

<https://attendance.lincoln.ac.uk/>



UNIVERSITY OF  
LINCOLN

## CMP3103 – AUTONOMOUS MOBILE ROBOTS

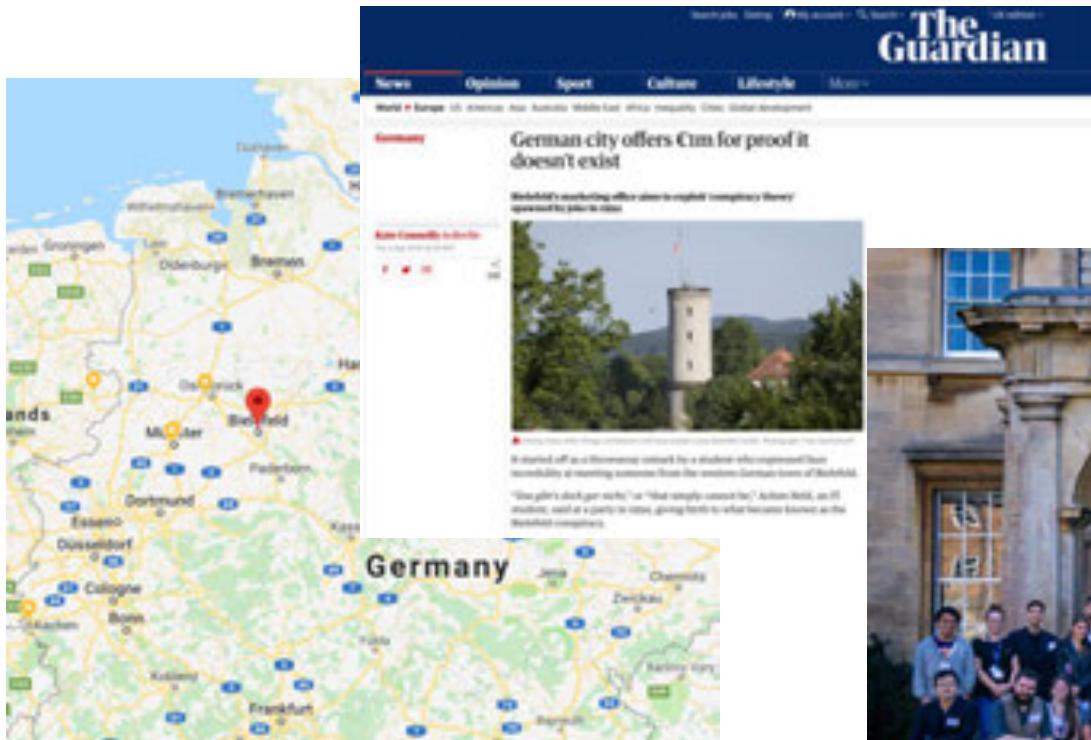
*Lincoln Centre for Autonomous Systems  
School of Computer Science*



Access Code: 160955



# About your Lecturer and L-CAS and the rest of it



<https://lcas.lincoln.ac.uk/wp/>





Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](http://PollEv.com/app)



Access Code: 160955

# About your Lecturer and L-CAS and the rest of it



UNIVERSITY OF  
BIRMINGHAM



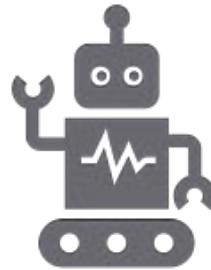
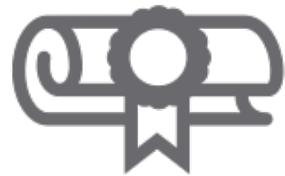
UNIVERSITY OF  
LINCOLN



robots that are *useful*

robots that have some  
*common-sense*

robots that can *adapt*



## Semester B: AMR

Prof Marc Hanheide, Dr Athanasios  
Polydoros

## Assessment items:

Robotics Practical Coursework (40%)  
Oral Exam (TCA) (60%)



## Semester B: AMR



### Lectures

You said: "We want to have experience with real robots"  
"We want flexibility to work on assignments"



### Workshops

Ubuntu 22.04 LTS in docker, home installation possible  
Robot programming in ROS2, in simulation & real robots



Attendance of all scheduled sessions is mandatory!



Please check your timetable & blackboard for any timing & updates



# What's on?

term week	Sem B week	Lecture (9-11)	Topic	Lecturer for Lecture	Workshop 11-13	Workshop Task
19	1	Tuesday, 30 January 2024	Intro	Marc	Wednesday, 31 January 2024	setup, running simulation
20	2	Tuesday, 6 February 2024	Robot Programming ROS	Marc	Wednesday, 7 February 2024	ROS intro, topics, make robot move
			Robot Sensing and Computer			
21	3	Tuesday, 13 February 2024	Vision	Marc	Wednesday, 14 February 2024	Rviz, point clouds, tf, more on event-driven programming, open_cv, etc.
22	4	Tuesday, 20 February 2024	Motion and Control	Geesara	Wednesday, 21 February 2024	kinematics, color chaser, position control
23	5	ENHANCEMENT WEEK		—	—	
24	6	Tuesday, 5 March 2024	Robot Behaviour + basic RL	Geesara	Wednesday, 6 March 2024	behaviour task + assignment support
25	7	Tuesday, 12 March 2024	Navigation	Athanasiос	Wednesday, 13 March 2024	assignment support
26	8	Tuesday, 19 March 2024	Localisation	Athanasiос	Wednesday, 20 March 2024	assignment support
27	9	EASTER		—	—	
28	10	EASTER		—	—	
29	11	Tuesday, 9 April 2024	Robot mapping - SLAM	Athanasiос	Wednesday, 10 April 2024	assignment support
30	12	Tuesday, 16 April 2024	Control Architecture	Athanasiос	Wednesday, 17 April 2024	assignment support
31	13	Tuesday, 23 April 2024	HRI 1	Athanasiос	Wednesday, 24 April 2024	HRI WS
32	14	Tuesday, 30 April 2024	HRI 2	Marc	Wednesday, 1 May 2024	
33	15	Tuesday, 7 May 2024	Revision	Marc	Wednesday, 8 May 2024	Q & A



# “AgileX Limo” – Our Workshop Buddy

This year's assignment: "TidyBot"  
(more soon...)

ROS  
Enabled



# Coursework

## Module Code and Title:

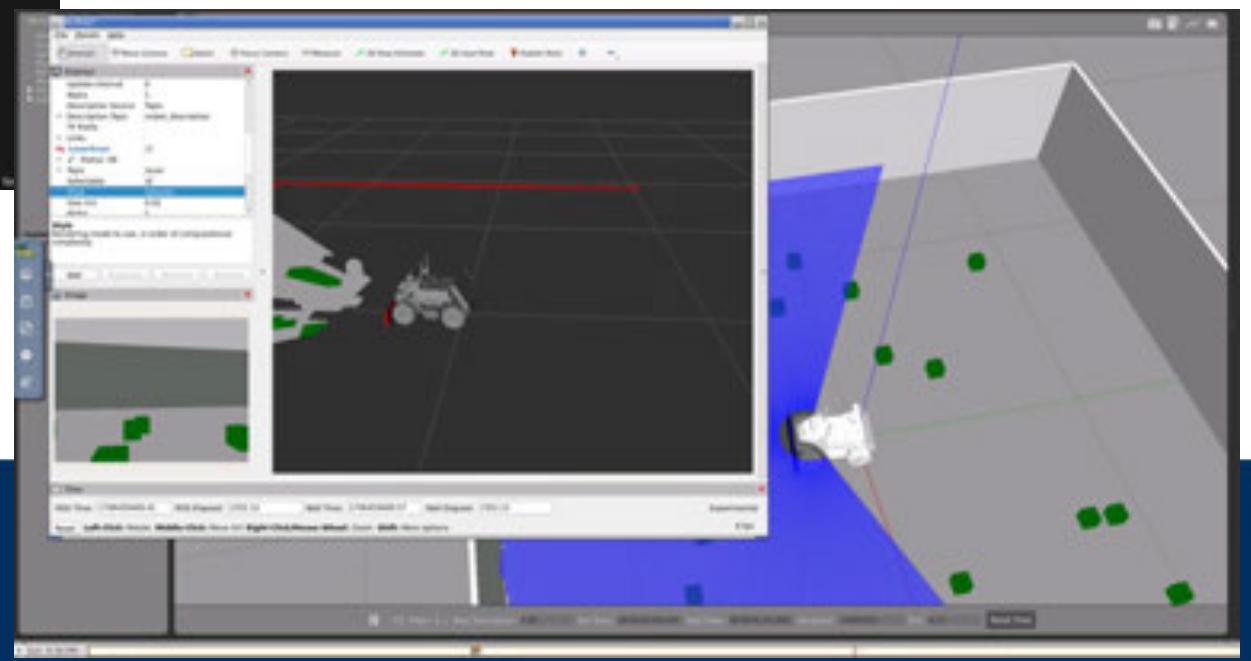
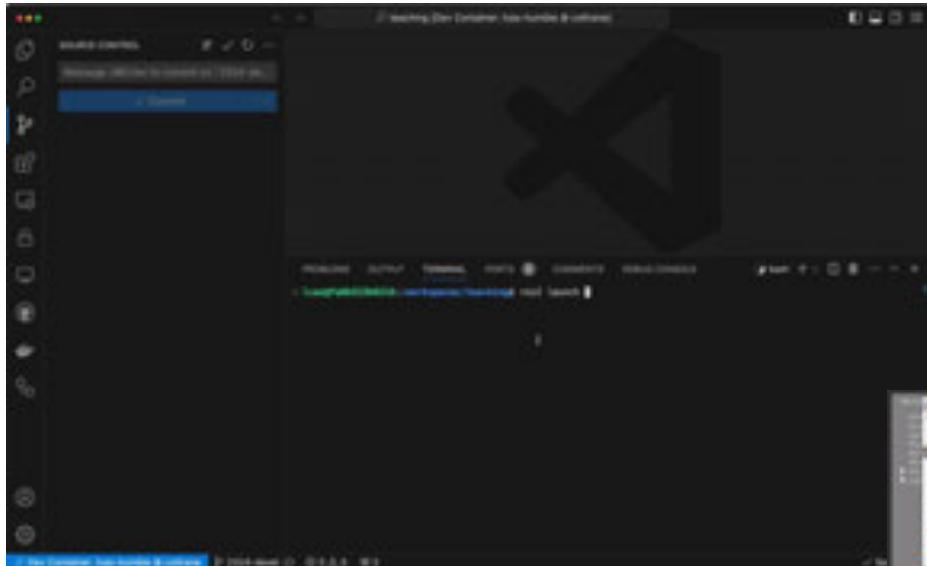
**CMP3103 – Autonomous Mobile Robots**

## Contribution to Final Module Mark: 40%

## Description of Assessment Task and Purpose:

In this assessment you are expected to build a software artefact, controlling simulated and real robots using Python as the programming language, using the **ROS2** middleware. You will implement a robot behaviour that tidies a given environment by pushing small boxes with a mobile robotic platform (a bit like a bulldozer). You will be using an **AgileX Limo robot**, in **simulation** and the **real world**, to accomplish this task in a given environment. The environment is a flat arena, delimited by walls on all sides. Small moveable, coloured boxes are distributed randomly around the centre of the arena. The robot is equipped with an RGBD camera and a LiDAR Scanner. You are provided with a “**DevContainer**”, pre-installed with all relevant software (Ubuntu, ROS2, relevant ROS packages and development tools).

# Simulated Limo in VSCode devcontainer (Docker)



# Coursework

There are different complexity levels for the general “tidy-up task”:

1. As the simplest solution, the task is to detect the movable (brightly coloured) objects in the environment and push them to the wall (within 30 cm of the wall). It doesn't matter where exactly they are pushed as long as they are moved towards any wall, clearing the centre of the environment. The environment at this level has no further static obstacles than the walls.
2. At the 2<sup>nd</sup> level, the arena features some static obstacles (different colour to the moveable objects) that must be avoided in the process. Otherwise, this level is the same as level 1.
3. The 3<sup>rd</sup>, advanced, level features moveable objects in two distinct colours (e.g. red and green), initially randomly mixed and distributed at the centre of the environment. The advanced task is to push all objects of 1 colour to one side of the arena and the other to the opposite side. This level is truly advanced as it requires you to implement not only reactive behaviour but equip the robot with a global sense of direction and location.

You'll mainly work in the provided simulation environment, but you will have the opportunity to work with the real robots provided. 10% of the overall assessment mark is awarded for evidencing your implementation working on the real robot (see CRG). While you will receive credit for making it work on the real robot and evidencing a solution working for a higher complexity level, you can pass this assessment with a working (and documented) solution in simulation at complexity level one.

# Coursework

## Assessment Submission Instructions:

### Implementation

You will submit your developed Python (version 3.x compatible) source code (optionally in the form of a properly formatted ROS2 package). Your code must be suitably commented and must reference ALL resources used in code comments (e.g. when you adapt code found online, you must include the URL to that resource in your comment and explain what the code does and how you have adapted it for your work).

Your code submission must include a file **README.md**, detailing precisely how to run your system, including which simulation environment to test the code in.

### Video

You are also required to submit a short video (of 3 minutes maximum duration, no more than 50Mb total size, MP4 format) of the simulation and the real robot running your implementation in action. This video must have a voice-over presentation by yourself explaining your approach and the video content. You must make sure that the video is compressed to less than 50Mb and in the correct format (MP4).

Your video (with the voice-over) should cover a presentation of the exemplary performance of your implementation (screen recording of the simulation with your implementation running), a brief description of your overall system design, including details of the distinctive features of your perception and control algorithms, and some reflection on your implementation's performance. You may move the robot manually during the recording or post-edit your video to show the different situations (in simulation and reality) you want to highlight. **Make sure you present all your hard work briefly in the video recording.**

# Coursework CRG

Learning Outcome Criterion		Pass	2:2	2:1	1
[LO2] understand and critically evaluate the range of possible applications for mobile robotic systems	Criterion 1: Tidy-Bot (in simulation), Implementation  50%	A working software component with basic functionality. Fair program structure and some code comments. The code works and a working implementation can be demonstrated.	A working software component with good functionality. Clear program structure and appropriate comments. The implementation is demonstrated successfully.	A good implementation with some extra functionality or originality. The concept implemented is flexible enough to work at least at complexity level 2. The program code is well structured and commented. Good demonstration of basic and some advanced features, making progress towards the goal of tidying objects.	An excellent implementation featuring original functionality and elements beyond the original specification. The program code is efficient, very well-structured and commented. The solution demonstrated works very well, including functionalities for higher complexity levels, accomplishing significant progress towards tidying objects, e.g., at different complexity levels.
[LO3] implement and empirically evaluate intelligent control strategies, by programming autonomous mobile robots to perform complex tasks in dynamic environments	Criterion 2: Tidy-Bot (in simulation), performance  10%	The implementation moves the simulated robot in a meaningful way.	The simulated robot makes successful attempts to solve the task at least at complexity level 1 or above.	The simulated robot successfully attempts to solve the task at least at complexity level 1 or above. It has the additional functionality, e.g., to address the challenges of complexity level 2.	The performance of the simulated robot is strong, with excellent accomplishment of tasks, at different complexity levels.
	Criterion 2: Tidy-Bot (in real world), performance  10%	The implementation moves the real robot in a meaningful way.	The real robot makes successful attempts to solve the task at least at complexity level 1 or above.	The real robot successfully attempts to solve the task at least at complexity level 1, with evidence of functionality and flexibility to e.g. solve higher complexity levels. Real- world challenges are successfully addressed	The performance of the real robot is strong, with specific consideration of the real-world challenges, at different complexity levels.
	Criterion 3: Tidy-Bot, Video Presentation  30%	A basic video showing the performance of the implementation.	A good presentation in a video of the system design and performance, with reflections on its performance, including evidence of performance evaluation.	A very good video presentation of the system design and reflections on its performance, that evidences a detailed understanding of the problem and solution, with evidence of non-trivial performance evaluation.	An excellent video presentation of the system design and reflections on its performance, including evidence of thorough testing and evaluation of the important system features with conclusions drawn from it.
Weighting is 40% of the module					

# Assessment Item 2 – Oral TCA

## Assessment Item 2 Documentation

Attached Files:   [CMP3103M\\_CMP9050M Assessment Item 1 2324 ORAL EXAM CRG.pdf](#)   (30.315 KB)

Assessment Item is a **Time Constraint Assessment (TCA)**, taking the form of an individual **Oral Examination**.

Individual oral examinations will be conducted according to a released schedule with up to 20 minutes per student. The examination will be audio-recorded for moderation purposes. All recordings will be deleted after the exam board takes place and marks are confirmed. Examiners will ask questions from a pre-defined set, divided into three thematic areas, with each area approximately being given equal waiting. Each question is annotated with a mark. The examiners will give opportunity to students to answer questions summing up to at least 40 marks and no more than 50 marks according to a defined marking scheme. Examiners will link the questions to the practical workshop material as well as the lecture material. Moderation will take place by examiners listening to the recording of the examination according to school policies. The choice of questions asked will be documented individually for each student and each question marked individually.

Please find attached the CRG for assessment item 2.

If you have questions pertaining to the requirements of this assessment item, please seek the advice of a member of the delivery team.

# The Reading List

## Resources for the Lectures (6 items)

Introduction to autonomous mobile robots - Roland Siegwart, Illah Reza Nourbakhsh, Davide Scaramuzza, c2011  
[Book](#) | Essential Reading

Mobile robotics: a practical introduction - Ulrich Nehmzow, ProQuest (Firm), 2003  
[Book](#) | Recommended Reading

Probabilistic robotics - Sebastian Thrun, Wolfram Burgard, Dieter Fox, ProQuest (Firm), 2005  
[Book](#) | Recommended Reading

Autonomous robots: from biological inspiration to implementation and control - George A. Bekey, 2005  
[Book](#) | Further Reading

Scientific methods in mobile robotics: quantitative analysis of agent behaviour - Ulrich Nehmzow, c2006  
[Book](#) | Further Reading

Plan, activity, and intent recognition: theory and practice - Christopher Geib, 2014  
[Book](#) | Further Reading

## Resources for the Workshops (6 items)

ROS2 Tutorials - ROS Wiki  
[Webpage](#) | [Essential Reading](#) | The Beginner Level Tutorials are quite essential for the practical work in this module.

A concise introduction to robot programming with ROS2 - Francisco Martín Rico, 2023  
[Book](#) | Recommended Reading

ROS robotics by example: bring life to your robot using ROS robotic applications - Carol Fairchild, Thomas L. Harman, 2016  
[Book](#) | Further Reading

Programming robots with ROS: a practical introduction to the Robot Operating System - Morgan Quigley, Brian Gerkey, William D. Smart, c2015  
[Book](#) | Further Reading

Learning robotics using Python: design, simulate, program, and prototype an interactive autonomous mobile robot from scratch with the help of Python, ROS, and Open-CV! - Lentin Joseph, 2015  
[Book](#) | Further Reading

ROS robotics projects: build a variety of awesome robots that can see, sense, move, and do a lot more using the powerful Robot Operating System - Lentin Joseph, 2017  
[Book](#) | Further Reading



# Support

- Use the module's MS Team (not Discord ;- ) ,
  - Also to help eachother!
- Be active in the workshops, utilise our team's expertise and experience
- Lots of resources on the module's GitHub page:  
<https://github.com/LCAS/teaching/wiki/CMP3103>
- Consider the extra materials on Blackboard
- Consider trying a home/laptop install if you want (early, not a week before submission), see  
<https://github.com/LCAS/teaching/wiki/Using-the-Docker-Image>

# Robot

- What is it?
  - “I can't define a robot, but I know one when I see one.”, J. Engelberger
- What is it for?
  - help/replace humans in monotonous, boring, repetitive, dangerous, difficult tasks
  - companion? helper? servant?



## What do you associate with the word “Robot”? - PollEV

- <https://pollev.com/mhanheide>



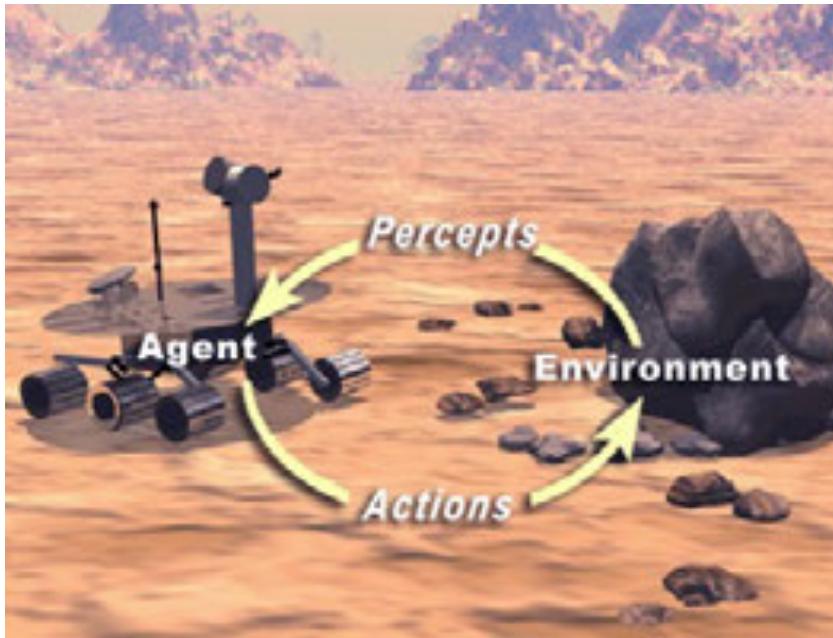
# What do you associate with the word robot?

Start the presentation to see live content. For screen share software, share the entire screen. Get help at [pollev.com/app](https://pollev.com/app)



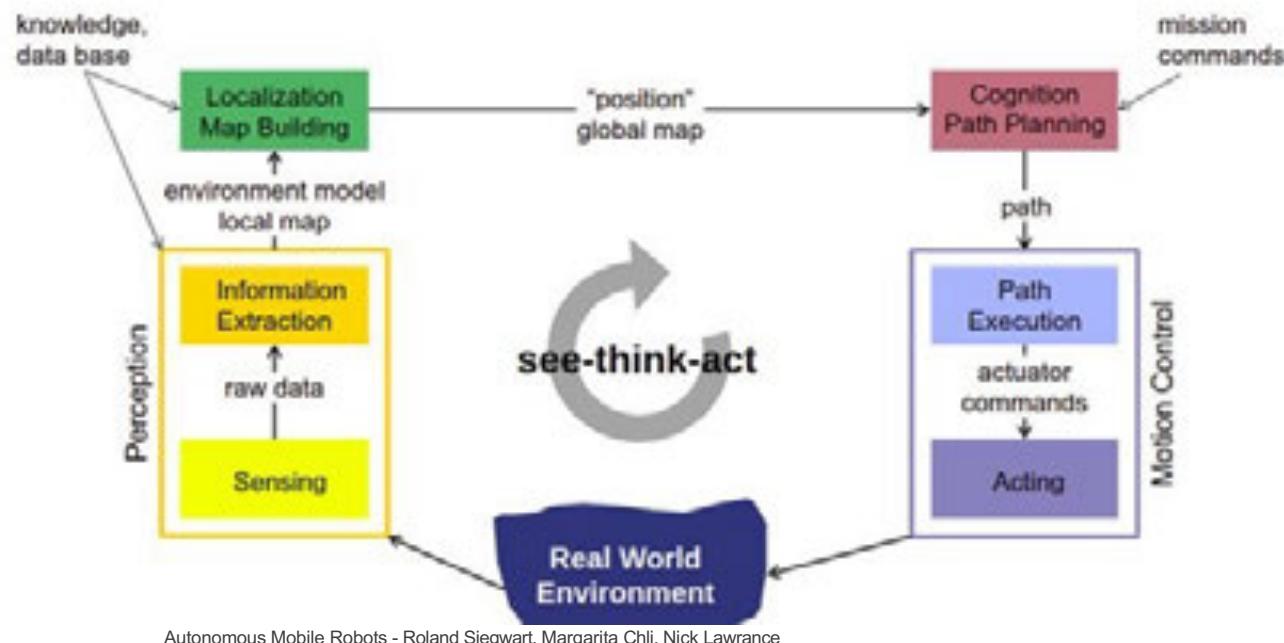
# *ROBOTICS - A light introduction*

# Robotics



Robotics = "the intelligent connection of perception and action" (Brady 1985)

# Typical Overview of Mobile Robots



## Which of the following features are essential for a robot

Hardware Controllers

Actuators

Locomotion (e.g. wheels  
or legs)

Embodiment (physical  
interaction with the world)

Internal Sensors

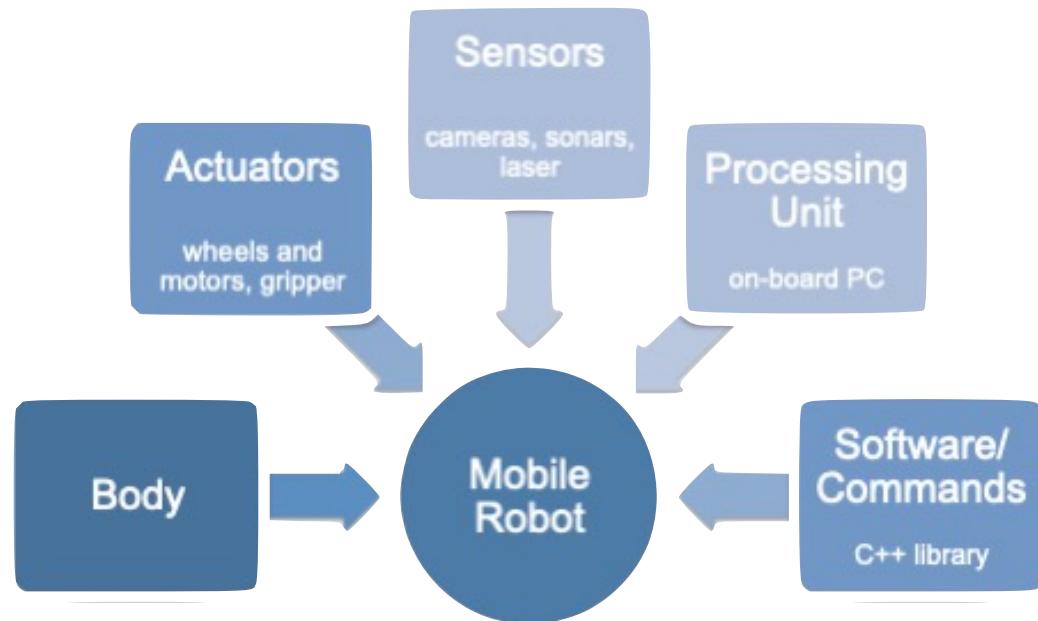
External Sensors

# Mobile Robots

- Mobility
  - opens possibilities for new tasks: transportation, surveillance, cleaning etc.
  - unstructured environments
  - main challenge: navigation
- Autonomy
  - reasoning: making decisions, plan
  - learning from experience
  - building representation of the (dynamic) environment

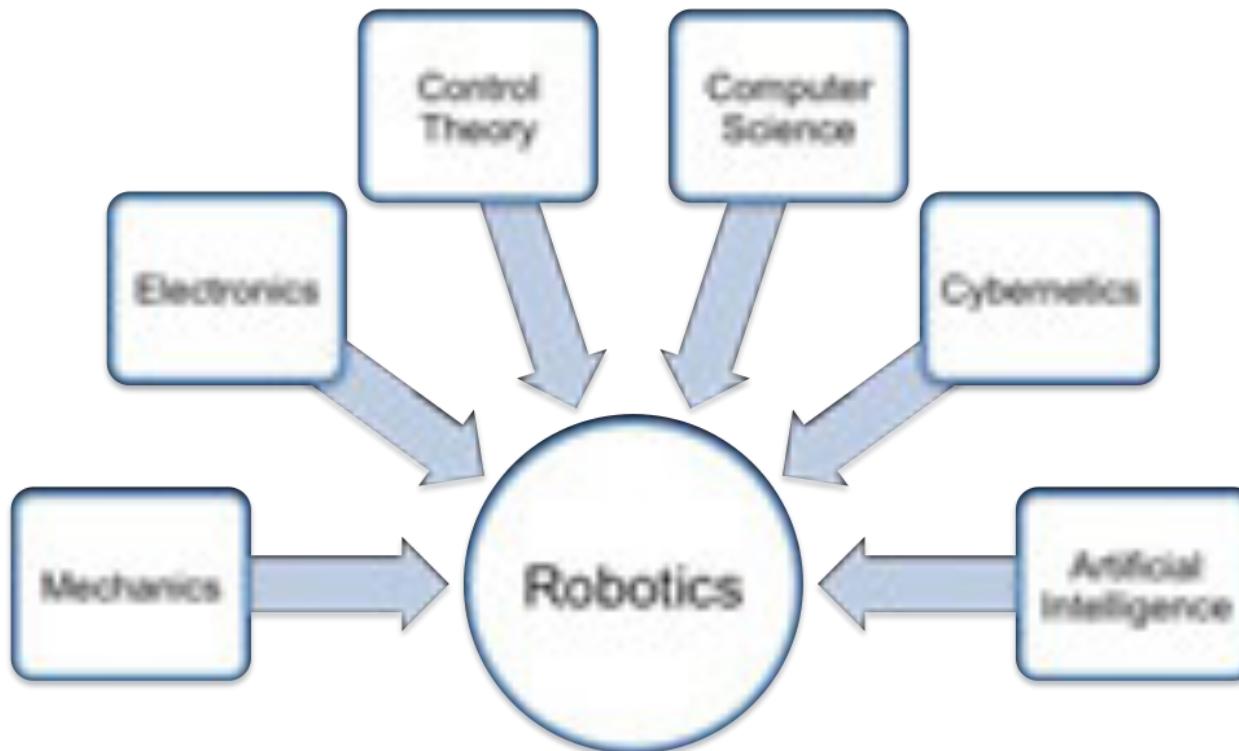


# Anatomy of a Mobile Robot

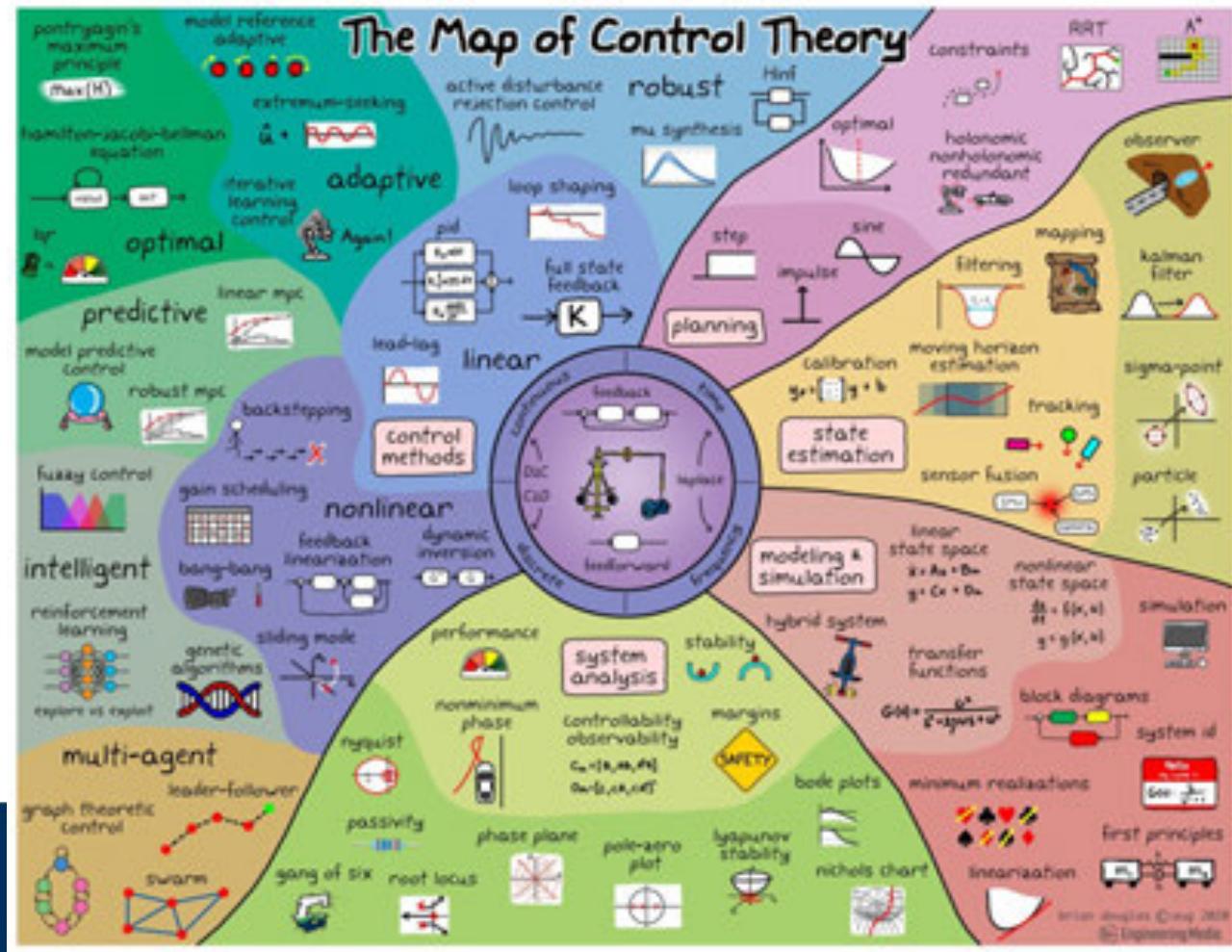
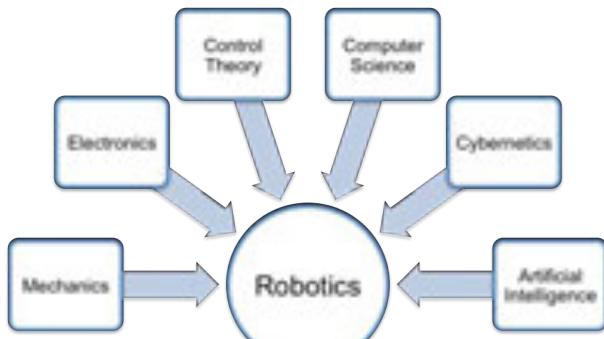


# Robotics

Science and technology of robots

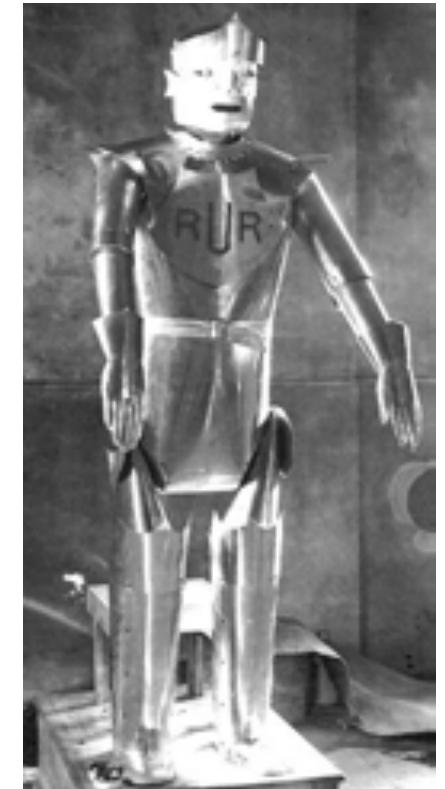


# “The map of control theory”



# R.U.R.

- Karel Čapek, 1921
  - R.U.R. - Rossum's Universal Robots, a stage play
  - the word 'robot' appears for the first time
  - 'roboťa' – forced/hard labour in Czech
  - robots – artificial men that can think but seem to be happy to serve
  - exploited (?) by humans
  - finally rebel against their creators
- E.g. see [https://www.youtube.com/watch?v=t\\_oI37K1B8](https://www.youtube.com/watch?v=t_oI37K1B8)



# Grey Walter's Turtles

- “Machina Speculatrix”, 1948
  - experiments in reflex behaviour
  - built of electronic valves and photo-cells
  - approach or escape a light source
  - => see “Braitenberg Vehicle” later



# Shakey

- Stanford Artificial Intelligence Centre, 1966
  - first mobile robot that could be programmed for various tasks
  - on-board I/O logic
  - radio-link
  - external computer (PDP-10)



# Industrial Robots

- Manipulators
  - “big arms”
  - precise, strong and fast
  - well studied
  - many manufactured
  - operate in controlled environments
  - limited sensory abilities
  - pre-programmed



# Industrial Robots



“Big arms”



AGVs

# Applications

- Exploration
- Surveillance
- Security
- Care assistants
- Agricultural robots
- Intelligent vehicles
- many others...



Which jobs are most likely to be replaced by Robots/AI? Put the most likely one to the top.

Bin man

Investment Banker

Law Associate

Dentist

Programmer

Brick layer

Fruit picker

Lecturer

Accountant

Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://PollEv.com/app)

# Replaced by Robot?

- <http://www.telegraph.co.uk/news/2017/09/27/jobs-risk-automation-according-oxford-university-one/>

## Which jobs are at risk?

Researchers at Oxford University published a widely referenced [study in 2013](#) on the likelihood of computerisation for different occupations.

Out of around 700 occupations, 12 were found to have a 99 per cent chance of being automated in the future:

- Data Entry Keyers
- Library Technicians
- New Accounts Clerks
- Photographic Process Workers and Processing Machine Operators
- Tax Preparers
- Cargo and Freight Agents
- Watch Repairers
- Insurance Underwriters
- Mathematical Technicians
- Sewers, Hand
- Title Examiners, Abstractors, and Searchers
- Telemarketers

**What is the hardest part to make this robot work?**





## *SENSORS AND ACTUATORS*



# Effectors and Actuators

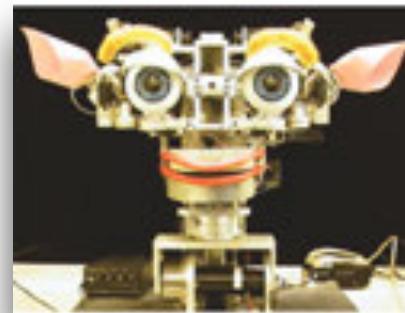
- Effectors:
  - hand, arm, gripper – manipulators
  - wheels, legs, tracks, rotors – mobile robots
- Actuators:
  - electric motors, pneumatic and hydraulic systems



## Effectors – Examples



pan-tilt unit



robotic head



parallel arm



gripper

# Sensors: Classification

- Proprioceptive
  - internal state of the robot, self-awareness
- Exteroceptive
  - state of the environment

## Which of the following are exteroceptive sensors?

Laser scanner

Sonar

CPU Thermometer

Wheel encoder

Radar

Microphone

Camera

Speaker

Gyroscope

GPS

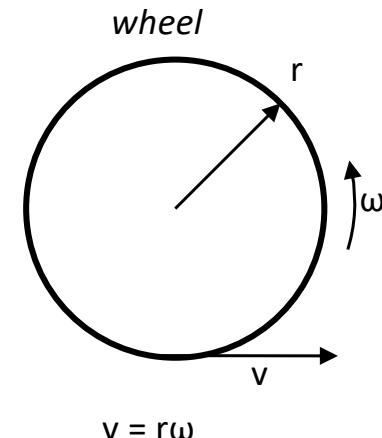
Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://PollEv.com/app)

# Sensors: Classification

- Proprioceptive
  - internal state of the robot, self-awareness
  - e.g. odometry, battery level, temperature
- Exteroceptive
  - state of the environment, light intensity, distance measurements
  - e.g. sonars, video cameras
- Passive vs. Active
  - measuring phenomena directly or indirectly (e.g. reflected light/sound)

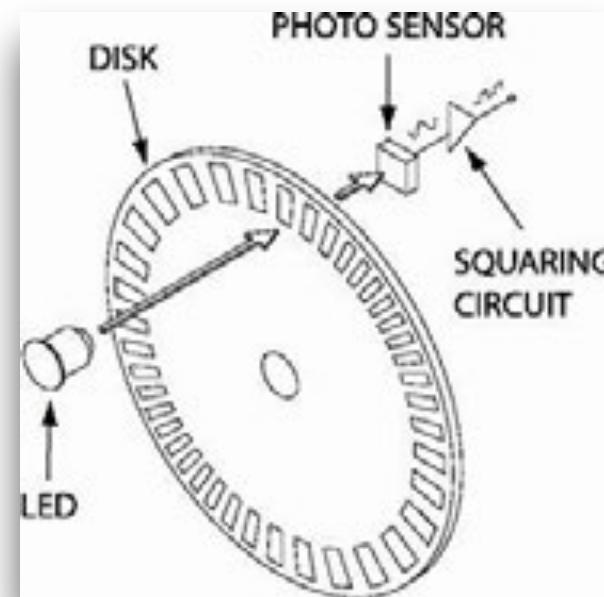
# Odometry – Where Am I?

- Dead reckoning
  - “deduced reckoning” – marine navigation
  - position estimation based on the previous known position
  - used by animals: pigeons, ants
- Mobile robots – odometry sensor
  - measure the speed of each wheel
  - use wheel geometry to calculate the velocity of a robot



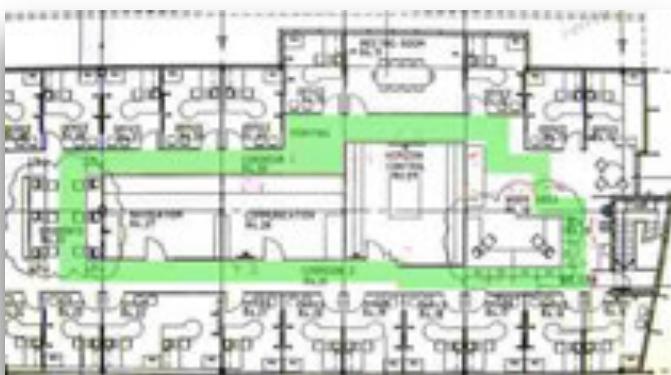
# Wheel Speed Estimation

- Nominal motor speed + gear ratio
  - provided by the motor manufacturer
  - may change under different loads
- Motor Encoder
  - sensor mounted on the wheel shaft
  - counts motor revolutions
  - similar to a computer mouse

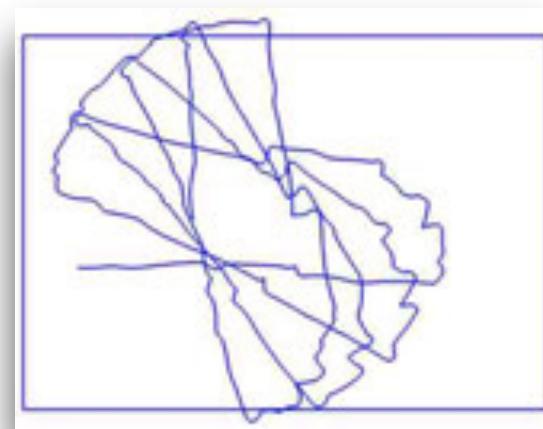


## Odometry – Limitations

- Wheel slippage, uneven friction and wheel size, etc. can cause errors that accumulate with time



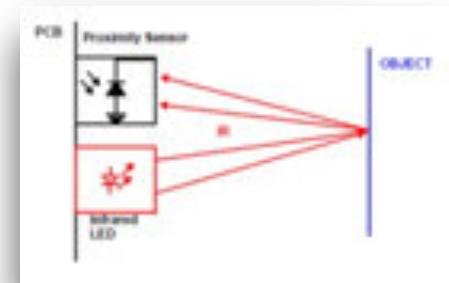
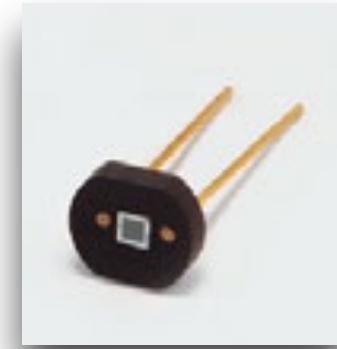
Floor plan and robot route



Robot's perceived trajectory  
using odometry

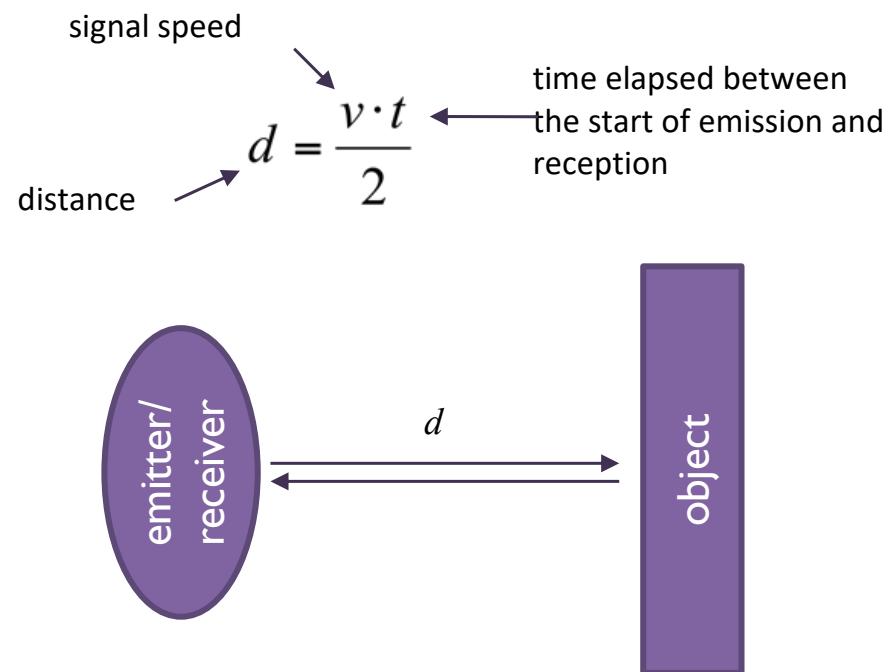
# Light Sensor

- Passive – measuring light intensity directly (e.g. photo-resistor)
  - light can be used to mark important places e.g. recharging station, exit from the room, etc.
- Active – measuring reflected light
  - e.g. IR sensor
  - inexpensive but short range



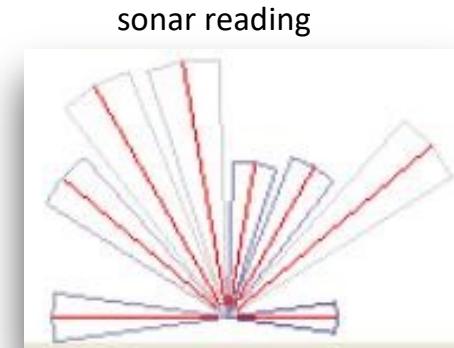
# Time of Flight Sensors

- Active distance measurements – reflected signal



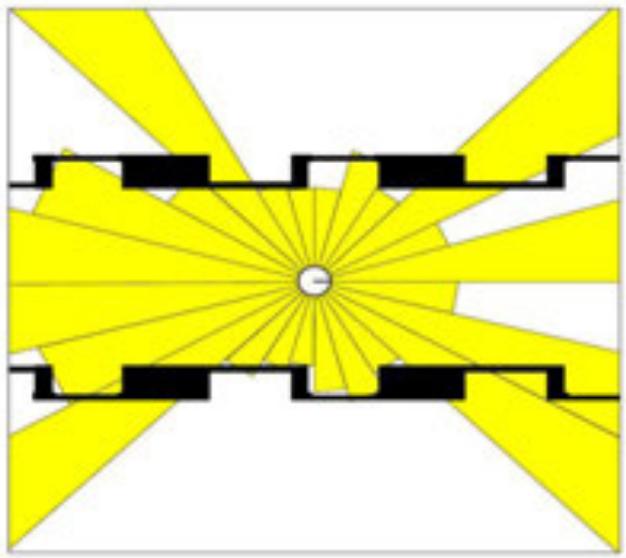
# Sonar Sensor

- Sonar
  - ultrasonic signal
  - $v$  is speed of sound = 343 m/s
  - processing is 'slow'

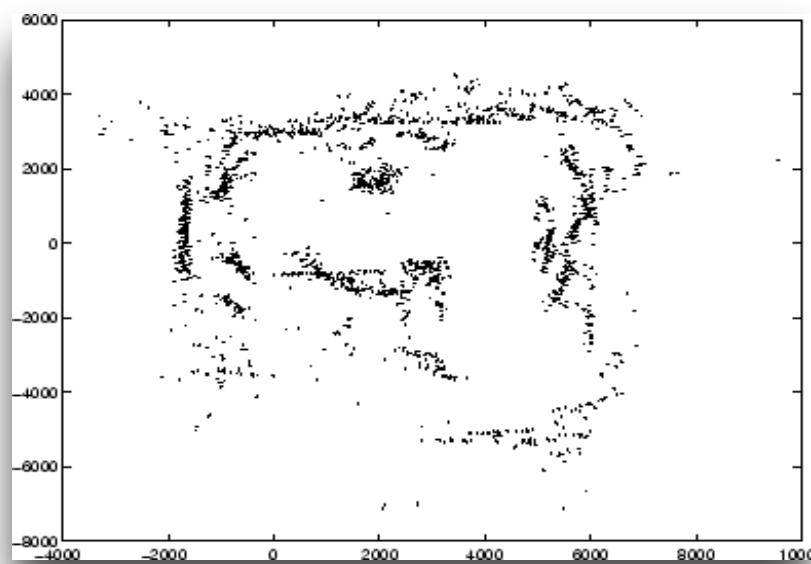


# Sonar – Applications

- Pros: cheap and good for obstacle avoidance
- Cons: slow and noisy



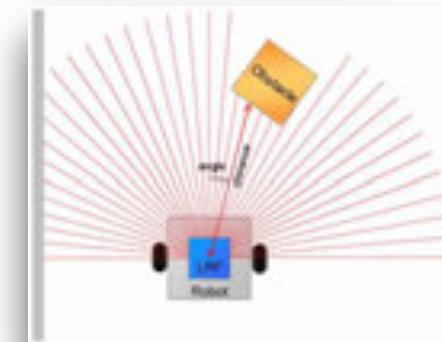
sonar reading in a corridor

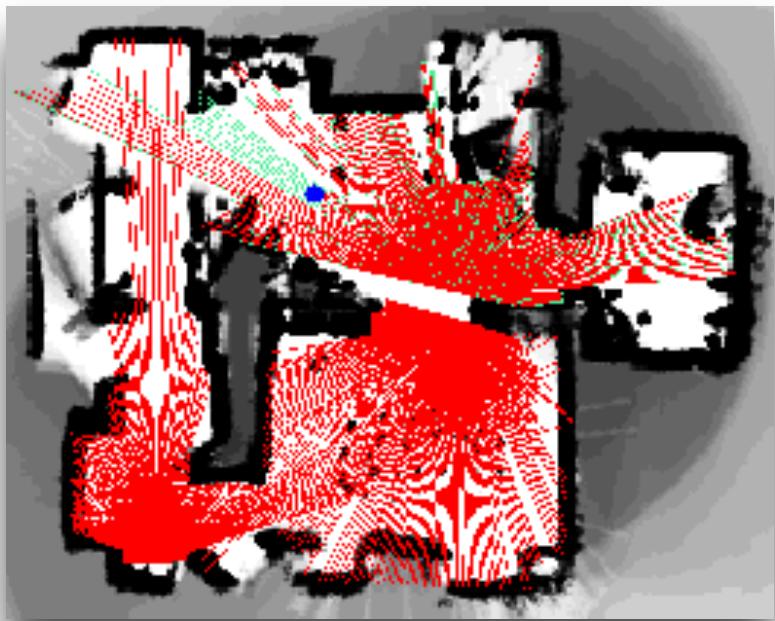


sonar map

# Laser Sensors

- Principle
  - time of flight sensor (light)
  - pulsed laser and rotating mirror
- Characteristics
  - high precision (mm)
  - long range (tens of meters)
  - wide field of view
  - fast (~30 scans/s)





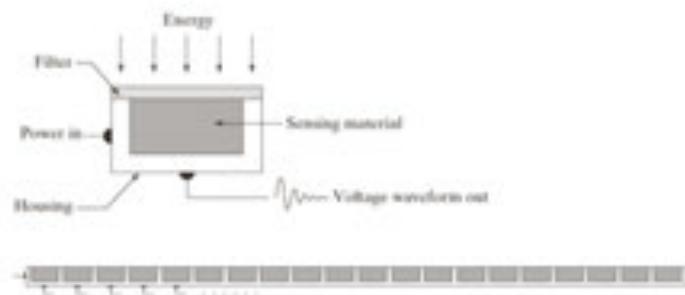
2d maps and people detection



3d maps

# Vision Sensor

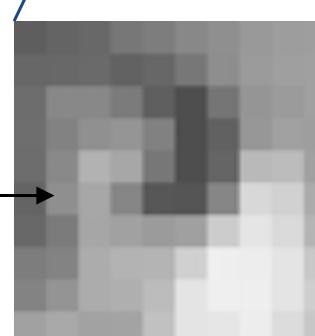
Vision Sensor



Digital Image



*a rat*



*the rat's  
nose*

# Different Type of Vision Sensors

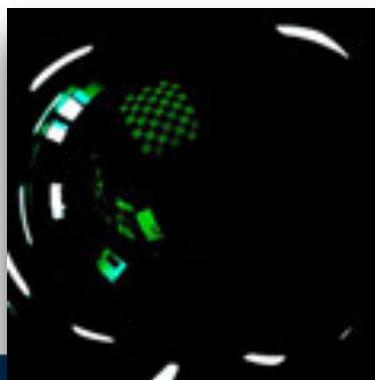
colour



thermal



omni-directional



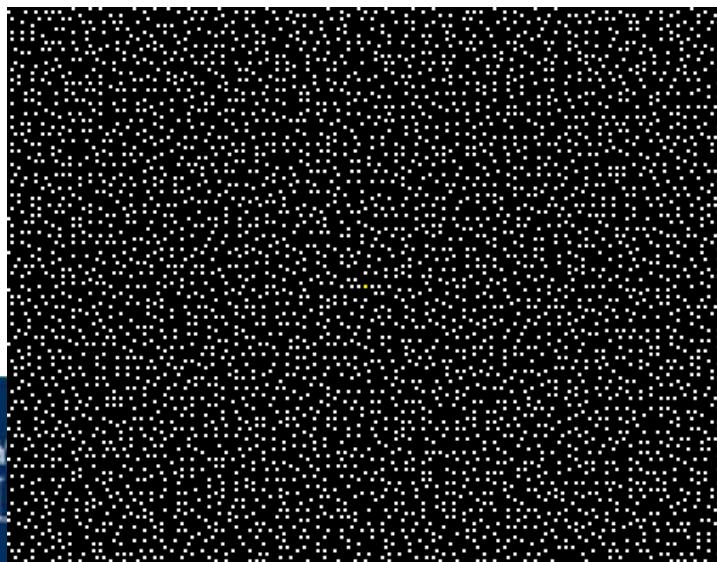
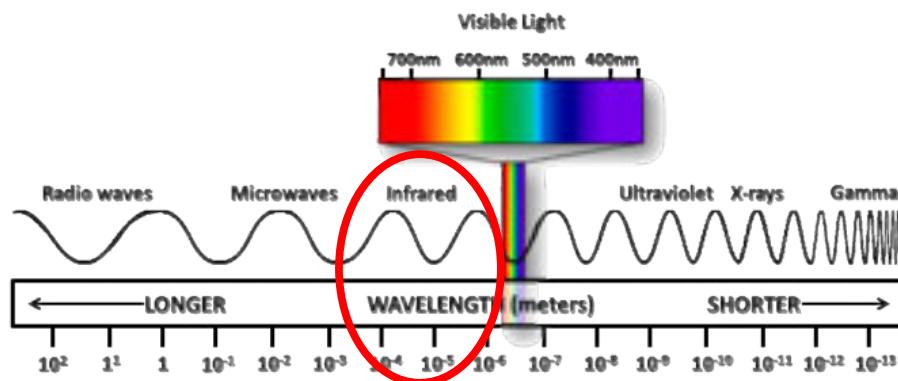
stereo



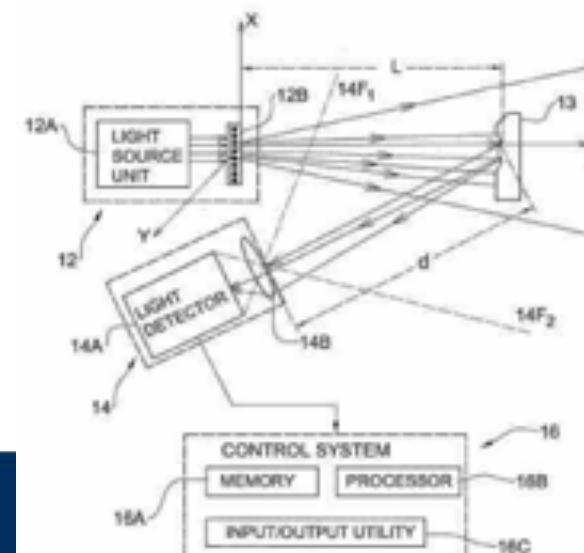
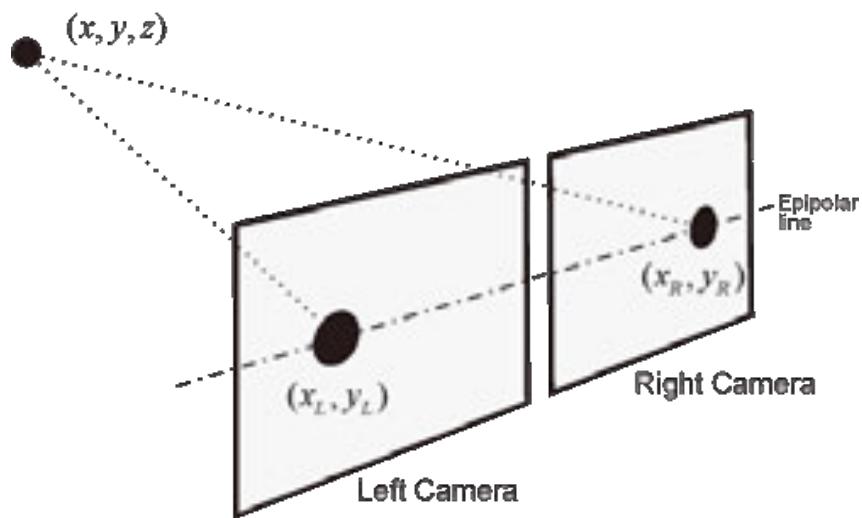
# Kinect / RGBD sensors



# Structured Light - Kinect

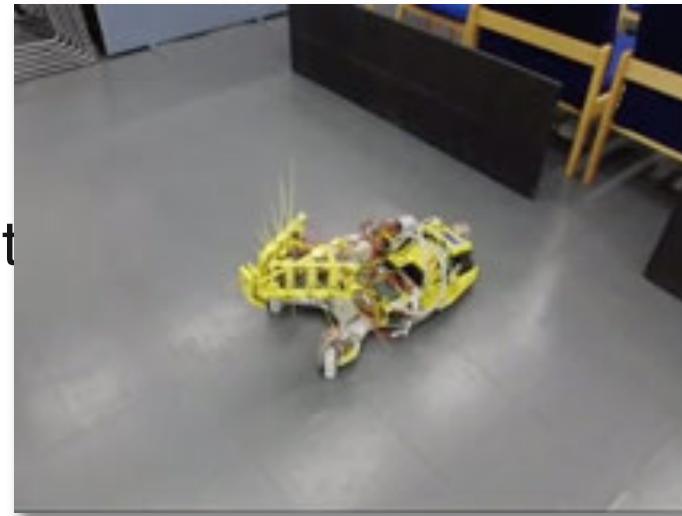


# Depth from Stereo



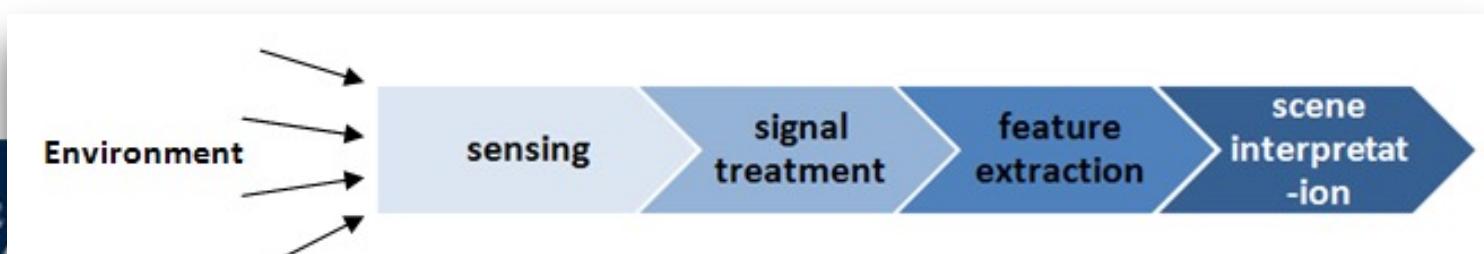
## Other Sensors

- Tactile
  - bumpers, whiskers, buttons
- Direction and orientation
  - compass, gyroscope, accelerometer
- Global position
  - GPS
- Motion
  - Gyroscope
- Temperature, sound, even smell!



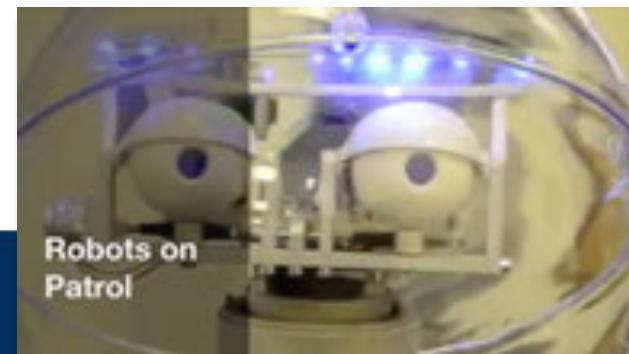
# Perception

- Data Interpretation and Processing
  - sensors do not provide direct measurements nor provide the exact values
  - some sensors provide very rich information (e.g. vision) that needs to be reduced
  - some sensors provide very sparse information that needs to be interpolated
  - The data bandwidth of sensors differs significantly!



# Robotics Research at Lincoln

- Our webpage: <https://lcas.lincoln.ac.uk>
  - Human-Centered Robotics
  - Agri-Food Technology
  - Bio-inspired Embedded Systems
  - Learning for Autonomous Systems
- Applications
  - Security
  - Assistive Care
  - Agricultural Robotics
  - Intelligent Transportation
  - Nuclear Robotics

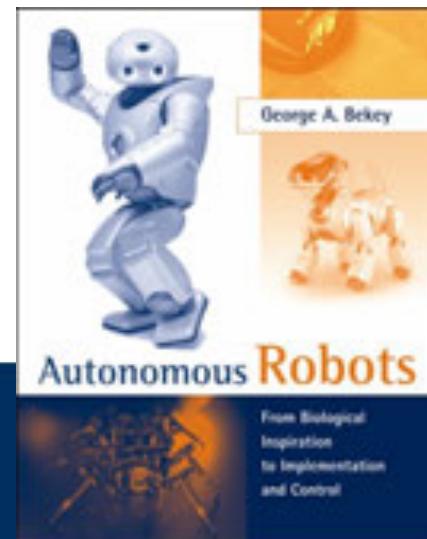




Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](http://PollEv.com/app)

# Recommended Reading

- Gates, B. A Robot in Every Home, Scientific American, 2006
- Siegwart et al. Autonomous Mobile Robots, 2004 (chapter 1)
- Bekey, G.A. Autonomous robots – Chapter 1 (also on Blackboard)



<https://attendance.lincoln.ac.uk/>



## CMP3103 – AUTONOMOUS MOBILE ROBOTS

*Lincoln Centre for Autonomous Systems  
School of Computer Science*



Access Code: 625951



# Syllabus

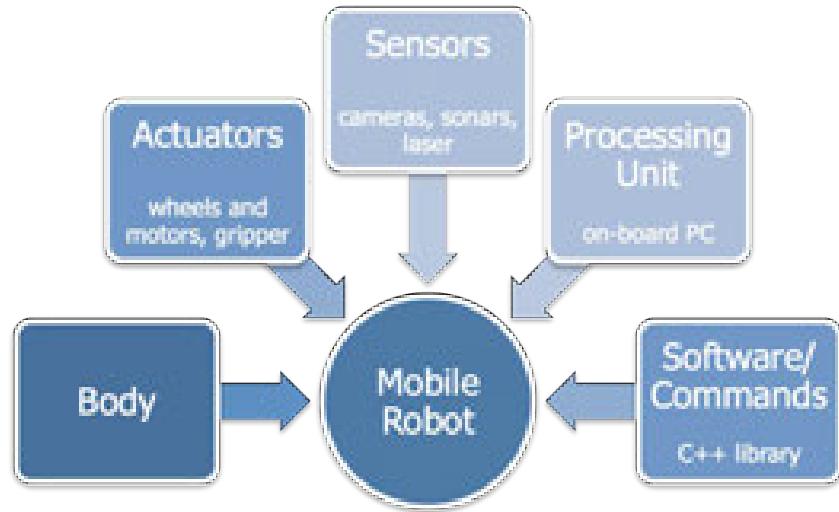
- Introduction to Robotics
- **Robot Programming**
- Robot Vision
- Robot Control
- Robot Behaviours
- Control Architectures
- Navigation Strategies
- Map Building

## Disclaimer:

These slides are not self-contained, this is going to be an interactive lecture

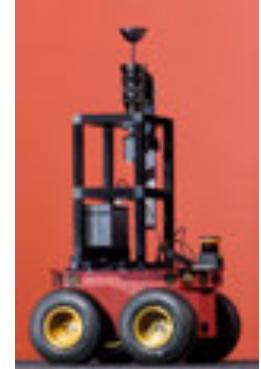


# Mobile Robot Components



# Processing Unit

- On-board
  - fast responses
  - embodied, processing power limited by the physical size of a robot
- Off-board
  - remote computer(s) - significant processing power
  - problems with communication, data transfer and synchronisation
- Hybrid architecture
  - on board for low-level tasks
  - PC for higher-level tasks



# Robot Software

- Hardware drivers, (real-time) operating system
- Audio/video encoders
- Command interface, firmware
- Sensor/Image processing library
- Software components implementing AI, navigation, decision making
- Simulator

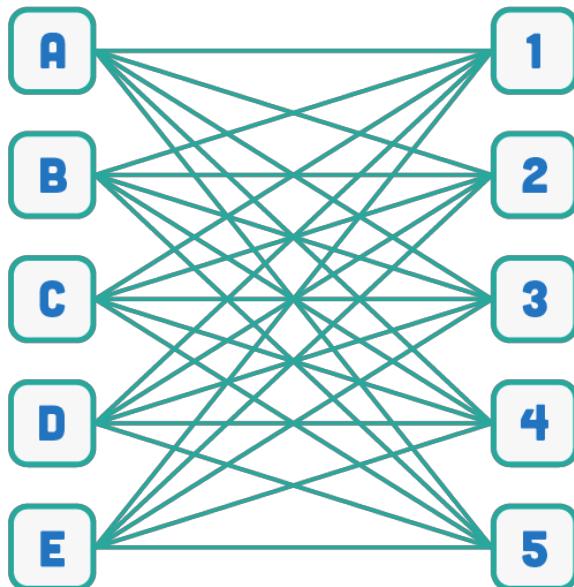
# How to Talk to a Robot

- We need to be able to:
  - send motor/actuator commands
  - read and process data from sensors
- in some robots there is an abstraction layer featuring a shared domain-specific command language to access all sensors and actuators (highly integrated)
  - implemented inside the robot
  - can be messages sent through serial port, Ethernet (Wifi)
- Our robots are more modular, different sensors talk via USB with the onboard NVIDIA Jetson nano
  - they have their dedicated ROS driver nodes to talk to us
- HENCE, we need to be able to **communicate!**

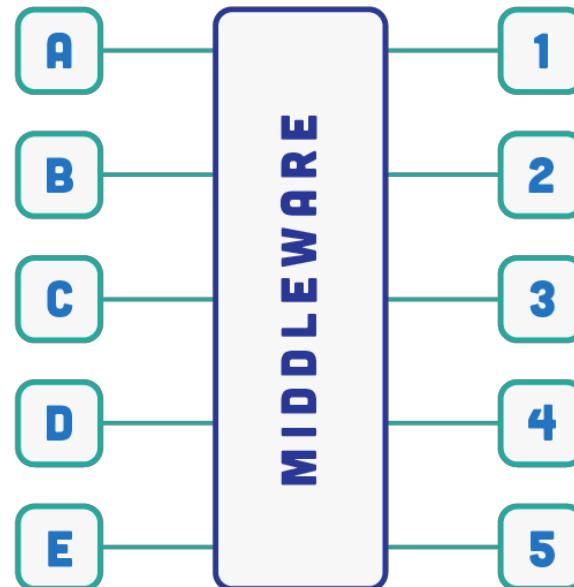


# Middleware?

## WITHOUT MIDDLEWARE

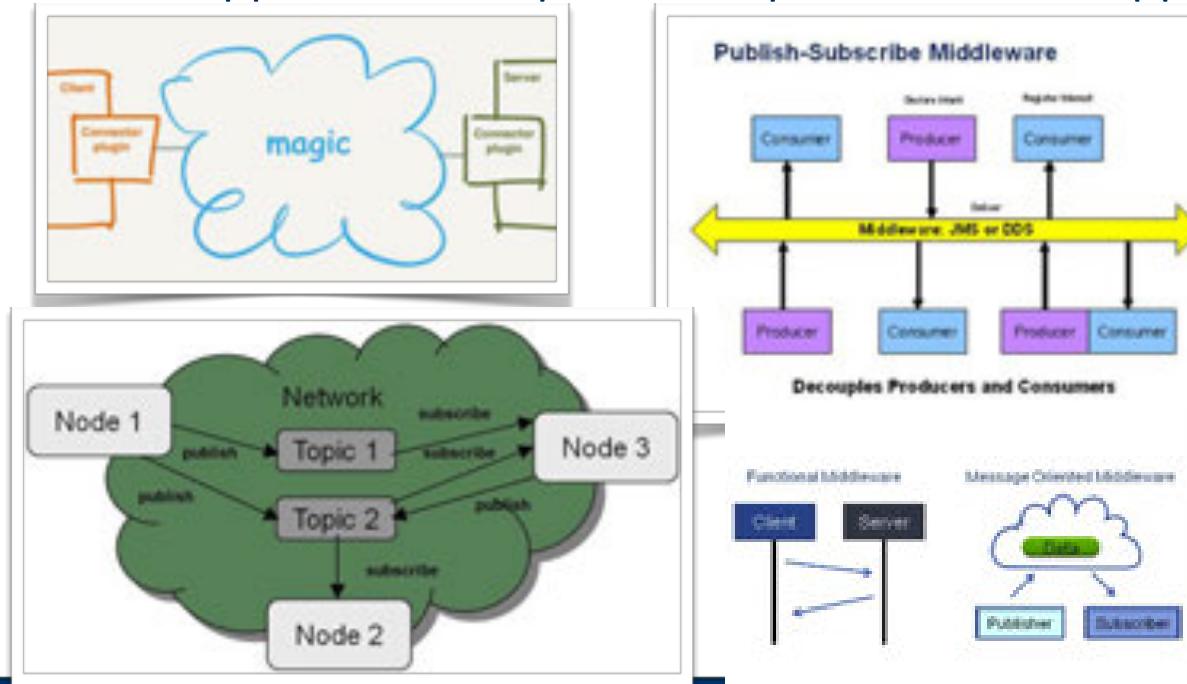


## WITH MIDDLEWARE



# Middleware?

Middleware supports and simplifies complex distributed applications.



# Robotic Middlewares

- Support for different robots, platforms, unified interface, plug-ins/modules (e.g. navigation, object recognition), simulator, etc.
  - **Robot Operating System (ROS)**
  - Microsoft Robotics Developer Studio for Windows
  - OROCOS
  - YARP
  - RSB
  - ...

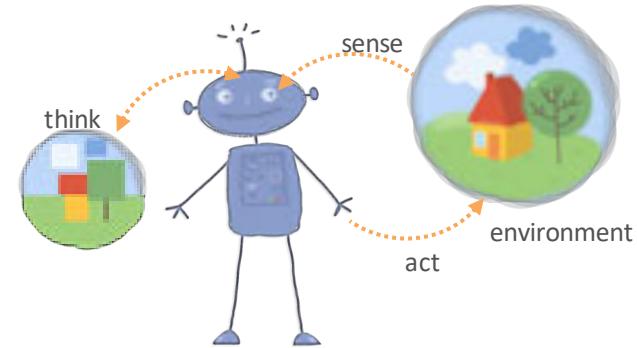


# ROS

- ROS is a communication middleware with a huge library of state of the art algorithms being freely available
- ROS encapsulates functionality into individual nodes
- Nodes communicate via **data streams**
  - nodes implement callback (**data push**)
- C++, java, Python (and others more) supported

# Implementing Tasks/Behaviours

- Scripts
  - A pre-programmed sequence of commands
- Continuous operation (robot control)
  - Sense
    - read sensor data
  - Think
    - process data and make decisions
  - Act
    - execute actions (send movement commands)



# sense - (think) - act

- Two Options
  - Synchronous:
    - like a while loop:

```
while (true)
{
    robot.sense();
    robot.think();
    robot.act();
}
```

- Asynchronous:
  - different threads with shared (and synchronised) memory access

data pull

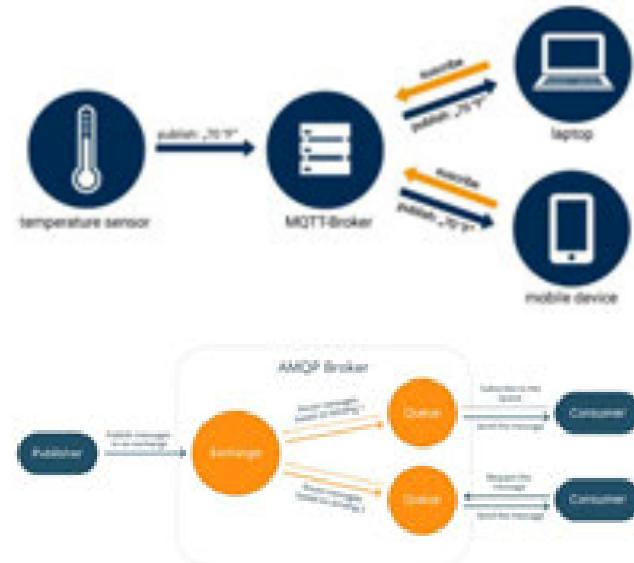
Easier

data callbacks (data push)

basically how ROS works

# Other publish subscribe architectures

- “Observer Pattern”: Publish-Subscribe (often over topics)
- MQTT: Used for a lot IoT applications
- Middlewares that support publish-subscribe pattern (often among other)
  - AMQP (Advanced Message Queuing Protocol)
  - Enterprise Service Bus (ESB)
- RabbitMQ: precursor to AMQP standard
- OMG standard: DDS (underlying middleware for ROS2)





Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://PollEv.com/app)

# *Introduction to ROS2*

With gratitude to Southwest Research Institute and the ROS industrial project

# Outline

- Intro to ROS
- ROS Workspaces & Colcon
- Installing packages (existing)
- Packages (create)
- Nodes
- Messages / Topics

# ROS1 and ROS2

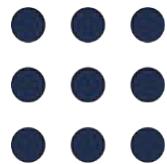
- ROS1 has been around since 2008
  - Uses custom TCP/IP middleware
- ROS2 is a ground-up reimaging of ROS
  - Started in 2014
  - Built on DDS, middleware proven in industry
  - Now on 9th named release

ROS2

# ROS1 and ROS2

- Community is currently in transition!
  - Final ROS1 release (Noetic) is out (EOL in 2025)
  - All critical features are now supported in ROS2
- ROS-Industrial will take time to transition
  - Many breaking changes / conceptual differences
  - Vision is industrial robots will become native ROS devices

# ROS Versions



**ROS 1**

Box Turtle  
Mar 2010

...



Lunar  
2017 - 2019

Melodic  
2018 - 2023

Noetic  
2020 - 2025

EOL



**ROS 2**



Ardent

...



Foxy (LTS)

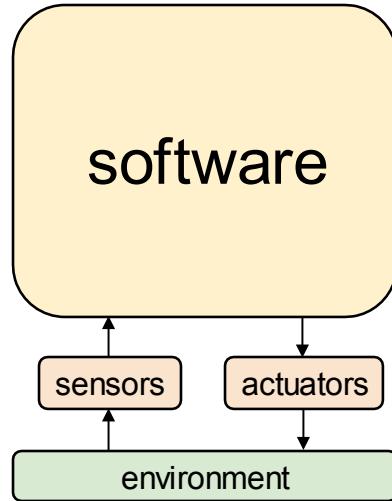


Galactic



Humble

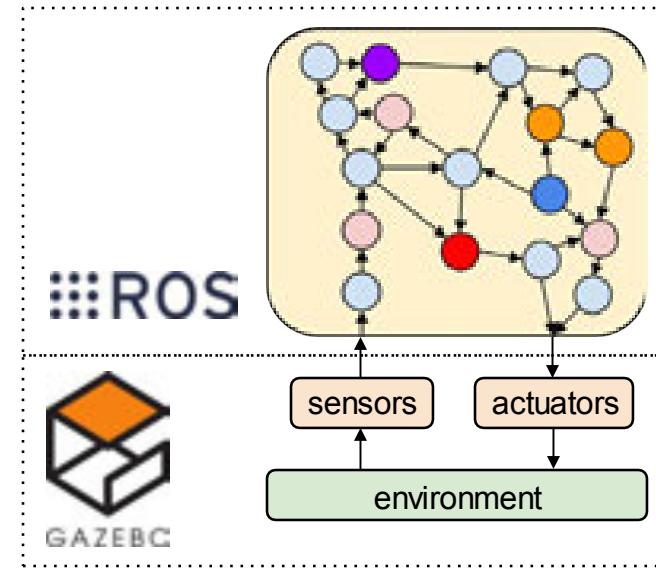
# ROS: The Big Picture



All robots are:  
Software connecting Sensors to Actuators  
to interact with the Environment

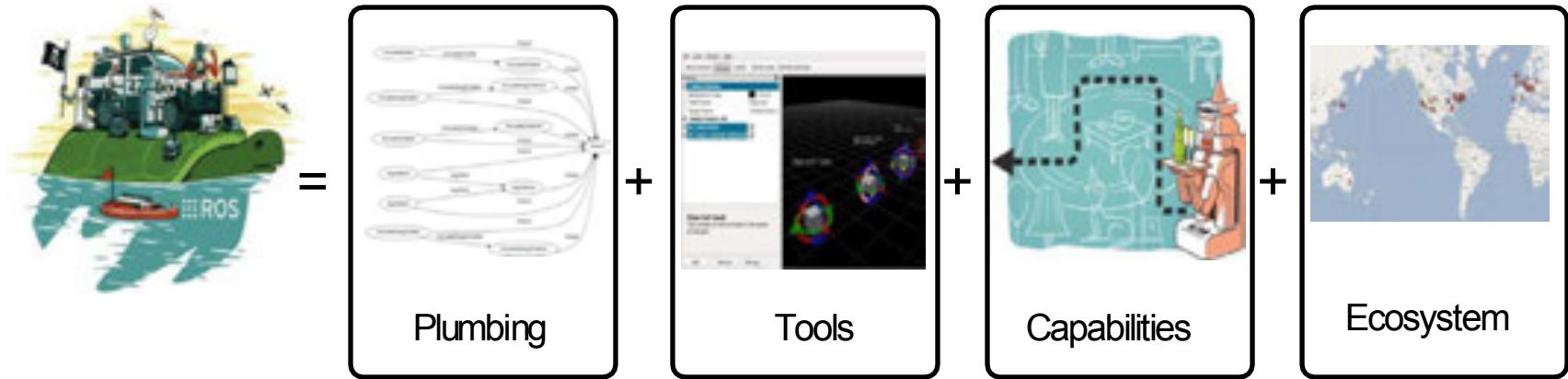
# ROS : The Big Picture

- Break Complex Software into Smaller Pieces
- Provide a framework, tools, and interfaces for distributed development
- Encourage re-use of software pieces
- Easy transition between simulation and hardware

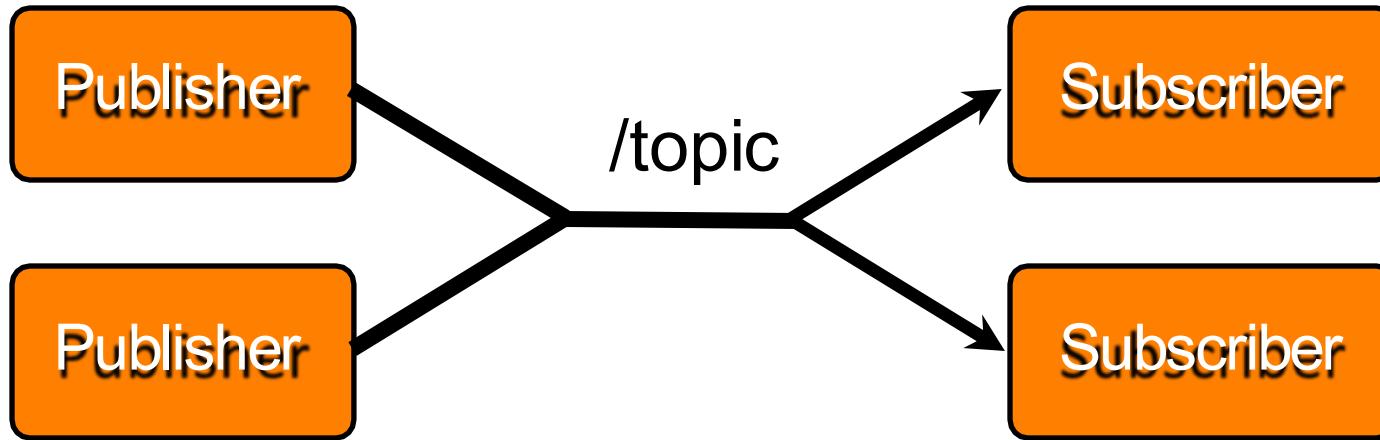


# What is ROS?

- ROS is...



# ROS is... plumbing

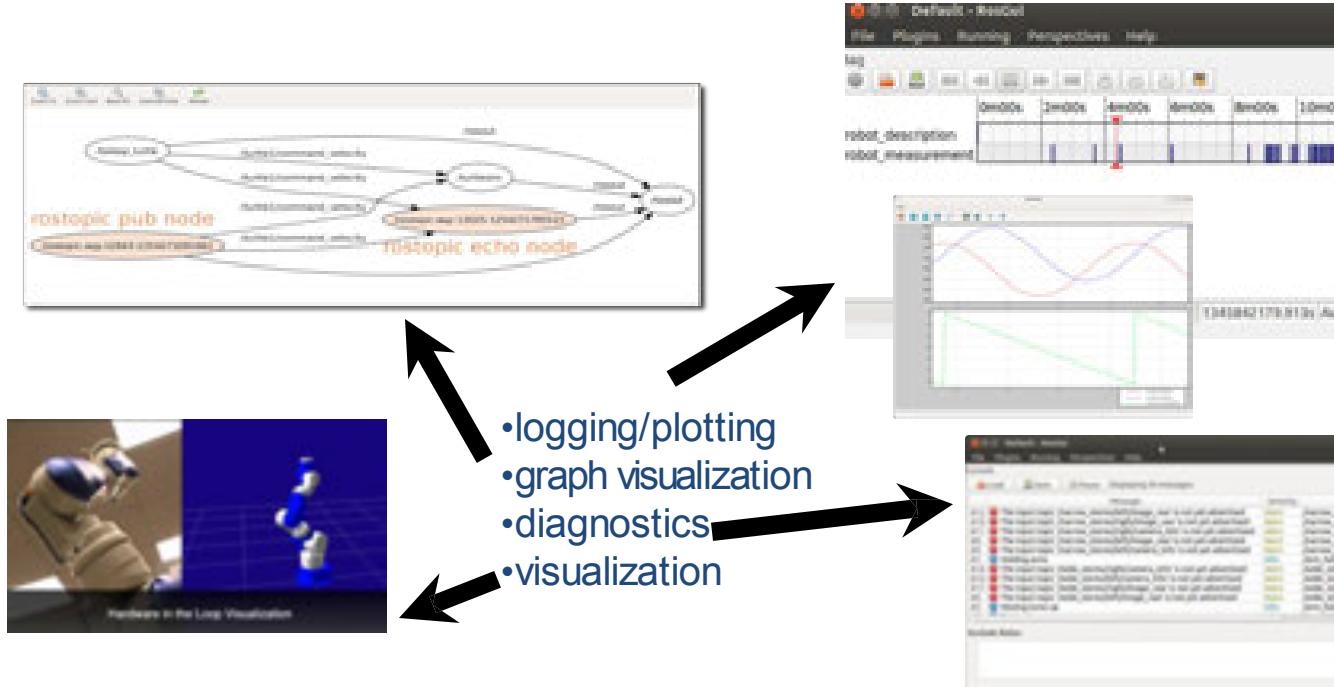


# ROS Plumbing : Drivers

- 2d/3d cameras
- laser scanners
- robot actuators
- inertial units
- audio
- GPS
- joysticks
- etc.

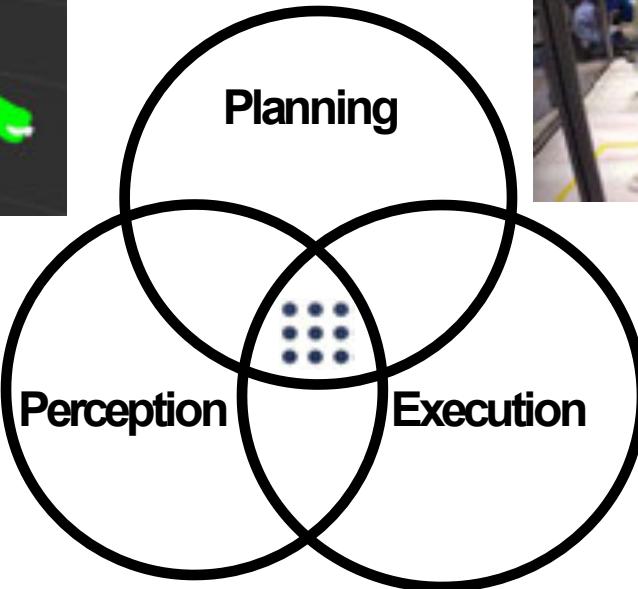
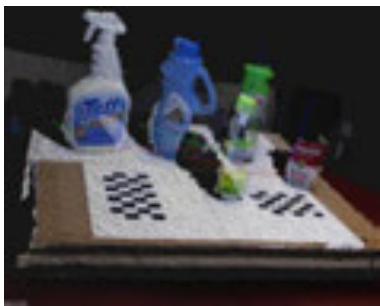


# ROS is ...Tools



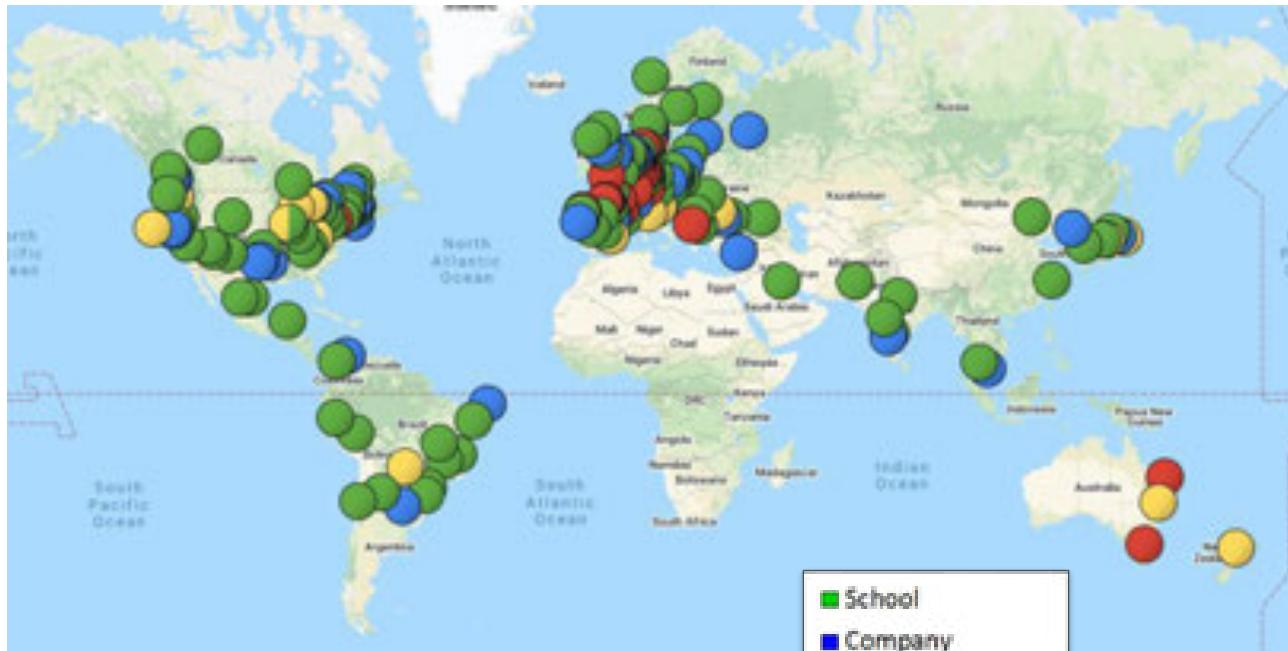
- logging/plotting
  - graph visualization
  - diagnostics
  - visualization

# ROS is...Capabilities



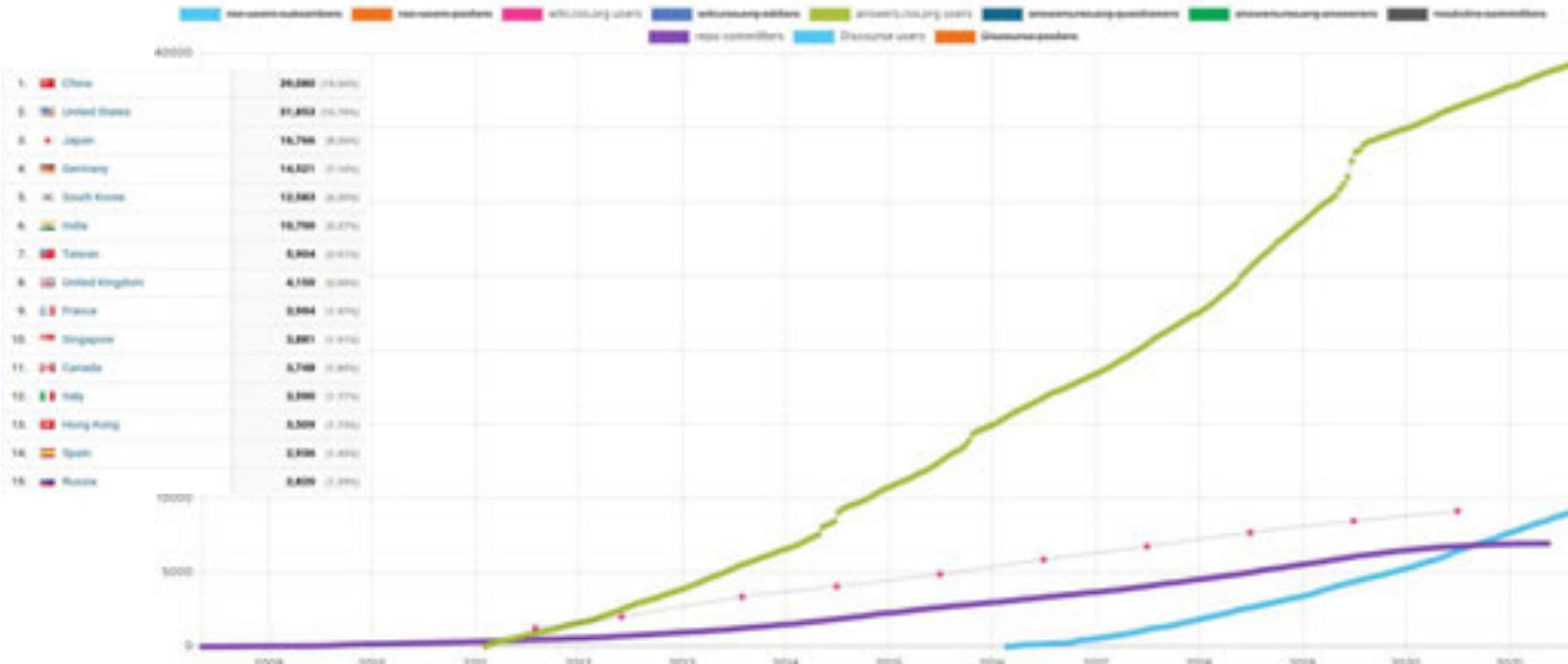
(Image from Willow Garage's "What is ROS?" Presentation)

# ROS is... an Ecosystem



# ROS is a growing Ecosystem

Number of ROS Users



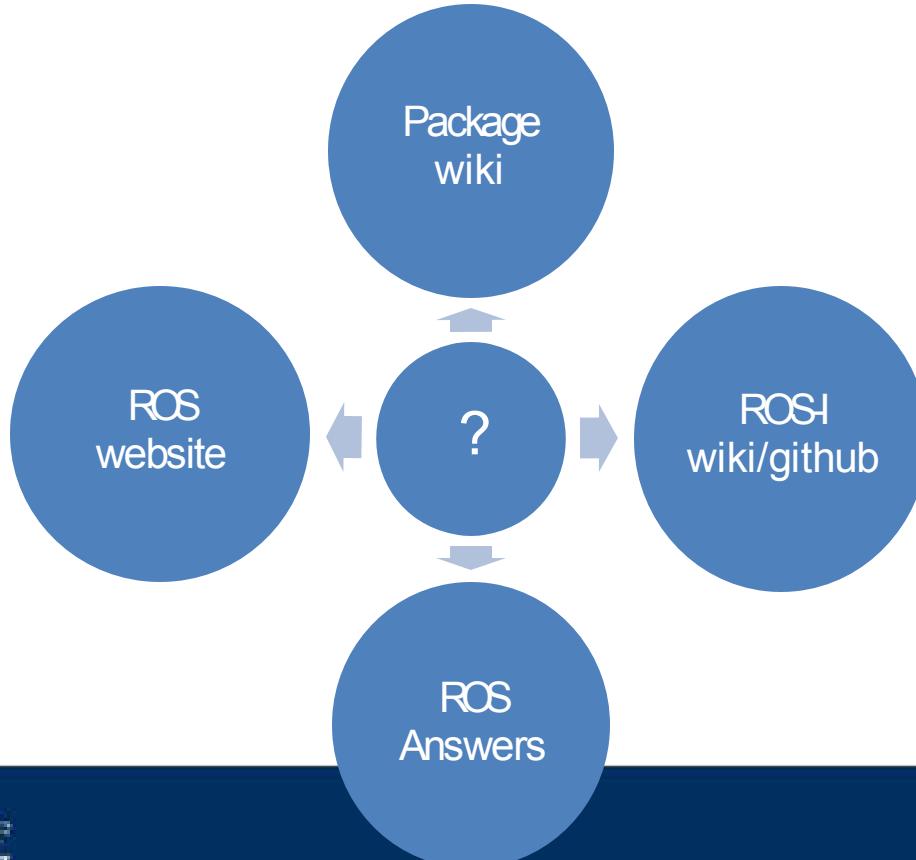
A collection of different metrics for measuring the number of users in the ROS community.

<https://metrics.ros.org/>

# ROS Programming

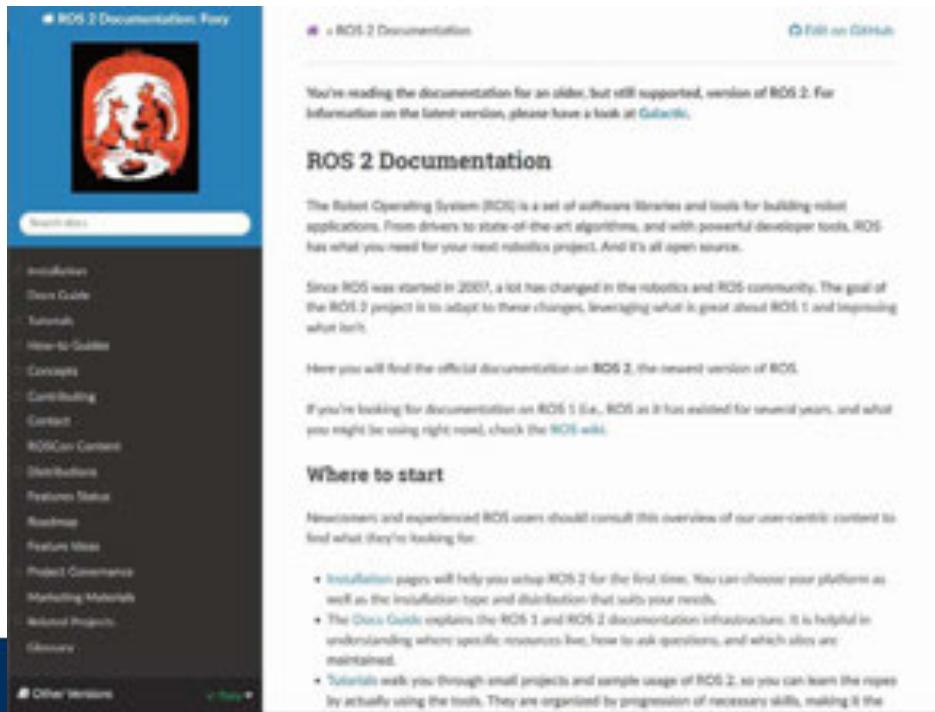
- ROS uses platform-agnostic methods for most communication
  - DDS, TCP/IP Sockets, XML, etc.
- Can intermix programming languages
  - Current 1st Tier support: C, C++, Python
  - We will be using Python for our exercises

# ROS Resources



# ROS2 Documentation

<http://docs.ros.org/en/humble/>



You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at [Gentoo](#).

## ROS 2 Documentation

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

Since ROS was started in 2007, a lot has changed in the robotics and ROS community. The goal of the ROS 2 project is to adapt to these changes, leveraging what is great about ROS 1 and improving what isn't.

Here you will find the official documentation on ROS 2, the newest version of ROS.

If you're looking for documentation on ROS 1 (i.e., ROS as it has existed for several years, and what you might be using right now), check the [ROS wiki](#).

### Where to start

Newcomers and experienced ROS users should consult this overview of our user-centric content to find what they're looking for:

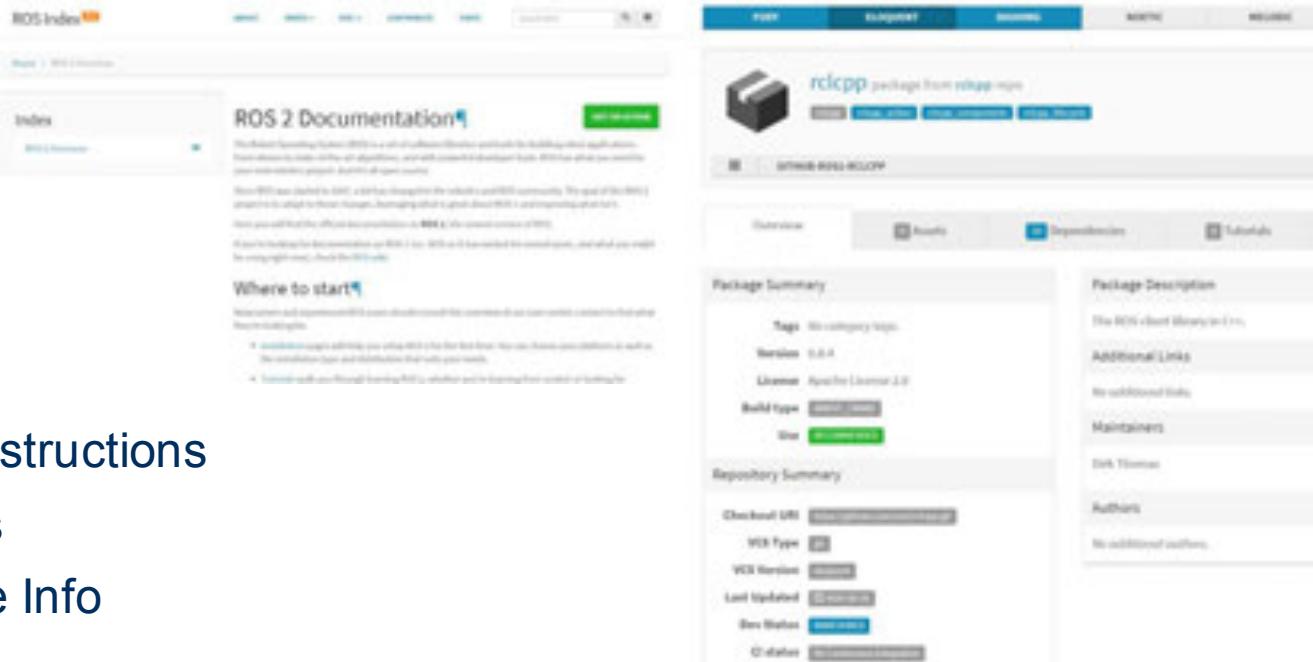
- Installation pages will help you setup ROS 2 for the first time. You can choose your platform as well as the installation type and distribution that suits your needs.
- The Docs Guide explains the ROS 1 and ROS 2 documentation infrastructure. It is helpful in understanding where specific resources live, how to ask questions, and which sites are maintained.
- Tutorials walk you through small projects and sample usage of ROS 2, so you can learn the ropes by actually using the tools. They are organised by progression of necessary skills, making it the

- Install
- Tutorials
- Concepts

...

# ROS Package Index

<http://index.ros.org>



The screenshot shows the ROS Package Index website. On the left, there is a search bar and a navigation menu. The main content area displays the 'rcicpp' package page. The page title is 'rcicpp package from rosdep-index'. It features a 3D cube icon representing the package. Below the title, there are tabs for 'Overview', 'Assets', 'Dependencies', and 'Tutorials'. The 'Dependencies' tab is currently selected. The 'Package Summary' section includes fields for 'Tags' (no category tags), 'Version' (0.0.4), 'License' (Apache License 2.0), 'Build type' (C++), and 'Use' (CMake). The 'Repository Summary' section shows 'Checkout URL' (https://github.com/rdl-rosdep/rdl-rcicpp), 'VCS Type' (git), 'VCS Revision' (0.0.4), 'Last Updated' (2023-07-10), 'Dev Status' (Development), and 'CI Status' (Green). The 'Package Description' section notes that it is a ROS client library in C++ and contains no additional links. The 'Maintainers' and 'Authors' sections both list 'Dirk Thomas'.

- Install Instructions
- Tutorials
- Package Info

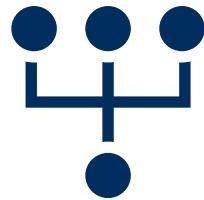
# ROS Answers

<http://answers.ros.org>



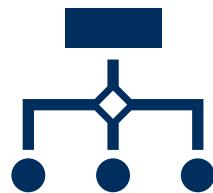
- Quick responses to Good Questions
- Search by text or tag
- Don't re-invent the wheel!

# ROS is a Community



## No Central “Authority” for Help/Support

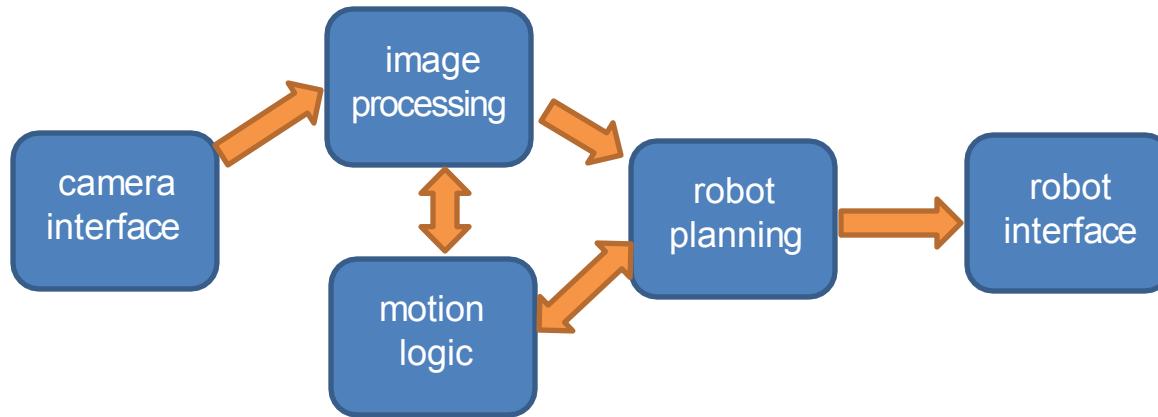
Many users can provide better (?) support  
ROS Consortium can help fill that need



## Most ROS-code is open-source

can be reviewed / improved by everyone  
we count on **YOU** to help ROS grow!

# ROS Architecture: Nodes



- A **Node** is a *standalone* piece of functionality
  - Most communication happens **between** nodes
  - Nodes can run on many different **devices**
  - Often one node per process, but not always

# ROS2 Command line tools!

- ros2 <command>
- ros2 node <command>
  - list
  - info
- ros2 topic <command>
  - list
  - info
- ros2 service ...
- Use rqt

always use [Tab] and “-h”  
when working on the  
command line!

Each **node** can **listen** or **publish** on a topic.

Messages types are defined using a dedicated syntax which is ROS specific:

### **MyMessage.msg**

```
# this is a very useful comment!
float64 myDouble
string myString
float64[] myArrayOfDouble
```

# Setting the Scene: Your workspace

Read the following:

- <https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html>
  - <https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html>
- Tasks
    - 1 Source ROS 2 environment
    - 2 Create a new directory
    - 3 Clone a sample repo
    - 4 Resolve dependencies
    - 5 Build the workspace with colcon
    - 6 Source the overlay
    - 7 Modify the overlay

# Setting the Scene: Create a Package

Read the following:

- <https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html>
  - <https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html>
- **Tasks**
    - 1 Create a package
    - 2 Build a package
    - 3 Source the setup file
    - 4 Use the package
    - 5 Examine package contents
    - 6 Customize package.xml

# Writing a Publisher

Main parts:

- Define a new Node, derived from the ROS2 Node class
- Initialise ROS framework
- Create Publisher(s) in the Initialiser of the Node object
- Develop the programme logic that publishes ROS2 data objects

• Our example:

[https://github.com/LCAS/teaching/blob/lcas\\_humble/cmp3103m\\_ros2\\_code\\_fragments/cmp3103m\\_ros2\\_code\\_fragments/chat\\_sender.py](https://github.com/LCAS/teaching/blob/lcas_humble/cmp3103m_ros2_code_fragments/cmp3103m_ros2_code_fragments/chat_sender.py)

• ROS2 Tutorial:

<https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html>

# Writing a Publisher

Imports:

- Ros Communication Layer for Python (rclpy)
- The ROS2 Node class
- Any ROS2 data type definitions (*interfaces*) we may need to send or receive, here

std\_msgs.msg.String

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node

from std_msgs.msg import String
```

[https://github.com/LCAS/teaching/blob/lcas\\_humble/cmp3103m\\_ros2\\_code\\_fragments/cmp3103m\\_ros2\\_code\\_fragments/chat\\_sender.py](https://github.com/LCAS/teaching/blob/lcas_humble/cmp3103m_ros2_code_fragments/cmp3103m_ros2_code_fragments/chat_sender.py)

# Writing a Publisher

## Class “Chatter”

- Subclass of Node
- \_\_init\_\_ “Constructur” initialises the object on creation, sets up any member variables
- Create a ROS2 Publisher with an *interface* and the name of the *topic* it will publish to
- Also create a *timer* object to call a function repeatedly (better than a `while` loop)

```
class Chatter(Node):  
    """ a simple "chatter" that publishes String messages on a topic.  
  
    Once this is running, you can use "ros2 topic echo /msg" to see the messages published on the topic.  
    """  
  
    def __init__(self):  
        """ Initialise the Node. ---  
        # calling the constructor of the super class with the name of the node  
        super().__init__('chatter')  
  
        # creating a ROS2 Publisher, for type "String" and topic name "/msg"  
        # the third argument is the length of the queue, i.e., only the last message is queued here  
        self.publisher = self.create_publisher(String, '/msg', 10)  
        timer_period = 1.0 # seconds  
  
        # Creating a timer that will trigger the "run_step" callback every second (event-driven programming)  
        self.timer = self.create_timer(timer_period, self.run_step)  
  
        # let's create a counter object in this Node, to show how to work with member variables in Python  
        self.counter = 0
```

# Writing a Publisher

## run\_step method

- Called by timer at a fixed rate
- Creates the data\_object of the type matching the interface of the Publisher
- Publish the data object

```
def run_step(self):  
    """ the main function that is run by a timer frequently """  
    # First create a ROS2 String object. See 'ros2 interface show std_msgs/msg/String' to see  
    # the interface / data type definition  
    data_object = String()  
  
    # put some data into the create object  
    data_object.data = "Hi! counter=%d" % self.counter  
  
    # increase the counter by 1  
    self.counter += 1  
  
    print("I'm going to publish %s" % data_object)  
  
    # now we are ready to publish the data  
    self.publisher.publish(data_object)
```

[https://github.com/LCAS/teaching/blob/lcas\\_humble/cmp3103m\\_ros2\\_code\\_fragments/cmp3103m\\_ros2\\_code\\_fragments/chat\\_sender.py](https://github.com/LCAS/teaching/blob/lcas_humble/cmp3103m_ros2_code_fragments/cmp3103m_ros2_code_fragments/chat_sender.py)

# Writing a Publisher

“Glue” code / entry point

- Initialise ROS2 framework
- Create object of the specific Node implementation we created
- Run (spin) the node until it is interrupted

```
def main(args=None):  
    # always run "init()" first  
    rclpy.init()  
  
    # let's catch some exceptions should they happen  
    try:  
        # create the Chatter object  
        node = Chatter()  
  
        # tell ROS to run this node until stopped (by [ctrl-c])  
        rclpy.spin(node)  
  
        # once stopped, tidy up  
        node.destroy_node()  
        rclpy.shutdown()  
  
    except KeyboardInterrupt:  
        print('Node interrupted')  
  
    finally:  
        # always print when the node has terminated  
        print("Node terminated")  
  
if __name__ == '__main__':  
    main()
```

# Writing a Subscriber

Similar as before,  
but...

- ... create a *Subscriber* in the Node initiliaser
- ... use a *callback* triggered when new data is received

```
def __init__(self):  
    """ Initialise the Node. """  
    # calling the constructor of the super class with the name of the node  
    super().__init__('ChatReceiver')  
  
    # creating a ROS2 Subscriber, for type "String" and topic name "/msgs"  
    # the forth argument is the length of the queue, i.e., only the last message is queued here  
    self.create_subscription(String, '/msgs', self.callback, 1)  
  
def callback(self, msg):  
    """ the main callback, triggered when a message is received.  
  
        The 'msg' field contains the actual ROS2 message object received.  
        ...  
  
