

CMP3103M/CMP9050M

AUTONOMOUS MOBILE ROBOTS



INTRODUCTION

Marc Hanheide

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



UNIVERSITY OF
LINCOLN

CMP3103M AUTONOMOUS MOBILE ROBOTS

- ▶ **Semester B: AMR**
 - ▶ Prof Marc Hanheide, Dr Athanasios Polydoros
 - ▶ Assessment items:
 1. Robotics Practical Coursework (40%)
 2. Exam (TCA) (60%)

SEMESTER B: AMR

- ▶ **Lectures**
- ▶ **Workshops**
 - ▶ Ubuntu 18.04 LTS, remote facility available if needed
 - ▶ Robot programming in ROS, partially in simulation, partially on real robots
- ▶ Attendance of all scheduled sessions is mandatory!
- ▶ Please check your timetable & blackboard for any timing & updates

WHAT'S ON?

term week	Semester B week w/c	Lecture (9-11)	Topic	Lecturer for Lecture
20	1 21/02/2022	Tuesday, 22 February 2022	Intro	Marc Hanheide
21	2 28/02/2022	Tuesday, 1 March 2022	Robot Programming ROS	Marc Hanheide
22	3 07/03/2022	Tuesday, 8 March 2022	Robot Sensing and Computer Vision	Marc Hanheide
23	4 14/03/2022	Tuesday, 15 March 2022	Motion and Control	Marc Hanheide
24	5 21/03/2022	Tuesday, 22 March 2022	Robot Behaviour	Athanasiос Polydoros
25	6 28/03/2022	Tuesday, 29 March 2022	Navigation	Athanasiос Polydoros
26	04/04/2022	Tuesday, 5 April 2022	EASTER BREAK	
27	11/04/2022	Tuesday, 12 April 2022		
28	7 18/04/2022	Tuesday, 19 April 2022	Localisation	Athanasiос Polydoros
29	8 25/04/2022	Tuesday, 26 April 2022	Robot mapping - SLAM	Athanasiос Polydoros
30	9 02/05/2022	Tuesday, 3 May 2022	Control Architecture	Athanasiос Polydoros
31	10 09/05/2022	Tuesday, 10 May 2022	HRI1	Marc Hanheide
32	11 16/05/2022	Tuesday, 17 May 2022	HRI2	Marc Hanheide
33	12 23/05/2022	Tuesday, 24 May 2022	Mock Exam	Athanasiос Polydoros

TURTLEBOT 2 – OUR WORKSHOP BUDDY

This year's
assignment:
escape the
maze
(more soon...)



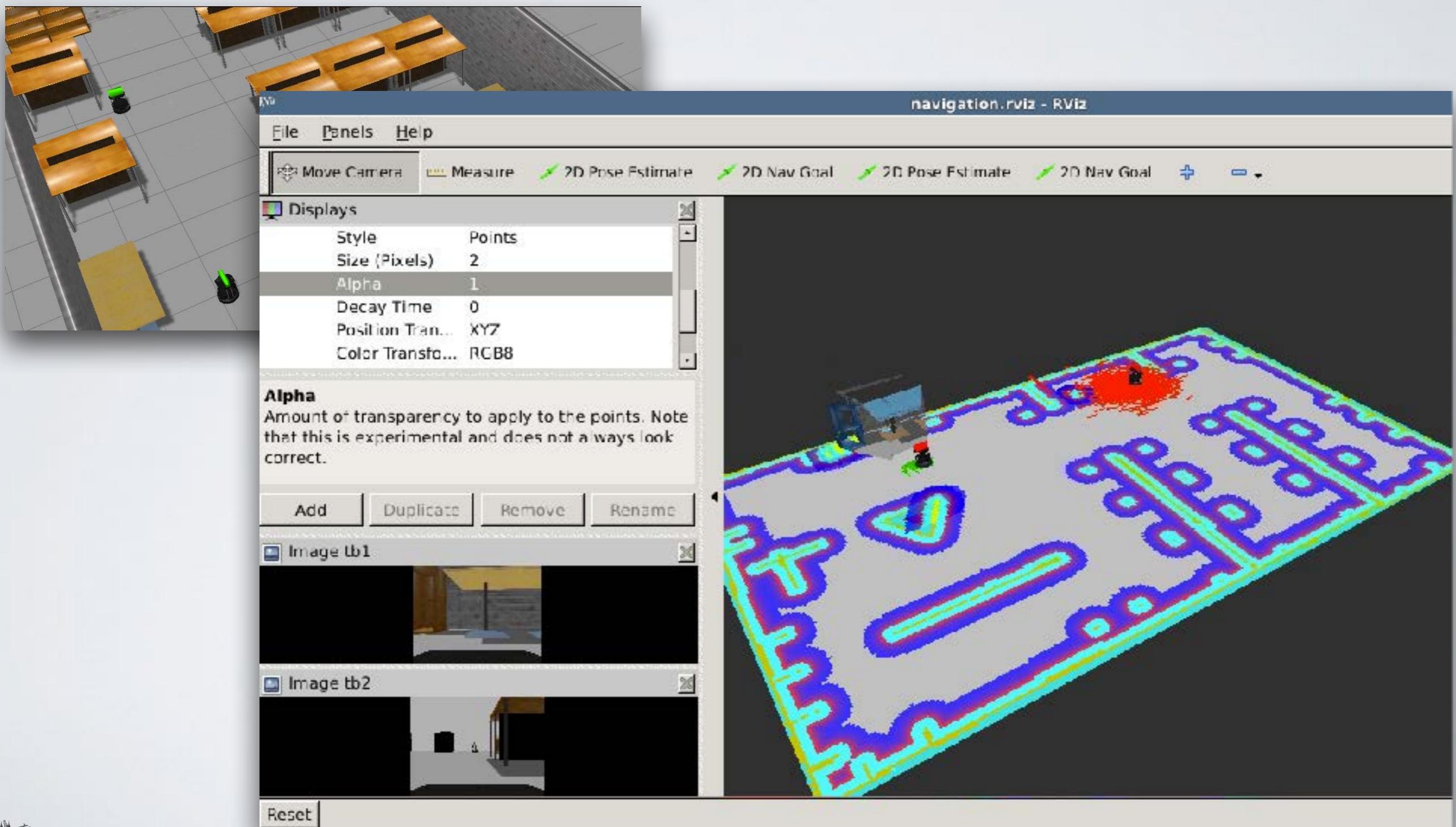
Access Code: 299262

ROS
Enabled

SIMULATED TURTLEBOTS



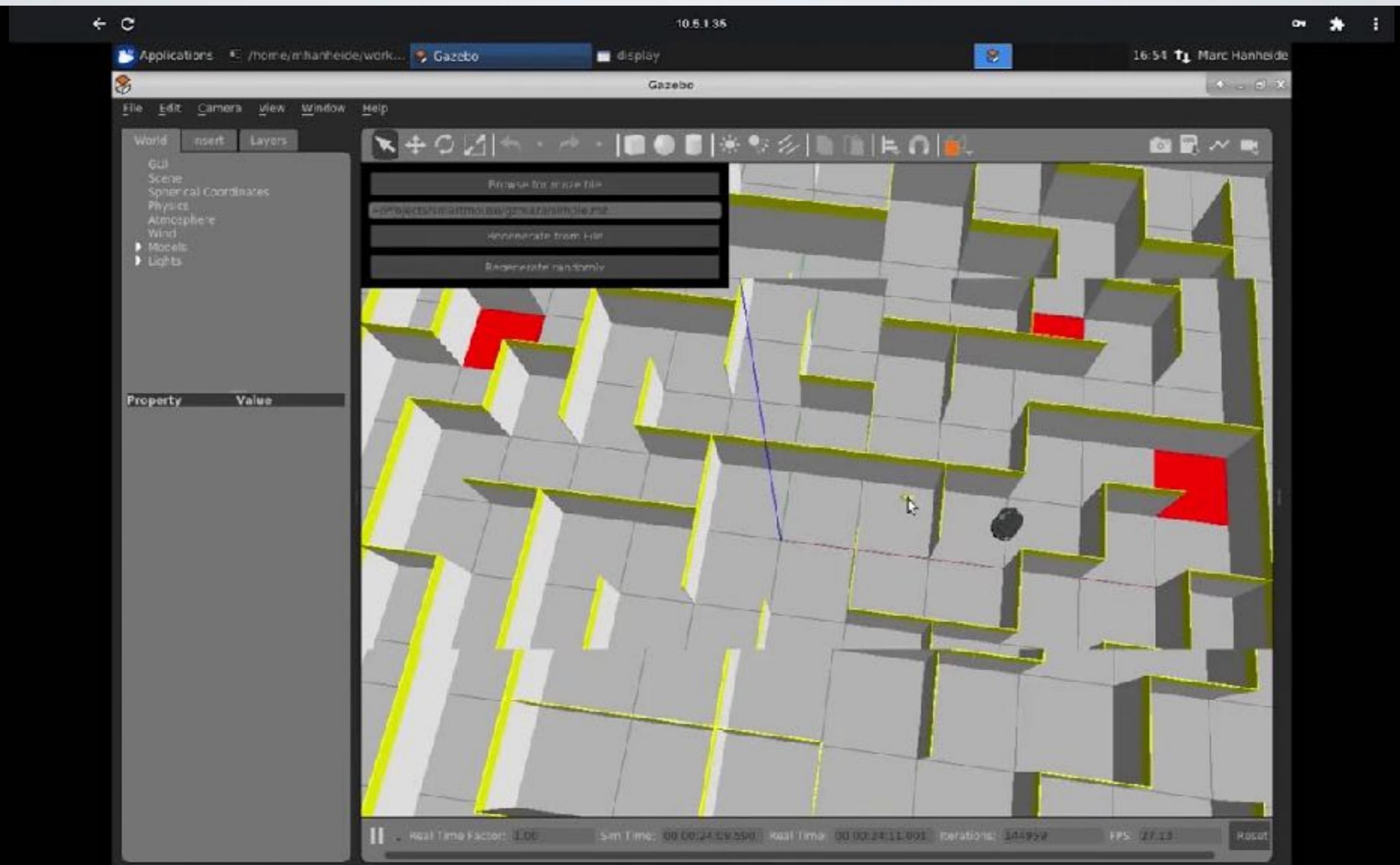
SIMULATED TURTLEBOTS



COURSEWORK

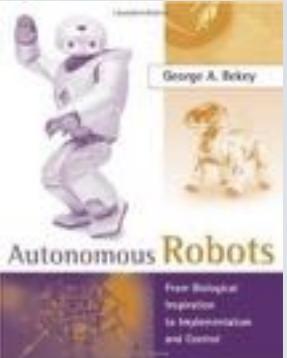
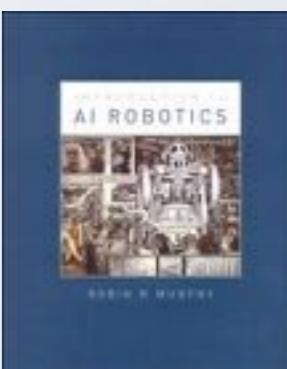
“Escape the Maze” Your task is to develop a reactive robot behaviour, programmed in Python using ROS (the Robot Operating System), that enables a robot to find a goal in a maze, utilising the robot’s sensors (LIDAR, Odometry and camera). This assessment is entirely assessed in simulation, and not on any real robot. As part of this task, you must submit an implementation (criterion 1) and a video presentation (criterion 2).

COURSEWORK



READING MATERIAL

<https://rl.talis.com/3/lincoln/lists/746F02F3-894F-7C65-99A7-5158F99727A8.html>

- ▶ 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 



LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

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?



POLLEV

- ▶ <https://pollev.com/mhanheide>

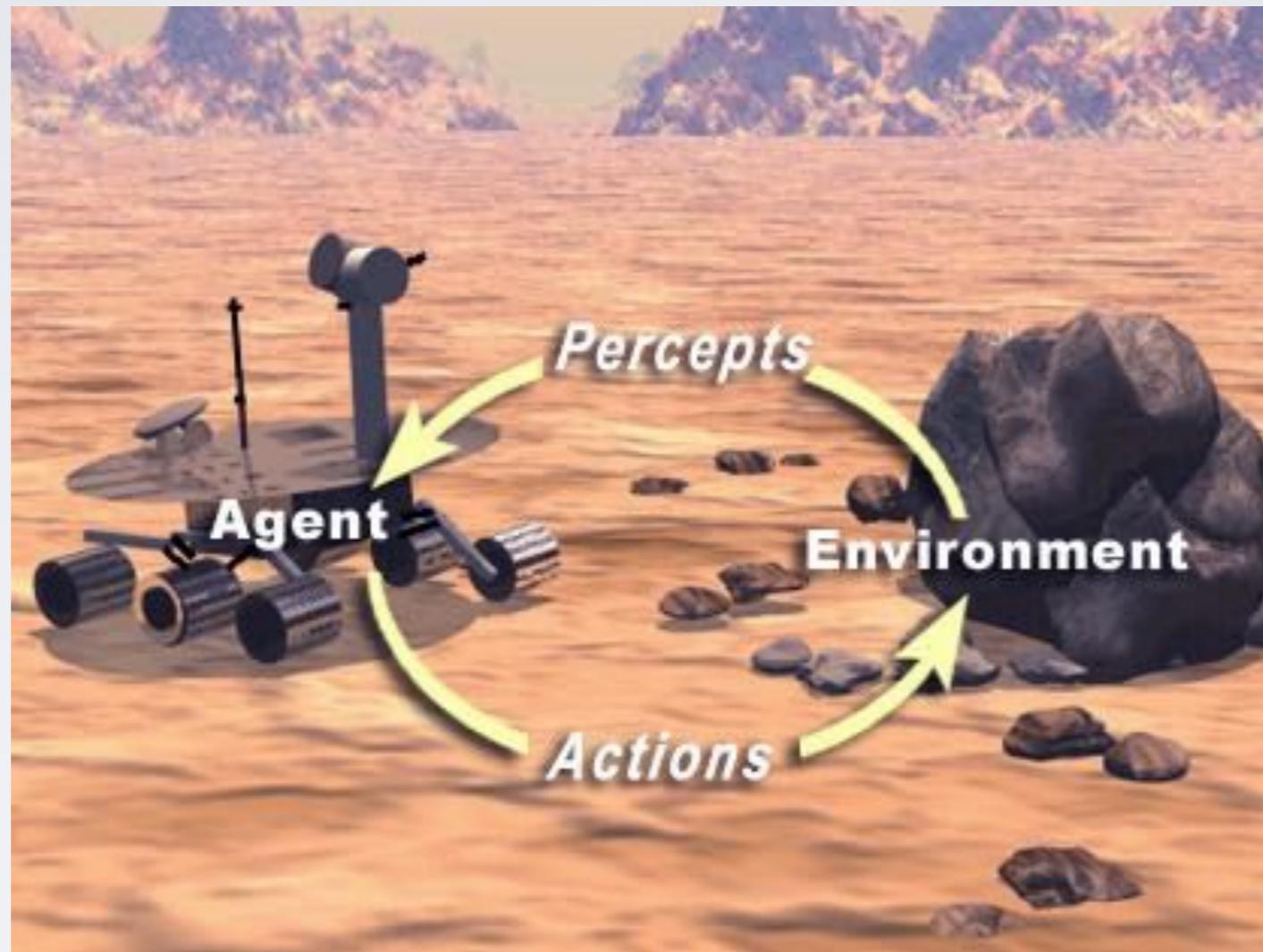


What do you associate with the word robot?



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

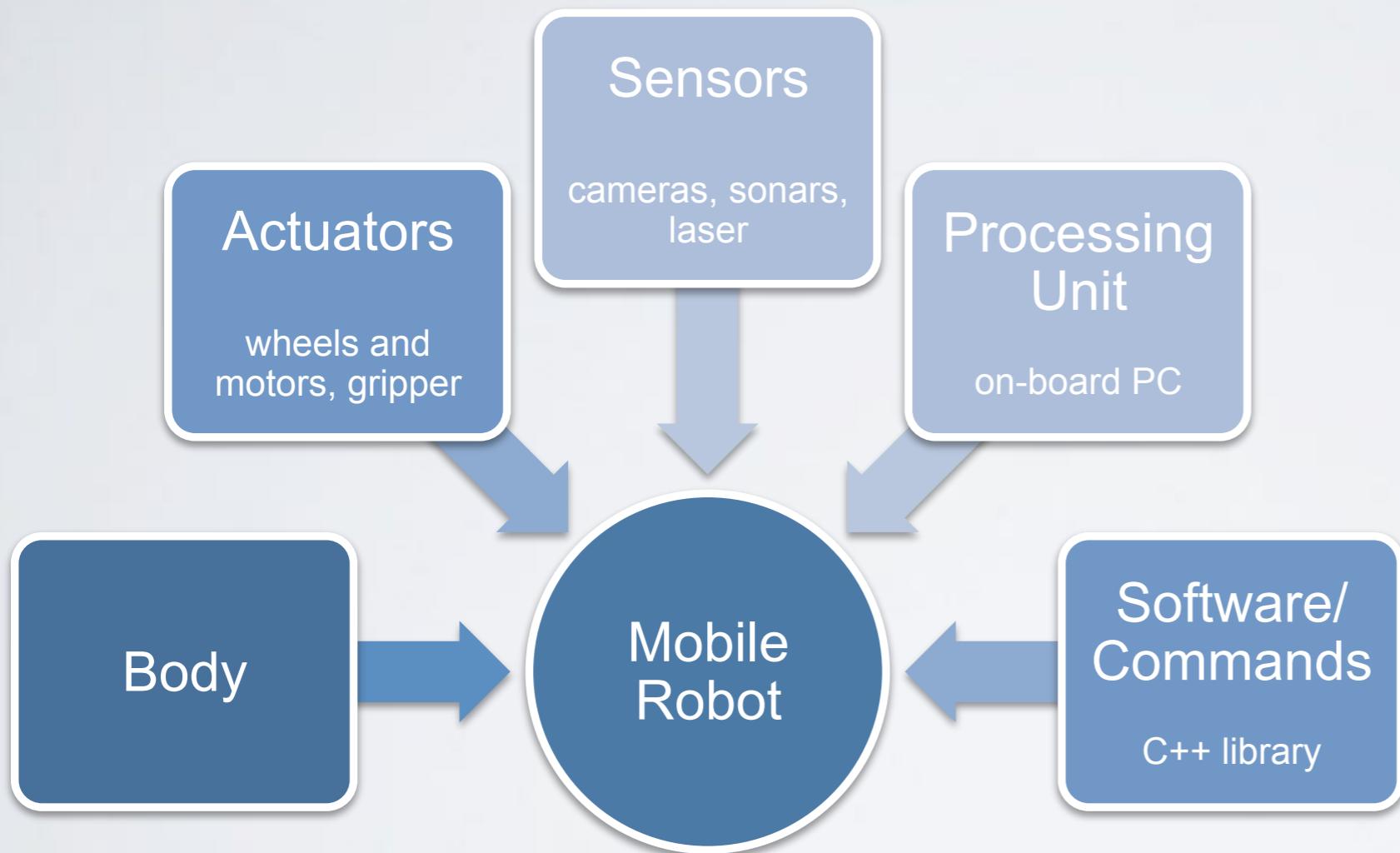
- Hardware Controllers
- Actuators
- Locomotion (e.g. wheels or legs)
- Embodiment (physical interaction with the world)
- Internal Sensors
- External Sensors

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

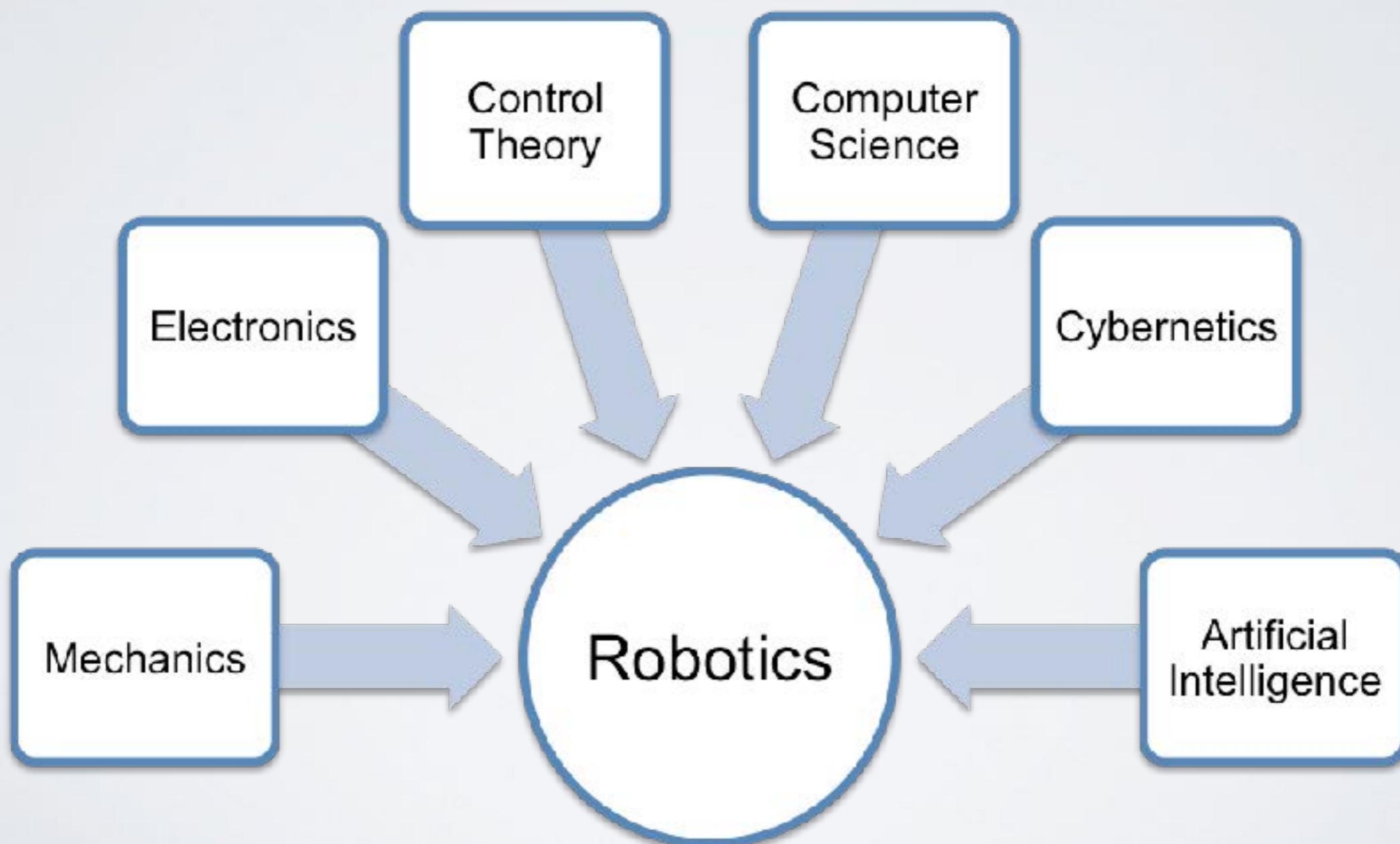


ANATOMY OF A MOBILE ROBOT



ROBOTICS

Science and technology of robots



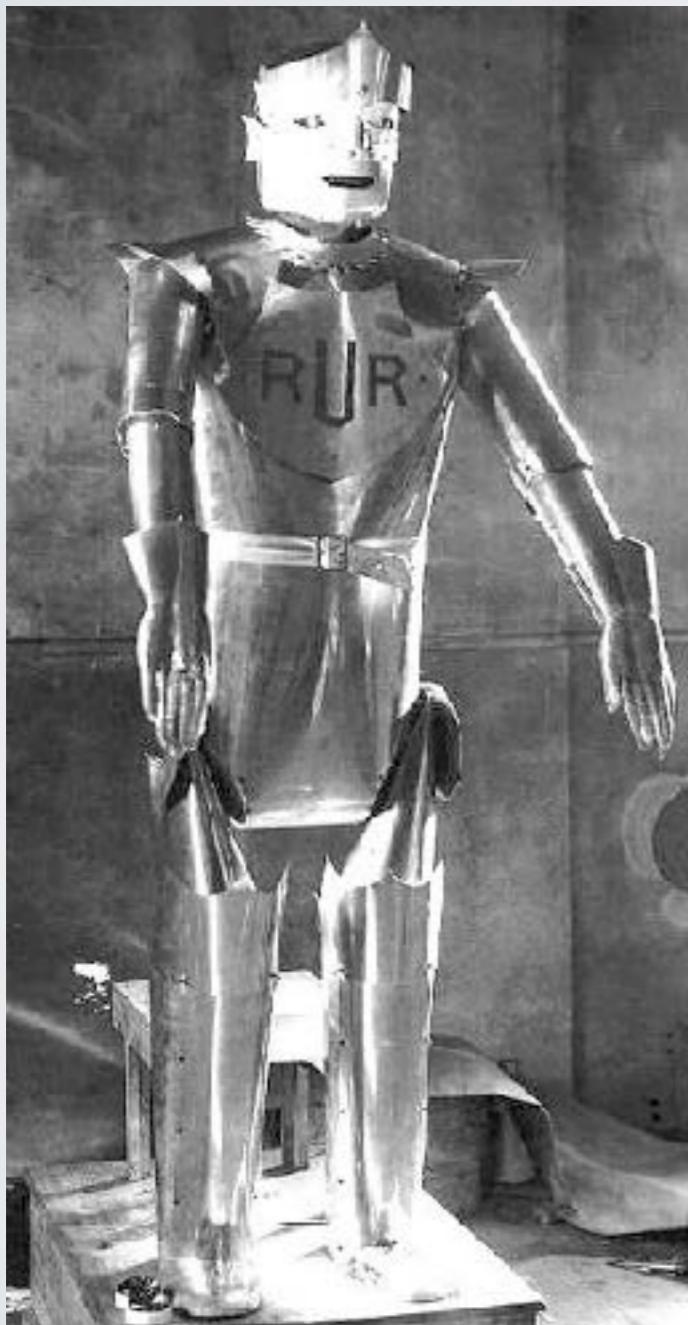


UNIVERSITY OF
LINCOLN



PAST AND PRESENT

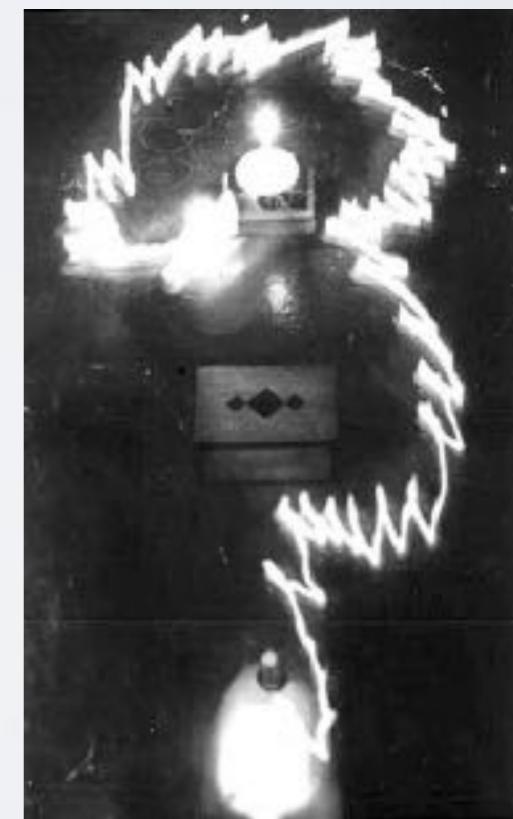
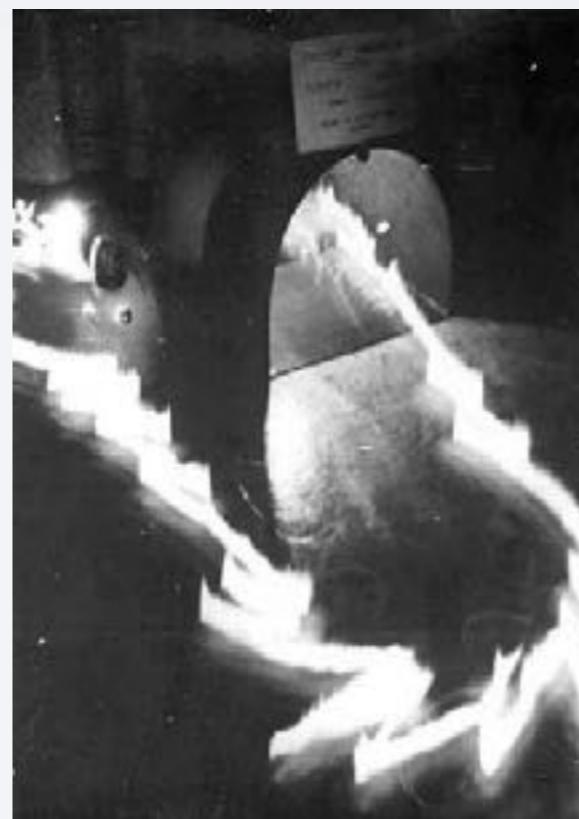
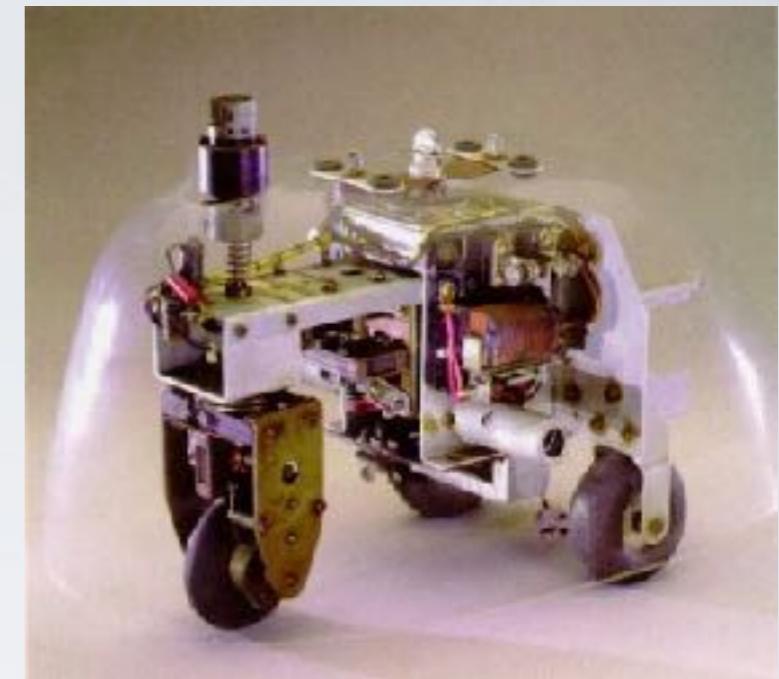
R.U.R.



- ▶ Karel Čapek, 1921
- ▶ R.U.R. - Rossum's Universal Robots, a stage play
- ▶ the word 'robot' appears for the first time
- ▶ 'roboťa' – forced/hard labour in Czech
- ▶ robots – artificial men that can think but seem to be happy to serve
- ▶ exploited (?) by humans
- ▶ finally rebel against their creators

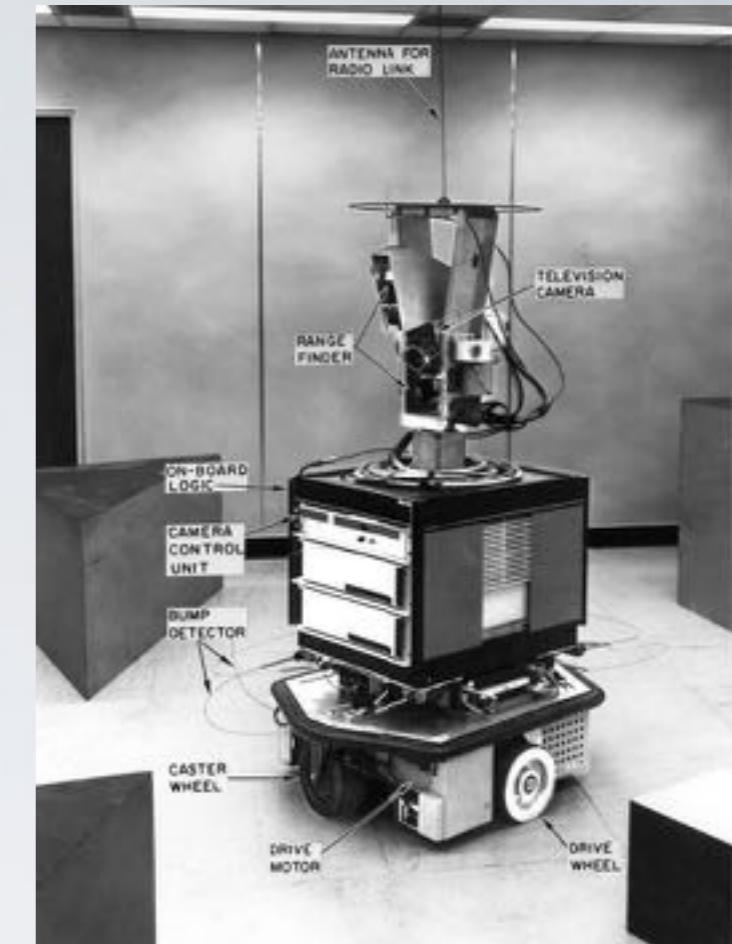
GREY WALTER'S TURTLES

- ▶ “Machina Speculatrix”, 1948
- ▶ experiments in reflex behaviour
- ▶ built of electronic valves and photo-cells
- ▶ approach or escape a light source



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

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

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



BREAK?!

SENSORS AND ACTUATORS

EFFECTORS AND ACTUATORS

- ▶ Effectors:

- ▶ hand, arm, gripper – manipulators
- ▶ wheels, legs, tracks, rotors – mobile robots



- ▶ Actuators:

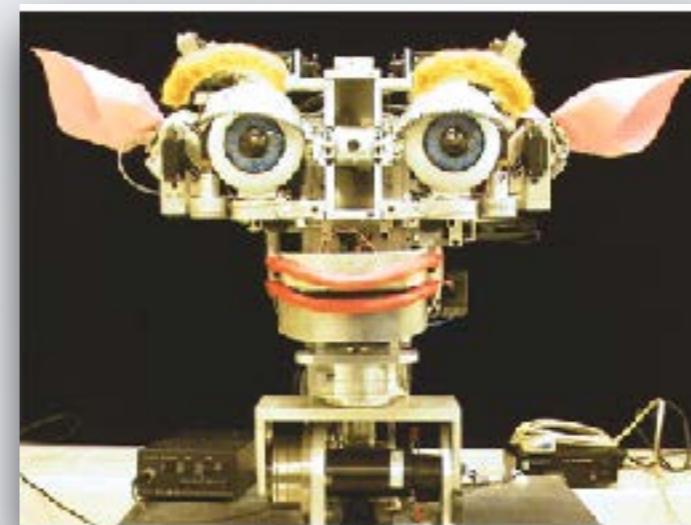
- ▶ electric motors, pneumatic and hydraulic systems



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 exteroceptive sensors?

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

SENSORS: CLASSIFICATION

▶ **Proprioceptive**

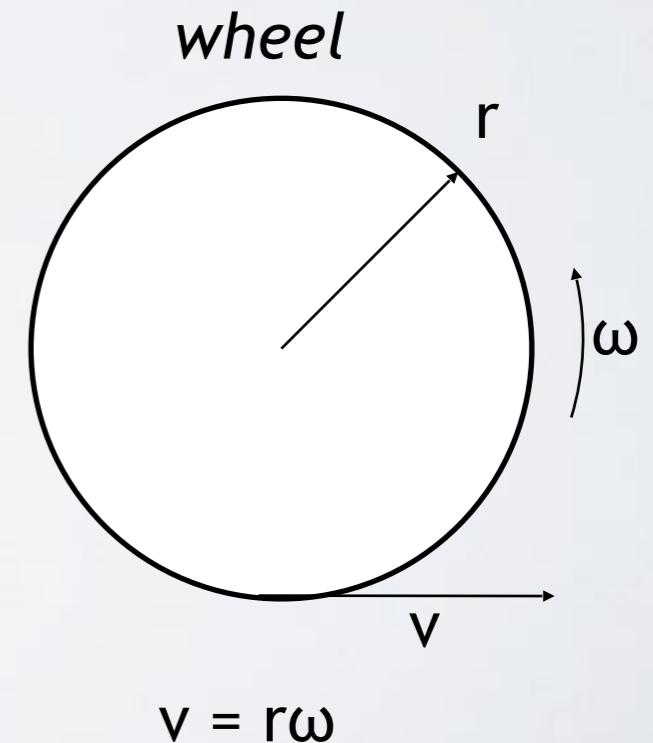
- ▶ internal state of the robot, self-awareness
- ▶ e.g. odometry, battery level, temperature

▶ **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)

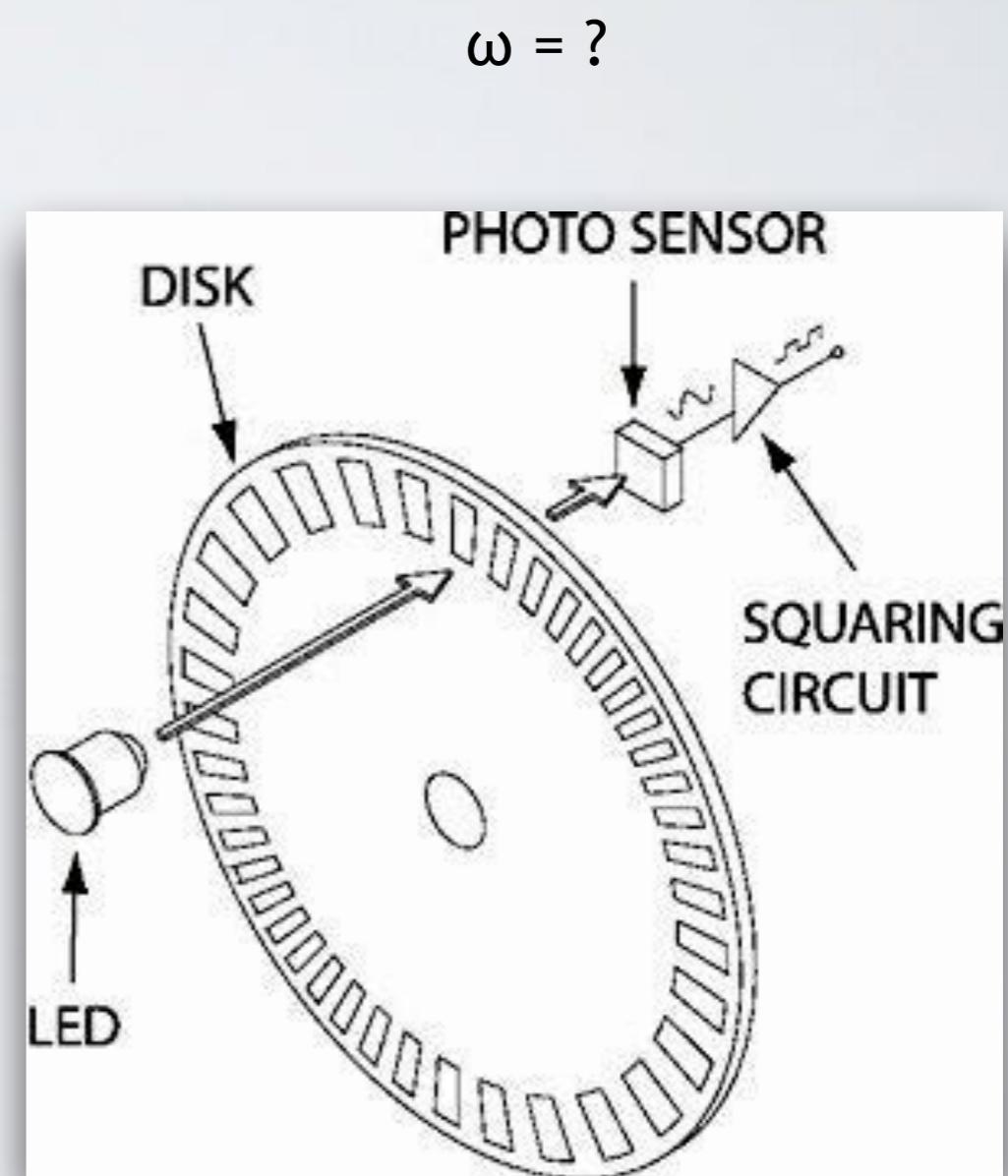
ODOMETRY – WHERE AM I?

- ▶ 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



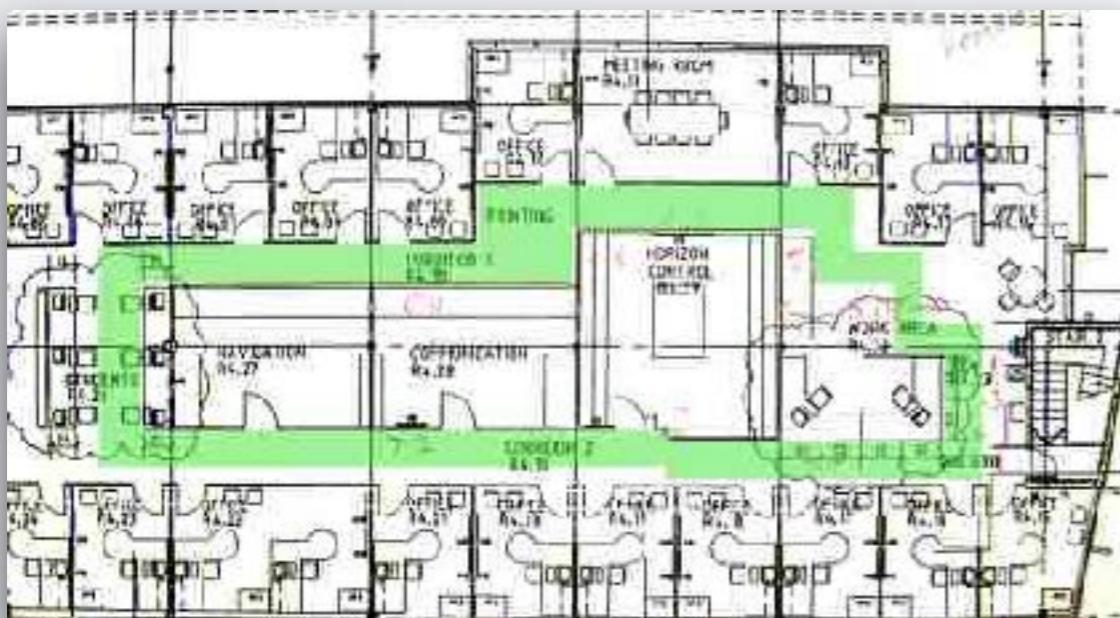
WHEEL SPEED ESTIMATION

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

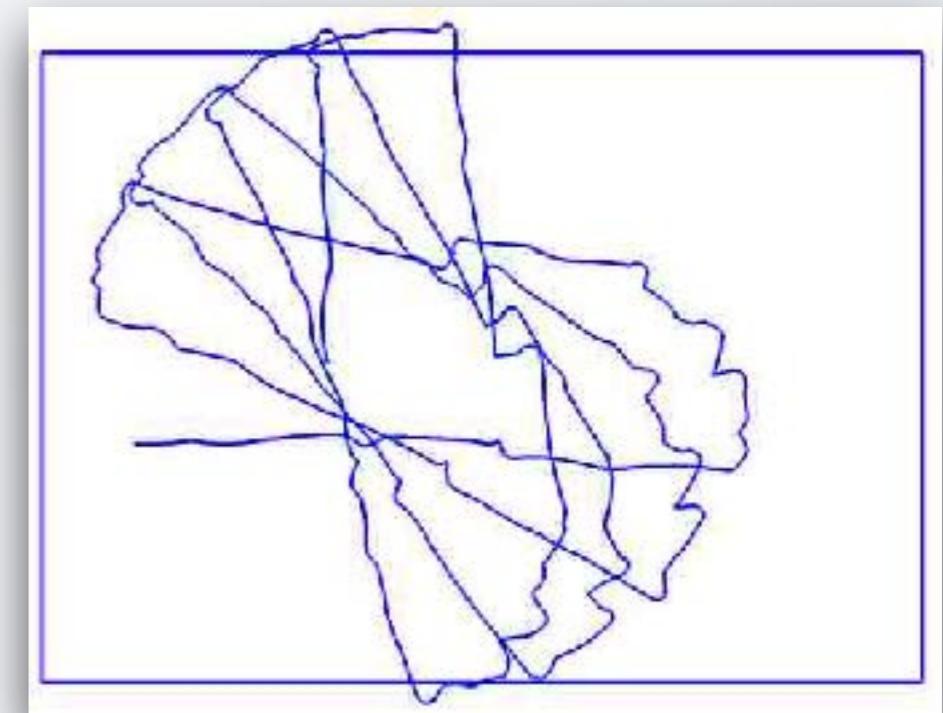


ODOMETRY – LIMITATIONS

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



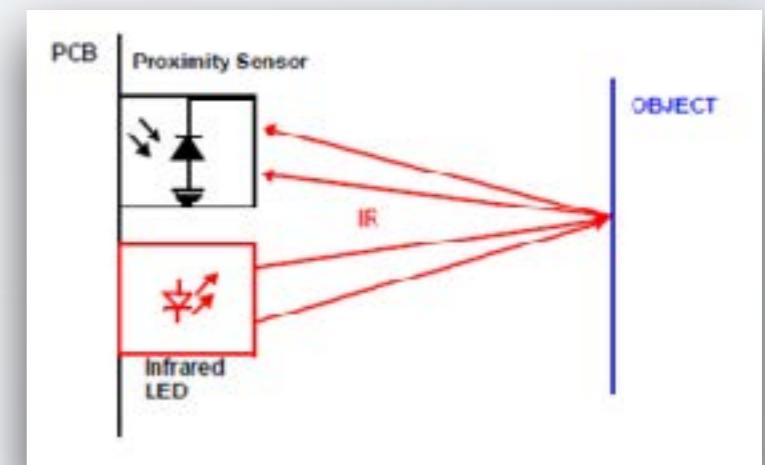
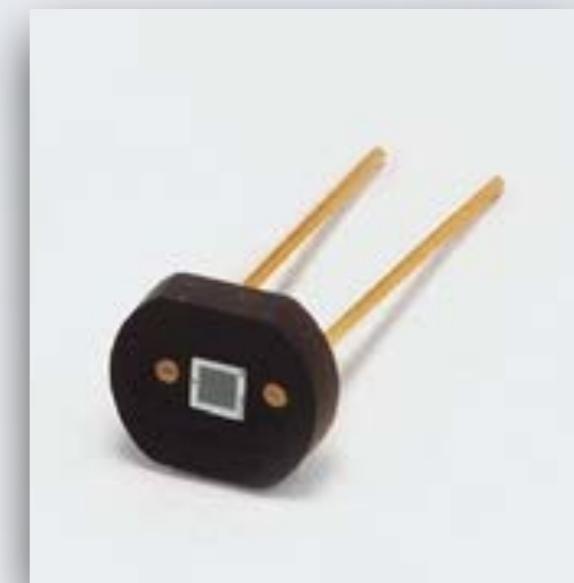
Floor plan and robot route



