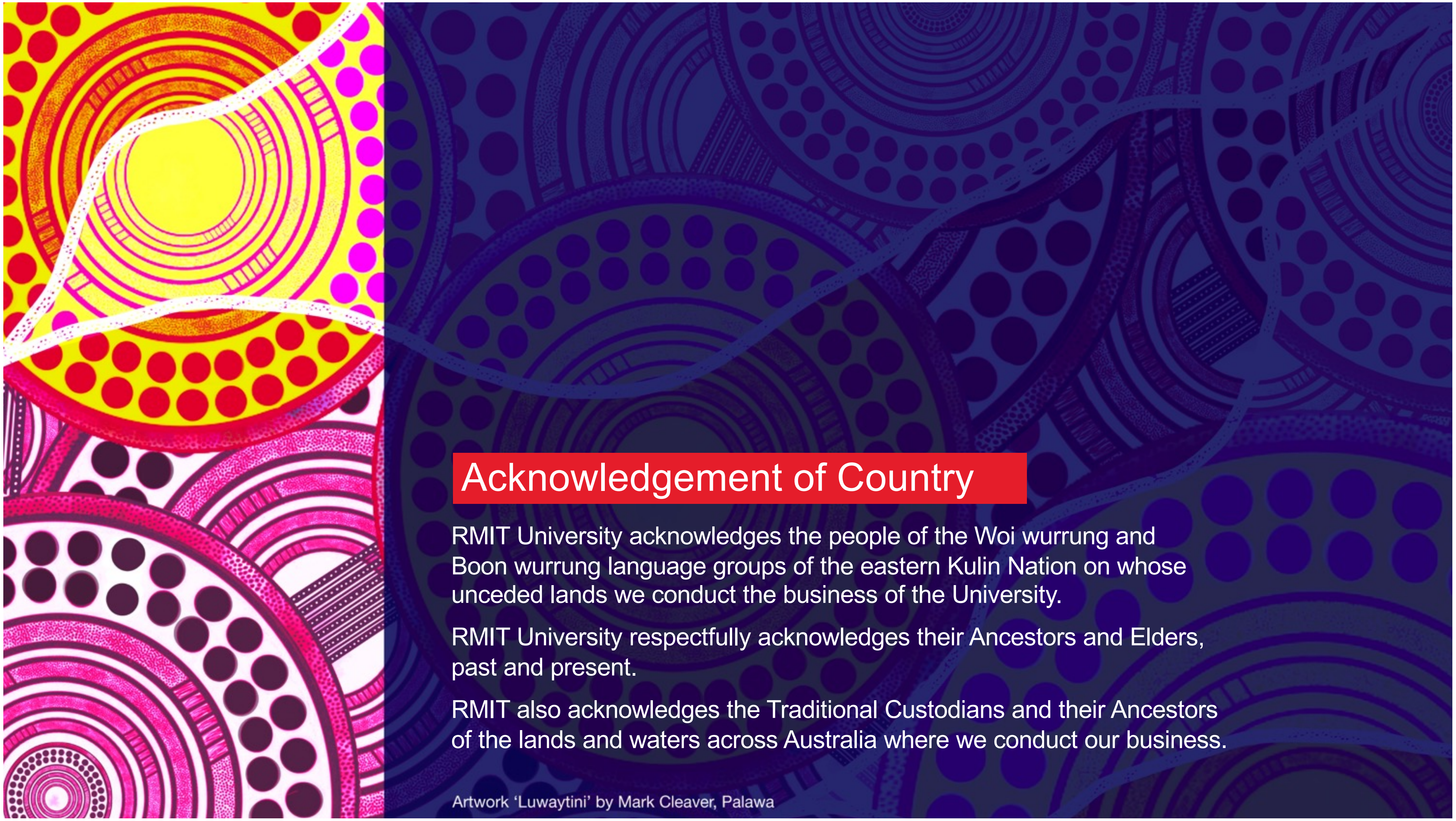
Practical Applications of Artificial Intelligence for Robotics

Day 3: Localisation, Mapping and Navigation

Dr. Timothy Wiley
School of Computing Technologies
RMIT University

ESSAI July 2023



Acknowledgement of Country

RMIT University acknowledges the people of the Woi wurrung and Boon wurrung language groups of the eastern Kulin Nation on whose unceded lands we conduct the business of the University.

RMIT University respectfully acknowledges their Ancestors and Elders, past and present.

RMIT also acknowledges the Traditional Custodians and their Ancestors of the lands and waters across Australia where we conduct our business.

Artwork ‘Luwaytini’ by Mark Cleaver, Palawa

Locomotion

ESSAI July 2023

Wheeled Locomotion

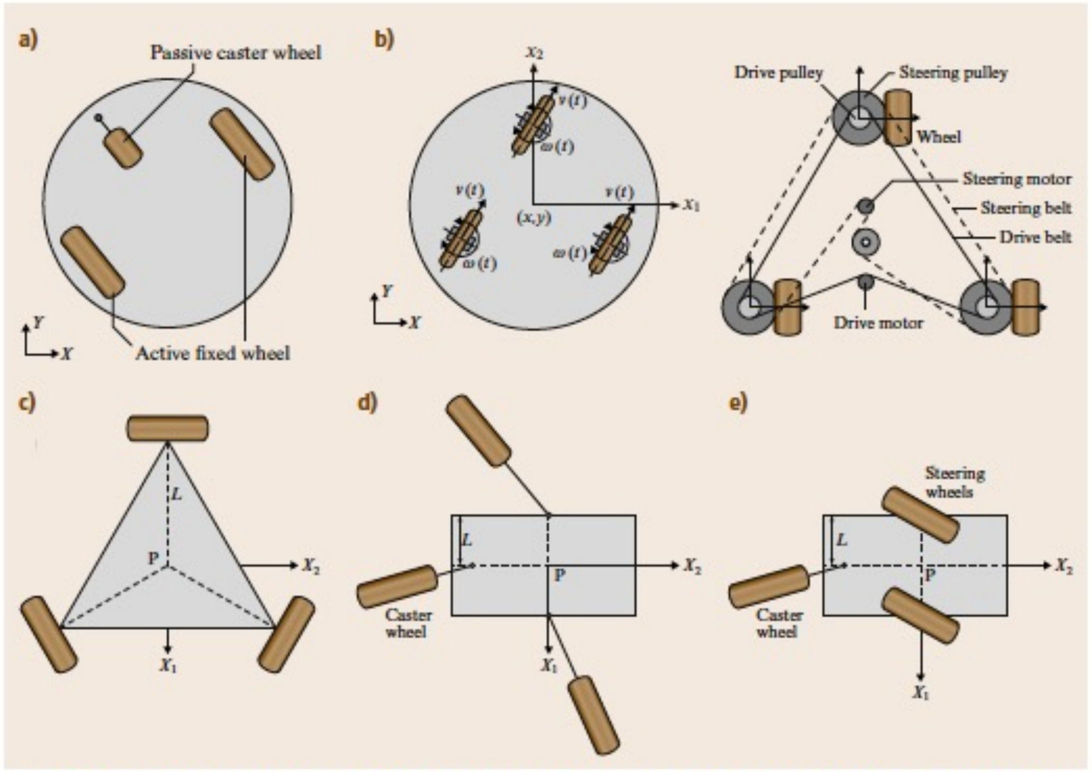


Fig. 24.7 (a) Two-wheel differential drive, (b) synchronous drive, (c) omnimobile robot with Swedish wheels, (d) omnimobile robot with active caster wheels, and (e) omnidirectional robot with active steerable wheels

