ECE 209AS.1 Computational Robotics

Fall Quarter–2024

Path Planning Algorithms: Summary & Mathematical Formulation

Documentation made by:

RENISH ISRAEL (UID: 606530590)
First Year M.S. Student
Samueli Electrical and Computer Engineering
University of California Los Angeles

November 24, 2024

Contents

1 Artificial Force Field

1.1 Key Points

1. Also called "Artificial Potential Field" Algorithm (APF)
2. Robot navigates using "attractive" forces towards the goal or destination, and "repulsive" forces away from the obstacles
3. Very common and useful approach to avoid obstacles
4. The destination has the lowest potential, and the obstacles have the highest. Intuitively, the robot moves from higher to lower potentials (in the direction of decreasing gradient of the potential field)
5. A potential field is a physical field that obeys Laplace's equation. In other words, the force field associated with such a potential field ϕ can be evaluated by taking the gradient of the potential field.

$$\vec{F} = -\nabla\phi$$

1.2 Attractive Force

The potential field corresponding to the goal can be given as:

$$\phi_{\text{goal}}(x, y) = c\sqrt{(x - x_{\text{goal}})^2 + (y - y_{\text{goal}})^2}$$

, where (x, y) is the coordinate of the robot (node) and $(x_{\text{goal}}, y_{\text{goal}})$ is the coordinate of the 'goal'. c is a positive constant. Also, the force field can then be obtained as:

$$\vec{F}_{\text{goal}} = \left(\frac{-c}{\sqrt{(x - x_{\text{goal}})^2 + (y - y_{\text{goal}})^2}}\right)\left((x - x_{\text{goal}})\,\hat{i} + (y - y_{\text{goal}})\,\hat{j}\right)$$

The formulation is such that the potential field ϕ_{goal} has a minima at $(x_{\text{goal}}, y_{\text{goal}})$

1.3 Repulsive Forces

Two kinds of repulsive forces are considered here, produced by the following sources:

1. The boundaries of the considered environment
2. All the obstacles in the environment

Boundary Potential:

The repulsive potential field ϕ_b caused by the boundaries can be formulated as follows:

$$\phi_b(x, y) = \frac{1}{\delta + \sum_{i=1}^{s}(g_i + |g_i|)}$$

, where δ is a small positive number to avoid ϕ_b going to ∞, s is the total number of boundary segments, and g_i is the linear equation that represents the boundary i mathematically.

Clearly, one can observe that there is a maxima for ϕ_b at each boundary.

Obstacle Potential:

Let the repulsive potential field generated by obstacle k is given by ϕ_k. If the obstacle k is a square of side l_k with center at (x_k, y_k), then ϕ_k is formulated as:

$$\phi_k(x, y) = \frac{\phi_{\max}}{1 + h(x, y)}$$

where, $h(x, y) = (x_k - l/2 - x) + |x_k - l/2 - x| + (y_k - l/2 - y) + |y_k - l/2 - y| + (x + 1 - x_k - l/2) + |x + 1 - x_k - l/2| + (y + 1 - y_k - l/2) + |y + 1 - y_k - l/2|$
Also, $\phi_{\max}$ is the maxima of the function ϕ_k for all values of k. The function h captures the side of the obstacle boundary the robot is in, and assigns potential accordingly. Finally, the obstacle potential ϕ_{obs} can be calculated as follows:

$$\phi_{\text{obs}} = \max\{\phi_k\}_{k=1}^{K}$$

where, K is the total number of obstacles in the environment.

1.4 Robot Trajectory

Now, the net force on the robot at time t can be calculated as follows:

$$\vec{F}_{\text{net}} = \vec{F}_{\text{goal}} - (\nabla\phi_b + \nabla\phi_{\text{obs}})$$

Hence, the robot's subsequent direction of motion should be given by the following direction vector: $\hat{d} = \cos\theta\,\hat{i} + \sin\theta\,\hat{j}$, where

$$\theta = \tan^{-1}\left(\frac{\vec{F}_{\text{net}} \cdot \hat{j}}{\vec{F}_{\text{net}} \cdot \hat{i}}\right)$$

1.5 References

1. https://medium.com/@rymshasiddiqui/path-planning-using-potential-field-algorithm-a30ad12bdb08
2. Robotic Planning: Chapter 4 by *Howie Choset*, CMU
3. Science Direct
4. https://youtu.be/Ls8EBoG_SEQ

2 Contact-based Mapping

2.1 Key Points

1. A robot directly interacts with the environment using active touching of objects with the help of tactile sensors that provide information about pressure, temperature, force etc.,
2. This method emulates the human sense of 'touch' by which we are able to 'feel' the happenings in our surroundings
3. The robot can use the tactile sensors to gain information about the texture and geometry of the environment, rather than completely using vision-based sensors
4. The end goal is to build a map of the environment just by using a robot with tactile sensors
5. One of the major advantages in this method is, it can work well in poorly-lit conditions like shadowy areas, deep dark forests etc.,

