Thèse de Master

Intelligence artificielle et robotique développementale bio-inspirée pour le positionnement visuel basse consommation d'un robot par rapport à une cible

Octobre 2021

Progress Internship Report

A dissertation submitted to

Université d'Evry Val d'Essonne

CY Université de Cergy-Pontoise

Laboratoire Traitement de l'Information et des Systèmes "ETIS" site St-Martin, France

for the degree of

European Master of Science in

Smart Aerospace and Autonomous Systems

Presented by

Fatma Khalil Gamal Khalil Mohamed

Under the excellent supervision of

Dr. Arnaud Blanchard
Maitre de Conférences
Laboratoire ETIS
CY Université de Cergy-Pontoise

Prof. Lola cañamero
Chaire de Robotique et Neurosciences
Laboratoire ETIS
CY Université de Cergy-Pontoise

Dedicated to my amazing parents Khalil and Entesar

To my sisters, and soul mates; Sarah, Toka and Maha

Acknowledgement

First of all, I would like to thank **Dr. Arnaud Blanchard** for having offered me the possibility of doing an internship under his excellent supervision, support and help but also for the freedom he gave me. Being part of Dr. Arnaud's group -Laboratoire Traitement de l'Information et des Systèmes "ETIS"- was a highly appreciated experience and led to many happy moments. I wish to express my sincere appreciation to my supervisor Dr. Arnaud, He convincingly guided and encouraged me to be professional and do the right thing even when the road got tough. Without his persistent help, the goal of this project would not have been realized.

I am thankful for **Prof. Lola cañamero** for having me offered the internship. I also wish to thank all my professors whose assistance was a milestone in the completion of this project. With the high appreciation to all the professors and staff efforts of SAAS M2 program specially **Prof. Naïma Ait Oufroukh**, for having offered me the place in the program, Year 2020 - 2021, and for her much appreciated efforts and help even before coming to France.

I would also like to thank my advisor and mentor since my undergraduate years till present **Prof. Osama Saaid** . who guided me all the way long willing only to help and support me throughout all aspects of life and career decisions, and I will be always grateful for his support and guidance.

Ultimately, I would to like to express my love and appreciation for the people I owe them my life, my stubborn great succesful mother **Entesar,** who always gave me the power to go on**,** my supportive lovely father **Khalil**, who inherited me the passion for Engineering, my sisters who always gave me unconditional love and support, **Sarah**, **Toka** and **Maha**. They have always encouraged me during my study and work, believed in my abilities and pushed me to be the best version of myself. They helped me take the right decisions during my life . They are the reason for this work to come to light, Thank you.

Fatma Khalil

Paris, October 2021

Abstract

In this dissertation, we discuss the possibility of creating a bio-inspired low cost visual positiong algorithm for a follower robot (following another robot, object or human). The technique we are using is new and innovative, that we couldn't find almost any references or previous work with the same idea. However, we strongly believe it's going to be the foundation for more researches in the same field.

However, a few scientists believe that our Central Nervous system is working in a way that's similar to using Z-transform on images to process them. In addition, Z-transform has many properties that we can get use of while dealing with images. This will create a much simpler algorithm which will result in reducing the cost of that algorithm.

At the beginning we are speaking about the possible solutions for visual positioning algorithms, After testing, many of them were robust, real-time and powerful like ORB SLAM, but also has its own limitations that's critical for some applications. However, we are more interested to create a bio-inspired algorithm that could act more or less as the human eyes. By the time of writing the thesis, robustness isn't achieved through our algorithm, but the significant simplicity over other commercial algorithms is an encouraging point.

By end of the report, we will speak more about the future work. That will include the performance and optimization methods for our current algorithm and will be able to deal with changes in brightness, saving landmarks in the memory and moreover to understand and analyze the emotions of the object it follows (in case it's Human).

Lab representation

ETIS , the Information Processing and Systems Teams laboratory, is a joint research team at ENSEA and the University of Cergy-Pontoise, recognized by the CNRS (UMR 8051).

ETIS has around 80 members , half of which are teacher-researchers (ENSEA and University of Cergy-Pontoise). ETIS is organized into **four research teams** , spread across the sites of ENSEA and the University of Cergy-Pontoise (St-Martin 1):

1. Architecture, Systems, Technologies for Reconfigurable Embedded Units (ASTRE) :

   The team is mainly interested in algorithm-architecture fit, IP design and Reconfigurable systems on a chip (RSoC)

2. Information, Communications, Imaging (HERE) :

   The team is interested in mainly four aspects but not enclosed to them.

   - linear algebra
   - Imagery
   - LDPC codes and iterative receivers
   - Resource allocation for the physical layer

3. Multimedia Indexing and Data Integration (MIDI) :

   The researchs in this team are interested in multimedia indexing and data integration.

4. Neurocybernetics, which is the team supervising this dissertion, the researchers' main interests are:

   - visual perception
   - learning and imitation
   - navigation and planning
   - complex systems

For more information : http://www-etis.ensea.fr

Contents

List of Figures

Listings

Nomenclature

Acronyms

ETIS	Equipes Traitement de l'Information et Systèmes
FFT	Raspberry Pi
FT	Fourier Transform
GBA	Global Bundle Adjustment
JPEG	Joint Photographic Experts Group
KSLAM	Keyframe-based monocular SLAM systems
LAN	Local Architecture Network
LBA	Local Bundle Adjustment
RPI	Raspberry Pi
SLAM	Simultaneous Localization and Mapping
VO	Visual Odometry
ZT	Z-Transform

Chapter 1

Introduction

1.1 Motivation and Objectives

In modern era, to lower the burden of labor on mankind effortlessly, physical tasks are being performed by machines. Nevertheless, there is a need of special thing that machines not only do physical tasks but also can think and make decision like human being[4]. To make things intelligent and to get this target, artificial intelligence knowledge has got important consideration. Path planning has been considered as the most-common problem for robot navigation, robots have to move from starting position to goal position and avoid obstacles [1]. In the beginning, the researchers have focused on 2D path planning, therefore avoiding the bio-inspired algorithms. Later on, They found out that bio-inspired algorithms can be the foundation to create a smarter and more intelligent robot. To understand more about bio-inspired algorithms, It was necessary to discover how the learning process is happening in all creatures (Humans and animals) and use a similar concepts for the algorithms of the robots.

The Naama region in Algeria has been chosen to set up a reforesting pilot site, is situated in the western area of the High Plains, close to the frontier between Morocco and Algeria. This territory is of a steppic nature located between two mountainous massifs in the Atlas, and is subject to silting and salinization [[18]]. The average density of the population is 6.85 inhabitants per sq.km. Cattle breeding is the main activity of a large proportion of the rural population, an activity that has gradually led to the continuing disappearance of the vegetation, and has contributed to further degradation of the soil over the past thirty years, a situation worsened by climatic conditions. During discussions with local authorities, it became apparent that robotics could offer a solution which — if costs involved were acceptable and implementation simple — would allow for high-scale reforesting and resilience of the biodiversity of these regions. In order to make the fleet of mobile robots autonmous, It was needed a robust navigation algorithm which is an interesting objective in common between our objective and the author's as well.

Mechanisms of imitation and social matching play a fundamental role in development, communication, interaction, learning and culture. Their investigation in different agents (animals, humans and robots) has significantly influenced our understanding of the nature and origins of social intelligence. Whilst such issues have traditionally been studied in areas such as psychology, biology and ethology, it has become increasingly recognized that a 'constructive approach' towards

imitation and social learning via the synthesis of artificial agents can provide important insights into mechanisms and create artifacts that can be instructed and taught by imitation, demonstration and social interaction rather than by explicit programming [10].

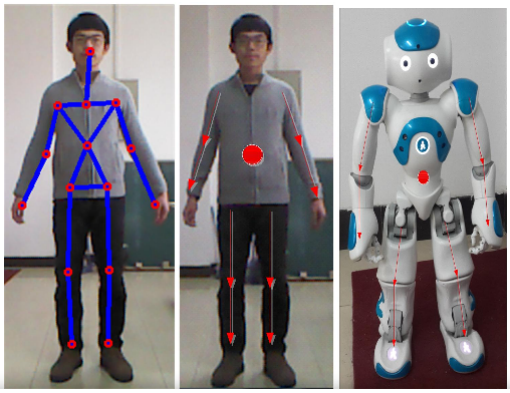

Figure 1.1: The imitating robot NAO

Bio-inspired robotic locomotion is a fairly new subcategory of bio-inspired design. It is about learning concepts from nature and applying them to the design of real-world engineered systems. Biomimicry and bio-inspired design are sometimes confused. *Biomimicry* is copying from nature while *bio-inspired* design is learning from nature and making a mechanism that is simpler and more effective than the system observed in nature. *Biomimicry* has led to the development of a different branch of robotics called *soft robotics*.

- **Hexa, The Six-Legged Robot:**Hexa from Vincross is a new six-legged robot that allows users to control it from their smartphone. The robot uses a variety of sensors to determine its orientation and navigate around terrain. It has the ability to scale steps and balance on uneven ground without having the need to control each individual leg. The six-legged design allows the robot to save energy by having better balance capability[12].
  According to Andy Xu, COO of Vincross, the robot “only needs three legs to stand on the ground, and we can use the other three legs to maintain balance or climb stairs.” The sensors in the Hexa bot are a digital camera with night vision, a 3-axis accelerometer, distance-measuring sensor, and an infrared transmitter. The programming language for the robot is open-source, allowing multiple users to experiment with the robot and fostering a programming marketplace. Vincross hopes to launch the robot for multiple uses as the user community grows, including exploration of dangerous situations like collapsed buildings.

Figure 1.2: Hexa, The Six-Legged Robot

- **Spot Mini from Boston Dynamics:** Boston Dynamics is one of the leaders of bio-robot design. Makers of possibly some of the best-known bio-robots like the *Wildcat* and *Atlas*, one of their latest inventions may have a future place in your home and office. The Spot Mini is a small four-legged robot that weighs 30 kilograms (including a robotic arm) and is completely electric, having a power range of 90 minutes on a single charge. The robot can pick up and handle objects using its 5 degree of freedom arm and is equipped with a sensor suit that includes stereo cameras, depth cameras, an inertial measurement unit, and position/force sensors in the limbs. The robot is small and quiet enough to be used in your office and home and has the ability to run autonomous tasks [12].

Figure 1.3: Spot Mini from Boston Dynamics

- **A Bio-Inspired Flying Robot:** The robot was designed by Biorobotics Lab Movement and Perception Institute Centre National de la Recherche Scientifique and University of the Mediterranean in Marseille, France. It was noticed that When insects are flying forward, the image of the ground sweeps backward across their ventral viewfield and forms an "optic flow," which depends on both the groundspeed and the groundheight.

  To explain how these animals manage to avoid the ground by using this visual motion cue, The team of the lab suggested that insect navigation hinges on a visual-feedback loop that have been called the optic-flow regulator, which controls the vertical lift. To test this idea, They built a micro-helicopter equipped with an optic-flow regulator and a bioinspired optic-flow sensor. This fly-by-sight microrobot can perform exacting tasks such as take-off, level flight, and landing[2].

  The control scheme accounts for many hitherto unexplained findings published during the last 70 years on insects' visually guided performances; for example, it accounts for the fact that honeybees descend in a headwind, land with a constant slope, and drown when travelling over mirrorsmooth water. While the opticflow regulator is quite simple in terms of its neural implementation and just as appropriate for insects as it would be for aircraft.

1.2 Literature review

1.2.1 VO vs. SLAM

In the past few decades, the area of mobile robotics and autonomous systems has attracted substantial attention from researchers all over the world, resulting in major advances and breakthroughs. Currently, mobile robots are able to perform complex tasks autonomously[1]. Mobile robotics has applications in various fields such as military, medical, space, entertainment and domestic appliances fields. In those applications, mobile robots are expected to perform complicated tasks that require navigation in complex and dynamic indoor and outdoor environments without any human input. In order to autonomously navigate, path plan and perform these tasks efficiently and safely, the robot needs to be able to localize itself in its environment.

As a result, the localization problem has been studied in detail and various techniques have been proposed to solve the localization problem. The simplest form of localization is to use wheel odometry methods that rely upon wheel encoders to measure the amount of rotation of robots wheels. In those methods, wheel rotation measurements are incrementally used in conjunction with the robot's motion model to find the robot's current location with respect to a global reference coordinate system. The wheel odometry method has some major limitations.

Firstly, it is limited to wheeled ground vehicles and secondly, since the localization is incremental (based on the previous estimated location), measurement errors are accumulated over time and cause the estimated robot pose to drift from its actual location. There are a number of error sources in wheel odometry methods, the most significant being wheel slippage in uneven terrain or slippery floors. To overcome those limitations, other localization strategies such using inertial measurement units (IMUs), GPS, LASER odometry and most recently Visual Odometry (VO) and Simultaneous Localization and Mapping (SLAM) [6] methods have been proposed.

VO is the process of estimating the egomotion of an agent (e.g., vehicle, human, and robot) using only the input of a single or multiple cameras attached to it [1]. Whereas SLAM is a process in which a robot is required to localize itself in an unknown environment and build a map of this environment at the same time without any prior information with the aid of external sensors (or a single sensor). Although VO does not solve the drift problem, researchers have shown that VO methods perform significantly better than wheel odometry and dead reckoning techniques while the cost of cameras is much lower compared to accurate IMUs and LASER scanners. The figure below shows an overview of VO and SLAM systems.