        # simply print the received message on the screen:  
    print("I received this message: %s" % msg)
```

[https://github.com/LCAS/teaching/blob/lcas\\_humble/cmp3103m\\_ros2\\_code\\_fragments/cmp3103m\\_ros2\\_code\\_fragments/chat\\_receiver.py](https://github.com/LCAS/teaching/blob/lcas_humble/cmp3103m_ros2_code_fragments/cmp3103m_ros2_code_fragments/chat_receiver.py)



Thank you for listening!  
Any questions ?

# Thank you for listening!

## Any questions ?

When survey is active, respond at [pollev.com/mhanheide](http://pollev.com/mhanheide)

**End of Lecture Feedback**

0 done

Start your presentation to earn free credits. For a weekly status summary, check this slide tomorrow. Get help at [pollev.com/app](http://pollev.com/app)

<https://attendance.lincoln.ac.uk/>



Access Code: 581193



UNIVERSITY OF  
LINCOLN

## CMP3103 – AUTONOMOUS MOBILE ROBOTS

*Lincoln Centre for Autonomous Systems  
School of Computer Science*



# Syllabus

- Introduction to Robotics
- Robot Programming
- **Robot Vision**
- Robot Control
- Robot Behaviours
- Control Architectures
- Navigation Strategies
- Map Building

**Let's make a colour  
chasing robot!**



## Which middleware paradigm do "ROS topics" implement?

0

data pull

0%

client-server

0%

synchronous processing

0%

publish-subscribe

0%

broadcast

0%

## What is the order of processing in this code?

1

The node is initialised

1st

A new publisher of type Twist is created

2nd

The laser scanner completes a scan and publishes the data

3rd

The callback is called

4th

A Twist message is published

5th

The Robot turns

6th

SEE MORE

## What is wrong with this code?

0

```
1 import rclpy
2
3 from rclpy.node import Node
4
5 from sensor_msgs.msg import LaserScan
6 from std_msgs.msg import String
7
8
9 class FirstSub(Node):
10     def __init__(self):
11         super().__init__('firstsub')
12
13         self.sub = self.create_subscription(
14             Odometry, "/scan",
15             self.callback, 1)
16
17         self.pub = self.create_publisher(String, '/warning', 1)
18
19     def callback(self, data):
20         for range in data.ranges:
21             if range < 1.0:
22                 print("ALERT")
23                 str = String()
24                 str.data = "ALERT"
25                 self.pub.publish(str)
26
27     def main(args=None):
28         rclpy.init()
29         node = FirstSub()
30         rclpy.spin(node)
31         node.destroy_node()
32         rclpy.shutdown()
33
34 if __name__ == '__main__':
35     main()
```

(A) Publisher lacks topic name to publish to

0%

(B) Syntax error in for loop

0%

(C) Wrong topic type

0%

(D) Incorrect import statement

0%

(E) Callback has the wrong number of arguments

0%

(F) Publish called with the wrong type of data.

0%

# What is wrong with this code?

```
 1 import rclpy
 2
 3 from rclpy.node import Node
 4
 5 from sensor_msgs.msg import LaserScan
 6 from std_msgs.msg import String
 7
 8 class FirstSub(Node):
 9     def __init__(self):
10         super().__init__('firstsub')
11
12         self.sub = self.create_subscription(
13             Odometry, "/scan",
14             self.callback, 1)
15
16         self.pub = self.create_publisher(String, '/warning', 1)
17
18     def callback(self, data):
19         for range in data.ranges:
20             if range < 1.0:
21                 print("ALERT")
22                 str = String()
23                 str.data = "ALERT"
24                 self.pub.publish(str)
25
26     def main(args=None):
27         rclpy.init()
28         node = FirstSub()
29         rclpy.spin(node)
30         node.destroy_node()
31         rclpy.shutdown()
32
33     if __name__ == '__main__':
34         main()
```

```
 1 import rclpy
 2
 3 from rclpy.node import Node
 4
 5 from sensor_msgs.msg import LaserScan
 6 from std_msgs.msg import String
 7
 8 class FirstSub(Node):
 9     def __init__(self):
10         super().__init__('firstsub')
11
12         self.sub = self.create_subscription(
13             LaserScan, "/scan",
14             self.callback, 1)
15
16         self.pub = self.create_publisher(String, '/warning', 1)
17
18     def callback(self, data):
19         for range in data.ranges:
20             if range < 1.0:
21                 print("ALERT")
22                 str = String()
23                 str.data = "ALERT"
24                 self.pub.publish(str)
25
26     def main(args=None):
27         rclpy.init()
28         node = FirstSub()
29         rclpy.spin(node)
30         node.destroy_node()
31         rclpy.shutdown()
32
33     if __name__ == '__main__':
34         main()
```



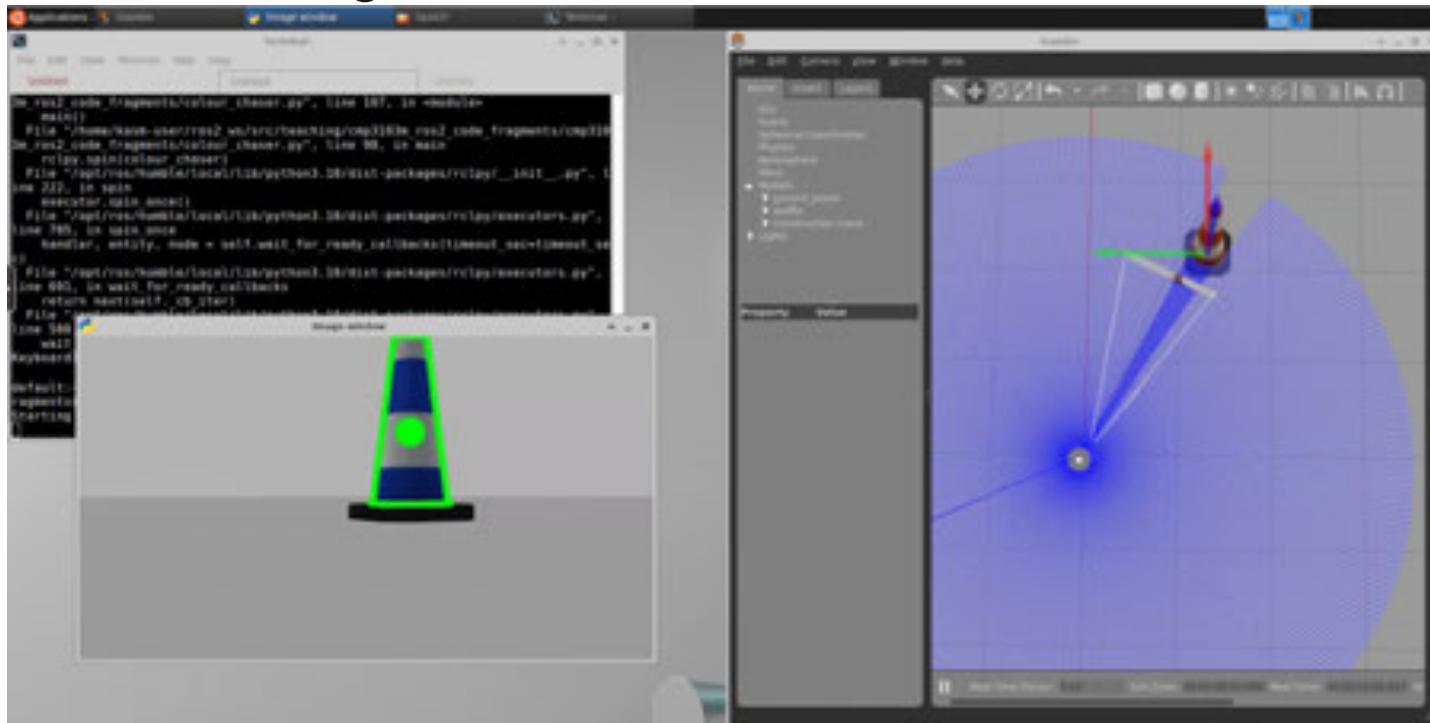
# Vision in ROS

```
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/camera/rgb/image_raw/compressedDepth
/camera/rgb/image_raw/compressedDepth/parameter_descriptions
/camera/rgb/image_raw/compressedDepth/parameter_updates
/camera/rgb/image_raw/theora
/camera/rgb/image_raw/theora/parameter_descriptions
/camera/rgb/image_raw/theora/parameter_updates
```



# Aim for Today

Make a colour chasing robot.



# Robot Vision

## Steps

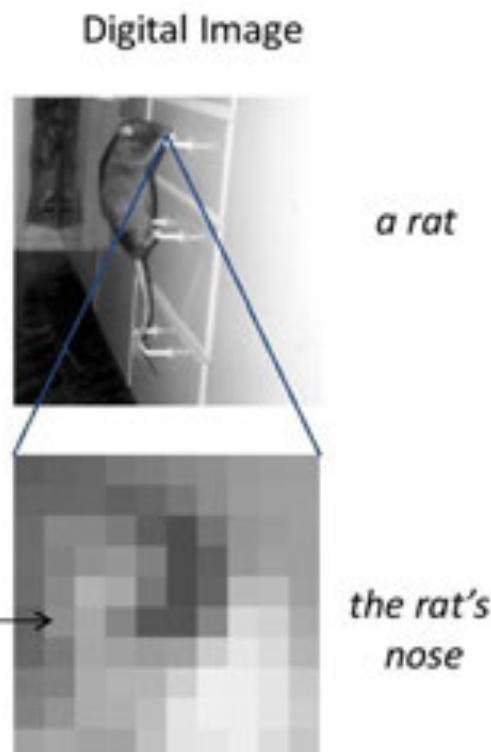
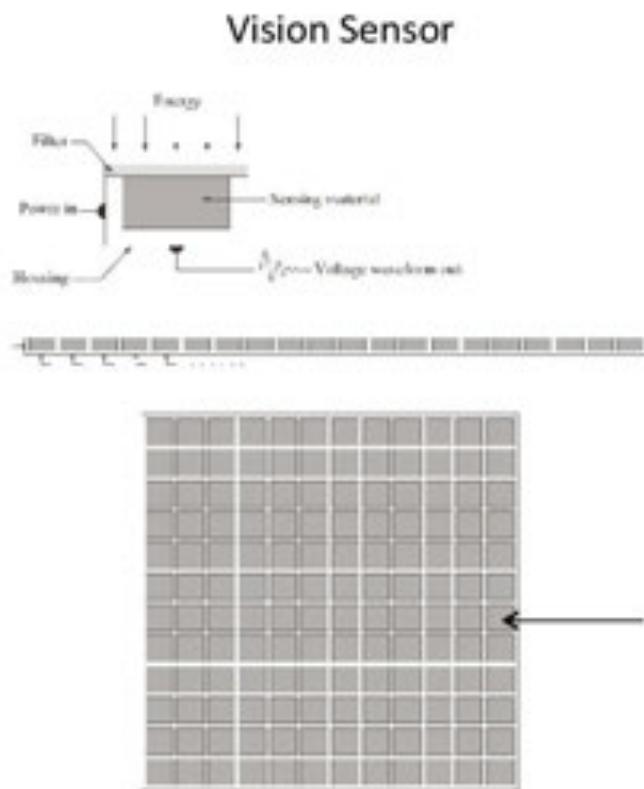
- acquisition
- pre-processing
- segmentation
- feature extraction
- pattern recognition
- **Robot control**

## Requirements for algorithms

- Fast (real-time)
- Robust (to noise and changes in environment)
- Non-stationary assumption (limited use of background subtraction methods)



# Vision Sensor





# OpenCV

- Initially launched by Intel in 1999
- Developed into the “gold standard” in computer vision processing
- Cross-platform, multi-language
- Applications:
  - 2D and 3D feature toolkits
  - Egomotion estimation
  - Facial recognition system
  - Gesture recognition
  - Human–computer interaction (HCI)
- **Mobile robotics**
- Motion understanding
- Object identification
- **Segmentation and recognition**
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- **Motion tracking**
- Augmented reality



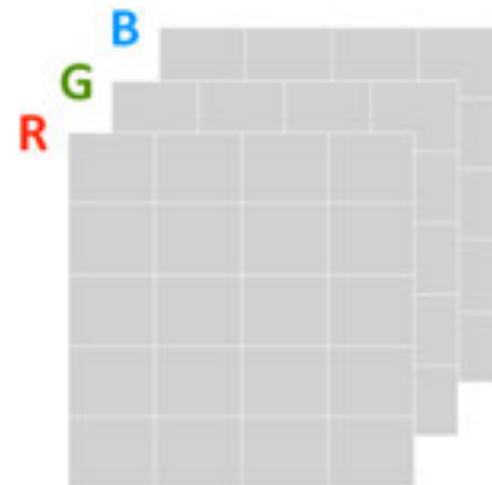
# Representation of Images in OpenCV

Matrices like in Matlab!

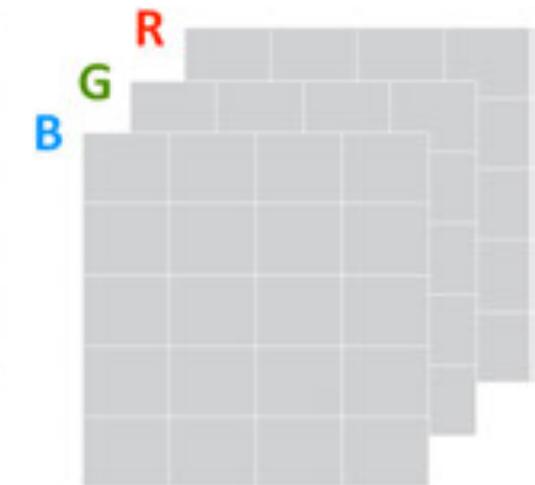
Based on numpy

Not RGB, but BGR!?

Standard



OpenCV



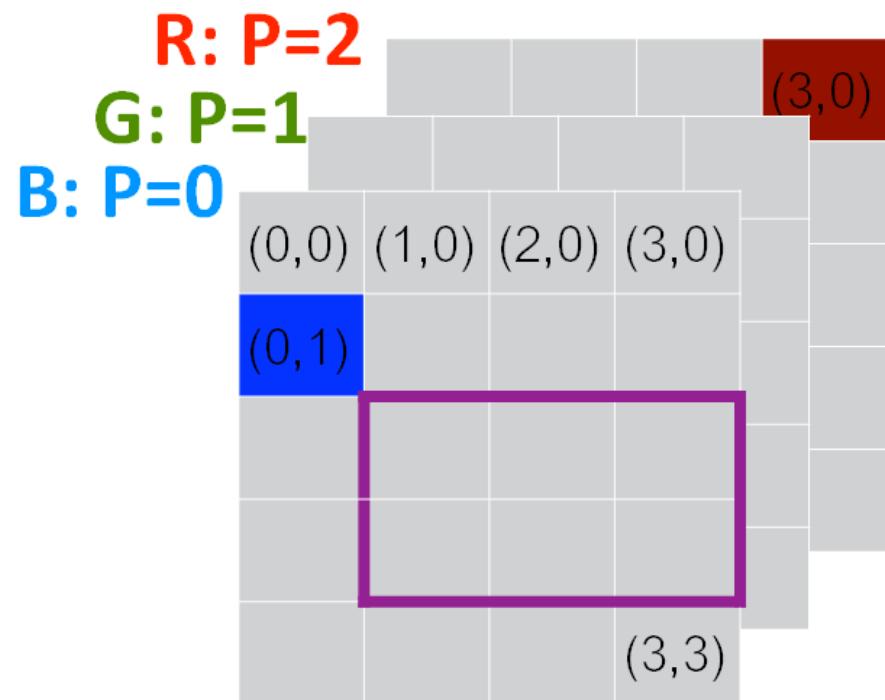
# Basic operations in OpenCV

Pixel access:

- `img[x, y, p]`
- `img[3, 0, 2]`
- `img[0,1,0]`

Ranges:

- `img[1:3, 2:3, 0]`



# A simple OpenCV example (opencv\_intro.py)

Read images

Manipulation pixels

Iterate over pixels

Show images

```
1 import cv2
2 import numpy as np
3
4 # declare windows you want to display
5 cv2.namedWindow("original")
6 cv2.namedWindow("blur")
7 cv2.namedWindow("canny")
8
9 img = cv2.imread('blofeld.jpg')
10 print('type: %s' % type(img))
11 cv2.imshow("original", img)
12
13 # create a new blurred image:
14 blur_img = cv2.blur(img, (7, 7))
15
16 # draw on the image:
17 cv2.circle(blur_img, (100, 100), 10, (255, 0, 255), 5)
18
19 # display the image:
20 cv2.imshow("blur", blur_img)
21
22 # canny is an algorithm for edge detection
23 canny_img = cv2.Canny(img, 10, 200)
24 cv2.imshow("canny", canny_img)
25 print('shape: %s' % str(canny_img.shape))
26
27 # the shape gives you the dimensions
28 h = canny_img.shape[0] # height
29 w = canny_img.shape[1] # width
30
31 # loop over the image, pixel by pixel
32 count = 0
33 # a slow way to iterate over the pixels
34 for y in range(0, h):
35     for x in range(0, w):
36         # threshold the pixel
37         if canny_img[y, x] > 0:
38             count += 1
39 print('count edge pixels: %d' % count)
40
41 # a fast way to iterate using numpy:
42 count = np.sum(canny_img > 0)
43 print('faster count edge pixels: %d' % count)
44
45 cv2.waitKey(0)
46 # good practice to tidy up at the end
47 cv2.destroyAllWindows()
```

# Displaying and drawing in OpenCV

```
import numpy as np
import cv2

# Create a black image
img = np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px
img = cv2.line(img,(0,0),(511,511),(255,0,0),5)
```

Output image = object(input image, (starting point), (end point), (colour), thickness)

```
img = cv2.circle(img,(447,63), 63, (0,0,255), -1)
```

Output image = object(input image, (centre point), radius, (colour), thickness)

# Getting Image Streams from a ROS topic

OpenCV and ROS play nicely

There is a dedicated CvBridge to help you getting and processing image from the robot

- Subscribe on Image topic
- Convert to OpenCV in a callback
- Process the image using OpenCV

[http://wiki.ros.org/cv\\_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython](http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython)

```
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

class OpencvBridge(Node):
    def __init__(self):
        super().__init__('opencv_bridge')

        self.create_subscription(Image, '/camera', self.camera_callback, 10)
        self.br = CvBridge()

    def camera_callback(self, data):
        cv2.namedWindow("Image window")
        cv2.namedWindow("blur")
        cv2.namedWindow("canny")

        cv_image = self.br.imgmsg_to_cv2(data, desired_encoding='bgr8')
```

# opencv\_bridge.py

- Subscribe on Image topic
- Convert to OpenCV in a callback
- Process the image using OpenCV
- Show the images

```
import rclpy
from rclpy.node import Node

from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np

class OpenCVBridge(Node):
    def __init__(self):
        super().__init__('opencv_bridge')
        self.create_subscription(Image, '/camera', self.camera_callback, 10)

        self.br = CvBridge()

    def camera_callback(self, data):
        cv2.namedWindow("Image window")
        cv2.namedWindow("blur")
        cv2.namedWindow("canny")

        cv_image = self.br.imgmsg_to_cv2(data, desired_encoding='bgr8')

        gray_img = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)
        print(np.mean(gray_img))

        blur_img = cv2.blur(gray_img, (3, 3))
        cv2.imshow("blur", blur_img)

        canny_img = cv2.Canny(blur_img, 10, 200)
        cv2.imshow("canny", canny_img)

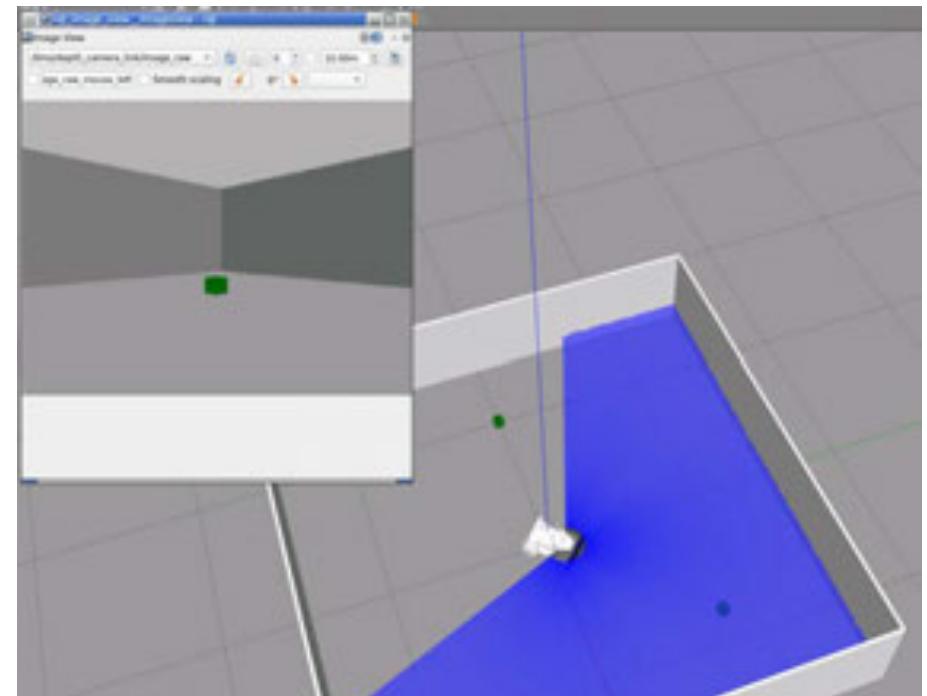
        cv2.imshow("Image window", cv_image)
        cv2.waitKey(1)

def main(args=None):
    rclpy.init(args=args)
    opencv_bridge = OpenCVBridge()
    rclpy.spin(opencv_bridge)
    opencv_bridge.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

# How would you implement a colour-chasing robot?

- Group discussion  
(Prepare a mini presentation on paper, no powerpoint!)
  1. General concept  
(describe without ROS terminology)
  2. How would the code structure look in ROS?
  3. What would be the challenges going from simulation to the real world?



# LEAVE YOUR MARK



**Final year students - shape the future  
of your university by completing the  
National Student Survey 2024.**

**[WWW.THESTUDENTSURVEY.COM](http://WWW.THESTUDENTSURVEY.COM)**



Cyngor Cylchol Addysg  
Llyw Cymru  
Higher Education Funding  
Council for Wales

hefcw



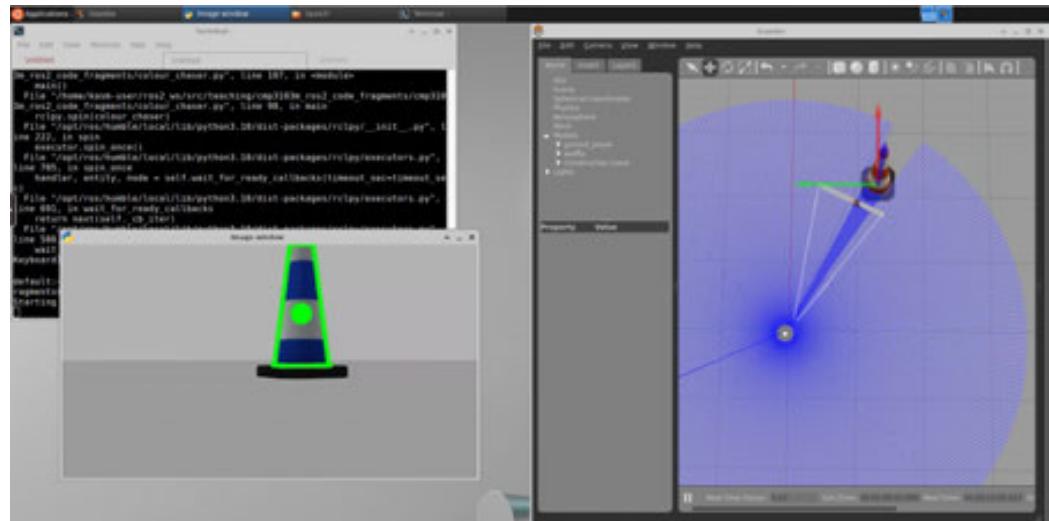
Visit [www.thestudentsurvey.com](http://www.thestudentsurvey.com)  
or scan the QR code below to  
complete the National Student Survey 2024 now!

Leave your  
mark and  
shape the  
future...

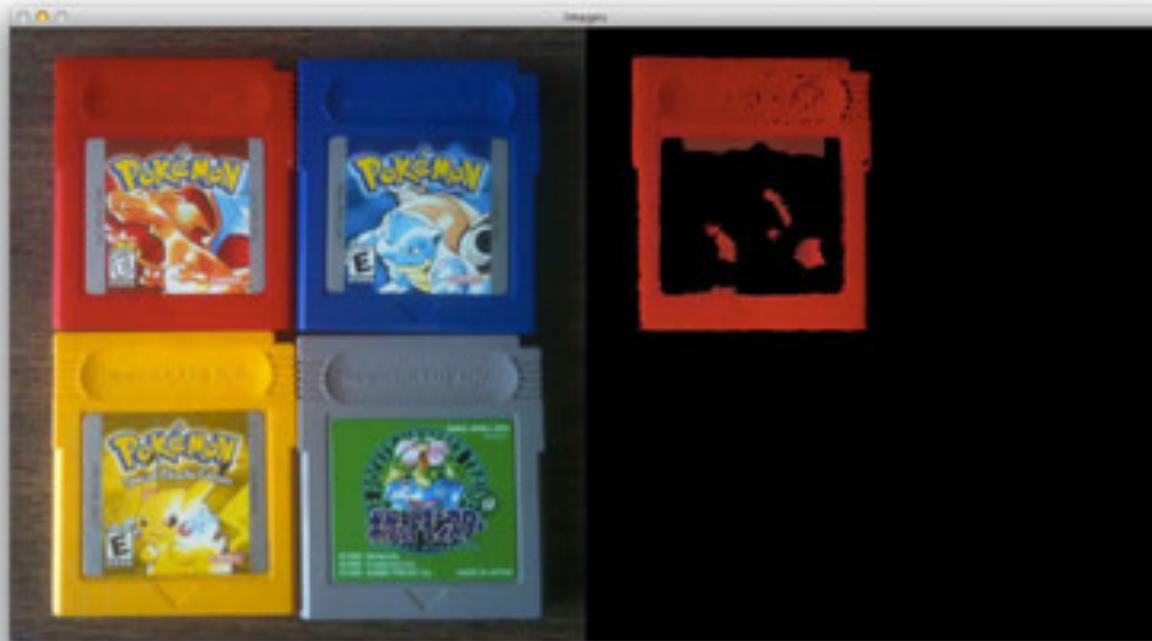


# Colour Chasing Robot

- Subscribe to the Image topic
- Convert to OpenCV in a callback
- Perform colour slicing/masking etc.
- Locating the coloured object
- Moving the robot



# Colour Slicing



<http://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection>

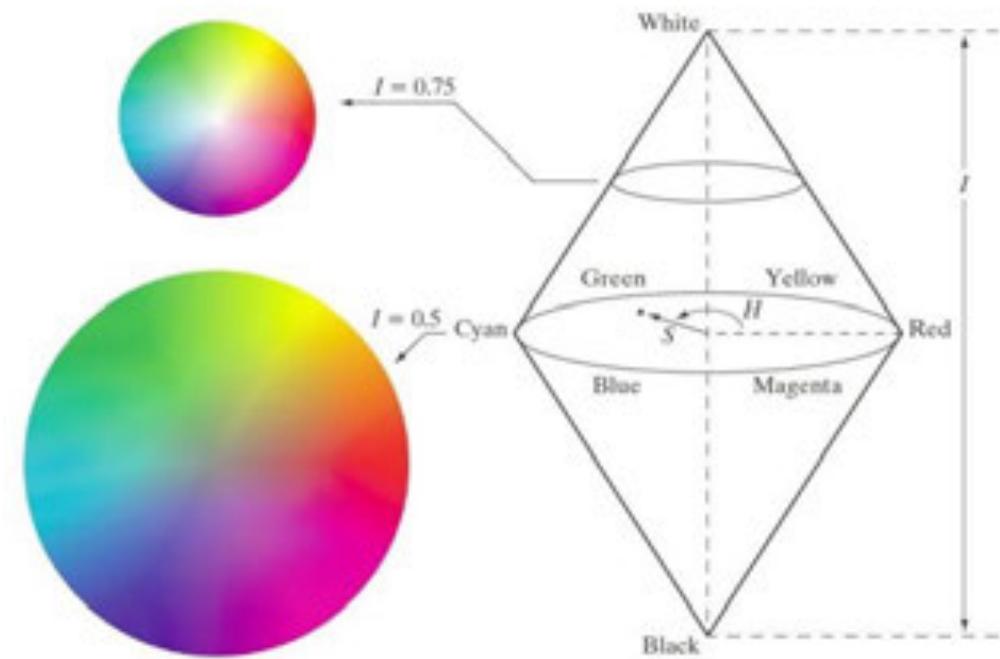
# Colour Models

Common models and their applications

- RGB (red, green, blue)
  - Colour monitors
- CMY, CMYK (cyan, magenta, yellow, black)
  - Colour printers
- HSI/HSV (hue, saturation, intensity/value)
  - Closely related to human perception of colour
  - Decouples hue and intensity – very useful for real word applications where the overall light intensity is often changing.
- Different colour image processing operations are easier or more difficult in different colour spaces

# HSI (Hue, Saturation, Intensity) Model

- Hue is the colour
- Saturation is the greyness
- Intensity is the brightness
- Separates intensity and hue
- Resembles human vision
- Difficult to display (transform to RGB necessary)
- ! Singularities (hue is undefined if the saturation is zero)!



# HSI (Hue, Saturation, Intensity) Model

Hue is expressed as an angle

- $0^\circ/360^\circ$  - Red
  - $120^\circ$  - Green
  - $240^\circ$  - Blue
- ! Take Care to check for the discontinuity  $0^\circ/360^\circ$  around red when processing HIS images!

And be aware if we have 8 bit HSV, then  $255 \sim 360$ !

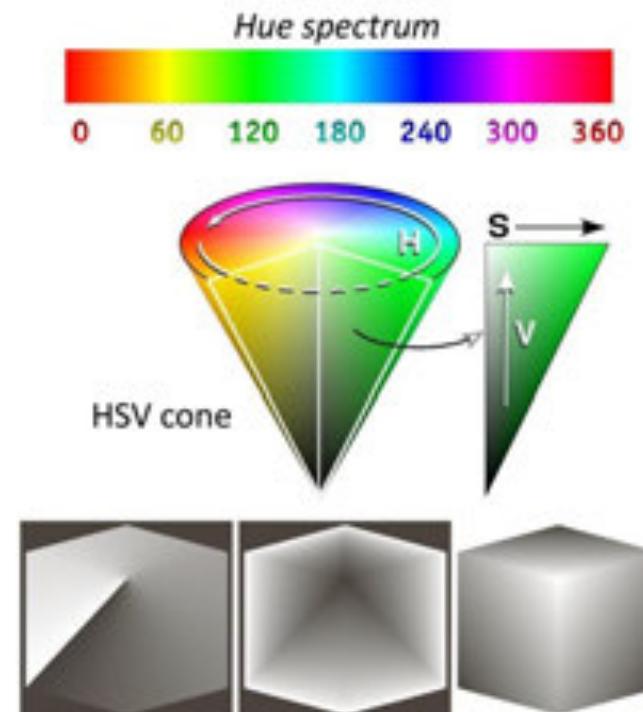


FIGURE 6.15 HSV components of the image in Fig. 6.8. (a) Blue, (b) saturation, and (c) intensity images.

# Colour Models Examples



Full color



Red



Green



Blue



Hue



Saturation

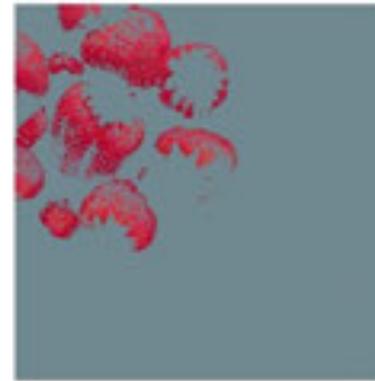


Intensity

Note the  
discontinuity  
(white/black) for  
red strawberries

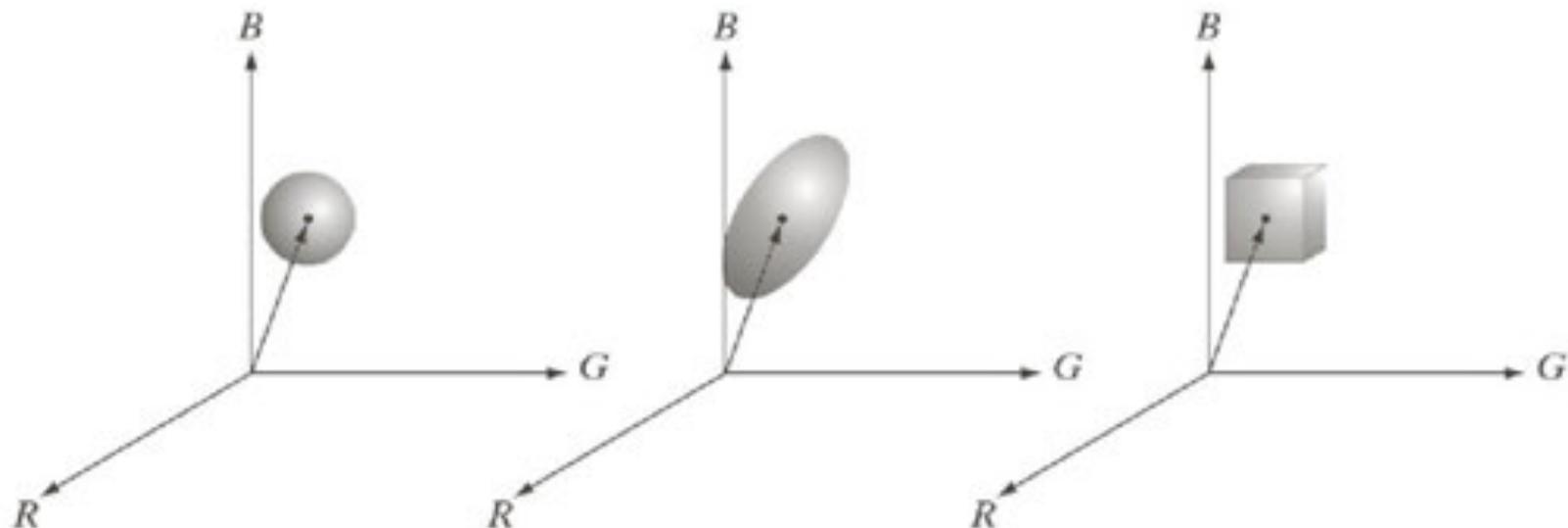
## Colour Slicing

- To highlight a specific range of colours.
- To define a mask for further processing.
- How to define the range of interest?
- Cube/sphere/etc in colour space (centred at a prototypical colour)



# Colour Slicing

Different ways to define the range of interest in RGB colour space



# Colour Slicing in Python

```
cv2.inRange(image,(low_h, low_s, low_i), (high_h, high_s, high_i))
```

- Subscribe on Image topic
- Convert to OpenCV in callback
- Optional: convert to different colour space (from BGR)
- Use inRange for colour slicing
- Output binary image
- Optional: post-process image (morphological operations e.g. erosion, dilation etc.)

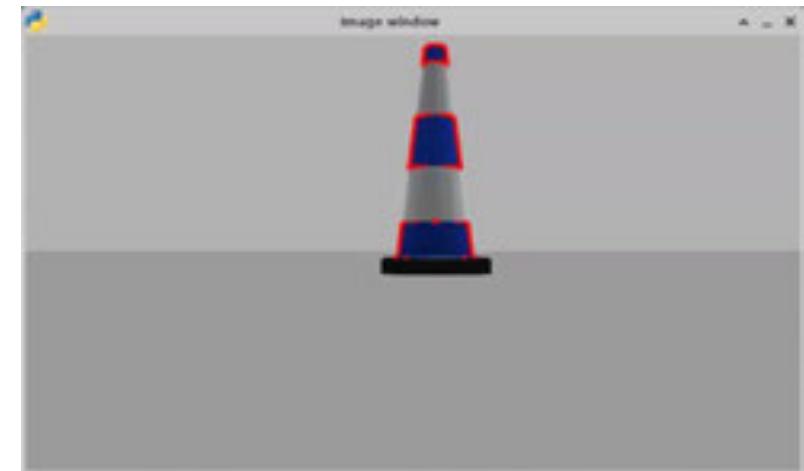
# Colour Slicing in Python

```
cv2.inRange(image,(low_h, low_s, low_i), (high_h, high_s, high_i))
```

- What colour space to slice in?
- What values to choose?
  - Save images with OpenCV `cv2.imwrite(filename, image)`
  - Save images using ros2 run rqt\_image\_view rqt\_image\_view
  - Use GIMP/Paint and its colour picker
  - Use <http://imagecolorpicker.com> (or others) and upload an image

## Finding Contours

- From colour slicing to regions!
- Contours are a curve that joins all continuous points, having same colour or intensity.
- Useful tool for shape analysis, object detection and recognition.
- For better accuracy use binary images.
- see color\_contours.py



# Colour Chasing Robot (colour\_chaser.py)

- Subscribe to the Image topic
- Convert to OpenCV image
- Convert colour model
- Filter/mask a range of colour values

```
class ColourChaser(Node):  
    def __init__(self):  
        super().__init__('colour_chaser')  
  
        # subscribe to the camera topic  
        self.create_subscription(Image, '/camera/image_raw', self.camera_callback, 10)  
  
        # CV Bridge to convert between ROS and OpenCV images  
        self.br = CvBridge()  
  
    def camera_callback(self, data):  
        self.get_logger().info("camera_callback")  
  
        cv2.namedWindow("Image window", 1)  
  
        # Convert ROS Image message to OpenCV image  
        current_frame = self.br.imgmsg_to_cv2(data, desired_encoding="bgr8")  
  
        # Convert image to HSV  
        current_frame_hsv = cv2.cvtColor(current_frame, cv2.COLOR_BGR2HSV)  
        # Create mask for range of colours (HSV low values, HSV high values)  
        current_frame_mask = cv2.inRange(current_frame_hsv, (70, 0, 50), (150, 255, 255))
```

# Colour Chasing Robot (colour\_chaser.py)

- Find the contours in the colour sliced mask
- Use only the largest contour
- Draw the contours on the image
- If there is a contour find it's centre – draw a circle
- Publish a Twist message to turn the robot towards the object

```
contours, hierarchy = cv2.findContours(current_frame_mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# sort by area (keep only the biggest one)
contours = sorted(contours, key=cv2.contourArea, reverse=True)[1]

# draw contour(s) (image to draw on, contours, colour number -1 to draw all contours, colour, thickness)
current_frame_contours = cv2.drawContours(current_frame, contours, -1, (0, 255, 0), 2)

self.tw.publish() # twist message to publish

if len(contours) > 0:
    M = cv2.moments(contours[0]) # only select the largest contour
    if M['area'] > 0:
        # find the centroid of the contour
        cx = int(M['m00']/M['m00'])
        cy = int(M['m00']/M['m00'])
        print("Centroid of the biggest area: ({}, {})".format(cx, cy))

        # draw a circle centered at centroid coordinates
        # cv2.circle(image, center_coordinates, radius, color, thickness) -1 px will fill the circle
        cv2.circle(current_frame, (round(cx), round(cy)), 50, (0, 255, 0), -1)

        # find height/width of robot camera image from roslaunch echo /camera/image_raw height: 1800 width: 1200
        # if center of object is to the left of image center move right
        if cx < 900:
            self.tw.angular.z=0.1
        # else if center of object is to the right of image center move left
        elif cx > 1200:
            self.tw.angular.z=-0.1
        else: # center of object is in a 100 px range in the center of the image so don't turn
            print("object in the center of image")
            self.tw.angular.z=0.0

        self.pub_cmd_vel.publish(self.tw)
    else:
        print("No Centroid Found")
        # turn until we can see a coloured object
        self.tw.angular.z=0.3

self.pub_cmd_vel.publish(self.tw)

# show the cv images
current_frame_contours_small = cv2.resize(current_frame_contours, (0,0), fx=0.5, fy=0.5) # reduce image size
cv2.imshow("Image window", current_frame_contours_small)
cv2.waitKey(1)
```

# Code on Github

# Some More Robot-related Vision Stuff

# Applications

Visual Path Following (Teach and Repeat), Lagadic project, INRIA, France.

Visual path following  
using only monocular vision  
for urban environments

- Lagadic project -

INRIA Rennes - IRISA

# Applications

Visual Path Following (Teach and Repeat)



# Image Features

## Features

- Compact, meaningful representation of the image

## What are the good features?

- Edges, corners, blobs, colour, texture

## State of the art

- SIFT - Scale-Invariant Feature Transform
- SURF - Speeded Up Robust Features
- Haar-like (rectangular like features)
- HOG - Histogram of Oriented Gradients
- etc. All in OpenCV!

## Application examples

- Object detection: template matching
- Scene reconstruction: extract depth information



# Vision in Robotics

SIGGRAPH Talks 2011

## KinectFusion:

Real-Time Dynamic 3D Surface  
Reconstruction and Interaction

Shahram Izadi 1, Richard Newcombe 2, David Kim 1,3, Otmar Hilliges 1,  
David Molyneaux 1,4, Pushmeet Kohli 1, Jamie Shotton 1,  
Steve Hodges 1, Dustin Freeman 5, Andrew Davison 2, Andrew Fitzgibbon 1

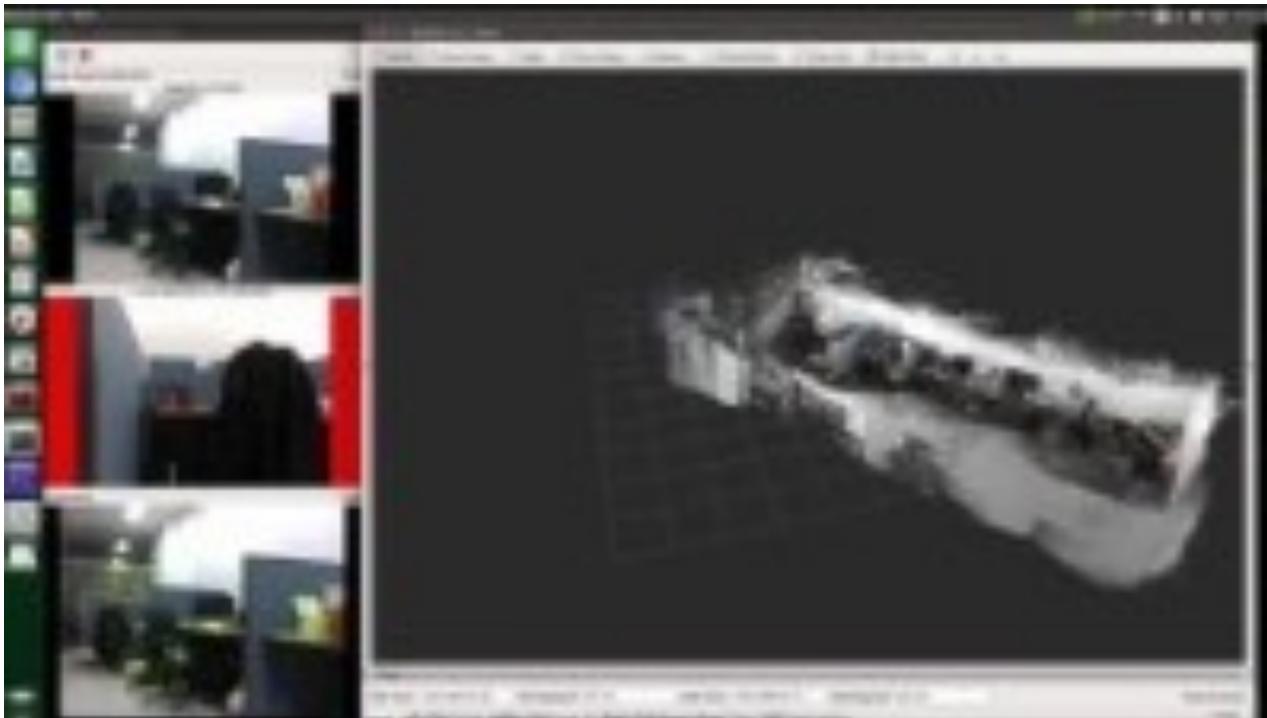
1 Microsoft Research Cambridge 2 Imperial College London

3 Newcastle University 4 Lancaster University

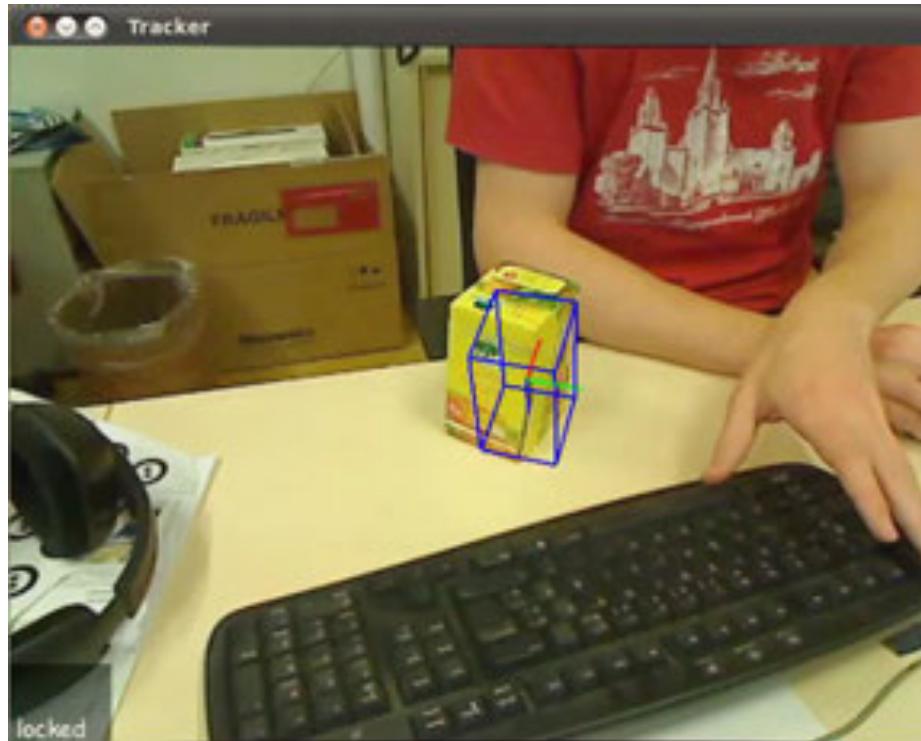
5 University of Toronto

# Applications

## Rtab Map 3D Mapping

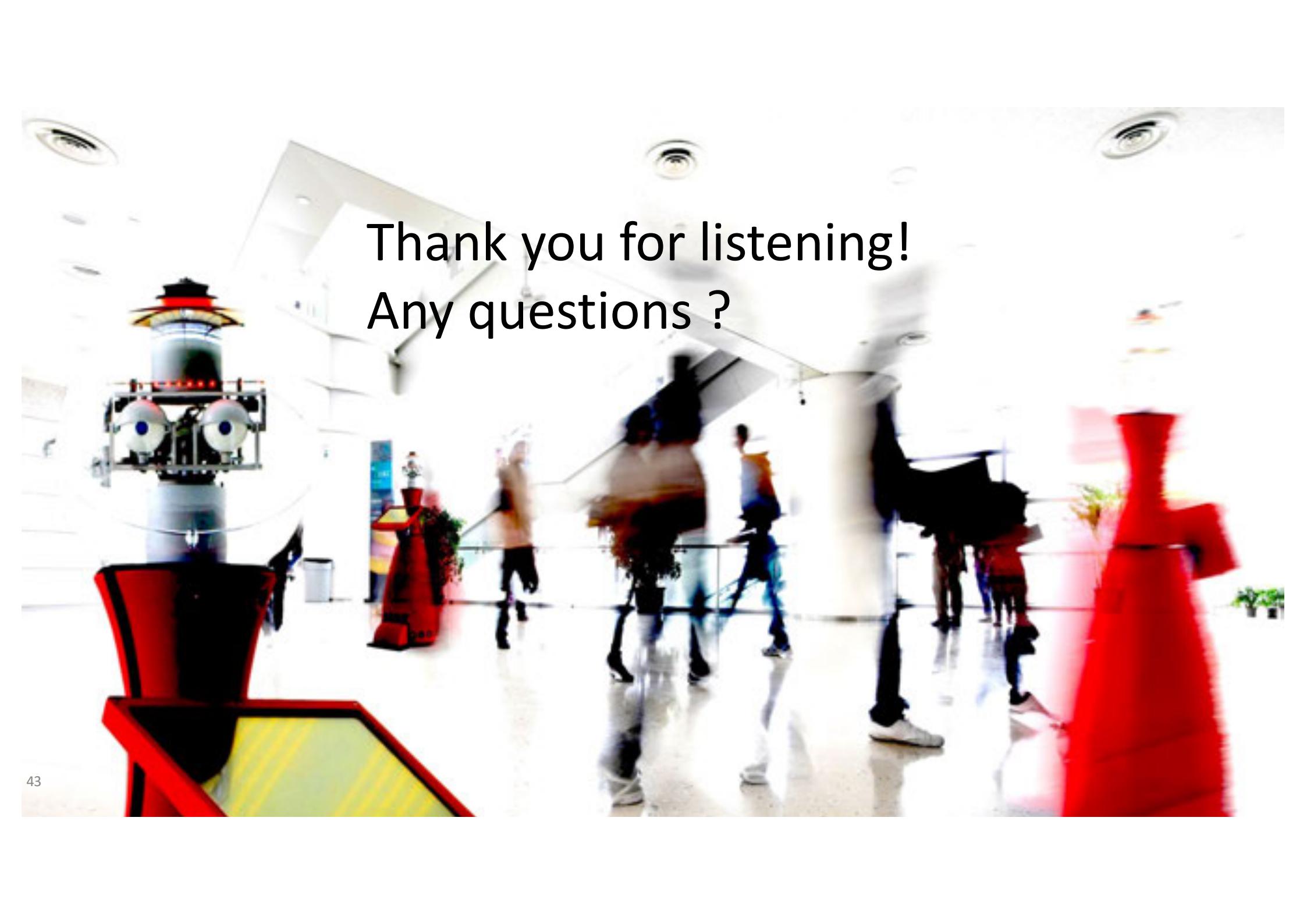


# Object Tracking



# Vision in Robotics





Thank you for listening!  
Any questions ?

# CMP3103M AMR

Prof. Marc Hanheide



UNIVERSITY OF  
LINCOLN





# SOME MORE ON ROS PROGRAMMING



Start the presentation to see live content. Still no live content? Install the app or get help at [PollEv.com/app](https://PollEv.com/app)

# What's wrong with this code?

```
import rospy
from std_msgs.msg import String
from sensor_msgs.msg import LaserScan

class FirstSub:
    def __init__(self):
        self.sub = rospy.Subscriber('/turtlebot_2/scan',
                                    LaserScan,
                                    callback=self.callback)
        self.pub = rospy.Publisher('/out', String)

    def callback(self, msg):
        for range in msg.ranges:
            if range < 1.0:
                s = String()
                s.data = "we are too close!"
                self.pub.publish(s)

fp = FirstSub()
rospy.init_node("first-subscriber")
rospy.spin()
```

node is initialised  
after Subscriber and  
Publisher are created

Illegal constructor for  
Publisher

an import statement  
is missing

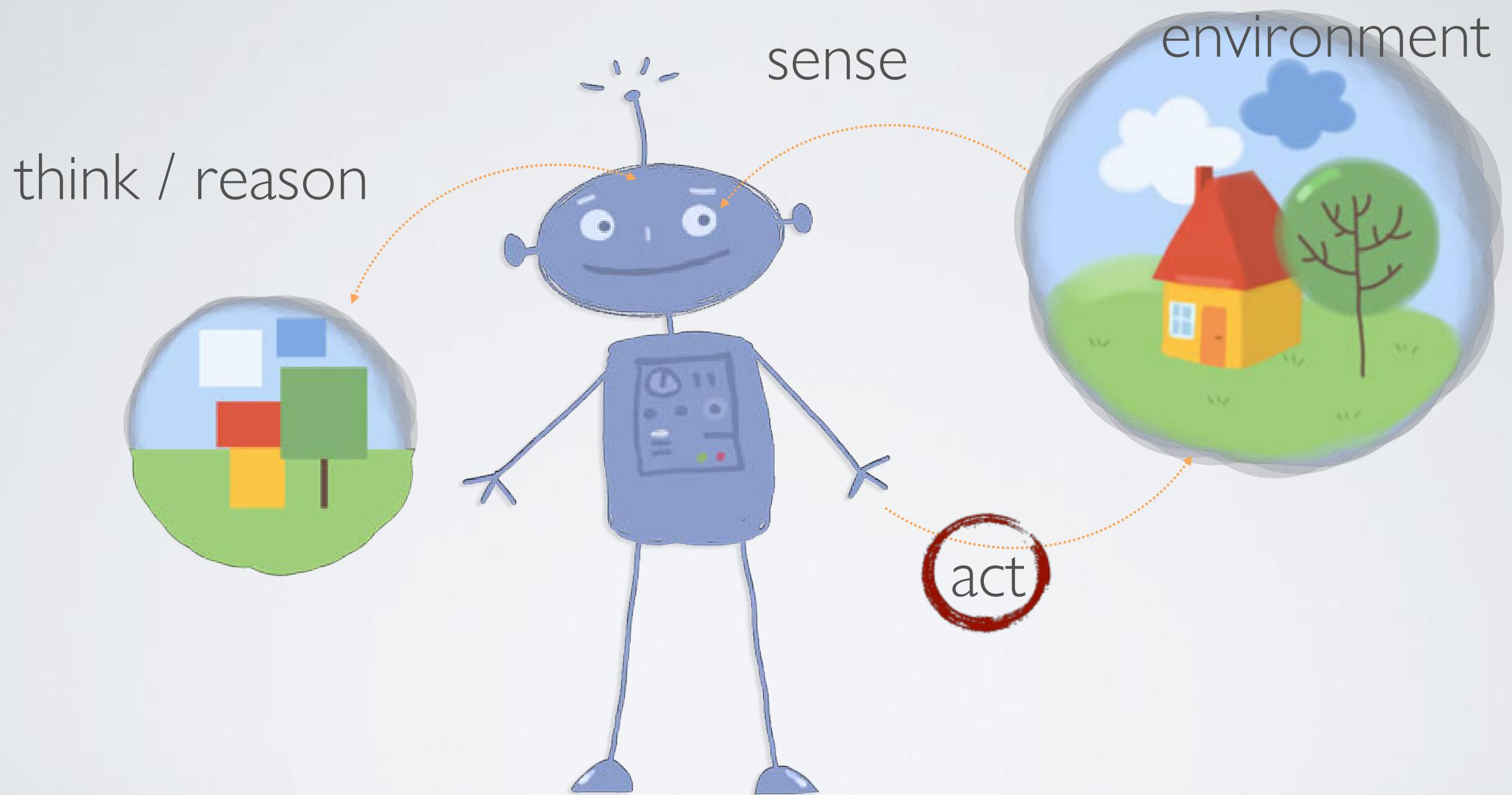
tries publishing  
wrong type

construction of object  
misses argument

# SYLLABUS

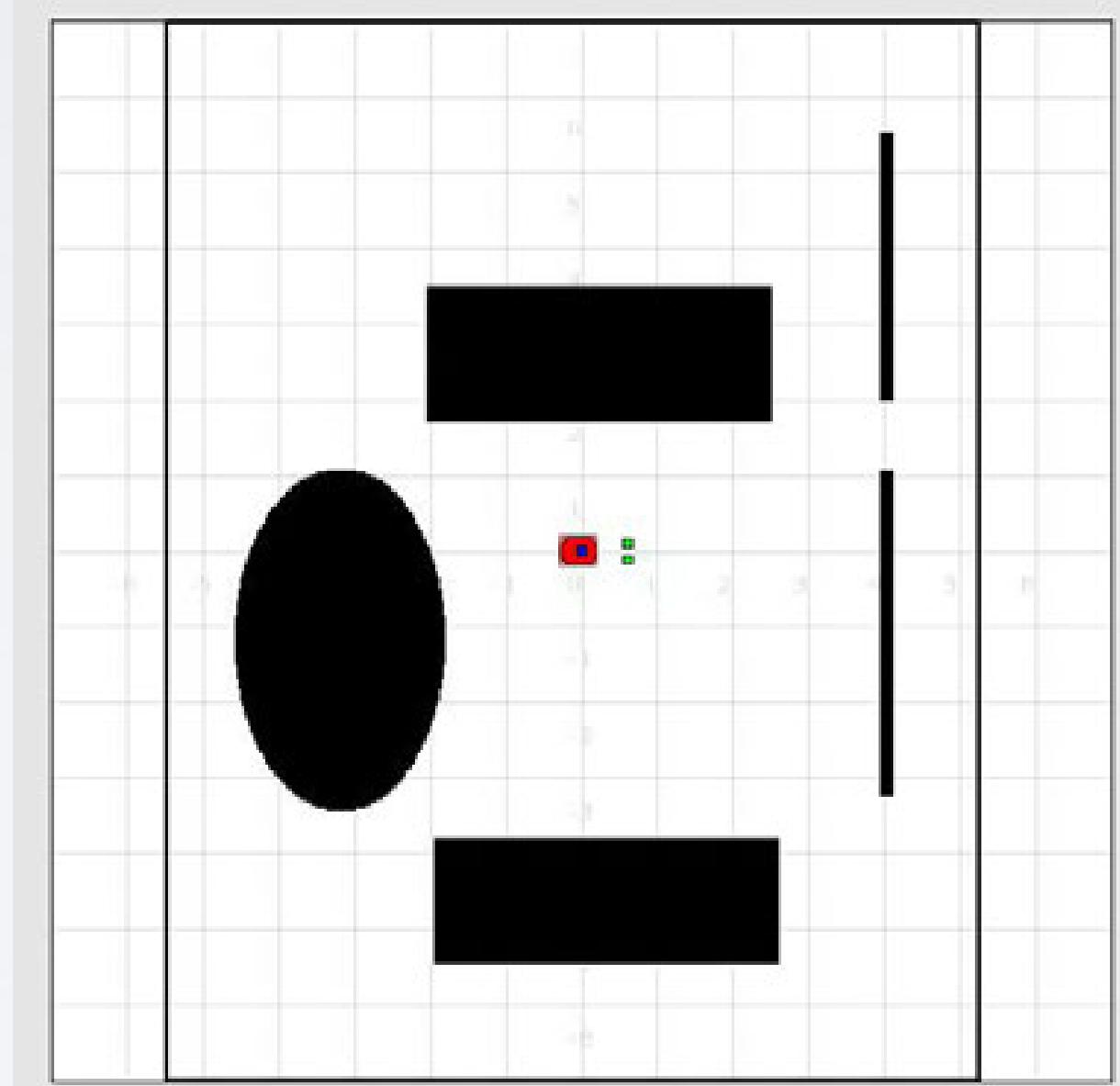
- ▶ Introduction to Robotics
- ▶ Robot Programming
- ▶ Robot Sensing & Vision
- ▶ **Robot Control**
- ▶ Robot Behaviours
- ▶ Control Architectures
- ▶ Navigation Strategies
- ▶ Map Building

# ROBOTIC AGENT



# MOBILE ROBOT CONTROL

- ▶ Move a robot in a desired way
  - ▶ position control
    - ▶ move to XY
    - ▶ follow a path
  - ▶ behaviours
    - ▶ follow the corridor
    - ▶ avoid obstacles
- ▶ Position control
  - ▶ open and closed loop control
- ▶ What Voltage/Current/Force to put on your motors?



# THE TWIST

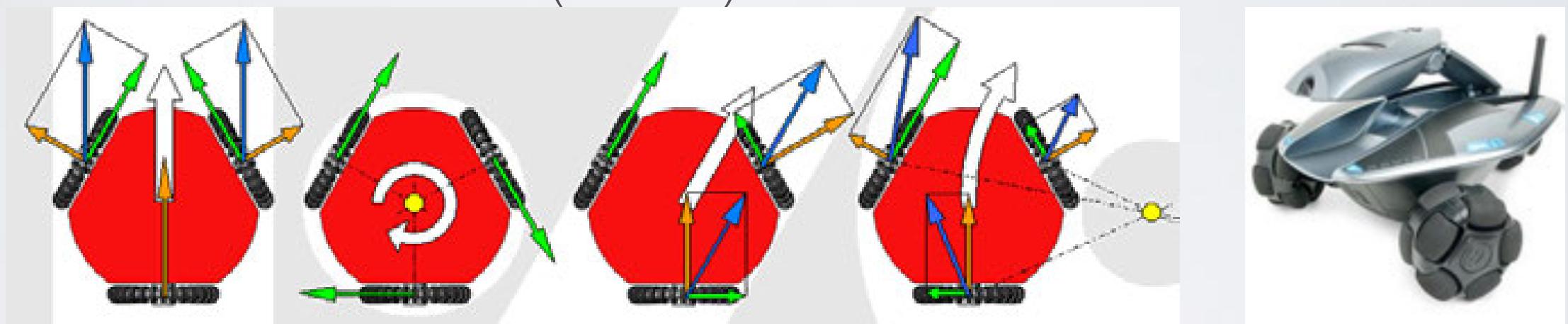
```
pub = rospy.Publisher(  
    '/cmd_vel',  
    Twist,  
    queue_size=1  
)  
t = Twist()  
pub.publish(t)
```

?

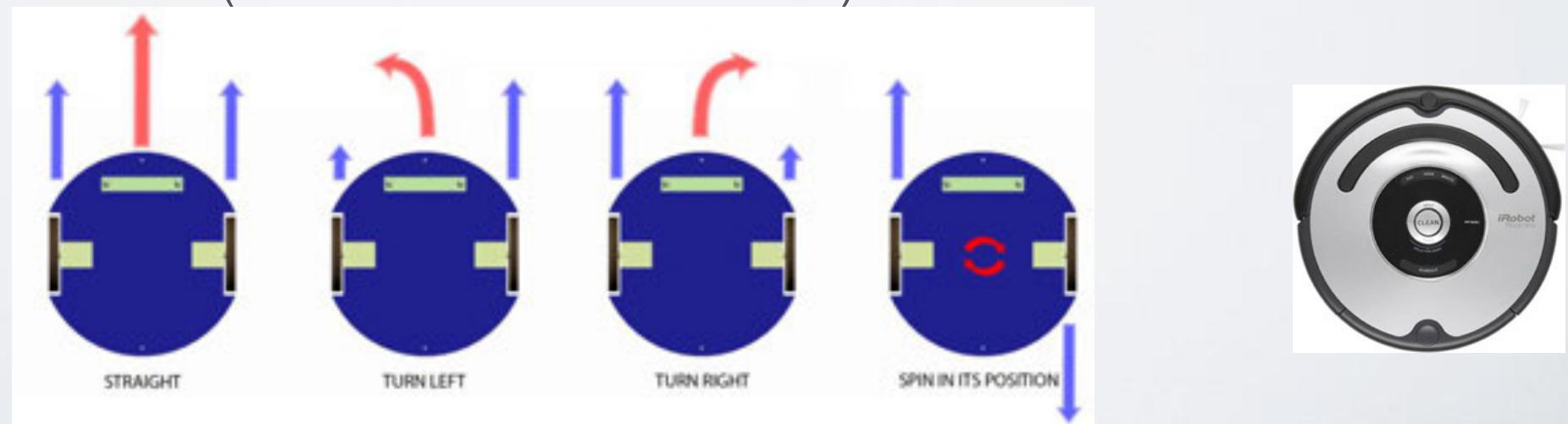


# MOVEMENT COMMANDS

- ▶ Dependent on the wheel configuration
- ▶ Omni-directional Drive (Rovio)



- ▶ Differential Drive (Turtlebot/Roomba)



# CONTROLLING TURTLEBOT

- ▶ Which speeds to turn each wheel to move in desired direction?
- ▶ That corresponds to the Force -> Current for every motor
- ▶ This is a control problem!

# FOR MOTION CONTROL

Model

Algorithm

Kinematics / Dynamics

control-parameter

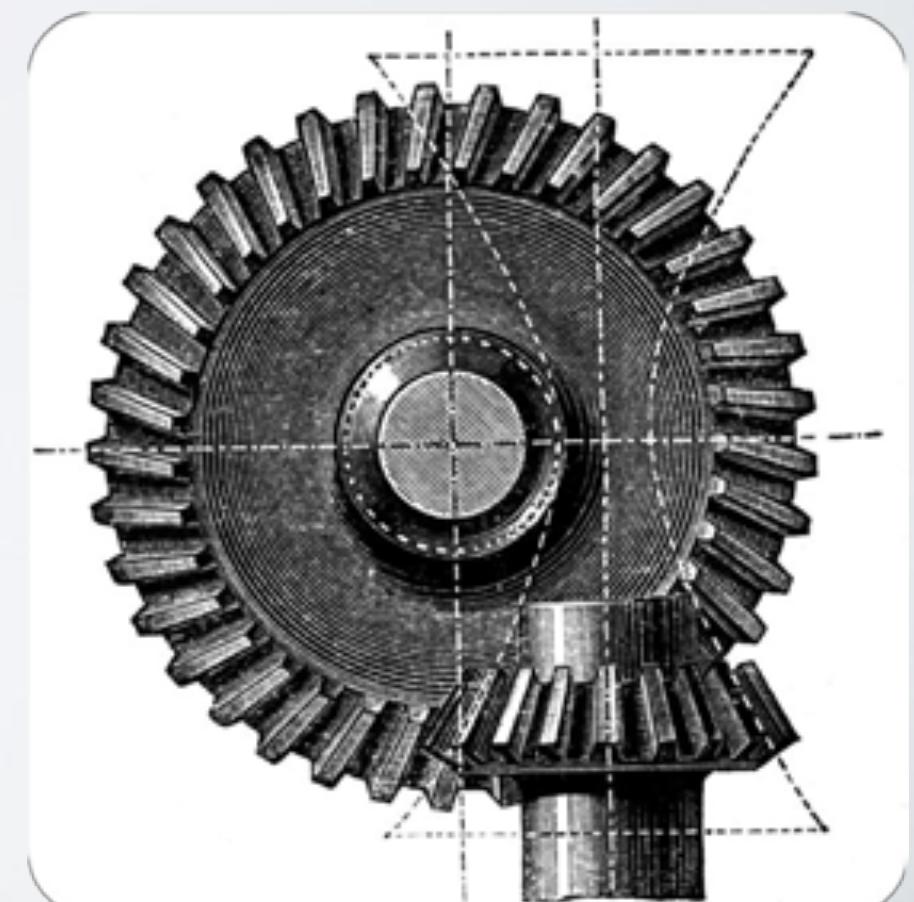
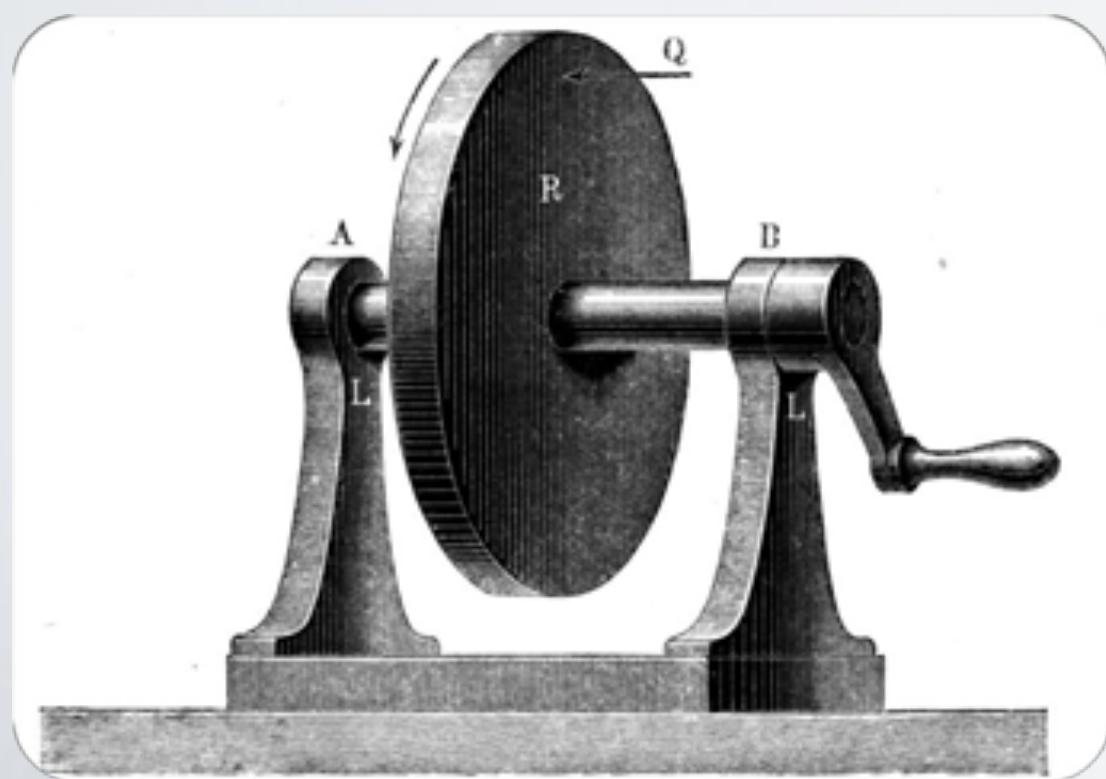
Engineering

control signal => actuators

# KINEMATICS

Model

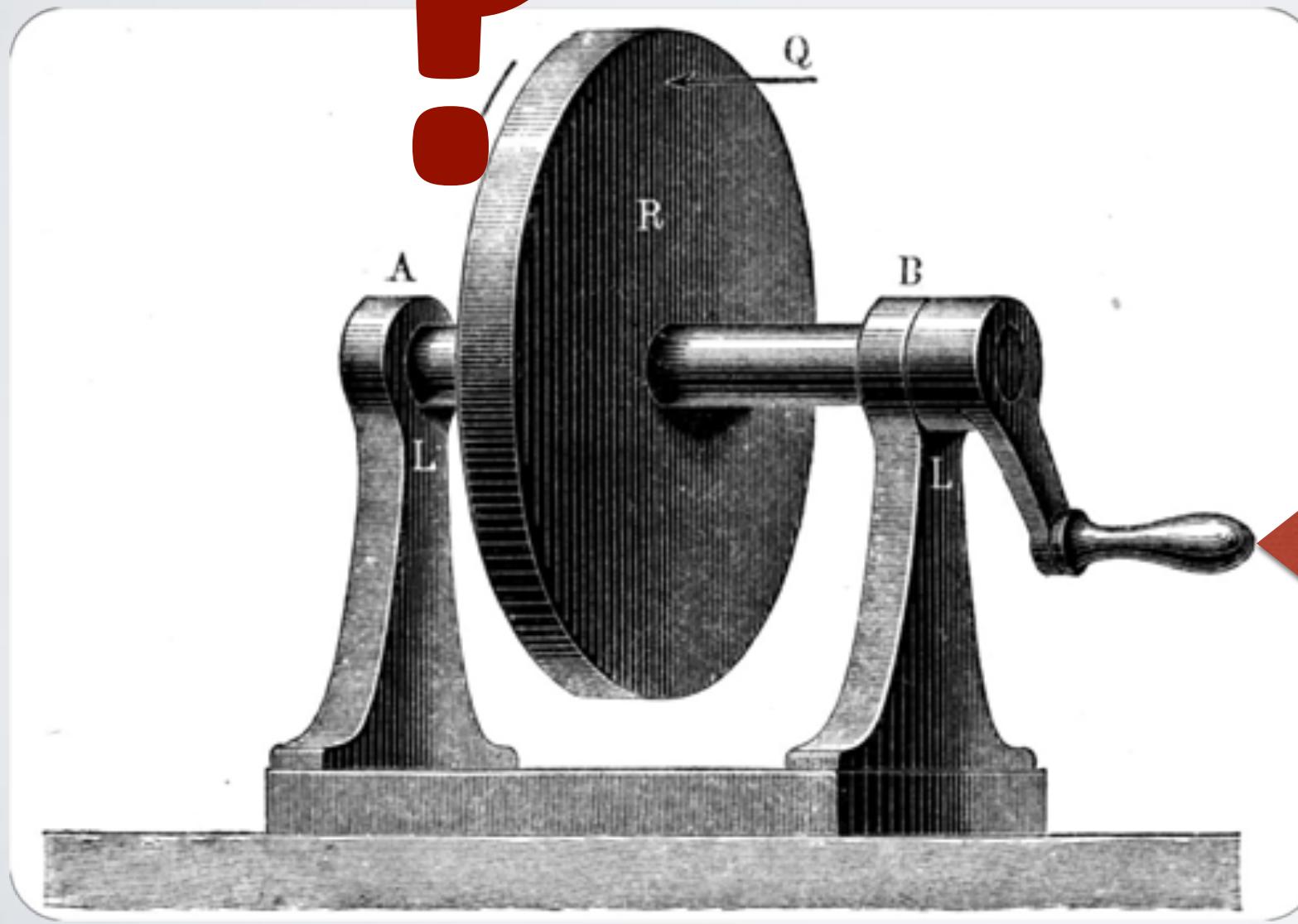
Kinematics / Dynamics



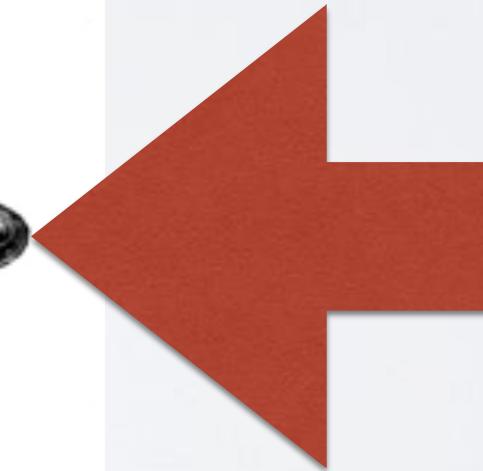
# AD-HOC MODEL

- ▶ Drive in a straight line: calculate the number of steps required to perform a distance unit e.g. 1m
  - ▶ different for each speed value (non(?) -linear dependency)
  - ▶ and other factors (e.g. surface type)
- ▶ Use this value to calculate the number of steps required for a desired distance
- ▶ Similar with pure rotations/angles

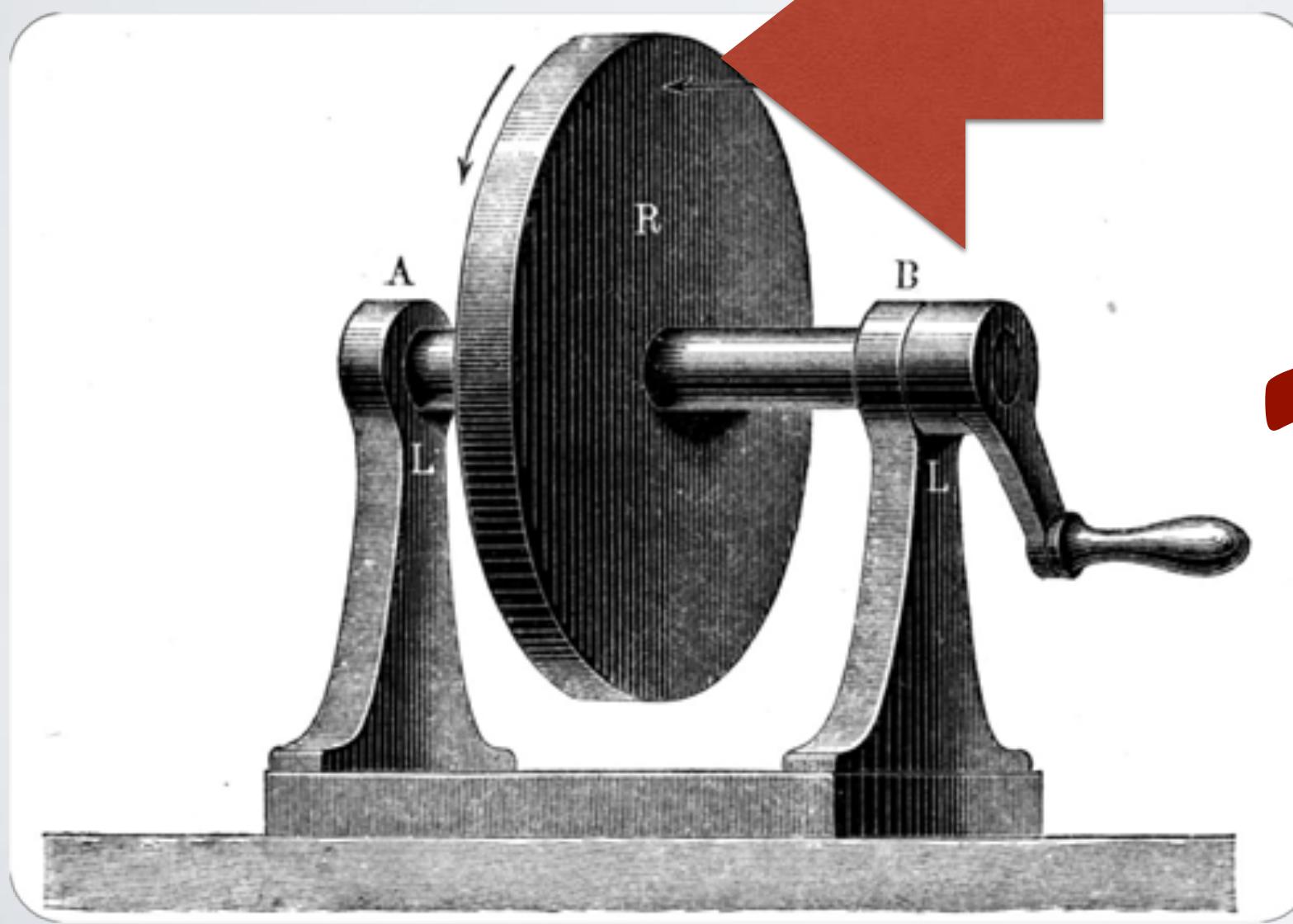
# FORWARD MODEL



?



# INVERSE MODEL



# AD-HOC FORWARD MODEL IS EASY?



# THE WHEEL



$$c = 2\pi r$$

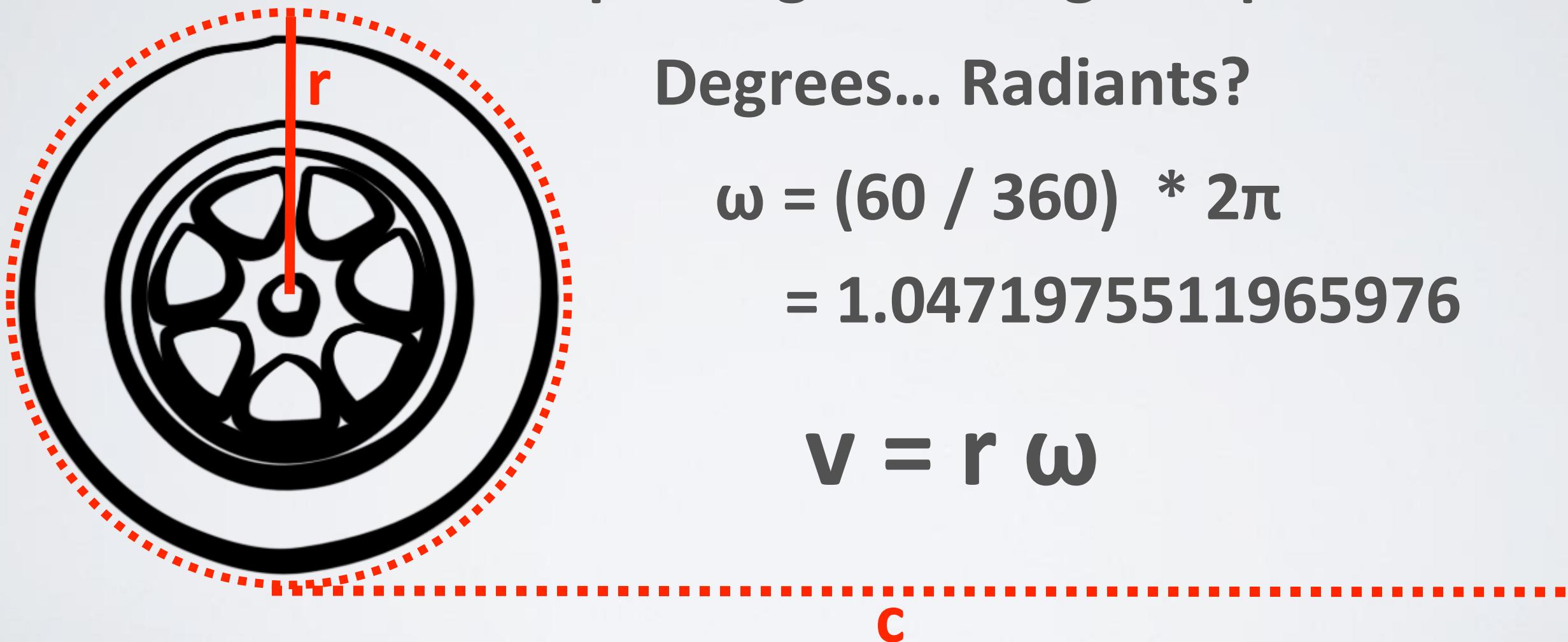
# A SIMPLE ROBOT WITH ONE WHEEL: FORWARD MODEL

Spinning at 60 degrees per second

Degrees... Radians?

$$\begin{aligned}\omega &= (60 / 360) * 2\pi \\ &= 1.0471975511965976\end{aligned}$$

$$v = r \omega$$



# AND WITH TWO WHEELS?

$$v = 1/2 r (\omega_l + \omega_r)$$

but  $\omega_l = \omega_r$  when going straight



# WHAT'S THE INVERSE MODEL?



Make your turtlebot go forward  
with  **$v = 1\text{m/s}$**  with  **$r = 6\text{cm}$**

$$v = r \omega$$

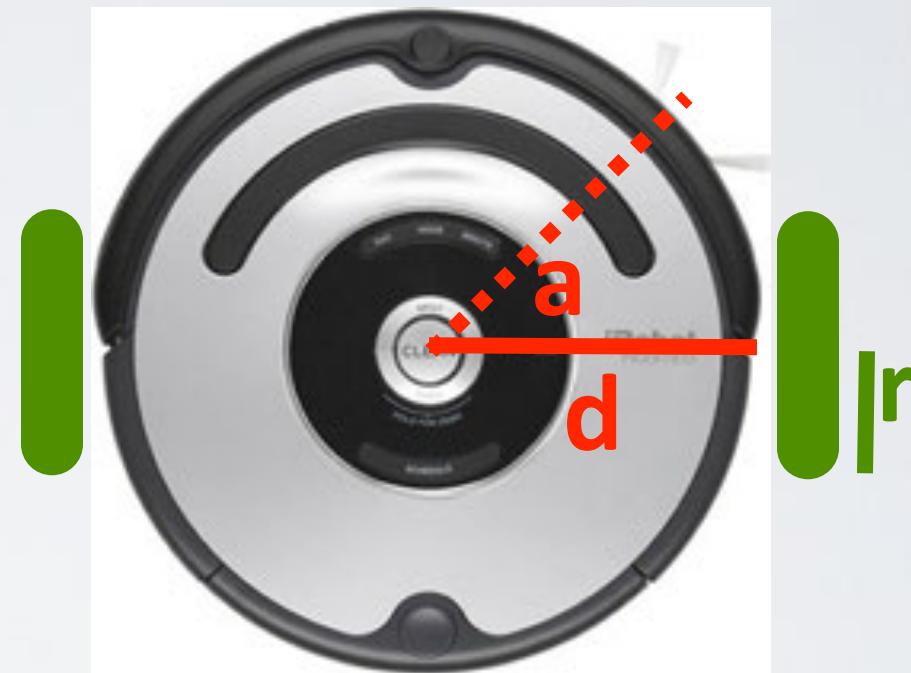
$$\omega = v/r$$

# AD-HOC FORWARD MODEL IS EASY?

$$v = 0$$

$$v = 1/2 r (\omega_l + \omega_r)$$

$$\Rightarrow \omega_l = -\omega_r$$

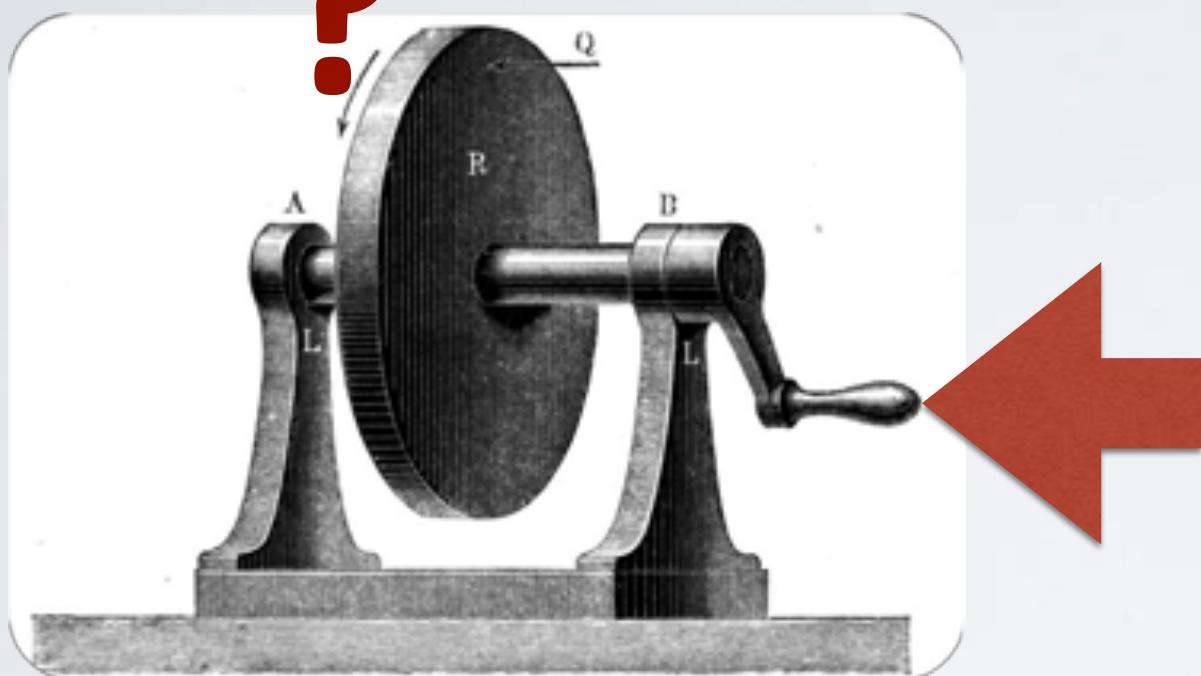


Hint: Assume both wheels moving same speed again!

$$a = r/d (\omega_l - \omega_r)$$

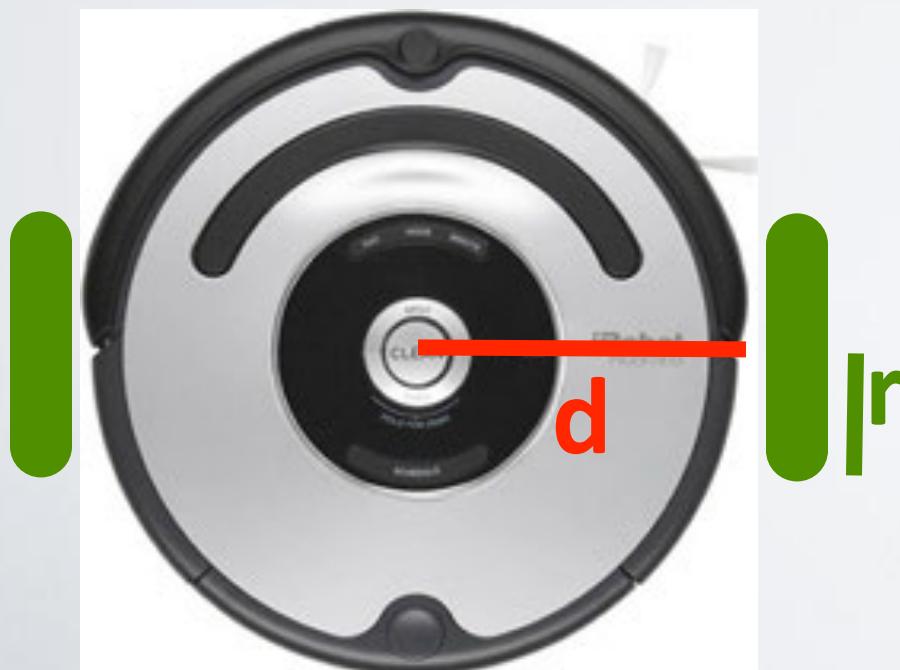
# FORWARD MODEL

?

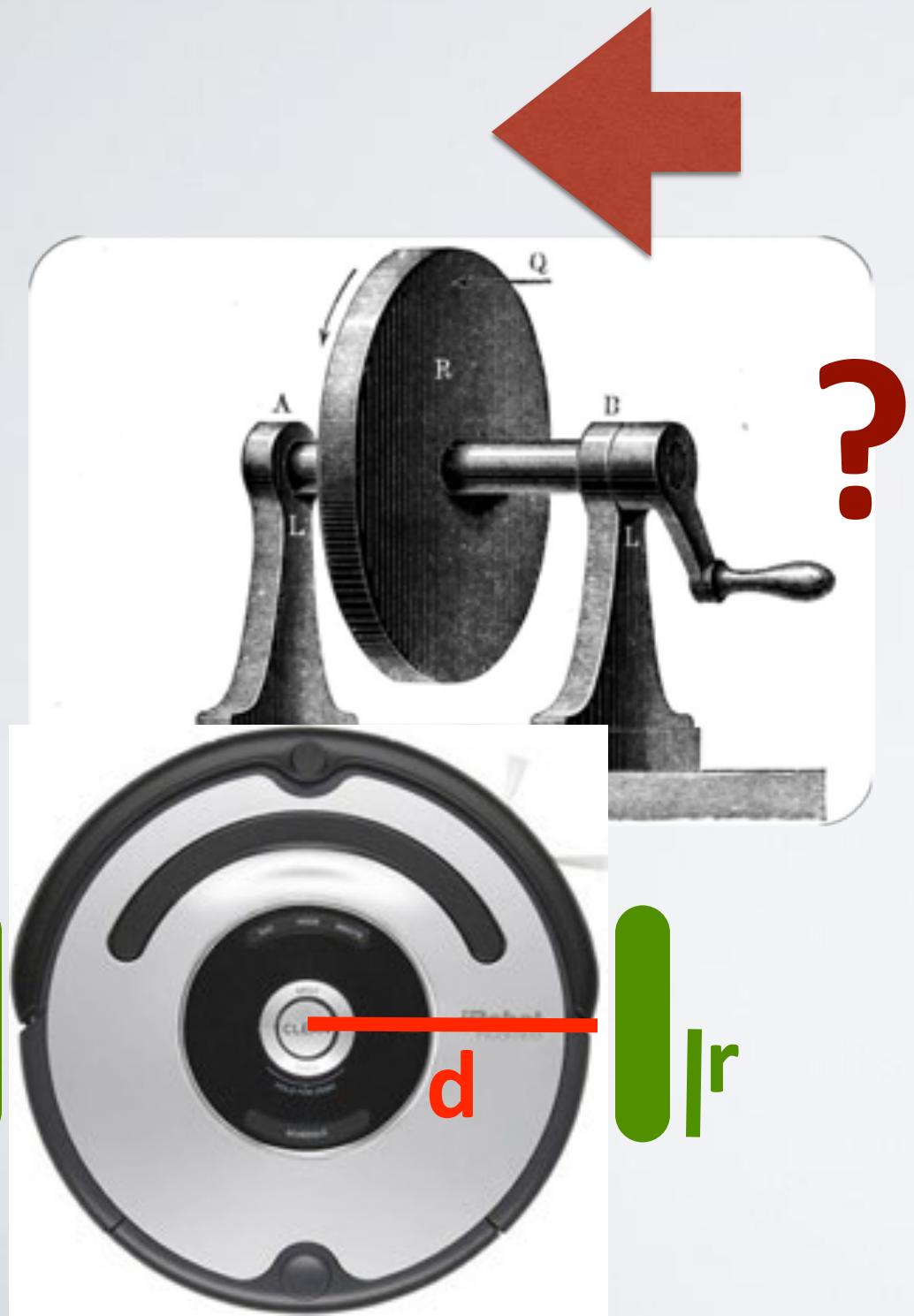


$$v = 1/2 r (\omega_l + \omega_r)$$

$$a = 1/d r (\omega_l - \omega_r)$$



# INVERSE MODEL



$$v = 1/2 r (\omega_l + \omega_r)$$

$$a = 1/d r (\omega_l - \omega_r)$$

$$\omega_l = (v + (d * a)/2)/r$$

$$\omega_r = (v - (d * a)/2)/r$$



# PYTHON TIME

```
import math
from geometry_msgs.msg import Twist
```





$$v = \frac{1}{2} r (\omega_l + \omega_r) \\ = (r\omega_l + r\omega_r) / 2$$

$$a = \frac{1}{d} r (\omega_l - \omega_r) \\ = (r\omega_l - r\omega_r) / d$$

# PYTHON TIME

```
import math
from geometry_msgs.msg import Twist

wheel_radius = 1
robot_radius = 1

# computing the forward kinematics for a differential drive
def forward_kinematics(w_l, w_r):
    c_l = wheel_radius * w_l
    c_r = wheel_radius * w_r
    v = (c_l + c_r) / 2
    a = (c_l - c_r) / robot_radius
    return (v, a)
```

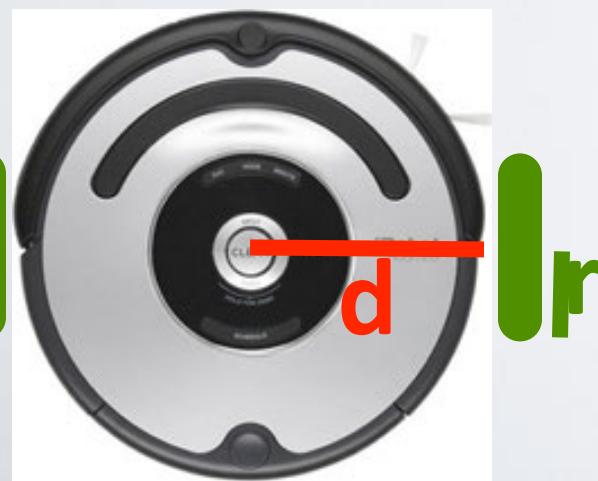




# PYTHON TIME

$$\omega_l = (v + (d * a) / 2) / r$$

$$\omega_r = (v - (d * a) / 2) / r$$



```
import math
from geometry_msgs.msg import Twist

wheel_radius = 1
robot_radius = 1

# computing the forward kinematics for a differential drive
def forward_kinematics(w_l, w_r):
    c_l = wheel_radius * w_l
    c_r = wheel_radius * w_r
    v = (c_l + c_r) / 2
    a = (c_l - c_r) / robot_radius
    return (v, a)

# computing the inverse kinematics for a differential drive
def inverse_kinematics(v, a):
    c_l = v + (robot_radius * a) / 2
    c_r = v - (robot_radius * a) / 2
    w_l = c_l / wheel_radius
    w_r = c_r / wheel_radius
    return (w_l, w_r)
```



# Forward and inverse Kinematics for Turtlebot

We define two functions `forward_kinematics` and `inverse_kinematics` respectively. They allow to convert from wheel speeds to robot motion and from desired robot motion to wheel speeds.

- $v$  denotes the linear velocity of the robot
- $a$  denotes the angular velocity of the robot
- $w_l$  is the angular velocity of the left wheel
- $w_r$  is the angular velocity of the right wheel

unit are m for geometry, m/s for linear velocity and rad/s for angular velocity

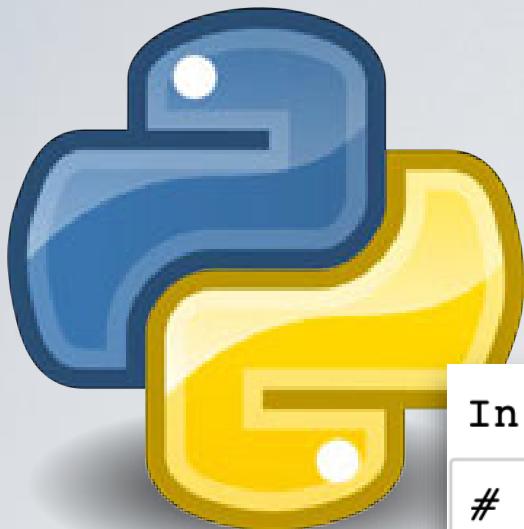
In [8]:

```
import math
#from geometry_msgs.msg import Twist

# some estimates for the robot geometry
wheel_radius = 0.05 # 5 cm radius of wheel
robot_radius = 0.25 # 25 cm radio of base

# computing the forward kinematics for a differential drive
def forward_kinematics(w_l, w_r):
    c_l = wheel_radius * w_l
    c_r = wheel_radius * w_r
    v = (c_l + c_r) / 2
    a = (c_l - c_r) / robot_radius
    return (v, a)

# computing the inverse kinematics for a differential drive
def inverse_kinematics(v, a):
    c_l = v + (robot_radius * a) / 2
    c_r = v - (robot_radius * a) / 2
    w_l = c_l / wheel_radius
    w_r = c_r / wheel_radius
    return (w_l, w_r)
```



# PYTHON TIME

In [13]:

```
# try out forward kinematics, both wheels turning at `2pi rad/s` (one full turn per second)

(v, a) = forward_kinematics(2*math.pi, 2*math.pi)
print "v = %f,\ta = %f" % (v, a)

v = 0.314159,    a = 0.000000
```

In [15]:

```
# try out forward kinematics, one wheel turning at `2pi rad/s` (one full turn per second), the other `-2pi rad/s`

(v, a) = forward_kinematics(2*math.pi, -2*math.pi)
print "v = %f,\ta = %f" % (v, a)

v = 0.000000,    a = 2.513274
```

In [16]:

```
# inverse kinematics:

(w_l, w_r) = inverse_kinematics(1.0, 0.0)
print "w_l = %f,\tw_r = %f" % (w_l, w_r)

# this should give us again the desired values:
(v, a) = forward_kinematics(w_l, w_r)
print "v = %f,\ta = %f" % (v, a)

w_l = 20.000000,          w_r = 20.000000
v = 1.000000,    a = 0.000000
```



# LiveSlides web content

To view

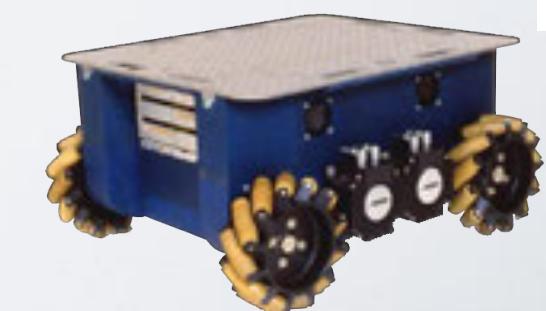
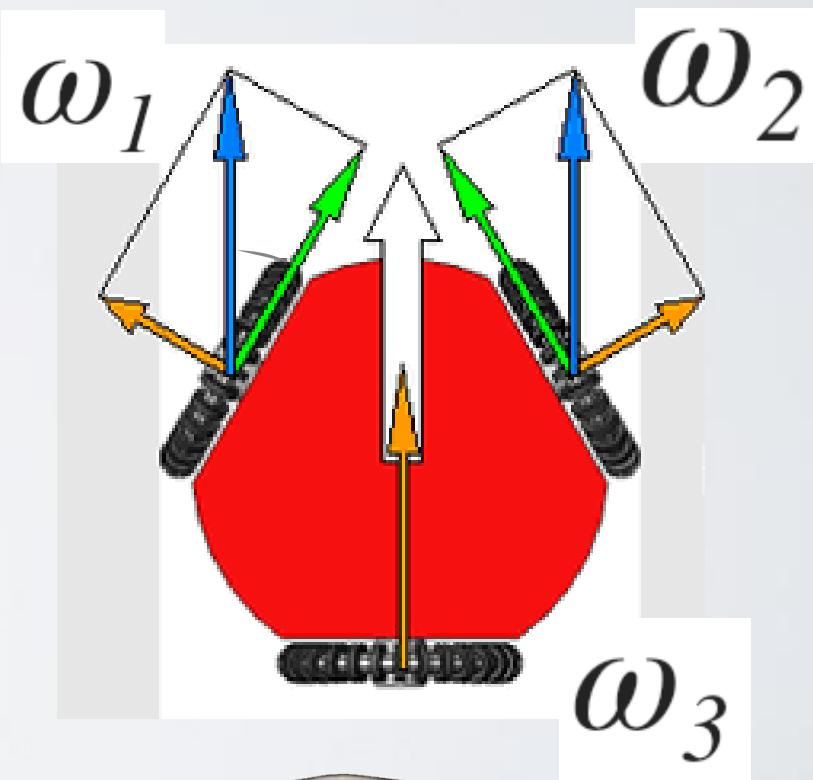
**Download the add-in.**

[liveslides.com/download](http://liveslides.com/download)

**Start the presentation.**

# A BIT MORE EXOTIC CONTROLLING ROVIO

- ▶ Which speeds to turn each wheel to move in desired direction?
- ▶ That corresponds to the Voltage for every motor
- ▶ This is a control problem!
- ▶ Rovio has “universal wheels” or “omni wheel”
  - ▶ simple design
  - ▶ significant slippage
  - ▶ improvement: “Mecanum wheel”



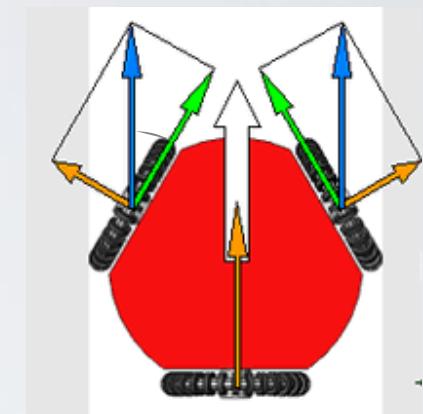
# USING THE KINEMATIC MODEL

- ▶ Straight line (simplified for wheel radius=1)

$$\omega = \omega_1 = -\omega_2, \quad \omega_3 = 0$$

$$V_R = \frac{\omega r}{\cos(\alpha)}$$

forward kinematics

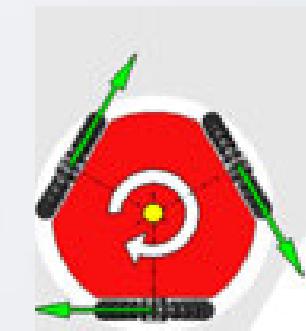


- ▶ Pure rotation

$$\omega = \omega_1 = \omega_2 = \omega_3$$

$$\omega_R = \frac{\omega r}{l}$$

forward kinematics



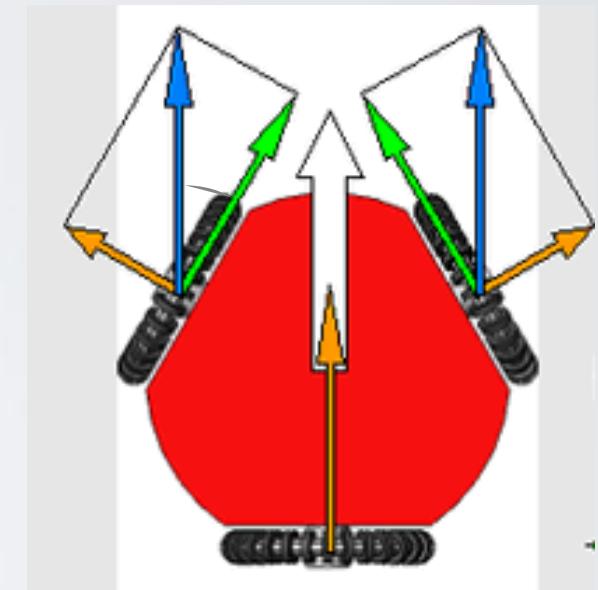
# INVERSE KINEMATICS FOR ROVIO

- General (omni-drive) velocity control example

$$\omega_i = \vec{w}_i \cdot \vec{d}$$

$$\vec{w}_i = \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}$$

$$\vec{d} = \begin{pmatrix} \cos(\delta) \\ \sin(\delta) \end{pmatrix} \cdot v$$

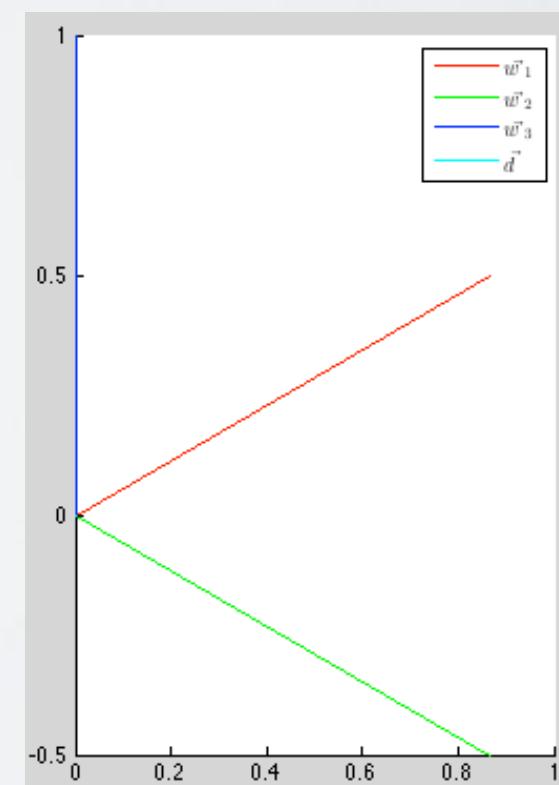
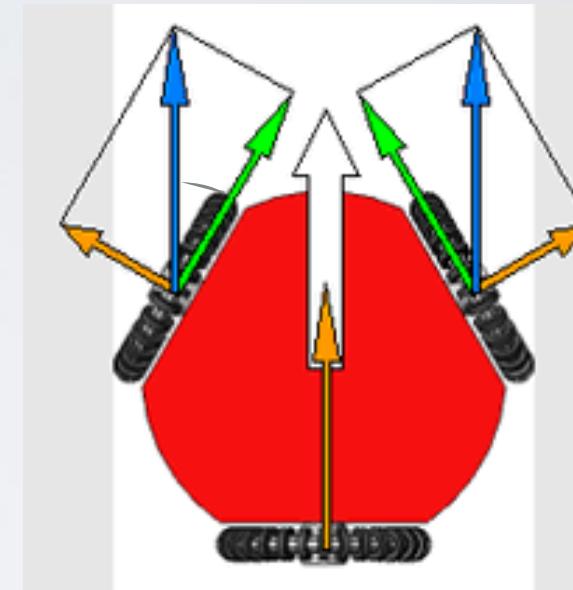


# ROVIO KINEMATICS IN MATLAB

$$\vec{w}_i = \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix}$$

```
% Rovio Geometry
% (angles of wheels w.r.t. forward vector)
alpha(1)=pi/6;
alpha(2)=-pi/6;
alpha(3)=pi/2;

% create vectors w(1-3), based on angles
for i=1:3
    w(1,i)=cos(alpha(i));
    w(2,i)=sin(alpha(i));
end;
```

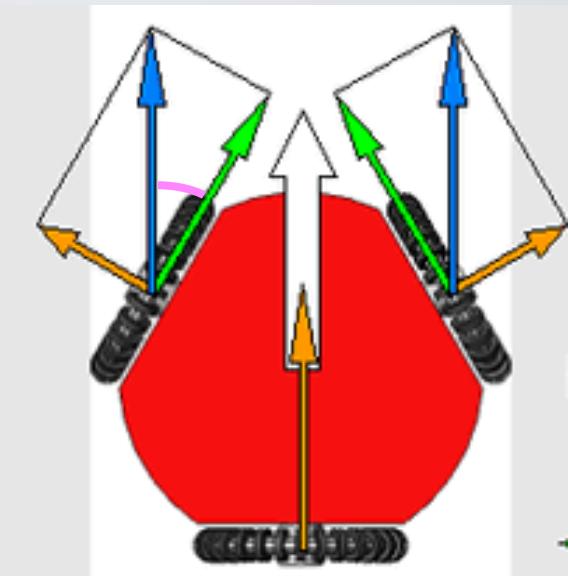


# ROVIO KINEMATICS IN MATLAB

```
% Rovio Geometry
% (angles of wheels with regard to forward vector)
alpha(1)=pi/6;
alpha(2)=-pi/6;
alpha(3)=pi/2;

% create vectors w(1-3) for wheel directions
for (i=1:3)
    w(1,i)=cos(alpha(i));
    w(2,i)=sin(alpha(i));
end;
```

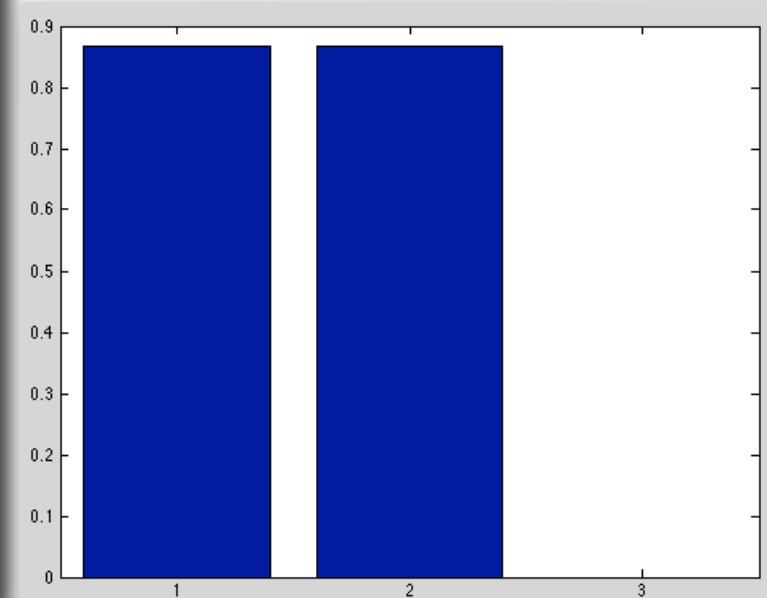
$$\begin{aligned}\omega_i &= \vec{w}_i \cdot \vec{d} \\ \vec{w}_i &= \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix} \\ \vec{d} &= \begin{pmatrix} \cos(\delta) \\ \sin(\delta) \end{pmatrix} \cdot v\end{aligned}$$



```
% now set the desired velocity and direction of the omnidrive
%desired velocity
v=1;
%desired direction
delta=0;
% -pi/3;

%resulting desired velocity vector
d=[cos(delta);sin(delta)] * v

for (i=1:3)
    omega(i)=w(:,i)'*d;
end;
```



# FOR MOTION CONTROL

Model



Algorithm

Kinematics / Dynamics

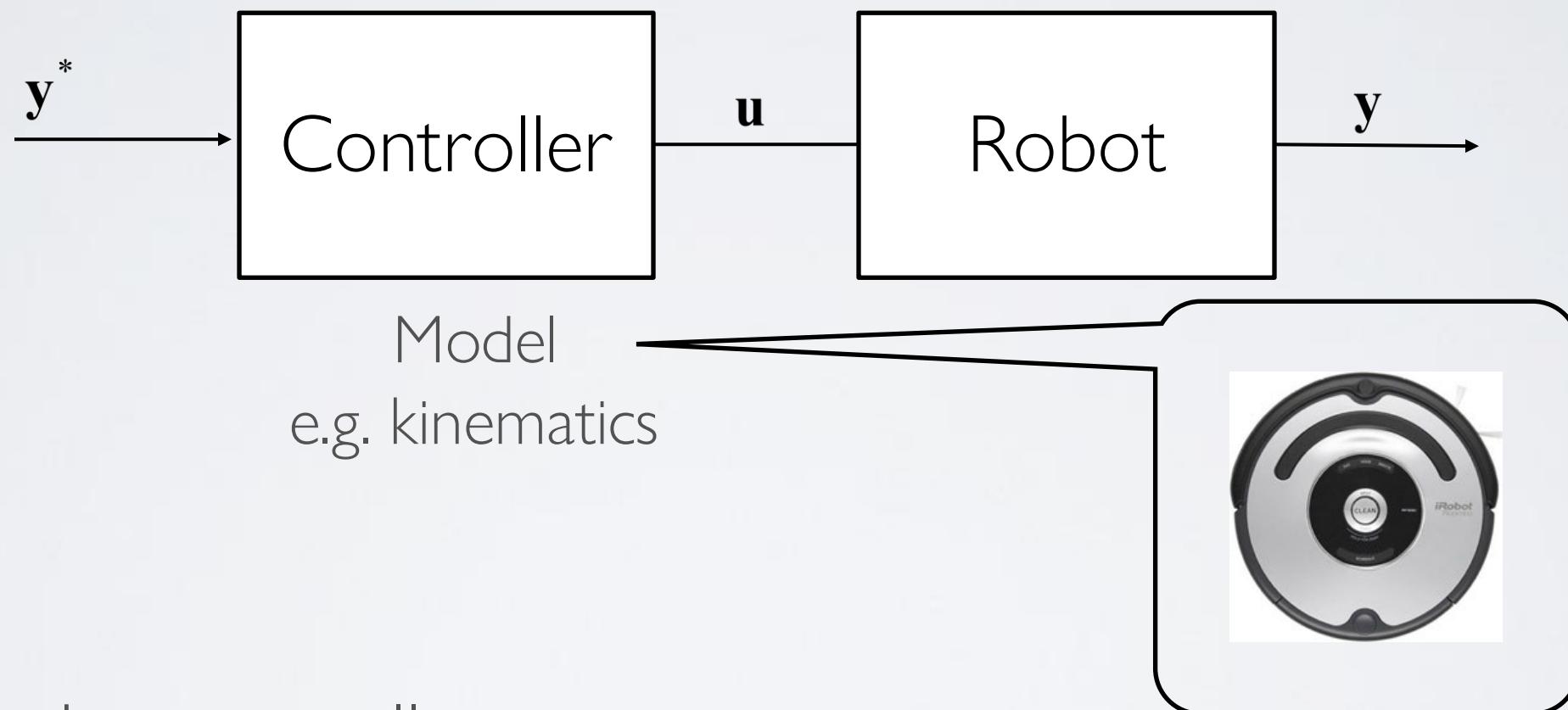
control-parameter

Engineering

control signal => actuators

# OPEN LOOP CONTROL

Reference  
(e.g. desired position/vel)      Input  
(e.g. drive command)      Output  
(e.g. robot's position/vel)



- ▶ Task for the controller:
  - ▶ generate control signals  $u$  so that  $y = y^*$

# YOU ARE A ROBOT

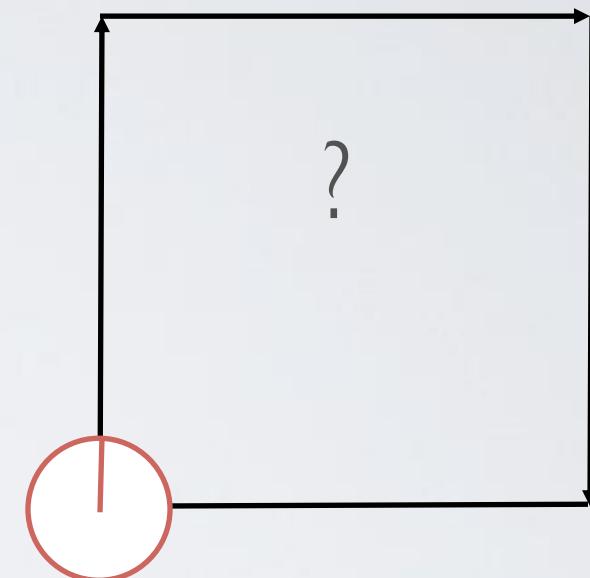
- ▶ close your eyes and hit the spot!



