TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Sahar Husseini

A SURVEY OF OPTICAL FLOW TECHNIQUES FOR OBJECT TRACKING

Bachelor of Science Thesis

ABSTRACT

There are many object tracking algorithms using optical flow methods. Existing literature in flow estimation are abundant, however, there is not any method found to extract perfect flow data. So making an optimal choice of the methods that is suitable for moving object tracking applications remains an open problem. The purpose of this Thesis is to provide a survey of existing optical flow techniques for object tracking applications. We experimented state of the art flow methods using several video clips of a static camera including indoor and outdoor scenes. Furthermore, we present a novel object tracking system for sparse and dense flow extraction methods by using inverse flow optimization technique. Sparse algorithm process only some pixels from whole image, which are easy to follow and dense optical flow algorithm can process all pixels in image.

The both flow methods have some hard parameters in their implementations. In hard parameter coding the source code, have to be changed any time the input data or desired format changes. It means users have to carefully test and forethought the system code performance. The best way to set the hard parameters is cross validation over user desire dataset. In order to show the performance of the proposed tracking algorithm, some experiments have been performed under higher frame per second video clips. The results show that selection and cross validation of hard parameters for flow extraction methods are vital to get the best results.

Tampere, 25.8.2017

CONTENTS

1 INTRODUCTION

Over the last decades computer vision has been widely used in our daily life. Humans use their visual system to see and analyze the world surrounding them and computer vision automates these tasks for them. The role of computer vision is to analyze, process, understand and extract the useful information of data captured by different types of sensors such as cameras, laser devices etc. There are many computer vision applications such as autonomous driving, face recognition, object segmentation and classification, object tracking etc [4]. Among them, moving object detection and tracking is one of the most significant ones. As it can be applied in various application areas such as traffic monitoring, people tracking and video surveillance.

Object detection and segmentation is one of the important application in computer vision, and its goal is to separate and label every object to various classes. Every object has its own shape and features. Thereby it is possible to use these special features to detect objects, which belongs to same class [9]. After object detection, it is possible also to track them in consecutive video frames. This method has its own advantages and disadvantages. It can be used for accurate and specific objects (such as pedestrians) in the scene. However, it's slow because it needs to firstly classify whole objects in the scene and then find the corresponding in adjacent frames.

Another approach is to track just moving objects in the frame sequences. In this method, which is faster, the purpose is to find moving objects over time by discovering their position in every frame of the video [13]. There are different tracking techniques such as temporal differencing, background subtraction and optical flow method etc [15]. For this reason, dense and accurate motion detection is one of the most significant ones, which might be done statically or using moving cameras.

Motion is a significant feature of frame sequences; it shows the dynamics of frames by relating spatial features to temporal changes in images. Optical flow estimation is the presentation of motion information from an image sequence; Optical flow is a 2D motion map, which shows the scene's 3D motion into the image plane. Note that optical flow module also could be used for extracting parallax in static stereoscopic vision if camera pose parameters are provided. Optical flow works on assumptions that pixel intensities do not change over time and neighboring pixels have similar motion [12].

Here we focus on the two well-known flow extractor methods, which are used widely in different computer vision applications and we try to know, which one has the best accuracy for object tracking application. In Section 2 we review, optical flow theory in more

details. Two selected flow extraction methods (Farneback and Lucas-Kanade) are presented and compared in subsection 2.2 and 2.3. We have made extensive experiments on large datasets, and moreover, optimal parameter settings are cross-validated against labelled datasets.

Optical flow methods are classified into global and local methods. In local methods we only need to process some pixels from the whole image, while global/dense methods process all pixels in image. For having more flow data in sparse/global flow extraction methods we proposed a novel method to estimate more object displacements using inverse flow. The results are presented and illustrated in Section 4. Note that in this thesis we just evaluate optical flow methods based on their accuracy, not timing nor subjective evaluation. As both mentioned models are quite fast and would be near real time using powerful machine equipped with GPUs, but the accuracy of these methods is still challenging.

2 THEORY

In this chapter, we review the concept and theory of optical flow techniques. There are lots of ways and methods represented for this purpose [11]. Among them we selected Farneback and Lukas-Kanade which are reviewed in subsection 2.2 and 2.3.

2.1 Optical flow

Optical flow is the distribution of apparent velocities of movement in captured imagery data. This apparent motions are extracted using comparing between two images, which might be considered between two images captured in two different times (temporal) or two images captured in exactly same time but using two cameras with known camera parameters (static optical flow). Generally, optical flow is a 2D vector where each vector is a displacement vector showing the movement of pixels from first frame to second in the perpendicular image axes [12]. In Figure1, the movement of pixels in two consecutive frames is shown, which are captured in different times t and $t + \Delta t$. The arrows shows the displacement of the pixels from first frame to next frame.

$$I(x, y, t) \qquad\qquad I(x + dx, y + dy, t + dt)$$
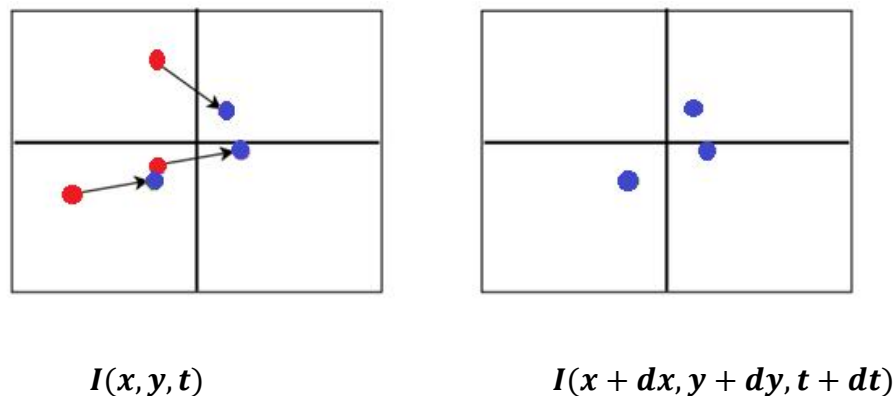
Figure 1. *Pixels displacement in two consecutive images. Red pixels and blue pixels are corresponding to Image at time* **t** *and* **t + dt** *respectivily*

Consider a pixel $I(x, y, t)$ in first frame, it moves to the next frame by taking dt time. The pixel displacement from first frame to next frame is (dx, dy) and since the image intensity and pixels are the same, we can say [12]:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{2.1}$$

To find pixels displacement we use differential method since they use partial derivatives with respect to the spatial and temporal coordinates. Assuming the displacement small, we take local Taylor series approximation of the image signal and we get:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \tag{2.2}$$
$$= I(x + y + t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t + c.$$

Where $I(x + y + t)$ is the assumed pixel at location (x, y, t) ,Δx, Δy, Δt are the movement between the two frames and c is a real valued constant number. From these equations, we get the flowing equation:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0, \tag{2.3}$$

which result in

$$\frac{\partial I}{\partial x}V_X + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0, \tag{2.4}$$

where $V_X = \frac{dx}{dt}$ and $V_y = \frac{dy}{dt}$ are the velocity or optical flow of $I(x + y + t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are image gradients at (x, y, t). The above equation V_X and V_y are unknown and cannot be solved with two unknown variables. So several differential methods are provided to solve this problem [11, 12]. Farneback and Lukas-Kanade are two of them, which we can classify into global and local methods, respectively [3].