Robot's perceived trajectory using odometry

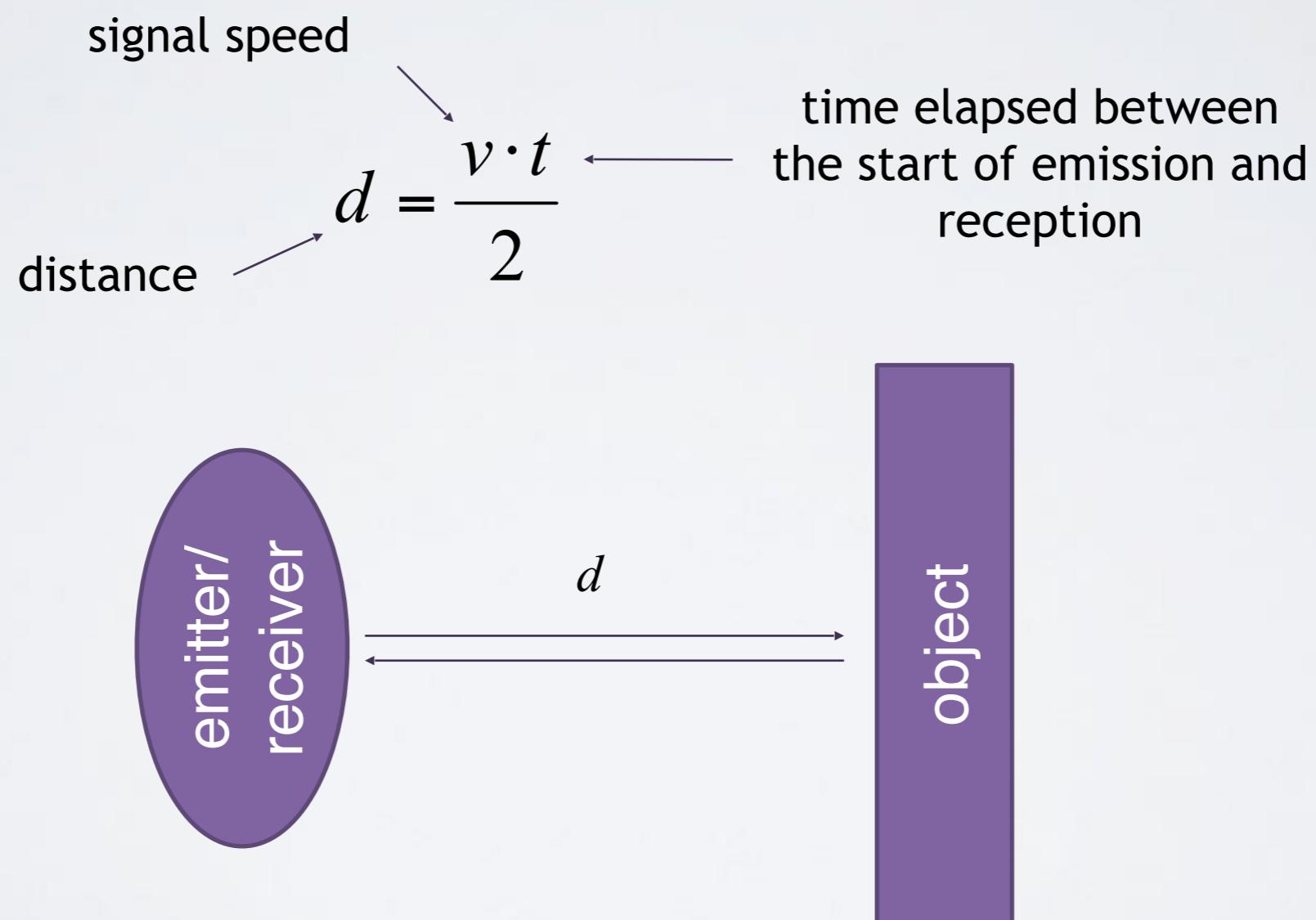
LIGHT SENSOR

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



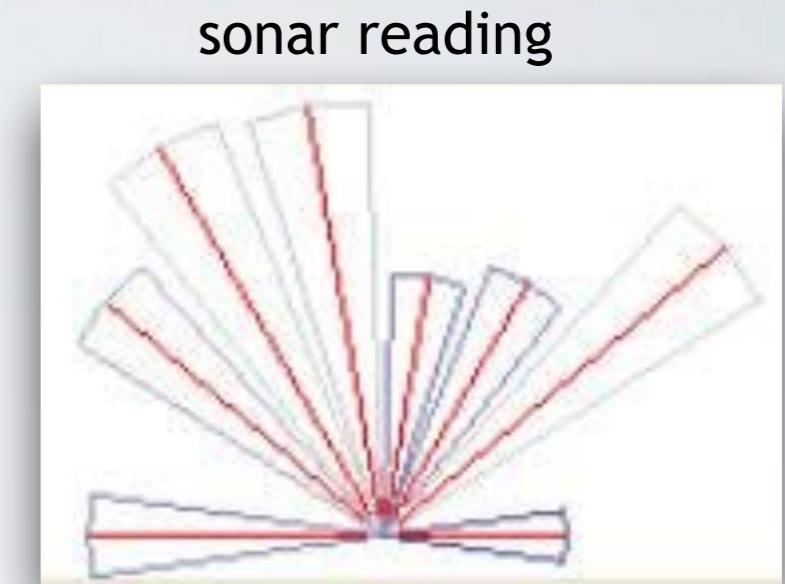
TIME OF FLIGHT SENSORS

- ▶ Active distance measurements – reflected signal



SONAR SENSOR

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

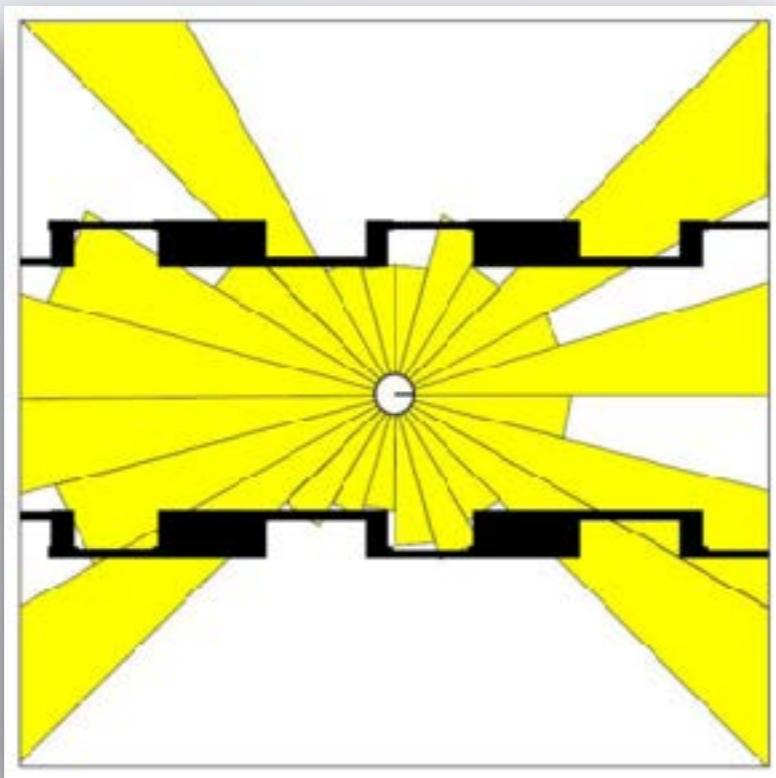


an array of
sonar sensors

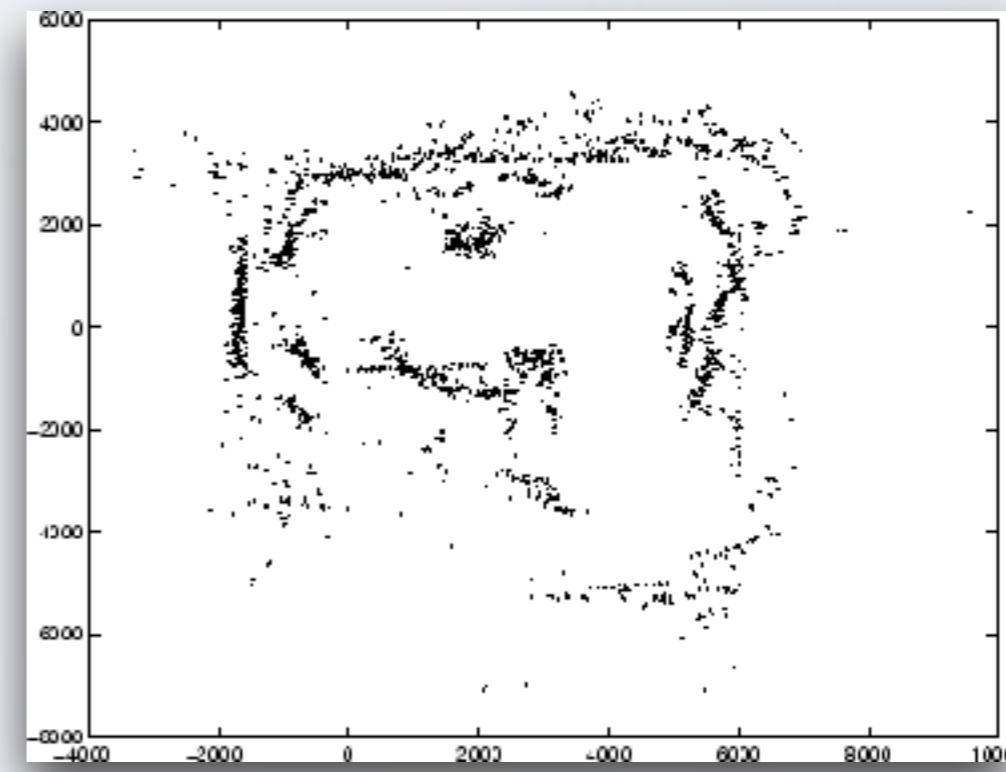


SONAR – APPLICATIONS

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



sonar reading in a
corridor



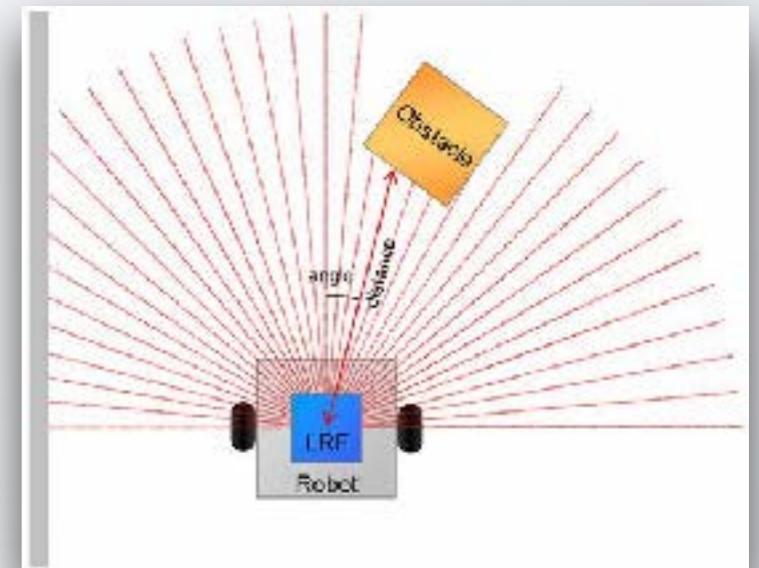
sonar map

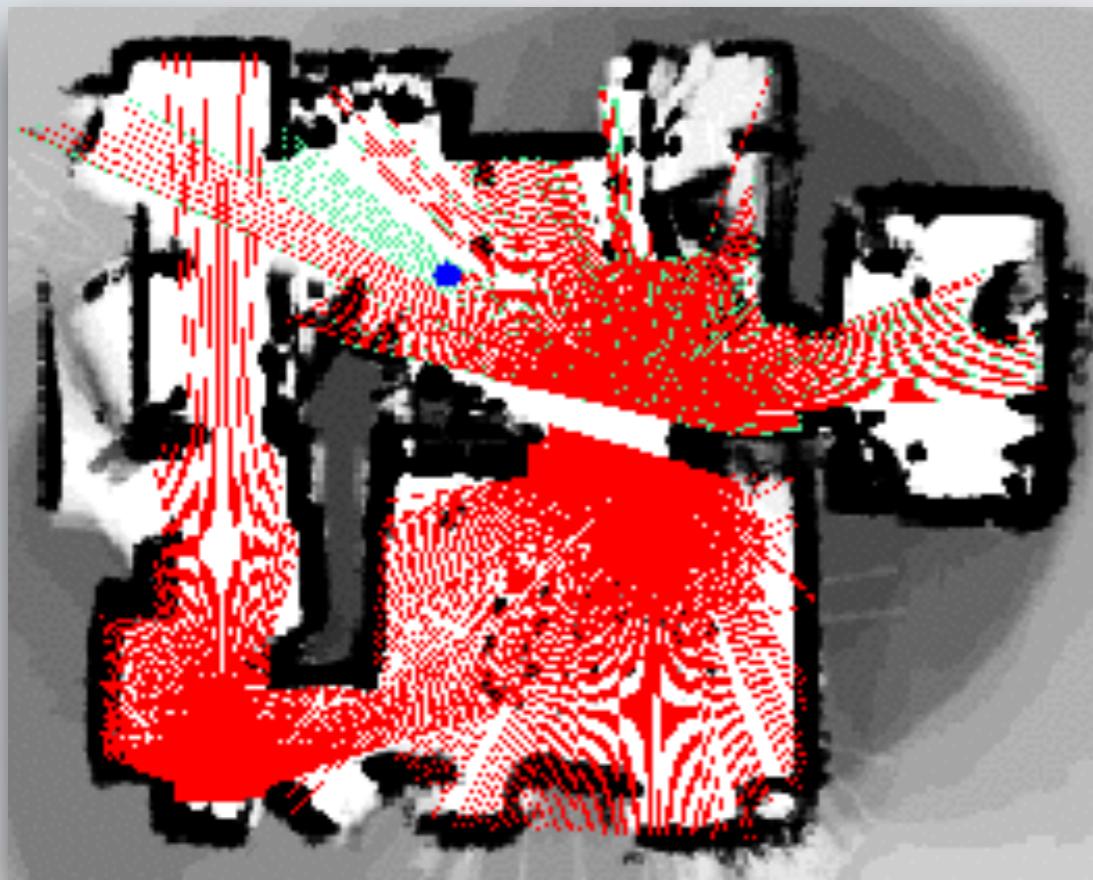
LASER SENSORS

- ▶ 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)





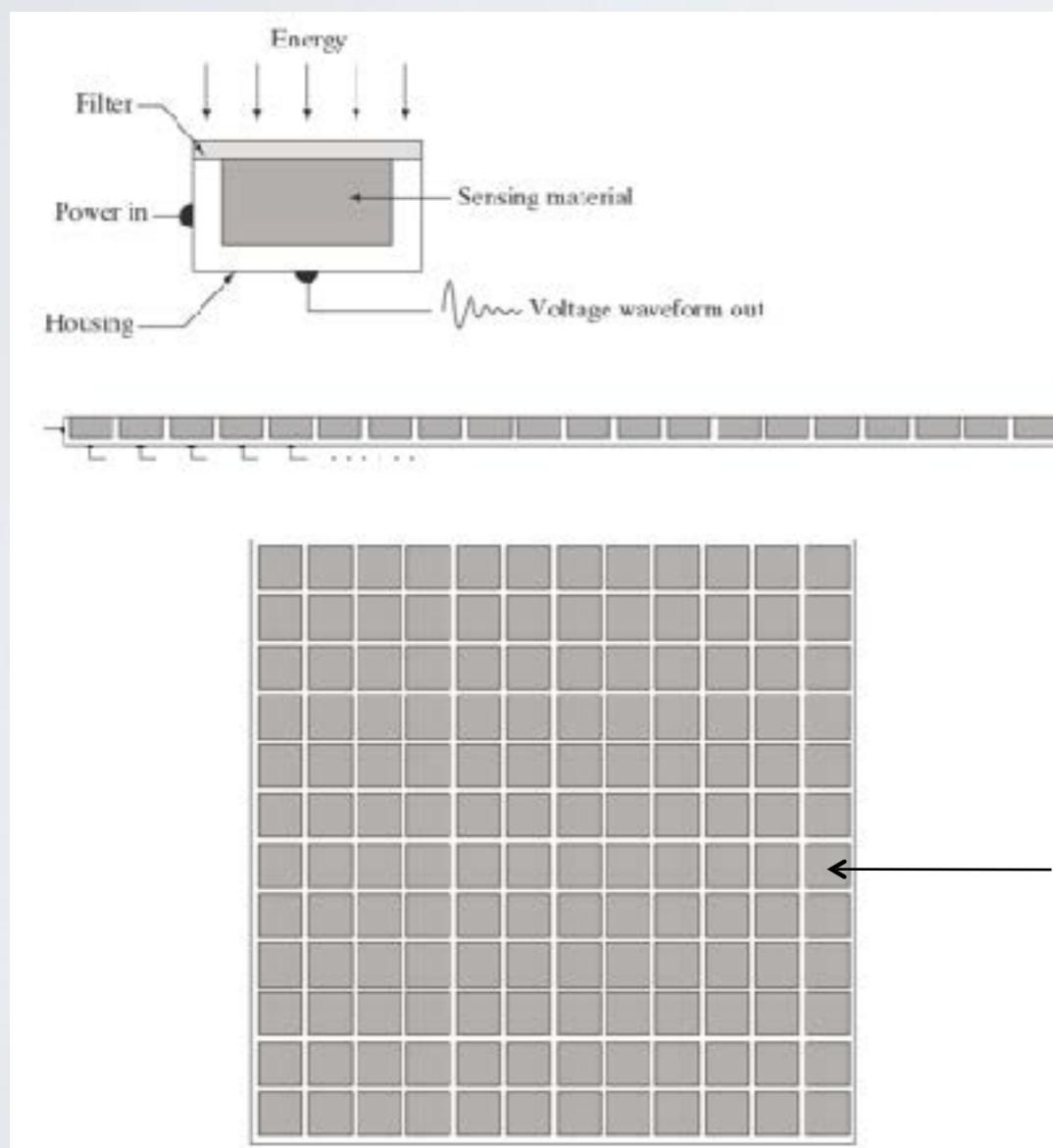
2d maps and people
detection



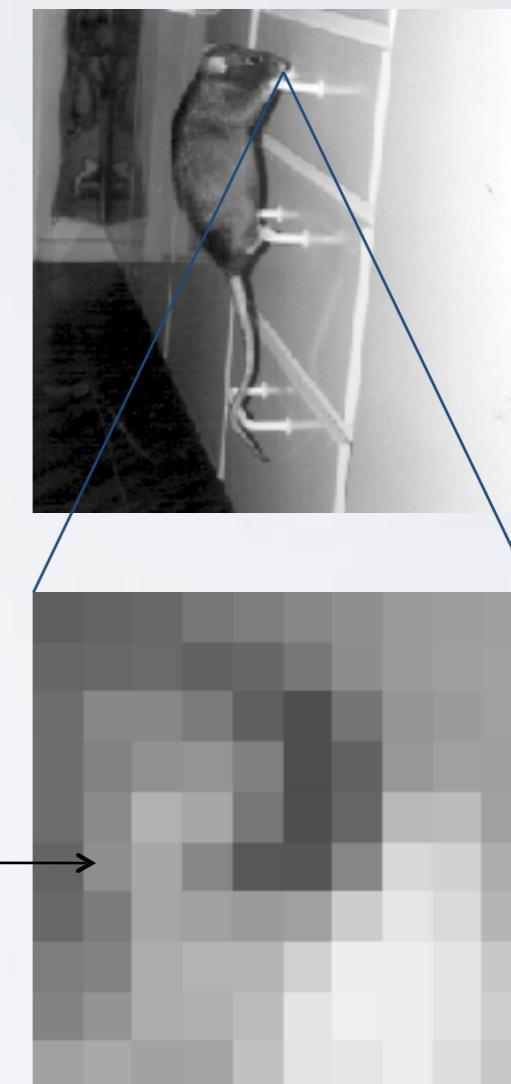
3d maps

VISION SENSOR

Vision Sensor



Digital Image



DIFFERENT TYPE OF VISION SENSORS

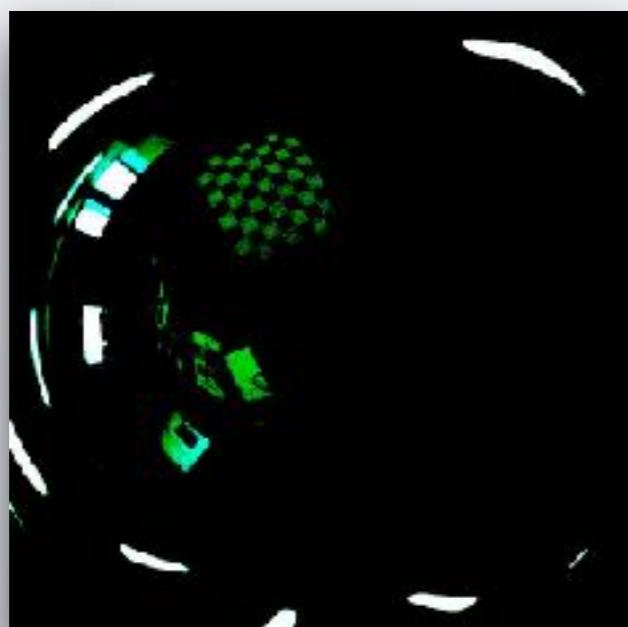
colour



thermal



omni-directional



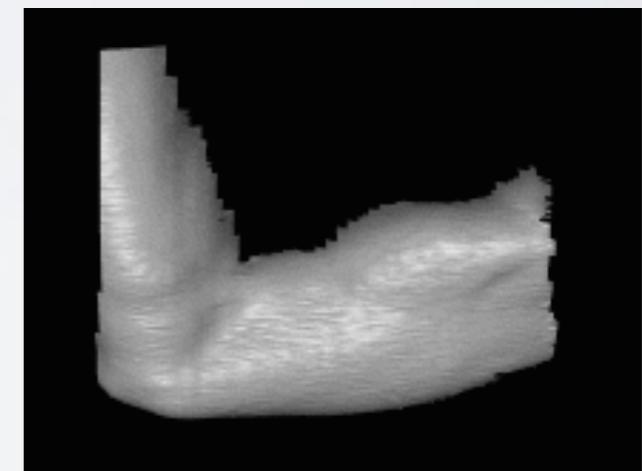
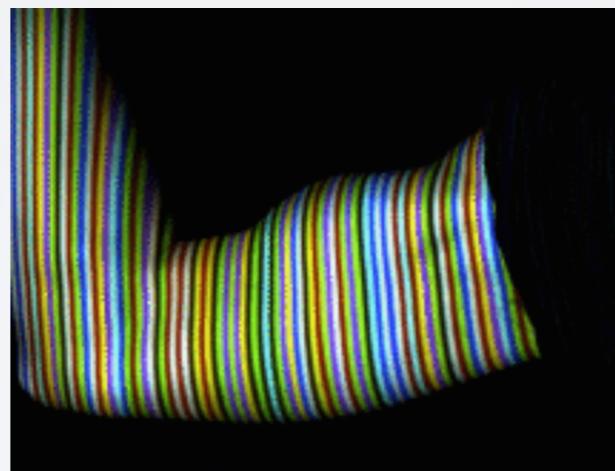
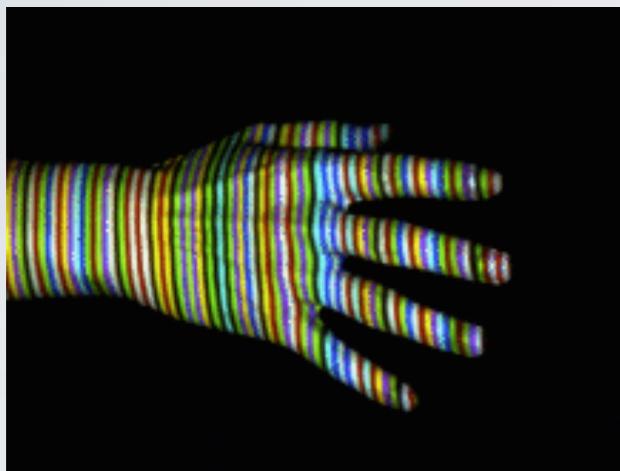
stereo



Kinect

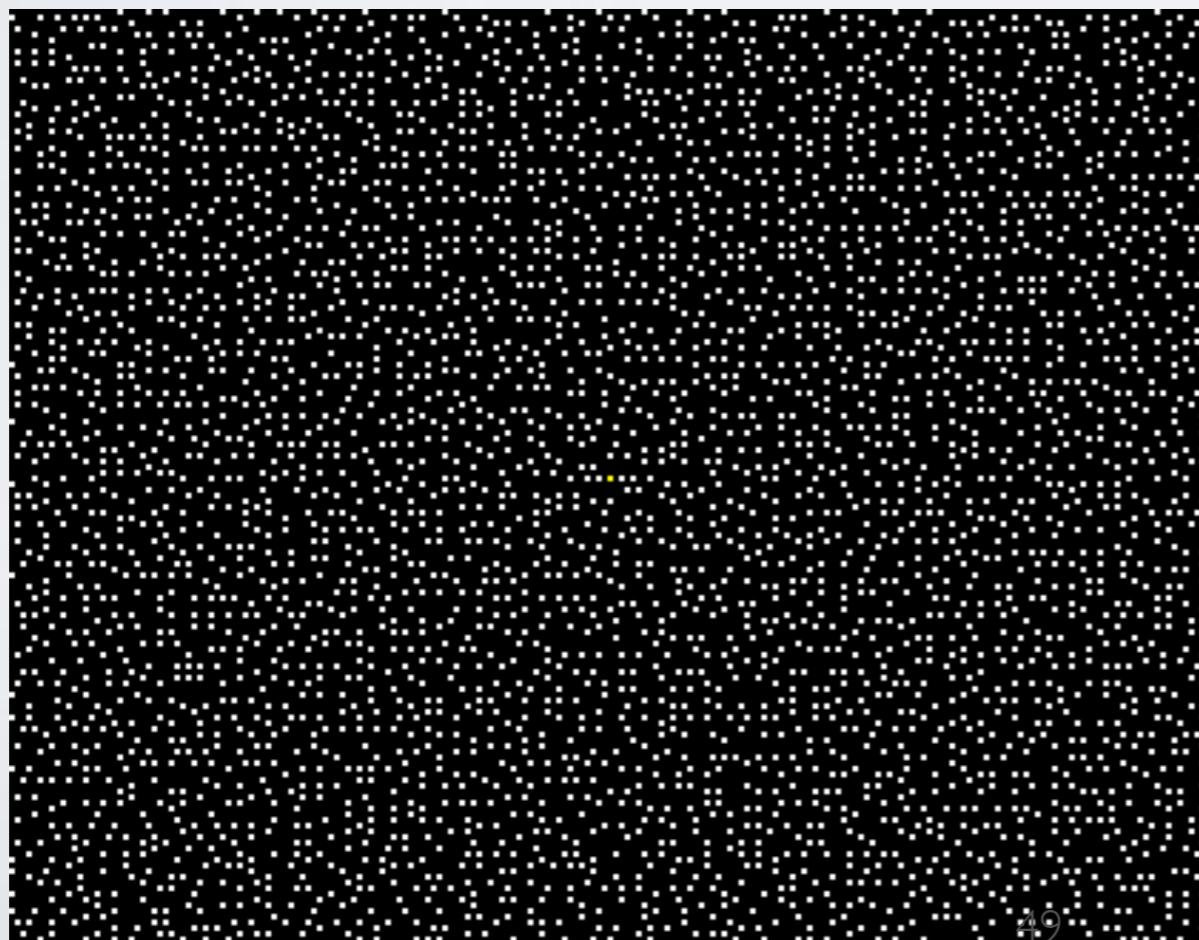
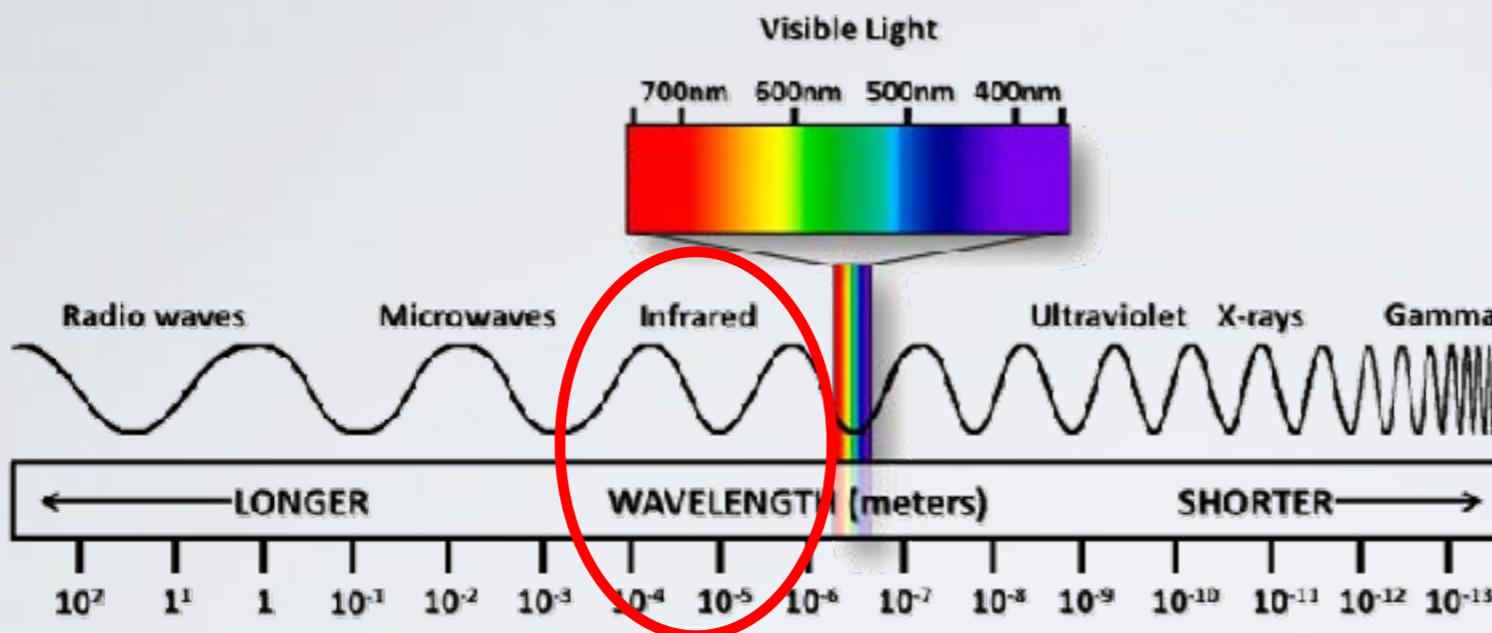


STRUCTURED LIGHT

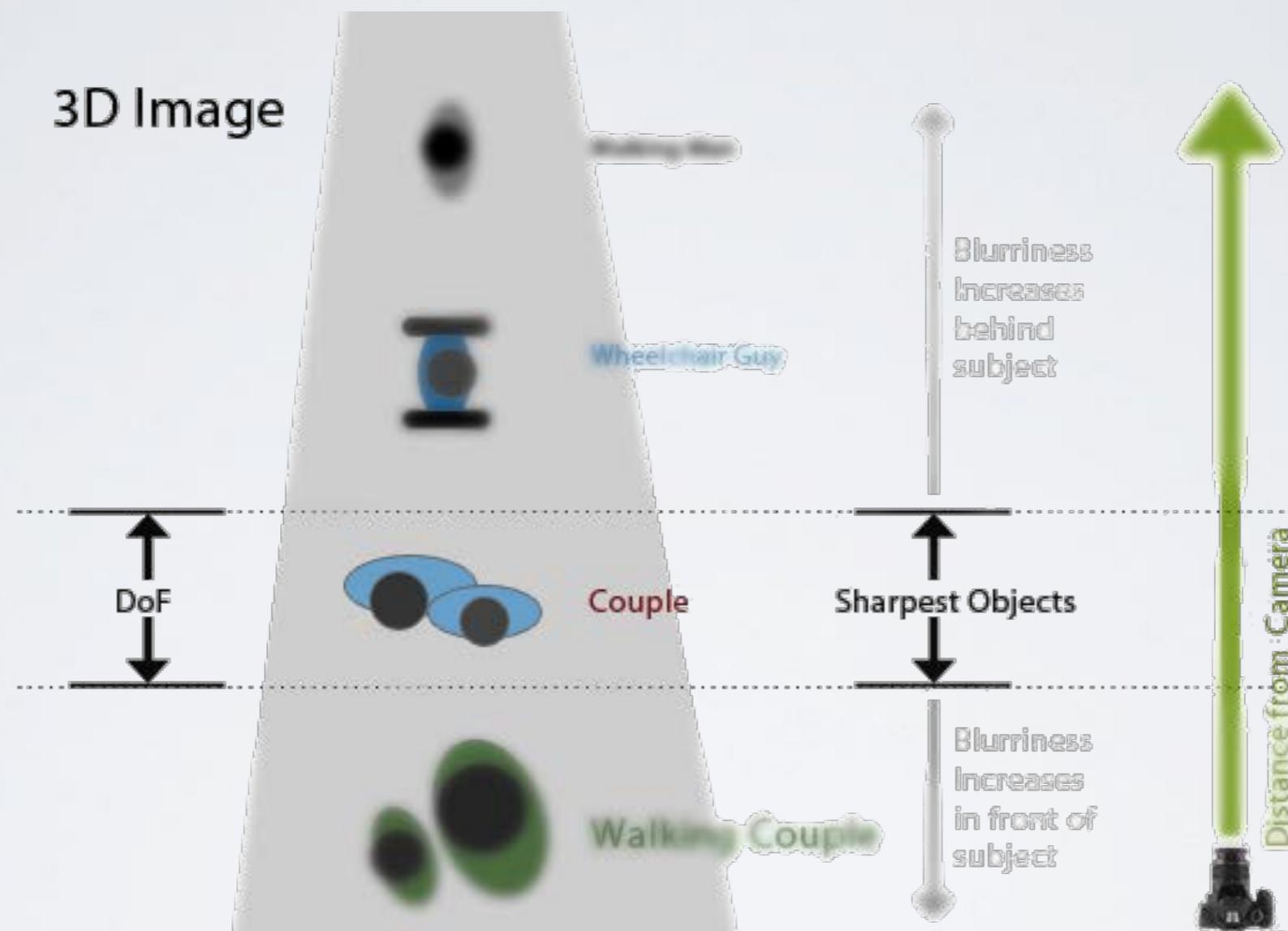


Zahng et al., 2002
<http://grail.cs.washington.edu/projects/moscan/>

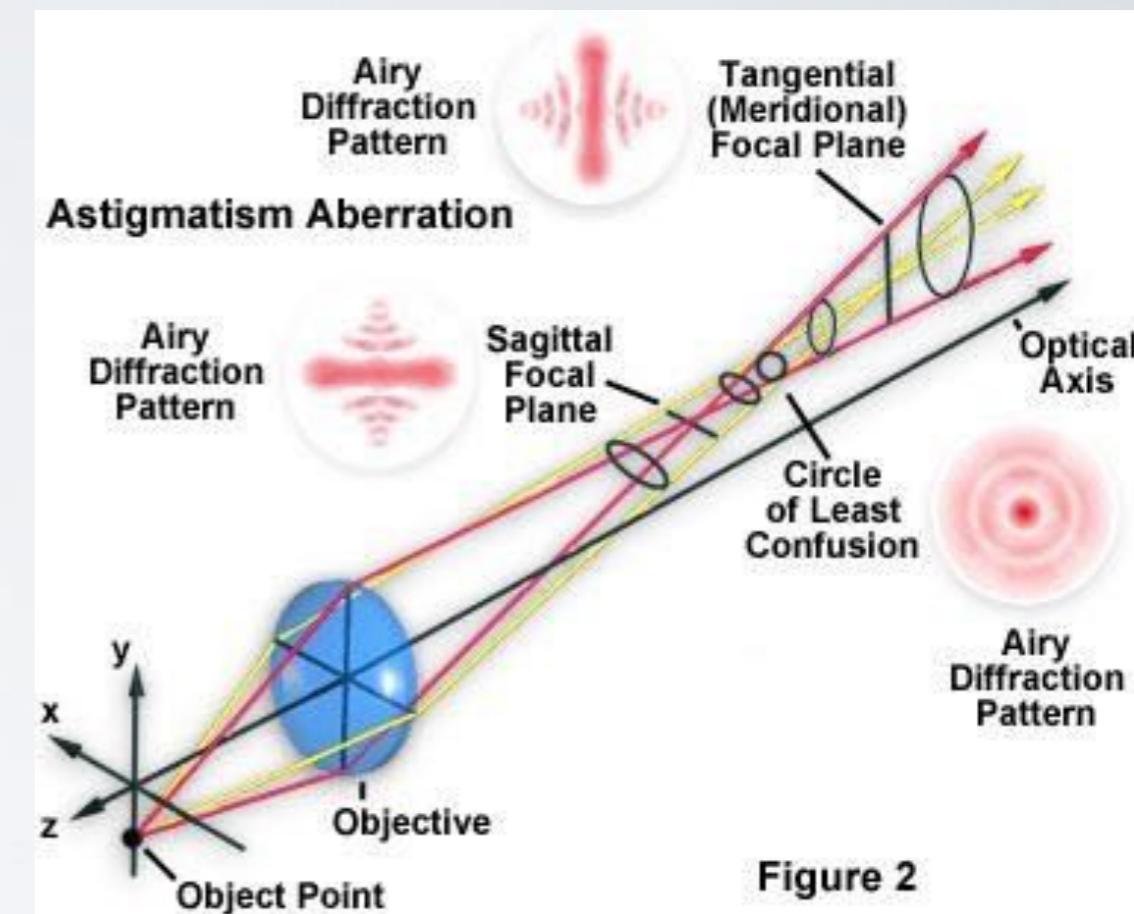
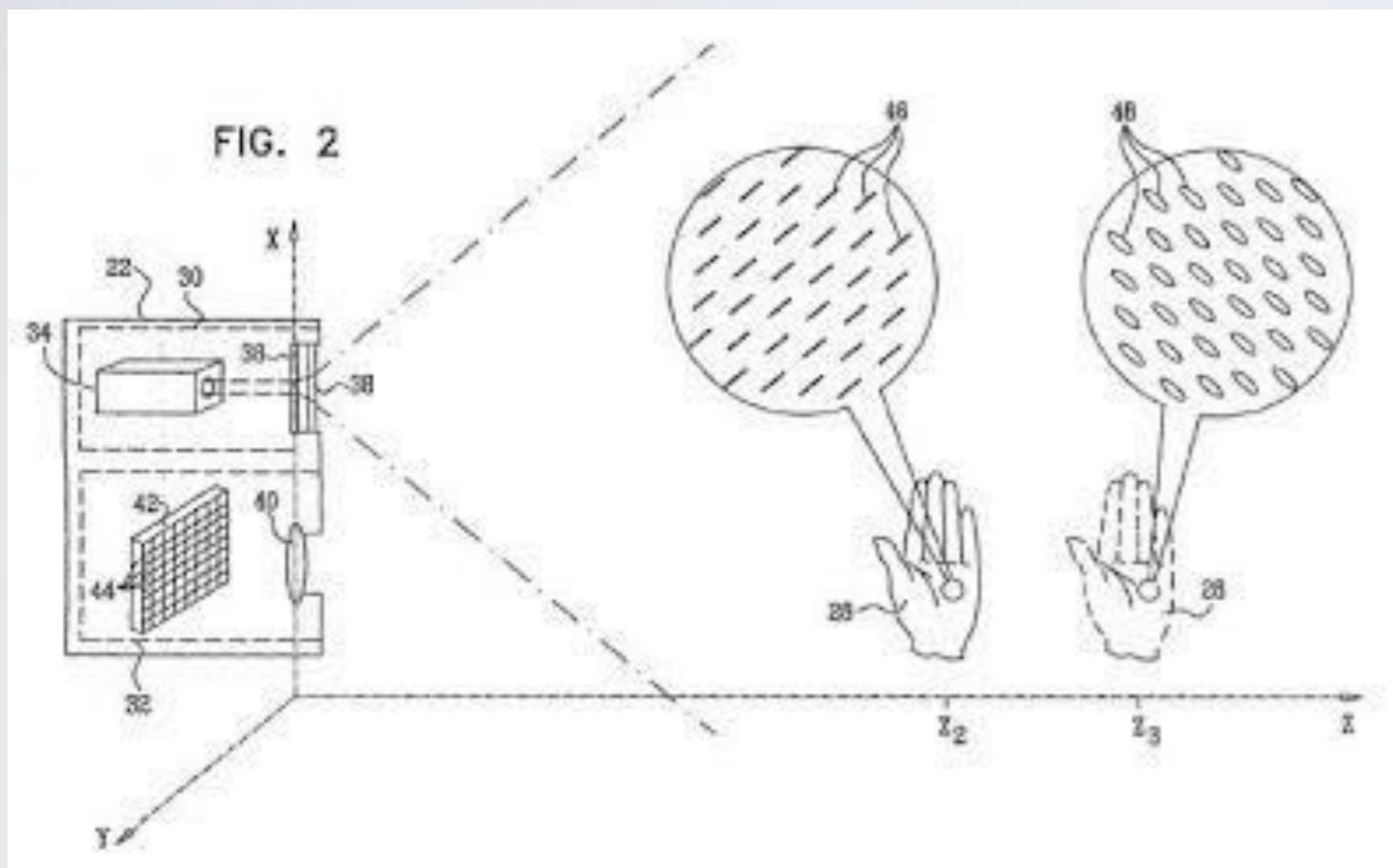
STRUCTURED LIGHT - KINECT



DEPTH FROM FOCUS

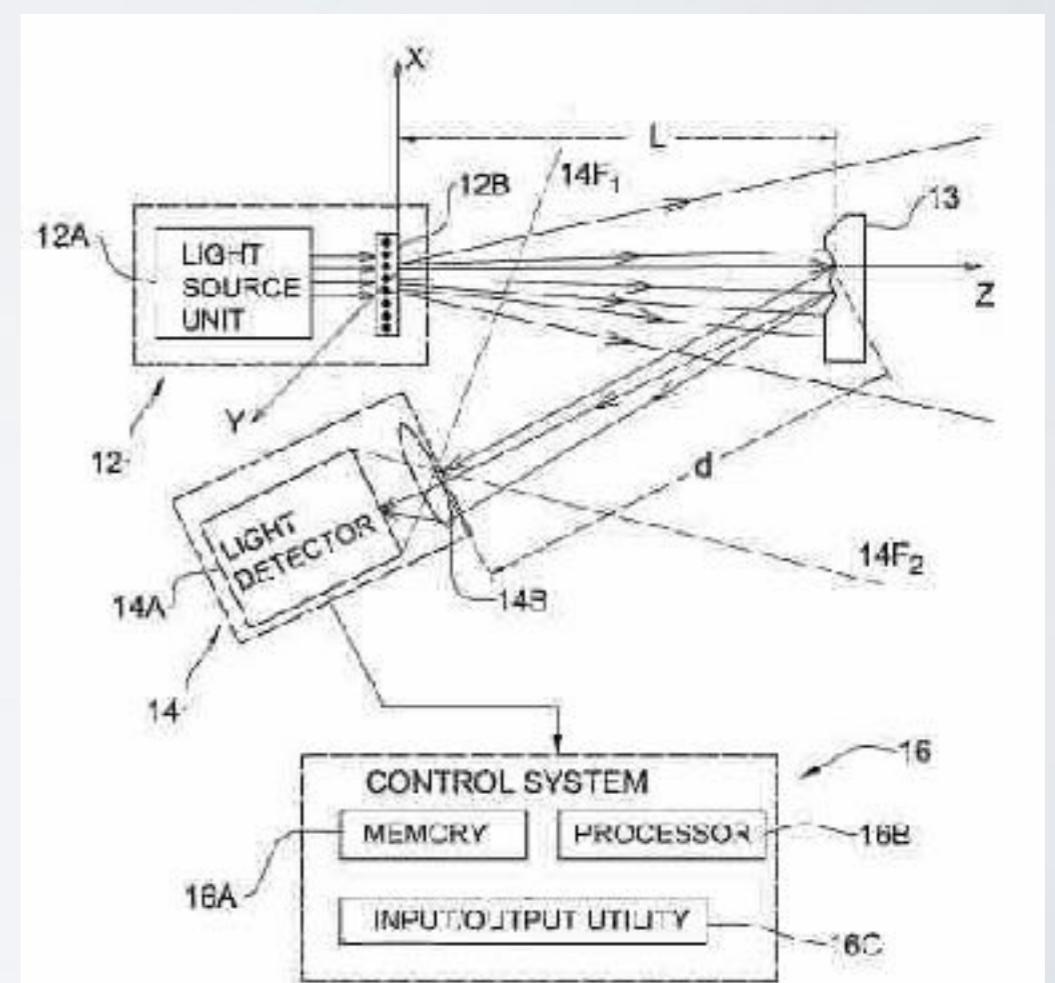
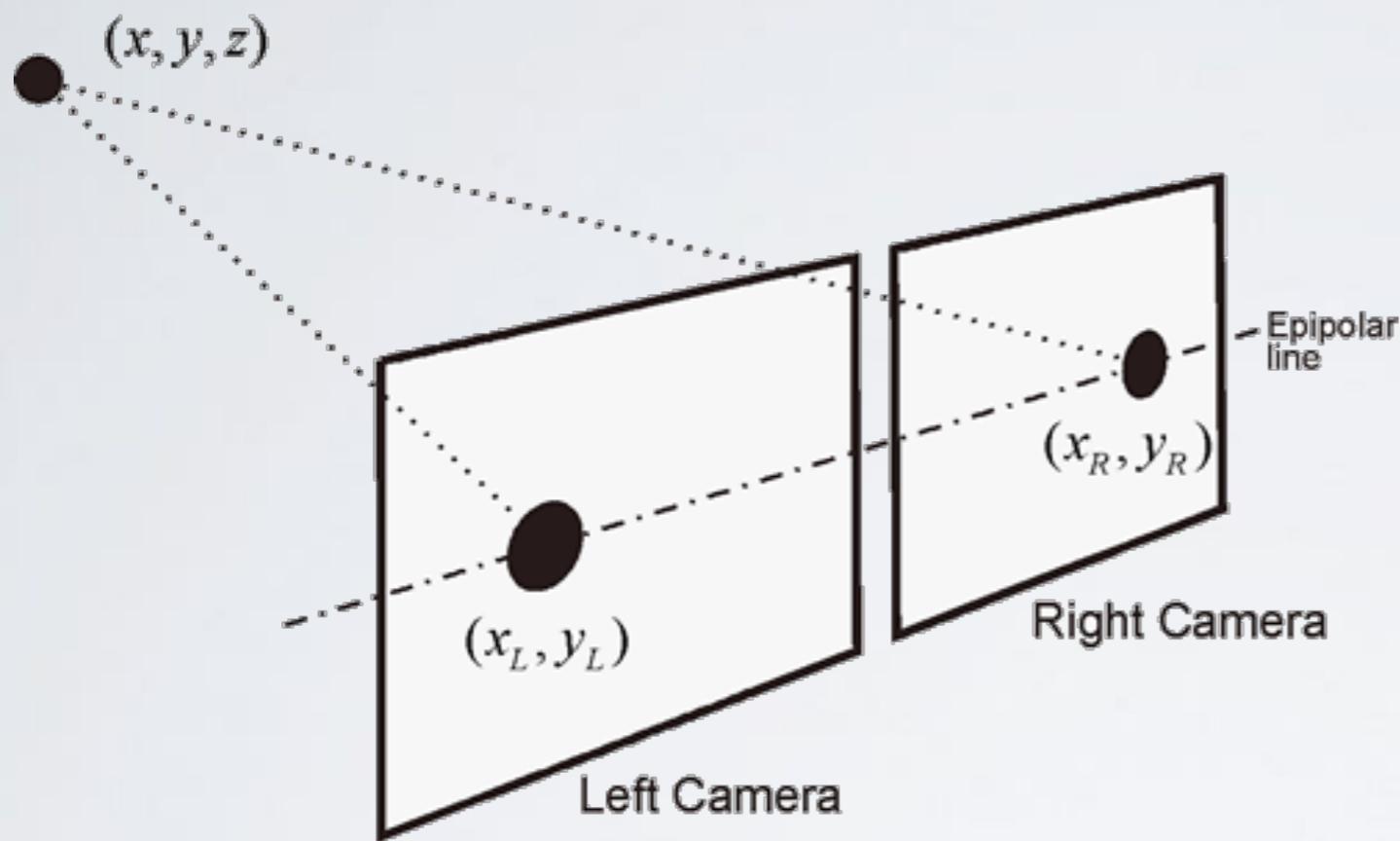


