Mobile marker based navigation

Master-Thesis
Yue Hu

Mobile marker based navigation
Master-Thesis

Eingereicht von Yue Hu
Tag der Einreichung: 21. May 2018

Gutachter: Prof. Dr.-Ing. Jürgen Adamy
Betreuer: Raúl Acuña Godoy

Technische Universität Darmstadt
Fachbereich ETiT
Institut für Automatisierungstechnik
Prof. Dr.-Ing. Jürgen Adamy

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in dieser oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 21. May 2018 Yue Hu

Contents

1 Abstract 1

2 Introduction 2

3 Error Representations and Three-dimensional Rigid Transformation 3
- 3.1 Error representations . . . 3
  - 3.1.1 Norms . . . 3
  - 3.1.2 SVD . . . 4
  - 3.1.3 Condition number . . . 5
  - 3.1.4 Root mean square error . . . 5
- 3.2 Three-dimensional rigid transformation . . . 5
  - 3.2.1 Euclidean transformation between coordinate systems . . . 6
  - 3.2.2 Elemental rotation matrix . . . 7
  - 3.2.3 Euler angles . . . 8
  - 3.2.4 Transform matrix and homogeneous matrix representation . . . 9

4 Camera Model 11
- 4.1 The relationship between pixel plane coordinate system and image plane coordinate system 12
- 4.2 The relationship between camera coordinate system and world coordinate system . . . 13
- 4.3 The relationship between camera coordinate system and image plane coordinate system . 13
- 4.4 Camera distortion and image . . . 15

5 Non-linear Optimization 16
- 5.1 Gradient descent method . . . 16
- 5.2 Non-linear least squares . . . 18

6 Pose Estimation Algorithm 20
- 6.1 3D-2D: PnP . . . 20
  - 6.1.1 Direct linear transformation . . . 20

7 Visual Mobile Marker Odometry 22

8 Accuracy in Optical Tracking with Fiducial Markers using Condition Number 24
- 8.1 The Accuracy function . . . 24
  - 8.1.1 Accuracy function in past studies . . . 25
  - 8.1.2 The desired accuracy function . . . 27
  - 8.1.3 Homography estimation . . . 30
  - 8.1.4 Connect accuracy function with condition number . . . 32
- 8.2 Simulation design . . . 34
  - 8.2.1 Simulation frame setup . . . 34
  - 8.2.2 Simulation approach . . . 38
- 8.3 Results . . . 38

9 Accuracy of Monocular Tracking using Error Covariance Matrix 54
9.1 Gaussian error distributions . . . 54
9.1.1 Multi-dimensional Gaussian distributions . . . 54
9.1.2 Confidence ellipse/ellipsoid . . . 55
9.2 Error propagation for Gaussian distributions . . . 58
9.3 Forward propagation . . . 58
9.4 Backward propagation . . . 58
9.5 Accuracy of monocular tracking . . . 59

10 Motion Planning 63
10.1 The selected pose estimation algorithm . . . 64
10.2 A* search algorithm . . . 66
10.3 Artificial potential fields method . . . 68
10.4 Compare accuracy of paths calculated with A* and artificial potential fields method . . . 71

11 Conclusion and Future Work 78

List of Figures

3.1 Transformation between coordinate systems . . . 6
3.2 Rotation about x-axis by angle α . . . 7
3.3 Transformation from S_a to S_b . . . 9

4.1 Pinhole camera model . . . 11
4.2 Real image plane, symmetrical image plane and normalized imaging plane . . . 11
4.3 Image plane coordinate system . . . 13
4.4 Image projection relationship . . . 14
4.5 Four coordinate systems and their relationships . . . 15
4.6 Image coordinate with 480 height and 640 width . . . 15

5.1 Derivative . . . 16
5.2 Minimize cost function $J(\theta_0,\theta_1)$ with gradient descent[9] . . . 17
5.3 Four Object points and their corresponding projection points in image plane with noisy, using least squares method to calculate the optimal camera post R, t . . . 19

7.1 Marker-based(MA) pose estimation and markerless environment(MAL-VO) pose estimation[2]. . . . 22
7.2 Two-robot caterpillar. At the beginning, the robot with camera is static and the robot with marker is mobile. And then, the marker and camera exchange states which means move in turns, now the camera is mobile and marker is static. Finally, we can get the pose estimation between them[2]. . . . 23

8.1 Accuracy as a function of camera distance and relative camera angle[1] . . . 25
8.2 Error for camera distance and rotation angle[13] . . . 26
8.3 One simple example for the form of accuracy function, the influencing factors could be the camera intrinsic parameters; camera extrinsic parameters; the size of planar marker; the position of camera and so on . . . 27
8.4 The detection region of the marker is defined as a semicircle with radius 3 meter, the $3m \times 6m$ rectangle region is grid-based and each cell is $0.1m \times 0.1m$, this region would be converted into a 30×60 matrix. . . . 27
8.5 The blue square represents one cell and each blue circle represents one camera position. Condition number of this cell: $c = mean(c_1 + c_2 + c_3 + c_4)$ . . . 28
8.6 An simple example of condition number distribution matrix, which is a 30×60 matrix, the above only shows part of the condition number distribution matrix . . . 28
8.7 Process...TODO . . . 29
8.8 Homography matrix . . . 30
8.9 Normalization of 4 image points, this process can reduce condition number of matrix `A` and maximize the robustness of the DLT homography estimation against noise . . . 32
8.10 The perspective projection of the 3D plane . . . 34
8.11 A marker with size $0.3m \times 0.3m$, four features at each corner. . . . 35
8.12 Invert camera and marker . . . 36
8.13 Two-robot Caterpillar . . . 36
8.14 Camera distribution . . . 37
8.15 Colormap magma from matplotlib library . . . 39

8.16 Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15) 40
8.17 Condition number distribution(Interaction of `angle` and `height`), object points and image points are not normalized. The camera distribution is on YZ plane. The `angle` $\theta = [0°,180°]$, the step of angle is $5°$, but in real world the angle can not be $0°$ because of the constrain of DLT(three points that must be not collinear). The `height`(distance between camera and marker) as radius $r = [0.1m,3.0m]$, the step of radius is $0.1m$. 41
8.18 Symmetrical camera positions with different condition number 42
8.19 When theta is equal to $0°$ or $180°$, image points exactly same. When theta closes to 90, image points exactly very similar. When theta closes to $45°$ or $135°$, image points have relative larger differences 43
8.20 Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15) 45
8.21 Condition number distribution, object points and image points are both regular normalized 46
8.22 Test 47
8.23 Normalization 47
8.24 Condition number distribution, object points are regular normalized and image points are normalized only with translation but no scaling(scale = 1) 48
8.25 Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15) 49
8.26 Condition number distribution, object points are regular normalized and image points are normalized only with translation but no scaling(scale = 1), but $conditionnumber = 1/conditionnumber$ 50
8.27 Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15) 51
8.28 Condition number distribution, object points are regular normalized and image points are normalized with $scale = radius * np.sqrt(2)/meandist$ 52

9.1 Probability density function of 2 dimensional Gaussian distribution 54
9.2 Confidence ellipses for normally distributed data with different confidence levels[17] 56
9.3 Visualization of covariance ellipsoid for a certain confidence level[3] 57
9.4 Setup for analyzing the theoretical accuracy of a monocular tracking system with planar fiducials[3] 59
9.5 Rotation in spherical coordinate system 60
9.6 Accuracy of the pose of the camera in marker coordinates. Display with covariance ellipsoids[3] 62

10.1 Configuration space of a point-sized robot. White = C_{free}, gray = C_{obs}, green point = Start, blue point = Goal[25]. 63
10.2 Convert configuration space into a $3m \times m$ grid. The center of the marker is at $(0,3m)$ in world coordinates 64
10.3 Rotational error and translational error for the selected `IPPE` method[6] 65
10.4 Rotational error and translational error for the selected `LM` and `EPnP` methods[11] 65
10.5 Rotational error and translational error for the selected `DLS` method 66
10.6 Calculated path with `A*` search algorithm on grid. The robot can move in 8 directions(`Diagonal distance` as heuristic function)[12]. 67
10.7 Attractive potential fields + repulsive potential fields = Total potential fields[5] 68
10.8 Artificial potential fields method in our situation, in this figure the start is $(1.55m,2.05m)$ and the goal is $(1.55m,4.05m)$. 69
10.10Expanding the 8 searching neighborhoods to 24 neighborhoods 69
10.9 Artificial potential fields method in our situation 70

10.11Start point: (1.55m,2.05m), goal point: (1.55m,4.05m). The center of the planar marker is at (0,3m). Blue line: path computed with regular A*, green line: path computed with modified A* and red line: path computed with artificial potential fields method. Each point means each step on the path. 71
10.12The blue intervals and red intervals represent the mean error(euclidean distance) between desired paths and measured paths 72
10.13Compare the errors of paths computed with regular A* and artificial potential fields method 72
10.14Blue line: Mean error of regular A* computed path, red line: mean error of artificial potential fields computed path. At each point represents the standard deviation of the error. 73
10.15Blue line: Rotational error and translational error of path computed with regular A*, red line: Rotational error and translational error of path computed with artificial potential fields method. At each point represents the error standard deviation. 73
10.16Compare the rotational errors of paths computed with regular A* and artificial potential fields method 74
10.17Compare the translational errors of paths computed with regular A* and artificial potential fields method 74
10.18The blue intervals and red intervals represent the mean error(euclidean distance) between desired paths and measured paths 75
10.19Compare the errors of paths computed with modified A* and artificial potential fields method 75
10.20Blue line: Mean error of modified A* computed path, red line: mean error of artificial potential fields computed path. At each point represents the standard deviation of the error. 76
10.21Blue line: Rotational error and translational error of path computed with modified A*, red line: Rotational error and translational error of path computed with artificial potential fields method. At each point represents the error standard deviation. 76
10.22Compare the rotational errors of paths computed with modified A* and artificial potential fields method 77
10.23Compare the translational errors of paths computed with modified A* and artificial potential fields method 77

List of Tables

9.1 Table caption text 56
9.2 Table caption text 58
9.3 Table caption text 58

Code Listings

10.1 A* pseudocode 67

Acronym and symbols

Acronym	
MRSs	Multi-Robot Systems
MOMA	Mobile visual marker
2D	Two dimensional
3D	Three dimensional
DLT	Discrete linear transformation
SVD	Singular value decomposition
RMSE	Root mean square error
DOF	Degrees-of-freedom
PnP	Perspective-n-Point
P3P	Perspective-3-Point
EPnP	Efficient PnP
MA	Marker-based
MAL	Markerless
MAL-VO	Markerless visual odometry
LM	Levenberg-Marquardt

Symbols in chapter 3	
v	Vector
α	Scale
A,B	Matrix
a_{ij}	Element of matrix
λ_1	Maximum eigenvalue of matrix A^TA
U,V	Unitary matrix
Σ	Diagonal matrix
V^*	The conjugate transpose of V
$\Delta x,\Delta b$ Error	
κ	Condition number
e_{rms}	Root mean square error
e_1,e_2,e_3	Linearly independent basis vectors
a_1,a_2,a_3	Scale
a,b 3-D vector	
R	Rotation matrix
t	Translation vector
R_x,R_y,R_z	Rotation matrix, rotate around x,y,z-axis
$\alpha,\beta\gamma$	Euler angles
${}^{a}p$	Point in system a
${}^{a}r_b$	Translation vector from system a to system b
${}^{a}R_b$	Rotation matrix from system a to system b
${}^{b}p$	Point in system b
Symbols in chapter 4	
X,Y,Z	3-D position vector in cartesian coordinate
f	Focal length of the camera
u,v	2-D position vector in pixel coordinate
x,y	2-D position vector in image coordinate
X_C,Y_C,Z_C	3-D position vector in camera coordinate
X_W,Y_W,Z_W	3-D position vector in world coordinate
dx,dy	The actual size of one pixel
$u0,v0$	The center of the image plane
R	Rotation matrix
t	Translation vector
K	Camera intrinsic parameters
Symbols in chapter 5	
α	Step size
ε_i	Image noise
x_i	Error-free image coordinate
$\tilde{x}$	Image coordinate with noise
$\hat{R},\hat{t}$	Optimal camera pose

Symbols in chapter 6	
A	DLT matrix
P	4-D position vector in homogeneous coordinate
$t_1 ... t_12$	Variables in DLT
Symbols in chapter 8	
$\Delta R, \Delta t$	Transformation error
x, y	2-D position vector in image coordinate
x_i, y_i, z_i	3-D position vector in homogeneous coordinate
p_1, p_2	Two points in different planes
H	Homography matrix
$h_1 ... h_9$	Variables in homography matrix
$d_1 ... d_4$	Distance to origin of coordinate
r_1, r_2, r_3	Column vector of rotation matrix R
s	non-zero scale factor
$\overline{u}$	3-D position vector in homogeneous coordinate
$\overline{X}$	4-D position vector in homogeneous coordinate
P	Mapping matrix
$p_1 ... p_4$	Column vector of mapping matrix P
$\overline{X_\pi}$	3-D position vector in homogeneous coordinate
h	Homography matrix with 8 unknowns
ξ_i	Image noise
$\widetilde{u_i}$	Pixel coordinate with noise
$\widetilde{A}$	Matrix A with noise
$\widetilde{v_9}$	The right singular vector of $\widetilde{A}$
$\widetilde{s_9}$	The least singular value of $\widetilde{A}$
$d_1 ... d_9$	The singular values
θ	Angle
Symbols in chapter 9	
μ	Mean of random variable vector
Σ	Covariance matrix
a, b, c	Length of semi axis
$\lambda_x, \lambda_y, \lambda_z$	Eigenvalue of covariance matrix
s	Scale
l_i	Length of the axes of ellipse or ellipsoid
v	Variable
$\overline{v}$	Mean of variable
J_f	Jacobian matrix
Z_C	Scale
${}^N R^B$	Rotation matrix from system N to system B
${}^N R^A$	Rotation matrix from system N to system A
${}^A R^B$	Rotation matrix from system A to system B
α, β, r	Parameters in spherical coordinate system

$\Sigma_{\alpha,\beta,r}$	Covariance matrix
R_x	Rotation matrix, rotate around x-axis
T	Translation matrix
Σ_{v_i}	Covariance matrix
Symbols in chapter 10	
D	Minimum cost of moving from one space to an adjacent space horizontally or vertically
$D2$	The cost of moving diagonally
ζ	The positive scaling factor
q	Position of robot
q_{goal}	Position of the goal

1 Abstract

In this thesis, we present a method to describe the accuracy function based on condition number. We introduce one special method to estimate homography with normalization. In addition, we introduce our simulation design in order to verify our assumption. Also we prove that why the method using error covariance matrix to describe the accuracy function from [3] does not work in our case. Finally, based on the condition number distribution in the given region we compare the accuracy of different paths computed with `A*` search algorithm and artificial potential fields method.

2 Introduction

Visual pose estimation and localization is a very important step for robotic applications e.g. navigation and mapping. If we know the amount of knowledge about the surrounding environment e.g. the structure and geometry, with the help of camera(s) configuration(monocular, stereoscope or multi-camera) we can get possible corresponding solutions.
Multi-Robot Systems(MRSs) were first proposed by researchers since the late 1980s, are becoming increasingly in many applications, such as surveillance, search and rescue, exploration, cooperative manipulation, and transportation of objects and so on[7]. Compared to single robot systems, MRSs have several advantages on faster task completion, more time-efficient, less prone to single-points of failure, and higher estimation accuracy through sensor fusion[7] [20].
Our work is an extension of visual mobile marker odometry based on marker-based method developed by Raul Acuna, Zaijuan Li and Volker Willert[2]. A mobile two-robot caterpillar system is applied, one robot with marker called MOMA and the other one with camera called observer. The observer follows the movement of the MOMA continuously. In this case, it allows the robot system to localize itself in an unstructured environment lacking enough features and reduces the accumulated error.
In many applications the tracking accuracy plays an important role, it is very important to know how accurate the results are for a given tracking system. Hence, in our two-robot caterpillar system the accuracy function is one thing worth to study. In some previously researches such as [1] and [13] the tracking accuracy function dependent on distance as well as angle between camera and marker, however their accuracy functions were derived from experimental data, which means it is just a distribution according to rotational error and translational error. To fill this gap in this field we propose a "real" function to describe the accuracy function.
In this thesis we analyze the visual marker based tracking system and then we propose the accuracy function based on condition number used to describe the tracking system. Using this accuracy function we can find the most accurate path in the detection region of the observer in the two-robot caterpillar system. After that we identify the important factors which can influence the tracking accuracy funciton. Finally, we compare the accuracy of different paths computed with `A*` search algorithm and artificial potential fields method in this tracking system.

3 Error Representations and Three-dimensional Rigid Transformation

In this section some basic knowledge of mathematics, which are related to this thesis, would be introduced.

3.1 Error representations

3.1.1 Norms

Vector Norm:
A vector norm is a measure for the size of a vector, on a real or complex vector space V is a mapping $V \rightarrow \mathbb{R}$ with following properties:

1. $\|v\| \geq 0 \quad \forall v$
2. $\|v\| = 0 \quad \Leftrightarrow \quad v = 0$
3. $\|\alpha v\| = |\alpha| \, \|v\|$
4. $\|v + w\| \leq \|v\| + \|w\|$

1-norm:
The sum of the absolute values of all elements of vector v.

$$\|v\|_1 = \sum_{i=1}^{n} |v_i|$$

2-norm:
The square root of the sum of all absolute values from vector v.

$$\|v\|_2 = (\sum_{i=1}^{n} |x_i|^2)^{\frac{1}{2}}$$

∞-norm:
The maximum value of all elements of vector v.

$$\|v\|_\infty = \max_i |v_i|$$

p-norm:
The the 1/p power of the sum of p-th power of all vector elements.

$$\|v\|_p = (\sum_{i=1}^{n} |v_i|^p)^{\frac{1}{p}}$$

Matrix Norm:
A matrix norm has following properties:

1. $||A|| \geq 0 \quad \forall A$
2. $||A|| = 0 \quad \Leftrightarrow \quad A = 0$
3. $||\alpha A|| = |\alpha| \, ||A||$
4. $||A + B|| \leq ||A|| + ||B||$
5. $||A \cdot B|| \leq ||A|| \cdot ||B||$

First, assume that we have a $m \times n$ matrix A(m rows, n columns).
1-norm:
The maximum value of the sum of the absolute values from column vectors of matrix.

$$||A||_1 = \max_j \sum_{i=1}^{m} |a_{ij}|$$

2-norm:
The square root of the maximum eigenvalue of matrix $A^T A$.

$$||A||_2 = \sqrt{\lambda_1}$$

∞-norm:
The maximum value of the sum of the absolute values from row vectors of matrix.

$$||A||_\infty = \max_j \sum_{i=1}^{n} |a_{ij}|$$

F-norm(Frobenius-norm):
F-norm is also call Euclidean norm, is the square root of the sum of the absolute squares of matrix elements.

$$||A||_F = (\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2)^{\frac{1}{2}}$$

3.1.2 SVD

Any matrix $m \times n$ A can be factored as

$$A = U \Sigma V^*,$$

where U($m \times m$), V($n \times n$) are unitary matrix and Σ is a diagonal $m \times n$ with non-negative real number, $\mathbf{V}^*$ is the conjugate transpose of $\mathbf{V}$.
The singular values of A are the square root of the eigenvalues of A^*A:

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m \geq 0$$

The eigenvectors of A^*A consist of the columns of V. The eigenvectors of AA^* consist of the columns of U.

3.1.3 Condition number

In the field of numerical analysis, the condition number of a function with respect to an argument measures how much the output value of the function can change for a small change in the input argument. This is used to measure how sensitive a function is to changes or errors in the input, and how much error in the output results from an error in the input[22].
We now consider a system of linear equation $Ax = b$.

$$Ax = b$$
$$A(x + \Delta x) = (b + \Delta b)$$

The condition number is:

$$\kappa = \max_{\Delta b} \frac{\|\Delta x\| / \|x\|}{\|\Delta b\| / \|b\|}$$

We find that

$$\kappa \leq \|A\| \cdot \left\|A^{-1}\right\|$$

And we can find a specific Δb to satisfy this bound:

$$\kappa = \|A\| \cdot \left\|A^{-1}\right\|$$

For the 2-norm, the condition number is the ratio of maximal and minimal singular values of A respectively.

$$\kappa = \frac{\sigma_{max}(A)}{\sigma_{min}(A)},$$

3.1.4 Root mean square error

Root mean square error(RMSE) error is a useful representation of errors in measurement systems. The RMSE is used to measure the sample standard deviation of the differences between predicted and observed values. In this thesis RMSE is a measure of accuracy.

$$e_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|x - x_i|^2}$$

3.2 Three-dimensional rigid transformation

The space in our daily life is three-dimensional and we are naturally accustomed to the movement of three-dimensional space. Three-dimensional space consists of three axes, so the position of one point in 3D can be described with 3 axes. However, the rigid body we are considering now, it not only has a position but also has his own pose. The camera can also be seen as a rigid body in 3D, so the position refers to where the camera is in space, and the pose refers to the orientation of the camera.
In mathematics, we use rigid transformation to describe the motion of a rigid object in n-dimensional space. In general, the rigid transformations include rotations, translations, reflections, or their combination. To avoid ambiguity, in this thesis the rigid transformation can be only decomposed as a rotation followed by a translation. Any object will keep the same shape and size after a rigid transformation[28].

3.2.1 Euclidean transformation between coordinate systems

Recall that a vector in 3D can be represented with 3 numbers in R^3. If we determine a coordinate system, which means we can get three linearly independent basis vectors $(e1,e2,e3)$, then we can represent the vector as a linear combination of $(e1,e2,e3)$:

$$v = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1 e_1 + a_2 e_2 + a_3 e_3$$

We can use outer product to represent the rotation of a vector. For example: $w = a \times b$. The rotation from a to b can be described with w. Similar to the rotation of vectors, we can also represent the relationship of rotation between two coordinate systems and adding translation, they are seen to be as the transformation relationship between coordinate systems. During the robot's movement, the usual method is to set an inertial coordinate system(or called world coordinate system), we can assume that the world coordinate system is fixed, as shown in figure 3.1 X_W,Y_W,Z_W. At the same time, the camera or robot is a moving coordinate system, as shown in figure 3.1 X_C,Y_C,Z_C. The movement of camera is a rigid body movement, it guarantees that the length and angle of the same vector will not change in each coordinate system(euclidean transformation). For one vector p, the coordinates of p in world coordinate system and in camera coordinate system are not same, the transformation can be described with transform matrix T.

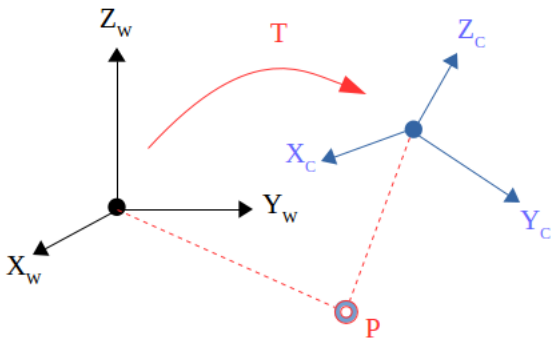


Figure 3.1: Transformation between coordinate systems

One euclidean transformation consists of rotation and translation. First we consider rotation, we assume that one orthonormal basis (e_1,e_2,e_3) after rotating becomes (e'_1,e'_2,e'_3). In this way, for one same vector a(this vector does not move!), the coordinates of vector a in both system are $\begin{bmatrix} a_1, & a_2, & a_3 \end{bmatrix}^T$ and $\begin{bmatrix} a'_1, & a'_2, & a'_3 \end{bmatrix}^T$. According to the definition of coordinate we get:

$$\begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} e'_1 & e'_2 & e'_3 \end{bmatrix} \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix}$$

In order to describe the relationship of both coordinate systems, the above equation should multiply by $\begin{bmatrix} e_1^T \\ e_2^T \\ e_3^T \end{bmatrix}$ on both left and right sides, and we get a unit matrix on the left side:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} e_1^T e_1' & e_1^T e_2' & e_1^T e_3' \\ e_2^T e_1' & e_2^T e_2' & e_2^T e_3' \\ e_3^T e_1' & e_3^T e_2' & e_3^T e_3' \end{bmatrix} \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} = Ra' \tag{3.1}$$

So the matrix R from equation 3.1 is called rotation matrix. We can find that the rotation matrix is composed of the inner product of two sets of orthonormal bases, it represents the transformation relationship of one same vector before and after rotating. we can use rotation matrix to describe camera rotation.
In addition to rotation there is also translation in euclidean transformation. One vector a in world coordinate system becomes a' after one time rotation and one time translation t, we can combine both rotation and translation as:

$$a' = Ra + t$$

Through the above formula, we describe coordinate transformation with rotation matrix R and translation vector t in euclidean space completely.

3.2.2 Elemental rotation matrix

A elemental rotation is a rotation about one of the axes of a coordinate system. The following elemental rotation matrix rotates vectors by an angle α about the x-axis in three dimensions and the right-hand rule is used.
`Right-hand rule`: right thumb points along the axis of rotation in the positive direction, and the direction of the rest curved fingers corresponds to the positive rotation direction.

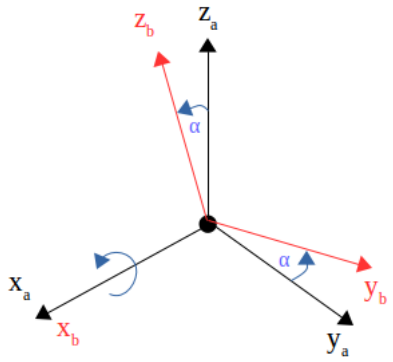


Figure 3.2: Rotation about x-axis by angle α

$$R_x(\alpha) = R(x;\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}$$

Analog formulas also work for the elemental rotation matrix by an angle β, γ about the y-axis and z-axis in 3D.

$$R_y(\beta) = R(y;\beta) = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix}$$

$$R_z(\gamma) = R(z;\gamma) = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.2.3 Euler angles

The Euler angles are three angles to describe the orientation of a rigid body with respect to a fixed coordinate system. They can also represent the orientation of a mobile frame of reference in physics or the orientation of a general basis in 3-dimensional linear algebra. Any orientation can be achieved by composing three elemental rotations, i.e. rotations about the axes of a coordinate system. Euler angles can be defined by three of these rotations[24].
In this thesis we denote euler angles as α, β, γ
The three elemental rotations may be extrinsic (rotations about the axes xyz of the original coordinate system, which is assumed to remain motionless), or intrinsic (rotations about the axes of the rotating coordinate system XYZ, solidary with the moving body, which changes its orientation after each elemental rotation)[24].
In general, euler angels can be divided in two groups(Proper Euler angles and Tait-Bryan angles):

- Proper Euler angles:

	Intrinsic rotations	Extrinsic rotations
1	$z-x'-z''$	$z-x-z$
2	$x-y'-x''$	$x-y-x$
3	$y-z'-y''$	$y-z-y$
4	$z-y'-z''$	$z-y-z$
5	$x-z'-x''$	$x-z-x$
6	$y-x'-y''$	$y-x-y$

- Tait-Bryan angles(Tait-Bryan angles are also called yaw, pitch, and roll)

	Intrinsic rotations	Extrinsic rotations
1	$x-y'-z''$	$x-y-z$
2	$y-z'-x''$	$y-z-x$
3	$z-x'-y''$	$z-x-y$
4	$x-z'-y''$	$x-z-y$
5	$z-y'-x''$	$z-y-x$
6	$y-x'-z''$	$y-x-z$

$$\text{Proper Euler angles} \begin{cases} \text{Intrinsic rotations: } \overbrace{R = Z_1(\alpha)X_2(\beta)Z_3(\gamma)}^{\text{post-multiply: } Z_1 \to X_2' \to Z_3''} \\ \text{Extrinsic rotations: } \underbrace{R = Z_3(\alpha)X_2(\beta)Z_1(\gamma)}_{\text{pre-multiply: } Z_1 \to X_2 \to Z_3} \end{cases}$$

$$\text{Tait Bryan angles} \begin{cases} \text{Intrinsic rotations: } \overbrace{R = X_1(\alpha)Y_2(\beta)Z_3(\gamma)}^{\text{post-multiply: } X_1 \to Y_2' \to Z_3''} \\ \text{Extrinsic rotations: } \underbrace{R = X_3(\alpha)Y_2(\beta)Z_1(\gamma)}_{\text{pre-multiply: } Z_1 \to Y_2 \to X_3} \end{cases}$$

There are two interpretations for chained rotations:

- pre-multiply: $R = (R_n \cdot (R_{n-1} \cdot \ldots \cdot (R_2 \cdot R_1)))$, rotation in the order 1, 2, ...,n-1, n
- post-multiply: $R = (((R_n \cdot R_{n-1}) \cdot \ldots R_2) \cdot R_1)$, rotation in the order n, n-1, ..., 2, 1

Sometimes, all kinds of sequences from Proper Euler angles and Tait-Bryan angles are called "Euler angles". In this thesis we give a example of derivation for Proper Euler angles, intrinsic rotations:

$$\begin{aligned} R = R(z;\alpha)R(y;\beta)R(z;\gamma) &= \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \\ &= \begin{bmatrix} cos(\alpha)cos(\beta)cos(\gamma) - sin(\alpha)sin(\gamma) & -cos(\alpha)cos(\beta)sin(\gamma) - sin(\alpha)cos(\gamma) & cos(\alpha)sin(\beta) \\ sin(\alpha)cos(\beta)cos(\gamma) + cos(\alpha)sin(\gamma) & -sin(\alpha)cos(\beta)sin(\gamma) + cos(\beta)cos(\gamma) & sin(\alpha)sin(\beta) \\ -sin(\beta)cos(\gamma) & sin(\theta)sin(\gamma) & cos(\beta) \end{bmatrix} \end{aligned} \tag{3.2}$$

3.2.4 Transform matrix and homogeneous matrix representation

In our case, we use rigid transformation to describe the 3D spacial transformation of one spacial point between two different coordinate systems, which is shown in figure 3.3.

