EvenVizion - video based camera localization component

Oxagile Research and Development department
Oxagile
rd@oxagile.com

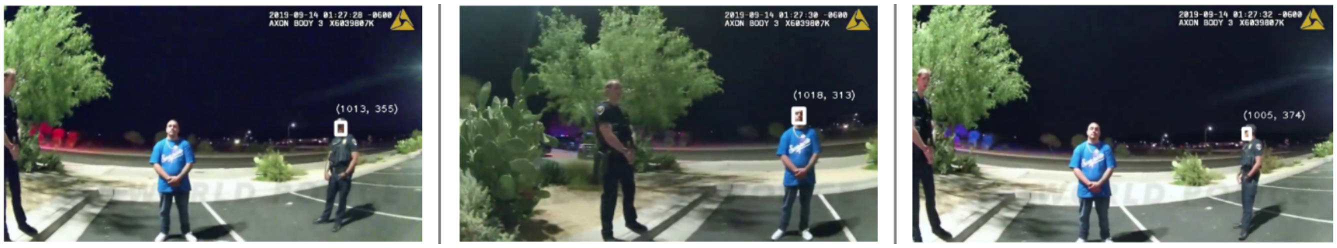

Figure 1: Same objects have different coordinates while camera moves

ABSTRACT

The power of the Computer Vision (CV) technology is definitely redefining the whole landscape of the Public Safety solutions market nowadays. While the traditional video monitoring systems are a common thing bordering with a necessity, the usage of artificial intelligence in this particular area is still experiencing an upsurge.

The whole area of CV in Public Safety is developing vibrantly, yet there's still much more to be said in terms of technological development.

There's a major task in Public Surveillance systems – to track an object. The task neither uncommon nor unresolved in 2020. But it's getting more complicated when some of the filming conditions change. It is a relatively common scenario in Public Safety when the video that needs to be analyzed afterwards is shot on a moving camera. For instance, body-worn cameras used by the police are constantly moving while an officer performs his/her duties. Naturally, it causes several problems with determining the relative position of objects.

Firstly, it's not exactly clear what made an object shift – the object movement itself of just the camera movement. Secondly, when a scene changes due to camera rotation, different objects can obtain same coordinates even if there were static (Figure 1).

Here we're not talking about the usual case where an object is constantly in the frame and changing its position gradually (because such cases are covered by classic trackers). We're also not considering the case where the filming conditions are normal or where a camera is stationary. If the filming conditions are bad or/and the movements are sharp, even the best trackers can fail the task. We detect an object and then lose it (e.g. people turn backs and face couldn't be detected) – which is the exact moment when the camera shifts. It leads to assuming the person we detected before as a new person. Tracker won't be of use here because it has lost its track while the person wasn't in the frame.Identification also doesn't work as the filming conditions are bad. We resolve these extreme cases with the EvenVizion component while tracking in usual lighting conditions can be achieved by easier means.

To determine the position of the object, the main task was set – the creation of a fixed coordinate system. To solve this, we created the EvenVizion component.

1 BACKGROUND

1.1 Homography

Assuming that (x, y) – are the object coordinates in plane α and (x', y') – are the object coordinates in plane β.

To solve the main problem, it is necessary to find a transformation that will translate all coordinates of the α plane into coordinates relative to the β plane. That is, if the stationary object had coordinates $(x, y) = (x_1, y_1)$ in α, then in β they should remain the same: $(x', y') = (x_1, y_1)$.

