My Lecture Note: Robotics

Phayuth

February 9, 2025

Contents

Chapter 1

Camera Modeling

This article presents the method of camera modeling and calibration. Projection, Collinearity Model equation, Distortion model, Bundle Adjustment.

Nomenclature

P	Projection Matrix
x	Pixel

1.1 Camera Model

1.1.1 Projection Model

A 3D point in the world coordinate is projected onto the image plane using a camera projection matrix. [2], [1]

$$sp = K[R|t]P_w$$

Here, P_w is a 3D point expressed in the world frame. p is a 2D pixel in the image plane. K is a camera intrinsic matrix. $[R|t]$ is a transformation matrix of the world frame to the camera frame. s is an arbitrary scaling of homogenous projection and is not a part of the model.

To project a 3D point expressed in the camera frame to the image pixel, we use the camera intrinsic matrix K.

$$p = KP_c$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

To transform the point in the world frame to the camera frame, we use the transformation matrix $[R|t]$.

$$P_c = [R|t]P_w$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

The normalized 3D point in the camera frame is determined by

$$x' = \frac{X_c}{Z_c}, y' = \frac{Y_c}{Z_c}, 1 = \frac{Z_c}{Z_c}$$

$$Z_c \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

A distortion-free projection model is

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

If $Z_c \neq 0$, the transformation above is equivalent to

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{X_c}{Z_c} \\ \frac{Y_c}{Z_c} \\ \frac{Z_c}{Z_c} \end{bmatrix} = \begin{bmatrix} f_x \frac{X_c}{Z_c} + c_x \\ f_y \frac{Y_c}{Z_c} + c_y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x x' + c_x \\ f_y y' + c_y \\ 1 \end{bmatrix}$$

1.1.2 Collinearity Model

The collinearity equation in photogrammetry relates the 3D coordinates of a point in object space to its corresponding 2D coordinates in an image through perspective projection. It expresses the condition that the image point, the 3D object point, and the camera's center of projection lie on the same line (are collinear).

$$x = x_0 - f_x \frac{r_{11}(X - X_s) + r_{12}(Y - Y_s) + r_{13}(Z - Z_s)}{r_{31}(X - X_s) + r_{32}(Y - Y_s) + r_{33}(Z - Z_s)}$$

$$y = y_0 - f_y \frac{r_{21}(X - X_s) + r_{22}(Y - Y_s) + r_{23}(Z - Z_s)}{r_{31}(X - X_s) + r_{32}(Y - Y_s) + r_{33}(Z - Z_s)}$$

Theoretically, the pixels calculated from the camera projection and collinearity models must be equal. However, these two models used different coordinate models. To be precise, they are flipped. To correct this issue, we flip the sign in the collinearity equation to make them both equal.

$$x = x_0 + f_x \frac{r_{11}(X - X_s) + r_{12}(Y - Y_s) + r_{13}(Z - Z_s)}{r_{31}(X - X_s) + r_{32}(Y - Y_s) + r_{33}(Z - Z_s)}$$

$$y = y_0 + f_y \frac{r_{21}(X - X_s) + r_{22}(Y - Y_s) + r_{23}(Z - Z_s)}{r_{31}(X - X_s) + r_{32}(Y - Y_s) + r_{33}(Z - Z_s)}$$

Where:

- (X, Y, Z): Coordinates of the object point in 3D space (object space). (Coordinates of the 3D point in world space.)
- (X_s, Y_s, Z_s): Coordinates of the camera's center of projection (camera location). (Camera's position in world space.)

- (f_x, f_y): Focal length of the camera.
- (x_0, y_0): Principal point coordinates (image center) on the image plane.
- r_{ij}: Elements of the rotation matrix that describe the orientation of the camera relative to the object space. R(3x3) matrix.
- (x, y): 2D image coordinates of the projected point in the image plane.

1.2 Distortion Model

The collinearity equations describe the ideal projection of 3D points onto the 2D image plane, assuming a pinhole camera model. However, in real-world cameras, lens distortions—primarily radial and tangential—affect the final position of the image points. We apply corrections after the collinearity equations, which compute the undistorted image coordinates to account for lens distortions.

1.2.1 Correction before projection

To incorporate the distortion in the model, we use

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{bmatrix}$$

where

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x' \frac{1+k_1 r^2 + k_2 r^4 + k_3 r^6}{1+k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2) \\ y' \frac{1+k_1 r^2 + k_2 r^4 + k_3 r^6}{1+k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2y'^2) + 2p_2 x' y' \end{bmatrix}$$

$$r^2 = x'^2 + y'^2$$

1.2.2 Correction after projection

Incorporating Distortion into the Collinearity Equations First, we compute the ideal image points (x_{ideal}, y_{ideal}) using the collinearity equations without distortion. These are the points where the 3D world points would project in a perfect pinhole camera model. Next, we apply distortion to the undistorted points (x_{ideal}, y_{ideal}) to get the distorted image points (x, y)

$$x_{distored} = x_{ideal}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [2p_1 x_{ideal} y_{ideal} + p_2(r^2 + 2x_{ideal}^2)]$$

$$y_{distored} = y_{ideal}(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + [p_1(r^2 + 2y_{ideal}^2) + 2p_2 x_{ideal} y_{ideal}]$$

$$r^2 = (x_{ideal} - x_0)^2 + (y_{ideal} - y_0)^2 distancefromcenter$$

The final coordinates $(x_{distored}, y_{distored})$ are the observed (distorted) image points. the ideal (x_{ideal}, y_{ideal}) are calculated from the collinearity equations.

