Politecnico di Bari

Department of Electrical and Information Engineering
Bachelor Degree in Computer and Automation Engineering

Dissertation in Digital Control

Calibration and trajectory estimation of a stereo camera using OpenCV and ORB-SLAM3

Supervisor:
Prof. Paolo Lino

Co-supervisor:
Nikolai Svishchev

Candidate
Mattia Tritto

Academic Year 2022 - 2023

Index

Index of figures

Chapter 1 - Introduction

1.1 - An overview of autonomous underwater vehicles

Autonomous underwater vehicles (AUVs) are an essential tool for exploring the vast and complex ocean environment. These vehicles can perform various tasks, such as oceanographic surveys, environmental monitoring, and search and rescue operations. However, the success of these tasks depends on the vehicle's ability to navigate autonomously and accurately in the underwater environment.

Unlike land-based robots, AUVs face significant challenges in underwater navigation, such as limited communication capabilities, harsh environmental conditions, and the complexity of the underwater environment. The development of advanced navigation techniques for AUVs is crucial to overcome these challenges and enable efficient and safe operations.

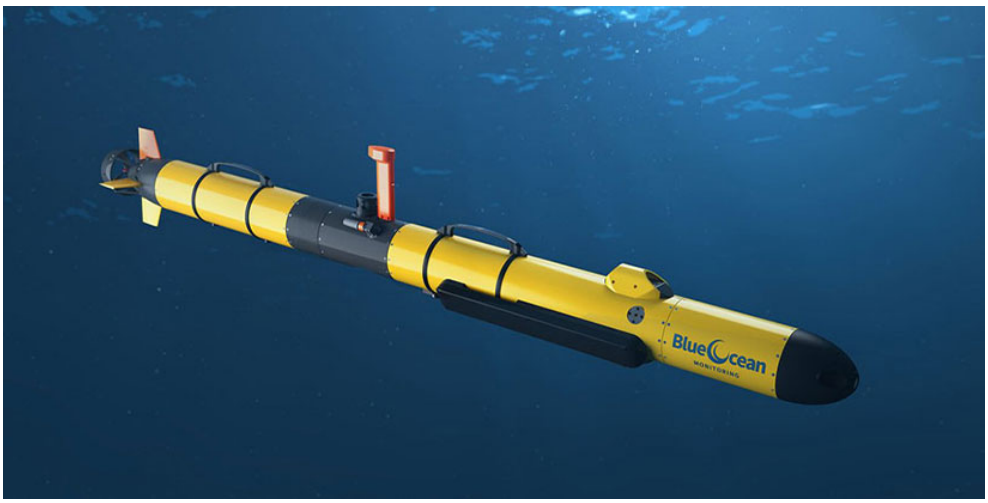


Figure 1: An example of AUV, credits: https://www.blueoceanmts.com/news/blog-post-three-tak7s

Simultaneous localization and mapping (SLAM) is a popular navigation technique that uses onboard sensors to create a map of the vehicle's surroundings and estimate its position relative to the map. However, the accuracy of SLAM is affected by various factors, such as sensor noise,

environmental conditions, and the presence of obstacles. Thus, there is a need to develop robust SLAM algorithms that can handle these challenges.

Path planning is another essential component of underwater autonomous navigation. AUVs need to plan their paths to optimize their mission objectives while avoiding obstacles and minimizing energy consumption.

This thesis focuses on the calibration process of stereo cameras for underwater autonomous navigation, and it consists of the following chapters:

- chapter 1 provides a brief introduction to Autonomous Underwater Vehicles (AUVs) and highlights the significance of stereo cameras in maritime environments. Emphasis is placed on the importance of camera calibration in the context of underwater autonomous navigation;
- chapter 2 explains the process of performing a simple monocular calibration. This serves as a fundamental step in understanding stereo calibration;
- chapter 3 explores the reconstruction of a 3D scene using two identical cameras that have already been calibrated. This chapter outlines the methodology and steps involved in achieving accurate 3D reconstruction;
- chapter 4 focuses on reconstructing a 3D scene using two identical cameras that are not calibrated. The objective is to calibrate these cameras and follow the steps described in Chapter 3 for accurate reconstruction;
- chapter 5 focuses on explaining the code for stereo reconstruction, the installation of a docker-version of ORB-SLAM3 and how to perform a trajectory estimation using Python.

The research will continue on improving the accuracy and reliability of the stereo calibration in the underwater environment and evaluating their performance through simulations and field experiments.

1.2 - Use cases of stereocameras underwater

Highly constrained patterns of stereo photographs can be used to automatically generate a detailed 3D model of a site, shipwreck or artefacts. Based on citations in the literature, underwater camera systems are now

widely employed in preference to manual methods as a non-contact, non-invasive technique to capture accurate length information and thereby estimate biomass or population fish distributions.

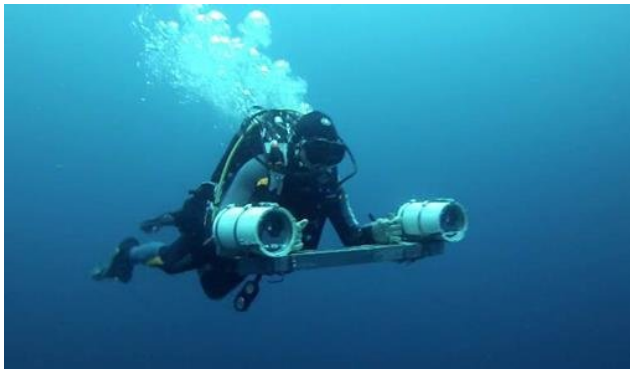


Figure 2: Stereocamera guided by a professional diver, credits: https://www.livingoceansfoundation.org/profile/jfreund/

There are many other applications of underwater photogrammetry. Stereo camera systems were used to conduct the first accurate seabed mapping applications and have been used to measure the growth of coral. Single and stereo cameras have been used for monitoring of submarine structures, most notably to support energy exploration and extraction in the North Sea, 3D models of sea grass meadows and inshore sea floor mapping.

1.3 - What is the purpose of camera calibration

Calibration of any camera system is essential to achieve accurate and reliable measurements. Small errors in the perspective projection must be modelled and eliminated to prevent the introduction of systematic errors in the measurements. In the underwater environment, the calibration of the cameras is of even greater importance because the effects of refraction through the air, housing and water interfaces must be incorporated.

The common factor for all these applications of underwater use cases is a specified level of accuracy. Photogrammetric surveys for heritage recording, marine biomass or fish population distributions are directly dependent on the accuracy of the 3D measurements. Any inaccuracy will lead to significant

errors in the measured dimensions of artefacts, under or over estimation of biomass or a systematic bias in the fish population distribution.

1.4 - Monocular cameras

Monocular cameras, also known as single-lens cameras, are cameras that use a single lens to capture images or videos. Unlike stereo cameras, which use two lenses to create a three-dimensional image, monocular cameras capture a 2D image that represents a flat projection of the scene.

Monocular cameras are commonly used in smartphones, digital cameras, and other consumer electronics. They are also used in robotics, autonomous vehicles, and other applications that require visual sensing.

One of the advantages of monocular cameras is their simplicity and cost-effectiveness. They require fewer components than stereo cameras and can be manufactured at a lower cost. Monocular cameras also have a smaller form factor, making them ideal for applications where space is limited.

However, one of the limitations of monocular cameras is their inability to capture depth information directly. Without depth information, it can be difficult to accurately estimate the distance between objects in the scene. This can limit their use in applications that require precise distance measurement, such as 3D modelling, augmented reality, and autonomous navigation.

To overcome this limitation, monocular cameras can be paired with other sensors, such as LiDAR or depth sensors, to capture depth information.

Figure 3: an example of a monocular camera that can be attached on a Raspberry PI (OV9281-160), credits: https://it.aliexpress.com/item/1005003962468246.html

1.5 - Binocular cameras

Binocular cameras, also known as stereo cameras or depth cameras, use two or more cameras to capture stereo images that can be used to create a 3D representation of the scene. The cameras are typically positioned a short distance apart, mimicking the way that our eyes capture images from slightly different viewpoints.

The two cameras capture images simultaneously, and the resulting stereo image pair can be used to calculate the depth information of the scene. The depth information is calculated by analysing the differences between the two images, such as disparities in position or texture. The greater the difference between the two images, the closer the object is to the cameras.

Stereo cameras have a wide range of applications, from computer vision and robotics to entertainment and virtual reality. They are commonly used in 3D scanning and modelling, object recognition, tracking, and autonomous navigation.
One of the advantages of stereo cameras is their ability to capture depth information using computer vision techniques. This makes them highly accurate and reliable in applications that require precise depth measurement.

However, stereo cameras can be more complex and expensive than monocular cameras, as they require multiple lenses and sensors. They can also be more difficult to calibrate, as the cameras must be precisely aligned and synchronised to ensure accurate depth calculation.¶

Figure 4: an example of a stereo camera (Intel RealSense D455), credits: https://www.amazon.it/Intel-Fotocamera-RealSense-profondit%C3%A0-D455/dp/B08HHHDRNM

1.6 - The mathematical model of thin lenses

The model of thin lenses is a simplified mathematical representation of how lenses work. It is based on the assumption that lenses are thin and can be modelled as a single curved surface with a constant thickness. The model is widely used in optics and is essential for understanding how light is refracted through lenses and how images are formed.

According to the model, a thin lens can be represented by a single curved surface with a focal point and a focal length. The focal point is the point on the optical axis where parallel rays of light converge after passing through the lens. The focal length is the distance between the center of the lens and the focal point.

The model also includes the lens equation, which relates the distance of an object from the lens, the distance of the image from the lens, and the focal length of the lens. The lens equation can be expressed as:

$$\frac{1}{f} = \frac{1}{p} + \frac{1}{q}$$

where f is the focal length, p is the object distance, and q is the image distance. This equation is used to calculate the position and size of the image formed by a lens.

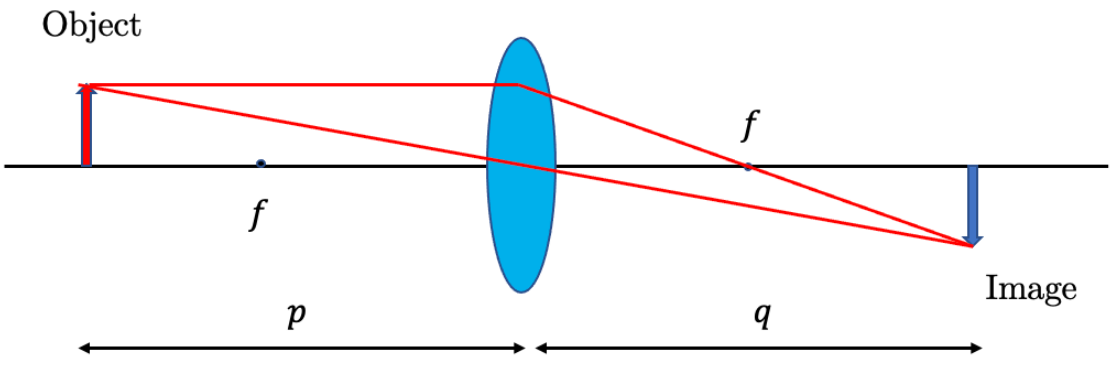


Figure 5: representation of the thin lenses model

The model of thin lenses is useful for understanding the basic principles of optics and for designing and analysing optical systems. It is also used in the design of eyeglasses, cameras, telescopes, and other optical instruments.

However, it is important to note that the thin lens model is a simplified representation of how lenses work and does not account for all the factors that can affect the behaviour of light, such as lens aberrations and the dispersion of light. These factors must be taken into account in more complex optical systems and designs.

1.7 - Types of distortion introduced by real lenses

Barrel and pincushion distortion are two types of radial distortion that can occur in photography. Each type of distortion affects the image differently and can be caused by different factors.

