Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Runar Jåtun

Efficient Perspective-n-Point With Graduated Non-Convexity for Outlier Robustness

EPnP with GNC for Outlier Robustness

Master's thesis in Mechanical Engineering
Supervisor: Olav Egeland
January 2023

NTNU
Norwegian University of
Science and Technology

Runar Jåtun

Efficient Perspective-n-Point With Graduated Non-Convexity for Outlier Robustness

EPnP with GNC for Outlier Robustness

Master's thesis in Mechanical Engineering
Supervisor: Olav Egeland
January 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This master's thesis has been written for the Department of Mechanical and Industrial Engineering at the Norwegian University of Science and Technology in the field of robotics and automation. The work was conducted during the autumn semester of 2022, and delivered in January of 2023. The supervisor for this project was Olav Egeland.

I would like to express my gratitude towards my supervisor Olav Egeland, who provided me with an interesting research topic and has provided answers, encouragement and guidance throughout the research process.

This thesis concludes my education at the Norwegian University of Science and Technology. The time at NTNU has been very educational, and I look back at the time with gratitude. There has been ups and downs, and I would like to thank my family and friends for their support during these years. A special thanks to Sigurd for the help in this research.

Abstract

The Perspective-n-Point (PnP) problem is a fundamental problem in computer vision of determining the position and orientation, i.e. the *pose*, of a camera in a 3D environment. Furthermore, this is based on a set of known 3D points and their corresponding 2D projections in an image. Several solutions to this problem have been proposed, but the accuracy decreases when the data is noisy or full of outliers that differ significantly from the other measurements. Therefore, there is a need for more *robust* methods of pose estimation, that is, methods that can handle data sets containing noisy measurements and large amounts of outliers.

This research presents a *new* method for robust pose estimation, called EPnP+GNC. The new method is a combination of the efficient pose estimation method Effective Perspective-n-Point (EPnP) with the outlier optimization method Graduated Non-Convexity (GNC). In order to assess the new method, EPnP+GNC was implemented in a programming language and tested in a variety of scenarios. In addition to EPnP+GNC, several existing methods of pose estimation were tested such that a comparison could be made.

Overall, the results of this study shows that EPnP+GNC is an effective pose estimation method. The method correctly calculates the pose in data sets with a high percentage of outliers, and has a relatively low running time. The results are particularly strong in situations with small rotations. It also shows promising results when compared to other methods, and has shown potential for further development.

Sammendrag

Perspective-n-Point (PnP) er et fundamentalt problem innen datasyn. Problemet dreier seg om å estimere posisjon og orientering av et kamera i en kjent situasjon. Mer bestemt så kjenner man et sett av tredimensjonale punkter og de tilsvarende punktene i et todimensjonalt bilde tatt av kameraet. Flere løsninger til problemet har blitt publisert, men mange metoder blir unøyaktige når dataene i datasettet er fulle av støy eller har ekstremverdier som skiller seg ut fra de resterende punktene. Det er derfor ønskelig med flere *robuste* metoder, altså metoder som klarer å håndtere slike vanskelige situasjoner.

Denne avhandlingen presenterer en ny metode for robust posisjons- og orienteringsestimering, kalt EPnP+GNC. Den nye metoden er en kombinasjon av den effektive metoden Effective Perspective-n-Point (EPnP) og optimaliseringsmetoden Graduated Non-Convexity (GNC). For å evaluere den nye metoden ble EPnP+GNC implementert i et programmeringsspråk, for så å bli testet i forskjellige situasjoner. I tillegg til EPnP+GNC ble også andre, mer veletablerte, metoder for posisjons- og orienteringsestimering testet for å kunne sammenligne resultatene.

Alt i alt viser resultatene av dette studiet at EPnP+GNC er en effektiv metode for estimering av posisjon og orientering. Denne nye metoden estimerer posisjon og orientering riktig, selv i datasett som inneholder høy prosentandel ekstremverdier. Resultatene er spesielt sterke i situasjoner med lite rotasjon i orienteringen. Testene viser også at EPnP+GNC har relativt høy hastighet. I tillegg peker resultatene mot at EPnP+GNC har et godt resultat sammenlignet med de andre metodene. EPnP+GNC viser et stort potensial for videre utvikling.

Contents

Figures

Tables

Code Listings

Acronyms

EPnP Effective Perspective-n-Point. v, vii, 1–3, 11, 14, 19–22, 25, 26, 28, 29, 37, 40, 42, 43, 45, 46

EPnP+GNC EPnP+GNC. v, vii, 2, 14, 19, 21, 23, 25–29, 31–34, 36, 37, 39–43, 45, 46

GN Gauss-Newton. 11, 14, 25, 26, 31, 43

GNC Graduated Non-Convexity. v, vii, 1–3, 11, 14, 16–20, 22, 25, 26, 28, 29, 32, 34, 37, 45, 46

PnP Perspective-n-Point. v, vii, 1, 3, 10, 11, 21, 23, 26, 27, 35, 45, 46

RANSAC Random Sample Consensus. 3, 10, 14–16, 26, 45, 46

TLS Truncated Least Squares. 16–18

Chapter 1

Introduction

1.1 Motivation

The Perspective-n-Point (PnP) problem [1] is a problem of figuring out the position and orientation, i.e. the *pose*, of a camera in a 3D environment. Furthermore, this is done based on a set of 3D points and their corresponding 2D projections in an image taken by the camera. This problem has many applications in computer vision, such as when trying to navigate a robot in a known environment, overlaying medical imaging on a patient, or trying to track and recognize objects in a scene captured by a camera [2, 3].

The PnP problem is a challenging one, as it involves estimating the pose of an object from a set of potentially noisy and incomplete measurements. Various algorithms and techniques have been developed to tackle the problem, including non-iterative methods, optimization-based approaches, and machine learning techniques [3–9].

This thesis highlights the Effective Perspective-n-Point (EPnP) method [4]. This is a fast and accurate algorithm in many cases, however, EPnP has been shown to be sensitive to the presence of outlier correspondences. Essentially, this means that its performance can deteriorate in the presence of noise and outliers.

Outlier optimization is a technique that can be used to improve the accuracy of pose estimation algorithms. The idea behind outlier optimization is to identify and remove "outliers" from the set of measurements used to estimate the pose of an object. By removing these outliers, the pose estimation algorithm can be more accurate, as it is not influenced by noisy or erroneous measurements.

Outlier optimization is achieved using various algorithms and techniques, such as the method used in this work; Graduated Non-Convexity (GNC) [10, 11]. These methods typically involve iterative fitting of a model to the data, and identifying and removing measurements that are not consistent with the model.

In this research a new method is proposed. By combining EPnP with GNC, the aim is to improve the accuracy and robustness in estimating the pose in these challenging scenarios. This new method henceforth called EPnP+GNC.

1.2 Objectives

The overarching goal of this thesis is

- Do a feasibility study on the potential benefits of combining EPnP with GNC.

The idea is that identifying and removing outliers from the data set will increase the accuracy of the pose estimation. This is done by evaluating the performance of the combined algorithms in a variety of scenarios, and to compare its accuracy and robustness to another robust pose estimation method.

In order to achieve the overarching goal, several smaller goals needs to be achieved:

- Establish the necessary equations needed to combine the methods.
- Implement EPnP+GNC in a programming language for testing.
- Test the implementation in a variety of scenarios.

By achieving these objectives, this thesis will contribute to the ongoing research in computer vision and pose estimation. It will provide a better understanding of potential methods of outlier robustness in pose estimation.

1.3 Thesis Overview

This master thesis consists of 6 chapters:

Chapter 2: Background Provides the necessary context and information for the reader to understand and evaluate the research being presented, and discusses how the research contributes to the existing body of knowledge.

Chapter 3: Method This chapter describes the methods and techniques that were used in the study, including the data collection methods, statistical analysis, and any other methods that were used to analyze the data.

Chapter 4: Results Presents the results of the experiments in the study, including any statistical analyses and findings.

Chapter 5: Discussion Discussing the implications of the results and their significance for the research question. Includes an evaluation of the method.

Chapter 6: Conclusion This chapter summarizes the main points of the thesis and provides a conclusion to the research question.

Chapter 2

Background

This chapter of the thesis provides the reader with the theory that is necessary to understand the rest of the thesis. An introduction to PnP and outlier optimization is included, with the necessary background theory. Existing research of the topic of PnP is discussed, and the EPnP method is presented. The topic of optimization and some existing research is also discussed. A few algorithms for outlier optimization are presented and discussed, namely RANSAC and GNC.

2.1 Preliminaries

2.1.1 Point Clouds

The set of points used in the calculation of the pose is usually given as a *point cloud*. A point cloud is a set of n vertices that represent n points in a 3 dimensional (3D) space [12]. The points can be represented in many ways, but the most common representation is in Euclidean space by the Cartesian coordinates in frame w as

$$\boldsymbol{x}_i^w = [x_i, y_i, z_i]^T,\ i = 1 \dots n \tag{2.1}$$

The vertices can also contain other information, such as color, brightness and surface normal, but in this thesis the position is the key parameter. Point clouds can be produced in several ways, most notably by 3D imaging systems, such as photogrammetry and 3D scanners [3, 13]. They can also be exported from 3D models, which is done in this project.

The point cloud can be expressed as

$$\boldsymbol{X}^w = [\boldsymbol{x}_1{}^w, \dots, \boldsymbol{x}_n{}^w] \tag{2.2}$$

2.1.2 Rotation Matrices

The points can be manipulated in certain ways, such as rotation. One way to do this is with rotation matrices [14]. Points $\boldsymbol{x}_i$ are mapped to $\hat{\boldsymbol{x}}_i$ with rotation matrix $\boldsymbol{R}$ with

$$\hat{\boldsymbol{x}}_i^w = \boldsymbol{R}\boldsymbol{x}_i^w \tag{2.3}$$

In the plane (2D) a rotation matrix of angle θ will look like

$$\boldsymbol{R}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.4}$$

Rotation matrices in 3D are similar to their 2D counterpart, but are 3×3 instead of 2×2. The 3D rotation matrices around the axis x, y and z look like

$$\boldsymbol{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.5}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.6}$$

$$\boldsymbol{R}_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

One can get composite rotations by multiplying rotation matrices, such as

$$\boldsymbol{R}_{tot} = \boldsymbol{R}_x\boldsymbol{R}_y\boldsymbol{R}_z \tag{2.8}$$

It should be noted that matrix multiplication is not commutative, so changing the order in which the matrices are multiplied will also change the resulting rotation.

There are other ways of representing a rotation, and one of them is the *axis-angle* representation [14]. The axis-angle representation defines the rotation as a angle θ about a rotation axis defined by the unit vector $\hat{\boldsymbol{v}}$. The rotation can then be described as

$$\boldsymbol{v} = \theta\hat{\boldsymbol{v}} \tag{2.9}$$

The rotation matrix can using this be defined by the *Rodrigues' formula* for rotation

$$\boldsymbol{R} = \text{Rot}(\hat{\boldsymbol{v}}, \theta) = \boldsymbol{I} + \sin\theta\,\hat{\boldsymbol{v}}^{\times} + (1 - \cos\theta)\hat{\boldsymbol{v}}^{\times}\hat{\boldsymbol{v}}^{\times} \tag{2.10}$$

In this equation, $\hat{\boldsymbol{v}}^{\times}$ is the *skew symmetric* form of the vector $\hat{\boldsymbol{v}}$. The Rodrigues' formula can be rewritten in multiple ways by using different trigonometric identities. In this work, one way is used. By recognising that

$$\theta = \|\boldsymbol{v}\|, \quad \hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|} \tag{2.11}$$

and

$$1 - \cos\theta = 2\sin^2\frac{\theta}{2} \tag{2.12}$$

one can rewrite the Rodrigues' formula as

$$\boldsymbol{R} = \boldsymbol{I} + \text{sinc}\,(\|\boldsymbol{v}\|)\,\hat{\boldsymbol{v}}^{\times} + \frac{1}{2}\text{sinc}^2\left(\frac{\|\boldsymbol{v}\|}{2}\right)\hat{\boldsymbol{v}}^{\times}\hat{\boldsymbol{v}}^{\times} \tag{2.13}$$

where $\text{sinc}\,(\theta) = \frac{\sin\theta}{\theta}$.

2.1.3 Transformation Matrices

Another way to manipulate the points is translation [14]. A translation $\boldsymbol{t}$ has the distance to be translated in each axis embedded, and is expressed

$$\boldsymbol{t} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{2.14}$$

One can combine translation and rotation as

$$\hat{\boldsymbol{x}}_i^w = \boldsymbol{R}\boldsymbol{x}_i^w + \boldsymbol{t} \tag{2.15}$$

In this equation, multiplication *and* addition is used. A better way would be to only use multiplication, and therefore the homogeneous vector is introduced.

$$\boldsymbol{x}_i^w = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}, \quad \tilde{\boldsymbol{x}}_i^w = \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2.16}$$

Using the homogeneous vector $\tilde{\boldsymbol{x}}_i^w$, equation 2.15 simplifies to

$$\hat{\tilde{\boldsymbol{x}}}_i^w = \boldsymbol{T}\tilde{\boldsymbol{x}}_i^w \tag{2.17}$$

where $\boldsymbol{T}$ is the *Homogeneous Transformation matrix* from $\tilde{\boldsymbol{x}}$ to $\hat{\tilde{\boldsymbol{x}}}$. $\boldsymbol{T}$ is a 4×4 matrix that includes the rotation and translation.

