Pose Estimation

Xiao Hu, emails: xiahaa@space.dtu.dk

March 5, 2019

Contents

1 3D to 3D case **2**

1.1 Problem Formulation . . . 2

1.2 Solution . . . 2

1.2.1 SVD . . . 2

1.2.2 Quaternion . . . 4

1.3 Summary . . . 6

2 3D to 2D case **7**

2.1 Introduction . . . 7

2.2 Theory and Implementation . . . 9

2.2.1 P3P . . . 10

2.2.2 Complex PnP . . . 12

2.2.3 POSIT & SoftPOSIT . . . 13

2.2.4 Iterative PnP . . . 15

2.3 Convex PnP . . . 17

2.4 $\mathcal{O}(n)$ PnP . . . 17

2.4.1 Algebraic Geometry PnP . . . 17

2.4.2 RPnP . . . 18

2.4.3 EPnP . . . 20

2.4.4 EPPnP . . . 21

2.4.5 REPPnP . . . 21

2.4.6 CEPPnP . . . 22

2.5 Very New PnP solution . . . 23

2.5.1 R1PPnP . . . 23

2.5.2 HARD-PnP . . . 24

3 2D to 2D case **25**

3.1 Essential Matrix . . . 25

3.1.1 Eight Point Algorithm . . . 25

3.1.2 Five Point Algorithm . . . 26

3.1.3 Normalization . . . 27

3.1.4 Decomposition . . . 27

Chapter 1

3D to 3D case

In this chapter, we will talk about general solutions for 3D to 3D pose estimation problem. Nonlinear optimization solution will be described later.

1.1 Problem Formulation

Let $\mathbf{P}_i$ and $\mathbf{Q}_i$ $(i = 1,\ 2,\ ,\ \cdots,\ n)$ be two sets of corresponding points in $\mathbb{R}^3$. The aim is to find a rigid transformation that align optimally the two sets in the least square sense:

$$\mathbf{R}, \mathbf{t} = \underset{\mathbf{R}\in\mathbb{SO}(3),\ \mathbf{t}\in\mathbb{R}^3}{argmin} \underbrace{\sum_{i=1}^{n} w_i ||\mathbf{R}\mathbf{P}_i + \mathbf{t} - \mathbf{Q}_i||^2}_{E}$$

where w_i denotes the corresponding weight for each point pair.

1.2 Solution

The following introduced solutions were documented in [1] and [2]. A good tutorial can be found in [3].

1.2.1 SVD

Translation It is to see that the objective function is defined in a quadratic form on $\mathbf{t}$. Consequently, if we assume $\mathbf{R}$ is known, $\mathbf{t}$ can be easily found by taking the derivative and letting it equal to zero (first-order optimality condition for differentiable function):

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{t}} &= \sum_{i=1}^{n} 2w_i(\mathbf{R}\mathbf{P}_i + \mathbf{t} - \mathbf{Q}_i) = 0 \Leftrightarrow \\ \mathbf{R}\underbrace{\frac{\sum_{i=1}^{n} w_i\mathbf{P}_i}{\sum_{i=1}^{n} w_i}}_{\bar{\mathbf{P}}} - \underbrace{\frac{\sum_{i=1}^{n} w_i\mathbf{Q}_i}{\sum_{i=1}^{n} w_i}}_{\bar{\mathbf{Q}}} &= -\mathbf{t} \Leftrightarrow \\ \bar{\mathbf{Q}} - \mathbf{R}\bar{\mathbf{P}} &= \mathbf{t}\end{aligned}$$

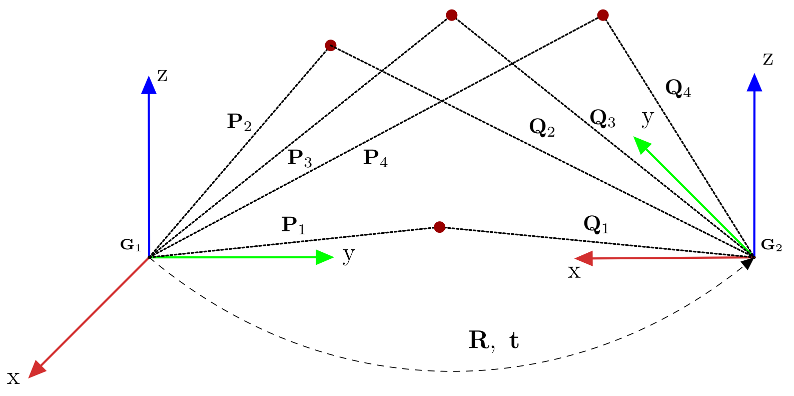


Figure 1.1: Principles of the 3D to 3D pose estimation problem.

Plug this into the objective function, we will have:

$$E = \sum_{i=1}^{n} w_i ||\mathbf{R}(\mathbf{P}_i - \bar{\mathbf{P}}) - (\mathbf{Q}_i - \bar{\mathbf{Q}})||^2 \Leftrightarrow$$

$$E = \sum_{i=1}^{n} w_i ||\mathbf{R}\mathbf{x}_i - \mathbf{y}_i||^2,\ \mathbf{x}_i = \mathbf{P}_i - \bar{\mathbf{P}},\ \mathbf{y}_i = \mathbf{Q}_i - \bar{\mathbf{Q}}$$

Rotation Expand one element as

$$||\mathbf{R}\mathbf{x}_i - \mathbf{y}_i||^2 = (\mathbf{R}\mathbf{x}_i - \mathbf{y}_i)^T(\mathbf{R}\mathbf{x}_i - \mathbf{y}_i) = \mathbf{x}_i^T\mathbf{x}_i - 2\mathbf{y}_i^T\mathbf{R}\mathbf{x}_i + \mathbf{y}_i^T\mathbf{y}_i$$

It can be seen that the first and third terms are constant, so it is equivalent to optimize only the second term

$$\mathbf{R} = \underset{\mathbf{R}\in\mathbb{SO}(3)}{argmax} \underbrace{\sum_{i=1}^{n} w_i \mathbf{y}_i^T \mathbf{R}\mathbf{x}_i}_{E'}$$

The new objective function E' can be written as

$$E' = trace(\mathbf{WYRX}) = \underbrace{diag(w_i)}_{\mathbf{W}_{n\times n}} \underbrace{\begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_n^T \end{bmatrix}}_{\mathbf{Y}_{n\times 3}} \mathbf{R} \underbrace{\begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \end{bmatrix}}_{\mathbf{X}_{3\times n}}$$

Recall that $trace\mathbf{AB} = trace\mathbf{BA}$, so

$$E' = trace(\mathbf{WYRX}) = trace(\mathbf{RXWY})$$

Use a new matrix $\mathbf{S}_{3\times 3} = \mathbf{XWY}$ and take the SVD of as $\mathbf{S} = \mathbf{U\Sigma V}^T$

$$E' = trace(\mathbf{RS}) = trace(\mathbf{RU\Sigma V}^T) = trace(\mathbf{\Sigma V^T R U})$$

where $\mathbf{V},\ \mathbf{R},\ \mathbf{U}$ are all unitary matrix. So their product is also an unitary matrix. Assuming their product is $\mathbf{M} = [\mathbf{m}_{(:,1)}\ \mathbf{m}_{(:,2)}\ \mathbf{m}_{(:,3)}]$, then we have

$$\mathbf{m}_{(:,i)}^T \mathbf{m}_{(:,i)} = \sum_{j=1}^{3} m_{ji}^2 = 1 \Rightarrow |m_{ji}| \leq 1$$

Recall the new objective function

$$\mathbf{R} = \underset{\mathbf{R}\in\mathbb{SO}(3)}{argmax}\ trace(\mathbf{\Sigma M}) \Leftrightarrow$$
$$\mathbf{R} = \underset{\mathbf{R}\in\mathbb{SO}(3)}{argmax}\ \sum_i \sigma_i m_{ii}$$

In order to maximize this objective function, we need to have $m_{ii} = 1$. Then, we have

$$\mathbf{I} = \mathbf{M} = \mathbf{V^T R U} \Leftrightarrow$$
$$\mathbf{R} = \mathbf{V}\mathbf{U}^T$$

This optimization omits one constraint imposed by $\mathbb{SO}(3)$, i.e. $det(\mathbf{R}) = 1$, since the result is only an orthogonal matrix, not a special orthogonal matrix. Actually, it could be a rotation matrix or a reflection matrix. In case we find a reflection matrix $det(\mathbf{R}) = -1$, we need to recover from this reflection matrix a rotation matrix:

$$\mathbf{R} = \mathbf{V}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(\mathbf{V}\mathbf{U}^T) \end{bmatrix}\mathbf{U}^T$$

This is the best since we only perturbate the smallest singular value.

The algorithm

- find $\bar{\mathbf{P}}$ and $\bar{\mathbf{Q}}$.
- form $\mathbf{x},\ \mathbf{y}$.
- find optimal $\mathbf{R}$.
- find optimal $\mathbf{t}$.
- non-linear optimization.

1.2.2 Quaternion

This closed-form pose estimation solution using unit-quaternion was firstly proposed in [2].

Quaternion The left and right matrix of a quaternion $\mathbf{q} = \{w,\ x,\ y,\ z\} = \{w, \mathbf{x}^T\}$ is given as

$$\mathbf{L_q} = \begin{bmatrix} w & -\mathbf{x}^T \\ \mathbf{x} & w\mathbf{I} + skew(\mathbf{x}) \end{bmatrix}$$
$$\mathbf{R_q} = \begin{bmatrix} w & -\mathbf{x}^T \\ \mathbf{x} & w\mathbf{I} - skew(\mathbf{x}) \end{bmatrix}$$
$$skew(\mathbf{x}) = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

By using quaternion, we also have the following property:

$$\mathbf{pq} \cdot \mathbf{r} = \mathbf{p} \cdot \mathbf{rq}^*$$

where $\mathbf{q}^* = \{w,\ -x,\ -y,\ -z\}$ is the conjugate quaternion of $\mathbf{q}$.

Rotation Similar to the SVD solution, the first two step is also to find $\bar{\mathbf{P}}$, $\bar{\mathbf{Q}}$, $\mathbf{x}$, and $\mathbf{y}$. Then a new objective function is defined as

$$\mathbf{R} = \underset{\mathbf{R} \in \mathbb{SO}(3)}{argmax} \sum_{i=1}^{n} \mathbf{y} \cdot \mathbf{Rx}$$

This can be seen as an optimization which best align the rays generated by one point pair since the $\cdot()$ operation is to find the projection of one vector to other. Using the corresponding quaternion of $\mathbf{R}$, it can be further written as

$$\mathbf{R} = \underset{\mathbf{R} \in \mathbb{SO}(3)}{argmax} \sum_{i=1}^{n} \mathbf{y} \cdot \mathbf{Rx} = \underset{\mathbf{R} \in (\mathbb{SO}(3)}{argmax} \sum_{i=1}^{n} \mathbf{qq_{x_i}} \cdot \mathbf{q_{y_i}q} = \underset{\mathbf{R} \in (\mathbb{SO}(3)}{argmax} \sum_{i=1}^{n} \mathbf{q}^T \mathbf{R}_{\mathbf{q_{x_i}}}^T \mathbf{L_{q_{y_i}}} \mathbf{q}$$

where $\mathbf{q_x}$ is the corresponding quaternion of $\mathbf{x}$ by making $w = 0$. We use $\mathbf{N} = \sum_{i=1}^{n} \mathbf{R}_{\mathbf{q_{x_i}}}^T \mathbf{L_{q_{y_i}}}$ and there is another convenient way of computing $\mathbf{N}$:

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{x_i y_i^T}$$

$$\mathbf{M} = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix}$$

$$\mathbf{N} = = \begin{bmatrix} S_{xx} + S_{yy} + S_{zz} & S_{yz} - S_{zy} & S_{zx} - S_{xz} & S_{xy} - S_{yx} \\ S_{yz} - S_{zy} & S_{xx} - S_{yy} - S_{zz} & S_{xy} + S_{yx} & S_{zx} + S_{xz} \\ S_{zx} - S_{xz} & S_{xy} + S_{yx} & -S_{xx} + S_{yy} - S_{zz} & S_{yz} + S_{zy} \\ S_{xy} - S_{yx} & S_{zx} + S_{xz} & S_{yz} + S_{zy} & -S_{xx} - S_{yy} + S_{zz} \end{bmatrix}$$

Now the objetcive function is in a quadratic form of $\mathbf{q}$. Since $\mathbf{N}$ is symmetric and positive-definite, we apply the Eigen-value decomposition to $\mathbf{N}$, i.e. $\mathbf{N} = \mathbf{U \Sigma U}^T$. Let $\mathbf{q}' = \mathbf{U}^T\mathbf{q} \Leftrightarrow \mathbf{q} = \mathbf{Uq}'$, so $\mathbf{q}$ can be seen as a linear combination of the basis spanned by $\mathbf{U}$

$$\mathbf{q} = \alpha_1 \mathbf{e}_1 + \alpha_2 \mathbf{e}_2 + \alpha_3 \mathbf{e}_3 + \alpha_4 \mathbf{e}_4,\ \ where\ \mathbf{U} = [\mathbf{e}_1,\ \mathbf{e}_2,\ \mathbf{e}_3,\ \mathbf{e}_4], \mathbf{q}' = [\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4]^T$$

So $\mathbf{q}^T\mathbf{Nq} = \mathbf{q}'^T \mathbf{\Sigma q}' = \sum \alpha_i^2 \lambda_i$. Recall the unit-quaternion has a constraint on its norm, i.e. $||\mathbf{q}|| = 1$, it is easy to validate that $||\mathbf{q}'|| = 1$. Together with $\lambda_1 \geq \lambda_i, i \neq 1$

$$\sum \alpha_i^2 \lambda_i \leq \lambda_1$$

The maximum will be achieved when $\alpha_1 = 1, \alpha_i = 0, i \neq 1$. Consequently, $\mathbf{q} = \mathbf{e}_1$ which corresponds to the eigen vector of the maximum eigen value. $\mathbf{t}$ can be found in a similar way to the SVD solution.

The algorithm

- find $\bar{\mathbf{P}}$ and $\bar{\mathbf{Q}}$.
- form $\mathbf{x}$, $\mathbf{y}$.
- form $\mathbf{M}$, $\mathbf{N}$.
- find $\mathbf{R}, \mathbf{t}$
- non-linear optimization.

1.3 Summary

There are actually other improvements or variants, like [4], [5], [6] [7]. A fairly detailed comparison of four methods was given in [8].

Several key points from [8]:

- SVD and Unit-Quaternion method performs similarly, better than others.
- Unit-Quaternion method is faster than SVD based solution.

Chapter 2

3D to 2D case

2.1 Introduction

The Perspective-n-Point (PnP) problem is a fundamental problem in geometric computer vision. Given a certain number (n) of correspondences between 3D world points and 2D image measurements, the problem consists of fitting the absolute position and orientation of the camera to the measurement data [9]. A considerable number of approaches have been proposed for the past twenty years. Generally speaking, they can be divided according to different characteristics:

- By whether or not the solver is run iteratively or not, Iterative V.S. Non-iterative.
- By the require number of correspondence, P3P, P4P and more generally, PnP.

In my opinion, P3P and scalable PnP would be our focus since other methods are kind of intermediate products of the developing procedure. The reason why P3P is still favoured especially after the invention of efficient PnP algorithm is as follows:

- P3P is the minimal number of correspondences required to solve the general PnP problem. It should be noted that by adding some constraints embedded by the rigid body, this number can go down to 1, e.g. for non-holonomic vehicle.
- Although PnP algorithm can use the data redundancy to suppress the influence of noises, it can not deal with outliers. The most famous way to deal with outliers is to cascade the PnP algorithm with a RANSAC routine. Since for RANSAC, we prefer to use the minimal set to solve a given problem. P3P shows its superiority in this sense.

The first P3P algorithm in a geometric view is given in [9], later improvements have been addressed in [10], [11], [12] and [13].

Scalable PnP algorithm means the complexity of the PnP algorithm is $\mathcal{O}(n)$, which means its complexity or computational time grows linearly with the number of correspondence. This is very important since we would like to benefit from data redundancy without paying too much cost. The earlier PnP algorithms, like [14] ($\mathcal{O}(n^8)$) and [15] ($\mathcal{O}(n^4)$), are not traceable if we are going to work with thousands of correspondences. EPnP [16] [17] is the first PnP algorithm with $\mathcal{O}(n)$ complexity, followed by RPnP [18], DLS [19] [20], AsPnP [21], OPnP [22], UPnP [23].

Among iterative solutions, LHM [24] is the most famous solution. Other solutions also exist, like RPPnP [25] and the recently published [26]. Iterative solution usually show superior performance under noisy conditions. However, it is easy to see that their computational burden grows at least

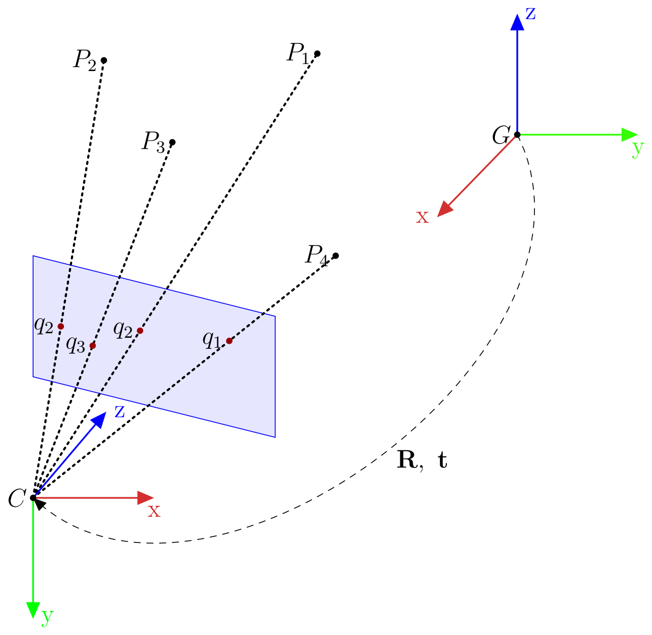


Figure 2.1: Principles of the Perspective n Point problem.

linearly with the number of correspondences. One of the biggest disadvantage of those iterative solutions is that the global optimality cannot be guaranteed since mathematically speaking, it is a nonlinear optimization problem which is nonconvex.

In this sense, [27] proposed a method based on Semi-Definite Programming (SDP) and Semi-Definite Relaxation (SDR). Unfortunately, after SDR, the global optimality cannot always be retained to the original problem. Moreover, solving a SDP is usually time-consuming or at least not in real-time.

With regarding to probabilistic optimality, several methods have been proposed to solve this problem in the sense of Maximum-Likelihood Estimation (MLE) [28], [29]. Currently, I have no plan to detail this kind of solutions. But the principle is to reformulate a cost function in order to maximize the likelihood and then solve this optimization problem iteratively. Also a reasonable estimation of the prior covariance should be built.

Planar case is one of the special case where multiple minimums can exist and hard to distinguish. Similar to [30], modern approaches may provide more than one solutions if ambiguity exists. This is one point we need to be aware of. The selection of the correct result can be done with the help of other sensors or motion dynamics.

It is also worthy to mention the famous POSIT algorithm which works for object-to-image registration problem [31] [32]. It is a iterative algorithm work under the assumption of weak orthogonal projection. An further improvement which allows POSIT works in unknown correspondence condition was published in [33].

All those aforementioned algorithms have no capability of dealing with outliers themselves. RANSAC is needed in order to filter out outliers. The conventional routine for outlier removal functions like:

- random select a set of correspondences.

- apply a algorithm and testify the consensus.
- iterate this until a certain number of iteration or inlier percentage is high enough.
- refinement with all inliers.

This conventional routine is sometimes time-consuming since solving PnP and verification (usually based on reprojection error) needs to be carried out in each iteration. So this is a still quite open area. Recent work by [34] proposes a very fast outlier removal technique. We will detail this in the following section.

Other related topics include: solving PnP problem with other intrinsics unknown, such as focus length and radial distortion [35] [36]; solving PnP problem with even less correspondence with the help of other constraints and sensors [37], [38], [39]; continuous solving the PnP problem with observers [40]; over-parameterization [41].

2.2 Theory and Implementation

There are two kind of objective functions for the PnP problem:

The first one is aiming to minimize the image error or the algebraic error:

$$f = \underset{\mathbf{R},\ \mathbf{t}}{argmin} \sum_{i=1}^{N} ||\mathbf{q}_i - \pi(\mathbf{R}\mathbf{P_i} + \mathbf{t})||^2 \Leftrightarrow$$
$$\underset{\mathbf{R},\ \mathbf{t}}{argmin} \sum_{i=1}^{N} \left((\frac{q_1}{q_3} - \frac{\mathbf{R}(1,:)\mathbf{P_i} + t_1}{\mathbf{R}(3,:)\mathbf{P_i} + t_3})^2 + (\frac{q_2}{q_3} - \frac{\mathbf{R}(2,:)\mathbf{P_i} + t_2}{\mathbf{R}(3,:)\mathbf{P_i} + t_3})^2 \right)$$

where π represents the projective function.

Another function is aiming to minimize the object-space error which is the alignment error of the back-tracing ray of the pixel and the ray from camera center to object points in camera coordinate frame:

$$f = \underset{\mathbf{R},\ \mathbf{t}}{argmin} \sum_{i=1}^{N} ||(\mathbf{I} - \mathbf{V}_i)(\mathbf{R}\mathbf{P_i} + \mathbf{t})||^2,\ with\ \mathbf{V}_i = \frac{\mathbf{q_i}\mathbf{q_i}^T}{\mathbf{q_i}^T\mathbf{q_i}}$$

Note: we assume here $\mathbf{q}_i$ has been normalized ($\mathbf{q}_i = \mathbf{K}^{-1}\mathbf{q}'_i$).

Generally speaking, the PnP problem is to find the answer for the aforementioned two optimization problem. What makes this problem challenging is the rotation matrix $\mathbf{R} \in \mathbb{SO}(3)$ which will implicitly impose nonlinear, nonconvex constraints. This is why the famous Direct-Linear-Transformation (DLT)[1] method cannot given us a promising result.

Here are some useful techniques which will help us solve this problem [42]:

- Branch-and-Bounding;
- Convex optimization, such as Second-Order-Cone-Programming (SOCP), SDP: this usually requires relaxation or even solve the prime problem by solving the dual problem;
- Algebraic Geometry: use Gröebner basis. This is feasible since the objective function and the constraints can be formulate as a polynomial system and the answer is the common root that makes those polynomials vanish [43].
- Manifold Optimization which applies optimization directly on $\mathbb{SO}(3)$ and $\mathbb{SE}(3)$.

[1]DLT means we omit those constraints and take each elements of $\mathbf{R}$ as separate variables. This can make the objective function as well as the constraints in affine or quadratic form.

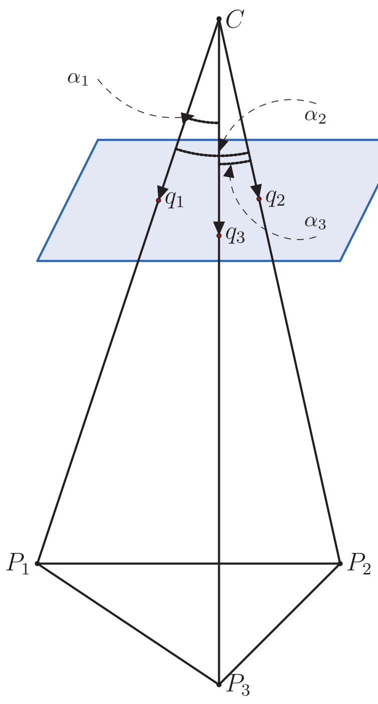


Figure 2.2: Principles of the Perspective 3 Point problem.

2.2.1 P3P

We start with the P3P algorithms. Generally speaking, the P3P problem solves this problem by firstly estimating the arc-length of each points in the camera frame, then reconstructs the point cloud in the camera frame by multiplying the arc-length with the back-tracing rays, and finally applies the 3D to 3D pose estimation routines in order to find the **R** and **t**. From Fig 2.3, by denoting $|CP_1| = d1$, $|CP_2| = d2$, $|CP_3| = d3$, $|P_1P_2| = d12$, $|P_1P_3| = d13$, $|P_2P_3| = d23$, , we will have the following relationship using the cosine rule:

$$\begin{aligned} d12^2 &= d1^2 + d2^2 - 2cos(\alpha_2)d1d2 \\ d13^2 &= d1^2 + d3^2 - 2cos(\alpha_1)d1d3 \\ d23^2 &= d2^2 + d3^2 - 2cos(\alpha_3)d2d3 \end{aligned}$$

where $\alpha_1 = acos(\mathbf{q}_1 \cdot \mathbf{q}_3)$, $\alpha_2 = acos(\mathbf{q}_1 \cdot \mathbf{q}_2)$, $\alpha_3 = acos(\mathbf{q}_2 \cdot \mathbf{q}_3)$. By some algebraic operations[2], we will have a 4th order polynomial equation:

$$A_4t^4 + A_3t^3 + A_2t^2 + A_1t^1 + A_0 = 0$$

[2]eliminate one variable by the other two and then continue

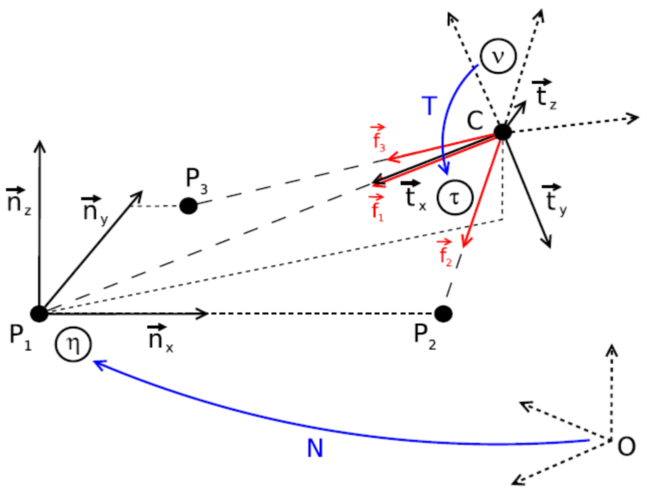


Figure 2.3: Figure from [13].

By solving this equation, we will have the arc-length. Continue this for other arcs, we can find them all. By multiplying $\mathbf{q}_i$ with d_i, we will have $\mathbf{Q}_i$, then the problem transforms to:

$$f = \underset{\mathbf{R},\ \mathbf{t}}{argmin} \sum_{i=1}^{N} ||\mathbf{Q}_i - \mathbf{R}\mathbf{P_i} + \mathbf{t}||^2$$

which is a typical 3D to 3D pose estimation problem that can be solved.

P3P methods [9], [11], and [12] all function in this way. [9] is more like a hard-code solution. [11] gives detailed illustration of the P3P problem and also their solution under some degenerate cases. However, [11] is too mathematical which is hard to read. But if you only care about the solution, then you can go directly to the normal case. [12] is the most clear paper that clearly explain this problem.

The only different P3P method was proposed in [13]. By introducing two intermediate frames, it changes the original 3D to 2D problem to a plane problem. After geometric operations, it finally results in a 4^{th} order polynomial equation. However, this 4^{th} order polynomial does not take arc-length as the variable, but an alternative parameter which represent the tilt angle of the $P_1 - P_2 - C$ plane. So the final $\mathbf{R}$, $\mathbf{t}$ have to be found in another way rather than solving the 3D to 3D pose estimation problem. This method is easy to understand and very fast.

Summary of P3P

Here are several points which remains unclear:

- which P3P gives the best performance in terms of stability, runtime?
- how can P3P benefit from data redundancy? Or P3P can only be used for outlier removal together with RANSAC?

2.2.2 Complex PnP

Here, Complex PnP means not-well scalable PnP algorithms such as [15] and [14]. I will begin with [15] and then talk about [14].

The linear PnP in [15] is based on the cosine rule function we have seen in P3P. [15] uses the Sylvester resultant to eliminate variable, which will result in a 8^{th} order polynomial which only contains even terms (equals to a 4^{th} order polynomial). The idea is quite straightforward: since each tuple-3, e.g $1, 2, 3 or 1, 4, 5$, will generate a 4^{th} order polynomial in, e.g. d_1. So, N points generates C_N^2 cosine equations which furthermore generates $\frac{(n-1)(n-2)}{2}$ 4^{th} order polynomials (eliminating redundant polynomials). By taking the linearization which means taking $\mathbf{t} = (1, x, x^2, x^3, x^4)^T$ as separate terms, we will have a equation like $\mathbf{At} = 0$.

$$\underbrace{\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_K \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \end{bmatrix} = \mathbf{0}$$

Then $\mathbf{t}$ can be found by taking the right singular vector. Supposing the dimension of the null space is over one, which means

$$x = \lambda \mathbf{v}_4 + \beta \mathbf{v}_5, \lambda, \beta \in \mathbb{R}.$$

In order to solve for λ, β, we should use the so-called re-linearization technique: Notice that $t_i t_j = t_k t_l,\ if\ i + j = k + l$, so we call formulate another normal system with $(\lambda^2, \lambda\beta, \beta^2$. We solve again for λ, β and then for d_i. The final pose estimation problem, once again, turns out to be a 3D to 3D pose estimation problem. As can be seen from here, data redundancy is utilized by stacking the 4^{th} order polynomial equation and find the null vector. However, since the aforementioned procedure has to be run for each possible arc, for sure, the complexity does not scale very well.

Another linear PnP algorithm is proposed in [14]. It also starts with the cosine equations. However, they applied linearization as follows:

recall: $x_1^2 + x^2 - 2x_1x_2cos(\alpha_{12}) - d_{12}^2 = 0$

linearization: $\mathbf{t} = x_1^2, x_1x_2, x_2^2$

do this for $N \Leftrightarrow$

$$\underbrace{\left[\begin{array}{cccc|ccccccc} -2cos(\alpha_{12}) & 0 & \cdots & 0 & 1 & 1 & 0 & \cdots & 0 & 0 & -d_{12}^2 \\ 0 & -2cos(\alpha_{13}) & \cdots & 0 & 1 & 0 & 1 & \cdots & 0 & 0 & -d_{13}^2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & vdots & vdots \\ 0 & 0 & \cdots & -2cos(\alpha_{n,n-1}) & 0 & 0 & \cdots & cdots & 1 & 1 & -d_{n,n-1}^2 \end{array}\right]}_{\mathbf{M}}$$

$\mathbf{Mt} = \mathbf{0}$

Finding $\mathbf{t}$ by choosing the rightmost singular vector and apply a similar relinearization to recover x. For N points, we have C_N^2 cosine equations. Then we have N terms in form of x_i^2, C_N^2 terms in form of x_ix_j and one slack variable 1. So in total, the solution vector is $\frac{N(N+1)}{2} + 1$. The kernel is of dimension $\frac{N(N-1)}{2}$, so we will have a null space of dimension $N + 1$. We still needs $\frac{N*N*(N-1)}{2}$ constraints to do the relinearization. As can be seen, this method is extremely complex. The computation time will increase significantly with the number of correspondences. This method is only theoretically meaningful, but cannot be used in practice, especially now we have the $\mathcal{O}(n)$ solutions.

Summary of Complex PnP

The linearization and relinearization techniques opens the gate for later $\mathcal{O}(n)$ PnP solutions. Regard to the Complex PnP, we could include them as the related work, but use other $\mathcal{O}(n)$ solutions in practice.

2.2.3 POSIT & SoftPOSIT

Original POSIT

POSIT was firstly proposed in [31] and later refined in [32] for solving planar case. The original formulation of POSIT is slightly different with the formulation given below. However, the two formulation can be easily transformed by doing some variable replacements. Here I just list the key equations for POSIT algorithm.

The principle of POSIT algorithm is that the perspective projection $\mathbf{x}$ can be corrected by some scales ϵ_i to match with the scaled orthographic projection $s\mathbf{P}_i$. So we can formulate two normal equations for $\mathbf{x} = [u,\ v]^T$ such as

$$\begin{aligned}
\mathbf{R} &= \begin{bmatrix} \mathbf{i}^T \\ \mathbf{j}^T \\ \mathbf{k}^T \end{bmatrix} \\
s\mathbf{i} &= \mathbf{I},\ s\mathbf{j} = \mathbf{J},\ s\mathbf{k} = \mathbf{K}, \\
\mathbf{P}_i \cdot \mathbf{I} &= u_i(1+\epsilon_i) - u_0 \\
\mathbf{P}_i \cdot \mathbf{J} &= v_i(1+\epsilon_i) - v_0 \\
&\Leftrightarrow \\
\mathbf{A}\mathbf{x} &= \mathbf{b}
\end{aligned}$$

By solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ using the pseudo-inverse, we can recover the pose and then update the $\epsilon_i = \frac{\overrightarrow{\mathbf{P}_0\mathbf{P}_i}\cdot\mathbf{k}}{t_z}$. This process iterates until convergence. For planar case, the rank of $\mathbf{A}$ is 2. Consequently, any solution x_0 plus the scaled null vector $\mathbf{x}_{null}$ of $\mathbf{A}$ could be the solution $\mathbf{x}_0 + \lambda\mathbf{x}_{null}$. We note the solution for $u,\ v$ as $\mathbf{I}_0 + \lambda\mathbf{t}_{null}$ and $\mathbf{J}_0 + \mu\mathbf{t}_{null}$ (the null vector is the same since $\mathbf{A}$ works both for u and v). Then since we know that the correct $\mathbf{I}$ and $\mathbf{J}$ are orthogonal to each other and of the same length:

$$\begin{aligned}
\lambda\mu &= \mathbf{I}_0\mathbf{J}_0 \\
\lambda^2 - \mu^2 &= ||\mathbf{J}_0||^2 - ||\mathbf{I}_0||^2
\end{aligned}$$

which will results in a 2^{rd} polynomial and brings us two solutions. As a result, two possible poses could be returned. As you can see from here, the number of solutions may grow exponentially ($1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow \cdots$), in order to suppress this curse of dimension, [32] suggests if over two solutions are generated, only keeps the best two possible solutions. In my experiments, the scheme works well although there is no theoretical proof.

SoftPOSIT

SoftPOSIT is an improvement for orginal POSIT algorithm which aims to estimate pose and correspondence simultaneuously. As can be seen from this statement, for SoftPOSIT solution, the correspondence could be unknown or partially known. This serves the biggest difference of SoftPOSIT algorithm relative to other solutions mentioned in this report.

POSIT routine Assume object points are $\mathbf{P}_i,\ i = 1,\ \cdots,\ M$ (homogeneous coordiantes) and image points $\mathbf{q}_j,\ j = 1,\ \cdots,\ N$ (normalized homogeneous coodrinates, so $f = 1$), then the coordinates of object points relative to an origin $\mathbf{P}_0$ can be represented as $\tilde{\mathbf{P}}_i = \mathbf{P}_i - \mathbf{P}_0$. From the perspective projection model, we have:

$$w\begin{bmatrix}x\\y\\1\end{bmatrix} = \begin{bmatrix}\mathbf{R}^T_{(1,:)} & tx\\ \mathbf{R}^T_{(2,:)} & ty\\ \mathbf{R}^T_{(3,:)} & tz\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{P}}\\1\end{bmatrix}$$

$$w\begin{bmatrix}x\\y\end{bmatrix} = s\begin{bmatrix}\mathbf{R}^T_{(1,:)} & tx\\ \mathbf{R}^T_{(2,:)} & ty\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{P}}\\1\end{bmatrix}$$

$$with\ s = \frac{1}{t_z},\ w = \frac{\mathbf{R}^T_{(3,:)}\tilde{\mathbf{P}}_i}{tz} + 1$$

If $t_z >> \mathbf{R}^T_3\tilde{\mathbf{P}}_i$, then $w \approx 1$. Furthermore, we will have

$$\begin{bmatrix}x\\y\end{bmatrix} = s\begin{bmatrix}\mathbf{R}^T_{(1,:)} & tx\\ \mathbf{R}^T_{(2,:)} & ty\end{bmatrix}\begin{bmatrix}\tilde{\mathbf{P}}\\1\end{bmatrix}$$

which defines the scaled orthographic projection. Taking the transpose of this equation, we will have

$$[X,\ Y,\ Z,\ 1]\underbrace{\begin{bmatrix}s\mathbf{R}_{(1,:)} & s\mathbf{R}_{(2,:)}\\ stx & sty\end{bmatrix}}_{\mathbf{Q}} = [wx,\ wy]$$

$\mathbf{Q}$ is the variable we are going to optimize. Once we have $\mathbf{Q}$, we can extract the corresponding blocks as $s\mathbf{R}_{(1,:)}$ and $s\mathbf{R}_{(2,:)}$, since we know that $||\mathbf{R}_1|| = 1$, s can be recorvered as $s = \sqrt{|s\mathbf{R}_{(1,:)}||s\mathbf{R}_{(2,:)}|}$. Then $\mathbf{R}_{(3,:)} = \mathbf{R}_{(1,:)} \times \mathbf{R}_{(2,:)}$, $t_x = \frac{st_x}{s}$, $t_y = \frac{st_y}{s}, t_z = \frac{1}{s}$. This process runs iteratively by firstly letting $w_k = 1,\ k = 1,\ 2,\ \cdots,\ M$ and update w_k with newly estimated $\mathbf{R}, \mathbf{t}$. The geometric interpretation of this equation is that w_k functions like a correction of the perspective projection $[x,\ y,\ 1]^T$ to the scaled orthographic projection. The overall objective function for POSIT can be written as:

$$E = \sum_{k=1}^{M}\left(\left(\mathbf{Q}_1^T\mathbf{P}_k - w_kx_k\right)^2 + \left(\mathbf{Q}_2^T\mathbf{P}_k - w_ky_k\right)^2\right)$$

where $\mathbf{Q}_1 = s[\mathbf{R_{(1,:)}}^T\mathbf{tx}]^T$, $\mathbf{Q}_2 = s[\mathbf{R_{(2,:)}}^T\mathbf{ty}]^T$, $\mathbf{P}_k = [\tilde{\mathbf{P}}_k^T, 1]^T$. By taking the derivative of E with respect to $\mathbf{Q}_1, \mathbf{Q}_2$, we can get the least-square solution for $\mathbf{Q}_1, \mathbf{Q}_2$ as

$$\mathbf{Q}_1 = \left(\sum_{k=1}\mathbf{P}_k\mathbf{P}_k^T\right)^{-1}\left(\sum_{k=1}w_kx_k\mathbf{P}_k\right)$$

$$\mathbf{Q}_2 = \left(\sum_{k=1}\mathbf{P}_k\mathbf{P}_k^T\right)^{-1}\left(\sum_{k=1}w_ky_k\mathbf{P}_k\right)$$

Correspondence Update routine SoftPOSIT [33] algorithm maintains a soft assign matrix $\mathbf{M} \in \mathbb{R}^{M\times N}$, (M is the number of object points, and N is the number of the image points) in which the element m_{ij} represents the likelihood of model point i matches with image point j. Since we have no information of the correspondence, so for each w_k, its correction has to be applied for each image points. Then we compute the pixel distance of each corrected image coordinates with the scaled orthographic projection of the k^{th} object point:

$$d_{kj}^2 = (\mathbf{Q}_1\mathbf{P}_k - w_k x_j)^2 + (\mathbf{Q}_2\mathbf{P}_k - w_k y_j)^2$$

An threshold α is used as the maximum tolerate distance for considering an correspondence. The modified optimization problem is described as:

$$E = \sum_{k=1}^{M}\sum_{j=1}^{N} m_{kj}\left((\mathbf{Q}_1\mathbf{P}_k - w_k x_j)^2 + (\mathbf{Q}_2\mathbf{P}_k - w_k y_j)^2\right)$$

where m_{kj} serves as a weight $(0 \leq m_{kj} \leq 1)$. This is a weighted least square problem which can be similarly solved.

Once we solve for the new pose, we do the update of the soft assign matrix $\mathbf{M}$ by:

- compute the matching distance d_{kj}^2.
- update $w_k = \gamma exp(-\beta(d_{kj}^2 - \alpha))$ and truncate this to be within $0 - 1$.
- the row and column of $\mathbf{M}$ should be 1 since one object point only matches to one image point. So re-normalization is necessary: this is done by alternatively normalize along the row direction and then along the column direction using the so called Sinkhorn algorithm.

Summary of POSIT & SoftPOSIT

SoftPOSIT [33] iterates to solve this problem, which is related to the general expectation-maximization (EM) algorithm. So generally speaking, global optimality is not guaranteed. Secondly, SoftPOSIT algorithm is usually very slow since its working condition is more challenging. From the experiments I have done, SoftPOSIT is not very stable. Sometimes it cannot find the correct solution.

2.2.4 Iterative PnP

We will introduce two iterative PnP solutions: the LHM [24], the PPnP [25].

LHM

The most famous iterative solution should be [24]. The biggest difference of this method compared with other methods would be classical methods apply optimization using Euler angles which constrain their performance and optimality, while this method does the so-called orthogonal iteration.

The objective function used in this method is the object-space error:

$$E = \sum_{i=1}^{N} ||(\mathbf{I} - \mathbf{V}_i)(\mathbf{R}\mathbf{P}_i + \mathbf{t})||^2$$

It is clear that if we make $\mathbf{R}$ fixed, then the objective function is in the quadratic form of $\mathbf{t}$, which means the optimal $\mathbf{t}$ can be computed as

$$\mathbf{t}(\mathbf{R}) = \frac{1}{N}(\mathbf{I} - \frac{1}{N}\sum_j \hat{\mathbf{V}}_j)^{-1}\sum_j(\hat{\mathbf{V}}_j - \mathbf{I})\mathbf{R}\mathbf{p}_j$$

With the equation, the iteration process starts with an initial estimation of the $\mathbf{R}_k$:

- compute for the $\mathbf{t}_k$.
- compute the transformed point coordiante $\mathbf{q}_i^k = \mathbf{R}_k\mathbf{p}_i^k + \mathbf{t}$.
- update $\mathbf{R}_k$ as the minimizer for $\sum_{i=1}^N ||\mathbf{R}_k\mathbf{p}_i + \mathbf{t} - \hat{\mathbf{V}}_i\mathbf{q}_i^k||^2$, which is a 3D to 3D pose estimation problem if we assume $\mathbf{q_i^k}$ is fixed.
- iterates this process until a local minimum achieves.

This method is proved to be globally convergent, which means given an initial guess, it will definitely converge to a local minimum. However, globally convergency does not mean globally optimal. As can be seen that this method is a nonlinear optimization method, which normally would require a initial value. Any PnP solution can be a front-end to provide the initial value. [24] also propose an initial method under weak perspective approximation. Weak perspective projection means the depths of all camera space coordinates are roughly equal to the principle depth and the object is close to the optical axis of the camera:

$$u_i = \frac{1}{s}(\mathbf{R}_1^T\mathbf{p}_i + t_x)$$
$$v_i = \frac{1}{s}(\mathbf{R}_2^T\mathbf{p}_i + t_y)$$

Recall the derivation of the POSIT [32], it is exactly the same approximation used in POSIT. The initial value can be obtained by simply using the image coordinates as the $\mathbf{q}_i$ for estimating the initial $\mathbf{R}$ and $\mathbf{t}$. [24] shows that this is approximately the optimization procedure with s, $\mathbf{R}$, $\mathbf{t}$ simultaneously.

PPnP

Although look quite differently, the PPnP algorithm has a strong connection with the POSIT algorithm and the LHM algorithm except its formulation is not in 2D image space but in object space. From the perspective projection:

$$\zeta_i\mathbf{q}_i = \mathbf{R}\mathbf{P}_i + \mathbf{t}, \text{ for all } i$$

$$\underbrace{\begin{bmatrix}\zeta_1 & 0 & \cdots & 0\\ 0 & \zeta_2 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & \zeta_n\end{bmatrix}}_{\mathbf{Z}}\underbrace{\begin{bmatrix}\mathbf{q}_1^T\\ \mathbf{q}_2^T\\ \vdots\\ \mathbf{q}_n^T\end{bmatrix}}_{\mathbf{Q}}\mathbf{R} + \underbrace{\begin{bmatrix}\mathbf{c}^T\\ \mathbf{c}^T\\ \vdots\\ \mathbf{c}^T\end{bmatrix}}_{\mathbf{1c}^T} = \underbrace{\begin{bmatrix}\mathbf{P}_1^T\\ \mathbf{P}_2^T\\ \vdots\\ \mathbf{P}_n^T\end{bmatrix}}_{\mathbf{S}}, \ \mathbf{c} = -\mathbf{R}^T\mathbf{t}$$

As can be easily derived, $\zeta_i\mathbf{q}_i\mathbf{R}$ is the point in world frame (with origin shifting to the camera center). By adding the coordinates of the camera center $\mathbf{c}$, it will theoretically align with the object point. So the objective function is to minimize the error in object space

$$E = \sum_i ||\mathbf{S} - \mathbf{ZQR} - \mathbf{1c}^T||_F^2$$

The PPnP algorithm does the optimization as

- assume $\mathbf{Z}$ fixed, then it is a 3D to 3D problem where $\mathbf{R}$ can be recovered using SVD. The paper [25] writes this implicitly, so it is not so clear. Let's do it explicitly. Minimizing E is equivalent to minimize $||\mathbf{S}^T - \mathbf{R}^T\mathbf{Q}^T\mathbf{Z} + \mathbf{R}^T\mathbf{t}\mathbf{1}||_F^2$, which can be further extend to

$||\mathbf{R}\mathbf{S}^T - \mathbf{Q}^T\mathbf{Z} + \mathbf{t}\mathbf{1}||_F^2$. The Frobenius norm of $\mathbf{A}$ is the trace of the $\mathbf{A}^T\mathbf{A}$. So this is exactly the 3D to 3D problem. We can eliminate $\mathbf{t}$ by taking the mean as $\bar{\mathbf{S}} = \mathbf{S}(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{n})$. Similar to the 3D to 3D problem, $\mathbf{R}$ can be found by applying SVD to $\mathbf{Q}^T\mathbf{Z}(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{n})(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{n})^T\mathbf{S}$. Since $(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{n})(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{n})^T = (\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{n})$, this is simplified to $\mathbf{Q}^T\mathbf{Z}(\mathbf{I} - \frac{\mathbf{1}\mathbf{1}^T}{n})\mathbf{S}$, which is exactly the equation given in [25].

- following the previous 3D to 3D problem, $\mathbf{t} = \mathbf{Q}^T\mathbf{Z}\mathbf{1}/n - \mathbf{R}\mathbf{S}^T\mathbf{1}/n$, so $\mathbf{c} = -\mathbf{R}^T\mathbf{t} = \frac{(\mathbf{S} - \mathbf{Z}\mathbf{Q}\mathbf{R}^T)\mathbf{1}}{n}$, which also coincides with the equation listed in [25].

- update $\mathbf{Z}$ by $\mathbf{R}(\mathbf{S}^T - \mathbf{c}\mathbf{1}^T) = \mathbf{P}^T\mathbf{Z} \rightarrow diag(\mathbf{P}\mathbf{R}(\mathbf{S}^T - \mathbf{c}\mathbf{1}^T)) = diag(\mathbf{P}\mathbf{P}^T\mathbf{Z})$, note $\mathbf{Z}$ is a diagonal matrix, so $\mathbf{Z} = diag(\mathbf{P}\mathbf{R}(\mathbf{S}^T - \mathbf{c}\mathbf{1}^T))diag(\mathbf{P}\mathbf{P}^T)^{-1}$.

- iterate until converge.

As can be seen from here, the PPnP functions similarly to POSIT under the scaled orthographic projection and LHM under weak perspective projection. Perhaps this is why all equations in this paper are written in such a implicit way.

Summary of iterative PnP algorithm

- LHM has been documented by many references to be the state-of-the-art iterative method. So this should be considered as the nonlinear optimization method to refine the result.

- The benchmark of LHM and POSIT is unclear.

- Manifold optimization can do the same, it remains a question which one is better.

2.3 Convex PnP

Convex PnP means solving the PnP problem with convex optimization routines like SOCP and SDP [27]. In my opinion, even if the method is named globally optimal, it cannot guarantee to find the optimum solution since it use some relaxations. Secondly, it is implicitly iterative method since SOCP and SDP are usually solving in an iterative way. Thirdly, solving a SOCP and SDP problem is time consuming even with advanced toolbox like Mosek. So generally speaking, they are not practical methods at least in current stage. However, it seems to be an trend that increasing number of papers have been proposed in this area. Recent works focus on solving this problem by solving the Langrange dual problem. The strong duality may guarantee the optimality under some mild conditions.

2.4 $\mathcal{O}(n)$ PnP

2.4.1 Algebraic Geometry PnP

Algebraic Geometry PnP means solving PnP problem by using Algebraic Geometry [22] [19] [23] [21] [44]. This kind of methods involves strong mathematical knowledge on Algebraic Geometry. Consequently, if you are not familiar with Algebraic Geometry, then it is extremely hard to understand. Since I am also on the way to understand this technique, I can only list several key points for this kind of algorithms.

Algebraic Geometry PnP algorithms all works in the following routines:

- formulate the objective function, constraints.
- carefully choose a parameterization which can write the objective function, constraints in the form of multi-variable polynomial (one key step);
- use the Langrange multiplier and take the first order optimality condition to construct a polynomial system (a batch of polynomials), e.g.

$$\begin{bmatrix} f_1(x_1, x_2, \cdots, x_n) \\ f_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ f_n(x_1, x_2, \cdots, x_n) \end{bmatrix} = \mathbf{0}$$

  So the problem transforms to find the common roots of this polynomial system.
- find the Greöbner basis. Then use the elimination and extension to find all solutions (could be complex solutions). This step is another key step. It can be automatically found by using the generator in [43]. However, special symmetric characteristic can significantly boosting the generated solver, which needs sufficient knowledge about Algebraic Geometry.
- recover the solutions and then recover $\mathbf{R}, \mathbf{t}$, compute the error defined and select the one with the minimum cost.

Summary

Algebraic Geometry PnP seems to be very efficient, especially the UPnP [23] and OPnP [22]. However, one thing is quite strange. If you carefully review those papers, you will find different methods performs quite differently in different experiments. That is to say: A is the second best in paper one, but the worst in paper two.

Two questions remains:

- So it will be interesting to see which one is actually the best one.
- Can RANSAC be integrated in another way?

2.4.2 RPnP

RPnP [18] is one of the first two $\mathcal{O}(n)$ PnP algorithm. It should be noted that the real implementation of the RPnP algorithm is slightly different with what is written in the paper.

Coordinate Transformation The first step is to transform the coordinates. What it does actually is to select the pair with the largest intersection angle from n random samples.

$$(i, j) = \underset{(i,j) \in n \ paris}{argmin} \ \mathbf{v}_i \cdot \mathbf{v}_j$$

Then choose the $\overrightarrow{\mathbf{P}_i\mathbf{P}_j}$ as the rotated x-axis. The corresponding rotation matrix can be found as

$\mathbf{x} = \overrightarrow{\mathbf{P}_i\mathbf{P}_j}$

Method 1 :

$\alpha = \mathbf{x} \cdot [1,0,0]^T, \mathbf{v} = [1,0,0]^T \times \mathbf{x},\ \mathbf{v} = \dfrac{\mathbf{v}}{||\mathbf{v}||},\ \mathbf{R} = cos(\alpha)\mathbf{I}_3 + (1 - cos(\alpha)) * \mathbf{v} * \mathbf{v}' + sin(\alpha) * \mathbf{v}_\times$

Method 2 :

if $|\mathbf{x} \cdot [0,1,0]^T| < \mathbf{x} \cdot [0,0,1]^T|$ *then* $\mathbf{z} = \dfrac{\mathbf{x} \times [0,1,0]}{||\mathbf{x} \times [0,1,0]||},\ \mathbf{y} = \dfrac{\mathbf{z} \times \mathbf{x}}{||\mathbf{z} \times \mathbf{x}||}$

else, $\mathbf{y} = \dfrac{[0,0,1] \times \mathbf{x}}{||[0,0,1] \times \mathbf{x}||},\ \mathbf{z} = \dfrac{\mathbf{x} \times \mathbf{y}}{||\mathbf{x} \times \mathbf{y}||}$

$\mathbf{R} = [\mathbf{x},\ \mathbf{y},\ \mathbf{z}]$

Transform $\mathbf{P}_i$ as $\mathbf{P}'_i = \mathbf{R}\mathbf{P}_i + \mathbf{t}_0,\ \mathbf{t}_0 = -\frac{\overrightarrow{\mathbf{P}_i\mathbf{P}_j}}{2}$. This is like a coordinate normalization.

3 Point constrain Recall the 3 point constrain in the P3P section using the cosine rule, here we can formulate $n-2$ 3 Point constrains. In stead of solving using linearization technique, the author propose to formulate a minimization problem:

$$\begin{aligned}&\text{assuming:} f_1, f_2, \cdots, f_{n-2} \text{ are the corresponding polynomials}\\ &f' = \sum_i^{n-2} f_i^2\end{aligned}$$

f' is a 8^{th} order polynomial, the optimal solution for f' should be the roots of its derivative. The solution can now be used to compute the $\overrightarrow{\mathbf{P}_i\mathbf{P}_j}$ in camera frame since we know the arc lengths as well as the ray direction.

Recover Rotation & Translation RPnP decomposes the $\mathbf{R}$ into two parts:

$$\mathbf{R} = \mathbf{R}' rot(x, \alpha)$$

where $\mathbf{R}'$ is constructed with the $\overrightarrow{\mathbf{P}_i\mathbf{P}_j}$ in camera frame using the aforementioned rotation matrix composition approach. $rot(x, \alpha)$ is a pure rotation along x-axis[3]. Together with translation vector, a linear function can be built as

$$\lambda_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{R}' \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \mathbf{P}_i + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Note here $cos(\alpha)$ and $sin(\alpha)$ are treated separately by ignoring the constraint of $cos(\alpha)^2 + sin(\alpha)^2 = 1$, this is actually a linearization. Again, we can formulate $\mathbf{Ax} = \mathbf{0}$ and solve.

Although theoretically we can directly recover the pose, the author just use the solution for 3D points reconstruction and then solve the 3D to 3D problem for pose recover. Perhaps the linearization may deteriorate the result. So they don't directly recover the pose.

[3]In their paper, they use $rot(z, \alpha)$.

Summary

According to the paper, RPnP outperforms others. However, in my experiments, it does show some instabilities.

- It may be a good idea to evaluate RPnP algorithm in a fair benchmark experiment.
- RPnP algorithm may perform badly in case of outliers. For example, recall its first step by selecting two points with maximum expansion angle, what if two outliers are selected? This may ruin the following steps. Where to add the RANSAC is worthy of investigating.

2.4.3 EPnP

EPnP [16] [17] is the first $\mathcal{O}(n)$ PnP algorithm. It was later enhanced in [34] for outlier removal and in [28] for maximum likelihood estimation.

Original EPnP

EPnP follows the same routine as other methods by firstly reconstruct the 3D points in the camera coordinate frame and do 3D to 3D pose estimation.

Parameterization using 4 control points Parameterization is the key technique which makes EPnP works with a $\mathcal{O}(n)$ complexity. It uses four control points $\mathbf{c}_j$ as the base to represent other points:

$$\mathbf{p}_i = \sum_{j=1}^{4} \alpha_{ij}\mathbf{c}_j,\ \sum_{j=1}^{4} \alpha_{ij} = 1,\ i = 1, 2, \cdots, n$$

As a result, in order to reconstruct n points' coordinates, we only need to find the coordinates of four control points $\mathbf{c}_j$. Rewriting the perspective projection with the help of $\mathbf{c}_j$, we will have:

$$\forall\ i,\ w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j \\ y_j \\ z_j \end{bmatrix} \Leftrightarrow$$

$$\sum_{j=1}^{4} \alpha_{ij} f_u x_j + \alpha_{ij}(u_c - u_i) z_j = 0$$

$$\sum_{j=1}^{4} \alpha_{ij} f_v y_j + \alpha_{ij}(v_c - v_i) z_j = 0 \Leftrightarrow$$

$$\mathbf{M}\mathbf{x} = 0$$

Now $\mathbf{x} = [\mathbf{c}_1^T, \mathbf{c}_2^T, \mathbf{c}_3^T, \mathbf{c}_4^T]^T$. The solution belongs to the null space of $\mathbf{M}$.

$$\mathbf{x} = \sum_{i=1}^{N} \beta_i \mathbf{v}_i$$

Because we have 12 unknowns and the dimension of $dim(\mathbf{M}) = 2n \times 12$, if $2n \leq 12$, then the solution is the linear combination of those null vectors. That's why we have β_i here.

Choosing the right combination EPnP will always find the β_i up to $N = 3$ ($N = 4$ will be checked if a certain condition meets).

- $N = 1$, the inter-distance is used to find the correct β:

$$||\beta \mathbf{v}^{[i]} - \beta \mathbf{v}^{[j]}||^2 = ||\mathbf{c}_i - \mathbf{c}_j||^2$$

  here $\mathbf{c}_i,\ \mathbf{c}_j$ are their values in the world coordinate frame which can be computed from $\mathbf{P}$. This can give us a least square problem such as $\mathbf{A}\beta = \mathbf{b}$.

- $N = 2$, the inter-distance is used to find the correct β:

$$||\beta_1 \mathbf{v}^{[i]} + \beta_2 \mathbf{v}^{[i]} - (\beta_1 \mathbf{v}^{[j]} + \beta_2 \mathbf{v}^{[j]})||^2 = ||\mathbf{c}_i - \mathbf{c}_j||^2$$

  Since $C_4^2 = 6$, this can give us a overdetermined least square problem such as $\mathbf{A}\boldsymbol{\beta} = \mathbf{b}$ where $\boldsymbol{\beta} = [\beta_1^2, \beta_1\beta_2, \beta_2^2]^T$.

- $N = 3$, use the same linearization technique with $\boldsymbol{\beta} = [\beta_1^2, \beta_1\beta_2, \beta_1\beta_3, \beta_2^2, \beta_2\beta_3, \beta_3^2]^T$.

- $N = 4$, the system is under-determined this time. We still form $\mathbf{A}\boldsymbol{\beta} = \mathbf{b}$ and search for the null vector $\boldsymbol{\gamma}$ that meet $\boldsymbol{\beta} = \sum_{i=1}^4 \lambda_i \boldsymbol{\gamma}_i$.
  Since $\boldsymbol{\beta} = [\beta_1^2, \beta_1\beta_2, \beta_1\beta_3, \beta_1\beta_4, \beta_2^2, \beta_2\beta_3, \beta_2\beta_4, \beta_3^2, \beta_3\beta_4, \beta_4^2]^T$, we need to use the relinearization technique to find λ_i. This is feasible since $\beta_{ab}\beta_{cd} = \beta_a\beta_c\beta_b\beta_d = \beta_{a'b'}\beta_{c'd'}$ where a', b', c', d' represents any permutation of the integers a, b, c, d.

Planar case Planar case is tackled as a special case since only 3 control points are needed. Then $\mathbf{M}$ is a $2n \times 9$ matrix.

Gauss-Newton Optimization Applying Gauss-Newton Optimization for EPnP is easy since there is no nonlinear constraints (unlike Gauss-Newton Optimization for $\mathbf{R}$ where $\mathbf{R} \in \mathbb{SO}(3)$ imposes several nonlinear constraints). The optimized variable is $\boldsymbol{\beta} = [\beta_1, \beta_2, \beta_3, \beta_4]^T$ and the objective function is as

$$E(\boldsymbol{\beta}) = \sum \left(||\mathbf{c}_i^c - \mathbf{c}_j^c||^2 - ||\mathbf{c}_i^w - \mathbf{c}_j^w||^2\right)$$

which corresponds to minimizing the error of inter-distances.

2.4.4 EPPnP

EPPnP is nothing more than reformulate EPnP in a more efficient way. By using the Kronecker product, the formulation can be written as

$$[\alpha_{i1},\ ,\alpha_{i2},\ \alpha_{i3},\ \alpha_{i4}] \otimes \begin{bmatrix} 1 & 0 & -u_i \\ 0 & 1 & -v_i \end{bmatrix} \mathbf{x} = \mathbf{0}$$

u_i, v_i are normalized coordinates. This will significantly speed-ups when building $\mathbf{M}$.

2.4.5 REPPnP

REPPnP is the variant of of EPnP which can deal with outliers. In noise-free case, the rank of the null space of $\mathbf{M}$ is one. Due to the noise, the rank may grow up to four. Although EPnP proceed each of the cases and retain the one with minimum reprojection error, it is still very sensitive to the presence of outliers. REPPnP works with the assumption that the rank of the null space of

$\mathbf{M}$ is one and actually use this as a criteria for outliers removal. Theoretically, REPPnP attemps to recover an approximation of $\mathbf{M}$ by the following optimization problem:

$$\underset{\mathbf{L},\ \mathbf{W}}{argmin} ||\mathbf{W}(\mathbf{M}-\mathbf{L})||^2$$
$$\text{subject to: } rank(\mathbf{L}) = rank(\mathbf{M}) - 1$$

where $\mathbf{W} = diag(w_1, w_1, w_2, w_2, \cdots, w_n, w_n) \in \mathbb{R}^{2n\times 2n}$ is indicator matrix with binary entries indicating each correspondence is considered as inlier ($w_i = 1$) or outlier ($w_i = 0$). Rewrite this with the null vector $\mathbf{x}$, we will have:

$$\underset{\mathbf{x},\ \mathbf{W}}{argmin} ||\mathbf{W}(\mathbf{M}-\mathbf{L})x||^2$$
$$\Leftrightarrow \underset{\mathbf{x},\ \mathbf{W}}{argmin} ||\mathbf{W}\mathbf{M}\mathbf{x}||^2$$

which is a unconstrained optimization problem. This is solved iteratively as follows:

- initialize $\mathbf{W}$ as $\mathbf{I}_{2n}$ and ξ as Inf.
- svd($\mathbf{M}^\mathbf{T}\mathbf{W}\mathbf{M}$) and make $\mathbf{x}$ the smallest right singular vector.
- compute error $\epsilon = \mathbf{M}\mathbf{x}$ and $\epsilon_i = ||(\epsilon_{2i-1}, \epsilon_{2i})||$.
- find $\epsilon_{25\%}$
- if $\epsilon_{max} = \epsilon_{25\%} > \xi$ then return $\mathbf{W}, \mathbf{x}$.
- else $\xi = \epsilon_{max}$.
- update w_i as 1 if $\epsilon_i < max(\epsilon_{max}, \delta_{max})$ and 0, otherwise.

δ_{max} is a must since without it, this process will always remove some samples even if they are all inliers. This outlier removal scheme is very efficient. It is reported to be able to execute with *ms* for over 1000 correspondences. Inspired from this paper, it seems a better idea of applying outliers removal before constructing $\mathbf{R}, \mathbf{t}$ because of its speed.

2.4.6 CEPPnP

CEPPnP algorithm was proposed by the same author of EPPnP with the capability of leveraging feature uncertainty. As said in the introduction section, it formulate a maximum likelihood estimation problem with the paramerization of control points. Then the MLE problem is optimized iteratively. If the output needs a covariance, this method could be considered.

Summary

EPnP has several advantages compared with other methods:

- because of its novel parameterization, applying Gauss-Newton optimization is fairly easy and fast.
- because of its novel parameterization, front-end outlier removal is possible.
- EPnP is definitely one of the solution we should consider. However, its reported performance varies a lot in different papers. So it would be interesting to do a benchmark test.

2.5 Very New PnP solution

In spite of many solutions available, most of these aforementioned solutions are not able to deal with outliers. Noises and outliers, which has more serious impact on the results. My intuitive answer would be outliers since outliers mean completely wrong while noises only mean partially wrong. So those very new PnP solutions all focus on improving the capability of dealing with outliers.

The conventional way of dealing with outliers, as said in the introduction, is to cascade PnP with RANSAC. It is also mentioned that RANSAC prefers minimum solver. So a 1 point RANSAC PnP solution is proposed in [26].

2.5.1 R1PPnP

Formulation Starts from perspective projection:

$$\begin{aligned}\mathbf{X}_i^c &= \lambda_i^* \mathbf{x}_i \\ \mathbf{X}_i^c &= \mathbf{R}\mathbf{X}_i^w + \mathbf{t}\end{aligned}$$

Choosing one point as the control point $\mathbf{X}_o$, we will have

$$\lambda_i^* \mathbf{x}_i - \lambda_o^* \mathbf{x}_o = \mathbf{X}_i^c - \mathbf{X}_o^c = \mathbf{R}(\underbrace{\mathbf{X}_i^w - \mathbf{X}_o^w}_{\mathbf{S}_i})$$

Dividing with λ_o^* and let $\mu = 1/\lambda_o^*,\ \lambda_i = \mu\lambda_i^*$, we will have

$$\begin{aligned}\lambda_i \mathbf{x}_i - \mathbf{x}_o &= \mu \mathbf{R}\mathbf{S}_i \\ \mathbf{t} &= 1/\mu \mathbf{x}_o - \mathbf{R}\mathbf{X}_o^w\end{aligned}$$

The objective function for R1PPnP algorithm is given as

$$E(\mathbf{R}, \mu, \lambda_\mathbf{i}) = \sum_{i=1}^{N} \left(\frac{1}{\lambda_i} ||\lambda_i \mathbf{x}_i - \mathbf{x}_o - \mu \mathbf{R}\mathbf{S}_i||^2 \right)$$

where $\frac{1}{\lambda_i}$ is a balanced weight (without this weight may cause points with larger depths are more weighted than points with smaller depths).

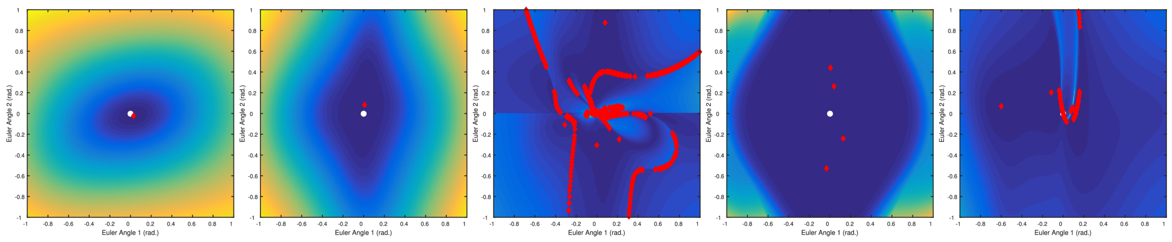
Denoting $\mathbf{p}_i = \mathbf{x}_o + \mu\mathbf{R}\mathbf{S}_i$ and $\mathbf{q}_i = \lambda_i \mathbf{x}_i$, the objective function can be rewritten as

$$E(\mathbf{R}, \mu, \lambda_\mathbf{i}) = \sum_{i=1}^{N} \left(\frac{1}{\lambda_i} ||\mathbf{p}_i - \mathbf{q}_i||^2 \right)$$

The benefits of using one control point are two fold. The first benefit is $\mathbf{t}$ is decoupled with $\mathbf{R}$. The second benefit will be described later.

As can be seen, this optimization cannot be done in a analytical way since $\mathbf{p}_i$ and $\mathbf{q}_i$ all contains some unknowns. Similar to LHM and other iterative methods, the optimization is done iteratively and alternatively: optimize $\mathbf{q}_i$ by fixing $\mathbf{p}_i$, then optimize $\mathbf{p}_i$ by fixing $\mathbf{q}_i$.

- estimating $\mathbf{q}_i$: estimating $\mathbf{q}_i$ is equivalent to estimate the relative depth λ_i since $\mathbf{x}_i$ is known. This can be achieved by projecting $\mathbf{p}_i$ along the unit vector of $\mathbf{x}_i$: $\lambda_i = \frac{\mathbf{x}_i^T \mathbf{p}_i}{\mathbf{x}_i^T \mathbf{x}_i}$.
- estimating $\mathbf{p}_i$ is equivalent to update $\mathbf{R}$, μ. Estimating $\mathbf{R}$ is nothing more than a 3D to 3D pose estimation problem. μ is updated as follows: 1) reproject $\mathbf{p}_i$ as $\mathbf{v}_i$, 2) let $\mathbf{B} = [\mathbf{v}_1 - \mathbf{x}_o, \cdots]_{3\times N}$ and $\mathbf{C} = [\mathbf{x}_1 - \mathbf{x}_o, \cdots]_{3\times N}$, then $\Delta\mu = \frac{||vector(\mathbf{C})||}{||vector\mathbf{B}||}$, $\mu_{new} = \mu\Delta\mu$. It is unclear where this updating rule comes from. However, I have tried with normal least square solution, but its performance is worse than the proposed updating rule.



(a) Cost surface (b) Proposed warp (c) Euler axis warp (d) Rot. vec. warp (e) Quaternion warp

Figure 2.4: Visualization of the PnP cost surface in the presence of noise and outliers, when optimized using different parameterizations. Color indicates the reprojection error from low (blue) to high (yellow). Fig. 1a plots this against initial camera orientation (defined by 2 Euler angles). Remaining subplots show the resulting error from a single refinement in various parameterization spaces. Diamonds indicate local minima and the white circle is the ground truth pose [45]. Image is made by [45].

Outlier Handling Outliers are handled with two schemes:

- Soft Re-weighting: use reprojection error e_i as the metrics.

$$w_i = \begin{cases} 1 & \text{if } e_i \leq H \\ \frac{H}{e_i} & \text{if } e_i > H \end{cases}$$

  where H is the inlier threshold.

- 1-point RANSAC: randomly choosing the control point.

2.5.2 HARD-PnP

HARD-PnP is another iterative optimization solution. The biggest difference of HARD-PnP, compared with all aforementioned methods, is its unique hybrid parameterization. Look the following images

From this figure, we can clearly see

- Euler angles contains lots of local minima. While rotation vector shows great advantage.
- Hybrid parameterization is more smooth.

Consequently, HARD-PnP formulate the objective function not only with one parameterization, but with several. They are jointly optimized via Convex Combination Descent.

This leaves us a new direction of how to optimize PnP algorithms.

Chapter 3

2D to 2D case

For 2D to 2D case, pose is often recorvered from either the Essential Matrix or the Homography matrix. We will explain the estimation methos for estimaing Essential matrix and Homography matrix as well as how we decompose from those matrices to obtain relevent pose.

3.1 Essential Matrix

Essential matrix or Fundamental matrix is the bridge to connect two images. By denoting image points in two images as $\mathbf{q}$ and $\mathbf{q}'$, respectively, together with the camera matrix $\mathbf{K}$, the epipolar constraint tells us:

$$\begin{aligned} \mathbf{q}'\mathbf{F}\mathbf{q} &= 0 \\ \mathbf{qn}'\mathbf{E}\mathbf{qn} &= 0,\ \textit{where}\ \mathbf{qn} = \mathbf{K}^{-1}\mathbf{q}, \mathbf{qn}' = \mathbf{K}^{-1}\mathbf{q}' \end{aligned}$$

If write explicitly,

$$\mathbf{E} = \mathbf{t}_\times \mathbf{R}$$

Clearly, the pose information is enclosed in the essential matrix, which provides us a key to recover $\mathbf{R}, \mathbf{t}$ from $\mathbf{E}$.

However, firstly, we need to know how to estimate $\mathbf{E}$ and then we can apply decomposition method to recover pose. We will use $\mathbf{q}$ instead of $\mathbf{qn}$ in the following sections for brevity.

3.1.1 Eight Point Algorithm

Eight point algorithm is the most straightforward solution for essential matrix estimation. It starts from the equation $\mathbf{qn}'\mathbf{E}\mathbf{qn} = 0$. Then, by vectorization the $\mathbf{E}$, we can have:

$$\tilde{\mathbf{q}}^T \tilde{\mathbf{E}} = 0$$

where $\tilde{\mathbf{q}} = \mathbf{q}'^T \otimes \mathbf{q}^T$. $\otimes$ means the Kronecker product. $\tilde{\mathbf{E}} = [E_{11}, E_{12}, E_{13}, E_{21}, E_{22}, E_{23}, E_{31}, E_{32}, E_{33}]$ is the vectorization of $\mathbf{E}$. Since the Degree-of-Freedom (DoF) of $\mathbf{E}$ is 8[1] and one point pair can provide one equation, we need at least eight point for estimation. That is also why this algorithm is often called Eight point algorithm. Similarly, the solution can be found by taking the smallest right singular vector.

[1]Not really 8, but the eight point algorithm omits other constraints, so the DoF is 8

Eight point algorithm is easy to implement and fast to solve. However, it doesn't work for planar case.

3.1.2 Five Point Algorithm

Theory Compared with the eight point algorithm, five point algorithm only uses 5 points to estimate **E**. This is feasible since the real DoF of **E** is 5 (3 for rotation matrix and 3 for translation matrix and minus 1 for scale).

An efficient five poitn algorithm was proposed by [46] [47]. It also starts from the normal equation. However, since we only have five point, we can only know that the solution should be in the corresponding null space:

$$\mathbf{E} = x\mathbf{X} + y\mathbf{Y} + z\mathbf{Z} + w\mathbf{W}$$

where $\mathbf{X}, \mathbf{Y}, \mathbf{Z}, \mathbf{W}$ are the corresponding null vector. In order to find **E**, we need to find x, y, z, w. We can set $w = 1$ since **E** is up to a scale. Then, we can see that if x, y, z are known, so does **E**. We need to use other two constraints for **E**.

$$det(\mathbf{E}) = 0$$
$$\mathbf{E}^T\mathbf{E}\mathbf{E} - \frac{1}{2}trace(\mathbf{E}^T\mathbf{E})\mathbf{E} = 0$$

Expand them explicitly, we will have an equation system:

A	x^3	y^3	x^2y	xy^2	x^2z	x^2	y^2z	y^2	xyz	xy	x	y	z
$\langle a\rangle$	1	·	·	·	·	·	·	·	·	·	[2]	[2]	[3]
$\langle b\rangle$	0	1	·	·	·	·	·	·	·	·	[2]	[2]	[3]
$\langle c\rangle$	0	0	1	·	·	·	·	·	·	·	[2]	[2]	[3]
$\langle d\rangle$	0	0	0	1	·	·	·	·	·	·	[2]	[2]	[3]
$\langle e\rangle$	0	0	0	0	1	0	0	0	0	0	[2]	[2]	[3]
$\langle f\rangle$	0	0	0	0	0	1	0	0	0	0	[2]	[2]	[3]
$\langle g\rangle$	0	0	0	0	0	0	1	0	0	0	[2]	[2]	[3]
$\langle h\rangle$	0	0	0	0	0	0	0	1	0	0	[2]	[2]	[3]
$\langle i\rangle$	0	0	0	0	0	0	0	0	1	0	[2]	[2]	[3]
$\langle j\rangle$	0	0	0	0	0	0	0	0	0	1	[2]	[2]	[3]

where $[N]$ denotes a polynomial of degree N in variable z. New equations can be further defined as:

$$\langle e\rangle - z\langle f\rangle = \langle k\rangle$$
$$\langle g\rangle - z\langle h\rangle = \langle l\rangle$$
$$\langle i\rangle - z\langle j\rangle = \langle m\rangle$$

Another equation system can be obtained as

B	x	y	1
$\langle k\rangle$	[3]	[3]	[4]
$\langle l\rangle$	[3]	[3]	[4]
$\langle m\rangle$	[3]	[3]	[4]

Since $[x, y, 1]$ is a null vector of $\mathbf{B}$, so the determinant of $\mathbf{B}$ must be zero, which results in a 10^{th} order polynomial of z. The roots of this tenth order polynomial can be found by finding the eigenvalues of the 10×10 companion matrix:

$$\begin{bmatrix} 0 & 0 & \cdots & -a_0 \\ 1 & 0 & \cdots & -a_1 \\ 0 & \ddots & \cdots & \vdots \\ 0 & \cdots & 1 & -a_9 \end{bmatrix}$$

where a_i is the coefficient of monomial z^i after polynomial normalization[2]

Implementation The implementation of the original Nistèr five point algorithm is a little bit complex. I have made an implementation which is available by requests. Again, Algebraic geometry can be used either by using the action matrix [48], using polynomial eigenvalue solution [49][3], and using the hidden variable resultant [50][4].

You will get up to 10 solutions using the five point algorithm. Five point algorithm also works for planar case.

3.1.3 Normalization

Normalization is a key trick to achieve more stable results when we do estimation towards essential, fundamental, and homography matrix [51]. The principle of normalization is nothing but shifting the coordinates to their center and rescaling the mean distance of each point to the center to $\sqrt{2}$.

```
pbar = mean(p,2);% p: 3xn
pdiff = p(1:2,:) - repmat(pbar(1:2), 1, n);
mdist = mean(sqrt(diag(pdiff'*pdiff)));
scale = sqrt(2)/mdist;
pnormalized(1:2,:) = scale.*pdiff;
pnormalized(3,:) = 1;
T = [scale, 0, -pbar(1)*scale; ...
  0, scale, -pbar(2)*scale; ...
  0,0,1];
```

3.1.4 Decomposition

Recover $\mathbf{R}, \mathbf{t}$ from $\mathbf{E}$ is easy.

- SVD $\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^T$.
- $\mathbf{B} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, $\mathbf{R}_1 = det(\mathbf{U}\mathbf{V}^T)\mathbf{U}\mathbf{B}\mathbf{V}^T$ and $\mathbf{R}_2 = det(\mathbf{U}\mathbf{V}^T)\mathbf{U}\mathbf{B}^T\mathbf{V}^T$.
- $\mathbf{L} = \mathbf{U} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{U}^T$ and $\mathbf{M} = -\mathbf{U} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{U}^T$, then $\mathbf{t}_1 = \frac{[L_{32}, L_{13}, L_{21}]^T}{||[L_{32}, L_{13}, L_{21}]^T||}$ and $\mathbf{t}_2 = \frac{[M_{32}, M_{13}, M_{21}]^T}{||[M_{32}, M_{13}, M_{21}]^T||}$.

[2]Normalize a n^{th} order polynomial means rescaling the coefficient of z^n to 1.
[3]This algorithm is fairly easy to implement, but the solutions can go up to 30.
[4]Easy to implement if using the Symbolic toolbox

- four solutions: $\mathbf{R}_1, \mathbf{t}_1$, $\mathbf{R}_1, \mathbf{t}_2$, $\mathbf{R}_2, \mathbf{t}_1$, $\mathbf{R}_2, \mathbf{t}_2$. Use Cheirality constraint to select the correct one.

Bibliography

[1] 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, pp. 698–700, Sep. 1987.

[2] B. K. Horn, “Closed-form solution of absolute orientation using unit quaternions,” JOSA A, vol. 4, no. 4, pp. 629–642, 1987.

[3] O. Sorkine-Hornung and M. Rabinovich, “Least-squares rigid motion using svd,” 2017.

[4] K. Kanatani, “Analysis of 3-d rotation fitting,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 16, pp. 543–549, May 1994.

[5] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour, “Closed-form solution of absolute orientation using orthonormal matrices,” J. Opt. Soc. Am. A, vol. 5, pp. 1127–1135, Jul 1988.

[6] S. Umeyama, “Least-squares estimation of transformation parameters between two point patterns,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 13, pp. 376–380, Apr. 1991.

[7] M. W. Walker, L. Shao, and R. A. Volz, “Estimating 3-d location parameters using dual number quaternions,” CVGIP: Image Underst., vol. 54, pp. 358–367, Oct. 1991.

[8] D. Eggert, A. Lorusso, and R. Fisher, “Estimating 3-d rigid body transformations: a comparison of four major algorithms,” Machine Vision and Applications, vol. 9, pp. 272–290, Mar 1997.

[9] M. A. Fischler and R. C. 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.

[10] Y. Wu and Z. Hu, “Pnp problem revisited,” Journal of Mathematical Imaging and Vision, 2006.

[11] 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.

[12] S. Li and C. Xu, “A stable direct solution of perspective-three-point problem,” International Journal of Pattern Recognition and Artificial Intelligence, vol. 25, no. 05, pp. 627–642, 2011.

[13] 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,” in Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pp. 2969–2976, IEEE, 2011.

[14] A. Ansar and K. Daniilidis, “Linear pose estimation from points or lines,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 5, pp. 578–589, 2003.

[15] L. Quan and Z. Lan, “Linear n-point camera pose determination,” IEEE Transactions on pattern analysis and machine intelligence, vol. 21, no. 8, pp. 774–780, 1999.

[16] V. Lepetit, F. Moreno-Noguer, and P. Fua, “Epnp: An accurate o (n) solution to the pnp problem,” International journal of computer vision, vol. 81, no. 2, p. 155, 2009.

[17] F. Moreno-Noguer, V. Lepetit, and P. Fua, “Accurate non-iterative o (n) solution to the pnp problem,” in Computer vision, 2007. ICCV 2007. IEEE 11th international conference on, pp. 1–8, IEEE, 2007.

[18] 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, no. 7, pp. 1444–1450, 2012.

[19] J. A. Hesch and S. I. Roumeliotis, “A direct least-squares (dls) method for pnp,” in Computer Vision (ICCV), 2011 IEEE International Conference on, pp. 383–390, IEEE, 2011.

[20] G. Nakano, “Globally optimal dls method for pnp problem with cayley parameterization.,” in BMVC, pp. 78–1, 2015.

[21] Y. Zheng, S. Sugimoto, and M. Okutomi, “Aspnp: An accurate and scalable solution to the perspective-n-point problem,” IEICE TRANSACTIONS on Information and Systems, vol. 96, no. 7, pp. 1525–1535, 2013.

[22] Y. Zheng, Y. Kuang, S. Sugimoto, K. Astrom, and M. Okutomi, “Revisiting the pnp problem: A fast, general and optimal solution,” in Proceedings of the IEEE International Conference on Computer Vision, pp. 2344–2351, 2013.

[23] L. Kneip, H. Li, and Y. Seo, “Upnp: An optimal o (n) solution to the absolute pose problem with universal applicability,” in European Conference on Computer Vision, pp. 127–142, Springer, 2014.

[24] C.-P. Lu, G. D. Hager, and E. Mjolsness, “Fast and globally convergent pose estimation from video images,” IEEE Transactions on Pattern Analysis & Machine Intelligence, no. 6, pp. 610–622, 2000.

[25] V. Garro, F. Crosilla, and A. Fusiello, “Solving the pnp problem with anisotropic orthogonal procrustes analysis,” in 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on, pp. 262–269, IEEE, 2012.

[26] H. Zhou, T. Zhang, and J. Jayender, “Re-weighting and 1-point ransac-base pnp solution to handle outliers,” IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–1, 2018.

[27] G. Schweighofer and A. Pinz, “Globally optimal o (n) solution to the pnp problem for general camera models.,” in BMVC, pp. 1–10, 2008.

[28] L. Ferraz Colomina, X. Binefa, and F. Moreno-Noguer, “Leveraging feature uncertainty in the pnp problem,” in Proceedings of the BMVC 2014 British Machine Vision Conference, pp. 1–13, 2014.

[29] 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 & Spatial Information Sciences, vol. 3, no. 3, 2016.

[30] G. Schweighofer and A. Pinz, “Robust pose estimation from a planar target,” IEEE transactions on pattern analysis and machine intelligence, vol. 28, no. 12, pp. 2024–2030, 2006.

[31] D. F. Dementhon and L. S. Davis, “Model-based object pose in 25 lines of code,” International journal of computer vision, vol. 15, no. 1-2, pp. 123–141, 1995.

[32] D. Oberkampf, D. F. DeMenthon, and L. S. Davis, “Iterative pose estimation using coplanar feature points,” Computer Vision and Image Understanding, vol. 63, no. 3, pp. 495–511, 1996.

[33] P. David, D. Dementhon, R. Duraiswami, and H. Samet, “Softposit: Simultaneous pose and correspondence determination,” International Journal of Computer Vision, vol. 59, no. 3, pp. 259–284, 2004.

[34] L. Ferraz, X. Binefa, and F. Moreno-Noguer, “Very fast solution to the pnp problem with algebraic outlier rejection,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 501–508, 2014.

[35] G. Nakano, “A versatile approach for solving pnp, pnpf, and pnpfr problems,” European Conference on Computer Vision, 2016.

[36] Y. Zheng and L. Kneip, “A direct least-squares solution to the pnp problem with unknown focal length,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1790–1798, 2016.

[37] T. Pylvänäinen, L. Fan, and V. Lepetit, “Revisiting the pnp problem with a gps,” in International Symposium on Visual Computing, pp. 819–830, Springer, 2009.

[38] L. D’Alfonso, E. Garone, P. Muraca, and P. Pugliese, “On the use of imus in the pnp problem,” in Robotics and Automation (ICRA), 2014 IEEE International Conference on, pp. 914–919, IEEE, 2014.

[39] L. D’Alfonso, E. Garone, P. Muraca, and P. Pugliese, “On the use of the inclinometers in the pnp problem,” in Control Conference (ECC), 2013 European, pp. 4112–4117, IEEE, 2013.

[40] T. Hamel and C. Samson, “Riccati observers for the nonstationary pnp problem,” IEEE Transactions on Automatic Control, 2018.

[41] S. Hadfield, K. Lebeda, and R. Bowden, “Hard-pnp: Pnp optimization using a hybrid approximate representation,” IEEE transactions on Pattern, 2018.

[42] R. Hartley and F. Kahl, “Optimal algorithms in multiview geometry,” in Asian conference on computer vision, pp. 13–34, Springer, 2007.

[43] Z. Kukelova, M. Bujnak, and T. Pajdla, “Automatic generator of minimal problem solvers,” in European Conference on Computer Vision, pp. 302–315, Springer, 2008.

[44] B. Triggs, “Camera pose and calibration from 4 or 5 known 3d points,” in 7th International Conference on Computer Vision (ICCV’99), vol. 1, pp. 278–284, IEEE Computer Society, 1999.

[45] S. Hadfield, K. Lebeda, and R. Bowden, “Hard-pnp: Pnp optimization using a hybrid approximate representation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, pp. 768–774, March 2019.

[46] D. NISTER, “An efficient solution to the five-point relative pose problem,” Proc. CVPR, June 2003, vol. 2, pp. 195–202, 2003.

[47] D. Nistér, “An efficient solution to the five-point relative pose problem,” IEEE transactions on pattern analysis and machine intelligence, vol. 26, no. 6, pp. 0756–777, 2004.

[48] H. Stewenius, C. Engels, and D. Nistér, “Recent developments on direct relative orientation,” ISPRS Journal of Photogrammetry and Remote Sensing, vol. 60, no. 4, pp. 284–294, 2006.

[49] Z. Kukelova, M. Bujnak, and T. Pajdla, “Polynomial eigenvalue solutions to the 5-pt and 6-pt relative pose problems.,” in BMVC, vol. 2, p. 2008, 2008.

[50] H. Li and R. Hartley, “Five-point motion estimation made easy,” in 18th International Conference on Pattern Recognition (ICPR’06), vol. 1, pp. 630–633, IEEE, 2006.

[51] R. I. Hartley, “In defence of the 8-point algorithm,” in Proceedings of IEEE international conference on computer vision, pp. 1064–1070, IEEE, 1995.