UNIVERSIDADE Ð
COIMBRA

Diogo Alfredo Filipe Correia

GPU-Aided Optical Flow Tracking Application for Arthroscopy Surgery

VOLUME 1

Dissertation within the scope of Integrated Master's in Electrical and Computer Engineering, specialization in Computers, oriented by Prof. Doctor Gabriel Falcão Paiva Fernandes and presented to the Faculty of Science and Technology of University of Coimbra, Department of Electrical and Computer Engineering.

September of 2022

FCTUC

UNIVERSITY OF COIMBRA
FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

GPU-Aided Optical Flow Tracking Application for Arthroscopy Surgery

Diogo Alfredo Filipe Correia

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering

Supervisor:
Prof. Doctor Gabriel Falcão Paiva Fernandes

Jury:
Prof. Doctor Jorge Manuel Moreira de Campos Pereira Batista
Prof. Doctor Luís Alberto da Silva Cruz
Prof. Doctor Gabriel Falcão Paiva Fernandes

Coimbra, September 2022

Agradecimentos

O trabalho apresentado neste documento não seria possível sem a ajuda de todos os que acompanham diariamente.

Em primeiro, gostaria de começar por agradecer ao meu orientador, professor Gabriel Falcão, por todo o apoio, ajuda e orientação.

A toda a minha família, pelo grande apoio ao longo destes últimos anos com especial agradecimento ao meu Pai, por tornar possível este percurso académico e por tudo o que me tem proporcionado ao longo da minha vida.

A todos os meus amigos de curso, colegas de casa e todos os que de alguma maneira se cruzaram comigo neste percurso académico e o tornaram especial, um enorme obrigado. Obrigado pela amizade, pela alegria e pelos momentos de entreajuda que ajudaram a amenizar todas fases mais difíceis neste percurso.

Por fim, agradeço também a esta cidade maravilhosa que me acolheu durante estes anos. Às amizades que aqui se fizerem e que ficam para a vida, obrigado Coimbra!

Este trabalho foi parcialmente financiado pelo Instituto de Telecomunicações e Fundação para a Ciência e a Tecnologia, Portugal, sob as bolsas UIDB/EEA/50008/2020 e EXPL/EEI-HAC/1511/2021.

A todos, muito obrigado!

Abstract

Computer Vision is not a recent area of investigation. Some algorithms date from decades ago and have been intensely scrutinized and analyzed, being present nowadays in various critical areas, for example, medicine, robotics, autonomous vehicles, aerial surveillance, etc.

One specific area that has been in development since long ago and is now regaining attention is object detection and tracking. Such algorithms have been studied for a long time, but have been limited by the existing hardware available, not allowing for reliable and usable implementations of these in real scenarios. With the recent improvement in hardware capabilities, visible in Moore´s law, new horizons have been opened and new possibilities once thought unachievable could now be attained. With these improvements also came new frameworks and APIs that allow the exploitation of more processing power, namely GPU parallel processing. As a consequence of this, the interest in object tracking has re-emerged, especially in medical and surgery areas where new robotic and augmented reality systems can take advantage of such algorithms.

Tracking algorithms are mathematically complex methods used to track features through a sequence of images, thus requiring advanced hardware to process them in real-time. Even though, these methods can be used as a possible and viable alternative to more recent methods, such as machine learning and deep learning, which do require a greater load of processing and are yet not viable, for example, in the case of higher image resolutions on real-time applications.

With this in mind, the objective of this dissertation is to explore the concept of object and feature tracking, as well as to directly compare different methods and algorithms to achieve real-time tracking. In the end, one implementation is also developed and described in detail using some of the most recent methods and APIs to approach this problem and create a faster alternative to the classical tracking methods. This dissertation also presents results obtained with both older and newer algorithms. Finally, a viewpoint of the obtained results is also presented, and options for optimizing the final solution are discussed.

Keywords

Kanade-lucas-Tomasi(KLT); Lucas-Kanade(LK); Sparse Optical Flow; Dense Optical Flow; Object tracking; GPU(Graphics Processing Unit); Parallel Computing; High Performance Computing; CPU(Central Processing Unit); OpenCV; CUDA; ArUco; Fiducial marker; Gradient; FPGA; Hessian Matrix;

Resumo

A Visão por Computador, não é uma área de investigação recente. Alguns algoritmos datam de décadas atrás e têm sido intensamente escrutinados e analisados, estando actualmente presentes em várias áreas críticas, por exemplo, medicina, robótica, veículos autónomos, vigilância aérea, etc.

Uma área específica que tem estado em desenvolvimento desde há muito tempo e que está agora a recuperar a atenção é a detecção e seguimento de objectos. Tais algoritmos foram estudados durante muito tempo, mas estiveram limitados pelo hardware existente, não permitindo implementações fiáveis e aplicáveis dos mesmos em cenários reais. Com a recente melhoria das capacidades de hardware, visível na lei de Moore, novos horizontes foram abertos e novas possibilidades, outrora consideradas inatingíveis, poderiam agora ser alcançadas. Com estas melhorias surgiram também novas estruturas e APIs que permitem a exploração de mais poder de processamento, nomeadamente o processamento paralelo em GPU. Como consequência disto, o interesse no seguimento de objectos voltou a surgir, especialmente em áreas médicas e cirúrgicas onde novos sistemas robóticos e de realidade aumentada podem tirar partido de tais algoritmos.

Os algoritmos de seguimento são métodos matematicamente complexos utilizados para rastrear características através de uma sequência de imagens, exigindo assim hardware avançado para as processar em tempo real. Ainda assim, estes métodos podem ser utilizados como uma alternativa possível e viável aos métodos mais recentes, como *Aprendizagem por Máquina* e a *Aprendizagem Profunda*, que requerem um maior processamento e ainda não são viáveis o suficiente, por exemplo, no caso de resoluções de imagem mais elevadas para aplicações em tempo real.

Com isto em mente, o objectivo desta dissertação é explorar o conceito de seguimento de objectos e características, bem como comparar directamente diferentes métodos e algoritmos para alcançar o seguimento em tempo real. No final, uma implementação é também desenvolvida e descrita em detalhe utilizando alguns dos métodos e APIs mais recentes para abordar este problema e criar uma alternativa mais rápida aos métodos clássicos de rastreio. Esta dissertação apresenta ainda resultados obtidos com algoritmos mais antigos e mais

recentes. Finalmente, é também apresentado um ponto de vista dos resultados obtidos, e são discutidas opções para optimizar a solução final.

Palavras Chave

Kanade-lucas-Tomasi(KLT); Lucas-Kanade(LK); Fluxo Ótico Esparso; Fluxo Ótico Denso ; Rastreamento de Ojetos; GPU(Unidade de Processamento Gráfico); Computação paralela; Computação de Alta Performance; CPU(Unidade Central de Processamento); OpenCV; CUDA; ArUco; Marcador Fiducial; Gradiente; FPGA; Matriz Hessiana;

"The beautiful thing about learning is nobody can take it away from you."

- B. B. King

List of acronyms

KLT	Kanade-Lucas-Tomasi
LK	Lucas-Kanade
CPU	Central Processing Unit
GPU	Graphics Processing Unit
NVOFA	NVIDIA Optical Flow Accelerator
FL	Federated Learning
ML	Machine Learning
FPGA	Field Programmable Gate Arrays
SDK	Software Development Kit
FCN	Fully Convolutional Network
RAM	Random Access Memory
ROI	Region Of Interest

Table of contents

List of acronyms **xi**

List of figures **xv**

List of tables **xvii**

1 Introduction and Motivation **1**

2 Background and State-of-the-Art **4**

2.1 Sparse Optical Flow 4

2.1.1 Kanade-Lucas-Tomasi Algorithm 4

2.1.2 Original Lucas-Kanade Algorithm 4

2.1.3 Inverse compositional algorithm 8

2.1.4 Gaussian Pyramid 11

2.1.5 GPU-based Implementations 12

2.1.6 FPGA-based Implementations 13

2.2 Dense Optical Flow algorithms 13

2.2.1 CPU and GPU Implementations 13

2.2.2 The NVIDIA Optical Flow SDK 14

2.3 Machine Learning and the KLT algorithm 15

2.4 Summary 15

3 Design of a Real-Time Optical Flow Based Tracker Application **16**

3.1 Methodology and frameworks 16

3.2 Hardware used in this Dissertation 17

3.3 KLT based tracker 17

3.3.1 CPU solution 18

3.3.2 GPU solution 18

3.4 Design of a Dense Optical Flow-based solution 19
3.4.1 Marker Detector 20
3.4.2 NVOFA based tracker algorithm 21
3.5 Summary 23

4 Experimental Results and Discussion 24
4.1 Detection Methods 24
4.1.1 KLT feature detection in CPU and GPU 25
4.1.2 Fiducial Marker Detector 26
4.2 Tracking step 27
4.2.1 KLT feature tracker in CPU and GPU 27
4.2.2 NVIDIA Optical Flow SDK based tracker 29
4.3 Comparison Between the KLT and NVOFA applications 32
4.4 Summary 34

5 Conclusion and Future Work 35

References 39

List of figures

1.1 Example of tracking performed with KLT. The images contain the detected features in a sequence of frames. 2
1.2 Tracking and integration of the various features present in each of the images of the frame sequence. (Image from: [1]) 3

2.1 Original Lucas-Kanade algorithm (also called forwards addictive method) (courtesy from: [2]) 7
2.2 Inverse Compositional Algorithm (courtesy from: [3]) 9
2.3 KLT with Gaussian Pyramid 11

3.1 Diagram illustrating the complete NVOFA-based tracking process. 20
3.2 Illustration of the algorithm calculations performed. 22
3.3 Illustration of the algorithm calculations performed. 23

4.1 Curves of time per frame taken, per number of features for each resolution in CPU selection. 25
4.2 Curves of time per frame taken, per number of features for each resolution in GPU selection. 26
4.3 Marker and the respective detection box. 27
4.4 Curves of time per frame taken per number of features for each resolution in CPU tracking step. 28
4.5 Curves of time per frame taken per number of features for each resolution in GPU tracking step. 29
4.6 Resulting vectors produced by the Optical Flow engine. 32
4.7 Time per number of features for each step of the algorithm in CPU and GPU. 33
4.8 Marker and the respective detection box. 33

5.1 Advantages and disadvantages of the application developed in the scope of this dissertation. 38

List of tables

4.1 Average results of the marker detection. 27
4.2 Average results of detection and tracking for 800 frames and a resolution of 1920x1080 with detection interval of 5 frames. 30
4.3 Average results of detection and tracking for 800 frames and a resolution of 1920x1080 with detection interval of 10 frames. 31
4.4 Average results of detection and tracking for 800 frames and a resolution of 1920x1080 with detection interval of 20 frames. 31

Chapter 1

Introduction and Motivation

Tracking objects in an image is a very common task nowadays. Medical [4–6], surveillance [7–12], 3D reconstruction [13, 14] and augmented reality [15] are computer vision areas where this feature can be applied, extending also to other emerging fields such as for example self-driving vehicles [16–20].

Most of these applications require that the object or marker to be tracked is primarily detected, through the usage of specific algorithms for that purpose, which is usually heavy in terms of processing workload [21].

In order to mitigate these drawbacks and thus improve the processing times, optical flow algorithms can be used as an addition a detection-only method, tracking the object in some of the frames. One of the best-known motion tracking algorithms is the Kanade-Lucas-Tomasi (KLT), a sparse optical flow algorithm that has been researched and implemented in the past two decades, showing some interesting feature tracking results [2]. This algorithm can detect and track a feature throughout a set of frames. An example of this tracking method in action can be seen in Fig. 1.1 where the different features of the same objects are detected in a sequence of frames. Then in Fig. 1.2 the final sequence of tracked features is compiled and virtual tracking could be obtained by connecting all the dots.

Fig. 1.1 Example of tracking performed with KLT. The images contain the detected features in a sequence of frames.

Although KLT is a powerful tool for tracking features in a sequence of images, it presents a major drawback. It is an algorithm that may require a high degree of computation depending on some variables such as the number of features to track, leading to a significant processing workload on the CPU (Central Processing Unit).

To address this issue, some approaches can be taken. The algorithm can be implemented using reprogrammable hardware such as FPGA boards which are severally more complex to program, or it can make use of GPU parallel computing power, using more common programming languages. One of the alternatives is to use the CUDA API developed by NVIDIA, allowing them to make full use of the available processing power of their graphical processing units.

