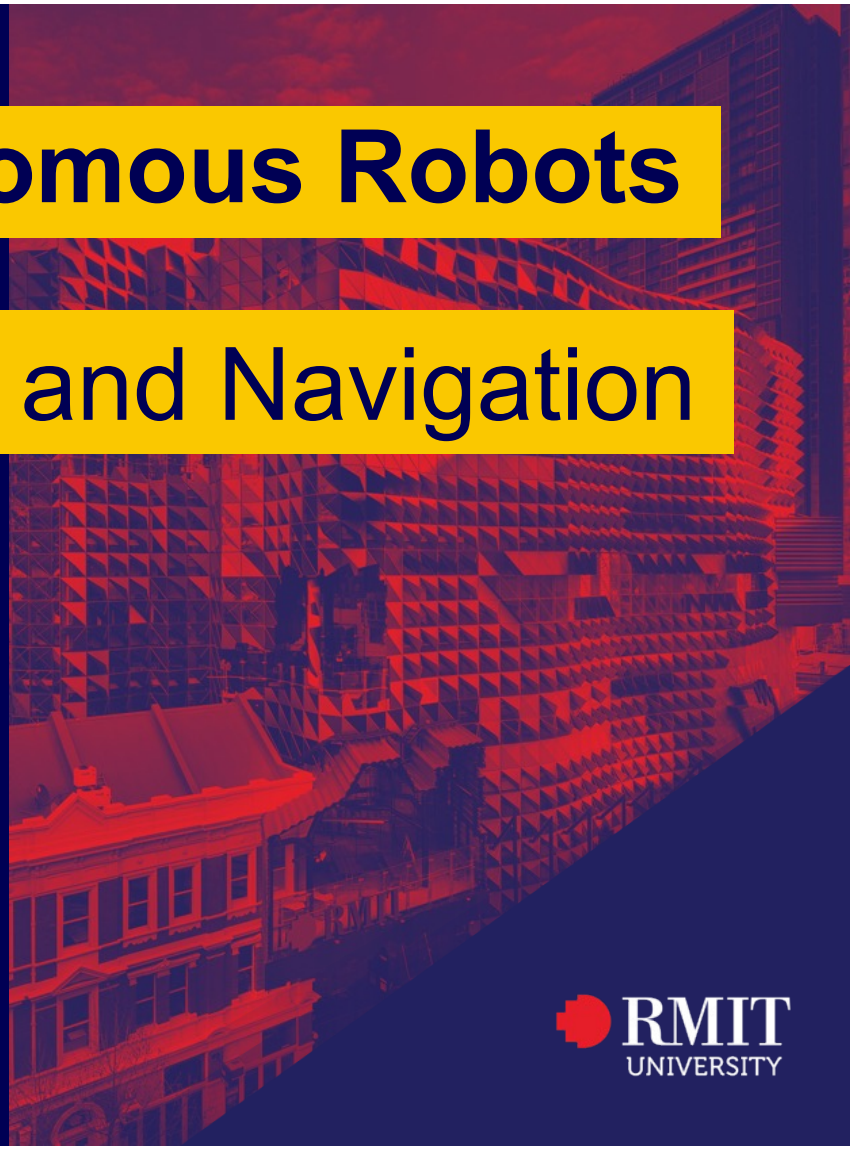
Practical AI for Autonomous Robots

Day 2: Motion Planning and Navigation

Dr. Timothy Wiley
School of Computing Technologies
RMIT University



ESSAI July 2025

RMIT UNIVERSITY

Motivation

My work in learning control

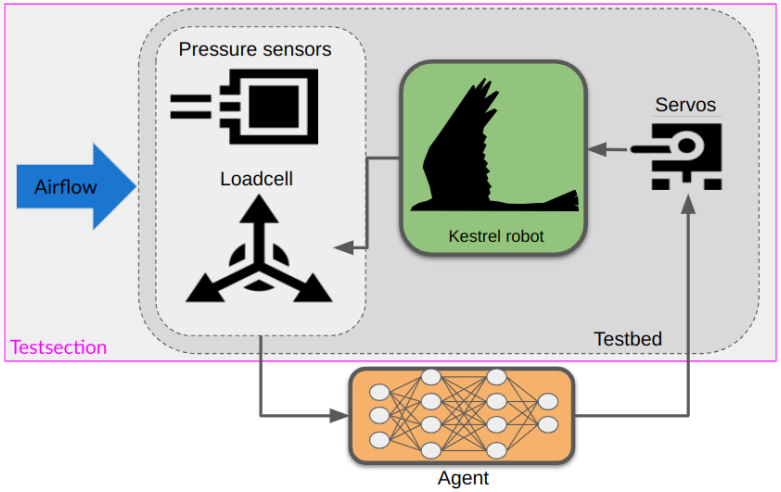
ESSAI July 2025

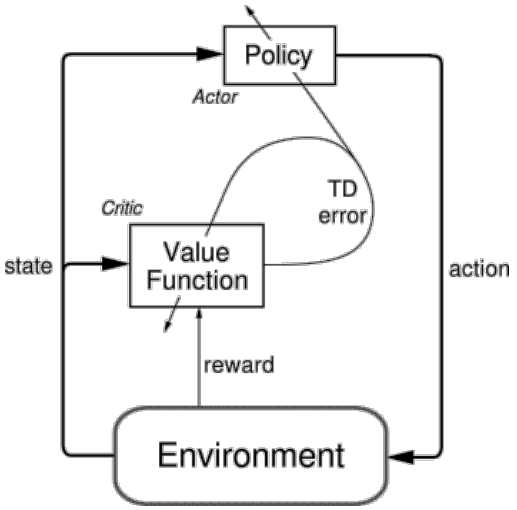
RMIT UNIVERSITY

Control of Bio-Inspired UAV

Online learning of robot control in real-world testing:

- Embedding action operators into the state space + action space
- Minimal shaping of reward function
- Actor-Critic RL structure

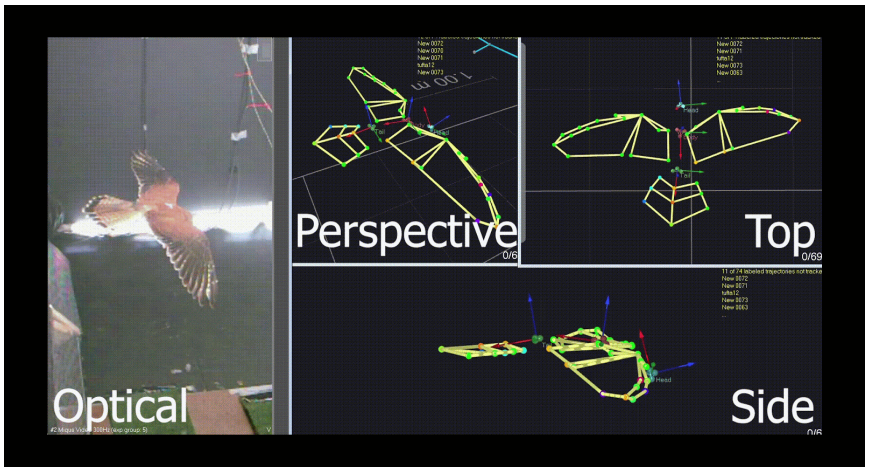




Control of Bio-Inspired UAV

Online learning of robot control in real-world testing:

- Embedding action operators into the state space + action space
- Minimal shaping of reward function
- Actor-Critic RL structure



Online Deep Reinforcement Learning

Kinematics

ESSAI July 2025

RMIT UNIVERSITY

Kinematics – Motivation

On Day 1, we finished by looking at Reinforcement Learning for complex control of robot systems.

We ‘hand-waved’ the state and action spaces of learning, as the methods we discussed largely function at the control level.

Other forms of control, along with motion planning techniques, navigation, and many other algorithms we’ll study require understanding the kinematic model of a robot.