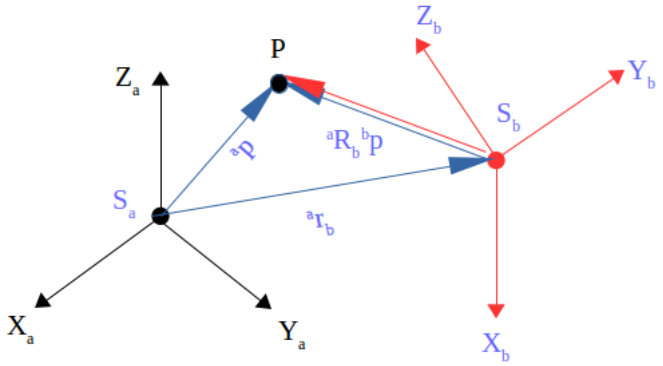


Figure 3.3: Transformation from S_a to S_b

And the transformation of point P from S_a to S_b is:

$$\boxed{{}^a p = {}^a r_b + {}^a R_b \, {}^b p} \tag{3.3}$$

- ${}^a p \in \mathbb{R}^3$: P coordinate in system S_a
- ${}^b p \in \mathbb{R}^3$: P coordinate in system S_b

- ${}^{a}r_{b} \in \mathbb{R}^3$: translation-vector
- ${}^{a}R_{b} \in \mathbb{R}^{3\times3}$: rotation between S_a and S_b

Equation 3.3 represents the rotation and translation in euclidean space. But there is still a little problem: the transformation is not linear. We assume that we make two times rotations continuously: R_1,t_1 and R_2,t_2:

$$b = R_1 a + t_1, c = R_2 a + t_2$$

But the transformation from a to c is:

$$c = R_2(R_1 a + t_1) + t_2$$

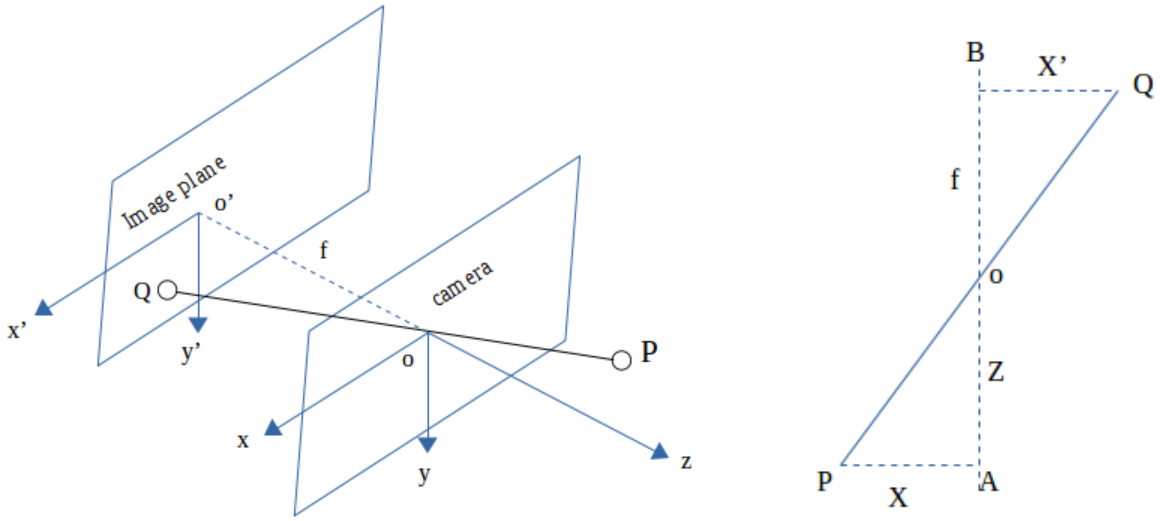
This kind of form is too complicated after many times transformations. We turn the n-dimensional euclidean space from R^n to $n+1$-dimensional by adding 1 at the end of the n-dimensional vector. It is called homogeneous coordinate.

$$\begin{bmatrix} x' \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ 1 \end{bmatrix}$$

The matrix T is called Transform Matrix. Now we get a linear transformation which additionally fulfills the requirements of additivity and homogeneity, this matrix representation is easier to use. Through matrix product of the respective homogeneous matrices we can compute the concatenation of multiple rigid transformations.

4 Camera Model

The process of the camera mapping the coordinate points(the unit is meter) in the 3D world to the 2D image plane(the unit is pixel) can be described by a geometric model. There are many different models. The simplest of them is called the pinhole model. The pinhole model is a very common and effective model, it describes a beam of light after passing through the pinhole, it's projection mapping onto the image plane.
Now let's build a geometrical model for this simple pinhole model, which is shown in figure 4.1.



(a) Camera coordinates o-x-y-z and pixel coordinates o'-x'-y'-z (b) Similar triangles

Figure 4.1: Pinhole camera model

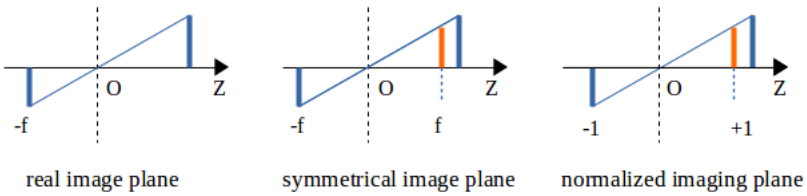


Figure 4.2: Real image plane, symmetrical image plane and normalized imaging plane

We assume that o-x-y-z is camera coordinate system, z axis is pointing to the front of the camera, x axis is pointing to the right and y axis is pointing to down. The camera's optical center is `o`, which is also the pinhole in the pinhole model. The real-world spatial point `P` through the small pinhole is projected on physical image plane, the image point is `Q`. We can assume that cartesian coordinate of `P` is $\left[X,Y,Z\right]^T$, `Q` is $\left[X',Y',Z'\right]^T$ and we assume that the distance from physical image plane to pinhole is `f`(camera focal length). Then, according to the triangle similarity theorems, we get equation 4.1:

$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'} \tag{4.1}$$

The negative sign in equation 4.1 indicates that the image is inverted. In order to simplify this model, we can move the image plane to the front of the camera symmetrically, on the same side of the camera coordinate system together with the 3D spatial point. Now we can eliminate the negative sign as shown in figure 4.2, make the equation more concise:

$$\frac{Z}{f} = \frac{X}{X'} = \frac{Y}{Y'} \tag{4.2}$$

After reforming equation 4.2, we get the following equations which are equal to equation 4.8(we will detailedly explain the relationship between camera coordinate system and image plane coordinate system in section 4.3):

$$\begin{aligned} X' &= f\frac{X}{Z} \\ Y' &= f\frac{Y}{Z} \end{aligned} \tag{4.3}$$

In next step the four plane coordinate systems in pinhole camera model and relationship between them would be introduced in the following sections:

1. Pixel plane coordinate system(u,v)
2. Image plane coordinate system(x,y)
3. Camera coordinate system(X_C,Y_C,Z_C)
4. World coordinate system(X_W,Y_W,Z_W)

4.1 The relationship between pixel plane coordinate system and image plane coordinate system

Before determining their relationship, we can assume that the physical dimensions of each pixel in the u-axis and v-axis directions are dx and dy. Their relationship is shown in figure 4.3 and we can derive the following formula:

$$u = \frac{x}{d_x} + u_0 \tag{4.4}$$

$$v = \frac{y}{d_y} + v_0 \tag{4.5}$$

dx,dy,u_0,v_0 are the parameters that we assume. dx,dy represent the actual size of the pixel on the light sensor chip. It is connected to the pixel coordinate system and the real size coordinate system. u_0,v_0 represent the center of the image plane.
We can use the knowledge of linear algebra to represent equation (4.4) and (4.5) in matrix form:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.6}$$

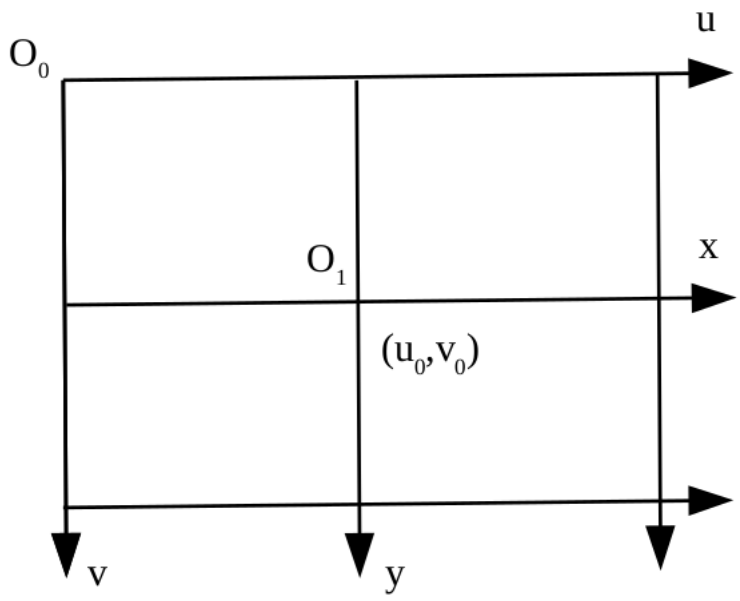


Figure 4.3: Image plane coordinate system

4.2 The relationship between camera coordinate system and world coordinate system

Through rotation matrix R and translation vector t we can get the following relationship between these two systems:

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{4.7}$$

In this equation, $R \in \mathbb{R}^{3\times3}$ rotates corresponding axes of each frame into each other and R is orthogonal($R^TR = RR^T = I$), $t \in \mathbb{R}^{3\times1}$ defines relative positions of each frame.

4.3 The relationship between camera coordinate system and image plane coordinate system

In figure 4.4, $O - X_C Y_C Z_C$ is camera coordinate system and $o_1 - xy$ is image plane coordinate system. According to the principle of triangle similarity theorems, we can derive following equations:

$$\left.\begin{aligned} \frac{x}{f} &= \frac{X_C}{Z_C} \\ \frac{y}{f} &= \frac{Y_C}{Z_C} \end{aligned}\right\} \Rightarrow \begin{cases} Z_C \cdot x = f \cdot X_C \\ Z_C \cdot y = f \cdot Y_C \end{cases} \tag{4.8}$$

Also we can use the matrix form to represent this equations:

$$Z_C \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} \tag{4.9}$$

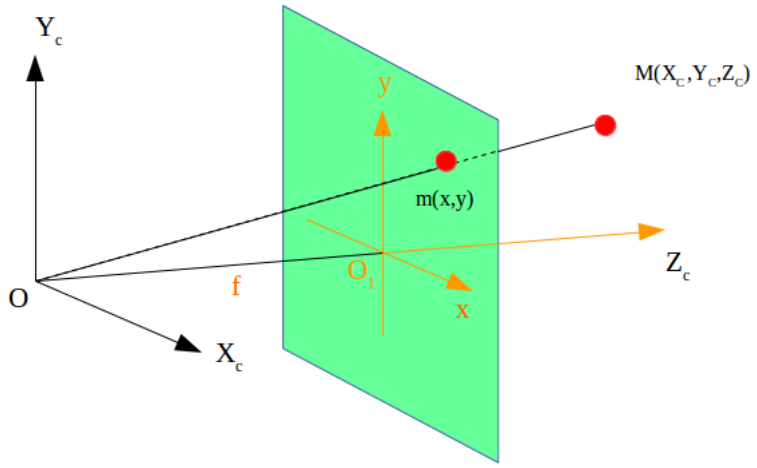


Figure 4.4: Image projection relationship

And we can get the new formula from equations 4.6, 4.7 and 4.9:

$$Z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\underbrace{\begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}}_{\text{camera and world}}}_{\text{projection relationship}}}_{\text{pixel and image}} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{4.10}$$

So the camera intrinsic parameters can be described as a matrix `K`:

$$K = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.11}$$

And `R`, `t` are the extrinsic parameters which denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates. The extrinsic matrix can be described as: $\begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$. Compared to the constant camera internal parameters, the extrinsic parameters will change with camera movement, `R` and `t` also denote the pose of the robot.

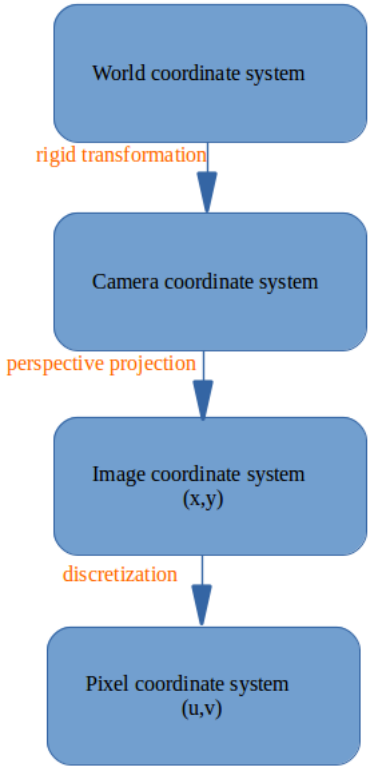


Figure 4.5: Four coordinate systems and their relationships

4.4 Camera distortion and image

"In geometric optics, distortion is a deviation from rectilinear projection; a projection in which straight lines in a scene remain straight in an image. It is a form of optical aberration[23]".
However, we do not consider the camera distortion in our project.
In mathematics images can be described by a matrix. In the computer, they occupy a continuous disk or memory space and they can be represented by a two-dimensional array. Figure 4.6 shows a image, which height is 480 pixels and width is 640 pixels:

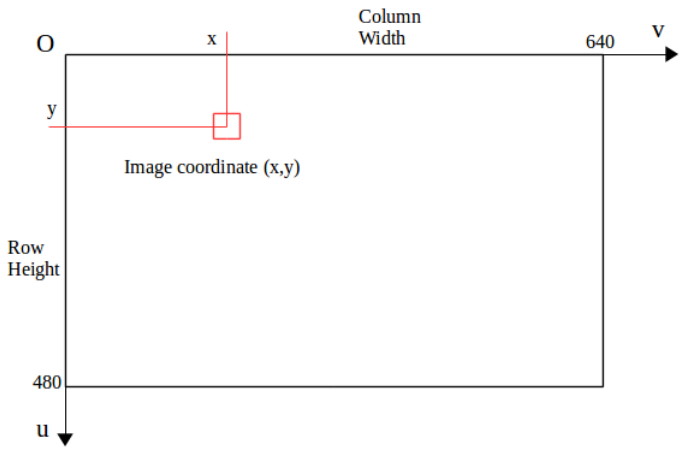


Figure 4.6: Image coordinate with 480 height and 640 width

5 Non-linear Optimization

In chapter 4, we introduce the function about camera model, and we know that we can use the transformation matrix T to describe camera pose. The transformation matrix T is computed through the camera's image model.
In the real case, because of the noises, the desired function and the measured function of T must be not exactly equal. Although the camera can be perfectly fitted with the pinhole model, but unfortunately, the data which we get is usually affected by various unknown noises. Even if we have a high-precision camera, the real function and measured function can only be approximately equal. So we have this question: how to make a camera pose estimation from noisy data accurately? This section will introduce how to handle noise data through optimization.

5.1 Gradient descent method

Gradient descent is also known as steepest descent, is first-order iterative optimization algorithm to find the minimum of a function.
Derivative:

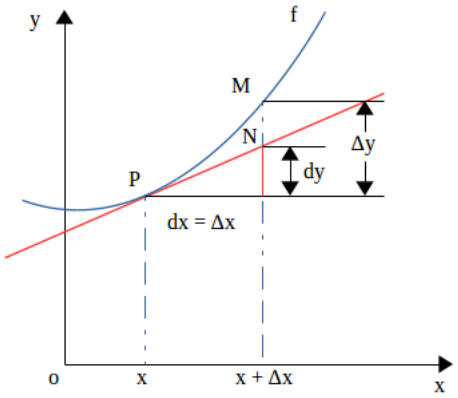


Figure 5.1: Derivative

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

Partial Derivative:

$$\frac{\partial}{\partial x_j} f(x_0, x_1, \ldots, x_n) = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x_0, \ldots, x_j + \Delta x, \ldots, x_n) - f(x_0, \ldots, x_j, \ldots, x_n)}{\Delta x}$$

The derivative and the partial derivative are essentially same. Intuitively, the partial derivative is the rate of change of the function at a certain point along the positive direction of the coordinate axis. The partial derivative is the derivative with respect to one of the variables in a multivariate function.

Directional Derivative:

$$\frac{\partial}{\partial l} f(x_0,x_1,\ldots,x_n) = \lim_{\rho \to 0} \frac{\Delta y}{\Delta x}$$
$$= \lim_{\rho \to 0} \frac{f(x_0+\Delta x_0,\ldots,x_j+\Delta x_j,\ldots,x_n+\Delta x_n) - f(x_0,\ldots,x_j,\ldots,x_n)}{\rho}$$
$$\rho = \sqrt{(\Delta x_0)^2+\cdots+(\Delta x_j)^2+\cdots+(\Delta x_n)^2}$$

In the definition of the preceding derivative and partial derivative, the change rate of the function is discussed along the positive direction of the coordinate axis. Then when we discuss the rate of change of the function in any direction, it also leads to the definition of the directional derivative, that is, the derivative value of a certain point in a direction of approach.
We must not only know the rate of change of the function in the positive direction of the coordinate axis (that is, the partial derivative), but also try to find the rate of change of the function in other specific directions. The directional derivative is the rate of change of the function in other specific directions.
Gradient:

$$\mathbf{grad} f(x_0,x_1,\ldots,x_n) = (\frac{\partial f}{\partial x_0},\ldots,\frac{\partial f}{\partial x_j},\ldots,\frac{\partial f}{\partial x_n})$$

The gradient of a function at a point is such a vector whose direction is the same as the direction in which the maximum directional derivative, and its modulus is the maximum of the directional derivative.
Gradient Descent:

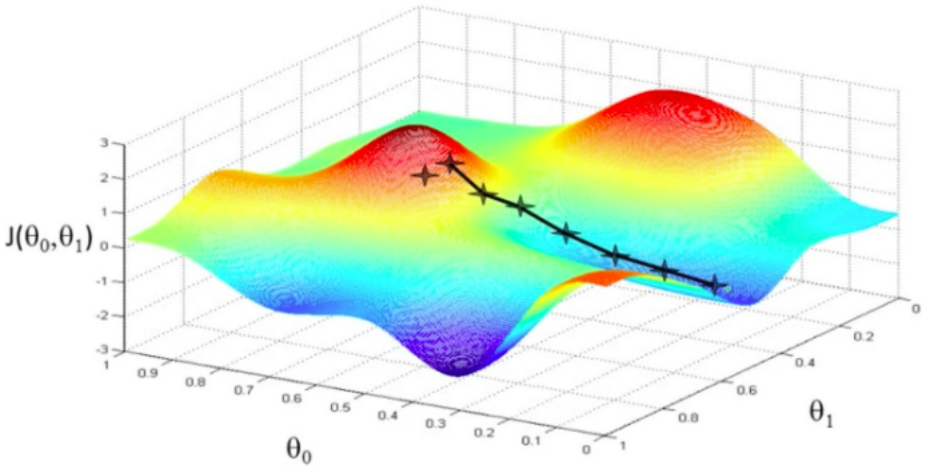


Figure 5.2: Minimize cost function $J(\theta_0,\theta_1)$ with gradient descent[9]

Since at some point in the variable space, the function has the greatest rate of change along the gradient direction, then when optimizing the objective function, it is natural to decrease the function value along the direction of the negative gradient to achieve our optimization goal.

$$\mathbf{grad} f(x_0,x_1,\ldots,x_n) = (\frac{\partial f}{\partial x_0},\ldots,\frac{\partial f}{\partial x_j},\ldots,\frac{\partial f}{\partial x_n})$$

$$
\begin{aligned}
&\text{repeat}\{\\
&\quad x_0 := x_0 - \alpha \frac{\partial f}{\partial x_0}\\
&\quad \cdots\\
&\quad x_j := x_j - \alpha \frac{\partial f}{\partial x_j}\\
&\quad \cdots\\
&\quad x_n := x_n - \alpha \frac{\partial f}{\partial x_n}\\
&\}
\end{aligned}
$$

The condition for using the gradient descent method is that the direction of gradient points to the optimal solution direction, so the gradient direction is close to the optimal solution, and the gradient is very effective information about the optimal solution. On the simpler issues such as convex function and quasi convex function, it is indeed able to satisfy this condition, so the gradient descent can work well. However, when the optimization target has a large number of local extrema, most gradient directions of the solution space no longer point to the optimal solution, so in such cases the gradient method cannot get the optimal solution any more.

5.2 Non-linear least squares

The purpose of the least squares method is to find the least squares of error. There are two kinds of the least squares method: linear and non-linear. In this thesis we only consider the non-linear case. We usually use iterative methods(e.g. Gradient descent method; Gauss-Newton method) to solve non-linear least squares problem. We update the unknown variables to get closer to the approximation solution at each step with iterative method.

$$\min_x \|f(x)\|_2^2$$

Now recall the camera model in previous section. Assume that we have n object points in world coordinate, the i-th point has it's corresponding projection on a plane with normalized image coordinate $x_i = [x_i, y_i]^T$. After adding the noise ε_i on the error-free image coordinates x_i we get noisy measurements of the image coordinates $\tilde{x} = x_i + \varepsilon_i$. Thus, we can solve this reprojection error $\|\varepsilon_i\|_2^2 = \|\tilde{x_i} - x_i\|_2^2$ of each points with least squares method for the optimal pose $(\hat{R}, \hat{t})$[15]

$$(\hat{R}, \hat{t}) = \underset{R,t}{\operatorname{argmin}} \sum_{i=1}^{n} \|\varepsilon_i\|_2^2, \qquad n \geq 3$$

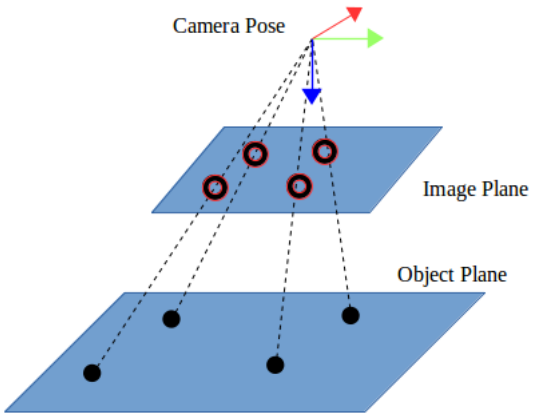


Figure 5.3: Four Object points and their corresponding projection points in image plane with noisy, using least squares method to calculate the optimal camera post R, t

6 Pose Estimation Algorithm

The camera pose consists of 6 degrees-of-freedom (DOF) which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world.

6.1 3D-2D: PnP

PnP(Perspective-n-Point) is a problem of determining the pose of a calibrated camera, which given a set of 3D points in the world coordinate system and their corresponding 2D projections in the image[27]. There are many methods to solve PnP problem, for example: we can use 3 pairs of pints to determine the camera pose(P3P); Direct Linear Transformation(DLT); Efficient PnP(EPnP); EPnP Iterative and so on. In addition, we can also use non-linear method to build a least squares problem and solve it iteratively, that is Bundle Adjustment(we did not use Bundle Adjustment method in our project).

6.1.1 Direct linear transformation

In projective geometry, direct linear transformation (DLT) is an algorithm to solve a set of variables from linear equation:

$$x_n = Ay_n, \text{ for n = 1,...,N}$$

where $\boldsymbol{x_n}$ and $\boldsymbol{y_n}$ are the known vectors and $\boldsymbol{A}$ is the matrix which contains the unknown variables need to be solved.
Considering one 3-D point P in world coordinates, it's homogeneous coordinate is $\boldsymbol{P} = (X,Y,Z,1)$. The projective points on image is $\boldsymbol{x_1} = (u_1,v_1,1)$(normalized homogeneous coordinate). At this moment, the camera pose $\boldsymbol{R}$, $\boldsymbol{t}$ is still unknown. We assume that augmented matrix $\left[\boldsymbol{R}|\boldsymbol{t}\right]$ is a 3×4 matrix which contains the information of rotation and translation(it is a little different to transformation matrix T). We write down it's expanded form:

$$s\begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{6.1}$$

Using the last row of above equation to eliminate s, we get 2 constraints:

$$u_1 = \frac{t_1X + t_2Y + t_3Z + t_4}{t_9X + t_{10}Y + t_{11}Z + t_{12}}, \qquad v_1 = \frac{t_5X + t_6Y + t_7Z + t_8}{t_9X + t_{10}Y + t_{11}Z + t_{12}} \tag{6.2}$$

We reform the formulas and get:

$$\begin{bmatrix} t_1 & t_2 & t_3 & t_4 \end{bmatrix} P - \begin{bmatrix} t_9 & t_{10} & t_{11} & t_{12} \end{bmatrix} u_1 = 0,$$
$$\begin{bmatrix} t_5 & t_6 & t_7 & t_8 \end{bmatrix} P - \begin{bmatrix} t_9 & t_{10} & t_{11} & t_{12} \end{bmatrix} v_1 = 0.$$

We notice that `t` is the variable to be solved, we can find that each feature point(`P` and $\boldsymbol{x_1}$) provide two linear constrains about `t`. Assume that we have N feature points all together, we can get following set of equations:

$$\begin{bmatrix} P_1^T & 0 & -u_1 P_1^T \\ 0 & P_1^T & -v_1 P_1^T \\ \vdots & \vdots & \vdots \\ P_N^T & 0 & -u_N P_N^T \\ 0 & P_N^T & -u_N P_N^T \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \\ t_{11} \\ t_{12} \end{bmatrix} = 0. \tag{6.3}$$

Because `t` has total 12 variables, so we need at least 6 pairs of feature points to solve this `t`. If we have more than 6 pairs of feature points, we can use `SVD` to solve overdetermined system of linear equations with least squares method.
Normally, `DLT` is a useful method to solve PnP problem. Also in our case we used `DLT` method to solve the condition number later.

7 Visual Mobile Marker Odometry

Visual pose estimation and localization is very important not only for robotic applications but also useful for augmented reality. Through camera configuration(monocular, stereoscopic or multi-camera) in different environment as well as the amount of cognition about the structure and geometry of the environment we can find relatively reliable solutions to this problem.
A mobile multi-robot system which was firstly introduced in [10] can obviously speed-up and improve the accuracy of the localization of each of the robots. In this case, the concept of Visual Mobile Marker Odometry(MOMA) was proposed with the idea of cooperative positioning for multiple robots[2].
Visual pose estimation can be divided into two different categories:

1. Marker-based(MA): using some detectable visual landmarks such as fiducial markers or 3D scene models with known coordinates of its features/keypoints[2].

2. Markerless(MAL): based on static scene features with unknown absolute coordinates in the scene(without any knowledges of 3D scene)[2].

Both cases of pose estimation are shown in figure 7.1.

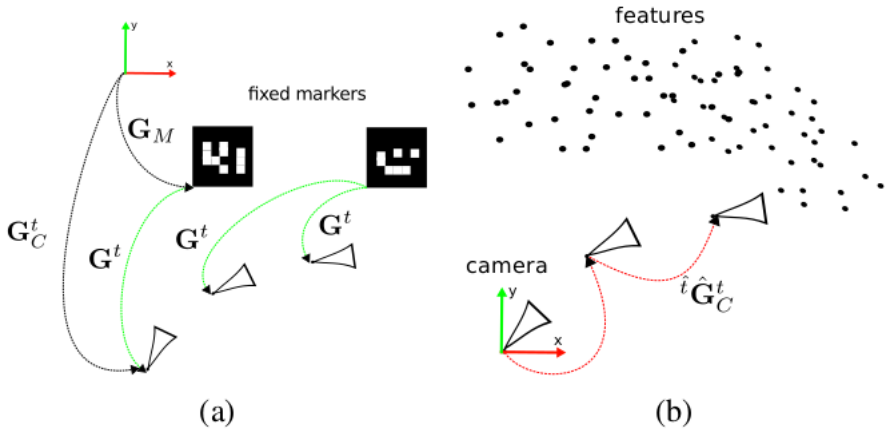


Figure 7.1: Marker-based(MA) pose estimation and markerless environment(MAL-VO) pose estimation[2].

Following we make a comparison of marker-based method and markerless method in some aspects:

	Pros	Cons
MA	not drift; reduce error from 3D to 2D only on image plane; relatively higher accuracy and lower computational complexity	require artificial markers in the environment
MAL	without modification of artificial marker; can be used to detect real-world objects	unavoidable drift; additional errors; need enough brightness and contrast of the environment

For our current work we only considered the case of marker-based(MA) method and we did the extended work based on MOMA with two-robot caterpillar: one robot is carried with camera(monocular) as observer and another is set with fiducial marker(MA) as MOMA.
The robot with camera and the robot with marker both have two states:

- Mobile: the robot is moving or permitted to move.
- Static: otherwise.