Barrel distortion is a type of radial distortion where straight lines near the edges of an image appear to bend outwards, creating a barrel shape. This type of distortion is commonly seen in wide-angle lenses, where the angle of view is wider than that of the human eye. It can also occur in images that have been digitally processed or scanned. Barrel distortion can be corrected using specialized software or by adjusting the lens settings.

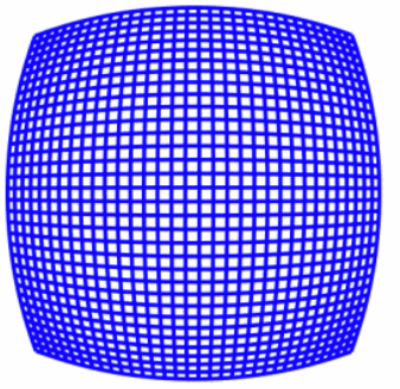


Figure 6: example of the barrel distortion, credits: https://learnopencv.com/understanding-lens-distortion/

Pincushion distortion, on the other hand, is the opposite of barrel distortion. It causes straight lines near the edges of an image to bend inwards, creating a pincushion shape. This type of distortion is often seen in telephoto lenses, where the angle of view is narrower than that of the human eye. Pincushion distortion can also be corrected using specialized software or by adjusting the lens settings.

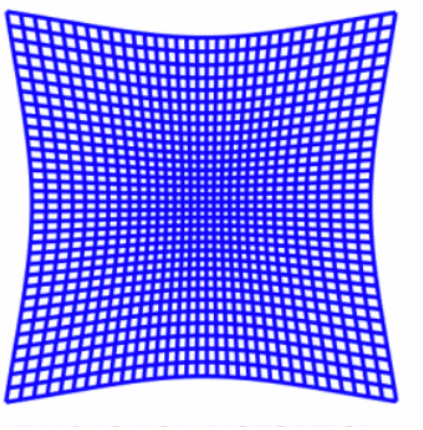


Figure 7: example of the pincushion distortion, credits: https://learnopencv.com/understanding-lens-distortion/

Tangential distortion is a type of lens distortion that causes straight lines near the edges of an image to appear curved. Unlike barrel distortion and pincushion distortion, which cause the entire image to bend outward or inward, tangential distortion creates a wavy pattern near the edges of the image.

Tangential distortion occurs when the lens elements are not perfectly aligned with the image sensor or film plane. This can cause some areas of the image to be in sharper focus than others, resulting in distortion. The distortion is more pronounced in wide-angle lenses, which have a wider field of view and therefore capture more of the scene.¶

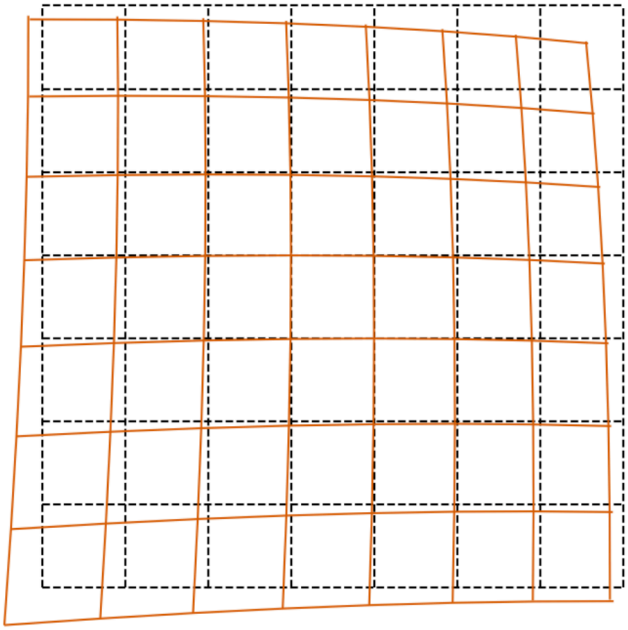


Figure 8: example of tangential distortion, credits: https://learnopencv.com/understanding-lens-distortion/

Chapter 2 - Monocular calibration using a linear camera model

2.1 - Overview of the calibration process

Suppose we have a 2D image. The main goal of camera calibration is to correctly project the 2D image points into the 3D world points. To do this we need:

1) position and orientation of the camera compared to the 3D world coordinates system (these are called **external parameters** of the camera);
2) parameters that belong to the camera used to take the photo, such as the focal length (these are called **internal parameters** of the camera).

Calibrating a camera means finding the internal and external parameters. Before trying to estimate internal and external parameters, we need a camera model. In our case, we use a simple linear camera model that is computationally simple to compute. The model is a single matrix called the projection matrix P.

2.2 - The 2D and 3D coordinate systems and the relationship between these three coordinate systems

Before going into the details of calibration, we first need to define three coordinate systems.

The first one is the **world coordinate system** (or 3D coordinate system). Every point in the 3D world is measured based on where we placed the origin of $\mathbb{W}$.

The second one is the **camera coordinate system** $\mathbb{C}$ (3D coordinate system). The Z axis of the camera coordinate frame is aligned with the optical axis of the camera. The origin of $\mathbb{C}$ is where our camera lies.

The third one is the **image coordinate system** (2D coordinate system).

Every point in the 3D-world is represented in the $\mathbf{x}_W$ vector:

$$\mathbf{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

In the 2D-image coordinate system, the same point is described in the $\mathbf{u}$ vector:

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

If we know the relative position and orientation of the camera coordinate frame with respect to the world coordinate frame then we can write an expression that takes all the way from the point P in $\mathbb{W}$ to its projection (u, v) on the image plane.

$$\mathbf{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \longrightarrow \mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \longrightarrow \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

World coordinates Camera coordinates Image coordinates

For each corresponding point i in scene and image, we have this correspondence:

$$\begin{bmatrix} u^{(i)} \\ y^{(i)} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w^{(i)} \\ y_w^{(i)} \\ z_w^{(i)} \\ 1 \end{bmatrix}$$

The known terms are the first vector and the last vector. We don't know the projection matrix P.

If we expand the matrix as linear equations we obtain:

$$u^{(i)} = \frac{p_{11}x_w^{(i)} + p_{12}y_w^{(i)} + p_{13}z_w^{(i)} + p_{14}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

$$v^{(i)} = \frac{p_{21}x_w^{(i)} + p_{22}y_w^{(i)} + p_{23}z_w^{(i)} + p_{24}}{p_{31}x_w^{(i)} + p_{32}y_w^{(i)} + p_{33}z_w^{(i)} + p_{34}}$$

If we re-arrange the terms in order to have a matrix A with known elements and a vector $\mathbf{p}$ with unknown elements we obtain:

$$\begin{bmatrix}
x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & 0 & 0 & 0 & 0 & -u_1x_w^{(1)} & -u_1y_w^{(1)} & -u_1z_w^{(1)} & -u_1 \\
0 & 0 & 0 & 0 & x_w^{(1)} & y_w^{(1)} & z_w^{(1)} & 1 & -v_1x_w^{(1)} & -v_1y_w^{(1)} & -v_1z_w^{(1)} & -v_1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & 0 & 0 & 0 & 0 & -u_ix_w^{(i)} & -u_iy_w^{(i)} & -u_iz_w^{(i)} & -u_i \\
0 & 0 & 0 & 0 & x_w^{(i)} & y_w^{(i)} & z_w^{(i)} & 1 & -v_ix_w^{(i)} & -v_iy_w^{(i)} & -v_iz_w^{(i)} & -v_i \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & 0 & 0 & 0 & 0 & -u_nx_w^{(n)} & -u_ny_w^{(n)} & -u_nz_w^{(n)} & -u_n \\
0 & 0 & 0 & 0 & x_w^{(n)} & y_w^{(n)} & z_w^{(n)} & 1 & -v_nx_w^{(n)} & -v_ny_w^{(n)} & -v_nz_w^{(n)} & -v_n
\end{bmatrix}
\begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{14} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{24} \\ p_{31} \\ p_{32} \\ p_{33} \\ p_{34} \end{bmatrix}
=
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In a compact form:

$$A\mathbf{p} = \mathbf{0}$$

Where $\mathbf{p}$ is the projection matrix P written as a vector.

2.3 - The scale propriety of p and how do we choose the scale

An important propriety of **p** is the scale propriety. This means that the projection matrix acts on homogeneous coordinates:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv k \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix}$$

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \equiv k \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

Where k is a constant, $k \neq 0 \in \mathbb{R}$. Therefore, projection matrices P and kP produce the same homogenous pixel coordinates (u, v). *The projection matrix P is defined only up to a scale.*

In other words, scaling the projection matrix implies simultaneously scaling the world and the camera, which does not change the image.

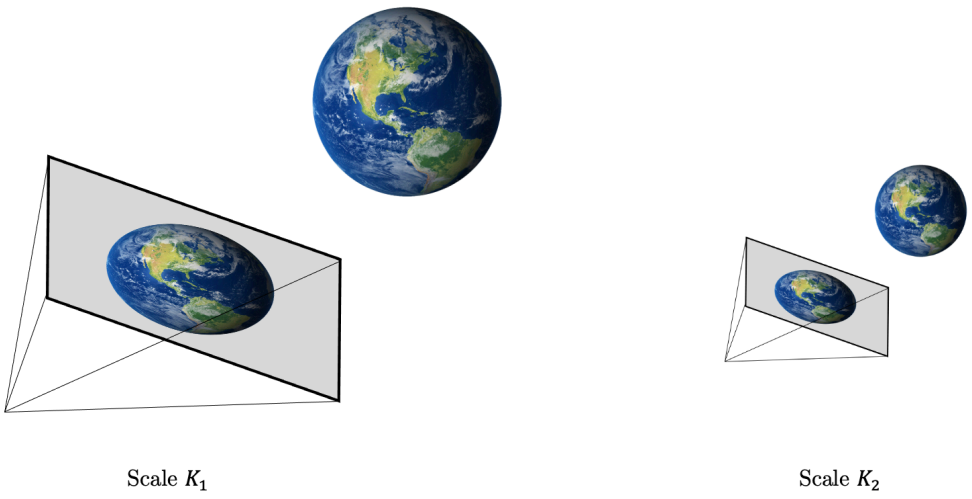


Figure 9: Example of scaling the projection matrix P

But how we choose the scale of P? We have two options:

1) set the scale so that one of the 12 elements of the projection matrix P is equal to one ($p_{ij} = 1$);
2) set the scale so that $\|\mathbf{p}\|^2 = 1$.

In this case, we're choosing the second option.

We want $A\mathbf{p}$ as close to 0 as possible, and $\|\mathbf{p}\|^2 = 1$. In mathematical terms:

$$\min_p \|A\mathbf{p}\|^2 \text{ such that } \|\mathbf{p}\|^2 = 1$$

Can be rewritten as:

$$\min_p(\mathbf{p}^T A^T A\mathbf{p}) \text{ such that } \mathbf{p}^T\mathbf{p} = 1$$

This kind of problem is called the **constraint least squares problem**.

We can define a loss function $L(\mathbf{p}, \lambda)$ as follows:

$$L(\mathbf{p}, \lambda) = \mathbf{p}^T A^T A\mathbf{p} - \lambda(\mathbf{p}^T\mathbf{p} - 1)$$

The goal is to minimize this function. We calculate the derivatives of $L(\mathbf{p}, \lambda)$ with respect to $\mathbf{p}$ and we place equal to 0:

$$2A^T A\mathbf{p} - 2\lambda\mathbf{p} = \mathbf{0}$$

This is equivalent to resolve this eigenvalue problem:

$$A^T A\mathbf{p} = \lambda\mathbf{p}$$

In other words, the $\mathbf{p}$ that we're looking for is the smallest eigenvalue λ of the matrix $A^T A$ that minimize the loss function $L(\mathbf{p})$. Once we have $\mathbf{p}$, we re-arrange the elements of $\mathbf{p}$ to form the projection matrix P.

