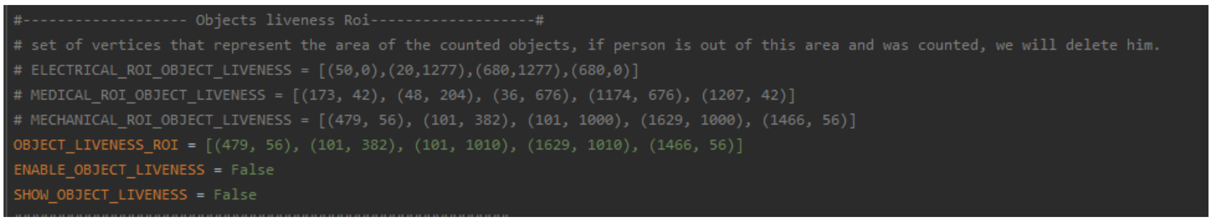
   ii. Configure the counting ROI polygon



```
#------------------Counting By Region-----------------#
# Specify a counting Region of Interest (ROI)
# i.e a set of vertices that represent the area (polygon) where you want counting to be
# [()-left up point ,()- left down point,()- right down point,()- right up point,()]
# example for counting roi for mechanical camera-
# [(418, 7), (730, 567), (1112, 542), (1279, 1), (1060, 4)]
COUNTING_ROI = [(418, 7), (730, 567), (1112, 542), (1279, 1), (1060, 4)]
```

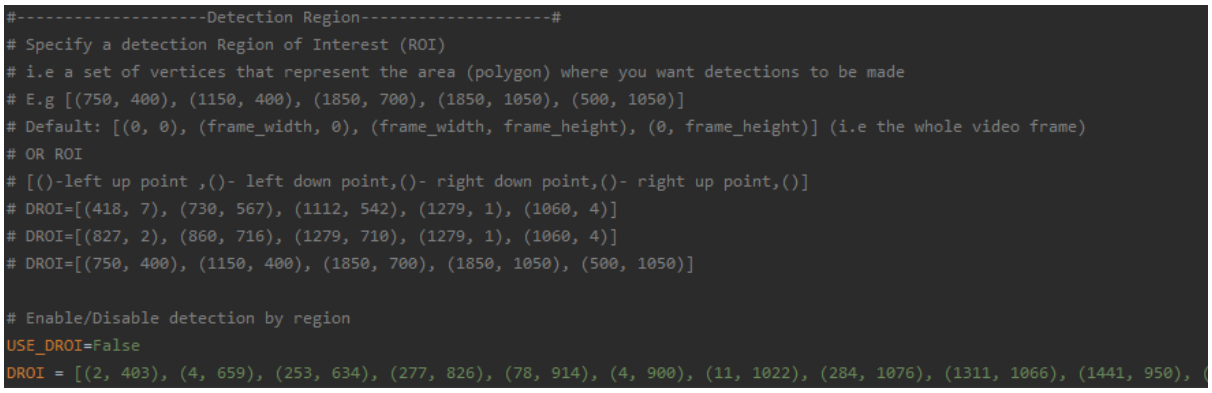
```
# orientation of counting region (options: True - the region is the out area , False - the region is inside
COUNTING_ROI_OUTSIDE = True
```

3. Object liveness:
   Configure the object liveness polygon and enable \ disable the flags



```
#------------------ Objects liveness Roi------------------#
# set of vertices that represent the area of the counted objects, if person is out of this area and was counted, we will delete him.
# ELECTRICAL_ROI_OBJECT_LIVENESS = [(50,0),(20,1277),(680,1277),(680,0)]
# MEDICAL_ROI_OBJECT_LIVENESS = [(173, 42), (48, 204), (36, 676), (1174, 676), (1207, 42)]
# MECHANICAL_ROI_OBJECT_LIVENESS = [(479, 56), (101, 382), (101, 1000), (1629, 1000), (1466, 56)]
OBJECT_LIVENESS_ROI = [(479, 56), (101, 382), (101, 1010), (1629, 1010), (1466, 56)]
ENABLE_OBJECT_LIVENESS = False
SHOW_OBJECT_LIVENESS = False
```

4. DROI:
   If you want to use detection ROI, configure the polygon and enable \ disable the flags.



```
#-------------------Detection Region-------------------#
# Specify a detection Region of Interest (ROI)
# i.e a set of vertices that represent the area (polygon) where you want detections to be made
# E.g [(750, 400), (1150, 400), (1850, 700), (1850, 1050), (500, 1050)]
# Default: [(0, 0), (frame_width, 0), (frame_width, frame_height), (0, frame_height)] (i.e the whole video frame)
# OR ROI
# [()-left up point ,()- left down point,()- right down point,()- right up point,()]
# DROI=[(418, 7), (730, 567), (1112, 542), (1279, 1), (1060, 4)]
# DROI=[(827, 2), (860, 716), (1279, 710), (1279, 1), (1060, 4)]
# DROI=[(750, 400), (1150, 400), (1850, 700), (1850, 1050), (500, 1050)]

# Enable/Disable detection by region
USE_DROI=False
DROI = [(2, 403), (4, 659), (253, 634), (277, 826), (78, 914), (4, 900), (11, 1022), (284, 1076), (1311, 1066), (1441, 950), (
```

5. Find the best MCDF for the camera by testing and configure the MCDF:

```
#-------------------Detection Failures--------------------#
# Maximum consecutive detection failures i.e number of detection failures
# before it's concluded that an object is no longer in the frame
MCDF=9
```

6. DI:
   configure the DI to the required interval (for 12 FPS, DI of 3 is recommended).

```
#-------------------Detection Interval--------------------#
# Detection interval i.e number of frames before detection is carried out again
# (in order to find new people and update the trackers of old ones)
DI=3
```

How to debug?

In order to debug the system, you need to run the program in the debug mode (Enable the DEBUG parameter).

- You can pause / stop the video stream by clicking on “P” or “Q”.
- You can use the logs and debug the events that occur during the system running.

The logs of the system include 2 types of messages:

1. Error messages – when the validation of the system parameters failed.
2. Information messages – inform about creation of a new object, updates to the position of an object, count event of an object, deletion of an object, pause / stop the video stream and the captures of the position pixel.