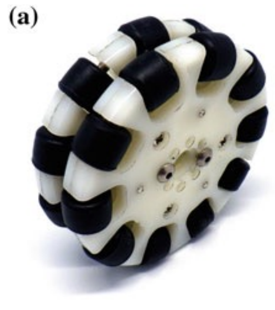


Image: Siciliano & Khatib (Eds), 2016, Springer Handbook of Robotics

Humanoid Locomotion

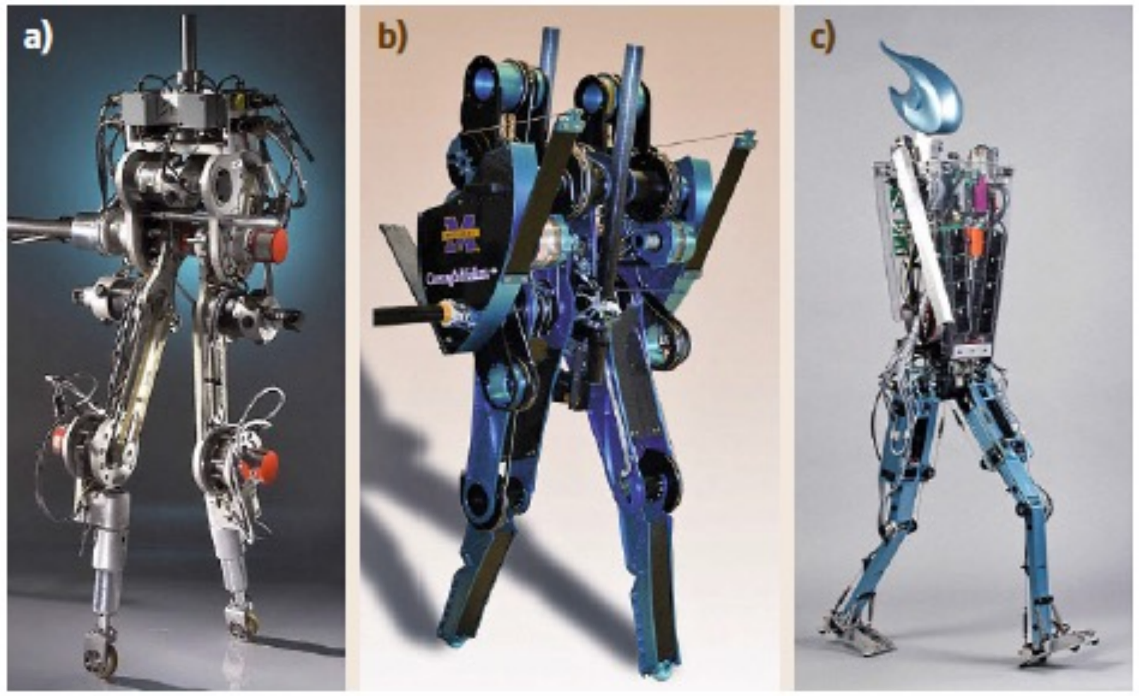


Image: Siciliano & Khatib (Eds), 2016, Springer Handbook of Robotics

Arial Locomotion

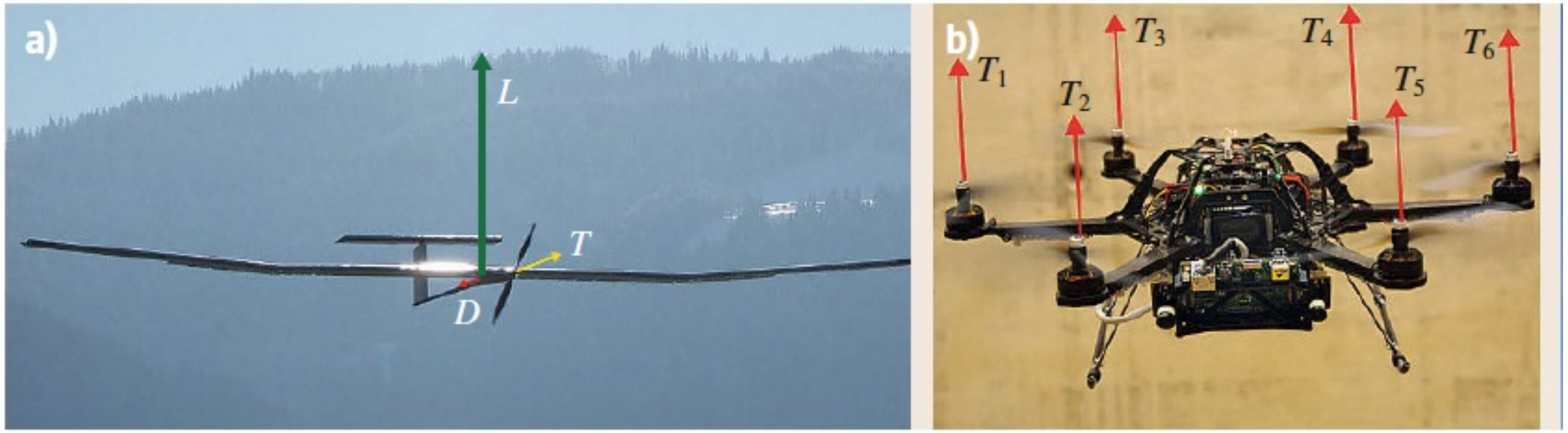


Image: Siciliano & Khatib (Eds), 2016, Springer Handbook of Robotics

ROS: MoveBase

Worked Example with ROSBot

Navigation

ESSAI July 2023

Map

… before Navigation

ESSAI July 2023

Occupancy Grid

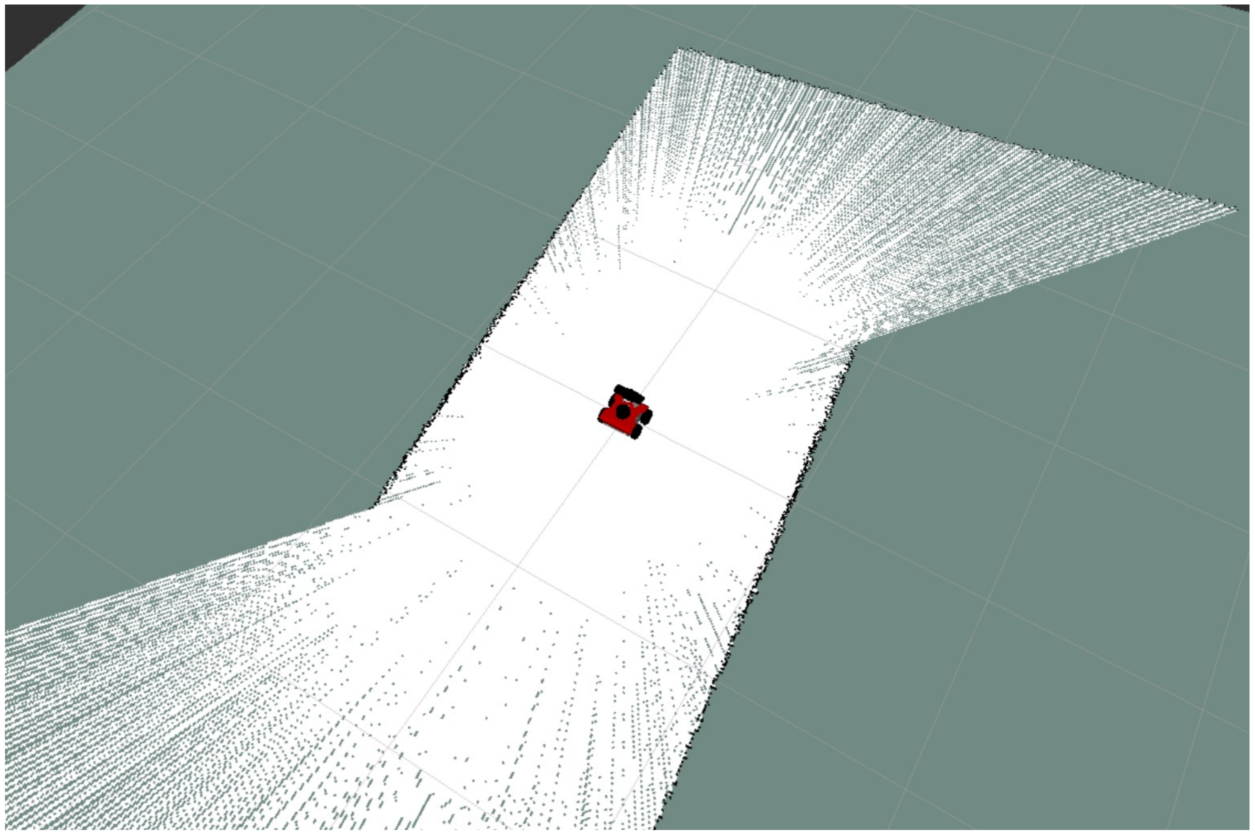
The most common representation of a Map in robotics is a 2D occupancy grid. As the game suggests:

- A 2D grid of cells
- Discretises the environment in the x/y-plane, at a fixed height
- General 5mm (0.005m) resolution (in ROS)
- Each cell has one of two values
  - Unknown
  - Probability of occupied from 100 (occupied) to 0.0 (unoccupied)

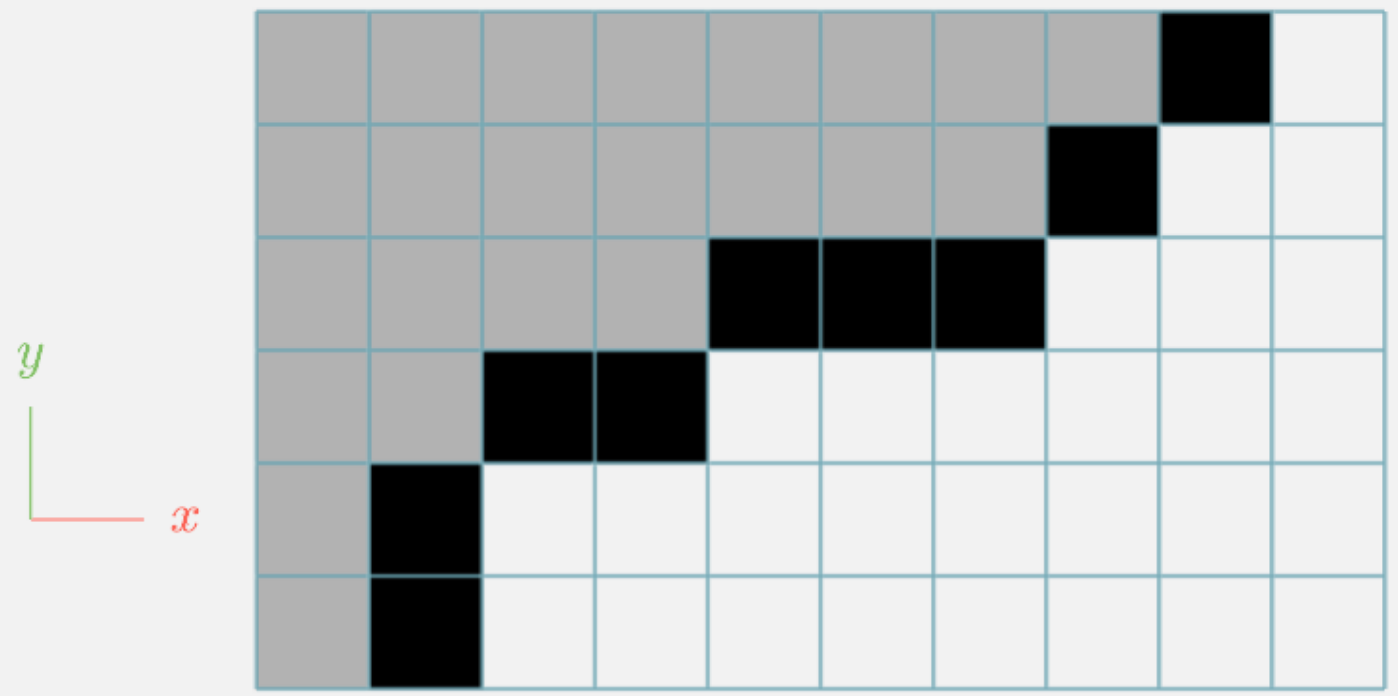
Question: Why have probabilities?

Occupancy Grid

ROS: http://docs.ros.org/en/noetic/api/nav_msgs/html/msg/OccupancyGrid.html



Occupancy Grid



Occupancy Grid

Generally we refer to this OG interchangeably as the “map”.

Maps can be saved & loaded from file with the map server package:

- http://wiki.ros.org/map_server
- Save
  - rosrun map_server map_saver -f room map:=/map
- Load
  - rosrun map_server map_server mymap.yaml

3D Occupancy Grid

Occupancy grids can be expanded to 3D where each cell becomes a 3D voxel (cube).

- 3D maps are, obviously, more computationally expensive
- Generally 2D is used for navigation, and only local 3D information is used when required

Generating Maps

The “live” construction of a Map as the robot moves about an environment is known as:

Simultaneous Localisation and Mapping (SLAM)

This will be covered at the end of this class.

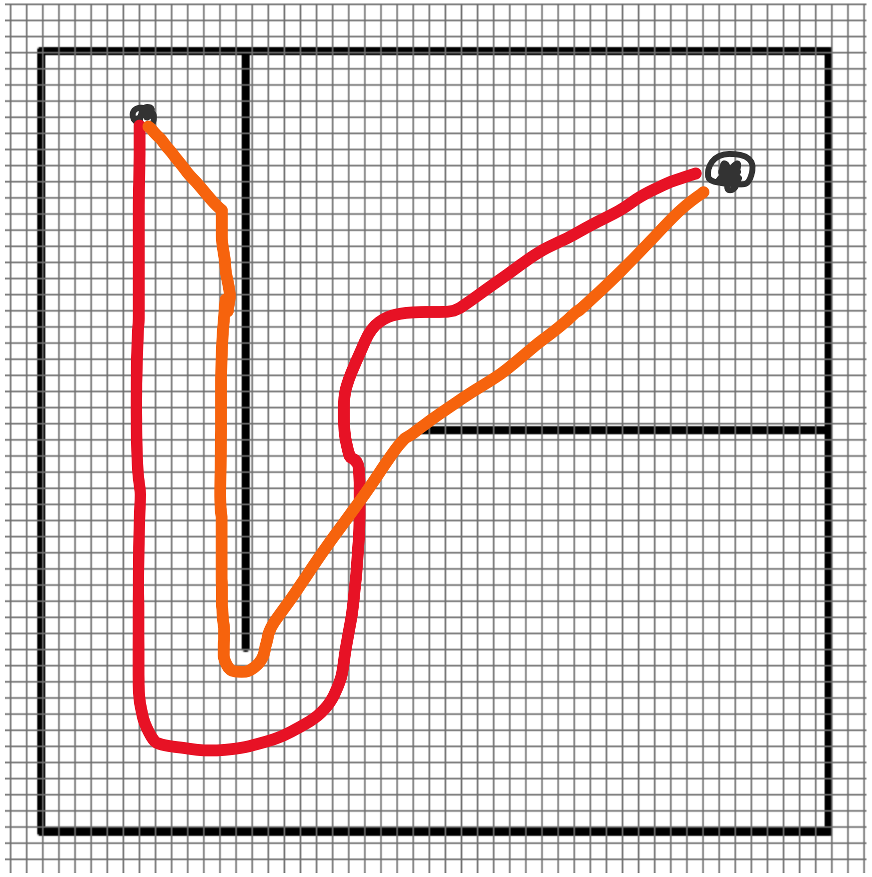
Navigation

... in a map

ESSAI July 2023

Motivation

Given a Map, devise an algorithm navigate between a starting and goal point



A* Navigation

Generally, the obvious answer is A*:

- The Map (Occupancy Grid) is interpreted as a graph
- Cell cost equation: $f(x) = g(x) + h(x)$
  - $g(x)$: shortest path to x
  - $h(x)$: Heuristics are typically:
    - Euclidian
    - Manhattan
  - The “cost” of an individual step is 1