A rather different approach to this tracking problem is the usage of a dense optical flow type algorithm. In normal conditions this would be an even more expensive approach in terms of computational cost than its sparse optical flow counterpart. However, the solution explored in this document takes on the use of a GPU (Graphics processing unit) parallel alternative offered by the NVIDIA Optical Flow SDK.

Throughout this dissertation, the concepts of optical flow and object tracking are discussed and a comparison between two of the existing methods is addressed. Implementations of each method are also explored as well as all the steps and methods applied to achieve a high frames-per-second rate application.

In a later phase, the implementation of a parallel dense optical flow solution is presented and compared with an existing KLT parallel solution to compare the advantages and disadvantages of each method and conclude which method suits better to be developed as a

real-life application. Finally, this solution is implemented using a real arthroscopic camera feed where fiducial markers take place as the object to be tracked.

Fig. 1.2 Tracking and integration of the various features present in each of the images of the frame sequence. (Image from: [1])

The structure of this dissertation and content of each chapter are the following:

- *Chapter 2* reviews relevant related work on optical flow algorithms, feature extraction and object tracking and the different existing methods;
- *Chapter 3* presents a step-by-step walk-through of a tracker implementation using the KLT method, both on CPU and GPU and suggests a design of an alternative approach to the KLT algorithm, using a dense optical flow method for the tracking of fiducial markers;
- In *Chapter 4*, the obtained results in the previous stages are presented and discussed;
- *Chapter 5* draws some conclusions, as well as some proposals for future work;

Chapter 2

Background and State-of-the-Art

This chapter intends to explain the origin and theory behind the original Lucas-Kanade method for image alignment, the main derivations that came from this method, and its use for feature tracking on an image. Subsequently, it is also described the Kanade-Lucas-Tomasi (KLT) algorithm, one of the algorithms that derived from Lucas-Kanade (LK) method, which makes use of feature extraction and selection mechanisms. Moreover an alternative dense optical flow method is also presented and described, which in opposition to the previous method, calculates the flow in every pixel or group of pixels between images of video or sequence.

This chapter also describes the state-of-the-art relative to the main implementations of the algorithms running on fast and efficient platforms.

2.1 Sparse Optical Flow

2.1.1 Kanade-Lucas-Tomasi Algorithm

In 1981, the original Lucas-Kanade method [22] for image alignment was proposed with the intent of solving the problem of optical flow and has since been applied to a wide range of applications [4–14, 16–20, 23].

2.1.2 Original Lucas-Kanade Algorithm

This algorithm presented in [24], essentially works by aligning a template image, say for example $J(x)$ with an input image $I(x)$, where x is a column vector that contains the pixel coordinates and $J(x)$ being an image or a patch of an image at t and $I(x)$ the image at $t+1$.

Then in [25], Shi and Tomasi present a solution for feature selection and a tracking algorithm based on the affine-photometric model, a more robust method when compared to a pure translational model, since the first one can accommodate rotations and changes in lighting. The criterion to detect favorable corners for tracking can be resumed as the two eigenvalues of a matrix Z, λ_1 and λ_2 that satisfy the following equation,

$$\min(\lambda_1, \lambda_2) > T \tag{2.1}$$

where T is a previously defined threshold and the matrix Z is defined as,

$$Z = \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix} \tag{2.2}$$

and is composed of g_x and g_y the image horizontal and vertical gradients, respectively.

Later in 2003 Baker and Matthews in [2] proposed a unifying framework for image alignment where the various existing algorithms and their extensions are analyzed and described with the main focus being on the inverse compositional algorithm, a more efficient version of the original Lucas and Kanade algorithm. This last one is described in more detail as the sum of square differences between the two previously referred images, $J(x)$ and $I(x)$, with J being the template image and I the image to which the warp is applied:

$$\sum_x [I(W(x;p)) - J(x)]^2 \tag{2.3}$$

The function $W(x;p)$ is called the warping function, with p being a vector of parameters, for example, the parameters of a pure translational transformation or a far more complex affine transformation, and x is a vector of the pixel coordinates. So the warping function is the function that transforms back the image I into the template coordinate frame.

By minimizing equation 2.3, what the algorithm does is move and possibly deform a template to minimize the difference between the template image $J(x)$ and the input image $I(x)$ bringing them as closely as possible until a certain predefined threshold is met, on which the image is considered to be coincident to the template on the previous frame, and so a feature can be tracked from a frame to the next. But, because in general the value of the pixels in $I(x)$ are non-linear in x, then the problem of minimizing this equation becomes a non-linear exercise [2]. So in order to solve this equation, it can be approximated as:

$$\sum_x [I(W(x;p+\Delta p)) - J(x)]^2 \tag{2.4}$$

solving by iterating for increments of Δp and updating the parameters with:

$$p \leftarrow p + \Delta p \tag{2.5}$$

This iterative process happens until the norm of Δp is bellow some threshold ε, such that $\|\Delta p\| < \varepsilon$.

The equation 2.4 can then be linearized with the use of a first order Taylor expansion on the function $I(W(x; p{+}\Delta p)), obtaining:$

$$\sum_x [I(W(x;p))\nabla I \frac{\partial W}{\partial p} \Delta p - J(x)]^2, \tag{2.6}$$

where ∇I is the gradient of the image I, and $\frac{\partial W}{\partial p}$ is the Jacobian of the warp $W(x;p)$.

After deriving equation 2.6 with respect to Δp, equaling the resulting expression to zero and solving it, we obtain the next formula for Δp:

$$\Delta p = H^{-1} \sum_x \left[\nabla I \frac{\partial W}{\partial p} \right]^T [J(x) - I(W(x;p))] \tag{2.7}$$

where **H** is called the Hessian matrix and it is by definition the Jacobian of the gradient of a function, which in this case is the warp function $W(x;p)$ and is written as:

$$H = \sum_x \left[\nabla I \frac{\partial W}{\partial p} \right]^T \left[\nabla I \frac{\partial W}{\partial p} \right] \tag{2.8}$$

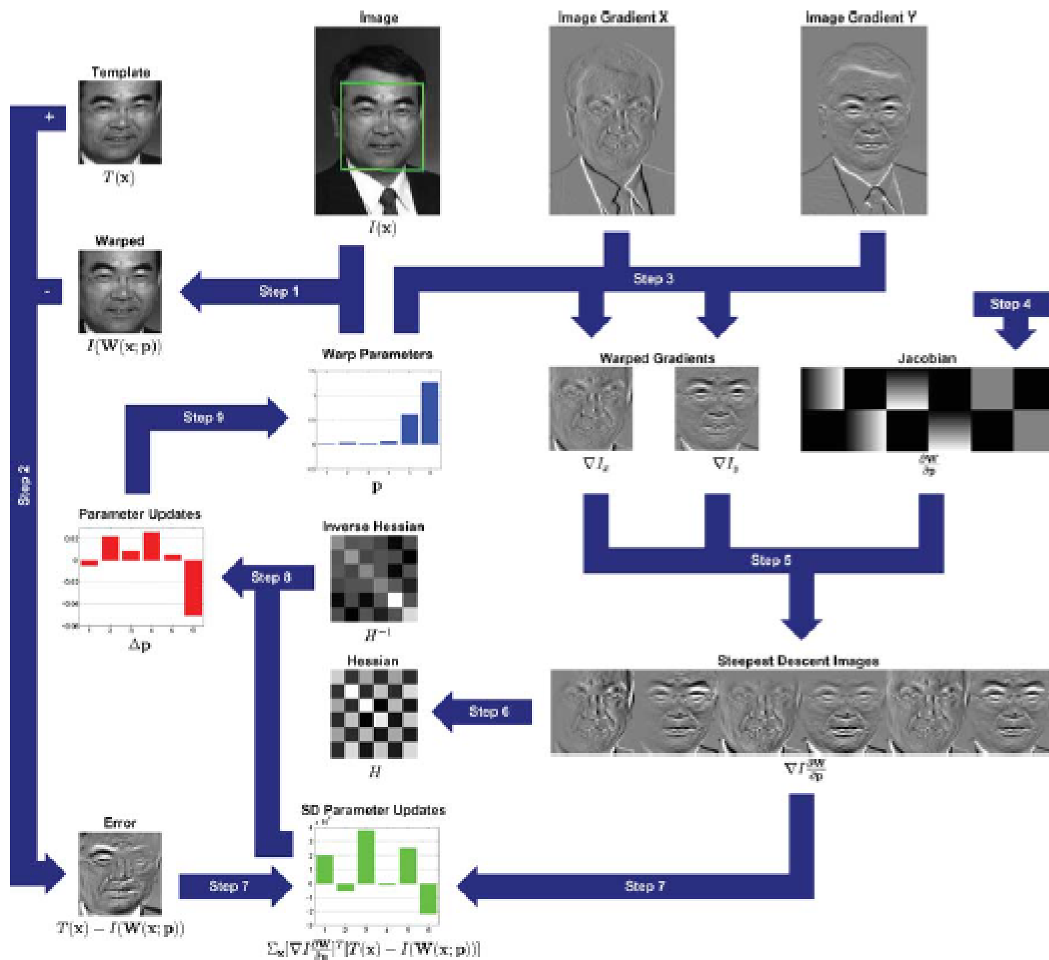

Fig. 2.1 Original Lucas-Kanade algorithm (also called forwards addictive method) (courtesy from: [2])

By applying the Lucas-Kanade algorithm, one is applying the equation 2.7 and updating the parameters with equation 2.5 iteratively, solving the problem of motion estimation, previously introduced. The algorithm can be described in Fig 2.1 and Algorithm 1. The total computation cost of this algorithm is $O(n^2N+n^3)$, where N is the number of pixels and n is the number of warp parameters.

Algorithm 1: Original Lucas-Kanade algorithm (Forwards Addictive)

Initialization: Obtain Template image J(x) from last frame and Input image I(x) from the current frame.

for: $\Delta p = 0 : \Delta p \leq \varepsilon$

Compute: $I(W(x;p))$;

Compute the error image with: $J(x) - I(W(x;p))$;

Compute the gradient of image I(x): ∇I;

Evaluate the Jacobian $\frac{\partial W}{\partial p}$ at (x;p);

Compute: $\nabla I \frac{\partial W}{\partial p}$;

Compute the Hessian matrix with: $H = \sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T \left[\nabla I \frac{\partial W}{\partial p}\right]$

Compute: $\sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T [J(x) - I(W(x;p))]$,

Compute Δp with: $\Delta p = H^{-1} \sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T [J(x) - I(W(x;p))]$

Update the parameters with: $p \leftarrow p + \Delta p$

2.1.3 Inverse compositional algorithm

One of the algorithms mentioned by Baker and Matthews in [2] is the inverse compositional algorithm. This method intends to solve a problem that is also mentioned in [2] and in [26] which is the high computational cost of calculating the Inverse of the Hessian matrix in every iteration of the algorithm.