2.4 - Decomposing the projection matrix P

From the projection matrix P, we can decompose it into the intrinsic matrix M_{int} and the extrinsic matrix M_{ext} (to isolate the intrinsic and extrinsic parameters):

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let's consider this $3x3$ sub-matrix $\tilde{P}$:

$$\tilde{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix}$$

The goal is to find to matrices K and R so that:

$$\tilde{P} = KR$$

K is the calibration matrix, which contains all the intrinsic parameters, and R is the rotation matrix. Given that K is an upper-right triangular matrix and R is an orthonormal matrix, it is possible to uniquely decouple K and R using QR decomposition:

$$\tilde{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = KR$$

To find the translation vector $\mathbf{t}$, let's extract the fourth column of the projection matrix P. This is equal to:

$$\begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = K\mathbf{t}$$

If we isolate **t**:

$$\mathbf{t} = K^{-1} \begin{bmatrix} p_{14} \\ p_{24} \\ p_{34} \end{bmatrix}$$

2.5 - From camera coordinates to image coordinates: perspective projection

$$\mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \longrightarrow \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

Camera coordinates Image coordinates

Let's assume for now that we know a point in the camera coordinate frame. The focal length (the distance between the central projection and the image plane of the camera) is f. Based on simple optical equations, we know that the point $\mathbf{x}_i$ in the image coordinate frame is defined as:

$$\begin{cases} x_i = \frac{x_c}{z_c} f \\ y_i = \frac{y_c}{z_c} f \end{cases}$$

Let's take a closer look at the image plane. In our case, the image plane is a sensor which is used to capture the image. The image sensor has pixels. We introduce pixel coordinates (u, v) in the image sensor plane. The goal is to figure out how we can transform the image coordinates in millimetres to pixels. If m_x and m_y are pixel densities in x and y directions, then pixel coordinates (u, v) are:

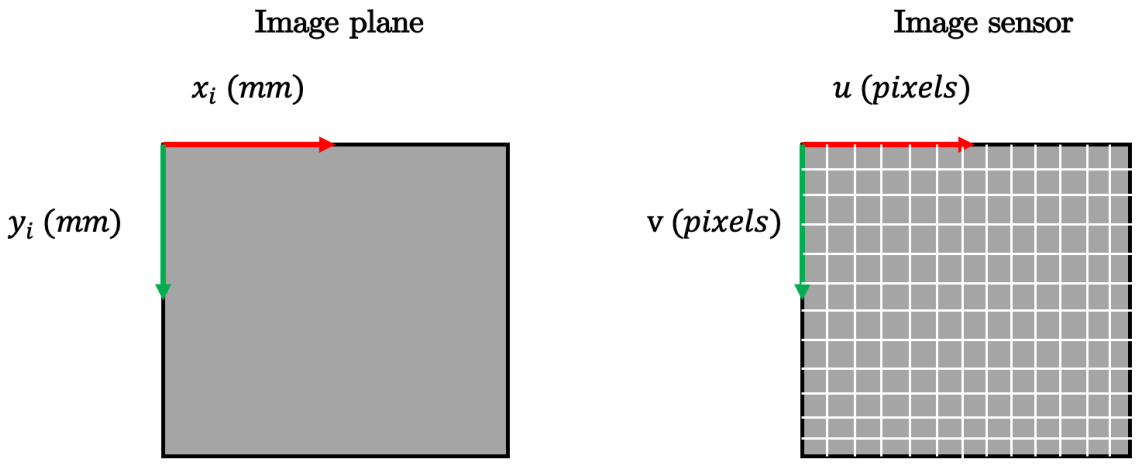


Figure 10: Representation of the image plane and the image sensor

$$\begin{cases} u = m_x x_i = m_x f \frac{x_c}{z_c} \\ v = m_y y_i = m_y f \frac{y_c}{z_c} \end{cases}$$

The top-left corner of the image sensor is its origin. If the pixel (o_x, o_y) is the principle point (where the optical axis pierces the sensor), then:

$$\begin{cases} u = m_x x_i = m_x f \frac{x_c}{z_c} + o_x \\ v = m_y y_i = m_y f \frac{y_c}{z_c} + o_y \end{cases}$$

Of course m_x, m_y and f are unknown and they're part of the calibration process. We combine these together in f_x and f_y (focal lengths in pixels in the x and y directions):

$$\begin{cases} u = m_x x_i = f_x \frac{x_c}{z_c} + o_x \\ v = m_y y_i = f_y \frac{y_c}{z_c} + o_y \end{cases}$$

(f_x, f_y, o_x, o_y) are the **intrinsic parameters of the camera**. They represent the camera's internal geometry. The equations that we have found are non-linear equations. We use homogenous coordinates to extract linear equations.

We can express (u, v) in homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix}$$

If we rearrange this equations by using the intrinsic matrix and a vector containing the homogeneous coordinates of the 3D point in the camera coordinate frame:

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

In a more compact form:

$$\tilde{\mathbf{u}} = M_{int} \tilde{\mathbf{x}}_c = [K \,|\, \mathbf{0}] \tilde{\mathbf{x}}_c$$

2.6 - From world coordinates to camera coordinates

$$\mathbf{x}_w = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \longrightarrow \mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

World coordinates Camera coordinates

Now we want to go from the world coordinates frame to the camera coordinates frame by using the position and orientation of the camera coordinate frame.

This can be done by using the position $\mathbf{c}_w$ and orientation R of the camera in the world coordinate frame $\mathbb{W}$. $\mathbf{c}_w$ and R are the camera's extrinsic parameters.

Given the extrinsic parameters (R, $\mathbf{c}_w$) of the camera, the camera-centric location of the point in the world coordinate frame is:

$$\mathbf{x}_c = R(\mathbf{x}_w - \mathbf{c}_w) = R\mathbf{x}_w + \mathbf{t}$$

Where the translation vector $\mathbf{t}$, is defined as follows:

$$\mathbf{t} = -R\mathbf{c}_w$$

If we expand this equation:

$$\mathbf{x}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Rearranging this equation using homogeneous coordinates results in:

$$\tilde{\mathbf{x}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

In a more compact form:

$$\tilde{\mathbf{x}}_c = M_{ext}\tilde{\mathbf{x}}_w$$

2.7 - From world coordinates to image coordinates: combining the intrinsic and extrinsic matrix

If we combine the two equations that we have found previously:

$$\begin{cases} \tilde{\mathbf{x}}_c = M_{ext}\tilde{\mathbf{x}}_w \\ \tilde{\mathbf{u}} = M_{int}\tilde{\mathbf{x}}_c \end{cases}$$

We obtain a single equation, that transforms the world coordinates in image coordinates using the projection matrix P:

$$\tilde{\mathbf{u}} = M_{int}M_{ext}\tilde{\mathbf{x}}_w = P\tilde{\mathbf{x}}_w$$

2.8 - Correction of radial and tangential distortion

After having estimated the intrinsic and extrinsic parameters of the camera, now we need to correct the radial and tangential distortion.

As we said in 1.5 paragraph, radial distortion causes straight lines to appear curved. Radial distortion becomes larger the farther points are from the center of the image. This type of distortion can be represented as follows:

$$\begin{cases} x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{cases}$$

Similarly, tangential distortion occurs because the image-taking lens is not aligned perfectly parallel to the imaging plane. So, some areas in the image may look nearer than expected. The amount of tangential distortion can be represented as below:

$$\begin{cases} x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \end{cases}$$

In short, we need to find five parameters, known as distortion coefficients given by:

$$(k_1, k_2, k_3, p_1, p_2)$$

Chapter 3 - Stereo reconstruction with calibrated cameras

3.1 - Why we use stereo cameras

Given a calibrated camera, we cannot find the 3D scene point from a single 2D image. A stereo system (a system of two cameras previously calibrated, displaced by a distance b) is a simple method for recovering the three dimensional structure of a scene from two images.

3.2 - Backward projection: from 2D to 3D

As we said earlier, given a calibrated camera, we cannot find the 3D scene point from a single 2D image. But we know that the corresponding 3D point must lie on an outgoing ray:

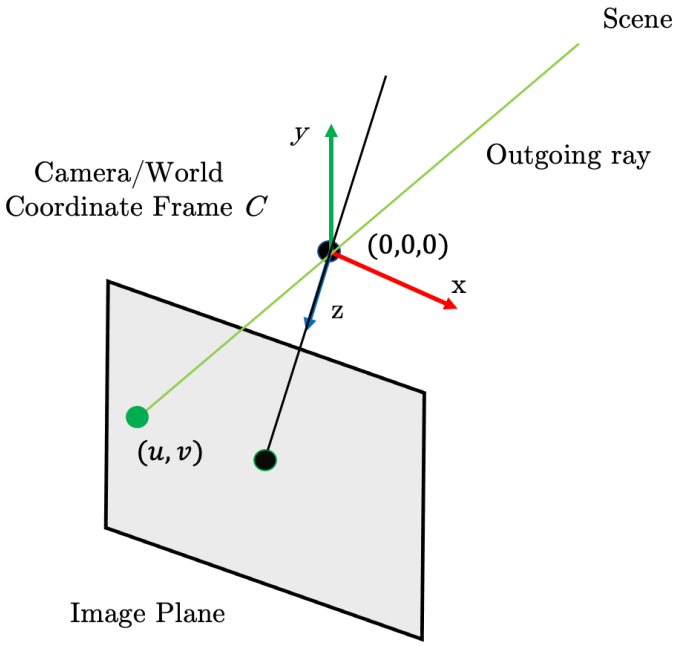


Figure 11: Representation of the ray where the 3D point lies

If we calibrate the camera with the steps provided in the first chapter, we're able to write the equation of this ray. The perspective equations are the following:

$$\begin{cases} u = f_x \frac{x_c}{z_c} + o_x \\ v = f_y \frac{y_c}{z_c} + o_y \end{cases}$$

The same equations can be used to figure out what the equation of the outgoing ray is, given a point (u, v) in the image:

$$\begin{cases} x = \frac{z}{f_x}(u - o_x) \\ y = \frac{z}{f_y}(v - o_y) \end{cases}$$

To reconstruct exactly where the point lies, we use another camera and we triangulate the position.

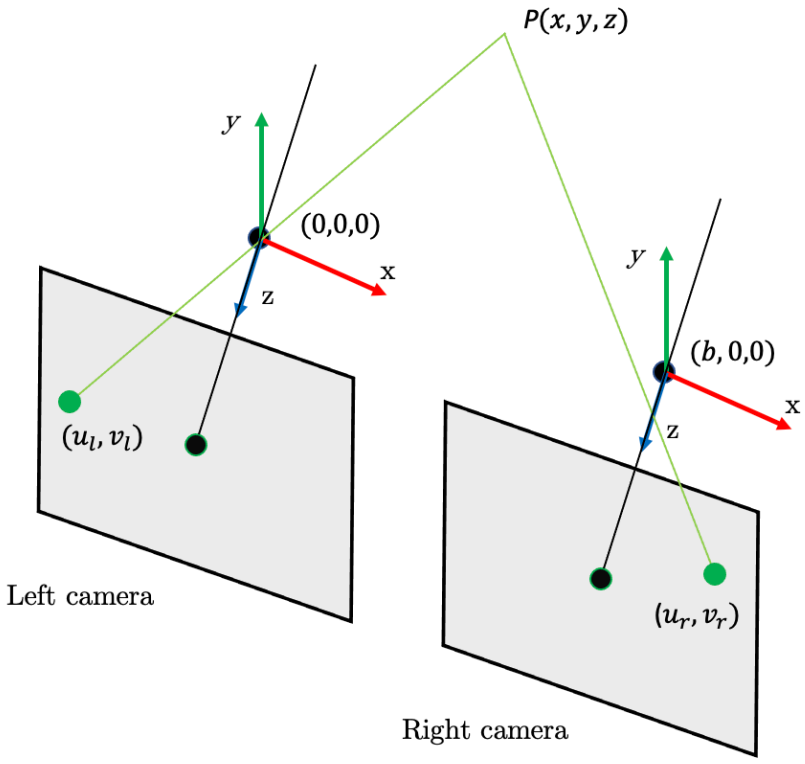


Figure 12: Example of triangulation using a pair of identical cameras

The left camera and the right camera are identical but displaced along the horizontal direction by a distance b. The distance b is called **baseline**. The system that include the two cameras is called a stereo system.