2.2 Robot state & Environment map

Let the state of the robot at time t be a two-dimensional real vector that describes its position:

$$\mathbf{s_t} = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

Let $\mathbf{M_t}$ be a matrix that mathematically describes the map of the environment the robot lives in, at time t. The environment can be thought of as a grid, that is divided into say, a $m \times n$ cells. Hence, the dimensions of $\mathbf{M_t}$ would also be $m \times n$.

$$\mathbf{M_t} = \begin{bmatrix} M_{11} & M_{12} & \dots & M_{1n} \\ M_{21} & M_{22} & \dots & M_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ M_{m1} & M_{m2} & \dots & M_{mn} \end{bmatrix}$$

$$M_{ij}(t) = \begin{cases} 1 & \text{if the cell is occupied by an obstacle} \\ 0 & \text{if the cell is a free space} \end{cases}$$

Contact-based tactile sensors are placed on the robot (at the outer edges). These sensors basically measure the force upon contact in both $x-$ and $y-$ directions. The force vector obtained from sensor 1 could be represented as follows:

$$\mathbf{f_1(t)} = \begin{pmatrix} f_{x1}(t) \\ f_{y1}(t) \end{pmatrix}$$

Each measured sensor output would be a function of time, and there could be k different sensors on the robot.

The overall measured force vector from all the k sensors can be represented by a $2k \times 1$ vector $\mathbf{f_{net}(t)}$ as follows:

$$\mathbf{f_{net}(t)} = \begin{pmatrix} f_{x1}(t) \\ f_{y1}(t) \\ f_{x2}(t) \\ f_{y2}(t) \\ \vdots \\ \vdots \\ f_{xk}(t) \\ f_{yk}(t) \end{pmatrix}$$

2.3 Goal and Methodology

The intention of this algorithm is to estimate the location of the contact point in cartesian co-ordinates with the help of the measurement feedback obtained from the tactile sensors on the robot. The contact point $\mathbf{p_{net}(t)}$ of the robot could be formulated as a vector relative to all the end-effectors of the robot. A vector relating the contact point position to sensor 1 could be:

$$\mathbf{p_1(t)} = \begin{pmatrix} p_{x1}(t) \\ p_{y1}(t) \end{pmatrix}$$

Here, the term 'end-effectors' refers to the k tactile sensors that are used in this method. The overall relative position vector $\mathbf{p_{net}(t)}$ would be:

$$\mathbf{p_{net}(t)} = \begin{pmatrix} p_{x1}(t) \\ p_{y1}(t) \\ p_{x2}(t) \\ p_{y2}(t) \\ \vdots \\ \vdots \\ p_{xk}(t) \\ p_{yk}(t) \end{pmatrix}$$

We require a 'bridge' between the contact point's location and the magnitude and direction of the forces measured by the k sensors. This is provided by the Jacobian Matrix, that relates the force vector $\mathbf{f_{net}(t)}$ to the relative-position vector $\mathbf{p_{net}(t)}$. The relation is given in the form of matrix-vector product as:

$$\mathbf{f_{net}(t)} = \mathbf{J(x_t)} \,.\, \mathbf{p_{net}(t)}$$

In the above equation, $\mathbf{J(x_t)}$ is the Jacobian Matrix containing the 1st-order partial derivatives both $x-$ and $y-$ force components of each of the k sensor outputs, with respect to the position of the robot itself (which can be tracked based on the robot's motion). The dimensions of $\mathbf{J(x_t)}$ is given by $2k \times 2k$. Its mathematical definition is shown in the expression, next page.

$$\mathbf{J}(\mathbf{x_t}) = \begin{bmatrix} \frac{\partial f_{x1}}{\partial x_1} & \frac{\partial f_{x1}}{\partial y_1} & \cdots & \frac{\partial f_{x1}}{\partial x_k} & \frac{\partial f_{x1}}{\partial y_k} \\ \frac{\partial f_{y1}}{\partial x_1} & \frac{\partial f_{y1}}{\partial y_1} & \cdots & \frac{\partial f_{y1}}{\partial x_k} & \frac{\partial f_{y1}}{\partial y_k} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial f_{xk}}{\partial x} & \frac{\partial f_{xk}}{\partial y} & \cdots & \frac{\partial f_{xk}}{\partial x_k} & \frac{\partial f_{xk}}{\partial y_k} \\ \frac{\partial f_{yk}}{\partial x} & \frac{\partial f_{yk}}{\partial y} & \cdots & \frac{\partial f_{yk}}{\partial x_k} & \frac{\partial f_{yk}}{\partial y_k} \end{bmatrix}$$

This matrix is known because it depends on the sensor positioning, and the modeling of the sensors that we use on the robot. The fundamental idea is that sensor-measured forces give an idea about the relative position of the contact point of the robot.