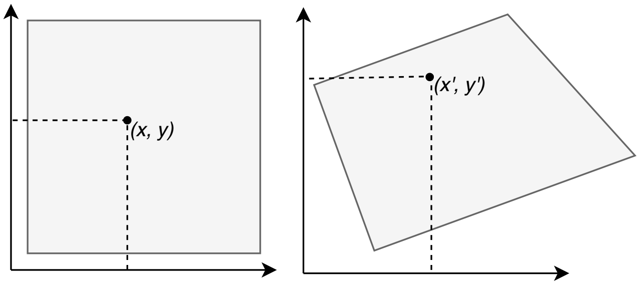

Figure 2: Comparison of two planes

To describe the change in the position of an object in the α plane as compared to the β plane it is possible to use a projective transformation (homography). You can read more about homography and matrix search in [1]. Let's consider by what parameters this transformation is characterized and how to obtain new coordinates with its help. The projective transformation is characterized by matrix H:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \tag{1}$$

Where:

- a, e – scaling in x and y respectively
- b, d – shift along the axes (affect the rotation together with a and e)
- c, f – offset along the axes
- g, h – change of the perspective

Using this matrix, you can describe the change in the coordinates of one object between two frames, if this change is caused by camera movement. You can get new coordinates (x', y') from (x, y) as follows:

$$x' = \frac{ax + by + c}{gx + hy + 1}, \qquad y' = \frac{dx + ey + f}{gx + hy + 1}, \tag{2}$$

or in matrix form:

$$t \cdot (x', y', 1)^T = H \cdot (x, y, 1)^T, \tag{3}$$

where $t = gx + hy + 1$

2 PIPELINE

Let's consider there is N consecutive video frames – $(f_1, \ldots, f_N)$ and fix the upper left corner of the first frame as the origin. We introduce the following notation:

O – origin;

$(X_k, Y_k) = \left((x_k^1, y_k^1), \ldots, (x_k^n, y_k^n)\right)$ – coordinates of n matching points in the plane of the k-th frame;

$(X_k', Y_k') = \left((x_k^{1\prime}, y_k^{1\prime}), \ldots, (x_k^{n\prime}, y_k^{n\prime})\right)$ – coordinates of n matching points on the k-th frame in a fixed coordinate system;

$(X_k'', Y_k'') = \left((x_k^{1\prime\prime}, y_k^{1\prime\prime}), \ldots, (x_k^{n\prime\prime}, y_k^{n\prime\prime})\right)$ – coordinates of n matching points on the k-th frame f_k in a fixed coordinate system in the plane of the $k - 1$st frame f_{k-1};

H_k – homography matrix between f_{k-1} and f_k.

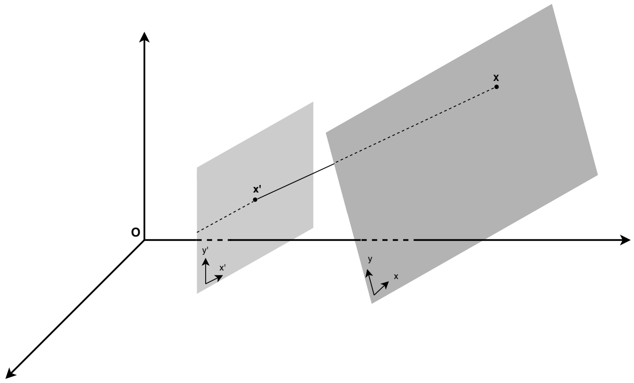

Figure 3: Projective transformation

To solve the basic problem, the following steps were identified:

1. comparison of two consecutive frames;
2. finding a transformation that will translate the coordinates of the object on the current frame into the coordinates relative to the point $(0, 0)$, while being projective (Figure 3).

The first step will be discussed in paragraph 3 in more detail. Now we consider that there is a set of common points for two frames and their original coordinates, which are relative to the frame.

Let's analyze the operation of the algorithm using the example of processing of the f_k.

It is necessary to find a transformation by which from (X_k, Y_k) we obtain (X_k', Y_k'). It is not always possible to find a homography matrix between f_1 and f_k, because during the movement of the camera, the scene could radically change and the frames may not contain common objects.

