People’s Democratic Republic of Algeria
الجمهورية الجزائرية الديمقراطية الشعبية
Ministry of Higher Education and Scientific Research
وزارة التعليم العالي و البحث العلمي

University of mohamed el bachir el ibrahimi

جامعة محمد البشير الإبراهيمي

End of studies thesis

To obtain the master’s degree in Computer Science

Option : Computer networks and multimedia

Tridimensional Stereoscopic Reconstruction

Realized by :
Mr. BELOUCHE Younes

Under the supervision of :
Dr. Belhadj Foudil

Defended on July 14, In front of the jury composed of:

Dr. Attia Abdelouaheb :	B.B.A	- President
Dr. Benaouda Nadjib :	B.B.A	- Examiner
Dr. Senouci Oussama :	B.B.A	- Examiner

Academic year : 2020/2021

Dedication

“

First of all, praise and thanks to Allah Almighty for giving me all the patience, courage, will and motivation that allowed me to accomplish this work.

To mom and dad, you might have no idea what I am doing, what I am studying, but you gave me your confidence and you have been always proud of me, thank you.

To my uncles and my aunts, especially Samra and Nadia, I give them the credit for all of this, may god have mercy on them and forgive them and put them in peace.

To my friends.

Thank you.

”

Acknowledgments

I would like to express my deep gratitude to my supervisor: **Dr. BELHADJ Foudil**, for his patient guidance, enthusiastic encouragement and useful critiques of this work, I always took him as my guide as well as my mentor. Also, I would like to offer a special thanks to **Mr Francois Zard**, a Senior software engineer, for his kind guidance.

Abstract

Computer vision is a sub field of artificial intelligence (AI) that offers the computers and systems the ability to extract meaningful information from digital images and videos, this way computers can take decisions or perform actions based on that information. Computers now can observe and see as if they have eyes.

Stereoscopy, also called stereo imaging, creates the illusion of three-dimensional depth from given two-dimensional images. The human vision is a great example of stereoscopy, the eyes perform a visual information gathering, they send this last one to the brain for processing, the brain reconstructs the real 3D scene and concludes several information from it.

Three-dimensional reconstruction is the process of obtaining a 3D representation of an object or a scene starting from a collection of 2D images.

The goal of this work is to build a system that imitates the human visual system by taking images for a given object and provides a 3D representation for this object.

Keywords: Computer vision, Stereoscopy, 3D reconstruction.

Résumé

La vision par ordinateur est un sous-domaine de l'intelligence artificielle (IA) qui offre aux ordinateurs et aux systèmes la capacité d'extraire des informations significatives à partir d'images et de vidéos numériques, de sorte que les ordinateurs peuvent prendre des décisions ou exécuter des actions basées sur ces informations. Les ordinateurs peuvent maintenant observer et voir comme s'ils avaient des yeux.

La stéréoscopie, aussi appelée imagerie stéréo, crée l'illusion d'une profondeur tridimensionnelle à partir d'images bidimensionnelles données. La vision humaine est un excellent exemple de stéréoscopie, les yeux effectuent une collecte d'informations visuelles, ils les envoient au cerveau pour traitement, le cerveau reconstruit la scène 3D réelle et en conclut plusieurs informations.

La reconstruction tridimensionnelle consiste à obtenir une représentation 3D d'un objet ou d'une scène à partir d'une collection d'images 2D.

Le but de ce travail est de construire un système qui imite le système visuel humain en prenant des images pour un objet donné et fournit une représentation 3D pour cet objet.

Mots Clés: Vision par ordinateur, Stéréoscopie, Reconstruction tridimensionnelle.

ملخص

الرؤية الحاسوبية هي فرع من فروع الذكاء الإصطناعي التي تقدم للحواسيب و الأنظمة القدرة علي إستخراج معلومات ذات معنى من الصور الرقمية و مقاطع الفيديو، بهذه الطريقة تستطيع الحواسيب إتخاذ قرارات أو تنفيذ عمليات بناءا على هاته المعلومات. الآن تستطيع الحواسيب الرؤية كما لو عندها عيون.

التجسيم أو التصوير المجسم، يشكل العمق الثلاثي الأبعاد من صور ثنائية الأبعاد. بصر الإنسان هو مثال رائع على التجسيم، العيون تنفذ عملية جمع المعلومات البصرية ثم ترسلها إلى العقل للمعالجة، العقل يقوم ببناء المشهد الثلاثي الأبعاد الحقيقي و يستنتج عدة معلومات منه.

البناء الثلاثي الأبعاد هو عملية إيجاد تمثيل ثلاثي الأبعاد لشيء فيزيائي أو مشهد إنطلاقا من مجموعة من الصور ثنائية الأبعاد.

الهدف من هذا العمل هو بناء نظام يقلد النظام البصري للإنسان عن طريق أخذ صور لشيء فيزيائي و تقديم تمثيل ثلاثي الأبعاد لهذا الشيء.

الكلمات المفتاحية: الرؤية الحاسوبية، التجسيم، البناء الثلاثي الأبعاد.

Contents

List of Figures

List of Tables

List of acronyms

AI *Artificial Intelligence*

CV *Computer Vision*

NASA *National Aeronautics and Space Administration*

ML *Machine Learning*

NN *Neural Network*

CNN *Convolutional Neural Network*

RNN *Recurrent Neural Network*

BPTT *Backpropagation Through Time*

LSTM *Long Short-Term Memory*

GRU *Gated Recurrent Unit*

SFM	*Structure From Motion*
SURF	*Speeded-Up Robust Features*
SIFT	*Scale Invariant Feature Transform*
ORB	*Oriented FAST and Rotated BRIEF*
SGBM	*Semi Global Block Matching*
BM	*Block Matching*
ICP	*Iterative Closest Point*

Chapter 1

Introduction

1.1 Context

One of the main, prominent and interesting fields in the artificial intelligence (AI) particularly and computer science generally, is computer vision (CV) that defines how computers can get high-level understanding from images or videos.

Computer vision is the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done in order to achieve some particular goal [25]. The idea behind some of CV techniques is about simulating the human visual system and perform its tasks. Computer vision systems can be classified into two categories: 2D and 3D systems. 2D systems are about processing two-dimensional images that contain only X and Y as coordinates. In the other hand, 3D systems are used to process three dimensional images that have an extra coordinate called the depth Z which leads us to the stereoscopy. Stereoscopy can be defined as the production of illusion of depth in a flat image by the presentation of slight differences in two images taken by the two left and right eyes, this is what a stereoscopic system actually does.

The 3D reconstruction of a scene with the use of a stereoscopic system is a core topic in CV, most of these systems rely on the triangulation method which requires a complete knowledge of the cameras, like their relative positions and orientations as well as their internal parameters like the optical center, skew and the focal length. Moreover, in order to perform this triangulation process, one needs ways of solving the correspondence problem, i.e. finding the point in the second image that corresponds to a specific point in the first image, or vice versa [32]. Generally, 3D reconstruction based on passive triangulation methods requires point correspondences among various viewpoints [29]

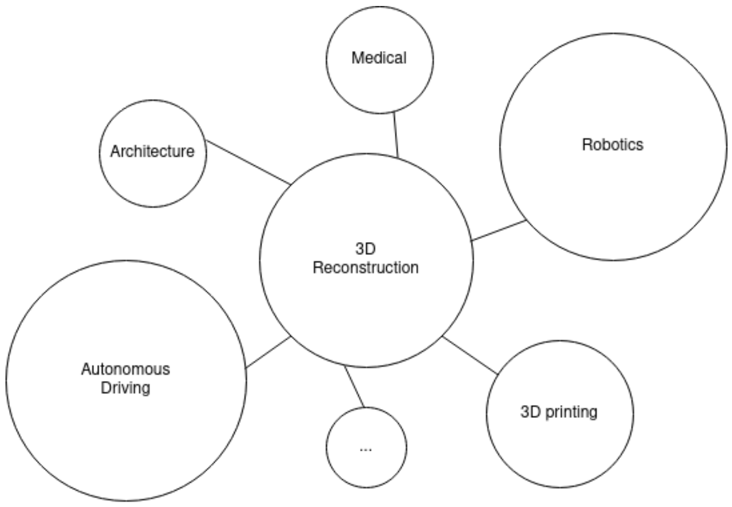


Figure 1.1: 3D reconstruction applications

Stereoscopic 3D reconstruction is the process that cares about the creation of the 3D shape of an object proceeding from stereo pair of images [18]. Stereo images are captured in the same way the human eyes capture scenes and can provide depth information, just as human eyes provide perception about depth [18]. The term depth perception refers to the ability of any entity, regardless of whether it is a biological organism or a man made system, to determine how far away the objects which surround it are, not necessarily in absolute terms or with great precision, but at least the capability to determine which of any two nearby objects is closer. If the resolution of the depth perception capability is sufficiently high, it suffices to allow perception of the shapes of surrounding objects [35].

The idea behind using more than one viewpoint is that triangulation can't be done from one single view, it needs two images taken from two viewpoints because when we project a 3D object in a an image, we are projecting it from a 3D space to 2D space, this is called the planar projection, and due to this kind of projection, we lose depth information and we can't recover it unless we have at least one other view so that we can triangulate a 3D point. We can make a little experiment by trying to close one of our eyes and then rapidly close the other one while opening the first one, we will see that objects that are close to you will appear to jump a significant distance while objects further away will move very little, that is the key point.

1.2 Problem statement

With the increased use of computer vision in lot of areas such as the industry and manufacturing area, the stereo-vision based systems in the 3D reconstruction context made a huge jump by enabling a significant reduction in production time and cost with less efforts. 3D object model can be reconstructed automatically using active and passive methods, structured light and laser range scanning belong to active methods, these methods demand special and expensive equipment unlike the passive methods which require cheep equipment comparing to the active ones. An example of passive methods which we are going to use is image-based methods, they depend on stereo image pairs, a 3D model will be reconstructed for each of these pairs using stereo matching algorithms. Creating 3D models with the help of some computer software and produce it to be a real object is not that hard because most of the important tasks will be managed and handled by the modeler and bringing the object to life is not a big deal after we make an accurate design and ship it toward production, but, the reverse process is not that easy, extracting the 3D shape from a real object and turning it to a 3D model can be quite difficult and challenging due to some problems like the correspondence problem which we will discuss later.

1.3 Goals

This project aims to build an efficient binocular stereo vision-based 3D reconstruction system to imitate the human visual system by digitally capturing the shape of a given physical object. The system will generate a 3D model starting from pairs of stereo images. The images will be taken simultaneously, each pair of them will be processed in order to extract depth information to be converted to a point cloud in the end. Each produced point cloud will be merged to one dense point cloud which presents our final object 3D model.

1.4 Thesis outline

This thesis is structured as follows:

The first chapter is an **introduction** which describes the main problematic; starting with a brief section that presents the context in order to get an overview about the project and provide an introduction to the **problem statement** section that explains the problematic we encounter. After getting an idea about the problem we face, the **goal** section gives an overview about the solution of our main problematic by defining the point from this work. The second chapter "**State of the art**" starts with the statement of the problem being resolved and discusses the state-of-the-art methods as well as the traditional methods in the context of 3D reconstruction. The third chapter "**Architecture and design**" is where we described our system in depth by exploring its architecture through multiple stages. The last chapter "**Implementation and results**" is about declaring which tools, programming languages and libraries that we have used in order to build this project, also, there is the evaluation section in which we discussed and criticized the results that we have reached.

Finally, we concluded this work in the **general conclusion** by presenting our prospects and future works.

Chapter 2

State of the art

2.1 Problem statement

The capacity to see with both eyes in comparable but somewhat different ways is referred to as stereoscopic vision. It enables humans to evaluate distance, allowing them to acquire real depth perception. The capacity of humans to see the world through stereoscopic vision has historically provided them a considerable edge over other beings and animals in the wild that do not possess this capability. This capability has given a new insights on the 3D reconstruction field.

Tridimensional stereoscopic reconstruction is about estimating the 3D geometry and structure of objects and scenes from a stereo view, which means a pair of image. This approach is used widely in many applications such as scene understanding, 3D modeling and industrial control. Recovering the lost dimension from 2D images made a challenge for the human being.

Recovering the lost dimension is known as depth perception, most of stereoscopic systems use the triangulation method to do the perception process, in order the get a 3D point **X** of an object or a scene, we triangulate it proceeding from two points **C1** and **C2** in which we will obtain two direction vectors **L1** and **L2**, the rays from the two points **C1** and **C2** through the point **X** as shown in the image below.

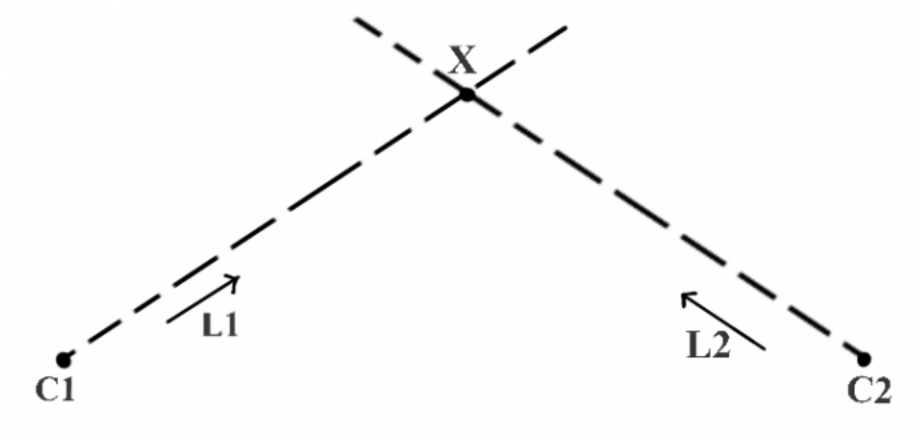


Figure 2.1: Locating a 3D point (X), at an unknown depth, with two known 3D points (C1 and C2) and direction vectors (L1 and L2) – Triangulation [27].