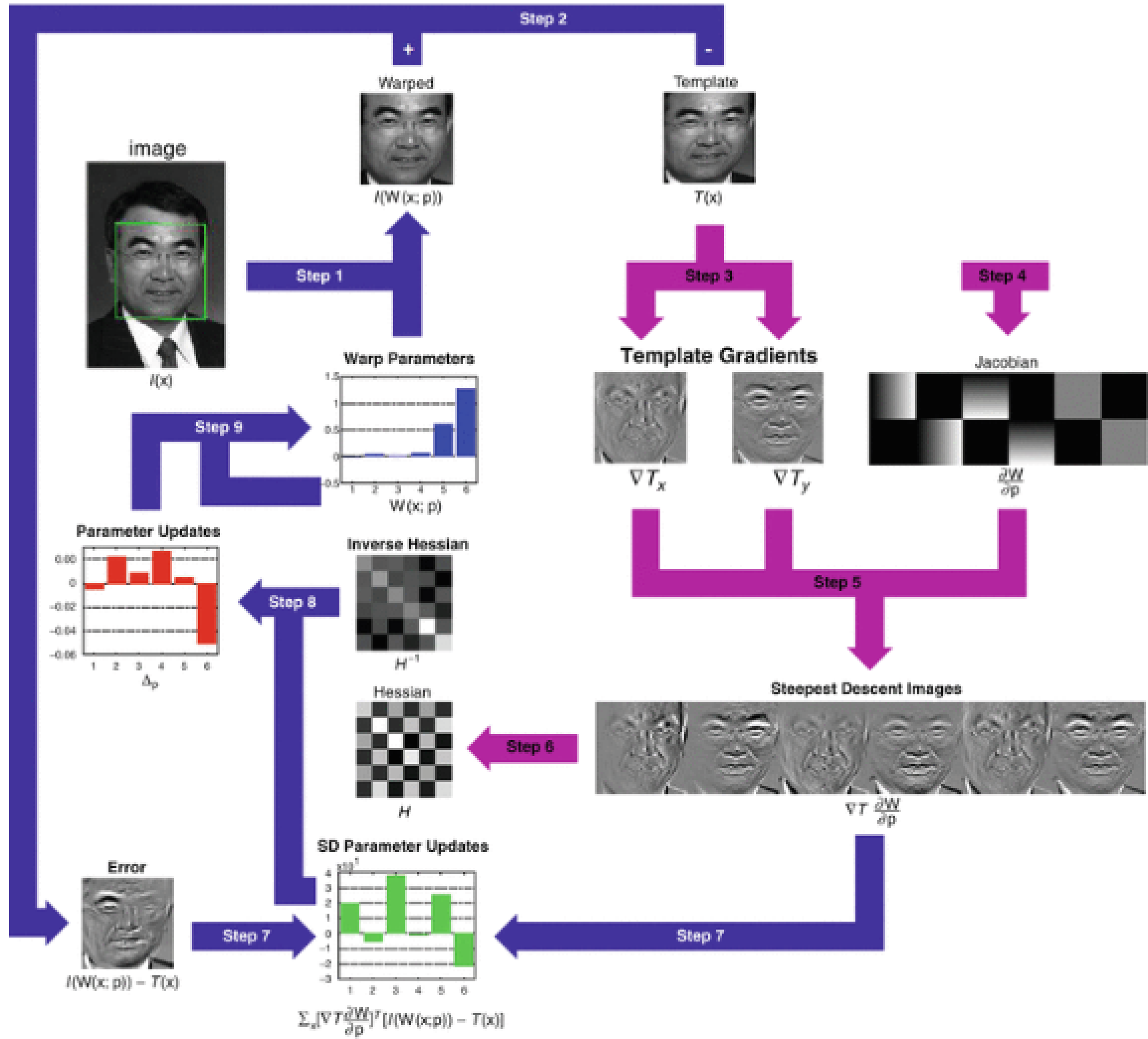

Fig. 2.2 Inverse Compositional Algorithm (courtesy from: [3])

This is solved by making a change of variables, inverting the roles of the template and the input image, concerning to what is done in the original Lucas-Kanade algorithm. This method solves the equation 2.3 minimization problem by updating W(x;p) which is the currently estimated warp, with $W(x;\Delta p)^{-}1$, an inverted incremental warp. Thus the original equation becomes:

$$\sum_x [I(W(x;p+\Delta p)) - J(x)]^2, \tag{2.9}$$

and the warp updating rule becomes:

$$W(x;p) \leftarrow W(x;p) \circ W(x;\Delta p)^{-1}, \tag{2.10}$$

which could be rewritten as the equivalent expression W(W(x; Δp);p), where the operator "$\circ$" is used to represent a function composition.

Following the same steps for deriving equation 2.9, just as was done on chapter 2.1.2, the next formula is obtained:

$$\Delta p = H^{-1} \sum_x \left[\nabla T \frac{\partial W}{\partial p} \right]^T \left[I(W(x;p)) - J(x) \right], \tag{2.11}$$

where **H** is again the Hessian matrix, but this time for the gradient of J and is written as:

$$H = \sum_x \left[\nabla J \frac{\partial W}{\partial p} \right]^T \left[\nabla J \frac{\partial W}{\partial p} \right] \tag{2.12}$$

Since the Jacobian is evaluated at (x;0), this makes the Hessian matrix independent of the warp parameters p, and so it becomes only necessary to compute this matrix once and no longer do it in every subsequent iterations, turning this algorithm a much more computationally efficient alternative to the original Lucas-Kanade algorithm. The total computational cost becomes $O(nN + n^3)$ per iteration plus $O(n^2 N)$ for the pre-computation of the Hessian matrix. The inverse compositional algorithm is described in fig 2.2 and Algorithm 2.

Algorithm 2: Inverse Compositional Algorithm

Initialization: Obtain Template image J from last frame and Input image I from the current frame.

Pre-computation:

Compute the gradient of image J(x): ∇T;

Evaluate the Jacobian $\frac{\partial W}{\partial p}$ at (x;0);

Compute: $\nabla J \frac{\partial W}{\partial p}$;

Compute the Hessian matrix with: $H = \sum_x \left[\nabla J \frac{\partial W}{\partial p} \right]^T \left[\nabla J \frac{\partial W}{\partial p} \right]$

for: $\Delta p = 0 : \Delta p \leq \varepsilon$

Compute: $I(W(x;p))$;

Compute the error image with: $I(W(x;p)) - J(x)$;

Compute: $\sum_x \left[\nabla J \frac{\partial W}{\partial p} \right]^T \left[I(W(x;p)) - J(x) \right]$,

Compute Δp with: $\Delta p = H^{-1} \sum_x \left[\nabla J \frac{\partial W}{\partial p} \right]^T \left[I(W(x;p)) - J(x) \right]$,

Update the warp with: $W(x;p) \leftarrow W(x;p) \circ W(x;\Delta p)^{-1}$

2.1.4 Gaussian Pyramid

The original KLT algorithm, by nature, presents limitations. One of them is that because it uses a linear approximation, it only works for small displacements between the Template and the Input image[27]. In **Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker** [28], the authors proposed a multi-resolution pyramidal method [29, 30] that can overcome the larger displacements between frames by calculating from a coarse to a fine level of displacement on the image features. For this, an image pyramid, like the one in image 2.3 is created in which every level of it has the same image with different resolutions, where level zero is the one containing the highest resolution image. Each level of the pyramid is computed by re-sampling the image at the previous level. For each level of the pyramid the KLT algorithm is applied starting at the highest level and up-sampling the displacement found, using it as the estimate for the next lower level of the pyramid. With this technique, it is possible to use the KLT algorithm with larger displacements per frame and still keep a high rate of successful features tracked. However, this solution brings a downside, which is a high computational cost, since the previous calculations of the algorithm need to be applied to every level of the pyramid.

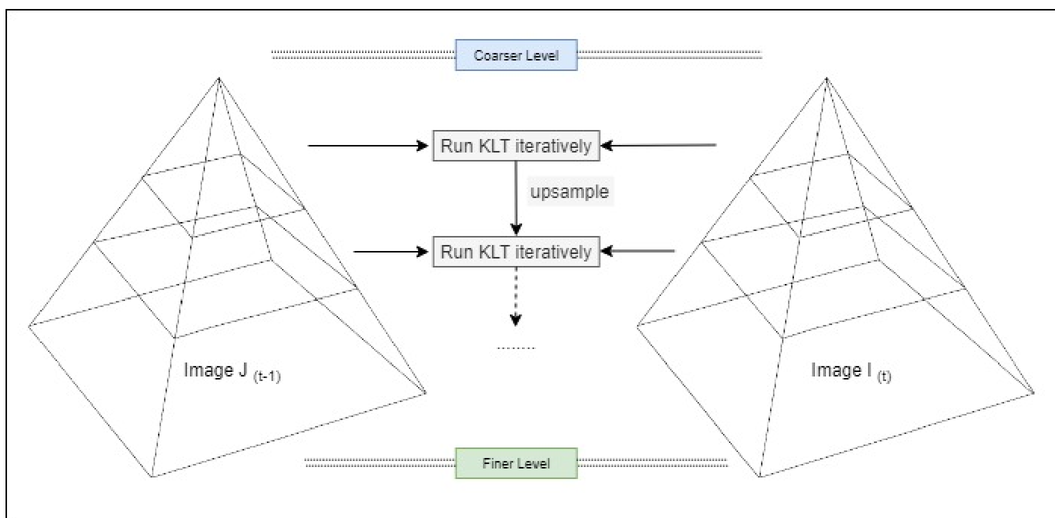

Fig. 2.3 KLT with Gaussian Pyramid

2.1.5 GPU-based Implementations

The KLT algorithm has been successful implemented on CPU throughout the past years [31]. In [32], Birchfield presents a famous open-source implementation of this algorithm in C language, which can achieve satisfying results on the number of selected and tracked features, implementing as well an option for reselecting features, proposed in [25]. As efficient as these CPU versions can be, they are always limited by the hardware. Depending on the window size of the tracking and the resolution of the images, this algorithm can be very computationally demanding, resulting in the impossibility of applying such an algorithm on a video streaming real-time application. Towards this end, GPGPU processing has been introduced to this problem and over the last few years, new works have been proposed. Hedborg et al. [33], presented a GPU-accelerated, real-time KLT implementation for a pure translational method, achieving an improvement on the scale of 10 to 15 times in speed relative to the previous CPU implementations. Sinha et al. [34] also proposed a GPU implementation using the translational model in OpenGL and testing the SIFT algorithm for feature extraction. In [35], Zach et al proposed another GPU OpenGL implementation of the algorithm, where they introduce a new parameter for gain adaptivity, making it less susceptible to illumination changes. It also analyzes the benefits of using OpenGL in comparison with CUDA. Later Ohmer and Redding [36], proposed a Monte-Carlo [37] based implementation following the same approach discussed by Sinha et al. in [34] and further discussed by Zack [35], where they try to mitigate the bottleneck generated by the selection of feature points using rapid feature selection with a monte-carlo method. In 2009, J.Kim et al. [26] introduced a new approach to the GPU-accelerated paradigm of KLT efficient implementations, using the CUDA framework. The used model was also changed from a pure translational to an affine one, called the affine-photometric model, which as mentioned before makes the algorithm less vulnerable to motions of the camera, such as rotations, translations, and a composition of these two. It also introduces two more parameters, allowing it do deal with lighting changes, making this a much more robust KLT application. This application was able to achieve an average of 50 to 60 frames-per-second (FPS) under any camera motions on images with a resolution of 640×480 and using the NVIDIA GeForce 8800 GTX graphics card. P. Mainali et al. [38] also developed a CUDA implementation of this algorithm in 2010, where the pyramidal approach is abandoned and the initial tracking position is predicted instead. With this new technique, it was achieved a speedup of around 25 times in comparison to the CPU version and about 236 times when compared to the KLT version that uses the image pyramid. These results were achieved for 960x960 resolution images, for 1000 features and using a GeForce 280 GTX GPU.

2.1.6 FPGA-based Implementations

GPUs can be often used to reduce the calculation times in applications with high computational requirements and with time constraints, as for example in the case of the KLT algorithm. However, GPUs show high power requirements and, sometimes, are not well optimized for the operations to be performed, and that is where FPGA-based accelerators can help. In [39], T. Sakayori and T. Ikenaga have done some work in this area, building a real-time version of the KLT algorithm on an FPGA board. This implementation was capable of obtaining an execution time of about 9ms running at 100MHz for a 640×480 image with 100 feature points. Later in [40], the algorithm implementation is improved by processing oversampling video sequences avoiding the use of an image pyramid procedure. However, the latency is still not low enough and real-time performance with a high frames-per-second rate is still not achievable. KLT own feature extraction method is more optimized in the context of use in the algorithm than Harris corner [41] or SIFT, but it still brings a lot of complexity, mainly on reprogrammable hardware platforms such as FPGAs. With this in mind, and aiming at a high frame rate and ultra-low latency, Hu et al. in [42] presented a solution based on the KLT but with some changes, such as the feature detection method which was replaced with Harris corner and a local search method is used instead, allowing to let go of the redundant hierarchy structure in the original KLT. This method achieved 0.762ms of latency tracking with a 640x480 resolution camera.

2.2 Dense Optical Flow algorithms

Object tracking has been around for at least two decades, and has been highly limited by the available hardware. For that reason, and with the recent evolution of parallel hardware such as GPUs, dense optical flow methods [43–46], have recently re-emerged to be of interest in object tracking and similar applications. Dense optical flow offers considerable advantages in comparison to other methods such as sparse algorithms since it does not require the detection of points of interest, relying on the motion of every pixel in the image.

With that in mind, the process of detecting objects becomes independent of the flow detection and allows for more robust and versatile tracking.

2.2.1 CPU and GPU Implementations

Amongst the many existing implementations of these optical flow algorithms, there are essentially two approaches that can be followed as it was mentioned in previous sections.

One is to use only the CPU to do all the processing, the other is to use a hybrid approach consisting of both the CPU and taking advantage of the parallel processing provided by the GPU. In [46], Besnerais and Champagnat pose a modified and much less costly method than the original dense version of the LK algorithm. The authors consider an alternative to mitigate the inconsistencies and erratic behavior of the algorithm and thus solve the Iterative-Warping Scheme divergence problems. T.Kroeger et al. [47], take a less complex approach in order to reduce processing time and at the same time maintain good accuracy and quality, proposing a method consisting of two components, and refinement in the end. Altogether, speed-ups of one to two orders of magnitude were achieved when compared to the state-of-the-art. Another worth mentioning document is [48] by N.Bauer et al. A combination between local and global or as it is also known, sparse and dense methods, is presented explaining the pros and cons of each. This combination of the Lucas-Kanade and Horn-Schunk algorithms aims to obtain a method that generates dense optical flow under image noisy conditions.