In local/sparse methods, we only need to process some pixels from the whole image. In the other word, we only look for features (corners, ridges, edges and blobs). As we do not have per pixels flow data, if flow parameters have not been selected accurately, we might lose some small objects movements. For this reason, we proposed to use inverse flow in some special conditions to compensate this as much as possible.

While global/dense methods process all pixels in image [14], Local method are more efficient to compute robustly under the noise but dense techniques are more accurate in cost of more timing. The next subsections, we will review Lucas-Kanade and Farneback in more details. In this case, the high quality optical flow map should meet the following requirements:

1. *Smoothness:* flow field should be smooth over the interior of objects in the scene.
2. *Sharpness:* Flow should be *sharp* at object boundaries.
3. Flow map should be well *aligned with object boundaries*
4. *Stability:* It should be *temporally stable over frame sequences*. [2]

2.2 Farneback method

The Farneback method is a two-frame motion estimation algorithm that uses polynomial expansion, where a neighborhood of each image pixel is approximated by polynomial. Here we are only interested on quadratic polynomials, which give us the local signal model represented in a local coordinate system [5],

$$f(x) \sim x^T A x + b^T x + c_1, \tag{2.5}$$

In this case the vector x is a 1×2 vector containing the running variables x and y and A_1 is a symmetric 2×2 symmetric matrix of unknowns, which captures information about the even part of the signal, b_1 is about odd part of signal and a 2×1 vector of unknowns and c_1 is a unknown scalar. Writing it in terms of tensors gives equation 2.6, where the unknown coefficients is defined by the symbol r_i.

$$f(x) \sim [x \quad y] \begin{bmatrix} r_4 & \dfrac{r_6}{2} \\ \dfrac{r_6}{2} & r_5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + [r_1 \quad r_3] \begin{bmatrix} x \\ y \end{bmatrix} + r_1 \tag{2.6}$$

By computing the neighborhood polynomials on two subsequent images, one can obtain directly the displacement d in case of ideal translation. In Figure 2, the image $f_1(x)$ is taken at time t and $f_2(x)$ at time $(t + dt)$. Note that x in 2.1 is 1×2 vector but in Figure 2, to simplify we just show one dimension of x.
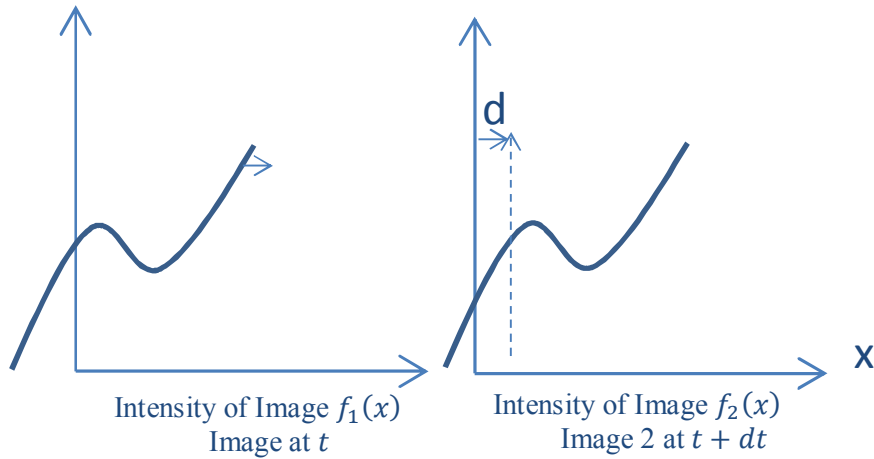
Figure 2. *Neighborhood polynomials on two subsequent images. d is the displacement.*

Therefore, if we can measure f_1 and f_2, may find $d = [d_1, d_2]$, which is the 2D displacement and hence the flow of the image pixel. In equations (2.7), we see how the polynomial

coefficients of the second frame are connected to the ones from the first frame and to the displacement d.

$$f_2(x) = f_1(x - d) = (x - d)^T A_1(x - d) + b_1^T(x - d) + c_1 \tag{2.7}$$

$$= x^T A_1 x + (b_1 - 2 A_1 d)^T x + d^T A_1 d - b_1^T d + c_1$$

$$= x^T A_2 x + b_2^T x + c_2$$

By assuming that the brightness is constant in two-sequence image, we can equating coefficients in the two polynomials.

$$A_2 = A_1, \tag{2.8}$$
$$b_2 = b_1 - 2A_1 d,$$
$$c_2 = d^T A_1 d - b_1^T d + c_1$$

By taking advantage of the equation for b_2 in (2.8), one can solve for d, as shown in equation (2.9).

$$d = -\frac{1}{2} A_1^{-1}(b_2 - b_1) \tag{2.9}$$

It is also stated in [5] that equation 2.7 is for the ideal case and it could be more realistic approximation when the $-\frac{1}{2}(b_2 - b_1)$ term in equation 2.9 be changed to Δb. In this case:

$$\Delta b = -\frac{1}{2}(b_2(x) - b_1(x)) \tag{2.10}$$

$$A(x) = \frac{A_1(x) + A_2(x)}{2} \tag{2.11}$$

This changes equation 2.9 into equation 2.12

$$A(x)d(x) = \Delta b(x) \tag{2.12}$$

In principle, equation (2.12) can be solved pointwise, but the results turn out to be too noisy. We assume all pixels in a small window I behave the same. If we can measure A_1, b_1 and c_1 from f_1 for each pixel in I and A_2, b_2 and c_2 from f_2 we can calculate d.

Thus, we try to find $d(x)$ satisfying (2.12) as well as possible over a neighborhood I of x, or more formally minimizing

$$\sum_{\Delta x \in I} w(\Delta x) \parallel A(x + \Delta x)d(x) - \Delta b(x + \Delta x)\parallel^2 \qquad (2.13)$$