Triangulation

Triangulation depends on the **Epipolar Geometry** that led to solve the correspondence problem, basically, this problem appears when we try to find the corresponding pixel in the right image starting from some pixel in the left image, this search operation can be computationally very expensive and can lead to false positives.

Epipolar Geometry

When two cameras observe a scene, there are a variety of geometric linkages between the 3D points and their projections onto the 2D images, this is explained by the **Epipolar**

Geometry that leads to solve the correspondence problem so that we can perform the triangulation process, we will discuss all of this in more detail in the next chapter.

2.2 Existing approaches

2.2.1 Deep learning-based approaches

Since 2015, image-based 3D reconstruction using deep learning techniques has attracted increasing interest and demonstrated an impressive performance.

Deep learning

Deep learning is an emerging area and a subclass of machine learning (ML). It comprises multiple hidden layers of artificial neural networks [38]. It has many architectures such as

- Convolutional neural networks (CNNs)
- Recurrent neural networks (RNNs)
- Graph neural networks (GNNs)
- Autoencoders (Aes)

These architectures have been used in fields like natural language processing(NLP), speech recognition, bioinformatics and computer vision.

Artificial neural network

Artificial neural networks (ANNs) are computer neural networks that seek to imitate the decision process in networks of nerve cells (neurons) in the biological (human or animal) central nervous system in a gross manner, as their name implies. They were introduced by the neurophysiologist **Warren McCulloch** and the mathematician **Walter Pitt** in their paper: " A logical calculus of the ideas immanent in nervous activity", in 1943 [30].

Neuron

A neuron is a nerve cell, these cells communicate using some links called synapses.

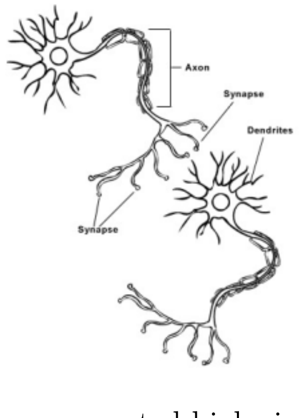


Figure 2.2: Two connected biological neurons [13]

Biological neurons are fired based on the intensity of the entering signals. This process can be simulated by this activation function :

$$y = \sigma(\sum w_i x_i) \tag{2.1}$$

Where y defines the output, σ is the activation function and w_i is the weight of input x_i

Activation function

Activation functions are functions used in neural networks **NN**s to compute the weighted sum of inputs and biases, which decides if a neuron can be fired or not [34]. Activation function has many types:

- Sigmoind function
- Hyperbolic tangent function (Tanh)
- Rectified Linear Unit function (ReLu)
- Leaky ReLU (LReLU)
- Softmax Activation Function

The image below describes a neuron in which the activation function takes the sum of inputs as **z** and outputs a value **a**.

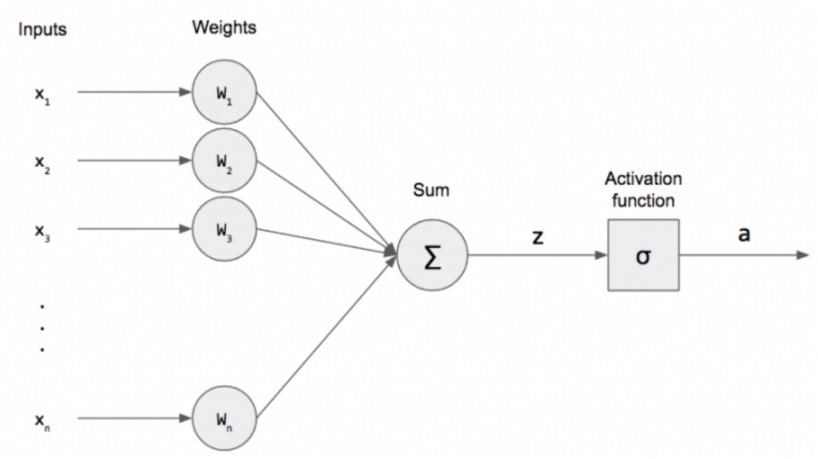


Figure 2.3: The structure of an artificial neuron [31]

Deep neural network

A deep neural network can take a single image, multiple images or a video stream as input. Using one single image in 3D reconstruction is extremely difficult due to the ambiguities unlike the video stream where we can go across all the frames to ensure the smoothness and the consistency of the reconstruction [19].

3D reconstruction based on single/multiple images

The 3D reconstruction based on one single image using deep learning techniques encounters multiple challenges, which slows down this approach towards its maturity. Generally, the 3D reconstruction from a single image faces the six following challenges:

- Uncertainty reconstruction of objects
- Shape complexity reconstruction of objects
- Reconstruction of fine-grained objects
- Memory requirements and calculation time
- Training datasets
- Selectivity of 3D shape representations

The output can be represented depending on the network architecture, it also decides how the quality and the efficiency will be. Some of representation methods can be described as:

- *Volumetric representations*, very expensive in terms of memory requirements, extensively used in the early stage deep learning-based 3D reconstruction techniques [19].
- *Surface-based representations*, introduced in many articles as meshes and point clouds.
- *Intermediation*, "while some 3D reconstruction algorithms predict the 3D geometry of an object from RGB images directly, others decompose the problem into sequential steps, each step predicts an intermediate representation" [19].

Dataset

Multiple popular datasets have been published such as ShapeNet, ModelNet, IKEA, Pix3D and PASCAL 3D+, each dataset has its own characteristics. For example, if we take the ModelNet dataset, we find that it has 662 object categories and covers about 127,915 CAD models.

Dataset	Data Type	Classes	Models/Images
ShapeNet	Synthetic	55	51,300/-
ModelNet	Synthetic	662	127,915/-
IKEA	Real	6	219/759
Pix3D	Real	9	395/10,069
PASCAL3D+	Real	12	3000+/-

Table 2.1: Popular datasets for 3D Reconstruction [16]

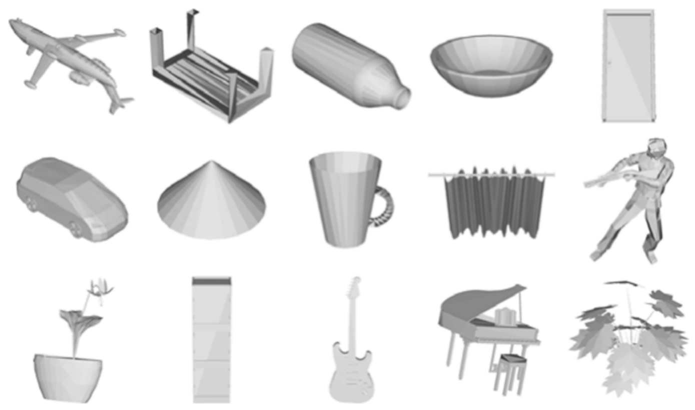


Figure 2.4: Some examples of the ModelNet dataset [33]

Convolutional neural network

Basically, CNNs and RNNs are the most used NNs in deep learning-based 3D reconstruction. Convolutional neural network is a class of ANNs, consists of input layer, output layer, lot of hidden layers and million of parameters that have the ability to learn complex objects and patterns. It uses a special technique called convolution.

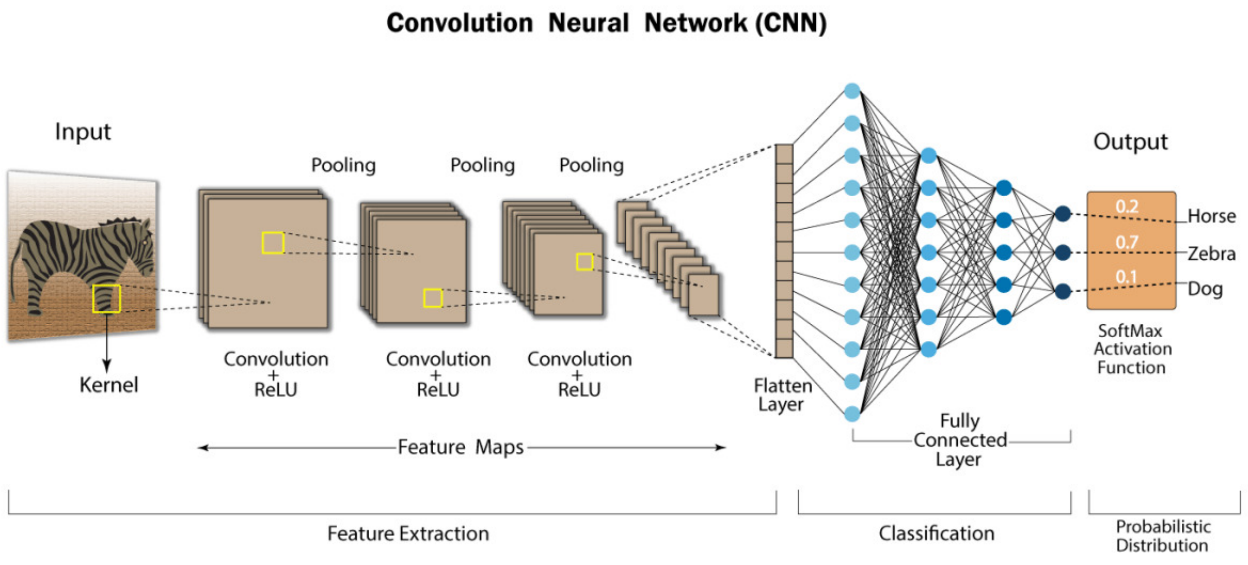


Figure 2.5: Convolutional neural network [11]

Convolution

Convolution is the process of combining two functions to produce the output of the other function. The input image is convolutioned with the application of filters in CNNs, resulting in a feature map.

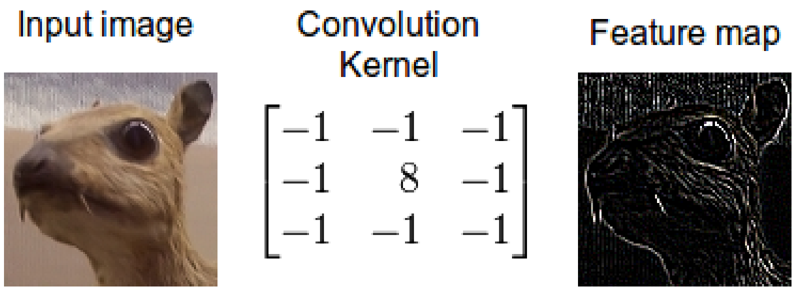

Input image
Convolution Kernel
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$
Feature map

Figure 2.6: Convolution of an image with an edge detector kernel, *nvidia developer*.

Convolution happens in the CNN's convolution layer following this equation:

$$InputImage * FeatureDetector = FeatureMap \tag{2.2}$$

Limitations

As much power as CNN has, it has limitations, several studies have shown that CNNs trained on ImageNet and other popular datasets fail to detect objects when they see them under different lighting conditions and from new angles.

Recurrent neural network

A recurrent neural network is also a neural network which uses sequential data or time series data. RNN is different from CNN, it is distinguished by its "memory" as it take information from prior inputs to influence the current input and output.

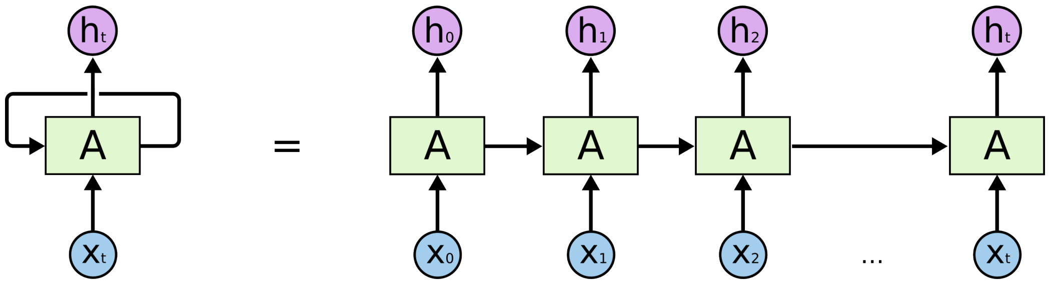


Figure 2.7: Basic architecture of a Recurrent Neural Network [7].

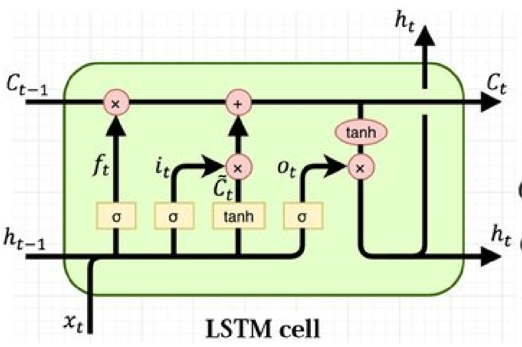
The above figure shows multiple RNN cells that take different inputs at different time steps, this allows RNNs to calculate the error for each step by using the **Backpropagation through time (BPTT)**.

Limitations

Like CNNs, RNNs also have their limitations, they can not learn long term dependencies in addition to the problem of vanishing/exploding gradients. These issues led to the new **LSTM** model.

Long Short-Term Memory network

Long Short-Term Memory networks, which called "LSTMs" are implementations of RNN which can be effectively trained and good at identifying long range dependencies. The main difference between RNNs and LSTMs is that they differ in the structure of the repeating model in which RNNs overwrite the hidden state and LSTMs add to the hidden state.



$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\
C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
$$

Figure 2.8: Structure of the LSTM cell and equations that describe the gates of an LSTM cell [39].