We're looking at one point in our left camera (u_l, v_l). That point corresponds to an outgoing ray. But let's say somehow we are able to find the corresponding point in the right camera (u_r, v_r) (the projection of the same scene point in the right camera). We can shoot out another outgoing ray from the right camera. Wherever those two rays intersect is where the scene point lies. For now, let's assume that the corresponding point in the right camera (u_r, v_r) is known for us (finding (u_r, v_r) given (u_l, v_l) is the stereo matching problem, the topic of the next paragraph).

We have four equations, that are the perspective projection equations for the left and right camera:

$$\begin{cases} u_l = f_x \frac{x}{z} + o_x \\ v_l = f_y \frac{y}{z} + o_y \end{cases} \qquad \begin{cases} u_r = f_x \frac{x-b}{z} + o_x \\ v_r = f_y \frac{y}{z} + o_y \end{cases}$$

By simply solving these four equations we get an equation for x, y and z:

$$\begin{cases} x = \frac{b(u_l - o_x)}{u_l - u_r} \\ y = \frac{b f_x (v_l - o_y)}{f_y (u_l - u_r)} \\ z = \frac{b f_x}{u_l - u_r} \end{cases}$$

The coordinate z is called the **depth** of the point in the scene.

The difference of the coordinates of the same scene point in the two images $u_l - u_r$ is called **disparity**.

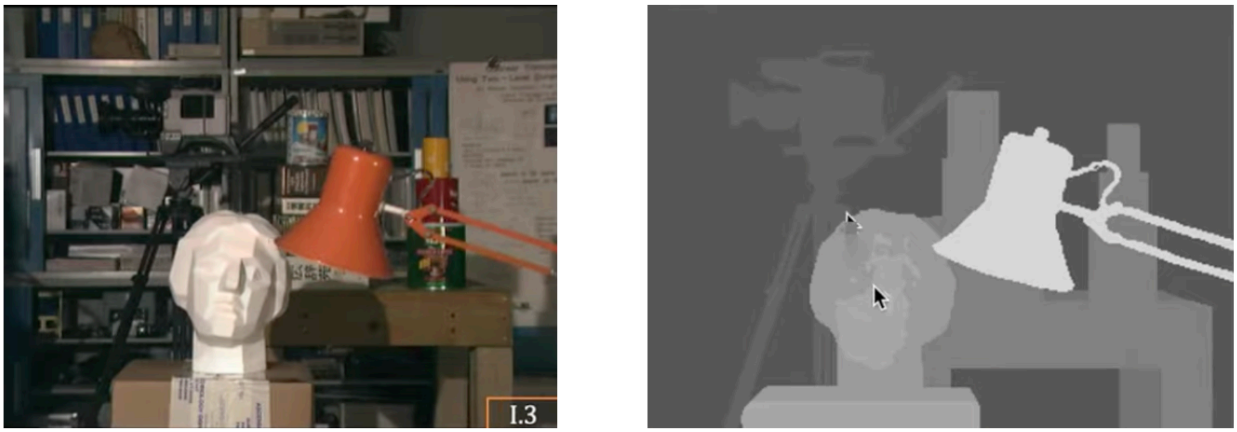
Depth is inversely proportional to disparity. If we have a scene at infinity and if you take two images of the scene doesn't really matter how far these cameras are with respect to each other (the baseline b), you're going to get

two identical images. As the scene gets closer and closer you're going to see differences in the projections.

We want to use a stereo configuration where the baseline is large, because the larger the baseline, the larger is the disparity, and this reduces the errors when we're trying to estimate the depth.

3.3 - Stereo matching: finding disparities

Measuring the disparity means estimating the depth of the scene. The goal is to find the disparity between left and right stereo pairs.



Left and right images

Disparity map (ground truth)

Figure 13: A disparity map of a particular scene, credits: https://www.researchgate.net/figure/Ground-truth-of-disparity-map-and-disparity-map-obtained-by-Belief-Propagation_fig6_224351635

On the right side we have the disparity map: the closer the points, the greater the disparity and the brighter it is in the disparity map.

In this example, the disparity in the y direction is 0. It means that corresponding points must lie on the same horizontal line in both the images. But how do we compute disparity? We use **template matching** to achieve our goal.

The first step is to take a little window on the left image, and then find the corresponding window in the right image. Of course we don't need to look for the corresponding point in the intere image, we know that the corresponding point in the right image must lie in the same horizontal scan line:

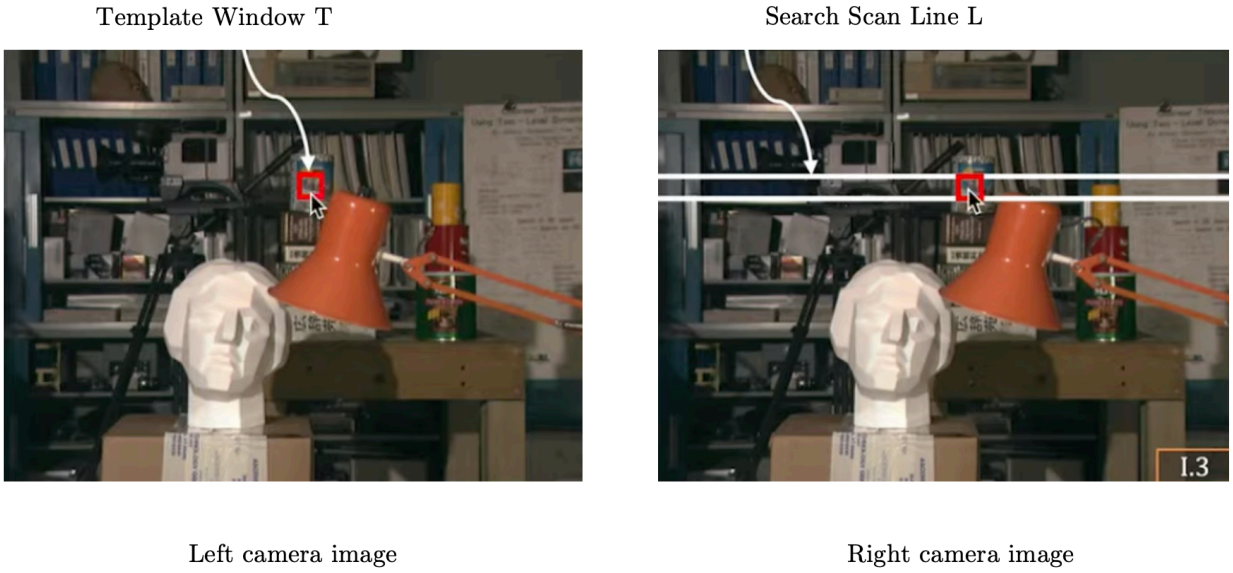


Figure 14: Representation of the template window in the left camera and the search line in the right camera, credits: https://www.researchgate.net/figure/Ground-truth-of-disparity-map-and-disparity-map-obtained-by-Belief-Propagation_fig6_224351635

After the matching, we can compute the disparity $u_l - u_r$ and the depth of the image:

$$z = \frac{bf_x}{u_l - u_r}$$

How large the windows should be? If the window is really small, we're going to get good localization but high sensitivity to noise. On the other hand, if we use larger windows, we're going to get more robust matches in terms of the depth of values but the disparity map is going to be more blurred especially at boundaries (poor localization).

Window Size = 5 pixels

(Sensitive to noise)

Window Size = 20 pixels

(Poor localization)

Figure 15: Differences between small and large windows sizes

Metrics used for template matching are:

1) finding the pixel $(k, l) \in L$ with minimum sum of absolute differences:

$$SAD(k,l) = \sum_{(i,j)\in T} |E_l(i,j) - E_r(i+k, j+l)|$$

2) finding the pixel $(k, l) \in L$ with minimum sum of squared differences:

$$SSD(k,l) = \sum_{(i,j)\in T} |E_l(i,j) - E_r(i+k, j+l)|^2$$

3) finding the pixel $(k, l) \in L$ with maximum normalised cross-correlation:

$$NCC(k,l) = \frac{\sum_{(i,j)\in T} E_l(i,j) E_r(i+k, j+l)}{\sqrt{\sum_{(i,j)\in T} E_l(i,j)^2 \sum_{(i,j)\in T} E_l(i+k, j+l)^2}}$$

Some problems of the stereo matching process are:

- surfaces used for stereo matching must have **non-repetitive texture** (if there aren't any texture there isn't any match, but if there are repetitive texture there are multiple matches, and the stereo matching is not unique);
- **foreshortening effect** makes matching really challenging. So often in stereo matching there are warping techniques to make the matching process more robust.

Chapter 4 - Stereo reconstruction with uncalibrated cameras

4.1 - Overview of the uncalibrated stereo case

Let's take two arbitrary pictures of a 3D scene. Of course we have no idea of where these picture were taken from with respect to each other. But it turns out that if we know the internal parameters of the two cameras, then from these two arbitrary views we can compute the translation and rotation of one camera with respect to another camera. And once that's done, we can reconstruct a three dimensional model of the scene. This is the problem of uncalibrated stereo.

The internal parameters of both cameras are generally known. Often they're available to us in terms of the method tag that goes with each image that's captured in modern day digital cameras. Or if we don't have these information, we can perform a simple calibration with the steps provided in the first chapter.

Next, we're going to use a small number of corresponding points in these two arbitrary views to estimate the fundamental matrix F. Once we have F, we can go ahead and find the rotation and translation of one camera with respect to each other. The stereo system is fully calibrated now.

The last step is to compute depth. In order to compute depth, we're going to find dense correspondences between the two images. Ideally, for every point in the left image, we want to know where the corresponding point in the right image is. This boils down to a 1D search in the right image using stereo matching.

4.2 - Epipolar geometry

In order to resolve this problem, we need to formulate the geometric relationship between the left and right camera. This is called epipolar geometry. Epipolar geometry tells us that points in the left and right image are related to each other through a single $3x3$ matrix called the **fundamental matrix**, which contains the rotation and translation of one camera with respect to each other. The calibration problem boils down to finding the fundamental matrix.