2.2.2 The NVIDIA Optical Flow SDK

NVIDIA has recently introduced a new set of tools dedicated to optical flow calculation. These SDK features are present in the most recent generations of GPUs, namely the Turing and Ampere, and consist of a hardware accelerator called NVOFA (NVIDIA Optical Flow Accelerator) which works independently of the GPU cores, usually called "Streaming processors" or "CUDA cores" in the case of NVIDIA products, as explained in [49]. This engine generates flow vectors between two given frames using sophisticated algorithms, which in turn can be used to track a specific object or marker previously detected. The API supports resolutions of up to **8192x8192** and a grid size of 1x1 meaning one motion vector per pixel creating a dense optical flow map. A trade-off between quality and performance is also exposed by the NVOFA API giving the user a possibility to choose from different options. Using this method on a real-time tracker is particularly advantageous for many reasons, such as leaving the GPU CUDA cores free to use at the same time as the vectors are calculated being fast, and helping to make the tracking process more accurate than other state-of-the-art algorithms. An example of an application that uses neural networks to detect objects and the vectors generated by the NVOFA API to assist with the tracking step is described in detail on [50].

2.3 Machine Learning and the KLT algorithm

In the last decade, and more specifically in the last few years, the field of machine learning has been highly explored, particularly in the computer vision area. Optical flow and motion tracking are not exceptions, and some work has been developed to build applications that can for example improve the precision of the KLT algorithm using neural networks. In [51] Hyochang Ahn and Han-Jin Cho, propose a system for multi-object detection that uses both Convolutional Neural Network (CNN) for recognizing objects and the KLT algorithm to keep the track of these detected objects. This makes the process much faster since the detected objects only need to be detected once and are then tracked with KLT and the algorithms for recognizing objects that use the existing CNNs are difficult to process in real-time. X.Liu et al. [52] also implemented a system that uses a Fully Convolutional Network (FCN) and the KLT algorithm, in order to count fruits on a tree. The network is trained to detect fruits and the KLT to track and count them in a stream of images. This kind of integration between feature detection using a neural network and tracking using KLT can in some cases, like the ones mentioned before, be beneficial and make the overall algorithm more robust and accurate.

2.4 Summary

During this chapter, it was presented the background and state-of-the-art consulted and used as a foundation for this dissertation. More concretely, a detailed mathematical explanation of the main existing optical flow trackers was presented, namely the KLT tracker, based on the optical flow principles studied in the works of Bruce D. Lucas and Takeo Kanade. Next some alternative methods were also presented with references to its implementations and to the results obtained. Among these, are CPU, GPU, FPGA and ML based approaches including some that will be used ahead in the development of this work.

Chapter 3

Design of a Real-Time Optical Flow Based Tracker Application

This chapter intends to describe an implementation of a real-time optical flow tracker in detail. It also reports the whole process and steps involved in order to achieve a final and working solution as well as the methodologies, frameworks and hardware used in the implemented algorithm.

The description steps, start by presenting an analysis on the already implemented LK based algorithm. Both a CPU and GPU versions achieved by J.Kim et al. in [26] already mentioned in chapter two, are described, including the necessary adaptions done in the course of this work in order to make possible the observation and analysis of the obtained results. Later, with the aim of meeting the goals of this dissertation, the development of a real-time optical flow based algorithm is also described in detail with all the steps involved in the process.

Following this, the first step of the chapter, intends to show the tools and frameworks used and studied in the initial phase of this work.

3.1 Methodology and frameworks

With the contributions of this dissertation in mind, the methodology mentioned below was followed:

- Become familiar with the frameworks and programming languages needed to develop computer vision solutions;

- Perform tests with an implementation of a solution based on KLT both in CPU and GPU to assess times and the consistency of the method and to create a means of comparison with a solution to be developed ahead.
- Perform a real implementation of a solution with NVIDIA NVOFA to achieve lower times and overall solid and accurate tracking results.
- Perform side-by-side comparisons of two different optical flow tracking algorithms and understand the advantages and disadvantages between the two;
- Further explore the method and optimize it for better and more accurate results.

Also during the first stages of this work, a study of the available tools was carried out on the main frameworks and APIs chosen.

3.2 Hardware used in this Dissertation

During this dissertation, the student´s machine was used for the development of the last solution and testing of both, KLT and NVOFA-based trackers. Therefore the work has been developed in a machine with average specifications and not a specifically assembled machine to achieve great calculation performance. This choice was due to the fact of a GPU containing the latest NVIDIA architecture was present in this machine, a necessary condition for the development of the NVOFA solution intended in this dissertation.

The machine specifications are:

- **RAM**: 8 GB;
- **GPU**: MSI GeForce RTX 3070 Ti GAMING X TRIO 8GB GDDR6X
- **CPU**: Intel 4770;
- **Storage**: 250 GB of SSD storage;
- **Motherboard**: Asus Z97-A;
- **PSU**: Corsair TX650 Bronze Certified 650 Watt High-Performance Power Supply.

3.3 KLT based tracker

The current dissertation development sits in one main objective, which is to find a solution for the tracking problem in a real-time application, more concretely, the tracking of a fiducial

marker type used in the ambit of the knee surgery to serve, for example, as a guide and help to the surgeon through an integration with an augmented reality system. Such a system could even be further used in the training and other applications of that kind for medical students, although that is beyond the scope of this dissertation. Following that, the first step of this work consists of the study and analysis of a method that could satisfy object tracking with good average fps, and accurate results. With that in mind, the KLT algorithm was chosen based on previous considerations presented before in earlier chapters.

3.3.1 CPU solution

In this section, the first approach to the KLT method is presented, which consists of a CPU version of the algorithm. The code used in this chapter was found as an open source, available online and is described by J.Kim et al. in [26]. This CPU method, also called the sequential method consists of applying the inverse compositional algorithm explained in chapter two. This implementation relies essentially in the use of *OpenCV* and *Intel IPP APIs* for the development. The first *API* is used to manipulate the image, such as to help with the extraction of frames from a video, and its display in the end, and provides methods and functions which can help in calculations, namely matrix-wise. *Intel IPP*, is also used to aid in specific calculations such as to apply filters and is used mainly in the calculation of the *Hessian* matrix and its inversion. However, the version of *OpenCV* used in this code was already relatively outdated. In the latest versions, some of these functions and methods became legacy and obsolete, making them impractical to use and even advised against by the developers. To be able to use the application, some changes had to be done. The main changes were relative to the image acquisition and display. Variable types were also changed between versions and so these were required to be updated in the original code too. Results of this implementation are presented and reflected ahead at chapter four in results and discussions, where it is noticeable a decrease in the performance of this algorithm with an increase in the image resolution. With the image data resolution being one of the main requirements for a final solution, other approaches had to be evaluated, including the parallel *CUDA* based method of KLT, to be introduced next.

3.3.2 GPU solution

This GPU version based on *CUDA* intends to mitigate the lack of performance noticed in the previous CPU algorithm. The base of this implementation is built with the same source code as the previous one, being the only difference, the parallelization of some parts

of it, more specifically in the tracking step. In this step, the usage of CUDA allows for a distribution of the workload of the calculations involved in tracking various detected features. Such is accomplished with the use of various techniques like coalesced memory access. Also, the usage of every memory present in the GPU is exploited, like global memory, texture memory, and shared memory. This last one is particularly important since it allows for faster parallel processing since memory is shared between threads of the same block. This type of memory is much faster than global memory and so it can make a significant difference in performance. As observed in chapter four in results and discussion, the performance has improved significantly in the GPU approach, which therefore means that even after taking into account the data transfer times taken between host and device i.e. between CPU and GPU in the case of a hybrid approach such as is the one described in his chapter the time reduction is clear. Nevertheless, the average FPS is still below the minimums required for a real-time application, for at least the *1920x1080* resolution required. This is mainly because the process with the higher processing cost i.e. the Inverse Hessian matrix calculation benefits more from a sequential process over a parallel or hybrid one since the CPU cores are much more power full than each core in a GPU processing unit. With that in mind, it was clear that a new and different approach was needed to fulfill all the requirements agreed on at the beginning of this thesis. The next chapter presents a solution conceived from ground zero and designed using some of the newest optical flow algorithms and tools existing.

3.4 Design of a Dense Optical Flow-based solution

Following the experiments made with the KLT algorithm, demonstrated in the previous chapter, a new alternative method for optical flow-based tracking was proposed. In this chapter the process is explained in the detail, as well as every tool and framework used during its development. The method can be divided into two main parts, detection, and tracking. Fig 3.1, resumes the sequence of events that happen during the execution.

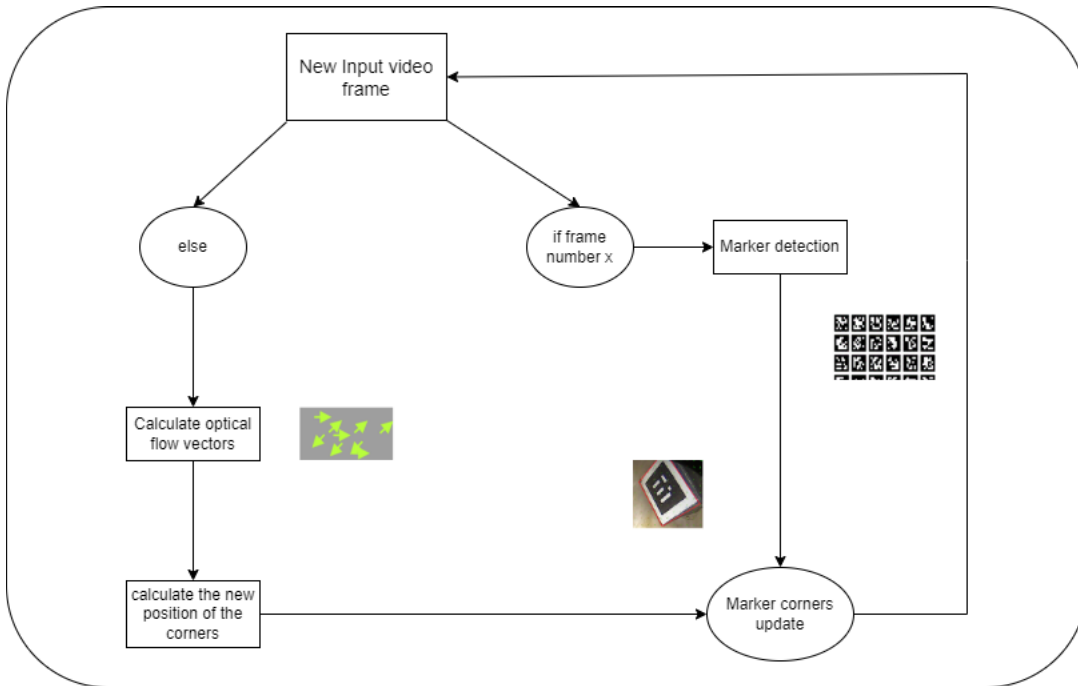

Fig. 3.1 Diagram illustrating the complete NVOFA-based tracking process.

The process starts with the image acquisition from a set of frames in a video sequence. Next, depending on the value chosen for the frame interval of detection, either the marker detector is called or the tracking algorithm. In the first case, the fiducial marker detector is called until the marker is detected and subsequently each corners new position is updated. In the second case, the tracking algorithm is called, and so the optical flow vectors are calculated. Based on these vectors, new positions for the marker are calculated and a box is drawn between those corners. Next, the detailed description of these two steps is described.

3.4.1 Marker Detector

Marker tracking can be achieved essentially in one of two ways. Either by detecting the marker in each frame or by detecting once in an interval of frames and tracking in the rest of the frames using a tracker algorithm. Detecting in every frame can be a very demanding task for most machines, especially if done at a high rate of FPS since it can be too heavy in terms of processing. Therefore the best option is to detect and track, and it is this method that is described in the current chapter.

A fiducial marker, more commonly called *ArUco* marker consists of a binary encoded image that can have different sizes, numbers of bits, and margin sizes. These markers are generated using specific libraries and in the case of this work, the *Alvar* library is used. These

libraries are called dictionaries, and consist of a matrix of values normalized in the interval of **0** to **255** which codifies a set of markers and can be used to decode already generated markers. The detector used is part of the *OpenCV* library and has proven to be a highly reliable and efficient method of marker detection. To detect the markers present in the footage data provided, a custom dictionary was generated with the matrix data and size provided. When performing the detection, the position of each corner and the id of the tracker are passed by the SDK to the algorithm that performs the marker position update, which will subsequently track the marker.