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{bmatrix} \tag{2.18}$$

Transformation matrices can also be used to change from one coordinate frame to another. Given the points $\tilde{\boldsymbol{x}}_i^w$ in frame w and the corresponding $\tilde{\boldsymbol{x}}_i^c$ in frame c, the relationship between the points is given by

$$\tilde{\boldsymbol{x}}_i^c = \boldsymbol{T}_w^c \tilde{\boldsymbol{x}}_i^w \tag{2.19}$$

where $\boldsymbol{T}_w^c = \begin{bmatrix} \boldsymbol{R}_w^c & \boldsymbol{t}_w^c \\ \boldsymbol{0} & 1 \end{bmatrix}$ is the transformation from frame w to frame c.

2.1.4 Angular Distance

It is important to quantify the success of the different methods used in the experiments in later chapters. In order to do this, some parts of the calculated transformation matrices were analysed to understand the difference between the true transformation and the estimated transformation.

The rotation can be quantified using *angular distance* [15]. As explained in Section 2.1.2, every rotation in 3D can be expressed as a rotation about some unit axis $\hat{\boldsymbol{v}}$ using an angle θ. The angle can always be chosen to be $0 \leq \theta \leq \pi$. Consider two rotation matrices $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$. The angular distance between the two matrices is defined as the angle of the rotation $\boldsymbol{R}_e = \boldsymbol{R}_1\boldsymbol{R}_2^T$ [15]. The angle of the rotation matrix can then be calculated from

$$\theta_e = \arccos \frac{\mathrm{tr}\boldsymbol{R}_e - 1}{2}, \quad 0 \leq \theta \leq \pi \tag{2.20}$$

The estimated translation also needed to be compared to the true translation. One method to find the error e_t is to take the norm of the difference between the true translation $\boldsymbol{t}_{true}$ and the estimated translation $\boldsymbol{t}_{calc}$.

$$e_t = \|\boldsymbol{t}_{true} - \boldsymbol{t}_{calc}\| \tag{2.21}$$

2.1.5 The Camera Model

The Camera Model is the mathematical relationship between the pixels taken by a camera in the 2D image plane and the 3D counterparts in Euclidean space [3]. One property of this mapping is the loss of a dimension, which can make it difficult to know the depth of a point in 3D from the 2D pixel. Another property is the perspective effect, which means that parallel lines may not be parallel in both 2D and 3D. There are methods of capturing images without change in perspective, but they are rarely used.

The Pinhole Camera Model

The standard model of explaining the camera model is *The Pinhole Camera*. Imagine a camera that is taking a photo of a 3D scene. The Figure 2.1 shows how light from one point will go through the *center of projection* in the middle of the *focal plane* and be mapped onto the *retinal plane*. The retinal plane lies at a distance f from the center of projection, and is called the *focal length* [3].

It is more intuitive to consider the *image plane*. The image plane is a fully virtual plane that lies *in front* of the focal plane, as opposed to the retinal plane that lies *behind* the focal plane. As shown in Figure 2.1, the point $\boldsymbol{x}^c$ is mapped to the image plane as $\hat{\boldsymbol{x}}^c$.

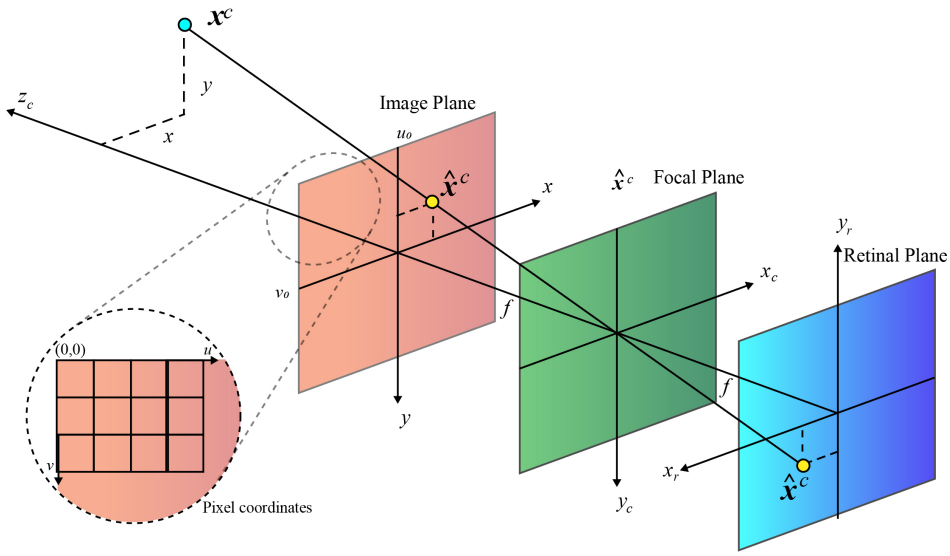


Figure 2.1: Image plane, focal plane and retinal plane. Notice the origin of the pixel coordinate frame.

To ease the math additionally, the *normalized image plane* is introduced. It is placed at a distance of *one* from the origin and is parallel to the image plane. This gives the *normalized image coordinates* $\tilde{\boldsymbol{s}}_i$

$$\tilde{\boldsymbol{s}}_i = \begin{bmatrix} \frac{x_i}{z_i} \\ \frac{y_i}{z_i} \\ 1 \end{bmatrix} = \frac{1}{z_i} \boldsymbol{x}_i^c \tag{2.22}$$

Calculating the pixel values in the image plane uses the *Intrinsic Camera Parameter Matrix* $\boldsymbol{K}$

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

Here f is the focal length and u_0 and v_0 are the pixel offsets of the image sensor, shown in Figure 2.1. These values are specific to each camera used. The pixel values $\boldsymbol{p}_i = [u_i, v_i]^T$ can then be calculated as

$$\tilde{\boldsymbol{p}}_i = \boldsymbol{K}\tilde{\boldsymbol{s}}_i \tag{2.24}$$

The points used to calculate the normalized image coordinates are not always expressed in the camera frame, and it is then necessary to transform the points to the camera frame. By using the transformation matrix, the pixel values can be calculated from points in the world frame using

$$\tilde{\boldsymbol{p}}_i = \boldsymbol{K}\boldsymbol{\Pi}\boldsymbol{T}_w^c \tilde{\boldsymbol{x}}_i^w \tag{2.25}$$

where $\boldsymbol{\Pi} = [\boldsymbol{I}_3 \ \ \boldsymbol{0}]$ is a 3×4 matrix used in the calculation of the homogeneous coordinates. $\boldsymbol{I}_3$ is the 3×3 identity matrix.

2.1.6 Procrustes Problem

The *Procrustes Problem* is a problem in linear algebra of estimating the matrix $\boldsymbol{\Omega}$ that best maps $\boldsymbol{A}$ to $\boldsymbol{B}$ [16, 17]. The solution to the problem is often used in estimating point correspondences in 3D with a rotation matrix. One can also consider the mapping of point correspondences with transformation matrices.

The solution to the problem where rotation and translation needs to be estimated can be calculated using the method presented in [17]. Consider the point clouds $\boldsymbol{A}$ and $\boldsymbol{B}$ consisting of the vectors $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ for $i = 1, .., n$. One point cloud corresponds to the other by the rotation $\boldsymbol{R}$, translation $\boldsymbol{t}$ and scaling c, such that

$$\boldsymbol{A} = c\boldsymbol{R}\boldsymbol{B} + \boldsymbol{t} \tag{2.26}$$

The solution is found from the mean squared error

$$e^2(\boldsymbol{R}, \boldsymbol{t}, c) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{b}_i - (c\boldsymbol{R}\boldsymbol{a}_i + \boldsymbol{t})\|^2 \tag{2.27}$$

The minimum value of equation 2.27 is

$$\epsilon^2 = \sigma_b^2 - \frac{\mathrm{tr}(\boldsymbol{D}\boldsymbol{S})^2}{\sigma_x^2} \tag{2.28}$$

where

$$\boldsymbol{\mu}_a = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{a}_i \tag{2.29}$$

$$\boldsymbol{\mu}_b = \frac{1}{n}\sum_{j=1}^{n} \boldsymbol{b}_i \tag{2.30}$$

$$\sigma_a = \frac{1}{n}\sum_{j=1}^{n} \|\boldsymbol{a}_i - \boldsymbol{\mu}_a\|^2 \tag{2.31}$$

$$\sigma_b = \frac{1}{n}\sum_{j=1}^{n} \|\boldsymbol{b}_i - \boldsymbol{\mu}_b\|^2 \tag{2.32}$$

$$\Sigma_{ab} = \frac{1}{n}\sum_{j=1}^{n} (\boldsymbol{b}_i - \boldsymbol{\mu}_b)(\boldsymbol{a}_i - \boldsymbol{\mu}_a)^T \tag{2.33}$$

where Σ_{ab} is a covariance matrix of $\boldsymbol{A}$ and $\boldsymbol{B}$, $\boldsymbol{\mu}_a$ and $\boldsymbol{\mu}_b$ are the mean of $\boldsymbol{A}$ and $\boldsymbol{B}$ and σ_a^2 and σ_b^2 are variances around the mean. In addition to this a singular value decomposition of Σ_{ab} will be $\boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$, and

$$\boldsymbol{S} = \begin{cases} \boldsymbol{I}, & \text{if } \det(\Sigma_{ab}) \geq 0 \\ \mathrm{diag}(1, 1, \cdots, 1, -1), & \text{if } \det(\Sigma_{ab}) < 0 \end{cases} \tag{2.34}$$

Using these parameters, the optimal transformation is determined as

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T \tag{2.35}$$

$$\boldsymbol{t} = \boldsymbol{\mu}_{\boldsymbol{b}} - c\boldsymbol{R}\boldsymbol{\mu}_a \tag{2.36}$$

$$c = \frac{1}{\sigma_x^2}\mathrm{tr}(\boldsymbol{D}\boldsymbol{S}) \tag{2.37}$$

2.2 Perspective-n-Point

The term Perspective-n-Point (PnP) was coined by Fischler and Bolles in the paper which introduced RANSAC [1]. This is a fundamental problem in computer vision that involves estimating the pose of a known object in 3D space from a set of 2D image point correlations. In this problem, the camera parameters are also known. The n in PnP refers to the number of points used to solve the problem. In general, the more points that are used, the more accurate the pose estimation will be. However, using too many points can also lead to computational inefficiency and decreased performance.

The PnP problem is, as previously explained, a problem of estimating the pose from given 3D-2D correspondences. Consider the 3D points $\boldsymbol{x}_i^w$ in world frame and the 2D pixels $\boldsymbol{p}_i$. The pixels are taken by a camera with a known camera parameter matrix $\boldsymbol{K}$. Suppose that there exists pixels that each corresponds to one point, and that the correspondences are known. Then the PnP problem consists of estimating the transformation matrix that correctly maps the points to the corresponding pixel, as shown in equation 2.25.

2.2.1 Existing Methods of Solving PnP

The PnP problem is an active area of research in computer vision, and there are many different algorithms and approaches that have been developed to solve it. The earliest approaches to solving the problem was the use of closed-form solutions, which involve finding an analytical solution to the problem using algebraic manipulations [3]. The problem has been addressed for varying numbers of correspondences. Moreover, the minimum number of points that is necessary to yield a finite solution is three, though this may result multiple unique solutions. Several solutions have been proposed to solve the P3P problem [6, 18], the P4P problem [1] and for n correspondences [4, 5, 7, 19, 20].

The solutions for small numbers of correspondences are often efficiently solved. However, they are not very *robust*, meaning that they are sensitive to noise. This is because the noise will be proportionally large compared to larger number of correspondences. In larger data sets this problem is less apparent, as the larger number of correspondences inherently inhibits some degree of redundancy. That means that algorithms that work with varying numbers of correspondences *can* be more robust towards noise and outliers. There are several methods for solving the generalized PnP problem, and they can be classified as either *iterative* or *non-iterative*.

Non-iterative methods have the advantage of being efficient, but they lack stability when dealing with noisy measurements, especially when $n \leq 5$ [20]. This can be combated by increasing the number of correspondences, but this is also increasing the complexity of the calculations. Many solutions dealing with the PnP problem are time consuming because of this [20]. The first non-iterative, linear complexity

algorithm with real-time capabilities was Effective Perspective-n-Point (EPnP) [4].

The other approach to solving the PnP problem is the use of iterative methods, which involve repeatedly refining an initial solution until convergence is reached. The aforementioned EPnP can include a Gauss-Newton optimization [4] to increase the estimation accuracy of the method. In addition to this, a reformulation of EPnP with improved robustness was introduced in 2014 called Robustified EPnP [8]. REPnP can for instance yield accurate results in data sets containing 50% outliers. These iterative methods can provide stable and accurate solutions for a wide range of problems, but they may be computationally expensive and may not be suitable for real-time applications. In addition to this, they often struggle with very high percentages of outliers.

