Imperial College
London

NOTES

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Advanced Robotics

Author:
W (CID: your college-id number)

Date: March 1, 2020

1 State Representation

A robot state is typically expressed by its position, orientation, and velocity. The robot frame is denoted by S, the cameras are denoted by C_1, C_2 and the world frame is denoted by W.

1.1 Representing Positions

Positions are represented using position vectors.

- ${}_W\boldsymbol{r}_S$ is the position of frame S expressed in frame W.
- ${}_S\boldsymbol{r}_{C_1}$ is the position of camera 1 expressed in frame S.
- ${}_S\boldsymbol{r}_{C_2}$ is the position of camera 2 expressed in frame S.

1.2 Representing Velocities

Velocities are represented by velocity vectors.

- ${}_W\boldsymbol{v}_{WS}$ is the velocity of S with respect to W expressed in frame W. It can also be written as ${}_W\boldsymbol{v}$.

1.3 Representing Orientation

Orientations are represented by rotation matrix. The rotation matrix $\boldsymbol{C}_{WS}$ transforms a coordinate in frame S a coordinate in frame W.

$$ {}_W\boldsymbol{a} = \boldsymbol{C}_{WS}\ {}_S\boldsymbol{a} $$

1.3.1 Euler Angles (Tait-Brian)

According to Euler's rotation theorem, any rotations can be described using three angles: yaw, pitch, and roll.

- The yaw angle (ψ) represents rotation around z-axis: $-\pi < \psi < \pi$
- The pitch angle (θ) represents rotation around y-axis: $-\pi/2 < \theta < \pi/2$
- The roll angle (ϕ) represents rotation around x-axis: $-\pi < \phi < \pi$

If the rotations are represented by the rotation matrix

$$ \boldsymbol{C}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{C}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad \boldsymbol{C}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} $$

then the general rotation matrix can be written as

$$ \boldsymbol{C} = \boldsymbol{C}_z(\psi)\boldsymbol{C}_y(\theta)\boldsymbol{C}_x(\phi) $$

A problem using Euler Angles is **Gimbal Lock Singularity**. When the pitch angle θ equals to $\pi/2$ or $-\pi/2$, rotating the robot around z-axis and x-axis will yield the same result. At this point, the rotation matrix loses one degree of freedom and leads to singularity.

1.3.2 Quaternions

Quaternions provide an alternative measurement technique that does not suffer from gimbal lock. A quaternion is a four-element vector composed of one real element and three complex elements.

$$\boldsymbol{q}_1 = \begin{bmatrix} a & b & c & d \end{bmatrix}^\top = a + bi + cj + dk$$
$$\boldsymbol{q}_2 = \begin{bmatrix} e & f & g & h \end{bmatrix}^\top = e + fi + gj + hk$$

Quaternion Addition

Adding two quaternions simply require adding each component

$$\boldsymbol{q}_1 + \boldsymbol{q}_2 = \begin{bmatrix} a+e & b+f & c+g & d+h \end{bmatrix}^\top$$

Quaternion Multiplication

Multiplying two quaternions is more complicated, the first step is to multiply every element, and then collect the like terms together. Be aware that quaternion multiplication is not commutative, i.e. $ij \neq ji$, so ordering must be preserved when multiplying out the elements.

Multiplication of complex numbers of shown in the table

	i	j	k
i	- 1	k	-j
j	-k	-1	i
k	j	-i	-1

$$\begin{aligned}
\boldsymbol{q}_1 \otimes \boldsymbol{q}_2 &= ae + afi + agj + ahk + bei + bfi^2 + bgij + bhik + \\
&\quad cej + cfji + cgj^2 + chjk + dek + dfki + dgkj + dhk^2 \\
&= ae + afi + agj + ahk + bei - bf + bgk - bh + j \\
&\quad cej - cfk - cg + chi + dek + dfj - dgi - dh \\
&= \begin{bmatrix} ae - bf - cg - dh \\ af + be + ch - dg \\ ag - bh + ce + df \\ ah + bg - cf + de \end{bmatrix}
\end{aligned}$$

Quaternion Inverse

The inverse of a quaternion is equivalent to its conjugate, which means that all the complex elements are negated.