DEPTH FROM FOCUS - KINECT



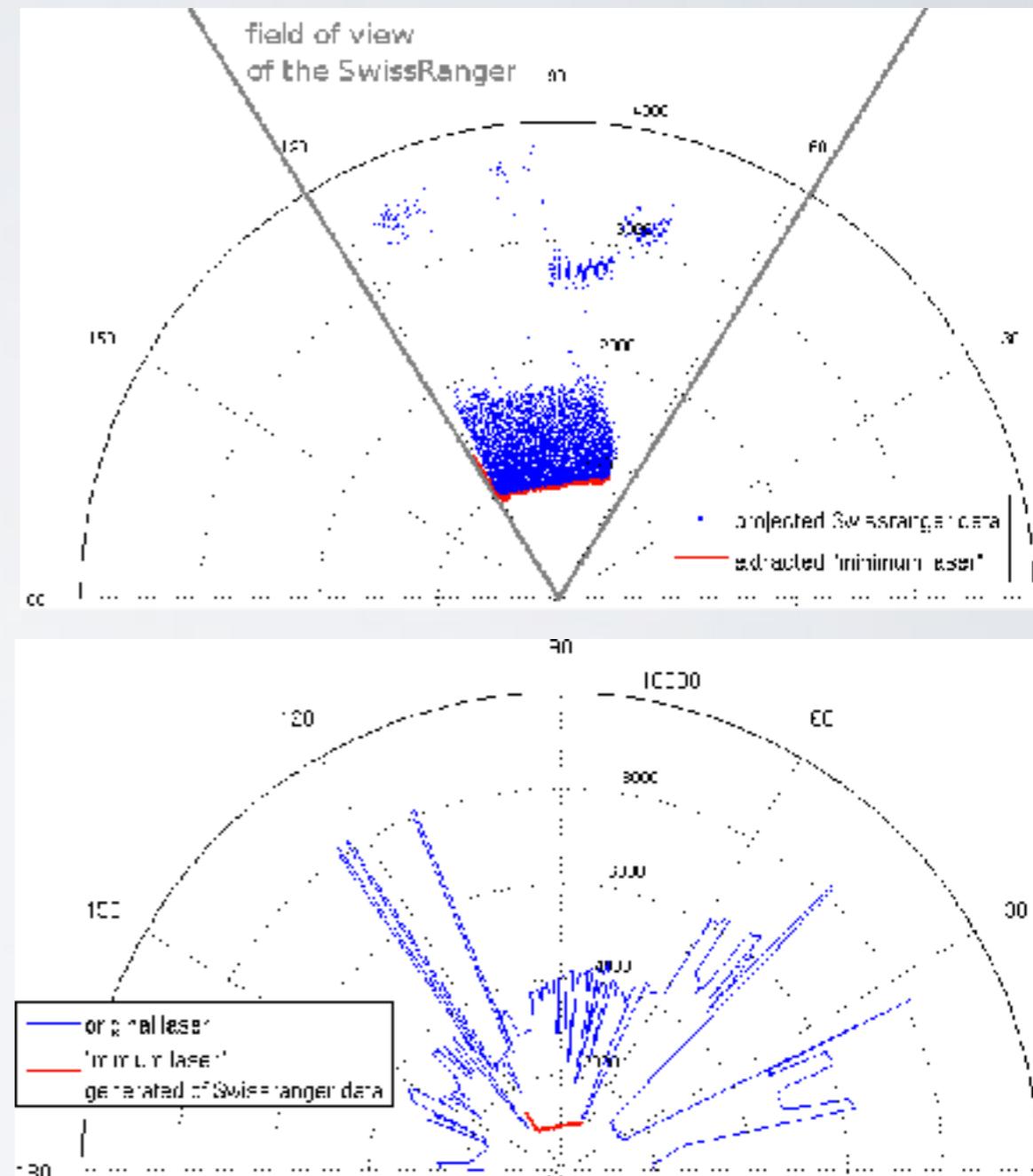
“astigmatic” lens with different focal length in x and y direction

DEPTH FROM STEREO

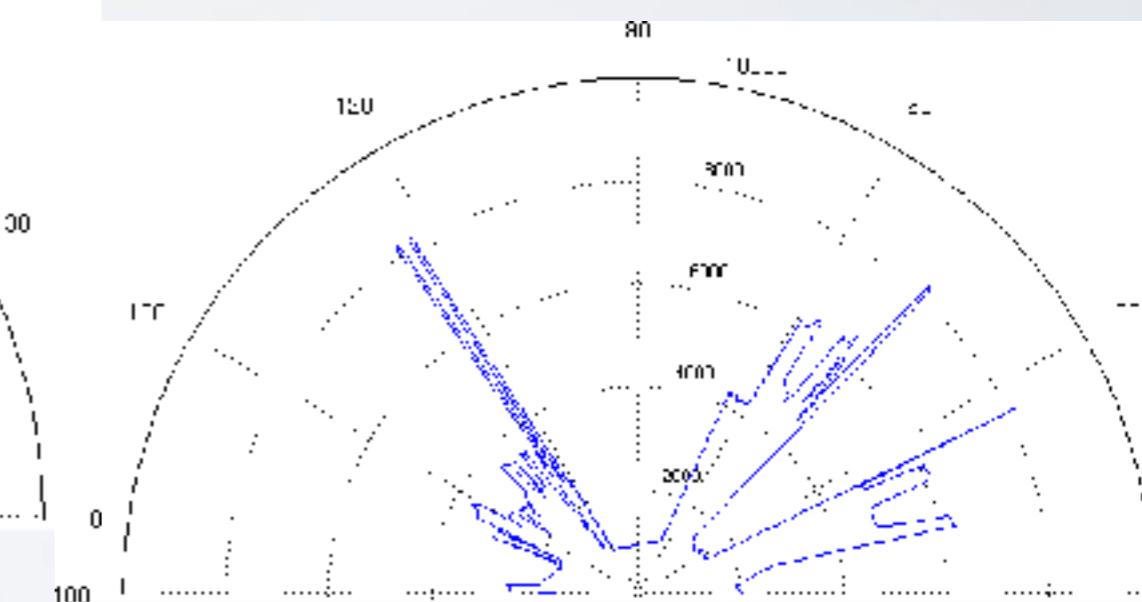
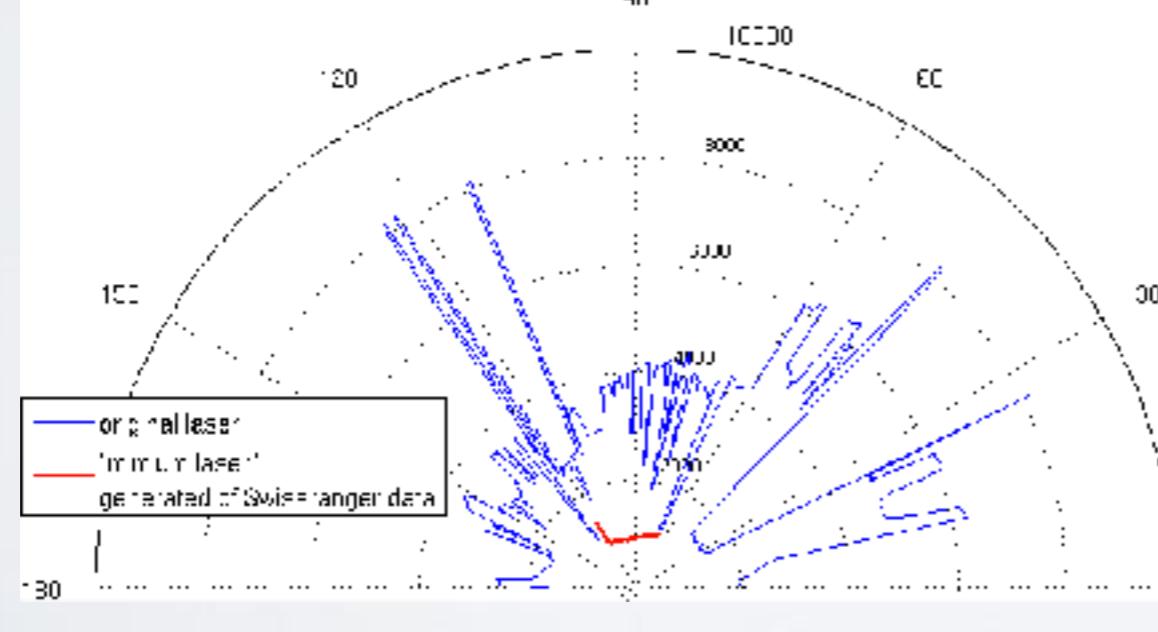
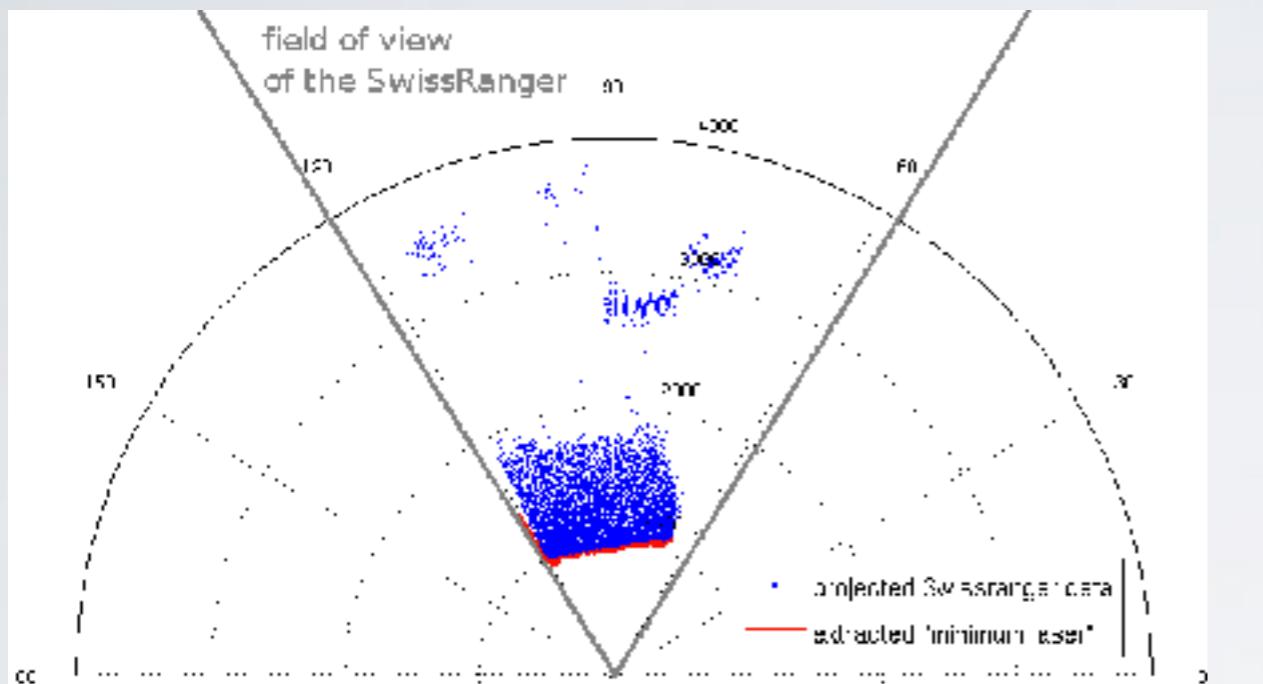


EMULATING A LASER SCANNER AND MERGING READINGS

- ▶ requires “calibration” of the two sensors (measuring where one is in relation to the other)
=> Rotation and Translation
- ▶ in 3D given by 6 degrees of freedom
- ▶ often represented as a 3×3 rotation matrix and a 3×1 translation vector

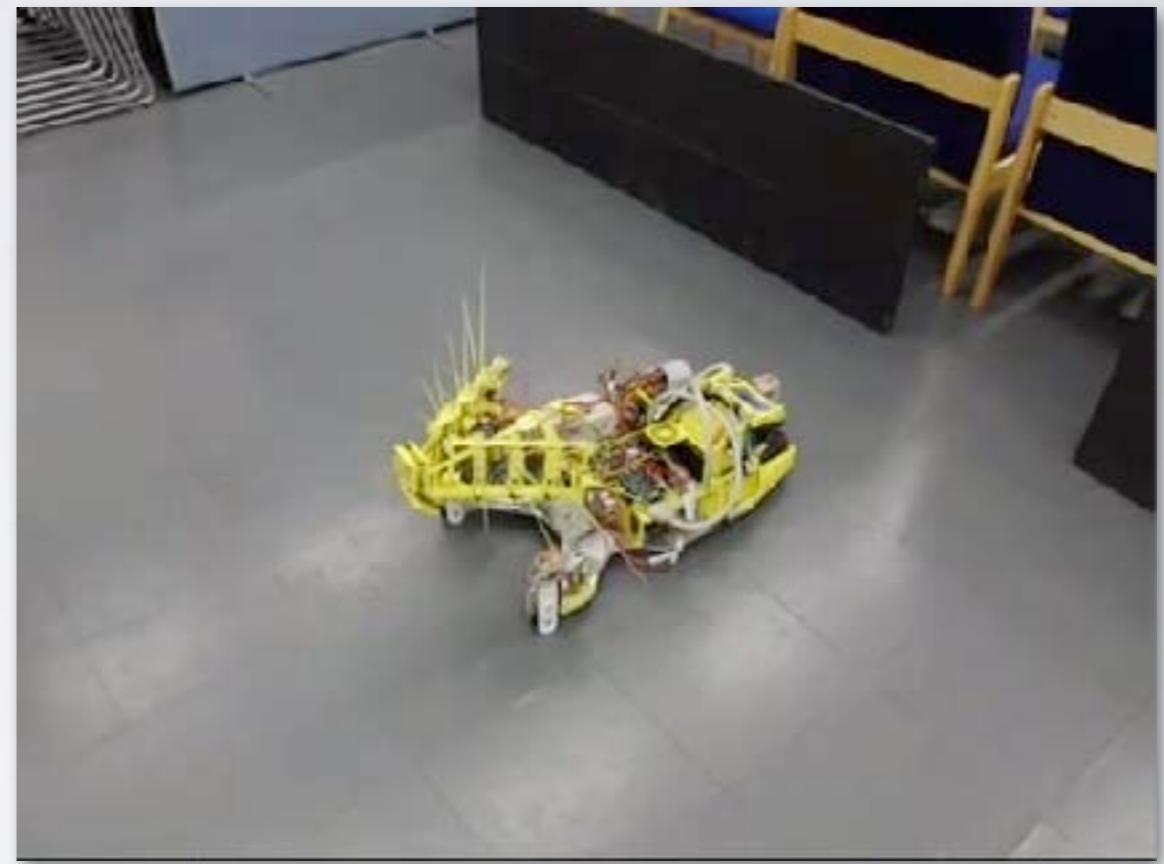


AVOIDING THE TABLE

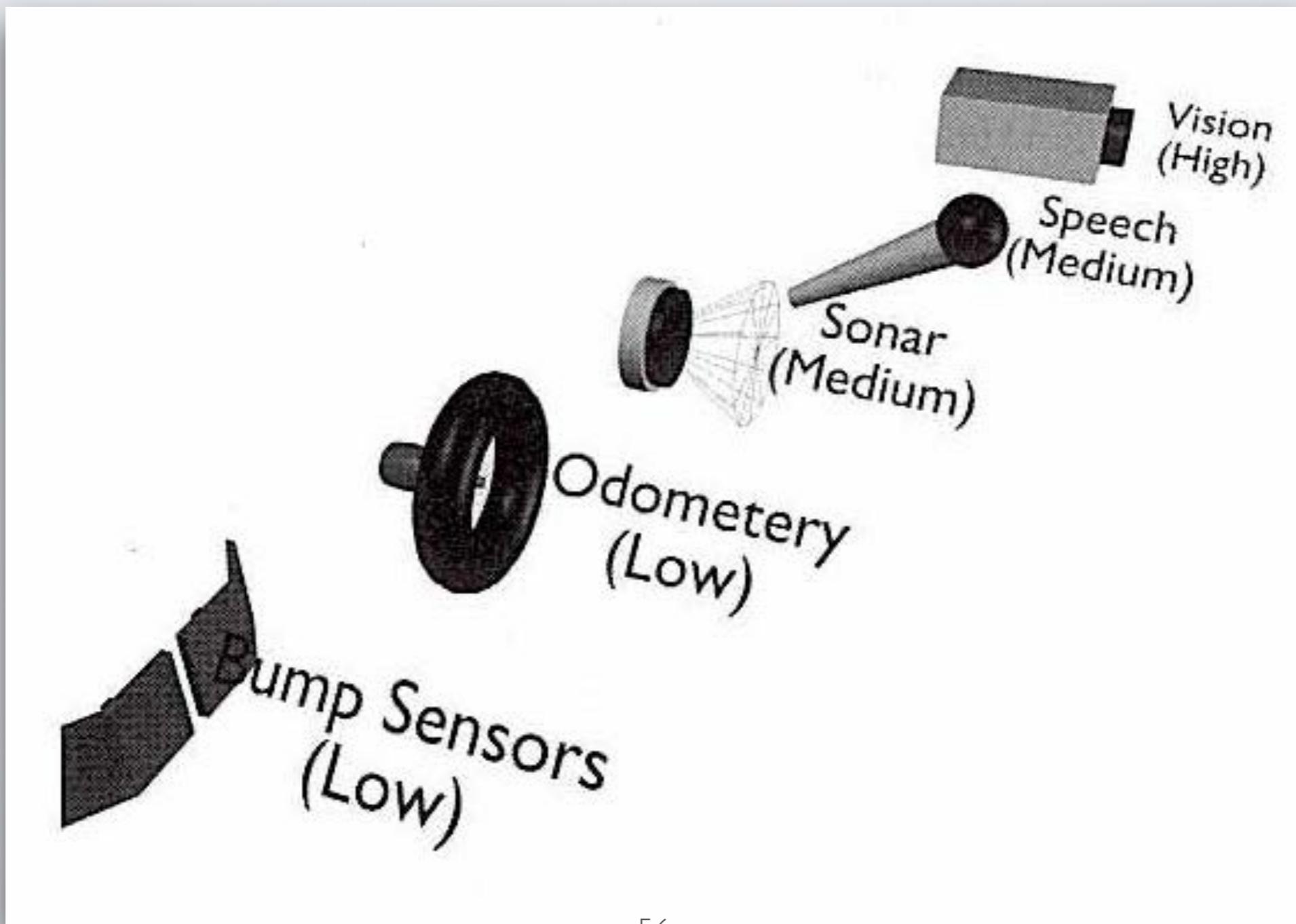


OTHER SENSORS

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



HIERARCHY OF PROCESSING REQUIREMENTS



PERCEPTION

- ▶ Data Interpretation and Processing
- ▶ sensors often 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!





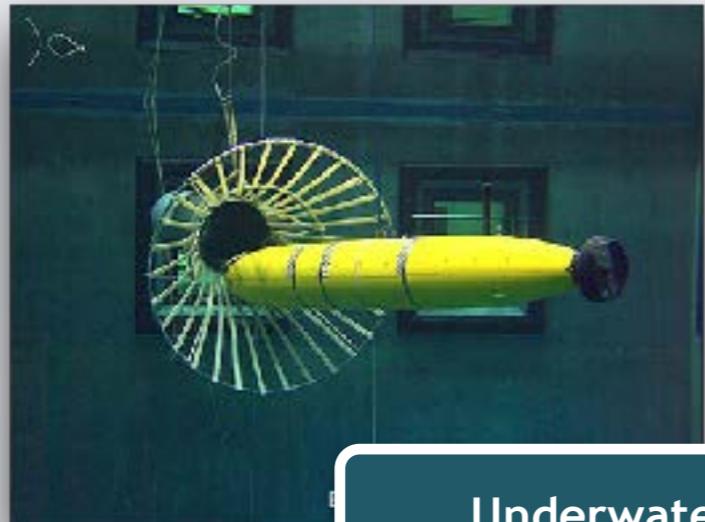
APPLICATIONS

Which robot is more autonomous?

Which one of these robots is more autonomous?



DIFFERENT ROBOT – DIFFERENT APPLICATION



Underwater

autonomous exploration



Tracked

inspection of radiated areas
(Pioneer)



Airborne

surveillance (MQ-1 Predator)



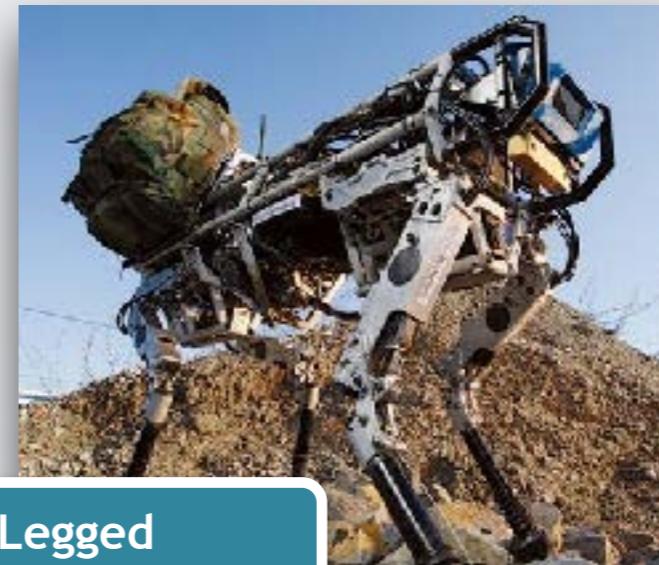
Tracked

gutter cleaner (Jool)

DIFFERENT ROBOT – DIFFERENT APPLICATION



Legged
exploration of volcanoes
(Sojourner)



Legged
transportation (BigDog)



Wheeled
space exploration



Wheeled
autonomous car (DARPA Grand
Challenge)

A ROBOT IN EVERY HOME

- ▶ Service Robots
 - ▶ cohabitant with humans in their everyday environments
- ▶ Bill Gates predicts (in 2006):
 - ▶ “in 20 years (2026) there will be a robot in every home”
 - ▶ read his [article](#) in Scientific American

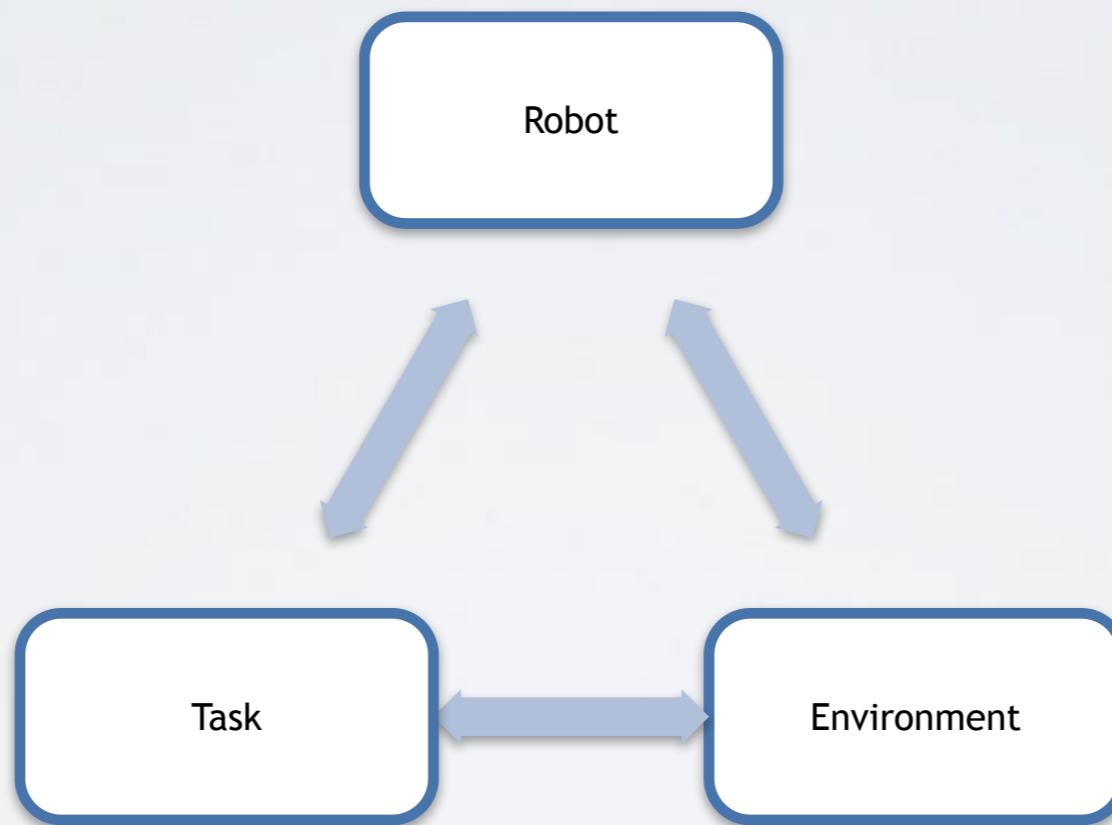


Overview/*The Robotic Future*

- The robotics industry faces many of the same challenges that the personal computer business faced 30 years ago. Because of a lack of common standards and platforms, designers usually have to start from scratch when building their machines.
- Another challenge is enabling robots to quickly sense and react to their environments. Recent decreases in the cost of processing power and sensors are allowing researchers to tackle these problems.
- Robot builders can also take advantage of new software tools that make it easier to write programs that work with different kinds of hardware. Networks of wireless robots can tap into the power of desktop PCs to handle tasks such as visual recognition and navigation.

ROBOT, TASK, ENVIRONMENT

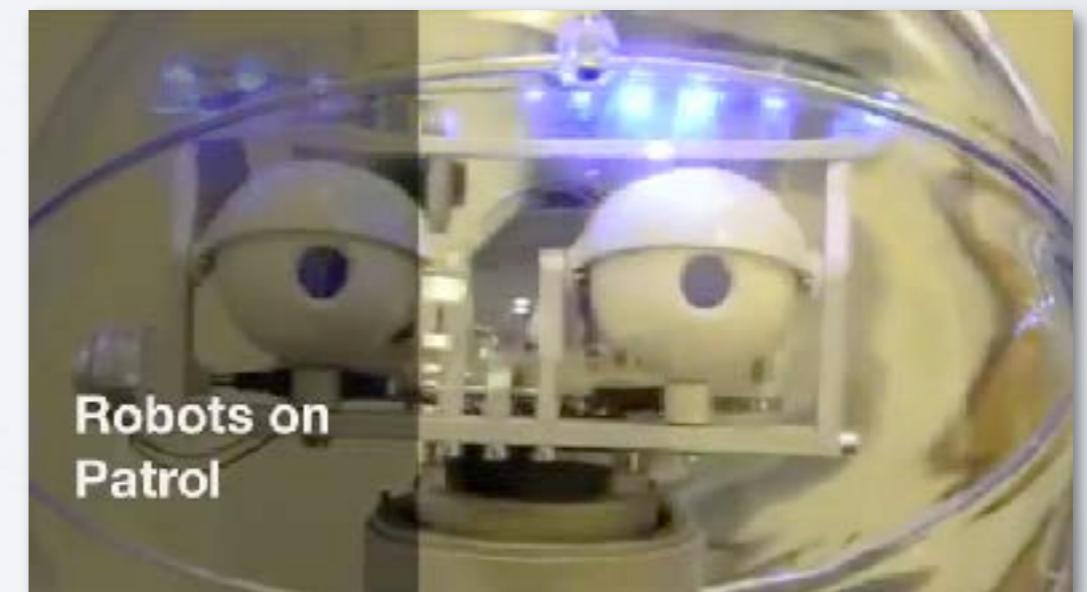
- ▶ There is no general purpose robot



- ▶ These three aspects are dependent upon and influence each other

ROBOTICS RESEARCH AT LINCOLN

- ▶ Our webpage: 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



[FP7 Project STRANDS](http://fp7-strands.eu)

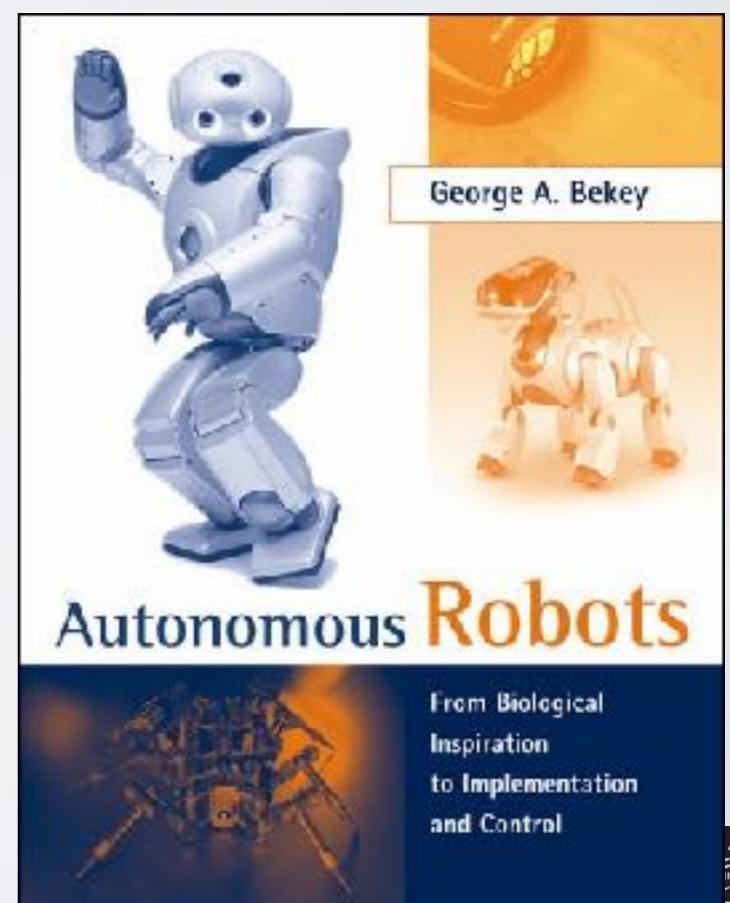
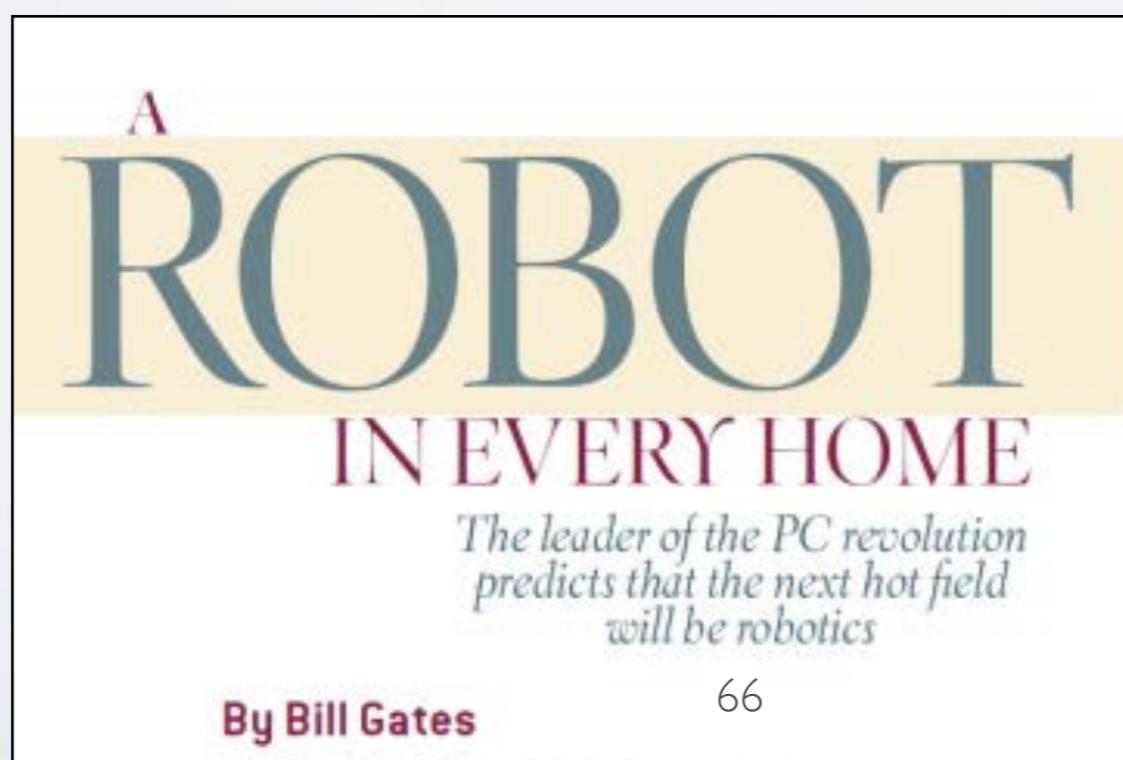
ROBOTICS RESEARCH AT LINCOLN

- ▶ Our webpage: 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

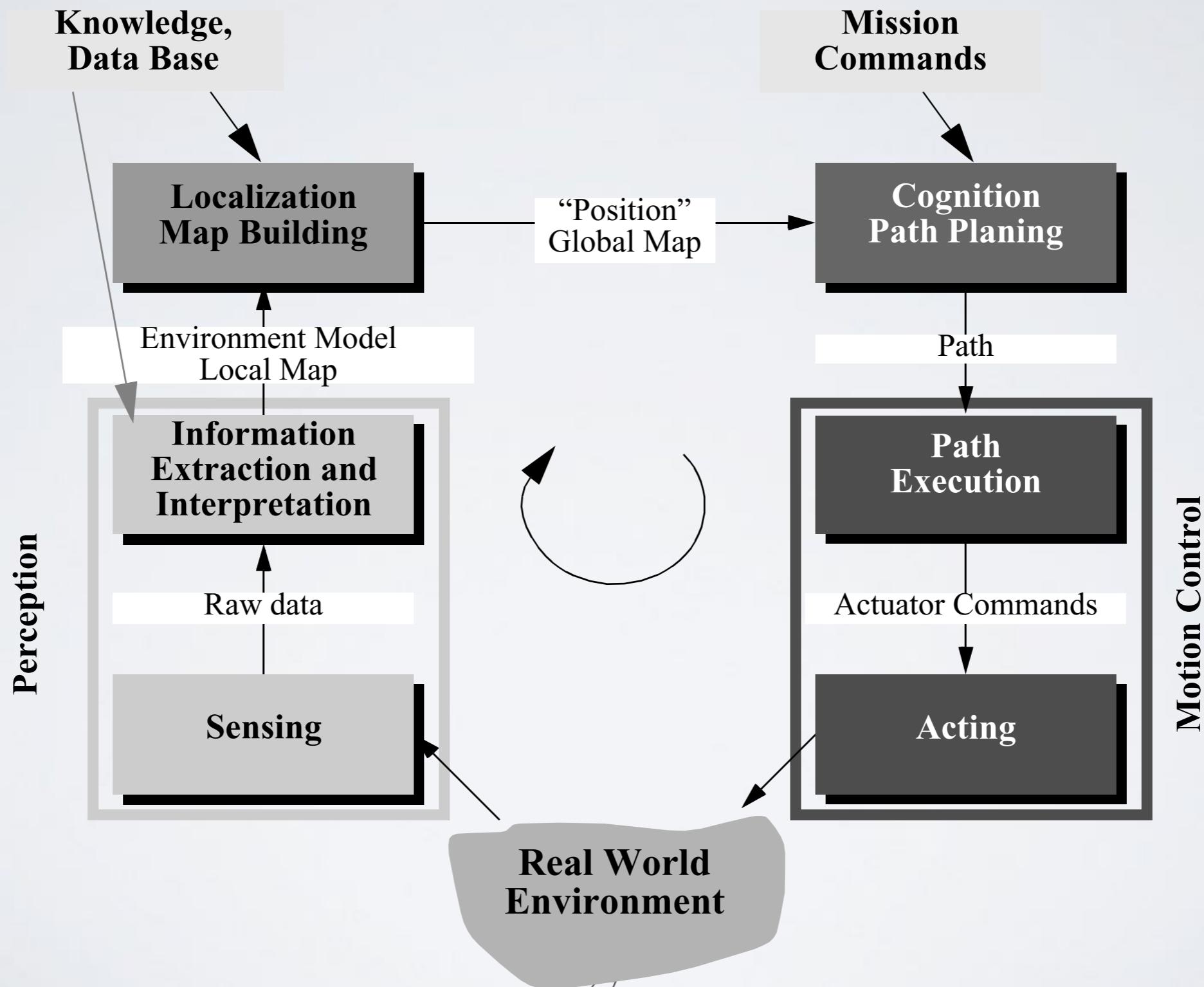


RECOMMENDED READING

- ▶ 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





LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

Thank you for listening!
Any questions ?



CMP3103M AUTONOMOUS MOBILE ROBOTS

Prof. Marc Hanheide

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



UNIVERSITY OF
LINCOLN



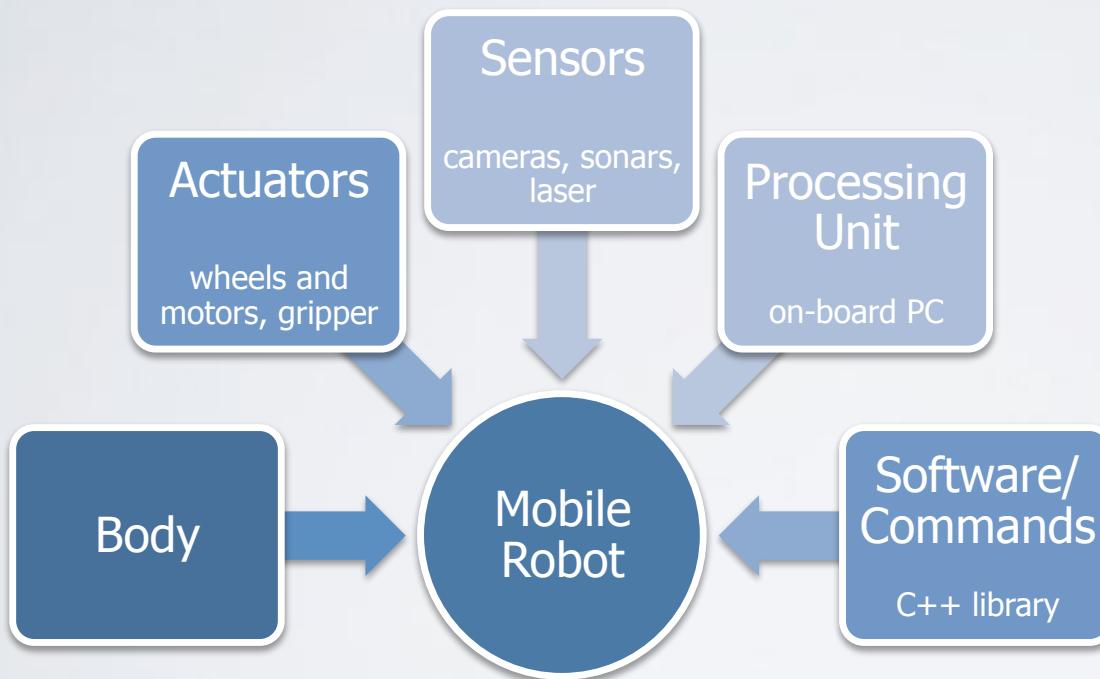
LINCOLN
ROBOTICS

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

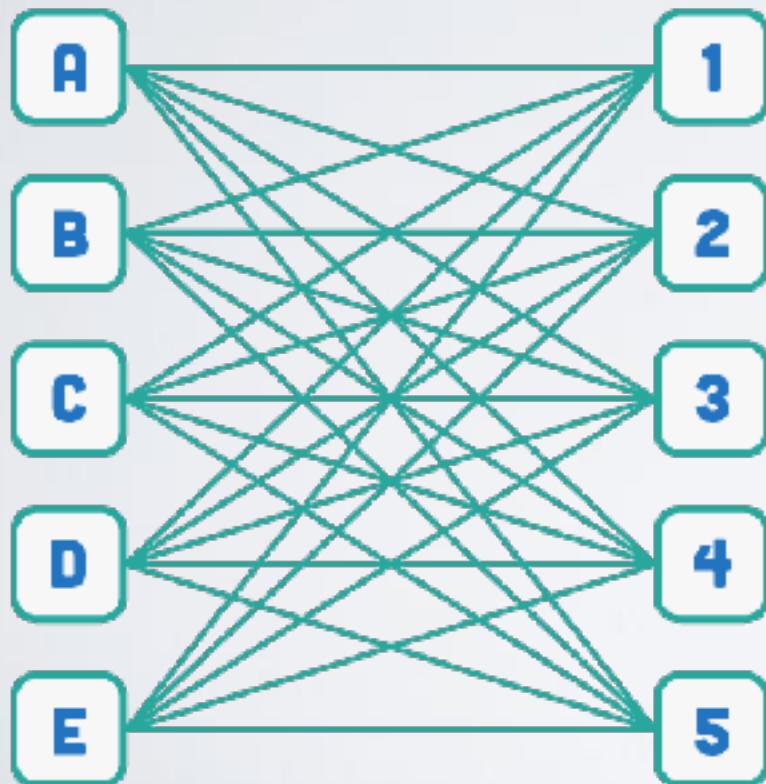
- ▶ Hardware **drivers**, (real-time) operating system
- ▶ Audio/video encoders
- ▶ Command interface, **firmware**
- ▶ Sensor/Image processing **library**
- ▶ Software **components** implementing AI, navigation, decision making
- ▶ **Simulator**

HOW TO TALK TO A ROBOT

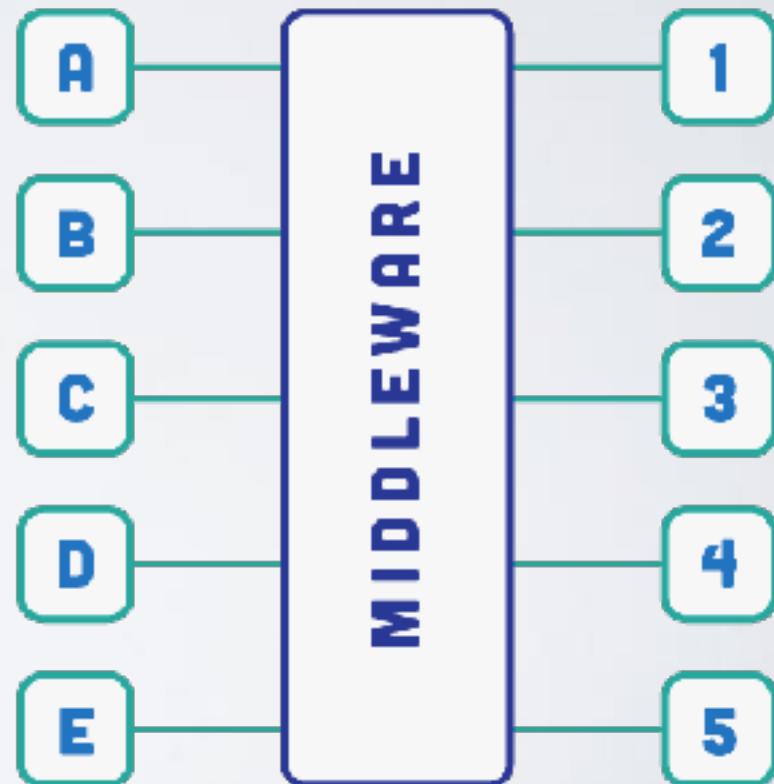
- ▶ We need to be able to:
 - ▶ send motor/actuator commands
 - ▶ read and process data from sensors
- ▶ in some robots there is an abstraction layer featuring a shared **domain-specific command language** to access all sensors and actuators (highly integrated)
 - ▶ implemented inside the robot
 - ▶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

MIDDLEWARE?

WITHOUT MIDDLEWARE

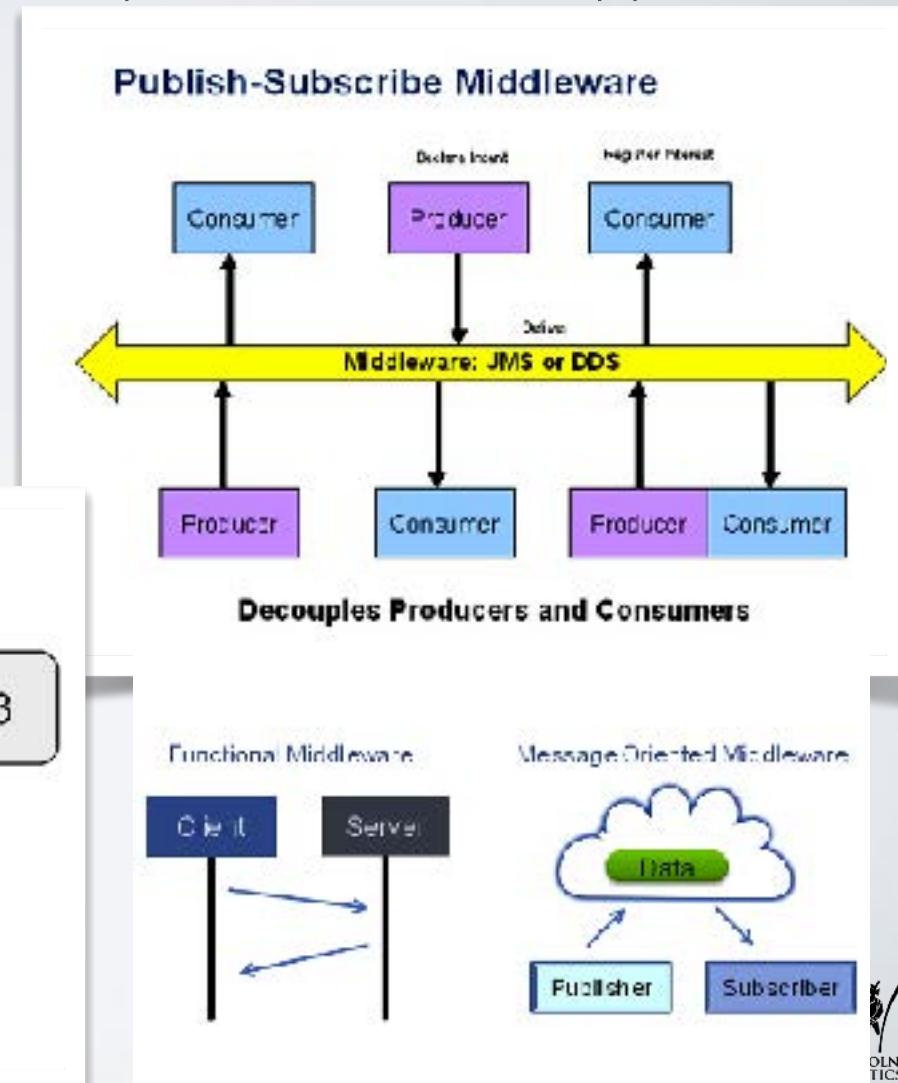
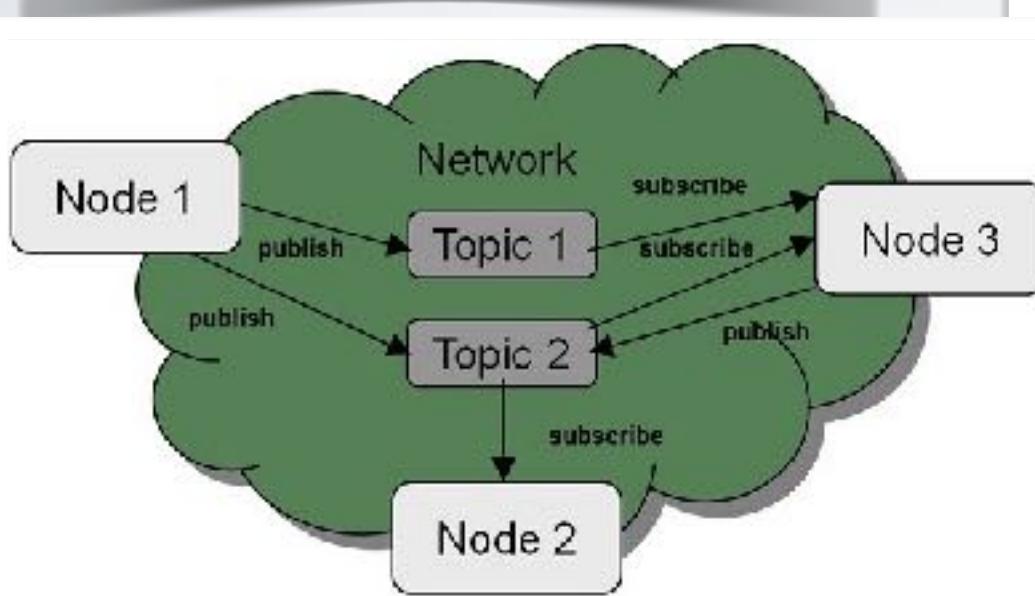


WITH MIDDLEWARE



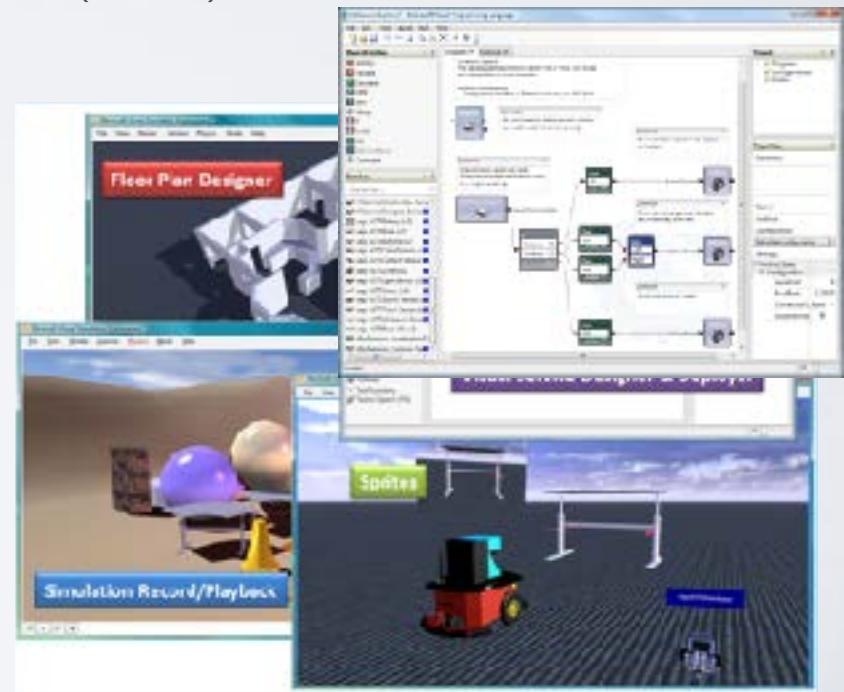
MIDDLEWARE?