Recently, the discovery of Kukelove et al. [21] of simplifying the use of Gröbner basis solvers has been used in several new methods of solving the PnP problem. Examples of this is SQPnP [5] and optDLS [19]. Another recent approach to solving the PnP problem is to use machine learning techniques, such as neural networks [9]. These methods can learn from a training data set and provide a flexible and robust solution, but they require a large amount of training data and may not generalize well to unseen data.

Even though it is not the state-of-the-art method any longer, EPnP is still a much used and well respected method thanks to its properties and results. It is for instance the default method used in *OpenCV* for PnP pose estimation [22]. The main advantage of EPnP is that it is very fast [7]. This is also the main reason it has been chosen in this work, as GNC iterates several times over the entire method, and having a fast method will then be very beneficial.

2.2.2 Efficient Perspective-n-Point

The Effective Perspective-n-Point (EPnP) method is a non-iterative method for solving the Perspective-n-Point (PnP) problem in computer vision. EPnP was published in 2008 by Lepetit, Moreno-Noguer and Fua [4], and was the first method of linear complexity that had real world applicability. It is a closed-form solution that provides an efficient and accurate solution to the PnP problem for a wide range of cases.

Derivation of EPnP

Most methods prior to EPnP calculated the pose in the PnP problem in an inefficient way. This can take a lot of time and be resource intensive, especially with large numbers of point correspondences. In order to increase efficiency, EPnP introduces *control points* to do the calculation.

Consider the *n reference* points $\boldsymbol{x}_i^w$ and the *four control points* $\boldsymbol{c}_i^w$ defined in the *world* coordinate frame.

$$\boldsymbol{x}_i^w,\ i=1,\dots,n \quad \text{and} \quad \boldsymbol{c}_j^w,\ j=1,\dots,4$$

Then the reference points $\boldsymbol{x}_i^w$ can be expressed as weighed sums of the control points $\boldsymbol{c}_i^w$ by using

$$\boldsymbol{x}_i^w = \sum_{j=1}^{4} \alpha_{ij}\, \boldsymbol{c}_j^w \qquad \text{with} \sum_{j=1}^{4} \alpha_{ij} = 1 \tag{2.38}$$

where α_{ij} are the weights. The weighted sum can then be formulated as

$$\begin{bmatrix} \boldsymbol{c}_1^w & \boldsymbol{c}_2^w & \boldsymbol{c}_3^w & \boldsymbol{c}_4^w \end{bmatrix} \boldsymbol{\alpha}_i = \boldsymbol{x}_i^w \tag{2.39}$$

where

$$\boldsymbol{\alpha}_i = \begin{bmatrix} \alpha_{i1} & \alpha_{i2} & \alpha_{i3} & \alpha_{i4} \end{bmatrix}^T \tag{2.40}$$

Furthermore, by using

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^w & \cdots & \boldsymbol{x}_n^w \end{bmatrix} \tag{2.41}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{c}_1^w & \boldsymbol{c}_2^w & \boldsymbol{c}_3^w & \boldsymbol{c}_4^w \end{bmatrix} \tag{2.42}$$

$$\boldsymbol{A} \begin{bmatrix} \boldsymbol{\alpha}_1 & \cdots & \boldsymbol{\alpha}_n \end{bmatrix} \tag{2.43}$$

the weighted sum becomes

$$\boldsymbol{C}\boldsymbol{A} = \boldsymbol{X} \tag{2.44}$$

The weights are then calculated using

$$\boldsymbol{A} = \boldsymbol{C}^{-1}\boldsymbol{X} \tag{2.45}$$

In theory, the control points can be chosen at random, but [4] suggest that the method is more stable if one of the control points is chosen as the centroid of the reference points, and the remaining control points chosen such that they form a basis aligned with the principal direction of the rest of the data.

The important observation is that the relation in 2.38 also holds in the camera coordinate system

$$\boldsymbol{x}_i^c = \sum_{j=1}^{4} \alpha_{ij}\, \boldsymbol{c}_j^c \tag{2.46}$$

It also follows that

$$\tilde{\boldsymbol{x}}_i^c = \boldsymbol{T}_w^c \tilde{\boldsymbol{x}}_i^w \quad \text{and} \quad \tilde{\boldsymbol{c}}_j^c = \boldsymbol{T}_w^c \tilde{\boldsymbol{c}}_j^w \tag{2.47}$$

In Section 2.1.5 it is shown that $\tilde{\boldsymbol{p}}_i = \boldsymbol{K}\tilde{\boldsymbol{s}}_i$ and $\lambda_i \tilde{\boldsymbol{s}}_i = \boldsymbol{x}_i^c$. Using this knowledge with equation 2.38 gives

$$\lambda_i \tilde{\boldsymbol{s}}_i = \boldsymbol{x}_i^c = \boldsymbol{\Pi} \boldsymbol{T}_w^c \tilde{\boldsymbol{x}}_i^w = \boldsymbol{\Pi} \sum_{j=1}^{4} \alpha_{ij}\, \boldsymbol{T}_w^c\, \tilde{\boldsymbol{c}}_j^w \tag{2.48}$$

$$\lambda_i \tilde{\boldsymbol{s}}_i = \sum_{j=1}^{4} \alpha_{ij}\, \boldsymbol{c}_j^c \tag{2.49}$$

$$\bar{\lambda}_i \tilde{\boldsymbol{p}}_i = \boldsymbol{K} \sum_{j=1}^{4} \alpha_{ij}\, \boldsymbol{c}_j^c \tag{2.50}$$

$$\bar{\lambda}_i \begin{bmatrix} u_0 \\ v_0 \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_i^c \\ y_i^c \\ z_i^c \end{bmatrix} \tag{2.51}$$

In the resulting system, the unknown parameters are the 12 control point coordinates and the n parameters $\bar{\lambda}_i$. $\bar{\lambda}_i$ is found from the last row in each point correspondence, as

$$\bar{\lambda}_i = \sum_{i=1}^{4} \alpha_{ij} z_j^c \tag{2.52}$$

which yields two linear equations for each reference point.

$$\sum_{j=1}^{4} (\alpha_{ij}\, f\, x_j^c + \alpha_{ij}\, (u_0 + u)\, z_j^c) = 0 \tag{2.53}$$

$$\sum_{j=1}^{4} (\alpha_{ij}\, f\, y_j^c + \alpha_{ij}\, (v_0 + v)\, z_j^c) = 0 \tag{2.54}$$

This is then rearranged to a linear system on the form

$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{0} \tag{2.55}$$

where $\boldsymbol{x} = [\boldsymbol{c_1^c}^T, \boldsymbol{c_2^c}^T, \boldsymbol{c_3^c}^T, \boldsymbol{c_4^c}^T]^T$ and $\boldsymbol{M}$ is a $2n \times 12$ matrix consisting of the elements in the two linear equations. The solution belongs to the null space of $\boldsymbol{M}$, and can be calculated with a *Singular Value Decomposition* (SVD) as

$$\boldsymbol{x} = \sum_{i=1}^{N} \beta_i \boldsymbol{v}_i \tag{2.56}$$

where $\boldsymbol{v}_i$ are the columns of the right-singular vector of M. The β_i is found from the fact that the distances between control points in the camera frame should be the same as the distances between the control points in the world frame. This is done different ways when the dimension of the null space of $\boldsymbol{M}^T\boldsymbol{M}$ varies from one to four [4].

Since the $\boldsymbol{x}$ contains the estimated control points in the camera frame and the control points are known in the world frame, the pose can be calculated using the Procrustes Method explained in Section 2.1.6. This comes from the relation in equation 2.47, which states that $\tilde{\boldsymbol{c}}_j^c = \boldsymbol{T}_w^c \tilde{\boldsymbol{c}}_j^w$.

As mentioned in the previous section, one can increase the accuracy of EPnP with Gauss-Newton optimizing. This involves iterating over the β_i found in equation 2.56 until a more accurate solution is found. Though this does increase the computation time, according to [4] the method converges quickly and the method remains at a linear complexity. As will be discussed later, this optimization step is not used in EPnP+GNC.

2.3 Outlier Optimization

Outlier optimization is a statistical method that is used to identify observations that are significantly different from the majority of a data set. These observations are often referred to as "outliers", and they are typically caused by noise or errors in the measurement process. By identifying the outliers it is possible to increase the accuracy of the pose estimation.

There are several algorithms for this sort of optimization, but the ones used in this project are Random Sample Consensus [1] and Graduated Non-Convexity [11]. They are explained in the sections below.

2.3.1 Random Sample Consensus

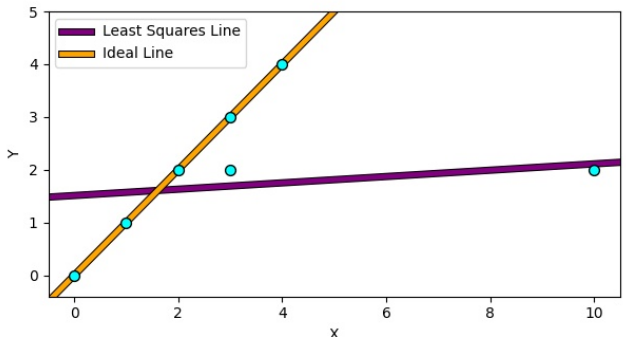
Random Sample Consensus (RANSAC) was introduced by Fischler and Bolles in 1981 [1], and has since then been a much used method in dealing with data sets containing outliers.

RANSAC is an iterative algorithm that works by fitting a model to a subset of the data points. The algorithm starts by selecting a random sample of data points and fitting a model to them. This model is then used to classify all the data points as either inliers or outliers, based on a specified error threshold. The algorithm then repeats this process multiple times, using different random samples and models, until it converges on a final pose estimate that is consistent with the largest number of inliers.

In general, the algorithm can be expressed with five simple steps:

1. Randomly choose $\boldsymbol{s}$ samples from the data set $\boldsymbol{P}$.
2. Fit the model $\boldsymbol{M}$ to the selected samples.
3. Use model $\boldsymbol{M}$ to count the number of points $\boldsymbol{S^\star}$ in $\boldsymbol{P}$ that are within some error tolerance of $\boldsymbol{M}$.
4. Iterate over step 1-3 $\boldsymbol{N}$ times.
5. Choose the model with the largest number of inliers.

In the presentation of RANSAC in [1] the initial example was line modeling using least squares. The points they used are shown in Figure 2.2b.



(a) Plotted points and lines

Points	x	y
1	0	0
2	1	1
3	2	2
4	3	2
5	3	3
6	4	4
7	10	2

(b) Points

Figure 2.2: Points and Lines Used by Fischler and Bolles

As seen in Figure 2.2a, the ideal model line is $x = y$. However, one can see that points 4 and 7 are outliers, and using least squares on the whole set of points therefore yields an incorrect answer. But by using RANSAC, one can find the ideal line and identify the inliers within a few iterations. Table 2.1 presents an example run.

Iteration	Samples	Calculated Model	Inliers	# inliers
1	1,4	y = 0.67 x	1,2,4	3
2	2, 5	y = x	1,2,3,5,6	5
3	1,7	y = 0.2 x	1,7	2
4	3,6	y = x	1,2,3,5,6	5
5	3,4	y = 2	3,4,7	3

Table 2.1: Example RANSAC Iterations

RANSAC has been the tool of choice in many methods since it was presented, mostly due to its simplicity and relatively satisfactory results. This is also true within the field of pose estimation. However, for tasks with a large outlier percentage the method can be very time- and computationally expensive [23]. Therefore it is the hope that GNC can be helpful in that regard.

2.3.2 Graduated Non-Convexity

Graduated Non-Convexity (GNC) is a method of outlier detecting in estimation problems, described in [10], and more recently in [11]. The algoritm is explaned in [11] by first stating the fact that common estimation problems are formulated as least squares optimizations:

$$\min_{\boldsymbol{x}\in\chi} \sum_{i=1}^{N} r^2(\boldsymbol{y}_i, \boldsymbol{x}) \tag{2.57}$$

where $\boldsymbol{y}_i$ are given measurements, $\boldsymbol{x}$ is what is to be estimated and $r(\boldsymbol{y}_i, \boldsymbol{x})$ is the residual error. In the context of this research, the $\boldsymbol{y}_i$ are the pixel values and 3D points while $\boldsymbol{x}$ is the pose.

Equation 2.57 may provide poor estimates when $\boldsymbol{y}_i$ has outliers. Therefore a surrogate cost is introduced, where instead of using the quadratic cost, a *robust cost* ρ_μ is used. This cost function depends on the parameter μ, and will for some values of μ be convex, and for other values be non-convex.

$$\min_{\boldsymbol{x}\in\chi} \sum_{i=1}^{N} \rho_\mu(r(\boldsymbol{y}_i, \boldsymbol{x})) \tag{2.58}$$

According to [11] this robust cost can be chosen from several different cost functions, such as a *Huber* loss, a *Geman-McClure* cost or a *Truncated Least Squares* cost (TLS). In this research the TLS is used.