Find the transition matrix H_k between f_{k-1} and f_k frames. We assume that a sequence of such transition matrices has already been found for the previous frames $(H_1, \ldots, H_{k-1})$. Note that if the H_k matrix describes changes between the coordinates (X_{k-1}, Y_{k-1}) and (X_k, Y_k), then we will not take into account that the relative position of objects could have changed dramatically from the beginning of the video.

Consider Figure 4:

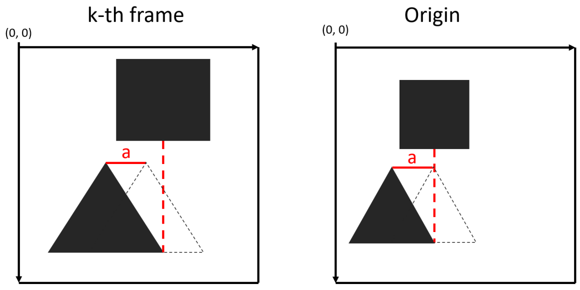

Figure 4: An example of camera movement that caused objects scaling

Here is an example of a camera movement in which objects are scaled. If the coordinate of the object has changed by a pixels: $x_k^1 = x_{k-1}^1 - a$ on the current frame, it doesn't mean that it moved up by a coordinates in the fixed system $x_k^{1\prime} \neq x_k^1 - a$. This is very clearly seen in Figure 4. Therefore, when searching for a transformation between two frames, you need to consider all the transformations that have occurred since the first frame. How to do so?

It is possible to describe the movement of the camera between all the previous pairs of consecutive frames using the well-known homography matrices $(H_1, \ldots, H_{k-1})$. But it is necessary to find such a superposition of these matrices, which will describe the complete movement from the f_1 to f_{k-1} frames and will allow finding the coordinates of matching points on the f_{k-1} in a fixed system. If you look at formula (3) in paragraph 1.1, it becomes clear that such a superposition is the multiplication of the matrices of all the previous transitions.

$$H_{sup} = H_1 \cdot (H_2 \cdot (\ldots \cdot H_{k-1})\ldots) \tag{4}$$

Thus, in order to find the homography matrix describing the movement of the camera from the f_{k-1} to f_k, it is necessary to: multiply the original coordinates (X_{k-1}, Y_{k-1}) and (X_k, Y_k) by the matrices' superposition – formula (4), and find the matrix H_k already for the obtained coordinates (X'_{k-1}, Y'_{k-1}) and (X''_k, Y''_k). As a result, we will receive a transformation that will translate the coordinates of the object on this frame (x_k^1, y_k^1) into the coordinates of this object relative to the origin $(x_k^{1'}, y_k^{1'})$.

Or in matrix form:

$$t \cdot (x', y', 1)^T = H_{sup} \cdot (x, y, 1)^T, \tag{5}$$

where $t \cdot (x', y', 1)^T = H_{sup} \cdot (x, y, 1)^T$

3 ADDITIONALLY

As the main idea of the whole pipeline is to calculate the homography matrix using some matching points, then the approaches to finding such points can be different. Let's introduce a definition: a special point will be called a point of the depicted object, which, with a high degree of probability, will be found on another image of the same object. The correspondence between singular points is established using their descriptors. In this version, we used SIFT (Scale-invariant feature transform), SURF (Speeded Up Robust Features) and ORB (Oriented FAST and Rotated BRIEF) to find special points and their descriptors. Three types of points were used to ensure that the points cover the maximum possible area of the frame.

The result is the following matching points highlighting scheme:

1. Special points and their descriptors are highlighted on the images.
2. By the coincidence of descriptors, the corresponding to each other special points (mentioned above matching points) are allocated.

More to the second point. For each descriptor we look for the two best matches using the k-nearest neighbor algorithm. Then we use Lowe's ratio test to remove the false positives, the constant `LOWES_RATIO` is responsible for the threshold for this test. Thus, for each point of the f_{k-1} frame, we obtain the corresponding to it on the f_k. Using the obtained matching points, we connect two frames, the following match is obtained, Figure 5.