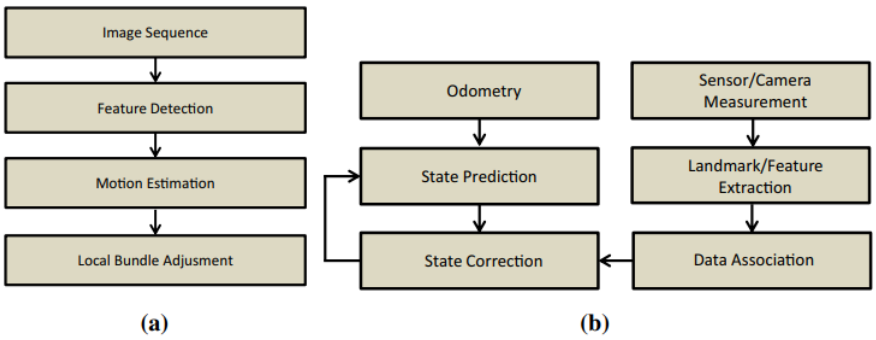

Figure 1.4: A block diagram showing the main components of a: (a) VO and (b) filter based SLAM system

While research on visual simultaneous localization and mapping (SLAM) systems and visual odometry (VO) has been an interesting research topic in the last few year, using cameras either alone or in combination with inertial sensors, has produced, during the last two decades, excellent systems, with increasing accuracy and robustness. Modern systems rely on maximum a posteriori (MAP) estimation which, in the case of visual sensors, corresponds to bundle adjustment (BA), either geometric BA that minimizes feature reprojection error, in feature-based methods, or photometric BA that minimizes the photometric error of a set of selected pixels, in direct methods[3] . With the recent emergence of VO systems that integrate loop closing techniques, the frontier between VO and SLAM is more diffuse. The goal of visual SLAM is to use the sensors on-board a mobile agent to build a map of the environment and compute in real time the pose of the agent in that map. In contrast, VO systems put their focus on computing the agent's ego-motion and not on building a map. The big advantage of a SLAM map is that it allows matching and using in BA previous observations performing three types of data association;

1. Short-term data association: matching map elements obtained during the last few seconds. This is the only data association type used by most VO systems, which forget environment elements once they get out of view, resulting in continuous estimation drift even when the systemmoves in the same area.

2. Mid-term data association: matching map elements that are close to the camera whose accumulated drift is still small. These can be matched and used in BA in the same way than short-term observations and allow to reach zero drift when the systems move in mapped areas. They are the key to the better accuracy obtained by our system compared against VO systems with loop detection.

3. Long-term data association: matching observations with elements in previously visited areas using a place recognition technique, regardless of the accumulated drift (loop detection), the current area being previously mapped in a disconnected map (map merging), or the tracking being lost (relocalization). Long-term matching allows to reset the drift and to correct the map using pose-graph (PG) optimization or, more accurately, using BA. This is the key to

SLAM accuracy in medium and large loopy environments

1.2.2 Introduction to SLAM

Monocular SLAM solutions are either *filter-based*, such as using a Kalman filter; or *keyframe-based*, relying on optimization to estimate both motion and structure. In filter-based systems, the localization and mapping are intertwined: the camera pose T_n, with the entire state of all landmarks in the map, are tightly joined and need to be updated at every processed frame as shown in 1.5 . On the other hand, in *keyframe-based* systems, Localization and Mapping are separated into two steps: camera localization takes place on regular frames over a subset of the map (gray nodes in 1.5), whereas keyframe-based optimization takes place on Keyframes[[6]]. As a consequence of these differences, *Strasdat et al. in 2010* showed that keyframe based methods outperform filter-based ones; and it is therefore not surprising to note that most new releases of monocular SLAM systems are keyframe-based.

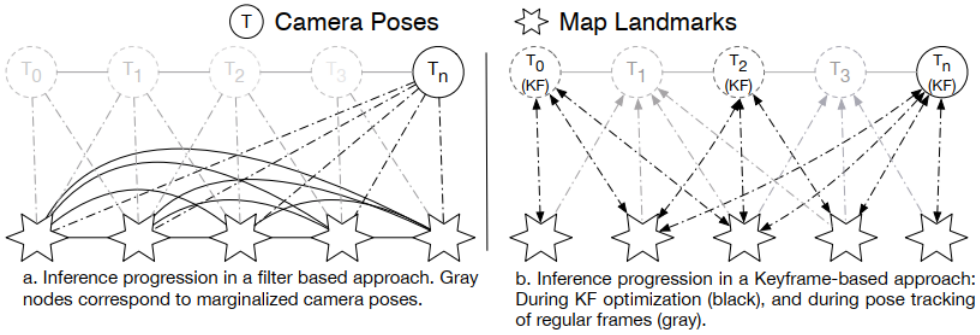

Figure 1.5: Inference in filter based vs. Keyframe-based VSLAM

The flow or almost all KSLAM (Keyframe-based VSLAM) algorithms at startup is as follows; KSLAM has no prior information about the camera pose nor the scene structure, the **visual initialization module** is responsible for establishing an initial 3D map and the first camera poses in the system. The visual initialization is the entry point of a KSLAM and runs only once at startup. When a new frame is available, **data association** uses the previous camera poses to guess a pose for the new frame; the predicted pose is used to establish associations with the 3D map. An error vector is then found as the difference between the true measurements and their associated matches generated using the guessed pose. The error vector is iteratively minimized in the **pose optimization module**, using the guessed pose as a starting point. If the minimization diverges or the data association fails, **failure recovery** procedures are invoked. For regular frames, the pipeline ends here however, if the frame was chosen as a keyframe, it is used to triangulate new landmarks,

thus expanding the 3D map. To ensure the coherency of the map, reduce errors and remove outliers, **map maintenance** continuously optimizes the map while another parallel process attempts to detect **loop closures** and accordingly minimize the errors accumulated over the traversed loop.

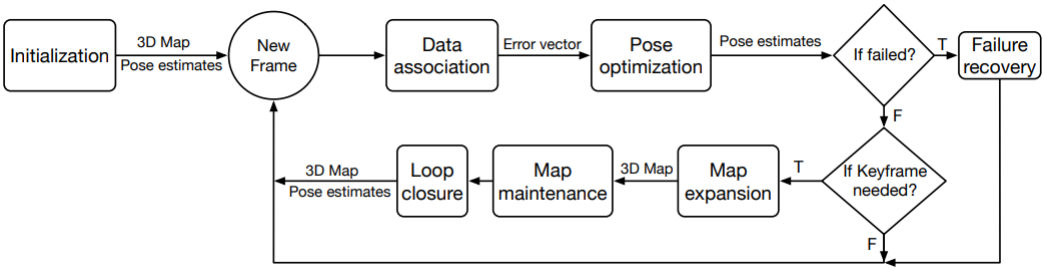

Figure 1.6: KSLAM Flowchart

1. **Visual initialization**:

   This module is responsible for establishing an initial 3D map and the first camera poses in the system. The visual initialization is the entry point of a KSLAM and runs only once at startup.

   Since monocular cameras are bearing-only sensors, which cannot directly perceive depth; nevertheless, a scaled depth can be estimated via temporal stereoscopy, after observing the same scene through at least two different viewpoints.

   After KSLAM is initialized, camera pose and 3D structure build on each other, in a heuristic manner, to propagate the system in time, by expanding the map to previously unobserved scenes, while keeping track of the camera pose in the map. The problem is more difficult during initialization, since neither pose, nor structure is known.

   In early monocular SLAM systems, such as in MonoSLAM , initialization required the camera to be placed at a known distance from a planar scene, composed of four corners of a two dimensional square; the user initialized SLAM by keying in the distance separating the camera from the square.

   Thereafter, to lessen these constraints, researchers adopted the methods developed by to simultaneously recover the camera pose and the 3D scene structure. The elimination of depth has significant ramifications on the recovered data: since the exact camera motion between the two views cannot be recovered, the camera translation vector is recovered up to an unknown scale λ . Since the translation vector between the two views defines the baseline used to triangulate 3D landmarks, scale loss also propagates to the recovered 3D landmarks, yielding a scene that is also scaled by λ.

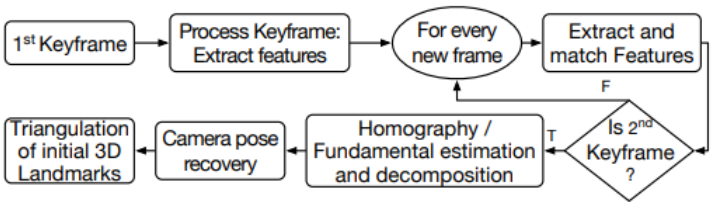

Figure 1.7: Generic model-based initialization flowchart.

Figure 1.7 shows the flowchart of a generic model-based initialization; the first frame processed by the KSLAM system is typically set as the first keyframe. Subsequent frames are processed by establishing 2D-2D data associations, which are monitored to decide whether the new frame is the second keyframe or not. The decision criteria is based on the 2D distances between the found matches in both images. The matches are then used to estimate a Homography (degenerate for nonplanar scenes) or a Fundamental matrix (degenerate for planar scenes) using a robust model fitting method. The estimated Homography or the Fundamental matrix are then decomposed into an initial scene structure and initial camera poses [7]. To mitigate degenerate cases, random depth initialization, as its name suggests, initializes a KSLAM by randomly assigning depth values with large variance to a single initializing

2. **Data association**: When a new frame is available after startup, data association uses the previous camera poses to guess a pose for the new frame; the predicted pose is used to establish associations with the 3D map.

   (a) Direct methods exploit the information available at every pixel, where pixel values surrounding a location of interest (in this case a triangle) are aligned by a transformation that minimizes the intensity values between the two locations of interest in both images. In practice a region of interest is defined as a square surrounding a pixel. The basic underlying principle for all direct methods is known as the brightness consistency constraint and is best described as:

   $$J(x,y,t) = I(x+u(x,y), y+v(x,y), t+1)$$

   where x and y are pixel coordinates; u and v denote displacement functions of the pixel (x,y) between two images I and J of the same scene taken at time t and$t+1$ respectively. The brightness consistency constraint is based on the assumption that a point from the world's surface, observed in image I, will have the same intensity when observed in a subsequent image J.

   Researchers used to solve this equation by FAIA Method (Forward Additive Image Alignmen) to replace all the individual pixel displacements u and v by a single general

motion model, in which the number of parameters is dependent on the implied type of motion. FAIA iteratively minimizes the squared pixel-intensity difference between the two images over the transformation parameters p : as explained in .

$$argmin\sum_{x,y}[I(W(x,y,p)) - J(x,y)]^2$$

where $W(.,.,p)$ is a warping transform that encodes the relationship relating the two images and p corresponds to the parameters of the transform. is non-linear and requires an iterative non-linear optimization process, with a computational complexity of $O(n^2N + n^3)$ per iteration, where n is the number of parameters in p and N is the number of pixels in the image. But the usage of this solution has been improved to ease some of it's complexity since 1981.

(b) **Feature-based methods** : Feature-based methods were introduced to reduce the computational complexity of processing each pixel; this is done by matching only salient image locations, referred to as features, or keypoints. A descriptor is associated to each feature, which is used to provide a quantitative measure of similarity to other keypoints. On one hand, features are expected to be distinctive, invariant to viewpoint and illumination changes, as well as resilient to blur and noise; on the other hand, it is desirable for feature extractors to be computationally efficient and fast [3]

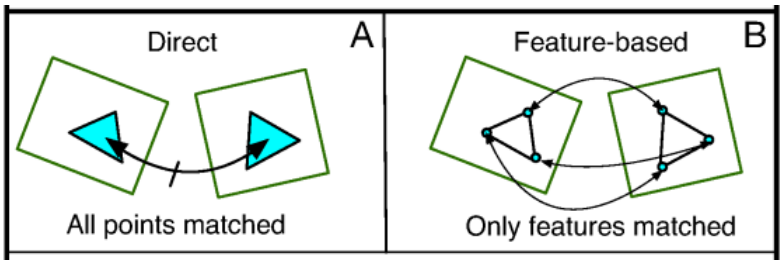

Figure 1.8: Direct vs. Feature-based data association

3. **Pose optimization**:

After guessing the pose for the new frame from the data association module. An error vector is then found as the difference between the true measurements and their associated matches generated using the guessed pose. The error vector is iteratively minimized in the pose optimization module, using the guessed pose as a starting point. the camera pose is estimated by different data association methods by minimizing a measure of error between frames; **direct methods** measure the photometric error, modeled as the intensity difference between pixels; in contrast, **feature-based** methods measure the re-projection error of landmarks from the map over the frame's prior pose. The re-projection error is formulated as the distance in pixels between a projected 3D landmark onto a frame, and its corresponding

2D position in the image. A motion model is used to seed the new frame's pose at C_m , and a list of potentially visible 3D landmarks from the map are projected onto the new frame. Data association takes place in a search window S_w surrounding the location of the projected landmarks. KSLAM then proceeds by minimizing an error vector (the geometric distance d in the case of feature-based methods or the intensity residuals in the case of direct methods) over the parameters of the rigid body transformation. To gain robustness against outliers, the minimization takes place over an objective function (Huber norm) that penalizes features with large errors. The camera pose optimization problem is then defined as in

$$T_i = \underset{T_i}{argmin} \sum_j Obj(e_j)$$

where T_i is a minimally represented Lie group of either $S\xi(3)$ or $sim(3)$ camera pose, $Obj(.)$ is an objective function and $e_j$is the error defined through data association for every matched feature j in the image. Finally, the system decides whether the new frame should be flagged as a keyframe or not.