Furthermore [11] explains that equation 2.58 is equivalent to

$$\min_{\boldsymbol{x}\in\chi, w_i\in[0,1]} \sum_{i=1}^{N} [w_i r^2(\boldsymbol{y}_i, \boldsymbol{x}) + \Phi_\rho(w_i)] \tag{2.59}$$

where $w_i \in [0,1]$ are binary weights and $\Phi_\rho(w_i)$ is a function that defines a penalty on the weight w_i. This result is found using Black-Rangarajan duality [24], and is called the *outlier process*. The penalty $\Phi_\rho(w_i)$ is different depending on the chosen cost function.

Then equation 2.59 can be solved with a alternating optimization; a variable update and a weight update:

$$\boldsymbol{x}^{(t)} = \arg\min_{\boldsymbol{x}\in\chi} \sum_{i=1}^{N} w_i^{(t-1)} r^2(\boldsymbol{y}_i, \boldsymbol{x}) \tag{2.60}$$

$$\boldsymbol{w}^{(t)} = \arg\min_{w_i\in[0,1]} \sum_{i=1}^{N} [w_i r^2(\boldsymbol{y}_i, \boldsymbol{x}^{(t)}) + \Phi_{\rho_\mu}(w_i)] \tag{2.61}$$

This is repeated with increasing or decreasing values of μ, and each iteration intensifies the non-convexity of the cost function. Eventually, $\boldsymbol{w}$ contains the inliers and outliers.

Truncated Least Squares

Truncated Least Squares (TLS) is the cost function used in this research in conjunction with GNC. TLS works by initializing the μ to be small, then increasing it until the cost function resembles a truncated least squares function. This is shown in Figure 2.3.

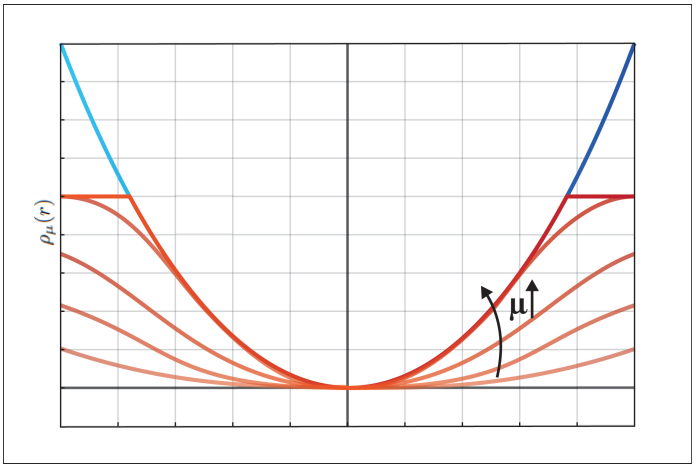


Figure 2.3: Blue: Least squares cost function. Orange: Truncated least squares cost function for different values of μ.

The TLS function is defined as

$$\rho(r) = \begin{cases} r^2 & \text{if } \; r^2 \in [0, \bar{c}^2] \\ \bar{c}^2 & \text{if } \; r^2 \in [\bar{c}^2, +\infty) \end{cases} \tag{2.62}$$

where $\bar{c}$ is the truncation threshold, i.e. the cutoff in the function. Furthermore the GNC surrogate function is defined as

$$\rho_\mu(r) = \begin{cases} r^2 & \text{if } \; r^2 \in [0, \frac{\mu}{\mu+1}\bar{c}^2] \\ 2\bar{c}|r|\sqrt{\mu(\mu+1)} - \mu(\bar{c}^2 + r^2) & \text{if } \; r^2 \in [\frac{\mu}{\mu+1}\bar{c}^2, \frac{\mu+1}{\mu}\bar{c}^2] \\ \bar{c}^2 & \text{if } \; r^2 \in [\frac{\mu+1}{\mu}\bar{c}^2, +\infty) \end{cases} \tag{2.63}$$

and is shown on Figure (2.3) along with the least squares function. According to [11], the function $\Phi_\rho(w_i)$ is defined as

$$\Phi_{\rho_\mu}(w_i) = \frac{\mu(1-w_i)}{\mu + w_i}\bar{c}^2 \tag{2.64}$$

Moreover, the weight update is solved by

$$\boldsymbol{w}^{(t)} = \begin{cases} 0 & \text{if } \; r^2 \in [\frac{\mu+1}{\mu}\bar{c}^2, +\infty) \\ \frac{\bar{c}}{r_i}\sqrt{\mu(\mu+1)} - \mu & \text{if } \; r^2 \in [\frac{\mu}{\mu+1}\bar{c}^2, \frac{\mu+1}{\mu}\bar{c}^2] \\ \bar{c}^2 & \text{if } \; r^2 \in [0, \frac{\mu}{\mu+1}\bar{c}^2] \end{cases} \tag{2.65}$$

Chapter 3

Method

This chapter addresses in detail the activities undertaken in order to achieve the overarching goal introduced in Chapter 1. This includes a derivation of EPnP+GNC, implementation, the data sets, analysis, and the experimental setup.

3.1 EPnP+GNC

The sub-goals of the research rests on whether it is possible to use GNC in combination with EPnP. This is something that has not been done before, and it is not immediately apparent how to do it. In GNC the least square cost function is replaced by a robust cost function. The problem lies with the fact that EPnP does not minimize a least squares cost function, therefore an equivalent then needs to be found.

In order to find a way of combining the methods, the equations presented in Section 2.2.2 and section 2.3.2 were analyzed thoroughly to find a method of doing this.

Using equation 2.50 it is possible to formulate a residual for each point as

$$\boldsymbol{r}_i = \bar{\lambda}_i \tilde{\boldsymbol{p}}_i - \boldsymbol{K} \sum_{j=1}^{4} \alpha_{ij} \, \boldsymbol{c}_j^c \tag{3.1}$$

This gives the loss function

$$L_0 = \sum_{j=1}^{n} \boldsymbol{r}_i^2 = \sum_{j=1}^{n} \left|\left| \bar{\lambda}_i \tilde{\boldsymbol{p}}_i - \boldsymbol{K} \sum_{j=1}^{4} \alpha_{ij} \, \boldsymbol{c}_j^c \right|\right|^2 \tag{3.2}$$

This is then combined with the GNC formulation from Section 2.3.2 to get

$$L = \sum_{j=1}^{n} w_i \boldsymbol{r}_i^2 = \sum_{j=1}^{n} w_i \left|\left| \bar{\lambda}_i \tilde{\boldsymbol{p}}_i - \boldsymbol{K} \sum_{j=1}^{4} \alpha_{ij} \, \tilde{\boldsymbol{c}}_j^c \right|\right|^2 \tag{3.3}$$

This corresponds to a residual

$$\sqrt{w_i} \boldsymbol{r}_i = \sqrt{w_i} \left(\bar{\lambda}_i \tilde{\boldsymbol{p}}_i - \boldsymbol{K} \sum_{j=1}^{4} \alpha_{ij} \, \tilde{\boldsymbol{c}}_j^c \right) \tag{3.4}$$

Just as in Section 2.2.2 this equation gives

$$\sqrt{w_i} \left(\sum_{j=1}^{4} (\alpha_{ij} \, f \, x_j^c + \alpha_{ij} \, (u_0 + u) \, z_j^c) \right) = 0 \tag{3.5}$$

$$\sqrt{w_i} \left(\sum_{j=1}^{4} (\alpha_{ij} \, f \, y_j^c + \alpha_{ij} \, (v_0 + v) \, z_j^c) \right) = 0 \tag{3.6}$$

This is then rearranged into

$$\boldsymbol{W} \boldsymbol{M} \boldsymbol{x} = \boldsymbol{0} \tag{3.7}$$

where $\boldsymbol{M}$ is the same as the $\boldsymbol{M}$ found in Section 2.2.2 and $\boldsymbol{W}$ is a $2n \times 2n$ diagonal matrix such that

$$\boldsymbol{W} = \begin{bmatrix} \sqrt{w_1} & 0 & \cdots & 0 & 0 \\ 0 & \sqrt{w_1} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \sqrt{w_n} & 0 \\ 0 & 0 & \cdots & 0 & \sqrt{w_n} \end{bmatrix} \tag{3.8}$$

This can then be solved the same way regular EPnP is solved as shown in Section 2.2.2. This is then iterated according to the variable update and weight update in GNC until a converging solution is found.

It is noted that this is a new formulation developed in this work, as GNC has not been used in combination with EPnP in previous work.

3.2 Programming

In order to begin the experiments, the implementation of EPnP+GNC must be established. There are several existing implementations of EPnP in different programming languages [22, 25, 26]. However, since the experiments of this project include incorporating additional ideas, it was decided that EPnP had to be implemented from scratch. The resulting implementation is found at [27].

The programming language chosen was Python [28]. There were to major reasons for choosing Python over other languages. Firstly, Python is an exceptionally easy language to write and read, so understanding the code will be easier. Secondly, Python is a general-purpose programming language with several packages and libraries in a variety of areas. In addition to this, the author is more familiar with Python compared to other languages.

There were a few main libraries used. In order to store and manipulate the data the library *NumPy* was used [29]. NumPy adds support for large, multi-dimensional arrays and matrices in Python and adds a large collection of high-level mathematical functions to operate on these arrays. Timing is done using the library Time [30], which adds various time-related functions.

To plot and present the findings the libraries *Matplotlib* [31] and *Open3D* [32] were used. Matplotlib is a plotting library for Python that is made to closely resemble the plotting capabilities found in Matlab, and can create clear and visually pleasing graphs. A limitation with Matplotlib is that it struggles when a large number of points need to be plotted in 3D, and for this reason Open3D was needed. Open3D is an open-source library that adds support to graphical visualization in 3D. Open3D was also used in order to unpack the files containing the 3D models used in the experiments.

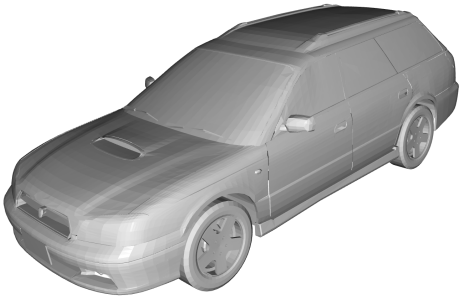
In order to assess both EPnP and EPnP+GNC a comparison is needed. The chosen library to do this was OpenCv [22]. OpenCv is a Python library that has many functionalities pertaining to computer vision and machine learning, including several implemented methods of estimating the PnP problem. In the later sections and chapters the methods implemented in OpenCV will have an OpenCV-prefix, while methods implemented for this research will not have any prefix. I.e. EPnP from OpenCV will be called OpenCV-EPnP, while the implementation from this research will simply be called EPnP.

Much of the code needed for running EPnP was created in the specialization project [33]. For this research, the code used in [33] was used as inspiration and rewritten to be easier to understand. The code was also written to include additional capabilities in order implement the equations for EPnP+GNC found in Section 3.1 as well as additional analytical functionality.

It needs to be stated that the implementation of EPnP used in the project was greatly inspired by the original code from [4]. In addition to this, other implementations were being used as inspiration, notably [22, 26]. Moreover, the implementation of GNC was inspired by the code provided in [34].

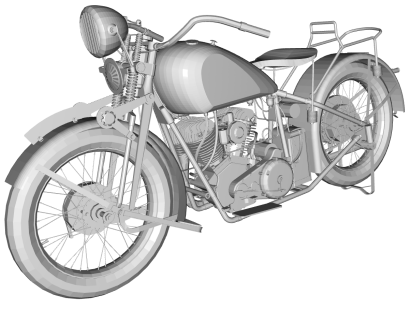
3.3 Data

The data used in the research was *PASCAL3D+*, a data set created in 2014 by Xiang et al. and published by Stanford University [35]. PASCAL3D+ is a data set consisting of several types of data used in 3D object detection and pose estimation.



(a) Car 06

(b) Bicycle 01



(c) Motorbike 04

(d) Sofa 02

Figure 3.1: Example 3D-Models From PASCAL3D+

While the data set has multiple types of data, this project has only used the CAD files provided. The CAD files are 3D models of various objects, with variations in each category. Some examples of the 3D models are shown in Figure 3.1.

3.4 Experimental Setup

3.4.1 Data Preparation

Importing to Numpy

The CAD files provided in PASCAL3D+ are provided as *.off*-files, and Open3D has the capability to turn the files into point clouds. These point clouds are then made into a NumPy array, which then can be further manipulated and used by EPnP+GNC. The vertices are non-homogeneous, so they were made homogeneous. In addition to this, the models contain different amounts of vertices, and in order to test EPnP+GNC it is important to test on known numbers of vertices. A function was made to down-sample the point cloud to a specific amount of vertices. All of these functionalities are shown in Listing 3.1.