Middleware supports and simplifies complex distributed applications.



ROBOTIC MIDDLEWARES

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

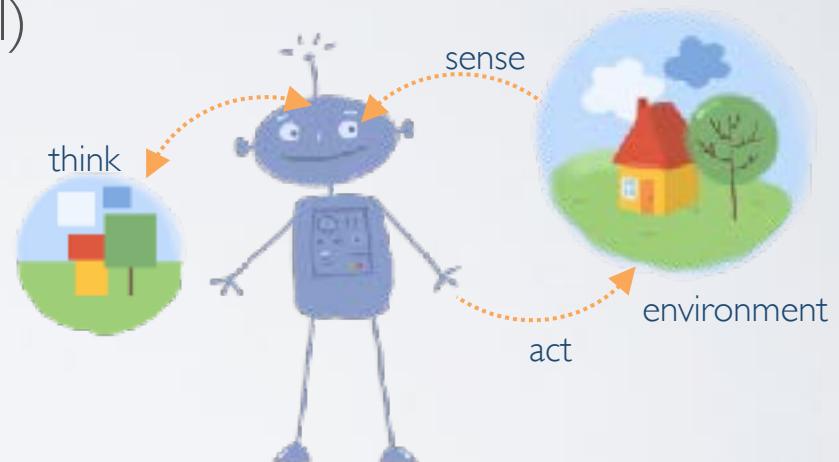


ROS

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

IMPLEMENTING TASKS/ BEHAVIOURS

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



SENSE - (THINK) - ACT

- ▶ Two Options

data pull

- ▶ Synchronous:

- ▶ like a while loop:

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

Easier

- ▶ Asynchronous:

data callbacks

- ▶ different threads with shared (and synchronised) memory access

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
- ▶ Another OMG standard: DDS (middleware for **ROS2**)



STOLEN ADOPTED INTRO ABOUT ROS

Introduction to ROS

Pierrick Koch, Séverin Lemaignan
based on slides by Thomas Moulard

LAAS robotics courses, January 2013

ROS.org

Using ROS

Jeremiah Via

23 February 2011

How Robotics
Research Keeps...

Re-Inventing the Wheel

First, someone
publishes...



...and they write
code that barely
works but lets
them publish...



...a paper with
a proof-of-
concept robot.



This prompts
another lab to
try to build on
this result...



But inevitably,
time runs out...



...and countless
sleepless nights
are spent
writing code
from scratch.



...but they can't
get any details
or the software
used to make it
work...



So, a grandiose
plan is formed
to write a new
software API...



...and all the
code used by
previous lab
members is a mess.

ROS.org

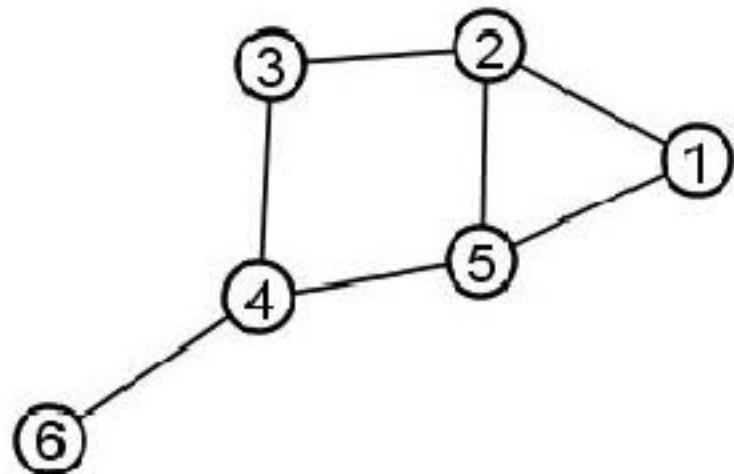
- Robot Operating System
- Meta-operating System
- Originally developed in 2007 at Stanford AI Lab
- Now developed at Willow Garage

So what is ROS?

- A component-oriented robotics framework,
- An Inter Process Communication middleware,
- A development suite,
- A (bad) package management system,
- An (active) community.

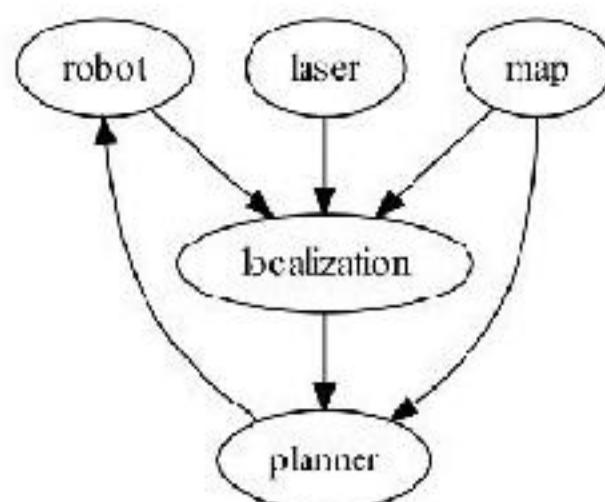
Node

- A process that performs computation



Peer-to-Peer

- Designed to have many nodes running
- Communication is managed by the master node



<http://www.ros.org/wiki/ROS/Tutorials/UnderstandingNodes>

The ROS framework is **component oriented**.

Each component is called a **node**. **Nodes** communicate using **topics** or **services**.

topics represents datastreams. For instance: camera images, robot joint configuration or position can be modelled as topics.

Topics values are typically published at regular rate to keep the whole system up-to-date.

services represents requests which are sent asynchronously, and usually at lower rate. Slower components such as a motion planning node will typically provide services.

Topic

- String which labels a stream of data, e.g., cmd_vel or scan
- Nodes can subscribe and publish to these topics



ROS

- ▶ roscore
- ▶ rosnode list
- ▶ rostopic list
- ▶ rosservice list
- ▶ rqt_graph

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

Order the processing steps when a message is to be send to a subscriber

Publisher provides socket descriptor where topic is streamed

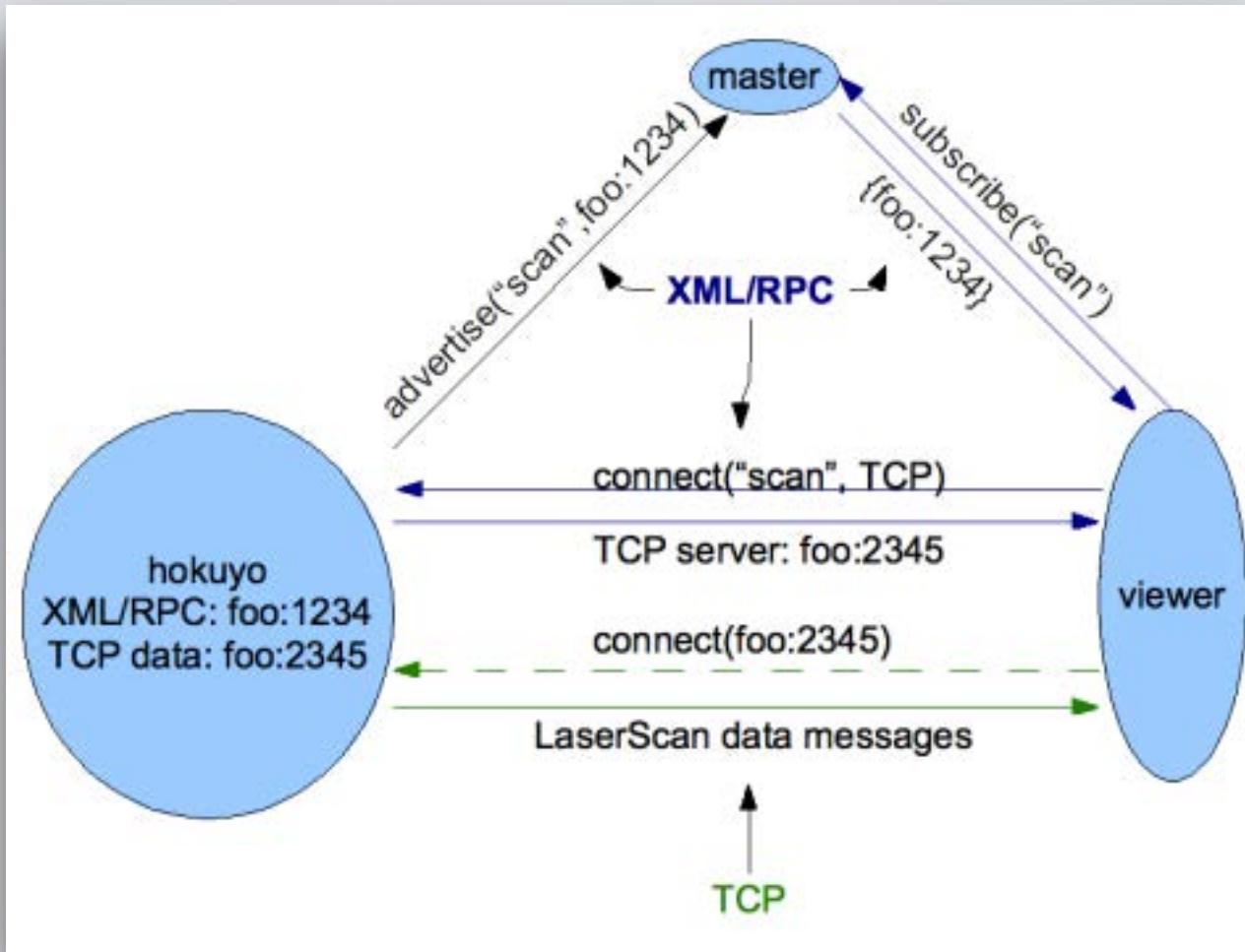
Subscriber connects to socket to receive data

Subscriber asks master for node(s) providing topics

Subscriber contacts Publisher (directly) with subscribe request

Publisher advertises topic to master

ROS PUBLISH SUBSCRIBE



ROS2: build upon DDS (Data Distribution Service)

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
```

DATA TYPES

- ▶ Let's have a look
- ▶ useful tools:
 - ▶ rostopic pub
 - ▶ rostopic echo
 - ▶ rostopic info
 - ▶ rosmsg
 - ▶ rossrv

message type define what is being exchanged between nodes

ROS as a framework (2)

Now that we have nodes with input and output, how do we make them communicate together?

If **A** publishes on *foo* and **B** listens on *foo*, **B** will receive topics data from **A**.

The services and topics names are used to match clients and servers.

A SIMPLE COMMUNICATION

- ▶ Let's implement a simple communication A->B
(without programming, just using **rostopic**)

ROS introspection tools (2)

```
# Show message type.  
rosmsg show geometry_msgs/Twist  
# Publish velocity  
rostopic pub -1 /robot/motion geometry_msgs/Twist \  
  "{linear: {x: 1.0}, angular: {z: 1.0}}"  
# See how often the robot pose is refreshed.  
rostopic hz /robot/odometry  
# Show one Odometry message  
rostopic echo -n1 /robot/odometry  
# Plot the pose.  
rxplot /robot/odometry/pose/pose/position/x,\  
        /robot/odometry/pose/pose/position/y \  
        /robot/odometry/pose/pose/orientation/z,\  
        /robot/odometry/pose/pose/orientation/w
```

Packaged Subsystems

- Some areas of research are mature enough to use standard algorithms
- No sense in reimplementing a new SLAM system for each new robot
- ROS allows multiple packages to be run together:
`roslaunch`

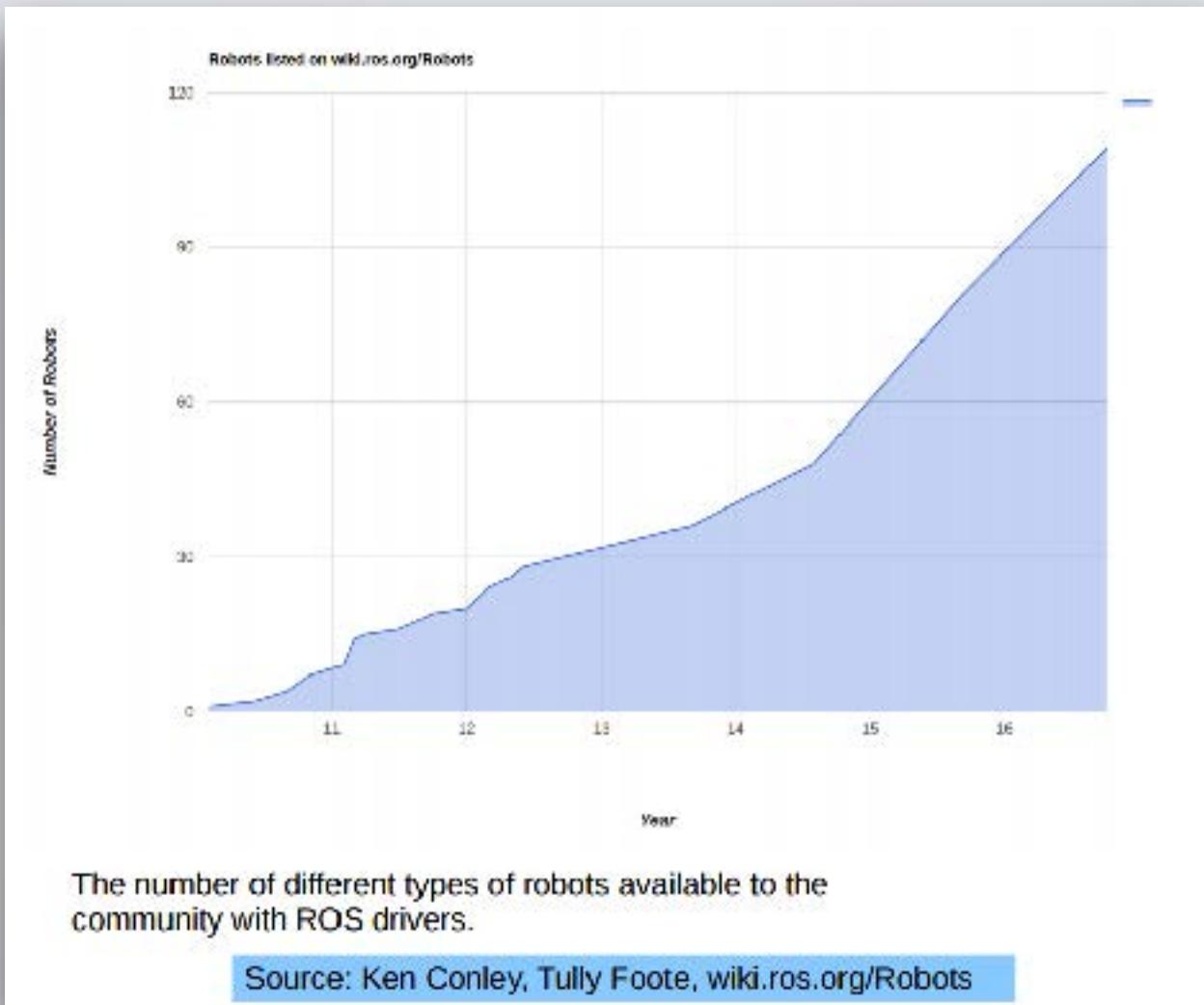
└ What is ROS?

└ Architecture

Language Agnostic



ROS DOMINATION?



TIME FOR SOME PYTHON

A FRESH START

- create your own **catkin** workspace (not 100% necessary for a single pure Python script):

- mkdir catkin_ws
- cd catkin_ws
- mkdir src http://wiki.ros.org/catkin/Tutorials/create_a_workspace
- cd src
- catkin_init_workspace

A FRESH START

- create your own package:
 - `catkin_create_pkg [-h]`
 - `cd ..`
<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>
 - `catkin_make`
 - `source devel/setup.bash`



<http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29>

COMPUTER VISION LIBRARIES

- ▶ OpenCV
 - ▶ ‘The’ Computer Vision Library – C++, Python, multiplatform
 - ▶ rich functionality, well tested, used in many systems, lots of examples

RECOMMENDED READING

- ▶ Programming Robots with ROS (Morgan Quigley, Brian Gerkey & And William D. Smart.) (ebook on reading list)
- ▶ **The ROS tutorials!!!**
- ▶ Resources:
 - ▶ <http://answers.ros.org>
 - ▶ <http://wiki.ros.org/ROS/Tutorials>
 - ▶ <https://github.com/LCAS/teaching/wiki/CMP3641M>

THANK YOU FOR LISTENING!

You can find **#RcbotTalk** on all your favourite podcasting providers:

Apple Podcasts [ow.ly/iOPs50CetaL](https://applepodcasts.com/ow.ly/iOPs50CetaL)

Amazon Music [ow.ly/GQIR50CetaN](https://music.amazon.com/ow.ly/GQIR50CetaN)

Stitcher [ow.ly/BD8650CetaM](https://stitcher.com/ow.ly/BD8650CetaM)

Deezer [ow.ly/AdPc50CetaK](https://deezer.com/ow.ly/AdPc50CetaK)

Spotify [ow.ly/3pU750CetaO](https://spotifyletter.com/ow.ly/3pU750CetaO)

Google Podcasts

CMP3103M AUTONOMOUS MOBILE ROBOTS

Prof Marc Hanheide

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

<https://www.lincoln.ac.uk/home/socs/computerscienceweek2021/>

SYLLABUS

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

Let's make a robot following a line!

Which middleware paradigm do "ROS topics" implement?

- data pull
- client-server
- synchronous processing
- publish-subscribe
- broadcast

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

The robot turns

The robot's laser sensor completes a scan of the environment (with an obstacle close to the front of the robot) and publishes the data

The callback of the subscriber is called

A subscriber is created, subscribing to the topic '/scan' of type 'sensor_msgs/LaserScan'

The node is initialised with the name 'receiver'

A message is published through the publisher

A new Twist object is created

A publisher is created, announcing to publish data on '/mobile_base/commands/velocity' of type 'geometry_msgs/Twist'

PROCESS FLOW IN ROS

```
import rospy
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist

class Receiver:

    def __init__(self):
        rospy.init_node('receiver')
        self.subscriber = rospy.Subscriber(
            '/scan', LaserScan, callback=self.cb)
        self.p = rospy.Publisher(
            '/mobile_base/commands/velocity', Twist, queue_size=1)

    def cb(self, incoming_data):
        # print len(incoming_data.ranges)
        if incoming_data.ranges[320] < 1.0:
            t = Twist()
            t.angular.z = 0.3
            self.p.publish(t)

rec = Receiver()
rospy.spin()
```

What is wrong with this code?

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

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

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

rospy.init_node('firstsubscriber')
fp = FirstSub()
rospy.spin()
```

publisher has no topic to publish to

syntax error in for-loop

wrong type

wrong import statement

callback not correctly registered

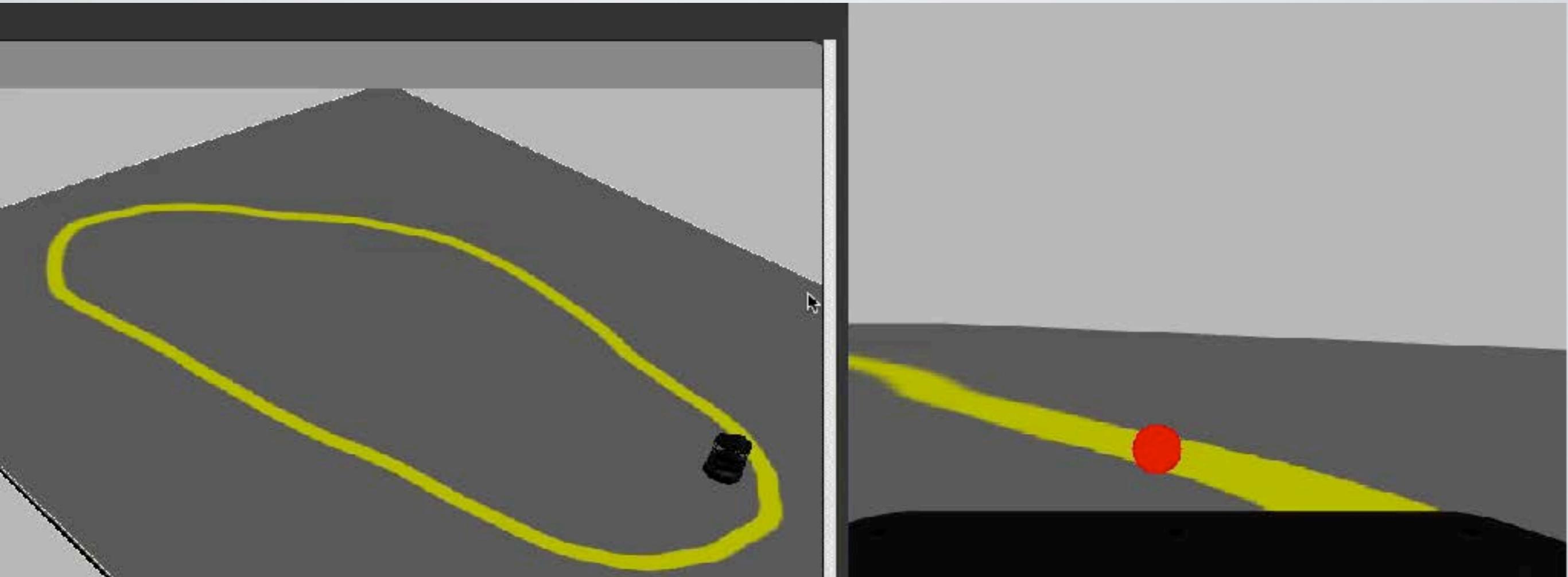
publish called with wrong data



VISION

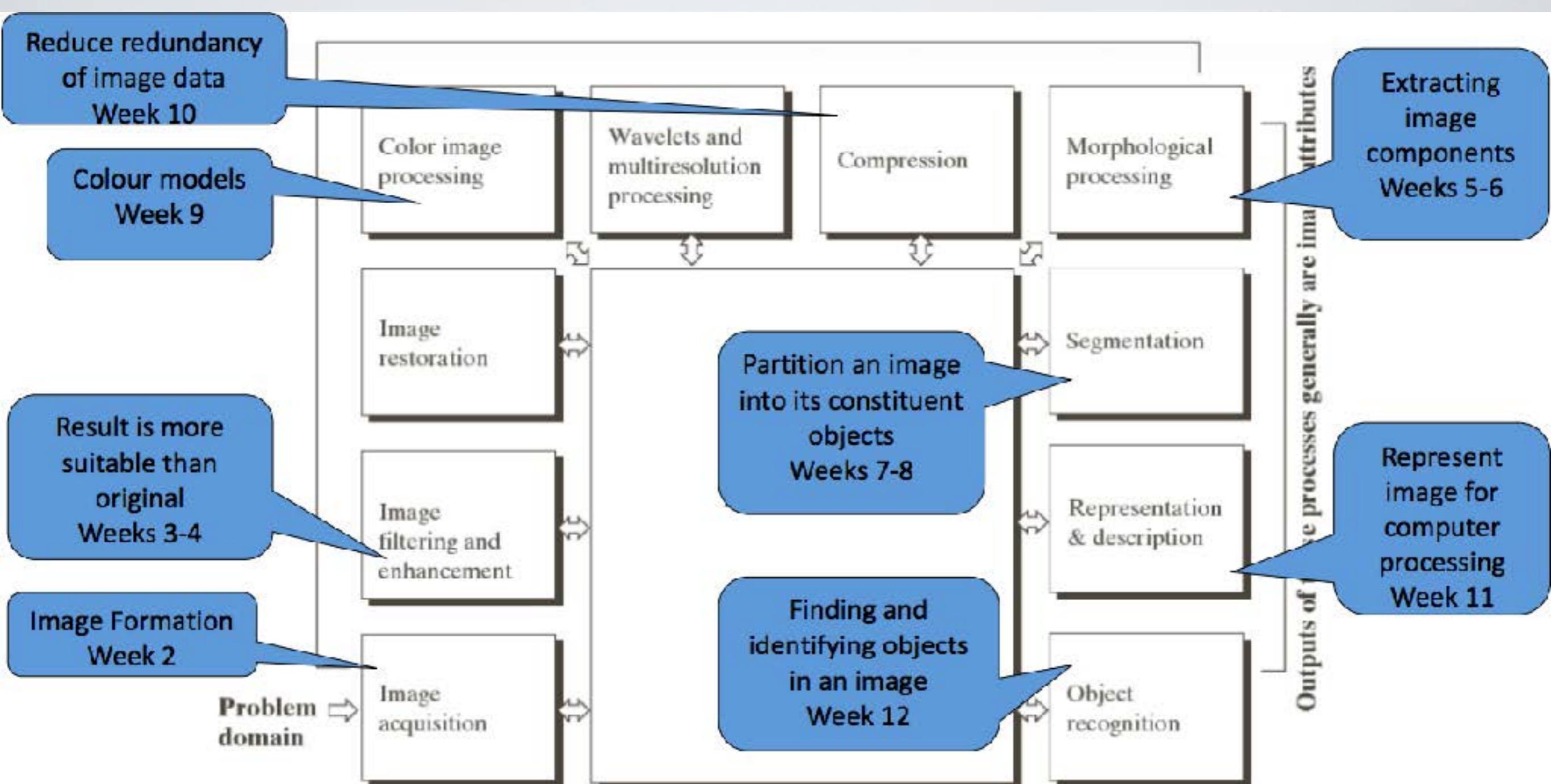
```
$> rostopic list
/camera/depth/camera_info
/camera/depth/image_raw
/camera/depth/points
/camera/parameter_descriptions
/camera/parameter_updates
/camera/rgb/camera_info
/camera/rgb/image_raw
/camera/rgb/image_raw/compressed
/camera/rgb/image_raw/compressed/parameter_descriptions
/camera/rgb/image_raw/compressed/parameter_updates
/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



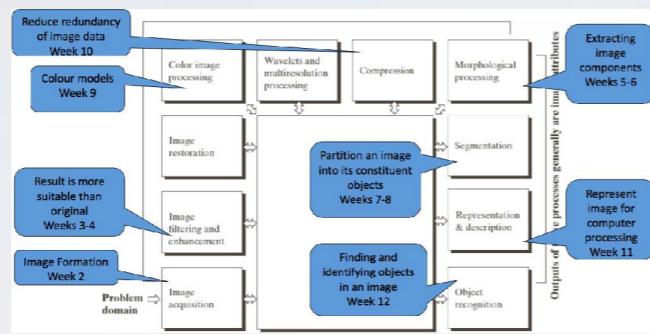
Morgan Quigley, Brian Gerkey & And William D. Smart. (2015)
Programming Robots with ROS [Chapter 12]

ROBOTVISION



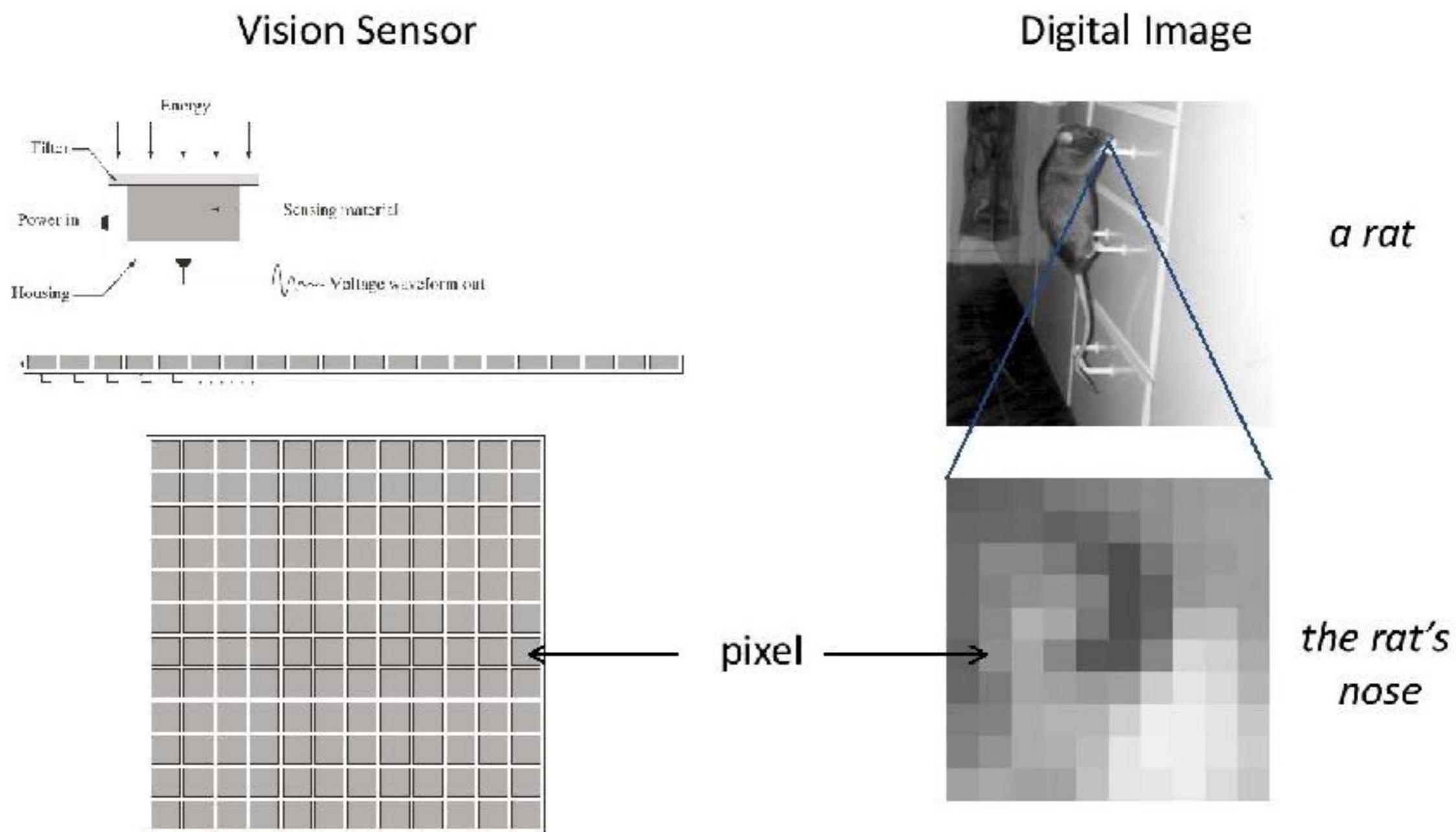
ROBOT VISION

- ▶ 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

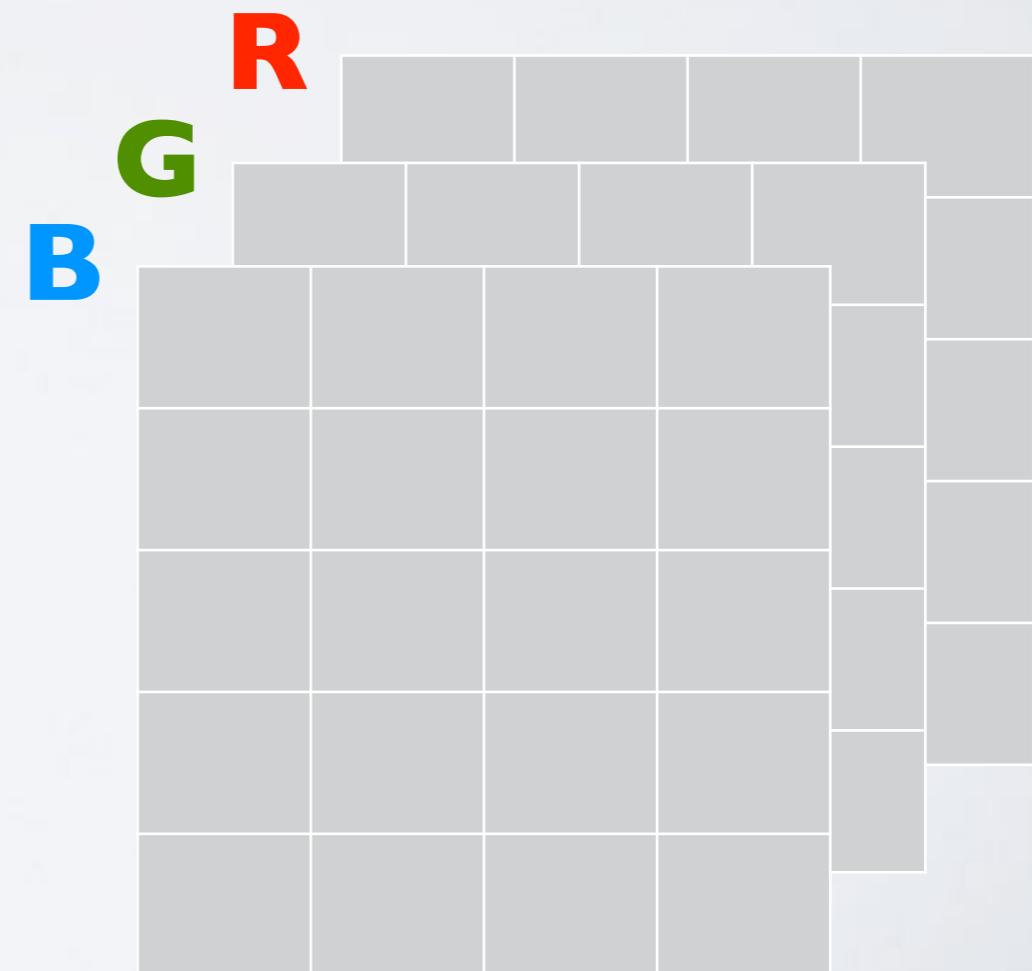
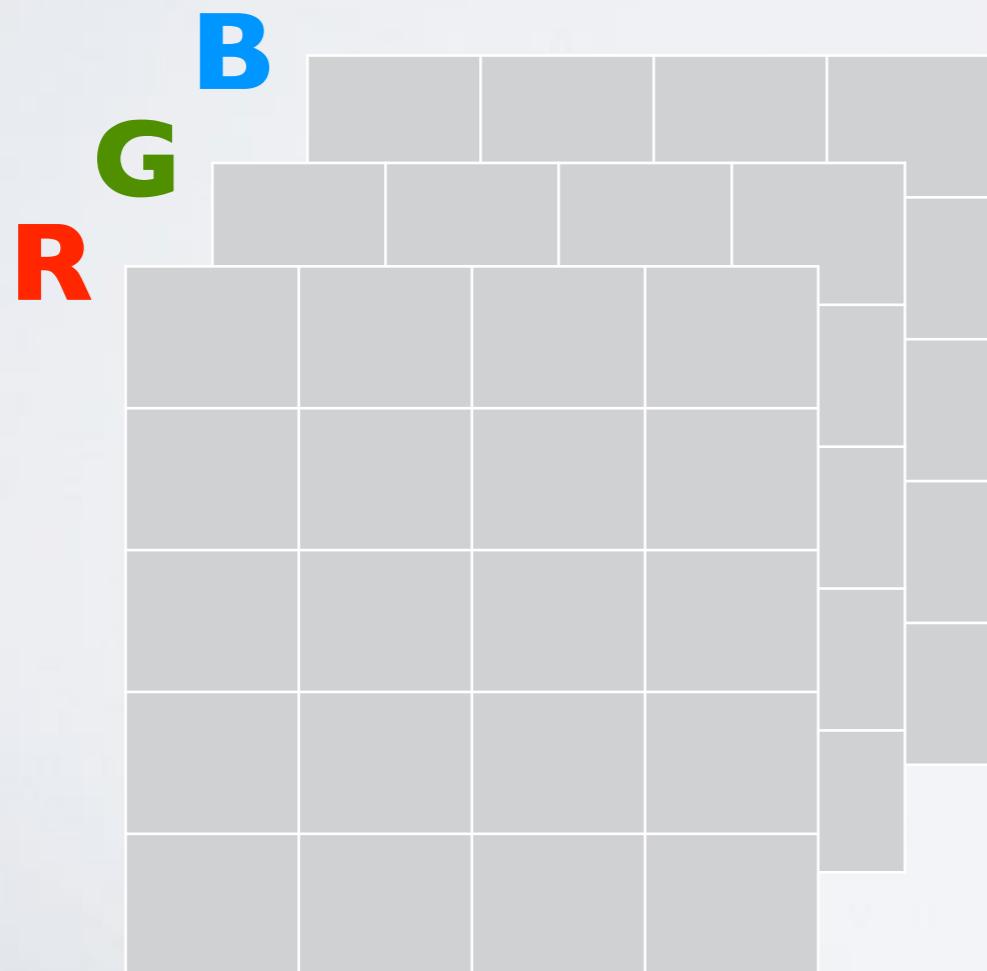
- ▶ Structure from motion (SFM)

- ▶ **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:

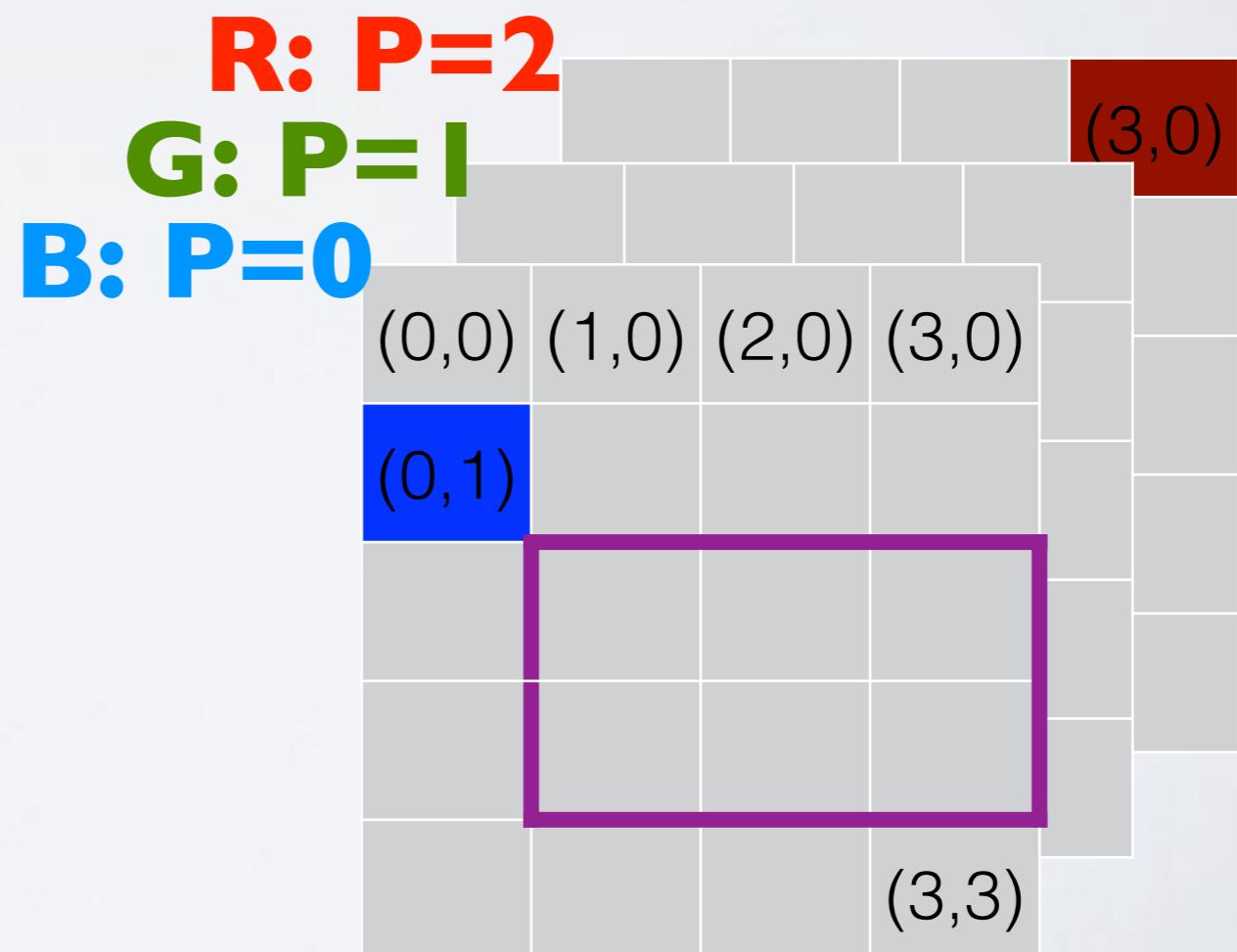
- ▶ $\text{img}[X, Y, P]$

- ▶ $\text{img}[3, 0, 2]$

- ▶ $\text{img}[0, 1, 0]$

- ▶ Ranges:

- ▶ $\text{img}[1:3, 2:3, 0]$



```
from cv2 import namedWindow, imread, imshow
from cv2 import waitKey, destroyAllWindows, startWindowThread
from cv2 import blur, Canny, circle

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

img = imread('..../blofeld.jpg')
imshow("original", img)
# create a new blurred image:
img2 = blur(img, (7, 7))
# draw on the image:
circle(img2, (100, 100), 30, (255, 0, 255), 5)
# display the image:
imshow("blur", img2)
# Canny is an algorithm for edge detection
img3 = Canny(img, 10, 200)
imshow("canny", img3)
# the shape gives you the dimensions
h = img3.shape[0]
w = img3.shape[1]
# 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 img3[y, x] > 0:
            count += 1
print('count edge pixels: %d' % count)
# wait key is also always needed to sync the GUI threads
waitKey(0)
# good practice to tidy up at the end
destroyAllWindows()
```

A SIMPLE OPENCV EXAMPLE (OPENCV_INTRO.PY)

- iterate over pixels
- manipulation pixels
- read and show images

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)
```

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

GETTING IMAGE STREAMS FROM A ROSTOPIC

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

- ▶ OpenCV and ROS play nicely
- ▶ There is a dedicated CvBridge to help you getting and processing image from the robot

http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython

- subscribe on Image topic
- convert to OpenCV in callback
- process the image using OpenCV

```
def __init__(self):
    self.bridge = CvBridge()
    self.image_sub = rospy.Subscriber("/camera/image_raw",
                                      Image, self.callback)
    # self.image_sub = rospy.Subscriber(
    #     "/camera/rgb/image_raw",
    #     Image, self.callback)

def callback(self, data):
    namedWindow("Image window")
    namedWindow("blur")
    namedWindow("canny")
    cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
```

opencv_bridge.py

```
import rospy
from cv2 import namedWindow, cvtColor, imshow
from cv2 import destroyAllWindows, startWindowThread
from cv2 import COLOR_BGR2GRAY, waitKey
from cv2 import blur, Canny
from numpy import mean
from sensor_msgs.msg import Image
from cv_bridge import CvBridge

class image_converter:

    def __init__(self):
        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("/camera/image raw",
                                         Image, self.callback)
        # self.image_sub = rospy.Subscriber(
        #     "/camera/rgb/image raw",
        #     Image, self.callback)

    def callback(self, data):
        namedWindow("Image window")
        namedWindow("blur")
        namedWindow("canny")
        cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")

        gray_img = cvtColor(cv_image, COLOR_BGR2GRAY)
        print mean(gray_img)
        img2 = blur(gray_img, (3, 3))
        imshow("blur", img2)
        img3 = Canny(gray_img, 10, 200)
        imshow("canny", img3)

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

rospy.init_node('image_converter')
ic = image_converter()
rospy.spin()

destroyAllWindows()
```

FIRST STEPS TOWARDS LINE FOLLOWING

```
#!/usr/bin/env python

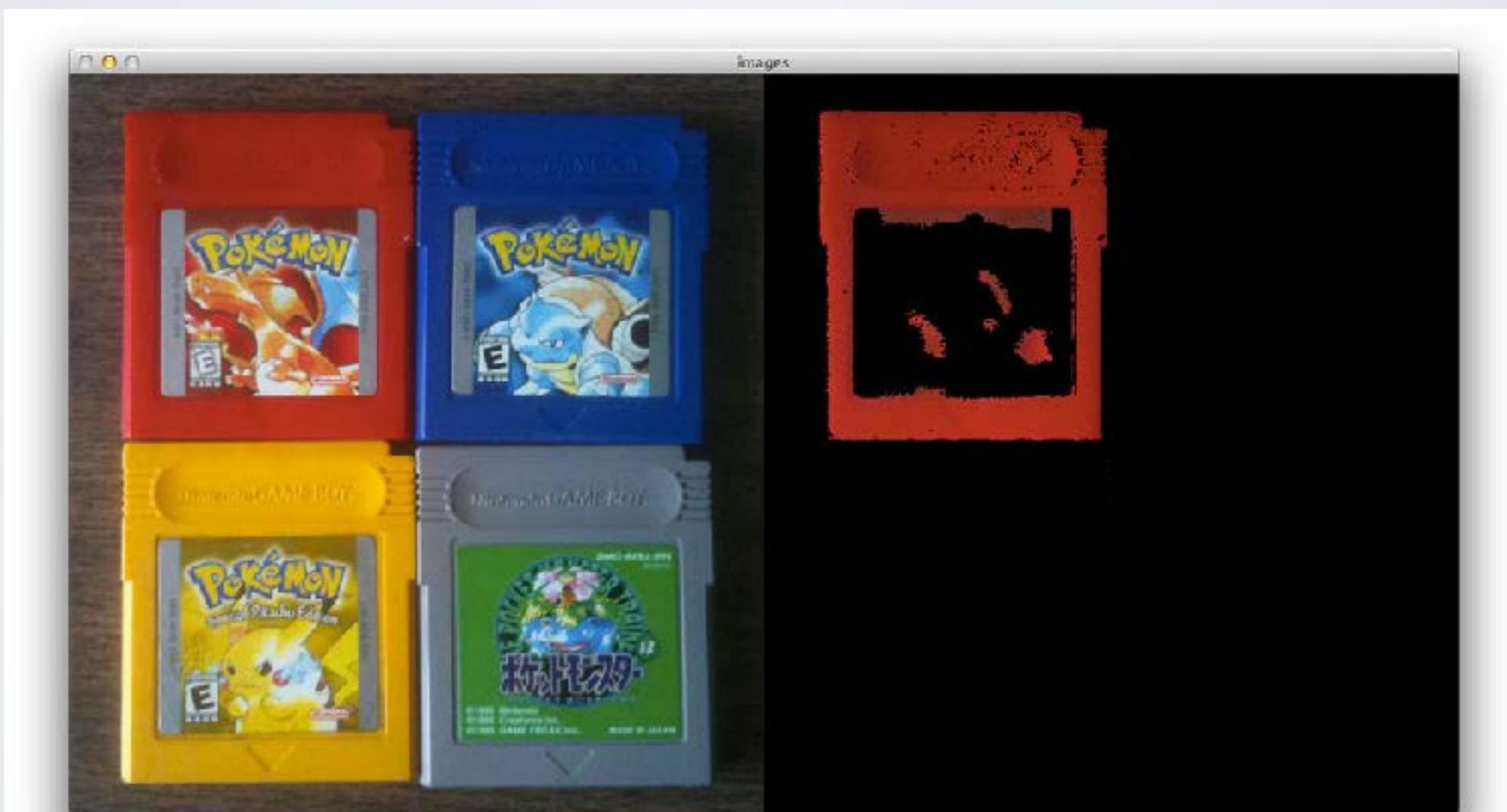
import rospy

from sensor_msgs.msg import Image

def image_callback(msg):
    pass

rospy.init_node('follower')
image_sub = rospy.Subscriber('camera/rgb/image_raw', Image,
image_callback)
rospy.spin()
```

