FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Localization of Mobile Robots Using Optical Flow Sensors and Sensor Fusion

Eduardo Vila-Chã

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Paulo Gomes Da Costa

August 21, 2022

Abstract

Motion estimation is one of the most relevant areas under discussion in robotics and computer vision. By extracting information from vision field, it's possible to build systems that help the localisation of mobile robots.

Optical Flow is a displacement of pixel values in the image sequence induced by a movement of a camera or a scene observed by it. This technique that has been studied since the 1980's, is of extreme importance to Visual Simultaneous and Localisation and Mapping System (VSLAM) through the use of Visual Odometry. This method is being studied deeply in the last 10 years, due to the low cost of cameras and the easiness of its configuration. Visual Odometry has been proved that it can work better in low-illuminance and slippery surfaces, something that wheel odometry, for example, is not capable of doing. Even though useful, it's necessary to use sensor fusion in order to have more robust localizationresults.

Scientifically, this thesis is focused on the study of visual odometry using optical flow. The only camera used is pointed at the ground, since there is less change of scenes compared to a camera pointed at the environment. Four traditional optical flow algorithms will be developed. They will be tested on a public dataset, on a floor sequence created from images of the selected camera and then applied on real-time.

Keywords: Optical Flow, Visual SLAM, Visual Odometry, ESP32, Autonomous Robot Navigation

Resumo

A estimativa de movimento é uma das áreas mais relevantes na discussão da robótica e visão computacional. Ao extrair informações do campo da visão, é possível construir sistemas que auxiliam a localização de robôs móveis.

O fluxo ótico é um deslocamento de valores de pixel, numa sequência de imagens, que é induzida por um movimento de uma câmera ou uma cena observada por ela. Esta técnica que é estudada desde a década de 1980, é de extrema importância para o Sistema Simultâneo e de Localização e Mapeamento Visual através do uso da Odometria Visual. Este método tem sido estudado mais profundamente nos últimos 10 anos, devido ao baixo custo das câmeras e à facilidade da sua configuração. A Odometria Visual funciona melhor em ambientes de baixa iluminância e com piso escorregadio, algo que a odometria de baseada em codificadores de rodas, por exemplo, não é capaz de fazer. Apesar de produzir resultados de localização consideravlemente bons, é necessário usar fusão de sensores para obter resultados mais robustos e consistentes.

Cientificamente, esta tese está foca-se no estudo da odometria visual usando o fluxo ótico. A câmera utilizada é apontada para o solo, pois há menos mudança de cena em relação a uma câmera apontada para o ambiente em redor do robô. Serão desenvolvidos quatro algoritmos tradicionais de fluxo ótico. Estes serão testados em um conjunto de dados públicos, em uma sequência de piso criada a partir de imagens da câmera selecionada (ESP32-Cam) e aplicada em tempo real.

Keywords: Fluxo Ótico, Visual SLAM, Odometria Visual, ESP32, Navegação Autónoma de Robôs

Acknowledgements

I would like to express my appreciation for the support of my supervisor, Paulo Costa, and co-supervisor, José Lima, who guided me throughout this project. I could not have undertaken this journey without my family and girlfriend.

This dissertation was supervised by Paulo Gomes da Costa and co-supervised by José Luís Sousa de Magalhães Lima.

Eduardo Vila-Chã

"Until I began to learn to draw,
I was never much interested in looking at art."

Richard P. Feynman

Contents

List of Figures

List of Tables

Abbreviations

OF	Optical Flow
VO	Visual Odometry
V-SLAM	Viusal Simultaneous Localization and Mapping

Chapter 1

Introduction

1.1 Motivation

Robots are becoming more and more part of the everyday life. The applications and the fields of work are several, like surgical operations, industry automation, and many more to enumerate. Mobile robots are vital since the locomotive part enables them to move to different regions, study their environment, and perform tasks. One of the most important advantages that mobile robots have is that they can work in environments where humans cannot, like in outer space or at the bottom of the sea. They are also valuable because they can do repetitive and dangerous tasks without tiring and with a very high level of accuracy.

A critical challenge for all mobile robots is the ability to answer the question, "Where am I?" [55]. The answer to this question can be obtained through the robot's sensors that acquire information about the environment. This information is processed and then used to estimate the robot's position. Localisation is a very important process that allows the mobile robot to know its position in relation to its surroundings, so it can plan its movements and achieve its objectives. This process becomes more difficult when the environment is unknown or unstructured, as is often the case.

The sensor's data from the mobile robot can be proprioceptive, that is, measure the state of the system (robot), for example, motor speed, wheel load, heading of the robot, battery status, etc. and exteroceptive, information from the robot's environment, for example, distances to objects, intensity of the ambient light, unique features, etc.

Sensors based on vision, generally cameras, are part of the exteroceptive sensor group, and are quite useful due to the rich perceptual input provided by vision [73] and to the navigation of autonomous robots. Autonomous not only means the freedom from any operator, but it also describes that, it does not utilise information from other devices, namely radars [81]. In particular, navigation based on vision needs to segment the traversable path and distinguish it from objects that need to be avoided [64].

The approach for vision-based robot navigation in an unstructured environment, where no prior knowledge of the robot environment is available, presently relies on optical flow calculation. Some stereo and pattern matching approaches are also being tested. Logically, optical flow only provides information about the environment, the control laws and decisions of the robot are made by a navigation system like V-SLAM (Visual Simultaneous Localisation and Mapping) and Obstacle Avoidance System, for example.

Several improvements and advancements have been made in order to compute optical flow more accurately and in less time in the present literature. But due to the recent interest in visual navigation system like V-SLAM, there isn't so much literature on visual navigation system and practical results. This is also due to the many application of optical flow outside of mobile robot navigation.

In this thesis, a visual odometry system will be created based on optical flow that will be used to help the localisation of the robot, being one of the sensor used in a sensor fusion algorithm. With visual odometry, it will be possible to determine the speed of the robot and subsequently, to determine the location of it. A navigation system will be discussed, but no implemented. This will be tested in real-life situations with the study of different types of floors (plain, rug, etc.) with the ESP32-Cam moving in just one direction with a specific height and resolution of image. The variations and characteristics of the experiments will be explained in the next Sections.

1.2 Previous Work

Before any practical implementation of optical flow algorithms, the concepts and theoretical background of optical flow had to be studied. This meant understanding concepts of computer vision like convolution filters, average filters and their applications among others concepts. With this concepts in mind, the next step of the study consisted in learning how to apply optical flow techniques to visual navigation systems. A number of important concepts like odometry and SLAM (Visual Simultaneous Localisation and Mapping) were studied and researched in the literature.

All this theoretical work was necessary to understand and develop the State-of-Art that explores all the advancements and improvements in optical flow and visual navigation systems. In the practical part of this work, since only traditional optical flow algorithms were implemented, this means that no convolution neural networks were used or any advanced algorithms, the practical work only consisted in the implementation of well-studied and accessible algorithms with no need of more advanced concepts explored in State-of-Art.

Another important aspect of the practical implementation had to do with use of the ESP32-Cam. In order to work with the ESP32-Cam, some knowledge of C++ language (Arduino platform) was needed, as well as library usage in order to use the OV2460 camera.

Finally, the structure of the experiments was studied in order to induce the minimal number of errors.

1.3 Objectives

The main objectives of this thesis consists in understanding different optical flow algorithms and their characteristics, and how to apply them to visual navigation system. The following list will add other important points of study and research:

1. Understand different optical flow algorithms and approaches to Visual Odometry
2. Design and implementation of optical flow algorithms
3. Application of optical flow algorithms on Public Datasets
4. Creation of floor sequences with ESP32-Cam
5. Test of optical flow on floor sequences
6. Development and test of real-time optical flow system (Visual Odometry)
7. Analysis of results and conclusions
8. Proposal of improvements and Future Work

Chapter 2

State of the Art

There have been many strategies proposed to decrease algorithm's runtime, enhance optical flow accuracy, and improve the navigation system of mobile robots. The development of deep learning networks has altered the course of optical flow estimation research [70].

Optical flow methods are now being developed with the aid of convolutional neural networks, and almost all top-performing algorithms include deep learning architectures in their approaches.

Localisation strategies based on optical flow also show a significant improvement in recent years. The main points of investigation have been Visual SLAM (Simultaneous Localisation and Mapping) [6] and localisation systems utilising afocal optical flow sensor (AOFS) based on sensor fusion for indoor service robots in low luminance and slippery environment [85].

The conditions mentioned (low luminance and slippery environment) are of great importance since these are the conditions that can lead to a lack of good results in the classic and some modern approaches. The OFS-based odometry is advantageous for mobile robots because it isn't hampered by wheel slip. However, when vertical height varies, systematic errors are generated in the movement distance of robots on uneven surfaces [84].

Another approach is to utilise a multi-model localisation system based on ceiling features, such as lights [20]. Ceiling vision has the benefit of allowing images to be analysed without being scaled, thus less affected by occlusions due to obstacles moving around, increasing the mapping database reliability [20].

2.1 Optical Flow

In recent years, several new approaches and ideas have emerged in this area. In the last decade, particularly, significant progress has been made.

The advance level datasets such as Middlebury [5], MPI-Sintel [13] and KITTI [31, 53] presented substantial novel challenges for the optical flow algorithms. The traditional methods such

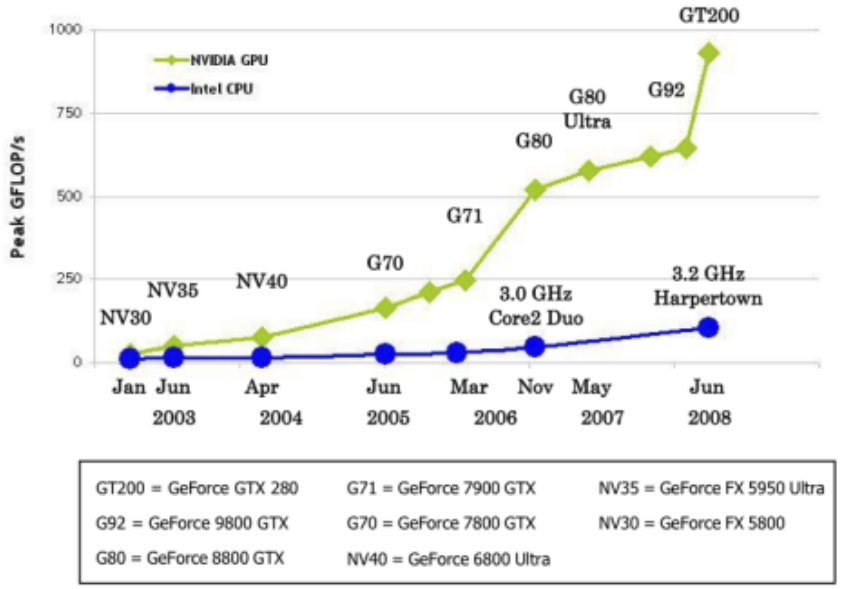


Figure 2.1: Computational power (FLOPS): CPU vs. GPU [Credits: nvidia]

as LDOF [12], DeepFlow [80], EpicFlow [65], DiscreteFlow [54], FlowFields [4] and MirrorFlow [41] came up with a significant number of novel strategies.

Innovations from this period provided outstanding success in addressing the correspondent issue. However, the enhanced precision lengthened the evaluation period. As a result, none of the traditional approaches runs in real-time.

Modern traditional techniques have significant computational costs, which makes their use in other applications difficult. In being realistic, there was a requirement of a unique and extraordinarily different approach than the contemporary methods to be incorporated. Meanwhile, easy access to computationally powerful Graphic Processing Units (GPUs) drew researchers' attention to focus on deep learning side.

Graphics Processing Units (GPU) were initially designed to compute graphics calculus. Computer graphics use a lot of linear algebra which is known to be easily parallelizable. As a consequence, GPUs evolved to many-core architectures dedicated to very specific and repetitive tasks. In the meantime traditional CPUs took also advantage of parallelism but evolved to multi-core architectures (less cores than in many-core). The Figure 2.1 shows the evolution of the computational power (in FLOPS: floating point operations per second) of CPUs and GPUs over the past years. Also in the Fig.2.1, it's possible to see that the computational power of the GPUs has become much more higher than the best of the CPUs.

In 2015, FlowNet [22] presented the first ever deep learning scheme based on convolutional neural networks. The scheme seems promising as it runs in real time. However, its efficiency remained below the state-of-the-art traditional methods.

The FlowNet2 [43] architecture was built by stacking several FlowNet [22] modules, which

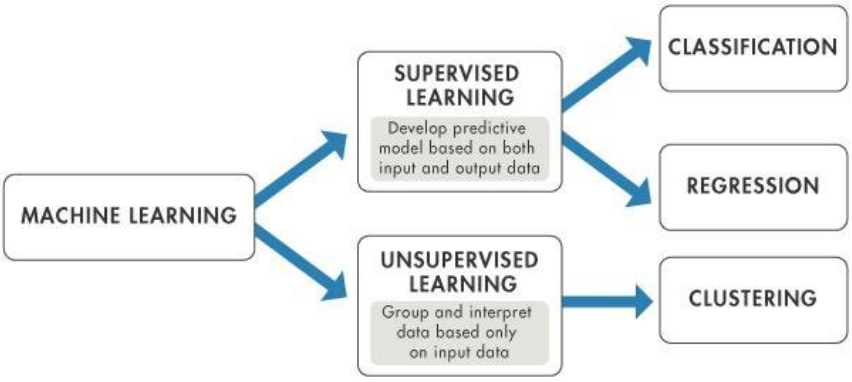


Figure 2.2: Supervised Learning and versus Unsupervised Learning [credit: Mathworks]

outperformed many traditional techniques such as DeepFlow [80] and EpicFlow [65] on MPI-Sintel benchmark [13].

FlowNet2's large model design, on the other hand, necessitates a lot of memory; thus it is not appropriate for mobile and embedded devices. Later efforts concentrated on developing lightweight modules without jeopardising the output's accuracy. This was accomplished by taking many well-known concepts from previous technologies and inserting them into deep learning solutions.

SpyNet [61] combines coarse-to-fine approach of traditional methods and deep learning methods, LiteFlow [40] uses traditional brightness inconsistency map to tackle occlusions, PWC-Net [75] integrates traditional stereo matching, feature extraction and cost volume with deep learning, Continual Flow [60] combines occlusions and cost volume together with optical flow, MFF [63] fuses the warped optical flow from previous frame with the current optical flow estimate, SelFlow [48] uses flow estimation from non-occluded pixels for self-learning, IRR [42] learns to refine its previous estimate by iteratively re-using the same network block with shared weights.

Optical flow methods based on deep learning have already outperformed traditional approaches in accuracy and execution speed. In fact, deep learning techniques are capable of running in real-time with considerably more accuracy. Deep learning's exceptional performance is owing to various concepts and new ideas that have been incorporated into conventional techniques.

2.1.1 Deep Learning based or CNN Methods

Optical flow is computed from a pair of input pictures using machine learning algorithms. In recent years convolutional networks have been used to estimate the optical flow with promising results [22, 60, 53].

Convolutional neural nets can go through supervised or unsupervised training for per-pixel image classification.

Supervised learning is a machine learning technique based on labelled data in the field of artificial intelligence (AI) and machine learning. These datasets are created to train or "supervise"

algorithms to correctly classify or predict data. The model can measure its accuracy and learn over time using labelled inputs and outputs [66].

Unsupervised learning uses machine learning algorithms to analyse and cluster unlabelled data sets. These algorithms discover hidden patterns in data without the need for human intervention (hence, they are "unsupervised") [66].

The above-mentioned tasks are comparable to optical flow estimation in using per-pixel predictions. On the other hand, optical flow needs the network to learn feature representations and match them at various locations across two images. In this sense, it differs from the previous convolutional neural network applications.

Convolutional neural networks with multiple layers can recognise intangible and multiscale elements.

One disadvantage of the deep learning method is its need for a large amount of labelled training data. Researchers have long used synthetic data to train deep learning models, but these data sets do not mimic the photometric qualities of genuine video sequences, which is a significant drawback for deep learning approaches.

Another disadvantage of deep learning techniques is the need for many parameters. As a result of this, the algorithm will store enormous amounts of data in memory and overfit. The excessive memory usage and the millions of parameters may hamper the network's performance and learning.

Supervised methods are a major class of deep learning that outperforms in terms of accuracy and execution time [43, 75] and require labelling for training algorithms.

One of the main challenges of supervised learning is the lack of real-world datasets annotated with ground truth [43]. The existing datasets [5, 13, 31] are too small to allow training.

However, the methods trained on synthetic data will not function well when applied to real-world photometric effects like lighting variances, picture blurring, and other more complicated atmospheric effects.

Given the mentioned challenges, researchers have focused on an unsupervised approach [74], where no labels or weights are given, and the learning algorithm has to find structure in its input.

These techniques are not yet on par with the supervised group. However, this concept is compelling enough and has the potential for an extensive body of research.

2.1.2 Supervised Learning Methods

The first proposed network type incorporates feature extraction and matching [22, 75, 61] in one net. The second performs only one of the two tasks [30, 68]. The most prominent in the first category is FlowNet [22].

FlowNet is a fully convolutional, end-to-end deep neural network that is trained in a supervised manner to generate optical flow using a pair of images.

Optical flow estimation may be seen as a supervised learning problem for the first time, and the use of a convolutional net for optical flow can be considered a breakthrough in the field due to its impact on changing the research paradigm from conventional to deep learning techniques.

The commonly used data sets (Middlebury, KITTI, MPI-Sintel) were not big enough for training neural networks. As a result, the researchers created a synthetic 2D dataset with a random background. At a frame rate of 5 to 10 frames per second, Flownet became competitive in its accuracy.

Following the initial research, the most important work based on FlowNet is SPyNet, a coarse-to-fine knowledge of variational methods by Ranjan and Black [61]. With the SPyNet, fewer model parameters are required and the accuracy is greater than FlowNetC and lower than FlowNetS [22].

FlowNet2 [43] has been improved and now performs at the same level as many state-of-the-art solutions, albeit a little slower than the original FlowNet. By stacking several networks, distorting the second picture with intermediate optical flow, and utilising subnetworks for minor displacement, FlowNet2 was able to reduce the estimation error by more than 50%.

Despite its performance, FlowNet2.0 has some limitations. Its model size is significantly larger (requiring over 160 M parameters) than the original FlowNet and its many modules need to be trained separately to avoid overfitting, all of which take longer computation time than FlowNet. These issues make FlowNet2.0 unsuitable for mobile and other embedded devices.

The present work of Sun [75] combines principles of pyramidal processing, warping, and cost volume with deep learning to produce PWC-Net, 17 times smaller and with better performance than FlowNet2.0. It is thus, the perfect balance between model size and its efficiency.

End-to-end learning, such as those described above, is a type of supervised learning. In this method, two images are given as input, and optical flow is computed by combining feature extraction and matching.

PatchBatch [30] used a Siamese net to compute descriptors for each pixel of the whole image. A sparse flow is generated by the PatchMatch [7] algorithm, which uses the fed descriptors.

Finally, to create a dense motion field, the Edge aware optimisation of EpicFlow [65] is applied to the sparse flow.

In addition, discrete flow [33] utilises Siamese net to learn features, but the optical flow is generated by discrete optimisation.

2.1.3 Unsupervised Learning Methods

The majority of supervised deep learning methods for optical flow in terms of accuracy and efficiency are presently the most effective type of deep learning techniques for optical flow.

However, these systems are extremely fallible and ineffective if there isn't enough proper ground-truth data available. Furthermore, the network may not function effectively in different circumstances of its training. These limitations led researchers to look further into unsupervised methods.

Unsupervised methods are knowledge-driven, and able to train neural nets using unlabelled image pairs to compute optical flow. Their performance isn't yet at the level of supervised methods, but their gradual improvements are leading the research community to look into it further.

The unsupervised network proposed by [2] is based on the classical constraints without regularisation. The optimisation of the error function is differentiable concerning the unknown flow field, and it enables back-propagation of errors to previous layers. The loss function proposed by [86] can learn optical flow in an unsupervised end-to-end manner. It has yet to outperform the original FlowNet [28] in terms of overall accuracy, except for real images of KITTI (where a 100% ground truth is unavailable).

This network [86] was further researched by [41] by introducing an unsupervised loss based on occlusion-aware bidirectional optical flow. To learn bidirectional flow, the model applies an unsupervised loss to FlowNetC [28]. The final product is achieved by combining multiple networks of FlowNet together in an iterative process.

The model developed by [88] learns optical flow with proxy ground truth generated using conventional techniques in an unsupervised manner. The loss is defined by [47] as the photometric error between the warped feature map from the input image and the target image.

The unsupervised neural networks [79] are concerned with occlusion and large displacement. They used the motion-induced occlusion map with the loss function to assess occlusion. The researchers proposed three additions for large displacement: a new warping approach for large motion learning, supplementary warped inputs during the decoder stage, and the use of histogram equalisation and channel representation in computing flow.

2.1.4 Advantages and Challenges of Deep Learning Methods

Deep learning methods rely on quality and quantity of the labelled training dataset. This is a major issue for deep learning schemes because for real scenes it is extremely cumbersome to obtain labelling. The other problem is the hazardous overfitting due to millions of parameters contained by neural nets. This issue along with large memory trail adversely affects the learning efficiency.

An important aspect is the balance between accuracy and size of deep learning architectures. The design of network is a basic factor behind its efficiency. FlowNet2 [43] stacked multiple modules of FlowNet. This improved accuracy but also demanded more than 150 million parameters. Large networks tend to consume high memory due to the extended number of parameters used. On practical grounds it is not very functional. SPyNet [61] is comparatively low in both ways. PWC-Net [75] exhibited the best balance between accuracy and size. Many follow up researchers adopted PWC-Net with improved performance [82, 42]. Another important aspect that affects the performance of deep learning methods is the kind of image sequences being used. Traditional and modern researchers have used different datasets. Middlebury [5], Sintel [13], KITTI [31] and Flying Chairs [28] are the most commonly used datasets. Each set has a unique challenge for creating the most accurate system. The Middlebury dataset has only eight frame pairs with ground truth data. The KITTI dataset has around fifty times as many samples as that of Middlebury, with

ground truth data, but the images are not labelled densely. MPI-Sintel datasets provided over 1,041 training samples with ground truth data. However, training deep learning models requires several thousands of parameters and a large number of marked samples. All major datasets are still relatively small to be used in a deep learning environment. Further, the small datasets often lead to an increase in the direct training difficulty and over-fitting, which results in reduced accuracy. Due to the reasons above, the Flying Chairs [28] and Freiburg [52] datasets were designed specifically to be used in deep learning schemes.

Both learning based and traditional methods are compared by their relative advantages and disadvantages. Firstly, deep learning schemes are more efficient in extracting images features to be used for optical flow estimation. This is because of the multi-layer architectures of these methods that allows them to extract more abstract, deeper, and multiscale features. Secondly, these methods avoid the disadvantages of hazardous and complex optimisation of the traditional schemes. In credit to the introduction of stochastic minimisation of loss function, deep learning methods can better model the intricate, non-linear transformations of the input images [79]. A very important advantage of deep learning schemes is their running speed. Generally, these methods run in real time. On the other hand, traditional methods with similar accuracy take a much longer time. This makes them impractical for mobile and other embedded devices.

The quality and quantity of the training dataset are fundamental for the deep learning methods. This is a significant concern for deep learning systems since it's difficult to get real-scene labelling. Furthermore, there may be significant overfitting due to the millions of parameters in neural networks. Both these concerns significantly affect learning efficiency.

As a result, the balance between accuracy and the size of deep learning algorithms is crucial, and this can be better accounted for with the design of the network.

FlowNet2 [43] stacked multiple modules of FlowNet, and the accuracy was improved, but it demanded more than 150 million parameters. Due to the larger number of parameters utilised, large networks consume a lot of memory. Practically, this is not very applicable. SPyNet is comparatively low in both measures. PWC-Net was responsible for the best balance between accuracy and size. Many of the subsequent researchers implemented PWC-Net with improved performance.

Besides other factors, the image sequences used also affect the performance of deep learning methods. Both traditional and modern researchers have used different datasets, but Middlebury [5], Sintel [13], KITTI [31] and Flying Chairs [28] are the most commonly used. Each set has a distinct problem that needs to be addressed in order to build the most accurate system.

Starting with the Middlebury dataset, it has only eight frame pairs with ground truth data. The KITTI dataset has around fifty times as many samples as that of Middlebury, with ground truth data, but the images are not labelled densely. MPI-Sintel datasets provided over 1,041 training samples with ground truth data.

Given their differences, all major datasets are still small to be used in a deep learning environment. They add several more challenges when it comes to training and over-fitting, resulting in

reduced accuracy. The Flying Chairs [28] and Freiburg [52] datasets were created specifically for use in deep learning algorithms.

Further, it is necessary to evaluate traditional and learning-based learning methods by looking at their benefits and drawbacks.

To extract image features for optical flow estimation, deep learning algorithms are more efficient. This is due to the multi-layer construction of such methods, which enables them to extract more abstract, deeper and multiscale features.

Besides their increasing ability to model the intricate, non-linear transformations of the input images [79], deep learning methods have a faster-running speed and run in real-time. This is not the case with traditional methods that take a much longer period to run for the same levels of accuracy, making them unsuitable for mobile and other embedded devices.

Despite the mentioned advantages, deep learning methods also have several drawbacks. The quality and size of the labelled dataset are two significant variables influencing their performance. This happens because the parameters of convolutional networks are learned from training data, and it is challenging to obtain dense ground truth labelling for real image sequences [28] [52]. When a computer-generated image replaces a real-world scene, the natural gap between artificial data and real-world settings is always there, and algorithms trained on synthetic data have problems with more generalised and complex real-world image sequences.

Convolutional networks contain millions of parameters. This aspect, along with the difficulty of obtaining suitable training data, adds the risk of overfitting to the disadvantages of deep learning algorithms. Deep learning methods, by their nature, need a large memory footprint given their dependency on a high number of parameters.

2.2 Mobile Robot Localisation Strategies

Recently, simultaneous localisation and mapping (SLAM) techniques for localisation using a mono camera have been proposed [56, 26].

However, in low illumination condition, the amount of light reaching image sensors decreases. The feature extraction and matching performance deteriorate, and conventional vision based localisation methods are difficult to be operated [85].

Furthermore, the localisation error is significantly higher when conventional wheel encoders are utilised in a slippery scenario [18, 62]. Typically, slippage that occurs on carpets, rugs, and thresholds is hard to recognise reliably by inertial sensors.

New approaches focus on systems for robot localisation using sensor fusion in a low illumination and slippery environment by combining low-cost motion sensors, like, an afocal optical flow sensor (AOFS), and a forward-viewing camera [85].

In a more holistic approach to mobile robot localisation, it's important to reference the multi-model localisation system based on ceiling's landmarks, more specifically lights [20]. Experimental results and analysis are presented in [15].

Ever more robust, accurate and detailed mapping using visual sensing has proven to be an enabling factor for mobile robots across a wide variety of applications.

For the next level of robot intelligence and intuitive user interaction, maps need extend beyond geometry and appearance — they need to contain semantics.

This challenge is addressed by combining Convolutional Neural Networks (CNNs) and a state of the art dense Simultaneous Localisation and Mapping (SLAM) system, Elastic Fusion, which provides long-term dense correspondence between frames of indoor RGB-D video even during loopy scanning trajectories. These correspondences allow the CNN's semantic predictions from multiple view points to be probabilistically fused into a map.

This not only produces a useful semantic 3D map, but it also shows on the NYUv2 dataset that fusing multiple predictions leads to an improvement even in the 2D semantic labelling over baseline single frame predictions.

It also shown that for a smaller reconstruction dataset with larger variation in prediction viewpoint, the improvement over single frame segmentation increases. The system is efficient enough to allow real-time interactive use at frame-rates of $\approx 25Hz$.

2.2.1 Visual SLAM

Most visual SLAM systems work by tracking set points through successive camera frames to triangulate their 3D position, while simultaneously using this information to approximate camera pose. Basically, the goal of these systems is to map their surroundings in relation to their own location for the purposes of navigation.

This is possible with a single 3D vision camera, unlike other forms of SLAM technology. As long as there are a sufficient number of points being tracked through each frame, both the orientation of the sensor and the structure of the surrounding physical environment can be rapidly understood.

All visual SLAM systems are constantly working to minimise re-projection error, or the difference between the projected and actual points, usually through an algorithmic solution called bundle adjustment. Visual SLAM systems need to operate in real-time, so often location data and mapping data undergo bundle adjustment separately, but simultaneously, to facilitate faster processing speeds before they're ultimately merged.

