An Indirect, Multiscale and Robust Approach to Global Motion Estimation

Filippo Daniotti
DISI, University of Trento
Trento, Italy
filippo.daniotti@studenti.unitn.it
232087

Samuele Conti
DISI, University of Trento
Trento, Italy
samuele.conti@studenti.unitn.it
229294

***Abstract*—The problem of global motion estimation (GME) deals with the separation, in a video sequence, of two different types of motion: the *egomotion* of the camera recording the video, and the actual motion of the objects recorded. The literature presents a number of possible approaches to the GME problem; here, we aim to combine some of the most effective strategies to compute the motion of the camera. We adopt an indirect approach based on the affine motion model. Following a multi-resolution approach, at each level we compute the parameters of the motion model that minimize the error of the motion field. Then, we use those parameters to spot the outliers of the motion field in the next level; when computing the optimal parameters in the next level, we remove the detected outliers, in order to obtain a robust estimation. Results are reported both in a qualitative and quantitative way.**

***Index Terms*—multi-resolution, robust, indirect, global motion estimation, computer vision, video processing**

I. Introduction

The problem of global motion estimation (GME) deals with the separation, in a video sequence, of two different types of motion: the egomotion of the camera capturing the video, and the actual motion of the objects in the scene. This task is crucial for a number of application, for instance:

- removal of camera shaking or noisy motion;
- study of the motion of the camera itself;
- object segmentation and recognition;
- object tracking.

For these reasons, motion estimation is a problem that has long been studied, hence in the literature we can find a variety of approaches to the problem. In this work we concentrated on some of the most effective approaches and combined them to get an indirect, multi-resolution and robust approach to GME.

The document is structured as follows: in Sec.II we sum up the fundamental aspects of the GME problem and the standard approaches to solve these problems. In Sec.III we discuss the strategies we chose for our implementation, and in Sec.IV we present the results of this work. The conclusions in Sec.V sum up the work and draw a path for future development.

II. Global Motion Estimation

Videos are, at their core, sequences of images called frames; we can spot the motion happening between a given pair of frames by detecting which pixels changed position with respect to that pair of frames.

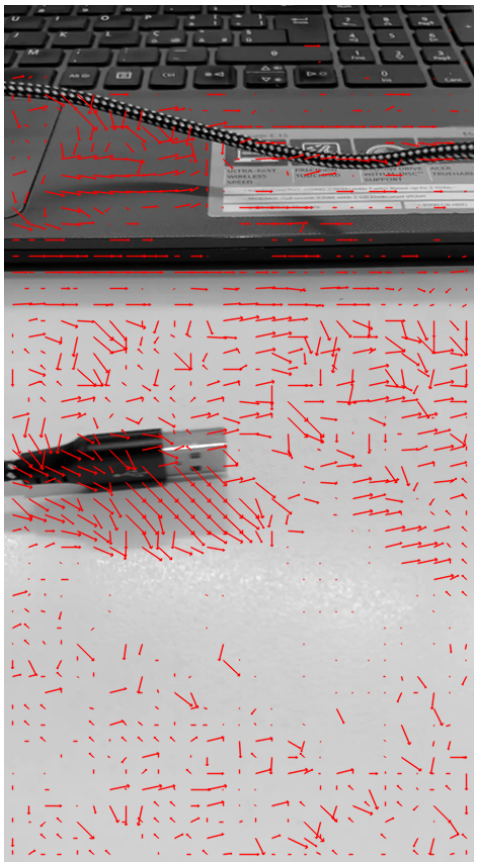


Fig. 1: Needle diagram between two frames of a video sequence.

This information can be encoded in a human-understandable format by presenting the *needle diagram* of the two frames; this kind of representation is shown in Figure 1.

It is pretty clear what information this figure is conveying: for each couple of frames we mark with an arrow the shift that the pixel has made. This gives us an indication of the actual movement of the recorded objects.

However, the motion that we are able to extract from the

two frames is what is usually called "apparent motion", which is a combination of two different factors:

- the global motion, which corresponds to the motion of the camera;
- the effective motion of the objects recorded.

An important task in computer vision is the global motion estimation, which aims to distinguish these two different types of motion. In particular, this information is highly important for a number of applications, amongst which we find video coding, motion compensation, object segmentation and ego-motion estimation.

The task of distinguishing which movement is to be classified as global motion and which one is to be classified as object motion instead is not so easy. In fact, there is no clear feature that distinguishes the moving objects from the static background, and this leads to the need for complex tools to differentiate the two types of motion.

A. Motion Models

The main approach to global motion estimation consists in using a motion model to describe the motion of the camera.

Motion models describe the motion of the entire visual field through an equation. Basically, the underpinning idea is to write an equation which, given a pixel in a certain position (x, y), returns the position that the same pixel will have in the next frame, if its motion is due only to camera motion.

There is a number of motion models that can be used to describe global motion, ranging from very simple translational models - that can describe only translation motion - to more complex models, allowing to describe pan, tilt, zoom and roll motion patterns. The reader can use as reference Figure 2 as an illustration of these types of motion.

All the motion models are defined as functions of some parameters; to get the camera motion in a certain sequence we need to estimate the parameters of the motion model for that precise sequence. Taking as example the motion model we used in this project - the affine motion model - and all the models that describe a rigid motion, the shift in the position of a pixel can be described as in Equation (1).

$$p' = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T_X \\ T_Y \\ T_Z \end{bmatrix} = Rp + T \quad (1)$$

Here p' represents the position of the pixel after the camera motion. The matrix R is used to describe rotation and the vector T for the translation component.

The affine motion model can describe simple motions that can be described by linear functions as translation, rotation and zooming (see [1] for in depth explanation).

Usually the affine model is used under the assumptions that the rotation and translation are on the image plane, and that depth Z is approximately constant (as from [2]); these assumptions are reasonable as long as the motion occurring between the two frames we are analyzing is not too large. Under these assumptions, the affine model can be represented as Equation (2), where we need to estimate the 6 parameters $a = [a_1, a_2, a_3, b_1, b_2, b_3]$.

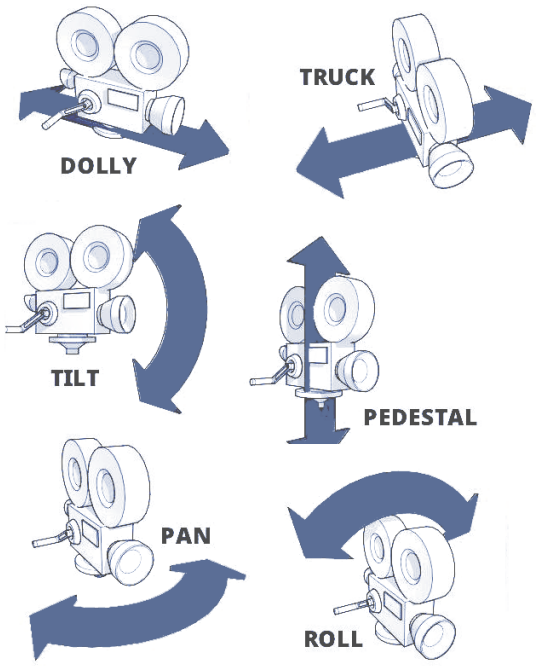


Fig. 2: Different types of camera motion. Source: https://cinestudy.org/2020/06/29/camera-movement/.

$$\begin{aligned} \Delta x &= a_1 x_0 + a_2 y_0 + a_3 \\ \Delta y &= b_1 x_0 + b_2 y_0 + b_3 \end{aligned} \quad (2)$$

B. Parameter Estimation

There are two different approaches to estimate the parameters of a motion model:

- a direct approach, which computes the best parameters by optimizing an overdetermined system in the pixel domain;
- an indirect approach, which minimizes the approximation error between the motion field obtained with an estimation and the motion field obtained by the motion model.

The first approach derives from the equation of the optical flow; in the case of the affine model, it describes the motion of each pixel as in Equation (2). This equation can be applied to any subset of pixels of the frames, therefore the system turns out to be overdetermined and we can solve it using some optimization method, for instance gradient descent.

The second approach - indirect estimation - aims to minimize some error measure between the ground truth motion field and the motion field returned by the motion model.

Therefore, indirect estimation needs, as a preprocessing step, to get the motion field from a motion estimation method - e.g. block-based motion estimation. This motion field is then used as *ground truth* to find the parameters of the motion model that minimize some error measure between this motion field and the one returned by the motion model.

An aspect of this procedure worth highlighting is that it works on the motion field rather than directly on the pixel domain. Hence, it can compute motion of blocks instead of single pixels, reducing the overall complexity of the procedure.

The process is summarized in Equation (3), where

- a is the parameter vector;
- p is the single pixel (or block);
- $gt(p)$ is the ground truth motion vector for p;
- $MM(p, a)$ is the motion vector that the motion model would predict for p with parameters a;
- E is some error measure that accounts for the difference between the two motion vectors, usually a p-norm distance.

$$a = \arg\min_{a} \sum_{p} E(gt(p) - MM(p, a)) \tag{3}$$

Once we minimize Equation (3), we obtain the values of the parameter vector a that encode the camera motion of the video sequence we are analyzing.

C. Robust Estimation

Since motion in video sequences can be pretty fast and complex, it is often the case that some parts of the scene move in such a way that they insert noise in the motion estimation process. In particular, it is quite expected to find fast moving objects which we refer to as *outliers*: they are objects manifesting some kind of intrinsic motion - not caused by camera motion - which pollutes the estimation of the parameters of the motion model.

In order to solve this problem, a common approach is to introduce some strategy to remove the outliers from the motion estimation process and, consequently, obtain a robust estimation of the motion field.

III. Implementation choices

As stated in Section II-A, we decided to use an indirect method with affine model for motion estimation as in Bergen et al. [3]; also, we decided to obtain a robust estimation by using a strategy similar to [4]–[6], where they adopt a hierarchical approach to obtain a robust estimation using coarse resolution to guide the search for the best parameters at a finer resolution.

In the following sections, we will provide an in-depth explaination of our solution and present the pseudocode of the algorithm (see Algorithm 1). A schematic representation of the procedure is presented in Figure 3.

A. Block-Based Motion Estimation

The first piece of the architecture is the algorithm to compute the motion field to be used as ground truth. In our case, we decided to implement a block-based motion estimation (BBME) pipeline, since computing the motion field on blocks rather than single pixels is much less computationally expensive.

The main idea of BBME is to divide the frame in blocks and search for each block in the previous frame the corresponding block in the next frame. This gives us the displacement of the

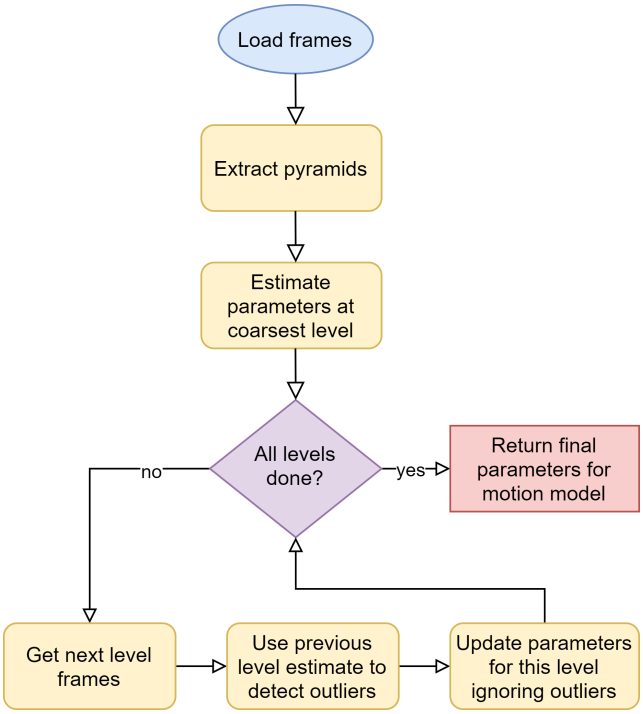


Fig. 3: Schema of the flow of implementation of the global motion estimation pipeline.

block between the two frames, which is accounted as *motion vector* of the block. From this fundational principle a number of implementations have been proposed, differing mainly in two aspects:

1) the size and shape of the block;
2) the path that the searching procedure follows.

During the project we implemented a number of these techniques:

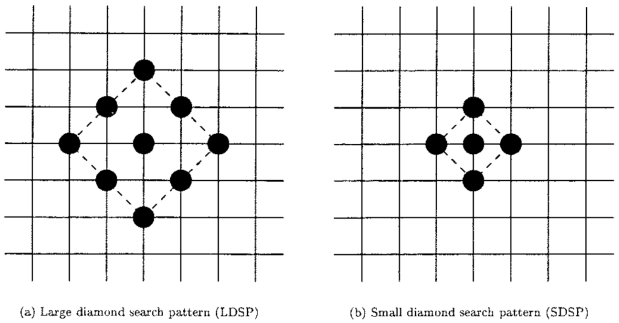
- exhaustive search;
- two-dimensional logarithmic search;
- new three-step search, from [7];
- diamond search, from [8];

In the final implementation, the block matching method used is the one presented in [8], known as diamond search (DS), with a mean square error measure to compute (dis-)similarity between blocks. Nonetheless, all the other methods are still present and by slightly changing the code they can be tested as well.

Here we briefly discuss the algorithm that we choose to implement and propose a small example for clarification. The DS algorithm uses a peculiar diamond-shaped search pattern, which defines which positions will be used as target points where to center the block in the next frame to see if it corresponds to the block in the previous frame. In particular, the algorithm uses two slighlty different shapes depending on the current iteration (see Figure 4):

- large diamond search pattern;
- small diamond search pattern.

The procedure is the following:



(a) Large diamond search pattern (LDSP) (b) Small diamond search pattern (SDSP)

Fig. 4: Representation of the two DS algorithm search patterns.

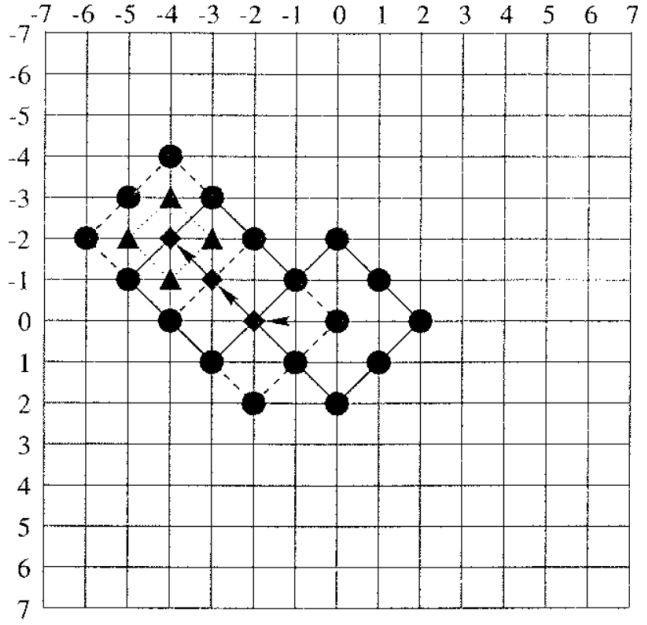


Fig. 5: Example of run of diamond search algorithm for block matching motion estimation.

1) the search starts with LDSP, meaning that we check the blocks centered in each one of the 9 positions; if the best match is the central position then we pass to the second step, otherwise we move to the best match position and re-use the LDSP;
2) in the second step we use SDSP and look again for the best matching position among the 5 available, then return the best match position.

In Figure 5 we can see an example of run. We can spot the three large-shape diamonds (the contour position are marked with circles) and the small shaped diamond (contour positions marked with triangles). In this example the block matching algorithm started from $(0, 0)$ then moved with the large diamond to $(-2, 0)$, because this was the position that minimized the error between the block in the previous and the next frame. Then again with the large diamond the algorithm reached $(-3, -1)$ and then $(-4, -2)$. Here the best matching position for the large diamond shape was $(-4, -2)$, therefore the algorithm stopped using the large diamond and started with the small diamond. Finally, the small diamond returned once again as best match position $(-4, -2)$, which was then returned as final matching position for the block in the previous frame.

B. Affine Model Parameter Estimation

In Section II-A we presented the affine motion model, which is the one we decided to use in our implementation.

To get the motion vector of a certain position $p = (x, y)$ in the previous frame, we can solve the following formulation of the affine model:

$$d(p, a) = A[p]\ a \tag{4}$$

where $d(p, a)$ is the displacement for the position $p = (x, y)$, given the parameter vector a; $A[p]$ is an intermediate matrix that is computed as

$$A[p] = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix}$$

By encoding the estimation of the motion field in this way, the p-norm error we are trying to minimize (see Equation (3)), gets rewritten in this way:

$$E = \sum_p |d(p, a) - d(p)|^P \tag{5}$$

Here we introduce $d(p)$ as the ground truth motion vector (the one we get from the BBME as in Section III-A) and P which specifies the grade of the p-norm used. In this case we decided to set $P = 2$, following the procedure explained in [2]. Pay attention to the difference, in the formula, between p (the position) and P (the grade of the p-norm).

The next step is to minimize the error in Equation (5) with respect to the parameter vector a. The procedure in [2] states that we need to compute the gradient of E w.r.t p, and then get the value of a that sets the gradient to 0.

By using the matrix formulation explained in Equation (4) and by setting the gradient $\nabla_a E = 0$, we obtain the following result:

$$a = \left(\sum_p A[p]^T A[p] \right)^{-1} \left(\sum_p A[p]^T d(p) \right) \tag{6}$$

There are three interesting notes about Equation (6):

1) we are iterating over a set of positions p: it's interesting to notice that these points do not need to cover the complete scene, i.e. we can use only a subset of the image to optimize the parameters;
2) each one of the points can be weighted with a weight $w(p)$ as shown in Equation (7);
3) we can actually split the computation of the parameter vector $a = [a_1, a_2, a_3, b_1, b_2, b_3]$ in two different slices, $a_x = [a_1, a_2, a_3]$ and $a_y = [b_1, b_2, b_3]$, which can be computed in a specular way as shown in Equation (8)[1]; this is useful to further reduce the complexity of the procedure.

[1]Notice the shape of the intermediate matrix $A_x(p) = [1, x, y]$ in Equation (8).

Algorithm 1: High-level pseudocode of our solution.

```
Input: previous_frame, current_frame
Output: a    // parameter vector

prev_pyrs = get_pyr(previous_frame)
cur_pyrs = get_pyr(current_frame)
a = first_estimation(prev_pyrs.next(), cur_pyrs.next())
foreach l in levels do
    prev_frame = prev_pyrs.next()
    cur_frame = cur_pyrs.next()
    ground_truth_mfield = BBME(prev_frame, cur_frame)
    estimated_mfield = affine(a)
    outliers = detect_outliers(ground_truth_mfield, estimated_mfield)
    a = minimize_error(prev_frame, cur_frame, outliers)
end
return a
```

$$a = \left(\sum_p w(p) A[p]^T A[p] \right)^{-1} \left(\sum_p w(p) A[p]^T d(x) \right) \quad (7)$$

$$a_x = \left(\sum_p w(p) A_x[p]^T A_x[p] \right)^{-1} \left(\sum_p w(p) A_x[p]^T d_x(p) \right) \quad (8)$$

The form of the solution that we used in the implementation is the one reported in Equation (8).

C. Hierarchical Robust Estimation

The last part of the project is a strategy used to provide a robust estimation of the parameters for the affine motion model. The proposed implementation is inspired by [4], where the computation of the motion field takes place at different levels of resolution subsequently.

The main idea is to compute the parameter vector a at a coarser resolution, and then use this coarser estimation to extract information useful for regularizing the estimation at finer resolutions. In practice, this translates in our code being structured as shown in Figure 3: first of all, the two frames we use as previous and next frames are transformed in multi-resolution pyramids, and then used iteratively for the process of parameter estimation.

Here we report a summary of the procedure used to implement the hierarchical robust estimation:

1) for each iteration we get as input the frames at the corresponding level l of the pyramid (at a finer resolution w.r.t the previous iteration at level $l-1$) and the parameter vector a_{l-1} computed in the previous iteration;
2) we use the scaled parameters a_{l-1} to compute the error for each position p in the frames at level l; the error is computed as difference between the ground truth motion vector in p and the motion vector obtained with the motion model in p with parameters a_{l-1};
3) to detect the outliers we order all positions p by the magnitude of the error, we set a fixed percentage `outlier_percentage` and all the positions that fall in that percentage of higer-error-positions are marked as outliers;
4) once we have marked all the outliers, we can proceed and compute the new estimate of the parameter vector a_l ignoring the outliers;
5) the new estimate will be returned to the next level $l+1$ of the hierarchical procedure, or simply returned to the caller of the function when we reach the last level.

In Algorithm 1 we present the pseudocode summarizing the procedure.

IV. Results

A. Qualitative results

The easiest and most effective way to show the results of a global motion estimation pipeline is to use qualitative information, as the reader will understand it straightforwardly.

One of the operations that shows intuitively the implications of global motion estimation is camera motion compensation. In short, if the video is recorded by a camera which is moving, we will see the two types of motion explained in Section I: apparent and real. The aim of camera-motion compensation is to detect and remove the apparent motion which is caused only by the (ego)motion of the camera.

To show the results obtained we reported in image Figure 6 an example in which the reader can observe:

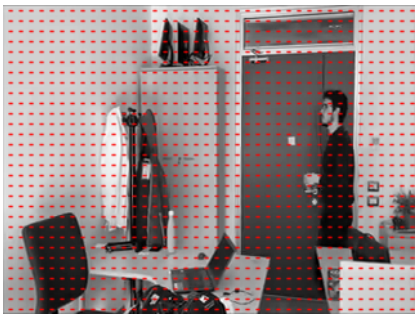
- the previous frame in Figure 6a;
- the next frame in Figure 6b;
- the compensated frame, obtained by compensating camera motion in the previous frame, in Figure 6c;
- the absolute difference between next and previous frame in Figure 6d, which gives an idea of how strong the motion in which parts of the image is;
- the motion field generated by the camera, as returned by our estimation procedure in Figure 6e;
- the absolute difference between the next frame and the compensated one in Figure 6f.

(a) Previous frame.

(b) Next frame.

(c) Compensated previous frame.

(d) Absolute difference between current and previous frame (white areas are where we detect motion).



(e) Global motion field estimated by our procedure, it corresponds to the motion generated by the camera.

(f) Absolute difference between current and compensated frame, the only object in white is the only actually moving.

Fig. 6: Qualitative results.

a) How to interpret qualitative results: We should be able to notice in the results that the motion that we detect between the two frames (previous and next) is significantly large and noticeable. In fact, by using the absolute difference between frames, we can see that a big part of the visual field seems to be moving in Figure 6d.

However, by computing the motion generated by the camera egomotion, we find out that most of the shift we register is apparent. By estimating the motion generated by the camera, which is reported in Figure 6e, we can compensate the apparent motion in the previous frame. When we compensate the previous frame we *remove* all the motion due to apparent motion; therefore, if we compare the current frame and the compensated frame (see Figure 6f) we are able to isolate real motion.

The results shown are consistent since the video from which the frames are taken was recorded with the camera moving in the horizontal direction while the person in the video was walking.

B. Quantitative results

To perform a numeric quantitative analysis of the results produced by the algorithm we used, once again, the compensation of the previous frame. Put it simply, in a video sequence where there is no real motion, but only camera motion, the compensation of the apparent motion should be enough to make the current and the compensated frame identical.

Therefore, in Table I, we present some PSNR values for different video sequences. We annotated the table with some considerations about the videos, as PSNR value is influenced also by properties of the video like the strength of motion or the presence/absence of real motion.

For instance, in the case where there are medium or big objects, like the last two entries of the table, the PSNR turns out to be low, since the model is not able to distinguish between object and camera motion.

Better results can be observed in the first two entries of the table, where the moving object are considerably smaller with respect to the background, and the background presents some complex pattern, which enables the algorithm to better detect its motion.

V. Conclusions

This project addresses the task of global motion estimation through the use of an indirect method for the estimation of apparent motion which is based on the affine motion model.

During the implementation of the tool, we built the following scripts:

- `bbme.py` as Python module containing all the functions related to the various block matching motion estimation procedures that we have been trying;
- `motion.py` as Python module containing all the functions and constants needed in order to perform camera motion estimation and motion compensation;
- `results.py` as example of use of the packages modules to produce the results we used also to create this report.

Video	Properties	Average	Variance	Maximum	Minimum
`pan240.mp4`	Fast motion	22.724	5.125	27.802	17.981
`coastguard_qcif.mp4`	Two object moving, background moving	22.733	2.194	26.875	15.158
`foreman.mp4`	Big object moving, still background	19.677	18.443	30.436	11.746
`numeri_del_piero.mp4`	Medium object moving, moving background	19.072	13.642	47.722	16.323

TABLE I: PSNR results on some meaningful video samples.

The approach we used proved to be efficient and robust, as long as the motion in the image is not too complex and as long as the background is not uniform and, therefore, the BBME algorithms are able to spot its motion.

The actual implementation is to be found in the following GitHub repository https://github.com/Samaretas/global-motion-estimation, where the reader can find all the commented scripts and try out the code. In the repository it will be possible to run the basic example with `pan240.mp4`, whilst the other videos of the dataset can be found here https://drive.google.com/drive/folders/1gZisWe4DEWpb_CoHkTi6OKnxhl5Ca_mT?usp=sharing.

For more information on the code and how to perform a full-cycle execution of the pipeline, please refer to the Github repository.

This project presents some limitations that will be handled in future developments:

- first, there is no automated way to set the distance between previous and next frame, in fact, often we need to set more than one frame of distance between *previous* and *next*, otherwise the motion is too small to compute GME;
- then, to have a comparison with other methods, like the ones cited before, we would need to insert this piece of code in a pipeline for video encoding, to record its performance both in accuracy and efficiency;
- finally, there are some intrinsic limitations to the task of GME, some of them are even presented in the results reported here, for instance the setting in which the scene is too noisy of the objects are bigger than the actual background.

References

[1] W. Ren, W. He, and Y. Cui, "An improved fast affine motion estimation based on edge detection algorithm for vvc," *Symmetry*, vol. 12, p. 1143, 07 2020.

[2] Y. Wang, J. Ostermann, and Y.-Q. Zhang, *Video processing and communications*, ser. Signal Processing Series. Prentice Hall, 2002, includes bibliographical references and index.

[3] J. R. Bergen, P. Anandan, T. J. Hanna, and R. Hingorani, "Hierarchical model-based motion estimation." Springer-Verlag, 1992, pp. 237–252.

[4] F. Dufaux and J. Konrad, "Efficient, robust, and fast global motion estimation for video coding," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 9, pp. 497–501, 02 2000.

[5] S. Ayer, P. Schroeter, and J. Bigün, "Segmentation of moving objects by robust motion parameter estimation over multiple frames," in *Proceedings of the Third European Conference on Computer Vision (Vol. II)*, ser. ECCV '94. Berlin, Heidelberg: Springer-Verlag, 1994, p. 316–327.

[6] Y. He, B. Feng, S. Yang, and Y. Zhong, "Fast global motion estimation for global motion compensation coding," in *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, vol. 2, 2001, pp. 233–236 vol. 2.

[7] R. Li, B. Zeng, and M. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp. 438–442, 1994.

[8] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE transactions on image processing : a publication of the IEEE Signal Processing Society*, vol. 9 2, pp. 287–90, 2000.