Where we let w be a weight function for the points in the neighborhood. Where solving for $d(x; y)$ in equation 2.13 gives equation 2.14

$$d(x) = \left(\sum w\ A^T A\right)^{-1} \sum w\ A^T \Delta\ b \qquad (2.14)$$

2.3 Pyramidal Lucas-Kanade method

Lucas-Kanade-Tomasi (KLT) is a two-frame motion estimation that uses minimization to solve the basic optical flow equation. Algorithm is widely used and developed by Bruce D. Lucas and Takeo Kanade [12]. The nature of Lucas-Kanade method is to compute flow only for certain points (feature) and it tracks very small changes or motions across frames. However, sometimes we have big motion or high speed and if we want to catch them across frames we have to use large window and it may breaks the coherent motion assumption . To circumvent this problem we use Lucas-Kanade in a pyramid. Following Figure describes Pyramid for optical flow computation [2].
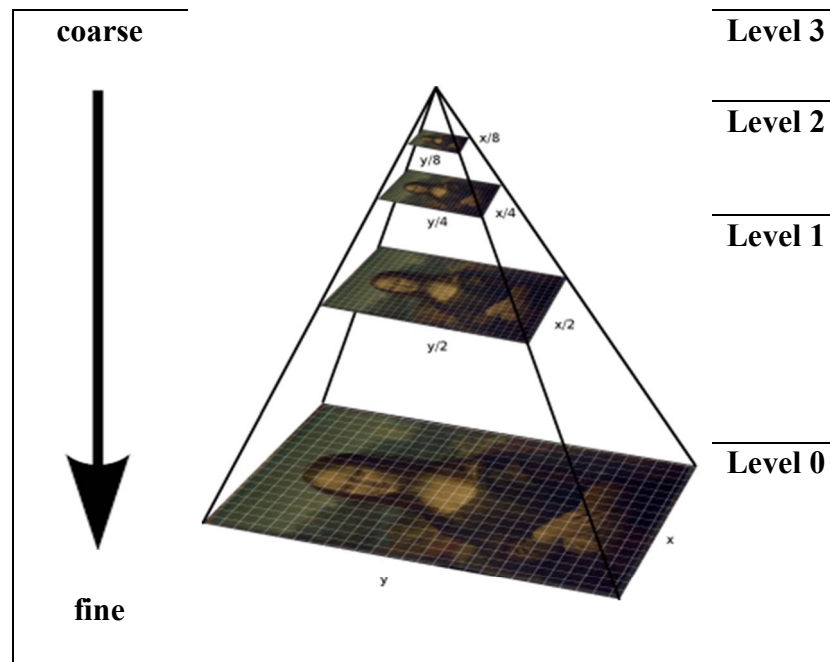
Figure 4. *Image pyramid with four levels. At each level, the image is downsized. Optical flow computation starts at the top of the pyramid (level 4) and ends at the bottom (level 0) [10].*

Figure 4 shows an image pyramid with four pyramid levels. In this case, the motion estimation starts at the top level of pyramid and it continue going down until we reach to the bottom level. Window size keeps the same over all levels. Thus, it is possible to detect large motions as they will be detected in the coarser levels of pyramid. In next subsection, we will discuss about the theory behind the Lukas-Kanade [2].

Theory of Lukas-Kanade

In the end of Section 2.1 we end up to an equation with two unknown variables (V_x, V_y) and solving an equation with two unknowns is not possible, therefore Lucas-Kanade use Spatial coherence assumption to solve this problem. Therefore, the method uses neighboring pixels to have more equations for example the 5-by-5 neighborhood around the current pixel give us 25 equations as follows.

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} V_x \\ V_y \end{bmatrix}}_{v} = - \underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_{b} \tag{2.15}$$

Where $I_x(p_i), I_y(p_i), I_t(p_i)$ are spatial derivative of the image I in position (x, y) and over time t, and $p_1, p_2, \dots, p_{25}$ are the pixels inside the window. In this case, A is a 25×2 matrix v is a local image flow 2×1 vector and b is a 25×1 vector. To simplify equation we write it in this form $Av = b$. This system contains more equations than unknowns and therefore it is an over-constrained system. To solve for this system least square minimization is found by multiplying the equation by A^T and, it solves the 2×2 system.

$$\underbrace{(A^T A)}_{2\times2} \underbrace{v}_{2\times1} = \underbrace{A^T b}_{2\times2} \tag{2.16}$$

Where A^T is the transpose matrix of A. from this relation we obtain equation (2.17), where the first matrix in equation 2.17 is an invertible matrix and the sums are from $i = 1$ to n.

$$\begin{bmatrix} \sum_i I_x(p_i)^2 & \sum_i I_x(p_i)I_y(p_i) \\ \sum_i I_y(p_i)I_x(p_i) & \sum_i I_x(p_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} -\sum_i I_x(p_i)I_t(p_i) \\ -\sum_i I_y(p_i)I_t(p_i) \end{bmatrix} \tag{2.17}$$

The solution to this equation is then:

$$(A^T A)^{-1} A^T b = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{2.18}$$

Equation 2.18 can be solved when $(A^T A)$ is invertible or $(A^T A)$'s eigenvalues (λ_1, λ_2) are large enough and have the same magnitude [17]

3 IMPELIMENTATION

The following chapter will describe the actual implementation process. It describes what kind of test materials, what methods and what environment were used in this thesis and it also includes comparisons of two algorithms Lucas-Kanade and Farneback. We compered the tracking accuracy of two-mentioned algorithm by isolating a moving object in video sequence. Different test video sequences were used and different parameters were examined for each algorithm to visualize, which algorithm and parameters are suitable for each video sequence. In next sections, before we go into the technical details the tools required and development process will be described.

3.1 Test data and ground-truth

The test data were downloaded from the Visual Tracker Benchmark website [6]. Tested data included video, which were constructed from consecutive JPEG-image files and the ground-truth file, which was formed from rows. In this case, ground truth-values isolate the location of moving object in each sequence. The location details for tracking object were given as a rectangle box, bounding the object. In ground-truth file each row represented x, y, box-width and box-height of the bounding box of the target in that frame. In used sequences, the first row belongs to the first frame and the last row to the last frame.
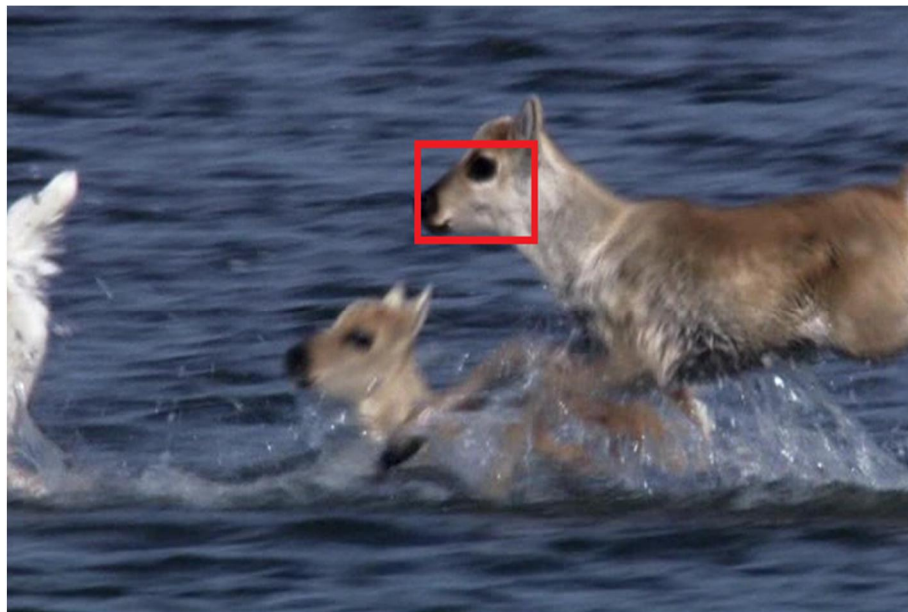
Figure 5. Bounding box around the target.