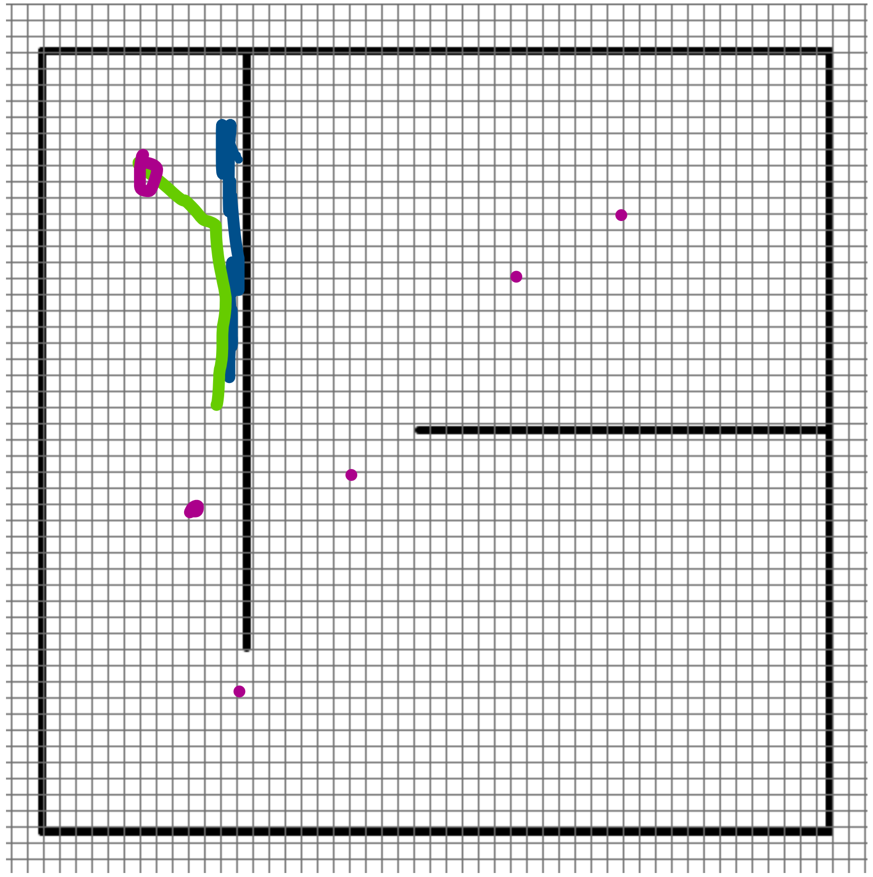
A* Navigation

```
function A_Star(start, goal, h):
    openSet := {start}
    cameFrom := an empty map
    gScore := map with default value of Infinity
    gScore[start] := 0

    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty:
        current := the node in openSet having the lowest fScore[] value
        if current = goal:
            return reconstruct_path(cameFrom, current) // or Done
        openSet.Remove(current)
        for each neighbor of current:
            tentative_gScore := gScore[current] + d(current, neighbor)
           … →
```

```
        … ←
        if tentative_gScore < gScore[neighbor]
            cameFrom[neighbor] := current
            gScore[neighbor] := tentative_gScore
            fScore[neighbor] := tentative_gScore + h(neighbor)
            if neighbor not in openSet
                openSet.add(neighbor)

return failure
```

A* Navigation Example

What does A* do?



Issues with basic A* Navigation

A* will "do the job", but has a number of issues:

- Path does not account for the robot body
- Computed path is not directly traversable
- Path Planning assumes full-observable information
- Not responsive to:
  - Dynamic Obstacles
  - Noise/Error and divergence in execution
- Computationally Expensive, especially for re-planning

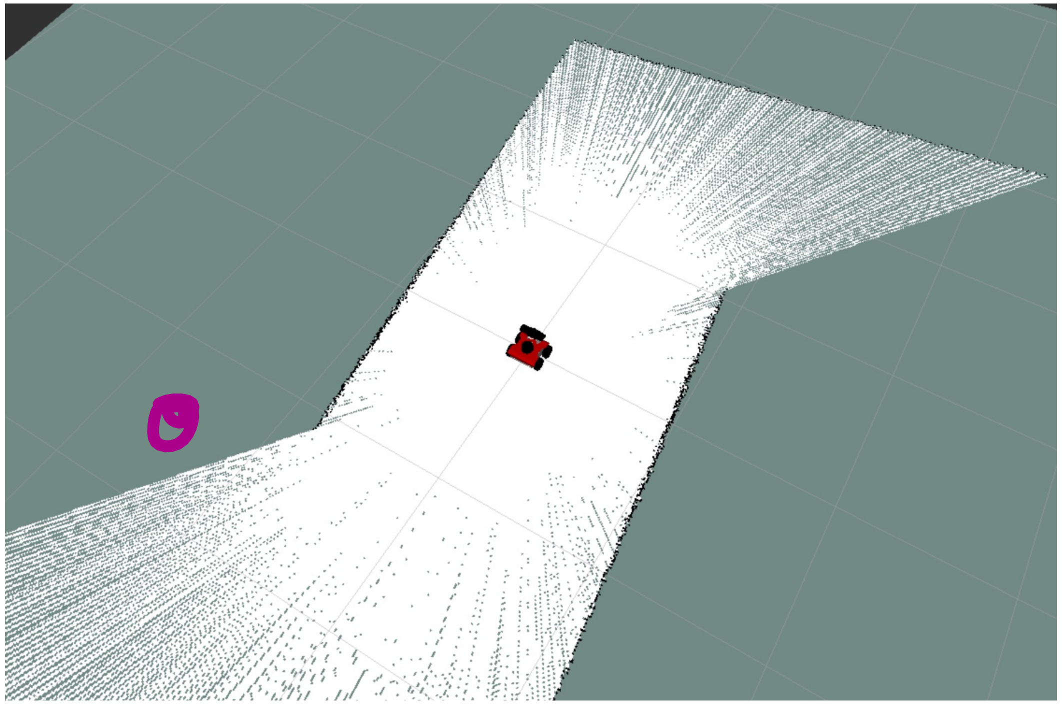
Navigation

CostMaps

ESSAI July 2023

Occupancy Grids are under-described

In an occupancy grid, a cell being unoccupied doesn't mean the robot can actually position itself at that cell.

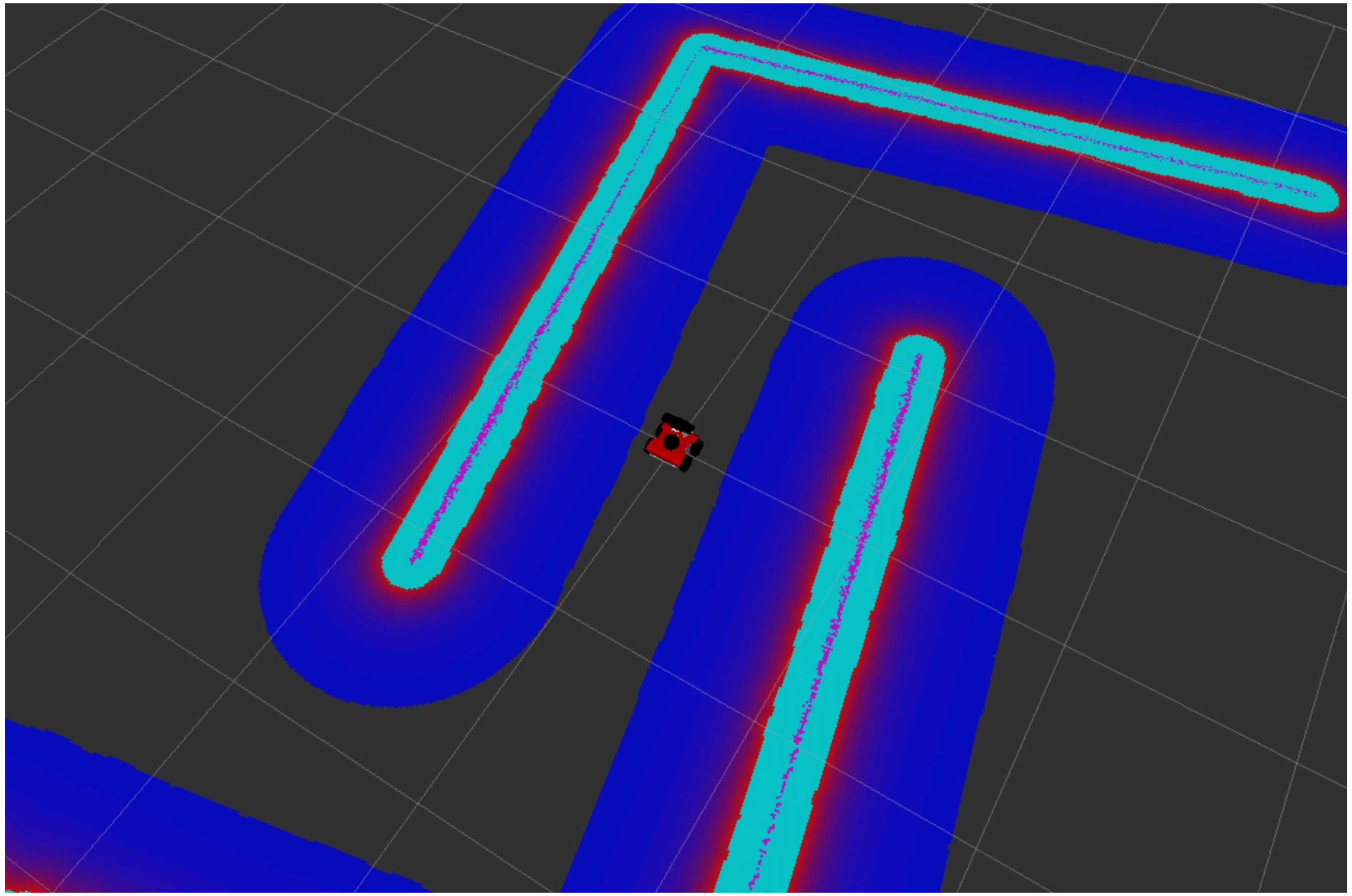


Costmap

A Costmap is a form of occupancy grid, where a cell may be:

- Occupied / Lethal – there is an world-obstacle at the cell, and the robot will hit it
- Inscribed – If the robot is here, it will hit an obstacle
- Dangerous – If the robot is here, it is close to an obstacle and care should be taken
- Free – the robot should be safe here
- Unknown – same as the occupancy grid

Costmap



Costmap Navigation

Navigation (Path Planning) with the CostMap adjusts the weights of the cell cost

- Cell cost equation: $f(x) = g(x) + h(x)$
  - $g(x)$: number of cells traversed to arrive at the current cell
  - + weighted ‘cost’ from the costmap, where free is:
    - Free is 0
    - Dangerous has a positive weight, the larger, the more the dangerous cells will be avoided
    - Unknown space also has a positive weight, so the robot can be allowed to explore unknown space!
  - Lethal/Inscribed cells become non-traversable

Navigation

Multi-Map planning & Re-Planning

ESSAI July 2023

Multiple Map resolutions

Planning in the global map is inefficient, especially for re-planning.

Instead, practical path planning uses multiple maps with :

- Different levels of resolution
  - Global map: course map
  - Local map: fine grained
- Different dimensions
- Different recalculation time frames

CostMaps can also be generated at both local and global views.

Re-Planning

A simple solution to both…

- Dynamic obstacles
- Unobserved spaces

… is replanning, that is, re-calculating the path periodically

However, this can lead to inefficiencies.