Frames of Reference

ESSAI July 2025

RMIT UNIVERSITY

Frames of Reference & Poses

To describe poses a Frame is defined by:

1. Origin Point
2. 3D axes orientation
3. Following Right-hand Rule

A Pose is the tuple:

$$[position, orientation]$$

$$[x, y, z, roll, pitch, yaw]$$

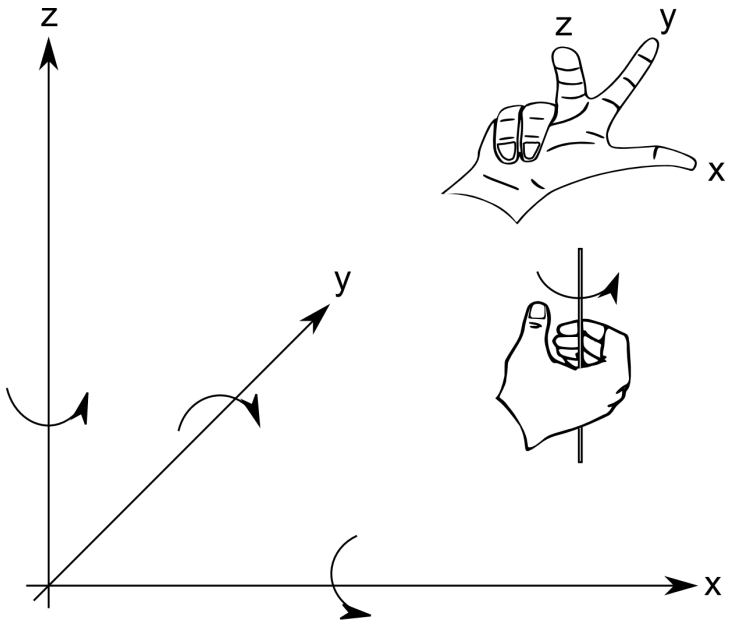


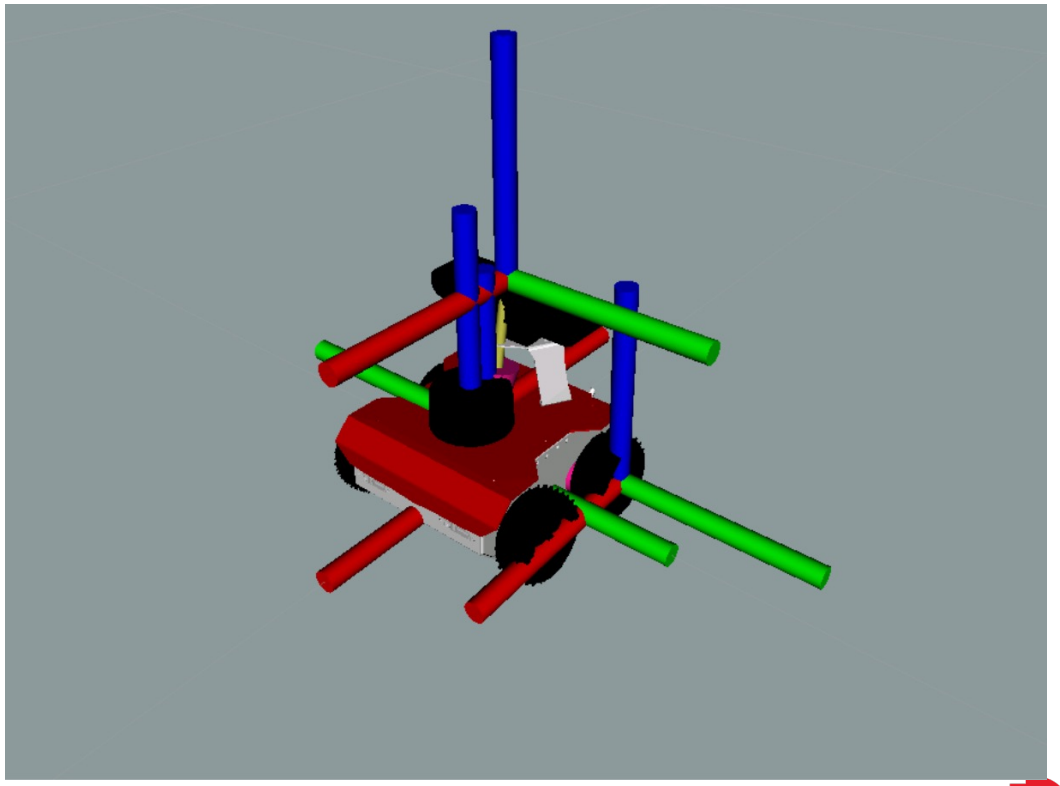
Image: Correll, 2022, introduction to Autonomous Robots

ROS Frames & Conventions

X – Red

Y – Green

Z - Blue



ROSBot Frames & Conventions

Conventionally:

- The “global” frame is /map
  - The x/y-axis parallel to the ground-place
  - The z-axis is vertical
- The default frame of robot is /base_link with
  - The x-axis is the ‘forward’ direction of the robot
  - The z-axis is vertical

Transforming Between Frames (links)

Defined by Linear Algebra Transformation with Homography Matrices

Point:

$$ {}^{A}P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} $$

Translation:

$$ T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Rotations

$$ R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Rotation representations

Euclidean (technically XYX transforms, thought this applies to all forms) are generally "easier" to understand and visualise as the matrices map directly to real-world concepts. However, they have a problem - some combinations of rotations result in singularities

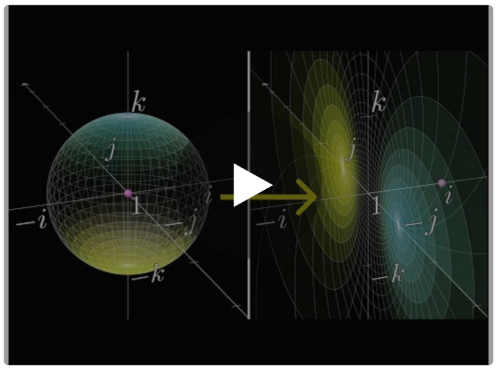
Singularities are places where there are not unique descriptions (mathematically) for a kinematic chain (set of transforms)

Singularities result in 'infinite solutions' for some joint positions when solving for inverse kinematics - that is, working out which transforms (ie joint angles) are required to reach a particular robot configuration

A common replacement for rotations are Quaternions

Ben Eater interactive video:

https://eater.net/quaternions



Transforming Between Frames (links)

Transforming a Point, Vector (or Pose) between frames:

$$^{W}P = {}^{W}_{R}T \times {}^{R}P$$

Transformation Matrices can be multiplied, but order matters

$${}^{W}_{R}T = T_x \times T_y \times R_x$$

Transforming Between Frames (links)

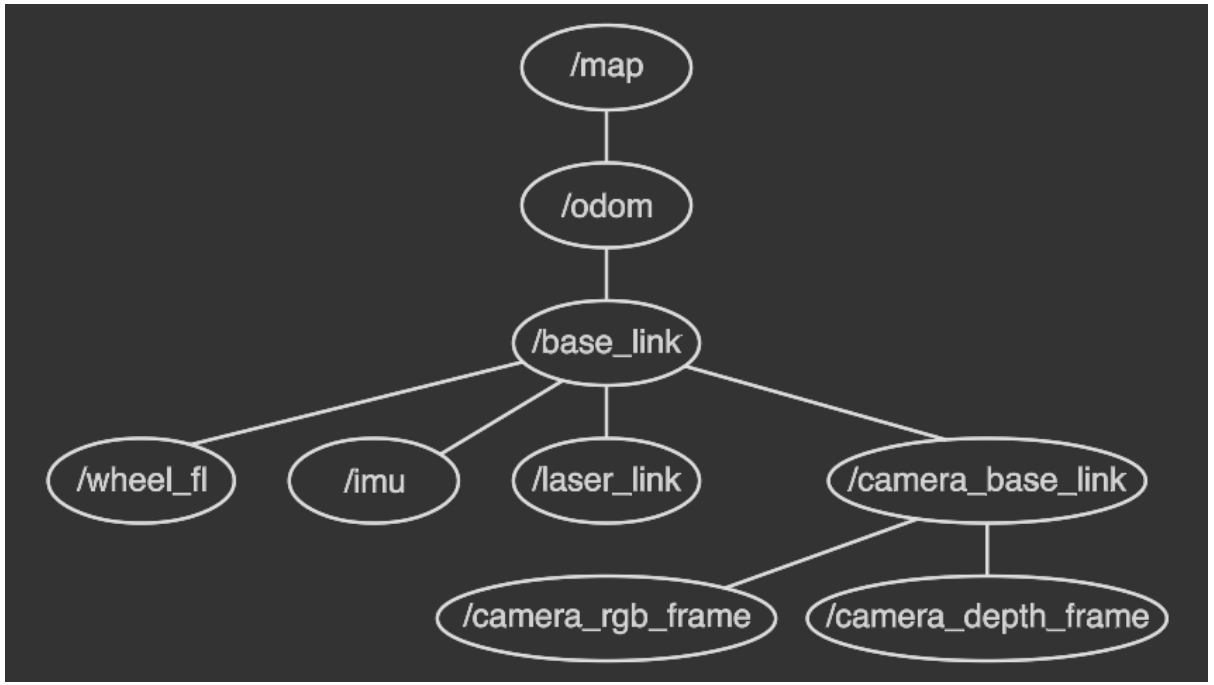
Links

Links (between frames) may be:

- Static – such as fixed displacements of robot geometry
- Actuator – that take dynamic values from actuator (joint) measurements of the robot. Typically these are rotational links.
- Mapping/Odometry – that describe the transform from the “world” frame to a moving robot/object position. Typically these are a combination of a translation and rotation

Transform Tree

Transform Tree



Forward Kinematics

ESSAI July 2025

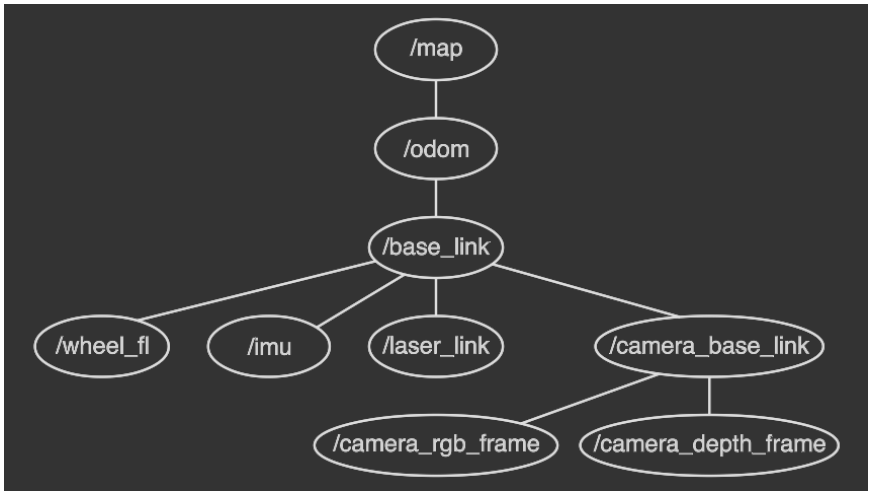
RMIT UNIVERSITY

Definition of Forward Kinematics

Forward kinematics is the “forward” application of the TF tree where:

- All TFs are known, that is, all joint angles are known
- Transform a point, vector, pose in one frame into another frame

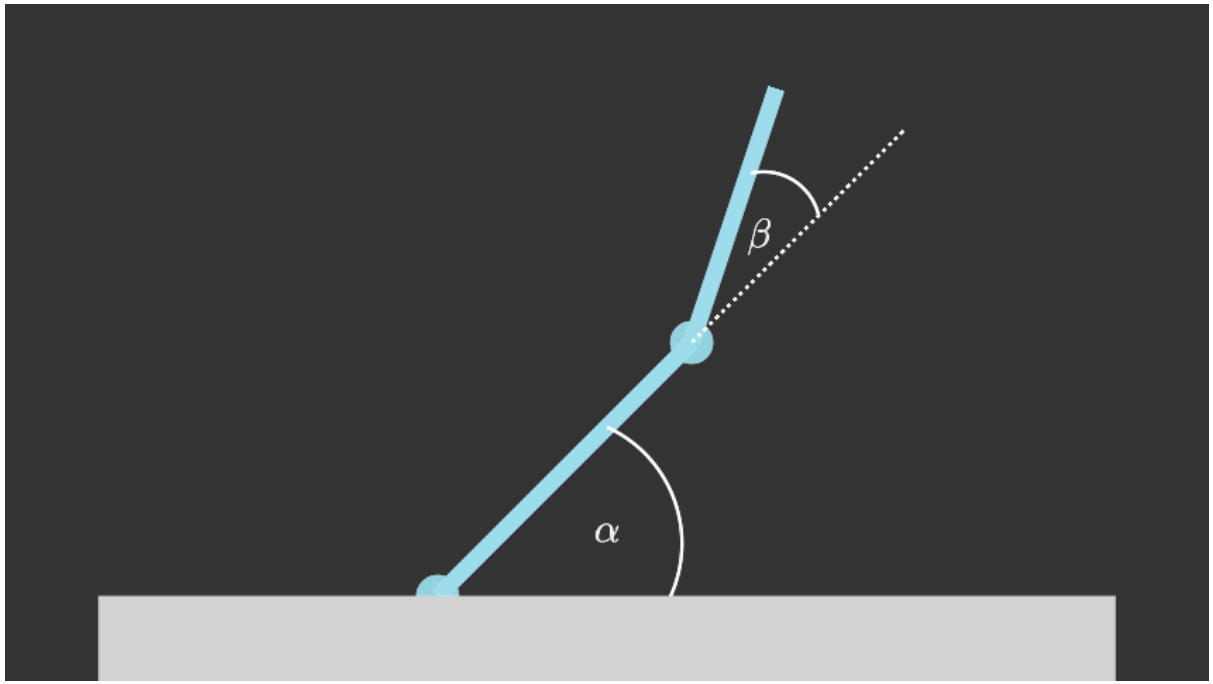
$$
{}^{\text{camera\_rgb\_frame}}P = {}^{\text{base\_link}}_{\text{laser\_link}}T^{-1} \times {}^{\text{base\_link}}_{\text{camera\_base\_link}}T \times {}^{\text{camera\_base\_link}}_{camera\_rgb\_frame}T \times {}^{\text{laser\_link}}P
$$



Simple Arm Example

In forward kinematics, compute the arm end-effector:

- The angles α and β are known

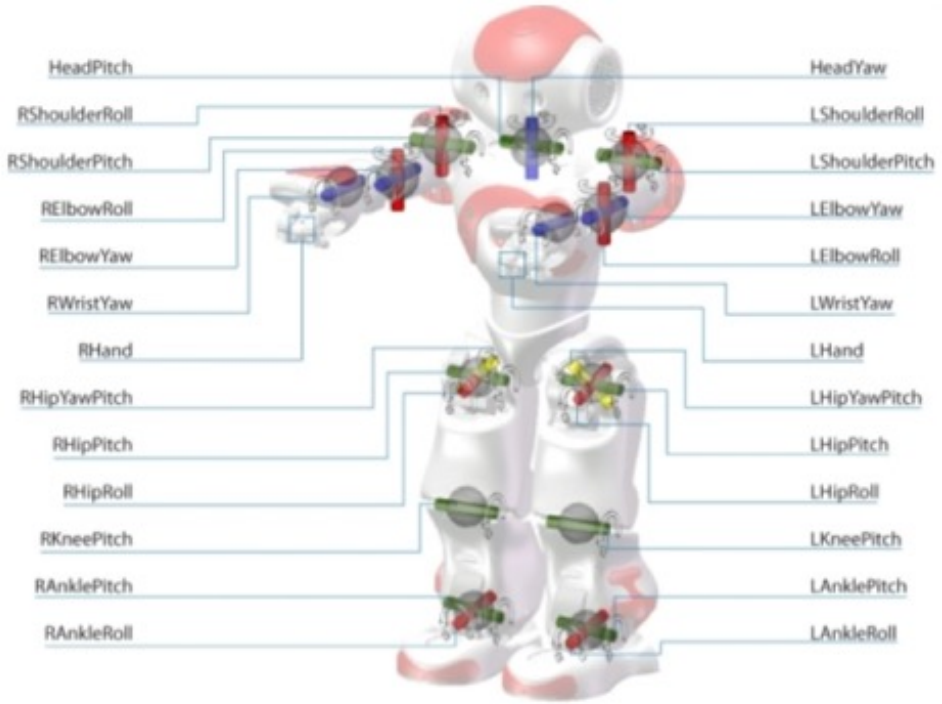


Transforms using the ROSBot in ROS

Worked Example

Nao Kinematics

A more complex robot has more transformation frames. But, this doesn't impact the overall mathematics on kinematic transforms