A FIRST EXERCISE: COLOUR SLICING



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

Colour Models

Common models and their applications

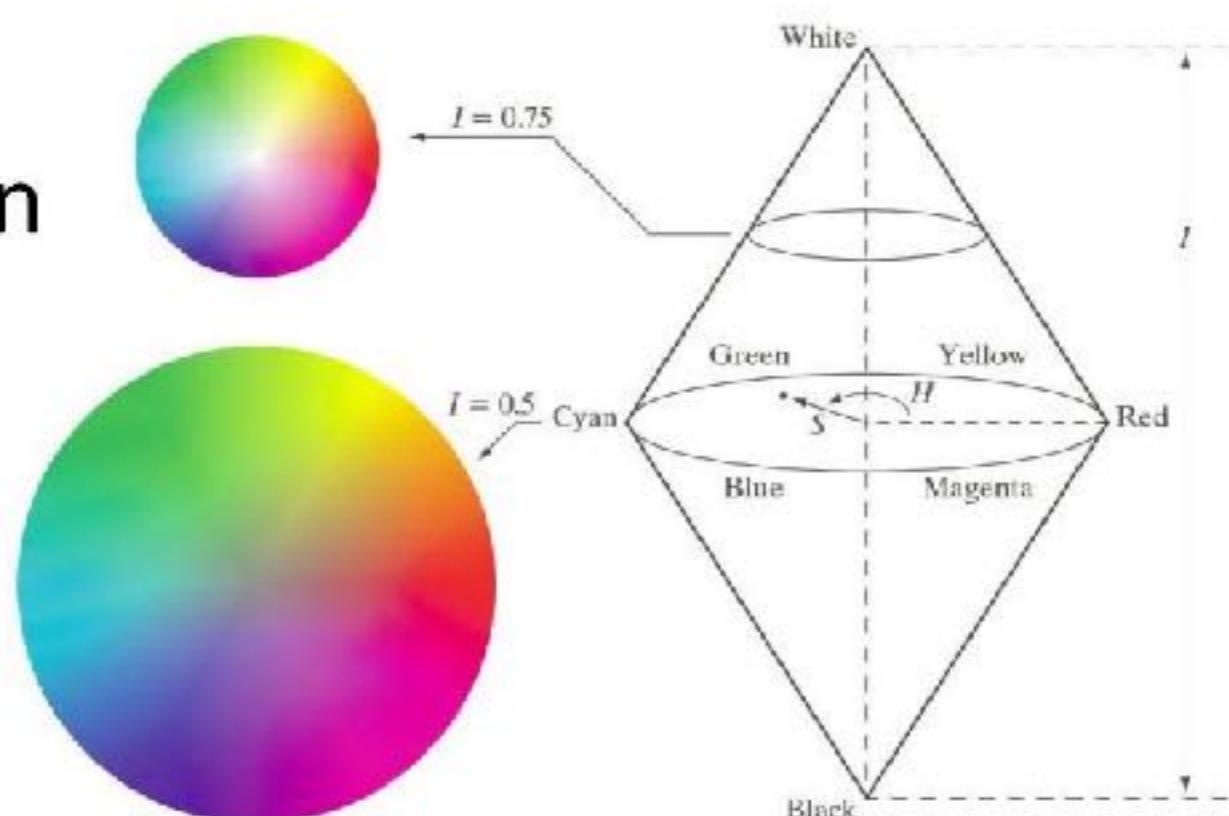
- RGB (red, green, blue)
 - colour monitors
- CMY, CMYK (cyan, magenta, yellow, black)
 - colour printers
- HSI (hue, saturation, intensity)
 - closely related to human perception of colour
 - decouples hue and intensity - very useful for real world applications where the overall intensity is often changing

Different colour image processing operations are easier or more difficult in different colour spaces

HSI Model

Hue, Saturation, Intensity

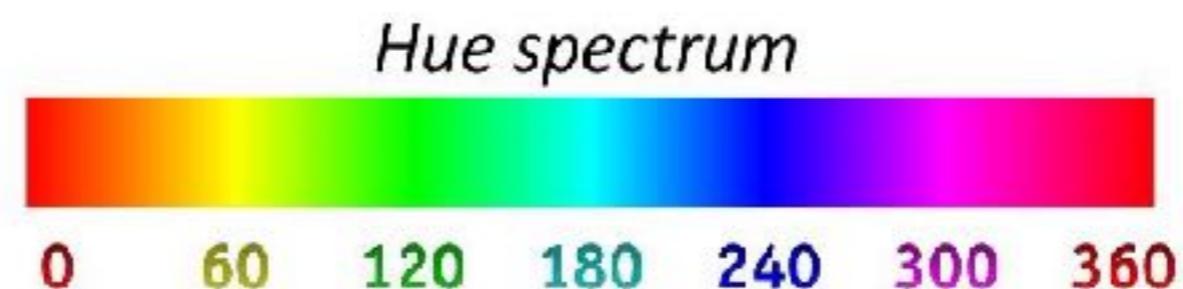
- separates intensity and hue
- resembles human vision
- difficult to display directly (transformation to RGB necessary)
- ! singularities (hue is undefined if the saturation is zero)



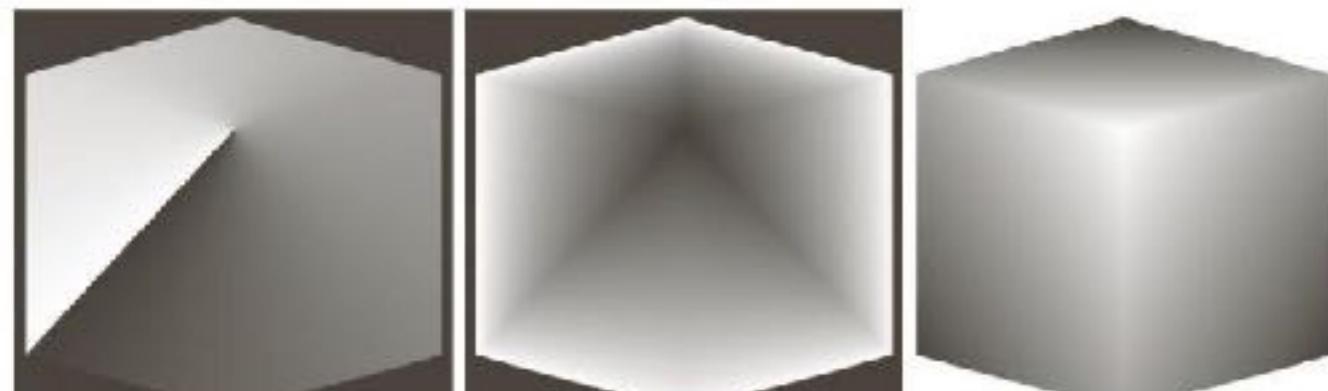
HSI Model

Hue component of HSI is expressed as an angle

- $0^\circ/360^\circ$ - Red
- 120° - Green
- 240° - Blue

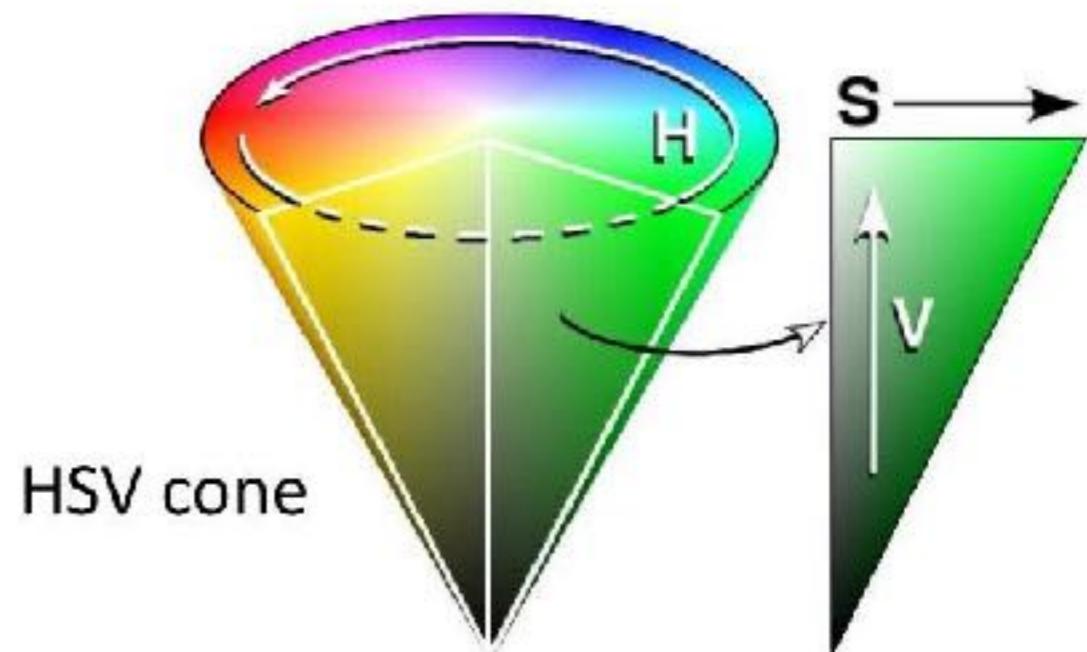


! take care to check for the discontinuity $0^\circ/360^\circ$ around “red” when processing HSI images!



a b c

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



HSV cone

Colour Models Example



Full color



Red

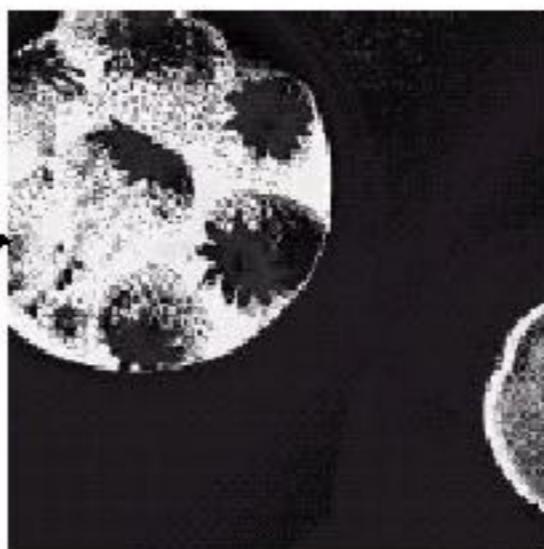


Green



Blue

Note the
discontinuity
(white/black) for
red strawberries



Hue



Saturation



Intensity

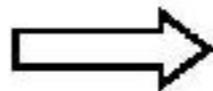
Colour Slicing

To highlight a specific range of colours

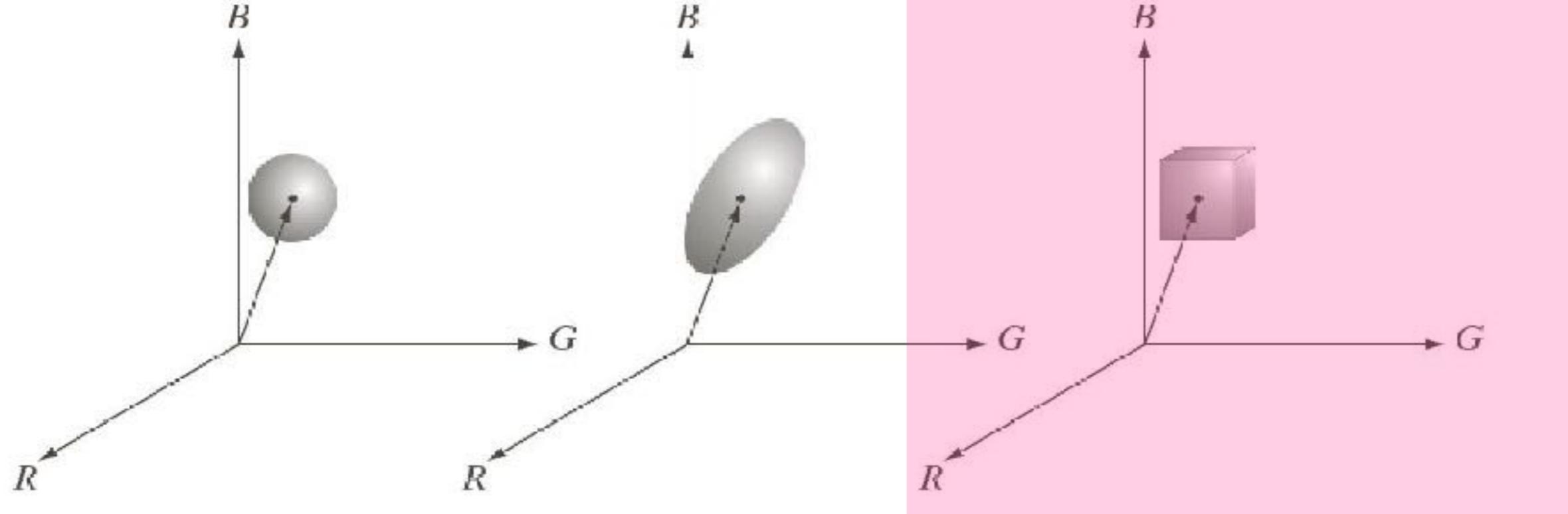
To define a mask for further processing

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



```
bgr_thresh = cv2.inRange(cv_image,  
                           numpy.array((200, 230, 230)),  
                           numpy.array((255, 255, 255)))
```

- subscribe on Image topic
- convert to OpenCV in callback
- optional: convert to different colour space (from BGR)
- use **inRange** for colour slicing
- display binary image
- optional: post-process image (morphological ops)

COLOUR SLICING IN PYTHON



```
bgr_thresh = cv2.inRange(cv_image,  
                           numpy.array((200, 230, 230)),  
                           numpy.array((255, 255, 255)))
```

- ▶ What colour space to slice in?
- ▶ What values to choose?
- ▶ grab frames:
`rosrun image_view image_saver image:=/camera/rgb/image_color`
- ▶ use “gimp” and its color picker
- ▶ use <http://imagecolorpicker.com/> and upload an image

FIRST STEPS TOWARDS LINE FOLLOWING (FOLLOWER_COLOR_FILTER.PY)

```
import numpy
import cv2
import cv_bridge
import rospy
from sensor_msgs.msg import Image

class Follower:
    def __init__(self):
        self.bridge = cv_bridge.CvBridge()
        self.image_sub = rospy.Subscriber(
            'camera/rgb/image_raw', Image,
            self.image_callback)

    def image_callback(self, msg):
        cv2.namedWindow("window", 1)
        image = self.bridge.imgmsg_to_cv2(msg)
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        lower_yellow = numpy.array([10, 60, 170])
        upper_yellow = numpy.array([255, 255, 255])
        mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
        cv2.bitwise_and(image, image, mask=mask)
        cv2.imshow("window", mask)
        cv2.waitKey(1)

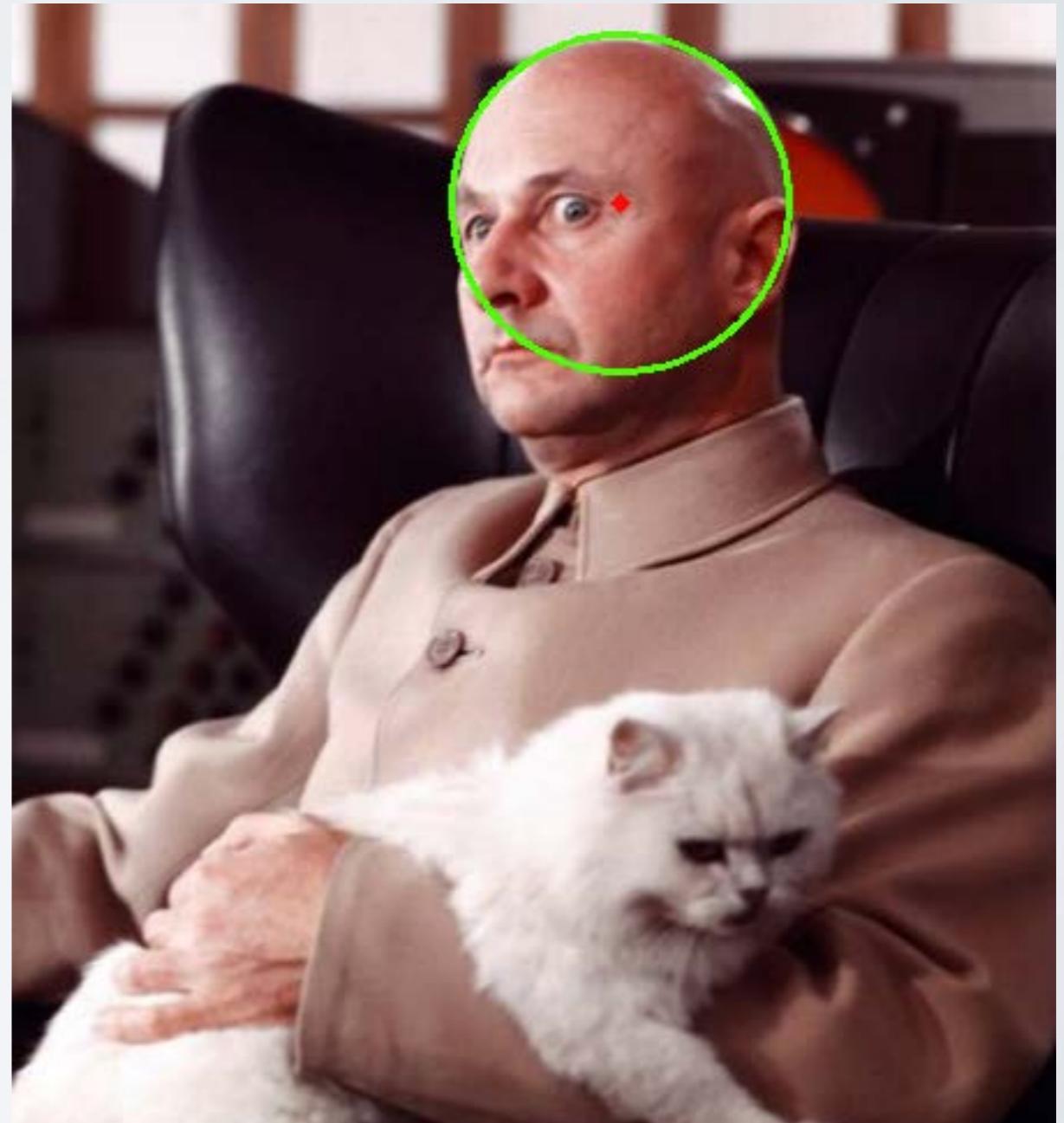
rospy.init_node('follower')
follower = Follower()
rospy.spin()
cv2.destroyAllWindows()
```

A SECOND EXERCISE: FINDING CONTOURS

- ▶ From colour slicing to regions!
- ▶ Work through http://opencv-python-tutorials.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_contours/py_contours_begin/py_contours_begin.html#contours-getting-started
- ▶ **see `color_contours.py`**

A THIRD EXTRA EXERCISE: FINDING CIRCLES

- ▶ Homework: Hough Transform
- ▶ What is it?





LiveSlides web content

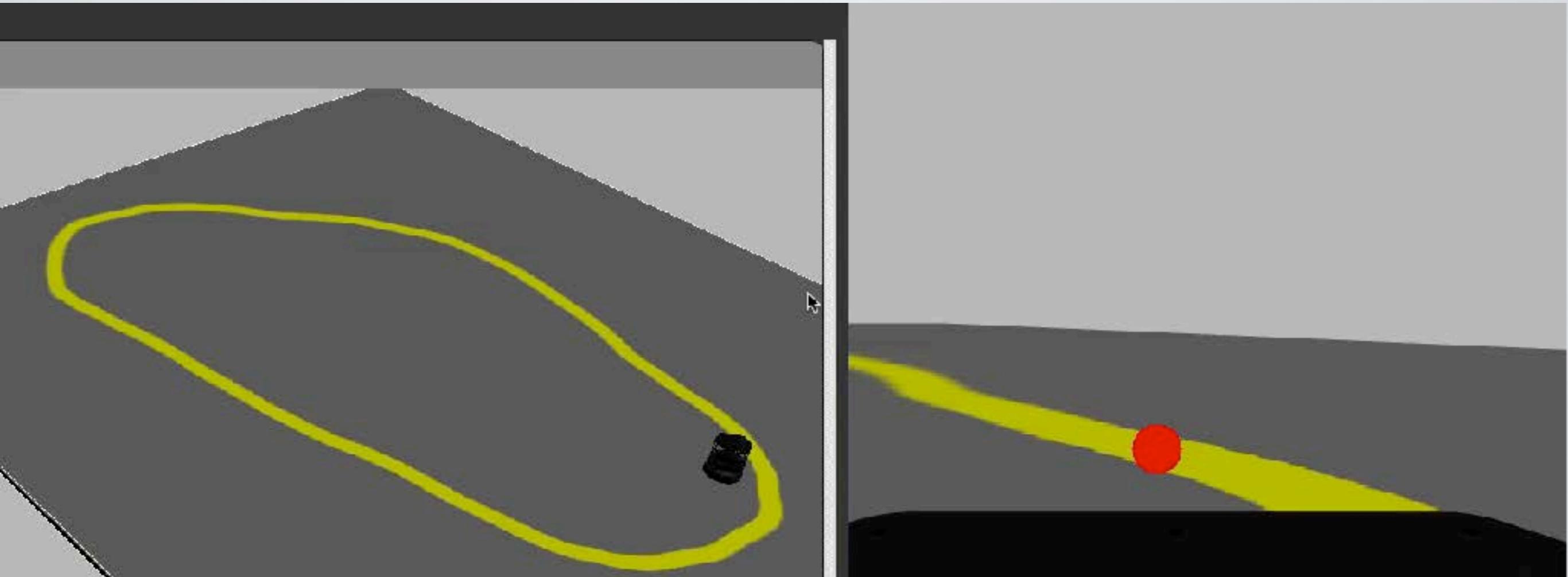
To view

Download the add-in.

liveslides.com/download

Start the presentation.

NOW FOR THE REAL STUFF



Morgan Quigley, Brian Gerkey & And William D. Smart. (2015)
Programming Robots with ROS [Chapter 12]

RESOURCES

- ▶ https://github.com/marc-hanheide/ros_book_sample_code/tree/master/chapter12
 - ▶ `catkin_init_workspace`
 - ▶ `git clone https://github.com/marc-hanheide/ros_book_sample_code.git`
 - ▶ `catkin_make (ignore errors)`
- ▶ Morgan Quigley, Brian Gerkey & And William D. Smart.
(2015) Programming Robots with ROS [Chapter 12]

FIRST STEPS TOWARDS LINE FOLLOWING (FOLLOWER_COLOR_FILTER.PY)

```
import numpy
import cv2
import cv_bridge
import rospy
from sensor_msgs.msg import Image

class Follower:
    def __init__(self):
        self.bridge = cv_bridge.CvBridge()
        self.image_sub = rospy.Subscriber(
            'camera/rgb/image_raw', Image,
            self.image_callback)

    def image_callback(self, msg):
        cv2.namedWindow("window", 1)
        image = self.bridge.imgmsg_to_cv2(msg)
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        lower_yellow = numpy.array([10, 60, 170])
        upper_yellow = numpy.array([255, 255, 255])
        mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
        cv2.bitwise_and(image, image, mask=mask)
        cv2.imshow("window", mask)
        cv2.waitKey(1)

rospy.init_node('follower')
follower = Follower()
rospy.spin()
cv2.destroyAllWindows()
```

FIRST STEPS TOWARDS LINE FOLLOWING (FOLLOWER_LINE_FINDER.PY)

[...]

```
def image_callback(self, msg):
    cv2.namedWindow("window", 1)
    image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_yellow = numpy.array([10, 60, 170])
    upper_yellow = numpy.array([255, 255, 255])
    mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
    h, w, d = image.shape
    search_top = 3*h/4
    search_bot = search_top + 20
    mask[0:search_top, 0:w] = 0
    mask[search_bot:h, 0:w] = 0
    M = cv2.moments(mask)
    if M['m00'] > 0:
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        cv2.circle(image, (cx, cy), 20, (0, 0, 255), -1)
    cv2.imshow("window", image)
    cv2.waitKey(3)
```

[...]



LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

CLOSING THE LOOP (FOLLOWER_.PY)

[...]

```
def image_callback(self, msg):
    cv2.namedWindow("window", 1)
    image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    lower_yellow = numpy.array([10, 10, 10])
    upper_yellow = numpy.array([255, 255, 250])
    mask = cv2.inRange(hsv, lower_yellow, upper_yellow)
    h, w, d = image.shape
    search_top = 3*h/4
    search_bot = 3*h/4 + 20
    mask[0:search_top, 0:w] = 0
    mask[search_bot:h, 0:w] = 0
    M = cv2.moments(mask)
    if M['m00'] > 0:
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        cv2.circle(image, (cx, cy), 20, (0, 0, 255), -1)
        err = cx - w/2
        self.twist.linear.x = 0.2
        self.twist.angular.z = -float(err) / 100
        self.cmd_vel_pub.publish(self.twist)
    cv2.imshow("window", image)
    cv2.waitKey(3)
```

[...]



LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

SOME MORE ROBOT-
RELATED VISION TO DISCUSS

APPLICATIONS

- ▶ Visual Path Following, Lagadic project, INRIA, France

Visual path following
using only monocular vision
for urban environments

- Lagadic project -

INRIA Rennes - IRISA

FEATURES

- ▶ Features
 - ▶ compact, meaningful representation of the image
- ▶ What are the good features?
 - ▶ edges, corners, blobs, colour, texture
- ▶ State of the art
 - ▶ SIFT, SURF, Haar-like, HOG, etc.
All in openCV!
- ▶ Applications - 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**

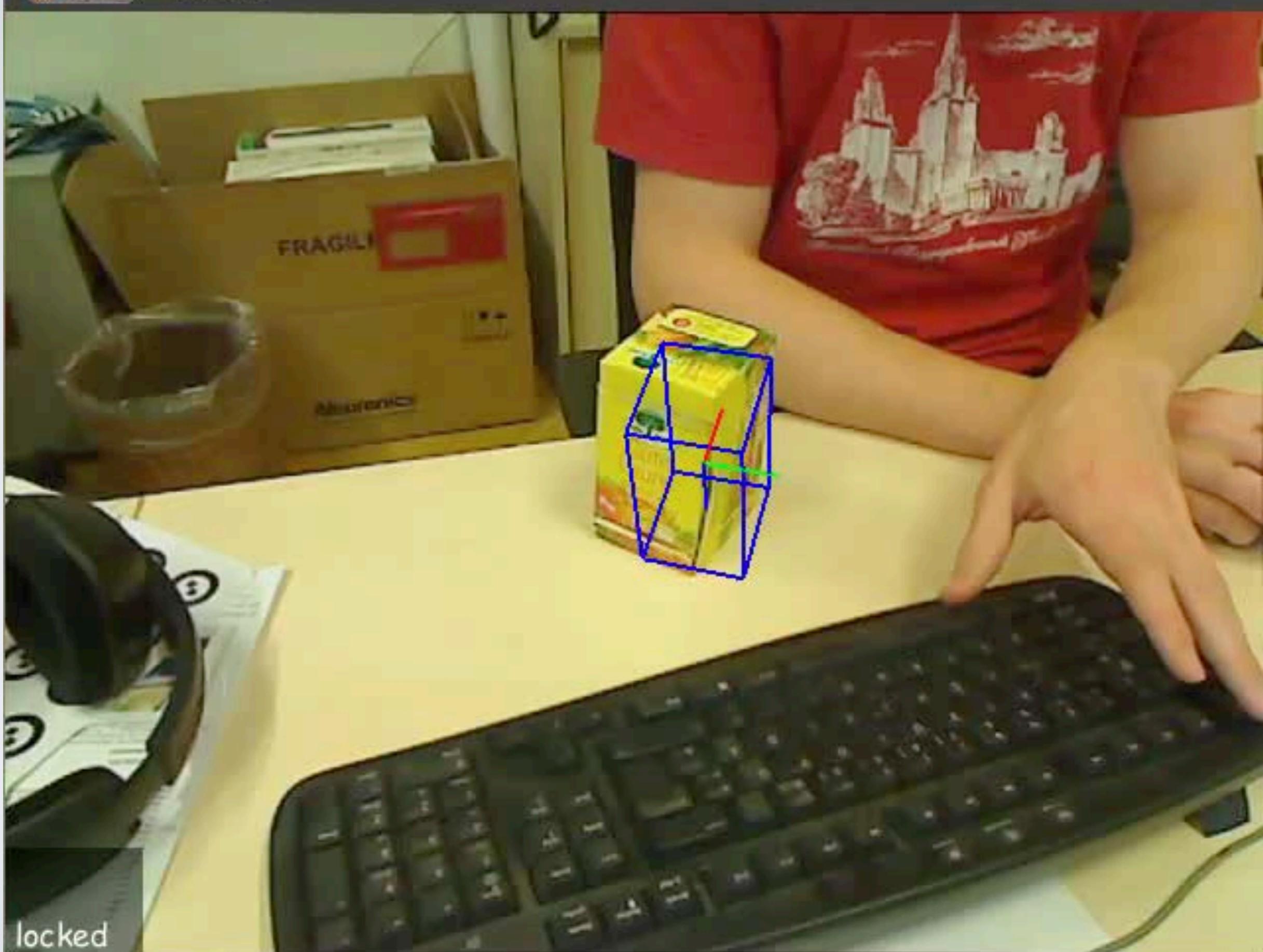
1 Microsoft Research Cambridge 2 Imperial College London

3 Newcastle University 4 Lancaster University

5 University of Toronto



Tracker



UNIVERSITY OF
LINCOLN

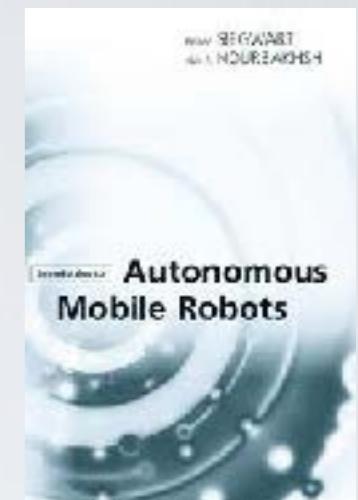


VISION IN ROBOTICS



RECOMMENDED READING (ALL ON BLACKBOARD)

- ▶ Siegwart, R. and Nourbakhsh, I.R.: Introduction to autonomous mobile robots MIT Press, 2004
- ▶ Section 4.3 – Feature Extraction
- ▶ Roth, P.M. And Winter, M., Survey of Appearance-based Method for Object Recognition, 2008

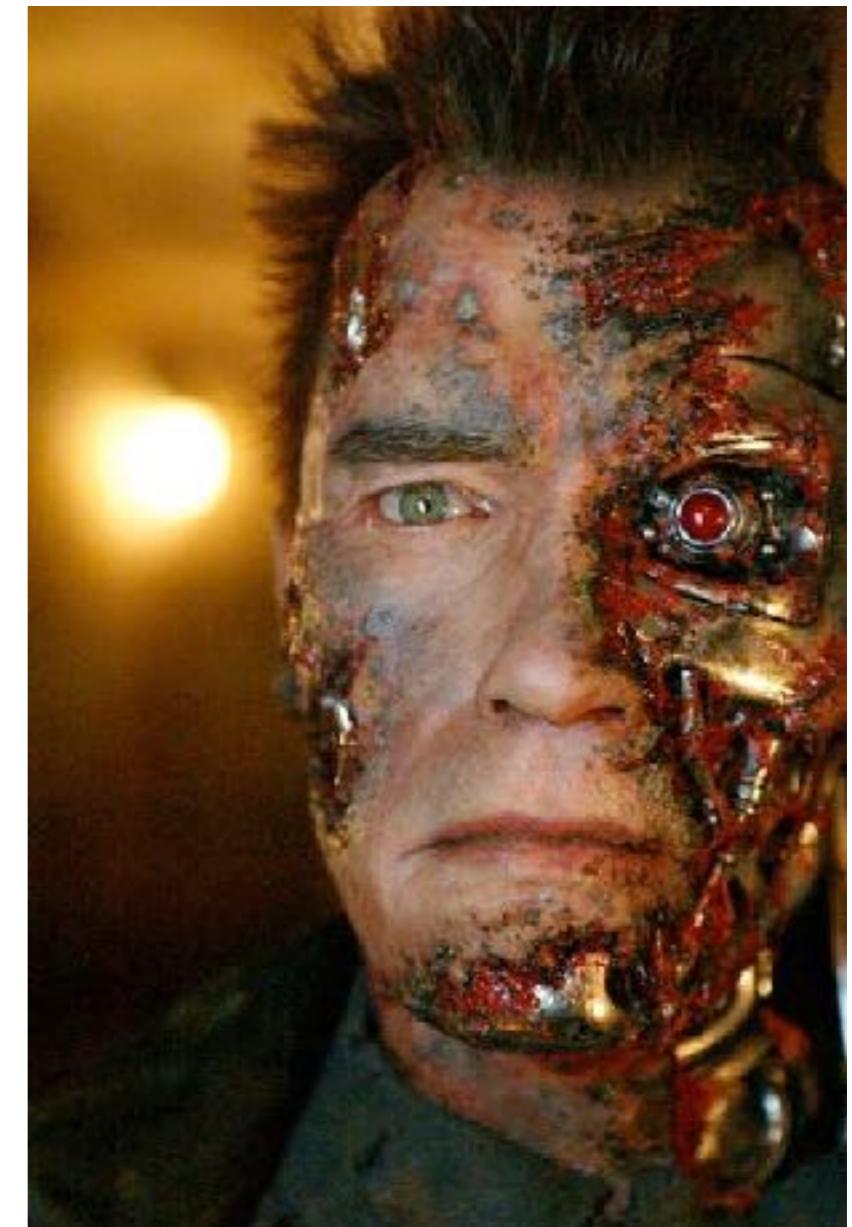


Morgan Quigley, Brian Gerkey & And William D. Smart. (2015)
Programming Robots with ROS [Chapter 12]

End of Lecture Feedback

When survey is active, respond at PollEv.com/mhanheide

THANK YOU
FOR LISTENING!



0 surveys done

⟳ 0 surveys underway

CMP3103M AMR

Prof. Marc Hanheide



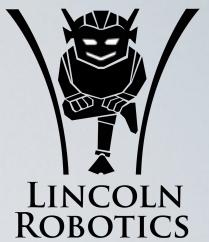
UNIVERSITY OF
LINCOLN



LINCOLN
ROBOTICS



UNIVERSITY OF
LINCOLN



SOME MORE ON ROS PROGRAMMING



Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app

What's wrong with this code?

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

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

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

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

node is initialised
after Subscriber and
Publisher are created

Illegal constructor for
Publisher

an import statement
is missing

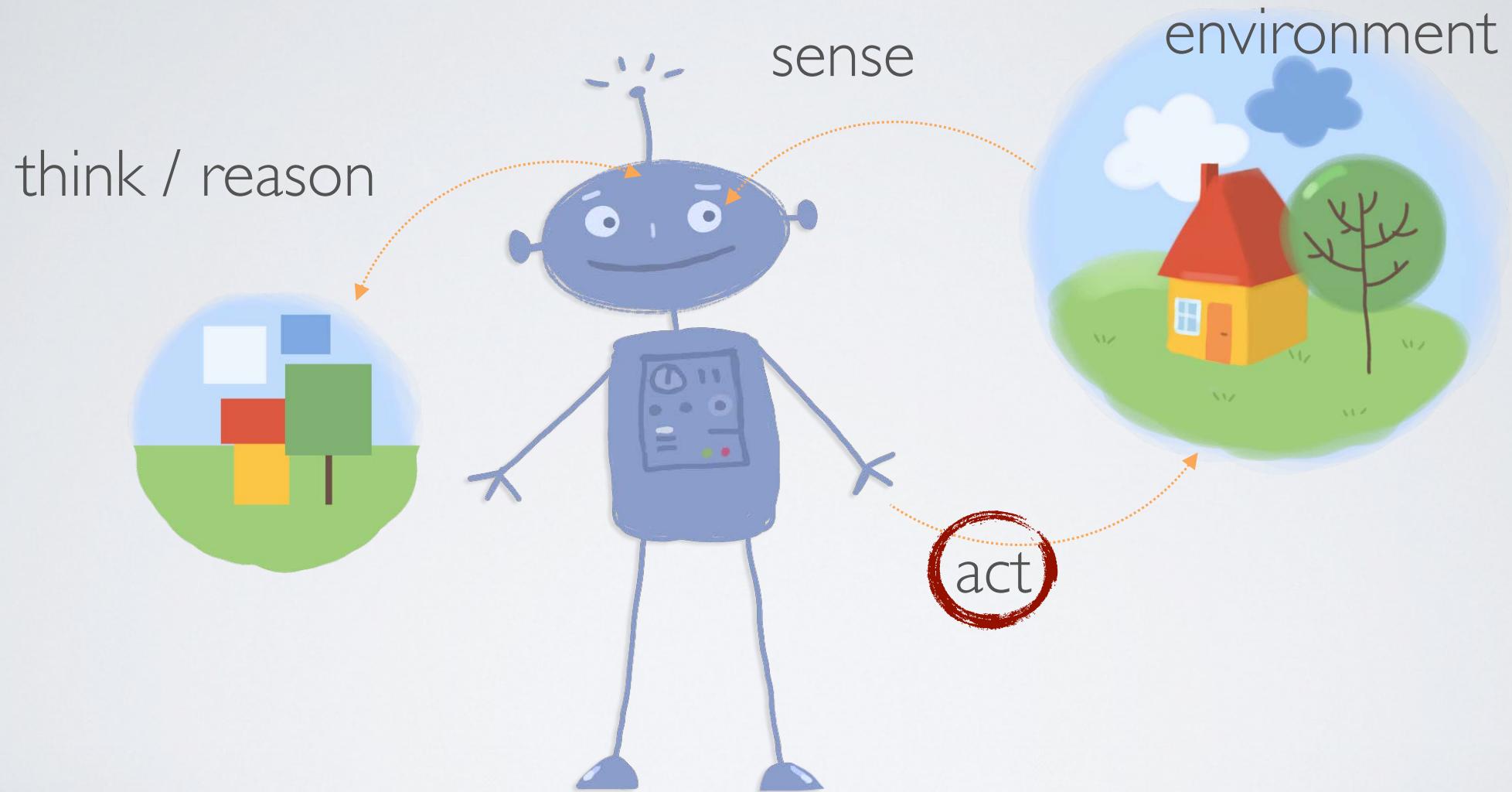
tries publishing
wrong type

construction of object
misses argument

SYLLABUS

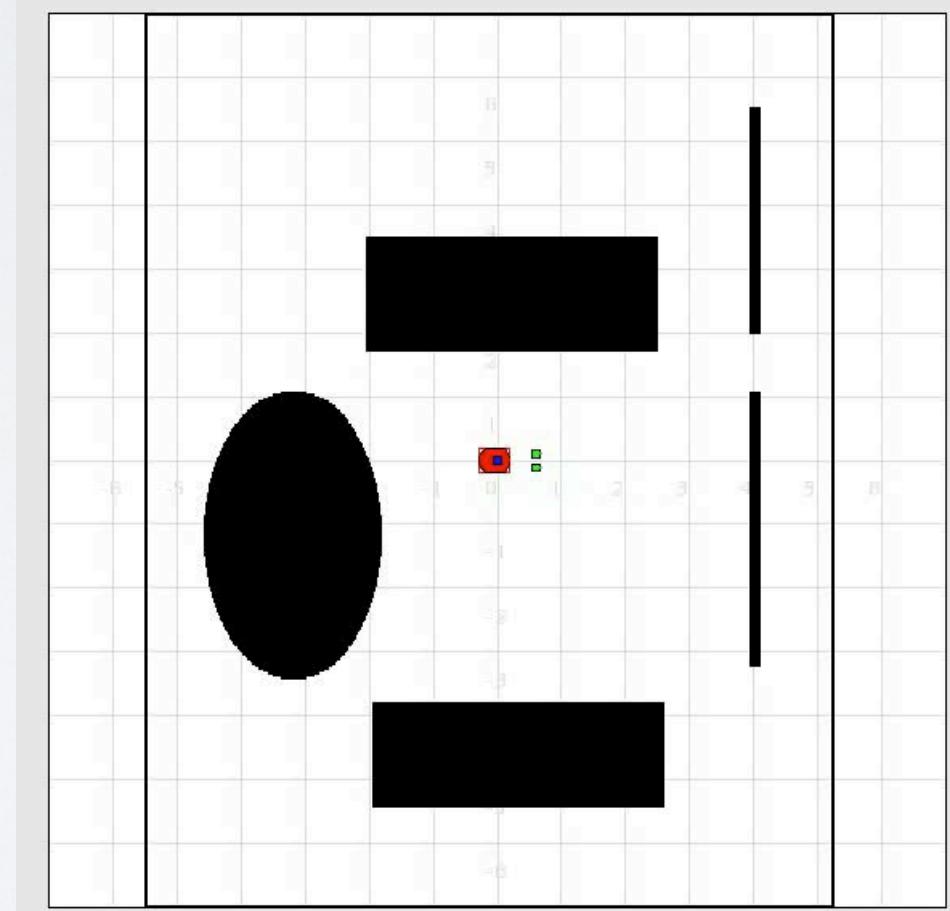
- ▶ Introduction to Robotics
- ▶ Robot Programming
- ▶ Robot Sensing & Vision
- ▶ **Robot Control**
- ▶ Robot Behaviours
- ▶ Control Architectures
- ▶ Navigation Strategies
- ▶ Map Building

ROBOTIC AGENT



MOBILE ROBOT CONTROL

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



THE TWIST

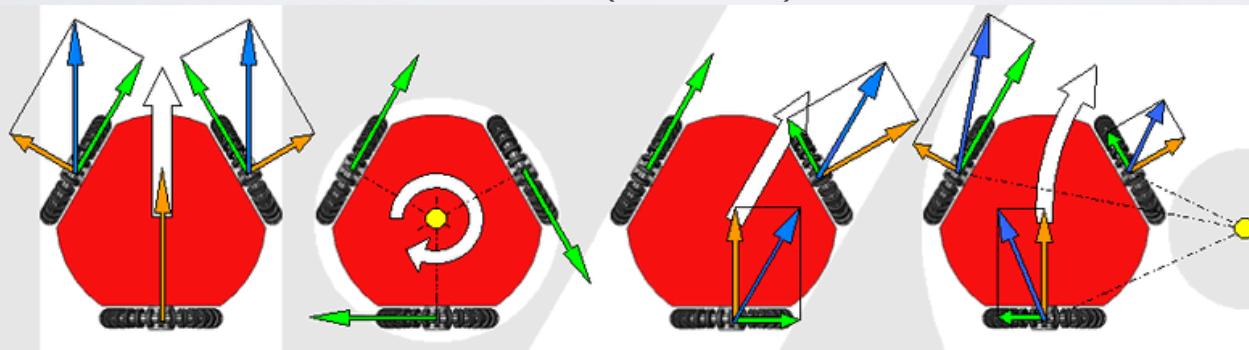
```
pub = rospy.Publisher(  
    '/cmd_vel',  
    Twist,  
    queue_size=1  
)  
t = Twist()  
pub.publish(t)
```

?

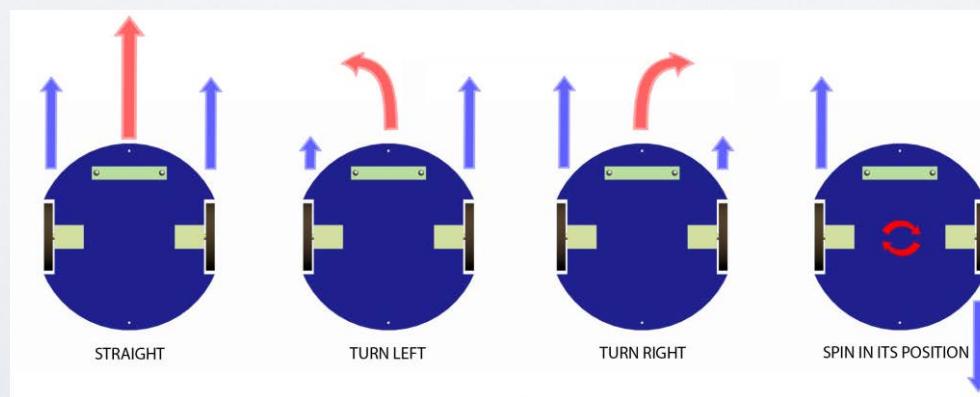


MOVEMENT COMMANDS

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



- ▶ Differential Drive (Turtlebot/Roomba)



CONTROLLING TURTLEBOT

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

FOR MOTION CONTROL

Model

Kinematics / Dynamics

Algorithm

control-parameter

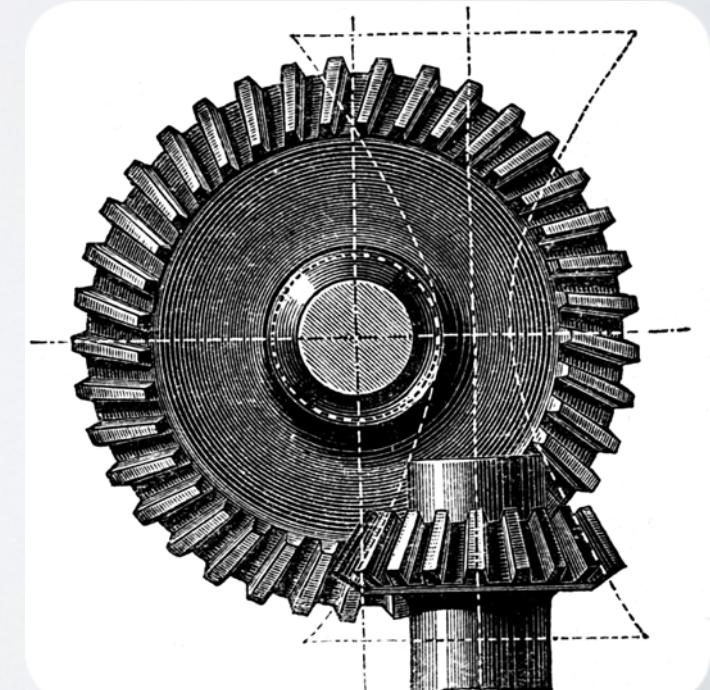
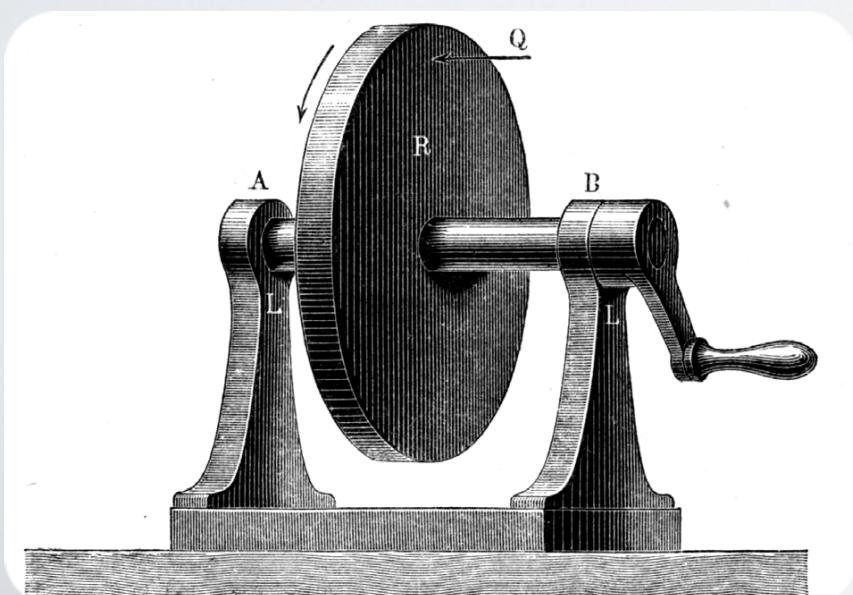
Engineering

control signal => actuators

KINEMATICS

Model

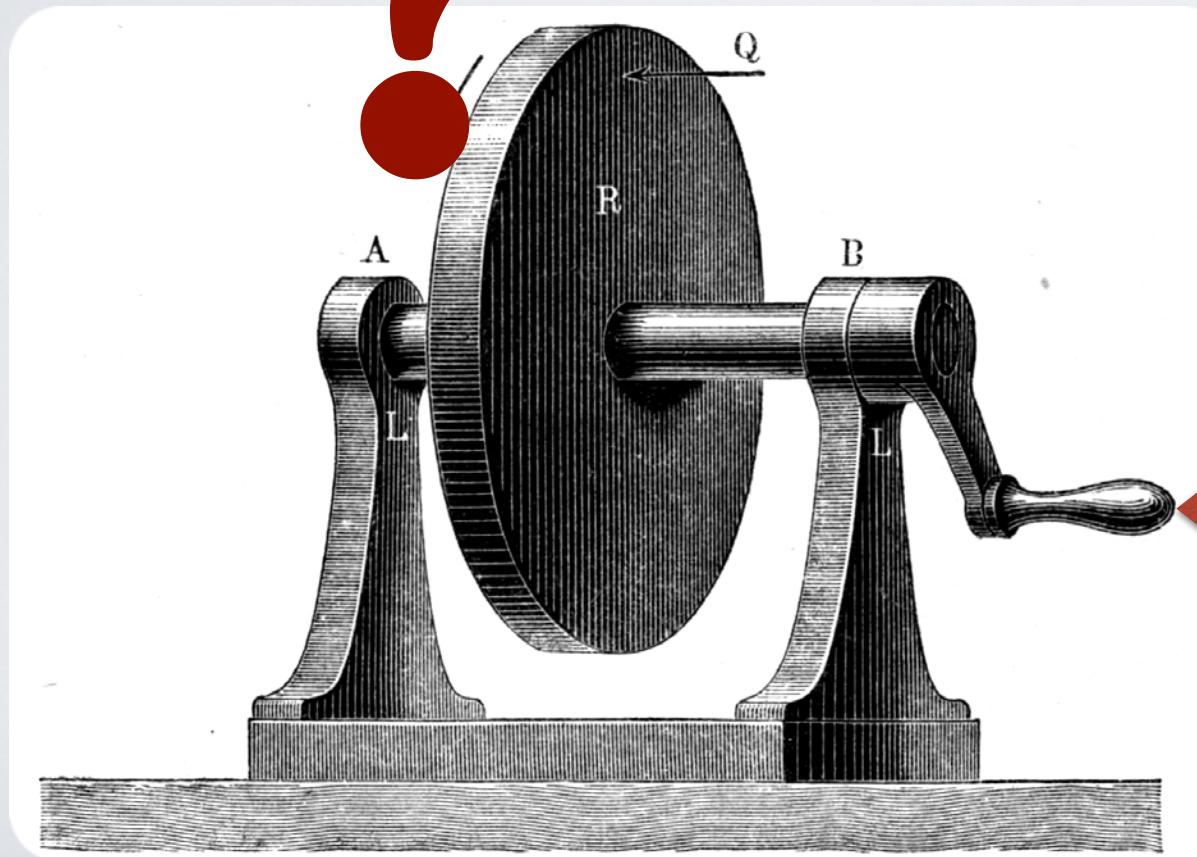
Kinematics / Dynamics



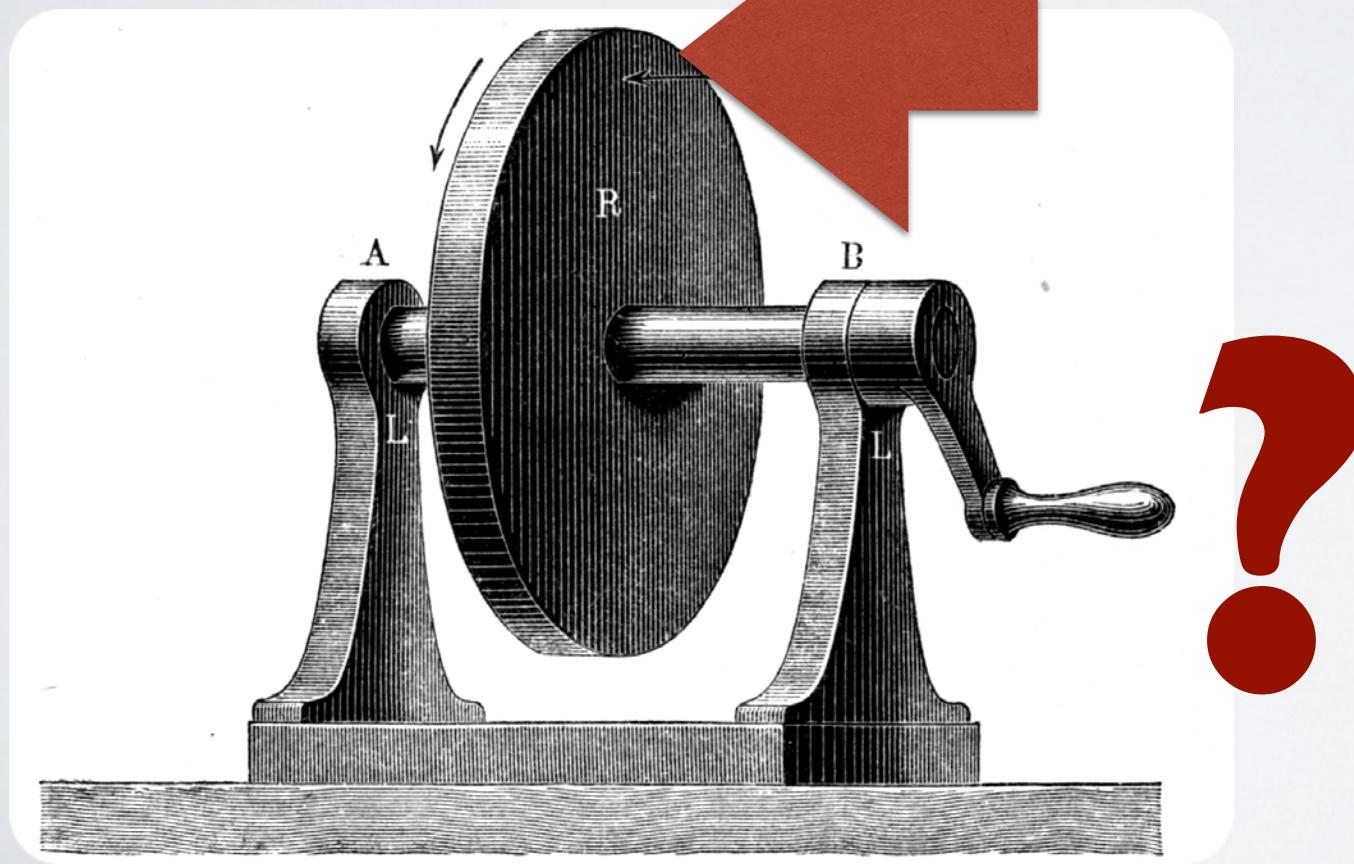
AD-HOC MODEL

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

FORWARD MODEL



INVERSE MODEL



AD-HOC FORWARD MODEL IS EASY?