Many robotics applications such as navigation and mapping require accurate and drift-free pose estimates of a moving camera. Previous solutions favour approaches based on visual features in combination with bundle adjustment or pose graph optimisation [36, 27].

Although these methods are state-of-the-art, the process of selecting relevant key points discards substantial parts of the acquired image data. Therefore, the goal should be to develop a dense SLAM method that (1) better exploits the available data acquired by the sensor, (2) still runs in real-time, (3) effectively eliminates drift, and corrects accumulated errors by global map optimisation

As mentioned previously, vision is extensive field of research in robot navigation due to the great amount of information available in vision. So, there has been an upsurge in interest in visual-based SLAM, often known as VSLAM, owing to the visual information accessible from passive, low-cost video sensors compared to LASER scanners.

However, the trade-off is a higher computational cost and the requirement for more sophisticated algorithms for processing the images and extracting the necessary information. One of these algorithms is optical flow or motion estimation.

Due to recent advances in CPU and GPU technologies, the real-time implementation of the required complex algorithms is no longer an insurmountable problem. Indeed, a variety of solutions using different visual sensors, including monocular [19], stereo [50], Omni-directional [45], time of flight (TOF) [71] and combined colour and depth (RGB-D) cameras [35], have been proposed.

2.2.2 Sensor Fusion

Afocal optical flow sensor (AOFS) and OFS-based odometry is advantageous for mobile robots because it is not affected by wheel slippage [83].

With this configuration, a system for robot localisation can be proposed in a low illumination and slippery environment by combining low-cost motion sensors, an afocal optical flow sensor (AOFS), and a forward-viewing camera [83].

Unlike some previous work, AOFS for robot localisation is adopted in a slippery environment. Also, the interior space structure from an image and robot orientation is estimated. Instead of the conventional point feature extraction and matching for localisation, the interior structure of environment from an image for robot orientation estimation is computed. [83]

In order to achieve this, a rolling guidance filter is applied to eliminate image noise and enhance the appearance of image boundary after applying histogram equalisation to brighten the image. Then, the vanishing point is extracted to figure out the dominant orientation of interior space [83].

2.2.3 Ceiling's Landmarks Based Navigation

Ceiling's landmarks based navigation has the advantage of dealing with less change in image sequences in navigation [57].

For experimental validation, navigation was performed considering the following aspects: i) via way points between two locations with wrong initial estimates; ii) along a large trajectory, describing two laps in a closed loop.

For the work [15], the experimental results showed that the proposed approach is able to stabilise the estimations and to move the robot toward its destination, along a clearance path, providing estimates close to the real trajectory. With this system for navigation, the mobile robot can navigate, localising itself only with the information provided from motion sensors installed on-board and the comparison of ceiling depth images with a database previous collected, through the PCA algorithm.

In addition, when closed loop trajectories are considered, the robot can know that it returned to the same place, without the need to recognise specific features from the environment that was previous viewed [15].

2.2.4 Fuzzy Controller Optical Flow Navigation System

Fuzzy controllers based on optical flow are also another way to enable autonomous robot navigation. In the work [59], it is presented a mobile robot visual navigation system for indoor environments based on fuzzy logic controllers (FLC) and optical flow (OF). Two Takagi–Sugeno fuzzy logic controllers for obstacle avoidance and goal-seeking utilising video acquisition and image processing technology are incorporated into the proposed control system.

The OF values calculated by the Horn–Schunck algorithm is used to detect and estimate the positions of obstacles in the first steering controller. The image is divided into two parts to extract information about the environment. The second FLC is used to steer the robot toward the final destination.

In the simulation, the proposed method is tested using the Visual Reality Toolbox. The visual-based control system, when simulated, shows that autonomous navigation without colliding with objects is feasible.

Chapter 3

Background

This chapter covers the theoretical foundation of this work. This study is required to better understand and evaluate the technical side of the project as well as to provide a more comprehensive framework for the State of the Art section.

3.1 Optical Flow

Optical Flow (OF) estimation has always been a core topic in computer vision. It is widely used in segmentation, 3D reconstruction and navigation. Historically, it was first studied in the context of neuroscience to understand motion perception in insects and mammals [47]. In computer science, optical flow is a displacement of pixel values in the image sequence induced by a movement of a camera or a scene observed by it.

Let $I(x,y,t)$ be an image function of the pixel position (x,y) and time t. The optical flow between two frames captured at times t and $t+\Delta t$ can then be represented by the displacement (u,v) and time difference Δt. The time difference Δt will be assumed as with value 1.

In figures 3.1a and 3.1b, there are two images taken sequentially. In the figure 3.1c, there is the optical flow result from the two images. Not all vectors are shown in the images since it could fill all the image, thus, only considerable displacements are represented with arrows. This is the approach for all the images in this body of work.

The most common optical flow approaches are based on a brightness constancy restriction (colour constancy). Colour constancy is the concept that a point in a reference frame should retain its colour when it is transplanted to a target frame.

Colour constancy alone is not reliable, since there are typically many locations in a target frame that will closely match a reference pixel's brightness (or colour). Indeed, the best match may not necessarily be the correct one. In [12], it's pointed out that a nearest neighbour algorithm will satisfy the colour constancy constraint admirably, but will produce completely useless flow vectors.

(a) Image at t (b) Image at $t+1$ (c) Optical Flow Result

Figure 3.1: Optical Flow Application for the images at t and $t+1$

If a point is not occluded in the target frame, it is expected the colour to approximately match the reference point. However, this goal often fails due to defects such as lighting changes and specular highlights.

There is a need to rely on some form of regularisation to choose flow vectors that exhibit spatial or temporal smoothness while simultaneously balancing the colour constancy constraint.

Mathematically, colour constancy for a certain brightness level is defined as:

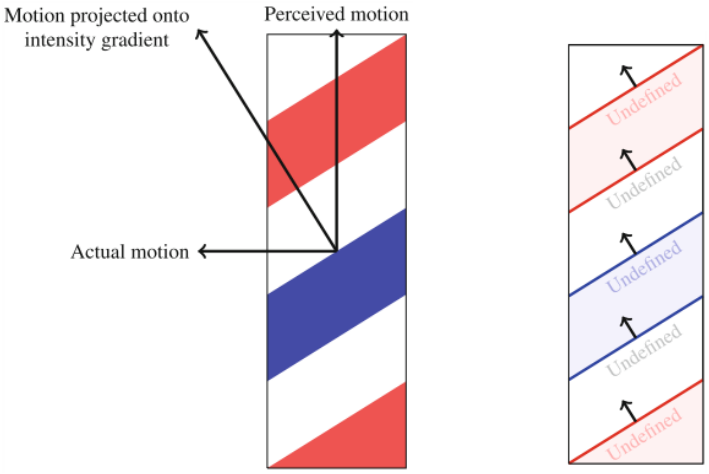
$$I(x,y,t) = I(x+u, y+v, t+1) \tag{3.1}$$

where $I(x,y,t)$ is the intensity of an image for a given location in space and time. Here u,v represent a horizontal and vertical motion again at a specific pixel in space and time, although for simplicity it is not written $u(x,y,t)$ or $v(x,y,t)$. $t+\Delta t$ is equal to $t+1$, where $\Delta t = 1$.

Here u,v represent a horizontal and vertical motion again at a specific pixel in space and time although, for simplicity, it is not written as u(x,y,t) or v(x,y,t). The gradient of the image is often used to supplement colour constancy. Edges are less sensitive to lighting changes. Sometimes a structure/texture separation is performed as a pre-processing step. The two are then blended back together with a strong emphasis on texture.

This captures some of the benefits of colour and gradient constancy with less computation in the optimisation stage. The majority of the algorithms use a single brightness channel for colour constancy. Using multiple colour channels does show some improvement [37], but somewhat surprisingly, the improvement is small. Colour channels are highly correlated and the colour constancy defects, previously mentioned, are likely to be present in all the channels. Colour constancy errors are not Gaussian, i.e. small variances in the image or flow produce corresponding small changes in the colour constancy error. The L^1 or linear penalty norm is more tolerant of outliers and is often called a robust penalty. Upgrading the colour constancy to an L^1 norm, which is non-smooth, complicates the optimisation, but offers dramatically better results.

Figure 3.2b shows an example of colour constancy, only informing of the magnitude flow perpendicular to an image gradient. The magnitude of flow along the image gradient is unknown. The homogeneous regions must be filled using some assumption about the flow. One popular choice is the assumption that the flow is generally smooth, except at the edges in the image.



(a) (b)

Figure 3.2: Colour constancy and aperture effect limitations. The aperture effect in [**?**] makes any of the proposed directions viable, [**?**] illustrates that even with a flow direction colour constancy only yields flow at the edges requiring some kind of smoothing to fill the undefined majority [Credit [32]]

The aperture problem refers to the ambiguity that results from viewing a homogeneous edge of a larger object through a smaller view port or aperture. If the edge is observed to be moving, it is possible to know the magnitude of the motion perpendicular to the edge, but it is not possible to know the magnitude of the motion perpendicular to the edge. Figure 3.2a shows a typical example of the aperture effect. The beginning of the aperture problem can be seen in Eq.3.1, which is under constrained with two variables for each equation.

The constraint of colour constancy is highly non-convex. Non-convex optimisation is hard since there is a presence of potentially many local minima, saddle points, very flat regions and widely varying curvature. In this context, it means that the colour constancy error does not increase as one gets further and further away from the ideal flow vector. It may, in fact, get much better, close to a false match. The decisions made to address this problem greatly impact the optical flow algorithm's characteristics. A common approach to the colour constancy constraint is to take a first-order Taylor series approximation of the colour constancy constraint:

$$u\frac{\partial I}{\partial x}+v\frac{\partial I}{\partial y}+\frac{\partial I}{\partial t}=0 \tag{3.2}$$

I_x, I_y and I_t can be also used to describe the derivatives $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ respectably.

This formulation is convex and amenable to computationally efficient solutions. However, this approximation is only valid for small motions of less than one pixel. To make this useful, a coarse-to-fine image pyramid is made with processing starting on the coarse scale. Now, a small relative motion can correspond to a large motion in the original image. The flow vectors are propagated to the next finer stage. Additionally, the target image is warped by the intermediate flow result such that the vectors that are being solved for are the small difference between the warped image and

the reference image.

This coarse-to-fine approach breaks when there is a large motion by small objects. This large motion can be recorded only at the coarse-scale in the linearised model. If an object or structure is too tiny to be seen at the coarse-scale, its movement will not be recorded and the finer scales will have no means of determining a new displacement.

Another approach is to avoid the linearisation of the color constancy constraint. The usual strategy is to solve (approximately) the Euler-Lagrange equations of the objective function. Since the L1 norm is not differentiable, it is replaced with a differentiable approximation, for example, by $\Psi\left(s^2\right) = \sqrt{s^2 + \varepsilon^2}$, where ε is a small fixed constant to avoid numerical problems. A linear approximation of the nonlinear Euler-Lagrange equations is created. This system of linear equations can be solved using SOR or a similar solver. To try to avoid all the local minimums that exist, this is embedded in some coarse-to-fine framework. While this sounds very similar to the previous linearisation approach it is subtly different in that the linearisation occurs while solving the original nonconvex problem Eq.3.1 The previous approach instead solves a lineasized version of the original problem.

In the first case, the Eq.3.2 is solved, but it is not the original problem. In the nonconvex case, a solution is approximated to the original problem Eq.3.1. Since it is non-convex, one does not know if they have found a global solution or indeed if there is a unique optimal solution. A multi-grid method is often used which moves from the coarse scale to the fine and back up to the coarse scale repeatedly in an effort to avoid falling into local minima.

At present, there are two major approaches for estimating optical flow. First, is the traditional approach that implies handcraft feature evaluation schemes into the main framework; second, the convolutional neural network approach based on deep learning principles.

Traditional methods dominated the field of optical flow for almost four decades. Traditional methods can further be divided into the following classes:

Pixel based [53], feature based [12] and energy based classes [38], however, there are no explicit limits to separate one class from another.

Among them, the most successful and widely used techniques are variational methods. These schemes estimate optical flow by minimising an energy functional derived on the basis of brightness constancy and smoothness assumptions [38]. Variational methods can further be divided into global and local categories.

Global techniques are based on brightness constancy and smoothness assumptions; this approach builds an energy functional whose minimising scheme yields the flow field. In 1981, Horn and Schunck [38] solved the under determined aperture problem by providing additional smoothness constraints. Horn and Schunck's approach was followed by many researchers; however, the algorithm faces difficulties in many practical situations such as large displacements and varying illumination.

The local (total least square) technique relies on the assumption that the essential flow is constant within a small area "R" consisting of "nxn" pixels. The flow constraint is evaluated at all pixels within the neighbourhood window, and the resulting equations are solved using the least

square method. The objective energy to be minimised is the weighted sum of the potentials provided by each pixel of R.

In relation to the traditional approaches to Optical Flow, the three methods that are worth mentioning are the Lucas-Kanade [49], Horn-Schunck [38] method (variational methods) and the Block-Matching algorithm, which is a pixel based method for locating matching macroblocks in a sequence of digital video frames for the purposes of motion estimation.

3.1.1 Lucas-Kanade

The Lucas-Kanade method [49] is based on the principle that for every pixel's (p), small neighbourhood W maintains a constant optical flow (u,v). The optical flow constraint (E 3.3) is then applied to all pixels within the given window W, yielding an over-determined set of equations. The flow is estimated by minimising the sum of deviations using the least-squares:

$$\min_{u,v} \sum_{p \in W(u,v)} [I_x(p)u + I_y(p)v + I_t(p)]^2 \tag{3.3}$$

The Lucas-Kanade algorithm estimates flow for a given set of pixels (ideally corners and textured patches detected by a feature tracker), its result is therefore a sparse optical flow.

The Lucas-Kanade algorithm makes some implicit assumptions:

1. For each pixel, assume Motion Field, and hence Optical Flow (u,v), is constant within a small neighbourhood, W
2. The images depict a natural scene containing textured objects exhibiting shades of grey (different intensity levels) which change smoothly
3. The method is valid only for small displacements

The algorithm does not use color information in an explicit way. It does not scan the second image looking for a match for a given pixel. It works by trying to guess in which direction an object has moved so that local changes in intensity can be explained.

Going back to the Eq.3.3, another way to interpret the set of equations is to write in the following matrix form, where for all points $(k,l) \in W : I_x(k,l)u + I_y(k,l)v + I_t(k,l) = 0$ and the size of window W is $n \times n$:

$$\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ I_x(k,l) & I_y(k,l) \\ \vdots & \vdots \\ I_x(n,n) & I_y(n,n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(1,1) \\ I_t(k,l) \\ \vdots \\ I_t(n,n) \end{bmatrix} \tag{3.4}$$

The first matrix A is a known matrix with dimension $n^2 \times 2$, u is the unknown matrix (vectors u and v) with dimensions 2×1, and B is a known matrix with dimension $n^2 \times 1$.

With a number of n^2 equations and two unknowns (u and v), this system is possible to compute with the Least Squares solution. It represents the solution to the equation $A\mathbf{u} = B$ as closely as possible, in the sense that the sum of the squares of the difference $B\,A\mathbf{u}$ is minimised. For this, the linear system needs to be solved:

$$\begin{aligned} A\mathbf{u} &= B \\ A^T A\mathbf{u} &= A^T B \end{aligned} \tag{3.5}$$

This representation shows the Least-Squares solution using Pseudo-Inverse matrices. In matrix forms:

$$\begin{bmatrix} \sum_W I_x I_x & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum_W I_x I_t \\ -\sum_W I_y I_t \end{bmatrix} \tag{3.6}$$

Indices (k,l) are not written for simplicity. The first matrix $A^T A$ is a known matrix with dimension 2×2, u is the unknown matrix (u and v), and $A^T B$ is a known matrix with dimension 2×1. Rewriting the system:

$$\mathbf{u} = \left(A^T A\right)^{-1} A^T B \tag{3.7}$$

This representation of the system enables a fast and easy way to compute the values of the matrix $\mathbf{u}$. With E.q.3.5, is possible to conclude that the computation is only possible if:

1. $A^T A$ must be invertible. That is $\det\left(A^T A\right) \neq 0$
2. $A^T A$ must be well-conditioned. If λ_1 and λ_2 are eigenvalues of $A^T A$, then

$$\lambda_1 > \varepsilon \text{ and } \lambda_2 > \varepsilon$$

$$\lambda_1 \geq \lambda_2 \text{ but not } \lambda_1 \gg \lambda_2$$

ε denotes a small quantity close to zero. The condition number is a derivative application, and it's defined as the asymptotic worst-case relative change in output for a relative change in input.

The "function" is the solution of a problem and the "arguments" are the data in the problem. The condition number is frequently applied to questions in linear algebra, in which case the derivative is straightforward, but the error could be in many different directions, and is thus computed from the geometry of the matrix. More generally, condition numbers can be defined for non-linear functions in several variables.

A problem with a low condition number is said to be well-conditioned, while a problem with a high condition number is said to be ill-conditioned. In non-mathematical terms, an ill-conditioned problem is one where, for a small change in the inputs (the independent variables) there is a large change in the answer or dependent variable. This means that the correct solution to the equation becomes hard to find. The condition number is a property of the problem. Paired with the problem

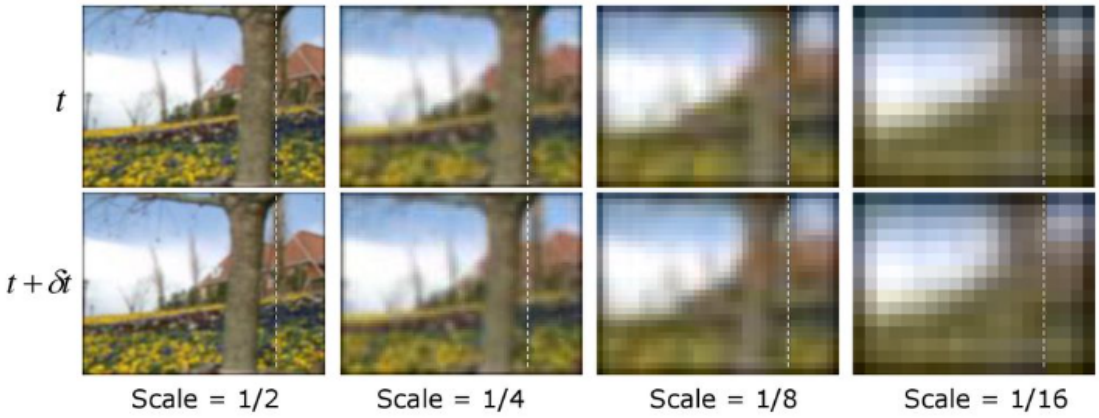


Figure 3.3: Resolution Pyramid [Credits: Columbia University]

are any number of algorithms that can be used to solve the problem, that is, to calculate the solution. Errors in the Lucas-Kanade algorithm happen when:

1. Brightness constancy is not satisfied
2. The motion is not small
3. A point does not move like its neighbours (motion segmentation). The window size can be too large

As mentioned, when the motion is too large, the Taylor series approximation $I(x+\Delta x, y+\Delta y, t+\Delta t)$ is no longer valid. In other words, the linear constraint equation is not valid.

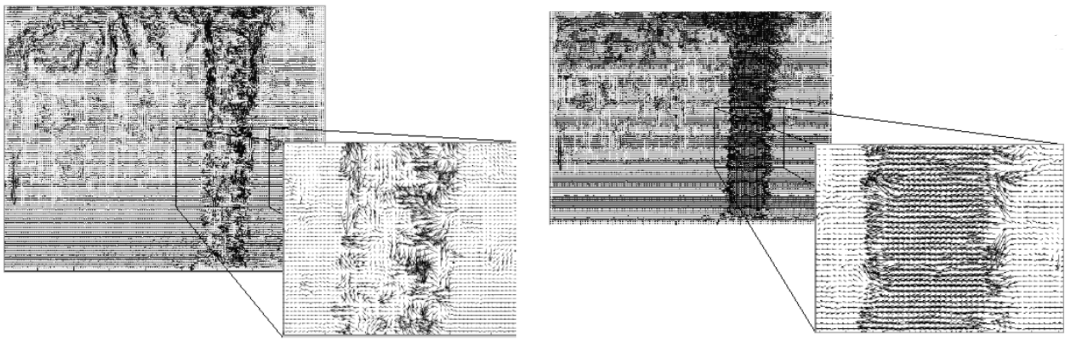
$$I_x u + I_y v + I_t \neq 0 \tag{3.8}$$

In order to compute optical flow, it can be used coarse-to-fine or multi-scale methods [3, 58, 11].

These methods accelerate convergence by allowing global motion features to be detected immediately, but also improve the accuracy of flow estimation because they provide a better approximation of image gradients via warping [14].

An image pyramid is a multi-scale representation of an image, like the one shown in Figure 3.3.

A pyramid of coarse-to-fine down sampled versions of the original image is created by filtering and re-sampling the images at lower resolution. A coarse match is done at the lower level which is used to define a small area with the next higher resolution. The solution is iteratively refined until reaching the full image resolution. Most modern methods choose this strategy for large displacements [73, 61, 39]. Though, earlier researchers had adopted this technique on empirical grounds, the most prominent work was carried out by Thomas Brox [11] who provided a theoretical ground to integrate the variational methods with coarse-to-fine.



(a) Lucas Kanade without Pyramids (b) Lucas Kanade with Pyramids

Figure 3.4: Visual Representation of Improvemments of Lucas Kanade with Pyramids [credit: From Khurram Hassan-Shafique CAP5415 Computer Vision 2003]

The coarse to fine strategies enormously improved the performance (Fig.3.4a and Fig.3.4b). However, they do possess intrinsic limitations. For instance, they may lead to a solution trapped into local minima. Secondly the objects, whose extent are smaller than their respective displacements, may be lost at coarser levels due to smoothing processes.

A third weakness is error-propagation. At coarser levels, different motion layers can overlap, and may propagate across scales. A prominent alternative to coarse to fine schemes is discrete optimisation [54] as is adopted by many stereo matching methods. However, optical flow requires full data cost volume while stereo matching does not require the image pyramid scheme because it is 1D problem. The optical flow estimation being 2D involves extremely large size of the label space, making the estimation process difficult with discrete optimisation.

3.1.2 Horn-Schmuck

The Horn-Schmuck algorithm assumes global smoothness of the optical flow field to solve the aperture problem. The flow is formulated as a global energy function which is then sought to be minimised. This function is given for two-dimensional image streams as:

$$E = \iint \left[(I_x u + I_y v + I_t)^2 + \alpha^2 \left(\|\nabla u\|^2 + \|\nabla v\|^2 \right) \right] \mathrm{d}x \, \mathrm{d}y \tag{3.9}$$

where I_x, I_y and I_t are the derivatives of the image intensity values along the x, y and time dimensions respectively, $\vec{V} = [u(x,y), v(x,y)]^\top$ is the optical flow vector (which is to be solved for), and the parameter α is a regularisation constant. Larger values of α lead to a smoother flow. This function can be minimised by solving the associated multi-dimensional Euler-Lagrange equations. These equations are:

$$\begin{aligned} \frac{\partial L}{\partial u} - \frac{\partial}{\partial x}\frac{\partial L}{\partial u_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial u_y} = 0 \\ \frac{\partial L}{\partial v} - \frac{\partial}{\partial x}\frac{\partial L}{\partial v_x} - \frac{\partial}{\partial y}\frac{\partial L}{\partial v_y} = 0 \end{aligned} \tag{3.10}$$

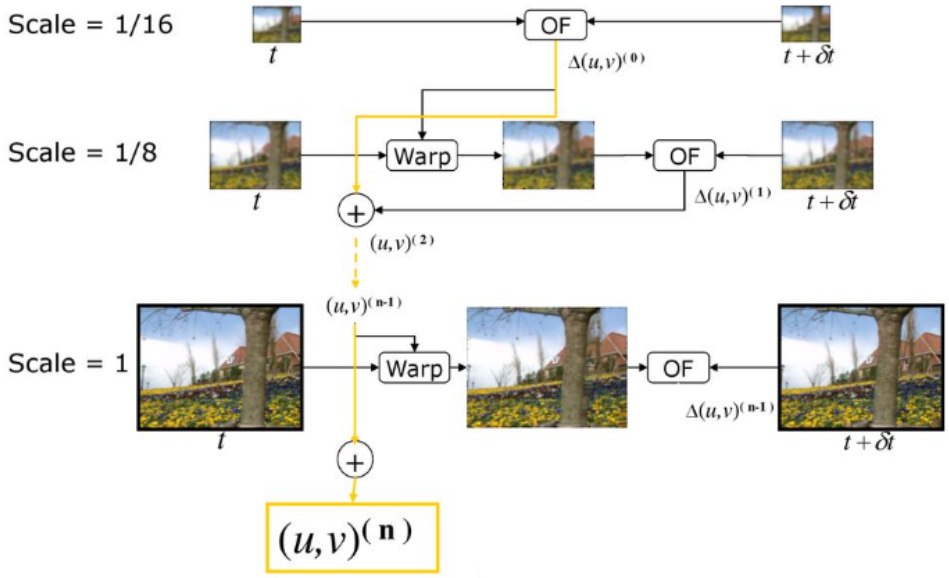


Figure 3.5: Coarse-to-Fine Estimation Algorithm [Credit: Bouget, 2000]

where L is the integrand of the energy expression, giving

$$\begin{aligned} I_x\left(I_x u + I_y v + I_t\right) - \alpha^2 \Delta u = 0 \\ I_y\left(I_x u + I_y v + I_t\right) - \alpha^2 \Delta v = 0 \end{aligned} \tag{3.11}$$

where subscripts again, denote partial differentiation and $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ denotes the Laplace operator. In practice, the Laplacian is approximated numerically, using finite differences, and may be written $\Delta u(x,y) = 4(\bar{u}(x,y) - u(x,y))$ where $\bar{u}(x,y)$ is a weighted average of u calculated in a neighbourhood around the pixel at location (x,y). Using this notation the above equation system may be written

$$\begin{aligned} \left(I_x^2 + 4\alpha^2\right) u + I_x I_y v = 4\alpha^2 \bar{u} - I_x I_t \\ I_x I_y u + \left(I_y^2 + 4\alpha^2\right) v = 4\alpha^2 \bar{v} - I_y I_t \end{aligned} \tag{3.12}$$

which is linear in u and v and may be solved for each pixel in the image. However, since the solution depends on the neighbouring values of the flow field, it must be repeated once the neighbours have been updated. The following iterative scheme is derived using Cramer's rule:

$$\begin{aligned} u^{k+1} = \bar{u}^k - \frac{I_x\left(I_x \bar{u}^k + I_y \bar{v}^k + I_t\right)}{4\alpha^2 + I_x^2 + I_y^2} \\ v^{k+1} = \bar{v}^k - \frac{I_y\left(I_x \bar{u}^k + I_y \bar{v}^k + I_t\right)}{4\alpha^2 + I_x^2 + I_y^2} \end{aligned} \tag{3.13}$$

where the superscript $k+1$ denotes the next iteration, which is to be calculated and k is the last calculated result. This is, in essence, a Matrix splitting method, similar to the Jacobi method, applied to the large sparse system arising when solving for all pixels simultaneously.

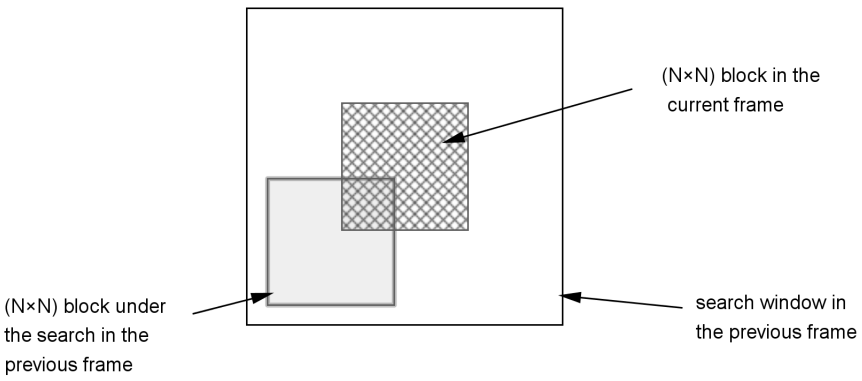


Figure 3.6: Block Matching Algorithm Search Windows ($N \times N$) [Credits: German iris]

Cramer's rule is an explicit formula for the solution of a system of linear equations with as many equations as unknowns, valid whenever the system has a unique solution.