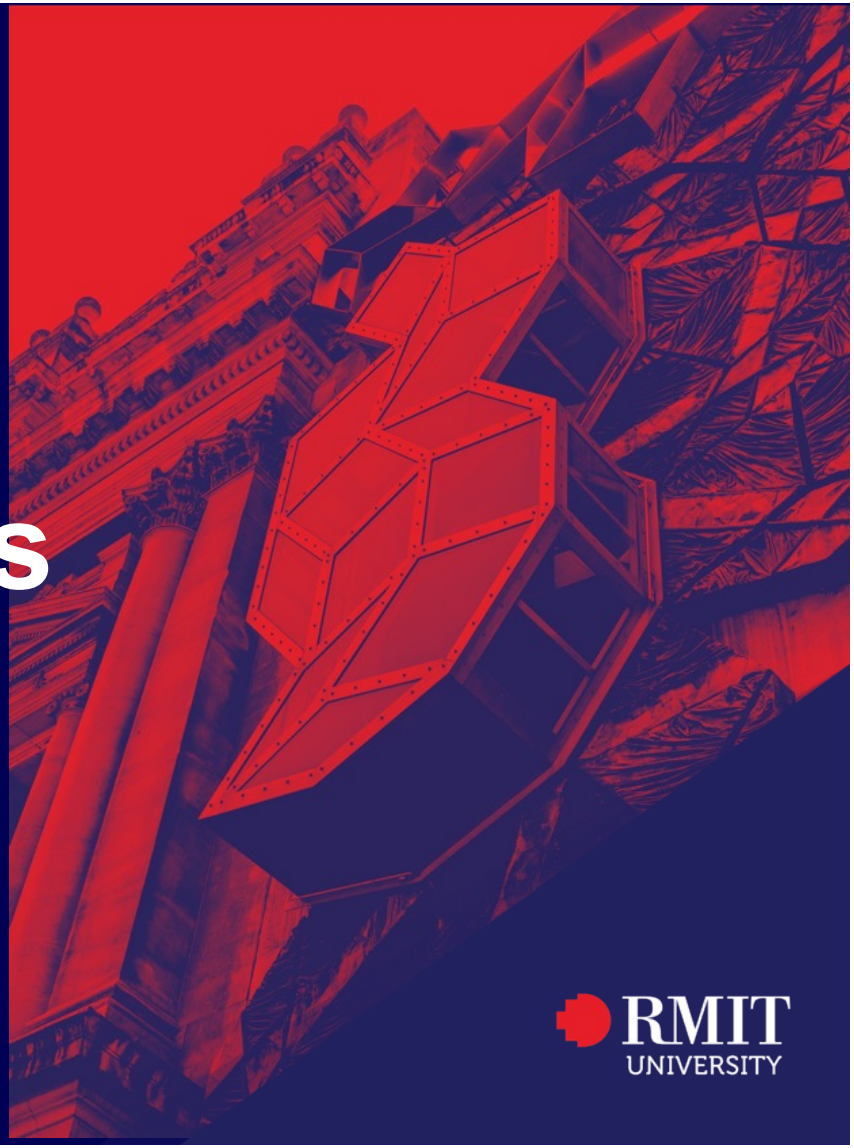


Real-time Kinematic Problems

The general issues to be aware of for any kinematics:

- Asynchronous updates of dynamic links
- Only approximate correspondence to sensor inputs, which also update asynchronously at a different rate to actuators
- Delays between TF update, processing, behaviour generation and final joint actuation

Inverse Kinematics



ESSAI July 2025

RMIT UNIVERSITY

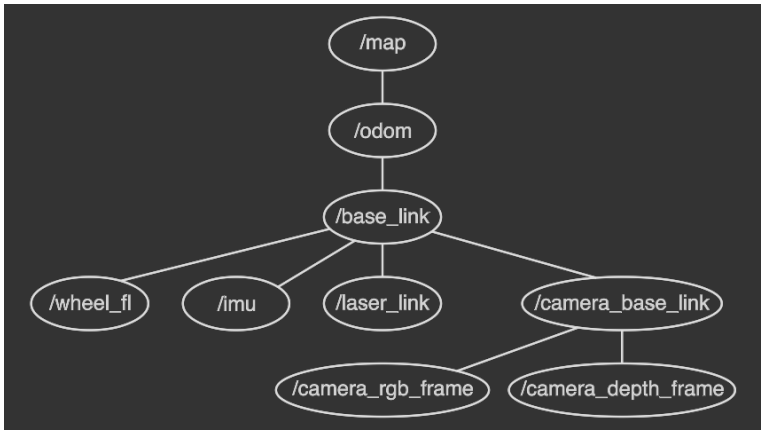
Inverse Kinematics

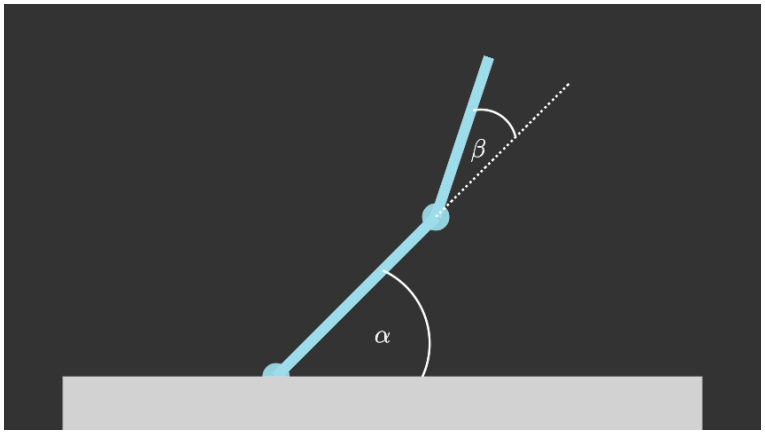
Inverse kinematics is the computation of the required joint states to achieve an end-effector position:

- Static TFs are known
- Dynamic TFs must be computed

Requires solving for the transform of the forward kinematic equation

$$^{A}P = {}^{A}_{B}T \times {}^{B}P$$





Closed Form Solvability

It is possible to solve the equation using closed form methods:

- Observe that robotics systems are highly non-linear
- Analytical solutions can be found, depending on the complexity of the non-linear system
- Likely to have multiple solutions
- A robotic system should have at least 6DOF

Worked example in Matlab with the Puma560 arm

Closed Form Solvability

Limitations:

- Computation for complex kinematic chains
- Unsolvable solutions for under-actuated systems
- Multiple solutions for over-actuated system
- Singularities, if using Euclidean Transforms

Approximate Solving via the Jacobian

Approximate closed-form solutions can be more efficient and “reasonable”:

- The kinematics define a multi-dimensional configuration space
- Finding the closed-form solution is the same as finding the minimum of this configuration space.
- This can be done by employing a technique similar to gradient descent to find the minimum
- That is, finding the derivative of the configuration space

Approximate Solving via the Jacobian

The Jacobian matrix, J, is all partial derivatives of a systems kinematics:

$$J = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

For an analytical solution, solving still requiring inverting the Jacobian, which may not be possible. However:

- An approximation to inverting J is: $J^+ = \frac{J^T}{J.J^T} = \frac{1}{J}$
- This can be used for an iterative convergence method: $\Delta j = J^+ \epsilon$
- Where ϵ is the error between the desired and actual position

Motion Control

ESSAI July 2025

RMIT UNIVERSITY

Types of control

The general form of control for a robotic actuation system uses:

- Position control
- Velocity control
- Acceleration control

Rigid-body motion mathematically formalises a robot's motion:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}.\dot{\mathbf{q}})\dot{\mathbf{q}} + \tau_{\mathbf{g}}(\mathbf{q}) = \tau$$

Rigid-Body Motion

Rigid-body motion mathematically formalises a robot's motion:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}.\dot{\mathbf{q}})\dot{\mathbf{q}} + \tau_{\mathbf{g}}(\mathbf{q}) = \tau$$

But what is this mean?

- Control of is a function of time, that is, we are controlling a dynamic motion, not an instantaneous snapshot
- Control requires effective measurement of the joint parameters
- This motion is a function of all joint parameters
- The desired control can be computed by solving the motion equation
- There are many different motion equations depending on the robot

Open-Loop Motion Control

Rigid-body motion mathematically formalises a robot's motion:

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}.\dot{\mathbf{q}})\dot{\mathbf{q}} + \tau_{\mathbf{g}}(\mathbf{q}) = \tau$$

This is what we have already looked at with inverse kinematics.

Practical Issues of Motion Control

Aside from the issue of solveability, open-loop control has issues with:

- Sensor noise
- Instantaneous actuation error
- Accumulative error
- Drift

Worked Example with ROSBot

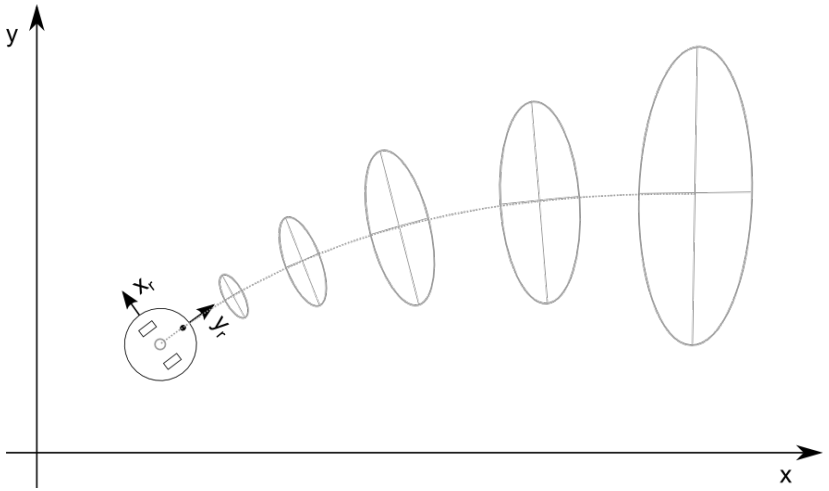


Image: Correll, 2022, introduction to Autonomous Robots

Closed-Loop Control

Closed-loop control uses actively monitors the error between a desired set-point of the actuator and actual-position. The motion-control (position, velocity and/or acceleration) is then adjusted based on the error.

