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

Access Code: 450006

UNIVERSITY OF LINCOLN

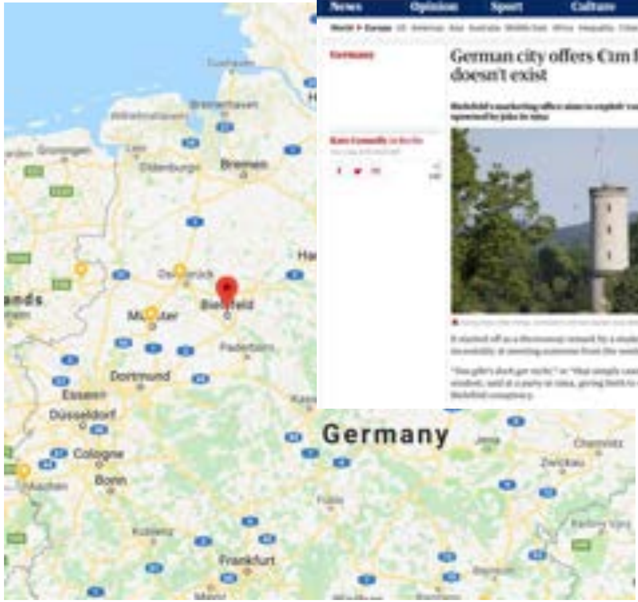
CMP3103 – AUTONOMOUS MOBILE ROBOTS

Lincoln Centre for Autonomous Systems

School of Computer Science

L-CAS

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



The Guardian

News Opinion Sport Culture Lifestyle

German city offers €1m for proof it doesn't exist

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

UNIVERSITY OF LINCOLN

Access Code: 450006

Access Code: 450006

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

Universität Bielefeld

UNIVERSITY OF BIRMINGHAM

UNIVERSITY OF LINCOLN

2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 | 2019

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%)

Access Code: 450006

Semester B: AMR

Lectures

Workshops

Ubuntu 22.04 LTS in docker, home installation possible

Robot programming in ROS2, in simulation, potentially with opportunity to try real robots later

Attendance of all scheduled sessions is mandatory!

Please check your timetable & blackboard for any timing & updates

Access Code: 450006

What’s on?

Semester B week	w/c	Lecture (9-11)	Topic	Lecturer for Lecture
1	30/01/2023	Tuesday, 31 January 2023	Intro	Marc Hanheide
2	06/02/2023	Tuesday, 7 February 2023	Robot Programming ROS	Marc Hanheide
3	13/02/2023	Tuesday, 14 February 2023	Robot Sensing and Computer Vision	Jonathan Cox
4	20/02/2023	Tuesday, 21 February 2023	Motion and Control	Athanasios Polydoros
5	27/02/2023	ENHANCEMENT WEEK		
6	06/03/2023	Tuesday, 7 March 2023	Robot Behaviour	Jonathan Cox
7	13/03/2023	Tuesday, 14 March 2023	Navigation	Athanasios Polydoros
8	20/03/2023	Tuesday, 21 March 2023	Localisation	Athanasios Polydoros
9	27/03/2023	Tuesday, 28 March 2023	Robot mapping - SLAM	Athanasios Polydoros
		EASTER		
		EASTER		
10	17/04/2023	Tuesday, 18 April 2023	Control Architecture	Athanasios Polydoros
11	24/04/2023	Tuesday, 25 April 2023	HRI1	Marc Hanheide
12	01/05/2023	Tuesday, 2 May 2023	HRI2	Marc Hanheide
13	08/05/2023	Tuesday, 9 May 2023	ASSESSMENTS	Athanasios Polydoros & Marc Hanheide

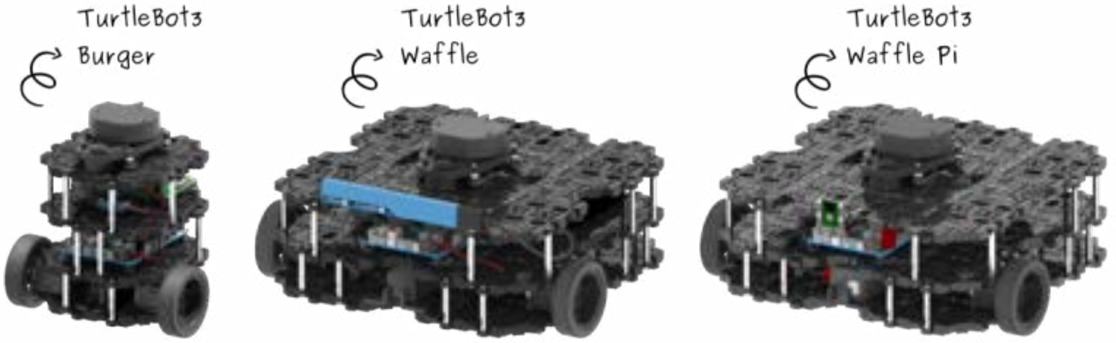
Access Code: 450006

Turtlebot 3 – Our Workshop Buddy

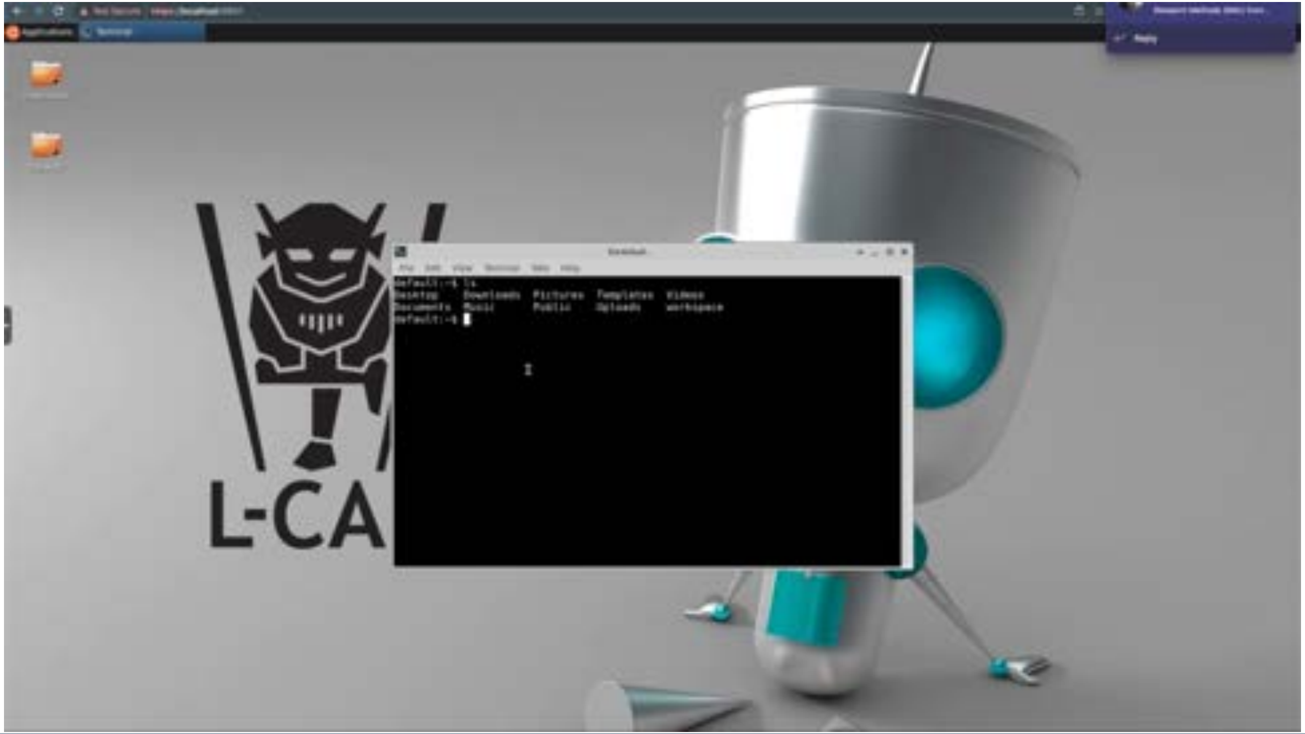
ROS Enabled

This year's assignment: "find the objects" (more soon…)

TURTLEBOT3



Simulated Turtlebots in Docker ROS2 Desktop



Coursework

University of Lincoln Assessment Framework

Assessment Briefing Document

Module Code & Title CMP3103 – Autonomous Mobile Robots (***ITEM 1 – COURSEWORK***)
Contribution to Final Module Mark 40% (CMP3103M) 30% (CMP9050M)
Description of Assessment Task and Purpose In this assessment you are expected to build a software artefact, controlling a simulated robot in Python, using the ROS2 middleware, running on Ubuntu Linux, for a robot to search an environment, looking for coloured items and reporting their location. This is a classical robotic visual search task. You will be marked on the quality of your implementation and the performance of your developed robotic solution.
Learning Outcomes Assessed • [LO2] understand and critically evaluate the range of possible applications for mobile robotic systems • [LO3] implement and empirically evaluate intelligent control strategies, by programming autonomous mobile robots to perform complex tasks in dynamic environments
Knowledge & Skills Assessed • Python, Robotics, and Computer Vision Programming • Use of a middleware for inter-component communication (ROS2) • Interaction with a Linux operating system, running in a Docker container

Coursework

Assessment Submission Instructions

"Visual Object Search"

Your task (relating to Criterion 1 and 2 in the CRG) is to develop an object search behaviour, programmed in Python using ROS2, that enables a robot to search for coloured objects visible in the robot's camera. This assessment is purely done in simulation, and not on the real robot, to provide a fair and equitable assessment of your skills. As part of this task, you must submit an implementation (criterion 1) and a video (criterion 2).

Implementation (Criterion 1)

Your task is to implement a behaviour that enables the robot in simulation to find ideally all designated objects distributed in a simulated environment. You need to utilise the robot's sensors (camera and LiDAR) and its actuators (wheels) to guide the robot to each item. Success in locating an item is defined as: (a) being less than 1m from the item, and (b) indication from the robot that it has found an object (a suitable statement printed on the screen is sufficient).

For the development and demonstration of your software component, you will be provided with a simulation environment (implemented in the "Gazebo" simulator). The required software is provided ready to go as a Docker image, running on any computer in the SoCS labs. The simulated environment includes several differently and brightly coloured objects hidden in the environment at increasing difficulty (some are visible directly from the starting position, while others are hidden behind walls and obstacles). Your robot starts from a predefined position. You will be provided with a "training arena" in simulation. This "training arena" will resemble the "test arena" in terms of structure and complexity (same floor plan of the environment), but the positions of the objects will vary to assess the generality of your approach. Your approach much implement a strategy to search objects anywhere in the environment, and not assume that the positions are known upfront.

You may choose any sensors available on the robot to drive your search behaviour. However, your system design should include the following elements:

1. Perception of the robot's environment using the RGBD sensor, either in RGB or Depth space, or using a combination of both RGB and Depth data in order find the object;
2. An implementation of an appropriate control law implementing a search behaviour on the robot. You may choose to realise this as a simple reactive behaviour or a more complex one, eg, utilising a previously acquired map of the environment;
3. Motor control of the (simulated) robot using the implemented control law.

The minimum required functionality consists of a simple reactive behaviour, allowing in principle to find objects. For an average mark, the behaviour should be able to successfully find some objects at unknown locations. Further extensions are possible to improve your mark in this assessment, also to enable you to find all objects. Possible extensions to the system may include (Note: These are just suggestion. You will be given credit for any creative and clever use us of technologies):

- An enhanced perception system – in-built colour appearance learning, use of additional visual cues (e.g. edges), combination of RGB and Depth features, etc.;
- Exploiting maps and other structural features in the environment or more clever search strategies.
- Utilising other existing ROS components that are available (like localisation, mapping, etc)

The software component must be implemented in Python and be supported by use of ROS2 to communicate with the robot. The code should be well-commented and clearly structured into functional blocks. You may reuse existing ROS2 packages or code you find online, but you <u>must</u> provide references to any resources you use. References should be included as comments in the code, including a proper referencing style, e.g., if it is an online resource from something like the ROS2 tutorials or from Stackoverflow, you must include the full URL to the resource).

Your implementation needs to be submitted via blackboard, with the source code containing good documentation and references.

Coursework

Video/Presentation (Criterion 2)

You are also required to submit a short video (of 3 minutes maximum duration, no more than 40Mb total size, MP4 format) of the simulation 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 40Mb and in the correct format (MP4). Note, we cannot accept a link to a video shared on YouTube or anywhere else online. You must submit the video file.

Your video (with the voice-over) should cover a presentation of 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 in the simulation during the recording or post-edit your video to show different situations you want to highlight. Make sure you present <u>all</u> your hard work briefly in the video recording.

Some Reading Materials

- Bekey, G.A.: Autonomous robots. MIT Press, 2005
- Siegwart, R. and Nourbakhsh, I.R.: Introduction to autonomous mobile robots. MIT Press, 2004
- Murphy, R.R.: An Introduction to AI Robotics. MIT Press, 2000
- More on reading list at https://rl.talis.com/3/lincoln/lists/D6049576-71B1-1B7B-6FB5-87587A983F82.html

Support

- Use the module's MS Team (not Discord ;-))
- Be active in the workshops, utilise the demonstrator **Ibrahim Hroob**
- 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 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

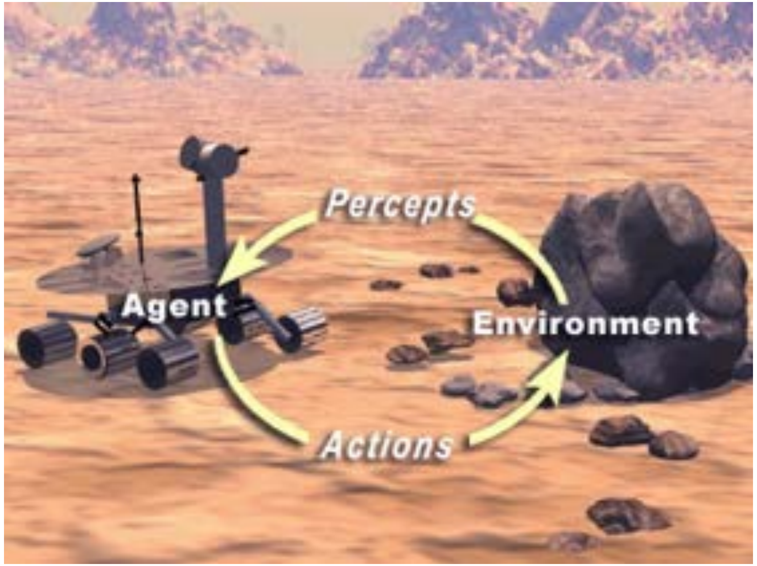
- https://pollev.com/mhanheide

Poll Everywhere

What do you associate with the word robot?

ROBOTICS - A light introduction

Robotics



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

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)

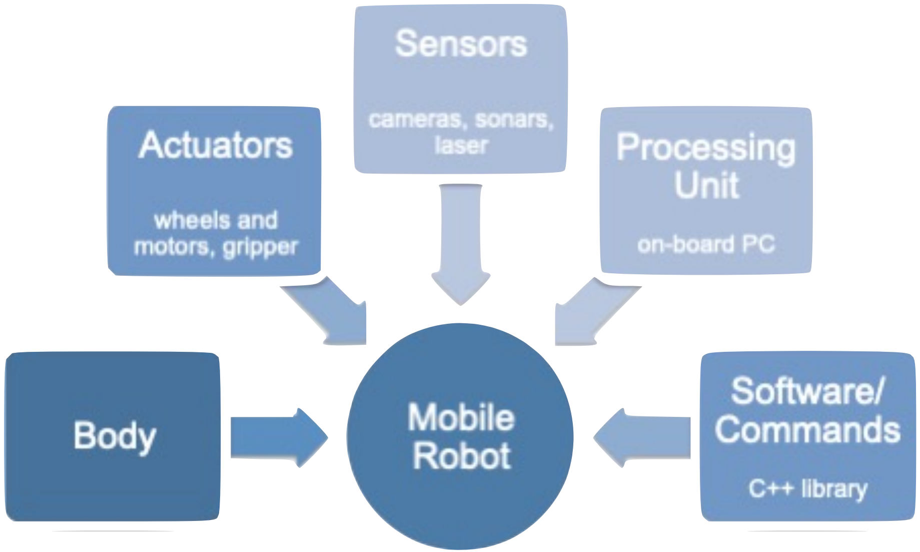
Internal Sensors

External Sensors

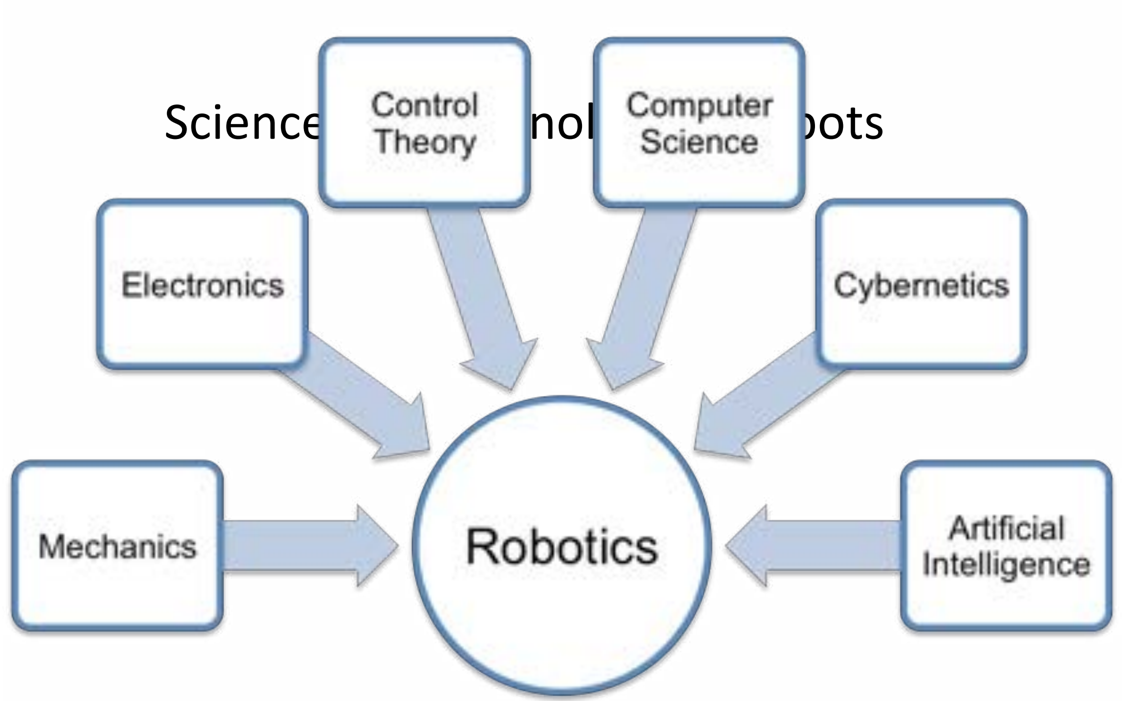
Mobile Robots

- 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

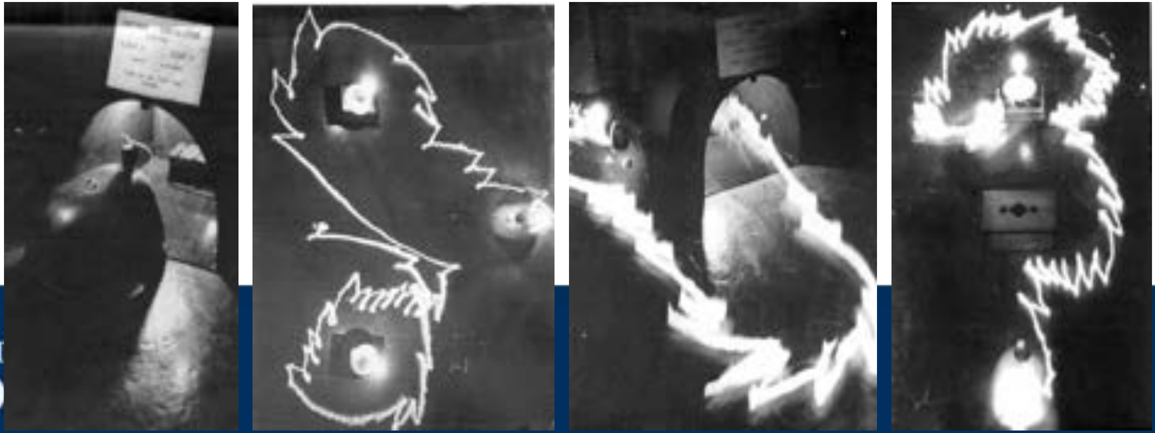

Control Theory
Computer Science
Electronics
Cybernetics
Mechanics
Robotics
Artificial Intelligence

R.U.R.

- Karel Čapek, 1921
  - R.U.R. - Rossum's Universal Robots, a stage play
  - the word 'robot' appears for the first time
  - 'robota' – 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

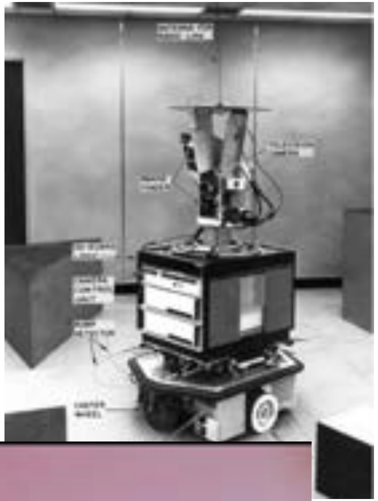
Grey Walter’s Turtles

- “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

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?

UNIVERSITY OF LINCOLN

BREAK?!

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 exterioceptive sensors?

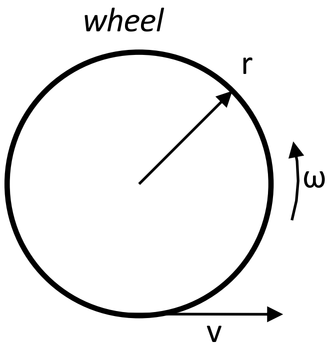
Laser scanner
Sonar
CPU Thermometer
Wheel encoder
Radar
Microphone
Camera
Speaker
Gyroscope
GPS

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

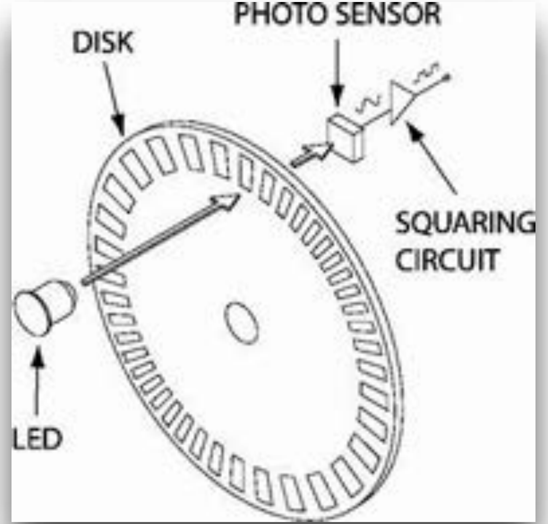


$v = r\omega$

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

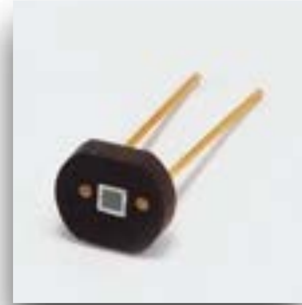
- Wheel slippage, uneven friction and wheel size, etc. can cause errors that accumulate with time

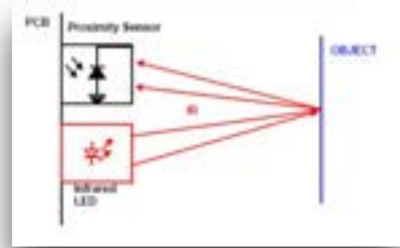
Floor plan and robot route

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

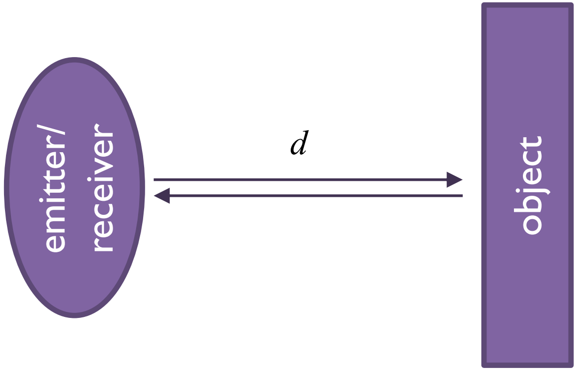
- Active distance measurements – reflected signal

signal speed

$$d = \frac{v \cdot t}{2}$$

distance

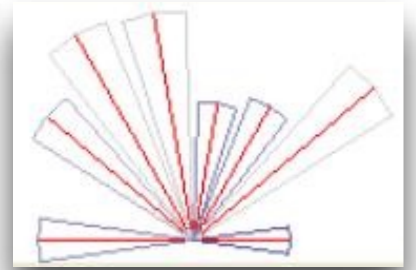
time elapsed between the start of emission and reception



Sonar Sensor

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

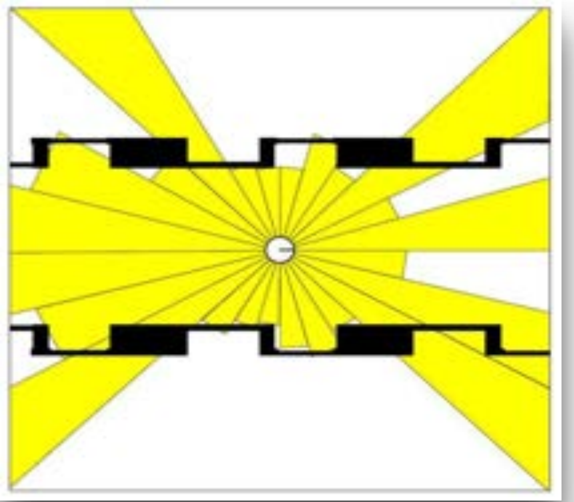
sonar reading



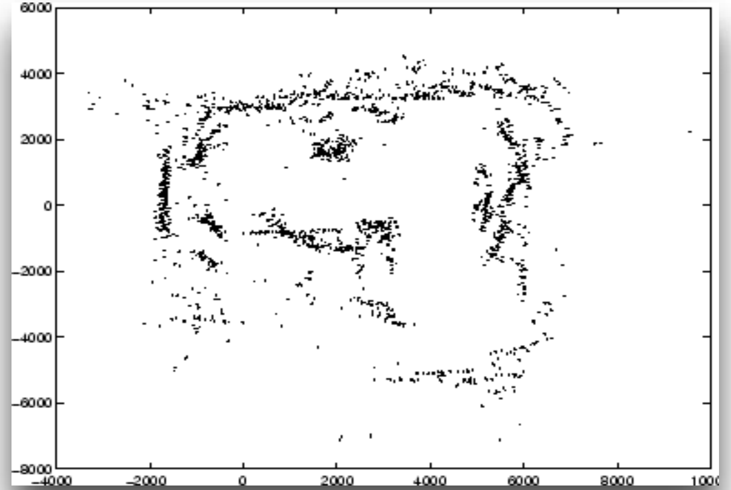
an array of
sonar sensors

Sonar – Applications

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



sonar reading in a corridor

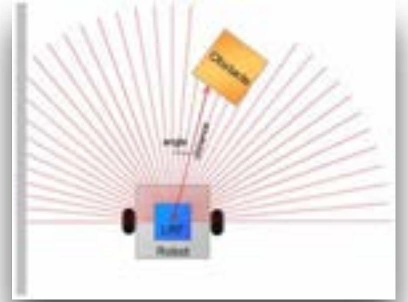


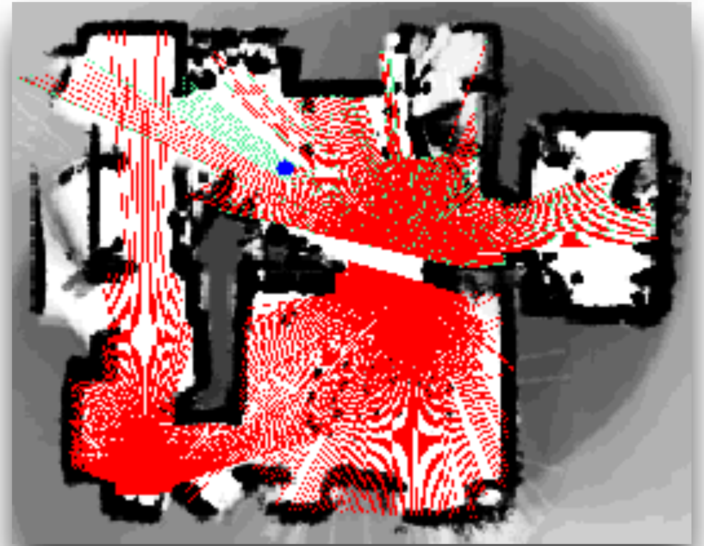
sonar map

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)



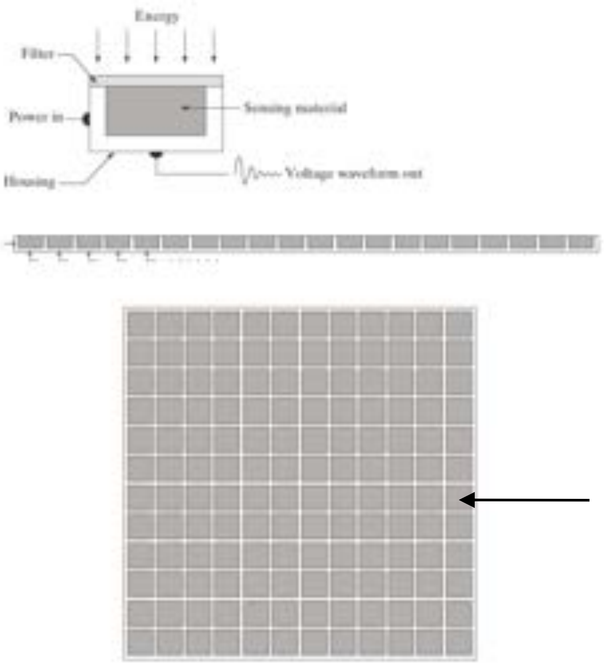


2d maps and people detection

3d maps

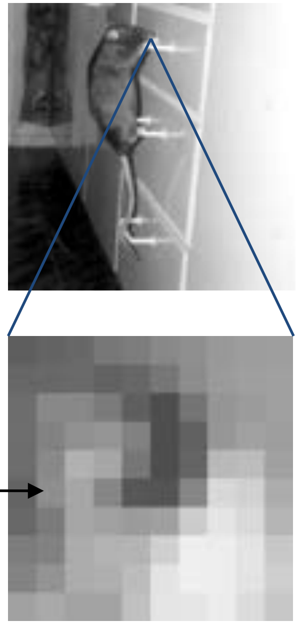
Vision Sensor

Vision Sensor



pixel

Digital Image



a rat

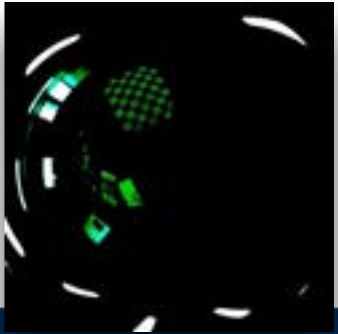
the rat's nose

Different Type of Vision Sensors

colour

thermal

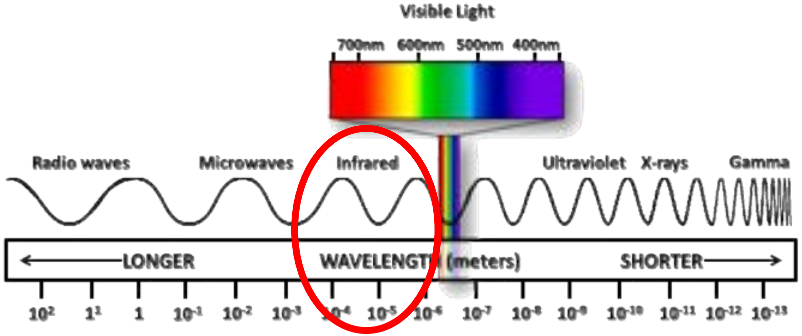
omni-directional

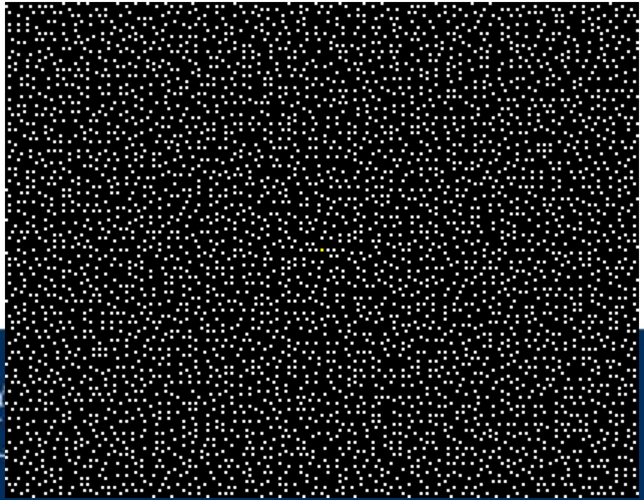


stereo

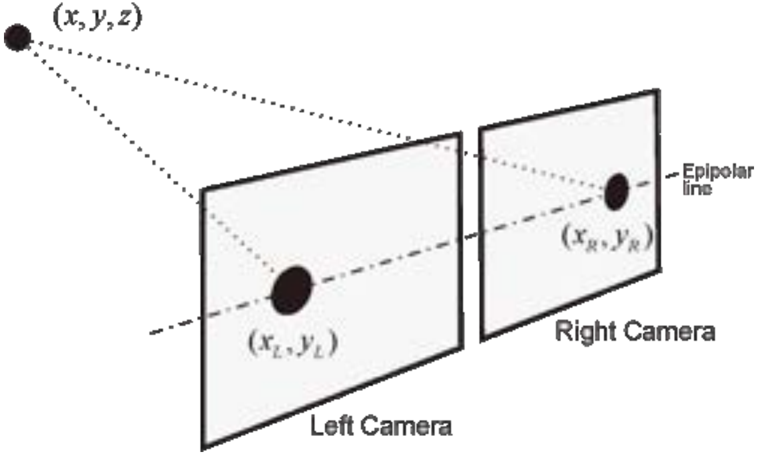
Kinect / RGBD sensors

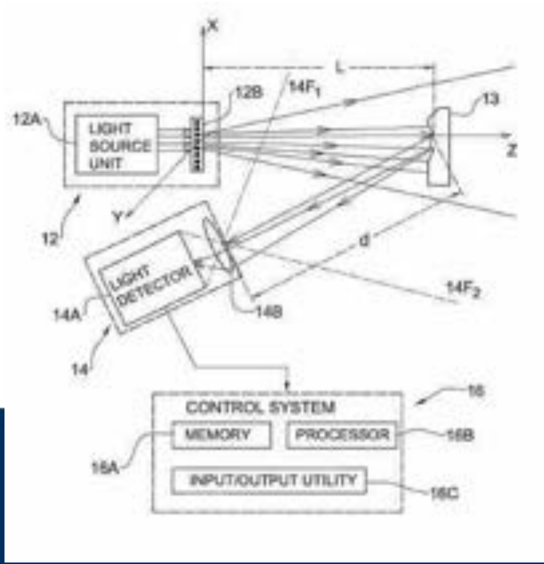
Structured Light - Kinect





Depth from Stereo





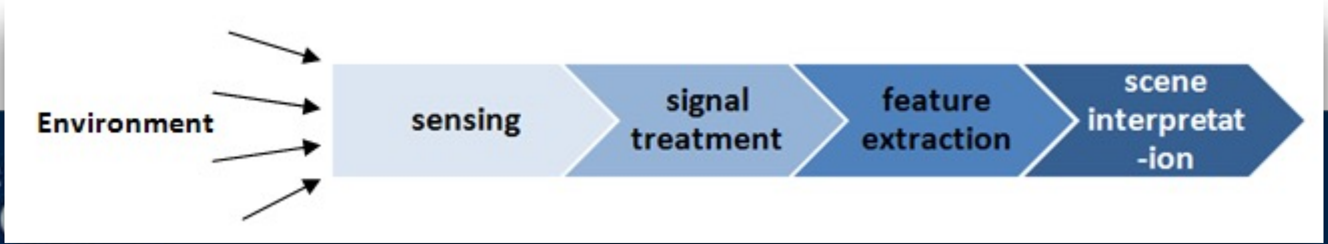
Other Sensors

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

mobile nose

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!

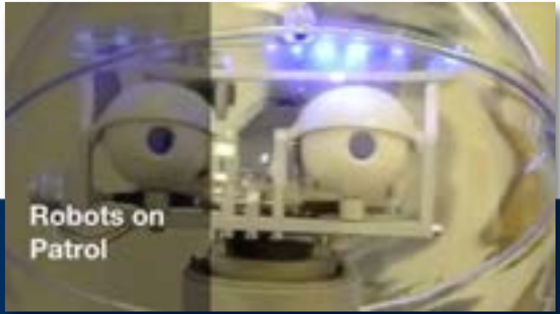


Which robot is more autonomous?

Which one of these robots is more autonomous?

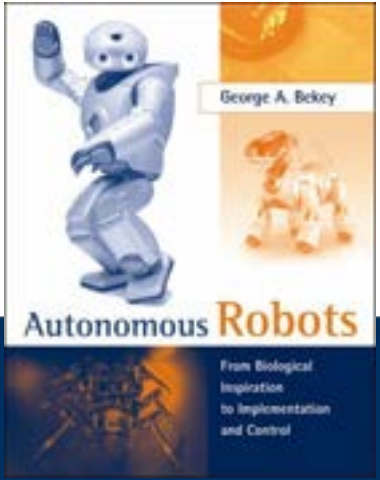
Robotics Research at Lincoln

- 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

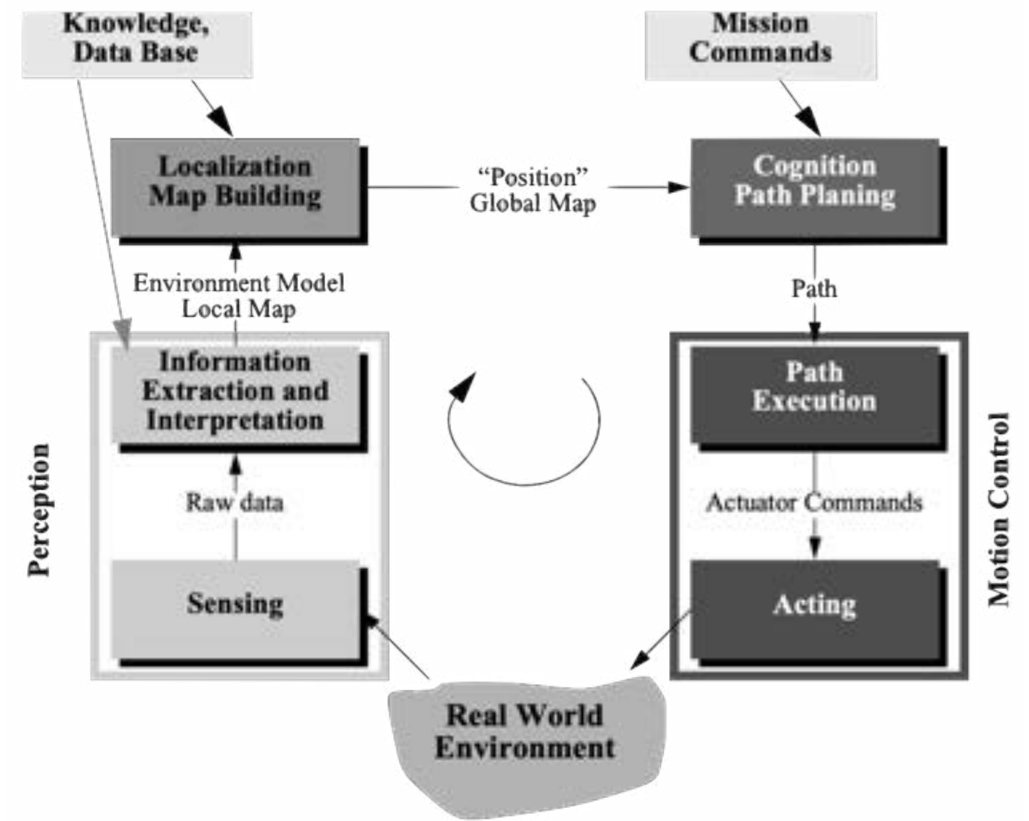


Recommended Reading

- Gates, B. A Robot in Every Home, Scientific American, 2006
- Siegwart et al. Autonomous Mobile Robots, 2004
  (chapter 1, also on blackboard)
- Bekey, G.A. Autonomous robots –
  Chapter 1, also Section 4.1



Summary



Knowledge,
Data Base
Mission
Commands
Localization
Map Building
"Position"
Global Map
Cognition
Path Planing
Environment Model
Local Map
Path
Information
Extraction and
Interpretation
Path
Execution
Perception
Raw data
Actuator Commands
Motion Control
Sensing
Acting
Real World
Environment

LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

UNIVERSITY OF LINCOLN

Introduction to ROS2

With gratitude to Southwest Research Institute and the ROS industrial project

UNIVERSITY OF LINCOLN

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 reimagining of ROS
  - Started in 2014
  - Built on DDS, middleware proven in industry
  - Now on 6th named release

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

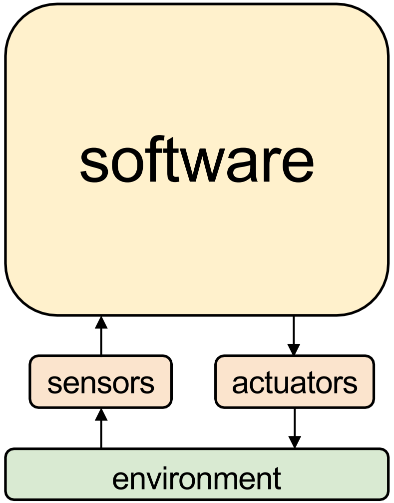
…

Foxy (LTS)

Galactic

Humble

ROS: The Big Picture

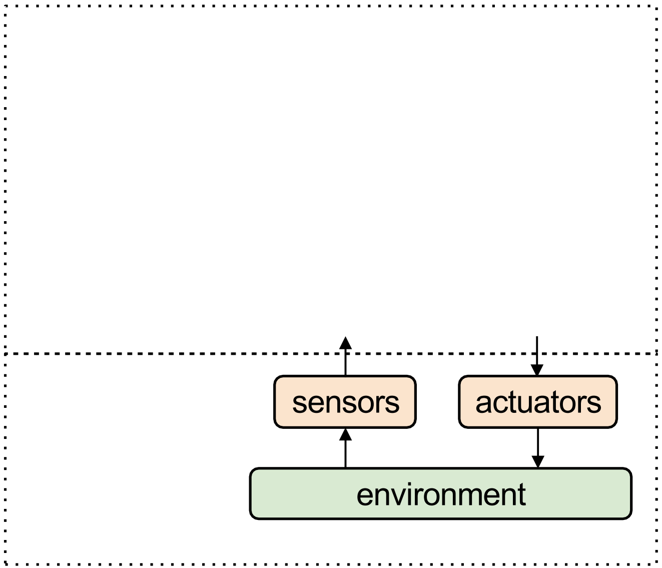


All robots are:

Software connecting Sensors to Actuators
to interact with the Environment

ROS : The Big Picture

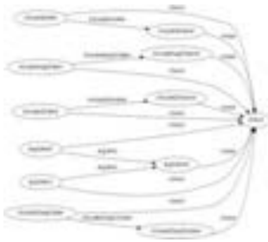
- Break Complex Software into Smaller Pieces
- Provide a framework, tools, and interfaces for distributed development
- Encourage re-use of software pieces
- Easy transition between simulation and hardware



What is ROS?

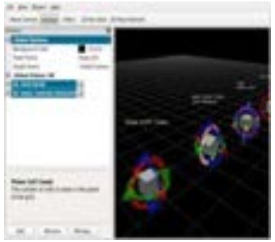
- ROS is…

=



Plumbing

+



Tools

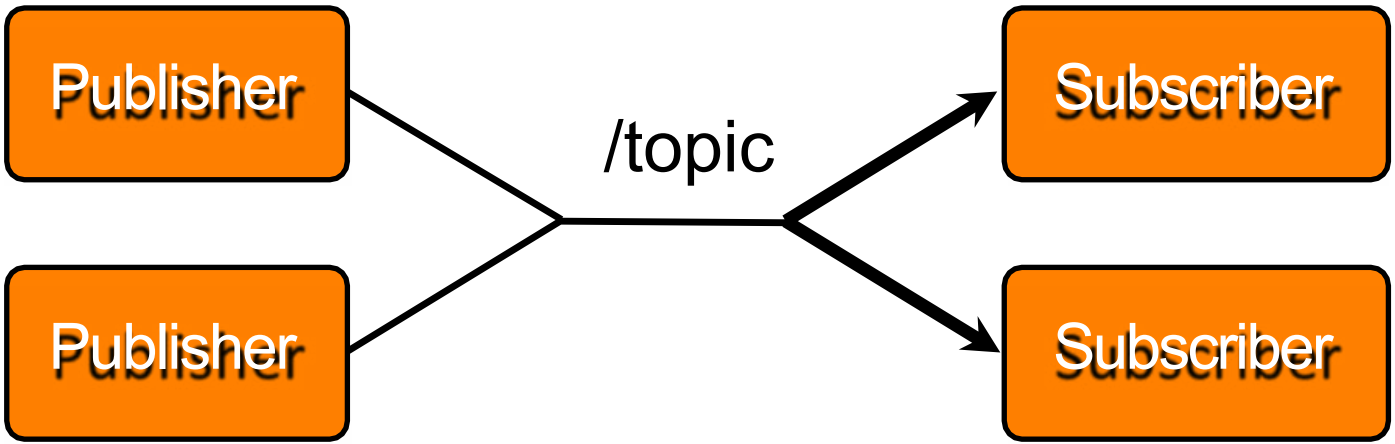
+

Capabilities

+

Ecosystem

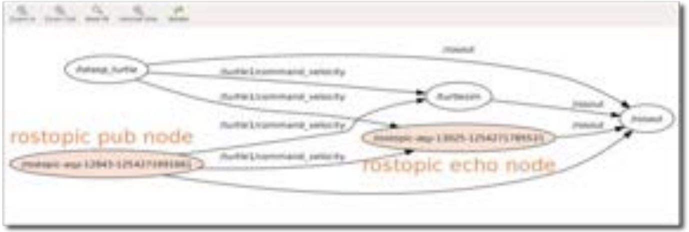
ROS is… plumbing

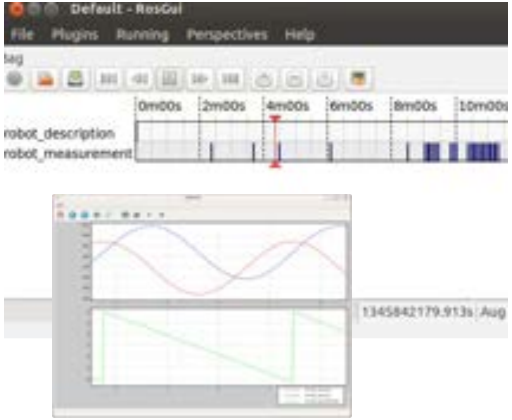


ROS Plumbing : Drivers

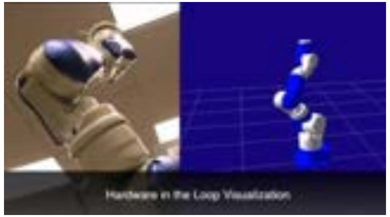
- 2d/3d cameras
- laser scanners
- robot actuators
- inertial units
- audio
- GPS
- joysticks
- etc.

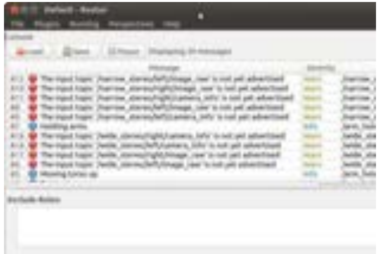
ROS is …Tools



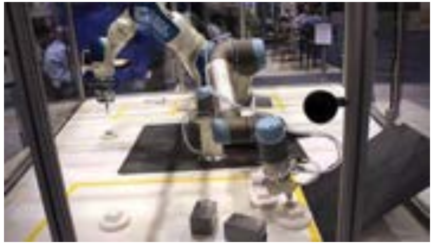


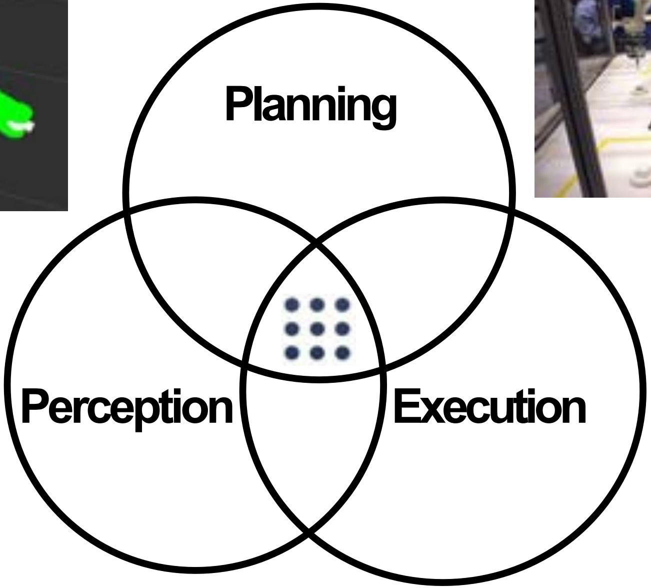
- logging/plotting
- graph visualization
- diagnostics
- visualization





ROS is…Capabilities

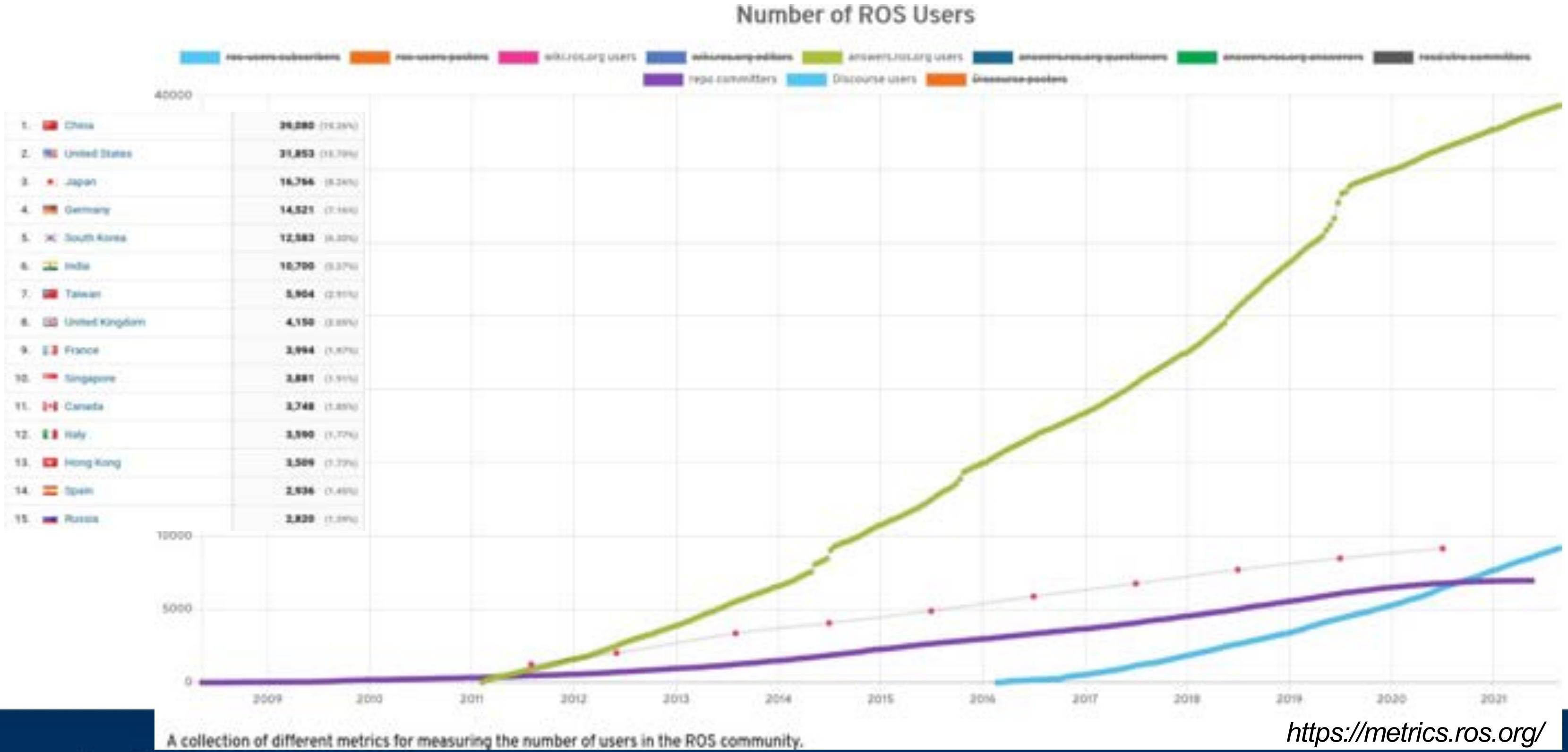




(Adapted from Willow Garage's "What is ROS?" Presentation)

ROS is… an Ecosystem

ROS is a growing Ecosystem



https://metrics.ros.org/

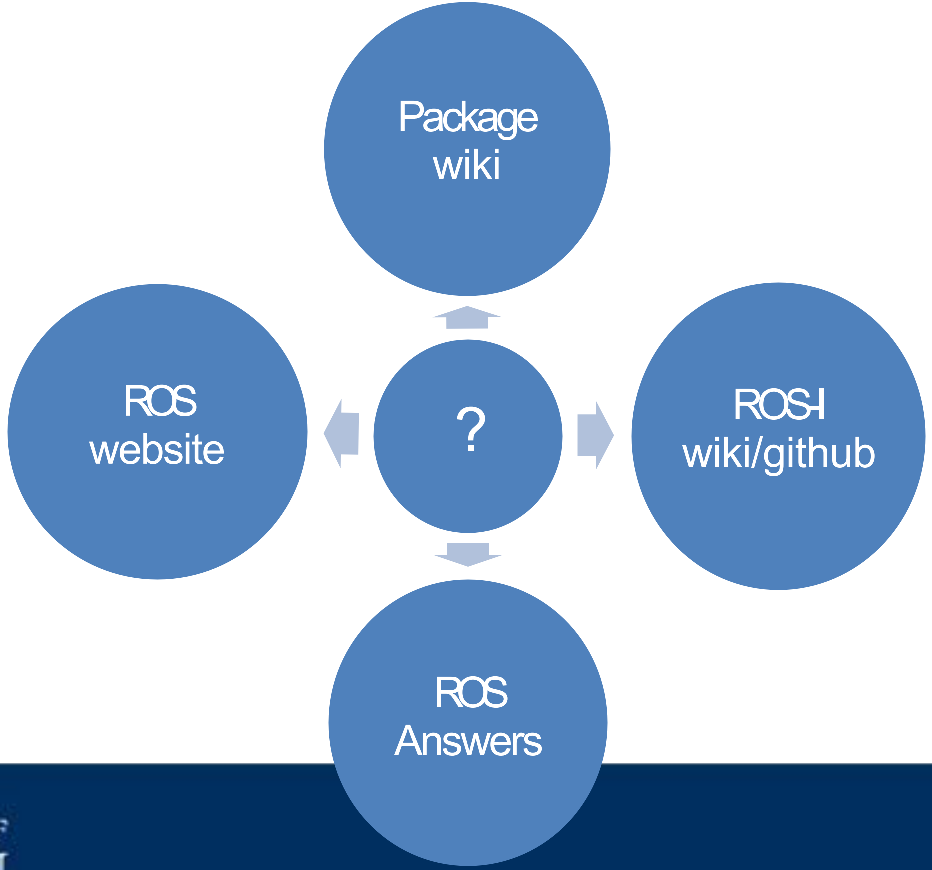
UNIVERSITY OF LINCOLN

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

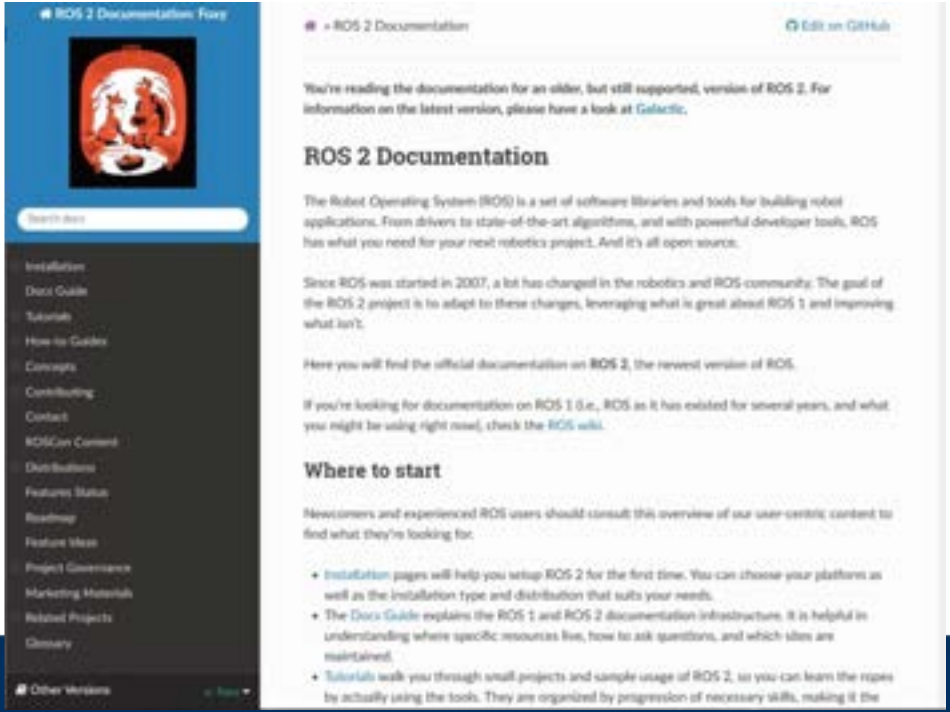


ROS2 Documentation

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

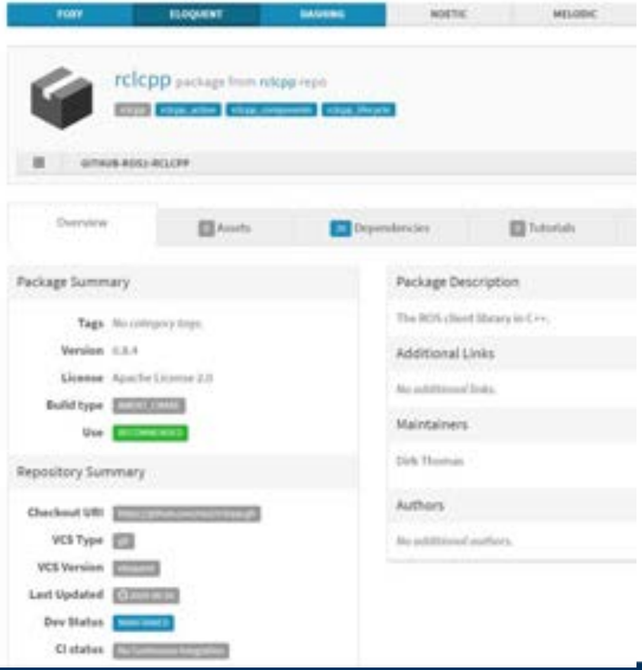
- Install
- Tutorials
- Concepts

…



ROS Package Index

http://index.ros.org



- 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

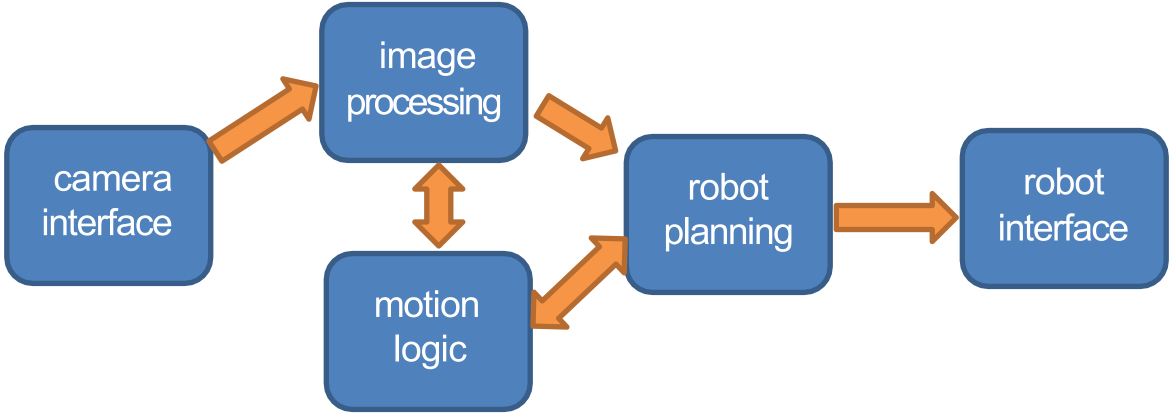
ROS Consortium can help fill that need

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

Thank you for listening!

Any questions ?

Thank you for listening!
Any questions ?


When survey is active, respond at **pollev.com/mhanheide**

End of Lecture Feedback

0 done

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**


https://attendance.lincoln.ac.uk/
UNIVERSITY OF LINCOLN
CMP3103 – AUTONOMOUS MOBILE ROBOTS
Lincoln Centre for Autonomous Systems
School of Computer Science
Access Code: 039313
L-CAS

When poll is active, respond at **pollev.com/mhanheide**

Content quality (1=really bad, 5=really good)

1

2

3

4

5

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

When poll is active, respond at **pollev.com/mhanheide**

Delivery quality (1=really bad, 5=really good)

1

2

3

4

5

What to keep?

Join by Web

1. Go to **PollEv.com**
2. Enter **MHANHEIDE**
3. Respond to activity

Instructions not active. **Log in** to activate

What to stop?

Join by Web

1. Go to **PollEv.com**
2. Enter **MHANHEIDE**
3. Respond to activity

Instructions not active. **Log in** to activate

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Which robot is more autonomous?

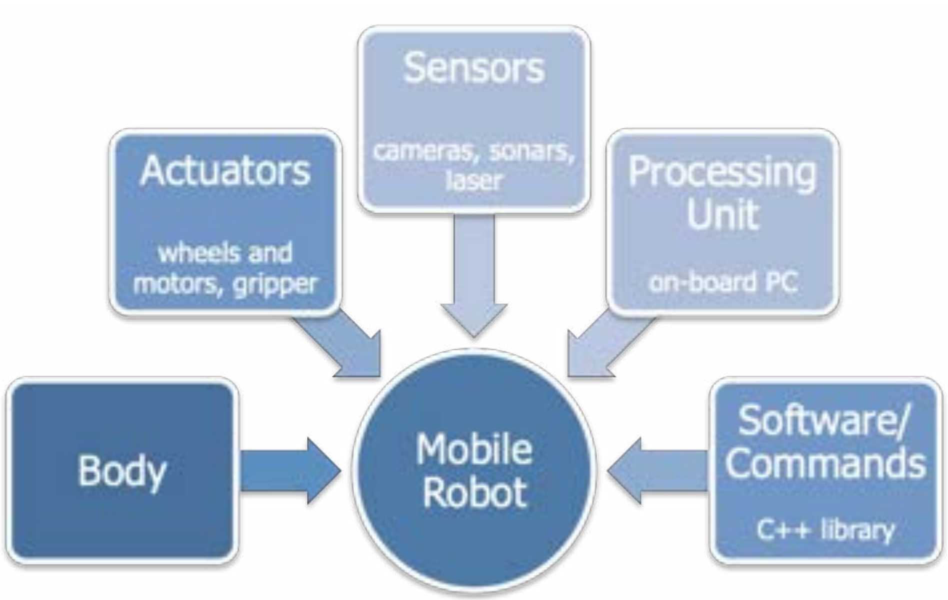
Which one of these robots is more autonomous?

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

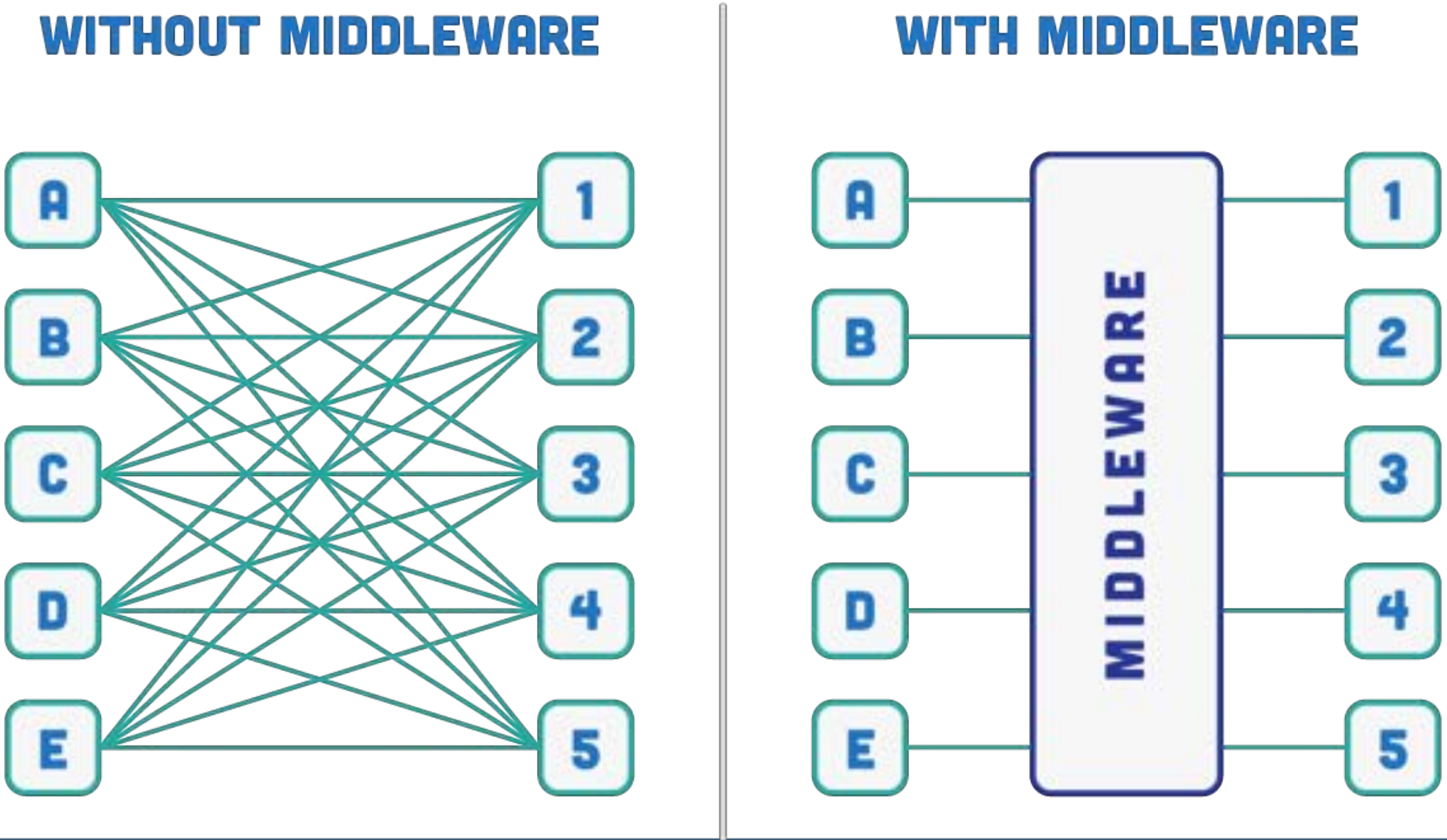
Robot Software

- 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 Turtlebots are more modular, different sensors talk via USB
  - the have their dedicated ROS driver nodes to talk to us
- HENCE, we need to able to **communicate**!

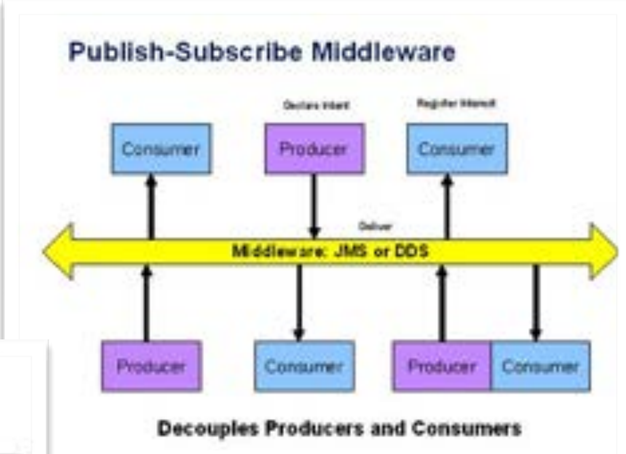
Middleware?

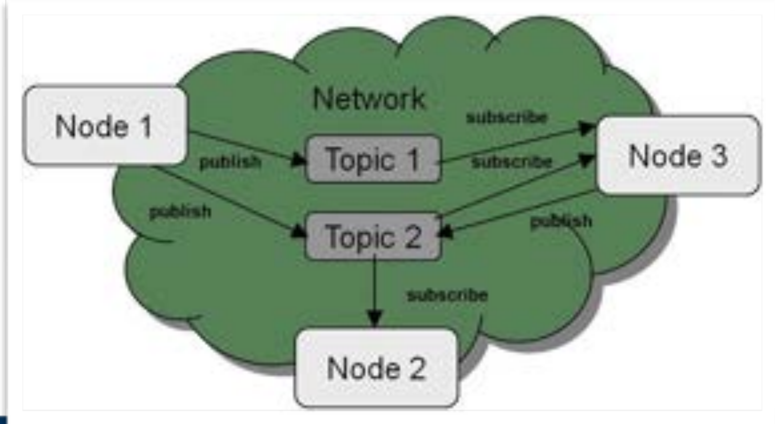


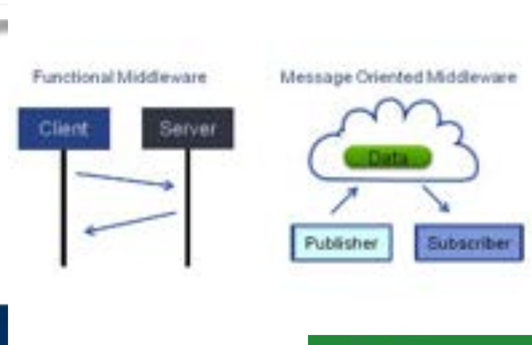
WITHOUT MIDDLEWARE
A
B
C
D
E
1
2
3
4
5
WITH MIDDLEWARE
A
B
C
D
E
MIDDLEWARE
1
2
3
4
5

Middleware?

Middleware supports and simplifies complex distributed applications.







Robotic Middlewares

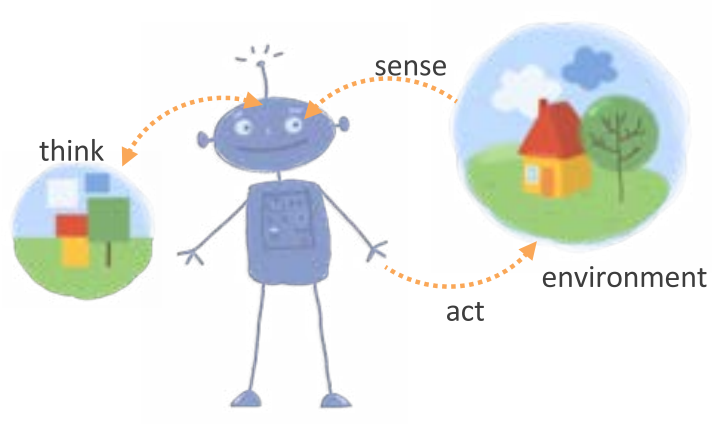
- Support for different robots, platforms, unified interface, plug-ins/modules (e.g. navigation, object recognition), simulator, etc.
  - **Robot Operating System (ROS)**
  - Microsoft Robotics Developer Studio for Windows
  - OROCOS
  - YARP
  - RSB
  - …

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 for ROS2)

Introduction to ROS2

With gratitude to Southwest Research Institute and the ROS industrial project

L-CAS

UNIVERSITY OF LINCOLN

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 reimagining of ROS
  - Started in 2014
  - Built on DDS, middleware proven in industry
  - Now on 6th named release

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

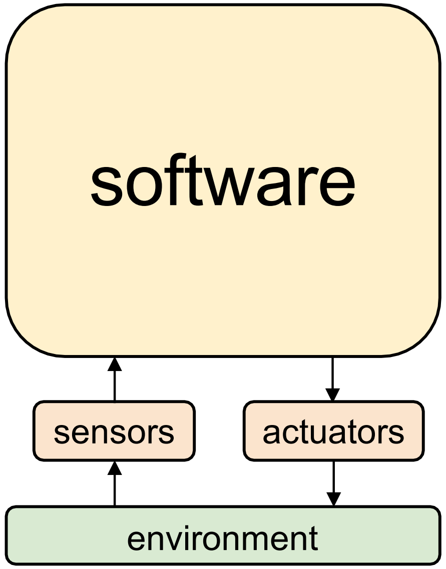
…

Foxy (LTS)

Galactic

Humble

ROS: The Big Picture

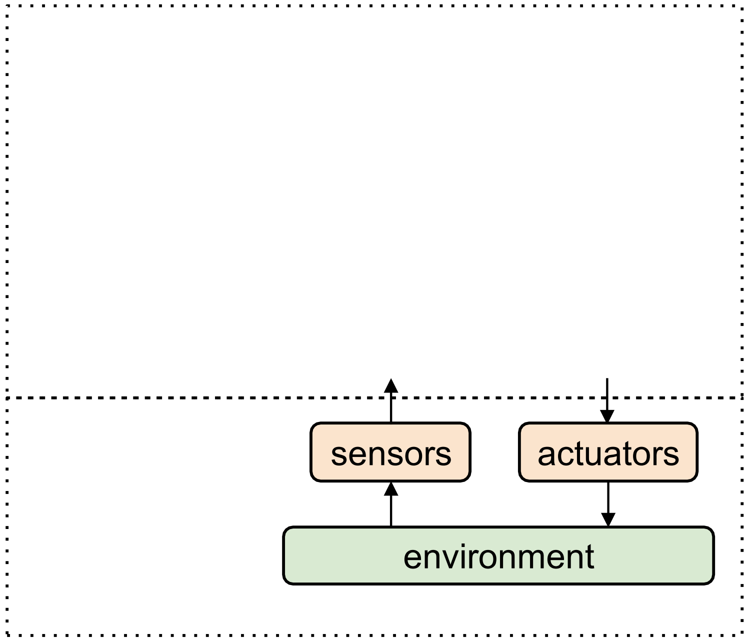


All robots are:

Software connecting Sensors to Actuators
to interact with the Environment

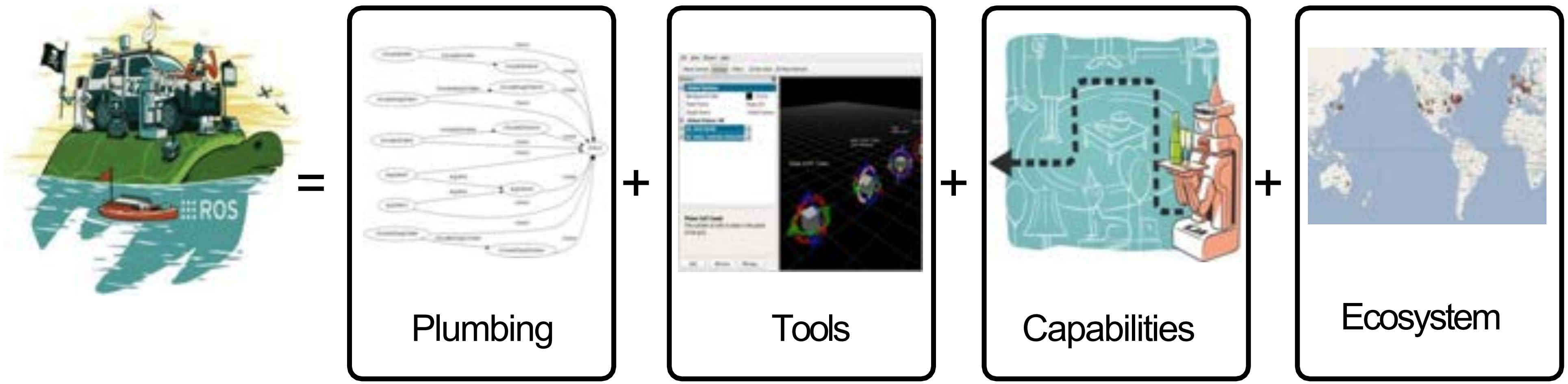
ROS : The Big Picture

- 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

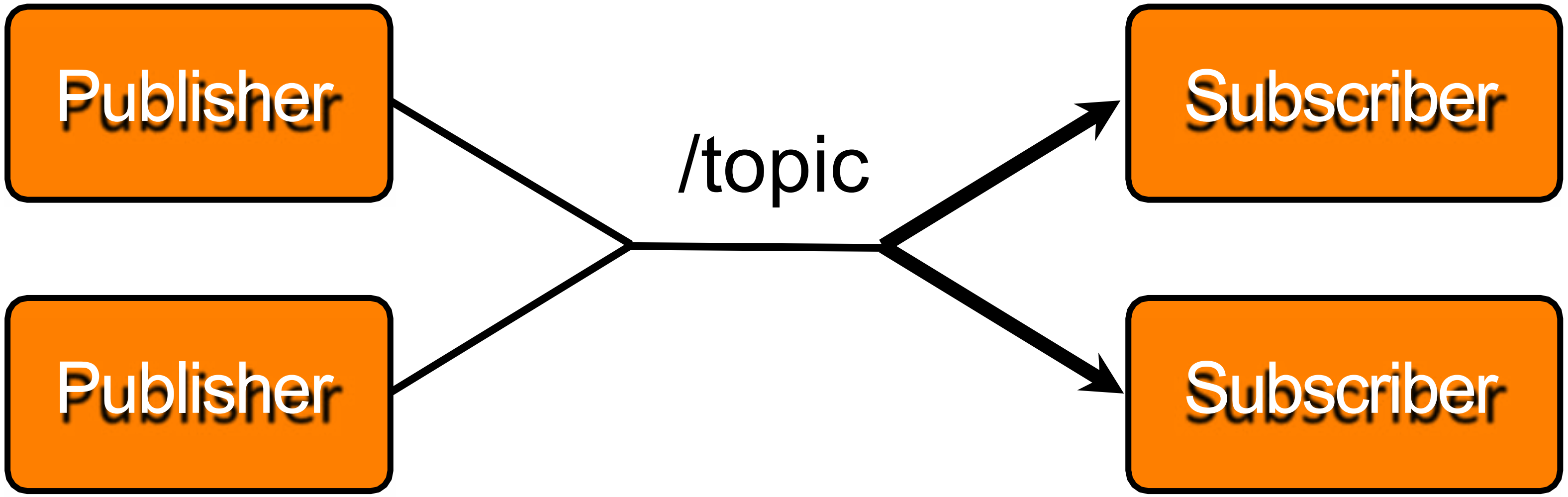


What is ROS?

- ROS is…



ROS is… plumbing

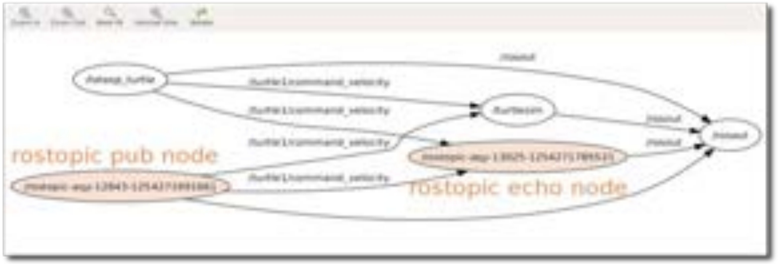


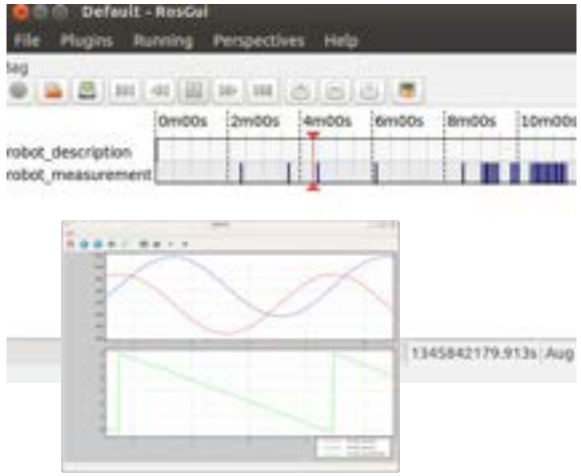
ROS Plumbing : Drivers

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

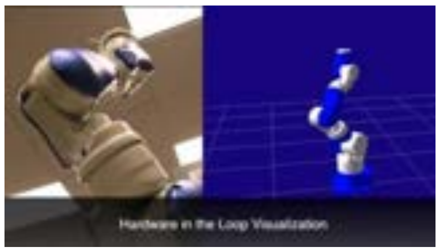
UNIVERSITY OF LINCOLN

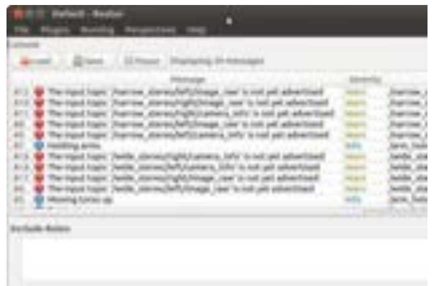
ROS is …Tools



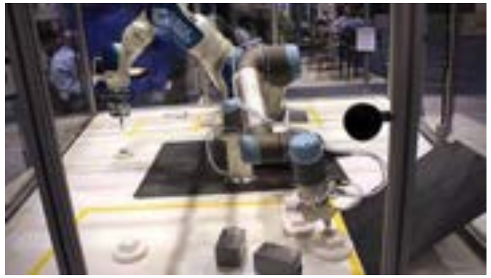


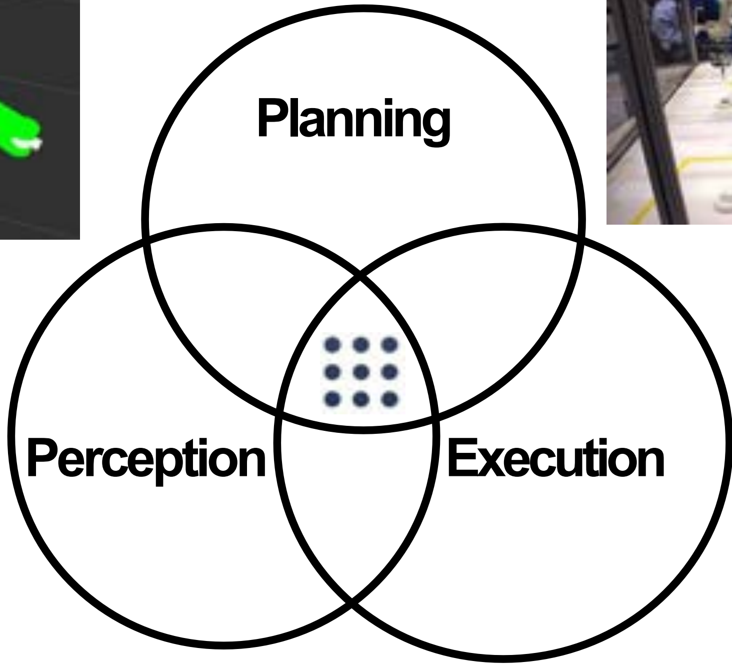
- logging/plotting
- graph visualization
- diagnostics
- visualization





ROS is…Capabilities

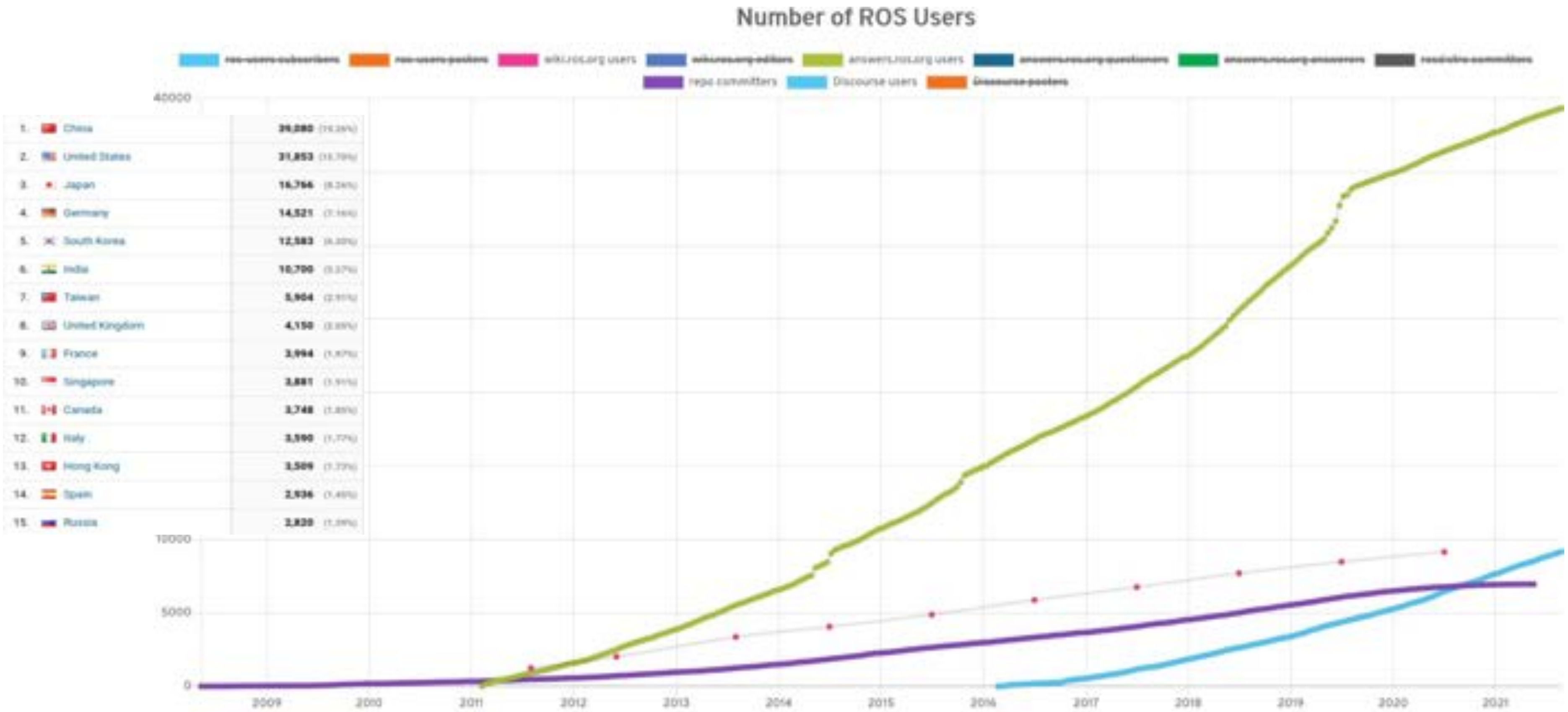




(Adapted from Willow Garage's "What is ROS?" Presentation)

ROS is… an Ecosystem

ROS is a growing Ecosystem



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

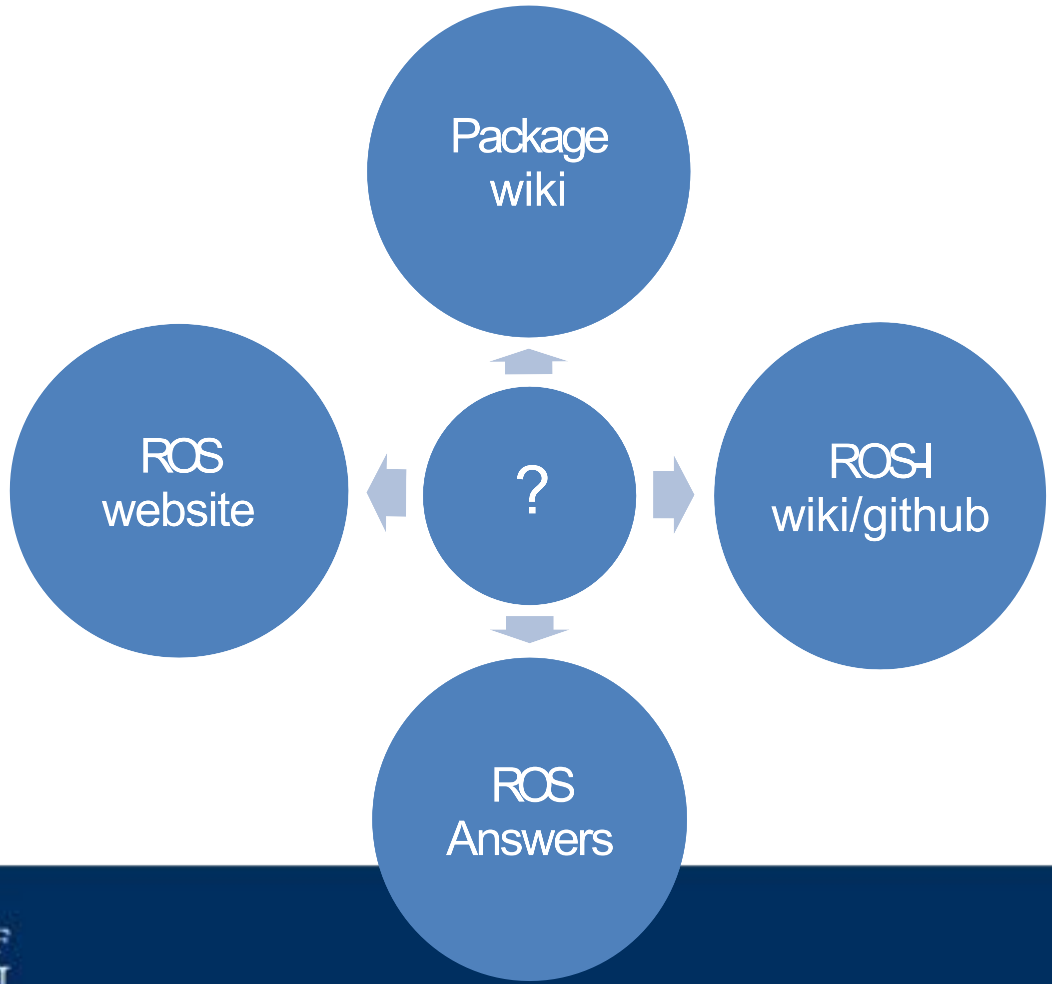
https://metrics.ros.org/

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

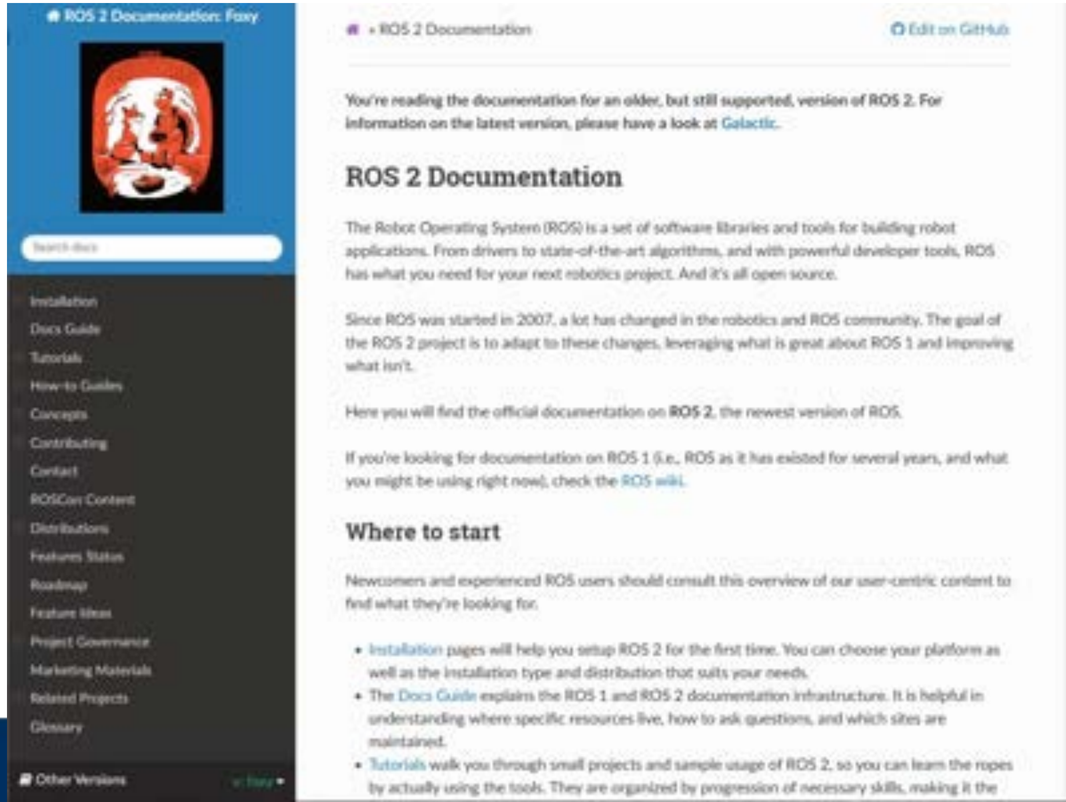


ROS2 Documentation

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

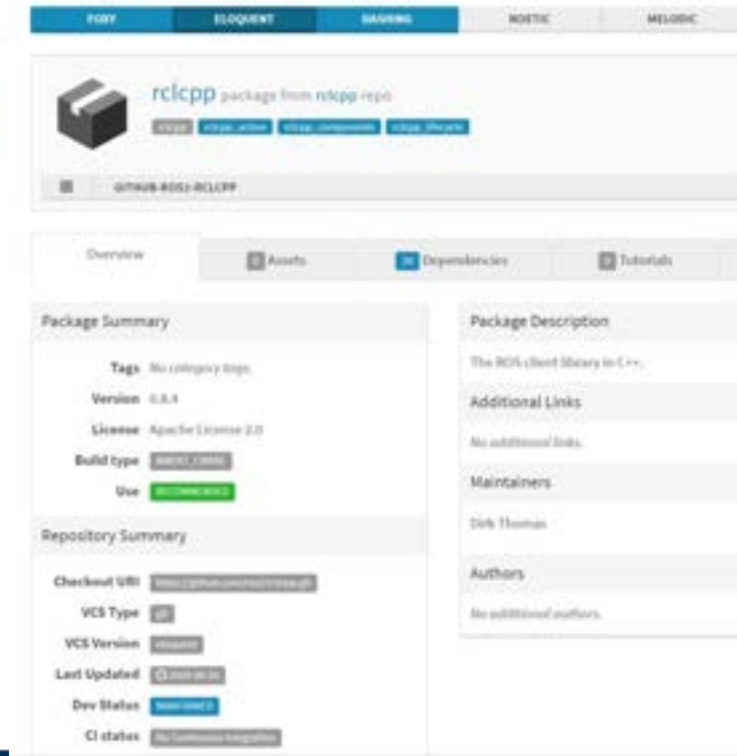
- Install
- Tutorials
- Concepts

…



ROS Package Index

http://index.ros.org



- 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

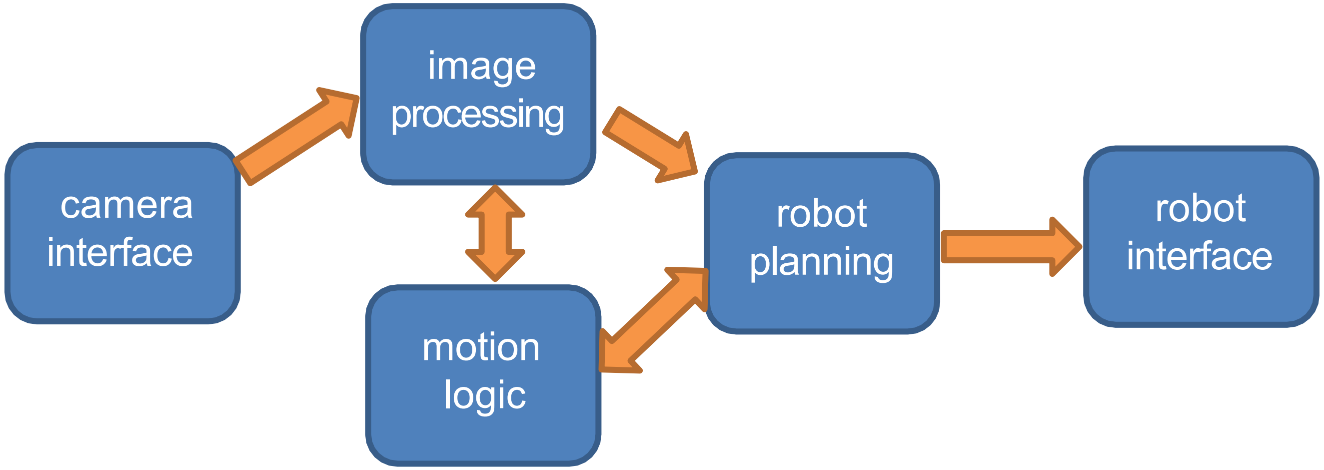
ROS Consortium can help fill that need

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 progamme 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
- 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`

```python
#!/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

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)

```python
class Chatter(Node):
    """ a simple "chatter" that publishes String messages on a topic.

        Once this is running, you can use `ros2 topic echo /msgs` 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 "/msgs"
        # the third argument is the length of the queue, i.e., only the last message is queued here
        self.publisher = self.create_publisher(String, '/msgs', 1)
        timer_period = 1  # 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

```python
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

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

```python
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

```python
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

Thank you for listening!
Any questions ?

Thank you for listening!

Any questions ?

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

UNIVERSITY OF LINCOLN

CMP3103 – AUTONOMOUS MOBILE ROBOTS

Lincoln Centre for Autonomous Systems

School of Computer Science

L-CAS

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!

Respond at **pollev.com/jonathancox637**

Which middleware paradigm do ROS topic implement?

Data pull

Client-server

Synchronous processing

Publish-subscribe

Broadcast

Powered by Poll Everywhere

What is the order of processing events in this ROS code?

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
import math

class Reciever(Node):

    def __init__(self):
        super().__init__('reciever')
        self.laser_sub = self.create_subscription(
            LaserScan,"/scan", self.callback, 1)
        self.twist_pub = self.create_publisher(
            Twist, '/cmd_vel', 1)

    def callback(self, data):
        if data.ranges[0] < 1.0:
            twist = Twist()
            twist.angular.z = 0.3
            self.twist_pub.publish(twist)

def main(args=None):
    rclpy.init()
    node = Reciever()
    node.destroy_node()

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

The robot turns

The robot's laser scanner completes a scan of the environment and publishes the data

The callback of the subscriber is called

The node is initialised with the name 'reciever'

A message is published through the publisher

A new twist topic is created

A publisher is created, announcing to publish data on 'cmd_vel' of type 'Twist'.

What is wrong with this code?

```python
import rclpy

from rclpy.node import Node

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

class FirstSub(Node):
    def __init__(self):
        super().__init__('firstsub')

        self.sub = self.create_subscription(
            Odometry,"/scan",
            self.callback, 1)

        self.pub = self.create_publisher(String, '/warning', 1)

    def callback(self, data):
        for range in data.ranges:
            if range < 1.0:
                print("ALERT")
                str = String()
                str.data = "ALERT"
                self.pub.publish(str)

def main(args=None):
    rclpy.init()
    node = FirstSub()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

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

- Publisher has no topic to publish to
- Syntax error in for-loop
- Wrong topic type
- Wrong import statement
- Callback not correctly registered
- Publish called with wrong data

What is wrong with this code?

```python
import rclpy

from rclpy.node import Node

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

class FirstSub(Node):
    def __init__(self):
        super().__init__('firstsub')

        self.sub = self.create_subscription(
            Odometry,"/scan",
            self.callback, 1)

        self.pub = self.create_publisher(String, '/warning', 1)

    def callback(self, data):
        for range in data.ranges:
            if range < 1.0:
                print("ALERT")
                str = String()
                str.data = "ALERT"
                self.pub.publish(str)

def main(args=None):
    rclpy.init()
    node = FirstSub()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

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

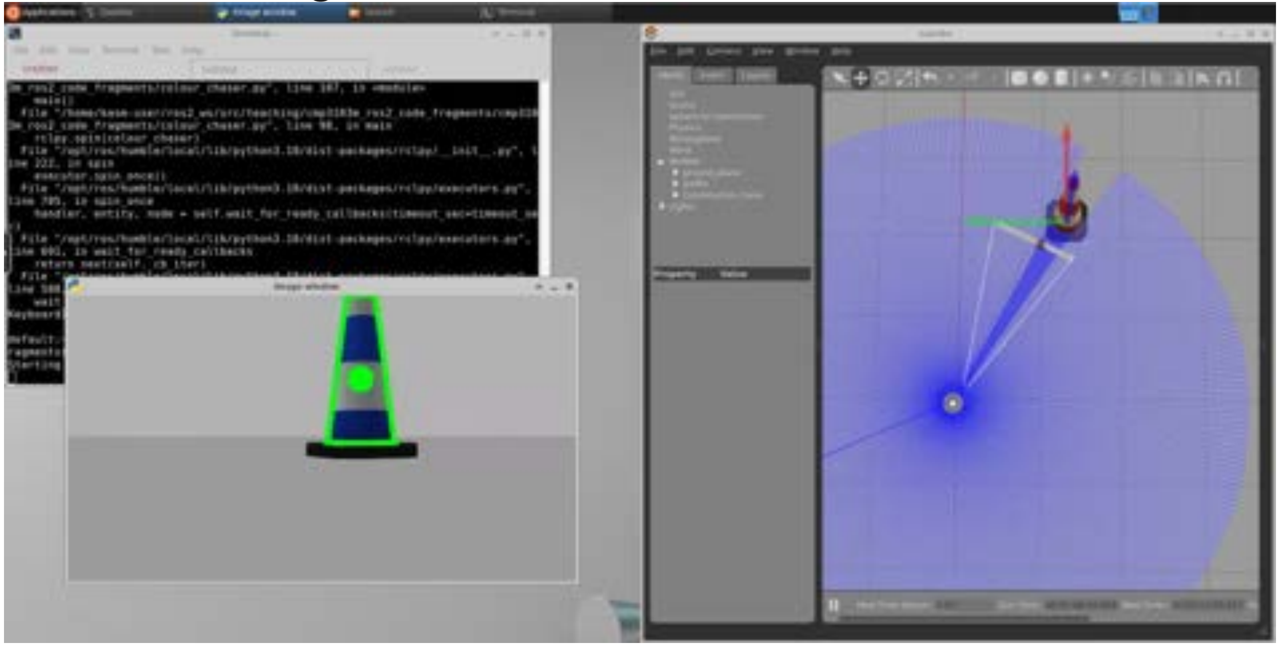
```python
import rclpy

from rclpy.node import Node

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

class FirstSub(Node):
    def __init__(self):
        super().__init__('firstsub')

        self.sub = self.create_subscription(
            LaserScan,"/scan",
            self.callback, 1)

        self.pub = self.create_publisher(String, '/warning', 1)

    def callback(self, data):
        for range in data.ranges:
            if range < 1.0:
                print("ALERT")
                str = String()
                str.data = "ALERT"
                self.pub.publish(str)

def main(args=None):
    rclpy.init()
    node = FirstSub()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    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

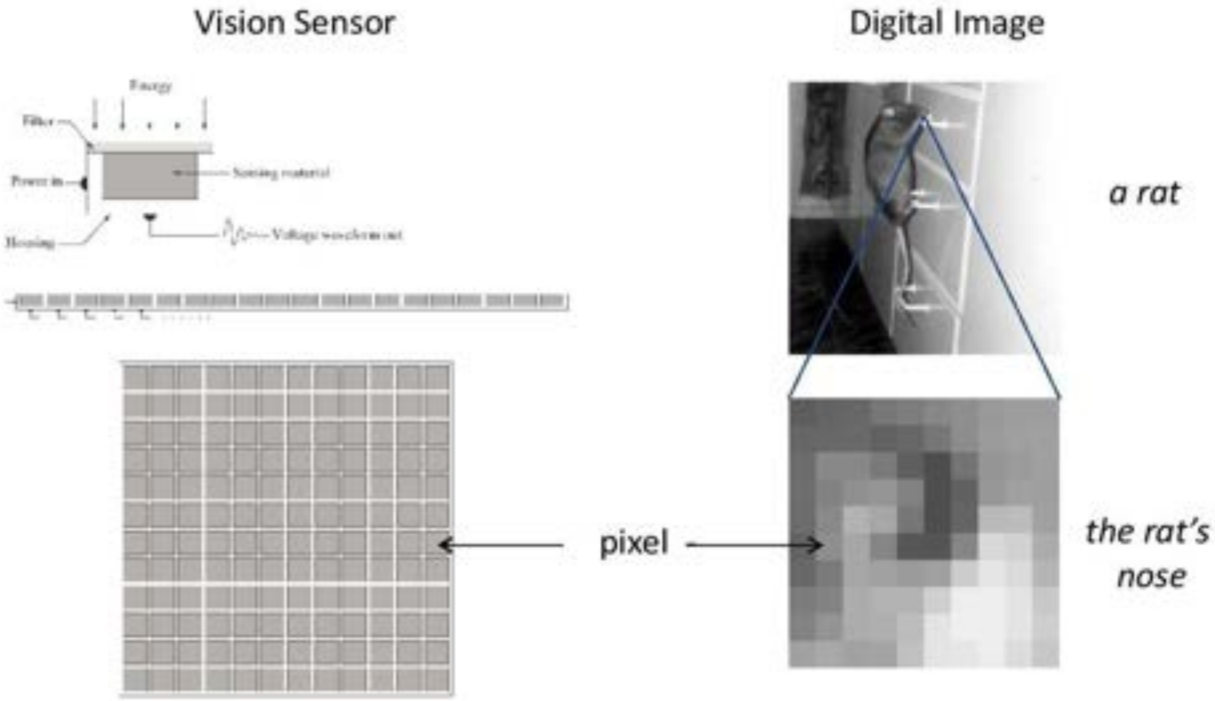
Steps

- 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

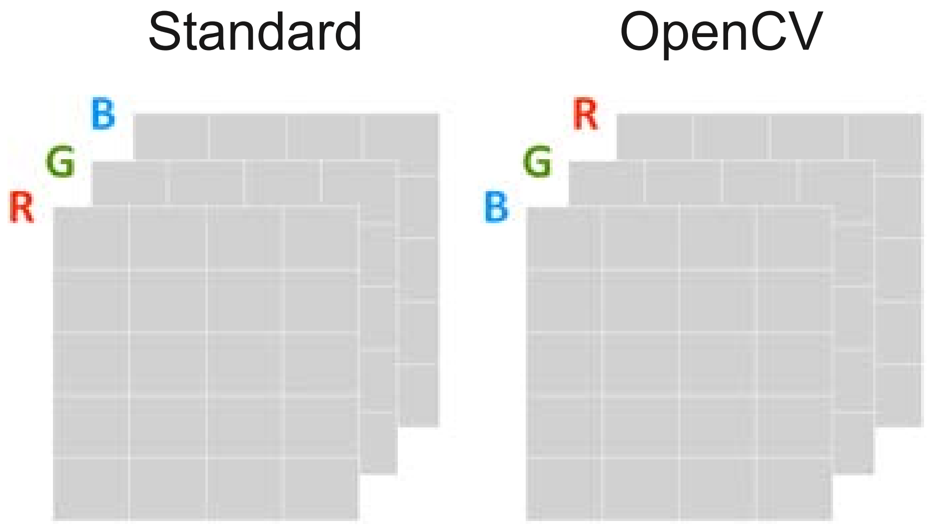
- Initially launched by Intel in 1999
- Developed into the “gold standard” in computer vision processing
- Cross-platform, multi-language
- Applications:
  - 2D and 3D feature toolkits
  - Egomotion estimation
  - Facial recognition system
  - Gesture recognition
  - Human–computer interaction (HCI)
- **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!?



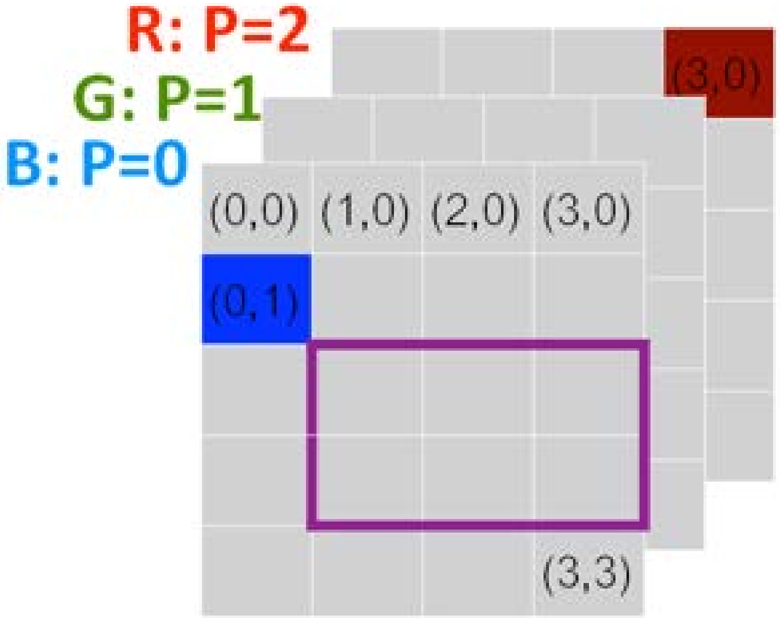
Basic operations in OpenCV

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

```python
import cv2
import numpy as np

# declare windows you want to display
cv2.namedWindow("original")
cv2.namedWindow("blur")
cv2.namedWindow("canny")

img = cv2.imread('blofeld.jpg')
print('type: %s' % type(img))
cv2.imshow("original", img)

# create a new blurred image:
blur_img = cv2.blur(img, (7, 7))

# draw on the image:
cv2.circle(blur_img, (100, 100), 30, (255, 0, 255), 5)

# display the image:
cv2.imshow("blur", blur_img)

# Canny is an algorith for edge detection
canny_img = cv2.Canny(img, 10, 200)
cv2.imshow("canny", canny_img)
print('shape: %s' % str(canny_img.shape))

# the shape gives you the dimensions
h = canny_img.shape[0]  # height
w = canny_img.shape[1]  # width

# loop over the image, pixel by pixel
count = 0
# a slow way to iterate over the pixels
for y in range(0, h):
    for x in range(0, w):
        # threshold the pixel
        if canny_img[y, x] > 0:
            count += 1
print('count edge pixels: %d' % count)

# a fast way to iterate using numpy:
count = np.sum(canny_img > 0)
print('faster count edge pixels: %d' % count)

cv2.waitKey(0)
# good practice to tidy up at the end
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

```
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')
```

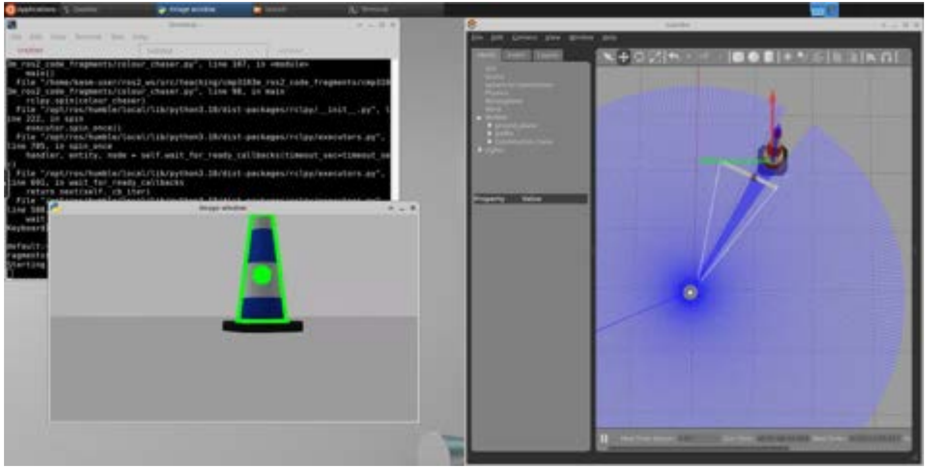
opencv_bridge.py

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

```python
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_small))

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

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

        cv2.imshow("Image window", cv_image_small)
        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()
```

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

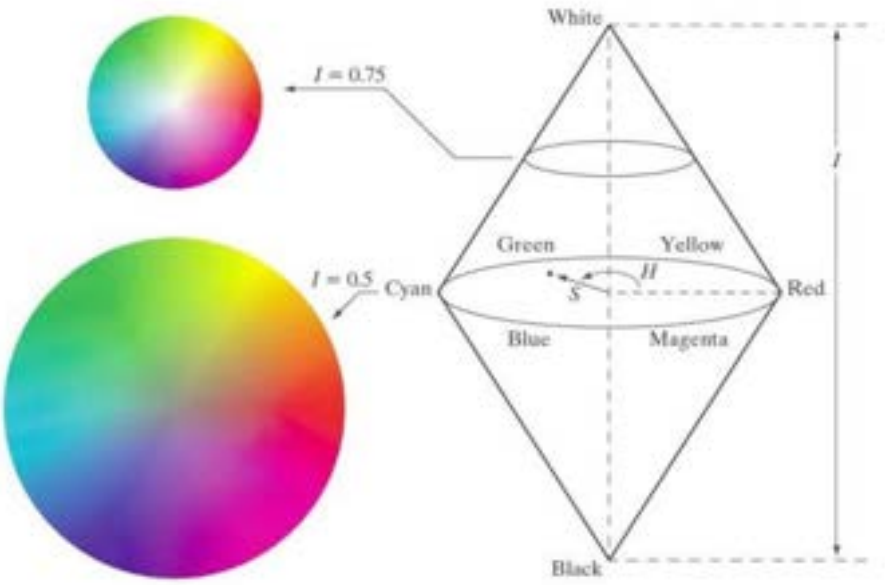
Common models and their applications

- 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)!

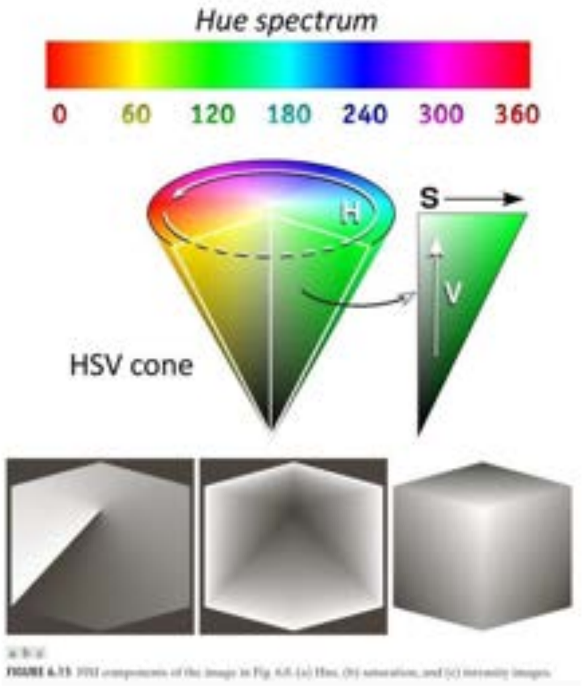


HSI (Hue, Saturation, Intensity) Model

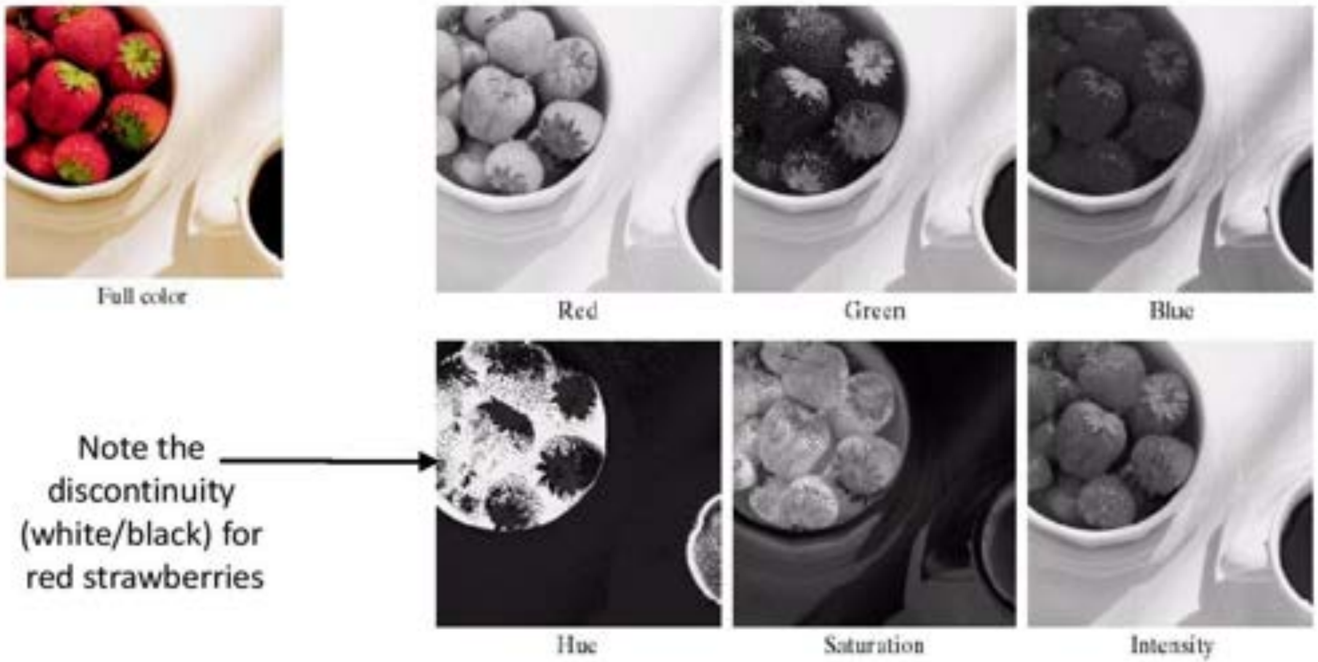
Hue is expressed as an angle

- 0°/360° - Red
- 120° - Green
- 240° - Blue

! Take Care to check for the discontinuity 0°/360° around red when processing HIS images!

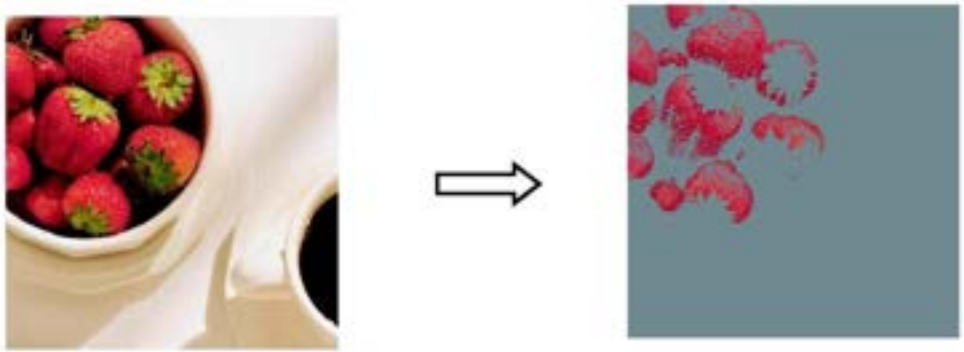


Colour Models Examples



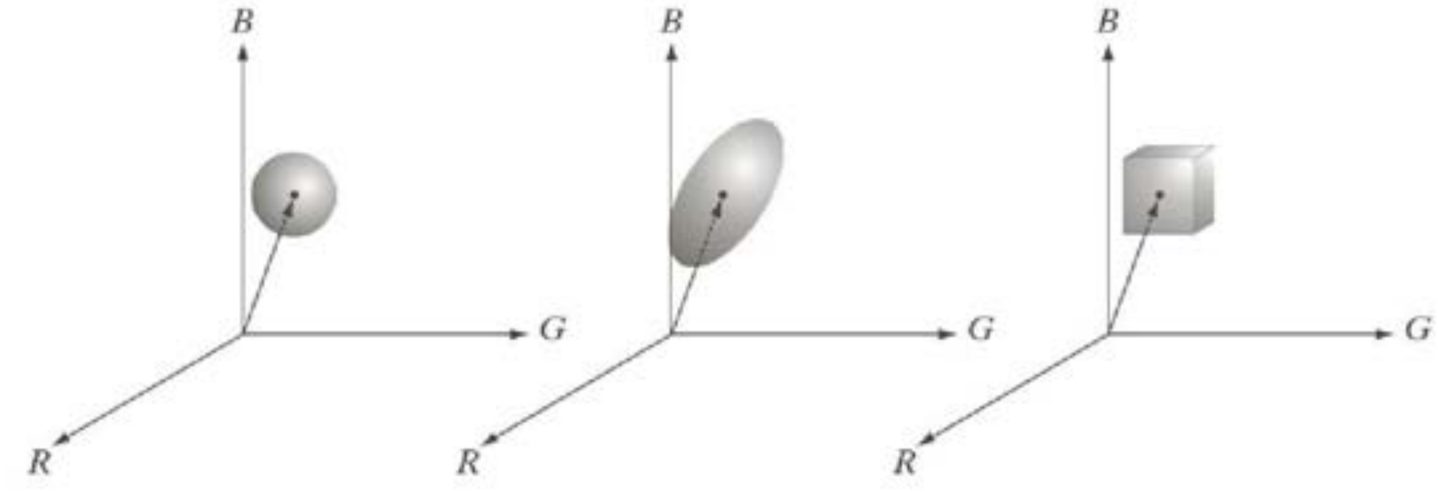
Colour Slicing

- 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.)

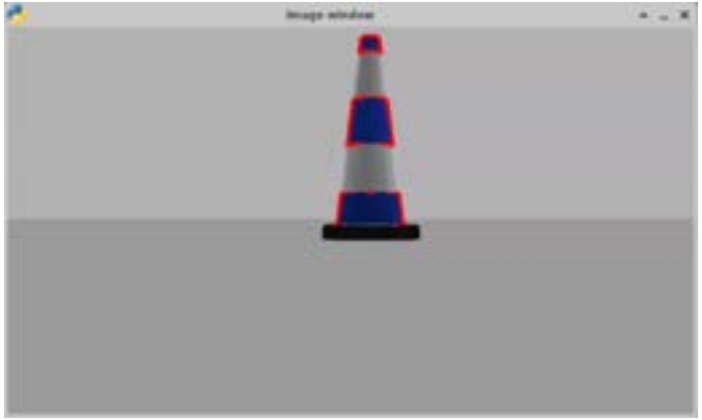
Colour Slicing in Python

```
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

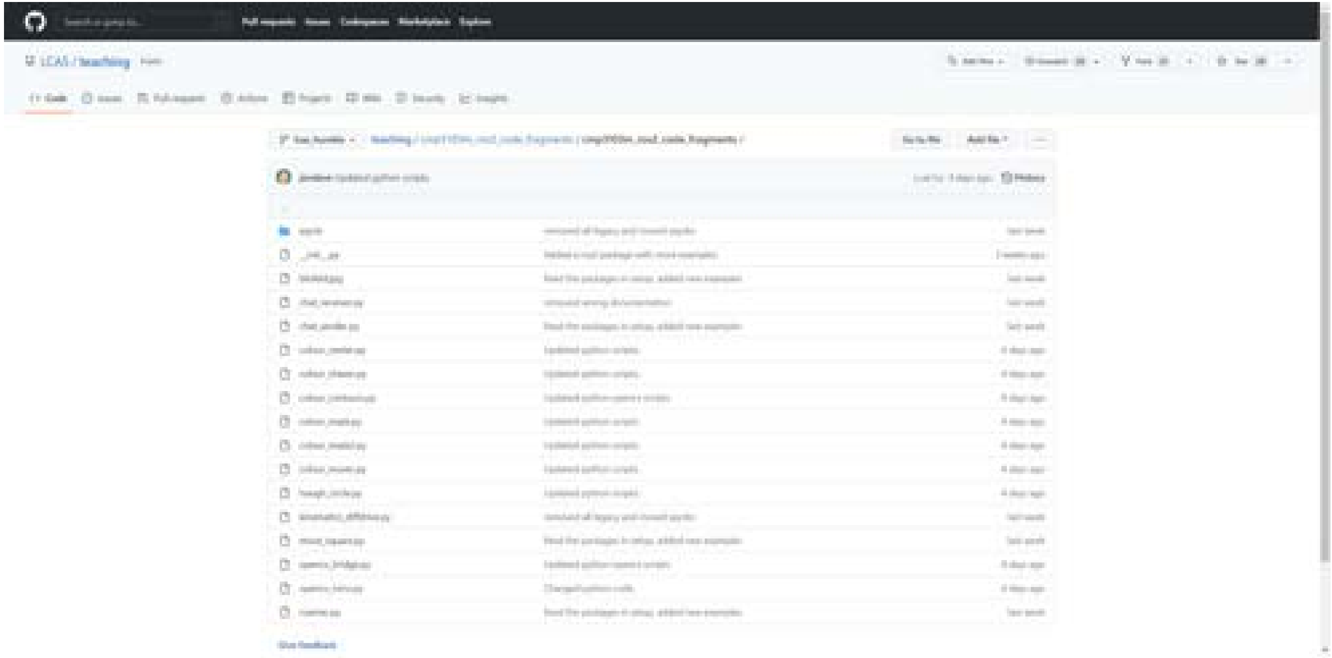
```python
class ColourChaser(Node):
    def __init__(self):
        super().__init__('colour_chaser')

        # 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, contour number -1 to draw all contours, colour, thickness):
current_frame_contours = cv2.drawContours(current_frame, contours, 0, (0, 255, 0), 20)

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

if len(contours) > 0:
    M = cv2.moments(contours[0]) # only select the largest controur
    if M['m00'] > 0:
        # find the centroid of the contour
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/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 ros2 topic echo /camera/image_raw height: 1080 width: 1920

        # if center of object is to the left of image center move right
        if cx < 900:
            self.tw.angular.z=0.3
        # else if center of object is to the right of image center move left
        elif cx >= 1200:
            self.tw.angular.z=-0.3
        else: # center of object is in a 100 px range in the center of the image so dont 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=0.4, fy=0.4) # 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.

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

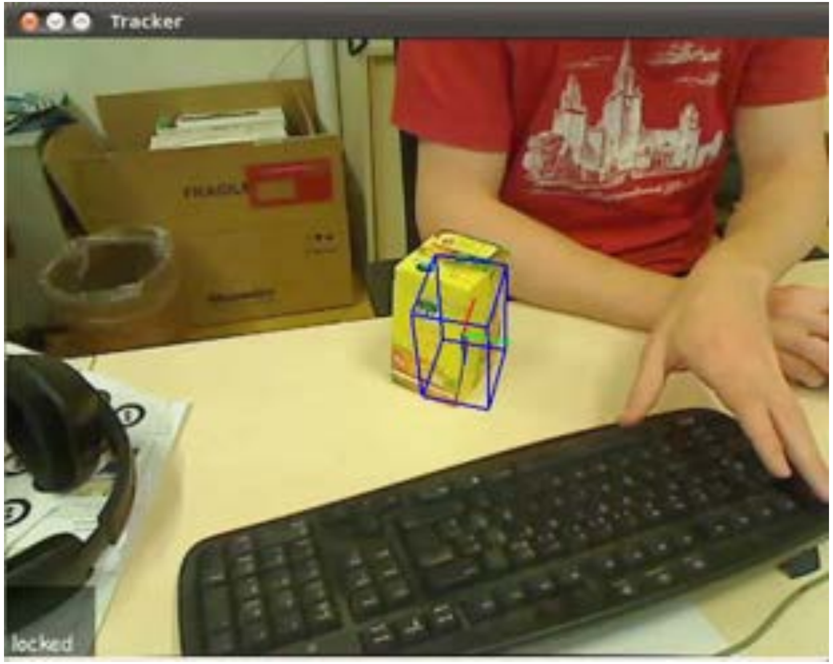
Shahram Izadi 1, Richard Newcombe 2, David Kim 1,3, Otmar Hilliges 1,
David Molyneaux 1,4, Pushmeet Kohli 1, Jamie Shotton 1,
Steve Hodges 1, Dustin Freeman 5, Andrew Davison 2, Andrew Fitzgibbon 1

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 ?

CMP 3103 AMR

Robot Control

•••

Dr. Athanasios Polydoros

Outline

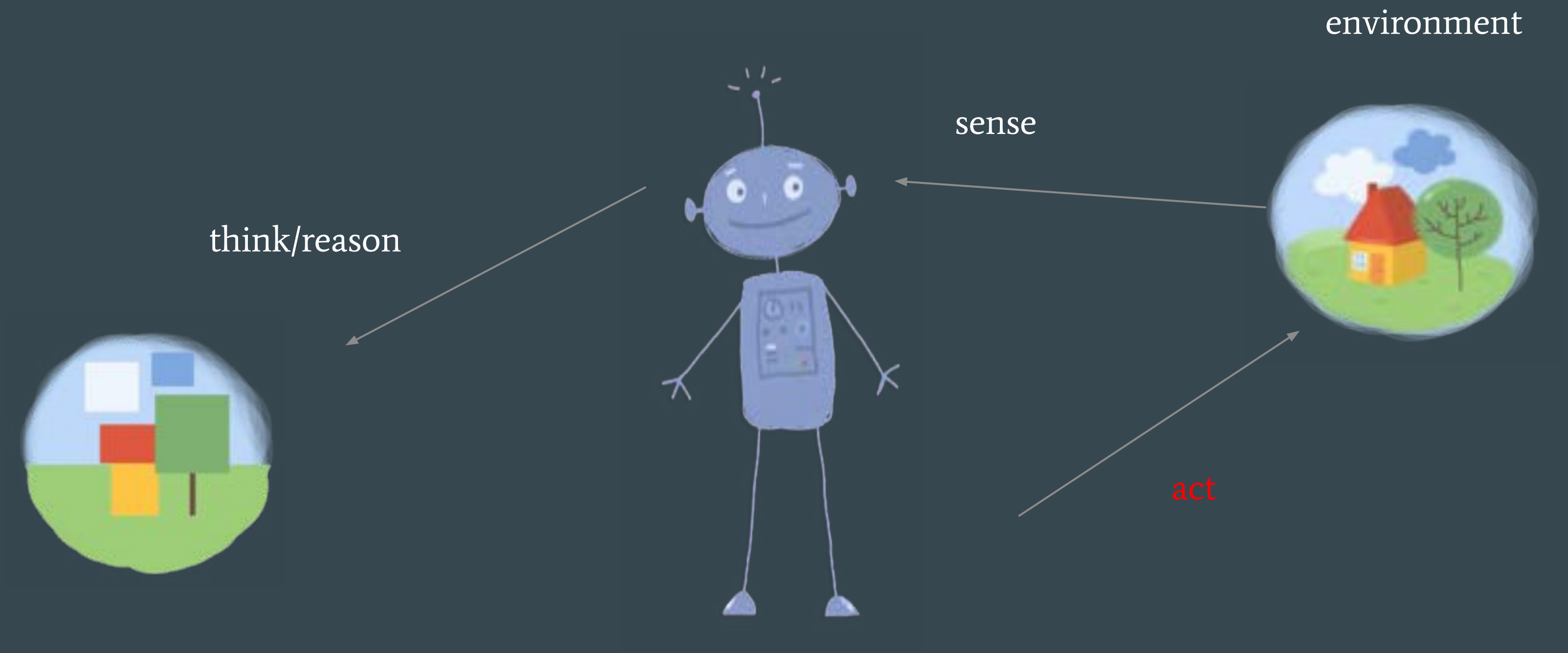
- Coding question
- Movement Commands
- Motion Control Models
  - Inverse and Forward Models
- Algorithms for robot control
  - Open/Closed loop
  - Bang-Bang Controller
  - PID controller
- Application Examples

Coding question

```python
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()
```

- What is the functionality of the code?
- Where there exists a mistake?

Movement Commands – Robotic Agent



Movement Commands – Types

Move in a desired way

- Position Control
  - Move to specific goal
  - Follow a desired path
- Behaviours
  - Corridor Following
  - Avoid obstacles
- Position Control
  - open/closed loop

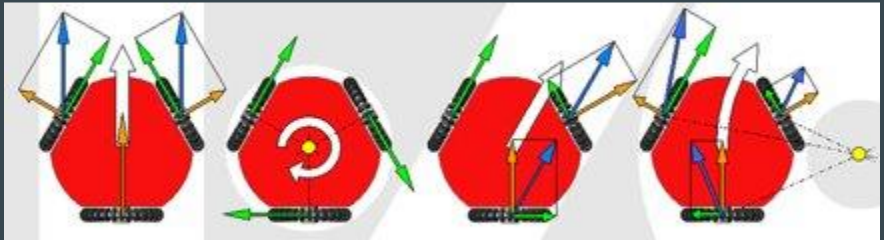
What voltage/current/Force commands to apply?

Movement Commands – The Twist

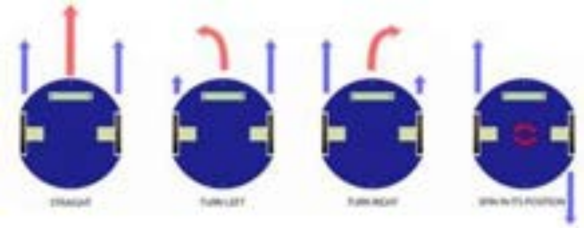
```
pub = rospy.Publisher(
                      '/cmd_vel',
                      Twist,
                      queue_size=1
            )
t = Twist()
pub.publish(t)
```

Movement Commands – Wheel Configuration

- Omni-directional Drive



- Differential Drive



Movement Commands – Controlling TurtleBot

Control Problem:

- Given a desired direction, what speed commands to apply on the wheels?

- Speed commands are translated to Torque/Current

Motion Control – Outline

- Models
  - Kinematics and Dynamics
- Control Algorithm
  - Provide appropriate commands to achieve a goal
- Engineering
  - From control signals to actuator commands

Motion Control Models

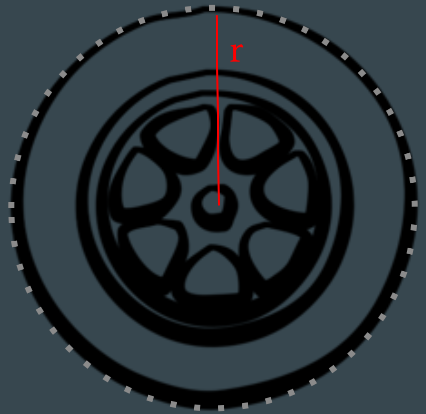
- Kinematics
  - Forward
    - Given wheels angular velocity, calculate velocity of the platform
  - Inverse
    - What angular velocity should be given to the wheels in order to achieve a desired velocity of the robotic platform.
- Dynamics
  - Forward
    - What is the angular velocity given the applied torques
  - Inverse
    - What torques to apply for achieving a desired angular velocity

Motion Control – Forward Kinematics

Given wheels angular velocity, calculate translation and rotation of the platform

$C = 2 \pi r$

- C = Wheel circumference
- r = radius



This is on rads!!!

Assume spinning at 60 deg per sec

$\omega = (deg/360)*2\pi = 1$ rad/s for 60 deg

Robot velocity: $V = r\,\omega$

Motion Control – Forward Kinematics

Two wheels:

- Simply take the mean

  $V = 0.5r(\omega_l + \omega_r)$

What are the angular velocities when the robot moves straight?

What when the robot rotates in place?

Motion Control – Inverse Kinematics

Make your turtlebot go forward with $v = 1m/s$ with $r = 6cm$

$V = r\,\omega$

Solving for angular velocity:

$$\omega = v/r$$

Motion Control – Angular velocity

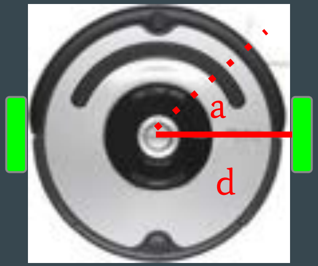
d : radius of robot

r: wheel radius

Angular velocity of the platform

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

What is the angular velocity if both wheels rotate with the same speed?



Motion Control – Forward and inverse kinematics

Forward:

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

Inverse:

- Solve for ω_l, ω_r
- $\omega_l = (v + (d * a)/2)/r$
- $\omega_r = (v - (d * a)/2)/r$

Motion Control – Python example

```python
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)
```

Motio

Forward and inverse Kinematics for Turtlebot

We define two functions `forward_kinemantics` 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 gemorty
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)
```

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 pe
r 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)
```

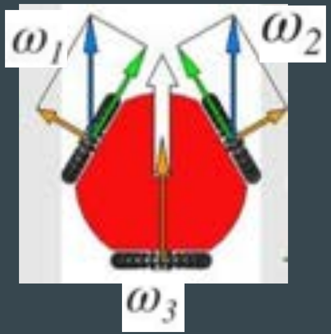
```
w_l = 20.000000,        w_r = 20.000000
v = 1.000000,   a = 0.000000
```

Kinematics of Omni-directional Drive

Which speeds to turn each wheel to move in desired direction?

Omni-wheels:

- Simple design
- Allow translation
- Significant slippage



Improvement:

- Mecanum Wheels

Kinematics of Omni-directional Drive

Straight line:

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

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

Pure rotation:

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

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

Kinematics of Omni-directional Drive

General velocity control:

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

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

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

Kinematics of Omni-directional Drive

General velocity control:

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

```
% 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;
```

Kinematics of Omni-directional Drive

General velocity control:

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

```
% 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;
```

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

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

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

```
% 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;
```

Algorithms for robot control

Open Loop

Three elements:

- Reference -> Desired robot position/velocity etc
- Motor command -> Wheel's velocity
- Output -> Actual position/velocity

Controller generates signals to meet the goal

Open Loop – Example

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)")
```

What can go wrong?

Open Loop – Disadvantages

No measurements - no feedback

Can rely on a model only

Better models – smaller errors, however the errors accumulate over time

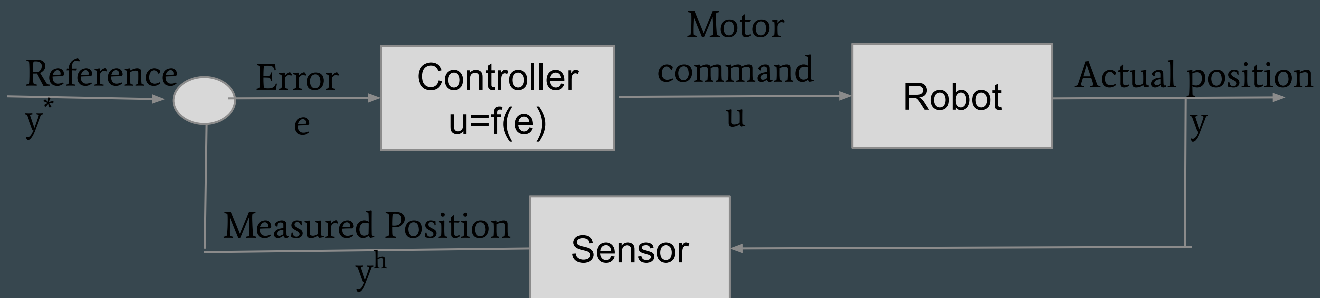
No possibility of correcting the errors since they are unknown

How to deal with unexpected events, changes, obstacles, etc.?

Closed Loop

Four elements:

- Reference -> Desired robot position/velocity etc
- Motor command -> Wheel's velocity
- Output -> Actual position/velocity
- Error -> Error between actual and desired state $e = y^{*} - y^{h}$



Controller generates signals to minimize error

Closed Loop Example

Repeat:

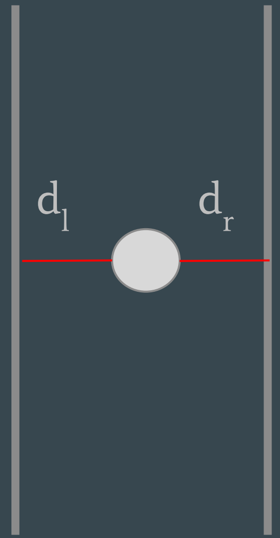
- Issue DriveForward command: u
- Read odometry and accumulate the total distance travelled y^h
- Stop if the total distance is greater than the specified value $y^* >= y^*$

Smaller time intervals – smaller errors

Closed Loop – Corridor Following

Scenario:

- two sensors measuring distance to the wall dl and dr
- robot moves constantly forward (v)
- controller affects the angular speed only ($u = \omega$)



Controller task:

- keep $d_l = d_r$

Can you think of a simple controller?

Closed Loop – Corridor Following

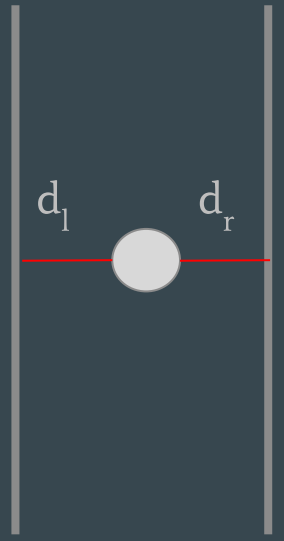
Define constant k= 5:

Loop:

- measure $e = d_l - dr_r$
- if $e > 0$ then $\omega = +K$
- if $e < 0$ then $\omega = -K$
- $\omega = K\,\mathrm{sign}(e)$

In Rovio language:

- if e > 0 then RotateLeft(5)
- if e < 0 then RotateRight(5)



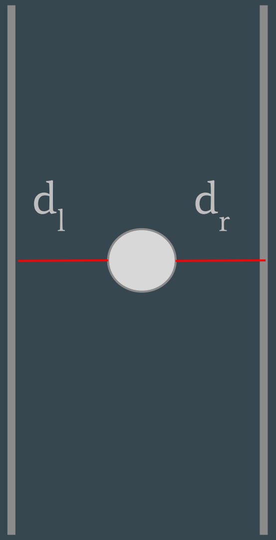
Closed Loop – Bang Bang Controller

Control input depends only on the sign of the error

$$u = \text{sign}(e)K$$

How K affects the movement?

What are the disadvantages?



Closed Loop – Proportional Controller

Change ω proportionally to the error value

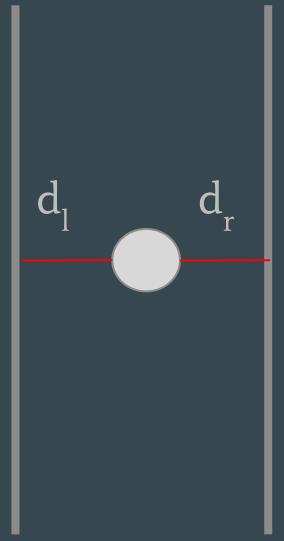
- $\omega = e*Kp$
  - small e – small correction
  - large e – large correction

Result:

- smoother actions and smaller errors

Kp parameter:

- large – faster reaction
- small – slower
- optimal value: smooth behaviour, robust to changes



Closed Loop – PID

Proportional (P)

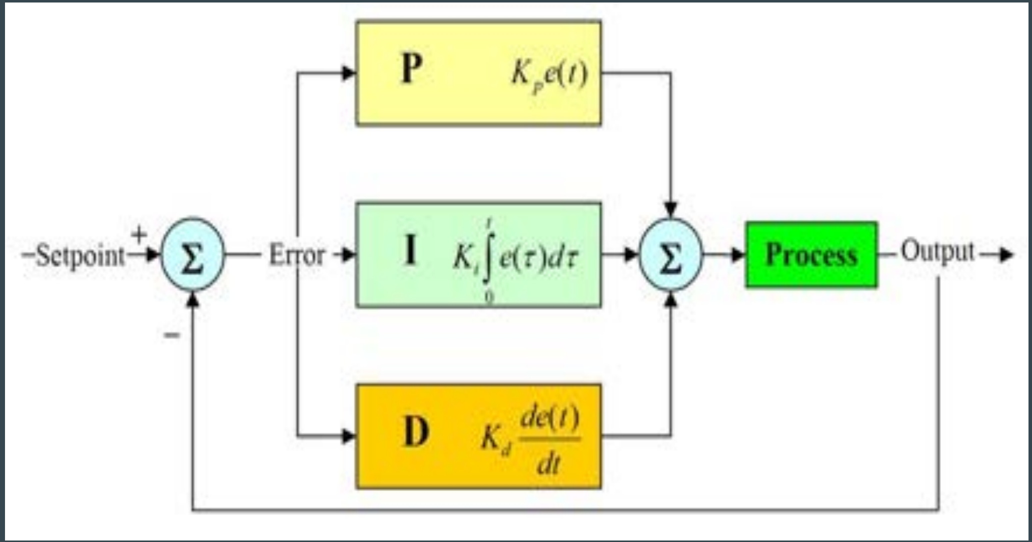
- Corrects error

Integral (I):

- Accelerates rate of change

Differential (D):

- Slows rate of change to avoid overshooting

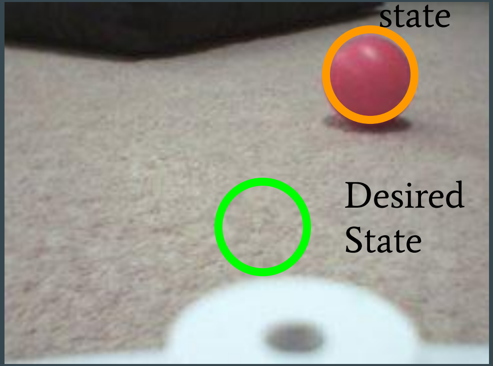


Closed Loop – Vision based control

Object state y:

- Position
- Size

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



Error:

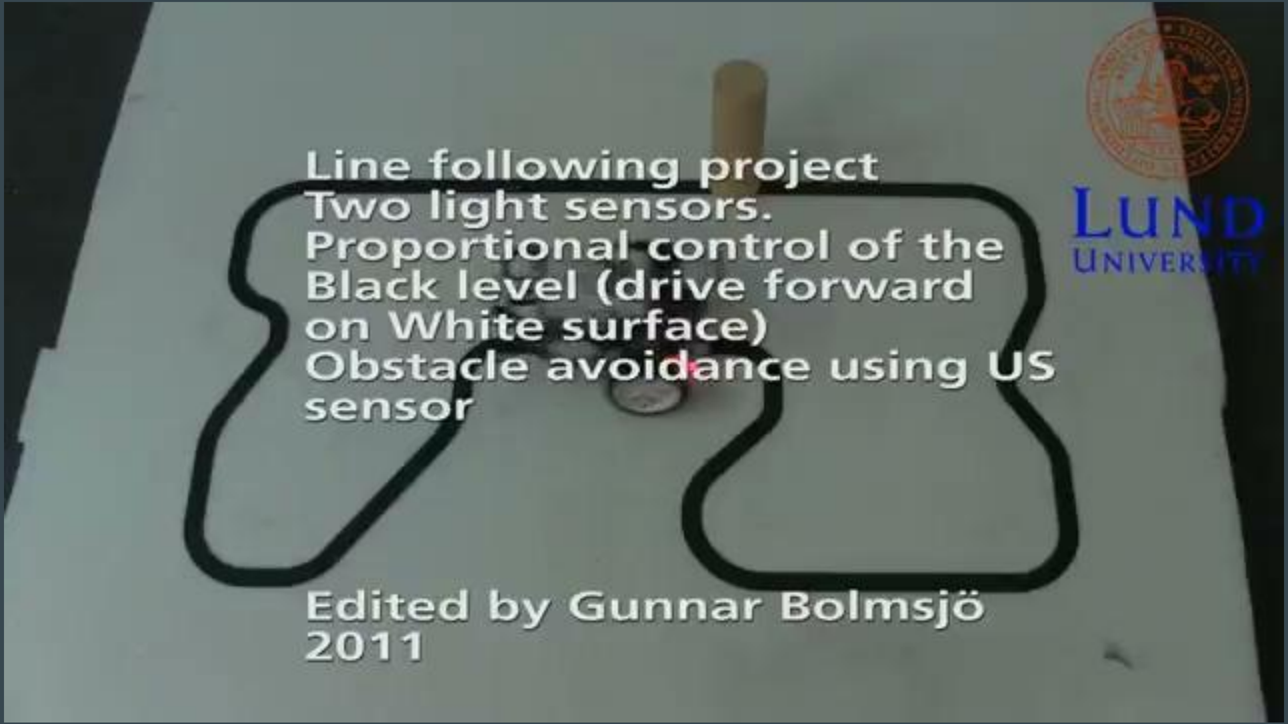
Difference between the desired and current state $\mathbf{e} = \mathbf{y}^* - \hat{\mathbf{y}}$

Control input:

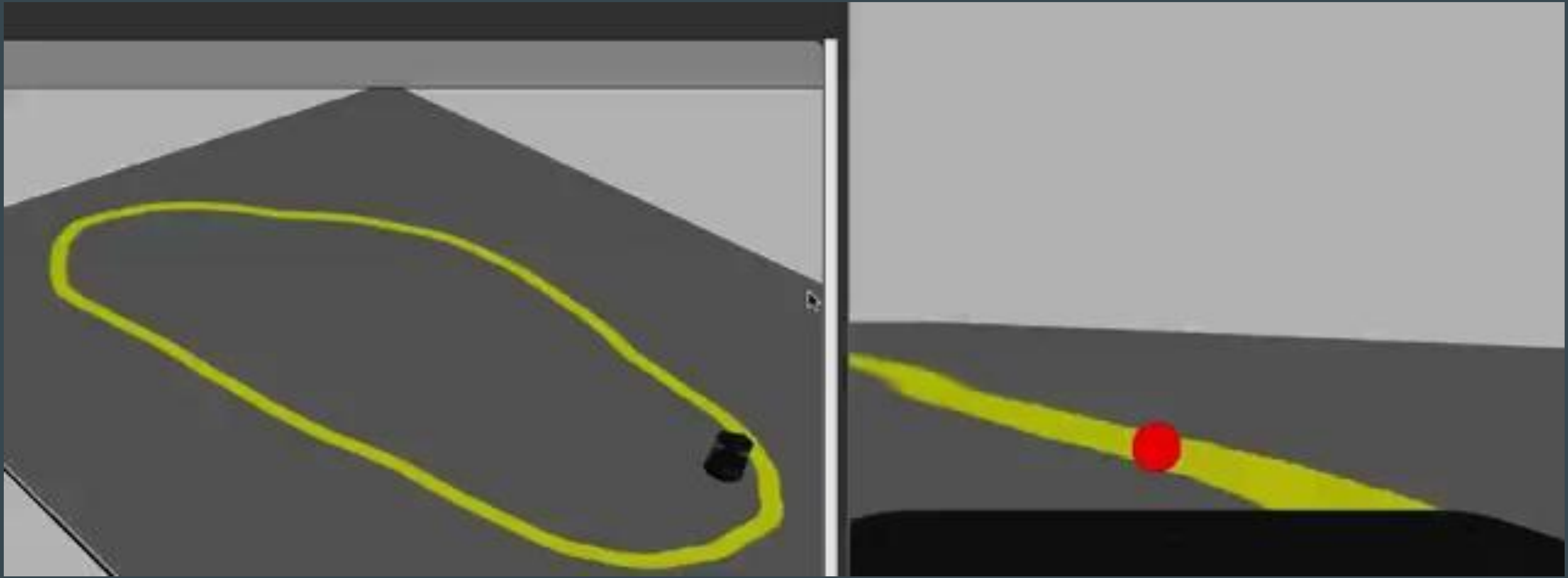
- Spin – adjust position
- Drive – adjust size

Engineering applications

Engineering applications



Engineering applications



Engineering applications

Balancing Robot

Engineering applications

Person following

Engineering applications

Control problems

Engineering applications

Control problems

Summary

- Movement Commands
- Motion Control Models
    - Inverse and Forward Models
- Algorithms for robot control
    - Open/Closed loop
    - Bang-Bang Controller
    - PID controller
- Application Examples

Further read: Siegwart book, chapter 4

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

UNIVERSITY OF
LINCOLN

CMP3103 – AUTONOMOUS MOBILE ROBOTS

Lincoln Centre for Autonomous Systems

School of Computer Science

L-CAS

Syllabus

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

Robot Behaviours

- Behaviour-based robotics
- Braitenberg vehicles
- Learning robot behaviours
- Combining robot behaviours
- Automatic composition of robot behaviours

Behaviour:
The way in which an animal or person behaves in response to a particular situation or stimulus.

Behaviour is a decision, an if statement is a decision.

Based on slides by Dr. A. Millard

Behaviour-Based Robotics (1980s)

Behaviour can be defined as a reaction to stimuli

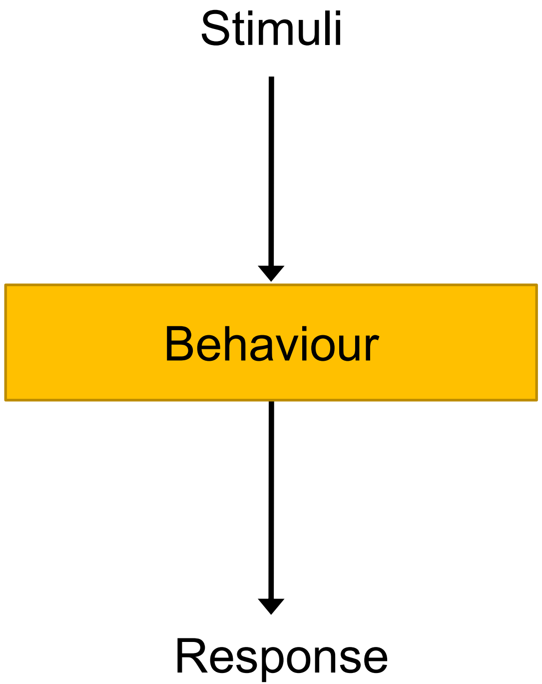
- Basic building block for robot actions

Often **reactive** behaviours

- No internal data interpretation (fast reactions!)

Reactive behaviours perform **closed loop** control

Different behaviours can be combined into complex ones – “emergent behaviours”



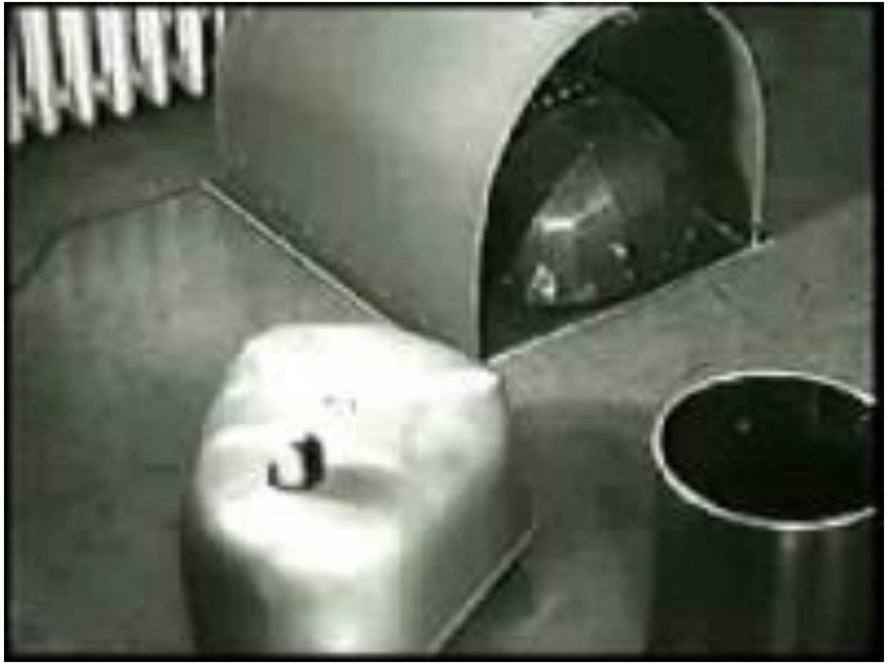
Robot behaviours

Example	Behaviour category
Wandering	Exploration/directional
Goal following	Goal-oriented, appetitive
Obstacle avoidance	Aversive/protective
Road following	Path following
Balance	Postural behaviours
Flocking	Social/cooperative
Visual search	Perceptual

Grey Walter’s tortoise robots

- “Machina Speculatrix”, 1948
- "it explores its environment actively, persistently, systematically, as most animals do"
- Neurophysiologist / cybernetician
- Experiments in reflex behaviour
- Analogue circuit built of vacuum tubes, electronic valves, photo-cells and a bumper sensor.
- Behaviour: approach or escape a light source
- Behaviour: obstacle avoidance.

Grey Walter’s tortoise robots



Grey Walter's tortoises - https://www.youtube.com/watch?v=lLULRlmXkKo

Tortoise robot behaviours

- Seek light (exploration)
- Move towards / back away from light
- Avoid obstacles
- Recharge battery

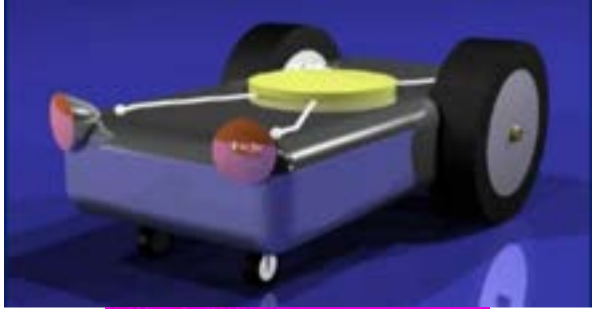
Reactive control

- Tight coupling of sensors and actuators

- No internal model of the world

- Well-suited to dynamic/unstructured environments

Braitenberg vehicles



Valentino Braitenberg, 1984

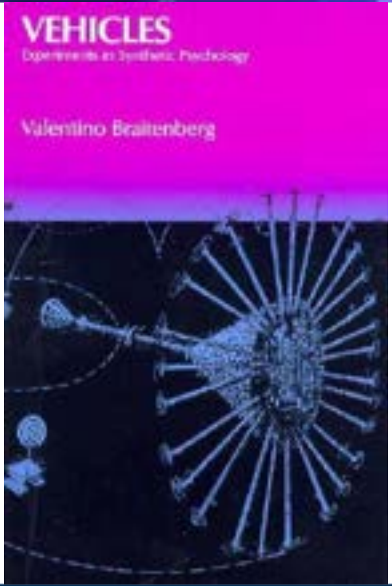
- German neuroscientist

Series of thought experiments

- Building seemingly complex behaviours from simple interactions

Inhibitory and excitatory influences

- **Direct coupling** of sensors and motors



Valentino Braitenberg, “Vehicles: Experiments in Synthetic Psychology”, MIT Press, 1984

Vehicle 1 Getting Around

One sensor

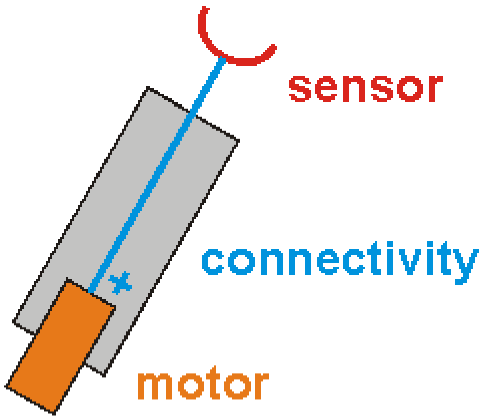
- e.g. light, temperature, sound

One motor

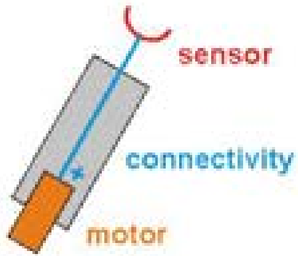
- Directly driven by sensor
- +/- indicates excitatory/inhibitory relationship

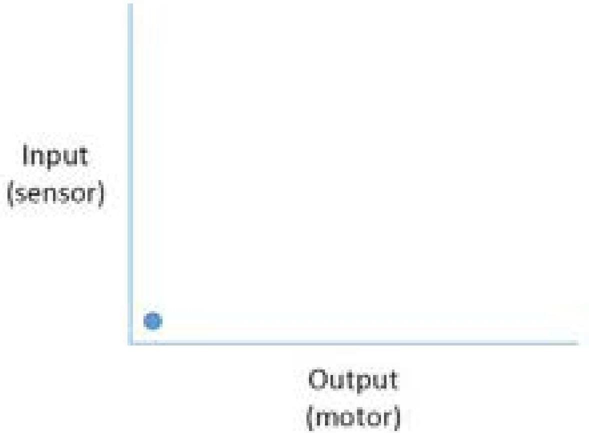
Motor speed is **proportional** to sensor reading

- e.g. greater light intensity == faster motor speed

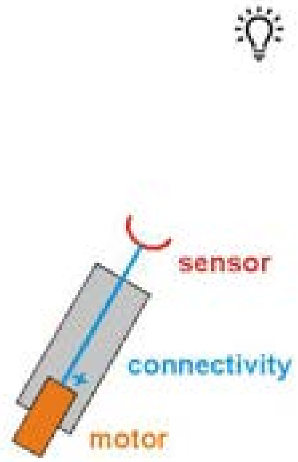


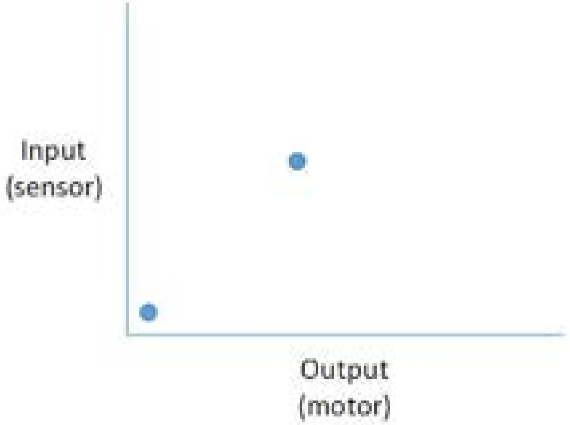
Vehicle 1



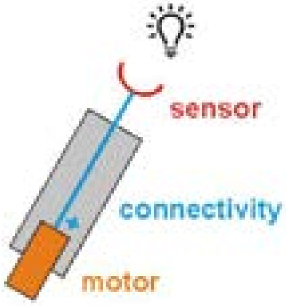


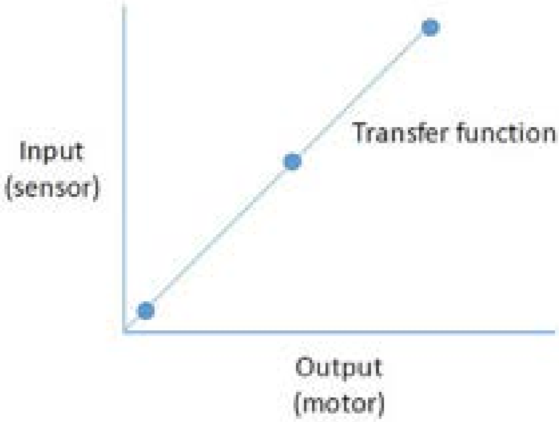
Vehicle 1





Vehicle 1


sensor
connectivity
motor


Input
(sensor)
Transfer function
Output
(motor)

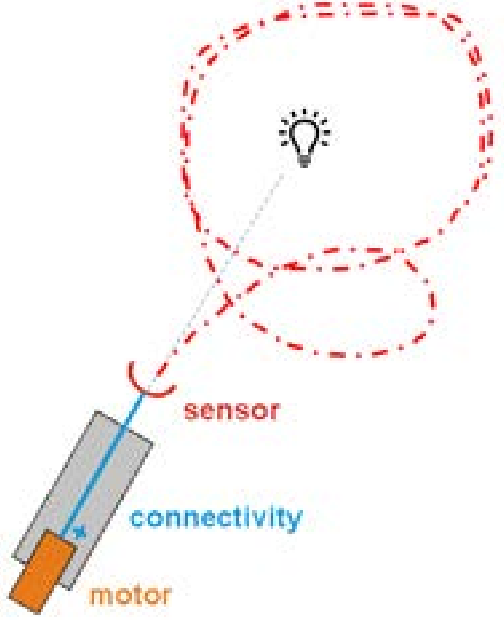
Vehicle 1

Under perfect conditions

- Vehicle stops on dark spots

If physics are added becomes more realistic

- e.g. non-symmetric drive or friction
- Interesting emergent behaviour
  - Looping continuously around sources
  - Brownian motion at group level



Vehicles 2a and 2b

Two sensors, two motors

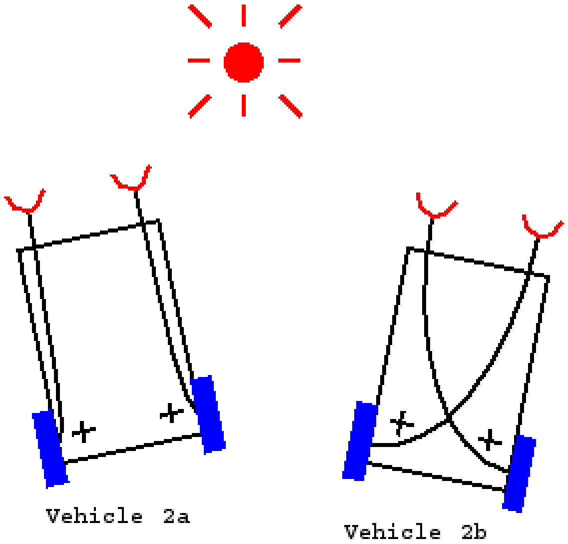
- Directly coupled
- Excitatory connections

Vehicle 2a

- Sensors linked to motors on same side

Vehicle 2b

- Sensors linked to motors on opposite side



Vehicles 2a and 2b

Two sensors, two motors

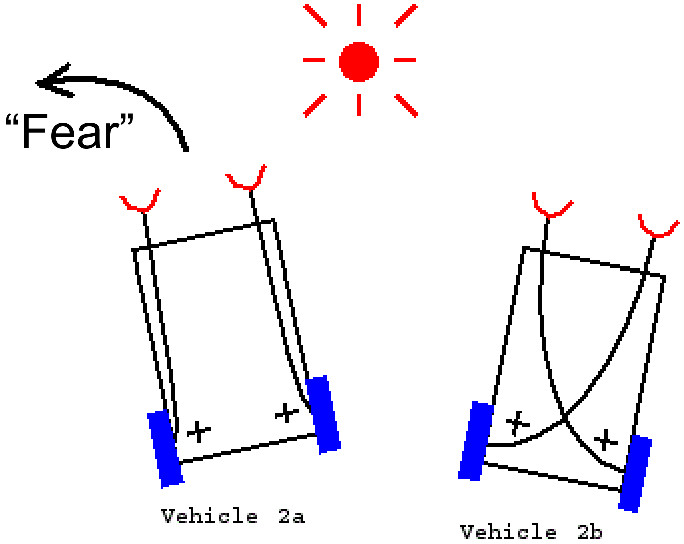
- Directly coupled
- Excitatory connections

Vehicle 2a - **Fear**

- Sensors linked to motors on same side

Vehicle 2b

- Sensors linked to motors on opposite side



Vehicles 2a and 2b

Two sensors, two motors

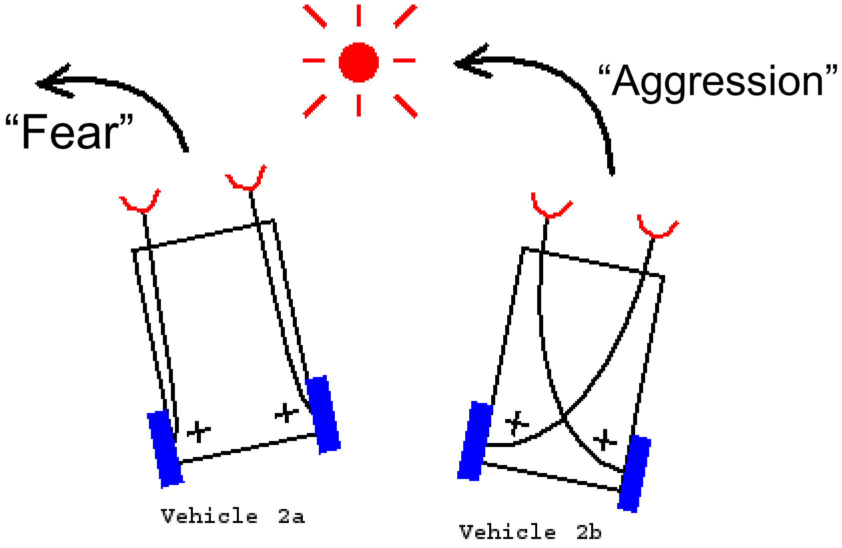
- Directly coupled
- Excitatory connections

Vehicle 2a - **Fear**

- Sensors linked to motors on same side

Vehicle 2b - **Aggression**

- Sensors linked to motors on opposite side



Vehicles 3a and 3b

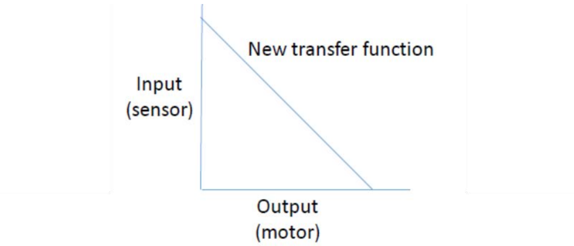
Same as vehicle 2, but with **inhibitory** connections

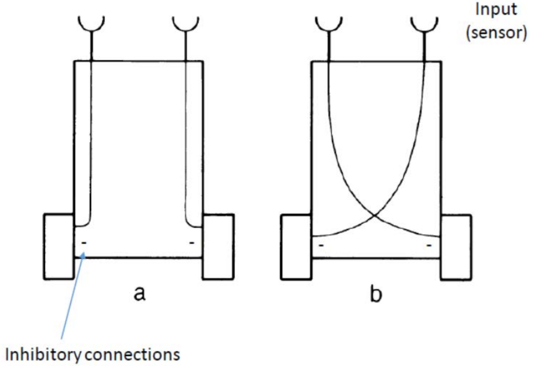
Vehicle 3a

- Sensors linked to motors on same side

Vehicle 3b

- Sensors linked to motors on opposite sides





Vehicles 3a and 3b

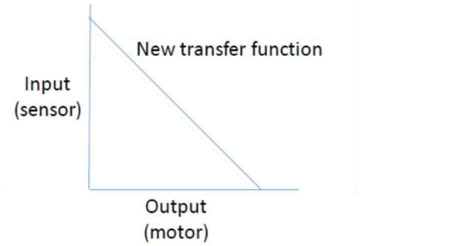
Same as vehicle 2, but with **inhibitory** connections

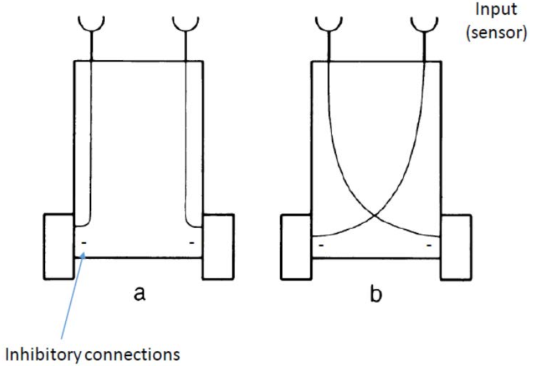
Vehicle 3a – **Love**

- Sensors linked to motors on same side

Vehicle 3b

- Sensors linked to motors on opposite sides





Vehicles 3a and 3b

Same as vehicle 2, but with **inhibitory** connections

Vehicle 3a – **Love**

- Sensors linked to motors on same side

Vehicle 3b – **Explorer**

- Sensors linked to motors on opposite sides

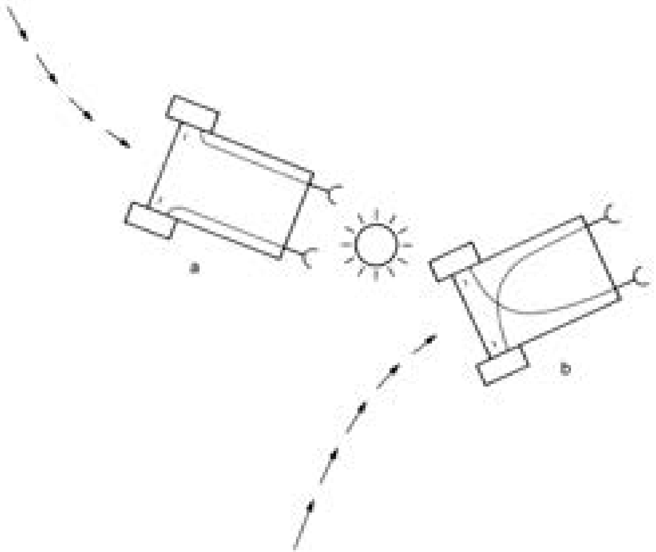
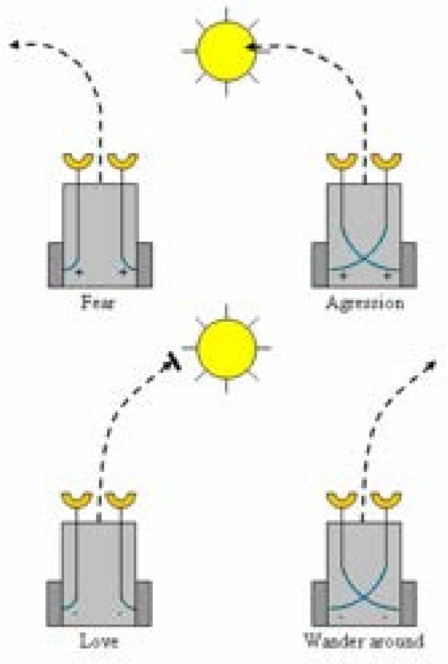


Figure 4

Vehicle 3, with inhibitory influence of the sensors on the motors.

Braitenberg Vehicles Behaviours

- Vehicle 2a – Fear
  - Turns away from stimuli
  - Slows down in the absence of stimuli
- Vehicle 2b – Aggression
  - Turns towards stimuli
  - Accelerates towards stimuli
- Vehicle 3a – Love
  - Turns towards stimuli and de-accelerates
- Vehicle 4a – Exploration
  - Turns away from stimuli and accelerates



http://www.harmendeweerd.nl/braitenberg-vehicles/

Vehicles 3a and 3b - Love

https://www.youtube.com/watch?v=RJmENOPq444

https://www.youtube.com/watch?v=TeyvKGIYQrE

Vehicles 3a and 3b - Explorer

https://www.youtube.com/watch?v=yz6ijhXEd9Y

Multi-sensory vehicles

Increasingly complex behaviours can be created by adding:

New sensors with changing profiles

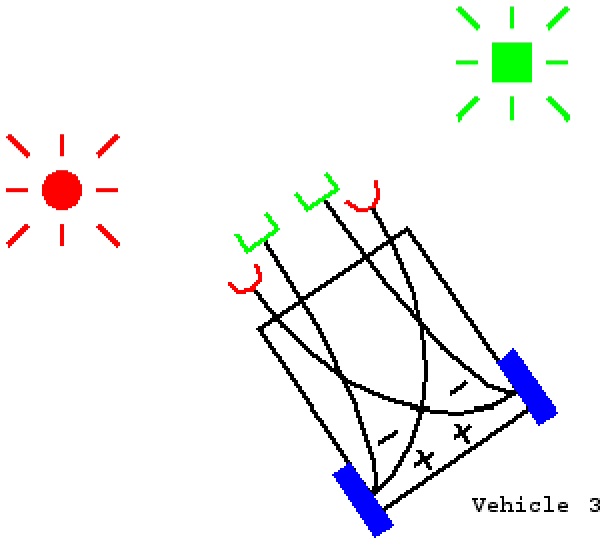
- Non-linear / digital

New transfer functions

- Non-linear / weighting

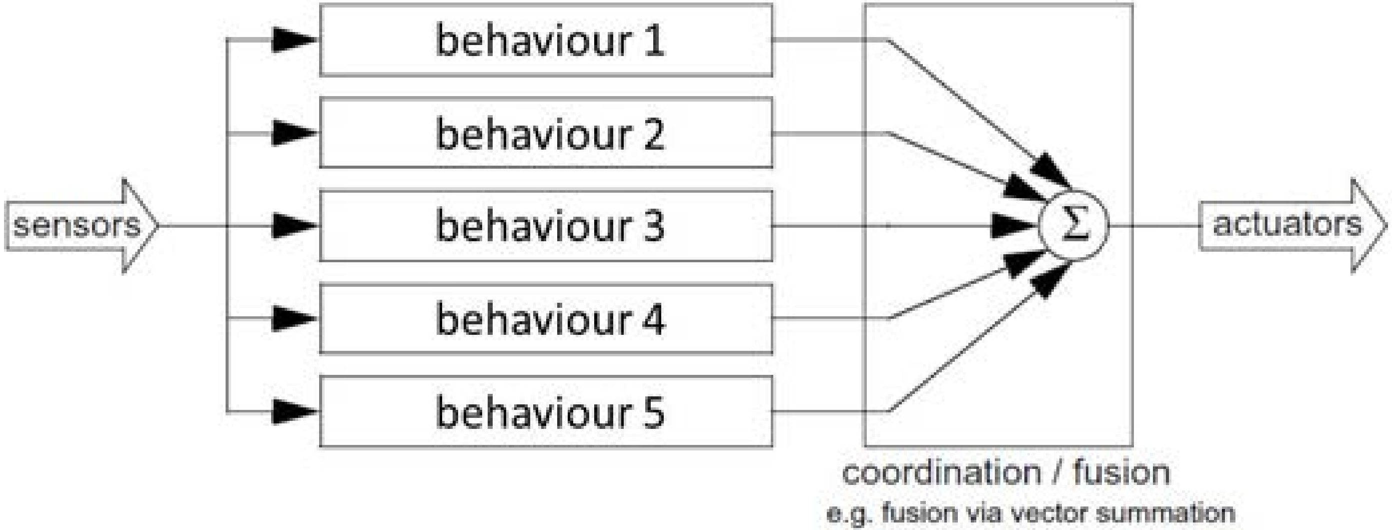
Varying excitatory and inhibitory connectivity

- e.g. goal seeking vs obstacle avoidance

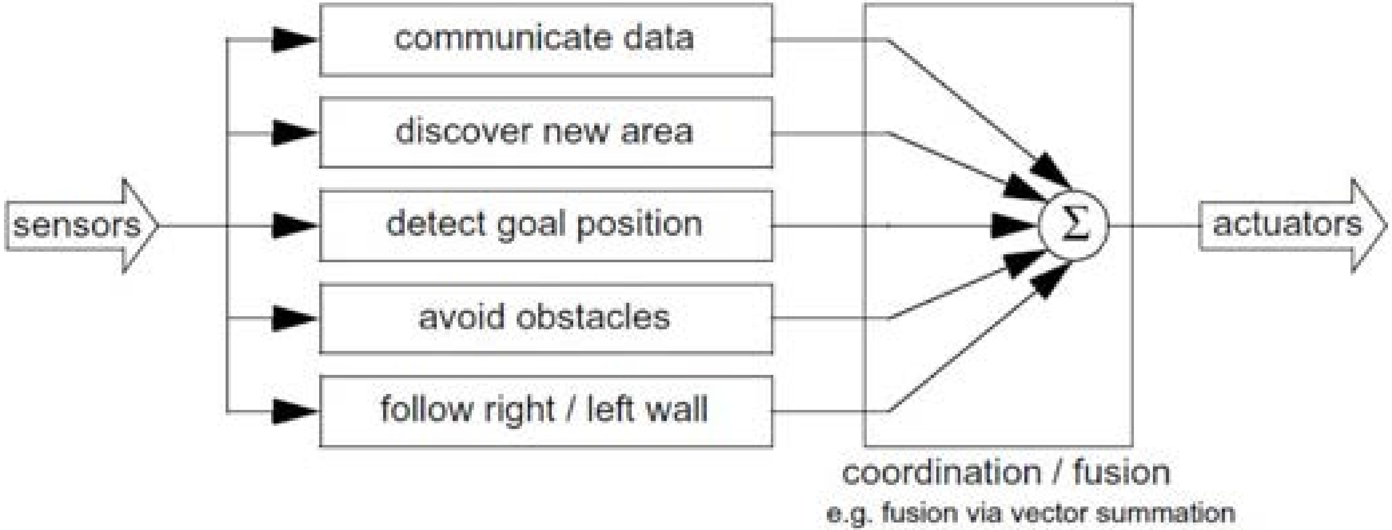


Combining robot behaviours

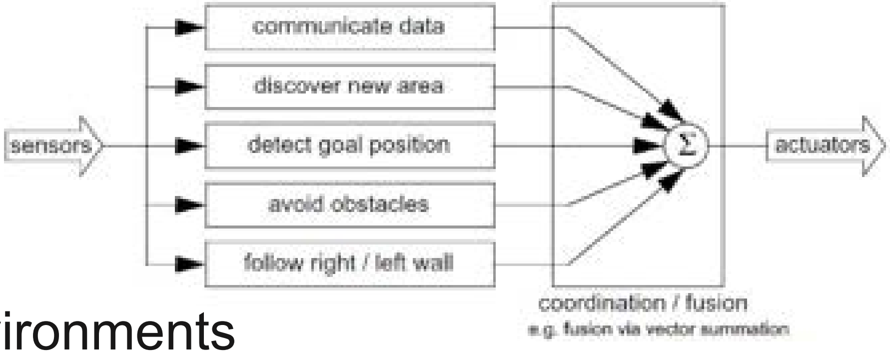
Multi-behaviour vehicles



Multi-behaviour vehicles



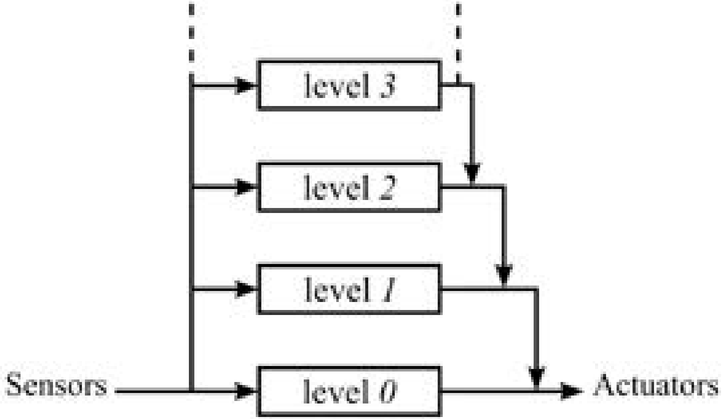
Vector summation



- Can be implemented quickly
- Does not directly scale to other environments
- Underlying procedures must be carefully designed to produce the desired behaviour
- How to coordinate **the order of behaviours** in time?
- How to coordinate behaviours **at the same time**?

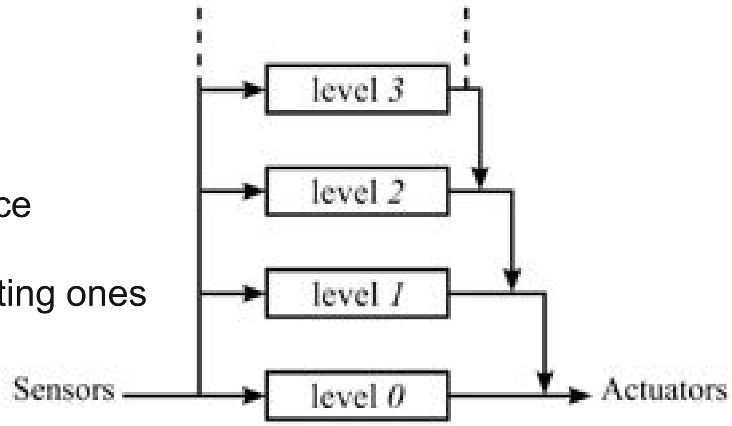
Subsumption architecture (1986)

- Behaviour decomposed into separate layers
- Priority-based hierarchy
  - Simplest low-level behaviours at the bottom
  - Complexity increases with higher layers
  - Levels are weighted so higher importance commands have higher weights.
- Layers operate asynchronously in parallel

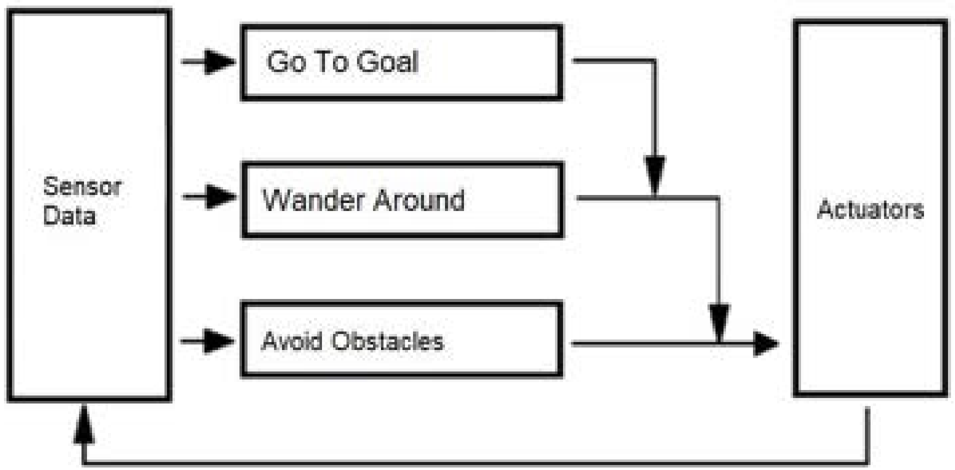


Subsumption architecture (1986)

- Principles of design:
  - systems are built from the bottom
  - layers are task achieving actions/behaviours (avoid obstacles, find-doors, visit rooms)
  - components are organized in layers,
    - bottom lowest layers handle most basic tasks
    - all rules can be executed in parallel, not in a sequence
    - newly added components and layers exploit the existing ones



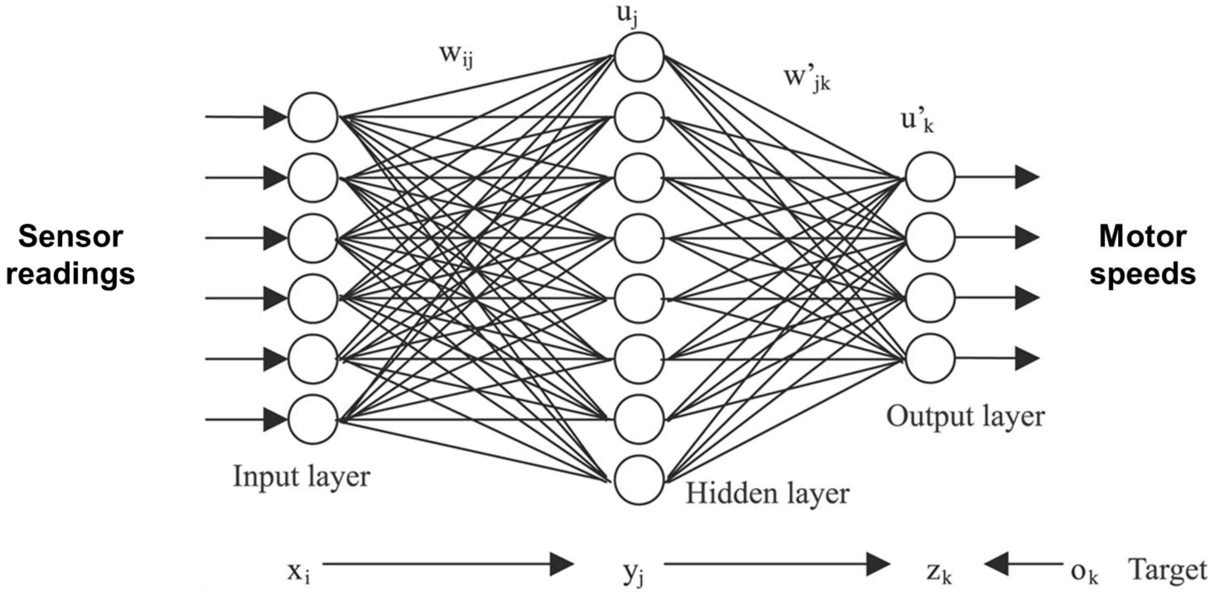
Subsumption architecture (1986)



Learning robot behaviours

UNIVERSITY OF LINCOLN

Artificial Neural Networks (ANNs)



$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$

In order to introduce a behaviour we need to train the network.

https://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they

Neural network learning of behaviours

- Reformulation of the multi-sensory Braitenberg vehicle as an **artificial neural network**
    - Inputs: sensor readings
    - Outputs: motor speeds
- Intended network output (motor speeds) provided by human
    - “Teacher” with a joystick

Neural network Visual Navigatoin Teach and Repeat

https://www.youtube.com/watch?v=3wRQKkFnQlE

Robot Behaviour Learning

- Learned behaviours included:
- Obstacle avoidance
- Wall following
- Box pushing

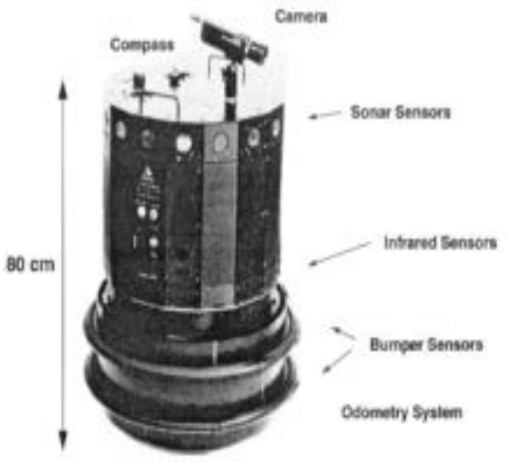


FIG. 3.17. THE NOMAD 200 MOBILE ROBOT *FortyTwo*

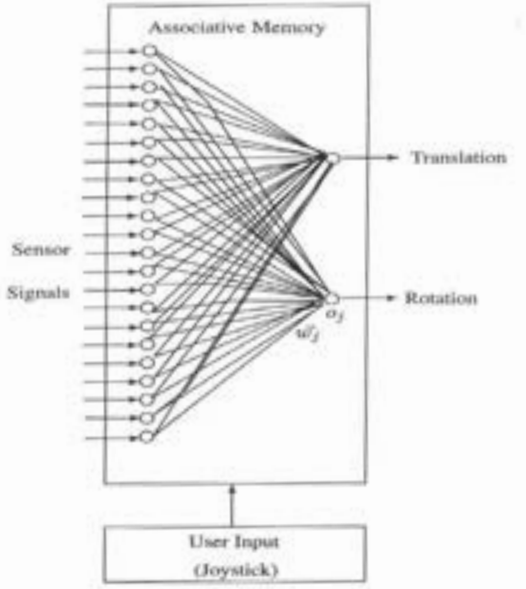


FIG. 4.15. THE CONTROLLER ARCHITECTURE

5 values	5 values	6 values	6 values
Left facing Sonars	Right facing Sonars	Left facing IRs	Right facing IRs

FIG. 4.16. THE INPUT VECTOR USED

From Nehmzow (2002)

Evolutionary robotics

- Robot self-learns behaviours
- Genetic Algorithm
  - Black-box optimization of a fitness function
  - Inspired from evolution theory
- Other evolutionary algorithms:
  - Ant colony optimization
  - Particle swarm optimization

Evolutionary robotics: what, why, and where - https://www.frontiersin.org/articles/10.3389/frobt.2015.00004/full

Evolutionary robotics - Genetic Algorithm

Phases of a Genetic Algorithm

- Fitness function: the function we want to optimize
- Initial population:
  - A set of a randomly selected optimization variables
- Selection:
  - Best performing Chromosomes are selected
- Crossover:
  - Exchange of genes based on a random crossover point
- Mutation:
  - Random change to genes of offsprings (low probability)
- Termination:
  - Population has converged (no significant differences)

Example

- Find best weights for a neural network.
- Random selected weights.
- Run the network
- Takes each set of weights and evaluates against the fitness function. Selects best weights from evaluation.
- Exchange weight values bases on a random point. This generate offspring's of those two weights.
- Applies some random changes to weights.
- Terminates when there is no significant changes in the weights of the offspring.

Optimase the weights of a NN. Gene

w1	-0.2	0.3	0.7	-0.5
w2	-0.3	0.8	0.5	-0.2
w3	0.2	0.3	-0.7	-0.5

Chromosome

Population

w1	-0.2	0.3	0.7	-0.5
w2	-0.3	0.8	0.5	-0.2

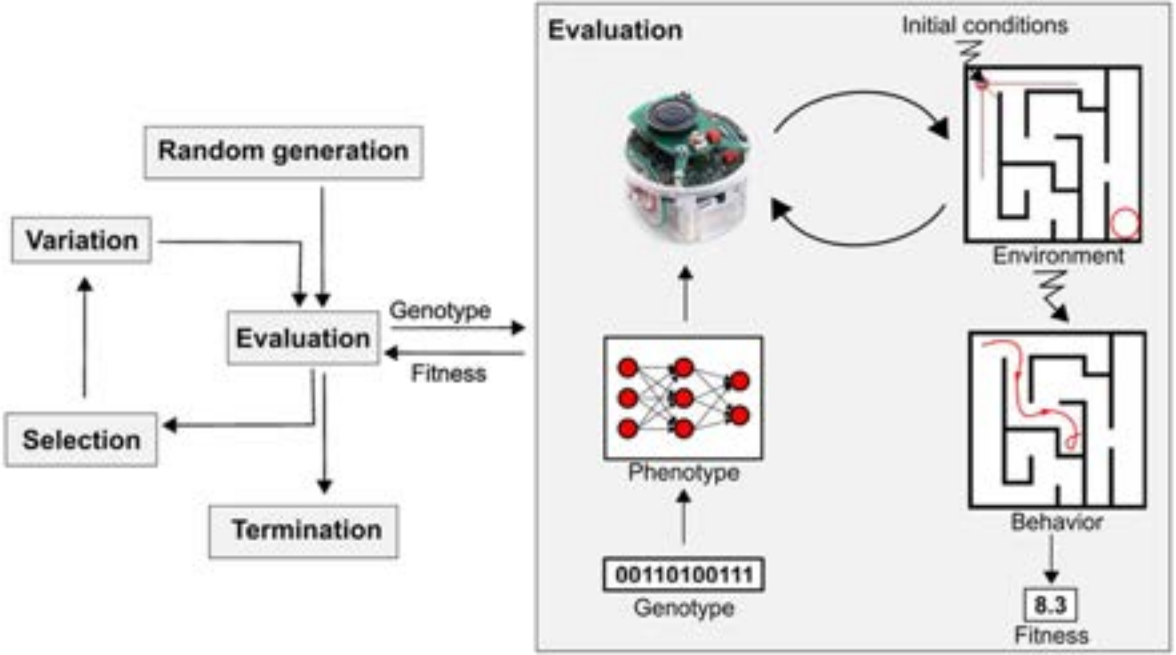
Crossover point (random)

w4	-0.3	0.8	0.7	-0.5
w5	-0.2	0.3	0.5	-0.2

New offsprings

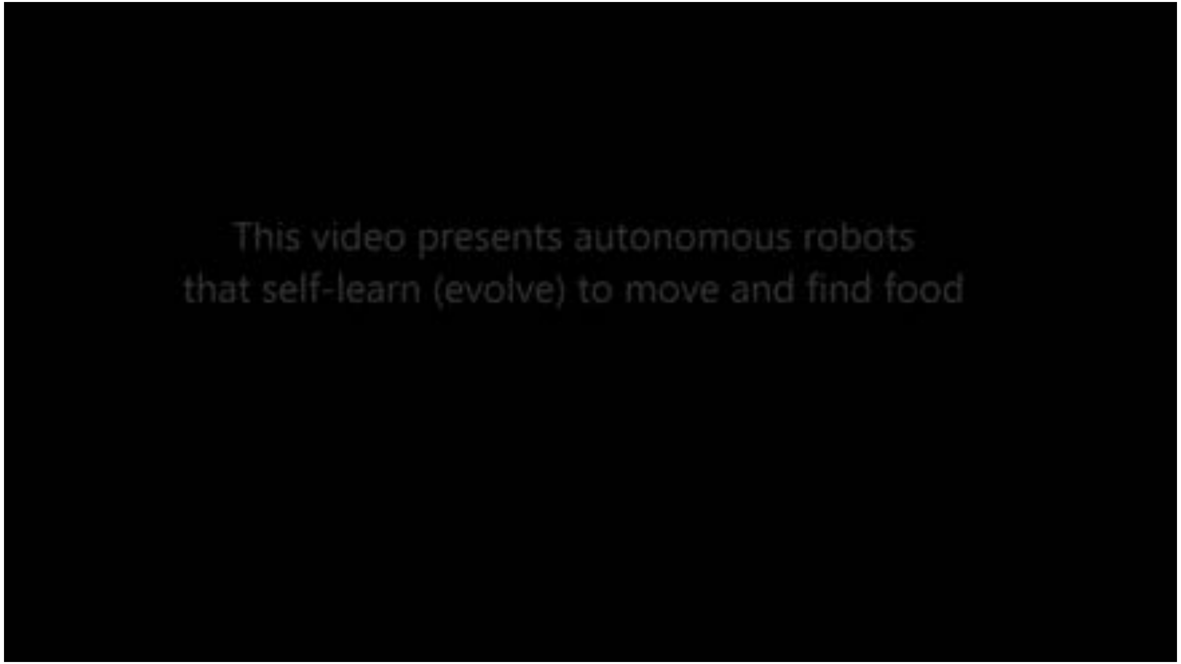
Evolutionary robotics: what, why, and where - https://www.frontiersin.org/articles/10.3389/frobt.2015.00004/full

Evolutionary robotics



Evolutionary robotics: what, why, and where - https://www.frontiersin.org/articles/10.3389/frobt.2015.00004/full

Evolutionary robotics

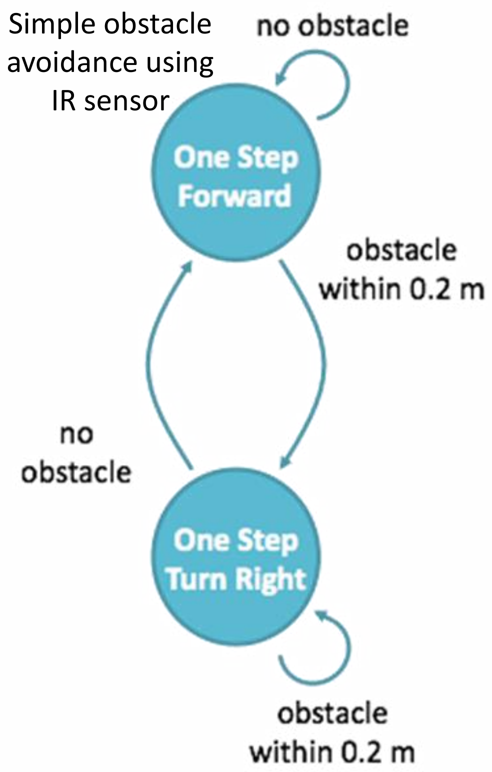


Evolutionary Robots - https://www.youtube.com/watch?v=VUddPhXdBnY

Explicitly coding behaviours

Finite State Machine (FSM)

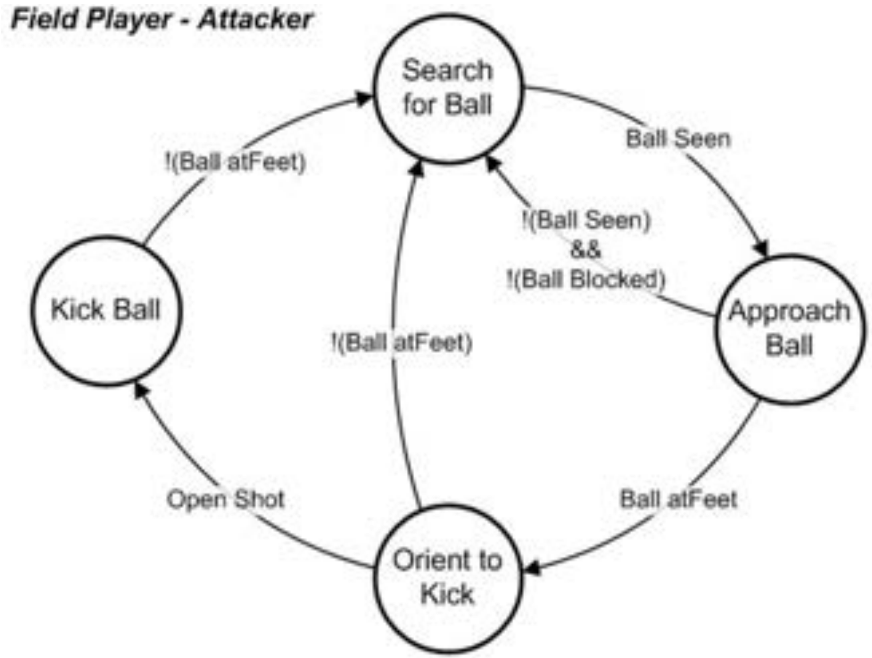
- Collection of states/actions
  - States: Nodes
  - Transitions: Arrows
- A discrete number of states (behaviours)
- Robot always in some defined **state** (behaviour)
  - Exploring, avoiding, waiting, etc
- Actions are associated with states
  - enter, exit, in-state
- Robot **conditionally** transitions between states
  - e.g. If sensor reading > threshold, then transition



Finite State Machine – Robot Footboall

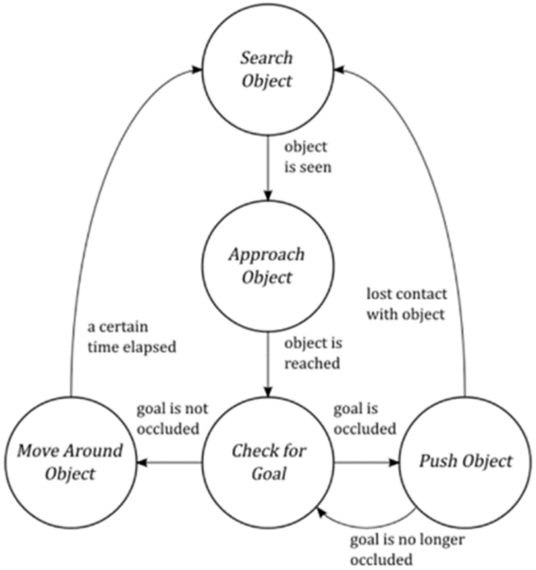
Challenges:

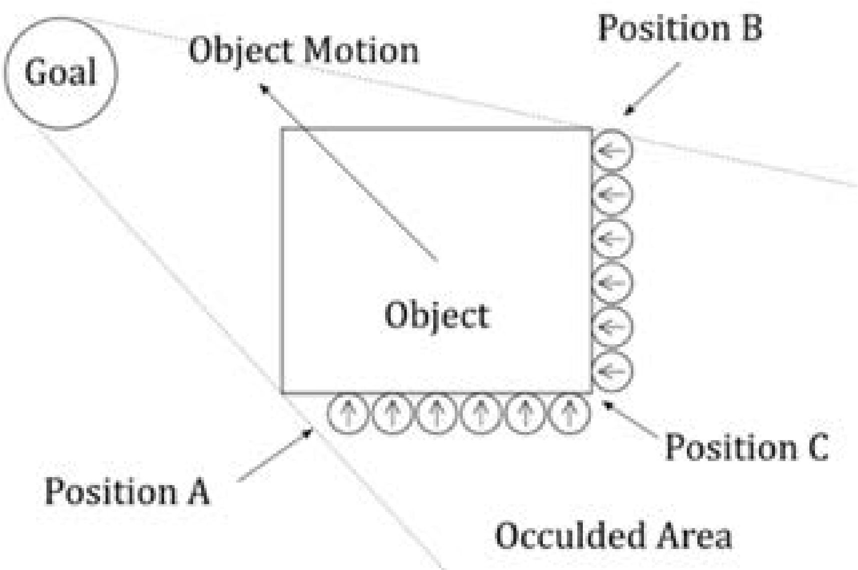
- Avoid dead ends
- Avoid infinite loops of states



A Finite State Machine for Robot Football

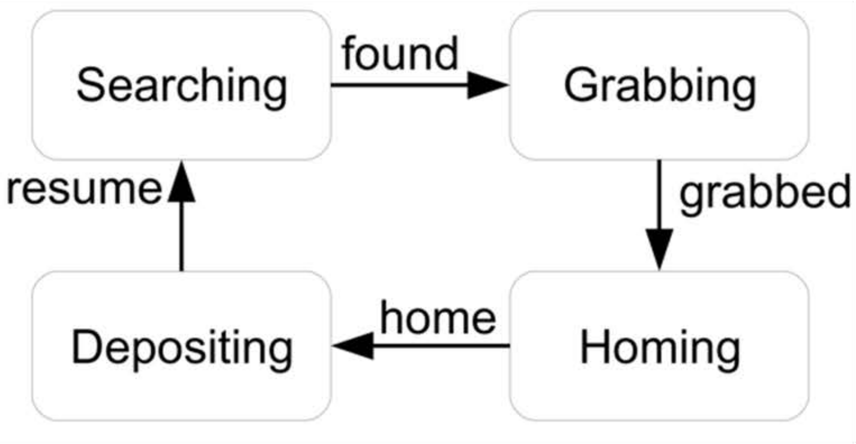
Finite State Machine – Object search

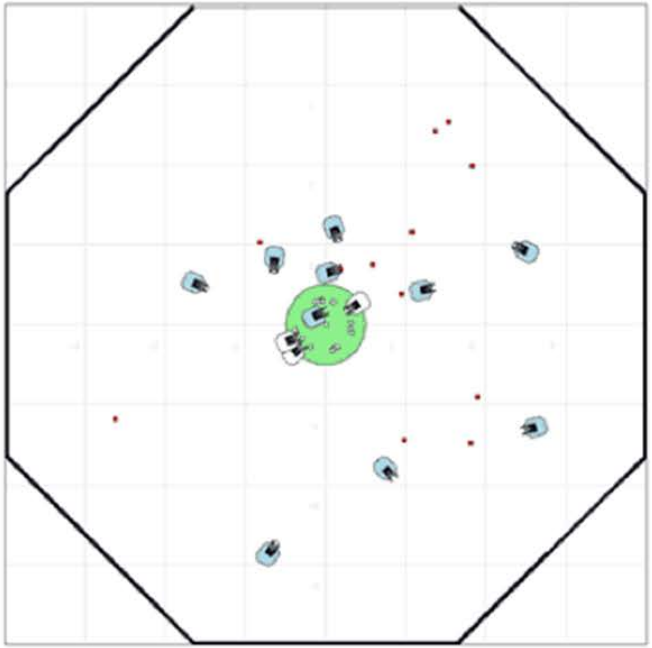




Implemented using a series of if-statements

Finite State Machine – Foraging example



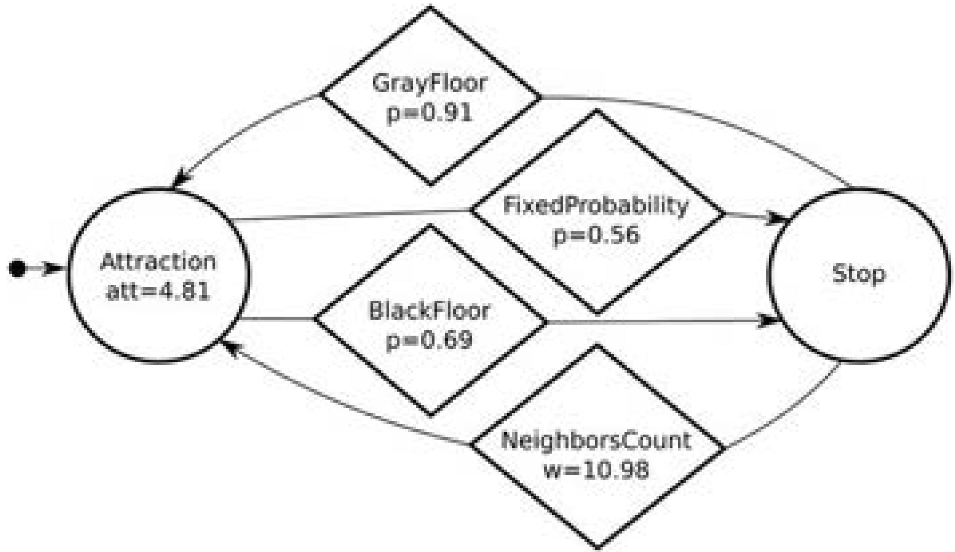


Probabilistic Finite State Machine (PFSM)

- Transitions are probabilistic
  - Introduction of stochasticity in the system
  - More accurate world representation

Concurrent design of control software and configuration of hardware for robot swarms under economic constraints: https://peerj.com/articles/cs-221/

Probabilistic Finite State Machine (PFSM)



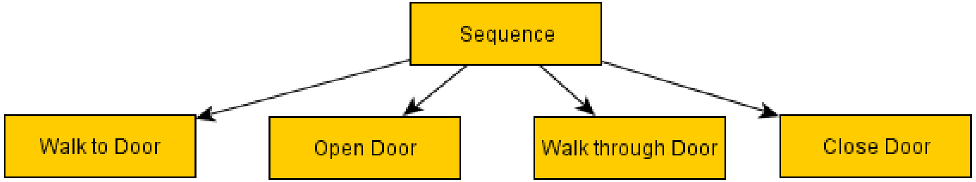
Concurrent design of control software and configuration of hardware for robot swarms under economic constraints: https://peerj.com/articles/cs-221/

Behaviour Trees

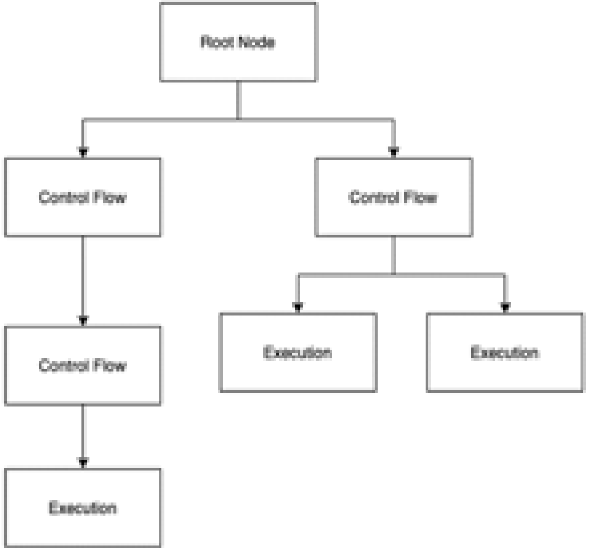
- Directed Graph
- Alternative to FSM
- Can model very complex behaviours by decomposing to simpler behaviours
- Represent transitions and sequences of tasks

What is a Behavior Tree and How do they work? - https://www.youtube.com/watch?v=DCZJUvTQV5Q

Behaviour Trees – Nodes



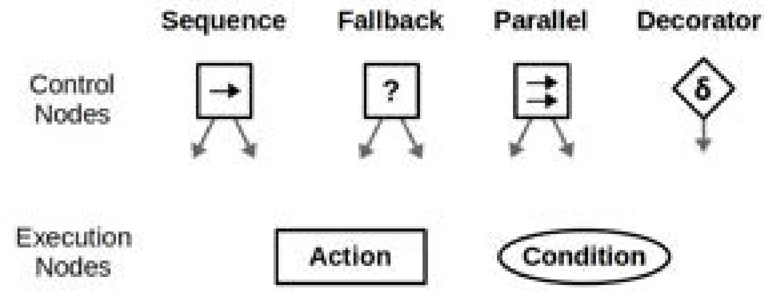
- Control Flow Nodes:
  - The internal nodes
- Execution Nodes (bottom nodes):
  - The leaf nodes
- Nodes can return the following states:
  - Running
  - Success
  - Failure
- Can create very complex behaviours composed of simple tasks without worrying how the simple tasks are implemented.



https://towardsdatascience.com/designing-ai-agents-behaviors-with-behavior-trees-b28aa1c3cf8a

Behaviour Trees – Nodes

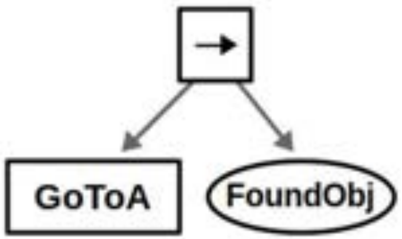
- Sequence:
  - Execution in order until all return success or one failure
- Fallback:
  - Execution in order until one returns success or all children return failure
- Parallel:
  - Execution in “parallel”. Returns success if all or some tasks have succeeded
- Decorator (delta):
  - Modifies the return of a node
- Action:
  - Performs a defined task
- Condition:
  - Evaluates a condition, true, false, threshold etc.



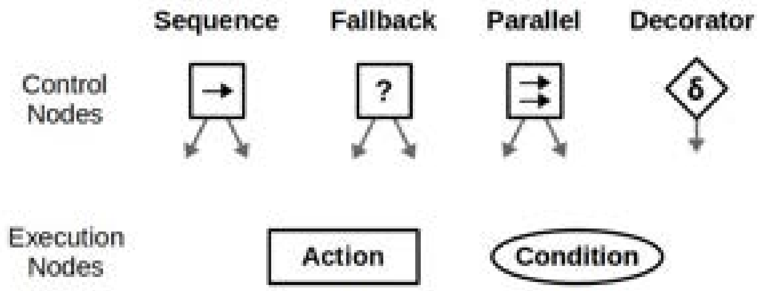
https://robohub.org/introduction-to-behavior-trees/

Behaviour Trees – Example

Task: Search for an object at specific locations

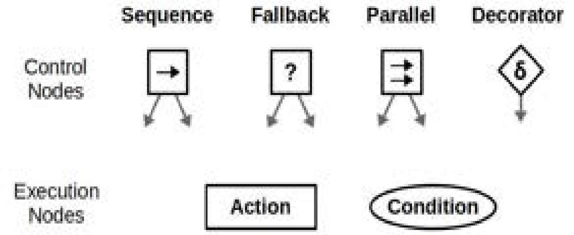


- Start at sequence node.
- Then Go To A.
- When this has been executed.
- Next is the condition have we found the object?
    - If true return to the node above.
    - If false continue to next node.

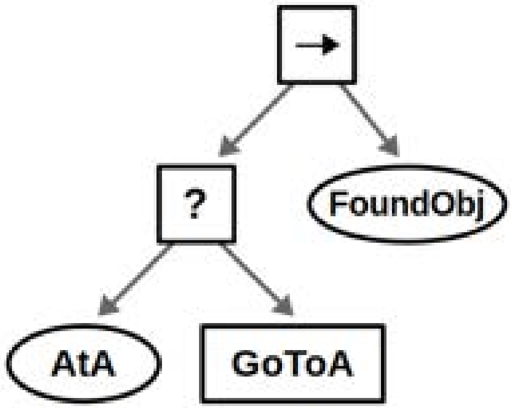


https://robohub.org/introduction-to-behavior-trees/

Behaviour Trees – Example



Task: Search for an object at various locations

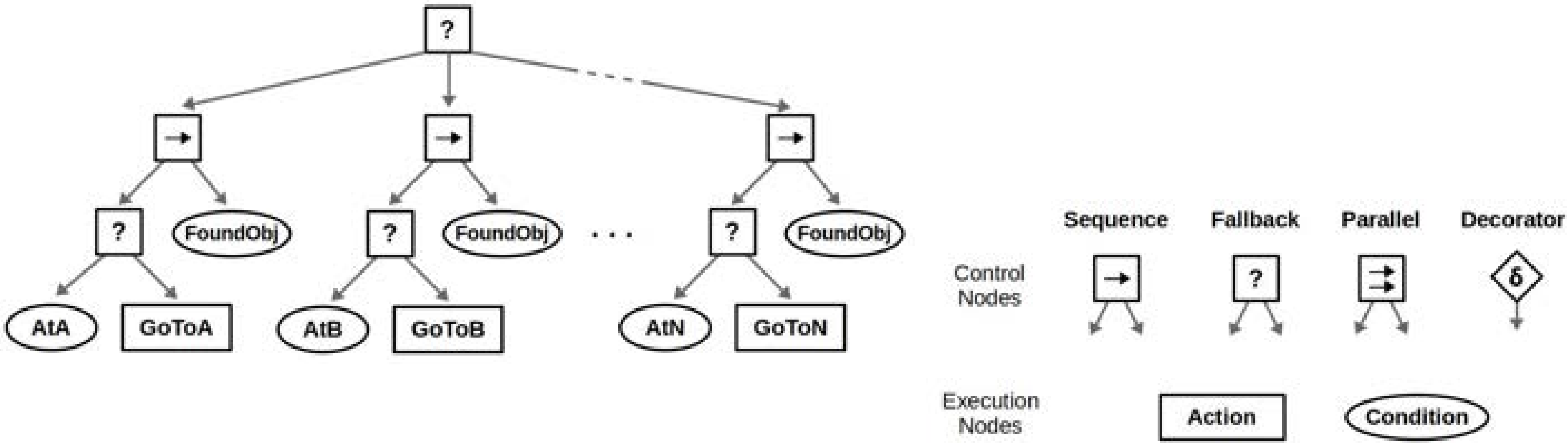


- Start at sequence node.
- Then go to fallback node
- Next is the condition are we At A?
  - If true, fall back node return success to sequence node
  - If false, then Go To A.
  - This will loop until we are at A or both nodes return failure.
- Fall back node return success.
- Next is the condition have we found the object?
  - If true return to the node above.
  - If false continue to next node.

https://robohub.org/introduction-to-behavior-trees/

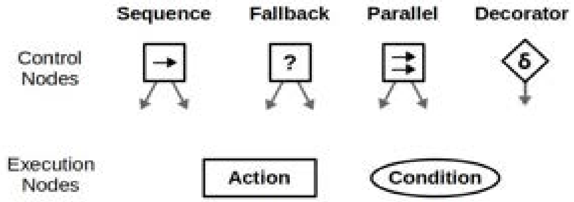
Behaviour Trees – Example

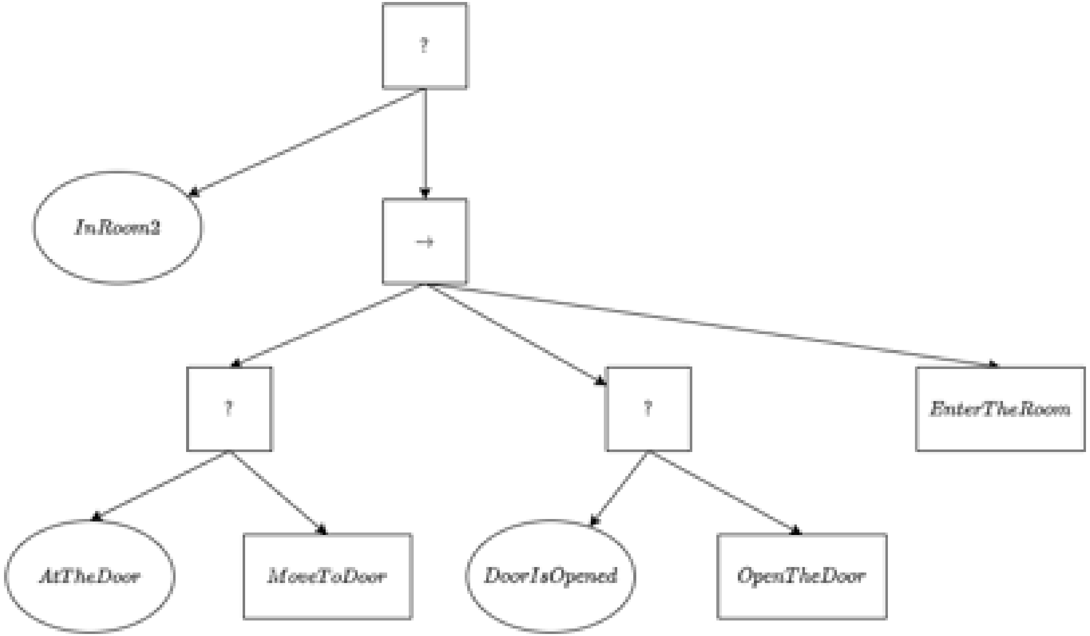
Task: Search for an object at various locations

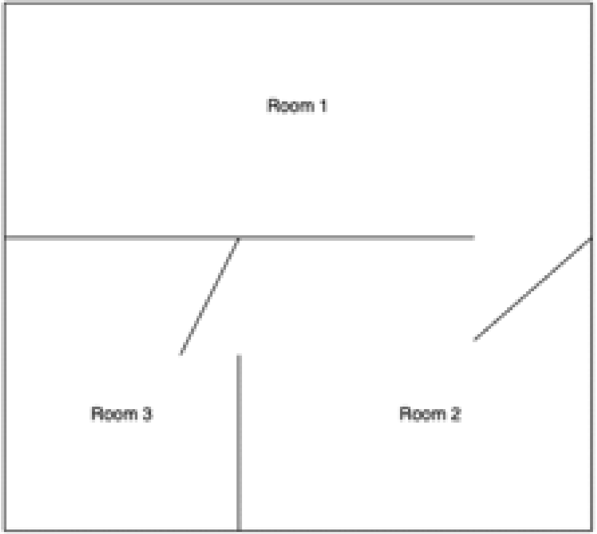


https://robohub.org/introduction-to-behavior-trees/

Behaviour Trees



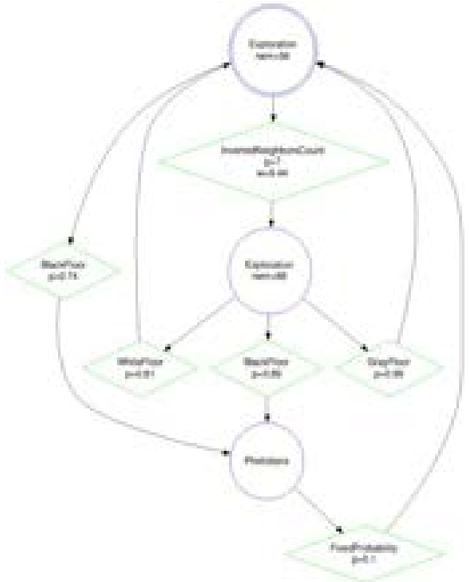




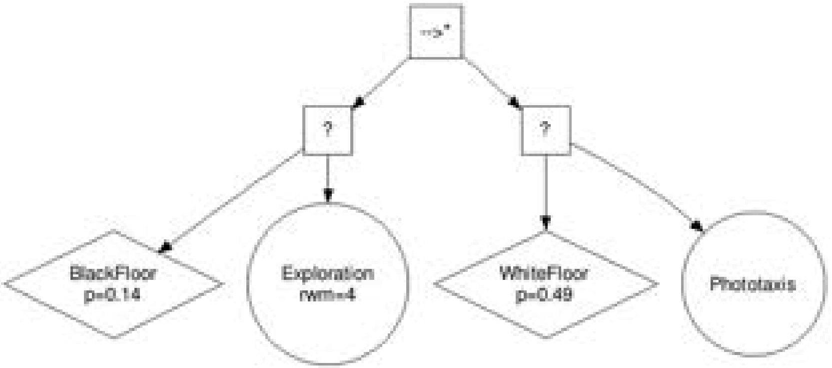
Designing AI Agents' Behaviors with Behavior Trees: https://towardsdatascience.com/designing-ai-agents-behaviors-with-behavior-trees-b28aa1c3cf8a

PFSM vs Behaviour Tree

FSM are better for simpler tasks with fewer conditions



Behaviour trees can model higher level tasks and can be divided in simpler and simpler tasks.



Automatic modular design of robot swarms using behavior trees as a control architecture: http://iridia.ulb.ac.be/supp/IridiaSupp2020-009/index.html

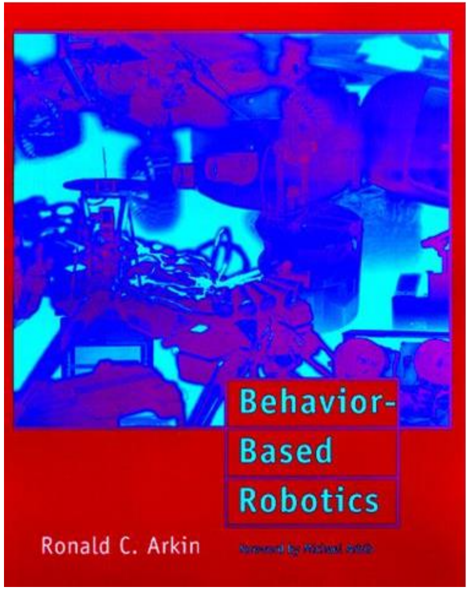
Summary

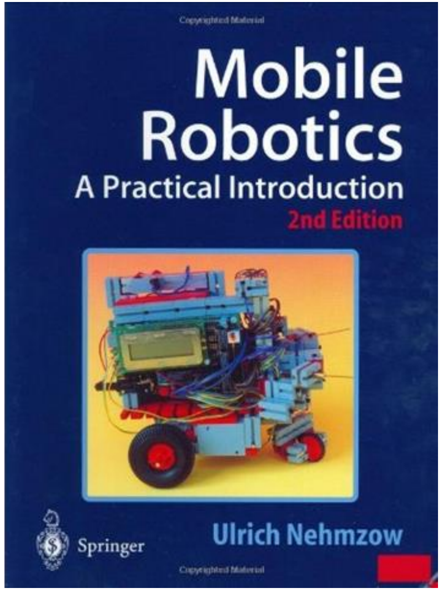
- What is behavioural robotics
- Braitenberg vehicles:
  - Achieving behaviours by coupling sensors and motors
- Combining robot behaviours
  - Vector summation
  - Subsumption architecture (1986)
- Learning robot behaviours
  - Supervised Learning
  - Self learning with evolutionary algorithms
- Automatic composition of robot behaviours
  - Finite State Machines
  - Probabilistic FSM
  - Behaviour Trees

Recommended reading

Ronald C. Arkin, “Behavior-Based Robotics”, MIT Press, 1998

Ulrich Nehmzow, “Mobile Robotics: A Practical Introduction”, Springer, 2nd edition, 2002





Thank you for listening!
Any questions ?

CMP3103M
Autonomous Mobile Robotics

Lecture 6: Navigation

•••

Dr. Athanasios Polydoros

Today

Contents:

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

Learning outcomes:

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

Quizz

Connect PollEveryehere

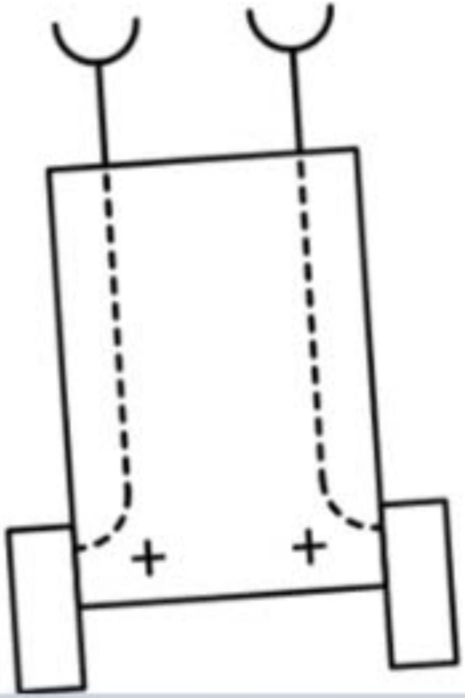
https://pollev.com/athanasiospolydoros472

Reactive behaviours of robot require the defintion of a goal/plan

True

False

What would be the behaviour of the illustrated Braitenberg vehicle?



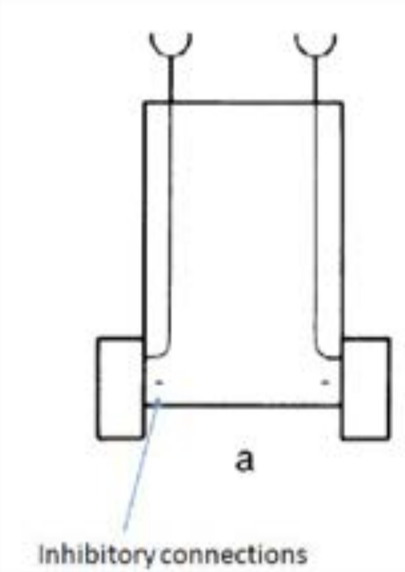
Fear

Aggression

Love

Exploration

What will be the behaviour of this vehicle?



Fear

Aggression

Love

Exploration

Self learning of behaviours can be achieved with:

Finite State Machine

Evolutionary Algorithm

Behaviour Tree

Execution order of tasks can be defined in

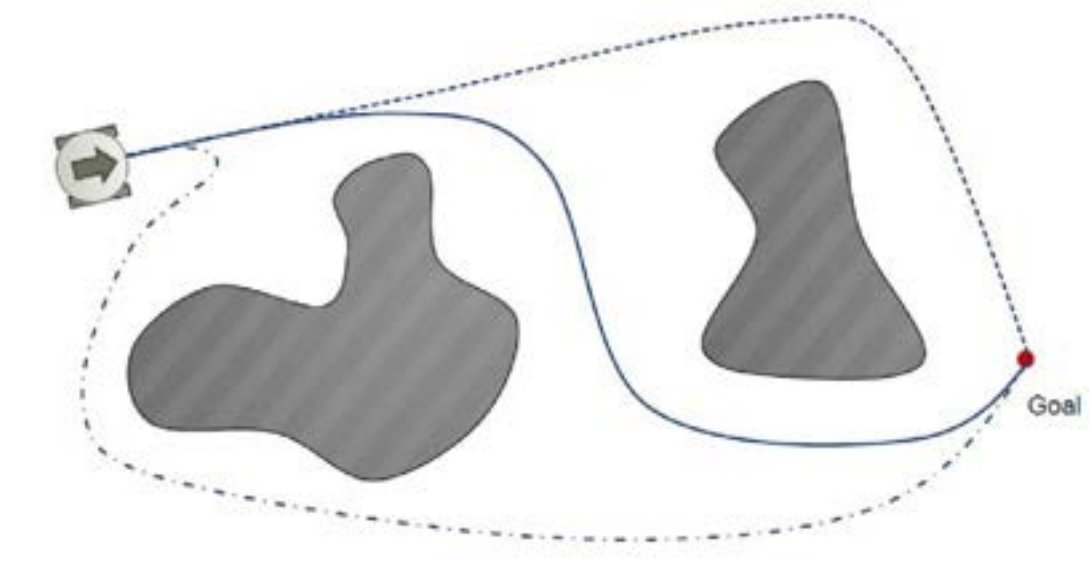
Finite State Machines

Behaviour Trees

Navigation

Navigation – key questions

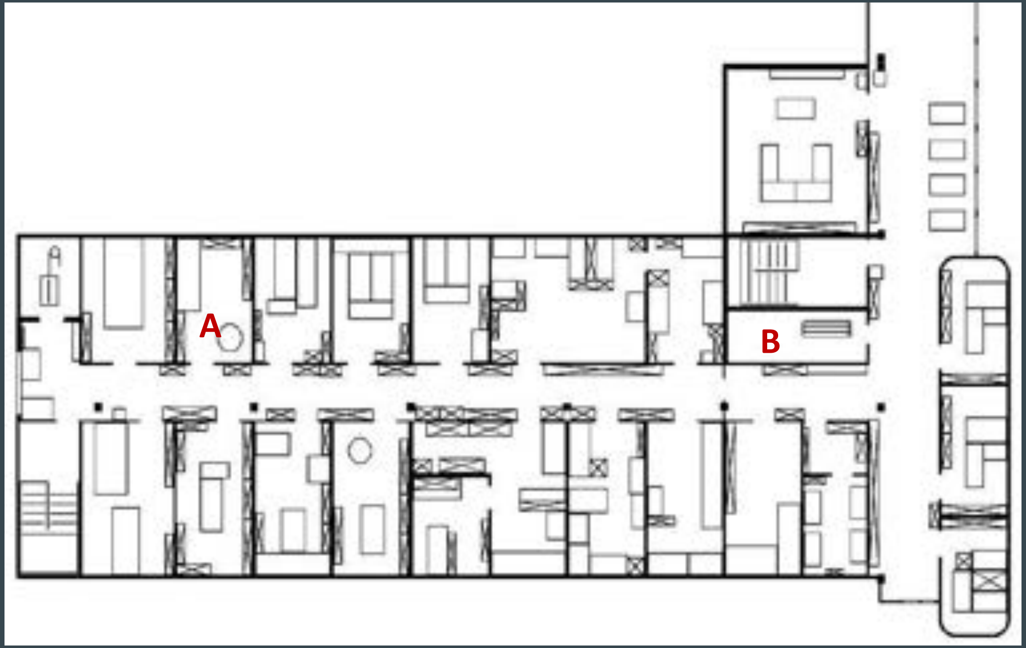
- Where am I?
- Where do I go?
- How do I get there?



Navigation - Getting from A to B

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

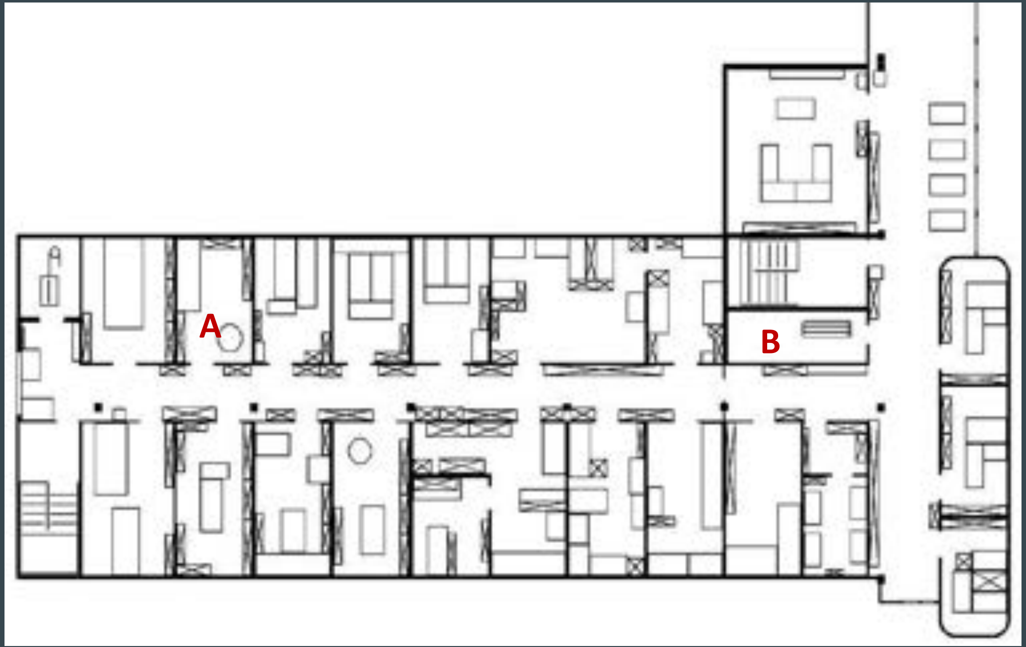
- Task – start at A and reach goal B:
  - Without hitting obstacles
  - Detect arrival at goal



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?

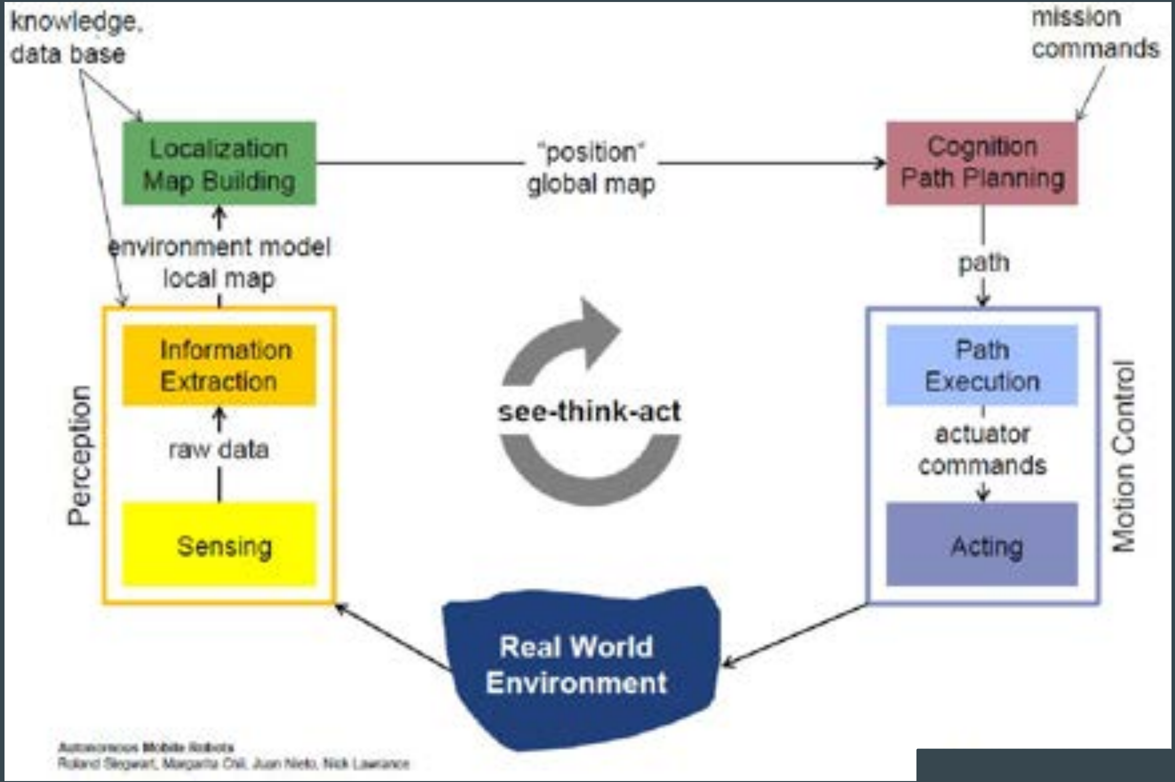


Robot navigation

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

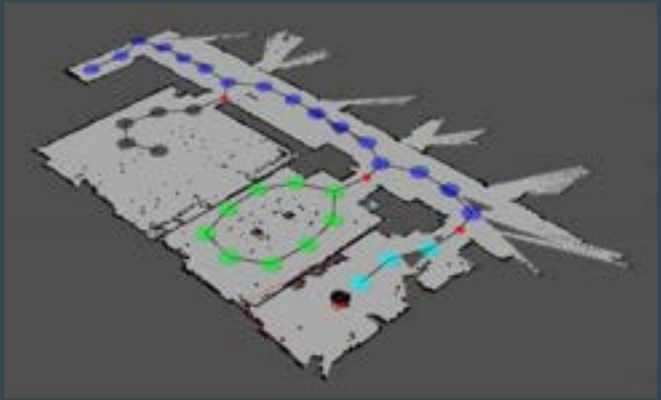


Types of navigation

- 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

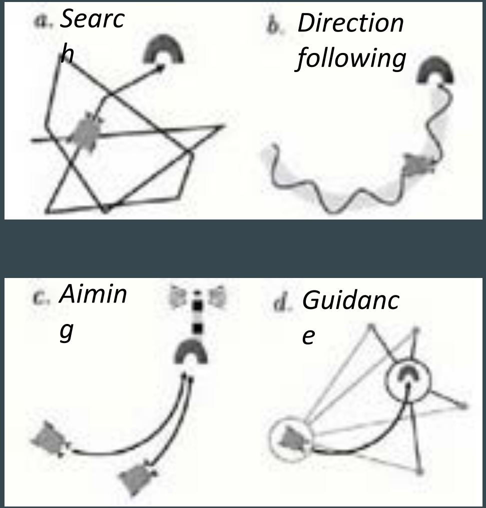
- e.g. compass direction, or trail following
- Finds goal from one direction

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

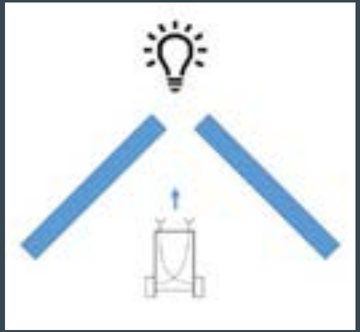
- Use a visual cue at the goal reactively
  - Detect goal from a distance, and maintain a course towards it
  - Landmark navigation / beaconing
- Use a visual cue in the simplest form of navigation
  - Remember visual surroundings to recognise when you arrive at the goal
- More generally
  - May be able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

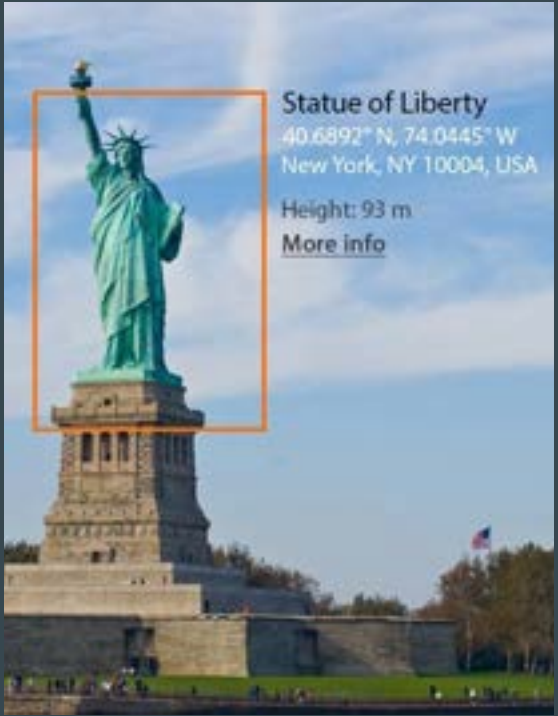
- More complex planning

  is needed in adverse conditions



Problems with landmark recognition

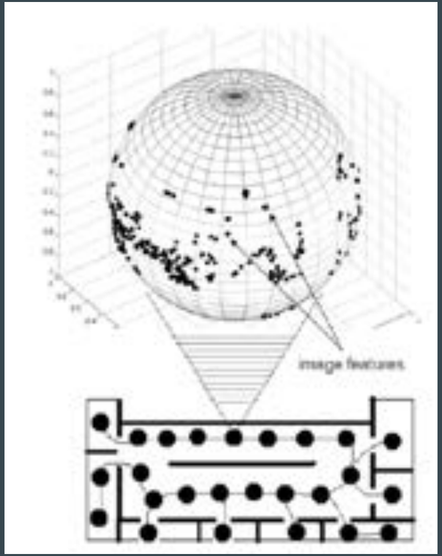
- How to define a landmark?
- How to match landmarks between current scene and memory?
  - Correspondence problem
- Robot's view must be aligned with recorded landmark



Feature based recognition

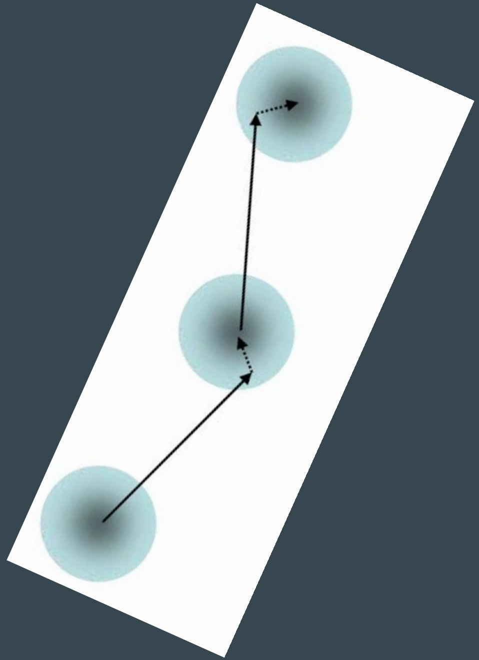
- Visual features detected via on-board camera sensors can be used to describe a location

- 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

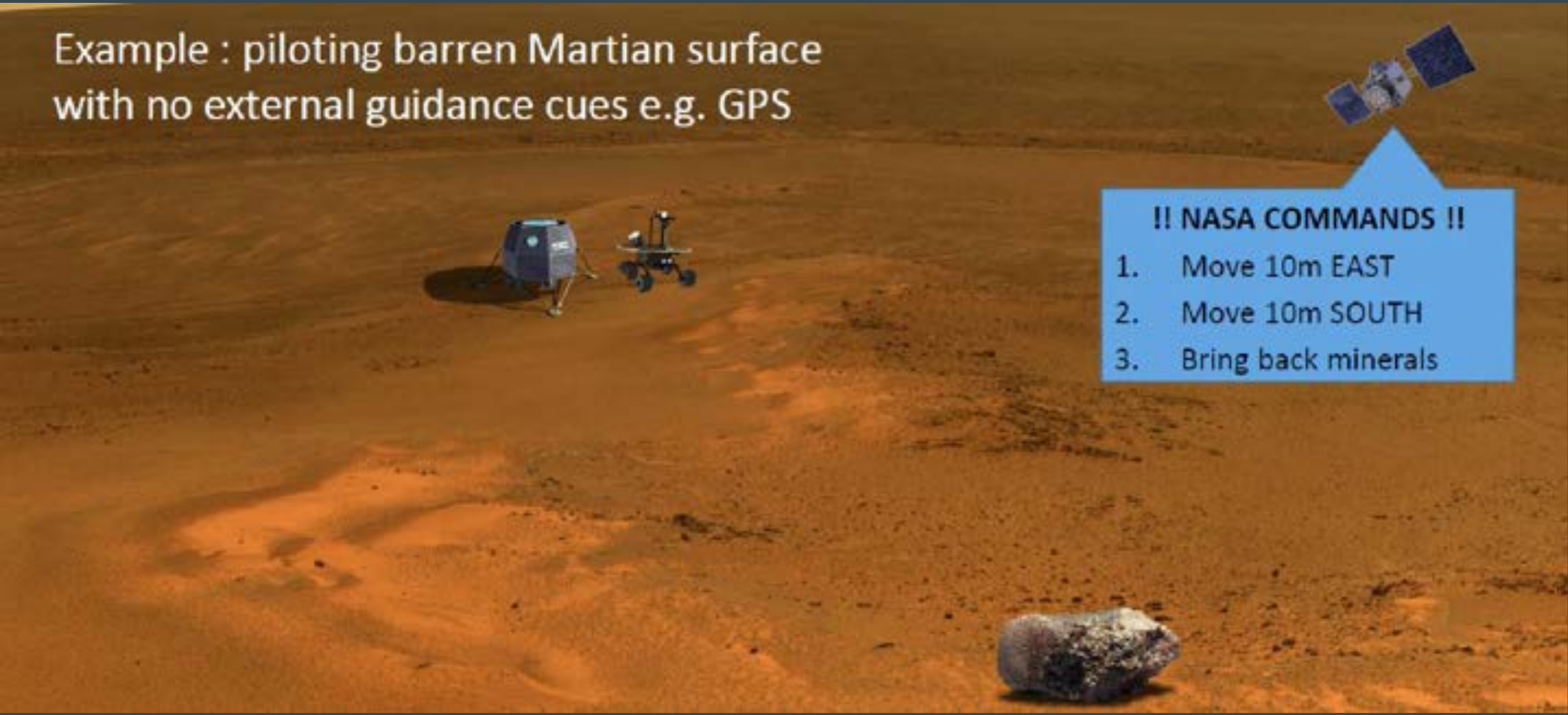
- Recognising where you are with respect to previous memories
  - Continues to be a problem in robotics
- **Loop closure:** ability to recognise a location even if perceived from a different pose
  - Allows correction of robot's position when simultaneously mapping
- "Kidnapped robot problem" can result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

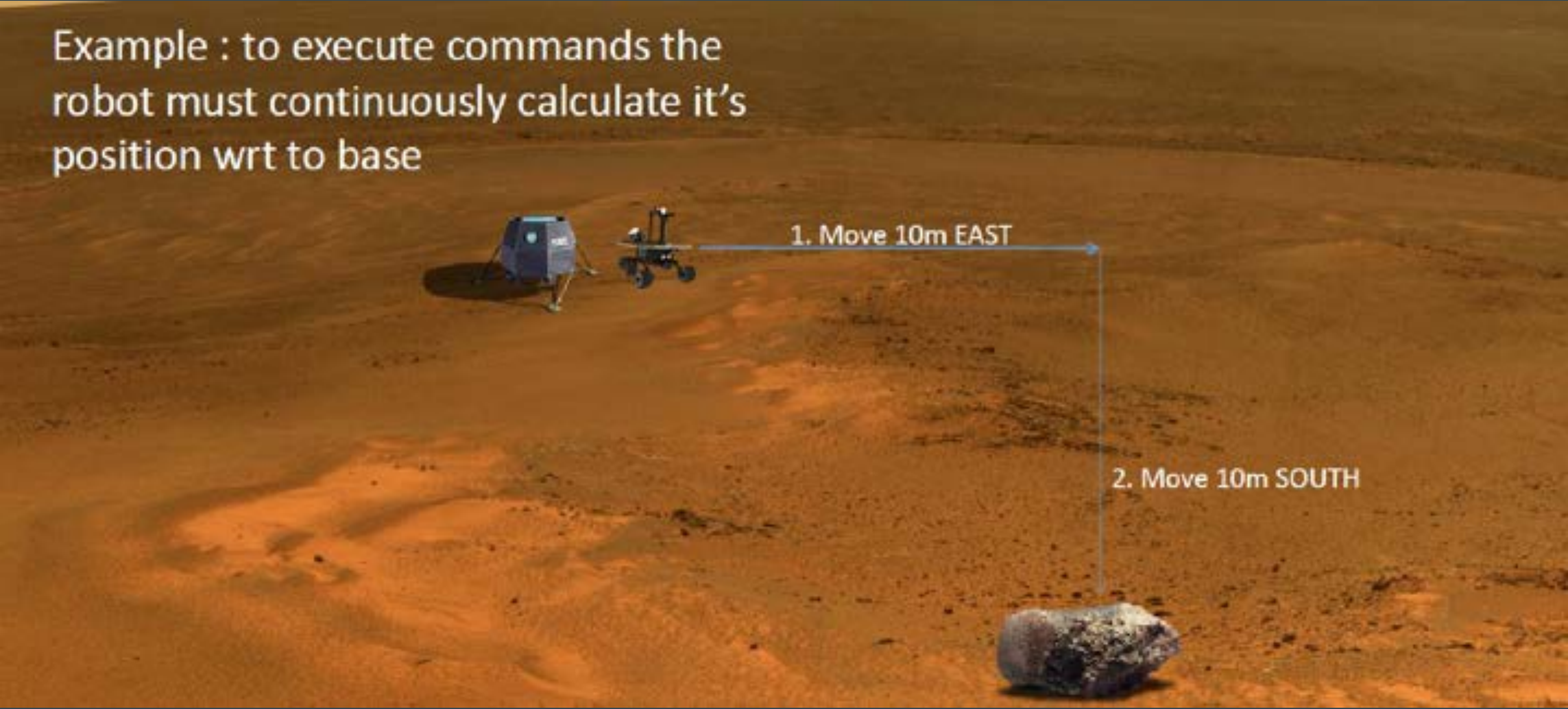
Odometry

Odometry / dead reckoning

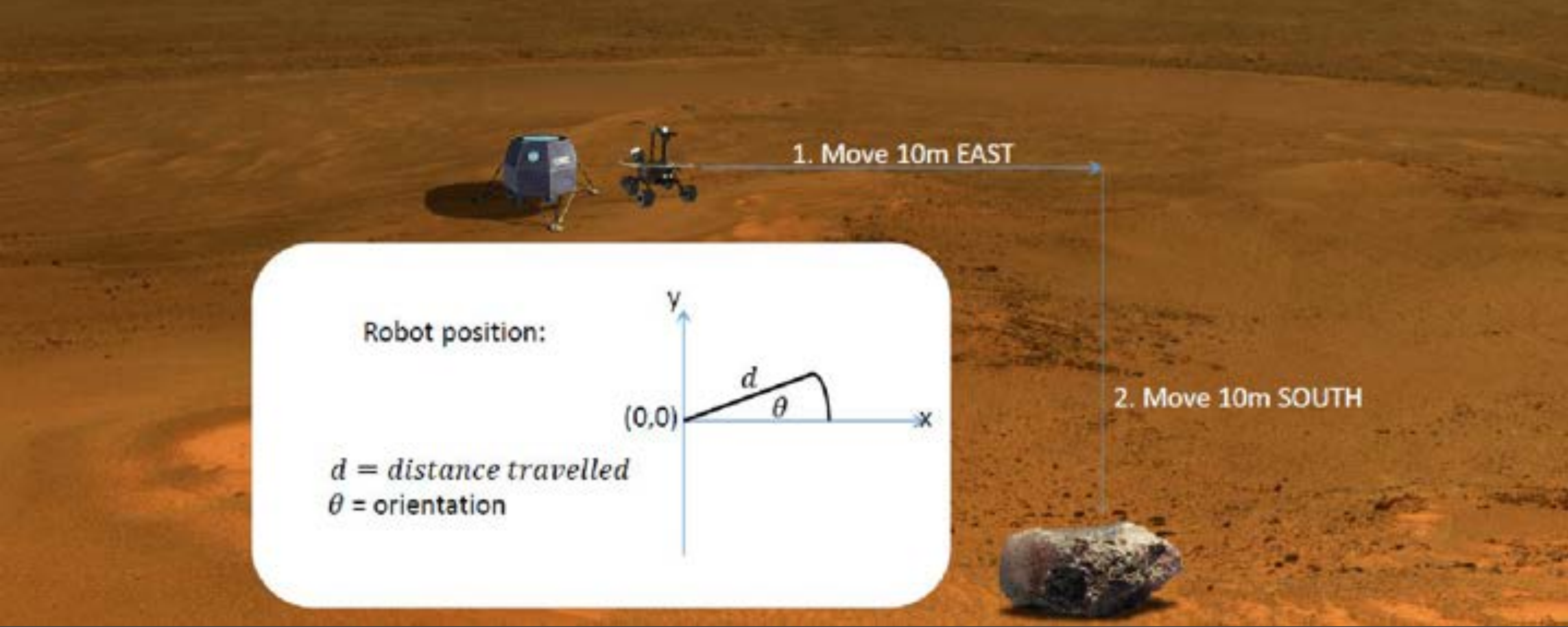
- **Approximate location** of a robot can be obtained by **repeatedly** computing the **distance moved**, and the **change in direction**, from the **velocity of the wheels** over a **short period of time**
- Also called **deduced reckoning** or **dead reckoning**
- Robot motion recovered by integrating 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

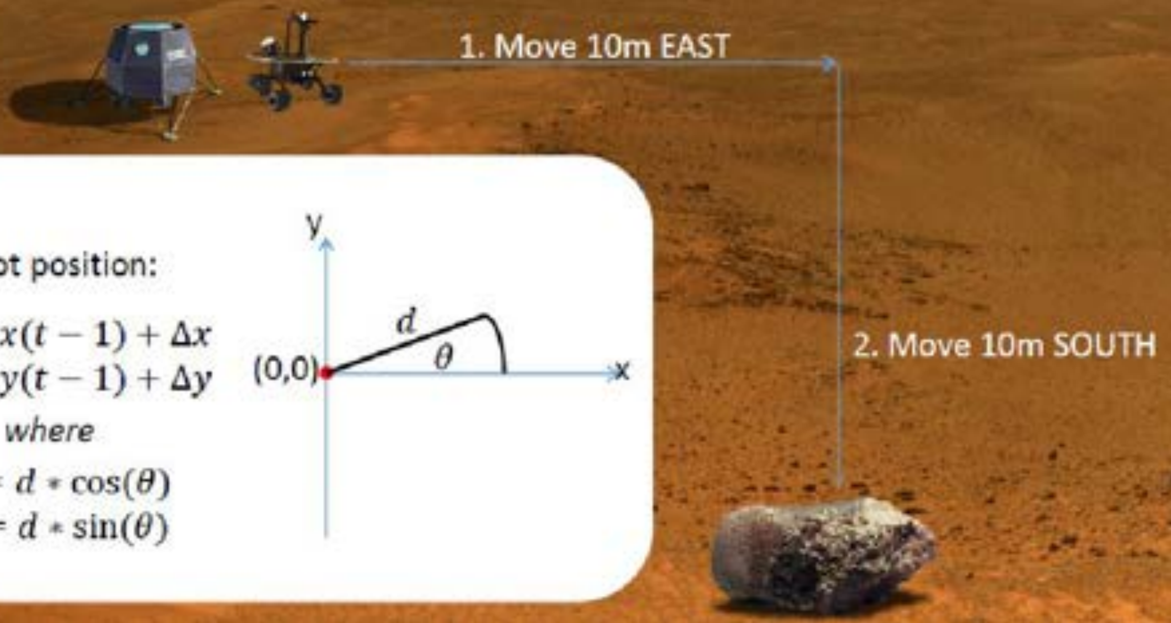

Example : piloting barren Martian surface with no external guidance cues e.g. GPS
!! NASA COMMANDS !!
1. Move 10m EAST
2. Move 10m SOUTH
3. Bring back minerals


Example : to execute commands the robot must continuously calculate it's position wrt to base
1. Move 10m EAST
2. Move 10m SOUTH

Example : formulation



Example : formulation



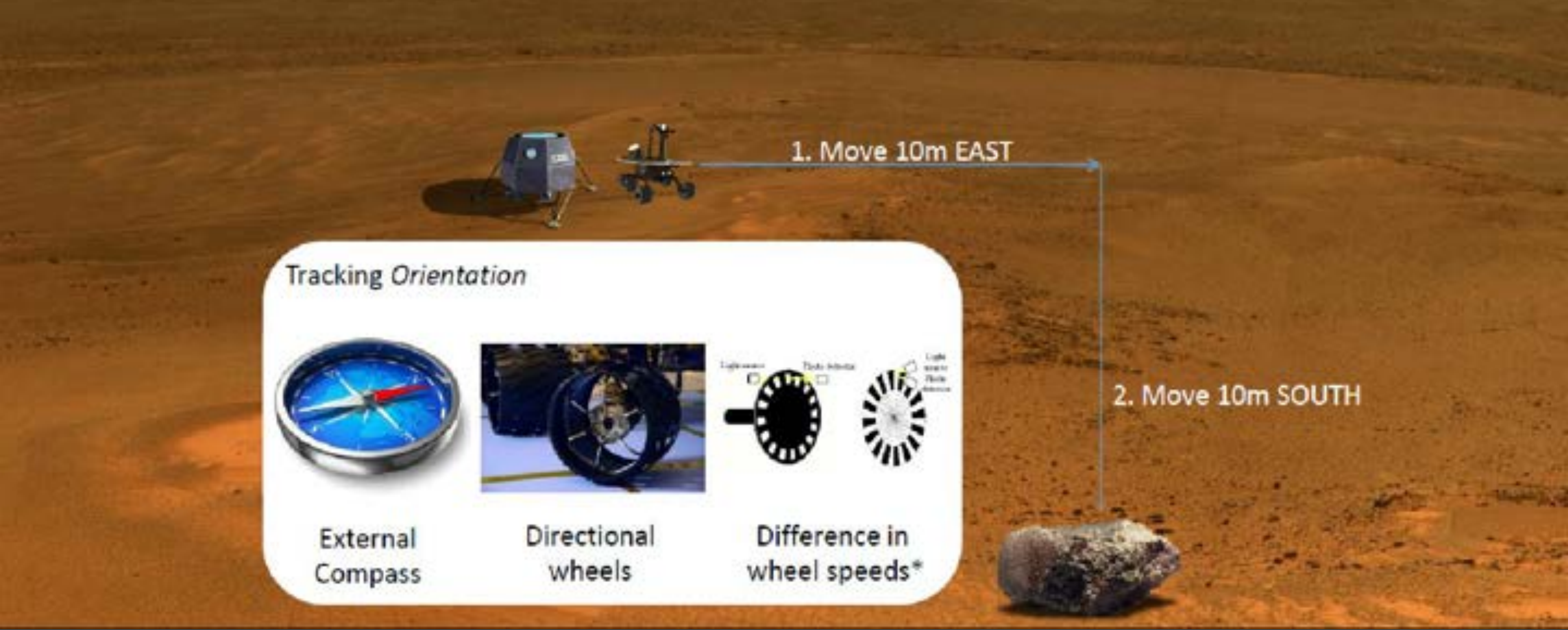
Robot position:

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

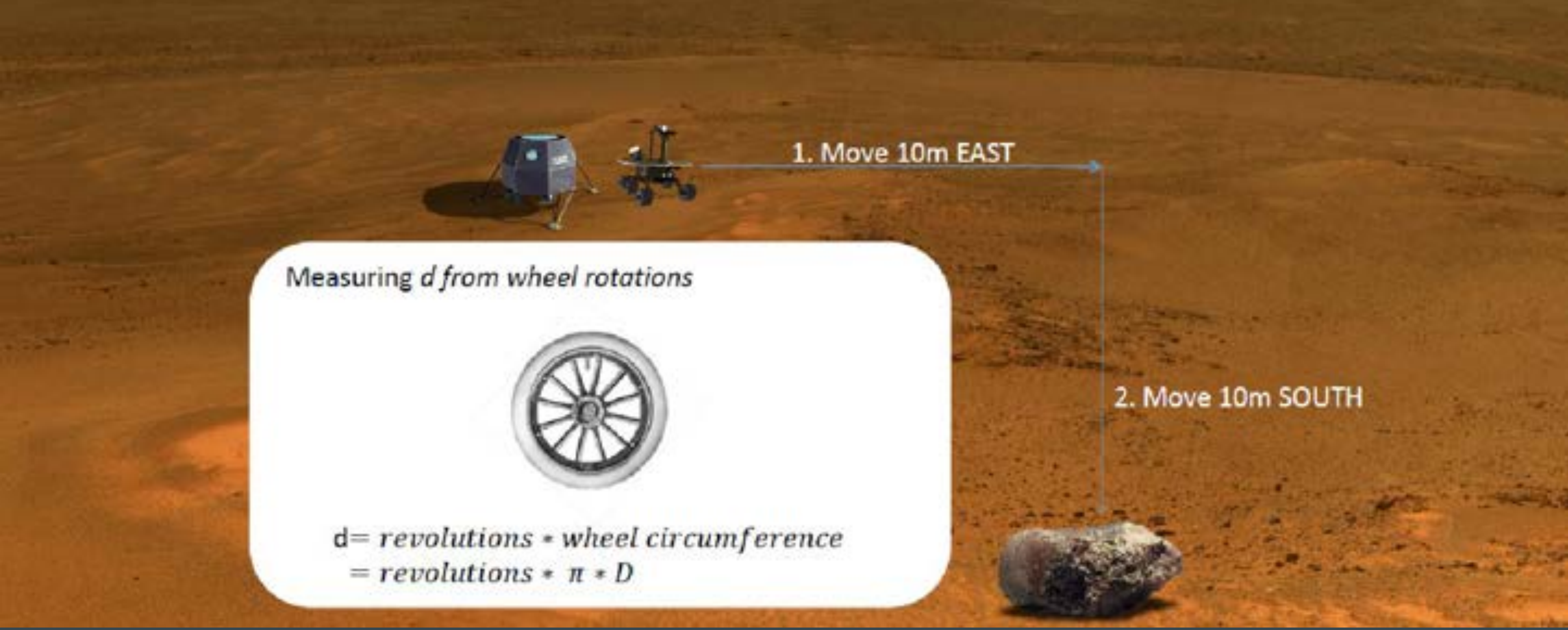
where

$$\Delta x = d * \cos(\theta)$$
$$\Delta y = d * \sin(\theta)$$

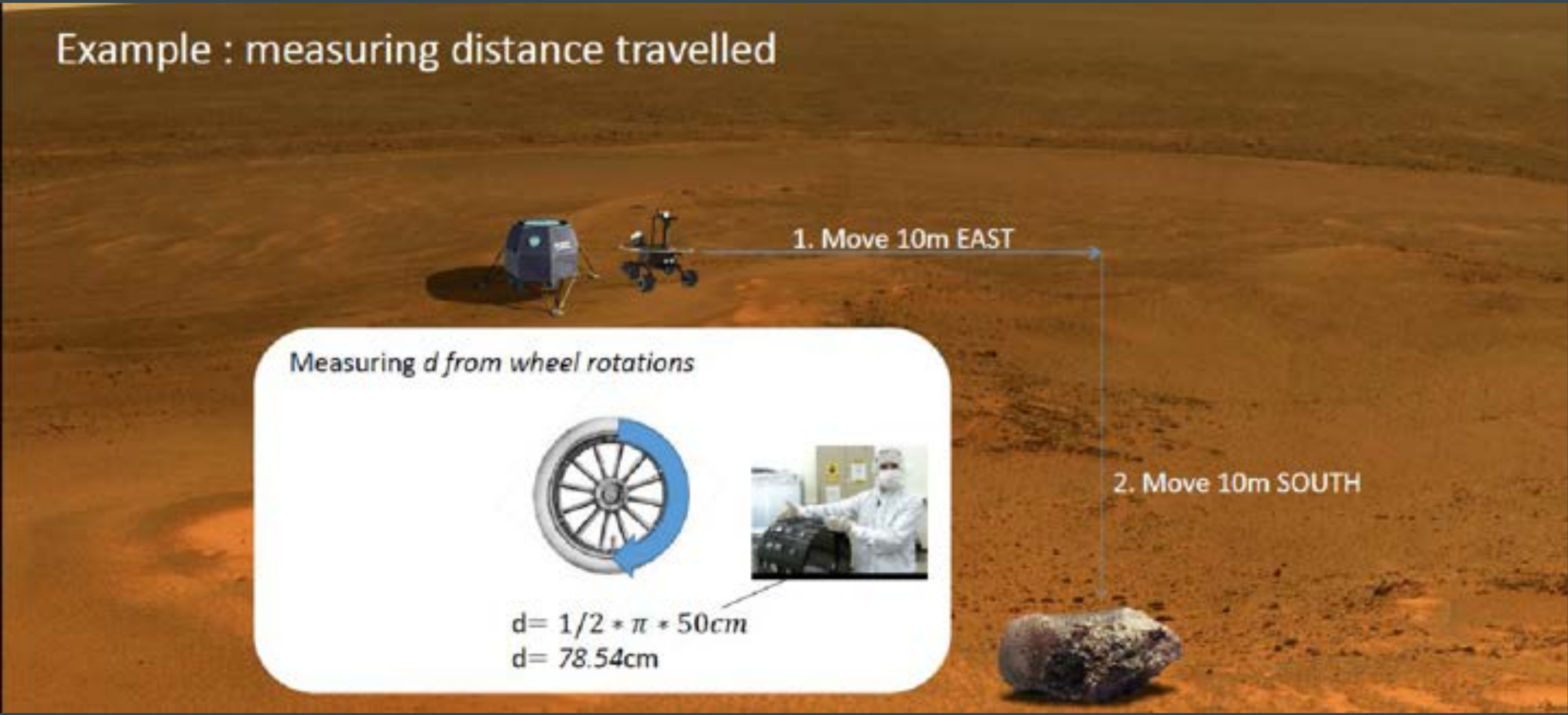
Example : monitoring orientation



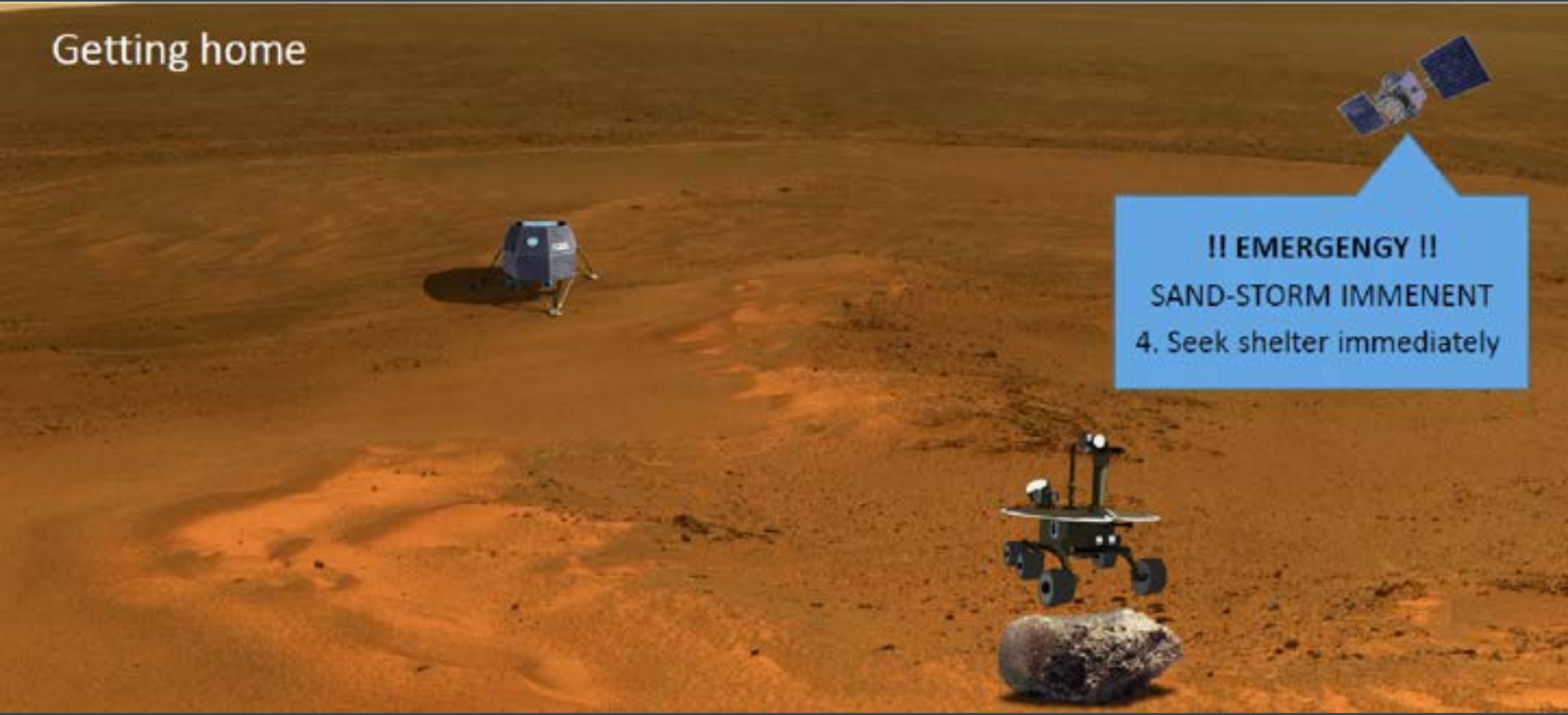
Example : measuring distance travelled



$$d = revolutions * wheel\ circumference$$
$$= revolutions * \pi * D$$

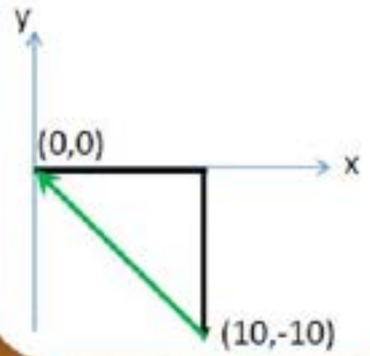

Example : measuring distance travelled
1. Move 10m EAST
2. Move 10m SOUTH
Measuring *d from wheel rotations*
d= $1/2 * \pi * 50cm$
d= *78.54*cm

Getting home



Getting home

Calculating the home vector

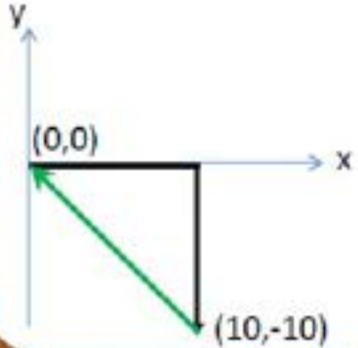


Cartesian format:

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

Getting home

Calculating the home vector



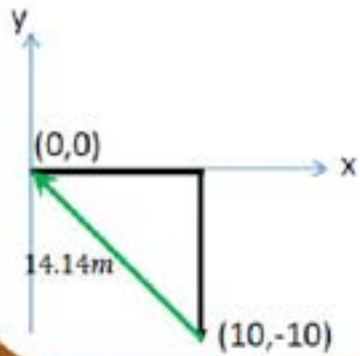
Polar format:

HV= $[r, \theta]$

where r is distance home,
and θ is bearing from x

Getting home

Calculating the home vector



Polar format:

HV= $[14.14, \theta]$

where r is distance home,
and θ is bearing from x

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

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

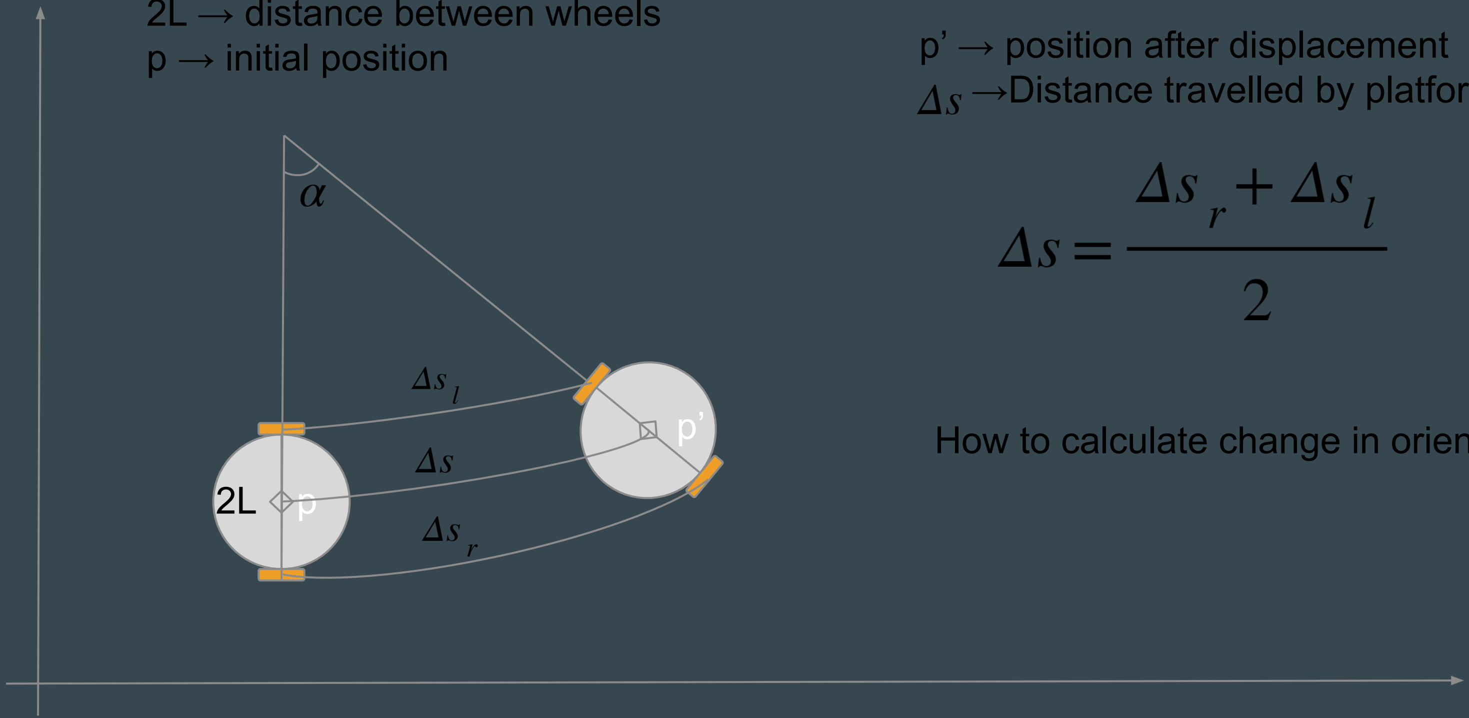
r= $14.14m$

Odometry – differential drive robot

2L → distance between wheels
p → initial position

p’ → position after displacement
Δs →Distance travelled by platform

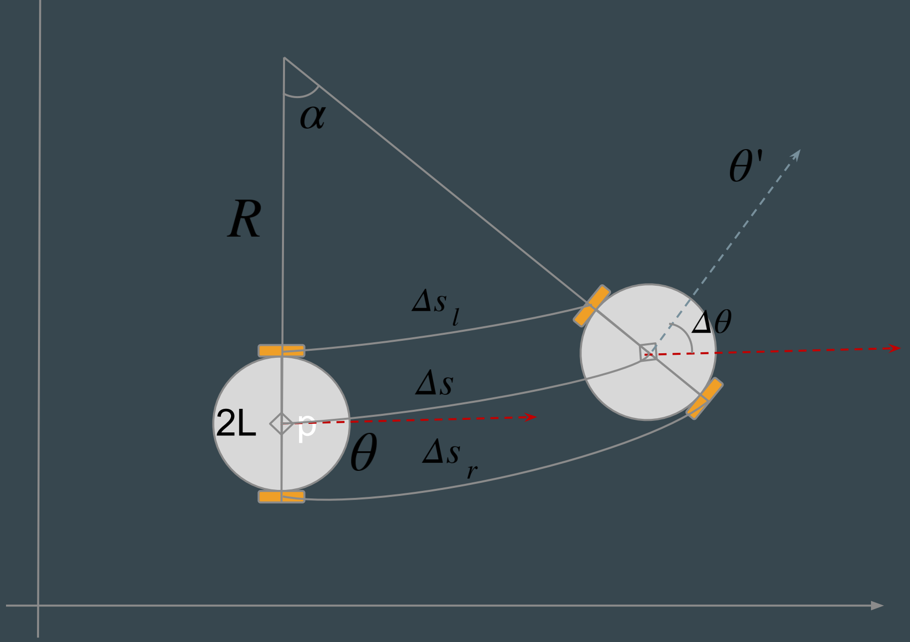
$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$



How to calculate change in orientation?

Odometry – differential drive robot

2L → distance between wheels
p → initial position



p’ → position after displacement

Δs →Distance travelled by platform

$$\Delta\theta = \alpha$$

Δs_l Δs Δs_r Arcs of circle with the same center and different radius

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

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

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

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

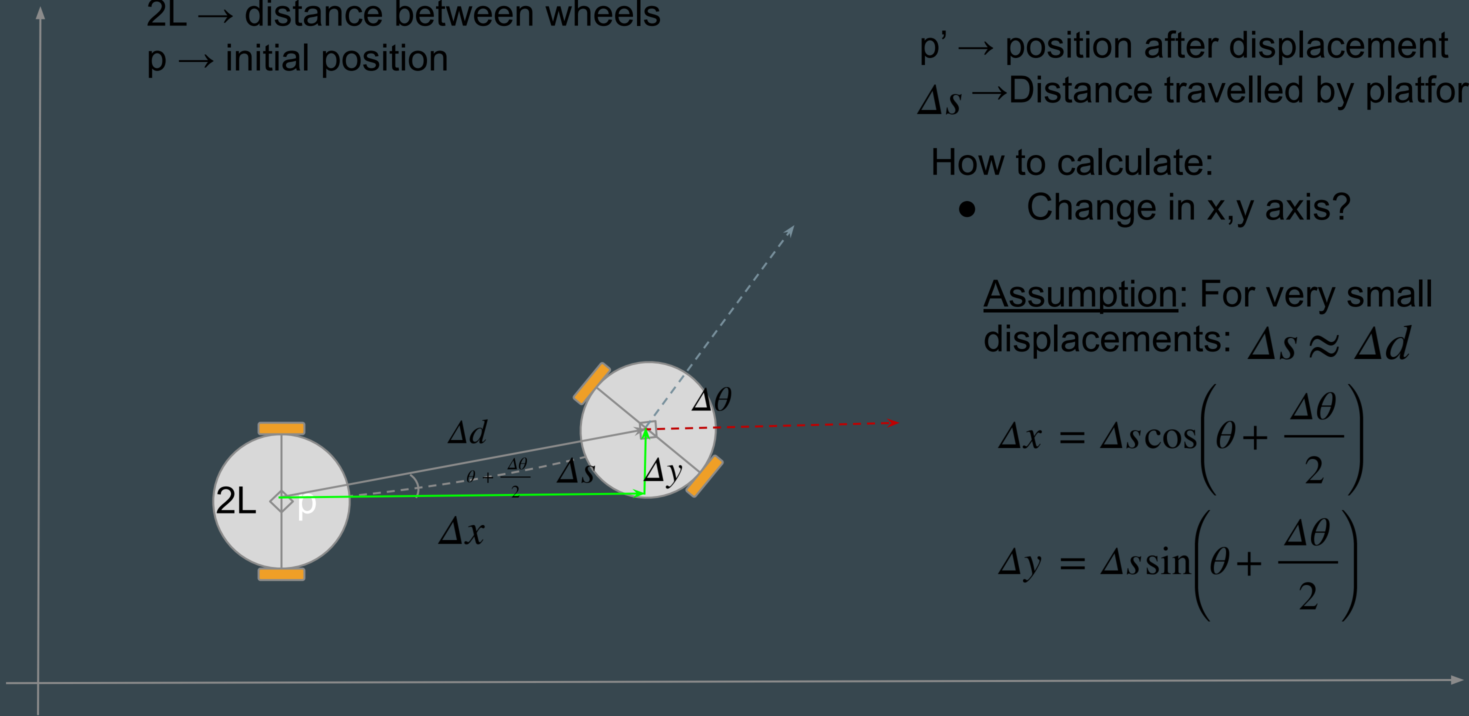
Substitute radius R in Δs_l

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

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

Odometry – differential drive robot

2L → distance between wheels
p → initial position



p’ → position after displacement
Δs →Distance travelled by platform

How to calculate:

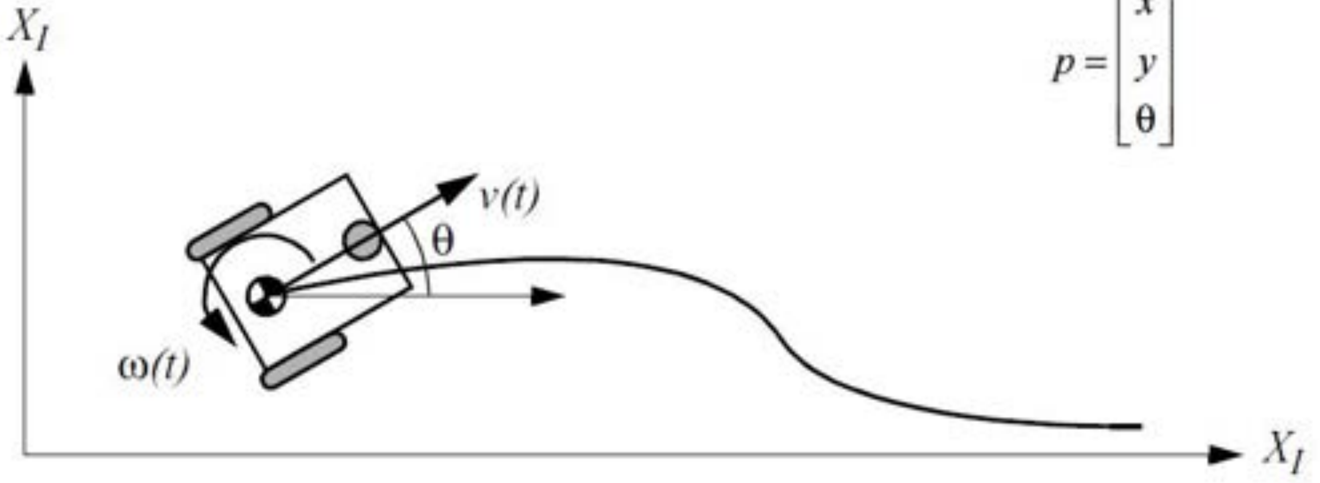
- Change in x,y axis?

<u>Assumption</u>: 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:

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

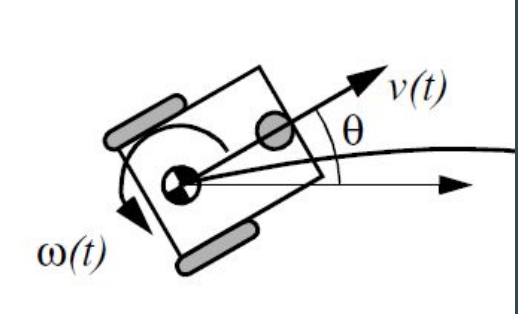
$$\Delta x = \Delta s \cos(\theta + \Delta\theta/2)$$

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

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

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

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



where

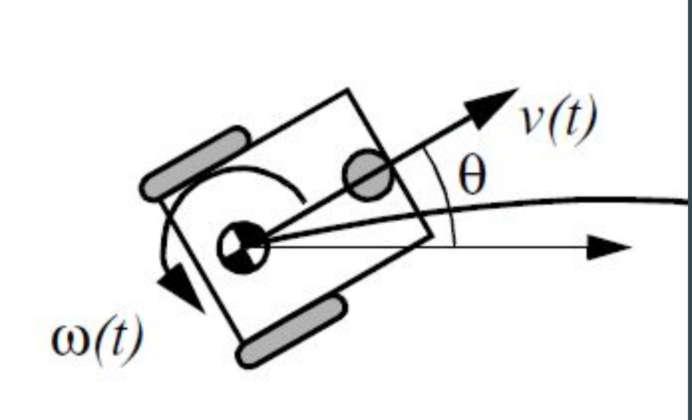
$(\Delta x;\Delta y;\Delta\theta)$ = path traveled in the last sampling interval;

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

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

Mobile robot odometry

Putting it all together….

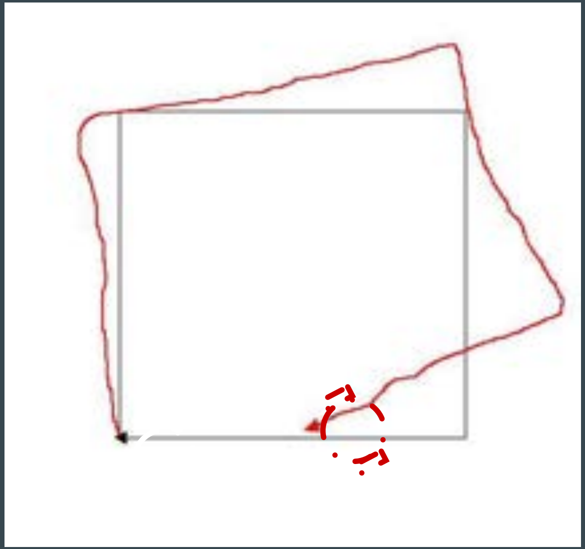


$$p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix}$$

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

Problems with Odometry

- Inaccuracies / noise cause estimated robot position to drift over time



Real path

Measured path

- Solution: use a map!
    - Combine odometry with sightings of known landmarks / environmental features

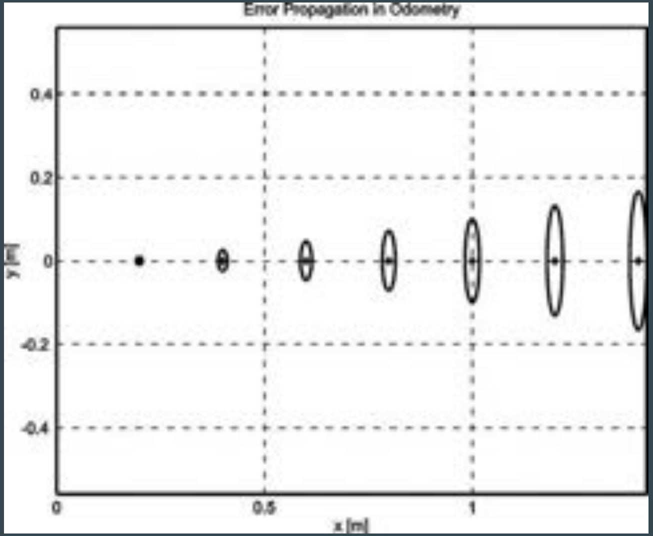
Odometry error types

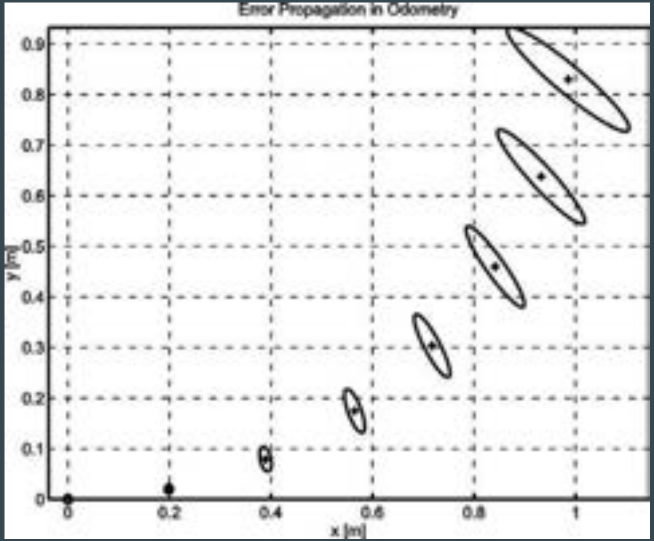
- **Range error**
  - Sum of the wheel motions leads to an error in the integrated path distance of the robot's movement
- **Turn error**
  - Difference of the wheel motions leads to an error in the robot's final orientation
- **Drift error**
  - Difference in the error of the wheels leads to an error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

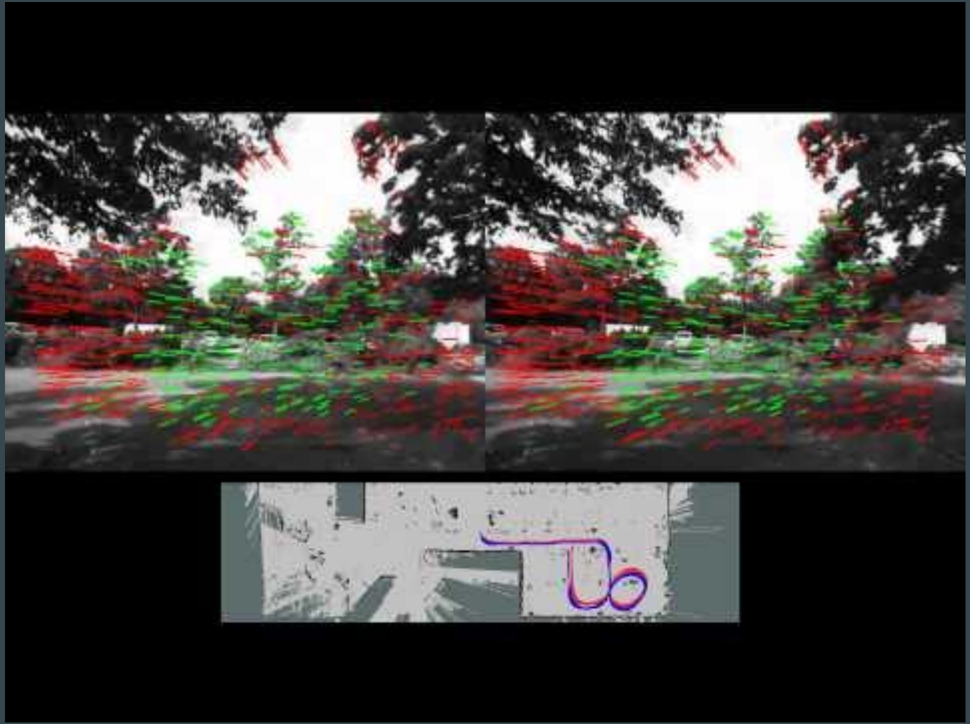
Error propagation in odometry





Visual odometry

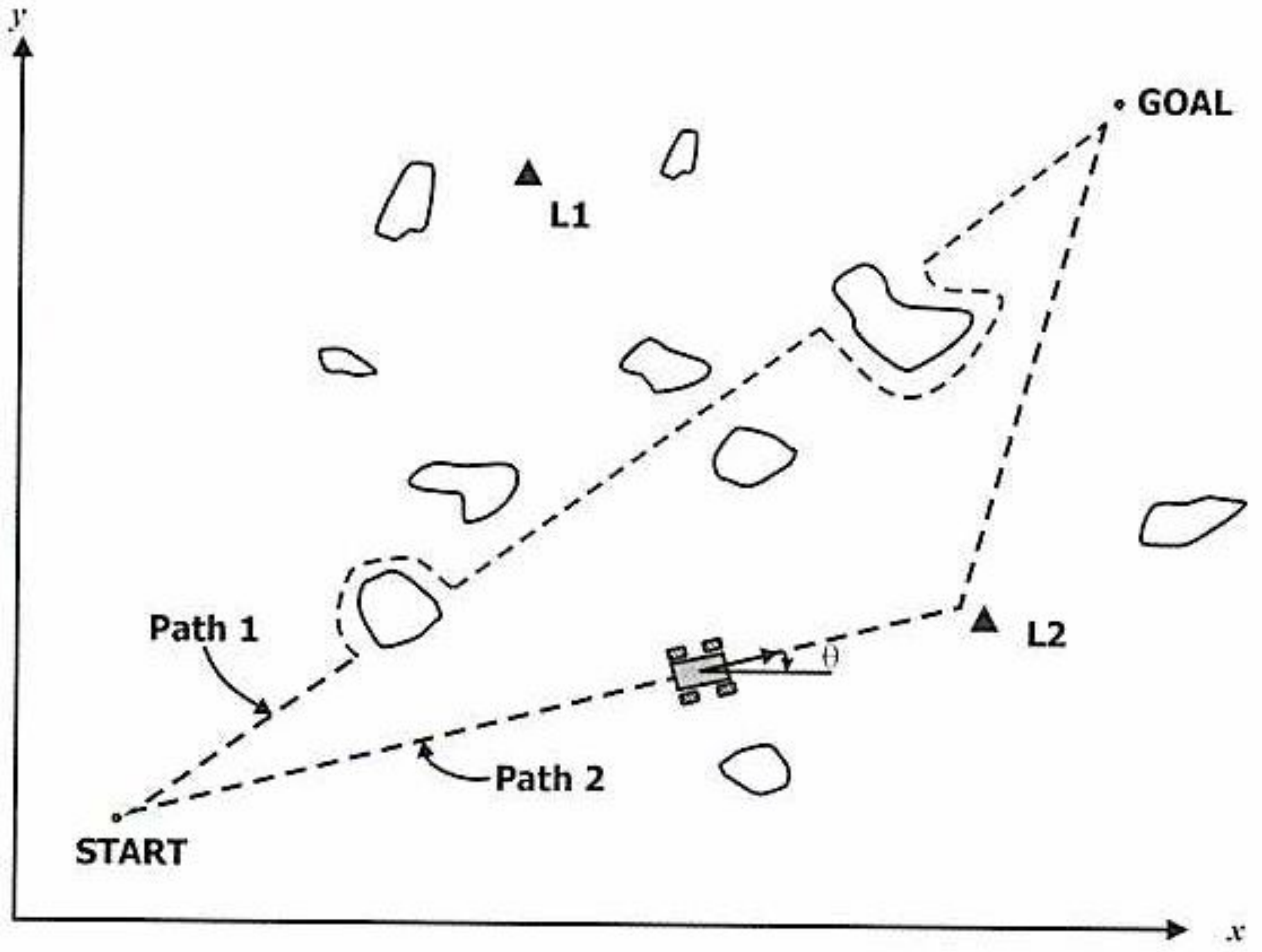
- Odometry is not limited to wheel encoders
- Consecutive camera images can be used to estimate velocity



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

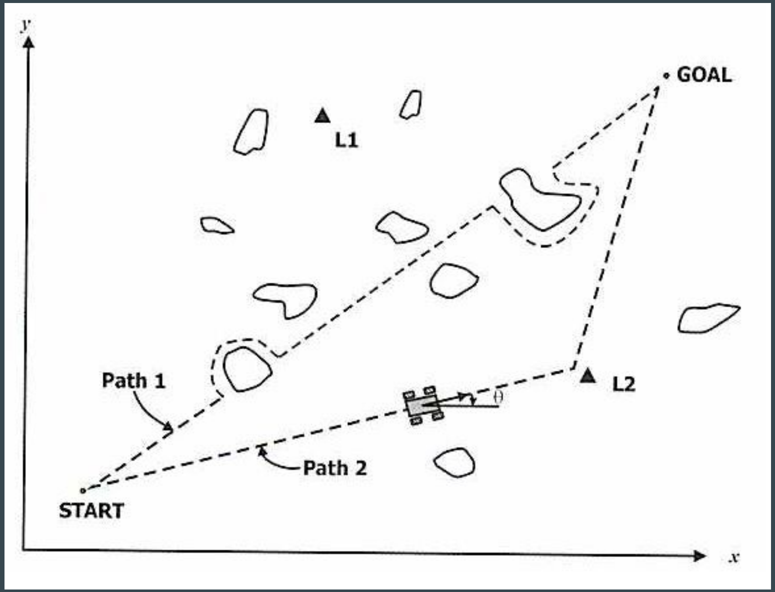


y
GOAL
L1
Path 1
L2
θ
Path 2
START
x

Navigation by vision and compass

Path 1

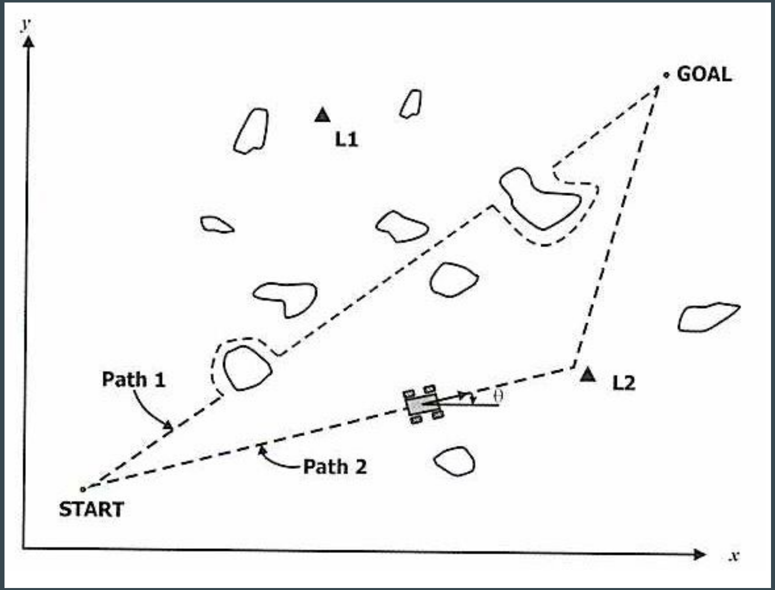
- Assuming visible goal, unknown distance
- Use vision to recognise goal
- Use compass to maintain heading towards goal
- When obstacle encountered (goal no longer visible):
  - Remember current compass reading
  - Travel around obstacle until the original compass direction is sensed again and goal is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

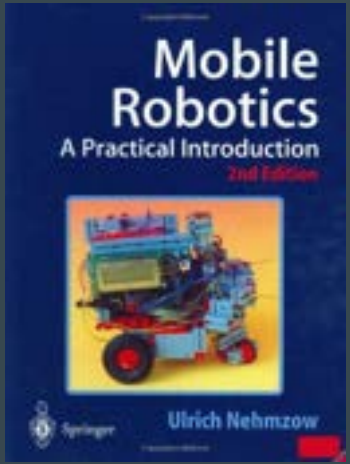
- **A representation of the navigable space**
  - e.g. graph or grid, derived from the global map of the environment
- **A path planner**
  - Use algorithms such as A* or Dijkstra to find the best route to the goal location (a set of waypoints)
- **An "auto-pilot"**
  - Control software to drive between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

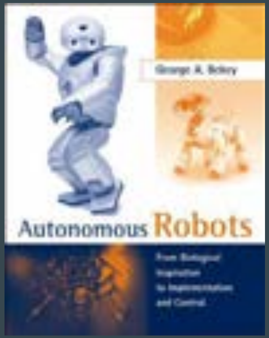
Summary

- Navigation
- Global & Local Navigation
- Odomentry
- 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

Overview of 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

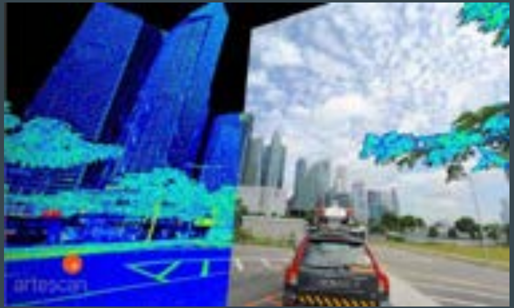
Metric maps

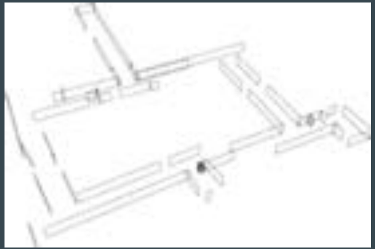
Record the location of objects in an absolute coordinate system

Metric maps – Types

Continuous / “vector” format

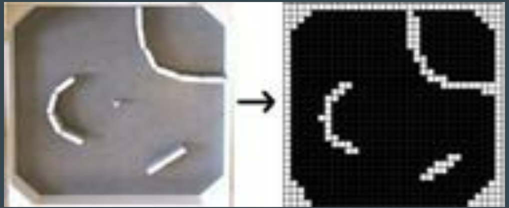
- Points, linear/curved segments, surface patches



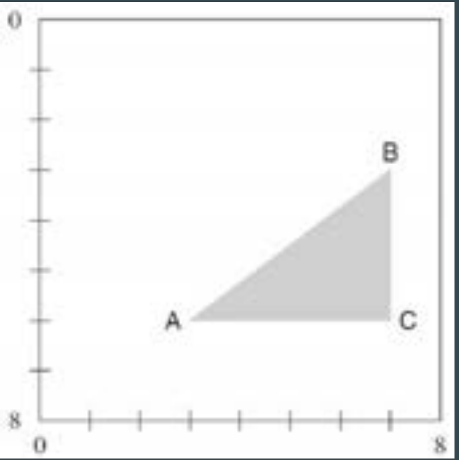


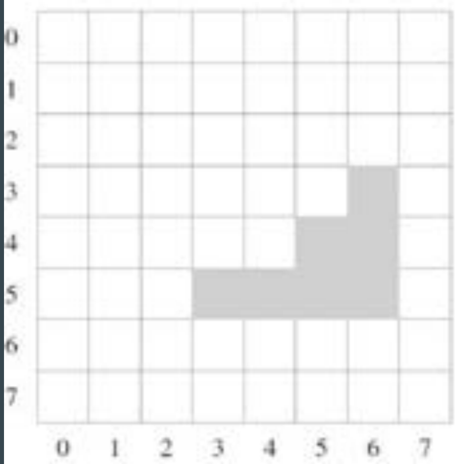
Discrete / “raster” format

- 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, **$P(occ_{x,y})$**

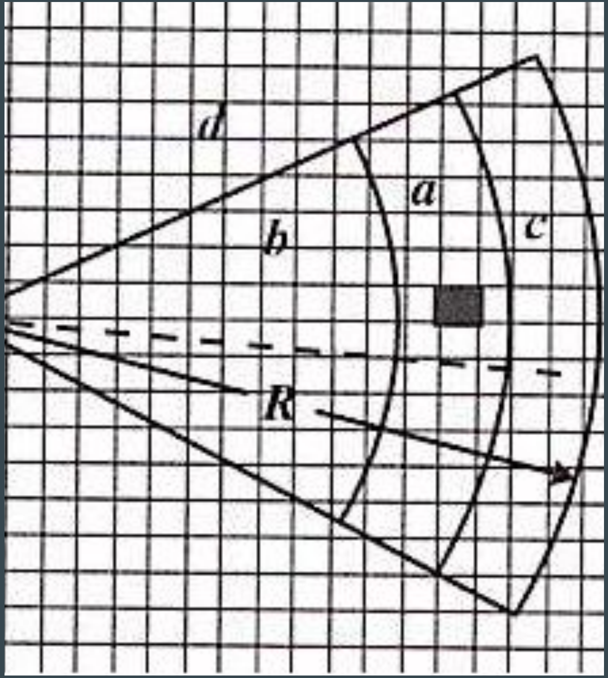
- Updated using current robot pose **(x, y, θ)** and depth measurements from range-finder sensors, e.g. sonar, laser, stereo-vision

	0	1	2	3	4	5	6	7
0								
1								
2						.4	.6	.3
3			.1	.2	.4	.6	.9	.5
4			.3	.6	.5	.7	.8	.6
5			.4	.8	1.0	1.0	.9	.5
6			.2	.3	.2	.3	.3	.1
7								

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):

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

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

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

Bayes' rule

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

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

posterior probability

normalisation factor

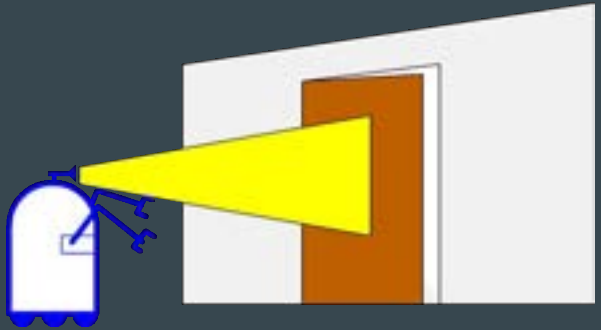
likelihood

prior probability

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

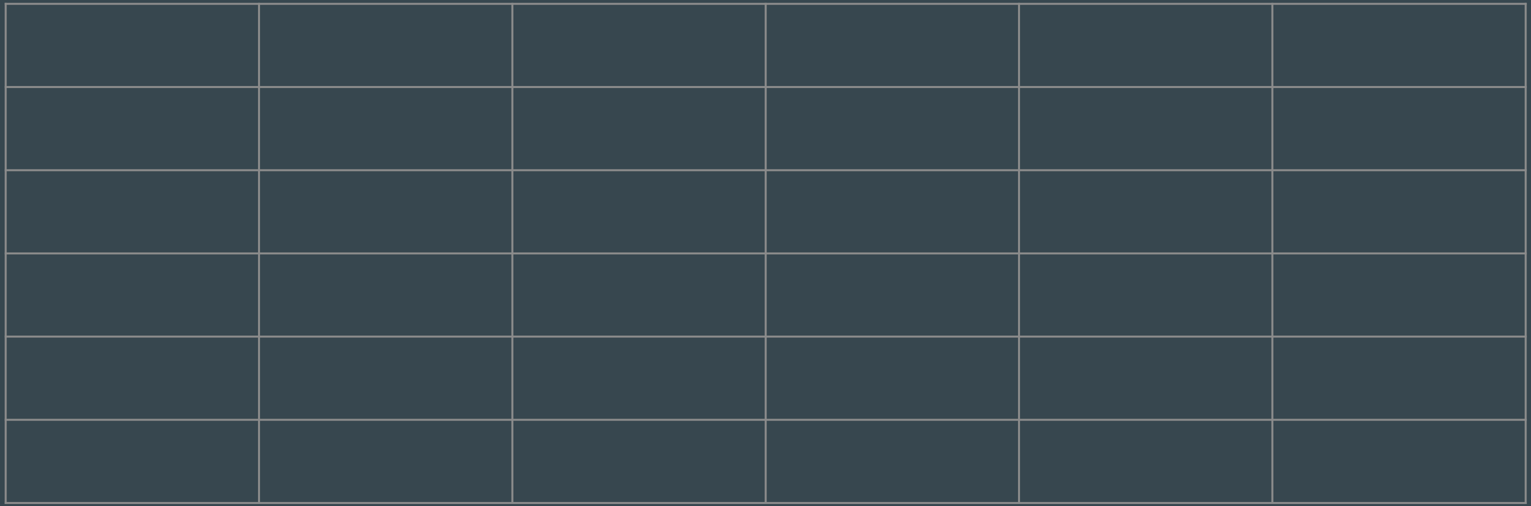


$$P(open \mid z) = \frac{P(z \mid open)P(open)}{P(z)}$$

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

Frontier Exploration Algorithm

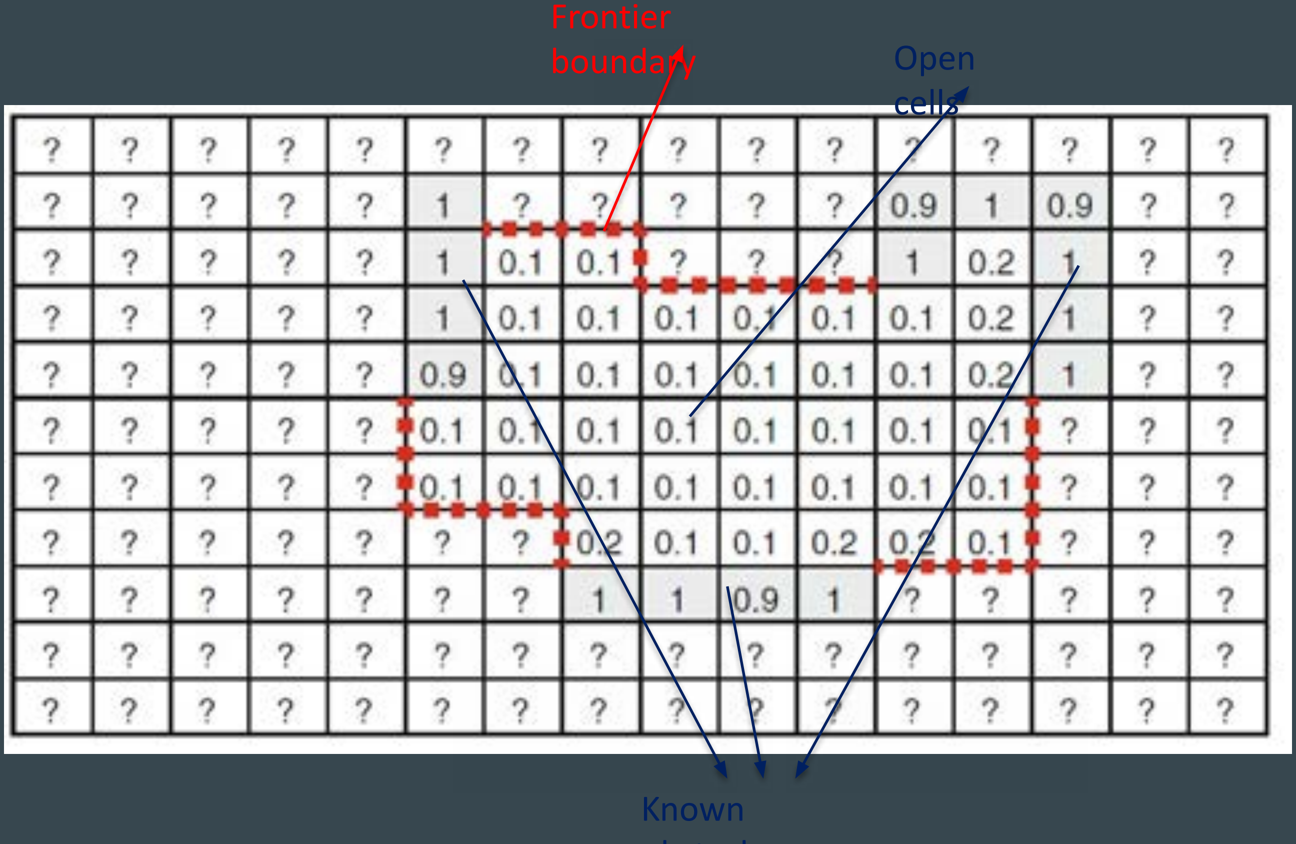
- Method to create discrete occupancy maps and explore the environment



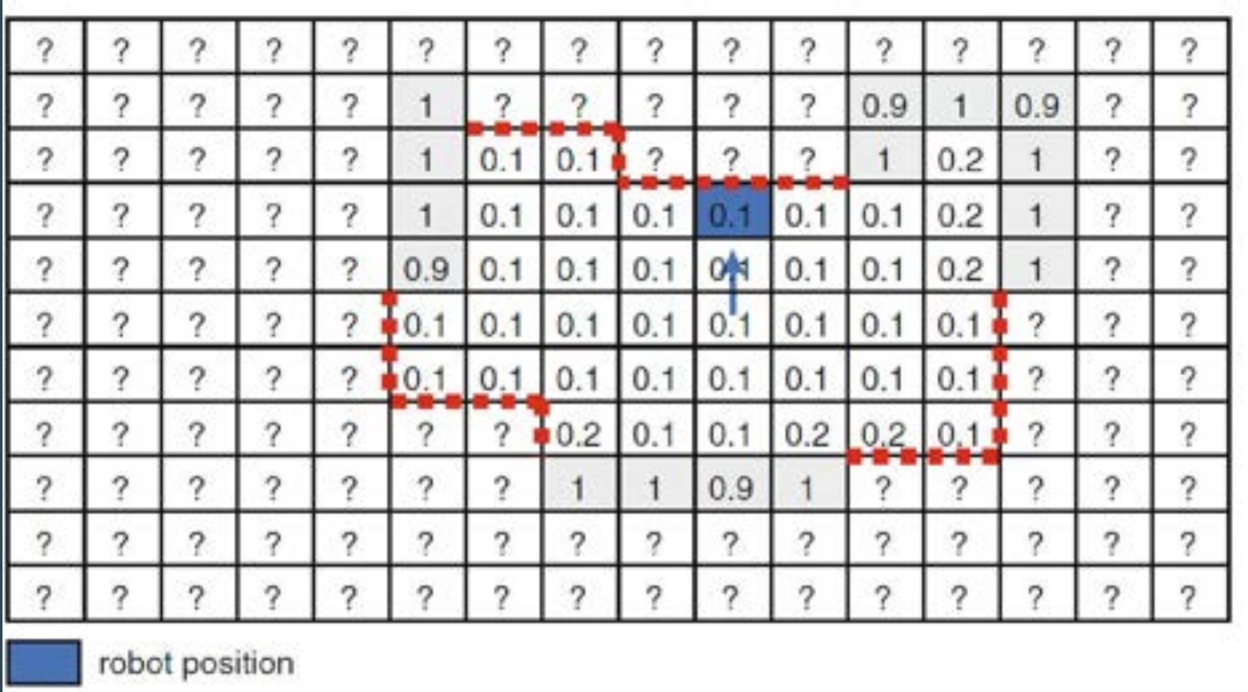
- Find boundaries between explored and unexplored areas
- Drive the robot to explore the boundaries

Frontier algorithm

Frontier Algorithm – Grid Maps with Occupancy Information

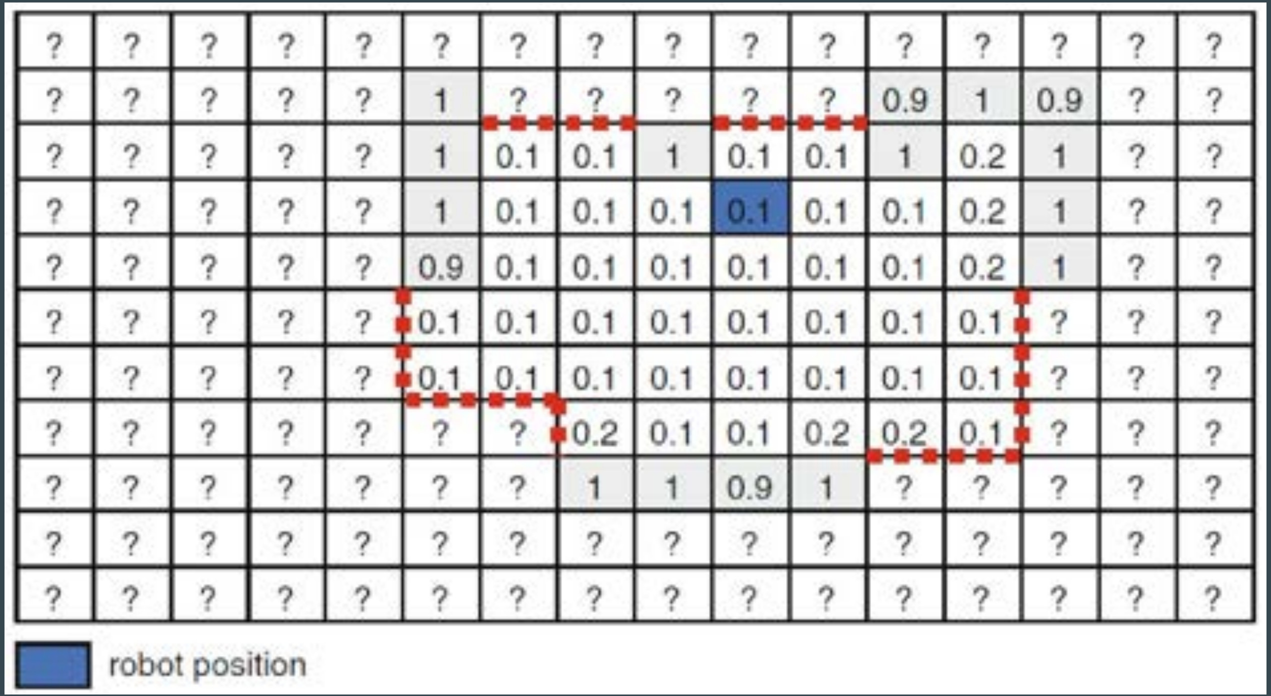


Frontier algorithm



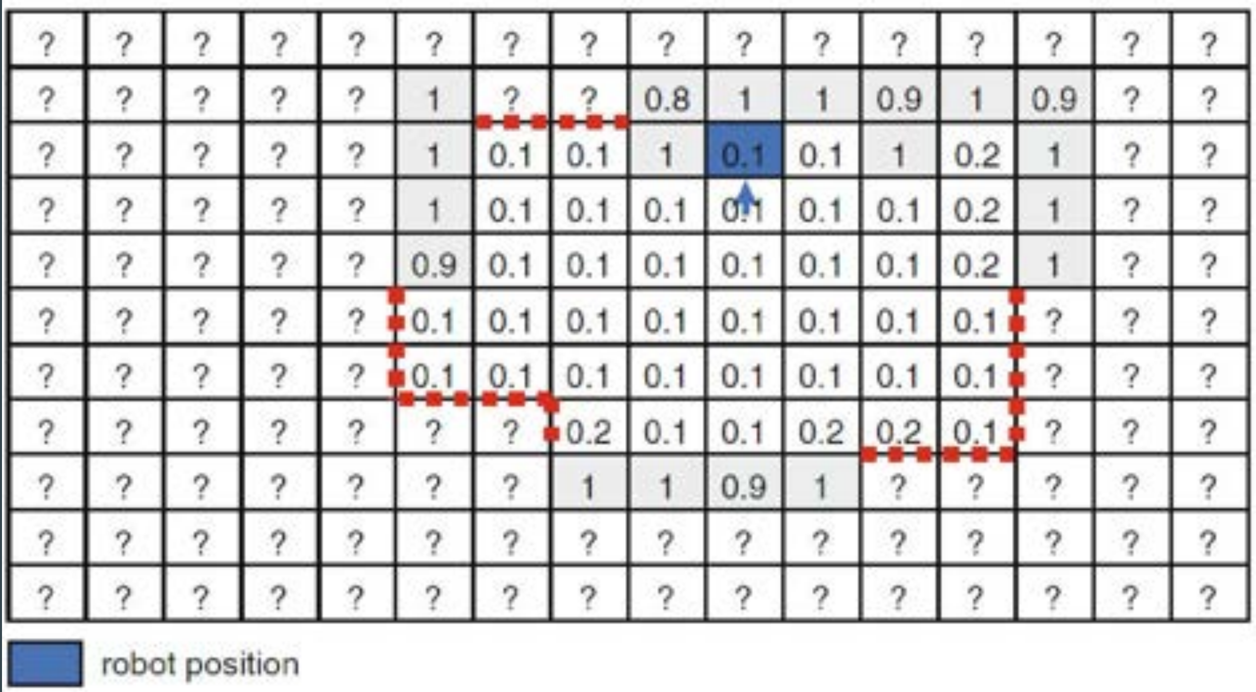
Frontier algorithm

Frontier Algorithm – Grid Maps with Occupancy Information



Frontier algorithm

Frontier Algorithm – Grid Maps with Occupancy Information



Frontier algorithm

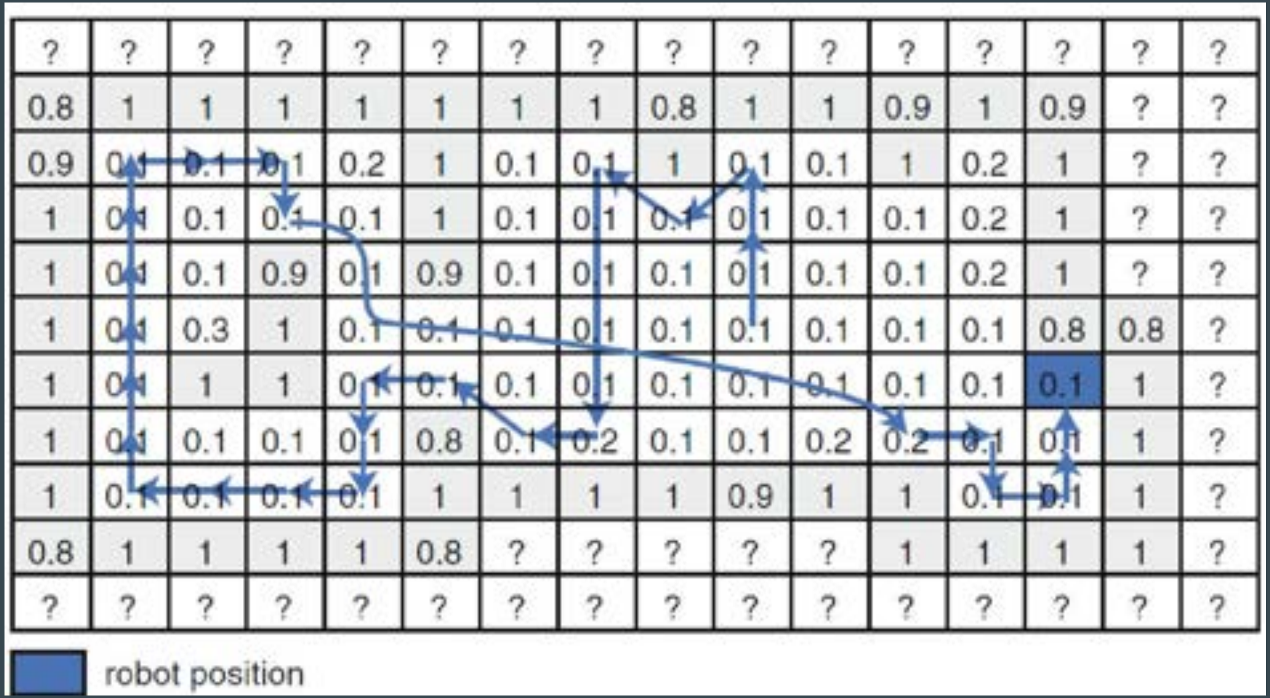
Frontier Algorithm – Grid Maps with Occupancy Information

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	1	?	?	0.8	1	1	0.9	1	0.9	?	?
?	?	?	?	?	1	0.1	0.1	1	0.1	0.1	1	0.2	1	?	?
?	?	?	?	?	1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.9	0.1	0.1	0.1	0.1	0.1	0.1	0.2	1	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	?	?	?
?	?	?	?	?	?	?	0.2	0.1	0.1	0.2	0.2	0.1	?	?	?
?	?	?	?	?	?	?	1	1	0.9	1	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

robot position

Frontier algorithm

Frontier Algorithm – Grid Maps with Occupancy Information



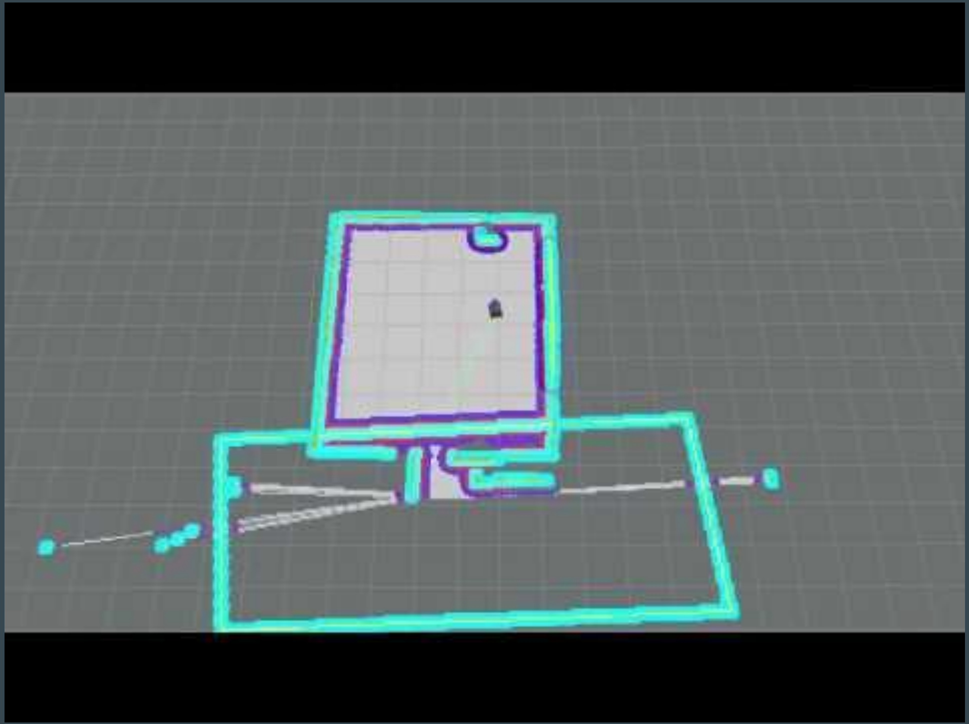
Priority of a frontier cell

Based on:

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

Frontier exploration

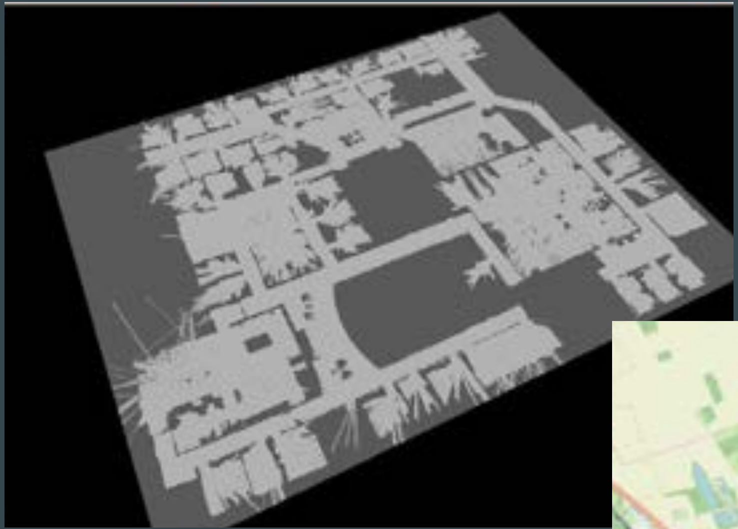
http://wiki.ros.org/frontier_exploration



Topological maps

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

Complex, large, structured environments?

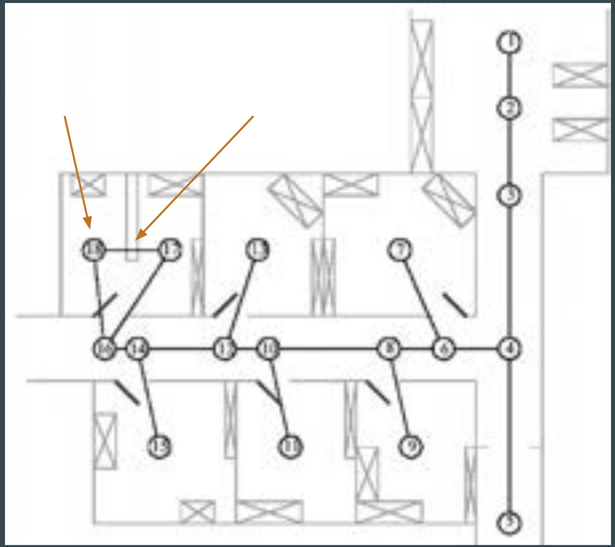


Complex, large, structured environments?

- 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

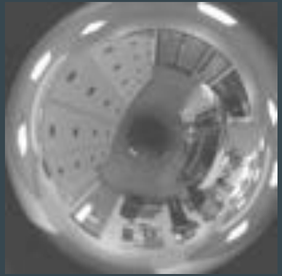


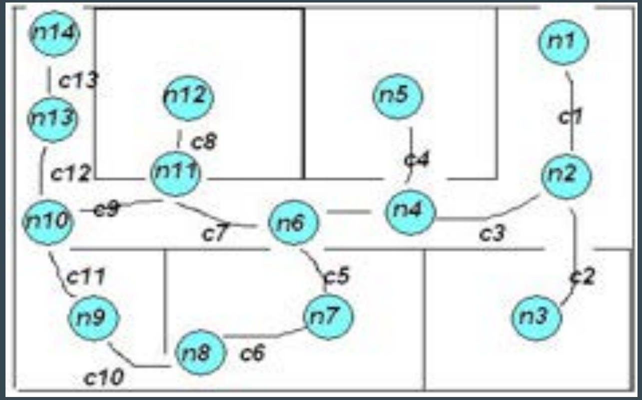
Constructing a topological map

- 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

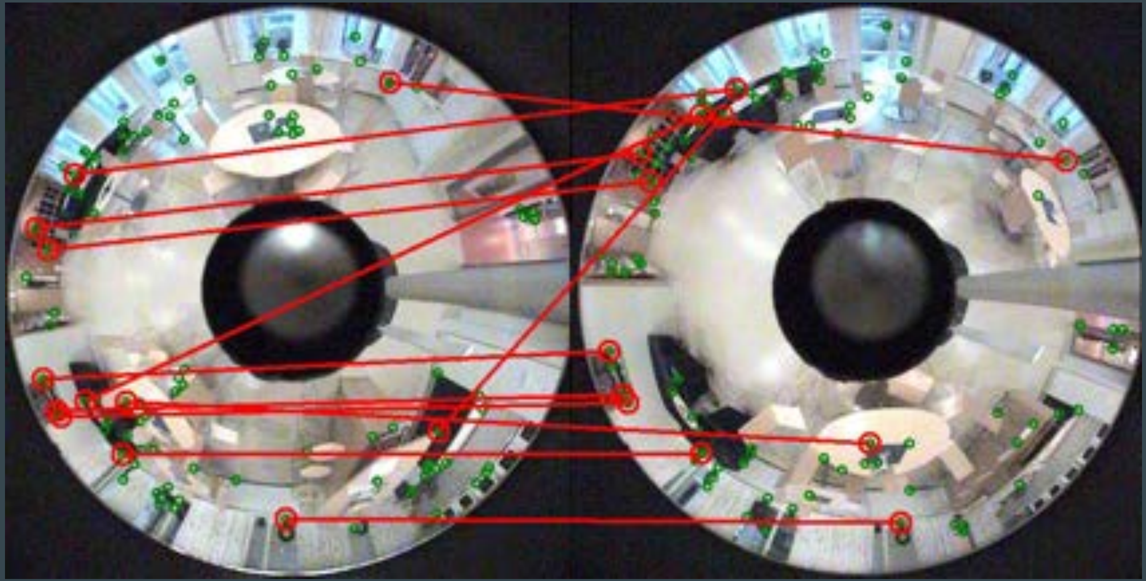
Example of topological mapping

- 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

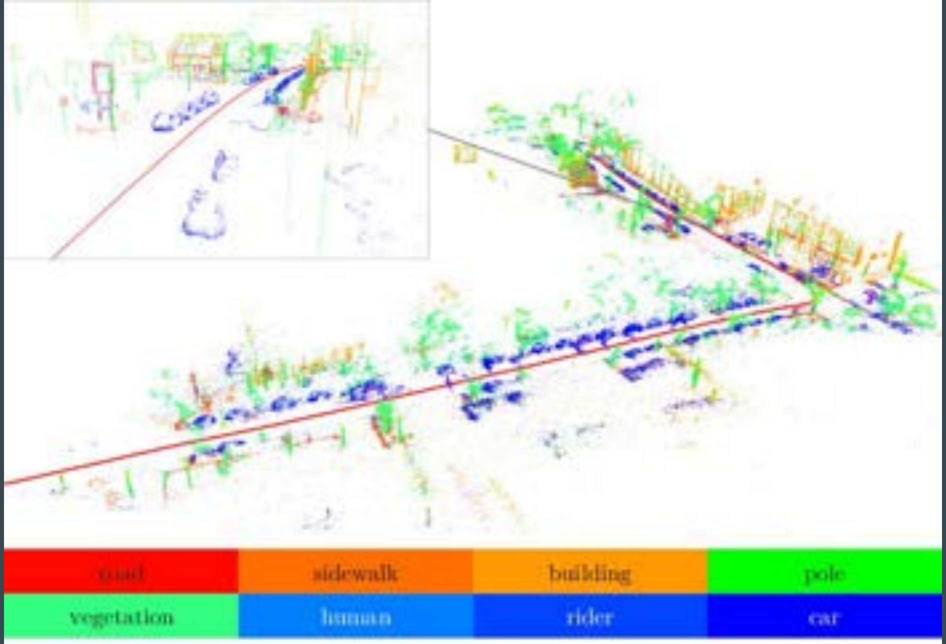
Metric maps

- 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

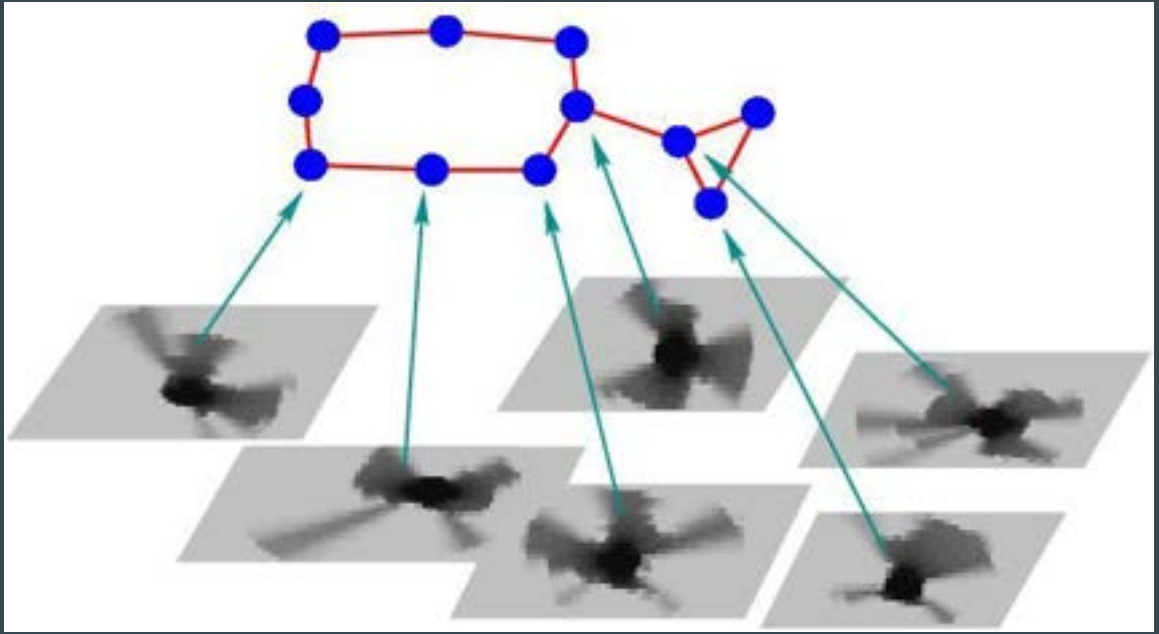
- Abstract, qualitative, “symbolic” representation
- May be more persistent/robust to environment dynamics
- Low computational cost - efficient path planning, scale better to large environments
- Require accurate place recognition - problem of perceptual aliasing (what if 2 or more places look alike?)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

Semantic maps



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

Hybrid maps

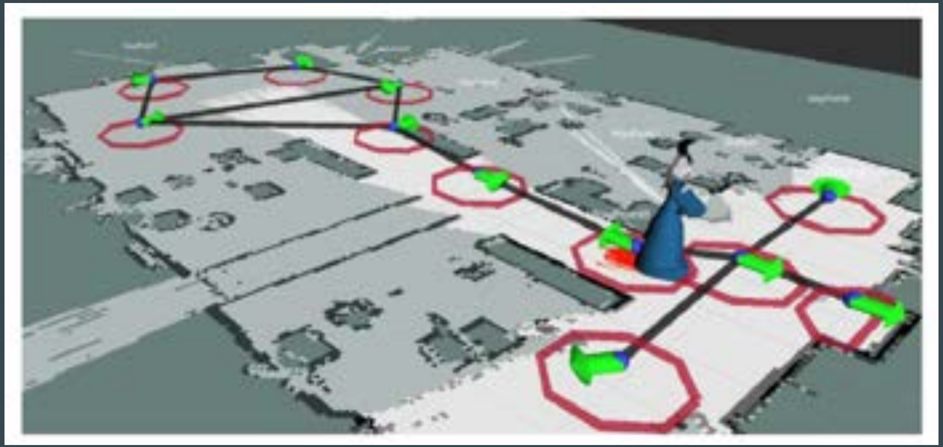


Combine complementary strengths of different representations (metric, topological, semantic maps)

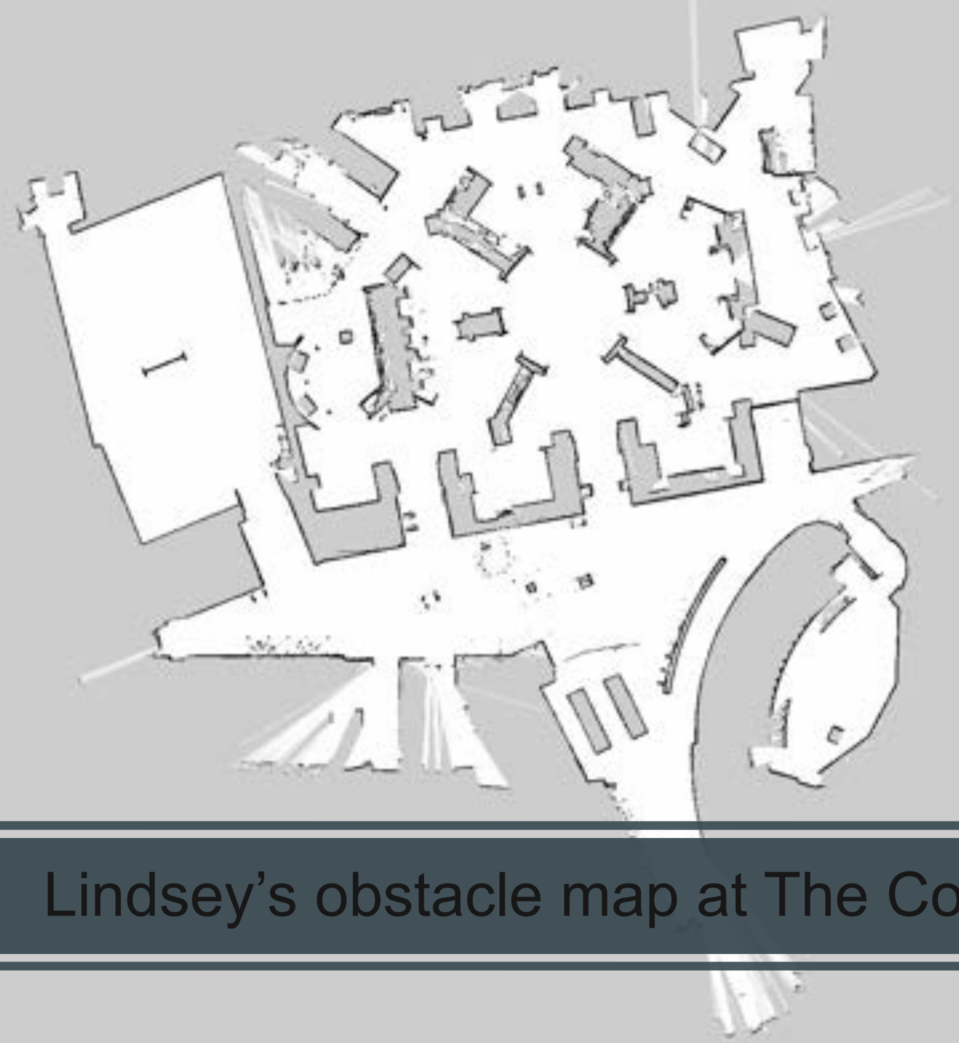
Hybrid maps

Example of a hybrid metric-topological map

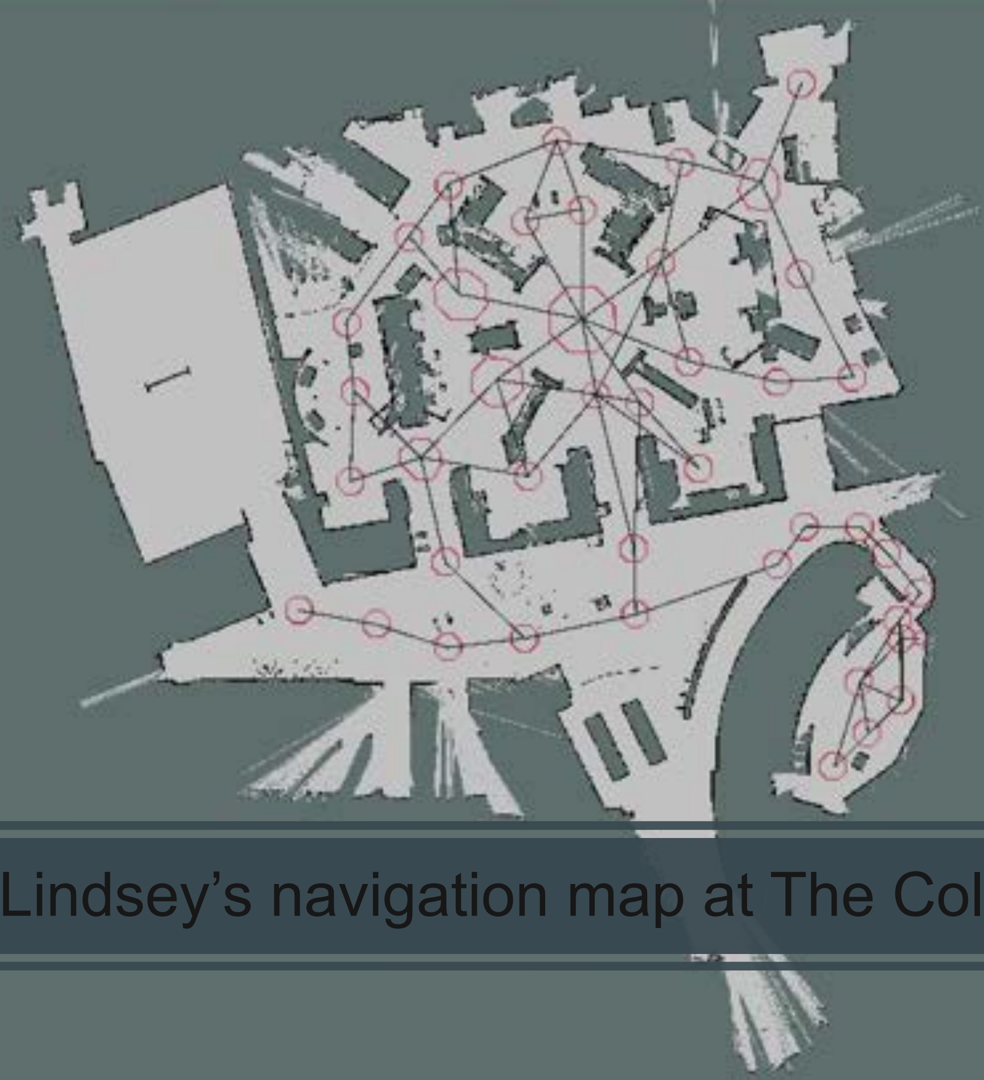
- **Topological level:** connected set of places
- **Metric level:** each place is associated with a local metric map



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

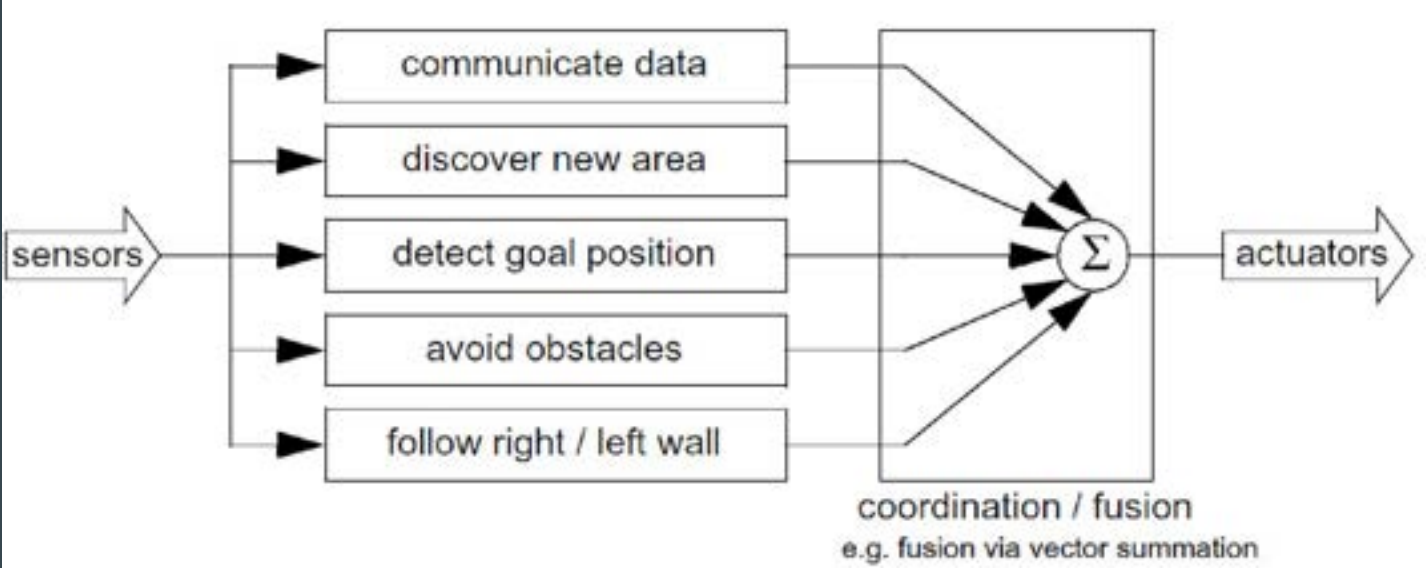
Odometry

Map-based

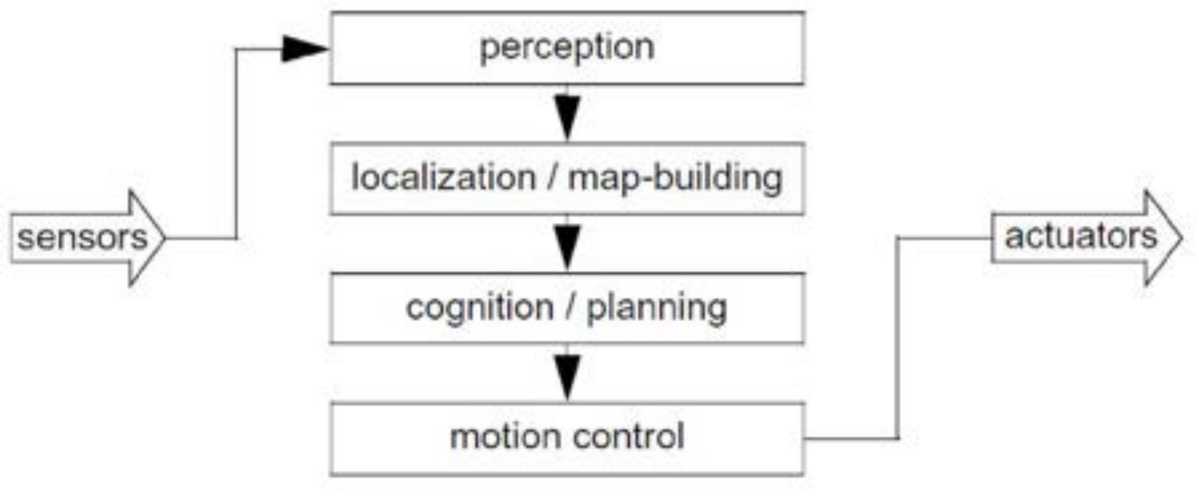
Odometry / dead reckoning

- **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)

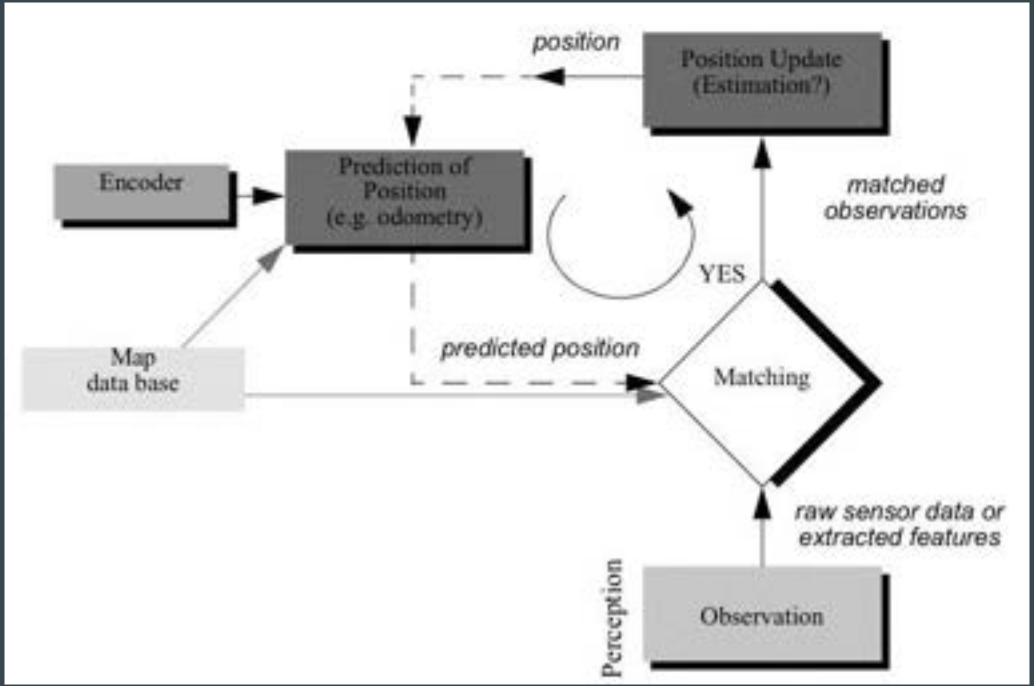
Particle Filters can address all of these cases

- a.k.a. Monte Carlo localisation

Metric localisation

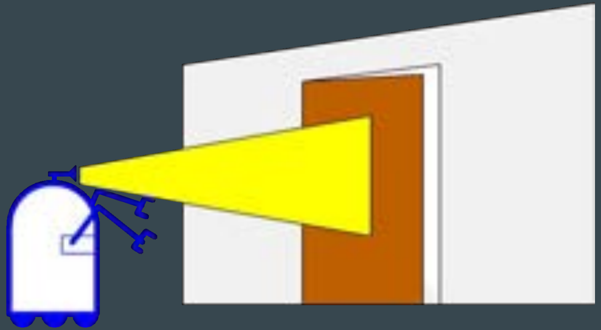
- Monte Carlo Localization
- Kalman Filter Localization
- Markov Localization

General process of map-based self-localisation



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



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

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

What is a particle?

An individual state estimate, defined by:

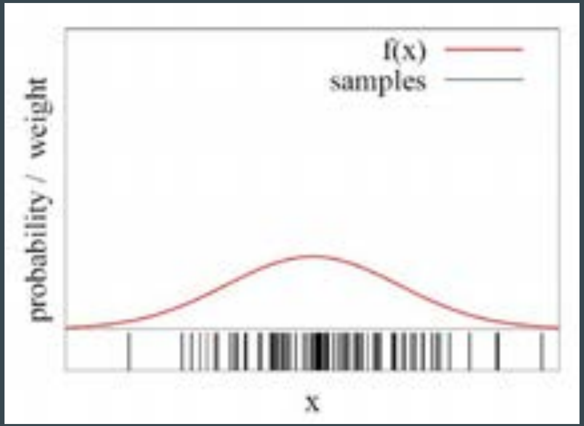
- State values that determine robot's pose (position and orientation)
  - e.g. [x, y, θ] for 2D self-localisation "in the plane"
- A weight that indicates its likelihood

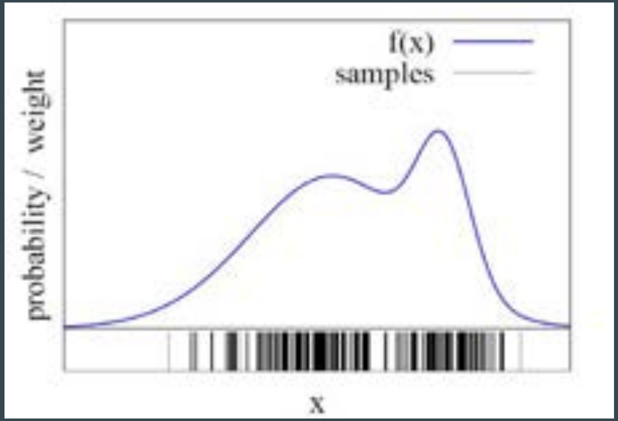
Particle filters use many particles to represent the belief state

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

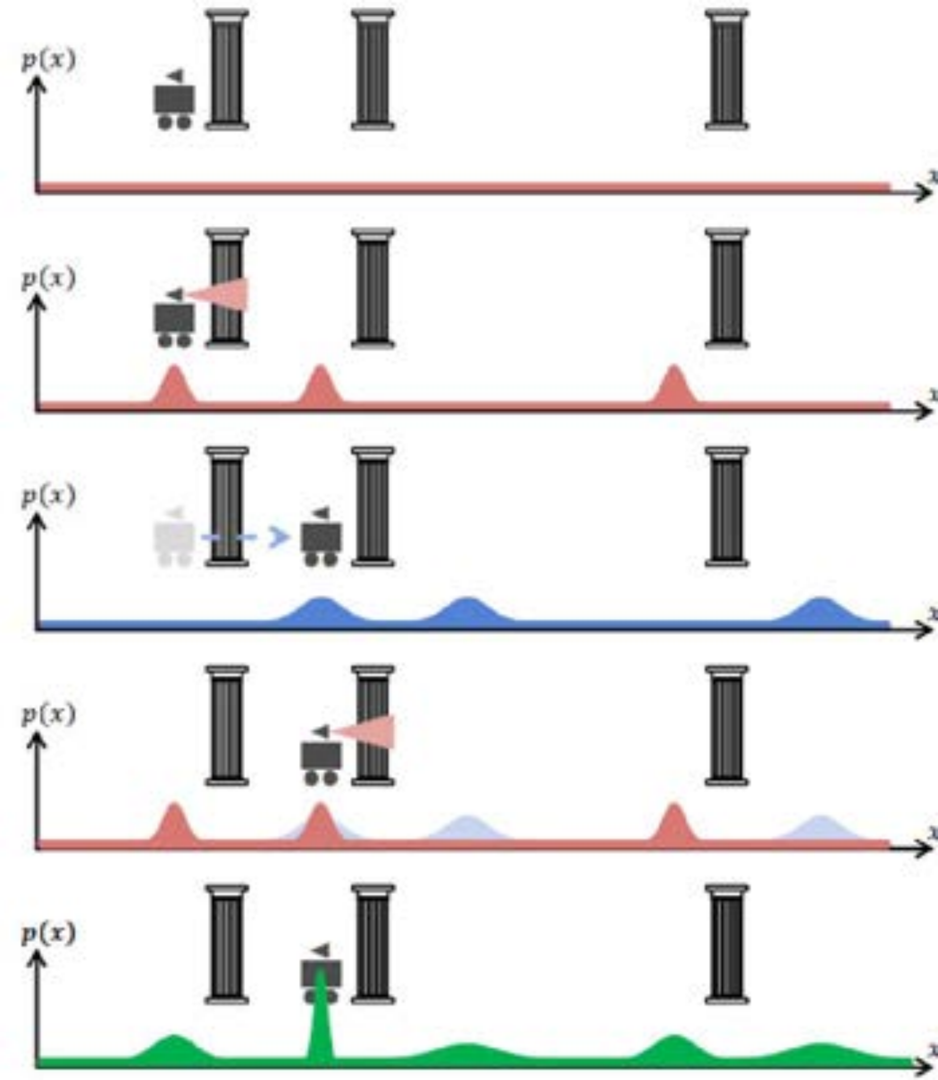
- Robot moves one metre forward
- Motion estimated by wheel encoder – accumulation of uncertainty

Perception update

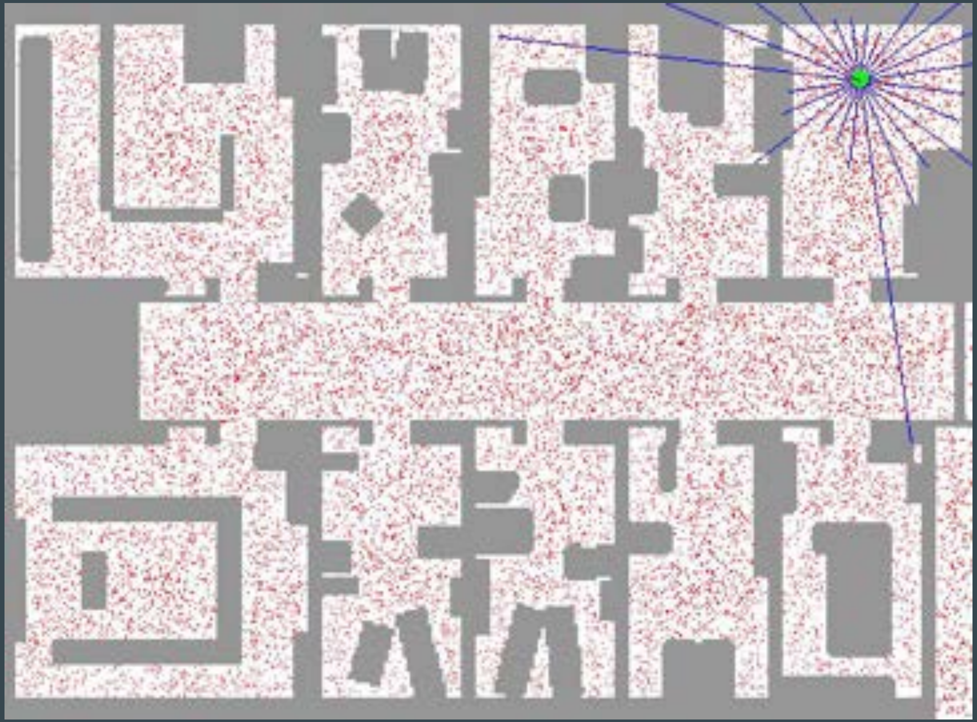
- Robot queries its sensors, and finds itself next to a pillar

Belief update

- Information fusion



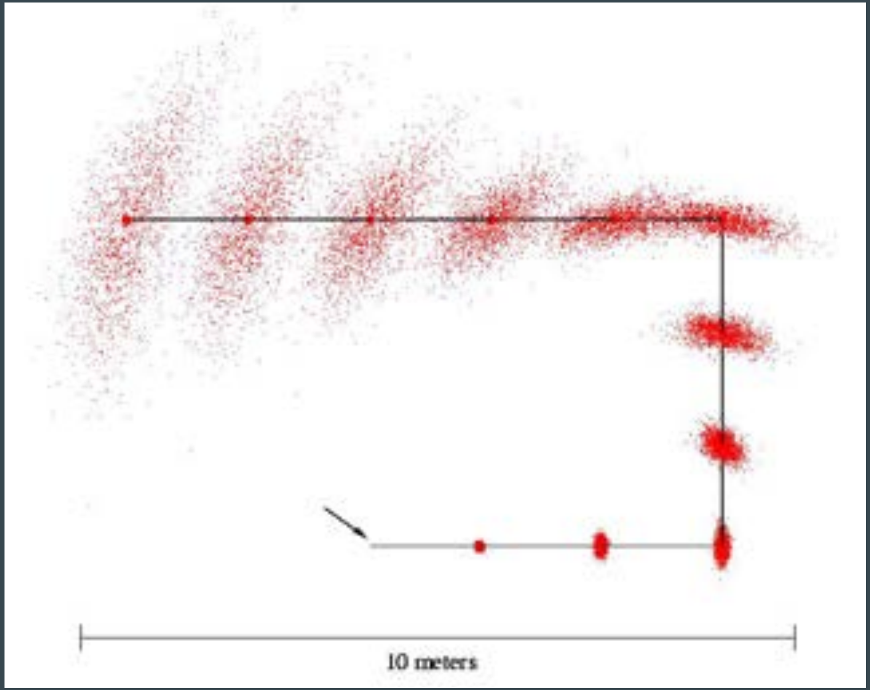
Randomised sampling
2D Monte Carlo localisation



Prediction step

- For each particle

- Sample and add random noisy values from the motion model



Motion model

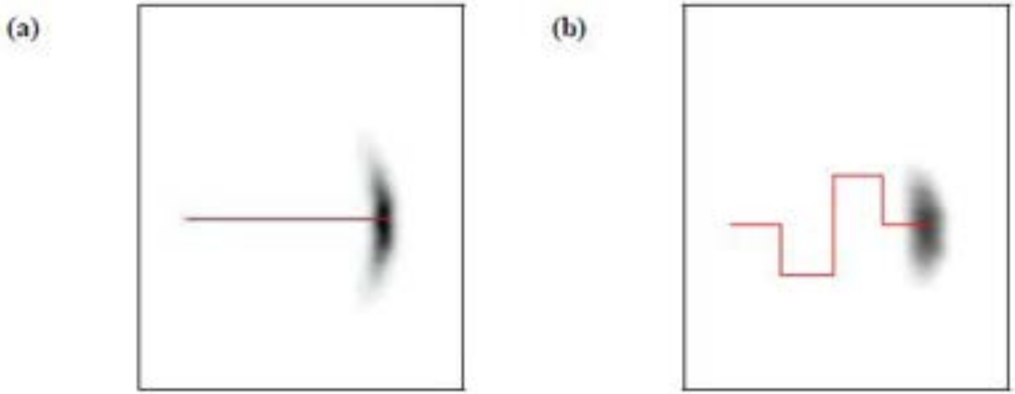


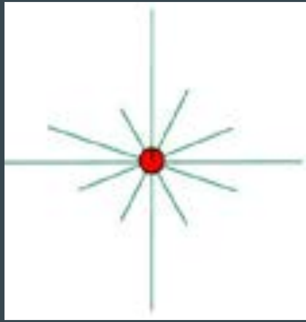
Figure 5.2 The motion model: Posterior distributions of the robot's pose upon executing the motion command illustrated by the solid line. The darker a location, the more likely it is. This plot has been projected into 2D. The original density is three-dimensional, taking the robot's heading direction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

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

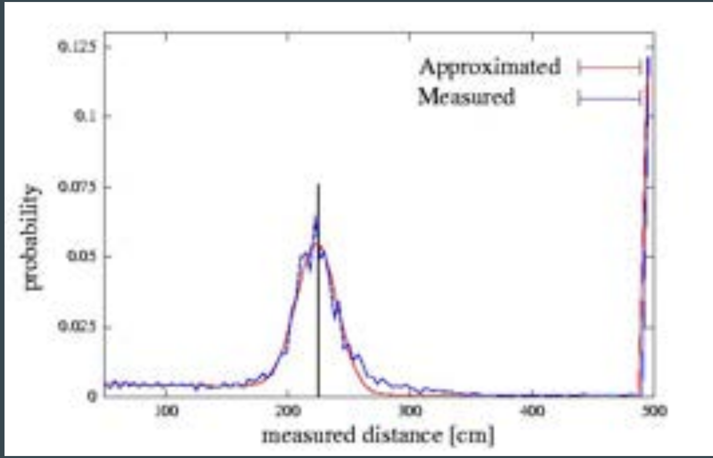
Update step

- 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

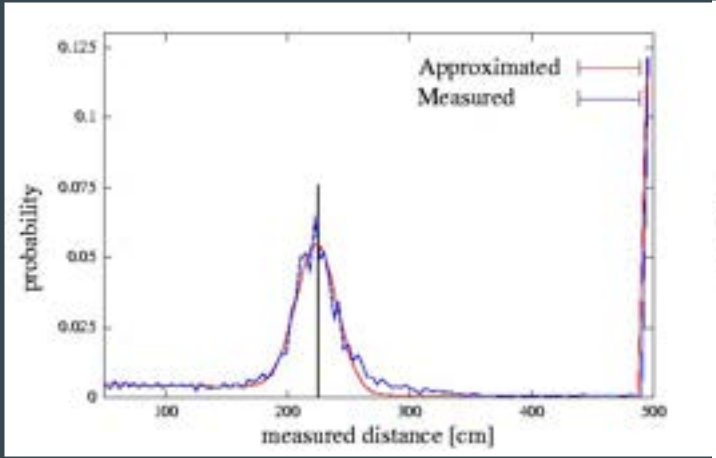


Laser sensor

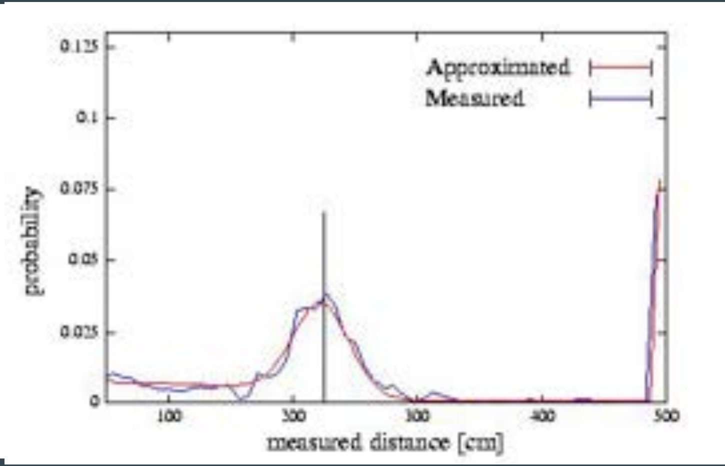


Sensor model

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

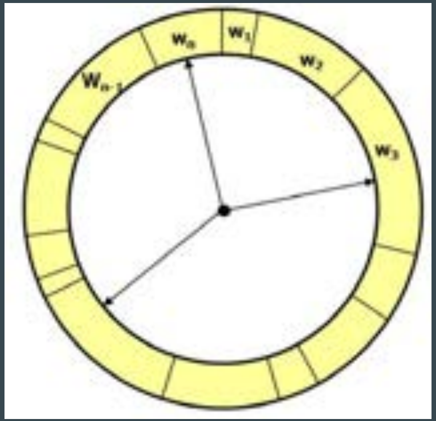


Laser sensor



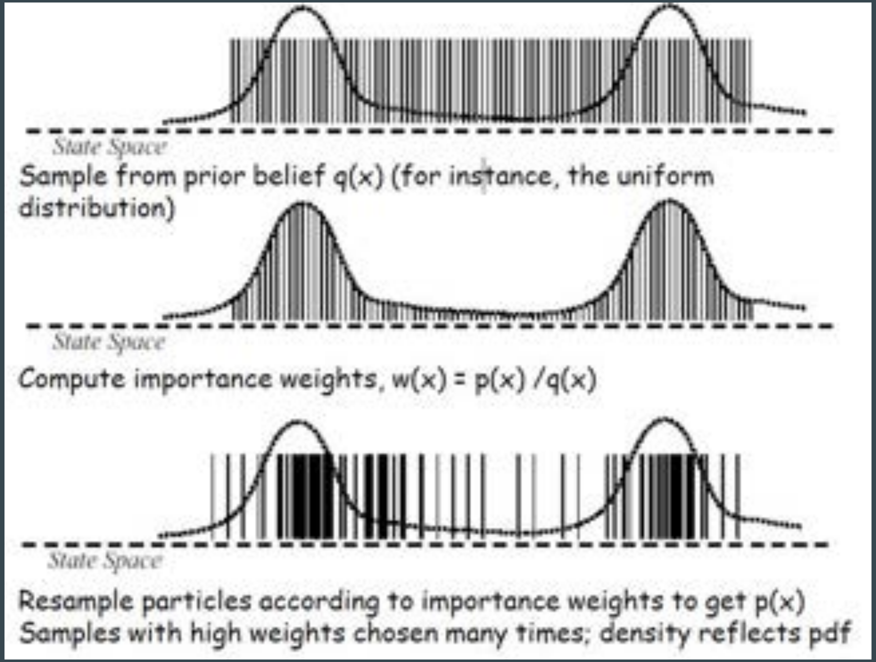
Sonar sensor

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


State Space
Sample from prior belief q(x) (for instance, the uniform distribution)
State Space
Compute importance weights, w(x) = p(x) /q(x)
State Space
Resample particles according to importance weights to get p(x)
Samples with high weights chosen many times; density reflects pdf

Particle filter algorithm

- We approximate $P(x_t)$ by a set of samples:
  - $P(x_t) = \{x_t^{(i)}, w_t^{(i)}\}_{i=1,\ldots,m}$
- 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

$x_{t+1} = \varnothing$

for $j = 1$ to m

    // `apply the transition model`

    generate a new sample $x_{t+1}^{(j)}$ from $x_t^{(j)}$, a_t and $\Pr(x_{t+1} \mid x_t, a_t)$

    // `apply the sensor model`

    compute the weight $w_{t+1}^{(j)} = \Pr(e_{t+1} \mid x_{t+1})$

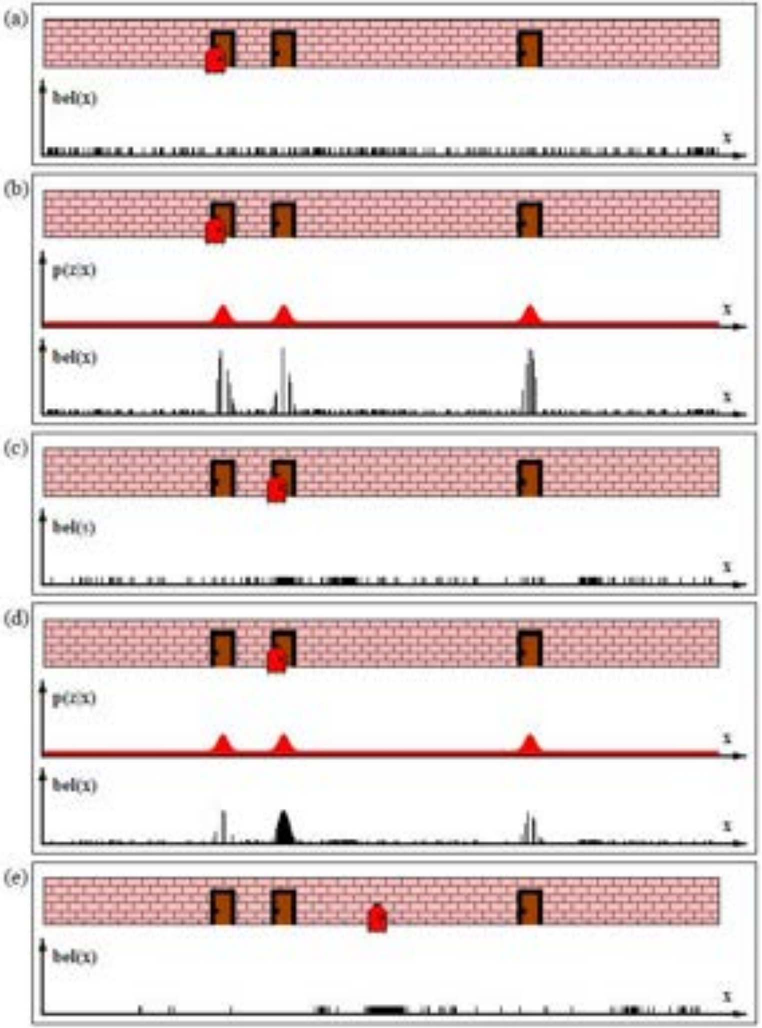
// `pick points randomly but biased by their weight`

for $j = 1$ to m

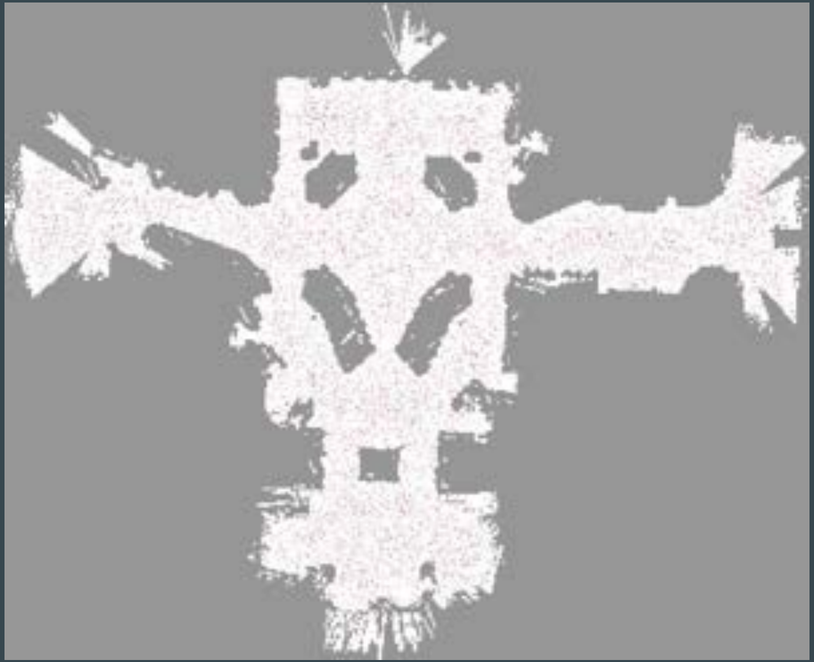
    pick a random $x_{t+1}^{(i)}$ from x_{t+1} according to $w_{t+1}^{(1)}, \ldots, w_{t+1}^{(m)}$

normalize w_{t+1} in x_{t+1}

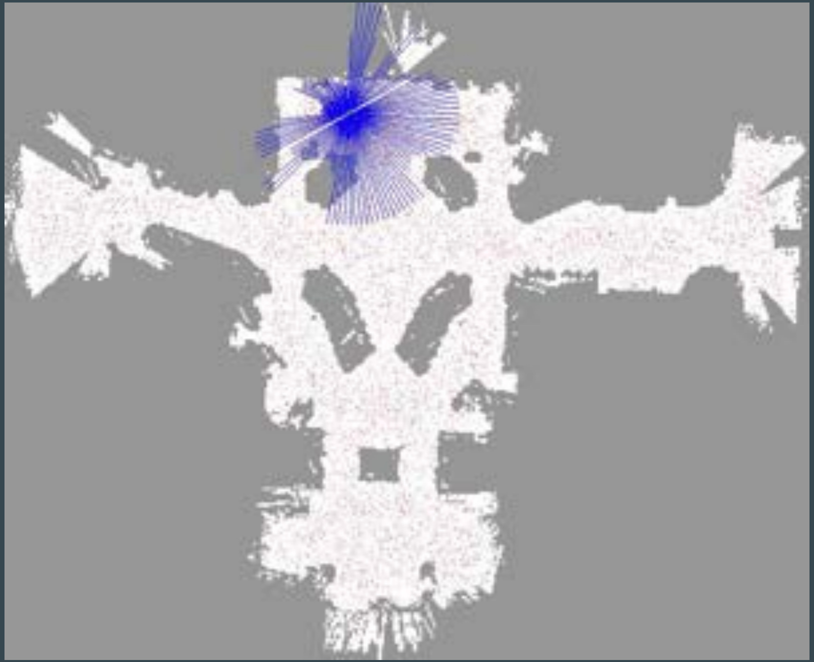
return x_{t+1}


(a)
bel(x)
x
(b)
p(z|x)
x
bel(x)
x
(c)
bel(x)
x
(d)
p(z|x)
x
bel(x)
x
(e)
bel(x)
x

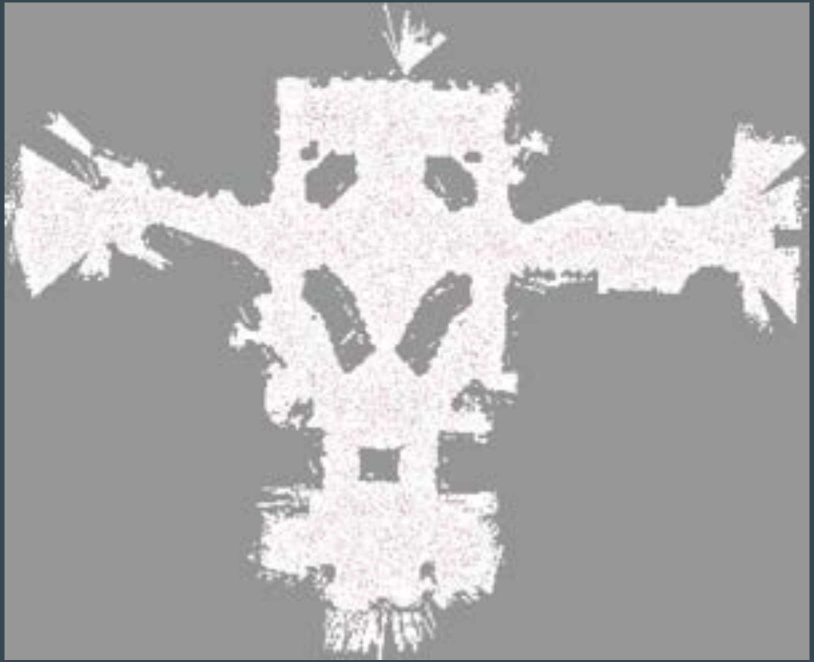
Initialisation



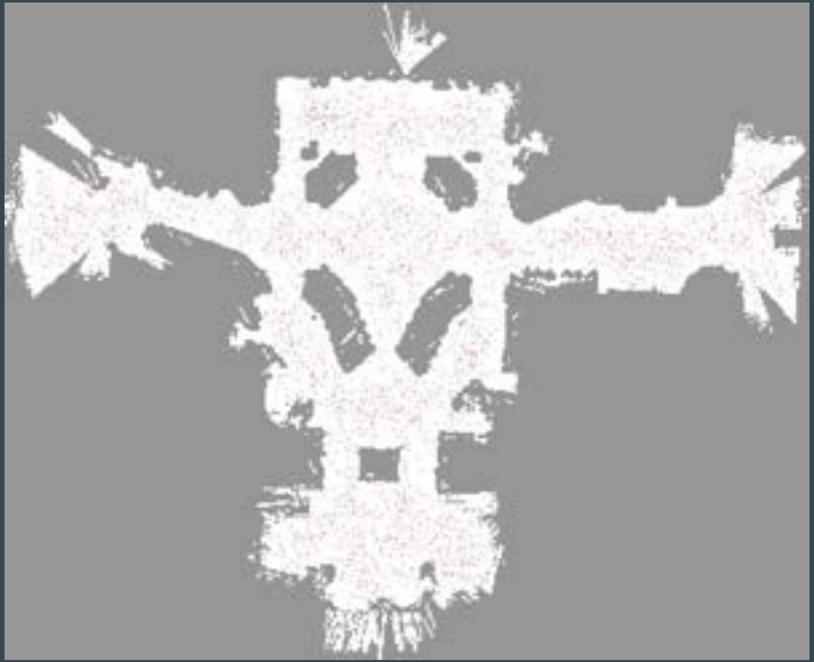
Measurement



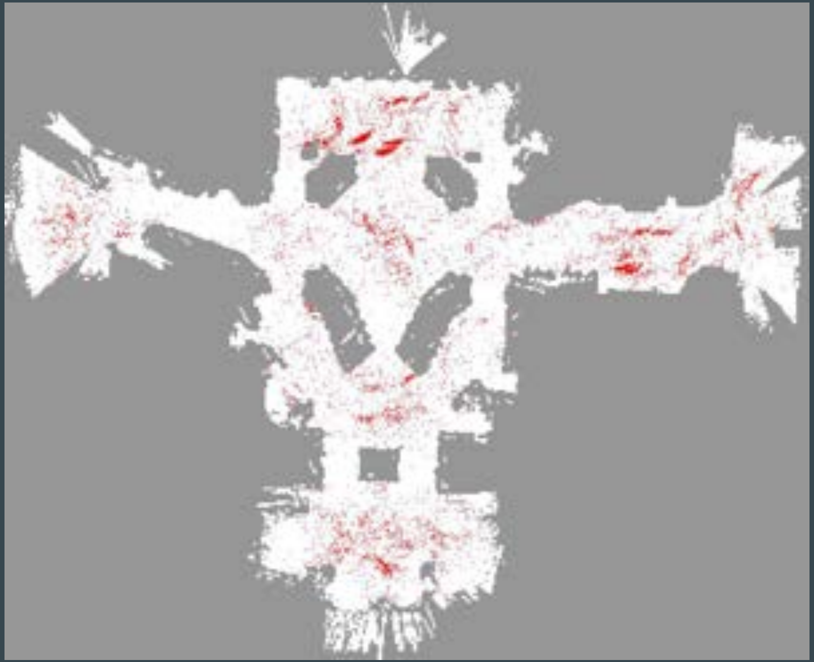
Weight update



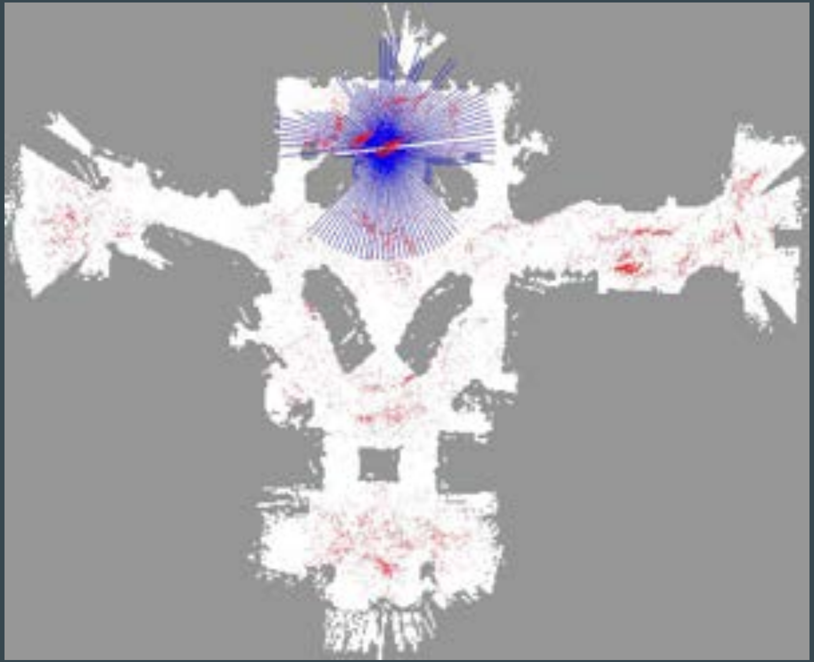
Resampling



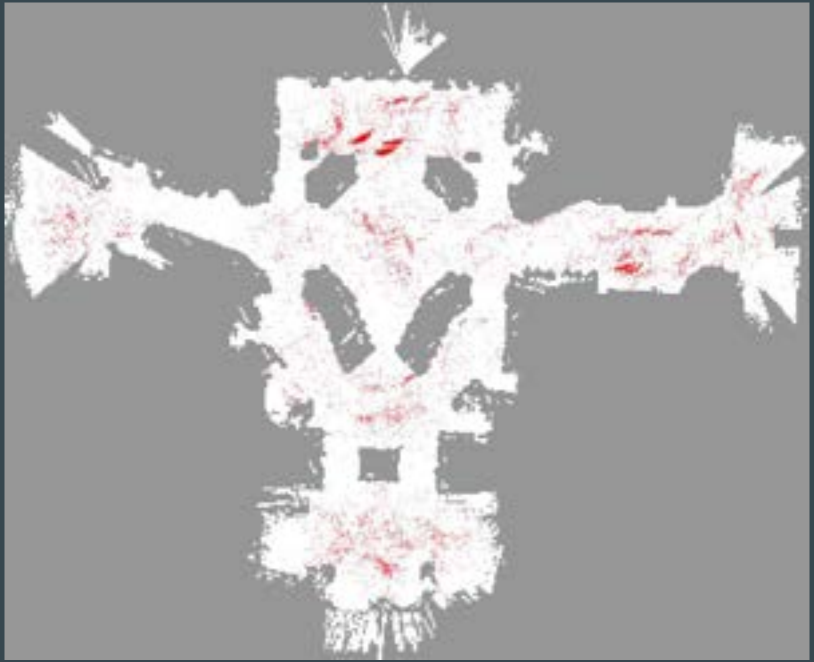
Motion update



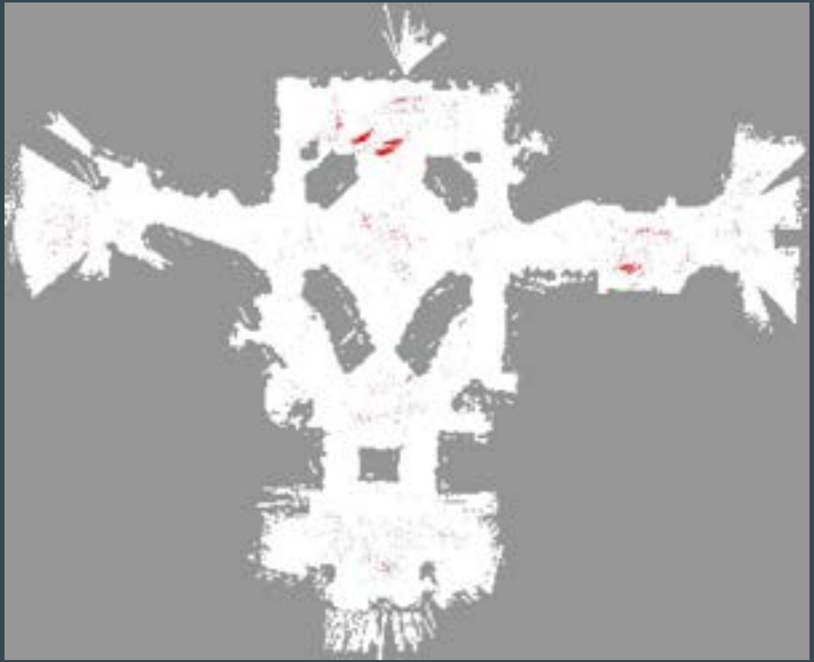
Measurement



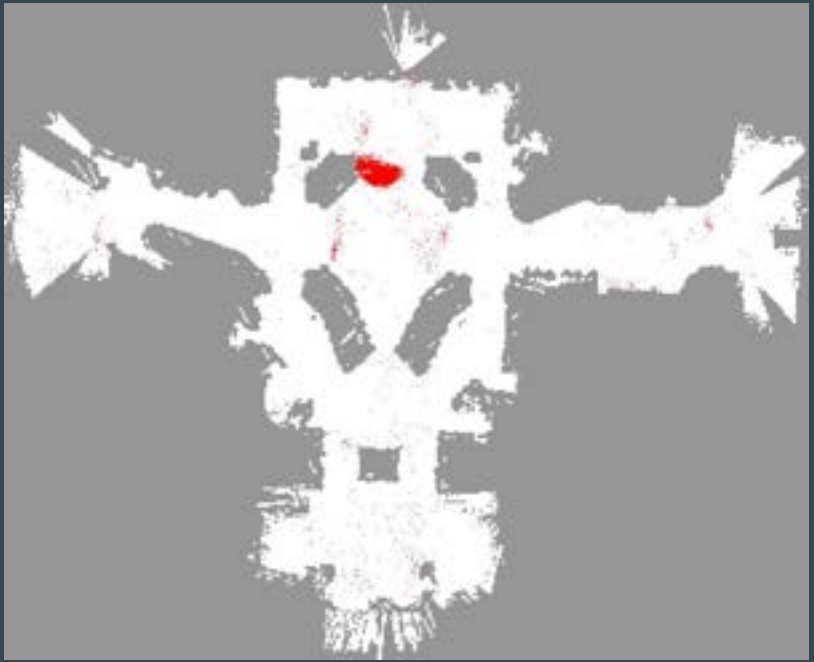
Weight update



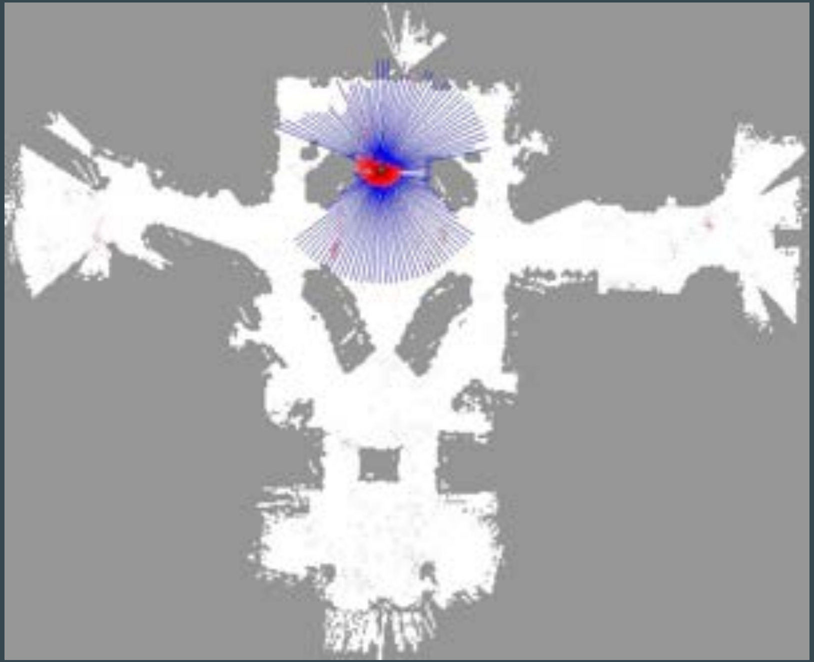
Resampling



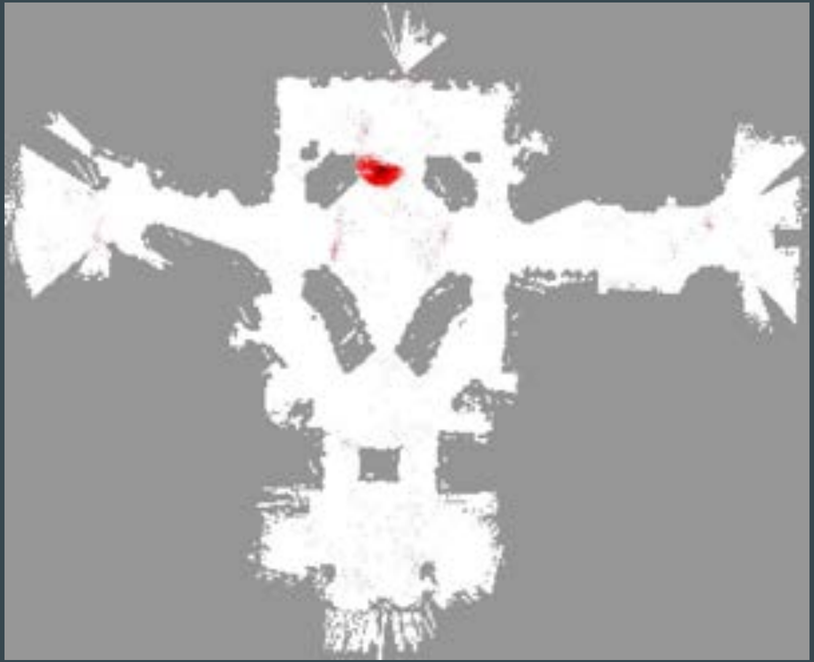
Motion update



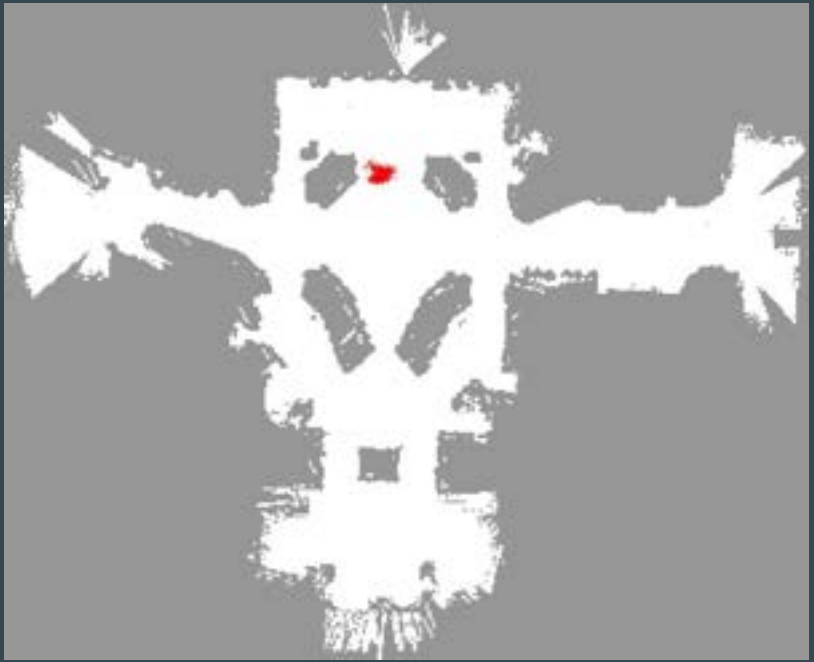
Measurement



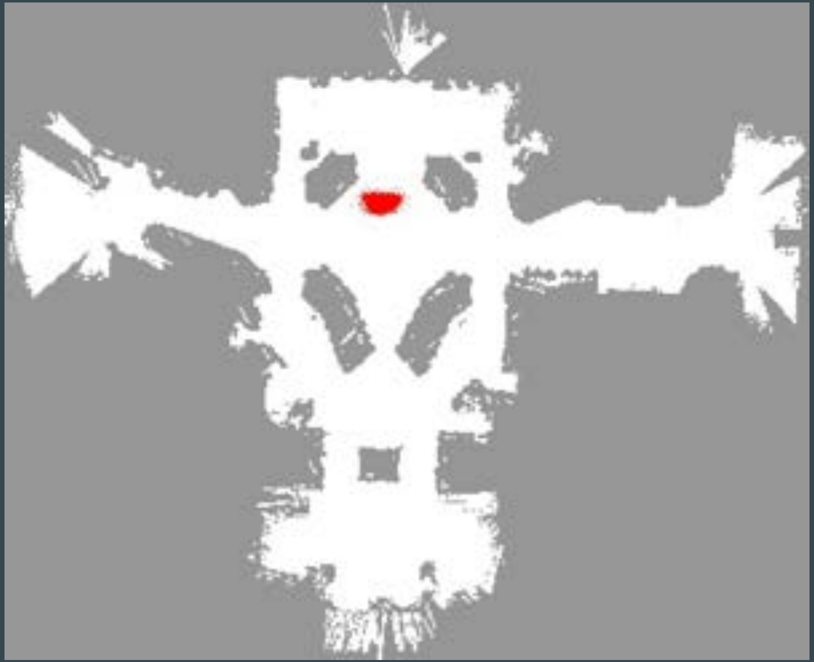
Weight update



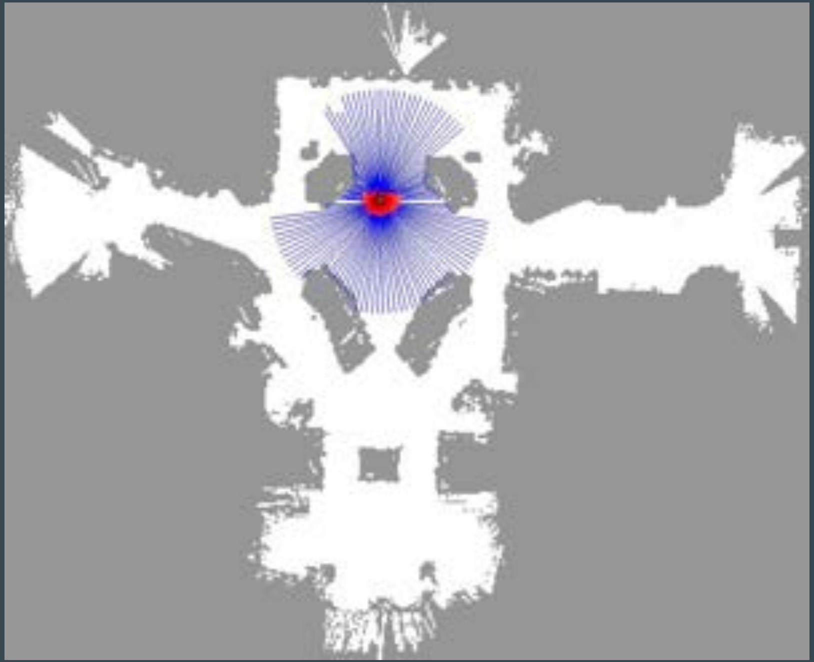
Resampling



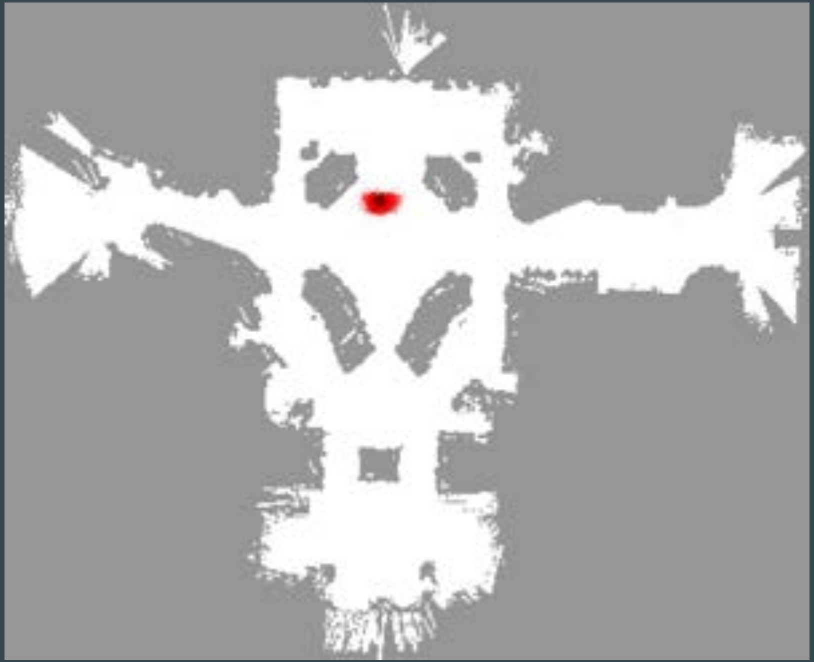
Motion update



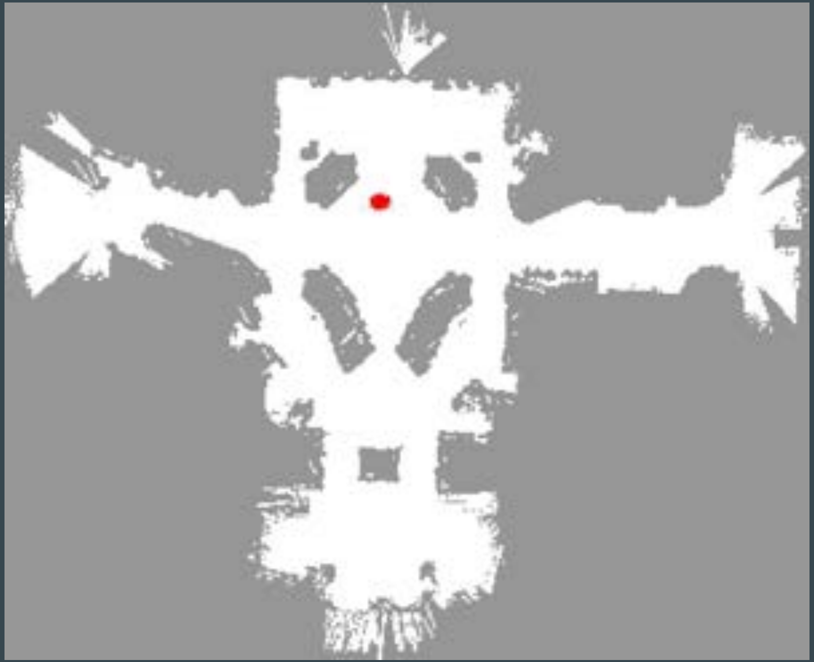
Measurement



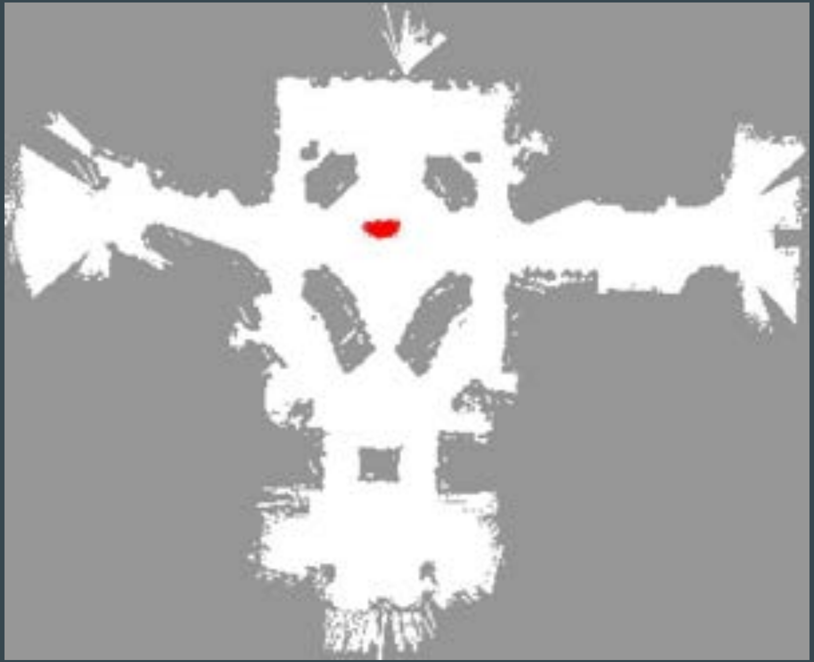
Weight update



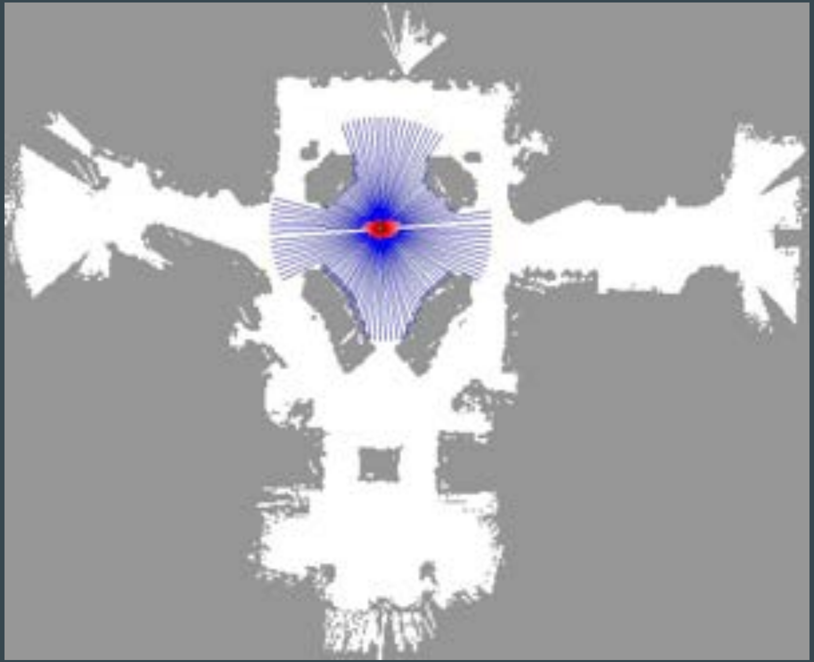
Resampling



Motion update



Measurement



Monte Carlo localisation summary

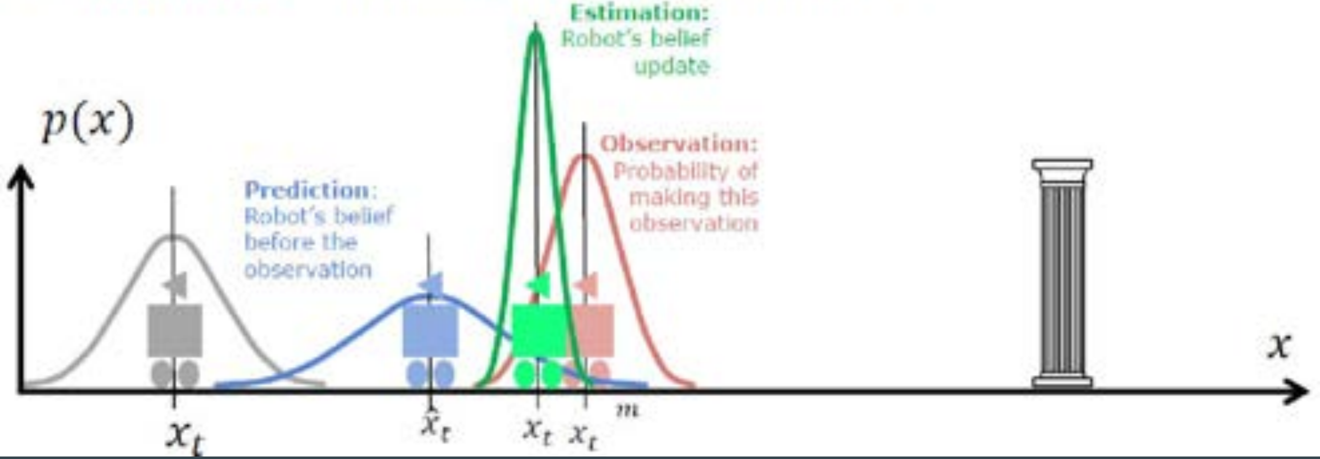
- Particle Filters (PFs) can represent arbitrary probability density functions (distributions) using samples
- PFs use sample importance resampling, with 4 main steps:
    - Initialisation
    - Predict
    - Update
    - Resample
- PFs can solve the “kidnapped robot problem” and handle perceptually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

for each location l **do** /* *initialize* the belief */

$$Bel(L_0 = l) \longleftarrow P(L_0 = l) \qquad (17)$$

end for

forever do

  if new sensory input s_T is received **do**

    $\alpha_T \longleftarrow 0$

    for each location l **do** /* apply the *perception model* */

$$\widehat{Bel}(L_T = l) \longleftarrow P(s_T \mid l) \cdot Bel(L_{T-1} = l) \qquad (18)$$

$$\alpha_T \longleftarrow \alpha_T + \widehat{Bel}(L_T = l) \qquad (19)$$

    end for

    for each location l **do** /* *normalize* the belief */

$$Bel(L_T = l) \longleftarrow {\alpha_T}^{-1} \cdot \widehat{Bel}(L_T = l) \qquad (20)$$

    end for

  end if

  if an odometry reading a_T is received **do**

    for each location l **do** /* apply the *motion model* */

$$Bel(L_T = l) \longleftarrow \int P(l \mid l', a_T) \cdot Bel(L_{T-1} = l')\, dl' \qquad (21)$$

    end for

  end if

end forever

Tab. 1. The Markov localization algorithm

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

ROS Navigation stack

https://navigation.ros.org/index.html - Ros2 Navigation package

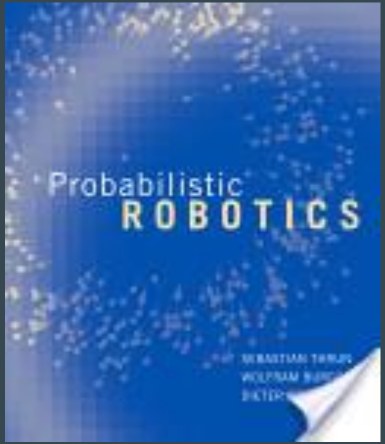
Map server - map_server provides the map_server ROS Node, which offers map data as a ROS Service. It also provides the map_saver command-line utility, which allows dynamically generated maps to be saved to file.

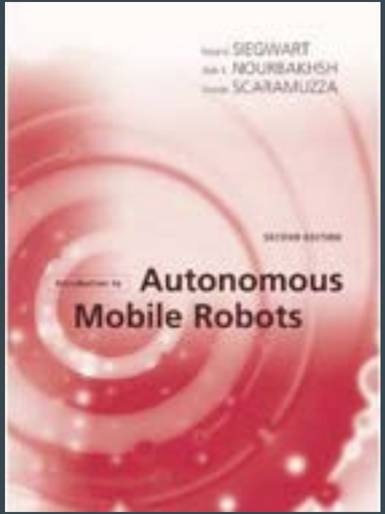
Slam - provides laser-based SLAM (Simultaneous Localisation and Mapping), as a ROS node called slam_gmapping. Using slam_gmapping, you can create a 2D occupancy grid map from laser and pose data collected by a mobile robot.

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 planning

Dr Athanasios Polydoros

Today's lecture

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

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

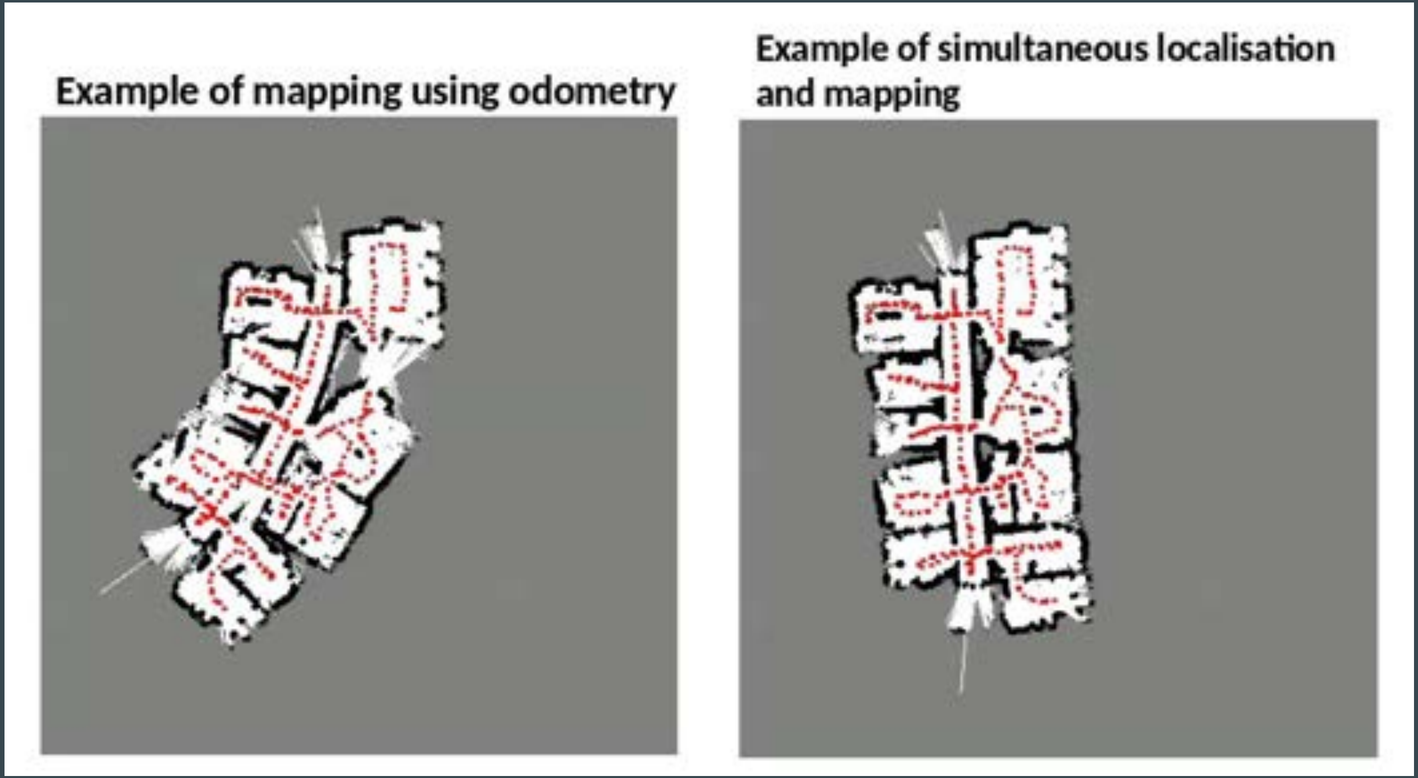
- How can a robot navigate a previously unknown environment, while constantly building and updating a map of its workspace using on-board sensors and computation?

- **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?



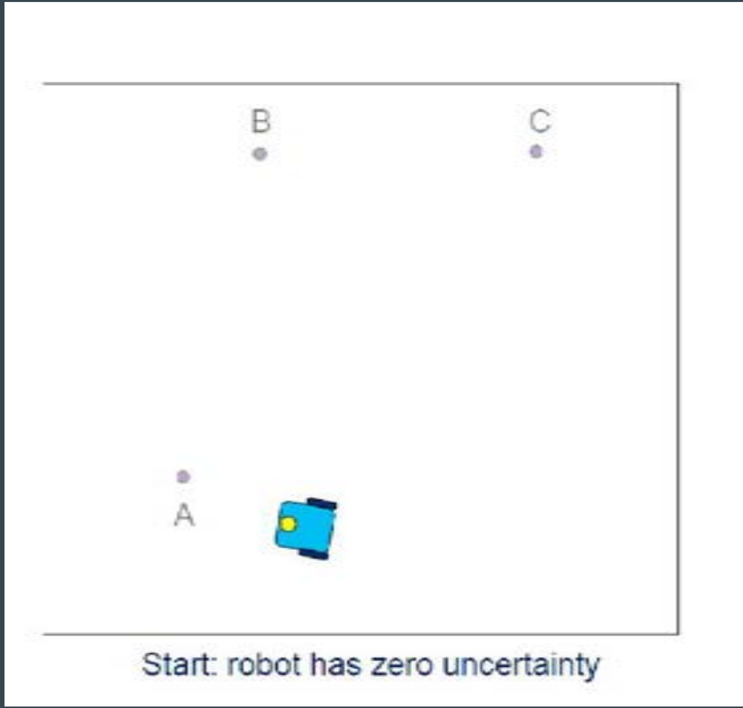
SLAM with a Gaussian filter

Use internal representations for:

- The positions of landmarks
  The camera parameters

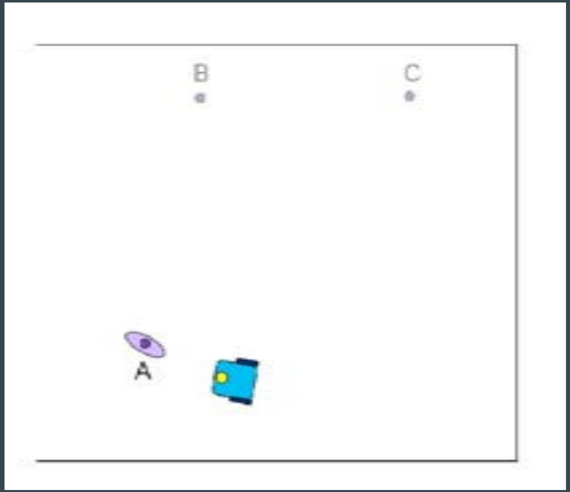
Assumption:

- Robot's uncertainty at starting point is zero



SLAM with Gaussian filter

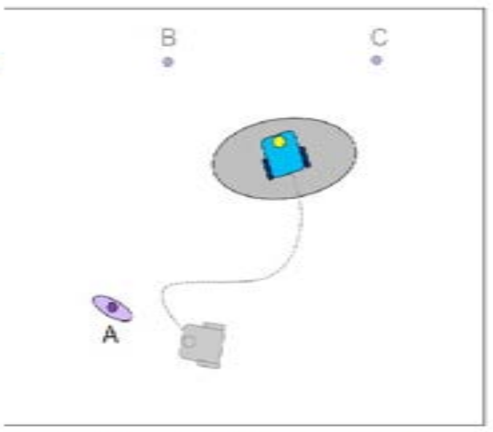
- Robot observes a feature (A)
- Mapped with uncertainty related to the measurement model
  - e.g. camera model describing how world points map into image pixels



SLAM with Gaussian filter

- As the robot moves, its pose uncertainty increases
  - Obeying the robot’s motion model

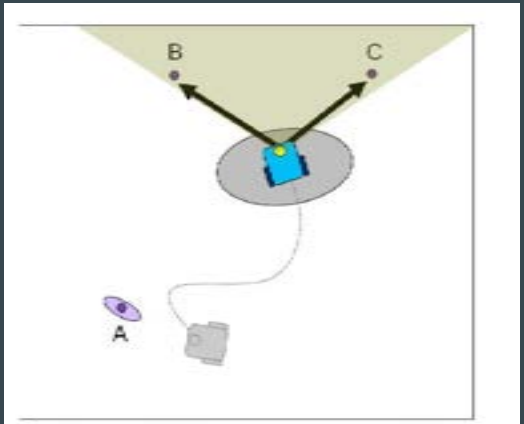
- e.g. control commands: turn right, drive on for 1m
  - Uncertainty is added due to wheel slippage and other imprecisions



Robot moves forwards: uncertainty grows

SLAM with Gaussian filter

- Robot observes two new features
  - B and C

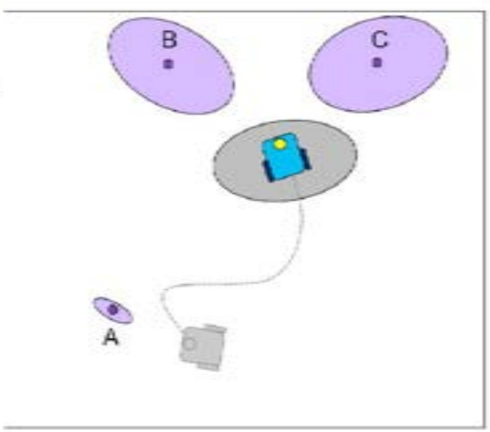


Robot makes first measurements of B & C

SLAM with Gaussian filter

- Position uncertainty of B and C results from combination of:
  - Measurement error
  - Robot pose uncertainty

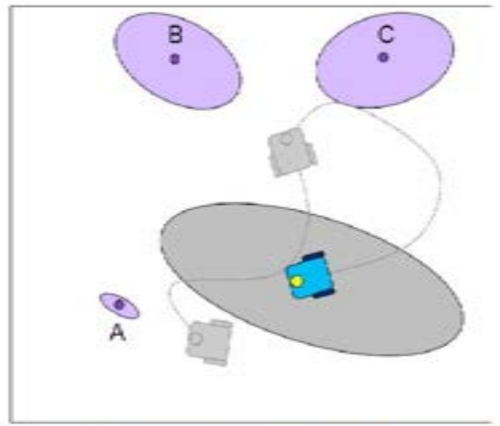
- Map becomes correlated with the robot pose estimate



Robot makes first measurements of B & C

SLAM with Gaussian filter

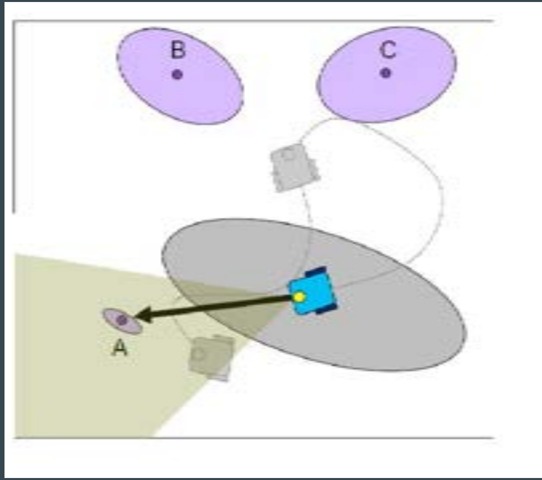
- Robot moves again
- Its uncertainty increases
  - Based on motion model



Robot moves again: uncertainty grows more

SLAM with Gaussian filter

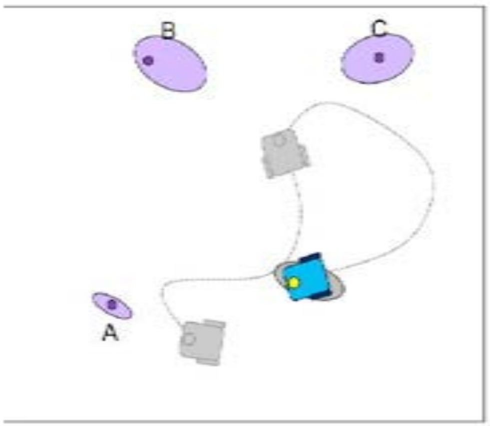
- Robot re-observes an old feature (A)
- **Loop closure** detection



SLAM with Gaussian filter

- Robot updates its position
  - Resulting pose estimate becomes correlated with the feature location estimates

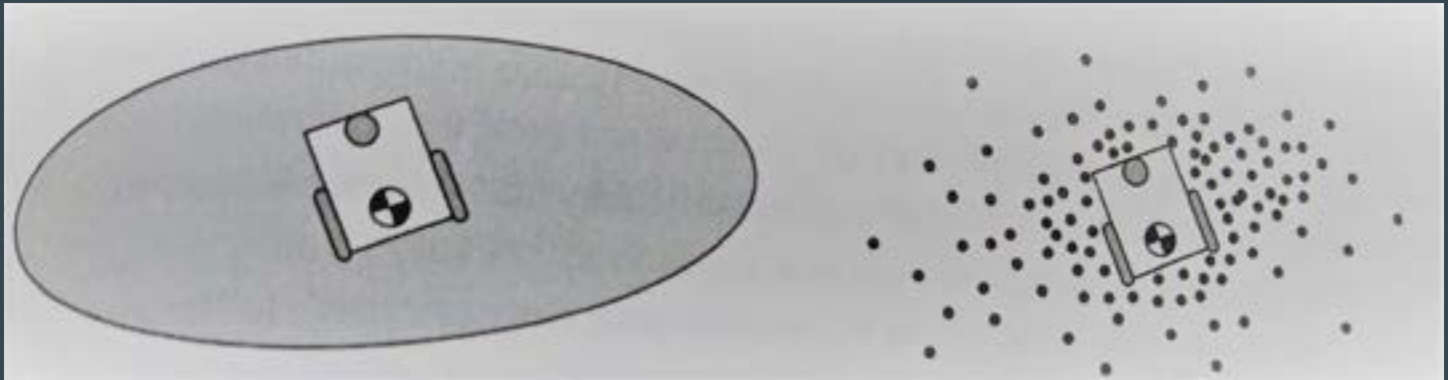
- Robot's uncertainty shrinks
  - So does uncertainty in the rest of the map



Robot re-measures A: "**loop closure**"
uncertainty shrinks

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

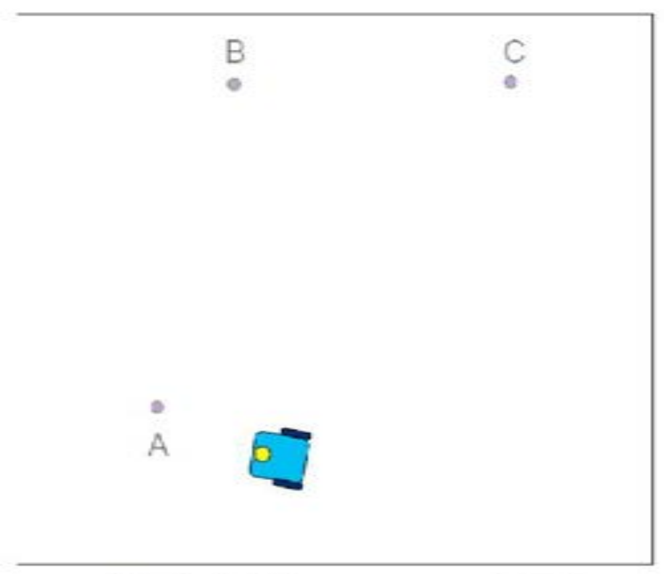
Use internal representations for:

- 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

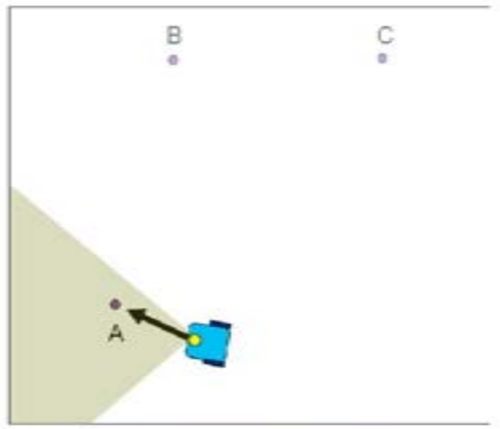


Start: robot has zero uncertainty

SLAM with particle filter

On every frame:

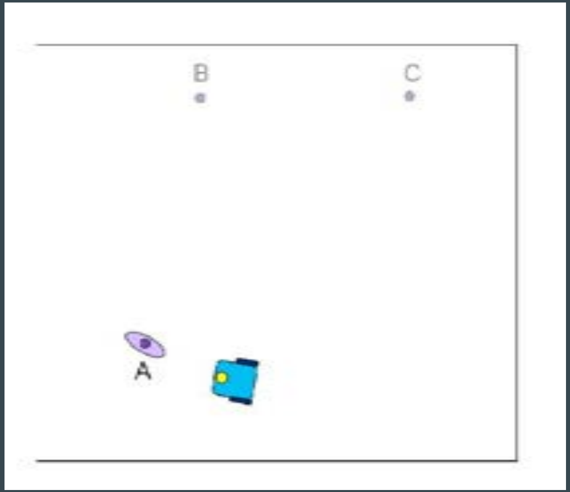
- **Predict** how the robot has moved
- **Measure** the world through sensors
- **Update** the internal representation



First measurement of feature A

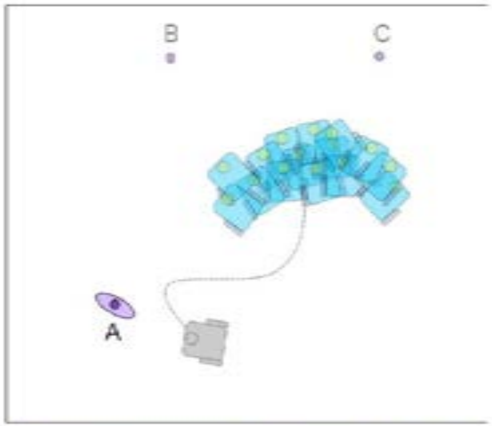
SLAM with particle filter

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



SLAM with particle filter

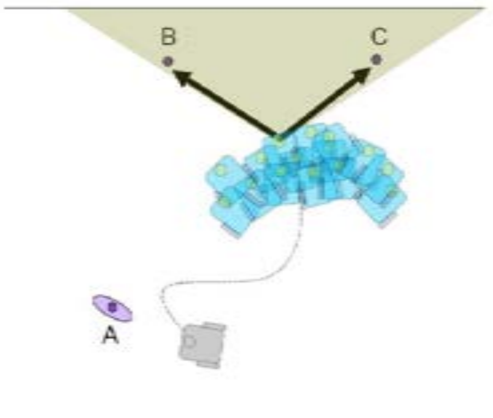
- As the robot moves, pose uncertainty increases
- Apply motion model to each particle



Robot moves forwards: uncertainty grows

SLAM with particle filter

- Robot observes two new features
  - B and C

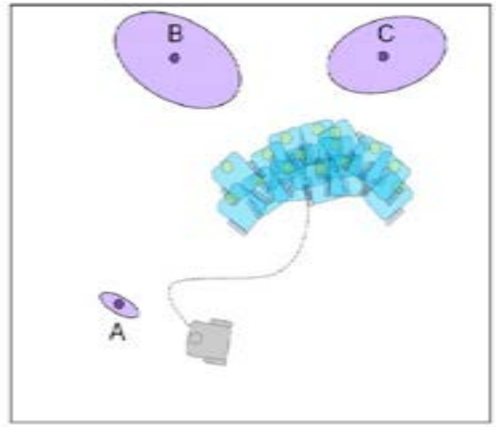


Robot makes first measurements of B & C

SLAM with particle filter

Position uncertainty encoded for each particle individually:

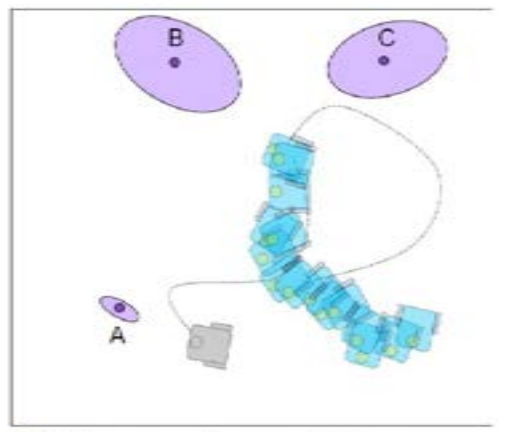
- Compare particle's predicted measurements with actual measurements
- Re-weight particles – those with good predictions get higher weight
- Renormalise particle weights
- Resample



Robot makes first measurements of B & C

SLAM with particle filter

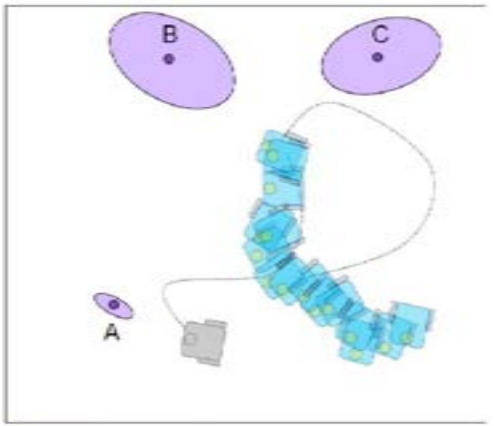
- Robot moves again and its uncertainty increases
- Apply motion model to each particle



Robot moves again: uncertainty grows more

SLAM with particle filter

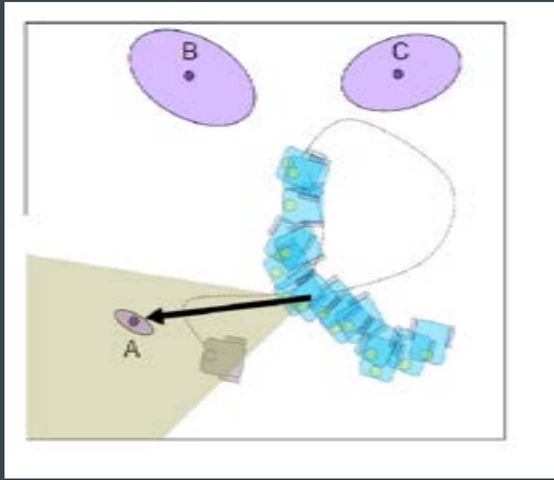
- Robot moves again and its uncertainty increases
- Apply motion model to each particle



Robot moves again: uncertainty grows more

SLAM with particle filter

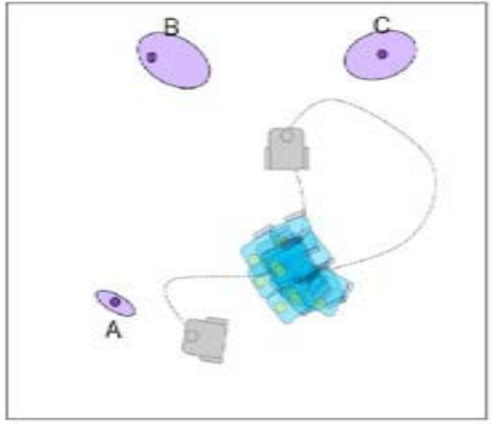
- Robot re-observes an old feature (A)
- **Loop closure** detection



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



Robot re-measures A: "loop closure" uncertainty shrinks

Motion planning

- Planning Spaces
- Algorithms

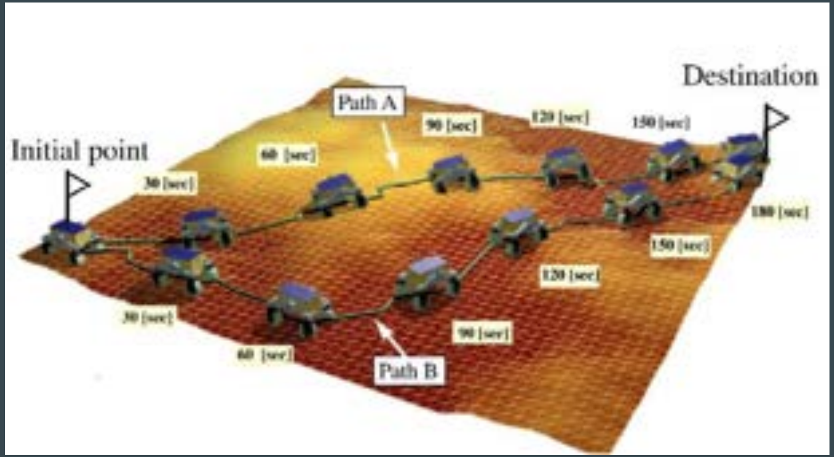
Robot motion planning

- 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?

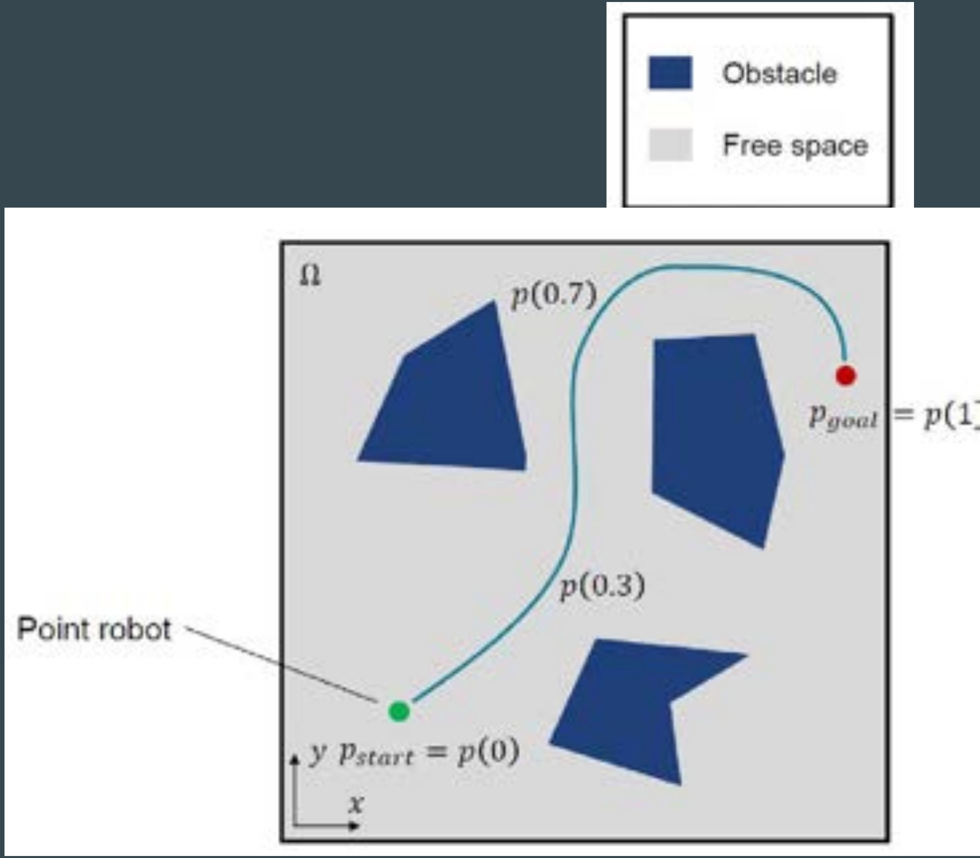
Assumptions:

- 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

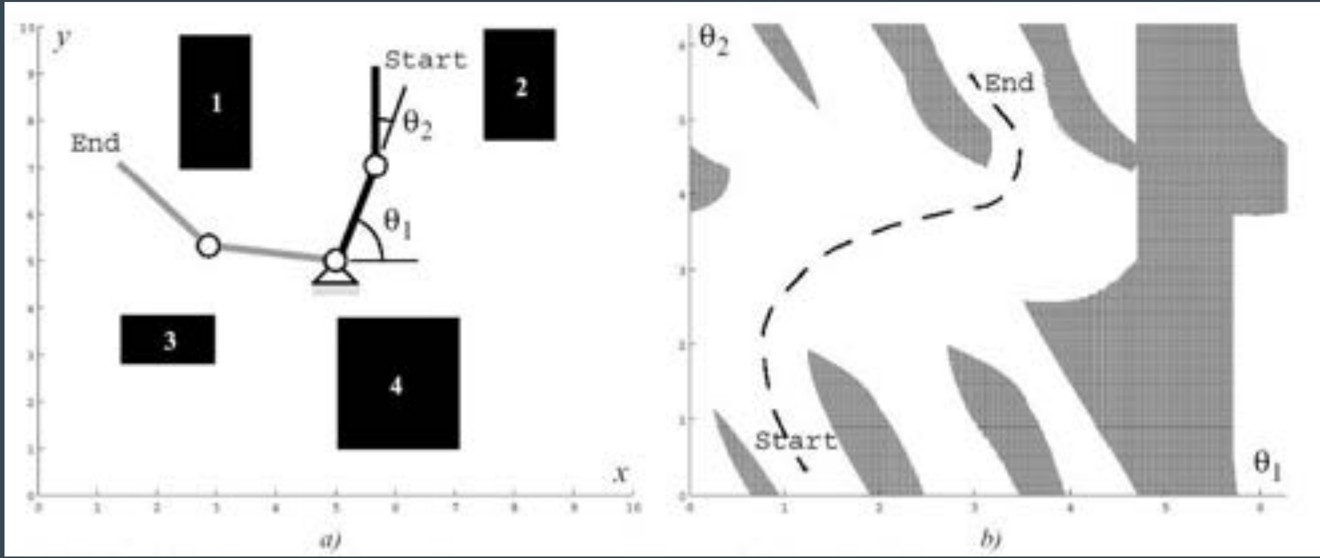
- **Workspace**
  - Reachable space within the environment

- **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

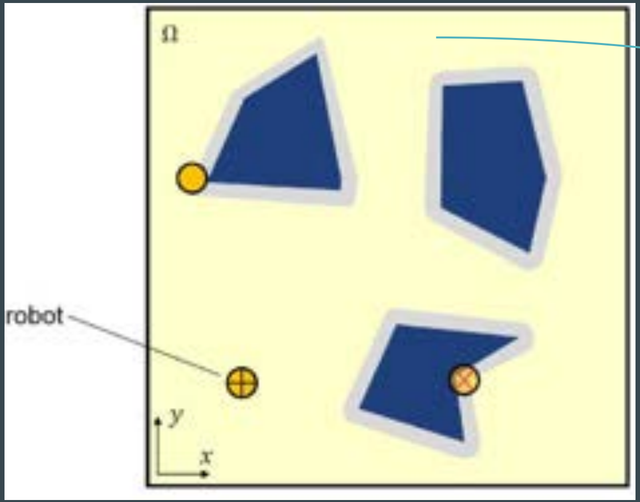


Path planning can be easier in configuration space

Configuration (C-)Space

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

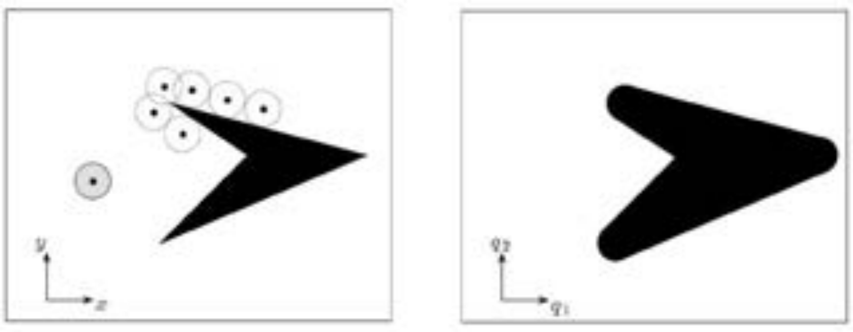
Configuration (C-)Space



Configuration space (C-space)

Configuration (C-)Space

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



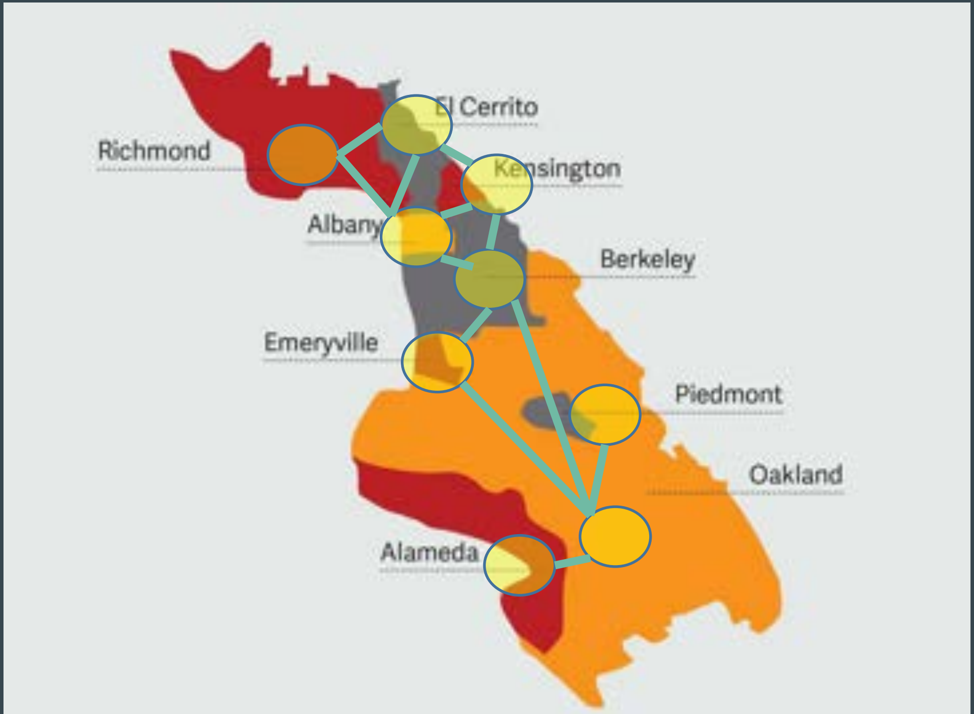
Continuous vs discrete state space representations

- 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

http://www.thornwallproperties.com/resources

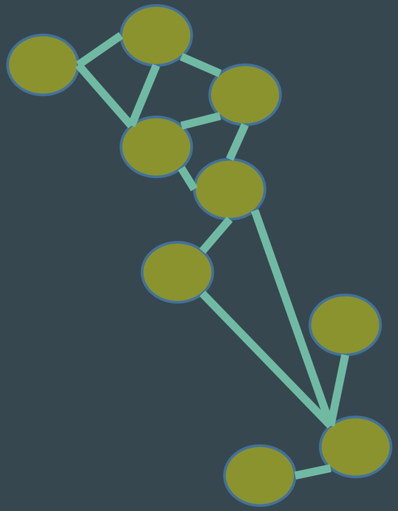
Example Graph
Neighborhood in the East Bay



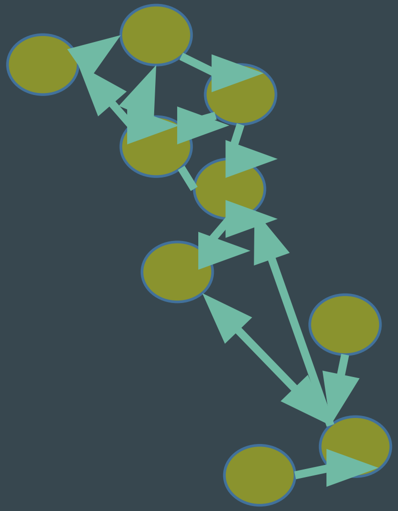
http://www.thornwallproperties.com/resources

Types of graphs

Undirected Graph

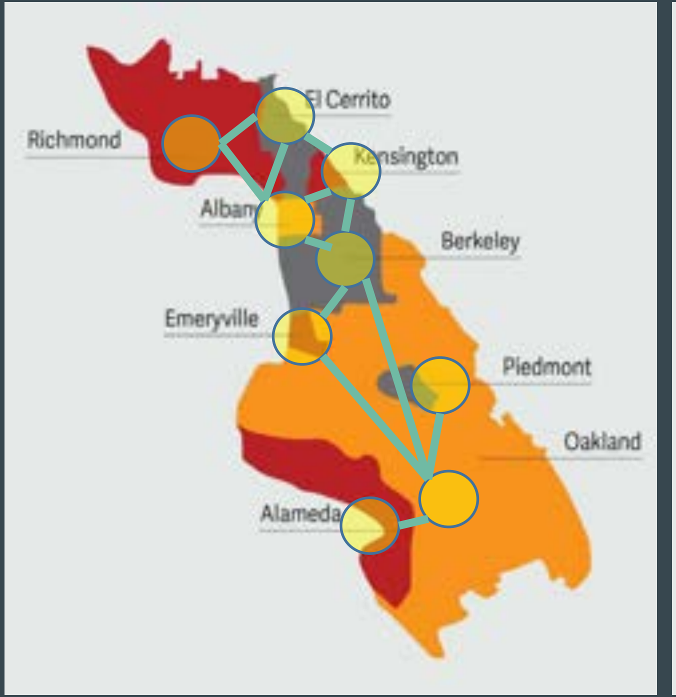


Directed Graph (Digraph)



Types of graphs

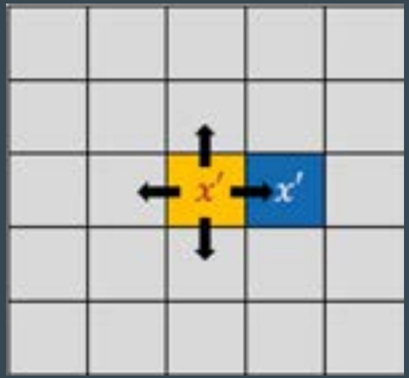
Unweighted Graph



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
  - $A(x) = \{a0, a1, \dots, an\}$
- And an associated transition function
  - $f(x, a) = x'$

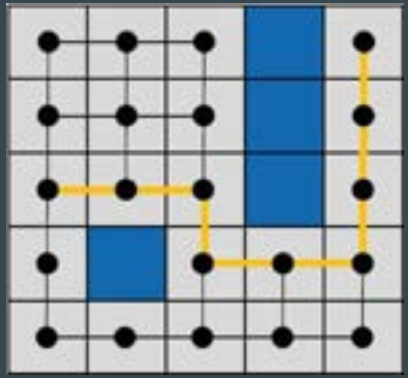


Grid to graph

- Consider:
    - States as vertices
    - Transitions as directed edges

- Add:
    - Start node, *xs*
    - Goal node, *xg*
    - Cost function $C: X \times A \rightarrow \mathbb{R}+$

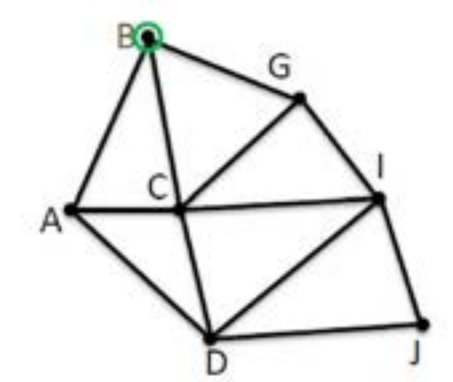
- Finding the shortest path can be treated as a graph search problem



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):	Closed:
{B}	{}

Our (BFS) queue will be FIFO:

- push (*Q.Insert*) onto the end
- pop (*Q.GetFirst*) from the front

```
FORWARD_SEARCH
Q.Insert(x_I) and mark x_I as visited
while Q not empty do
    x ← Q.GetFirst()
    if x ∈ X_G
        return SUCCESS
    forall u ∈ U(x)
        x' ← f(x, u)
        if x' not visited
            Mark x' as visited
            Q.Insert(x')
        else
            Resolve duplicate x'
return FAILURE
```

Figure 2.4: A general template for forward search.

LaValle, Steven M. *Planning algorithms*. Cambridge university press, 2006, p. 33

Breadth-first search (BFS)

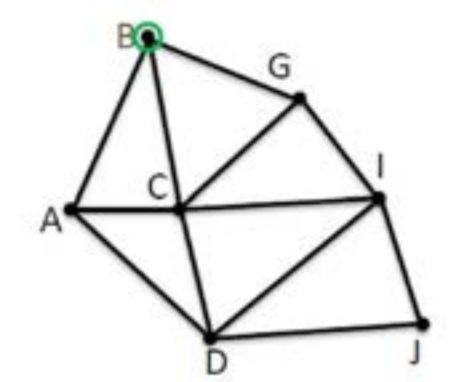
- Complete (will find the solution if it exists)
- Guaranteed to find the shortest path (number of edges, no weights)
- First solution that is found is the optimal path
- Time complexity: $O(|V|+|E|)$ V: Vertices, E: Edges
- Names in robotics:
  - Wavefront
  - Forest fire

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:

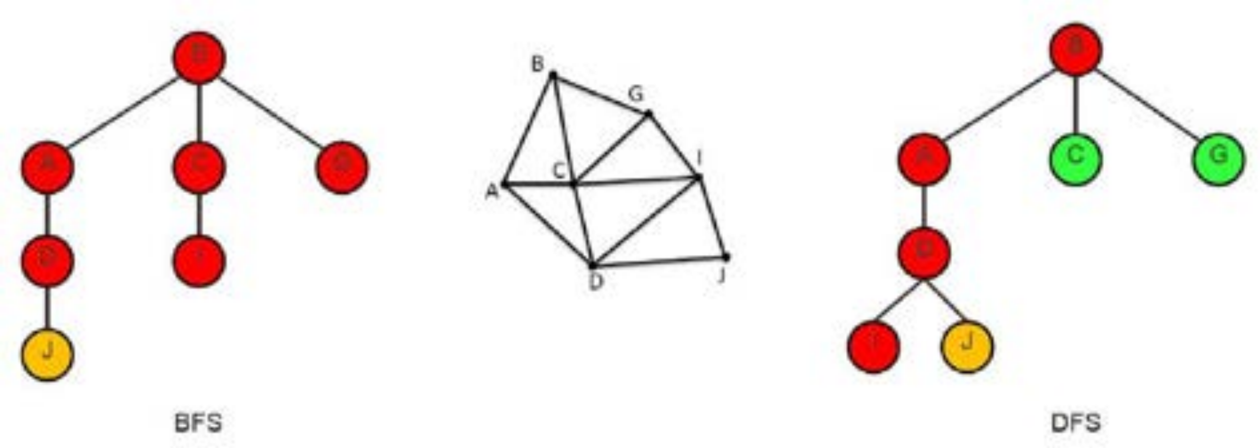
- push (*Q.Insert*) onto the front
- pop (*Q.GetFirst*) from the front

```
FORWARD_SEARCH
    Q.Insert(x_I) and mark x_I as visited
    while Q not empty do
        x ← Q.GetFirst()
        if x ∈ X_G
            return SUCCESS
        forall u ∈ U(x)
            x' ← f(x, u)
            if x' not visited
                Mark x' as visited
                Q.Insert(x')
            else
                Resolve duplicate x'
    return FAILURE
```

Figure 2.4: A general template for forward search.

LaValle, Steven M. *Planning algorithms*. Cambridge university press, 2006, p. 33

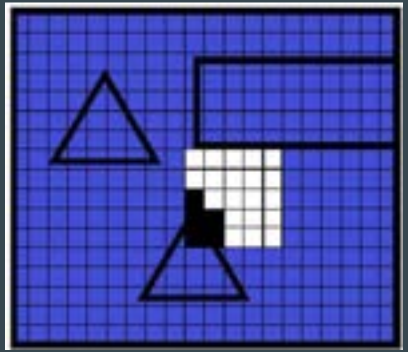
BFS vs DFS



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



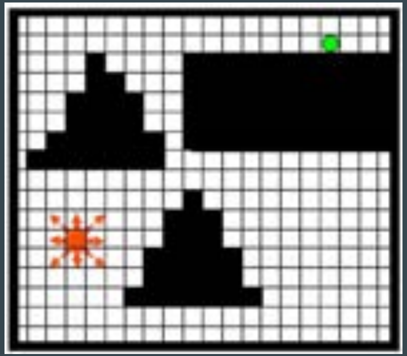
A grid square is in the C-space if it is:

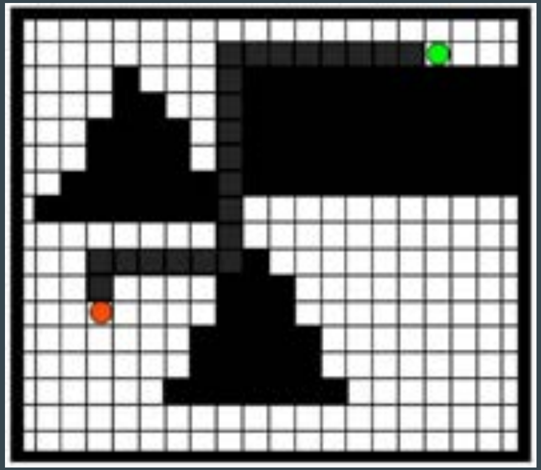
- Not inside an obstacle
- Further than the radius of the robot from all obstacle edges

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”

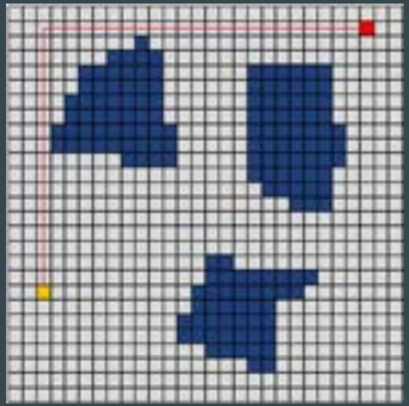
Perform search





Issues with grid-based representations

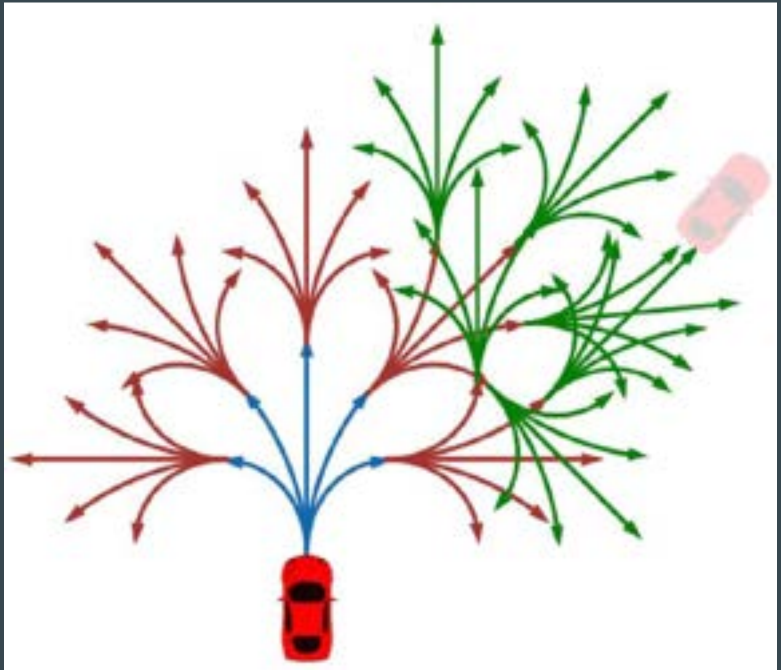
- Loss of precision
- Selecting grid resolution
- Type of output path
- Poor scaling in higher dimensions



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

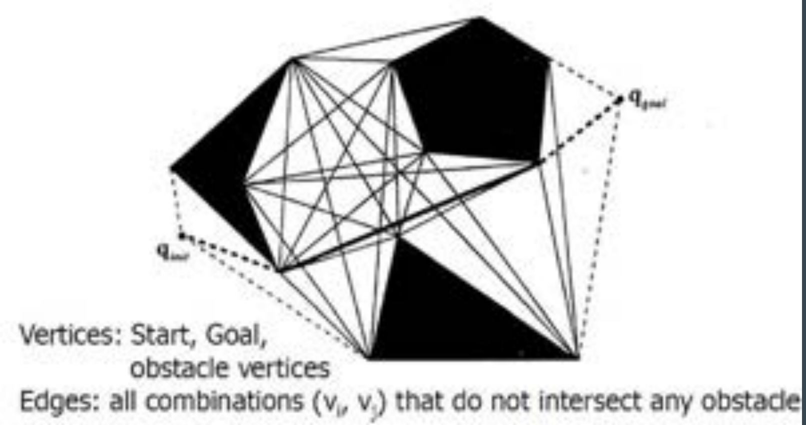
Create edges between all pairs of mutually visible vertices, and search resulting graph

Pros:

- Optimal plan
- Good in sparse environments

Cons:

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



Graph construction: Voronoi diagram

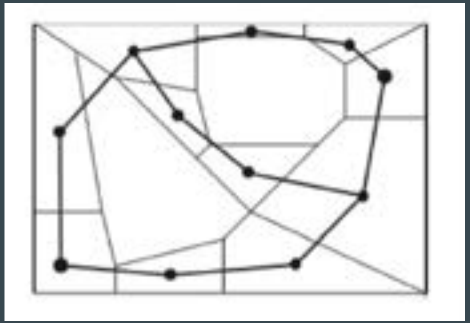
- Maximise the distance between the robot and the obstacles
- Draw equidistant lines
- Search resulting graph

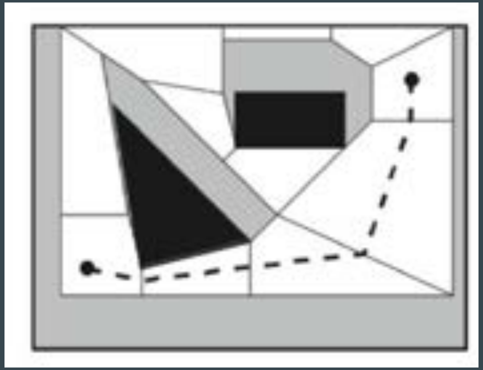
Pros:

- Complete
- Executability

Cons:

- Not optimal
- Need long-range sensing





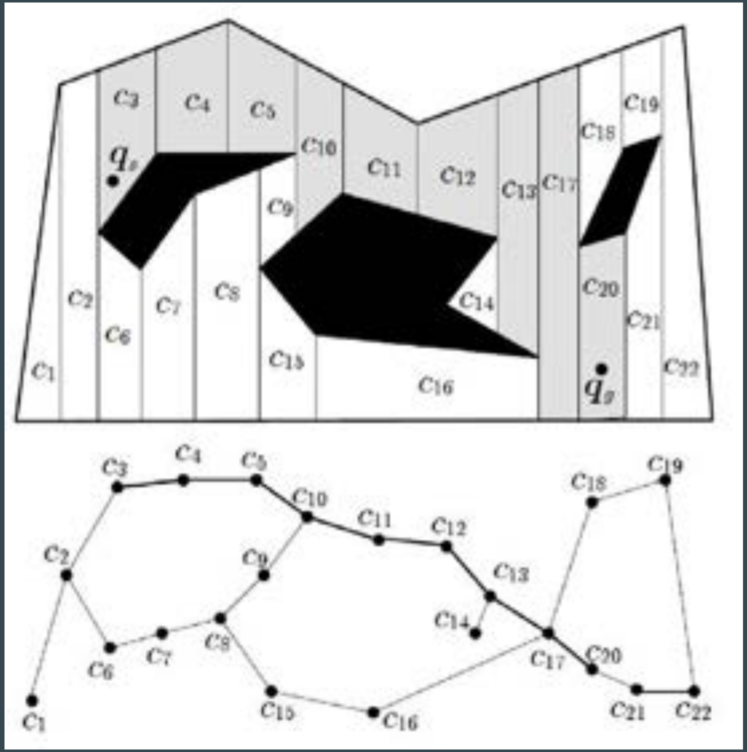
Graph construction: Exact cell decomposition

Pros:

- Complete

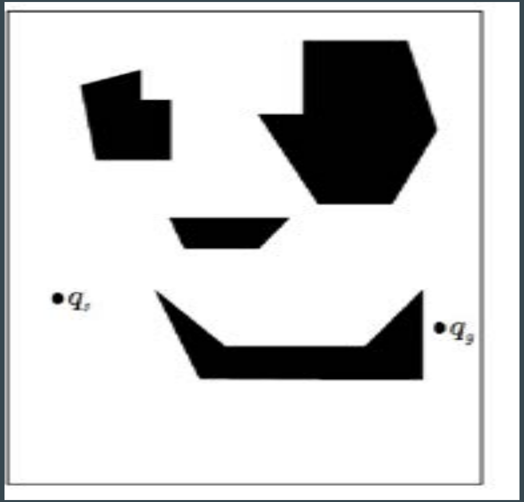
Cons:

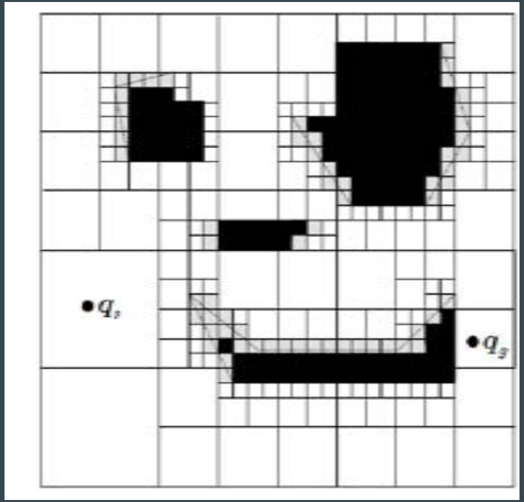
- Only good in extremely sparse environments



Graph construction:
Approximate cell decomposition

- Recursively decompose area into smaller rectangles
- Low computational complexity



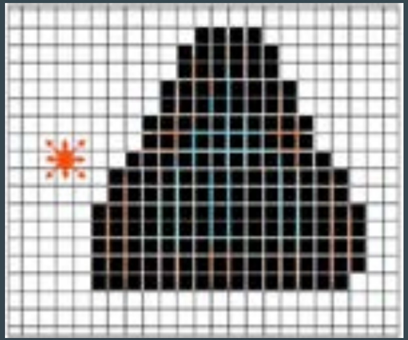


Planning as search

- 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

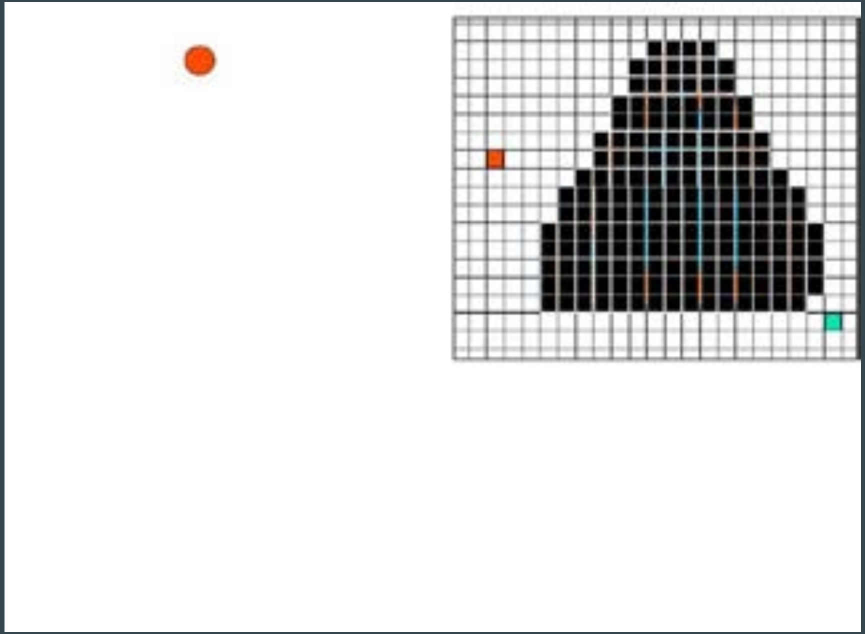
Setting up the state space

- Real space
- Configuration space
- State space

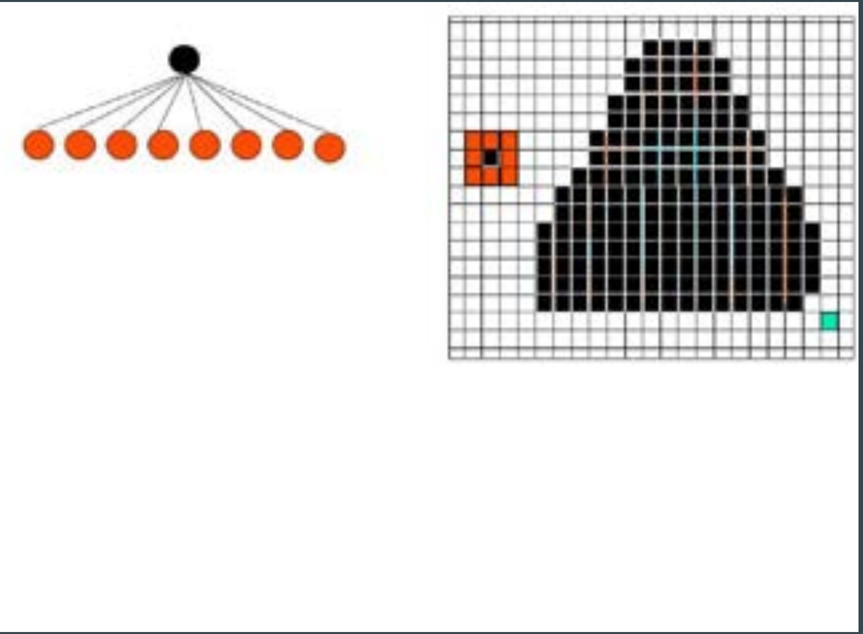


- Actions get you from one state to another
- Objective is to find a path from the start to the goal

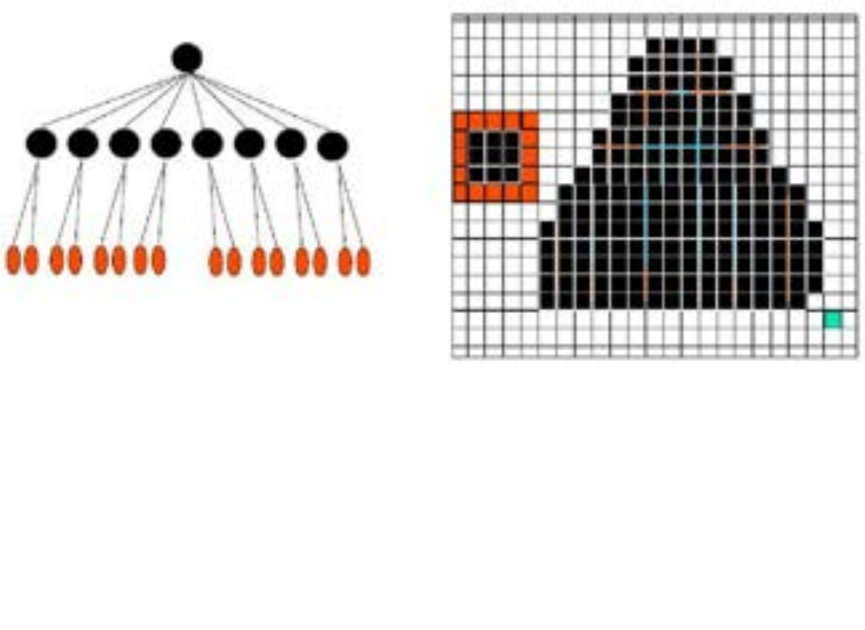
Tree search



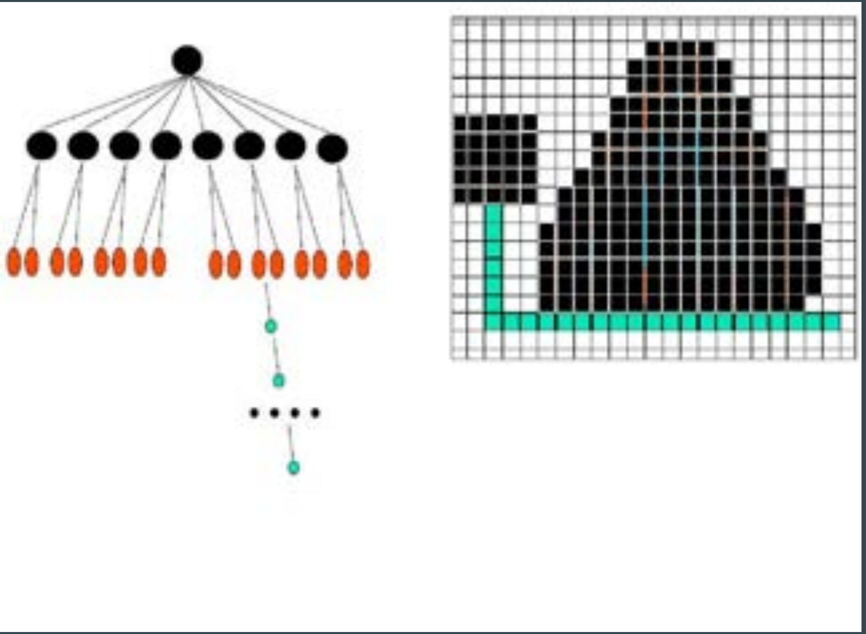
Tree search



Tree search

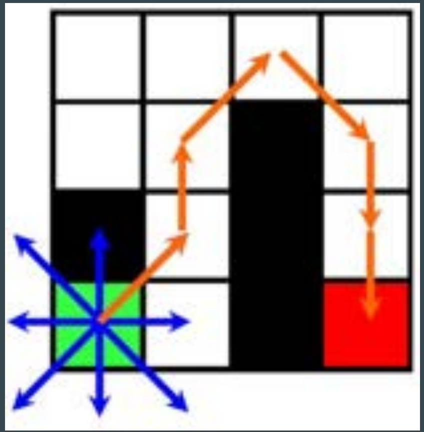


Tree search



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

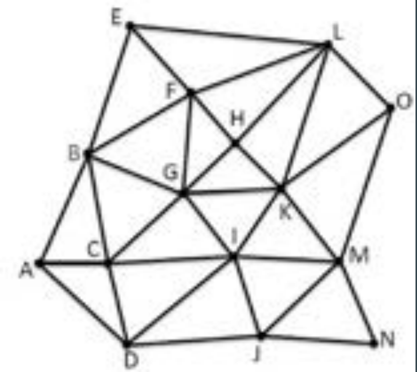
- Search over the underlying graph

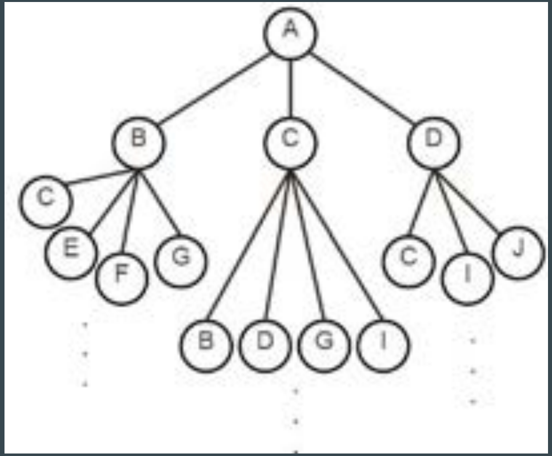
- Solve for paths from any point to any other point

- Assume all edge transitions are dynamically feasible

Search trees

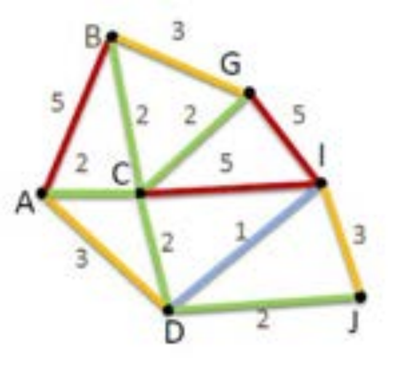
Construct a “tree” to search for optimal paths through the environment





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



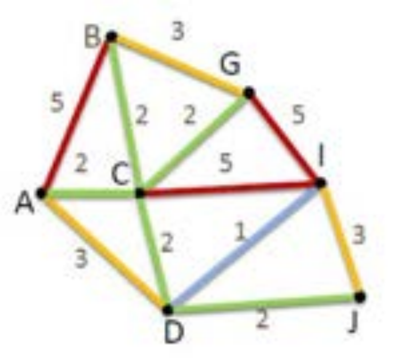
Dijkstra’s shortest-path algorithm

Open (Q): Closed:

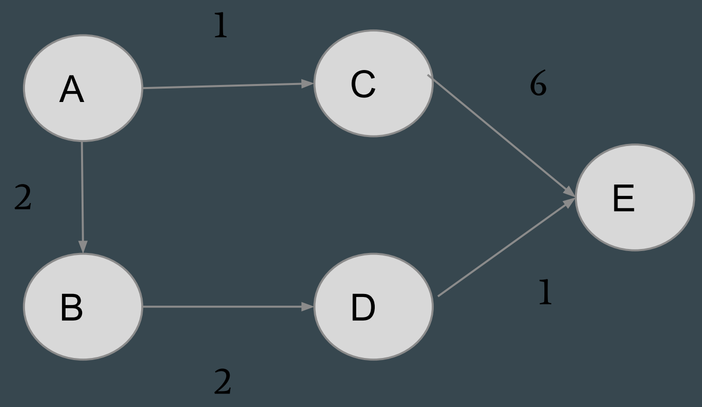
{B(0)} {B(0)}

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



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*

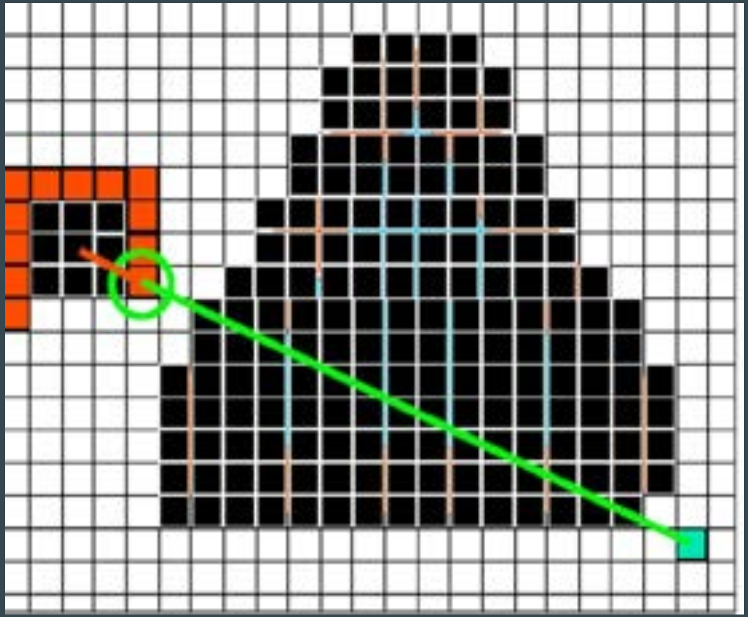
Use domain knowledge to bias the search

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

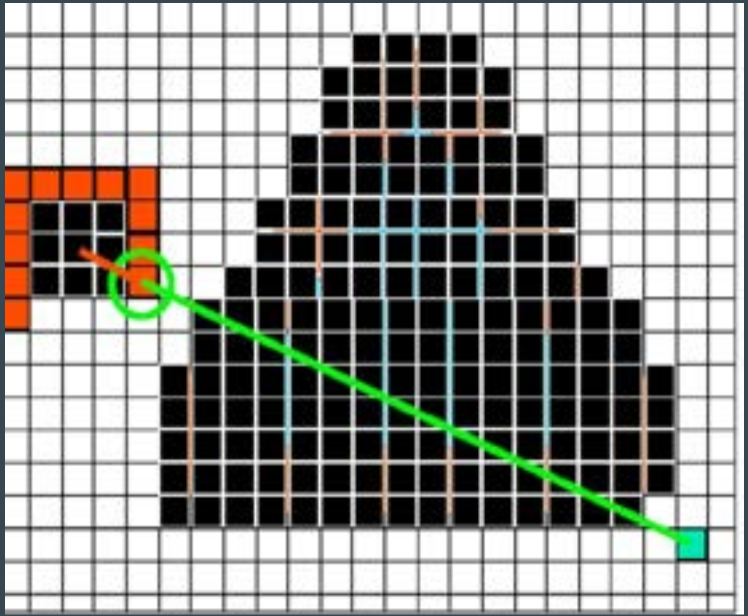
Favour actions that might get closer to the goal

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

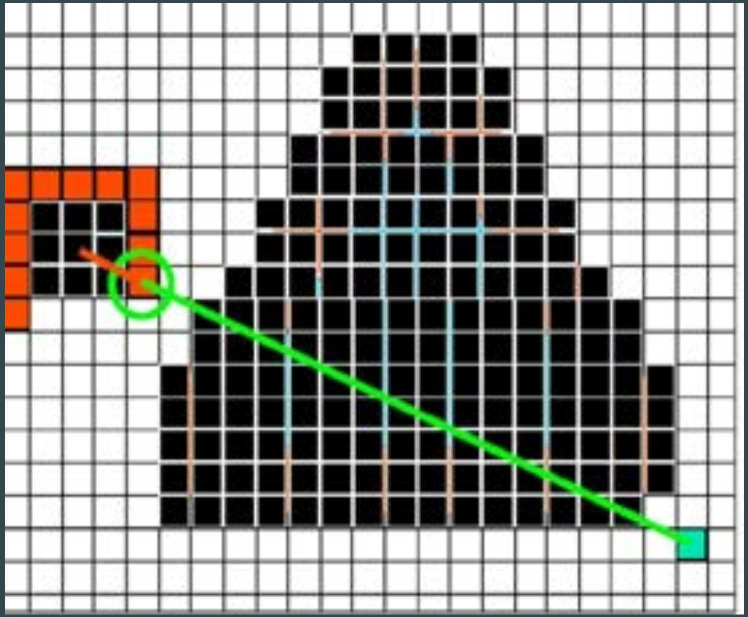
Favour actions that might get closer to the goal

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

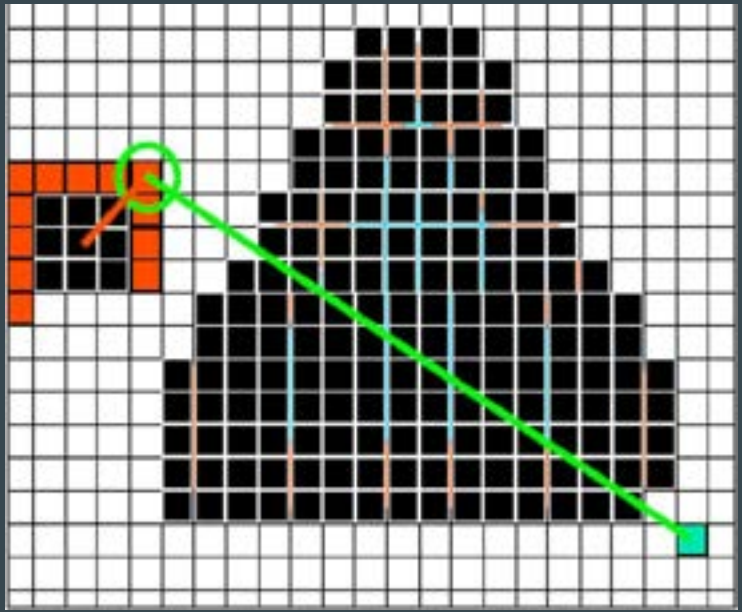
Favour actions that might get closer to the goal

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|| = $\mathrm{sqrt}(11^2+18^2)$ = 21.1

f(x) = 25.1

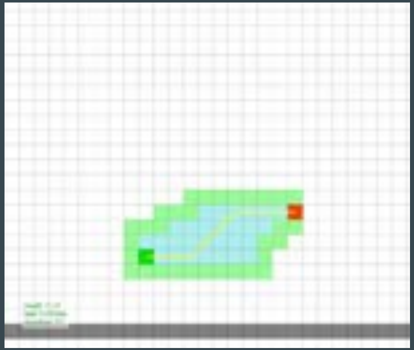
How to choose heuristics?

- 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

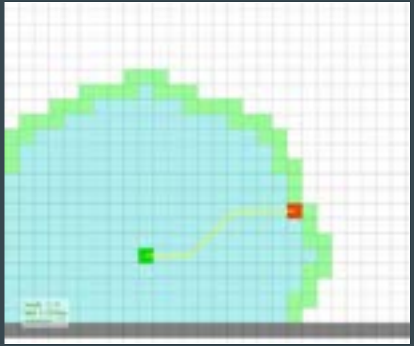
Decisions, decisions...

- **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.

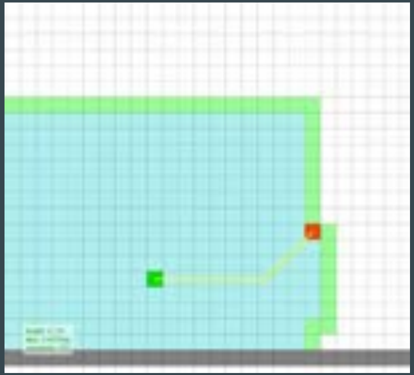
Comparison



A*



Dijkstra



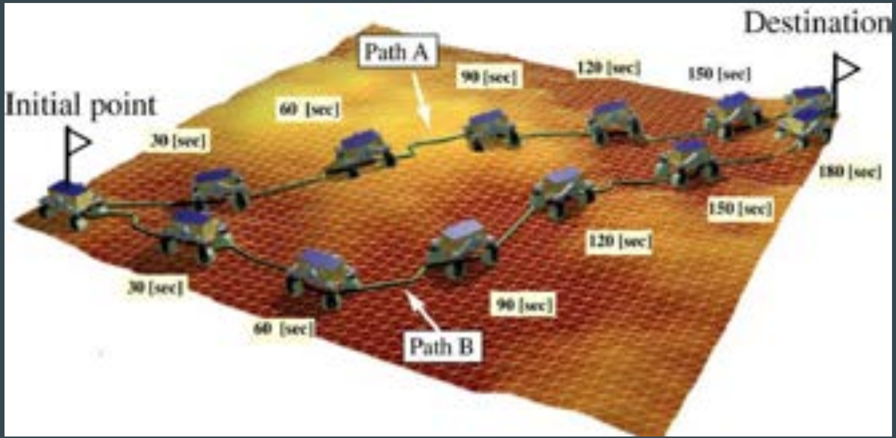
BFS

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 100x100 = 10^4
- 3D grid 100x100x100 = 10^6
- 6D grid 100 cells per d is 10^{12}

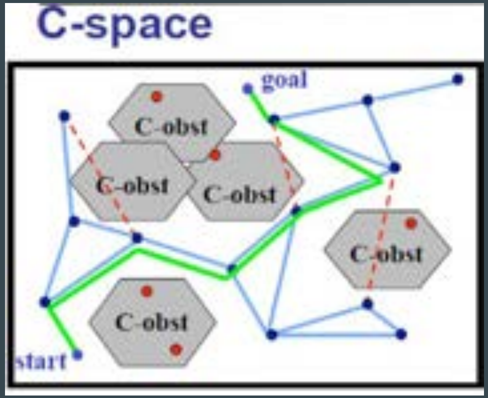


Randomised graph search

- 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)

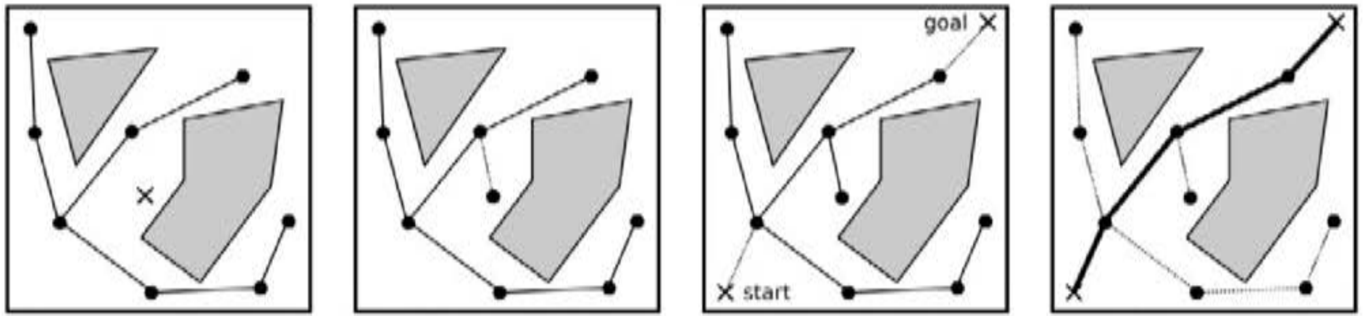
Probabilistic road maps (PRM)
(Kavraki et al. 1996)

- 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 ×, 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.

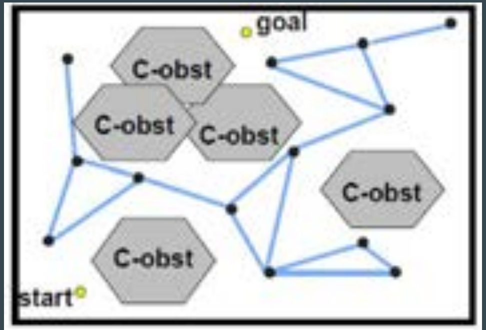
(d) A graph search algorithm is used to connect the start and goal through the roadmap.

Short et al. Recent Progress on Sampling Based Dynamic Motion Planning Algorithms, AIM 2016

PRMs

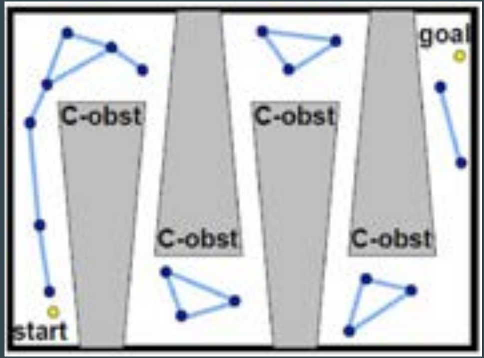
Pros

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



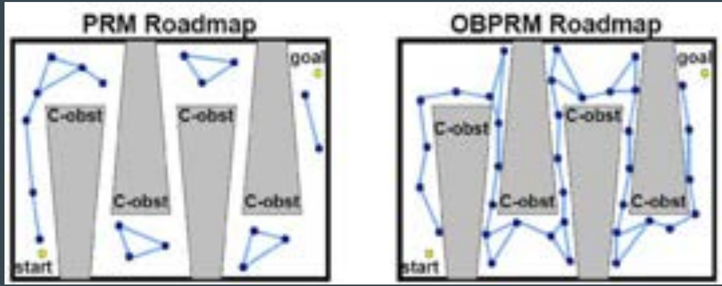
Cons

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

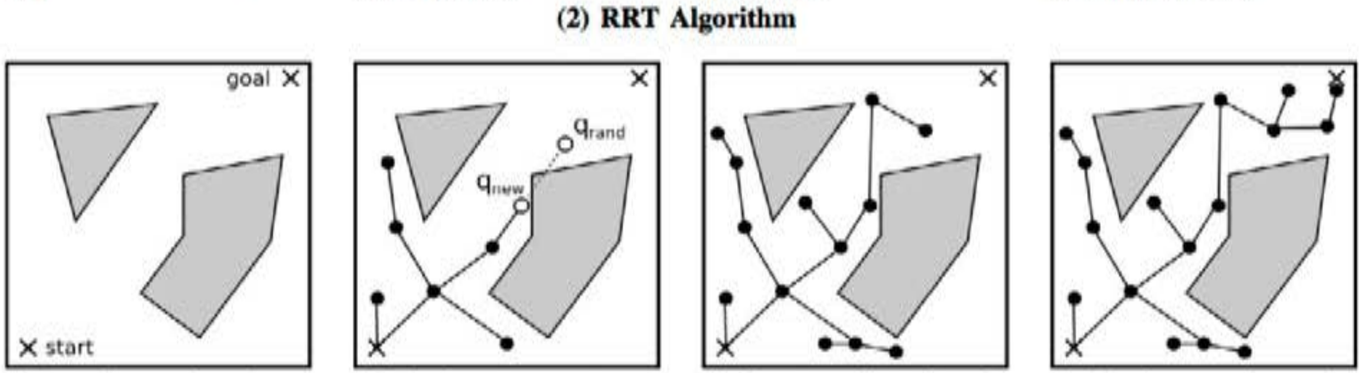


Sampling around obstacles
(Amato et al. 1998)

- 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



(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.

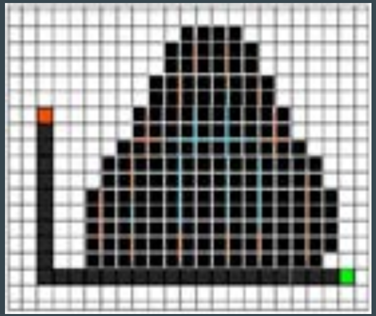
(d) The planner terminates when a node is close to the goal node. Common implementations will connect directly to the goal.

Short et al. Recent Progress on Sampling Based Dynamic Motion Planning Algorithms, AIM 2016

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

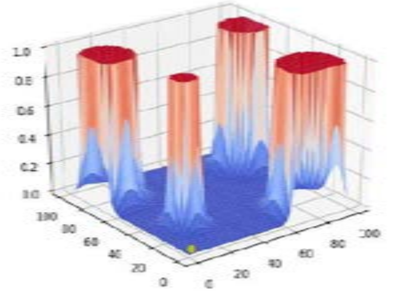
$$U_{rep} = \frac{1}{\|x - x_c\|}$$

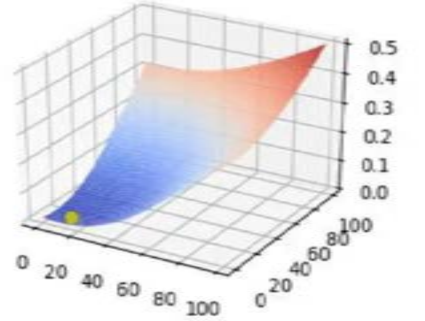
and the potential of the goal generates an attractive force

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

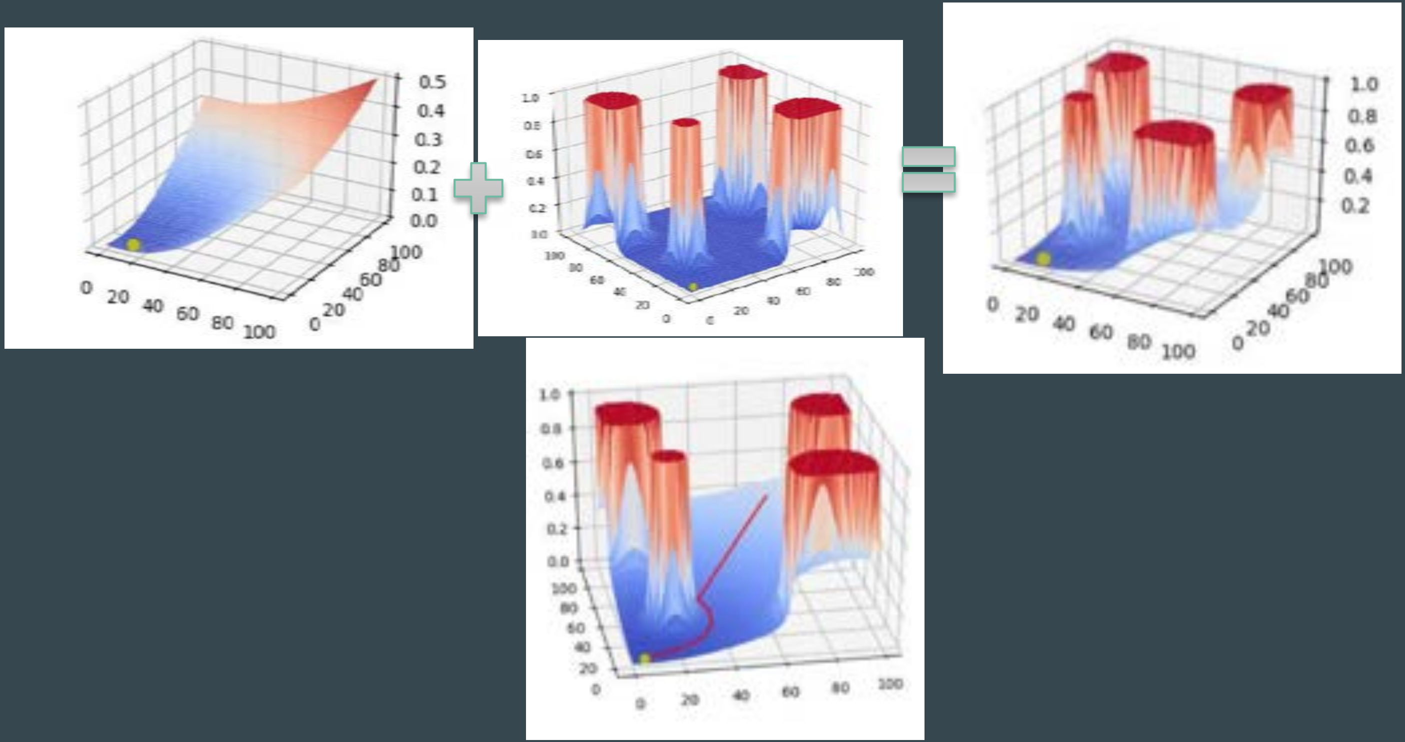
Easy and fast to compute

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 – WEEK 10

CONTROL ARCHITECTURES

Prof. Marc Hanheide

Lecture (9-11)	**Topic**	**Lecturer for Lecture**
Tuesday, 31 January 2023	Intro	Marc Hanheide
Tuesday, 7 February 2023	Robot Programming ROS	Marc Hanheide
Tuesday, 14 February 2023	Robot Sensing and Computer Vision	Jonathan Cox
Tuesday, 21 February 2023	Motion and Control	Athanasios Polydoros
ENHANCEMENT WEEK		
Tuesday, 7 March 2023	Robot Behaviour	Jonathan Cox
Tuesday, 14 March 2023	Navigation	Athanasios Polydoros
Tuesday, 21 March 2023	STRIKE	Athanasios Polydoros
Tuesday, 28 March 2023	Localisation	Athanasios Polydoros
EASTER		
EASTER		
Tuesday, 18 April 2023	Robot mapping - SLAM	Athanasios Polydoros
Tuesday, 25 April 2023	**Control Architecture**	**Marc Hanheide**
Tuesday, 2 May 2023	HRI	Marc Hanheide
Tuesday, 9 May 2023	Revision	Athanasios Polydoros & Marc Hanheide

Assessment Item 2 - Oral TCA

- Individual oral examinations will be conducted according to a released schedule with up to a maximum of 20 minutes per student
- Students much be present in front of the exam room at least 20 minute prior to their allocated slot, and must be available up to an hour past their allocated start time to allow for slippage and delays
- The examination will be audio recorded for moderation purposes. All recordings will be deleted after the exam board took place and marks are confirmed.
- Examiners will ask questions from a defined set, divided into three thematic areas: Control & Behaviour, Navigation, Control Architectures and HRI
- Example questions for revision will be released prior to the assessment

Assessment Item 2 - Oral TCA - Schedule 15 May 2023

Slot	Start Time	End Time	Marc Hanheide (DCB1103)	Athanasios Polydoros (DCB1104)
1	**11:00**	11:20	Alex Griffiths (25282168)	--
2	**11:25**	11:45	Samuel Petrie (18679110)	Haani Mahmood (25270480)
3	**11:50**	12:10	Elliot Smith (26684396)	Jack Sutcliffe (25300646)
4	**12:15**	12:35	Brandon Gruber-Murray (19705193)	Eleanor Docherty (17686350)
5	**12:40**	13:00	Mira Melhem (27362334)	Malquiel Owens (19697949)
6	**13:05**	13:25	Thomas Wilcockson (25273418)	Josh Cooper (25466645)
7	**13:30**	13:50	Kieran Steel (19701658)	Hala Nassar (27359556)
8	**13:55**	14:15	Layan Joudeh (27367892)	Mia Hartley (25179600)
9	**14:20**	14:40	Jacob Swindell (25105508)	Joseph Smith (19701852)
10	**14:45**	15:05	Vladislav Rankov (17643529)	Alexander Staniforth (15610188)
11	**15:10**	15:30	Philip Mackay (25051805)	Alex Leather (25215067)
12	**15:35**	15:55	Harry Ledgerton (25052510)	Michael Edwards (25072301)

Who said robots were easy?

Source: IEEE Spectrum – full video at https://www.youtube.com/watch?v=g0TaYhjpOfo

Problems…

Humans

Complex Environment

Noisy / Unpredictable Environment

Actions don't always lead to intended consequences

Misleading sensors

Responses are too slow

Complex behaviour

Why Control Architectures?

- Seen a range of competencies in previous weeks…
  - Sensing, motion, navigation, mapping, localisation, etc
  - And how to use these in code

- 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

Title T

DELIBERATIVE ARCHITECTURES

HYBRID ARCHITECTURES

REACTIVE ARCHITECTURES

COGNITIVE ARCHITECTURES

DELIBERATIVE ARCHITECTURES

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

- Symbolic AI

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

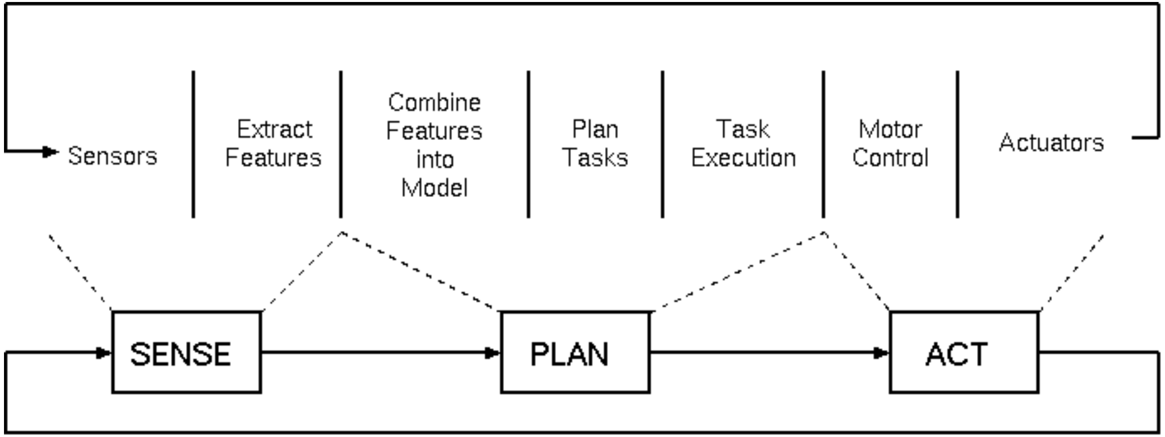
- Partly inspired by human introspection

Basic process:

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

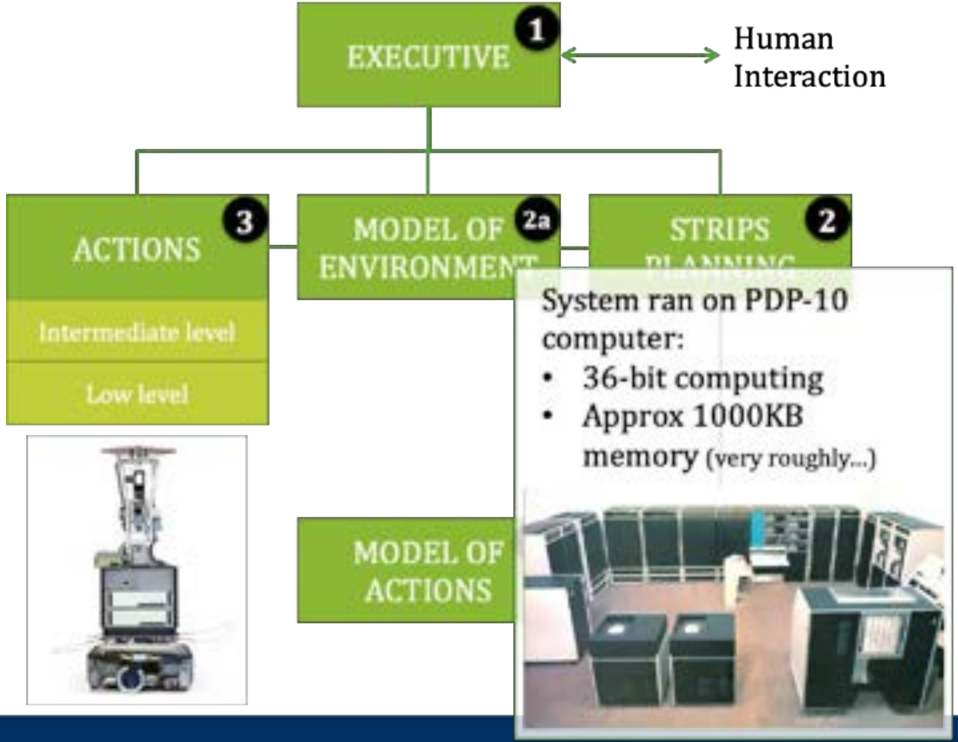
Horizontal decomposition

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




SHAKEY

Shakey 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

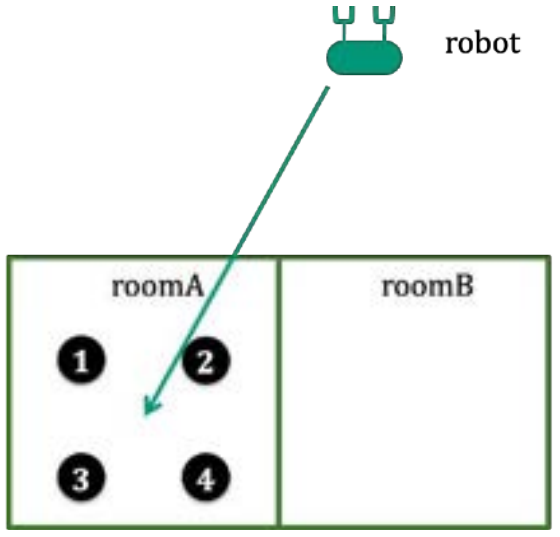
- Planning Domain Definition Language (PDDL)
  - STRIPS plus extensions, common assumptions, benefits/shortfalls…

- 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

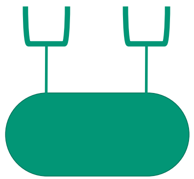
A brief PDDL example: World

```
(define (problem strips-gripper-x-1)
   (:domain gripper-strips)
   (:objects rooma roomb ball4 ball3 ball2 ball1 left right)
   (:init (room rooma)
          (room roomb)
          (ball ball4)
          (ball ball3)
          (ball ball2)
          (ball ball1)
          (at-robby rooma)
          (free left)
          (free right)
          (at ball4 rooma)
          (at ball3 rooma)
          (at ball2 rooma)
          (at ball1 rooma)
          (gripper left)
          (gripper right))
```



A brief PDDL example: Robot

```
fine (domain gripper-strips)
 :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 ?room ?gripper)
    :precondition  (and  (ball ?obj) (room ?room) (gripper ?gripper)
             (at ?obj ?room) (at-robby ?room) (free ?gripper))
    :effect (and (carry ?obj ?gripper)
         (not (at ?obj ?room))
         (not (free ?gripper))))

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



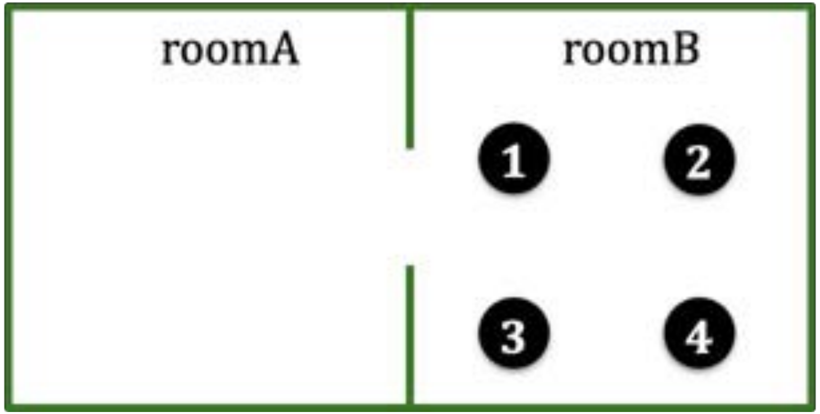
Robot can:

- Move
- Pick
- Drop

With left and right grippers

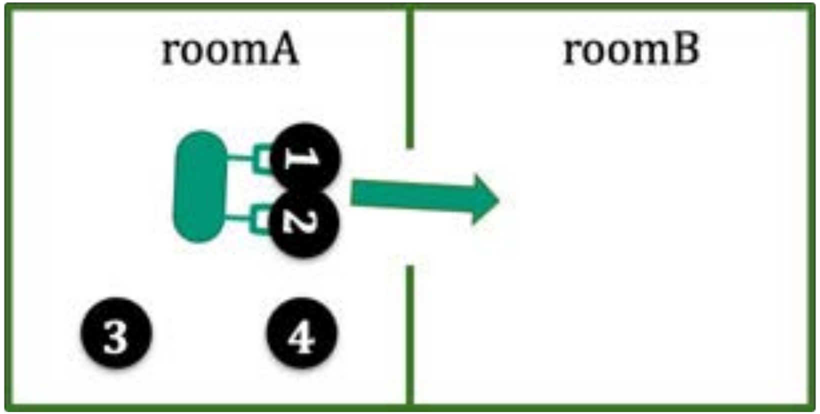
A brief PDDL example: Goal

```
(: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 effected 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

Reactive Architectures

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:

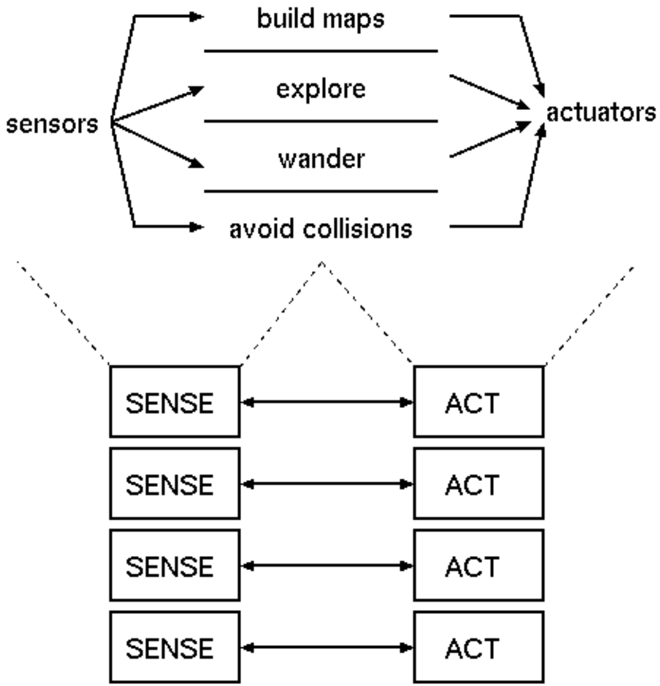
- Sensory input acquired
- Multiple parallel behaviours result in overt agent action(s)

SENSE ⟷ ACT

REACTIVE ARCHITECTURES

Vertical decomposition

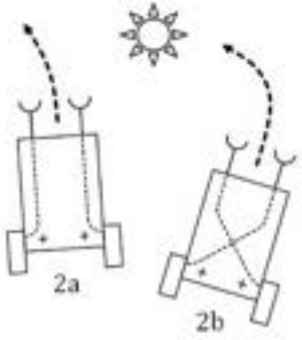
- Contrast to the deliberative approach
- Vertical decomposition of tasks
  - Simultaneously operating pairs of sensing and acting



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

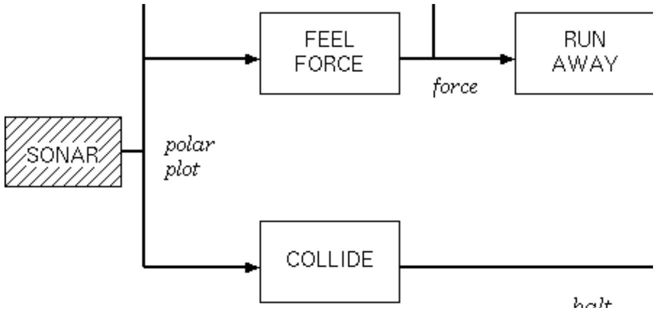
Behaviour-based Robotics



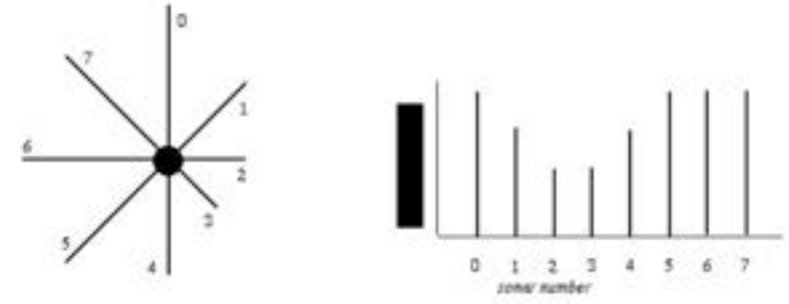
- These are typically reactive
  - Tightly coupled to sensory information (no “planning”)
  - Lacking in (or only minimal) internal state
  - Hence fast acting
- In the design of the behaviours, strong implicit role for the embodiment of the robot
  - i.e. the behaviour depends on the specific array 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

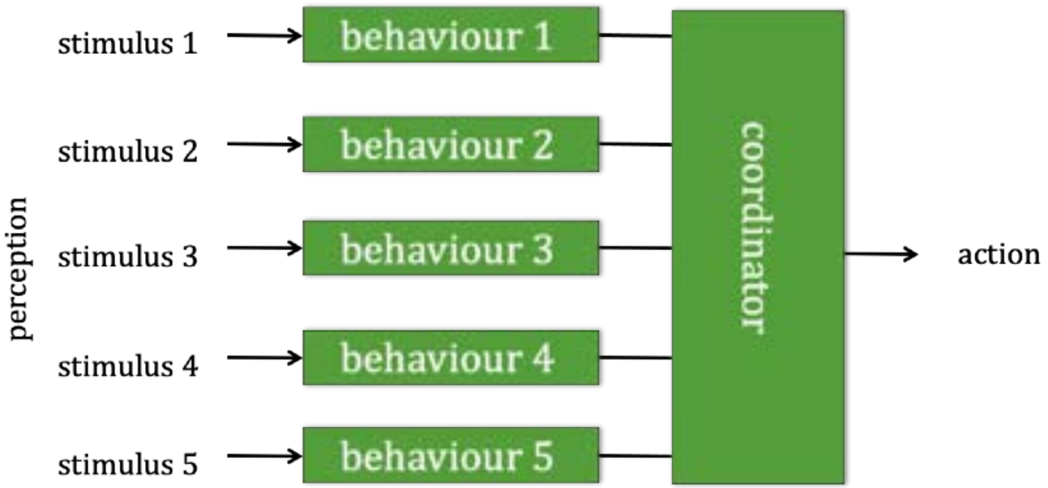


- one way to deal with this limitation is to use ego-centric representations
  - Provides some structure: helps with choosing what to do



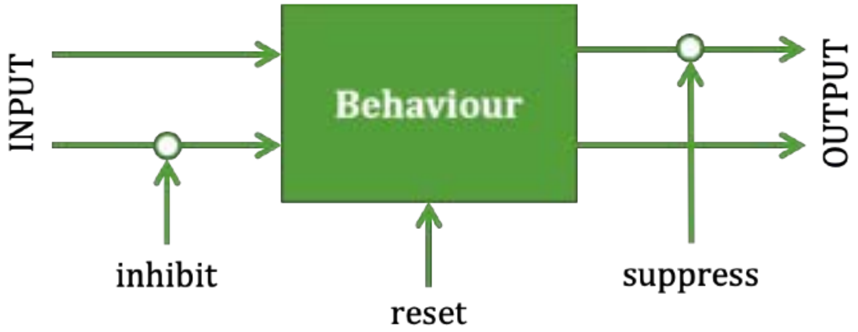
Multiple Behaviours *(recap)*

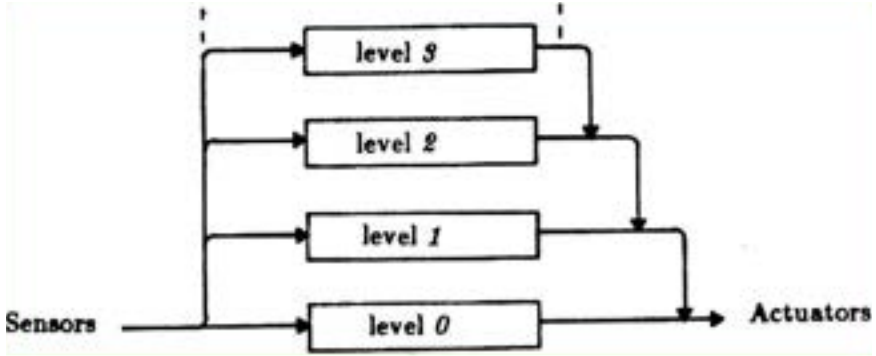
- What role for the coordinator?
  - Competitive: e.g. winner-takes-all
  - Cooperative: e.g. blending outputs through addition
  - Hybrid: e.g. activation/inhibition dynamics (Maes, 1989)



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



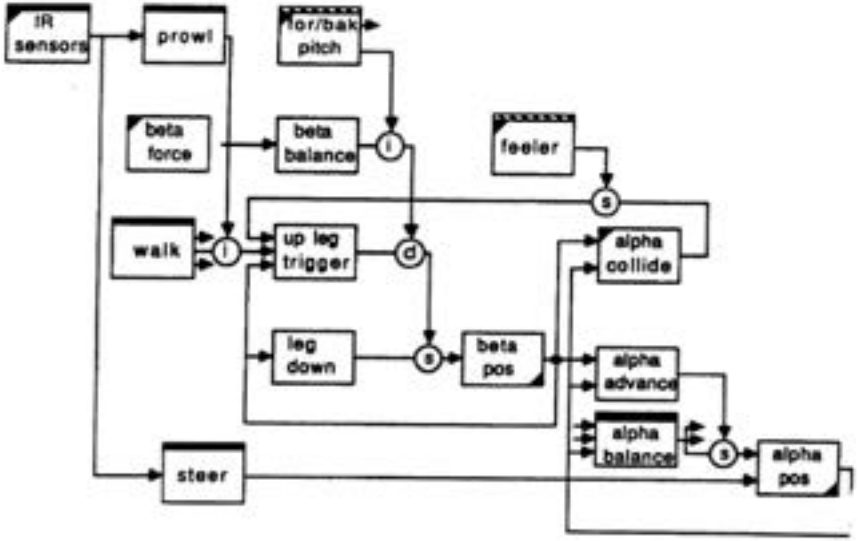


Example: Genghis

- Rodney Brooks, late 80's
  - Use of the subsumption architecture
  - (nearly) independent behaviour-based control for each leg

Example: Genghis

- Example architecture of Genghis
- Blocks:
  - Sensor input
  - Behaviours
  - …
- Note the relative complexity of the inter-connections
- <u>Hand-designed and tuned behaviours</u>



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

Contrasting

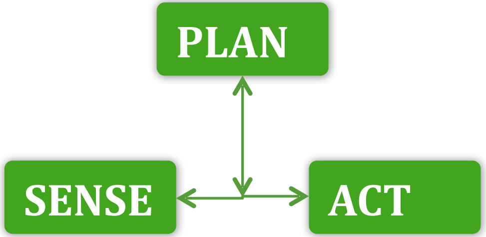
Deliberative (symbolic)	Reactive (reflexive)
Speed of response	
Predictive capabilities, completeness of world models	
Needs internal representation Slow response High-level intelligence (cognition)	No internal representation Real-time response Low-level intelligence Simple (analogue) computation

Combine it!

Trying to get the best of both Deliberative and Reactive paradigms

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

HYBRID ARCHITECTURES

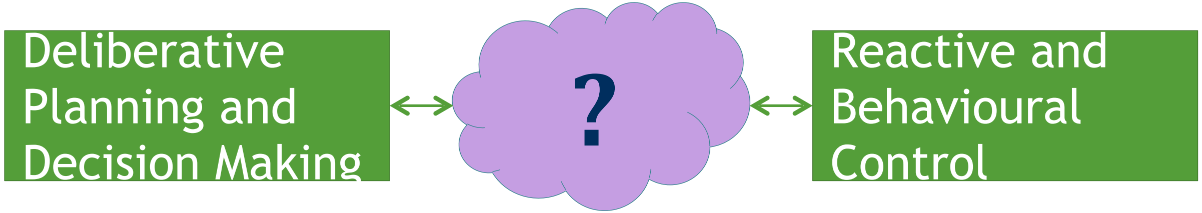


Multiple levels of control, each focused on a different aspect

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?

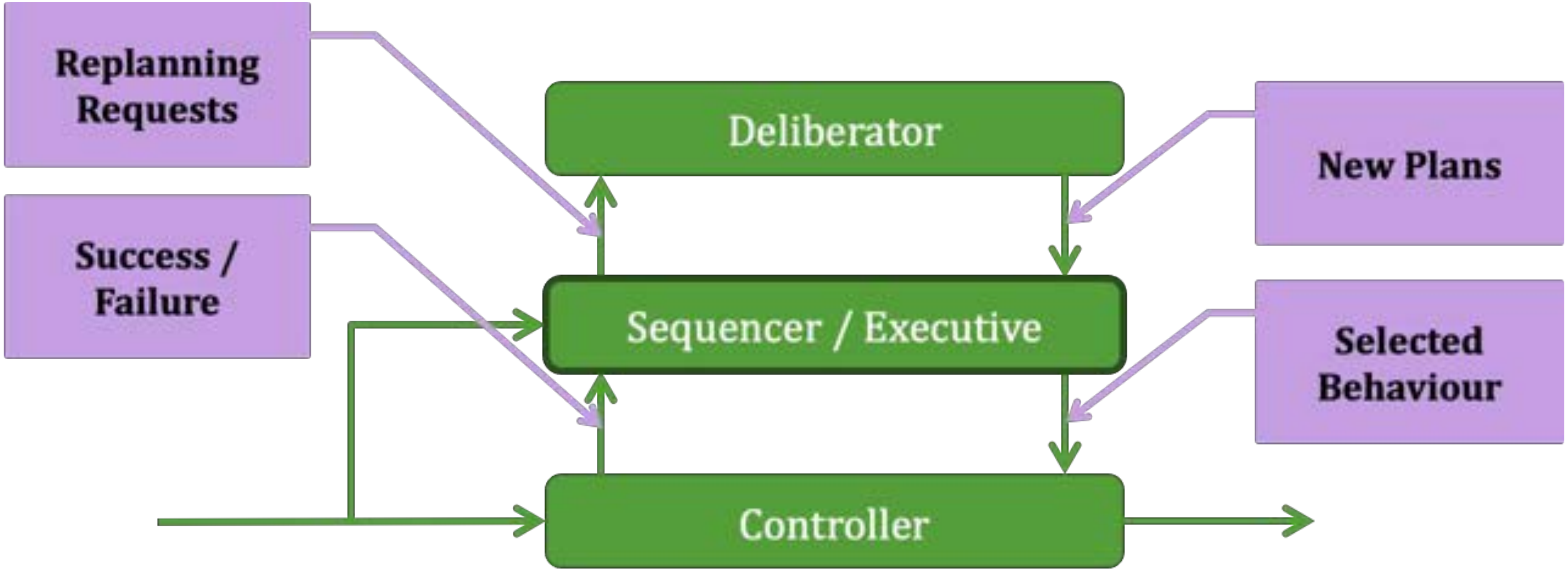


Temporal decomposition

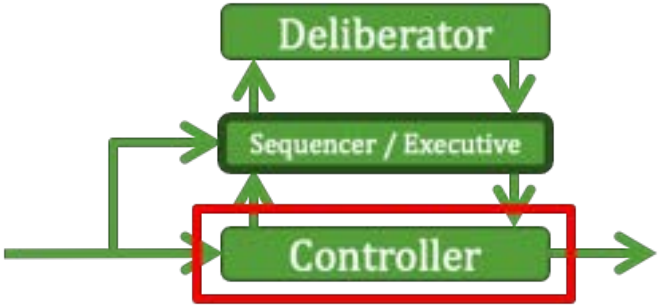
- In other architectures:
  - Horizontal decomposition of the task in Deliberative architectures
    - Can be slow…
  - Vertical decomposition of tasks in Reactive architectures
    - Can be too quick? (sensitive to noise)

- Resolve this by using time-appropriate processes:
  - Different layers with different ‘speeds’ of processing

Three Tier (3T) Architectures



Controller

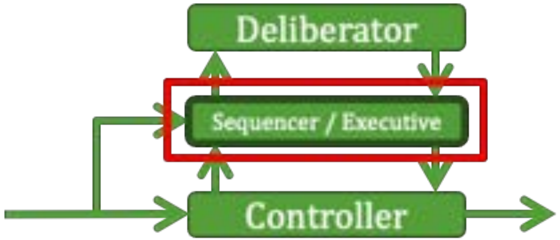


- Library of Behaviours
  - May be handcrafted
  - E.g. the behaviour-based approach

- Must be fast:
  - Avoiding internal state (except to estimate state), planning, etc
  - Stable closed-loop control

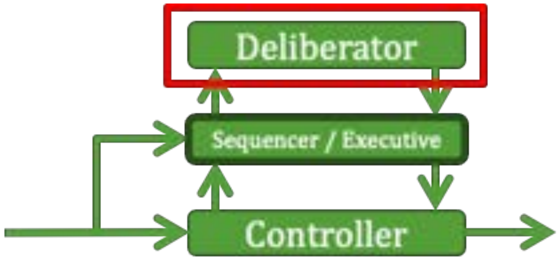
- Must be able to detect failure
  - This allows the Sequencer/Executive to call another behaviour, or call for a re-planning

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…

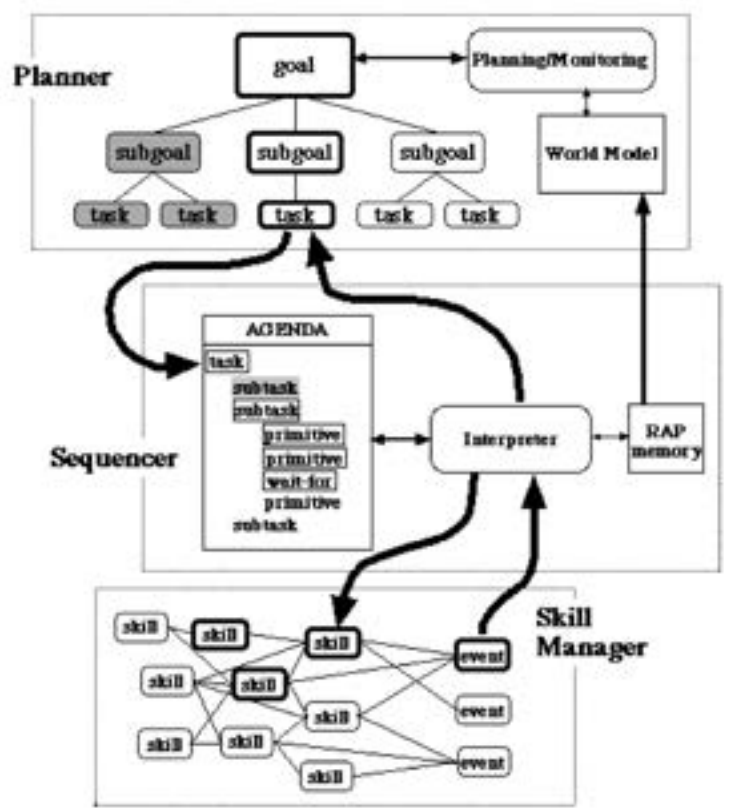
- What do I need to take into account?
  - static • Where is it?
  - static • How to get there?
  - dynamic • Walk and Open doors… (physical conventions)
  - dynamic • Don't walk into people… (personal conventions)
  - dynamic • Queue… (cultural conventions)
  - dynamic • Pay… (legal conventions)
  - target! • Refreshment

Both planning and reactive components required to solve the task

Example Systems: NASA

- A 3T architecture used by NASA
  (right)
  - 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



From Kortenkamp et al (1998)

Example Systems: Google Car

WAYMO

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

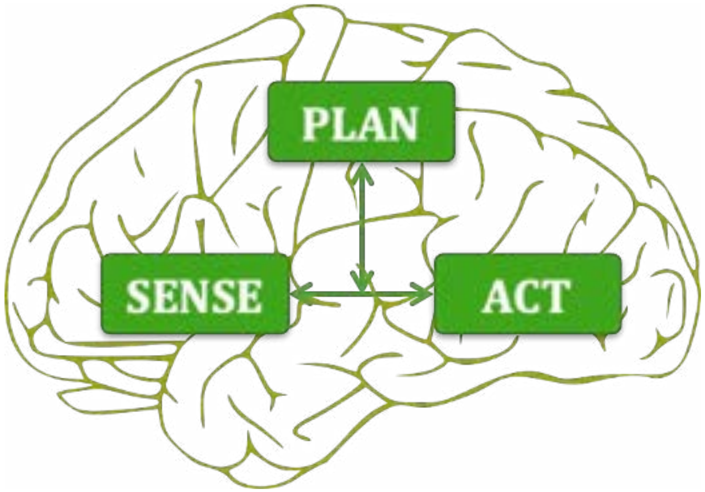
- (Software) Engineering challenges due to complexity!

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



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 overt 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

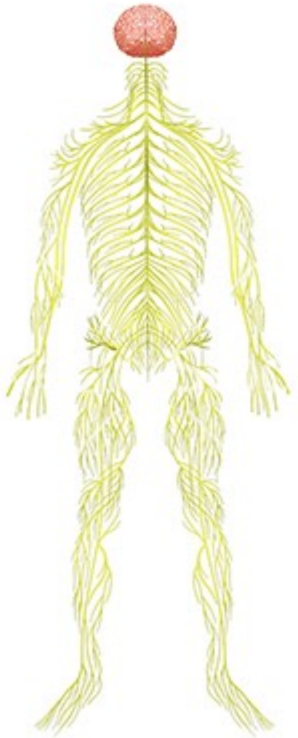
- Structure and Process
  - The mechanisms (and knowledge) required to achieve this

Why use Cognitive Architectures?

Prof. Ashok Goel
Udacity/Georgia Tech: https://www.youtube.com/watch?v=bjEyYKaXUvw

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 week…

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)

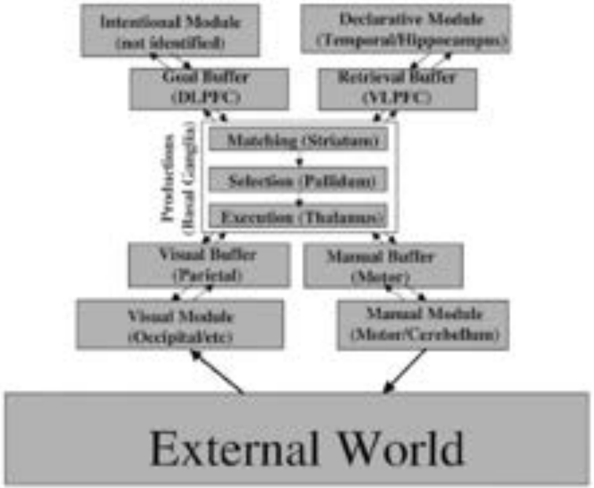
2. Try to model fundamental principles of organisation of cognition, and implement these
    - A “bottom-up” perspective
    - Try to match to certain aspects of (human) behaviour
    - Typically derived from data closer to biology

There are many, many cognitive architectures, two examples follow

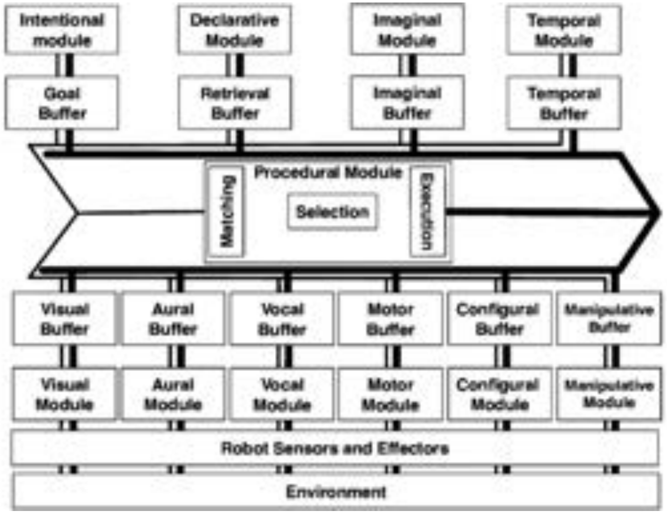
Example of Top-down: ACT-R

- 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/



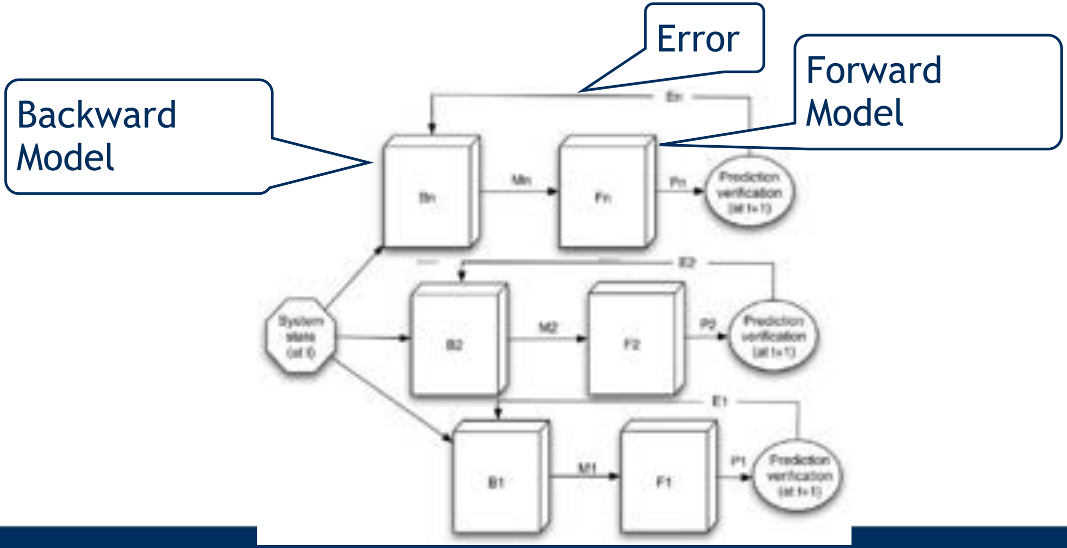
Example of Top-down: ACT-R



Images taken from (Trafton et al, 2013)

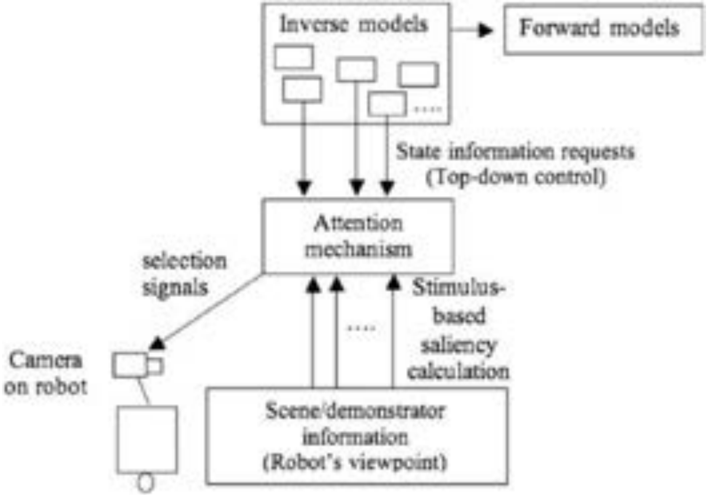
Example of Bottom-up: HAMMER

- 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

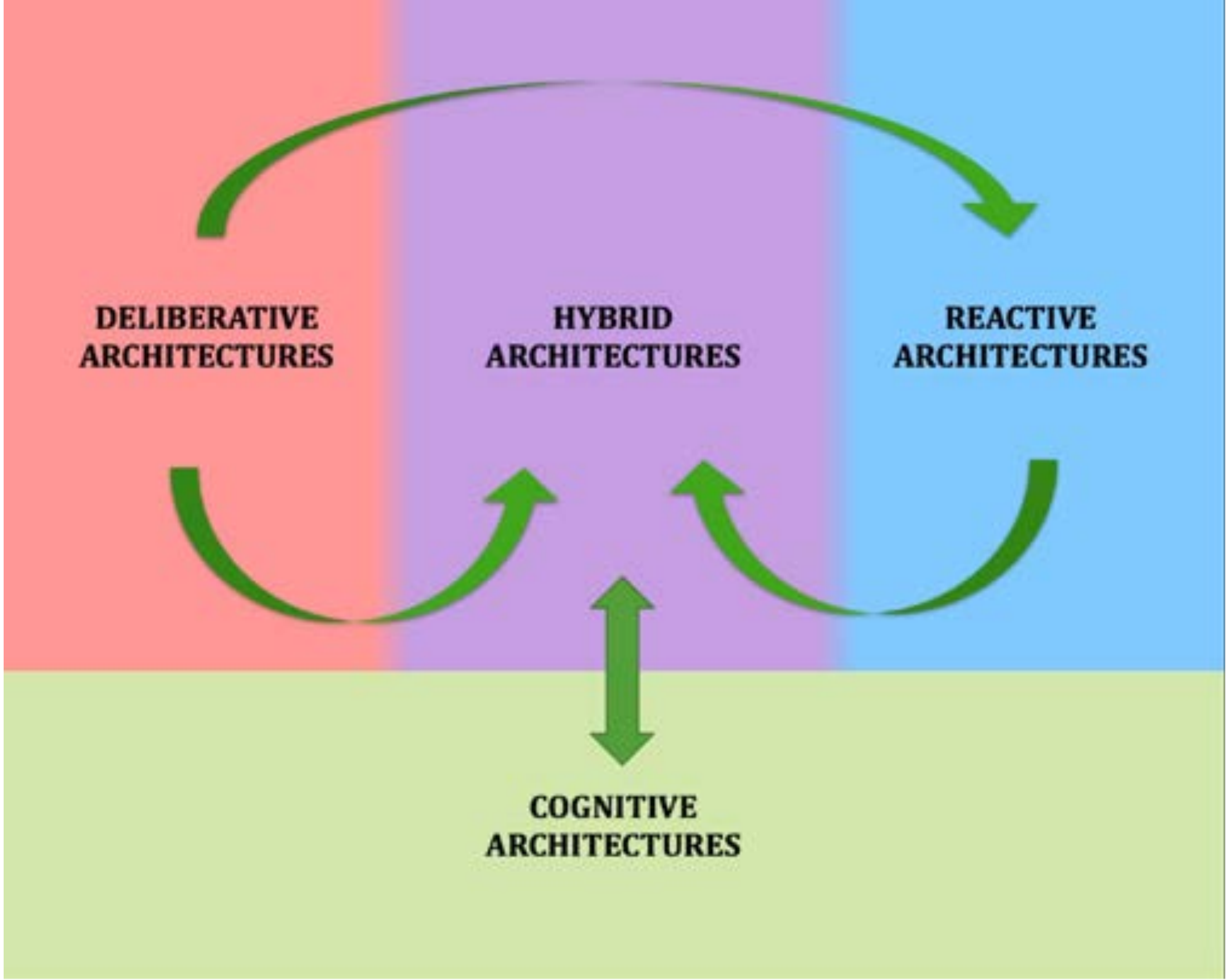


Prediction of intentions

See https://www.youtube.com/watch?v=CaH82_WzAeQ 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?


DELIBERATIVE ARCHITECTURES
HYBRID ARCHITECTURES
REACTIVE ARCHITECTURES
COGNITIVE ARCHITECTURES

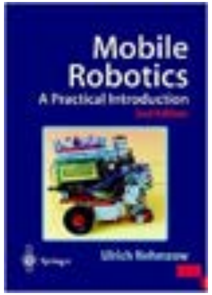
Tip of the Iceberg…

- 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…

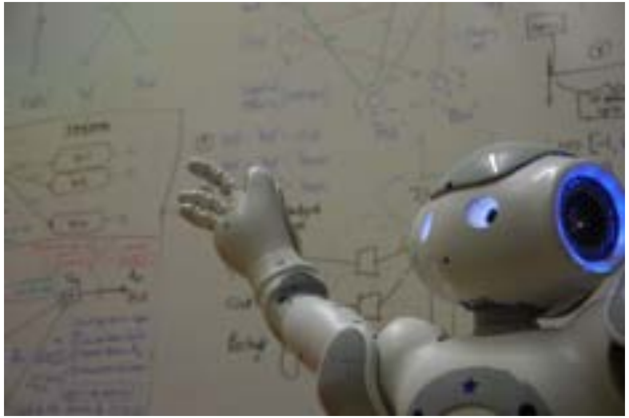
References / Reading

- 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.



Thank you

See you in next week's lecture!



CMP3101 AMR – WEEK 12

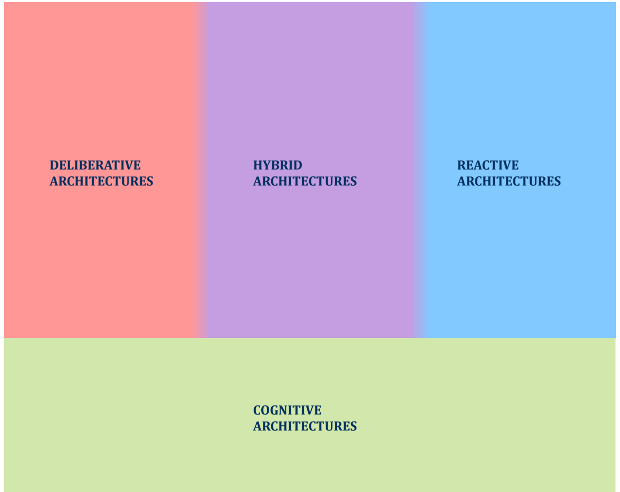
HUMAN-ROBOT INTERACTION

Prof Marc Hanheide (with gratitude to Dr Paul Baxter)

mhanheide@lincoln.ac.uk

Last Week…

- Control Architectures for Autonor
- 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/

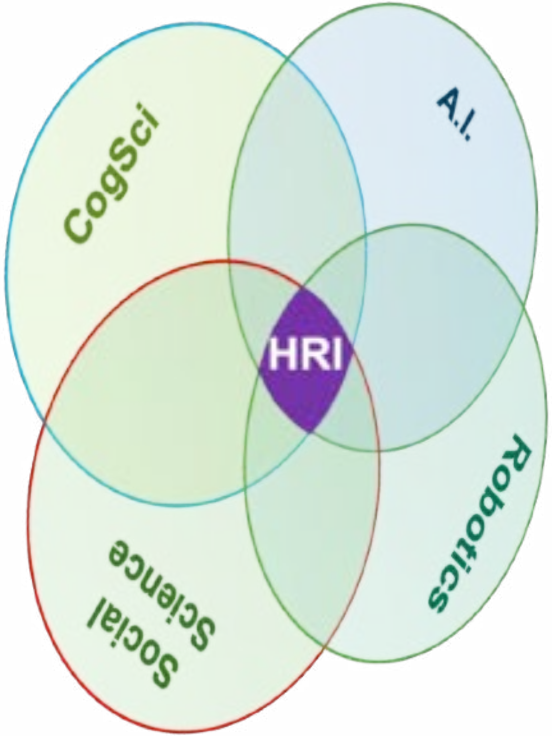
- Some overlaps with Human-Computer Interaction, so refresh your memory of last year's HCI module…
  - In fact, origins of HRI lie in HCI
  - Particularly in terms of development and evaluation methodologies

Aspects of HRI

- Psychology
- Human Factors

- Sociology / Social Science
- Human-Computer Interaction

- Computer Science
- Mechatronics



From (Baxter et al, 2016)

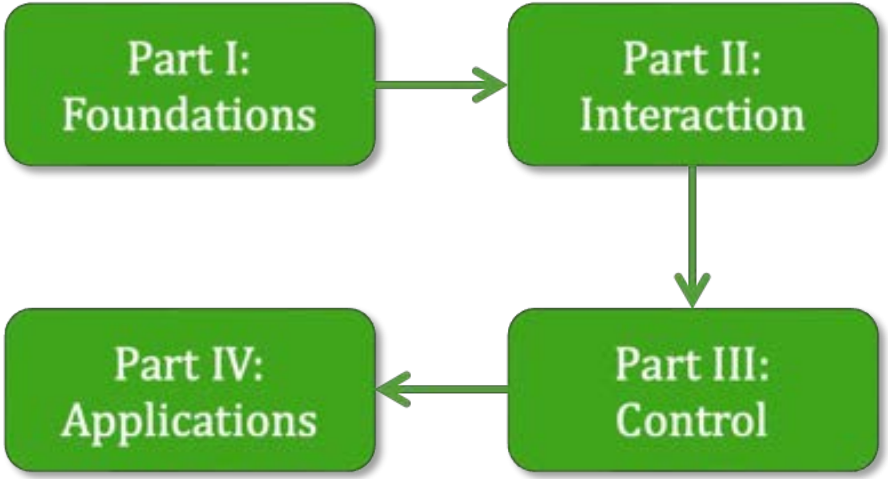
Humans and Robots

Humans

- Interaction partner
- Social agent
- Target of research
- Source of knowledge
- Recipient of help
- Caregiver
- Companion
- …

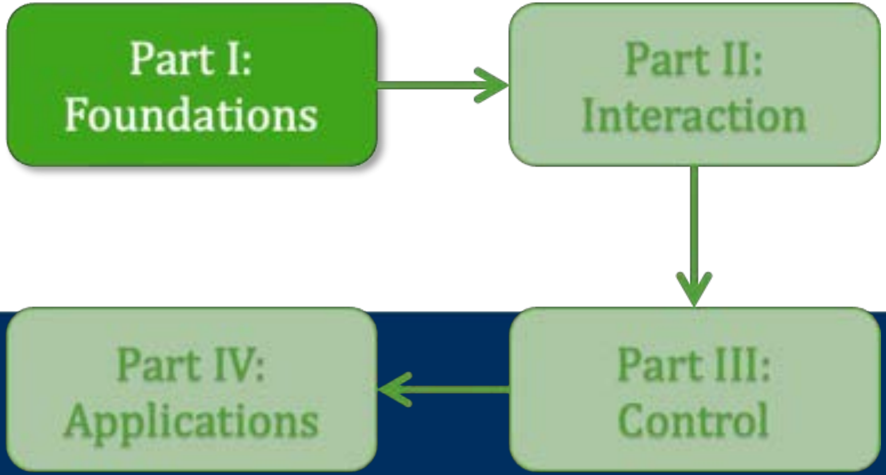
Robots

- Interaction partner
- Social agent?
- Target of development
- Source of knowledge
- Recipient of help
- Caregiver
- Companion
- …


Part I:
Foundations
Part II:
Interaction
Part III:
Control
Part IV:
Applications

Part I: Foundations

Assumptions and characteristics

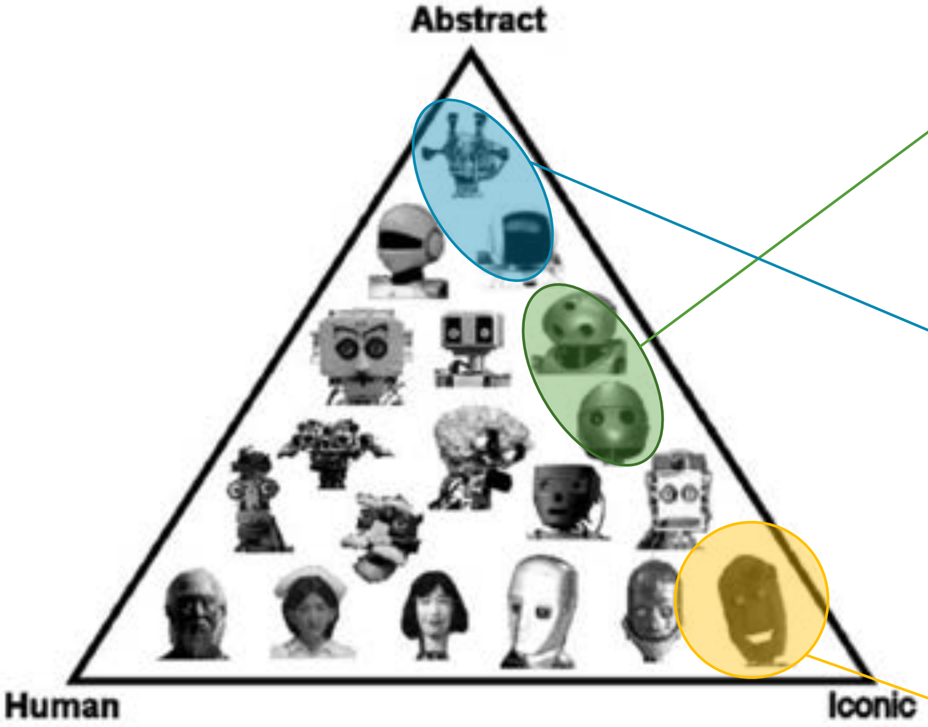


Anthropomorphism

- 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

Anthropomorphism: Robot Faces



Anthropomorphism: Behaviour

Anthropomorphism: Behaviour

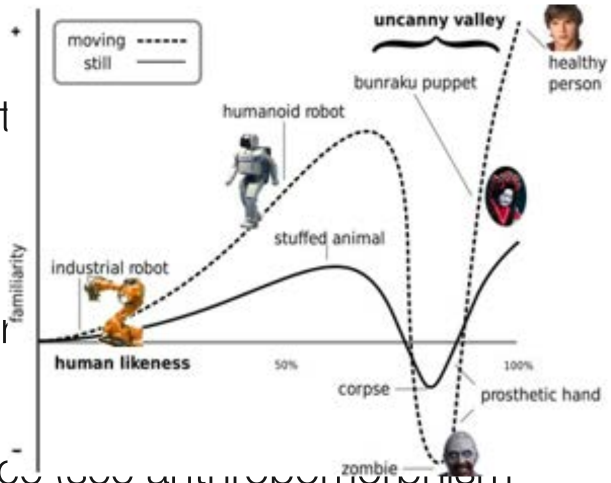
- While working on your assignment, how many of you:
  - Swore at your robot?
  - Complimented your robot?
  - Referred to your robot as he/she?
  - Gave your robot a name?

Anthropomorphism: Behaviour

Anthropomorphism: Robots

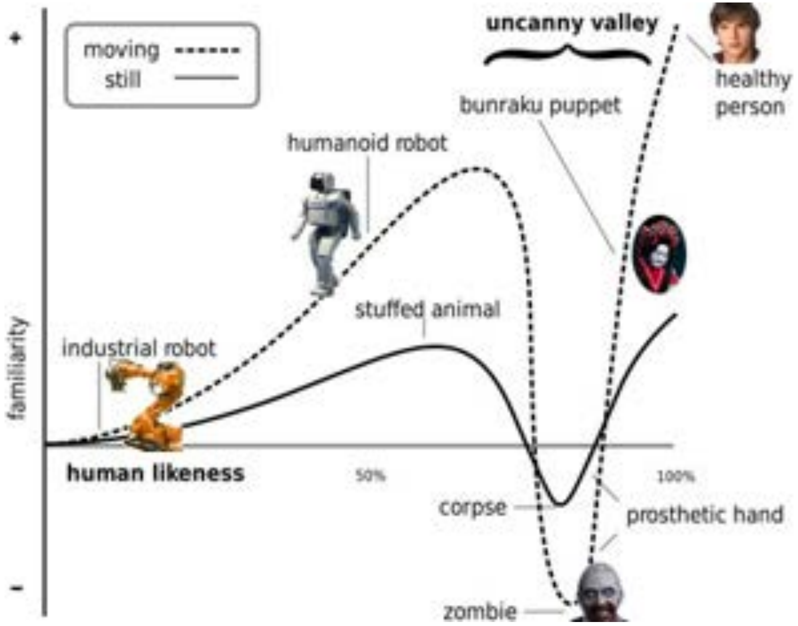
The Uncanny Valley

- The “Uncanny Valley”
  - Increasing human likeness increases familiarity…
  - …however, at a certain point, a sharp drop in familiarit
- Refer to (Mathur et al, 2016) for an overview
- Reasons for this not fully understood
- One possible explanation: a conflict between cues (Moor
  - Hence why the “moving” curve more pronounced
  - Greater mismatch between movement and appearanc
    notes above)



http://spectrum.ieee.org/automaton/robotics/humanoids/040210-who-is-afraid-of-the-uncanny-valley

The Uncanny Valley

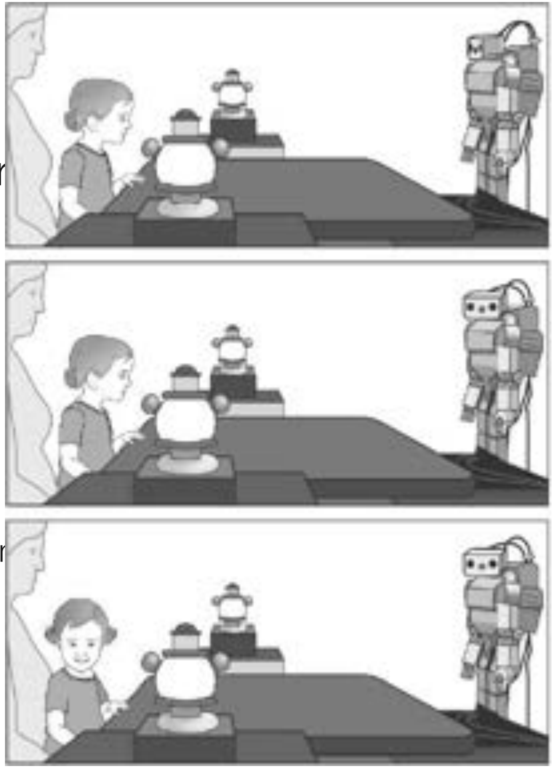


http://spectrum.ieee.org/automaton/robotics/humanoids/040210-who-is-afraid-of-the-uncanny-valley

Attribution of Agency to Robots

- Particularly if certain characteristics are present, people will natur
inanimate object
  - Though this illusion can be broken in the interaction

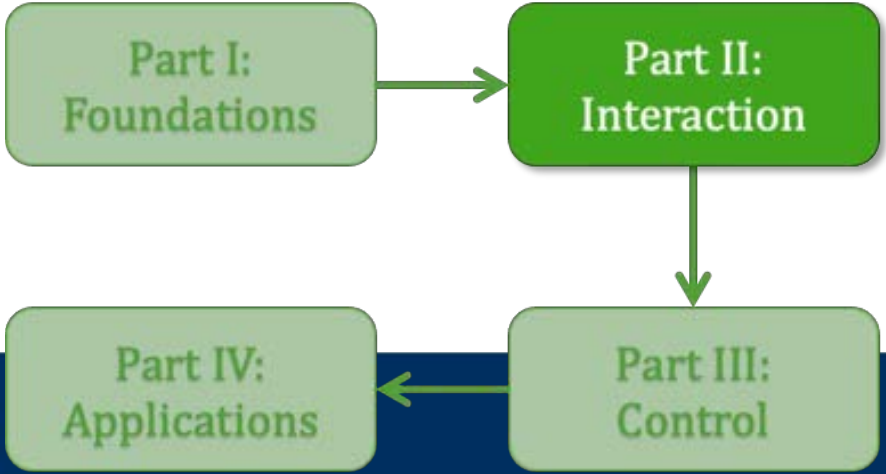
- This is (at least partly) learned from experience
- Seen in children with a robot for example
  - Meltzhoff et al (2010)
  - Children watch adult interact with robot (joint attention)
  - These children then more likely to follow gaze when they interaction with robot ther



From Meltzhoff et al (2010)

Part II: Interaction

Types and contexts



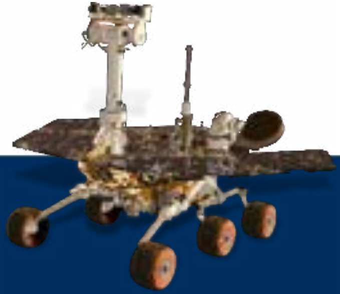
Fundamentally two modes…

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, su



Two types of Interaction

Explicit

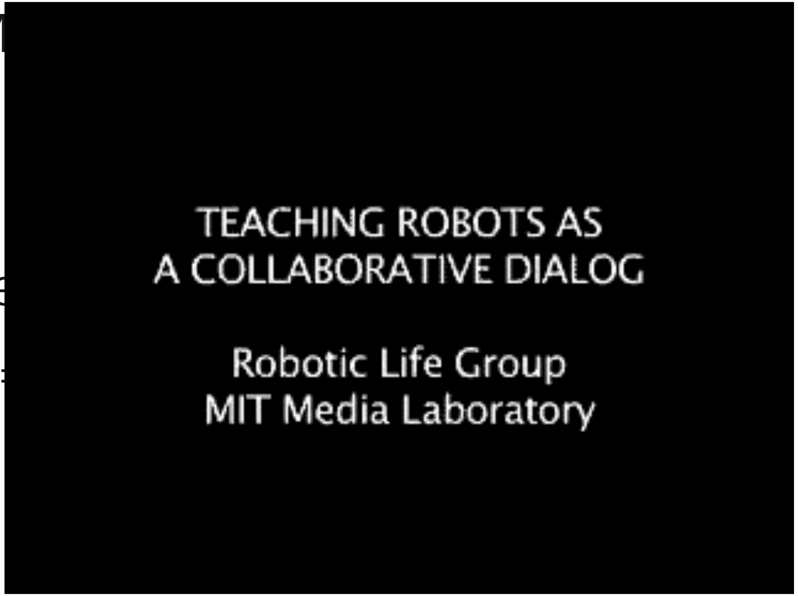
- An action performed with the purpose of eliciting a reaction
- Dialogue: e.g. asking a question
- Manipulation: e.g. handing over an object, or pointing

Implicit

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 Thomasz (M
- Explicit interaction
  - Pointing from human
  - Gaze gestures from robot to indicate re
- See: https://www.youtube.com/watch?v=



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

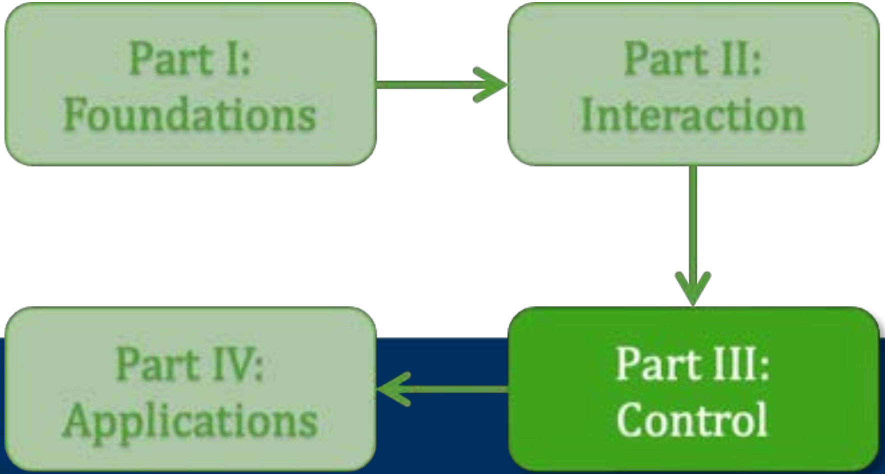
- FROG project: robot museum guid[redacted]
- Video from HRI 2014 conference
- Implicit interaction in terms of navi[redacted] and other aspects of behaviour (m[redacted]

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



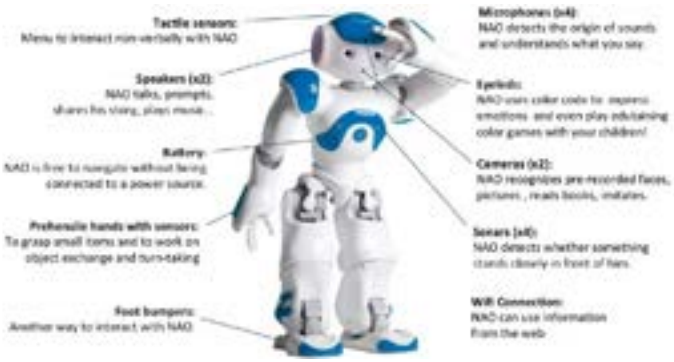
Control for Autonomous Robots

This should be familiar from last week!

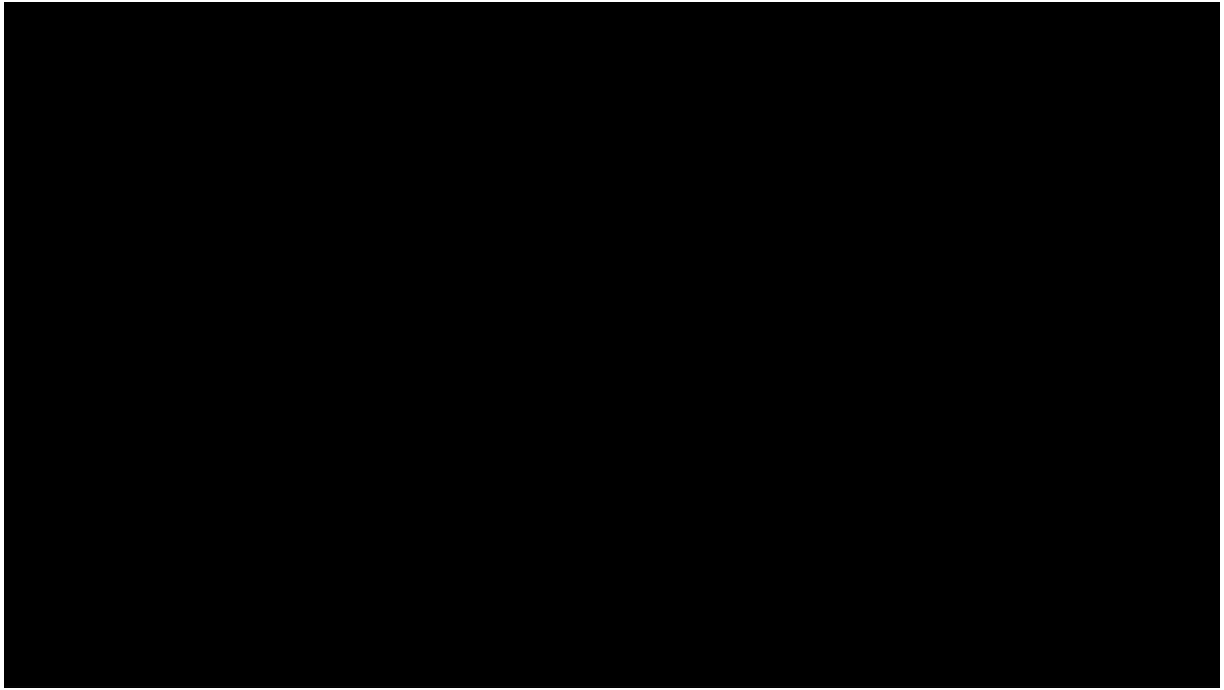
- Sensing
- Decision making
- Acting

Sensing

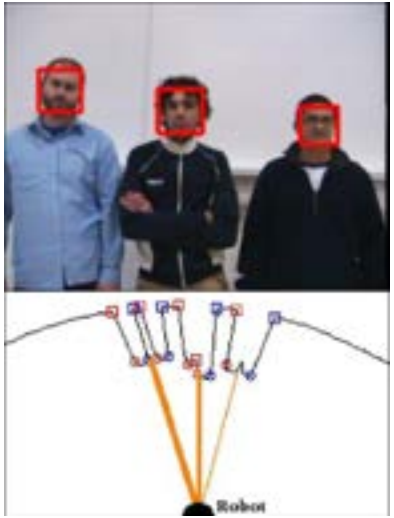
- Recall the sensing you did in your assignment:
  - Array of sensors (depth, vision, bump, …)
  - This environment was static (nothing moving except the robot)
- Taking into account humans adds significant difficulty
  - Not just because people move…
  - … also unpredictability, occlusions, etc
  - And: the meaning of underlying expressions, gestures, etc
- People are not good at being reliable…
- Sensors suffer from noise…



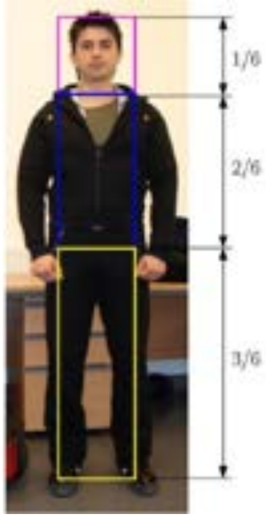
Sensing is difficult…



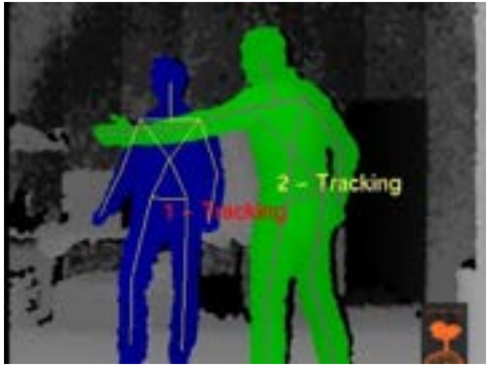
Human Detection



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!"

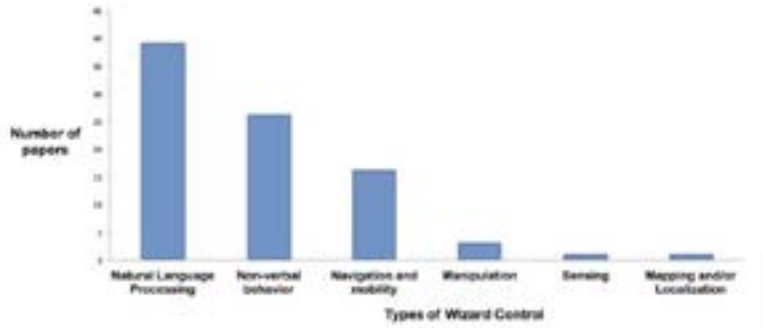
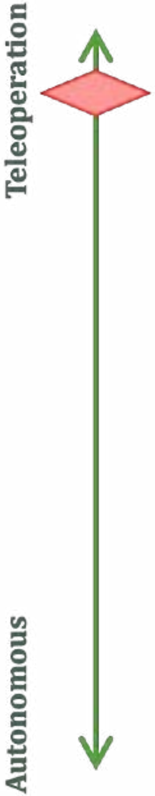


Figure 4. Chart depicting the types of Wizard control employed in the included papers. Some papers described using more than one type of control.

UNIVERSITY OF LINCOLN

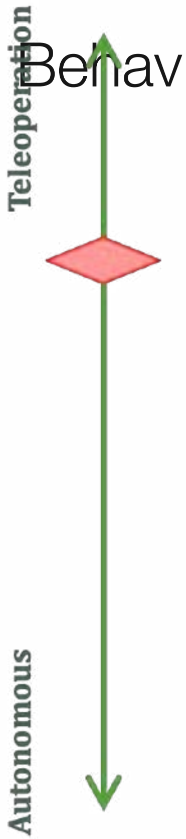
L-CAS

1. Wizard of Oz



- 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

2. Scripted Robot Behaviour

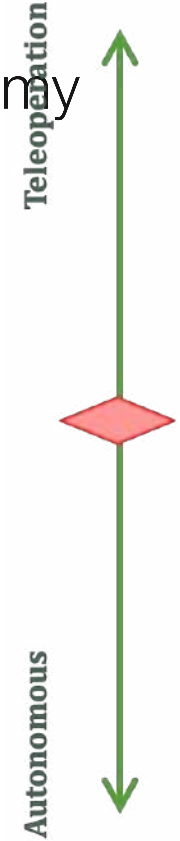


- Mediated Teleoperation
- Reducing the workload on a Wizard
- E.g. studying human behaviour when interacting with a robot

UNIVERSITY OF LINCOLN

L-CAS

3. Partial Autonomy

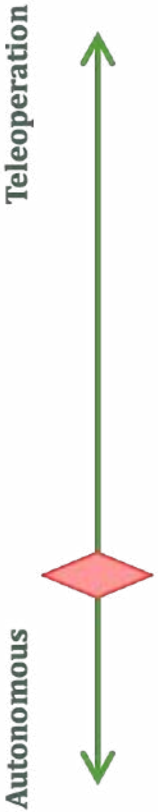


- Testing subsystems of a robot control system
  - Components that are ready can be run autonomously
  - Those that are not can be wizarded

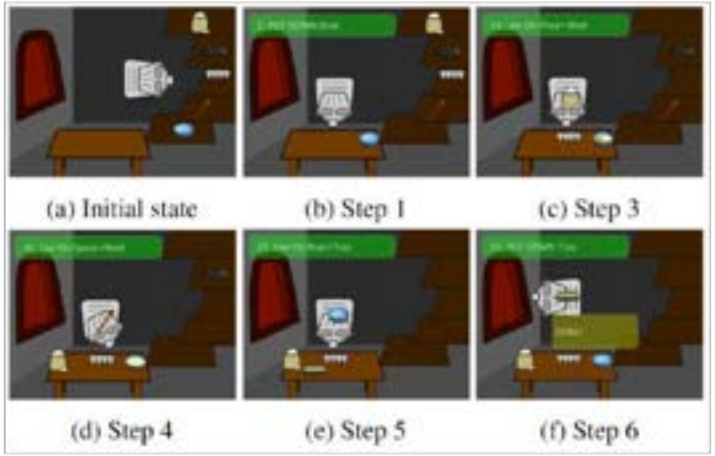
- Use of shared autonomy
  - Hand-off between robot and human team-mates
  - E.g. in disaster scenarios: partially autonomous search prior to rescue

http://www.tradr-project.eu/

4. Supervised Autonomous Behaviour

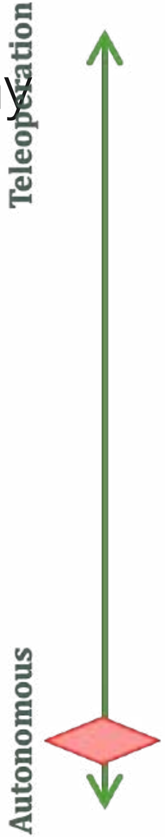


- Attempting to overcome noisy sensors
    - If very unreliable, then a helping hand may be needed
- Guidance for a learning robot system
    - Helping it to learn


(a) Initial state (b) Step 1 (c) Step 3
(d) Step 4 (e) Step 5 (f) Step 6

Senft, Baxter, et al (2017)

5. Full Autonomy



- 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...

Contrasting WoZ and Autonomy

Social Sciences Point of View

WoZ

- Pros:
  - Remove uncertainty
  - Focus on evaluation of interaction rather that 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

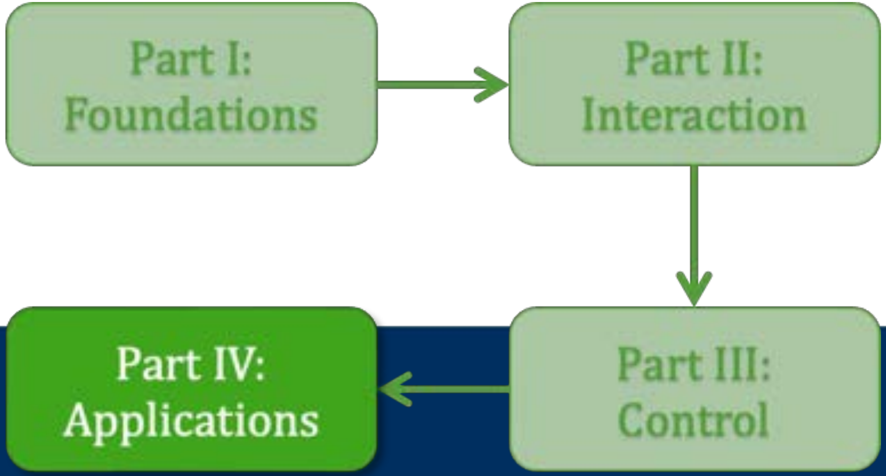
Level of autonomy!?

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...

Part IV: Applications

Robot roles and ethics



Robot Roles: Learning Peer

- Robot as a learning partner for a child: h
  abstract/document/6249477/
- Robot and child co-located, facing each
  can both interact with the touchscreen
- Robot plays the role of a peer: not perfe
  mistakes, and adapts its behaviour

Robot Roles: Therapy Aid

- Probo robot: a green elephant-like cudd
- Used in autism therapy for children, und
- Used in Robot-Enhanced Therapy, see h

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**

“Engagement” with Robots

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

↓

2. Maintain Engagement

↓

3. End 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

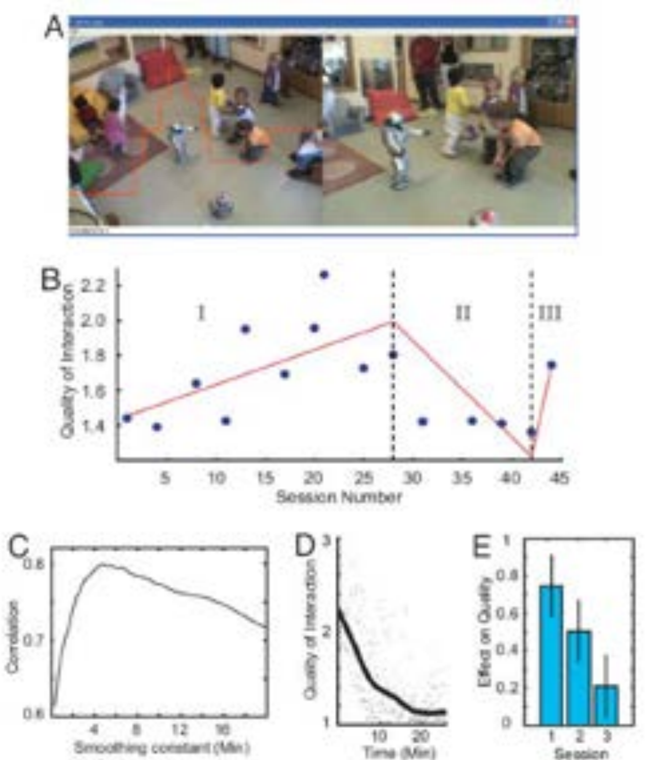
- Personal related
  - Illusion of agency has gone
  - Boredom!

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?

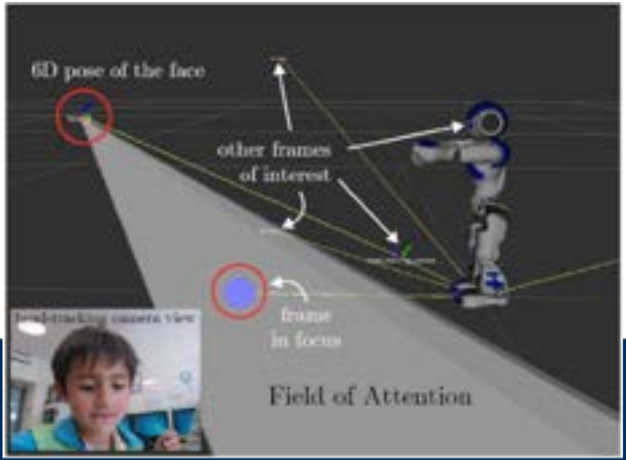
- This is a difficult task!

- 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'



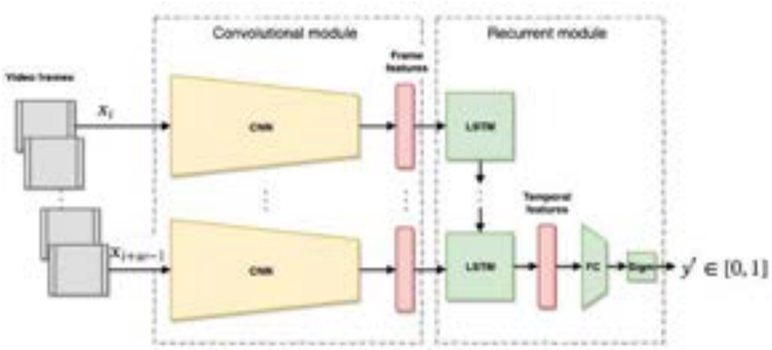
How to detect Engagement?

- 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

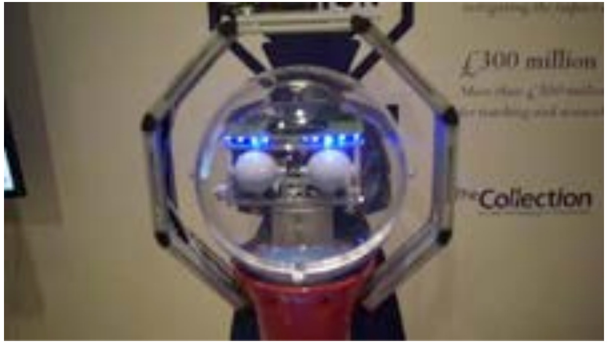


How to detect Engagement?

- 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



Title Text

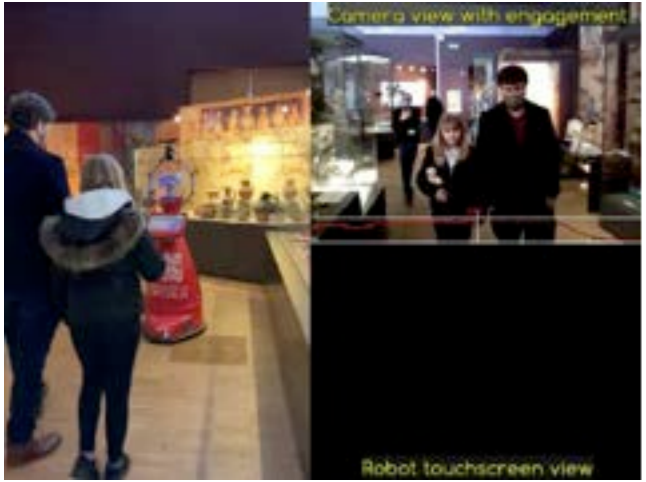


UNIVERSITY OF
LINCOLN

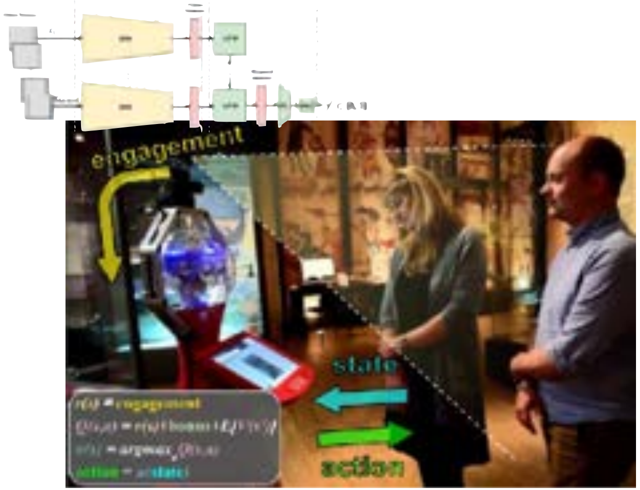
L-CAS

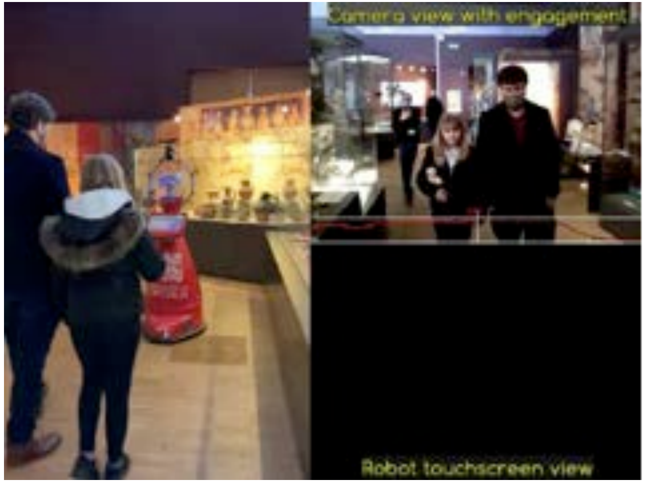
Reinforcement learning in the public domain?

- Lindsey is a tour guide robot in The Collection Archeological Museum
- *Can we learn better tours through long-term interaction with the public?*



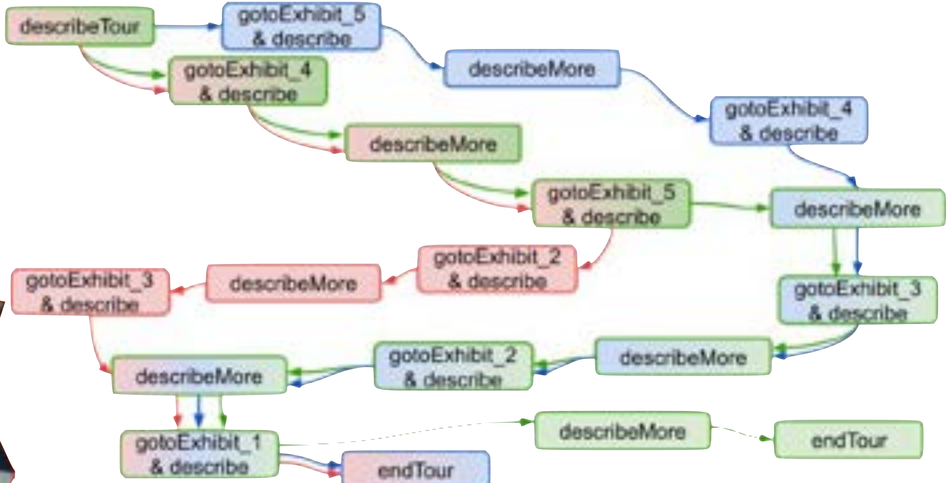
Reinforcement learning in the public domain?





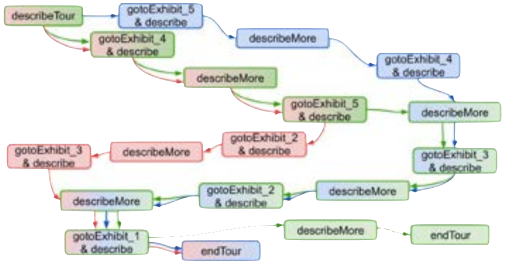
Del Duchetto, F., Baxter, P., & Hanheide, M. (2020). Are you still with me? Continuous engagement assessment from a robot's point of view. *Frontiers in Robotics and AI*, 116.

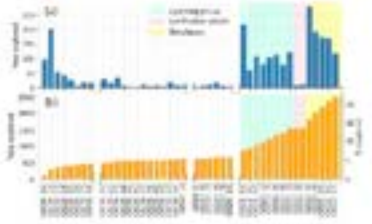
Tour structure depending on current engagement and state

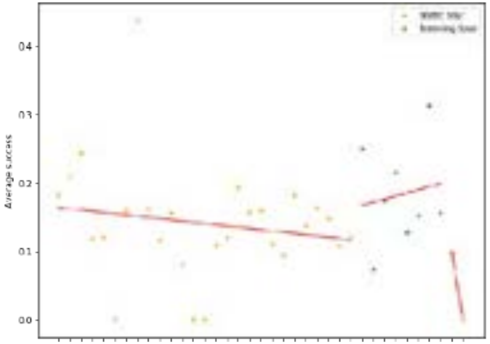


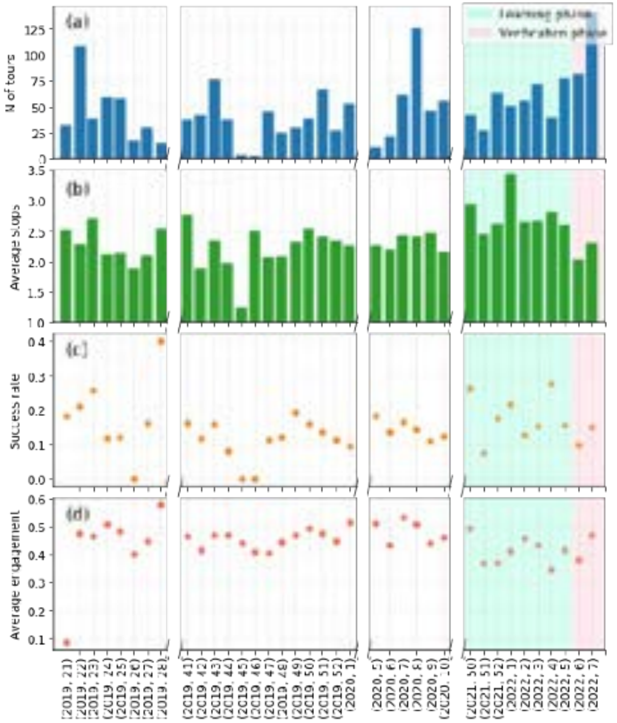
Different actions chosen by the learned policy for the tour art at different levels of engagement. Engagement values are red for LOW, blue for MEDIUM and green for HIGH

Getting better on the job!









[to be properly published soon]

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

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.

CMP3101 AMR – WEEK 12

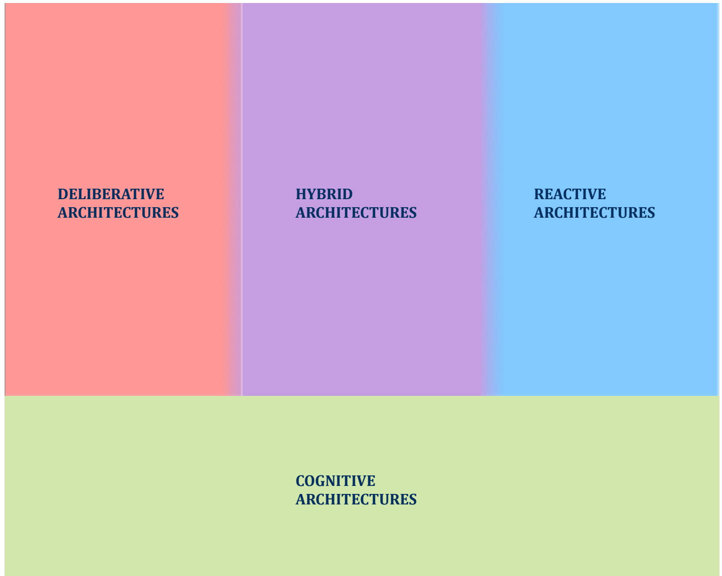
HUMAN-ROBOT INTERACTION

Prof Marc Hanheide (with gratitude to Dr Paul Baxter)

mhanheide@lincoln.ac.uk

Last Week…

- Control Architectures for Autonor
- 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/

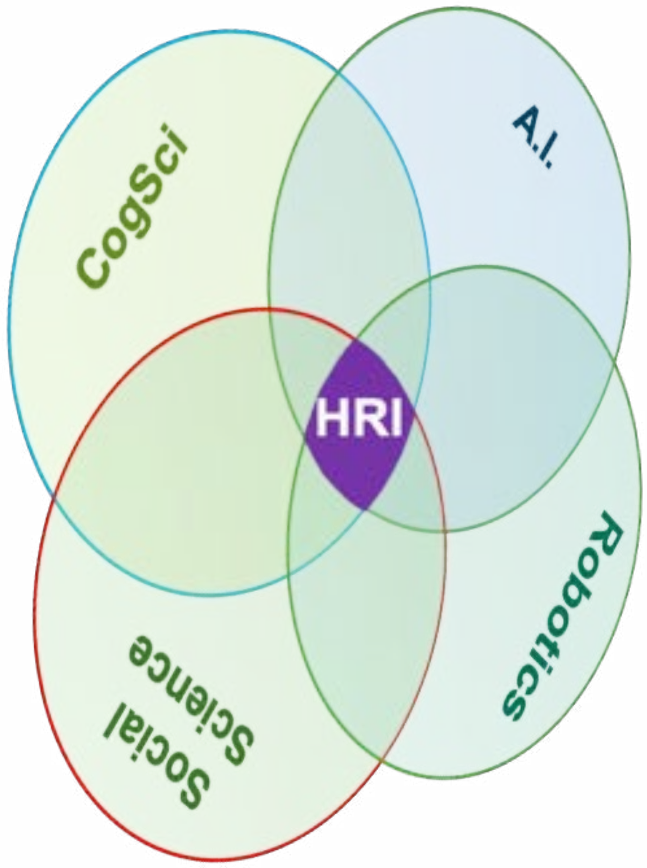
- Some overlaps with Human-Computer Interaction, so refresh your memory of last year's HCI module…
  - In fact, origins of HRI lie in HCI
  - Particularly in terms of development and evaluation methodologies

Aspects of HRI

- Psychology
- Human Factors

- Sociology / Social Science
- Human-Computer Interaction

- Computer Science
- Mechatronics



From (Baxter et al, 2016)

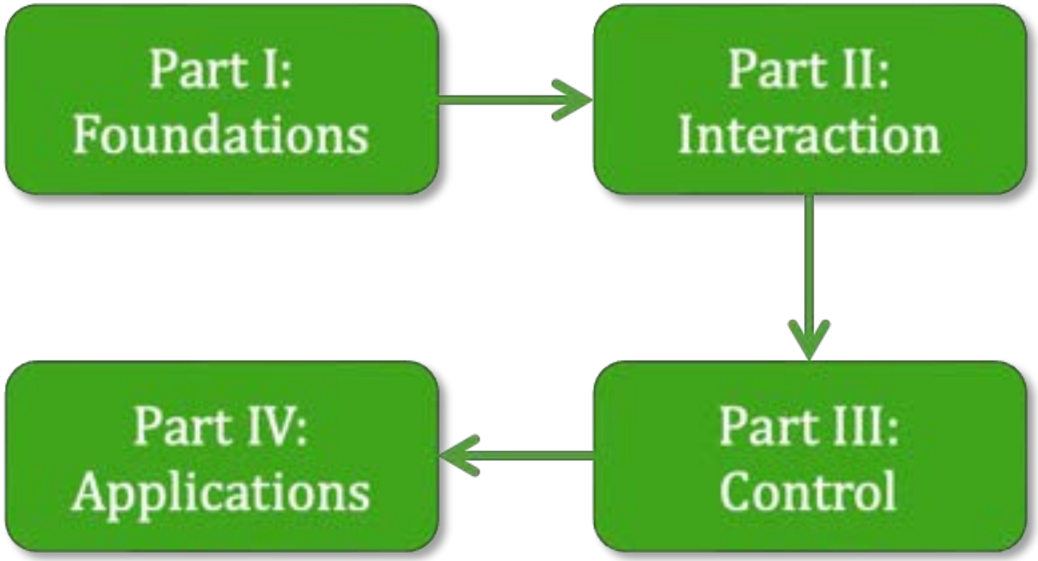
Humans and Robots

Humans

- Interaction partner
- Social agent
- Target of research
- Source of knowledge
- Recipient of help
- Caregiver
- Companion
- …

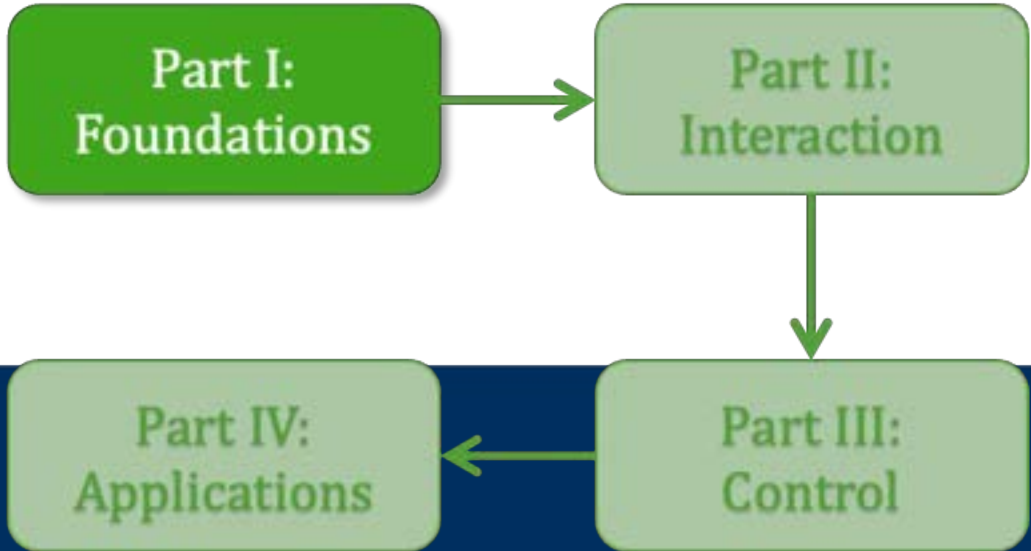
Robots

- Interaction partner
- Social agent?
- Target of development
- Source of knowledge
- Recipient of help
- Caregiver
- Companion
- …


Part I: Foundations
Part II: Interaction
Part III: Control
Part IV: Applications

Part I: Foundations

Assumptions and characteristics

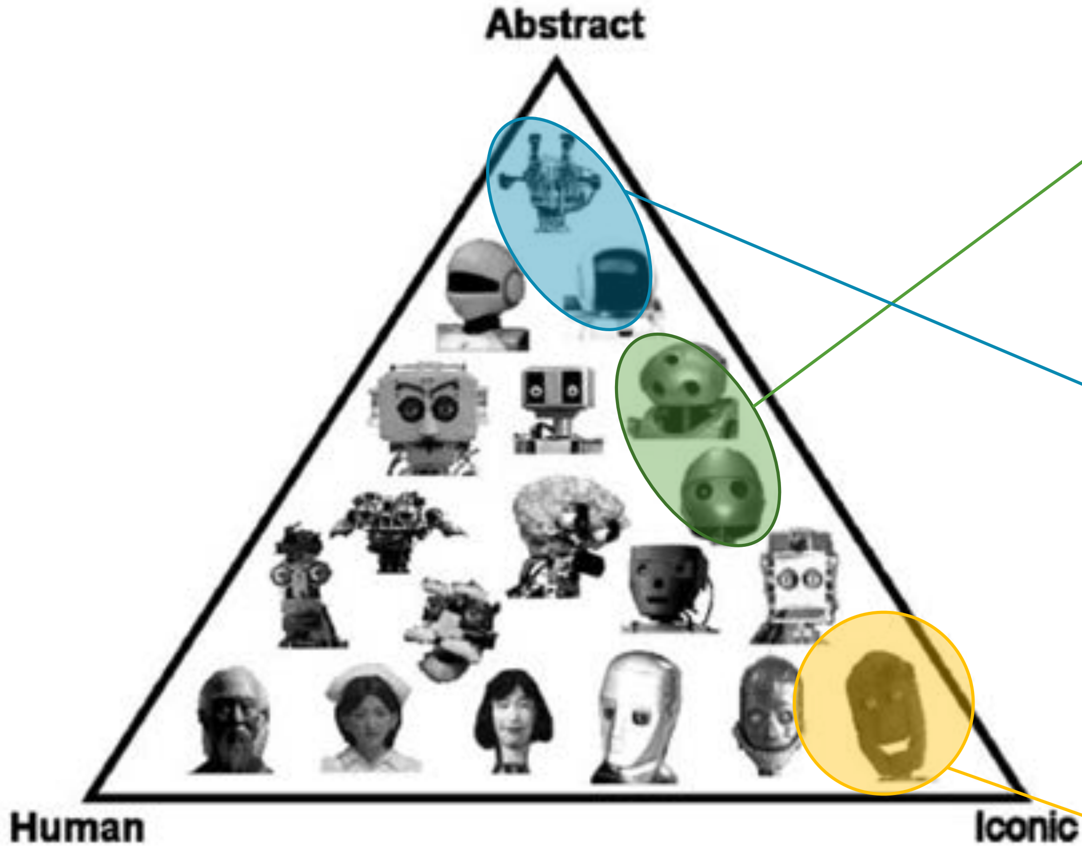


Anthropomorphism

- 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

Anthropomorphism: Robot Faces



Anthropomorphism: Behaviour

Anthropomorphism: Behaviour

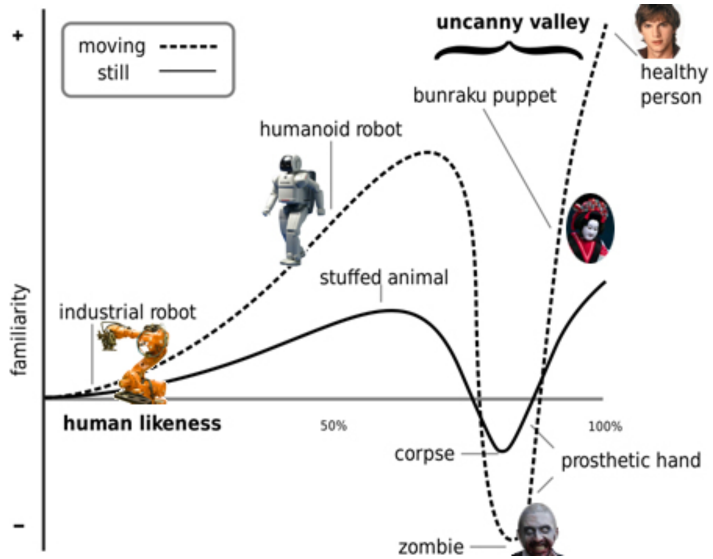
- While working on your assignment, how many of you:
  - Swore at your robot?
  - Complimented your robot?
  - Referred to your robot as he/she?
  - Gave your robot a name?

Anthropomorphism: Behaviour

Anthropomorphism: Robots

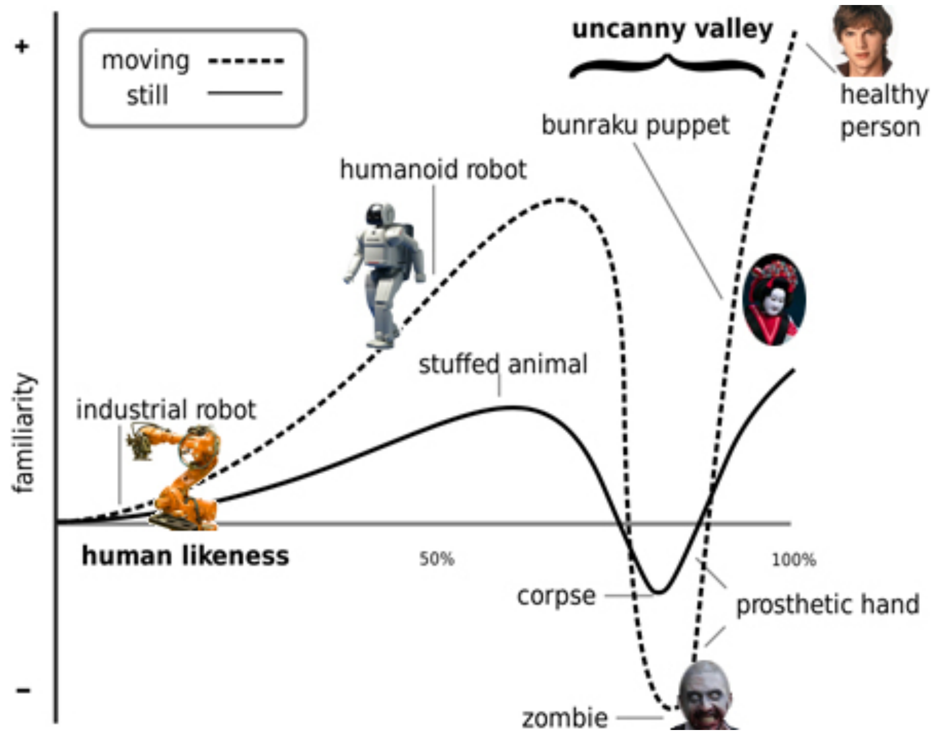
The Uncanny Valley

- 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
- 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)



http://spectrum.ieee.org/automaton/robotics/humanoids/040210-who-is-afraid-of-the-uncanny-valley

The Uncanny Valley

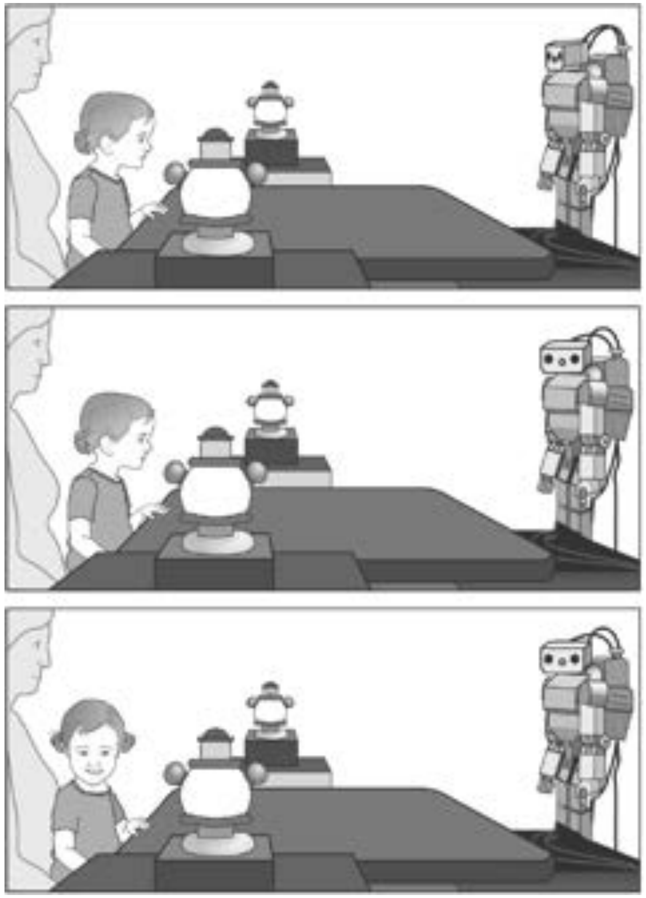


http://spectrum.ieee.org/automaton/robotics/humanoids/040210-who-is-afraid-of-the-uncanny-valley

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

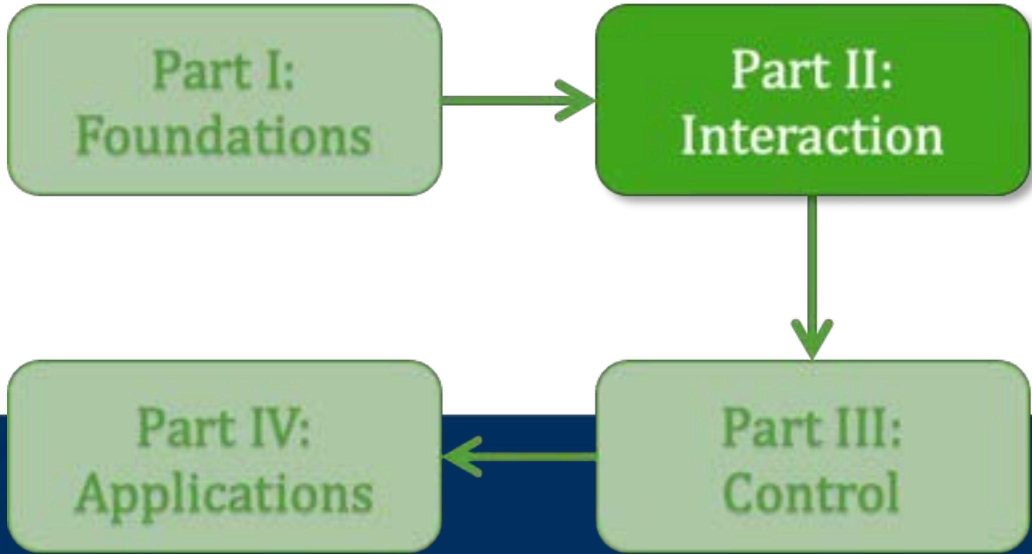
- This is (at least partly) learned from experience
- Seen in children with a robot for example
  - Meltzhoff et al (2010)
  - Children watch adult interact with robot (joint attention)
  - These children then more likely to follow gaze when they interaction with robot themselves



From Meltzhoff et al (2010)

Part II: Interaction

Types and contexts



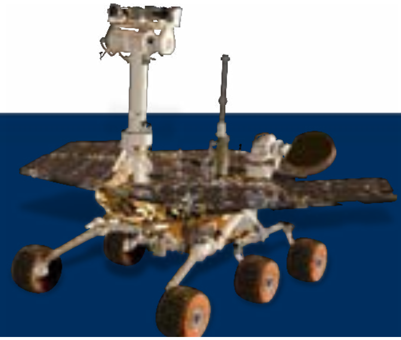
Fundamentally two modes…

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

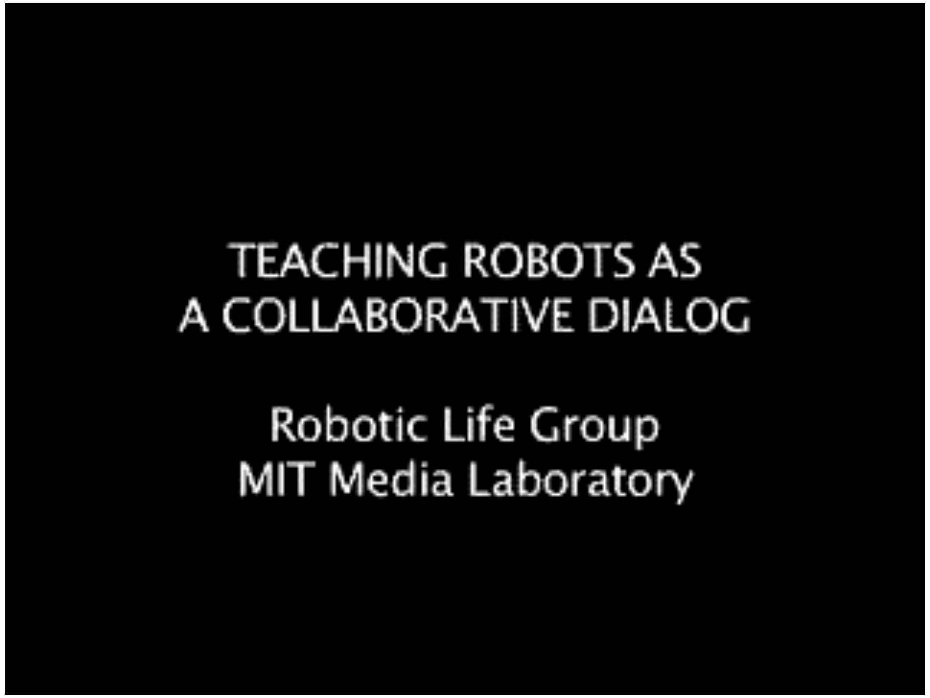
- An action performed with the purpose of eliciting a reaction
- Dialogue: e.g. asking a question
- Manipulation: e.g. handing over an object, or pointing

Implicit

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 Thomasz (MIT, 2006)
- Explicit interaction
  - Pointing from human
  - Gaze gestures from robot to indicate ready for next instruction
- See: https://www.youtube.com/watch?v=GHIlFrL7dKM



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

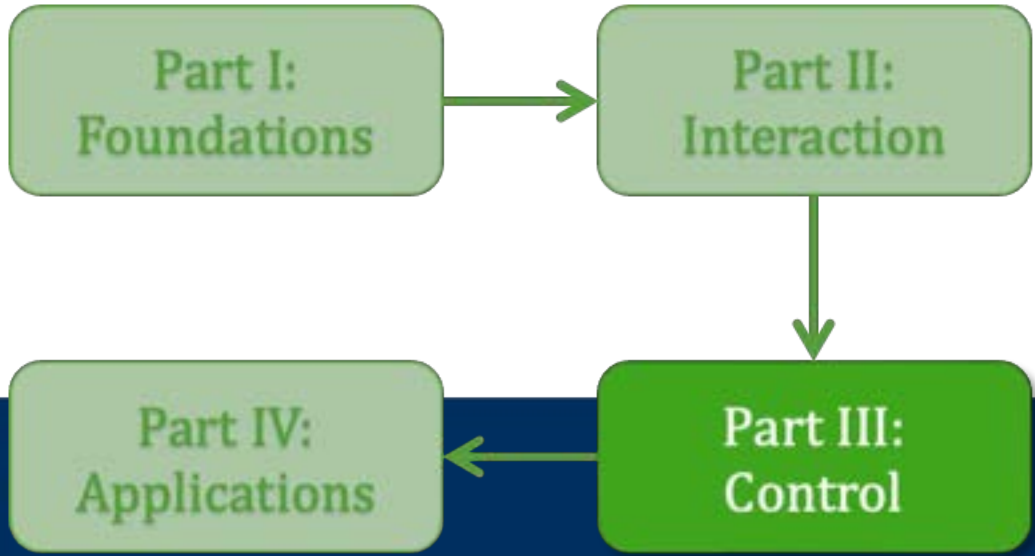
- FROG project: robot museum guide
- Video from HRI 2014 conference
- Implicit interaction in terms of navigation, avoidance, spatial positioning, and other aspects of behaviour (more on this next week!)

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

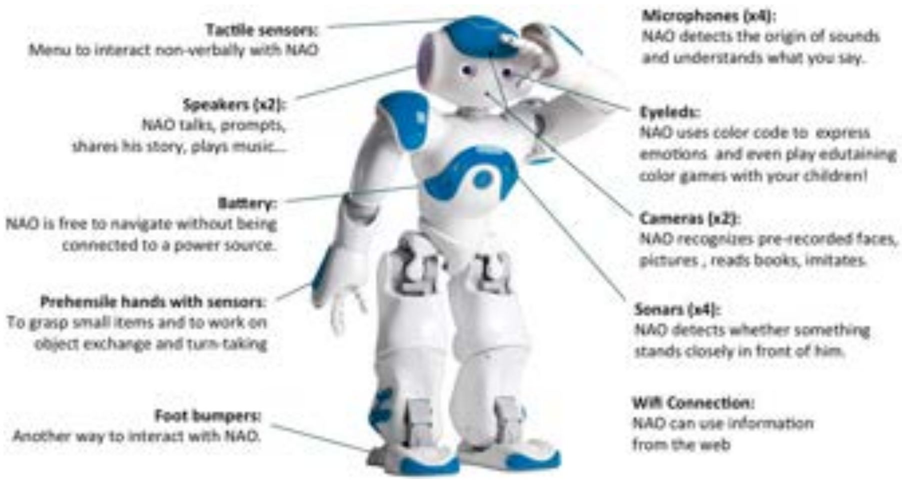
Sensing

- Recall the sensing you did in your assignment:
  - Array of sensors (depth, vision, bump, …)
  - 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

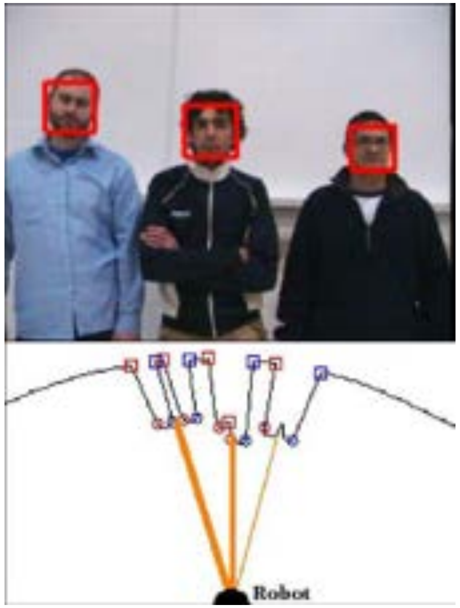
- People are not good at being reliable…

- Sensors suffer from noise…

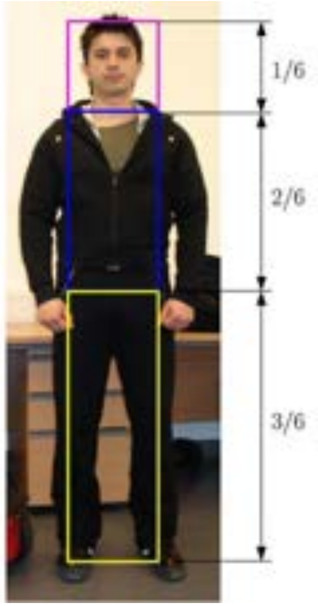


Sensing is difficult…

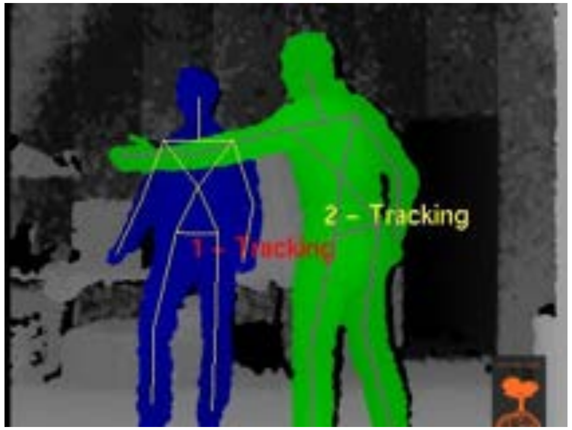
Human Detection



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!"

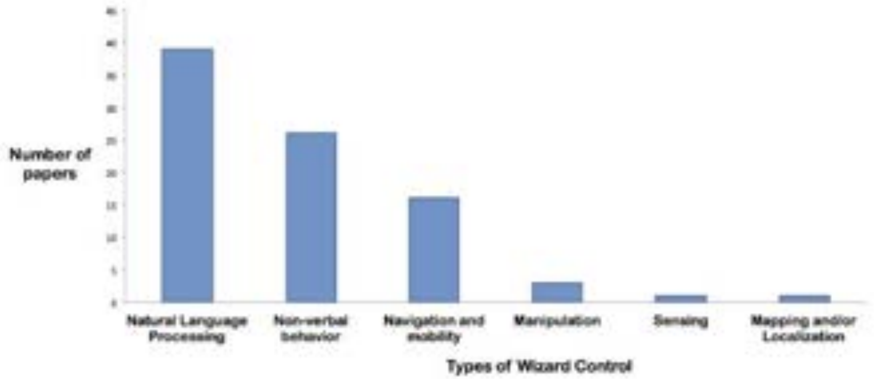
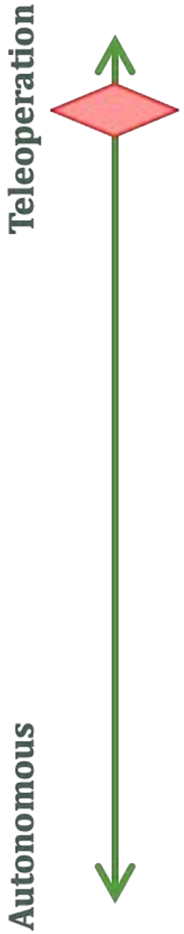


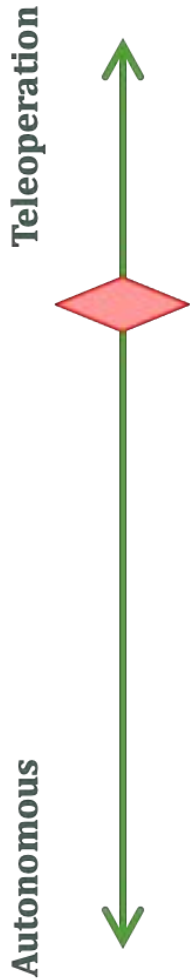
Figure 4. Chart depicting the types of Wizard control employed in the included papers. Some papers described using more than one type of control.

1. Wizard of Oz



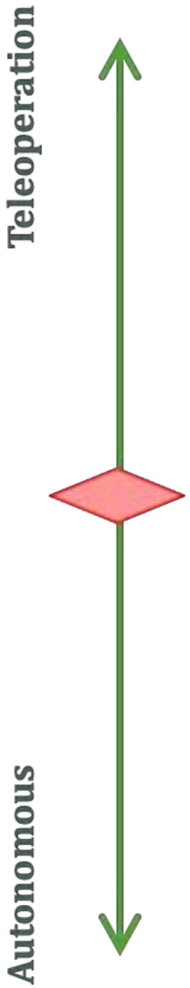
- Teleoperation
- Giving the impression of autonomy
- The Wizard is hidden from view, or not obviously associated with the control of the robot
  - Either way, the participant is unaware of the remote control.
- Complete WoZ entails full remote control of all aspects of behaviour

2. Scripted Robot Behaviour



- Mediated Teleoperation
- Reducing the workload on a Wizard
- E.g. studying human behaviour when interacting with a robot

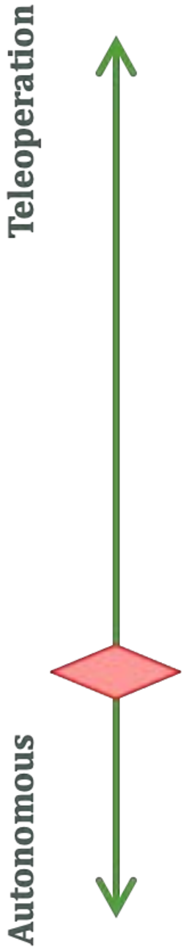
3. Partial Autonomy



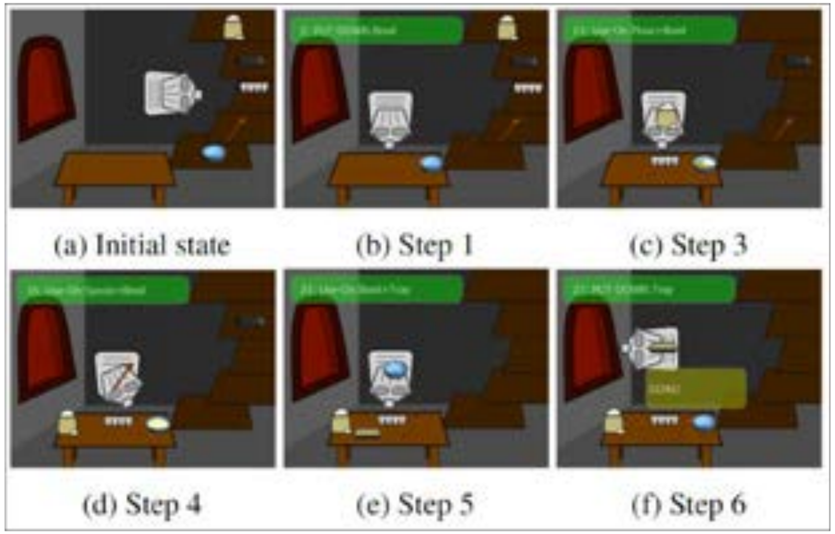
- Testing subsystems of a robot control system
  - Components that are ready can be run autonomously
  - Those that are not can be wizarded
- Use of shared autonomy
  - Hand-off between robot and human team-mates
  - E.g. in disaster scenarios: partially autonomous search prior to rescue

http://www.tradr-project.eu/

4. Supervised Autonomous Behaviour



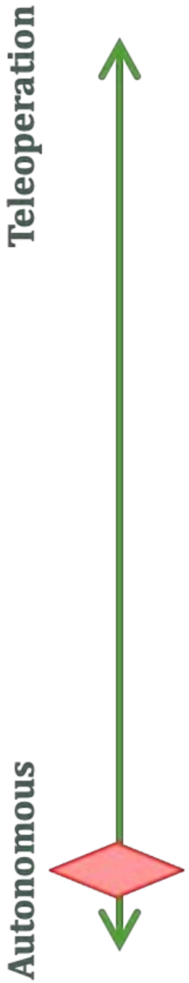
- Attempting to overcome noisy sensors
  - If very unreliable, then a helping hand may be needed
- Guidance for a learning robot system
  - Helping it to learn



(a) Initial state (b) Step 1 (c) Step 3
(d) Step 4 (e) Step 5 (f) Step 6

Senft, Baxter, et al (2017)

5. Full Autonomy



- 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…

Contrasting WoZ and Autonomy
Social Sciences Point of View

WoZ

- Pros:
  - Remove uncertainty
  - Focus on evaluation of interaction rather that 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

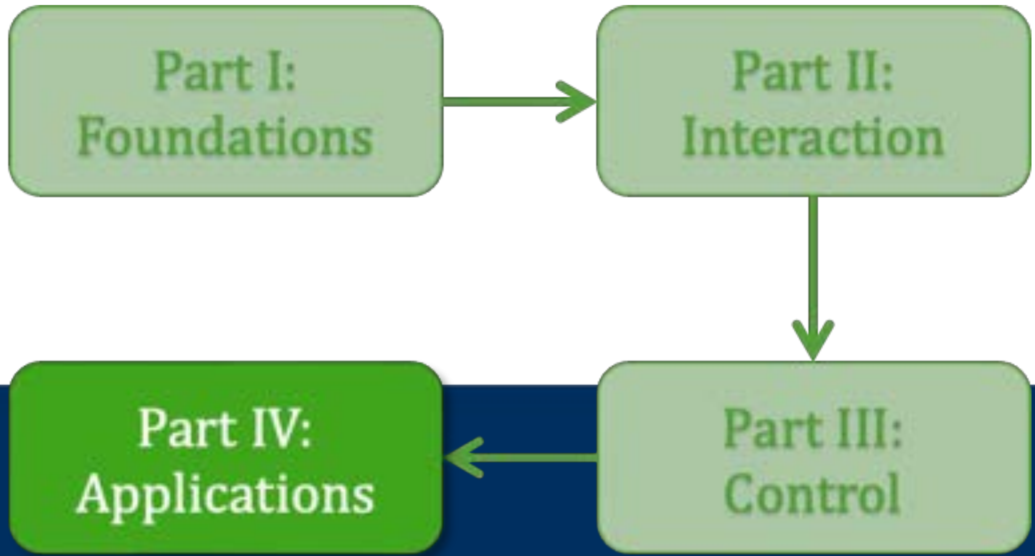
Level of autonomy!?

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…

Part IV: Applications

Robot roles and ethics



Robot Roles: Learning Peer

- 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**

“Engagement” with Robots

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
↓
2. Maintain Engagement
↓
3. End 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

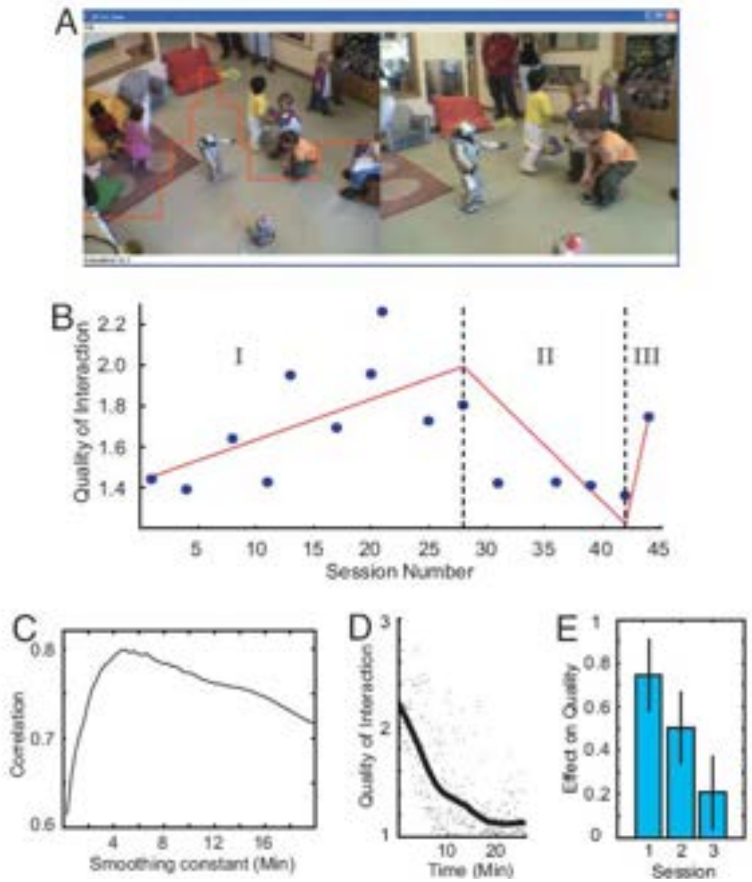
- Personal related
  - Illusion of agency has gone
  - Boredom!

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?

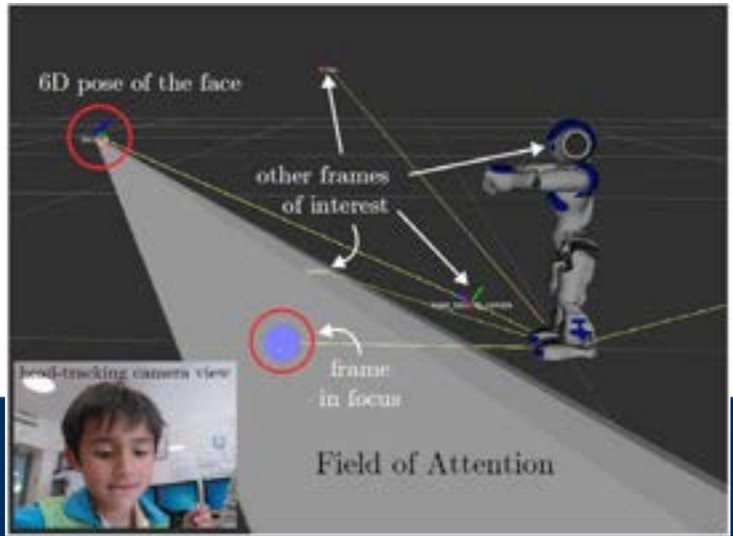
- This is a difficult task!

- 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’



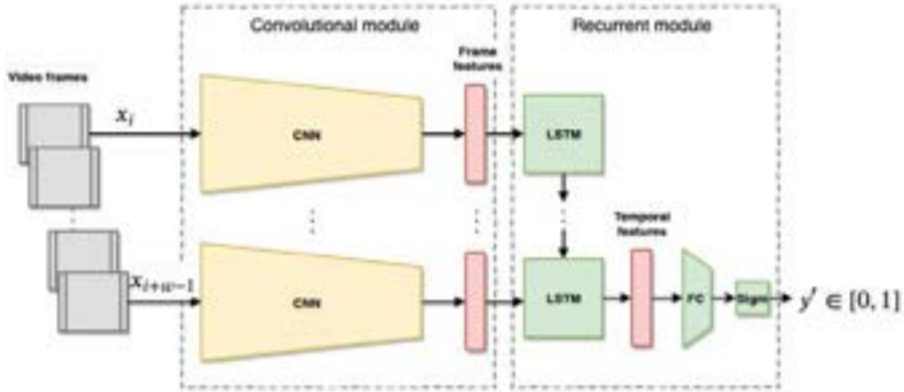
How to detect Engagement?

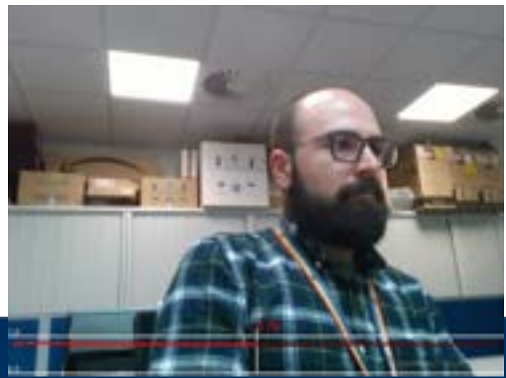
- Gaze is often used as an indicator of engagement
  - E.g. Baxter et al, 2014
- The problem is that this is often post hoc analysis
  - I.e. not in real-time, but only afterwards
- Recent developments trying to achieve this in real-time, using robot sensors:
  - E.g. the GAZR gaze estimator, which can be used for an estimation of engagement (Lemaignan et al, 2016)
  - ROS package: https://github.com/severin-lemaignan/gazr

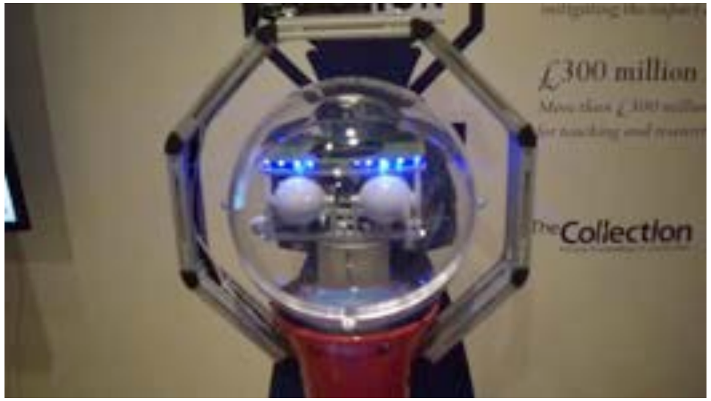


How to detect Engagement?

- 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





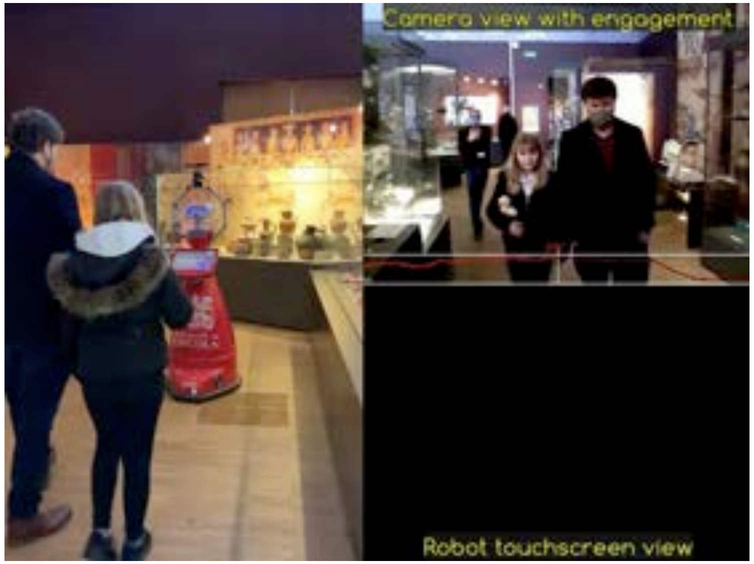


UNIVERSITY OF LINCOLN

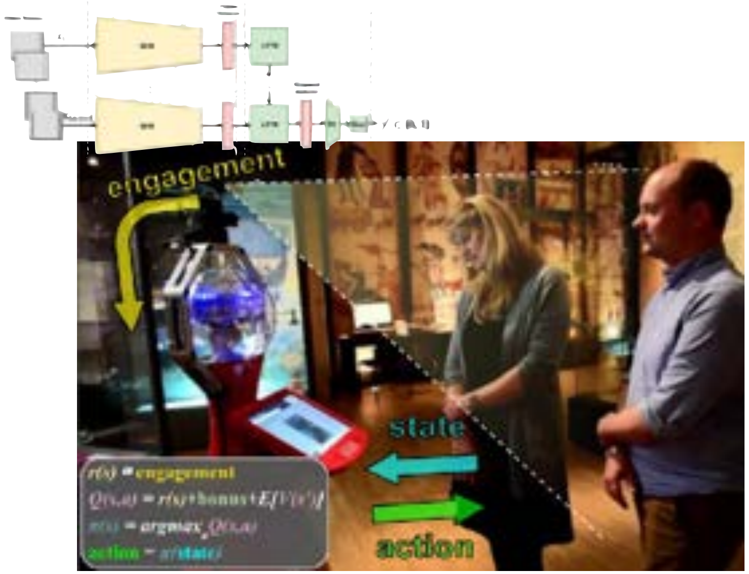
L-CAS

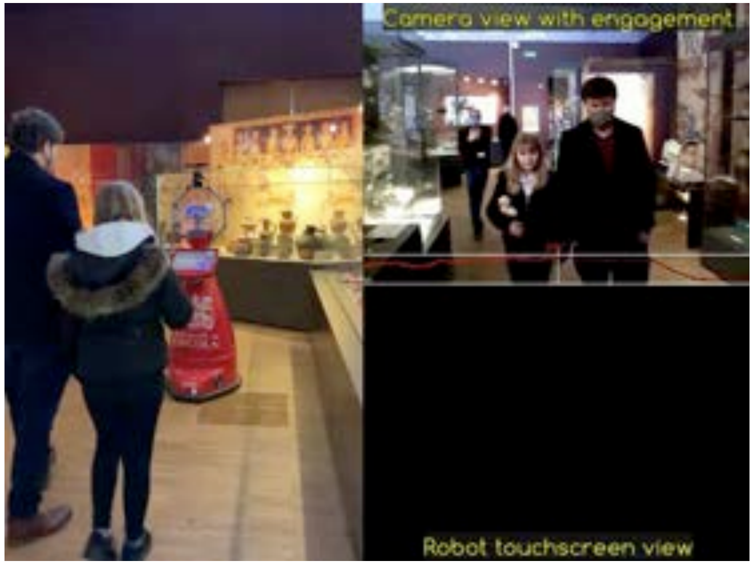
Reinforcement learning in the public domain?

- Lindsey is a tour guide robot in The Collection Archeological Museum
- *Can we learn better tours through long-term interaction with the public?*



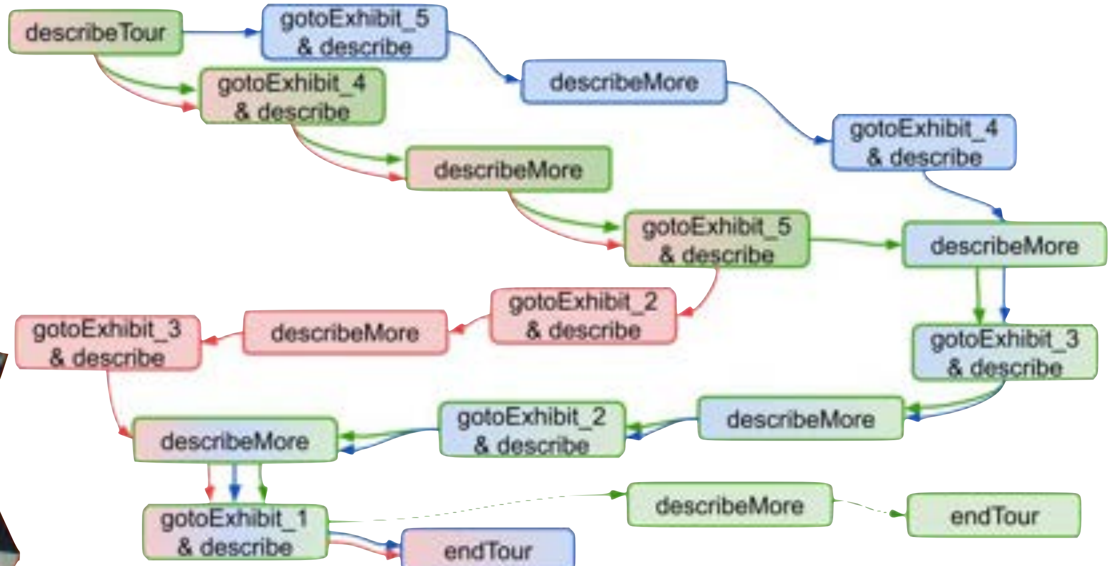
Reinforcement learning in the public domain?





Del Duchetto, F., Baxter, P., & Hanheide, M. (2020). Are you still with me? Continuous engagement assessment from a robot's point of view. *Frontiers in Robotics and AI*, 116.

Tour structure depending on current engagement and state



Different actions chosen by the learned policy for the tour art at different levels of engagement. Engagement values are red for LOW, blue for MEDIUM and green for HIGH

Getting better on the job!

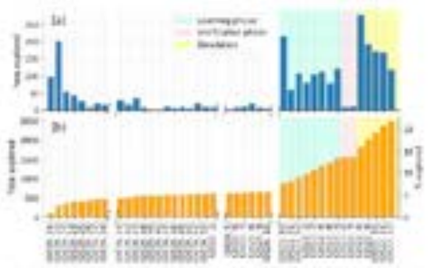
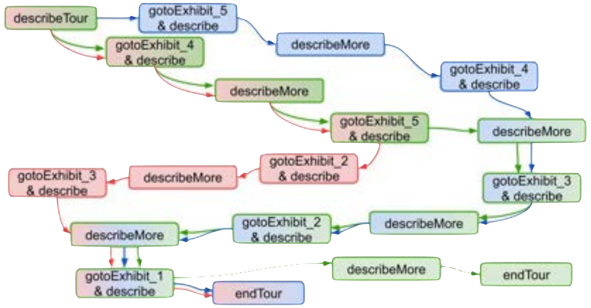
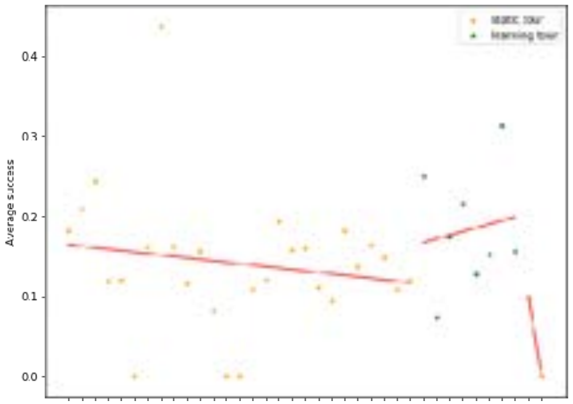
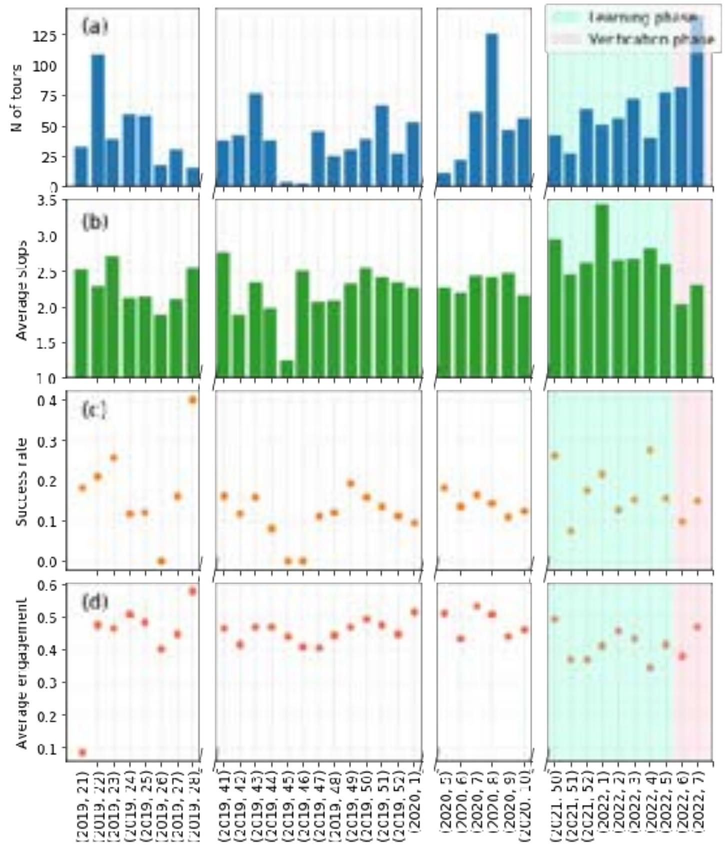


Fig. 9: Exploration of the state-action space of the policy per week. (a) Number of newly explored state-action pairs, (b) cumulative exploration.

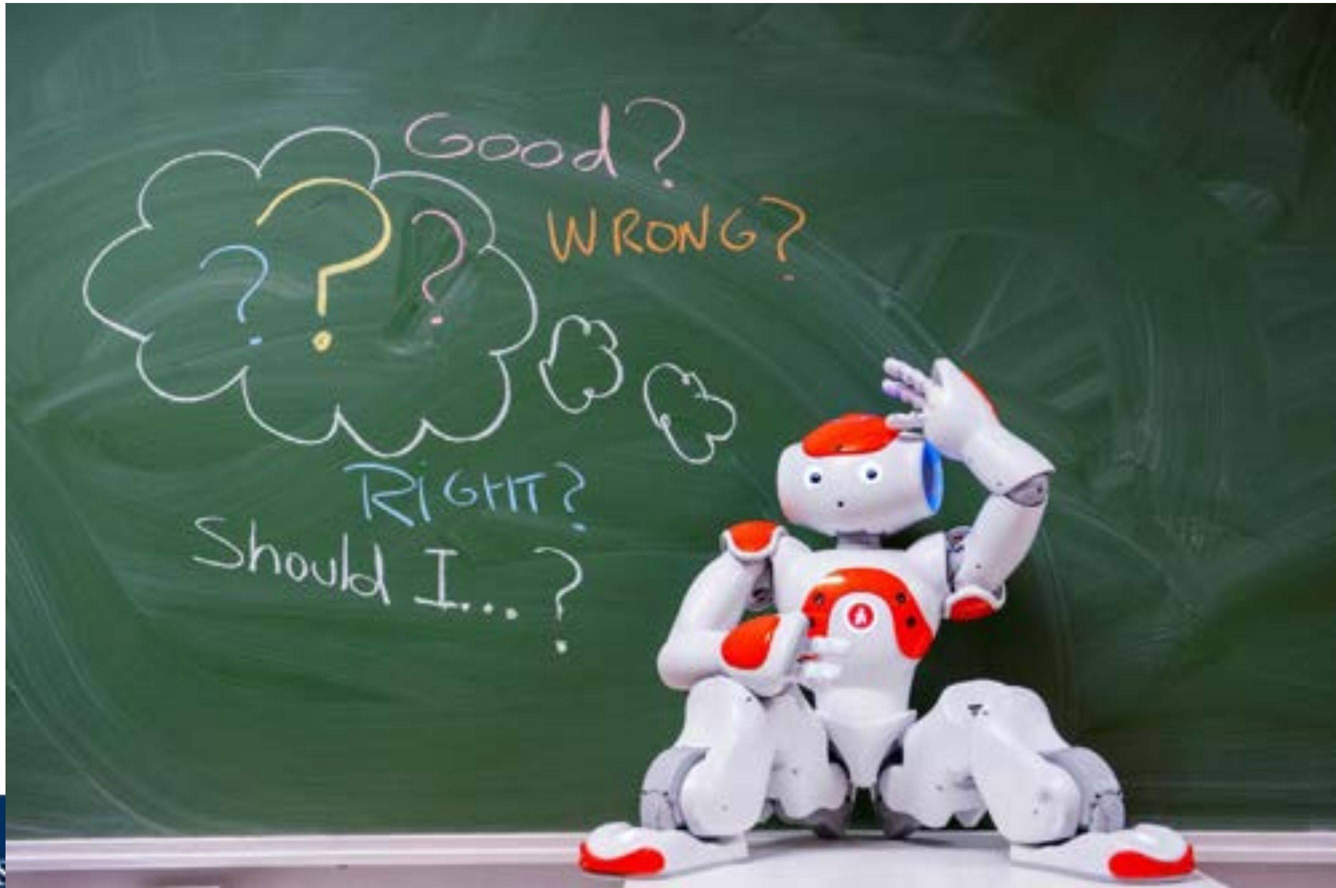






[to be properly published soon]

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

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 13

Human-Robot Interaction

part 2

Prof Marc Hanheide (and again, thanks to Dr Paul Baxter)

mhanheide@lincoln.ac.uk

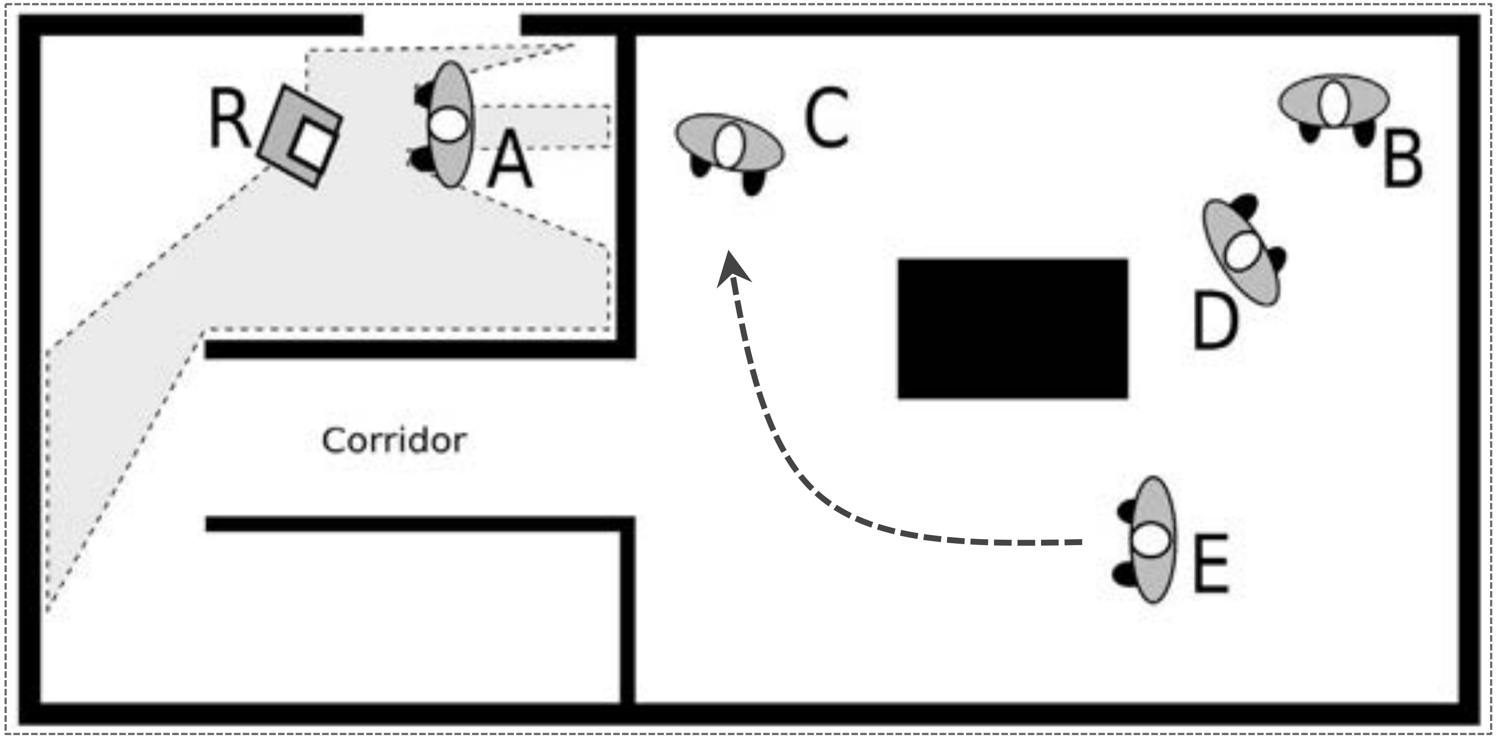
"Will I bother here? - A robot anticipating its influence on pedestrian walking comfort", HRI 2013, Tokyo, Japan
Paper: http://ieeexplore.ieee.org/abstract/document/6483597/

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

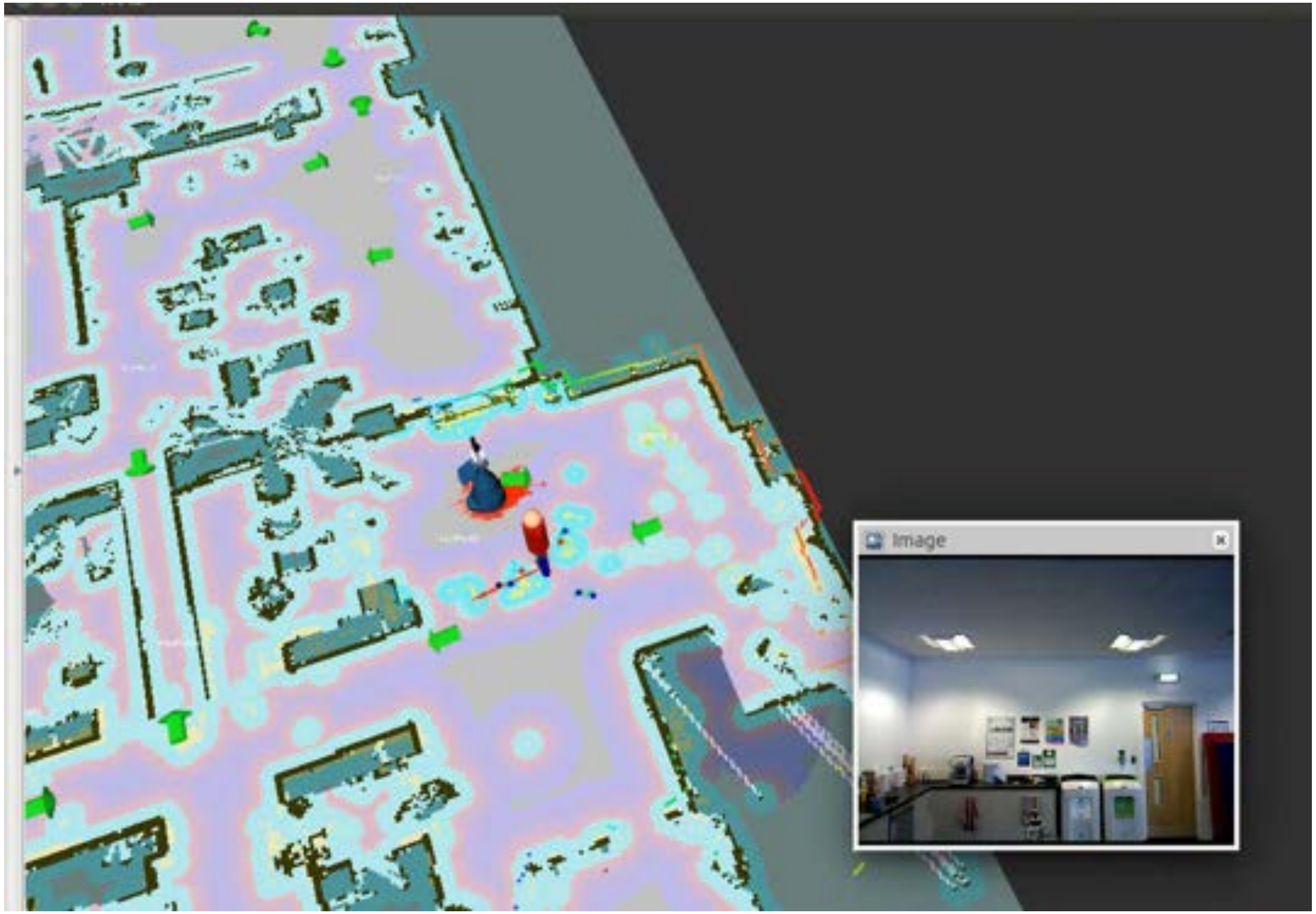
An example

From Kruse et al, 2013:



How should the robot guide person A to person B?

Humans are Awkward Obstacles…

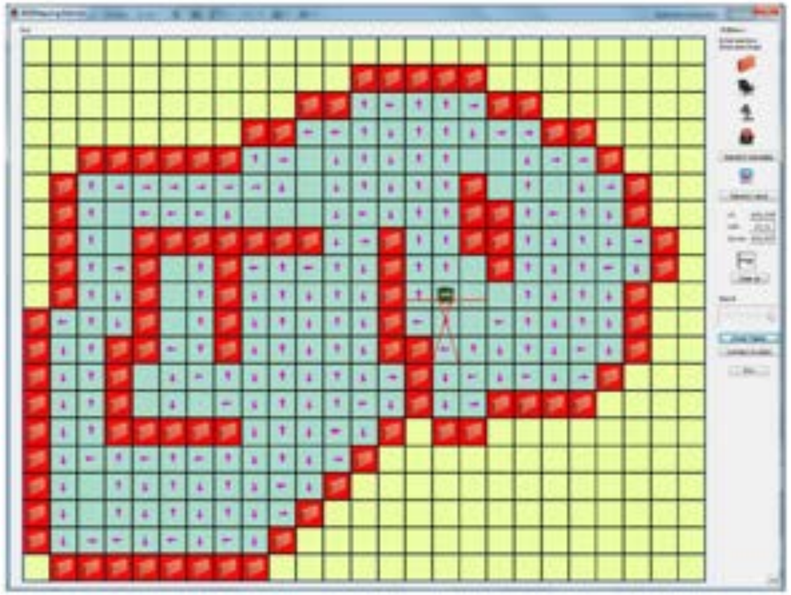


Also see this L-CAS video: https://www.youtube.com/watch?v=zdnvhQU1YNo

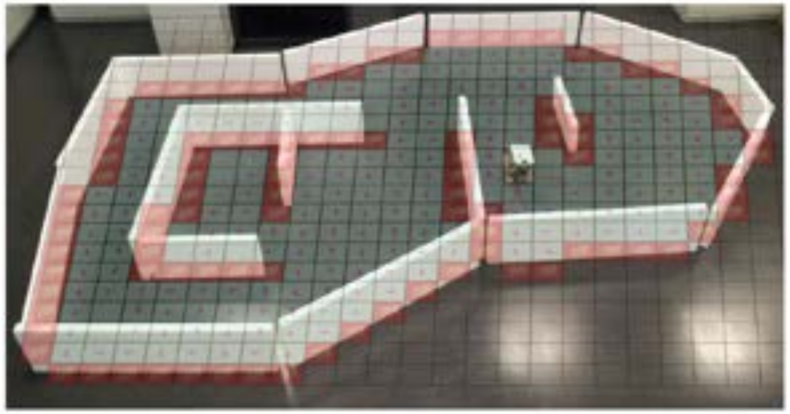
(a) Navigation and Costmaps

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)

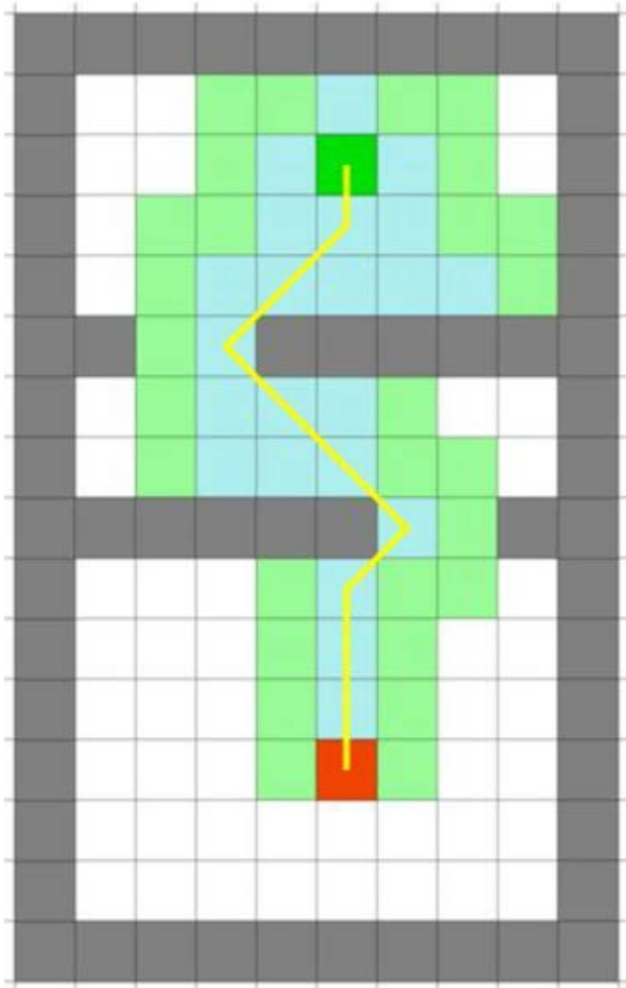


(b)

See http://qiao.github.io/PathFinding.js/visual/

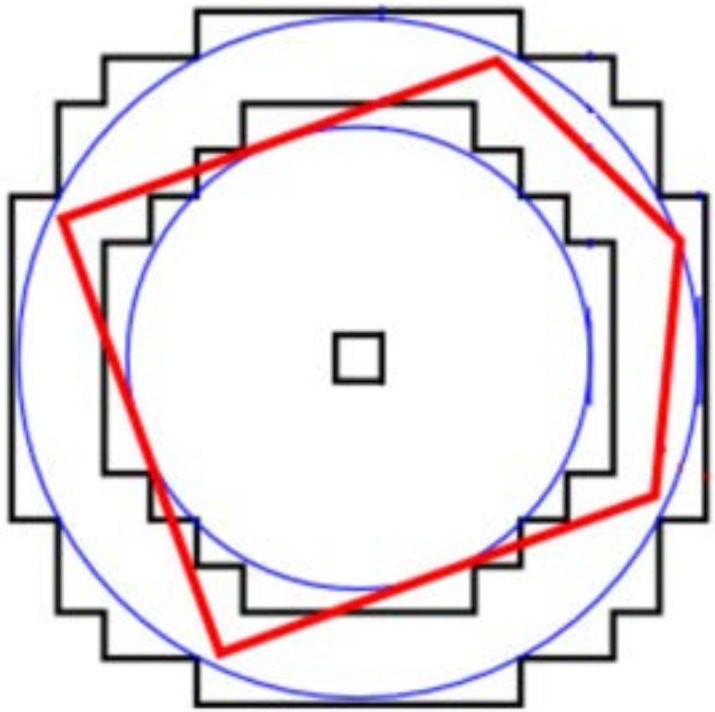
Path Planning

- 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

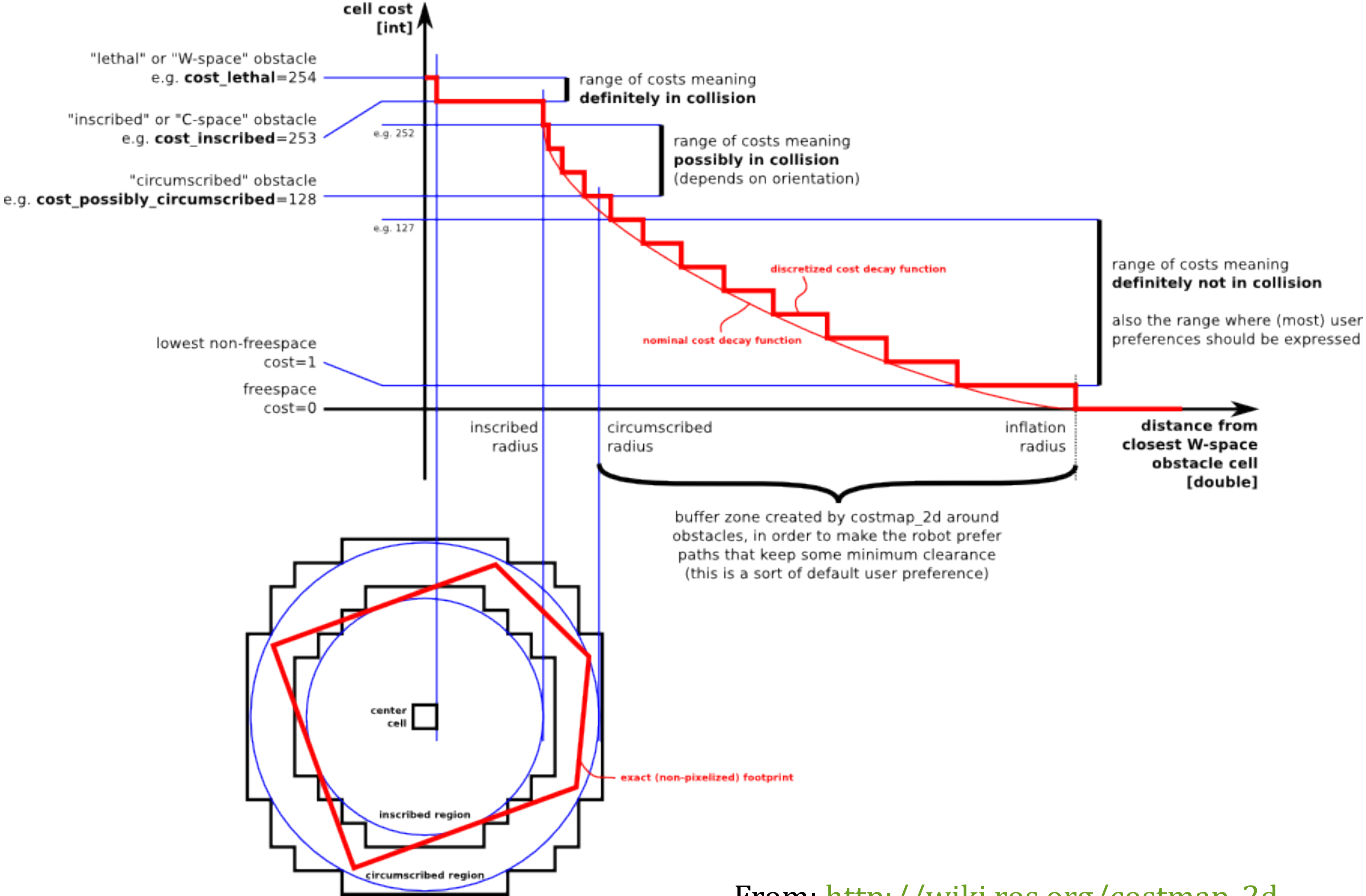


Robot Distances and Costs

- 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

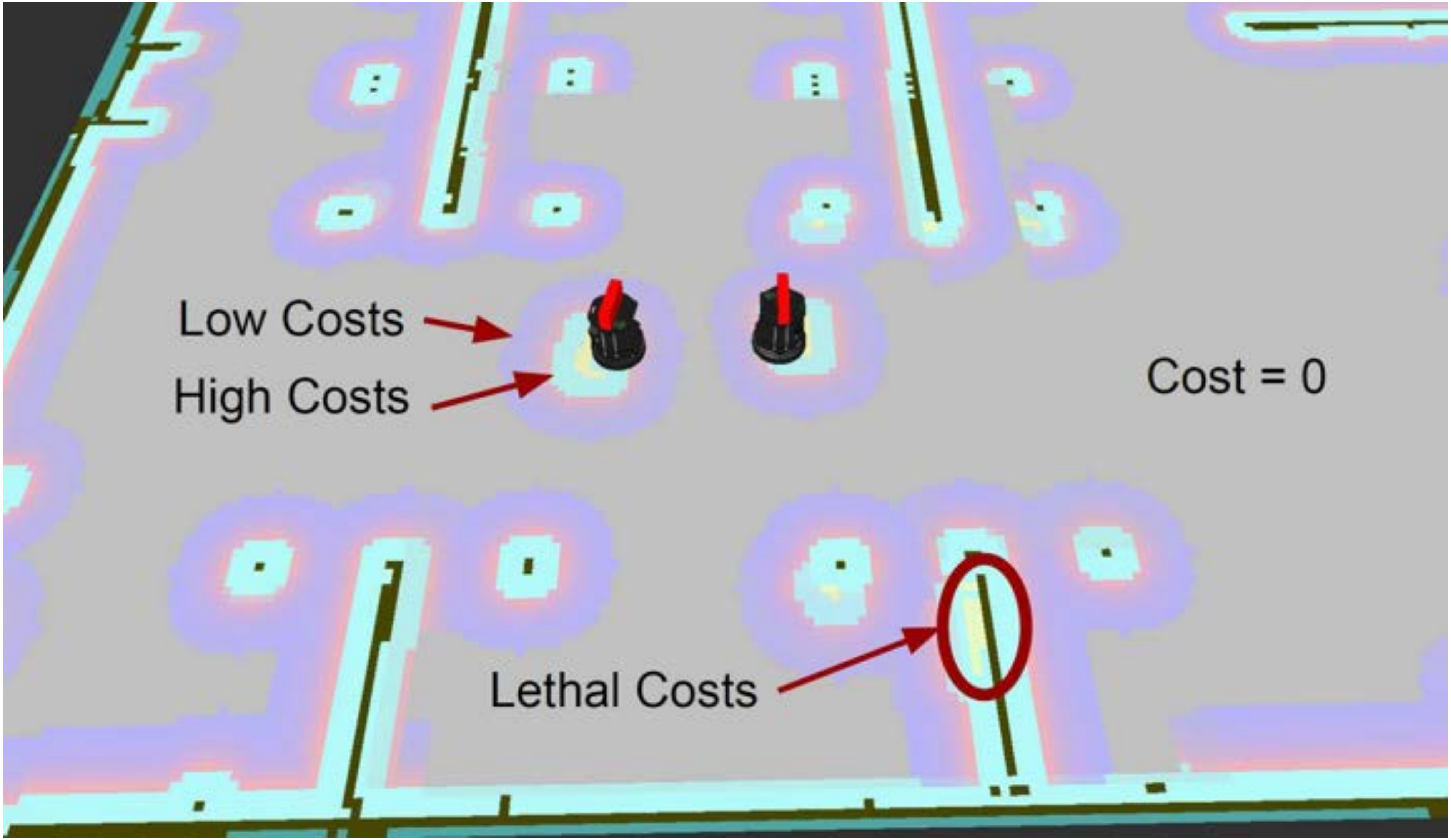


From: http://wiki.ros.org/costmap_2d

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



(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

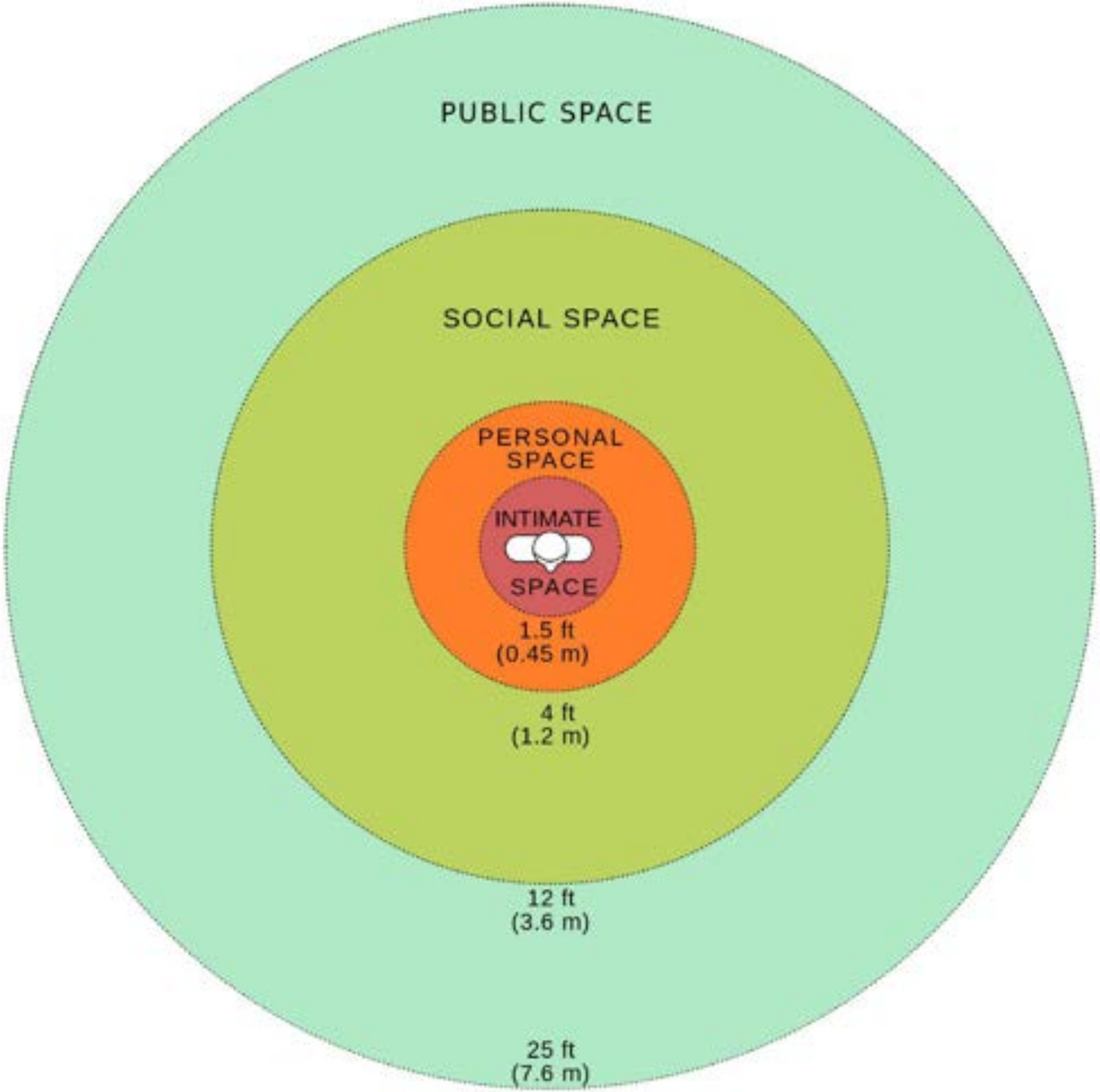
- Human-Aware Navigation
  - Specifically taking the human into account
  - Recall the three goals: comfort, naturalness, and sociability
  - Two main methods:
    1. Stop-and-wait: human does all the hard work
    2. Cost functions based on principles of Proxemics

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


PUBLIC SPACE
SOCIAL SPACE
PERSONAL SPACE
INTIMATE
SPACE
1.5 ft
(0.45 m)
4 ft
(1.2 m)
12 ft
(3.6 m)
25 ft
(7.6 m)

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

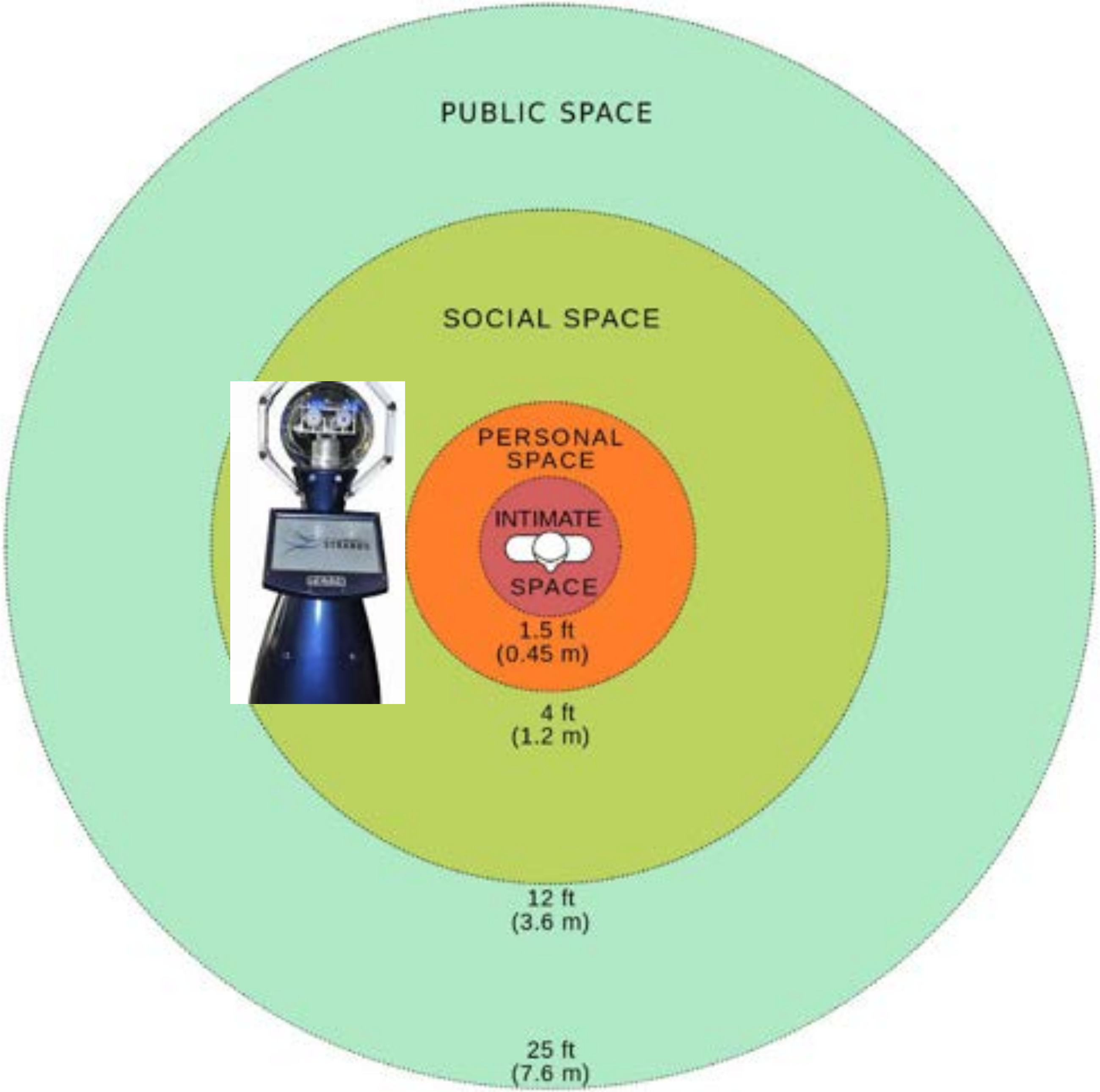
- 45 – 120cm: Family and good friends
- Close Phase (45-75cm)
  - Can grasp other person: peripersonal space
  - No visual distortion of other's features
  - Perception of 3-dimensional qualities of objects
- Far Phase (75-120cm)
  - Outside of grasping distance: at arm's length
  - Other's features 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


PUBLIC SPACE
SOCIAL SPACE
PERSONAL SPACE
INTIMATE SPACE
1.5 ft (0.45 m)
4 ft (1.2 m)
12 ft (3.6 m)
25 ft (7.6 m)

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?

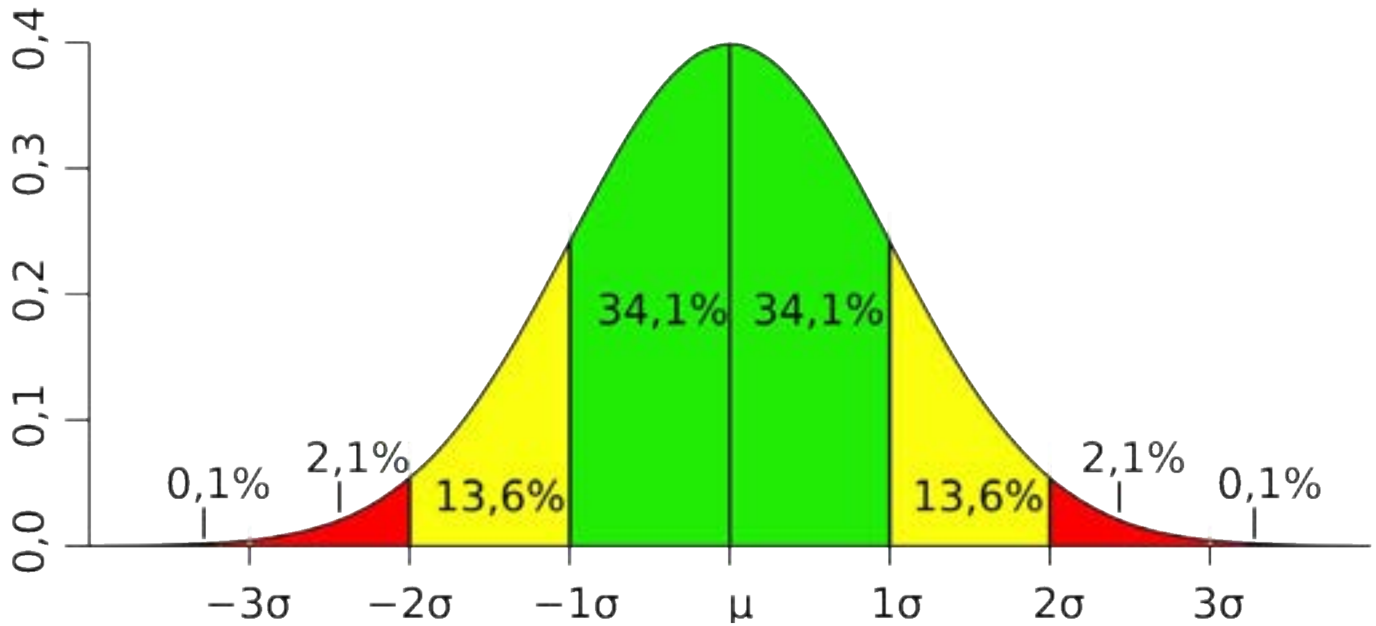
(c) Bringing These Together

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

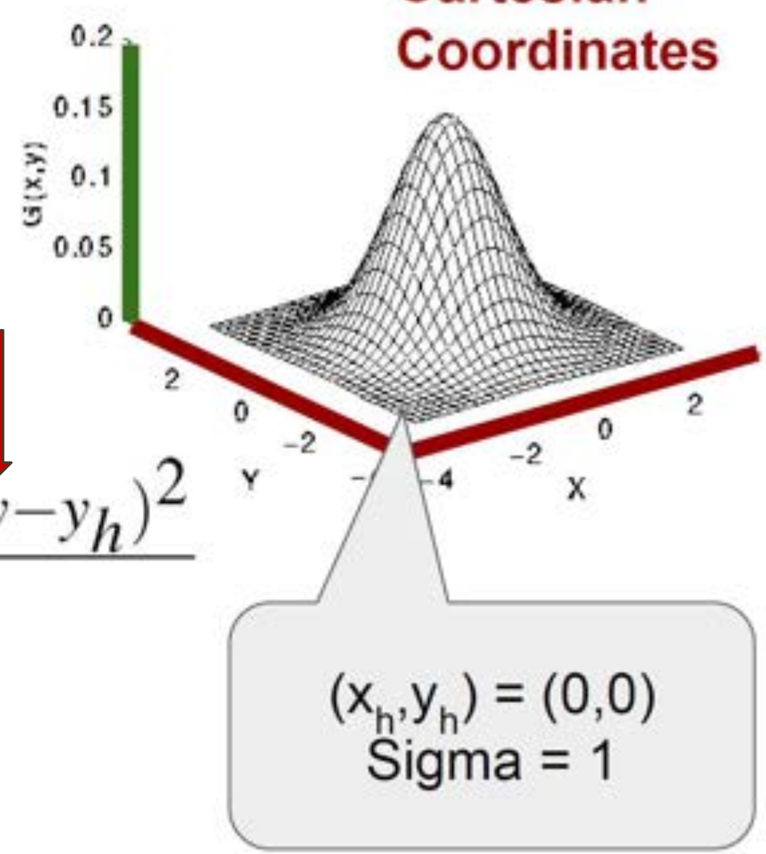


Extending to 2D…

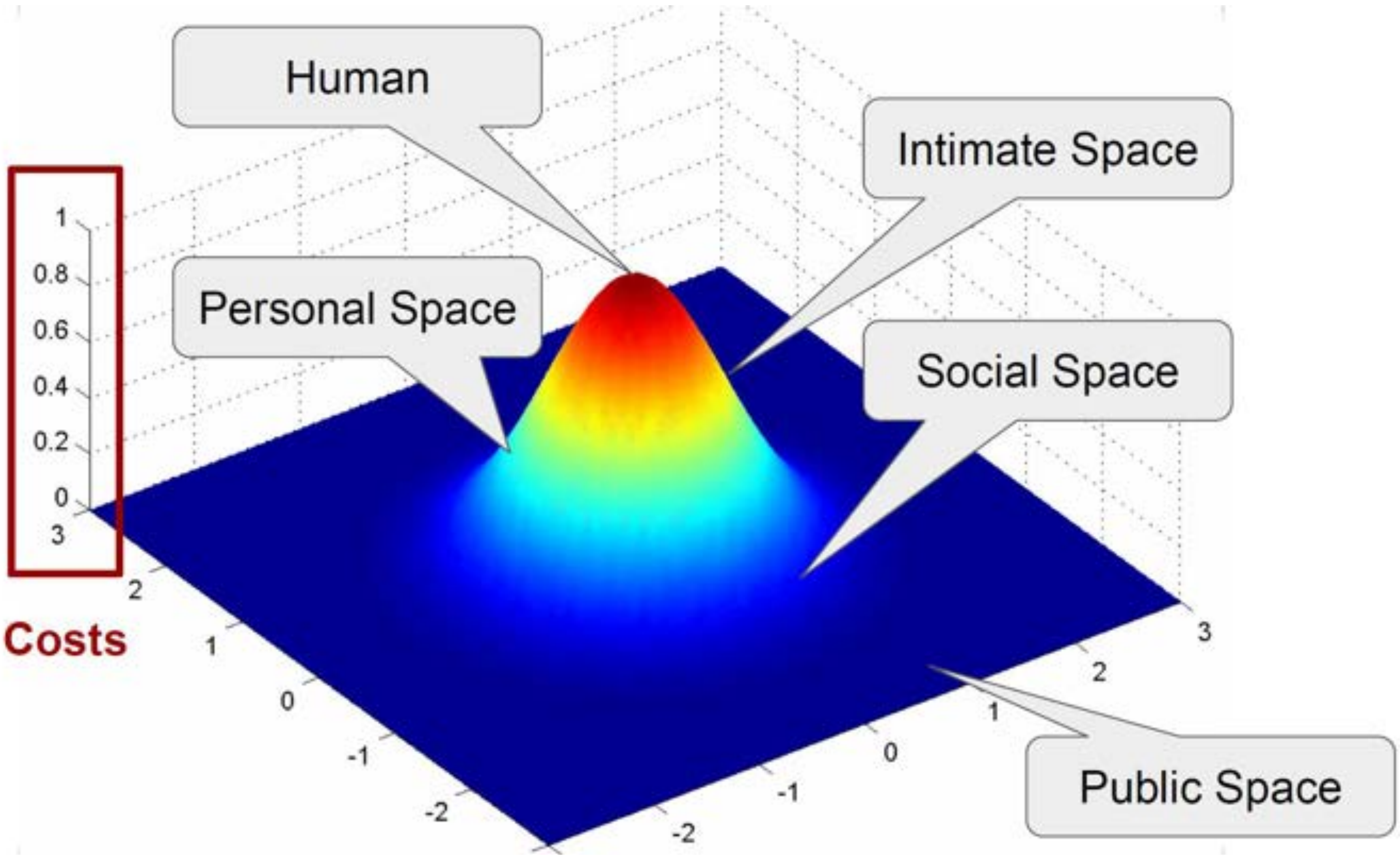
- 2D Gaussian equation – example parameters shown
- Cartesian coordinates centred on the position of the human

Height of the peak

$$G(x,y) = \boxed{\frac{1}{2\pi\sigma^2}} e^{-\frac{(x-x_h)^2+(y-y_h)^2}{2\sigma^2}}$$

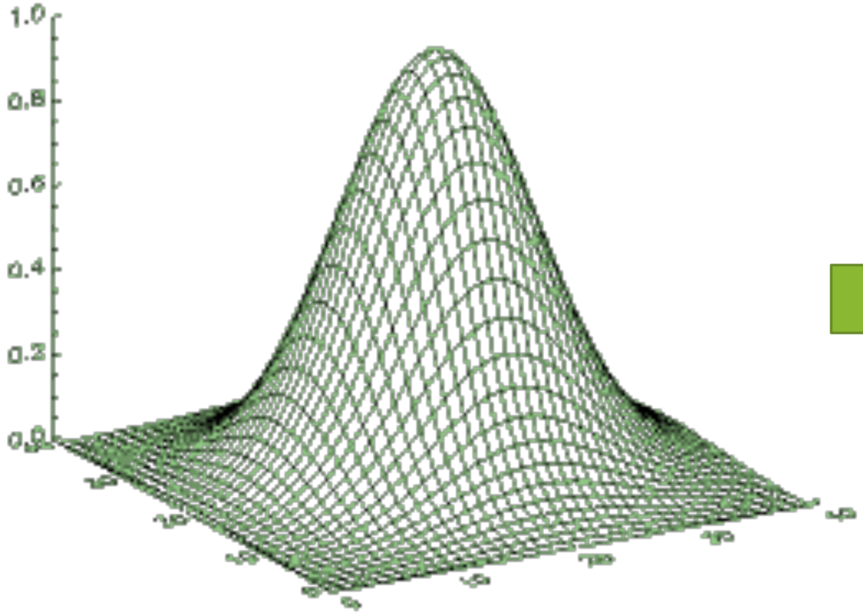


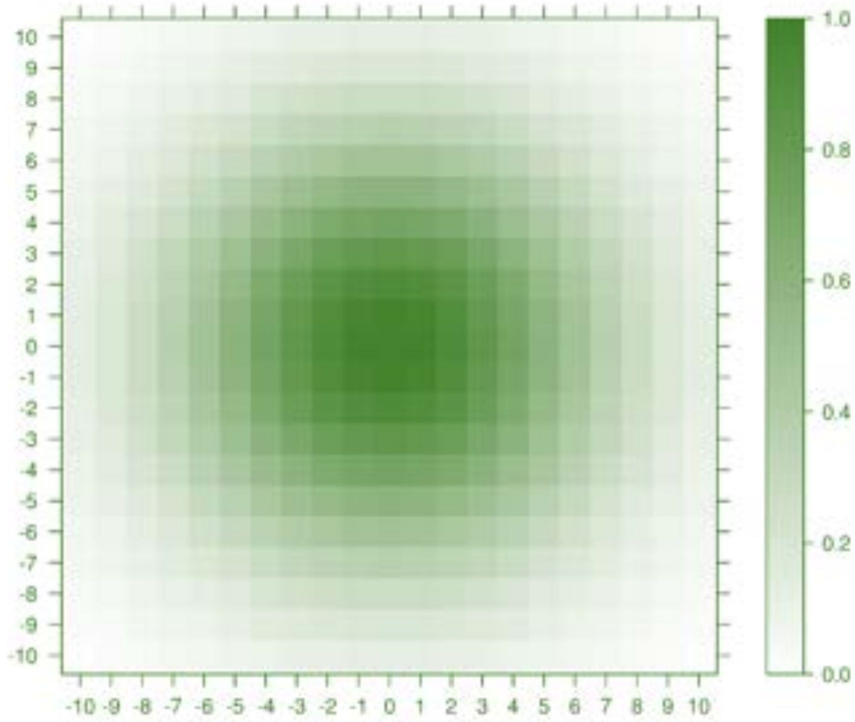
Proxemic Gaussian



Discretisation

- Have a continuous function, need it to be discretised

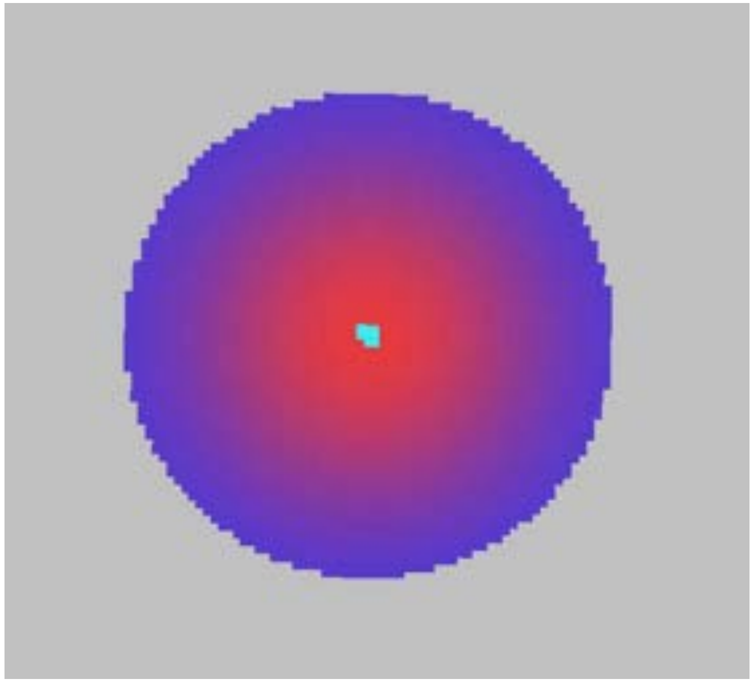




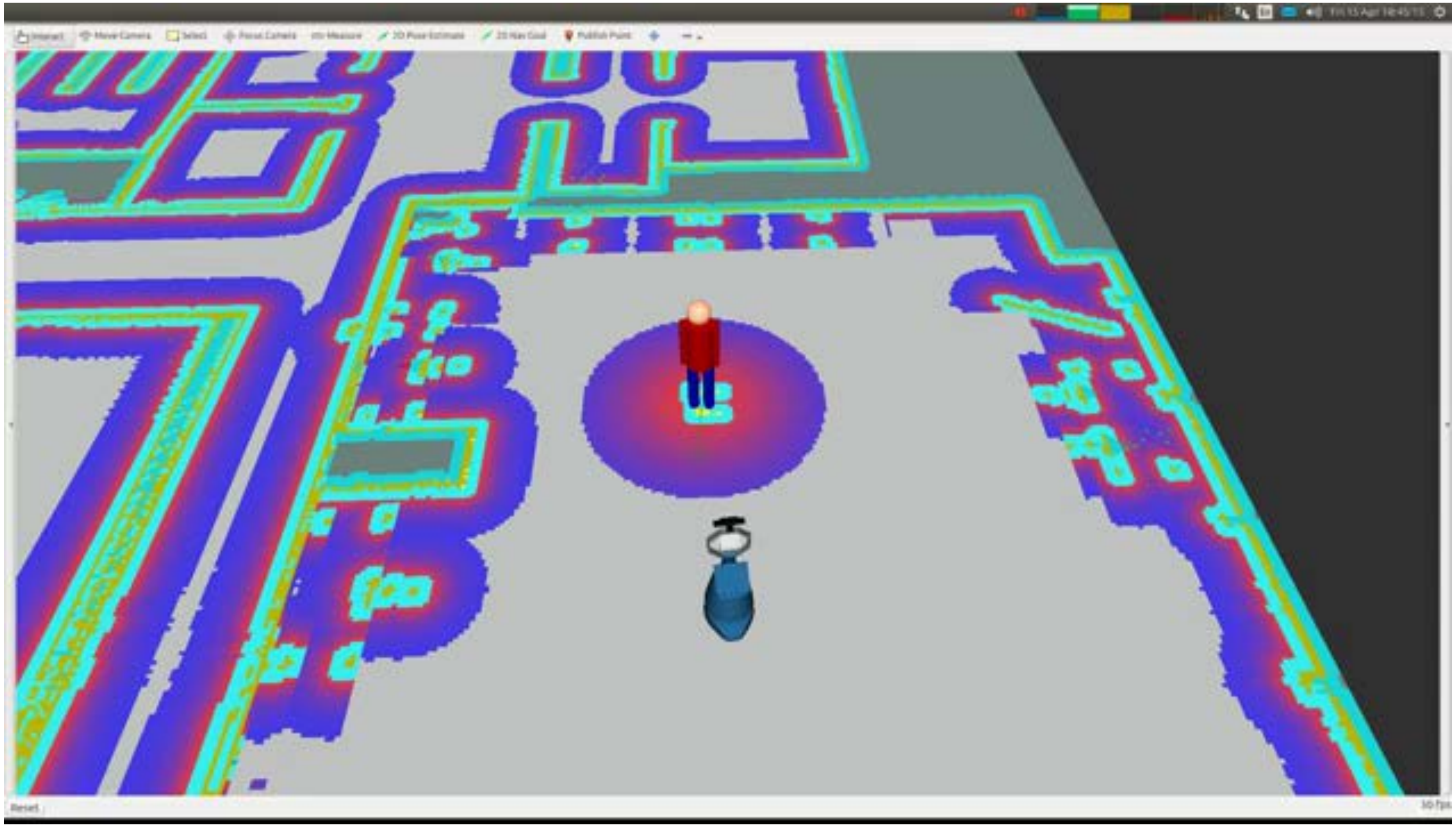
And back to Costmaps…

- 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

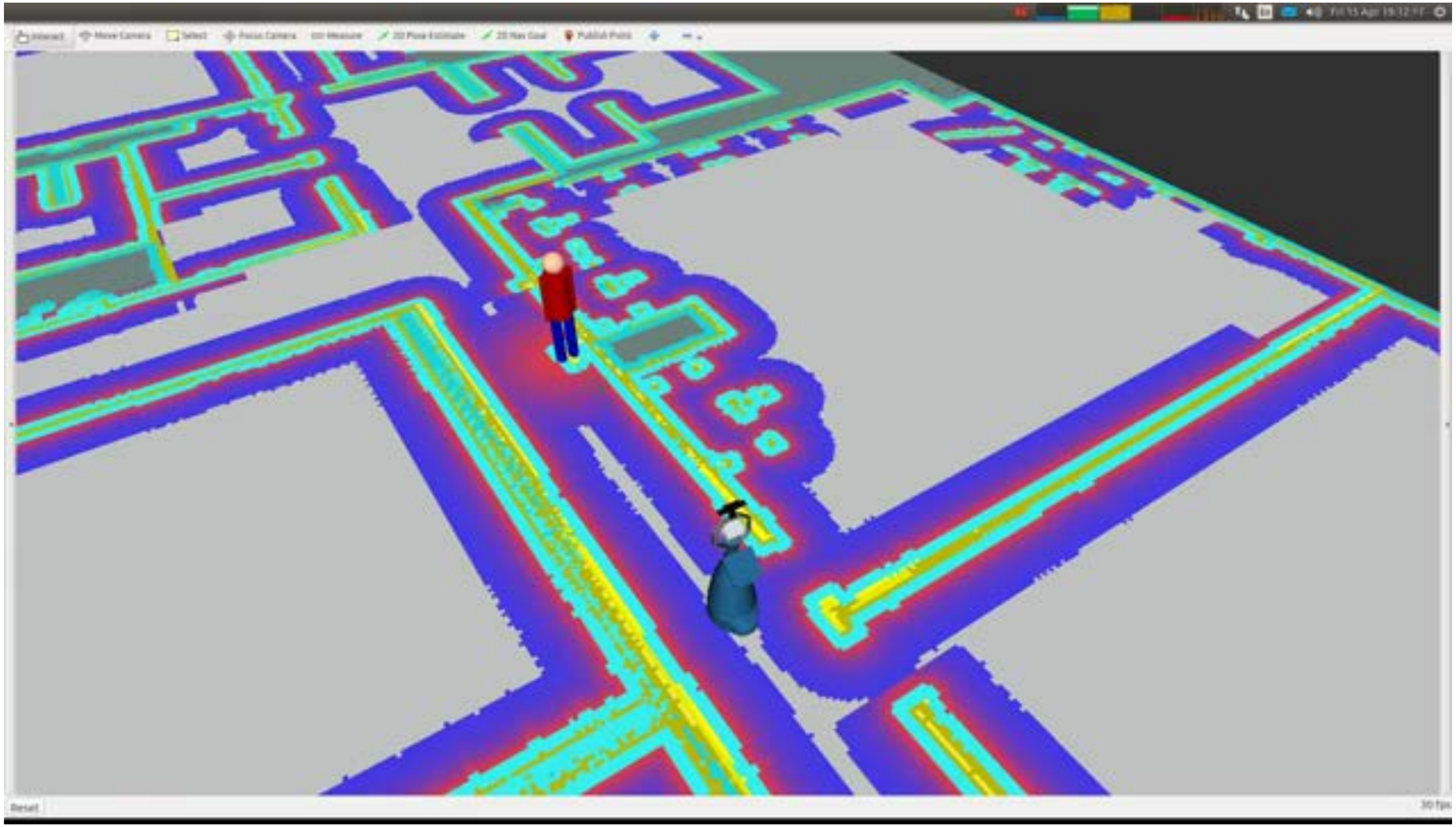


Navigation in Open Space



Full video: https://www.youtube.com/watch?v=pg7g7qv8OMU

Navigation in Corridor



Full video: https://www.youtube.com/watch?v=pg7g7qv8OMU

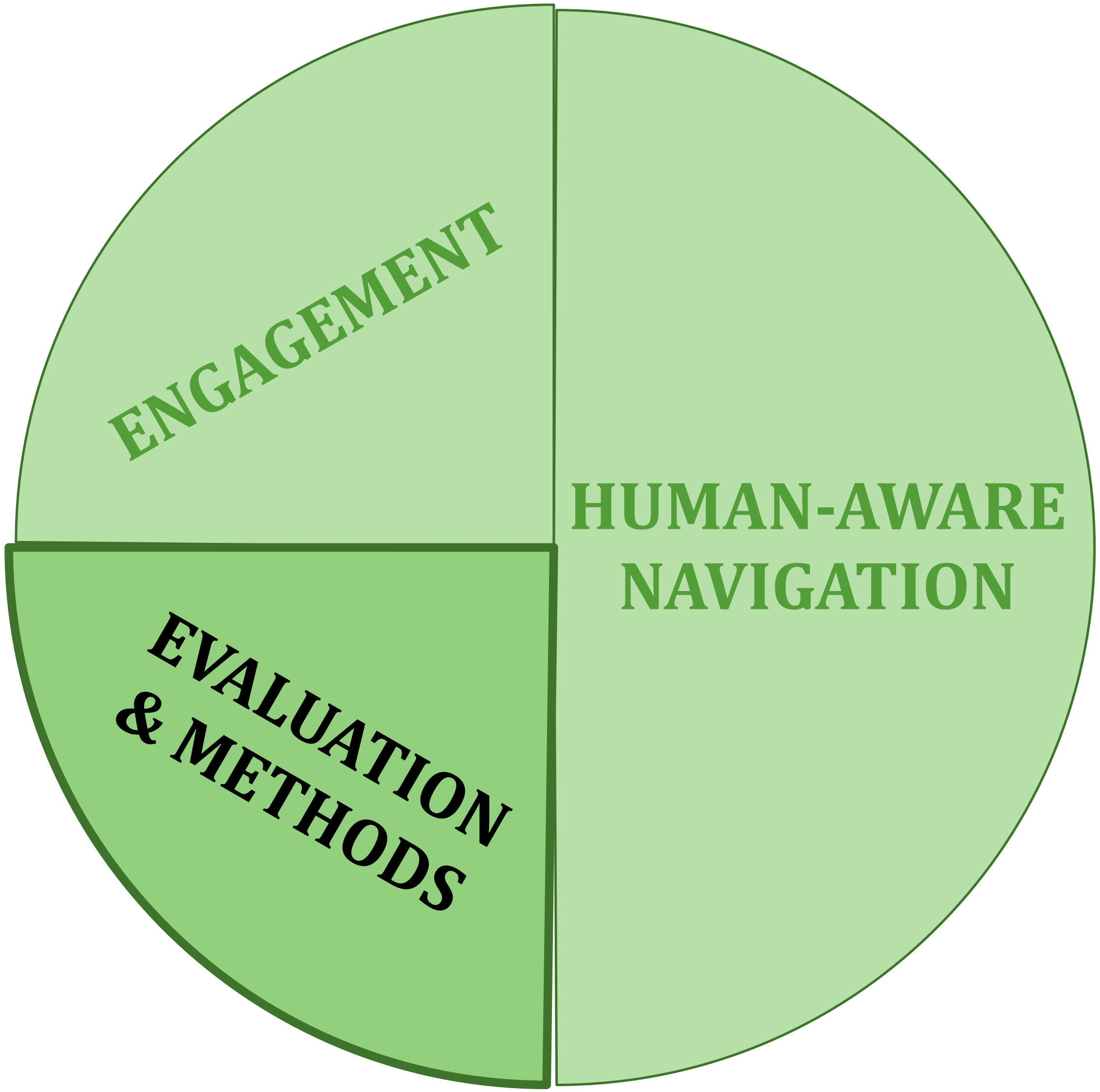
Gaussian Cost functions

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

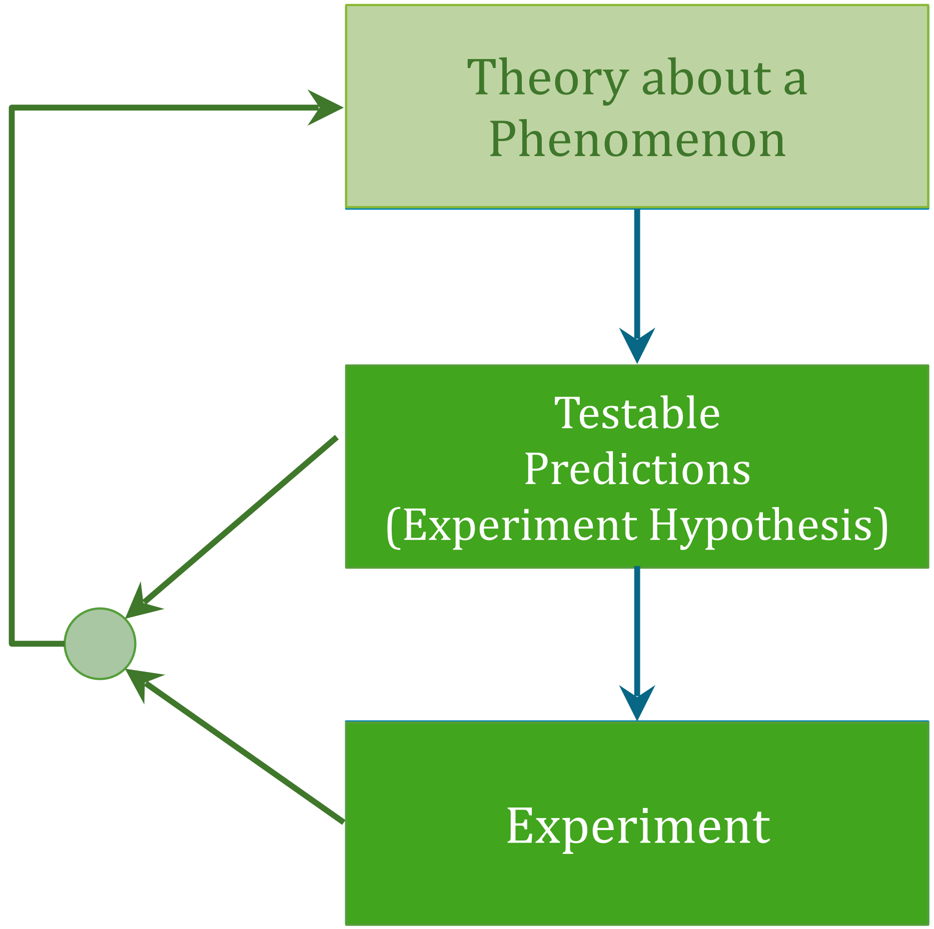

ENGAGEMENT
HUMAN-AWARE NAVIGATION
EVALUATION & METHODS

Is my Robot Successful?

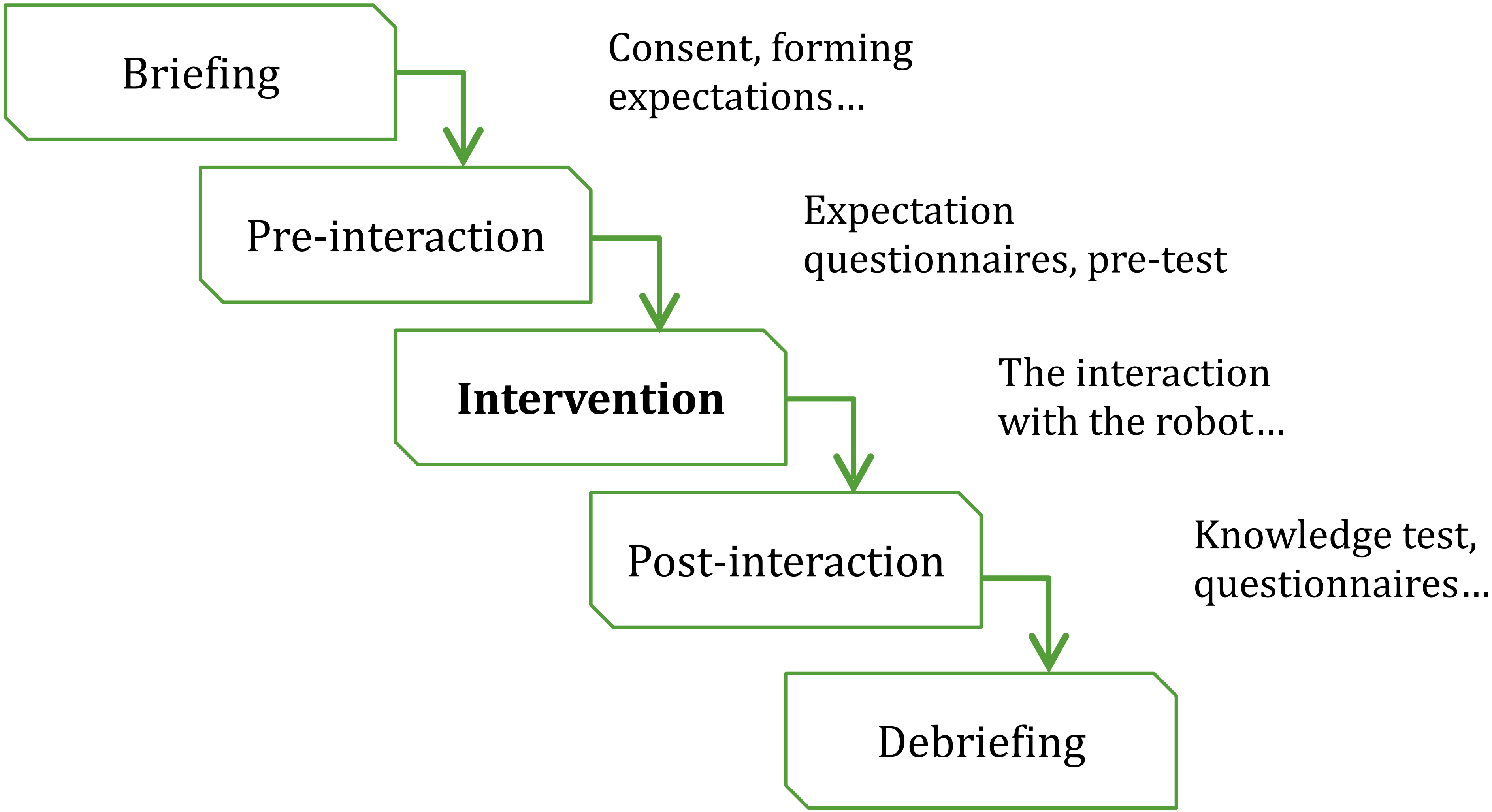
- You have implemented a complete robotic system – how do you know if it is successful?
  - Performance metrics (remember your assignment!)
  - Test with real robots/simulation, etc
- But what if it is an HRI system?
  - Humans are problems…
  - Non-predictable, they come with prior expectations/experience
- 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)

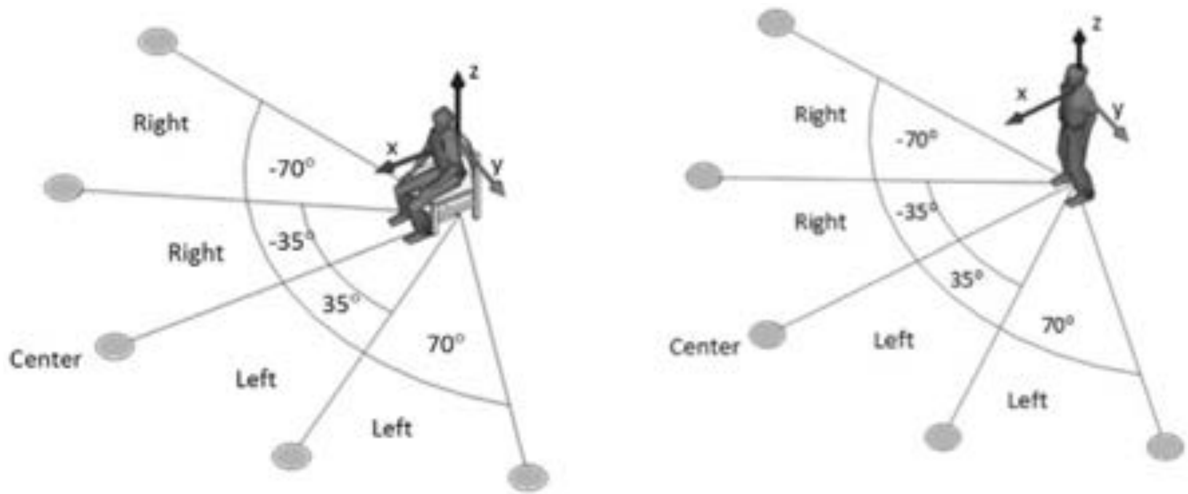


The structure of an experiment…



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).

Table2: 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%)

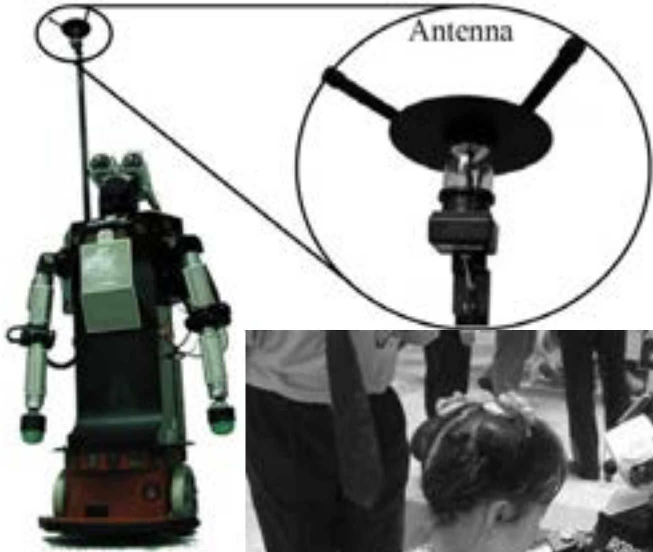
Pilot studies

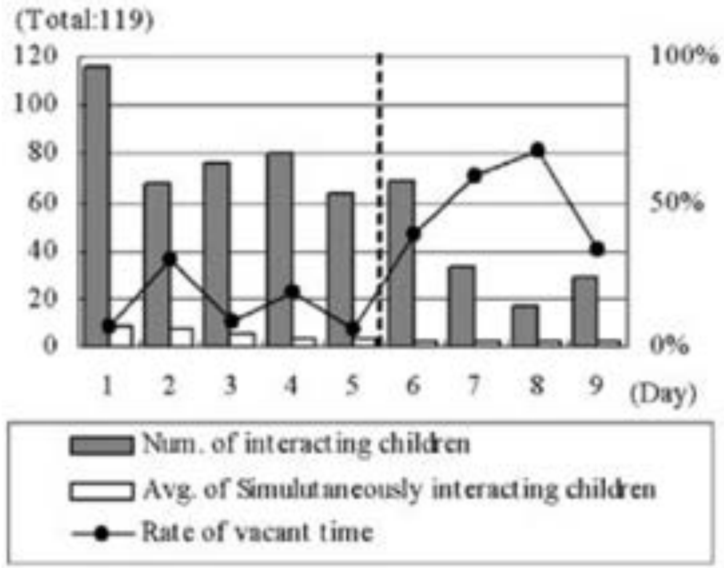
… or *Exploratory Studies*

- 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 .