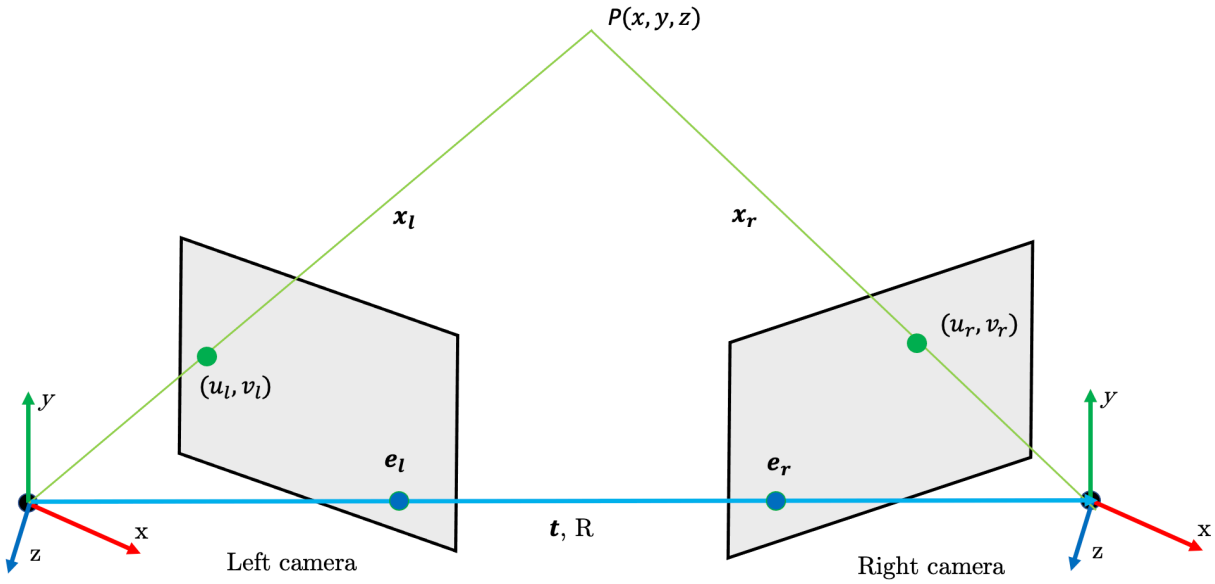


Figure 16: Representation of the epipoles and the epipolar plane

The projection of the center of the left camera into the right camera image and the projection of the center of the right camera on the left camera image these are referred to as the epipoles of the stereo system ($\mathbf{e}_l$ and $\mathbf{e}_r$). Any given stereo system, it has a unique pair of $\mathbf{e}_l$ and $\mathbf{e}_r$.

The epipolar plane of the scene point P is the plane formed by camera origins O_l and O_r, epipoles $\mathbf{e}_l$ and $\mathbf{e}_r$ and scene point P. So every scene point lies on a unique epipolar plane.

We're going to use the epipolar plane to set up what's called the epipolar constraint. This includes the parameters $\mathbf{t}$ and R.

Let's define a normal vector that is normal to this plane, defined as:

$$\mathbf{n} = \mathbf{t} \times \mathbf{x}_l$$

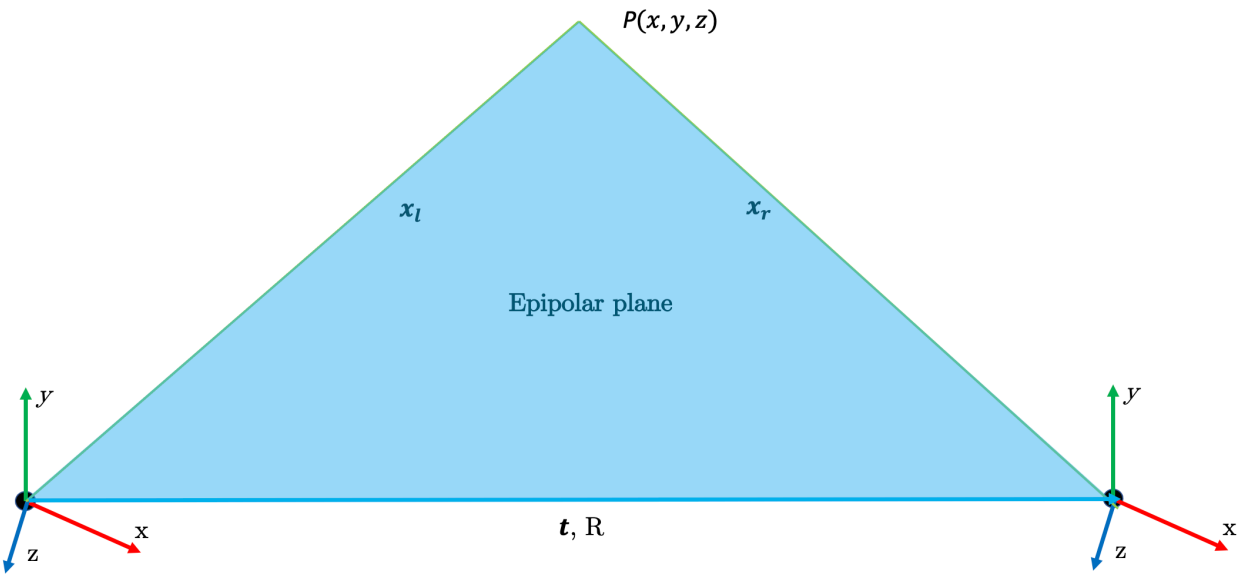


Figure 17: Representation of the epipolar plane

So the **epipolar constraint** is the dot product of $\mathbf{n}$ and $\mathbf{x}_l$, and this is equal to zero:

$$\mathbf{x}_l \cdot (\mathbf{t} \times \mathbf{x}_l) = 0$$

If we rewrite the epipolar constraint in matrix form we obtain:

$$\begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \begin{bmatrix} t_y z_l - t_z y_l \\ t_z x_l - t_x z_l \\ t_x y_l - t_y x_l \end{bmatrix} = 0$$

We can rearrange this equation, isolating in a single matrix the translation parameters t (translation matrix T_x):

$$[x_l \quad y_l \quad z_l] \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = 0$$

Now we add another constraint. We know $\mathbf{t}_{3\times1}$ is the position of the right camera in the left camera's frame and $R_{3\times3}$ is the orientation of the left camera in the right camera's frame. Using these two, we can relate the 3D coordinates of a point P in the left camera to the 3D coordinates of the same point in the right camera. So we have:

$$\mathbf{x}_l = R\mathbf{x}_r + \mathbf{t}$$

If we expand this equation:

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

The product of translation matrix T_x and rotation matrix R is called the essential matrix E. The main propriety of E is that we can actually decompose into T_x and R. Given that T_x is a skew-symmetric matrix ($a_{ij} = -a_{ij}$) and R is an orthonormal matrix, it is possible to decouple T_x and R from E by using **singular value decomposition**:

$$\begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

If E is known, we can calculate $\mathbf{t}$ and R. In the end, we obtain:

$$[x_l \quad y_l \quad z_l] \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0$$

Unfortunately, we don't have $\mathbf{x}_l$ and $\mathbf{x}_r$. But what we do know corresponding points in image coordinates. Let's go back and take the perspective equations for the left camera:

$$\begin{cases} z_l u_l = f_x^{(l)} x_l + z_l o_x^{(l)} \\ z_l v_l = f_y^{(l)} y_l + z_l o_y^{(l)} \end{cases}$$

Now using homogenous coordinates we can write:

$$z_l \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = \begin{bmatrix} z_l u_l \\ z_l v_l \\ z_l \end{bmatrix} = \begin{bmatrix} f_x^{(l)} x_l + z_l o_x^{(l)} \\ f_y^{(l)} y_l + z_l o_y^{(l)} \\ z_l \end{bmatrix} = \begin{bmatrix} f_x^{(l)} & 0 & o_x^{(l)} \\ 0 & f_y^{(l)} & o_y^{(l)} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} = K_l \mathbf{x}_l$$

Where K_l is known to us, and is the camera matrix of the left camera. We get such an equation for each one of our views:

$$z_l \begin{bmatrix} u_l \\ v_l \\ 1 \end{bmatrix} = \begin{bmatrix} f_x^{(l)} & 0 & o_x^{(l)} \\ 0 & f_y^{(l)} & o_y^{(l)} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix} \qquad z_r \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = \begin{bmatrix} f_x^{(r)} & 0 & o_x^{(r)} \\ 0 & f_y^{(r)} & o_y^{(r)} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix}$$

We can rewrite these two in a more compact form:

$$\mathbf{x}_l^T = \begin{bmatrix} u_l & v_l & 1 \end{bmatrix} z_l {K_l^{-1}}^T \qquad \mathbf{x}_r = K_r^{-1} z_r \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix}$$

If we substitute these two in the epipolar constraint, we obtain:

$$\begin{bmatrix} x_l & y_l & z_l \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} = 0$$

$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} z_l {K_l^{-1}}^T \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} K_r^{-1} z_r \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

But in this expression we still have z_l and z_r which are the 3D coordinates that we don't know. Assuming that $z_l \neq 0$ and $z_r \neq 0$ (the depth of any point cannot be zero, if it is zero the point lies at the center of projection) the rest of the equation should be equal to 0. So we can simply eliminate z_l and z_r from this equation to get this:

$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} K_l^{-1^T} \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} K_r^{-1} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

Now we have scene points expressed in terms of the image coordinates. The only unknown matrix is E. The product of $K_l^{-1^T}$, E and $K_r^{-1^T}$ is called the fundamental matrix F. So the equation becomes:

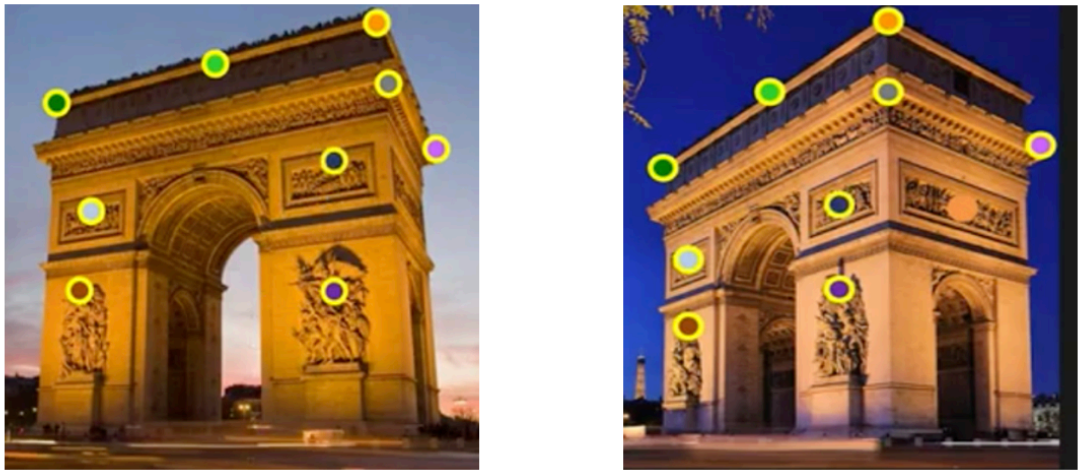
$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

If we find the fundamental matrix, it is simple to extract the essential matrix E:

$$E = K_l^{-1^T} F K_r$$

4.3 - Estimating the fundamental matrix F

The first step is to find a small number m of corresponding features in the two images given to us using SIFT algorithm:



Left image Right image

Figure 18: Example of SIFT algorithm applied on these two photos of *Arc de Triomphe,* credits: https://youtu.be/erpiFudDBlg

The image coordinates of these points are the following:

$$(\mathbf{u}_l^{(1)}, \mathbf{v}_l^{(1)}) \ldots (\mathbf{u}_l^{(m)}, \mathbf{v}_l^{(m)}) \qquad (\mathbf{u}_r^{(1)}, \mathbf{v}_r^{(1)}) \ldots (\mathbf{u}_r^{(m)}, \mathbf{v}_r^{(m)})$$

For each correspondence i, we can write out our epipolar constraint:

$$\begin{bmatrix} u_l^{(i)} & v_l^{(i)} & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u_r^{(i)} \\ v_r^{(i)} \\ 1 \end{bmatrix} = 0$$

If we simply write out this expression for very correspondence i we get m linear equations, that can be written to form a linear system:

$$\begin{bmatrix} u_l^{(1)}u_r^{(1)} & u_l^{(1)}v_r^{(1)} & u_l^{(1)} & v_l^{(1)}u_r^{(1)} & v_l^{(1)}v_r^{(1)} & v_l^{(1)} & u_r^{(1)} & v_r^{(1)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_l^{(i)}u_r^{(i)} & u_l^{(i)}v_r^{(i)} & u_l^{(i)} & v_l^{(i)}u_r^{(i)} & v_l^{(i)}v_r^{(i)} & v_l^{(i)} & u_r^{(i)} & v_r^{(i)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_l^{(m)}u_r^{(m)} & u_l^{(m)}v_r^{(m)} & u_l^{(m)} & v_l^{(m)}u_r^{(m)} & v_l^{(m)}v_r^{(m)} & v_l^{(m)} & u_r^{(m)} & v_r^{(m)} & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

In a more compact form:

$$A\mathbf{f} = 0$$

If we look at the fundamental matrix, it is acting on homogeneous coordinates. So if we take the epipolar constraint we obtain:

$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0 = \begin{bmatrix} u_l & v_l & 1 \end{bmatrix} \begin{bmatrix} kf_{11} & kf_{12} & kf_{13} \\ kf_{21} & kf_{22} & kf_{23} \\ kf_{31} & kf_{32} & kf_{33} \end{bmatrix} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix}$$

If we multiply the fundamental matrix F with a constant k, it doesn't matter. In other words, the fundamental matrix F and kF describe the same epipolar geometry. F is defined only up to a scale.