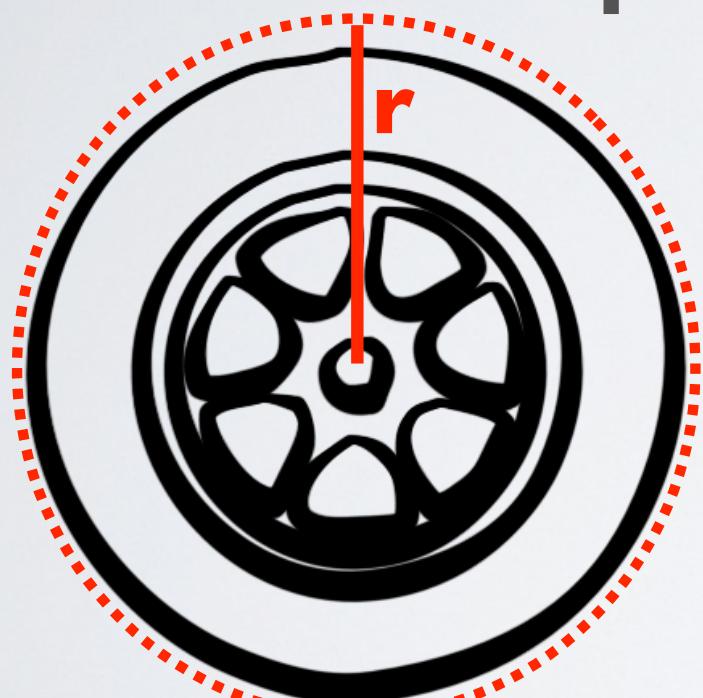
THE WHEEL



$$c = 2\pi r$$

c

A SIMPLE ROBOT WITH ONE WHEEL: FORWARD MODEL



Spinning at 60 degrees per second

Degrees... Radians?

$$\omega = (60 / 360) * 2\pi$$

$$= 1.0471975511965976$$

$$v = r \omega$$

c

AND WITH TWO WHEELS?

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



but $\omega_l = \omega_r$ when going straight

WHAT'S THE INVERSE MODEL?



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

$$v = r \omega$$

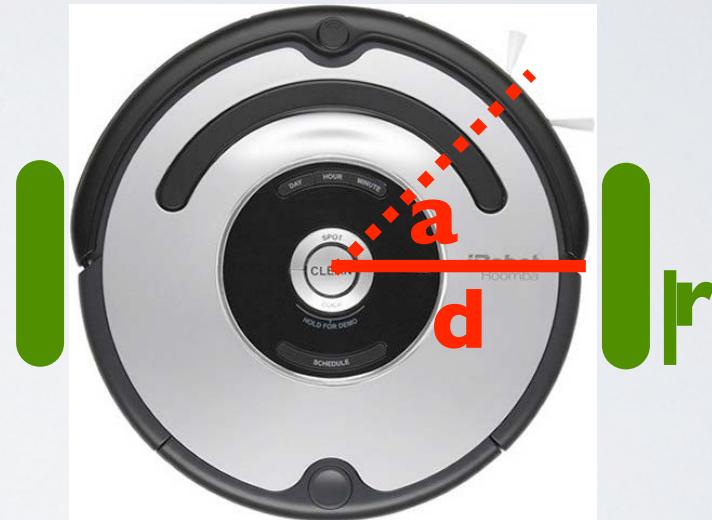
$$\omega = v/r$$

AD-HOC FORWARD MODEL IS EASY?

$$v = 0$$

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

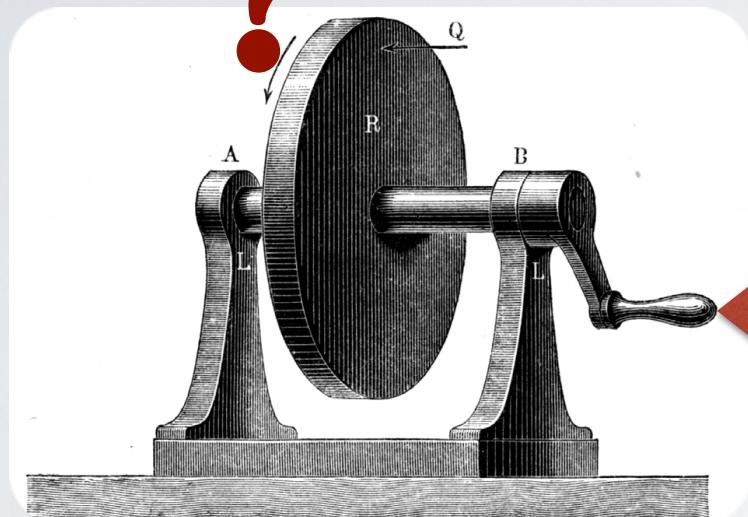
$$\Rightarrow \omega_l = -\omega_r$$



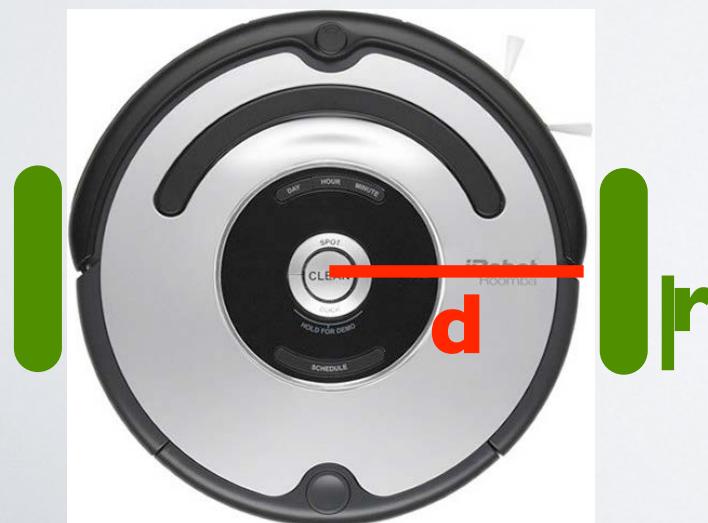
Hint: Assume both wheels moving same speed again!

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

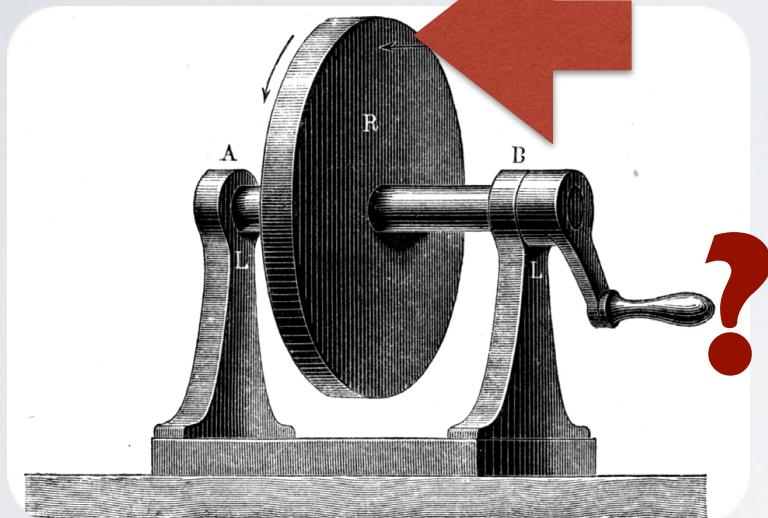
FORWARD MODEL



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



INVERSE MODEL



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



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



PYTHON TIME

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





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

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

PYTHON TIME

```
import math
from geometry_msgs.msg import Twist

wheel_radius = 1
robot_radius = 1

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





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

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



PYTHON TIME

```
import math
from geometry_msgs.msg import Twist

wheel_radius = 1
robot_radius = 1

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

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



Forward and inverse Kinematics for Turtlebot

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

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

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

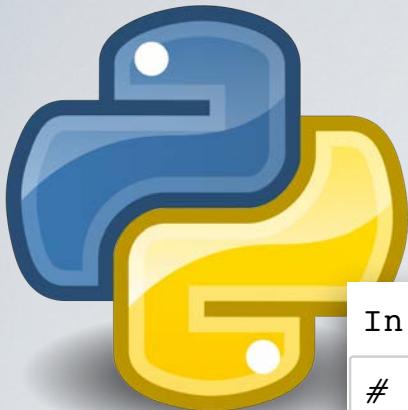
In [8]:

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

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

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

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



PYTHON TIME

In [13]:

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

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

v = 0.314159,    a = 0.000000
```

In [15]:

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

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

v = 0.000000,    a = 2.513274
```

In [16]:

```
# inverse kinematics:

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

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

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



LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

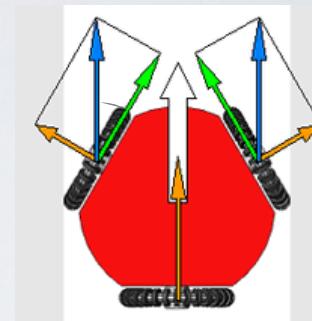
USING THE KINEMATIC MODEL

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

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

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

forward kinematics



- ▶ Pure rotation

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

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

forward kinematics



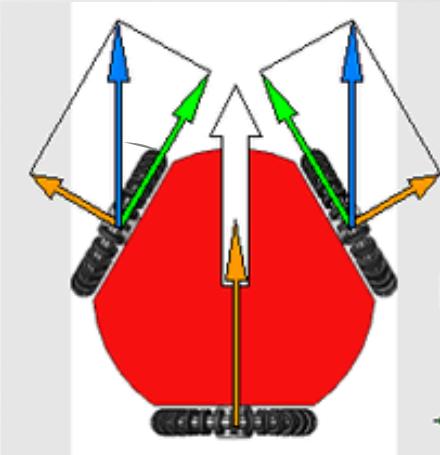
INVERSE KINEMATICS FOR ROVIO

- ▶ General (omni-drive) velocity control example

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

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

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

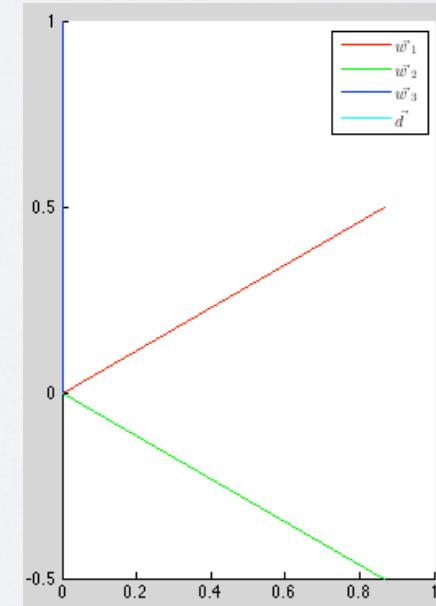
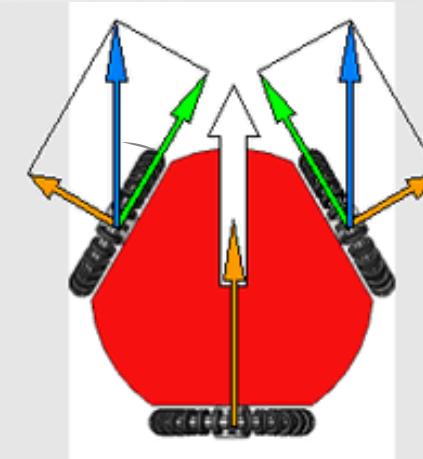


ROVIO KINEMATICS IN MATLAB

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

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

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

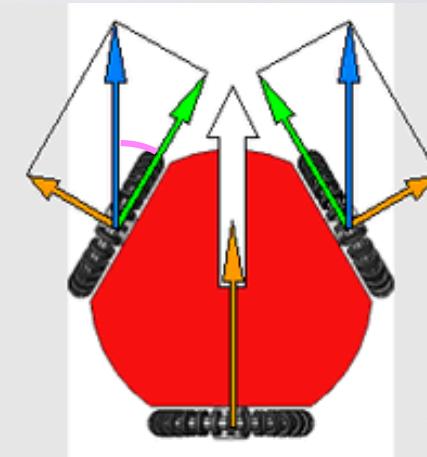


ROVIO KINEMATICS IN MATLAB

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

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

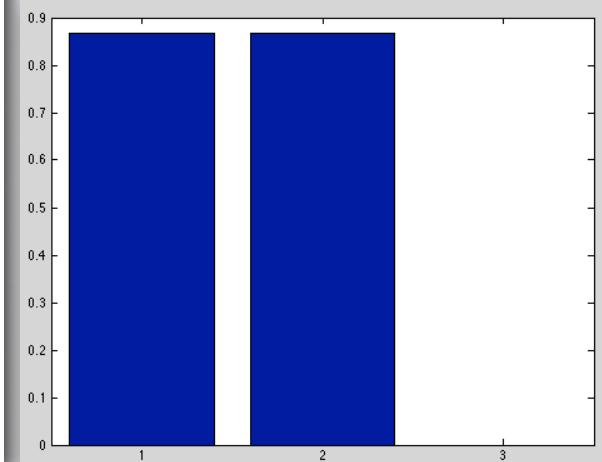
$$\begin{aligned}\omega_i &= \vec{w}_i \cdot \vec{d} \\ \vec{w}_i &= \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix} \\ \vec{d} &= \begin{pmatrix} \cos(\delta) \\ \sin(\delta) \end{pmatrix} \cdot v\end{aligned}$$



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

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

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



FOR MOTION CONTROL

Model



Kinematics / Dynamics

Algorithm

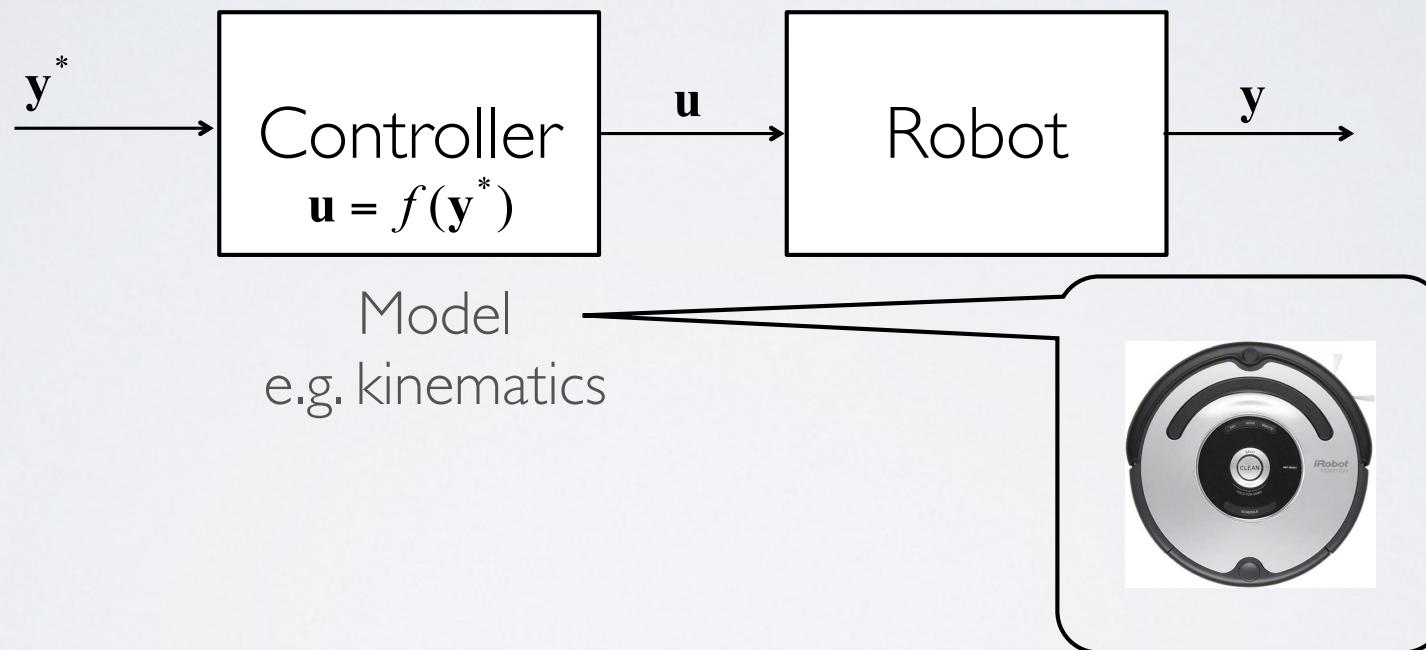
control-parameter

Engineering

control signal => actuators

OPEN LOOP CONTROL

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



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

YOU ARE A ROBOT

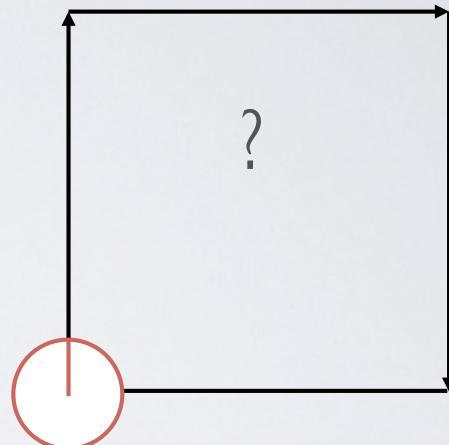
- ▶ close your eyes and hit the spot!



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

OPEN LOOP CONTROL

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



OPEN LOOP CONTROL, MOVE IN A SQUARE

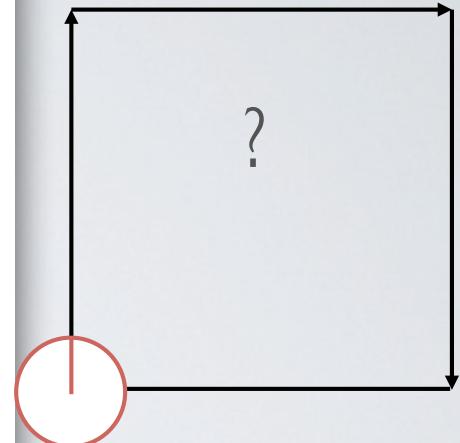
```
# 5 HZ
r = rospy.Rate(5);