4. **Topological/metric map generation** :

The map generation module is responsible for generating a representation of the previously unexplored, newly observed environment. Typically, the map generation module represents the world as a dense (for direct) or sparse (for feature-based) cloud of points [3]. The different viewpoints of an unexplored scene are registered with their corresponding camera poses through the pose tracking module. The map generation module then re-establishes data association between the new keyframe and a set of keyframes surrounding it, looking for matches. It then triangulates 2D points of interest into 3D landmarks; it also keeps track of their 3D coordinates, and expands the map within what is referred to as a metric representation of the scene. There are two types of maps to be generated;

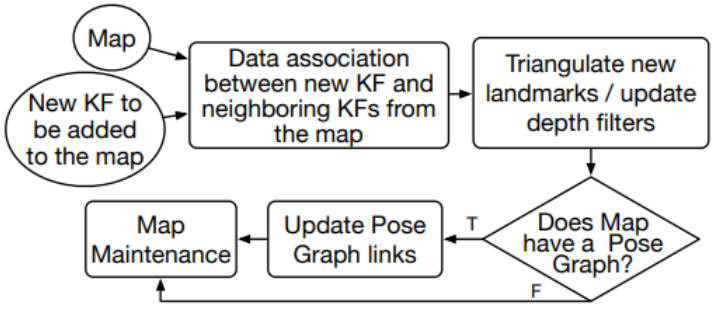

Figure 1.9: Map generation flowchart

(a) **Metric Maps:** In a metric map, the structure of new 3D landmarks is recovered, given

the pose transformation relating two keyframes observing the landmark, from epipolar geometry using the corresponding data associations between the keyframes.

(b) **Topological maps:** When a new keyframe is added into systems that employ hybrid maps, their topological map is updated by incorporating the new keyframe as a node, and searching for data associations between the newly added node and surrounding ones; edges are then established to other nodes (keyframes) according to the number of found data associations: the thickness of the edges connecting the nodes is proportional to the number of common landmarks observed in those nodes.

5. **Map maintenance**:

   The map maintenance module continuously optimizes the map while another parallel process attempts to detect loop closures and accordingly minimize the errors accumulated over the traversed loop. Map maintenance optimizes the map through either **bundle adjustment** or **pose graph optimization** . During map exploration, new 3D landmarks are triangulated based on the camera pose estimates; after some time, system drift manifests itself in wrong camera pose measurements, due to accumulated errors in previous camera poses that were used to expand the map. Map maintenance proceeds by establishing data association between the entire set of keyframes in the map or a subset of keyframes and performs a global bundle adjustment (GBA) or a local bundle adjustment (LBA) respectively. Outlier landmarks flagged from the optimization are then culled (removed from the map). To reduce the complexity of the optimization, redundant keyframes are also culled [8]. Map maintenance is also responsible for detecting and optimizing loop closures as well as performing a dense map reconstruction for systems that allow for it.

6. **Failure recovery**:

   The ability to recover the system and correct the map due to to any failure that may occur, whether due to wrong user movement, such as abrupt changes in the camera pose resulting in motion blur, or due to observing a featureless region, or for any other reason. Accordingly, a key module essential for the usability of any monocular SLAM system is its ability to correctly recover from such failures.

7. **Loop closure**:

   This module always attempts to minimize the errors accumulated over the traversed loop. Since keyframe-based monocular SLAM is an optimization problem, it is prone to drifts in camera pose estimates. Returning to a certain pose after an exploration phase may not yield the same camera pose measurement, as it was at the start of the run. Such camera pose

drift can also manifest itself in a map scale drift, which will eventually lead the system to erroneous measurements, and fatal failure. To address this issue, some algorithms detect loop closures in an online monocular SLAM session, and optimize the loops track in an effort to correct the drift and the error in the camera pose and in all relevant map data that were created during the loop. The loop closure thread attempts to establish loops upon the insertion of a new keyframe, in order to correct and minimize any accumulated drift by the system over time depending on **G2o** Libraries.

1

[1] **G2o** is an open-source C++ framework for optimizing graph-based nonlinear error functions. G2o has been designed to be easily extensible to a wide range of problems and a new problem typically can be specified in a few lines of code. The current implementation provides solutions to several variants of SLAM and BA.[14]

1.3 Contributions and problem statement

This report examins in details the issue of robot vision-based navigation,where the algorithms provided in this domain uses computer vision algorithms and optical sensors, including laser-based range finder and photometric cameras to extract the visual features required to the localization in the surrounding environment. However, there are a range of techniques for navigation and localization using vision information, the main components of each technique are:

representations of the environment. sensing models. localization algorithms. While our main objective is to create a bio-inspired, robust, low cost positioning algorithm, we had to go through the process of studying the current algorithms and implement many of them on different platforms, which will present the most part of this thesis. There are many different algorithms that can sole a navigation problem for a robot, but the most robust, verstaile and accurate alogrithm currently used in different aspects is ORBSLAM. This is why it was chosen to be implemented first to compare the results from our algrithm to the results from the ORBSLAM algorithm. This will also encourage us to study three aspects

- The different updated methods and how SLAM algorithms work in first place then test the robustness and cost of each algorithm.
- How our Central Nervous System work.
- Imitate our Nervous System methods of processing information from the eyes to the control system.

1.4 Structure of this dissertion

This dissertion as mentioned in Chapter 1 discussed many concepts of different visual navigation algorithms used in computer vision field, thier advantages and disadvantages. We don't make a dierence any biological inspiration at this stage yet. We focused on discovring different methods similar to SLAM methods stated in the Literature review.

In Chapter 2, We try to investigate other solutions that may enable us creating a low cost visual algorithm, We start the investigation with defining different image transforms , checking different limitations and benefits of using those image transforms. Later on, the chapter introduces the z-transform from a mathematical point of view and moreover disscusses why Z-transform is bio-inspired. After, we progressively present our Method and algorithms to apply the Z-transform on the visual positioning of a robot.

In Chapter 3, We explain the Methodology, Hardware configuration and guides to easily implement the same project again and avoid all errors and possible mistakes that can occur due to the complexity of connections between different components.

In Chapters 4, we go through the advantages and limitations of both algorithms, according to robustness, computational cost and accuracy. Summarizing our work in chapter 5, where our outcomes are placed into point of view comparative with signicant level applications, for example, Robustness and Mapping algorithms are discussed in the Future work.

Chapter 2

Image Transforms

2.1 Introduction

presenting the Z-transform as a possible solution has forced us to clarify why not any other image transforms, although there exist many algorithms depending on Fourier transform for example. To discuss that we need to know about the limitations and benefits of each image transform. An image transform can be applied to an image to convert it from one domain to another. Viewing an image in domains such as frequency or Hough space enables the identification of features that may not be as easily detected in the spatial domain. Common image transforms include[12]:

- **Hough Transform**: used to find lines in an image
- **Discrete Cosine Transform**: used in image and video compression
- **Discrete Fourier Transform**: used in filtering and frequency analysis
- **Wavelet Transform**: used to perform discrete wavelet analysis, denoise, and fuse images.
- **Z-transform**: used to transfer images through networks, i.e., image compression and regeneration. and this is the transform we are using in this thesis.

2.2 The 2D Fourier Transform

It is the series expansion of an image function (over the 2D space domain) in terms of "cosine" image (orthonormal) basis functions. The definitons of the transform (to expansion coefficients) and the inverse transform are given below:

$$F(u,v) = \sum f(x,y) * e^{-2j\pi*(u*x+v*y)/N} \tag{2.1}$$

where $f(x,y)$ denotes the image and $F(u,v)$denotes the Fourier transform.

For example, (Fig.2.1) shows 2 images with their Fourier Transforms directly underneath. The images are a pure horizontal cosine of 8 cycles and a pure vertical cosine of 32 cycles. Notice that the FT for each just has a single component, represented by 2 bright spots symmetrically placed about the center of the FT image. The center of the image is the origin of the frequency coordinate system. The u-axis runs left to right through the center and represents the horizontal component of frequency. The v-axis runs bottom to top through the center and represents the vertical component of frequency. In both cases there is a dot at the center that represents the (0,0) frequency term or average value of the image. Images usually have a large average value (like 128) and lots of low frequency information so FT images usually have a bright blob of components near the center. Notice that high frequencies in the vertical direction will cause bright dots away from the center in the vertical direction. And that high frequencies in the horizontal direction will cause bright dots away from the center in the horizontal direction.

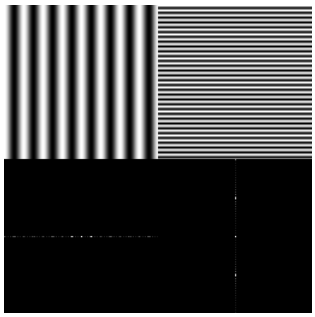

Figure 2.1: Fourier transform of two images

To understand more how the Fourier transform is used for real images, we can explain this example. Notice there is very little structure. You can see a top left to bottom right slanting line in the girl image FT. It is probably due to the edge between her hat and her hair. There are also some small edge effects in both images. The mandril image appears to have more high frequency power, probably due to the hair.

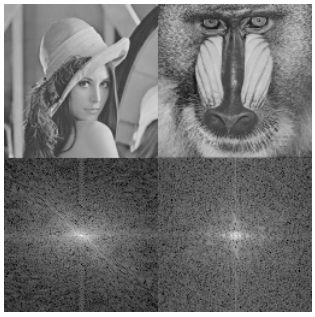

Figure 2.2: Fourier transform for Real Images

We can conclude that the FT tries to represent all images as asummation of cosine-like images as explained by equation2.1. Therefore,images that are pure cosines have particularly simple FTs.

Anytime an image has a strong-contrast,sharp edge the gray values must change very rapidly. It takes lots of high frequency power to follow such an edge as shown in (Fig.2.3). As a sequence, we can't use the Fourier transform in our algorithm.

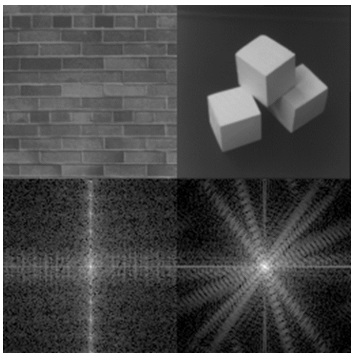

Figure 2.3: Fourier transform of sharp edges images

2.3 Wavelet Transform

A wavelet is a mathematical function useful in digital signal processing and image compression . The use of wavelets for these purposes is a recent development, although the theory is not new. The principles are similar to those of Fourier analysis, which was first developed in the early part of the 19th century[[13]].

While the basic wavelet equation is

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a}) \tag{2.2}$$

Where a represents the dilation (Scale) and b representsthe translation.

In signal processing, wavelets make it possible to recover weak signals from noise . This has proven useful especially in the processing of X-ray and magnetic-resonance images in medical applications. Images processed in this way can be "cleaned up" without blurring or muddling the details.

In Internet communications, wavelets have been used to compress images to a greater extent than is generally possible with other methods. In some cases, a wavelet-compressed image can be as small as about 25 percent the size of a similar-quality image using the more familiar JPEG method. Thus, for example, a photograph that requires 200 KB and takes a minute to download in JPEG format might require only 50 KB and take 15 seconds to download in wavelet-compressed format.

However, a more interesting usage of wavelet transform in compressing the images while keeping the good quality is JPEG 2000 transformation as shown in Fig.3.42.4 where the upper image is the original one. you can notice the difference in the quality between the second and

third images. The second image represents the 19Kbyte compression using JPEG2000 which has almost preserved all the image details while the third image has signifcant loss of quality and details although it's the same 19Kbyte compression. Wavelet compression works by analyzing an image and converting it into a set of mathematical expressions that can then be decoded by the receiver.

Figure 2.4: JPEG2000 compression using wavelets

2.4 Z-Transform

2.4.1 Z-Transform Algorithm

The z-transform is a generalization of the Fourier analysis in that it uses a larger set of basis functions. The set becomes larger because the basis functions include exponentially varying amplitude sinusoids. Obviously, the Fourier analysis is a special case of the z-transform, if the z-transform exists on the unit circle.

The z-transform (bilateral) X(z) of a discrete signal x(n) is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) * z^{-n} \tag{2.3}$$

where z is a complex variable. The z-transform is defined in the region of its convergence of the complex plane. Constant-amplitude sinusoids correspond to the independent variable$z = e\,j\omega$ on the unit circle. Decaying-amplitude sinusoids correspond to z inside the unit circle. Growing-amplitude sinusoids correspond to z outside the unit circle. The previous information regarding z-transform will be needed later to distingush the good and bad results.

2.4.2 Why Z-transform

As mentioned before that the FFT requires that your signal waveform have the same value at start and finish. This requirement is a consequence of FFT time wrapping, which makes the endpoint and beginning point actually adjacent in FFT time.

The normal way of harmonizing the beginning and ending of your excitation signal is to drive the system with a pulse, which first steps up, holds longs enough for you to see the resulting waveform, and then steps back down. After a suitable waiting interval, the system will have stabilized and the FFT time window may come to a close with zero excitation . The total FFT time window must therefore exceed the pulse duration plus one system stabilization time. I usually make the pulse duration half the width of the FFT time window and ensure that the total FFT time window exceeds twice the system stabilization time.