$$\boldsymbol{q}_1^{-1} = \begin{bmatrix} a & -b & -c & -d \end{bmatrix}^\top$$

Quaternion Rotation

Any ration can be achieved by rotating around a vector $\boldsymbol{n}$ by angle α. We can express this using quaternion operation. The first step is to define a quaternion $\boldsymbol{q}$ using $\boldsymbol{n}$ and α.

$$\boldsymbol{q} = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) & \boldsymbol{n}_x \sin\left(\frac{\alpha}{2}\right) & \boldsymbol{n}_y \sin\left(\frac{\alpha}{2}\right) & \boldsymbol{n}_z \sin\left(\frac{\alpha}{2}\right) \end{bmatrix}^\top$$

Be aware that the axis of rotation $\boldsymbol{n}$ is a **unit vector**.

To rotate a vector $\boldsymbol{v}$, we treat the vector like a quaternion with zero real part. And then multiply it by $\boldsymbol{q}$ and $\boldsymbol{q}^{-1}$.

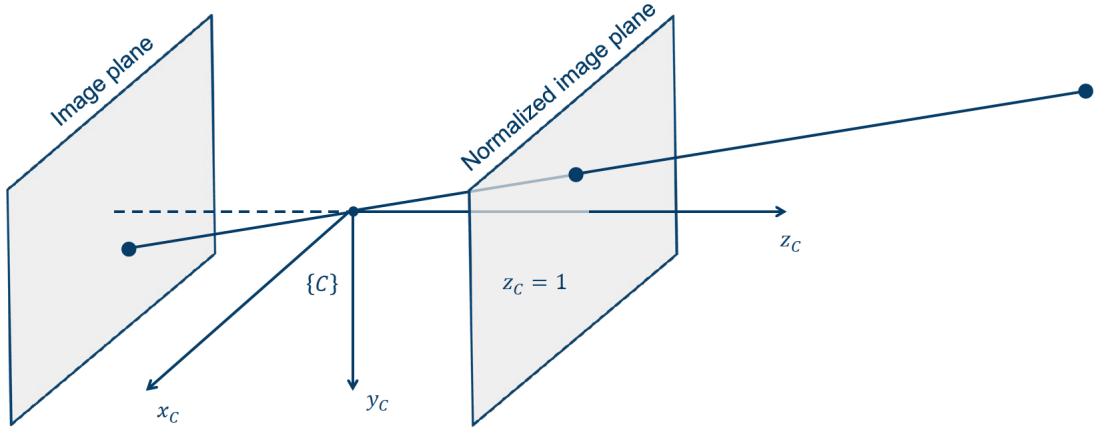
$$\begin{bmatrix} 0 & \boldsymbol{v}'_x & \boldsymbol{v}'_y & \boldsymbol{v}'_z \end{bmatrix}^\top = \boldsymbol{q} \otimes \begin{bmatrix} 0 & \boldsymbol{v}_x & \boldsymbol{v}_y & \boldsymbol{v}_z \end{bmatrix}^\top \otimes \boldsymbol{q}^{-1}$$

$\boldsymbol{v}'$ is the vector after rotation $\boldsymbol{v}$ by angle α around vector $\boldsymbol{n}$

2 Sensor Models

2.1 Camera Projection (World To Camera)

The perspective camera model is a mathematical model describing the correspondence between observed points in the world and pixels in the captured image. The camera is represented by the following coordinate frame.



Projecting a point in the world (x_W, y_W, z_W) to a point of the captured image (u, v) requires three steps:

Projecting onto the Plane

$$\begin{bmatrix} x_C \\ y_C \end{bmatrix} = \begin{bmatrix} x_W / z_W \\ y_W / z_W \end{bmatrix}$$

Applying Distortion Model

The geometry of the perspective camera is simple since we assume the pinhole to be infinitely small. In reality the light passes through a lens that complicates the camera intrinsics. Many wide-angle lenses have noticeable radial distortion which basically means that lines in the scene appear as curves in the image.

Radial distortion can be described using simple polynomial model, to reduce the geometric error introduced by the lens. A simple radial distortion model is something like

$$\hat{x}_C = x_C\left(1 + k_1\left(x_C^2 + y_C^2\right) + k_2\left(x_C^2 + y_C^2\right)^2\right)$$
$$\hat{y}_C = y_C\left(1 + k_1\left(x_C^2 + y_C^2\right) + k_2\left(x_C^2 + y_C^2\right)^2\right)$$

Where k_1 and k_2 are parameters for the distortion model.

Calibrating the Camera

The camera calibration matrix K has 5 parameters.

$$K = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}$$

Where (c_u, c_v) is the image centre in pixels.

Where f_u and f_v are the focal length in the x and y directions respectively in pixels.

Where s is a skew parameter. However, in modern cameras this is often ignored.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_u & 0 & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \hat{x}_C \\ \hat{y}_C \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix}\begin{bmatrix} \hat{x}_C \\ \hat{y}_C \end{bmatrix} + \begin{bmatrix} c_u \\ c_v \end{bmatrix}$$

2.2 Camera Un-Projection (Camera To World)

This operation inverts the previous model to find a **ray** from image pixel. The actual 3D point cannot be found because some information is loss during projection.

The first step is to inverse camera calibration.

$$\begin{bmatrix} \hat{x}_C \\ \hat{y}_C \end{bmatrix} = \begin{bmatrix} 1/f_u & 0 \\ 0 & 1/f_v \end{bmatrix}\begin{bmatrix} u - c_u \\ v - c_v \end{bmatrix}$$

The second step is to inverse the distortion model.The step is usually done numerically.

$$\begin{bmatrix} x_C \\ y_C \end{bmatrix} = d^{-1}\left(\begin{bmatrix} \hat{x}_C \\ \hat{y}_C \end{bmatrix}\right)$$

The last step is to add an extra dimension to make it a ray.

$$\begin{bmatrix} x_C & y_C & 1 \end{bmatrix}^\top$$

3 Robot Dynamics

Robot Kinematics describes the motion of points, bodies, and systems of bodies without considering the cause of motion. It is often referred to as the geometry of motion.

Robot Dynamics is the study of forces that is responsible for the motion. We can model the robot dynamics as a system that takes in input $\boldsymbol{u}$ and outputs a state vector $\boldsymbol{x}$. State $\boldsymbol{x}$ can be described by some sensor outputs $\boldsymbol{z} = h(\boldsymbol{x})$

3.1 Continuous-time Nonlinear System

The evolution of state $\boldsymbol{x}$ is described by a system of differential equations.

$$\dot{\boldsymbol{x}}(t) \doteq \frac{\partial \boldsymbol{x}(t)}{\partial t} = \boldsymbol{f}_c(\boldsymbol{x}(t),\, \boldsymbol{u}(t),\, \boldsymbol{w}(t))$$
$$\boldsymbol{z}(t) = h(\boldsymbol{x}(t)) + \boldsymbol{v}(t)$$

Where $\boldsymbol{w}(t)$ models the process noise and $\boldsymbol{v}(t)$ models the measurement noise.

For example in the following system, the change in component x_1 in $\boldsymbol{x}$ depends on the x_2 component of the previous state. The change in component x_2 depends on x_2 and also input u_1.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ k_1 \sin x_2 + u_1 \end{bmatrix}$$

3.1.1 Linearisation

Linearisation is a linear approximation of nonlinear system in a small region. In other words, it aims to find the tangent line around an operation point $\bar{x}$ of a non-linear function f.

Linearisation can be handy for analysis.

Performing linearisation on a system results in a **Jacobian Matrix**

$$\boldsymbol{F} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \qquad \boldsymbol{G} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_n} \end{bmatrix}$$

Using these Jacobian Matrices, we can rewrite the system of differential equations as **Linear Time Invariant Systems**

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{F}\boldsymbol{x}(t) + \boldsymbol{G}\boldsymbol{u}(t) + \boldsymbol{L}\boldsymbol{w}(t)$$
$$\boldsymbol{z}(t) = \boldsymbol{H}\boldsymbol{x}(t) + \boldsymbol{v}(t)$$

In the above example

$$\boldsymbol{F} = \begin{bmatrix} 0 & 1 \\ 0 & k_1 \cos x_2 \end{bmatrix} \qquad \boldsymbol{G} = \begin{bmatrix} 0 \\ k_2 \end{bmatrix}$$

3.1.2 Discretise

To obtain the state transition function for a time interval Δt, we need to perform integration. However, nonlinear differential equations typically can't be integrated analytically. Instead, we can discretise the continuous function and represent the state at time t by

$$\boldsymbol{x}(t) = \boldsymbol{x}(t-1) + \Delta t \boldsymbol{f}\left(\boldsymbol{x}(t-1),\, \boldsymbol{u}(t-1)\right)$$
$$\boldsymbol{z}(t) = \boldsymbol{h}\left(\boldsymbol{x}(t)\right) + \boldsymbol{v}(t)$$

4 Kalman Filter

Kalman Filter are used on systems that are continuously changing, it makes an educated guess about the robot. state.

4.1 Problem

Assume a robot has state $\boldsymbol{x}_k = (p, v)$ which contains a position and a velocity.

The robot has a GPS **sensor** that gives a rough estimate of the robot's position.

The robot also knows knows the commands sent to the wheel motors and can make a **prediction** of the next state.

Both the sensor and prediction gives a rough estimation of the robot's state, however both of them contains noise that will affect the accuracy of the estimation. Kalman Filter combines all the available information to compute the best guess of the robot state.

4.2 Prediction

At anytime k, Kalman Filter models the state as a Gaussian distribution with mean $\hat{\boldsymbol{x}}_k$ and covariance matrix $\boldsymbol{P}_k$.

$$\hat{\boldsymbol{x}}_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} \qquad\qquad \boldsymbol{P}_k = \begin{bmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{pv} & \Sigma_{vv} \end{bmatrix}$$

We can predict the next state $\hat{\boldsymbol{x}}_k$ from the current state $\hat{\boldsymbol{x}}_{k-1}$ using a system matrix $\boldsymbol{F}$. The system matrix takes every point in the original estimation to the new predicted location.

The robot also knows the commands $\boldsymbol{u}_k$ sent to the motors, it can take this commands into consideration to increase the accuracy of the prediction. The Gain matrix $\boldsymbol{G}$ is used to describe how a command affects the state of the robot.

Finally, there are external noises that might affect the state. If we?re tracking a wheeled robot, the wheels could slip, or bumps on the ground could slow it down.

To encapsulate these noises, we assume that every point in the current state $\hat{\boldsymbol{x}}_{k-1}$ will be moved in a Gaussian blob with covariance $\boldsymbol{Q}_k$.

Putting everything together, the new mean and covariance of the next state is

$$\begin{aligned}\hat{\boldsymbol{x}_k} &= \boldsymbol{F}_k\hat{\boldsymbol{x}}_{k-1} + \boldsymbol{G}_k\boldsymbol{u}_k \\ \boldsymbol{P}_k &= \boldsymbol{F}_k\boldsymbol{P}_{k-1}\boldsymbol{F}_k^\top + \boldsymbol{Q}_k\end{aligned}$$

4.3 Update

Sensor readings also give us information about the robot state. A robot at state $\boldsymbol{x}_k$ will give a sensor reading $\boldsymbol{z}_k$ with some noise. We will model this sensor reading with a Gaussian distribution with mean $\boldsymbol{z}_k$ and noise covariance $\boldsymbol{R}_k$.

$$\mathcal{N}(\boldsymbol{z}_k,\ \boldsymbol{R}_k)$$

The goal of Kalman Filter is to combine the knowledge of the **predicted state** and sensor reading to get the most likely state of the robot. To do this, we first need to transform the predicted state to sensor reading space using transformation matrix $\boldsymbol{H}_k$.

$$\mathcal{N}\left(\boldsymbol{H}_k\hat{\boldsymbol{x}}_k,\ \boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^\top\right)$$

Now we have two Gaussian distributions that predicts the robot state. One around the sensor reading, another around the predicted state. Given two Gaussian distributions $\mathcal{N}(\mu_0,\ \Sigma_0)$ and $\mathcal{N}(\mu_1,\ \Sigma_1)$, the joint distribution is $\mathcal{N}(\mu',\ \Sigma')$, where

$$\begin{aligned}\mu' &= \mu_0 + K(\mu_1 - \mu_0) \\ \Sigma' &= \Sigma_0 - K\Sigma_0 \\ K &= \Sigma_0(\Sigma_0 + \Sigma_1)^{-1}\end{aligned}$$

Applying the equations to our two Gaussian blobs,

$$\begin{aligned}\boldsymbol{H}_k\hat{\boldsymbol{x}}_k' &= \boldsymbol{H}_k\hat{\boldsymbol{x}}_k + \boldsymbol{K}\left(\boldsymbol{z}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k\right) \\ \boldsymbol{H}_k\boldsymbol{P}_k'\boldsymbol{H}_k^\top &= \boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^\top - \boldsymbol{K}\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^\top \\ \boldsymbol{K} &= \boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^\top\left(\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^\top + \boldsymbol{R}_k\right)^{-1}\end{aligned}$$

We can simplify the equations by removing $\boldsymbol{H}_k$ in front of each term.

$$\begin{aligned}\hat{\boldsymbol{x}}_k' &= \hat{\boldsymbol{x}}_k + \boldsymbol{K}\left(\boldsymbol{z}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k\right) \\ \boldsymbol{P}_k' &= \boldsymbol{P}_k - \boldsymbol{K}\boldsymbol{H}_k\boldsymbol{P}_k \\ \boldsymbol{K} &= \boldsymbol{P}_k\boldsymbol{H}_k^\top\left(\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^\top + \boldsymbol{R}_k\right)^{-1}\end{aligned}$$

$\boldsymbol{K}$ is called the Kalman gain. $\hat{\boldsymbol{x}}_k'$ and $\boldsymbol{P}_k'$ is the best predicted mean and covariance of the robot state. We can apply this update step every n iterations to predict the state of the robot.

5 Feedback Controller

An open-loop control system takes input under the consideration and doesn't react on the feedback to obtain the output. A closed loop system is also referred as a feedback control system. These systems record the output instead of input and modify it according to the need.

5.1 PID Controller

The purpose of a PID controller is to force feedback to match a setpoint. PID controller maintains the output such that there is zero error between process variable and set point/desired output by closed loop operations.

The Proportional Controller gives output which is proportional to current error. The resulting error is multiplied with proportional constant k_p to get the output. Speed of the response is increased when the proportional constant increases. However, using Proportional Controller alone requires biasing because it never reaches the steady state condition.

Due to the limitation of Proportional Controller where there always exists an offset between the process variable and set point, the Integral Controller is used to eliminate the steady state error. It integrates the error over a period of time until error value reaches to zero. Decreasing the integral constant k_i decreases the steady state error.

Derivative Controller predicts the future behavior of the error. Its output depends on rate of change of error with respect to time, multiplied by derivative constant k_d. Increasing the derivative gain increases speed of response.

6 Nonlinear Least Square

In Nonlinear least square problem, the aim is to find an $\boldsymbol{x} \in \mathbb{R}^N$ that minimise

$$f(\boldsymbol{x}) = \sum_{m=0}^{M} r_m(\boldsymbol{x})^2$$

Where $r : \mathbb{R}^N \to \mathbb{R}^M$ is a residual function. In the linear case,

$$r(\boldsymbol{x}) = \boldsymbol{z} - \boldsymbol{h}(\boldsymbol{x})$$

6.1 Gauss Newton Method

One way of solving the nonlinear least square problem is using Gaussian Newton Method. In Gauss Newton Method:

1. Initialise a starting guess $\boldsymbol{x}_0$.

2. Update guess $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \nabla^2 f(\boldsymbol{x})^{-1} \nabla f(\boldsymbol{x})$.
3. Repeat step 2 until $\boldsymbol{x}$ converges.

The Gradient and Hessian of $f(\boldsymbol{x})$ can be expressed in terms of Jacobian.

$$\begin{aligned}
\nabla f(\boldsymbol{x}) &= \sum_{m=1}^{M} r_m(\boldsymbol{x}) \nabla r_m(\boldsymbol{x}) = J(\boldsymbol{x})^\top r(\boldsymbol{x}) \\
\nabla^2 f(\boldsymbol{x}) &= \sum_{m=1}^{M} \nabla r_m(\boldsymbol{x}) \nabla r_m(\boldsymbol{x})^\top + \sum_{m=1}^{M} r_m(\boldsymbol{x}) \nabla^2 r_m(\boldsymbol{x}) \\
&= J(\boldsymbol{x})^\top J(\boldsymbol{x}) + \sum_{m=1}^{M} r_m(\boldsymbol{x}) \nabla^2 r_m(\boldsymbol{x})
\end{aligned}$$

Where

$$J(\boldsymbol{x}) = \begin{bmatrix} \nabla r_1(\boldsymbol{x})^\top \\ \nabla r_2(\boldsymbol{x})^\top \\ \vdots \\ \nabla r_m(\boldsymbol{x})^\top \end{bmatrix}$$

In Gauss Newton Method, is assumes the second term in $\nabla^2 f(\boldsymbol{x})$ is 0, and therefore approximates the second derivative as

$$\nabla^2 f(\boldsymbol{x}) \approx J(\boldsymbol{x})^\top J(\boldsymbol{x})$$

In matrix form, the update rule in step 2 becomes

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k - \left(J(\boldsymbol{x})^\top J(\boldsymbol{x})\right)^{-1} J(\boldsymbol{x})^\top r(\boldsymbol{x}) \\
\left(J(\boldsymbol{x})^\top J(\boldsymbol{x})\right)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) &= -J(\boldsymbol{x})^\top r(\boldsymbol{x}) \\
\left(J(\boldsymbol{x})^\top J(\boldsymbol{x})\right)\Delta\boldsymbol{x} &= -J(\boldsymbol{x})^\top r(\boldsymbol{x})
\end{aligned}$$

We can solve for $\Delta\boldsymbol{x}$ using Cholesky Decomposition and update the current prediction until it converges.

6.2 Levenberg-Marquardt

Levenberg-Marquardt is another optimization method that combines Gauss Newton and Gradient Descent. In Levenberg-Marquardt method

1. Initialise starting guess $\boldsymbol{x}_0$
2. Update guess $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left(\nabla^2 f(\boldsymbol{x}) + \mu_k \boldsymbol{I}\right)^{-1} \nabla f(\boldsymbol{x})$
3. Repeat step 2 until $\boldsymbol{x}$ converges

When μ_k approaches ∞ it is similar to gradient descent with a small step size. When μ_k approaches 0, it is the same as Gauss Newton Method. In practice, the method starts with a small μ and increases gradually until a descent condition is satisfy.

7 Simultaneous Localisation and Mapping

Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. SLAM requires solving multiple challenges including:

- Extracting Useful Features from Camera
- The PnP Problem
- Reconstructing the 3D map
- Place Recognition

7.1 Feature Extraction

Feature detection and matching are an essential component of many computer vision applications. The most common approach is to extract corners as interest points in the image.

7.1.1 Harris Detector

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image.

1. Compute x and y derivatives of an image, ∇f_x and ∇f_y
2. At each pixel compute the matrix

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} (\nabla f_x)^2 & \nabla f_x \nabla f_y \\ \nabla f_x \nabla f_y & (\nabla f_x)^2 \end{bmatrix}$$

3. Perform eigen decomposition on M to find eigen values λ_1 and λ_2
4. Compute response $R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$
5. Compute threshold for R and perform non maximum suppression

7.1.2 Multiple Scale Harris Detector

When computing M, we find the scale σ which gives the largest corner response

$$M = \sum_{x,y} w(x,y)\sigma^2 \begin{bmatrix} (\nabla f_x)^2 & \nabla f_x \nabla f_y \\ \nabla f_x \nabla f_y & (\nabla f_x)^2 \end{bmatrix}$$

7.1.3 Scale-Invariant Feature Transform (SIFT)

There are mainly four steps involved in the SIFT algorithm.

1. Scale-space peak selection: Potential location for finding features.
2. Keypoint Localization: Accurately locating the feature keypoints.
3. Orientation Assignment: Assigning orientation to keypoints.
4. Keypoint descriptor: Describing the keypoints as a high dimensional vector.

7.2 PnP Problem

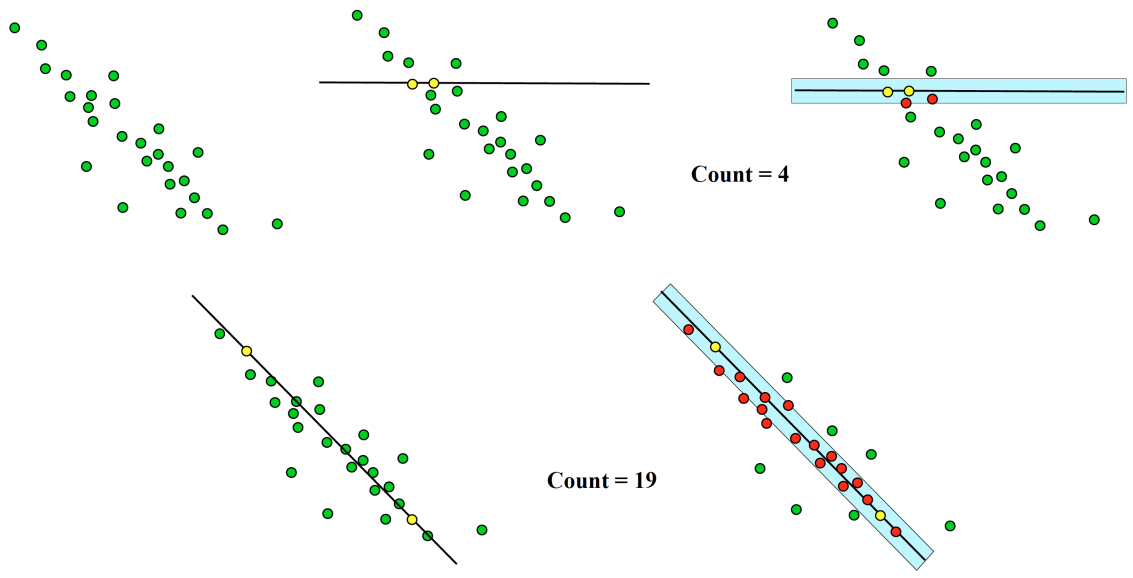
Perspective-n-Point is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. The camera pose consists of 6 degrees-of-freedom (DOF) which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world.

One way to solve the problem is to project 3D to 2D plane and solve for nonlinear least square estimation. i.e. minimising the errors between all projected points and their actual 2D location. However, nonlinear least square estimation is sensitive to outliers, instead we want to use a method that is robust to outliers.

7.2.1 RANdom SAmpling Consensus (RANSAC)

RANSAC performs estimation in a two step process

1. Classifier data points as inliers and outliers
2. Fit model to inliers and ignore outliers.



```
Determine:
    s — the smallest number of points required
    N — the number of iterations required
    d — the threshold used to identify a point that fits well
    T — the number of nearby points required
      to assert a model fits well
Until N iterations have occurred
    Draw a sample of s points from the data
      uniformly and at random
    Fit to that set of s points
    For each data point outside the sample
      Test the distance from the point to the line
        against d if the distance from the point to the line
        is less than d the point is close
    end
    If there are T or more points close to the line
      then there is a good fit. Refit the line using all
      these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion
```

7.3 Reconstructing the 3D Map

Given the corresponding 2D points in two different views, construct the point on the 3D map. Given the same 2D point from two different views, we can obtain two rays. In reality, the two rays will not intersect, and so we have find the points along the ray with minimum distance to each-other.

7.4 Place Recognition

The algorithm should be able to detect regions it has previously seen. The most common approach is using Bag of Visual Words.

7.4.1 Bag of Visual Words

Bag of visual words (BOVW) is commonly used in image classification. The general idea of bag of visual words (BOVW) is to represent an image as a set of features. We use the keypoints and descriptors to construct vocabularies and represent each image as a frequency histogram of features that are in the image.

During Run time, we can compute histogram similiarity score for each location in the database. If we found a match, apply individual keypoints matching and verify geometrically using some RANSAC otherwise we insert the place to the database.