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

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

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

#two keep drawing squares. Go forward for 2 seconds (10 x 5 HZ) then turn for 2 second
count = 0
while not rospy.is_shutdown():
    # go forward 0.4 m (2 seconds * 0.2 m / seconds)
    rospy.loginfo("Going Straight")
    for x in range(0,10):
        self.cmd_vel.publish(move_cmd)
        r.sleep()
    # turn 90 degrees
    rospy.loginfo("Turning")
    for x in range(0,10):
        self.cmd_vel.publish(turn_cmd)
        r.sleep()
    count = count + 1
    if(count == 4):
        count = 0
    if(count == 0):
        rospy.loginfo("TurtleBot should be close to the original starting position (but it's probably way off)")
```





LiveSlides web content

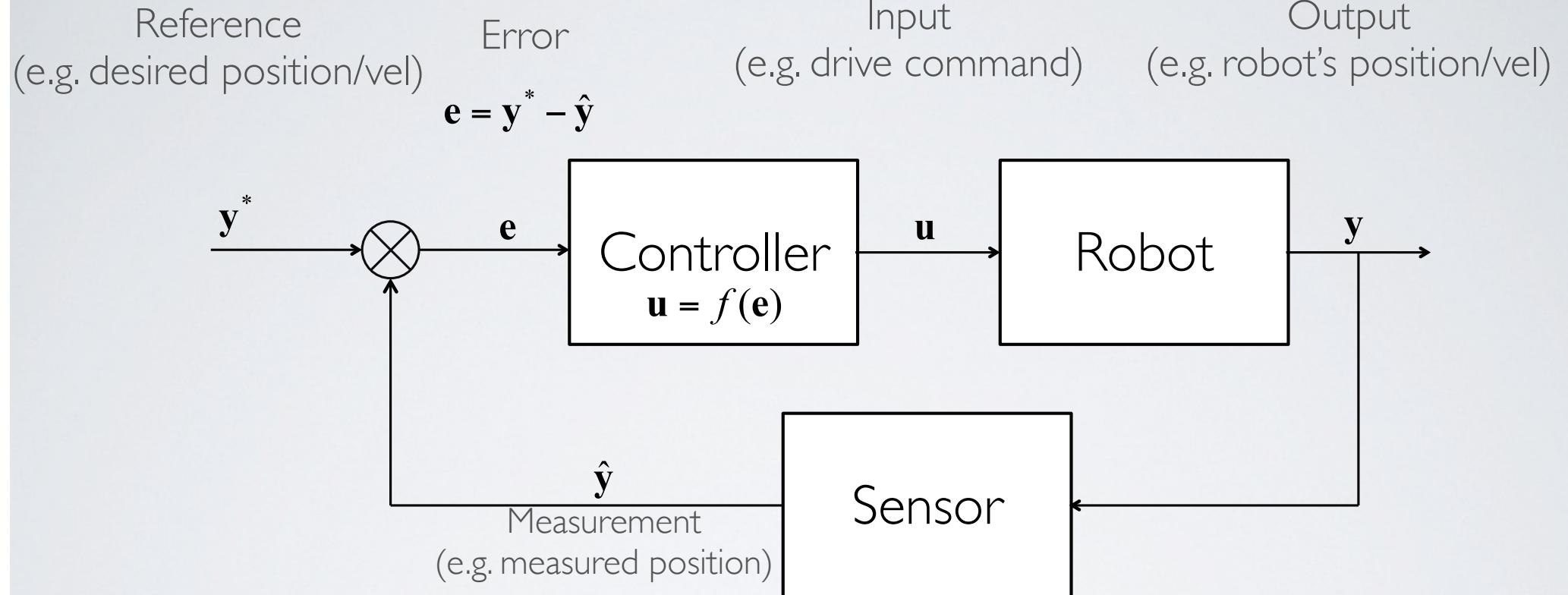
To view

Download the add-in.

liveslides.com/download

Start the presentation.

CLOSED LOOP CONTROL



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

DRIVE DISTANCE - FEEDBACK CONTROL

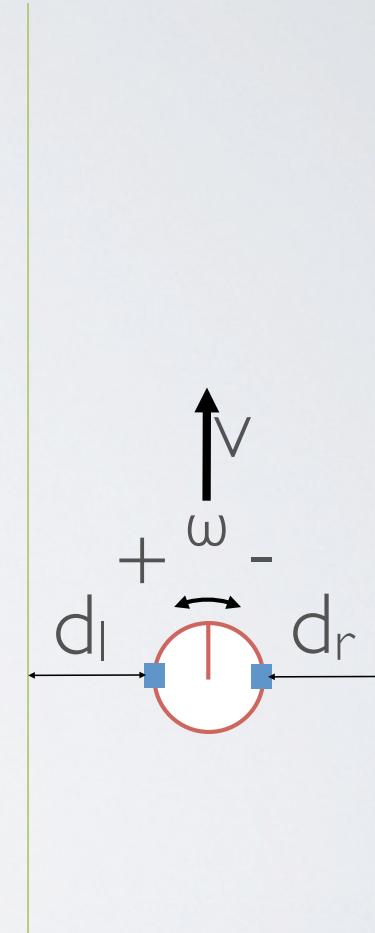
- ▶ Repeat
 - ▶ issue DriveForward command **u**
 - ▶ read odometry (proprioception again) and accumulate the total distance travelled **ŷ**
 - ▶ stop if the total distance is greater than the specified value $\hat{y} \geq y^*$
- ▶ Smaller time intervals – smaller errors

CORRIDOR FOLLOWING

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

▶ Controller task:

- ▶ keep $d_l = d_r$



SIMPLE CONTROLLER

- ▶ Loop:

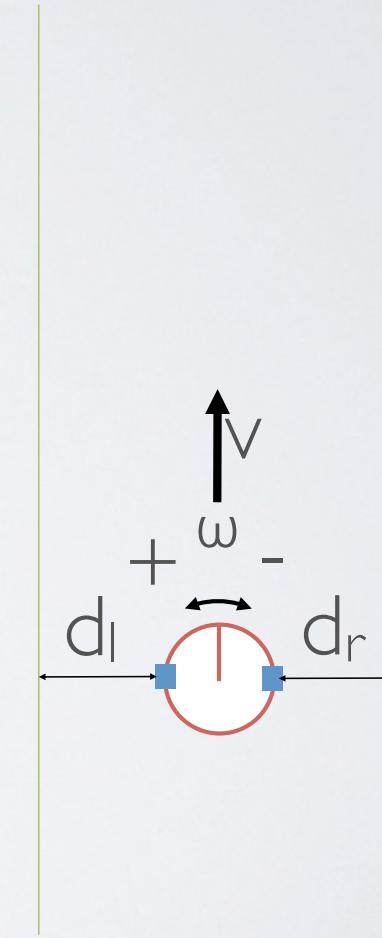
- ▶ measure $e = d_l - d_r$
- ▶ if $e > 0$ then $\omega = +K$
- ▶ if $e < 0$ then $\omega = -K$
- ▶ $\omega = K \text{ sign}(e)$

- ▶ In Rovio language:

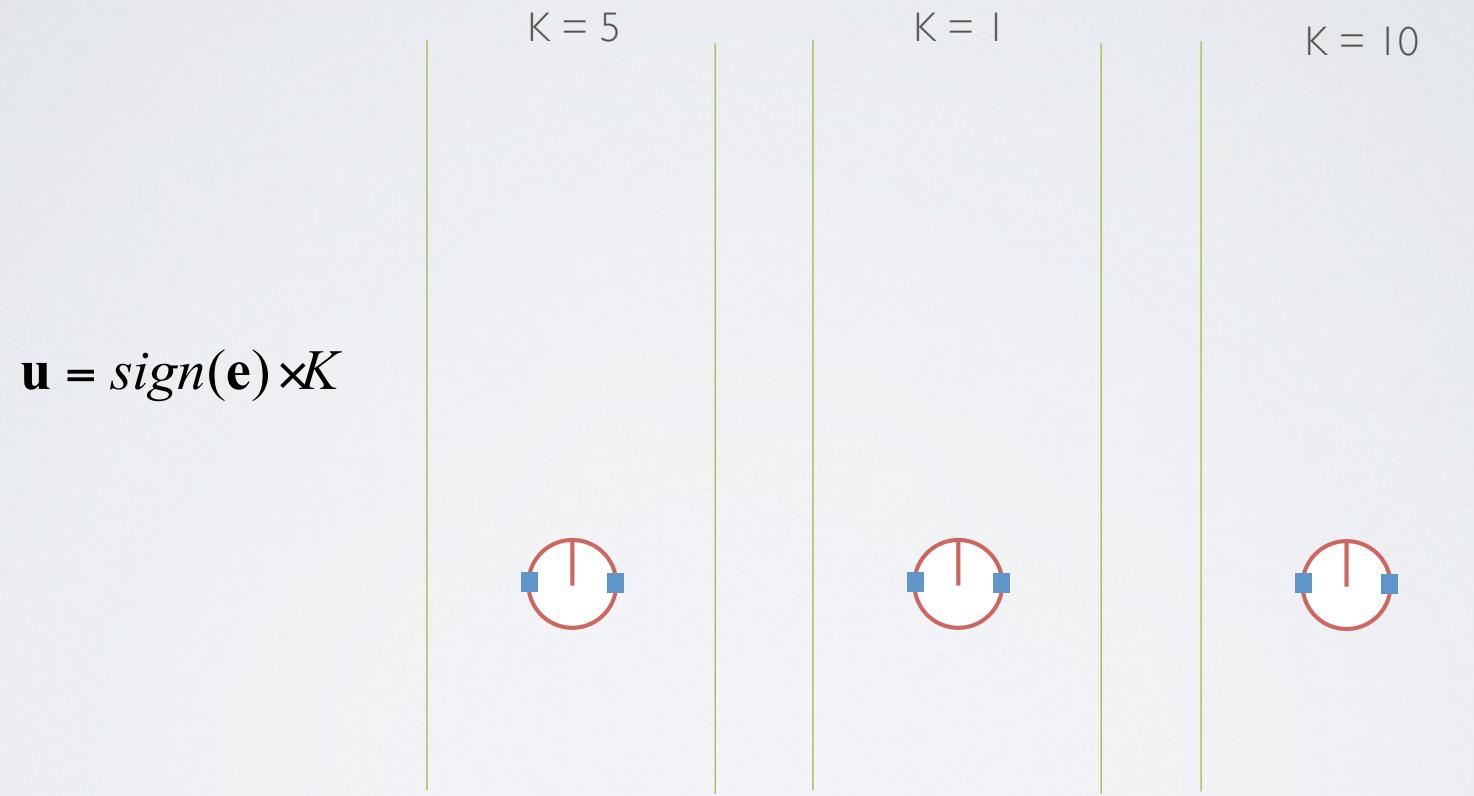
- ▶ if $e > 0$ then RotateLeft(5)
- ▶ if $e < 0$ then RotateRight(5)

constant
parameter

$$K = 5$$



BANG-BANG CONTROLLER



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



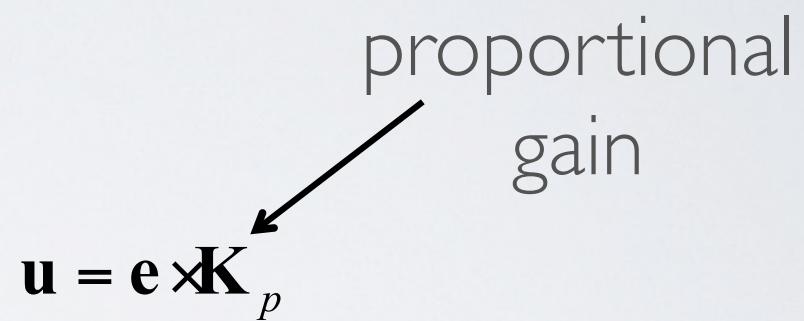
PROPORTIONAL CONTROLLER

- ▶ Change ω proportionally to the error value

- ▶ $\omega = e * K_p$
- ▶ small e – small correction
- ▶ large e – large correction

$$u = e \times K_p$$

proportional
gain



- ▶ Result

- ▶ smoother actions and smaller errors

- ▶ K_p parameter

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

VISION-BASED CONTROL

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

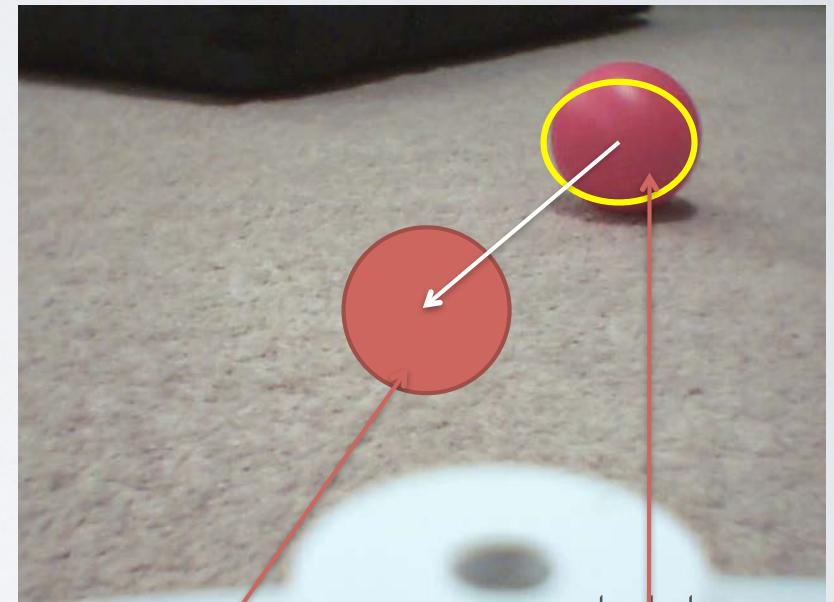
$$\mathbf{y} = \begin{bmatrix} x \\ w \times h \end{bmatrix}$$

- ▶ Error - difference between the desired and current state

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

- ▶ Control input u:

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



desired state

current state:
information from
the object detector

FOR MOTION CONTROL

Model



Kinematics / Dynamics

Algorithm



control-parameter

Engineering

control signal => actuators

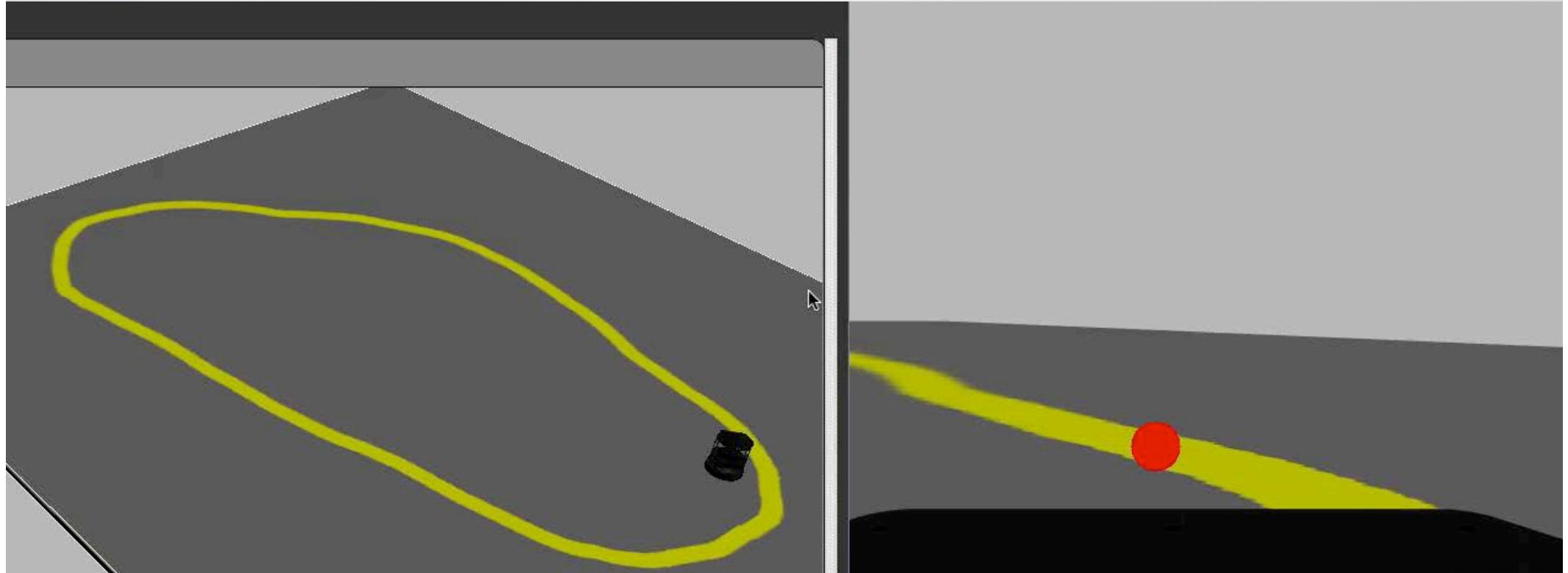


LUND
UNIVERSITY

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

Edited by Gunnar Bolmsjö
2011

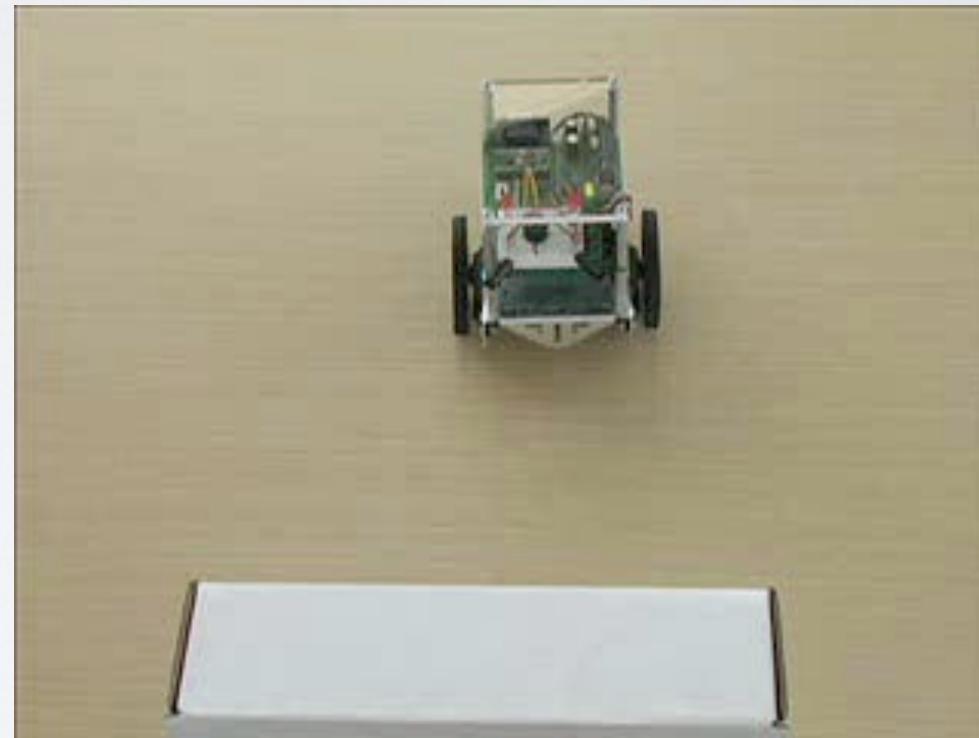
NOW FOR THE REAL STUFF



Morgan Quigley, Brian Gerkey & And William D. Smart. (2015)
Programming Robots with ROS [Chapter 12]

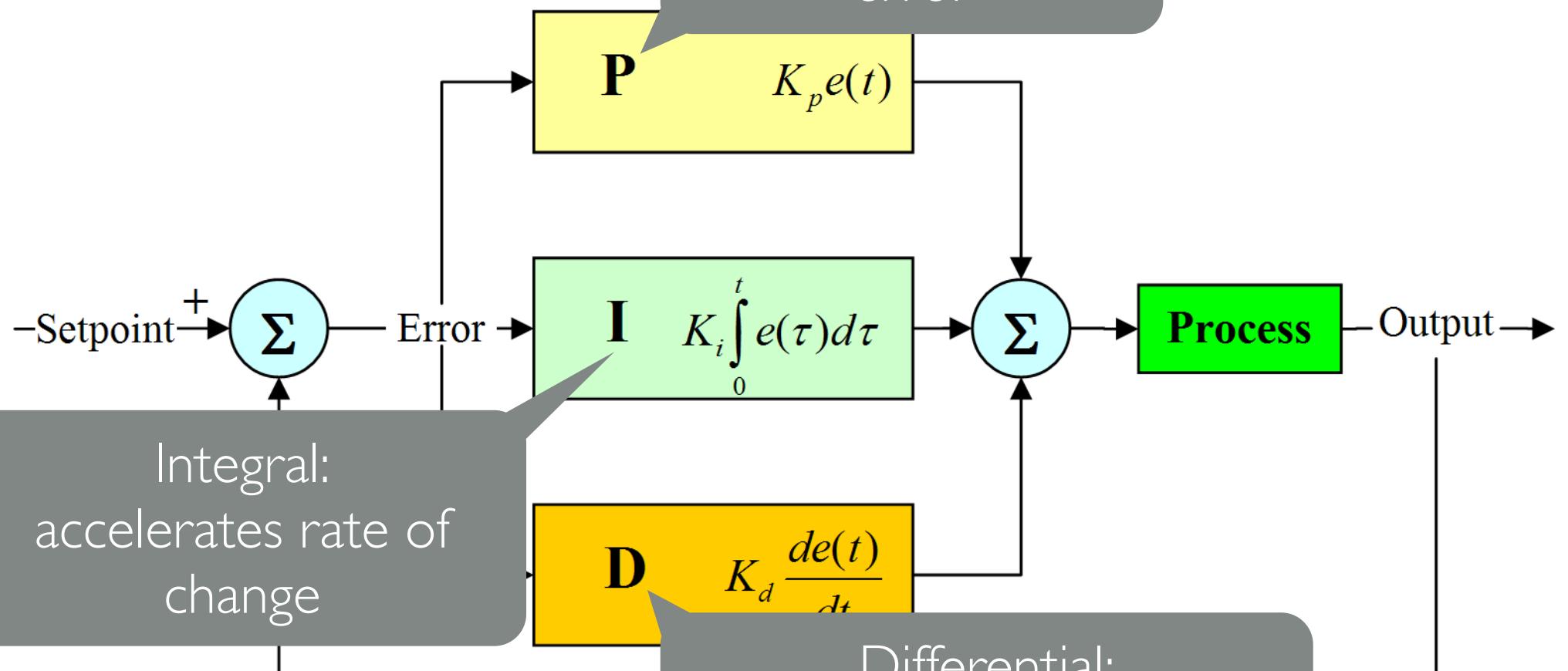
PROBLEMS WITH PROPORTIONAL CONTROLLER?

- ▶ What's the problem?



PID CONTROLLER

Proportional:
corrects the actual
error

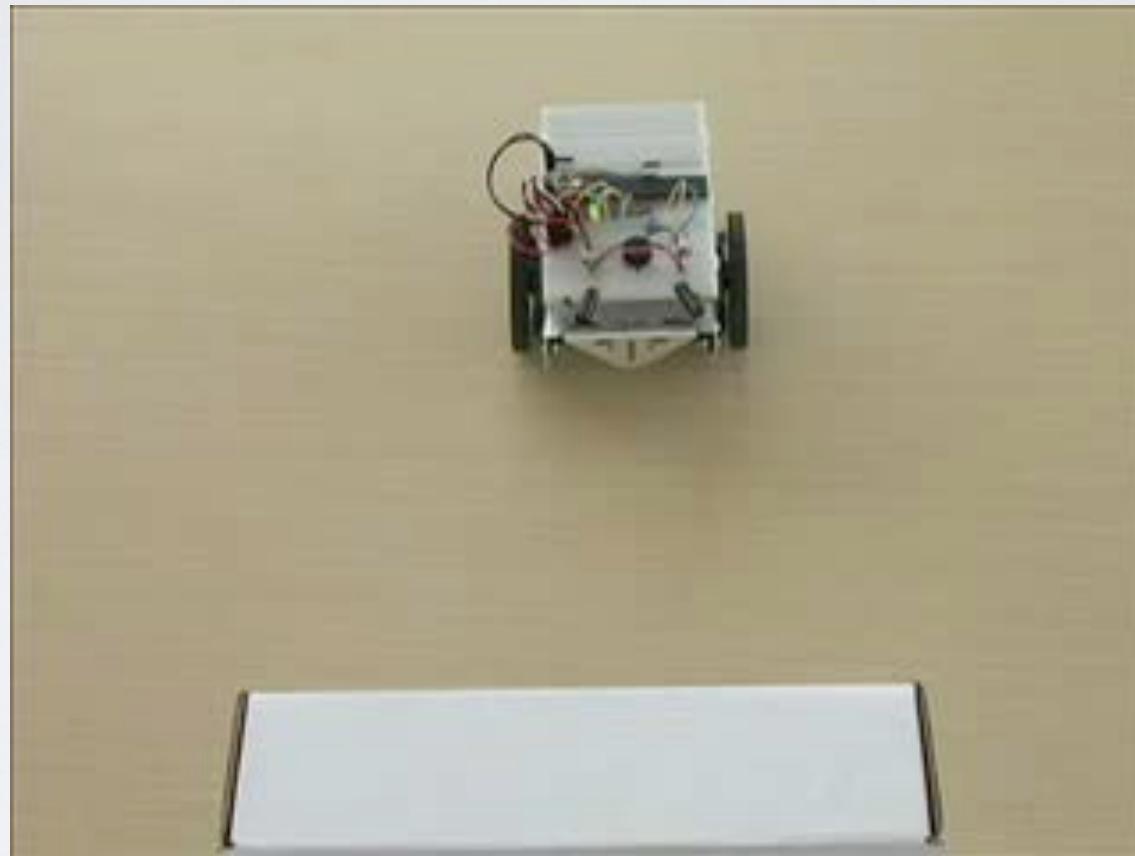


Integral:
accelerates rate of
change

Differential:
slows the rate of change
(reduces overshooting)

© SilverSTart@Wikipedia

PID CONTROLLER



APPLICATIONS – EXAMPLES

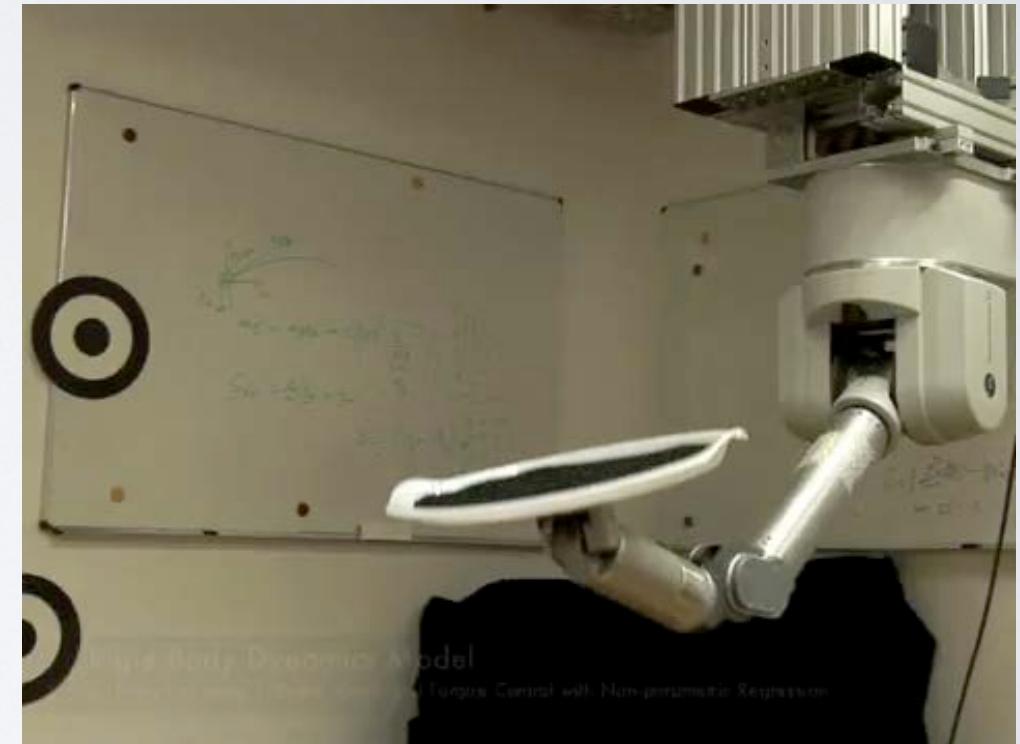
- ▶ Balancing Robot

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

- ▶ Person Following



ANY OTHER CONTROL PROBLEMS YOU COULD THINK OF IN ROBOTICS?



CONTROL THEORY

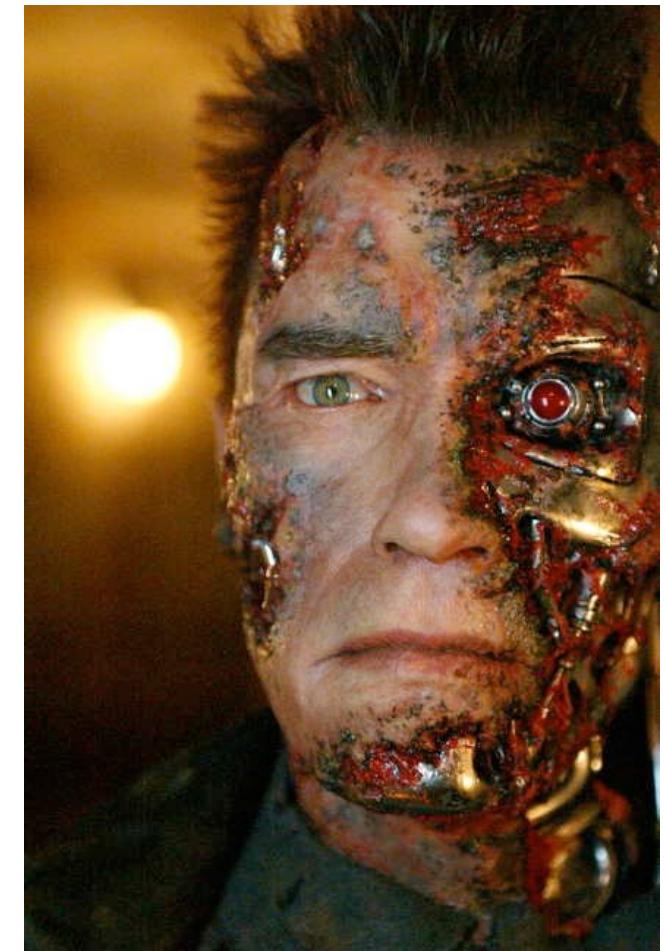
- ▶ Optimal control, minimise
 - ▶ errors
 - ▶ energy
 - ▶ response time
- ▶ Design and analysis of controllers
 - ▶ linear (P, PD, PID controllers)
 - ▶ non-linear
 - ▶ adaptive controllers
 - ▶ changing parameter values in time
 - ▶ models
 - ▶ kinematics and dynamics in robotics

read Siegwart book, chapter 4!

End of Lecture Feedback

When survey is active, respond at PollEv.com/mhanheide

THANK YOU
FOR LISTENING!



0 surveys done

⟳ 0 surveys underway

Start the presentation to see live content. Still no live content? Install the app or get help at PollEv.com/app

CMP3103M

Autonomous Mobile Robotics

Lecture 5: Robot behaviour

Dr. Athanasios Polydoros

apolydoros@lincoln.ac.uk

Based on slides of Dr. A.Millard

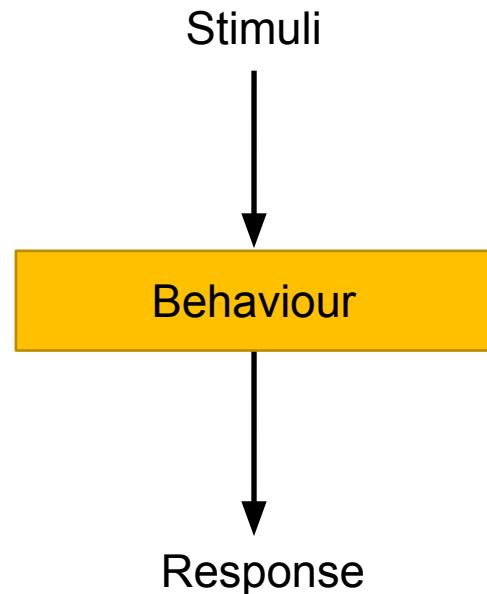
Today's lecture

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

Robot behaviours

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”

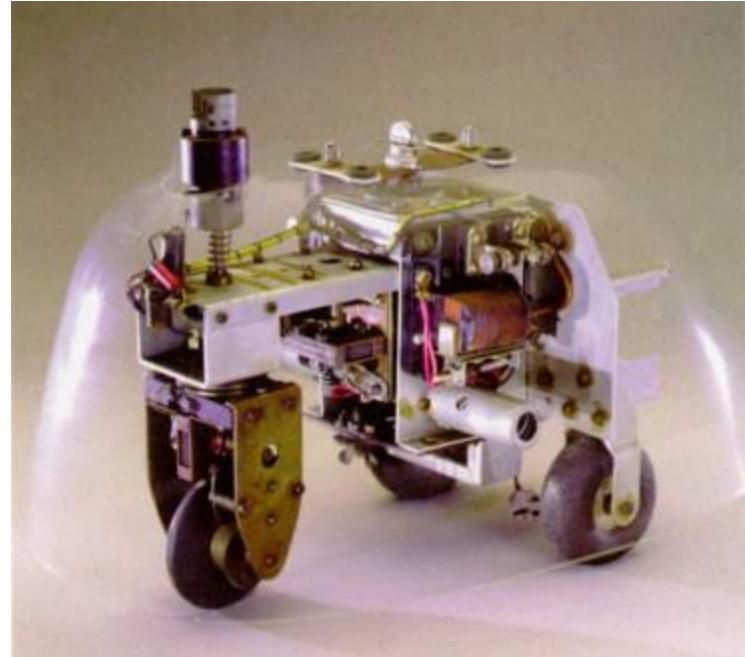


Robot behaviours

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

- Neurophysiologist / cybernetician
 - Experiments in reflex behaviour
- *Machina Speculatrix*, 1948
 - Analogue circuit built from vacuum tubes
 - Follow light (photocell) and avoid obstacles (bump sensor)



Tortoise robot behaviours

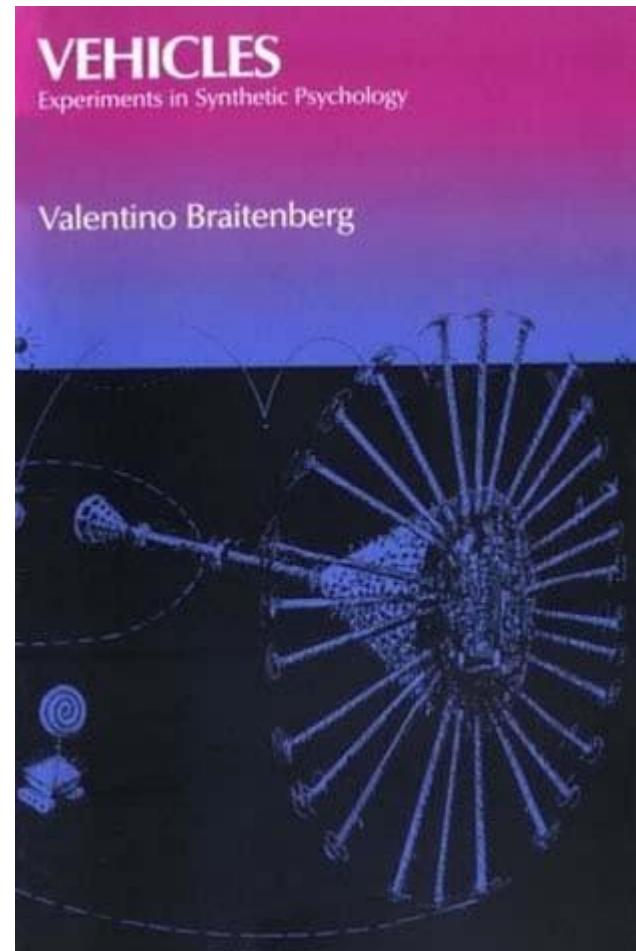
- Seek light (exploration)
- Move towards / back away from light
- Avoid obstacles
- Recharge battery

Reactive control

- Tight coupling of sensors and actuators
- No internal model of the world
- Well-suited to dynamic/unstructured environments

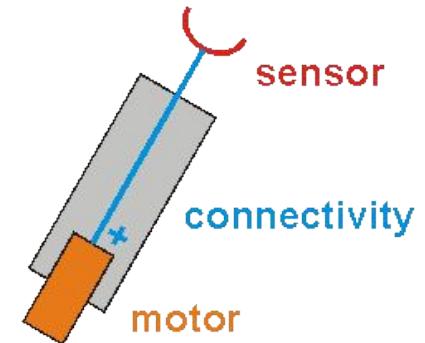
Braitenberg vehicles

- 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

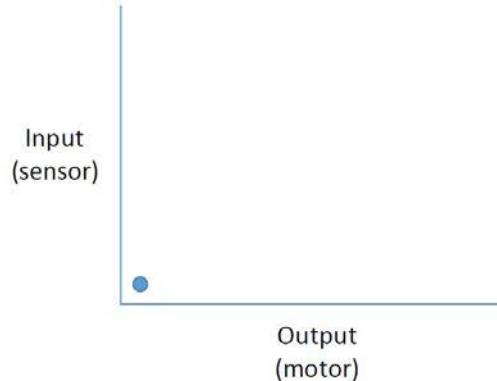
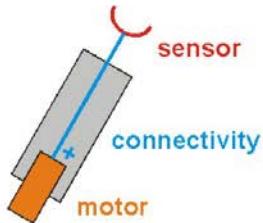


Vehicle 1

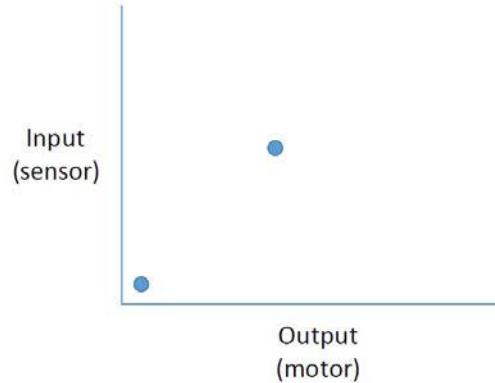
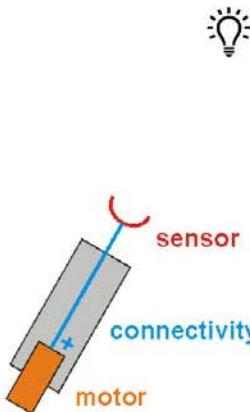
- One sensor
 - e.g. light, temperature, sound
- One motor
 - Directly driven by sensor
 - +/- indicates excitatory/inhibitory relationship
- Motor speed is **proportional** to sensor reading
 - e.g. greater light intensity == faster motor speed



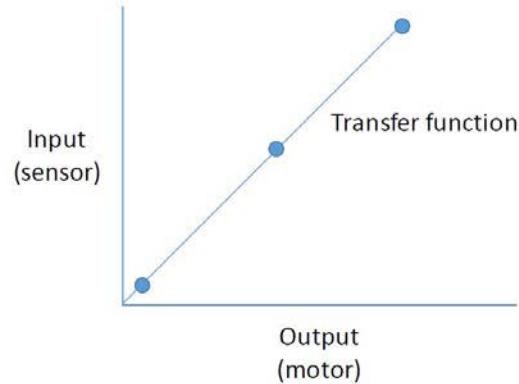
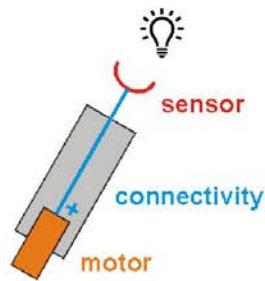
Vehicle 1



Vehicle 1

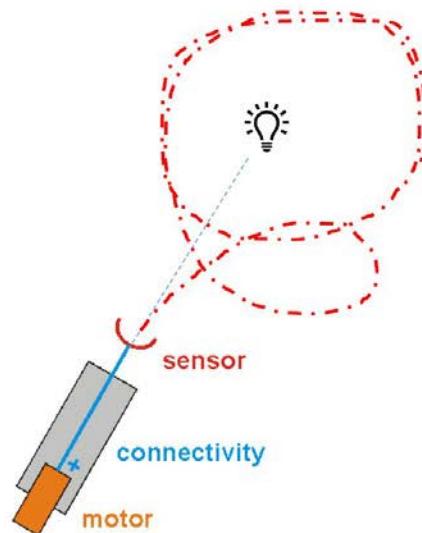


Vehicle 1



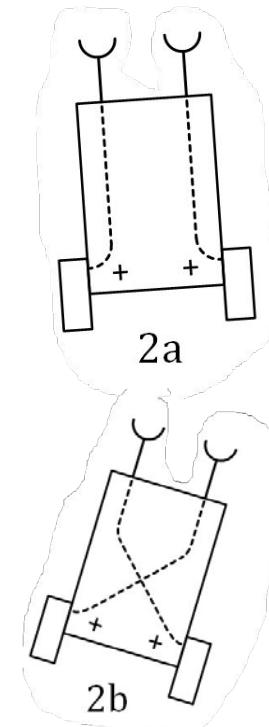
Vehicle 1

- Under perfect conditions
 - Vehicle stops on dark spots
- More realistic if physics are added
 - e.g. non-symmetric drive or friction
- Interesting emergent behaviour
 - Looping continuously around sources
 - Brownian motion at group level



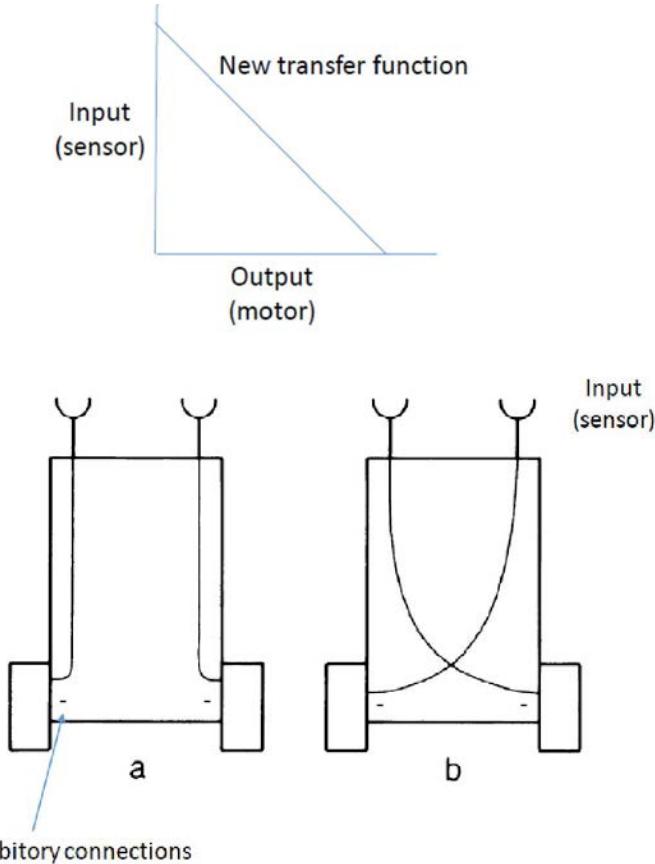
Vehicles 2a and 2b

- Two sensors, two motors
 - Directly coupled
- Vehicle 2a
 - Sensors linked to motors on same side
 - Excitatory connections
- Vehicle 2b
 - Sensors linked to motors on opposite side
 - Excitatory connections



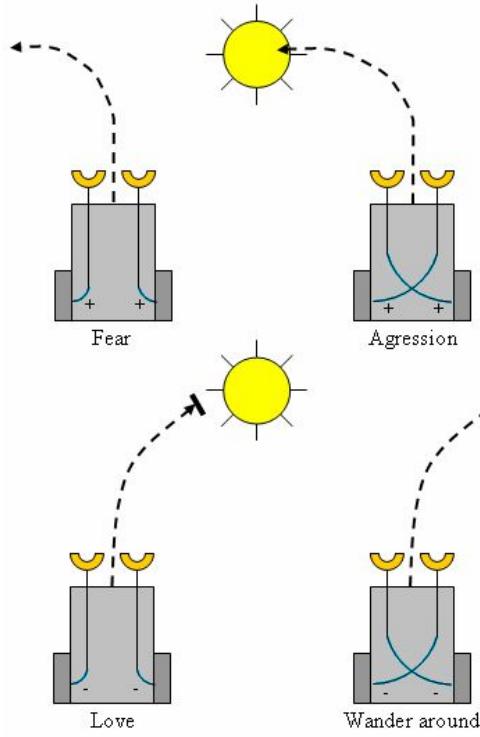
Vehicles 3a and 3b

- 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



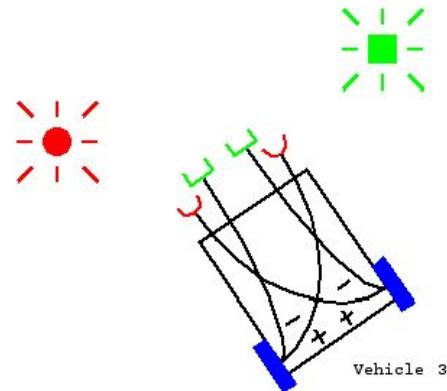
Braitenberg Vehicles Behaviours

- 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



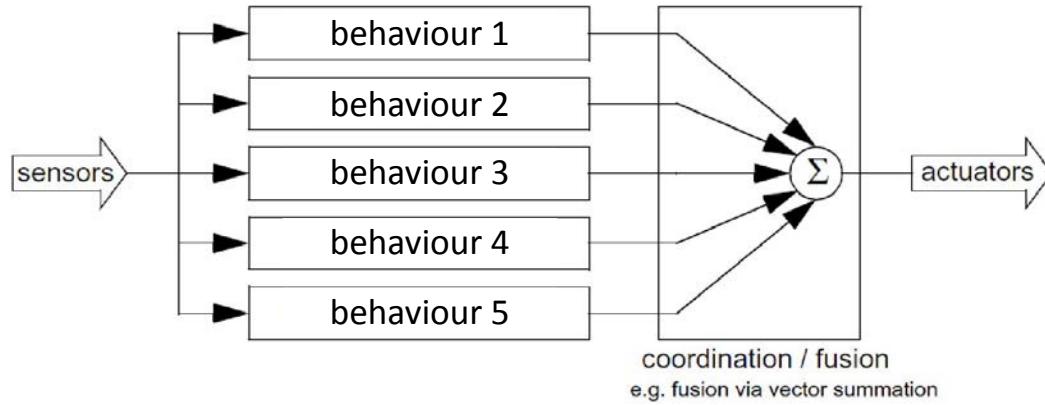
Multi-sensory vehicles

- 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

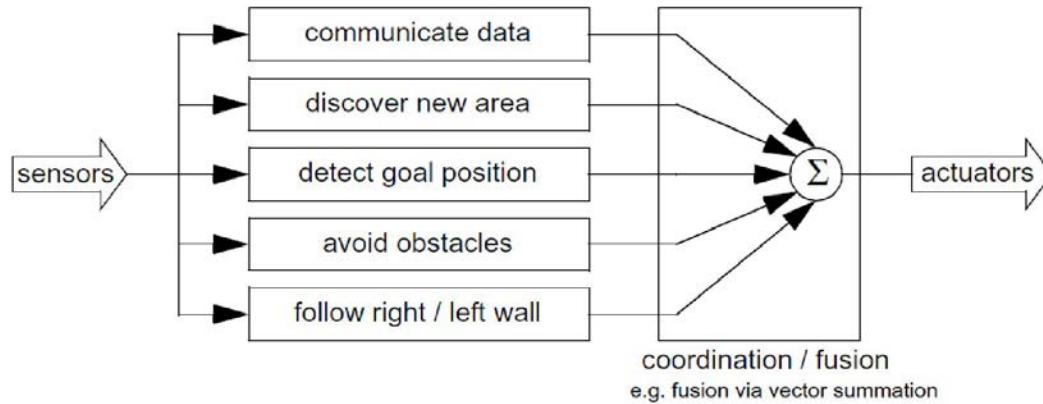


Combining robot behaviours

Multi-behaviour vehicles

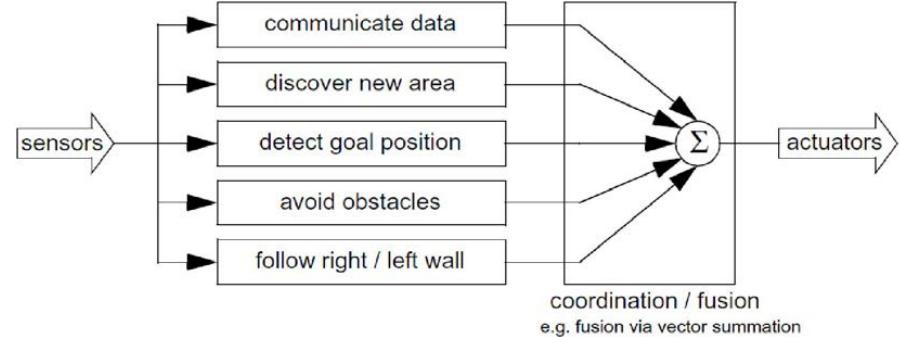


Multi-behaviour vehicles



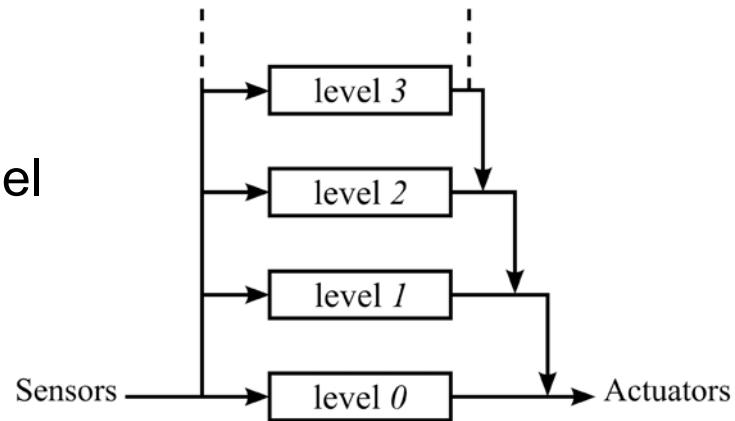
Vector summation

- 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**?



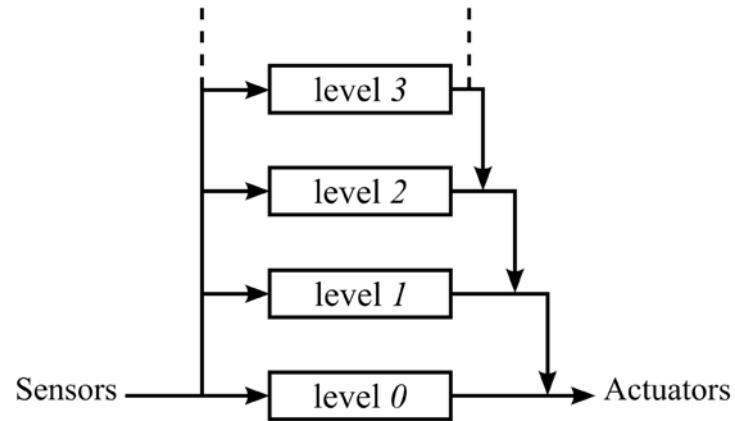
Subsumption architecture (1986)

- Behaviour decomposed into separate layers
- Priority-based hierarchy
 - Simplest low-level behaviours at the bottom
 - Complexity increases with higher layers
- Layers operate asynchronously in parallel

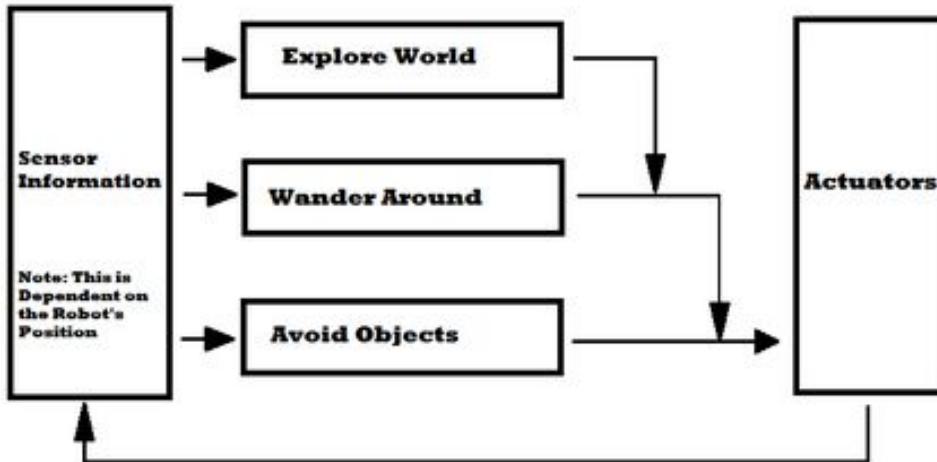


Subsumption architecture (1986)

- Principles of design:
 - systems are built from the bottom
 - layers are task achieving actions/behaviors (avoid obstacles, find-doors, visitrooms)
 - 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

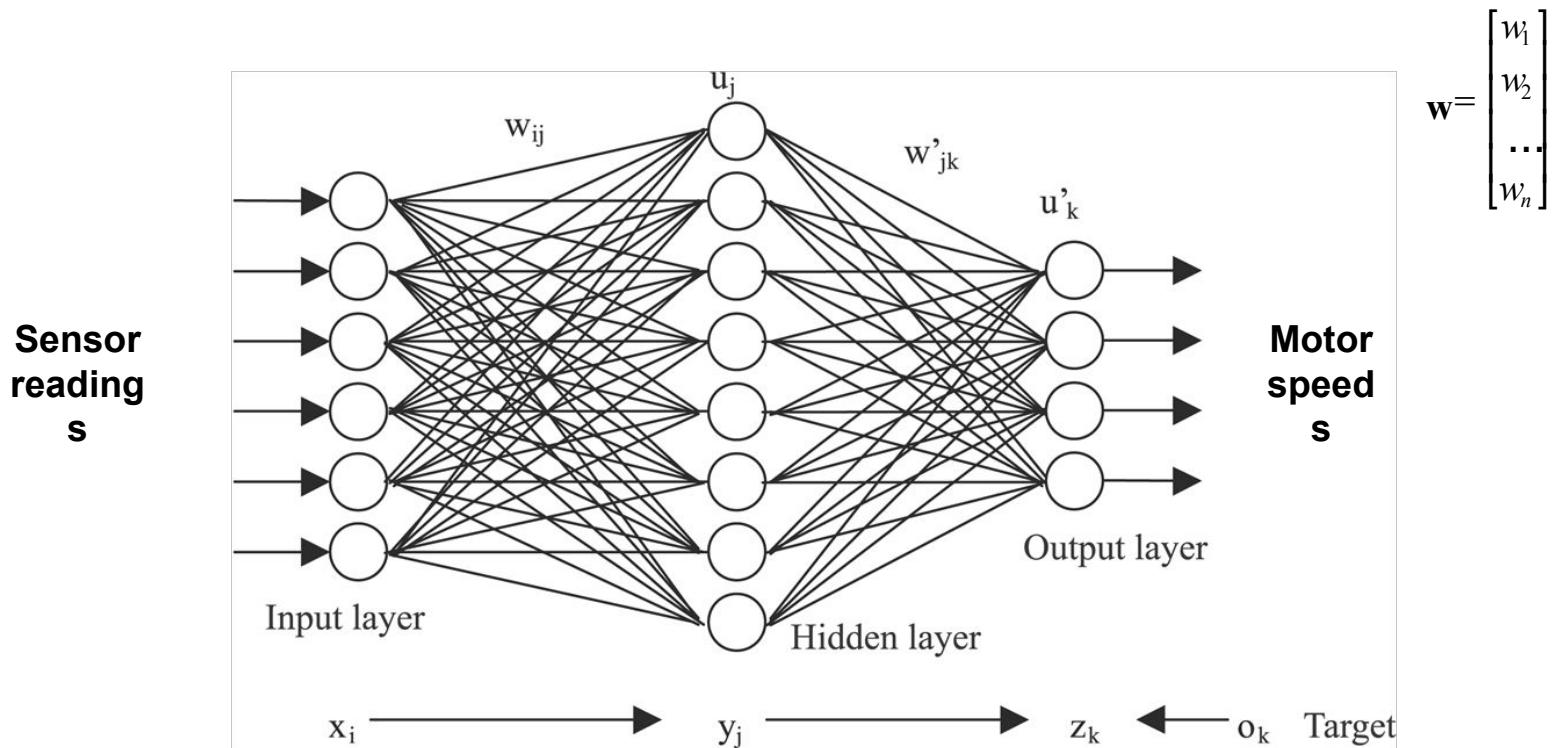


Subsumption architecture (1986)



Learning robot behaviours

Artificial Neural Networks (ANNs)



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

Robot behaviour learning

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

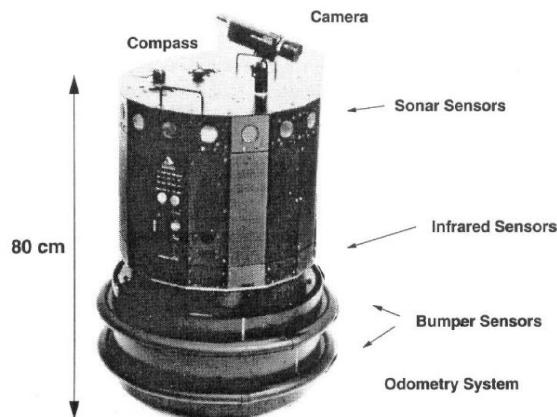


FIG. 3.17. THE NOMAD 200 MOBILE ROBOT *FortyTwo*

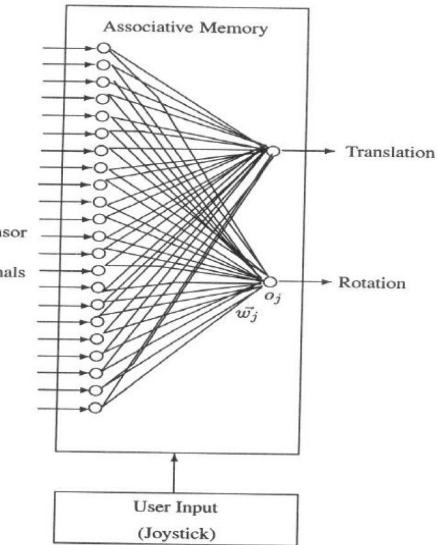


FIG. 4.15. THE CONTROLLER ARCHITECTURE

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

Evolutionary robotics

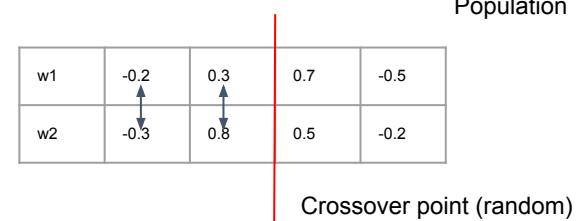
- Robot self-learns behaviours
- Genetic Algorithm
 - Black-box optimization of a fitness function
 - Inspired from evolution theory
- Other evolutionary algorithms:
 - Ant colony optimization
 - Particle swarm optimization

Evolutionary robotics - Genetic Algorithm

- Phases of a Genetic Algorithm

- Fitness function: the function we want to optimize
- Initial population:
 - A set of multiple 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)

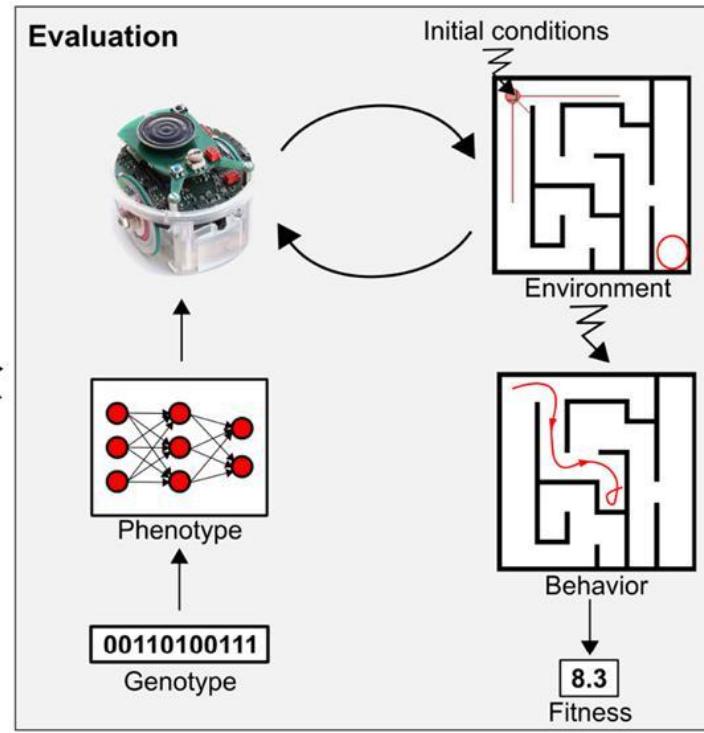
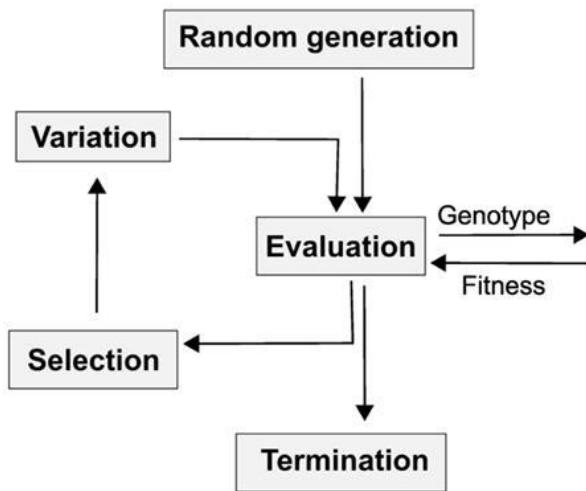
Gene				
w1	-0.2	0.3	0.7	-0.5
w2	-0.3	0.8	0.5	-0.2
w3	0.2	0.3	-0.7	-0.5



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

New offsprings

Evolutionary robotics

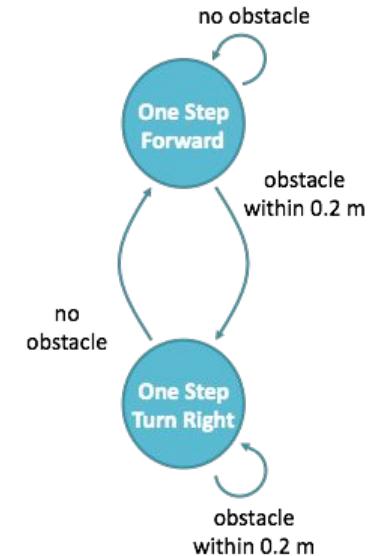


Explicitly coding behaviours

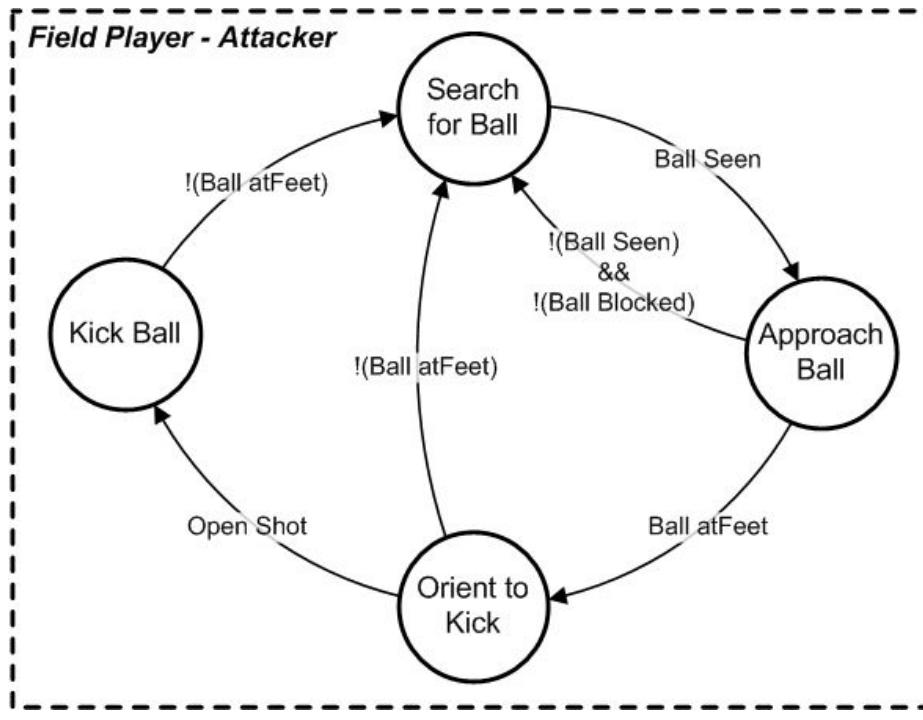
Finite State Machine (FSM)

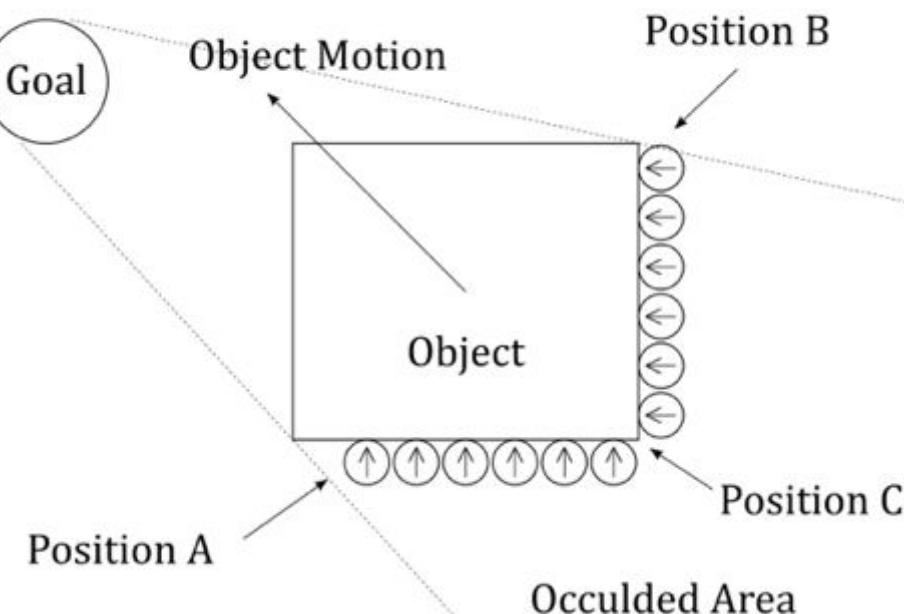
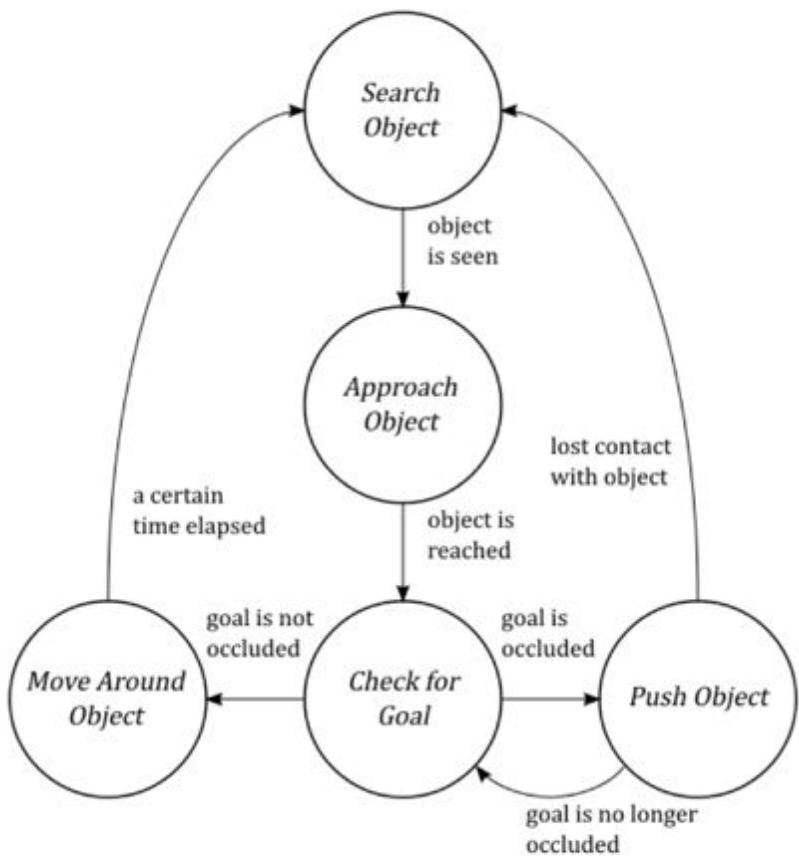
Simple obstacle avoidance using IR sensor

- Collection of states/actions
 - States: Nodes
 - Transitions: Arrows
- Robot always in some defined **state** (behaviour)
 - Exploring, avoiding, waiting, etc
- Robot **conditionally** transitions between states
 - e.g. If sensor reading > threshold, then transition