1.3 Bundle Adjustment

Bundle adjustment is an optimization technique used in computer vision, photogrammetry, and 3D reconstruction to refine the estimates of camera parameters. It simultaneously refines intrinsic parameters (like focal length, principal point, distortion coefficients), extrinsic parameters (camera rotation and translation), and 3D points' positions in the scene, hence the name "bundle," because both camera poses and the bundle of rays (from the camera to the 3D points) are optimized. It works by minimizing the re-projection error between observed and predicted image points. Non-linear least squares optimization (e.g., Levenberg-Marquardt) is used to minimize the re-projection error iteratively.

1.3.1 Calibration

The calibration procedure required the 2D Image Coordinates of Calibration Points and the 3D World Coordinates of Calibration Points.

2D Image Coordinates of Calibration Points. These are the observed pixel coordinates of known points (often from a calibration pattern like a chessboard or a set of precisely placed markers) in the image(s). These points are typically captured using multiple images taken from different angles. (x_i, y_i) in pixel units, where i is the index of a point.

3D World Coordinates of Calibration Points. These are the known coordinates of the calibration points in the real-world reference system (object space). For example, if a chessboard pattern is used, the coordinates of the corners are known in 3D space, usually in some convenient metric (e.g., millimeters). (X_i, Y_i, Z_i). (Z_i could be 0 for points on a flat plane (like a chessboard)).

1.3.2 Optimization Variable

The goal of calibration is to solve for the intrinsic $(f_x, f_y, x_0, y_0, k_1, k_2, k_3, p_1, p_2)$ and extrinsic parameters (camera position ($T = (X_s, Y_s, Z_s)$ and orientation R) by minimizing the difference between the predicted and observed image coordinates.

- (f_x, f_y): the focal lengths in the x and y directions, typically measured in pixel units. In many cases, the focal lengths are assumed to be the same, but they could differ depending on the camera's aspect ratio.

- (x_0, y_0): Principle Point is the point where the optical axis intersects the image plane, ideally near the center of the image but it might slightly shift due to lens alignment issues.

- $(k_1, k_2, k_3), (p_1, p_2)$: Radial Distortion, Tangential Distortion.

- R: the 3×3 rotation matrix defines the orientation of the camera relative to the world coordinate system.

- $T = (X_s, Y_s, Z_s)$: The 3×1 translation vector defines the position of the camera in the world coordinate system.

1.3.3 Calibrate

The goal of bundle adjustment is to minimize the re-projection error, which is the difference between the actual observed 2D image coordinates and the projected 2D points based on the current estimates of the 3D points and camera parameters.

Formulating the Error Function:

$$Error = \sum_{i=1}^{N}((x_{observed} - x_{projection})^2 + (y_{observed} - y_{projection})^2)$$

$(x_{observed}, y_{observed})$ are calculated using the collinearity equations. $(x_{projection}, y_{projection})$ are the actual image points.

- Initial Estimates: You start with initial guesses of the 3D points in the scene and the camera parameters, which might come from other methods such as feature matching or structure-from-motion (SfM). This can be done using Direct linear transformation (DLT) or another simple method. Intrinsic camera parameters (like focal length) can be approximated based on the sensor size and field of view.
- Projection of 3D Points: Each 3D point in the scene is projected into the 2D image plane of each camera according to the camera model and its parameters (such as focal length, distortion coefficients, and camera position/orientation).
- Re-projection Error: The difference between where the 3D points project onto the image (based on the estimated camera parameters) and where the actual observed points are in the image is calculated. This difference is called the re-projection error.
- Non-linear Optimization: The bundle adjustment process uses non-linear least squares optimization (e.g., Levenberg-Marquardt algorithm) to minimize the re-projection error by simultaneously adjusting the 3D points and the camera parameters.
- Refined Results: After optimization, the adjusted 3D points and camera parameters more accurately represent the scene and the imaging setup. This results in a better 3D reconstruction, more precise camera calibration, or improved pose estimation.
- The optimization continues until the re-projection error converges below a certain threshold, meaning the estimated parameters are consistent with the observed image points.

1.3.4 Calibration Result Evaluation

Evaluation of Calibration (Re-projection Error) The overall quality of the calibration is typically evaluated using the mean re-projection error MRE:

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\sqrt{((x_{observed} - x_{projection})^2 + (y_{observed} - y_{projection})^2)}$$

A low re-projection error indicates good calibration quality.

Bibliography for Chapter 1

[1] *Camera Calibration and 3D Reconstruction 2014; OpenCV documentation — docs.opencv.org.* https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. [Accessed 08-02-2025] (cit. on p. 5).

[2] *Camera Models Explained — calib.io.* https://calib.io/blogs/knowledge-base/camera-models. [Accessed 08-02-2025] (cit. on p. 5).