We set the scale so that:

$$\|\mathbf{f}\|^2 = 1$$

So, we want $A\mathbf{f}$ as close to 0 as possible and $\|\mathbf{f}\|^2 = 1$.

In mathematical terms:

$$\min_f \|A\mathbf{f}\|^2 \text{ such that } \|\mathbf{f}\|^2 = 1$$

This is equivalent to resolve this eigenvalue problem:

$$A^T\mathbf{f} = \lambda\mathbf{f}$$

In other words, the $\mathbf{f}$ that we're looking for is the smallest eigenvalue λ of the matrix A^TA that minimize the loss function $L(\mathbf{f})$. Once we have $\mathbf{f}$, we rearrange the elements of $\mathbf{f}$ to form the fundamental matrix F. Now we can compute the essential matrix E by using this equation:

$$E = K_l^{-1^T} F K_r$$

And now we can decompose E in R and $\mathbf{t}$ using singular value decomposition:

$$E = T_x R$$

4.4 - Finding correspondences

In uncalibrated stereo, finding correspondences it is equivalent to the calibrated stereo case scenario (1D search after having found R and $\mathbf{t}$). But the question is on which line should be searching? This bring back to the epipolar geometry, in particular to the definition of epipolar line:

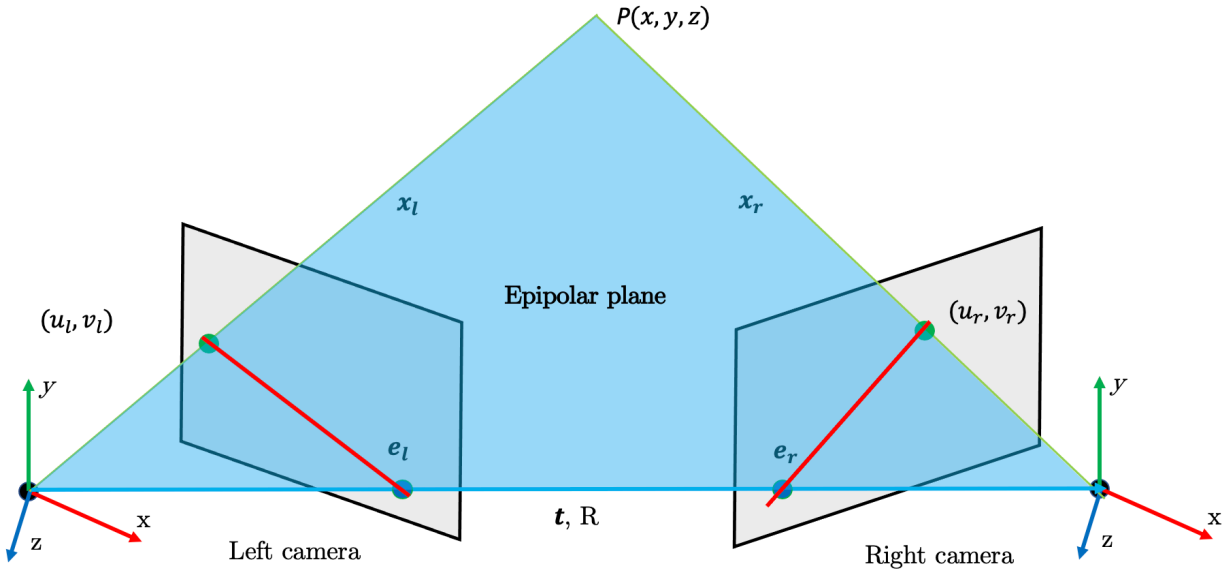


Figure 19: Representation of epipolar lines

The epipolar lines are the intersection of image planes and the epipolar plane. Every scene has two corresponding epipolar lines, one each on the two image planes.

So given a point in one image, the corresponding point in the other image must lie on the epipolar line.

If we know the fundamental matrix F, we can derive the equation of the straight line in the right image along which the search needs to be done.

The epipolar constraint is:

$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

In this case we only don't know $\begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix}$. Expanding the matrix equation:

$$(f_{11}u_l + f_{21}v_l + f_{31})u_r + (f_{12}u_l + f_{22}v_l + f_{32})v_r + (f_{13}u_l + f_{23}v_l + f_{33}) = 0$$

So the equation for the right epipolar line is:

$$a_l u_r + b_l v_r + c_l = 0$$

Likewise, we can calculate epipolar line in the left image for a point in the right image.

This is an example:

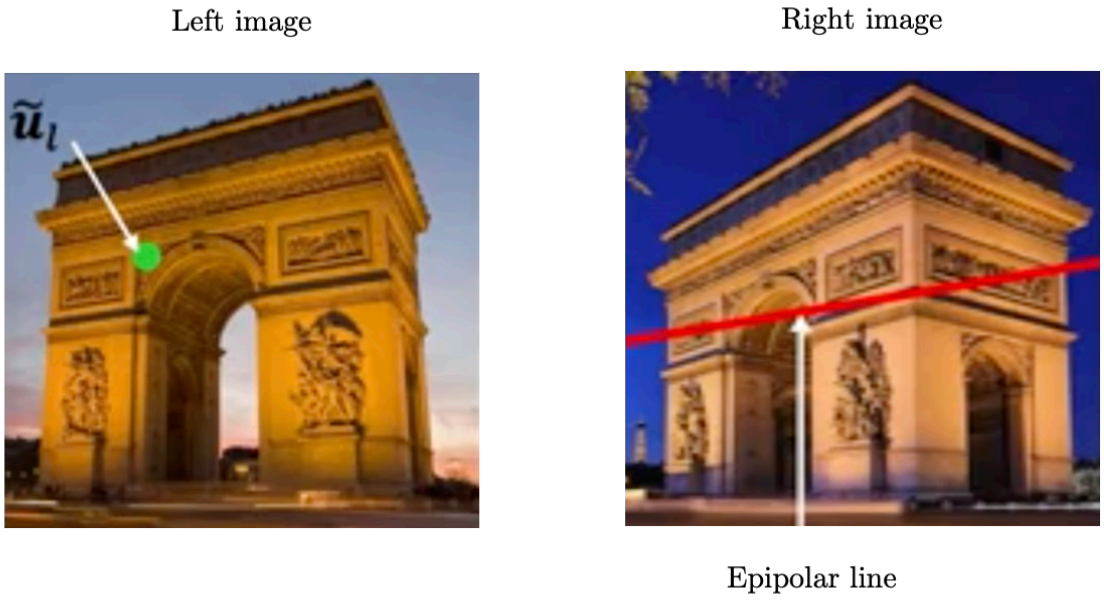


Figure 20: Given a point in the left image, the correspondent point in the right image must lie on the epipolar line, credits: https://youtu.be/erpiFudDBlg

Now if we want to find the point on the left image on the right image, we have to search in the epipolar line, using the stereo matching algorithm described in the 3.3 paragraph.

Chapter 5 - ORB-SLAM3

5.1 - What is ORB-SLAM3

ORB-SLAM3 is a SLAM system (*Visual Simultaneous Localization and Mapping*). It is a software library developed for robots and other devices equipped with a camera that allows them to build a map of their environment and localize themselves within it, all in real-time.

The system is based on the ORB *(Oriented FAST and Rotated BRIEF)* feature detector and descriptor, which allows it to efficiently detect and match visual features in an image. ORB-SLAM3 uses a combination of monocular, stereo, and RGB-D cameras to estimate the camera pose and build a 3D map of the environment.

The system has several advanced features, such as loop closing, relocalization, and dynamic object detection, that improve its accuracy and robustness in real-world scenarios. It has been widely used in robotics, augmented reality, and autonomous driving applications.

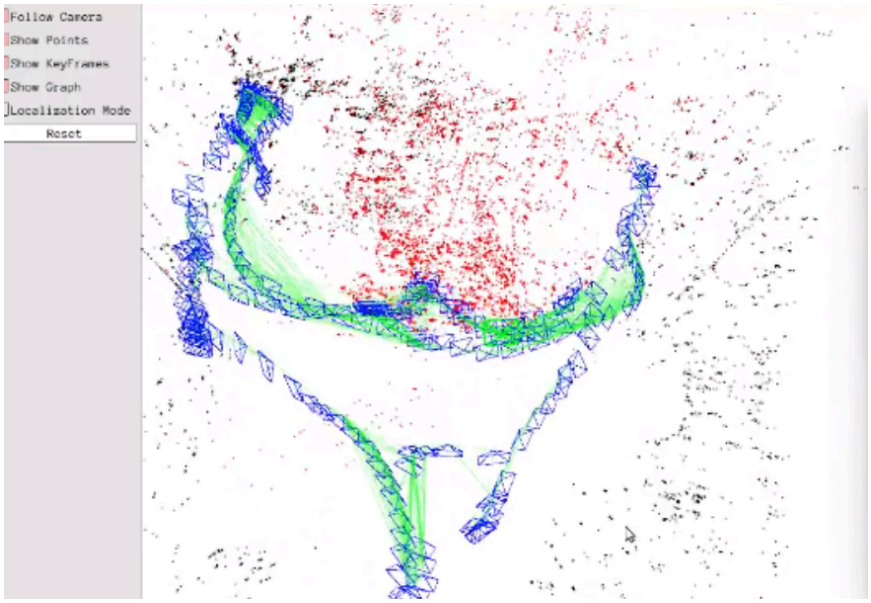


Figure 21: Map Viewer of ORB-SLAM3

5.2 - Stereo camera calibration using OpenCV

This paragraph describes the setup for stereo calibration using a dataset of stereo images of chessboards patterns taken at different angles and views. The image dataset was taken online, and contains 25 stereo images of checkerboard patterns (https://www.kaggle.com/datasets/danielwe14/stereocamera-chessboard-pictures).

The script used for stereo calibration is written in a generic way, making it adaptable for any stereo camera setup. It doesn't need to know the baseline: it automatically detects and calculates it from the stereo images.

The first thing to do is to specify the number of squares on the x and y axis, and also the frame sizes of the images:

```
chessboardSize = (7, 11)
frameSize = (964, 686)
```

After loading all the images present in the *stereoLeft* and *stereoRight* directories, we can loop through every stereo pairs and transform the images to black and white. These images are passed to the *findChessboardCorners()* function, that requires what kind of pattern we are looking for and the black and white images.

It returns the corner points and a boolean value which will be true if the pattern is recognised:

```
imgL = cv.imread(imgLeft)
imgR = cv.imread(imgRight)
grayL = cv.cvtColor(imgL, cv.COLOR_BGR2GRAY)
grayR = cv.cvtColor(imgR, cv.COLOR_BGR2GRAY)

retL, cornersL = cv.findChessboardCorners(grayL,
chessboardSize, None)
retR, cornersR = cv.findChessboardCorners(grayR,
chessboardSize, None)
```

Once the script find the corners, we can increase their accuracy using *cv.cornerSubPix()*. We can also draw the pattern using *cv.drawChessboardCorners()*:

```
cornersL = cv.cornerSubPix(grayL, cornersL, (11, 11),
(-1,-1), criteria)
imgpointsL.append(cornersL)
cornersR = cv.cornerSubPix(grayR, cornersR, (11, 11),
(-1, -1), criteria)
imgpointsR.append(cornersR)


cv.drawChessboardCorners(imgL, chessboardSize, cornersL,
retL)
cv.imshow('img left', imgL)
cv.drawChessboardCorners(imgR, chessboardSize, cornersR,
retR)
cv.imshow('img right', imgR)
```

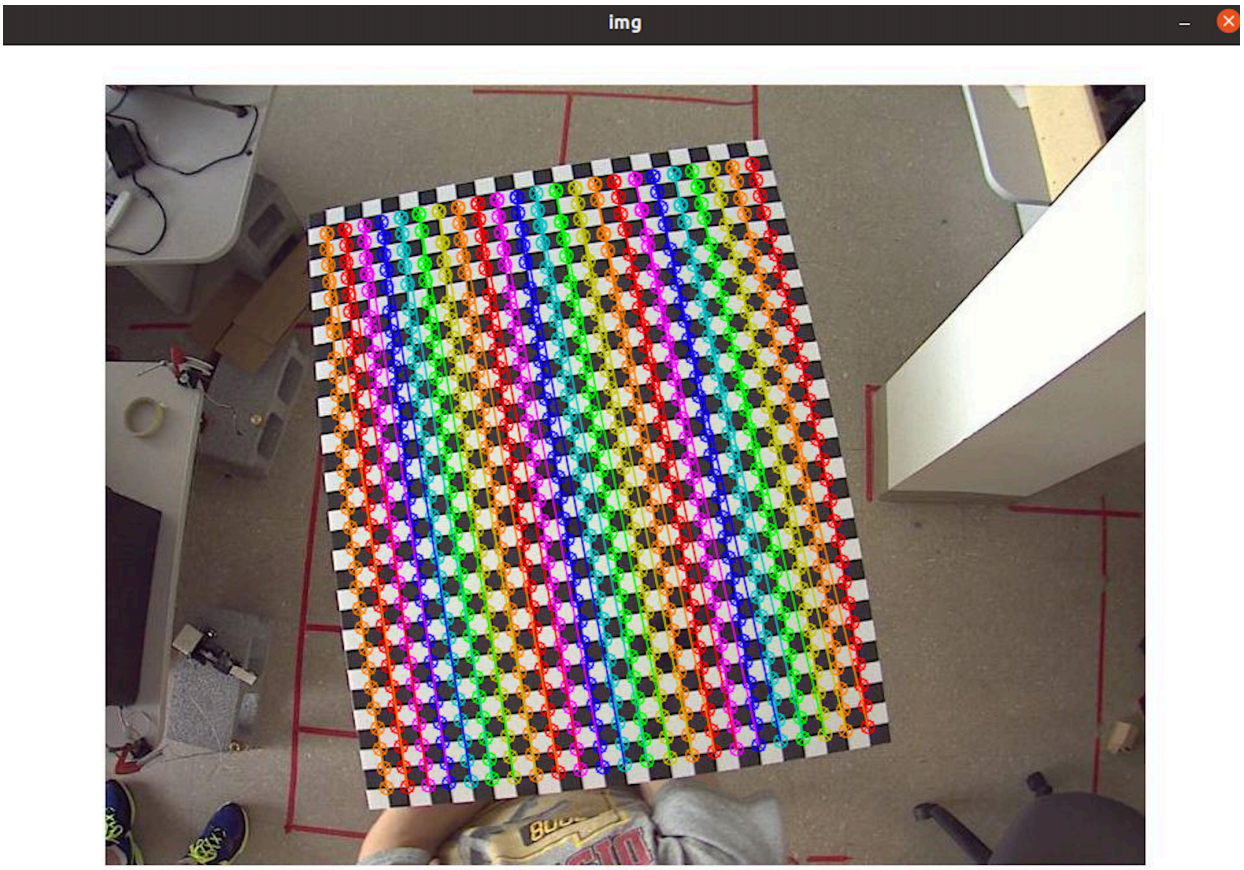


Figure 22: OpenCV recognizes and highlights the chessboard corners on the original image

Now that we have our object points and image points, we are ready to go for the stereo calibration. We can use the function *cv.calibrateCamera()* which returns the camera matrix, distortion coefficients, rotation and translation vectors for both the left and right camera:

```
retR, cameraMatrixR, distorsionCoefficientsR,
rotationVectorsR, traslationVectorR =
cv.calibrateCamera(objectPointsoints, imgpointsR,
frameSize, None, None)
```

We can refine the camera matrix based on a free scaling parameter using *cv.getOptimalNewCameraMatrix()*. If the scaling parameter $\alpha = 0$, it returns undistorted image with minimum unwanted pixels. So it may even remove some pixels at image corners. If $\alpha = 1$, all pixels are retained with some extra black images. This function also returns an image ROI which can be used to crop the result:

```
heigthR, widthR, channelsR = imgR.shape
newCameraMatrixR, roi_R =
cv.getOptimalNewCameraMatrix(cameraMatrixR,
distorsionCoefficientsR, (widthR, heigthR), 1, (widthR,
heigthR))
```

```
flags = 0
flags |= cv.CALIB_FIX_INTRINSIC
criteria_stereo = (cv.TERM_CRITERIA_EPS +
cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)


retStereo, newCameraMatrixL, distorsionCoefficientsL,
newCameraMatrixR, distorsionCoefficientsR,
rot ,trans ,essentialMatrix, fundamentalMatrix =
cv.stereoCalibrate(objectPointsoints, imgpointsL,
imgpointsR, newCameraMatrixL, distorsionCoefficientsL,
newCameraMatrixR, distorsionCoefficientsR, grayL.shape [::
-1], criteria_stereo, flags)
```

Now we want to find the relative position of one camera with respect to the other camera. The function *cv.stereoCalibrate()* computes (R, T) such that:

$$\begin{cases} R_2 = RR_1 \\ T_2 = RT_1 + T \end{cases}$$

Therefore, one can compute 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:

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}$$

This function also computes the essential matrix E and the fundamental matrix F:

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

$$F = cameraMatrix2^T \cdot E \cdot cameraMatrix^{-1}$$

The function *cv.stereoRectify()* computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by *cv.stereoCalibrate()* as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates.

```
rectifyScale = 1
rectL, rectR, projMatrixL, projMatrixR, Q, roi_L, roi_R =
cv.stereoRectify(newCameraMatrixL, distorsionCoefficientsL,
newCameraMatrixR, distorsionCoefficientsR,
grayL.shape[::-1], rot, trans, rectifyScale, (0,0))
```

The new matrices, together with R_1 and R_2, can then be passed to *cv.initUndistortRectifyMap()* to initialize the rectification map for each camera. Computes the undistortion and rectification transformation map.

The function computes the joint undistortion and rectification transformation and represents the result in the form of maps. The undistorted image looks like original, as if it is captured with a camera with zero distortion.

The function actually builds the maps for the inverse mapping algorithm that is used by remap. That is, for each pixel (u, v) in the destination (corrected and rectified) image, the function computes the corresponding coordinates in the source image (that is, in the original image from camera):

Here below, we can see a comparison between the original image, and the corresponding rectified image:

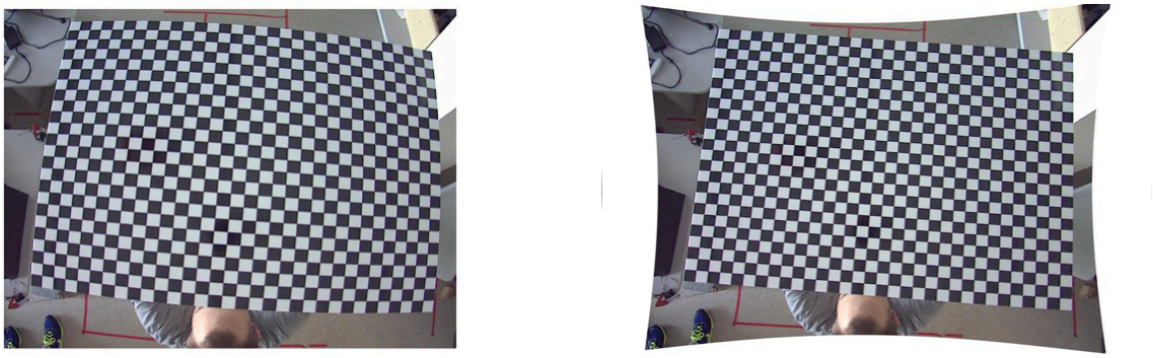


Figure 23: original image VS rectified image. Radial and tangential distortion is no longer visible to the naked eye

5.3 - ORB-SLAM3 installation using Docker

For the installation of ORB-SLAM3, a Docker environment was utilized. Docker provides a flexible and independent platform that allows the seamless deployment of ORB-SLAM3 regardless of the host machine's specifications.

Docker ensures compatibility and ease of installation, enabling future researchers and developers to replicate the experimental setup without concerns regarding system dependencies or configurations.

The initial step involved the installation of Docker by following the instructions provided on its official website (https://docs.docker.com/engine/install/ubuntu/). This included carefully following the installation guidelines, including any necessary post-installation steps outlined in the documentation.

Then, we've cloned the official git repository of ORB-SLAM3 (https://github.com/UZ-SLAMLab/ORB_SLAM3) with the command:

```
git clone https://github.com/UZ-SLAMLab/ORB_SLAM3
```

After placing the Dockerfile inside the ORB-SLAM3 directory (Appendix A.2), the next step involved executing the command:

```
docker build --tag orb-slam3:1.0 .
```

This command initiated the build process, leveraging the Dockerfile to construct a Docker image specifically tailored for ORB-SLAM3. The provided tag *orb-slam3:1.0* served as an identifier for the generated image, ensuring its distinctiveness and version control. This build command triggered the execution of the instructions specified in the Dockerfile, including the installation of necessary dependencies, configuration settings, and the setup of the ORB-SLAM3 environment.

Following the successful build of the ORB-SLAM3 Docker image, the subsequent step involved executing the command:

```
docker create --name orb-slam3 orb-slam3:1.0
```

This command created a Docker container named *orb-slam3* based on the previously built image *orb-slam3:1.0.*

After creating the ORB-SLAM3 Docker container, the next step involved executing the command:

```
xhost +local:root
```

This command granted permission for the root user inside the container to access the host's X server. The X server is responsible for managing graphical user interfaces, and by allowing the container's root user to connect to it, it enabled the display of visual elements from within the container on the host system.

To execute the ORB-SLAM3 Docker container, the following command was utilized:

```
docker run -it --rm --net=host --volume="/dev:/dev" -e DISPLAY=unix$DISPLAY -v /
tmp/.X11-unix:/tmp/.X11-unix:rw --privileged orb-slam3:1.0
```

By utilizing this command, the ORB-SLAM3 Docker container was launched, enabling the execution of the ORB-SLAM3 application while leveraging the host's resources and graphical capabilities.

Inside the ORB-SLAM3 Docker container, the next step involved navigating to the ORB-SLAM3 folder and running the *build_ ros.sh* script. To do this, execute the following commands within the container:

```
./build_ros.sh
```

The *build_ ros.sh* script is designed to facilitate the building and configuration of ORB-SLAM3 with ROS. By executing this script, the necessary dependencies, libraries, and configurations for ROS integration are set up, ensuring the smooth operation of ORB-SLAM3 within the ROS ecosystem.

Once the ORB-SLAM3 compilation process is complete, you can proceed with connecting the camera and running the subsequent commands:

1. Start the ROS core:

```
roscore &
```

2. Launch the *usb_ cam* package to interface with the USB camera:

```
roslaunch usb_cam usb_cam-test.launch &
```

5.4 - Trajectory estimation using Python

After setting up our stereo system with complete calibration, our next step is to estimate the trajectory. To achieve this, we will be using a python script that is available in the ORB-SLAM3 repository (*associate.py* and *evaluate_ate_scale.py* in the evaluation folder).

We used the EuRoC dataset as an example: it includes a sequence of "ground truth" images, making it a suitable choice for our purposes.

EuRoC dataset is a publicly available dataset for evaluating visual odometry, visual SLAM and related algorithms. It was created by the *European Roboticists Association* and contains stereo and monocular sequences recorded from a *Micro Aerial Vehicle* (MAV) flying in different environments, such as indoor, outdoor, and forest areas.

The dataset provides ground-truth camera poses and high-precision measurements of the MAV's motion, making it a valuable resource for researchers and developers working on visual odometry and SLAM algorithms.

The first thing to do is to launch the simulation using this command:

```
./Examples/Stereo/stereo_euroc ./Vocabulary/ORBvoc.txt ./Examples/Stereo/EuRoC.yaml ~/
Datasets/EuRoC/MH01 ./Examples/Stereo/EuRoC_TimeStamps/MH01.txt dataset-MH01_stereo
```

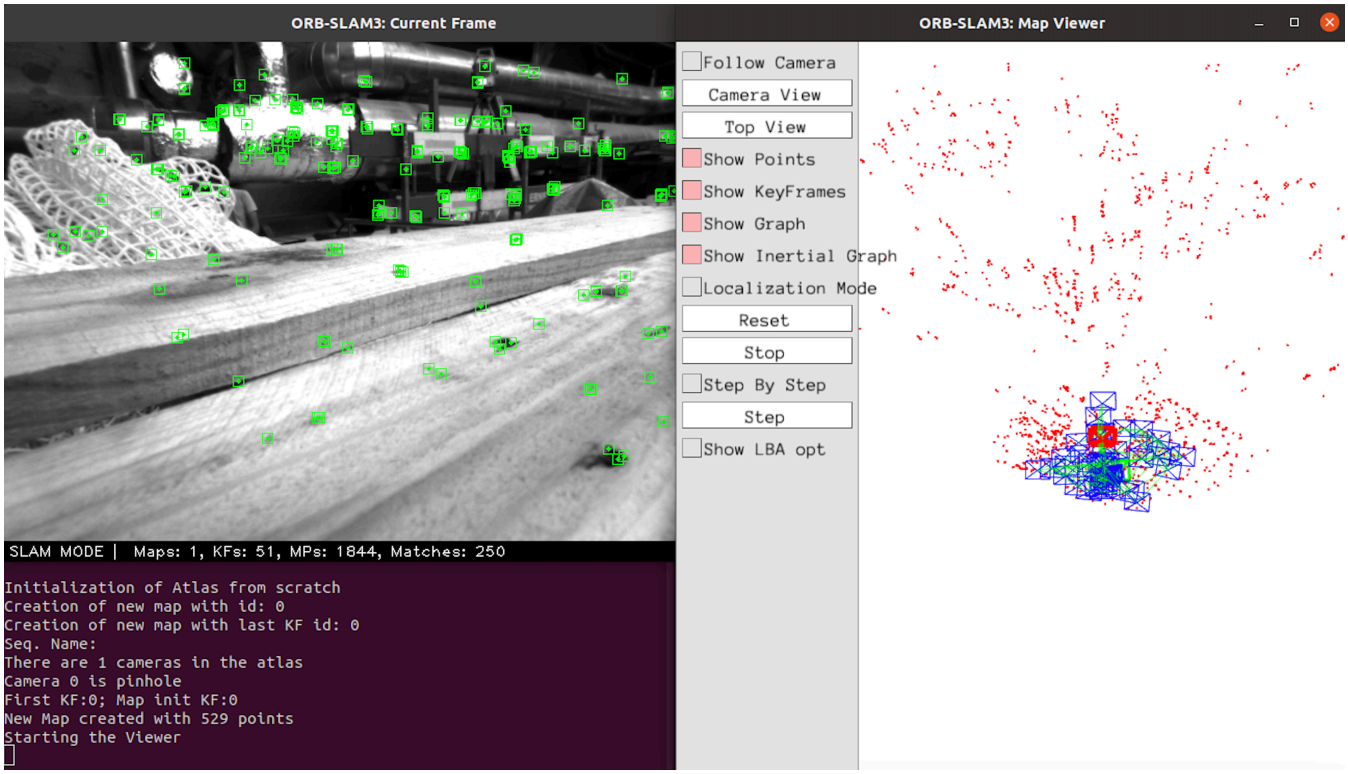


Figure 24: example of the output followed by the previous command

After launching the simulation, we can run the *evaluation.py* script which estimates the trajectory followed by the camera and, with the —*verbose* flag, provides a series of errors that can be used to estimate the goodness of the algorithm (it is essential to use the Python 2.7 interpreter and not version 3 for this code, as it has not been updated to run on version 3):

```
python2.7 evaluation/evaluate_ate_scale.py --verbose evaluation/Ground_truth/
EuRoC_left_cam/MH01_GT.txt f_dataset-MH01_stereo.txt --plot MH01_stereo.pdf
```

Which gives the following output:

```
compared_pose_pairs 3638 pairs
absolute_translational_error.rmse 0.035956 m
absolute_translational_error.mean 0.033244 m
absolute_translational_error.median 0.036838 m
absolute_translational_error.std 0.013701 m
absolute_translational_error.min 0.002904 m
absolute_translational_error.max 0.104954 m
```

The results are saved in the *MH01_stereo.pdf*:

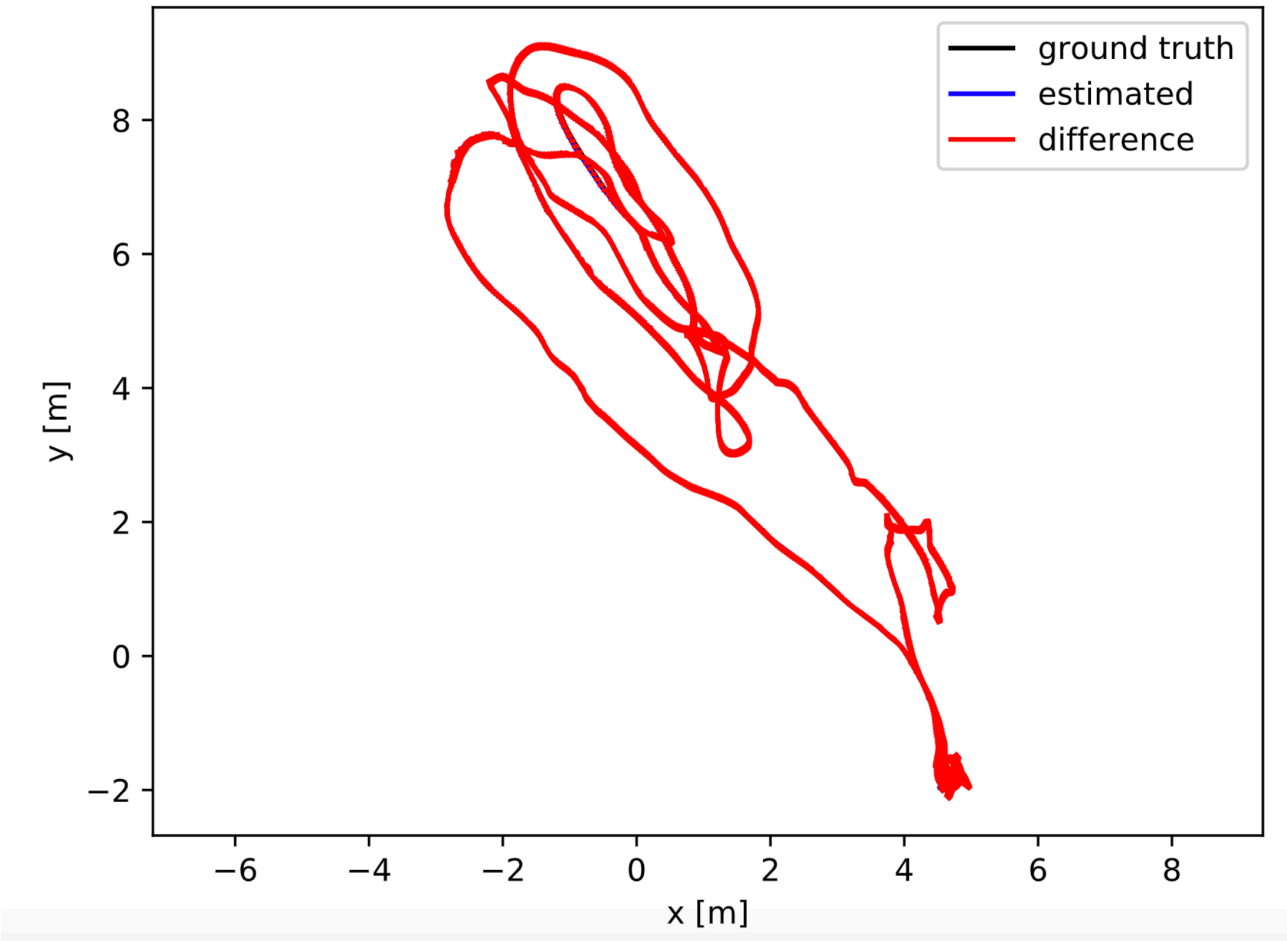


Figure 25: trajectory estimation of the EuRoC stereo dataset on the XY plane

As is evident from the figure, the estimated trajectory in blue aligns precisely with the ground truth trajectory in red. This indicates that the parameters specified in the .yaml file are appropriate for the given dataset.¶

Conclusions

In conclusion, the successful implementation of the stereo camera calibration script, as well as the installation of ORB-SLAM3 highlight the significant advancements in the field of simultaneous localization and mapping (SLAM).

The demonstrated effectiveness of the stereo camera calibration script reaffirms its ability to accurately estimate intrinsic, extrinsic parameters and the essential matrix E, enabling precise depth perception and trajectory estimation. Moreover, the satisfactory performance of ORB-SLAM3 showcases its robustness and efficiency in real-time camera tracking and mapping.

Additionally, the introduction of Isaac SLAM (developed by NVIDIA) as an alternative solution further expands the options available for researchers and developers, with its unique features and capabilities. These advancements hold great promise for various applications, including robotics and autonomous navigation. As technology continues to evolve, the combination of accurate calibration scripts and advanced SLAM algorithms paves the way for even more sophisticated and reliable systems, ultimately driving progress in the field and opening up new possibilities for future research and development.¶

Bibliography

- McCarthy, J. J., Benjamin, J., Winton, T., & Van Duivenvoorde, W. (2019). The Rise of 3D in Maritime Archaeology. *Springer EBooks*, 1–10. https://doi.org/10.1007/978-3-030-03635-5_1
- *ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM.* (2021, December 1). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/document/9440682
- First Principles of Computer Vision. (2021, April 18). *Linear Camera Model | Camera Calibration* [Video]. YouTube. https://www.youtube.com/watch?v=qByYk6JggQU
- First Principles of Computer Vision. (2021, April 18). *Simple Stereo | Camera Calibration* [Video]. YouTube.
- First Principles of Computer Vision. (2021, April 25). *Epipolar Geometry | Uncalibrated Stereo* [Video]. YouTube. https://www.youtube.com/watch?v=6kpBqfgSPRc
- *OpenCV: Camera Calibration.* https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
- Docker: Accelerated, Containerized Application Development. (2023) https://www.docker.com/
- *Chessboard Pictures for Stereocamera Calibration.* (2022, January 20). Kaggle. https://www.kaggle.com/datasets/danielwe14/stereocamera-chessboard-pictures
- Jin, C., & Jeong, H. (2008). Intermediate View Synthesis for Multi-view 3D Displays Using Belief Propagation-Based Stereo Matching. https://doi.org/10.1109/iccit.2008.212