Gated recurrent unit

There is another architecture which called **Gated Recurrent Unit (GRU)**. Basically, GRU works with two parameters called update gate and reset gate, the first one controls the amount of the past information still want to remember and the second one decides how much old states will be passed in the future.

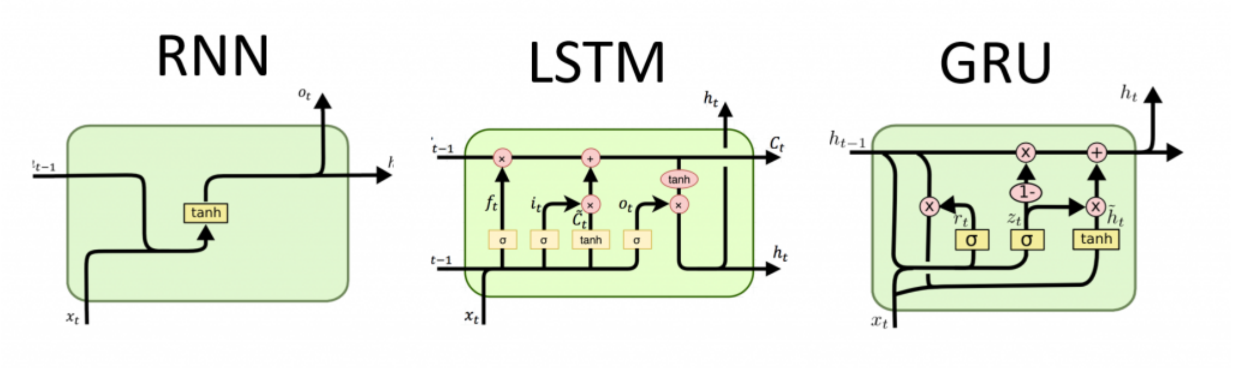


Figure 2.9: Structure of the cell in each of RNN, LSTM and GRU [9]

2.2.2 Traditional approaches

Shape from shading

Shape from shading or shading-based surface reconstruction, introduced by Horn in 1970. It is the process of computing the three-dimensional shape of a surface starting from the brightness of one black and white image of the given surface [36].

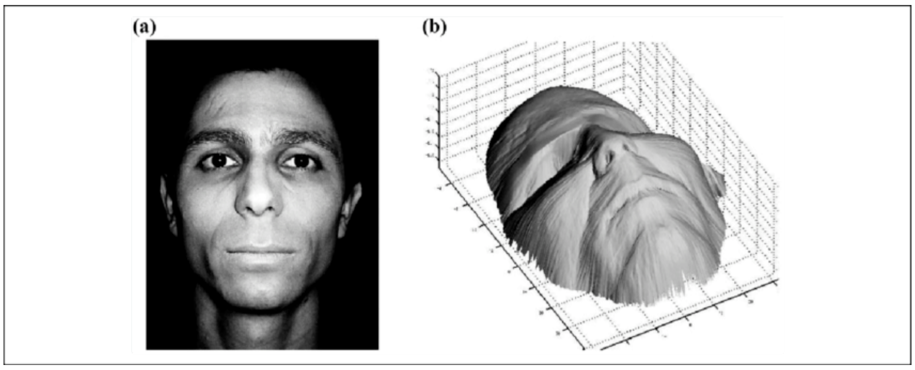


Figure 2.10: (a) A real face image. (b) Surface recovered from (a) [22]

Shape from shading technique depends on given assumptions about the lighting conditions and surface reflectance properties. Many efforts have been dedicated to solve this problem, some of these efforts have led to the partial differential equation(PDE).

Limitations

Shape from shading technique has been improved but not enough the reach a maturity that enables it to get accurate results.

Shape from silhouette

Shape from silhouette, first introduced by Baumgart in 1974 in his PhD thesis, it is a technique that uses a given silhoutte images to estimate the shape of an object. After Baumgart's introduction, multiple variations of the shape from silhouette paradigm have been proposed [15]. Some of these variations are:

- Representing the reconstructed shape by using volumetric descriptions, proposed by Aggarwal et al.
- Using octree data structure to speed up the SFS, suggested by Potmesil, Noborio et al and Ahuja et al.
- Shanmukh and Pujari proposition for taking silhouette images.
- Szeliski's non-invasive 3D digitizer that uses a single camera and a turntable.

The output of the shape from silhouette algorithm is called Visual Hull(VH), it has been used for a decade by the researchers to obtain the estimated shape following the shape from silhouette principle [15].

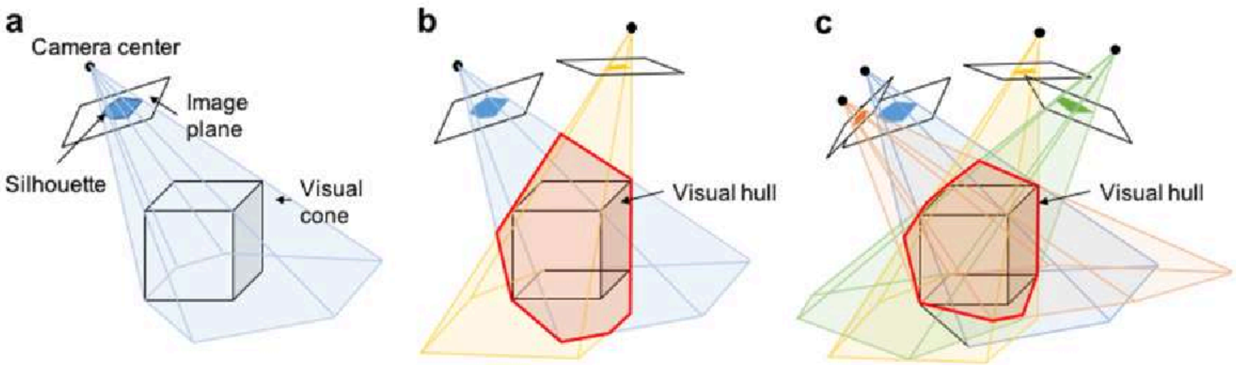


Figure 2.11: (a) visual cone obtained by back projection of silhouette image; (b) visual hull obtained by intersection of two visual cones; and, (c) 3D shape acquired by bounding geometry of resultant visual hull with multiple visual cones [28]

Structure from motion

Structure from motion(SFM) is a technique that uses a sequence of 2D images taken from different places to obtain the 3D structure of a scene or an object, it has a lot of similarities with our system, for example, SFM uses the triangulation method to calculate the relative 3D positions (x,y,z) of objects. SFM technique depends on matching features instead of matching pixels, we use feature extractors such as SURF, SIFT and ORB.

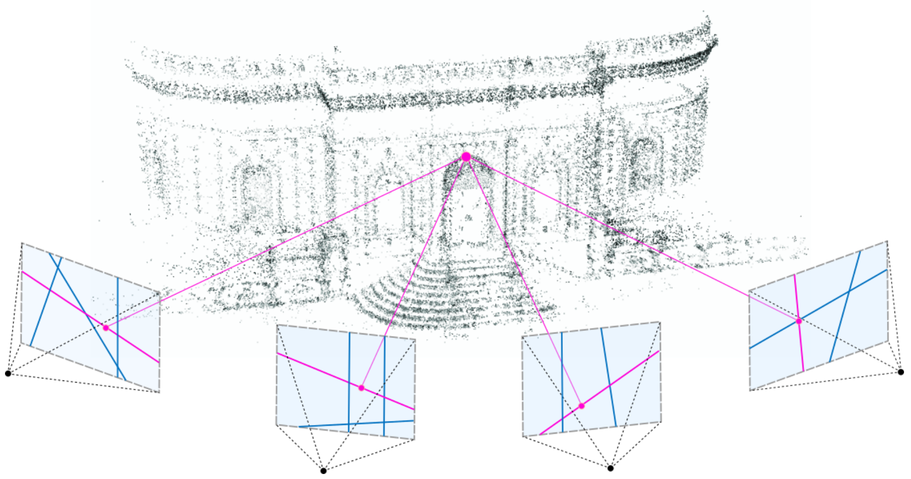


Figure 2.12: 3D point from a scene and it's corresponding 2D point on each image plane [28]

The scale-invariant feature transform (SIFT) is an algorithm that detect features in images,it was published by David Lowe in 1999. SIFT allows corresponding features to be matched even with large variations in scale and viewpoint and under conditions of partial occlusion and changing illumination.

Limitations

Since SFM depends on matching features instead of pixels, this can be considered as limitation, so it can not be used in applications that depend on accurate 3D reconstruction.

2.3 Proposed system description

Our system is basically a stereo rig in which two cameras are separated, horizontally aligned and well fixed, acting as human eyes. These cameras will be calibrated so that we obtain the intrinsic and the extrinsic parameters in which they will be used to rectify images by removing the lens distortion. Intrinsic parameters are the focal length, skew, distortion coefficients and the image center and for the extrinsic parameters, they can be defined as the camera position and its orientation. **NASA** uses a similar system on its new Mars 2020 Rover.

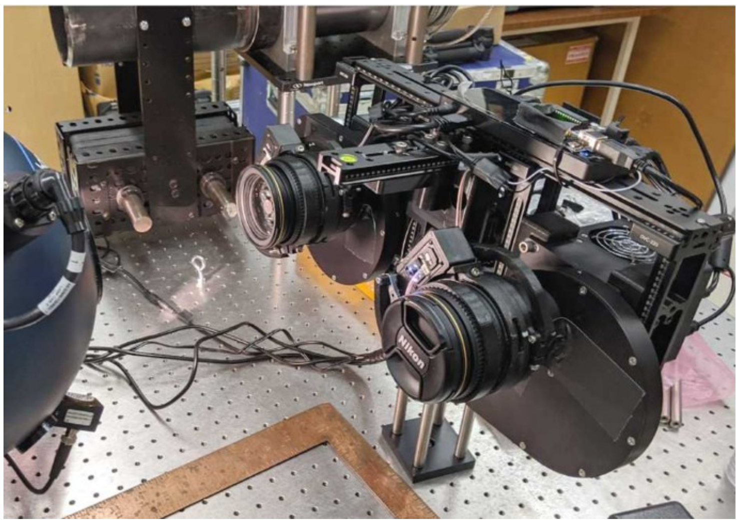


Figure 2.13: The NASA Perseverance rover Mast Camera Zoom [21]

The Mastcam-Z system aboard the NASA Perseverance rover consists of a pair of zoomable, focusable, multi-spectral, and color charge-coupled device (CCD) cameras, as well as accompanying electronics and two calibration targets, installed on top of a 1.7 m Remote Sensing Mast. The cameras use identical optical assemblies with focal lengths ranging from 26 mm (25.5 ×19.1 FOV) to 110 mm (6.2 ×4.2 FOV). Data will be collected at pixel scales of 148-540 m at a range of 2 m and 7.4-27 cm at a distance of 1 km. With a stereo baseline of 24.3±0.1 cm and a toe-in angle of 1.17±0.03 , the cameras are positioned atop the rover's mast (per camera) [21].

Our system's cameras take photos simultaneously so in each iteration, we will obtain two images that are almost identical except they were taken from two different points.

Each pixel in the left image will be mapped to some pixel in the right image, after the mapping of pixels, now we have a disparity map which is the distance of a pixel that has moved between the left and the right image. The disparity map will be converted to a point cloud, the process from taking images to the point cloud part will be repeated as much as we cover all the scene or the given object and as a final step, all the produced point clouds will be merged to one dense point cloud that represents our final 3D model. The system can be controlled from a mobile device that establishes a socket connection, which enables us to send commands from distance.

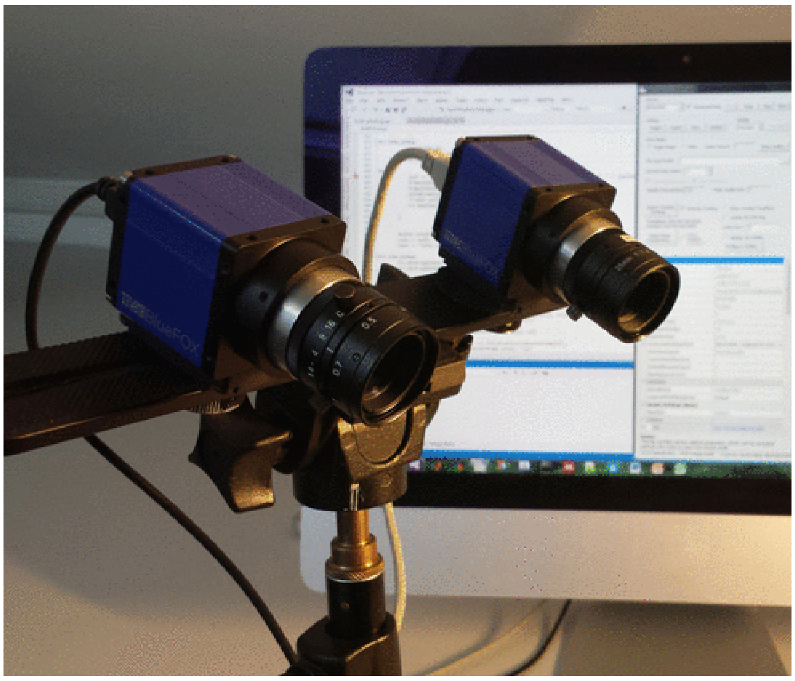


Figure 2.14: Similar setup to our system

2.4 Goals

The 3D reconstruction of scenes/objects can be described as taking as input a pair of 2D stereo photos and producing as output a group of 3D points constructed using those photos, this output is called a point cloud which is the representation of the 3D-reconstructed scene/object, this process called point cloud registration.

One of the main goals that we focus on reaching is the ability of our system to convert physical objects into precise digital models, this enables us to obtain our object's shape and geometries quickly and accurately. This system can be used extensively in manufacturing, quality inspection, analysis...etc.