3.4.2 NVOFA based tracker algorithm

NVIDIA has recently released an SDK named **NVIDIA Optical Flow SDK** to take advantage of an engine designed exclusively to produce motion vectors. This resource is only available on the newer NVIDIA chipset architectures namely *Ampere* and *Turing* present on the **RTX 3000 and 2000** series respectively. This makes the SDK the perfect candidate to be used in the final solution as far as object tracking is concerned. Following this, the development started with the study and exploration of this optical flow method. A set of parameters can be chosen, these are, "**slow**", "**medium**" and "**fast**" for quality and "**1x1**", **2x2** and **4x4** for granularity settings. The first one is relative to the quality of the produced vectors by the engine ie the accuracy of the value, "**low**" being the lowest quality and accuracy possible and "**high**" the highest. The second setting is relative to the number of vectors generated per frame. This value is created as **gridsize** and the three existing settings are relative to the number of pixels per vector. The higher this value, the lower the vector count is and so the lower the time per frame becomes. Results containing all setting combinations can be seen in chapter four, on results and discussion. Another interesting feature offered by this API, is the possibility of generating a ROI (Region of Interest) on which the vectors are going to be calculated. This means everything around that area is ignored, bringing two positive effects. First, and this varies from case to case, the number of vectors to be calculated can be reduced by a great margin, specifically in the medical case in study, since an arthroscopic camera only produces a circular useful area in the middle of the image plane as seen in 4.3, thus decreasing the load and memory needed on the GPUs optical flow engine side. This will finally result in a performance increase of the final application. The field relative to the ROI can be seen in figure 3.2 in the initialization function. The ROI can be provided in two ways, a text file containing the pixel coordinates of the four corners of a polygon or using the function "**Rect**" in the code itself with these coordinates as inputs. Moving on to the

implementation phase, the first step in tracking, assuming marker corner coordinates have already been acquired, is to obtain the vectors representing displacement. Such vectors are provided by methods that can be found in the API with its initialization being represented in figure 3.2.

```
// optical flow initializer
Ptr<NvidiaOpticalFlow_2_0> nvof = NvidiaOpticalFlow_2_0::create(
    frameL.size(), roiData, perfPreset, outBufGridSize, hintBufGridSize,
    enableTemporalHints, enableExternalHints, enableCostBuffer, gpuId);
```

Fig. 3.2 Illustration of the algorithm calculations performed.

The vectors taken into account are present in a virtual circle with the center in the center point of the four corners and the radius the distance between the center and one of the marker edges. An average operation is iteratively performed between the obtained vectors, with the result converging to a single vector representative of the total movement of the marker in the image. This process is shown in algorithm 3.

Algorithm 3: Pseudo-Code of the tracking algorithm

for *i in* ***frame_rows/gridsize*** **do**
  for *j in* ***frame_cols/gridsize*** **do**
    if distance < radius Get **vector(i,j)**
    Calculate the average between the last vector values and the current one both in rows and columns;
Update corners with: **new_corners=old_corners + average_vector**;

This single vector is represented in blue in figure 3.3 and is used to calculate the position of the new corners from the previous set of points originated in the previous iteration. This is a simple and effective way to create an object-tracking algorithm with good performance. Although to make this process more reliable and increase the tracking accuracy an affine model should be implemented as well to accommodate rotations and changes in perspective. One example of an interesting method that could be applied in this case would be the one presented by L.Li et al. on [53]. With the corner positions and also applying a method similar to the one explained before and illustrated in 3.3 but this time in each corner, it is possible to apply an efficient four-parameter affine motion estimator such as the one mentioned in the earlier citation and therefore make the algorithm much more reliable and accurate with a minimal computational cost.

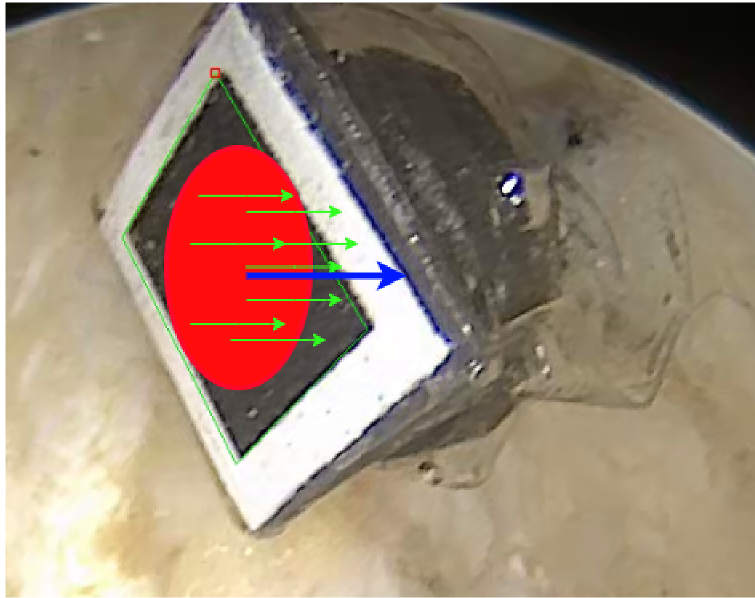

Fig. 3.3 Illustration of the algorithm calculations performed.

3.5 Summary

During this chapter, were presented two alternative methods for optical flow motion estimation. One is based on the LK algorithm, the KLT tracking algorithm, and another is based on a new and innovative method that makes use of one of NVIDIAs most recent technologies present on modern GPUs and implemented from scratch as part of this dissertation work. In addition, detailed information about the implementation of each of these algorithms is given. Finally, some thoughts are shared regarding other state-of-the-art methods that could be employed to increase the accuracy of the developed algorithm.

Chapter 4

Experimental Results and Discussion

From the previous chapters, it is possible to have an insight into the path taken to develop the dissertation. The research carried out *a priori* made it possible to have a clear idea of the vantages and disadvantages of the main state-of-the-art methods and make a comparison between two, in order to choose which fits best as a solution for the given tracking problem.

Although the LK based tracker presented in the last chapter detects points of interest in the image and tracks them as an altogether algorithm, in the case of the NVIDIA optical flow SDK based solution a separate detector is needed before the tracking algorithm is applied. In the particular case of this dissertation an ArUco marker was used in the images provided, and therefore, the first step was to find out how efficient are the already existing detectors for this type of marker. This leads to the first section of this chapter, where the KLT detector, both in CPU and in its parallel GPU version and the Fiducial Marker detector are tested and analyzed.

4.1 Detection Methods

The first step of this dissertation was to find a suitable detector to be used together with the tracker in a final developed solution. Therefore, the KLT feature detector, also known as *good features to track* [25] algorithm and a fiducial marker detector were tested and analysed in different phases of this dissertation development, in order to have a mean of comparison between these two solutions and to test the feasibility of each of them in the real life scenario present in the first chapters of this document and thus accomplishing the objectives proposed on it.

4.1.1 KLT feature detection in CPU and GPU

As stated before, in previous chapters, the KLT is one of the most researched and used tracking algorithms worldwide in various applications. It relies heavily on a powerful feature detector called *good features to track* [25], as it was also mentioned and explained in more detail prior to this chapter. As such the logical first step would be to do some testing with the footage provided to be used in the scope of these dissertation in order to compare the obtained results with the state-of-the-art already documented. Results were then collected using two different versions of the same algorithm. At first, a CPU version was tested with four different resolutions and an incremental number of detected features, ranging from 0 to 1000, as shown in Figure 4.1.

Fig. 4.1 Curves of time per frame taken, per number of features for each resolution in CPU selection.

As it can be observed, the time per frame suffers a substantial loss with the increase in the image resolution and number of features. This is mainly due to the fact that the CPU processes pixel by pixel and feature by feature while it goes through the whole image. This means a *1920x1080* will take a lot longer to be processed. In the specific case of the *320x240* image, it is noticeable a time consistency with the increase in the features number. Such can be justified with the image size, since the number of pixels is so poor that the processor cache is enough to keep the whole picture in memory reducing to barely none the existing latency in the transfers between the different levels in the memory hierarchy.

Afterwards and in order to mitigate this effects, a *CUDA* version was tested. This version takes advantage of the great parallel processing power present in modern GPUs and allows

for more throughput, which should help to at least decrease the gradient seen before as the feature number increases. Figure 4.2, illustrates this behavior and shows a significant improvement in the overall time taken for every resolution as expected. Nevertheless, the time difference between each resolution curve is preserved, from the sequential case to the present one, i.e. between CPU and GPU, which indicates that even the GPU can not mitigate the processing times taken when the image presents a high number of pixels. Nonetheless, the time taken in the *1920x1080* resolution case, decreased by more than half of the one presented in the CPU case, taking the data transfer times between CPU and GPU into account.

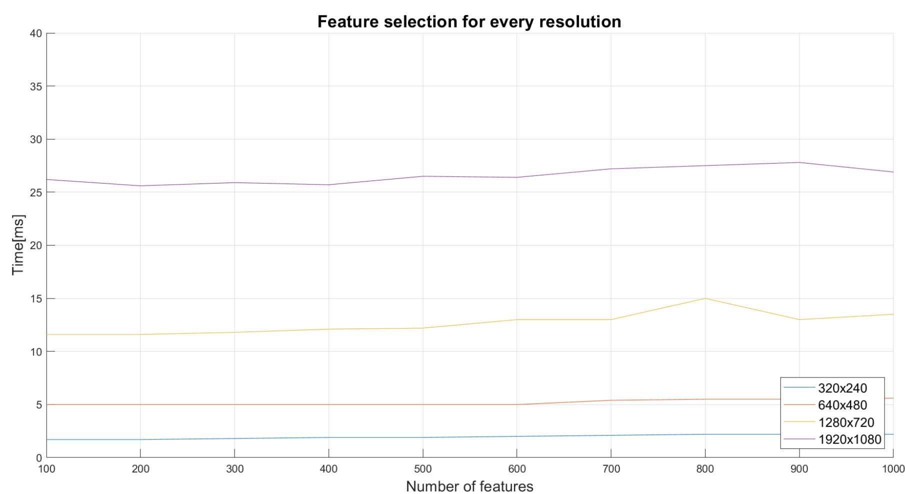

Fig. 4.2 Curves of time per frame taken, per number of features for each resolution in GPU selection.

4.1.2 Fiducial Marker Detector

The proposed application developed in this work, intends to track a set of fiducial markers, commonly known as *ArUco* markers. In this specific case, the markers were generated with the *Alvar* library, and so the detector must also use this one and the same parameters that generate this specific set of markers. The first step in the development of this solution, is then to choose a good detector, that is robust, supports the *Alvar* library and has the best performance possible. With all these requirements in mind, it was decided to use the *OpenCV* built in detector. It offers a good performance, good enough robustness and it is accessible to use, while it is also easy to integrate with the rest of the developed code. Figure 4.3 illustrates the markers in the scene with a green bounding box and the corresponding marker **id** printed in blue at the middle.

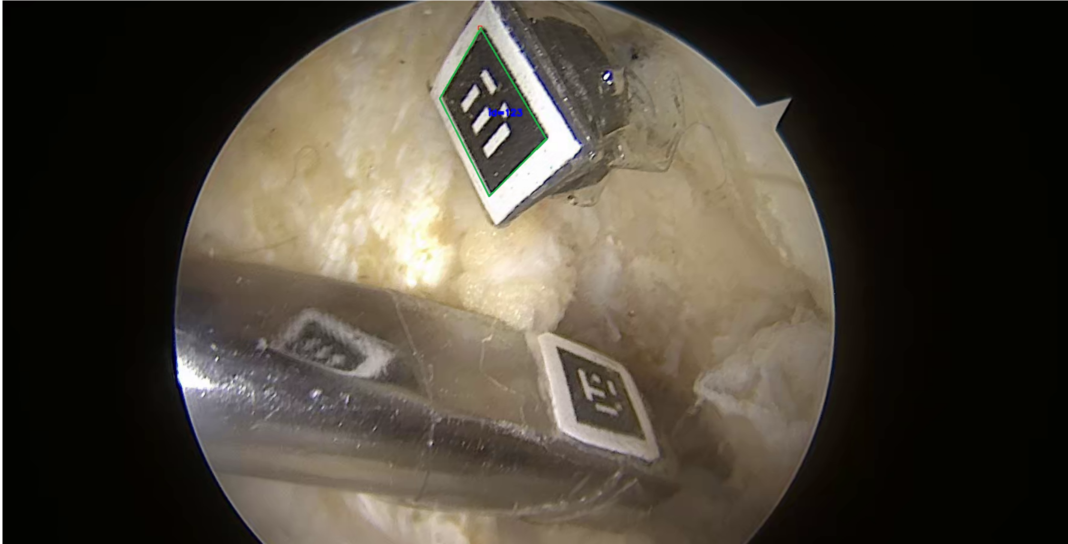

Fig. 4.3 Marker and the respective detection box.