In Figure 5 we see red rectangle, which is the bounding box around the moving object and the location for the bounding box provided by ground-truth values. By restricting a region of an image, we compute the flow only for that area and we compare different algorithms accuracy only for that areas. Table 3 represents the bounding box tracker location and dimension used in our experiment. The x and y represent the corner of the tracker-bounding box and w and h represent the width and height of the bounding box respectively.

X	Y	W	H
306	5	95	65
313	15	98	70
318	40	95	66
326	74	96	58

Table 3. *An example of ground truth file. In above table, the values of ground-truth are presented only for four frames.*

To have a better quality measurements we selected two datasets, which are different in terms of temporal and spatial features. The first dataset includes 71 frames for outdoor scene objects, with high-speed object movements and the frame rate is higher. As a consequence some of the frames are blurred and should be harder to flow system to work perfectly. In the second dataset, the performance of flow estimator systems are measured in indoor environment, in the different lighting condition with slower object displacements.

Figure 6. *Left hand side image is an image example of outdoor sequence and right hand side image is indoor image sequence.*

3.2 Software implementation

The solution is experimented in Python (release 2.7) while this language delivers the performance needed and we used computer vision library OpenCV, which orders a good possibility to have easier access to image processing functions. In this thesis, we implemented both dense and sparse optical flow functions. In next subsection we will explain the processing step in more detail.

3.3 Farneback code

Now we will discuss about dense optical flow method, function `calcOpticalFlow-Farneback()` and its parameters. The function has many inputs we will briefly explain them here but also we have done some modification in code, which will be explained in experimental Section 4. The following code block describes the default parameters of OpenCV function `calcOpticalFlowFarneback()` implemented in python, which we used in this thesis :

```
cv2.calcOpticalFlowFarneback(
prev, next, pyr_scale, levels,winsize,it-
erations, poly_n, poly_sigma, flags)
```

Code Block 1: *Prototype for `calcOpticalFlowFarneback()` algorithm in OpenCV [10]*

Parameters of `cv2.calcOpticalFlowFarneback()`	
prev	Previous gray frame captured at t time.
next	Current gray frame captured in $(t + 1)$ time
Pyr_scale	Image pyramid or simple image scale. This parameter specify the image scale (<1) to build pyramids for each image.
Levels:	The number of pyramid layers. If level = 1 mean that flow is calculated only from previous image and no extra layers are created.
Winsize	Flow is computed over the window. If the value of window is larger the algorithm robustness to image noise increase, detection is faster but produce more blurred motion field.
Iterations	Number of iterations the algorithm does at each pyramid level.
Poly_n	Size of the pixel neighborhood used to find polynomial expansion in each pixel in other word kit is polynomial degree expansion and recommended values are 5 or 7.

Parameters of cv2.calcOpticalFlowFarneback()	
Poly_sigma	Standard deviation used to smooth derivatives used as a basis for polynomial expansion. Recommended values are from 1.1-1.5.
flags	Sets the input flow as an initial flow approximation or use a Gaussian averaging filter with size *winSize * winSize* instead of a box filter, which usually gives more accurate flow at the cost of speed.

Table 1. *function parameters of farneback for calculating optical flow by using OpenCV [10].*

3.4 Pyramidal Lucas-Kanade code

Now we will discuss about the pyramidal Lucas-Kanade method, `calcOpticalFlow-PyrLK()` and its parameters. The function has many inputs we will briefly explain them here but also we have done some modification in code, for example, this function uses '*good features to track*' function, which determines strong corners on image and when the algorithm finds good points, function is able to compute flow for those points.

In this case, it is Lucas-Kanade's parameter **prevPts,** which is Vector of 2D points for, which the flow needs to be found. In this experiment we did not use this function and instead of it we chose every 40 pixel in image in x and y direction.

```
cv2.calcOpticalFlowPyrLK(    prevImg,
next, prevPts, nextPts , status, err,
winSize, maxLevel, criteria, flags- )
```

Code Block 2: *Prototype for `calcOpticalFlowPyrLK()` algorithm in OpenCV [10].*

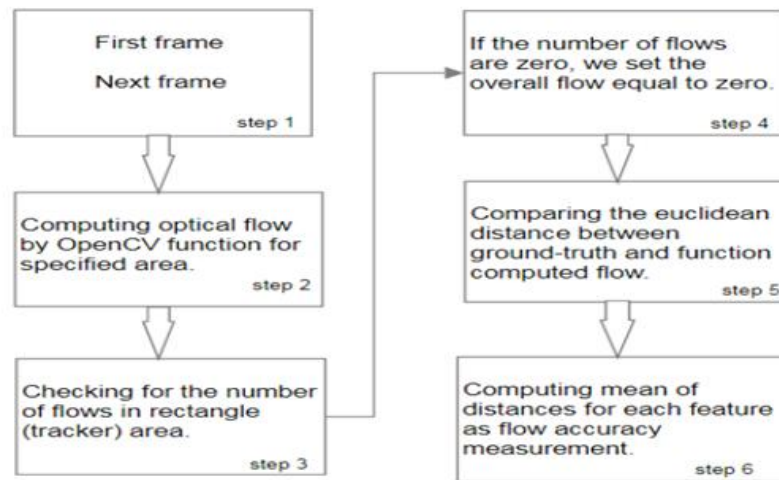| Parameters of cv2.calcOpticalFlowPyrLK ()	
prevImg	Previous gray frame captured at t time or pyramid constructed by cv2.cal-cOpticalFlowPyrLK.
next	Current gray frame captured in $(t + 1)$ time or pyramid, which is the same size and same type as prevImg.
prevPts	Vector of 2D points for, which the flow needs to be found; point coordinates must be single-precision floating-point numbers.

Parameters of `cv2.calcOpticalFlowPyrLK ()`	
nextPts:	output vector of 2D points (with single-precision floating-point coordinates) containing the calculated new positions of input features in the second frame; when OPTFLOW_USE_INITIAL_FLOW flag is passed, the vector must have the same size as in the input.
status	Output status vector (of unsigned chars); each element of the vector is set to 1 if the flow for the corresponding features has been found, otherwise, it is set to 0.
err	output vector of errors; each element of the vector is set to an error for the corresponding feature, type of the error measure can be set in flags parameter; if the flow wasn't found then the error is not defined (use the status parameter to find such cases).
winSize	Size of the search window in each pyramid level.
maxLevel	0-based maximal pyramid level number; if set to 0, pyramids are not used (single level), if set to 1, two levels are used, and so on; if pyramids are passed to input then algorithm will use as many levels as pyramids have but no more than maxLevel.
criteria	parameter, specifying the termination criteria of the iterative search algorithm (after the specified maximum number of iterations criteria.maxCount or when the search window moves by less than criteria.epsilon.
flags –	lags allow for some control of the routine's internal Bookkeeping; It may Be set to any or all (using bitwise OR) of the following.