Another limitation of the FFT has to do with the maximum allowable rate of change for signals represented in discrete time. The Nyquist sampling theorem says that no band -limited signal can transition perfectly from one value to another in one time step without causing ripples in the adjacent samples. For example, if you try to work with a signal that incorporates a step change in signal amplitude at the end of the time window, it will cause ripples in both directions, distorting both the signal at the end of the time window and the samples near the beginning. To solve this problem, make sure the signal amplitude has by design the same amplitude (usually zero) at start and finish. So, we can conclude the reasons for two reasons;

1. The limitation of the FT has to do with the maximum allowable rate of change for signals represented in discrete time.which isn't the case with the Z-transform as we will explain more clearly in the follwing figure;

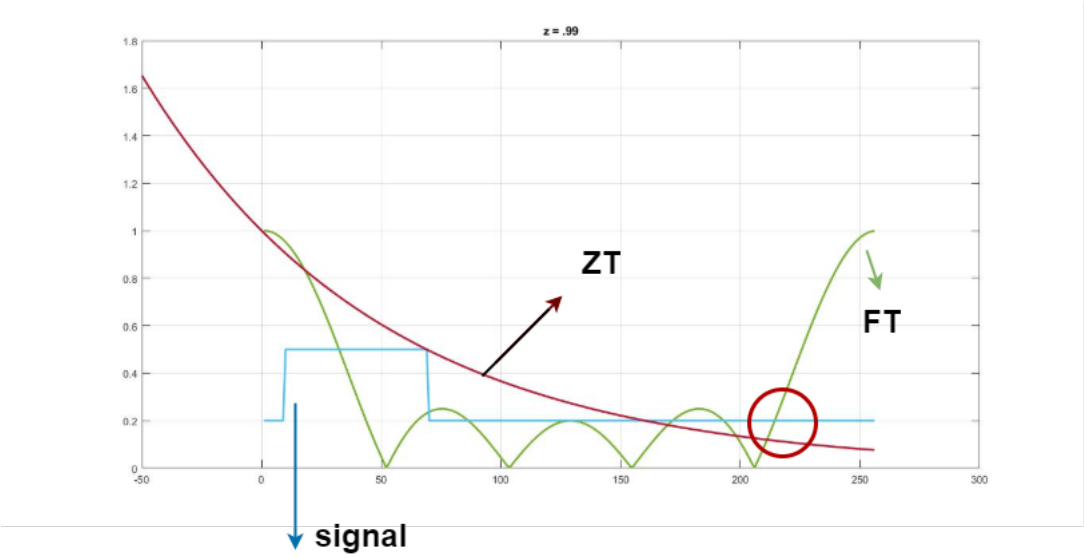

Figure 2.5: Fourier Transfrm vs. Z-Transfrm

Assuming a signal of a value 0.5 from (10 to 70) and a value 0.2 elsewhere; Calculating

both FT and ZT for it will result as shown above, the green line is the FT and the red line is the ZT.

As we can notice, the part of the signal marked with the red circle is still (pixel values) or a part of the image details. we can neglect this specific part using the ZT because the value of it is already too small (as the curve decaying) but we can't neglect it if we used the FT because it's fluctuating (similar to sinosidal wave). This is the main reason why we are using ZT not FT.

2. ZT is more bio-inspired algorithm than FT. as stated by (Howard et al, June 2018) [11] , who explained that;

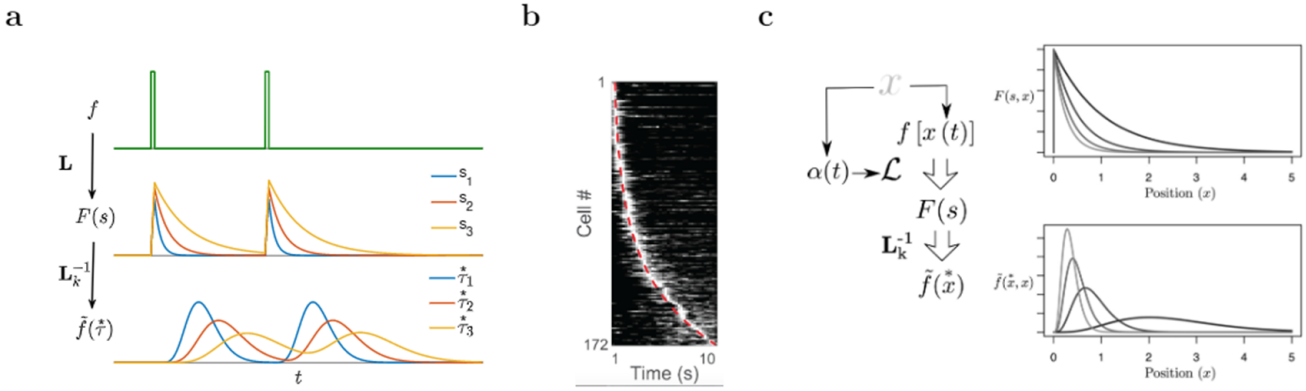

Figure 2.6: Laplace Transform in Human brain

- In fig(a). The Laplace transform of functions of time. Consider an input $f(t)$ that is non-zero for a brief period. After the input, the different neurons corresponding to the Laplace transform, $F(s)$ activate and then decay exponentially. Different neurons have different values of s and thus decay at different rates. Neurons in the inverse transform $\tilde{f}(\overset{*}{\tau})$ respond a characteristic time after the input. They fire when the past stimulus enters their receptive fields, defined by their value of $\overset{*}{\tau}$
- In fig.(b). Neural evidence for a function of time. If the brain contained sets of neurons that represented functions of time, we would see sequentially-activated neurons. Here we see sequentially activated neurons in the hippocampus recorded using calcium imaging. Each row gives the activation of one neuron as a function of time during the delay. The neurons are sorted according to their peak time. Note the curvature. This means that there are progressively fewer neurons coding for later in the delay. The dashed red line gives an analytic curve under the assumption of logarithmic compression.

- In fig. (c). Laplace transform for variables other than time.

  Left: Schematic for the method for constructing the Laplace transform of variables x in the world. With access to the time derivative of x , the method modifies the differential equation for maintaining the Laplace tranform of time by $\alpha(t) = dxdt$. If the input f is a function of $x(t)$, then $F(s)$ maintains the Laplace transform of $f(x)$.

  Right: Consider a case in which the animal encounters a spatial landmark at$x = 0$ causing an input via f and then moves in the neighborhood of the landmark$(withx > 0)$. As the animal moves away, $dx = dt > 0$ and the cells in $F(s)$ decay exponentially. But if the animal turns around and heads back toward the startng point, $dx = dt < 0$ and the cells grow exponentially. However, firing is always just a function of x. Under these circumstances, the Laplace transform behaves like border cells with different space constants and neurons participating in the inverse Laplace transform behave like place cells.

Chapter 3

Methodology

3.1 Configuration and errors (Hardware)

First of all, It was needed to create a hardware environment in order to test the algorithm. It needs to be cheap, simple and with acceptable camera quality (which isn't bio-inspired) but the algorithm should be a bio-inspired algorithm. The main components are as in 3.1

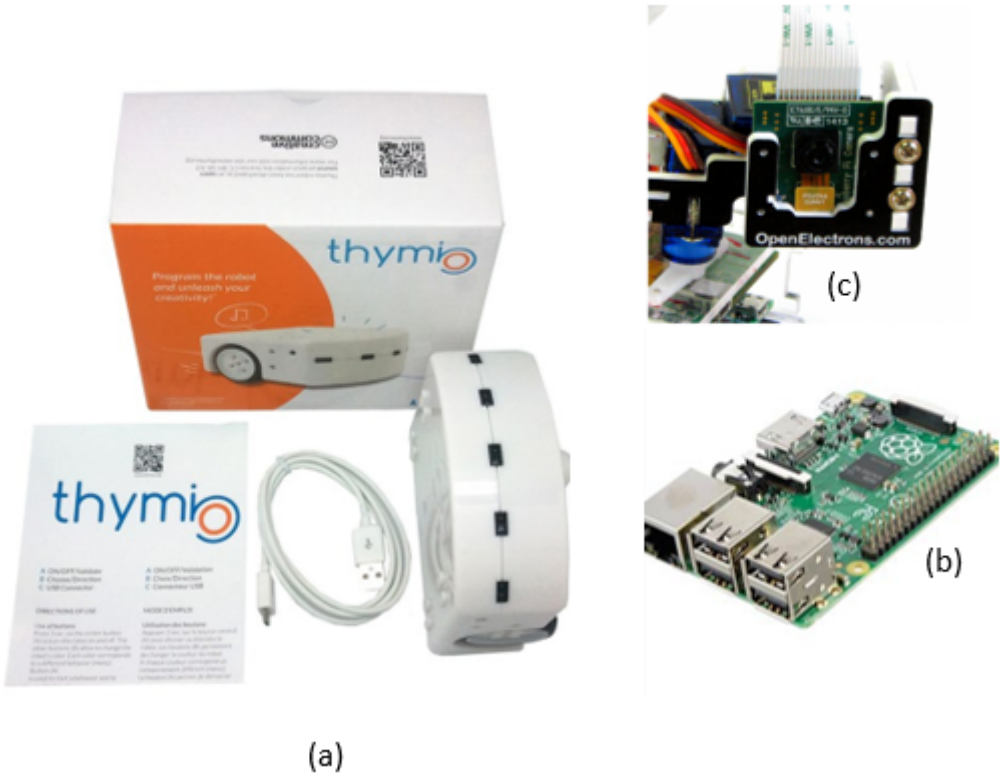

Figure 3.1: Hardware setup

(a) Thymio Robot: Thymio II is an affordable educational robot, costs around 100 euros. The robot was developed at the EPFL in collaboration with ECAL, both in Lausanne, Switzer-

land. A purely-visual programming language was developed at ETH Zurich . All components, both hardware and software, are open source It provides three main features: a large amount of sensors and actuators, a specific interactivity based on light and touch, aimed at increasing the understanding of the robot functionalities and a very efficient programming environment based on Aseba.

(b) Raspberry Pi 3 : The tiny desktop computer that will control all other hardware stated above equipped with 2018 raspbian-stretch Operating system, OpenCV 3.1 with python 3.5 alongside with blaar** library as a communication tcp library to transmit the camera feed and receive the thymio orders. [1]

(c) Pi-pan, a Pan-Tilt Kit for Raspberry Pi Camera: Pi-Pan provides Pan with 180 degrees (from left to right) and tilt with 110 degrees (top to bottom) movements for your Raspberry Pi Camera, can be easily controlled from Python program.

(d) Desktop Pc, a Desktop in ETIS Lab with the following specs; core i7, 2.4 Gz , with Ubuntu 20.04 installed.

(e) D-Link Router (DWR-921); Since the Raspberry Pi isn't authorized to connect through the university's network. we needed to create our own local network, the router is just powered on but not connected to any network in fact. The Pc is connected to the router through LAN cable, while the raspberry pi is connected through wifi. In the end the configuration is as follows in the figure below;

The assembled final robot will look like;

[1] **blaar libraries: These libraries are a set of basic tools simplifying programmation. The programmer keeps full control on the program. You have shortcuts for classical functionalities and very little abstractions. You can freely mix blibs and low level C/C++ functions.

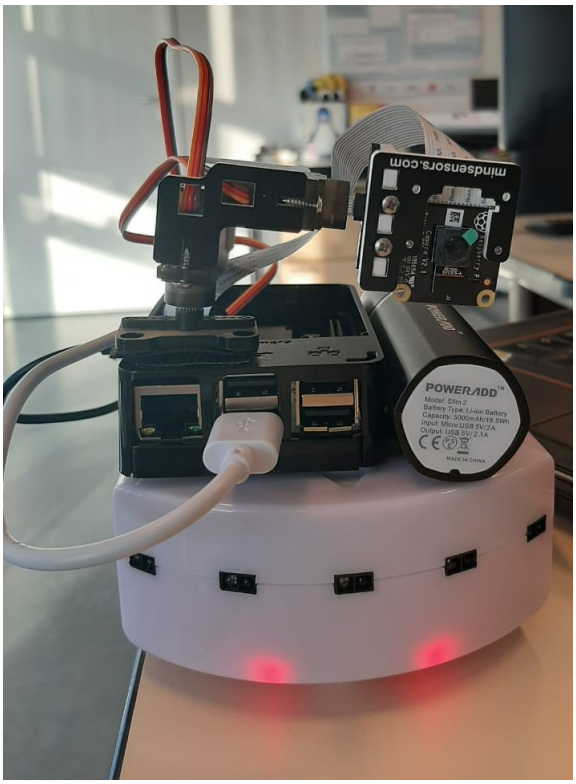

Figure 3.2: Assembled final version of the robot

The goal was to consifer the raspberry Pi as a transmitter and receiver, but not responsible for any calculations, processing or any decisions, so the communication between the Raspberry Pi and the Pc was a first using the LAN then we switched to wireless network through the tcp connection library. so the connection is as shown in the figure below;

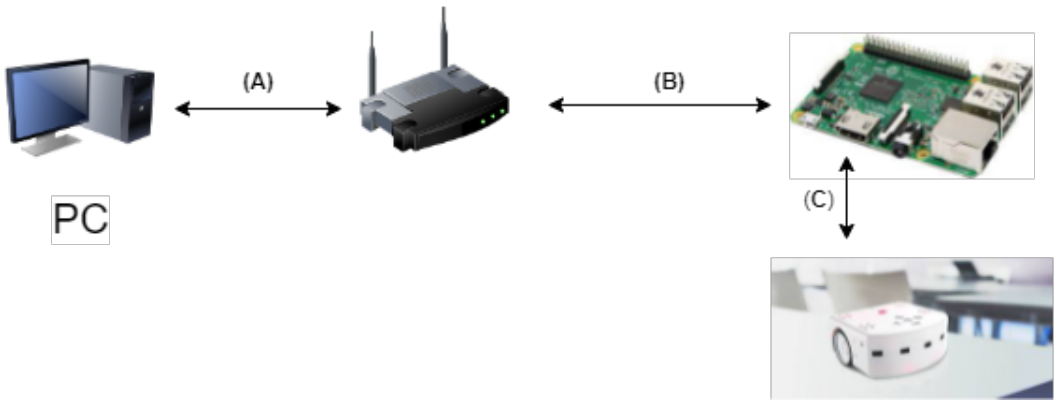

Figure 3.3: Connection Map

(A) connection A is between the router and the PC(with the Picamera) using LAN Cable.