Mapping Calculation and Re-Planning Hierarchy

For practical efficiency, at each level, both mapping and re-planning are computed periodically at different frequencies

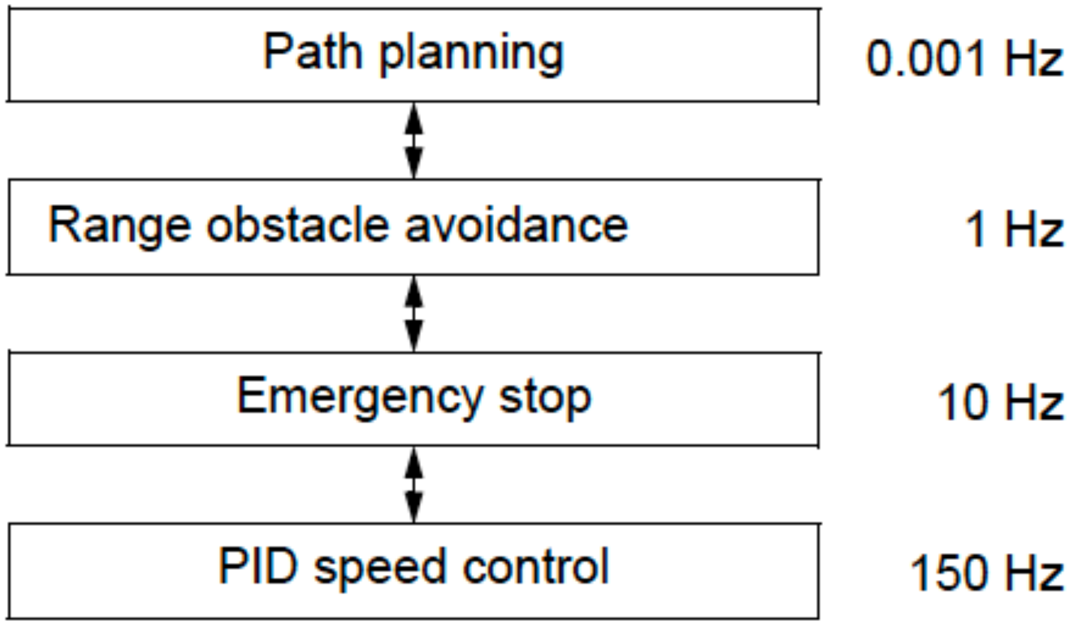


Image: Arkin, 2011, Introduction to Autonomous Mobile Robots

Map Generation at different levels

As noted earlier, the live generation of a “global map” is done through SLAM. However this is also computationally intense.

“Maps” at other levels are generally taken from direct sensor input.

For a typical robot with a laser, this is converting the laser scan directly into an occupancy grid

Worked example with ROSBot on TheConstruct

Obstacle Avoidance

With relatively quick re-planning at the local-map level, dynamic obstacles can be avoided as:

- The local occupancy map (and costmap) are quick to compute
- The dynamic obstacles are generally “small”
- Avoidance can be computed by re-planning in the local map

Worked example with ROSBot

More Complex Navigation

D*/D*-Lite

RRT

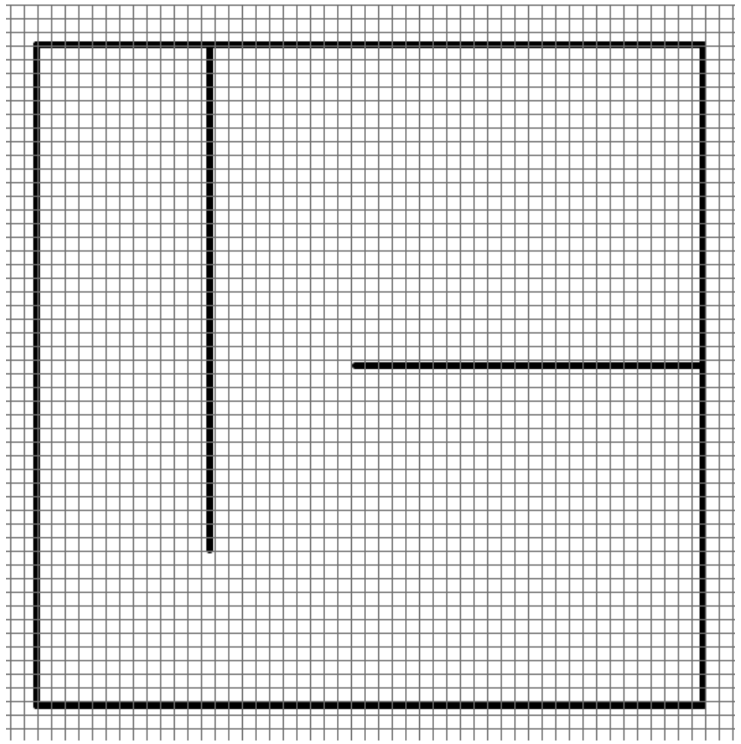
ESSAI July 2023

Wall Following

As silly as it may sound, one of the most effective navigation techniques in a environment is wall-following. Method:

1. Stick you “hand” on a left/right wall
2. Only move around the space following the wall

To overcome “islands” fake walls are added using the path taken in exploration.



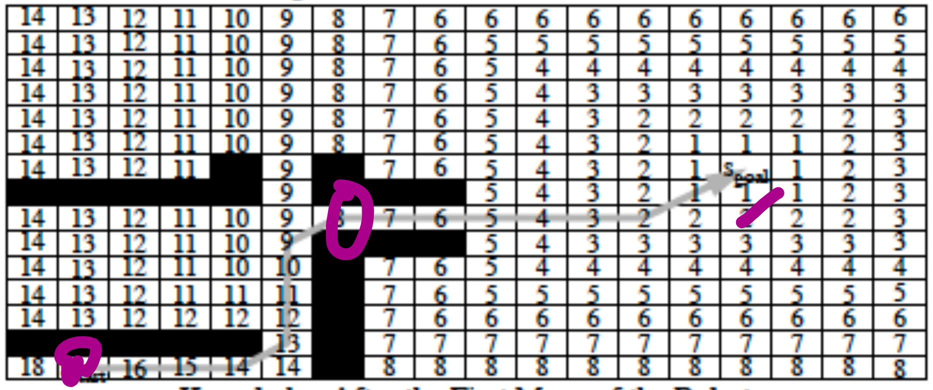
D* Lite

To overcome issues of replanning, the D*/D*-lite family of algorithms by Anthony Stentz, Sven Koenig, and Maxim Likhachev perform incremental replanning (via A*) in a local space around the affected area where new obstacles are detected.

D*-Lite (Koenig & Likhachev 2002) is designed for dynamic re-updates as a robot moves and detects obstacles or updates to the map that affect it’s pre-computed path

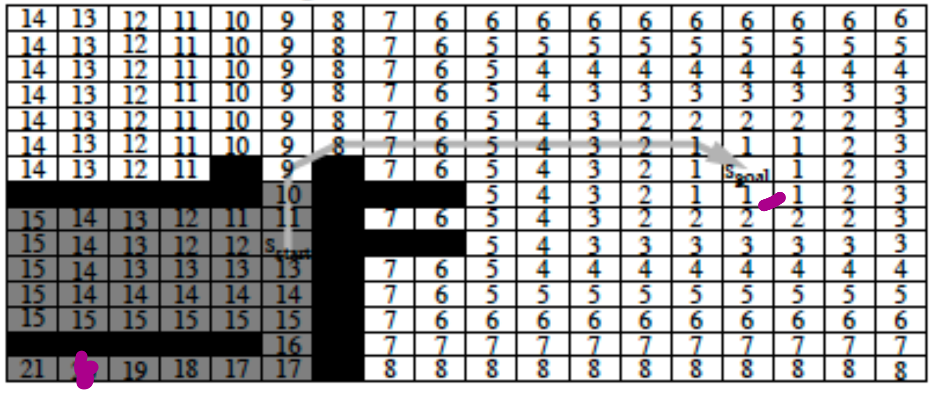
D* Lite: Concept

Knowledge Before the First Move of the Robot



14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11	■	9	■	7	6	5	4	3	2	1	S_{goal}	1	2	3
■	■	■	■	■	9	■	■	■	5	4	3	2	1	1	1	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	■	■	■	5	4	3	3	3	3	3	3	3
14	13	12	11	10	10	■	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	11	11	■	7	6	5	5	5	5	5	5	5	5	5
14	13	12	12	12	12	■	7	6	6	6	6	6	6	6	6	6	6
■	■	■	■	■	13	■	7	7	7	7	7	7	7	7	7	7	7
18	S_{start}	16	15	14	14	■	8	8	8	8	8	8	8	8	8	8	8

Knowledge After the First Move of the Robot



14	13	12	11	10	9	8	7	6	6	6	6	6	6	6	6	6	6
14	13	12	11	10	9	8	7	6	5	5	5	5	5	5	5	5	5
14	13	12	11	10	9	8	7	6	5	4	4	4	4	4	4	4	4
14	13	12	11	10	9	8	7	6	5	4	3	3	3	3	3	3	3
14	13	12	11	10	9	8	7	6	5	4	3	2	2	2	2	2	3
14	13	12	11	10	9	8	7	6	5	4	3	2	1	1	1	2	3
14	13	12	11	■	9	■	7	6	5	4	3	2	1	S_{goal}	1	2	3
■	■	■	■	■	10	■	■	■	5	4	3	2	1	1	1	2	3
15	14	13	12	11	11	■	7	6	5	4	3	2	2	2	2	2	3
15	14	13	12	12	S_{start}	■	■	■	5	4	3	3	3	3	3	3	3
15	14	13	13	13	13	■	7	6	5	4	4	4	4	4	4	4	4
15	14	14	14	14	14	■	7	6	5	5	5	5	5	5	5	5	5
15	15	15	15	15	15	■	7	6	6	6	6	6	6	6	6	6	6
■	■	■	■	■	16	■	7	7	7	7	7	7	7	7	7	7	7
21	20	19	18	17	17	■	8	8	8	8	8	8	8	8	8	8	8

Image: Koenig, Sven, and Maxim Likhatchev. "D lite." Eighteenth national conference on Artificial intelligence. 2002.*

D* Lite: Comparison to A*

The broad differences of D*-Lite to A* are:

- Costs (and heuristics) are computed "in reverse" from the goal to the robot position. That is $g(s_{goal}) = h(s_{goal}) = 0$
- Costs of edges in the traversable graph are tracked for changes
  - A non-traversable edge has cost of ∞
- Maintains a secondary "1-step look ahead" value for each state

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{goal}, \\ \min_{s' \in Pred(s)} (g(s') + c(s', s)) & \text{otherwise} . \end{cases}$$

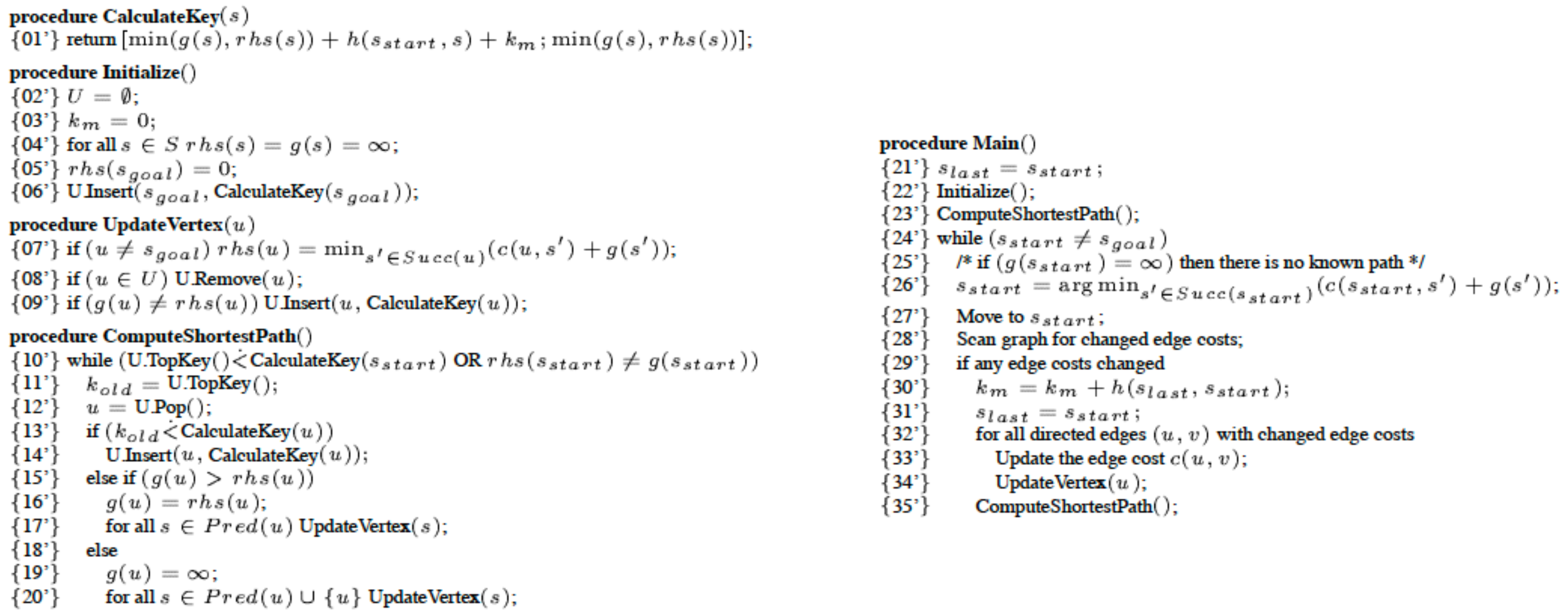
  - This is "more accurate" than $g(s)$ for changing edge costs
- Track k_m - an optimistic estimate of the cost to the goal