The motion pattern for two-robot caterpillar is shown in figure 7.2. This is the background of our simulation scene.

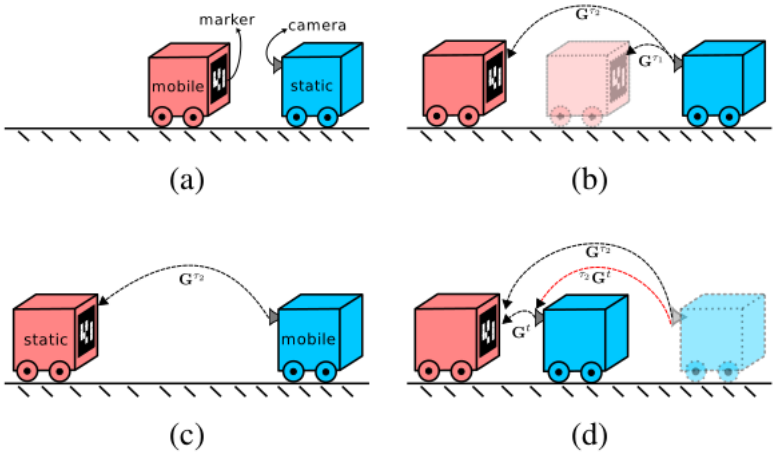


Figure 7.2: Two-robot caterpillar. At the beginning, the robot with camera is static and the robot with marker is mobile. And then, the marker and camera exchange states which means move in turns, now the camera is mobile and marker is static. Finally, we can get the pose estimation between them[2].

8 Accuracy in Optical Tracking with Fiducial Markers using Condition Number

It was already proven that a square is a very stable and robust configuration for all camera poses in planar fiducial markers and using four optimal control points with maximal equal distance to each other has the biggest influence on the improvement of accuracy[15]. Based on this conclusion, we also placed four control points on a planar visual fiducial marker, which are set as features in the world coordinate system.
We already know that the optical tracking with fiducial markers is commonly used in robotic applications or in augmented reality systems. One part of our work based on the MOMA odometry system which was shown in figure 7.2 was to find the accuracy function. The accuracy function in our case was used to describe the specific distribution of tracking accuracy dependent on positional relationship(such as distance or angle) between camera and marker. With this accuracy function we can calculate how the robot with camera and marker should be located and oriented, then we can get a optimal camera pose with minimal transformation error(ΔR,Δt).
In previous analysis from other researchers it was already clear proven that the error in homography estimation is dependent on the singular values of the matrix A in the DLT algorithm[4]. Furthermore in [15] there is a research for optimal control point configurations for homography and pose estimation. The authors investigated a approach to find the optimal control point based on condition number of the matrix A in the DLT algorithm. There is a detailed derivation why we can use condition number of the matrix A in the DLT algorithm as the solution to get the optimal control point configurations and their research results indicated that there is a relationship between the rotational error, translational error of pose estimation and condition number of matrix A: They have the same trend, which means if the condition number gets larger the rotational error, translational error also get larger. According to their research results we proposed our assumption:
`The condition number of matrix A in the DLT algorithm can be used to interpreted as the accuracy function.`

- If one camera pose has relative larger condition number we can assume that this pose estimation of this position has low accuracy.
- If one camera pose has relative smaller condition number we can assume that this pose estimation of this position has high accuracy.

Following through building program simulations like the scheme in [15] ,we would verify this assumption step by step.

8.1 The Accuracy function

The accuracy function is used to describe:
In one given detected area of a fiducial marker how accurate one position is for a given tracking situation. The ultimate goal of accuracy function is in order to calculate how virtual objects should be located and oriented.
We wanted to find an accuracy function depending on the distance and angle between marker and camera.

8.1.1 Accuracy function in past studies

Due to the widespread use of monocular tracking systems, several groups have already worked on analyzing the accuracy of planar marker tracking systems. In the past studies about tracking accuracy in optimal tracking with fiducial markers are all based on experimental data, their accuracy functions were based on rotational error and translational error. Nobody actually proposed a general function to describe the tracking accuracy.
In [1] the researchers described the accuracy as a function with experimental data, they used ARToolKit to track a single cardboard marker. They set the marker on a stable construction as fixed and allowed a variation around the y-axis. They mounted the camera in a mechanical jig and adjusted in a way that the center of the lens and the center of the marker were on the same height[1].
Their "accuracy function" assesses how accurate the tracking parameters delivered by ARToolKit are in the specific distance and angle to the marker if the marker was correctly recognized in the first place.
Their results show a specific distribution of tracking accuracy dependent on distance as well as angle between camera and marker. Their result is shown in figure 8.1.

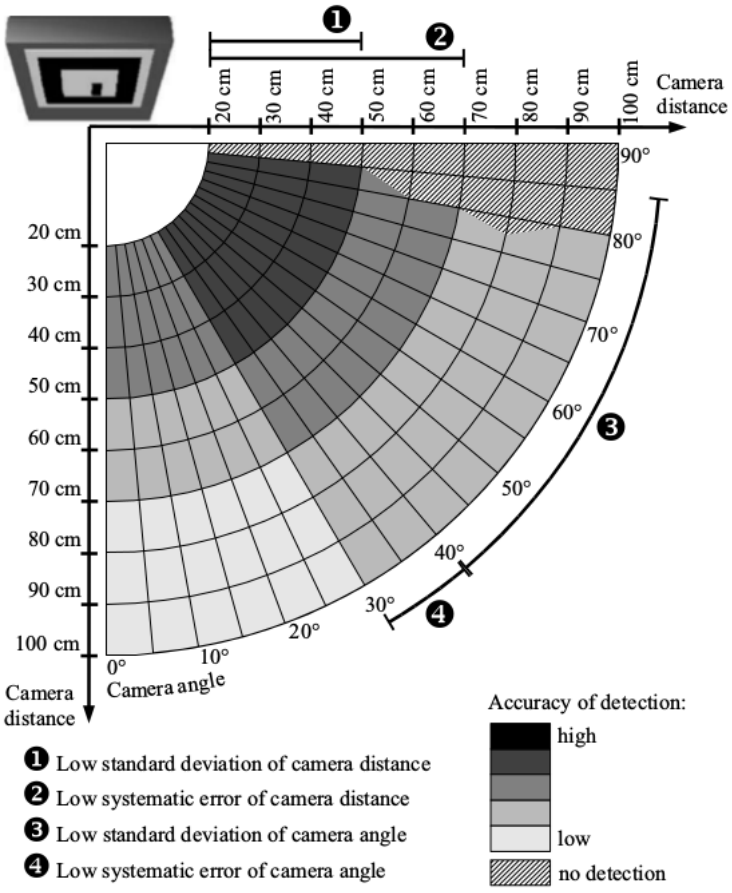


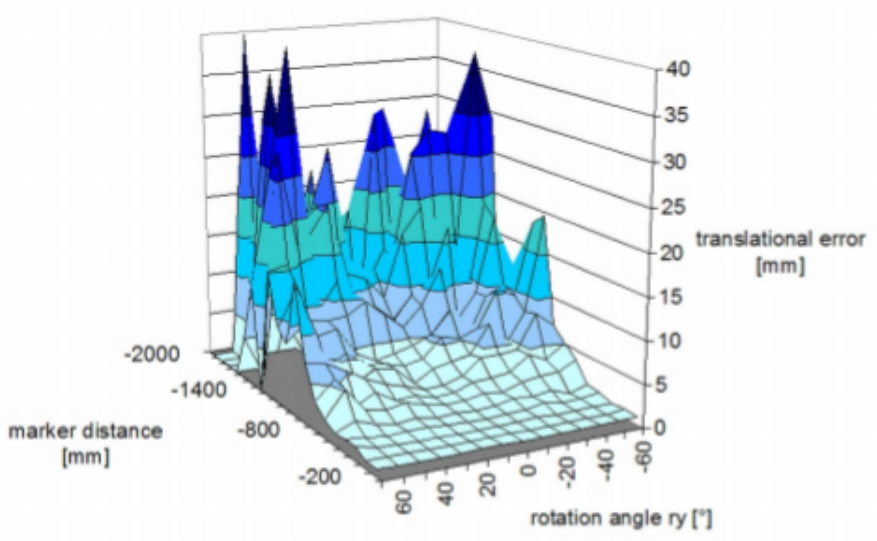
Figure 8.1: Accuracy as a function of camera distance and relative camera angle[1]

Another research "Analysis of Tracing Accuracy for Single-Camera Square-Marker-Based Tracking"[13] also presented a tracking accuracy analysis based on simulated ground truth data. Their accuracy evaluation of the visual marker based tracking system are based on the error and standard deviation. Their study only looks at the corner detection error but not consider pose estimation errors. Their experimental design follows "one-factor-at-a-time" principle and then also consider the combinations of all input factors. I also learned from this idea, which means firstly I studied the influence of height(distance of the marker) and angle(viewing angles) separately, after that I studied the combination of these influenced

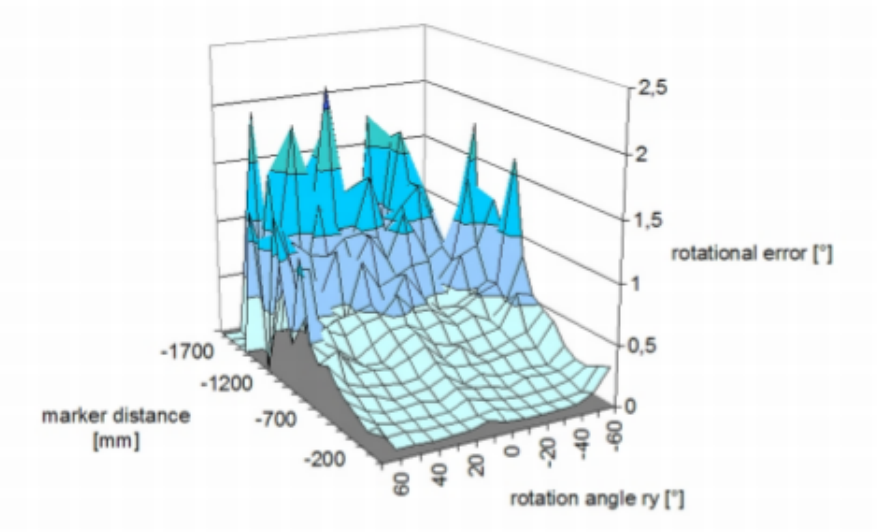
factors which means height and angle would be considered at the same time. They provided the following results:

- For larger distances the translational pose error and rotational pose error both increase.
- When the camera and marker form a certain small angle, it gets the smallest translational pose error and rotational pose error. But if the angle too small or the camera gradually becomes vertical to the marker, the translational pose error and rotational pose error would get bigger.

These results are shown in figure 8.2 which are actually similar to the results in [1], but just in different form to represent it.
We can find that both researches [1] and [13] were based on experimental data to describe the tracking accuracy, but they actually did not provide a "traditional function" to describe the accuracy, just like this simple form $f(x,y,z) = ax + by + cz$, where `f` is the desired accuracy function, `x`, `y`, `z` are the possible influencing factors of the accuracy function.



(a) Translation error



(b) Rotation error

Figure 8.2: Error for camera distance and rotation angle[13]

8.1.2 The desired accuracy function

To fill this gap in this field, in our project we created the real "accuracy function", which means in the detection area of the given marker and camera, if one position coordinate is given we can obtain the information of accuracy at this corresponding position from the "accuracy function".

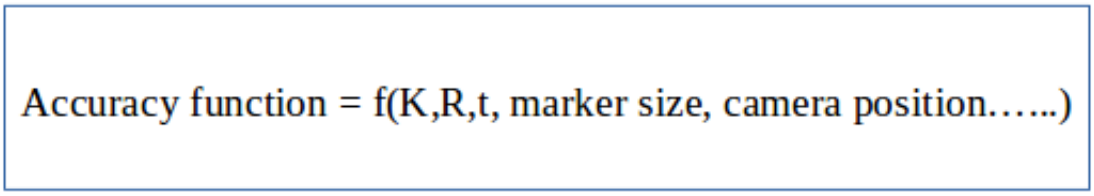


Figure 8.3: One simple example for the form of accuracy function, the influencing factors could be the camera intrinsic parameters; camera extrinsic parameters; the size of planar marker; the position of camera and so on

In this project we proposed the assumption that the condition number distribution can be interpreted as our desired accuracy function.
Firstly, we convert the selected grid-based rectangle region which contains the detection region of the marker into a $m \times n$ matrix. This process is shown in figure 8.4.

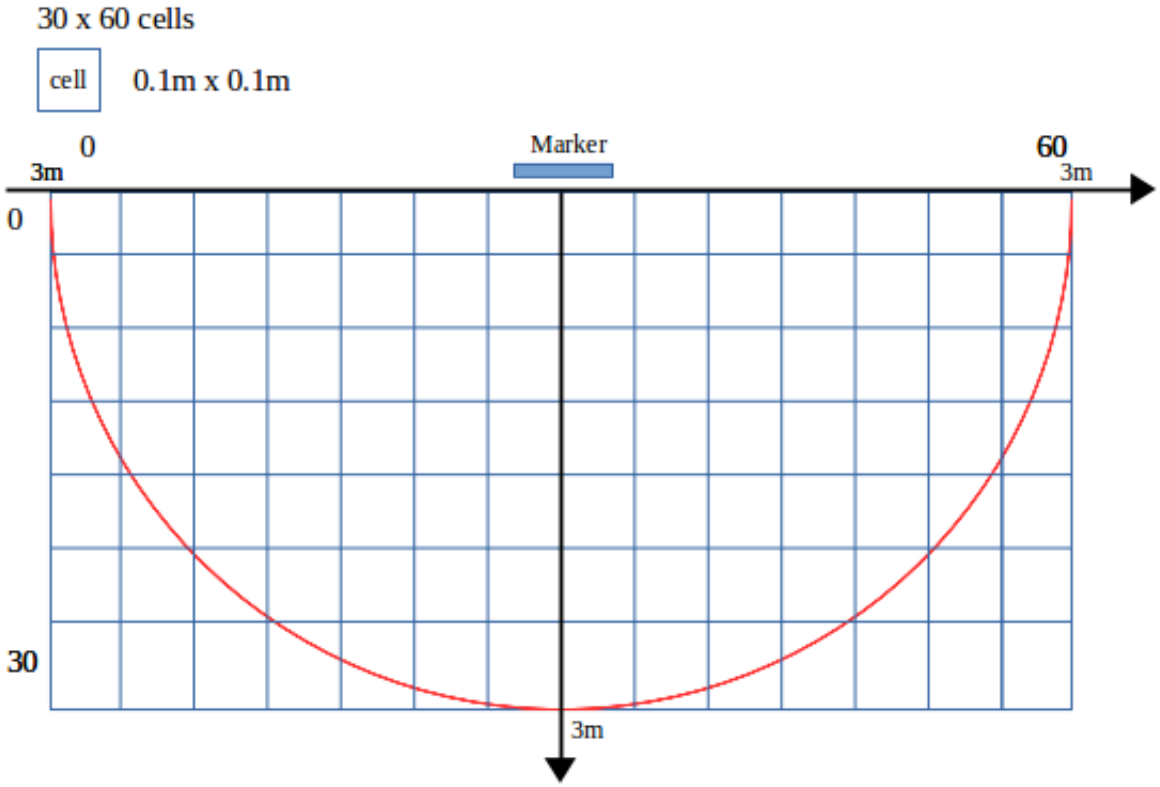


Figure 8.4: The detection region of the marker is defined as a semicircle with radius 3 meter, the $3m \times 6m$ rectangle region is grid-based and each cell is $0.1m \times 0.1m$, this region would be converted into a 30×60 matrix.

For each grid-cell there maybe several camera poses locate on it. Then we compute the average value c of all condition numbers of these camera poses fall onto the same cell, this average value c can be treated as the condition number of this cell. This process is shown in figure 8.5.

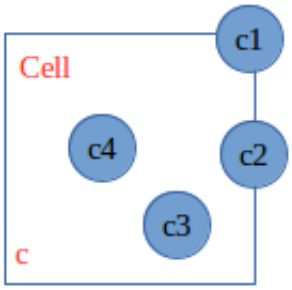


Figure 8.5: The blue square represents one cell and each blue circle represents one camera position. Condition number of this cell: $c = mean(c_1 + c_2 + c_3 + c_4)$

After that we can get a similar condition number distribution like the matrix shown in figure 8.6.

```
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
1.811762 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.811762 1.725612
1.954545 2.243047 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 2.243047 1.954545 1.847819
2.049936 2.151494 2.388954 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 2.388954 2.151494 2.049936 1.966950
2.142366 2.255670 2.369003 2.534848 0.000000 0.000000 0.000000 0.000000 2.534848 2.369003 2.255670 2.142366 2.081250
2.253009 2.298274 2.418171 2.526683 2.607755 2.648836 2.648836 2.607755 2.526683 2.418171 2.298274 2.253009 2.204429
2.348455 2.398833 2.455166 2.522025 2.580096 2.613640 2.613640 2.580096 2.522025 2.455166 2.398833 2.348455 2.310038
2.440128 2.517620 2.502842 2.565374 2.611258 2.631553 2.631553 2.611258 2.565374 2.502842 2.517620 2.440128 2.465963
2.571016 2.640031 2.592441 2.649694 2.685119 2.694226 2.694226 2.685119 2.649694 2.592441 2.640031 2.571016 2.633222
2.726870 2.723442 2.710033 2.755953 2.784570 2.788112 2.788112 2.784570 2.755953 2.710033 2.723442 2.726870 2.736317
2.890135 2.828608 2.846743 2.882465 2.906462 2.902740 2.902740 2.906462 2.882465 2.846743 2.828608 2.890135 2.837923
3.052322 2.949870 2.990031 3.020841 3.041913 3.034506 3.034506 3.041913 3.020841 2.990031 2.949870 3.052322 2.995137
3.222848 3.101038 3.140544 3.170300 3.189954 3.175292 3.175292 3.189954 3.170300 3.140544 3.101038 3.222848 3.178705
3.352716 3.269236 3.303004 3.328184 3.345173 3.327769 3.327769 3.345173 3.328184 3.303004 3.269236 3.352716 3.351056
3.490134 3.439652 3.468495 3.490500 3.505529 3.487279 3.487279 3.505529 3.490500 3.468495 3.439652 3.490134 3.534308
3.639648 3.611091 3.639221 3.660210 3.673942 3.647320 3.647320 3.673942 3.660210 3.639221 3.611091 3.639648 3.721664
3.761769 3.786832 3.814158 3.832096 3.843987 3.816735 3.816735 3.843987 3.832096 3.814158 3.786832 3.761769 3.901684
3.936863 3.966102 3.989780 4.007232 4.018975 3.989827 3.989827 4.018975 4.007232 3.989780 3.966102 3.936863 4.090275
4.119677 4.148292 4.171090 4.185579 4.195495 4.165960 4.165960 4.195495 4.185579 4.171090 4.148292 4.119677 4.270856
4.305033 4.329447 4.349182 4.364193 4.374461 4.344639 4.344639 4.374461 4.364193 4.349182 4.329447 4.305033 4.401403
4.492534 4.516234 4.535054 4.548961 4.557951 4.516829 4.516829 4.557951 4.548961 4.535054 4.516234 4.492534 4.558981
4.681850 4.701381 4.717153 4.731547 4.740848 4.699425 4.699425 4.740848 4.731547 4.717153 4.701381 4.681850 4.659059
4.864215 4.887903 4.906318 4.917750 4.925276 4.883615 4.883615 4.925276 4.917750 4.906318 4.887903 4.864215 4.899784
5.056651 5.076071 5.091438 5.103256 5.111034 5.069188 5.069188 5.111034 5.103256 5.091438 5.076071 5.056651 5.033704
5.246177 5.265213 5.280573 5.291737 5.299248 5.255967 5.255967 5.299248 5.291737 5.280573 5.265213 5.246177 5.222956
5.436566 5.455696 5.467954 5.479481 5.487232 5.443805 5.443805 5.487232 5.479481 5.467954 5.455696 5.436566 5.413637
5.628223 5.643579 5.658836 5.670170 5.676122 5.632580 5.632580 5.676122 5.670170 5.658836 5.643579 5.628223 5.610168
5.820486 5.839132 5.850794 5.859682 5.865816 5.822186 5.822186 5.865816 5.859682 5.850794 5.839132 5.820486 5.797767
6.013827 6.028730 6.040752 6.049910 6.056229 6.012532 6.012532 6.056229 6.049910 6.040752 6.028730 6.013827 5.996038
6.203607 6.218966 6.231351 6.240782 6.247286 6.203542 6.203542 6.247286 6.240782 6.231351 6.218966 6.203607 6.185267
```

Figure 8.6: An simple example of condition number distribution matrix, which is a 30×60 matrix, the above only shows part of the condition number distribution matrix

After that if we want, we can divide the condition numbers into several degrees between the maximum and minimum, treating them as accuracy distribution. Finally, we can get the accuracy degree at given camera position. This whole precess is shown in figure 8.7.

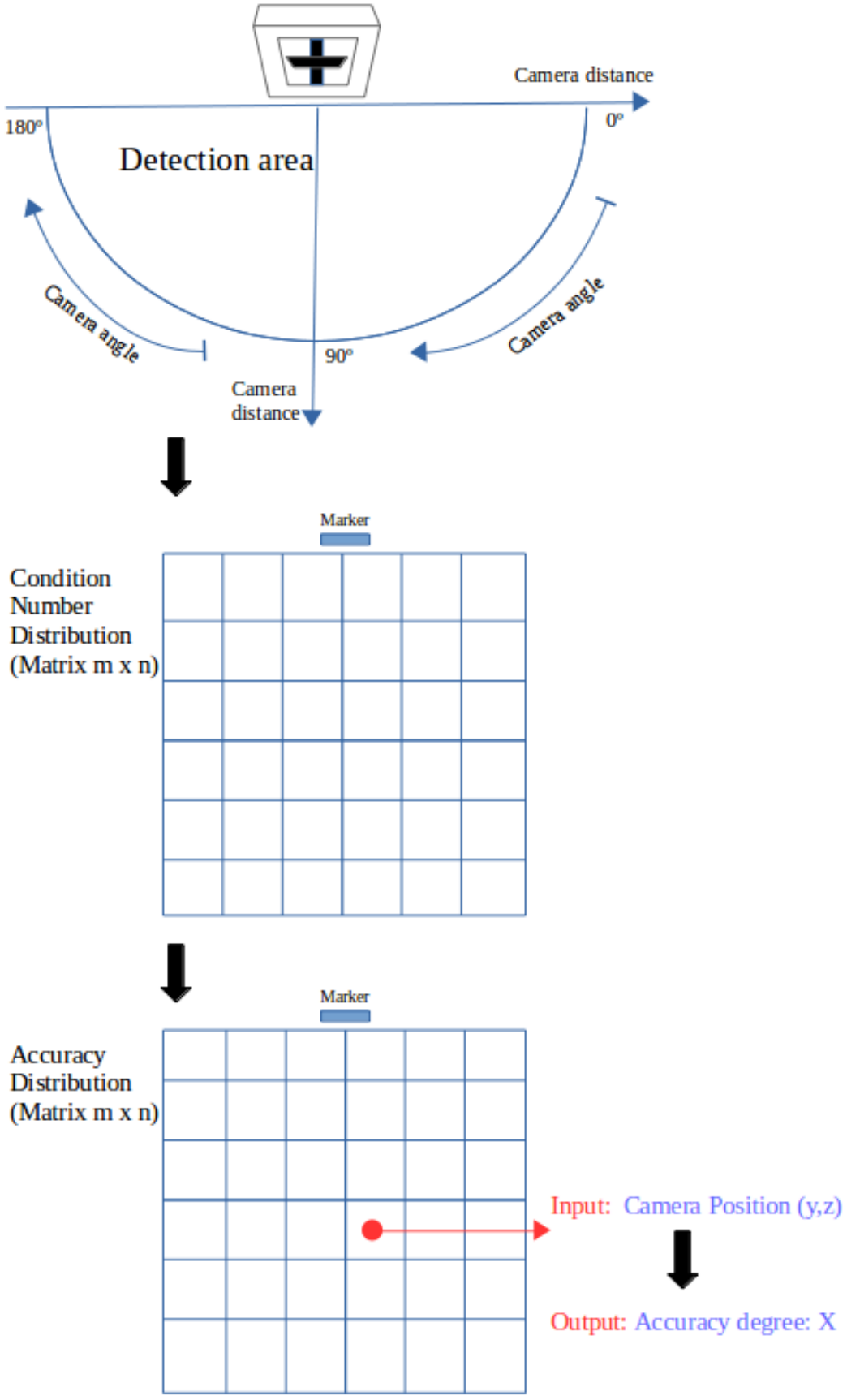


Figure 8.7: Process...TODO

From this we can find that the accuracy distribution(accuracy function) is affected by camera intrinsic, extrinsic parameters, the size of marker and the camera pose. And the key to find the accuracy function is : `Find out the condition number distribution.`
Also the condition number distribution was used for our motion planning algorithms in chapter 10. In the following sections I will explain detailedly why the condition number distribution can be interpreted as the accuracy function and how we can get the condition number distribution in our simulations.

8.1.3 Homography estimation

What is a homography?
A 2D point (x,y) in the image plane can be represented as (x_i,y_i,z_i) in 3D plane, where $x = \frac{x_i}{z_i}$ and $y = \frac{y_i}{z_i}$. This process is called homogeneous representation of a 2D point. And the homography matrix is a non-singular 3×3 matrix which describes the linear mapping relationship between two planes in 2D homogenous coordinates(usually describes the transformation of some points in the common plane between two images, shown in figure 8.8).

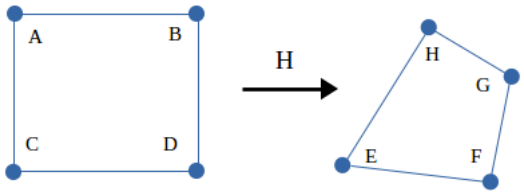


Figure 8.8: Homography matrix

We assume that there is a pair of matched feature points p_1 and p_2 in two different planes. Directly we can use homography matrix to describe the transformation between p_1 and p_2:

$$p_2 = Hp_1$$

$$\begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

where H is the homography matrix. We set $h_9 = 1$ and then reform the above formula:

$$h_1u_1 + h_2v_1 + h_3 - h_7u_1u_2 - h_8v_1u_2 = u_2$$
$$h_4u_1 + h_5v_1 + h_6 - h_7u_1v_2 - h_8v_1v_2 = v_2$$

From above equations we find that there are now 8 unknowns($h_1 \sim h_8$) existed in 2 equations. In order to solve `H` we still need additional six equations, which means we need totally 4 pairs of points.
The Direct Linear Transform(DLT) is a common standard algorithm used to solve for the homography matrix `H` given a sufficient set of corresponding point. We notice that `H` can be changed by multiplying by an arbitrary non-zero constant without altering the projective transformation. So usually h_9 should be set as $h_9 = 1$. In this case a homogeneous matrix only has 8 degrees of freedom even though it contains 9 elements, which means we only need to solve these 8 unknown variables. From above equation we know that each pair of matched points can build one constrain for DLT. So we need 4 pairs of matched points to solve homography matrix `H`. And in our project the detected marker has 4 corner points for

estimating of homography. After estimating the homography we can get our expected rotation matrix R and translation vector t from corresponding homography matrix H.

`Normalizing transformations:`

For DLT algorithm the normalization of the measurements is a important step to improve the quality of the estimated homography. Normalization of the data consists of translation and scaling of image coordinates. This normalization should be carried out before applying the DLT algorithm.

Recall that the DLT formula 6.3 for 6 pairs of points, we can derive a similar formula for 4 pairs of points:

$$Ah = \begin{bmatrix} P_1^T & 0 & -u_1 P_1^T \\ 0 & P_1^T & -v_1 P_1^T \\ \vdots & \vdots & \vdots \\ P_4^T & 0 & -u_4 P_4^T \\ 0 & P_4^T & -u_4 P_4^T \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{bmatrix} = 0. \tag{8.1}$$

The normalization is basically a preconditioning to decrease condition number of the matrix A(when there is noise in the measurement data, the more severe the change in the condition number, the more sensitive the method is to errors, and the less robust the method is. So low condition number of matrix `A` for our simulation is desired). And it is already proven that the DLT algorithm is in practice invariant to similarity transformations[8].

"Apart from improved accuracy of results, data normalization provides a second desirable benefit, namely that an algorithm that incorporates an initial data normalization step will be invariant with respect to arbitrary choices of the scale and coordinate origin. This is because the normalization step undoes the effect of coordinate changes, by effectively choosing a canonical coordinate frame for the measurement data.[8]"

For our case we also used the `isotropic scaling.` method which is also mentioned in [8]. We can summarize the steps of isotropic scaling as follows:

1. The four image points(also the four object points) are translated so that their centroid is at the origin.
2. Then the four points are scaled so that the mean distance from the origin is $\sqrt{2}$.

This process is shown in figure 8.23.

`Why is normalization essential?`

There is already a complete good answer in [8]. All in all we apply normalization is in order to set all entries in matrix `A` have similar magnitude and increase the robustness of accuracy function.