Once an estimate of the position of the contact point, and the magnitude of the forces acting on the contact point is obtained, the overall map $\mathbf{M_t}$ of the environment can be updated, as new information is available at each time instant t. Initially, $\mathbf{M_0}$ is a Null matrix. Gradually, cells occupied by obstacles would be filled by 1s. The larger the cell density, the better the final map would capture the environment.

2.4 References

1. "Inductive learning of force-based robot skills," IEEE International Conference on Robotics and Automation (ICRA), 2011.

2. "Robotic Grasping and Contact: A Review," IEEE International Conference on Robotics and Automation (ICRA), 2005.

3 Random Walk

3.1 Key Points

1. A computational method wherein a Robot chooses a random direction at each pre-defined time instant
2. Very useful algorithm if one needs to cover large unexplored areas
3. The robot may miss important areas due to the algorithm being completely random
4. Adaptive random walks can greatly improve inefficiencies in environments where there are are a lot of obstacles

3.2 Probability Theory of the 1-D Walk

Let us assume that the robot starts at a point $x_0 = 0$. Now, on the 1-D line, it has two options to choose from. If the probability of choosing to go to the right is p, then the probability of going left would be $q = 1 - p$. If the position of the robot at time k is given by x_k, then we have the following:

$$\mathbb{P}(x_{k+1} = x_k + 1) = p$$

$$\mathbb{P}(x_{k+1} = x_k - 1) = q$$

The direction taken at each time instant t can be denoted by a Bernoulli Random Variable ξ_t with parameter p. Here, ξ_t can take the values ± 1. So, x_k for $k \geq 1$ can be written as:

$$x_k = x_0 + \sum_{t=1}^{k} \xi_t$$

The mean and variance of x_k are given as:

$$\mathbb{E}(x_k) = k(p - q)$$

$$\text{Var}(x_k) = 4kpq$$

The variance will be maximum when $p = q = 0.5$, and is linearly proportional to k. Hence, the "walk" would be more random as k is higher.

3.3 Applying the theory to 2-D space

The same concept can be used in the exploration of a 2-D environment. 8 possible directions can be considered, and one of them is chosen at random. Assuming the robot occupies $m \times n$ pixels, one can assume the magnitude of each stride in the walk to be r pixels in the orthogonal directions, and $\lfloor r/\sqrt{2} \rfloor$ pixels in the diagonal directions. During each walk, the total area covered (in %) is calculated.

3.4 References

1. "Random Walk-based approach for addressing of navigation algorithms in service and surveillance robots, Journal of Theoretical and Applied Information Technology," December 2023, Vol. 101 No.23

4 Reactive Bounce-back

4.1 Key Points

1. In this algorithm, the robot is made to react instantly to the obstacles it encounters during an exploration.
2. Real-time information is provided by tactile sensors
3. "Behavior-based control" is used that helps to assign an action to the robot, for specific situations it might encounter
4. Quite the opposite of proactive algorithms which do not use touch-based sensors to detect and react

Flowchart depicting the Algorithm

The robot's state $\mathbf{s_t}$ can be described by a 4×1 real vector that comprises of its position and velocity at time t:

$$\mathbf{s_t} = \begin{pmatrix} x(t) \\ y(t) \\ v_x(t) \\ v_y(t) \end{pmatrix}$$

Detection of Obstacle:

- Proximity sensors like LiDAR, ultrasound, or cameras are used to get information about location of obstacles in the near surrounding of the robot
- $d_{\text{obs}}(t) = \min(\text{Distances between each obstacle and the robot at time } t)$
- If $d_{\text{obs}}(t)$ is lower than a threshold value d_{th}, the robot should react to the situation instantly
- The above condition is called the "Bounce-back Trigger Condition"

Behavior:

- If the Bounce-back trigger condition is satisfied, then the robot must reverse its own velocity to proceed further in the exploration, omitting the encountered obstacle
- Hence, the velocity vector should undergo the following change:

$$\begin{pmatrix} v_x(t+1) \\ v_y(t+1) \end{pmatrix} = -\alpha \begin{pmatrix} v_x(t) \\ v_y(t) \end{pmatrix}$$

Here, $\alpha \in (0, 1]$. For slow reversing, $\alpha \ll 1$ can be used.