D* Lite: Overview

The general overview of D* Lite is:

1. Initialise with computing the shortest path (A*) to the start
2. If a change in the cost of a link is detected:
    1. Use a priority queue open list, U, for node evaluation order
    2. Modify the cost of the node s_{robot}
        1. Recalculate $rhs(s)$
        2. If $g(s) \neq rhs(s)$ add s to U using k_m to modify the priority of the nodes
    3. Recompute the shortest path for all nodes in U, always updating the cost of the node by (2)

D* Lite: Algorithm



```
procedure CalculateKey($s$)
{01'} return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;

procedure Initialize()
{02'} $U = \emptyset$;
{03'} $k_m = 0$;
{04'} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{05'} $rhs(s_{goal}) = 0$;
{06'} U.Insert($s_{goal}$, CalculateKey($s_{goal}$));

procedure UpdateVertex($u$)
{07'} if ($u \neq s_{goal}$) $rhs(u) = \min_{s' \in Succ(u)}(c(u, s') + g(s'))$;
{08'} if ($u \in U$) U.Remove($u$);
{09'} if ($g(u) \neq rhs(u)$) U.Insert($u$, CalculateKey($u$));

procedure ComputeShortestPath()
{10'} while (U.TopKey() $\dot{<}$ CalculateKey($s_{start}$) OR $rhs(s_{start}) \neq g(s_{start})$)
{11'}   $k_{old}$ = U.TopKey();
{12'}   $u$ = U.Pop();
{13'}   if ($k_{old} \dot{<}$ CalculateKey($u$))
{14'}     U.Insert($u$, CalculateKey($u$));
{15'}   else if ($g(u) > rhs(u)$)
{16'}     $g(u) = rhs(u)$;
{17'}     for all $s \in Pred(u)$ UpdateVertex($s$);
{18'}   else
{19'}     $g(u) = \infty$;
{20'}     for all $s \in Pred(u) \cup \{u\}$ UpdateVertex($s$);

procedure Main()
{21'} $s_{last} = s_{start}$;
{22'} Initialize();
{23'} ComputeShortestPath();
{24'} while ($s_{start} \neq s_{goal}$)
{25'}   /* if ($g(s_{start}) = \infty$) then there is no known path */
{26'}   $s_{start} = \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;
{27'}   Move to $s_{start}$;
{28'}   Scan graph for changed edge costs;
{29'}   if any edge costs changed
{30'}     $k_m = k_m + h(s_{last}, s_{start})$;
{31'}     $s_{last} = s_{start}$;
{32'}     for all directed edges $(u, v)$ with changed edge costs
{33'}       Update the edge cost $c(u, v)$;
{34'}       UpdateVertex($u$);
{35'}     ComputeShortestPath();
```

Image: Koenig, Sven, and Maxim Likhachev. "D lite." Eighteenth national conference on Artificial intelligence. 2002.*

Rapidly-exploring Random Tree (RRT)

Both A* and D* have problems:

- Level of discretisation of an environment
- Computational overheads for large spaces
- Computation overhead for replanning
- Execution of navigation at the level of resolution suitable for the robot control

Rapidly-exploring Random Tree (RRT)

RRT is a sampling method for constructing global path plans. It builds a tree of points from the robot location to the goal. Algorithm:

1. Randomly choose a point in the map
2. Find the closest point in the tree to the chosen point
3. Add the chosen point to the tree iff:
   1. The point is within a configurable radius, and
   2. There is no obstacle between the chosen point and tree point
4. Repeat until the space is explored as desired

Rapidly-exploring Random Tree (RRT)

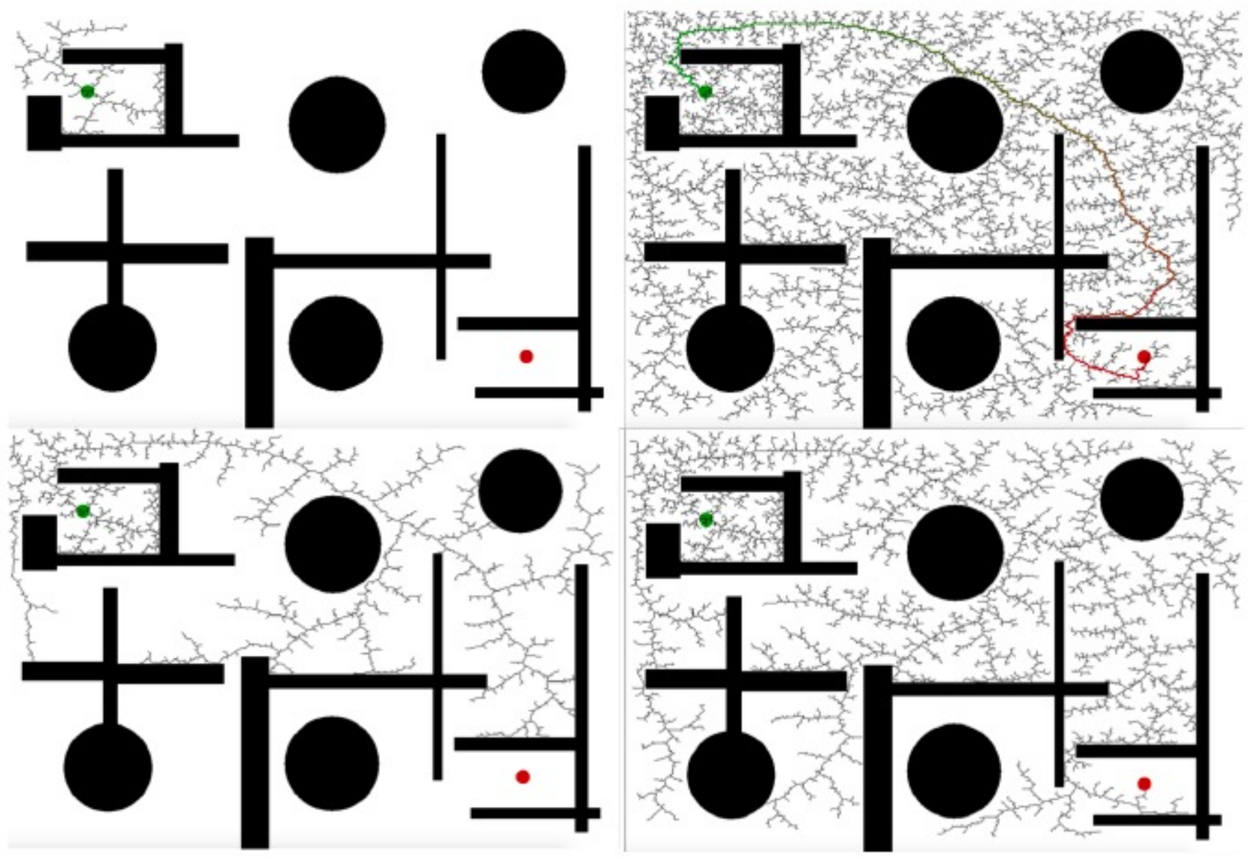


Image: Correll, 2022, introduction to Autonomous Robots

SLAM

Map Generation

ESSAI July 2023

Localisation

… First this

ESSAI July 2023

Probabilistic Localisation

Localisation estimating the robot position in known map given:

- The estimate of the robots previous position
- How the robot moved
- What the robot has observed

This is computing the probability:

$$p(x_t | x_{t-1}, u_t, z_t)$$

- At time: t
- Pose: x_t
- Motion (odometry): u_t
- Measurement: z_t

Bayes Rule

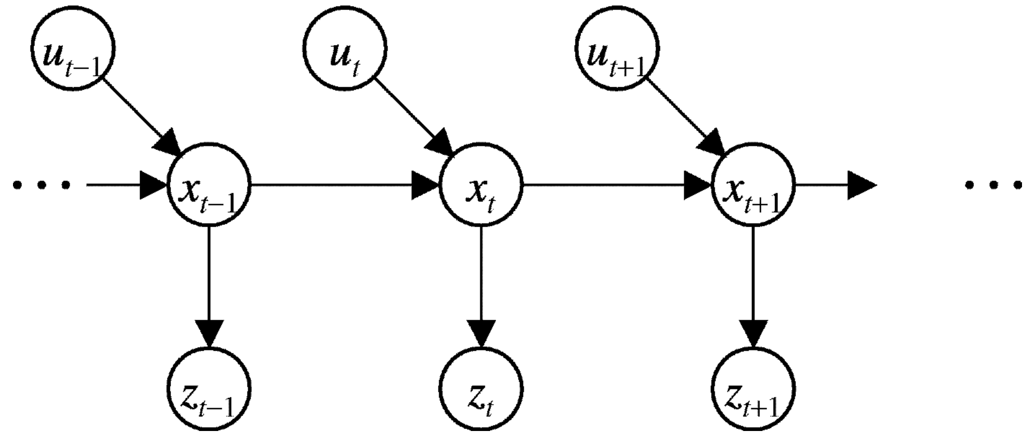
$$P(x, y) = P(x \mid y)P(y) = P(y \mid x)P(x)$$

$$\Rightarrow$$

$$P(x \mid y) = \frac{P(y \mid x)\ P(x)}{P(y)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$$

$$p(x_t | x_{t-1}, u_t, z_t) = \frac{p(x_{t-1}, u_t, z_t | x_t) p(x_t)}{p(x_{t-1}, u_t, z_t)}$$

Markov Assumption



$$p(z_t \mid x_{0:t}, z_{1:t}, u_{1:t}) = p(z_t \mid x_t)$$
$$p(x_t \mid x_{1:t-1}, z_{1:t}, u_{1:t}) = p(x_t \mid x_{t-1}, u_t)$$

Underlying Assumptions

- Static world
- Independent noise
- Perfect model, no approximation errors

Slide: Burgard, Fox & Thrun, 2006, Probabilistic Robotics

Bayes Filters

$$\boxed{Bel(x_t)} = P(x_t \mid u_1, z_1 \ldots, u_t, z_t)$$

Bayes $$= \eta\ P(z_t \mid x_t, u_1, z_1, \ldots, u_t)\ P(x_t \mid u_1, z_1, \ldots, u_t)$$

Markov $$= \eta\ P(z_t \mid x_t)\ P(x_t \mid u_1, z_1, \ldots, u_t)$$

Total prob. $$= \eta\ P(z_t \mid x_t) \int P(x_t \mid u_1, z_1, \ldots, u_t, x_{t-1})\ P(x_{t-1} \mid u_1, z_1, \ldots, u_t)\ dx_{t-1}$$

Markov $$= \eta\ P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\ P(x_{t-1} \mid u_1, z_1, \ldots, u_t)\ dx_{t-1}$$

Markov $$= \eta P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\ P(x_{t-1} \mid u_1, z_1, \ldots, z_{t-1})\ dx_{t-1}$$

$$\boxed{= \eta\ P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\ Bel(x_{t-1})\ dx_{t-1}}$$

Slide: Burgard, Fox & Thrun, 2006, Probabilistic Robotics

Bayes Rule

Markov Assumption and Conditional Independence allows the calculation of the “belief” in the robot’s pose: $Bel(x_t)$

to a two-part method:

- Prediction: $\overline{Bel(x_t)} = \int p(x_t|u_t, x_{t-1})Bel(x_{t-1})dx$
- Correction: $Bel(x_t) = \eta p(z_t|x_t)\overline{Bel(x_t)}$
    - Where: η is a normalisation term
    - Note, the term belief is used as a synonym for the state probability

Bayes Rule: Decomposition

Why does it breakdown?