However, the normalization still has some disadvantages:

- The estimated homography depends on the similarity transforms.
- The normalization matrices are computed from noisy pixel measurements, and therefore sensitive to outliers.

To overcome these disadvantages a (non-iterative)method for estimating homographies does not rely on coordinate normalization is proposed in [14] which is by avoiding normalization and instead using a `Taubin estimator`, that is unaffected by similarity transformations of the correspondences in the two views.

In our project we will compare the results of the accuracy function for estimating homographies without normalization and with normalization.

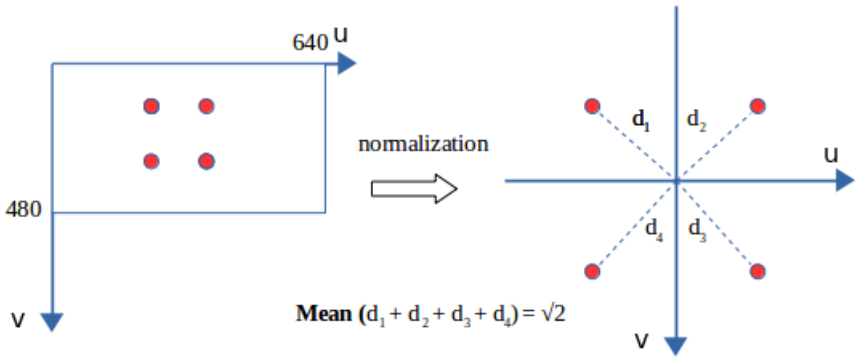


Figure 8.9: Normalization of 4 image points, this process can reduce condition number of matrix `A` and maximize the robustness of the DLT homography estimation against noise

8.1.4 Connect accuracy function with condition number

In this section I will explain why the condition number distribution can be interpreted as the accuracy function.

`Planar Point Configuration for Pose Estimation:`
In previous section we already introduced the pinhole camera model and the pose estimation from 3D points. And in our project we applied a planar marker to estimate the camera pose. Specially if the control points are all on a plane, we can reduce the degree of freedom of the rotational matrix `R`.(Now we can use two angles instead of three angles to represent rotational matrix `R`:

$$\boxed{R = [r_1, r_2, r_3] \rightarrow R = [r_1, r_2]}$$

Because all control points are located on a plane, we can define a 2D subspace instead of three-dimensional coordinates in 3D world, set the Z-coordinate as 0:

$$X_i^W = \begin{bmatrix} X_i^W \\ Y_i^W \\ Z_i^W \end{bmatrix} \Longrightarrow X_i^W = \begin{bmatrix} X_i^W \\ Y_i^W \\ 0 \end{bmatrix}$$

Under the perspective projection model, through the projection matrix `P` mapping three-dimensional space points `X` onto `u` on the image plane. The transformation can be represented as following equation:

$$s\overline{u} = P\overline{X}$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P \begin{bmatrix} X^W \\ Y^W \\ Z^W \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix} \begin{bmatrix} X^W \\ Y^W \\ Z^W \\ 1 \end{bmatrix}$$

where $\overline{u} = \begin{bmatrix} u, & v, & 1 \end{bmatrix}$, $\overline{X} = \begin{bmatrix} X^W, & Y^W, & Z^W, & 1 \end{bmatrix}$ are represented as homogeneous coordinates and `s` is the non-zero scale factor. We assume that plane π is the plane where our fiducial marker are located(plane

π is the `XOY` plane of the object coordinate system). For points on the plane π, there is $Z_i = 0$, then we can get following deformation:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P\begin{bmatrix} X^W \\ Y^W \\ 0 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix}}_{\mathrm{H}}\begin{bmatrix} X^W \\ Y^W \\ 1 \end{bmatrix}$$

where $H = \begin{bmatrix} p_1, & p_2, & p_3 \end{bmatrix}$, we can describe $\overline{X_\pi}$ as $\overline{X_\pi} = \begin{bmatrix} X^W, & Y^W, & 1 \end{bmatrix}$, then we get following equation:

$$s\bar{u} = H\overline{X_\pi} \tag{8.2}$$

Similar to 6 pairs of points DLT algorithm derivation in previous section and in our project we used a fiducial marker with four control points(any 3 points are not collinear) on a plane. Solving equation 8.2 is equal to solve the following equation, which has the similar form as formula 8.1:

$$\boxed{Ah = 0}$$

$$A = \begin{bmatrix} 0 & 0 & 0 & -X_1 & -Y_1 & -1 & v_1X_1 & v_1Y_1 & v_1 \\ X_1 & Y_1 & 1 & 0 & 0 & 0 & -u_1X_1 & -u_1Y_1 & -u_1 \\ 0 & 0 & 0 & -X_2 & -Y_2 & -1 & v_2X_2 & v_2Y_2 & v_2 \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -u_2X_2 & -u_2Y_2 & -u_2 \\ 0 & 0 & 0 & -X_3 & -Y_3 & -1 & v_3X_3 & v_3Y_3 & v_3 \\ X_3 & Y_3 & 1 & 0 & 0 & 0 & -u_3X_3 & -u_3Y_3 & -u_3 \\ 0 & 0 & 0 & -X_4 & -Y_4 & -1 & v_4X_4 & v_4Y_4 & v_4 \\ X_4 & Y_4 & 1 & 0 & 0 & 0 & -u_4X_4 & -u_4Y_4 & -u_4 \end{bmatrix}$$

where $A \in \mathbb{R}^{8\times 9}$ and $h = \begin{bmatrix} r_1^T, & r_2^T, & t^T \end{bmatrix}^T \in \mathbb{R}^9$. Under additional constraint $\|h\| = 1$, $\mathbf{h}$ is the homography matrix with 8 unknowns.

However in the actual situation we set noisy(Gaussian distribution) for each image points, again assuming noisy measurements for image points $\widetilde{u_i} = u_i + \xi_i$, we get the matrix $\widetilde{A} = A + E$ with noise. Due to the presence of noise, in the most cases $\widetilde{A}h \neq 0$, so $\mathbf{h}$ would be estimated by minimization of $\widetilde{A}h$, due to $Ah = 0$, this problem can be solved also by minimization of $\|Eh\|_2^2$:

$$h = argmin_h \left\|\widetilde{A}h\right\|_2^2 = argmin_h \|Eh\|_2^2, \quad with \quad \|h\| = 1 \tag{8.3}$$

Solving equation 8.3 is actually equivalent to apply a singular value decomposition(SVD) of $\widetilde{A} = \widetilde{U}\widetilde{S}\widetilde{V}$ for this DLT algorithm, where the solution of equation 8.3 is $h = \widetilde{v_9}$ with $\widetilde{v_9}$ being the right singular vector of $\widetilde{A}$, associated with the least singular value $\widetilde{s_9}$.

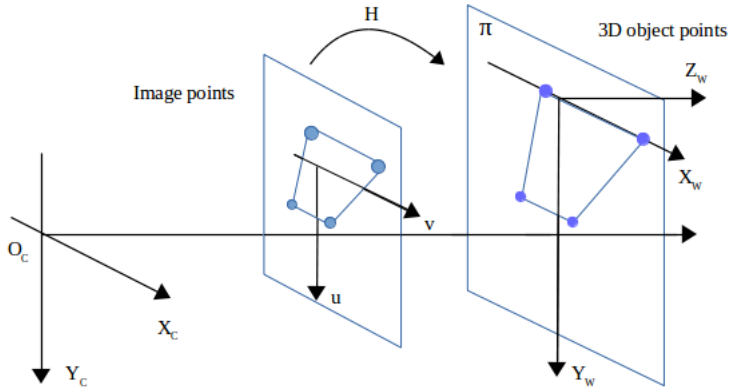


Figure 8.10: The perspective projection of the 3D plane

`Why is condition Number?`
For the linear problem, the condition number of the measurement matrix is an important factor affecting its numerical stability.
If the singular values of matrix A are $d_1, d_2, \cdots, d_9 (d_1 \geq d_2 \cdots \geq d_9 = 0)$, then the condition number of matrix A is:

$$c(A) = \frac{d_1}{d_8}$$

In previous researches from [1] and [13] they described the accuracy function with the rotational error and translational error. And in [15] there is a interesting results, when the rotational error and translational error decrease or increase, the condition number at corresponding position also becomes smaller or bigger. They have a similar trend in terms of different camera pose. Through this discovery we therefore made our assumption that we can use condition number to describe the accuracy function. And through our simulations we already verified this assumption.
In following sections I will explain how our simulations were performed.

8.2 Simulation design

8.2.1 Simulation frame setup

For the simulation the two most important objects are: `marker` and `camera`. In order to make simulation between marker and camera, we need to set some properties of both.

Marker

We used a square fiducial marker for camera pose estimation. More precisely the four vertices of the square marker are defined as the control points, which are then needed to project onto the camera image. Because in [15] it was already verified that for 4 points in most cases the vertices of a square marker are the optimal control points, therefore these four vertices are represented as our detected features for camera.
We set that the center of the marker is the origin of world coordinate system$(O - X_W Y_W Z_W)$. And the fiducial planar marker was located on `XOY` plane in the world coordinate system. The positive direction of Z-axis points to the region where the cameras should be located on. The coordinates of the four control points under world coordinate system are:

1. Point 1: $(0.15,0.15)$
2. Point 2: $(0.15,-0.15)$
3. Point 3: $(-0.15,-0.15)$
4. Point 4: $(-0.15,0.15)$

In the simulation the default size of the marker was set as $0.3m \times 0.3m$ and we set the marker as fixed at the origin of the coordinate system, of course we can change different size of the marker to fit more testing scenarios.
The simulation scene of the planar marker is illustrated in figure 8.11.

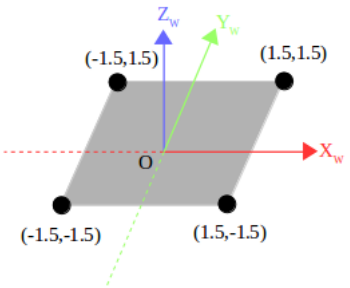


Figure 8.11: A marker with size $0.3m \times 0.3m$, four features at each corner.

Camera

We already introduce the camera model in previous section and it is well-known that we often use $\begin{bmatrix} u & v & 1 \end{bmatrix}^T$ to represent the position of a 2D point in pixel coordinates, corresponding $\begin{bmatrix} X_W & Y_W & Z_W & 1 \end{bmatrix}^T$ is used to represent the position of 3D point in world coordinate system. According to the pinhole camera model, we know the following projective mapping from world coordinate system to pixel coordinate system:

$$Z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R|t \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{8.4}$$

where K is the intrinsic parameters and we set it's default value as $\mathbf{K} = \begin{bmatrix} 800 & 0 & 0 \\ 0 & 800 & 240 \\ 0 & 0 & 1 \end{bmatrix}$, the extrinsic parameters `R` and `t` are the variables we are interested in. By setting different camera positions in our simulation we obtain different camera pose, that is the rotation matrix `R` and translation vector `t`. It is worth noting that in our project we did not consider the camera distortion problem for now.
We set the image default size as $640 \times 480[pixel^2]$, through derivation of camera intrinsic parameters `K` in previous section we can easily find that parts of `K` parameters are related to the height and width of the image. Therefore if we change the size of the image, we also need to change the camera intrinsic parameters `K` correspondingly.
Although we do not need to consider the camera distortion problem, in the real environment camera noise is an unavoidable factor. In order to simulate the reality of camera we set the image noise manually. For each image point we added a 2-dimensional Gaussian distribution:

```
def addnoise_imagePoints(self, imagePoints, mean =0, sd =2):
    """ Add Gaussian noise to image points
    imagePoints: 3xn points in homogeneous pixel coordinates
    mean: zero mean
    sd: pixels of standard deviation
    """
    imagePoints =np.copy(imagePoints)
    if sd >0:
        gaussian_noise =np.random.normal(mean,sd,(2,imagePoints.shape[1]))
        imagePoints[:2,:] =imagePoints[:2,:] +gaussian_noise
    return imagePoints
```

camera distribution:

Normally each camera has a fixed detection range, in our simulation we fixed the position of planar marker, in this way we can image that each marker has a detection area just like the camera. Therefore the cameras can be treated as moved in the detection region of the marker. This idea is shown in figure 8.12.

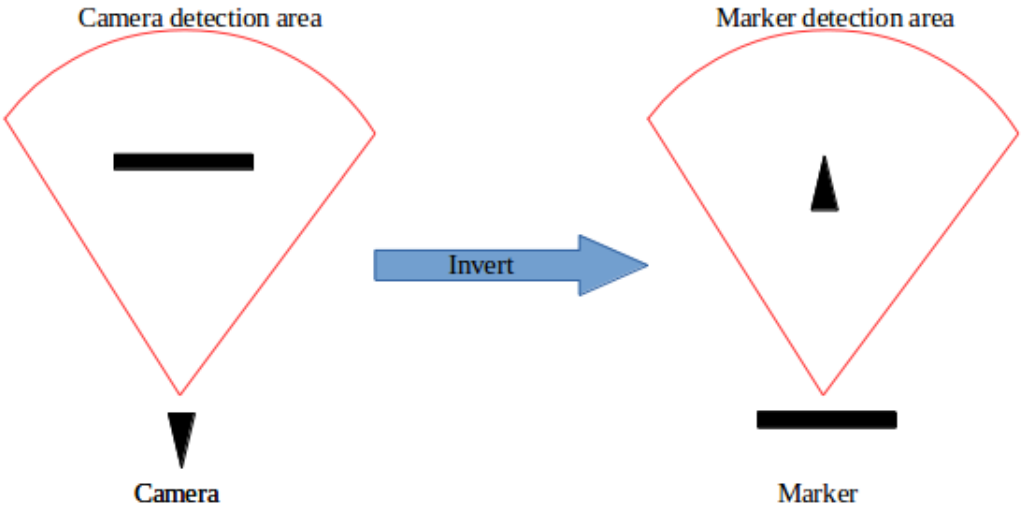


Figure 8.12: Invert camera and marker

Because in our situation the robot moves only on YZ-plane of world coordinates and the heights of the canter of the marker and camera lens always keep at the same level, this scenario is shown in figure 8.13. Therefore we can simplify the 3D camera distribution shown in figure 8.14(a) into the 2D camera distribution shown in figure 8.14(b).

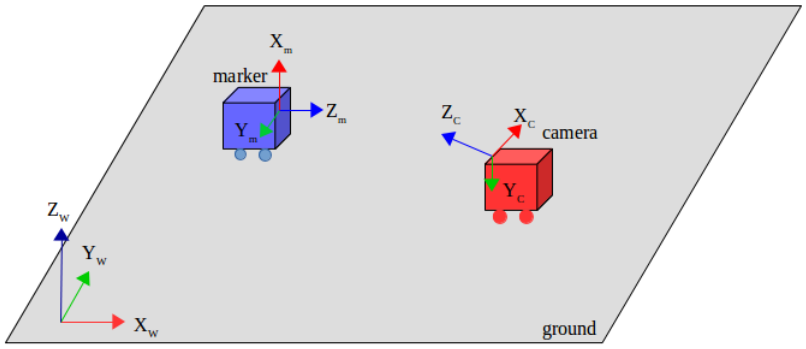
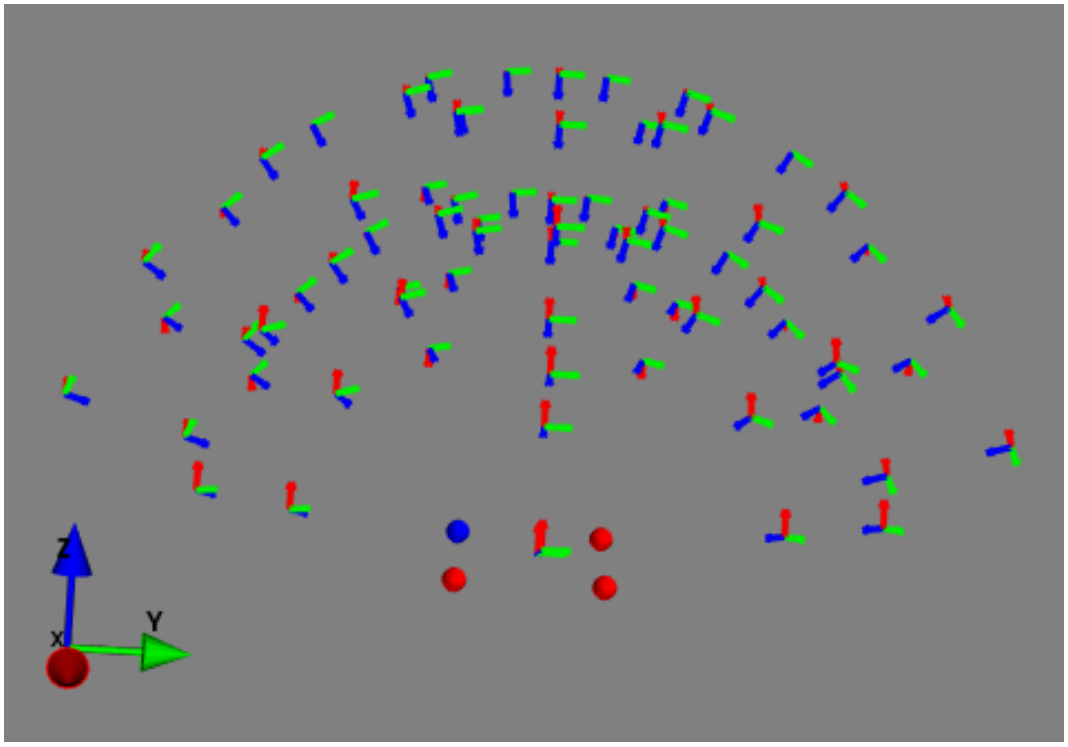
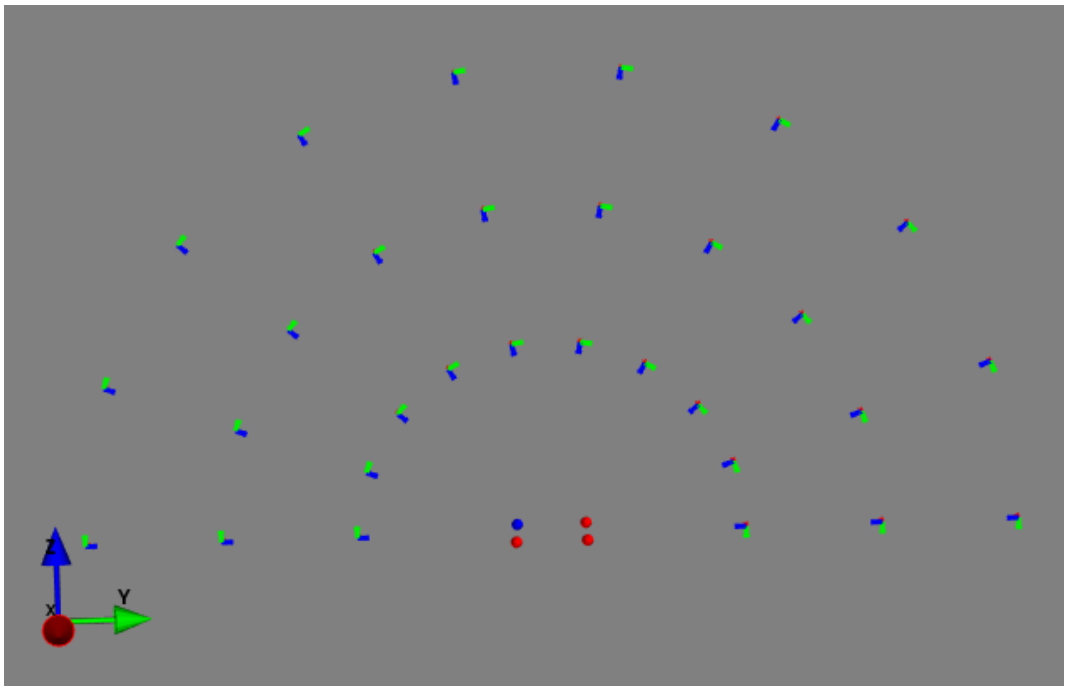


Figure 8.13: Two-robot Caterpillar



(a) An Example for camera distribution in 3D. The camera are distributed evenly on spheres of different radius. Each camera(positive direction of Z-axis) should look at the origin of the plane.



(b) An Example for camera distribution in 2D. The camera are distributed evenly on YZ-plane. Each camera(positive direction of Z-axis) should look at the origin of the plane. We apply this 2D case in our simulation.

Figure 8.14: Camera distribution

In our simulation we set a very important rule:
`Each camera should look at the center of the planar marker!`

8.2.2 Simulation approach

A very common approach for simulations or experiments is the "one-factor-at-a-time" method[26], which means we should study only one affected factor while all others are kept constant. But this approach fails to consider the effect of multiple variable combinations on one system or on one function. So beyond this approach we also need to consider the combinations of all inputs. For example in our simulation we need to consider not only the influence of height and angle separately but also the interactions of height and angle.
Therefore we applied "one-factor-at-a-time" method at the beginning of the simulation, trying to find out some rules or clues for accuracy function, we can treat it as heuristics. On the basis of that we estimated interactions between different factors and trying to find out the final experimental conclusion from our simulation.
In our simulation we studied two important factors:

- `Height`: the distance between camera and marker, we can also call it `radius`.
- `Angle`: the angle between Z-axis of camera and marker plane.

8.3 Results

We have already pointed out that the condition number distribution can be interpreted as the accuracy function and later the condition number distribution can be used in our pathfinding. Therefore in this section we provide the results all about condition number distribution.
Also according to the research in [13], our desired results are:

- If the angle between camera and marker plane is fixed, the closer the distance between camera and marker, the smaller the condition number is.
- If the distance between camera and marker is fixed, the smaller the angle between camera and marker plane, the smaller the condition number is.

In previous sections we already mentioned that how important the normalization of homography is. In this part different results without normalization and with normalization would be compared detailedly. Then Therefrom we found out the best solution which is satisfied our desired results.
In order to compute the condition number distribution the object points and image points were transformed like follows in four ways:

1. object points: no normalized, image points: no normalized
2. object points: regular normalized, image points: regular normalized

```
scale =np.sqrt(2) /meandist # Translation and scaling
```

3. object points: regular normalized, image points: normalized

```
scale =1 # Only translation, no scaling
```

4. object points: regular normalized, image points: normalized

```
scale =radius*np.sqrt(2) /meandist
```

where in the second case the `regular normalized` means the four points are scaled so that the mean distance from the origin is $\sqrt{2}$, for the third case the four image points are only translated but not scaling, for the fourth case the four image points are scaled so that the mean distance from the origin is $radius * \sqrt{2}$(radius is the distance from camera to marker).
In order to represent the results clearly, we used colormap `magma` form `matplotlib` library[19] to represent our data set:

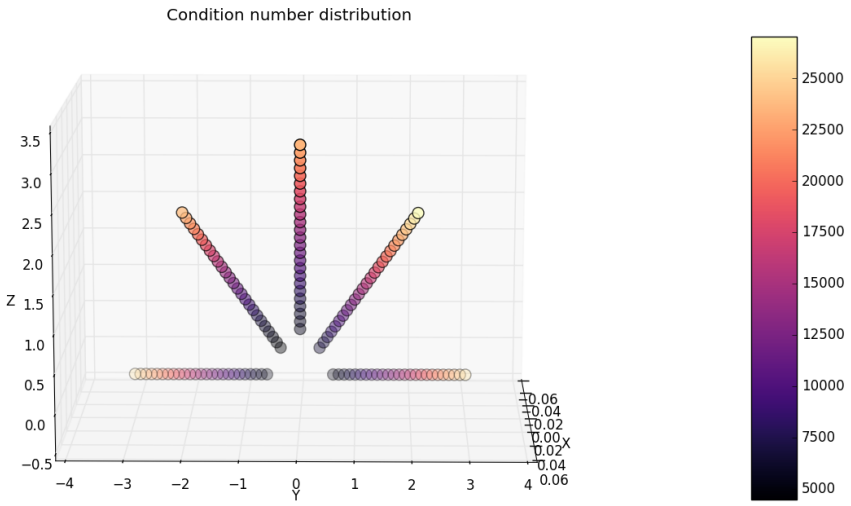
- Dark color(black) → small condition number
- Light color(yellow) → big condition number

Figure 8.15: Colormap magma from matplotlib library

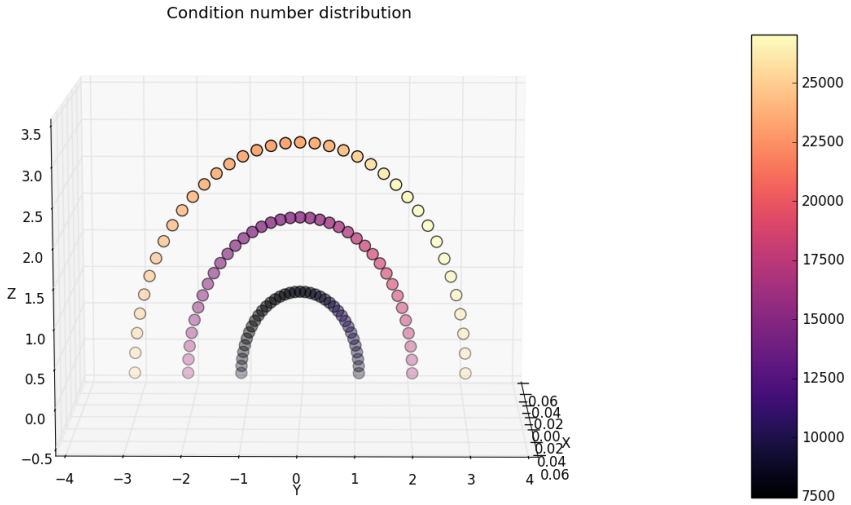
`Condition number distribution without normalized homography:`
At first, we computed the condition number of each camera pose without normalization of the homography.

```
# Compute the condition number without normalization
cond_num =gd.matrix_condition_number_autograd(*input_list, normalize =False)
```

And we applied "one-factor-at-a-time" method that we made simulations for `angle` and `height` separately and we got the following results which are shown in figure 8.16:



(a) Influence of the height(distance between the cam1era and the marker) on the condition number of different camera positions while the angle is fixed as 0°,45°,90°,135° and 180°



(b) Influence of the angle on the condition number of different camera positions while the height(distance between camera and marker) is fixed as 1.06666667m, 2.0333333 and 3m

Figure 8.16: Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15)

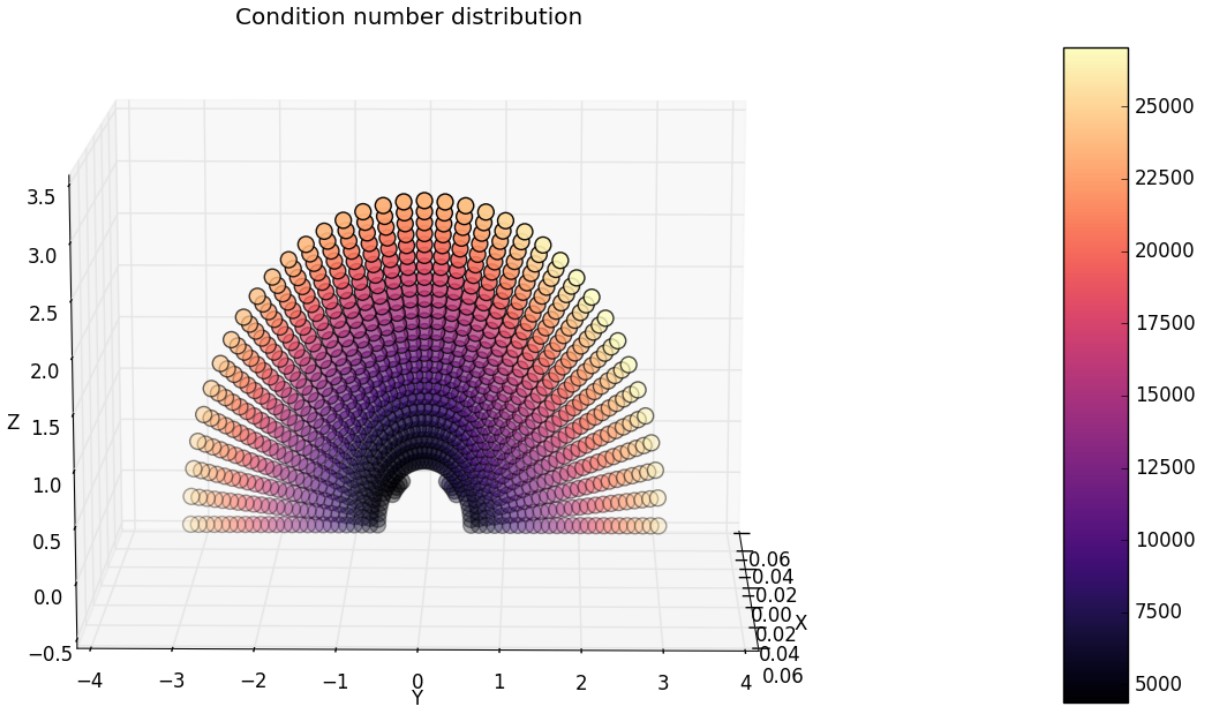


Figure 8.17: Condition number distribution(Interaction of `angle` and `height`), object points and image points are not normalized. The camera distribution is on YZ plane. The `angle` $\theta = [0°,180°]$, the step of angle is $5°$, but in real world the angle can not be $0°$ because of the constrain of DLT(three points that must be not collinear). The `height`(distance between camera and marker) as radius $r = [0.1m,3.0m]$, the step of radius is $0.1m$.