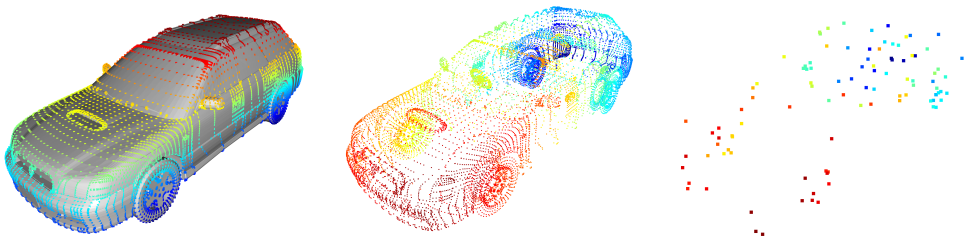
Code listing 3.1: Data Importing and Preparation

```
def load_points_from_file(file_loc):
    if file_loc[-4:] == '.off':
        CAD_off = o3d.io.read_triangle_mesh(file_loc)
        return make_points_homogenous(np.asarray(CAD_off.vertices))

def make_points_homogenous(points):
    if points.shape[1] == 3:
        return np.c_[points, np.ones(points.shape[0])]
    elif points.shape[1] == 4:
        print("Already homogeneous")
        return points

def downsample_points(points, n_corr):
    p = np.random.permutation(len(points))
    points = points[p]
    return points[:n_corr,:]
```

One important note is that the down-sampling function has a randomizing effect. This means that each time the point cloud is down-sampled, a new subset of the points is chosen. The Figure 3.2 shows how the points are spread throughout the 3D-model. It also shows an example of down-sampling to 100 vertices.

Virtual Picture

The PnP problem needs corresponding 3D points and 2D pixels. The 2D points were synthesized from the 3D points by taking a virtual picture. This was done following the procedure shown in Section 2.1.5. A transformation matrix was made, which was used to calculate the pixels. The same transformation was saved for comparison to the estimated pixels later in the experiments.



(a) All vertices on model **(b)** All vertices **(c)** 100 vertices

Figure 3.2: Example of Point Cloud Using Car 06

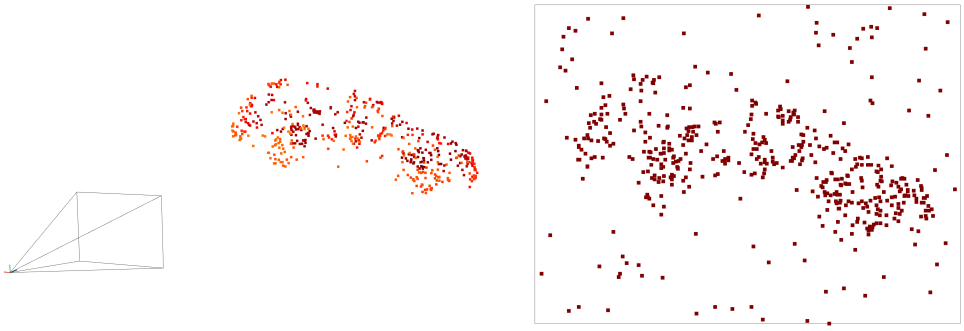
In real life the pixels of a camera are whole numbers, and therefore the calculated pixels were rounded to the closest integer. In addition to this, there is usually noise associated with the capturing of the image, or at least in the detection of corresponding features. Therefore a function that adds synthetic Gaussian noise to the pixels was made, where the desired deviation can be determined. In all of the experiments the Gaussian noise had a standard deviation of five pixels.

The "virtual camera" function also has the capabilities of creating outliers, by inputting the desired percentage of outliers. These outliers got a random pixel value within the size of the image, i.e. $0 \leq u \leq 2u_0$ and $0 \leq v \leq 2v_0$ This is so the experiments can be tested using different amount of outliers. The function for calculating the pixel values, with noise and outliers, is shown in Listing 3.2.

Code listing 3.2: Virtual Camera With Noise and Outlier Generation

```
def compute_pixels(xh_w, T, C, sigma = 0, outlier_percentage = 0):
    sn = (np.eye(3,4) @ T @ xh_w.T).T
    snorm = sn/sn[:,2].reshape((sn.shape[0],1))
    pix = (C @ snorm.T).T
    pix = np.rint(pix)

    # Noise
    if not sigma < 1:
        noise = np.rint(np.random.normal(0, sigma, (pix.shape[0], 2)))
        pix[:,:2] = pix[:,:2] + noise

    # Outliers
    if not outlier_percentage < 1:
        outliers = np.rint(pix.shape[0]*(outlier_percentage/100)).astype(int)
        pix[:outliers,0] = np.random.randint(0,C[0,2]*2, outliers)
        pix[:outliers,1] = np.random.randint(0,C[1,2]*2, outliers)

    return pix
```

An example image is shown in Figure 3.3. In this figure the camera placement is shown as well as an image of the situation with 20% outliers.



(a) Camera Position **(b)** Image with 20% outliers

Figure 3.3: Example of a Virtual Image

Shuffling

When taking an image of an object in real life, the pixels that would become the outliers are likely spread evenly through the image. As shown in Listing 3.2, the first n vertices were used as outliers. Therefore the function 3.3 was implemented to shuffle the points and pixels equally, so the correspondences are still valid.

Code listing 3.3: Point Shuffling

```
def shuffle_points(pix, points):
    assert len(pix) == len(points)
    p = np.random.permutation(len(pix))
    return pix[p], points[p]
```

3.4.2 Method Implementation

All of the code written in this research is available as a GitHub repository [27]. In the repository one can find the implementations of EPnP and EPnP+GNC as well as the files for the experiments, including the experiments themselves and files that add utility. The data sets are also included.

In order to assess EPnP+GNC it was necessary to implement it in a programming language. The equations derived in Chapter 2 pertaining to EPnP and EPnP+GNC were implemented, as well as additional analytical functionality.

At first, the entire process in the EPnP+GN method was tried in the GNC iteration process. However, doing the GN iteration inside GNC iteration increased the running time substantially, and the implementation of the Gauss-Newton (GN) optimization has not yielded satisfactory results. Therefore it was concluded that this step was not to be used in the GNC iterations.

In the GNC iteration process, several parameters needs to be defined. In this research the parameters used were mostly the same as the parameters proposed

in [11]. According to [11] the truncation threshold $\bar{c}$ is supposed to be the max expected error. In this implementation $\bar{c} = 1000$ was used as it resulted in satisfactory results and run time. In addition to this, the initialization of the GNC iterations is a transformation matrix with no rotation and slight translation in the z-axis, specifically $t_z = 4$. The reason this was done will be apparent in the results, see Section 4.3.

In the implementation of EPnP from [33], the transformation matrix that is returned is the best estimation using both the basic EPnP and the Gauss-Newton optimization. In some cases the basic method yields a better estimation, and in other the GN optimization does. When implementing EPnP+GNC in this research the same idea is used. The results of EPnP+GNC might be worse than the previous methods, and a check on the error is used to determine the best pose.

One problem that came about when using EPnP+GNC was that in some cases the reprojection error of EPnP+GNC was higher than the previous methods even when the estimated transformation matrix was more correct. This was solved by recalculating the pose using EPnP with only the inliers calculated using EPnP+GNC. This resulted in dramatically reduced reprojection errors when the estimation was more accurate.

3.4.3 Comparison to OpenCV

As mentioned before, the library OpenCV [22] is used to compare the results of EPnP+GNC to established methods. The methods used in this comparison are shown in Table 3.1.

1	EPnP
2	OpenCV-EPnP
3	OpenCV-SQPnP
4	OpenCV-RANSAC
5	EPnP+GNC

Table 3.1: Methods Used in the Experiments

These methods are chosen as they are the working methods in OpenCV that solve the PnP problem. There are more methods, but these methods are either for specific situations or have broken implementations. OpenCV-EPnP includes the GN optimalization steps. The method OpenCV-RANSAC uses OpenCV-EPnP for estimating the pose using the subset of points required for RANSAC [22].

EPnP+GN is not explicitly used for comparisons, but since the code includes a GN optimization step the result is used in EPnP+GNC. This is because the result of EPnP+GN might be better than EPnP or EPnP+GNC, and therefore could be included in the resulting best estimation.

3.4.4 Analysis

Error Measurement

The equations from Section 2.1.4 were implemented into Python functions. These functions are shown in Listing 3.4. The resulting angular distances and translation distances are plotted in boxplots for each PnP estimation method tested.

Code listing 3.4: Angular Distance and Translation Distance Error Functions

```
def angular_distance_mat(R_true, R_calc):
    R_inc = R_true @ R_calc.T
    return np.degrees(np.arccos(( (np.trace(R_inc) - 1 ) /2 )))

def translation_error(t_true, t_calc):
    return np.linalg.norm(t_true-t_calc)
```

Timing

In addition to testing the angular distance and translation distance of the experiments, the time taken for each method is also calculated. This is done in hopes that it is possible to compare efficiency versus accuracy. It should be noted that the methods implemented in OpenCV have been implemented in the programming language C++, so a direct comparison cannot be done.

There has been work done that tries to quantify the speed differences between C++ and Python [36, 37]. These generally show that C++ is faster than Python, but how much faster depends on the problem and implementation. However, one can generally assume that one can get around 10 to 100 times the speed with a C++ program compared to Python. This problem of speed differences will be discussed more thoroughly in Chapter 5.

3.5 Experiments

To test the applicability of EPnP+GNC, numerous experiments were carried out. Unless otherwise expressed, every experiment was on a point cloud of a CAD-model with 100 vertices. These point cloud were given an increasing percentage of outliers, and the pose was estimated on each iteration with the methods highlighted in Section 3.4.3. The experiments were conducted on a Windows 10 laptop with an Intel Core i7-6700HQ CPU @ 2.60GHz processor and 16 GB of RAM.

In all of the experiments the 2D pixels were calculated using the method explained in Section 3.4.1. Every test was done with the same known camera parameters, shown in Table 3.2. The transformation matrix used to calculate the pixels was also the transformation matrix that was to be estimated using the different PnP-methods.

f	u_0	v_0
800	320	240

Table 3.2: Camera Parameters

3.5.1 Random Transformation

The first experiment was to test a wide number of transformation matrices. For each percentage a different transformation matrix was created using the function in Listing 3.5. It calculates a random transformation matrix with rotation defined by equation 2.13 using a vector of random angles $-\pi \leq \theta \leq \pi$ in x, y and z axis. In addition a random translation $-0.5 \leq t \leq 0.5$ was used. This is true for all axis, except for the z axis, as the translation in this axis needs to be positive in order for a realistic photo to be taken of the point cloud. In other words, one cannot take a picture of something behind the camera.

Code listing 3.5: Transformation Matrix Creation

```
Tr_random = compute_T(
    np.random.uniform(-np.pi, np.pi),   # x-rotation
    np.random.uniform(-np.pi, np.pi),   # y-rotation
    np.random.uniform(-np.pi, np.pi),   # z-rotation
    np.random.uniform(-0.5,0.5),        # x-translation
    np.random.uniform(-0.5,0.5),        # x-translation
    np.random.uniform(-0.5,0.5)*3+10)   # x-translation
```

3.5.2 No Rotation

Another experiment was to see what would happen if the transformation matrix had no rotation included. Here the transformation matrix only had translations, and the rotation was the identity matrix. This is essentially estimating the translation, as the rotation has no impact in the transformation matrix.

While testing this non-rotating transformation matrix, an interesting experiment came to mind. If one was running EPnP+GNC in a real-time situation, the change in the transformation matrix from one point in time $\boldsymbol{T}_k$ to the next $\boldsymbol{T}_{k+1}$ would be minimal.

To test this situation an experiment very similar to the random transformation experiment was conducted. The only difference in the calculated transformation matrices was the angles were tuned down to $-\frac{\pi}{20} \leq \theta \leq \frac{\pi}{20}$ in all axis.

3.5.3 Different Initialization of GNC

While GNC does not require initialization [11], or at least starts with the identity matrix, the first iteration of calculation effectively runs EPnP in its base form. As this is already done previously, one might suggest that initializing the weights

using the already estimated pose would give GNC a better starting point for the resulting iterations.

Therefore, EPnP+GNC was tested with different initialization, specifically by EPnP and OpenCV-EPnP. The reason for choosing these is because they are the best possible guess done before the GNC iteration. In addition to these a transformation matrix only consisting of a positive translation of $t_z = 4$ in the z-axis was tested. This is due to the fact that the object that is imaged needs to be in front of the camera. Initializing GNC with this in mind means that some translation might be beneficial.

3.5.4 Varying Number of Correspondences

As discussed in the previous chapter, the accuracy of EPnP increases with higher number of correspondences. Therefore it was interesting to test on using the same transformation with different amounts of correspondences. The test used a random transformation, where the number of correspondences tested went from 4 up to 200, where each iteration was tested with 25%, 50% and 75% outliers.

In addition to this, one test was done to check the time use of the algorithm when presented with large amount of correspondences. Some of the 3D models contain several thousand vertices, which made this experiment easy to test. The test used a random transformation with 50% outliers, where the number of vertices went from 25 to 2000 in increments of 25.

Chapter 4

Results

This chapter presents the results of the study on the EPnP+GNC method for estimating a pose in an outlier rich data set. The method is tested in various scenarios and compared to other established methods of pose estimation. The implications of the results presented in this chapter will be discussed in the following chapter. The figures presented here are available in grater resolution in the thesis repository [27].

A figure that is much used compares accuracy to outlier percentage. This figure has the angular distance on the top row, and translation distance on the bottom row. Each column represents a different method, explicitly expressed at the top. In each graph the outlier percentage increases from 0% up to 99%.

4.1 Random Transformation

Accuracy

This experiment was conducted in order to test a variety of different transformation matrices, and a different transformation was used for each iteration. The accuracy results are shown in Figure 4.1.