- ▶ ... and that's not even really open-loop control
- ▶ Proprioception: "one's own" and perception, is the sense of the relative position of neighbouring parts of the body and strength of effort being employed in movement.

# OPEN LOOP CONTROL

- ▶ No measurements - no feedback
- ▶ can rely on a model only
- ▶ better models – smaller errors, however the errors accumulate over time
- ▶ no possibility of correcting the errors since they are unknown
- ▶ how to deal with unexpected events, changes, obstacles, etc.?



# OPEN LOOP CONTROL, MOVE IN A SQUARE

```
# 5 HZ
r = rospy.Rate(5);

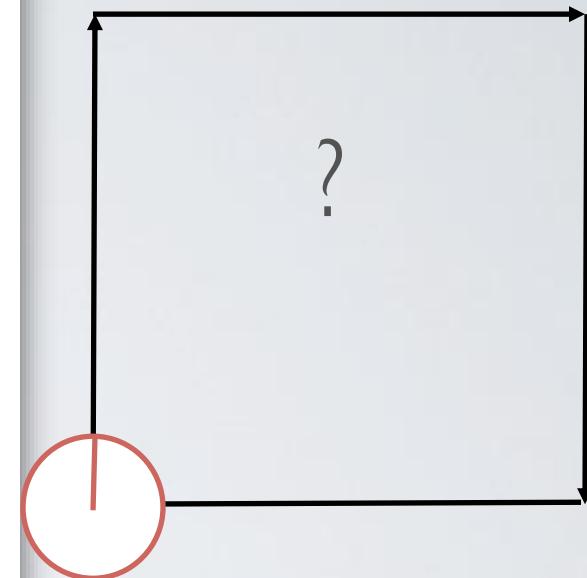
# create two different Twist() variables. One for moving forward. One for turning 45 degrees.

# let's go forward at 0.2 m/s
move_cmd = Twist()
move_cmd.linear.x = 0.2
# by default angular.z is 0 so setting this isn't required

#let's turn at 45 deg/s
turn_cmd = Twist()
turn_cmd.linear.x = 0
turn_cmd.angular.z = radians(45); #45 deg/s in radians/s

#two keep drawing squares. Go forward for 2 seconds (10 x 5 HZ) then turn for 2 second
count = 0

while not rospy.is_shutdown():
    # go forward 0.4 m (2 seconds * 0.2 m / seconds)
    rospy.loginfo("Going Straight")
    for x in range(0,10):
        self.cmd_vel.publish(move_cmd)
        r.sleep()
    # turn 90 degrees
    rospy.loginfo("Turning")
    for x in range(0,10):
        self.cmd_vel.publish(turn_cmd)
        r.sleep()
    count = count + 1
    if(count == 4):
        count = 0
    if(count == 0):
        rospy.loginfo("TurtleBot should be close to the original starting position (but it's probably way off)")
```





# LiveSlides web content

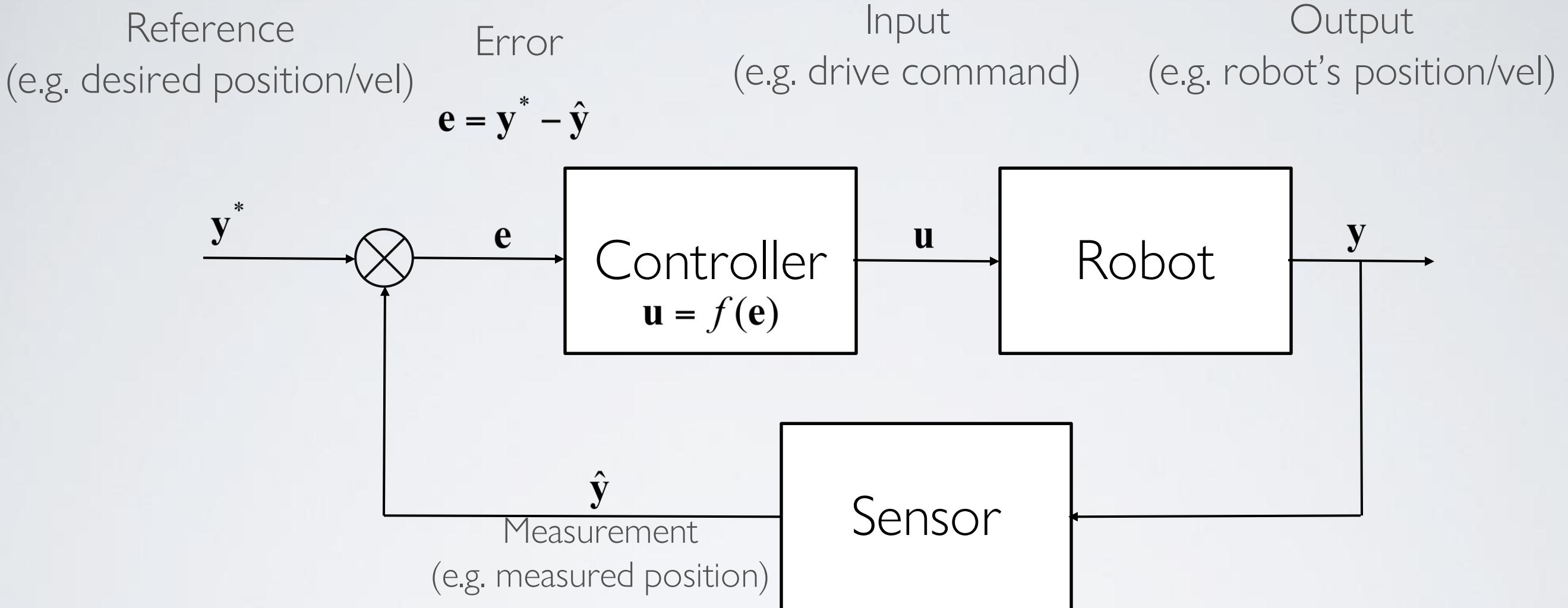
To view

**Download the add-in.**

[liveslides.com/download](http://liveslides.com/download)

**Start the presentation.**

# CLOSED LOOP CONTROL



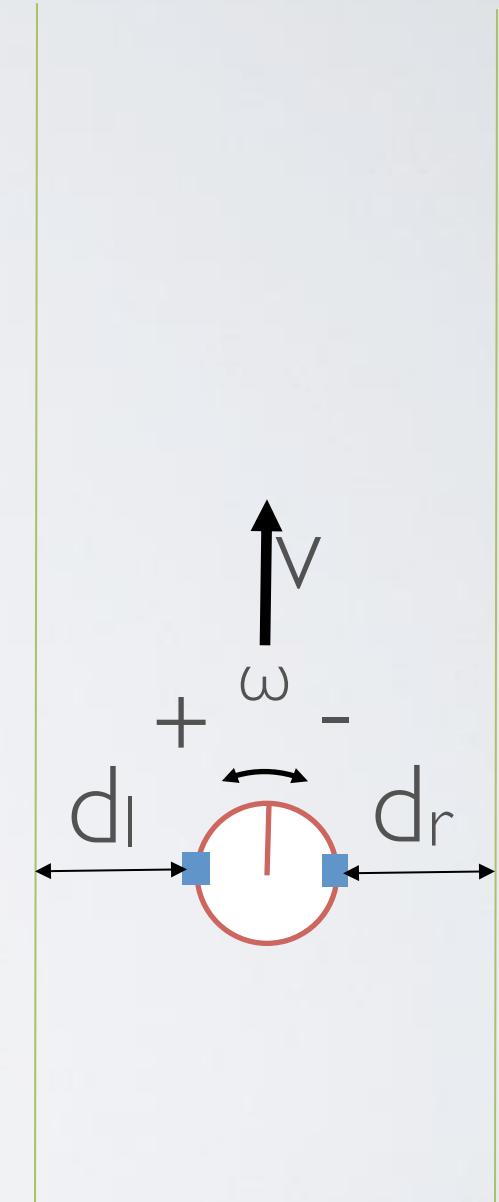
- ▶ Task for the controller:
  - ▶ generate control signals  $u$  that will keep error  $e$  as small as possible (0 would be the best)

# DRIVE DISTANCE - FEEDBACK CONTROL

- ▶ Repeat
  - ▶ issue DriveForward command  $\mathbf{u}$
  - ▶ read odometry (proprioception again) and accumulate the total distance travelled  $\hat{\mathbf{y}}$
  - ▶ stop if the total distance is greater than the specified value
$$\hat{\mathbf{y}} \geq \mathbf{y}^*$$
- ▶ Smaller time intervals – smaller errors

# CORRIDOR FOLLOWING

- ▶ Scenario:
  - ▶ two sensors measuring distance to the wall  $d_l$  and  $d_r$
  - ▶ robot moves constantly forward ( $v$ )
  - ▶ controller affects the angular speed only ( $u = \omega$ )
- ▶ Controller task:
  - ▶ keep  $d_l = d_r$

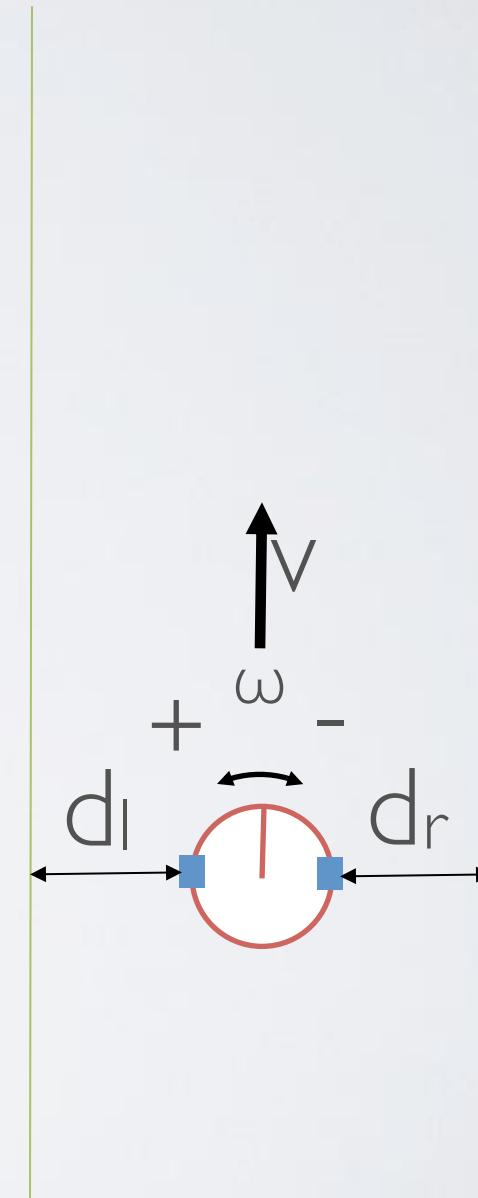


# SIMPLE CONTROLLER

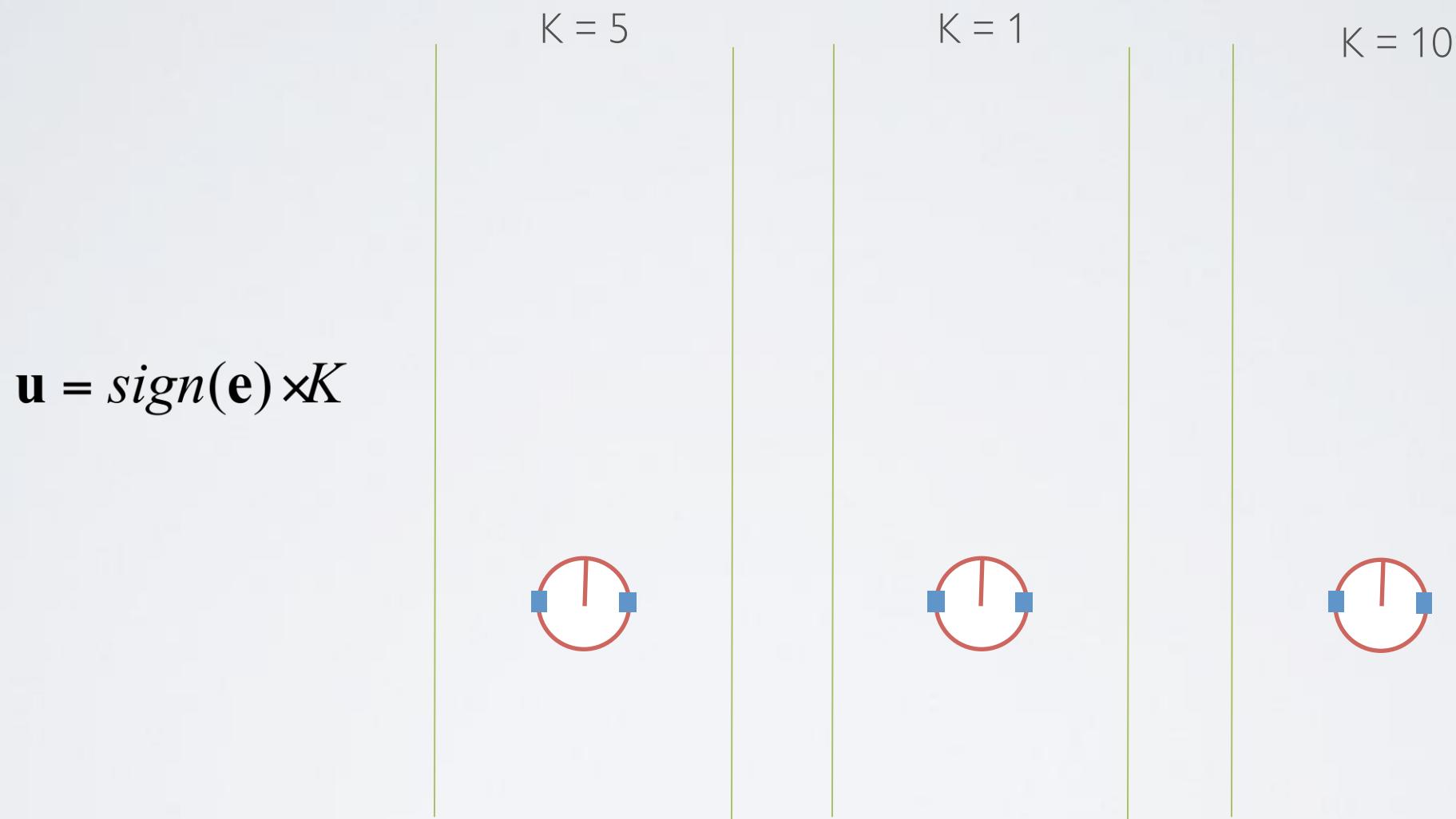
- ▶ Loop:
  - ▶ measure  $e = d_l - d_r$
  - ▶ if  $e > 0$  then  $\omega = +K$
  - ▶ if  $e < 0$  then  $\omega = -K$
  - ▶  $\omega = K \text{ sign}(e)$
- ▶ In Rovio language:
  - ▶ if  $e > 0$  then RotateLeft(5)
  - ▶ if  $e < 0$  then RotateRight(5)

constant  
parameter

$$K = 5$$



# BANG-BANG CONTROLLER



- ▶ Control input depends only on the sign of the error



# PROPORTIONAL CONTROLLER

- ▶ Change  $\omega$  proportionally to the error value

- ▶  $\omega = e * K_p$

- ▶ small  $e$  – small correction

- ▶ large  $e$  – large correction

- ▶ Result

- ▶ smoother actions and smaller errors

- ▶  $K_p$  parameter

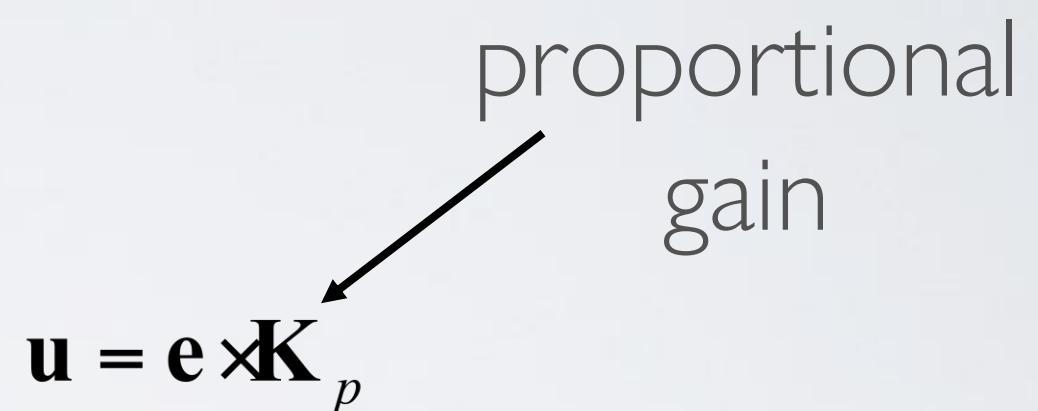
- ▶ large – faster reaction

- ▶ small – slower

- ▶ optimal value: smooth behaviour, robust to changes

$$\mathbf{u} = \mathbf{e} \times \mathbf{K}_p$$

proportional  
gain



# VISION-BASED CONTROL

- ▶ Object state – in our case: x position and size

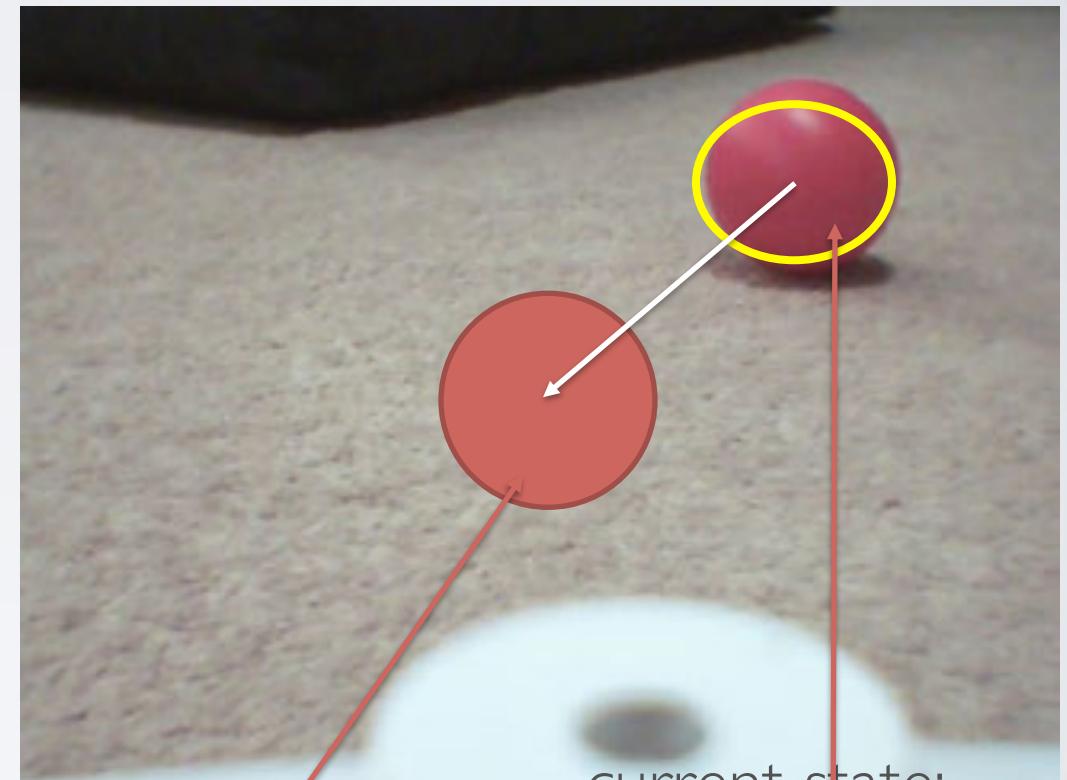
$$\mathbf{y} = \begin{bmatrix} x \\ w \times h \end{bmatrix}$$

- ▶ Error - difference between the desired and current state

$$\mathbf{e} = \mathbf{y}^* - \hat{\mathbf{y}}$$

- ▶ Control input u:

- ▶ Spin – to adjust the x position
- ▶ DriveForward - to adjust the size



desired state

current state:  
information from  
the object detector

# FOR MOTION CONTROL

Model



Algorithm



Kinematics / Dynamics

control-parameter

Engineering

control signal => actuators



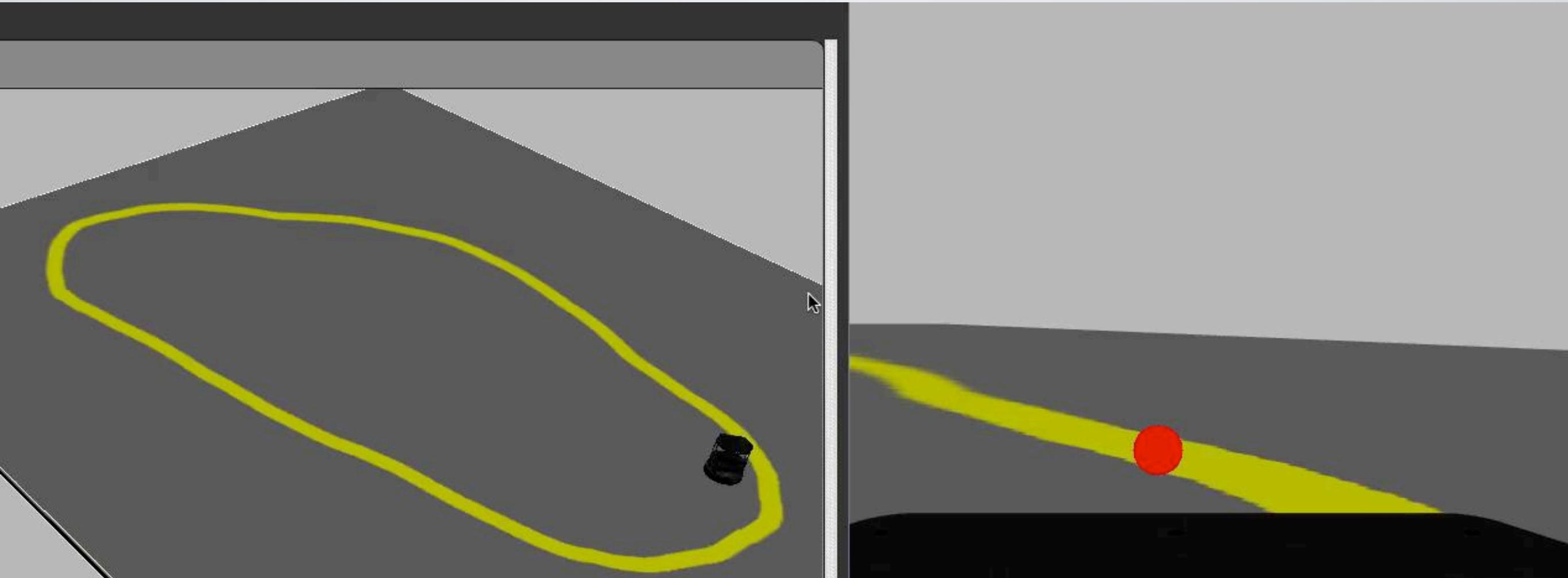
**Line following project**  
**Two light sensors.**  
**Proportional control of the**  
**Black level (drive forward**  
**on White surface)**  
**Obstacle avoidance using US**  
**sensor**



**LUND**  
UNIVERSITY

**Edited by Gunnar Bolmsjö**  
**2011**

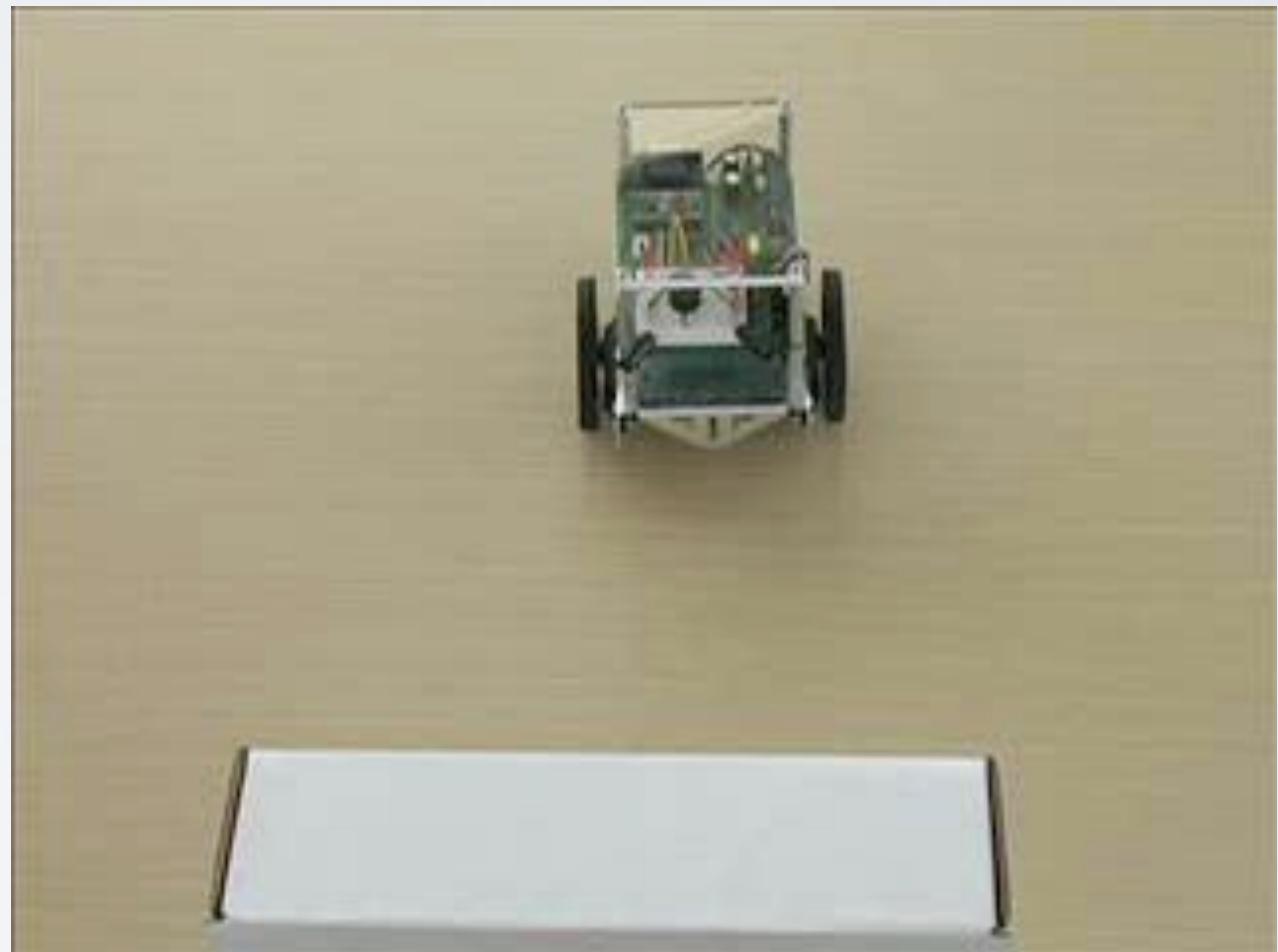
# NOW FOR THE REAL STUFF



rian Gerkey & And William D. Smart. (2015) *Programming Robots with*

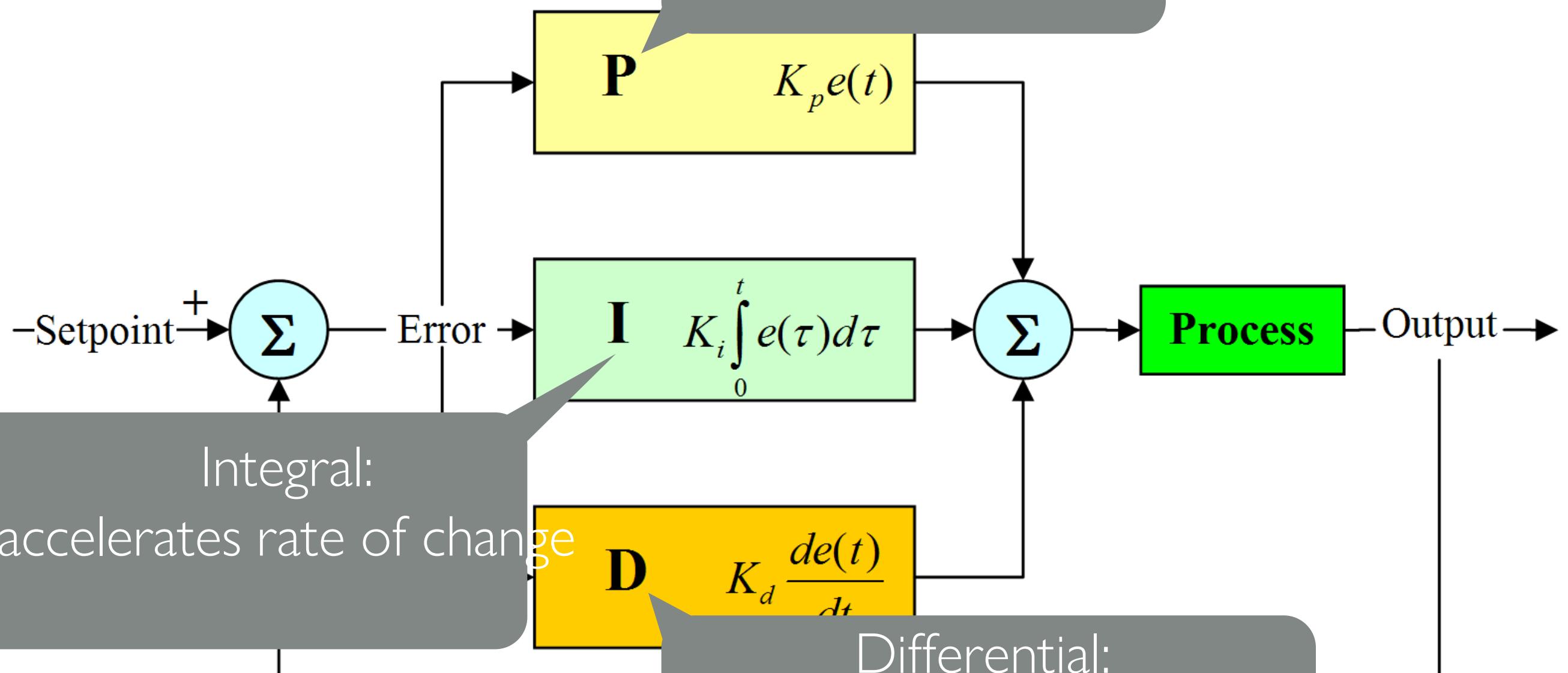
# PROBLEMS WITH PROPORTIONAL CONTROLLER?

- ▶ What's the problem?



# PID CONTROLLER

Proportional: corrects the actual error

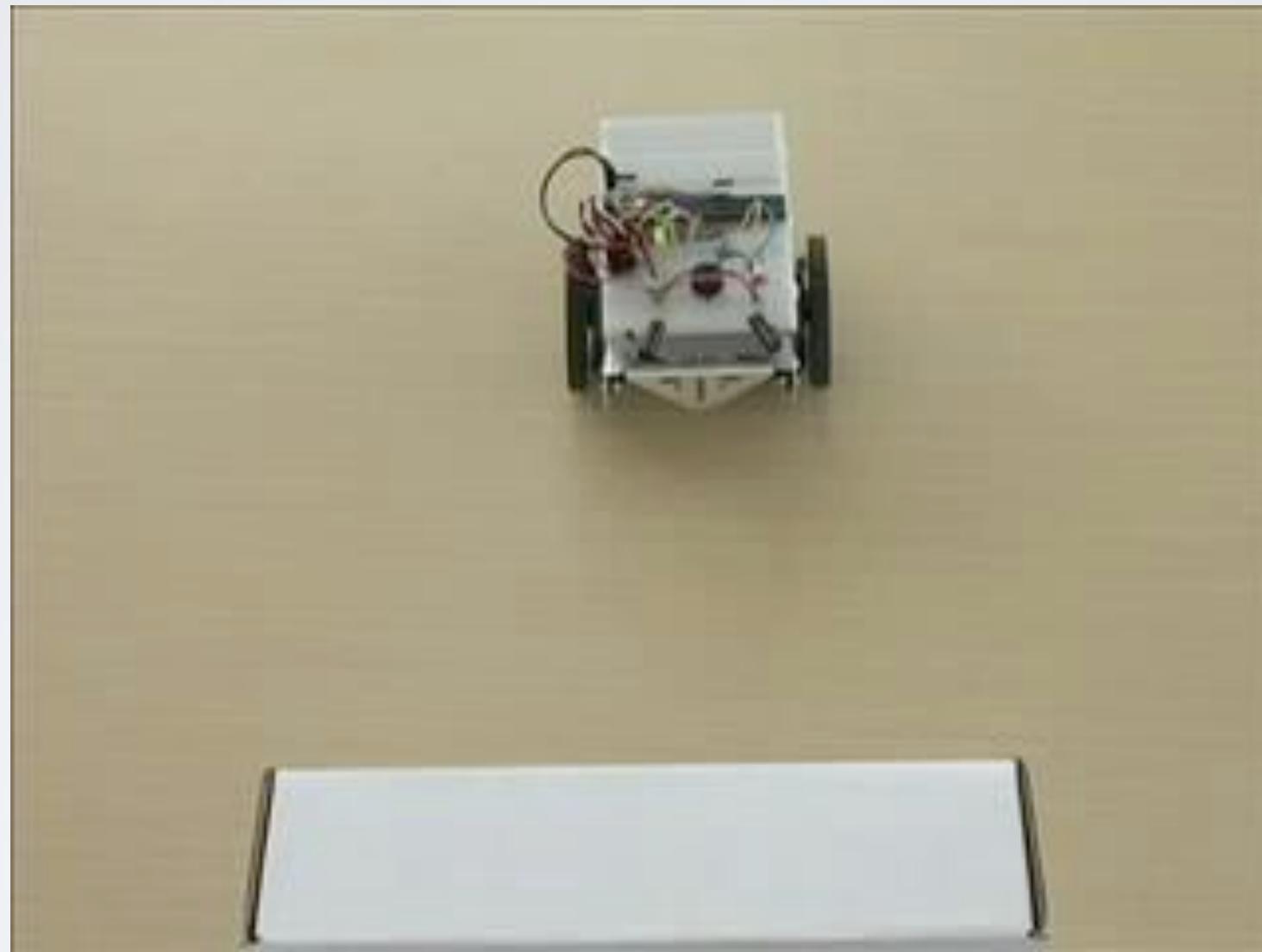


Integral:  
accelerates rate of change

Differential:  
slows the rate of change  
(reduces overshooting)

© SilverSTart@Wikipedia

# PID CONTROLLER



# APPLICATIONS – EXAMPLES

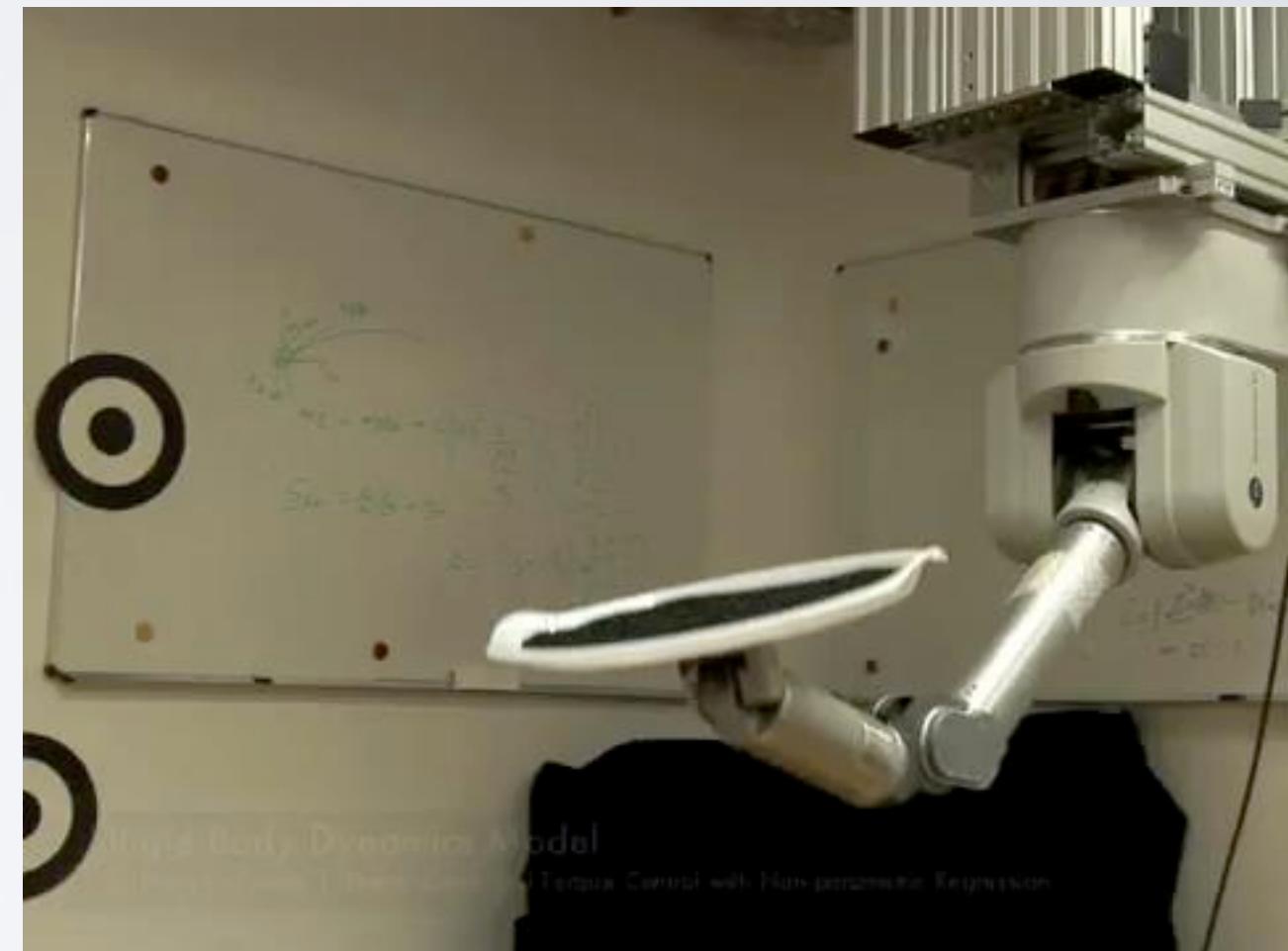
- ▶ Balancing Robot

**NXTway-G**  
Designed, built and  
programmed by  
**Ryo Watanabe**  
**Waseda University**  
**Japan**

- ▶ Person Following



# ANY OTHER CONTROL PROBLEMS YOU COULD THINK OF IN ROBOTICS?



# CONTROL THEORY

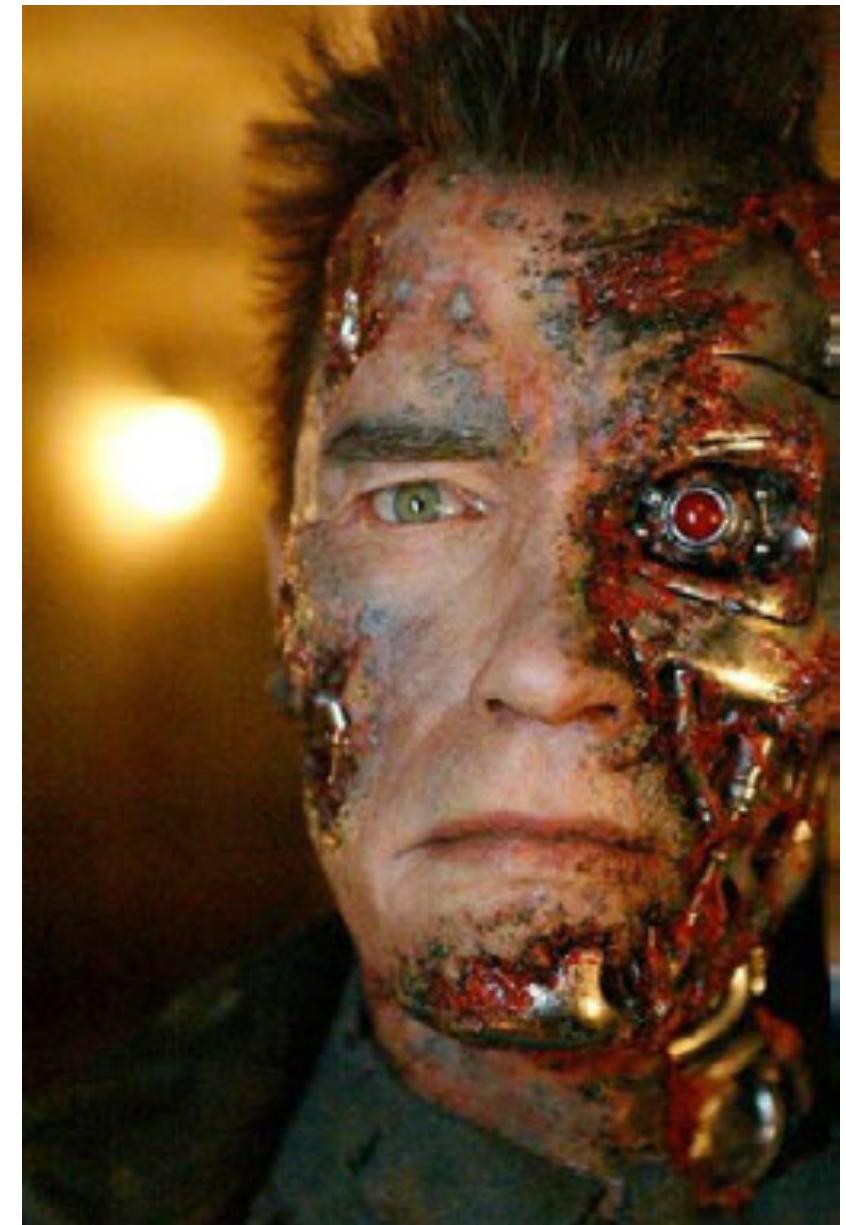
- ▶ Optimal control, minimise
  - ▶ errors
  - ▶ energy
  - ▶ response time
- ▶ non-linear
- ▶ adaptive controllers
- ▶ changing parameter values in time
- ▶ models
- ▶ kinematics and dynamics in robotics

read Siegwart book, chapter 4!

# End of Lecture Feedback

When survey is active, respond at [PollEv.com/mhanheide](https://PollEv.com/mhanheide)

THANK YOU  
FOR LISTENING!



**0 surveys done**

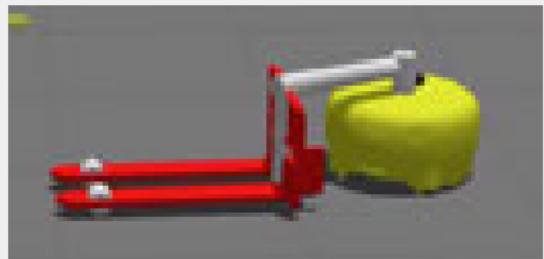
⟳ 0 surveys underway

# CMP3103    AUTONOMOUS    MOBILE ROBOTICS

## ROBOT CONTROL

GEESARA KULATHUNGA

APRIL 29, 2024



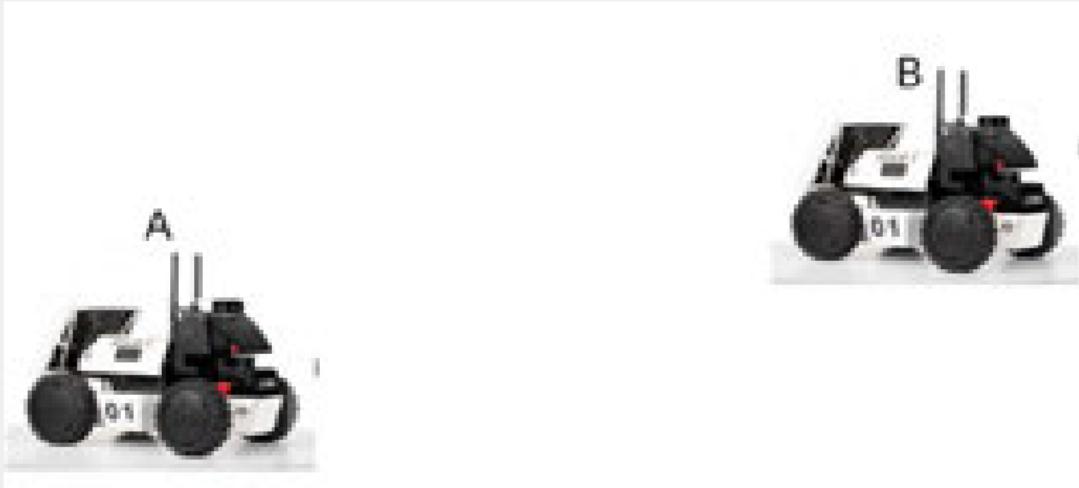
# CONTROL OF MOBILE ROBOTS

# CONTENTS

- **Kinematics of wheeled mobile robots:** internal, external, direct, and inverse
  - ▶ Differential drive kinematics
- Open loop and closed loop control
- Wheeled Mobile System Control: **pose** and **orientation**
  - ▶ Control to reference pose
  - ▶ Control to reference pose via an intermediate point
  - ▶ Control to reference pose via an intermediate direction



# TIME TO ROLL UP OUR SLEEVES. LET'S GET TO WORK!



- How do we **best move** the robot from A to B?
- **Optimal robot path** from A to B?
- Identify the **optimal motion control strategy** for robot navigation from configuration A to configuration B?
- Determine the **minimum-cost path** for robot motion under the constraints of starting pose A and target pose B?

# KINEMATICS OF WHEELED MOBILE ROBOTS

- The process of moving an autonomous system from one place to another is called **Locomotion**



[www.proantic.com/en/display.php](http://www.proantic.com/en/display.php)



# KINEMATICS OF WHEELED MOBILE ROBOTS

- The process of moving an autonomous system from one place to another is called **Locomotion**
- **Kinematic model** describes **geometric relationship** of the system and the system velocities and is presented by **a set of first-order differential equations**



[www.proantic.com/en/display.php](http://www.proantic.com/en/display.php)

# KINEMATICS OF WHEELED MOBILE ROBOTS

- The process of moving an autonomous system from one place to another is called **Locomotion**
- **Kinematic model** describes **geometric relationship** of the system and the system velocities and is presented by **a set of first-order differential equations**
- **Dynamic models** describe a **system motion when forces are applied** to the system and the model is described by **a set of second-order differential equations**



[www.proantic.com/en/display.php](http://www.proantic.com/en/display.php)

# KINEMATICS OF WHEELED MOBILE ROBOTS

- The process of moving an autonomous system from one place to another is called **Locomotion**
- **Kinematic model** describes **geometric relationship** of the system and the system velocities and is presented by **a set of first-order differential equations**
- **Dynamic models** describe a **system motion when forces are applied** to the system and the model is described by **a set of second-order differential equations**
- For mobile robotics **kinematic model is sufficient**



[www.proantic.com/en/display.php](http://www.proantic.com/en/display.php)



# KINEMATICS OF WHEELED MOBILE ROBOTS

- The number of directions in which motion can be made is called DOF

# KINEMATICS OF WHEELED MOBILE ROBOTS

- The number of directions in which motion can be made is called DOF
- A car has 2 DOF

# KINEMATICS OF WHEELED MOBILE ROBOTS

- The number of directions in which motion can be made is called DOF
- A car has 2 DOF
- Highly efficient on hard surfaces compared to Legged locomotion

# KINEMATICS OF WHEELED MOBILE ROBOTS

- The number of directions in which motion can be made is called DOF
- A car has 2 DOF
- Highly efficient on hard surfaces compared to Legged locomotion
- There is no direct way to measure the current pose



# KINEMATICS OF WHEELED MOBILE ROBOTS

- The number of directions in which motion can be made is called DOF
- A car has 2 DOF
- Highly efficient on hard surfaces compared to Legged locomotion
- There is no direct way to measure the current pose
- **Holonomic Systems** - a robot is able to move instantaneously in any the direction in the space of its degree of freedom (**Omnidirectional** robot)



# KINEMATICS OF WHEELED MOBILE ROBOTS

- The number of directions in which motion can be made is called DOF
- A car has 2 DOF
- Highly efficient on hard surfaces compared to Legged locomotion
- There is no direct way to measure the current pose
- **Holonomic Systems** - a robot is able to move instantaneously in any the direction in the space of its degree of freedom (**Omnidirectional** robot)
- **Non-holonomic Systems** - a robot is not able to move instantaneously in any direction in the space of its degree of freedom



# KINEMATICS OF WHEELED MOBILE ROBOTS

Several types of kinematic models exist

- **Internal kinematics**: consider internal variables (wheel rotation and robot motion)



# KINEMATICS OF WHEELED MOBILE ROBOTS

Several types of kinematic models exist

- **Internal kinematics**: consider internal variables (wheel rotation and robot motion)
- **External kinematics**: robot pose relative to a reference frame



# KINEMATICS OF WHEELED MOBILE ROBOTS

Several types of kinematic models exist

- **Internal kinematics**: consider internal variables (wheel rotation and robot motion)
- **External kinematics**: robot pose relative to a reference frame
- **Direct kinematics**: robot states as a function of its inputs (wheel speed and joints motions)



# KINEMATICS OF WHEELED MOBILE ROBOTS

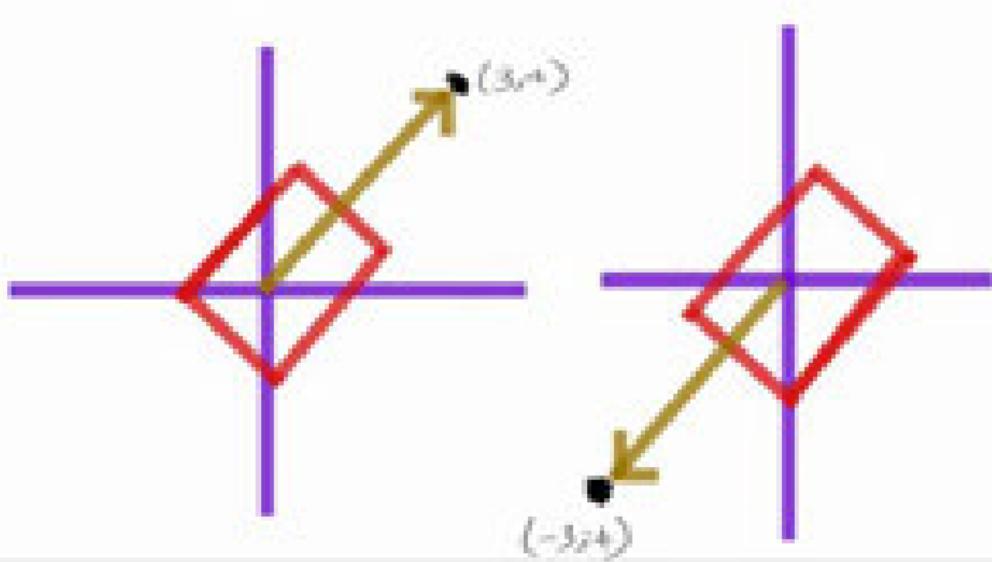
Several types of kinematic models exist

- **Internal kinematics**: consider internal variables (wheel rotation and robot motion)
- **External kinematics**: robot pose relative to a reference frame
- **Direct kinematics**: robot states as a function of its inputs (wheel speed and joints motions)
- **Inverse kinematics**: robot inputs as a function of the desired robot pose



# THE DIFFERENCE BETWEEN ATAN AND ATAN2

Can you estimate the orientation of the robot?



# THE DIFFERENCE BETWEEN ATAN AND ATAN2

Quadrant	Angle	sin	cos	tan
I	$0 < \theta < \pi/2$	+	+	+
II	$\pi/2 < \theta < \pi$	+	-	-
III	$\pi < \theta < 3\pi/2$	-	-	+
IV	$3\pi/2 < \theta < 2\pi$	-	+	-

- $|A \cdot B| = |A||B|COS(\theta)$  and  $|A \times B| = |A||B|SIN(\theta)$

# THE DIFFERENCE BETWEEN ATAN AND ATAN2

Quadrant	Angle	sin	cos	tan
I	$0 < \theta < \pi/2$	+	+	+
II	$\pi/2 < \theta < \pi$	+	-	-
III	$\pi < \theta < 3\pi/2$	-	-	+
IV	$3\pi/2 < \theta < 2\pi$	-	+	-

- $|A \cdot B| = |A||B|\cos(\theta)$  and  $|A \times B| = |A||B|\sin(\theta)$
- $\text{arctan2}(|A \times B|/|A \cdot B|)$

# THE DIFFERENCE BETWEEN ATAN AND ATAN2

Quadrant	Angle	sin	cos	tan
I	$0 < \alpha < \pi/2$	+	+	+
II	$\pi/2 < \alpha < \pi$	+	-	-
III	$\pi < \alpha < 3\pi/2$	-	-	+
IV	$3\pi/2 < \alpha < 2\pi$	-	+	-

- If  $\tan(\alpha)$  is **positive**, it could come from either the **first** or **third** quadrant and if it is **negative**, it could come from either the **second** or **fourth** quadrant. Hence, `atan()` returns an angle from the first or fourth quadrant (i.e.  $-\pi/2 \leq \text{atan}() \leq \pi/2$ ), regardless of the original input to the tangent

# THE DIFFERENCE BETWEEN ATAN AND ATAN2

Quadrant	Angle	sin	cos	tan
I	$0 < \alpha < \pi/2$	+	+	+
II	$\pi/2 < \alpha < \pi$	+	-	-
III	$\pi < \alpha < 3\pi/2$	-	-	+
IV	$3\pi/2 < \alpha < 2\pi$	-	+	-

- If  $\tan(\alpha)$  is **positive**, it could come from either the **first** or **third** quadrant and if it is **negative**, it could come from either the **second** or **fourth** quadrant. Hence, `atan()` returns an angle from the first or fourth quadrant (i.e.  $-\pi/2 \leq \text{atan}() \leq \pi/2$ ), regardless of the original input to the tangent
- To **get full information**, the values of the **sine and cosine** are **considered separately**. And this is what `atan2()` does. It takes both, the  $\sin(\alpha)$  and  $\cos(\alpha)$  and **resolves all four quadrants** by adding  $\pi$  to the result of `atan()` whenever the **cosine is negative**



# THE DIFFERENCE BETWEEN ATAN AND ATAN2

Quadrant	Angle	sin	cos	tan
I	$0 < \alpha < \pi/2$	+	+	+
II	$\pi/2 < \alpha < \pi$	+	-	-
III	$\pi < \alpha < 3\pi/2$	-	-	+
IV	$3\pi/2 < \alpha < 2\pi$	-	+	-

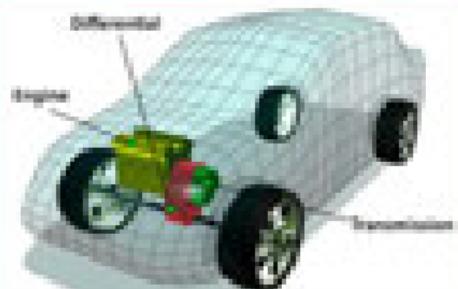
- If  $\tan(\alpha)$  is **positive**, it could come from either the **first** or **third** quadrant and if it is **negative**, it could come from either the **second** or **fourth** quadrant. Hence, `atan()` returns an angle from the first or fourth quadrant (i.e.  $-\pi/2 \leq \text{atan}() \leq \pi/2$ ), regardless of the original input to the tangent
- To **get full information**, the values of the **sine and cosine** are **considered separately**. And this is what `atan2()` does. It takes both, the  $\sin(\alpha)$  and  $\cos(\alpha)$  and **resolves all four quadrants** by adding  $\pi$  to the result of `atan()` whenever the **cosine is negative**
- $\text{atan2: } -\pi < \text{atan2}(y,x) < \pi$  and  $\text{atan: } -\pi/2 < \text{atan}(y/x) < \pi/2$

# DIFFERENTIAL DRIVE KINEMATICS

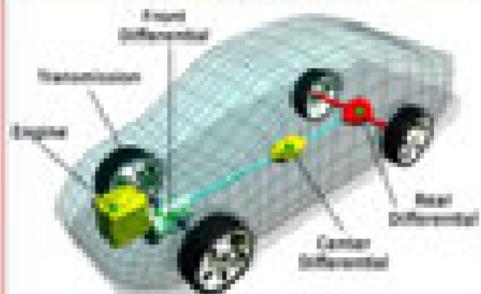
Rear-Wheel Drive



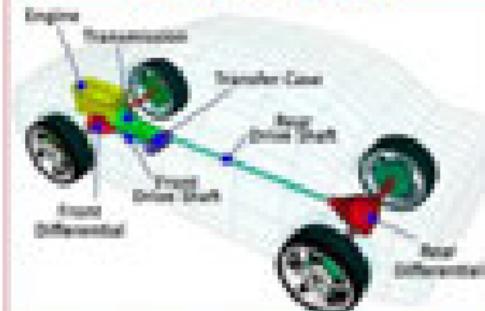
Front-Wheel Drive



All-Wheel Drive



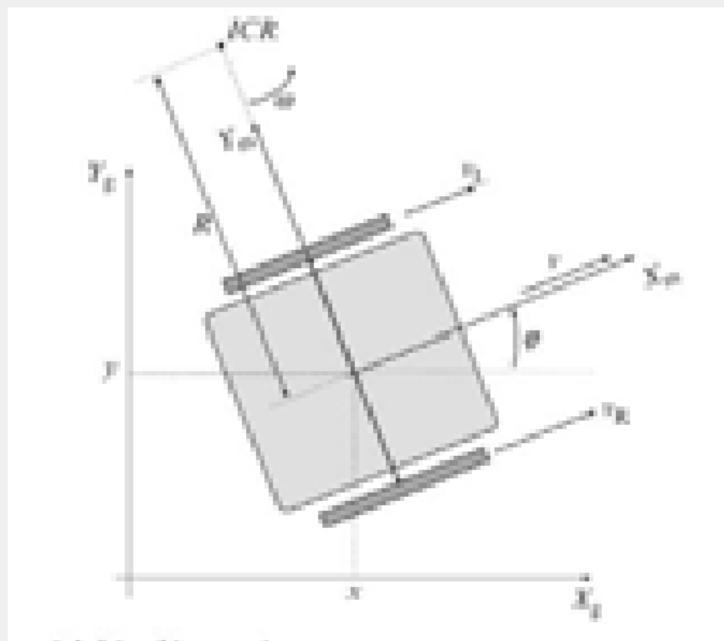
Four-Wheel Drive



<https://cartreatments.com/types-of-differentials-how-they-work/>

# DIFFERENTIAL DRIVE KINEMATICS

# DIFFERENTIAL DRIVE KINEMATICS



# DIFFERENTIAL DRIVE KINEMATICS

- Well-fit for **smaller mobile robots**



# DIFFERENTIAL DRIVE KINEMATICS

- Well-fit for **smaller mobile robots**
- Usually have one or two castor wheels



# DIFFERENTIAL DRIVE KINEMATICS

- Well-fit for **smaller mobile robots**
- Usually have one or two castor wheels
- **Velocity of each wheel control** separately



# DIFFERENTIAL DRIVE KINEMATICS

- Well-fit for **smaller mobile robots**
- Usually have one or two castor wheels
- **Velocity of each wheel control** separately
- According to Fig. 11,



# DIFFERENTIAL DRIVE KINEMATICS

- Well-fit for **smaller mobile robots**
- Usually have one or two castor wheels
- **Velocity of each wheel control** separately
- According to Fig. 11,
  - ▶ Terms  $v_R(t)$ ,  $v_L(t)$ , denoted velocity of right and left wheels, respectively

# DIFFERENTIAL DRIVE KINEMATICS

- Well-fit for **smaller mobile robots**
- Usually have one or two castor wheels
- **Velocity of each wheel control** separately
- According to Fig. 11,
  - ▶ Terms  $v_R(t)$ ,  $v_L(t)$ , denoted velocity of right and left wheels, respectively
  - ▶ Wheel radius  $r$ , distance between wheels  $L$ , and term  $R(t)$  depicts the vehicle's instantaneous radios (ICR). **Angular velocity** is the **same for both left and right wheels around the ICR**.



# DIFFERENTIAL DRIVE KINEMATICS

## ■ Tangential velocity

$$\mathbf{v}(t) = \omega(t)R(t) = \frac{\mathbf{v}_R(t) + \mathbf{v}_L(t)}{2} \quad (1)$$

, where  $\omega = \mathbf{v}_L(t)/(R(t) - L/2) = \mathbf{v}_R(t)/(R(t) + L/2)$ . Hence,  $\omega$  and  $R(t)$  can be determined as follows:

$$\begin{aligned} \omega(t) &= \frac{\mathbf{v}_R(t) - \mathbf{v}_L(t)}{L} \\ R(t) &= \frac{L}{2} \frac{\mathbf{v}_R(t) + \mathbf{v}_L(t)}{\mathbf{v}_R(t) - \mathbf{v}_L(t)} \end{aligned} \quad (2)$$



# DIFFERENTIAL DRIVE KINEMATICS

## ■ Tangential velocity

$$\mathbf{v}(t) = \omega(t)R(t) = \frac{\mathbf{v}_R(t) + \mathbf{v}_L(t)}{2} \quad (1)$$

, where  $\omega = \mathbf{v}_L(t)/(R(t) - L/2) = \mathbf{v}_R(t)/(R(t) + L/2)$ . Hence,  $\omega$  and  $R(t)$  can be determined as follows:

$$\begin{aligned} \omega(t) &= \frac{\mathbf{v}_R(t) - \mathbf{v}_L(t)}{L} \\ R(t) &= \frac{L}{2} \frac{\mathbf{v}_R(t) + \mathbf{v}_L(t)}{\mathbf{v}_R(t) - \mathbf{v}_L(t)} \end{aligned} \quad (2)$$

## ■ Wheels tangential velocities (estimated **relative to the center of the respective wheel**)

$$\mathbf{v}_L = r\omega_L(t), \quad \mathbf{v}_R = r\omega_R(t) \quad (3)$$

# DIFFERENTIAL DRIVE KINEMATICS

## ■ Internal robot kinematics

$$\begin{bmatrix} \dot{x}_m(t) \\ \dot{y}_m(t) \\ \dot{\Phi}(t) \end{bmatrix} = \begin{bmatrix} v_{X_m}(t) \\ v_{Y_m}(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ -r/L & r/L \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix} \quad (4)$$

, where  $\omega_L(t)$  and  $\omega_R(t)$  are the control variables



# DIFFERENTIAL DRIVE KINEMATICS

## ■ Internal robot kinematics

$$\begin{bmatrix} \dot{x}_m(t) \\ \dot{y}_m(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} v_{X_m}(t) \\ v_{Y_m}(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ -r/L & r/L \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix} \quad (4)$$

, where  $\omega_L(t)$  and  $\omega_R(t)$  are the control variables

## ■ External robot kinematics

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} \cos(\phi(t)) & 0 \\ \sin(\phi(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}(t) \\ \omega(t) \end{bmatrix} \quad (5)$$



# DIFFERENTIAL DRIVE KINEMATICS

## ■ Internal robot kinematics

$$\begin{bmatrix} \dot{x}_m(t) \\ \dot{y}_m(t) \\ \dot{\Phi}(t) \end{bmatrix} = \begin{bmatrix} v_{X_m}(t) \\ v_{Y_m}(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ -r/L & r/L \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix} \quad (4)$$

, where  $\omega_L(t)$  and  $\omega_R(t)$  are the control variables

## ■ External robot kinematics

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\Phi}(t) \end{bmatrix} = \begin{bmatrix} \cos(\Phi(t)) & 0 \\ \sin(\Phi(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}(t) \\ \omega(t) \end{bmatrix} \quad (5)$$

## ■ Discrete time dynamics using Euler integration

$$\begin{aligned} x(k+1) &= x(k) + v(k)T_s \cos(\Phi(k)) \\ y(k+1) &= y(k) + v(k)T_s \sin(\Phi(k)) \\ \Phi(k+1) &= \Phi(k) + \omega(k)T_s \end{aligned} \quad (6)$$

, where discrete time instance  $t = kT_s$ ,  $k=0,1,2,\dots$ , for  $T_s$



# DIFFERENTIAL DRIVE KINEMATICS

- Forward robot kinematics (given a set of wheel speeds, determine robot velocity)

$$\begin{aligned}x(k+1) &= x(k) + v(k)T_s \cos(\Phi(k)) \\y(k+1) &= y(k) + v(k)T_s \sin(\Phi(k)) \\ \Phi(k+1) &= \Phi(k) + \omega(k)T_s\end{aligned}\tag{7}$$

, where discrete time instance  $t = kT_s$ ,  $k=0,1,2,\dots$ , for  $T_s$  sampling time

# DIFFERENTIAL DRIVE KINEMATICS

- Forward robot kinematics (given a set of wheel speeds, determine robot velocity)

$$\begin{aligned}x(k+1) &= x(k) + v(k)T_s \cos(\Phi(k)) \\y(k+1) &= y(k) + v(k)T_s \sin(\Phi(k)) \\ \Phi(k+1) &= \Phi(k) + \omega(k)T_s\end{aligned}\tag{7}$$

, where discrete time instance  $t = kT_s$ ,  $k=0,1,2,\dots$ , for  $T_s$  sampling time

- We can also try trapezoidal numerical integration for better approximation

$$\begin{aligned}x(k+1) &= x(k) + v(k)T_s \cos(\Phi(k) + \omega(k)T_s/2) \\y(k+1) &= y(k) + v(k)T_s \sin(\Phi(k) + \omega(k)T_s/2) \\ \Phi(k+1) &= \Phi(k) + \omega(k)T_s\end{aligned}\tag{8}$$



# DIFFERENTIAL DRIVE KINEMATICS

- Inverse robot kinematics (given desired robot velocity, determine corresponding wheel velocities)



# DIFFERENTIAL DRIVE KINEMATICS

- Inverse robot kinematics (given desired robot velocity, determine corresponding wheel velocities)
  - ▶ The **most challenging case compared to direct or forward kinematics**



# DIFFERENTIAL DRIVE KINEMATICS

- Inverse robot kinematics (given desired robot velocity, determine corresponding wheel velocities)
  - ▶ The **most challenging case compared to direct or forward kinematics**
  - ▶ Given the target pose **how many possible ways to get there?**



# DIFFERENTIAL DRIVE KINEMATICS

- Inverse robot kinematics (given desired robot velocity, determine corresponding wheel velocities)
  - ▶ The **most challenging case compared to direct or forward kinematics**
  - ▶ Given the target pose **how many possible ways to get there?**
  - ▶ What if the **robot** goes can perform only **two types of motions: forward and rotations**

$$\begin{aligned}\mathbf{v}_R = \mathbf{v}_L = \mathbf{v}_R, \omega(t) = 0, \mathbf{v}(t) = \mathbf{v}_R // \text{forward} \\ \mathbf{v}_R = -\mathbf{v}_L = \mathbf{v}_R, \omega(t) = 2\mathbf{v}_R/L, \mathbf{v}(t) = 0 // \text{rotation}\end{aligned}\tag{9}$$

# DIFFERENTIAL DRIVE KINEMATICS

- Inverse robot kinematics (given desired robot velocity, determine corresponding wheel velocities)

# DIFFERENTIAL DRIVE KINEMATICS

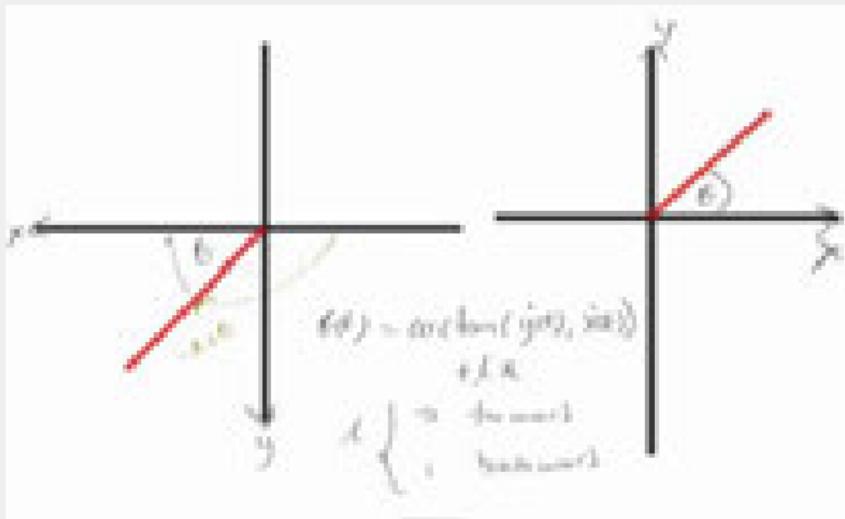
- Inverse robot kinematics (given desired robot velocity, determine corresponding wheel velocities)
  - ▶ If there is a disturbance in the trajectory and know the desired pose at time  $t$ , i.e.,  $x(t), y(t)$

$$\begin{aligned}\mathbf{v}(t) &= \pm \sqrt{\dot{x}^2(t) + \dot{y}^2(t)} // + \text{forward and - reverse} \\ \Phi(t) &= \arctan 2(\dot{y}(t), \dot{x}(t)) + l\pi, \quad l \in \{0, 1\} \\ &\quad // 0 \text{ forward and } 1 \text{ reverse} \\ \omega(t) &= \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{\dot{x}^2(t) + \dot{y}^2(t)} = v(t)k(t)\end{aligned}\tag{10}$$

, where  $k(t)$  is the **path curvature** and  $\omega(t) = \dot{\Phi}(t)$



# DIFFERENTIAL DRIVE KINEMATICS



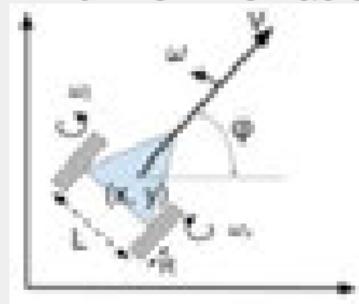
# DEFINE MOBILE ROBOTS WITH KINEMATIC CONSTRAINTS

## Unicycle kinematics



$$\begin{cases} \dot{x} = v \cos(\phi) = r \cos(\phi) \dot{\theta} \\ \dot{y} = v \sin(\phi) = r \sin(\phi) \dot{\theta} \\ \dot{\phi} = \omega \end{cases}$$

## Diffdrive kinematics



$$\begin{cases} \dot{x} = \frac{1}{2}(v_r + v_l) \cos(\phi) \\ \dot{y} = \frac{1}{2}(v_r + v_l) \sin(\phi) \\ \dot{\phi} = \frac{1}{L}(v_r - v_l) \end{cases}$$

After considering these listed models,

$$v_r = \frac{2v + \omega L}{2}, v_l = \frac{2v - \omega L}{2}$$



# DEFINE MOBILE ROBOTS WITH KINEMATIC CONSTRAINTS

- The **unicycle** and **differential drive** models share the generalized control inputs:  $v$  **vehicle speed** and  $\omega$  **vehicle angular velocity**

<https://nl.mathworks.com/help/robotics/ref/ackermannkinematics.html>



# DEFINE MOBILE ROBOTS WITH KINEMATIC CONSTRAINTS

- The **unicycle** and **differential drive** models share the generalized control inputs:  $v$  **vehicle speed** and  $\omega$  **vehicle angular velocity**
- **Unicycle Kinematic Model**  
The **simplest** way to represent **mobile robot vehicle kinematics** is with a unicycle model, which has **a wheel speed set by a rotation about a central axle** and can pivot about its z-axis. Both the **differential-drive** and **bicycle kinematic models reduce** down to **unicycle kinematics** when inputs are provided as vehicle speed and vehicle heading rate and **other constraints are not considered**.

<https://nl.mathworks.com/help/robotics/ref/ackermannkinematics.html>



# DEFINE MOBILE ROBOTS WITH KINEMATIC CONSTRAINTS

## ■ Differential-Drive Kinematic Model

**uses a rear driving axle to control both vehicle speed and heading rate.** The wheels on the driving axle can **spin in both directions.**

<https://nl.mathworks.com/help/robotics/ref/ackermannkinematics.html>



# DEFINE MOBILE ROBOTS WITH KINEMATIC CONSTRAINTS

## ■ Differential-Drive Kinematic Model

uses a **rear driving axle** to control both vehicle speed and heading rate. The wheels on the driving axle can **spin in both directions**.

## ■ Bicycle Kinematic Model

treats the robot as a **car-like model** with two axles: a **rear driving axle**, and a **front axle that turns about the z-axis**. The bicycle model assumes that wheels on each axle can be modelled as a single, centred wheel and that the front wheel heading can be directly set, like a bicycle.

<https://nl.mathworks.com/help/robotics/ref/ackermannkinematics.html>



# DEFINE MOBILE ROBOTS WITH KINEMATIC CONSTRAINTS

## ■ Ackermann Kinematic Model

is a modified **car-like model** that assumes Ackermann steering. In most car-like vehicles, the **front wheels do not turn about the same axis**, but instead, **turn on slightly different axes to ensure that they ride on concentric circles about the centre of the vehicle's turn**. This **difference** in turning angle is called **Ackermann steering** and is typically enforced by a mechanism in actual cars. From a vehicle and wheel kinematics standpoint, it can be enforced by treating the steering angle as a rated input.

<https://nl.mathworks.com/help/robotics/ref/ackermannkinematics.html>



# WHEELED MOBILE SYSTEM CONTROL

- Can navigate from a start pose to a goal pose by classical control, where intermediate state trajectory is not prescribed or reference trajectory tracking



# WHEELED MOBILE SYSTEM CONTROL

- Can navigate from a start pose to a goal pose by classical control, where intermediate state trajectory is not prescribed or reference trajectory tracking
- **Nonholonomic constraints** need to be considered, in such occasions, the controller is twofold: **feedforward** control and **feedback** control, namely **two-degree-of-freedom control**.



# WHEELED MOBILE SYSTEM CONTROL

- Can navigate from a start pose to a goal pose by classical control, where intermediate state trajectory is not prescribed or reference trajectory tracking
- **Nonholonomic constraints** need to be considered, in such occasions, the controller is twofold: **feedforward** control and **feedback** control, namely **two-degree-of-freedom control**.
- **Open-loop** control: **feedforward** control is calculated from the reference trajectory and those control actions are fed to the system



# WHEELED MOBILE SYSTEM CONTROL

- Can navigate from a start pose to a goal pose by classical control, where intermediate state trajectory is not prescribed or reference trajectory tracking
- **Nonholonomic constraints** need to be considered, in such occasions, the controller is twofold: **feedforward** control and **feedback** control, namely **two-degree-of-freedom control**.
- **Open-loop** control: **feedforward** control is calculated from the reference trajectory and those control actions are fed to the system
- However, **feedforward** control is **not practical as it is not robust to disturbance**, feedback needs to be applied



# WHEELED MOBILE SYSTEM CONTROL

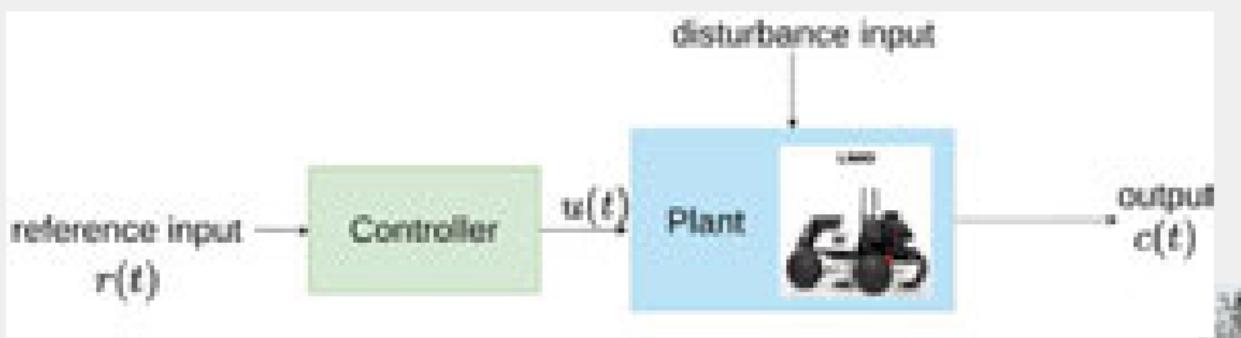
- Can navigate from a start pose to a goal pose by classical control, where intermediate state trajectory is not prescribed or reference trajectory tracking
- **Nonholonomic constraints** need to be considered, in such occasions, the controller is twofold: **feedforward** control and **feedback** control, namely **two-degree-of-freedom control**.
- **Open-loop** control: **feedforward** control is calculated from the reference trajectory and those control actions are fed to the system
- However, **feedforward** control is **not practical as it is not robust to disturbance**, feedback needs to be applied
- Wheeled mobile robots are dynamic. Thus, the motion controlling system has to incorporate the dynamics of the system, in general, which systems are designed as **cascade control schemes**: **outer controller** for velocity control and **inner controller** to handle torque, force, etc.



# OPEN-LOOP AND CLOSED-LOOP CONTROL

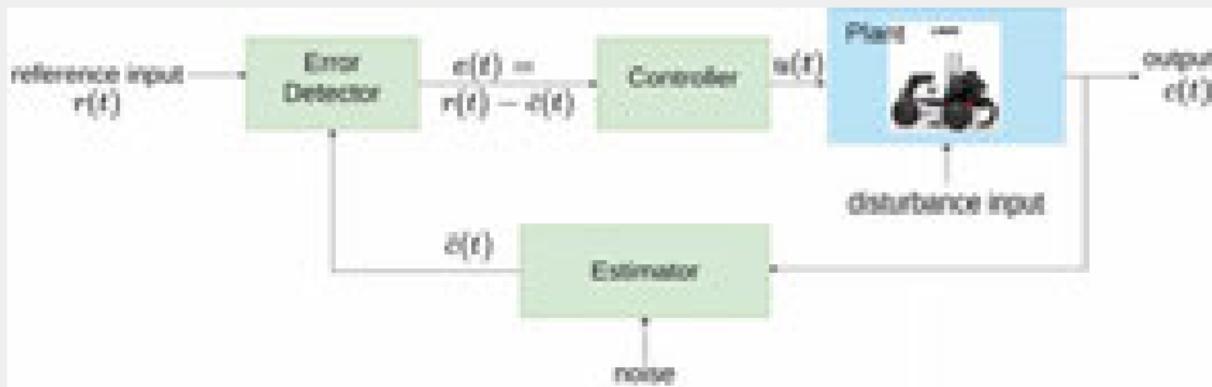
**Open-loop control:** system output ( $c(t)$ ) does not affect the input control ( $u(t)$ )

- No feedback mechanism: no connection between the actual output of the plant and its control inputs
- Control inputs are independent of the actual output (or assuming the system will behave predictably)
- Examples: Washing machine, Toaster, Traffic lights



# OPEN-LOOP AND CLOSED-LOOP CONTROL

**Closed-loop control:** system output ( $c(t)$ ) does affect the input control ( $u(t)$ )



# OPEN-LOOP AND CLOSED-LOOP CONTROL

**Closed-loop control:** system output ( $c(t)$ ) does affect the input control ( $u(t)$ )

- Feedback mechanism: the actual output of the plant and its control inputs are synchronized in a way to estimate the desired output
- Control inputs regulated based on feedback: the controller takes corrective residual control ( $r(t) - \bar{c}(t)$ ) to minimize any discrepancies between the desired and actual output
- More accurate and stable: they can adapt to disturbances and maintain consistent performance.
- Examples: Temperature control in air conditioners, Cruise control in cars



## TASK 01

Let's try to control the differential drive robot

Consider you are given the following vehicle parameters:  
sampling period  $T_s = 0.033\text{s}$ , wheel radius  $r = 0.04\text{ m}$ , the distance  
between the wheels  $L = 0.08\text{ m}$

- For each specified case, determine the shape of the path traced by the robot using both analytical calculations and simulation.
  - ▶  $v(t) = 0.5\text{ m/s}$ ,  $\omega(t) = 0\text{ rad/s}$
  - ▶  $v(t) = 1\text{ m/s}$ ,  $\omega(t) = 2\text{ rad/s}$
  - ▶  $v(t) = 0\text{ m/s}$ ,  $\omega(t) = 2\text{ rad/s}$
  - ▶ wheels angular velocities are  $\omega(t)_L = 20\text{ rad/s}$  and  $\omega(t)_R = 18\text{ rad/s}$



# TARGET (REFERENCE) POSE CONTROL

- Pose = position + orientation



## TARGET (REFERENCE) POSE CONTROL

- Pose = position + orientation
- Feasible path, which can be **optimal**, should satisfy the **kinematic, dynamic, and other constraints including disturbances**, appropriately



# TARGET (REFERENCE) POSE CONTROL

- Pose = position + orientation
- Feasible path, which can be **optimal**, should satisfy the **kinematic, dynamic, and other constraints including disturbances**, appropriately
- Reference pose control, in general, is performed as two sub-controlling tasks: **orientation control** and **forward-motion control**. However, **these are interconnected** with each other



# TARGET (REFERENCE) ORIENTATION CONTROL

- **Orientation** control **cannot be performed** independently from the **forward-motion control**



# TARGET (REFERENCE) ORIENTATION CONTROL

- **Orientation control cannot be performed independently from the forward-motion control**
- Let robot orientation  $\Phi(t)$ , at time  $t$ , and reference orientation is  $\Phi_{ref}(t)$

$$e_{\Phi}(t) = \Phi_{ref}(t) - \Phi(t) \quad (11)$$



# TARGET (REFERENCE) ORIENTATION CONTROL

- **Orientation control cannot be performed** independently from the **forward-motion control**
- Let robot orientation  $\Phi(t)$ , at time  $t$ , and reference orientation is  $\Phi_{ref}(t)$

$$e_\Phi(t) = \Phi_{ref}(t) - \Phi(t) \quad (11)$$

- **How fast can we drive the control error** to zero? It depends on additional factors: energy consumption, actuator load, and robustness



# TARGET (REFERENCE) ORIENTATION CONTROL

- **Orientation control cannot be performed** independently from the **forward-motion control**
- Let robot orientation  $\Phi(t)$ , at time  $t$ , and reference orientation is  $\Phi_{ref}(t)$

$$e_\Phi(t) = \Phi_{ref}(t) - \Phi(t) \quad (11)$$

- **How fast can we drive the control error** to zero? It depends on additional factors: energy consumption, actuator load, and robustness
- Since  $\dot{\Phi}(t) = \omega(t)$  is the input for control for diff drive, a proportional controller is able to drive control error of an integral process to 0

$$\omega(t) = K(\Phi_{ref} - \Phi(t)) \quad (12)$$

, where  $K$  is an arbitrary positive constant

# TARGET (REFERENCE) ORIENTATION CONTROL

- $\dot{\phi}(t) = \frac{v_r}{d} \tan(\alpha(t))$  is the input for control for Ackermann drive. The control variable is  $\alpha$ , which can be chosen proportional to the orientation error:

$$\begin{aligned}\alpha(t) &= K (\Phi_{ref}(t) - \Phi(t)) \\ \dot{\phi}(t) &= \frac{v_r}{d} \tan(K (\Phi_{ref}(t) - \Phi(t)))\end{aligned}\tag{13}$$



# TARGET (REFERENCE) ORIENTATION CONTROL

- $\dot{\phi}(t) = \frac{v_r}{d} \tan(\alpha(t))$  is the input for control for Ackermann drive. The control variable is  $\alpha$ , which can be chosen proportional to the orientation error:

$$\begin{aligned}\alpha(t) &= K (\Phi_{ref}(t) - \Phi(t)) \\ \dot{\phi}(t) &= \frac{v_r}{d} \tan(K (\Phi_{ref}(t) - \Phi(t)))\end{aligned}\tag{13}$$

- **For small angle** and constant velocity of rear wheels  $v_r(t) = V$ , a linear approximation can be obtained,

$$\dot{\phi}(t) = \frac{V}{d} (K (\Phi_{ref}(t) - \Phi(t)))\tag{14}$$



## TARGET (REFERENCE) FORWARD-MOTION CONTROL

- Forward-motion control is inevitably interconnected with orientation control, i.e., **forward-motion alone can not drive to goal pose** without **correct orientation**

$$\mathbf{v(t)} = K \sqrt{((x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2)} \quad (15)$$

# TARGET (REFERENCE) FORWARD-MOTION CONTROL

- Forward-motion control is inevitably interconnected with orientation control, i.e., **forward-motion alone can not drive to goal pose** without **correct orientation**

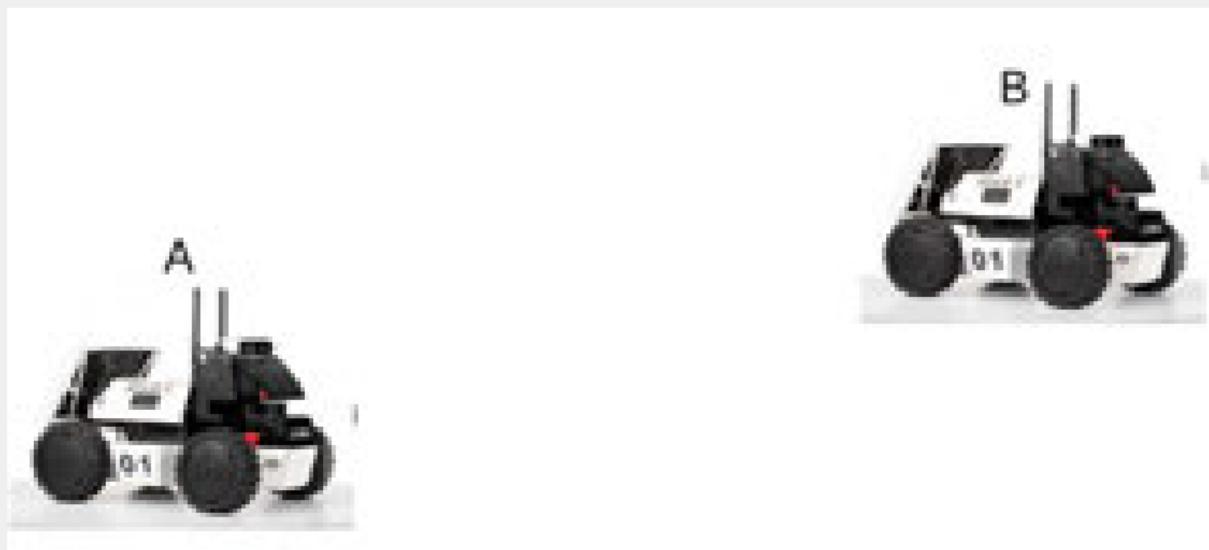
$$\mathbf{v}(t) = K \sqrt{((x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2)} \quad (15)$$

- However,  $\mathbf{v}(t)$  has a maximum limit, which is due to actuator limitations driving surface conditions. On the other hand, when the robot gets **closer to the goal**, it might try to **overtake the reference pose**, which **eventually leads to acceleration**, which is not desired



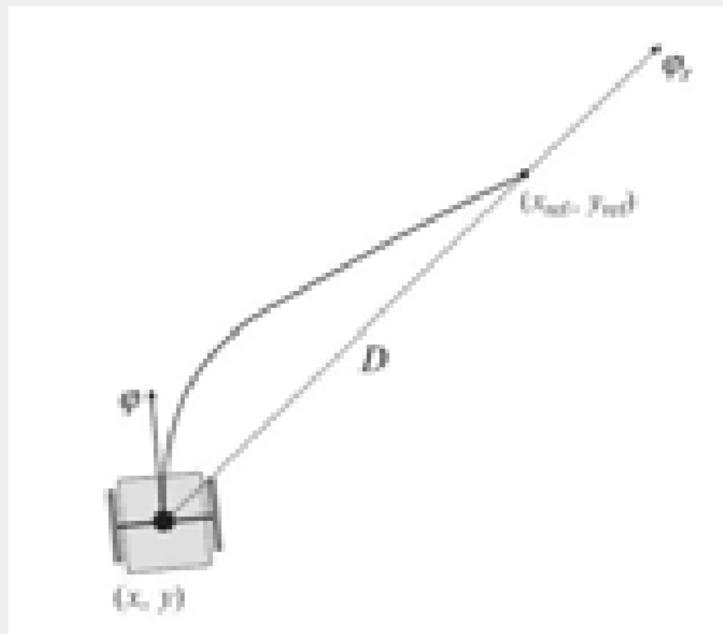
## CONTROL TO REFERENCE POSE

What is the optimal way to navigate the robot from pose A to pose B?



# CONTROL TO REFERENCE POSE

# CONTROL TO REFERENCE POSE



## CONTROL TO REFERENCE POSE

- It is required to reach the target position where the final orientation is not prescribed, hence the direction of the reference position

$$\Phi_r(t) = \arctan \frac{y_{ref} - y(t)}{x_{ref} - x(t)}, \omega(t) = K_1(\Phi_r(t) - \Phi(t))$$
$$\mathbf{v(t)} = K \sqrt{((x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2)}$$
(16)

## CONTROL TO REFERENCE POSE

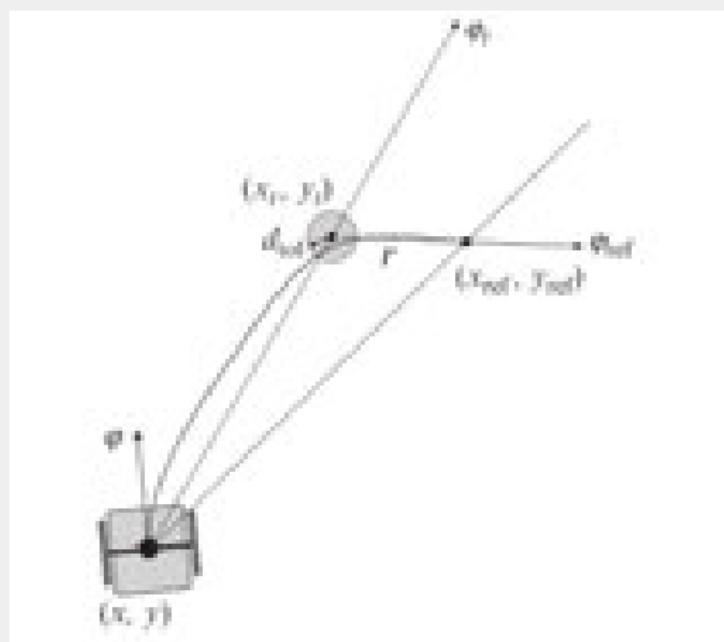
- It is required to reach the target position where the final orientation is not prescribed, hence the direction of the reference position

$$\begin{aligned}\Phi_r(t) &= \arctan \frac{y_{ref} - y(t)}{x_{ref} - x(t)}, \quad \omega(t) = K_1(\Phi_r(t) - \Phi(t)) \\ \mathbf{v(t)} &= K \sqrt{((x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2)}\end{aligned}\tag{16}$$

- What will happen when the orientation error abruptly changes ( $\pm 180$  degrees)? if the absolute value of orientation error exceeds 90 degrees, orientation error increases or decreases by 180 degrees

$$\begin{aligned}e_\Phi(t) &= \Phi_{ref}(t) - \Phi(t), \quad \omega(t) = K_1 \arctan(\tan(e_\Phi(t))) \\ \mathbf{v(t)} &= K \sqrt{((x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2)} \cdot \text{sgn}(\cos(e_\Phi(t)))\end{aligned}\tag{17}$$

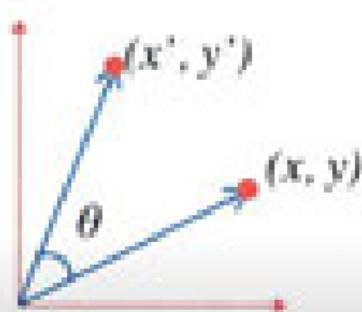
# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE POINT



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE POINT

## Rotation by a counterclockwise angle

### 2D Rotation



$$\begin{aligned}x' &= x \cos(\theta) - y \sin(\theta) \\y' &= x \sin(\theta) + y \cos(\theta)\end{aligned}$$

$$\mathbf{M} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE POINT

- Idea is to shape in a way that the correct orientation is obtained



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE POINT

- Idea is to shape in a way that the correct orientation is obtained
- Intermediate point is determined by

$$\begin{aligned}x_t &= x_{ref} - r \cos(\Phi_{ref}) \\y_t &= y_{ref} - r \sin(\Phi_{ref})\end{aligned}\tag{18}$$

, where the distance from the reference point to the intermediate point denoted  $r$



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE POINT

- Idea is to shape in a way that the correct orientation is obtained
- Intermediate point is determined by

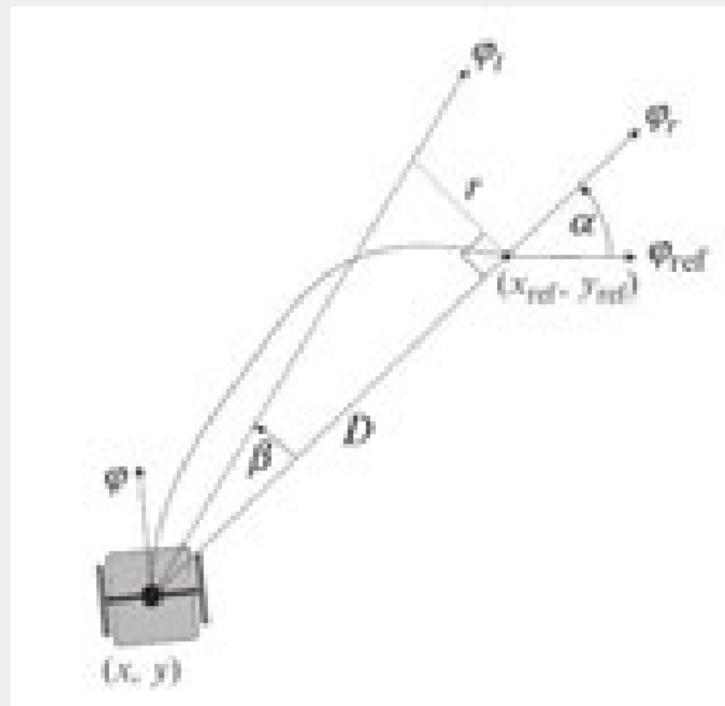
$$\begin{aligned}x_t &= x_{ref} - r \cos(\Phi_{ref}) \\y_t &= y_{ref} - r \sin(\Phi_{ref})\end{aligned}\tag{18}$$

, where the distance from the reference point to the intermediate point denoted  $r$

- If distance between current and intermediate position  $\sqrt{(x - x_t)^2 + (y - y_t)^2} < d_{tol}$ , where term  $d_{tol}$  depicts threshold, robot starts controlling to reference point



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE DIRECTION



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE DIRECTION

- Distance between current pose and target pose

$$D = \sqrt{((x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2)} \quad (19)$$



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE DIRECTION

- Distance between current pose and target pose

$$D = \sqrt{((x_{ref}(t) - x(t))^2 + (y_{ref}(t) - y(t))^2)} \quad (19)$$

- Let the perpendicular distance to D from the reference point be r. Then,

$$\begin{aligned} \alpha(t) &= \Phi_r(t) - \Phi_{ref} \\ \beta(t) &= \begin{cases} \arctan \frac{+r}{D} & \alpha(t) > 0 \\ -\arctan \frac{r}{D} & \text{otherwise} \end{cases} \end{aligned} \quad (20)$$

, where  $\alpha(t)$  and  $\beta(t)$  are always of the same sign unless  $\alpha = 0$



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE DIRECTION

- To define the control law, these facts have to consider:  $\alpha(t)$  reduces when approaching the target, however,  $\beta$  increases. Thus, there are two phases:

$$e_{\Phi}(t) = \Phi_r(t) - \Phi(t) + \begin{cases} \alpha(t) & |\alpha(t)| < |\beta(t)| \\ \beta(t) & \text{otherwise} \end{cases} \quad (21)$$
$$\omega(t) = K e_{\Phi}(t)$$



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE DIRECTION

- To define the control law, these facts have to consider:  $\alpha(t)$  reduces when approaching the target, however,  $\beta$  increases. Thus, there are two phases:

$$e_{\Phi}(t) = \Phi_r(t) - \Phi(t) + \begin{cases} \alpha(t) & |\alpha(t)| < |\beta(t)| \\ \beta(t) & \text{otherwise} \end{cases} \quad (21)$$
$$\omega(t) = K e_{\Phi}(t)$$

- In the first phase,  $|\alpha(t)|$  is large, and the robot's orientation is controlled toward the intermediate direction  $\Phi_t(t) = \Phi_r(t) + \beta(t)$ . When  $\alpha$  and  $\beta$  become the same, the current reference orientation switches to  $\Phi_t(t) = \Phi_r(t) + \alpha(t)$



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE DIRECTION

- To define the control law, these facts have to consider:  $\alpha(t)$  reduces when approaching the target, however,  $\beta$  increases. Thus, there are two phases:

$$e_{\Phi}(t) = \Phi_r(t) - \Phi(t) + \begin{cases} \alpha(t) & |\alpha(t)| < |\beta(t)| \\ \beta(t) & \text{otherwise} \end{cases} \quad (21)$$
$$\omega(t) = K e_{\Phi}(t)$$

- In the first phase,  $|\alpha(t)|$  is large, and the robot's orientation is controlled toward the intermediate direction  $\Phi_t(t) = \Phi_r(t) + \beta(t)$ . When  $\alpha$  and  $\beta$  become the same, the current reference orientation switches to  $\Phi_t(t) = \Phi_r(t) + \alpha(t)$
- $e_{\Phi}(t)$  is not reducing to zero but is always slightly shifted



# CONTROL TO REFERENCE POSE VIA AN INTERMEDIATE DIRECTION

- To define the control law, these facts have to consider:  $\alpha(t)$  reduces when approaching the target, however,  $\beta$  increases. Thus, there are two phases:

$$e_{\Phi}(t) = \Phi_r(t) - \Phi(t) + \begin{cases} \alpha(t) & |\alpha(t)| < |\beta(t)| \\ \beta(t) & \text{otherwise} \end{cases} \quad (21)$$
$$\omega(t) = K e_{\Phi}(t)$$

- In the first phase,  $|\alpha(t)|$  is large, and the robot's orientation is controlled toward the intermediate direction  $\Phi_t(t) = \Phi_r(t) + \beta(t)$ . When  $\alpha$  and  $\beta$  become the same, the current reference orientation switches to  $\Phi_t(t) = \Phi_r(t) + \alpha(t)$
- $e_{\Phi}(t)$  is not reducing to zero but is always slightly shifted
- Desired velocity is determined as  $\mathbf{v} = K_p D$ , where  $K_p \in \mathbb{R}^+$  is a constant



# REFERENCES

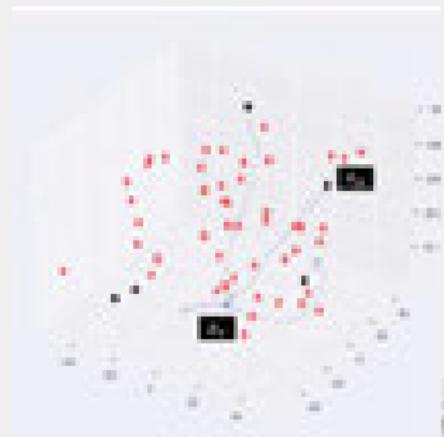
-  GREGOR KLANCAR, ANDREJ ZDESAR, SASO BLAZIC, AND IGOR SKRJANC.  
**WHEELED MOBILE ROBOTICS: FROM FUNDAMENTALS TOWARDS AUTONOMOUS SYSTEMS.**  
Butterworth-Heinemann, 2017.
-  ROLAND SIEGWART, ILLAH REZA NOURBAKHSH, AND DAVIDE SCARAMUZZA.  
**INTRODUCTION TO AUTONOMOUS MOBILE ROBOTS.**  
MIT press, 2011.
-  SEBASTIAN THRUN.  
**PROBABILISTIC ROBOTICS.**  
*Communications of the ACM*, 45(3):52–57, 2002.

# CMP3103    AUTONOMOUS    MOBILE ROBOTICS

## ROBOT BEHAVIOR

GEESARA KULATHUNGA

APRIL 29, 2024



# BEHAVIOR OF MOBILE ROBOTS

# CONTENTS

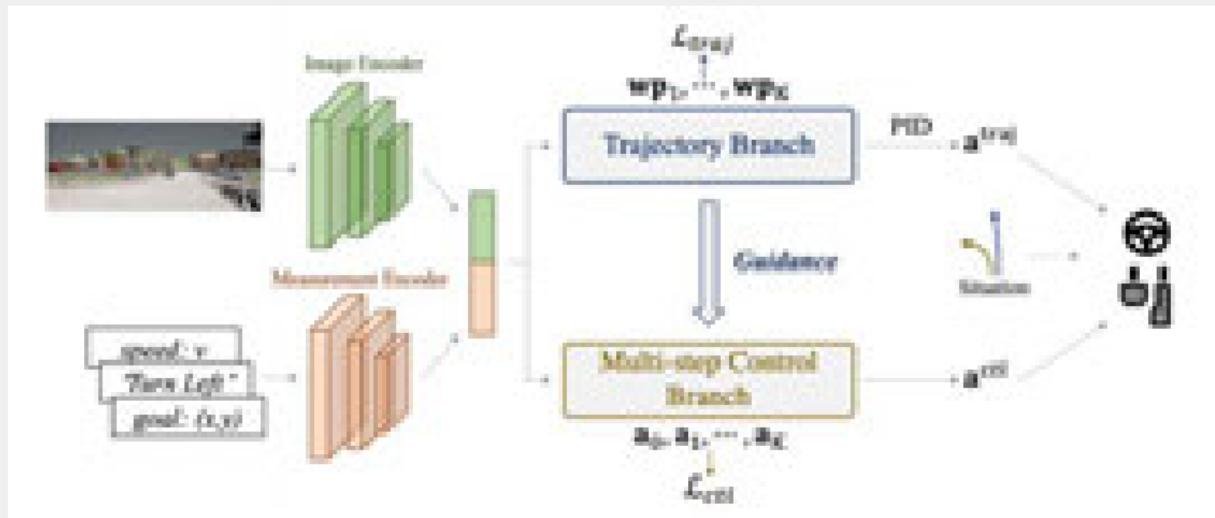
---

- Motivation
- Basic of robot behaviours
- Behaviours in Robotics
  - ▶ Reactive behaviour
  - ▶ Finite State Machine (FSM)
  - ▶ Behavior Tree (BT)



# MOTIVATION

## Trajectory-guided Control Prediction for End-to-end Autonomous Driving

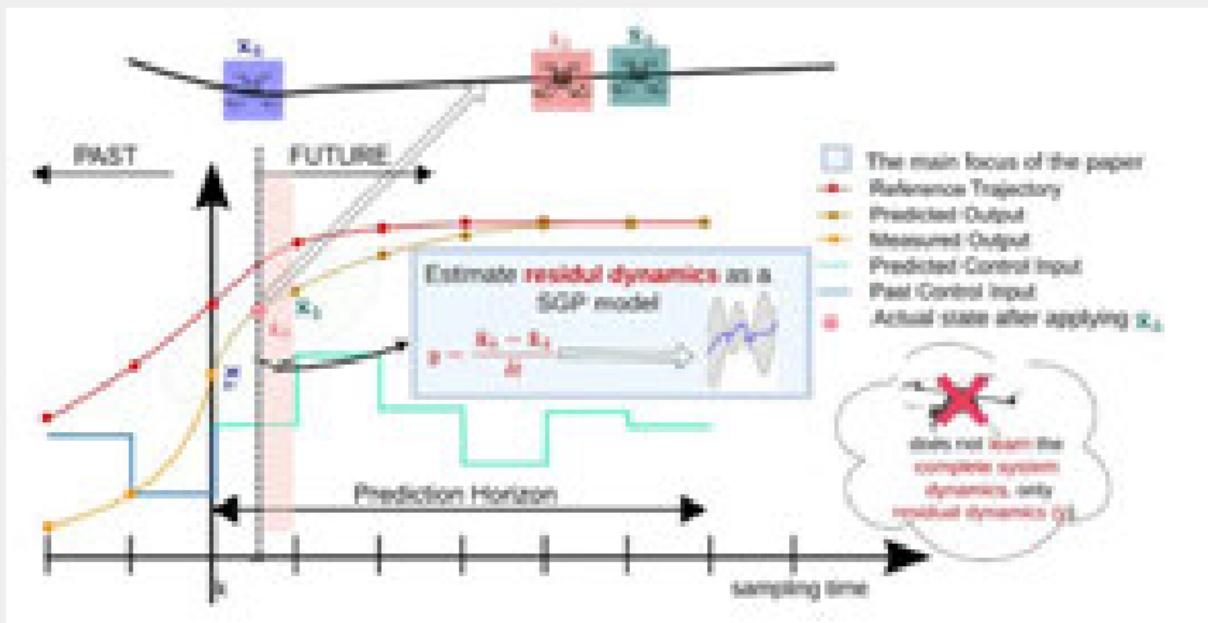


Trajectory-guided Control Prediction for End-to-end Autonomous Driving: A Simple yet Strong Baseline Penghao Wu\*,  
Xiaosong Jia\*, Li Chen\*, Junchi Yan, Hongyang Li, Yu Qiao



# MOTIVATION

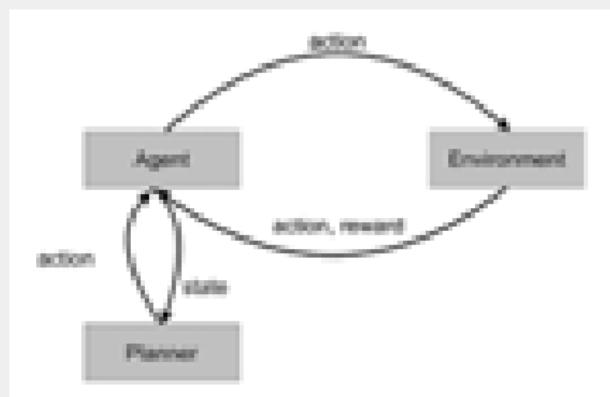
## Residual dynamics learning for trajectory tracking for multi-rotor aerial vehicles



Kulathunga, G., Hamed, H. & Klimchik, A. Residual dynamics learning for trajectory tracking for multi-rotor aerial vehicles. *Sci Rep* 14, 1858 (2024). <https://doi.org/10.1038/s41598-024-51822-0>

# MOTIVATION

## A Reinforcement Learning based Path Planning Approach in 3D Environment

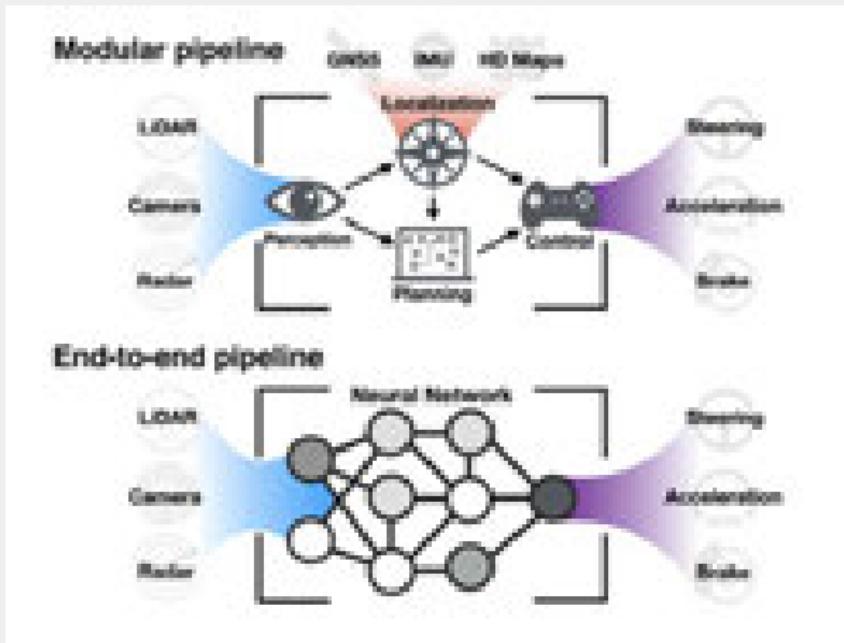


Kulathunga, Geesara. "A reinforcement learning based path planning approach in 3d environment." Procedia Computer Science 212 (2022): 152-160.



# BASIC OF ROBOT BEHAVIOURS

Sense -> Plan -> Act



A. Tampuu, T. Matiisen, M. Semikin, D. Fishman and N. Muhammad, "A Survey of End-to-End Driving: Architectures and Training Methods," in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 4, pp. 1364-1384, April 2022, doi: 10.1109/TNNLS.2020.3043505. keywords: Task analysis; Pipelines; Training; Neural networks; Autonomous vehicles; Roads; Automobiles; Autonomous driving; end-to-end; neural networks



# BASIC OF ROBOT BEHAVIOURS

- **Robot behavior** can include: motor control, driving, picking some objects
- **Sense -> Plan -> Act: Sequential decision-making** (observe, take action, observe, take action, ...)
- How to solve such problems?
  - ▶ Model-free (policy gradients, actor-critic methods, **Q-learning**) and model-based Reinforcement Learning (RL)
  - ▶ Imitation learning (behaviour cloning, inverse RL)
  - ▶ Multi-task and meta RL



# BEHAVIOURS IN ROBOTICS

Reactive behaviour occurs when the **action** is related only to the **occurrence of an event** and does not depend **on data stored in memory (state)**. [1]



[1]. Ben-Ari, M., Mondada, F. (2018). Reactive Behavior. In: Elements of Robotics. Springer, Cham.  
[https://doi.org/10.1007/978-3-319-62533-1\\_3](https://doi.org/10.1007/978-3-319-62533-1_3)

# REACTIVE CONTROL

**Reactive control**, also known as behaviour-based control, is a type of robot control architecture where **the robot's actions are directly determined by its current sensor readings, without the need for an internal world model or pre-planned path**.

- **Sensor-based**
- **Real-time**: The robot makes decisions and takes actions in real-time, without the need for pre-computation or planning.
- **Emergent behavior**: Complex behaviors can emerge from the interaction of simple rules, without the need for explicit programming.
- **Robust**: to changes in the environment, as they do not rely on a perfect model of the world.



The **Braitenberg Vehicles** are a fascinating thought experiment from the book "[Vehicles: Experiments in Synthetic Psychology\[1\]](#)" by Valentino Braitenberg. They serve as a minimalist approach to understanding how seemingly **intelligent behavior** can emerge from **very simple sensory-motor connections**.

## Concept:

- Simple robots with basic sensors (like light detectors) and motors.
- Sensors directly connect to motors, creating a "reflexive" response to stimuli.
- No complex internal processing or planning involved

<https://mitpress.mit.edu/9780262521123/vehicles/>

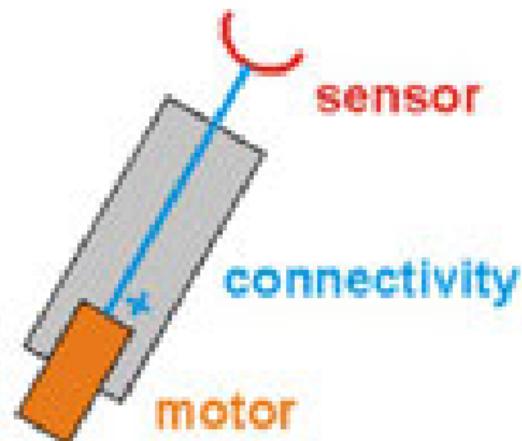


**Variety and Behaviors: Different vehicle configurations lead to diverse behaviours.** The **light** sensor was considered in the original experiments. However, sensors can be any, e.g., **light, temperature, and sound**

- Light follower or avoider: Attracted to light sources or repelled by light sources (**Vehicle 1**).
- Shadow follower: Reacts to edges and gradients of light, effectively following shadows (**Vehicle 2a and Vehicle 2b**).
- "Lover" and "fighter": Vehicles reacting to other vehicles with light sensors, exhibiting attraction or repulsion (**Vehicle 3a and Vehicle 3b**)



## Vehicle 1

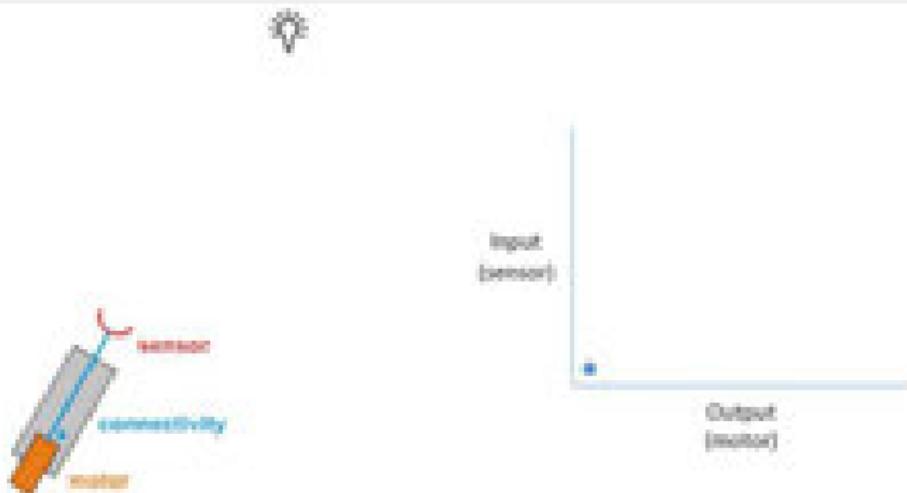


- One sensor, e.g., light, temperature, sound
- One motor
  - ▶ Directly driven by sensor
  - ▶ +/- indicates excitatory/inhibitory relationship
- **Motor speed is proportional to sensor reading**, e.g., greater light intensity == faster motor speed



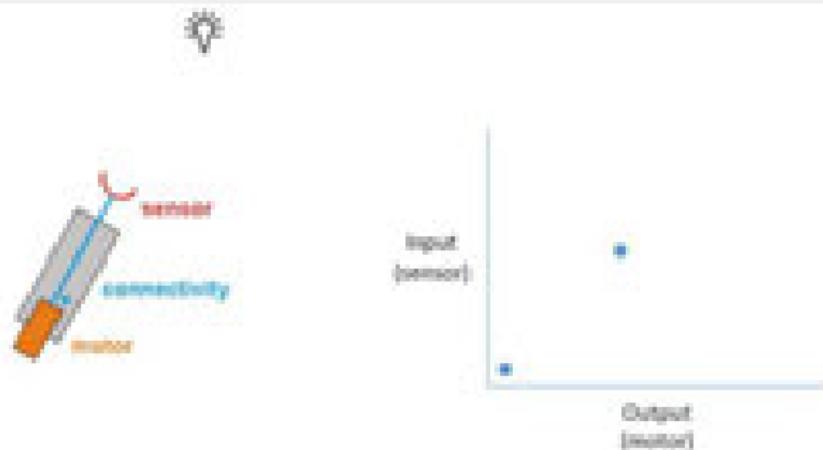
# BRAITENBERG VEHICLES

## Vehicle 1: Initial location



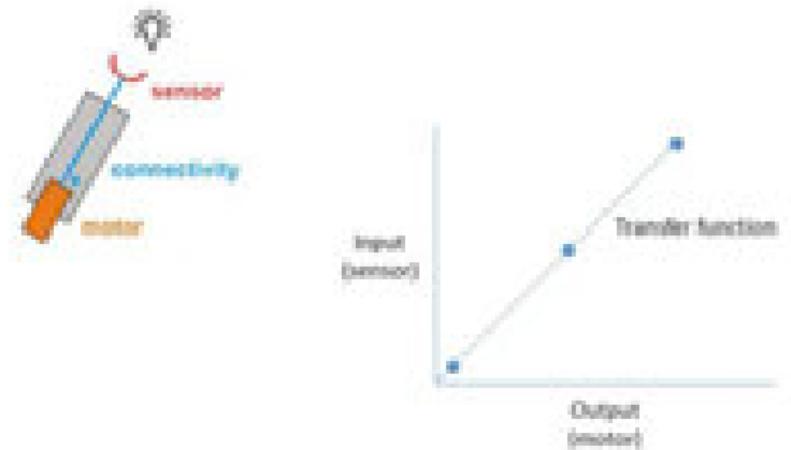
# BRAITENBERG VEHICLES

## Vehicle 1: Intermediate location



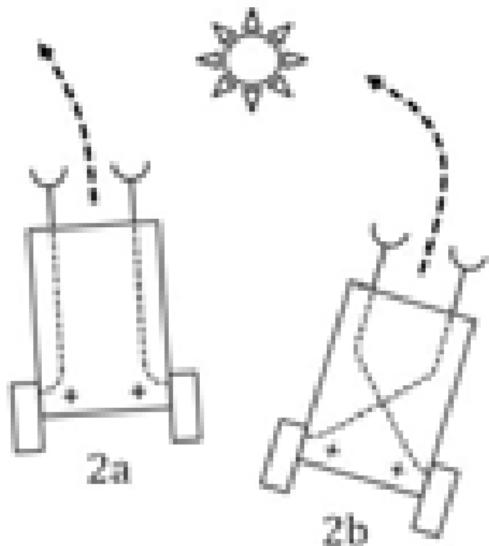
# BRAITENBERG VEHICLES

## Vehicle 1: Final location



# BRAITENBERG VEHICLES

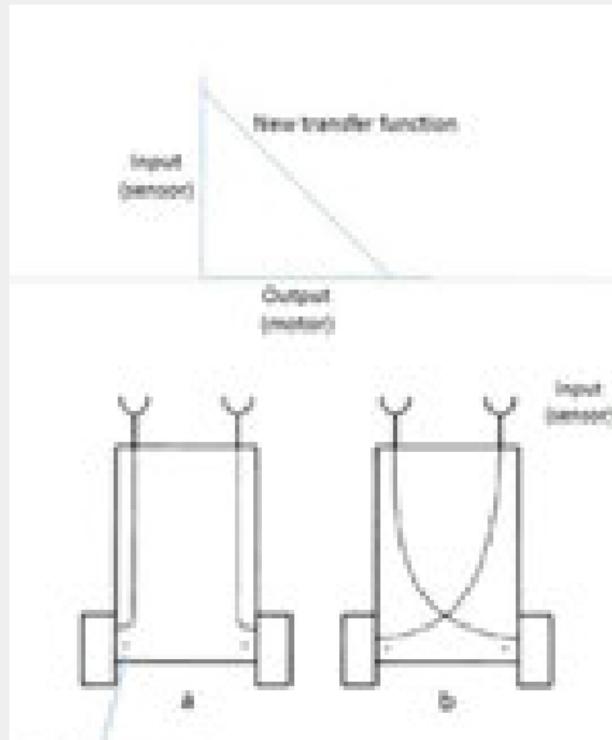
## Vehicles 2a and 2b



- Two sensors, two motors (directly coupled)
- Vehicle 2a – **Fear**, i.e., sensors linked to motors on same side
- Vehicle 2b – **Aggression**, i.e., sensors linked to motors on opposite side

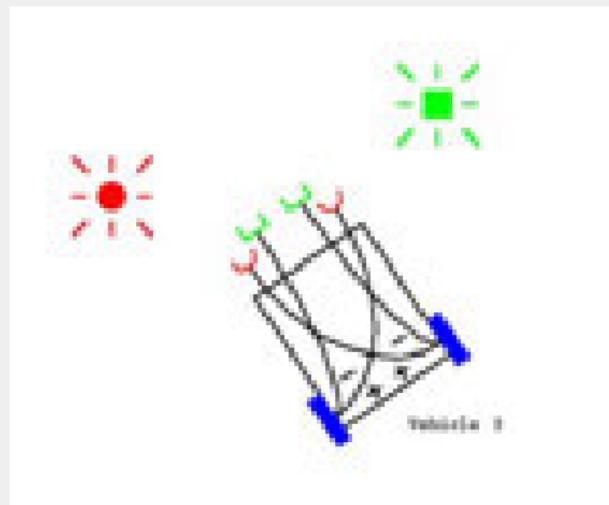


## Vehicles 3a and 3b



- Same as Vehicle 2, but with **inhibitory connections**
- Vehicle 3a – **Love**, i.e., sensors linked to motors on same side
- Vehicle 3b – **Explorer**, i.e., sensors linked to motors on opposite sides

## Multi-sensory vehicles



Increasingly complex behaviours can be created by adding

- New sensors with changing profiles, e.g., Non-linear/Digital
- New transfer functions, e.g., Non-linear/weighting
- Varying excitatory and inhibitory connectivity, e.g., goal-seeking vs obstacle avoidance



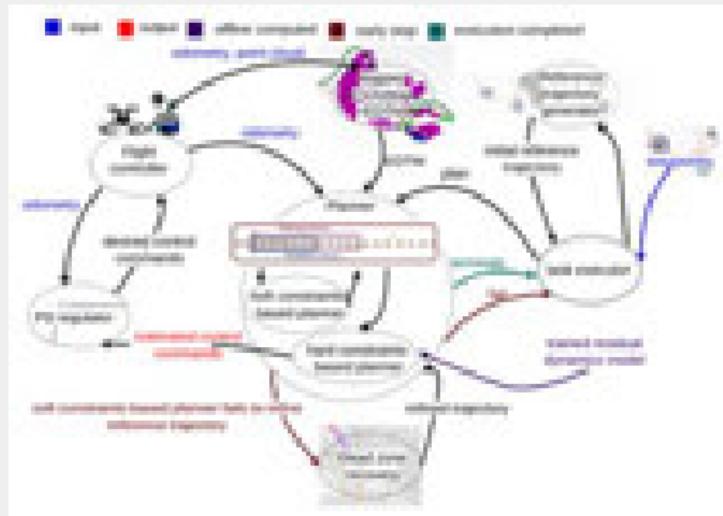
# BEHAVIOURS IN ROBOTICS

## Finite State Machine (FSM)

**States:** Each phase is a distinct state within the FSM.

**Transitions:** These connect states and define how the robot moves between them.

**Actions:** Within each state, the robot can perform specific actions



# BEHAVIOURS IN ROBOTICS

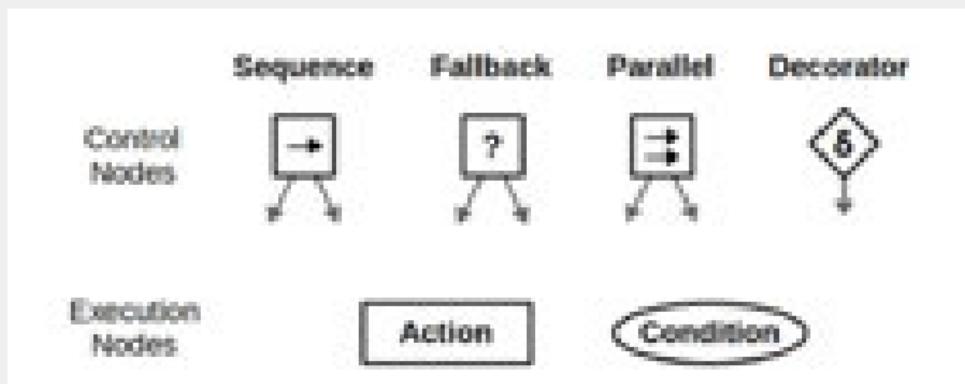
## Behavior Trees (BT)

**Nodes:** Represent tasks or conditions

**Root node:** Where the execution starts

**Leaf nodes:** Tasks the robot directly performs (e.g., move forward, turn left)

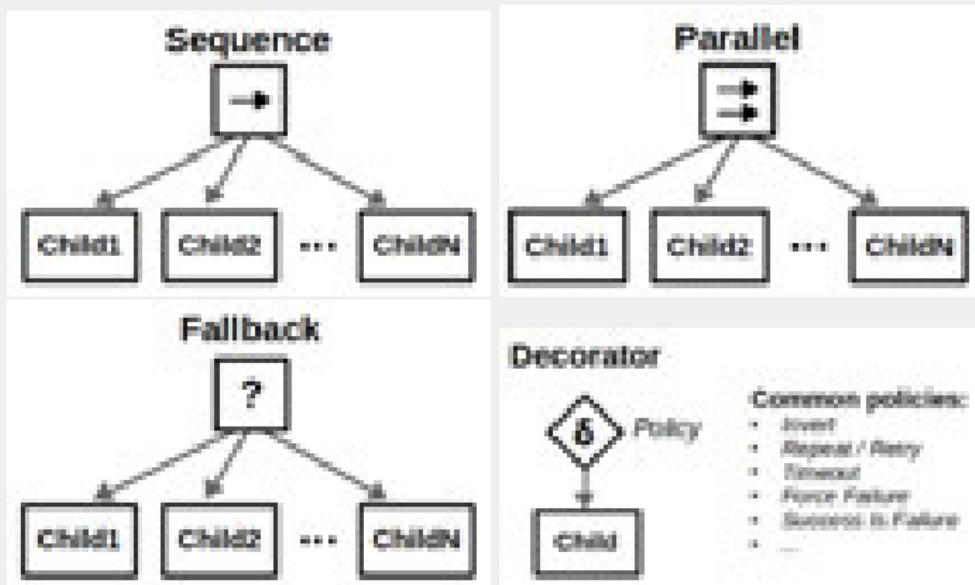
**Control flow nodes:** Decide how the tree gets traversed (e.g., sequence, repeat, priority)



[2]. Colledanchise, M., Ögren, P. (2018). Behavior trees in robotics and AI: An introduction. CRC Press.

# BEHAVIOURS IN ROBOTICS

Behavior tree's basic building blocks



# BEHAVIOURS IN ROBOTICS

## BTs vs. FSMs

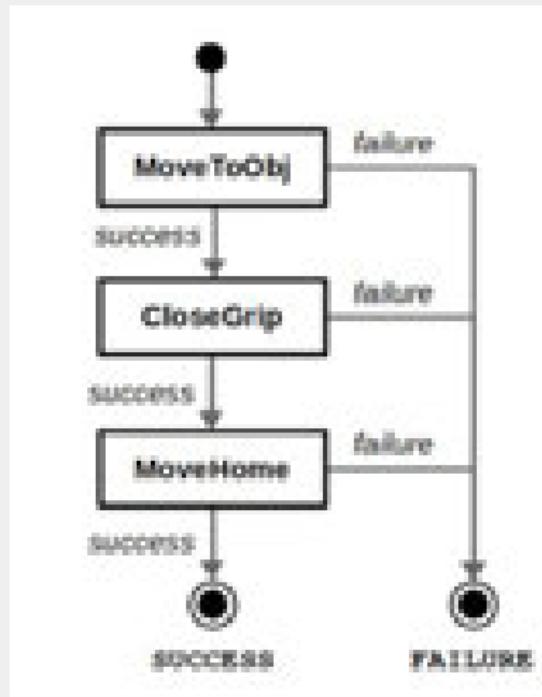
While both control robot behaviour, they have key differences:

- **Structure:** BTs are hierarchical, FSMs are state-based.
- **Complexity:** BTs handle complex tasks better, FSMs excel in simpler scenarios.
- **Dynamicity:** BTs adapt easier, FSMs provide predictable execution.



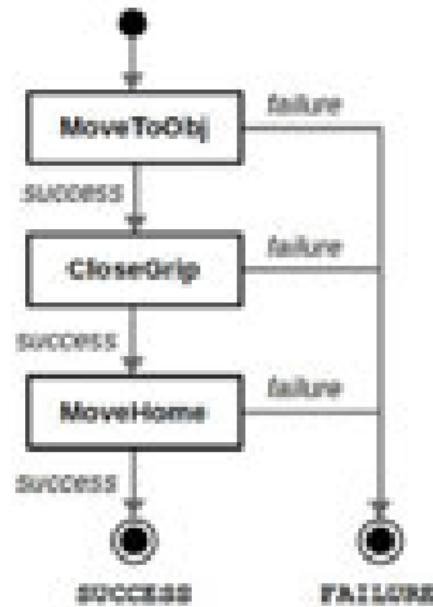
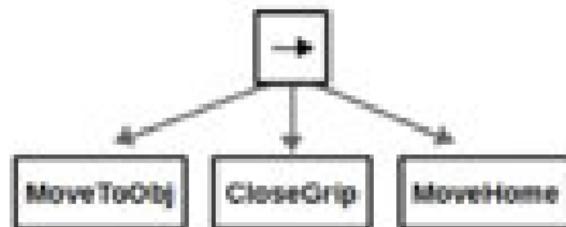
# BEHAVIOURS IN ROBOTICS

Can you convert this FSM into a BT?



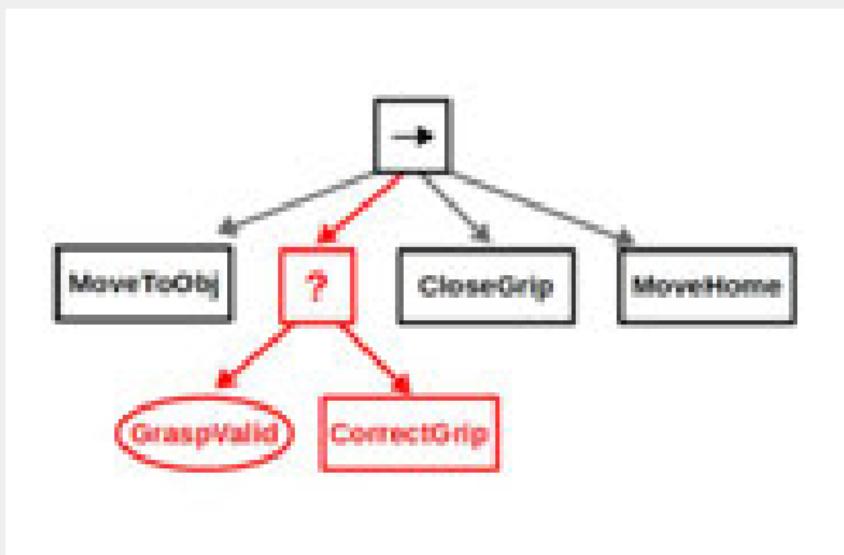
# BEHAVIOURS IN ROBOTICS

## State machine and behaviour tree



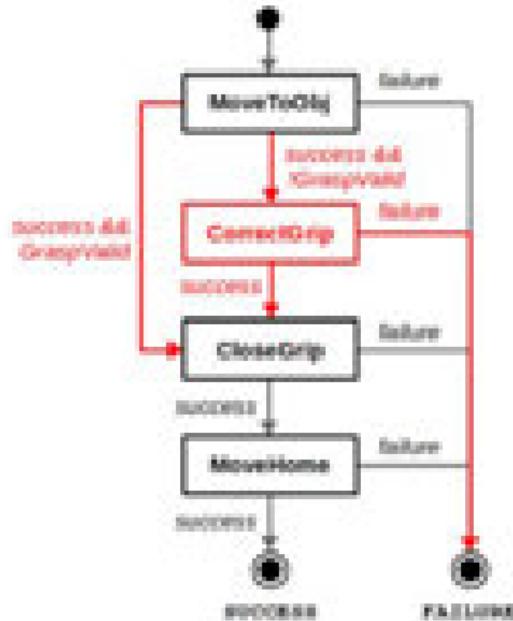
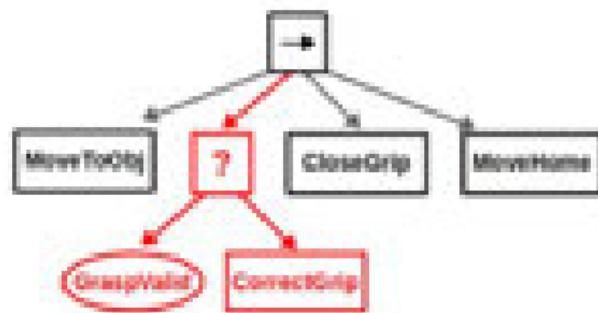
# BEHAVIOURS IN ROBOTICS

Let's try to extend existing BT and FTM?



# BEHAVIOURS IN ROBOTICS

## State machine and behaviour tree (adding new state or node)



# CMP3103M

# Autonomous Mobile Robotics

## Lecture 6: Navigation

• • •

Dr. Athanasios Polydoros

# Today

## Contents:

- Quiz
- Navigation Overview
- Global & Local Navigation
- Odometry
- Navigation Strategies

## Learning outcomes:

- Define Robot Navigation
- Describe various navigation types and strategies
- Compute robot odometry

# Quizz

Connect PollEveryhere



<https://pollev.com/athanasiospolydoros472>

## Reactive behaviours of robot require the definition of a goal/plan

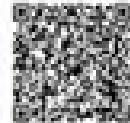
0

(A) True

0%

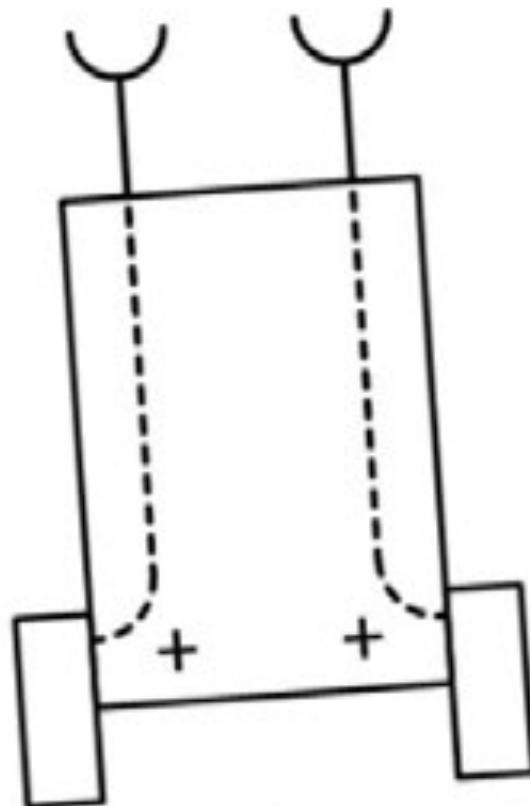
(B) False

0%



# What would be the behaviour of the illustrated Braitenberg vehicle?

0



Fear

0%

Aggression

0%

Love

0%

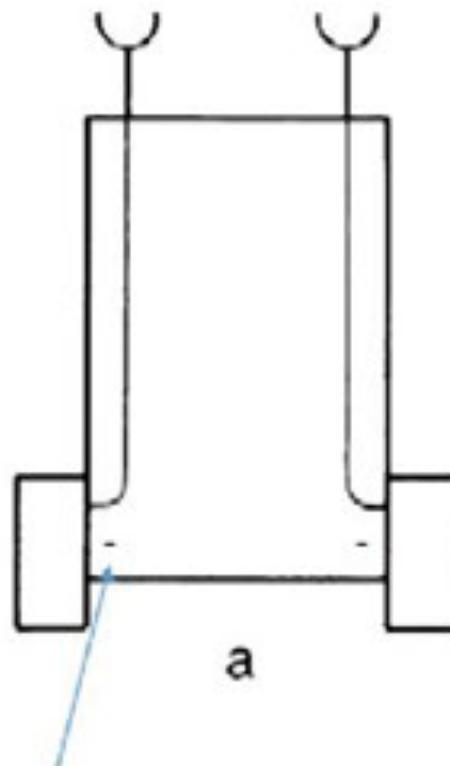
Exploration

0%



## What will be the behaviour of this vehicle?

0



Inhibitory connections

Fear

0%

Aggression

0%

Love

0%

Exploration

0%



## Self learning of behaviours can be achieved with:

0

Finite State Machine

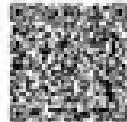
0%

Evolutionary Algorithm

0%

Behaviour Tree

0%



Execution order of tasks can be defined in

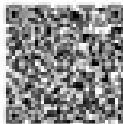
0

Finite State Machines

0%

Behaviour Trees

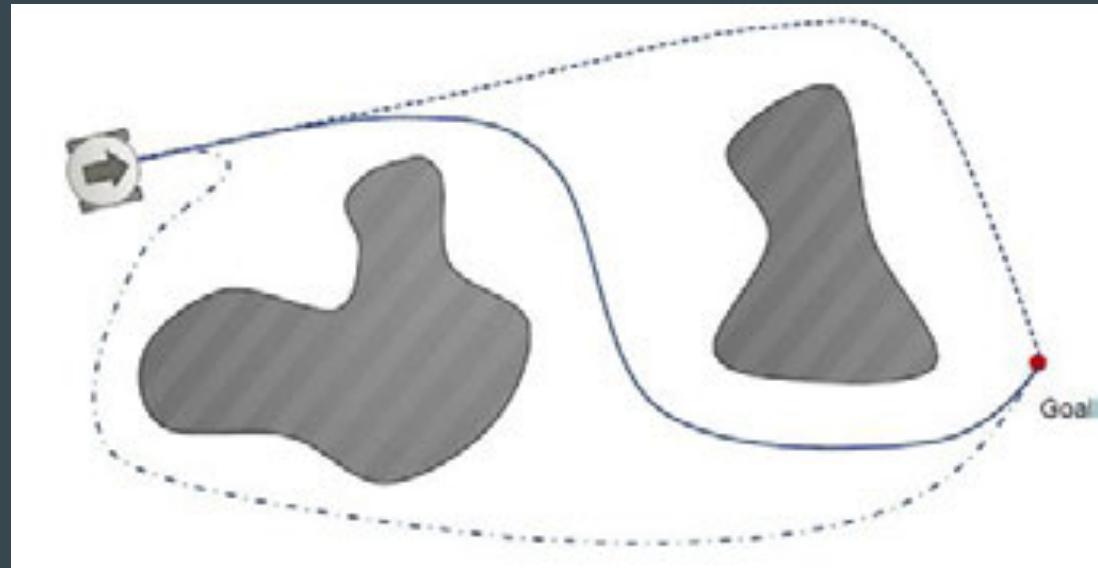
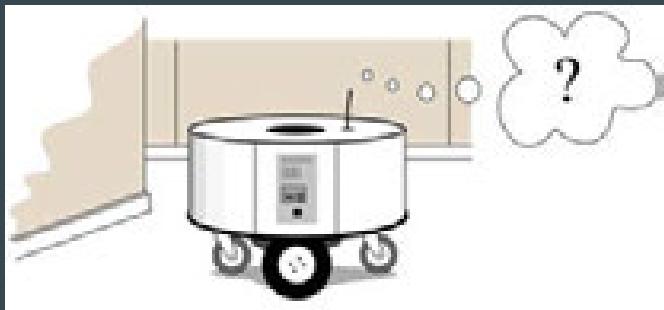
0%



# Navigation

# Navigation – key questions

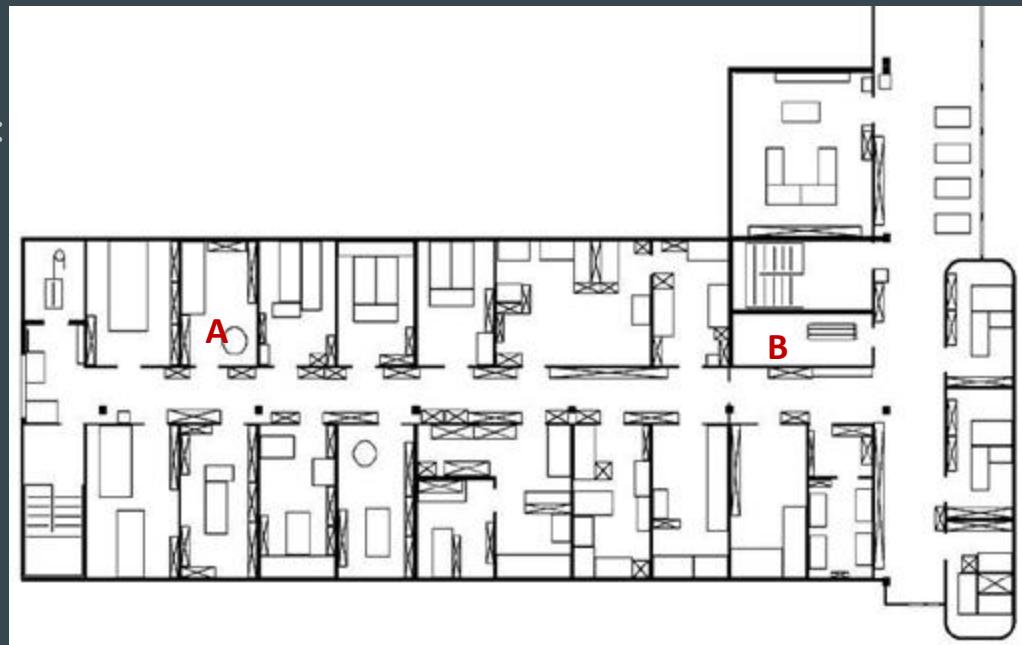
- Where am I?
- Where do I go?
- How do I get there?



# Navigation - Getting from A to B

- Does a robot actually need to know where it is?

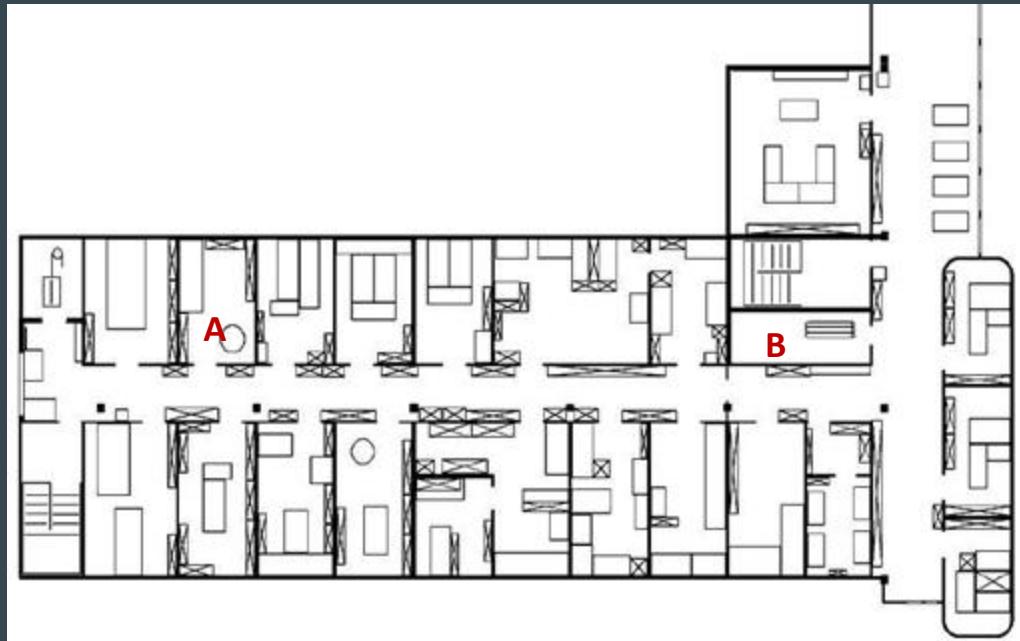
- Task – start at A and reach goal B:
  - Without hitting obstacles
  - Detect arrival at goal



# Behaviour-based navigation

Following the left wall:

- How do we know when the goal is reached?
- Will this work in all environments?
- How accurately / reliably does the robot reach the goal?

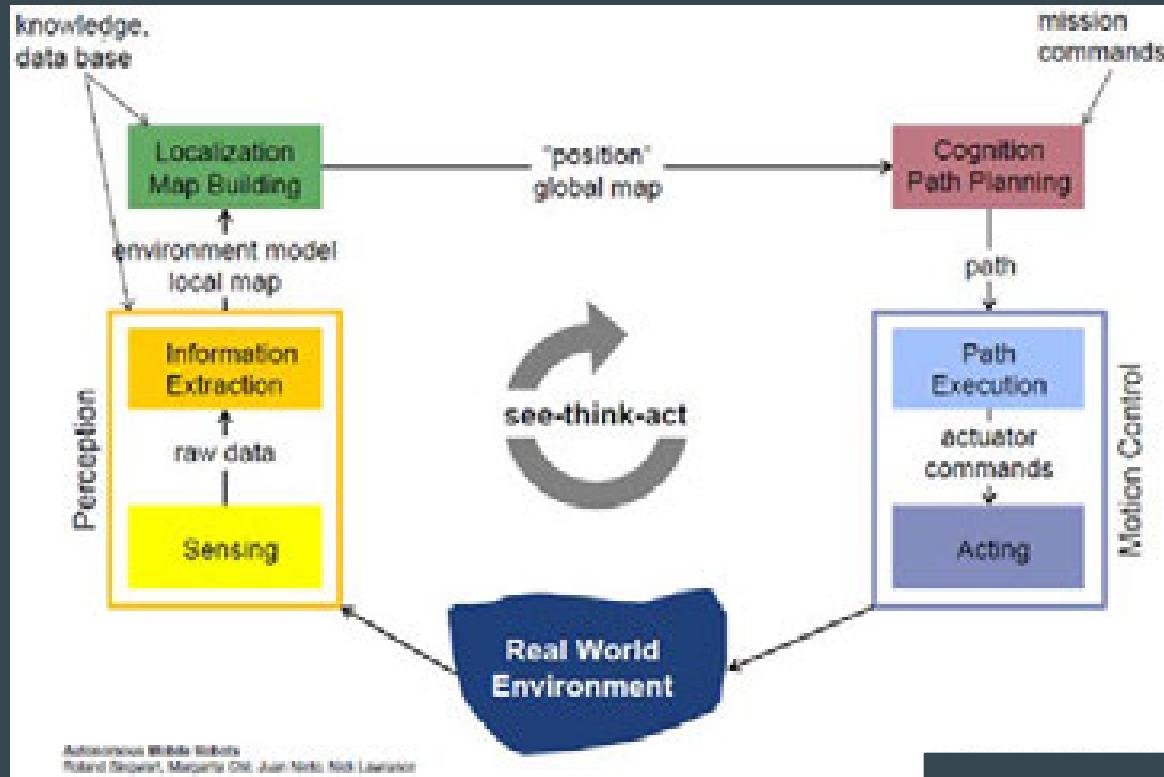


# Robot navigation

To navigate successfully and efficiently, a robot needs to:

- Perceive and understand the environment
- Localise itself within the environment
- Plan a route and execute that plan (motion control)

# Sense / think / act cycle

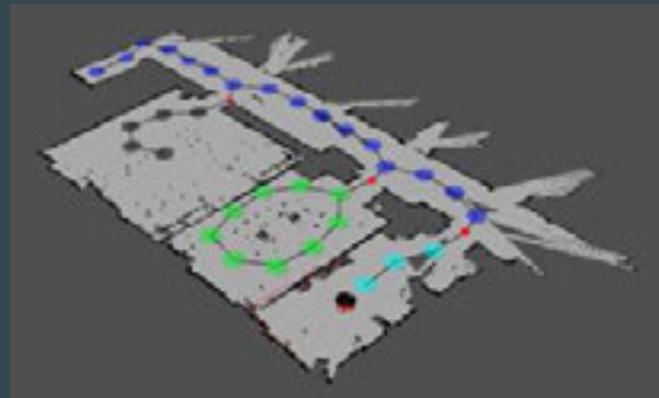


# Types of navigation

- Global navigation (way-finding, requires a map)
  - Robot is **not** told its initial position
  - Position should be estimated from scratch
- **Position tracking (continuous localisation)**
  - Robot knows its initial position
  - It has to accommodate small errors in its odometry as it moves

# Global navigation strategies

- **Recognition-triggered response**
  - Local navigation triggered by last recognised place
- **Topological route**
  - Based on topological maps (no geometric information)
  - Graph with places (nodes), and routes between them (edges)
- **Survey navigation**
  - All known places and spatial relations embedded into the same frame of reference
  - Can discover new paths (e.g. shortcuts / detours)



# Local navigation strategies

## Search

- Can recognise arrival at the goal
- Finds goal by chance

## Direction following

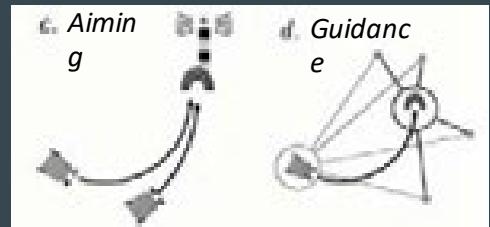
- e.g. compass direction, or trail following
- Finds goal from one direction

## Aiming

- Keeps goal in front while moving
- Finds obvious goal (e.g. beacon) in local area

## Guidance

- Finds goal defined by its relation to the surroundings



(Franz & Mallot, 2000)

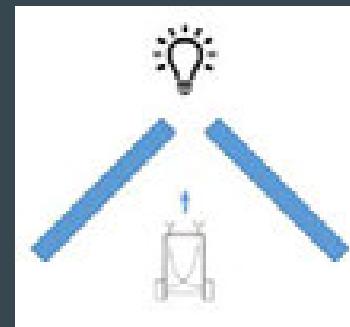
# Local strategies using vision

- Use a visual cue at the goal reactively
  - Detect goal from a distance, and maintain a course towards it
  - Landmark navigation / beaconing
- Use a visual cue in the simplest form of navigation
  - Remember visual surroundings to recognise when you arrive at the goal
- More generally
  - May be able to use landmarks around the goal to determine the course towards it (visual homing)



# Beaconing problems

- Cues must be visible, but may be:
  - Temporarily obscured
  - Only visible at short ranges
- More complex planning  
is needed in adverse conditions



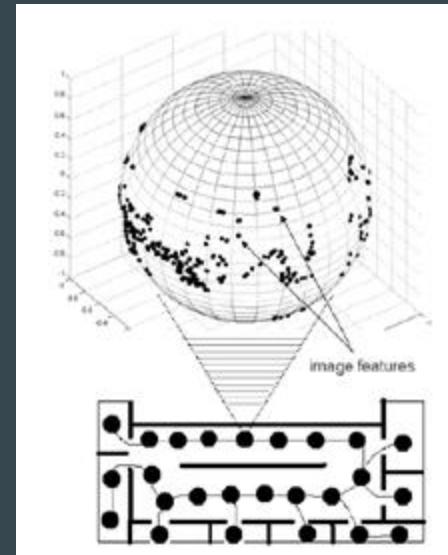
# Problems with landmark recognition

- How to define a landmark?
- How to match landmarks between current scene and memory?
  - Correspondence problem
- Robot's view must be aligned with recorded landmark



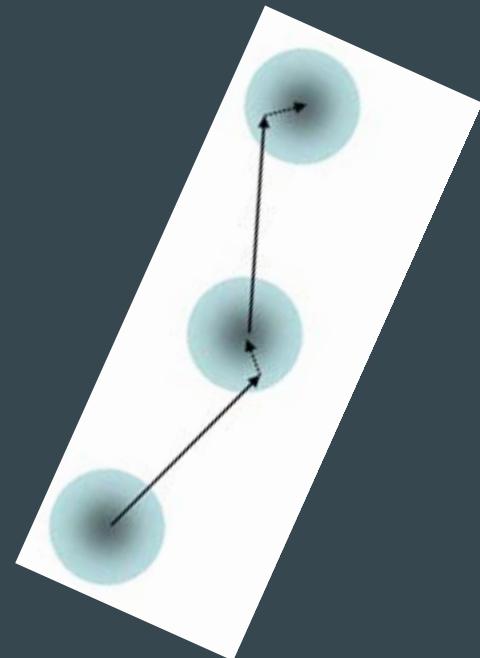
# Feature based recognition

- Visual features detected via on-board camera sensors can be used to describe a location
- Matching features/descriptors currently visible against a list of previously visited locations
  - Can be used for place recognition and localisation



# From local strategies to routes

- Local strategies
  - Target location can be found from surrounding area using memory of the target location
  - Outbound route can be used to calculate vector direction to return to starting point
- Multiple memories and/or vectors can be linked together to form route memories



# Landmark recognition along routes

- Recognising where you are with respect to previous memories
  - Continues to be a problem in robotics
- **Loop closure:** ability to recognise a location even if perceived from a different pose
  - Allows correction of robot's position when simultaneously mapping
- “Kidnapped robot problem” can result in failed localisation

# Odometry

# Odometry / dead reckoning

- Approximate location of a robot can be obtained by repeatedly computing the **distance moved**, and the **change in direction**, from the **velocity of the wheels** over a **short period of time**
- Also called **deduced reckoning** or **dead reckoning**
- Robot motion recovered by integrating proprioceptive sensor velocities readings
  - Advantages: straightforward
  - Disadvantages: errors are integrated (unbound)
- Heading sensors (e.g. IMU) help to reduce the accumulated errors, but drift remains

Example : piloting barren Martian surface  
with no external guidance cues e.g. GPS



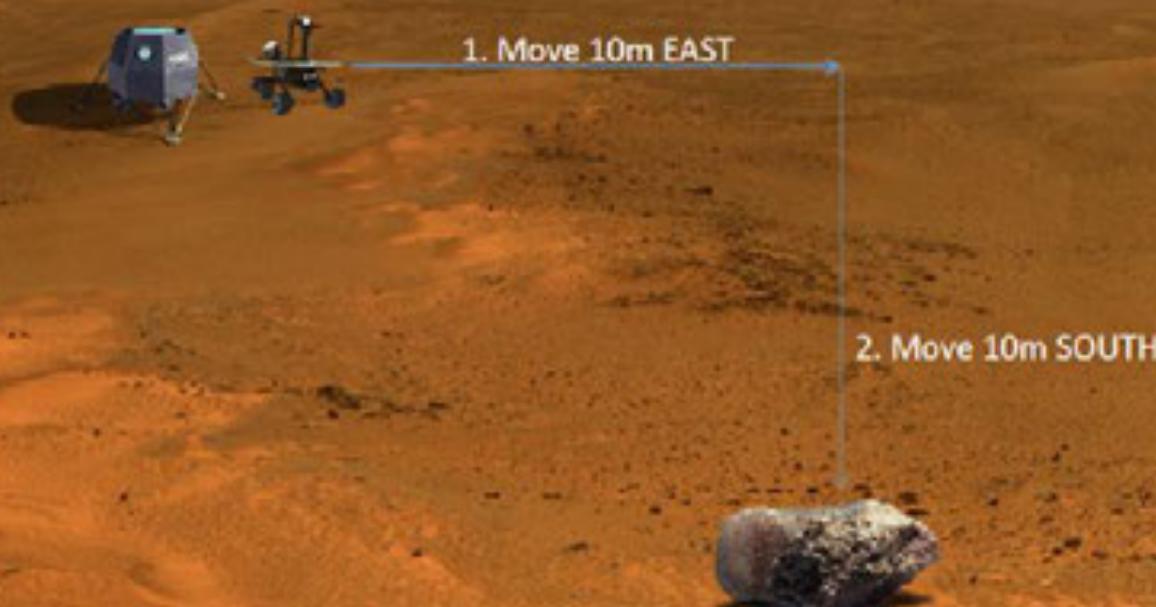
Example : piloting barren Martian surface  
with no external guidance cues e.g. GPS



!! NASA COMMANDS !!

1. Move 10m EAST
2. Move 10m SOUTH
3. Bring back minerals

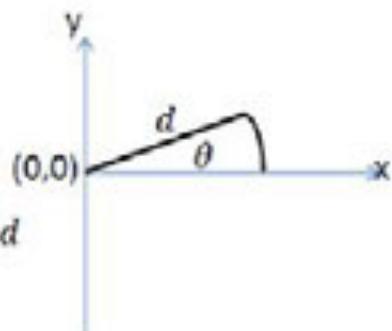
Example : to execute commands the robot must continuously calculate it's position wrt to base



## Example : formulation



Robot position:



$d$  = distance travelled

$\theta$  = orientation

1. Move 10m EAST

2. Move 10m SOUTH

## Example : formulation



1. Move 10m EAST

2. Move 10m SOUTH

Robot position:

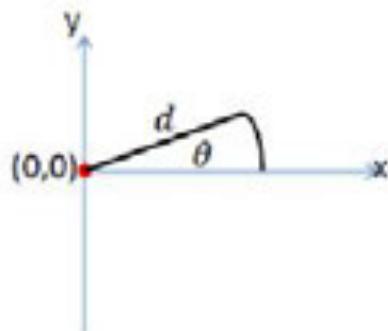
$$x(t) = x(t-1) + \Delta x$$

$$y(t) = y(t-1) + \Delta y$$

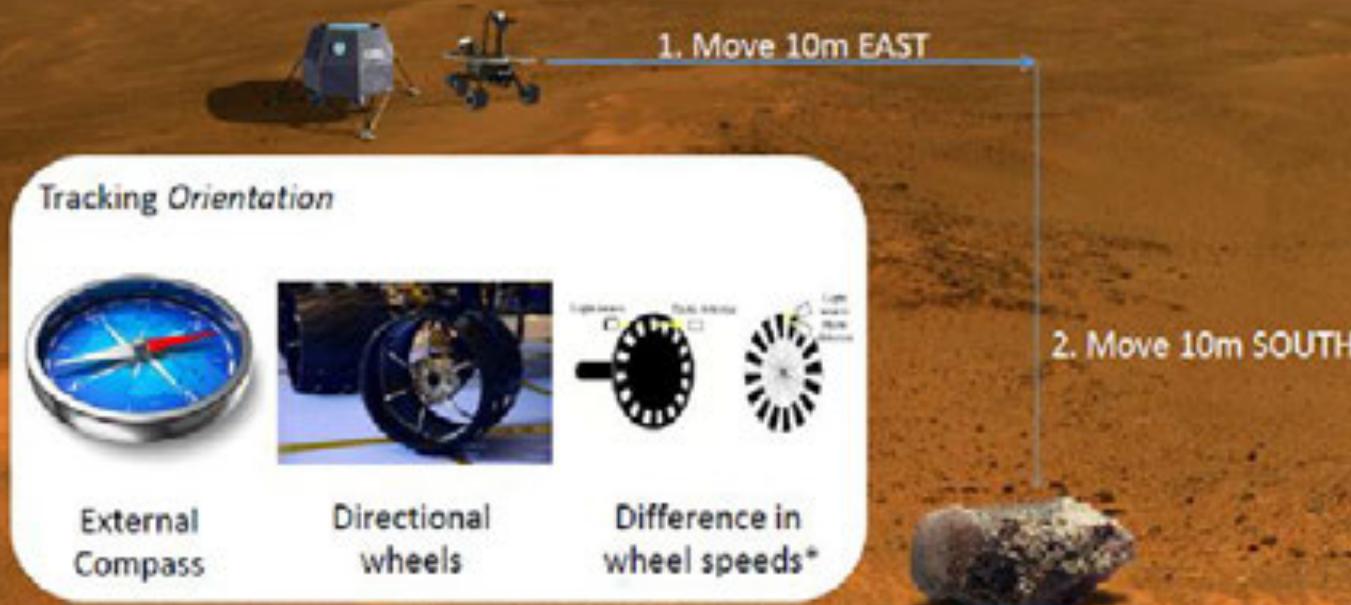
where

$$\Delta x = d \cdot \cos(\theta)$$

$$\Delta y = d \cdot \sin(\theta)$$



## Example : monitoring orientation



## Example : measuring distance travelled



1. Move 10m EAST

Measuring  $d$  from wheel rotations



2. Move 10m SOUTH

$$\begin{aligned}d &= \text{revolutions} \cdot \text{wheel circumference} \\&= \text{revolutions} \cdot \pi \cdot D\end{aligned}$$



## Example : measuring distance travelled



1. Move 10m EAST



2. Move 10m SOUTH



*Measuring  $d$  from wheel rotations*



$$d = 1/2 * \pi * 50\text{cm}$$
$$d = 78.54\text{cm}$$

## Getting home



!! EMERGENCY !!

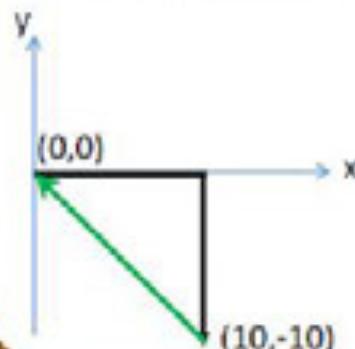
SAND-STORM IMMINENT

4. Seek shelter immediately

## Getting home



Calculating the home vector



Cartesian format:

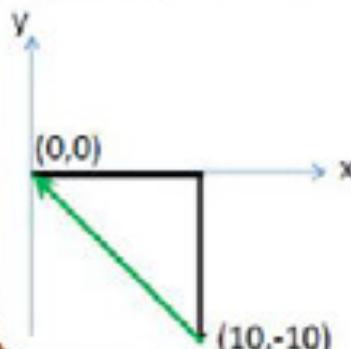
$$\begin{aligned} \text{HV} &= [x_1 - x_2, y_1 - y_2] \\ &= [0 - 10, 0 - (-10)] \\ &= [-10, 10] \end{aligned}$$



# Getting home



Calculating the home vector



Polar format:

$$HV = [r, \theta]$$

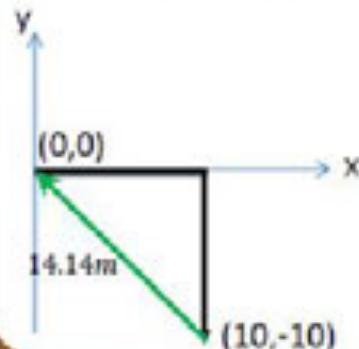
where  $r$  is distance home,  
and  $\theta$  is bearing from x



# Getting home



Calculating the home vector



Polar format:

$$HV = [14.14, 0]$$

where  $r$  is distance home,  
and  $\theta$  is bearing from x

$$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$r = \sqrt{(10 - 0)^2 + (-10 - 0)^2}$$

$$r = 14.14m$$



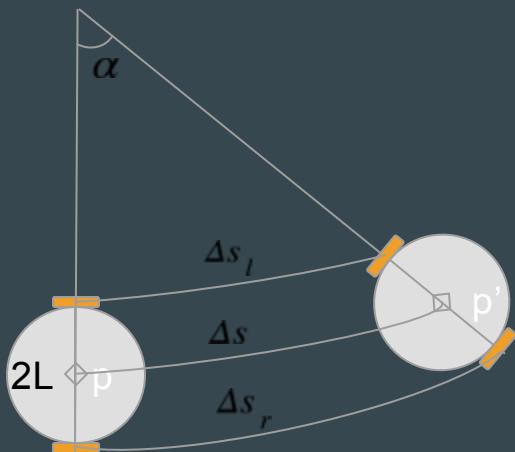
# Odometry – differential drive robot

$2L \rightarrow$  distance between wheels  
 $p \rightarrow$  initial position

$p' \rightarrow$  position after displacement  
 $\Delta s \rightarrow$  Distance travelled by platform

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$

How to calculate change in orientation?



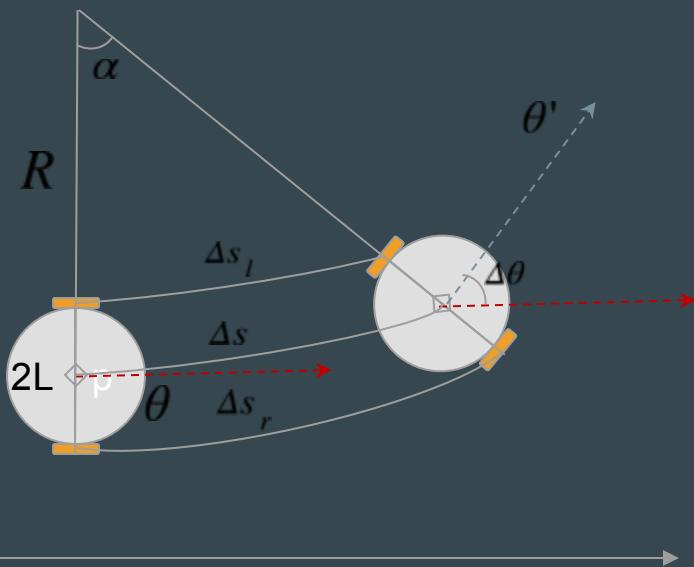
# Odometry – differential drive robot

$2L \rightarrow$  distance between wheels  
 $p \rightarrow$  initial position

$p' \rightarrow$  position after displacement  
 $\Delta s \rightarrow$  Distance travelled by platform

$$\Delta\theta = \alpha$$

$\Delta s_l, \Delta s, \Delta s_r$  Arcs of circle with the same center and different radius



$$\Delta s_l = Ra, \quad \Delta s_r = (R + 2L) a$$

$$a = \frac{\Delta s_l}{R}, \quad a = \frac{\Delta s_r}{(R + 2L)}$$

$$\frac{\Delta s_l}{R} = \frac{\Delta s_r}{(R + 2L)}$$

$$R = \frac{2L \Delta s_l}{(\Delta s_r - \Delta s_l)}$$

Substitute radius  $R$  in  $\Delta s_l$

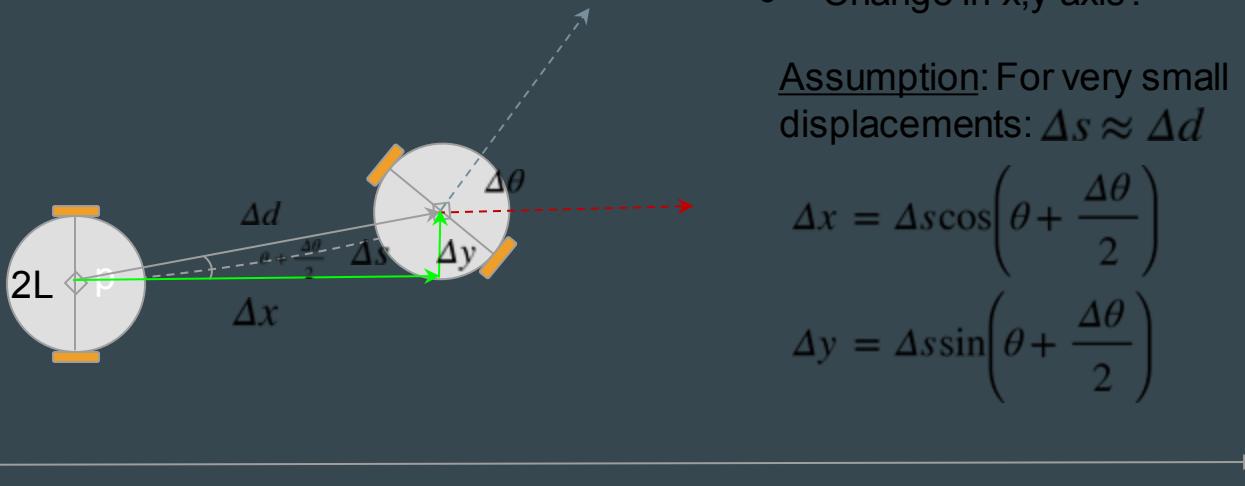
$$\alpha = \frac{\Delta s_l}{R} = \frac{\Delta s_l (\Delta s_r - \Delta s_l)}{2L \Delta s_l}$$

$$a = \frac{(\Delta s_r - \Delta s_l)}{2L} = \Delta\theta$$

# Odometry – differential drive robot

$2L \rightarrow$  distance between wheels  
 $p \rightarrow$  initial position

$p' \rightarrow$  position after displacement  
 $\Delta s \rightarrow$  Distance travelled by platform



How to calculate:

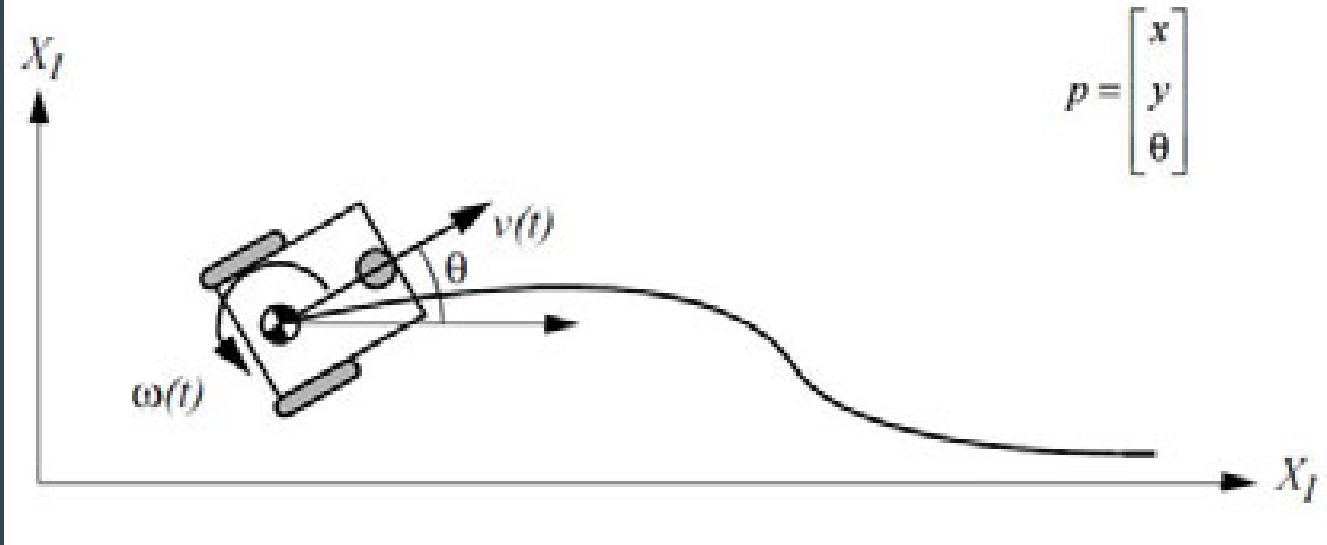
- Change in x,y axis?

Assumption: For very small displacements:  $\Delta s \approx \Delta d$

$$\Delta x = \Delta s \cos\left(\theta + \frac{\Delta\theta}{2}\right)$$

$$\Delta y = \Delta s \sin\left(\theta + \frac{\Delta\theta}{2}\right)$$

# Odometry – differential drive robot



$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

$$p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}$$

# Wheel odometry

Incremental travel distances for a discrete system with fixed sampling interval:

$$p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}$$

$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2)$$

$$\Delta y = \Delta s \sin(\theta + \Delta\theta/2)$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b}$$

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$

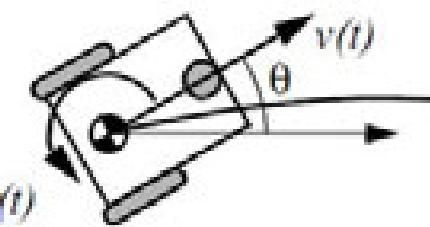
where

$(\Delta x; \Delta y; \Delta\theta)$  = path traveled in the last sampling interval;

$\Delta s_r; \Delta s_l$  = traveled distances for the right and left wheel respectively;

$b$  = distance between the two wheels of differential-drive robot.

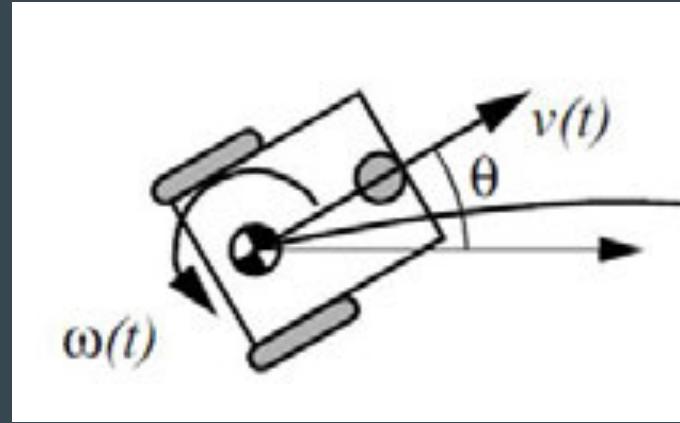
These terms comes from the application of the Instant Centre of Rotation



# Mobile robot odometry

Putting it all together....

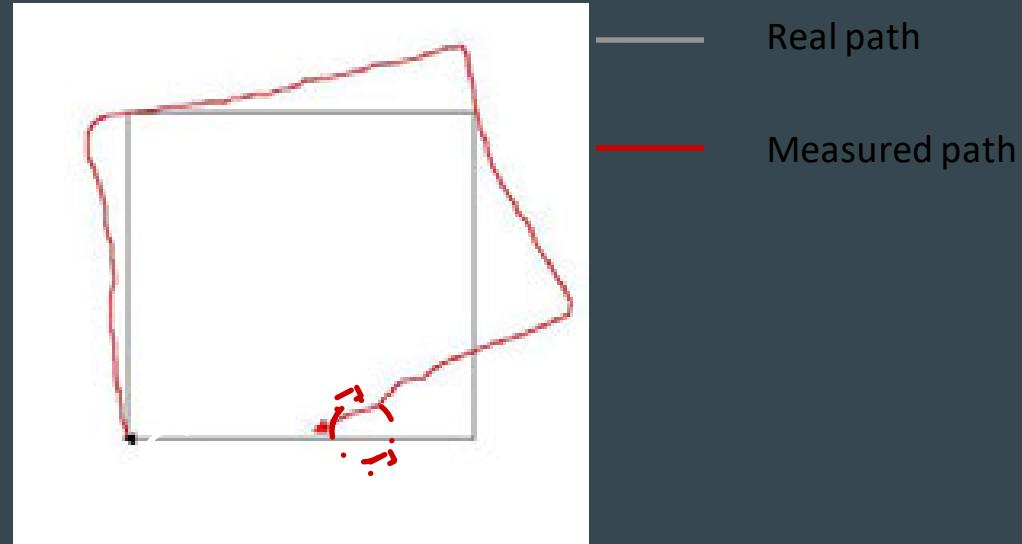
$$p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}$$



$$p' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = p + \begin{bmatrix} \Delta s \cos(\theta + \Delta\theta/2) \\ \Delta s \sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\theta + \Delta\theta/2) \\ \Delta s \sin(\theta + \Delta\theta/2) \\ \Delta\theta \end{bmatrix}$$

# Problems with Odometry

- Inaccuracies / noise cause estimated robot position to drift over time
- Solution: use a map!
  - Combine odometry with sightings of known landmarks / environmental features



# Odometry error types

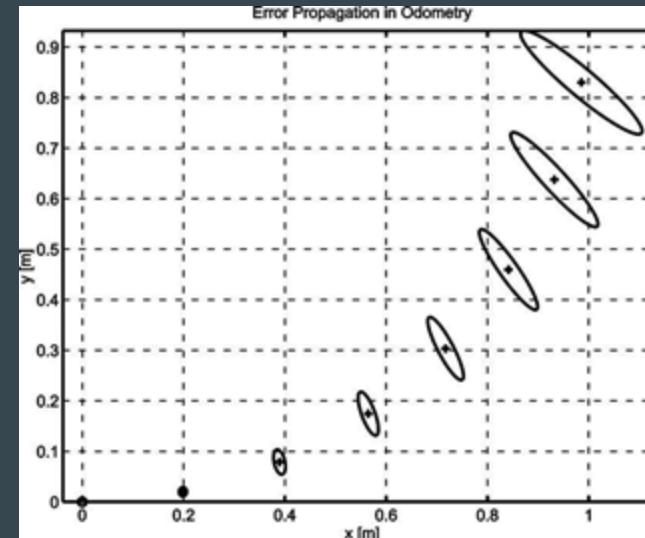
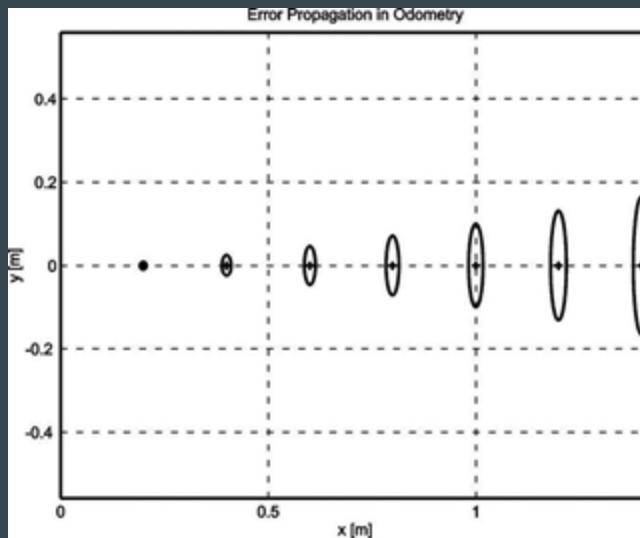
- **Range error**
  - Sum of the wheel motions leads to an error in the integrated path distance of the robot's movement
- **Turn error**
  - Difference of the wheel motions leads to an error in the robot's final orientation
- **Drift error**
  - Difference in the error of the wheels leads to an error in the robot's angular orientation

# Odometry error sources

- **Systematic**
  - Misalignment of the wheels
  - Unequal wheel diameter
- **Non-systematic**
  - Variation in the contact point
  - Unequal floor contact of the wheel (slippage)
- Limited resolution during integration
  - Time increments, measurement resolution

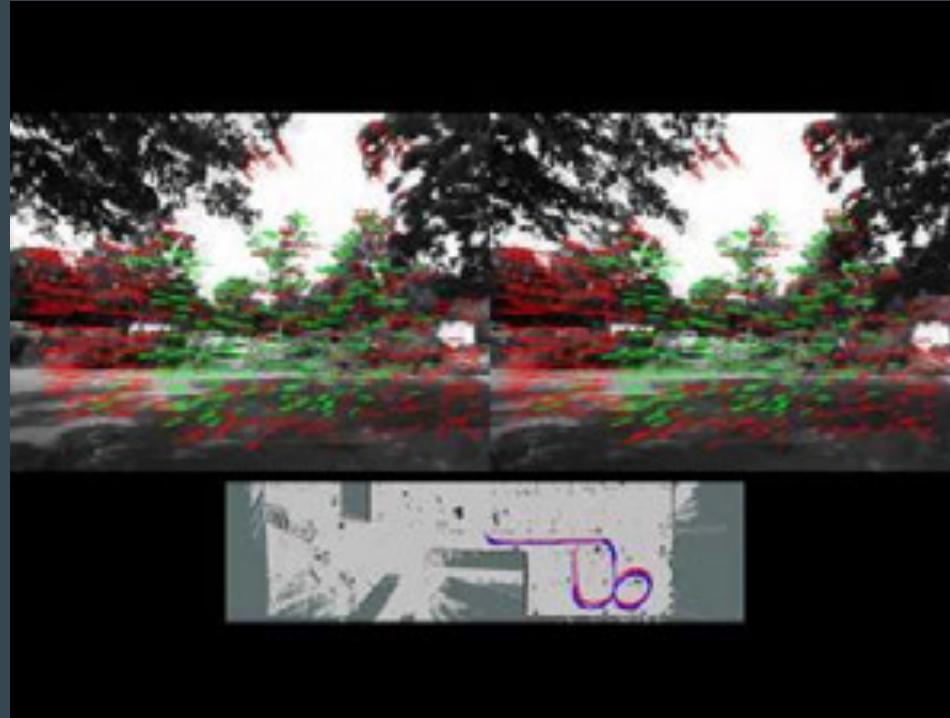


# Error propagation in odometry



# Visual odometry

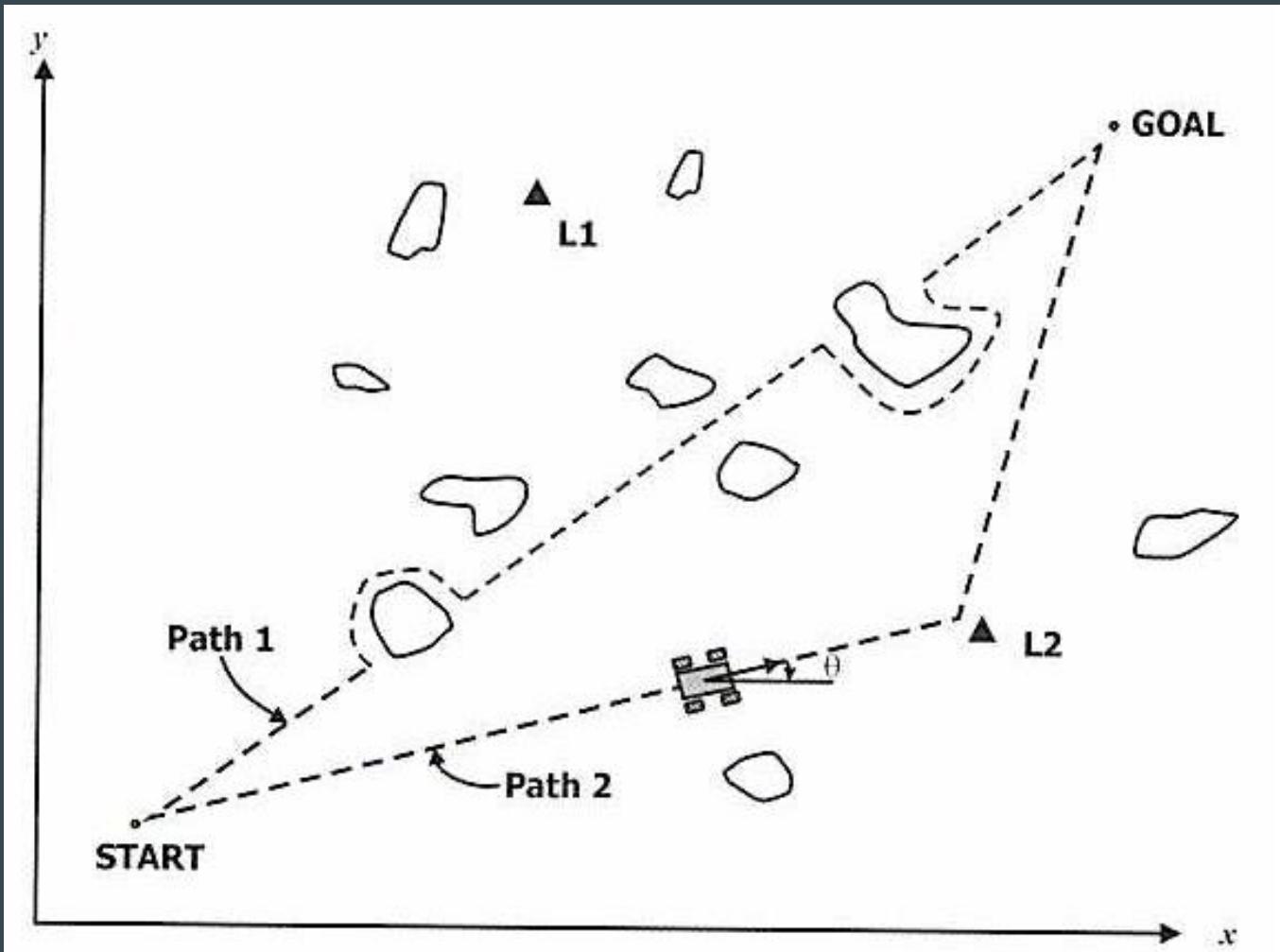
- Odometry is not limited to wheel encoders
- Consecutive camera images can be used to estimate velocity



# Odometry summary

- Main advantage: can function independent of external cues and sensors (e.g. GPS) and without maps
  - However, it accumulates error over time
- SLAM provides the best of both worlds
  - Uses odometry for rapid position updates, which are re-aligned periodically using the map

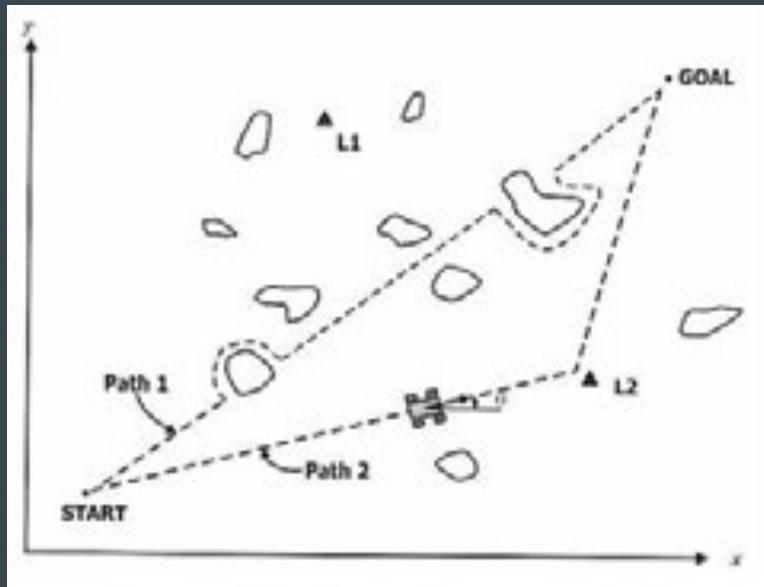
# Navigation strategies



# Navigation by vision and compass

## Path 1

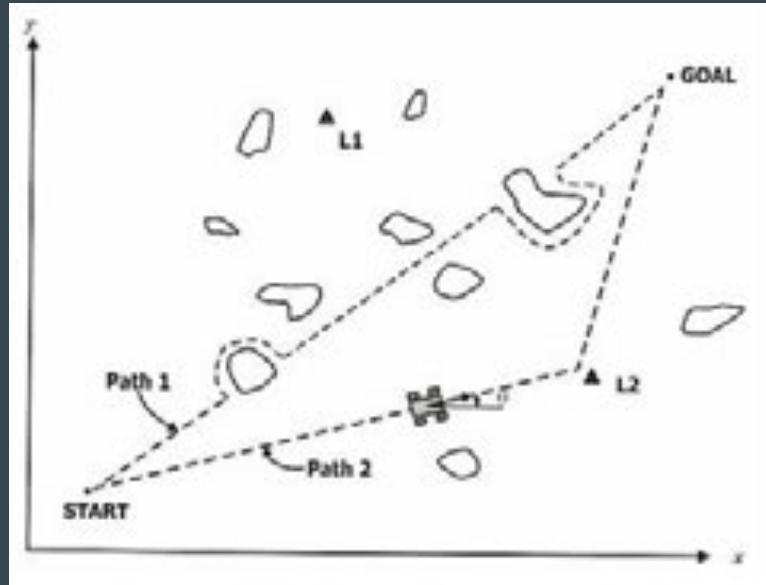
- Assuming visible goal, unknown distance
- Use vision to recognise goal
- Use compass to maintain heading towards goal
- When obstacle encountered (goal no longer visible):
  - Remember current compass reading
  - Travel around obstacle until the original compass direction is sensed again and goal is visible



# Navigation using landmarks as beacons

## Path 2

- Assuming goal not visible from start
- Aim towards landmark 2, then towards goal



# Navigation by position tracking

- Goal not visible, but known coordinates
- Use GPS (i.e. SatNav)
  - Does not always work and/or not perfectly accurate
- Cannot use only odometry due to drift errors (cumulative over time)
- Use a map with positions of known landmarks to correct your odometry
  - Problem of self-localisation (next lecture)



# Navigation strategies

General purpose solution to the navigation problem in mobile robots also requires:

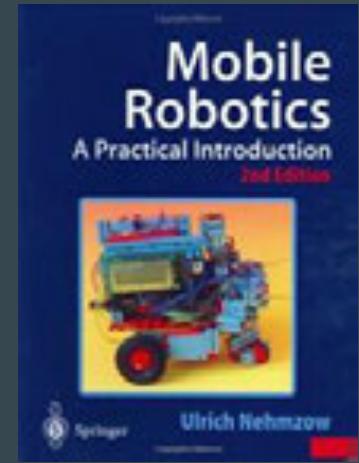
- **A representation of the navigable space**
  - e.g. graph or grid, derived from the global map of the environment
- **A path planner**
  - Use algorithms such as A\* or Dijkstra to find the best route to the goal location (a set of waypoints)
- **An “auto-pilot”**
  - Control software to drive between waypoints, taking into account account kinematics/dynamics
- **Also need to avoid unexpected obstacles along the way**

# Summary

- Navigation
- Global & Local Navigation
- Odometry
- Navigation Strategies

# Recommended reading

- Nehmzow, U., *Mobile Robotics: A Practical Introduction*, (Springer, 2003). Chapter 5.
- Bekey, G.A., *Autonomous Robots: From biological inspiration to implementation and control*, (MIT Press). Chapter 14.
- Siegwart R. et al., *Autonomous Mobile Robots*, (MIT Press). Chapter 5.



# CMP3103M

# Autonomous Mobile Robotics

...

## Lecture 7: Localisation

Dr Athanasios Polydoros

# Overview

## Contents:

- Quiz
- Maps
  - Metric
  - Topological
  - Semantic
  - Hybrid
- Localization
  - Monte Carlo
  - Kalman Filter
  - Markov

## Learning Outcomes:

- Describe different types of Maps for localization
- Explain various localization methods

# Interactive Quiz:

Join:



<https://pollev.com/athanasiospolydoros472>

# In global navigation the robot's initial position has to be known

True

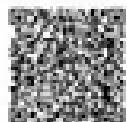
Fals



## Odometry is used to:

Get robot's initial  
position

Estimate robot's position  
after it has moved



## Odometry error sources can be:

Unequal diameter  
of wheels

Slippage

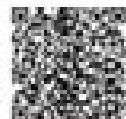
Both



# Visual Odometry uses

Only camera  
images

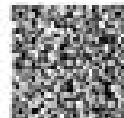
Only sonar  
readings



To calculate odometry we do not need the initial robot position

True

False



# Intro

- What is robot localization?
- What we need to localize a robot within its environment?

# Maps

# Maps – Definition

What is a map?

- Collection of elements or features at some scale of interest, alongside with a representation of the spatial and semantic relationships among them

# Maps – Types

Definition: Collection of elements or features at some scale of interest, alongside with a representation of the spatial and semantic relationships among them

## Types:

- **Metric maps**
  - Record the location of objects in an absolute coordinate system
- **Topological maps**
  - Record the connections (edges) between a set of places (nodes)
- **Semantic maps**
  - Record semantic information (metadata), e.g. place/object names
- **Hybrid maps**
  - Combine two or more of the map type above

# Metric maps

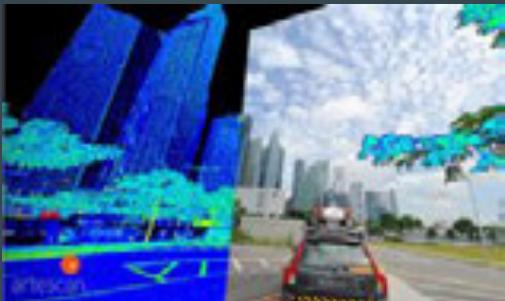


Record the location of objects in an absolute coordinate system

# Metric maps – Types

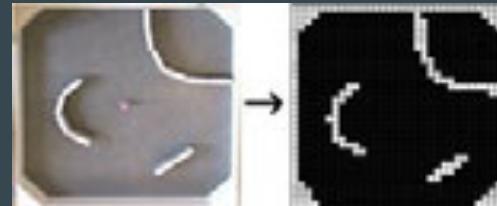
## Continuous / “vector” format

- Points, linear/curved segments, surface patches

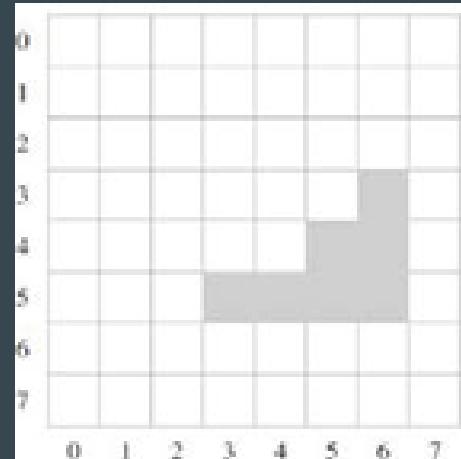
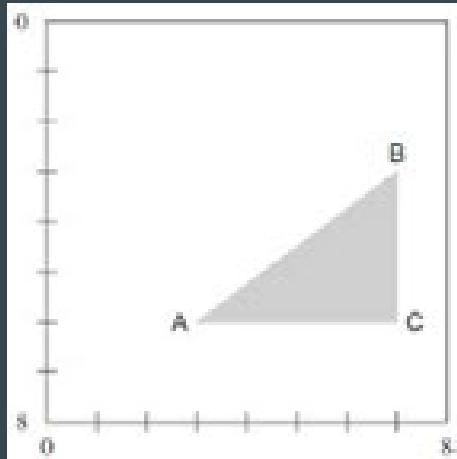


## Discrete / “raster” format

- Occupancy grids



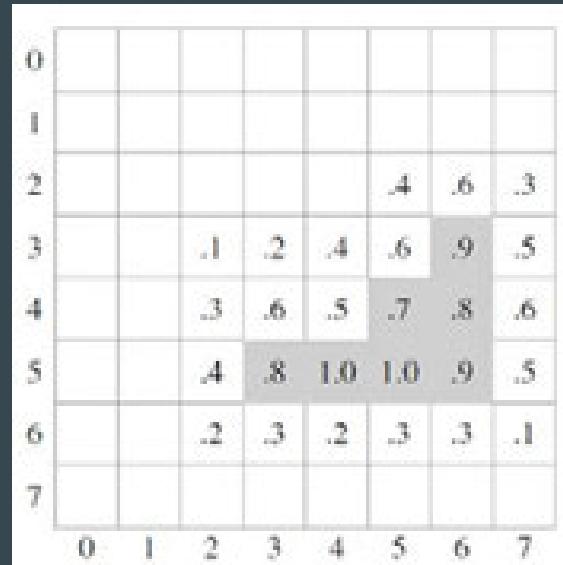
# Metric maps – Continuous vs discrete



What are the pros/cons of each?

# Metric maps – occupancy grids

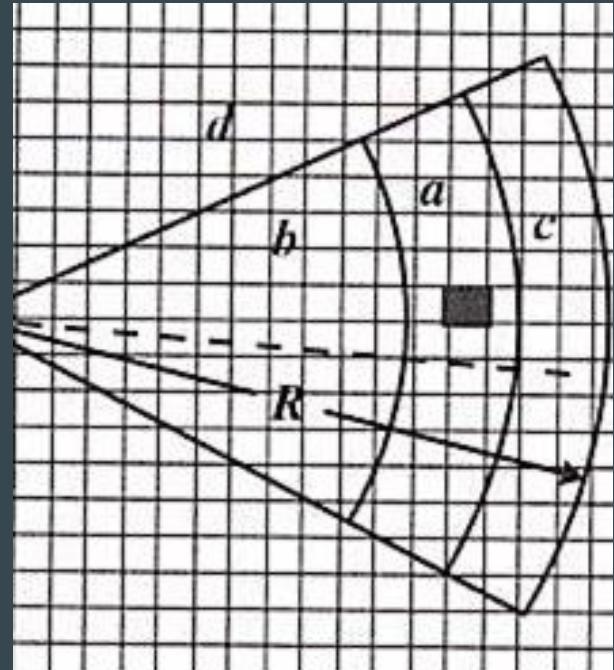
- Robot doesn't have complete and accurate prior knowledge about obstacles
- Each cell is associated with a probability that the cell is occupied,  $P(\text{occ}_{x,y})$
- Updated using current robot pose  $(x, y, \theta)$  and depth measurements from range-finder sensors, e.g. sonar, laser, stereo-vision



# Model of a sonar beam

We need a model of the range sensor, e.g. for sonar we would define the following regions for a range measurement  $R$ :

- Probably occupied
- Probably empty
- In the shadow of the detected object, so status unknown
- Outside the beam, so status unknown



# Reasoning with probabilities

- Probabilistic reasoning formalises the process of accumulating evidence, and updating probabilities based on new evidence
- **Prior** probability – belief **before** the new evidence
- **Posterior** probability – belief **after** the new evidence

# Bayes' rule

- General formula for Bayes' Theorem (discrete case):

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)}$$

- Expresses the relation between a conditional probability and its inverse
- Or, another way of writing it:

$$P(A | B) = \frac{1}{c} P(B | A)P(A)$$

# Bayes' rule

- The quantities in Bayes' rule are often described as follows:

$$P(A | B) = \frac{1}{c} P(B | A) P(A)$$

posterior probability

prior probability

likelihood

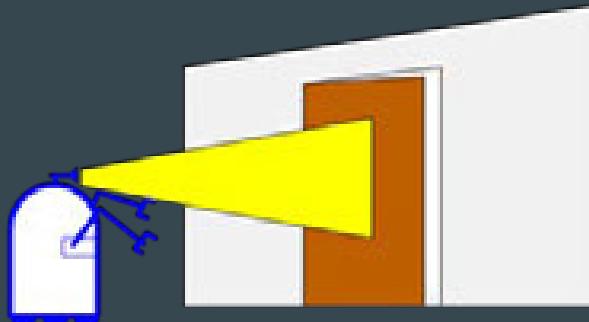
normalisation factor

$c$

# Bayes' Rule example

# Probabilistic robotics

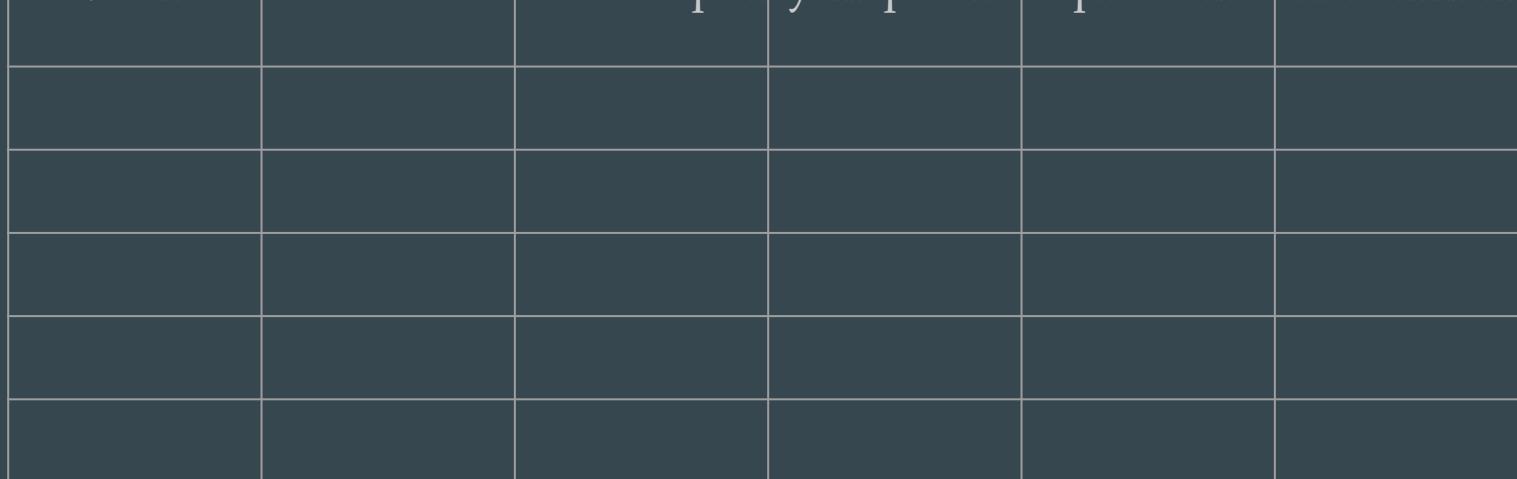
- Explicit representation of uncertainty using the calculus of probability theory
- Probability of the door being open, given observation  $z$
- Based on:
  - Probability of observation  $z$ , given the door is open
  - Probability of doors being open (in general)
  - Probability of observation  $z$



$$P(\text{open} \mid z) = \frac{P(z \mid \text{open})P(\text{open})}{P(z)}$$

# Frontier Exploration Algorithm

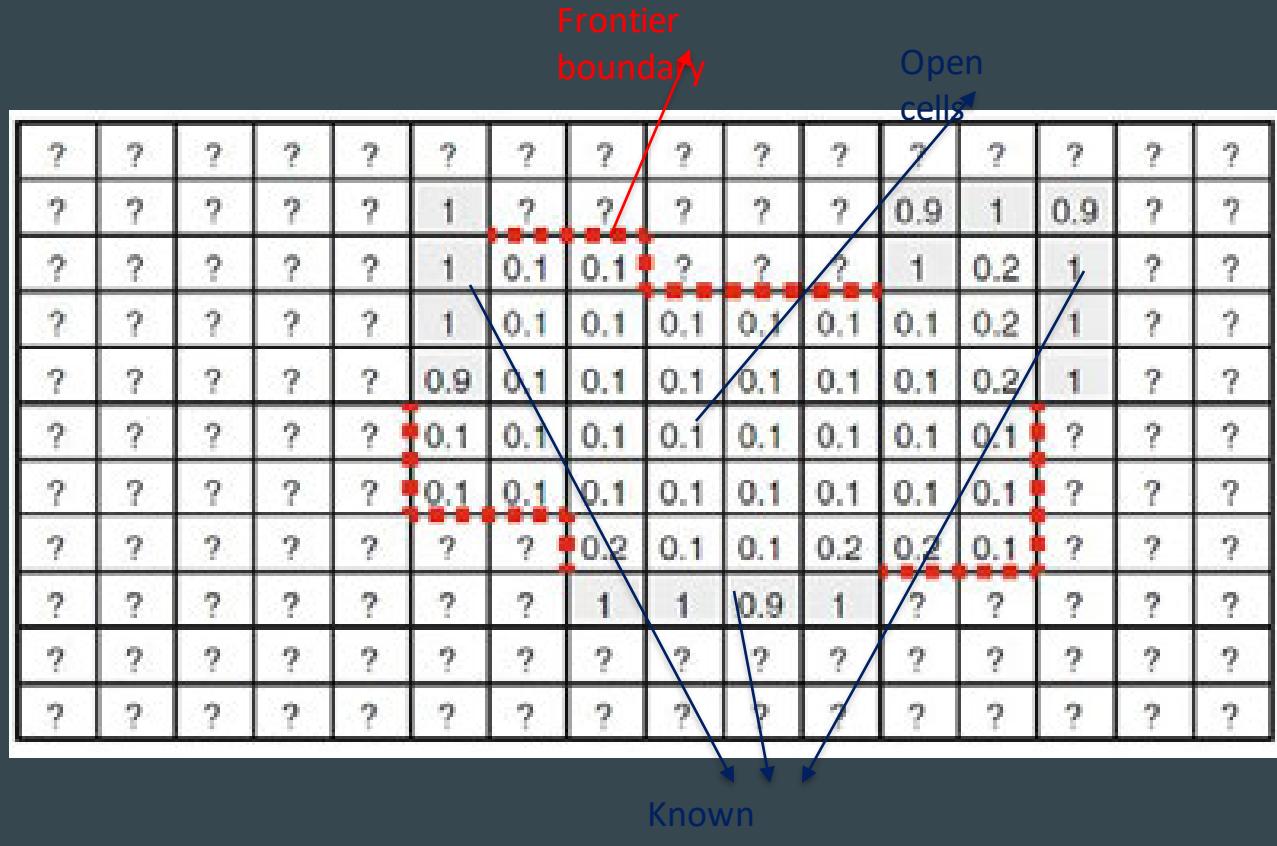
- Method to create discrete occupancy maps and explore the environment



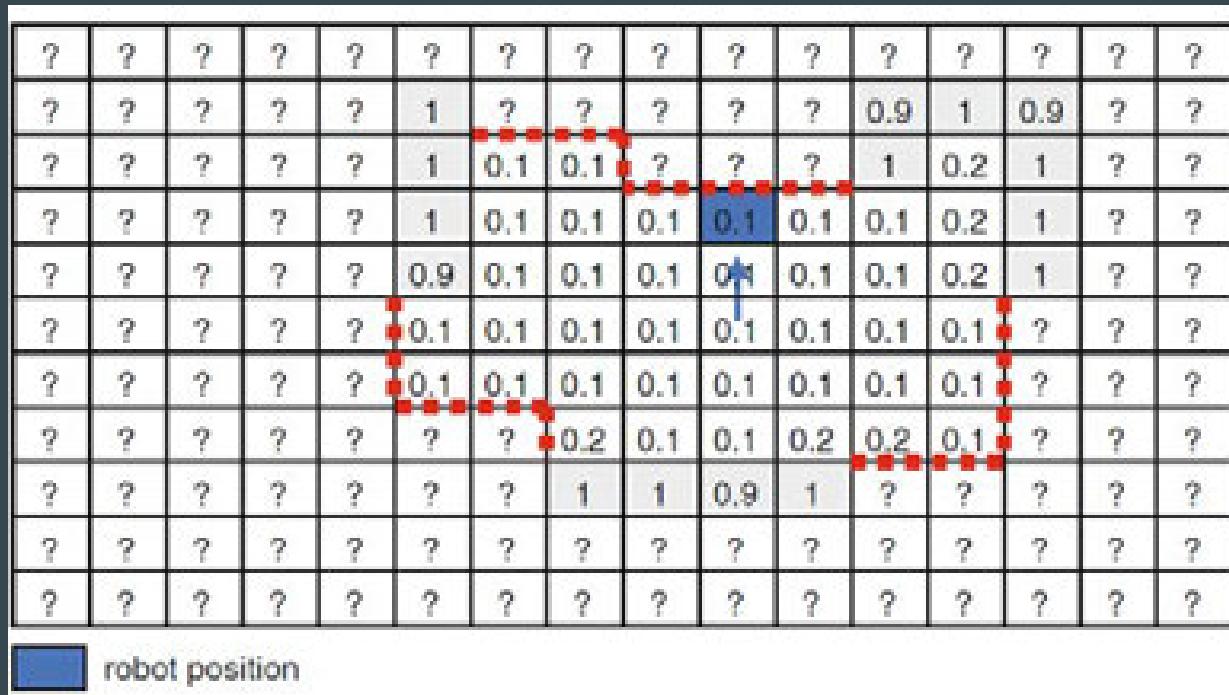
- Find boundaries between explored and unexplored areas
- Drive the robot to explore the boundaries

# Frontier algorithm

## Frontier Algorithm – Grid Maps with Occupancy Information

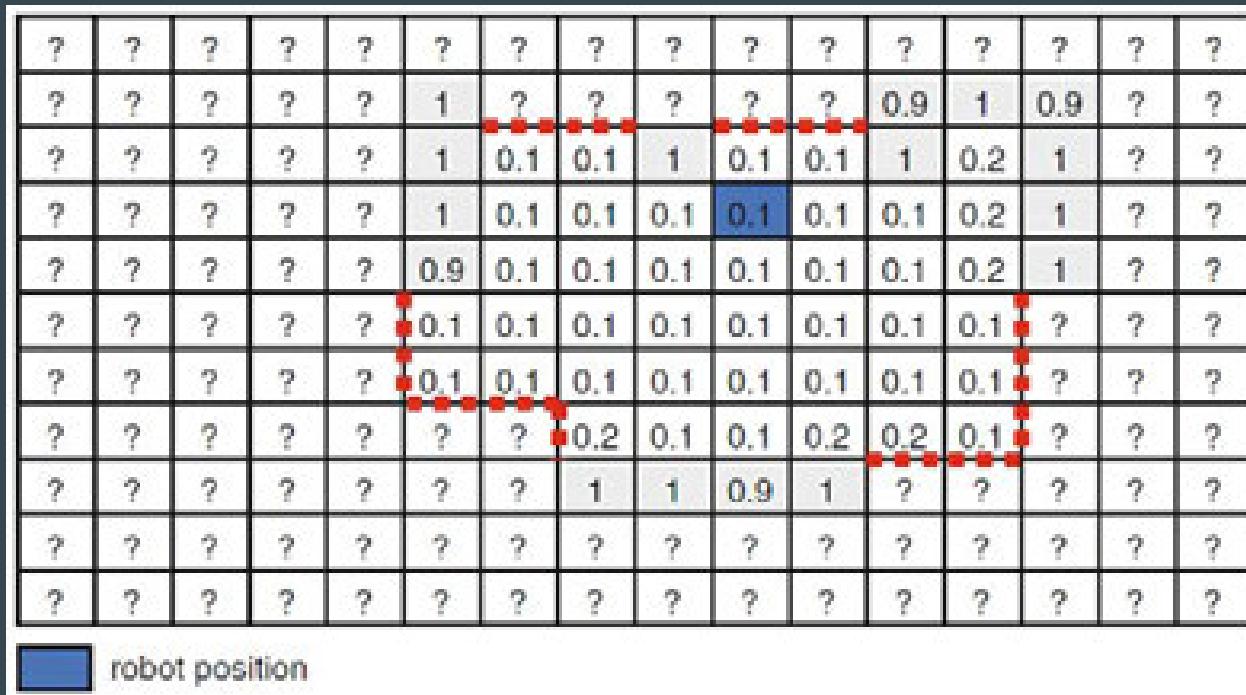


# Frontier algorithm



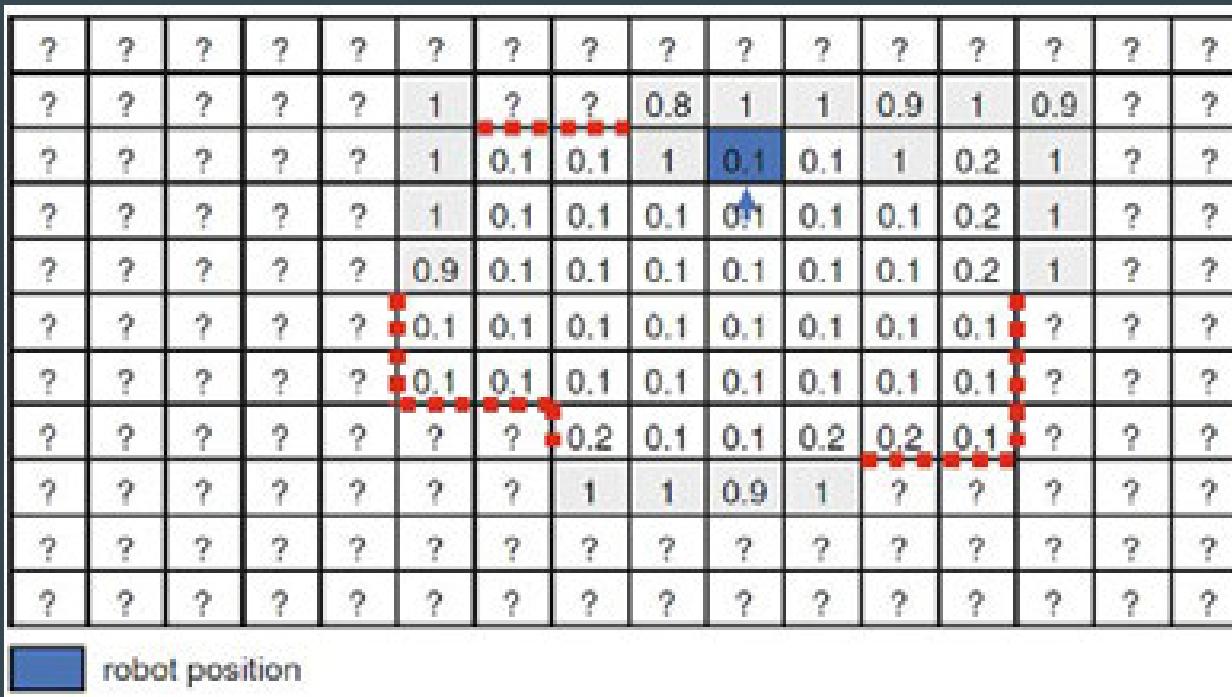
# Frontier algorithm

## Frontier Algorithm – Grid Maps with Occupancy Information



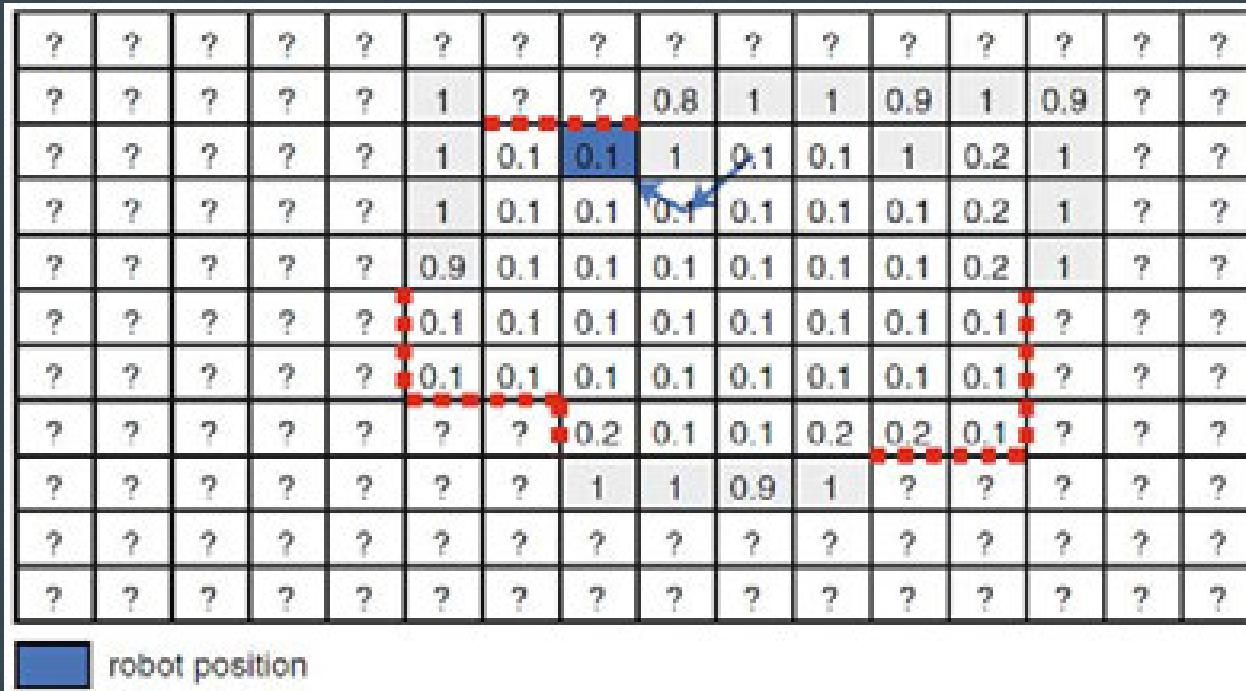
# Frontier algorithm

## Frontier Algorithm – Grid Maps with Occupancy Information



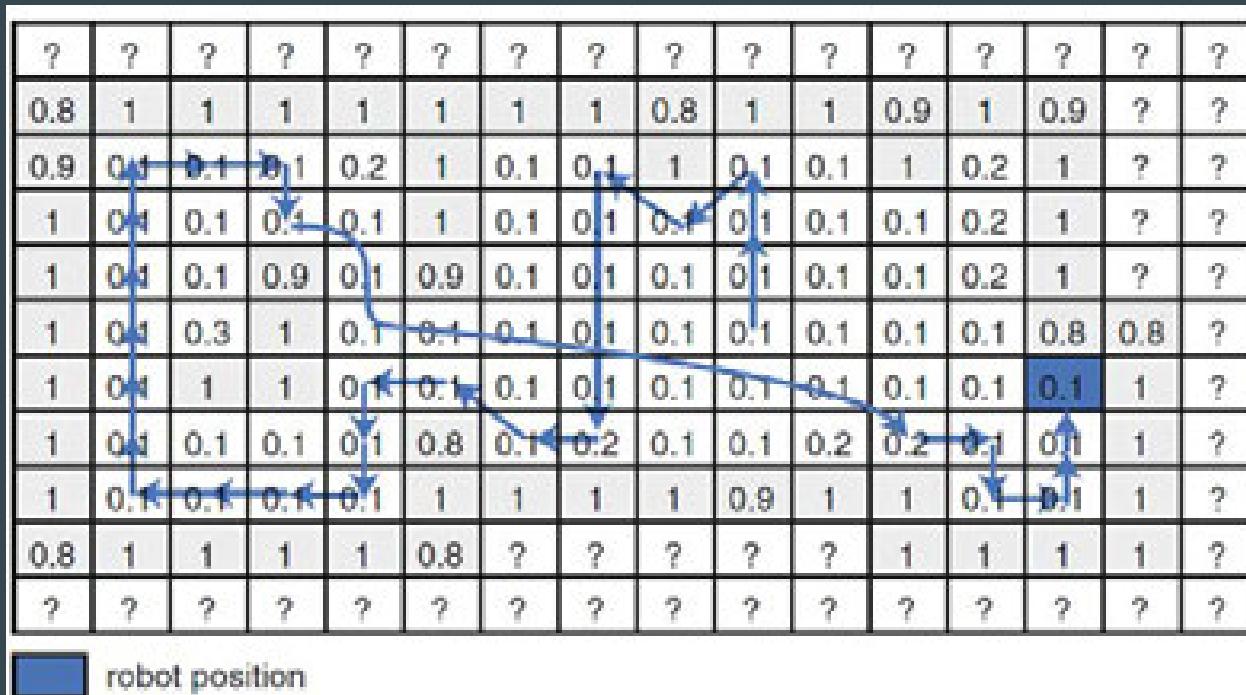
# Frontier algorithm

Frontier Algorithm – Grid Maps with Occupancy Information



# Frontier algorithm

## Frontier Algorithm – Grid Maps with Occupancy Information



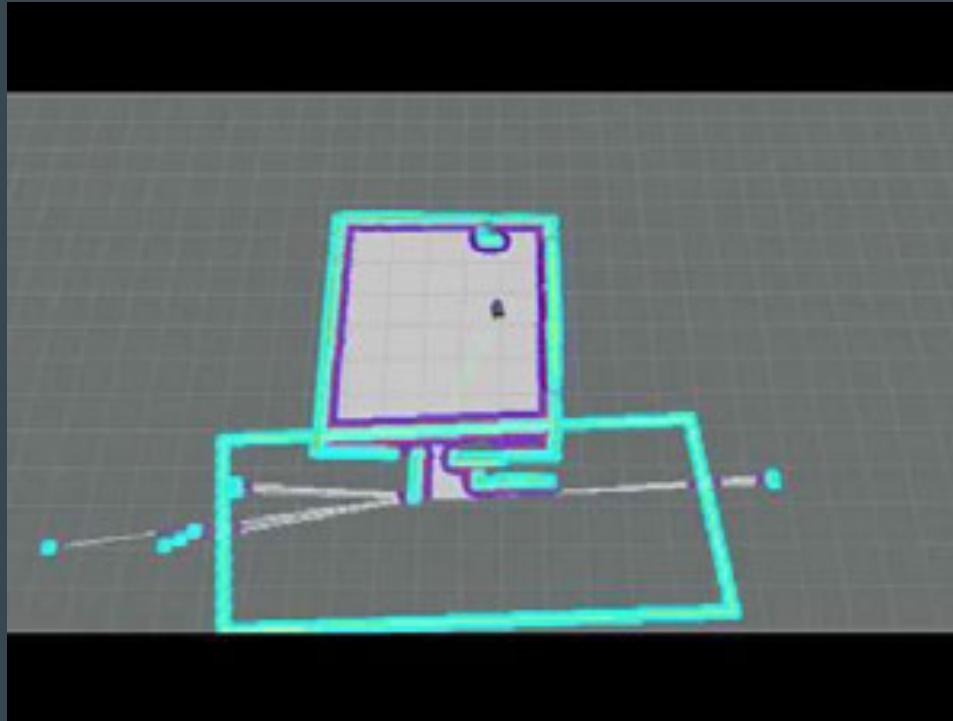
# Priority of a frontier cell

Based on:

- Distance
- Number of unknown cells adjacent to a frontier cell

# Frontier exploration

[http://wiki.ros.org/frontier\\_exploration](http://wiki.ros.org/frontier_exploration)

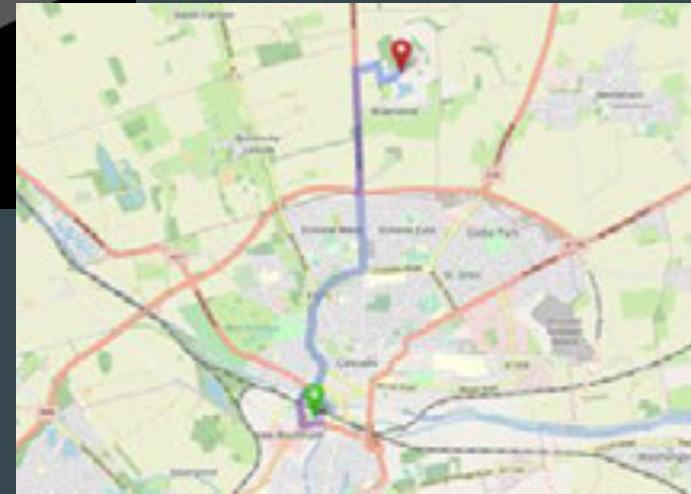
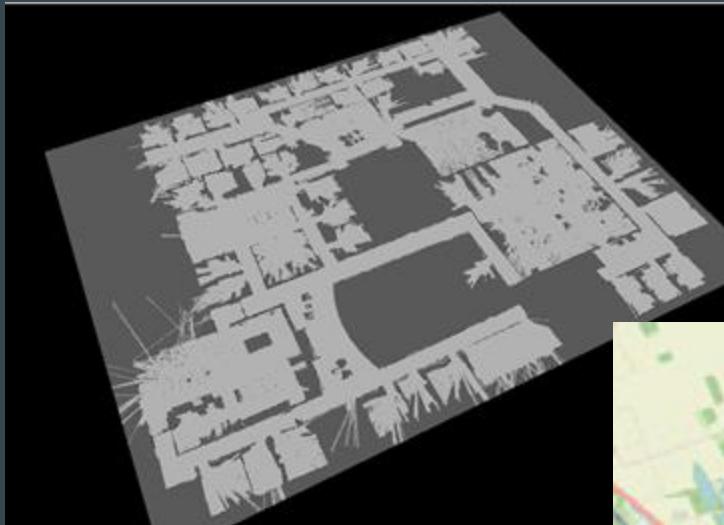


# Topological maps



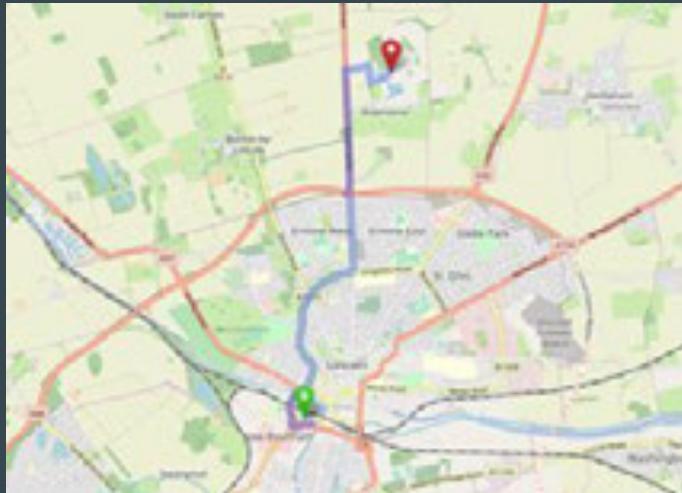
Represent known locations and connections between them as a set of nodes and edges in a graph

# Complex, large, structured environments?



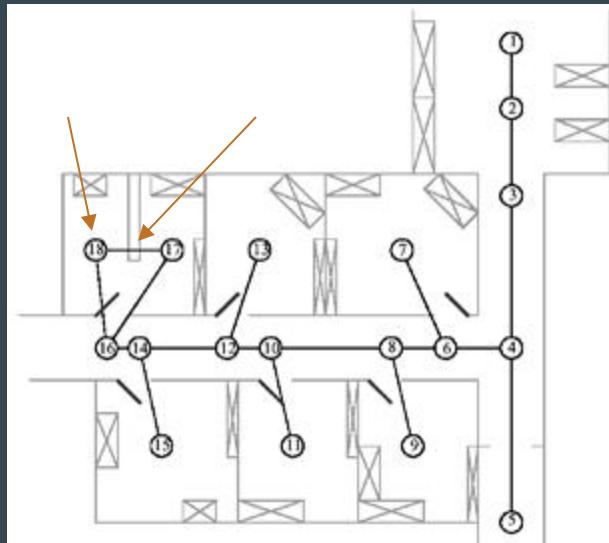
# Complex, large, structured environments?