In discrete domain, the function can be written in the following way:

$$\min_{u,v} \sum_{i,j} \{E_s(i,j) + \lambda E_d(i,j)\} \tag{3.14}$$

where $E_s(i,j)$ represents the smoothness factor, $E_d(i,j)$ represents the brightness factor and λ the weight, or regularisation constant.

The advantage of the Horn–Schunck algorithm include that it yields a high density of flow vectors, i.e. the flow information missing in inner parts of homogeneous objects is filled in from the motion boundaries. However, it is more susceptible to noise than local approaches.

3.1.3 Block-Matching

The block matching algorithm is one of the simplest methods to compute the optical flow. For every pixel (x,y) in the original image, the closest match $(x+u, y+v)$ in the subsequent image is found by minimising the Sum of Absolute Differences (SAD). The SAD value is computed by comparing a small (usually square) window $(M \times N)$ around the pixel (Figure3.6):

$$SAD = \sum_{i=-\frac{M}{2}}^{\frac{M}{2}} \sum_{j=-\frac{N}{2}}^{\frac{N}{2}} |I(x+u+i, y+v+j, t+\Delta t) - I(x+i, y+j, t)| \tag{3.15}$$

This method can be accelerated by computing the flow only for a portion of the image pixels' rather than the entire image matrix, leading to a sparse optical flow. Because each pixel's flow may be calculated independently, it is easily parallelised.

Since the mid-1980s, researchers have studied Block Matching algorithms. They have developed many of these algorithms, but some remain more widely used than others. The section below describes only a few of the most basic or commonly employed ones:

1. **Exhaustive Search**: By calculating the cost function at each possible location in the search window, this algorithm leads to the best match of a macro-block between two frames. The signal-to-noise ratio in the compensating image is higher than any other block matching algorithm, but because it requires more computations, it is also slower. A wider search window means even more time will be needed for processing.

2. **Optimised hierarchical block matching (OHBM)** The optimised hierarchical block matching algorithm speeds up the exhaustive search based on the optimised image pyramids.

3. **Three Step Search**: It starts with a search location at the center, it sets the step size S = 4 and searches for the parameter p = 7, it searches for 8 locations +/- S pixels around location (0,0) and the location (0,0), it picks among the 9 locations searched, the one with minimum cost function, it sets the new search origin to the above picked location, it sets the new step size as S = S/2, and repeats the search procedure until S = 1. The resulting location for S=1 is the one with minimum cost function, and the macro block at this location is the best match.

   There is a reduction in computation by a factor of 9 in this algorithm. For p=7, while ES evaluates cost for 225 macro-blocks, TSS evaluates only for 25 macro blocks.

There exists other kinds of block-matching algorithms, like the new Three Step Search and the Two Dimensional Logarithmic Search.

A metric for matching a macroblock with another block is based on a cost function. The most popular in terms of computational expense is the Mean difference or Mean Absolute Difference (MAD):

$$MAD = \frac{1}{N^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left| C_{ij} - R_{ij} \right| \tag{3.16}$$

And the Mean Squared Error as the following equation:

$$MSE = \frac{1}{N^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(C_{ij} - R_{ij} \right)^2 \tag{3.17}$$

where N is the size of the macro-block, and C_{ij} and R_{ij} are the pixels being compared in the current macroblock and reference macroblock, respectively.

These four algorithms for optical flow have been chosen for this study since they are the most common traditional optical flow methods.

3.2 Visual Robot Navigation

The term visual navigation refers to the use of data gathered by visual sensors in order to control motion. Visual navigation is an area of particular relevance because of the rich perceptional input provided by vision. These systems require an accurate representation of the environment and its geometry. The approach for vision-based robot navigation for an unstructured environment, where no prior knowledge of the robot environment exists, is generally based on optical flow calculation [73].

The goal of this work is to create optical flow algorithms and methods for robust visual navigation of autonomous mobile robots. In other words, an analysis of some optical flow techniques and their benefits and drawbacks in regard to mobile robot navigation, as well visual navigation techniques, being the one explored, visual odometry in the context of V-SLAM.

The input of the robot's sensors (cameras) consists of a sequence of images continuously required by the navigation system while the robot is in motion. The image sequence is provided by a monocular vision system (the sensor system only has one camera) for an actual physical robot. This last parameter is a focus of the investigation since many approaches to optical flow navigation techniques require a multiple number of cameras [81], like stereo vision and trinocular vision. The multiple camera approach follows a corridor behaviour, increasing the calculation and execution costs.

The advantages, disadvantages, and different approaches to navigation with multiple cameras will not be studied in this work, as it is more focused on studying optical flow techniques as the primary differentiators in the visual navigation system.

After the optical flow is computed, the robot tries to understand its environment by extracting the crucial features from this image sequence. In this case, the optical flow is the main cue for navigation and the robot uses this information as its guide for motion. One common strategy to avoid navigation obstacles is to balance the right and left optical flow vectors.

The process can be summarised in the following: first, the optical flow vectors are computed from an image sequence. Computing the FOE (focus of expansion) position in the image plane is necessary to decide the robot's orientation as the control law is given concerning the focus of expansion. Then, the depth map is computed to determine whether the obstacle is close enough to warrant immediate attention, is at such a distance as to give an early warning, or is so far away from the robot that the robot can ignore it.

This navigation system is considered an obstacle avoidance system since the main goal is to interpret the optical flow and, with this information, avoid objects. The following sub-sections will study each part of the navigation system in more detail.

However, there is another kind of visual robot navigation, which consists in Visual Simultaneous Localisation and Mapping (SLAM) using visual odometry, which is the method used in the experimental part of this work.

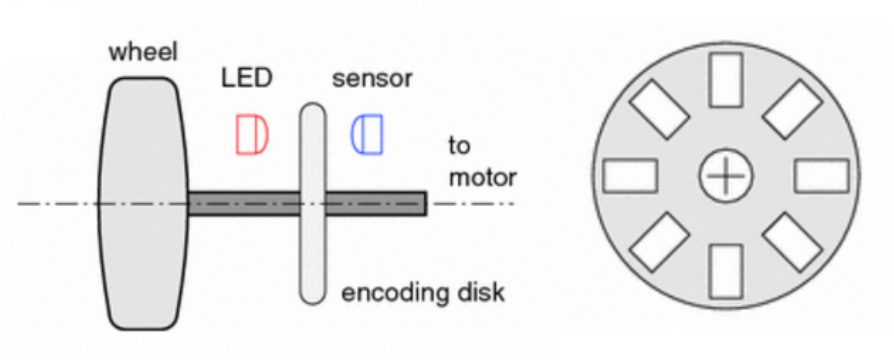


Figure 3.7: Wheel/Encoder Odometry System

3.2.1 Visual Simultaneous Localization and Mapping (VSLAM)

A SLAM method allows a robot to localise itself in an unknown environment while gradually building a map of its surroundings. SLAM is a topic that has been intensely studied over the last decades [29] with various approaches for different sensors, including sonar sensors [46], IR sensors [1] and LASER scanners [17].

As mentioned previously, vision is a very rich field of research in robot navigation due to the great amount of information available in the field. So, there has been an upsurge in interest in visual-based SLAM, often known as VSLAM, owing to the rich visual information accessible from passive, low-cost video sensors when compared to LASER scanners.

Visual Simultaneous Localisation and Mapping, as the name suggests, has two main components: localisation and mapping.

In terms of localisation, Visual Odometry (VO) is the process of estimating the camera's relative motion by analysing a sequence of camera images. Similar to wheel odometry, estimates obtained by VO are associated with errors that accumulate over time [25]. However, VO has been shown to produce localisation estimates that are much more accurate and reliable over longer periods of time when compared to wheel odometry [34]. VO is also not affected by wheel slippage usually caused by uneven terrain.

In order to work with the image information, it's necessary to define the Motion Estimation. This is the process of determining motion vectors that describe the transformation from one 2D image to another; usually from adjacent frames in a video sequence. In general, there are three commonly used VO motion estimation techniques called: 3D to 3D, 3D to 2D and 2D to 2D methods. The 3D to 3D and 3D to 2D Motion Estimation will be analysed.

3D to 3D Motion Estimation: In this approach, the motion is estimated by triangulating 3D feature points observed in a sequence of images. The transformation between the camera frames is then estimated by minimising the 3D Euclidean distance between the corresponding 3D points, as shown below.

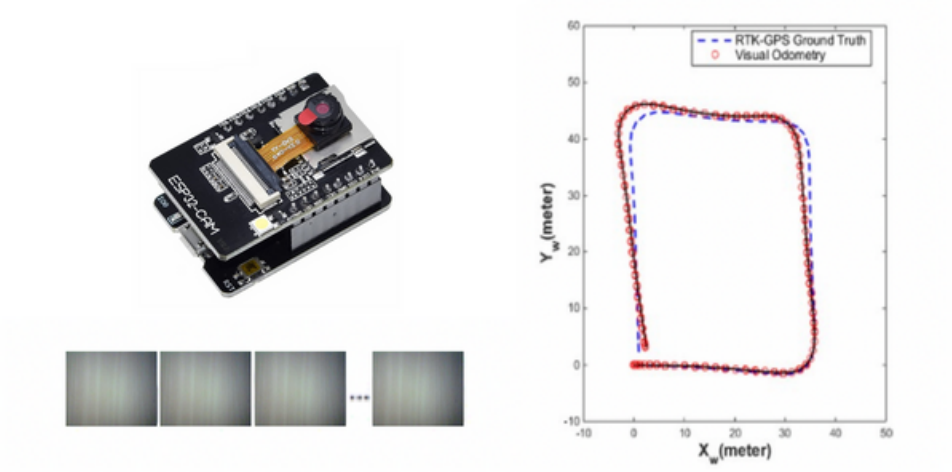


Figure 3.8: Visual (Camera) Odometry System and Result of VO and reference (RTK-GPS Ground Truth) [credit: Aqel et al. 2016]. The input of the system is a image sequence (video stream or sequentially captured photos) and the output is the camera trajectory (translation and rotation)

$$\mathbf{T} = \underset{T}{\operatorname{argmin}} \sum_{i} |\mathbf{X_i} - \mathbf{T}\mathbf{X_i}|^2 \tag{3.18}$$

In the above equation, $\mathbf{T}$ is the estimated transformation between two consecutive frames, $\mathbf{X}$ is the 3D feature point observed by the current frame F_k, $\mathbf{X}$ is the corresponding 3D feature point in the previous frame F_{k-1} and i is the minimum number of feature pairs required to constraint the transformation. The minimum number of required points depends on the system's DOF and the type of modelling used. Although using more points means more computation, better accuracy is achieved by including more points than the minimum number required.

3D to 2D Motion Estimation: This method is similar to the previous approach but here, the 2D re-projection error is minimised to find the required transformation. The cost function for this method is as follows:

$$\mathbf{T} = \underset{T}{\operatorname{argmin}} \sum_{i} \left|\mathbf{z} - f\left(\mathbf{T}, \dot{\mathbf{X}}_\mathbf{i}\right)\right|^2 \tag{3.19}$$

where T is the estimated transformation between two consecutive frames, $\mathbf{z}$ is the observed feature point in the current frame F_k, $f\left(\mathrm{T}, \hat{X}_i\right)$ is the re-projection function of it's corresponding 3D feature point in the previous frame F_{k-1} after applying a transformation $\mathbf{T}$ and i is the number of feature pairs. Again, the minimum number of points required varies based on the number of constraints in the system.

For this case, the 3D motion of the objects, as well as the camera's movement, generates 2D motion on the image plane via a suitable projection mechanism, such as the pinhole camera projection model presented in Fig.3.9.

A point in space $X = [X, Y, Z]^T$ is projected by a pinhole camera (Figure 3.9). The corresponding point on the image plane $p = [x, y, f]^T$ can be computed as

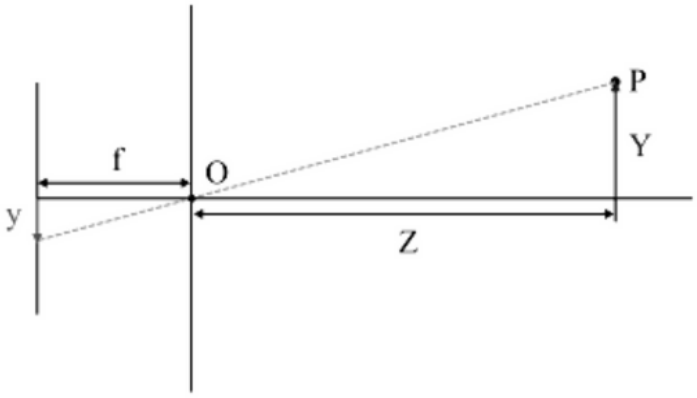


Figure 3.9: Pinhole Camera Projection Model

$$p = -\frac{f}{Z}P \tag{3.20}$$

where f, the distance between the image plane and projection origin O, is the camera's focal length. Since the camera is mounted perpendicularly to a vehicle body, the coordinate Z is equal to the distance between ground and camera's projection origin.

The ground distance Z must be obtained from an external sensor, such as an ultrasonic or laser distance sensor or a barometric pressure sensor. Given the ground distance Z is approximately constant between two consecutive frames, a displacement in the image plane $(\Delta x, \Delta y)$ can be converted to a real word displacement $(\Delta X, \Delta Y)$:

$$\Delta X = -\frac{1}{f}\Delta x \cdot Z, \Delta Y = -\frac{1}{f}\Delta y \cdot Z \tag{3.21}$$

The displacement in the image plane can be obtained using one of the Optical flow algorithms described in optical flow subsection. As the computed displacement (u, v) is usually in pixels, it is required to convert it into real-world units (e.g. meters). Equation 3.21 then changes to:

$$\Delta X = -\frac{s}{f}u \cdot Z, \Delta Y = -\frac{s}{f}v \cdot Z \tag{3.22}$$

where s is the pixel size.

In terms of mapping, in most real-world robotic applications, maps of the environment in which the mobile robot is required to localise and navigate are not available. Therefore, in order to achieve true autonomy, generating a map representation of the environment is one of the important competencies of autonomous robots [77]. In general, mapping the environment is considered to be a challenging task. The most commonly used mapping representations are:

1. Metric Maps: In metric maps, the environment is represented in terms of geometric relations between the objects and a fixed reference frame. The most common forms of metric maps

are Feature Maps and Occupancy Grids. Feature maps [16] represent the environment in a form of sparse geometric shapes such as points and straight lines. Each feature is described by a set of parameters such as its location and geometric shape. Localisation in such environments is performed by observing and detecting features and comparing those with the map features that have already been stored. Occupancy Grid maps [10] are represented by an array of cells in which each cell (or pixel in an image), represents a region of the environment. Unlike feature maps which are concerned with the geometric shape or type of the object, occupancy grids are only concerned about the occupancy probability of each cell.

2. Topological Maps: In contrast to metric maps which are concerned about the geometric relations between places or landmarks, topological maps are only concerned about adjacency information between objects [23] and avoid metric information as far as possible [57]. Topological maps are usually represented by a graph in which nodes define places or landmarks and contain distinctive information about them and connecting arcs that manifest adjacency information between the connected nodes. Topological maps are particularly useful in representing a large environment in an abstract form where only necessary information are held. This information includes high level features such as objects, doors, humans and other semantic representation of the environments.

SLAM is an attempt to solve both of those problems at the same time. SLAM approaches have been categorised into filtering approaches (such as EKF-SLAM [72] and particle filter based SLAM [77] and smoothing approaches (such as GraphSLAM [78], RGBD SLAM [35], Smoothing and Mapping. Filtering approaches are concerned with solving the online SLAM problem in which only the current robot state and the map are estimated by incorporating sensor measurements as they become available. Whereas smoothing approaches address the full SLAM problem in which the posterior is estimated over the entire path along with the map, and is generally solved using a least square (LS) error minimisation technique [7].

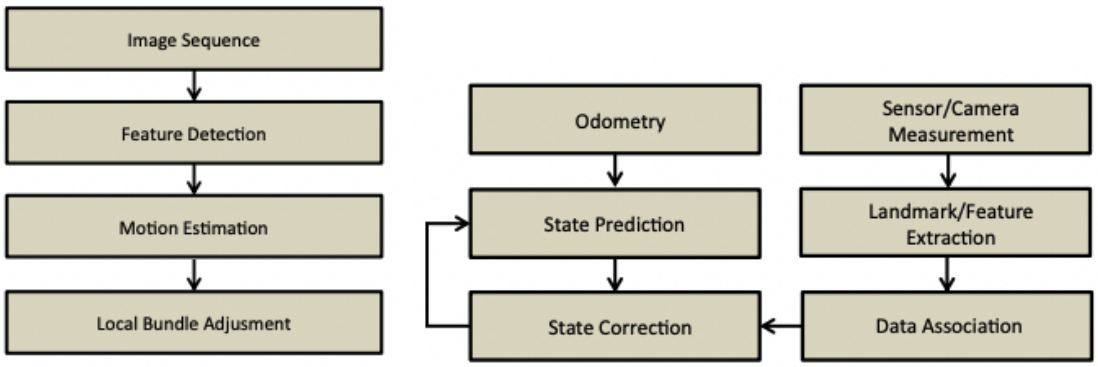
The main components of Visual SLAM are shown in Figure 3.10 and a resulting map in Figure 3.10b.

3.2.2 Visual Odometry

Visual Odometry (VO) was mentioned in the V-SLAM system as the main component of the localisation part of it. Introducing again, the concept of odometry, it estimates the robot's position and orientation by monitoring the robot's drive [9]. For most drives, this information comes out-of-the-box and is updated at a high frequency. Hence, odometry can be seen as a localisation method that comes for "free".

Odometry computes changes in pose, which need to be integrated in order to obtain the robot's pose estimation (Figure 3.11). However, one disadvantage of such relative localisation methods is the increasing pose error over time.

VO is the process of estimating the camera's relative motion by analysing a sequence of camera images. Similar to wheel odometry, estimates obtained by VO are associated with errors that



(a) Visual Odometry System (b) Filter based SLAM System

Figure 3.10: A block diagram showing the main components of a Visual Odometry 3.10a and a Visual SLAM 3.10b, respectibly [credit: Springer]

accumulate over time [24]. However, VO has been shown to produce localisation estimates that are much more accurate and reliable over longer periods of time, compared to wheel odometry [34]. VO is also not affected by wheel slippage usually caused by uneven terrain.

VO is defined as the process of estimating the robot's motion (translation and rotation with respect to a reference frame) by observing a sequence of images of its environment. VO is a particular case of a technique known as Structure From Motion (SFM) that tackles the problem of 3D reconstruction of both the structure of the environment and camera poses from sequentially ordered or unordered image sets [67].

SFM's final refinement and global optimisation step of both the camera poses and the structure is computationally expensive and usually performed off-line. However, the estimation of the camera poses in VO is required to be conducted in real-time. In recent years, many VO methods have been proposed which those can be divided into monocular [14] and stereo camera methods [51].

These methods are then further divided into feature matching (matching features over a number of frames) [76], feature tracking [21] (matching features in adjacent frames) and optical flow techniques [87] (based on the intensity of all pixels or specific regions in sequential images).

The method used in this study is optical flow, as mentioned previously. With the 2D displacements (u,v) for every pixel, the 3D camera motion can be fully recovered. It was described an equation for retrieving the 6DOF motion parameters of a camera which consist of three translational (T_x,T_y,T_z) and three rotational components $(\Omega_x,\Omega_y,\Omega_z)$ [44]. Their approach is based on solving the following equations:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{-f_c T_x + x T_z}{Z} + \frac{xy}{f_c}\Omega_x - \left(\frac{x^2}{f_c} + f_c\right)\Omega_y + y\Omega_z \\ \frac{-f_c T_y + x T_z}{Z} - \frac{xy}{f_c}\Omega_y + \left(\frac{y^2}{f_c} + f_c\right)\Omega_x + x\Omega_z \end{bmatrix} \tag{3.23}$$

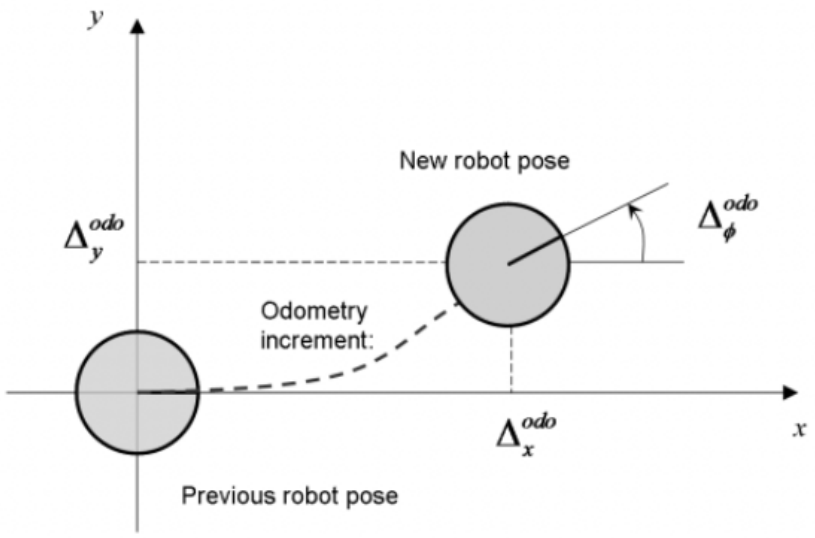


Figure 3.11: Robot change in position and odometry increment

where f_c is the camera's focal length and (x,y) are the image coordinates of the 3D point (X,Y,Z).

Assuming that the depth Z is known, there are six unknowns and a minimum of three points are required to fully constrain the transformation. However, in many situations, additional constraints are imposed such as moving on a flat plane, therefore reducing both the of DOF and the minimum number of points required to constrain the transformation. A visual odometry system can be represented in 3.7.

3.2.3 Sensor Fusion

Instead of only relying on the optical flow computation or sensor, the encoder data will be used in order to get a more accurate estimation, even though the dead-reckoning problem exists.

Commonly, open-loop estimation (dead-reckoning) is used for intermediate estimation of position during path execution. Open-loop estimation is used because the encoders are available for motor control, which provides actual angular displacement of the wheels. However, due to errors in kinematic model parameters or wheel slip, poor position estimates may occur. Poor estimates in position during path execution require more frequent localisation's to be made, incurring extra overhead and possibly slowing the movement of the robot. It is therefore, important to minimise positional errors during the path execution phase. In cases where wheel-slip occurs, open-loop estimation methods usually fail.

The extended Kalman filter is one of the most frequently used methods of estimation using multiple measurements. The optical flow sensor, in particular, needs a vertical distance of 2.5 mm from the surface that is quite rigorous. The distance varies depending on the surface condition,

even with the specially built sensor bracket with pre-tensioned springs to maintain the recommended distance. As a result, if the distance variation is significant, the optical flow sensor will not detect motion. Thus, relying solely on the optical flow sensor for estimation is insufficient. This section describes the method and model of determining a robot's configuration from the extended Kalman filter. The plant model of a mobile robot is represented by the following expression:

$$\mathbf{x}(k+1) = \Phi[\mathbf{x}(k), \mathbf{u}(k)] + \mathbf{v}(k) \tag{3.24}$$

$\Phi()$ is the state transition function and v() is a zero-mean Gaussian noise with covariance $\mathbf{C}_v(k)$. State vector, x at time k is defined as:

$$\mathbf{x}(k) = \begin{bmatrix} V_x(k) & V_y(k) & \omega(k) \end{bmatrix}^T \tag{3.25}$$

The state vector is integrated to compute the mobile robot's configuration.

$$\begin{bmatrix} x(t) \\ y(t) \\ \theta(t) \end{bmatrix} = \begin{bmatrix} \int_0^t V_x dt \\ \int_0^t V_y dt \\ \int_0^t \omega dt \end{bmatrix} \tag{3.26}$$

For this robot, the control input u(k) can be represented as follows.

$$\mathbf{u}(k) = \begin{bmatrix} V_R(k) & V_L(k) \end{bmatrix}^T \tag{3.27}$$

Respectively the $V_R(k)$ and $V_R(l)$, are right and left wheel velocities. Finally, the nonlinear plant model is defined as:

$$\Phi[\mathbf{x}(k), \mathbf{u}(k)] = \begin{bmatrix} V_x(k) + \dfrac{V_R(k) + V_L(k)}{2} \cos\theta(k) \\ V_y(k) + \dfrac{V_R(k) + V_L(k)}{2} \sin\theta(k) \\ \omega(k) + \dfrac{V_R^2(k) - V_L(k)}{D} \end{bmatrix} \tag{3.28}$$

The following is the measurement model:

$$\mathbf{x}_i(k) = \mathbf{h}[\mathbf{x}(k), \xi] + \mathbf{w}_i(k) \tag{3.29}$$

where $\mathbf{Z}_i$ is a vector of measurement of sensor i:

$$\mathbf{z}_i(k) = \begin{bmatrix} v_{i,x}(k) & v_{i,y}(k) \end{bmatrix}^T \tag{3.30}$$

$w_i(k)$ is the zero-mean Gaussian noise with a covariance of $\mathbf{C}_v(k)$). The measurement model is linear when the optical flow sensors are used:

$$\mathbf{z}_i(k) = \Lambda_E \mathbf{x}(k) + \mathbf{w}_i(k) \tag{3.31}$$

From the measurements, the state may be reconstructed as follows:

$$\hat{\mathbf{x}}(k) = \Lambda_E^{-1} \mathbf{z}_i(k) \tag{3.32}$$

The measurement model depicts the kinematic relationship between an optical flow sensor measurement and a mobile robot's velocity:

$$\begin{bmatrix} v_{i,x}(k) \\ v_{i,y}(k) \end{bmatrix} = \begin{bmatrix} V_x(k) - \omega(k) r_{i,y} \\ V_y(k) + \omega(k) r_{i,x} \end{bmatrix} + \mathbf{w}_i(k) \tag{3.33}$$

Through rearrangement of the equation,

$$z_i(k) = \begin{bmatrix} 1 & 0 & -r_{i,y} \\ 0 & 1 & r_{i,x} \end{bmatrix} \begin{bmatrix} V_x(k) \\ V_y(k) \\ \omega(k) \end{bmatrix} + \mathbf{w}_i(k) \tag{3.34}$$

Thus,

$$\Lambda_E = \begin{bmatrix} 1 & 0 & -r_{i,y} \\ 0 & 1 & r_{i,x} \end{bmatrix} \tag{3.35}$$

and Λ_E is not square. If there is no kinematic violation (for example, no side slip), pseudo inverse should be employed:

$$\hat{\mathbf{x}}(k) = \Lambda_E^{\#} \mathbf{z}_i(k) \tag{3.36}$$

where,

$$\Lambda_E^{\#} = \Lambda_E^T \left(\Lambda_E \Lambda_E^T \right)^{-1} \tag{3.37}$$

The uncertainty of the state is updated as follows:

$$\mathbf{P}(k+1 \mid k) = \Phi(k) \mathbf{P}(k) \Phi^T(k) + \mathbf{C}_v(k) \tag{3.38}$$

The Kalman gain can be computed as:

$$\begin{aligned} \mathbf{K}(k+1) = \mathbf{P}(k+1 \mid k) \Lambda_E^T \\ \left[\Lambda_E(k+1) \mathbf{P}(k+1 \mid k) \Lambda_E^T(k+1) + \mathbf{C}_w(k+1) \right]^{-1} \end{aligned} \tag{3.39}$$

As a result, the difference between real sensor data and the predicted one based on the state estimate is:

$$\mathbf{r}(k+1) = \mathbf{z}(k+1) - \mathbf{h}[\hat{\mathbf{x}}(k+1 \mid k), \xi] \tag{3.40}$$

Finally, the revised state estimate is provided by:

$$\hat{\mathbf{x}}(k+1) = \hat{\mathbf{x}}(k+1 \mid k) + \mathbf{K}(k+1)\mathbf{r}(k+1) \tag{3.41}$$

and the revised state covariance matrix is computed by:

$$\mathbf{P}(k+1) = [\mathbf{I} - \mathbf{K}(k+1)\Lambda_E]\,\mathbf{P}(k+1 \mid k) \tag{3.42}$$

To evaluate the extended Khan filter's performance, a large number of random numbers are deliberately introduced into the optical flow sensor readings. The estimation is deflected and the final estimated position is far from the real measured position when using a single optical flow sensor. The robot's path is smooth, and dead-reckoning calculations are superior to those obtained using an optical flow sensor. The Kalman filter-based prediction technique, which incorporates both the encoder and optical flow sensor data, delivers the best result. However, if there is side-slip, the plant model is invalid during the slip and the extended Kalman filter does not offer any useful estimation.