Chapter 2

Torus Planning

Path planning in robotic manipulators can be done in $\mathcal{W}$ and $\mathcal{C}$. Unlike mobile robotic where its $\mathcal{C}$ is living in $\mathcal{R}^n$, it is living in $\mathcal{T}^n$ since each joint in a revolute $\mathcal{S}^1$. However, since the physical system (motor) has a joint limit, the $\mathcal{C}$ has discontinuity that lets people see $\mathcal{T}$ as $\mathcal{R}$. Sampling-based path planning algorithms often sample a point between this interval of $\mathcal{R}$, therefore treating $\mathcal{R}$ as $\mathcal{T}$, which is not the case. This report shows path-planning studies on Torus Planning using various Sampling-based algorithms.

Nomenclature

$\mathcal{W}$	Workspace
$\mathcal{C}$	configuration space
q	configuration node
T	Torus
S	S^1 one dimensional sphere
E	E^1 one-dimensional euclidean (flat)

2.1 Introduction

The problem is why robots move in a long distance. Problem introduction. the solution is 1. Grid-based search using A*. discretized space to the grid search index and wrap index. 2. geodesic graph of convex set by Tom Mit.

2.2 Literature Review

2.2.1 Modern Robotic

Read more : [5]
P.359

Figure 10.2 shows the robot's $\mathcal{C}$-spacein torus. The author uses the grid method over the discretized space K^n. K is the number of segments per dimension and n is the number of dimensions. In the grid method, the neighbor is a four/eight directional index.
P.371

Although it is easy to implement for low-dimension, the number of grids increases exponentially with high-dimension. We might miss the free path if we reduce the grid

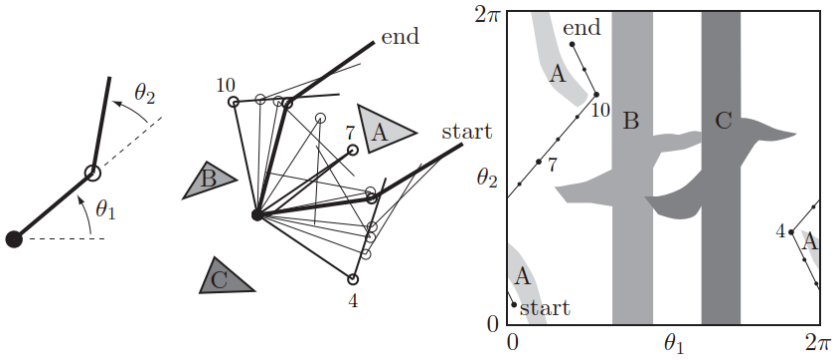

Figure 2.1: (Left) The joint angles of a 2R robot arm. (Middle) The arm navigating among obstacles A, B, and C. (Right) The same motion in C-space. Three intermediate points, 4, 7, and 10, along the path are labeled.

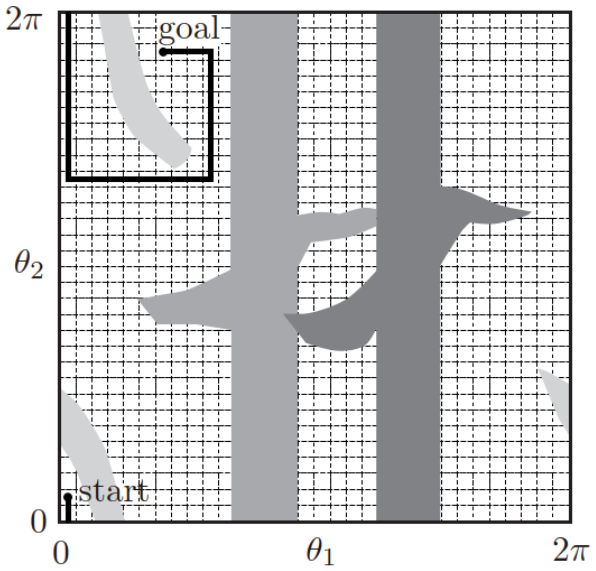

Figure 2.2: Grid-based planning using manhattan distance.

resolution to save space. We conclude that the grid's torus is already solved in a low-dimensional system.
P....

$\mathcal{C}$-spacehas its natural underlying geometry. But how can we parameterize it? There are two ways to parameterize the geometry (coordinate system). Explicit parameterization: the number of parameters equal to the number of the system's state, using the real number in $\mathcal{R}$. Implicit parameterization: the parameter number is higher than the number of the system's state, defined with constraint equation i.e. $x^2 + y^2 = 1^2$.
Note

Normally, we perform sampling in the bounded limit (explicit) per every parameter. We use real numbers in the domain R. We use the Euclidean norm as the distance metric. Therefore, we have accidentally put the system from $\mathcal{T}^n$ to $\mathcal{R}^n$.

2.2.2 Handbook of Robotic

Read more : [2]
P.124 : Keywords are topology of $\mathcal{C}$-space, manifold, homeomorphism.

The majority of the planning problem has a $\mathcal{C}$-spacemanifold. The system that has its $\mathcal{C}$-spacethat is NOT a manifold is a closed unit square. It defined as $[0, 1] \times 0, 1] \subset R^2$.

We can represent a $\mathcal{C}$-spacemanifold with just n parameters where n is equal to the number of dimensions (explicit). Although an n-dimensional manifold can be represented using as few as n parameters, due to some constraints, it might be easier to use a representation that has a higher number of parameters. i.e $S^1 = \{(x, y)|x^2 + y^2 = 1^2\}$, S^1 is embedded in R^2, T^2 is embedded in R^3.

2.2.3 Robot Motion Planning

Read more : [3]
P.387

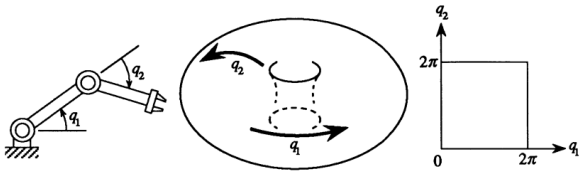

Figure 2.3: Embedded Torus in Euclidean Space.

Figure 9. For a 2-joint planar robot, every configuration is represented as $(q_1, q_2) \in [0, 2\pi) \times [0, 2\pi)$. The opposite edge of the square shown in the $\mathcal{C}$-spacemust be procedurally identified through modulo 2π arithmetic to capture multiple connectedness of the torus.

Robots with revolute joints have multiply connected $\mathcal{C}$-spaceand several charts are necessary to form an ATLAS. However, it is often sufficient to consider a single chart in practice with each angular coordinate belonging to $[0, 2\pi)$ and applying modulo 2π arithmetic to the coordinate.

Consider the 2-joints robot. The $\mathcal{C}$-spaceis $S^1 \times S^1 = T^2$. Hence, it is a torus in the embedded three-dimensional Euclidean space.

A chart on this manifold can be defined by associating an angle in $[0, 2\pi)$ with each of the two joints. The torus is multiply connected and several charts are required to form

an atlas. Nevertheless, in most cases, it is appropriate to consider a single chart and let the two angles q_1 and q_2 vary on $[0, 2\pi)$, with modulo 2π arithmetic. This simplification corresponds to representing the torus with a square $[0, 2\pi] \times [0, 2\pi]$ whose opposite edges are procedurally identified.
P.388

In general, the relative motion of two objects connected by a joint is further limited by a pair of mechanical stops, which restrict the value range of the corresponding configuration coordinate to a connected interval. The mechanical stops act like invisible obstacles. The set of configurations where no joint has reached a mechanical stop is an open subset of $\mathcal{C}$-space. The complement of this subset is the "C-obstacle region" to which the invisible obstacles due to the mechanical stops map. The mechanical stops in every joint are usually independent of the configuration of the robot. Then the subset of configurations that are left feasible by the mechanical stops is simply a cross-product of intervals in the Cartesian space (chart) representing C.

Consider a revolute joint connecting two bodies A and B. If the mechanical stop in this joint prevents B from making a full rotation at 2π with respect to A, the limitation eliminates the need for applying modulo 2π arithmetic to the corresponding configuration coordinate. If, instead, the mechanical stop allows B to perform $r(r > 1)$ full rotations, then it is appropriate length $2r\pi$ (r does not have to be an integer). In this way, the configuration is explicitly represented with respect to both mechanical stops. One drawback of this representation, however, is that the same obstacle may map to several Cobstacle regions of identical shape.

2.2.4 Principles Of Robot Motion

Read more : [1]
P.40

Robot moves in the two and three-dimensional Euclidean ambient space, represented by R^2 or R^3, respectively. We sometimes refer to this ambient space as the workspace. Other times the term workspace has a more specific meaning. For example, for a robot arm, often we call the workspace the set of points of the ambient space reachable by a point on the hand or end effector (see figure 3.3).
P.41

Each joint angle θi corresponds to a point on the unit circle S1, and the configuration space is T^2, the two-dimensional torus. It is common to picture a torus as the surface of a doughnut because a torus has a natural embedding in R3, just as a circle S1 has a natural embedding in R2.

By cutting the torus along the $\theta 1 = 0$ and $\theta 2 = 0$ curve, we flatten the torus onto the plane, With this planar representation, we are identifying points on S1 by points in the interval $[0, 2\pi) \subset R$. While this representation covers all points in $S1$, the interval $[0, 2\pi)$, being a subset of the real line, does not naturally wrap around like S1, so there is a discontinuity in the representation. As we discuss in section 3.4, this is because S1 is topologically different from any interval of R.
P.45

Complex robot $\mathcal{C}$-spaceis difficult to compute. For this reason, the grid-based representation of $\mathcal{C}$-spaceis used. For a 2-joint robot, we define a grid on the surface of the torus.
P.54

We are often concerned only with the local properties of configuration spaces. Local properties are defined on neighborhoods. For metric spaces (A metric space is a space equipped with a distance metric), neighborhoods are most easily defined in terms of open balls. For a point p of some manifold M, we define an open ball of radius ϵ by

$$B_\epsilon(p) = \{p' \in M | d(p, p') < \epsilon\} \tag{2.1}$$

where d is metric on M.

A neighborhood of a point $p \in M$ is any subset $U \subset= M$ with $p \in U$ such that for every $p' \in U$, there exists an open ball $B_\epsilon(p') \subset U$. Any open ball is itself a neighborhood.

For example, The open disk in the plane. For the point (x0, y0) in the plane, an open ball defined by the Euclidean metric is

$$B_\epsilon(x0, y0) = \{(x, y) | (x - x0)^2 + (y - y0)^2 < \epsilon^2\} \tag{2.2}$$

We say that S is locally diffeomorphic (resp. locally homeomorphic) to T if for each $p \in S$ there exists a diffeomorphism (resp. homeomorphism) f from S to T on some neighborhood U with $p \in U$.

For the circular mobile robot, the workspace and the configuration space are diffeomorphic. This is easy to see, since both are copies of $R2$. In this case, the identity map $\phi(x) = x$ is a perfectly fine global diffeomorphism between the workspace and configuration space. In contrast, the two-joint manipulator has a configuration space that is T^2, the torus. The torus T^2 is not diffeomorphic to R^2, but it is locally diffeomorphic. If the revolute joints in the two-joint manipulator have lower and upper limits, $\theta_l i < \theta_i < \theta_u i$, so that they cannot perform a complete revolution, however, then the configuration space of the two-joint manipulator becomes an open subset of the torus, which is diffeomorphic to R^2 (globally). This follows from the fact that each joint angle lies in an open interval of R^1, and we can "stretch" that open interval to cover the line.

P.56

The term chart is analogous to its use in cartography since the subset U is "charted" onto Rk in much the same way that cartographers chart subsets of the globe onto a plane when creating maps. Charts are sometimes referred to as coordinate systems because each point in the set U is assigned a set of coordinates in an Euclidean space. The inverse diffeomorphism, $\phi^{-1} : Rk \to U$, is referred to as a parameterization of the manifold.

We referred to $\theta 1$, and $\theta 2$ as parameters that represent a configuration of the two-joint arm. Recall that $(\theta 1, \theta 2) \in R2$, and that when considered as a representation of the configuration, they define a point in $T2$, the configuration space, which is a manifold. We now see that in section 3.1, when we represented a configuration of the planar arm by the pair $(\theta 1, \theta 2)$, we were in fact creating a chart from a subset of the configuration space to a subset of R2.

Mapping from T2 to R2 encounters continuity problems at $\{0, 2\pi\}$. For many other $\mathcal{C}$-space, it will be the case that we cannot construct a single chart whose domain contains the entire $\mathcal{C}$-space. In these cases, we have to construct collections of charts that cover the $\mathcal{C}$-space. We are not free to choose any arbitrary type of chart. They must be compatible with the manifold on which their domains overlap. Two compatible charts are said to be $C\infty$-related. When two charts are related, we can switch back and forth smoothly over the overlap domain.

A set of charts that are $C\infty$-related, and whose domains cover the entire configuration space Q, form an atlas for Q. An atlas is sometimes referred to as the differentiable structure for Q. Together, the atlas and Q comprise a differentiable manifold.

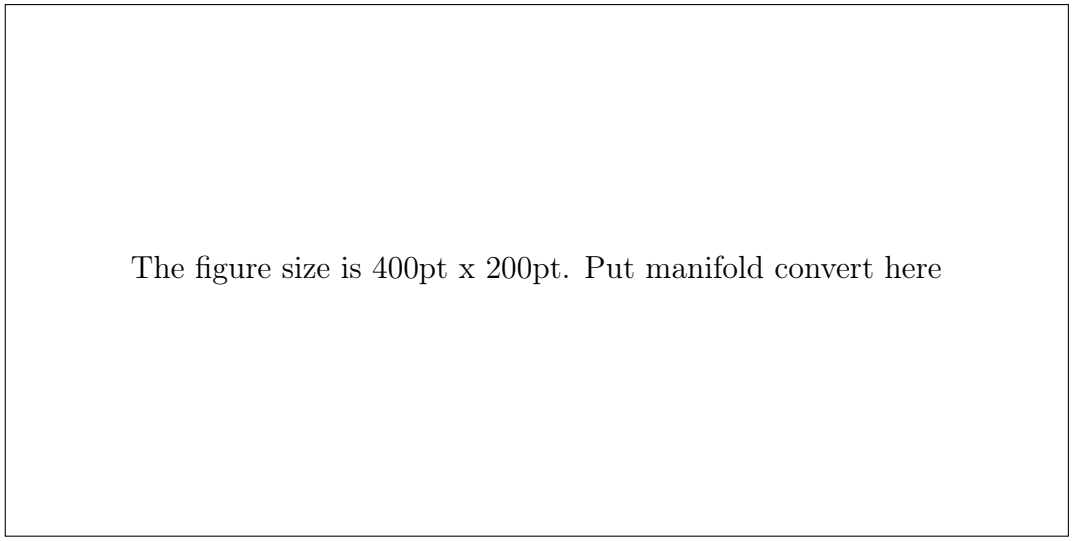

Figure 2.4: Caption here.

2.2.5 Planning Algorithm

Read more : [4]
P.29

Discrete planning. State $x \in X$, action $u \in U(x)$, State transition function $f(x, u) \in X$.

For example, suppose a robot moves on a grid, and each grid point has integer coordinate (i, j). Robots take the action set (up, down, left, and right).

$$\begin{aligned} \text{Action } U &= \{(0,1),(0,-1),(1,0),(-1,0)\} \\ \text{Transition } f(x,u) &= x+u \end{aligned} \tag{2.3}$$

For example if $x = (3, 4)$ and $u = (0, 1)$, then $f(x, u) = (3, 5)$.

Forward search is like a forest fire. we can use best-first, breadth-first, A*, -etc.
P.186

Virtually, all the sampling-based planning requires a function that measures the distance between two points in $\mathcal{C}$-space. Metric space conditions are nonnegative, reflexivity, symmetry, and triangle inequality.

SO(2) Metric by comparing angle

$$\rho(\theta 1, \theta 2) = min\{|\theta 1 - \theta 2|, 2\pi - |\theta 1 - \theta 2|\} \tag{2.4}$$

P.190

For example 5.7, Tn metrics. The cartesian product rules 2.5, and 2.6 can be extended over every copy of $S1$ (one for each parameter θ). This leads to n arbitrary coefficient $c1, c2, ..., cn$.

$$\rho(z, z') = \rho(x, y, x', y') = c1\rho_x(x, x') + c2\rho_y(y, y') \tag{2.5}$$

$$\rho(z, z') = (c1\rho_x(x, x')^p + c2\rho_y(y, y')^p)^{1/p} \tag{2.6}$$

If p=2, we get an Euclidean distance.

2.3 Torus Geometry

A manipulator where every joint is revolute has a general configuration space in the torus. The total configuration of the torus is the cartesian product of one-dimensional unit sphere given as:

$$T^n = S^1 \times S^1 \times ... \times S^1 \tag{2.7}$$

2.3.1 Distance

There are two ways for a point on a circle to move to another point (i.e, θ_1 to θ_2) based on forward or backward direction. As a result, there are two values of distance between them. They are the positive direction ($|\theta_1 - \theta_2|$) or a negative direction ($2\pi - |\theta_1 - \theta_2|$). To get the shortest distance, we take the minimum value as in the given value of ρ.

$$\rho \leftarrow \min\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\} \tag{2.8}$$

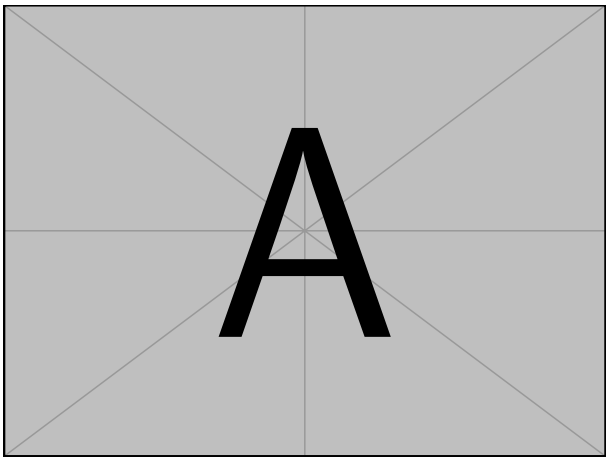

Figure 2.5: Two directions of moving from point to point on a circle.

The distance above is for a single robot joint. In the case of multiple joints, the minimum distance between two configurations is the Euclidean distance of the minimum value of each joint. The algorithm below provides a detailed equation.

Algorithm 1 Distance on Torus

function DISTTORUS(q_a, q_b)
- $\Delta q \leftarrow |q_b - q_a|$
- $\Delta q_w \leftarrow 2\pi - \Delta q$
- $m \leftarrow \min\{q, q_w\}$ ▷ *element wise minimum*
- **return** $\rho \leftarrow ||m||_2$

In a circle, the possible equivalent values are repeats of every $2k\pi$. To find the value in the lowest range of $\pm\pi$, we use a wrap function based on modular math.

Each alternative configuration value of the robot can be represented within a unit circle. Algorithm 3 outlines the method for determining all feasible alternative configurations that will be used in the planning process.

Algorithm 2 Wrap to $\pm\pi$ range

function WRAPTOPI(q)
  return $q_{wrap} \leftarrow (q + \pi) \mod 2\pi - \pi$

In this method, all possible combinations of the v are computed using the cartesian product up to the number of degrees of the system (3^d) denoted as $Q_{shifted}$. The summation result between each $q_{shifted}$ and the wrapped value of the original configuration is the alternative configurations denoted as q_{alts}. In such cases, there are many candidate configurations to plan a path toward. However, it is unrealistic to consider every value. Therefore, any infeasible candidate outside the physical joint limit is filtered out. The remains are feasible candidates denoted as $Q_{feasible}$.

$$v \leftarrow \{-2k\pi, ..., 0, ..., 2k\pi\}, k \in 1, 2, ..., n \tag{2.9}$$

ps: the candidate vector v is chosen with k=1 because the real hardware such as ur5e has a joint limit from -2pi to 2pi.

Algorithm 3 Find Alternative Configuration

function FINDALTCONFIG(q)
  $d \leftarrow$ SYSTEM DIMENSION
  $v \leftarrow \{-2\pi, 0, 2\pi\}$
  $Q_{feasible} \leftarrow \{\emptyset\}$
  $q_{wrap} \leftarrow$ WRAPTOPI(q)
  $Q_{shifted} \leftarrow \{v \times v \times ...(d)\}$ ▷ *CartesianProduct d times*
  for $q_{shifted} \in Q_{shifted}$ **do**
    $q_{alt} \leftarrow q_{wrap} + q_{shifted}$
    if ISINJOINTLIMIT(q_{alt}) **then**
      $Q_{feasible} \leftarrow Q_{feasible} \cup \{q_{alt}\}$
  return $Q_{feasible}$

2.4 Sampling-based Path Planning

A unique tree is all trees in space are unique, the same as planning as if the space is not wrapped at all. A redundant tree is a tree that is the same for every space when extended as it has torus periodic properties.

2.4.1 Unique Tree

Multiple Query - Unique Tree

Please memorize the following text:

When using the Probabilistic Roadmap (PRM) in the 2pi range (physical joint limit), we construct the roadmap on the C-space as usual. However, the search phase must be adjusted to account for multiple goals. First, we use the "find alt" function to determine alternative goal configurations and connect them to the roadmap. We then search for multiple goals and stop when one goal is reached. If A* is used for the search, the cost-to-go heuristic can be represented as the sum of the Euclidean distances to all goal configurations.

$$costtogo(q) \leftarrow \sum_{i=1}^{numgoal} ||q_{goal,i} - q||_2 \tag{2.10}$$

Single Query - Unique Tree

Here, are the extensions for a single query algorithm that can be easily used with an existing motion planning library. We extend the sampling space at the joint limit.

Single Tree

Simple extension for all algorithm with single tree such as RRT. Instead of single goal, we use multiple alternative goal during goal check once the new node is created. The algorithm is shown in Algorithm 4. The resulted path can be obtained by searching for the node in Q_{soln} with the shortest cost with its corresponding goal value.

Algorithm 4 Single Tree Extension

```
function InGoalsRadiusCheck(q_new)
    Q_alts ← FindAltConfig(q_g)
    for q_alt ∈ Q_alts do
        if ||q_alt − q_new||_2 < η then
            Q_soln ← Q_soln ∪ {(q_new, q_alt)}
```

Goal bias sampling has shown a better result than simple uniform sampling in a single-tree algorithm. Here, we extend bias sampling toward multiple goals.

- Single Bias Initial: for a small probability, focus bias to the alternative configuration that is the nearest to the initial configuration.

$$q_{bias} \leftarrow \underset{q_g \in Q_g}{argmin}(||q_g - q_s||_2) \tag{2.11}$$

- Single Bias Nearest Goal: for a small probability, focus bias to an alternative configuration that is the nearest to the current tree node.

$$q_{bias} \leftarrow \underset{q \in T}{argmin}(\underset{q_g \in Q_g}{argmin}(||q - q_g||_2)) \tag{2.12}$$

- Multiple Bias: for a small probability, extend the RRT tree toward each goal $q \in Q_{goal}$

Bi-Tree

This is a simple extension for bidirectional trees such as RRT-Connect. Initialize multiple alternative goals on the backward tree. The method is similar to [ref].

Algorithm 5 Bi Tree Extension

$Q_{alts} \leftarrow$ FindAltConfig(q_g)
$T_a \leftarrow \{q_s\}$
$T_b \leftarrow \{q \in Q_{alts}\}$
$Q_{soln} \leftarrow \{\emptyset\}$
for $i = 1, \ldots, I_{max}$ **do**
  $q_{rand} \leftarrow$ RandomSampleFree$(\pm 2\pi)$
  if Extend$(T_a, q_{rand}) \neq Trapped$ **then**
    Connect(T_b, q_{new})
    $Q_{soln} \leftarrow Q_{soln} \cup \{q_{new}\}$
  Swap(T_a, T_b) ▷ *Swap Tree*

The solution is determined by:

$$q_{opt} \leftarrow \underset{q \in Q_{Soln}}{argmin}(\text{Cost}(q, T_a) + \text{Cost}(q, T_b)) \tag{2.13}$$

Reducing Candidates : With many states, there are many candidate configurations to plan a path toward. However, it is unrealistic to consider everything. One solution is filtering out any infeasible candidate :

- Physical joint limit for every joint. This has been implemented in IsInLimit function.
- any candidates that is too far from the initial configuration

$$Q_{feasible} \leftarrow \{q_g \in Q_g : ||q_g - q_s||_2 < ||q_{og} - q_s||_2\} \tag{2.14}$$

2.4.2 Redundant Tree

Multiple Query - Redundant Tree

PRM in pi range. Here, our goal is to stitches the nodes on the border of the boundary to all connected edge using near function with torus distance.

Once the edges are connected in PRM, the A* algorithm is used to search for a path with some modification.

Single Query - Redundant Tree

Single Tree

Single Tree based - finding nearest and near using distance on torus function. since the space is redundant, we plan only in the lowest possible space by wrap all configuration down to pi range.

Bi-Tree

Algorithm 6 PRM-Torus Build Phase

$G \leftarrow \{\emptyset\}$
for $i = 1, \ldots, I_{max}$ **do**
  $q_{rand} \leftarrow$ RandomSampleFree($\pm\pi$)
  $Q_{near} \leftarrow$ NearTorus(G, q_{rand}, R)
  for $q_{near} \in Q_{near}$ **do**
    if ObFree(q_{rand}, q_{near}) $\wedge$ $notconnectedalready$ **then**
      $q_{rand}.edge \leftarrow q_{near}$
      $q_{near}.edge \leftarrow q_{rand}$
return T

Algorithm 7 Search Phase in Torus

function SearchPhase(G, q_{sog}, q_{gog})
  $q_s \leftarrow$ WrapToPi(q_{sog})
  $q_g \leftarrow$ WrapToPi(q_{gog})
  $\sigma \leftarrow$ A*Search(G, q_s, q_g)
  $\sigma_{unwrap} \leftarrow$ UnWrapPath(σ)
return σ_{unwrap}

Algorithm 8 RRT-Torus

$q_s \leftarrow$ WrapToPi(q_{sog})
$q_g \leftarrow$ WrapToPi(q_{gog})
$T \leftarrow \{q_s\}$
$Q_{soln} \leftarrow \{\emptyset\}$
for $i = 1, \ldots, I_{max}$ **do**
  $q_{rand} \leftarrow$ RandomSampleFree($\pm\pi$)
  $q_{nearest} \leftarrow$ NearestTorus(T, q_{rand})
  $q_{new} \leftarrow$ SteerTorus($q_{nearest}, q_{rand}, \eta$)
  if IsColFree($q_{nearest}, q_{new}$) **then**
    $T \leftarrow T \cup \{q_{new}\}$
    if IsInGoal(q_{new}, q_g, R) **then**
      $Q_{soln} \leftarrow Q_{soln} \cup \{q_{new}\}$
$\sigma_{wrap} \leftarrow$ GetPath(Q_{soln})
return $\sigma \leftarrow$ UnWrapPath(σ_{wrap})

Algorithm 9 RRT-Connect-Torus

$q_s \leftarrow$ WrapToPi(q_{sog})
$q_g \leftarrow$ WrapToPi(q_{gog})
$T_a \leftarrow \{q_s\}$
$T_b \leftarrow \{q_g\}$
for $i = 1, \ldots, I_{max}$ **do**
  $q_{rand} \leftarrow$ RandomSampleFree($\pm 2\pi$)
  if Extend(T_a, q_{rand}) $\neq$ $Trapped$ **then**
    Connect(T_b, q_{new})
  Swap(T_a, T_b) ▷ *Swap Tree*

Steer on torus is to created a new node in between the line of two node while taking into account for torus warp. the new created node is in the minimum direction.

Transform path in pi range to original range. The founded path on redundant tree

Algorithm 10 Steer on Torus

function STEERTORUS(q_a, q_b, η)
- $di \leftarrow q_b - q_a$ not correct yet
- $\Delta q \leftarrow di/||di||_2 * \eta$
- $q_{new} \leftarrow q_a + \Delta q$

is represented in a lowest range +- pi. However, the robot configuration live in +- 2pi, and thus we need to correct the resulted path by lift it up to the original value. because the physical system is not teleport. The result path found must be the shortest, respect continuous motion (not jump), and respect joint limit. For example, it can cross pi boundary but it can not cross 2pi boundary even if it is wrap around.

Algorithm 11 UnWrap Path

function UNWRAPPATH(σ)
- $\sigma_{unwrap} \leftarrow \{q_{sog}\}$
- **for** $i = 2, \ldots, len(\sigma)$ **do**
  - $Q_c \leftarrow$ FINDALTCONFIG(σ_i)
  - $\sigma_{unwrap} \leftarrow \sigma_{unwrap} \cup \{\underset{q_c \in Q_c}{argmin}(||q_c - \sigma_{unwrap,i-1}||_2)\}$

2.5 Experimental Demonstration

2.5.1 Simulation Experiment

Case 1: design a simple experiment case where the robot can move back to goal in the shortest distance instead of forward in a long distance.

Case 2: design a experiment where there are no path if we do a simple plan. however, there is path if we consider wrap around. like the cspace is block in half.

2.5.2 Real Experiment

Show the time save for real robot operation. I think this will be the talking point to sell our idea to the audience.

2.5.3 Analytical

The redundant tree will have much smaller node in space to be densely filled. maybe reduce computation time and reduce collision check because when we check one space, we get all of them.

2.6 Conclusions

Some conclusions here.

Bibliography for Chapter 2

[1] Howie Choset et al. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005 (cit. on p. 14).

[2] Lydia E Kavraki and Steven M LaValle. "Motion planning". In: *Springer handbook of robotics*. Springer, 2016, pp. 139–162 (cit. on p. 13).

[3] Jean-Claude Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012 (cit. on p. 13).

[4] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006 (cit. on p. 16).

[5] Kevin M Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. English (US). Cambridge Univeristy Press, 2017. ISBN: 978-1107156302 (cit. on p. 11).