This means the actuator is “servo-ed” into position gradually.

Closed-loop control can also be used in-place of inverse kinematic solving.

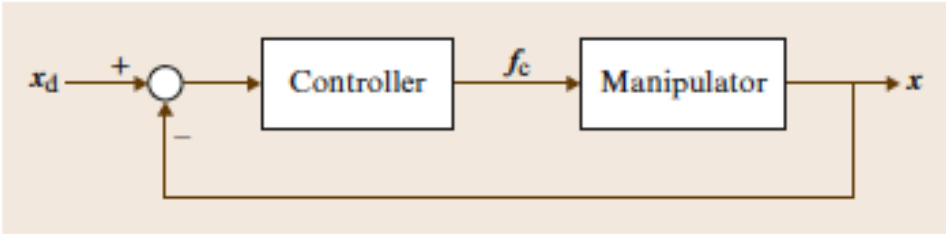


Image: Siciliano & Khatib (Eds), 2016, Springer Handbook of Robotics

PID Control

The most common form of closed-loop control is PID-control.

The general PID controller equation is:

$$\tau = K_p \mathbf{e_q} + K_i \int \mathbf{e_q} dt + K_d \frac{\mathbf{e_q}}{dt}$$

Where:

- P – Proportion of the instantaneous error
- I – Integral of the cumulative error
- D – Derivative of the instantaneous change in the error
- Each ‘K’ term is the ‘Gain’ of how much each PID term should be weighted

PID Control: Online example

The general PID controller equation is:

$$\tau = K_p \mathbf{e_q} + K_i \int \mathbf{e_q} dt + K_d \frac{\mathbf{e_q}}{dt}$$

Online PID example: http://grauonline.de/alexwww/ardumower/pid/pid.html

PID Control: ROSBot

The general PID controller equation is:

$$\tau = K_p \mathbf{e_q} + K_i \int \mathbf{e_q} dt + K_d \frac{\mathbf{e_q}}{dt}$$

ROSBot worked example of PID Control

PID Tuning

There are a wide variety of approaches for tuning the parameters of PID controllers including:

- Manual tuning
- Process Reaction Curve
- Ziegler-Nichols Method
- Cohen-Coon Method
- Lambda Tuning Method
- Internal Model Control
- Reinforcement Learning
- etc.

PID Tradeoffs

PID is still very common as it:

- Simple
- Directly uses known control dynamics of common systems
- Surprisingly robust to noise

However, the tuning of a PID system is critical. A poorly tuned system:

- May oscillate rather than converge
- May diverge with an exploding error

Bi-Pedal Motion

ESSAI July 2025

RMIT UNIVERSITY

Bi-Pedal Walk

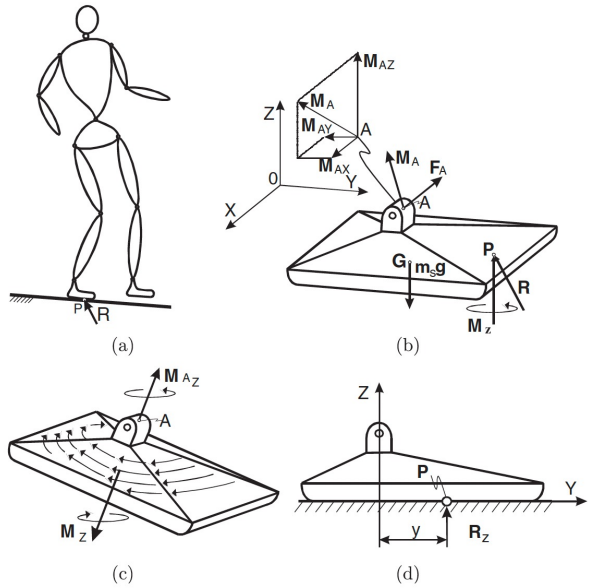
Earlier we've noted that control can be governed by known equations (models) of motion for a particular system.

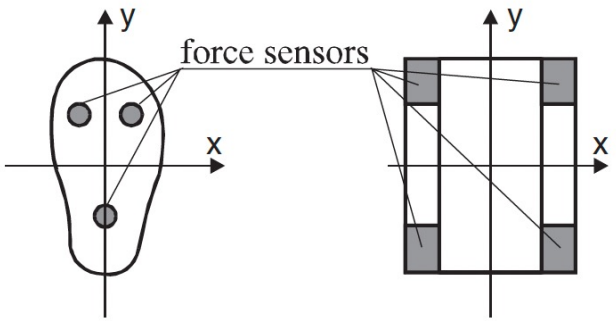
Bi-pedal motion has been studied quite some time, on Nao we typically modelled with two concepts:

- Zero-Moment Point (for foot control)
- Inverted Pendulum of dynamic walk motion

Bi-Pedal Walk (ZMP)

The ZMP concept simplifies bi-pedal walk by reducing all the forces acting on the robot to a single force termed the ‘Zero-Moment Point’.



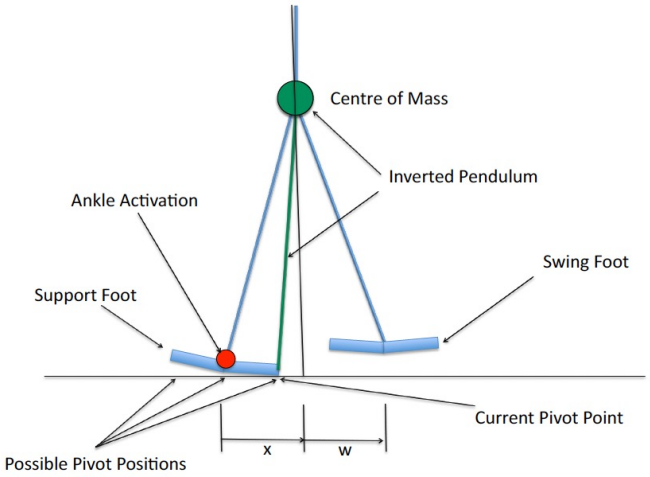


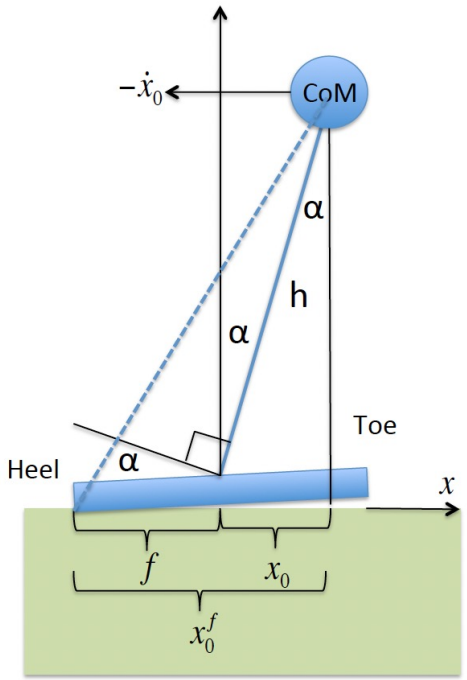
Images: Vukobratović, Miomir, and Branislav Borovac. "Zero-moment point—thirty five years of its life." International journal of humanoid robotics 1.01 (2004): 157-173.

Bi-Pedal Walk (Inverted Pendulum)

Bi-pedal walk also has a ‘rocking’ motion as:

- Weight is transferred between feet
- Single leg moving back/forwards



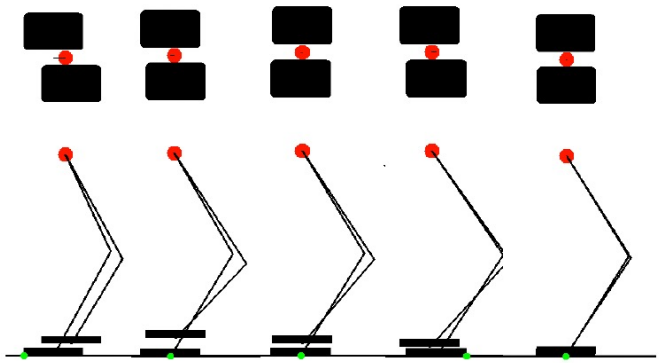


Images: Hengst, Bernhard, Manuel Lange, and Brock White. "Learning to control a biped with feet.”, Tech. Report 2014.
Hengst, Bernhard. "Reinforcement learning inspired disturbance rejection and Nao bipedal locomotion." 15th IEEE RAS Humanoids Conference. 2015.