- Conditional independence gives that with complete information, two different probabilities are sufficient to give the full state
  - State Transition Probability:
    $$p(x_t \mid x_{t-1}, u_t, z_t) = p(x_t \mid x_{t-1}, u_t)$$
  - Measurement Probability: $p(x_t \mid x_{t-1}, u_t, z_t) = p(z_t \mid x_t)$
- With incomplete information, we can combine these in the prediction and correction steps

The prediction and correction (measurement) steps are a core part of all localisation and mapping algorithms

Bayes Rule: Example

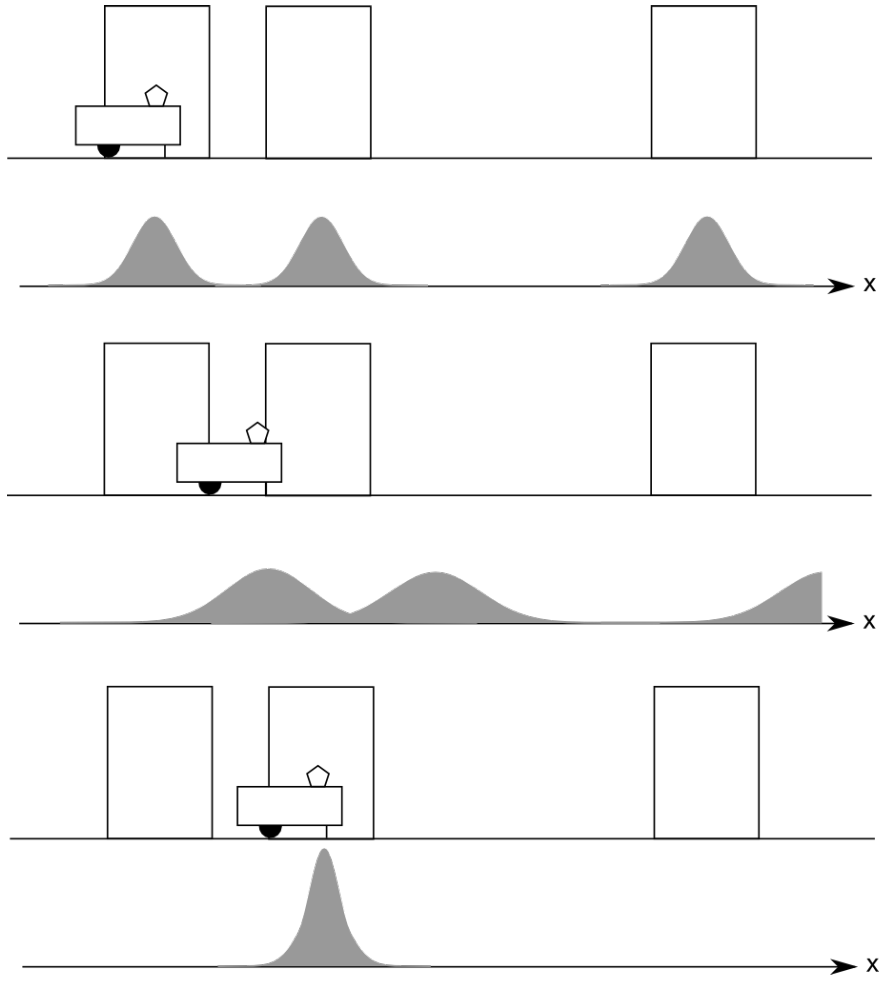


Image: Correll, 2022, introduction to Autonomous Robots

Diversions

Odometry &
ICP Matching

ESSAI July 2023

Odometry

Odometry is the estimation of the robot's pose based only on the actuators of the robot.

For example, on the ROSBot, odometery is computed using the encoder to estimate the "joint" position.

- Measures the rate at which the wheel turns
  - Estimate position of the robot from starting position
- Accurate, only if:
  - Encoder reliable
  - Time reliable
  - No slip
- Still drifts over time

Odometry

In ROS:

- If a robot supports Odometry, then the frame typically is /odom
- The ROSBot software launches with an odometry measurement

ICP: Iterative Closet Point Matching

For localisation (using laser scans and occupancy grids) a way to “match up” a laser scan to an occupancy grid is required.

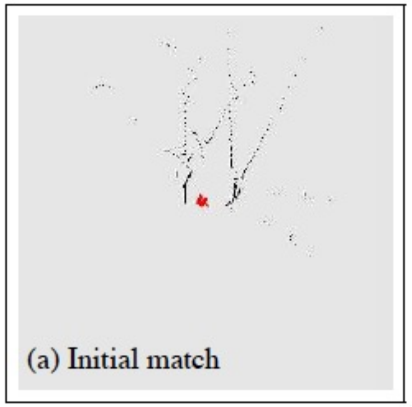
ICP (Iterative Closest Point) laser scan matching is a technique to align two or more scans. The goal is to estimate the transformation (i.e., rotation and translation) that relates the scans to each other.

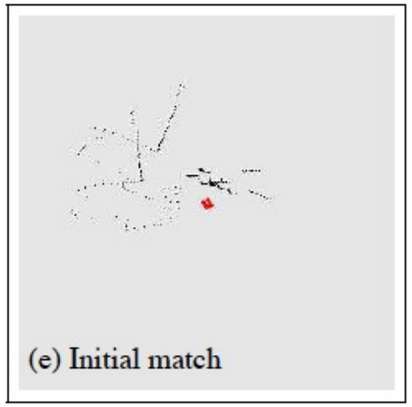
ICP: Iterative Closet Point Matching

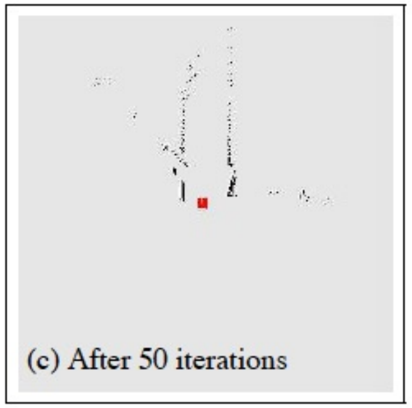
ICP laser scan matching works by:

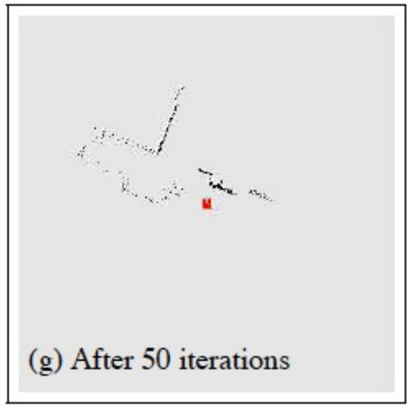
1. Identifying corresponding points between two scans, which are typically found by searching for the nearest point in one scan to each point in the other scan.
2. Iteratively updates the transformation estimate to minimize the distance between the corresponding points. There are many ways to transform the pose:
    1. Gradient descent
    2. Nonlinear optimization technique, such as the Gauss-Newton method, which adjusts the transformation estimate to reduce the overall distance between the points.

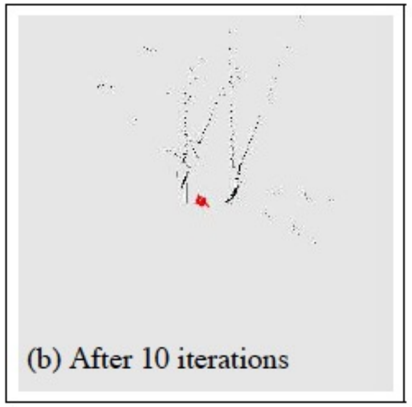
ICP: Iterative Closet Point Matching

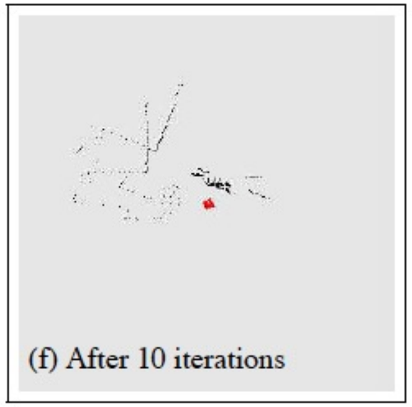

(a) Initial match

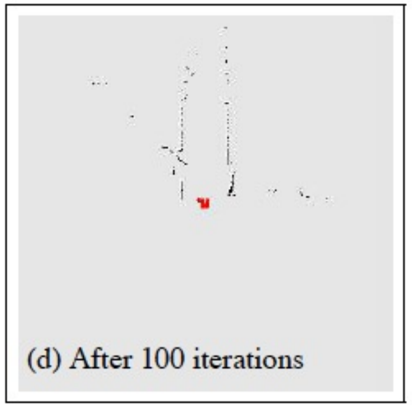

(e) Initial match


(c) After 50 iterations


(g) After 50 iterations


(b) After 10 iterations


(f) After 10 iterations


(d) After 100 iterations

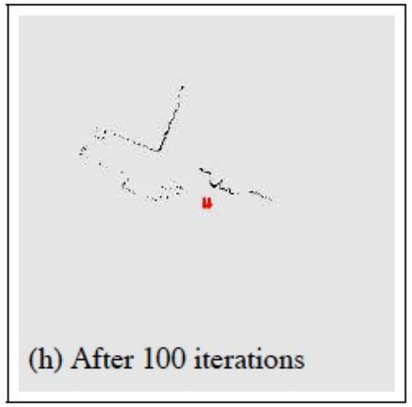

(h) After 100 iterations

Image: Burgard, Fox & Thrun, 2006, Probabilistic Robotics

Localisation

Particle Filter

ESSAI July 2023

Particle Filter Localisation

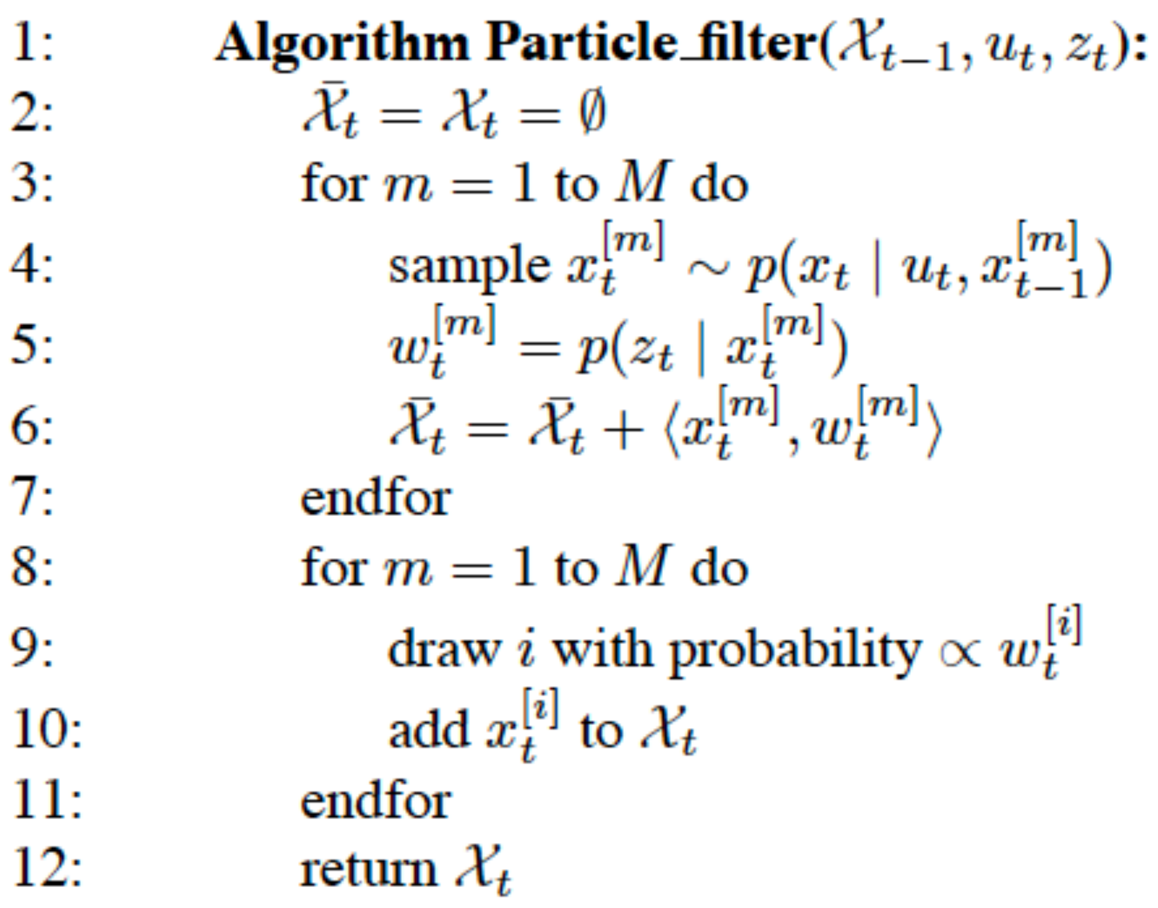
The Particle Filter approach does away with trying to measure and calculate the probabilities.