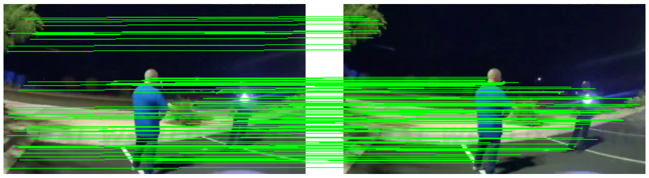

Figure 5: Matching visualization

When using these algorithms to find matching points, the problem arises: some points can cling to an object, the movement of which depends not only on the movement of the camera (static object), but also on the movement of the object itself – a person, a car etc. (moving object). Therefore, it is necessary to filter these points out. The following filtering option has been developed:

- We get "dirty" matching points with coordinates (X_k, Y_k), which include points on moving objects
- Find the homography matrix H which will translate the coordinates of the f_k into the plane of the f_{k-1}.
- We apply it to the found points and obtain their new coordinates (X'_k, Y'_k),
- Filter points as follows:
  - We are looking for the movement of each point $r = \sqrt{(x_k^{1''} - x_{k-1}^{1'})^2 + (y_k^{1''} - y_{k-1}^{1'})^2}$
  - We group and then round the displacement to integers. Points with the same displacement fall into one group.
  - If the number of points in the group is less than the threshold, we consider that these are the points that are caught on a moving object and remove them from consideration. In the script:

$$threshold = LENGTH\_ACCOUNTED\_POINTS * l, \tag{6}$$

where $l = len(matchingpoints)$.

The only points left are those on stationary objects. We use them particularly when calculating the homography matrix H.

But the matrix found on the dirty points will describe not only the movement of the camera, but also partially the movement of motion objects. And therefore, the coordinates of objects, whose movement will be described accurately enough, will not change and will fall into a group with static objects. Thus, the homography matrix may not be calculated accurately. In this regard, the introduction of motion video segmentation is considered to improve filtering.

4 ABOUT THE PROJECT

The project consist of 3 files:

- *evenvizion_component.py*
- *evenvizion_visualization.py*
- *compare_evenvizion_with_original_video.py*

evenvizion_component.py

The implementation of the algorithm of the fixed coordinate system itself and all the main results occur in the *evenvizion_component.py* file. The script accepts the path to the video as input and returns JSON with homography matrices describing the movement between the f_{k-1} and the f_k. It is important that in JSON there're matrices that are describing movement only between two frames. To obtain coordinates in a fixed system, it is first necessary to obtain a superposition of matrices, which was described in 2.

If you want to recalculate the coordinates of some specific objects, you need to specify the path to the json file in: *path_to_original_coordinate* with these objects and you will get *recalculated_coordinates.json* with recalculated coordinates, as well as their visualization.

JSON file format:

{"frame_no": [{"x1": x coordinate, "y1": y coordinate}, ...], ...}

This way, after running the *evenvizion_component.py* script, you get three visualizations (you can control the matching and heatmap of the visualization using the following arguments: *show_matches* and *visualize_fixed_coordinate_system arguments*, respectively).

Modules: *compare_evenvizion_with_original_video.py* and *evenvizion_visualization.py* are used exclusively to get the visualizations of the component's operation. You can find more about the arguments and constants in README. To demonstrate and check the operation of the algorithm, several visualizations were developed, a little more to that.

4.1 Visualization and interpretation

You can see the Matching points visualization on Figure 6. Its description has been stated above.

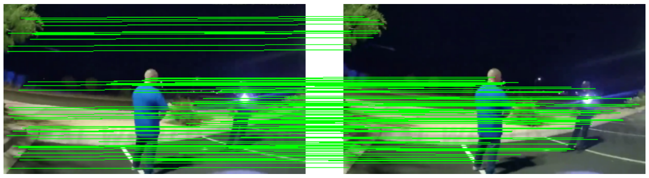

Figure 6: Matching visualization

To demonstrate how a fixed coordinate system works exactly, the following heatmap visualization was developed: Figure 7.