To primarily understand its capabilities, the time was taken with the detector alone, for 800 frames of the provided video and with its native resolution of *1920x1080*. Table 4.1 shows the results obtained.

Table 4.1 Average results of the marker detection.

Number of Frames	Frames per Second(FPS)	Time per Frame (s)	Total time (s)
800	20	0.051	40.87

4.2 Tracking step

The second step of this dissertation was to find a tracking algorithm for the final solution. Therefore, the KLT algorithm and the NVIDIA Optical Flow SDK based tracker developed from scratch for the purpose of this dissertation, were tested and analysed to support the decision on which method to use in the posed real-time application.

4.2.1 KLT feature tracker in CPU and GPU

Following what was already emphasized in chapter 4.1.1, KLT is essentially composed of two main parts, a detection step already explored and analyzed as well as a tracking step. This second one relies on the optical flow algorithm developed by *Bruce Lucas* and

Takeo Kanade called the LK algorithm [22] also mentioned and explained in more depth on previous chapters. Equivalently to what happened earlier in the detection step, some testing was performed, both in CPU and GPU versions of the algorithm and based on that, a discussion on the results is presented. Figure 4.4 shows the curves obtained with the values of time over the number of features per resolution.

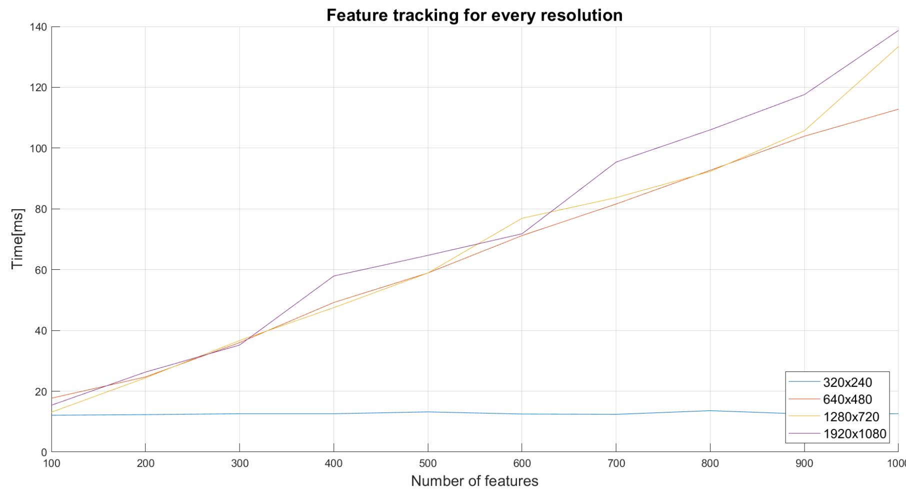

Fig. 4.4 Curves of time per frame taken per number of features for each resolution in CPU tracking step.

Analysing these results, it is noticeable a degradation in time, that approaches a linear curve, for every resolution along with the increase in the number of features, with the single exception of the *320x240* resolution case in which the same explanation given in the selection step applies, i.e. the information is stored in cache due to its reduced size, which in turn neutralizes the bottleneck created in the memory access. In the cases of higher resolutions it is possible to verify a great increase in time spent per frame, which makes this form of the algorithm barely impracticable, even more in the case of a real time application as this work intends to achieve. With this goal in mind, a GPU version using *CUDA* was also tested and debated, with the respective results obtained displayed in figure 4.5. In it, it is possible to observe a much less steeper curve in every case, reducing the processing times up to a sixth of the times obtained with CPU. This is due to the fact that in these case the use of a GPU architecture, allows for the utilization of multiple processors in parallel, which in turn makes the processing of numerous features possible simultaneously, increasing the throughput and thus reducing processing time considerably. The *320x240* case draws attention once again, because in this specific case opposite to what happened in the previous cases the time increases even if just marginally. This can be attributed to the transfer times between CPU

and GPU, usually called host-device data transfers, which occurs via a memory bus and can take a considerable time. This is something to take into account when choosing the parallel approach in spite of a sequential one, as it may overlap any advantages, contrary to what is seen in the current case in which considerable gains are achieved.

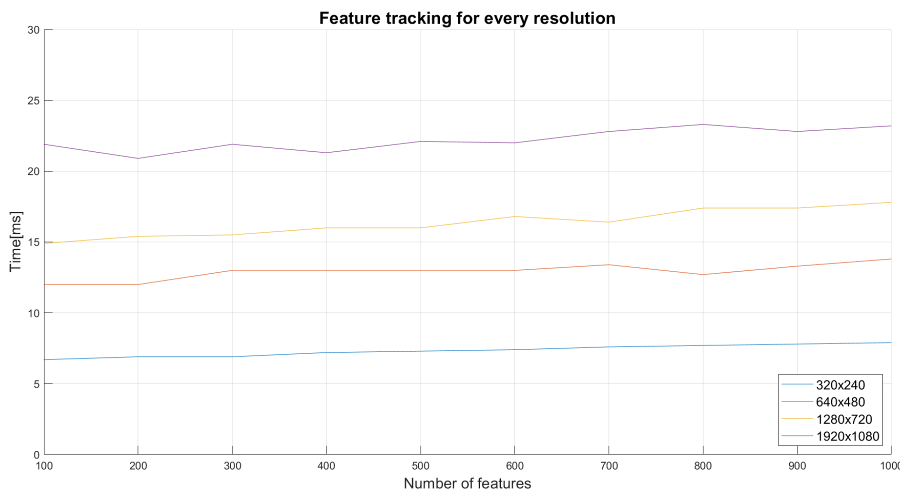

Fig. 4.5 Curves of time per frame taken per number of features for each resolution in GPU tracking step.

4.2.2 NVIDIA Optical Flow SDK based tracker

Following the fiducial marker detector results presented before, the second part of the NVOFA application consists of an optical flow tracker based on the recent *NVIDIA Optical Flow SDK* as explained earlier in chapter three of this dissertation in more detail. The developed algorithm, makes use of the flow vectors provided by the *SDK* to perform the calculations and predict the position of a given point of the current frame in the next one. In this case, it is used to predict the location of the four corners of each detected marker. For a sake of simplicity and to focus on the performance of the algorithm *per se*, the registered values were only taken with the detection of a single marker, since in the case studied on this document, which focus more on the surgical field with emphasis on the arthroscopic knee surgery, the number of markers present simultaneously is reduced, usually no more than two or three, and so this application can be relatively easily escalated. The results obtained with NVOFA tracker are the composition of both the detection and the tracking and were taken by altering various parameters, such as the **preset**, **grid size** and the detection frame interval. **Preset** and **grid size** are internal parameters of the *API* that can be changed from among a set of existing values.

The majority of the existing tracking methods, for example, the ones based on machine learning, do the object detection in every frame, and that is the way in which tracking is achieved. The developed algorithm in this dissertation, intends to reduce the number of frames in which detection is performed, and with this, it is expected to obtain lower running times and thus a better performance. With this in mind tests were put in place for a different detection frame interval of 5, 10 and finally 20 frames. In short, the detection frame interval is the interval of frames in which the detector is called to give a new set of points and consequently update the marker position which in those intervals is updated by the tracker only. This method allows for the algorithm to get a significant speed up and increase the performance relatively to a detection only method. The results for the different combination of parameter values are shown in tables 4.2, 4.3 and 4.4 .

Table 4.2 Average results of detection and tracking for 800 frames and a resolution of 1920x1080 with detection interval of 5 frames.

Average FPS	Time per Frame (s)	preset	grid size	detection frame interval
65	0,01538	fast	4	5
35	0,02858	fast	2	5
15	0,06667	fast	1	5
62	0,01613	medium	4	5
31	0,03226	medium	2	5
11	0,09090	medium	1	5
43	0,023256	slow	4	5
24	0,04167	slow	2	5
9	0,11111	slow	1	5

Table 4.3 Average results of detection and tracking for 800 frames and a resolution of 1920x1080 with detection interval of 10 frames.

Average FPS	Time per Frame (s)	preset	grid size	detection frame interval
69	0,01449	fast	4	10
36	0,02778	fast	2	10
14	0,07142	fast	1	10
64	0,01563	medium	4	10
31	0,032258	medium	2	10
11	0,090909	medium	1	10
44	0,02273	slow	4	10
25	0,04	slow	2	10
9	0,11111	slow	2	10

Table 4.4 Average results of detection and tracking for 800 frames and a resolution of 1920x1080 with detection interval of 20 frames.

Average FPS	Time per Frame (s)	preset	grid size	detection frame interval
69	0,01449	fast	4	20
36	0,02778	fast	2	20
14	0,07143	fast	1	20
66	0,01515	medium	4	20
32	0,03125	medium	2	20
11	0,09090	medium	1	20
45	0,02222	slow	4	20
25	0,04	slow	2	20
9	0,11111	slow	2	20

According to the results obtained, the best scenario is the one with the preset "fast" and grid size 4. The preset has three levels, "fast", "medium" and "slow" which can be chosen. "Fast" is the level which takes less resources of the optical flow engine but as a counterpart is the level with less precision. The other two levels increase the precision and the resources used as well, meaning less performance but better accuracy. Grid size varies between 4, 2 and 1, corresponding to the number of pixels per flow vector, i.e. it is a parameter of

granularity. The higher the value the lesser vectors are produced and lesser time is taken per frame. Figure 4.6 shows the produced vectors used in the algorithm.

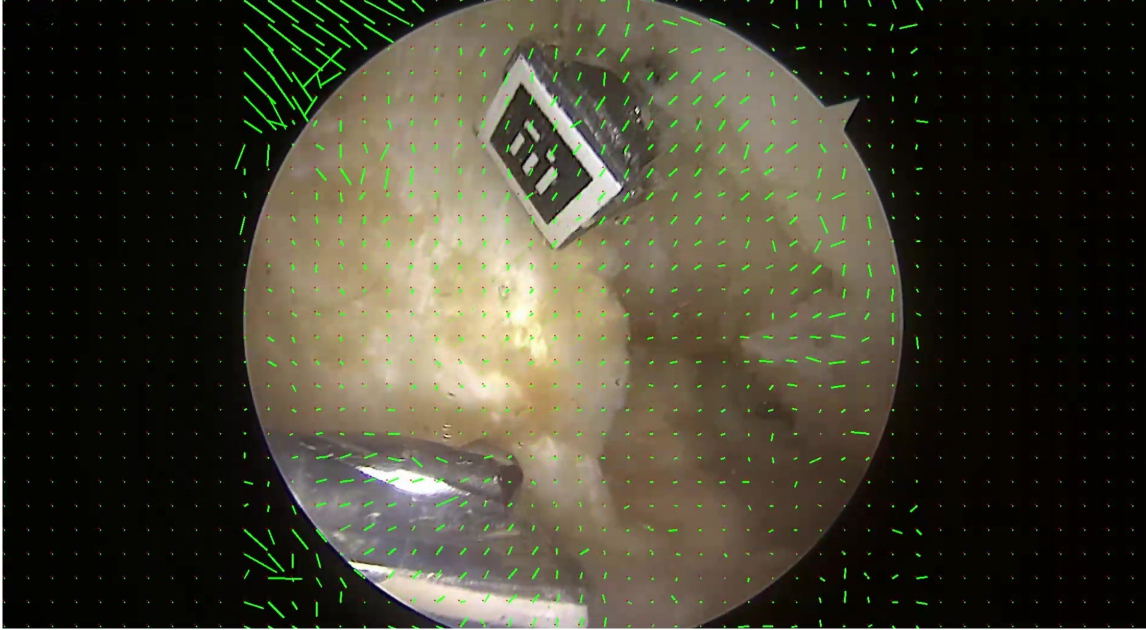

Fig. 4.6 Resulting vectors produced by the Optical Flow engine.

In this figure, it is also clear to see that vectors are only calculated in the center part of the image, and are zero in every other point of the dark side. Exception is on corners of a square drawn around the circle with length equal to the circle diameter. Since the function only admits coordinates of a polygon, this necessarily implies that the dark spots near the corners of the image are required to have the vectors calculated even though the results cannot be taken into account since the values are not viable.