(B) connection B is between the router and the Raspberry Pi (with the Picamera) using wifi.

(C) Connection C is between the Raspberry Pi (with the Picamera) and the Thymio Robot. The Raspberry Pi (RPI) is wired to the Thymio robot by cable, but the RPI controls the thymio using Python code [A] which uses the DBUS library to communicate with the thymio.

3.2 Guide for Hardware and software configuration

3.2.1 Thymio control using Python

- Make sure Aseba 1.5.5 (works on RPI 3 and both Python 2.7 or 3.5) is installed.

[2]

- To correctly install Aseba 1.5.5 from the official website or following the commands below will result the same;

```
mkdir aseba_install
cd aseba_install
# getting dashel
wget http://wiki.thymio.org/local--files/
en:linuxinstall/libdashel_1.3.0_armhf.deb
# getting enki
wget http://wiki.thymio.org/local--files/
en:linuxinstall/libenki_2.0~git.20161121_armhf.deb
# getting aseba
wget http://wiki.thymio.org/local--files/
en:linuxinstall/aseba_1.5.5_armhf.deb
# installing dashel
sudo dpkg -i libdashel_1.3.0_armhf.deb
# installing enki
sudo dpkg -i libenki_2.0~git.20161121_armhf.deb
# installing aseba
sudo dpkg -i aseba_1.5.5_armhf.deb
# deleting deb files and temp folder
cd ..
sudo rm -rf aseba_install
```

- After installing them some dependencies will be broken and you will have to fix them with the command. (apt --fix-broken install)

- If some dependencies are still missing use this line to install all noted dependencies of aseba according to their github repo:

```
sudo apt-get install libqt4-dev libqtwebkit-dev
qt4-dev-tools libqwt5-qt4-dev libudev-dev
libxml2-dev libsdl2-dev libavahi-compat-libdnssd-dev
cmake g++ git make
```

[2]**The software package aseba is a bundle including all necessary software to control a Thymio2 robot from a chosen controller system.

- Then install sql:

```
sudo apt install libqt4-sql-sqlite
```

- Since our final goal is to use the previously installed aseba package from python3. To do so, we need to install a few dependencies to use pycairo and **PyGObject** both necessary to interface python3 with asebamedulla. [3]

- If python3.7 is not the default python3 version (i.e. you have additional python3.x version as a default) you will need to do the following so that PyGObject can build properly.

Listing 3.1: Install Python3.7

```
sudo apt-get install python3.7-dev
```

- Install the dependencies of pycairo and PyGObject.

Listing 3.2: PyGObject and Pycairo installation

```
sudo apt install libgirepository1.0-dev gcc libcairo2-dev
sudo apt install pkg-config python3-dev gir1.2-gtk-3.0
```

- To meet the requirements of dbus-python you might also need to install:

Listing 3.3: Install dbus library

```
sudo apt install libdbus-1-3 libdbus-1-dev
```

- Now, you need to ensure correct Thymio-II Driver. Thymio firmwares (on the actual robot) use a so called "aseba protocol version". If aseba (controller side, Pi3) and Thymio firmware (slave side, Thymio-II) has different "aseba protocol versions", they will not work together anymore. Try to use the "Thymio Firmware Upgrader" but it probably won't work and it will be better to install the firmware directly from the website of Thymio from (Thymio2-V14.hex) and download it manually from the Thymio firmware upgrader as in the following official guide. Link

- To make sure you installed everything correctly; try the following command on your RPI terminal, and the output should saying that Thymio-II device was found.

[3] **PyGObject** is a Python package which provides bindings for GObject based libraries such as GTK, GStreamer, and many more. It supports Linux, Windows and macOS and works with Python 3.7+ and PyPy3. PyGObject uses glib, gobject, girepository, libffi and other libraries to access the C library (libgtk-3.so) in combination with the additional metadata from the accompanying typelib file (Gtk-3.0.typelib) and dynamically provides a Python interface based on that information.

Listing 3.4: Asebamedulla service start

```
asebamedulla "ser:device="
```

- At the end of these steps you should have; A thymio-II robot with firmware v12 A Raspberry Pi 3 with Stretch OS with Aseba v1.5.5

3.2.2 Install and Run ORBSLAM on PC

In order to run our code for ORBSLAM, which receives the camera feed from the RPI and sends commands to RPI to send them to Thymio-II you need to follow the following steps;

(1) Install Dependencies :

Listing 3.5: Install Dependences

```
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev
sudo apt-get install pkg-config libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install python-dev python-numpy libtbb2
sudo apt-get install libtbb-dev libjpeg-dev libpng-dev libtiff-dev
sudo apt-get install libjasper-dev libdc1394-22-dev
```

(2) Install OPenCV3;[4]

Listing 3.6: Install OPENCV

```
git clone https://github.com/opencv/opencv.git
git chekcout 3.3.1
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local ..
make -j3
make install
```

(3) Install Eigen3;[5]

Listing 3.7: Install Eigen3

```
sudo apt install libeigen3-dev
```

[4]OpenCV (Open Source Computer Vision Library: is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. Its core functionality that it's a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

[5]Eigen is a high-level C++ library of template headers for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms. Eigen is open-source software licensed under the Mozilla Public License 2.0 since version 3.1.1.

(4) Install Pangolin;[6]

Listing 3.8: Install Pangolin

```
git clone https://github.com/stevenlovegrove/Pangolin.git
cd Pangolin
mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
make -j 3
make install
```

(5) Now, go to the ORBSLAM3 directory and build the project (./build.sh).

(6) To run the ORBSLAM library on your PC you need to run the command below after running the code on the RPI.

Listing 3.9: Run ORBSLAM3

```
./Examples/Monocular/mono_euroc ./Vocabulary/ORBvoc.txt
./Examples/Monocular/EuRoC.yaml ./MH01
./Examples/Monocular/EuRoC_TimeStamps/MH01.txt dataset-MH01_mono
```

(6) Please note that the files (Optimizer.cc , monoeuroc.cc) have been edited to achieve the purpose of the project. They are attached with the digital version of the thesis.

3.2.3 Communication protocol between RPI and PC.

Attempts on ROS have almost took a month. It was too complicated that both ROS versions (on RPI and PC) need to be the same, while the OS on RPI is too old to install ROS neotic, on the other side, Ubuntu 20.04 no longer supports installation of ROS melodic (on RPI). Eventually, the communication through TCP connection was implemented instead of ROS.

Using the TCP connection library on both sides as stated before in the Hardware configuration; Now the router is giving IP address to both the PC and the RPI. In order to know what is the IP address of the RPI, if you didn't set it to be static, you can open a web page on your PC (wired connection to the router) and write 192.168.0.1, the password and the IP is usually provided according to your router brand and model. from the router interface login you can see that the RPI is connected through WiFi and for example has the IP address (192.168.0.51). Note that, you will need to know the IP address in order to edit your monoeuroc.cc code to add the IP address to the code.

- On the RPI side: you need to install and build the blaar library [16] as stated in its official website.

[6]Pangolin is a set of lightweight and portable utility libraries for prototyping 3D, numeric or video based programs and algorithms. It is used quite widely in the field of Computer Vision as a means to remove platform-specific boilerplate and make it easy to visualize data.

– Open two terminals of the RPI through ssh connections; The first one will include the line to open the aseba-medulla service as follows:

Listing 3.10: First terminal on RPI

```
asebamedulla "ser:device=/dev/ttyACM0"
```

– While the second terminal will run the python code (RPI_Code.py) using Python3.5, note that the blaar library isn't compatible with python 2.7.

Listing 3.11: Second terminal on RPI

```
cd /path to the code installed inside the blaar build folder
python3 RPI_code.py
```

- On the PC side; you also need to install the same library. open only one terminal and run the ORBSLAM3 as stated before in .

3.3 ORBSLAM / Z-Transform Algorithm

3.3.1 ORBSLAM3 Algorithm

As stated before in the Literature review, there are many SLAM algorithms, open source or closed source. We have chosen to work with ORBSLAM since it's quite new and can be integrated to our OS (Ubuntu 20.04). In this section, we will go through any specifications in the ORBSLAM code that's different that the general SLAM algorithms. Also, we will go through the different implementation methods and its compatibility to other libraries.

The main advantages of ORBSLAM3 over other SLAM algorithms are:

1. The initialization method proposed was previously presented. Here, we add its integration with ORB-SLAM visual–inertial, the extension to stereo-inertial SLAM, and a thorough evaluation in public data sets. The monocular and stereo visual–inertial systems are extremely robust and significantly more accurate than other visual–inertial approaches, even in sequences without loops [[3]].

2. Many recent visual SLAM and VO systems, solve place recognition using the DBoW2 bag of words library. DBoW2 requires temporal consistency, matching three consecutive key frames to the same area, before checking geometric consistency, boosting precision at the expense of recall. As a result, the system is too slow at closing loops and reusing previously mapped areas. ORBSLAM3 proposes a novel place recognition algorithm, in which candidate key frames are first checked for geometrical consistency, and then for local

consistency with three co-visible key frames, which in most occasions are already in the map. This strategy increases recall and densifies data association improving map accuracy, at the expense of a slightly higher computational cost.

3. ORB-SLAM3 Atlas. The first complete multi map SLAM system able to handle visual and visual–inertial systems in monocular and stereo configurations. The Atlas can represent a set of disconnected maps and apply to them all the mapping operations smoothly: place recognition, camera re-localization, loop closure, and accurate seamless map merging. This allows to automatically use and combine maps built at different times, performing incremental multi-session SLAM. A preliminary version of ORB-SLAM Atlas for visual sensors was presented in [[17]]. Here we add the new place recognition system, the visual–inertial multi-map system, and its evaluation on public data-sets.

The full edited code that we used to run the ORBSLAM library and control the Thymio is attached in Appendix A A

3.3.2 Z-transform Algorithm;

The main purpose of the z-transform algorithm in the current phase is to detect the shift in the images, then the Zoom and moreover to be able to save a point and find the same point again even with different brightness or background. In the following section, we will state some attempts to do that in different ways;

Let's first define the algorithm and solution steps;

- Assuming an image of intensity matrix I (m x n), The following algorithm will be applied only one one row of an image (take any row) as in the figure below;

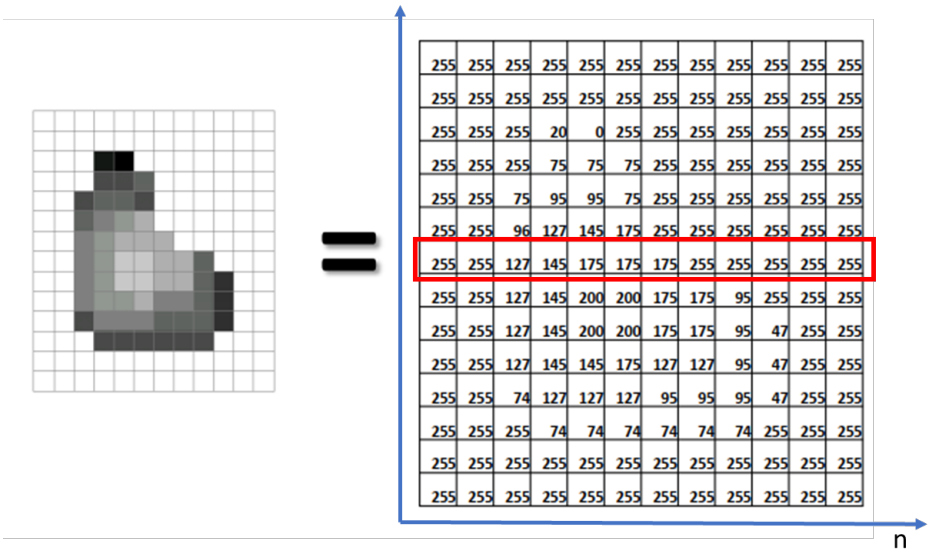

Figure 3.4: Image matrix and row exaction

- The row can be considered as x[n] , a signal , that we can apply the equation of the Z-transform on, for different Z-values.

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) * z^{-n} \tag{3.1}$$

- Each application of the theorem is expected to be as in the following graph.