2.5 Conclusion

In this chapter, we started with an introduction about the problem to be processed, also, we have gone through multiple approaches in the context of 3D reconstruction, this led us to discover some techniques and methods as well as their limitations, following by a brief description of our proposed system and its goals to be achieved.

Chapter 3

Architecture and design

3.1 Introduction

Our brains usually get almost identical pictures of a scene taken from two neighboring places at the same horizontal level because of the way our eyes are positioned and operated. The relative locations of two objects in the two eyes will change if they are separated in depth from the viewer. Our brains are capable of measuring this difference and estimating depth utilizing it [8].

Depth perception has been one of the important tasks in computer vision systems. A stereo camera system is the most often used system for obtaining depth information by taking images as input. Stereo correspondence is the process of matching each point in the left image by its corresponding point in the right image which outputs what we called a disparity map, this process is quite challenging and consumes a lot of resources, however, this heavy search can be reduced to one dimensional search by applying an accurate stereo rectification which handles the lens distortions in the images.

Mapping a point in the left image to its corresponding point in the right images is called the point correspondence, this needs a good camera calibration that takes the responsibility of finding the intrinsic parameters and the extrinsic ones for the cameras, so the position of each camera is known. The point correspondence has been considered as a problem because the point that has to be matched should be distinguishable from the pixels around it, this led to several constraints that have been imposed in order to minimise the false correspondences in the image pair. In 1979, Marr and Poggio imposed the uniqueness constraint which requires that a pixel from one image cannot match to more than one pixel in the other image [17]. The problem of finding the corresponding point in the presence of occluded regions made a big challenge, however, another constraint was imposed by Baker and Binford in 1981, the ordering constraint which states that the pixels ordering is maintained throughout the images, this constraint can be violated if an object in the scene is very close to the camera more than the background. Marr and Poggio introduced another constraint which called the continuity constraint, ”the disparity map should vary smoothly almost everywhere in the image” [17].

Some of stereo vision applications require real-time 3D reconstruction, there is always a trade-off between accuracy and speed when it comes to disparity maps, especially when the images have big size or the disparity range is long, using powerful machines that have large computational resources may tackle this problem, even the point clouds have big weight in the storage, storing and processing them increase the need for such powerful machines that have a parallelism and pipelining architectures.

3.2 System architecture description

Our system is a binocular stereo vision system, consisting of a pair of cameras rigidly fixed with respect to each other, these cameras take photos simultaneously, by following a pipeline, we can obtain a 3D model from these photos. The image below describes the followed pipeline, each step of this pipeline will be explored in depth.

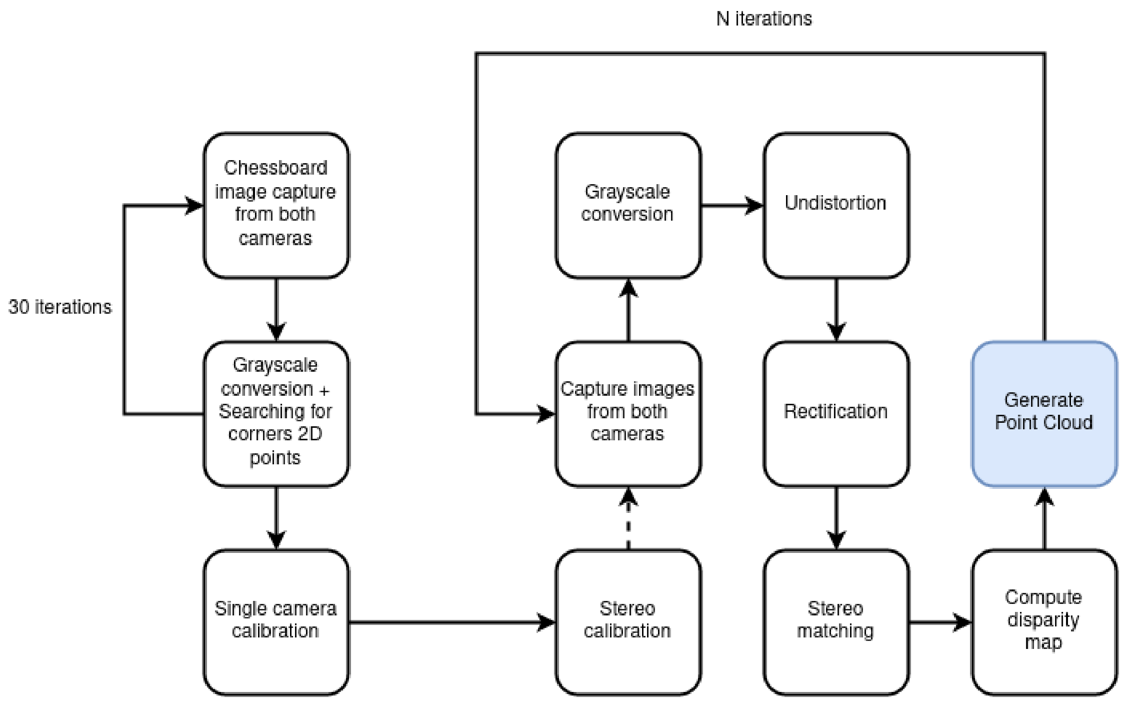

N iterations
Chessboard image capture from both cameras
30 iterations
Grayscale conversion + Searching for corners 2D points
Single camera calibration
Grayscale conversion
Capture images from both cameras
Stereo calibration
Undistortion
Rectification
Stereo matching
Generate Point Cloud
Compute disparity map

Figure 3.1: Pipeline

3.2.1 Camera calibration

When it is used as a visual sensor, a camera is an essential component of a variety of domains, including robotics, surveillance, industrial automation, space exploration...etc.

Camera calibration is the process of estimating some parameters that we need to determine an accurate relationship between a 3D point in the real world and its corresponding pixel in the image captured by that calibrated camera. Those parameters are the intrinsic parameters and extrinsic parameters.

Intrinsic parameters

Also known as internal parameters , they are the camera's focal length, optical center, and distortion coefficients of the lens...etc. These parameters are defined by a 3×3 upper triangular matrix **K**:

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

f_x, f_y are the x and y focal lengths, usually they are the same.
c_x, c_y define the coordinates of the optical center.
γ is the skew between the axes, it is usually 0.

The **Focal length** is one of the basic characteristics of a photographic lens, we represent it in millimeters (mm), but it is not the actual real length of a lens. It is determined when the lens is focused at infinity [2]. It controls the angle of view, i.e, how much of the scene will be captured, the zoom, i.e, how large individual elements will be. The shorter the focal length, the wider the angle of view and the lower the zoom, and vice versa.

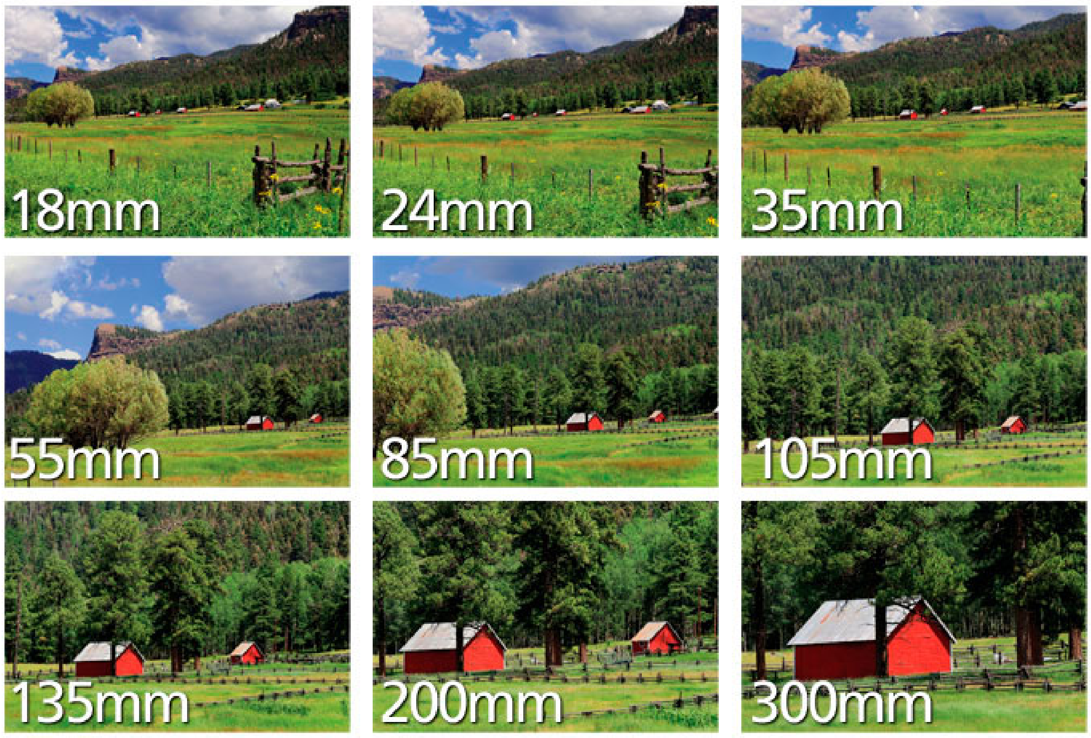


Figure 3.2: The same scene with different focal lengths [2]

The **Optical center** is a point on the principal axis of the lens, such that a ray of light passing through it goes without any deviation.

The **Distortion coefficients** describe mathematically the distortions in the image, we will explore them in depth later.

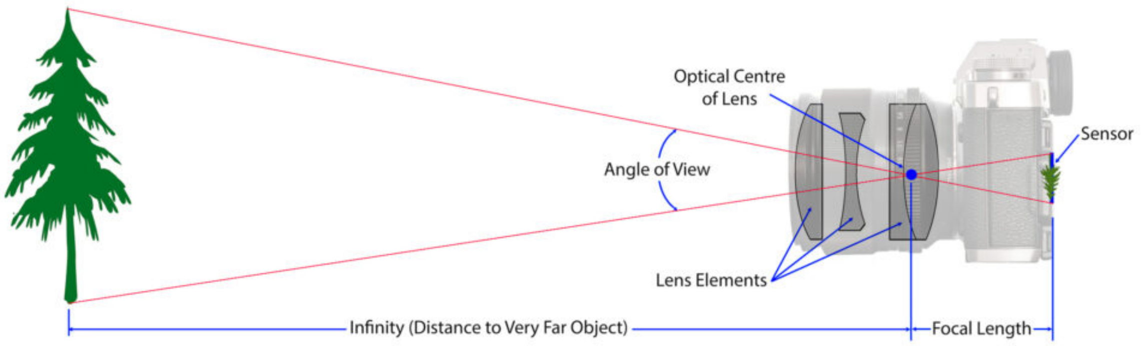


Figure 3.3: Focal length, angle of view and the optical center [10]

Extrinsic parameters

Sometimes they are called by external parameters, they define the camera's rotation and translation, with respect to a world coordinate system. Generally, we define the external parameters by one matrix called the extrinsic matrix that combines a 3×3 rotation matrix **R** and a 3×1 translation vector **t**.

Extrinsic parameters enable us to find the 3D point projection onto the image plane by transforming the point from the world coordinate system to the camera coordinate system.

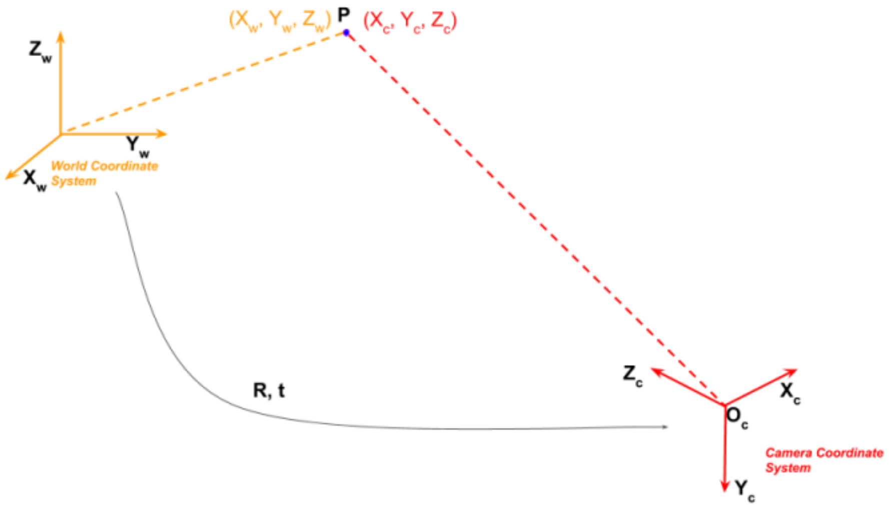


Figure 3.4: World coordinate system and camera coordinate system [12]

In order to project the 3D point from the world coordinate system onto the image plane, we use the intrinsic parameters. The equations below elaborates the relation between a 3D point (X_w,Y_w,Z_w) in the world coordinate system and its projection (u,v).

$$\begin{bmatrix} u' \\ v' \\ z' \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.1}$$

$$u = \frac{u'}{w'} \tag{3.2}$$

$$v = \frac{v'}{w'} \tag{3.3}$$

P refers to the projection matrix, it is a 3×4 matrix that consists of the intrinsic matrix K and the extrinsic matrix [R|t].