4.3 Comparison Between the KLT and NVOFA applications

After the results of both tracking methods are presented we are now able to make a comparison between them. First, the KLT algorithm was presented and analyzed and it was concluded that the GPU version surpasses by a great margin its CPU counterpart. Figure 4.7 summarizes the results obtained with KLT. Even though, with a resolution of *1920x1080* the *CUDA* version still struggles and can only manage to get around a maximum of 20 fps and

would still require an algorithm such as the one proposed in [15]. The detection and tracking, working in real-time, can be observed in figure 4.8.

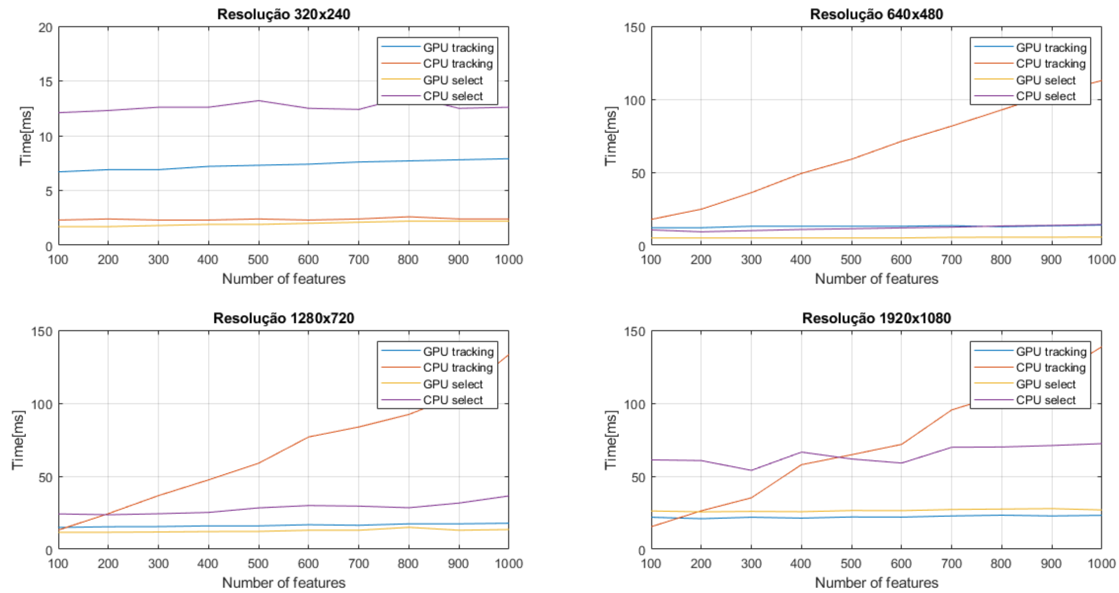

Fig. 4.7 Time per number of features for each step of the algorithm in CPU and GPU.

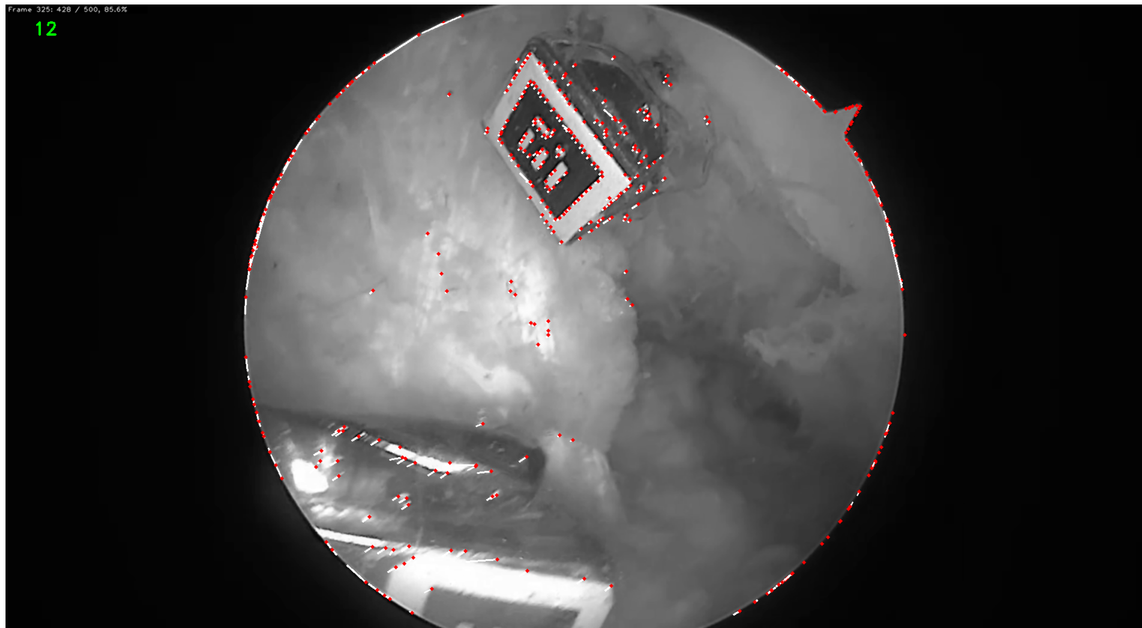

Fig. 4.8 Marker and the respective detection box.

On the other hand, the NVOFA version presents better results, with the best case achieving around 70 fps and with full marker tracking capabilities. This approach achieves a speedup

of around four times when compared with the observed in KLT. This makes it possible to achieve the real-time needed in a real world application, and brings other advantages and possibilities inherent from the NVIDIA platform itself.

4.4 Summary

In this chapter, the implementation of both KLT and NVOFA applications is addressed. The sequence of events that goes from the analyzing and testing of KLT to the development of a final solution, as well as, the comparison between both methods and their results are described. The KLT algorithm analyzed contains two possible solutions, a CPU and a GPU-based one. Both are analyzed in detail and tested for this use case. Also, the developed application using NVOFA is tested and analyzed in detail, offering a good alternative in terms of performance to KLT, and is intended to serve as a starting point for other future works.

Chapter 5

Conclusion and Future Work

The goals identified for this dissertation in the first chapter were partly achieved. The objectives proposed were to explore existing feature tracking methods and implement an optical flow solution in the real life case scenario of arthroscopic surgery, and afterwards compare this method with KLT.

In order to achieve this goal, a solution based on the NVIDIA recent SDK, called NVIDIA Optical Flow SDK, which brings a different and innovative approach to the optical flow and object tracking was implemented.

This solution demonstrates that it is possible to achieve real-time object tracking resorting to modern state-of-the-art methods and that LK despite being an outdated method is still a classical and powerful algorithm that can be used even nowadays in some scenarios as an efficient and cheap alternative to other methods.

However, it should be noted that this solution, built based on NVIDIA Optical Flow SDK presents some disadvantages, such as:

- NVIDIA SDKs require the usage of an NVIDIA graphics card. This disadvantage is also shared with the case of *CUDA* KLT algorithm.
- The Optical Flow SDK not only requires an NVIDIA graphics card, as it requires one with the newer architectures starting with *Turing*, which contains the optical flow dedicated engine.
- As a consequence of the two previous disadvantages comes a third one. The price of the hardware could increase the cost of the final solution, and added to that the physical space needed could make it impractical for some use cases, but not for the medical context.

- This method as is presented, does not offer a very precise tracking yet, due to the limitations of the pure translational method that is used. Although, it presents a good margin for improvements and could easily be adapted into a more robust an complex algorithm, for example with the integration of an affine motion model.

However, NVOFA presents very important strengths when compared to the classical tracking methods such as KLT. The performance is clearly superior, allowing to reach real-time processing and more than 60 FPS in a video sequence. When compared to other state-of-the-art approaches specifically ML ones, this algorithm offers great advantages starting from the fact that it can take images with great resolution, something that is required nowadays. It also offers more possibilities and versatility regarding the detection method to be used. With all that said, such an algorithm could feature as a complement in an ML solution. This method also enables an easy integration with an augmented reality application since it already provides the vectors with the relative movement of the object in the image. With camera calibration algorithms this could be achieved with few complexity and keeping the performance gains.

Also one of the advantages and as such one of the reasons this method was chosen, is the fact that the heavier part of this type of algorithm, which is the tracking part, is calculated in a specific engine built in the GPU itself. This releases the processing load from the CPU side, and even from the GPU streaming multiprocessors, allowing to use other GPU capabilities such as *CUDA*, already mentioned and used before in the development of this dissertation, to perform other tasks, such as the operations relative to the update of the corners with the generated vectors. That can be especially beneficial since both resources are present in the device and so data transfers become negligible, boosting the performance even more.

We conclude that NVOFA presents advantages in some aspects over KLT and other classical tracking methods. However, there are some aspects to consider regarding the implementation in real-world that sticks mainly with the costs associated as mentioned before.

This said, optical flow algorithms justifies implementation for example in medical, surveillance and augmented reality scenarios.

Considering the work developed and the results obtained, there are several possibilities for future work, such as:

- Improve the accuracy of the algorithm in order to implement this solution in a real world arthroscopic surgery procedure;

- Implement this method in an augmented reality solution with the usage of a camera calibration algorithm and lens distortion, such as the one explored and proposed by R.Melo et al. in [54], which also focus on arthroscopic surgery, computer vision algorithms.
- Improve the tracking algorithm in order to increase the number of markers tracked simultaneously and make it more robust, specifically in the case of rotations.
- Improve the performance of the algorithm with the usage of CUDA.
- Develop a way to better evaluate the error between the calculated position of the marker and its real position, in order to have a feedback that can help to improve accuracy, altogether with all the improvements suggested previously.

Figure 5.1 shows a summary diagram of the advantages and disadvantages presented during this chapter.

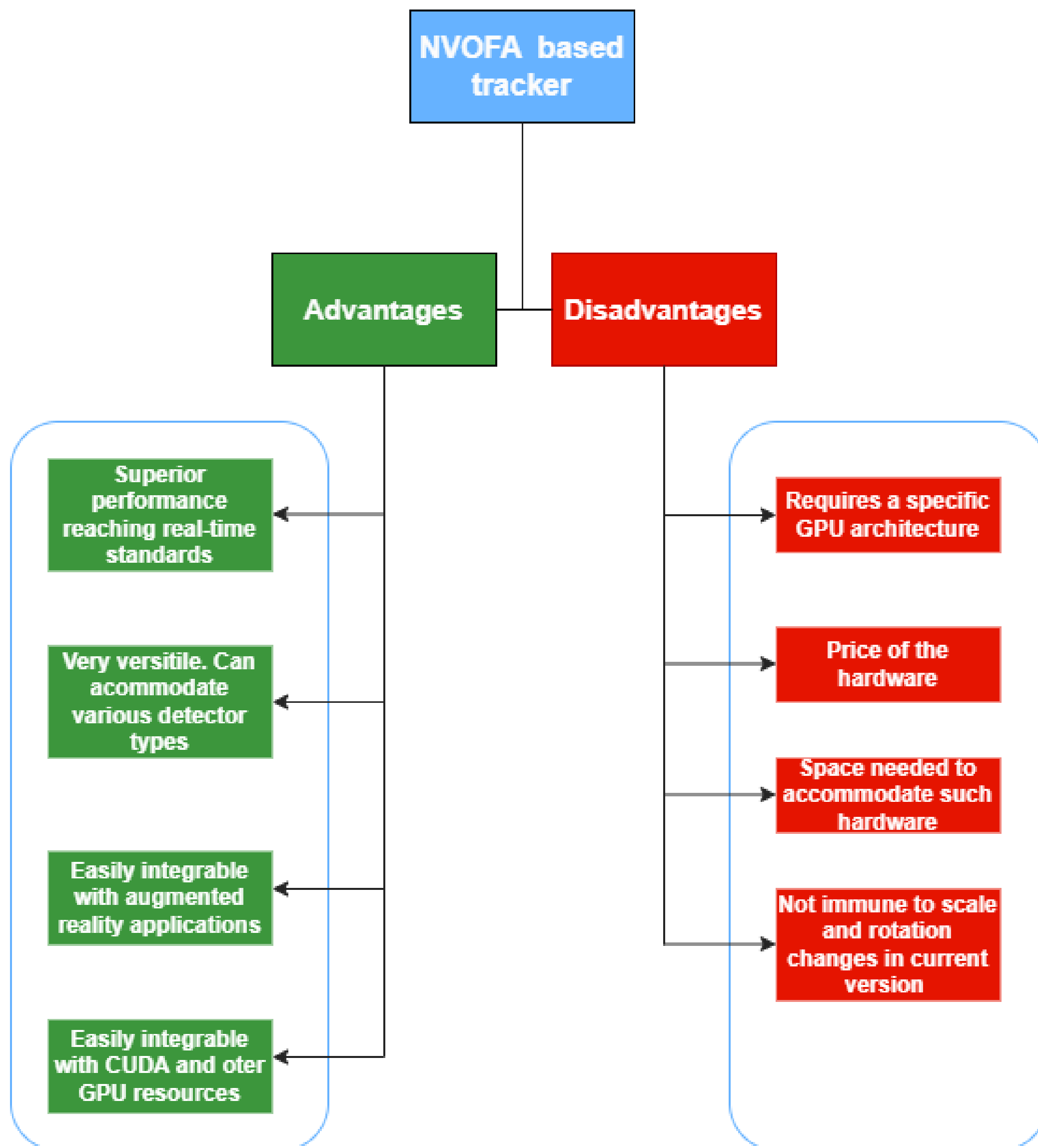

Fig. 5.1 Advantages and disadvantages of the application developed in the scope of this dissertation.

References

[1] S. Chen, Y. Wang, and C. Cattani, “Key issues in modeling of complex 3d structures from video sequences,” *Mathematical Problems in Engineering*, vol. 2012, 01 2012.

[2] S. Baker, R. Patil, G. Cheung, and I. Matthews, “Lucas-Kanade 20 Years On: An Unifying Framework: Part 5,” *CMU-RI Report*, vol. 56, no. 3, p. 14, 2004.

[3] S. Baker, *Inverse Compositional Algorithm*. Boston, MA: Springer US, 2014, pp. 426–428. [Online]. Available: https://doi.org/10.1007/978-0-387-31439-6_759

[4] D. Khan, M. A. Shirazi, and M. Y. Kim, “Single shot laser speckle based 3D acquisition system for medical applications,” *Optics and Lasers in Engineering*, vol. 105, no. September 2017, pp. 43–53, 2018. [Online]. Available: https://doi.org/10.1016/j.optlaseng.2018.01.001

[5] R. J. Devrim Unay, Ahmet Ekin, “MEDICAL IMAGE SEARCH AND RETRIEVAL USING LOCAL BINARY PATTERNS AND KLT FEATURE POINTS Devrim Unay , Ahmet Ekin , Radu Jasinschi Video Processing and Analysis Group,” *Image (Rochester, N.Y.)*, pp. 997–1000, 2008.

[6] C. Hung Hsieh, J. Der Lee, and C. Tsai Wu, “A Kinect-based Medical Augmented Reality System for Craniofacial Applications Using Image-to-Patient Registration,” *Neuropsychiatry*, vol. 07, no. 06, pp. 927–939, 2017.

[7] B. Benfold and I. Reid, “Stable multi-target tracking in real-time surveillance video,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3457–3464, 2011.

[8] P. Bagherpour, S. A. Cheraghi, and M. Bin Mohd Mokji, “Upper body tracking using KLT and kalman filter,” *Procedia Computer Science*, vol. 13, pp. 185–191, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2012.09.127

[9] R. C. Joshi, A. G. Singh, M. Joshi, and S. Mathur, “A Low Cost and Computationally Efficient Approach for Occlusion Handling in Video Surveillance Systems,” *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 5, no. 7, p. 28, 2019.

[10] F. Jabar, S. Farokhi, and U. U. Sheikh, “Object tracking using SIFT and KLT tracker for UAV-based applications,” pp. 65–68, 2016.

[11] D. Thirde, M. Borg, J. Aguilera, J. Ferryman, K. Baker, and M. Kampel, “Evaluation of object tracking for aircraft activity surveillance,” *Proceedings - 2nd Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance, VS-PETS*, vol. 2005, pp. 145–152, 2005.

[12] R. Boda and M. J. P. Priyadarsini, “Face detection and tracking using klt and viola jones,” *ARPN Journal of Engineering and Applied Sciences*, vol. 11, no. 23, pp. 13 472–13 476, 2016.

[13] M. Salehpour and A. Behrad, “3D face reconstruction by KLT feature extraction and model consistency match refining and growing,” *2012 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications, SETIT 2012*, no. 1, pp. 297–302, 2012.

[14] R. From and A. N. Image, “Reconstruction from an image sequence,” no. 1, pp. 3–6, 2003.

[15] O. Toole and D. Dolben, “Marker Detection and Tracking for Augmented Reality Applications.” [Online]. Available: http://xanthippi.

[16] P. C. Lusk and R. W. Beard, “Visual Multiple Target Tracking from a Descending Aerial Platform,” *Proceedings of the American Control Conference*, vol. 2018-June, pp. 5088–5093, 2018.

[17] W. Ye, R. Zheng, F. Zhang, Z. Ouyang, and Y. Liu, “Robust and Efficient Vehicles Motion Estimation with Low-Cost Multi-Camera and Odometer-Gyroscope,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 4490–4496, 2019.

[18] X. Cao, J. Lan, P. Yan, and X. Li, “KLT feature based vehicle detection and tracking in airborne videos,” *Proceedings - 6th International Conference on Image and Graphics, ICIG 2011*, pp. 673–678, 2011.

[19] L. Yang, J. Johnstone, and C. Zhang, "A Multi-camera Approach to Vehicle Tracking Based on Features," pp. 79–80, 2008.

[20] W. Jang, S. Oh, and G. Kim, "A hardware implementation of pyramidal KLT feature tracker for driving assistance systems," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pp. 220–225, 2009.

[21] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, no. June, pp. 2544–2550, 2010.

[22] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, August 24-28, 1981*, P. J. Hayes, Ed. William Kaufmann, 1981, pp. 674–679. [Online]. Available: http://ijcai.org/proceedings/1981-1

[23] N. Sharmin and R. Brad, "Optimal filter estimation for Lucas-Kanade optical flow," *Sensors (Switzerland)*, vol. 12, no. 9, pp. 12 694–12 709, 2012.

[24] C. Tomasi, "Detection and Tracking of Point Features," *School of Computer Science, Carnegie Mellon Univ.*, vol. 91, no. April, pp. 1–22, 1991. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.5899&rep=rep1&type=pdf

[25] J. Shi and C. Tomasi, "Good Features," *Image (Rochester, N.Y.)*, pp. 593–600, 1994.

[26] J. S. Kim, M. Hwangbo, and T. Kanade, "Realtime affine-photometric KLT feature tracker on GPU in CUDA framework," *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009*, no. March 2014, pp. 886–893, 2009.

[27] N. Ramakrishnan, T. Srikanthan, S. K. Lam, and G. R. Tulsulkar, "Adaptive Window Strategy for High-Speed and Robust KLT Feature Tracker," in *Image and Video Technology*, T. Bräunl, B. McCane, M. Rivera, and X. Yu, Eds. Cham: Springer International Publishing, 2016, pp. 355–367.

[28] J. Bouguet, "Pyramidal implementation of the affine lucas kanade feature tracker," *Intel Corporation*, vol. 1, no. 2, pp. 1–10, 2001. [Online]. Available: jean-yves.bouguet@intel.com

[29] K. Derpanis, "The Gaussian Pyramid," *WWW: http://www. cse. yorku. ca/∼ kosta/ . . .* , pp. 3–4, 2005. [Online]. Available: http://www.cse.yorku.ca/$\sim$kosta/CompVis_Notes/gaussian_pyramid.pdf

[30] C. H. Anderson, P. J. Burt, and G. S. van der Wal, "Change Detection and Tracking Using Pyramid Transform Techniques," in *Intelligent Robots and Computer Vision IV*, D. P. Casasent, Ed., vol. 0579, International Society for Optics and Photonics. SPIE, 1985, pp. 72–78. [Online]. Available: https://doi.org/10.1117/12.950785

[31] S. BIRCHFIELD, "Klt : An implementation of the kanade-lucas-tomasi feature tracker," *http://www.ces.clemson.edu/stb/klt/*. [Online]. Available: https://ci.nii.ac.jp/naid/10020490241/en/

[32] S. Birchfield, "KLT: An implementation of the Kanade- Lucas-Tomasi feature tracker." [Online]. Available: https://cecas.clemson.edu/$\sim$stb/klt/

[33] J. S. Johan Hedborg and M. Felsberg, "KLT TRACKING IMPLEMENTATION ON THE GPU Johan Hedborg , Johan Skoglund and Michael Felsberg Computer Vision Laboratory Department of Electrical Engineering ,," no. 1, pp. 2–4.

[34] S. N. Sinha, J. M. Frahm, M. Pollefeys, and Y. Genc, "Feature tracking and matching in video using programmable graphics hardware," *Machine Vision and Applications*, vol. 22, no. 1, pp. 207–217, 2011.

[35] C. Zach, D. Gallup, and J. M. Frahm, "Fast gain-adaptive KLT tracking on the GPU," *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops*, no. 1, 2008.

[36] J. F. Ohmer and N. J. Redding, "GPU-accelerated KLT tracking with monte-carlo-based feature reselection," *Proceedings - Digital Image Computing: Techniques and Applications, DICTA 2008*, pp. 234–241, 2008.

[37] G. Qian and R. Chellappa, "Structure from Motion Using Sequential Monte Carlo Methods," *International Journal of Computer Vision*, vol. 59, no. 1, pp. 5–31, 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000020669.68126.4b

[38] P. Mainali, Q. Yang, G. Lafruit, R. Lauwereins, and L. Van Gool, "Robust low complexity feature tracking," *Proceedings - International Conference on Image Processing, ICIP*, pp. 829–832, 2010.

[39] T. Sakayori and T. Ikenaga, "“Implementation of Hardware Engine for Real-Time KLT Tracker”," *Journal of the Institute of Image Electronics Engineers of Japan*, vol. 38, no. 5, pp. 656–663, 2009.

[40] A. H. A. El-Shafie and S. E. Habib, "Survey on hardware implementations of visual object trackers," *IET Image Processing*, vol. 13, no. 6, pp. 863–876, 2019.

[41] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," pp. 23.1–23.6, 2013.

[42] T. Hu, H. Wu, and T. Ikenaga, "FPGA implementation of high frame rate and ultra-low delay tracking with local-search based block matching," *Proceedings - 2017 International Conference on Machine Vision and Information Technology, CMVIT 2017*, pp. 93–98, 2017.

[43] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0004370281900242

[44] E. Mémin and P. Pérez, "Dense estimation and object-based segmentation of the optical flow with robust techniques," *IEEE Transactions on Image Processing*, vol. 7, no. 5, pp. 703–719, 1998.

[45] J. BARRON, D. FLEET, and S. BEAUCHEMIN, "Performance of Optical Flow Techniques," 1994.

[46] G. L. Besnerais and F. Champagnat, "Dense optical flow by iterative local window registration," *Proceedings - International Conference on Image Processing, ICIP*, vol. 1, pp. 134–137, 2005.

[47] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool, "Fast Optical Flow Using Dense Inverse Search," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 471–488.

[48] N. Bauer, P. Pathirana, and P. Hodgson, "Robust Optical Flow with Combined Lucas-Kanade/Horn-Schunck and Automatic Neighborhood Selection," pp. 378–383, 2006.

[49] Nvidia, "NVIDIA Optical Flow SDK," no. June, pp. 1–9, 2018. [Online]. Available: https://developer.nvidia.com/opticalflow-sdk

[50] N. Tracker, "NVIDIA Optical Flow Engine-Assisted Object Tracker," no. June, 2021.

[51] H. Ahn and H. J. Cho, "Research of multi-object detection and tracking using machine learning based on knowledge for video surveillance system," *Personal and Ubiquitous Computing*, 2019.

[52] X. Liu, S. W. Chen, S. Aditya, N. Sivakumar, S. Dcunha, C. Qu, C. J. Taylor, J. Das, and V. Kumar, "Robust Fruit Counting: Combining Deep Learning, Tracking, and Structure from Motion," *IEEE International Conference on Intelligent Robots and Systems*, pp. 1045–1052, 2018.

[53] L. Li, H. Li, D. Liu, Z. Li, H. Yang, S. Lin, H. Chen, and F. Wu, "An Efficient Four-Parameter Affine Motion Model for Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 8, pp. 1934–1948, 2018.

[54] R. Melo, J. P. Barreto, and G. Falcão, "A new solution for camera calibration and real-time image distortion correction in medical endoscopy-initial technical evaluation (IEEE Transactions on Biomedical Engineering (2012) 59, 3, (634-644))," *IEEE Transactions on Biomedical Engineering*, vol. 59, no. 7, p. 2095, 2012.