- Due to this change in velocity, the position of the robot would be updated as follows:

$$\begin{pmatrix} x(t+1) \\ y(t+1) \end{pmatrix} = -\alpha \begin{pmatrix} v_x(t) \\ v_y(t) \end{pmatrix} \Delta t$$

  with Δt being the time step

Path Re-planning:

- Once the velocity of the robot gets reversed, and the state vector $\mathbf{s_t}$ has been updated, the new path that the robot must take, needs to be planned
- For this purpose, algorithms like the Artificial Force Field (AFF) or the Dynamic Window Approach (DWA) can be used
- The DWA has been described in detail, in the next section

4.2 Dynamic Window Approach (DWA)

Suppose the state vector of the robot also contains the Azimuthal angular orientation θ that accurately describes the robot's direction with respect to some reference direction (say, North). Hence, the angular velocity $\omega = \dot{\theta}$ can be an element of $\mathbf{s_t}$ as well. Hence, using kinematics, one can write:

$$v(t) = \sqrt{(v_x(t))^2 + (v_y(t))^2}$$

$$v_x(t) = v(t)\cos(\theta(t))$$
$$v_y(t) = v(t)\sin(\theta(t))$$
$$\dot{\theta}(t) = \omega(t)$$

The Dynamic Window defines a set of time-dependent and state-dependent feasible values for the linear velocity $v(t)$ and angular velocity $\omega(t)$ as shown below:

$$v_{\min} \leq v(t) \leq v_{\max}$$

$$\omega_{\min} \leq \omega(t) \leq \omega_{\max}$$

For a short time horizon T_H in the future, the state of the robot is simulated using candidate pairs $(v_c(t), \omega_c(t))$. The updates can be written as follows:

$$\theta(t + T_H) = \theta(t) + T_H\, \omega_c(t)$$

$$x(t + T_H) = x(t) + T_H\, v_c(t)\cos(\theta(t))$$
$$y(t + T_H) = y(t) + T_H\, v_c(t)\sin(\theta(t))$$

For each of the candidate pairs, once the final state is generated, the overall cost of the trajectory $\mathcal{J}$ is evaluated by summing individual cost functions based on:

(a) Distance of robot to the final goal (d_{goal}) : $J_{\text{goal}} = -d_{\text{goal}}$

(b) Distances of robot from each obstacle :

$$J_{\text{obs}} = \sum_{i=1}^{n_{\text{obs}}} \left(\frac{1}{d_i}\right)^2$$

(c) Smoothness of change in velocities : $J_{\text{vel}} = |\Delta v| + |\Delta \omega|$

$$\mathcal{J} = J_{\text{goal}} + J_{\text{obs}} + J_{\text{vel}}$$

The optimal pair $(v_{\text{opt}}(t), \omega_{\text{opt}}(t))$ is the one that has the least trajectory cost $\mathcal{J}$.

Reactive Control and Bounce-back:

- The state variables of the robot are carefully updated for $(t + T_H)$ using the obtained optimal velocity pair
- The process repeats at each time instant, till the robot reaches the goal state

4.3 References

1. "The dynamic window approach to collision avoidance," IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997, doi: 10.1109/100.580977.
2. https://techxplore.com/news/2023-04-built-robotics-drone-cope-collisions.html

5 Frontier Selection

5.1 Key Points

1. A 'Frontier' is a boundary that separates the explored and unexplored regions of the map, by the robot
2. The speed of exploration in this method is very high
3. Redundancy of visiting the same cells is greatly reduced

5.2 Frontier Representation

Let $\mathbf{M_t}$ be a matrix that mathematically describes the map of the environment the robot lives in, at time t. The environment can be thought of as a grid, that is divided into say, a $m \times n$ cells. Hence, the dimensions of $\mathbf{M_t}$ would also be $m \times n$.

$$\mathbf{M_t} = \begin{bmatrix} M_{11} & M_{12} & \dots & M_{1n} \\ M_{21} & M_{22} & \dots & M_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ M_{m1} & M_{m2} & \dots & M_{mn} \end{bmatrix}$$

$$M_{ij}(t) = \begin{cases} 0 & \text{if the cell is still unexplored} \\ 1 & \text{if the cell is explored, and is free} \\ 2 & \text{if the cell is occupied by an obstacle} \end{cases}$$

Based on the map matrix $\mathbf{M}$, the set F which contains all the points on the frontier can be defined as:

$$F = \{(i,j) \mid M_{ij} = 1 \text{ and } \textit{atleast} \text{ one adjacent } M_{i'j'} = 0\}$$

In other words, all the frontier points must have been explored by the robot, free of obstacles, and must be adjacent to *atleast* one cell in the grid that has not been explored yet.