$$\mathbf{P} = \mathbf{K} \times [\mathbf{R}|\mathbf{t}] \tag{3.4}$$

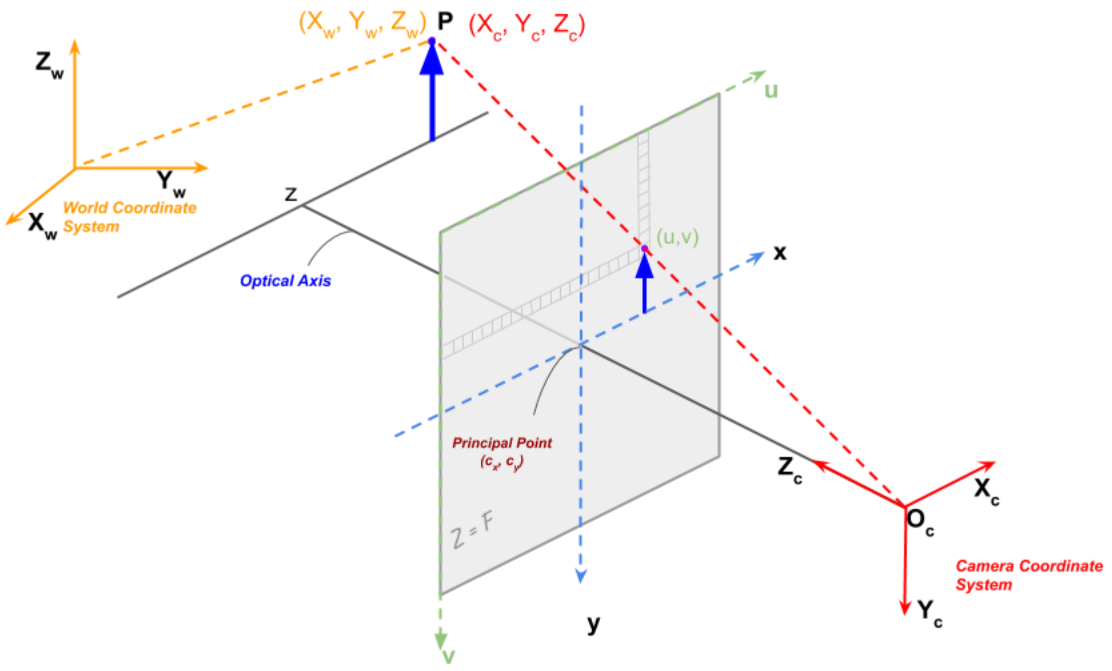


Figure 3.5: World coordinate system, camera coordinate system and the image coordinate system(P is a 3D point here) [12]

Calibration methods

Many methods have been imposed to do the calibration process such as the calibration using a pattern. It can be done by capturing multiple images of a known dimensions pattern from different view points. One of the widely used methods that belongs to the pattern-based calibration is the calibration using a checkerboard, it is easy because the checkerboard has squares and these squares have corners that have sharp gradients in two directions and that is what makes them ideal to be localized robustly.

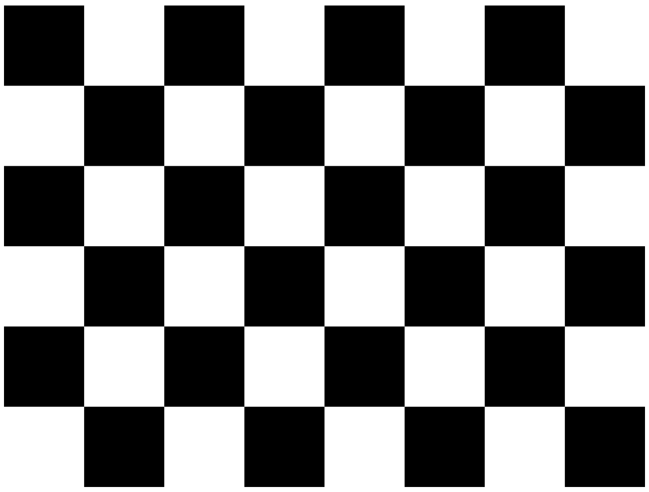


Figure 3.6: Checkerboard pattern

Calibration process

The calibration process is illustrated by the flowchart below.

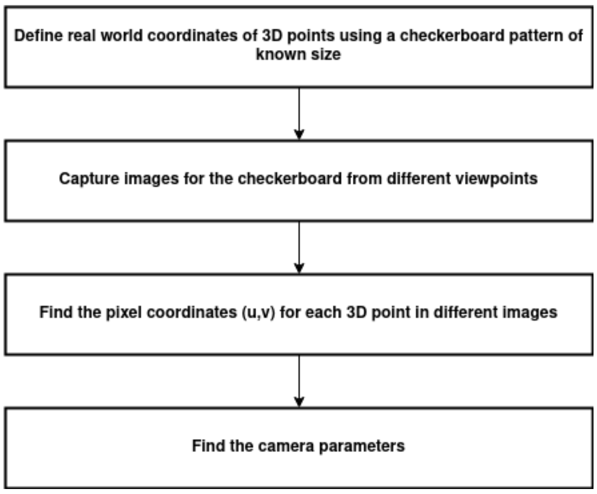

Define real world coordinates of 3D points using a checkerboard pattern of known size
Capture images for the checkerboard from different viewpoints
Find the pixel coordinates (u,v) for each 3D point in different images
Find the camera parameters

Figure 3.7: Calibration process flowchart

The first step is to define real world coordinates using a checkerboard pattern in which the size is known and the printing quality is good, also it would be recommended to not fitting the checkerboard to the page. The 3D points are defined by the squares corners of the checkerboard.

The second step is to capture images for the checkerboard from different viewpoints, to achieve this, we can glue the checkerboard on a flat and solid object, our printed pattern must be completely flat. As minimum, we capture at least 20 images, as we said before, these image must be taken from different angles and different distances.

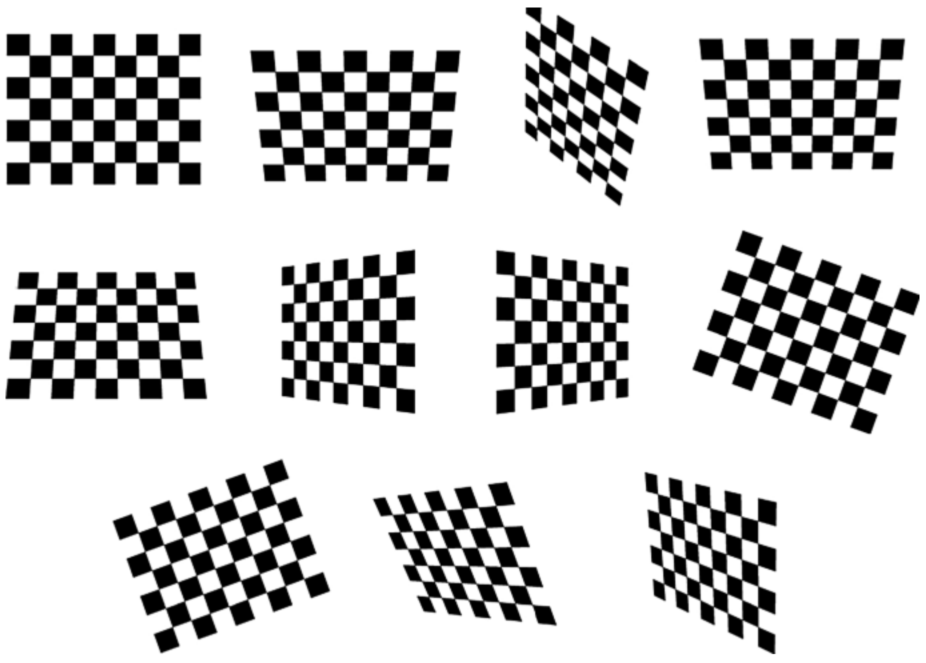


Figure 3.8: Checkerboard from different viewpoints [24]

Since we have taken multiple images for the checkerboard, it is time to find the pixel coordinates (u,v) for each 3D point in different images. As we know the 3D location of the checkerboard corners in the world coordinate system, now we need the 2D pixel locations of these corners in the images. **findChessboardCorners** is an OpenCV built-in function that attempts to find whether the input image is a view of the chessboard pattern and locate the internal chessboard corners. The detected points are not completely accurate, the function calls **cornerSubPix** to determine the accurate positions [4]. The function returns the coordinates of the corners by the taking the following arguments as input:

- **image:** Source chessboard view. It must be an 8-bit grayscale or color image.
- **patternSize:** Number of inner corners per a chessboard row and column.
- **flags:** Various operation flags.

A Python example for this function would be like this:

```
retval, corners = cv2.findChessboardCorners(image, patternSize, flags)
```

where **corners** is the output array of detected corners.

Now, in order to get the camera parameters, it remains one last step, which is passing the 3D point in world coordinate system and their 2D location in all the images as input to the OpenCV built-in function **calibrateCamera**. A python example for this function would be like this:

```
retval, cameraMatrix, distCoeffs, rvecs, tvecs =
↪  cv2.calibrateCamera(objectPoints, imagePoints, imageSize)
```

Inputs and outputs are explained in the following

- **objectPoints:** A vector of vectors of 3D points.
- **imagePoints:** A vector of vectors of the 2D image points.
- **imageSize:** Image size.
- **cameraMatrix:** Intrinsic camera matrix.
- **distCoeffs:** Lens distortion coefficients.
- **rvecs:** Rotation specified as a 3×1 vector. The direction of the vector specifies the axis of rotation and the magnitude of the vector specifies the angle of rotation.
- **tvecs:** 3×1 Translation vector.

3.2.2 Stereo calibration

After calibrating each camera individually, we need to perform a stereo calibration in order to find the transformation between the stereo camera pair,the essential matrix, and the fundamental matrix. If the poses of an item relative to the first and second cameras are computed, **(R1,T1)** and **(R2,T2)**, respectively, for a stereo camera where the relative position and orientation between the two cameras are fixed, then those poses definitely relate to each other. It is possible to compute **(R2,T2)** when **(R1,T1)** is given and the relative position and orientation (**R,T**) of the two cameras is known [3].(**R,T**) is computed such that:

$$R_2 = RR_1 \tag{3.5}$$

$$T_2 = RT_1 + T \tag{3.6}$$

Then, we can calculate the coordinate representation of a 3D point for the second camera's coordinate system when given the point's coordinate representation in the first camera's coordinate system:

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} \tag{3.7}$$

And the essential matrix E, which is a matrix that relates the corresponding points in the stereo images and it can be used only in relation to calibrated cameras for determining the relative position and orientation between these cameras and the 3D position of corresponding image points. It can be computed like this:

$$E = \begin{bmatrix} 0 & -T_2 & T_1 \\ T_2 & 0 & -T_0 \\ -T1 & T_0 & 0 \end{bmatrix} R \tag{3.8}$$

T_i are components of the translation vector:

$$T : T = [T0, T1, T2]^T \tag{3.9}$$

The fundamental matrix F which is similar to the essential matrix but it deals with uncalibrated cameras and has different number of parameters [1]:

$$F = K_1^{-1} \cdot E \cdot K_2^{-T} \tag{3.10}$$

where K_1 and K_2 are the camera matrices for the two cameras.

The OpenCV's built-in function **stereoCalibrate** does the previous computation, it minimizes the total re-projection error for all the point in each views from both cameras and returns a value as the final re-projection error [3]. The function can also perform calibration for each of the two cameras but it is recommended to calibrate each camera individually due to the high dimensionality of the parameter space and noise in the input data which can lead to inaccurate solution. A Python implementation for this function would be like this:

```
retVal, cameraMatrixL, distCoeffsL, cameraMatrixR,
distCoeffsR, R, T, E, F = cv2.stereoCalibrate(objectPoints,
                                              imagePointsL,
                                              imagePointsR,
                                              cameraMatrixL,
                                              distCoeffsL,
                                              cameraMatrixR,
                                              distCoeffsR,
                                              imageSize,
                                              criteria,
                                              flags)
```

Inputs and outputs are defined in the following:

- **objectPoints:** Vector of vectors of the calibration pattern points.
- **imagePointsL:** Vector of vectors of the projections of the calibration pattern points, observed by the left camera. .
- **imagePointsR:** Vector of vectors of the projections of the calibration pattern points, observed by the right camera.
- **cameraMatrixL:** Input/output camera intrinsic matrix for the left camera.
- **distCoeffsL:** Input/output vector of distortion coefficients.
- **cameraMatrixR:** Input/output camera intrinsic matrix for the right camera.
- **distCoeffsR:** Input/output lens distortion coefficients for the second camera.
- **imageSize:** Size of the image used only to initialize the camera intrinsic matrices.
- **R:** Output rotation matrix.
- **T:** Output translation vector.
- **E:** Output essential matrix.
- **F:** Output fundamental matrix.
- **criteria:** Termination criteria for the iterative optimization algorithm.
- **flags:** Various operation flags.

3.2.3 Stereo rectification

Stereo rectification is about using the camera parameters and both rotation and translation between the cameras to make the images planes of the cameras in the same plane. Before we dive deeper in the stereo rectification process, we should understand what lens distortion means.

Generally, there are two types of distortions: the perspective and the optical. Both change the image formation, from slight to noticeable deformation. Optical distortion is the one that concerns us because it is related to how the lens is designed. Basically, there are two major types of distortion:

- Radial distortion
- Tangential distortion

Radial distortion happens when light rays bend more at the lens's borders than at the lens's optical center. The smaller the lens, the higher the distortion. There are two types of radial distortion:

- **Barrel distortion:** Positive radial displacement.
- **Pincushion distortion:** Negative radial displacement.

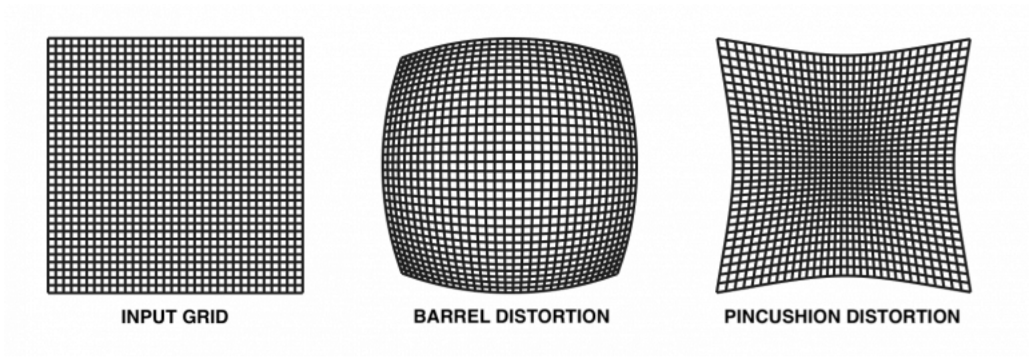


Figure 3.9: Without distortion, barrel distortion, and pincushion distortion.