3.3 Hardware

The hardware used in the work consists of the ESP32-Cam module with the OV2640 camera. The following sub-chapter provides an overview of the hardware and its associated libraries.

3.3.1 ESP32-CAM

The ESP32-CAM is a Wi-Fi and Bluetooth dual-mode development board that uses PCB on-board antennas and cores based on ESP32 chips. It can work independently as a minimum system. The ESP32-CAM can be widely used in various IoT and machine vision applications. It has a very small size of 27x40.5x4.5mm and features a low-power 32-bit CPU for application processors. The minimum object distance is approximately 19.05 centimetres. The module can be seen in Fig. 3.12.

3.3.2 OV2640

The OV2640 is a 1/4-inch 2 megapixel CMOS image sensor that can output up to 1600x1200 pixels (UXGA). It has a wide field of view and can be used for security, surveillance, or machine vision applications.

The OV2640 is a SOC sensor with an on-chip ISP with auto-exposure, and auto-white balance in such a tiny sensor package. The sensor interface is a digital video port (DVP), a sort of parallel source synchronisation camera interface that uses 8bit data, horizontal/vertical synchronisation signal, and pixel clock.

Its primary feature is the hardware JPEG encoder, which off-loads CPU and memory space usage from the micro controllers. JPEG images with higher compression ratios and lower image

Figure 3.12: ESP32 with OV2640 camera (ESP32-CAM) [credit: Espressif]

resolution, on the other hand, may be readily stored and processed in micro controllers' internal RAM memory because the output JPEG image size is even smaller.

The module's performance parameters are the following:

1. Sensor size: 1/4 inch
2. Array Size: $1600 * 1200 (200w)$
3. Output format: RGB565 / JPEG/YUV/YCbCr, etc
4. Control interface: SCCB
5. Working voltage: 3.3 V
6. Sensitivity: 0.6V/Lux-sec
7. Module size: 27 mm x 27 mm

Users have total control over image quality, data format and transmission mode. The SCCB programming interface allows users to adjust all aspects of image processing, including gamma curve, white balance, saturation and chroma.

The OV46840 is a CMOS sensor camera with motion detection that utilises the OmniVision image sensor application of sensor technology, which helps to minimise or eliminate optical or electronic flaws in pictures such as fixed pattern noise, tail, float-away, etc.

3.3.3 AGV Robot

A mobile robot known as an Automatic Guided Vehicle (AGV) Robot is a type of portable robot that uses radio waves, vision cameras, magnets, or lasers for navigation. This robot serves to

Figure 3.13: AGV robot [credit: RM groups]

transport goods from one place to another. In warehouses and factories, as well as transport-intensive sectors like retail, it's particularly useful. It was by the end of the 20th century that the use of automated guided vehicles grew.

One type of robot that is currently being developed and researched is the multi-robot. This robot is more focused on terms of communication between robots so that the robots will not collide with each other [69]. These robots may communicate with one another to achieve well-defined objectives, and their strength lies in the collaboration between them.

It has been shown that multi-robot systems (MRS) are considerably more cost-effective than constructing a single expensive robot with all the capabilities. Because these technologies are typically decentralised, they contribute to the system's fault tolerance and security

3.4 Software

In terms of software, both the Arduino platform and OpenCV library were used. The following subsection will provide an overview of these tools.

3.4.1 Arduino Platform

Arduino is an open-source hardware and software platform that supports community design and the manufacture of single-board microcontrollers and microcontroller kits for building digital devices and can be programmed using C and C++ programming languages.

The boards feature digital and analog input/output (I/O) connections, which may be connected to different expansion boards, breadboards, and other circuits.

The Arduino project began in 2005 as a tool for students at the Interaction Design Institute Ivrea in Italy to make devices that interact with their environment using sensors and actuators.

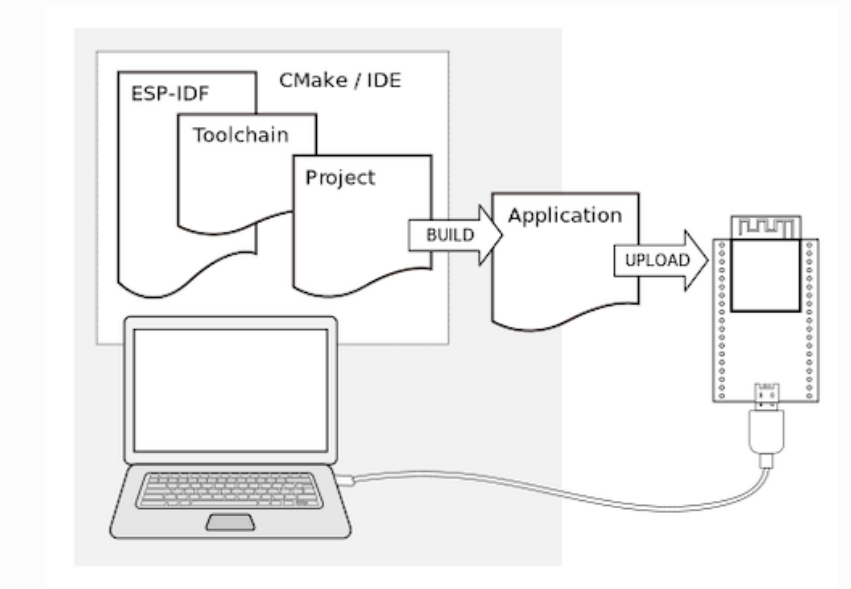


Figure 3.14: Espressif System between computer and ESP32 [Credits: Espressif]

3.4.2 OpenCv

Open Source Computer Vision Library (Open CV) is a computer vision and machine learning library originally developed by Intel. The open-source Apache 2 License allows users to utilise and modify the code.

There are over 2500 optimised algorithms in the library, including a broad range of both classic and cutting-edge computer vision and machine learning techniques.

These algorithms can be used to detect and recognise faces, identify objects, track camera movements, track moving objects, find similar images from an image database, etc.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilise and modify the code.

3.4.3 Espressif Systems

Espressif Systems is a Shanghai, China-based semiconductor company that designs and develops low-power Wi-Fi and Bluetooth chips for the Internet of Things.

Espressif's first product was the ESP8266, a Wi-Fi microcontroller module that can be programmed to run a variety of applications. The ESP8266 has been used in a number of commercial products, including the Philips Hue smart light bulbs and the Amazon Echo Dot.

In 2016, Espressif released the ESP32, a successor to the ESP8266 that offers both Wi-Fi and Bluetooth support.

3.4.4 Selenium

Selenium software is designed to help test web applications. It allows users to write tests in several different programming languages, including Java, C, and Python. Selenium software can be used to automate testing of web applications, including regression testing and load testing.

Selenium tests can be run in any browser, and can be used to test both static and dynamic content. Selenium also includes a tool called Selenium IDE, which allows users to record and play back tests without writing any code.

Chapter 4

Methodology

This chapter addresses the methodology implemented in order to achieve the proposed goals of this project. The primary goal is the implementation of optical flow algorithms, but other important aspects are discussed in order to develop a functioning robot navigation system based on optical flow methods that do not require human feedback, like real-time and computation analysis of optical flow.

In Section 4.1, the quality of the optical flow vectors generated by the developed algorithms are evaluated and the structure of the experiments. It will be presented also several public datasets of optical flow that can be used in the first part of the evaluation of optical flow, as well as the evaluation metrics. The practical implementation of the optical flow algorithms will be discussed, so the theoretical part of the work can be better explained and developed. Some optimisations and improvements to the algorithms will be discussed, but not to a great extent. Finally, the creation of the floor sequence dataset and structure created to assure that the results are the most accurate possible will be explained. The structure of the real-time navigation system, more specifically the optical flow calculation part (visual odometry), is also discussed.

In the next Section (4.2), the approach to the application of traditional optical flow methods for robot localisation is discussed. Four algorithms are evaluated and the results analysed in two different datasets, one public and one generated (floor sequences). The first one is the evaluation of the optical flow techniques on a selected dataset (Middlebury). After the analysis of the dataset, a number of sequences of frames of the floor with different characteristics were analysed by the same algorithms. These experiments were all made "offline", that is, the pictures were taken with the module ESP32-CAM and the optical flow computed on a computer. This approach will enable to test the algorithms and their correct sequence, as well as a later and easier adaptation to a real time system online or on the ESP32-Cam. This is also explained in Section (4.1). Finally, the visual odometry system will be tested in real time and the results analysed. Since the runtime of the algorithms is an aspect of great importance, an alternative to the real-time navigation system was proposed, where the the algorithm Horn and Schunck were adapted in order to work on

the ESP32-Cam module and the optical flow vector field computed on it. There will be a small discussion about the protocol to be used in order to send information to the robot, since the ESP32-Cam module would need to send this information to the robot directly or send it to the computer for the odometry computation.

The discussion of the results, improvements, limitations and achievements of the visual odometry system will be explored in the next Section.

4.1 Evaluation Methodology for Optical Flow

Optical flow estimation is an extensive field, posing ambiguous problems in diverse ways. The major issues making optical flow a complicated subject include occlusions, large displacements, non-rigid motion, discontinuities, mixed pixels, varying illumination, motion, and camera blur. In the classic period, Barron [8] established optical flow benchmarks dealing with simple transformations (translation, rotation) and small displacements (Yosemite) [70]. Modern researchers created more challenging benchmarks. So, the measures of performance will be the initiation point.

An optical flow algorithm finds two dimensional velocity vectors describing the motion field. The degree of success of a method is measured by reporting its errors.

There are mainly two measures for optical flow algorithms: End point error (EE) and Angular error (AE).

End Point Error is the fundamental measure and describes the Euclidean distance between the estimated vectors and the ground truth vectors:

$$EE = |(v - v_g)| = \sqrt{(u - u_g)^2 + (v - v_g)^2} \tag{4.1}$$

$V = (u, v)$ is the estimated vector and $V_g = (u_g, v_g)$ is the ground truth vector.

Angular error (AE) is also a measure used by Barron [8].

This is the second most common error measure. So, let $(u, v, 1)$ be the extended 3D flow vector and $(u_g, v_g, 1)$ be the ground truth.

$$AE = \cos^{-1}\left[(u.u_g + v.v_g + 1.0) \,/\, \sqrt{(u^2 + v^2 + 1.0)\left(u_g^2 + v_g^2 + 1.0\right)}\right] \tag{4.2}$$

Significance AE is appropriate for small displacements and is more inclined to under-estimate large motion. EE is good for large vectors. Advance level sequences contain significantly large displacements. This has led most of the modern researchers to report EE instead of AE.

But, datasets play an important role in computer vision. Image processing domains such as stereo, face, and object recognition have challenging datasets.

Optical flow was one of the first to introduce standard datasets for quantitative comparisons [8]. The improving estimation techniques and modern algorithms demanded an advanced level of datasets for better comparisons of the latest methods.

Classic work on optical flow relied on synthetic datasets [8] with the Yosmite Sequence being the most well-known.

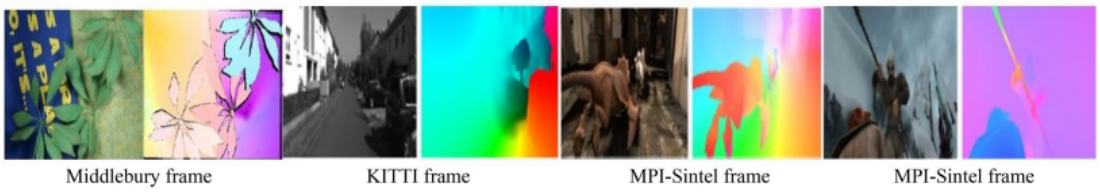


Figure 4.1: Few frames and the corresponding ground truth from modern datasets

S. Baker presented Middlebury datasets [5] with dense ground truth bringing in new evaluation standards, fol- lowed by the path-breaking work of KITTI [31, 53], MPI-Sintel [13], flying chairs [28] and Freiburg [52]. These datasets pose advanced challenges when compared to previously used sequences. The researchers can freely use the dataset for training algorithms and upload the evaluated flow to compare the efficiency of their proposed methods. The online access to these datasets and evaluated optical flow is a prominent feature of the present-day research in the related field.

Figure 4.2 depicts some images from modern datasets used for optical flow estimation. Some of their salient features are discussed below.

Middlebury is a leading benchmark to address advanced level problems, covering evaluation at a broader spectrum and wider range of statistical measures [5]. Comprising subpixel ground truth and ample difficulty, these are realistic synthetic sequences with non-rigid motions, complex scenes and higher texture. However, the motions are small as compared to more advance datasets [13, 31]. In contrast to the previously used simple synthetic sequences (Yosemite [8]), these datasets are considerably more challenging which include additional complex scenes, larger ranges, higher realistic texture and independent motion. The sequences are divided into training and testing categories.

The ground truth is provided only for the first one. Although the dataset contains advanced level complex motions, most of the motions are small.

For training sets, the percentage of the pixels having motion over 20 pixels is less than 3%. The Middlebury are the primary standard datasets posing advanced level of challenges, used by modern algorithms for the estimation of stereo disparity and optical flow.

Another important aspect of these datasets is the use of several new measures to test the performance of flow-algorithms. The most important of these measures are average angular error (AE) [8] and endpoint error (EPE).

The KITTI datasets were created by Geiger in 2012 [31] and contain 194 training and 195 test pairs of images with sparse ground truth flow. All images are grey and include complex lighting conditions with large displacements.

Later in 2015, Menze [53] annotated the dynamic scenes with 3D CAD models for all vehicles in motion and obtained an extended version, with 200 training and 200 test scenes. The KITTI datasets contain stereo videos of road scenes from a calibrated pair of cameras mounted on a car.

Figure 4.2: Middlebury frame 0, frame 1 and Ground Truth Flow [credit: Middlebury]

Ground truth is obtained from real world scenes combining recordings from the camera and a 3D laser scanner.

Although the datasets are real, the density of the ground truth varies from 75 to 90% and underlies the static parts of the scene such as sky or distant objects. For the latest version of KITTI-2015, 3D models of cars were fitted to the point clouds to obtain a denser labelling which also included moving objects. However, the ground truth in these areas is still an approximation. Besides adopting statistical measures described in Sect. 3, KITTI has introduced particular metrics for evaluation to be performed under special circumstances for both versions.

Before the introduction of KITTI, MPI-Sintel [13] were the largest datasets for optical flow and disparity estimation. Derived in 2012, from an open source 3D-animated film, these datasets are completely synthetic. The original movie frames were modified to pose new challenges for estimation methods. With a resolution of 1024×436, the scenes are designed to be strictly realistic with fog and added motion blur. These datasets provide a naturalistic video sequences containing flow fields, motion boundaries, unmatched regions, and image sequences. The complete data consists of 23 training sequence 1064 frames and 12 testing sequences with 564 frames. Ground truth is available for the training set only. For the modern algorithms, Sintel datasets play a vital role as they provide dense ground truth with adequate occlusion for both small and large displacements. The datasets provide three versions: albedo, clean and final. Albedo is the simplest set and contains no additional effects. The clean version introduces negligible variations to illumination while the final version adds more intricate features such as atmospheric effects, fog, shadows and blur of different types. Researchers commonly use only clean and final versions.

These Flying chair's images were designed specifically for training Convolutional Neural Networks (CNNs) used in deep learning methods to evaluate optical flow. Created by the application of affine transformations to real images and synthetically rendered chairs, the dataset contains 22,872 image pairs; 22,232 training and 640 test samples according to the standard evaluation split. Dosovitskiy [22] created these datasets to train his convolutional network for optical flow estimation. These datasets are large, do not contain any 3D motions, and hence are limited to single-view optical flow.

Freiburg-Berkeley is the latest and the largest data collection for optical flow, stereo and scene flow evaluation [52]. Containing 34,801 stereo training frames and 4248 test frames in 960x540

Datasets	Frame Pairs	Frames with ground truth	Ground truth density (%)
Middlebury	72	8	100
KITTI-2012	195	194	50
KITTI-2015	200	200	75-90
MPI-Sintel	1041	1041	100
Flying Chairs	22,872	22,872	100
Freiburg	34,801	34,801	100

Table 4.1: Different datasets and the available ground truth

resolutions, these datasets are synthetically produced by 3D suit blended mainly to be employed in deep learning schemes. The first large-scale datasets to enable training and evaluation of scene flow methods consists of three subsets as described below. Table 4.1 gives salient features of the famous dataset used by modern researchers for optical flow estimation.

4.2 Practical Implementation of Optical Flow Algorithms

After having two consecutive frames, the first step of the Lucas-Kanade algorithm is to define the size of the window. This parameter is important since a window size that is too large can cause a point to not move like its neighbours. With the color constancy constraint (E.q.3.2), it will be assumed that in a local area of one pixel, the motion field is very smooth. In fact, it is assumed that in a small rectangular window around this pixel, the value is the same for every (u,v). For example, if there is a 5×5 window, there are 25 pixels in that window. Now, if assuming there was one (u,v) for all of these the (u,v) for the center pixel, that would result in 25 equations per pixel.

Every pixel has associated an equation:

$$I_x(p_n)u + I_y(p_n)v = -I_t(p_n) \tag{4.3}$$

Being n the number of the pixel in the window and p the pixel. It can be written as a single vector equation:

$$\vec{\nabla} I(p_n) \cdot \vec{u}(p_n) = -I_t(p_n) \tag{4.4}$$

where $\vec{\nabla} I$ and $\vec{u}$ are, respectibly:

$$\vec{\nabla} I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \vec{u} = \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.5}$$

In E.q.3.4, it's used the localisation (k,l) of the pixel inside the window. This results in the following set of equations:

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix} \tag{4.6}$$

This set of equation is equal to the set of equations found in E.q.3.4. Although, this set of equations assumes that the images are in grey scale, that varies from white to black in different intensities of grey. If working with RGB images, there would be 75 instead of 25 equations. That is, $5 \times 5 \times 3$, where 3 is the number of channels:

$$\begin{bmatrix} I_x(p_1)[0] & I_y(p_1)[0] \\ I_x(p_1)[1] & I_y(p_1)[1] \\ I_x(p_1)[2] & I_y(p_1)[2] \\ \vdots & \vdots \\ I_x(p_{25})[0] & I_y(p_{25})[0] \\ I_x(p_{25})[1] & I_y(p_{25})[1] \\ I_x(p_{25})[2] & I_y(p_{25})[2] \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1)[0] \\ I_t(p_1)[1] \\ I_t(p_1)[2] \\ \vdots \\ I_t(p_{25})[0] \\ I_t(p_{25})[1] \\ I_t(p_{25})[2] \end{bmatrix} \tag{4.7}$$

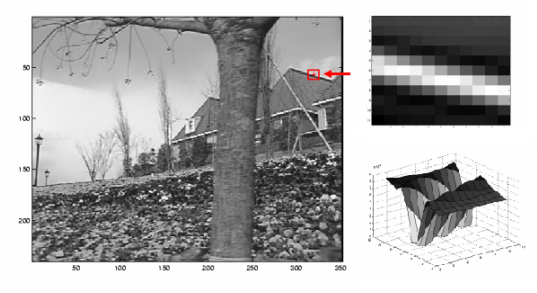
It can be said that the window was only a single pixel and that there was one equation and two unknowns. But with RGB there are now three equations and two unknowns. Well, there is no way to solve it. The problem is that the RGB images are quite correlated so it isn't possible to just use the different color planes in order to solve it.

So, the first step is to convert the images into grey scale, in order to reduce the number of equations, without losing too much information in relation to the RGB images. All the algorithms will have this conversion from RGB to grey scale.

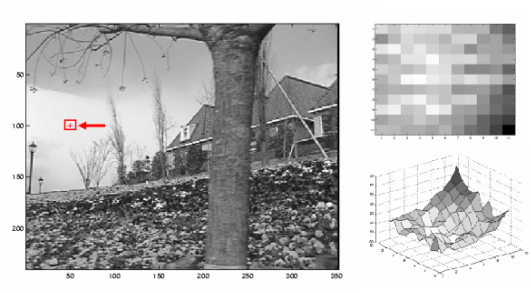
The Lucas-Kanade algorithm in order to work properly, needs to analyse the parts of the images, with the window size defined, where exist regions of interest. These regions of interest are defined as regions where the eigenvalues λ_1 and λ_2 of $\mathbf{A}^T\mathbf{A}$ (E.q.3.7) are not too small. That is, $\mathbf{A}^T\mathbf{A}$ should not be too small due to noise. The division of the eigenvalues should also λ_1/λ_2 should not be too large (λ_1 being the larger eigenvalue). Meaning that $\mathbf{A}^T\mathbf{A}$ should well-conditioned.

Regions with edges are not good candidates since the gradients are to large and all the same or λ_1 is large and λ_2 is small (Fig.4.3a). Low texture regions shouldn't be considered also since the gradients have small magnitude, that is λ_1 and λ_2 are small (Fig.4.3a). Finally, the best candidates are the high textured regions, where the gradients are different and have large magnitude (Fig.4.3c). The eigenvalues λ_1 and λ_2 are both large.

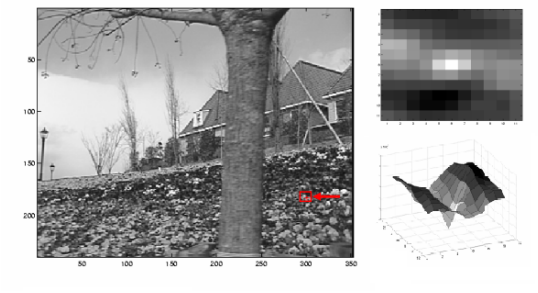
Another approach consists of computing the determinant of $\mathbf{A}^T\mathbf{A}$ for the whole image and in every window, evaluate if it's bigger than zero. If this is the case, the optical flow is computed in that window of the image. This approach is more simple and more prone to errors, but the implementation is very simple and intuitive. This method was used in the practical implementation,



(a) Region in image with edge



(b) Region in image with low texture



(c) Region in image with high texture

Figure 4.3: Regions in the image (windows of size $n \times n$), the red square, with different characteristics (region of interests and topographic view of the region) [credits: Washington University]

since the detection of points of interests needed other algorithm implementation.

With the regions of interest identified, it's necessary to compute the image derivatives I_x, I_y, I_t. This step is fundamental since the minimum least squares solution (that minimise $\|Au-B\|^2$) requires these derivatives and it needed to find the regions of interest. The minimum least squares solution (E.q.3.5) of **u** given by solution:

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \tag{4.8}$$

In order to compute the image derivatives, the convolution operation for the frame t will be used with the following masks:

$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} \tag{4.9}$$

These masks are f_{x1}, f_{y1}, f_{t1}. The masks for the frame $t+1$ or $t+dt$ are the same for f_{x1}, f_{y1} ($f_{x1} = f_{x2}$, $f_{y1} = f_{y2}$), but f_{t2} as the values of matrix positive:

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{4.10}$$

A particular moving object mask will be used in order to find the values of f_x, f_y and f_t, and then, convolve it with the t image. Further, the mask with the image will be used in the next frame,

represented by $t + dt$. Next, the average of both of these images will be used while applying convolution, resulting in the value of I_x. In the same way, it is possible to find I_y, which is the y direction of the mask, and I_t, which is a derivation in time.

Representing the entire masking process mathematically, the result should be the following.

$$I_x = \frac{img\_t[n][n] * f_{x1} + img\_t + dt[n][n] * f_{x2}}{2} \tag{4.11}$$

$$I_y = \frac{img\_t[n][n] * f_{y1} + img\_t + dt[n][n] * f_{y2}}{2} \tag{4.12}$$

$$I_t = \frac{img\_t[n][n] * f_{t1} + img\_t + dt[n][n] * f_{t2}}{2} \tag{4.13}$$

The $*$ character represents the operation of convolution and $img\_t[n][n]$ and $img\_t + dt[n][n]$ represent the matrices with the intensity values of the images in grey scale at time t and $t + dt$, respectively. The determinant of $\mathbf{A}^T\mathbf{A}$ was computed the following way:

$$I_{xx} \times I_{yy} - I_{xy}^2 \tag{4.14}$$

Being I_{xx}, I_{yy} and I_{xy}^2 the summation of the image derivatives of $I_x \times I_x$, $I_y \times I_y$ and $I_x \times I_y$, respectively in the system.

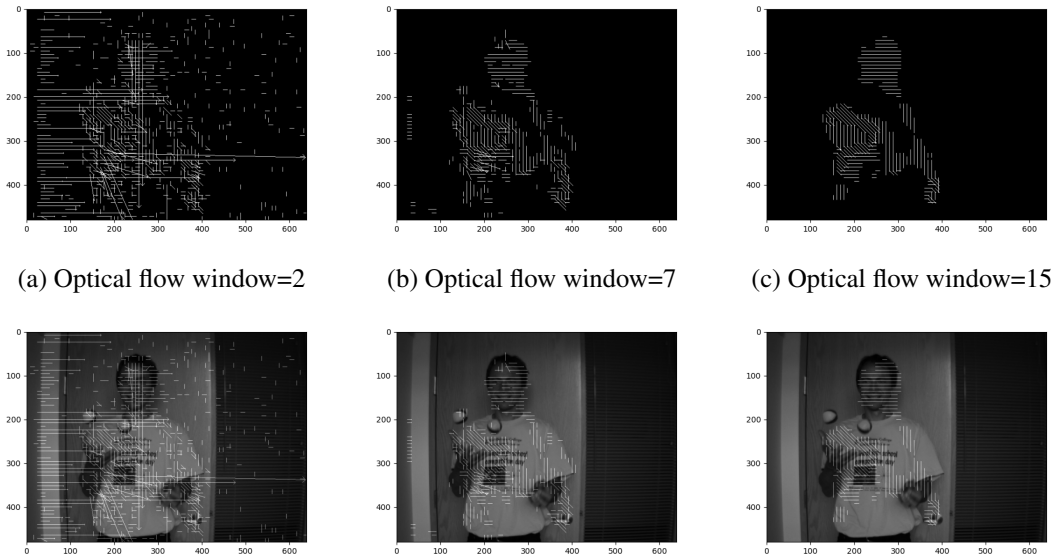
In order to detect points of interest, OpenCV has the function (cv2.goodFeaturesToTrack()). This way there is no need to compute the eigenvalues, or just in the windows with good features in order to check their value. A property of this function is the possibility of specifying a maximal number of feature points to be detected and tracked. Although, since the goal is the implementation of an algorithm in a micro controller (ESP32-Cam), OpenCV is not going to be explored, even though this solution is presented.

In Figure 4.4 is possible to see an example of the Lucas-Kanade algorithm. The optical flow is sparse as expected, or in other words, it doesn't have a density of 100% in any case. In these examples, the bigger the window size, the more defined are the regions.

The Lucas-Kanade algorithm doesn't produce good results when the dislocation is large. In order to compute optical flow fields for sequences of frames that have bigger displacements, a pyramid coarse-to-fine version of the Lucas-Kanade algorithm needs to be implemented.

The Pyramidal Lucas-Kanade optical flow algorithm estimates the 2D translation of sparse feature points from a previous frame to the next. Image pyramids are used to improve the performance and robustness of tracking overly large translations. Inputs are previous image pyramid, the next image pyramid, and the feature points on the previous image. Outputs are the feature points on the next image and the tracking status of each feature point. Each feature point defines their location in the image with (x, y) coordinates. These points are then tracked in the next image. The tracking status will inform whether the feature point is being tracked successfully or not.

The implementation of this algorithm consisted in the previous algorithm for the Lucas-Kanade method in a loop where the number of level it's defined. In the first part of the algorithm the frames



(a) Optical flow window=2 (b) Optical flow window=7 (c) Optical flow window=15

(d) Optical flow on image win=2 (e) Optical flow on image win=7 (f) Optical flow on image win=15

Figure 4.4: Optical Flow using Lucas-Kanade with different windowsize (2, 7, 15 pixels) on the image and on black and white background

are down-sampled using a gaussian blur. Then the increments (dx, dy) are computed with Lucas-Kanade algorithm and summed to the (u, v) vectors.

In the Figure 4.5, it's possible to see that the resolution diminishes along the sequence and the number of optical flow vectors increases. Another important aspect is that some vectors increase their magnitude and represent a largr motion.