- Specific navigation behaviours depending on the location
- Planning complexity over a grid map may be huge



# Topological maps

- Represents environment as a graph with nodes and edges
    - Nodes correspond to locations
    - Edge correspond to physical routes between locations
  - Lack scale and distances
    - Topological relationships (e.g., left, right) are maintained

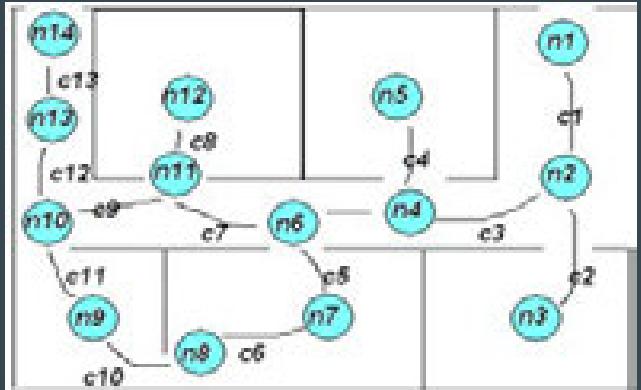


# Constructing a topological map

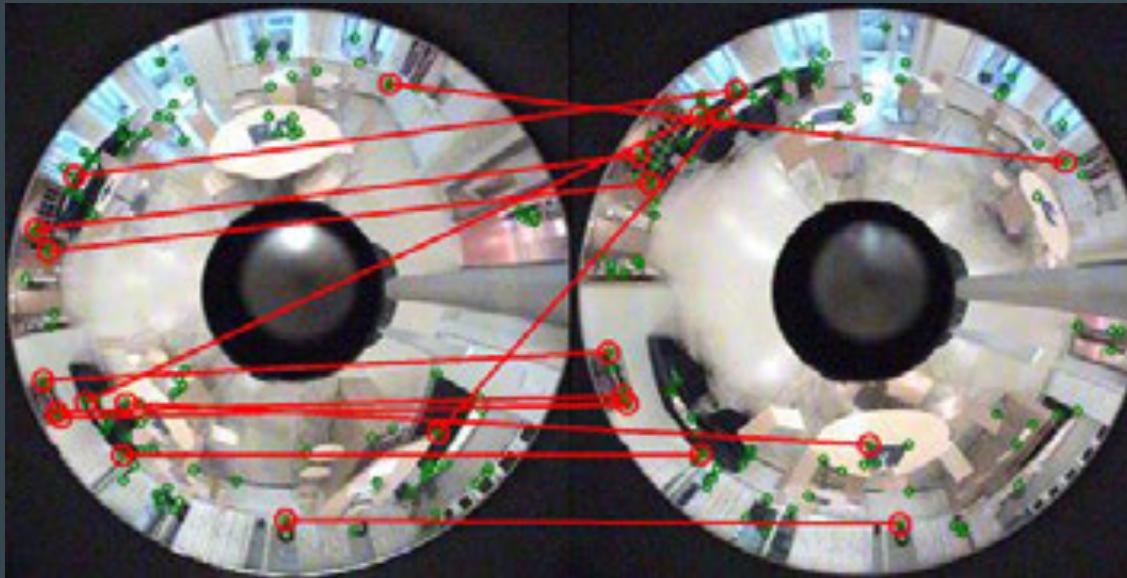
- New node = new place
- Two nodes are connected when travelling from one node to another (unless already connected)
- Localisation: use adjacency information in the graph
  - Given tracked position, search is limited to the nodes in the adjacency

# Example of topological mapping

- Represent the environment as a adjacency graph
- Each node corresponds to a certain place, and each link represents a traversable path
- A group of image features with their descriptors is used as a signature for the node
- A similarity score based on the number of matched points is used for localisation



# Topological localisation as image retrieval (using local features)



Matching is useful here for: 1) loop closing during topological mapping; 2) self-localisation in a previously acquired map

# Metric vs Topological

Metric maps

Topological maps

# Metric vs Topological

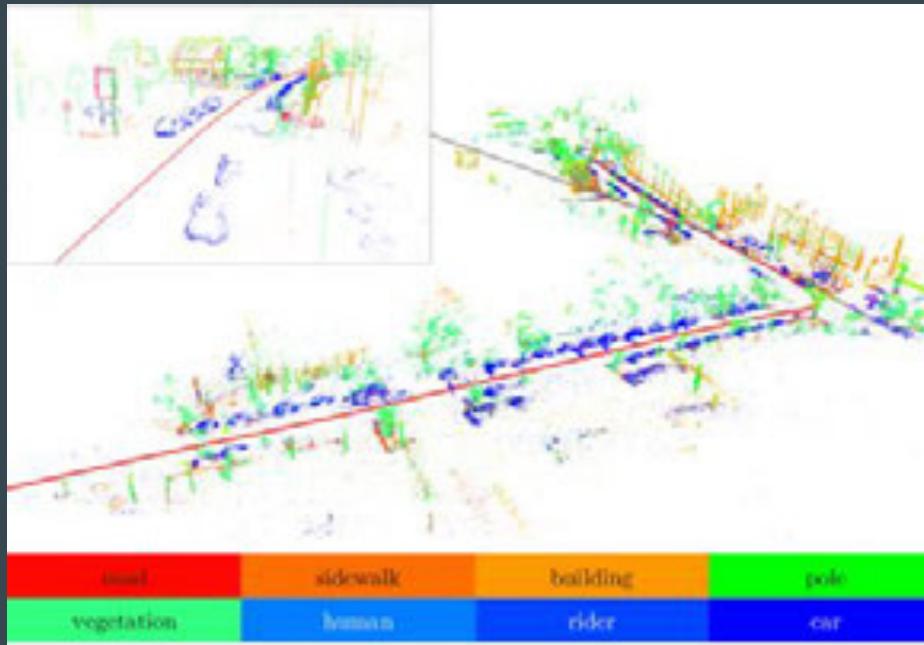
## Metric maps

- Detailed, quantitative, “sub-symbolic” representation
- Good for representing (and avoiding) known, static obstacles
- High computational cost of storage and processing
- Require very accurate position tracking – reliance on accurate odometry and range-finder sensors
- How to determine an appropriate resolution?

## Topological maps

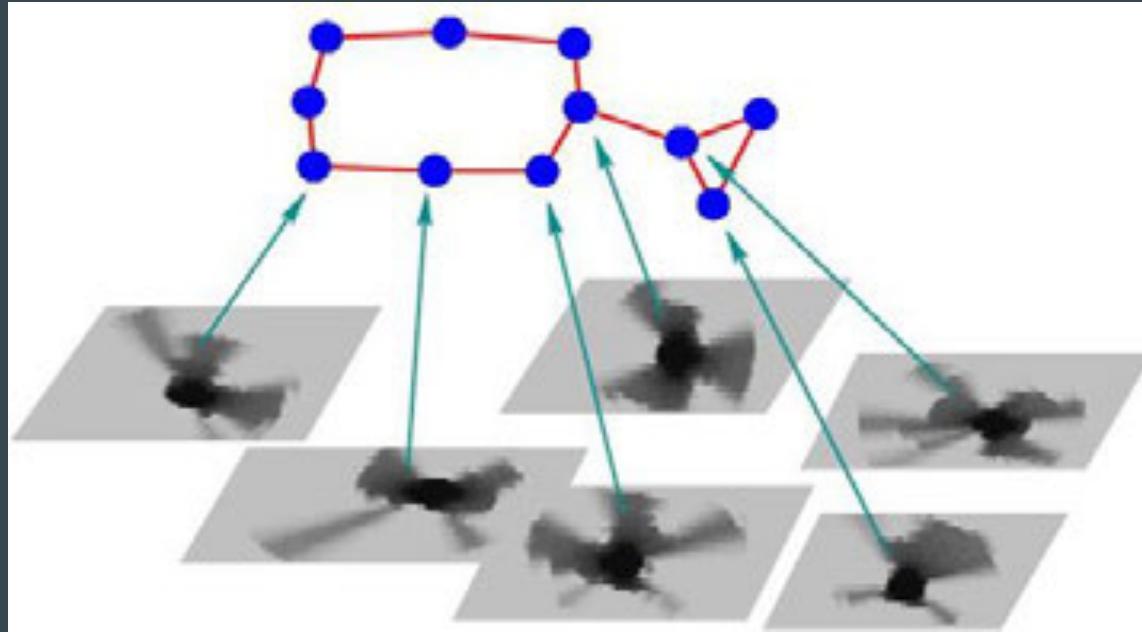
- Abstract, qualitative, “symbolic” representation
- May be more persistent/robust to environment dynamics
- Low computational cost - efficient path planning, scale better to large environments
- Require accurate place recognition - problem of perceptual aliasing (what if 2 or more places look alike?)
- How to determine what makes a “place” ?

# Semantic maps



Record of semantic information (metadata) – e.g. place/object names

# Hybrid maps

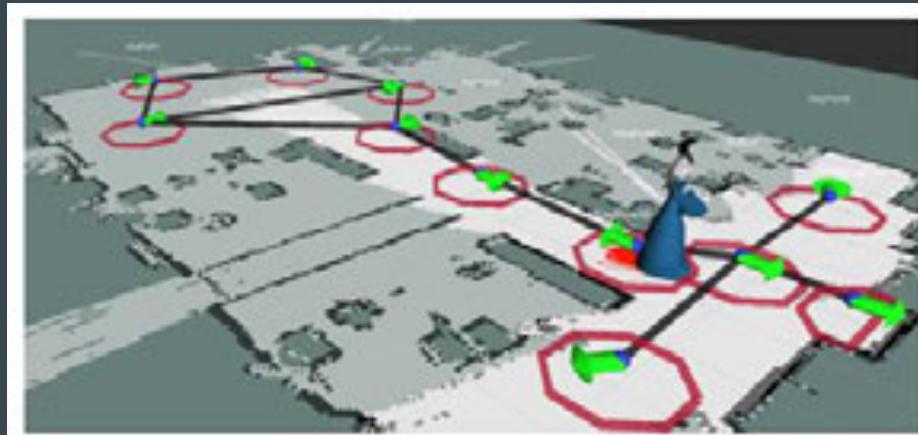


Combine complementary strengths of different representations (metric, topological, semantic maps)

# Hybrid maps

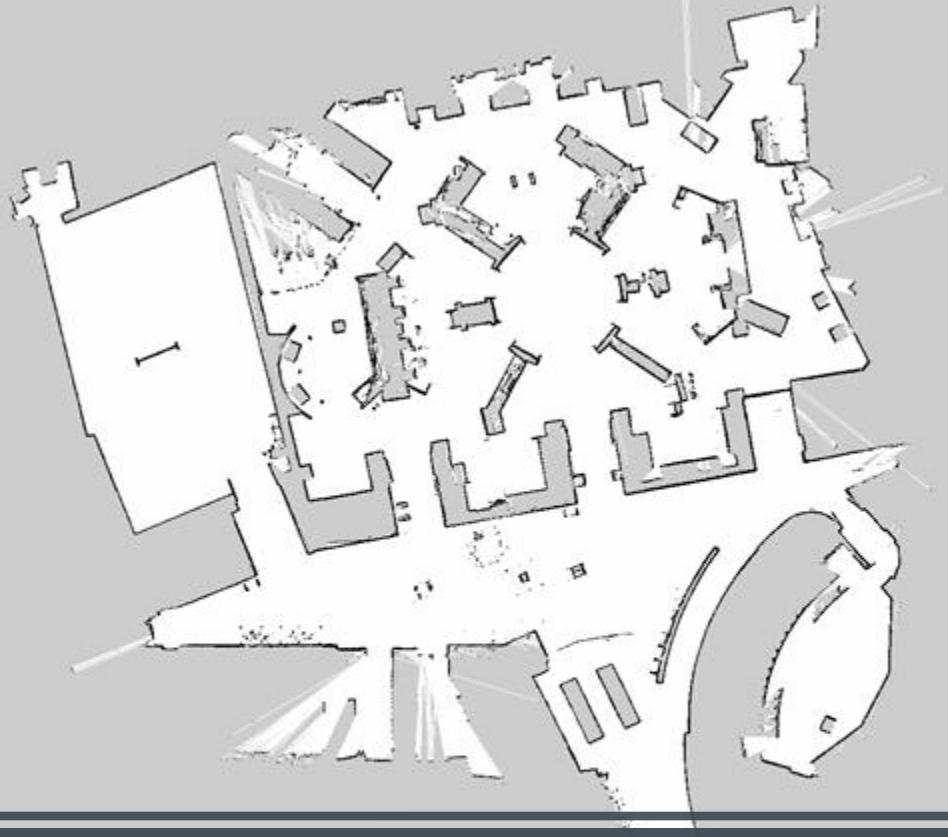
Example of a hybrid metric-topological map

- **Topological level:** connected set of places
- **Metric level:** each place is associated with a local metric map

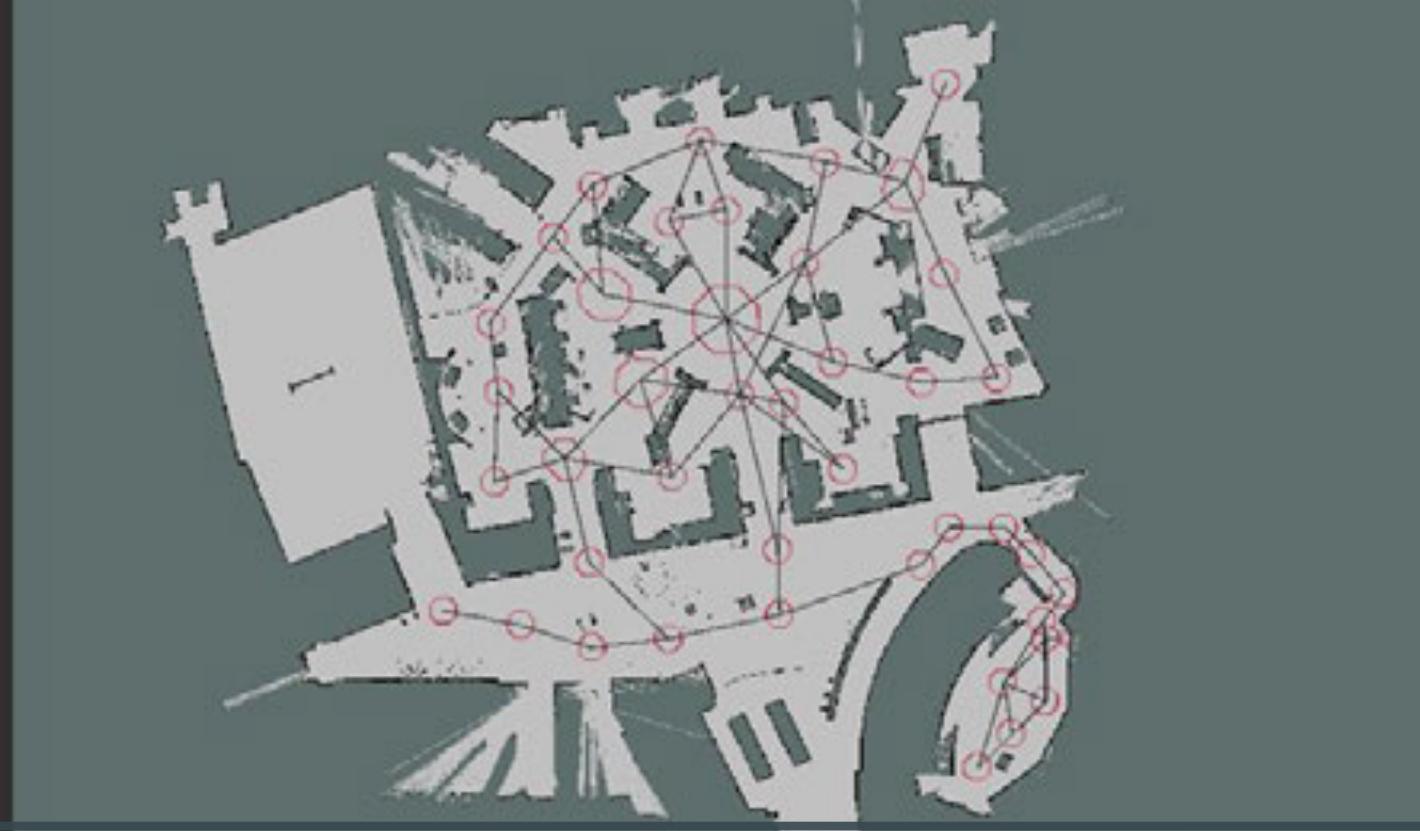


# Lindsey at The museum





Lindsey's obstacle map at The Collection

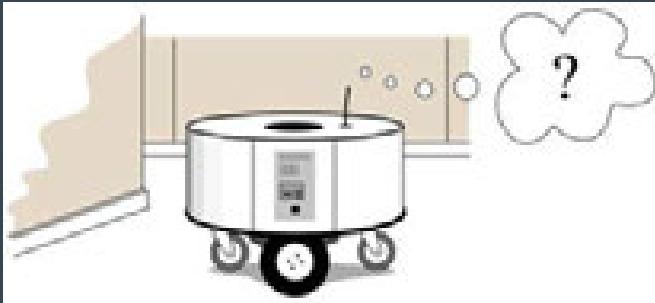


Lindsey's navigation map at The Collection

# Localisation

# Navigation – key questions

- Where am I?
- Where do I go?
- How do I get there?



To navigate successfully, a robot needs to:

- Perceive and understand the environment
- **Localise itself within the environment**
- Plan a route and execute that plan (motion control)

# Localisation approaches



Based on external  
sensors, beacons,  
landmarks



Odometry



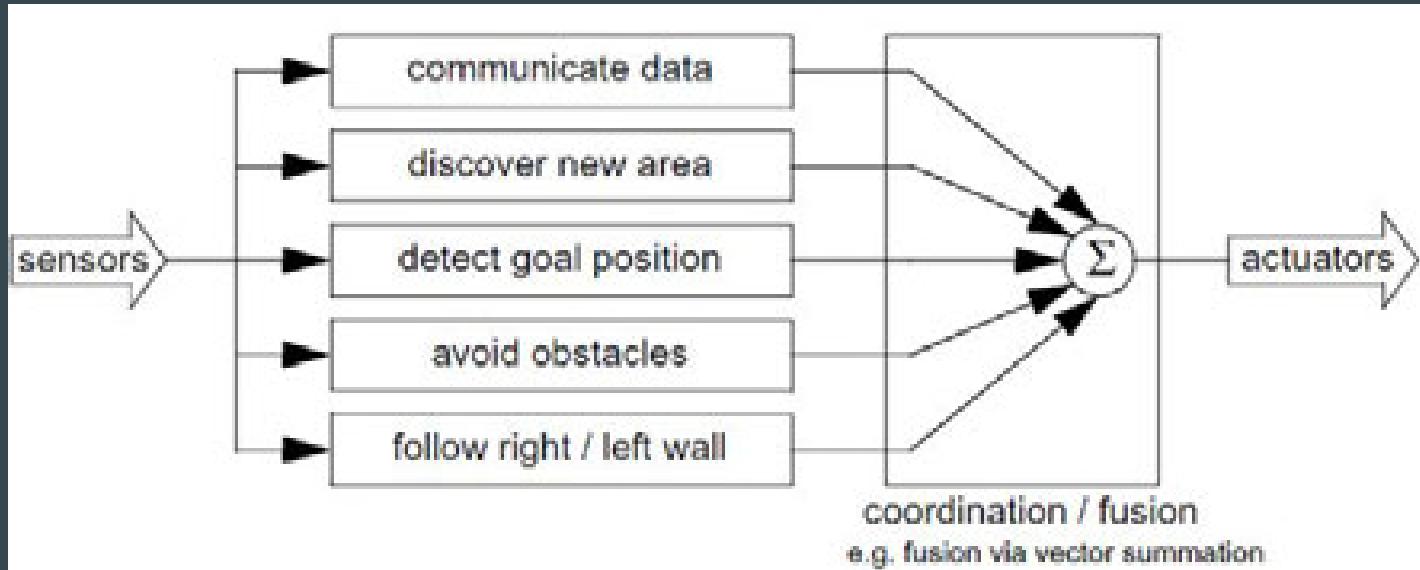
Map-based

# Odometry / dead reckoning

- Approximate location of a robot can be obtained by repeatedly computing the **distance moved**, and the **change in direction**, from the **velocity of the wheels** over a **short period of time**
- Also called **deduced reckoning** or **dead reckoning**
- Robot motion recovered by integrating proprioceptive sensor velocities readings
  - Advantages: straightforward
  - Disadvantages: errors are integrated (unbound)
- Heading sensors (e.g. IMU) help to reduce the accumulated errors, but drift remains

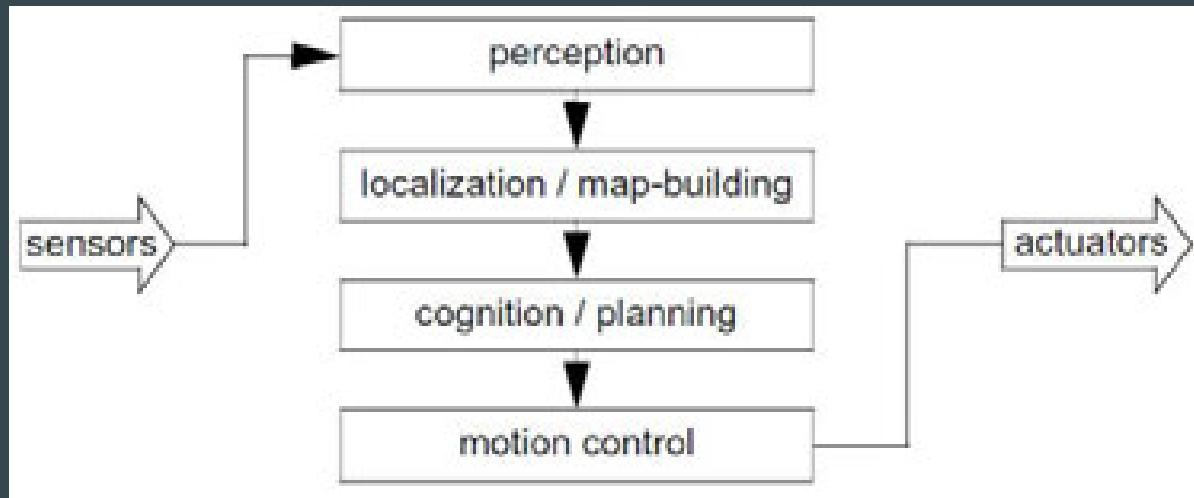
# To localise or not to localise

## Behaviour-based approach



# To localise or not to localise

## Map-based approach



# Map-based localisation

# Mobile robot self-localisation

Often divided into 3 main problems:

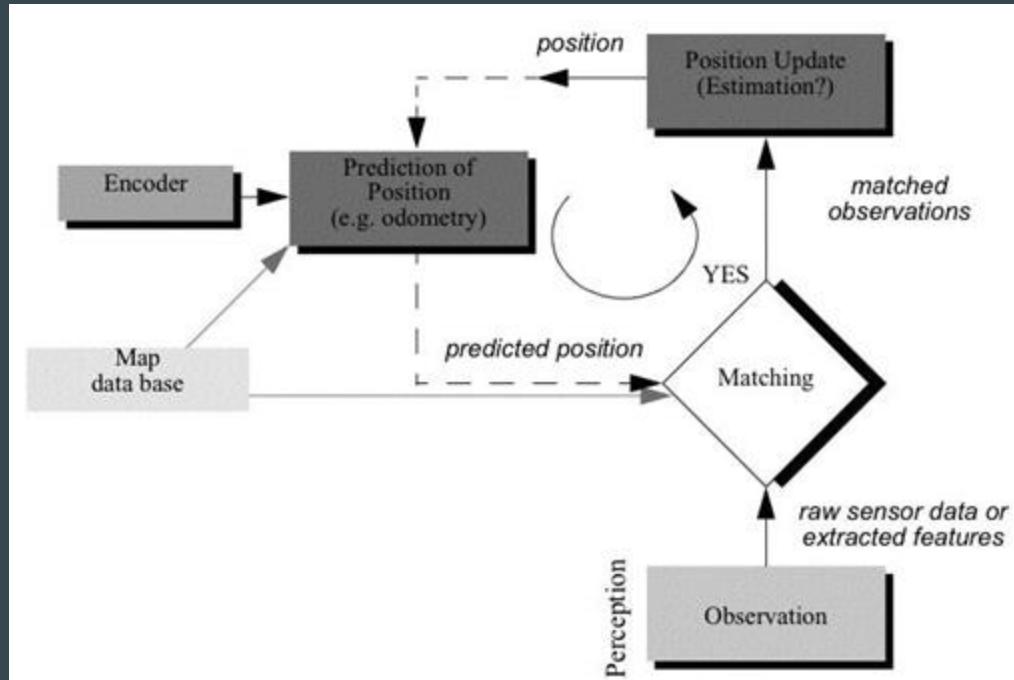
- Position tracking (good prior estimate)
- Global localisation (no prior estimate)
- “Kidnapped robot problem” (prior estimate is wrong)

Particle Filters can address all of these cases

- a.k.a. Monte Carlo localisation



# General process of map-based self-localisation



# Metric localisation

- Monte Carlo Localization
- Kalman Filter Localization
- Markov Localization

# What is a particle?

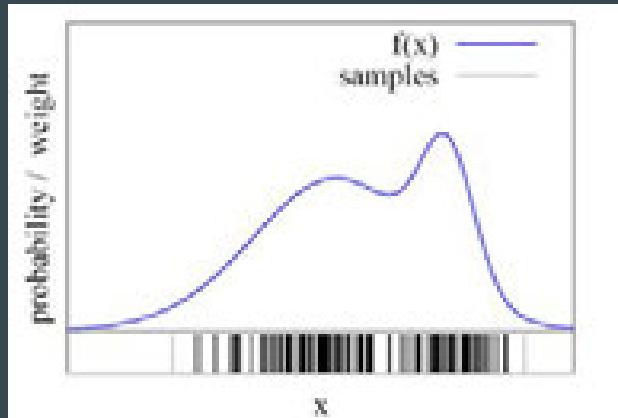
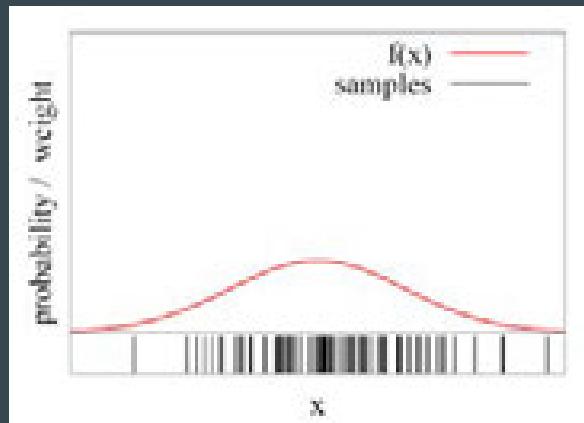
An individual state estimate, defined by:

- State values that determine robot's pose (position and orientation)
  - e.g.  $[x, y, \theta]$  for 2D self-localisation “in the plane”
- A weight that indicates its likelihood

Particle filters use many particles to represent the belief state

# Function approximation

- Particle sets can be used to approximate functions
- The more particles fall into an interval, the higher the probability of that interval



# Monte Carlo Localisation (MCL)

## Particle filter algorithm – main steps

- Initialisation
  - Sample from initial distribution
  - No idea where robot is – throw particles everywhere
- For each time step, loop with three phases:
  - Prediction
  - Update
  - Resample

# Monte Carlo localisation

## Perception update

- Robot queries its sensors, and finds itself next to a pillar

## Prediction update

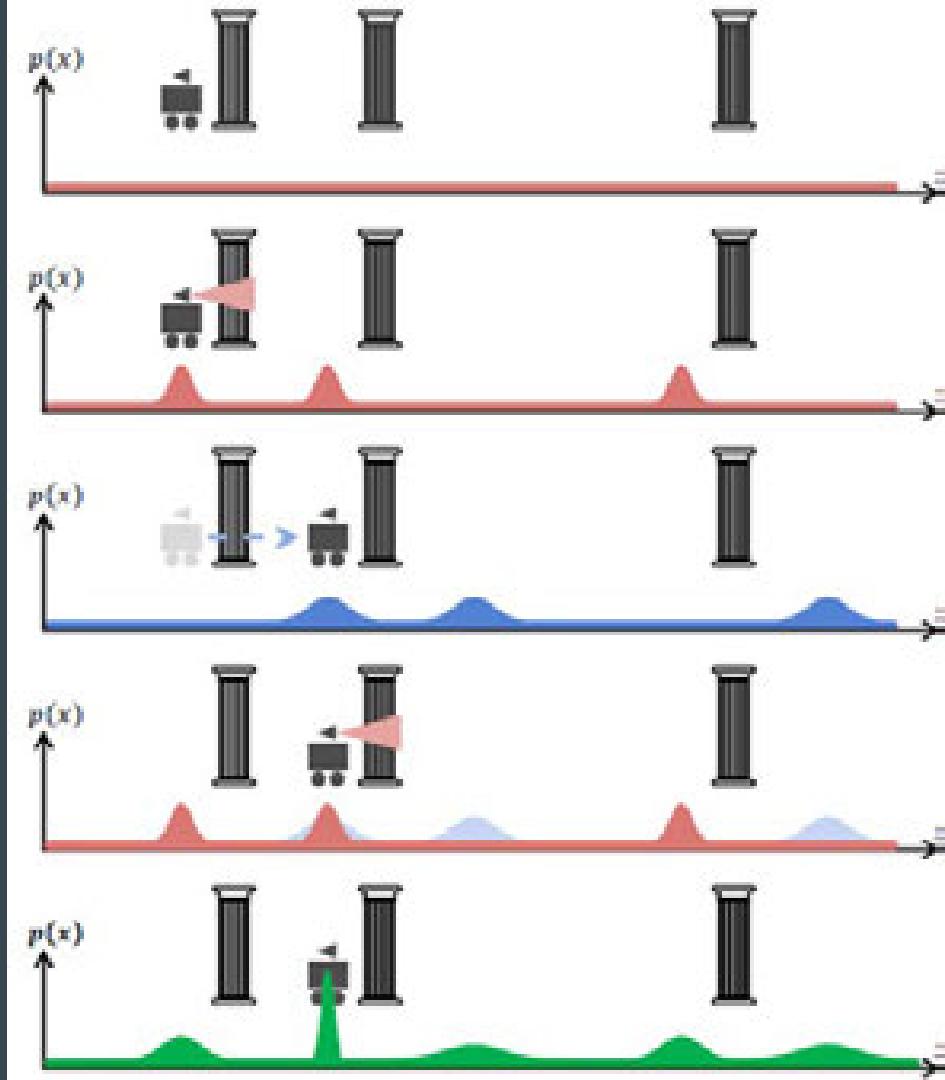
- Robot moves one metre forward
- Motion estimated by wheel encoder – accumulation of uncertainty

## Perception update

- Robot queries its sensors, and finds itself next to a pillar

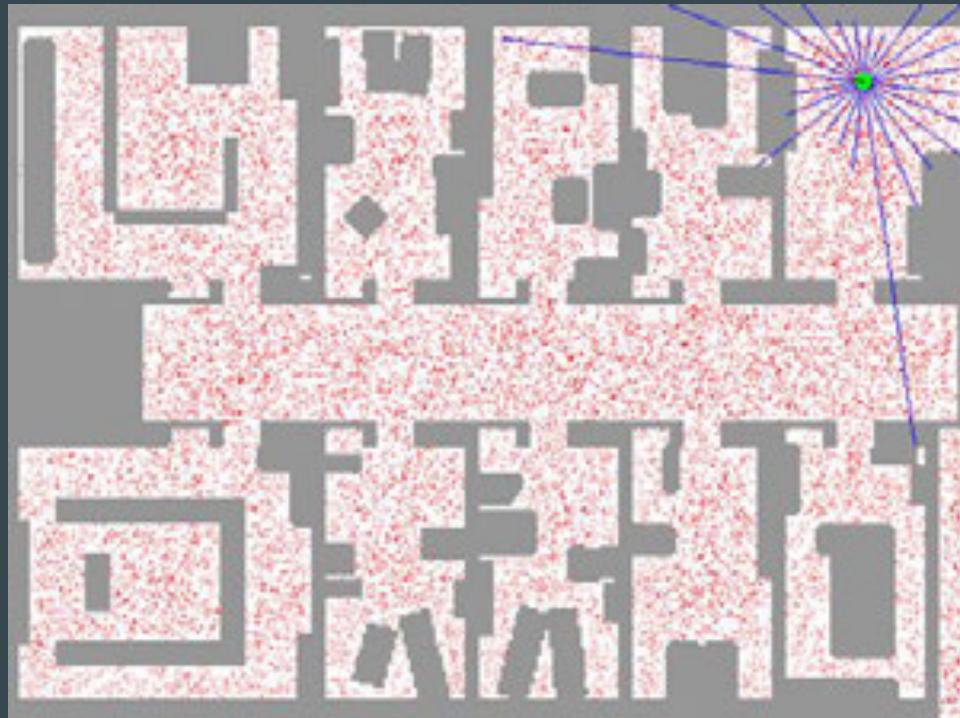
## Belief update

- Information fusion



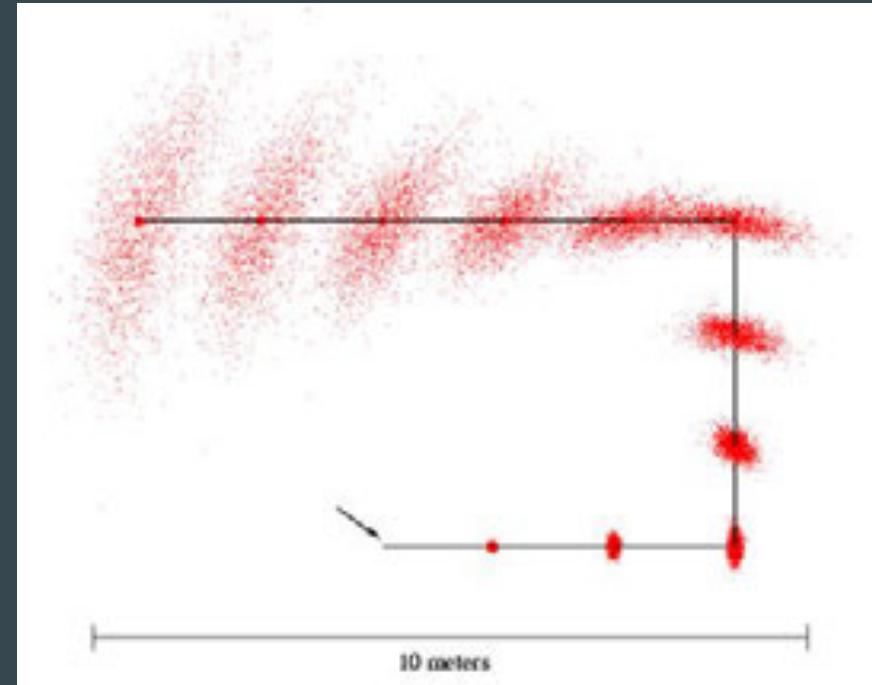
# Randomised sampling

## 2D Monte Carlo localisation

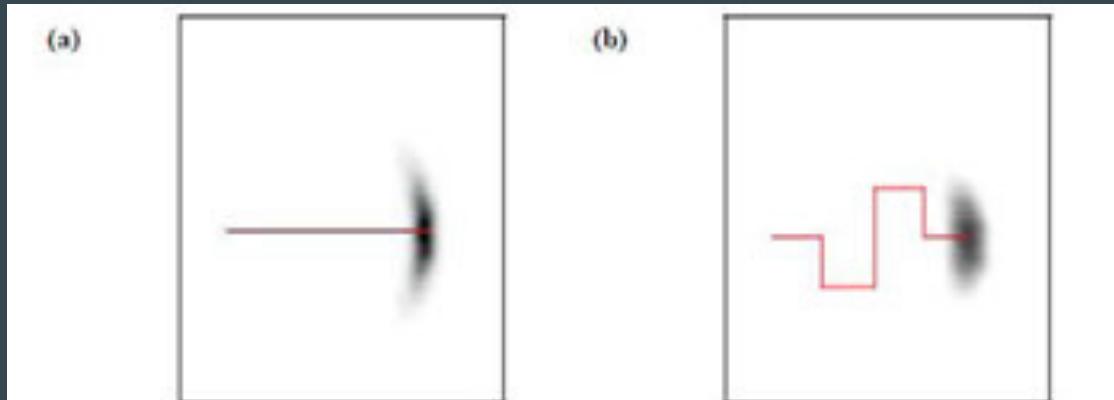


# Prediction step

- For each particle
- Sample and add random noisy values from the motion model



# Motion model

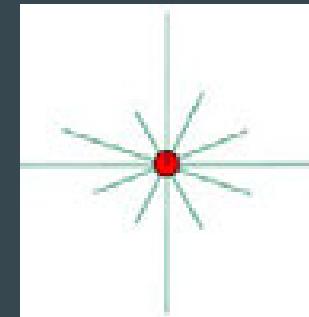


**Figure 5.2** The motion model: Posterior distributions of the robot's pose upon executing the motion command illustrated by the solid line. The darker a location, the more likely it is. This plot has been projected into 2D. The original density is three-dimensional, taking the robot's heading direction  $\theta$  into account.

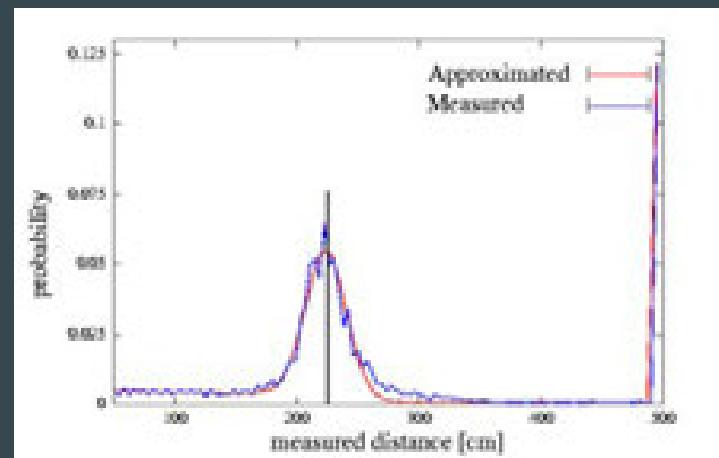
S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

# Update step

- Each particle's weight is the likelihood of getting the current sensor readings from that particle's hypothesis
- Compared to the predicted readings from the map

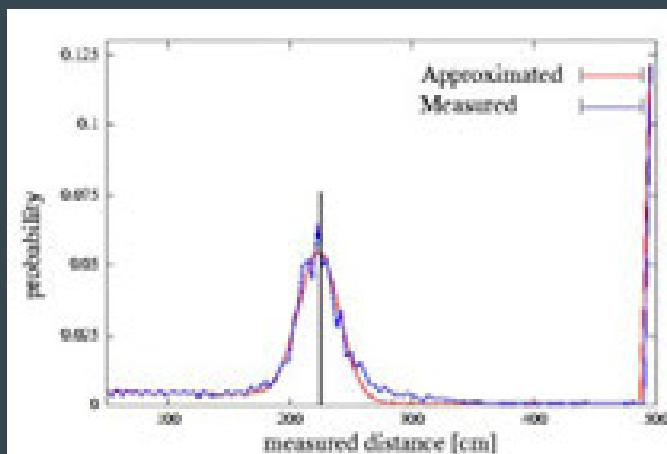


Laser sensor

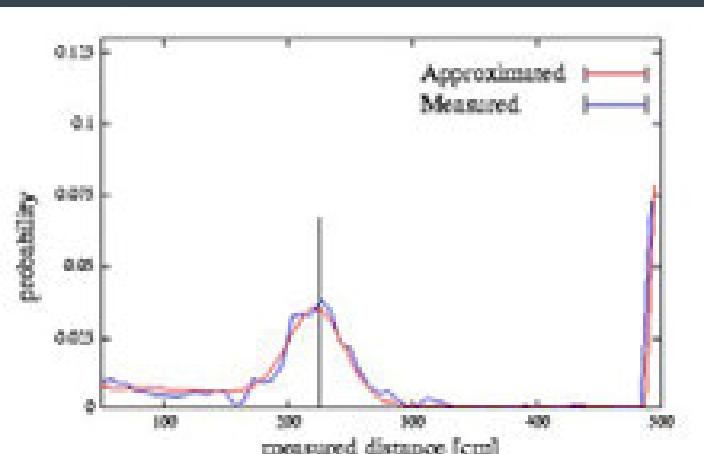


# Sensor model

- How likely are the current sensor measurements compared to what the map says?



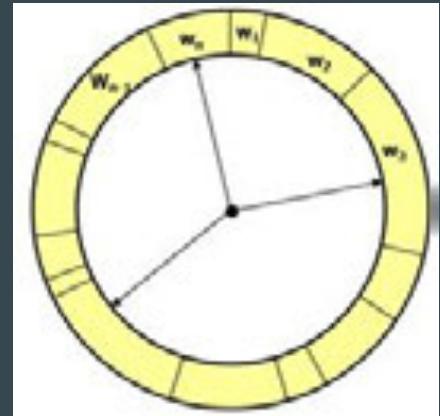
Laser  
sensor



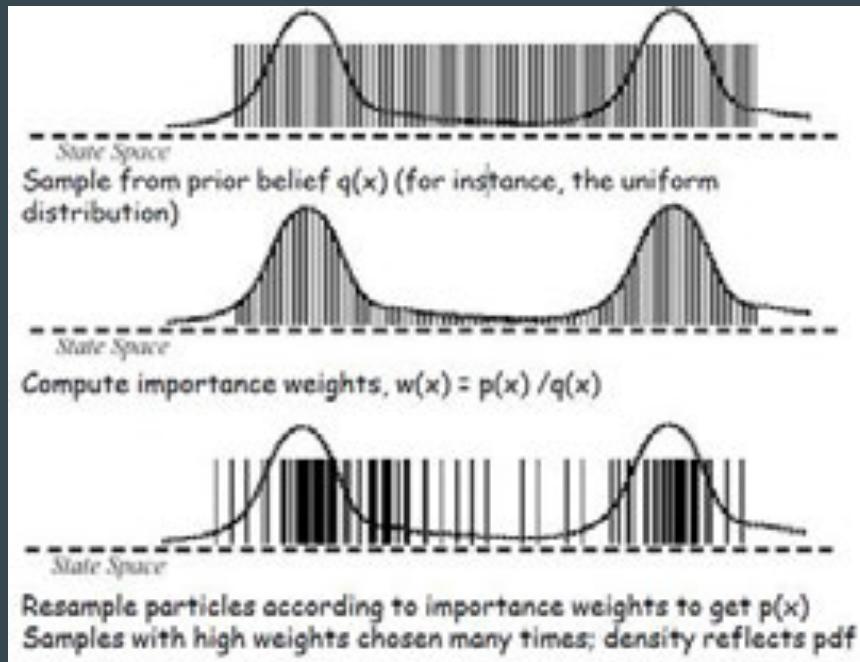
Sonar  
sensor

# Resample step

- New set of particles chosen
- “Survival of the fittest”
  - Each particle survives in proportion to its weight
  - Replace unlikely samples by more likely ones
- “Roulette wheel” resampling



# Resampling



# Particle filter algorithm

- We approximate  $P(x_t)$  by a set of samples:
  - $P(x_t) = \{x_t^{(i)}, w_t^{(i)}\}_{i=1,\dots,m}$
- Each  $x_t^{(i)}$  is a possible value of  $x$ , and each  $w_t^{(i)}$  is the probability of that value (also called an importance factor)
- Initially, we have a set of samples (typically uniform) that give us  $P(x_0)$
- Then we update with the following algorithm

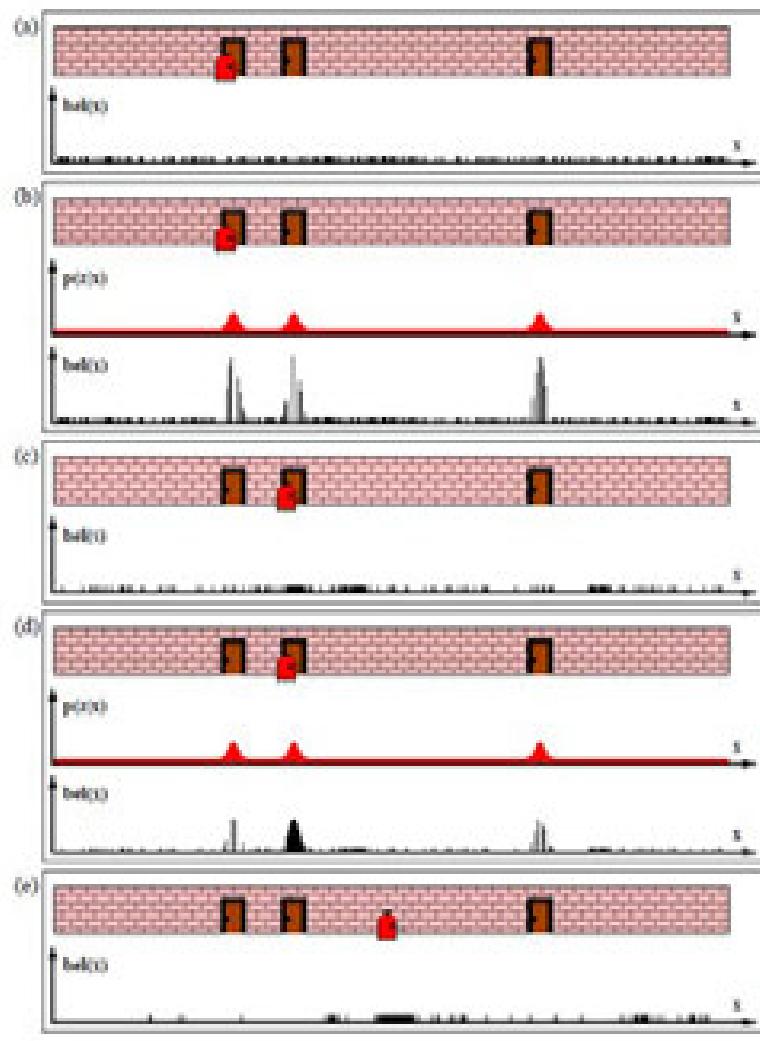
# Particle filter algorithm

```
 $x_{t+1} = \emptyset$ 

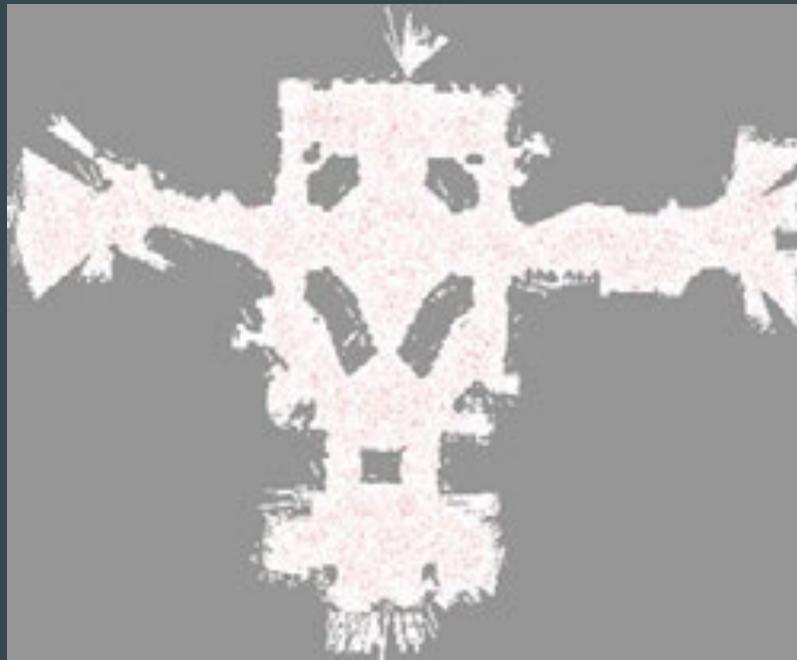
for  $j = 1$  to  $m$ 
    // apply the transition model
    generate a new sample  $x_{t+1}^{(j)}$  from  $x_t^{(j)}$ ,  $a_t$  and  $\Pr(x_{t+1} | x_t, a_t)$ 
    // apply the sensor model
    compute the weight  $w_{t+1}^{(j)} = \Pr(e_{t+1} | x_{t+1})$ 

    // pick points randomly but biased by their weight
for  $j = 1$  to  $m$ 
    pick a random  $x_{t+1}^{(i)}$  from  $x_{t+1}$  according to  $w_{t+1}^{(1)}, \dots, w_{t+1}^{(m)}$ 

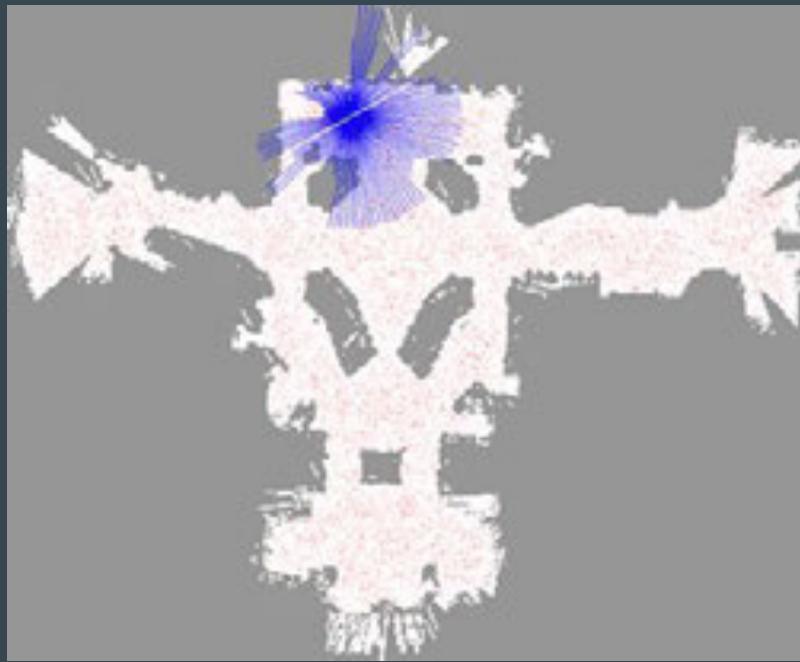
normalize  $w_{t+1}$  in  $x_{t+1}$ 
return  $x_{t+1}$ 
```



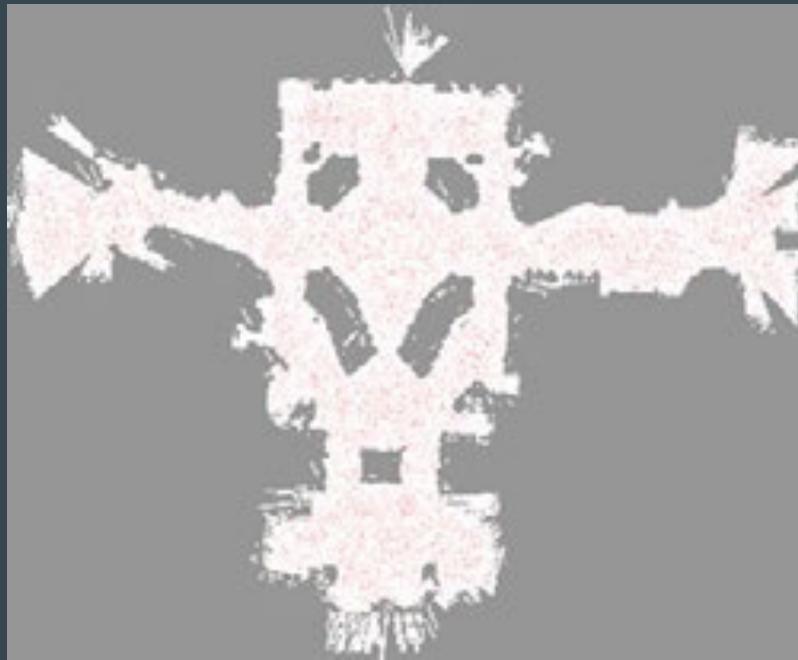
# Initialisation



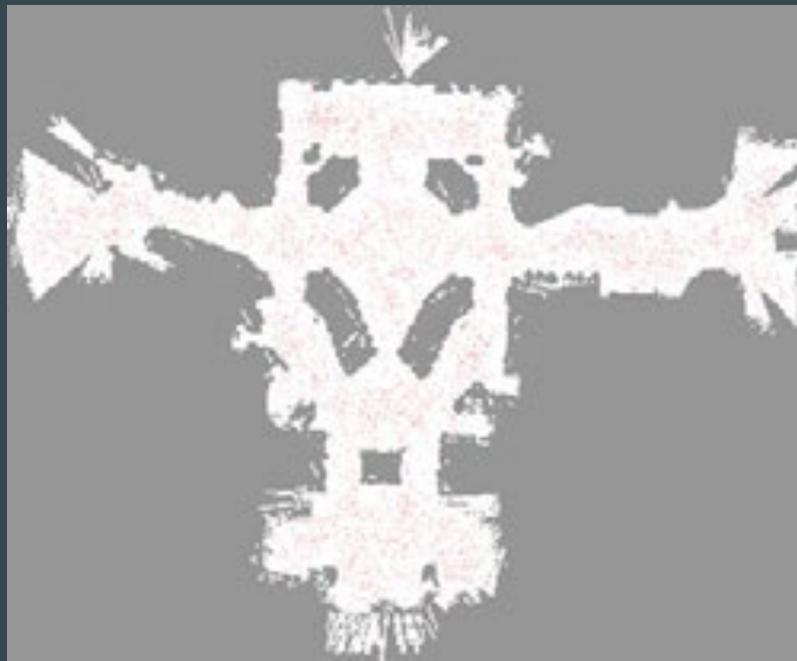
# Measurement



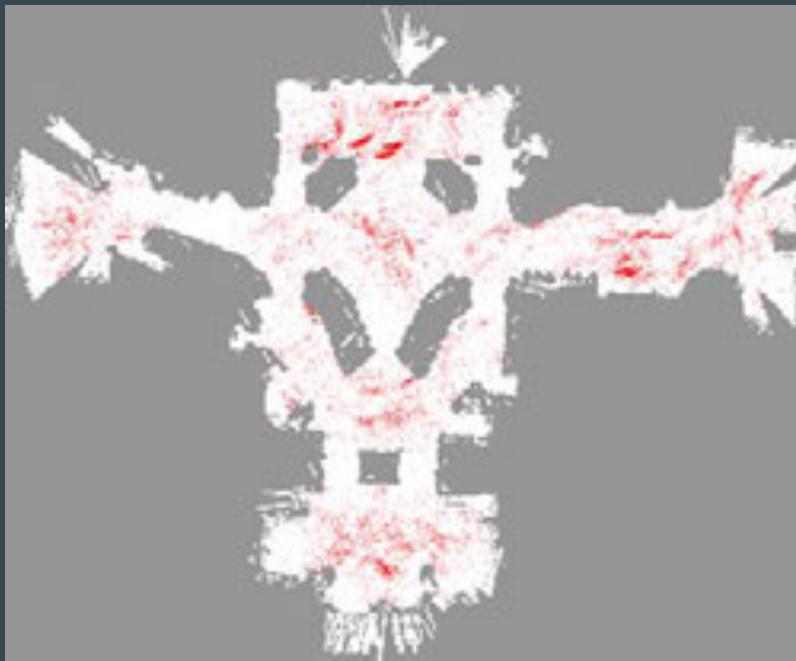
# Weight update



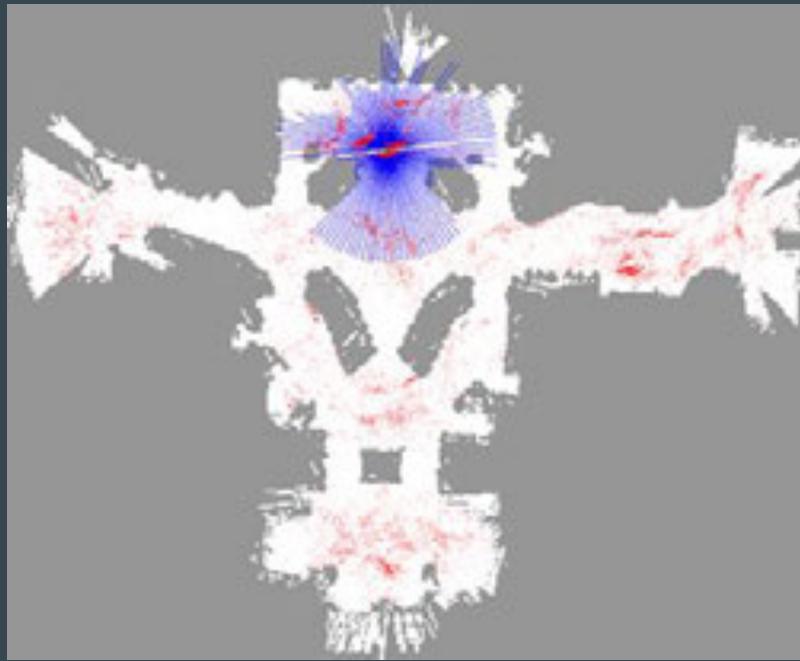
# Resampling



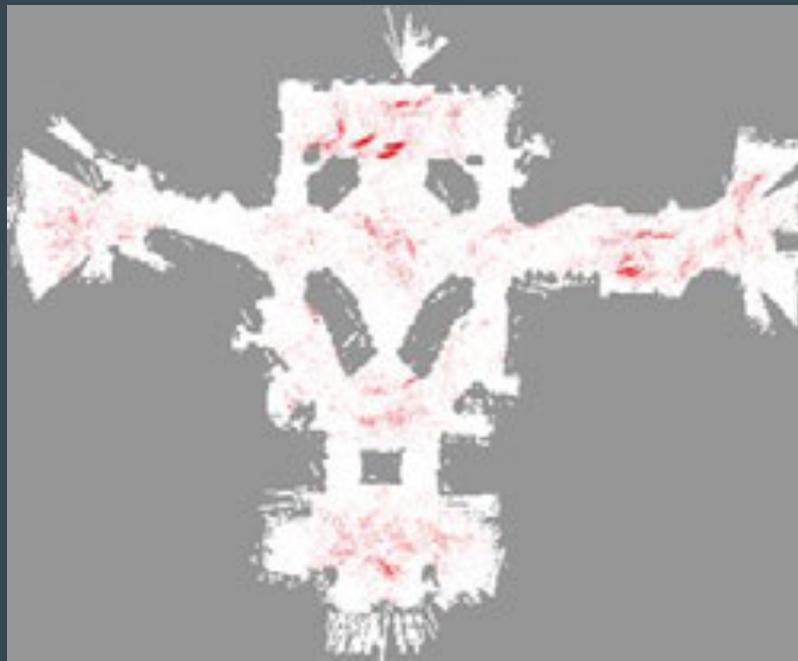
# Motion update



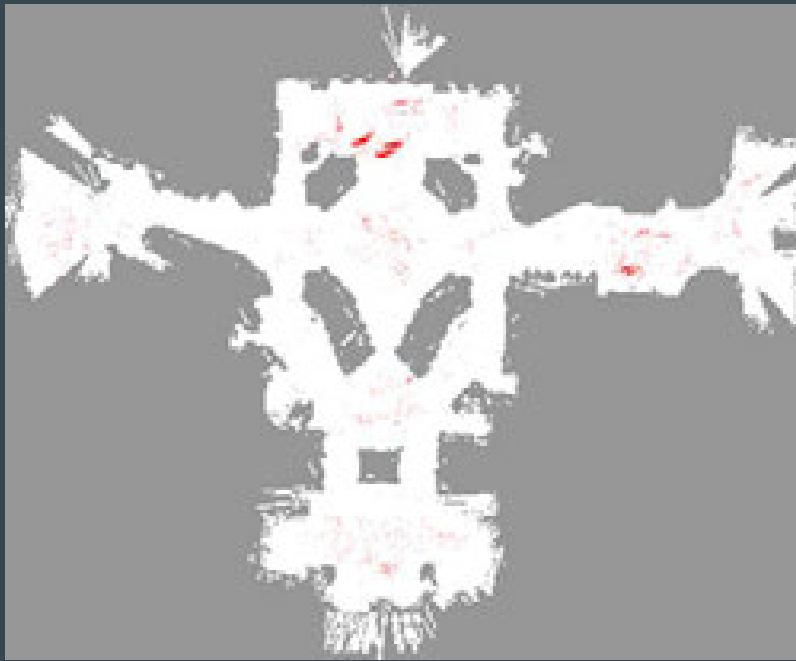
# Measurement



# Weight update



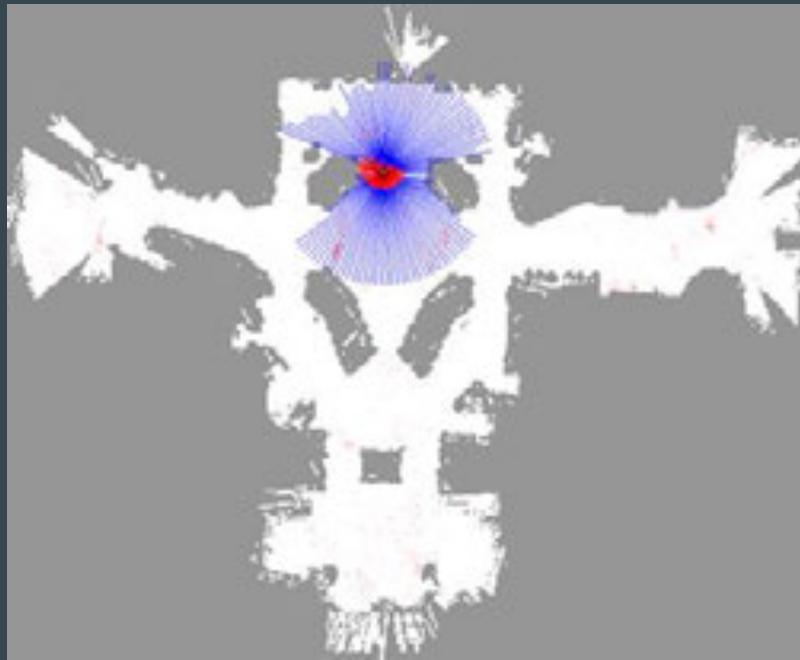
# Resampling



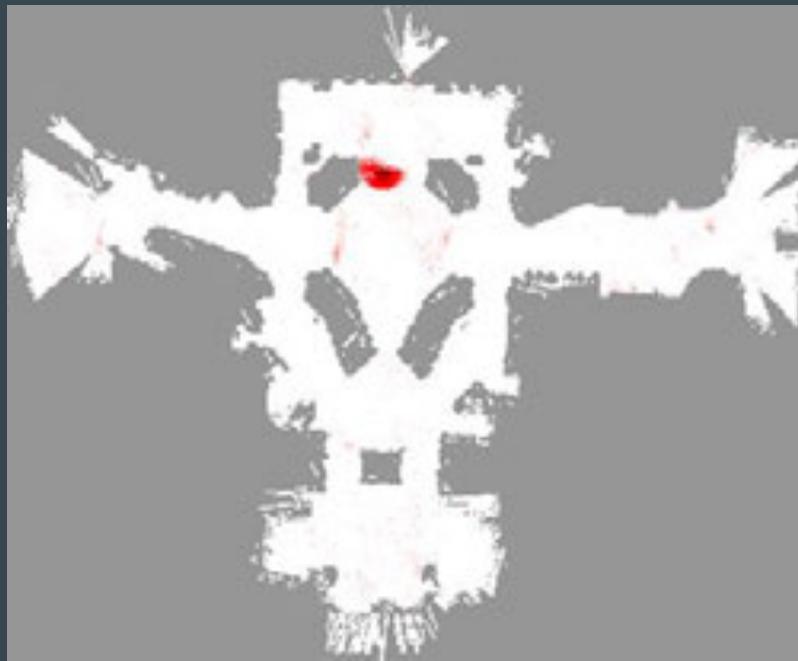
# Motion update



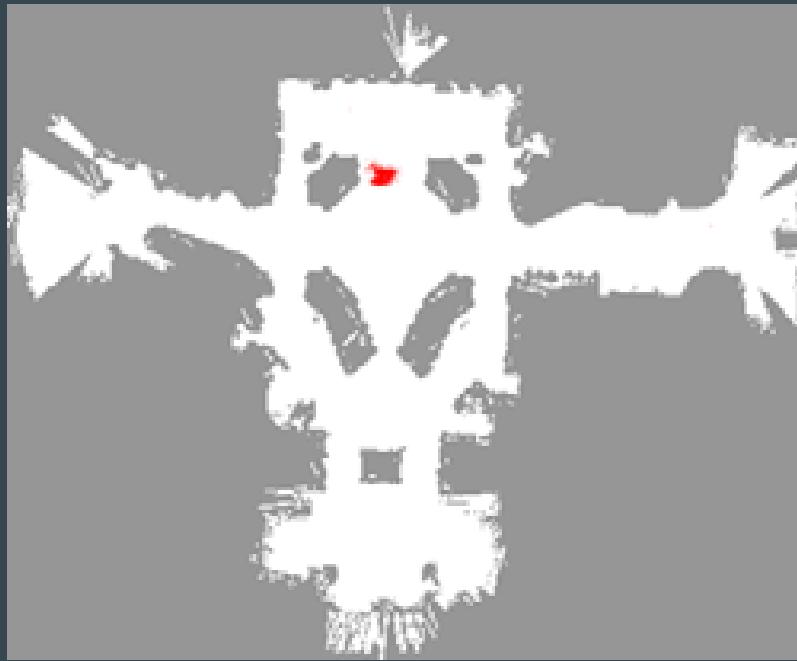
# Measurement



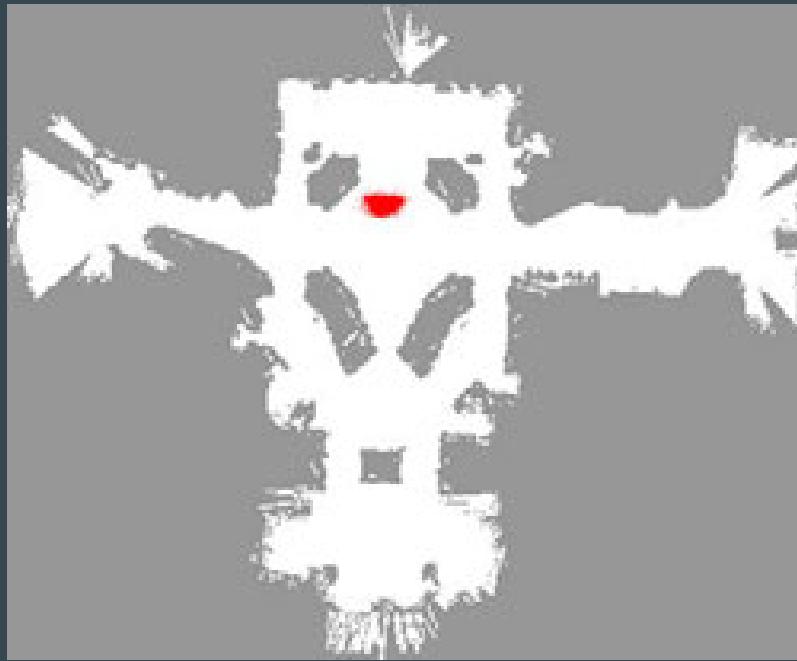
# Weight update



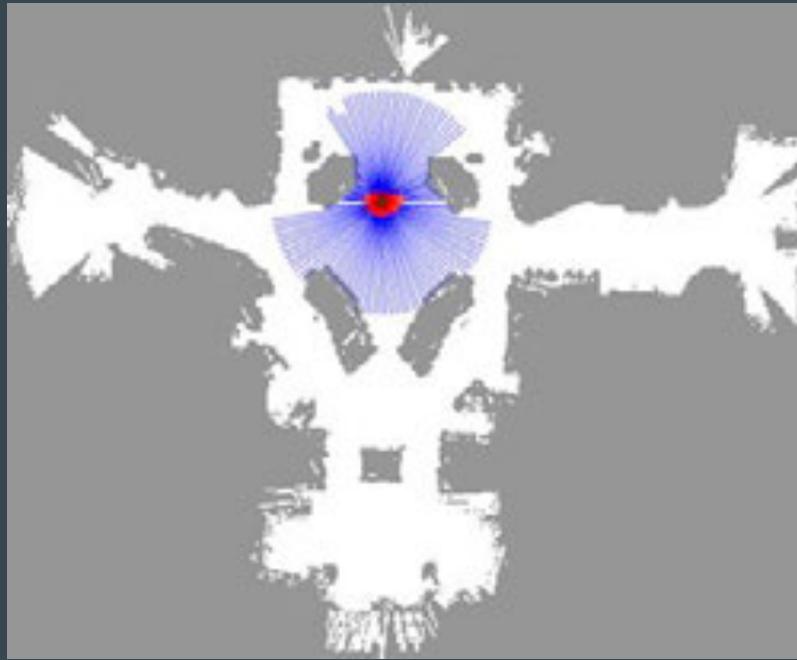
# Resampling



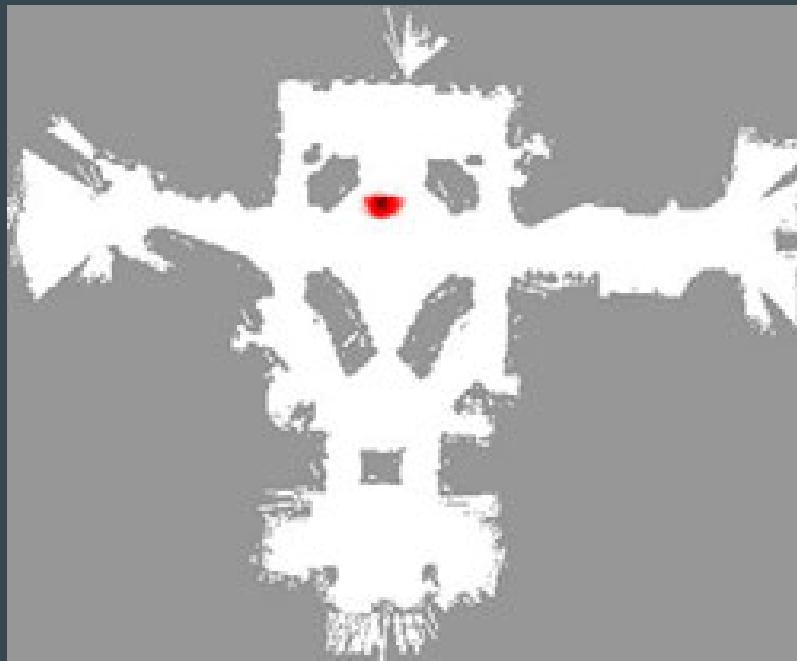
# Motion update



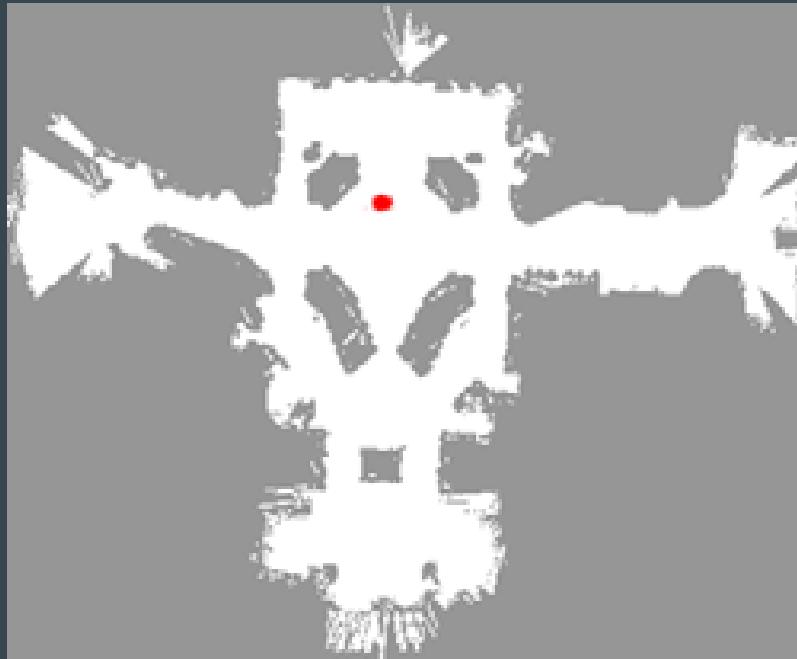
# Measurement



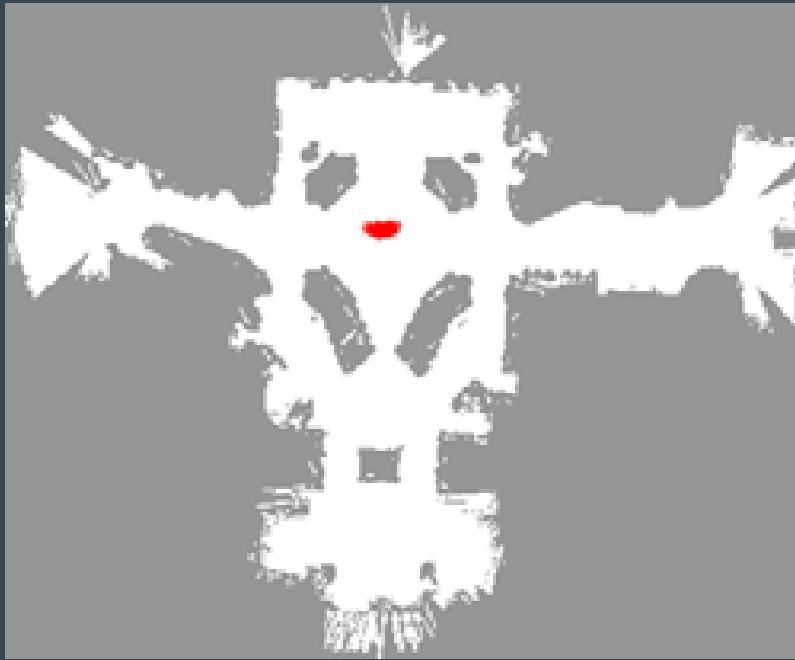
# Weight update



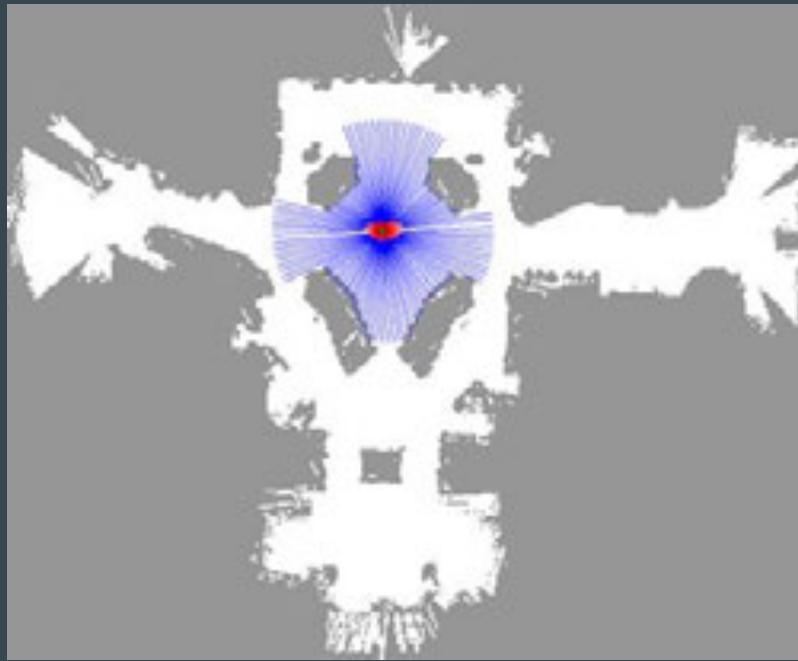
# Resampling



# Motion update



# Measurement



# Monte Carlo localisation summary

- Particle Filters (PFs) can represent arbitrary probability density functions (distributions) using samples
- PFs use sample importance resampling, with 4 main steps:
  - Initialisation
  - Predict
  - Update
  - Resample
- PFs can solve the “kidnapped robot problem” and handle perceptually aliased environments
- Also quite easy to implement

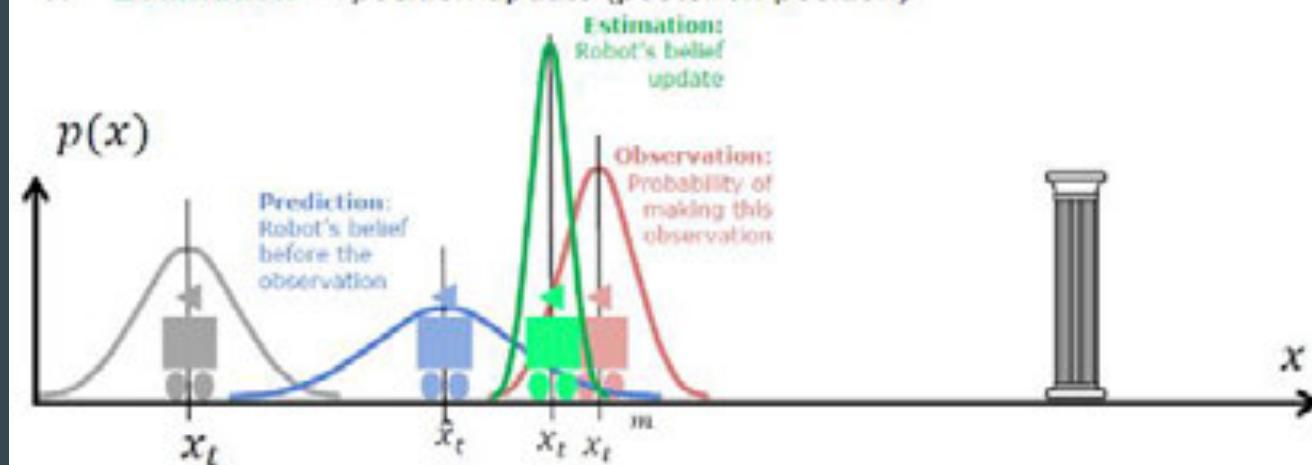


# Kalman Filter (KF) localisation

- Instead of an arbitrary density function, KF uses Gaussians for robot belief, motion and measurement models
- Only mean and covariance need to be updated – efficient computation
- Since initial belief is also Gaussian, initial position shall be known with a certain approximation
- KF localisation addresses position tracking, not global localisation or kidnapped robot problem

# Kalman Filter (KF) localisation

1. **Prediction (ACT)** based on previous estimate and odometry
2. **Observation (SEE)** with on-board sensors
3. **Measurement prediction** based on prediction and map
4. **Matching** of observation and map
5. **Estimation** → position update (posteriori position)



# Kalman Filter localisation pros / cons

Uses a Gaussian probability density representation of robot position and scan matching for localization

## Pros:

- Tracks the robot from an initially known position
- Precise and efficient
- Can be used in continuous world representations

## Cons:

- If uncertainty of robot becomes large (e.g. collision with an object):
- It can fail to capture the multitude of possible robot positions and can become irrevocably lost

# Markov localisation

- Applies the same ideas to a discrete map
- Motion model only needs to consider transitions between discrete locations
- Sensor model is also discrete
- **Markov localization addresses position tracking, global localization and kidnapped robot problem**

# Markov localisation pros / cons

Uses an explicitly specified probability distribution across all possible robot positions

## Pros:

- Allows for localisation starting from any unknown position
- Can recover from ambiguous situations

## Cons:

- Requires a discrete representation of the space, such as a geometric grid or a topological graph
- Consumes significant memory and computational resources

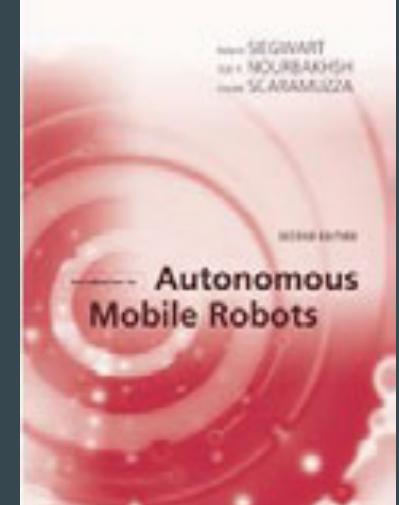
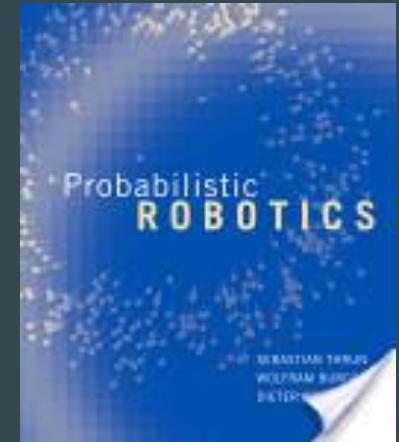
# Summary

## Learning Outcomes:

- Describe different types of Maps for localization
  - Metric
  - Topological
  - Semantic
  - Hybrid
- Explain various localization methods
  - Monte Carlo
  - Kalman Filter
  - Markov

# Recommended reading

- S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005. Chapter. 4
- Siegwart R. et al., *Autonomous Mobile Robots*, (MIT Press). Chapter 5.



# CMP3103M Autonomous Mobile Robotics

•••

## Lecture 8: SLAM and Motion Planning

Dr Athanasios Polydoros

# Today's lecture

- Quiz
- Simultaneous Localisation and Mapping (SLAM)
  - Gaussian filter and particle filter
- Motion planning
  - Problem representation
  - Graph search
  - Potential fields

# Interactive Quiz:

Join:



<https://pollev.com/athanasiospolydoros472>

# Topological Maps contain a detailed quantitative representation of the environment

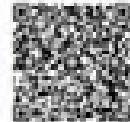


True

0%

False

0%



## Metric Maps have a high computational cost for storage and processing

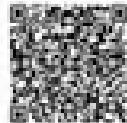
0

True

0%

False

0%



Which of the following type of maps are the best to represent static obstacles

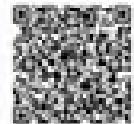
0

Metric

0%

Topological

0%



## What each particle represents in the particle filter algorithm

0

The state of the robot

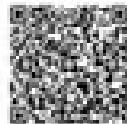
0%

The state of the environment

0%

The state of the obstacles

0%



Which of the following should we use if the initial position of the robot is known

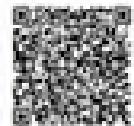


Particle Filter

0%

Kalman Filter

0%



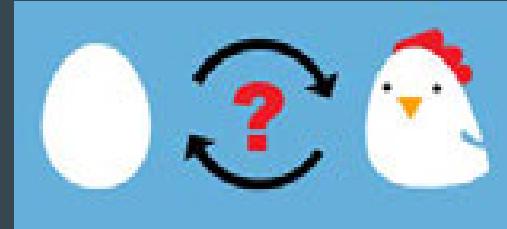
# Simultaneous Localisation and Mapping (SLAM)

# Autonomous map building

Starting from an arbitrary initial point, a mobile robot should be able to:

- Autonomously explore the environment with its on-board sensors
- Gain knowledge about it
- Interpret the scene
- Build an appropriate map
- Localise itself relative to this map

# Chicken or the egg?



- Making maps is a “chicken or egg” problem
- If we don’t know where we are, we cannot make a good model
- If we don’t have a good model, we cannot know where we are
- Solution is known as Simultaneous Localisation and Mapping
  - Commonly referred to as SLAM

# The SLAM problem

- How can a robot navigate a previously unknown environment, while constantly building and updating a map of its workspace using on-board sensors and computation?
- **Simultaneous Localisation and Mapping** required

# When is SLAM necessary?

- When a robot must be truly autonomous (no human input)
- When there is no prior knowledge about the environment
- When we cannot rely exclusively on external positioning systems (e.g. GPS)
- When the robot needs to know where it is

# Why is self-localisation needed?

Example of mapping using odometry



Example of simultaneous localisation and mapping



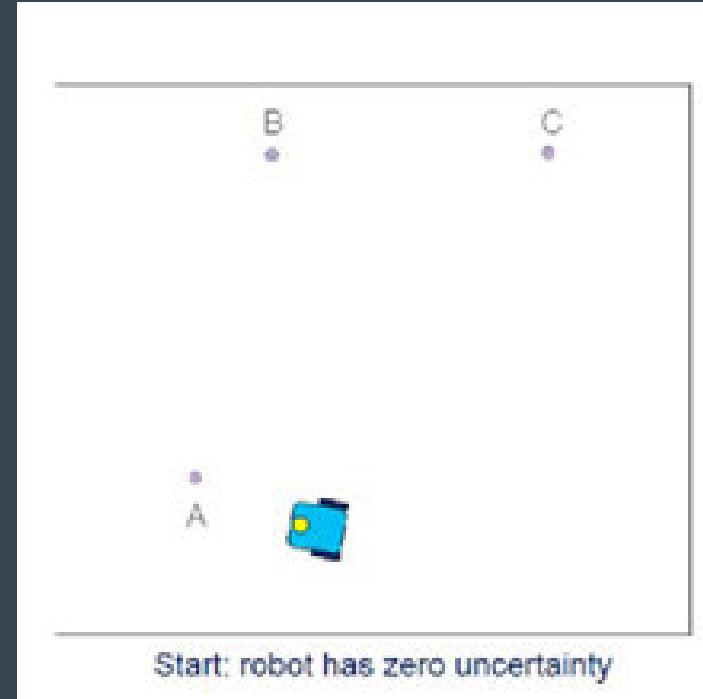
# SLAM with a Gaussian filter

Use internal representations for:

- The positions of landmarks
- The camera parameters

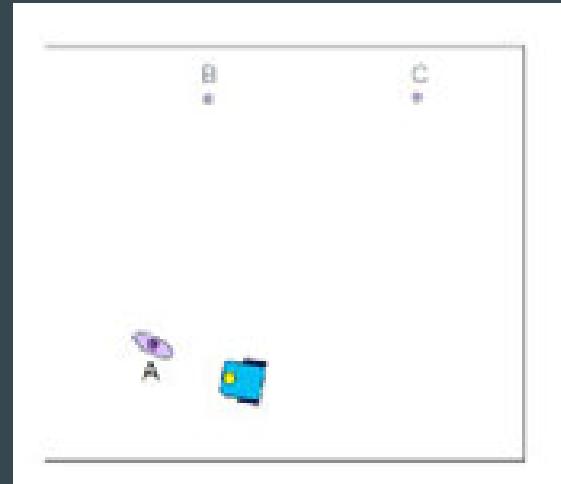
Assumption:

- Robot's uncertainty at starting point is zero



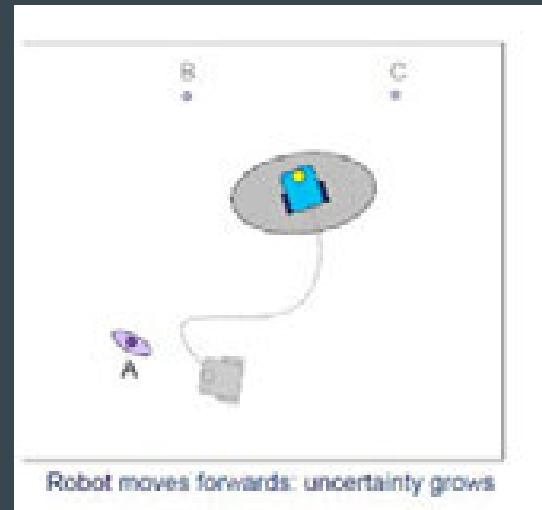
# SLAM with Gaussian filter

- Robot observes a feature (A)
- Mapped with uncertainty related to the measurement model
  - e.g. camera model describing how world points map into image pixels



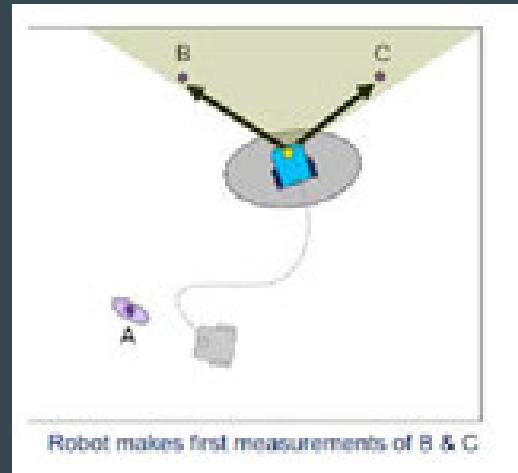
# SLAM with Gaussian filter

- As the robot moves, its pose uncertainty increases
  - Obeying the robot's motion model
- e.g. control commands: turn right, drive on for 1m
  - Uncertainty is added due to wheel slippage and other imprecisions



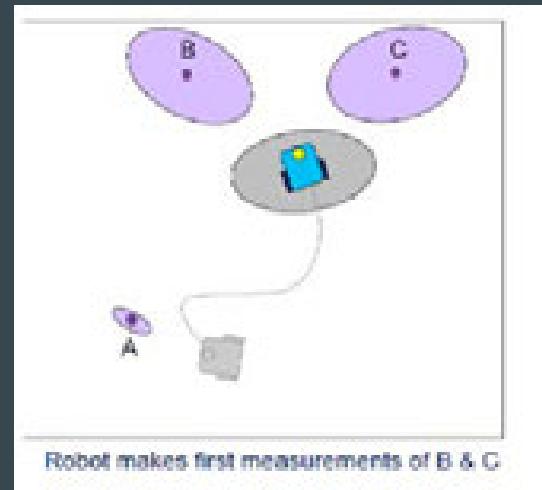
# SLAM with Gaussian filter

- Robot observes two new features
  - B and C



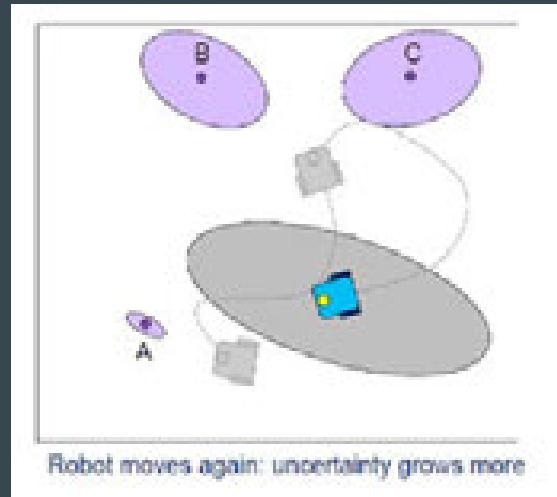
# SLAM with Gaussian filter

- Position uncertainty of B and C results from combination of:
  - Measurement error
  - Robot pose uncertainty
- Map becomes correlated with the robot pose estimate



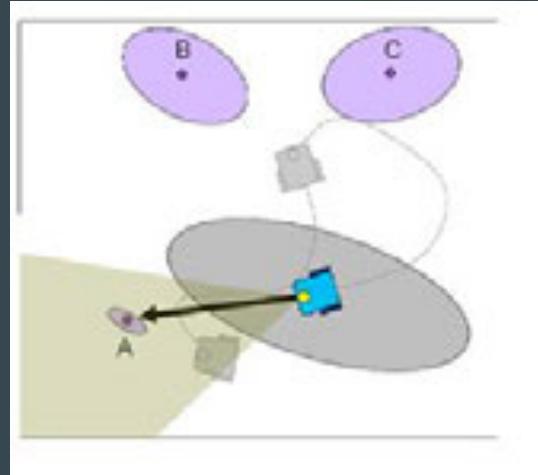
# SLAM with Gaussian filter

- Robot moves again
- Its uncertainty increases
  - Based on motion model



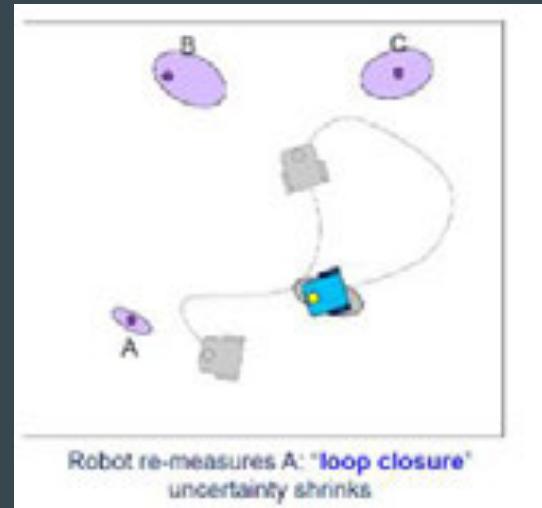
# SLAM with Gaussian filter

- Robot re-observes an old feature (A)
- Loop closure detection



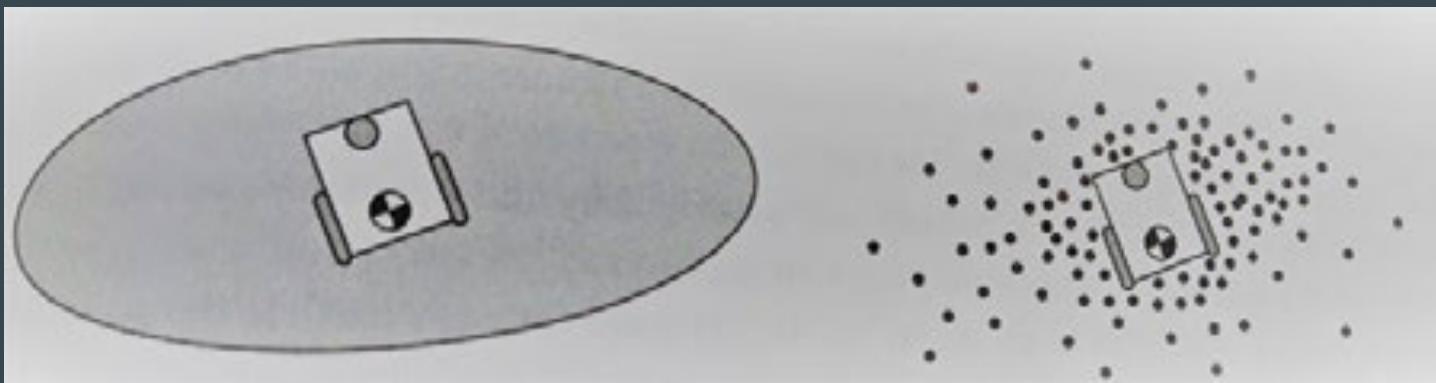
# SLAM with Gaussian filter

- Robot updates its position
  - Resulting pose estimate becomes correlated with the feature location estimates
- Robot's uncertainty shrinks
  - So does uncertainty in the rest of the map



# Extended Kalman Filter SLAM vs Particle Filter SLAM

- Standard EKF SLAM represents the probability distribution in parametric form with a Gaussian distribution
- Particle filter SLAM represents the probability distribution as a set of particles drawn randomly from the parametric distribution
  - Density of particles is higher toward the centre of the Gaussian



# SLAM with a particle filter

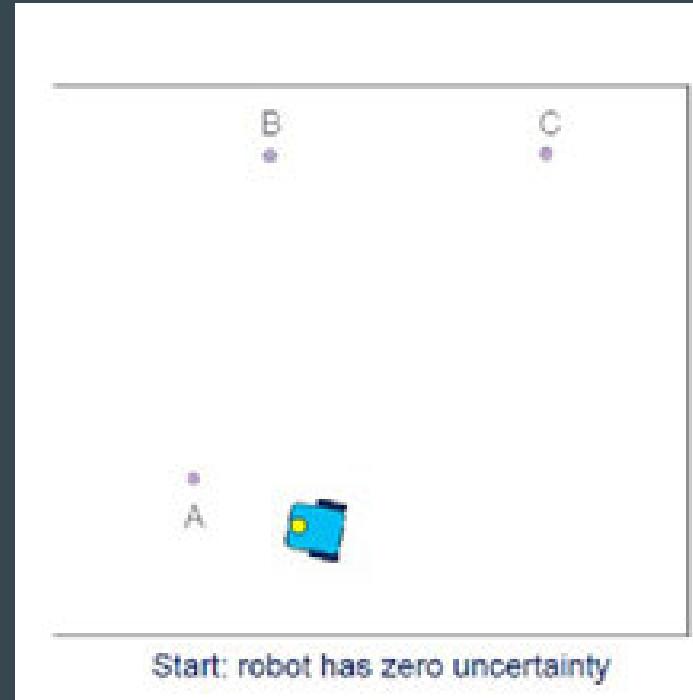
Use internal representations for:

- The positions of landmarks
- The camera parameters

Assumption:

- Robot's uncertainty at starting point is zero

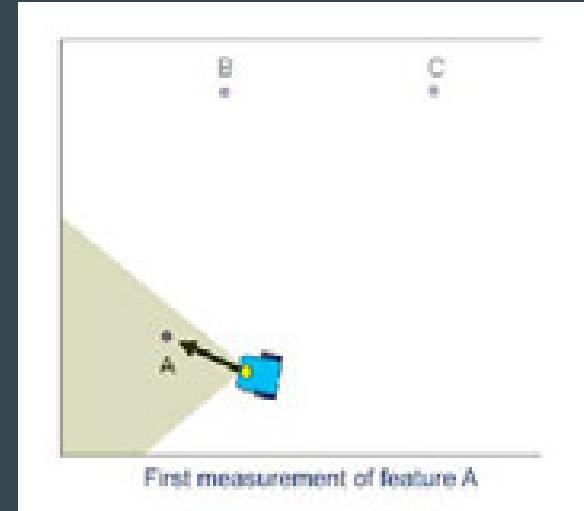
Initialise  $N$  particles at the origin, with weight  
 $1/N$



# SLAM with particle filter

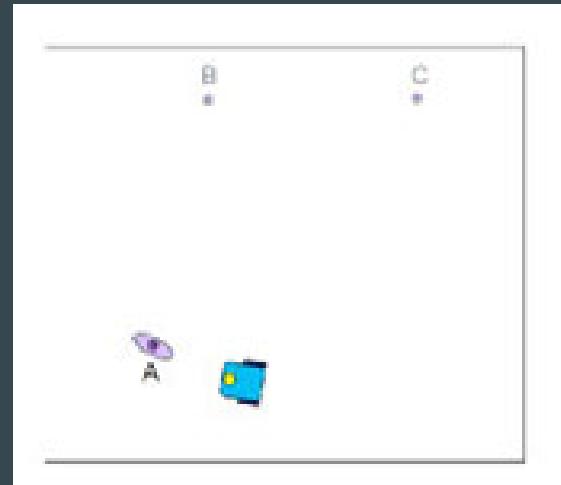
On every frame:

- **Predict** how the robot has moved
- **Measure** the world through sensors
- **Update** the internal representation



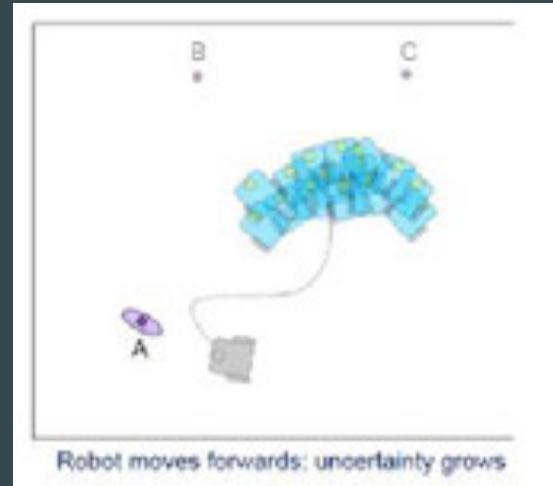
# SLAM with particle filter

- Robot observes a feature (A)
- Mapped with uncertainty related to the measurement model



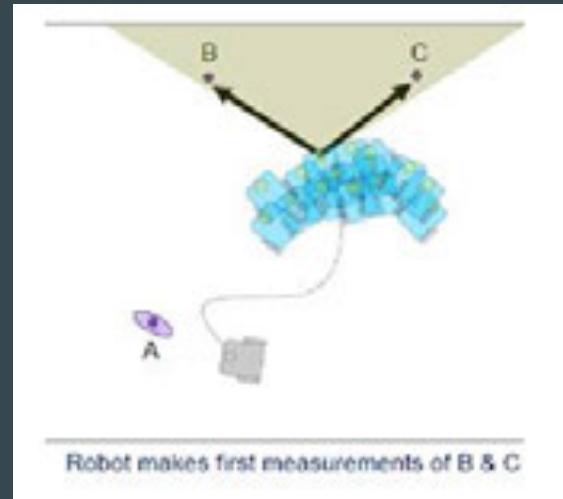
# SLAM with particle filter

- As the robot moves, pose uncertainty increases
- Apply motion model to each particle



# SLAM with particle filter

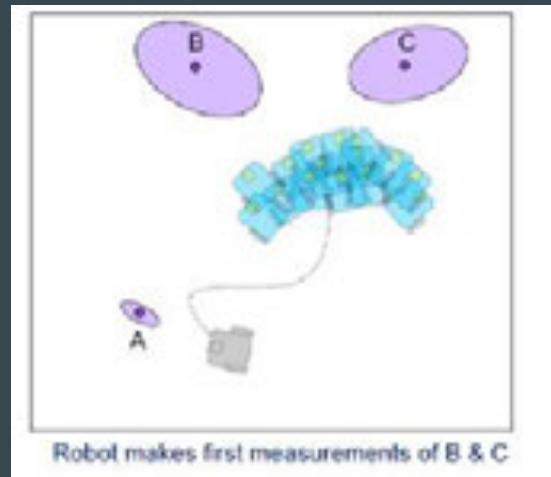
- Robot observes two new features
  - B and C



# SLAM with particle filter

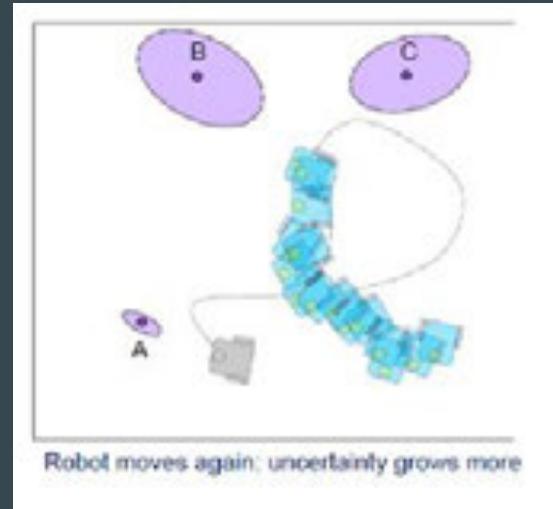
Position uncertainty encoded for each particle individually:

- Compare particle's predicted measurements with actual measurements
- Re-weight particles – those with good predictions get higher weight
- Renormalise particle weights
- Resample



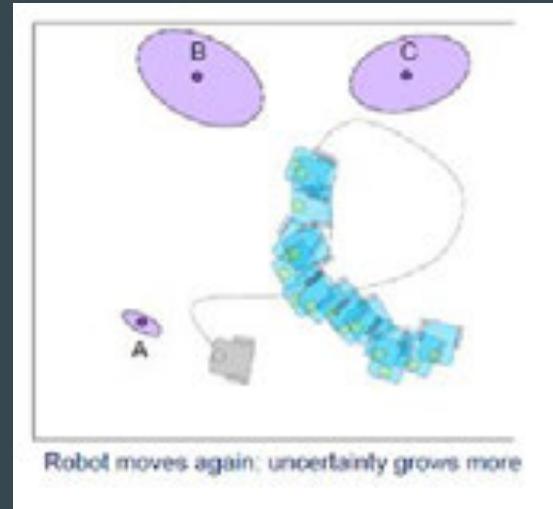
# SLAM with particle filter

- Robot moves again and its uncertainty increases
- Apply motion model to each particle



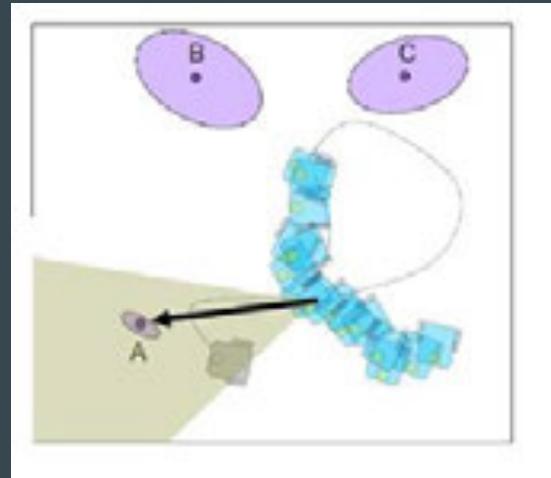
# SLAM with particle filter

- Robot moves again and its uncertainty increases
- Apply motion model to each particle



# SLAM with particle filter

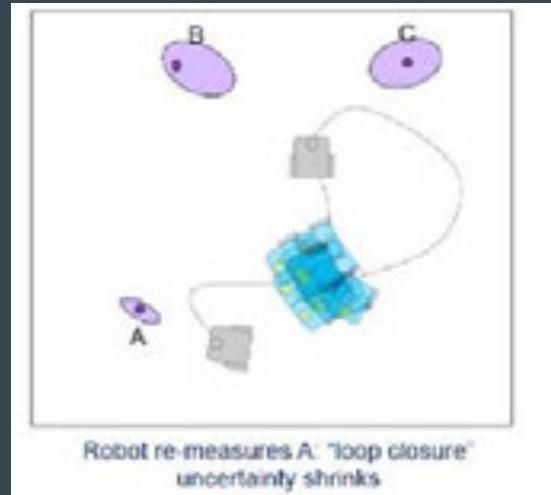
- Robot re-observes an old feature (A)
- Loop closure detection



# SLAM with particle filter

For each particle:

- Compare particle's predicted measurements with actual measurements
- Re-weight particles – those with good predictions get higher weight
- Renormalise particle weights
- Resample



# SLAM with particle filter

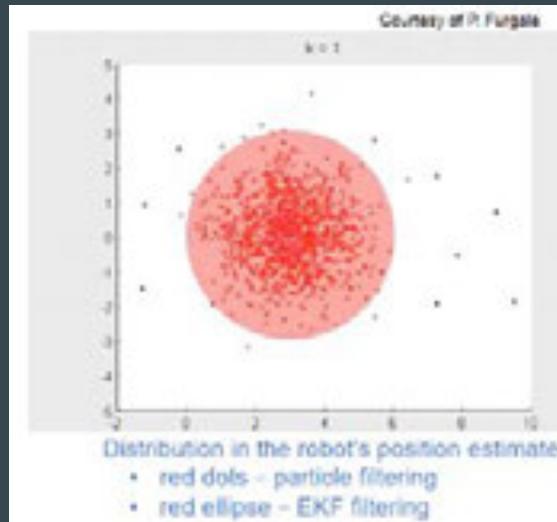
- Represents belief with a series of samples
- Each particle denotes a hypothesis of the state with an associated weight
- Predict/measure/update

## Pros:

- Noise densities from any distribution
- Works for multimodal distributions
- Easy to implement

## Cons:

- Does not scale to high dimensional problems
- Requires many particles to have good convergence



# Motion planning

- Planning Spaces
- Algorithms

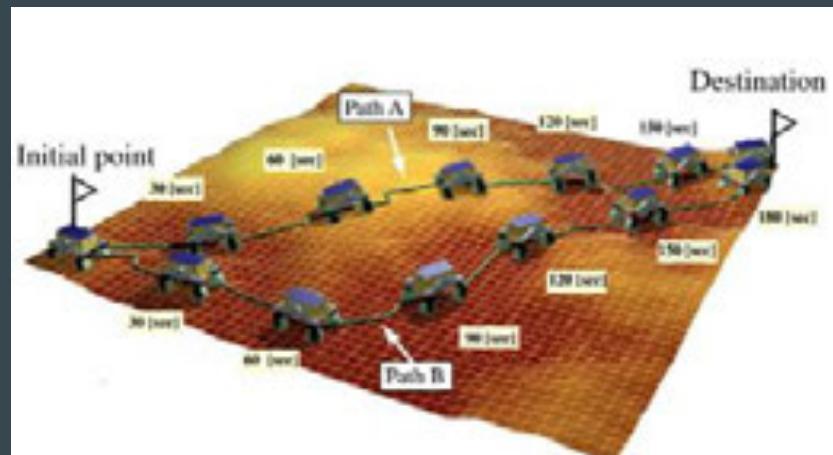
# Robot motion planning

- Representing planning problems
  - Planning spaces (Configuration space and Workspace)
- Graph search methods
  - Breadth-first search, depth-first search, Dijkstra's shortest path, A\*
- Potential fields

# How to get from point A to point B?

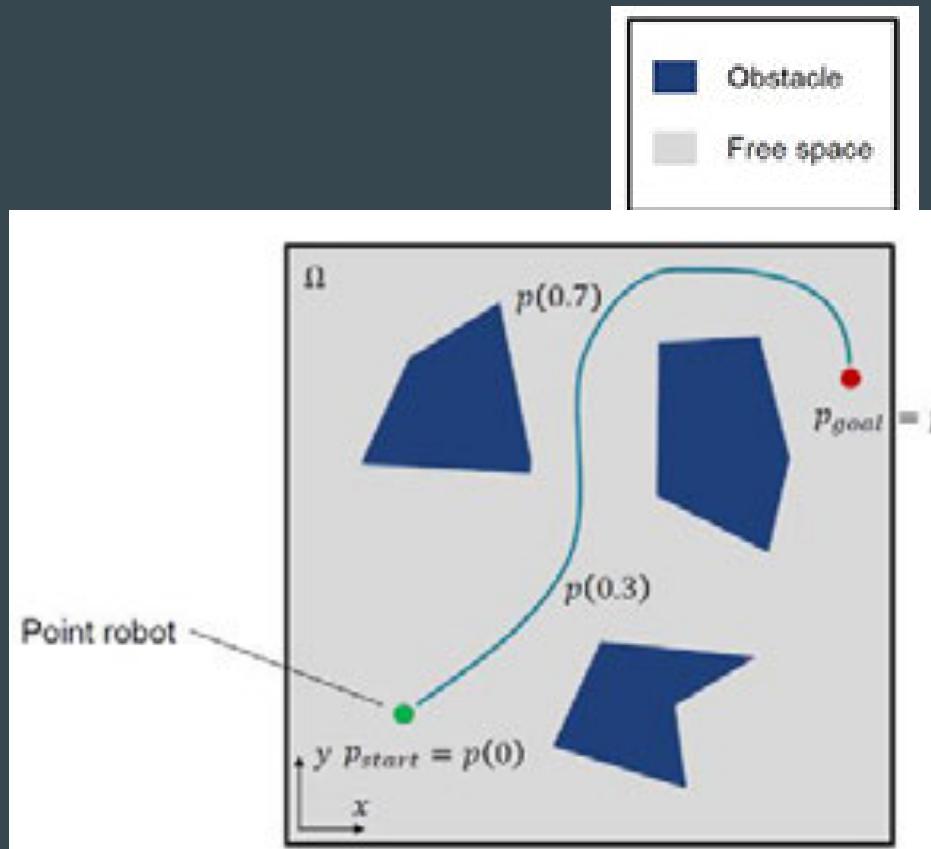
Assumptions:

- Representation of robot and world is sufficiently expressive
- Robot knows where it is and where it needs to go
- We have a motion model



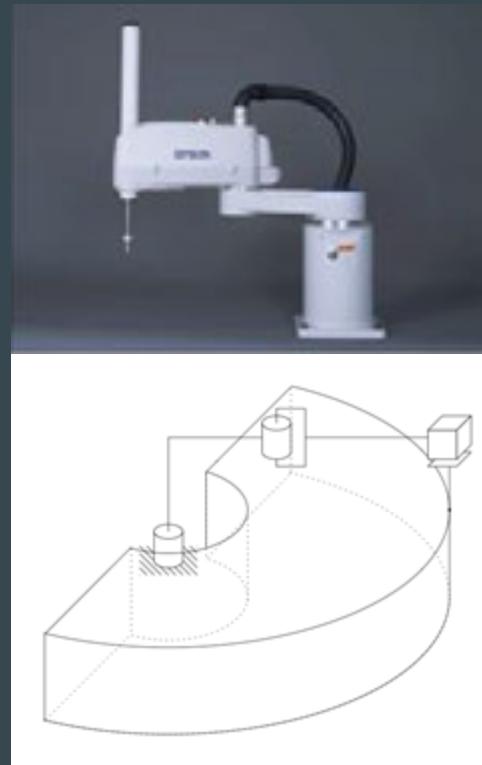
# Representing the world

- How the world is represented and understood by the planner (robot) is important
- Usually some degree of simplification in choosing a representation
- By choosing a suitable representation of the world, we may be able to apply existing algorithms to solve our planning problem



# Workspace and Configuration space

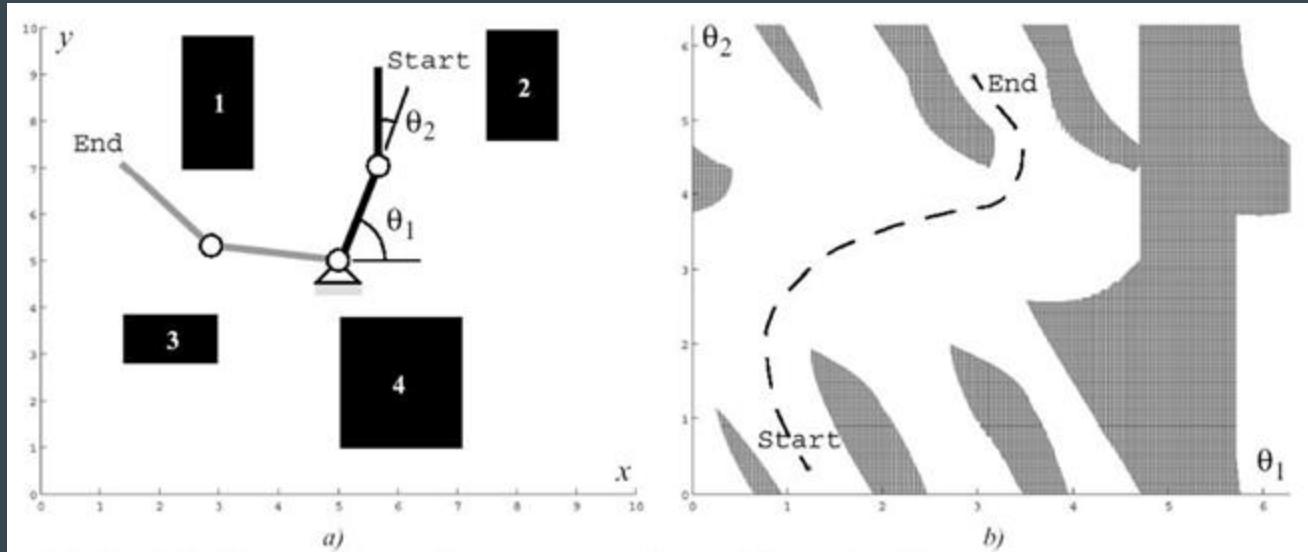
- **Workspace**
  - Reachable space within the environment
- **Configuration space:**
  - Full state of the robot in the environment



# Why use a Configuration space?

- Positions in configuration space tend be close together for the robot
- Can be easier to solve collision checks, and join nearby poses
- Allows a level of abstraction that means solution methods can solve a wider range of problems
- Sometimes helps with wraparound conditions (rotational joints)

# Configuration (C-)Space

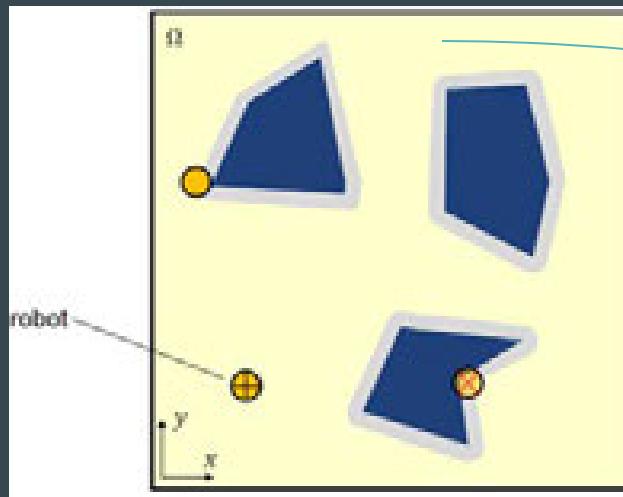


Path planning can be easier in configuration space

# Configuration (C-)Space

- For holonomic mobile robots, the configuration space is just the pose:  $(x, y, \theta)$
- We often assume the robot is holonomic
- Not a bad assumption for differential drive robots

# Configuration (C-)Space

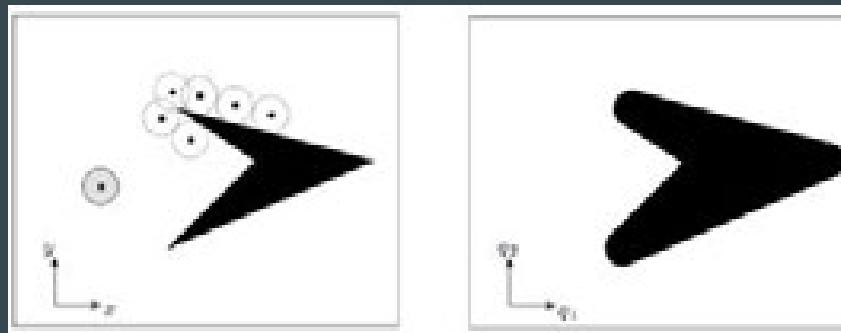


Configuration  
space (C-space)



# Configuration (C-)Space

- The robot is not a point, so expand obstacles by the diameter of the robot



# Continuous vs discrete state space representations

- Convert a planning problem to some kind of discrete representation
  - Then use the discrete decomposition for path planning
- Graph search: a connectivity graph is constructed (offline) and searched
- Potential field planning: a mathematical function is imposed on the free space. The gradient of the function is followed to reach goal.

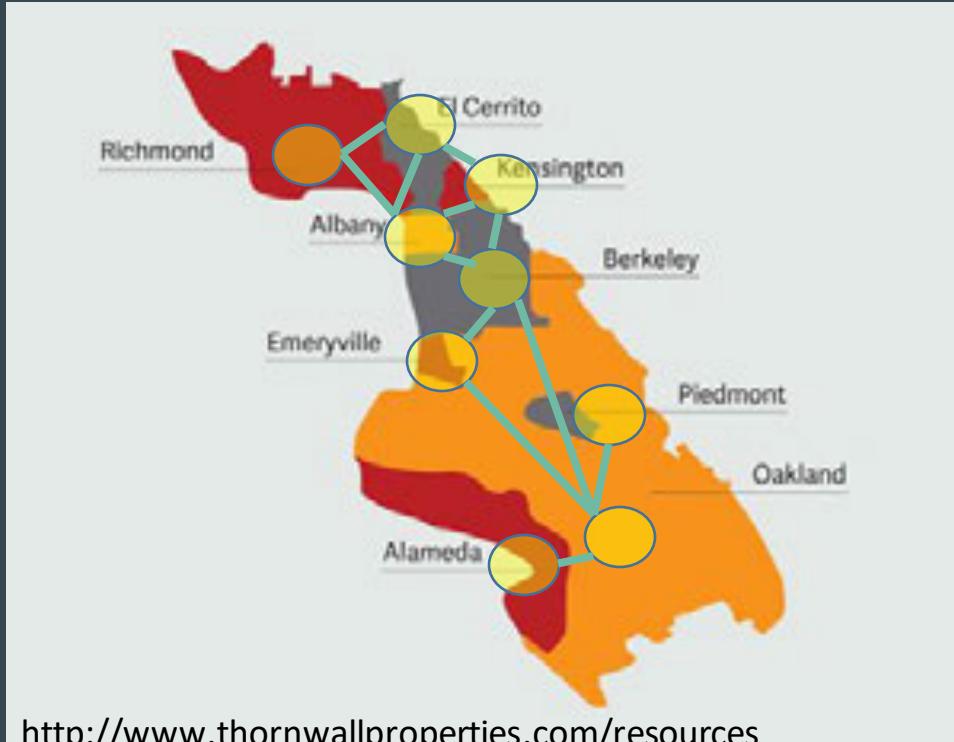
# Example Graph

## Neighborhood in the East Bay



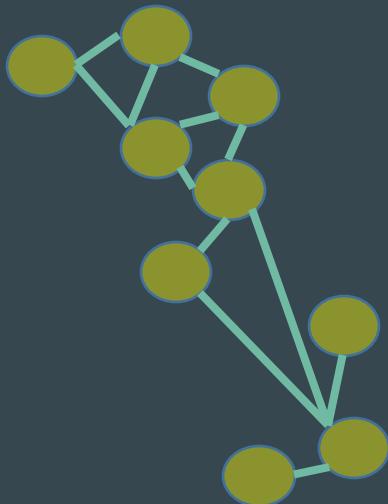
# Example Graph

## Neighborhood in the East Bay

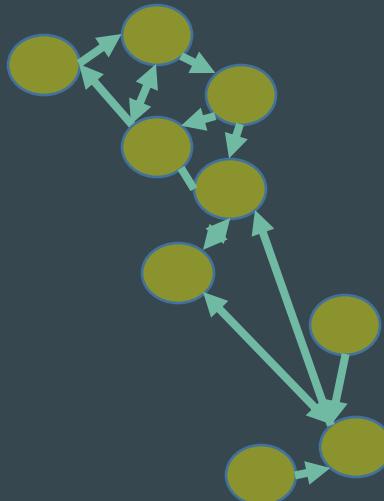


# Types of graphs

Undirected Graph



Directed Graph (Digraph)



# Types of graphs

Unweighted Graph

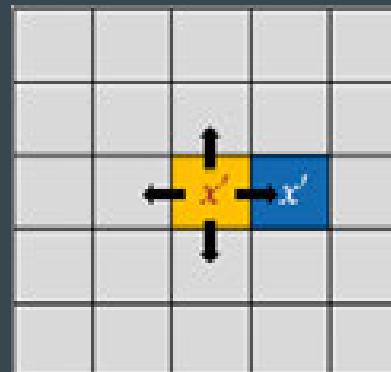


Weighted Graph



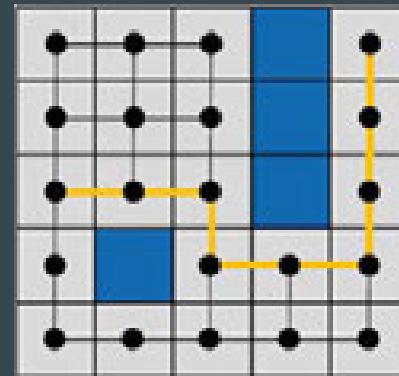
# Discrete state space representation

- Reduce continuous state space to a finite set of discrete states (discrete state space representation)
  - $x \in X$
- Define feasible actions from each state
  - $A(x) = \{a_0, a_1, \dots, a_n\}$
- And an associated transition function
  - $f(x, a) = x'$



# Grid to graph

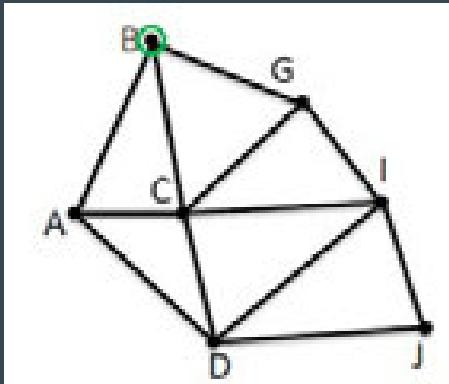
- Consider:
  - States as vertices
  - Transitions as directed edges
- Add:
  - Start node,  $x_s$
  - Goal node,  $x_g$
  - Cost function  $C: X \times A \rightarrow \mathbb{R}^+$
- Finding the shortest path can be treated as a graph search problem



# Forward search – node expansion

- Mark a node as “active”
- Explore its neighbours and mark them as “open”
  - Open set: list of frontier (unexpanded) plans
  - Keeps track of what nodes to expand next
  - For each node in the open list, we know of at least one path to it from the start
- Mark the parent node as “visited”
  - Closed set: nodes that have been expanded
  - For each node in the closed list, we’ve already found the lowest-cost path to it from the start

# Breadth-first search (BFS)



Open (Q):

{B}

Closed:

{}

- Our (BFS) queue will be FIFO:
- push ( $Q.Insert$ ) onto the end
  - pop ( $Q.GetFirst$ ) from the front

#### FORWARD SEARCH

```
1   $Q.Insert(x_0)$  and mark  $x_0$  as visited
2  while  $Q$  not empty do
3       $x \leftarrow Q.GetFirst()$ 
4      if  $x \in X_G$ 
5          return SUCCESS
6      forall  $u \in U(x)$ 
7           $x' \leftarrow f(x, u)$ 
8          if  $x'$  not visited
9              Mark  $x'$  as visited
10              $Q.Insert(x')$ 
11         else
12             Resolve duplicate  $x'$ 
13 return FAILURE
```

Figure 2.4: A general template for forward search.

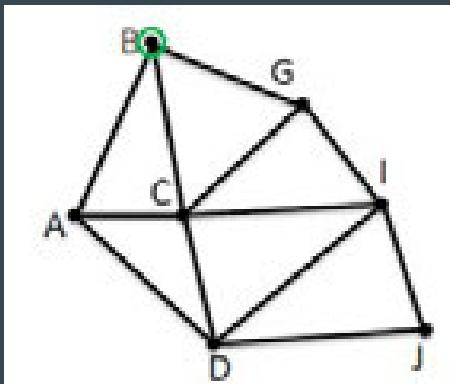
# Breadth-first search (BFS)

- Complete (will find the solution if it exists)
- Guaranteed to find the shortest path (number of edges, no weights)
- First solution that is found is the optimal path
- Time complexity:  $O(|V|+|E|)$  V: Vertices, E: Edges
- Names in robotics:
  - Wavefront
  - Forest fire

# Depth-first search (DFS)

- Starts at the root node and explores as far as possible along each branch
- Similar implementation to BFS, but with a stack (last-in first-out) queue

# Depth-first search (DFS)



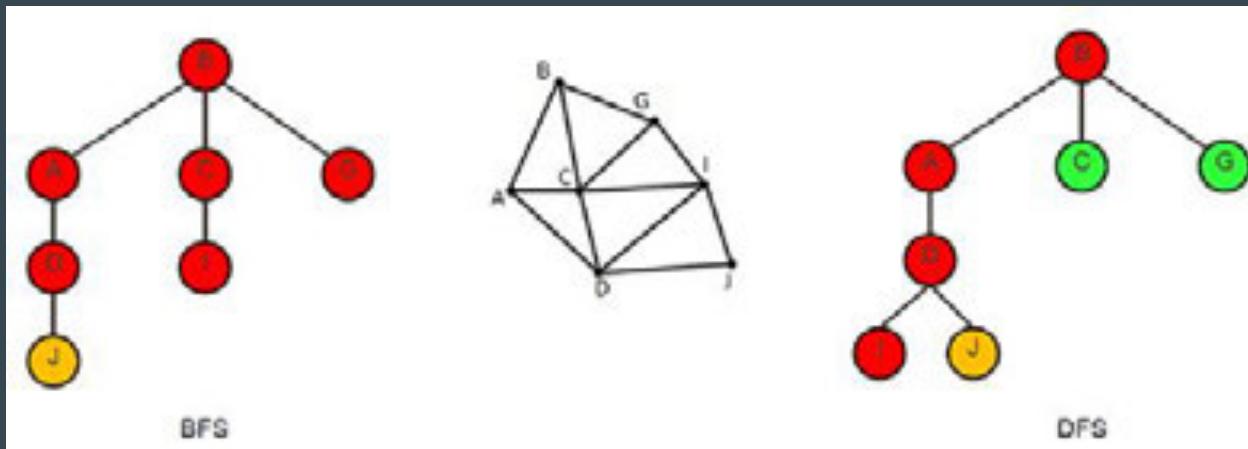
Open (Q):      Closed:  
{B}

- Our (DFS) queue will be LIFO:
- push ( $Q.Insert$ ) onto the front
  - pop ( $Q.GetFirst$ ) from the front

```
FORWARD-SEARCH
1    $Q.Insert(x_0)$  and mark  $x_0$  as visited
2   while  $Q$  not empty do
3        $x \leftarrow Q.GetFirst()$ 
4       if  $x \in X_G$ 
5           return SUCCESS
6       forall  $u \in U(x)$ 
7            $x' \leftarrow f(x, u)$ 
8           if  $x'$  not visited
9               Mark  $x'$  as visited
10               $Q.Insert(x')$ 
11           else
12               Resolve duplicate  $x'$ 
13   return FAILURE
```

Figure 2.4: A general template for forward search.

# BFS vs DFS



# BFS vs DFS

- DFS not complete for very deep trees
  - May explore an incorrect branch infinitely deep and never come back up
- BFS is complete
- DFS has lower memory footprint than BFS with high-branching
- Not often used for path search, but to completely explore a graph
- Both are simple to implement
- Both have time complexity  $O(|V|+|E|)$

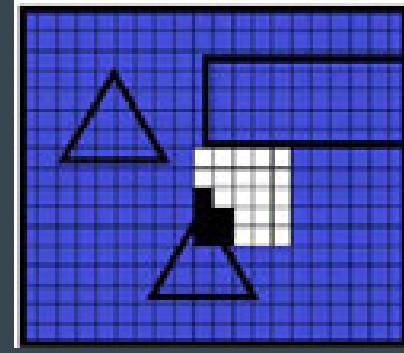
# Turning a polygonal C-space into a grid

A grid square is in the C-space if it is:

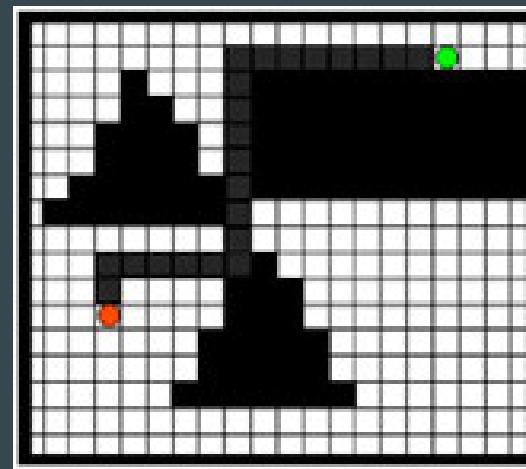
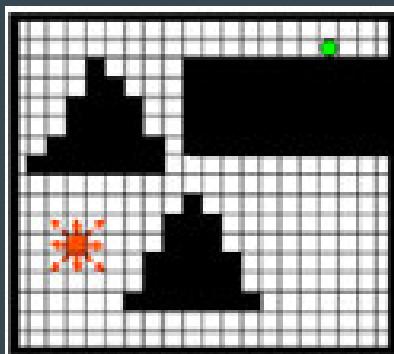
- Not inside an obstacle
- Further than the radius of the robot from all obstacle edges

Algorithm:

- Pick a grid square you know is in free space
- Do breadth-first search (“flood-fill”) from that start square
- As each square is visited by the search, compute the distance to all obstacle edges
- Label as “free” if the distance is greater than the radius of the robot, or “occupied” if the distance is less
- Once breadth-first search is done, also label all unlabelled squares as “occupied”

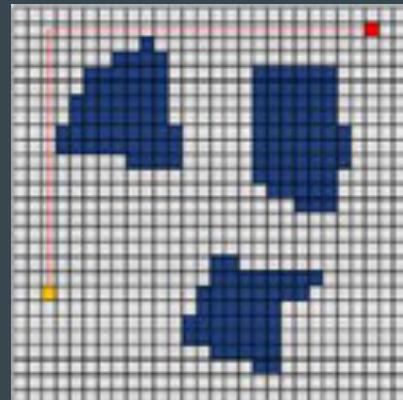


# Perform search



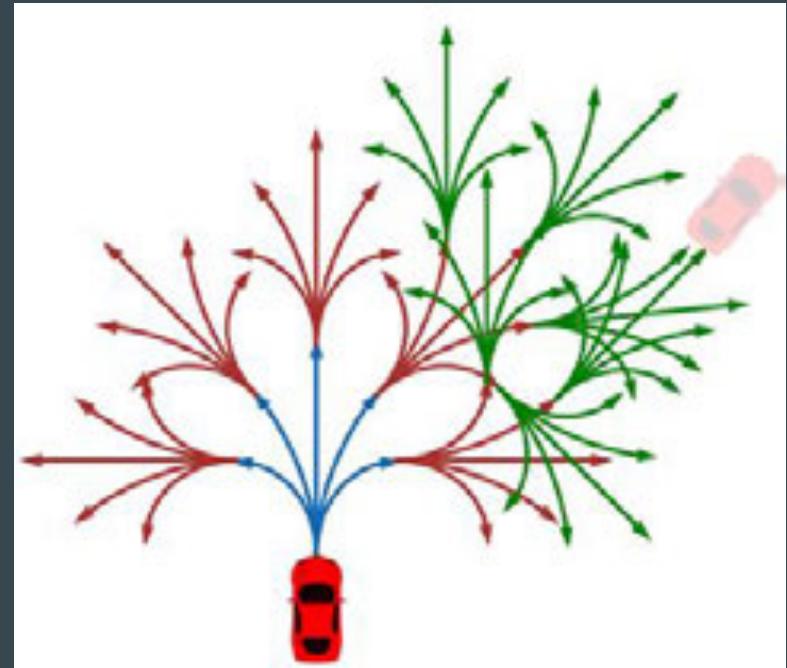
# Issues with grid-based representations

- Loss of precision
- Selecting grid resolution
- Type of output path
- Poor scaling in higher dimensions



# Grid lattice

- Create a set of feasible motion primitives
- Construct a tree (graph) that chains the motions into a sequence (plan)



# Graph construction: Visibility graph

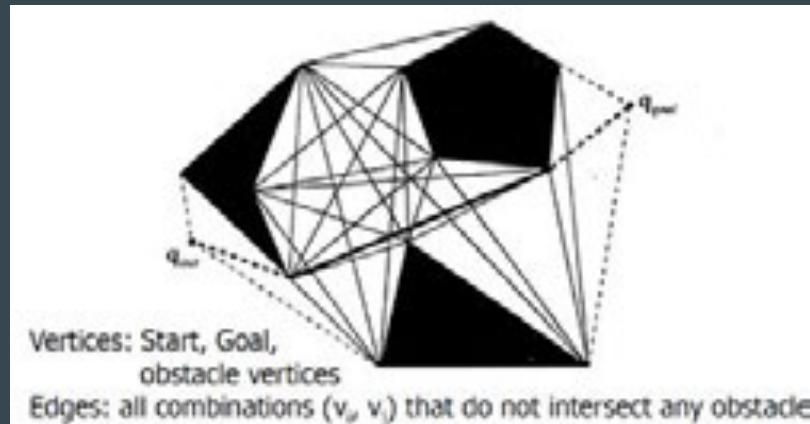
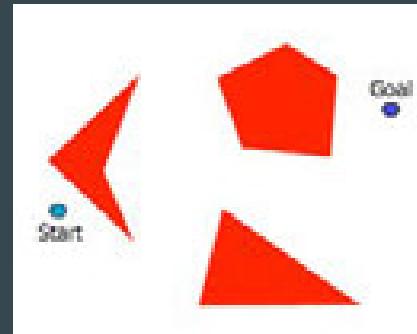
Create edges between all pairs of mutually visible vertices, and search resulting graph

## Pros:

- Optimal plan
- Good in sparse environments

## Cons:

- Limited to straight 2D motion
- Need polygonal obstacles
- Safety at stake



# Graph construction: Voronoi diagram

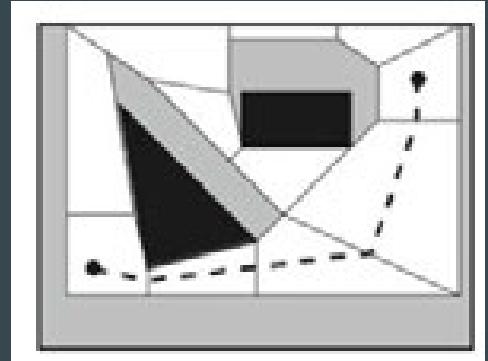
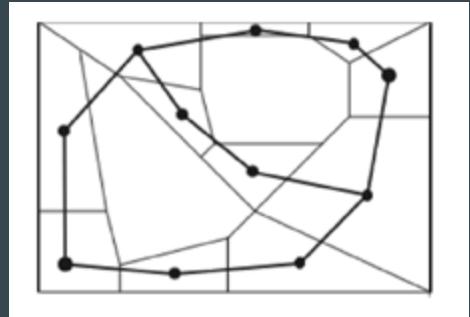
- Maximise the distance between the robot and the obstacles
- Draw equidistant lines
- Search resulting graph

Pros:

- Complete
- Executability

Cons:

- Not optimal
- Need long-range sensing



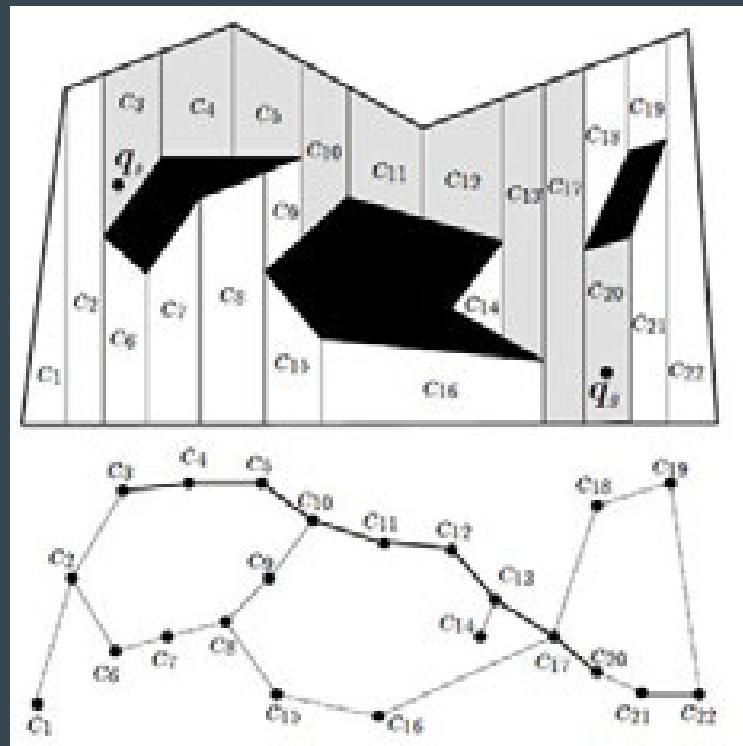
# Graph construction: Exact cell decomposition

## Pros:

- Complete

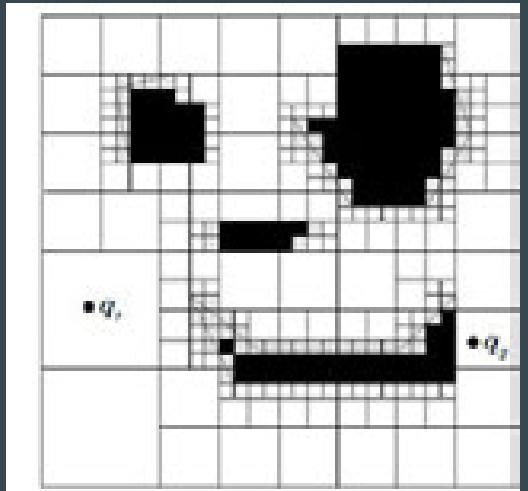
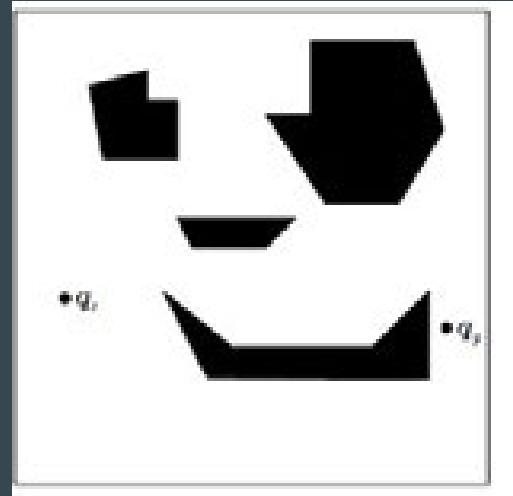
## Cons:

- Only good in extremely sparse environments



# Graph construction: Approximate cell decomposition

- Recursively decompose area into smaller rectangles
- Low computational complexity

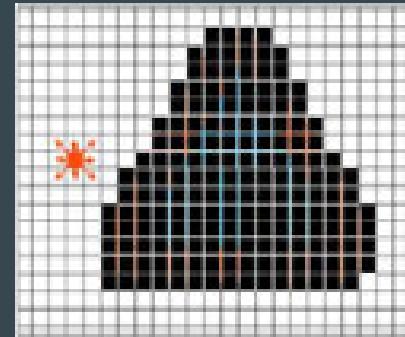


# Planning as search

- Given a representation, a start, a goal, and a motion model:
  - How do we actually generate a plan?
- We know how to search graphs
  - Computers are very good at this
  - Convert problem to a graph, and search it

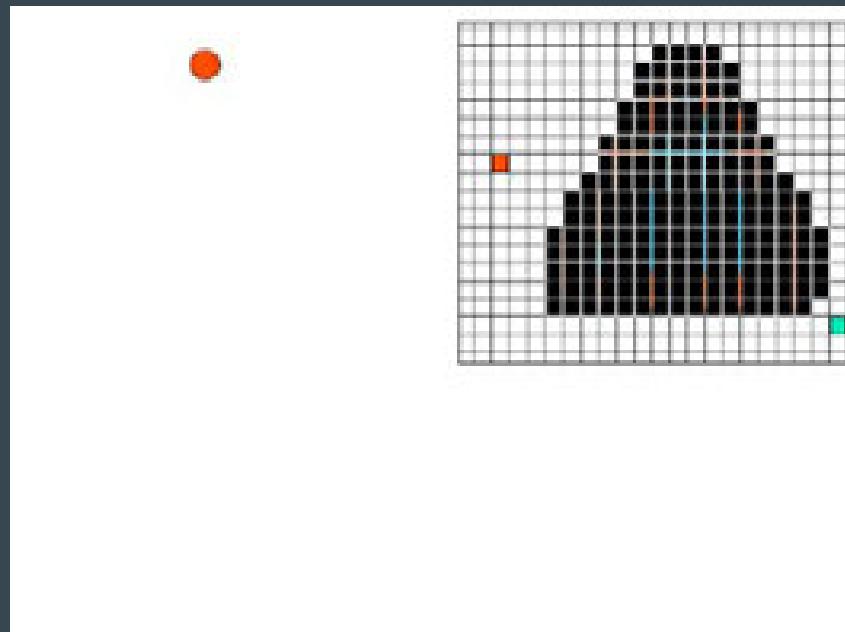
# Setting up the state space

- Real space
- Configuration space
- State space

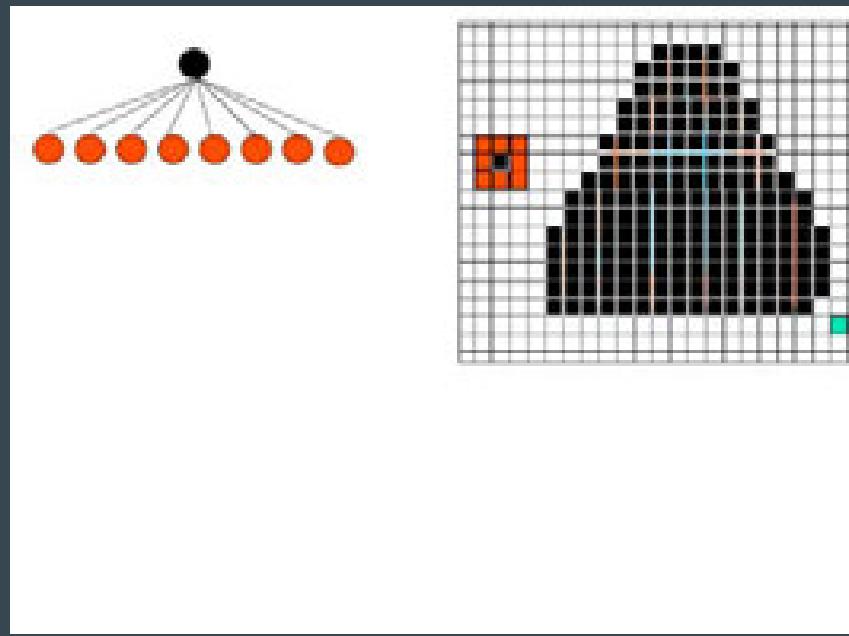


- Actions get you from one state to another
- Objective is to find a path from the start to the goal

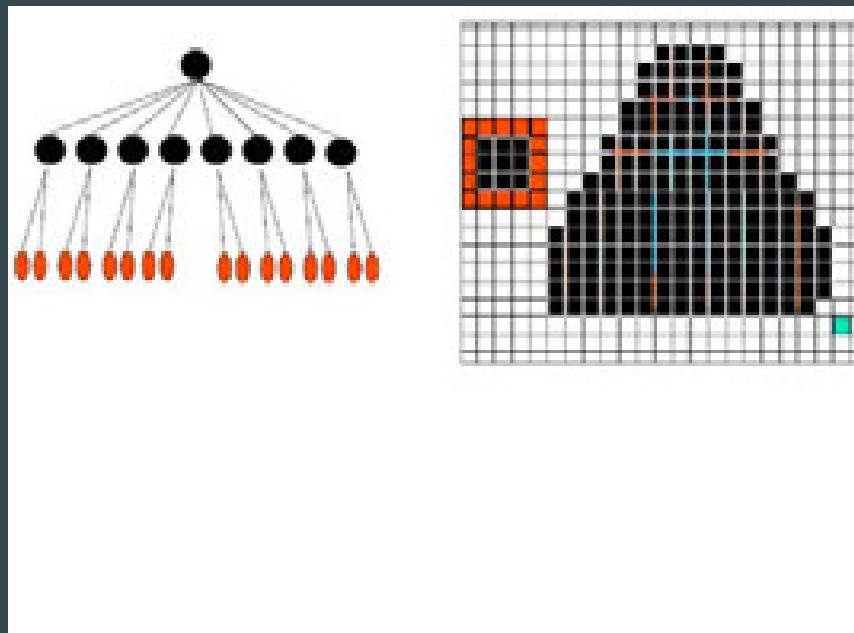
# Tree search



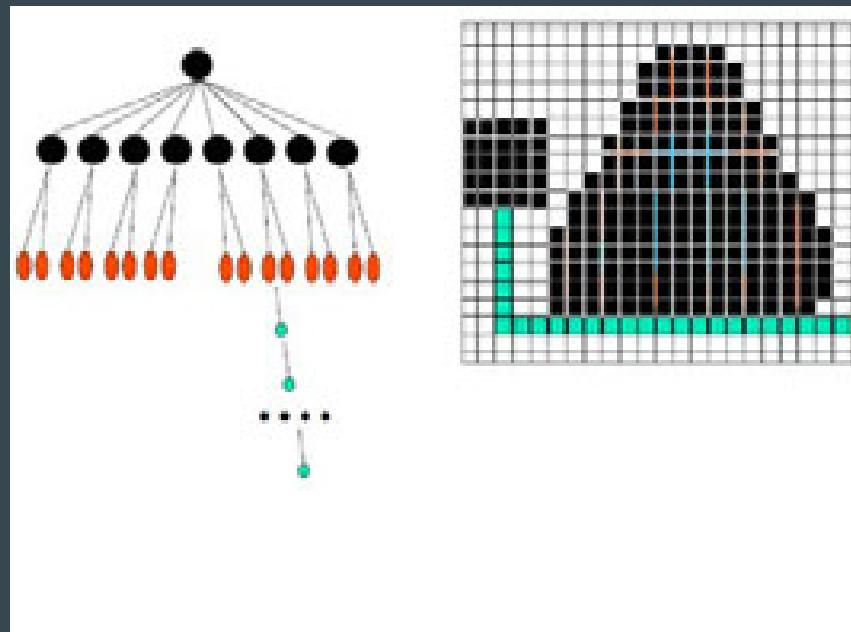
# Tree search



# Tree search

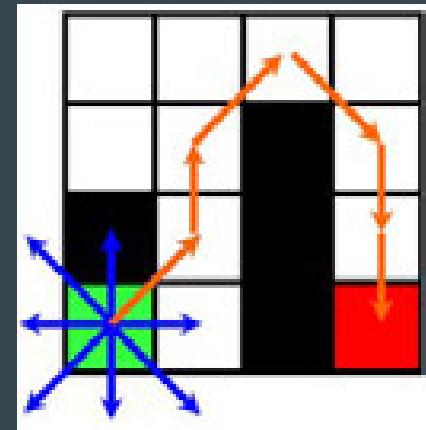


# Tree search



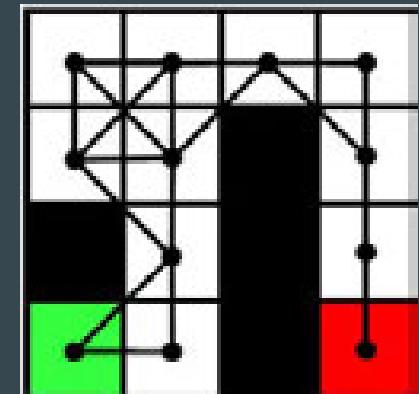
# Setting up the state space

- Configuration space
- State space
- Actions get you from one state to another
- Objective is to find a path from the start to the goal



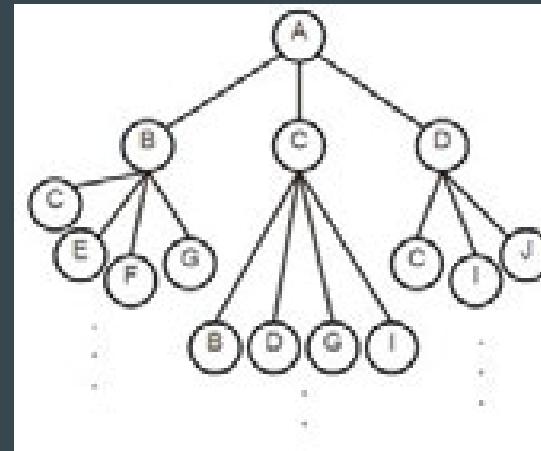
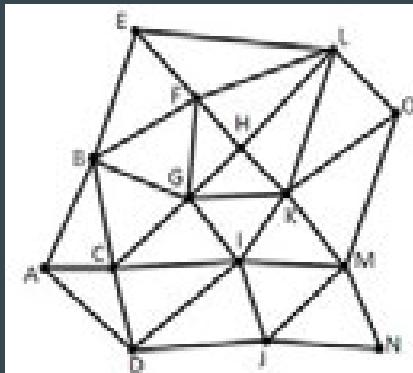
# Setting up the state space

- Search over the underlying graph
- Solve for paths from any point to any other point
- Assume all edge transitions are dynamically feasible



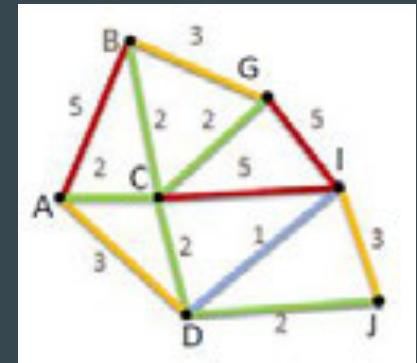
# Search trees

Construct a “tree” to search for optimal paths through the environment



# Dijkstra's shortest-path algorithm

- BFS with edge costs: Expanding in order of closest to start
- Asymptotically the fastest known shortest path algorithm for arbitrary directed graphs
- Open queue is ordered according to currently known best cost to arrive



# Dijkstra's shortest-path algorithm

Open (Q):

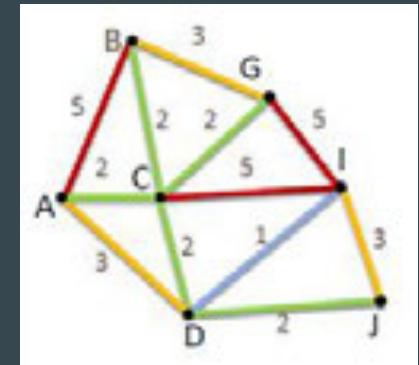
{B(0)}

Closed:

{B(0)}

Our Dijkstra queue will be ordered by cost to arrive:

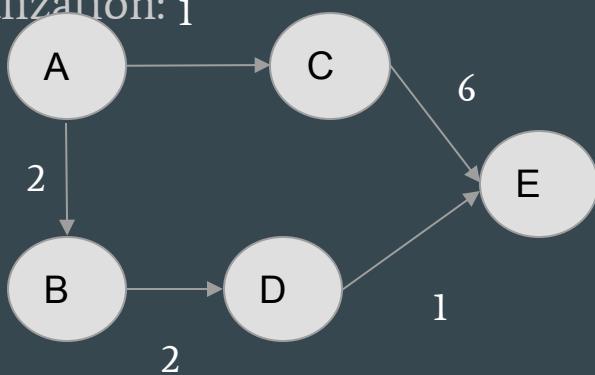
- push (*Q.Insert*) by cost
- pop (*Q.GetFirst*) from the front, and add it to the closed list



# Dijkstra's shortest-path algorithm

## Example

Initialization: 1

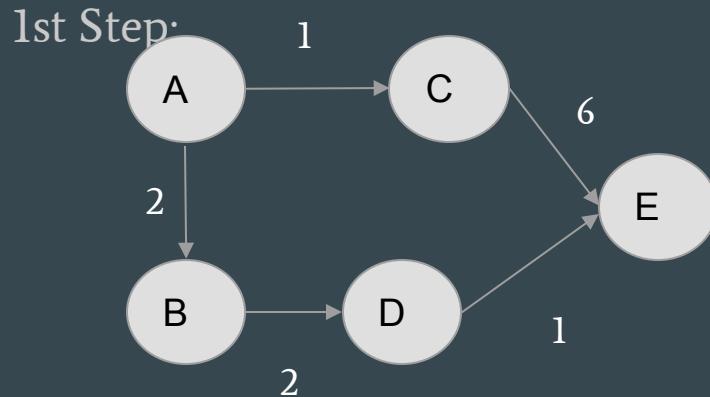


Unvisited: [A,B,C,D,E]

Node	Distance from A	Previous Node
A	0	
B	inf	
C	inf	
D	inf	
E	inf	

# Dijkstra's shortest-path algorithm

## Example

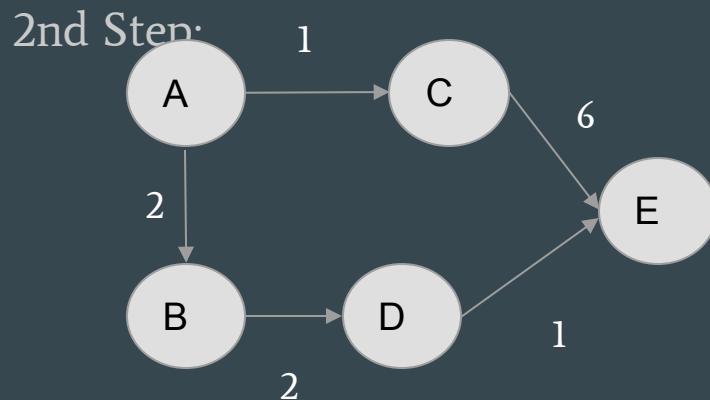


Unvisited: [A,B,C,D,E]

Node	Distance from A	Previous Node
A	0	
B	2	A
C	1	A
D	inf	
E	inf	

# Dijkstra's shortest-path algorithm

## Example



Unvisited: [A,B,C,D,E]

Node	Distance from A	Previous Node
A	0	
B	2	A
C	1	A
D	inf	
E	7	C

Repeat until all states are visited

# Dijkstra's shortest-path algorithm

- Can recover the lowest-cost route from the start to any node
  - Or any node with cost < goal if we terminate at a goal
- Easy to implement, with management with the priority queue
- Due to heap operations, time complexity becomes  $O(|V|\log|V|+|E|)$
- Doesn't really know the goal exists until it reaches it

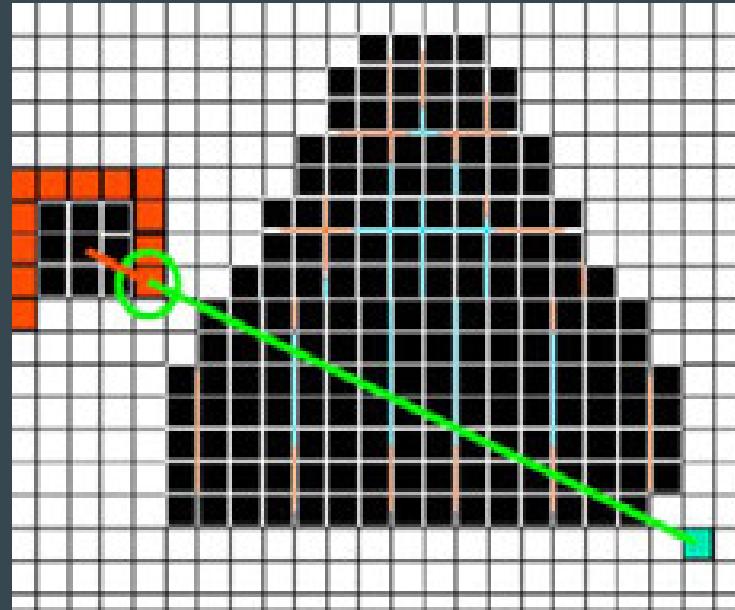
## Informed Search – A\*

Use domain knowledge to bias the search

Favour actions that might get closer to the goal

Each state gets a value  
 $f(x) = g(x) + h(x)$

Choose the state with best  $f$



## Informed Search – A\*

Use domain knowledge to bias the search

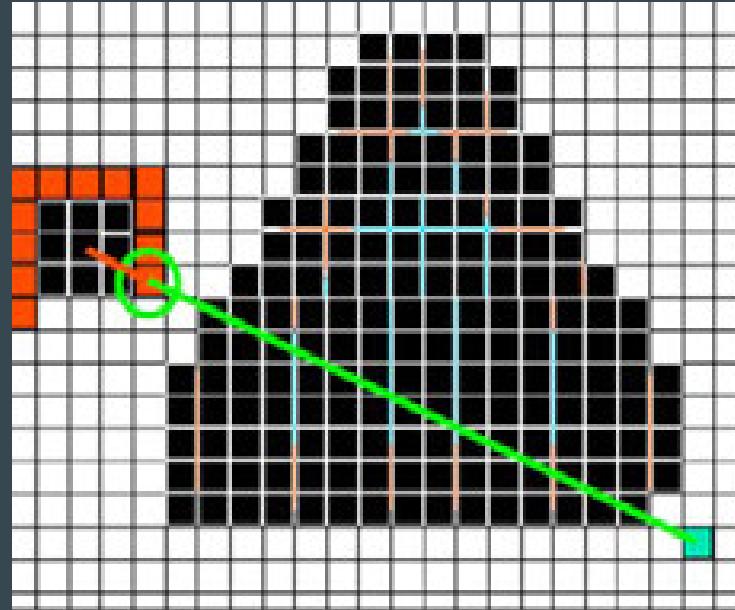
Favour actions that might get closer to the goal

Each state gets a value

$$f(x) = g(x) + h(x)$$

Cost incurred so far, from the start state

Estimated cost from here to the goal: “heuristic” cost



## Informed Search – A\*

Use domain knowledge to bias the search

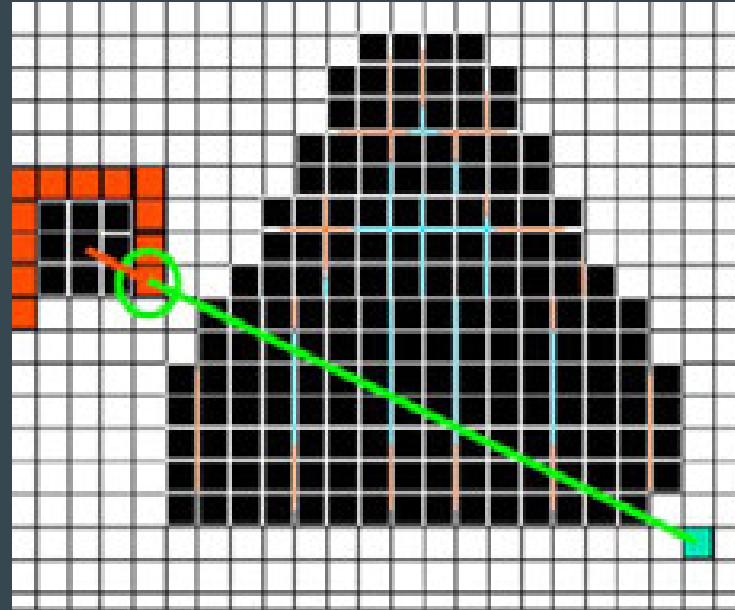
Favour actions that might get closer to the goal

Each state gets a value

$$f(x) = g(x) + h(x)$$

Cost incurred so far, from the start state

Estimated cost from here to the goal:  
“heuristic” cost



Example:

$$g(x) = 3$$

$$h(x) = ||x-g|| = \sqrt{8^2+11^2} = 19.7$$

$$f(x) = 22.7$$

## Informed Search – A\*

Use domain knowledge to bias the search

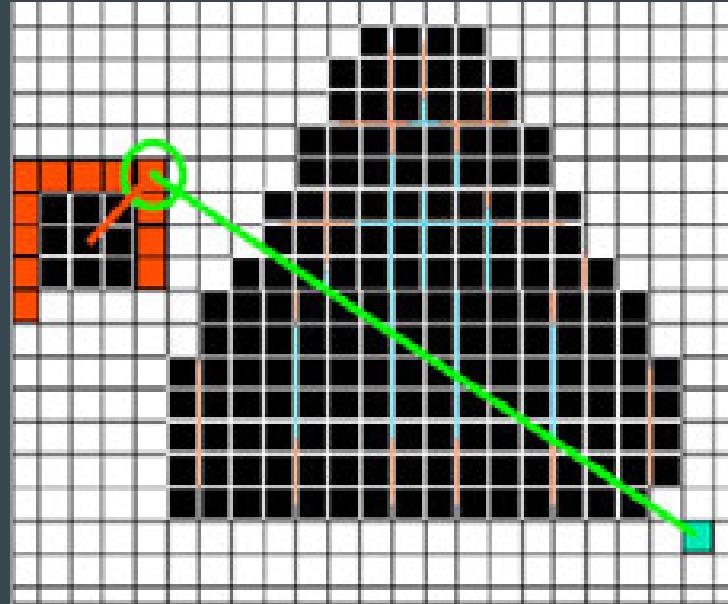
Favour actions that might get closer to the goal

Each state gets a value

$$f(x) = g(x) + h(x)$$

Cost incurred so far, from the start state

Estimated cost from here to the goal: “heuristic” cost



Example:

$$g(x) = 4$$

$$h(x) = ||x-g|| = \sqrt{11^2+18^2} = 21.1$$

$$f(x) = 25.1$$

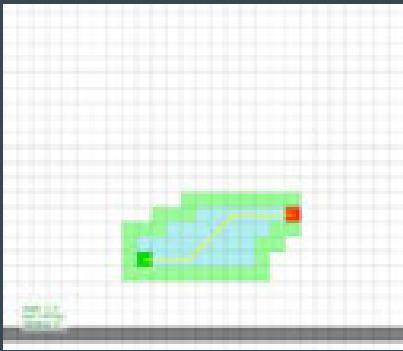
# How to choose heuristics?

- The closer  $h(x)$  is to the optimal cost to the goal,  $h^*(x)$ , the more efficient the search!
- The heuristic must be **admissible**
  - It never overestimates the cost
  - $h(x) \leq h^*(x)$  to guarantee that A\* finds the lowest-cost path
- The heuristic must be **consistent**
  - $h(x) \leq d(x,y) + h(y)$  for any pair of adjacent nodes x and y
- Euclidean Distance is a good choice of heuristic

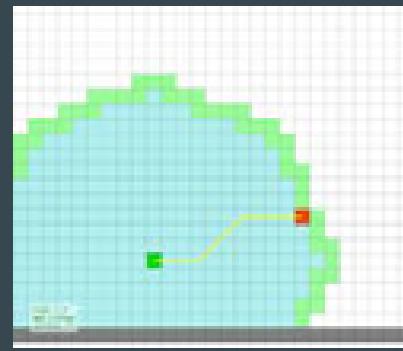
# Decisions, decisions...

- **How is your map described?**
  - Is it a grid map? Is it a list of polygons?
- **What kind of controller do you have?**
  - Do you just have controllers on distance and orientation?
  - Do you have behaviours, e.g. follow walls?
- **What do you care about?**
  - The shortest path? The fastest path?
- **What kind of search to use?**
  - Do you have a good heuristic? If so, then maybe A\* is a good idea.

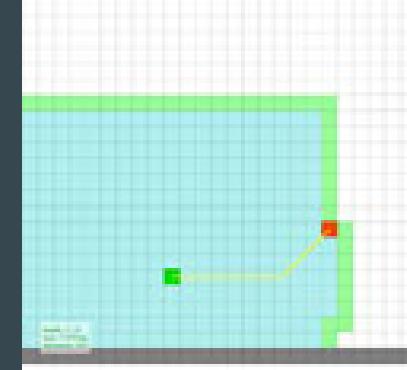
# Comparison



A\*



Dijkstra



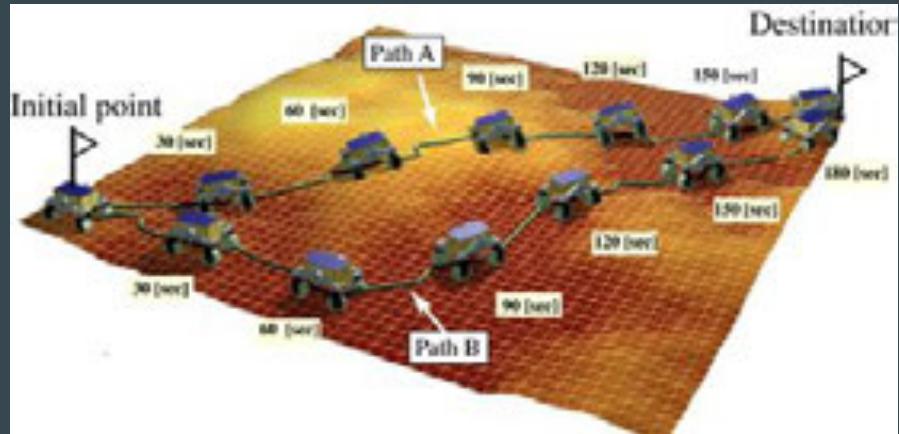
BFS

# Randomised graph search

- Complexity of breadth-first algorithm in a uniform grid as a function of the number of dimensions  $O(|V|+|E|)$

Number of nodes in a:

- 2D grid  $100 \times 100 = 10^4$
- 3D grid  $100 \times 100 \times 100 = 10^6$
- 6D grid 100 cells per d is  $10^{12}$



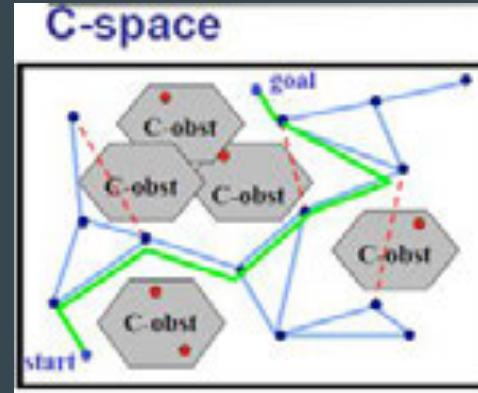
# Randomised graph search

- Divide the region uniformly into small cells
  - One approach is to randomly sample locations in the space and try to connect the samples
- A large proportion of the working volume is usually free space
  - If two points are ‘near’ each other, it is often the case that they can be connected by a simple path (e.g. straight line)

# Probabilistic road maps (PRM)

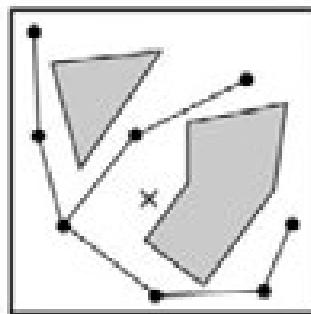
## (Kavraki et al. 1996)

- Roadmap construction (pre-processing)
  - Randomly generate robot configurations (nodes)
    - Discard nodes that are invalid
  - Connect pairs of nodes to form roadmap
    - Simple, deterministic local planner (e.g., straight line)
    - Discard paths that are invalid
- Query processing
  - Connect start and goal to roadmap
  - Find path in roadmap between start and goal
    - Regenerate plans for edges in roadmap
- Primitives Required:
  - Method for Sampling points in C-Space
  - Method for “validating” points in C-Space

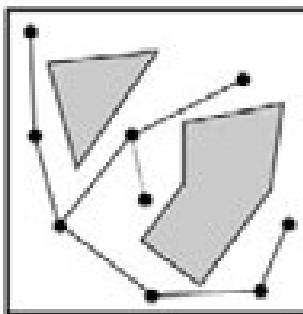


# PRM algorithm

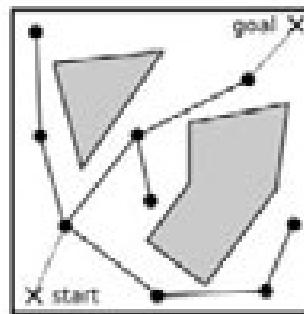
(1) PRM Algorithm



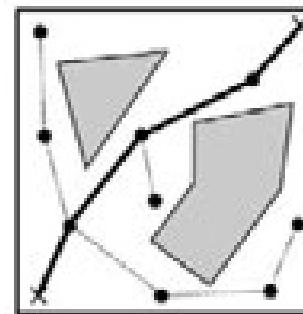
(a) The learning phase: a random sample, denoted by  $x$ , is generated.



(b) A local planner is used to connect the new sample to nearby roadmap vertices.



(c) The query phase: the start and goal configurations are added to the roadmap.

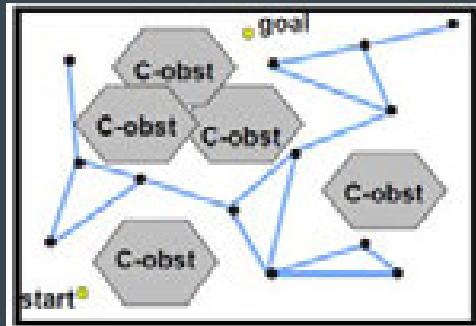


(d) A graph search algorithm is used to connect the start and goal through the roadmap.

# PRMs

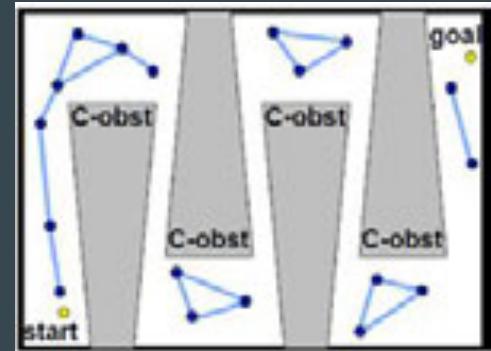
## Pros

- *Probabilistically complete*
- Applied easily to high-dimensional C-space
- Support fast queries with enough pre-processing



## Cons

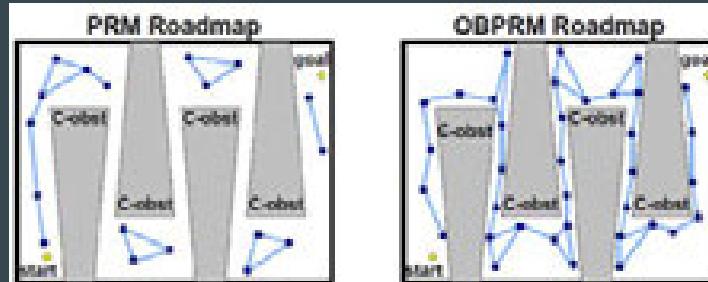
- Don't work as well for some problems:
  - Unlikely to sample nodes in *narrow passages*
  - Only *probabilistically complete*



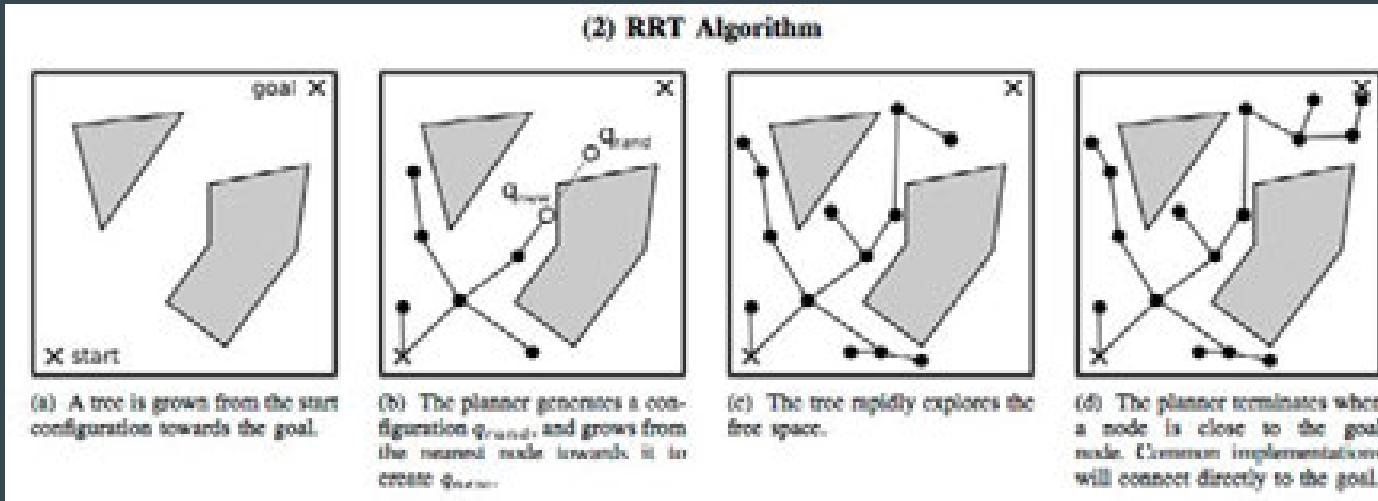
# Sampling around obstacles

(Amato et al. 1998)

- To navigate narrow passages we must sample in them
  - Most PRM nodes are where planning is easy (not needed)
- Can we sample nodes near C-obstacle surfaces?
  - We cannot explicitly construct the C-obstacles...
  - We do have models of the (workspace) obstacles...



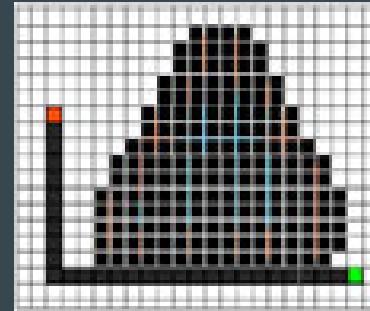
# RRT algorithm



# Divergence from a plan?

What happens if we take an action that causes us to leave the plan?

- Use behaviours
- Replan
- Keep a cached conditional plan
- Keep a policy



# Collision avoidance

- Try to move back onto the planned trajectory (global plan), while avoiding collisions (local planning)
- Potential field methods: create a field (or gradient) that pushes the robot away from obstacles, and towards the goal

# Potential fields

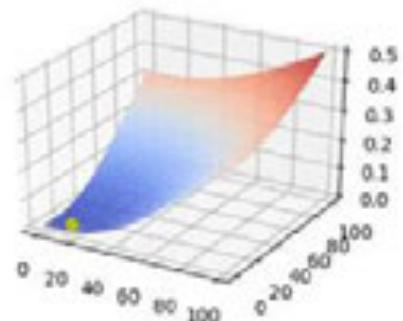
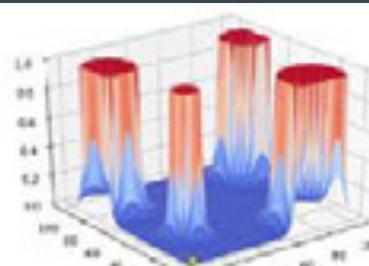
The potential of each obstacle generates a repulsive force

$$U_{rep} = \frac{1}{\|x - x_c\|}$$

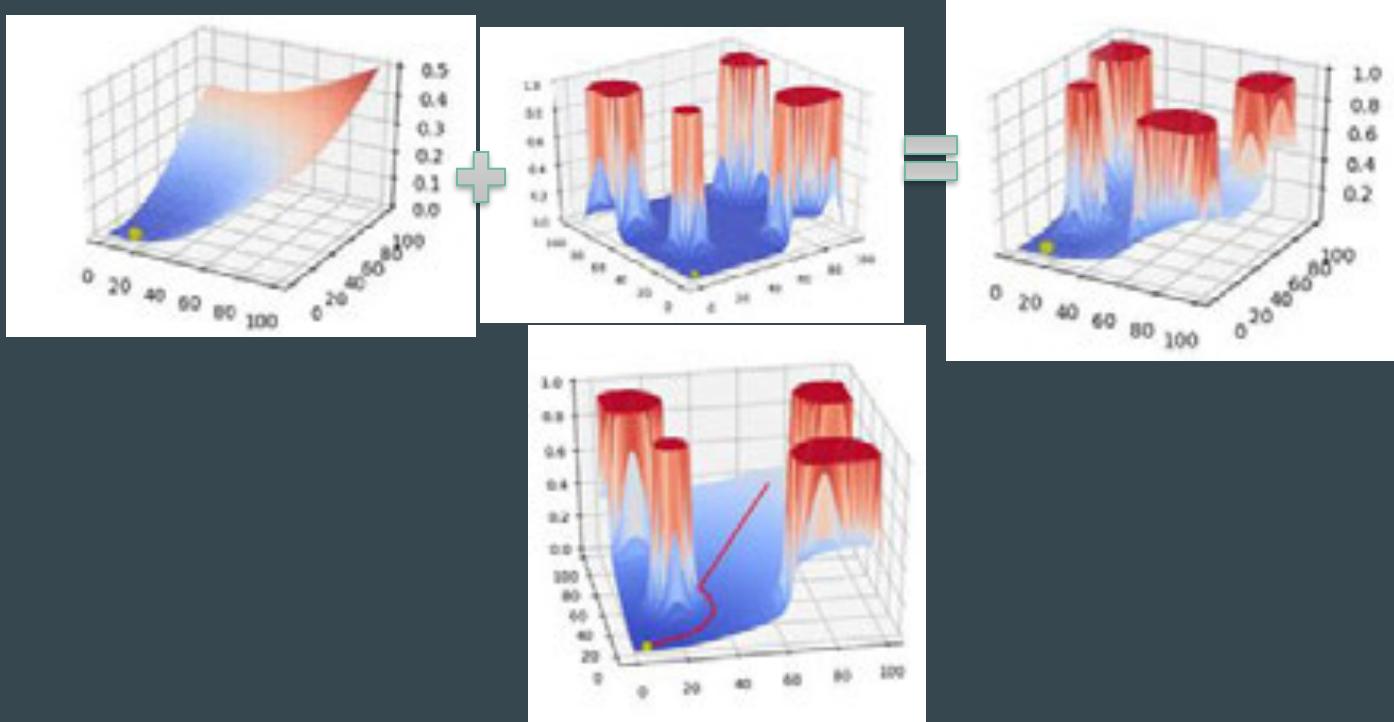
and the potential of the goal generates an attractive force

$$U_{att} = \frac{1}{2} \|x - x_{goal}\|^2$$

Easy and fast to compute  
Susceptible to local minima



# Potential fields



# Planning in practice

In general planning is done in a hierarchical manner:

- **Global planner**
  - Construct a path from initial position to the goal
  - A\*, RRT, etc
  - Path smoothing is usually performed to clean up solutions
- **Local planning**
  - Continuously run to adapt the planned global path to changes
  - Avoids the need to compute the entire global path
- **Reactive**
  - For collision avoidance in case of fast dynamic objects

CMP3101M AMR –

# Motion Planning & Control Architectures

Dr. Athanasios Polydoros

# Previous lecture

- SLAM
  - Particle Filter
  - Gaussian Filter
- Motion Planning
  - Representation as graph
  - Graph Search methods
    - Depth First Search
    - Breadth first Search
    - Dijkstra
    - A\*

# This lecture

- Some Graph Search Methods
  - Probabilistic Roadmaps
  - Rapidly exploring Random Tree (RRT)
  - Potential Fields
- Control Architectures
  - Deliberative – PDDL
  - Reactive
  - Hybrid

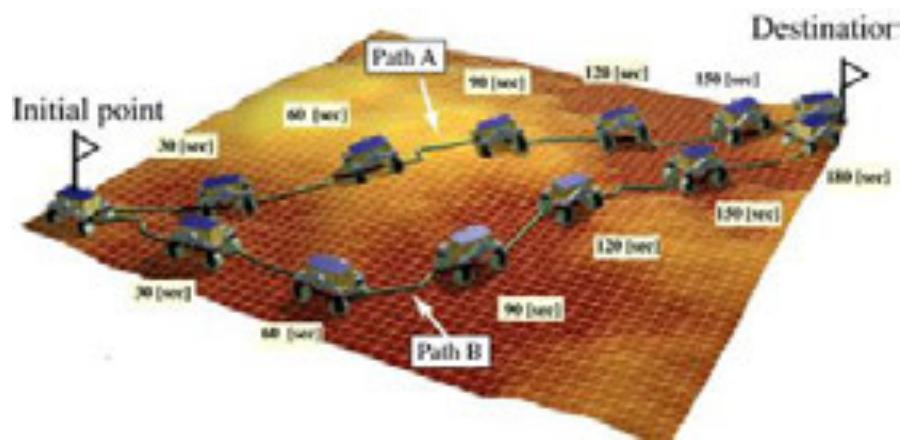
# Graph search methods

# Randomised graph search

- Complexity of breadth-first algorithm in a uniform grid as a function of the number of dimensions  $O(|V|+|E|)$

Number of nodes in a:

- 2D grid  $100 \times 100 = 10^4$
- 3D grid  $100 \times 100 \times 100 = 10^6$
- 6D grid 100 cells per d is  $10^{12}$

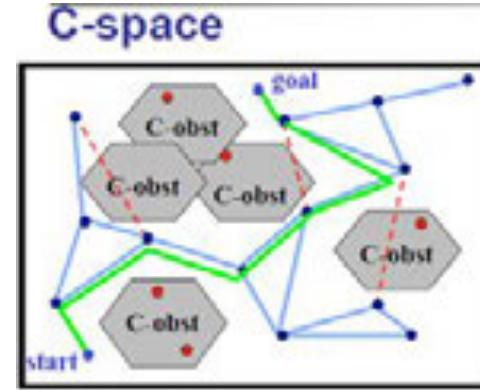


# Randomised graph search

- Divide the region uniformly into small cells
  - One approach is to randomly sample locations in the space and try to connect the samples
- A large proportion of the working volume is usually free space
  - If two points are ‘near’ each other, it is often the case that they can be connected by a simple path (e.g. straight line)

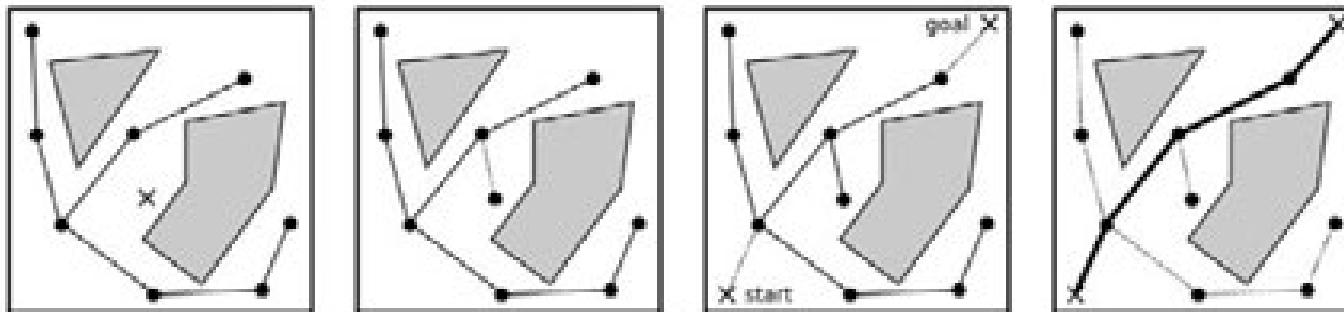
# Probabilistic road maps (PRM) (Kavraki et al. 1996)

- Roadmap construction (pre-processing)
  - Randomly generate robot configurations (nodes)
    - Discard nodes that are invalid
  - Connect pairs of nodes to form roadmap
    - Simple, deterministic local planner (e.g., straight line)
    - Discard paths that are invalid
- Query processing
  - Connect start and goal to roadmap
  - Find path in roadmap between start and goal
    - Regenerate plans for edges in roadmap
- Primitives Required:
  - Method for Sampling points in C-Space
  - Method for “validating” points in C-Space



# PRM algorithm

(1) PRM Algorithm



(a) The learning phase: a random sample, denoted by  $\times$ , is generated.

(b) A local planner is used to connect the new sample to nearby roadmap vertices.

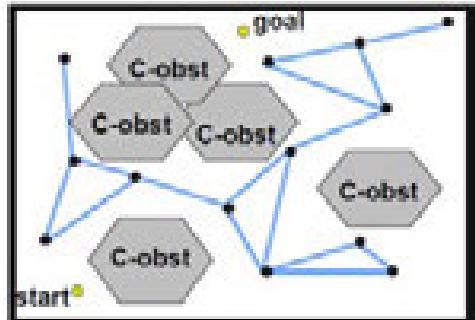
(c) The query phase: the start and goal configurations are added to the roadmap.

(d) A graph search algorithm is used to connect the start and goal through the roadmap.

# PRMs

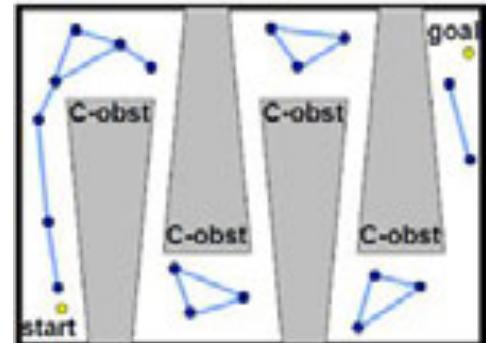
## Pros

- *Probabilistically complete*
- Applied easily to high-dimensional C-space
- Support fast queries with enough pre-processing



## Cons

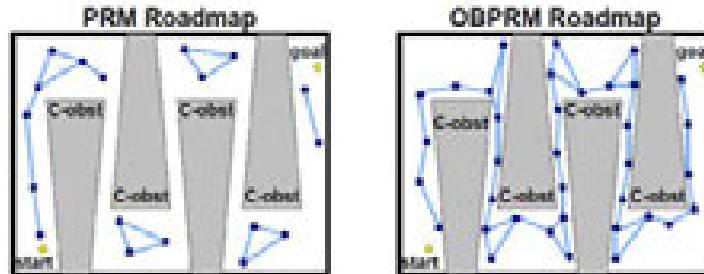
- Don't work as well for some problems:
  - Unlikely to sample nodes in *narrow passages*
  - Only *probabilistically complete*



# Sampling around obstacles

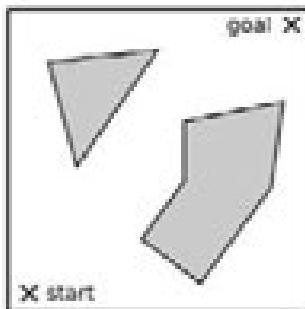
(Amato et al. 1998)

- To navigate narrow passages we must sample in them
  - Most PRM nodes are where planning is easy (not needed)
- Can we sample nodes near C-obstacle surfaces?
  - We cannot explicitly construct the C-obstacles...
  - We do have models of the (workspace) obstacles...

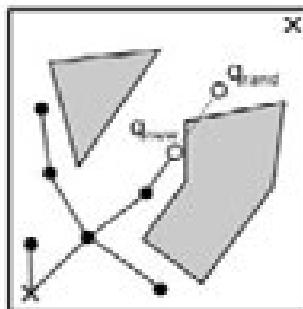


# RRT algorithm

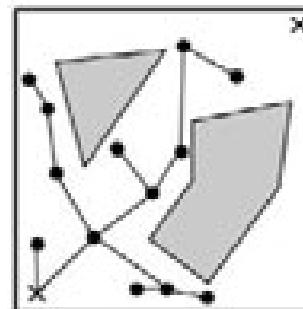
(2) RRT Algorithm



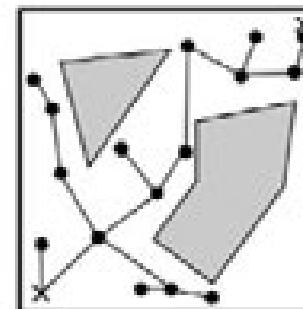
(a) A tree is grown from the start configuration towards the goal.



(b) The planner generates a configuration  $q_{rand}$ , and grows from the nearest node towards it to create  $q_{new}$ .



(c) The tree rapidly explores the free space.

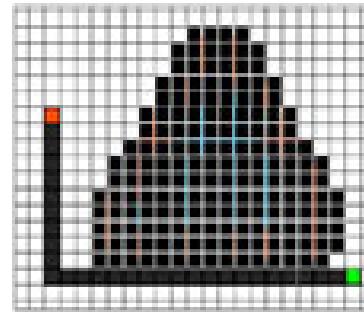


(d) The planner terminates when a node is close to the goal node. Common implementations will connect directly to the goal.

# Divergence from a plan?

What happens if we take an action that causes us to leave the plan?

- Use behaviours
- Replan
- Keep a cached conditional plan
- Keep a policy



# Collision avoidance

- Try to move back onto the planned trajectory (global plan), while avoiding collisions (local planning)
- Potential field methods: create a field (or gradient) that pushes the robot away from obstacles, and towards the goal

# Potential fields

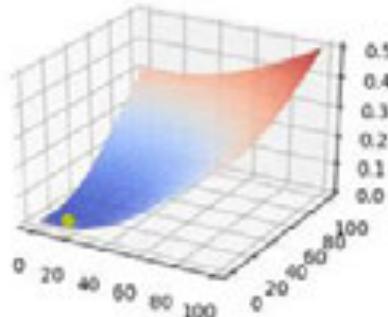
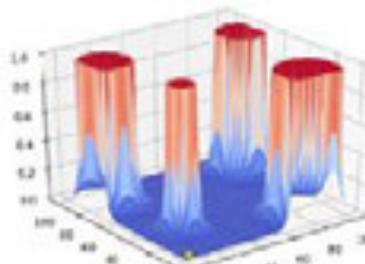
The potential of each obstacle generates a repulsive force

$$U_{rep} = \frac{1}{\|x - x_c\|}$$

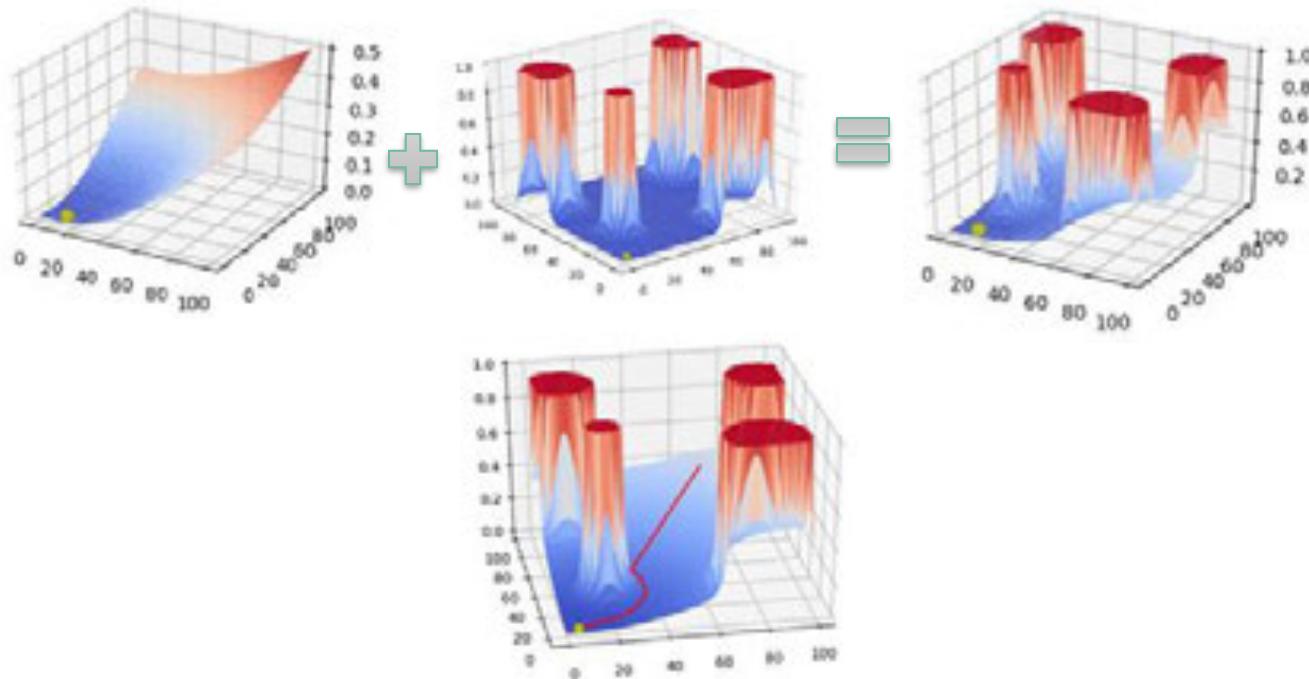
and the potential of the goal generates an attractive force

$$U_{attract} = \frac{1}{2} \|x - x_{goal}\|^2$$

Easy and fast to compute  
Susceptible to local minima



# Potential fields



# Planning in practice

In general planning is done in a hierarchical manner:

- **Global planner**
  - Construct a path from initial position to the goal
  - A\*, RRT, etc
  - Path smoothing is usually performed to clean up solutions
- **Local planning**
  - Continuously run to adapt the planned global path to changes
  - Avoids the need to compute the entire global path
- **Reactive**
  - For collision avoidance in case of fast dynamic objects

# Control Architectures

# Problems with robot control...

Complex Environment

Noisy / Unpredictable  
Environment

Actions don't always lead  
to intended consequences

Misleading sensors

Responses are too slow

# Why Control Architectures?

- Seen a range of competencies in previous weeks...
  - Sensing, motion, navigation, mapping, localisation, etc
- The issue remaining is how to bring it all together into a single system that can operate autonomously
  - ...and reliably, in a complex world
- Using a set of organising principles for the robot control system (primarily the software)
  - Building blocks
  - Requirements and Constraints

# Control Architecture Paradigms

- Robot control paradigm:

*“...a philosophy or set of assumptions and/or techniques which characterize an approach to a class of problems.”*

R. Murphy, 2000, p5

- Three main paradigms:

1. Deliberative
2. Reactive
3. Hybrid

- Each have advantages and disadvantages: in some cases, one approach may be more appropriate than another

- Typical means of characterising these paradigms is through the fundamental primitives: **sense, plan and act**

# Sensing, Planning, Acting

SENSE

Sense the Environment

- In: Raw sensor data
- Out: The sensed information (some processing)

PLAN

Decide what to do (using some model of the world)

- In: sensory information
- Out: directives

ACT

Act on the Environment

- In: Information (sensory or directives)
- Out: actuator commands

**DELIBERATIVE  
ARCHITECTURES**

**HYBRID  
ARCHITECTURES**

**REACTIVE  
ARCHITECTURES**

**COGNITIVE  
ARCHITECTURES**

## DELIBERATIVE ARCHITECTURES

Planning what action to take, assuming you have a world model

- Symbolic AI

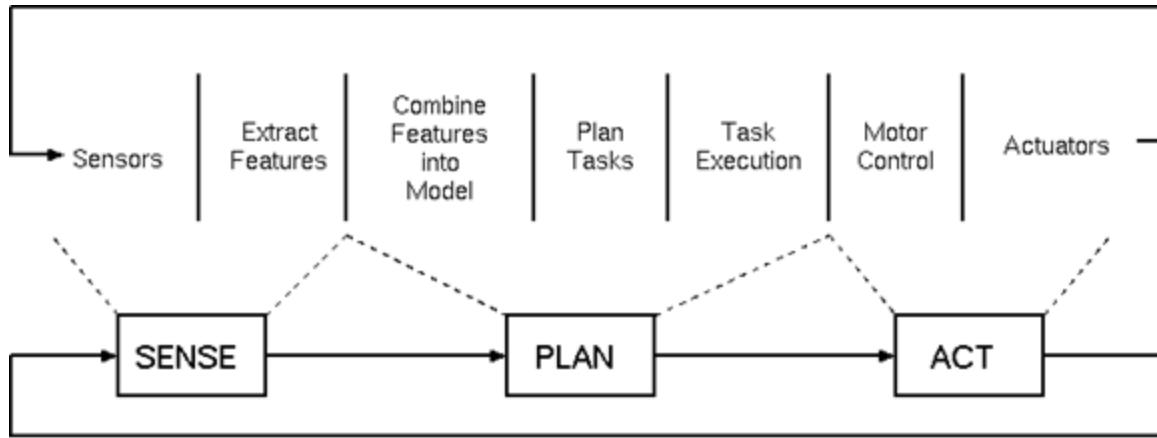
Emphasis on this ‘top-down’ (hierarchical) planning process

- Partly inspired by human introspection

Basic process:

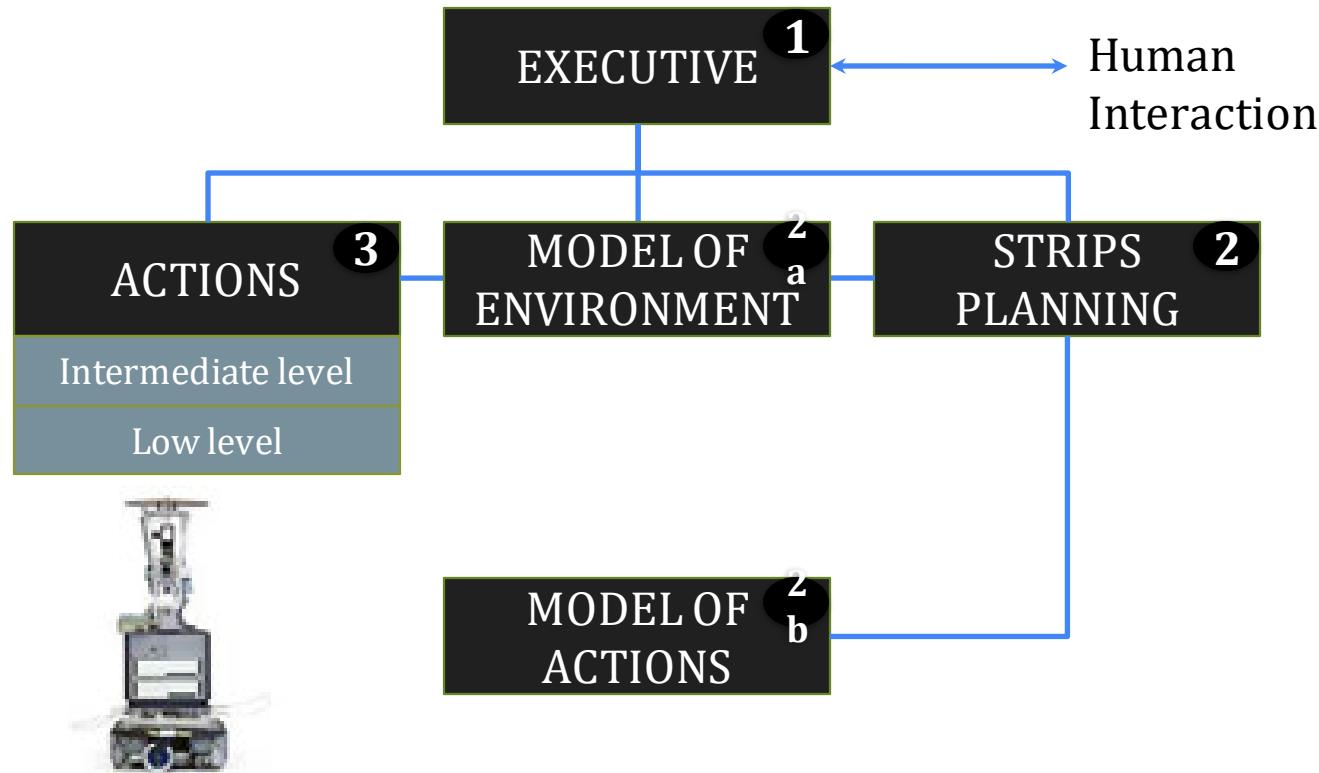
1. Gather currently available information, integrate into world model
2. Plan what to do
3. Execute the plan; return to step 1

# Horizontal decomposition



- Horizontal decomposition of tasks
  - A pipeline model: planning follows sensing, acting follows planning
- Plan first, then act out plan (open-loop)

# Control Architecture



# Planning: STRIPS

- Stanford Research Institute Problem Solver
- Planning to accomplish a goal
  - Break down into sub-goals to reduce difference between current state and goal state
- Symbolic representation of all information
  - The world model: everything about the state of the environment
  - The capabilities/properties of the robot itself (operators)
  - Initial and goal states
  - Difference evaluator (how close to the goal state am I?)

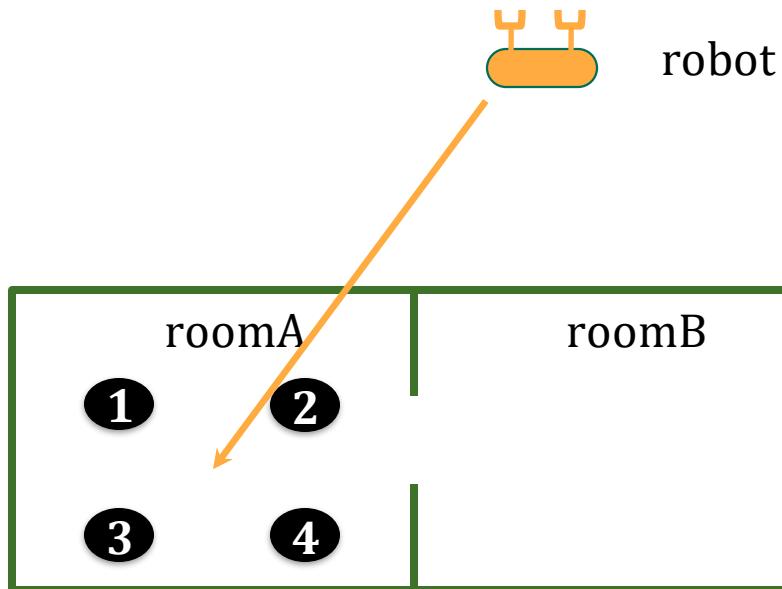


# Standardised Planning with PDDL

- Where STRIPS is a specific planner/language, a more recent standardised planner has been created
- Planning Domain Definition Language (PDDL)
  - STRIPS plus extensions, common assumptions, benefits/shortfalls...
- See <http://lcas.lincoln.ac.uk/fast-downward/> for a planner that you can play around with
- Example from this planner: moving objects
  - Robot domain
  - Robot problem

# A brief PDDL example: World

```
(define (problem strips-gripper-x-1)
  :domain gripper-strips
  :objects roomA roomB ball14 ball13 ball12 ball11 left right)
  :init (room roomA
    room roomB
    ball ball14
    ball ball13
    ball ball12
    ball ball11
    at-robbby roomA
    free left
    free right)
  :at ball14 roomA
  :at ball13 roomA
  :at ball12 roomA
  :at ball11 roomA
  :gripper left)
  :gripper right))
```



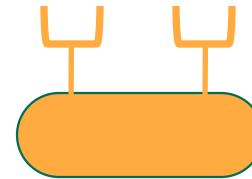
# A brief PDDL example: Robot

```
line domain gripper-strips
:s predicates (room ?r)
  (ball ?b)
  (gripper ?g)
  (at-robby ?r)
  (at ?b ?r)
  (free ?g)
  (carry ?o ?g)

action move
:parameters (?from ?to)
:precondition (and (room ?from) (room ?to) (at-robby ?from))
:effect (and (at-robby ?to)
  (not (at-robby ?from)))

action pick
:parameters (?obj ?from ?gripper)
:precondition (and (ball ?obj) (?from ?from) (?gripper ?gripper))
  (at ?obj ?from) (at-robby ?from) (free ?gripper))
:effect (and (carry ?obj ?gripper)
  (not (at ?obj ?from))
  (not (free ?gripper)))

action drop
:parameters (?obj ?from ?gripper)
:precondition (and (ball ?obj) (?from ?from) (?gripper ?gripper))
  (carry ?obj ?gripper) (at-robby ?from))
:effect (and (at ?obj ?from)
  (free ?gripper)
  (not (carry ?obj ?gripper))))
```



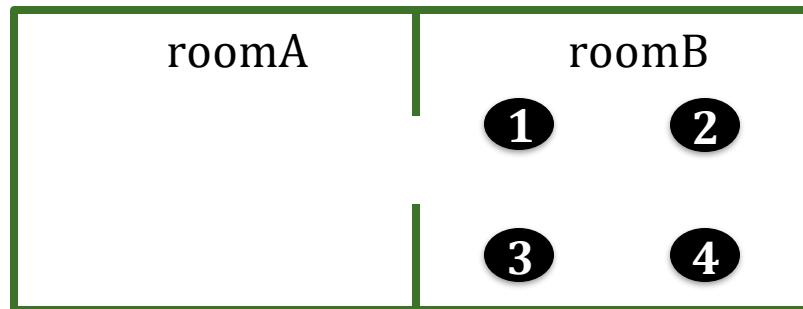
Robot can:

- Move
- Pick
- Drop

With left and  
right grippers

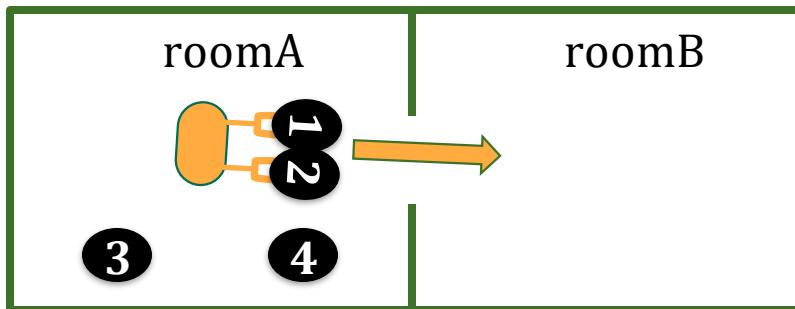
# A brief PDDL example: Goal

```
(:goal (and (at ball4 roomb)
 (at ball3 roomb)
 (at ball2 roomb)
 (at ball1 roomb)))
```



# A brief PDDL example: Plan

```
(pick ball1 rooma left)
(pick ball2 rooma right)
(move rooma roomb)
(drop ball1 roomb left)
(drop ball2 roomb right)
(move roomb rooma)
(pick ball3 rooma left)
(pick ball4 rooma right)
(move rooma roomb)
(drop ball3 roomb left)
(drop ball4 roomb right)
: cost = 11 (unit cost)
```



# Deliberative Architecture Limitations

- Closed World problem
  - All information is present – nothing unexpected, no unanticipated consequences, etc
- The Frame problem
  - What is and is not relevant? Should enumerate all states, even if unchanged – becomes intractable...
- Brittleness problem
  - Can't handle change not affected by the agent (wrong model?)
- Uncertainty problem
  - How should this be handled in a symbolic planner that assumes crisp knowledge, and true/false conditionals?
- Computational load
  - High load leads to slow reactivity

## Direct reaction against deliberative models

Emphasis on fast reaction to low-level sensory information, without involved processing and planning

- Partly inspired by work in biology/CogSci
- Integration of sensory information not necessary

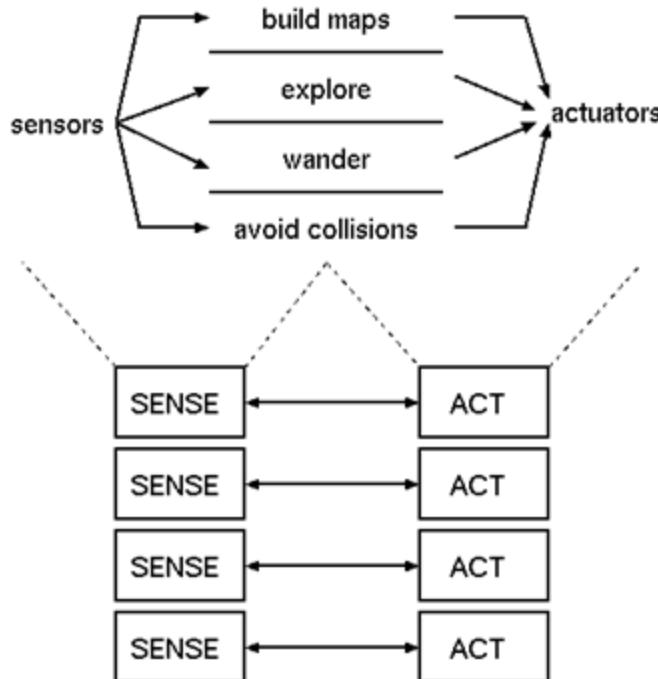
Basic process:

1. Sensory input acquired
2. Multiple parallel behaviours result in overt agent action(s)

### REACTIVE ARCHITECTURES



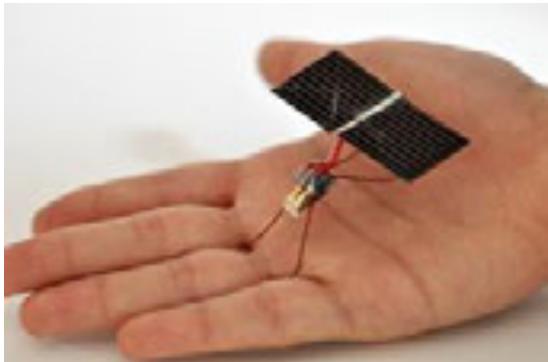
# Vertical decomposition



- Contrast to the deliberative approach
- Vertical decomposition of tasks
  - Simultaneously operating pairs of sensing and acting

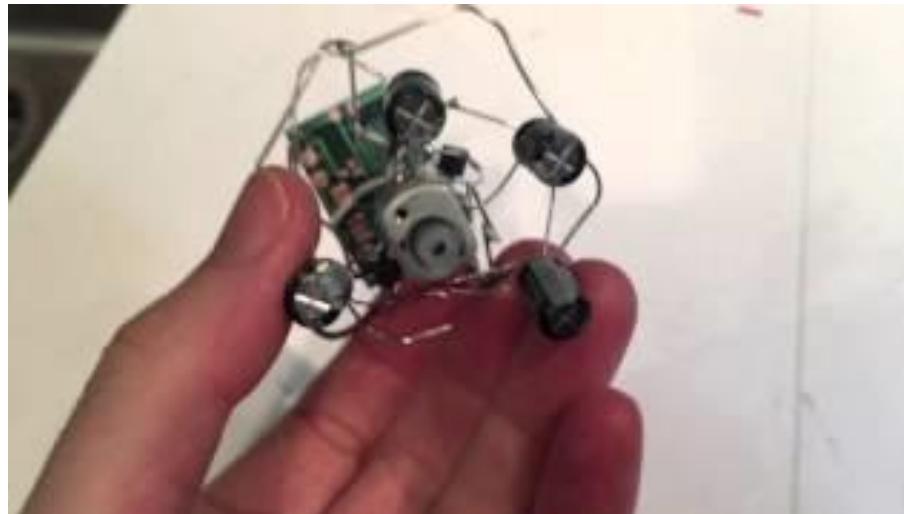
# Benefits of Reactive Architectures

- No internal world model needed
  - “the world is its own best model”
  - Cheap and fast!
- Real-time behavioural control
- Can have emergence of complex behaviour with little design effort



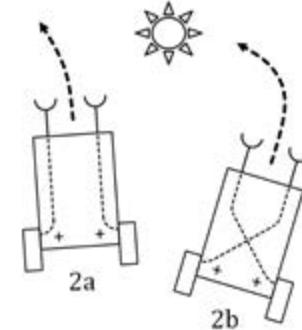
# Benefits of Reactive Architectures

- Solar powered robot jumping around when it gets hit with 9000 lumens



# Behaviour-based Robotics

- These are typically reactive
  - Tightly coupled to sensory information (no “planning”)
  - Lacking in (or only minimal) internal state
  - Hence fast acting
- In the design of the behaviours, strong implicit role for the embodiment of the robot
  - i.e. the behaviour depends on the specific array of sensors, motors and body used
  - Remember the Braitenberg vehicles...
- Hence also interaction with the environment
  - Cannot necessarily fully account for behaviour just by looking at the control architecture, also need to consider the environment



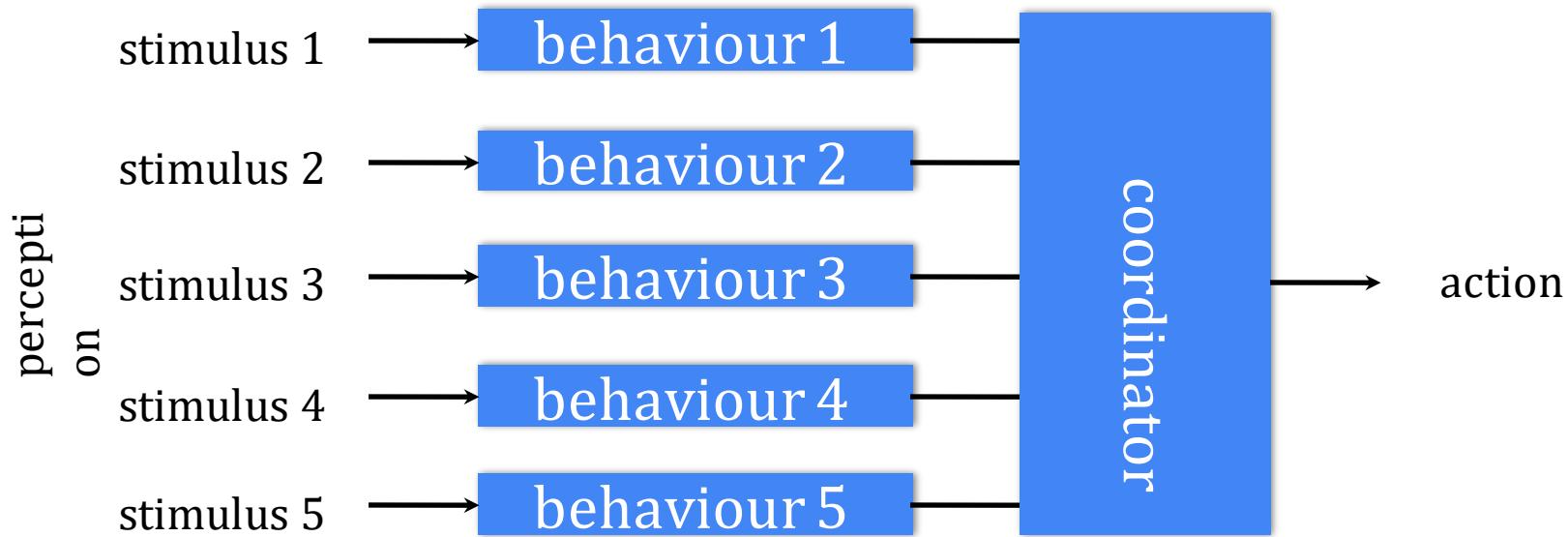
# No World Model...

- no memory (no internal state, no model of the world)
  - Can be difficult to choose the most appropriate behaviour



- one way to deal with this limitation is to use ego-centric representations
  - Provides some structure: helps with choosing what to do

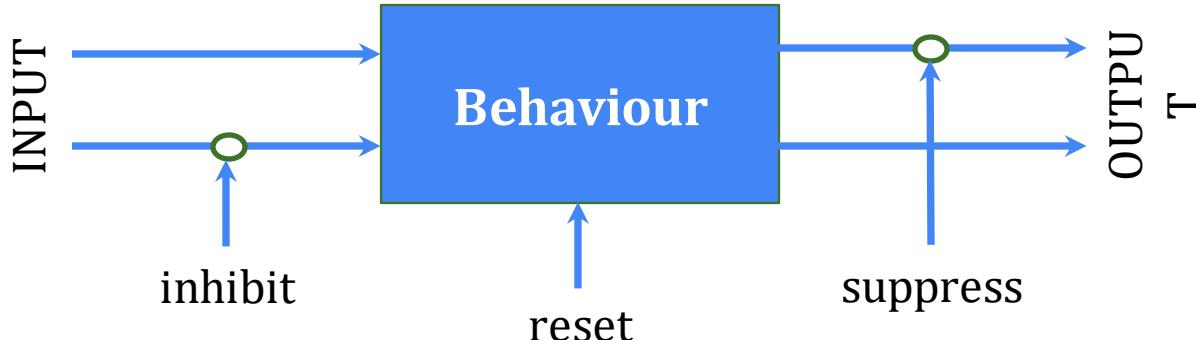
# Multiple Behaviours (*recap*)



- What role for the coordinator?
  - Competitive: e.g. winner-takes-all
  - Cooperative: e.g. blending outputs through addition
  - Hybrid: e.g. activation/inhibition dynamics (Maes, 1989)

# Subsumption Architecture

- A single example case of behaviour-based control architecture
  - Though the best known
  - A *design methodology*
- Gets around the coordinator problem by having higher level behaviours “subsume” lower level behaviours
  - i.e. some behaviours can over-ride others



# Example: Genghis

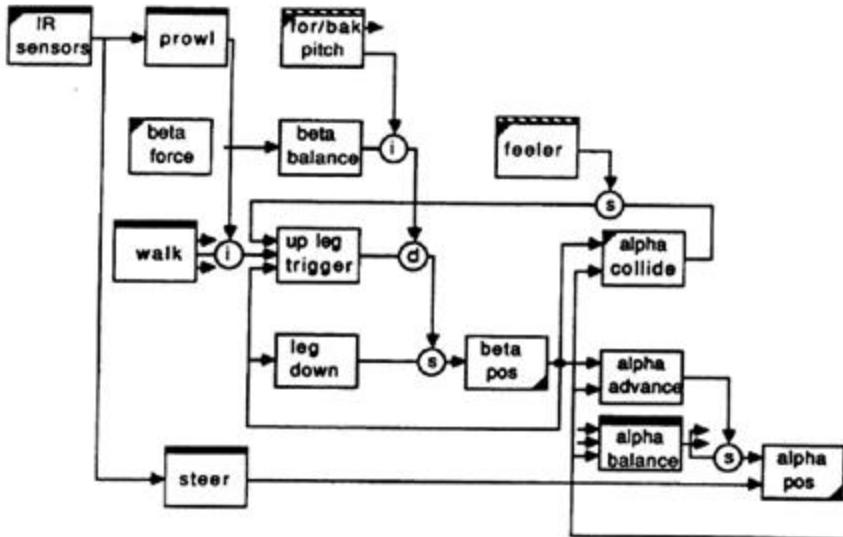
- Example architecture of Genghis

- Blocks:

- Sensor input
- Behaviours
- ...

- Note the relative complexity of the inter-connections

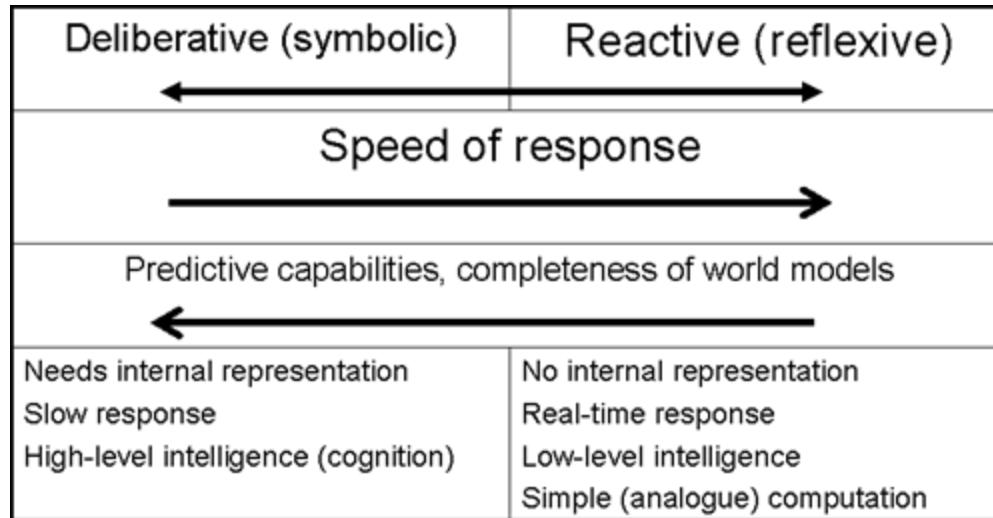
- Hand-designed and tuned behaviours



# Reactive Architecture Limitations

- Oriented to specific Task (lack of generalisation)
- Based on, and constrained by, particular robot embodiment
- Sensitivity to Sensor noise
- Lack of Planning
  - Lack of internal state
  - Learning as a problem
- Stimulus-Response alone insufficient to account for intelligence
- Emergence of complex behaviour is a design problem

# Story so far...

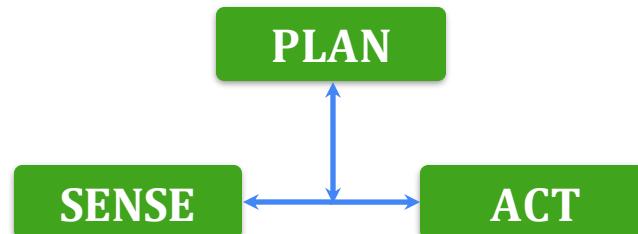


Trying to get the best of both Deliberative and Reactive paradigms

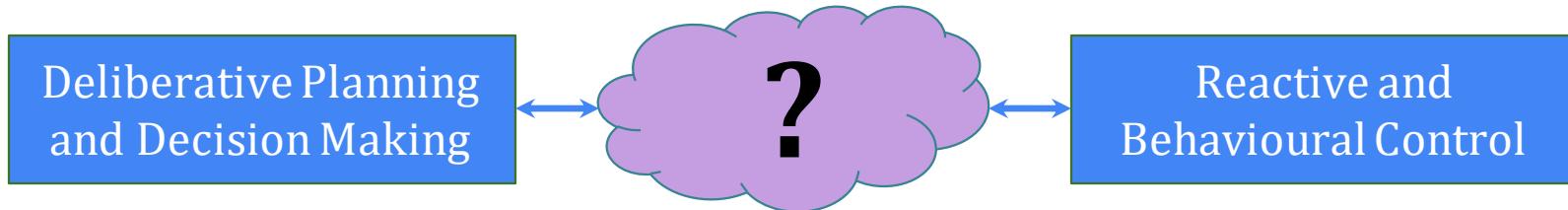
Some planning where appropriate, but maintaining ability to respond quickly to the environment

Multiple levels of control, each focused on a different aspect

## HYBRID ARCHITECTURES



# Linking Deliberative with Reactive

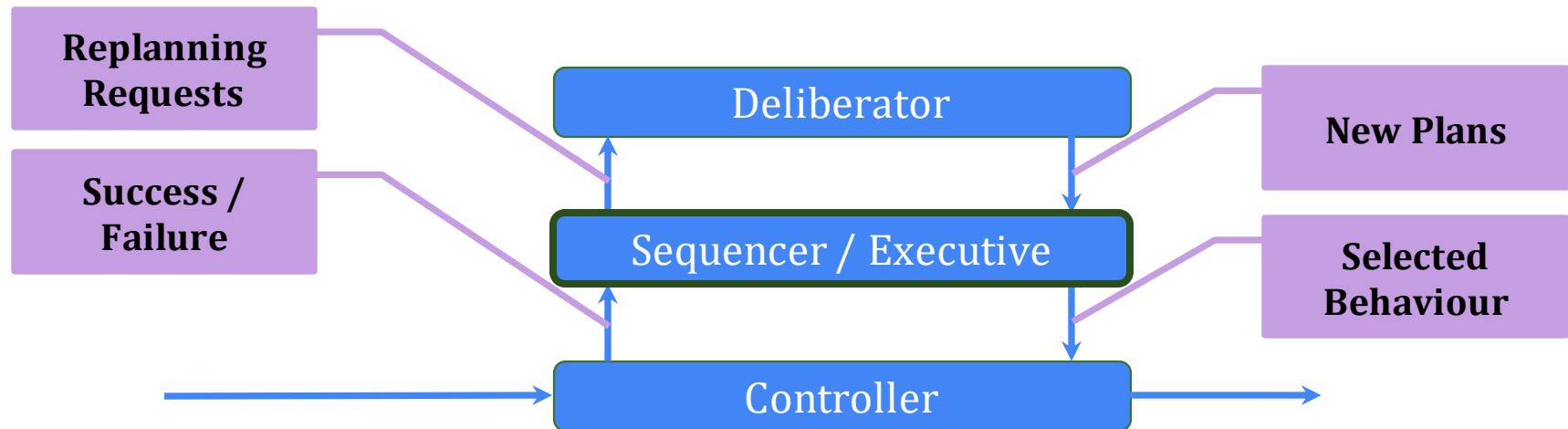


- To get best of both, use both...
  - Symbolic processing and world models/maps for planning
  - Reactive behaviours for fast, responsive action
- How best to link the two together?

# Temporal decomposition

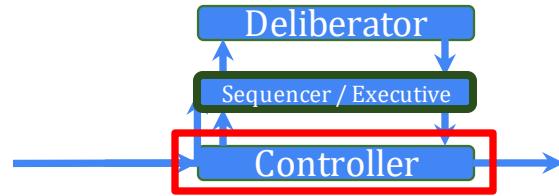
- In other architectures:
  - Horizontal decomposition of the task in Deliberative architectures
    - Can be slow...
  - Vertical decomposition of tasks in Reactive architectures
    - Can be too quick? (sensitive to noise)
- Resolve this by using time-appropriate processes:
  - Different layers with different 'speeds' of processing

# Three Tier (3T) Architectures

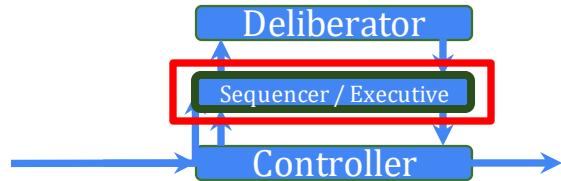


# Controller

- Library of Behaviours
  - May be handcrafted
  - E.g. the behaviour-based approach
- Must be fast:
  - Avoiding internal state (except to estimate state), planning, etc
  - Stable closed-loop control
- Must be able to detect failure
  - This allows the Sequencer/Executive to call another behaviour, or call for a re-planning

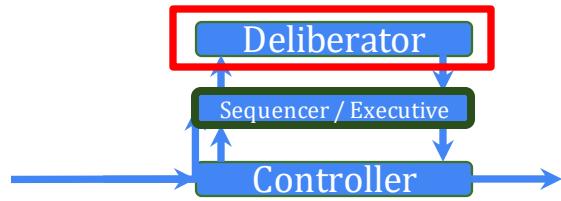


# Sequencer / Executive



- This drives the control of the system
  - Not strictly speaking hierarchical, since this middle layer is doing the coordination...
- Selects which behaviour will be active
  - Sequences, loops, conditionals, threads, etc
- Initiates behaviour and planning:
  - Examines state of the world
  - Gets success/failure of controller
  - Queries deliberator if necessary
- E.g find and follow maze wall, remember sequence of turns

# Deliberator



- Operates on its internal state – the world model
  - No sensing
- Time consuming and computationally intensive tasks
  - Maps and route planning
- No commitment to the means of processing here
  - Could be STRIPS-style, or anything else
- Its operation is directed (start/stop) by the Sequencer/Executive
- Example:
  - In a maze, once the location is recognised, plan a route to the exit

# Performing a sample task

- I want coffee...

And I want it now...

- What do I need to take into account?

- Where is it?

**static**

- How to get there?

**static**

- Walk and Open doors... (physical conventions)

**dynamic**

- Don't walk into people... (personal conventions)

**dynamic**

- Queue... (cultural conventions)

**dynamic**

- Pay... (legal conventions)

**dynamic**

- Refreshment

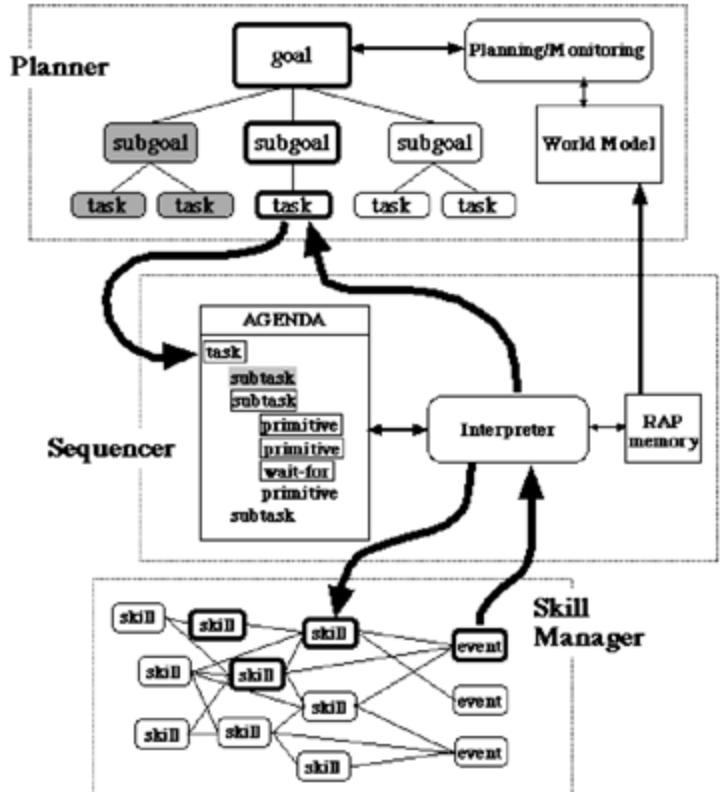
**dynamic**

**target!**



Both planning and reactive components required to solve the task

# Example Systems: NASA



- A 3T architecture used by NASA (left)
  - Automated control of systems and devices
  - Also shared autonomy at various layers in the hierarchy
- Generally, 3T architectures are among the most prevalent
  - Hybrid approaches in general form the basis of the majority of systems in use today

# Advantages

- Part of the advantage is the specification of overall structure, and principle of operation, rather than precise mechanism
  - Maintains flexibility
  - If one algorithm not appropriate, swap it out for another
- According to Erann Gat (1998):

*“lines between the components of the three layer architecture can be blurred to accommodate reality”*

  - Flexibility depending on the application
  - “If, as seems likely, there is no One True Architecture, and intelligence relies on a hodgepodge of techniques, then the three layer architecture offers itself as a way to help organise the mess”*
  - Guiding principle rather than prescriptive on mechanism

# Hybrid Architecture Issues

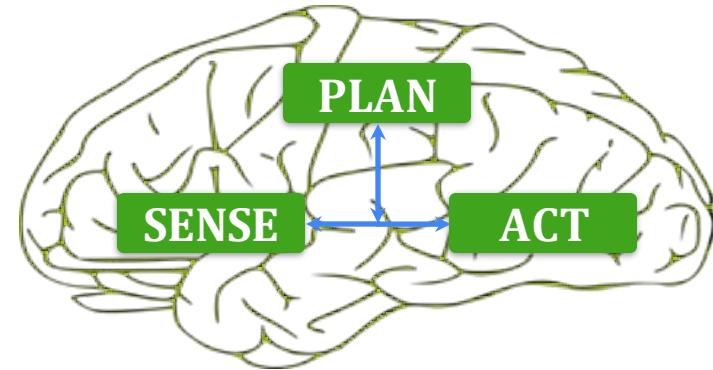
- The central role of the Sequencer / Executive
  - Clearly of central importance in the 3T approach
  - Thus needs particular attention
- “Symbol grounding”
  - Relevance of the representations (e.g. symbolic) to the instantiation/actions of the robot system it is planning for
  - Circumvented by appropriate design

Explicitly taking into account the way that humans may process information and act

- Not introspection, but evidence...

Broad category, including inspiration from psychology, CogSci, neuroscience, etc

Overlaps with the previous paradigms, particularly hybrid



## COGNITIVE ARCHITECTURES

# What is a Cognitive Architecture?

Cognitive Architecture...

*“...is the overall, essential structure and process of a domain-  
generic computational cognitive model, used for a broad, multiple-  
level, multiple-domain analysis of cognition and behavior...”*

(Sun, 2004)

# Unpacking the definition

- Cognition and Behaviour
  - Both an account of the internal workings, and also how this generates behaviour
- Multiple-level
  - Macro and micro aspects, over multiple time-scales
  - Similar in this sense to hybrid architectures
- Domain-general
  - Not restricted to a specific task, but general modes of operation applicable across domains
  - Compare/contrast with deliberative/reactive architectures
- Structure and Process
  - The mechanisms (and knowledge) required to achieve this

# Levels of control

- Brain

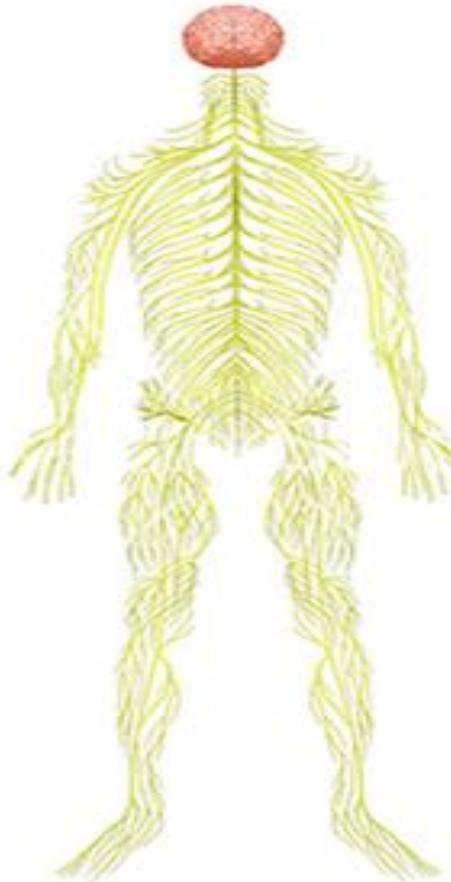
- Cognitive processing
- Memory, etc

- Joint between the two...

- Embodied cognition (influence of the body)
- Influence of drives (e.g. hunger)
- Sensory information

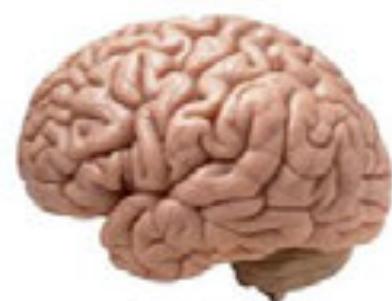
- Body

- Reflexes
- Reactions
- Independent of brain...



# Why take inspiration from humans?

- Humans demonstrate the best example of highly complex intelligence, and so could form useful design guides
  - To solve the difficult problem of general-purpose intelligence, start somewhere...
- If robots are to interact with humans in human environments, then having them endowed with some human-like cognitive features could be useful
  - To understand humans and their behaviour, to facilitate interaction
- We will return to this in the coming weeks...



# Two main approaches (and others besides...)

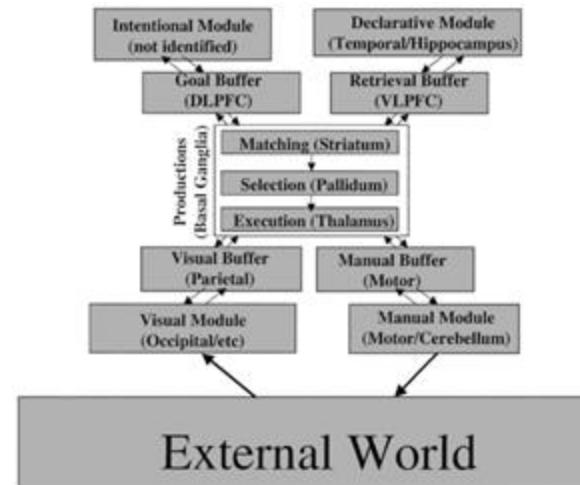
1. Derive set of mechanisms to use from human behavioural or other data
  - A “top-down” perspective
  - Can use this as a model of human behaviour
  - Typically based on data from psychology (overt behaviour)
2. Try to model fundamental principles of organisation of cognition, and implement these
  - A “bottom-up” perspective
  - Try to match to certain aspects of (human) behaviour
  - Typically derived from data closer to biology

# Note...

There are many, many cognitive architectures, these are only two examples...

# Example of Top-down: ACT-R

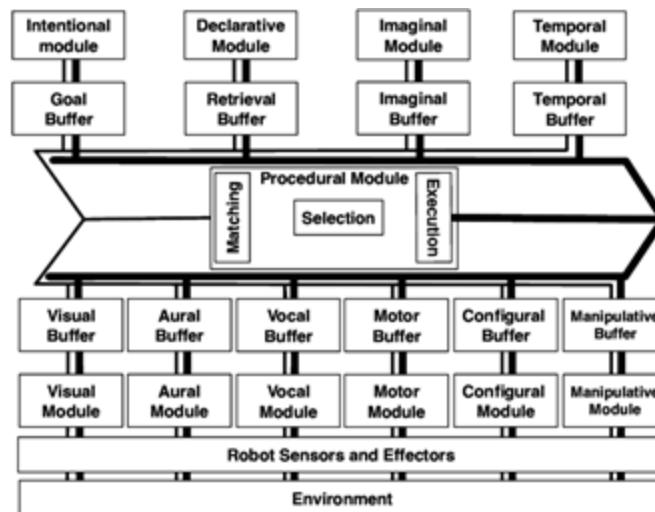
- ACT-R is an established cognitive architecture that has been applied to numerous models of human behaviour (reaction times, cognitive load, etc)
- Based on evidence from psychology, physiology, etc
- Hybrid Symbolic/Sub-symbolic processing
- <http://act-r.psy.cmu.edu/about/>



# Example of Top-down: ACT-R

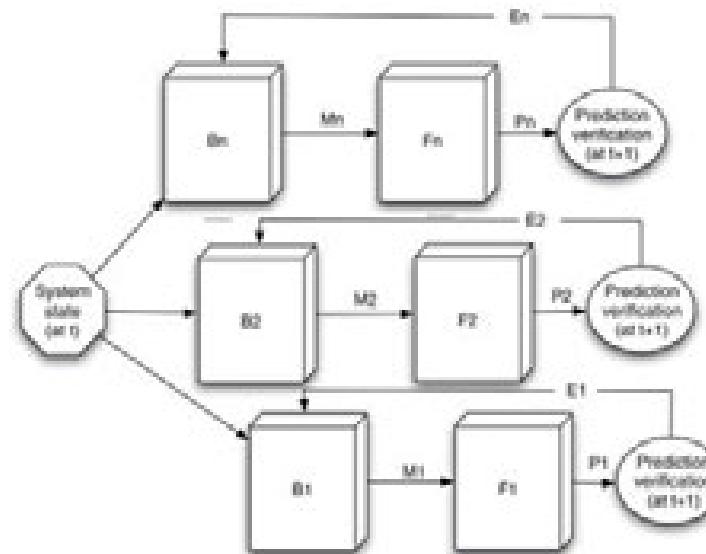


Images taken from (Trafton et al, 2013)



# Example of Bottom-up: HAMMER

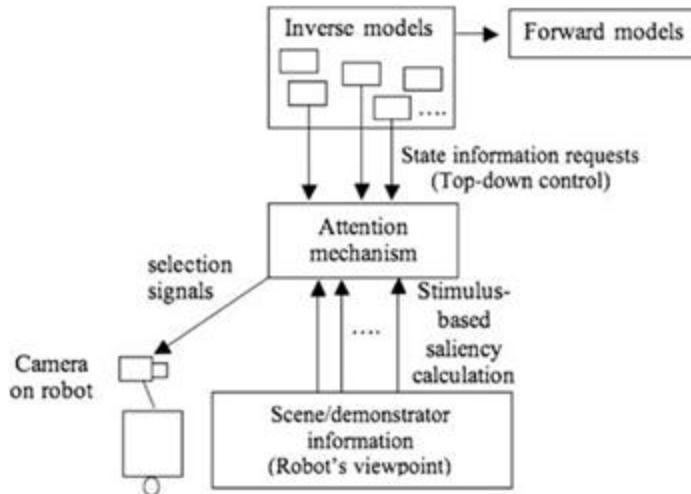
- Hierarchical, Attentive, Multiple Models for Execution and Recognition (HAMMER)
- Fundamentally based on Forward /Inverse model couplings, and applied to imitation, learning, assistance, etc
  - Strong theoretical foundations in psychology, neuroscience...
  - Comparing the different applications in a principled manner



# Example of Bottom-up: HAMMER



Cooperative  
wheelchair control

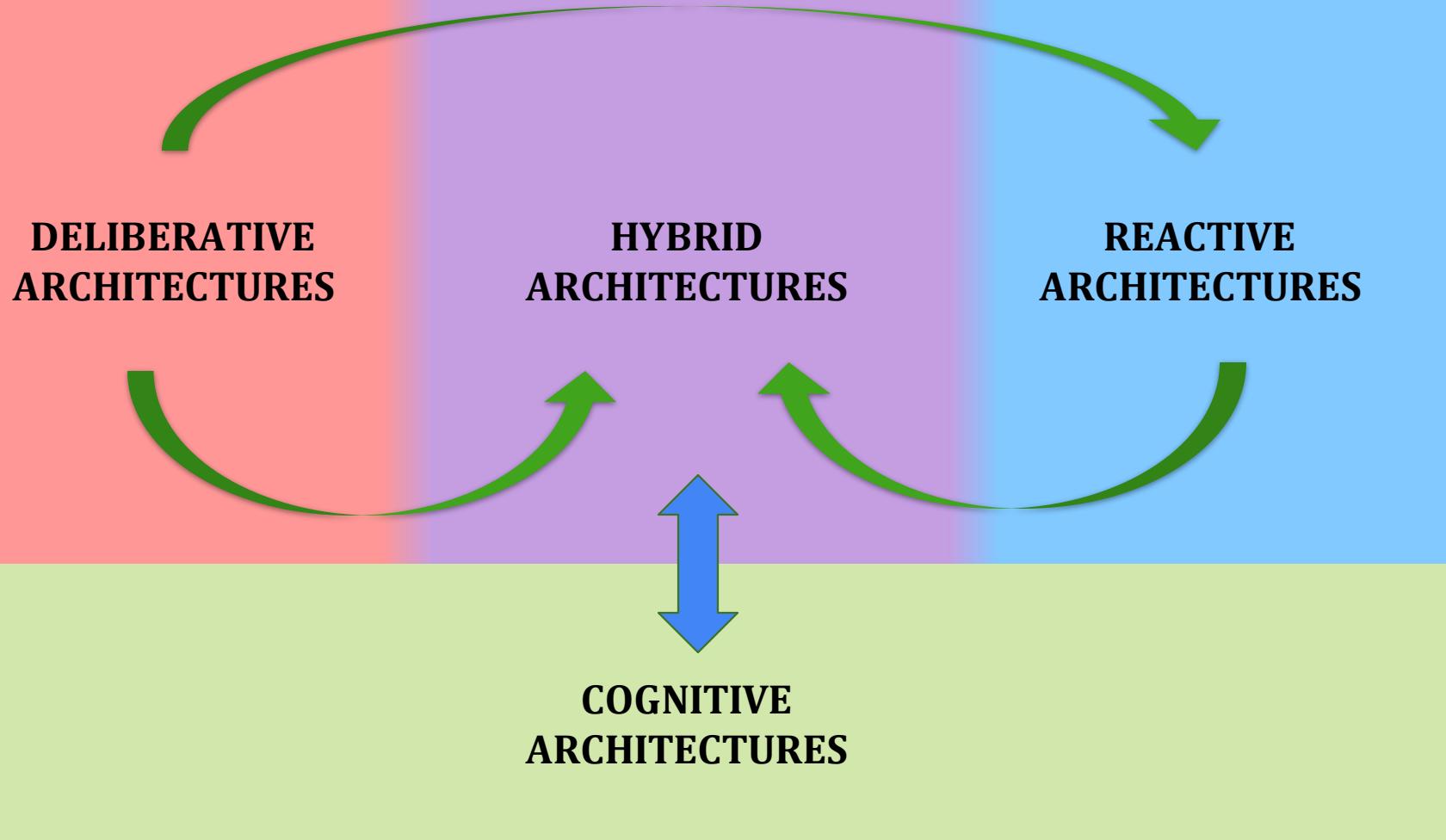


Prediction of  
intentions

See [https://www.youtube.com/watch?v=CaH82\\_WzAeQ](https://www.youtube.com/watch?v=CaH82_WzAeQ) for a lecture by Prof. Yiannis Demiris on this, and other, work

# Cognitive Architecture Issues

- How related to biology should it be?
  - Inspiration or constraints?
- Suitable level of abstraction?
  - Behaviour or mechanism?
- What is the purpose of using a Cognitive Architecture?
  - Functional or explanatory?

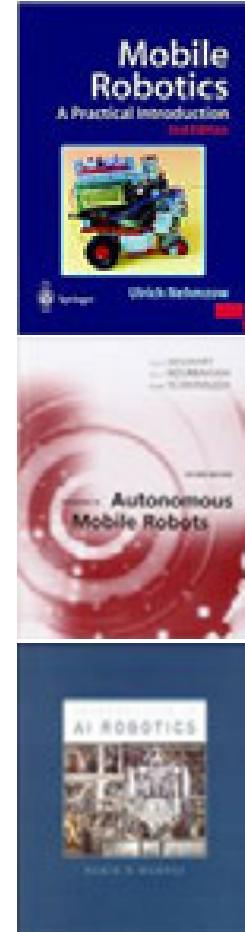


# Tip of the Iceberg...

- Many individual examples of systems, each with variations, for each of the types of architecture mentioned
  - Have focused on control of individual robots
- Some control architectures/methods not detailed:
  - Multi-Agent Systems
    - Coordinating multiple robots
    - Emergent behaviour from swarms of robots
  - Inter-Agent Communication/Synchronisation
    - Social interaction, etc
    - Though something related next week...
  - Etc...

# References / Reading

- Brooks, R.A., 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), pp.14–23.
- Brooks, R.A., 1990. Elephants don't play chess. *Robotics and Autonomous Systems*, 6, pp.3–15.
- Gat, E., 1998. "On Three-Layer Architectures." Artificial Intelligence and Mobile Robotics, in D. Kortenkamp, R. P. Bonnasso and R. Murphy (eds.), AAAI Press, pp.195-210.
- Kortenkamp, D. et al, 1998. Three NASA application domains for integrated planning, scheduling and execution. *AAAI AIPS Workshop*, pp.88-93
- Maes, P., 1989. How to do the right thing. *Connection Science*, 1(3), pp.291–232.
- Murphy, R., 2000. *Introduction to AI Robotics*, MIT Press.
- Sun, R., 2004. Desiderata for cognitive architectures. *Philosophical Psychology*, 17(3), pp.341–373.



# Next week...

- Human-Robot Interaction
- Relating to the concepts from today, but also considering foundational concepts (and some applications)

**CMP3101M AMR – Week 11**

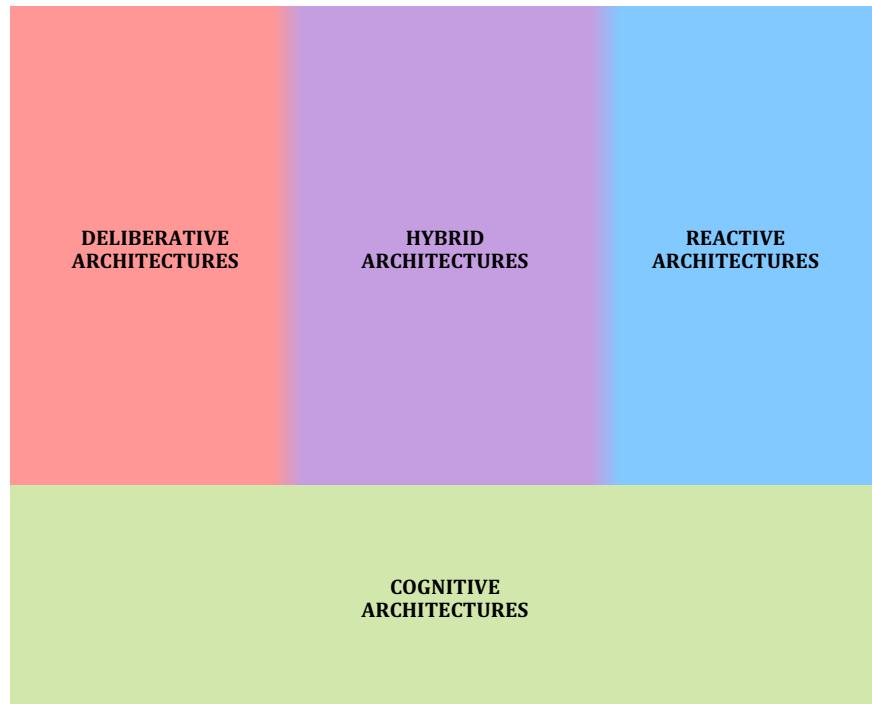
# **Human-Robot Interaction**

## **part 1**

**Dr. Athanasios Polydoros\***

\*slides from Prof. Marc Hanheide & Paul Baxter

# Last Week...



- Control Architectures for Autonomy
- Why they are necessary
- Three (+1) main paradigms

# What is HRI?

Human-Robot Interaction (HRI) is:

*... a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with humans. Interaction, by definition, requires communication between robots and humans.*

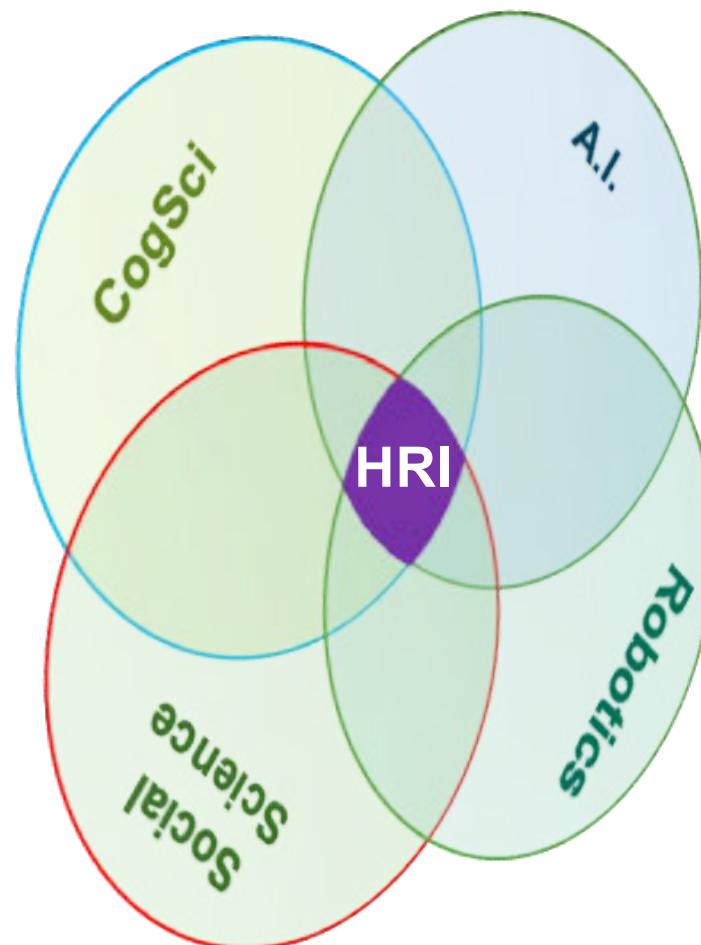
<http://humanrobotinteraction.org/1-introduction/>

Some overlaps with Human-Computer Interaction, so refresh your memory of last year's HCI module...

- In fact, origins of HRI lie in HCI
- Particularly in terms of development and evaluation methodologies

# Aspects of HRI

- Psychology
  - Human Factors
- 
- Sociology / Social Science
  - Human-Computer Interaction
- 
- Computer Science
  - Mechatronics



From (Baxter et al, 2016)

# Humans and Robots

## Humans

- Interaction partner
- Social agent 
- Target of research 
- Source of knowledge
- Recipient of help
- Caregiver
- Companion
- ...



## Robots

- Interaction partner
- Social agent?
- Target of development
- Source of knowledge
- Recipient of help
- Caregiver
- Companion
- ...

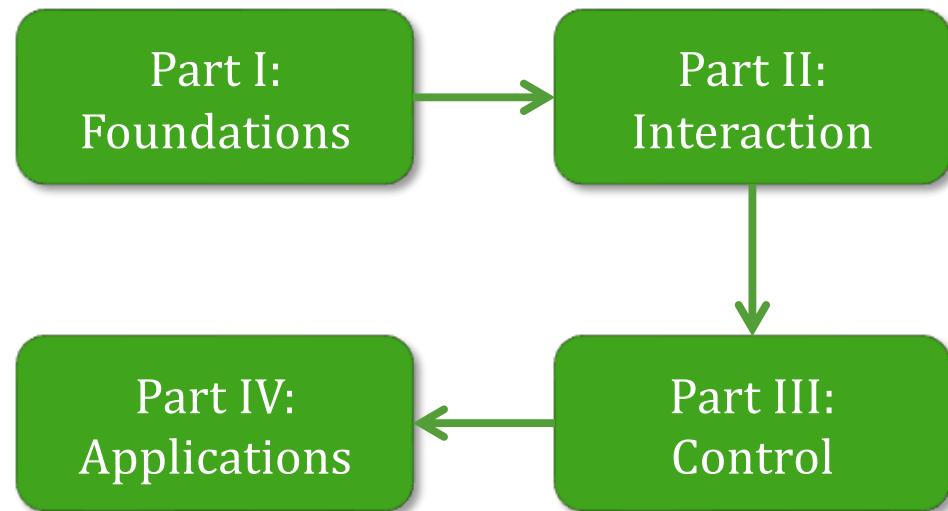


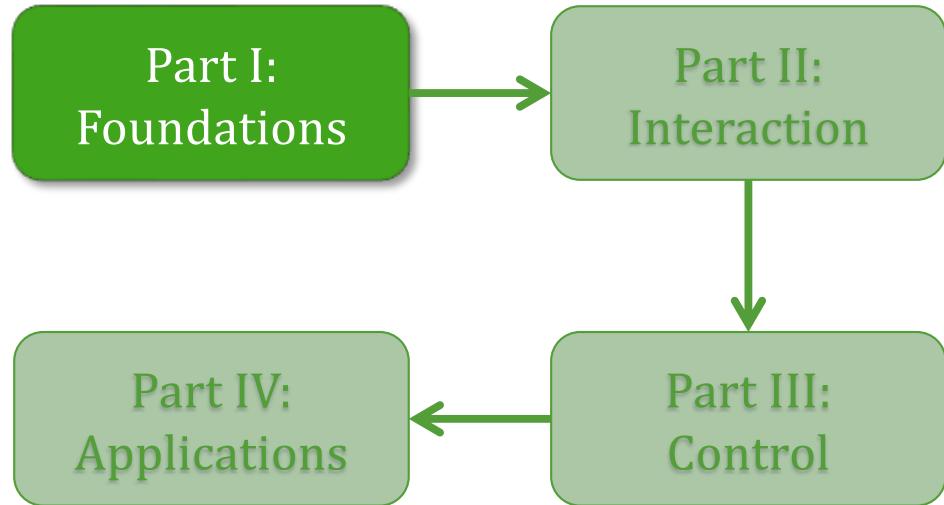
# Why HRI in AMR?

- Purpose of robotics:
  - Automation: doing (boring/repetitive) jobs for people
  - Replacing people in risky/dangerous situations
  - ...
- Could also use for helping people:
  - Physical rehabilitation (exoskeletons, haptics, etc)
  - Social therapy, etc (e.g. socially-assistive robotics - SAR)
  - ...
- And as a tool for understanding people...?
  - The psychology/social science perspective

**The interaction of humans and robots is pervasive in robotics**  
**-> hence HRI...**

# Today...





# Part I: Foundations

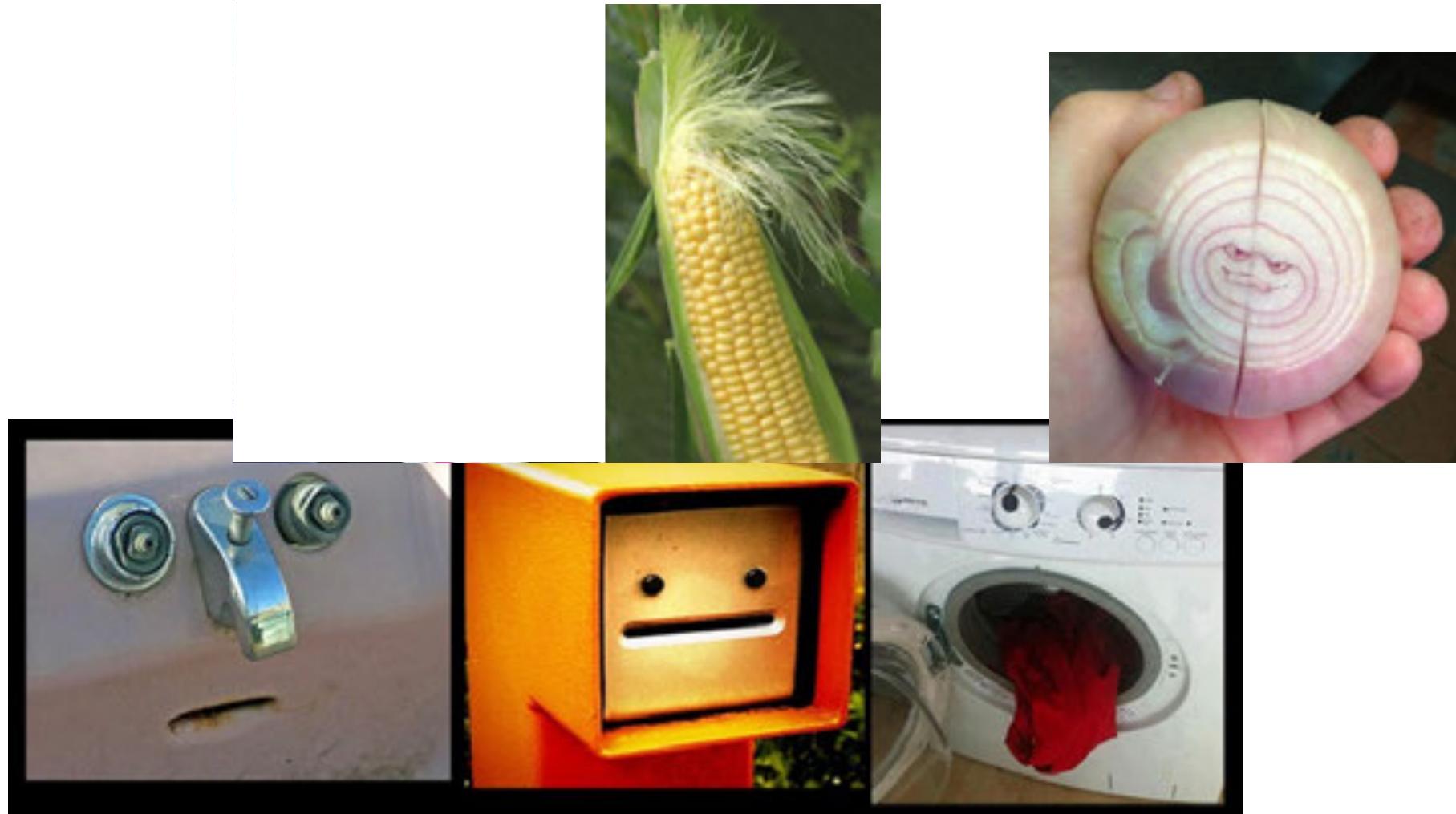
Assumptions and characteristics

# Anthropomorphism

- Is the tendency to attribute human characteristics to inanimate objects, animals and others with a view to helping us rationalise their actions
  - (Duffy, 2003)
- Many examples in cartoons (Disney being particularly prolific)
- “the strategy of interpreting the behaviour of an entity (person, animal, artefact, whatever) by treating it as if it were a rational agent who governed its ‘choice’ of ‘action’ by a ‘consideration’ of its ‘beliefs’ and ‘desires’”
  - (Dennet, 1996): the intentional stance
- Embracing this concept in HRI
  - Taking advantage of it rather than trying to avoid it
  - Appearance and Behaviour
- Related concepts: active perception, and gestalt psychology from HCI – wanting to find and group information in ‘meaningful’ ways

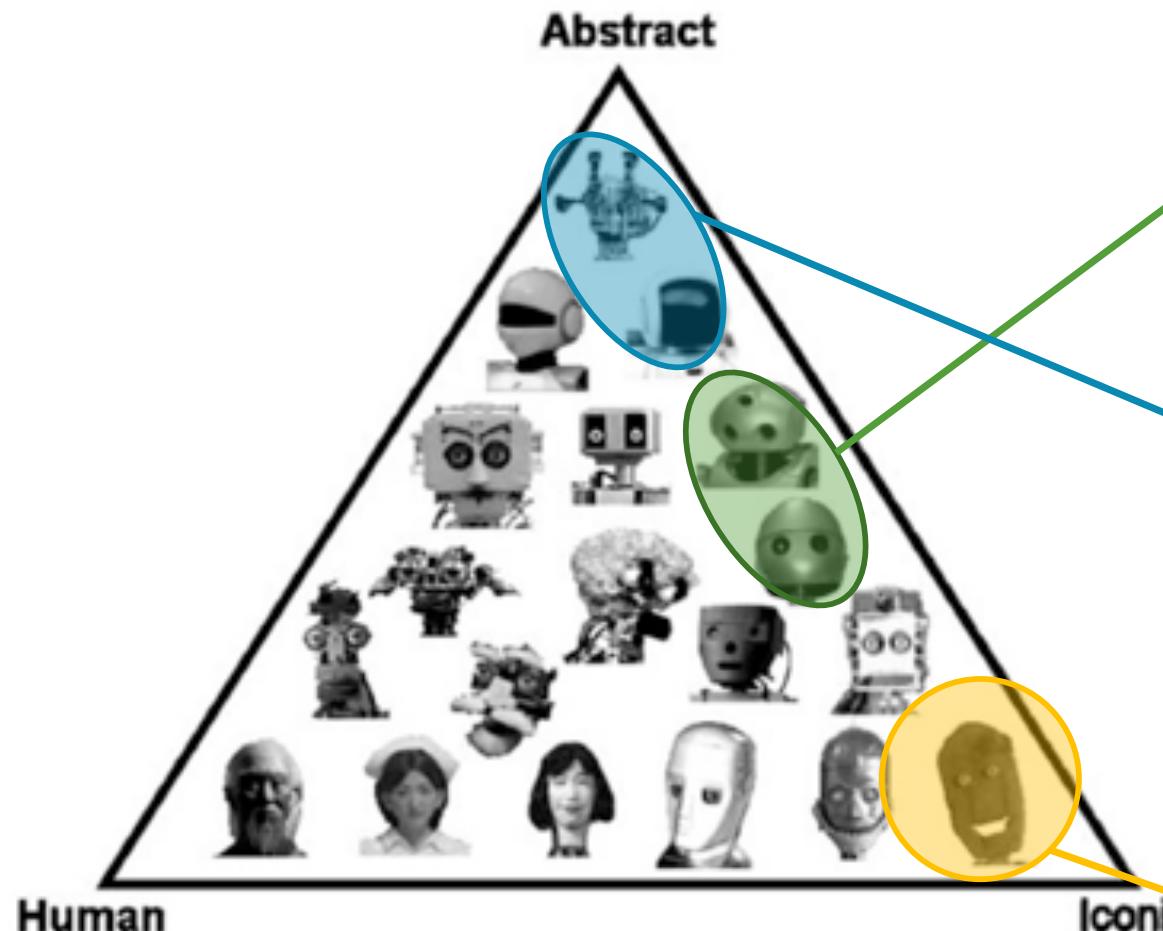


# Anthropomorphism: Appearance



**Pareidolia** – perceiving a familiar pattern (typically face) where there is none

# Anthropomorphism: Robot Faces



Design space for humanoid heads (Duffy, 2003)



# Anthropomorphism: Behaviour



Apparent Behaviour (Heider & Simmel, 1944)

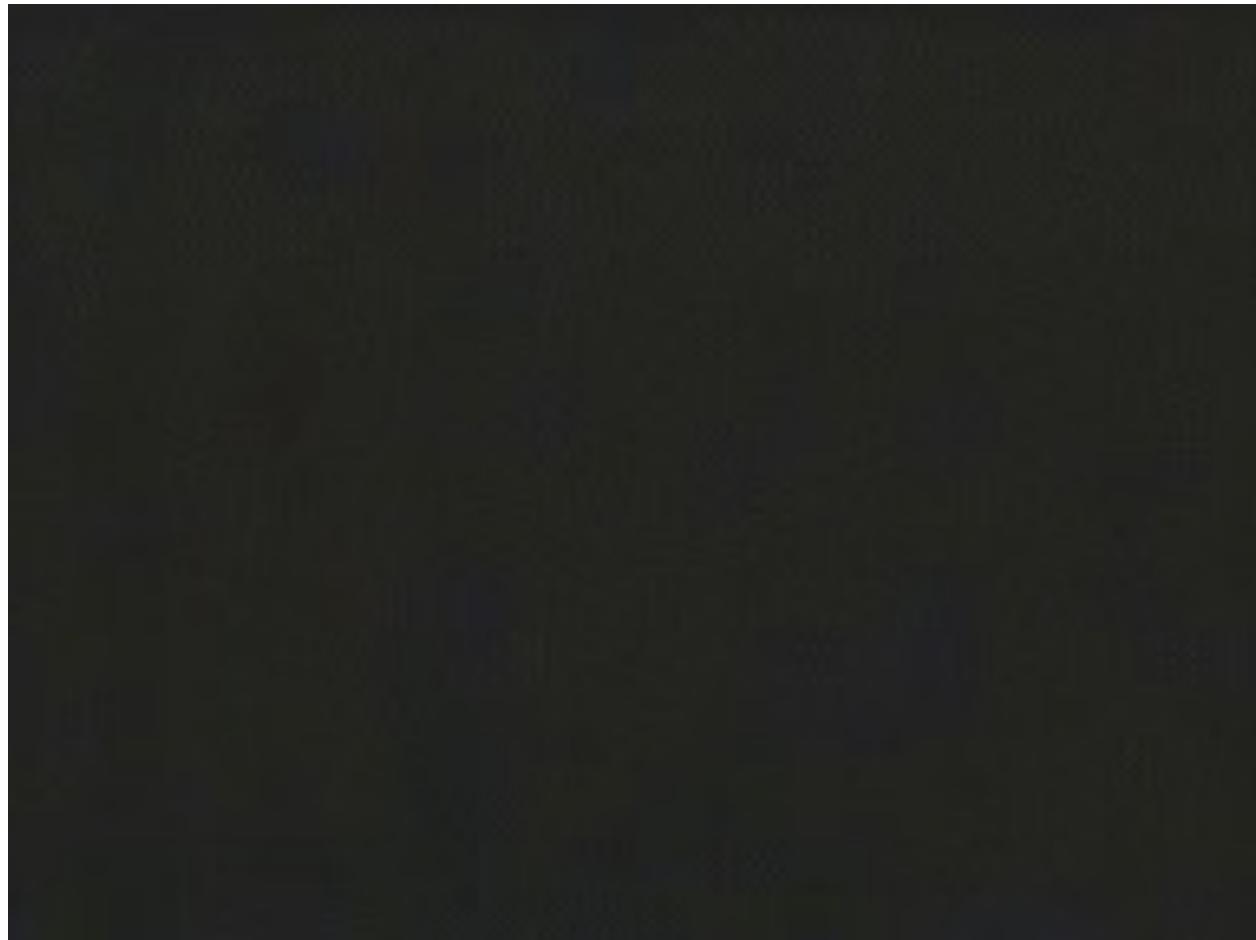
# Anthropomorphism: Behaviour

While working on your assignment, how many of you:



- Swore at your robot?
- Complimented your robot?
- Referred to your robot as he/she?
- Gave your robot a name?

# Anthropomorphism: Behaviour



Full video: <https://www.youtube.com/watch?v=VWeRC6j0fW4>

Also see: [http://cosmo.nyu.edu/hogg/lego/braitenberg\\_vehicles.pdf](http://cosmo.nyu.edu/hogg/lego/braitenberg_vehicles.pdf)

# Anthropomorphism: Robots



Geminoids (Ishiguro et al)

# The Uncanny Valley

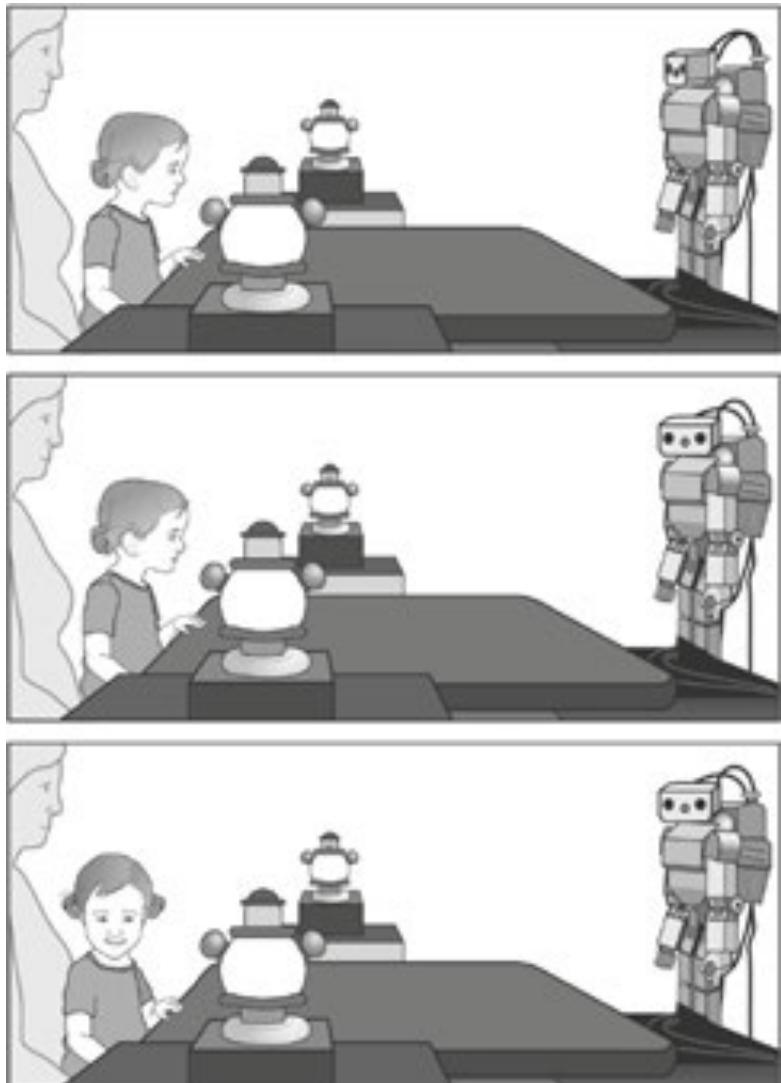


- One possible explanation: a conflict between cues (Moore, 2012)
  - Hence why the “moving” curve more pronounced
  - Greater mismatch between movement and appearance (see anthropomorphism notes above)

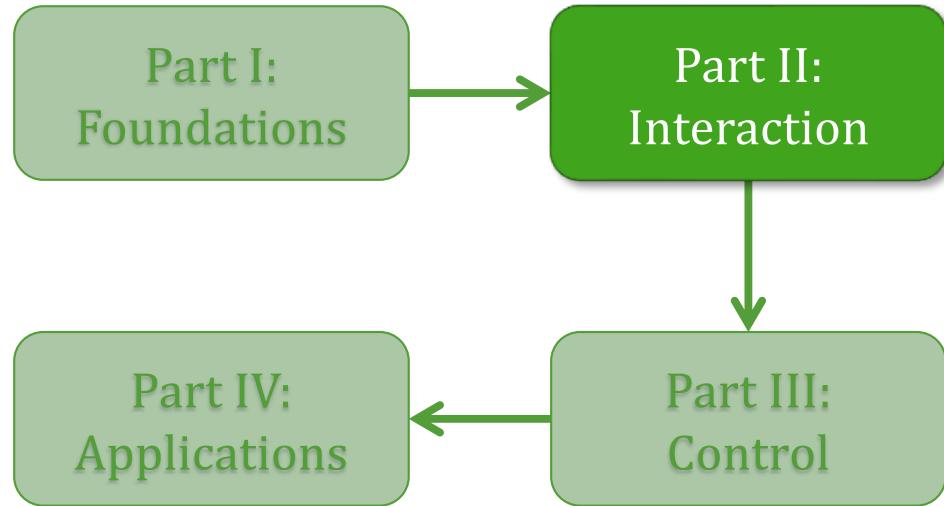
- The “Uncanny Valley”
  - Increasing human likeness increases familiarity...
  - ...however, at a certain point, a sharp drop in familiarity, resulting in revulsion, etc
- Refer to (Mathur et al, 2016) for an overview
- Reasons for this not fully understood

# Attribution of Agency to Robots

- Particularly if certain characteristics are present, people will naturally attribute agency to an inanimate object
  - Though this illusion can be broken in the interaction
- This is (at least partly) learned from experience
- Seen in children with a robot for example
  - Meltzhoff et al (2010)
  - Children watch adult interact with robot (joint attention)
  - These children then more likely to follow gaze when they interact with robot themselves



From Meltzhoff et al (2010)



# Part II: Interaction

Types and contexts

# Fundamentally two modes...

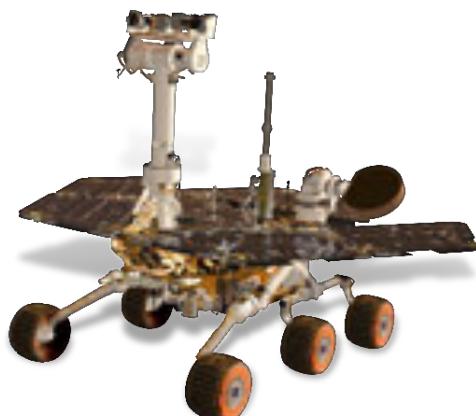
## Proximate

- The human and the robot are in the same space
- Physical and social interactions fall in this category
  - E.g. service robots



## Remote

- The human and the robot are in different locations
  - Maybe even temporally removed
- E.g. teleoperation, supervised control, ...



# Two types of Interaction

## Explicit

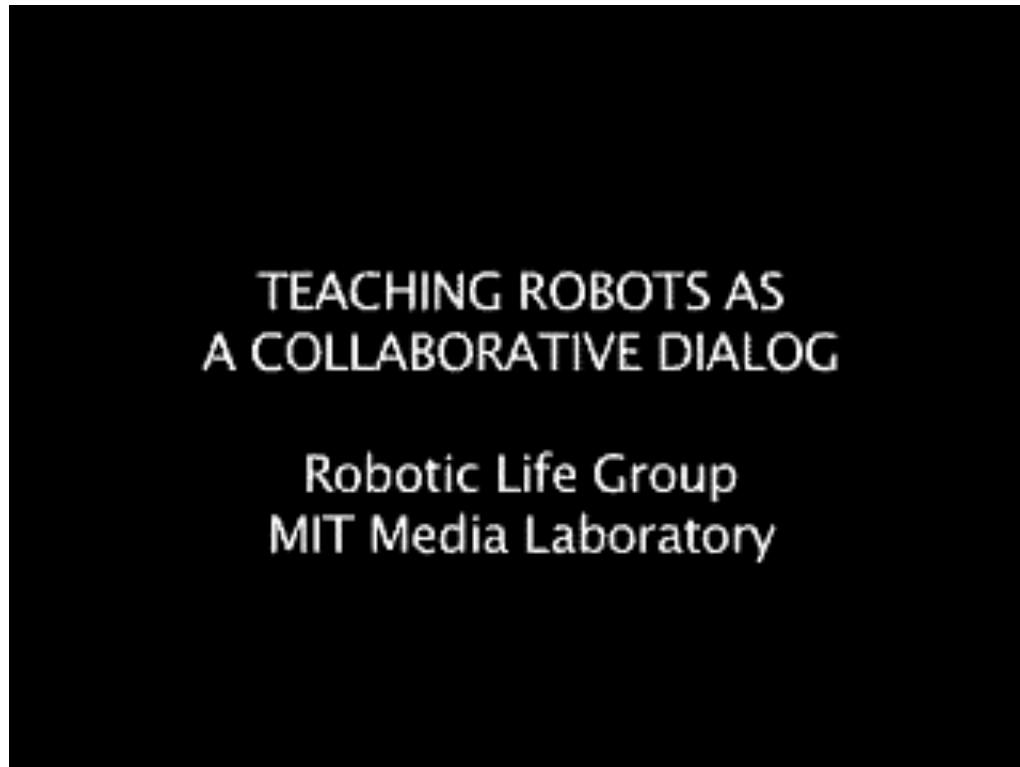
- An action performed with the purpose of eliciting a reaction
- Dialogue: e.g. asking a question
- Manipulation: e.g. handing over an object, or pointing

## Implicit

- Actions are modified due to presence of an other, but no reaction is expected
- Navigation: e.g. avoiding humans in the way

Capacity for overlap between Explicit and Implicit: e.g. avoiding humans, but engaging in some strategies to encourage humans to move out of the way (next week!)

# Explicit Interactions: example



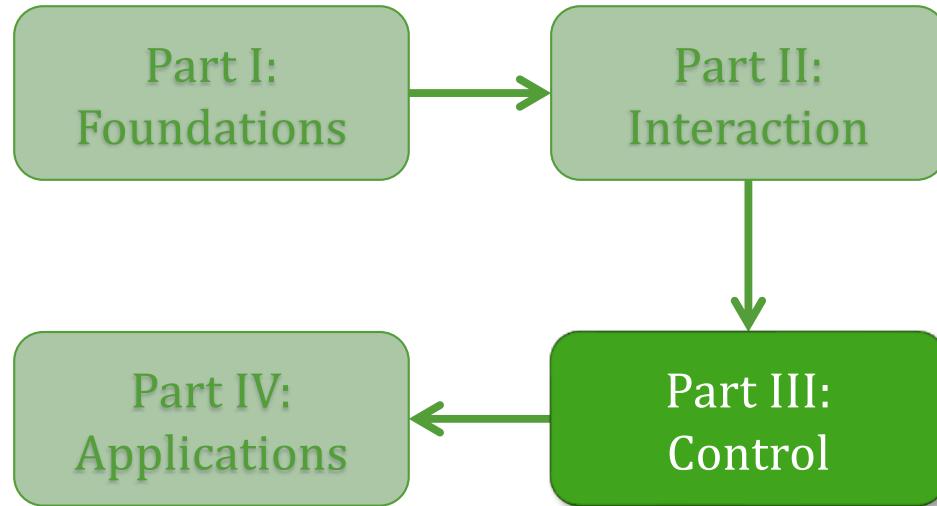
- Leonardo robot with Andrea Thomaz (MIT, 2006)
- Explicit interaction
  - Pointing from human
  - Gaze gestures from robot to indicate ready for next instruction
- See: <https://www.youtube.com/watch?v=GHllFrL7dKM>

# Explicit Interactions: example

- Explicit interaction scenario
  - Robot and human facing each other
  - Using the same workspace
  - Human providing explicit instruction, with gestures
  - Robot performs actions, looks back at human
- Human teaching:
  - Names, attributes, affordances
- Robot learns from:
  - Speech, observation

# Implicit Interactions: example

- E.g. robot navigates through populated environment
- Humans change their behaviour
  - They stop
  - Deviate path to avoid robot
- Interaction not strictly necessary for the task of navigation however:
  - If done correctly, can improve efficiency
  - E.g. shorter path found/planned due to person moving out of the way
  - (this may require explicit interaction strategies – e.g. “please move out of my path”)
- Queueing has similar implicit interaction characteristics



# Part III: Control

Autonomy and architectures

# Control for Autonomous Robots

This should be familiar from last week!

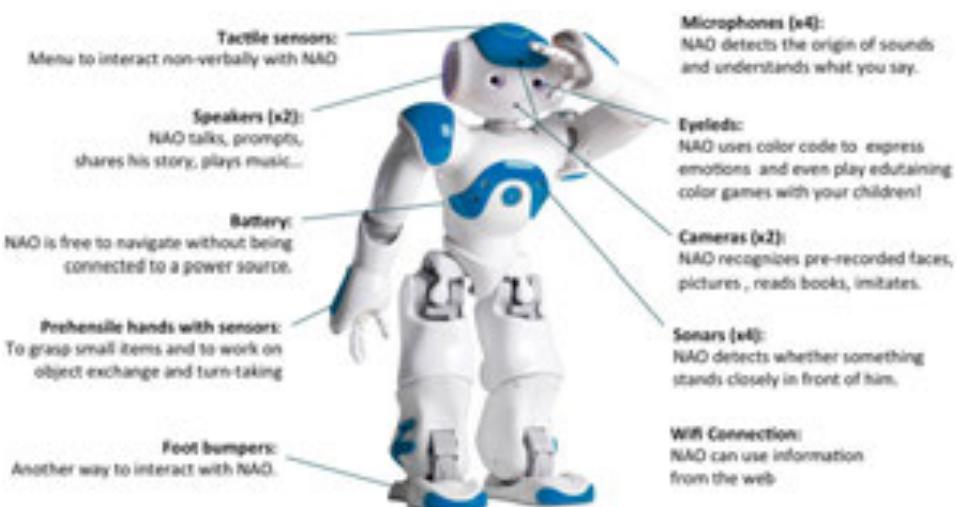
- Sensing
- Decision making
- Acting



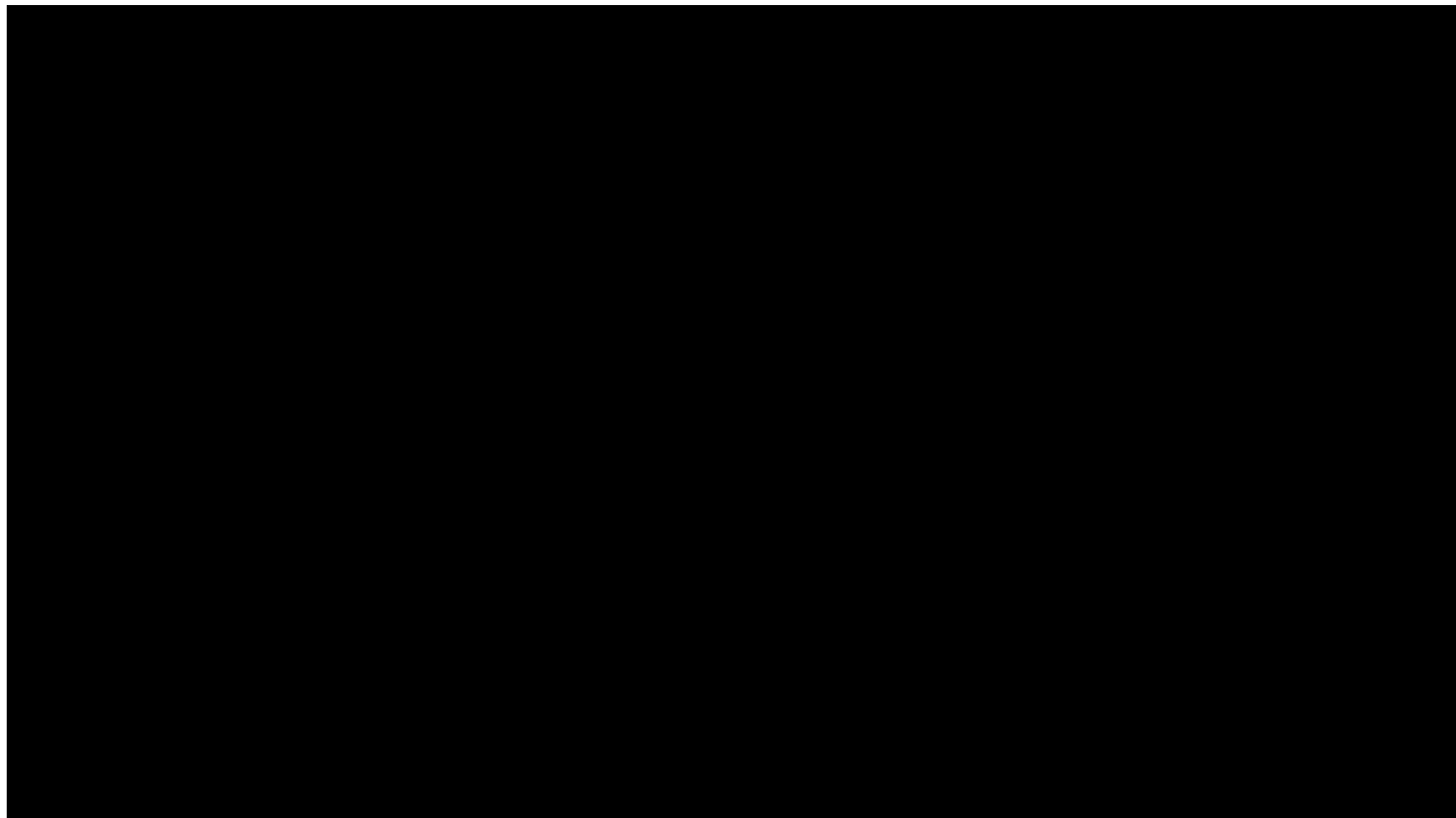
Sphero Robotics

# Sensing

- Recall the sensing you did in your assignment:
  - Array of sensors (depth, vision, ...)
  - This environment was static (nothing moving except the robot)
- Taking into account humans adds significant difficulty
  - Not just because people move...
  - ... also unpredictability, occlusions, etc
  - And: the meaning of underlying expressions, gestures, etc
- People are not good at being reliable...
- Sensors suffer from noise...

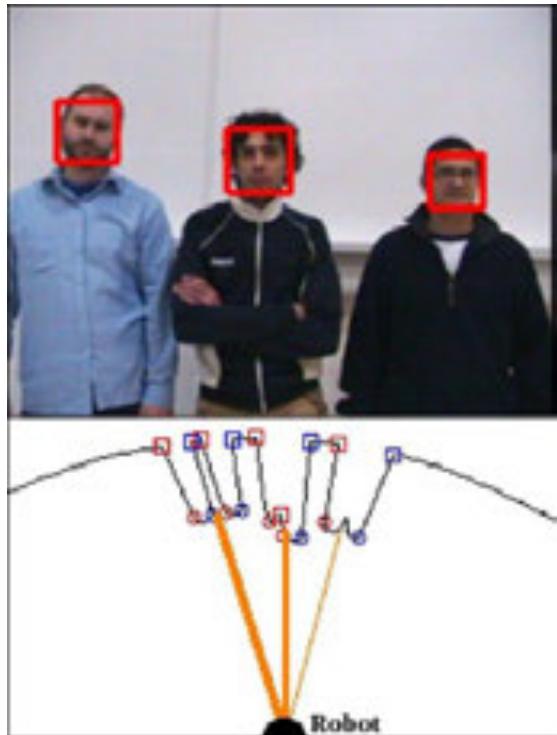


# Sensing is difficult...

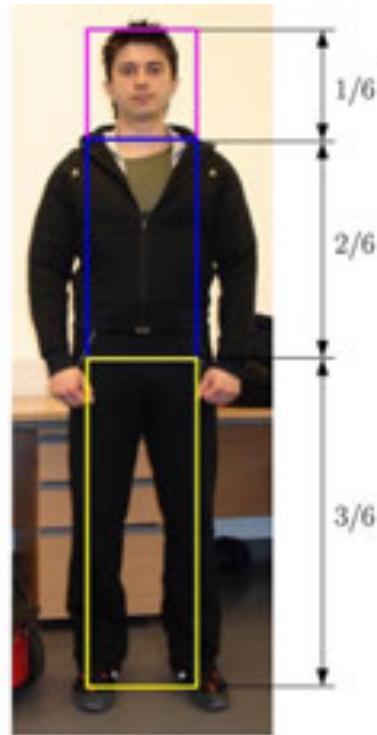


See paper and video (2012) at <http://dl.acm.org/citation.cfm?doid=2559636.2559650>

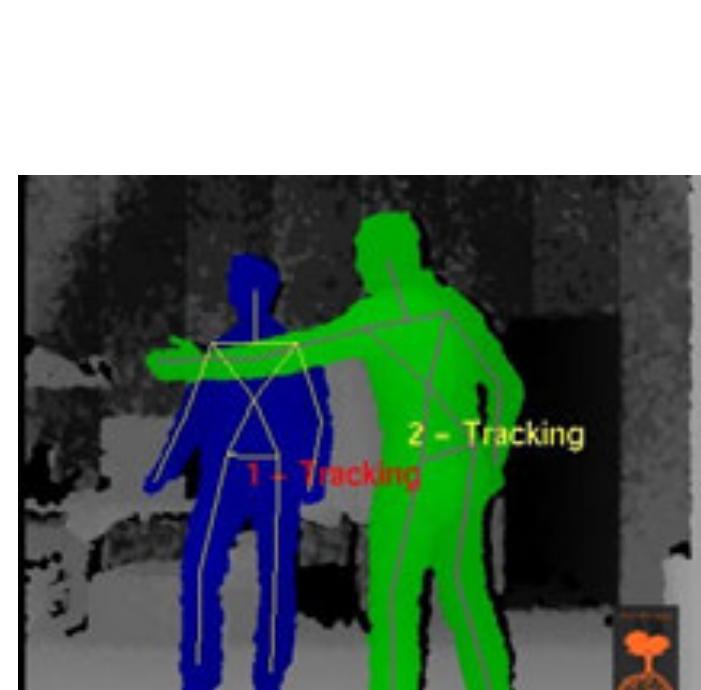
# Human Detection



Person detection from  
laser scans of legs



Person detection from  
camera-based clothing  
and face recognition



Person/skeleton  
tracking from RGB-D  
information (e.g.  
Kinect):  
MS Kinect SDK, OpenNI  
e.g. <https://structure.io/openni>

# Decision Making: what is Autonomy?

- Implicit assumption so far (see last week) that our robots are *autonomous*
- Many (very involved) definitions that are philosophically-inclined...
  - E.g. based on autopoiesis...
- Practical characterisations:
  - <http://humanrobotinteraction.org/autonomy/>
  - **The amount of time that a robot can be 'neglected' by the designer/operator**
    - High autonomy: long periods acting on its own
    - Low autonomy: no/short periods of acting alone

# Levels of Autonomy

Teleoperation



1. Wizard of Oz
  - Teleoperation
2. Scripted Robot Behaviour
  - Operator runs pre-scripted behaviours
3. Partial Autonomy
  - Hybrid autonomous/controlled system
4. Supervised Autonomous Behaviour
  - System acts autonomously, but is supervised, with human take-over in uncertainty
5. Full Autonomy
  - No human intervention required

Autonomous

# 1. Wizard of Oz

- Remote control of a robotic system, or aspects thereof
  - May be mixed with varying levels of autonomy
  - Typically used to stand in for technical aspects that are currently too difficult/unreliable/under test
- From 2012:
  - Most uses of WoZ for Natural Language Processing

*“Pay no attention to the man behind the curtain!”*

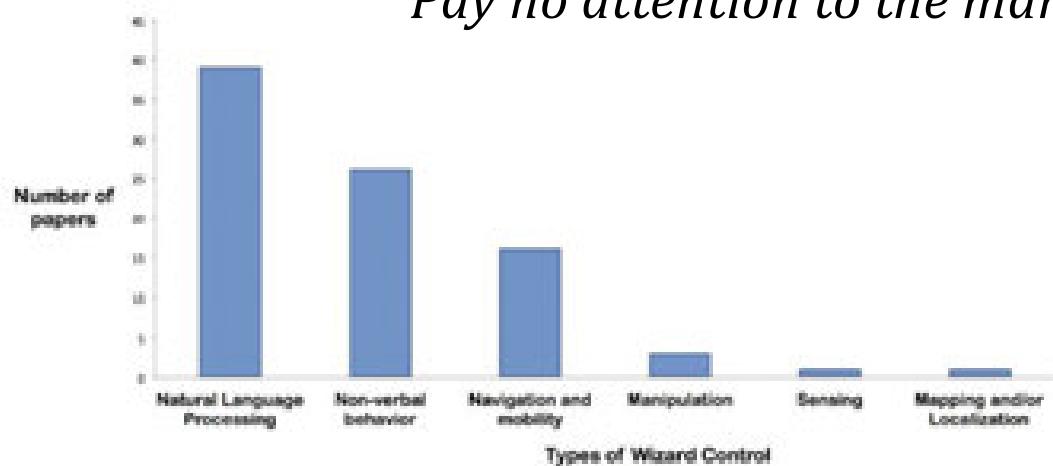
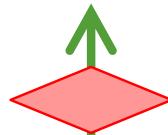


Figure 4. Chart depicting the types of Wizard control employed in the included papers. Some papers described using more than one type of control.

From Riek (2012)

# 1. Wizard of Oz

Teleoperation



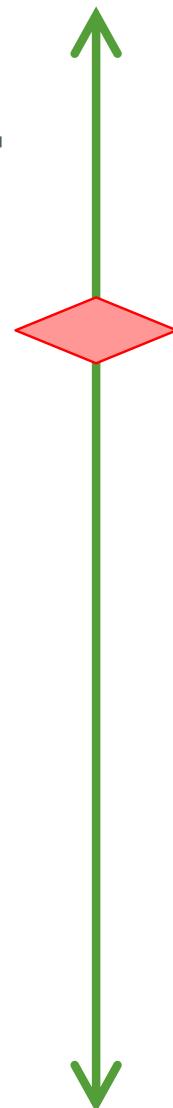
- Teleoperation
- Giving the impression of autonomy
- The Wizard is hidden from view, or not obviously associated with the control of the robot
  - Either way, the participant is unaware of the remote control.
- Complete WoZ entails full remote control of all aspects of behaviour

Autonomous



## 2. Scripted Robot Behaviour

Teleoperation

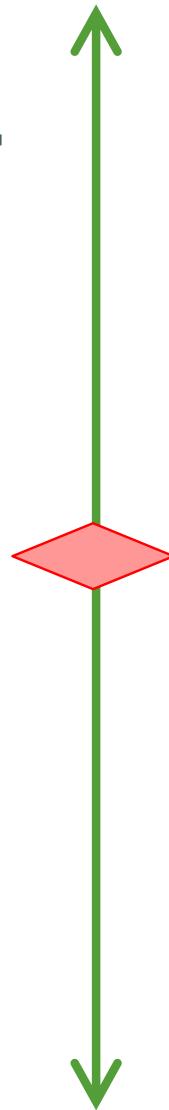


- Mediated Teleoperation
- Reducing the workload on a Wizard
- E.g. studying human behaviour when interacting with a robot

Autonomous

### 3. Partial Autonomy

Teleoperation



- Testing subsystems of a robot control system
  - Components that are ready can be run autonomously
  - Those that are not can be wizarded
- Use of shared autonomy
  - Hand-off between robot and human team-mates
  - E.g. in disaster scenarios: partially autonomous search prior to rescue

<http://www.tradr-project.eu/>



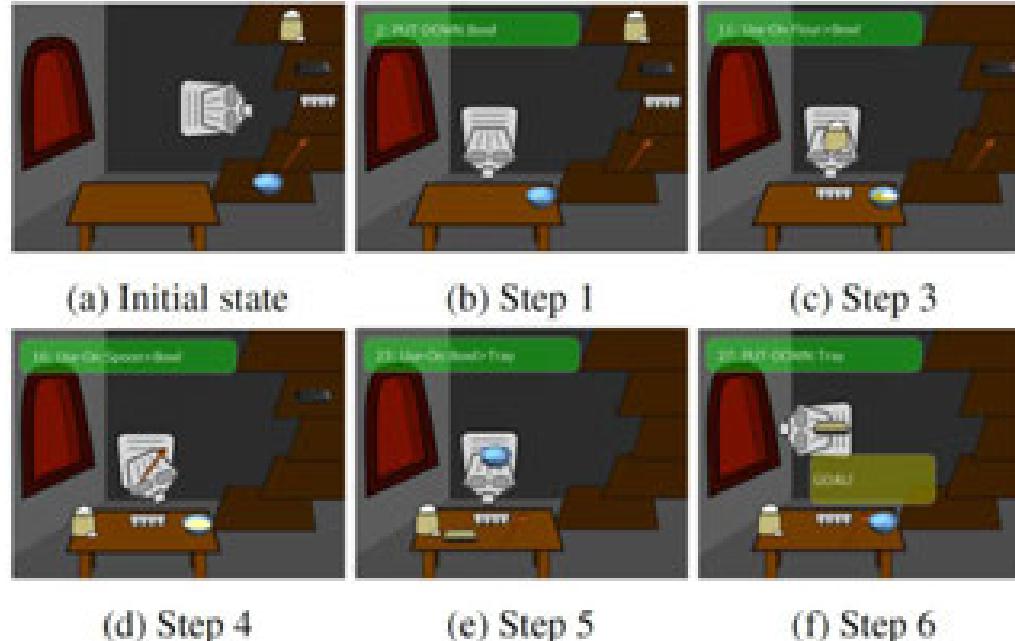
Autonomous

# 4. Supervised Autonomous Behaviour

Teleoperation

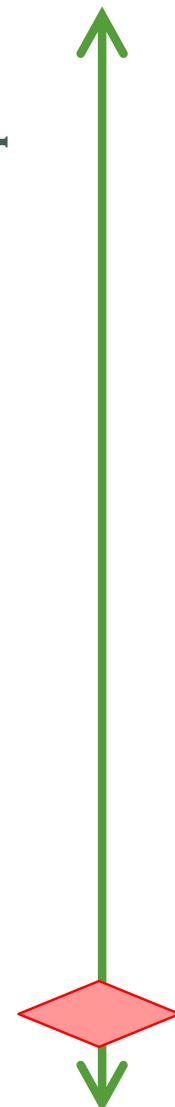
Autonomous

- Attempting to overcome noisy sensors
  - If very unreliable, then a helping hand may be needed
- Guidance for a learning robot system
  - Helping it to learn



## 5. Full Autonomy

Teleoperation



- The typical goal for robot development
  - No requirement for human remote-controller present
  - Though environment/context may require someone close by for safety...
- Full autonomy implies capacity for adaptation
  - E.g. only using predefined behaviours or waypoints may be autonomously executed, but useless if something changes (e.g. obstacles)
  - Adaptation suggests learning...

Autonomous

# Contrasting WoZ and Autonomy

## *Social Sciences Point of View*

### WoZ

- Pros:
  - Remove uncertainty
  - Focus on evaluation of interaction rather than robot
  - Full control over robot behaviour
  - Repeatable experiment setup (??)
  - Easier to implement
- Cons:
  - Human-human interaction with a robot in the middle?
  - Not consistent...

### Autonomous

- Pros:
  - Study with state-of-the-art robot instead of dummy
  - Testing system robustness
  - How to replicate human-like learning on robot
- Cons:
  - High uncertainty
  - Not necessarily repeatable
  - High maintenance
  - Can be slow...

# Contrasting WoZ and Autonomy

## *Computer Science Point of View*

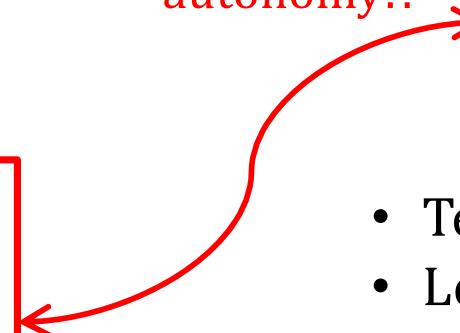
### WoZ

- Pros:
  - Remove uncertainty
  - Evaluate robot design
  - Repeatable experiment
- Cons:
  - No evaluation of:
    - Perception
    - Reasoning
    - Learning
    - World Model
    - Action selection
  - Evaluates only robot design

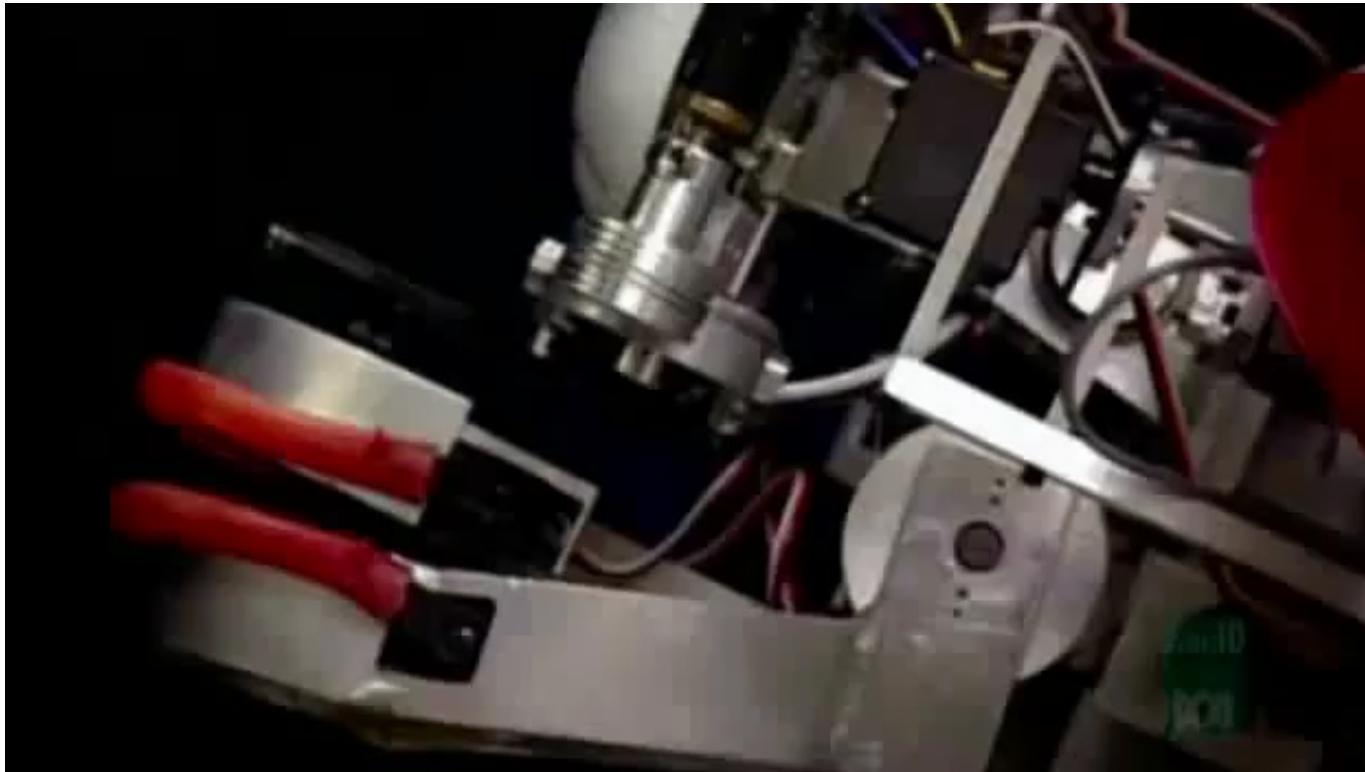
### Autonomous

- Pros:
  - Evaluation of:
    - Perception
    - Reasoning
    - Learning
    - World Model
    - Action selection
  - Testing system robustness
  - Learning from interaction
- Cons:
  - High uncertainty
  - Not necessarily repeatable
  - Can be slow...

Level of  
autonomy!?

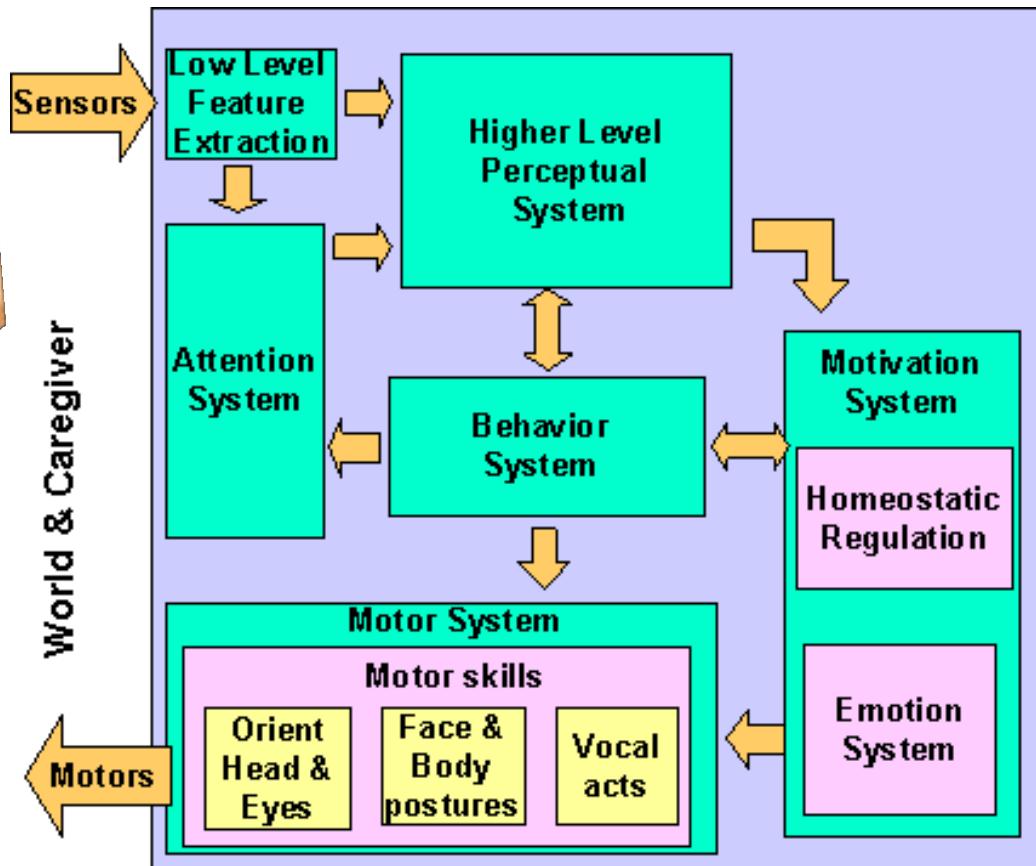
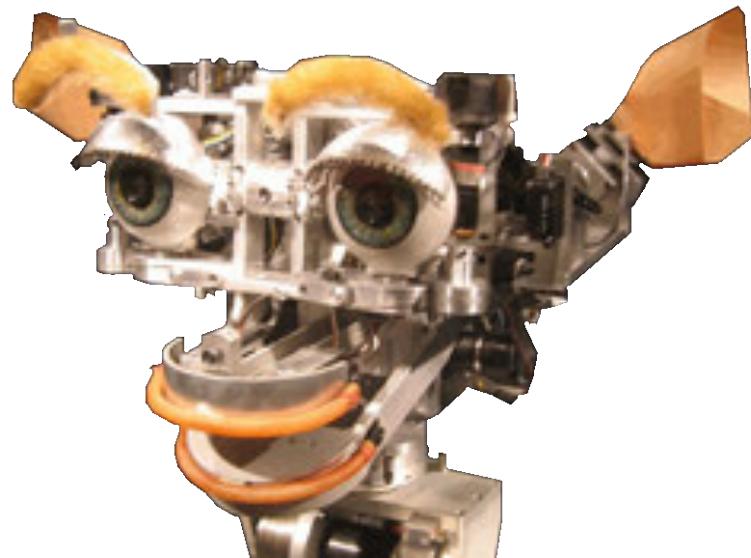


# Control Architectures: Reactive



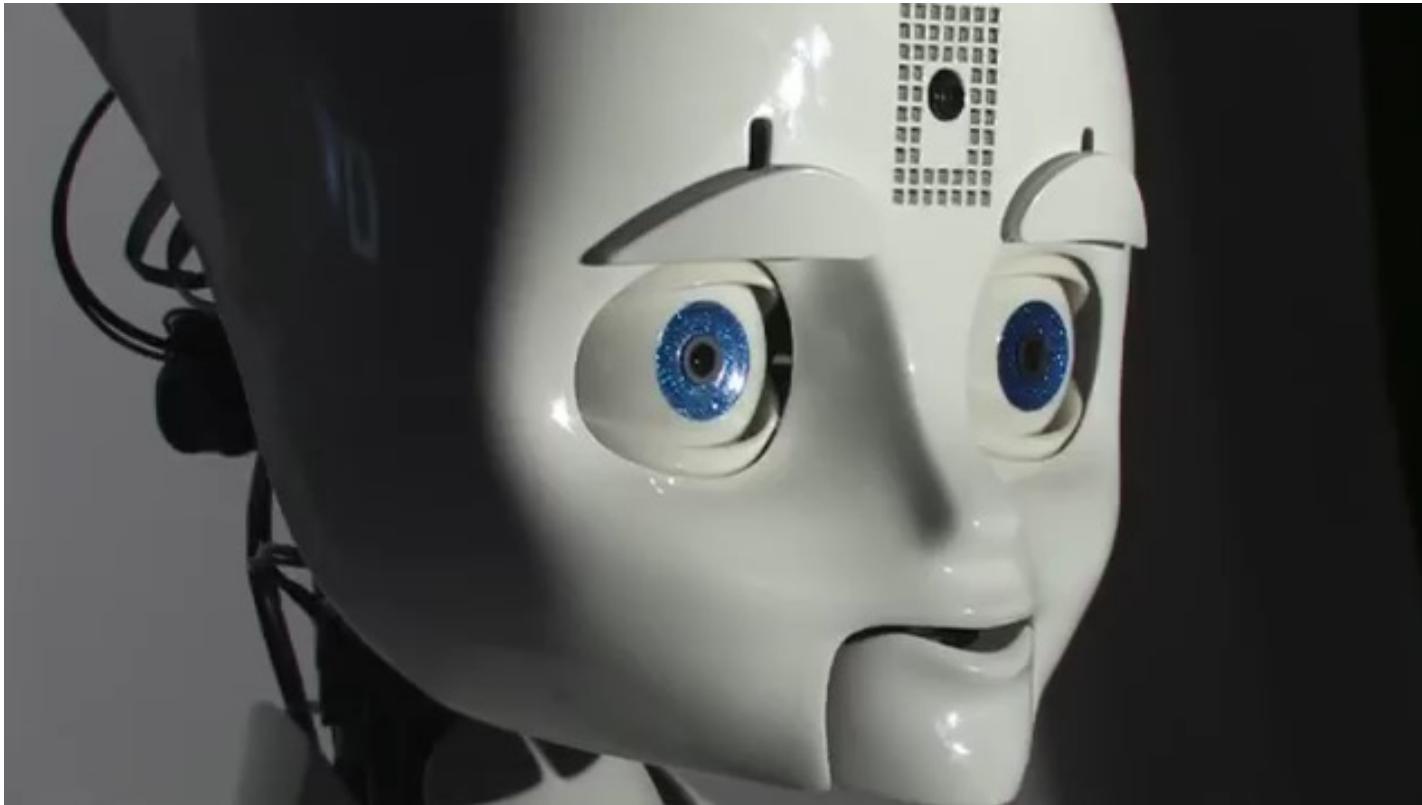
- Kismet (MIT, 1990's)
- See video at: <https://www.youtube.com/watch?v=8KRZX5KL4fA>
- Note the brief discussion of anthropomorphisation...

# Control Architectures: Reactive



- Kismet control architecture
- See <http://www.ai.mit.edu/projects/sociable/kismet.html>

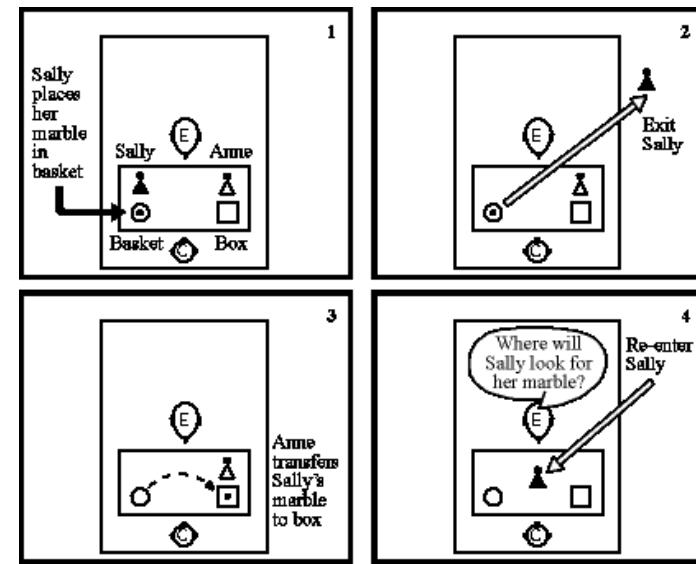
# Control Architectures: Cognitive



- MDS – (Octavia @ NRL)
- Video: <https://www.youtube.com/watch?v=aQS2zxmrrrA>
- Controlled with cognitive architecture ACT-R/E (Trafton, 2013)

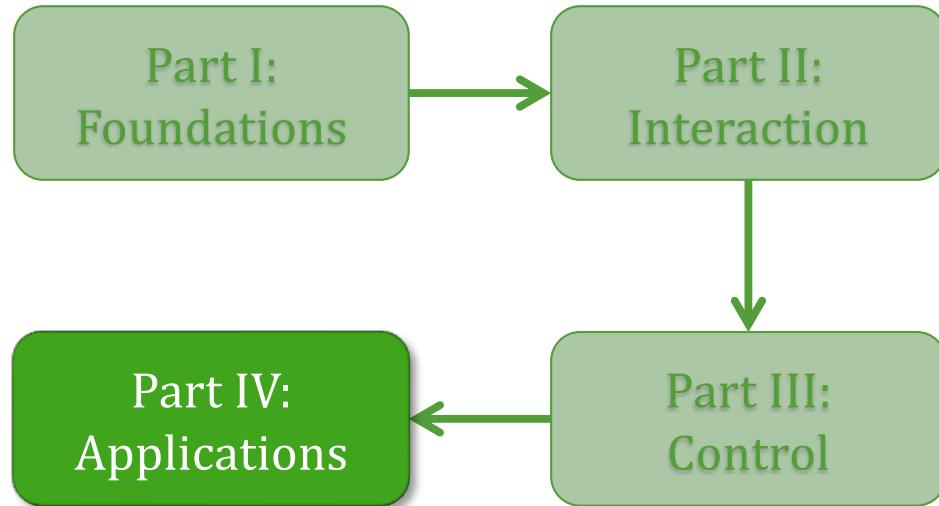
# Control Architectures: Cognitive

- Octavia case study: Theory of Mind
  - The capacity of humans to attribute mental states to others and to oneself, and to understand that others may have different states than oneself



<http://www.holah.karoo.net/baronstudy.htm>

- Further relevant concepts to CogArch:
  - Belief-Desire-Intention architectures, e.g.  
<http://www.aaai.org/Papers/ICMAS/1995/ICMAS95-042.pdf>
  - Affective Computing related to HRI, e.g.  
<http://cseweb.ucsd.edu/~lriek/papers/riek-acd-aisb09.pdf>



# Part IV: Applications

Robot roles and ethics

# Robot Roles: Learning Peer

ROBOTICS  
WITH  
PLYMOUTH  
UNIVERSITY

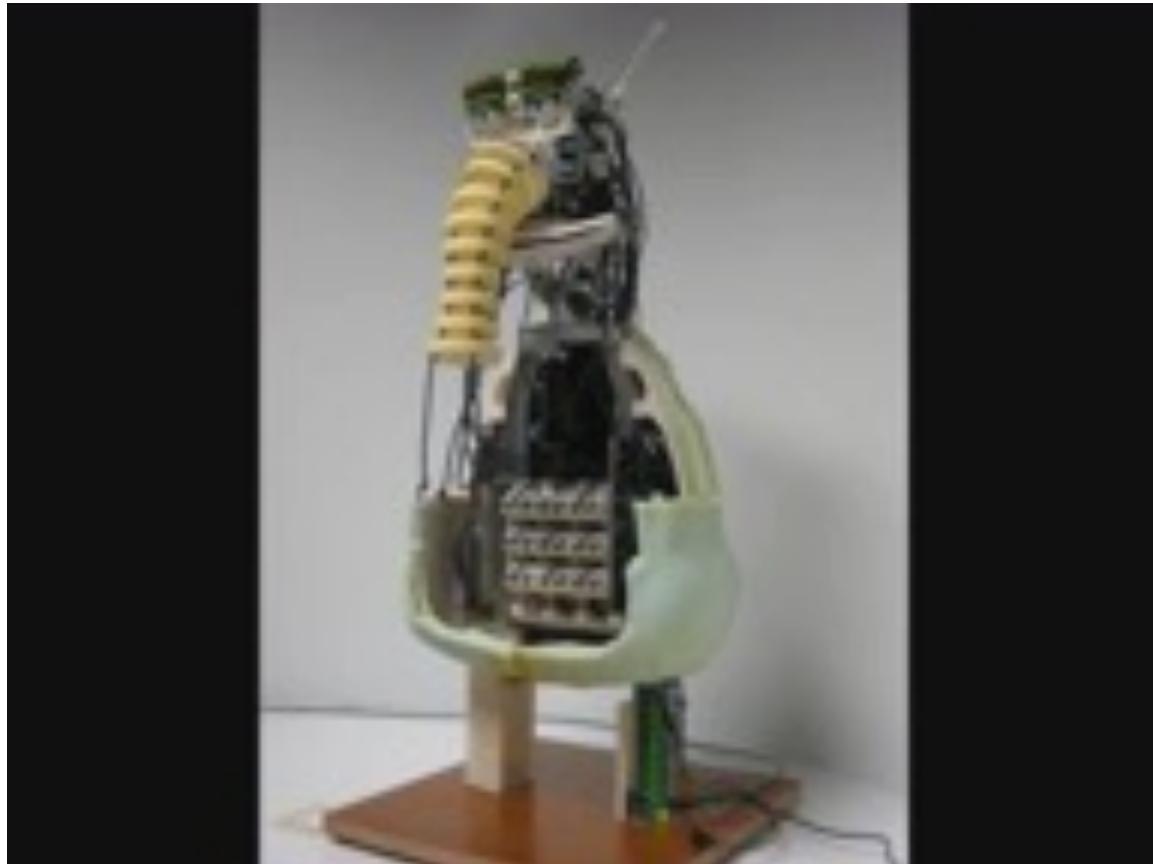


## The 'Sandtray' Mediating Social Human-Robot Interaction

Plymouth University, U.K.

- Robot as a learning partner for a child:  
<http://ieeexplore.ieee.org/abstract/document/6249477/>
- Robot and child co-located, facing each other over a touchscreen, and can both interact with the touchscreen
- Robot plays the role of a peer: not perfect performance, makes mistakes, and adapts its behaviour

# Robot Roles: Therapy Aid



- Probo robot: a green elephant-like cuddly robot
- Used in autism therapy for children, under the supervision of a therapist
- Used in Robot-Enhanced Therapy, see <http://www.dream2020.eu/>

# Other Robot Roles

Have seen:

- Robot as learning partner
- Robot as assistant
- Robot as therapy tool

Could be:

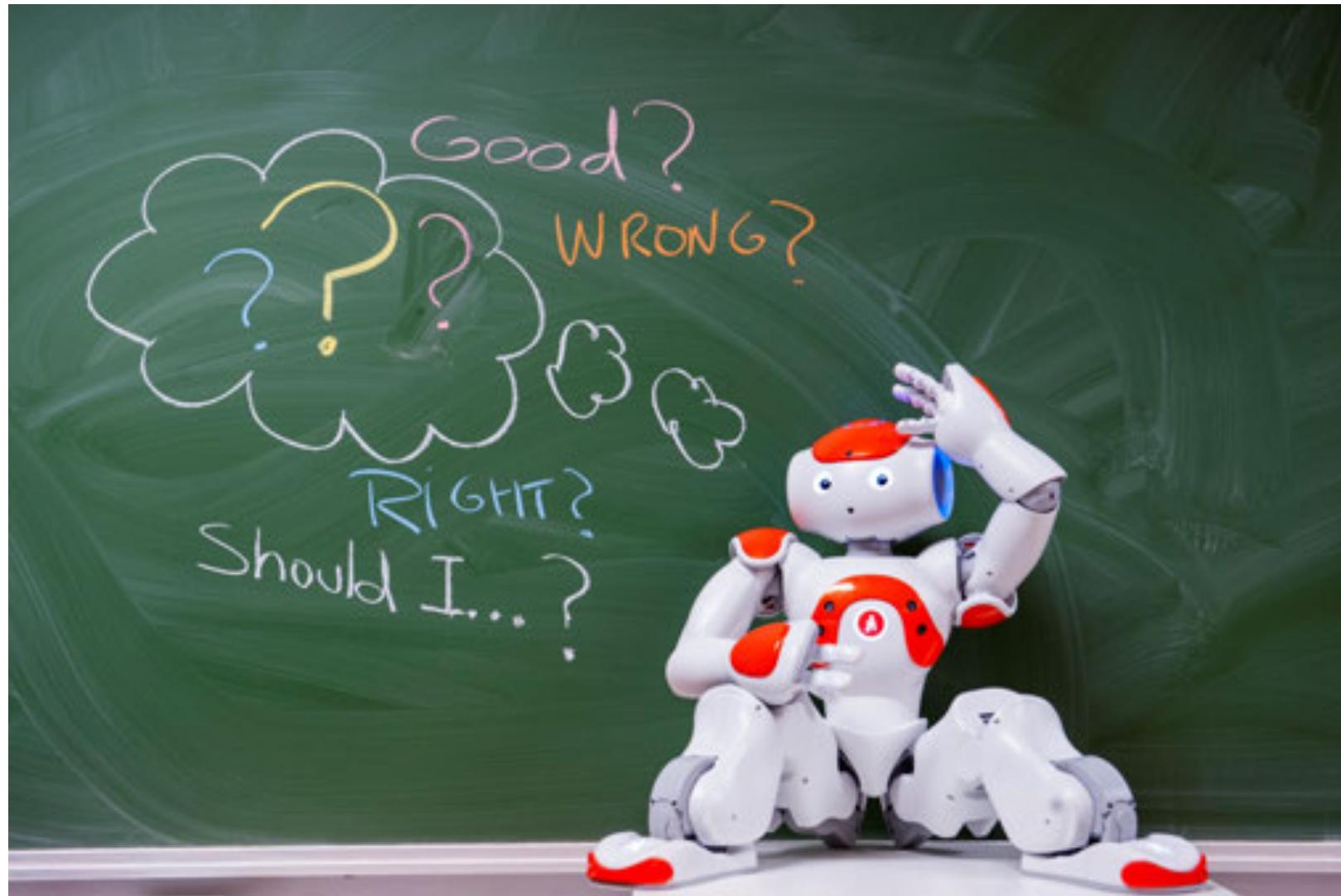
- Robot as teacher/tutor
- Robot as tool
- Robot as therapist
- Robot as team member

**(Autonomous) Tool**

**Social Agent**

- Above, level, below in social hierarchy

# Ethical Issues

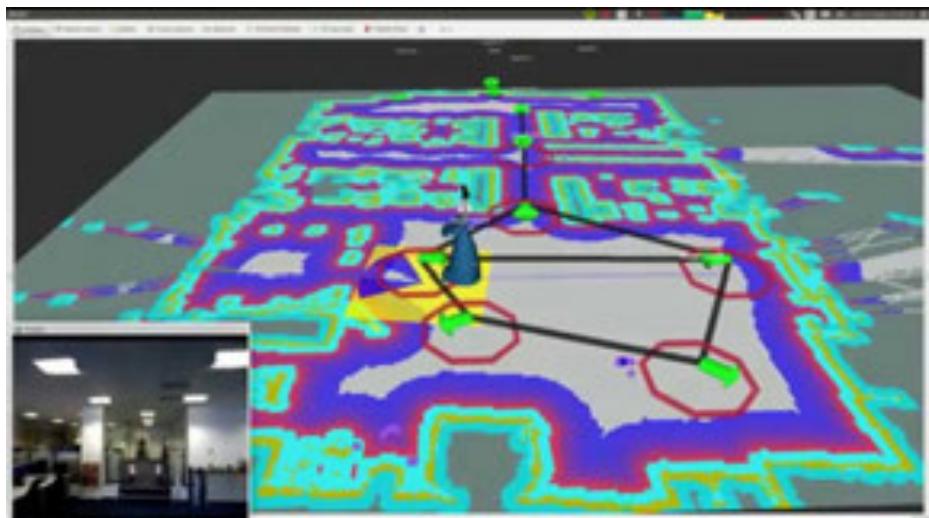


# Ethical Issues

- Is it right to use robots as social agents in this way?
  - Issue of deception?
  - Are we replacing social contact with other humans if we use these devices?
  - In the case of child therapy, are the problems made worse (e.g. getting used to interacting with robot rather than people)?
  - Attachment to robots rather than humans...
- Technical/Legal concerns:
  - Data protection, and the role of data recording/capture
  - Memory of prior interactions and privacy of this information

# Next Week...

- Engagement in HRI
- Navigation in HRI
  - Human-aware planning
- Methods for HRI Evaluation



# References / Reading

- Baxter, P. et al., 2016. From Characterising Three Years of HRI to Methodology and Reporting Recommendations. In *HRI 2016*. Christchurch, New Zealand: ACM Press, pp. 391–398.
- Duffy, B.R., 2003. Anthropomorphism and the social robot. *Robotics and Autonomous Systems*, 42(3–4), pp.177–190.
- Heider, F., Simmel, M., 1944. An experimental study of apparent behavior. *The American Journal of Psychology*, Vol 57, 243-259
- Mathur, M.B. & Reichling, D.B., 2016. Navigating a social world with robot partners: A quantitative cartography of the Uncanny Valley. *Cognition*, 146, pp.22–32.
- Meltzoff, A.N. et al., 2010. “Social” robots are psychological agents for infants: a test of gaze following. *Neural networks : the official journal of the International Neural Network Society*, 23(8–9), pp.966–72.
- Moore, R.K., 2012. A Bayesian explanation of the “Uncanny Valley” effect and related psychological phenomena. *Scientific Reports*, 2, p.864.
- Riek, L., 2012. Wizard of Oz Studies in HRI: A Systematic Review and New Reporting Guidelines. *Journal of Human-Robot Interaction*, 1(1), pp.119–136.
- Senft, E., Baxter, P., Kennedy, J., Lemaignan, S., & Belpaeme, T., 2017. Supervised Autonomy for Online Learning in Human-Robot Interaction. *Pattern Recognition Letters*, 99, p77–86.
- Trafton, J.G. et al., 2013. ACT-R/E : An Embodied Cognitive Architecture for Human-Robot Interaction. *Journal of Human-Robot Interaction*, 2(1), pp.30–54.

**CMP3101M AMR – Week 12**

# **Human-Robot Interaction**

## **part 2**

Prof Marc Hanheide & Dr Paul Baxter

Location: Online

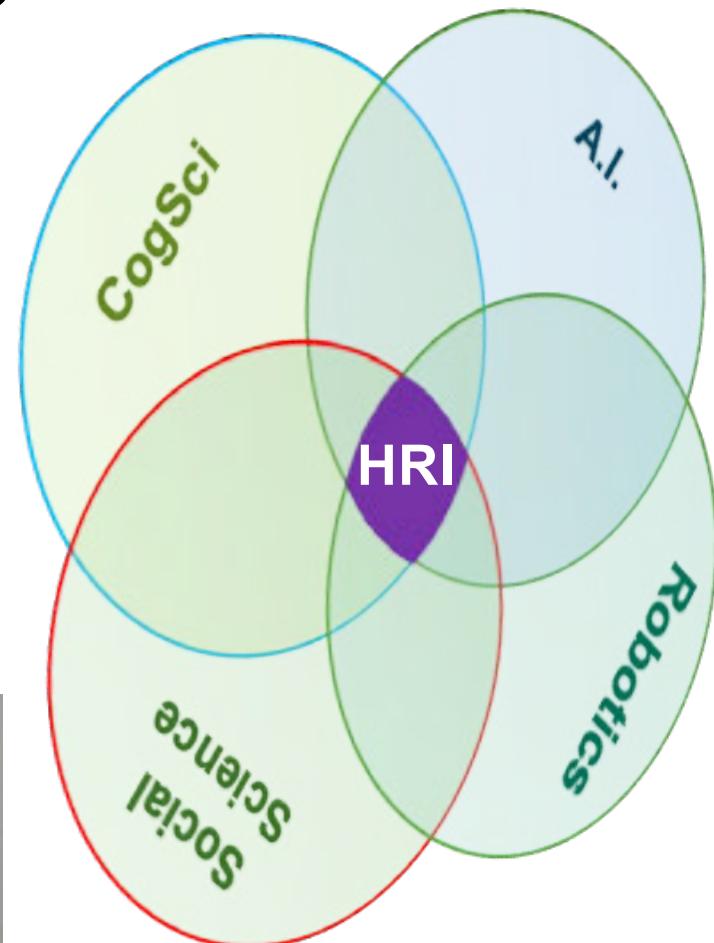
[mhanheide@lincoln.ac.uk](mailto:mhanheide@lincoln.ac.uk)

# Week 11: Intro to HRI

- HRI is “*dedicated to understanding, designing, and evaluating robotic systems for use by or with humans*”
- Anthropomorphism as an important factor:
 

Is the “tendency to attribute human characteristics to inanimate objects, animals and others with a view to helping us rationalise their actions”

(Duffy, 2003)
- The Uncanny Valley



From (Baxter et al, 2016)

# Modes of Interaction

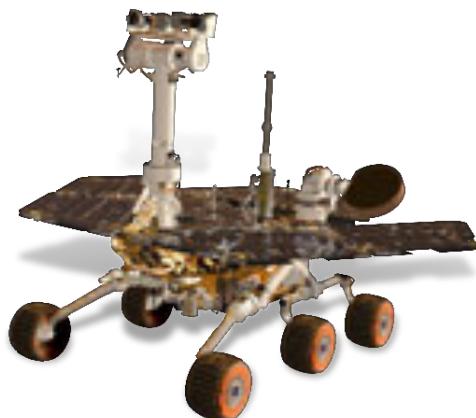
## Proximate

- The human and the robot are in the same space
- Physical and social interactions fall in this category
  - E.g. service robots



## Remote

- The human and the robot are in different locations
  - Maybe even temporally removed
- E.g. teleoperation, supervised control, ...



# Two types of Interaction

## Explicit

- An action performed with the purpose of eliciting a reaction
- Dialogue: e.g. asking a question
- Manipulation: e.g. handing over an object, or pointing

## Implicit

- Actions are modified due to presence of an other, but no reaction is expected
- Navigation: e.g. avoiding humans in the way

Capacity for overlap between Explicit and Implicit: e.g. avoiding humans, but engaging in some strategies to encourage humans to move out of the way (more today!)

# Levels of Autonomy

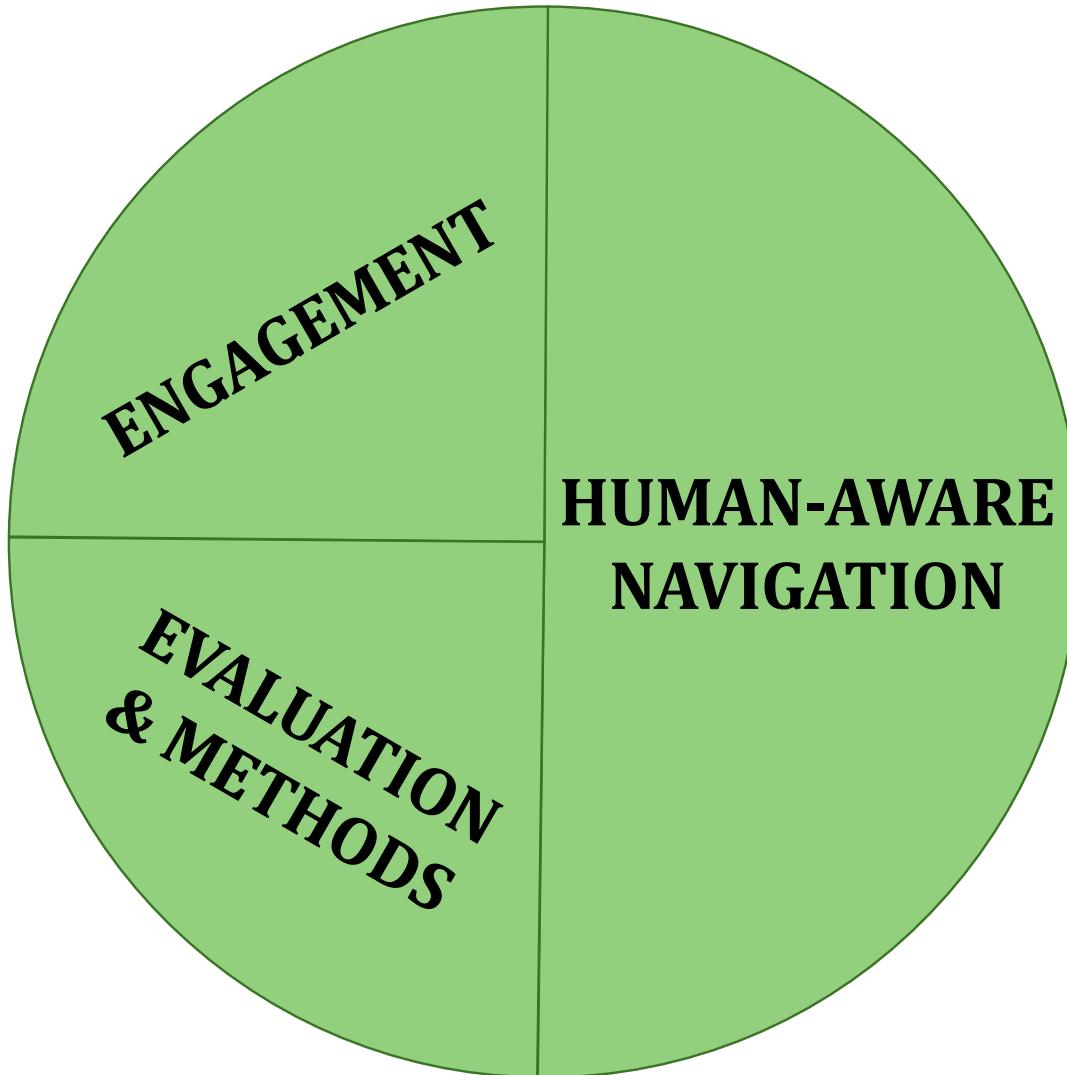
- 
- Autonomous
- Teleoperation
1. Wizard of Oz
    - Teleoperation
  2. Scripted Robot Behaviour
    - Operator runs pre-scripted behaviours
  3. Partial Autonomy
    - Hybrid autonomous/controlled system
  4. Supervised Autonomous Behaviour
    - System acts autonomously, but is supervised, with human take-over in uncertainty
  5. Full Autonomy
    - No human intervention required

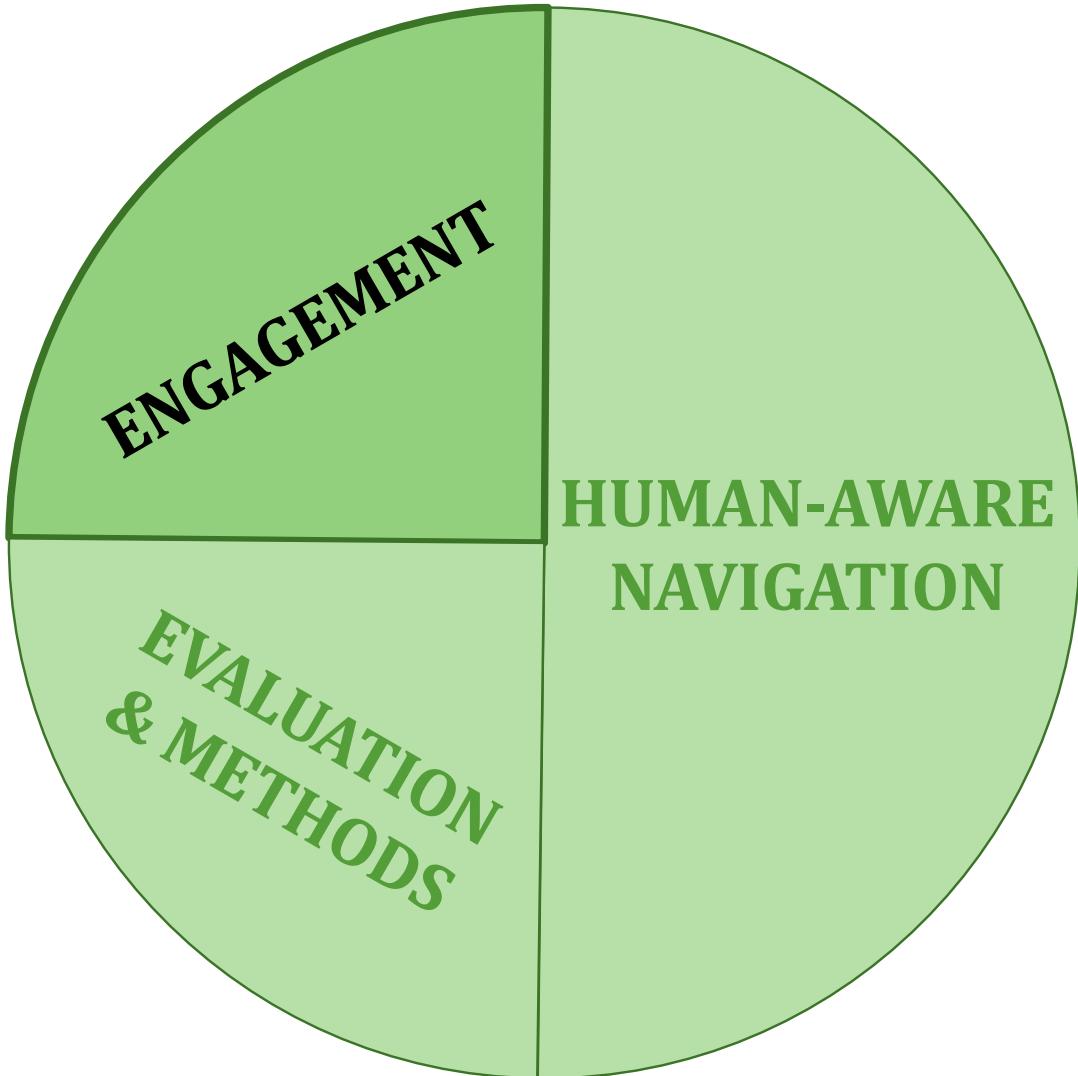
# Robot Roles and Ethics

- Robot could take on a number of roles in an interaction, both social and non-social:
  - Learning peer / tutor
  - Therapy assistant / tool
  - Remote presence
  - Team member
  - ...
- Ethical issues can arise in each of these:
  - Deception
  - Attachment (particularly, but not just, with children)
  - There are consequences for technical implementation: e.g. data protection and deletion



# Today



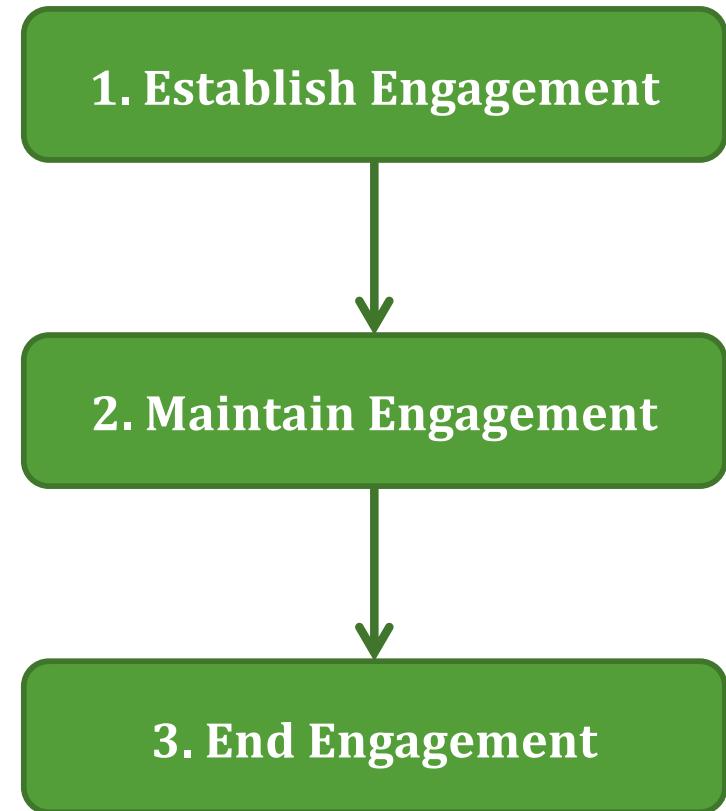


# What is engagement?

- Many different definitions...  
See e.g. Glas & Pelachaud, 2015 for an overview of these
- Sidner et al 2005:  
*“...the process by which two (or more) participants establish, maintain and end their perceived connection during interactions they jointly undertake.”*
- It encompasses all types of behaviour:
  - Implicit/Explicit
  - Verbal/Nonverbal



# Stages in Engagement



# 1. Establish Engagement

- Attracting attention
- Drawing into an (ongoing?) interaction



# 1. Establish Engagement

## Social Strategies

- Speech
  - Saying something to someone
- Gesture
  - Only useful if have limbs...
  - Waving, etc
- Behaviour
  - Could engage in attention seeking behaviour
- Physical Interaction
  - Equivalent of a tap on the shoulder... (clearly safety implications)

## Non-Social Strategies

- Sound
  - Pre-recorded speech
  - Alarm / beeps / motor noise
  - White noise
- Vision
  - Flashing lights
- Behaviour
  - Bumping into human (not advisable...)



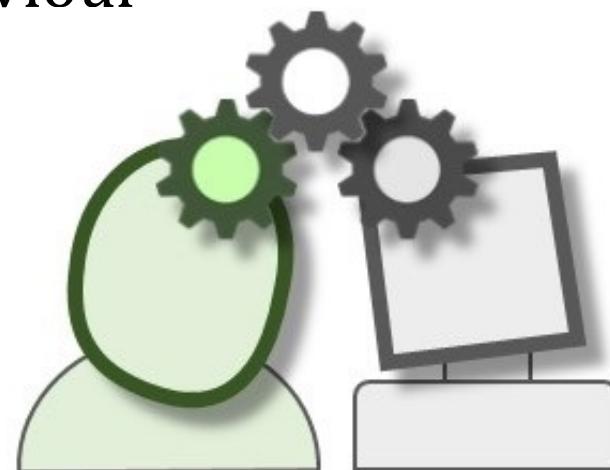
Video from:  
<https://www.youtube.com/watch?v=asHaWo04mIo>

## 2. Maintain Engagement

- Recall that we are at least partially relying on:
  - Anthropomorphism: appearance and behaviour
  - Attribution of agency

*This can only get you so far!*

- The necessity for going beyond this with “deeper” complexity and adaptivity of behaviour
  - Refer to Control Architectures Lecture, Week 9
  - This also underlies part of the motivation for incorporating Cognitive Architectures into HRI systems



See: <https://sites.google.com/site/cogarch4socialhri2016/>

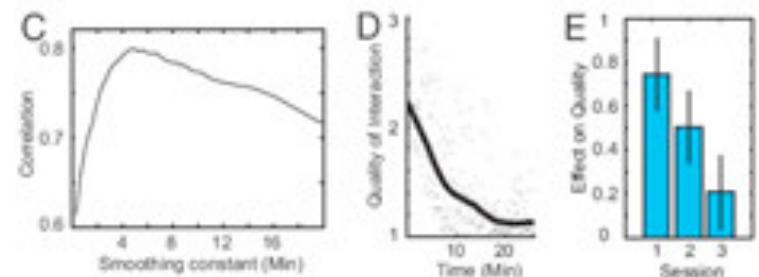
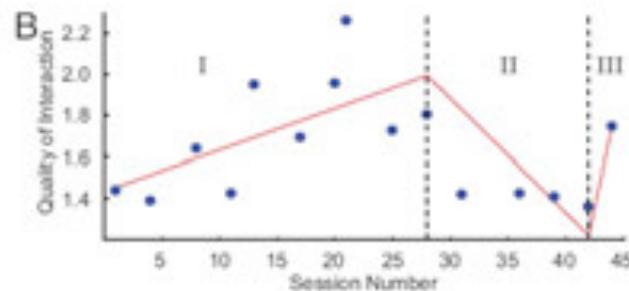
### 3. End Engagement

- Ending the interaction, and hence any engagement in the interaction
  - Possibly to be resumed at a later time
- Task related:
  - Joint task has completed successfully/unsuccessfully
  - Task no longer relevant
- Personal related
  - Illusion of agency has gone
  - Boredom!



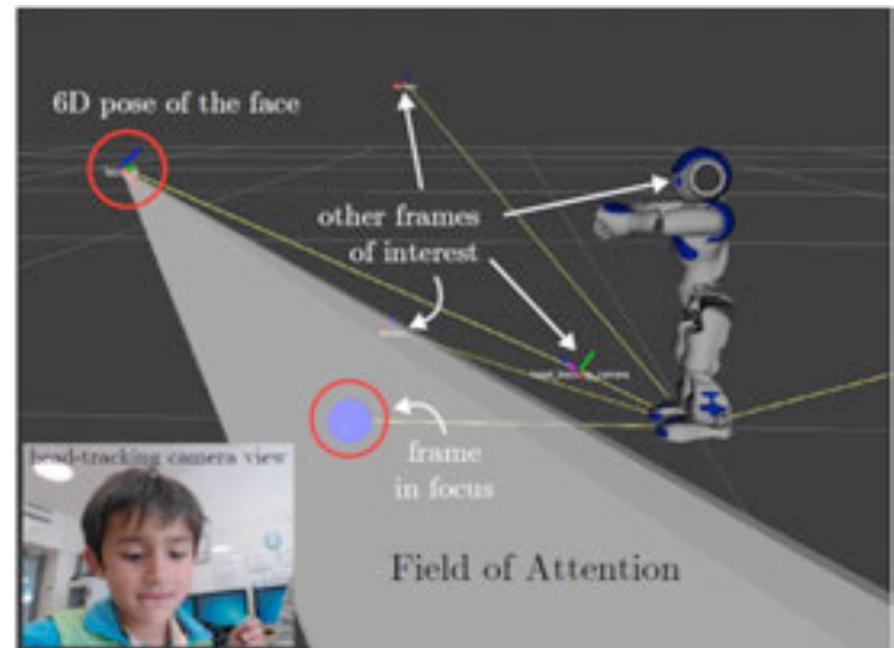
# How to detect Engagement?

- How to tell whether someone is engaged in an interaction?
  - And, ideally, how to do so automatically?
  - What should you examine?
- This is a difficult task!
- Humans have an intuitive sense of engagement, based on extensive experience:
  - Developed from baby onwards
  - Can take advantage of this
  - E.g. Tanaka et al (2007): rating using a 'slider'



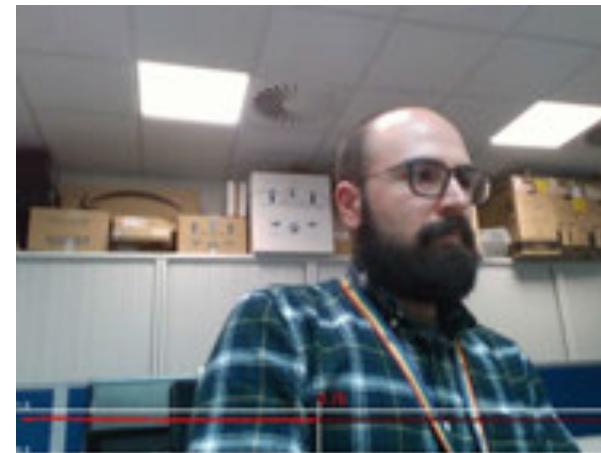
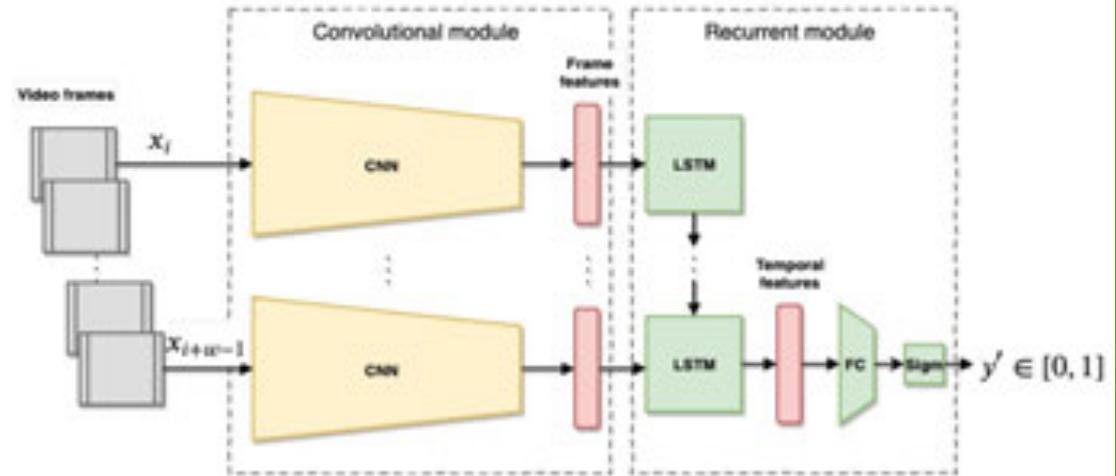
# How to detect Engagement?

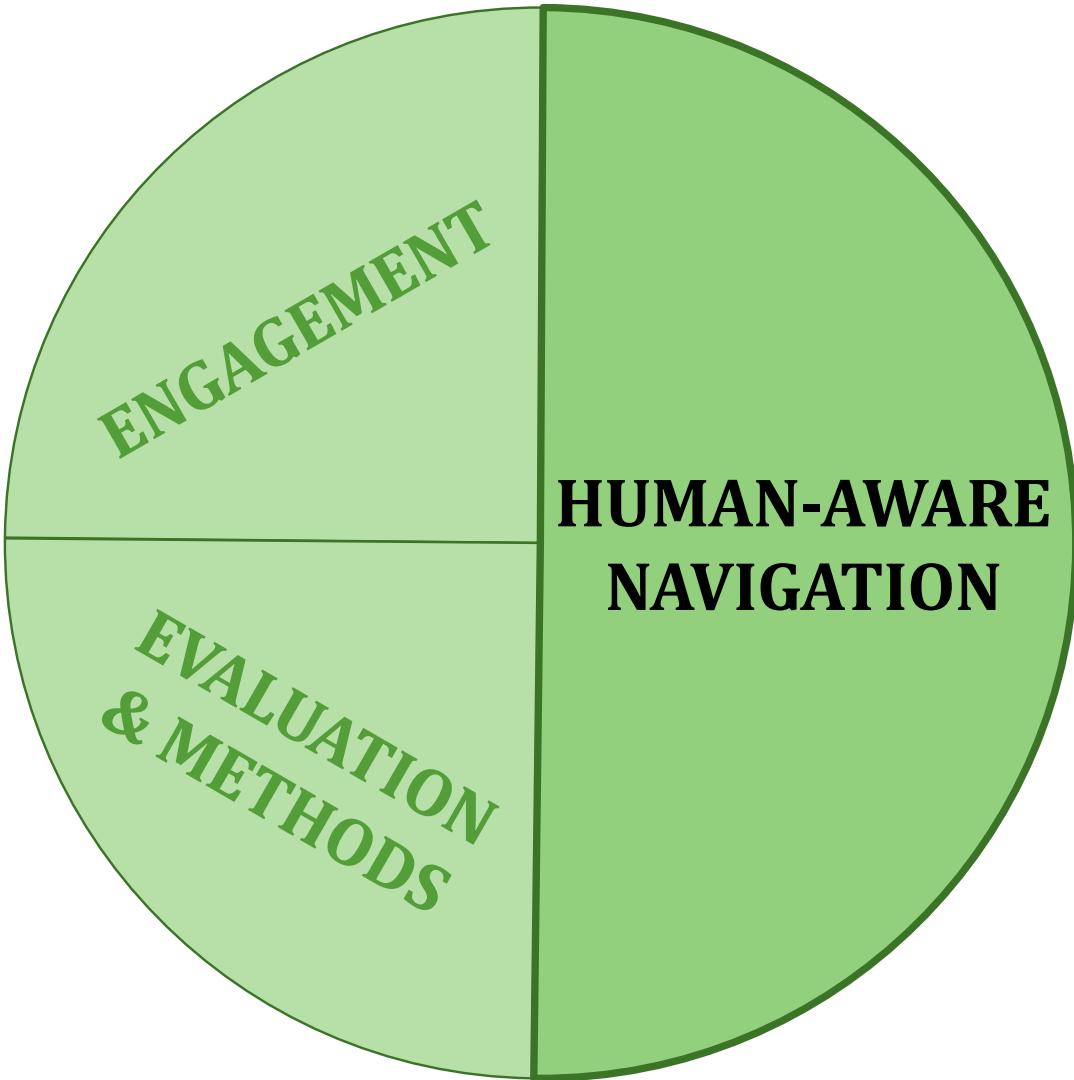
- Gaze is often used as an indicator of engagement
  - E.g. Baxter et al, 2014
- The problem is that this is often post hoc analysis
  - I.e. not in real-time, but only afterwards
- Recent developments trying to achieve this in real-time, using robot sensors:
  - E.g. the GAZR gaze estimator, which can be used for an estimation of engagement (Lemaignan et al, 2016)
  - ROS package:  
<https://github.com/severin-lemaignan/gazr>



# How to detect Engagement?

- Rather than using a model (gaze), use machine (deep) learning
  - E.g. del Duchetto et al, 2020
- Engagement is modelled as a continuous value between 0 and 1





# Will I bother here?

## - A robot anticipating its influence on pedestrian walking comfort -

“Will I bother here? - A robot anticipating its influence on pedestrian walking comfort”, HRI 2013, Tokyo, Japan

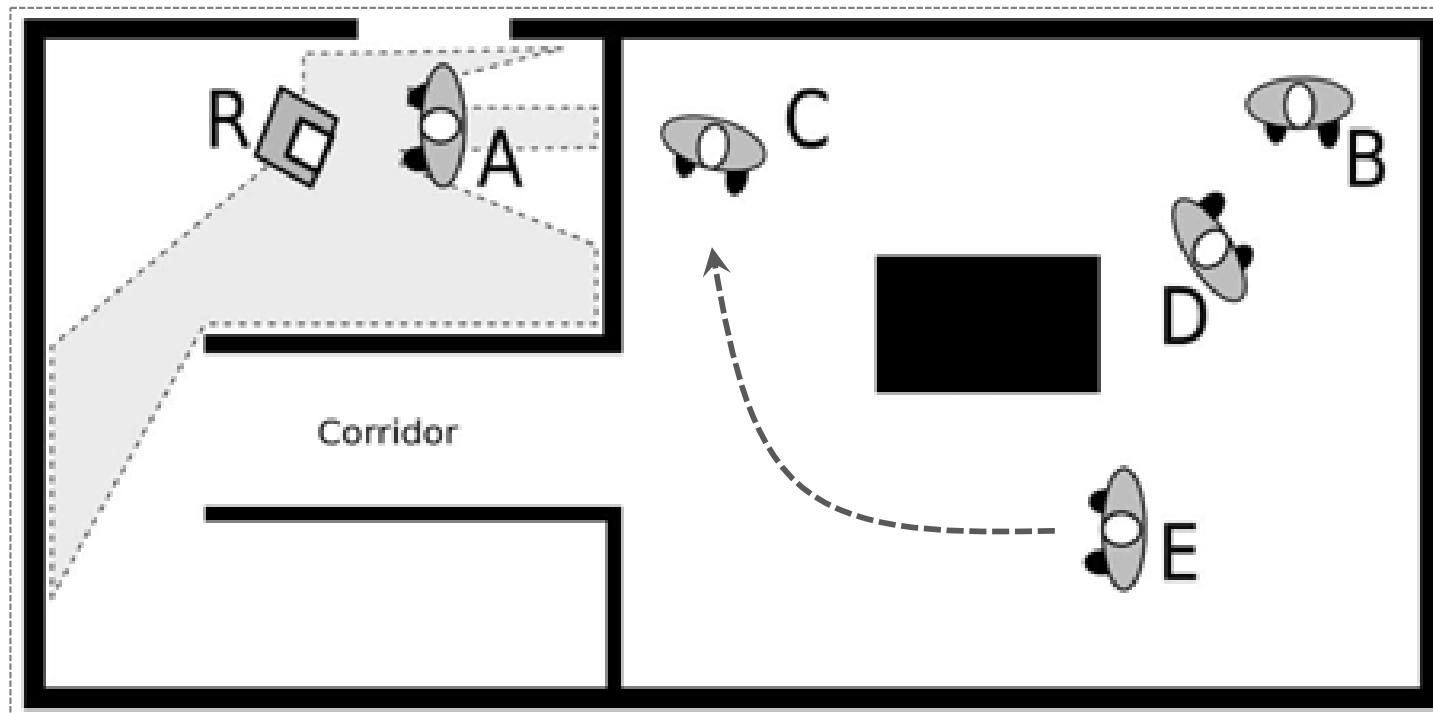
Paper: <http://ieeexplore.ieee.org/abstract/document/6483597/>

# Human-Aware Navigation

- Autonomy for mobile robots requires navigation capabilities
- Obstacle avoidance clearly required
  - Preventing robot damage
  - Human safety!
- Goals (Kruse et al, 2013):
  1. Comfort: absence of annoyance and stress for humans
  2. Naturalness: similarity of robot behaviour to humans
  3. Sociability: adherence to high-level cultural constraints
- May be necessary to reduce efficiency (in terms of speed/distance to goal) in the service of these goals

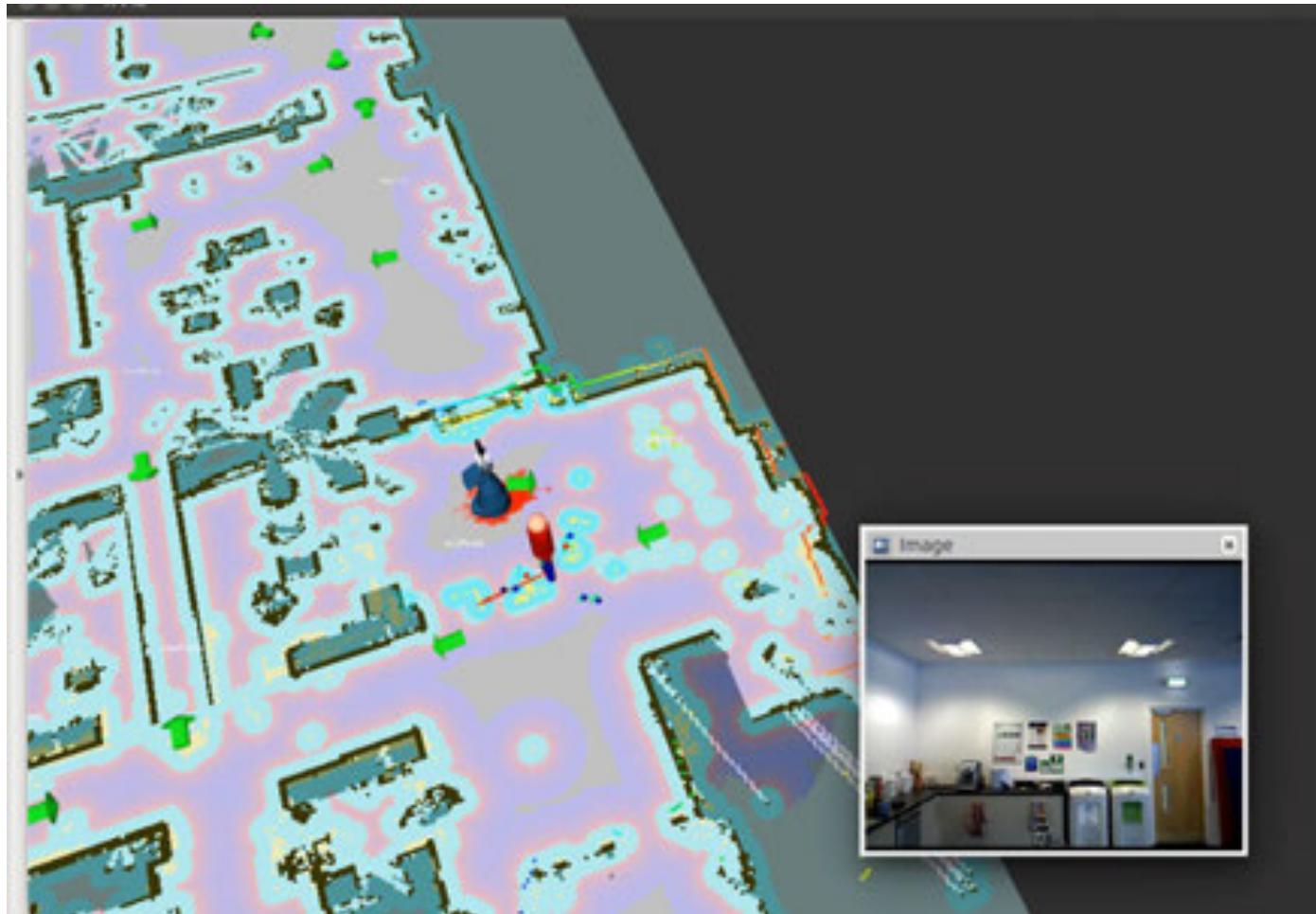
# An example

From Kruse et al, 2013:



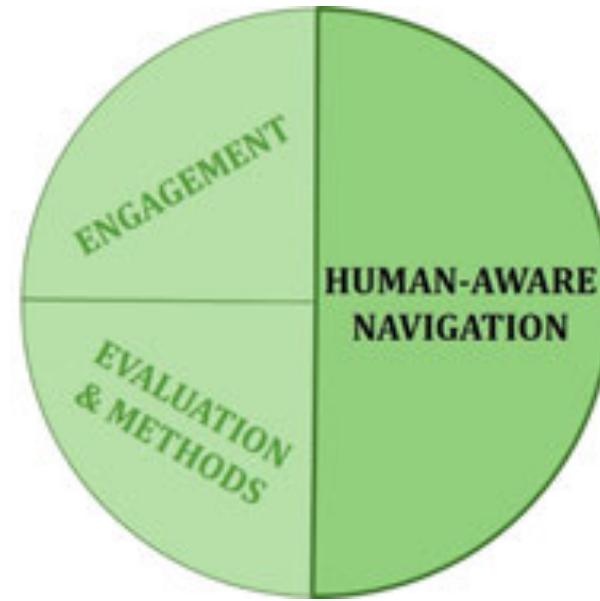
How should the robot guide person A to person B?

# *Humans are Awkward Obstacles...*



Also see this L-CAS video:

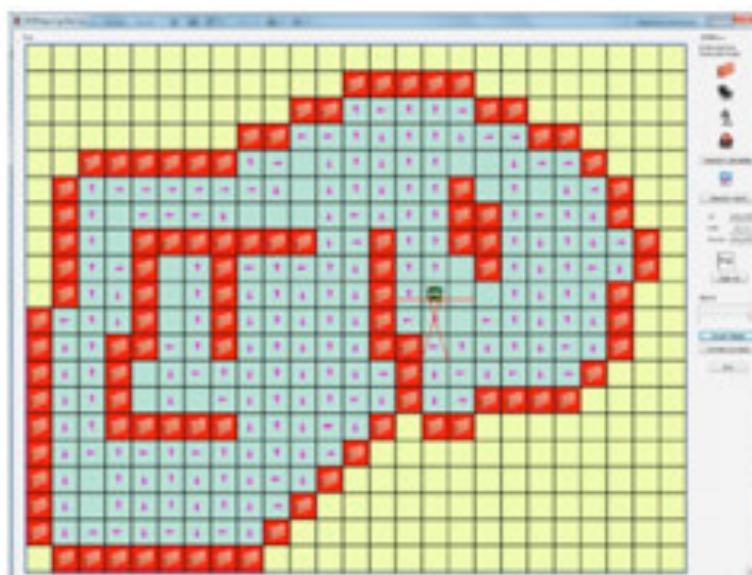
<https://www.youtube.com/watch?v=zdnvhQU1YNo>



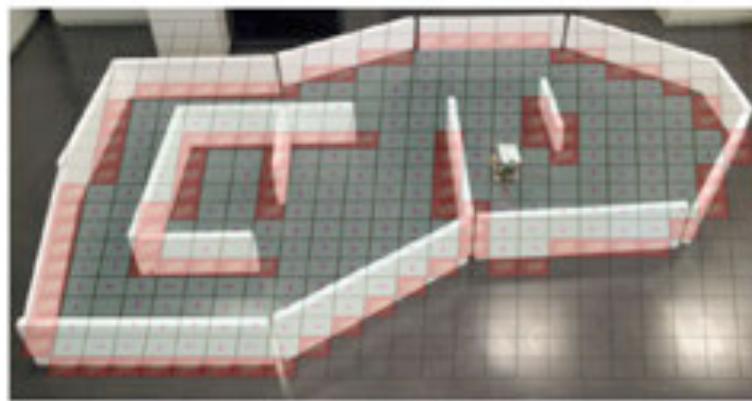
## (a) Navigation and Costmaps

# Recap: path planning

- Refresh your memory of Lecture Week 7: Navigation 1
  - Dijkstra and A\*
  - Include movement cost into node
- Application to discrete states, in this example a network of nodes
- Equally applies to a 2D space discretised in a grid
  - “Occupancy” grid
  - Image from (Gonzalez et al, 2013)



(a)

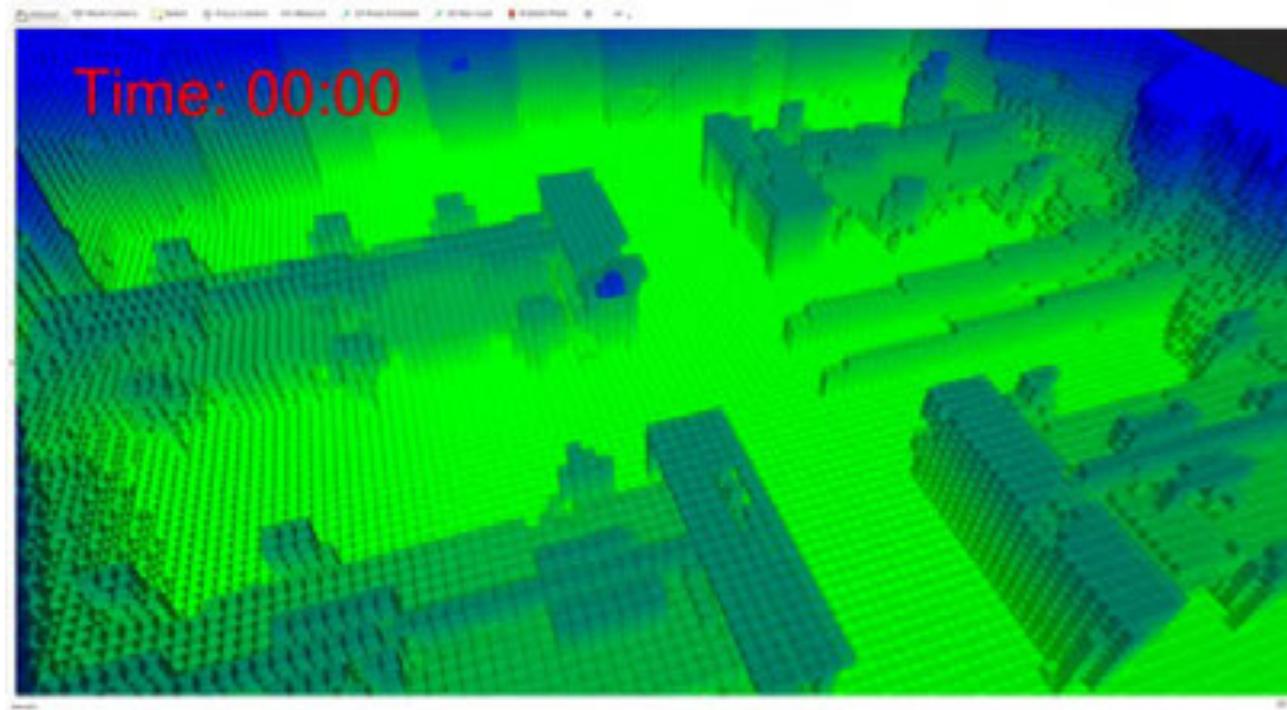


(b)

See <http://qiao.github.io/PathFinding.js/visual/>

# *An aside: 3D discretisation (voxels)*

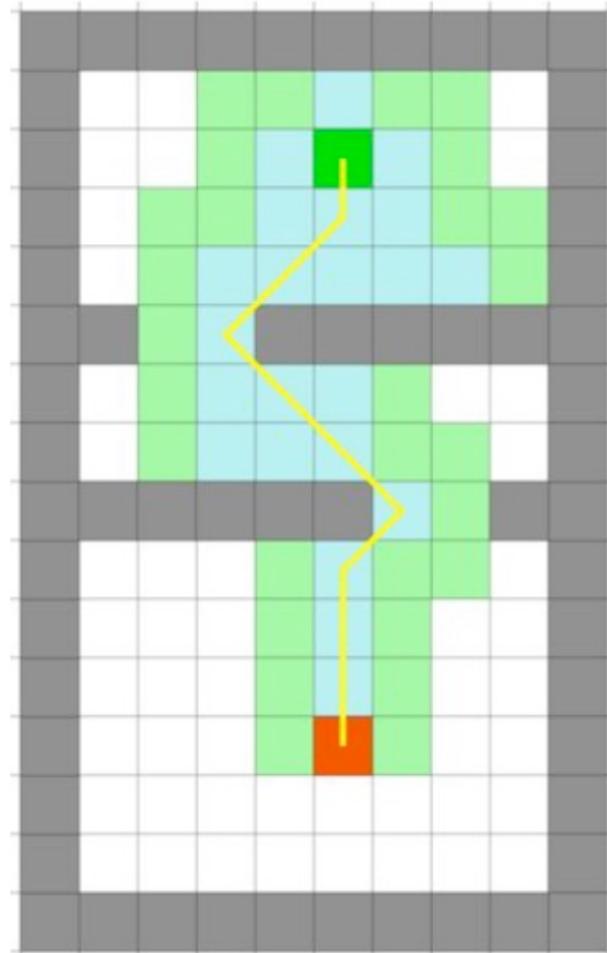
- Regular grid in 3D space
- Can use to discretise a 3D space in a similar way as done for 2D spaces



Video: [https://www.youtube.com/watch?v=CUT3t-ico5Y&list=PLnS6TQ\\_QsDUXCrv1fbzPHn0LoTwcn8NF&index=2](https://www.youtube.com/watch?v=CUT3t-ico5Y&list=PLnS6TQ_QsDUXCrv1fbzPHn0LoTwcn8NF&index=2)

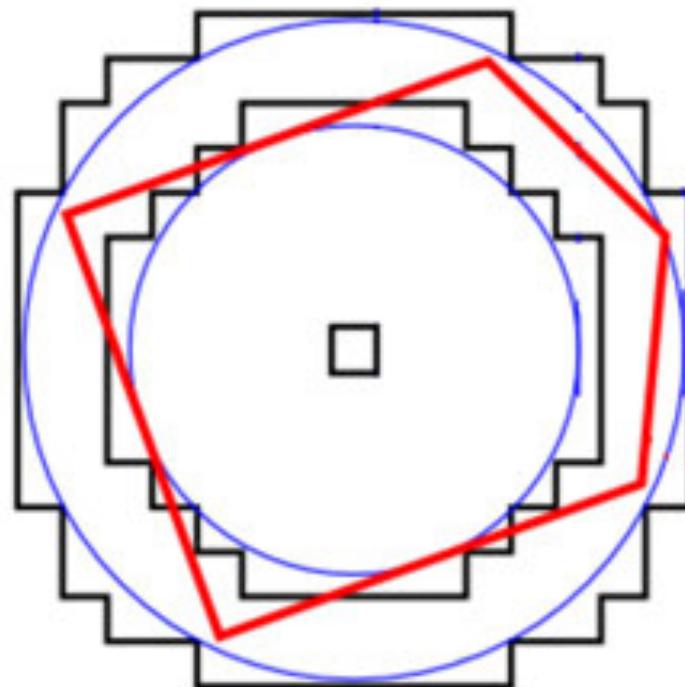
# Path Planning

- Have what we need to plan path
  - E.g. Dijkstra
- However, Dijkstra not ideal:
  - Path close to obstacles
  - Next to walls
  - Not good for robots!
- Want to keep our robot safe:
  - If in corridor, drive in middle
  - Keep distance from obstacles

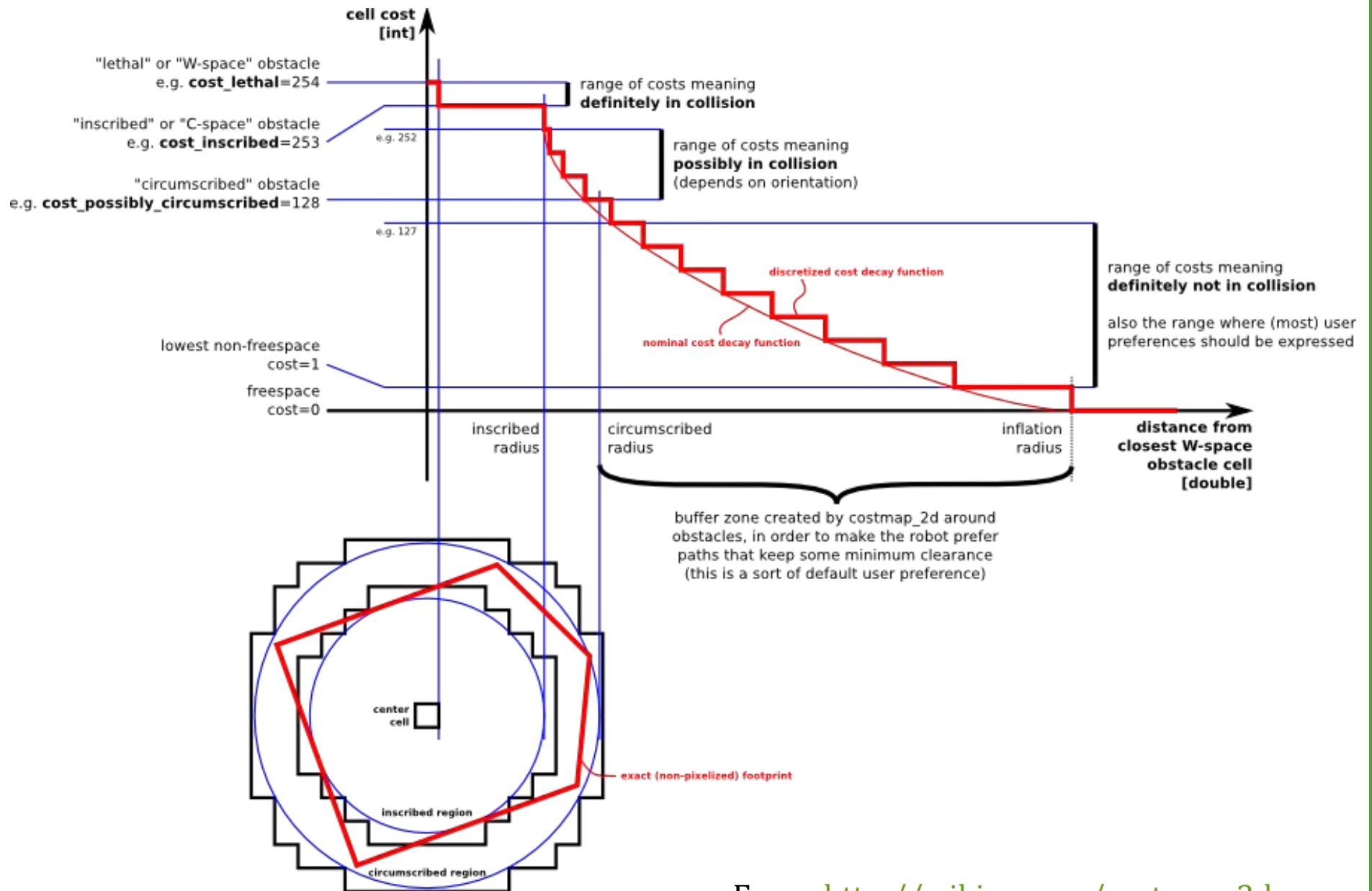


# Robot Distances and Costs

- Robot centre point assumed as point of rotation
  - E.g. the turtlebots
- Obstacles are “lethal”
  - In map
  - Sensed by laser
- Lethal obstacles “inflated” based on the robot size
  - Helps determine distance from obstacles



**RED** – actual robot footprint  
**OUTER CIRCLE** –circumscribed region  
**INNER CIRCLE** – inscribed region

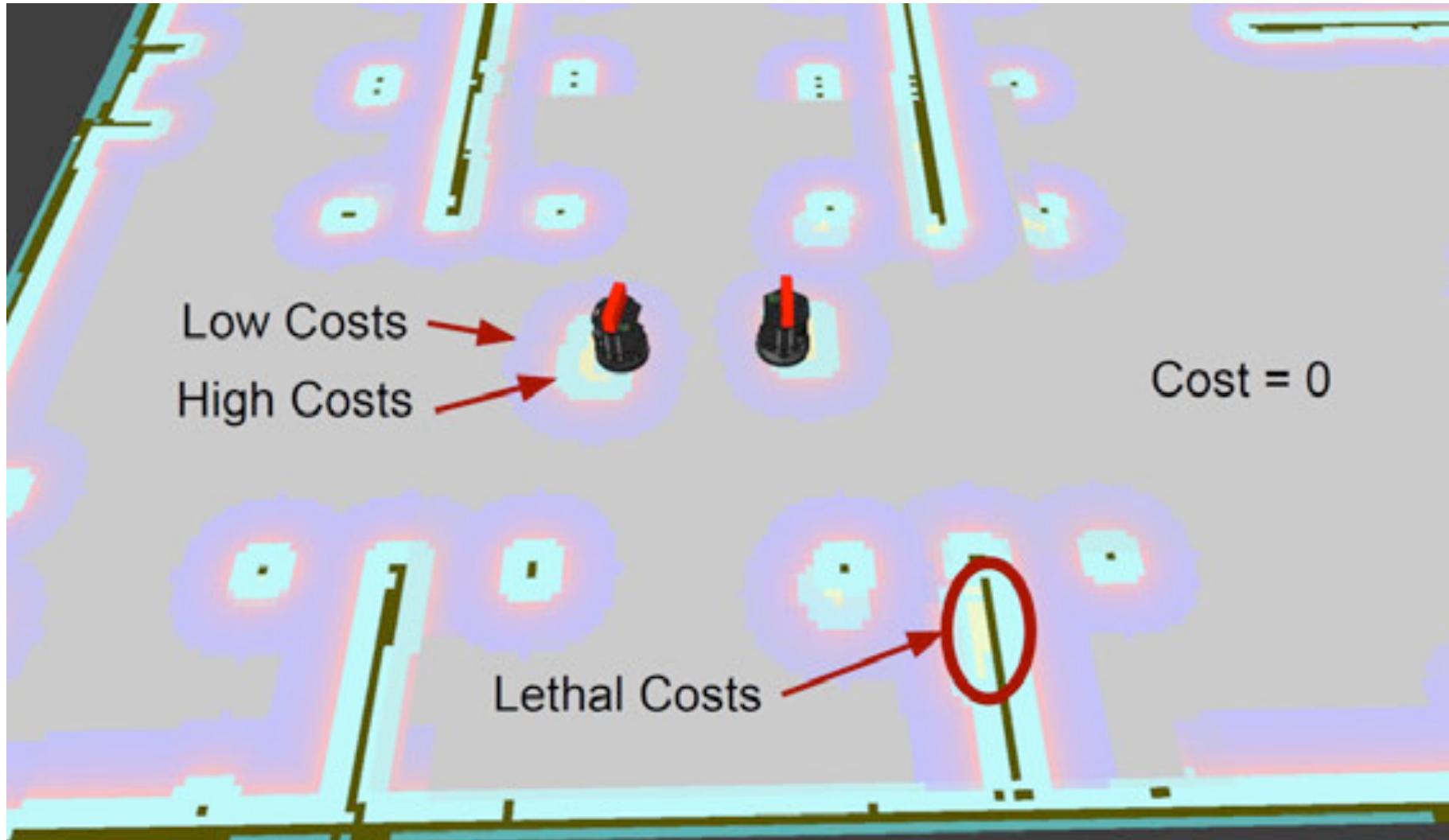


From: [http://wiki.ros.org/costmap\\_2d](http://wiki.ros.org/costmap_2d)

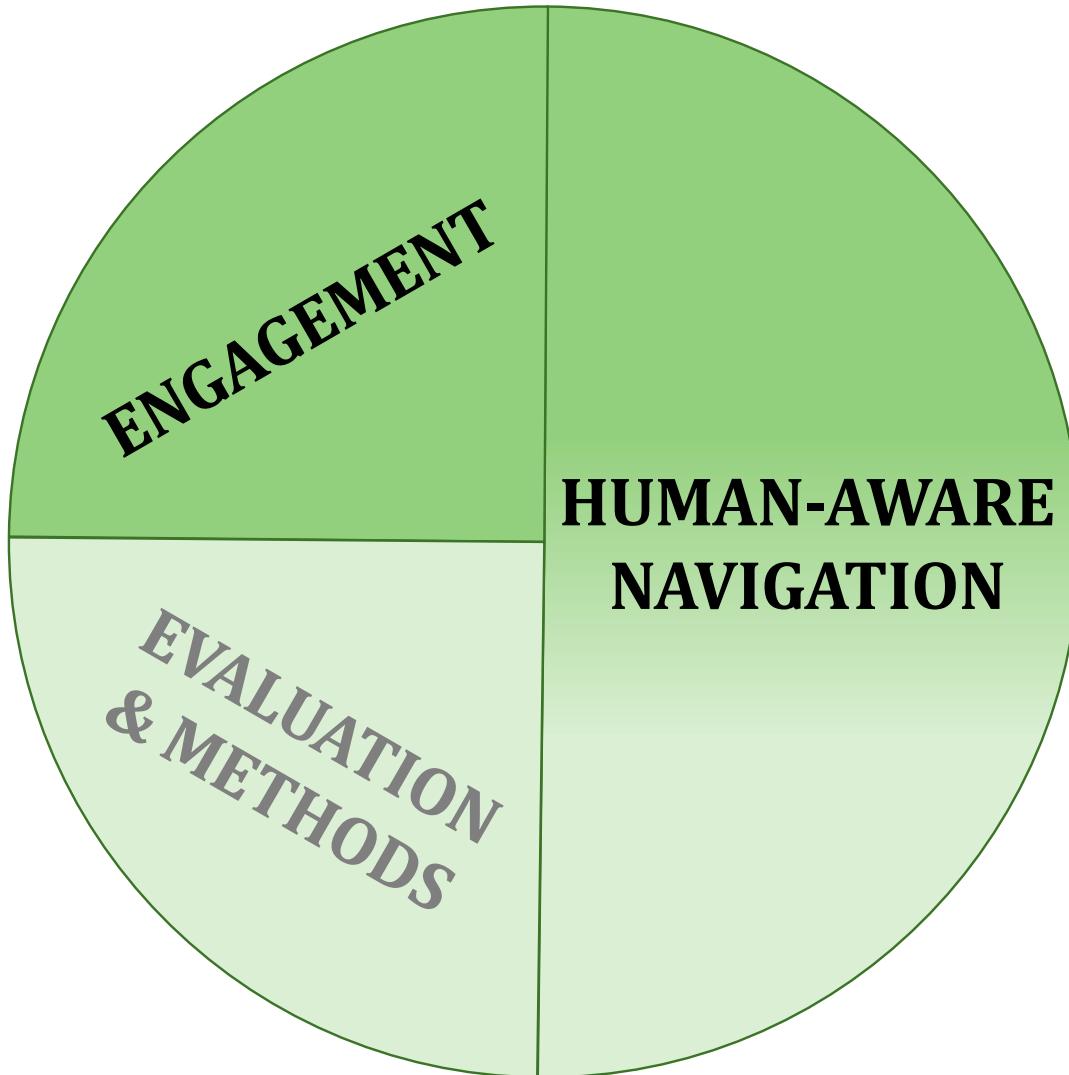
# Using these costs

- Costs encoded into the cost-map, for each obstacle
- In Dijkstra example:
  - Using movement as part of the cost (1 for straight,  $\sqrt{2} = 1.414$  for diagonal)
  - Now also use the cost of the relevant cells in the costmap to calculate the next neighbour
- Result:
  - For navigation, the robot can stay clear of obstacles, even when using Dijkstra for planning

# Costmaps



# Before the break...





## (b) The Human Obstacle

# Humans are not just obstacles...

- Human-Robot Spatial Interaction

The study of joint movement of robots and humans through space and the social signals governing these interactions

- Movement of robots and humans
- Focus on social signals

- Human-Aware Navigation

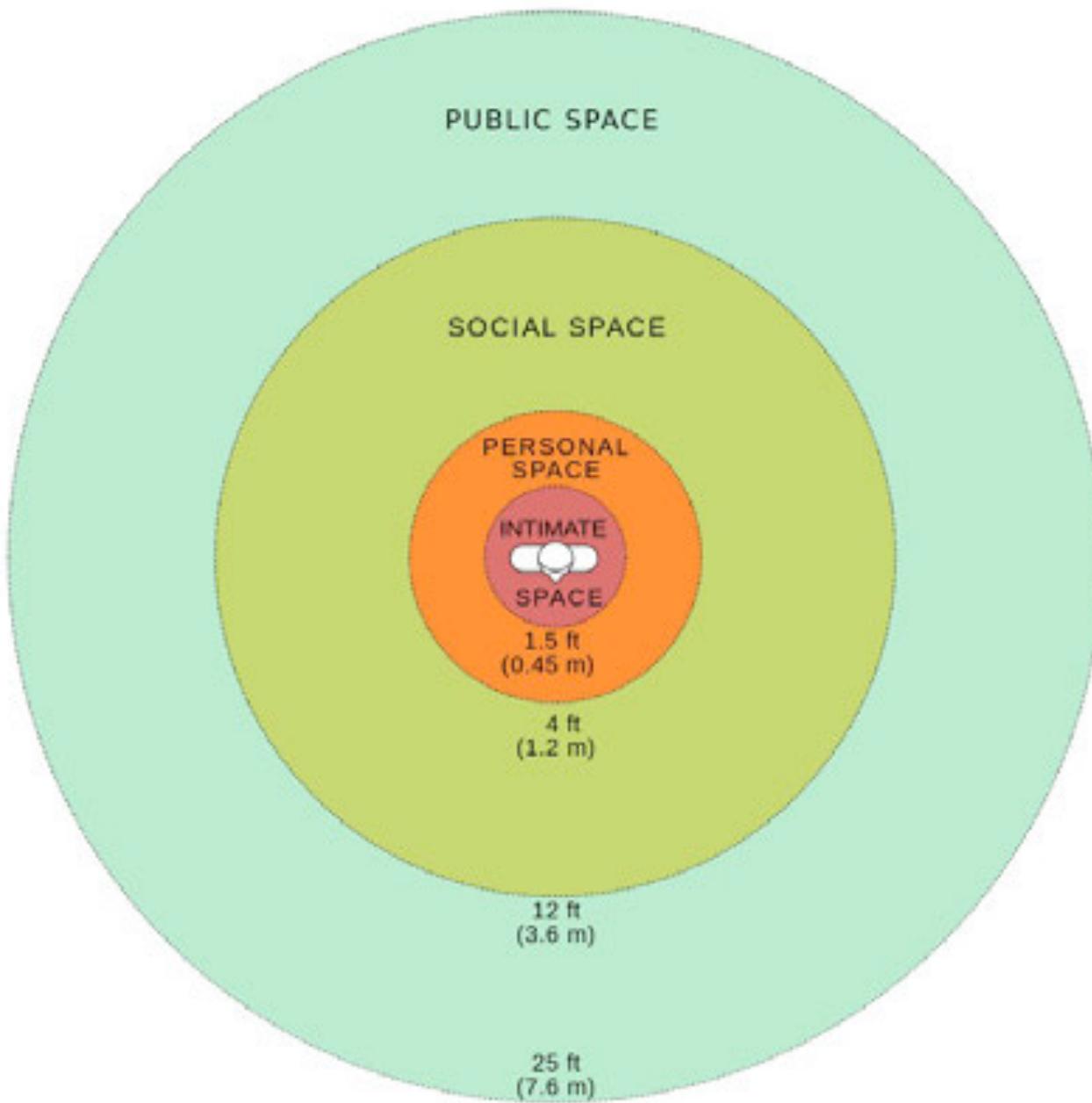
- Specifically taking the human into account
- Recall the three goals: comfort, naturalness, and sociability
- Two main methods:
  1. Stop-and-wait: human does all the hard work
  2. Cost functions based on principles of Proxemics

# Proxemics

- A virtual personal space around an individual
  - Edward Hall, 1966
- Divided into four main zones
  - Each zone at a different distance, and with different interaction characteristics
  - Also dependent on relationship
  - Two 'phases' per zone

The four zones:

1. Intimate
2. Personal
3. Social
4. Public



# Intimate Space

- 0 – 45cm: Intimacy
- Close Phase (0-15cm)
  - Intimacy, comforting
  - Vision blurred, vocalisations whispered
  - Senses of smell and radiant heat effective
  - Arms can encircle
- Far Phase (15-45cm)
  - Hand can reach and grasp extremities
  - Heads, thighs, and pelvis are not easily brought into contact
  - Able to focus the eye easily, peripheral vision includes the outline of the head and shoulders
  - Heat and odour of the other person's breath might be detected

# Personal Space

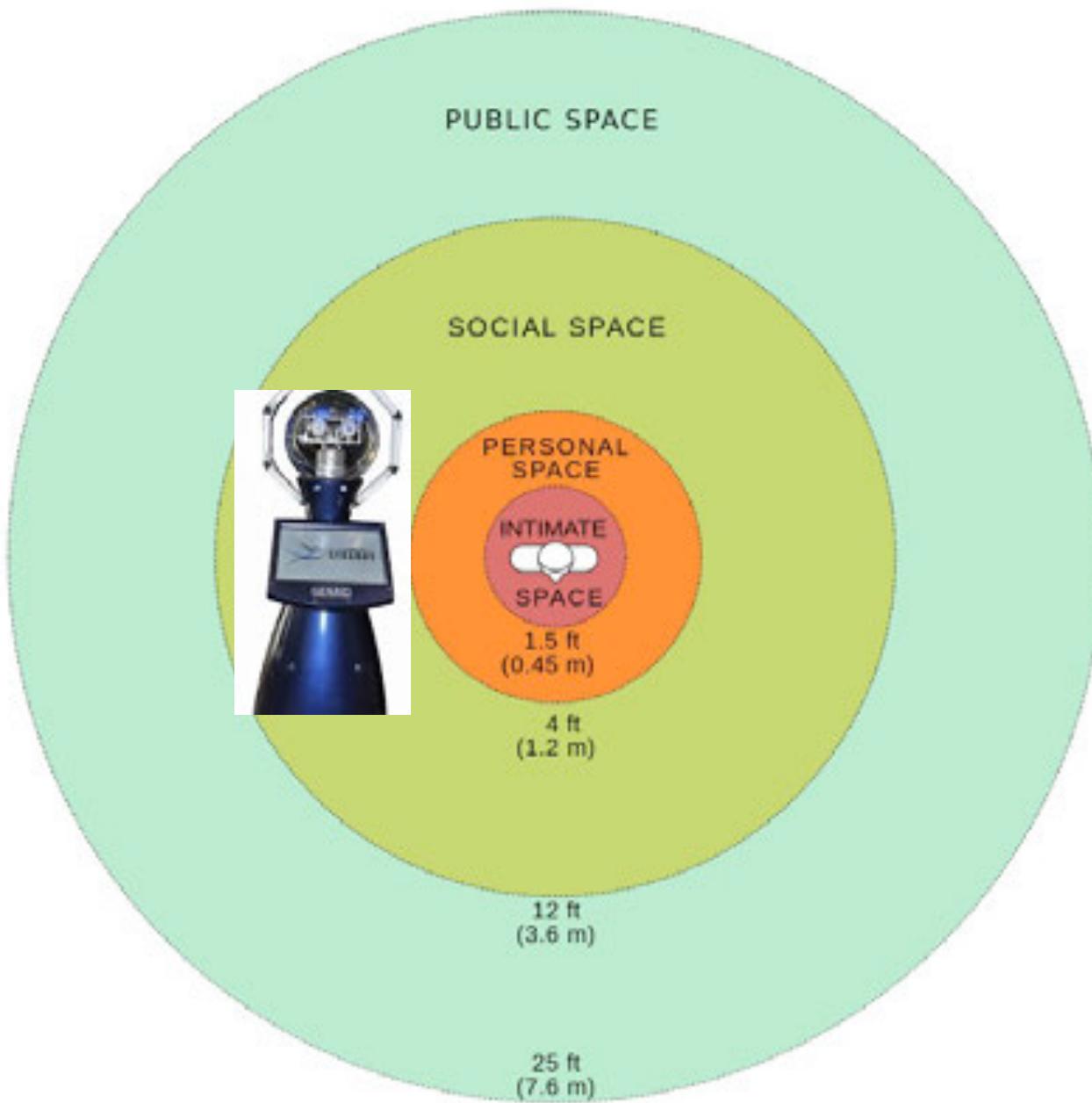
- 45 – 120cm: Family and good friends
- Close Phase (45-75cm)
  - Can grasp other person: peripersonal space
  - No visual distortion of other's features
  - Perception of 3-dimensional qualities of objects
- Far Phase (75-120cm)
  - Outside of grasping distance: at arm's length
  - Other's features clearly visible
  - Moderate voice volume
  - No perception of body heat
  - Lower levels of olfaction

# Social Space

- 1.2m – 3.6m: Interactions with acquaintances/strangers
- Close Phase (1.2-2.1m)
  - No touching without special effort
  - Normal voice volume – can be heard from a moderate distance
  - Visual focus extends to nose and parts of both eyes, or, nose, mouth and one eye
- Far Phase (2.1-3.6m)
  - Fine details of face are lost
  - Skin texture, hair, teeth, and condition of clothes readily visible
  - Odour not detectable

# Public Space

- > 3.6m: public speaking
- Close Phase (3.7-7.6m)
  - Can take evasive actions
  - Voice loud but not full volume
  - Can see whole face, fine details not visible
  - Only whites of eyes visible
- Far Phase (>7.6m)
  - Subtleties of meaning in voice is lost, as are details of facial expressions
  - Vocal, facial, and bodily expression must be exaggerated
  - Foveal vision takes in increasingly more of the other person



# Caveats

- Dependent on culture (Hall's studies were in US)
- Equal spacing around individual
  - No difference between front and back
  - No accounting for movement (e.g. warping of space when walking)
- Do not take into account environmental conditions
  - E.g. dim lighting, loud environments, etc
- Enforced violations?
  - E.g. public transport
  - Changes behaviour
- These zones may be applicable to humans, but do they apply to robots?





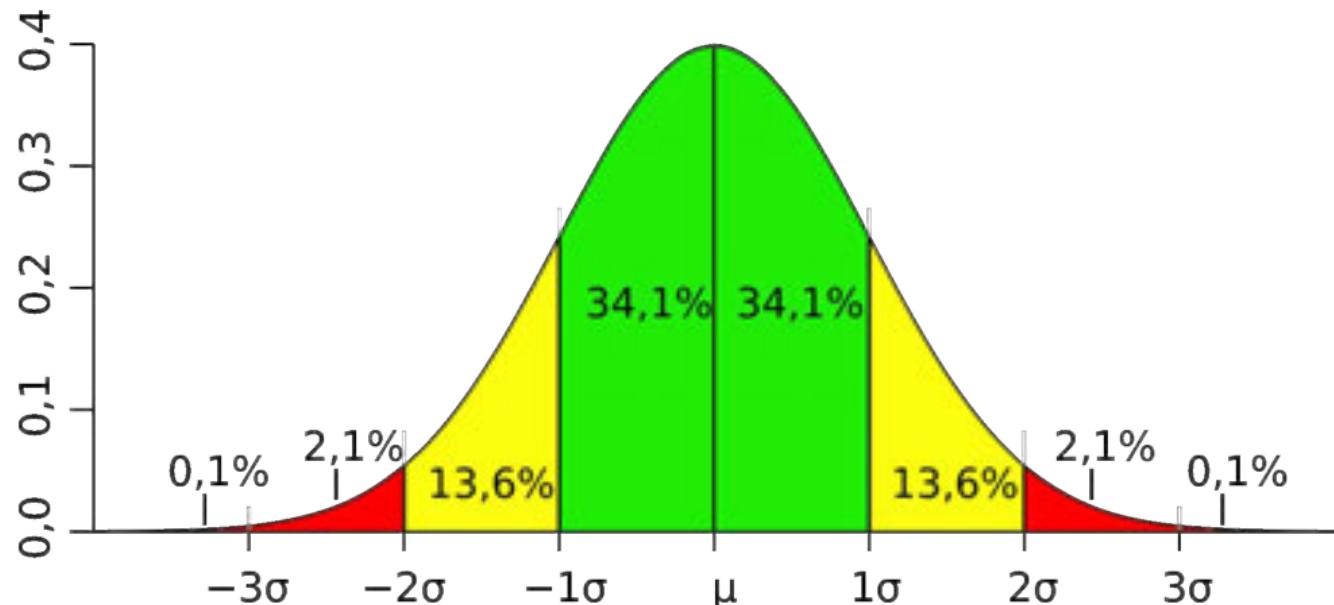
## (c) Bringing These Together

# Combining Costmaps and Proxemics

- Including costmaps as part of the human representation in the map:
  - Including proxemics as part of this
  - What is the benefit?
- Keeping a greater distance to the human
  - Perceived as safer
  - Reduced stress
  - Even though less “efficient”
- Not necessarily either of the other two goals
  - Doesn’t guarantee more natural behaviour
  - Doesn’t incorporate societal/cultural norms (e.g. drive on the left or the right?)
- How to put Proxemics into Costmap?

# Gaussian Distribution and Proxemics

- Sigma – standard deviation (mean of zero)
  - Mean could be position in one dimension (x or y)
- Theoretically infinite, so cut-off at 3-sigma
- Set sigma to size of the Intimate Space
  - This is in only one dimension...
  - For quick visualisation: <http://homepage.stat.uiowa.edu/~mbognar/applets/normal.html>

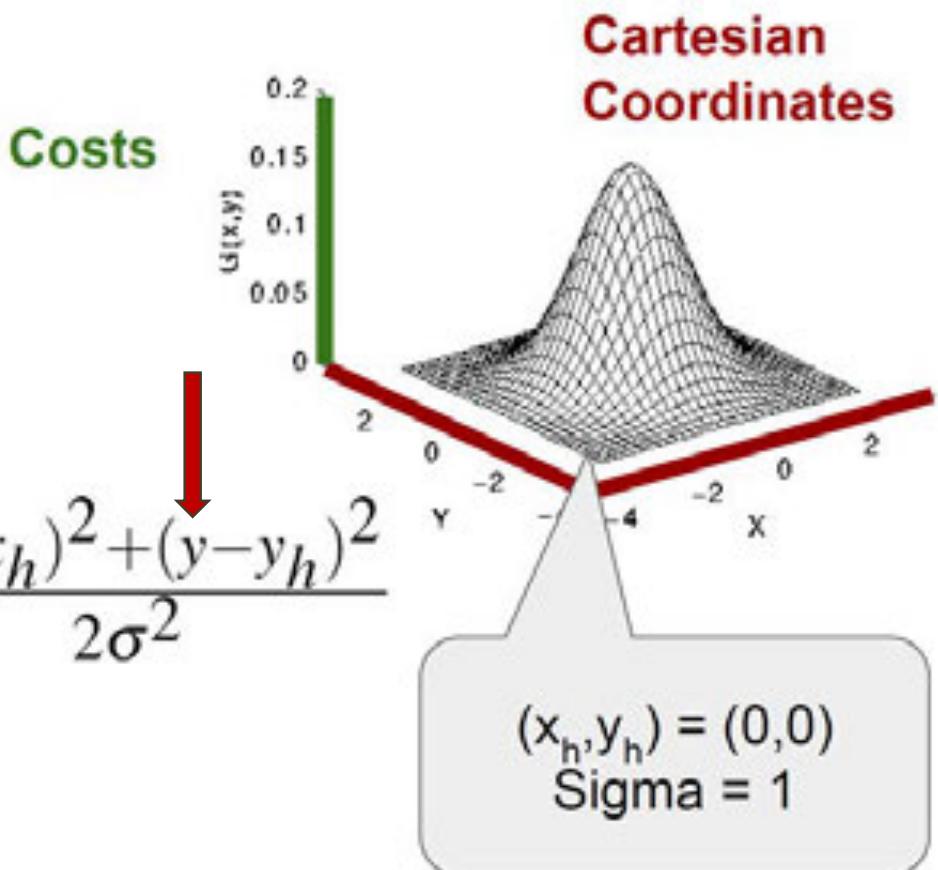


# Extending to 2D...

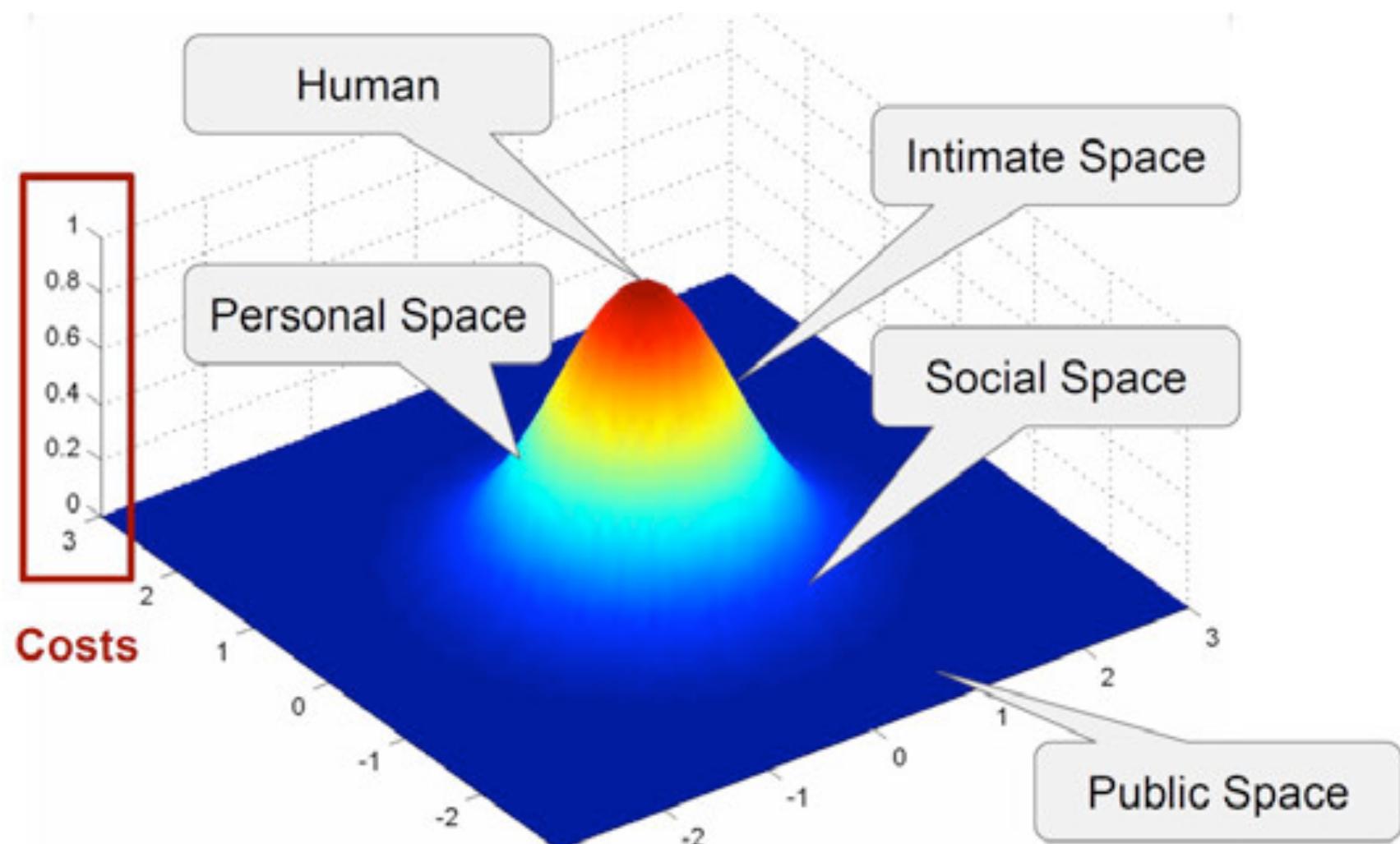
- 2D Gaussian equation – example parameters shown
- Cartesian coordinates centred on the position of the human

Height of the peak

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_h)^2+(y-y_h)^2}{2\sigma^2}}$$

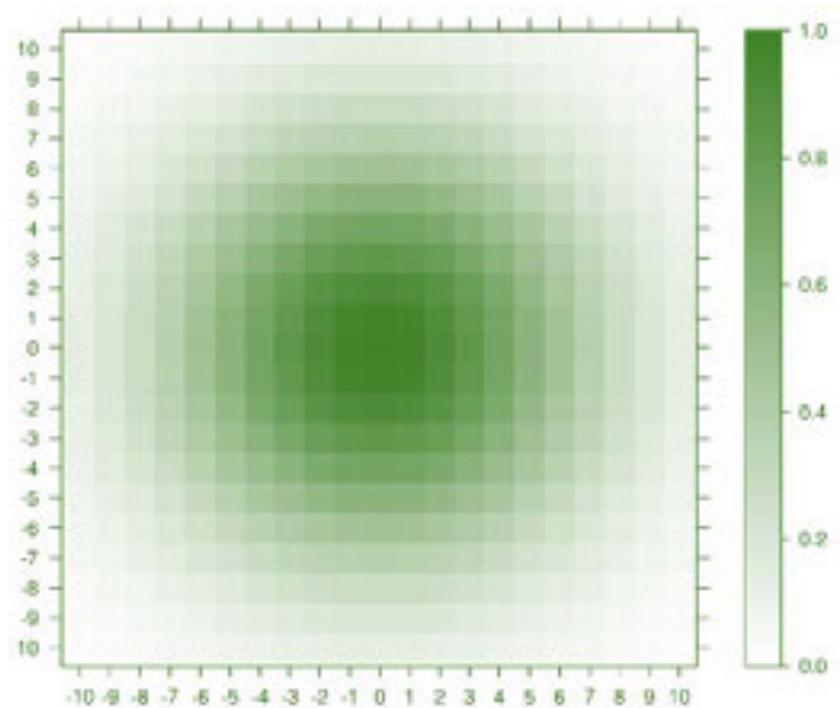
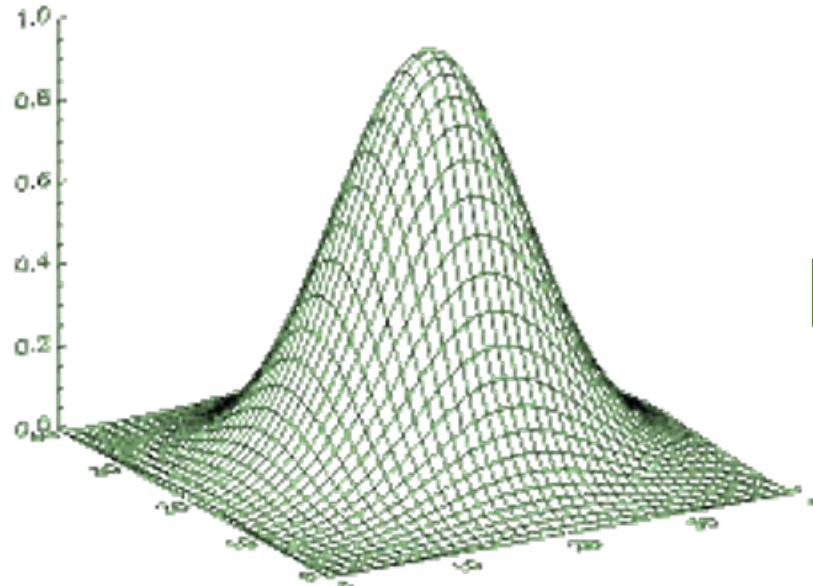


# Proxemic Gaussian



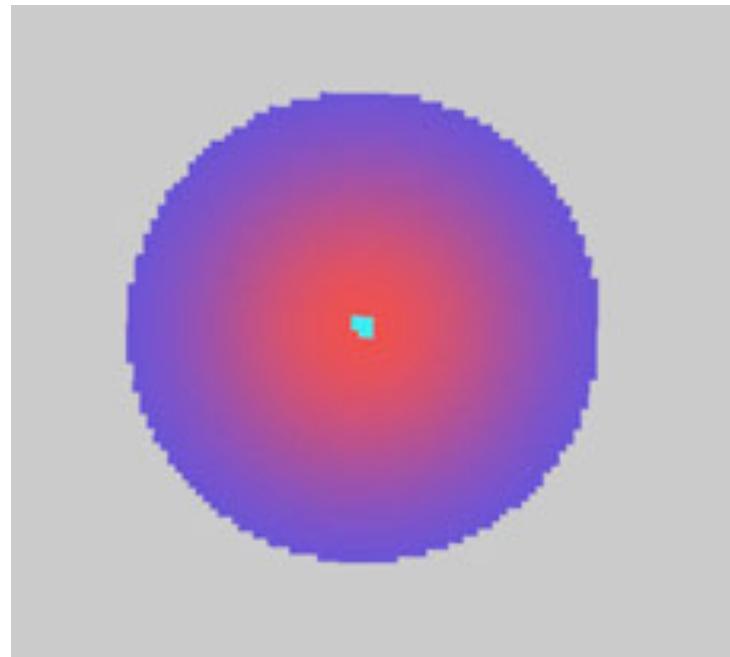
# Discretisation

- Have a continuous function, need it to be discretised

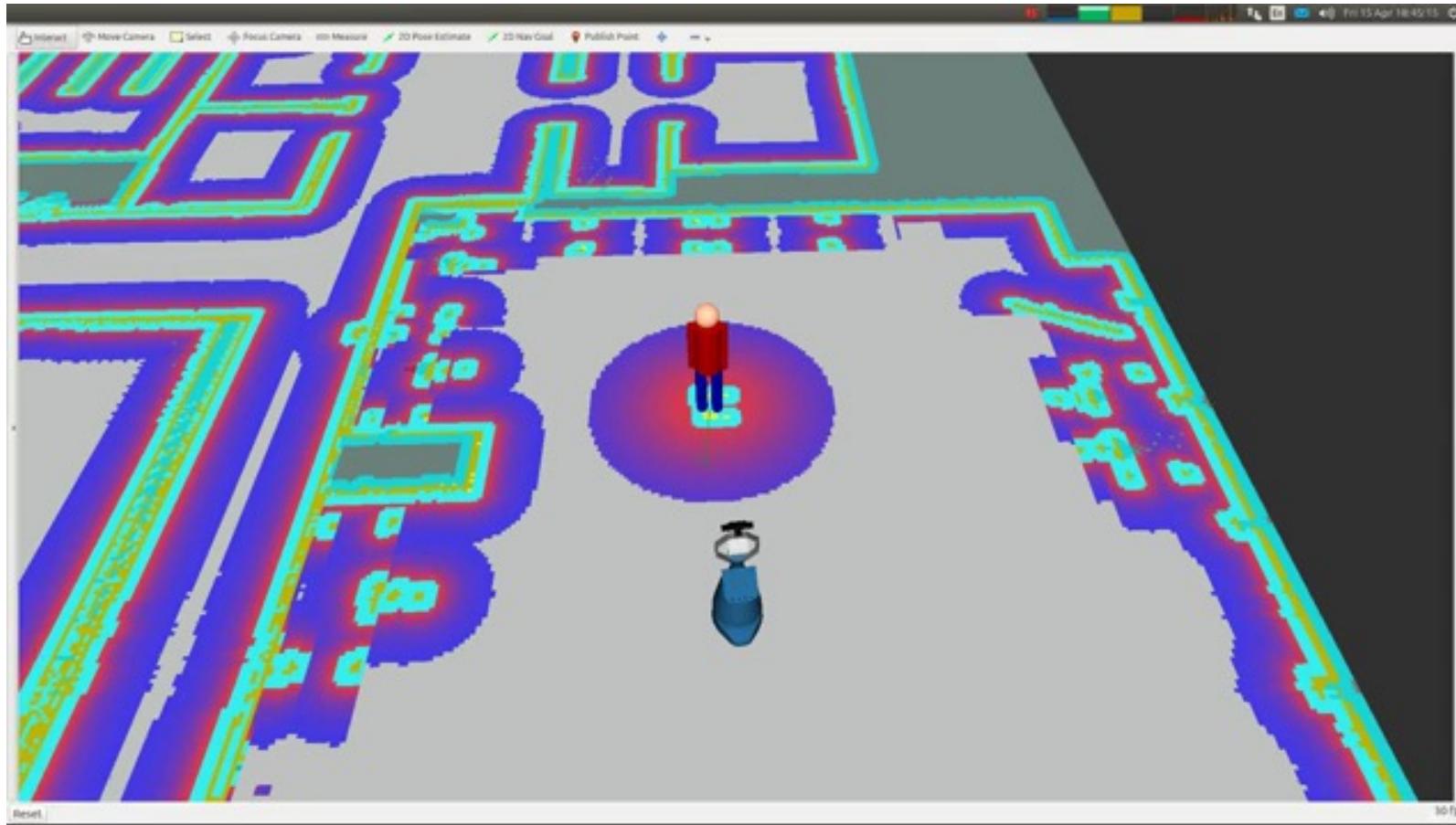


# And back to Costmaps...

- This information can now be added to the overall costmap, with the other obstacles
  - Or added to a separate layer of the costmap (Lu et al, 2014)
- Path planning in this space as described before
  - Dijkstra

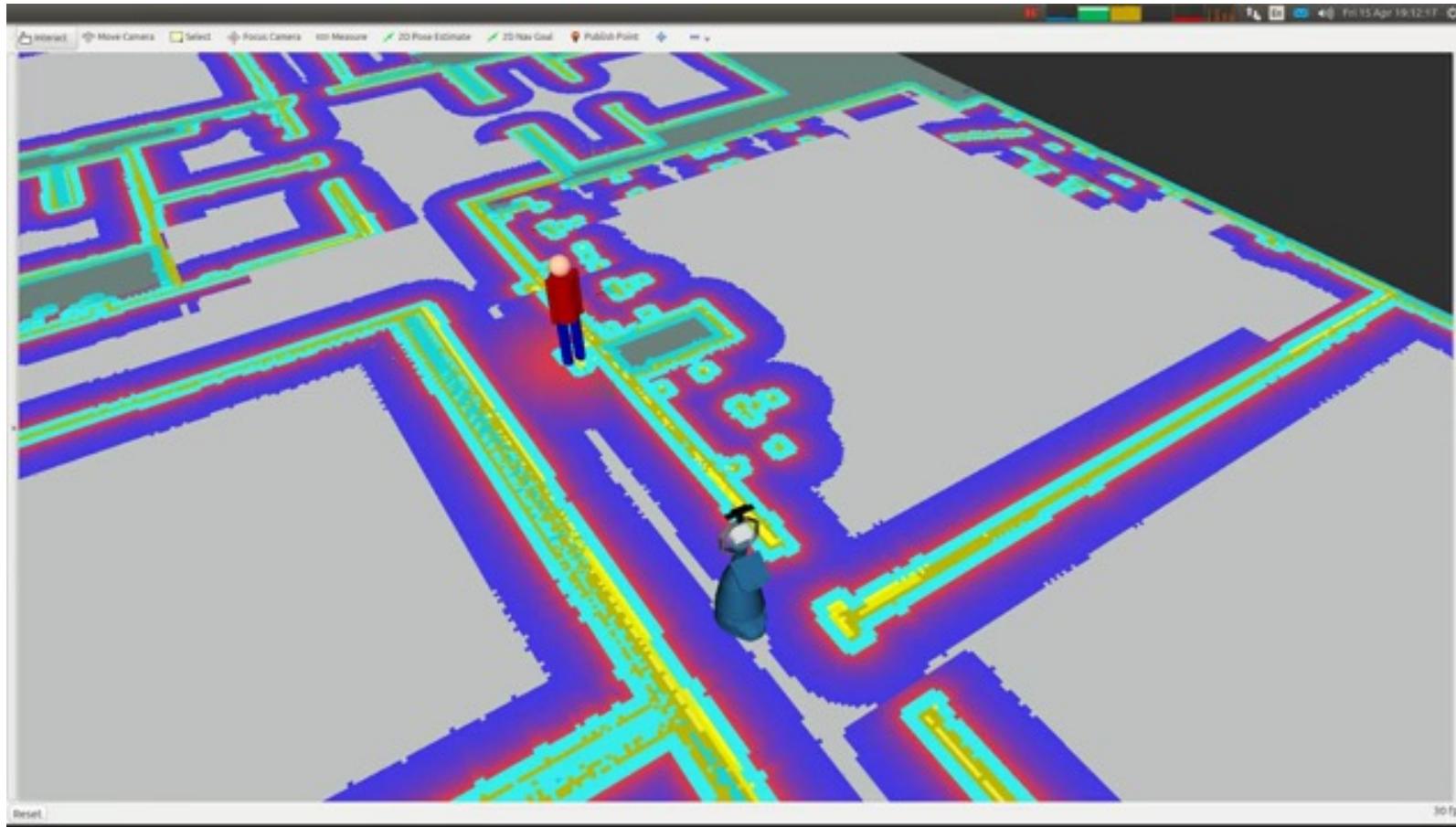


# Navigation in Open Space



Full video: <https://www.youtube.com/watch?v=pg7g7qv80MU>

# Navigation in Corridor



Full video: <https://www.youtube.com/watch?v=pg7g7qv80MU>

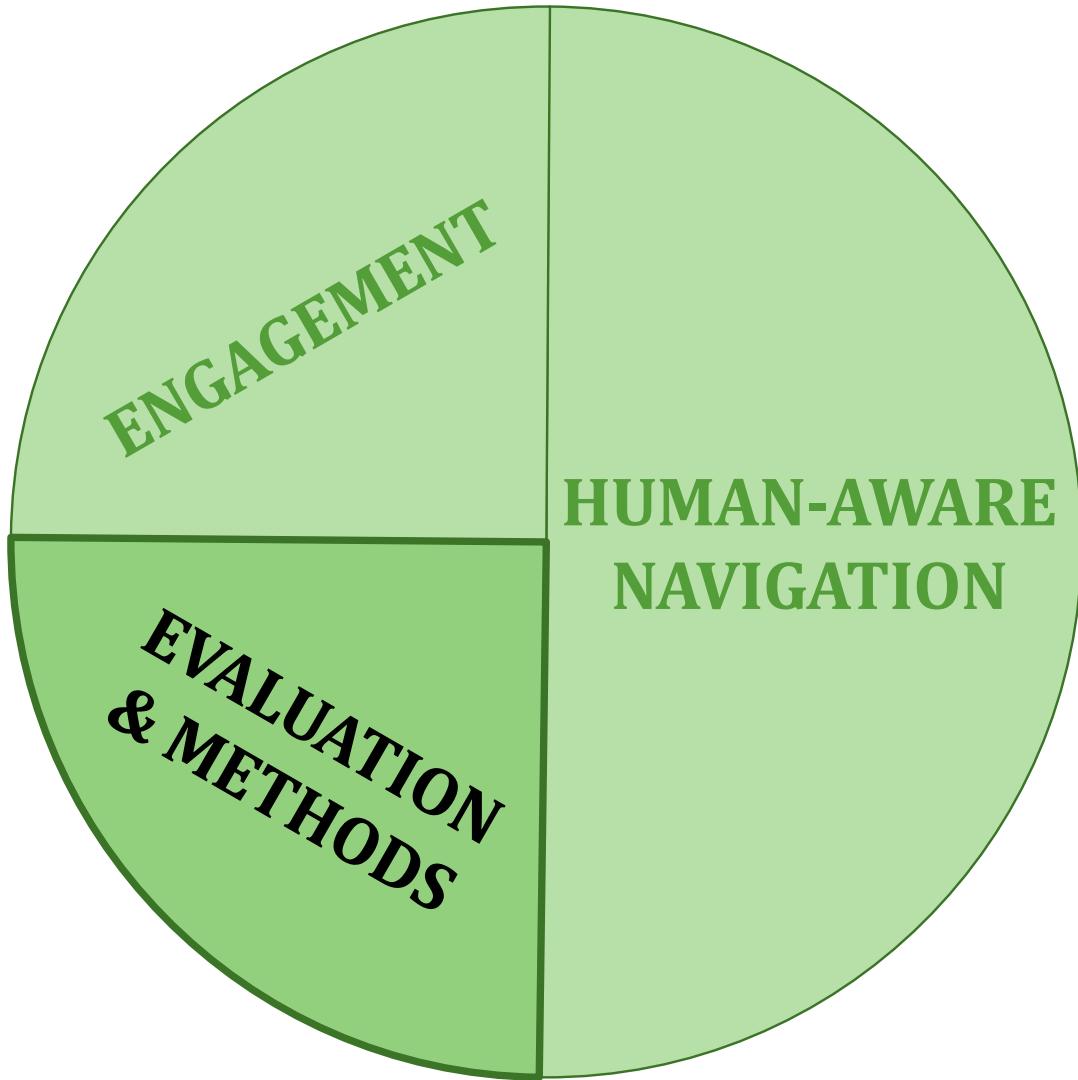
# Gaussian Cost functions

## Positives

- Straightforward implementation
- Is relevant for all environments
- Takes into account proxemics
- Ensures interaction is safe, and perceived to be as such

## Negatives

- Only influences distances
- Sociability and naturalness not guaranteed
- Does not take into account social context
  - Only based on proxemics
  - Could drive through groups?
- Works with Dijkstra
  - Slow/inefficient



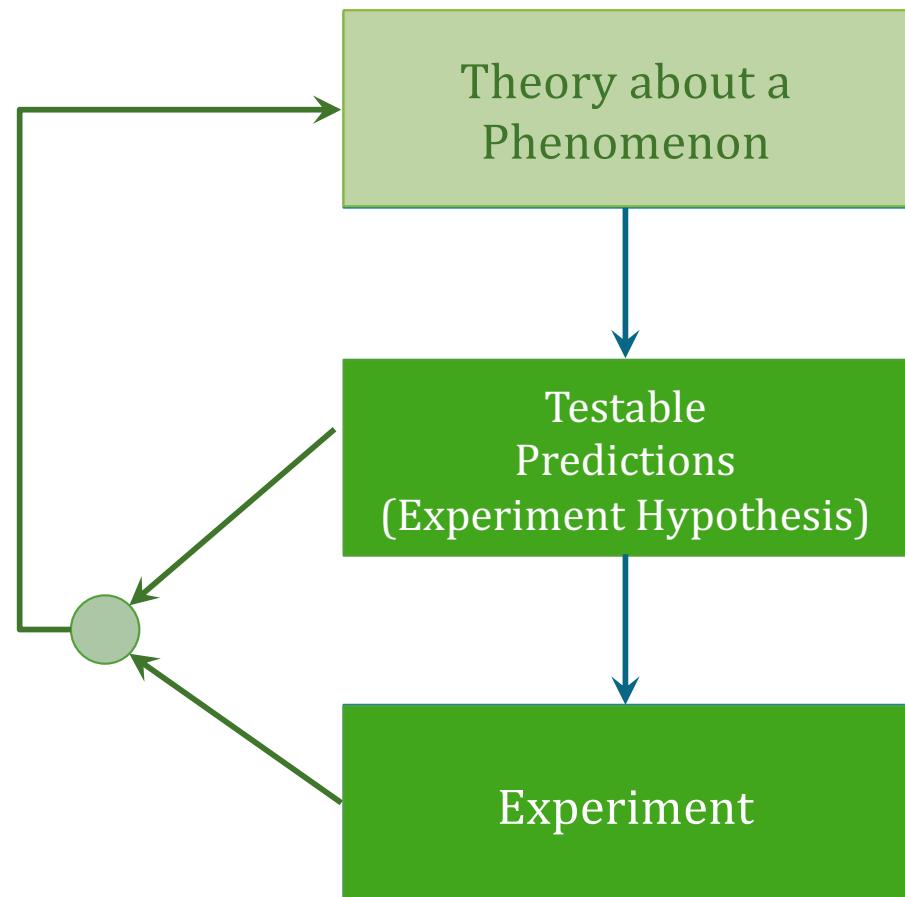
# Is my Robot Successful?

- You have implemented a complete robotic system – how do you know if it is successful?
  - Performance metrics (remember your assignment!)
  - Test with real robots/simulation, etc
- But what if it is an HRI system?
  - Humans are problems...
  - Non-predictable, they come with prior expectations/experience
- Need to evaluate your system with people...
  - Running experiments
  - To verify that the robot has an effect (or is perceived) in the way you (as a designer) intended

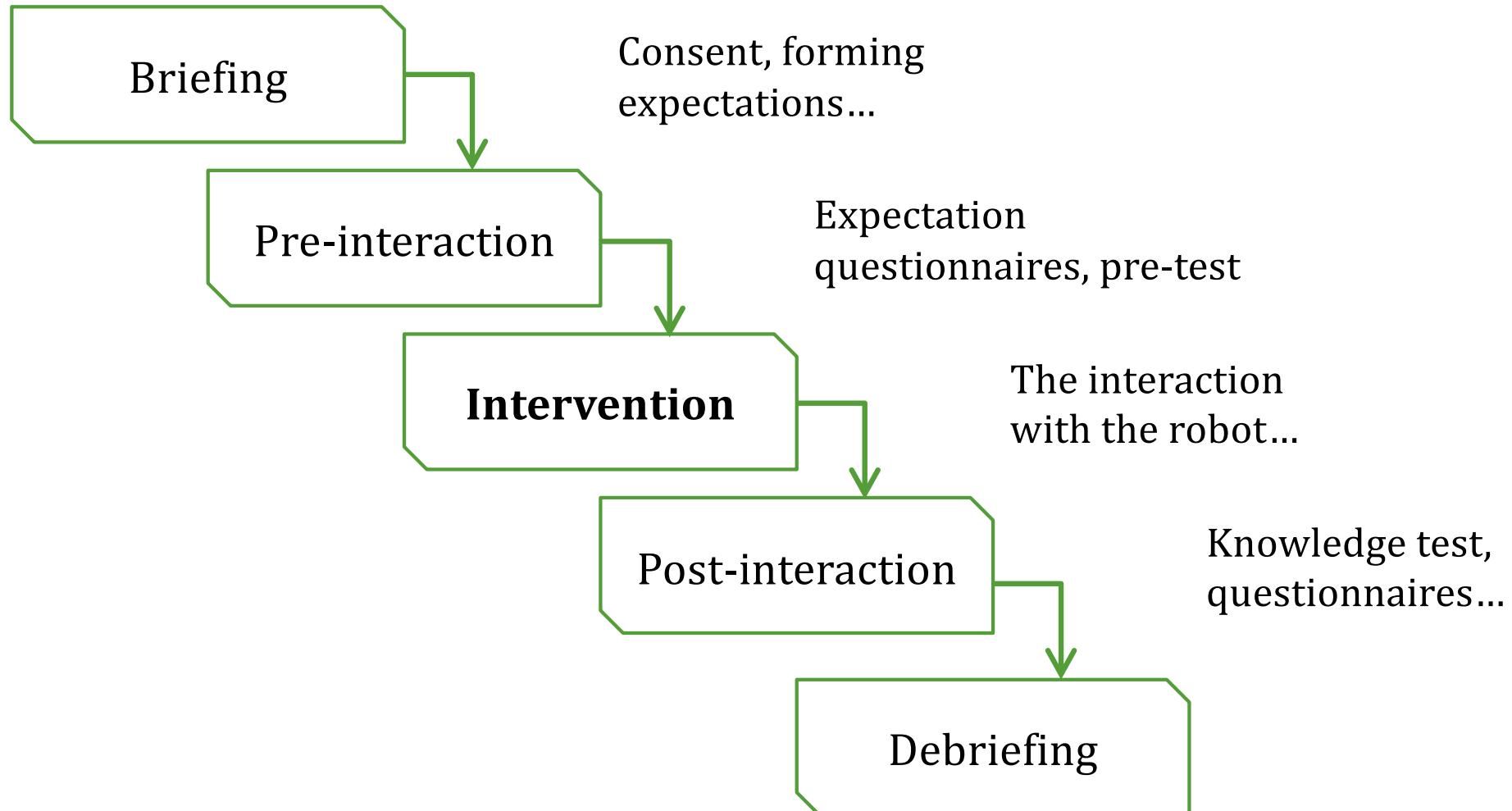


# The Scientific Method

- The “classic” view of the scientific method
  - Generate testable predictions from a theory
  - Perform experiment specifically generated to address the hypothesis
  - Assess the experimental hypotheses: observations in context of the theory
  - Update/refine the founding theory as necessary
- Does not deny role for exploratory studies (later)

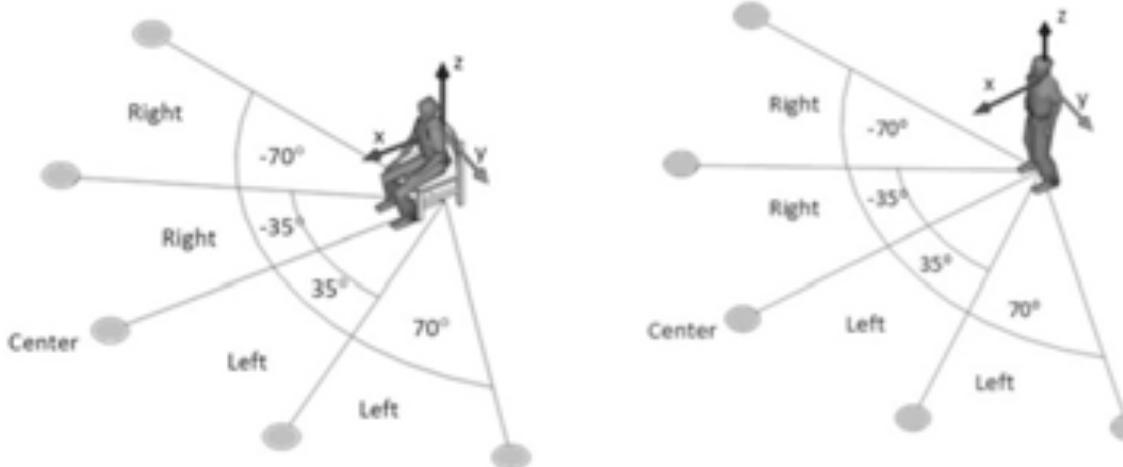


# The structure of an experiment...



# Example HRI Experiment (1)

- Study by (Torta et al, 2013): Design of a parametric model of personal space for robotic social navigation
- Small humanoid (Nao) approached human participants in one of two conditions: sitting or standing
  - Approach from a range of angles: humans would stop it when they wanted to
- Indication that the HRI distance is longer than would be expected in HHI
  - Also see (Walters et al, 2009)
- Effect of approach angle, and whether standing or sitting



# Example HRI Experiment (2)

- Study by (Beck et al, 2010): exploring how body posture of a Nao robot relates to the interpretation of emotion
  - Displaying “key poses” and assessing interpretation
- Research question:
 

Is the interpretation of the key poses displayed consistent with their positions in the Affect Space?
- Participants better than chance at interpreting the five key poses
- Some blending was also possible, though this made it more difficult to identify



Figure 3: Five Key poses generated by the system (A: 100% Sadness. B: 70% Sadness 30% Fear. C: 50% Sadness 50% Fear. D: 30% Sadness 70% Fear. E: 100% Fear).

Table 2: Postures and their main interpretations.

“None” indicates that the question was left unanswered.

Posture	Most common Primary Interpretation (Pi)	Most common Secondary Interpretation (Si)
100% Sadness	Sadness (74%)	None (70%)
50% Sadness 50% Neutral	Neutral (52%)	None (70%)
70% Sadness 30% Pride	Sadness (61%)	None (70%)
50% Sadness 50% Pride	Neutral (52%)	None (52%)

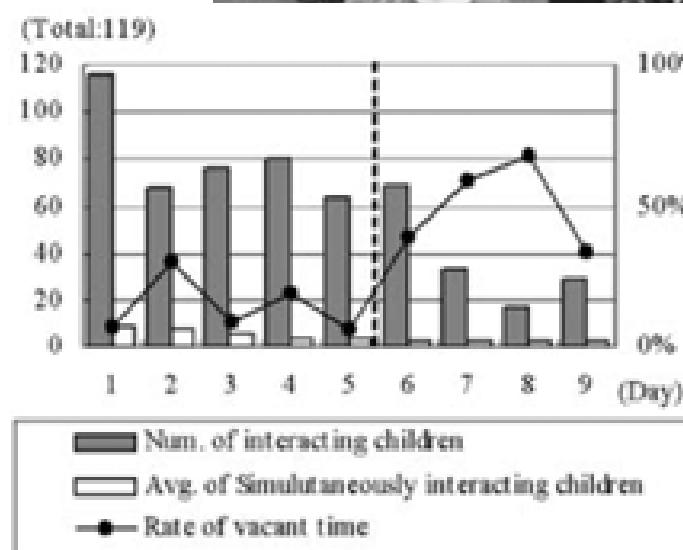
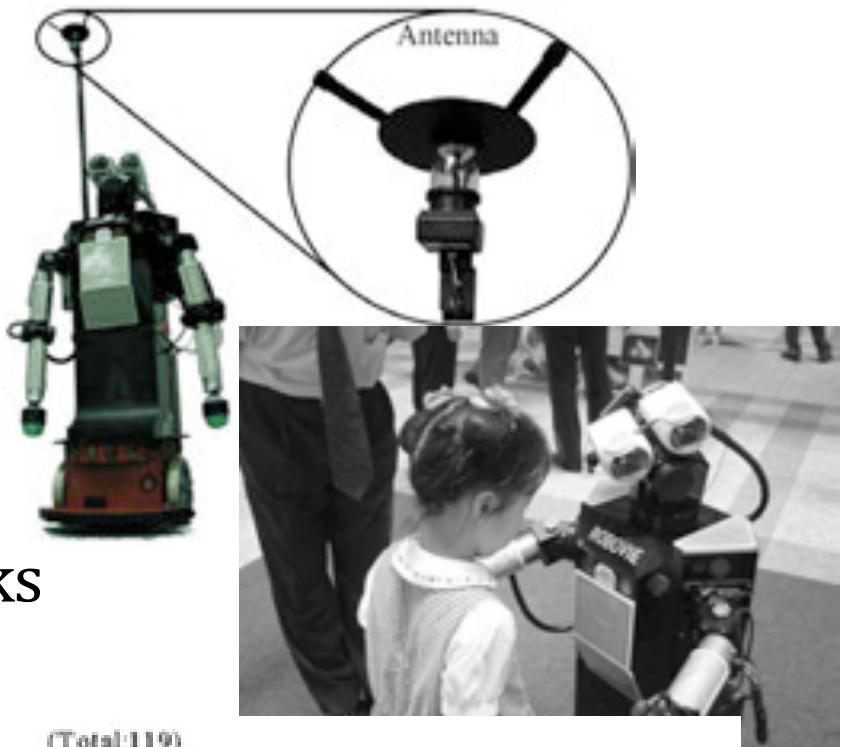
# Pilot studies

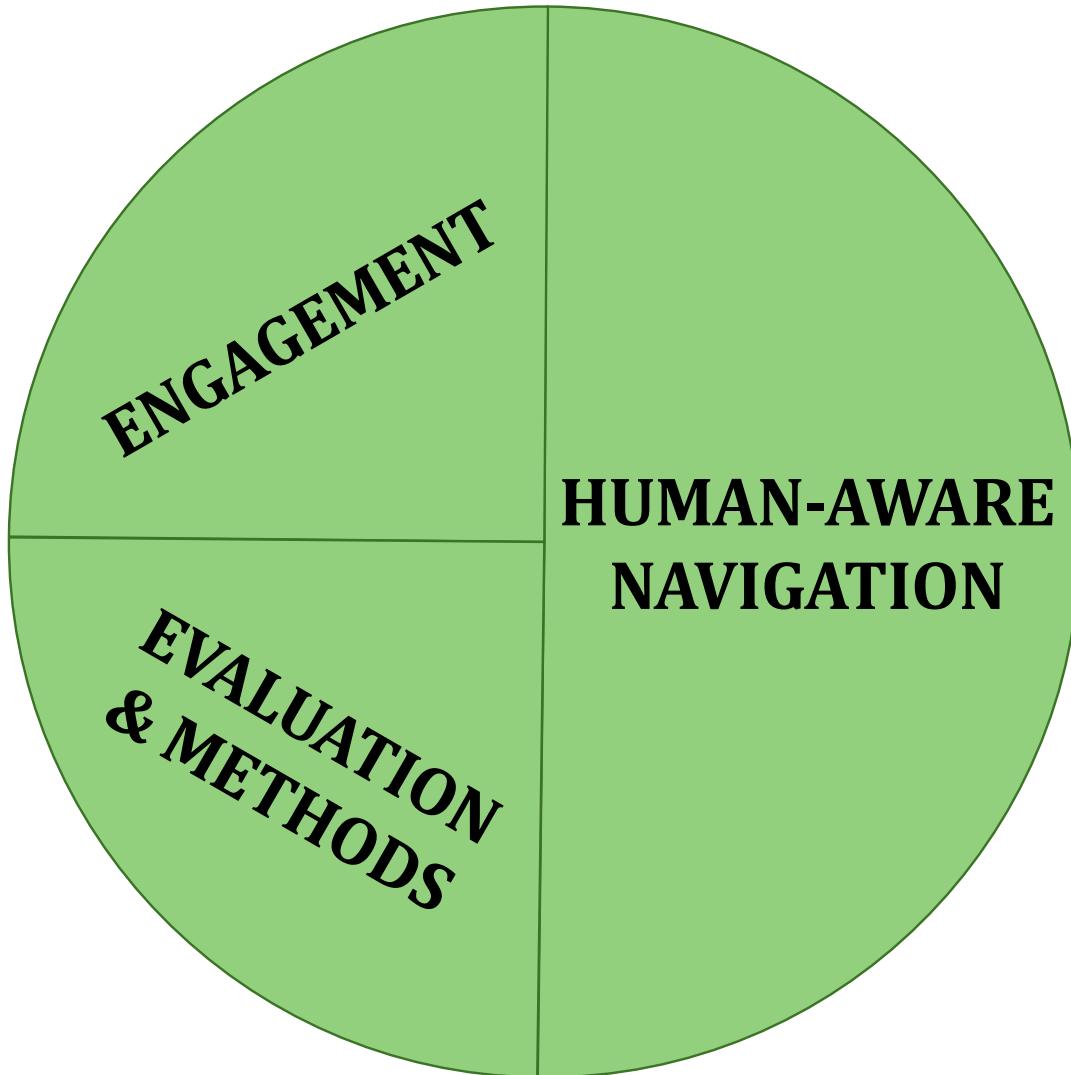
... or *Exploratory Studies*

- Can learn from studies that don't formally fit within the scientific method outlined:
  - Studies with no hypothesis (though would still have some expectations of what will/could happen – if only for ethics)
  - Perhaps a hypothesis, but only a single experimental condition
- Some degree of control and rigour are still required to draw meaningful observations (e.g. Kanda et al, 2007)
  - Reduce the incidence of confounds, or even discover what the confounds are
  - Finding data from which hypotheses can be formed

# Example Pilot Study

- Field trial:
  - (Kanda et al, 2004)
  - Two weeks in a corridor speaking English with Japanese children
  - Watch to see what happens
- Various pre/post expt checks
  - E.g. English level etc
- No explicit hypothesis, one experimental condition
  - Could examine rates of interaction over time
  - And a look at learning...





# References / Reading

- Baxter, P. et al., 2014. Tracking gaze over time in HRI as a proxy for engagement and attribution of social agency. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14*. Bielefeld, Germany: ACM Press, pp. 126–127.
- Beck, A., Hiolle, A., Mazel, A., & Cañamero, L. (2010). Interpretation of emotional body language displayed by robots. *Proceedings of the 3rd International Workshop on Affective Interaction in Natural Environments - AFFINE '10*, 37.
- Glas, N. & Pelachaud, C., 2015. Definitions of Engagement in Human-Agent Interaction. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*. Xi'an, China: IEEE Press, pp. 944–949.
- Gonzalez-Arjona, D. et al., 2013. Simplified Occupancy Grid Indoor Mapping Optimized for Low-Cost Robots. *ISPRS Int. J. Geo-Information*, 2, pp.959–977.
- Hall, E., 1966. The Hidden Dimension, Anchor Books
- Kanda, T. et al., 2004. Interactive Robots as Social Partners and Peer Tutors for Children: A Field Trial. *Human-Computer Interaction*, 19(1), pp.61–84.
- Kruse, T. et al., 2013. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12), pp.1726–1743.
- Lemaignan, S. et al., 2016. From real-time attention assessment to “with-me-ness” in human-robot interaction. In *ACM/IEEE International Conference on Human-Robot Interaction*. Christchurch, New Zealand.
- Lu, D. V, Hershberger, D. & Smart, W.D., 2014. Layered Costmaps for Context-Sensitive Navigation. In *IROS 2014*. Chicago, USA: IEEE, pp. 709–715.
- Sidner, C.L. et al., 2005. Explorations in engagement for humans and robots. *Artificial Intelligence*, 166(1–2), pp.140–164.
- Tanaka, F., Cicourel, A., & Movellan, J. R. (2007), “Socialization between toddlers and robots at an early childhood education center”, *PNAS*, 102(46), 17954–17958.
- Torta, E., Cuijpers, R.H. & Juola, J.F., 2013. Design of a Parametric Model of Personal Space for Robotic Social Navigation. *International Journal of Social Robotics*, 5(3), pp.357–365.
- Walters , M L , et al, 2009 , 'An empirical framework for human-robot proxemics' in *Procs of New Frontiers in Human-Robot Interaction : symposium at the AISB09 convention* . pp. 144-149 .