Tangential distortion occurs when the lens and the image plane are not parallel. As we discussed before, lens distortion is represented mathematically by some coefficients called the distortion coefficients. The distortion coefficients returned from the camera calibration process are represented by values from K_1 to K_6 as radial distortion and P_1,P_2 as tangential distortion.

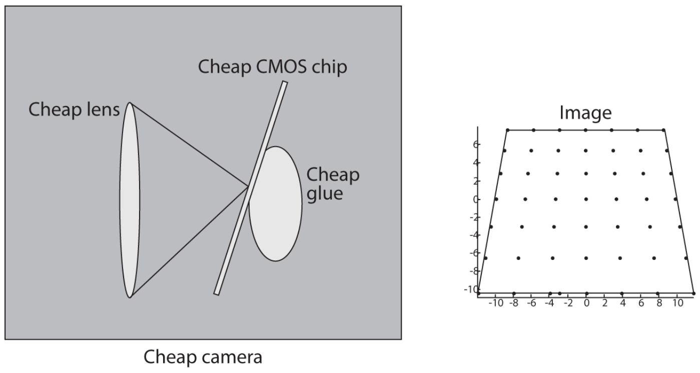


Figure 3.10: Tangential distortion results when the lens is not fully parallel to the image plane [25]

Stereo pair rectification is a transformation of each image in which the **epipolar lines** become parallel to one of the image axes, this process minimizes the search for the corresponding points to 1D search.

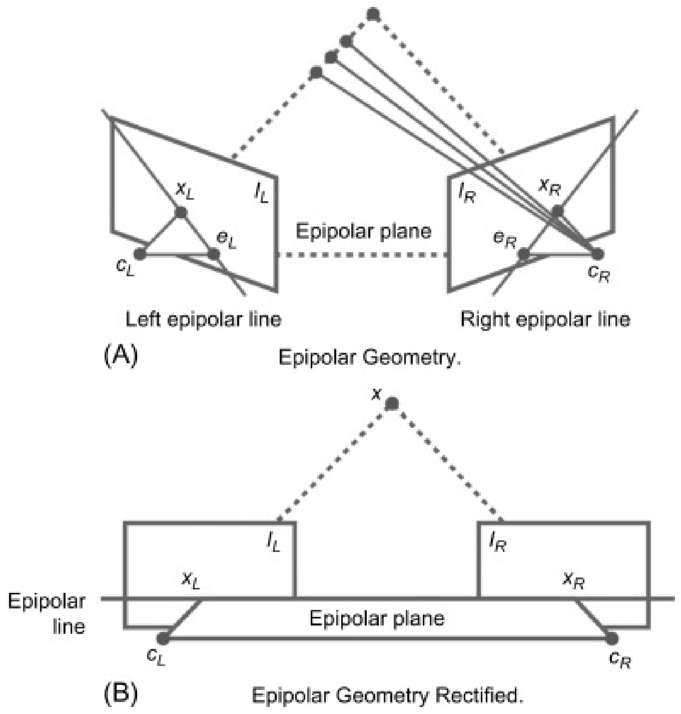


Figure 3.11: (A) Epipolar geometry, (B) Epipolar geometry rectified.

Rectification algorithm can be summarized in these basic steps:

- Rotate the left camera to make its image plane parallel to the baseline of the system.
- Apply the same rotation to the right camera to recover the original geometry.
- Rotate the right camera so that its image plane will be parallel to the baseline.
- Adjust the scale in both the camera reference frames.

The OpenCV's built-in function **stereoRectify** computes the rotation matrices for each camera, it takes as input the matrices calculated by **stereoCalibrate** and it outputs two rotation matrices as well as two projection matrices in the new coordinates. The stereo rig can take two positions:

- **Horizontal position**: The left and the right cameras views are shifted along the x-axis relatively to each other, which makes the corresponding epipolar lines horizontal in the left and the right cameras and they have the same y-coordinate in the rectified images. the two projection matrices P1 and P2 look like the following:

$$P1 = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.11}$$

$$P2 = \begin{bmatrix} f & 0 & cx_2 & T_x * f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.12}$$

T_x is the horizontal shift between the cameras [3].

- **Vertical position**: The left and the right cameras views are shifted alongside the y-axis relatively to each other. The epipolar lines in the rectified images have the same x-coordinate. The two projection matrices P1 and P2 look like:

$$P1 = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.13}$$

$$P2 = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_2 & T_y * f \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.14}$$

T_y is the vertical shift between the cameras [3].

The first three columns of P1 and P2 are the new rectified camera matrices that will be passed with R1 and R2 to **initUndistortRectifyMap** function that initializes the rectification map for each camera.

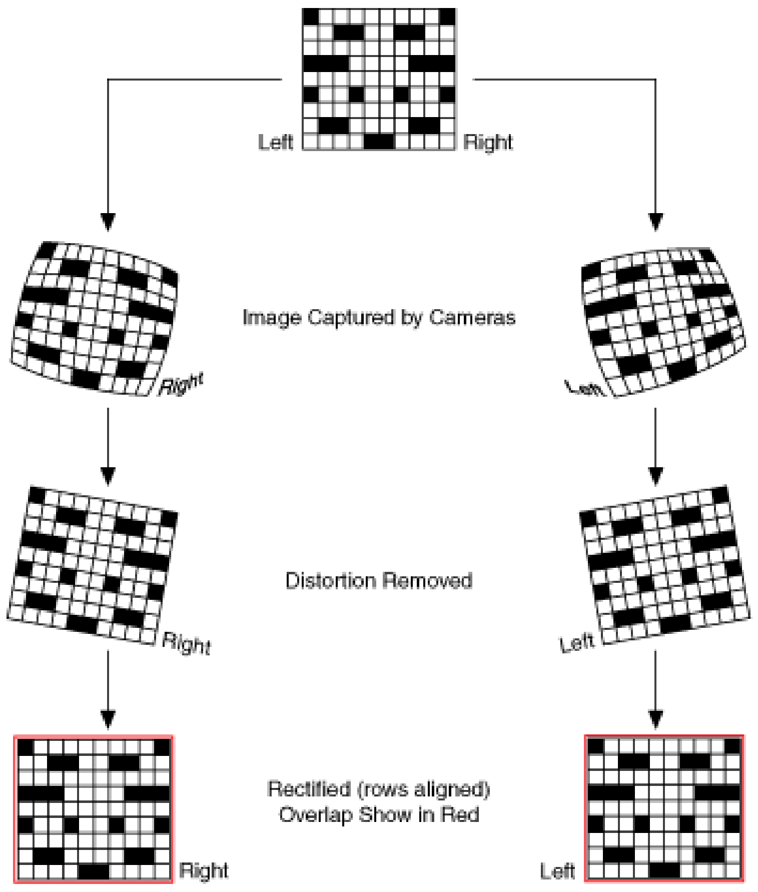


Figure 3.12: Distortion removing and rectification

A Python example for stereoRectify function would be like this:

```
R1, R2, P1, P2, Q, validPixROI1,validPixROI2 =
                             cv2.stereoRectify(cameraMatrixLeft,
                                               distCoeffsLeft,
                                               cameraMatrixRight,
                                               distCoeffsRight,
                                               imageSize,
                                               R,
                                               T,
                                               alpha,
                                               flags)
```

where,

- **R1**: Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.
- **R2**: Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.
- **P1**: Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.
- **P2**: Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.
- **Q**: Output 4×4 disparity-to-depth mapping matrix
- **alpha**: Free scaling parameter. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases.
- **validPixROI1**: Optional output rectangles inside the rectified images where all the pixels are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller.
- **validPixROI2**: Optional output rectangles inside the rectified images where all the pixels are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller.

The **initUndistortRectifyMap** function calculates the joint undistortion and rectification transformation and represents the result in the form of maps as output for remap. It computes the corresponding coordinates in the original image for each pixel (u,v) in the rectified image. A Python example for this function would be like the following:

```
map1, map2 = cv2.initUndistortRectifyMap(cameraMatrix,
                                         distCoeffs,
                                         Rect,
                                         newCameraMatrix,
                                         imageSize,
                                         flags)
```

It will be called twice, one for each of the two cameras.

3.2.4 Stereo matching

Stereo matching can be described as one of the core topics in computer vision, performing stereo matching means recovering the 3D structure of a given scene or object from 2D images. This topic lies on a lot of interesting areas such as augmented reality, robotics and autonomous driving. Given a pair of rectified stereo images, stereo matching has the responsibility of computing the disparity of each pixel in the left image(the reference image). The disparity is the horizontal shift between a pair of corresponding pixels in the left and right images. Stereo matching methods can be divided to two major methods, local methods and global methods. We are interested with the block matching method which is considered as a local method. Block matching searches one image for the best corresponding region for a template in the other image, i.e, shifting the template along the epipolar line in a predefined disparity range.

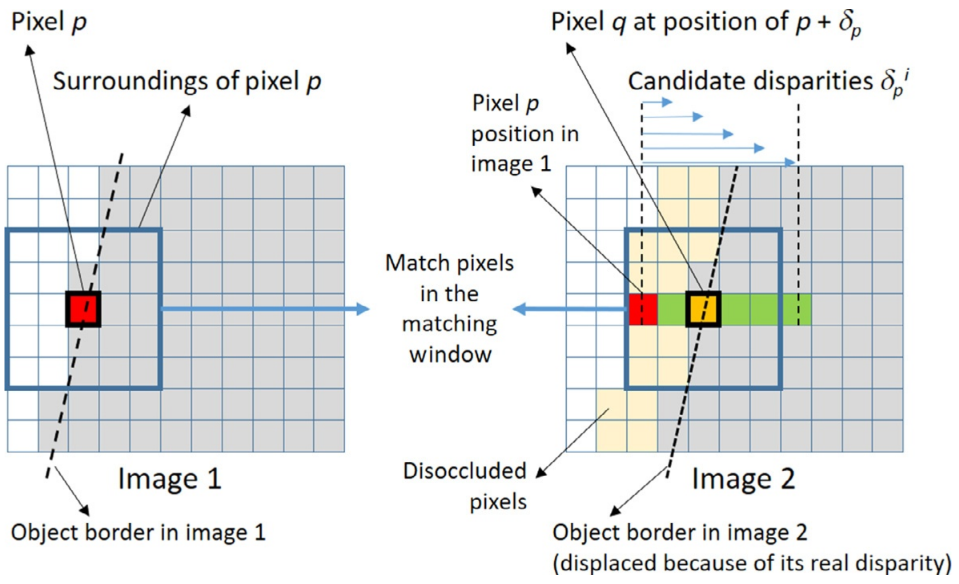

Pixel p
Surroundings of pixel p
Image 1
Object border in image 1
Match pixels in the matching window
Pixel q at position of $p + \delta_p$
Candidate disparities δ_p^i
Pixel p position in image 1
Disoccluded pixels
Image 2
Object border in image 2 (displaced because of its real disparity)

Figure 3.13: Pixel matching and disparity [37]

OpenCV has two different algorithm implementations for the stereo correspondence/-matching algorithms, they share common object interface. The first one is the block matching BM algorithm which is an effective and fast algorithm, it uses small SADs windows, SAD stands for the sum of absolute difference, it uses them to map points between the left and right stereo rectified images.

$$SAD = \sum |I_l(x, y) - I_r(x + d, y)| \tag{3.15}$$

The second algorithm is the SGBM, semi-global block matching, it differs from the first algorithm, this one does the matching at subpixel level and tries to enforce the global smoothness. BM is faster but not accurate and reliable as SGBM [26].

Since this project requires as much accuracy as possible, we decided to go with the SGBM algorithm. The following Python code creates an SGBM with several parameters, these parameters should be tuned repeatedly until we find the best result:

```python
minDisparity = 0
numDisparities = 64
blockSize = 8
disp12MaxDiff = 1
uniquenessRatio = 10
sepeckleWindowSize = 10
speckleRange = 8

sgbm = cv2.StereoSGBM_create(minDisparity = minDisparty,
                             numDisparities = numDisparities,
                             blockSize = blockSize,
                             disp12MaxDiff = disp12MaxDiff,
                             uniquenessRatio = uniquenessRatio,
                             speckleWindowSize = speckleWindowSize,
                             speckleRange = speckleRange)
```

Where,

- **minDisparity**: Minimum possible disparity value, normally it is 0.
- **numDisparities**: Maximum disparity minus minimum disparity, always greater than 0.
- **blockSize**: Matched block size, an odd number greater or equal to 1.
- **disp12MaxDiff**: Maximum allowed difference in the left right disparity check.
- **uniquenessRatio**: Margin in percentage by which the best (minimum) computed cost function value should ”win” the second best value to consider the found match correct. Normally, a value within the 5-15 range is good enough.
- **speckleWindowSize**: Maximum size of smooth disparity regions to consider their noise speckles and invalidate.
- **speckleRange**: Maximum disparity variation within each connected component.

3.2.5 Disparity map

As we have seen before, the goal of stereo matching is to calculate the disparity map which is the horizontal shift between a pair of corresponding pixels in the left and right images.