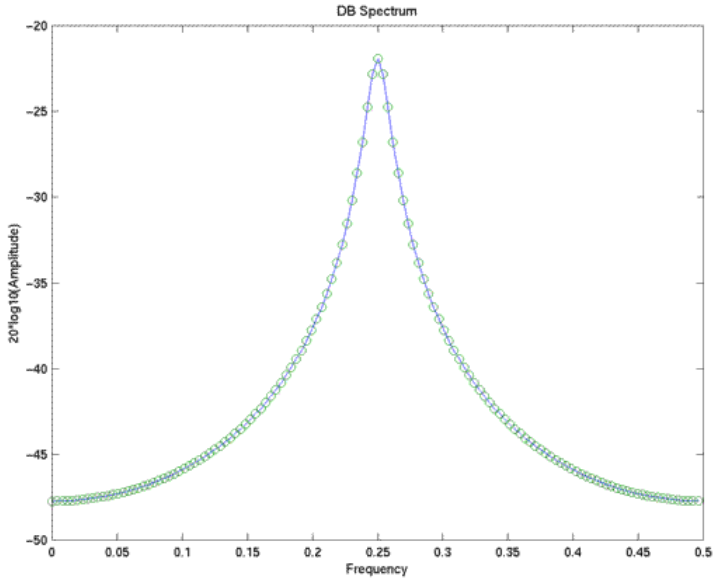

Figure 3.5: z-transform on x[n] signal

- We are using the Unilateral Z-transform as follows; Assuming that the zero position is by mid-width of the image.

$$X(z) = \sum_{n=0}^{\infty} x(n) * z^{-n} \tag{3.2}$$

- Where the shift property for the right shift by n will be

$$f(k-n) = Z^{-n} * F(z) \tag{3.3}$$

- The left shift by n

$$f(k+n) = Z^{n} * [F(z) - \sum_{k=0}^{n-1} f(k) * z^{-k}] \tag{3.4}$$

The problem that would have existed before using the unilateral z-transform is the left shift by n while substituting $f(-n)$ and we don't have any information about it. Now we can work on half the image only to be able to have a value for $f(-n)$.

To better illustrate the algorithm steps; The following flowchart states that for each consequent frames, we will perform greyscale filter on them, then apply the z-transform equation on one row

of each of them which will give us the signature for each frame, picture, feature, or even object that the robot is required to memorize. After that, mathematical operations are implemented to detected the shift, brightness change, zooming in or out,.. etc.

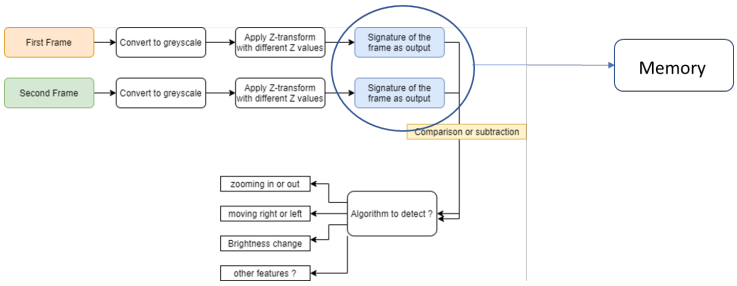

Figure 3.6: Algorithm Flow chart

Chapter 4

Results and Performance Evaluation

In the previous chapter we provided the theory for our both Z-transform and ORBSLAM3 algorithms . Furthermore, we showed that this shift property in z-transform is mathematically correct but needs the proper implementation. We also talked a little about the modifications added to the ORBSLA3 code that has enabled us to install it on any platform and control a mobile robot with it. Moreover, we will speak about our remarks and conclusions about the ORBSLAM3 code and the z-transform.

4.1 Performance results of ORB SLAM

As stated by many system Evaluators and it was the same result that we concluded as well that, in monocular-inertial configuration, ORB-SLAM3 is five to ten times more accurate than the best systems available in the literature. It summarizes the running time of the main operations performed in the tracking and mapping threads, showing that ORBSLAM3 system is able to run in real time at 30–40 frames and at 3–6 key frames per second. The inertial part takes negligible time during tracking and, in fact, can render the system more efficient as the frame rate could be safely reduced.

As the tracking and mapping threads work always in the active map, multi-mapping does not introduce significant overhead. Also, the running time of the main steps for loop closing and map merging. The novel place recognition method only takes 10 ms per key frame [[3]]. Times for merging and loop closing remain below 1 s, running only a PG optimization. For loop closing, performing a full BA may increase times up to a few seconds, depending on the size of the involved maps.

- **Calibration of monocular camera**

Camera calibration is the process of computing a camera's internal properties such as focal length, lens distortion coefficients, etc. These properties are intrinsic to the camera, which once estimated, do not change. However, different calibration software (or different trials) might produce slightly different results; these slight variations are due to noise in the imaging system.

In computer/robotics vision, it is always advisable to work with calibrated cameras as it is a one-shot process and it makes things much easier there on-wards.

Since, we know that a single camera can't have access to the depth information. As a sequence, A monocular camera is needed to be calibrated before usage in almost any visual navigation algorithm. But, that's another positive point of ORBLAM3 that its coordinate system is that of the first frame camera pose in first succeeded triangulation. There's no way monocular SLAM can know the real world scale, so it uses an arbitrary one. but if you didn't calibrate the camera and ran the program, your mapping and saved Key-frames will be correct of course with an arbitrary unit. As for example in 4.1 The tracking and mapping part was correct although the camera was never calibrated.

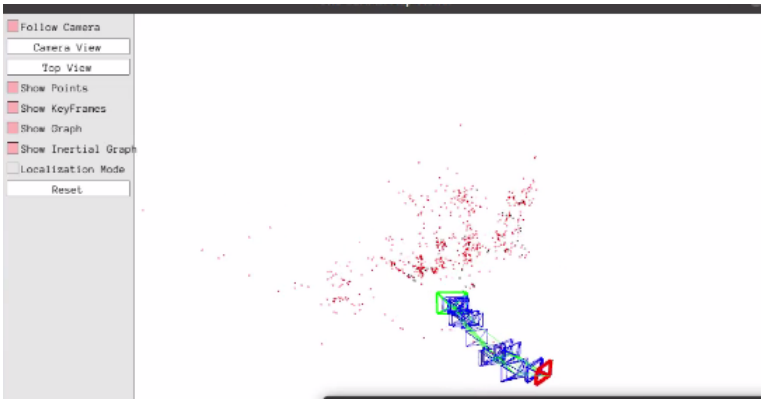

Figure 4.1: Tracking without calibration

- **Shift detection:**

Shift detection is the most basic feature that a visual navigation algorithm has to do. The process is quite easy in ORBSLAM, as it's the main output of the monocular system. the matrix Tcw is the world pose in camera reference, 4x4 matrix.

Tcw in that order because we usually read the transform from right to left, which will be easier to understand for a long position. For example when we say $p4 = T43 * T32 * T21 * p1$ it means to transform the point from frame 1 to frame 4.

$$\text{While Tcw (the transformation matrix) } = \begin{matrix} S_x * R_{00} & R_{01} & R_{02} & \triangle x \\ R_{10} & S_y * R_{11} & R_{12} & \triangle y \\ R_{20} & R_{21} & S_z * R_{22} & \triangle z \\ 0 & 0 & 0 & 1 \end{matrix}$$

The shifting in x, y, z can be easily detected in a very precise accuracy.

- **Limitations of the ORBSLAM library;**

  Although it's a robust and accurate library which has shown great results over other visual navigation alternatives. However, It has some limitations:

  1. Brightness and noise background; the abrupt change in brightness can be confusing for the Orb and make it lose it's location in the map until the robot moves again and he starts to create a loop again.
  2. Facing a feature-less area can cause the program to crash if it's not optimized, and out of the box it's not optimized, you have to optimize the code by your self in order to enable the orb to continue looping over the system even if there's no features in the area.
  3. ORB is more for cities and places where it can detect features. For example, the case of robots in mines between the mountains. ORB isn't the right system to use then.

4.2 Performance results of Z-Transform Algorithm

The z-transform code is no longer complete, but the results accomplished is quite good and worth mentioning. So, We will mention some algorithms that worked in a faulty way and some algorithms that are working correctly but still not robust enough.

- **Using shifting property:**

The algorithm will be as follows and the original code is in the Appendix (A)

Algorithm 4.1 Detect shift using z-transform

$XL_1(z)$
$XR(z)$
$XL(z) = Z^n * [F(z) - \sum_{k=0}^{n-1} f(k) * z^{-k}]$
for $n0 = 1 : 100$ **do**
  $XL(z) == XL_1(z)??Findn0$
end for

Two Images artificially shifted as follows, and the rows used represented as in the second figure;

Figure 4.2: Two images artificially shifted

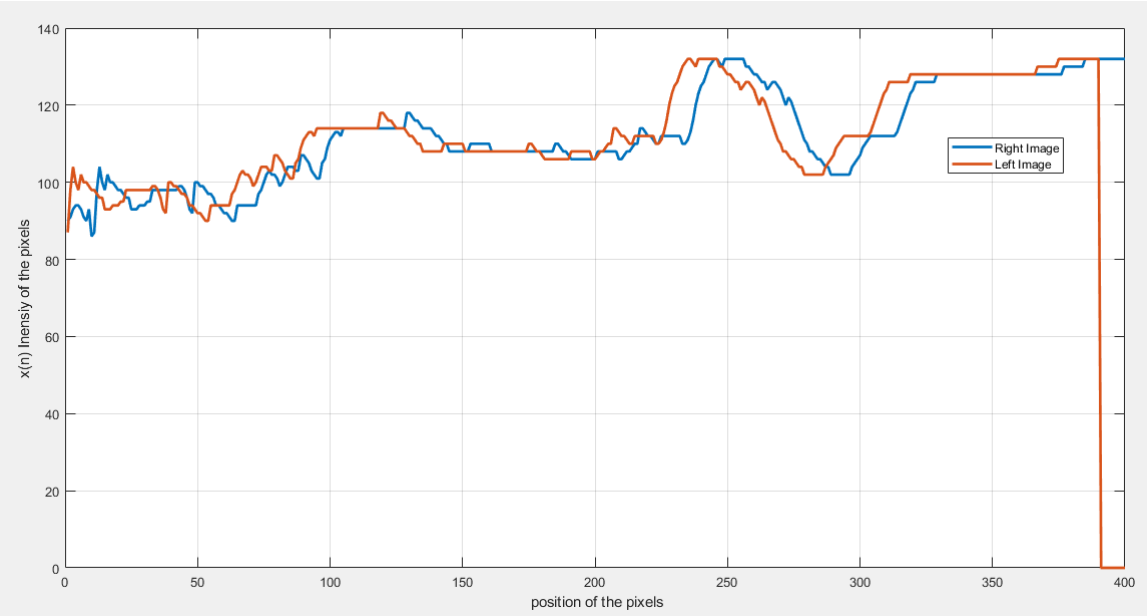

Figure 4.3: Representation of the row to be used as $x(n)$

Choosing z= 0.9; The result was as follows; from which we can easily understand that the solution doesn't converge and changing the parameters won't be the solution. but changing the algorithm itself might have more promising results.

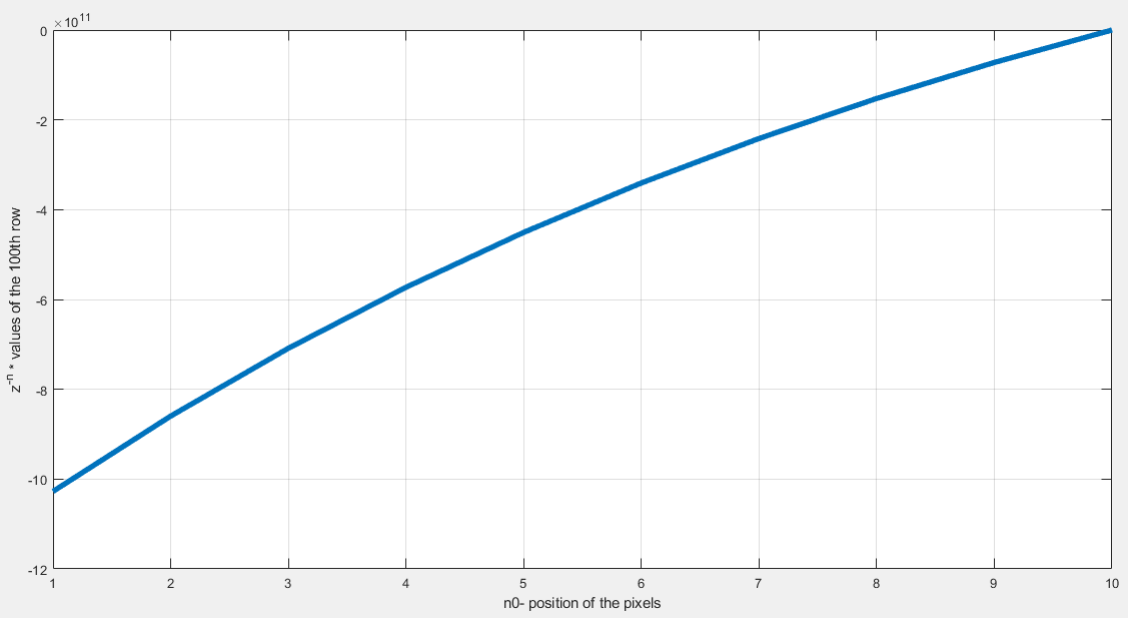

Figure 4.4: Result for equation solving technique

- The algorithm to be viewed now doesn't depend on the shifting property but depends on the z-transform as a specific value for each region by itself; The code itself is attached in the Appendix (code 4)

Algorithm 4.2 Algorithm for z-transform

$divide the image into equal portions (s = sampling ratio) e.g. 4$
for $z = [.7, 0.8, 0.9]$ **do**
  $Calculate X_1(z), X_2(z), X_3(z), X_4(z) for the first frame$
  $Calculate X_5(z), X_6(z), X_7(z), X_8(z) for the second frame$
end for
$shift_1 == X_1(z) - X_5(z)$
$shift_2 == X_1(z) - X_5(z)$
$....for all sample regions$
$The maximum change in z values is the region of the shift$

Two Images artificially shifted as follows, and the rows used represented as in the second figure;

Figure 4.5: Two images artificially shifted

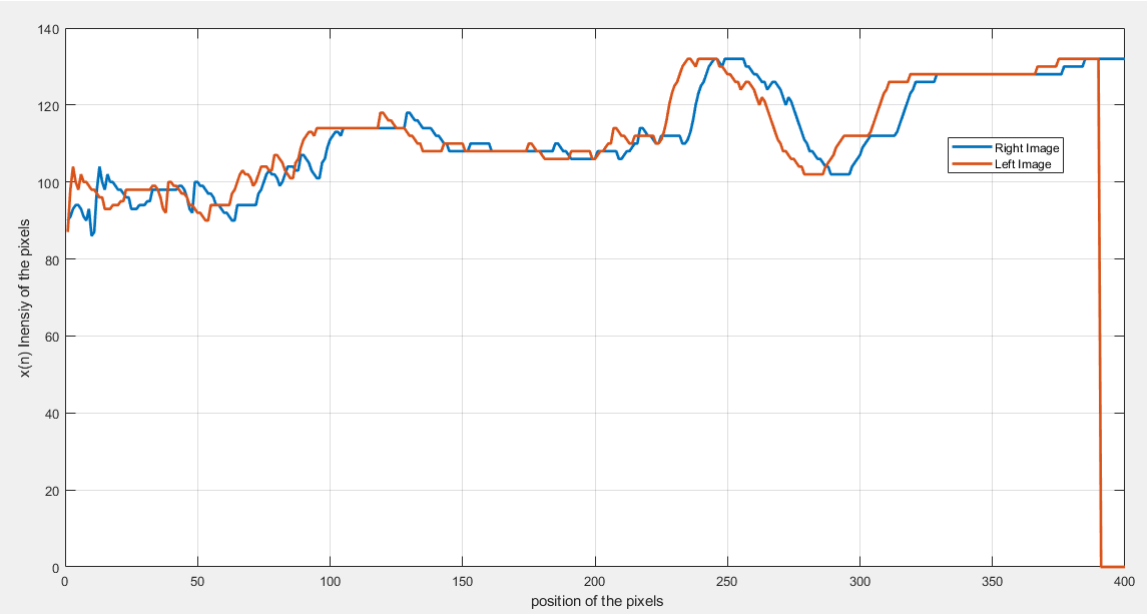

Figure 4.6: Representation of the row to be used as $x(n)$

Choosing z= 0.9; The result was as follows; we can notice that the max. difference in the signature is in the first region which is correct because we shifted the image -20 (negative means to the left) pixel.

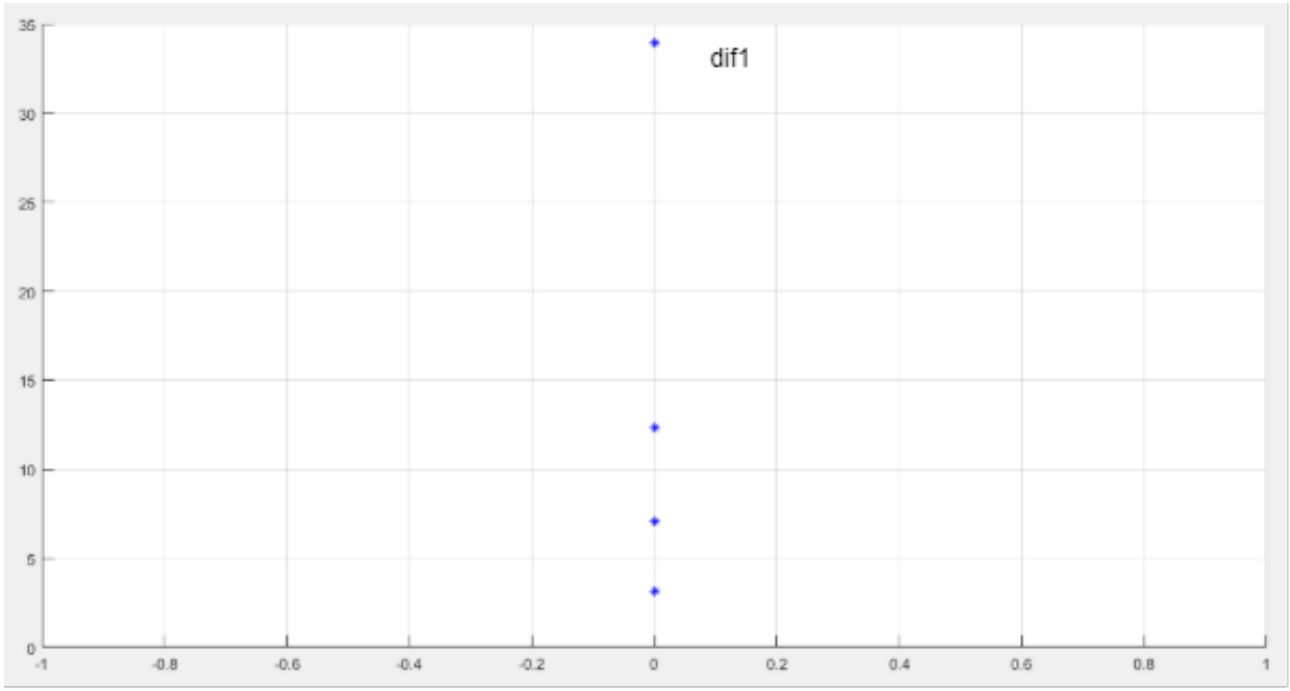

Figure 4.7: Maximum difference in z-values of each region

Chapter 5

Conclusion and Future Work

In this chapter, I will sum-up my work progress during my master thesis and the future work to develop a **low cost visual navigation system**. Our current state till now we have reached to a quite good working algorithm with the z-transform and a perfect understanding for the advantages, disadvantages and complications in hardware configuration of implementing ORBSLAM library which consumed the most of the internship time because it's never well documented and that's why we wanted to document all the process step by step.

Till this step, we can successfully run Orb code on Ubuntu remote controlling RPI as a a mini-discovery robot connected to a camera. Furthermore, We worked on the implementation of the z-transform code which has shown quite good results and positive gestures, so we are willing to continue working on it after.

Finally, It was surprising that there is no papers reviewing the ORBSLAM code mentioning any of the limitations we found regarding the brightness, or the feature-less places. Although, it's an interesting and essential applications for mobile robots that work in dessert and most of the times between mountains, so No GPS signal can help the localization and Mapping. Although ORBSLAM showed good accuracy and quality in both indoor and outdoor navigation. Still, this specific application can't use the ORBSLAM library.

A comparison between ORBSLAM and Z-transform algorithm won't be fair or accurate. The orb algorithm is a powerful library that one can't deny that. Also, our z-transform code isn't much interested about the mapping part. we are more interested in overcoming the very high computational cost of the ORB system, Also the complexity of its implementation, Also the brightness failure in the code and finally the feature-less areas that can break the whole code. Our z-transform didn't reach a stable point to be in front of the ORB in comparison. but it's good to know that we are on the right way.

For the future work we need to work on the zooming algorithm of the z-transform code. Also, the recursive method of the algorithm [[16]] has been implemented by Dr.Blanchard and it's accuracy is very good and how to overcome the brightness problems.

Appendix A

Appendix A: Codes

- Code Running on RPI to control Thymio and send camera feed to PC

Listing A.1: Main code on RPI

```
1  import dbus
2  import dbus.mainloop.glib
3  from gi.repository import GLib
4  from gi.repository import GObject
5  import time
6  from optparse import OptionParser
7  import sys
8  import cv2
9  from blc_network import BlcArrayTCP4Server
10 import numpy
11 proxSensorsVal=[0,0,0,0,0]
12
13 def dbusReply():
14     pass
15
16 def dbusError(e):
17     print ('error %s')
18     print (str(e))
19
20 #the communication part
21 ## new communication part ################## IMAGE ####################
22 vid=cv2.VideoCapture(0)
23 #vid.set(cv2.CAP_PROP_FPS, 5)
24 height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))
25 width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))
26 deep=1 # 1 if it is in gray
27 print(height,"x", width)
28 server_image=BlcArrayTCP4Server("31440", 'UIN8', 'RGB3', [height, width])
29 print("server_camera connected")
30 server_commands=BlcArrayTCP4Server("31441", 'UIN8', 'TEXT', [2])
31 print("server_commands connected")
32 server_image.wait_connection()
33 server_commands.wait_connection()
34
35
36 #server_image.wait_connection()
37    #We create an numpy array (equivalent to cv:Mat in C++ ) associated with
       the data of server
38 server_array = numpy.ndarray([height, width], dtype=numpy.uint8, buffer=
       server_image.data, offset=0, strides=None, order='C')
```

```
39 ######
40
41 def Braitenberg():
42         t = time.time()
43     ## new communication part ##############################
44         ret, img = vid.read()
45         gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
46     #we copy the pixels of the image to the array of the server (i.e. the
           buffer: server.data)
47         numpy.copyto(server_array, gray)
48         server_image.send_data()
49         print("Image sent")
50 ######################################################################
51
52     #get the values of the sensors
53         network.GetVariable("thymio-II", "prox.horizontal",reply_handler=
               get_variables_reply,error_handler=get_variables_error)
54
55         server_commands.recv_data()
56         print("command received")
57         val_left=server_commands.data[0]
58         val_right=server_commands.data[1]
59
60     #send motor value to the robot
61         network.SetVariable("thymio-II", "motor.left.target", [val_left])
62         network.SetVariable("thymio-II", "motor.right.target", [val_right])
63
64       # network.SetVariable("thymio-II","motor.left.target",[0])
65         ##time.sleep(0.01)
66         if (proxSensorsVal[2]>1000) or (proxSensorsVal[4]>1000) or (
               proxSensorsVal[3]>2000):
67             network.SetVariable("thymio-II", "motor.left.target", [0])
68             network.SetVariable("thymio-II", "motor.right.target", [0])
69
70
71         Braitentime=time.time() - t
72         print("run time=",Braitentime)
73         return True
74
75 def get_variables_reply(r):
76     global proxSensorsVal
77     proxSensorsVal=r
78
79 def get_variables_error(e):
80     print ('error:')
81     print (str(e))
82     loop.quit()
83
84 if __name__ == '__main__':
85     parser = OptionParser()
86     parser.add_option("-s", "--system", action="store_true", dest="system",
           default=False,help="use the system bus instead of the session bus")
87
88     (options, args) = parser.parse_args()
89
90     dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
91
92     if options.system:
93         bus = dbus.SystemBus()
```

```
94      else:
95          bus = dbus.SessionBus()
96
97      #Create Aseba network
98      network = dbus.Interface(bus.get_object('ch.epfl.mobots.Aseba', '/'),
            dbus_interface='ch.epfl.mobots.AsebaNetwork')
99      #network.SetVariable("thymio-II", "motor.left.target", [0])
100     #network.SetVariable("thymio-II", "motor.right.target", [0])
101
102     #print in the terminal the name of each Aseba NOde
103     print (network.GetNodesList())
104     #GObject loop
105     #print 'starting loop'
106     #gobject.threads_init()
107     mainloop = GObject.MainLoop()
108     #call the callback of Braitenberg algorithm
109     handle = GObject.timeout_add (10, Braitenberg) #every 0.1 sec
110     mainloop.run()
```

- Code Running on PC to receive the camera feed, process the information and send commands to RPI