The Horn and Schunck algorithm implementation begins with computation of the image derivatives I_x, I_y and I_t. The next step, consists in trying to reduce the method's sensitivity to noise. This is done by computing the Laplacian operator $\Delta^2 u$ by using a spatial filter or a Laplacian mask which is usually applied after the smoothing process of the image, usually the result of a Gaussian blur.

$$\begin{bmatrix} \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ \frac{1}{6} & -1 & \frac{1}{6} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \end{bmatrix} \tag{4.15}$$

Analysing the matrix in Eq4.15, it's possible to see that the diagonals are given less weight compared to the four neighbours, which have higher weight. So, the Laplacian mask can be mathematically represented written as:

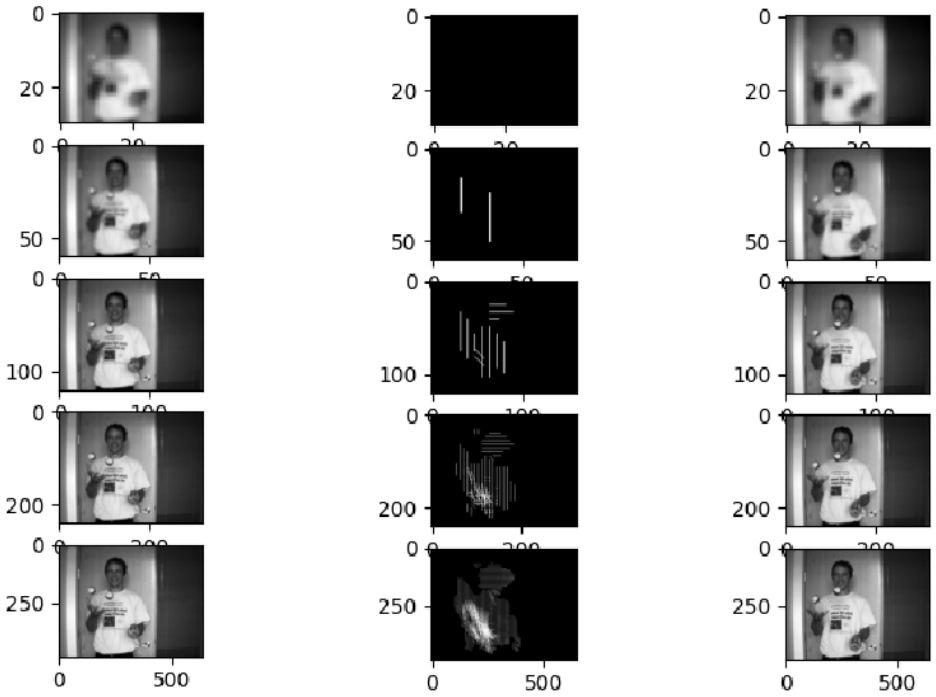


Figure 4.5: Lucas-Kanade algorithm example with pyramid of six images of Beanbags frames [credit: Middlebury]

$$\begin{aligned} \left(\Delta^2 u\right) &= \left(u_{avg} - u\right) \\ \left(\Delta^2 v\right) &= \left(v_{avg} - v\right) \end{aligned} \tag{4.16}$$

In the equations above, u_{avg} and v_{avg} represent the average of neighbourhood pixels, respectively. After finding the average, the middle pixel is written as an average of its neighbouring pixels as $u_{avg} - u$. It is possible to perform the same averaging operation for v as well.

In other words, this represents a transformation in the continuous to the discrete domain. The only difference is that instead of using the term $\Delta^2 u$, this is representing the same thing with the term $u_{avg} - u$ and $u_{avg} - u$, respectively. Solving these two equations.

$$\left(f_x u + f_y v + f_t\right) f_x - \lambda \left(u_{avg} - u\right) = 0 \tag{4.17}$$

By manipulating the equation, the following equation is obtained:

$$f_x f_y u + \left(f_y^2 + \lambda\right) v = -f_t f_y + \lambda v_{avg} \tag{4.18}$$

With this configuration, it's possible to use the Cramer's rule by creating three determinants,

represented by D, D_u and D_v respectively. Applying the Cramer's rule, the three determinants are defined:

$$D = \begin{vmatrix} (f_x^2 + \lambda) & f_x f_y \\ f_x f_y & (f_y^2 + \lambda) \end{vmatrix} \tag{4.19}$$

$$D_u = \begin{vmatrix} -f_t f_x + \lambda u_{avg} & f_x f_y \\ -f_t f_y + \lambda v_{avg} & (f_y^2 + \lambda) \end{vmatrix} \tag{4.20}$$

$$D_v = \begin{vmatrix} (f_x^2 + \lambda) & -f_t f_x + \lambda u_{avg} \\ f_x f_y & -f_t f_y + \lambda v_{avg} \end{vmatrix} \tag{4.21}$$

The determinant D is solved the following way:

$$\begin{aligned} D &= (f_x^2 + \lambda)(f_y^2 + \lambda) - (f_x f_y)^2 \\ &\Leftrightarrow f_x^2 f_y^2 + \lambda f_y^2 + \lambda f_x^2 + \lambda^2 - (f_x f_y)^2 \\ &\Leftrightarrow \lambda (f_y^2 + f_x^2 + \lambda) \end{aligned} \tag{4.22}$$

The determinant D_u is represented this way:

$$\begin{aligned} D_u &= (-f_t f_x + \lambda u_{avg})(f_y^2 + \lambda) - (-f_t f_y + \lambda v_{avg}) f_x f_y \\ &\Leftrightarrow (-f_t f_x f_y^2 - f_t f_x \lambda + f_y^2 \lambda u_{avg} + \lambda^2 u_{avg} + f_t f_x f_y^2 - \lambda v_{avg} f_x f_y) \end{aligned} \tag{4.23}$$

Finally, D_v is solved the following way:

$$\begin{aligned} D_v &= (-f_t f_y + \lambda v_{avg})(f_x^2 + \lambda) - (-f_t f_x + \lambda u_{avg}) f_x f_y \\ &\Leftrightarrow (-f_t f_x^2 f_y - f_t f_y \lambda + f_x^2 \lambda v_{avg} + \lambda^2 v_{avg} + f_t f_x^2 f_y - \lambda u_{avg} f_x f_y) \end{aligned} \tag{4.24}$$

In order to determine u and v, the following equations are computed:

$$\begin{aligned} u = \frac{D_u}{D} &= \frac{(-f_t f_x \lambda + f_y^2 \lambda u_{avg} + \lambda^2 u_{avg} - \lambda v_{avg} f_x f_y)}{\lambda (f_y^2 + f_x^2 + \lambda)} \\ u = \frac{D_u}{D} &= \frac{(-f_t f_x + f_y^2 u_{avg} + \lambda u_{avg} - v_{avg} f_x f_y)}{(f_y^2 + f_x^2 + \lambda)} \end{aligned} \tag{4.25}$$

$$\begin{aligned} u = \frac{D_u}{D} &= \frac{(-f_t f_x \lambda + f_y^2 \lambda u_{avg} + \lambda^2 u_{avg} - \lambda v_{avg} f_x f_y)}{\lambda (f_y^2 + f_x^2 + \lambda)} \\ u = \frac{D_u}{D} &= \frac{(-f_t f_x + f_y^2 u_{avg} + \lambda u_{avg} - v_{avg} f_x f_y)}{(f_y^2 + f_x^2 + \lambda)} \end{aligned} \tag{4.26}$$

Finally arriving at the equations describing u and v in terms of u_{avg} and v_{avg}:

$$
\begin{aligned}
u = \frac{D_u}{D} &= \frac{\left(f_y^2 u_{avg} + f_x^2 u_{avg} + \lambda u_{avg} - f_x^2 u_{avg} - v_{avg} f_x f_y - f_t f_x\right)}{\left(f_y^2 + f_x^2 + \lambda\right)} \\
u = \frac{D_u}{D} &= \frac{u_{avg}\left(f_y^2 + f_x^2 + \lambda\right) - f_x\left(f_x u_{avg} + f_y v_{avg} + f_t\right)}{\left(f_y^2 + f_x^2 + \lambda\right)}
\end{aligned}
\tag{4.27}
$$

$$
\begin{aligned}
v = \frac{D_v}{D} &= \frac{\left(f_y^2 u_{avg} + f_x^2 v_{avg} + \lambda v_{avg} - f_y^2 u_{avg} - u_{avg} f_x f_y - f_t f_y\right)}{\left(f_y^2 + f_x^2 + \lambda\right)} \\
v = \frac{D_v}{D} &= \frac{v_{avg}\left(f_y^2 + f_x^2 + \lambda\right) - f_y\left(f_x u_{avg} + f_y v_{avg} + f_t\right)}{\left(f_y^2 + f_x^2 + \lambda\right)}
\end{aligned}
\tag{4.28}
$$

In terms of simplification and the fact that it simplifies the implementation of the algorithm, two equations are used in order to represent u and v the following way:

$$
\begin{aligned}
u &= u_{avg} - f_x \frac{P}{D} \\
v &= v_{avg} - f_y \frac{P}{D}
\end{aligned}
\tag{4.29}
$$

Where P and D are defined the following way, respectively:

$$
P = \left(f_x u_{avg} + f_y v_{avg} + f_t\right) \tag{4.30}
$$

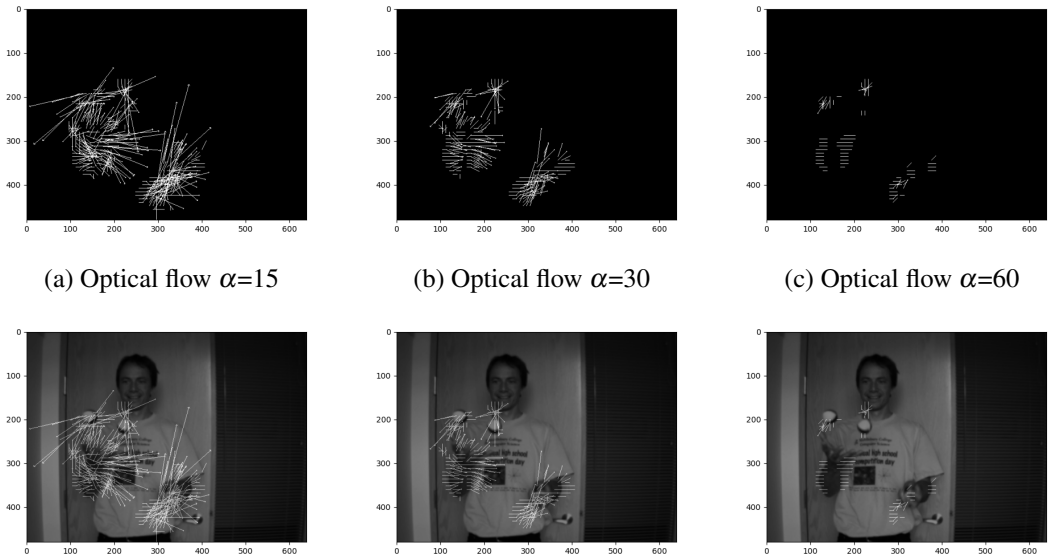
$$
D = \left(f_y^2 + f_x^2 + \lambda\right) \tag{4.31}
$$

Now that is possible to compute u and v, the algorithm ends when the Cost Function is minimised. This can be achieved by setting a number of iterations or defining a δ where if the previous u matrix and the actual u matrix have a 2-norm (largest sing. value) smaller than the δ defined. The Horn and Schunck algorithm is defined in the algorithm number 1. In Figure 4.6, the same example used in Lucas-Kanade method is used. In the variation presented in this images, the α values varies from 15, 30 and 60.

The optical flow becomes more contained as the penalisation value increases. The regions where optical flow is more present are the arms and the juggling balls, it matches exactly with the motion expected. It's important to note that the Horn and Schunck algorithm produces flow fields with 100% density, but in order to be easier to see the difference between examples, the optical flow values with less magnitude are not represented.

The last implementation is the SAD Blocking Matching algorithm that is present in some form in the sensor PX4FLOW Smart Camera. This algorithm consists in an extensive search with the cost function of Mean Absolute Difference (MAD).

The parameters that vary in this algorithm are the stride, spatial shift and the window size. The stride represents the number of rows and columns traversed per slide (from macroblock to macroblock), spatial shift refers to number of columns and rows that shift in relation to the start of the comparison window in relation to the start of the image, and the window size represents the size of the comparison window



(a) Optical flow α=15 (b) Optical flow α=30 (c) Optical flow α=60

(d) Optical flow on image α=15 (e) Optical flow on image α=30 (f) Optical flow on image α=60

Figure 4.6: Optical Flow using Horn-Schunck with different alpha (α = 15, 30, 60) on the image and on black and white background

The algorithm consists of two loops, where the first window of comparison is defined and the min_{dist} is set to *None*, in order to set the first distance value (2). This variable is used later in order to compare the distance after the computation of the sum of absolute differences (SAD) and update the value. The comparison block is defined and the sum of absolute differences computed. If the minimum distance is greater than the distance computed (SAD), the value is updated. This process continues until the comparison between all blocks as finished.

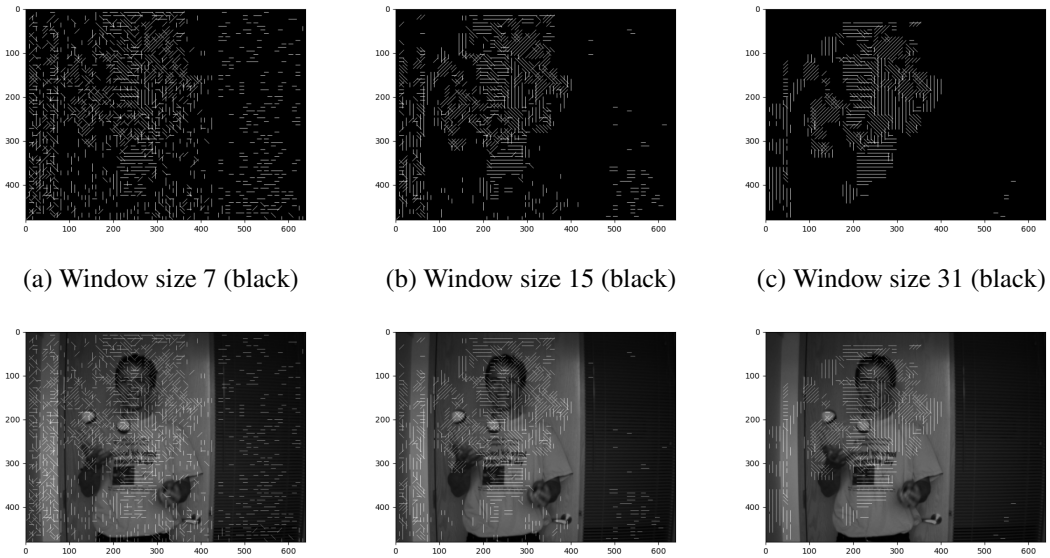
Some results can be seen in Figure 4.7 and 4.8. The parameter variation of the first parameter is the window size that varies from 7, 15 and 31 pixels. The size of the window must be odd. The computation time and computer resources are only going to be discussed in the next section where the the algorithms are going to be tested with datasets. In the second image, the parameter variation is the shift that in relation to the previous example changes from 1 to 5. The stride parameter evaluation is now represented.

4.2.1 Structure of the Experiments

The structure of the experiments is divided in the setup and methodology of the floor sequence dataset creation and the more software oriented real-time optical flow calculation.

First, it was necessary to create an adaptable setup to take the photos with the ESP32-Cam module. For this end, the setup created is represented in Figures 4.9b and 4.9a.

This setup is composed of a metal structure that has three height level starting at 20 cm, and ending at 50 cm. In each level its added a small object to hold the ruler in a way that, in theory, only horizontal or vertical components of the optical flow can be represented at the time. In the



(a) Window size 7 (black) (b) Window size 15 (black) (c) Window size 31 (black)

(d) Window size 7 (on image) (e) Window size 15 (on image) (f) Window size 31 (on image)

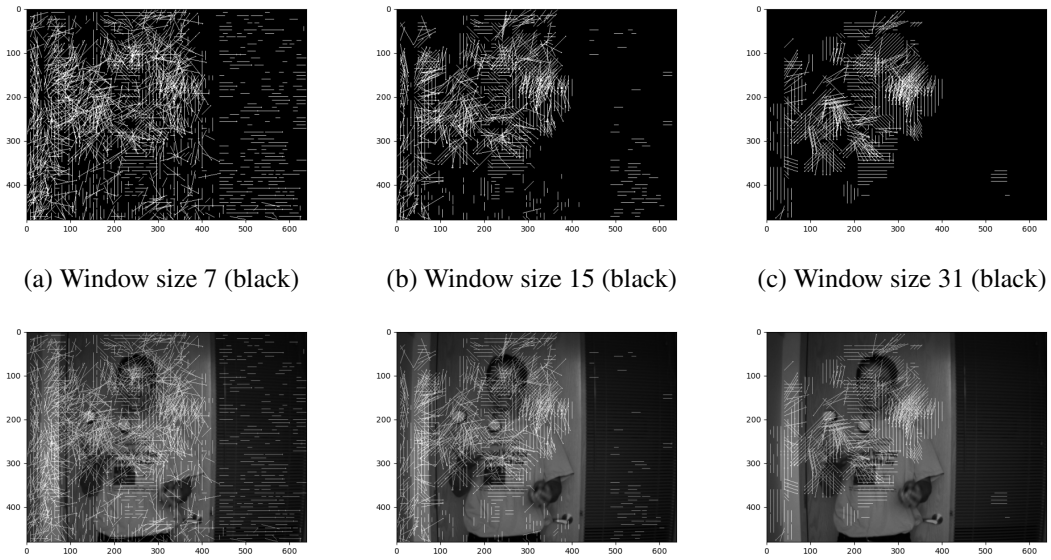
Figure 4.7: Optical Flow using Block Matching (SAD) with different window (7, 15, 31) and shift equal to 1 on the image and on black and white background [credit: Middlebury]

ruler, the ESP32-Cam is attached with glue and elastic rubbers in order to make the ESP32-Cam immobile in relation to the ruler, and to move along with the movement of the ruler. The movement of the ruler simulates the dislocation of the robot. In this scenario, there are two soft boxes that illuminate the scene and are placed in two different places in order to erase shadows if needed and control the illuminance in the room. Several types of floors are introduced in the bottom of the structure. This scheme is later used to implement the visual navigation system in real time, but the intermediary step of saving the photos and then applying the optical flow algorithms are automated. This scheme is presented in Figure 4.12.

Having the ESP32-Cam module taken the photos, the photos are sent to a local server. This simple website has the three options (Figure 4.10). The first button (ROTATE) rotates the image, the second (CAPTURE PHOTO) sends the command to the ESP32-Cam to take a photo, and finally the third button (REFRESH PAGE) just refreshes the image in order to appear a new image in the website.

In the ESP32-Cam, an Arduino program was created with the ESP32-Cam firmware that enables to select the settings of the images sent to the website and to create the routines to take the photo and send them to the website. With these photos in the computer (the photos are downloaded from the website), it's possible to compute the optical flow with four algorithms. All these experiments are considered "offline", since the optical flow calculation is not performed in the ESP32-Cam, but on the computer. Later, it will be proposed a configuration where all the processing will happen in the ESP32-Cam.

Having the setup for the creation of the floor sequence, it's necessary to define the parameters of the tests. Since there is a multitude of parameters, some of them have to be fixed in other to



(a) Window size 7 (black) (b) Window size 15 (black) (c) Window size 31 (black)

(d) Window size 7 (on image) (e) Window size 15 (on image) (f) Window size 31 (on image)

Figure 4.8: Optical Flow using Block Matching (SAD) with different window (7, 15, 31) and shift equal to 5 on the image and on black and white background [credit: Middlebury]

generate relevant results without spending a considerable amount of time switching and testing all the variations. The resolutions available for the ESP32-Cam module are represented in Table 4.3. Not all resolutions are suited to be used since some are too high like SXGA (1280 x 1024) that, without any resizing, would make the optical flow calculation very time consuming.

Name of Resolution	*Resolution (pixels)*
UXGA	1600 x 1200
QVGA	320 x 240
CIF	352 x 288
VGA	640 x 480
XGA	1024 x 768
SXGA	1280 x 1024

Table 4.2: Resolutions available in ESP32-Cam Espressif Firmware

In the experiments, the height is fixed at 21 cm (first level of the structure) and the illuminance at the surface of floor is of 500 lux. This way, the parameters that can be changed are:

1. Displacement of the ESP32-Cam (0.5cm, 1 cm, 2cm, etc.)
2. Algorithm (Horn and Schunck, Lucas-Kanade, etc.)
3. Parameters of the algorithms (window size, etc.)
4. Resolutions (QVGA, VGA, etc.)

(a) ESP32-Cam Setup Top-View (b) ESP32-Cam Setup Close-View

Figure 4.9: Setup for the ESP32-Cam Floor Sequence Creation and Real-Time Optical Flow Calculation

With this scheme, it is possible to evaluate the effect of the parameters of the algorithms and understand the runtime and resources used by each type of resolution.

For this sequence of frames, they were taken with the ESP-CAM module and transmitted to the PC using Arduino and a simple web server. By being connected to the internet, several photos were taken and saved temporarily on the ESP32-Cam SPIFSS and immediately saved in the PC through the local server. After having the photos, the algorithms were applied and the optical flow field are computed. In Figure 4.10 it's possible to see the webpage that enables the ESP32-Cam to take and send photos.

This scheme is later used to implement the visual navigation system in real time, but the intermediary step of saving the photos and then applying the optical flow algorithms are automated. This scheme is presented in Figure 4.12.

4.3 Robot Localisation with Traditional Optical Flow Methods

From the previous Sections, there was an overlook of different optical flow methods, the obstacle avoidance system, as well as the State of Art and evaluation of optical flow fields. With all of this information, a natural question arises: "How to select the more appropriate optical flow algorithm for visual navigation?"

In relation to the optical flow techniques, even though each method has its own benefits and limitations, the majority of successful traditional methods choose variational frameworks and resort to the energy minimisation scheme. From the Middlebury [5], MPI-Sintel [13] and KITTI [31]

Figure 4.10: Website page with Rotate, Capture and Refresh button for commands to the ESP32-Cam

benchmarks, it is obvious that globally regularised models based on joint estimation, segmentation and approximate nearest neighbour field, can produce improved results and perform well for datasets containing small displacements, moderate intensity changes and piece wise smooth displacements of large structures. Alternatively, for the image sequences offering more challenging situations, these techniques remain competitive. Hence, the need for further improvements to overcome shortcomings and the foremost problems associated with each method, especially regarding the computational time.

Since the main goal of this work is to develop and study optical flow techniques, the algorithms mentioned in Section 2 will be implemented in Python and applied in a dataset to take and study the evaluation metrics (EE, AE, AEE, AAE, R0.5, R1.0, R2.0. R5.0) and performance of each algorithm, like computational resources and time. Optimisation of optical flow algorithms are not going to be taken in account, even though a comparison to other previous work (like [8] will be made.

Since the context and application of the different datasets was analysed before, it's important to analyse each dataset. The Middlebury dataset is considered to be selected since the ground truth is available. In more technical terms, it is a leading benchmark that addresses advanced level problems, covering evaluation at a broader spectrum and a wider range of statistical measures [5]. Comprising sub pixel ground truth and ample difficulty, these are realistic synthetic sequences with non-rigid motions, complex scenes and higher texture. However, the motions are small as compared to more advance datasets [13, 31]. In contrast to the previously used synthetic sequences

(Yosemite [8]), these datasets are considerably challenging presenting more complex scenes, larger ranges, higher realistic texture and independent motion.

Another important aspect to consider is the fact that the robot moves and in consequence all the pixels in the frame move in respect to the movement. The global techniques, that are based on brightness constancy and smoothness assumptions, are the most appropriate to the visual navigation system.

One equal important aspect of the evaluation is that for the navigation system, a dense optical flow for a obstacle avoidance system is needed since the goal of the control laws is to balance both sides of the optical flow field. Since the sparse optical flow doesn't have 100% density. This type of optical flow results maybe be suited for another control law, but trying to balance optical flow values may result in poor results.

Even though the analysis of the datasets is important to the implementation of optical flow in the visual navigation system, the "real" aspect of this kind of navigation is lost. That is, synthetic and real, but not contextualised sequence of frames, lose important aspects of the robot navigation, like the texture of the ground.

For this, a set of sequences of frames with different conditions like angle to the ground, height, types of ground (flat, rug, etc.), light, variation of light, and quality of frame will be created with the use of the ESP32-Cam module.

Finally, with all of this information, a implementation on a ES32-Cam will be implemented with the Horn and Schunck algorithm. The algorithms under study are the Lucas-Kanade with and without image pyramids, Horn and Schunck and SAD Blocking Matching.

4.3.1 Evaluation of Traditional Optical Flow Methods on Datasets

One of the main challenges of analysing optical flow algorithms is interpreting and comparing results. That is, the optical flow values should evaluate a type of movement, occlusion, or other kinds of visual variation, and it should be possible to evaluate the algorithm's efficiency and results when compared to the motion that occurred with other algorithms. That's exactly the reason why the use of the algorithms developed was tested in frames belonging to datasets where the ground truth is known. This way, it is possible to characterise the algorithms in terms of the evaluation metrics mentioned previously, like the end-point error (EE) and angular error (AE).

The dataset used is the Middlebury dataset [5]. This dataset was selected since the ground truth for training is available, and the motion characteristics vary. Even though it's not possible to compare the results with the public benchmarks without submitting, it is not problematic since the point of this work is not to create a State-of-Art optical flow algorithm (every algorithm implemented is traditional). The results will only indicate if the algorithm has overall good results and which of them performs better in terms of the selected metrics, process time and memory used.

Although, if necessary to submit any values in the future, a python program was made to convert the optical flow fields u and v into $.flo$ files to submit publicly. In Figure 4.11 it's possible to see a public benchmark of optical flow evaluation for end-point error (average).