First of all it is interesting to note the differences in EPnP and OpenCV-EPnP. This is because OpenCV-EPnP also used GN optimization, something that is not shown in the EPnP graph. Though GN optimization was implemented for this research, the results using this method was not satisfactory enough to include in the graphs. The reason for this is assumed to be that the implementation was not sufficiently well implemented.

The three first methods are the methods that do not use any form of outlier optimization. In this experiment these methods do not produce exceptional results, although OpenCV-EPnP does work fairly well up to around 30% outliers, where the accuracy is greatly reduced.

The two most interesting graphs are the last methods, namely OpenCV-RANSAC and EPnP+GNC. These methods are the methods used which also incorporate outlier detection. Both methods produce excellent results compared to the methods without outlier optimization. In both of these methods errors start to appear around 40% outliers, but EPnP+GNC has more errors that are outliers in this region. Both methods also start to see significant errors appearing at about 60% outliers. However, in EPnP+GNC the median angular distance and translation distance stays low up till $\sim$ 75% outliers, while the median of OpenCV-RANSAC only stays low up till $\sim$ 65% outliers.

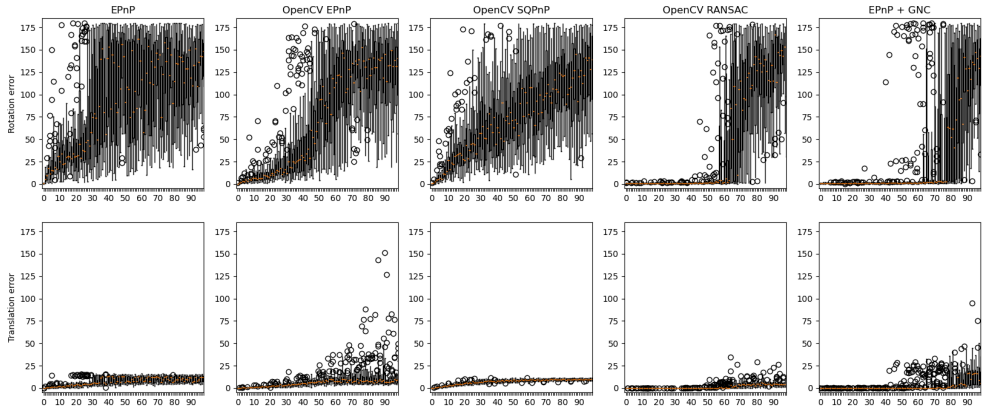


Figure 4.1: Accuracy compared to outlier percentage using a transformation matrix with random parameters.

Running Time

In this experiment the running time of the methods was also measured. The results of this experiment is shown in Figure 4.2. The first thing to note is how fast the methods implemented in OpenCV run compared to the implementations done in this research. In fact, the running time of both OpenCV-EPnP and OpenCV-SQPnP is too fast to accurately sense, and is often measured to be zero. This makes the results difficult to compare, as there is not enough precision to get a good understanding of how much faster OpenCV-EPnP is compared to EPnP.

Another thing to note is how the approximate running time of EPnP+GNC seems to be correct, assuming the time of EPnP is correct. This is because in these test the number of GNC iterations is around 60 to 70. Multiplying 60 iterations by 10ms yields around 0.6 seconds in total, which is in accordance with the results.

The mean running time of OpenCV-RANSAC when the percentage of outliers is above 50 is 10ms, while the running time of EPnP+GNC in the same region is 600ms. This means that in this experiment, OpenCV-RANSAC is 60 times quicker than EPnP+GNC. As described in Chapter 3, one can generally assume that C++ is 10 to 100 times faster than Python. Moreover, the implementation of EPnP+GNC

is not as well optimized as OpenCV-RANSAC. Therefore, one can assume that the methods are comparable in running time, given that they would be given a similar implementation.

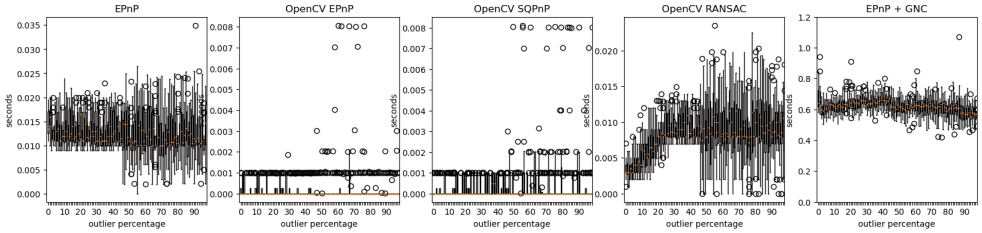


Figure 4.2: Timing the different methods.

4.2 No Rotation

This experiment was conducted in order to see what the pose estimation would be with a transformation matrix without rotation, essentially calculating the translation, or with very small rotations. The results are presented in Figure 4.3 and Figure 4.4.

No Rotation

This result is highly interesting as EPnP+GNC has noticeably good results compared to OpenCV-RANSAC. The results of OpenCV-RANSAC is again very similar to previous results, where significant errors start appearing at around 60% outliers. However, with EPnP+GNC significant errors do not appear before 80% outliers, an increase of 20% outliers!

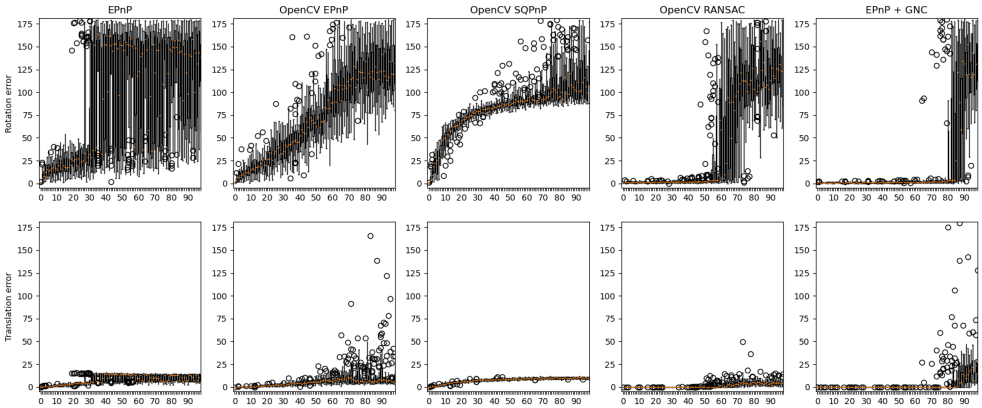


Figure 4.3: Accuracy compared to outlier percentage using a transformation matrix without rotation.

Small Rotation

In the experiment where the methods were tested with transformations with small rotations yielded similar results to the previous test. By comparison to OpenCV-RANSAC, the poses generated by EPnP+GNC are very accurate, even with very high percentage of outliers.

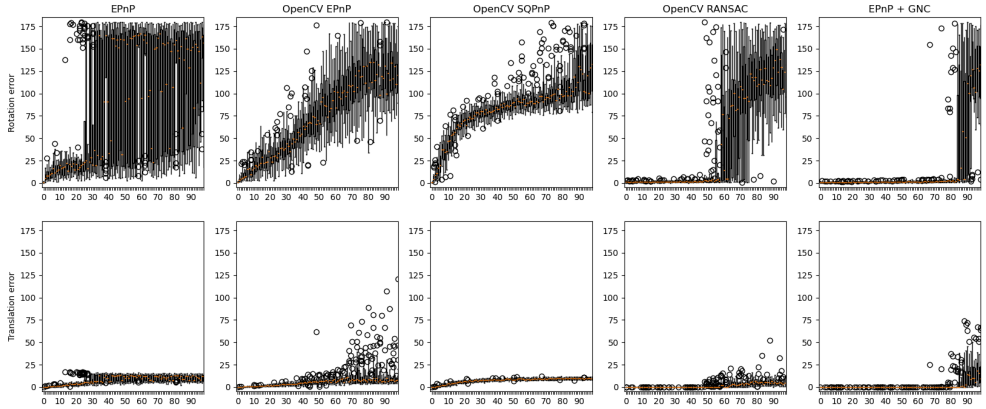


Figure 4.4: Accuracy compared to outlier percentage using a transformation matrix with small rotations.

4.3 Different Initialization of GNC

This experiment was done to see if initializing the GNC iterations with different transformation matrices would have any effect, either on accuracy or running time. The accuracy result is shown in Figure 4.5 and the running time result in Figure 4.6. Note that in the results *Eye* means that the initializing transformation matrix is the 4D identity matrix $\boldsymbol{I}_4$.

Accuracy

When comparing the accuracy of the different initialization there is not much to differentiate. Both in angular distance and translation distance the results are very similar. This suggests that there is not much to gain in the accuracy of the method with different initialization.

Running Time

However, there is an interesting result when comparing the running time of the different initializations. Using $\boldsymbol{I}_4$ as the initialization yields a running time that is up to four times the running time of different initializations. Moreover, it seems that using an initialization without rotation but with some translation is more stable than using the previous best result.

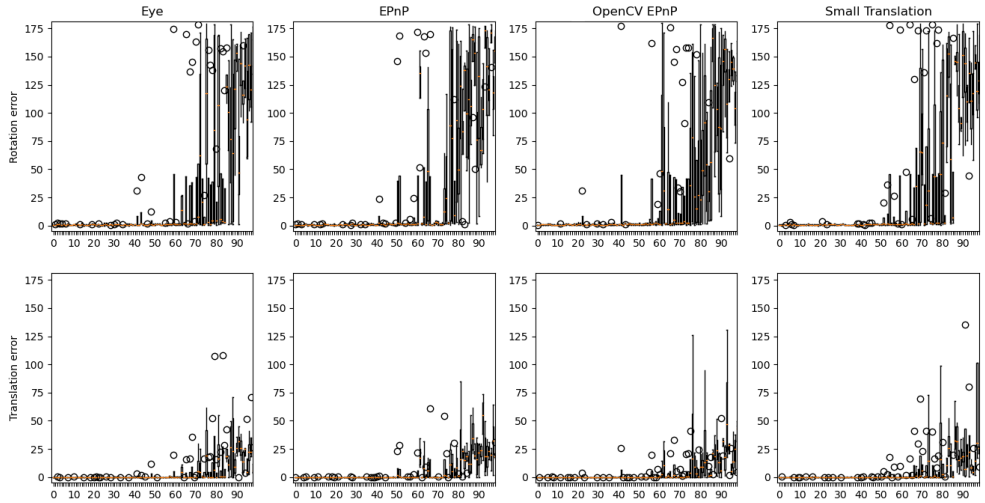


Figure 4.5: Accuracy compared to outlier percentage with different initialization of GNC. Note that Eye means $\boldsymbol{I}_4$.

To explain the difference between using $\boldsymbol{I}_4$ and a transformation with only translation, keep in mind that the PnP problem consists of imaging an object in 3D space. In order for that object to be imaged, it needs to be in front of the camera. This means that using $\boldsymbol{I}_4$ as initialization essentially means initializing the camera at the center of the object. Therefore it is reasonable to think that initializing the camera such that the object is in front of the camera will result in fewer iterations needed to find the correct pose.

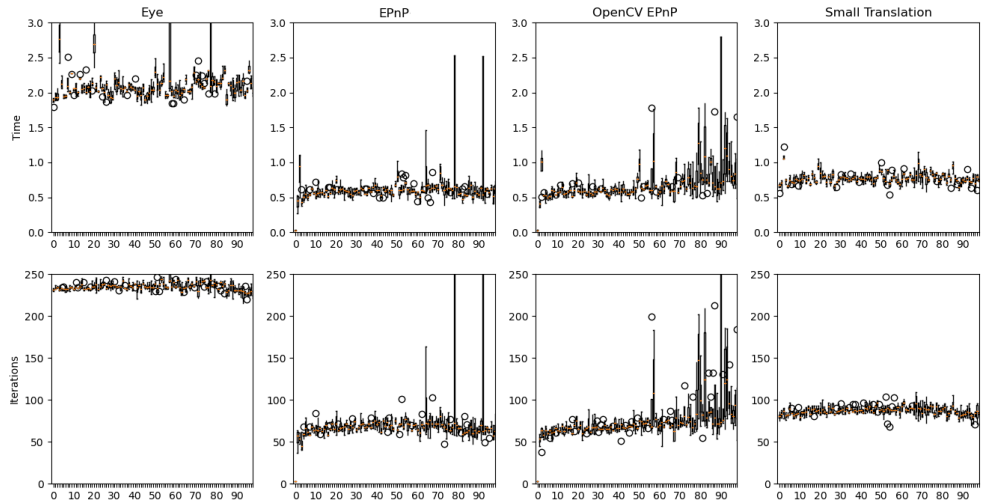


Figure 4.6: Running time and iteration count of GNC with different initializations.

4.4 Varying Number of Correspondences

This experiment was done in order to test the accuracy and running time of EPnP+GNC with different number of correspondences. In the experiments with results shown previously, the number of correspondences was set to 100, while in this experiment the number is varying. The results are shown in Figure 4.7 and Figure 4.8.

Accuracy

As discussed in the background chapter, most methods are more accurate with higher number of correspondences. The results of this experiment yielded the same result, which was expected. More correspondences yield better estimations for all of the outlier percentages tested.

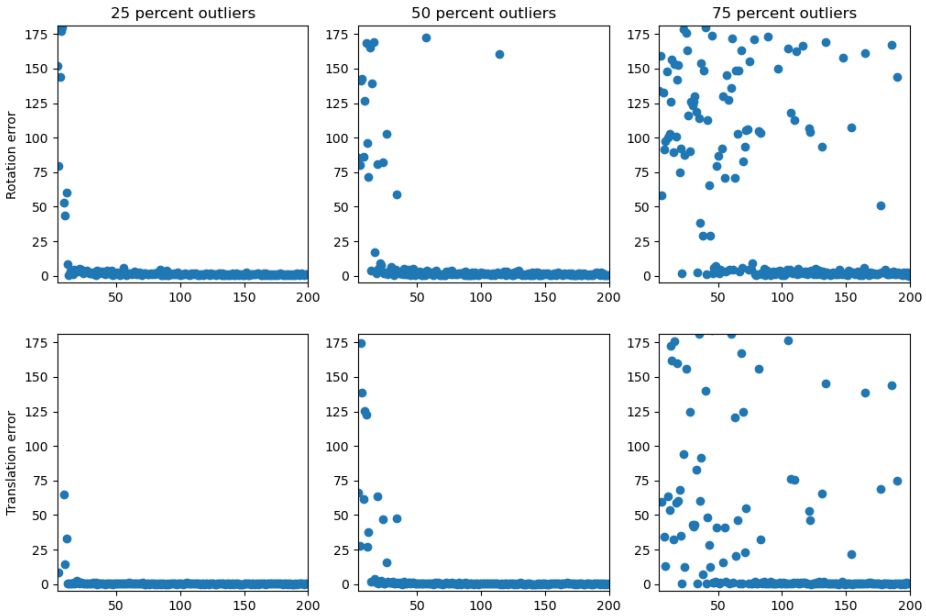


Figure 4.7: Accuracy compared with number of correspondences and outlier percentage.

Time Use

This experiment yielded clear results. When looking at Figure 4.8 it is evident that having more correspondences results in EPnP+GNC needing more time to estimate the pose. One might assume that since EPnP is a method of linear complexity, the time would also be linear. However, this is not the case. This might be explained from the fact that having more correspondences also seems to increase the number of iterations needed in GNC.

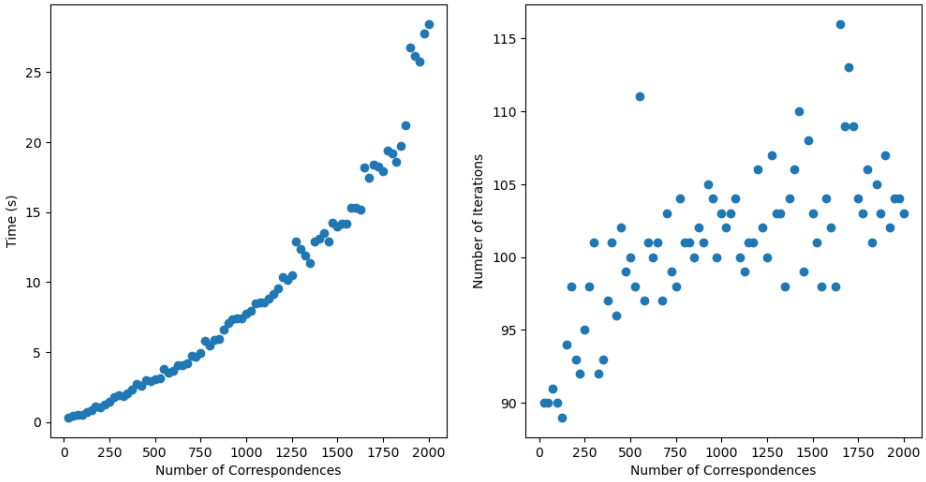


Figure 4.8: Time use and number of iterations compared to number of correspondences.

Chapter 5

Discussion

In this chapter the implications of the results is discussed in relation to the research goal. Different aspects are discussed, mainly accuracy and efficiency. A discussion on the methodology is also included.

5.1 Accuracy

Accuracy is a critical factor to consider when evaluating the potential benefits of the proposed method (EPnP+GNC). In the experiments conducted, the accuracy of EPnP+GNC was compared to other established methods, including EPnP, OpenCV-EPnP, and OpenCV-SQPnP. The results indicated that EPnP+GNC is a more robust method compared to these approaches, which is expected given that none of these methods include outlier optimization.

The more interesting comparison is with OpenCV-RANSAC. Both EPnP+GNC and OpenCV-RANSAC are methods of pose estimation *with* outlier optimization. Overall, it was determined that the accuracy of EPnP+GNC is very similar to that of OpenCV-RANSAC in most cases. This was tested using the random transformation matrix, where the rotation and translation could be very different from iteration to iteration.

In the experiments, it was observed that OpenCV-RANSAC seemed to be a more stable method, as it often produced similar results regardless of changes in the data or transformation. This can be seen by comparing the results of OpenCV-RANSAC to EPnP+GNC in Figures 4.1, 4.3 and 4.4. In these figures, it is clear that OpenCV-RANSAC had very similar results across all experiments, and that EPnP+GNC seemed to be more susceptible to changes in the data and transformation. In addition to this, even though the median accuracy of EPnP+GNC fails at a higher outlier percentage than OpenCV-RANSAC, the accuracy of EPnP+GNC has higher variability than OpenCV-RANSAC, and errors start appearing at a lower outlier percentage.

This could suggest that OpenCV-RANSAC is a more stable method, or at least that it is implemented in a more robust way. Given OpenCV's reputation as a leading computer vision library in Python, the latter possibility seems plausible. A better implementation of EPnP in EPnP+GNC could improve the result.

The most interesting result however was the notable differences in performance in certain cases. When the transformation matrices used to test the methods contained no/small rotations it seemed to favour EPnP+GNC. In these cases, it was found that EPnP+GNC outperformed OpenCV-RANSAC in terms of accuracy. Specifically, EPnP+GNC was able to achieve high accuracy when estimating the pose in data sets consisting of significantly higher outlier percentage. This could indicate that EPnP+GNC is a method that works exceptionally with small rotations, but struggles more and more with larger rotations. This could also explain why the errors appear with lower outlier percentages in EPnP+GNC, as these cases could be cases where the rotation matrix had a large rotation.

The fact that EPnP+GNC in some cases outperforms OpenCV-RANSAC is highly interesting. It means that it might be possible to set up the problem in such a way that one can use this result to get a better estimation using EPnP+GNC compared to OpenCV-RANSAC.

One idea could be to limit how the camera could be moved in relation to the object. If the camera is limited to a small rotation around the object that is photographed, then using EPnP+GNC could be a good method of estimating the pose of the camera in this situation.

Another idea could be to rotate the point cloud that is used in the calculation to a pose that closely resembles the rotation in the actual transformation. If one considers a real-time situation where images are being analyzed continuously, then it might be possible to rotate the 3D object with the previously estimated pose. This would in turn create a situation where the next estimated pose would have minimal rotation, a scenario where EPnP+GNC could excel.

In the end one would probably say that in the general case EPnP+GNC does *not* outperform OpenCV-RANSAC. In some certain cases however, notably in cases with small rotation, one might see some improvements by using EPnP+GNC compared to OpenCV-RANSAC.

5.2 Efficiency

In this research the efficiency is mostly defined by the running time of the methods. As it was briefly discussed in the results chapter, the results from the experiments in regards to the running time is difficult to compare directly.

The main reason for this is because the methods implemented in OpenCV are implemented in the programming language C++, which is a much faster language than Python. This is a problem because the implementation of EPnP and

EPnP+GNC in this research is implemented in Python. This was apparent in the results, where the running time of EPnP and EPnP+GNC were up to several orders of magnitude larger than the implementations from OpenCV.

Another reason that it is difficult to compare the methods is the fact that the methods from OpenCV were so quick that the running time was not measured accurately. More specifically, for OpenCV-EPnP and OpenCV-SQPnP the running time was so quick that for most cases the running time was measured to be zero. When the measurement was not zero, it seemed to land on multiples of 1ms.

The hope of measuring the time was that one might compare EPnP to EPnP+GNC with OpenCV-EPnP to OpenCV-RANSAC. Since the results are not very accurate, the comparison will not be very accurate as well. Using the smallest timed result, OpenCV-EPnP runs at 1ms for outlier percentages over 50%. In the same region, the median running time of OpenCV-RANSAC is just under 10ms, ten times as slow. If one does the same comparison of the methods implemented in this work, the difference is about 60 times slower running time with EPnP+GNC compared to EPnP. This means that EPnP+GNC compared to OpenCV-RANSAC is around six times slower. Again, this comparison is uncertain, as one cannot do a real comparison when the implementations are so different.

However, this might not be detrimental to the results. When one considers that EPnP+GNC could be six times slower than OpenCV-RANSAC, it still means that EPnP+GNC could have a running time of around 60ms. This is definitely fast enough to potentially being used in a real application, with a frequency of 16Hz. In addition to this the implementation of EPnP and EPnP+GNC in this research includes several analytical tools and unimportant parts that decrease the efficiency of the implementations. If the methods were implemented in a more efficient manner, using optimised functions or equations, one might assume that the running time could be reduced.

As a summary, the running time of EPnP+GNC is comparable to that of OpenCV-RANSAC. The experiments were however not conclusive, as there were issues in how the methods are implemented and timed.

5.3 Varying Number of Correspondences

In a real life situation one cannot always know beforehand how many point correspondences one might end up using. Therefore it is important to know how EPnP+GNC deals with different numbers of correspondences.

The first result that was apparent from the experiments was the fact that the accuracy tended to increase with higher number of correspondences. This was true for all percentages of outliers tested. This was to be expected, as this is true for most pose estimation methods.

The second thing to note is how the running time increases when increasing the number of correspondences. The points in Figure 4.8 seem to indicate an higher than linear increase in running time with increasing number of correspondences. This is slightly unexpected, as EPnP is a method of linear complexity. However, this can be partially explained with the fact that an increasing number of correspondences also need more iterations to converge to a result.

Just as with running time in the general case, this result is not necessarily representative of an implementation in a real application. EPnP+GNC was as previously discussed implemented with several unnecessary tools and functions for analysis, which are probably negatively impacting. Since the computer is doing several runs of this experiment, all with increasing numbers of correspondences, the computer might also need to allocate resources that otherwise would not be necessary, such as more memory.

All in all, this experiment ended up having the expected results. Higher number of correspondences lead to longer running times. It was slightly unexpected how fast the running time increased, but more testing would need to be done in order to fully determine the true result.

5.4 The Methodology

The methodology of the study involved comparing the performance of the proposed method EPnP+GNC to several established pose estimation methods, including EPnP, OpenCV-EPnP, OpenCV-SQPnP and OpenCV-RANSAC. To do this, synthetic 3D point clouds and corresponding 2D projections were generated and used to simulate different pose configurations. Random transformation matrices were used to test the performance of the methods under different conditions. The evaluation metrics used to measure the accuracy of the pose estimates included the angular distance and translation distance.

There were several issues with the methodology of the study. Firstly, if more 3D models had been used a more comprehensive result could have been achieved. The models used were chosen because they had a fairly uniform distribution of points. In this regard uniform distribution means that the vertices were not grouped together, such as in wheels or writing. This in turn meant that the down-sampled point cloud would also be fairly uniform in distribution. This does mean that the results of the study might be unique to the 3D models used, and not a general result. If the experiments were to be conducted again, improvements could include the use of a larger and more diverse range of data sets, including both synthetic and real-world data. This would allow for testing the performance of the methods under a wider range of conditions and provide a more comprehensive evaluation.

In relation to this, the experiments should have prioritized randomizing the vertices some more. In each experiment, when the point cloud was prepped, a subset of the vertices in the 3D model were chosen. These vertices were then used for all of the tests in that specific experiment. If, by chance, the vertices chosen were especially fitting for one method over another, it would not be possible to show the discrepancy in the results. If one were to do the experiments again, either a different subset of the 3D model should be used for each test, or a completely different 3D model.

In addition to this, it is important to discuss the Gauss-Newton optimization in the implementation of EPnP done for this research. As it was apparent in the results, the results are different in EPnP and OpenCV-EPnP. This is because the results of OpenCV-EPnP include the GN optimization, while EPnP does not. The reason for this is because the implementation of GN in EPnP is not satisfactory. If one were to implement GN correctly in EPnP+GNC, the results would probably be more impressive.

Even though there are reasons to believe that the results are not necessarily fully representative of reality, they do in fact underline the potential of EPnP+GNC. Especially in the transformations with no rotation or small rotations, where the different experiments used different subgroups of the vertices. This strengthens the idea that EPnP+GNC might be a good method to use in these scenarios. Nonetheless, further tests need to be done in order to fully examine the potential of EPnP+GNC.

Chapter 6

Conclusion

This chapter presents a summary of the research and some concluding remarks, such as suggestions for further research.

6.1 Conclusion

The overarching goal of this thesis has been to run a feasibility study on the potential benefits of combining Effective Perspective-n-Point (EPnP) and Graduated Non-Convexity (GNC). By combining these methods, the resulting method EPnP+GNC would presumably increase the robustness of the pose estimation by EPnP with the outlier registration by GNC.

In order to test this hypothesis, several experiments to assess the properties of EPnP+GNC were conducted. EPnP+GNC was compared to several established methods for pose estimation in varying circumstances. How EPnP+GNC reacts to various number of correspondences was also studied. The data used in the study was synthetic 3D point clouds and corresponding 2D projections.

The results of the study showed that EPnP+GNC is a more robust method compared to EPnP, OpenCV-EPnP, and OpenCV-SQPnP. This was expected, considering EPnP+GNC includes outlier optimization while the aforementioned do not. In comparison to OpenCV-RANSAC, EPnP+GNC achieved similar accuracy in most cases, but outperformed RANSAC in cases where the transformation matrices contained no/small rotations. The results also indicated that EPnP+GNC might be comparable in running time compared RANSAC, but this was difficult to assess.

Overall, the results of this study suggest that EPnP+GNC is a promising method for pose estimation in the Perspective-n-Point problem. The method does in fact work, as the accuracy of the pose estimation works well with high percentages of outliers. While there were some limitations in the experimentation process,

such as the limited data set usage and few methods of comparison, the proposed method demonstrated potential for further development and application in real-world scenarios.

6.2 Further Work

The research conducted in this thesis has led to the proposal of a set of suggestions for further work on the topic of EPnP+GNC. These suggestions are based on the findings and insights gained from the research, and are intended to identify areas for future study and investigation that will help to advance the understanding and knowledge of EPnP+GNC. It is hoped that these suggestions will provide valuable guidance for future research efforts in this area.

- To improve the efficiency of the method, it may be useful to implement it in a faster language and explore optimization techniques. Additionally, it could be interesting to examine new strategies for combining EPnP and GNC.
- Incorporate a wider range of data sets, both synthetic and real world data, into the testing process to improve the reliability and validity of the results. In addition to this it is important to test EPnP+GNC in a wide range of conditions and situations in order to better understand its capabilities and limitations and to determine its suitability for different environments and scenarios.
- In order to get a more comprehensive understanding of how EPnP+GNC compares to other state-of-the-art methods of pose estimation in the PnP problem, it would be useful to compare EPnP+GNC to more methods than just RANSAC. As OpenCV-RANSAC is the only robust pose estimation method used for comparison in this research, it is difficult to get conclusive results.
- Investigating EPnP+GNC's potential advantage in handling minimal rotations in the estimated poses. This could involve studying the performance of EPnP+GNC in such situations and identifying potential applications where its ability to handle minimal rotations could be beneficial.
- Exploring how changing different parameters in GNC might affect the end result in terms of accuracy or run time. Understanding how changing different parameters might affect the end result could help to optimize the performance of EPnP+GNC and identify the most effective configurations for different applications.

Bibliography

[1] M. Fischler and R. Bolles, ‘ Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography,’ *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[2] D. A. Forsyth and J. Ponce, *Computer Vision - A Modern Approach, Second Edition*. Pitman, 2012, pp. 1–791, ISBN: 978-0-273-76414-4.

[3] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2003, ISBN: 0521540518.

[4] V. Lepetit, F. Moreno-Noguer and P. Fua, ‘EPnP: An Accurate O(n) Solution to the PnP Problem.,’ *Int. J. Comput. Vis.*, vol. 81, no. 2, pp. 155–166, 2009.

[5] G. Terzakis and M. Lourakis, ‘A Consistently Fast and Globally Optimal Solution to the Perspective-n-Point Problem,’ Sep. 2020.

[6] L. Kneip, D. Scaramuzza and R. Siegwart, ‘A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation,’ *CVPR 2011*, pp. 2969–2976, Jan. 2011.

[7] S. Urban, J. Leitloff and S. Hinz, ‘MLPnP - A Real-Time Maximum Likelihood Solution to the Perspective-n-Point Problem,’ *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III-3, pp. 131–138, Jun. 2016. DOI: 10.5194/isprs-annals-III-3-131-2016.

[8] L. Fc, X. Binefa and F. Moreno-Noguer, ‘Very Fast Solution to the PnP Problem with Algebraic Outlier Rejection,’ *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 2014. DOI: 10.1109/CVPR.2014.71.

[9] R. Sheffer and A. Wiesel, ‘PnP-Net: A hybrid Perspective-n-Point Network,’ Mar. 2020.

[10] A. Blake and A. Zisserman, *Visual reconstruction*, eng, Cambridge, Massachusetts, 2003.

[11] H. Yang, P. Antonante, V. Tzoumas and L. Carlone, ‘Graduated Non-Convexity for Robust Spatial Perception: From Non Minimal Solvers to Global Outlier Rejection,’ *IEEE Robotics Autom. Lett.*, vol. 5, no. 2, pp. 1127–1134, 2020.

[12] M. Weinmann, *Reconstruction and Analysis of 3D Scenes - From Irregularly Distributed 3D Points to Object Classes*. Apr. 2016, ISBN: 978-3-319-29244-1. DOI: 10.1007/978-3-319-29246-5.

[13] M. Daneshmand, A. Helmi, E. Avots, F. Noroozi, F. Alisinanoglu, H. Arslan, J. Gorbova, R. Haamer, C. Ozcinar and G. Anbarjafari, '3D Scanning: A Comprehensive Survey,' Jan. 2018.

[14] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*, 1st. USA: Cambridge University Press, 2017, ISBN: 1107156300.

[15] R. Hartley, J. Trumpf, Y. Dai and H. li, 'Rotation Averaging,' *International Journal of Computer Vision*, vol. 103, Jul. 2013. DOI: 10.1007/s11263-012-0601-0.

[16] K. S. Arun, T. S. Huang and S. D. Blostein, 'Least-Squares Fitting of Two 3-D Point Sets,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 698–700, 1987. DOI: 10.1109/TPAMI.1987.4767965.

[17] S. Umeyama, 'Least-squares estimation of transformation parameters between two point patterns,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991. DOI: 10.1109/34.88573.

[18] X.-S. Gao, X.-R. Hou, J. Tang and H.-F. Cheng, 'Complete solution classification for the perspective-three-point problem,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 930–943, 2003. DOI: 10.1109/TPAMI.2003.1217599.

[19] G. Nakano, 'Globally Optimal DLS Method for PnP Problem with Cayley parameterization,' Jan. 2015, pp. 78.1–78.11. DOI: 10.5244/C.29.78.

[20] S. Li, C. Xu and M. Xie, 'A Robust O(n) Solution to the Perspective-n-Point Problem,' *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, Jan. 2012. DOI: 10.1109/TPAMI.2012.41.

[21] Z. Kukelova, M. Bujnak and T. Pajdla, 'Automatic Generator of Minimal Problem Solvers,' in *Computer Vision – ECCV 2008*, D. Forsyth, P. Torr and A. Zisserman, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 302–315, ISBN: 978-3-540-88690-7.

[22] G. Bradski, 'The OpenCV Library,' *Dr. Dobb's Journal of Software Tools*, 2000.

[23] G. Truong, H. Le, Á. Parra, S. Z. Gilani, S. Islam and D. Suter, 'Fast Semantic-Assisted Outlier Removal for Large-scale Point Cloud Registration,' Feb. 2022. [Online]. Available: https://www.researchgate.net/publication/358795755_Fast_Semantic-Assisted_Outlier_Removal_for_Large-scale_Point_Cloud_Registration.

[24] M. Black and A. Rangarajan, 'On the Unification Line Processes, Outlier Rejection, and Robust Statistics with Applications in Early Vision,' *International Journal of Computer Vision*, vol. 19, pp. 57–91, Jul. 1996. DOI: 10.1007/BF00131148.

[25] CVLAB @ EPFL, *EPnP_Code*. [Online]. Available: https://github.com/cvlab-epfl/EPnP.git.

[26] WeiyanCai, *EPnP_Python*. [Online]. Available: https://github.com/WeiyanCai/EPnP_Python.git.

[27] R. Jåtun, *EPnP With GNC for Outlier Robustness - Source Code*. [Online]. Available: https://github.com/runarjaa/TPK4960-EPnP_with_GNC_for_Outlier_Robustness.git.

[28] *Python*. [Online]. Available: https://www.python.org/.

[29] *NumPy*. [Online]. Available: https://www.numpy.org/.

[30] *Python Time Library*. [Online]. Available: https://docs.python.org/3/library/time.html.

[31] *Matplotlib*. [Online]. Available: https://www.matplotlib.org/.

[32] *Open3D*. [Online]. Available: http://www.open3d.org/.

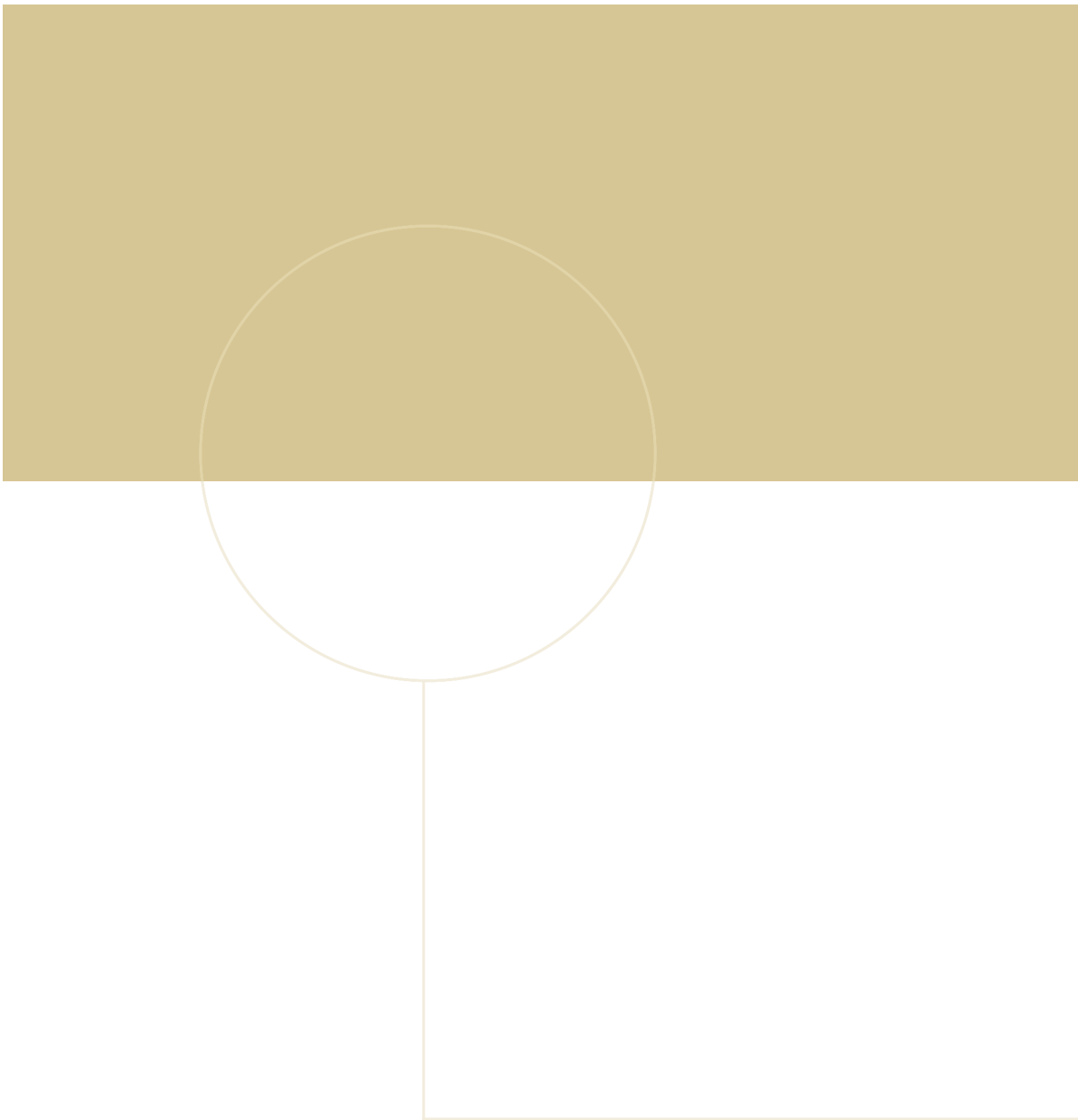
[33] R. Jåtun, *TPK4560 Project Pose Estimation*. [Online]. Available: https://github.com/runarjaa/22v-TPK4560-Prosjektoppgave-Pose_estimation.

[34] O. Egeland, *Registration*, Working notes.

[35] Y. Xiang, R. Mottaghi and S. Savarese, 'Beyond PASCAL: A Benchmark for 3D Object Detection in the Wild,' in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014.

[36] F. Zehra, M. Javed, D. Khan and M. Pasha, 'Comparative Analysis of C++ and Python in Terms of Memory and Time,' Dec. 2020. DOI: 10.20944/preprints202012.0516.v1.

[37] P. Fua and K. Lis, 'Comparing Python, Go, and C++ on the N-Queens Problem,' Jan. 2020.



NTNU

Norwegian University of
Science and Technology