5.3 Choosing the Frontier

Selection of the optimal frontier involves defining and evaluating certain metrics:

(a) Information Gain:

The expected information gain $I(f)$ of a frontier point $f \in F$, is a measure of how much new information could be gained by exploring it. The larger the gain, the better it is to explore the point because of higher number of surrounding unexplored cells.

$$I(f) = \sum_{(i,j) \in \mathcal{N}(f)} \mathbf{1}(M_{ij} = 0)$$

$\mathcal{N}(f)$ is the set of all neighboring cells of f with respect to the map-matrix $\mathbf{M}$.

(b) Path Length:

The path length $L(f)$ from the current robot position (x_r, y_r) to the frontier position (x_f, y_f)

needs to be taken into account. The higher the length, the higher the energy consumption of the robot, and the longer it takes for the robot to reach the frontier. Let the path from the robot to the frontier be given as a sequence of $(n+1)$ coordinates:

$$P_f = \{(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots (x_n, y_n)\}$$

Here, $(x_0, y_0) \equiv (x_r, y_r)$ and $(x_n, y_n) \equiv (x_f, y_f)$. The sequence P_f can be obtained by using algorithms like Dijkstra or A^*.

$$L(f) = \sum_{i=0}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

The overall cost function corresponding to a frontier f can then be given as:

$$C(f) = -\alpha \left(\frac{I(f)}{I_{\max}} \right) + \beta \left(\frac{L(f)}{L_{\max}} \right)$$

where, $I_{\max}$ and $L_{\max}$ are parameters of feature-normalization. α and β are positive real numbers that act as weights for the individual metrics. The optimal frontier f_{opt} is simply chosen as:

$$f_{\text{opt}} = \arg\min_{f \in F} C(f)$$

5.4 References

1. https://github.com/Brandonio-c/Frontier-Exploration

2. "Autonomous Robots: Modeling, Path Planning, and Control", by *George A. Bekey*

6 Simultaneous Localization And Mapping (SLAM)

6.1 Key Points

1. This appears to be a "Chicken or Egg" problem since both the state of the robot and the environment map have to be estimated at the same time

2. But several algorithms have been found to solve it, such as: Particle Filter, Extended Kalman Filter, Covariance Intersection, and GraphSLAM

6.2 Problem Description

Given a series of control signals $\{u_\tau\}_{\tau=1}^{t-1}$ and sensor observations $\{z_\tau\}_{\tau=1}^{t-1}$ over discrete time steps, the SLAM algorithm should be able to estimate the trajectory of the robot ie., sequence of states of the robot $\{\mathbf{x}_\tau\}_{\tau=1}^{t}$ and the environment mapping $\mathbf{m}$.

$$\textbf{Objective: To Find } \mathbb{P}(\mathbf{x_1}, \mathbf{x_2}, \dots \mathbf{x_t}, \mathbf{m} \,|\, \mathbf{u_{1:t-1}}, \mathbf{z_{1:t-1}})$$

6.3 Bayes Filter

Given a time series of control inputs $\{u_\tau\}_{\tau=1}^{t}$ and sensor observations $\{z_\tau\}_{\tau=1}^{t}$, the current robot state $\mathbf{x_t}$ can be estimated by this method. The posterior Bayes probability of a future possible state, given information about the controls and observations of the system, is termed as 'Belief' of the particular future state:

$$\begin{aligned}
\text{Bel}(\mathbf{x_t}) &= \mathbb{P}(\mathbf{x_t} \,|\, z_{1:t}, \mathbf{u_{1:t}}) \\
&= \eta\, \mathbb{P}(\mathbf{z_t} \,|\, \mathbf{x_t}, \mathbf{z_{1:t-1}}, \mathbf{u_{1:t}})\, \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{z_{1:t-1}}, \mathbf{u_{1:t}}) \\
&= \eta\, \mathbb{P}(\mathbf{z_t} \,|\, \mathbf{x_t})\, \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{z_{1:t-1}}, \mathbf{u_{1:t}}) && \text{[Markov Property]} \\
&= \eta\, \mathbb{P}(\mathbf{z_t} \,|\, \mathbf{x_t}) \int_{\mathbf{x_{t-1}}} \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{x_{t-1}}, \mathbf{z_{1:t-1}}, \mathbf{u_{1:t}})\, \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{z_{1:t-1}}, \mathbf{u_{1:t}})\, d\mathbf{x_{t-1}} \\
& && \text{[Law of Total Probability]} \\
&= \eta\, \mathbb{P}(\mathbf{z_t} \,|\, \mathbf{x_t}) \int_{\mathbf{x_{t-1}}} \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{x_{t-1}}, u_t)\, \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{z_{1:t-1}}, \mathbf{u_{1:t}})\, d\mathbf{x_{t-1}} \\
&= \eta\, \mathbb{P}(\mathbf{z_t} \,|\, \mathbf{x_t}) \int_{\mathbf{x_{t-1}}} \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{x_{t-1}}, u_t)\, \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{z_{1:t-1}}, u_{1:t-1})\, d\mathbf{x_{t-1}} \\
&= \eta\, \mathbb{P}(\mathbf{z_t} \,|\, \mathbf{x_t}) \int_{\mathbf{x_{t-1}}} \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{x_{t-1}}, u_t)\, \text{Bel}(\mathbf{x_{t-1}})\, d\mathbf{x_{t-1}} \\
& && \text{[Recursive Equation]}
\end{aligned}$$