Listing A.2: Main code on PC-monoeuroc

```
1
2  /**
3  * This file is part of ORB-SLAM3
4  *
5  * Copyright (C) 2017-2020 Carlos Campos, Richard Elvira, Juan J.  , M.M.
       Montiel and Juan D. , University of Zaragoza.
6  * Copyright (C) 2014-2016  Mur-Artal, M.M. Montiel and Juan D. , University
       of Zaragoza.
7  *
8  * ORB-SLAM3 is free software: you can redistribute it and/or modify it under
       the terms of the GNU General Public
9  * License as published by the Free Software Foundation, either version 3 of
       the License, or
10 * (at your option) any later version.
11 *
12 * ORB-SLAM3 is distributed in the hope that it will be useful, but WITHOUT
       ANY WARRANTY; without even
13 * the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE
       . See the
14 * GNU General Public License for more details.
15 *
16 * You should have received a copy of the GNU General Public License along
       with ORB-SLAM3.
17 * If not, see <http://www.gnu.org/licenses/>.
18 */
19
20 #include <stdio.h>
21 #include <termios.h>
22 #include <unistd.h>
23 #include <fcntl.h>
24 #include<iostream>
25 #include<algorithm>
26 #include<fstream>
27 #include<chrono>
```

```
28 #include <ctime>
29 #include<opencv2/core/core.hpp>
30 #include<opencv2/opencv_modules.hpp>
31 #include<System.h>
32 #include <blc.h>
33 using namespace std;
34
35
36
37 int main(int argc, char **argv){
38     if(argc < 3){
39         cerr << endl << "Usage: ./mono_euroc path_to_vocabulary
              path_to_settings" << endl;
40         return 1;
41     }
42
43     bool bFileName= (((argc-3) % 2) == 1);
44     string file_name;
45     if (bFileName)
46     {
47         file_name = string(argv[argc-1]);
48         cout << "file name: " << file_name << endl;
49     }
50
51     //***** IMAGE CONNECTION *****//
52     blc_array_tcp4_client client_image("192.168.0.51", "31440");
53     fprintf(stderr, "Connected_image\nPress 'q' to quit\n");
54     //***** COMMANDS CONNECTION *****//
55      blc_array_tcp4_client client_commands("192.168.0.51", "31441");
56      fprintf(stderr, "ConnectedÂ°commands\nPress 'q' to quit\n");
57     // ****** END OF CONNECTION DEF *****//
58
59     // Create SLAM system. It initializes all system threads and gets ready
          to process frames.
60     ORB_SLAM3::System SLAM( argv[1], argv[2], ORB_SLAM3::System::MONOCULAR,
          true);
61     auto start = std::chrono::steady_clock::now();
62
63     const int FRAME_HEIGHT = 480;
64     const int FRAME_WIDTH = 640;
65     cv::Mat im = cv::Mat::zeros(FRAME_HEIGHT, FRAME_WIDTH, CV_8UC1);
66
67     client_image.data = im.data;
68     client_image.size = FRAME_HEIGHT*FRAME_WIDTH;
69
70
71      double tframe;
72     // Main loop
73
74 while(1){
75         //add your code here
76 int i=0;
77 //cv::Mat Tcw = SLAM.TrackMonocular(im,tframe);
78
79         for(i=0;i<5;i++){
80         tframe = (chrono::steady_clock::now() - start).count()/1e9;
81         cout << tframe << endl;
82
83               // Pass the image to the SLAM system
```

```
84
85                client_image.recv_data();
86                SLAM.TrackMonocular(im,tframe);
87              //   cout<<Tcw.at<double>(1,4);
88              //   cout<<Tcw<<endl;
89
90              client_commands.chars[0]=100;
91              client_commands.chars[1]=100;
92              client_commands.send_data();
93          }
94          for(i=5;i<15;i++){
95          tframe = (chrono::steady_clock::now() - start).count()/1e9;
96          cout << tframe << endl;
97
98                 // Pass the image to the SLAM system
99
100               client_image.recv_data();
101               SLAM.TrackMonocular(im,tframe);
102               cv::Mat Tcw = SLAM.TrackMonocular(im,tframe);
103             //   cout<<Tcw.at<double>(1,4);
104               cout<<Tcw<<endl;
105
106             client_commands.chars[0]=100;
107             client_commands.chars[1]=100;
108             client_commands.send_data();
109         }
110         for(i=15;i<20;i++){
111         tframe = (chrono::steady_clock::now() - start).count()/1e9;
112         cout << tframe << endl;
113
114                // Pass the image to the SLAM system
115
116               client_image.recv_data();
117               SLAM.TrackMonocular(im,tframe);
118               cv::Mat Tcw = SLAM.TrackMonocular(im,tframe);
119             //   cout<<Tcw.at<double>(1,4);
120               cout<<Tcw<<endl;
121
122             client_commands.chars[0]=2*16-100;
123             client_commands.chars[1]=100;
124             client_commands.send_data();
125         }
126
127         for(i=20;i<25;i++){
128         tframe = (chrono::steady_clock::now() - start).count()/1e9;
129         cout << tframe << endl;
130
131                // Pass the image to the SLAM system
132
133               client_image.recv_data();
134               SLAM.TrackMonocular(im,tframe);
135               cv::Mat Tcw = SLAM.TrackMonocular(im,tframe);
136             //   cout<<Tcw.at<double>(1,4);
137               cout<<Tcw<<endl;
138
139             client_commands.chars[0]=100;
140             client_commands.chars[1]=2*16-100;
141             client_commands.send_data();
142         }
```

```
143         for(i=25;i<35;i++){
144         tframe = (chrono::steady_clock::now() - start).count()/1e9;
145         cout << tframe << endl;
146 
147               // Pass the image to the SLAM system
148 
149                client_image.recv_data();
150               SLAM.TrackMonocular(im,tframe);
151                cv::Mat Tcw = SLAM.TrackMonocular(im,tframe);
152              //   cout<<Tcw.at<double>(1,4);
153                cout<<Tcw<<endl;
154 
155              client_commands.chars[0]=-100;
156              client_commands.chars[1]=-100;
157              client_commands.send_data();
158         }
159         for(i=35;i<40;i++){
160         tframe = (chrono::steady_clock::now() - start).count()/1e9;
161         cout << tframe << endl;
162 
163               // Pass the image to the SLAM system
164 
165                client_image.recv_data();
166               SLAM.TrackMonocular(im,tframe);
167              //   cout<<Tcw.at<double>(1,4);
168              //   cout<<Tcw<<endl;
169 
170              client_commands.chars[0]=-100;
171              client_commands.chars[1]=-100;
172              client_commands.send_data();
173         }
174 
175     }
176 
177     // Stop all threads
178    SLAM.Shutdown();
179 
180    //uncommment this to return everything back to zero
181 
182     // Save camera trajectory
183     if (bFileName)
184     {
185         const string kf_file =  "kf_" + string(argv[argc-1]) + ".txt";
186         const string f_file =  "f_" + string(argv[argc-1]) + ".txt";
187         SLAM.SaveTrajectoryEuRoC(f_file);
188         SLAM.SaveKeyFrameTrajectoryEuRoC(kf_file);
189     }
190     else
191     {
192         SLAM.SaveTrajectoryEuRoC("CameraTrajectory.txt");
193         SLAM.SaveKeyFrameTrajectoryEuRoC("KeyFrameTrajectory.txt");
194         printf("trajectory saved - el mfrod y3ni - ");
195     }
196 
197 
198     return 0;
199 }
```

- First Code for Z-Transform

Listing A.3: First Code for Z-Transform

```
clc;clear all; close all;
%read the two images
Img1=imread('normal_50.jpeg');
% Img2=imread('rightshifted_50.jpeg');

%Full images
I1=rgb2gray(Img1);
%I2=rgb2gray(Img2);
I2= imtranslate(I1,[-10, 0]);
RR=double(I1(200,:));
LL=double(I2(200,:));
% RR=1:1:100; %right image
% LL=5:1:104; %left image

w=length(RR);  %width
rowL=LL((w/2):w);  %row left
rowR=RR((w/2):w);  %row right

figure(1)
plot(1:w,RR,'Linewidth',2)
hold on
grid on
plot(1:w,LL,'Linewidth',2)
legend('RR','LL')
xlabel(' position of the pixels', 'LineWidth',28)
ylabel('x(n) Inensiy of the pixels', 'LineWidth',28)
legend('Right Image','Left Image')

%%
% z= complex(0.9,0.9);
z=0.9;
%% summation : z^-n * x[n]
% x(z)_1 of the left image ( half image )
for n=1:length(rowR);
mul_1(n)=z^(-(n-1))*rowR(n);
end
X_Z1=sum(mul_1);

%%
% x(z)_2 of the left image
for n=1:length(rowL);
mul_2(n)=z^(-(n-1))*rowL(n);
end
X_Z2=sum(mul_2);

%%assuming a shift of 5 pixels n0=[4,5,7,20] n0start--
for n0=1:10;
     m=n0:-1:1;
      rowexcess=LL(((w/2)-n0):((w/2)-1));
      excess(m)=rowexcess.*(z.^m);

    term = sum(excess);

    XZ2(n0)=(z.^n0)*(X_Z1 - term);

end
    error=X_Z2-XZ2;

```

```
59 %% plotting
60 figure(2)
61   plot((1:n0),error,'LineWidth',4)
62   grid on
63 %   ylim([-0.5e2 0.5e2])
64 %   xlim([0 10])
65 %   axis equal
66 %   scatter(n0,error)
67   hold on
68 %   plot(x_pos,y_pos,'r*')
69   xlabel('n0- position of the pixels', 'LineWidth',28)
70 ylabel('z^{-n} * values of the 100th row', 'LineWidth',28)
71 % legend('shift theory','normal')
72
73 figure(3)
74 subplot(1,2,1);
75 imshow(I1);
76 subplot(1,2,2);
77 imshow(I2);
```

- Second Attempt for Z-Transform

Listing A.4: Second Attempt for Z-Transform

```
1  clc;clear all; close all;
2  %read the two images
3  Img1=imread('normal_50.jpeg');
4  % Img2=imread('rightshifted_50.jpeg');
5
6  %Full images
7  I1=rgb2gray(Img1);
8  %I2=rgb2gray(Img2);
9  I2= imtranslate(I1,[-10, 0]);
10 RR=double(I1(200,:));
11 LL=double(I2(200,:));
12 % RR=1:1:100; %right image
13 % LL=5:1:104; %left image
14
15 w=length(RR);  %width
16 rowL=LL((w/2):w);  %row left
17 rowR=RR((w/2):w);  %row right
18
19 figure(1)
20 plot(1:w,RR,'Linewidth',2)
21 hold on
22 grid on
23 plot(1:w,LL,'Linewidth',2)
24 legend('RR','LL')
25 xlabel(' position of the pixels', 'LineWidth',28)
26 ylabel('x(n) Inensiy of the pixels', 'LineWidth',28)
27 legend('Right Image','Left Image')
28
29 %%
30 % z= complex(0.9,0.9);
31 z=0.9;
32 %% summation : z^-n * x[n]
33 % x(z)_1 of the left image ( half image )
34 for n=1:length(rowR);
```

```
35 mul_1(n)=z^(-(n-1))*rowR(n);
36 end
37 X_Z1=sum(mul_1);
38
39 %%
40 % x(z)_2 of the left image
41 for n=1:length(rowL);
42 mul_2(n)=z^(-(n-1))*rowL(n);
43 end
44 X_Z2=sum(mul_2);
45
46 %%assuming a shift of 5 pixels n0=[4,5,7,20] n0start--
47 for n0=1:10;
48     m=n0:-1:1;
49      rowexcess=LL(((w/2)-n0):((w/2)-1));
50      excess(m)=rowexcess.*(z.^m);
51
52    term = sum(excess);
53
54    XZ2(n0)=(z.^n0)*(X_Z1 - term);
55
56 end
57    error=X_Z2-XZ2;
58
59 %% plotting
60 figure(2)
61  plot((1:n0),error,'LineWidth',4)
62  grid on
63 %  ylim([-0.5e2 0.5e2])
64 %  xlim([0 10])
65 %  axis equal
66 %  scatter(n0,error)
67  hold on
68 %  plot(x_pos,y_pos,'r*')
69  xlabel('n0- position of the pixels', 'LineWidth',28)
70 ylabel('z^{-n} * values of the 100th row', 'LineWidth',28)
71 % legend('shift theory','normal')
72
73 figure(3)
74 subplot(1,2,1);
75 imshow(I1);
76 subplot(1,2,2);
77 imshow(I2);
```

Bibliography

[1] D. Nister, O. Naroditsky and J. Bergen, "Visual odometry," Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004., 2004, pp. I-I, doi: 10.1109/CVPR.2004

[2] Franceschini, N., Ruffier, F., & Serres, J. (2007). A Bio-Inspired Flying Robot Sheds Light on Insect Piloting Abilities. Current Biology, 17(4), 329–335. doi:10.1016/j.cub.2006.12.032

[3] ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, Juan D. Tardós

[4] A comprehensive study for robot navigation techniques

[5] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in IEEE Robotics & Automation Magazine, vol. 13, no. 2, pp. 99-110, June 2006, doi: 10.1109/MRA.2006.1638022.

[6] Younes, Georges & Asmar, Daniel & Shammas, Elie & Zelek, John. (2017). Keyframe-based monocular SLAM: Design, survey, and future directions. Robotics and Autonomous Systems. 98. 10.1016/j.robot.2017.09.010.

[7] R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, 2003

[8] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, W. Burgard, G2o: A general framework for graph optimization, in: Robotics and Automation (ICRA), IEEE International Conference on, IEEE, 2011, pp. 3607–3613.

[9] S. Santra, S. Mandal, K. Das, J. Bhattacharjee and A. Deyasi, "A Comparative Study of Z-Transform and Fourier Transform applied on Medical Images for Detection of Cancer Segments," 2019 3rd International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech), 2019, pp. 1-4, doi: 10.1109/IEMENTech48150.2019.8981005.

[10] Yousif, K., Bab-Hadiashar, A. & Hoseinnezhad, R. An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. Intell Ind Syst 1, 289–311 (2015).

[11] Howard, Marc & Luzardo, André & Tiganj, Zoran. (2018). Evidence accumulation in a Laplace domain decision space.

[12] https://fr.mathworks.com/discovery/image-transform.html

[13] Wavelet and JPEG based Image Compression: An Experimental Analysis February 2012 Conference: International Conference & Workshop on Emerging Trends in TechnologyAt: TCET, Mumbai, India.

[14] https://github.com/RainerKuemmerle/g2o [G2o documentation]

[15] http://wiki.thymio.org/en:linuxinstall#toc3

[16] https://git.cyu.fr/blaar/blaar.git

[17] Kim, Giseop & Kim, Ayoung. (2021). LT-mapper: A Modular Framework for LiDAR-based Lifelong Mapping.

[18] Z. Mohamed, V. Flavien and B. Pierre, "Mobile robotics for restoring degraded ecosystems," 2015 IEEE Global Humanitarian Technology Conference (GHTC), 2015, pp. 273-278, doi: 10.1109/GHTC.2015.7343984.