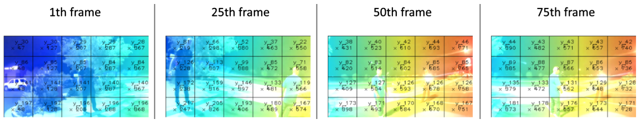

Figure 7: Heatmap visualization

In this visualization, the frame is divided into 20 rectangles, and at each frame the coordinates of the center of the rectangle are recalculated in the new coordinate system. That is, you can see how stationary objects retain their coordinates as the camera moves. This uses the colorization of the frame depending on how far from the origin the object is located. The staining color is determined as follows: the distance from the object to the origin is calculated $r = \sqrt{x^2 + y^2}$, normalized using the *heatmap_constant* argument and, depending on the obtained value, is colored in a certain color. The range of colors: 0 - blue, 1 - red.

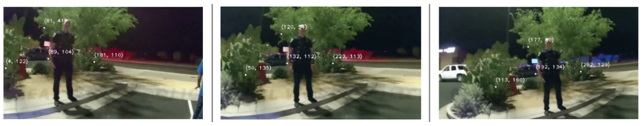

Figure 8: Fixed_coordinate_system_visualization

If you have specified JSON file with coordinates of objects that need to be recalculated you will get *fixed_coordinate_system_visualization* (Figure 8).

The *evenvizion_visualization.py* and *compare_evenvizion_with_original_video.py* are used to render camera stabilization, and only cover camera movement (scaling and rotation is not taken into account by such rendering). Figure 9 show the results of these scripts, respectively.

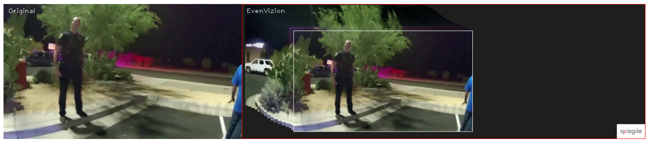

Figure 9: Original_video_with_EvenVizion

5 KNOWN ISSUES

Currently there are the following unresolved issues:

- N/A coordinates
  The coordinates can't be defined when the matching points are clinging to moving objects. This means that the filtration isn't working well enough. The coordinates can't be defined also when camera rotation angle is more than 90°. As a solution to the first problem we now consider applying video motion segmentation to distinguish static points from motion points (taking into consideration the nature of movement). As a solution to the second problem we see the transfer to the cylindrical coordinate system.
- $H = None$
  To find the homography you need at least 4 matching points. But in some cases the 4 points can't be found, and the homography matrices are $H = None$. In a current algorithm version we process such cases this way: if the argument *none_H_processing* is set for *True* we consider the matrix of the previous frame matches the matrix for the current frame $H_k = H_{k-1}$. If set for *False*, then
  $$H = E_3 \tag{7}$$
  meaning that there were no movement in the frame. It is necessary to think over better handling of such situations.
- Error
  There's an inaccuracy in the coordinates. Poorly obtained homography matrix distorts the results of coordinate recalculation. The reasons for that are
  - Poor filtration. If the points catch on a motion object, then the homography matrix will describe not only the camera movement, but also the independent movement of objects (for example, a person's walking).
  - Using the built-in *findHomography()* method from OpenCV library. This function already assumes there is an error in the calculation of the homography matrix.

6 CONCLUSION

Thereby, we get the component which allows assessing the real position of objects relative to each other and to translate the coordinates of the object relative to the frame into a fixed coordinate system. As the main point of this solution was to evaluate plane transformation using key points, we show that the task can be solved even in bad filming conditions (sharp camera movement, bad weather conditions, filming in the dark and so on).