The above equation can be split-up and written in 2 steps as follows:

$$\begin{aligned}
\text{Prediction Step: } \overline{\text{Bel}}(\mathbf{x_t}) &= \int_{\mathbf{x_{t-1}}} \mathbb{P}(\mathbf{x_t} \,|\, \mathbf{x_{t-1}}, u_t)\, \text{Bel}(\mathbf{x_{t-1}})\, d\mathbf{x_{t-1}} \\
\text{Correction Step: } \text{Bel}(\mathbf{x_t}) &= \eta\, \mathbb{P}(\mathbf{z_t} \,|\, \mathbf{x_t})\, \overline{\text{Bel}}(\mathbf{x_t})
\end{aligned}$$

η is the normalizing factor.

6.4 Multivariate Gaussian Random Variables

If $\mathbf{x}$ is a $n-$dimensional gaussian vector with mean vector μ and covariance matrix $\mathbf{\Sigma}$, then its Probability Mass Function (PMF) is given as:

$$\mathbb{P}(\mathbf{X} = \mathbf{x}) = \left(\frac{1}{(2\pi)^{n/2}\sqrt{\det(\mathbf{\Sigma})}} \right) \exp\left(-\frac{1}{2}(\mathbf{x} - \mu)^{\mathbf{T}} \mathbf{\Sigma^{-1}} (\mathbf{x} - \mu) \right)$$

Joint and Conditional Distributions:

Let $\mathbf{X_a}$ be a n_a-dimensional gaussian random variable with mean vector $\mu_\mathbf{a}$ and variance matrix $\mathbf{\Sigma_{aa}}$, and let $\mathbf{X_b}$ be a n_b-dimensional multivariate gaussian random variable with mean vector $\mu_\mathbf{b}$ and variance matrix $\mathbf{\Sigma_{bb}}$. If the covariance matrices associated with these 2 random variables are $\mathbf{\Sigma_{ab}}$ and $\mathbf{\Sigma_{ba}}$, then we have the following Joint PMF:

$$\mathbb{P}\left(\mathbf{X} = \mathbf{x} = \begin{pmatrix} \mathbf{x_a} \\ \mathbf{x_b} \end{pmatrix} \right) = \left(\frac{1}{(2\pi)^{(n_a+n_b)/2}\sqrt{\det(\mathbf{\Sigma_j})}} \right) \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_\mathbf{j})^{\mathbf{T}} \mathbf{\Sigma_j^{-1}} (\mathbf{x} - \mu_\mathbf{j}) \right)$$

$$\text{where, } \mu_\mathbf{j} = \begin{pmatrix} \mu_\mathbf{a} \\ \mu_\mathbf{b} \end{pmatrix} \text{ and } \mathbf{\Sigma_j} = \begin{pmatrix} \mathbf{\Sigma_{aa}} & \mathbf{\Sigma_{ab}} \\ \mathbf{\Sigma_{ba}} & \mathbf{\Sigma_{bb}} \end{pmatrix}$$

Also, the Conditional PMF of $\mathbf{X_a}$ given $\mathbf{X_b}$ is shown below:

$$\mathbb{P}(\mathbf{X_a} = \mathbf{x_a} \,|\, \mathbf{X_b} = \mathbf{x_b}) = \left(\frac{1}{(2\pi)^{n_a/2}\sqrt{\det(\mathbf{\Sigma_c})}} \right) \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_\mathbf{c})^{\mathbf{T}} \mathbf{\Sigma_c^{-1}} (\mathbf{x} - \mu_\mathbf{c}) \right)$$

$$\text{where, } \mu_\mathbf{c} = \mu_\mathbf{a} + \mathbf{\Sigma_{ab}\Sigma_{bb}^{-1}}(\mathbf{b} - \mu_\mathbf{b}) \text{ and } \mathbf{\Sigma_c} = \mathbf{\Sigma_{aa}} - \mathbf{\Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}}$$

6.5 Kalman Filter

It is a Bayes Filter and is used as an estimator for the linear gaussian case. It provides the optimal solution for Linear Models and Gaussian Distributions.