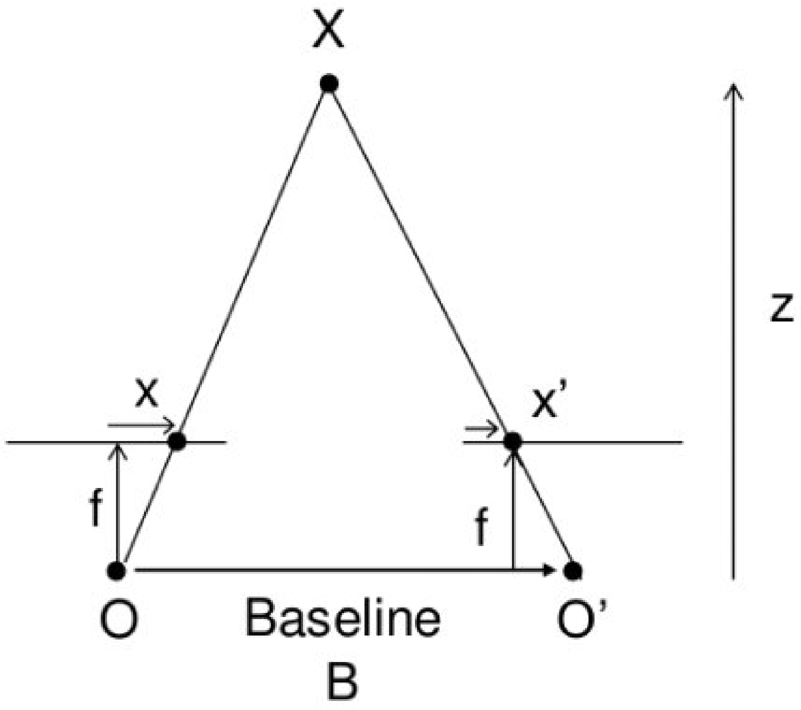


Figure 3.14: The baseline B, focal length f and the distance z.

The below equation demonstrates how to calculate the disparity between points in the image plane corresponding to the scene 3D point.

$$disparity = x - x' = \frac{Bf}{Z} \tag{3.16}$$

we can derive the depth of all the pixels in some image because the depth of a point is inversely proportional to the difference in distance of corresponding image points and their camera centers. A simple example of the disparity map of an image is illustrated in the image below:

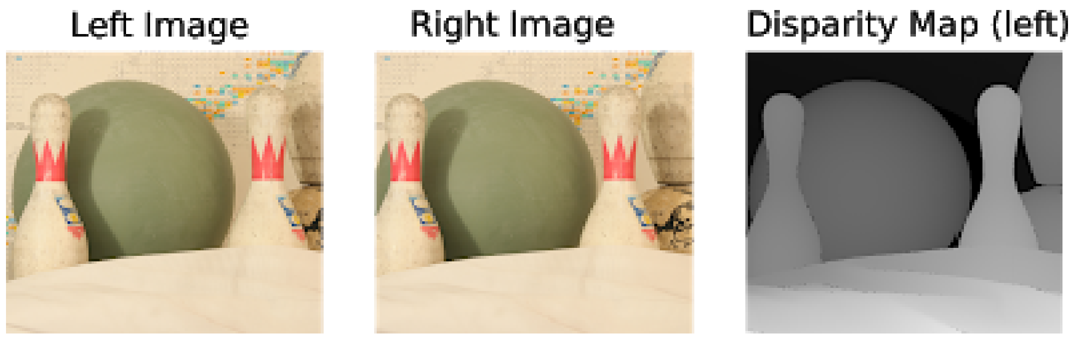


Figure 3.15: Disparity map

Using the SGBM object that we have created, we can compute the disparity map with this one line Python code:

```
disparity_map = sgbm.compute(leftImg, rightImg).astype(np.float32)
```

3.2.6 Point cloud generation

Generating a complete 3D scene is a significant technique for multiple computer vision applications such as high-precision 3D map reconstruction in autonomous driving and 3D reconstruction in general [23]. A point cloud is a set of data points in space. In a 3D coordinates system, these points construct the shape of an object to represent it, each point has its X, Y and Z coordinates. Point clouds can be used for many purposes such as creating 3D models for manufactured parts, quality inspection, visualization...etc.

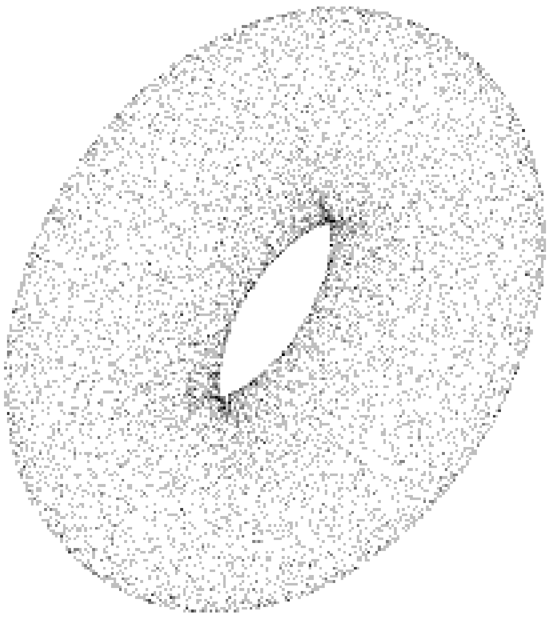


Figure 3.16: Simple Point Cloud

To get the whole given object shape, we need to go around it, this enables us to capture it fully, each pair of the images will produce a point cloud, all the point clouds will be merged to one dense point cloud. OpenCV provides a function that transforms a single-channel disparity map to a three-channel image representing a 3D surface, it takes as input the disparity map and a 4×4 perspective transformation matrix from stereo rectification.

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \begin{bmatrix} x \\ y \\ disparity(x,y) \\ z \end{bmatrix} \tag{3.17}$$

The process of merging point clouds together is called the point cloud registration, multiple algorithms exist for that, one of them is ICP (iterative closest point). The algorithm takes two point clouds as input, it works as following:

- For each point in the first point cloud (the source point cloud), it matches the closest point in the second point cloud.
- It performs an estimation for the combination of rotation and translation using a root mean square point to point distance metric minimization technique, this leads to good alignment between each point and its match.
- It transforms the source points using the obtained transformation.
- Re-associating the points, i.e, iterating.

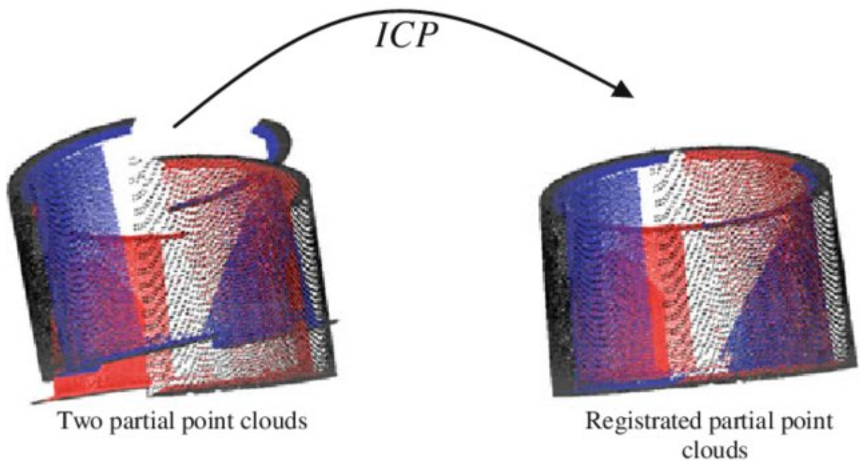


Figure 3.17: Point cloud registration using the ICP algorithm [14]

3.3 Conclusion

In this chapter, we have presented the system in depth, we have given similar setups like the one the NASA's Mars 2020 Rover uses, we have gone through different steps from image gathering to the point cloud generation, each step was explained in detail alongside with its proper implementation (Python) and mathematical proof.

Chapter 4

Implementation and results

During the journey of building the system, from the design to the last line of code, we have used multiple programming languages and several tools to get to this point. In this chapter, we will introduce these tools and these programming languages, evaluate and discuss results.

4.1 Tools and programming languages

Programming languages

Figure 4.1: Python

Python, it is an interpreted high-level and general-purpose programming language well suited to tasks such as cleaning data, interacting with web resources and parsing text [20], its first appearance was in February 1991 by Guido van Rossum, but he started working on it in the late 1980s. Python supports multiple programming paradigms such as procedural, object-oriented(OOP) and functional programming, it dynamically-typed with a garbage collector. It has many uses in major domains like artificial intelligence and its sub-domains such as machine learning, deep learning and computer vision using famous libraries such as Numpy, pandas, Keras, TensorFlow and OpenCV. Also, Python is used widely in web development as scripting language by web frameworks like Django, Flask, FastApi...etc. In this project, we have used python as main programming language due to its power and simplicity.

Figure 4.2: Java

Java, "write once, run anywhere", it is a general-purpose, high-level, class-based, object-oriented programming language that was originally developed by James Gosling at Sun Microsystems in 1995, it has been acquired by Oracle later. Latest version of Java is Java SE 16 which has been released in March 16, 2021. In this project we have used Java for creating a socket client to integrate it in an Android application which enables us to send commands to the system and the system will respond according to these commands.

Figure 4.3: Kotlin

Kotlin, it is a statically-typed, cross-platfrom, general-purpose programming language, designed with full interoperability with Java and more concise syntax. In May 7th, 2019, Google announced that Kotlin win be the number one programming language for Android. In this project, we have used Kotlin to create a simple android application that controls our system from distance and we take the benefit from its interoperability with Java by integrating the socket code that we have created using Java.

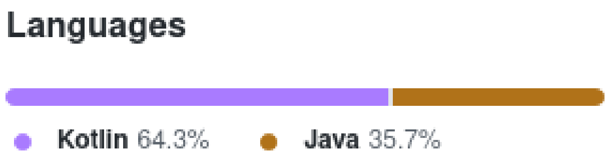


Figure 4.4: Github's statistics

Libraries

Figure 4.5: OpenCV

OpenCV (Open Source Computer Vision Library), an open source software library for computer vision and machine learning [5]. The goal behind building the OpenCV is to provide a common infrastructure for computer vision applications and to increase the use of machine perception in the commercial products. It has a lot of both classic and state-of-the-art computer vision and machine learning algorithms, more than 2500 optimized algorithms for multiple use cases, from the detection and recognizing of faces and objects to 3D point clouds production from stereo cameras. The library is the heart of this project, we have relied on it for the whole work.

Figure 4.6: Numpy

Numpy is an open source project created in 2005 that provides numerical computing ability with Python, it combines the expressive power of array programming alongside with the performance of C, use the Python's power that is represented in the readability, usability and versatility [20]. Numpy was initially developer by students and researchers to provider a powerful array programming for Python. In this project we have used Numpy mostly in saving and loading parameters to perform different operations.

Figure 4.7: Open3D

Open3D is an open source library that provides a great support for rapid development of software that deals with 3D data [40]. It has a highly optimized backend that uses parallelization techniques, it works on Windows, macOS and Linux.

Tools

Two IDEs have been used as tools for this project:

- **Pycharm** is an integrated development environment IDE developed by JetBrains, dedicated for Python language. Pycharm works on Windows, macOS and linux with two editions, the community edition and the professional edition. It provides a wonderful user experience with smart assistance that provides intelligent code completion and easy project navigation [6].

- **Android Studio** it is the official IDE for the Android operating system managed by Google, it was built based on JetBrains's product Intellij IDEA, it works on Windows, macOS and Linux. Android Studio was first announced on May 16, 2013 by the Google I/O conference. It comes with a lot of features and support all the same programming languages of Intellij such as C++, Java and Kotlin and the user can integrate more languages through extensions.

4.2 Evaluation and discussion

In this section, we present the results of our system, discuss and evaluate them. Our binocular stereo vision system consists of a pair of webcams rigidly fixed with respect to each other, these cameras take photos simultaneously for a given object and this leads to obtain a 3D model from these photos. The image below shows the setup of our system.