Learning Bi-Pedal Walk

These combine to give a parameterised control equation, where the regularity of the pendulum is maintained, and the foot placement is controlled to maintain the ZMP within the foot boundary.

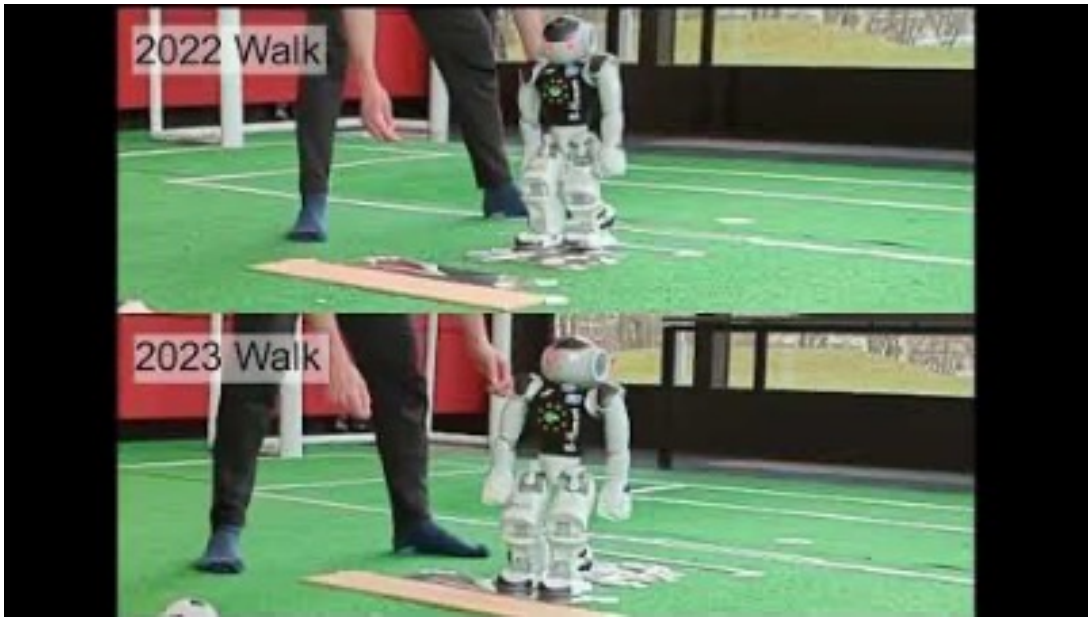
The parameters of the walk can are learnt through RL.



Hengst, Bernhard. "Reinforcement learning inspired disturbance rejection and Nao bipedal locomotion." 15th IEEE RAS Humanoids Conference. 2015.

Learning Joint Wear

Additional parameters can be introduced into the model to adjust for wear and tear in joints. Tuning (or learning) these parameters on a per-robot basis allows for correction of joint error.



Reichenberg, Philip, and Thomas Röfer. "Dynamic joint control for a humanoid walk." Robot World Cup. Cham: Springer Nature Switzerland, 2023. 215-227.

Motion Planning

ESSAI July 2025

RMIT UNIVERSITY

Motion Planning: Example

Motion Planning

Simple closed-loop control assumed that the robot actuators can move through all intermediate ranges without conflict. For complex actuation this assumption is not true, due to:

- Self collisions
- Environment collusions

Motion planning overcomes these problems, by finding a sequence of intermediate positions for actuators before the end-position is reached.

A motion plan is the sequence of joint movements to move the joints from one position to another position.

Motion planning is one of the hardest problems now in robotics due to:

- Need for precise fine-grained motor control motion
- The complex nature of solving motion equations

Example with the Puma560

Worked example in Matlab with the Puma 560

Configuration Spaces

Motion planning can be solved analytically, but this is computational expensive.

An different approach to motion planning uses Configuration Spaces.

A configuration space described all valid actuator positions. When visualised, this is the actuator space, *not* the 3D space of the environment.

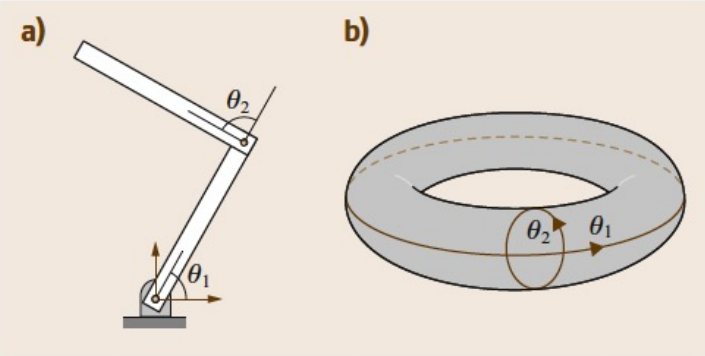


Fig. 7.2 (a) A two-joint planar arm in which the links are pinned and there are no joint limits. (b) The C-space

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

Configuration Spaces

Configuration Spaces

Configuration spaces:

- May have complex shapes for self collisions
- Include boundaries (or breaks) for environment collisions

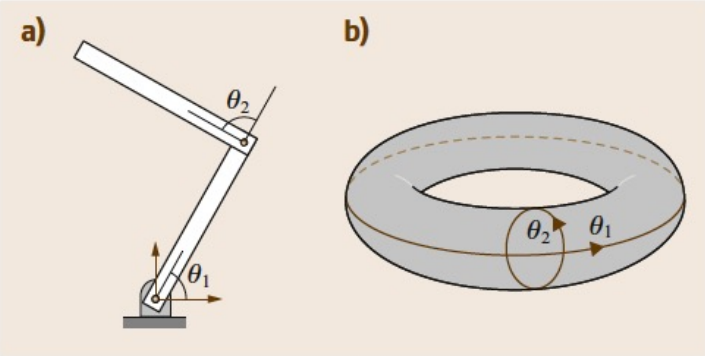


Fig. 7.2 (a) A two-joint planar arm in which the links are pinned and there are no joint limits. (b) The C-space

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

Sampling Based Motion Planning

A common approach to Motion Planning is to randomly sample the configuration space and build a traversable graph through the space. Algorithm:

1. Initialise a Graph, G, with the starting & end configuration.
2. Randomly sample a configuration, $\alpha(i)$, from with the configuration space, and add as a vertex to G
3. Find all vertices in G within a neighbourhood of $\alpha(i)$ and connect the vertices if possible
4. Repeat (2) & (3)
5. Terminate when a path is found and at least *N* vertices are added to G

MoveIt, the ROS package, uses sampling based approaches, and you can visualise the planning