Table 2. *function parameters of Lucas-kanade for calculating optical flow by using OpenCV[10].*

3.5 Processing step

In this section, we will explain the steps of experiment in more details. Figure 7 also helps to follow the steps better. The first step was to load first and next frames (step 1), then the OpenCV function computed flow for the first and next frame inside the bounding box (step 2). The main challenges we encountered was that in some frames or inside the bounding box the function was not able to find any good points to calculate flows, thus we got a none value for flow and the calculation of mean for each feature as flow accuracy was not possible. This problem happened when we used Lucas-Kanade and the reason is that this function is a sparse function, it means that the function does not compute flow for all pixels, but only for some areas where it find good points to track. To solve the problem we firstly checked if there is any frames or bounding box, where the algorithm cannot find any flow then we set the overall flow for that frames to zero (step 3 & 4).

First frame Next frame step 1		If the number of flows are zero, we set the overall flow equal to zero. step 4
Computing optical flow by OpenCV function for specified area. step 2		Comparing the euclidean distance between ground-truth and function computed flow. step 5
Checking for the number of flows in rectangle (tracker) area. step 3		Computing mean of distances for each feature as flow accuracy measurement. step 6

Figure 7. *Processing steps of implementation.*

The next step (step 5) was comparing the Euclidean distance between ground-truth and function computed flow. In last step (step 6), we computed the mean distances for each feature as flow accuracy measurement. The same process repeated for images until we got the last image.

Figure 8 represents first data set with modification. The target to track is inside the blue bounding box and red arrows illustrate the pixels displacement from previous frame to next frame. We are interested in flows, which are inside the bounding box and as we can see in Figure 8 in right hand side frame, we do not have any flows inside the bounding box. To solve the problem we calculate the inverse flow and used it for flow estimation. Because in case of perfect flow extraction, the flow and inverse flow components should be equal but with inverse sign. It means that we estimate the flow that goes from frame at time $t + 1$ to frame at time t. As can be seen in Table 6 the accuracy of the systems with inverse flow will be increased. Note that it's not good idea to always calculate the both flow, because of time consuming, and we just did it when we could not find any flow in our region of interest. If direct and inverse flow do not have any flow value in the interested area, so in this case we assumed the overall flow for tracker is equal to zero.
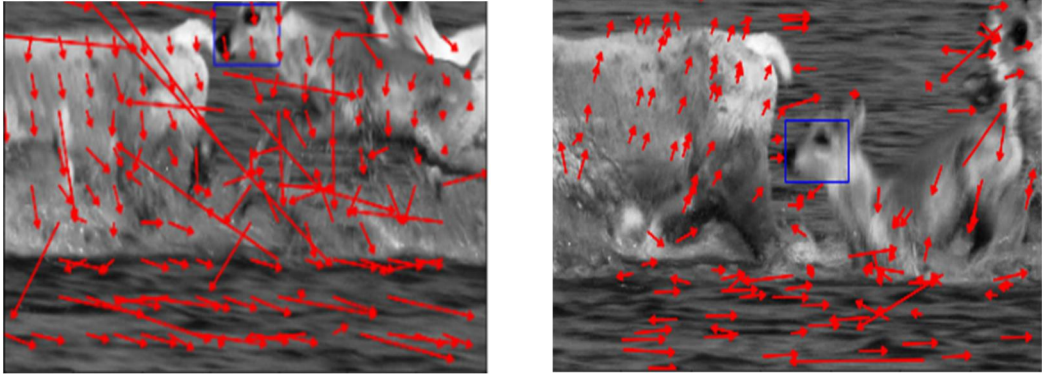
Figure 8. *Tracking object results using optical flow. The blue rectangle is the region of interest and red arrows are pixels displacements. Left image: the flow estimator method find some features in interested area. Right Image: Flow is not dense in interested area and we assume the displacement is zero*

4 EXPERIMENTAL RESULTS

In this chapter, we discuss about experimental results of the proposed object tracking algorithms. The accuracy of Lucas-Kanade and Farneback are represented for different parameter values and image sequences to have a better idea of algorithms accuracy when we change frame sequences or parameters of algorithm. Thus, reader have to keep in mind that these results are only for test data we used in this thesis and it might be changed if we use another dataset. In subsection 4.1 we will try different parameter values for Farneback and in subsection 4.2 we will repeat the same process for Lucas-Kanade algorithm. In the end of each subsection, we will represent the optimal values for each function.
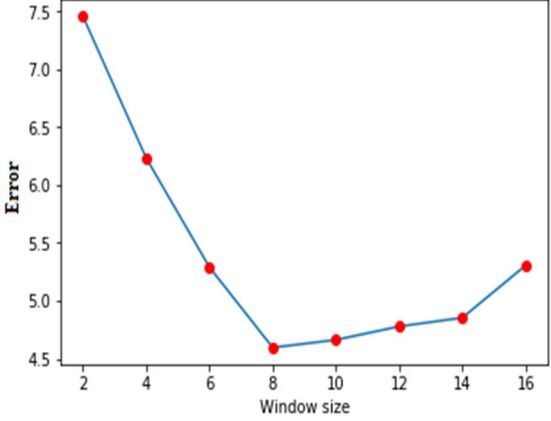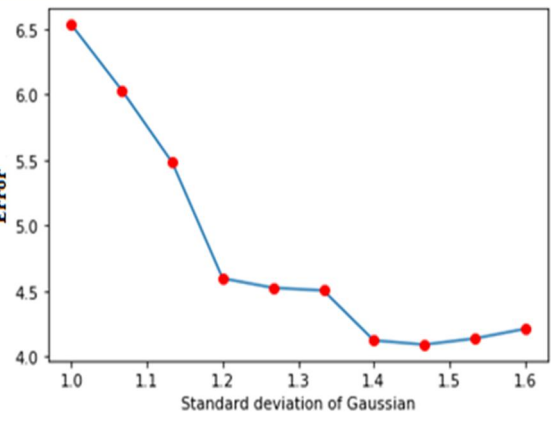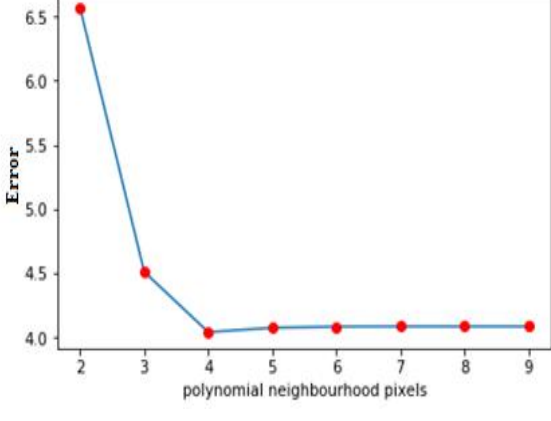
4.1.1 Optimal parameters for Farneback (First dataset)

Figure 9 shows part of first image sequences we used in this thesis with some modifications.

| Image 1 | Image 2 | Image 3 |
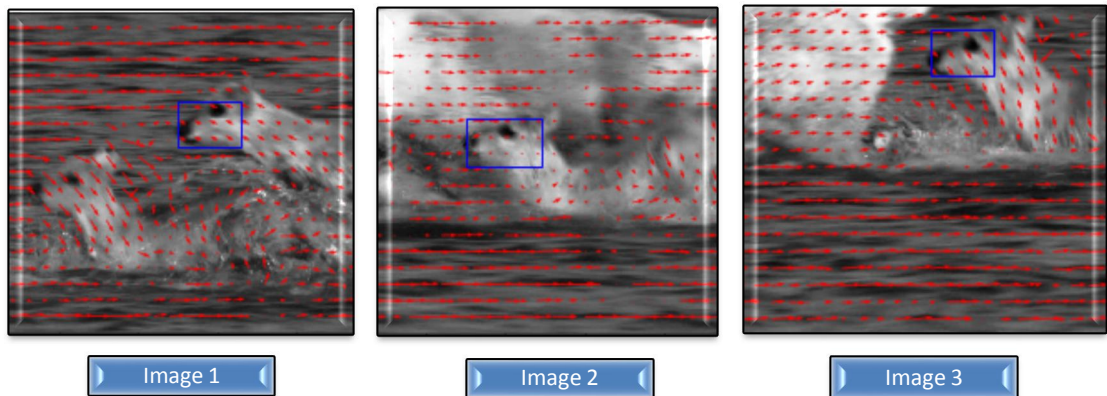
Figure 9. Part of deer images used in test sequence.

Next four values stayed the same over the implementation process pyr_scale = 0.5, levels = 3, iterations = 3 and flags = 0 and. Table 3 presents the different values we tried for each parameter and we see can see that first accuracy was equal to **4.596** and we improved it to **4.079** by choosing the best value for each parameter.

	Trying different values for window size: • Poly_n = 5 • Poly_sigma = 1.2 • Winsize = 2 - 16 • best **accuracy = 4.596** when window size is equal to 8
	Trying different values for standard deviation of Gaussian: • Poly_n = 5 • Window size = 8 • Poly_sigma = 1 - 1.6 • best **accuracy = 4.079** when poly_sigma = 1.46
	Trying different values for polynomial neighborhood pixels: • poly_sigma = 1.46 • Window size = 8 • Poly_n = 2 - 9 • best **accuracy = 4.079** when poly_n = 4

Table 3. *Left hand side shows Average distances between the transformed rectangle and ground truth and right hand side are values used for each parameters in Farneback first test data set.*

4.1.2 Optimal parameters for Farneback (Second dataset)

Figure 13 shows part of image sequences from second dataset. In this dataset the fps is lower compared to first dataset and the moving object is isolated well from background. As can be seen in Table 4, by changing the hard parameters the accuracy slowly improved from 2.345 to 2.341.
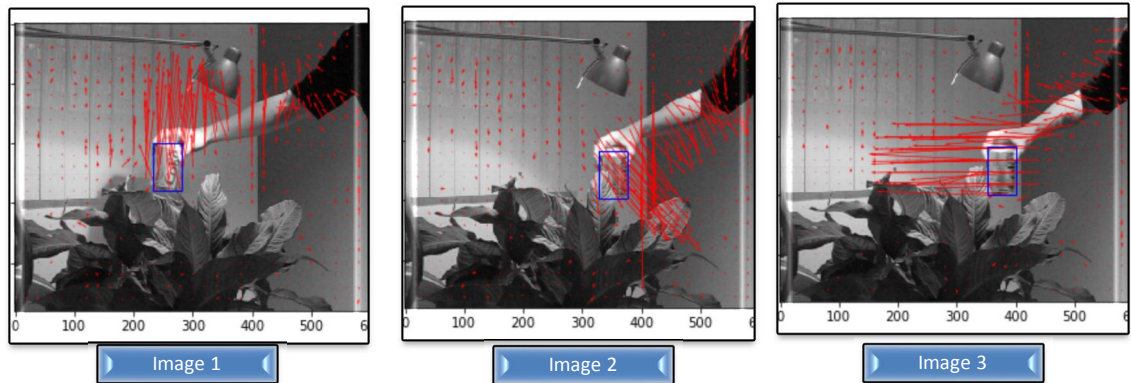
Figure 10. Part of Coke images used in Farneback test.

	Trying different values for window size: • Poly_n = 5 • Poly_sigma = 1.2 • Winsize = 2 - 26 • best **accuracy** = **2.345** when window size is equal to 24
	Trying different values for standard deviation of Gaussian: • Poly_n = 5 • Window size = 24 • Poly_sigma = 1 - 1.6 • best **accuracy** = **2.342** when poly_sigma = 1.133

Trying different values for polynomial neighborhood pixels:

- poly_sigma = 1.133
- Window size = 24
- Poly_n = 2 - 9
- best **accuracy = 2.341** when poly_n = 3

Table 4. *Left hand side shows Average distances between the transformed rectangle and ground truth and right hand side are values used for each parameters in Farneback first test data set.*
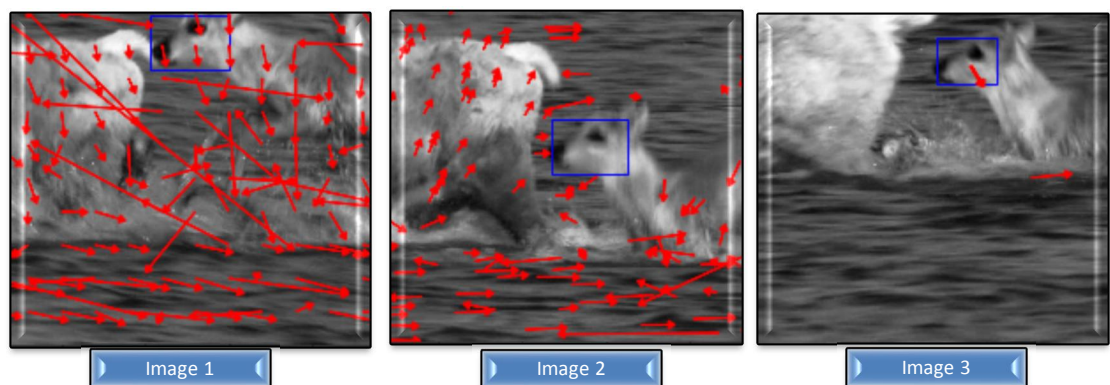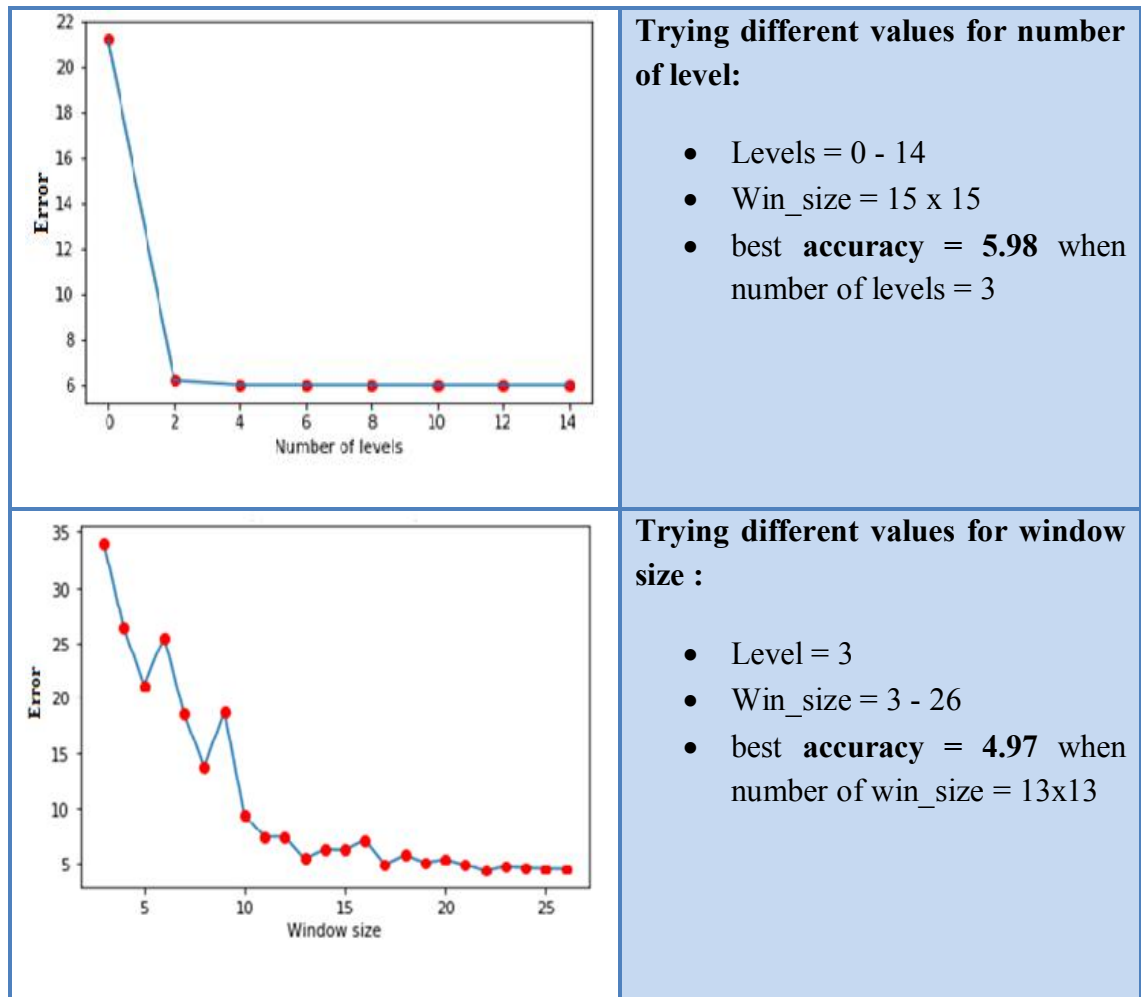
In this experiment, the accuracy did not change too much. One probable reason might be that, moving object is clear to follow and the speed of movement is slow.

4.2.1 Optimal parameters for Lucas-Kanade (First dataset)

In this subsection, we change some parameters of Lucas-Kanade function and determine which values reach the best accuracy. Here also the number of frames was 71 for each data set. The parameters we chose to test were number of levels (maxLevel) and window size (win_size) and other values stayed the same over the implementation process. Figure 11 shows first test data. In Lukas-Kanade algorithm, the resolution of image is very important. In some cases algorithm could not find any flow for image because the resolution of image was low. In table 5 we see that the accuracy improve from 5.98 to 4.97 when we try different values for chosen parameters.

Figure 11. *Part of deer images used Lucas-Kanede test.*

	Trying different values for number of level: • Levels = 0 - 14 • Win_size = 15 x 15 • best **accuracy = 5.98** when number of levels = 3
	Trying different values for window size : • Level = 3 • Win_size = 3 - 26 • best **accuracy = 4.97** when number of win_size = 13x13

Table 5. *Left hand side shows Average distances between the transformed rectangle and ground truth and right hand side are values used for each parameters in Lucas-Kanade first test data set.*

4.2.2 Optimal parameters for Lucas-Kanade (second dataset)

Figure 12 represents the results from second dataset with some modifications. As can be seen, the flow map is dense enough and the flow method could easily extract flow from the scene.
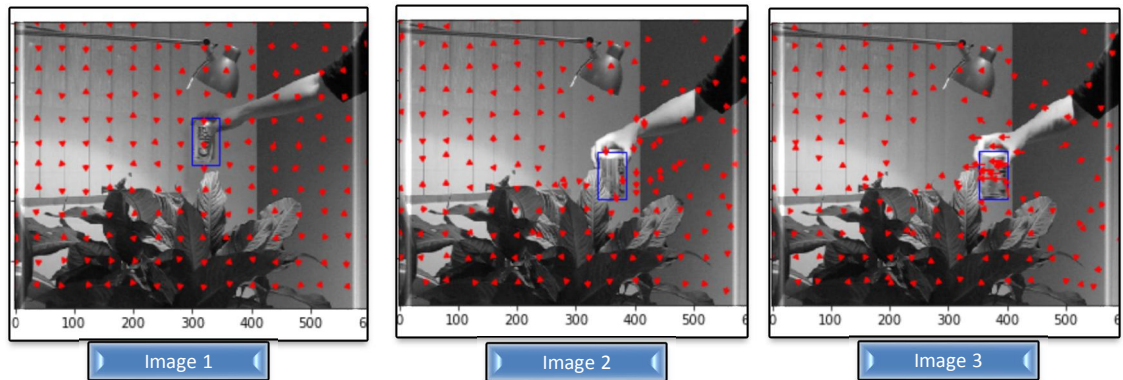
Figure 12. *Part of Coke images used in Lucas-Kanade test.*

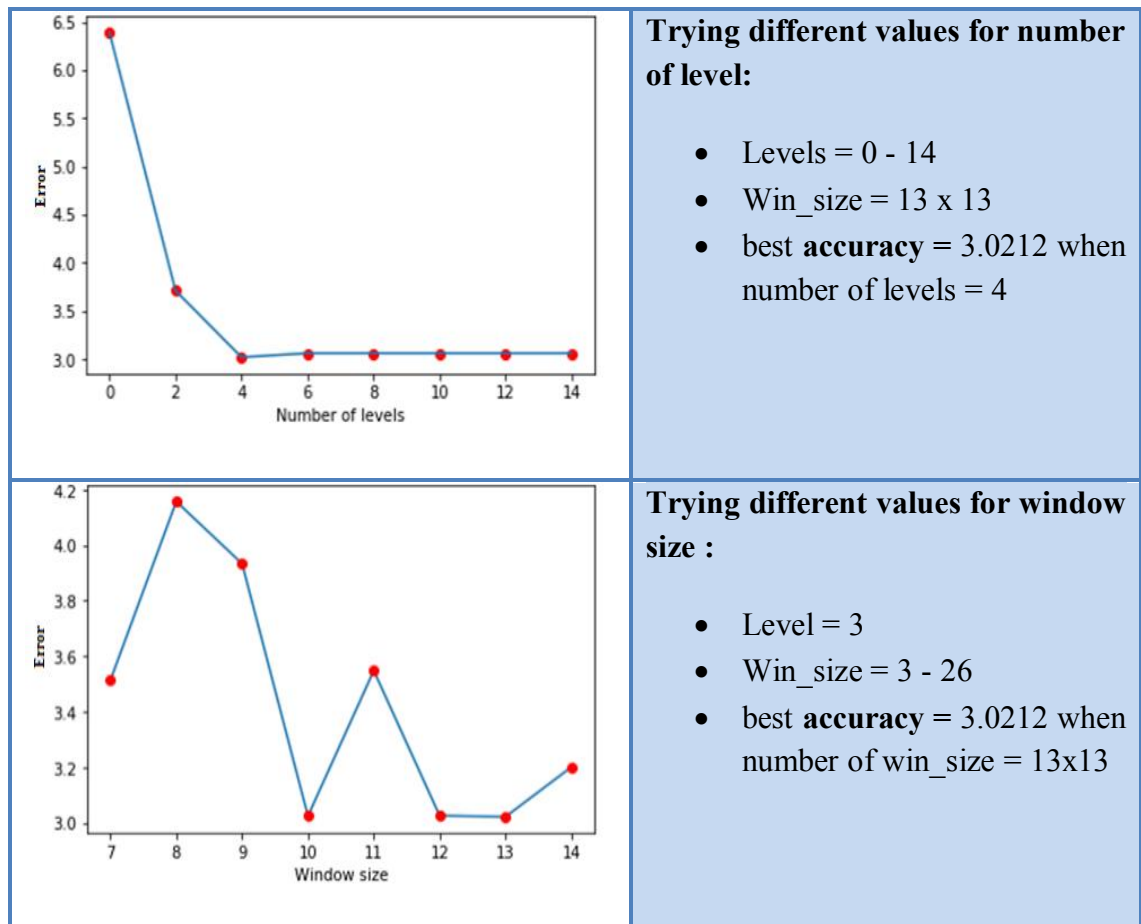|	
	Trying different values for number of level: • Levels = 0 - 14 • Win_size = 13 x 13 • best **accuracy** = 3.0212 when number of levels = 4
	Trying different values for window size : • Level = 3 • Win_size = 3 - 26 • best **accuracy** = 3.0212 when number of win_size = 13x13

Table 5. *Average distances between the transformed rectangle and ground truth. Best accuracy and parameters on the right hand side of the table.*

5 CONCLUTION

In this thesis, we evaluate two state of the art optical flow estimation methods for moving object tracking in static video sequence. We experimented the quality of flow extraction methods using several video clips including objects indoor and outdoor scenes. None of the methods described in this thesis are absolute and in fact, should be questioned and brought to more studies. The results showed that selection and cross validation of hard parameters for flow extraction methods are vital to get the best results.

The best performance of each flow systems are summarized in Table 6. Based on our results Farneback model is more accurate evaluated on both datasets. However, it suffers of inaccuracy in outdoor scene.

There are some reasons for inaccuracy of these two models:

1. To have a better quality measurement we selected two datasets, which are different in terms of temporal and spatial features. The first dataset includes the outdoor scene objects, with high-speed object movements and the frame rate is higher. Therefore, some of the frames are blurred and should be harder to flow system to work perfectly. In the second dataset, the performance of flow estimator systems are measured in indoor environment, in the different lighting condition with slower object displacements.
2. The Lucas-Kanade is sparse flow estimator, and do not estimate flow for whole image pairs pixels.
3. As both systems are tested on the same datasets, and the hard parameters are cross validated. One of the reasons reflects the difference between Farneback and Lucas-Kanade methods accuracy is most probably related to nature designing of their algorithm.

Mean distance measurements	Farneback	Lucas-Kanade	Lucas-Kanade + inverse flow
First dataset (outdoor)	4.07	4.97	4.84
Second dataset (indoor)	2.34	3.02	2.88

Table 6. *Quantitively evaluation of flow methods.*

In the Lucas-Kanade method evaluation the main challenges we encountered was that in some frames or inside the bounding box the function was not able to find any good points to calculate flows. We presented a novel object tracking system for sparse flow extraction methods by using inverse flow optimization. By using our proposed method, the tracking accuracy has been increase around 10 percent.

6 REFRENCES

[1] Beauchemin, Steven S., and John L. Barron. "The computation of optical flow." *ACM computing surveys (CSUR)* 27.3 (1995): 433-466.

[2] Bradski, Gary, and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[3] Bruhn, Andrés, Joachim Weickert, and Christoph Schnörr. "Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods." *International Journal of Computer Vision* 61.3 (2005): 211-231.

[4] Category:Applications of computer vision. (2017, July 18). Retrieved August 01, 2017, from https://en.wikipedia.org/wiki/Category:Applications_of_computer_vision

[5] Farnebäck, Gunnar. "Two-frame motion estimation based on polynomial expansion." *Image analysis* (2003): 363-370.

[6] Tracker Benchmark v1.0 - Visual Tracker Benchmark. (n.d.). Retrieved August 14, 2017, from https://sites.google.com/site/trackerbenchmark/benchmarks/v10

[7] Image Pyramids with Python and OpenCV. (2015, March 16). Retrieved August 08, 2017, from http://www.pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/

[8] Lucas, Bruce D., and Takeo Kanade. "An iterative image registration technique with an application to stereo vision." (1981): 674-679.

[9] Mikolajczyk, Krystian, Bastian Leibe, and Bernt Schiele. "Multiple object class detection with a generative model." *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. IEEE, 2006.

[10] Motion Analysis and Object Tracking. (n.d.). Retrieved April 01, 2017, from http://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html

[11] Nagel, Hans-Hellmut. "On the estimation of optical flow: Relations between different approaches and some new results." *Artificial intelligence* 33.3 (1987): 299-324.

[12] OpenCV: Optical Flow. (n.d.). Retrieved April 01, 2017, from http://docs.opencv.org/trunk/d7/d8b/tutorial_py_lucas_kanade.html

[13] Porikli, Fatih, and Alper Yilmaz. "Object detection and tracking." *Video Analytics for Business Intelligence* (2012): 3-41.

[14] Wulff, Jonas, and Michael J. Black. "Efficient sparse-to-dense optical flow estimation using a learned basis and layers." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.

[15] Yilmaz, Alper, Omar Javed, and Mubarak Shah. "Object tracking: A survey." *Acm computing surveys (CSUR)* 38.4 (2006): 13.