Linear Model:

$$\mathbf{x_t} = \mathbf{A_t x_{t-1}} + \mathbf{B_t u_t} + \epsilon_\mathbf{t}$$
$$\mathbf{z_t} = \mathbf{C_t x_t} + \delta_\mathbf{t}$$

If the state vector $\mathbf{x_t}$ has dimensions $(n \times 1)$, the control input space is $\mathbb{R}^m$ and the output space is $\mathbb{R}^k$, then the following points can be made about the Linear Model:

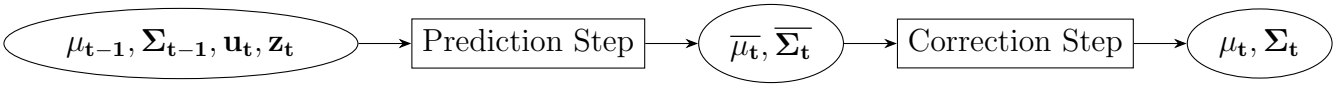
- $\mathbf{A_t}$ is a $(n \times n)$ matrix that describes how the state $\mathbf{x_t}$ changes with time, in the absence of control inputs and noise
- $\mathbf{B_t}$ is a $(n \times m)$ matrix that describes how the control input signals affect the system state
- $\mathbf{C_t}$ is a $(k \times n)$ matrix that maps the state of the system to its output
- $\epsilon_\mathbf{t}$ and $\delta_\mathbf{t}$ are independent gaussian process and observation noise vectors with dimensions $(n \times 1)$ and $(k \times 1)$ respectively. They have 0 mean, and covariance matrices $\mathbf{R_t}$ and $\mathbf{Q_t}$ respectively

Now, the following conditional probability mass functions could be evaluated using the above model:

$$\mathbb{P}(\mathbf{x_t} \mid \mathbf{x_{t-1}}, \mathbf{u_t}) = \left(\frac{1}{(2\pi)^{n/2}\sqrt{\det(\mathbf{R_t})}} \right) \exp\left(-\frac{1}{2}(\mathbf{x_t} - \mathbf{A_t x_{t-1}} - \mathbf{B_t u_t})^{\mathbf{T}} \mathbf{R_t}^{-1} (\mathbf{x_t} - \mathbf{A_t x_{t-1}} - \mathbf{B_t u_t}) \right)$$

$$\mathbb{P}(\mathbf{z_t} \mid \mathbf{x_t}) = \left(\frac{1}{(2\pi)^{k/2}\sqrt{\det(\mathbf{Q_t})}} \right) \exp\left(-\frac{1}{2}(\mathbf{z_t} - \mathbf{C_t x_t})^{\mathbf{T}} \mathbf{Q_t}^{-1} (\mathbf{z_t} - \mathbf{C_t x_t}) \right)$$

The Algorithm:



Flowchart depicting the Kalman Filter

Algorithm 1 The Kalman Filter Algorithm

Require: $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$

Prediction Step:

$\overline{\mu_t} \leftarrow \mathbf{A_t}\mu_{t-1} + \mathbf{B_t u_t}$
$\overline{\Sigma_t} \leftarrow \mathbf{A_t}\Sigma_{t-1}\mathbf{A_t^T} + \mathbf{R_t}$

Correction Step:

$\mathbf{K_t} = \overline{\Sigma_t}\mathbf{C_t^T}(\mathbf{C_t}\overline{\Sigma_t}\mathbf{C_t^T} + \mathbf{Q_t})^{-1}$
$\mu_t = \overline{\mu_t} + \mathbf{K_t}(\mathbf{z_t} - \mathbf{C_t}\overline{\mu_t})$
$\Sigma_t = (\mathbf{I} - \mathbf{K_t C_t})\overline{\Sigma_t}$

Return: μ_t, Σ_t

6.6 Extended Kalman Filter (EKF)

- The Kalman Filter fails when the system transitions cannot be modeled using linear functions, or if the noise vectors are not gaussian
- The EKF fixes this problem using "Local Linearization" using Taylor-Series expansion
- The underlying concept remains the same as the Kalman Filter

Non Linear Dynamics:

$$\mathbf{x_t} = \mathbf{g}(\mathbf{x_{t-1}}, \mathbf{u_t}) + \epsilon_\mathbf{t}$$
$$\mathbf{z_t} = \mathbf{h}(\mathbf{x_t}) + \delta_\mathbf{t}$$

where, $\mathbf{g}(.)$ and $\mathbf{h}(.)$ are non-linear functions.