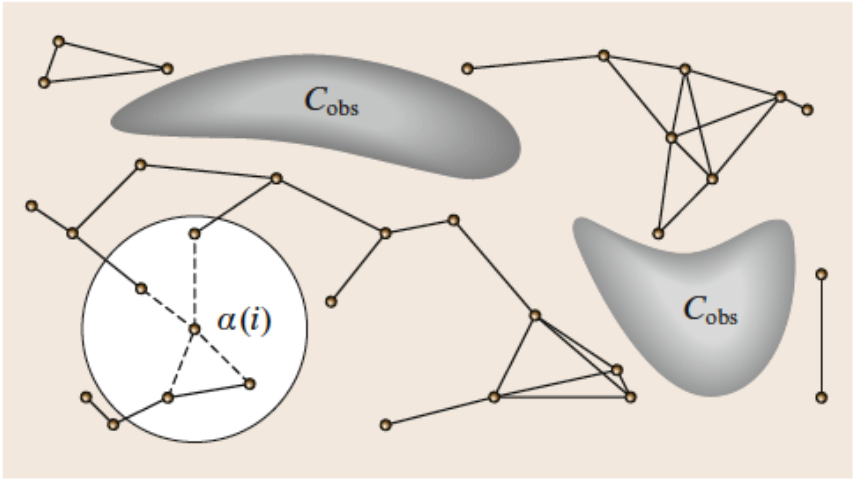


Image: Siciliano & Khatib (Eds), 2016, Springer Handbook of Robotics

Navigation

ESSAI July 2025

Map

… before Navigation

ESSAI July 2025

Occupancy Grid

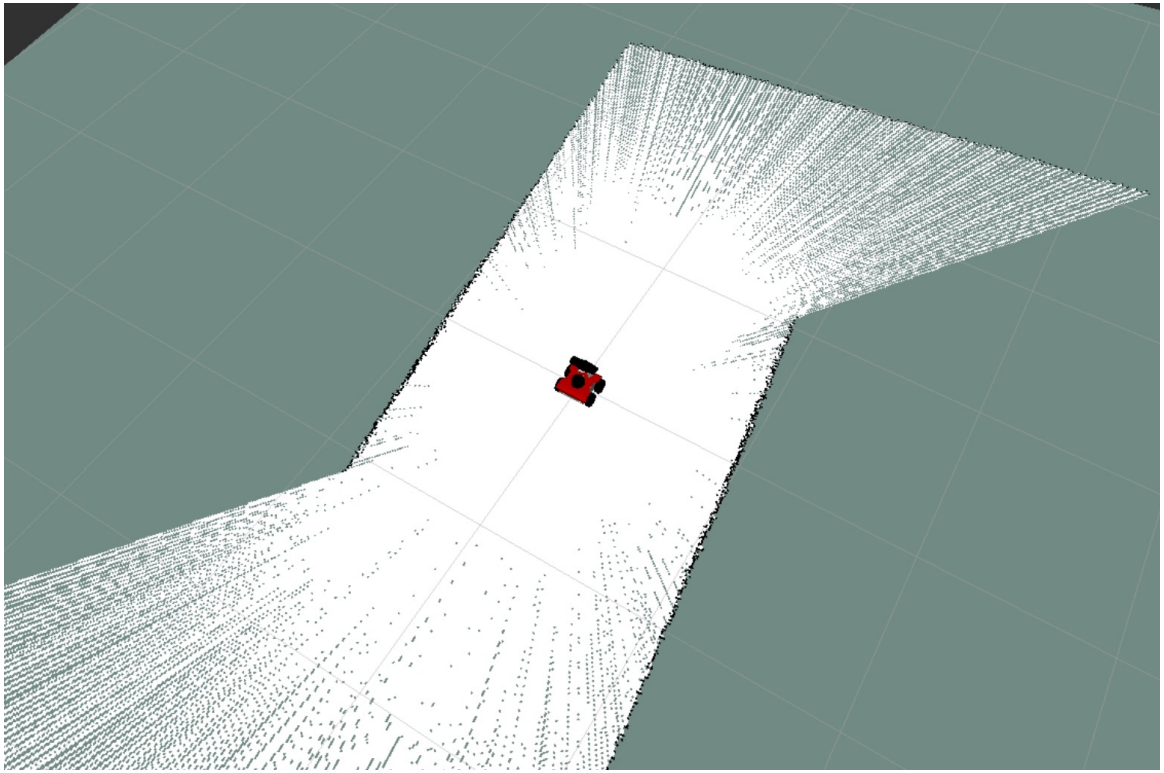
The most common representation of a Map in robotics is a 2D occupancy grid. As the game suggests:

- 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)

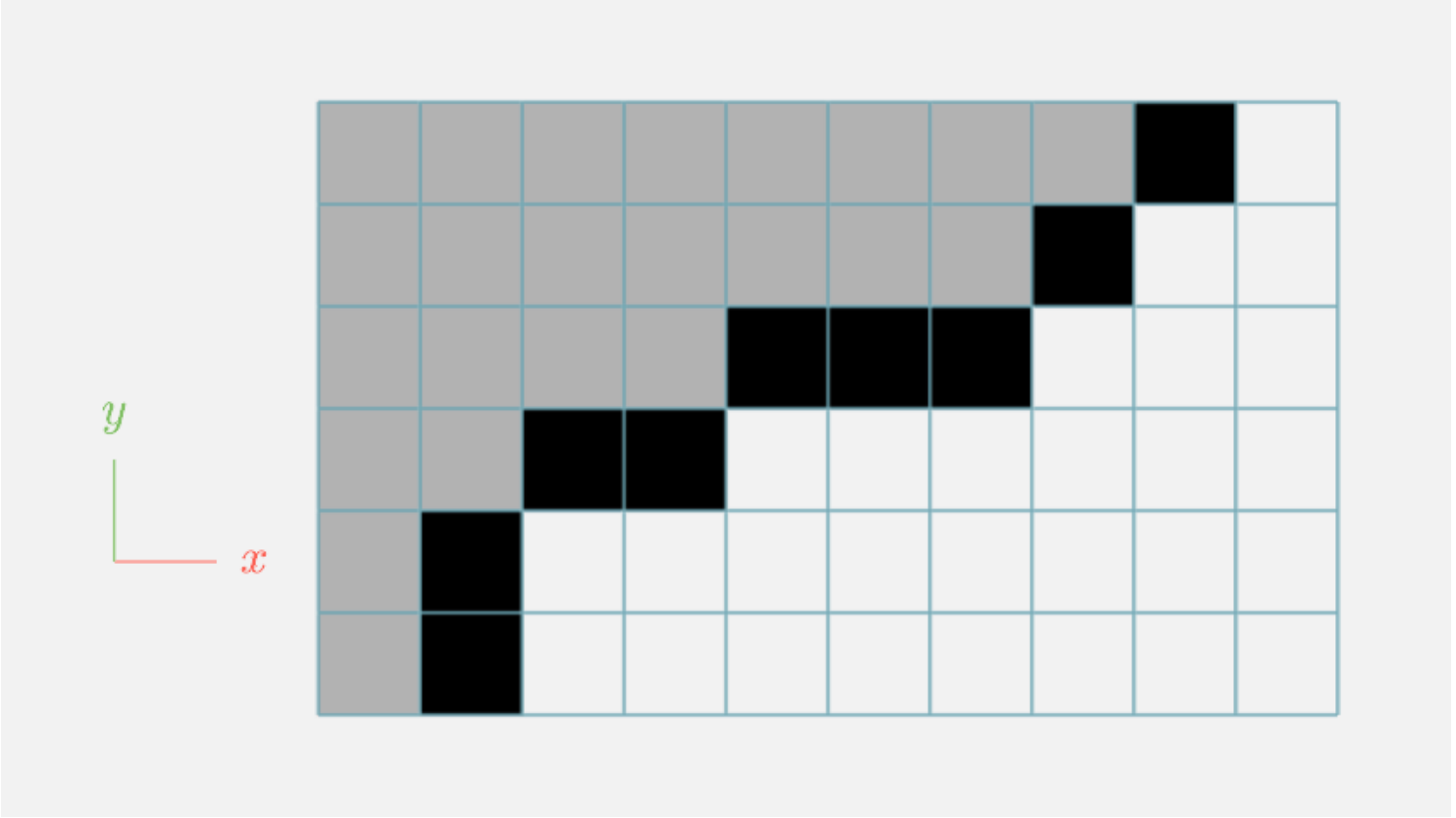
Question: Why have probabilities?

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.

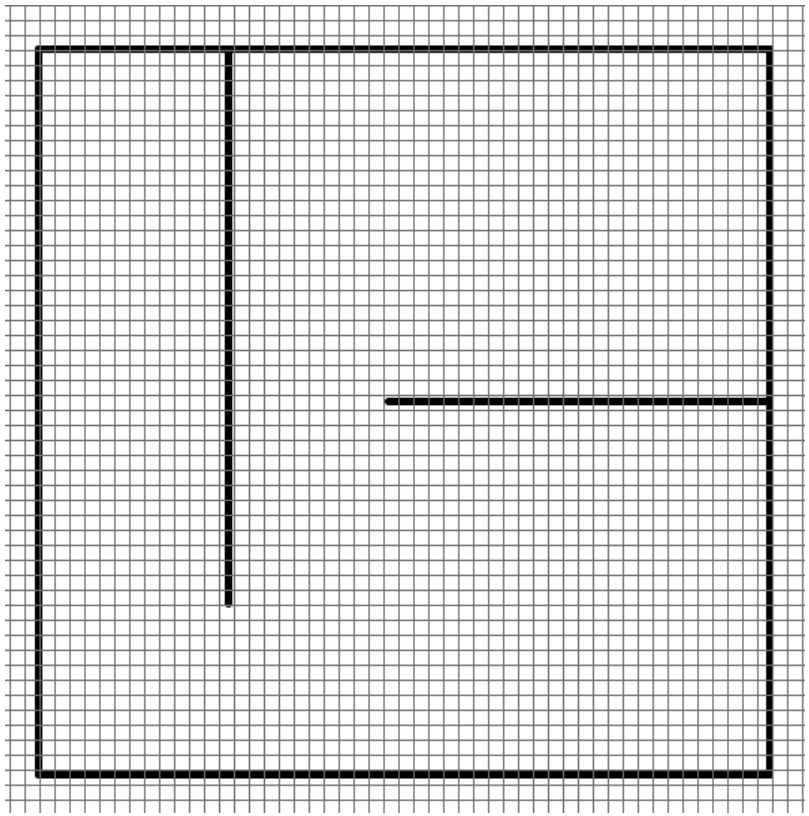
Navigation

… in a map

ESSAI July 2025

Motivation

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



A* Navigation

Generally, the obvious answer is A*:

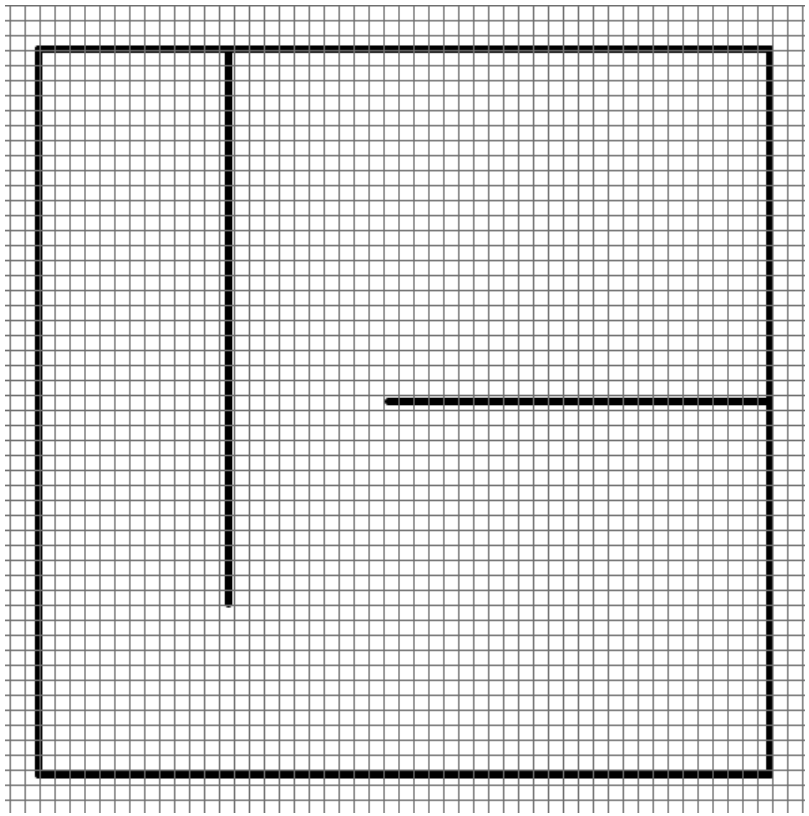
- The Map (Occupancy Grid) is interpreted as a graph
- Cell cost equation: $f(x) = g(x) + h(x)$
  - $g(x)$: shortest path to x
  - $h(x)$: Heuristics are typically:
    - Euclidian
    - Manhattan
  - The “cost” of an individual step is 1

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

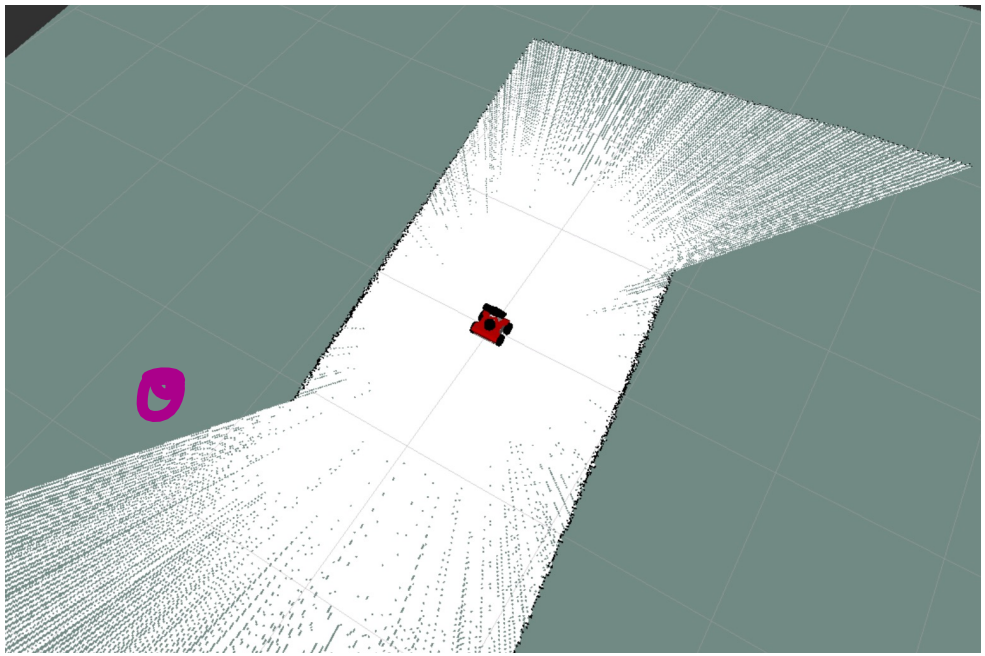
Navigation

CostMaps

ESSAI July 2025

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.

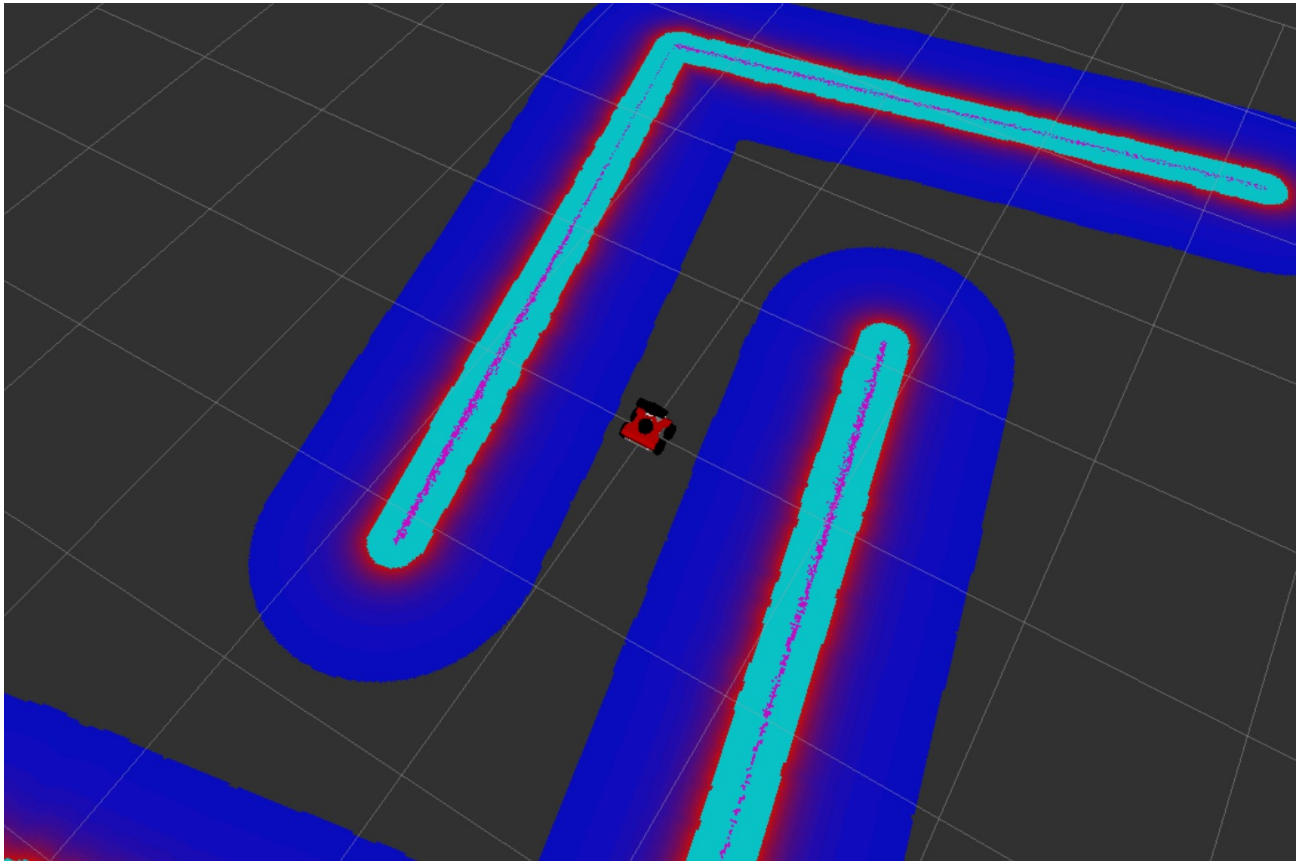


Costmap

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

Noon Gudgin

Thank you

Day 3: Localisation, Mapping and Navigation

ESSAI July 2025

RMIT UNIVERSITY