1 Motion

Two **coord frames**: **World** W (3D), **Robot** R carried by and stays fixed to robot (2D). Its location is the transformation $W \to R$.
The **degrees of freedom** v is the number of dimensions needed to represent motion:

- Translate along 1D path: $v = 1$.
- Translate and rotates on 2D plane: $v = 3 \quad (x, y, \theta)$.
- Translate and rotate in 3D space: $v = 6 \quad (x, y, z, \text{roll}, \text{pitch}, \text{yaw})$.

A **holonomic** robot can move instantaneously in any direction in the space of its DOG. A **non-holonomic** has constraints on motion.

1.1 Differential Drive

Differential Drive: two driving wheels on each side, with their own motor. Steering achieved by varying the relative speeds of the two wheels. Wheels at equal speeds run in a straight line, in opposite directions to turn on the spot, and different speeds to move in some arc with radius R and angle $\Delta\theta$. We also know the distance W between the two wheels:
- Left wheel has arc radius $R - \frac{W}{2}$.
- Right wheel has arc radius $R + \frac{W}{2}$.

Then we can find the angle as:

$$\Delta\theta = \frac{v_L \Delta t}{R - \frac{W}{2}} = \frac{v_R \Delta t}{R + \frac{W}{2}} = \frac{(v_R - v_L)\Delta t}{W}$$

$$R = \frac{W(v_R + v_L)}{2(v_R - v_L)}$$

1.2 Drive & Steer

Two motors, *one to drive*, *one to steer*. Cannot turn on the spot. Fixed speed and steering angle will follow a circular path. With four wheels, we need a **rear differential** and **variable (Ackerman) linkage** for steering. For a tricycle, assuming no sideways slip, where:

- L is the distance from the rear wheel to the front wheels.
- s is the steering angle of the front wheels.
- R is the turning radius of the robot.
- R_d is the turning radius of the rear wheel (slightly different from R).

$$R = \frac{L}{\tan s} \qquad R_d = \frac{L}{\sin s} \qquad \Delta\theta = \frac{v \Delta t \sin s}{L}$$

1.3 Wheel Velocity

The rate of rotation of a wheel (**angular velocity** ω) gives the **linear velocity** $v = r_w \omega$. DC motors set the desired angular rate by applying a specific voltage. They use encoders (sensors) to measure and control the angular position and velocity. The motor uses **pulse width modulation** to control the voltage supplied to the motor. This is done by switching the supply on and off at a high frequency.

- Error $e(t)$ is the demand minus the actual position/velocity.
- **Proportional Integral Differential (PID) Control** sets power as a function of error, $P(t) = k_p e(t) + k_i \int_{t_0}^{t} e(\tau)\mathrm{d}\tau + k_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}$.
- k_p **main gain constant** gives rapid response but possible oscillation.
- k_i **integral gain constant** eliminates steady state error but may cause overshoot.
- k_d **differential gain constant** damps oscillations and reduces settling time.

1.4 Planar Motion

On a 2D plane, we can specify its location state $\vec{x} = \begin{pmatrix} x & y & \theta \end{pmatrix}^T$.

- If we move in a straight line with distance d then $x' = \begin{pmatrix} x + d\cos\theta & y + d\sin\theta & \theta \end{pmatrix}^T$.
- If we rotate by angle α then $x' = \begin{pmatrix} x & y & \theta + \alpha \end{pmatrix}^T$.

More generally, if we move along an arc of radius R and angle $\Delta\theta$ then:

$$x' = \begin{pmatrix} x + R(\sin(\theta + \Delta\theta) - \sin\theta) \\ y - R(\cos(\theta + \Delta\theta) - \cos\theta) \\ \theta + \Delta\theta \end{pmatrix}$$

1.5 Path Planning

Position based planning: Assuming a robot has *localisation* (i.e. it knows its current state $\vec{x}$), we plan a path by first rotating to face the goal, then moving in a straight line to it.
Local planning considers robot dynamics and possible changes in motion it can make within a small time $\mathrm{d}t$. For each possible postions look ahead for a longer time τ. Calcualte the best trajectory based on distance from target and obstacles.
Global planning uses algorithms to determine the best path from start to goal, considering obstacles. Common algorithms include A*, Dijkstra's, and RRT (Rapidly-exploring Random Tree).

2 Sensors

Sensors can be **proprioceptive** (*self-sensing*) or **exteroceptive** (*outward looking*). Proprioceptive measure internal states as a function of the robot's state, $z_p = z_p(x)$. It may also depend on the *state history* (x_i, x_{i-1}) or *rate of change* $\dot{x}$. Exteroceptive sensors depend on the state of the robot and the world around it, $z_o = z_o(x, y)$.

2.1 Servoing

Servoing: control parameters (e.g. motor speed) are coupled directly to a asensor reading and updated regularly in a *negative feedback loop*. Also known as **closed loop control**, this needs high frequency sensor updates (or motion may oscillate). A control demand is set which over time aims to bring the currect value of a sesnor reading into agreement with a desired value.
With **proportional control**, we set the demand proportional to the negative error, e.g. $v = -k_p(z_{\text{desired}} - z_{\text{actual}})$, where $k + p$ is the **proportional gain constant**. *This is a special case of PID control.*

2.2 Control Steering

For a tricycle or car-type wheel configuration, we can use a **steering law** to guide robots to avoid obstacles at a safe radius:

$$s = k_p(\alpha - \sin^{-1}\frac{R}{D})$$

Where s is the steering demand, α is the robot's current heading, R is the desired turning radius and D is the distance to the nearest obstacle.

2.3 Wall Following

Here, we use a *sideways-looking sonar* to measure distance z to the wall. To maintain a desired distance d, we set the difference to the left and right wheel velocities proportional to the error: $v_R - v_L = k_p(z - d)$. Symmetric behaviour is achieved with a constant offset v_C:

$$v_R = v_C + \tfrac{1}{2}k_p(z - d)$$

$$v_L = v_C - \tfrac{1}{2}k_p(z - d)$$

2.4 Probabilistic Sensor Modelling

Sensors are *uncertain*. By characterizing a sensor we can build a probabilistic model for how it works, $p(z_o|x, y)$. This is often **gaussian**, $p(z|m) \propto e^{-\frac{(z-m)^2}{2\sigma_s^2}} + K$, where m is the true measurement and σ_s is the standard deviation of the noise and K is a small constant. To mitigate errors due to noise, we can do:

- **Temporal Filtering**: smoothing /finding median of the last few measurements of a sensor. Good at reducing the effect of occasional large **outliers**.
- **Geometric Fitting** to data from a sensor which reports an array of measurements, where we might fit geometric shapes such as straight lines or corners to the measurements and output the parameters of those shapes rather than the raw measurements.

3 Probabilistic Robots

After calibration the robot should *on average* return to the desired location, but will scatter due to uncontrollable factors. These are **zero mean errors**, occurring incrementally, the size of the distribution growing with distance travelled. This can be modelled as a **gaussian distribution**. In reality, we have uncertain terms e, f and g with a zero-mean and a Gaussian distribution:

$$\begin{pmatrix} x_{\text{new}} \\ y_{\text{new}} \\ \theta_{\text{new}} \end{pmatrix} = \begin{pmatrix} x + (D + e)\cos(\theta) \\ y + (D + e)\sin(\theta) \\ \theta + f \end{pmatrix}$$

$$\begin{pmatrix} x_{\text{new}} \\ y_{\text{new}} \\ \theta_{\text{new}} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$$

Simple sensing / actions procedures are locally effective but limited in complex problems in the real-world. We can build incrementally updating **probabilistic models** to estimate the position of our robot on the map.
Every action & sensor measurement is **uncertain**. When estimating a robot's state, we use these, so the state estimate is also uncertain. Usually, we take an uncertain measurement and then take action to receive new information. This then updates the estimate.

3.1 Bayesian Probabilistic Inference

Prior knowledge combined with new measurements is generally modelled as a **Bayesian Network** - a series of weighted combinations of old & new information.
Sensor fusion is the process of combining multiple uncertain measurements to produce a better estimate.
Baysian is the *measure of subjective belief*. Probabilties describe our state of knowledge. The **Bayes' Rule** relates probabilities of discrete statements: $P(X|Z) = \frac{P(Z|X)P(X)}{P(Z)}$. Here, $P(X)$ is the **prior** probability of X before observing Z. $P(Z|X)$ is the **likelihood** of observing Z given X. $P(Z)$ is the **marginal likelihood**. $P(X|Z)$ is the **posterior** probability of X after observing Z.
We use Bayes' rule to incrementally *digest* new information from sensors about a robot's state.

3.2 Probability Distributions

Discrete probabilistic inference generalizes to large numbers of possible states, so we can use a continuous **PDF**: $P\, a \to b = \int_a^b p(x)dx$.
A **gaussian** distribution often represents uncertainty in measurements well $p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, where μ is the mean and σ is the standard deviation.
The *prior* (wide Gaussian) is updated with a *likelihood* (narrow Gaussian) to produce a *posterior* (narrower Gaussian).

3.3 Particle Models

Here, a probability distribution is represented by a finite set of *weighted samples* of the state $\{x_i, w_i\}$, where $\sum_i w_i = 1$. These are simple and can represent *multimodal distributions*.

4 Monte Carlo Localisation

In MCL, a cloud of weighted particles represents the uncertain position of a robot. A particle is a point estimate x_i of the state of a robot with a weight w_i, $x_i = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix}$. The full particle set is $\{(x_i, w_i)\}_{i=1}^{N}$. The distribution is **noramlized** if $\sum_{i=1}^{N} w_i = 1$.

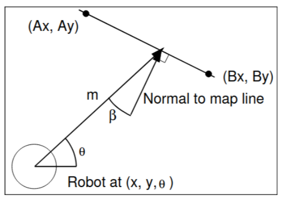
- **Continuous Localisation**: a tracking problem - given a good estimate of where the robot was at the last timestep, and some new measurements, estimate where it is now. Here, the initial position is known, so we start with $x_1 = x_2 = \cdots = x_N = x_{\text{init}}$ and $w_1 = w_2 = \cdots = w_N = \frac{1}{N}$.
- **Global Localisation**: the environment is known, but the robot's position ins completely uncertain. Here the initial particles are sampled randomly from all the positions, so $x_i \sim \text{Uniform(all positions)}$ and $w_1 = w_2 = \cdots = w_N = \frac{1}{N}$.

If we need to, we can make a point estimate of the robot's position from the particles, e.g. the weighted mean: $\bar{x} = \sum_{i=1}^{N} w_i x_i$. So, for MCL:

1. *Motion is predicted* based on **odometry**.
2. *Measurements updated* based on **sensors**.
3. **Normalization**, then **resampling**.

4.1 (1) Measurement Update

A measuerement update consists of applying **Bayes Rule** $P(X|Z) = \frac{P(Z|X)P(X)}{P(Z)}$ to each particle: $w_i' = w_i \cdot P(z|x_i)$ (*as $P(Z)$ is a constant later removed by normalization*) where $P(z|x_i)$ is the likelihood of the measurement z given the particle state x_i.
A **likelihood function** $p(z|v)$ fully describes the sensors performance, determined by repeated experiments with the sensor where z is the measurement and v is the *ground truth*.



If the robot is at (x, y, θ) then its forward distance to an infinite wall passing through (A_x, A_y) and (B_x, B_y) is given by:

$$m = \frac{(B_y - A_y)(A_x - x) - (B_x - A_x)(A_y - y)}{(B_y - A_y)\cos\theta - (B_x - A_x)\sin\theta}$$

Then the world coordinates at which the forward vector from the robot would meet the wall are:

$$\begin{pmatrix} x + m\cos\theta \\ y + m\sin\theta \end{pmatrix}$$

The likelihood of a sonar update depends on the difference $z - m$ where z is the sonar measurement and m is the expected measurement from the particle. A common model is a Gaussian:

$$p(z|m) \propto \exp\left(-\frac{(z-m)^2}{2\sigma_s^2}\right)$$

Where σ_s is the standard deviation of the sonar noise.

A **robust likelihood** function takes into account that real sensors sometimes report *garbage* values values which are not close to the ground truth:

$$p(z|m) \propto \exp\left(-\frac{(z-m)^2}{2\sigma_s^2}\right) + K$$

Possibly relevant for real sensors may be the angle between sonar direction and the normal to the wall. If β is too large, sonar will not give a sensible reading:

$$\beta = \cos^{-1}\left(\frac{\cos\theta(A_y - B_y) + \sin\theta(B_x - A_x)}{\sqrt{(A_y - B_y)^2 + (B_x - A_x)^2}}\right)$$

4.2 (2) Normalization

The weights of all particles are scaled again so that they add to 1:

$$w_i' = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

4.3 (3) Resampling

Generate a new set of N particles which all have equal weights $\frac{1}{N}$, but whose spatial distribution now reflects the probability density.

Done by generating the *cumulative probability distribution*, generating a *uniformly distributed random number* between 0 and 1 and picking the particle whose cumulative probability this intersects.

5 Sonar Measurements

Recognition learn characteristics of many chosen locations on the map, then uses measurements to recognize which location we are at. This is more robust to noise, but requires more computation and training data.

1. Place robot in each target location, taking raw measurements (called a **signature**).
2. Afterward, to decide on the location it must compare histograms with a *correlation test*. The square difference D_k between new measurement histogram $H_m(i)$ and saved signature histogram $H_k(i)$ is: $D_k = \sum_i (H_m(i) - H_k(i))^2$. The location with the smallest D_k is the most likely location. A **threshold** is used to decide if the recognition is good enough to be used as a measurement update in MCL.

If the test histogram can be brought into close agreement by only a shift, then the robot is in the same place but rotated. To save computation, we can build a signature invariant to robot's rotation.

5.1 Grid Mapping

Probablistic Occupancy Grid Mapping infers which parts of the environment are navigable free space, and which contain obstacles. An **occupancy grid** accumulates uncertain information from sensors into precise maps. For each cell i:

- $P(O_i)$ fis the probability that cell i is occupied.
- $P(E_i)$ is the probability that cell i is free.
- $P(O_i) + P(E_i) = 1$.
- Initialize probabilities to 0.5 (unknown).

On sonar measurement Z (reporting depth $Z = d$), we get evidence for cells distance d in front of the root are more likely to be occupied, and cells closer than d are more likely to be free. We can update the probabilities using **Bayes Rule** $P(O_i|Z) = \frac{P(Z|O_i)P(O_i)}{P(Z)}$. We dont need to calculate $P(Z)$ as we could calculate $P(E_i|Z) = \frac{P(Z|E_i)P(E_i)}{P(Z)}$ and then normalize the probabilities. If we take the ratio of the probabilities, and use odds notation $o(A) = \frac{P(A)}{P(\bar{A})}$, we get:

$$\left(\frac{P(O_i|Z)}{P(E_i|Z)}\right) = \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) * \left(\frac{P(O_i)}{P(E_i)}\right)$$

$$o(O_i|Z) = \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) * o(O_i)$$

$$\ln o(O_i|Z) = \ln\left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) + \ln o(O_i)$$

In this form, each cell stores $\ln o(O_i)$, and is updated additively. Cells with probability 0.5 will have log odds 0 a positive log means probability > 0.5, and a negative log means probability < 0.5.

To model the sensor's likelihood function, we consider the ratio of likelihoods we need to update log odds:

- The log odds update $U = \ln \frac{P(Z|O_i)}{P(Z|E_i)}$.
- For cells within the sonar beam but closer than the measured depth $z = d$, $\frac{P(Z|O_i)}{P(Z|E_i)} < 1$ so we can choose a constant negative value for U.
- For cells within the sonar around $Z = d$, $\frac{P(Z|O_i)}{P(Z|E_i)} > 1$ so we can choose a constant positive value for U.

6 Camera Measurements

A camera is a **projective sensor**, so each pixel measures light intensity and colour arriving from a specific direction through the camera centre. *It does not measure depth, so a single image cannot distinguish between small and close and large and far objects without additional information.* Hence, geometric reconstruction requires either **prior knowledge** about the scene or **multiple viewpoints**. If we know the camera is observing a **ground plane**, each image pixel corresponds to a specific 3D point on that plane. This extra structure allows us to recover geometric information from a single image. The mapping depends on **intrinsic parameters** (*focal length & principal point*) and **extrinsic parameters** (*camera position & orientation*). The vector from the camera centre to a ground point, expressed in the camera frame C, is:

$$\mathbf{c}_C = R_{CR}(\mathbf{r}_R - \mathbf{t}_R)$$

Where R_{CR} is the rotation from robot frame R to camera frame C, $\mathbf{t}_R$ is the translation from robot centre to camera centre, and $\mathbf{r}_R$ is the ground point in robot coordinates.

A perspective camera projects a 3D point into image coordinates using the calibration matrix K:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \propto K\mathbf{c}_C$$

$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$

The vector is written in **homogeneous ciirds**, where the actual image coords are obtained by dividing by the 3rd component. This simplifies projective transformations using matmul.

6.1 Ground Plane Homography

If the point lies on the ground plane:

$$\mathbf{r}_R = \begin{pmatrix} x_R \\ y_R \\ 0 \end{pmatrix} \implies \mathbf{r}_R - \mathbf{t}_R = \begin{pmatrix} x_R - t_{Rx} \\ y_R - t_{Ry} \\ -t_{Rz} \end{pmatrix} = T\begin{pmatrix} x_R \\ y_R \\ 1 \end{pmatrix}$$

$$\text{where } T = \begin{bmatrix} 1 & 0 & -t_{Rx} \\ 0 & 1 & -t_{Ry} \\ 0 & 0 & -t_{Rz} \end{bmatrix}$$

Combining everything: $\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \propto KR_{CR}T\begin{pmatrix} x_R \\ y_R \\ 1 \end{pmatrix}$.

We define a single matrix $H = KR_{CR}T$ where H is the **ground plane homography**, a 3×3 matrix that maps ground coordinates to image coordinates. Thus:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \propto H\begin{pmatrix} x_R \\ y_R \\ 1 \end{pmatrix}$$

Instead of separately estimating intrinsic and extrinsic parameters, we can estimate the homography H directly from point correspondences. This requires at least four known points on the ground plane and their corresponding image coordinates. Using more than four points improves accuracy through least-squares estimation. Since homographies are defined up to scale, we write:

$$H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

For one correspondence $(x, y) \to (u, v)$:

$$\begin{pmatrix} ku \\ kv \\ k \end{pmatrix} = \begin{pmatrix} ax + by + c \\ dx + ey + f \\ gx + hy + 1 \end{pmatrix}$$

From rearranging:

$$(gx + hy + 1)u = ax + by + c$$
$$(gx + hy + 1)v = dx + ey + f$$

Each point provides two linear equations. With four correspondences, we obtain eight equations to solve for $a \ldots h$ using least squares. The system can be written in matrix form $A\mathbf{h} = \mathbf{b}$ where $\mathbf{h} = [a\ b\ c\ d\ e\ f\ g\ h]^T$. This can be solved using least squares (e.g., `np.linalg.lstsq()`). Once H is estimated, we can invert it to recover ground coordinates from image coordinates:

$$\begin{pmatrix} x_R \\ y_R \\ 1 \end{pmatrix} \propto H^{-1}\begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

This is only valid for points known to lie on the ground plane.

6.2 Practical Applications

One application is detecting point-like ground obstacles using segmentation and blob detection. The inverse homography converts detected image points into metric ground coordinates. We can also analyse uncertainty by placing markers at known locations and measuring reconstruction error.

Boundary-Based Distance Measurement: If we segment an image into ground and obstacles, the boundary between them lies on the ground plane. These boundary points can be transformed into ground coordinates using the homography. This effectively turns the camera into a simple laser-like range sensor, measuring distances to walls or obstacles across multiple points simultaneously.

7 SLAM

Simultaneous localization and mapping (SLAM): a body with quantative sensors moves through a previously unknown, static environment, mapping it and calculating its egomotion. Most SLAM algorithms make makes of natural scene *features*, Features should be easily recognisable from different viewpoints to enable reliable matching. In SLAM, we store & update a joint distribution over the states of both the robot and the mapped world. Features are gradually discovered as the robot explores, so the dimension of this joint distribution problem will grow.

The most common way to efficiently represent the high-dimensional probability distributions is a joint Gaussian distribution, updated via the **Extended Kalman Filter**. The PDF is represented by a **state vector** & **covariance matrix**:

$$\hat{x} = \begin{pmatrix} \hat{x}_v \\ \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{pmatrix}, \quad P = \begin{bmatrix} P_{xx} & P_{xy_1} & P_{xy_2} & \cdots \\ P_{y_1x} & P_{y_1y_1} & P_{y_1y_2} & \cdots \\ P_{y_2x} & P_{y_2y_1} & P_{y_2y_2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The state vector contains the robot's state (x_v) and all feature states y_i. Purely metric probabilistic SLAM is limited to small domains due to *poor computational scaling of probabilistic filters* and *growth in uncertainty at large distances from the starting point*. Instead, lots of little local maps are made which are then related at a higher level. Modern solutions follow a *metric / topological approach* to approximate full metric SLAM, avoiding maintaining a single massive joint covariance matrix. They need:

- Local metric mapping for accurate short-term estimation.
- Place recognition for loop closure.
- Global optimisation for consistency.

7.1 Loop Closure Detection

Done using image retrieval techniques such as Visual Bag-of-Words. When an old place is recognised, strong constraints are added to the map. Place recognition converts perceptual similarity into geometric constraints that reduce accumulated drift.

7.2 Purely Toplogical SLAM

Here, the environment is represented as a graph of places and connections. No explicit metric coordinates are stored. This representation is compact and well-suited for symbolic planning but lacks precise geometric detail.

7.3 Pose Graph Optimisation

In graph-based SLAM, nodes represent robot poses and edges represent relative motion constraints. When loops are detected, pose graph optimisation adjusts all poses to maximise global consistency. Optimisation only meaningfully changes the map when loops are present to constrain accumulated drift.

A **factor graph** expresses the SLAM posterior as a product of measurement likelihoods $p(x) = \prod_s f_s(x_s)$. Each factor has Gaussian form:

$$f_s(x) = K\exp\left(-\frac{1}{2}(z_s - h_s(x_s))^\top \Lambda_s (z_s - h_s(x_s))\right)$$

where Λ_s is the measurement precision matrix. Solving a factor graph means computing the most probable configuration of variables or their marginals. Sparsity of the graph allows scalable optimisation. Common approaches include:

- Batch optimisation (bundle adjustment)
- Incremental filtering (EKF-style)
- Incremental smoothing (e.g., iSAM)
- Belief propagation