Optical flow evaluation results

Show images: below table above table in window

Statistics: Average SD R0.5 R1.0 R2.0 A50 A75 A95
Error type: endpoint angle interpolation normalized interpolation

Average endpoint error	avg. rank	Army (Hidden texture) GT im0 im1			Mequon (Hidden texture) GT im0 im1			Schefflera (Hidden texture) GT im0 im1			Wooden (Hidden texture) GT im0 im1			Grove (Synthetic) GT im0 im1			Urban (Synthetic) GT im0 im1			Yosemite (Synthetic) GT im0 im1			Teddy (Stereo) GT im0 im1		
		all	disc	untext	all	disc	untext	all	disc	untext	all	disc	untext	all	disc	untext	all	disc	untext	all	disc	untext	all	disc	untext
NNF-Local [75]	5.1	**0.07** 1	0.20 2	**0.05** 1	0.15 2	0.51 5	0.12 6	0.18 2	0.37 2	0.14 2	0.10 3	0.49 9	0.06 3	**0.41** 1	**0.61** 1	0.21 2	0.23 4	0.66 5	0.19 4	0.10 15	0.12 14	0.17 27	0.34 2	0.80 7	0.23 2
PMMST [112]	12.3	0.09 35	0.21 5	0.07 21	0.18 14	0.51 5	0.16 36	0.21 9	0.42 8	0.17 22	0.10 3	0.33 3	0.08 15	0.51 5	0.74 4	0.28 8	0.24 5	0.65 4	0.20 7	0.11 32	0.12 14	0.17 27	0.37 5	0.74 3	0.35 6
RAFT-TF_RVC [179]	12.4	0.10 52	0.30 77	**0.05** 1	0.18 14	0.55 12	0.14 24	0.21 9	0.43 9	0.19 46	**0.06** 1	**0.23** 1	**0.04** 1	0.51 5	0.75 6	0.25 4	**0.14** 1	**0.42** 1	**0.11** 1	0.07 2	0.12 14	**0.08** 1	0.37 5	0.80 7	0.27 3
OFLAF [78]	13.6	0.08 9	0.21 5	0.06 7	0.16 7	0.53 7	0.12 6	0.19 3	0.37 2	0.14 2	0.14 14	0.77 41	0.07 8	0.51 5	0.78 8	0.25 4	0.31 16	0.76 6	0.25 22	0.11 32	0.12 14	0.21 62	0.42 13	0.78 5	0.63 29
MDP-Flow2 [68]	14.2	0.08 9	0.21 5	0.07 21	0.15 2	**0.48** 1	**0.11** 1	0.20 5	0.40 5	0.14 2	0.15 31	0.80 50	0.08 15	0.63 23	0.93 23	0.43 26	0.26 8	0.76 6	0.23 13	0.11 32	0.12 14	0.17 27	0.38 8	0.79 6	0.44 9
NN-field [71]	15.5	0.08 9	0.22 16	**0.05** 1	0.17 11	0.55 12	0.13 14	0.19 3	0.39 4	0.15 7	0.09 2	0.48 8	0.05 2	**0.41** 1	**0.61** 1	**0.20** 1	0.52 73	0.64 3	0.26 27	0.13 62	0.13 46	0.20 56	0.35 3	0.83 10	**0.21** 1
ComponentFusion [94]	18.7	**0.07** 1	0.21 5	**0.05** 1	0.16 7	0.55 12	0.12 6	0.20 5	0.44 10	0.15 7	0.11 7	0.65 14	0.06 3	0.71 45	1.07 50	0.53 49	0.32 20	1.06 36	0.28 30	0.11 32	0.13 46	0.15 19	0.41 11	0.88 16	0.54 17
CoT-AMFlow [174]	21.5	0.08 9	0.22 16	0.07 21	0.15 2	**0.48** 1	0.12 6	0.21 9	0.45 11	0.15 7	0.16 45	0.86 64	0.08 15	0.67 35	0.96 29	0.56 54	0.27 10	0.82 11	0.24 18	0.12 48	0.12 14	0.18 43	0.42 13	0.85 11	0.60 24
TC/T-Flow [77]	25.6	**0.07** 1	0.21 5	**0.05** 1	0.19 23	0.68 39	0.12 6	0.28 32	0.66 38	0.14 2	0.14 14	0.86 64	0.07 8	0.67 35	0.98 36	0.49 42	0.22 3	0.82 11	0.19 4	0.11 32	0.11 2	0.30 112	0.50 35	1.02 37	0.64 32
PRAFlow_RVC [177]	26.2	0.11 65	0.27 58	0.08 52	0.24 69	0.64 30	0.19 63	0.28 32	0.61 28	0.23 75	0.12 11	0.62 13	0.06 3	0.60 15	0.87 14	0.38 15	0.18 2	0.50 2	0.16 2	0.07 2	0.12 14	**0.08** 1	0.49 29	0.92 21	0.51 13
WLIF-Flow [91]	26.3	0.08 9	0.21 5	0.06 7	0.18 14	0.55 12	0.15 30	0.25 22	0.56 25	0.17 22	0.14 14	0.68 17	0.08 15	0.61 19	0.91 20	0.41 23	0.43 47	0.96 20	0.29 36	0.13 62	0.12 14	0.21 62	0.51 41	1.03 41	0.72 55
UnDAF [187]	27.2	0.09 35	0.26 48	0.07 21	0.16 7	0.53 7	**0.11** 1	0.22 12	0.48 14	0.15 7	0.17 59	0.93 85	0.08 15	0.70 43	1.04 45	0.48 37	0.29 13	0.92 16	0.24 18	0.12 48	0.12 14	0.18 43	0.45 19	0.95 25	0.58 21
NNF-EAC [101]	28.0	0.09 35	0.22 16	0.07 21	0.17 11	0.53 7	0.13 14	0.23 14	0.49 15	0.15 7	0.16 45	0.80 50	0.09 39	0.60 15	0.89 16	0.40 20	0.38 34	0.78 8	0.28 30	0.12 48	0.12 14	0.18 43	0.57 60	1.24 65	0.69 46
Layers++ [37]	28.7	0.08 9	0.21 5	0.07 21	0.19 23	0.56 16	0.17 42	0.20 5	0.40 5	0.18 34	0.13 12	0.58 12	0.07 8	0.48 3	0.70 3	0.33 16	0.47 59	1.01 26	0.33 60	0.15 91	0.14 74	0.24 77	0.46 23	0.88 16	0.72 55
IROF++ [58]	29.5	0.08 9	0.23 24	0.07 21	0.21 37	0.68 39	0.17 42	0.28 32	0.63 31	0.19 46	0.15 31	0.73 30	0.09 39	0.60 15	0.89 16	0.42 25	0.43 47	1.08 39	0.31 47	0.10 15	0.12 14	0.12 10	0.47 25	0.98 29	0.68 45
LME [70]	29.8	0.08 9	0.22 16	0.06 7	0.15 2	0.49 3	**0.11** 1	0.30 43	0.64 33	0.31 105	0.15 31	0.78 45	0.09 39	0.66 30	0.96 29	0.53 49	0.33 21	1.18 57	0.28 30	0.12 48	0.12 14	0.18 43	0.44 16	0.91 20	0.61 25
nLayers [57]	30.8	**0.07** 1	**0.19** 1	0.06 7	0.22 46	0.59 20	0.19 63	0.25 22	0.54 21	0.20 57	0.15 31	0.84 58	0.08 15	0.53 8	0.78 8	0.34 13	0.44 52	0.84 13	0.30 43	0.13 62	0.13 46	0.20 56	0.47 25	0.97 28	0.67 43
ProFlow_ROB [142]	31.7	0.08 9	0.26 48	0.06 7	0.18 14	0.70 47	0.13 14	0.31 50	0.80 64	0.17 22	0.14 14	0.75 36	0.06 3	0.80 61	1.21 67	0.49 42	0.33 21	1.18 57	0.20 7	0.07 2	0.13 46	0.13 17	0.50 35	1.19 57	0.58 21
HAST [107]	32.1	**0.07** 1	0.20 2	**0.05** 1	0.18 14	0.54 10	0.13 14	**0.17** 1	**0.32** 1	**0.12** 1	0.15 31	0.90 77	0.06 3	0.49 4	0.74 4	0.22 3	0.58 86	1.09 41	0.44 89	0.19 123	0.17 113	0.47 144	**0.32** 1	**0.64** 1	0.33 6
FC-2Layers-FF [74]	32.9	0.08 9	0.21 5	0.07 21	0.21 37	0.70 47	0.17 42	0.20 5	0.40 5	0.18 34	0.15 31	0.76 39	0.08 15	0.53 8	0.77 7	0.37 14	0.49 65	1.02 28	0.33 60	0.16 101	0.13 46	0.29 107	0.44 16	0.87 15	0.64 32
PH-Flow [99]	32.9	0.08 9	0.24 33	0.07 21	0.21 37	0.68 39	0.17 42	0.23 14	0.49 15	0.19 46	0.16 45	0.83 56	0.09 39	0.56 11	0.83 11	0.38 15	0.30 14	0.81 9	0.24 18	0.15 91	0.13 46	0.30 112	0.43 15	0.85 11	0.66 40

Figure 4.11: Public Benchmark for Optical Flow Evaluation for end-point error (average), Date: 12/07/2022 [credit: Middlebury]

This comparison and application on datasets, in this case Middlebury, can also inform about the potentialities of optical flow, since the results of traditional methods can't compete in terms of evaluation metrics in relation to state-of-the-art algorithms, like the ones shown in the State of the Art Section. This informs that in the proposed visual navigation system, the main component, the optical flow calculation, can be improved in terms of speed and accuracy in order to enable a more "real-time" system. Even though the memory usage can be much higher and for a micro controller like the ESP32-CAM can be very difficult to implement. This "trade-off" between accuracy/speed and memory is a subject of future investigation.

The four algorithms mentioned previously are going to be implemented and their results evaluated in terms of the metrics of end-point error (EE), angular error (AE), R0.5, R1.0, R2.0, R3.0, R5.0, that represent the percentage of end-point error bigger than 0.5, 1.0, 2.0, 3.0, 5.0, respectably (the last two metrics are used to evaluate outliers), process time in seconds and memory used in Mbit. There was no attempt to normalise for the programming environment, CPU speed, number of cores, or other hardware acceleration. The values of results should be treated as a very rough guideline of the inherent computational complexity of the algorithms.

4.3.2 Evaluation of Traditional Optical Flow Methods on Floor Sequences

After the evaluation of the algorithms in the datasets, the evaluation of floor sequences is required since the datasets don't represent two very important aspects of the visual navigation system.

The first is the characteristics of the camera (OV246) like the noise, the resolution, the response to the light variation, all important aspects for the navigation system. The other are the frames themselves, since some of the frames are synthetic and most of them don't have the characteristics of real-life situations. Additionally, there are no databases for photos taken with the specific circumstances needed and with the ESP32-Cam.

The first experiment consisted in testing some of the limitations of ESP32-Cam module when placed in a specific height and with a specific resolution. These two parameters in the computation of optical flow and visual odometry, are crucial and will be a focus of this study.

For this, a range of a possible height was defined from the start as being 21cm, the distance close to the point where the ESP32-Cam can't focus (the minimum object distance is approximately 19.05 centimetres) and 40cm, a height where the field of view, due to the focal length of the ESP32-Cam could start to capture robot chassis and a lot of navigation information would be a potential lost. In terms of ESP32-Cam resolutions, they are represented in Table 4.3. In the first part, of the experiment, only the resolution QVGA and VGA were studied, since these resolutions are high enough to compute optical flow and since a bigger resolution would increase substantially the runtime of the algorithms.

Name of Resolution	*Resolution (pixels)*
UXGA	1600 x 1200
QVGA	320 x 240
CIF	352 x 288
VGA	640 x 480
XGA	1024 x 768
SXGA	1280 x 1024

Table 4.3: Resolutions available in ESP32-Cam Espressif Firmware

The first part of the experiment consists in attaching the ESP32-Cam to a height of 21cm and maintaining a constant illuminance of approximately 500 lux. A board eraser was placed in the floor in order to measure the field of view. This information will allow to define an approximately a maximum distance which the optical flow can be theoretically measured and also a first example of the calibration of the ESP32-Cam. The measurements are represented in Table 4.4.

This experiment will also be used to determine the influence of an obstacle in the calculation of the optical flow and the model for the pixel/distance relation.

The next experiment consisted in computing the optical flow with a plain floor. The characteristics of the experiments are represented in the Table 4.5.

The total number of samples used to define the noise and distribution of the average were $10 \times 4 = 40$, since between every displacement ten calculations were made and since there is a dislocation until 3 cm, the total number is 40.

It's worth noting that there are two types of variation in relation to the measurement of the average distribution in these experiments. The variation in terms of noise between photos in the same place, that will computed with photos in the other position with different noise levels. The other factor is the variation in place of the different photos taken. That is, the photos of VGA_21cm_0cm and VGA_21cm_1cm when computing the optical flow, will have a different distribution of u and v components compared to VGA_21cm_1cm and VGA_21cm_2cm, because

Group and number of photos	Height	Resolution	Dislocation
VGA_21cm_0cm (5 photos)	21cm	VGA	0 cm (none)
QVGA_21cm_0cm (5 photos)	21cm	QVGA	0 cm (none)
VGA_21cm_1cm (5 photos)	21cm	VGA	1cm (right to left)
QVGA_21cm_1cm (5 photos)	21cm	QVGA	1cm (right to left)
VGA_21cm_2cm (5 photos)	21cm	VGA	2cm (right to left)
QVGA_21cm_2cm (5 photos)	21cm	QVGA	2cm (right to left)
VGA_21cm_3cm (5 photos)	21cm	VGA	3cm (right to left)
QVGA_21cm_3cm (5 photos)	21cm	QVGA	3cm (right to left)
VGA_21cm_14cm (5 photos)	21cm	VGA	14cm (right to left)
QVGA_21cm_14cm (5 photos)	21cm	QVGA	14cm (right to left)
VGA_21cm_15cm (5 photos)	21cm	VGA	15cm (right to left)
QVGA_21cm_15cm (5 photos)	21cm	QVGA	15cm (right to left)

Table 4.4: Group of photos taken for the first experiment (board eraser in the same position, moving ESP32-Cam module)

Group and number of photos	Height	Resolution	Dislocation
VGA_21cm_0cm (10 photos)	21cm	VGA	0 cm (none)
QVG_21cm_0cm (10 photos)	21cm	QVGA	0 cm (none)
VGA_21cm_1cm (10 photos)	21cm	VGA	1cm (right to left)
QVGA_21cm_1cm (10 photos)	21cm	QVGA	1cm (right to left)
VGA_21cm_2cm (10 photos)	21cm	VGA	2cm (right to left)
QVGA_21cm_2cm (10 photos)	21cm	QVGA	2cm (right to left)
VGA_21cm_3cm (10 photos)	21cm	VGA	3cm (right to left)
QVGA_21cm_3cm (10 photos)	21cm	QVGA	3cm (right to left)

Table 4.5: Group of photos taken for the second experiment (plain floor)

the patterns in the floors are different and so the components of the optical flow will not be exactly the same.

The next experiment will use the same configuration, but with the dislocation of the ESP32-Cam made in a parallel way in relation to the last experiment. This will expose one limitation of the optical flow, since the pattern of the floor doesn't seem to change from photo to photo. The experiments are represented in Table 4.6.

Less photos were taken from this experiment since the goal of this experiment wasn't to define the distribution of average values and the model, but to demonstrate the aperture problem.

In relation to group of photos taken 4.5, this experiment has a change in the type of floor. The new floor is a rug with some patterns. With the results of the experiments in Table 4.5, it will be possible to define the correspondence between the number of pixels of the horizontal component and the dislocation in cm of the ESP32-Cam module. The experiments are represented in Table 4.7.

This experiment was briefly repeated with a uniform pattern rug, in order to reinforce the

Group and number of photos	Height	Resolution	Dislocation
VGA_21cm_0cm (5 photos)	21cm	VGA	0 cm (none)
QVG_21cm_0cm (5 photos)	21cm	QVGA	0 cm (none)
VGA_21cm_1cm (5 photos)	21cm	VGA	1cm (right to left)
QVGA_21cm_1cm (5 photos)	21cm	QVGA	1cm (right to left)
VGA_21cm_2cm (5 photos)	21cm	VGA	2cm (right to left)
QVGA_21cm_2cm (5 photos)	21cm	QVGA	2cm (right to left)
VGA_21cm_3cm (5 photos)	21cm	VGA	3cm (right to left)
QVGA_21cm_3cm (5 photos)	21cm	QVGA	3cm (right to left)
VGA_21cm_3cm (5 photos)	21cm	VGA	3cm (right to left)
QVGA_21cm_3cm (5 photos)	21cm	QVGA	3cm (right to left)

Table 4.6: Group of photos taken for the third experiment (plain floor, different orientation)

Group and number of photos	Height	Resolution	Dislocation
VGA_21cm_0cm (10 photos)	21cm	VGA	0 cm (none)
QVG_21cm_0cm (10 photos)	21cm	QVGA	0 cm (none)
VGA_21cm_1cm (10 photos)	21cm	VGA	1cm (right to left)
QVGA_21cm_1cm (10 photos)	21cm	QVGA	1cm (right to left)
VGA_21cm_2cm (10 photos)	21cm	VGA	2cm (right to left)
QVGA_21cm_2cm (10 photos)	21cm	QVGA	2cm (right to left)
VGA_21cm_3cm (10 photos)	21cm	VGA	3cm (right to left)
QVGA_21cm_3cm (10 photos)	21cm	QVGA	3cm (right to left)

Table 4.7: Group of photos taken for the fourth experiment (rug)

idea that the patterns in the floor influence greatly the quality of the optical flow. The table of experiments is similar to Table 4.6.

This experiment ends the first batch of tests in order to test the viability of visual odometry using optical flow and the ESP32-Cam at 21cm from the ground. With all this information is possible to create a model (distribution) that defines the average of the number of pixels that represent a certain change in position and how much in centimetres.

Only horizontal measurements were made, but the vertical response was extrapolated from the measurements from the horizontal tests (all the experiments until this point).

The next phase of experiments consisted in changing the height of the ESP32-Cam. First, the model used previously was tested in order to identify if it needed some adjustments or the change in parameter (height) didn't affect substantially its precision.

4.4 Practical Implementation of Visual Odometry System in Real-Time

From this point onward, the visual navigation system is the focus of the study. There are two strategies that could be implemented.

The first consists in a system that sends the photos to a web server and then downloaded to the PC. This system starts with ESP32-Cam ready to send photos to the local web server at the request of the Python program. The python program is structured in two cycles where in the first, a Selenium function sends the CAPTURE PHOTO command to the ESP32-Cam and updates the web server when the photo is sent.

After that, the photo is saved and when there are two recent photos, taken subsequently, the optical flow is computed and the result analysed. In Figure 4.12, there is a representation of this system.

This system has the advantage of using the computational power of the computer in order to compute optical flow and save the results. It depends on the internet connection and has the drawback of being possibly much slower, even though the computational power of a regular computer in order of magnitude is higher than the one from the ESP32-CAM. This approach is easier to implement and is specially useful in the test phase of the work.

The second consists in an embedded system in the ESP32-Cam where a similar system is applied, in terms of computing the optical flow from recent images, but no information (images) is sent to the computer. All the image data is processed in the ESP32-Cam and the optical flow vectors are computed by implementing the algorithm in the ESP32-Cam. The ESP32-Cam module also computes the average optical flow vectors and the displacement/velocity of the robot. The final step is to send the information to the robot. This system is a visual odometry implementation only using the ESP32-Cam, even though the last two steps of the optical flow cycle, average optical flow vectors computation and computation of robot's displacement/velocity, can be computed in the robot. The cycle continues until the ESP32-Cam is no longer used and the connection to the robot is made again. In Figure 4.13, a representation of this system is shown.

This system has the advantage of possibly being faster and utilise all the resources from the ESP32. In the disadvantages of using this system, it can be mentioned the higher complexity of implementing an algorithm in the ESP32-Cam, since it's necessary to conjugate the usage of the ESP32 firmware and the adoption of the Python's algorithm to the Arduino language (C++ and C). Another important point is that the flash memory of the ESP32 is limited.

Finally, it's important to define the relation between optical flow vectors and displacement. As mentioned previously, a pixel displacement in a two subsequent images relates to a displacement in a point in the 3D scene. It's possible to use several models in order to relate the 3D and the 2D information. But, since it is possible to compute the optical flow vectors average (Δu and Δv) and knowing the displacement of the ESP32-Cam, it is possible to create an approximately linear relation between displacement and the average optical flow vector.

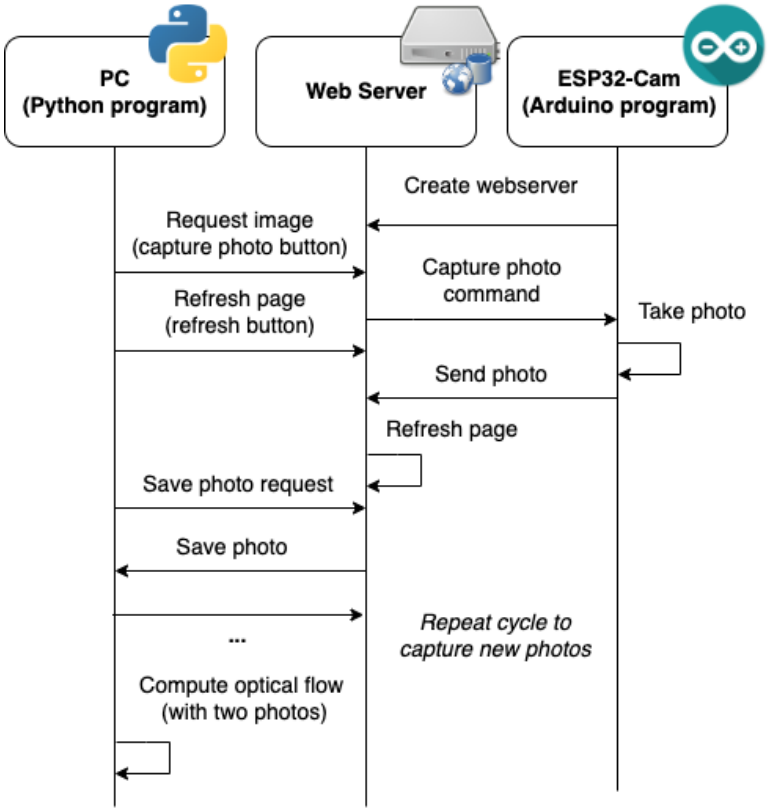


Figure 4.12: Scheme of real time implementation for optical flow implementation using the computer, a local web server and the ESP32-Cam

In order to obtain the average value of a determined distance, like 1cm for example, it is necessary to analyse two types of variation in relation to the measurement of the average distribution in these experiments. The variation in terms of noise between photos in the same place, that was computed with photos in the other position with different noise levels. The other factor is the variation in place of the different photos taken. This is important since the patterns in the floors are different and so the components of the optical flow will not be exactly the same, and it is needed to take in account the noise response.

So, knowing the displacement of the robot, using the relationship between average optical flow vectors and displacement, and the runtime of the algorithm, it is possible to determine the velocity of the robot, using the Equation 4.32.

$$V = \frac{\Delta x}{t} \tag{4.32}$$

Being Δx the displacement, either in x or y coordinate, and t is the time of the the runtime of the optical flow algorithm

In relation to the practical implementation of the embedded optical flow system in the ESP32-Cam, it is necessary to mention that one of the main problem is that it is necessary to work with

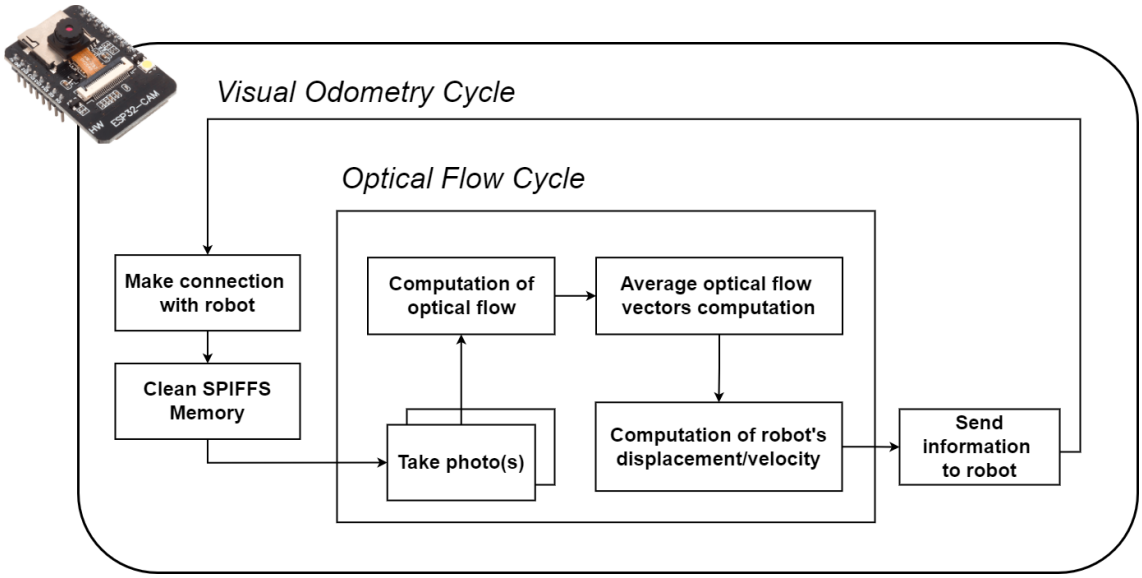


Figure 4.13: Scheme of real time implementation for optical flow implementation using only the ESP32-Cam

a framework that will enable the usage of a camera feature and optical flow computation. In the first attempt of the implementation, a MicroPython environment was created, in order to reuse the code of the algorithms in Python but the ESP32-Cam module didn't have the necessary memory to work.

A second attempt consisted in implementing the OpenCV library and use the optical flow algorithms available in that Library. Some of the feature track function of OpenCV worked, but it wasn't possible to implement the algorithms using the OpenCV library.

So, the final attempt consisted in using the firmware from the previous real-time system and write the algorithms in C/C++ where it could be adapted to the Arduino framework. It was possible to study the memory usage of the system, with the C/C++ program, and with the Arduino platform implementation in the real-time system.

Chapter 5

Results

In this chapter, the results from the last Section will be interpreted and analysed. The Results Section follow exactly the same structure of the last Section. First the results of the dataset (Middlebury) will be shown and subsequently, the results of the floor sequences. The real-time computation of the optical flow results and the visual odometry system are also shown in this Chapter. For the experiments, all the tests were made with the machine with the following characteristics:

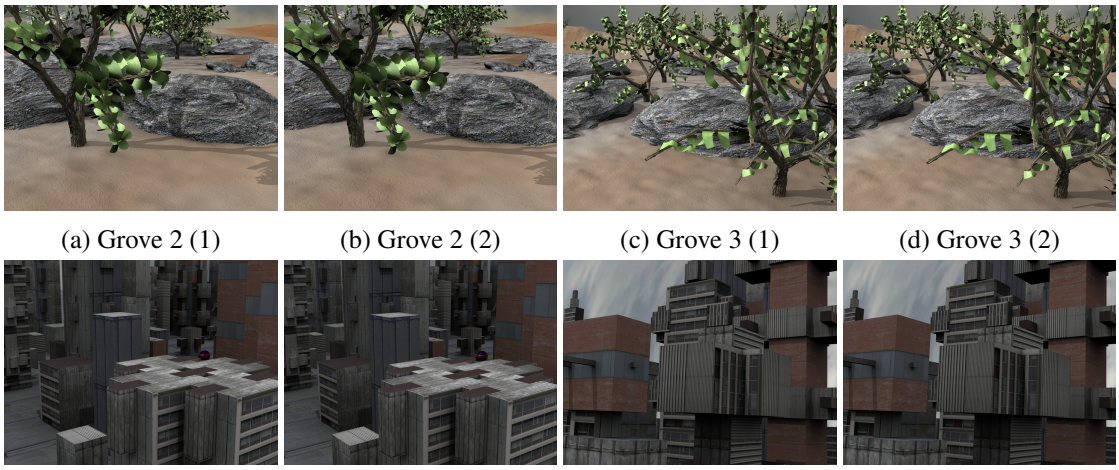
Model	Aspire E5-575G
Processor	Intel(R) Core(TM) i5-7200U CPU @ 2.50 GHz 2,70 GHz
Memory (RAM)	16 GB (15,9 GB usable)
Kind of System	Operating System of 64 bits, processor based on x64

Table 5.1: Machine (Computer) Characteristics

Some runtime and memory could be influenced by other processes running on the machine, but this aspect was taken into account and avoided.

5.1 Results of Robot Localisation with Traditional Optical Flow Methods

In this Section the results of the optical flow algorithms are going to be presented. This includes the dataset, floor sequences and real-time calculations of the optical flow algorithms. The results are shown and analysed simultaneously, being the last part of the analysis focused on error causes and improvements.



(a) Grove 2 (1) (b) Grove 2 (2) (c) Grove 3 (1) (d) Grove 3 (2)

(e) Urban2 (1) (f) Urban2 (2) (g) Urban3 (1) (h) Urban3 (2)

Figure 5.1: Synthetic Frames of analysed sequence of Middlebury with public ground-truth [credit: Middlebury]

5.1.1 Middlebury Dataset

For the first part of the experiments, the optical flow algorithms were applied on the Middlebury dataset. The first experiments were performed on the synthetic frames 5.1 and then the real imagery frames. Both sets have public ground-truths.

In relation to the datasets, the first used dataset was the Middlebury dataset. The group of photos analysed were the Grove 2, Grove 3, Urban 2, Urban 3.

In the Middlebury dataset, the common elements of the most difficult sequences (like Grove and Urban) are the presence of large motions and strong motion discontinuities. The complex discontinuities and fine structures of Grove seem to cause the most problems for the current algorithms. Grove contains a close-up view of a tree, with a substantial parallax and motion discontinuities (Fig.5.1g, Fig.5.2h). The Urban contains scenarios of synthetic urban buildings.

As mentioned previously, the α can be varied from application to application. In this context, three types of α were used (α = 30, 60, 120). The number of iterations was set to 20, in order to perform competitively in relation to other algorithms, since the increase of number of iterations reflects in an increase in computational time. For a number of iterations of 300, the average computational time was more than a minute (60 seconds), which is not ideal for visual navigation given that the computational time is of great importance. In tables 5.2, 5.3, A.1, A.2, A.3, A.4, it is possible to see the results of the application of the Horn and Schunck algorithm.

The results of the Horn and Schunck (Tables 5.2, 5.3) were as expected by the public dataset for the same algorithm. Clearly the runtime is still not ideal for the real-time system, since the average value is around 5 seconds. For the Urban 2 and Urban 3 sequences, the AEE was the metric that got the worst results, having really a high end-point error (EE) standard deviation. This represents a significant number of outliers, specially in the metric R5.0.