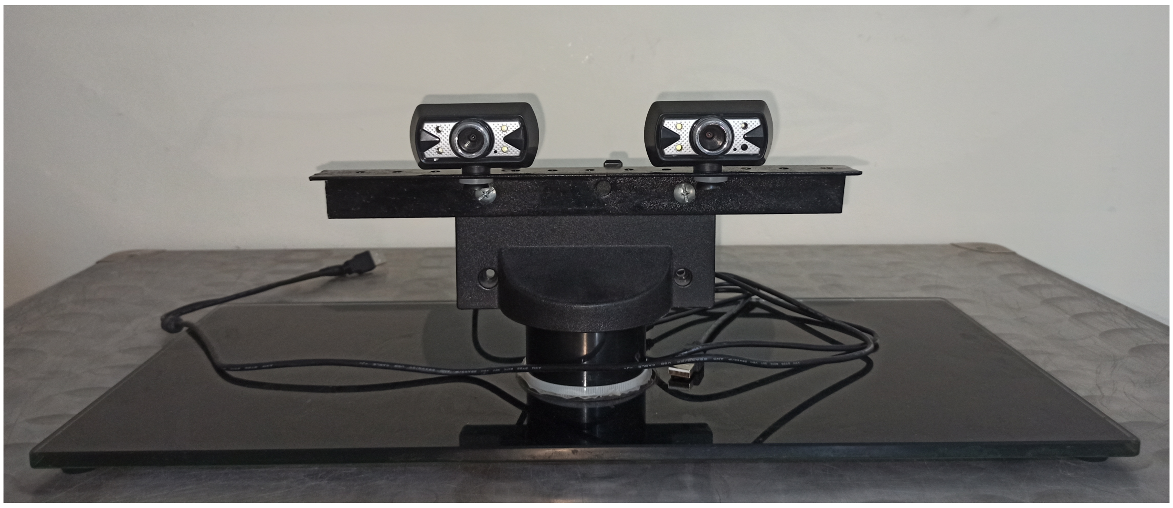
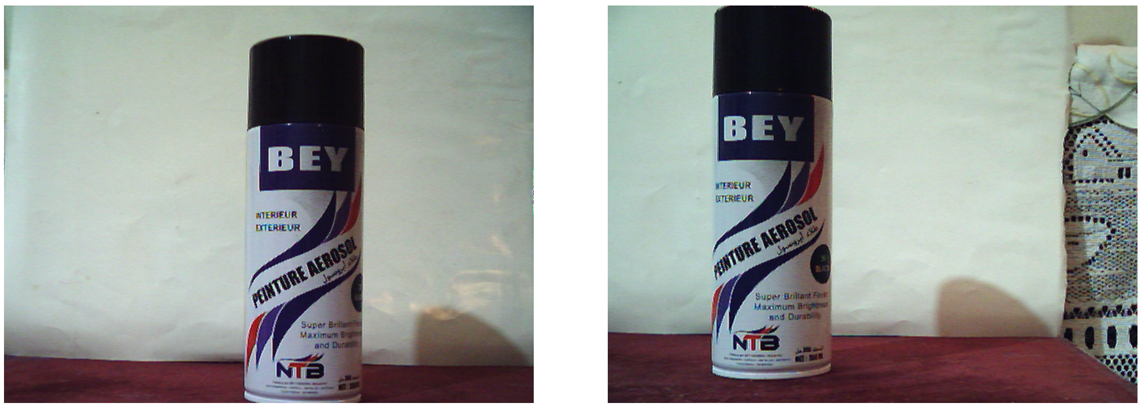


Figure 4.8: Our setup

This kind of projects needs a lot of experiments to reach a respectful results, the camera calibration is a one time operation, which means that it will be performed only once, it consumes time and efforts, taking a lot of checkerboard pattern photos is a good practise, we need to make sure that the photos are taken from different angles and different distances while we have a good lighting conditions. The cameras should be known and professional, the image taken by them will have high quality in which the final 3D model will be more than acceptable. As we mentioned before, this kind of projects needs a lot of experiments, one of the reasons for this is the **SGBM** algorithm, it should be tuned differently from one project to another, basing on results given by each project, its parameters will be optimised by observing the results each time and comparing between them. Another point should be considered is the distance between the cameras as well as the distance between the cameras the object being reconstructed.

We tested our system with an object (Bottle), this result is acceptable considering the used hardware, the left and right images are the following:



(a) Left image (b) Right image

Figure 4.9: Test images

We got the following results displayed on Meshlab:

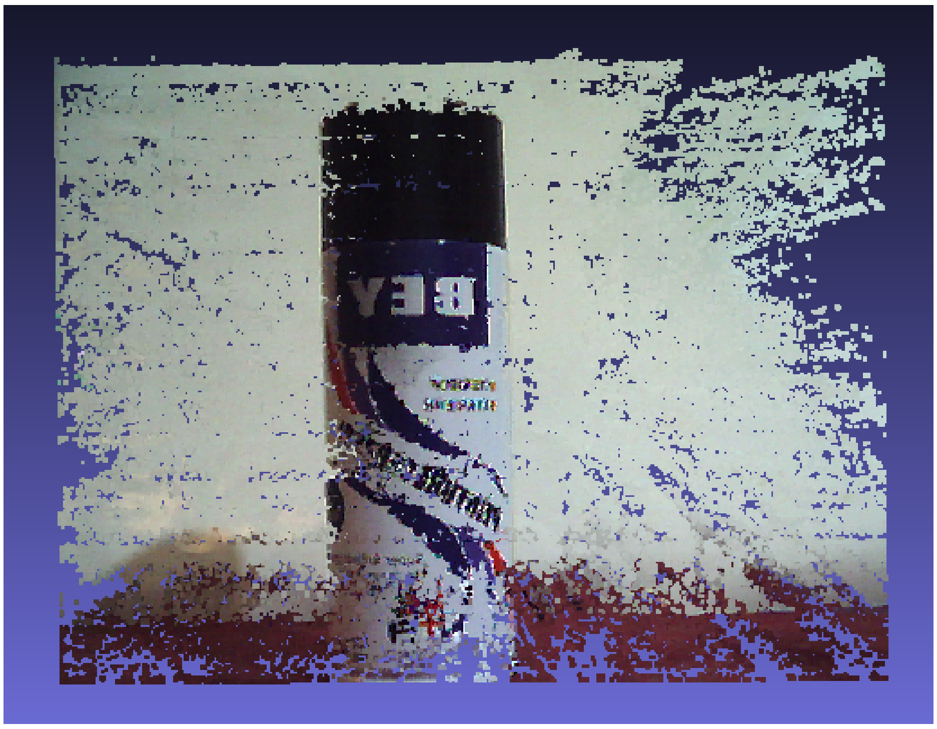


Figure 4.10: The front

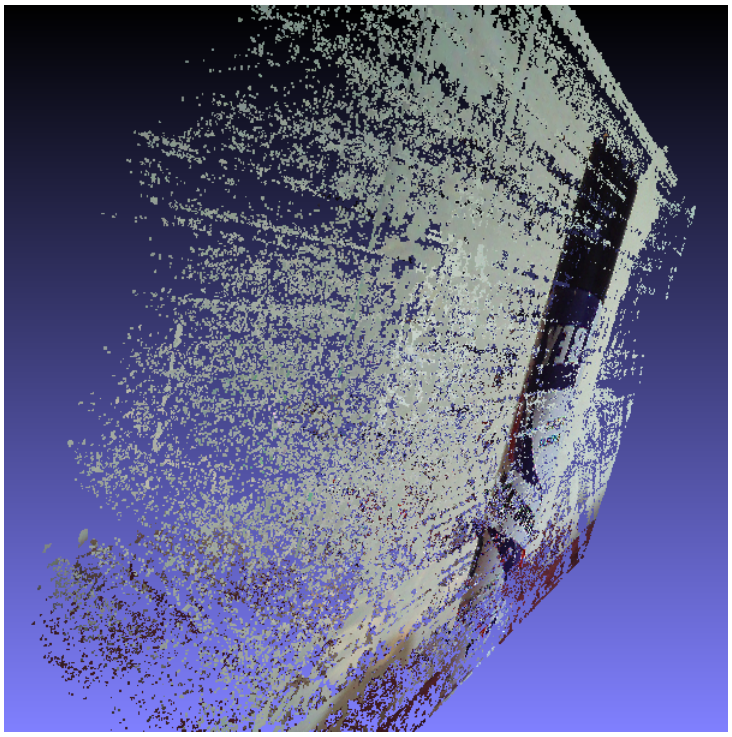


Figure 4.11: The left side

4.3 Conclusion

In this chapter we presented the used tools for the implementation part such programming languages, libraries and development environments, also, we discussed the system, the factors that affect the final 3D model and we presented and evaluated the results of this system.

General conclusion and future work

General conclusion

Computer vision-based systems in the 3D reconstruction context made a huge jump in multiple areas by enabling a significant reduction in production time and cost with less efforts, a lot of tasks became automated and the innovation ratio has been extremely increased. Three-dimensional reconstruction has always been a difficult goal, our work shows acceptable results, we have not done many experiments due to the hardware limitations and the absence of proper lighting conditions.

Future work

For future, we would try to provide the best conditions to reach better results and add many improvements to get the maximum accuracy possible. Many ideas for improving the system, such as:

- Background subtraction can reduce the computation time with less used resources.
- Using a raspberry pi would solve a lot of problems like the USB bus limitation in computers and make the calibration process easier.
- C++ is the more suitable programming language for this kind of projects, using it instead of Python could decrease the processing time which provides best real-time experience.
- A laser scanner is a great addition, the accuracy will be extremely increased.
- High-quality cameras are a must.

Bibliography

[1] Essential and fundamental matrices.

[2] Focal Length | Understanding Camera Zoom & Lens Focal Length | Nikon.

[3] OpenCV documentation v3.4.

[4] OpenCV documentation v4.5.2.

[5] Opencv official website.

[6] PyCharm: the Python IDE for Professional Developers by JetBrains.

[7] Understanding LSTM Networks – colah's blog.

[8] A computational theory of human stereo vision. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 204(1156):301–328, May 1979.

[9] RNN, LSTM & GRU, April 2019.

[10] What Is Focal Length in Photography?, April 2019.

[11] Convolutional Neural Network | Deep Learning, August 2020.

[12] Geometry of Image Formation | Learn OpenCV, February 2020.

[13] Louise Behiel. Given that you have one, do you know how it works?

[14] Youssef Bokhabrine, Ralph Seulin, Patrick Gorria, Gouenou Girardin, Miguel Gómez, and Daniel Jobard. 3d characterization of hot metallic shells during industrial forging. *Machine Vision and Applications - MVA*, 23, 05 2012.

[15] Kong-man Cheung, Simon Baker, and Takeo Kanade. Shape-from-silhouette across time part i: Theory and algorithms. *International Journal of Computer Vision*, 62:221–247, 05 2005.

[16] Kui Fu, Jiansheng Peng, Qiwen He, and Hanxiao Zhang. Single image 3D object reconstruction based on deep learning: A review. *Multimedia Tools and Applications*, 80(1):463–498, January 2021.

[17] Christos Georgoulas, Georgios Ch., and Ioannis Andreadis. Real-Time Stereo Vision Applications. In Ales Ude, editor, *Robot Vision*. InTech, March 2010.

[18] Stavros Hadjitheophanous, Christos Ttofis, A. Georghiades, and Theo Theocharides. *Towards hardware stereoscopic 3D reconstruction: A real-time FPGA computation of the disparity map.* 03 2010.

[19] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. Image-Based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(5):1578–1604, May 2021.

[20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[21] Alexander G. Hayes, P. Corlies, C. Tate, M. Barrington, J. F. Bell, J. N. Maki, M. Caplinger, M. Ravine, K. M. Kinch, K. Herkenhoff, B. Horgan, J. Johnson, M. Lemmon, G. Paar, M. S. Rice, E. Jensen, T. M. Kubacki, E. Cloutis, R. Deen, B. L. Ehlmann, E. Lakdawalla, R. Sullivan, A. Winhold, A. Parkinson, Z. Bailey, J. van Beek, P. Caballo-Perucha, E. Cisneros, D. Dixon, C. Donaldson, O. B. Jensen, J. Kuik, K. Lapo, A. Magee, M. Merusi, J. Mollerup, N. Scudder, C. Seeger, E. Stanish, M. Starr, M. Thompson, N. Turenne, and K. Winchell. Pre-Flight Calibration of the Mars 2020 Rover Mastcam Zoom (Mastcam-Z) Multispectral, Stereoscopic Imager. *Space Science Reviews*, 217(2):29, March 2021.

[22] Yu He and Shengyong Chen. Advances in sensing and processing methods for three-dimensional robot vision. *International Journal of Advanced Robotic Systems*, 15:172988141876062, 03 2018.

[23] Xiaoshui Huang, Guofeng Mei, J. Zhang, and Rana Abbas. A comprehensive survey on point cloud registration. *ArXiv*, abs/2103.02690, 2021.

[24] Jun Jiang, Liangcai Zeng, Bin Chen, Yang Lu, and Wei Xiong. An accurate and flexible technique for camera calibration. *Computing*, 101(12):1971–1988, December 2019.

[25] Adrian Kaehler and Gary Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 1st edition, 2016.

[26] Adrian Kaehler and Gary R. Bradski. *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. O'Reilly Media, Sebastopol, CA, first edition, second release edition, 2017. OCLC: ocn968936315.

[27] Sadekar Kaustubh. Introduction to Epipolar Geometry and Stereo Vision, December 2020.

[28] Keishi Koyama, Masayuki Takakura, T. Furukawa, and S. Maruo. 3d shape reconstruction of 3d printed transparent microscopic objects from multiple photographic images using ultraviolet illumination. *Micromachines*, 9, 2018.

[29] Dongdong Li, Hui Bingwei, Shaohua Qiu, and Wen Gongjian. 3d reconstruction with two webcams and a laser line projector. 9282, 09 2014.

[30] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943.

[31] Deshpande Mohit. Perceptrons: The First Neural Networks, September 2020.

[32] Theo Moons, Luc Van Gool, and Maarten Vergauwen. 3d reconstruction from multiple images: Part 1 - principles. *Foundations and Trends in Computer Graphics and Vision*, 4:287–404, 01 2009.

[33] Weizhi Nie, Kun Wang, Hongtao Wang, and Yuting Su. The assessment of 3d model representation for retrieval with cnn-rnn networks. *Multimedia Tools and Applications*, 78, 06 2019.

[34] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. 12 2020.

[35] val pertran. *Binocular depth perception, probability, fuzzy logic, and continuous quantification of uniqueness*. PhD thesis, Case Western Reserve University, 2018.

[36] E. Prados and O. Faugeras. Shape From Shading. In Nikos Paragios, Yunmei Chen, and Olivier Faugeras, editors, *Handbook of Mathematical Models in Computer Vision*, pages 375–388. Springer-Verlag, New York, 2006.

[37] Olgierd Stankiewicz, Gauthier Lafruit, and Marek Domański. Chapter 1 - multiview video: Acquisition, processing, compression, and virtual view rendering. In Rama Chellappa and Sergios Theodoridis, editors, *Academic Press Library in Signal Processing, Volume 6*, pages 3–74. Academic Press, 2018.

[38] Rocio Vargas, Amir Mosavi, and Ramon Ruiz. Deep learning: A review. *Advances in Intelligent Systems and Computing*, 5, 06 2017.

[39] Savvas Varsamopoulos, Koen Bertels, and Carmen Almudever. Designing neural network based decoders for surface codes. 11 2018.

[40] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.