Instead it uses a random sampling approach via “particles”. The assumption is that with enough sampling the estimate of the robot’s pose can be found to within an acceptable error margin.

The particle filter follows the same two step process:

1. Predict where the robot has gone creating new particles
2. Correct and Update the most likely particles

Particle Filter Localisation: Algorithm



```
1:  Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):
2:      $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3:      for $m = 1$ to $M$ do
4:          sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
5:          $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
6:          $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7:      endfor
8:      for $m = 1$ to $M$ do
9:          draw $i$ with probability $\propto w_t^{[i]}$
10:         add $x_t^{[i]}$ to $\mathcal{X}_t$
11:     endfor
12:     return $\mathcal{X}_t$
```

Image: Burgard, Fox & Thrun, 2006, Probabilistic Robotics

Particle Filter Localisation: Algorithm

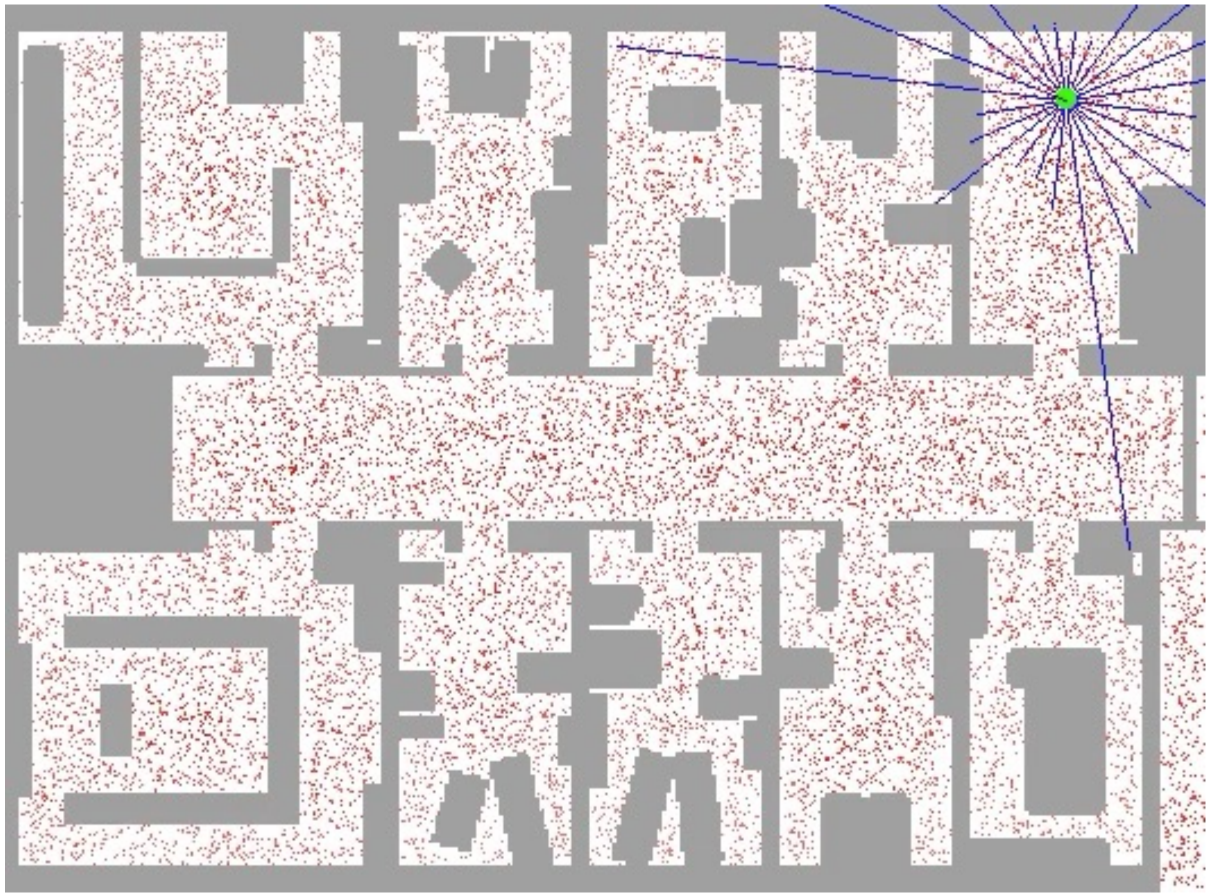
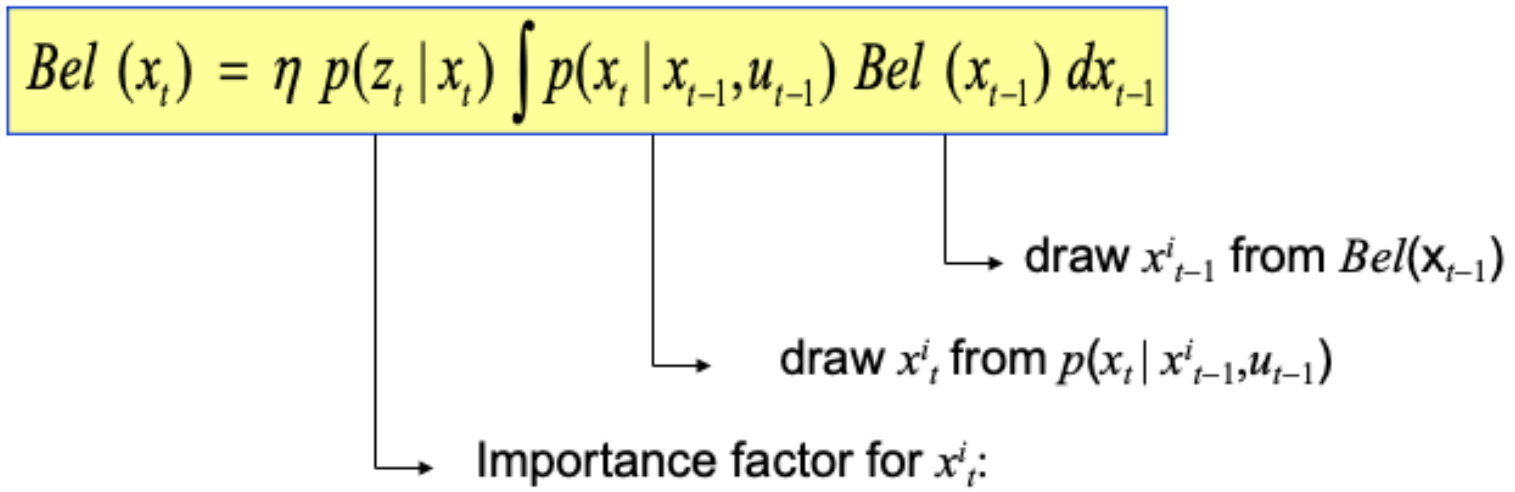


Image: Burgard, Fox & Thrun, 2006, Probabilistic Robotics

Particle Filter Localisation: Algorithm

An interesting property is that the particle filter maps to the bayes estimation:



$$Bel(x_t) = \eta\, p(z_t \mid x_t) \int p(x_t \mid x_{t-1}, u_{t-1})\, Bel(x_{t-1})\, dx_{t-1}$$

draw x^i_{t-1} from $Bel(x_{t-1})$
draw x^i_t from $p(x_t \mid x^i_{t-1}, u_{t-1})$
Importance factor for x^i_t:

Image: Burgard, Fox & Thrun, 2006, Probabilistic Robotics

SLAM

Map Generation

ESSAI July 2023

Simultaneous Localisation and Mapping

It is as the name suggests SLAM creates a map of the environment, at the same time as determining the robot's position within the map

The two cannot be separated, since generating the map from the robot's sensor observations requires knowledge of where the robot is within said map!

Simultaneous Localisation and Mapping

Thus SLAM estimates what the map should look like. SLAM is an extension to the localisation equation:

$$p(x_t, m | x_{t-1}, u_t, z_t)$$

- where m is the map

FastSLAM

FastSLAM is a particle filter-based SLAM approach. In FastSLAM:

- Each particle represents a possible location and map of the environment.
- As the robot moves, each particle is updated to refine its estimates of the location and map
- Particles are resampled based on an error measurement of both the location and map step

FastSLAM

FastSLAM uses "landmarks" in the environment. Landmarks are stored in a tree structure

- Each node in the tree represents a cluster of landmarks that are close together
- The root node represents the entire environment
- Each level of the tree represents a smaller region of space
- When a new landmark is detected, it is assigned to the node that corresponds to its location.
  - If the node does not yet, a new node is created.
  - As the robot moves and observes more landmarks, the nodes in the tree are updated.

FastSLAM

The key advantages of the landmark tree is that only a small part of the map needs to be updated without having to recompute the entire map every time new laser data is received. Only the affected nodes need to be updated.

FastSLAM

(a) (b) (c)

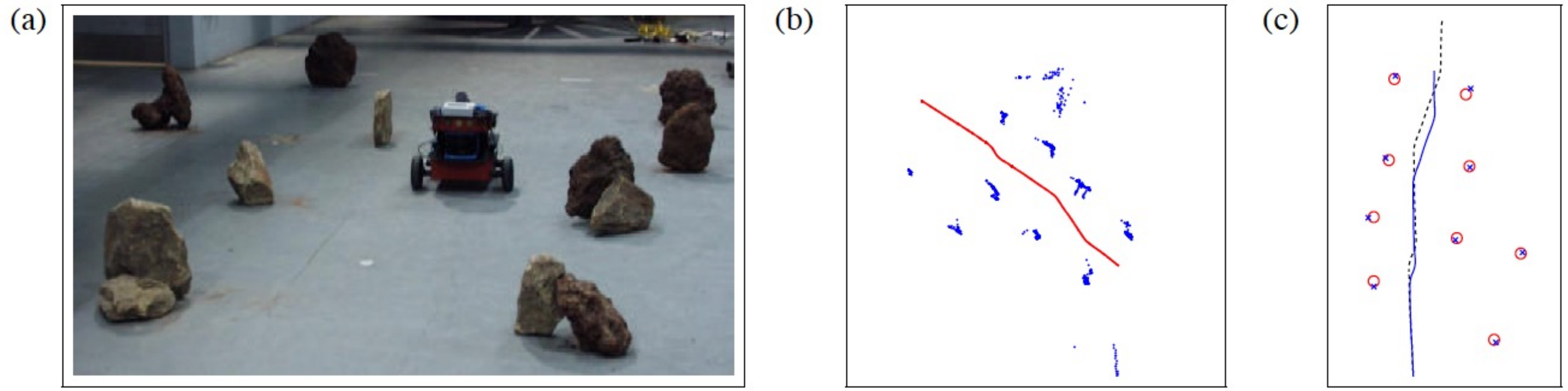
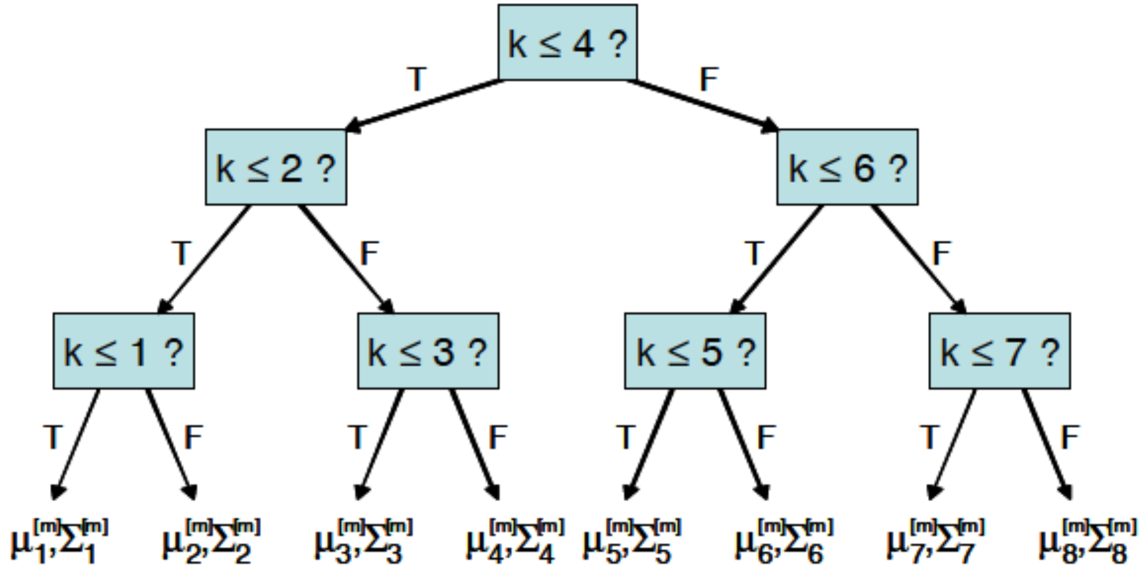