In the Lucas-Kanade results (Tables 5.4, 5.5), the results are very similar to the Horn-Schunck

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	3.11056	0.52697	1.27204	0.06458	4.68599	109.352
Grove 3	3.94084	2.29362	1.24937	0.24877	5.52953	109.738
Urban 2	8.39806	8.07650	1.21764	0.35352	4.75989	109.270
Urban 3	7.31328	4.36791	1.38071	0.12585	4.95893	109.488

Table 5.2: Results of Horn and Schunck with $\alpha = 120$ and 20 iterations Middlebury Dataset Synthetic Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	42.97949	0.15625
Grove 3	98.22493	92.79915	80.51399	61.39453	26.79622
Urban 2	94.35156	83.75488	73.40950	64.12369	40.14811
Urban 3	100	100	95.50390	89.44433	56.12434

Table 5.3: Results of Horn and Schunck with $\alpha = 120$ and 20 iterations Middlebury Dataset Synthetic Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	3.59113	0.36745	1.61564	0.26924	0.12735	125.637
Grove 3	3.60032	2.10409	0.98086	0.52840	0.14775	127.004
Urban 2	8.55061	8.25367	1.29892	0.51587	0.15395	125.441
Urban 3	6.81390	4.40718	1.00473	0.28544	0.12507	125.289

Table 5.4: Results of Lucas Kanade without Pyramids with window size equal to 15 from Middlebury Dataset Synthetic Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	96.79036	0.05566
Grove 3	98.22493	92.79915	80.51399	61.39453	26.79622
Urban 2	98.15429	88.26595	68.91048	60.80436	44.92154
Urban 3	99.14257	97.375	92.11946	81.49609	51.51757

Table 5.5: Results of Lucas Kanade without Pyramids with window size equal to 15 from Middlebury Dataset Synthetic Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

algorithm, analysed previously. But, one very important aspect is that the runtime is significantly lower then the average of the Horn and Schunck algorithm. This is crucial for real-time system applications. Since the results are similar, it's a better algorithm overall.

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	6.157622	4.05180	1.26592	0.24498	0.96119	148.594
Grove 3	5.30792	3.62931	0.94529	0.50184	1.30655	149.137
Urban 2	9.02752	7.95760	1.08495	0.47640	1.41928	148.102
Urban 3	5.53856	4.31995	0.74677	0.59599	1.11706	148.652

Table 5.6: Results of Lucas Kanade with Pyramids with window size equal to 15 and 5 figures from Middlebury Dataset Synthetic Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	80.54589	40.86425
Grove 3	98.58984	93.16894	83.74414	70.4547526	42.09147
Urban 2	95.47493	83.43424	75.05403	67.02929	51.93554
Urban 3	98.99348	96.13606	84.37760	71.64518	71.64518

Table 5.7: Results of Lucas Kanade with Pyramids with window size equal to 15 from Middlebury Dataset (R0.5, R1.0. R2.0, R3.0, R5.0) Synthetic Frames

In the Lucas-Kanade with Pyramids (Tables 5.6, 5.7), the results are similar to the Lucas-Kanade without Pyramids in terms of runtime, but considerably worst in terms of outliers, like the R5.0 metric. It's also important to mention that the memory used is slightly higher than the Lucas-Kanade algorithm without pyramids. This result can be explained by the fact that displacement of the objects in the scene are not large displacement. The Lucas-Kanade with Pyramids is a way to compensate for the large displacements that are not well computed in the original Lucas-Kanade algorithm.

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	3.08896	0.51526	1.25095	0.04715	3.98584	68.797
Grove 3	3.91333	2.29184	1.22222	0.24122	3.656568	68.891
Urban 2	8.39281	8.07619	1.21260	0.35285	3.70195	68.727
Urban 3	7.30574	4.36940	1.37347	0.12815	4.07583	68.895

Table 5.8: Results of Block Matching with window size equal to 51, stride of 21 and shift of 1 from Middlebury Dataset Synthetic Frames

Finally, the Block Matching algorithm (Tables 5.8, 5.9) is the one that uses the least memory. The runtime is on average 4 seconds and the results, in terms of AEE, std EE, AAE, std AE and R metrics are similar to the Horn-Schunck, and better than the Lucas-Kanade algorithms. This runtime can also not be ideal for real-time optical flow and displacement calculations. It's also important to mention that this algorithm is the one that has the highest number of parameters, being

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	41.13378	0.01041
Grove 3	98.17675	92.62825	80.15234	60.68782	26.63444
Urban 2	94.28808	83.73437	73.40592	64.04622	40.08756
Urban 3	100	100	95.49414	89.01822	56.06250

Table 5.9: Results of Block Matching with window size equal to 51, stride of 21 and shift of 1 from Middlebury Dataset Synthetic Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

the one that is more difficult to optimise, even though it's evident that some choices contribute to the runtime increase. In this example, the window size (51) can be considered to big, in relation to the window size of Lucas-Kanade algorithm (15) for example.

The real imagery sequences of the Middlebury dataset will be the next in study. These figures can be seen in Fig. 5.2. The group of photos analysed were the Hydrangea, Dimetrodon, RubberWhale, and Venus.

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AAE (degrees)*	*Process Time (seconds)*	Memory (MiB)
Hydrangea	3.74387	1.15899	1.30732	0.15533	3.47822	101.051
Dimetrodon	2.06779	0.69379	1.11571	0.17145	3.40710	100.938
RubberWhale	1.26325	0.48444	0.88485	0.17469	3.35218	100.789
Venus	3.81596	1.79347	1.25497	0.21892	2.46839	93.211

Table 5.10: Results of Horn and Schunck with $\alpha = 120$ and 20 iterations of Middlebury Dataset Real Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Hydrangea	99.62448	99.62448	91.12568	84.41137	84.41137
Dimetrodon	99.99953	97.51922	48.40746	13.92410	0.0
RubberWhale	98.50069	74.73920	5.30923	1.66748	0.0
Venus	98.12406	96.18609	96.18609	63.92042	27.89473

Table 5.11: Results of Horn and Schunck with $\alpha = 120$ and 20 iterations of Middlebury Dataset Real Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

In general, the results presented in the real imagery sequences are very similar to the ones of the synthetic sequences. This is important since the synthetic sequence does not transmit important information about the application of these algorithms in real scenarios. It's important to mention that even though they are real imagery, the settings of these results are very well optimised and don't reflect characteristics of robot navigation, like small occlusions and changes in light. For

(a) Grove 2 (1) (b) Grove 2 (2) (c) Grove 3 (1) (d) Grove 3 (2)

(e) Urban2 (1) (f) Urban2 (2) (g) Urban3 (1) (h) Urban3 (2)

Figure 5.2: Synthetic Frames of analysed sequence of Middlebury with public ground-truth [credit: Middlebury]

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AAE (degrees)*	*Process Time (seconds)*	Memory (MiB)
Hydrangea	3.26981	1.08577	0.94961	0.36125	0.10641	109.582
Dimetrodon	2.65991	0.71765	1.55434	0.26047	0.10812	109.688
RubberWhale	1.19456	0.73513	0.76463	0.43680	0.24974	109.660
Venus	3.62651	1.67664	1.12247	0.44729	0.14290	93.211

Table 5.12: Results of Lucas Kanade without Pyramids with window size equal to 15 from Middlebury Dataset Real Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Hydrangea	99.52482	97.34545	91.08175	76.27390	5.404984
Dimetrodon	100.0	99.56723	80.29932	33.90325	0.151051
RubberWhale	81.61053	57.24043	57.24043	2.60752	0.20899
Venus	99.49185	96.87844	84.75626	56.36842	27.74310

Table 5.13: Results of Lucas Kanade without Pyramids with window size equal to 15 from Middlebury Dataset Real Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

this, the floor sequence dataset was created in order to test the algorithms in frames taken by a camera (OV2640) in close to real scenarios.

This study was mainly conducted in order to understand the functioning and characteristics of the four algorithms created (Horn-Schunck, Lucas-Kanade without Pyramids, Lucas-Kanade with Pyramids, and Block Matching). In order to understand if the algorithms were working correctly it was necessary to test them in a known dataset with a public ground-truth. In this aspect, all the tests were successes, since they produced valid results with varying degrees of success in terms of

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	6.51345	4.34513	1.13455	0.23466	0.95751	148.594
Grove 3	5.30345	3.62671	0.94367	0.50167	1.33567	149.137
Urban 2	9.02673	7.93673	1.08567	0.47760	1.45637	148.102
Urban 3	5.53567	4.31967	0.74677	0.59596	1.11676	148.652

Table 5.14: Results of Lucas Kanade with Pyramids with window size equal to 15 and 5 figures from Middlebury Dataset Real Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	80.54589	40.86425
Grove 3	98.58984	93.16894	83.74414	70.4547526	42.09147
Urban 2	95.47493	83.43424	75.05403	67.02929	51.93554
Urban 3	98.99348	96.13606	84.37760	71.64518	71.64518

Table 5.15: Results of Lucas Kanade with Pyramids with window size equal to 15 from Middlebury Dataset (R0.5, R1.0. R2.0, R3.0, R5.0) Real Frames

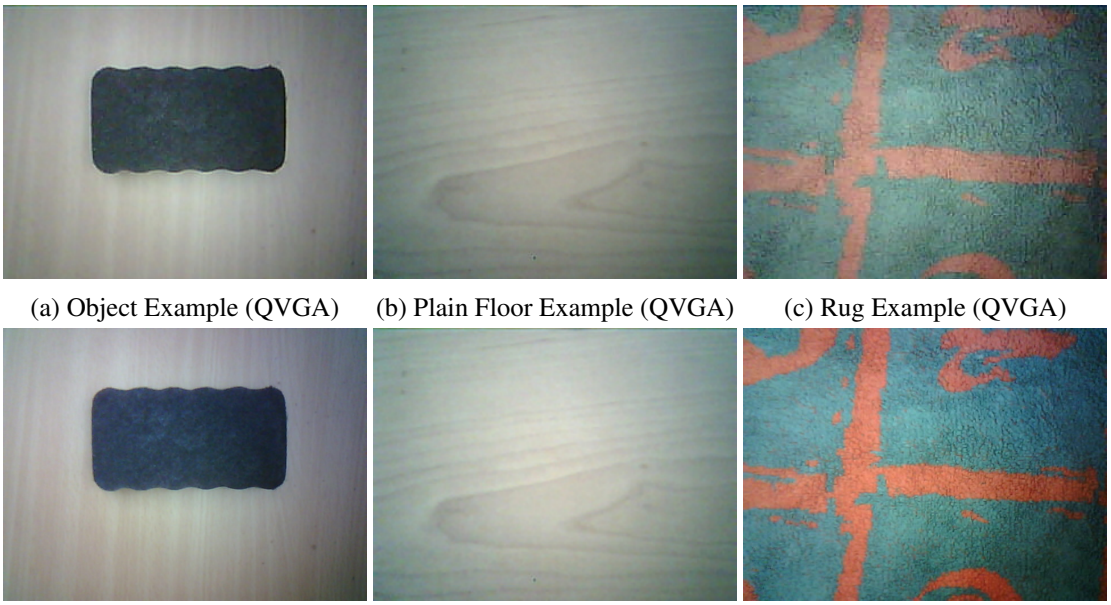
Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	3.08896	0.51526	1.25095	0.04715	3.98584	68.797
Grove 3	3.91333	2.29184	1.22222	0.24122	3.656568	68.891
Urban 2	8.39281	8.07619	1.21260	0.35285	3.70195	68.727
Urban 3	7.30574	4.36940	1.37347	0.12815	4.07583	68.895

Table 5.16: Results of Block Matching with window size equal to 51, stride of 21 and shift of 1 from Middlebury Dataset Real Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	41.13378	0.01041
Grove 3	98.17675	92.62825	80.15234	60.68782	26.63444
Urban 2	94.28808	83.73437	73.40592	64.04622	40.08756
Urban 3	100	100	95.49414	89.01822	56.06250

Table 5.17: Results of Block Matching with window size equal to 51, stride of 21 and shift of 1 from Middlebury Dataset Real Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

the selected metrics. One important aspect to mention is the influence of the algorithm parameters in the results. In Tables A.1, A.2, A.3, A.4, it's possible to see the influence of the change of parameters. This aspect is not studied in this body of work, but it has significant relevancy. The results in the dataset an floor sequence tests are the ones obtained with the best parameters overall.



(a) Object Example (QVGA) (b) Plain Floor Example (QVGA) (c) Rug Example (QVGA)

(d) Object Example (VGA) (e) Plain Floor Example (VGA) (f) Rug Example (VGA)

Figure 5.3: Frames from Plain, Rug and Object Floor Sequence with lux 500 and QVGA resolution (320×240) (5.3a, 5.3b, 5.3c) and VGA (640×480) (5.3d, 5.3e, 5.3f)

In a more holistic view of the work, the study of end-point errors average and their standard deviation (AEE, std EE, respectably) give important information about the possible application on real-time and floor environments. If the results were too high, this could mean that the localisation of the robot would be considerably difficult and not reliable. The AEE (average end-point error) also represented the average error that could be expected in the calculation of the average optical flow vectors for the horizontal and vertical components. The evaluation of outliers metrics are also extremely important, not just because they represent an important aspect of the algorithm's result, but also because these metrics and their quantities influence greatly the calculation of the average optical flow vector. This influence can affect negatively the goal of trying to create a linear, or almost linear, relationship between the displacement of the ESP32-Cam module and the average optical flow vectors of the u and v components.

5.1.2 Floor Sequence

In relation to the Floor Sequence, in Figure 5.3, it's possible to see the photos and respectively quality of each experiment at a height of 21 cm and illuminance of 500 lux.

The result of the floor sequences were separated in margin of error, average memory (Mbit) and average runtime (seconds), in the first table. In terms of floor sequence, the first results from the obstacle (board eraser), the calculation from the average values of the v (vertical) component for a displacement of 1cm, 2cm, and 3cm gave the results from Tables 5.18.

In relation to the displacement and the average value of v, in the Horn and Schunck algorithm in a plain floor with an object scenario (Table 5.18), the displacement of 1 cm as the average value

Algorithm	***Displacement (cm)***	*Margin of error*	*Average Memory (Mbit)*	***Average Runtime (seconds)***
Horn-Schunck	1cm	1.8957 ± 0.066 ($\pm 3.48\%$)	85.000	40.1929
Lucas-Kanade	1cm	-0.7344 ± 0.0979 ($\pm 13.33\%$)	79.523	0.04707
Lucas-Kanade (Pyramids)	1cm	593.2016 ± 23.747 ($\pm 4.00\%$)	83.801	0.06944
Block-Matching	1cm	0.2672 ± 0.0284 ($\pm 10.64\%$)	66.711	16.3941
Horn-Schunck	2cm	3.0281 ± 0.0237 ($\pm 0.78\%$)	85.000	41.1948
Lucas-Kanade	2cm	-0.2335 ± 0.486 ($\pm 208.32\%$)	79.523	0.04707
Lucas-Kanade (Pyramids)	2cm	634.5269 ± 9.142 ($\pm 1.44\%$)	83.801	0.06027
Block-Matching	2cm	0.31652 ± 0.00445 ($\pm 26.95\%$)	66.711	11.9046
Horn-Schunck	3cm	3.0803 ± 0.0309 ($\pm 1.00\%$)	85.000	51.127
Lucas-Kanade	3cm	1.3407 ± 0.0433 ($\pm 3.23\%$)	79.523	0.05518
Lucas-Kanade (Pyramids)	3cm	642.4372 ± 2.222 ($\pm 0.35\%$)	83.801	0.06653
Block-Matching	3cm	0.3711 ± 0.00705 ($\pm 2.60\%$)	66.711	12.1957

Table 5.18: Optical Flow Average Results for component v (Horizontal) with a Confidence Level of $95\%, 1.960 s_{\bar{x}}$ for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a plain floor with an object

of 1.8957, the displacement of 2 cm as the average value of 3.0281, the displacement of 3 cm as the average value of 3.0803. It's immediately evident that the values don't represent a linear relation, even though a higher value is represented as the displacement grows. This can be explained by the fact the linear equation of color constancy is not met, or in other words, for a displacement of 3 cm or more, the Horn and Schunck algorithm can't compute the relationship well (optical flow average vectors are not well computed). Possible improvements could be a way to, for bigger values, using bigger penalisation factor in order to create a smoother flow. In terms of memory used, it's the algorithm that uses the most memory and has the biggest runtime. At 45 seconds on average, it represents a significant challenge to real-time navigation. With different parameters, the results weren't consistent and very small overall.

For the Lucas-Kanade without pyramids, the displacement of 1 cm as the average value of 0.7344, the displacement of 2 cm as the average value of 0.2335, the displacement of 3 cm as the average value of 1.3407. It's important to notice that in the 2 cm displacement, the margin

of error was 208.32%, being a very high value. This can be explained by the homogeneity of object's body (board eraser), as mentioned in the Background Section. This will also be explored in the next Section, but smaller (but not too small) objects could have better results. As the displacement grows, the average optical flow vector decreases from negative values to positive values. A possible explanation for this will be studied in the next Section. In terms of memory used, it's the algorithm that uses 79.523 Mbit of memory and a relatively small runtime, very suitable for real-time visual navigation.

For the Lucas-Kanade without pyramids, the average optical flow v-component were very high when compared with the previous two algorithms (Lucas-Kanade and Horn and Schunck), not representing a purely linear relationship but still growing as the displacement grows. It presents a similar runtime and memory as the Horn and Schunck and Lucas-Kanade without pyramids algorithms.

For the Block-Matching algorithm, the results were considerable better, since as the displacements grows, the average values increases in an average of 0,5 pixels, representing an approximately linear behaviour. The Block-Matching algorithm presents the less usage of memory and runtime of on average of 12 seconds, still slighly high for real-time usage.

For the plain floor, the average optical flow results for component v (horizontal) with a confidence level of $95\%, 1.960 s_{\bar{x}}$ for QVGA resolution for 1cm, 2cm, and 3cm displacement at height of 21 cm are represented in Tables 5.19 in 5.22.

In the case of a plain floor for the Horn and Schunck algorithm (Table 5.19), the displacement of 1 cm as the average value of 0.1206, the displacement of 2 cm as the average value of 0.1604, the displacement of 3 cm as the average value of 0.1665. As the previous results (object in plain floor), there exists a relationship between the increase of displacement and average optical flow result, even though it's not exactly linear.

For the Lucas-Kanade without pyramids, it produced results without significance, since it starts with a positive value for a 1cm displacement and then gets negative as the displacement grows.

For the Lucas-Kanade with pyramids, the results of the average optical flow v component were not high in terms of the object sequence (around 0,1 pixel, compared to the 600 pixels of the object on the plain floor sequence), and don't represent a purely linear relationship, but still growing as the displacement grows.

For the Block-Matching algorithm, the results were considerably better, since as the displacements grows, the average values increases on average 0,03 pixels, representing an approximately linear behaviour. The Block-Matching algorithm presents a runtime of, on average 15 seconds, slightly higher than the object on the plain floor scenario.

Compared to the previous case, it's noticeable that the margins of error are bigger. This can mean that a single object has more characteristics that will enable a more accurate calculation of computation of optical flow.

Finally, for the rug floor, the optical flow average results for the component v (Horizontal) with a Confidence Level of $95\%, 1.960 s_{\bar{x}}$ for QVGA Resolution for 1cm displacement at height of 21 cm, are represented at the Table 5.20, 5.23.

Algorithm	***Displacement (cm)***	*Margin of error*	*Average Memory (Mbit)*	***Average Runtime (seconds)***
Horn-Schunck	1cm	0.1206 ± 0.01078 ($\pm 8.94\%$)	85.004	40.3742
Lucas-Kanade	1cm	0.7588 ± 0.0307 ($\pm 4.05\%$)	79.523	0.05047
Lucas-Kanade (Pyramids)	1cm	0.0858 ± 0.00461 ($\pm 5.38\%$)	83.801	0.08192
Block-Matching	1cm	0.01568 ± 0.00134 ($\pm 8.56\%$)	66.711	18.305
Horn-Schunck	2cm	0.1604 ± 0.01142 ($\pm 7.12\%$)	85.004	18.7178
Lucas-Kanade	2cm	-0.2242 ± 0.3430 ($\pm 153.14\%$)	79.523	0.05544
Lucas-Kanade (Pyramids)	2cm	0.2601 ± 0.00642 ($\pm 2.47\%$)	83.801	0.09652
Block-Matching	2cm	0.01652 ± 0.00445 ($\pm 26.95\%$)	66.711	11.9046
Horn-Schunck	3cm	0.1665 ± 0.00102 ($\pm 6.13\%$)	85.004	22.4623
Lucas-Kanade	3cm	-0.1824 ± 0.2141 ($\pm 117.39\%$)	79.523	0.05718
Lucas-Kanade (Pyramids)	3cm	0.4692 ± 0.0063 ($\pm 1.36\%$)	83.856	0.07111
Block-Matching	3cm	0.018251 ± 0.00587 ($\pm 32.16\%$)	66.716	14.8726

Table 5.19: Optical Flow Average Results for component v (Horizontal) with a Confidence Level of $95\%, 1.960 s_{\bar{x}}$ for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a plain floor

In the case of a rugged floor for the Horn and Schunck algorithm (Table 5.19), the average of optical flow vector is inversely related with the displacement. Even though it's not a normal pattern expected, the inverse relationship can mean that the majority of the values can be closer to an expected value, but some outliers or zones of outliers can alter the average value. This pattern is also verified on the Block-Matching algorithm. In the other results, like runtime and memory, they are very similar to the ones studied before.

For the Lucas-Kanade without pyramids, it produced again, results without significance, since it starts with a positive value for a 1cm displacement and then gets negative as the displacement grows.

For the Lucas-Kanade with pyramids, the results were not high in terms of the object sequence (around 0,2 pixel, compared to 600 pixels of the object on the plain floor sequence), and don't represent a purely linear relationship, but still growing as the displacement grows, like the plain floor sequence.

Algorithm	***Displacement (cm)***	*Margin of error*	*Average Memory (Mbit)*	***Average Runtime (seconds)***
Horn-Schunck	1cm	0.1606 ± 0.0144 ($\pm 8.94\%$)	85.000	10.4787
Lucas-Kanade	1cm	0.5588 ± 0.0227 ($\pm 4.05\%$)	79.523	0.05047
Lucas-Kanade (Pyramids)	1cm	0.2858 ± 0.1296 ($\pm 45.36\%$)	83.801	0.06192
Block-Matching	1cm	0.04568 ± 0.0065 ($\pm 14.23\%$)	66.711	18.305
Horn-Schunck	2cm	0.1104 ± 0.00565 ($\pm 5.12\%$)	85.000	15.7178
Lucas-Kanade	2cm	-0.0742 ± 0.114 ($\pm 153.14\%$)	79.523	0.05544
Lucas-Kanade (Pyramids)	2cm	0.4101 ± 0.109 ($\pm 26.47\%$)	83.801	0.07652
Block-Matching	2cm	0.01652 ± 0.00445 ($\pm 26.95\%$)	66.711	11.9046
Horn-Schunck	3cm	0.08245 ± 0.00669 ($\pm 8.11\%$)	85.000	19.4323
Lucas-Kanade	3cm	-0.0824 ± 0.0968 ($\pm 117.39\%$)	79.523	0.05518
Lucas-Kanade (Pyramids)	3cm	0.5692 ± 0.0572 ($\pm 10.06\%$)	83.801	0.07111
Block-Matching	3cm	0.004251 ± 0.00349 ($\pm 82.16\%$)	66.711	14.8726

Table 5.20: Optical Flow Average Results for component v (Horizontal) with a Confidence Level of $95\%, 1.960 s_{\bar{x}}$ for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a Rugged Floor

For the Block-Matching algorithm, the inverse relationship between displacement and average value was already analysed. The use of this algorithm maybe not be ideal without further investigation.

Compared to the previous case, it's noticeable that some relationships between displacement and average optical flow results were not expected and possibly can be expected of frames that have texture and more defined patterns, and so, are more difficult to analyse. The runtime and memory were similar to the previous experiments, as expected.

Overall, the algorithms produced good results, expected from the Lucas-Kanade without Pyramids in all sequences and the Horn-Schunck and Block-Matching in the last sequence (rug).

In relation to the Lucas-Kanade without Pyramids, it's possible to conclude that it doesn't work for displacements bigger than 1 pixel. This results are consistent with the existing literature and noticed on the Background part. So, this algorithm can be excluded from the implementation in real-time.

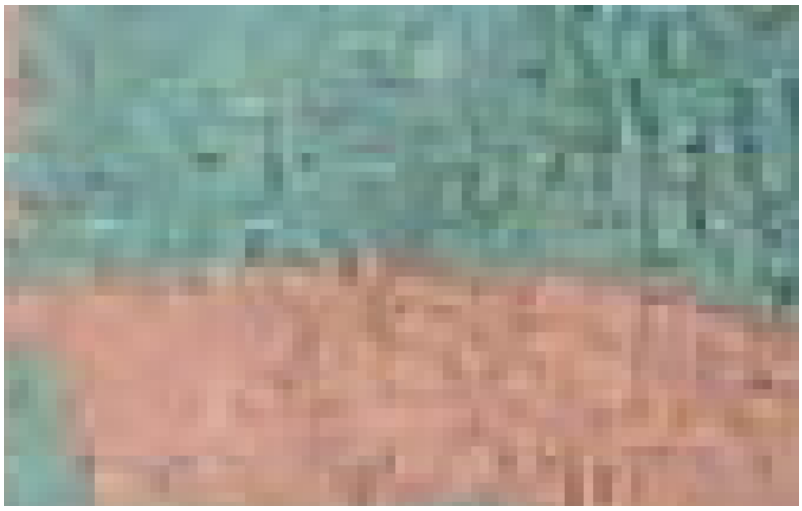


Figure 5.4: Visual artefacts from rug floor sequence image dataset

In relation to the Horn-Schunck and Lucas-Kanade with Pyramids, one of the possible explanations consist on the poor image quality and the introduction of artefacts in the images due to compression. Visual artefacts (also artefacts) are anomalies apparent during visual representation, as in digital graphics and other forms of imagery, especially photography and microscopy.

This can alter the values and produce wrong average optical flow values. This sequence is the only one that has considerable texture and the ESP32-Cam module seems to not produce great results. In Figure 5.4, it's possible to see visual artefacts from the one of the images in the rug floor sequence dataset.

This can be caused due to the lack of flash memory in the ESP32-Cam module, since the option for JPEG compression is disabled in the library used.

The analysis of these three scenarios is very important, even though they represent a very small sample of possible visual navigation scenarios. With this information, we can extrapolate to more challenging scenarios where the illuminance of the scene is lower or changing, where exists the presence of boundaries and/or occlusions, and different objects heights and appearances. The resolution of the images can change, as well as the height of the robot from 21cm to the ground. In summary, these sample represent an almost perfect scenario, where:

1. The illuminance is very high (at 500 lux) and there are no shadows or reflexes from the floor
2. The ESP32-Cam (or robot) only moves in one direction
3. The distance to the ground is controlled and not aleatory

These conditions enable a study where a possible visual odometry system could be implemented. The real-application of this kind of system is very subjective to the types of conditions in the path of the robot. If the path is indoors, it's possible to hypothesise that the optical flow algorithms can be optimised for a specific environment in terms of parameters.

Another very important aspect is that of the algorithms. The algorithms studied are traditional and represent, in some aspects, the begging of the study of variational methods, The Horn-Schucnk [38] and Lucas-Kanade [49] algorithms are 40 years old and don't have the same quality results of more recent approaches (Figure 4.11). This can very be limiting, not just in terms of the quality of results, but in terms of runtime. So, a better approach would be to use a modern optical flow algorithm or an improvement of one of the optical flow algorithms.

Outliers can increase the average value of optical flow vectors, being optical flow algorithms that produce more and bigger quantities of outliers ill represented. One possibility is to implement a filter to exclude the higher values, even though for a single object this exclusion can remove important information about the scene.

Finally, it's important to mention that when trying to introduce no compression in the images, sometimes the images were distorted, what would produced bad results.

So, in general, some limitations of the visual odometry system can also be derived, like:

1. VO requires moderate-good lighting conditions. This was not represented in this work, since all the results were very poor due to the noise added.

2. VO fails when features are limited. This is evident when the path analysed doesn't have some easy to recognise patterns, or it's almost impossible to see a change in pattern when moving from one place to another.

3. The Lucas-Kanade with Pyramids was the algorithm that presented better results under the studied conditions. It was competitive in relation to runtime and the usage of memory was relatively small.

One important aspect is that, even though it produced good results, the Lucas-Kanade with Pyramids results still could be improved if the parameters were optimised and more scenarios, like lower illuminance, occlusions, and boundaries were tested.

5.2 Results of Robot Localisation with Real-Time Optical Flow Calculation

For the real-time optical flow calculation, the first important aspect to study are the average frequency in Hertz and the average velocity in $cm \times s^{-}1$ in which the algorithms studied produced the results. This can be seen in Tables 5.21, 5.22, 5.23.

In terms of frequency and velocity of optical flow algorithms, the Lucas-Kanade algorithms are the ones competitive enough to be implemented in a real-time system, at least without any kind of parameter optimisation. These algorithms can reach a maximum of approximately 54Hz. Even though for an average velocity of an AVG ($400cm \times s^{-}1$ $600cm \times s^{-}1$) can not be enough without compromising the robot's velocity.

Algorithm	***Displacement (cm)***	*Average Frequency (Hertz)*	*Average Velocity* $(cm \times s^{-}1)$
Horn-Schunck	1cm	0.02488	0.02488
Lucas-Kanade	1cm	21.24495	21.24495
Lucas-Kanade (Pyramids)	1cm	14.40092	14.40092
Block-Matching	1cm	0.06099	0.06099
Horn-Schunck	2cm	0.02427	0.04854
Lucas-Kanade	2cm	20.60156	41.20313
Lucas-Kanade (Pyramids)	2cm	16.59200	33.18400
Block-Matching	2cm	0.08400	0.16800
Horn-Schunck	3cm	0.01955	0.03911
Lucas-Kanade	3cm	18.122508	54.367524
Lucas-Kanade (Pyramids)	3cm	15.03081	45.09243
Block-Matching	3cm	0.06653	0.19959

Table 5.21: Optical Flow Average Results for Frequency and Velocity for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a plain floor with an object

Algorithm	***Displacement (cm)***	*Average Frequency (Hertz)*	***Average Velocity*** $(cm \times s^{-}1))$
Horn-Schunck	1cm	0.02476	0.02476
Lucas-Kanade	1cm	19.8137	1981376
Lucas-Kanade (Pyramids)	1cm	12.2070	12.2070
Block-Matching	1cm	66.711	18.305
Horn-Schunck	2cm	0.0546	0.1092
Lucas-Kanade	2cm	18.0375	36.0750
Lucas-Kanade (Pyramids)	2cm	10.3605	20.7210
Block-Matching	2cm	0.0840	0.168
Horn-Schunck	3cm	0.0445	0.1335
Lucas-Kanade	3cm	17.4886	52.4658
Lucas-Kanade (Pyramids)	3cm	14.02627	42.1881
Block-Matching	3cm	0.06723	0.2017

Table 5.22: Optical Flow Average Results for Frequency and Velocity for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a plain floor

For example, the PX4FLOW Smart Camera operates at 400Hz and has a bigger resolution of 752×480, producing probably better results than this implementation. Although, this work and study can still be implemented in some cases with relative success.

Algorithm	***Displacement (cm)***	*Average Frequency (Hertz)*	*Average Velocity* $(cm \times s^{-}1)$
Horn-Schunck	1cm	0.09543	0.09543
Lucas-Kanade	1cm	19.81375	19.81375
Lucas-Kanade (Pyramids)	1cm	16.14987	16.14987
Block-Matching	1cm	0.05462	0.05462
Horn-Schunck	2cm	0.063622	0.127244
Lucas-Kanade	2cm	18.03751	36.07503
Lucas-Kanade (Pyramids)	2cm	13.06847	26.13695
Block-Matching	2cm	0.084001	0.168002
Horn-Schunck	3cm	0.05146	0.15438
Lucas-Kanade	3cm	18.12250	54.367524
Lucas-Kanade (Pyramids)	3cm	14.06271	42.18815
Block-Matching	3cm	0.06723	0.20171

Table 5.23: Optical Flow Average Results for Frequency and Velocity for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a rugged floor

For the real-time analysis is important to do a linear regression in order to compute values intermediary between 1cm and 3cm, and possibly results under and above this range. In Table 5.24 is possible to see the results.

Algorithm	***Object Sequence*** $(pixel \times cm^{-}1)$	*Plain Sequence* $(pixel \times cm^{-}1)$	***Rug Sequence*** $(pixel \times cm^{-}1)$
Horn-Schunck	$(0.5923 \pm 0.3118)x+$ (1.483 $\pm$0.6736)	$(0.02295 \pm 0.009728)x+$ (0.1033 $\pm$0.02102)	$(-0.03908 \pm 0.006423)x+$ (0.1960 $\pm$0.01388)
Lucas-Kanade	$(1.038 \pm 0.3098)x+$ (-1.951 $\pm$0.6693)	$(-0.4706 \pm 0.2958)x+$ (1.059 $\pm$0.6391)	$(-0.3206 \pm 0.1804)x+$ (0.7753 $\pm$0.3896)
Lucas-Kanade (Pyramids)	$(51.62 \pm 25.23)x+$ (502.2 $\pm$54.51)	$(0.1917 \pm 0.01005)x+$ (-0.1117 $\pm$0.02170)	$(0.1417 \pm 0.01005)x+$ (0.1383 $\pm$0.02170)
Block-Matching	$(0.05195 \pm 0.001518)x+$ (0.2144 $\pm$0.003280)	$(0.001286 \pm 0.0002572)x+$ (0.01425 $\pm$0.0005556)	$(-0.02071 \pm 0.004876)x+$ (0.06358 $\pm$0.01053)

Table 5.24: Linear regression for Optical Flow values of Horn-Schuck, Lucas-Kanade, Lucas-Kanade (Pyramids), Block-Matching for QVGA Resolution and a height of 21 cm

The linear regression can be a not optimal choice. This is do to the non-linearity of the relationship between displacement and average optical flow vector component, like for example the limitation about large displacements. This can cap the maximum valid results, like for example 3 cm, and create nonsensical or plain wrong results. Another non-linear behaviour can be derived from the algorithm result itself, where another kind of function can describe the relationship better.

Algorithm	***Average Displacement Object Sequence (cm)***	***Average Displacement Plain Floor Sequence (cm)***	***Average Displacement Rugged Floor Sequence (cm)***
Horn-Schunck	1.15 ± 0.08	1.23 ± 0.12	1.18 ± 0.11
Lucas-Kanade	1.22 ± 0.23	1.55 ± 0.3	0.78 ± 0.35
Lucas-Kanade (Pyramids)	1.15 ± 0.06	1.12 ± 0.12	1.14 ± 0.1
Block-Matching	0.95 ± 0.12	0.98 ± 0.15	1.098 ± 0.11

Table 5.25: Displacement Results for Optical Flow values of Horn-Schuck, Lucas-Kanade, Lucas-Kanade (Pyramids), Block-Matching for QVGA Resolution for 1cm Displacement at a height of 21 cm

The tests under 1cm, like 0.5cm, 0.2cm, 0.1cm, etc. and above 3cm are necessary since there is too little data points and the non-linear behaviour wasn't tested in detail, being potentially some important information lost. Still in relationship to the tests under 1cm, like 0.5cm, 0.2cm, 0.1cm, these tests are also important to verify if the algorithms produce more consistent results for small displacements, even though if the runtime remained similar, this would exclude immediately their application, since the average velocity would be very slow and not applicable to real autonomous robots.

Lastly, it's important to note that the linear regression was made with the average points of the optical flow results, being lost a lot of uncertainty (margin of errors) and another method with more data points should be studied.

With this information, some tests were performed in the same types of floor (object in plain floor, plain floor, rugged floor), in order to test these functions. The results were made only for 1cm and are presented in the Table 5.25.

The results are considerably good, even though for some the margin of error is to big and in long distance can accumulate error. As noted previously, the best algorithm for real-time optical flow calculation is the Lucas-Kanade with Pyramids.

Choosing the best algorithm, still the results vary a lot, even for this controlled experiment. So, a sensor fusion (kalman filter) is a necessary step for the correct localisation of the robot. The errors in the compression and difficult scenes, could also produce bad results without significance, giving even more importance to a sensor fusion approach with an Inertial measurement unit (IMU), for example. This aspect is not a subject of this study.

Chapter 6

Conclusions

This Chapter is divided in Achievements and Further Investigation. In this last Section, some of the limitations, as well as suggestions, are studied in order to propose a deeper and more comprehensive guide to implement a visual odometry system in the future, based on the ESP32-Cam module. The Achievements is an analysis of the overall results of the experiments performed and their respectably success. All the parts explored previously in the thesis (Datasets, Floor Sequences, etc) of the Results are analysed in this Chapter.

6.1 Achievements

The first important achievement is the elaboration of four working traditional optical flow algorithms. These algorithms were tested under a public optical flow dataset (Middlebury) and floor sequences images created by the ESP32-Cam module, and produced, overall, consistent results. The relationship between displacement and average optical flow vectors had several challenges, being the fact that the Lucas-Kanade was not reliable and in some tests, the Horn and Schunck and Block-Matching algorithms had a reversed relationship between displacement and average optical flow value. The margin of error was considerable for the linear regression, but in the practical application was possible to get good results, specially for the Lucas-Kanade with Pyramids algorithm.

The real-time system had relatively good results for the computation of the displacements, even though the frequency was limited by the real-time system chosen, that was composed of the ESP-Cam, a local web server and the ESP32-Cam. The internet connection was a very important aspect of this system. Being the local web server, the way to send the images to the computer, is crucial and could be interrupted if ESP32-Cam didn't have connection to the internet. In this aspect, the embedded real-time system of optical flow has a more robust functioning.

6.2 Further Investigation

Future work can consist on trying to define the covariance of the result of the displacement computed by the average optical flow value. This can be achieved by performing an FFT of the image or relating the result in some way to the characteristics of the image sequence. The covariance is fundamental, since this is necessary data for a sensor fusion algorithm.

Another important aspect is the investigation of a more robust and modern optical flow algorithm. As shown in the dataset and floor sequence dataset, the four algorithms don't present State-of-Art results in terms of quality and runtime metrics. This can lead to problems related with the localisation system of the mobile robot. A State-of-Art algorithm can be achieved by the use of CNN methods or not, but the creation of a dataset for this kind of application in the optical flow field of study could be very interesting for further improvements.

Another important aspect to mention is the fact that the localisation system used in this work may not be the best and other kinds of implementation should be studied. The ESP32-Cam embedded system is one that should be explored in future work and investigation.

Appendix A

Optical Flow

A.1 Optical Flow Algorithms Tables

Algorithm 1 Horn and Schunck Algorithm

Require: $img1$ $img2$ ▷ Matrices with image values
Ensure: $img1 \leftarrow$ valid and smoothed (gaussian), $img2 \leftarrow$ valid and smoothed (gaussian)

```
u ← 0
v ← 0
while True do
    u_avg = convolution(u, avg_kernel)
    v_avg = convolution(v, avg_kernel)
    p = f_x u_avg + f_y v_avg + f_t
    d = 4α^2 + f_x^2 + f_y^2
    prev = u
    u = u_avg − fx(p/d)
    v = v_avg − fy(p/d)
    diff = norm(u − prev)
    if diff < delta or iter counter > 300 then ▷ The number of iteration is defined case by case
        break
    end if
end while
```

A.2 Optical Flow Dataset Tables

Algorithm 2 Block Macting Algorithm (SAD)

Require: *im*1 *im*2, *window_size*, stride, shift
Ensure: *img*1 ← valid and float values, *img*2 ← valid and float values
  $u \leftarrow 0$, $v \leftarrow 0$
  size1 = im2.shape[0]; size2 = im2.shape[1]; wh = *window_size* / 2
  round1 = im2.shape[0] - wh - 1; round2 = im2.shape[1] - wh - 1
  tx = 0; ty = 0
  for x in range(wh, round1, stride) **do**
    for y in range(wh, round2, stride) **do**
      nm = im2[x-wh:x+wh+1, y-wh:y+wh+1].flatten() ▷ Convert a nxn matrix into 1d array
      min_{dist} = None, $u = 0$, $v = 0$
      for i in range(max(x - shift, wh), min(x + shift + 1, im1.shape[0] - wh - 1)) **do**
        for j in range(max(y - shift, wh), min(y + shift + 1, im1.shape[1] - wh - 1)) **do**
          om = im1[i-wh:i+wh+1, j-wh:j+wh+1].flatten()
          dist = ssd(nm, om)
          if not min_{dist} or dist < min_{dist} **then**
            min_{dist} = dist; $flow_x$ = x - i; $flow_y$ = y - j;
          end if
        end for
        u[-tx,ty] = $flow_y$; v[-tx,ty] = $flow_x$; ty += 1
      end for
      tx += 1; ty = 0
    end for
  end for
  return u, v

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	3.32775	0.73056	1.43750	0.25758	5.23827	109.875
Grove 3	4.21916	2.37944	1.451660	0.39593	6.37485	109.887
Urban 2	8.45857	8.08681	1.26878	0.38732	4.95693	109.637
Urban 3	7.39493	4.34389	1.44945	0.21299	6.10708	109.539

Table A.1: Results of Horn and Schunck with $\alpha = 30$ and 20 iterations Middlebury Dataset Synthetic Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	59.48535	4.19726
Grove 3	98.59016	94.04459	83.39485	68.56022	30.00976
Urban 2	94.79557	84.23535	73.50651	64.93001	40.70182
Urban 3	100	99.99251	96.04069	91.10188	57.27669

Table A.2: Results of Horn and Schunck with $\alpha = 30$ and 20 iterations MiddleBury Dataset Synthetic Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

Group of images	*AEE (pixels)*	*std EE (pixels)*	*AAE (degrees)*	*std AE (degrees)*	*Process Time (seconds)*	*Memory (MiB)*
Grove 2	3.16623	0.57132	1.32194	0.13781	4.78053	109.453
Grove 3	3.94084	2.29362	1.24937	0.24877	5.31955	109.410
Urban 2	8.41162	8.07849	1.23067	0.35993	5.33271	109.395
Urban 3	7.33218	4.36107	1.39883	0.14011	5.46214	109.566

Table A.3: Results of Horn and Schunck with $\alpha = 60$ and 20 iterations MiddleBury Dataset Synthetic Frames

Group of images	*R0.5 (percentage)*	*R1.0 (percentage)*	*R2.0 (percentage)*	*R3.0 (percentage)*	*R5.0 (percentage)*
Grove 2	100	100	100	47.47102	1.25130
Grove 3	98.22493	92.79915	80.51399	61.39453	26.79622
Urban 2	94.45670	83.85546	73.41601	64.34114	40.23046
Urban 3	100	100	95.59570	90.23372	56.45084

Table A.4: Results of Horn and Schunck with $\alpha = 60$ and 20 iterations MiddleBury Dataset Synthetic Frames (R0.5, R1.0. R2.0, R3.0, R5.0)

Appendix B

Results

B.1 Floor Sequence Results

Algorithm	***Displacement (cm)***	*Margin of error*	*Average Memory (Mbit)*	***Average Runtime (seconds)***
Horn-Schunck	1cm	0.06062 ± 0.0532 ($\pm 87.74\%$)	85.000	35.5961
Lucas-Kanade	1cm	-0.102 ± 0.0744 ($\pm 72.96\%$)	79.523	0.05047
Lucas-Kanade (Pyramids)	1cm	593.2016 ± 23.747 ($\pm 4.00\%$)	83.801	0.06944
Block-Matching	1cm	0.0869 ± 0.0302 ($\pm 34.71\%$)	66.711	12.644
Horn-Schunck	2cm	0.0856 ± 0.0766 ($\pm 89.52\%$)	85.000	46.7983
Lucas-Kanade	2cm	-0.0742 ± 0.114 ($\pm 153.14\%$)	79.523	0.04548
Lucas-Kanade (Pyramids)	2cm	139.2906 ± 62.629 ($\pm 44.96\%$)	83.801	0.07419
Block-Matching	2cm	0.05522 ± 0.0483 ($\pm 87.51\%$)	66.711	12.4993
Horn-Schunck	3cm	0.1214 ± 0.0635 ($\pm 52.32\%$)	85.000	47.6144
Lucas-Kanade	3cm	-0.0824 ± 0.0968 ($\pm 117.39\%$)	79.523	0.05518
Lucas-Kanade (Pyramids)	3cm	163.3861 ± 67.202 ($\pm 41.13\%$)	83.801	0.06653
Block-Matching	3cm	0.08874 ± 0.064 ($\pm 72.14\%$)	66.711	12.6632

Table B.1: Optical Flow Average Results for component v (Horizontal) with a Confidence Level of $95\%, 1.960 s_{\bar{x}}$ for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a plain floor (different orientation)

Algorithm	***Displacement (cm)***	*Average Frequency (Hertz)*	*Average Velocity* $(cm \times s^{-}1)$
Horn-Schunck	1cm	0.02809	0.02809
Lucas-Kanade	1cm	19.81375	19.81375
Lucas-Kanade (Pyramids)	1cm	14.40092	14.40092
Block-Matching	1cm	0.07908	0.07908
Horn-Schunck	2cm	0.02136	0.04273
Lucas-Kanade	2cm	21.98768	43.97537
Lucas-Kanade (Pyramids)	2cm	13.47890	26.95781
Block-Matching	2cm	0.0800	0.1600
Horn-Schunck	3cm	0.02100	0.06300
Lucas-Kanade	3cm	18.12250	54.367524
Lucas-Kanade (Pyramids)	3cm	15.03081	45.09243
Block-Matching	3cm	0.07896	0.23690

Table B.2: Optical Flow Average Results for Frequency and Velocity for QVGA Resolution for 1cm, 2cm, and 3cm Displacement at a height of 21 cm on a plain floor (different orientation)

References

[1] Fabrizio Abrate, Basilio Bona, and Marina Indri. Experimental ekf-based slam for mini-rovers with ir sensors only. 01 2007.

[2] Aria Ahmadi and Ioannis Patras. Unsupervised convolutional neural networks for motion estimation. pages 1629–1633, 09 2016.

[3] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, 1989.

[4] Christian Bailer, Bertram Taetz, and Didier Stricker. Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4015–4023, 2015.

[5] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[6] Xicheng Ban, Hongjian Wang, Tao Chen, Ying Wang, and Yao Xiao. Monocular visual odometry based on depth and optical flow using deep learning. *IEEE Transactions on Instrumentation and Measurement*, 70:1–19, 2021.

[7] Connelly Barnes, Eli Shechtman, Dan B. Goldman, and Adam Finkelstein. The generalized patchmatch correspondence algorithm. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 29–43, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] John Barron, David Fleet, and S. Beauchemin. Performance of optical flow techniques. volume 12, pages 43–77, 02 1994.

[9] Bastian Bischoff, Duy Nguyen-Tuong, Felix Streichert, Marlon Ewert, and Alois Knoll. Fusing vision and odometry for accurate indoor robot localization. pages 347–352, 12 2012.

[10] J. Borenstein and Y. Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, 1991.

[11] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 25–36, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[12] Thomas Brox and Jitendra Malik. Large displacement optical flow: Descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):500–513, 2011.

[13] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 611–625, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[14] J. Campbell, R. Sukthankar, I. Nourbakhsh, and A. Pahwa. A robust visual odometry and precipice detection system using consumer-grade monocular vision. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3421–3427, 2005.

[15] Fernando Carreira, João Calado, Carlos Cardeira, and Paulo Oliveira. Navigation system for mobile robots using pca-based localization from ceiling depth images: Experimental validation. In *2018 13th APCA International Conference on Automatic Control and Soft Computing (CONTROLO)*, pages 159–164, 2018.

[16] R. Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 138–145, 1985.

[17] D.M. Cole and P.M. Newman. Using laser range data for 3d slam in outdoor environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1556–1563, 2006.

[18] J.A Cooney, W.L Xu, and G Bright. Visual dead-reckoning for motion control of a mecanum-wheeled mobile robot. *Mechatronics*, 14(6):623–637, 2004.

[19] Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1403–1410 vol.2, 2003.

[20] Francisco Dias, Hanna Schafer, Leonardo Natal, and Carlos Cardeira. Mobile robot localisation for indoor environments based on ceiling pattern recognition. In *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, pages 65–70, 2015.

[21] Christian Dornhege and Alexander Kleiner. Visual odometry for tracked vehicles. 01 2006.

[22] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.

[23] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865, 1991.

[24] Rye D. Nebot Durrant-Whyte, H. *Localization of autonomous guided vehicles*. roceedings of the IEEE International Conference on Robotics and Automation, vol. 7, 1996.

[25] Benjamin Dutton and Elbert Maloney. *Dutton's Navigation piloting*. Naval Institute Press, 1978.

[26] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 834–849, Cham, 2014. Springer International Publishing.

[27] Nikolas Engelhard, Felix Endres, Jürgen Hess, Jürgen Sturm, and Wolfram Burgard. Real-time 3 d visual slam with a hand-held rgb-d camera. 2011.

[28] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. 04 2015.

[29] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In Tomáš Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 224–237, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[30] David Gadot and Lior Wolf. Patchbatch: A batch augmented loss for optical flow. pages 4236–4245, 2016.

[31] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[32] Joel Gibson and Oge Marques. Optical flow and trajectory estimation methods. In *SpringerBriefs in Computer Science*, 2016.

[33] Fatma Güney and Andreas Geiger. Deep discrete flow. In *ACCV*, 2016.

[34] D.M. Helmick, Yang Cheng, D.S. Clouse, L.H. Matthies, and S.I. Roumeliotis. Path following using visual odometry for a mars rover in high-slip environments. In *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*, volume 2, pages 772–789 Vol.2, 2004.

[35] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[36] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. *RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments*, pages 477–491. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[37] Steven Hordley and Graham Finlayson. Reevaluation of color constancy algorithm performance. *Journal of the Optical Society of America. A, Optics, image science, and vision*, 23:1008–20, 06 2006.

[38] Berthold Horn and Brian Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 08 1981.

[39] Yinlin Hu, Rui Song, and Yunsong Li. Efficient coarse-to-fine patch match for large displacement optical flow. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5704–5712, 2016.

[40] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8981–8989, 2018.

[41] Junhwa Hur and Stefan Roth. Mirrorflow: Exploiting symmetries in joint optical flow and occlusion estimation. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 312–321, 2017.

[42] Junhwa Hur and Stefan Roth. Iterative residual refinement for joint optical flow and occlusion estimation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5747–5756, 2019.

[43] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1647–1655, 2017.

[44] Michal Irani and Shmuel Peleg. Motion analysis for image enhancement: Resolution, occlusion, and transparency. *Journal of Visual Communication and Image Representation*, 4(4):324–335, 1993.

[45] Sunhyo Kim and Se-Young Oh. Slam in indoor environments using omni-directional vertical and horizontal line features. *Journal of Intelligent and Robotic Systems*, 51:31–43, 01 2008.

[46] L. Kleeman. Advanced sonar and odometry error modeling for simultaneous localisation and map building. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 699–704 vol.1, 2003.

[47] Min Liu and Tobi Delbruck. Block-matching optical flow for dynamic vision sensors: Algorithm and fpga implementation. In *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2017.

[48] Pengpeng Liu, Michael R. Lyu, Irwin King, and Jia Xu. Selflow: Self-supervised learning of optical flow. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4566–4575, 2019.

[49] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision (ijcai). volume 81, 04 1981.

[50] Ian Mahon, Stefan B. Williams, Oscar Pizarro, and Matthew Johnson-Roberson. Efficient view-based slam using visual loop closures. *IEEE Transactions on Robotics*, 24(5):1002–1014, 2008.

[51] L. Matthies and S. Shafer. Error modeling in stereo navigation. *IEEE Journal on Robotics and Automation*, 3(3):239–248, 1987.

[52] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. pages 4040–4048, 06 2016.

[53] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3061–3070, 2015.

[54] Moritz Menze, Christian Heipke, and Andreas Geiger. Discrete optimization for optical flow. In Juergen Gall, Peter Gehler, and Bastian Leibe, editors, *Pattern Recognition*, pages 16–28, Cham, 2015. Springer International Publishing.

[55] Thomas Moore and Daniel W. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *IAS*, 2014.

[56] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[57] Alan Mutka, Damjan Miklic, Ivica Draganjac, and Stjepan Bogdan. A low cost vision based localization system using fiducial markers. *IFAC Proceedings Volumes*, 41(2):9528–9533, 2008. 17th IFAC World Congress.

[58] Etienne Mémin and Patrick Pérez. Hierarchical estimation and segmentation of dense motion fields. *International Journal of Computer Vision*, 46(2):129–155, 2002.

[59] Mohamed Nadour, Mohamed Boumehraz, Lakhmissi Cherroun, and Vicenç Puig. Mobile robot visual navigation based on fuzzy logic and optical flow approaches. *Int. J. Syst. Assur. Eng. Manag.*, 10:1654–1667, 2019.

[60] Michal Neoral, Jan Sochman, and Jiri Matas. Continual occlusions and optical flow estimation. *ArXiv*, abs/1811.01602, 2018.

[61] Anurag Ranjan and Michael J. Black. Optical flow estimation using a spatial pyramid network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2720–2729, 2017.

[62] G. Reina, L. Ojeda, A. Milella, and J. Borenstein. Wheel slippage and sinkage detection for planetary rovers. *IEEE/ASME Transactions on Mechatronics*, 11(2):185–195, 2006.

[63] Zhile Ren, Orazio Gallo, Deqing Sun, Ming-Hsuan Yang, Erik B. Sudderth, and Jan Kautz. A fusion approach for multi-frame optical flow estimation. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2077–2086, 2019.

[64] Jörg Rett and Jorge Manuel Miranda Dias. Autonomous robot navigation-a study using optical flow and log-polar image representation. 2005.

[65] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. 01 2015.

[66] R. Sathya and Annamma Abraham. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2:34–38, 2013.

[67] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics Automation Magazine*, 18(4):80–92, 2011.

[68] T. Schuster, L. Wolf, and D. Gadot. Optical flow requires multiple strategies (but only one network). In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6921–6930, Los Alamitos, CA, USA, jul 2017. IEEE Computer Society.

[69] Aan Eko Setiawan, Angga Rusdinar, Rina Mardiati, and Eki Ahmad Dzaki Hamidi. Ann design model to recognize the direction of multi-robot agv. *2021 7th International Conference on Wireless and Telematics (ICWT)*, pages 1–4, 2021.

[70] Syed Shah and Xiang Xuezhi. Traditional and modern strategies for optical flow: an investigation. volume 3, 03 2021.

[71] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[72] Randall Smith, Matthew Self, and Peter Cheeseman. *Estimating Uncertain Spatial Relationships in Robotics*, pages 167–193. Springer New York, New York, NY, 1990.

[73] Kahlouche Souhila and Achour Karim. Optical Flow Based Robot Obstacle Avoidance. volume 4, page 2, 2007.

[74] Frank Steinbrücker, Thomas Pock, and Daniel Cremers. Large displacement optical flow computation withoutwarping. *2009 IEEE 12th International Conference on Computer Vision*, pages 1609–1614, 2009.

[75] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[76] A. Talukder, S. Goldberg, L. Matthies, and A. Ansar. Real-time detection of moving objects in a dynamic scene from moving robotic vehicles. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 2, pages 1308–1313 vol.2, 2003.

[77] Sebastian Thrun. Probabilistic robotics. *Commun. ACM*, 45(3):52–57, mar 2002.

[78] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[79] Yang Wang, Yi Yang, Zhenheng Yang, Liang Zhao, Peng Wang, and Wei Xu. Occlusion aware unsupervised learning of optical flow. pages 4884–4893, 06 2018.

[80] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *2013 IEEE International Conference on Computer Vision*, pages 1385–1392, 2013.

[81] Neha D. Yadav and Shridhar Khandekar. Overview of optical flow technique for mobile robot obstacle avoidance. In *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*, pages 619–622, 2018.

[82] Gengshan Yang and Deva Ramanan. Volumetric correspondence networks for optical flow. In *NeurIPS*, 2019.

[83] Dong-Hoon Yi, Tae-Jae Lee, and Dong-Il "Dan" Cho. Afocal optical flow sensor for reducing vertical height sensitivity in indoor robot localization and navigation. *Sensors*, 15(5):11208–11221, 2015.

[84] Dong-Hoon Yi, Tae-Jae Lee, and Dong-Il Dan" Cho. Afocal optical flow sensor for mobile robot odometry. In *2015 15th International Conference on Control, Automation and Systems (ICCAS)*, pages 1393–1397, 2015.

[85] Dong-Hoon Yi, Taejae Lee, and Dong-Il Cho. A new localization system for indoor service robots in low luminance and slippery indoor environment using afocal optical flow sensor based sensor fusion. *Sensors*, 18:171, 01 2018.

[86] Jason J. Yu, Adam W. Harley, and Konstantinos G. Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In Gang Hua and Hervé Jégou, editors, *Computer Vision – ECCV 2016 Workshops*, pages 3–10, Cham, 2016. Springer International Publishing.

[87] Tianguang Zhang, Xiaodong Liu, Kolja Kühnlenz, and Martin Buss. Visual odometry for the autonomous city explorer. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3513–3518, 2009.

[88] Yi Zhu, Zhenzhong Lan, Shawn Newsam, and Alexander Hauptmann. Guided optical flow learning. 02 2017.