We can find that in figure 8.16(a) the condition number increases while the distance between camera and marker increases. This trend of condition number is consistent with previous results based on experiments in [1] and [13] and also is one of our desired result.
However, in figure 8.16(b) it shows the condition number distribution is asymmetric. This is because that according to the equation mentioned in [15], we know that:

$$c(A) = \|A\|_2 \left\|A^{-1}\right\|_2 = \frac{s_{max}}{s_{min}} = \frac{s_1}{s_8}$$

and A is composed of four object points and four image points in our case, we know that the object points are always set as constant(the coordinates in world coordinate system are fixed), so the only affected factor is related to the four image points. So we can see what is happened by image points:
Different camera position(Symmetrical cam position) on YZ plane has different rotation matrix R. And according to the corresponding rotation matrix R we can get different image points(camera model).
The symmetrical camera positions, which are closer to Z-axis ($\theta = 90$) and Y-axis ($\theta = 0$), get almost similar image points, similar image points lead to similar condition number. And the condition numbers on these symmetrical cam positions have smaller difference. At somewhere θ between $0-90$, we get the biggest difference of condition number on symmetrical cam positions.

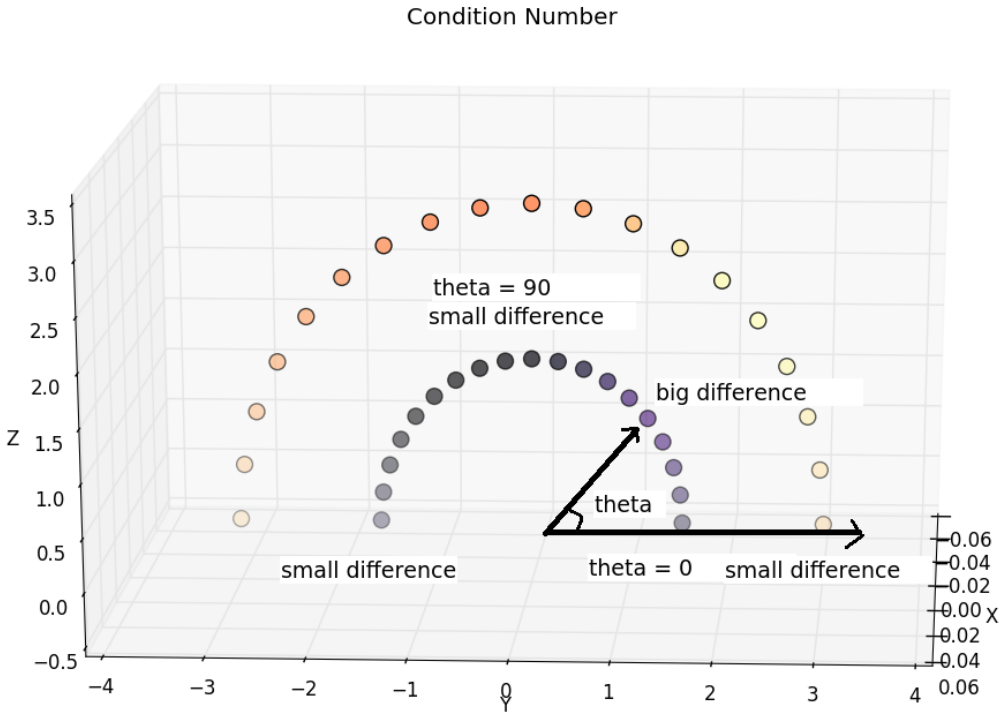
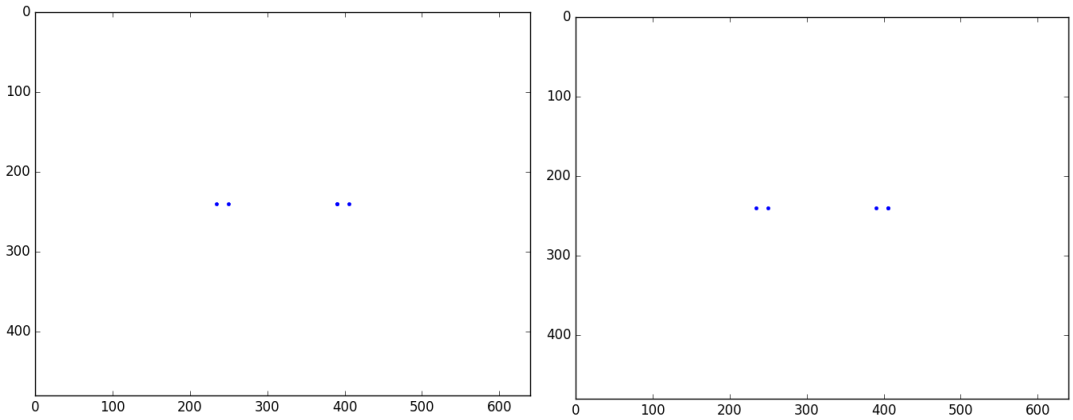
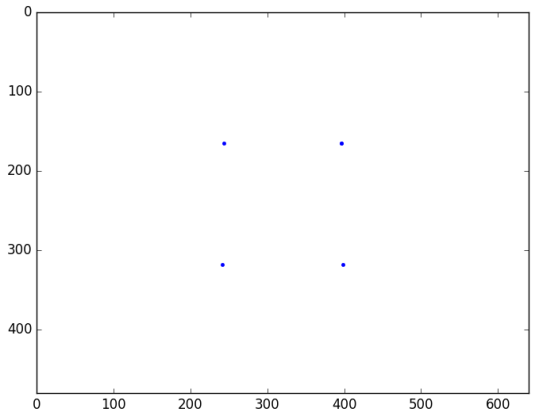


Figure 8.18: Symmetrical camera positions with different condition number

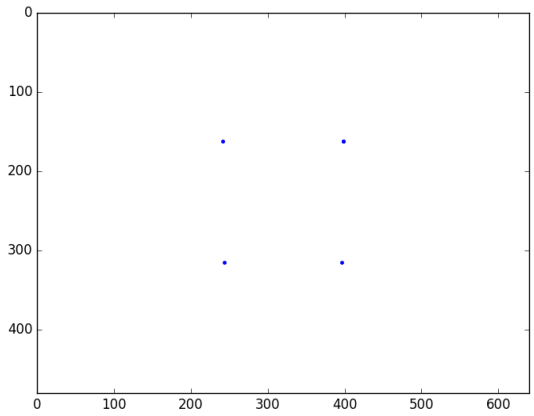


(a) theta = 0°

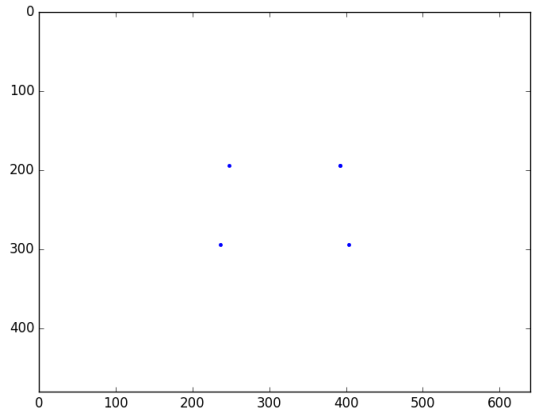
(b) theta = 180°



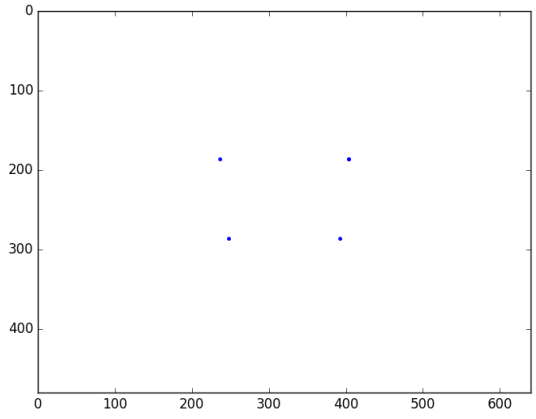
(c) theta ≈ 90°,$e.g. theta = 89°$



(d) theta ≈ 90°,$e.g. theta = 91°$



(e) theta ≈ 45°



(f) theta ≈ 135°

Figure 8.19: When theta is equal to 0° or 180°, image points exactly same. When theta closes to 90, image points exactly very similar. When theta closes to 45° or 135°, image points have relative larger differences

In addition to asymmetric problem, there is another problem:
The condition number is biggest when the angle of camera position is close to 90°, but this trend of condition number based on the change of angle is not a ideal and desired result. The reason for this problem is that without normalization of homography the differences of object points' order and image points' order are particularly large[8]. Therefore, we made regular normalization for both object points and image points in next step.

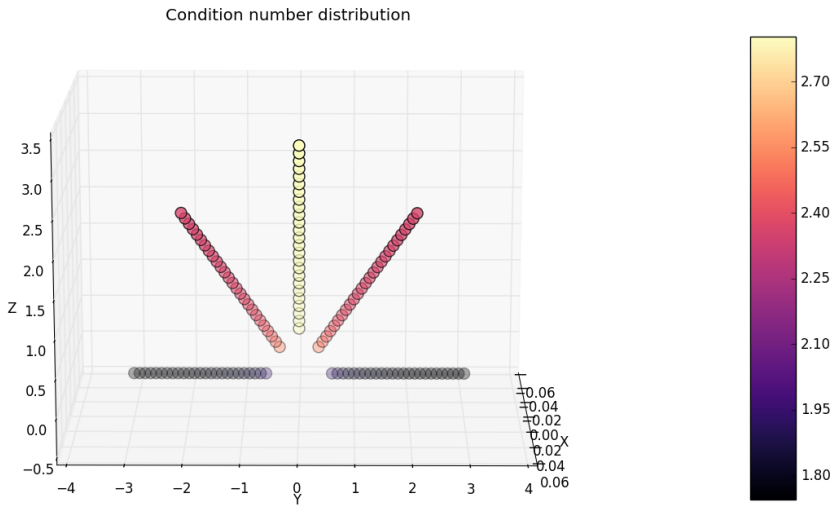
`Condition number distribution with normalized homography(regular normalization):`

We applied regular normalization for both object points and image points and made simulations under the same conditions as before.

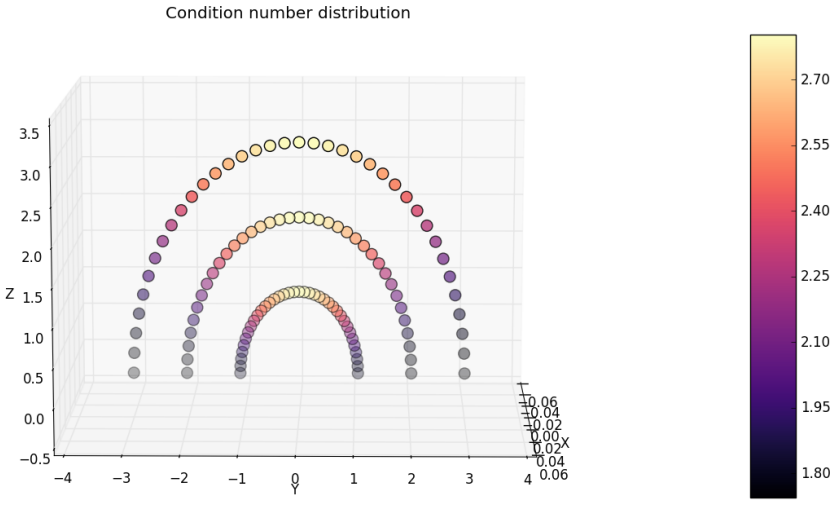
```
dist =[]
pts =pts/pts[2, :]
for i in finiteind:
    c =np.mean(pts[0:2, i].T, axis=0).T
    newp1 =pts[0, i] -c[0]
    newp2 =pts[1, i] -c[1]
    dist.append(np.sqrt(newp1 **2 +newp2 **2))

dist =np.array(dist)
meandist =np.mean(dist)
scale =np.sqrt(2) /meandist # Translation and scaling
T =np.array([[scale, 0, -scale *c[0]], [0, scale, -scale *c[1]], [0, 0, 1]])
newpts =np.dot(T, pts)
return newpts,T
```

Figure 8.20 shows separately the condition number distribution in terms of `height` and `angle`, figure 8.21 shows the condition number distribution at all positions.



(a) Influence of the height(distance between the cam1era and the marker) on the condition number of different camera positions while the angle is fixed as 0°,45°,90°,135° and 180°



(b) Influence of the angle on the condition number of different camera positions while the height(distance between camera and marker) is fixed as 1.06666667m, 2.0333333 and 3m

Figure 8.20: Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15)

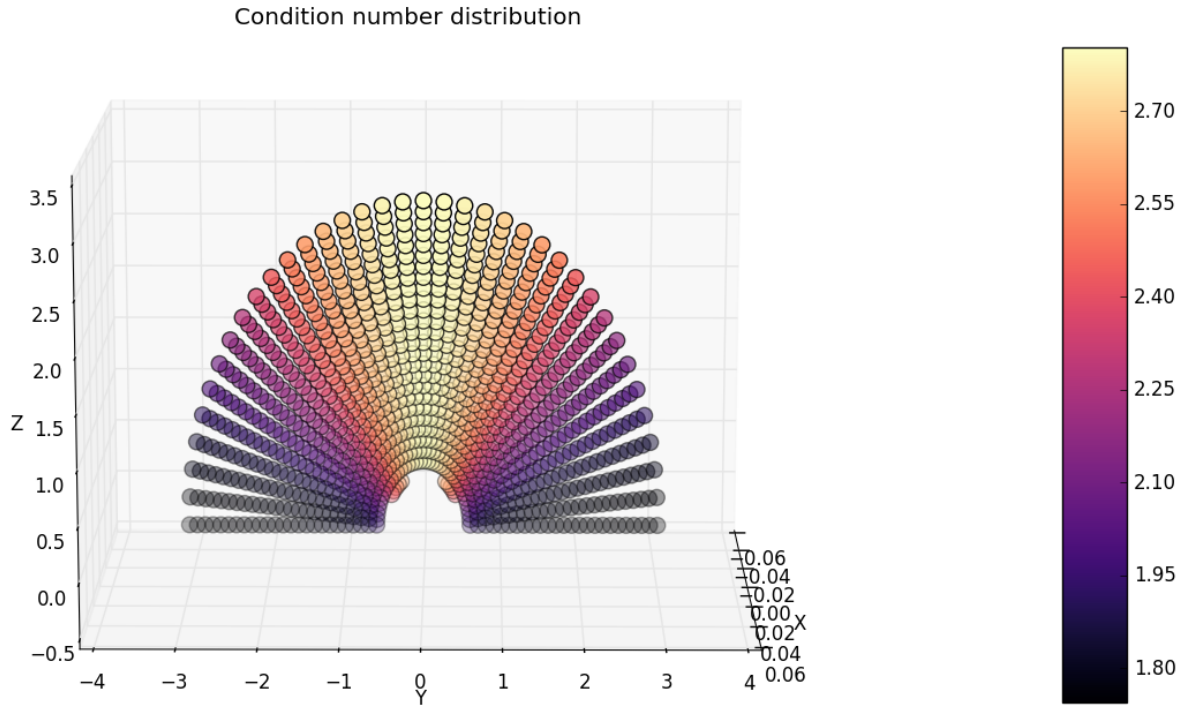


Figure 8.21: Condition number distribution, object points and image points are both regular normalized

From figure 8.20(b) we can notice that this time the trend of condition number based on the change of the `angle` is our desired result, it is consistent with one of our desired results. However this time for each fixed angle shown in figure 8.20(a) when the `height`(distance between camera and marker) increases, the condition number is still equal($angle = 0°$) or changed slightly($angle \neq 0°$). This is not a correct result.
The reason for this problem is that e.g. when the camera is set right above the marker($angle = 0°$) and moves directly above the marker, no matter where the camera is, the four image points in pixel coordinate system after normalization always locate at same positions. This leads to the equal condition number when the camera moves directly above the marker. This process is shown in figure 8.22 and figure 8.23. Similarly, when the $angle \neq 0°$, the four image points in pixel coordinate system after normalization only change slightly, this leads to similar condition number.

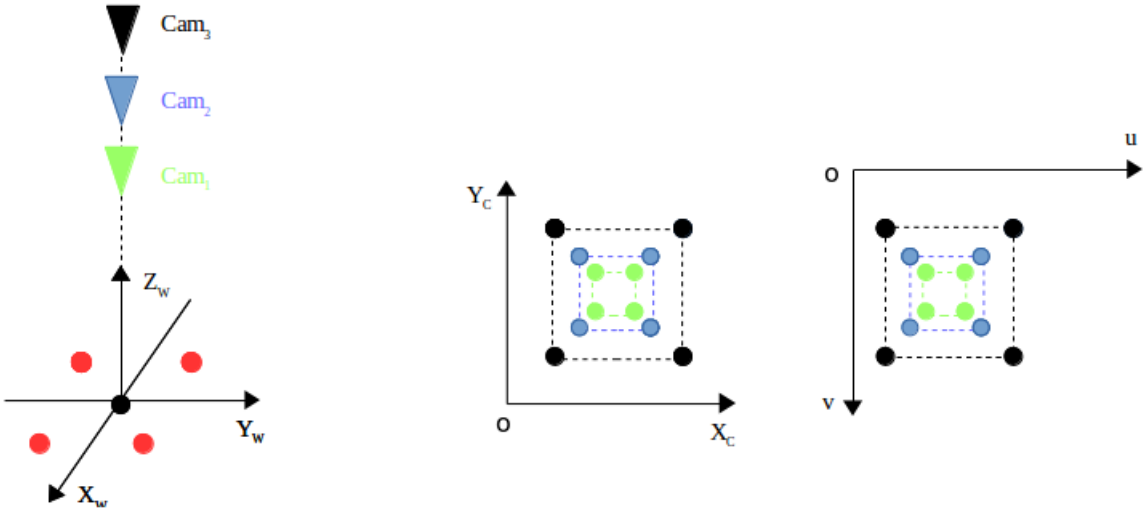


Figure 8.22: Test

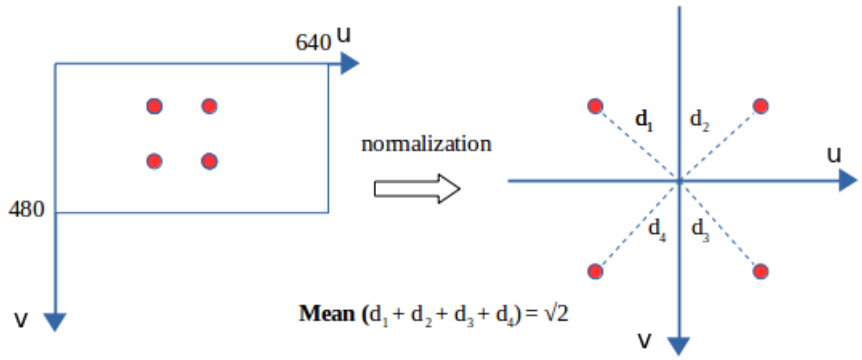


Figure 8.23: Normalization

`Condition number distribution with normalized homography(only translation but no scaling):` We already know that we also need to consider the influence of the `height` on condition number. So we got one assumption: by normalization if we just implement the translation of the image points but not implement the scaling, maybe we we can get one ideal result. In order to verify this assumption we set the $scale = 1$ this times which means no scaling after translation of image points:

```
dist =[]
pts =pts/pts[2, :]
for i in finiteind:
    c =np.mean(pts[0:2, i].T, axis=0).T
    newp1 =pts[0, i] -c[0]
    newp2 =pts[1, i] -c[1]
    dist.append(np.sqrt(newp1 **2 +newp2 **2))

dist =np.array(dist)
meandist =np.mean(dist)
scale =1 # Only translation, no scaling
T =np.array([[scale, 0, -scale *c[0]], [0, scale, -scale *c[1]], [0, 0, 1]])
newpts =np.dot(T, pts)
return newpts,T
```

But obviously we got the result still with problem which is shown in figure 8.24. This figure shows that when the camera is closer to the marker, the condition number gets bigger. It is exactly the opposite of what we want.
The reason for this problem is that if the camera locates closer to the marker, the closer the four image points are to the origin of pixel coordinate. And according to the matrix `A` of condition number, if the image points locate closer to the origin of pixel coordinate, the smaller the condition number is. Therefore in this case we got a opposite conclusion.

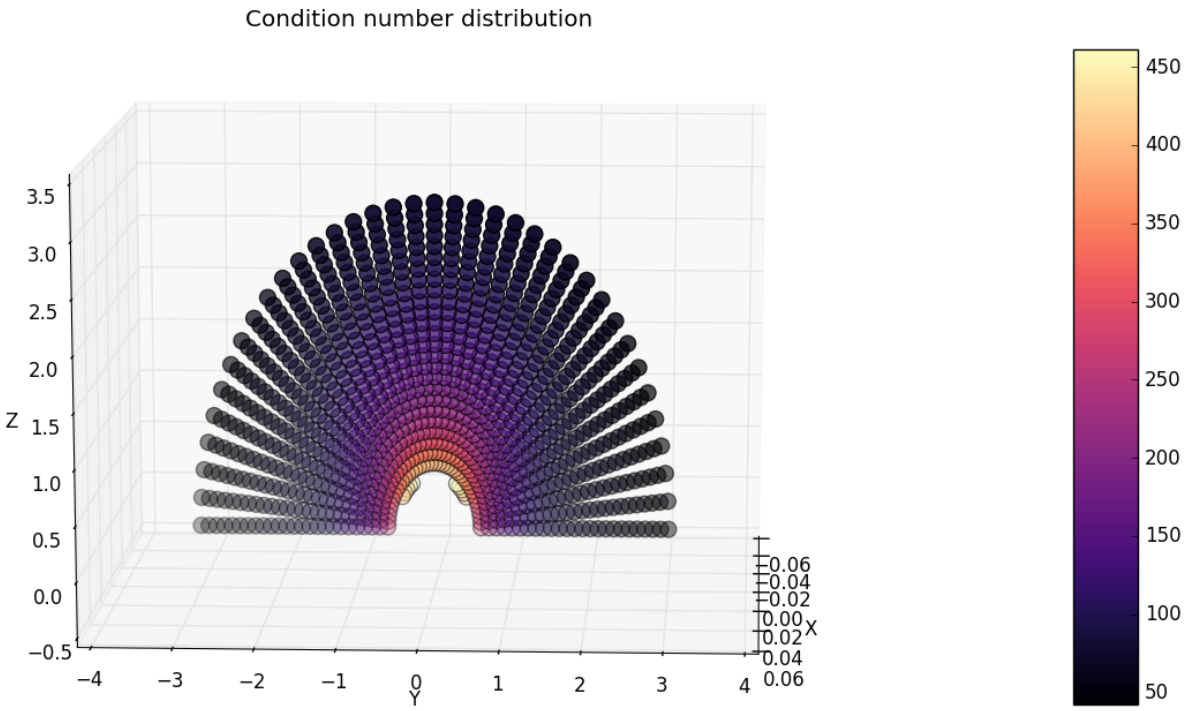
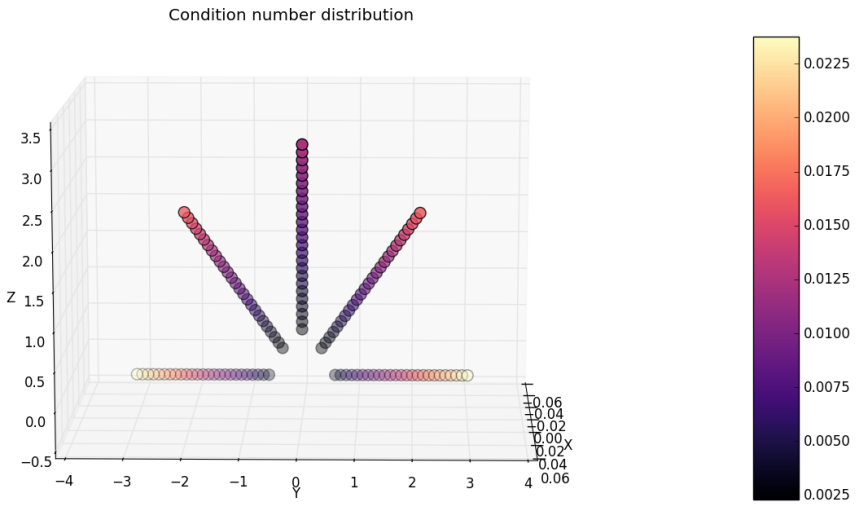


Figure 8.24: Condition number distribution, object points are regular normalized and image points are normalized only with translation but no scaling(scale = 1)

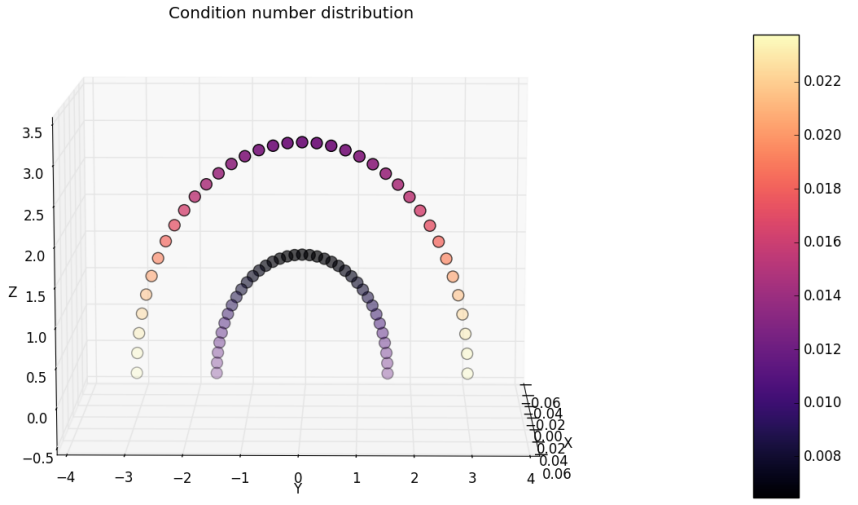
If in this case we got a opposite conclusion, can we just make a little trick that we set condition number as it's reciprocal?

```
mat_cond =gd.matrix_condition_number_autograd(*input_list, normalize =normalized)
# TODO normalized with no scaling (scale = 1) , invert the condtion number
mat_cond =1 /mat_cond
```

After performing this little trick, we found that the trend of condition number in terms of the change of `angle` is still not right, which is shown in figure 8.25(b). The reason is that before we applied the reciprocal of condition number, when the height is fixed, it has a lower condition number if the angle is smaller. Therefore when we applied the reciprocal of condition number, the lower angle has a bigger condition number.



(a) Condition number distribution when the angle is fixed as 0°,45°,90°,135° and 180°($conditionnumber = 1/conditionnumber$



(b) Condition number distribution when the height fixed at 1.55m and 3m($conditionnumber = 1/conditionnumber$

Figure 8.25: Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15)

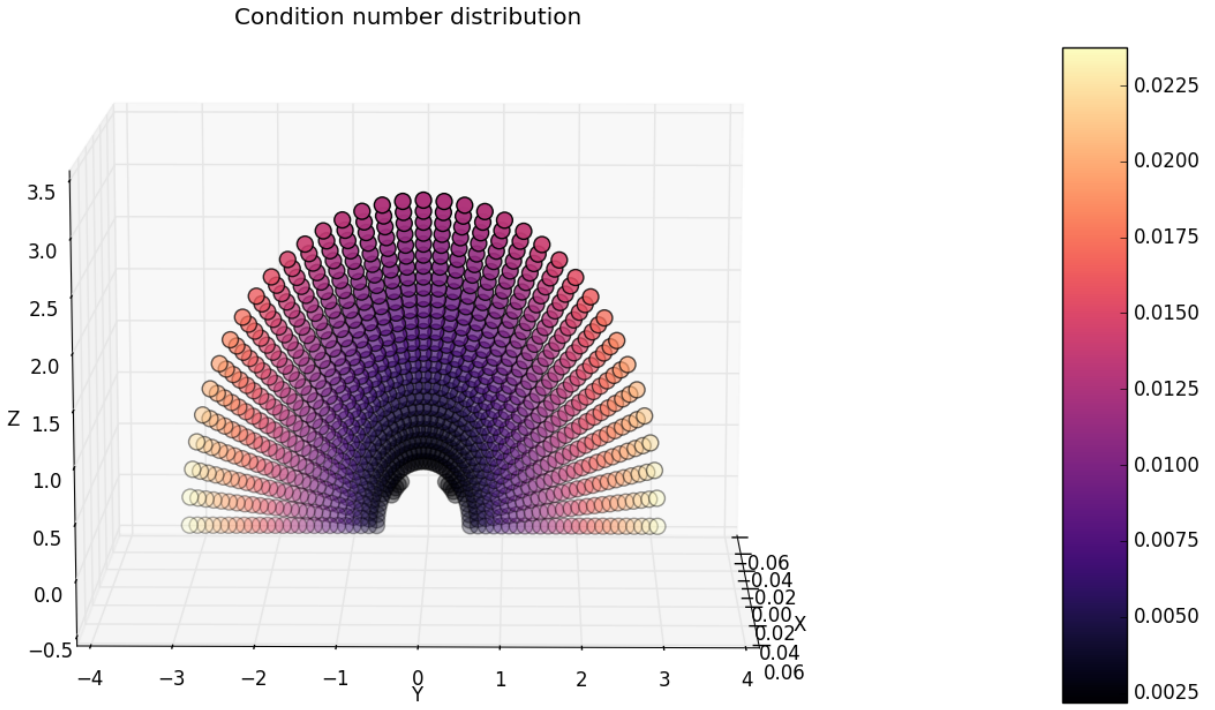


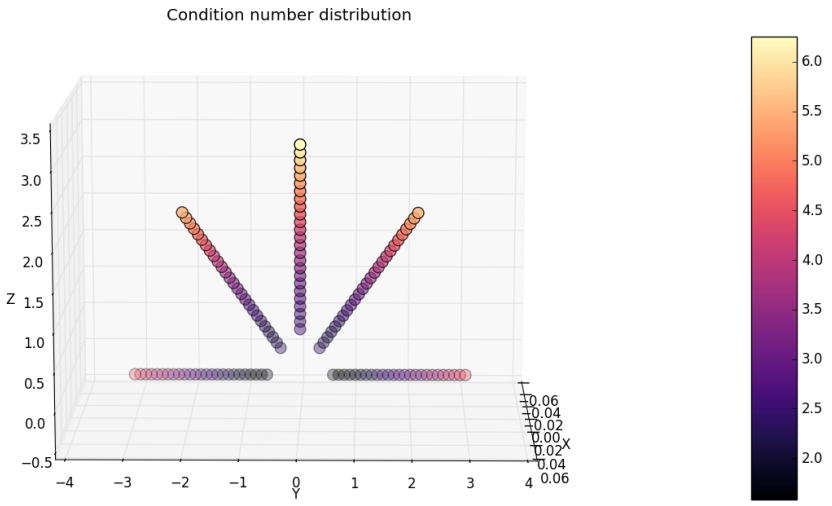
Figure 8.26: Condition number distribution, object points are regular normalized and image points are normalized only with translation but no scaling(scale = 1), but $conditionnumber = 1/conditionnumber$

Condition number distribution with normalized homography(scale based on height):
After all of these simulations, we used another trick: we set the product of height and regular normalized scale as the new scale of normalization:

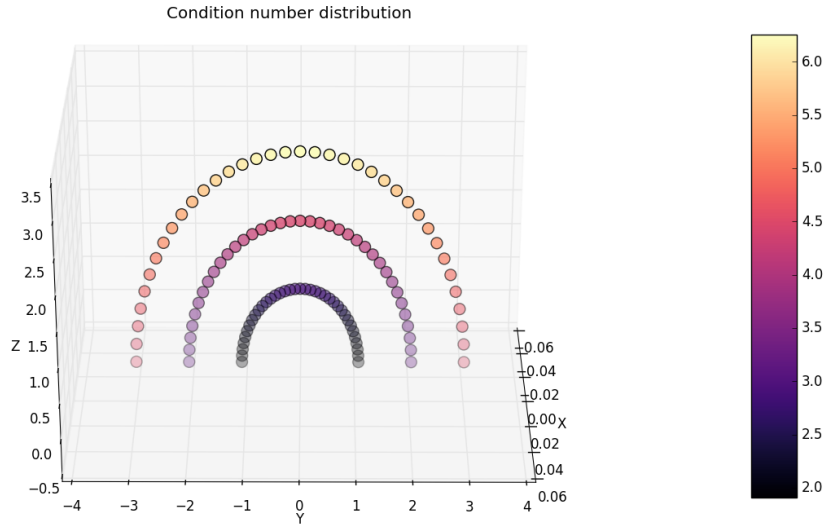
```
dist =[]
pts =pts/pts[2, :]
for i in finiteind:
    c =np.mean(pts[0:2, i].T, axis=0).T
    newp1 =pts[0, i] -c[0]
    newp2 =pts[1, i] -c[1]
    dist.append(np.sqrt(newp1 **2 +newp2 **2))

dist =np.array(dist)
meandist =np.mean(dist)
scale =radius*np.sqrt(2)/meandist # Set the scale increasing with distance from camera to marker
T =np.array([[scale, 0, -scale *c[0]], [0, scale, -scale *c[1]], [0, 0, 1]])
newpts =np.dot(T, pts)
return newpts,T
```

And we got following desired results which are shown in figure 8.27, this condition number distribution computed with this approach can be interpreted as the accuracy function.



(a) Condition number distribution, object points are regular normalized and image points are normalized with $scale = radius * np.sqrt(2)/meandist$. The angle is fixed as 0°,45°,90°,135° and 180°



(b) Condition number distribution, object points are regular normalized and image points are normalized with $scale = radius * np.sqrt(2)/meandist$. The height is fixed as 1.06666667m, 2.0333333 and 3m

Figure 8.27: Condition number distribution of different camera positions, each circle represents each camera position, different colors represent different values(figure 8.15)

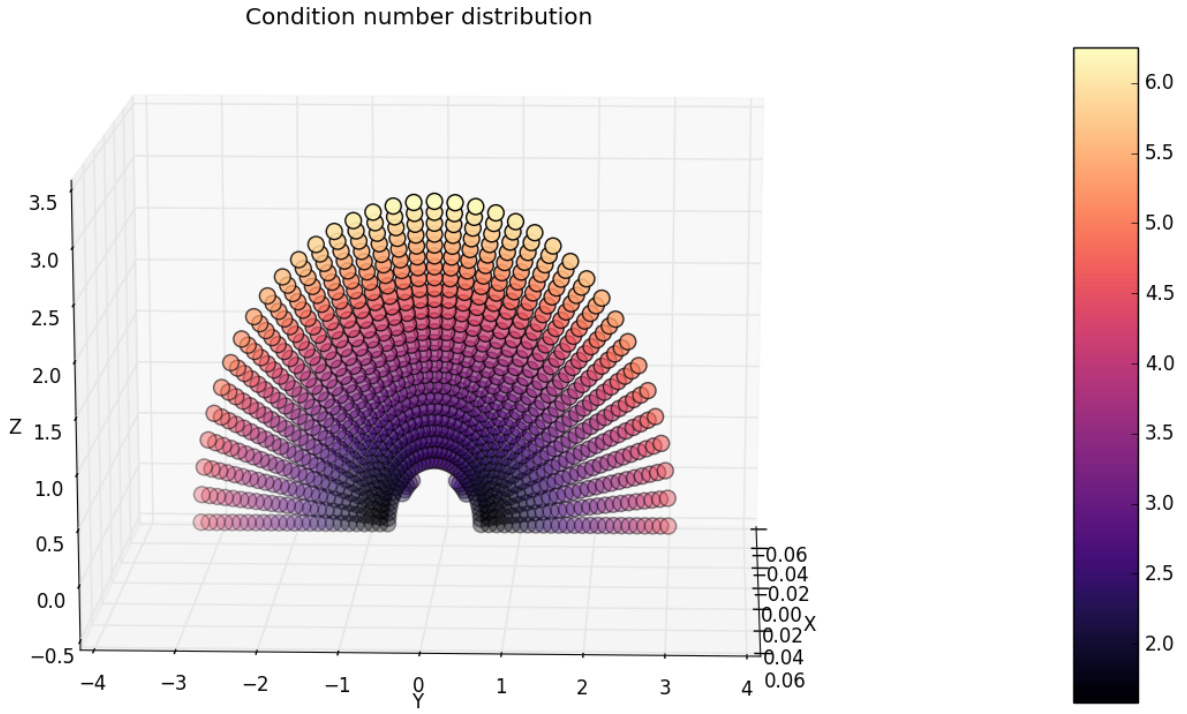


Figure 8.28: Condition number distribution, object points are regular normalized and image points are normalized with $scale = radius * np.sqrt(2)/meandist$

The figure 8.27(a) shows the trend of condition number based on the change of `h`. We can find that when the camera is close to the origin of marker, it has a relative small condition number at this corresponding position. This is consistent with one of our desired results.
The figure 8.27(b) shows the trend of condition number based on the change of `angle`. We can find that when the camera is close to the x-axis, it has a relative small condition number at this corresponding position. And the condition number distribution is exactly symmetrical. This is also consistent with another one of our desired results.

`Results summary:`
There is no doubt that normalization of homopraphy is an essential step to compute condition number of different camera positions for the accuracy function. However how to implement the meaningful normalization(e.g. how should the coordinates should be scaled?) is one step worth to think about.
After many simulations and considering different cases, we used one little trick to compute our desired condition number: we chose to scale the coordinates: the average distance of a point x from the origin is equal to $radius \times \sqrt{2}$.
Using this little trick we got our desired results mentioned at the beginning of this section: The closer the camera is to the marker, the smaller the condition number is which means the more accurate the camera pose. However in the real case in view of the camera's detection range the camera can not be placed too close to the marker and the angle between the camera and the marker plane can not be too small.

We can treat the condition number distribution as our accuracy function. Our results about condition number distribution were similar to the accuracy as a function of camera distance and relative camera angle in [1] and [13].

9 Accuracy of Monocular Tracking using Error Covariance Matrix

In [3] an approach using error covariance matrix to represent the accuracy of monocular tracking is introduced. However this approach does not apply to our situation and I regard the method mentioned in this paper with suspicion. In this section I will explain in detail why this method does not work for our situation.

9.1 Gaussian error distributions

9.1.1 Multi-dimensional Gaussian distributions

Gaussian distribution is a very common continuous probability distribution. In our project we also set the image noisy(errors) as Gaussian distribution. We directly focus on the n-dimensional Gaussian distribution which is shown in figure 9.1, the form of corresponding probability density function is:

$$f_{\mu,\Sigma}(x) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} exp^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)} \tag{9.1}$$

where $\mu \in \mathbb{R}^n$ denotes the mean of random variable vector and $\Sigma \in \mathbb{R}^{n\times n}$ covariance matrix.

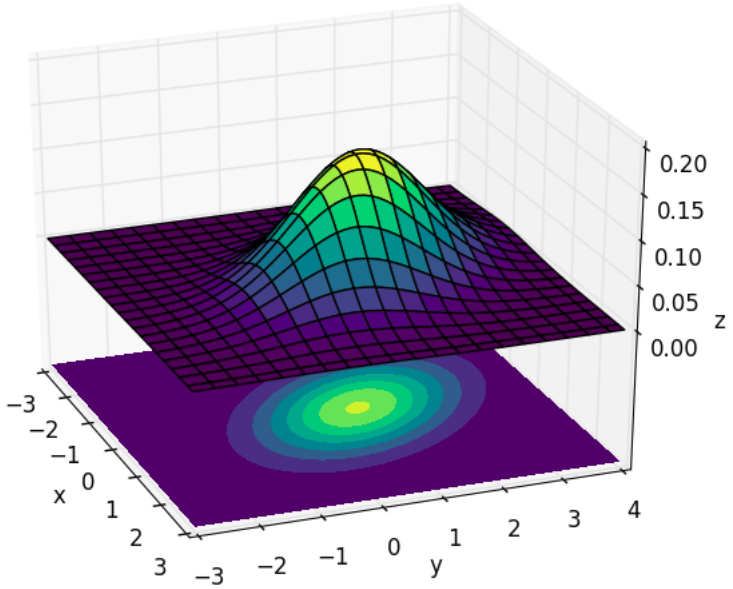


Figure 9.1: Probability density function of 2 dimensional Gaussian distribution

A n-dimensional covariance matrix $\mathbf{\Sigma}$ is always symmetric and positive semi-definite, which means the eigenvalues of covariance matrix are non-negative.

$$\Sigma = \begin{bmatrix} var(x1) & cov(x_1,x_2) & \cdots & cov(x_1,x_n) \\ cov(x_1,x_2) & var(x2) & \cdots & cov(x_n,x_2) \\ \vdots & \vdots & \ddots & \vdots \\ cov(x_1,x_n) & cov(x_2,x_n) & \cdots & var(x_n) \end{bmatrix}$$

9.1.2 Confidence ellipse/ellipsoid

The confidence ellipse which is also known as error ellipse represents an isocontour of the two dimensional Gaussian distribution and allow you to visualize the selected confidence level. In contrast to confidence ellipse, the confidence ellipsoid represents a visualization of three dimensional Gaussian distribution.
Now we make a derivation from the probability function(equation 9.1) and assume that the mean μ of random variable is zero:

$$p = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} exp^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$
$$\Downarrow$$
$$p\sqrt{(2\pi)^n|\Sigma|} = exp^{\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)}$$
$$\Downarrow$$
$$(x-\mu)^T\Sigma^{-1}(x-\mu) = -2\ln p\sqrt{(2\pi)^n|\Sigma|}$$
$$\Downarrow$$
$$x^T\Sigma^{-1}x = -2\ln p\sqrt{(2\pi)^n|\Sigma|} = s$$

In general, we can use following equations to describe ellipse and ellipsoid:

$(\frac{x}{a})^2 + (\frac{y}{b})^2 = 1$ a,b: length of semi-major axis and length of semi-minor axis

$(\frac{x}{a})^2 + (\frac{y}{b})^2 + (\frac{z}{c})^2 = 1$ a,b,c: semi-axes

From covariance matrix we can derive the forms similar to the above equations:

$$\frac{x^2}{\lambda_x} + \frac{y^2}{\lambda_y} = s$$
$$\frac{x^2}{\lambda_x} + \frac{y^2}{\lambda_y} + \frac{z^2}{\lambda_z} = s$$

where λ_x, λ_y and λ_z are the eigenvalues of the corresponding covariance matrix, s defines the scale of the ellipse which could be any arbitrary number. Then we turn the question into how to find s, in our case we use confidence level to represent the scale s. The relationship between confidence level and scale s for different dimensions is shown in table 9.1.
Now we make a transform of ellipse and ellipsoid, multiply $\frac{1}{s}$ on both sides of equations:

$$(\frac{x}{\sqrt{s\lambda_x}})^2 + (\frac{y}{\sqrt{s\lambda_y}})^2 = 1$$
$$(\frac{x}{\sqrt{s\lambda_x}})^2 + (\frac{y}{\sqrt{s\lambda_y}})^2 + (\frac{z}{\sqrt{s\lambda_z}})^2 = 1$$

we find that the length of the axes of ellipse or ellipsoid can be represented as $l_i = \sqrt{s\lambda_i}(i = 1,2,3)$. We can use this formula to calculate the length of axes and then plot the covariance ellipse or ellipsoid for a certain confidence level as an example shown in figure 9.2 and figure 9.3.

Confidence Level						
Dimension	25%	50%	75%	95%	97%	99%
n = 1	0.10153	0.45494	1.3233	3.84146	4.70929	6.6349
n = 2	0.57536	1.38629	2.77259	5.99146	7.01312	9.2103
n = 3	1.21253	2.36597	4.10834	7.81473	8.94729	11.3449

Table 9.1: The value of scale s in different confidence levels for different dimensions. e.g. if n = 2 and confidence level is 95% then the scale s is 5.99146[3]

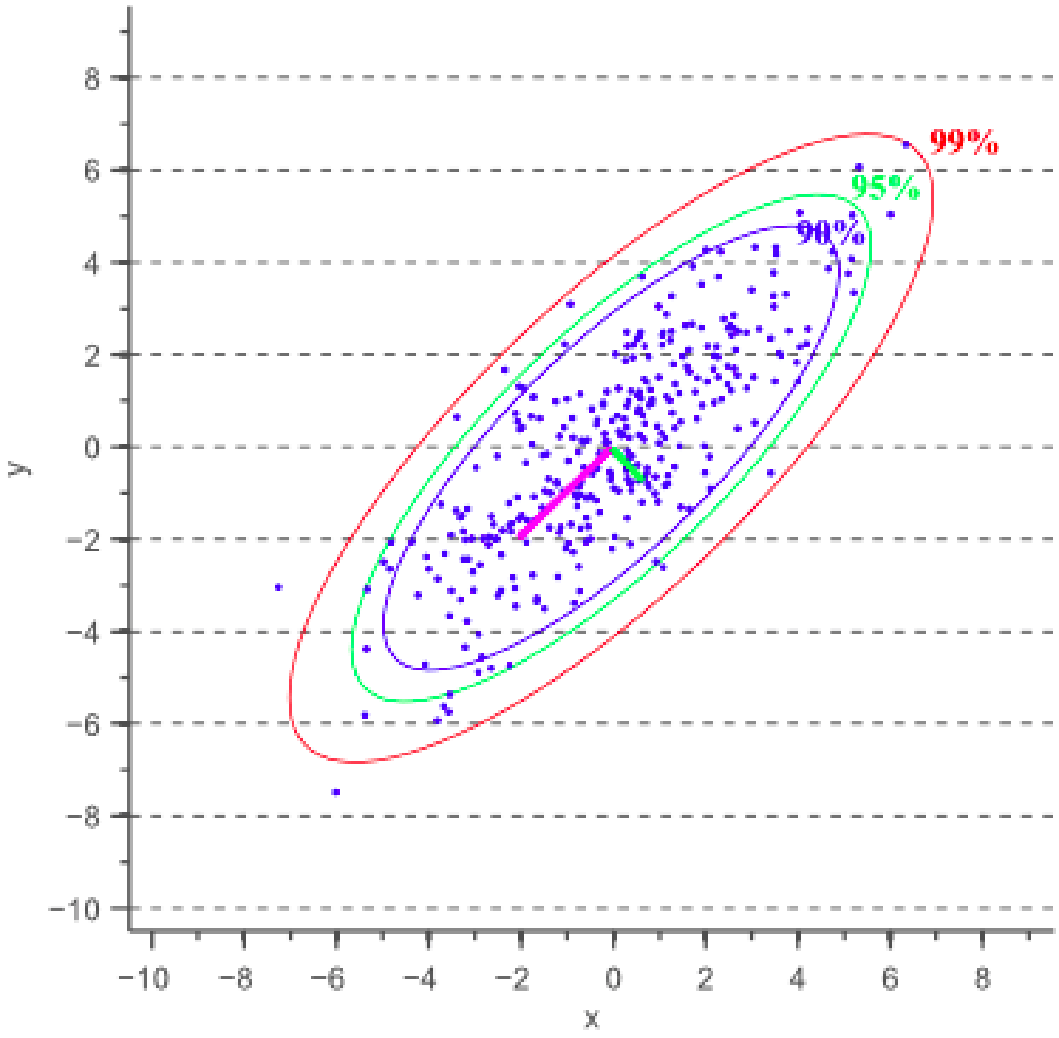


Figure 9.2: Confidence ellipses for normally distributed data with different confidence levels[17]

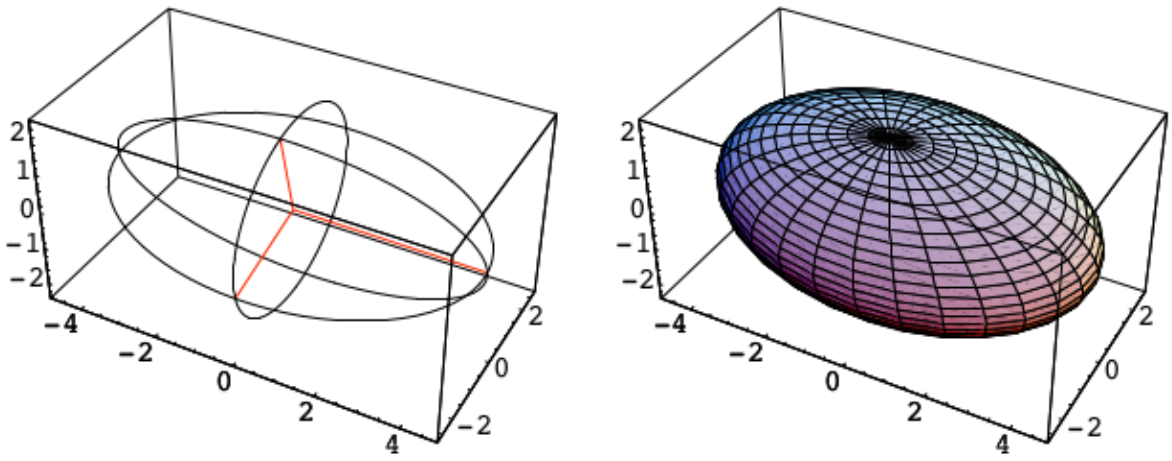


Figure 9.3: Visualization of covariance ellipsoid for a certain confidence level[3]

9.2 Error propagation for Gaussian distributions

Two propagation methods are introduced in ...which are used to describe Gaussian errors propagation in linear or non-linear functions of monocular tracking system.

9.3 Forward propagation

For the linear function, if the random variable v is a Gaussian distribution then the linear function $f(v)$ is also a Gaussian distribution. We assume that the linear function is describe as $f(v) = Av$, the mean of variable v is $\overline{v}$ and the covariance matrix is Σ. Then the linear function $f(v)$ is also a random variable with mean $f(\overline{v})$ and covariance matrix Σ_f. This relationship is shown in table 9.2:

	mean	covariance matrix
v	$\overline{v}$	Σ
$f(v)$	$f(\overline{v})$	$\Sigma_f = A\Sigma A^T$

Table 9.2: Linear function: $f(v) = Av$

For the non-linear function $f(v)$, in order to apply the Gaussian distribution forward propagation we need to build an approximation of the non-linear function. We can use Taylor expansion to present the first order approximation:

$$f(v) = f(v_0 + \Delta_v) = f(v_0) + J_f(v_0)\Delta_v + \mathcal{O}v^2$$

For the non-linear function $f(v)$ we assume that the mean of variable v is $\overline{v}$ and the covariance matrix is Σ. Then the non-linear function $f(v)$ is a random variable with mean $f(\overline{v})$ and covariance matrix Σ_f. This relationship is shown in table 9.3:

	mean	covariance matrix
v	$\overline{v}$	Σ
$f(v)$	$f(\overline{v})$	$\Sigma_f = J_f \Sigma J_f^T$

Table 9.3: Non-linear function: $f(v)$

where J_f is the Jacobian matrix of non-linear function $f(v)$ evaluated at $\overline{v}$.

9.4 Backward propagation

In some situation if the covariance matrix Σ_f of function $f(v)$ is given and we want to know the covariance Σ of the variable v. We can use the following equation to compute the backward propagation:

$$\Sigma = (J_f^T \Sigma_f^{-1} J_f)^{-1}$$

where J_f is the Jacobian matrix of non-linear function $f(v)$ evaluated at $\overline{v}$. This backward propagation is the main idea to compute the accuracy of monocular tracing in [3]

9.5 Accuracy of monocular tracking

In [3] the computation of accuracy of monocular tracking is under a theoretical model, which the topology of the target is known exactly and all the features from the target can be well recognized and that the 2D position estimation is unbiased but not noise free. In [3] a spherical coordinate system is used to represent the pose of camera in world.

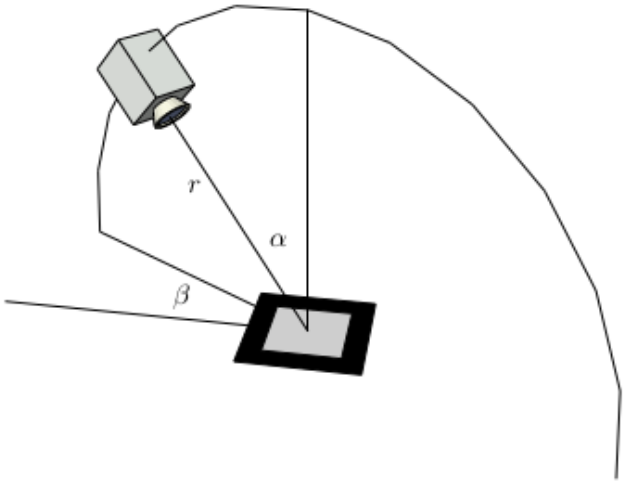


Figure 9.4: Setup for analyzing the theoretical accuracy of a monocular tracking system with planar fiducials[3]

In this case the camera pose can be described with 3 parameters(α,β,r) instead of six-dimensional pose(x,y,z,α,β,r), so a covariance matrix $\Sigma_{\alpha,\beta,r} \in \mathbb{R}^{n\times n}$ can represent the error for each camera position. Recall that the pinhole camera model, the relationship between world coordinate system and pixel coordinate system.

$$Z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KT \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \tag{9.2}$$

where camera intrinsic parameters K is constant and in our situation we set it as:

$$K = \begin{bmatrix} 800 & 0 & 320 \\ 0 & 800 & 240 \\ 0 & 0 & 1 \end{bmatrix}$$

According to [3] without loss of generality the camera extrinsic parameters T([R|t]) can be parametrized with α,β,γ and the following equations show how to get the rotation matrix R based on coordinate transformation, this process is shown in figure 9.5.

$$\begin{aligned} {}^{N}R^{B} &= {}^{N}R^{A} \cdot {}^{A}R^{B} \\ &= \begin{bmatrix} cos(\alpha)cos(\beta) & -sin(\beta) & sin(\alpha)cos(\beta) \\ cos(\alpha)sin(\beta) & cos(\beta) & sin(\alpha)sin(\beta) \\ sin(\alpha) & 0 & cos(\alpha) \end{bmatrix} \end{aligned}$$

Because the positive direction of the camera z-axis always points to the origin where the marker is placed in the world coordinate system, the rotation matrix ${}^{N}R^{B}$ still needs to rotate 180° around x-axis:

$$R = R_x(\pi) \cdot {}^{N}R^{B}$$

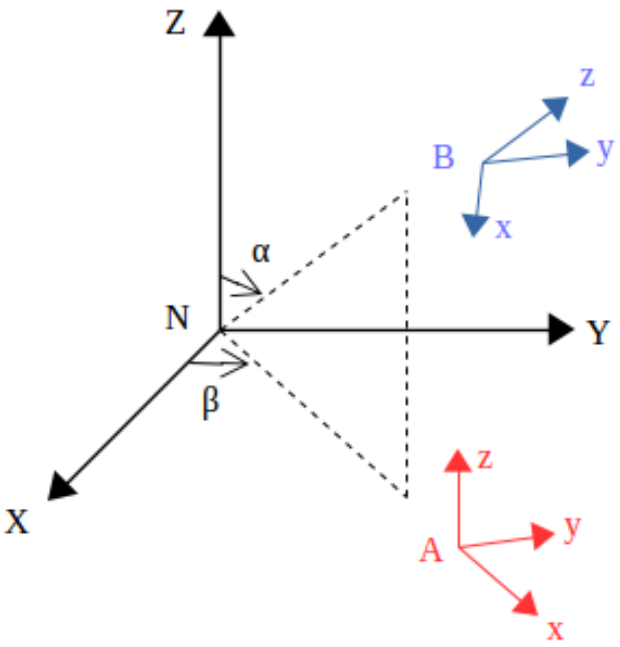


Figure 9.5: Rotation in spherical coordinate system

And from Cartesian coordinates to spherical coordinates transformation we know that the translation vector t can be presented using there parameters α,β,γ as:

$$x = r \cdot cos(\beta) \cdot sin(\alpha)$$
$$y = r \cdot sin(\beta) \cdot sin(\alpha)$$
$$z = r \cdot cos(\alpha)$$

Now the camera extrinsic parameters T([R|t]) can be presented as:

$$T = \begin{bmatrix} cos(\alpha)cos(\beta) & -sin(\beta) & sin(\alpha)cos(\beta) & rcos(\beta)sin(\alpha) \\ cos(\alpha)sin(\beta) & cos(\beta) & sin(\alpha)sin(\beta) & rsin(\beta)sin(\alpha) \\ sin(\alpha) & 0 & cos(\alpha) & rcos(\alpha) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because we already know the expression of the camera extrinsic parameters T with α,β,γ and the camera intrinsic parameters K is constant, now we can split the equation 9.2 into two parts:

$$f(\alpha,\beta,r) : \begin{cases} u := A_1\alpha + B_1\beta + C_1 r \\ v := A_2\alpha + B_2\beta + C_2 r \end{cases} \tag{9.3}$$

where A_1,B_1,C_1,A_2,B_2,C_2 are the coefficients come from KT. Sequentially we can compute the Jacobian of function 9.3 evaluated at the origin:

$$J_f = \left.\frac{\partial f}{\partial(\alpha,\beta,r)}\right|_0 \in \mathbb{R}^{2n\times 3} \tag{9.4}$$

where n is the number of image points(we set $n = 4$ in our simulation).
In our case we set image noises as 2-dimensional Gaussian distribution, so the error for each image point can be detected with uncertainty given by the covariance matrices $\Sigma_{v_i} \in \mathbb{R}^{2\times 2}$. And reference the backward propagation theory the covariance matrix for detecting the planar marker can be represented as:

$$\Sigma_{\alpha,\beta,r} = \left(J_f^T \begin{bmatrix} \Sigma_{v1} & & \\ & \ddots & \\ & & \Sigma_{vn} \end{bmatrix}^{-1} J_f \right)^{-1} \in \mathbb{R}^{3\times 3}$$

From equation 9.4 the Jacobian of f is evaluated at the origin($\alpha = 0, \beta = 0, \gamma = 0$), which means J_f is always a constant matrix no matter where the image point is. If the value of $\mathbf{J_f}$ is constant, this leads to a big problem: The value of $\Sigma_{\alpha,\beta,r}$ only depends on the set of covariance matrices $\begin{bmatrix} \Sigma_{v1} & & \\ & \ddots & \\ & & \Sigma_{vn} \end{bmatrix}^{-1}$. And image noise of each point is set as 2-dimensional Gaussian distribution with the mean 0 and standard deviation 2. So the diagonal terms of the covariance matrix Σ_{v_i} of each image points with error are closed to 4(it depends on the size of samples) and the rest terms of Σ_{v_i} are closed to 0. So we get the following approximate value when n = 4:

$$\begin{bmatrix} \Sigma_{v1} & & & \\ & \Sigma_{v2} & & \\ & & \Sigma_{v3} & \\ & & & \Sigma_{v4} \end{bmatrix}^{-1} \approx \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}^{-1}$$

We can easily find that the diagonal terms of the covariance matrix Σ_{v_i} play a decisive role in backward propagation formula. Based on these almost similar Σ_{v_i} the covariance matrix $\Sigma_{\alpha,\beta,r}$ of different image points only has a extremely tiny change, the value of $\Sigma_{\alpha,\beta,r}$ can be almost regarded as constant!
According to the above explanation we can not display covariance ellipsoids exactly like figure 9.6. From this view the accuracy of monocular tracking method introduced in [3] can not be applied to our situation! Whether this method can be applied to other situations, I am still skeptical about this.

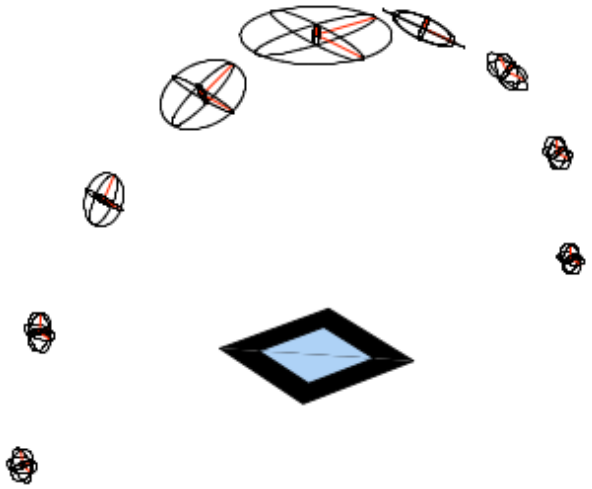


Figure 9.6: Accuracy of the pose of the camera in marker coordinates. Display with covariance ellipsoids[3]

10 Motion Planning

"Motion planning (also known as the navigation problem or the piano mover's problem) is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement. A basic motion planning problem is to produce a continuous motion that connects a start configuration `S` and a goal configuration `G`, while avoiding collision with known obstacles. The robot and obstacle geometry is described in a 2D or 3D workspace, while the motion is represented as a path in (possibly higher-dimensional) configuration space"[25].

`Configuration space`
Although the motion planning problem is applied in the real world, in computer science we need to convert the real space into another space: `configuration space`.
"A configuration describes the pose of the robot, and the configuration space `C` is the set of all possible configurations"[25].
According to the definition of configuration space we know that a robot configuration is a specification of the positions of all robot points relative to a fixed coordinate system. And in our case the robot only moves on the ground which means the workspace is a 2-dimensional plane and currently we do not consider translation and rotation of the robot, we can treat the robot as a singe point(zero-sized) translating in this 2-dimensional workspace `C`, and the configuration can be represented with two parameters (x, y).

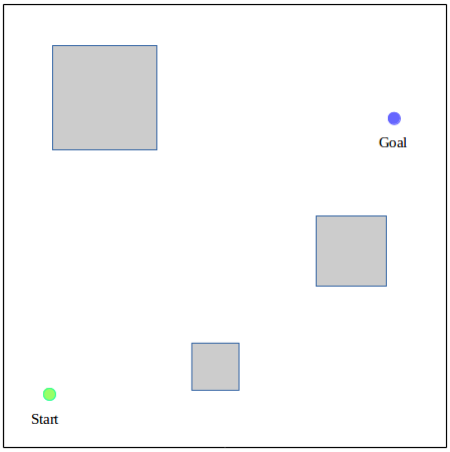


Figure 10.1: Configuration space of a point-sized robot. White = C_{free}, gray = C_{obs}, green point = Start, blue point = Goal[25].

`Grid-based search`
For path planning the continuous terrain needs to be discretized, so we need to use some methods to discretize the configuration space. Because of low-dimensional of our workplace and the grid-based algorithm is known as one of common approaches in robotics, the configuration space can be planed on a 2D occupancy grid map, which means we can overlay a grid on top of configuration space.
"Grid-based approaches overlay a grid on configuration space, and assume each configuration is identified with a grid point. At each grid point, the robot is allowed to move to adjacent grid points as long as the line between them is completely contained within C_{free} (this is tested with collision detection)"[25].

If we use grid-based approaches, a grid resolution is required to set. In addition to this, we also need to consider the width and height of the grid, e.g. if we assume that the camera's detection region is a half circle with radius 3 meters, we can set the grid as a $3m \times 6m$ square. This gird is shown in figure 10.2.

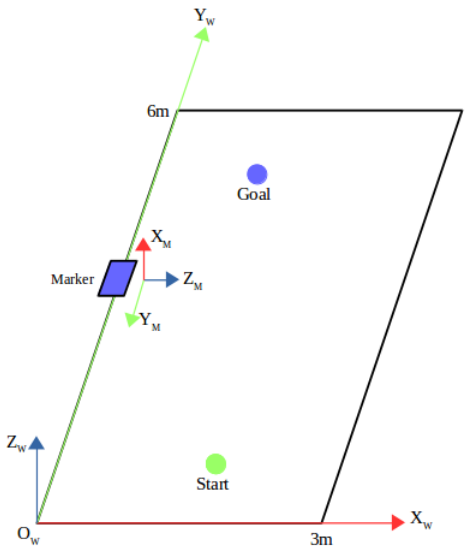


Figure 10.2: Convert configuration space into a $3m \times m$ grid. The center of the marker is at $(0,3m)$ in world coordinates

There are so many known search algorithms which can be applied in different cases, in our simulation we applied `A*` and `Artificial potential fields` as our search algorithm and we modified `A*` and `Artificial potential fields` based on the condition number distribution.

- Regular A*(without considering condition number distribution)
- Modified A*(with considering condition number distribution)
- Artificial potential fields

We compared the accuracy of the paths which are computed with these three algorithms. In following sections I will describe the details how to apply the `A*` search algorithm and `Artificial potential fields` related to our condition number distribution in our situation.

10.1 The selected pose estimation algorithm

In our simulations in order to compute the measured path, four different pose estimation algorithms were performs. And we compared the rotational error and translational error of each algorithm for our camera distribution on YZ-plane of marker coordinates. The error results are shown in figure 10.3, figure 10.4 and figure 10.5. Therefrom we found that the iterative method based on the Levenberg-Marquardt optimization denoted as `LM` has the relative smaller R-error than others, even though the t-error form these four algorithms are similar. Therefore we applied `LM` as our pose estimation algorithm for `A*` and artificial potential fields method.

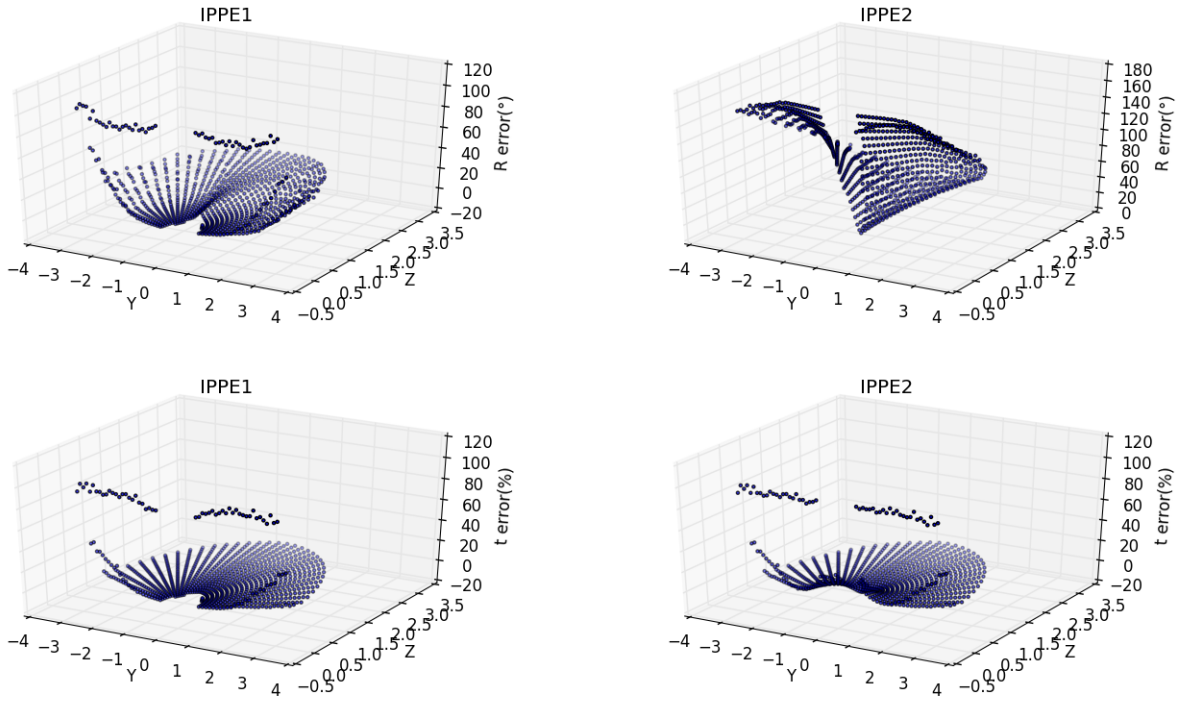


Figure 10.3: Rotational error and translational error for the selected `IPPE` method[6]

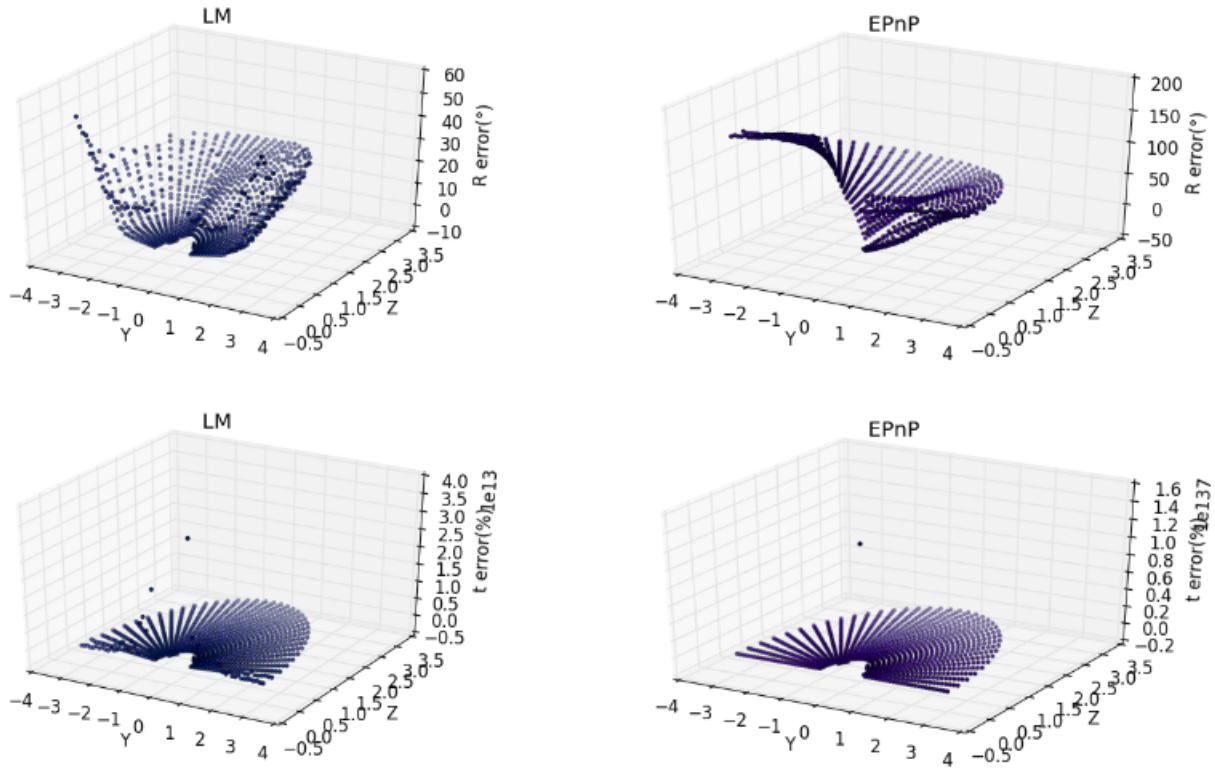


Figure 10.4: Rotational error and translational error for the selected `LM` and `EPnP` methods[11]

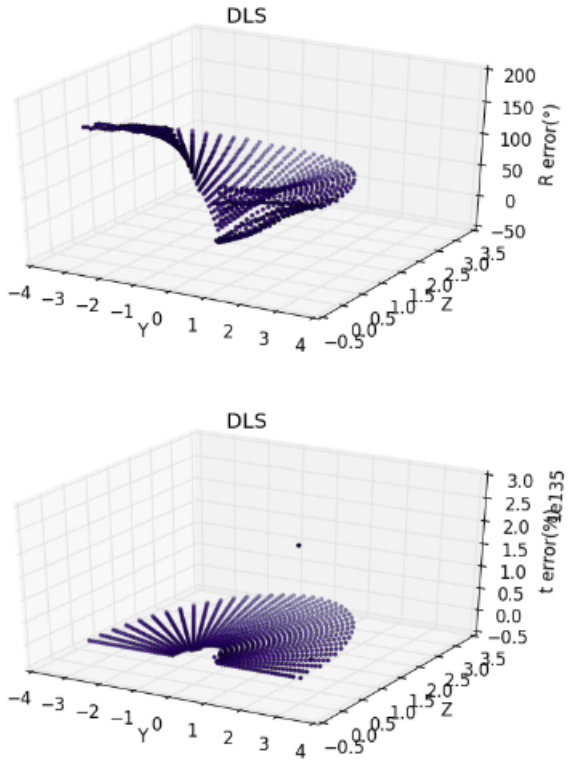


Figure 10.5: Rotational error and translational error for the selected `DLS` method

10.2 A* search algorithm

`A*` is a widely used informed search algorithm in pathfinding. In our situation we applied the regular `A*` algorithm without considering the condition number distribution to compute the least cost path, which means the actual shortest path from start node to goal node. Also we modified `A*` algorithm with considering the condition number distribution to compute the least cost path, which means the most accurate path from start node to goal node.

`Regular A*`

We know that `A*` is based on an estimate of the cost(total weight) from start node to goal node. Specifically, `A*` selects the path that minimizes:

$$\boxed{f(n) = g(n) + h(n)}$$

where n is the current node, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic that estimates the cost of the cheapest path from n to the goal, f(n) is the total cost of the current node. `A*` computes $f(n) = g(n) + h(n)$. To add two values, those two values need to be at the same scale. Because the heuristic is problem-specific[21] and on a grid, there are some well-known heuristic functions to use e.g. `manhattan distance`, `diagonal distance`, `euclidean distance` and so on. And considering that in our situation the robot can move in 8 directions on the given square grid, therefore `diagonal distance` is suitable for our situation. The following shows how to compute the `diagonal distance`:

$$\begin{aligned} dx &= abs(node.x - goal.x) \\ dy &= abs(node.y - goal.y) \\ h &= D * (dx + dy) + (D2 - 2 * D) * min(dx, dy) \end{aligned}$$

where `D` is the minimum cost of moving from one space to an adjacent space horizontally or vertically and `D2` is the cost of moving diagonally. And we set $D = 10$ and $D2 = 14$ in our simulation, we take these values because the distance along the diagonal line is $\sqrt{2}$ or about 1.414 times that it takes to move horizontally or vertically. For simplicity, we use 10 and 14 approximations avoiding to calculate the root and decimals. Using such integers is also faster for calculation of computers.

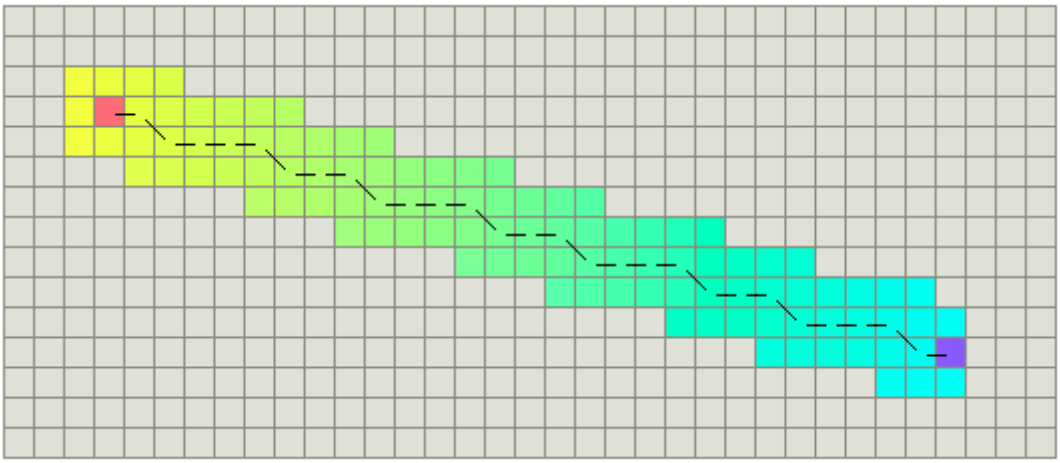


Figure 10.6: Calculated path with `A*` search algorithm on grid. The robot can move in 8 directions(`Diagonal distance` as heuristic function)[12].

The following pseudocode describes the `A*` algorithm[21]:

Algorithm 10.1: A* pseudocode

```
input: start node, goal node
output: least cost path from start to goal

Initial:
        openList = {start}
        closedList := {}
        cameFrom := {}
        g(start) := 0
        h(start) := heuristic_function(start, goal)
        f(start) := g(start) + h(start)

while openList is not empty:
        current := the node in openList with the lowest f value
        if current = goal
            return
        remove current from openList
        add current to closedList

        for each neighbor of current
            if neighbor in closedList
                continue

            if neighbor not in openList
                add neighbor to openList

            tentative_g := g(current) + dist_between(current, neighbor)
            if tentative_g >= g(neighbor)
                continue

            cameFrom(neighbor) := current
            g(neighbor) := tentative_g
            f(neighbor) := g(neighbor) + heuristic_function(neighbor, goal)
return failure
```

The key to find the shortest path(optimal solution) is how to select the heuristic function $h(n)$[12]:

- If h(n) is 0, A* turns into Dijkstra's Algorithm.
- If h(n) is always lower than(or equal to) the cost of moving from n to the goal, in this case the lower $h(n)$ is, the more nodes A* expands, low efficiency(slower). However A* is guaranteed to find a shortest path.
- If h(n) is exactly equal to the cost of moving from n to the goal, A* will find the shortest path and the search will be strictly along the shortest path. The search efficiency at this time is the highest.
- If h(n) is greater than the cost of moving from n to the goal, the number of search points is few, the search range is small, and the efficiency is high, but the optimal solution cannot be guaranteed.
- If h(n) is very high compared to g(n), A* turns into Greedy Best-First-Search.

`Modified A*`
We modified the A* search algorithm adding condition number to heuristic function:

```
condNum =ccn.getCondNum_camPoseInRealWord(x_w, y_w, grid_reso, width, height)*50
h =h_normal +condNum
```

Currently, in order to find a balance between normal heuristic function and condition number, we enlarged the condition number 50 times to get a proper weight, which means the normal heuristic function and condition number distribution have almost similar weights.
Temporarily, we did not add obstacles in grid(not considering the problem of collision avoidance).

10.3 Artificial potential fields method

"One approach is to treat the robot's configuration as a point (usually electron) in a potential field that combines attraction to the goal, and repulsion from obstacles. The resulting trajectory is output as the path.The Artificial potential fields can be achieved by direct equation similar to electrostatic potential fields or can be drive by set of linguistic rules"[25].
The artificial potential fields include the attractive fields and repulsive fields. The target point(goal) generates attractive force on the object and guides the object moving toward it. The obstacles generate repulsive force on the object and prevent the object from colliding with them. The total force of an object at each point in the path is equal to the sum of all attractive forces and repulsive forces at that point.

$$U(q) = U_{att}(q) + U_{rep}(q)$$

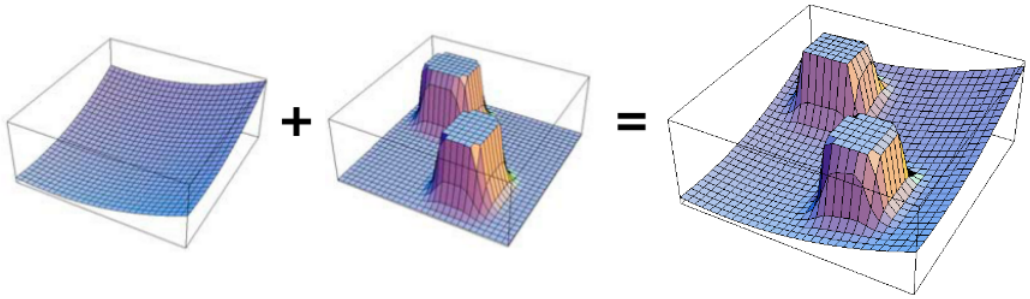


Figure 10.7: Attractive potential fields + repulsive potential fields = Total potential fields[5]

The key of artificial potential fields method is how to construct the attractive fields and the repulsive fields. In our situation we did not consider the obstacles on the gird temporarily, but we treated the

condition number distribution on the grid as repulsive fields. In our case the order of condition number is 1 to 10, therefore the order of the attractive potential should be less than 10^2. This is because if the values of the attractive potential are much larger than the condition numbers, only the attractive potential plays the role and the robot would not be affected by the repulsive potential fields(condition number distribution). The following shows our used attractive potential function and repulsive potential function:

$$U_{att}(q) = \frac{1}{2}\zeta d(q,q_{goal})$$
$$U_{rep}(q) = \text{condition number distribution on grid}$$

where, ζ is a positive scaling factor, $d(q,q_{goal})$ is the distance between the robot q and the goal q_{goal}. We set $\zeta = 5$ in our simulation in order to get similar orders of attractive potential and condition number.

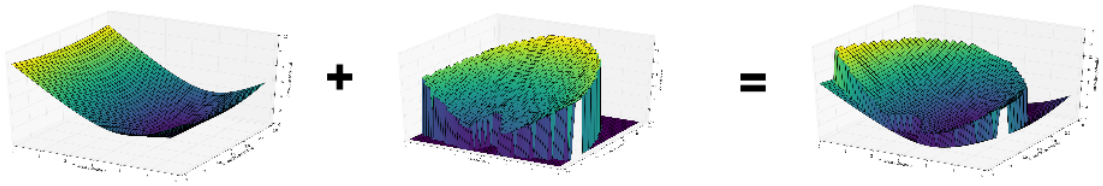


Figure 10.8: Artificial potential fields method in our situation, in this figure the start is $(1.55m,2.05m)$ and the goal is $(1.55m,4.05m)$.

For artificial potential fields method in order to get a smoother computed path, in each iteration we need to expand the search region from 8 adjacent nodes to 24 adjacent nodes.

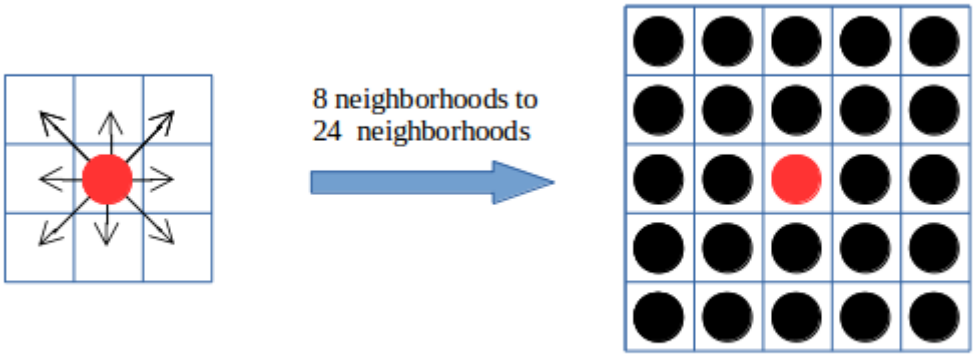
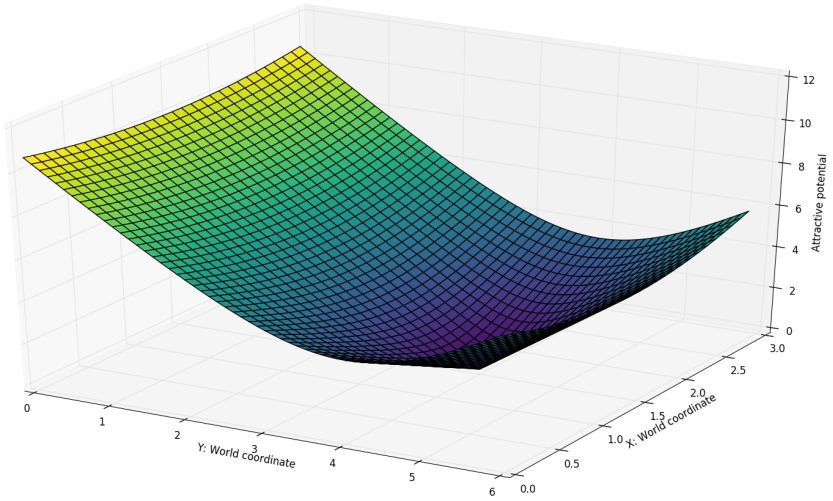
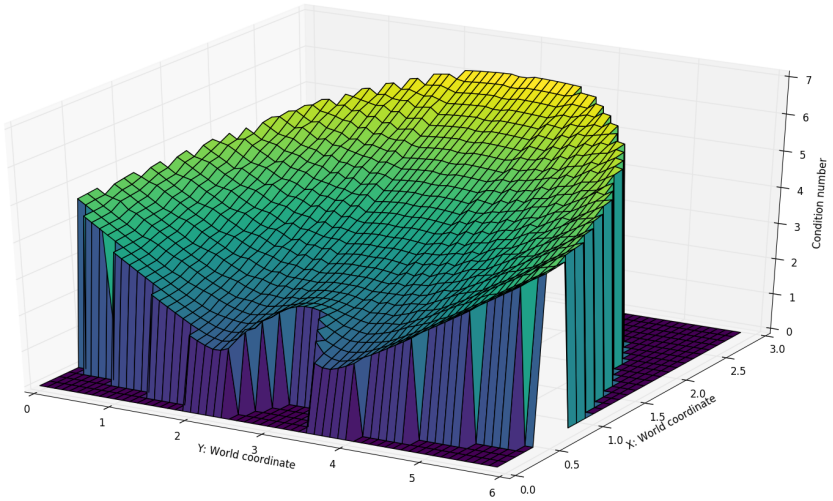


Figure 10.10: Expanding the 8 searching neighborhoods to 24 neighborhoods

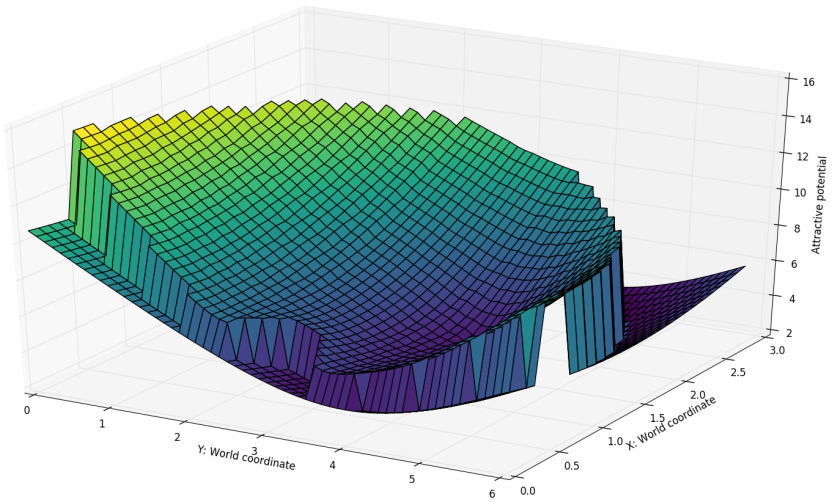
The artificial potential fields method has advantages in that it's has simple structure, easy to implement the underlying real-time control and the path is produced with little computation. However, when the attractive force and repulsive force has similar value but on the opposite direction, the total potential force of the robot would be zero, then it will cause robot to be trapped in local minima or oscillations and fail to find a path[16][25].



(a) Attractive potential fields



(b) Condition number distribution as repulsive potential fields



(c) Total potential fields

Figure 10.9: Artificial potential fields method in our situation

10.4 Compare accuracy of paths calculated with A* and artificial potential fields method

In our simulations we set the grid as a $3m \times 6m$ square and the resolution of the grid as 0.1m. The Iterative method based on Levenberg-Marquardt optimization as the pose estimation algorithm was run at each iteration[18]. Three desired paths, one was computed with regular `A*`, one was was computed with modified `A*` and one was computed with artificial potential fields method are shown in figure 10.11.

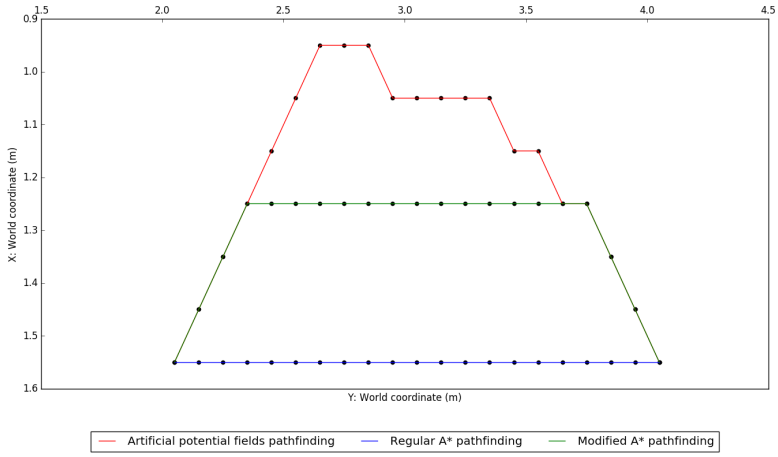


Figure 10.11: Start point: (1.55m,2.05m), goal point: (1.55m,4.05m). The center of the planar marker is at (0,3m). Blue line: path computed with regular `A*`, green line: path computed with modified `A*` and red line: path computed with artificial potential fields method. Each point means each step on the path.

For each desired path, 100 times simulation of the measured paths were performs. For each iteration, 1000 runs of the pose estimation of each step on the desired path were performed. We calculated the mean and standard derivation of errors(euclidean distance) of these 100 times iterations.
In order to better represent the accuracy comparison of different paths with different search methods, we made following comparison separately:

1. Regular `A*` versus artificial potential fields
2. Modified `A*` versus artificial potential fields

`Regular A* versus artificial potential fields`
In figure 10.12 the intervals are built from the mean error of each step, we can find that the errors of the path computed with regular `A*` are obviously larger than the path computed with artificial potential fields method. Figure 10.13 represents the comparison of two paths with another form, we can still notice that the errors of these two paths have huge differences, the path computed with artificial potential fields method is more accurate then the path computed with regular `A*`.

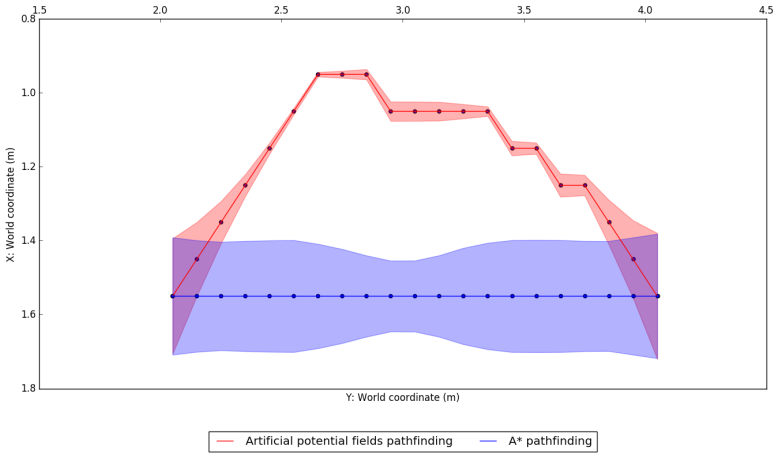


Figure 10.12: The blue intervals and red intervals represent the mean error(euclidean distance) between desired paths and measured paths

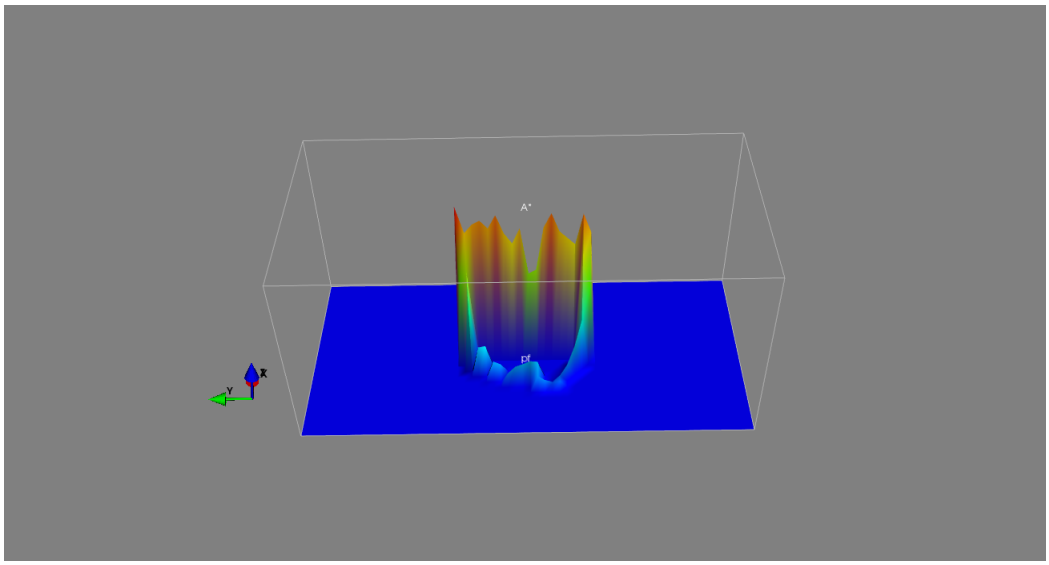


Figure 10.13: Compare the errors of paths computed with regular **A*** and artificial potential fields method

We also compared the errors of both paths with it's mean and it's standard deviation in figure 10.14. It is obviously to notice that the mean error value and standard deviation of path computed with artificial potential fields method are smaller than that on path computed with regular **A*** almost at each step. Sequentially the rotational error and translational error of both paths were compared which are shown in figure 10.15. We can find that the path computed with artificial potential fields method has a smaller rotational error and translational error that computed with regular **A*** almost at each step. Another form to represent the comparisons of rotational error and translational error are shown in figure 10.16 and 10.17.

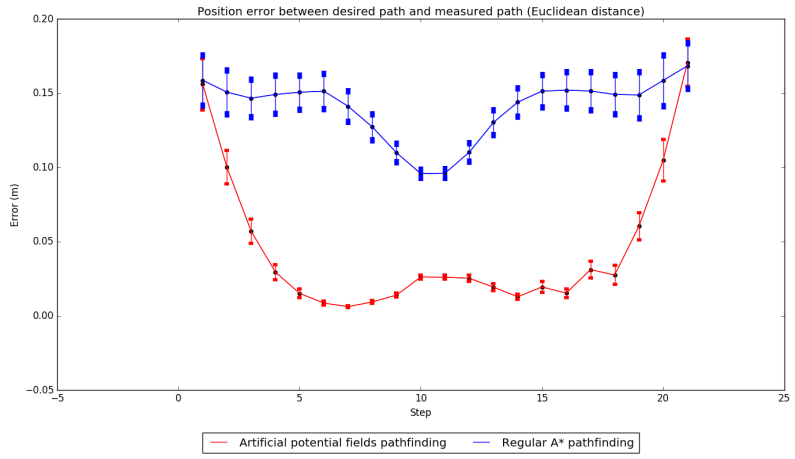


Figure 10.14: Blue line: Mean error of regular **A*** computed path, red line: mean error of artificial potential fields computed path. At each point represents the standard deviation of the error.

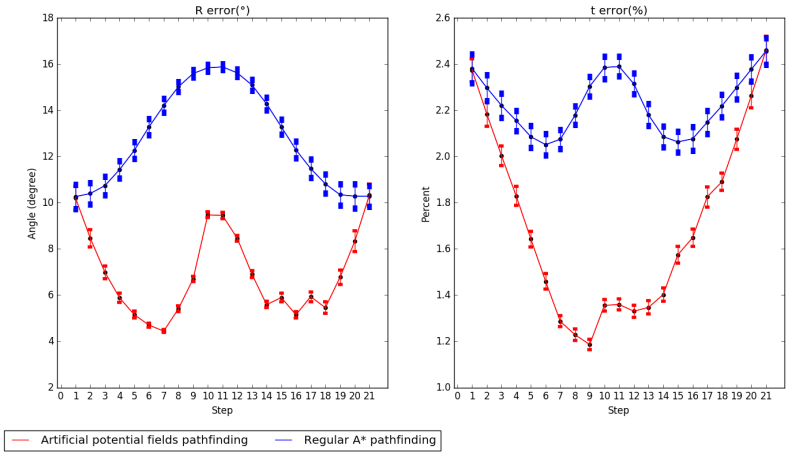


Figure 10.15: Blue line: Rotational error and translational error of path computed with regular **A***, red line: Rotational error and translational error of path computed with artificial potential fields method. At each point represents the error standard deviation.

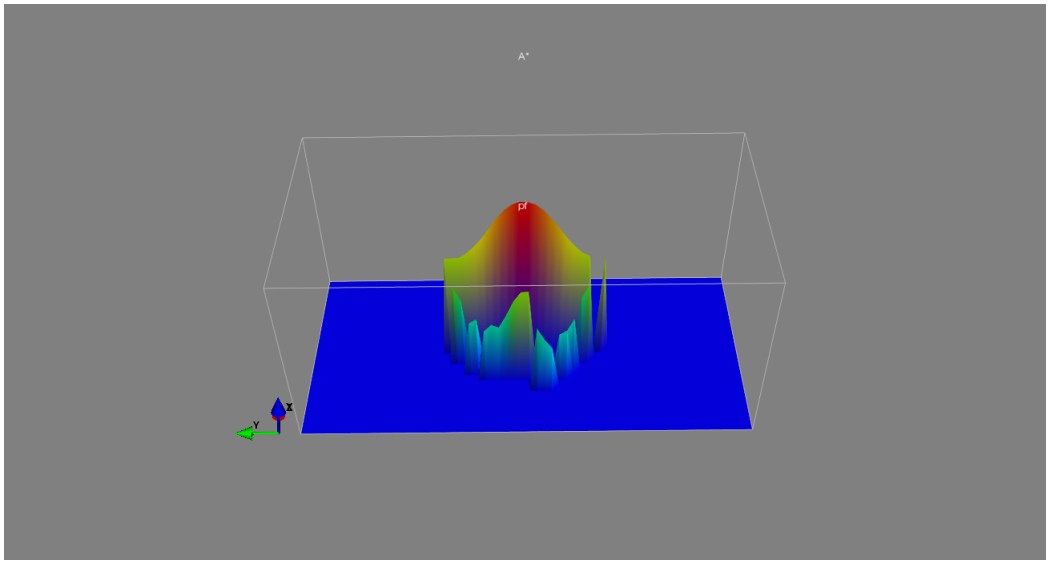


Figure 10.16: Compare the rotational errors of paths computed with regular `A*` and artificial potential fields method

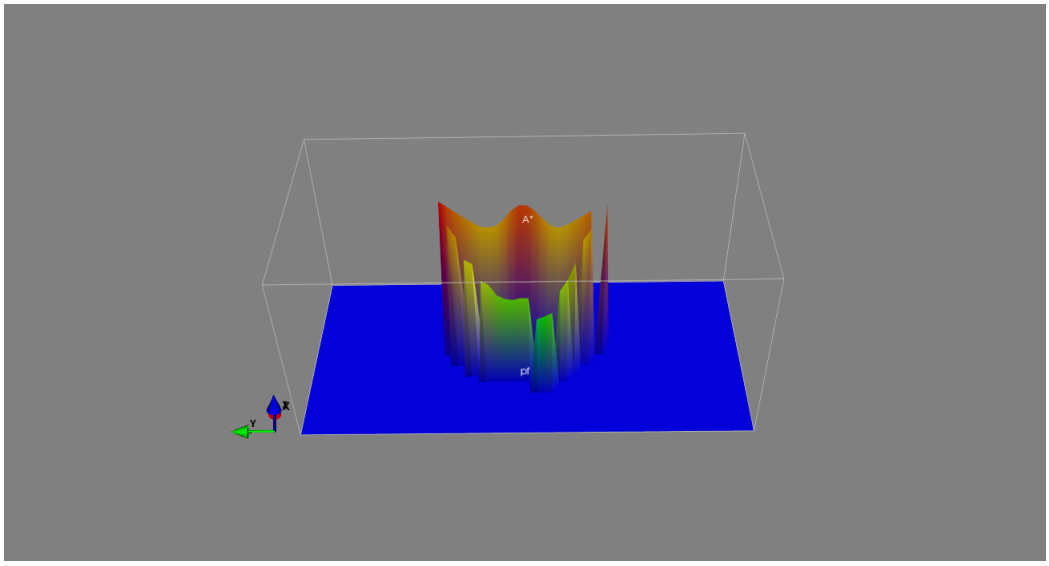


Figure 10.17: Compare the translational errors of paths computed with regular `A*` and artificial potential fields method

`Modified A* versus artificial potential fields`

In figure 10.18 the intervals are built from the mean error of each step, we can find that the errors of the path computed with modified `A*` are obviously larger than the path computed with artificial potential fields method. Figure 10.19 represents the comparison of two paths with another form, we can still notice that the errors of these two paths have huge differences, the path computed with artificial potential fields method is more accurate then the path computed with modified `A*`.

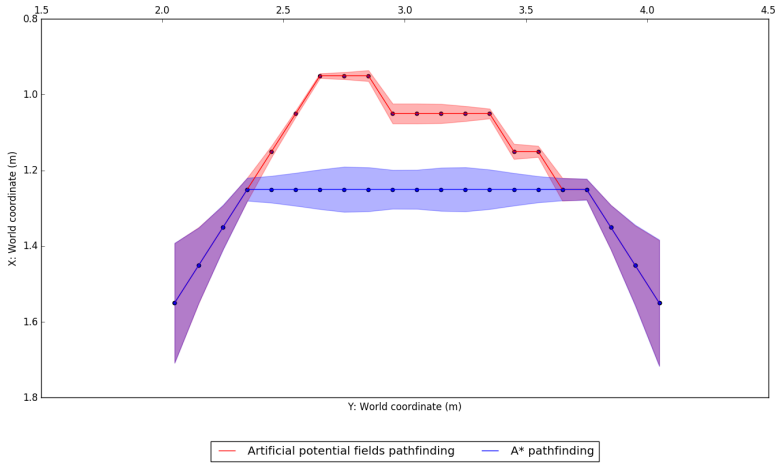


Figure 10.18: The blue intervals and red intervals represent the mean error(euclidean distance) between desired paths and measured paths

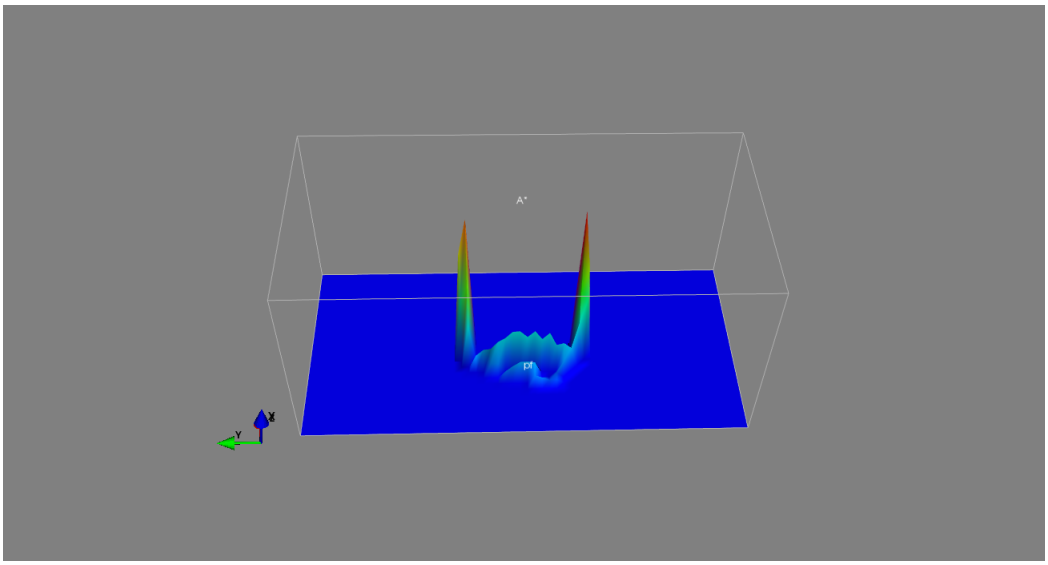


Figure 10.19: Compare the errors of paths computed with modified **A*** and artificial potential fields method

We also compared the errors of both paths with it's mean and it's standard deviation in figure 10.20. It is obviously to notice that the mean error value and standard deviation of path computed with artificial potential fields method are smaller than that on path computed with modified **A*** almost at each step. Sequentially the rotational error and translational error of both paths were compared which are shown in figure 10.21. We can find that the path computed with artificial potential fields method has a smaller rotational error and translational error that computed with modified **A*** almost at each step. Another form to represent the comparisons of rotational error and translational error are shown in figure 10.22 and 10.23.

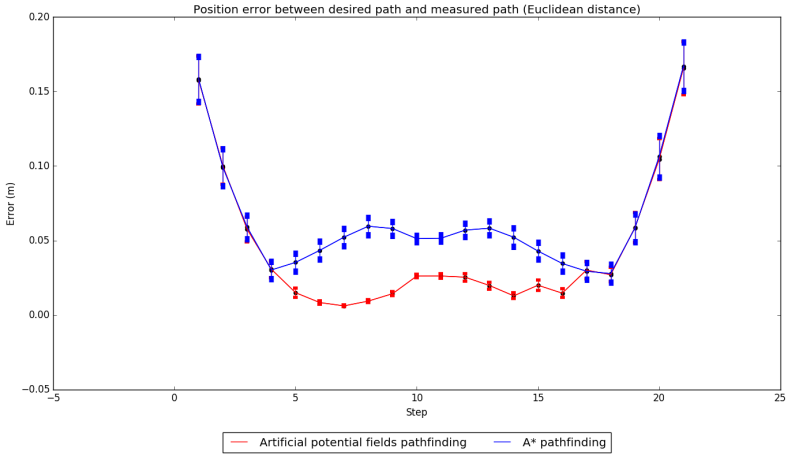


Figure 10.20: Blue line: Mean error of modified **A*** computed path, red line: mean error of artificial potential fields computed path. At each point represents the standard deviation of the error.

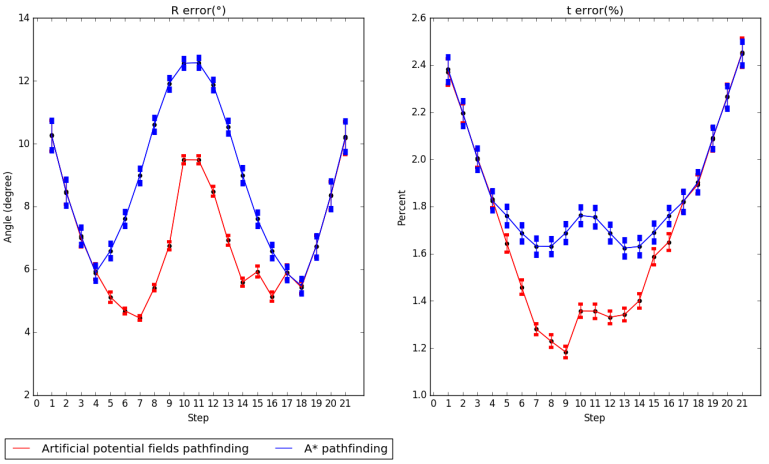


Figure 10.21: Blue line: Rotational error and translational error of path computed with modified **A***, red line: Rotational error and translational error of path computed with artificial potential fields method. At each point represents the error standard deviation.

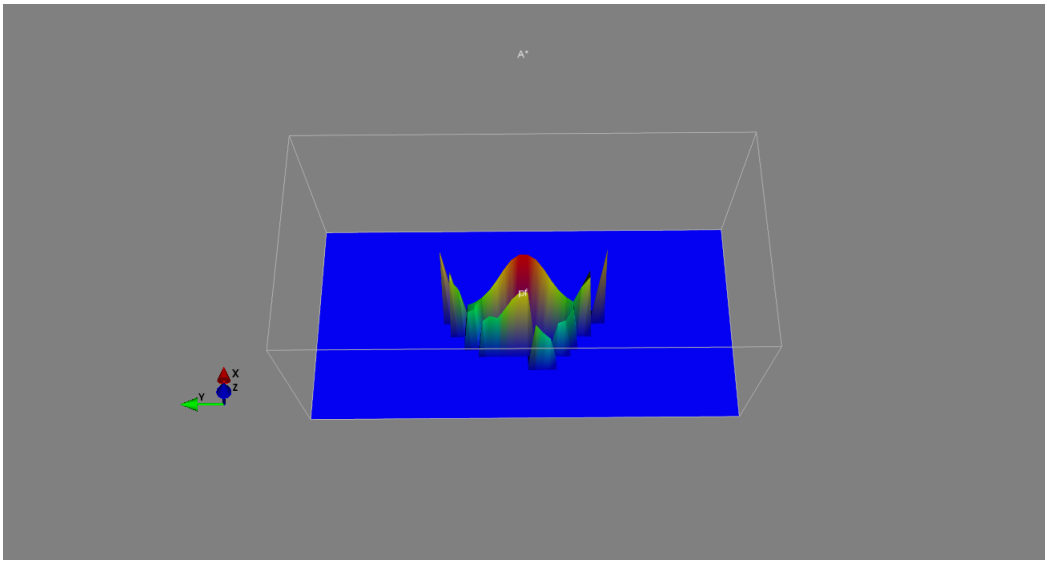


Figure 10.22: Compare the rotational errors of paths computed with modified **A*** and artificial potential fields method

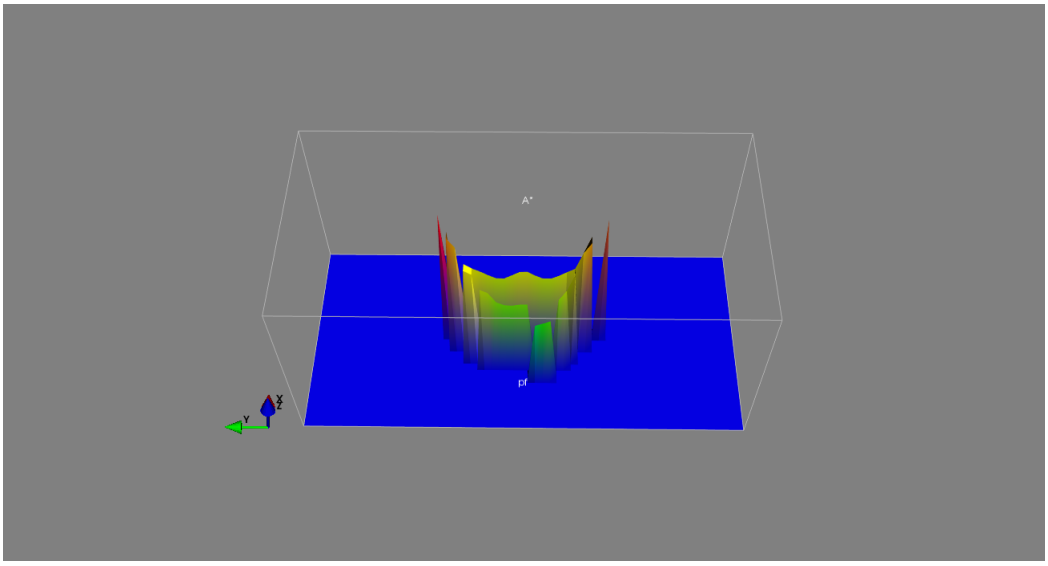


Figure 10.23: Compare the translational errors of paths computed with modified **A*** and artificial potential fields method

All in all, the path computed with artificial potential fields method was more accurate than the path computed with regular **A*** or modified **A*** search algorithm.

11 Conclusion and Future Work

We proposed that the condition number distribution in detection region can be interpreted as the accuracy function. Through this accuracy function we can easily compute the accuracy in the tracking system at given camera pose. Also we already verified the it's feasibility and robustness through comparing the rotational error and translational. Through comparison of different paths computed with `A*`(regular and modified) and artificial potential fields method based on condition number distribution, we know that one possible most accurate path in the detection region of the marker is like a curve, the robot should move firstly towards the marker and then outwards the marker.
In the future work maybe we can find out a better solution for normalization of the image points. We also need to optimize `A*` search algorithm, which means optimizing the heuristic function based on condition number distribution on the grid. We also need to increase the resolution of the grid, e.g. 0.05m or 0.01m. Then We should compare the performance of different search algorithms in following ways:

1. Completeness: does the algorithm find out the solution when there exists one?
2. Optimality: is the solution the best one of all possibility in terms of path cost?
3. Time complexity: how long does it take to find a solution?
4. Space complexity: how much memory is needed to perform the search algorithm?

Bibliography

[1] Daniel F Abawi, Joachim Bienwald, and Ralf Dorner. "Accuracy in optical tracking with fiducial markers: an accuracy function for ARToolKit". In: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality. IEEE Computer Society. 2004, pp. 260–261.

[2] Raul Acuna, Zaijuan Li, and Volker Willert. "MOMA: Visual Mobile Marker Odometry". In: arXiv preprint arXiv:1704.02222 (2017).

[3] Martin Bauer. "Tracking Errors in Augmented Reality". PhD thesis. Technische Universität München, 2007.

[4] Pei Chen and David Suter. "Error analysis in homography estimation by first order approximation tools: a general technique". In: Journal of Mathematical Imaging and Vision 33.3 (2009), pp. 281–295.

[5] Howie Choset. "Robotic Motion Planning: Potential Functions". In: Robotics Institute, Carnegie Mellon University (2010).

[6] Toby Collins and Adrien Bartoli. "Infinitesimal plane-based pose estimation". In: International journal of computer vision 109.3 (2014), pp. 252–286.

[7] Rachael N Darmanin and Marvin K Bugeja. "A review on multi-robot systems categorised by application domain". In: Control and Automation (MED), 2017 25th Mediterranean Conference on. IEEE. 2017, pp. 701–706.

[8] Richard Hartley and Andrew Zisserman. "Multiple View Geometry in Computer Vision Second Edition". In: Cambridge University Press (2000), pp. 106–108.

[9] Liu Jianping. Gradient descent. URL: https://www.cnblogs.com/pinard/p/5970503.html (visited on 2018).

[10] Ryo Kurazume, Shigemi Nagata, and Shigeo Hirose. "Cooperative positioning with multiple robots". In: Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on. IEEE. 1994, pp. 1250–1257.

[11] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem". In: International journal of computer vision 81.2 (2009), p. 155.

[12] Amit Patel. A*'s Use of the Heuristic. URL: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html#heuristics-for-grid-maps (visited on 04/11/2018).

[13] Katharina Pentenrieder, Peter Meier, Gudrun Klinker, et al. "Analysis of tracking accuracy for single-camera square-marker-based tracking". In: Proc. Dritter Workshop Virtuelle und Erweiterte Realitt der GIFachgruppe VR/AR, Koblenz, Germany. 2006.

[14] Prasanna Rangarajan and Panos Papamichalis. "Estimating homographies without normalization". In: Image Processing (ICIP), 2009 16th IEEE International Conference on. IEEE. 2009, pp. 3517–3520.

[15] Volker Willer Raul Acuna. "Dynamic Markers: Optimal control point configurations for homography and pose estimation". In:

[16] RoboTemad2. Artificial Potential Field Approach and its Problems. URL: https://www.robotshop.com/letsmakerobots/artificial-potential-field-approach-and-its-problems (visited on 10/25/2014).

[17] Vincent Spruyt. How to draw a covariance error ellipse? URL: http://www.visiondummy.com/2014/04/draw-error-ellipse-representing-covariance-matrix/ (visited on 04/03/2014).

[18] opencv dev team. Camera Calibration and 3D Reconstruction. URL: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#solvepnp (visited on 05/07/2018).

[19] The Matplotlib development team. matplotlib. URL: https://matplotlib.org/examples/color/colormaps_reference.html (visited on 10/05/2017).

[20] Xiaoqin Wang, Y Ahmet Şekercioğlu, and Tom Drummond. "Vision-based cooperative pose estimation for localization in multi-robot systems equipped with RGB-D cameras". In: Robotics 4.1 (2014), pp. 1–22.

[21] Wikipedia. A* search algorithm. URL: https://en.wikipedia.org/wiki/A*_search_algorithm (visited on 04/28/2018).

[22] Wikipedia. Condition number. URL: https://en.wikipedia.org/wiki/Condition_number (visited on 11/28/2017).

[23] Wikipedia. Distortion (optics). URL: https://en.wikipedia.org/wiki/Distortion_(optics) (visited on 03/12/2018).

[24] Wikipedia. Euler angles. URL: https://en.wikipedia.org/wiki/Euler_angles (visited on 03/14/2018).

[25] Wikipedia. Motion planning. URL: https://en.wikipedia.org/wiki/Motion_planning (visited on 03/14/2018).

[26] Wikipedia. One-factor-at-a-time method. URL: https://en.wikipedia.org/wiki/One-factor-at-a-time_method (visited on 09/07/2017).

[27] Wikipedia. Perspective-n-Point. URL: https://en.wikipedia.org/wiki/Perspective-n-Point (visited on 09/24/2017).

[28] Wikipedia. Rigid transformation. URL: https://en.wikipedia.org/wiki/Rigid_transformation (visited on 10/18/2017).