Figure 4: (a) Physical robot mapping rocks, in a testbed developed for Mars Rover research. (b) Raw range and path data. (c) Map generated using FastSLAM (dots), and locations of rocks determined manually (circles).

Images: Montemerlo, Michael, et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem." AAAI (2002)

FastSLAM



k ≤ 4 ?
T
F
k ≤ 2 ?
k ≤ 6 ?
T
F
T
F
k ≤ 1 ?
k ≤ 3 ?
k ≤ 5 ?
k ≤ 7 ?
T
F
T
F
T
F
T
F
$\mu_1^{[m]}, \Sigma_1^{[m]}$
$\mu_2^{[m]}, \Sigma_2^{[m]}$
$\mu_3^{[m]}, \Sigma_3^{[m]}$
$\mu_4^{[m]}, \Sigma_4^{[m]}$
$\mu_5^{[m]}, \Sigma_5^{[m]}$
$\mu_6^{[m]}, \Sigma_6^{[m]}$
$\mu_7^{[m]}, \Sigma_7^{[m]}$
$\mu_8^{[m]}, \Sigma_8^{[m]}$

Figure 2: A tree representing $K = 8$ landmark estimates within a single particle.

Images: Montemerlo, Michael, et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem." AAAI (2002)

Loop Closure

Loop closure occurs when the robot revisits the same part of the environment twice, and the map should "close the loop" of map, that is, ensure the walls (or features) of the map align at the location where the robot revisited.

Loop closure is challenging because:

- Odometry errors from robot motor encoders
- Estimation errors in map computation
- Mapping algorithms that perform partial map updates

Loop Closure

Generally, most SLAM methods fail at proper loop closure unless it is explicitly handled by the algorithm:

The landmark tree in FastSLAM allows errors in landmarks when loop closure is detected to be "distributed" among the landmark tree.

FastSLAM: Loop Closure

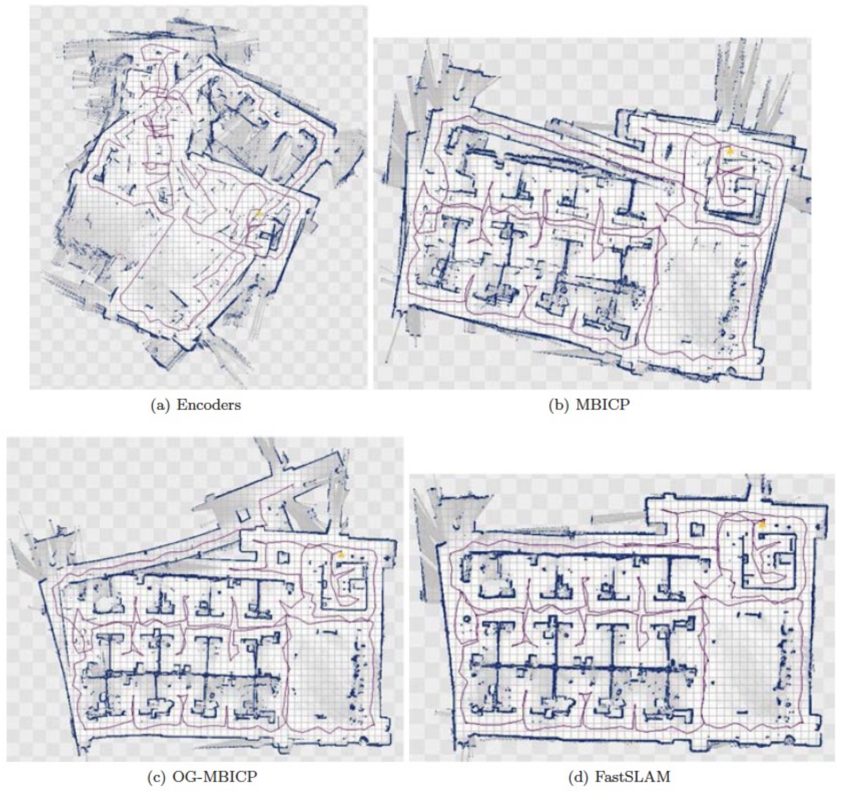

(a) Encoders (b) MBICP (c) OG-MBICP (d) FastSLAM

Figure 3. Full map of office environment.

Images: Milstein, A., McGill, M., Wiley, T., Salleh, R. & Sammut, C. A Method for Fast Encoder-Free Mapping in Unstructured Environments. Journal of Fields Robotics, 28, 817–831 (2011)

GraphSLAM

GraphSLAM is a graph-based approach to SLAM that uses a graph representation of the map, which is then optimised as loops are detected. In GraphSLAM:

- Nodes of the graph are small localmaps that include a pose of the map, and map landmarks
- Edges represent kinematic displacements between the localmaps.
- In each localmap, ICP can be used to align new laser scans, and build-up the local map
- A new localmap (and node) is created if:
  - the robot moves too far to be outside a localmap
  - The ICP alignment error is too great

GraphSLAM

Loop closure is specifically detected by comparing the current localmaps to all other localmaps in the graph

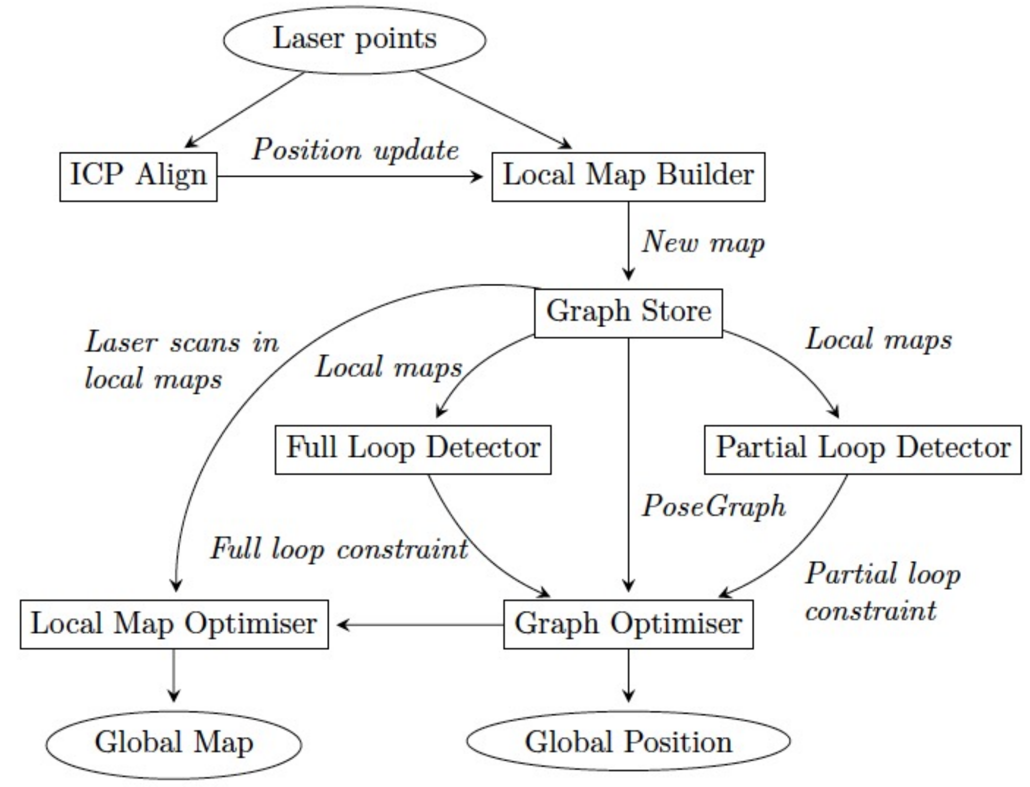
GraphSLAM

Loop closure is specifically detected by comparing the current localmaps to all other localmaps in the graph:

- When a loop is detected, a loop constraint is added to the graph
- A pose error is computed between the localmaps (for both displacement and rotation)
- This error is then "smoothed-out" or distributed between all edges of the graph to refine the kinematic transform between each localmap
- This results in a more accurate map

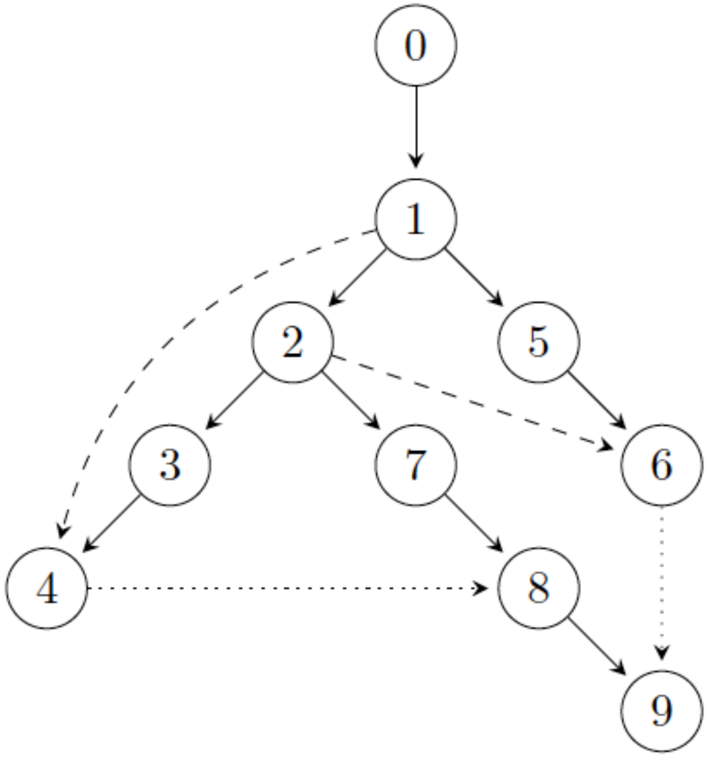
However, GraphSLAM is reliant on loop-closure to correct for mapping errors.

GraphSLAM



Images: Ratter, A. & Sammut, C. Local Map Based Graph SLAM with Hierarchical Loop Closure and Optimisation. in (2015).

GraphSLAM



Images: Ratter, A. & Sammut, C. Local Map Based Graph SLAM with Hierarchical Loop Closure and Optimisation. in (2015).

Noon Gudgin

Thank you

Day 4: Robotic Vision

ESSAI July 2023