Local Linearization:

$$\mathbf{g}(\mathbf{x_{t-1}}, \mathbf{u_t}) = \mathbf{g}(\mu_{\mathbf{t-1}}, \mathbf{u_t}) + (\mathbf{x_{t-1}} - \mu_{\mathbf{t-1}}) \underbrace{\left.\frac{\partial \mathbf{g}(\mathbf{x_{t-1}}, \mathbf{u_t})}{\partial \mathbf{x_{t-1}}}\right|_{\mathbf{x_{t-1}}=\mu_{\mathbf{t-1}}}}_{=: \mathbf{G_t}}$$

$$\mathbf{h}(\mathbf{x_t}) = \mathbf{h}(\overline{\mu_\mathbf{t}}) + (\mathbf{x_t} - \overline{\mu_\mathbf{t}}) \underbrace{\left.\frac{\partial \mathbf{h}(\mathbf{x_t})}{\partial \mathbf{x_t}}\right|_{\mathbf{x_t}=\overline{\mu_\mathbf{t}}}}_{=: \mathbf{H_t}}$$

Here, $\mathbf{G_t}$ and $\mathbf{H_t}$ are Jacobian Matrices of the the functions $\mathbf{g}(.)$ and $\mathbf{h}(.)$ with respect to $\mathbf{x_t}$.

The Bayes Conditional PMFs and the Filter Algorithm would be similar to the Kalman Filter. Only the linear model needs to be replaced by the linear approximation model obtained in this section.

Algorithm 2 The Extended Kalman Filter Algorithm

Require: $\mu_{\mathbf{t-1}}, \mathbf{\Sigma_{t-1}}, \mathbf{u_t}, \mathbf{z_t}$

Prediction Step:

$\overline{\mu_\mathbf{t}} \leftarrow \mathbf{g}(\mu_{\mathbf{t-1}}, \mathbf{u_t})$
$\overline{\mathbf{\Sigma_t}} \leftarrow \mathbf{G_t \Sigma_{t-1} G_t^T} + \mathbf{R_t}$

Correction Step:

$\mathbf{K_t} = \overline{\mathbf{\Sigma_t}} \mathbf{H_t^T} (\mathbf{H_t} \overline{\mathbf{\Sigma_t}} \mathbf{H_t^T} + \mathbf{Q_t})^{-1}$
$\mu_\mathbf{t} = \overline{\mu_\mathbf{t}} + \mathbf{K_t}(\mathbf{z_t} - \mathbf{h}(\overline{\mu_\mathbf{t}}))$
$\mathbf{\Sigma_t} = (\mathbf{I} - \mathbf{K_t H_t}) \overline{\mathbf{\Sigma_t}}$

Return: $\mu_\mathbf{t}, \mathbf{\Sigma_t}$

6.7 Particle Filter

- The Kalman Filter, EKF and other variants can only model Gaussian Distributions
- To deal with arbitrary distributions, the key idea is to use multiple random samples

Particle Set:

Let the number of samples used to represent the posterior probability distribution be J. In general, J is very large for the algorithm to work as expected. The Particle Set Ψ consists of ordered pairs of the J samples denoted by $x^{[j]}$ and their corresponding weights, denoted by $w^{[j]}$.

$$\Psi = \left\{ \left\langle x^{[j]}, w^{[j]} \right\rangle \right\}_{j=1,2,\dots J}$$

The PMF represented by Ψ is given as:

$$\mathbb{P}(X = x) = \sum_{j=1}^{J} w^{[j]} \delta_{x^{[j]}}(x)$$

where, $\delta_{x^{[j]}}(x)$ is the Dirac-Delta function, having a spike at $x = x[j]$

The Algorithm:

There are 3 key steps involved in this algorithm:

1. Sampling using a proposal distribution $\pi(.)$:

$$x_t^{[j]} \sim \pi(x_t | \dots)$$

2. Importance weights computation using the target distribution $p(.)$:

$$w_t^{[j]} = \frac{p\left(x_t^{[j]}\right)}{\pi\left(x_t^{[j]}\right)}$$

3. Resampling by drawing sample i with probability $w^{[i]}$ and repeating the process for J times

The complete algorithm that is used to obtain the Particle Set is given in the next page.

Algorithm 3 The Particle Filter Algorithm

Require: $\Psi_{t-1}, \mathbf{u_t}, \mathbf{z_t}$

$\overline{\Psi_t} = \Psi_t = \Phi$

for $j = 1$ to J **do**
  $x_t^{[j]} \sim \pi(x_t)$
  $w_t^{[j]} = \frac{p\left(x_t^{[j]}\right)}{\pi\left(x_t^{[j]}\right)}$
  Add $\left\langle x_t^{[j]}, w_t^{[j]} \right\rangle$ to $\overline{\Psi_t}$
end for

for $j = 1$ to J **do**
  Draw $i \in \{1, 2, \ldots J\}$ with probability $\propto w^{[i]}$
  Add $x_t^{[i]}$ to Ψ_t
end for

Return: Ψ_t

6.8 References

1. "SLAM Course," by *Prof. Cyril Stachniss,* University of Bonn

2. https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping