Design and implementation of a technique to track moving objects based on optical flow

Eric Guerrero Font | Carlos García Rodríguez

1. Abstract

This short project shows a methodology to track objects with the same shape and size using optical flow, and with not knowing the number of trajectories to analyze. The objects have many different trajectories and thanks to the optical flow method we can get their velocity and direction easily.

This is a conceptual project that can be adapted to be used in multiples and different fields which may need the tracking of any object with some precision. Some fields for the application could be as a tool to analyze the movements done by a workers in a workbench to optimize the space or may be as a tracker of the usual movements of an animal in a zoo.

To develop the project, we have impose the following conditions:

- The objects are balls of the same size, no matter the color.
- The number of trajectories along the video is not defined.
- The trajectories are not necessarily separated, they can cross.

2. Introduction

The optical flow estimates the velocities of an object in movement between two different images (or between two different frames of a video).

2.1. Horn-Schunck Method:

The optical flow idea was introduced by Horn and Schunk. The idea is that we have a three dimensional function f(x,y,t). Rows and columns are x and y, and different frames are t.

We get two frames, frame at time t and frame at time t+1. These frames are taken in short time. Typically we get 30 frames per second, so the frames have changed very few.

So, if a pixel is taken in [x y] in a frame taken at time t, and we look at a pixel close to x (dx) and y (dy), taken in the frame t+1 they will have the same intensity. That is called brightness constancy assumption.

$$f(x,y,t) = f(x+dx, y+dy, t+dt)$$

Then, we apply the Taylor series.

$$f(x,y,t) = f(x,y,t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt$$

Developing this equation, we obtain:

$$f_x dx + f_y dy + f_t dt = 0$$

And dividing everything by dt we obtain the Optical Flow Constraint Equation:

$$f_x u + f_y v + f_t = 0$$

Where,

- f_x, f_y and f_t are the spatiotemporal image brightness derivatives.
- u is the horizontal optical flow.
- v is the vertical optical flow.

We have two unknowns (u and v) and one equation. So we need some method to solve this constrained equation. In that case, we have decided to use the Horn-Schunk Method.

They adapted this to an optimization problem. The Brightness constancy must be zero and the constraint forces the optical flow to be smooth. That way, we have the next equation.

$$\iint [(f_x u + f_y v + f_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2)]dxdy$$

The squares are because we need a very small area and smooth small. Smooth means not much change, and that's why we square the values. The double integral is used for evaluate every pixel (rows and columns). Ideally the first part will be zero and the second party very small. The Horn-Schunck method minimizes this function to obtain the velocity u and v for each pixel in the image.

The Horn-Schunk method follows the next technique to obtain the velocities. (Matlab, 2014)

1. Compute f_x and f_y using the Sobel convolution kernel: [-1 -2 -1 ; 0 0 0 ; 1 2 1], and its transposed form for each pixel in the first image.
2. Compute it between frames 1 and 2 using the [-1 1] kernel.
3. Assume the previous velocity to be 0, and compute the average velocity for each pixel using [0 1 0 ; 1 0 1 ; 0 1 0] as a convolution kernel.
4. Iteratively solve for u and v.

In a mathematical way (α scales the global smoothness term):

$$u_{x,y}^{k+1} = \bar{u}_{x,y}^{k} - \frac{f_x\left(f_x \bar{u}_{x,y}^{k} + f_y \bar{v}_{x,y}^{k} + f_t\right)}{\alpha^2 + f_x^2 + f_y^2}$$

$$v_{x,y}^{k+1} = \bar{v}_{x,y}^{k} - \frac{f_y\left(f_x \bar{u}_{x,y}^{k} + f_y \bar{v}_{x,y}^{k} + f_t\right)}{\alpha^2 + f_x^2 + f_y^2}$$

3. The explanation of the method

3.1. Evolution of the short project

The first contact with the optical flow was a little confusing. We were moving objects with our hands and trying to detect and track them using their velocity, but the arm was a big impediment because our objects weren't the only thing in the image that was moving. The optical flow detected our objects and our arms. That's why the optical flow is used (usually) as a car tracking, because it works much better when the objects are moved by themselves.

So, the first approach that we made was to avoid the arms directly, and for that, we throw the balls to the floor and let them roll. Doing this, the method worked much better. After applying different techniques we have been able to threshold the movement of the arm and detect all the trajectory of the ball.

For a smaller computational cost we developed a technic to search in the reduced area where is located the movement.

Then, the problems appear when more than one ball was in our work field and one passes near the other. One level was to distinguee different separated trajectories, and a harder one was with closer trajectories. Thanks to the optical flow we could separate them using the directions of the balls for each position.

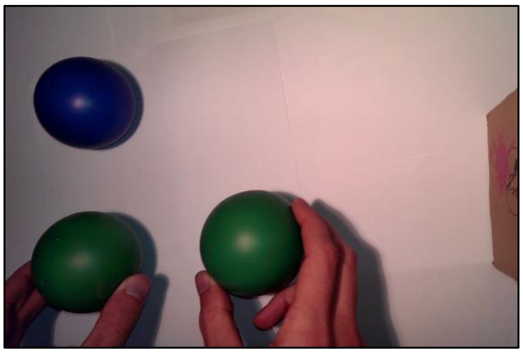
Figure 1. One of the first samples videos

Finally, we wanted to adapt the code to real time, but Matlab was too slow to calculate every frame with the enough time. So we had to make some changes in the code to economize and finally it was useful, but depending on the characteristics of the light and other parameters, it must be improved better to have good a real time tracking.

3.2. Definitive method

In this section we are going to go in depth with our code.

Using a zenithal stationary camera we have recorded some testing videos with moving objects for work directly with those. Once we have our sample videos, we will study and implement different steps. For a better understanding, the next figure will help us to follow the method.

The code is clearly separate in two main parts, a loop that analyses all the frames along the video and another part with two functions that generate and plot the trajectories of the objects. The main code has as input the serial of frames of the video and as outputs the position and directions of each detected object in each frame.

Since there is a long iteration process at the beginning are located the settings of the functions that are used recursively along the iteration process. Then we have the block for the video selection, that is linked to an interface with this propose, a function which will import some necessary data depending on the video.

3.2.1. Main code (MatlabCode)

Initial Settings

To develop all the code, some variables have been predefined. We can find the optical flow, defined with the Horn-Schunk algorithm, with a frame delay of one, what will calculate the difference between two consecutive frames. As an output of the Optical Flow function, is obtained an image with complex numbers in each pixel, which represents the magnitudes and the directions of the velocity in this pixel. There are also other settings for plotting results, assigning constants, the paths of the sources, etc.

Video Reader

Once the chosen video is selected in the interface, the "VideoReader" function will set the video to be read when another function "step" calls it.

Frame Acquisition

When all the settings are executed, we enter in the while loop, where for each frame will be done all the different operations. It calls the function video reader and changes the sizes and the format for every frame for adjusting it to the needs of the program.

Optical Flow

Once the frame is adapted (greyscale, resized…) the method optical flow is applied over the frame. Optical flow works with two consecutive frames; this is the reason why we have created a first

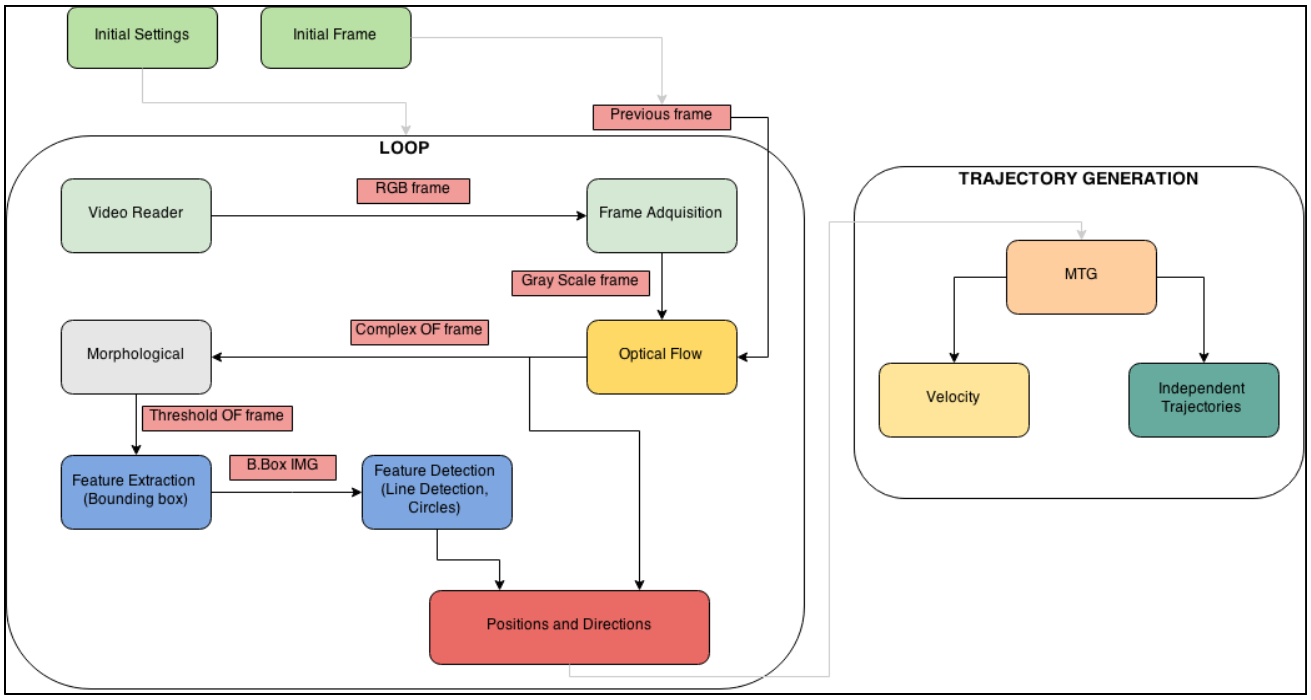

Figure 2. Coding Scheme

frame, outside the loop, for being able to start the calculations. And also, a Gaussian filtering is applied to reduce the noise that may be formed in the image.

Morphological

The morphological operations have as aim to erase the moving zones of the frame with optical flow applied, that are not our studying objects. For that, the frame with complex numbers has been transformed in the magnitude frame and then a threshold is applied to erase the slowest motion areas.

In order to fulfill this target we have applied closing, filling and opening operations, in this same order to obtain a very precise binary image where our moving object is the only (or almost) that will be represented.

Feature Extraction

In order to reduce the computational cost, we have developed an algorithm that analyses only the moving part(s) of the frame. This algorithm consists in extract the area and the bounding boxes of each region. Then apply a function to search the objects only in the bounding box of each region if the area of this region is greater than a threshold. Finally, we obtain a vector with all the centers of the objects and, in this case, the radius of the circles.

Positions and directions

On the one hand we have the positions, which are the centers of the objects founded in the frame. On the other hand, the direction is calculated by the average value of the pixels where the object is located, in the complex frame of the optical flow results. In our case the object is a ball so the pixels selected are all the neighbors inside of a circle with the same center and radius than the ball detected.

So, the product of this phase are two arrays, one for the positions, two columns per frame (x, y) and as rows as points are detected per frame, and another for the orientations with the same dimensions.

Video players

This part of the code is for a better visualization of the process, there is the plotting of the lines that give the velocity field obtained with optical flow, the bounding boxes that surround the moving areas, and the circles that mark the position of the balls.

Generation and plotting of trajectories

When the loop is finished, there are not more frames to analyze, starts the phase to build and plot the trajectories. That is done by means of two functions designed on porpoise for this method.

3.2.2. Main functions

FindCircles ***(Code)***

Given a bounding box of a moving area it extracts a sub image of the original frame to apply an edge detector, with a Sobel mask, to improve the results of the function *'imfindcircles()'* that uses the Hough transform, a line detector.

MTG (MultiTrajectory Generation) ***(Code)***

This function is able to separate different trajectories from the Position and Direction arrays. It consist in a cascade of loops to create trajectories and assign new points to the existent ones. After some array initializations a principal loop goes along all the frames and look for the following points in order to classify the points in an indefinite number of trajectories, depending on the video:

- Look if a point can be included in more than one trajectory, it happen when the analyzed point is close enough to more than one point assigned to different trajectories, if that is the case looks for the trajectory with the minimum change of direction. This part is mandatory when the objects are to close and crossing, but not needed if they are further than the distance threshold imposed.
- If there isn't a conflict as the previous and a point is inside the distance threshold of a trajectory, the point is assigned.

- For a point that not accomplish the distance threshold for any trajectory a new trajectory is created
- In each assignation, the assigned point in the position vector is set to zero. When all the points for a frame are zero the non-updated trajectories get the same position than before and a mark of a delay.

Finally when all the points have been assigned there is a filter to avoid noise trajectories. It has a threshold of number of points per trajectory.

MTP (MultiTrajectory Plotting) ***(Code)***

Principally with the separated trajectories and directions this function plots the results, position and velocities for the deferent objects.

4. Testing

To start we have to execute the Interface.m and selected the video that we want to test, for adding new videos one have to add the new path in the code.

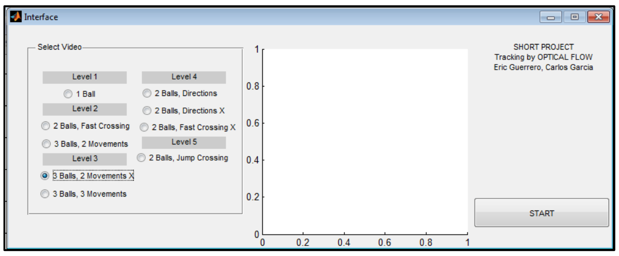

Figure 3. Interface of the program

After some initialisations the MainCode starts to run plotting the direction points and playing some video frames, this are:

- The Magnitude Player, which shows the magnitude of each pixel of the complex frame given from the optical flow. It gives an impression of where is focused the intensity movement to analyze the results.

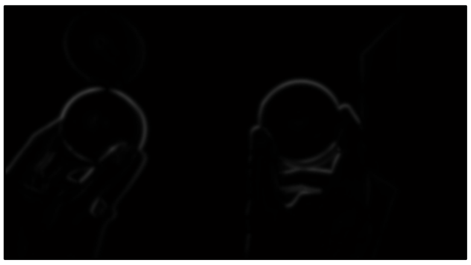

Figure 4. Magnitud Player

- The OPThreshold Player, which displays the binary image after apply all the morphological operations in the moving part of the image. One can see the precision of the captured movement.

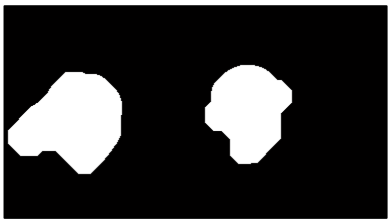

Figure 5. OPThreshold Player

- The Shapes player has a lot of information, the blue lines mark the velocity field obtained by means of the optical flow, the rectangles are the bounding boxes of the areas present in the OPThreshold, which are the ones with movement to analyze and look for the balls, that are marked with red circles.

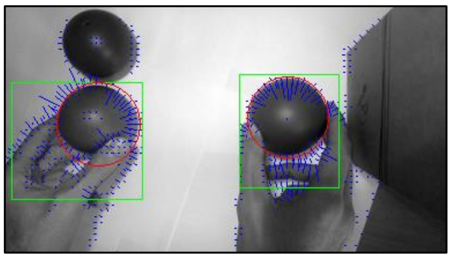

Figure 6. Shapes Player

- Also there is a plot that is refreshed with the new points to see the points and directions registered in the arrays.

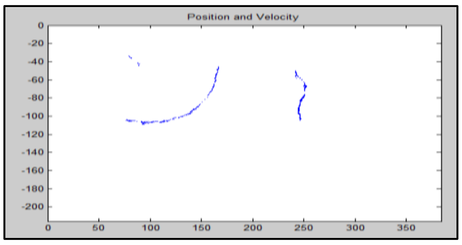

Figure 7. Plot with directions

Finally, when the loop ends with all the frames, we get the trajectories and velocities in the following form.

The first plot shows the trajectory separated for each object in millimetres. The second one shows the velocity for each object in each one of all the frames of the video.

After checking the videos, we can appreciate how accurate are the trajectory representations. We have tried to draw in the air some letters and symbols, and we have seen later very well represented in the trajectory plot.

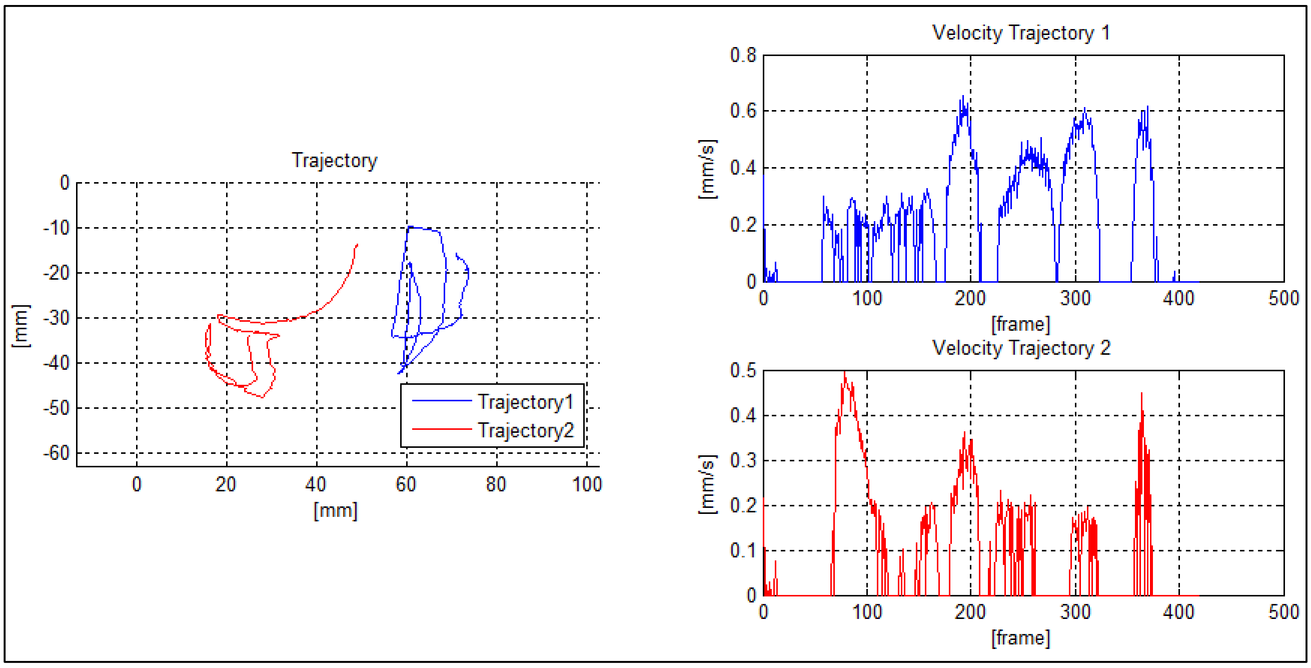

Figure 8. Trajectories and Velocities

5. Conclusions

Comparison with Optical flow			
Name	**Link**	**Frame average (s/frame)**	**Comment**
Optical Flow (BBx)	MainCode.m	0,2742	
Optical Flow	comp_nobbox.m	0,3173	
Eco Code	comp_eco.m	0,1040	Without unnecessary things, resize, etc.

Table 1

Comparison of Optical flow			
Name	**Link**	**Frame average (s/frame)**	**Comment**
Optical Flow (BBx)	compOF.m	0,2046	Without evaluating trajectories, only locating centres
Optical Flow	compOF_nobbox.m	0,2278	
No Optical Flow	compOF_noOF.m	0,2204	

Table 2

After doing an intensive work on our code modifying some parameters and techniques we have done two comparison tables with computer time processing, using as example de “VIDEO0159”.

As we can see in the Table 1, the bounding boxes are very useful, mainly when the objects are recorded from a farther position and then, they will fill less proportion in the screen and it means that the code will compute it faster. It will be very helpful if the object has a very difficult morphology. Also, in the Table 2 we can see that without using the optical flow, it is much faster than without de Bounding Boxes.

This project began maybe a little confusing, because we had a lot of problems with the arm, which was moving with our objects too, and didn’t let as to work correctly with the optical flow. After a lot of background teamwork, we find the way to avoid the arm and then start with the soul of the optical flow. This code is much optimized and has a lot of future possibilities, just adding some news functions to the main code.

For the next version of the code, we want to integrate the generation of the different trajectories online inside of the loop, and not at the end of all the calculation. And also improve the online video acquisition and processing, because it still having some errors in this beta version, because the computational time is higher than the frame per second relation. For this, the Eco Code (Table 1) has been created. This code deletes all the unnecessary and visual parts reducing the computational time to it minimum.

On the other hand, a very complete program has been realized, the powerful technique to separate trajectories using their directions allows us to track objects even if the crosses one over the other. Even if all the objects are identical, we can also detect them and differentiate the trajectories thanks to the directions and de proximity of the positions.

6. References

MathWorks. (2014). Retrieved November 2014, from Optical flow for motion estimation in video: http://es.mathworks.com/discovery/optical-flow.html

MathWorks. (2014). Retrieved November 2014, from Tracking Cars Using Optical Flow: http://es.mathworks.com/help/vision/examples/tracking-cars-using-optical-flow.html?refresh=true

Matlab. (2014). *Optical Flow.* Retrieved from http://es.mathworks.com/help/vision/ref/vision.opticalflow-class.html

Vaca-Castano, G. (2014). Retrieved from Matlab Tutorial. Optical Flow: http://www.cs.ucf.edu/~gvaca/REU2013/p4_opticalFlow.pdf