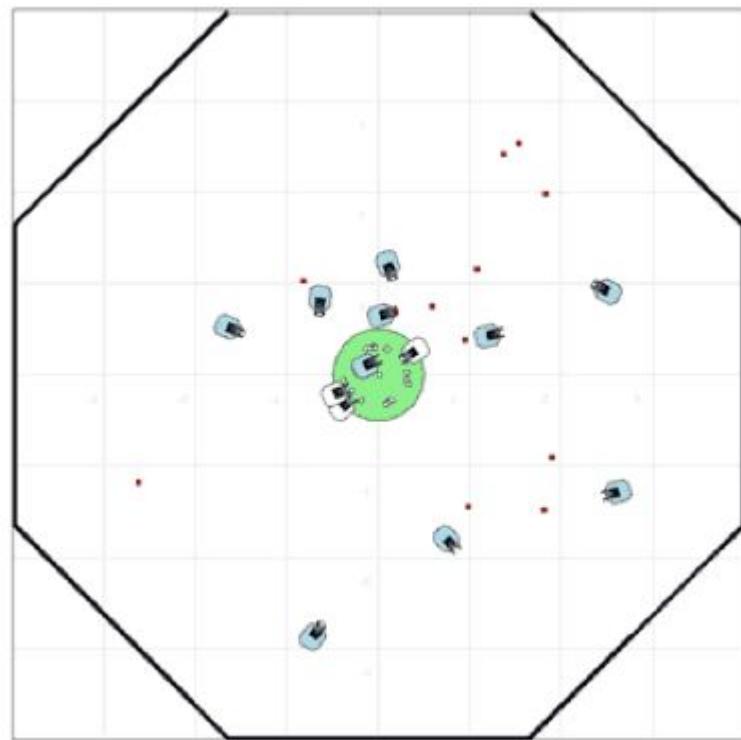
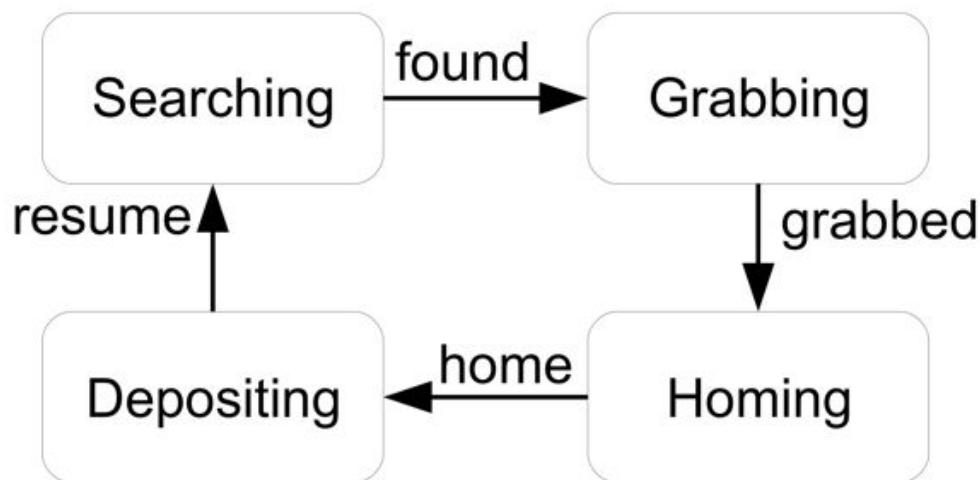


Finite State Machine (FSM)





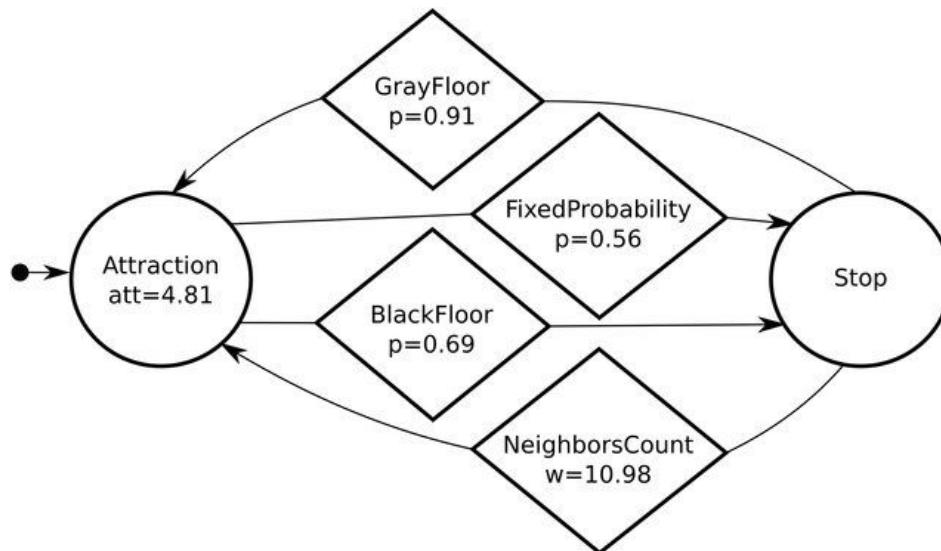
Foraging example



Probabilistic Finite State Machine (PFSM)

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

Probabilistic Finite State Machine (PFSM)

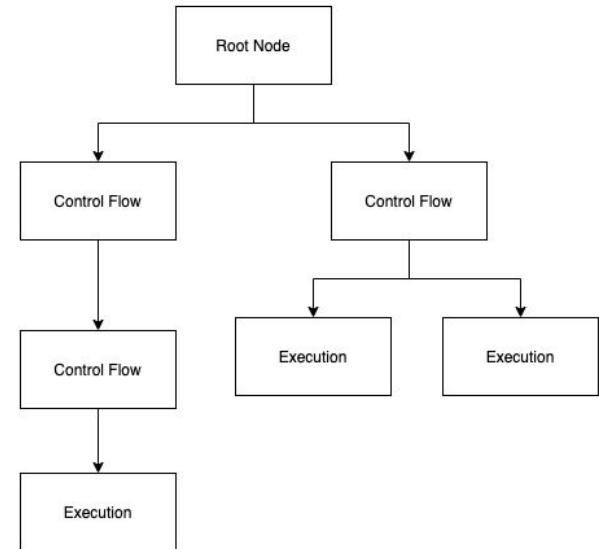


Behaviour trees

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

Behaviour trees – Nodes

- Control Flow Nodes:
 - The internal nodes
- Execution Nodes:
 - The leaf nodes
- Nodes can return the following states:
 - Running
 - Success
 - Failure



<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):**

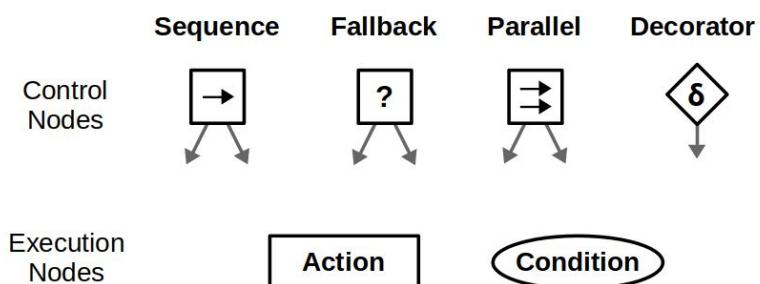
Modifies the return of a node

- **Action:**

Performs a defined task

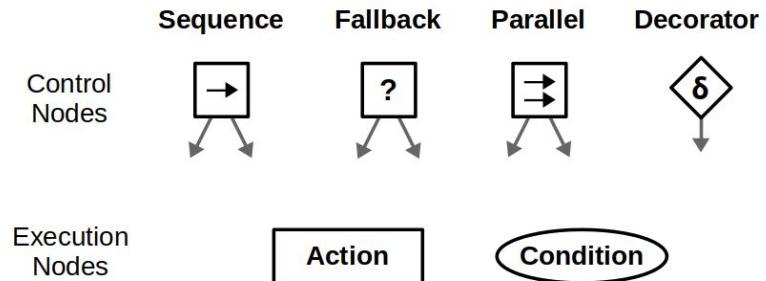
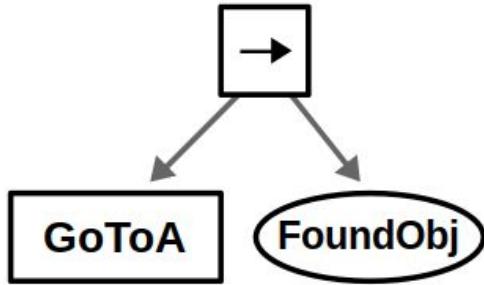
- **Condition:**

Evaluates a condition



Behaviour trees – Example

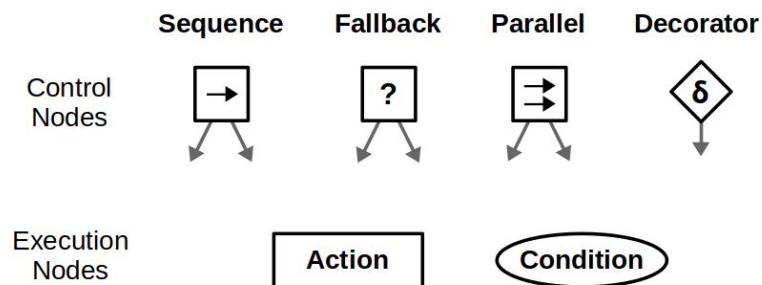
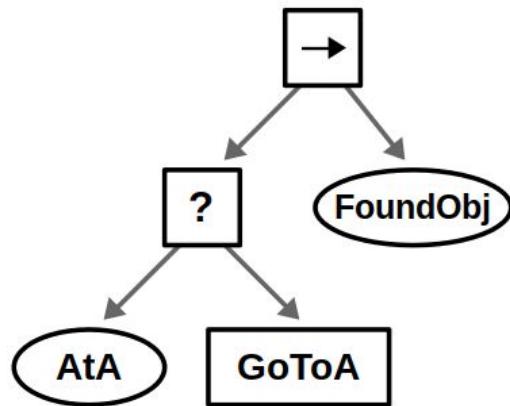
- Task: Search for an object at various locations



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

Behaviour trees – Example

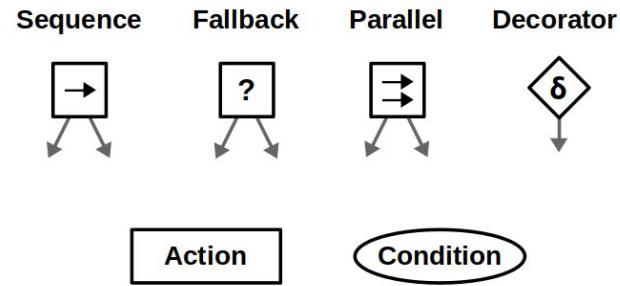
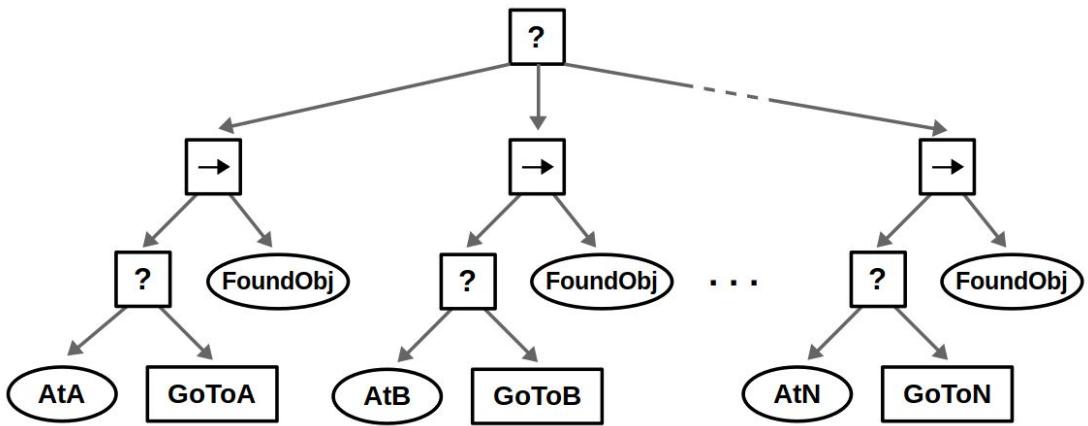
- Task: Search for an object at various locations



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

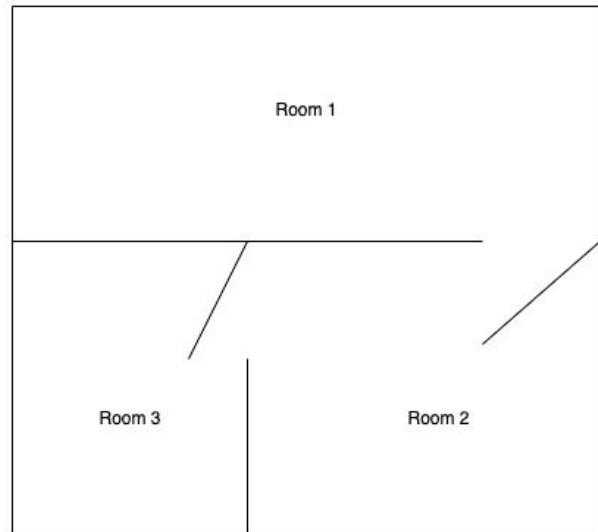
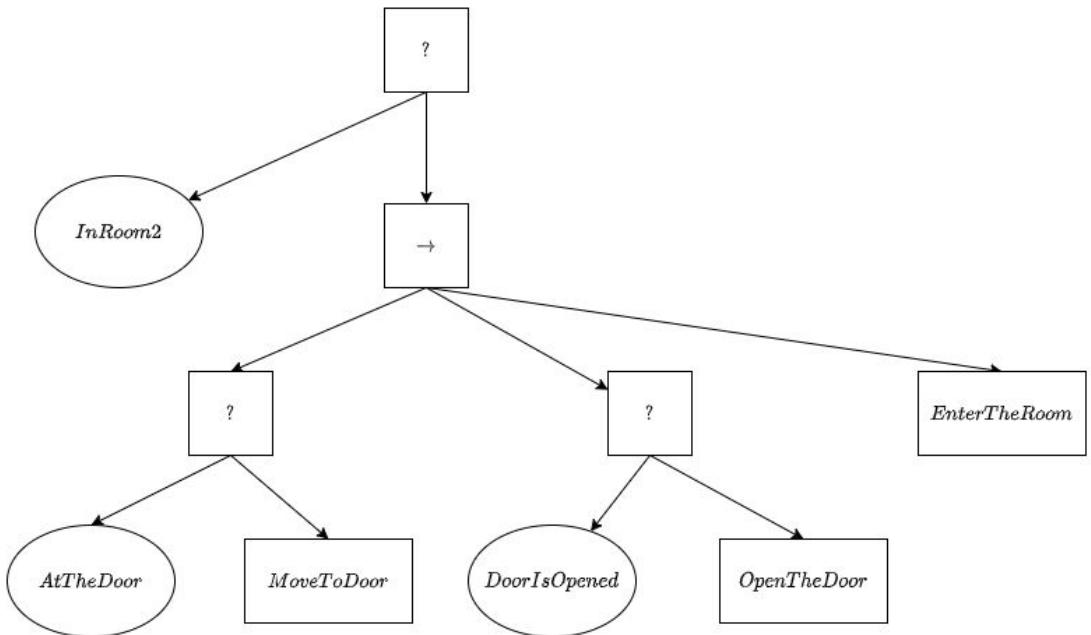
Behaviour trees – Example

- Task: Search for an object at various locations

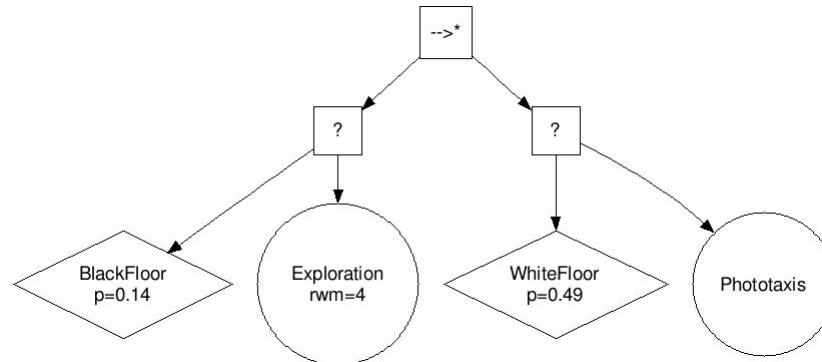
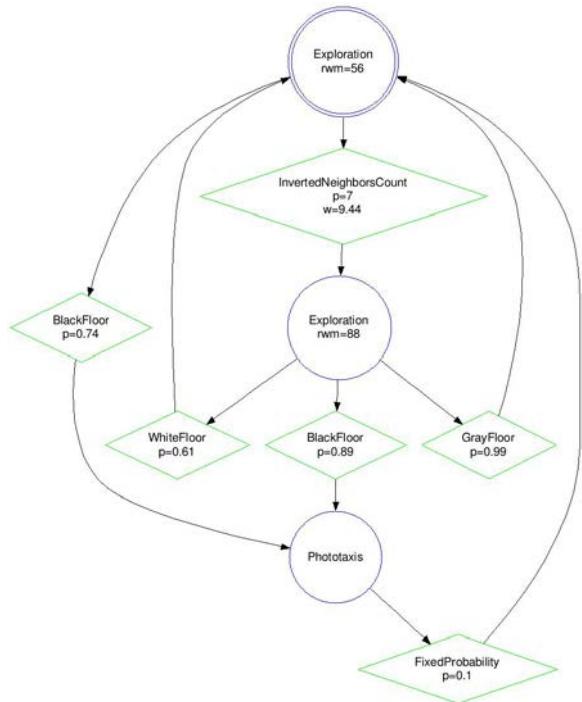


<https://robohub.org/introduction-to-behavior-trees/>

Behaviour trees



Synthesised PFSM vs Behaviour tree

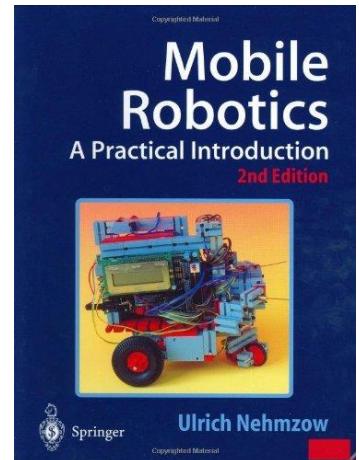
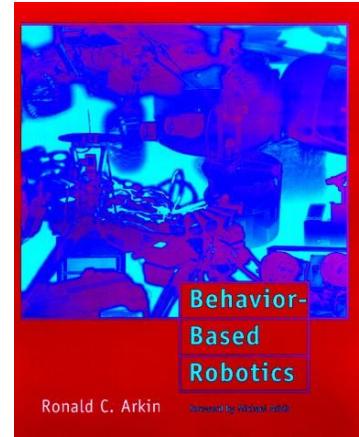


Summary

- What is behavioural robotics
- Braitenberg vehicles:
 - Achieving behaviours by coupling sensors and motors
- Learning robot behaviours
 - Supervised Learning
 - Self learning with evolutionary algorithms
- Combining robot behaviours
- Automatic composition of robot behaviours
 - Behaviour Trees
 - Probabilistic FSM

Recommended reading

- Ronald C. Arkin, “Behavior-Based Robotics”,
MIT Press, 1998
- Ulrich Nehmzow, “Mobile Robotics: A
Practical Introduction”, Springer, 2nd edition,
2002



Next Lecture – Navigation

- Overview
- Global vs local approaches
- Odometry (dead reckoning)
- Navigation strategies

CMP3103M

Autonomous Mobile Robotics

Lecture 6: Navigation

Dr Athanasios Polydoros

apolydoros@lincoln.ac.uk

Based on the slides of Dr. Alan Millard

Today's lecture

- Quiz
- Navigation overview
- Global vs local approaches
- Odometry (dead reckoning)
- Navigation strategies

Interactive Quiz:

Join:



<https://pollev.com/athanasiospolydoros472>

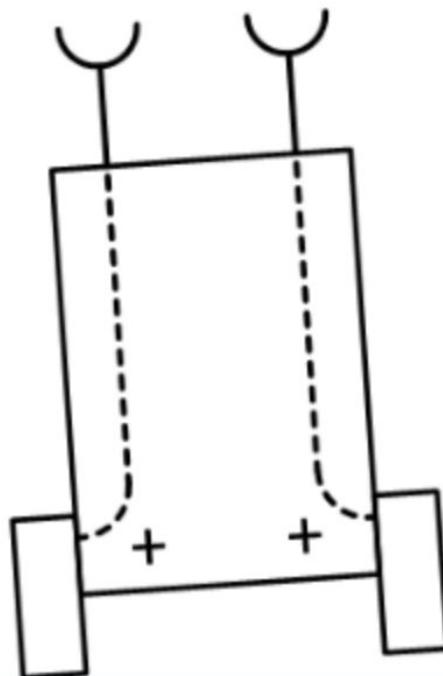
Reactive behaviours of robot require the definition of a goal/plan

True

False



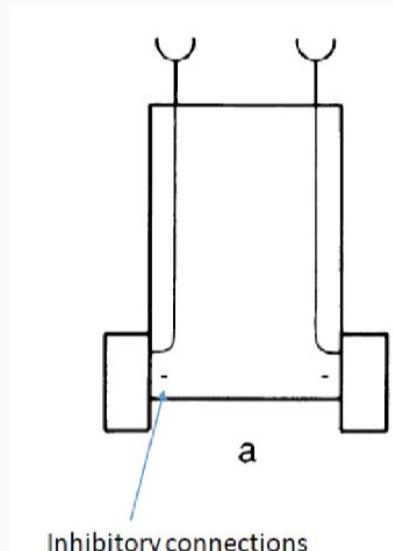
What would be the behaviour of the illustrated Braitenberg vehicle?



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

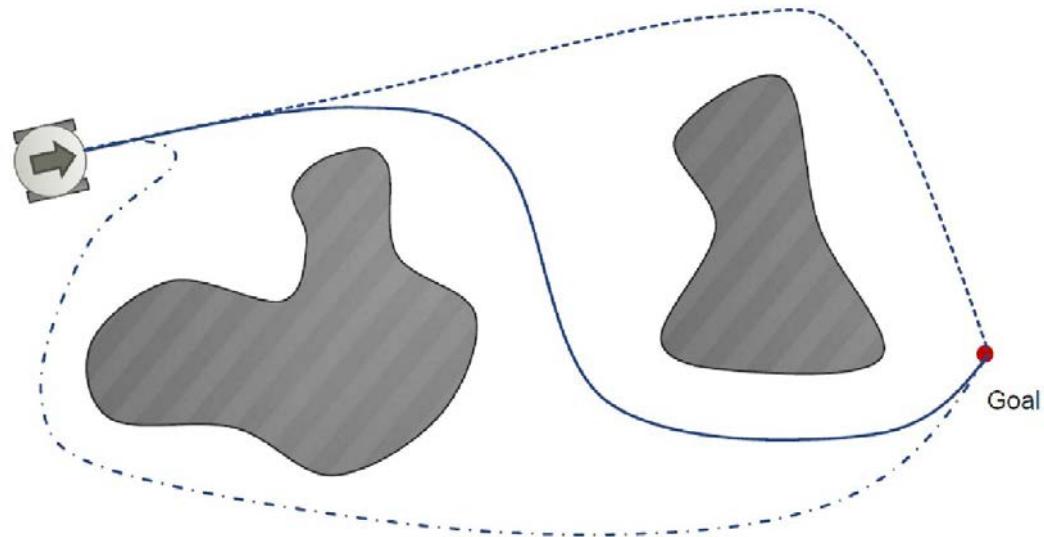
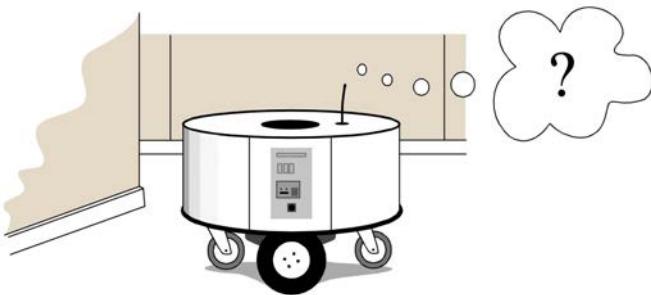
Finite
State
Machines

Behaviour
Trees



Navigation – key questions

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

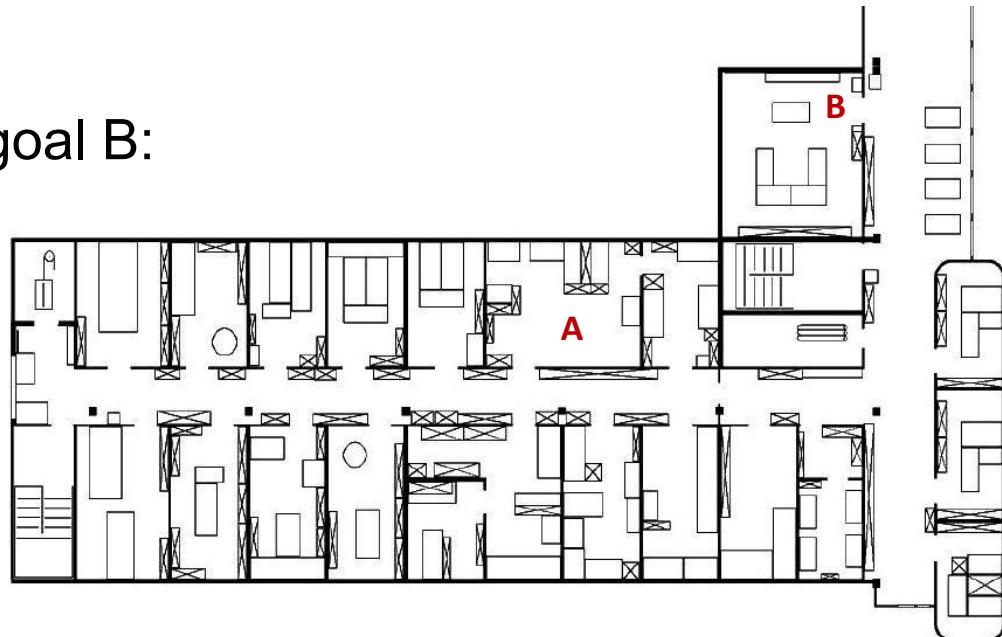


Getting from A to B

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

- Task – start at A and reach goal B:

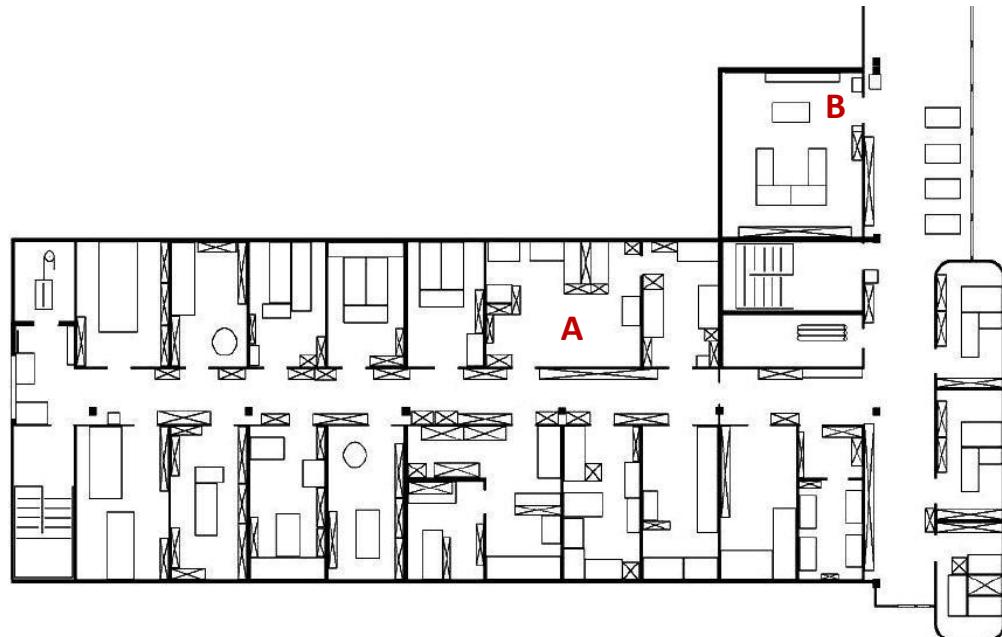
- Without hitting obstacles
- Detect arrival at goal



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?

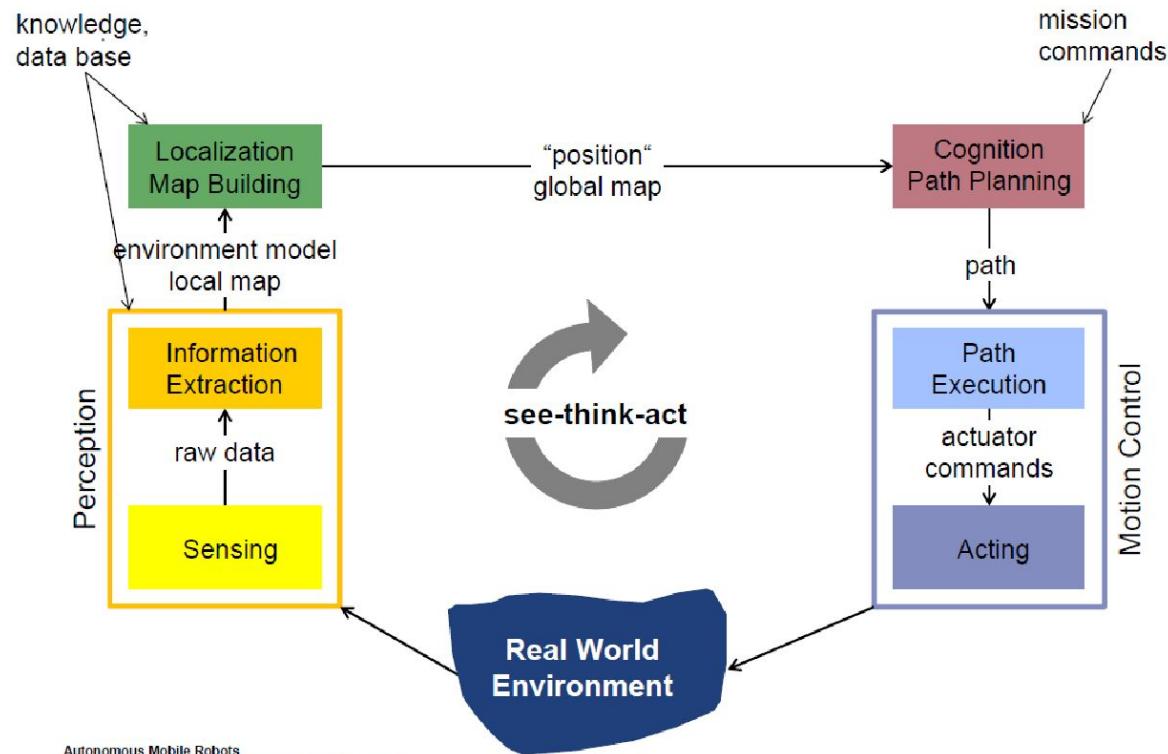


Robot navigation

To navigate successfully, 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



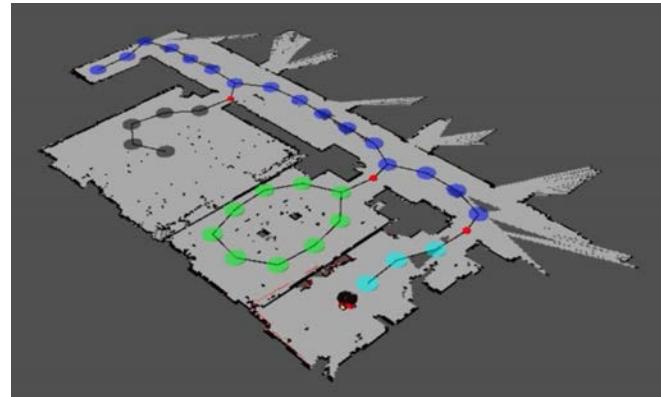
Global vs local approaches

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

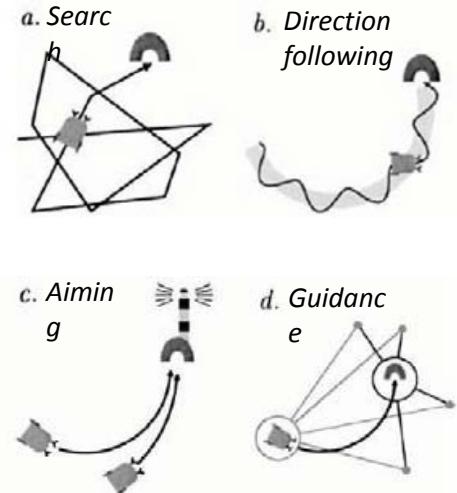
- e.g. compass direction, or trail following
- Finds goal from one direction

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)

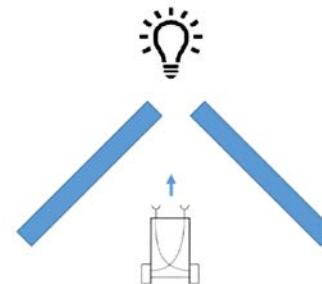
Local strategies using vision

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



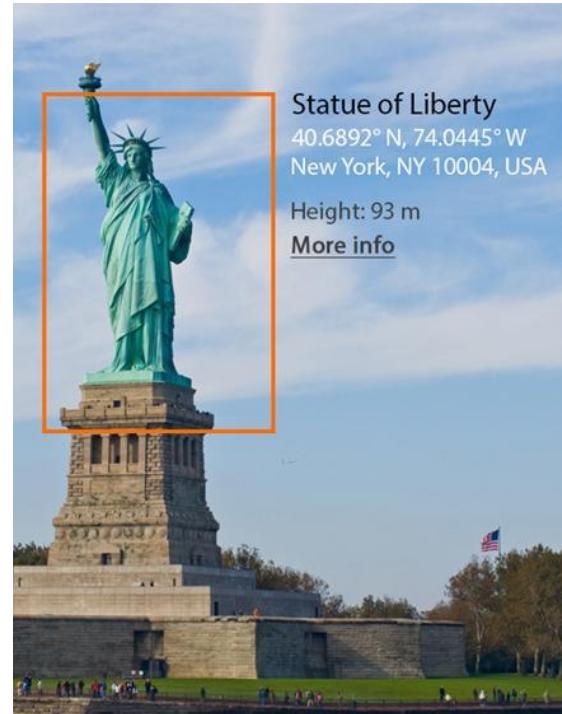
Beaconing problems

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



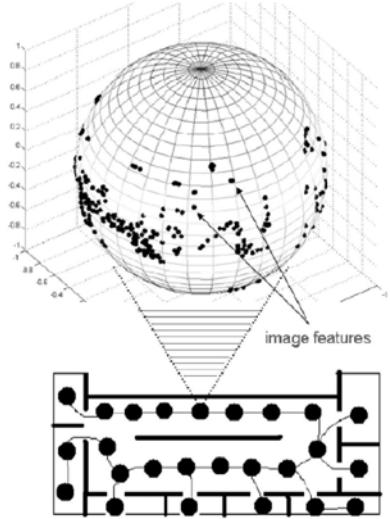
Problems with landmark recognition

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



Feature based recognition

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



From local strategies to routes

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



Landmark recognition along routes

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

Odometry

Odometry / dead reckoning

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

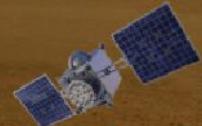
Example : piloting barren Martian surface
with no external guidance cues e.g. GPS



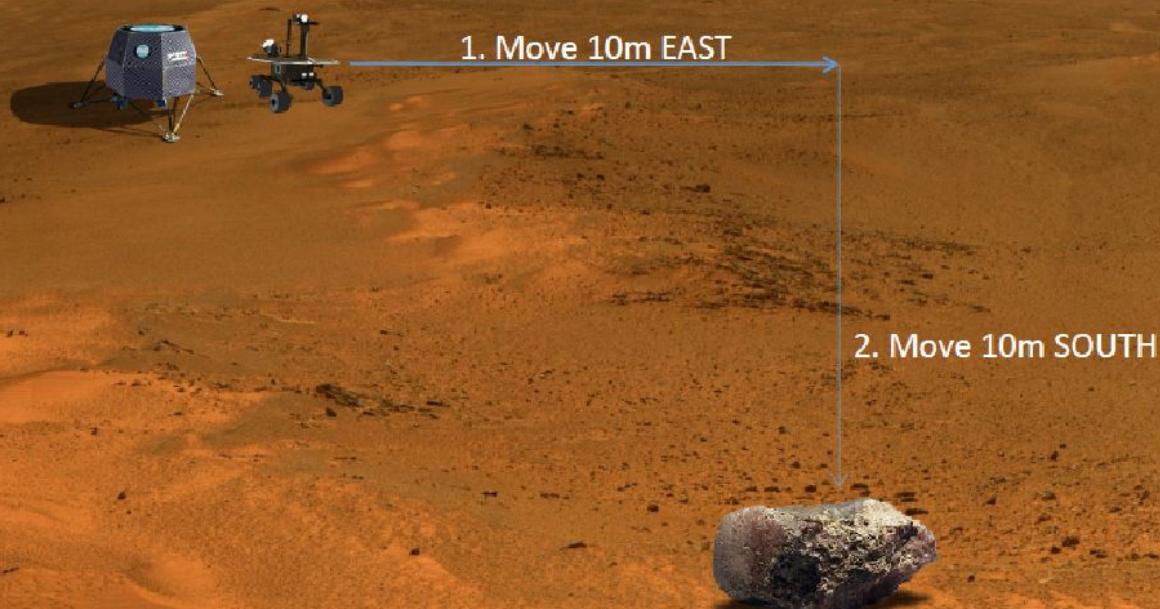
Example : piloting barren Martian surface
with no external guidance cues e.g. GPS



- !! NASA COMMANDS !!
1. Move 10m EAST
 2. Move 10m SOUTH
 3. Bring back minerals



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

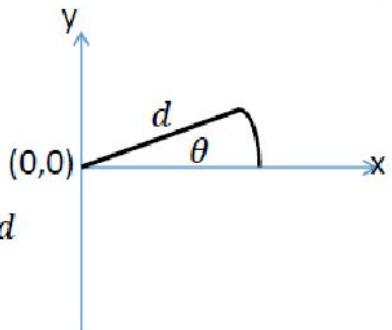


Example : formulation



1. Move 10m EAST

Robot position:



2. Move 10m SOUTH

d = distance travelled

θ = orientation

Example : formulation



1. Move 10m EAST

Robot position:

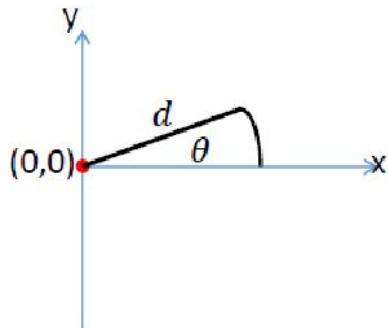
$$x(t) = x(t - 1) + \Delta x$$

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

where

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

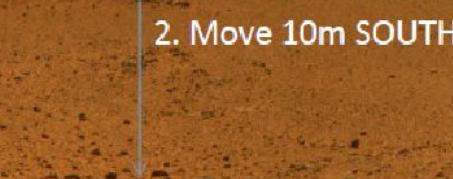
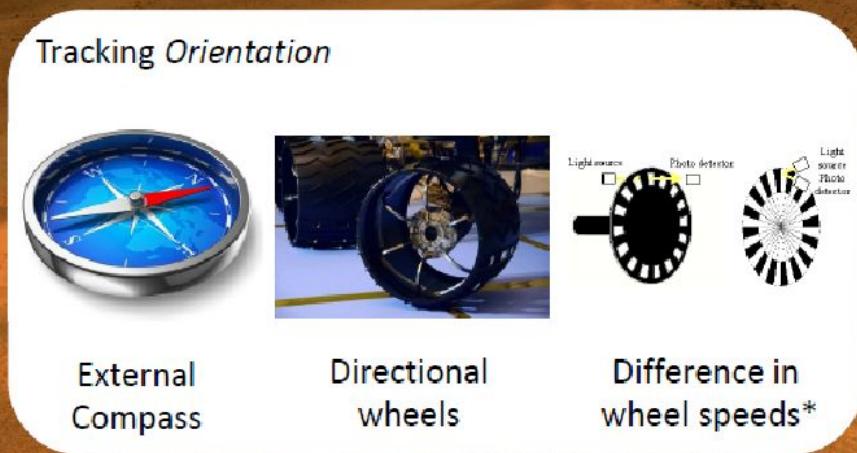
$$\Delta y = d * \sin(\theta)$$



2. Move 10m SOUTH



Example : monitoring orientation



Example : measuring distance travelled



1. Move 10m EAST

Measuring d from *wheel rotations*



2. Move 10m SOUTH



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

Example : measuring distance travelled

Measuring d from wheel rotations

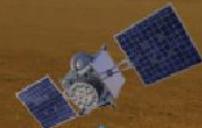

$$d = 1/2 * \pi * 50\text{cm}$$
$$d = 78.54\text{cm}$$


1. Move 10m EAST

2. Move 10m SOUTH



Getting home



!! EMERGENCY !!

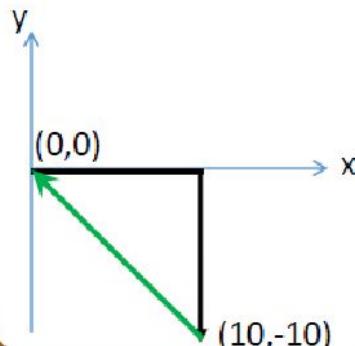
SAND-STORM IMMINENT

4. Seek shelter immediately

Getting home

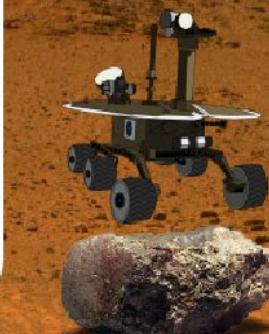


Calculating the home vector

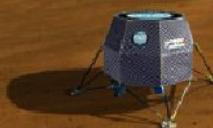


Cartesian format:

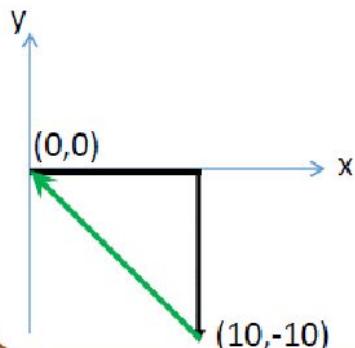
$$\begin{aligned} \text{HV} &= [x_1 - x_2, y_1 - y_2] \\ &= [0 - 10, 0 - (-10)] \\ &= [-10, 10] \end{aligned}$$



Getting home



Calculating the home vector



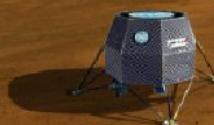
Polar format:

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

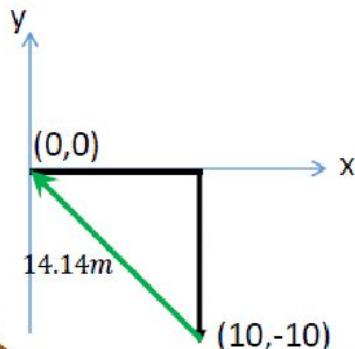
where r is distance home,
and θ is bearing from x



Getting home



Calculating the home vector



Polar format:

$$HV = [14.14, \theta]$$

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

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

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

$$r = 14.14m$$

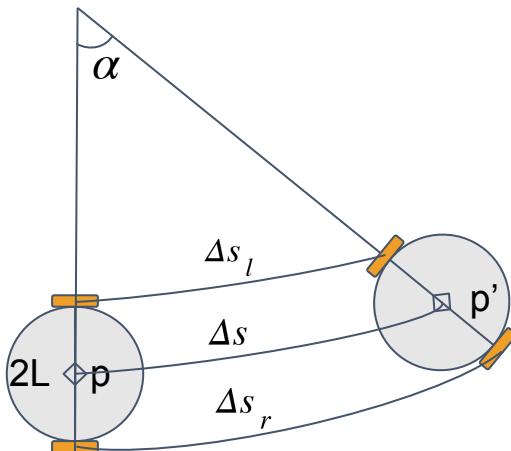


Odometry – differential drive robot

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

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

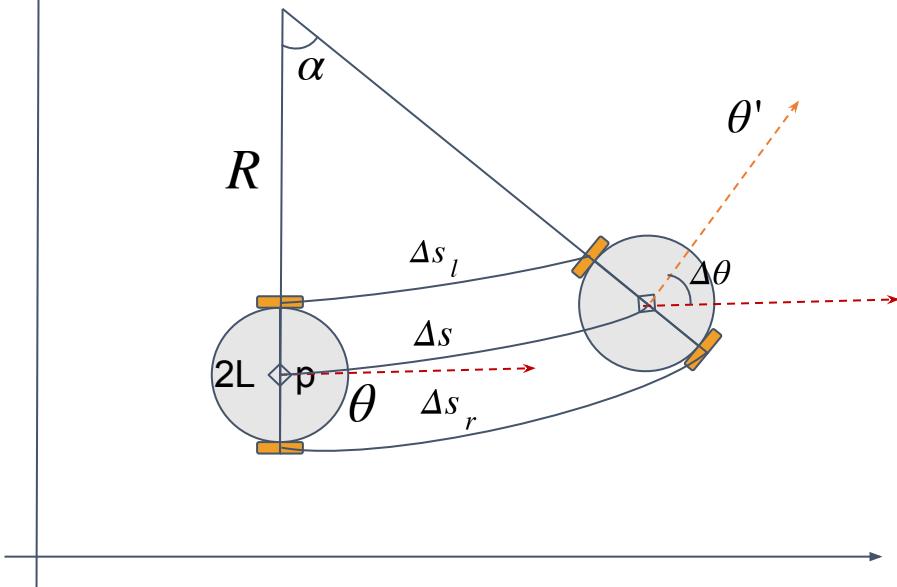
$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$$



How to calculate change in orientation?

Odometry – differential drive robot

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



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

$$\Delta\theta = \alpha$$

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

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

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

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

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

Substitute radius R in Δs_l

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

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

Odometry – differential drive robot

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

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

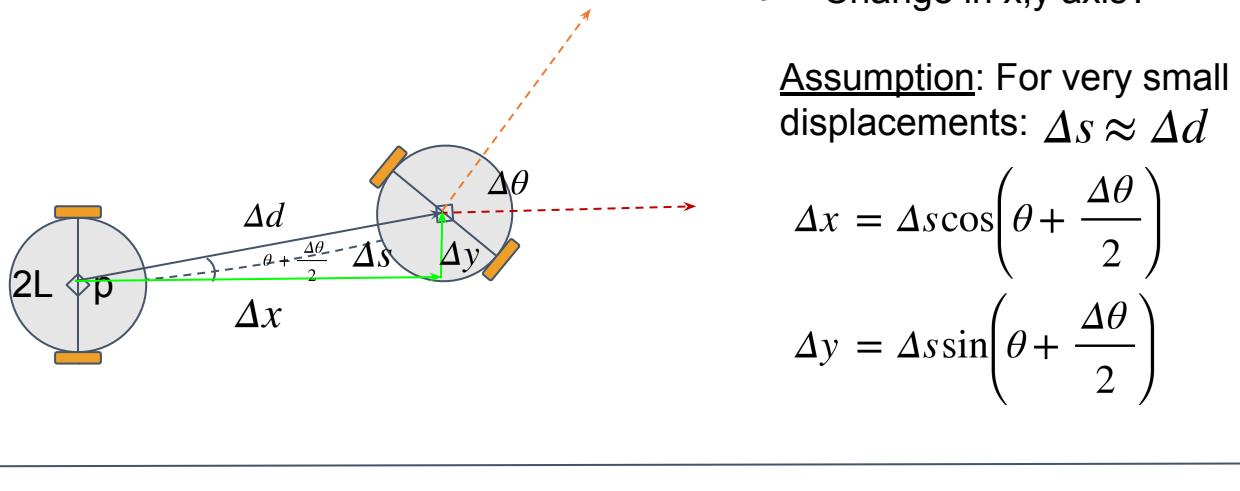
How to calculate:

- Change in x,y axis?

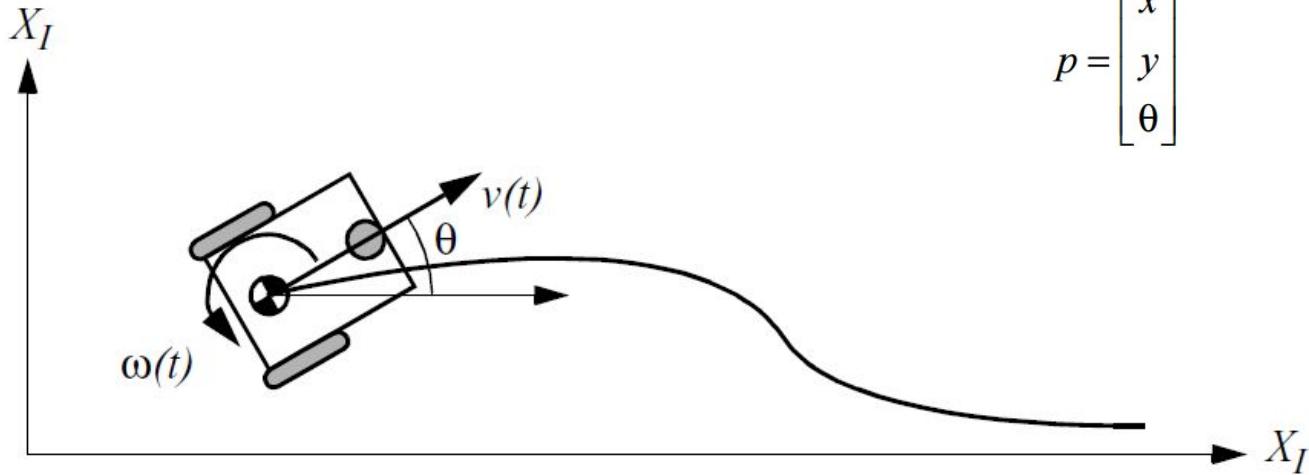
Assumption: For very small displacements: $\Delta s \approx \Delta d$

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

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



Odometry – differential drive robot



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

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

Wheel odometry

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

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

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

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

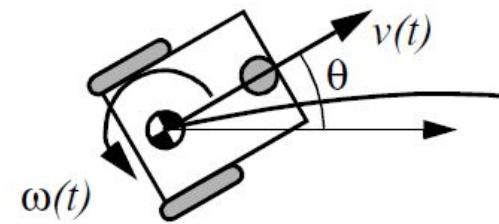
where

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

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

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

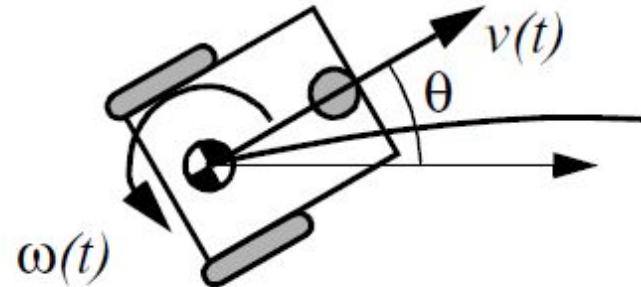
These terms comes from the application of the Instantaneous Centre of Rotation



Mobile robot odometry

Putting it all together....

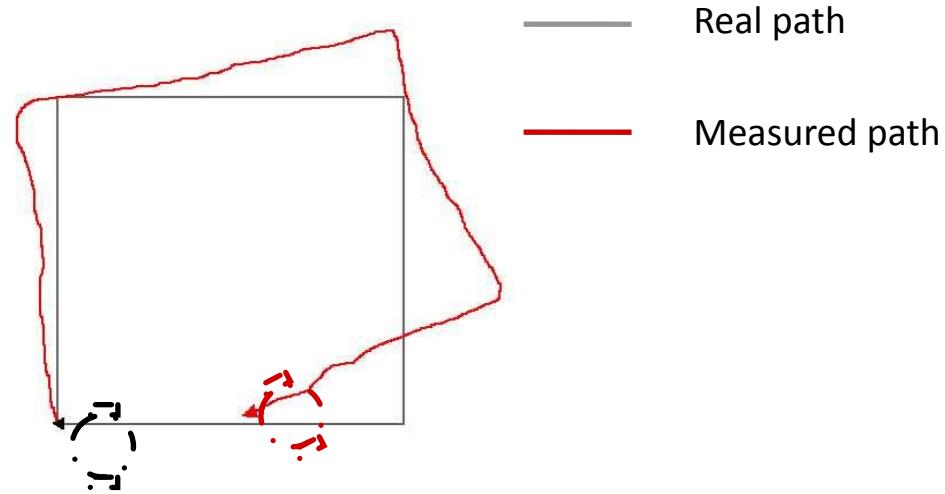
$$p' = p + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}$$



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

Problems with dead reckoning

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



Distance = 0 ft



Odometry error types

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

Odometry error sources

- **Systematic**

- Misalignment of the wheels
- Unequal wheel diameter

- **Non-systematic**

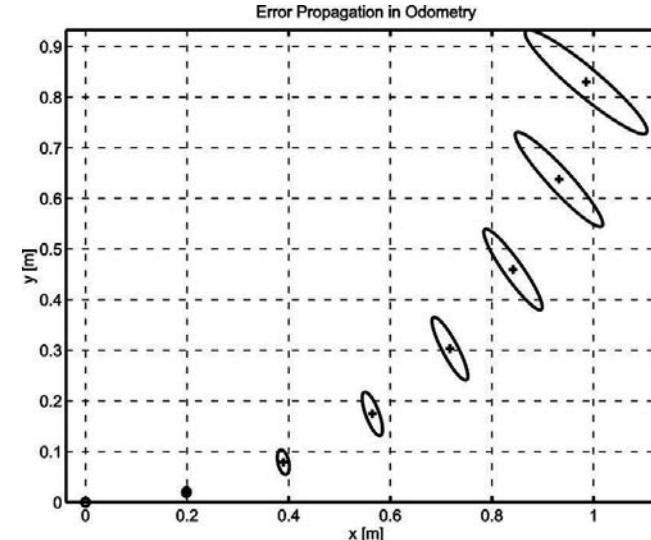
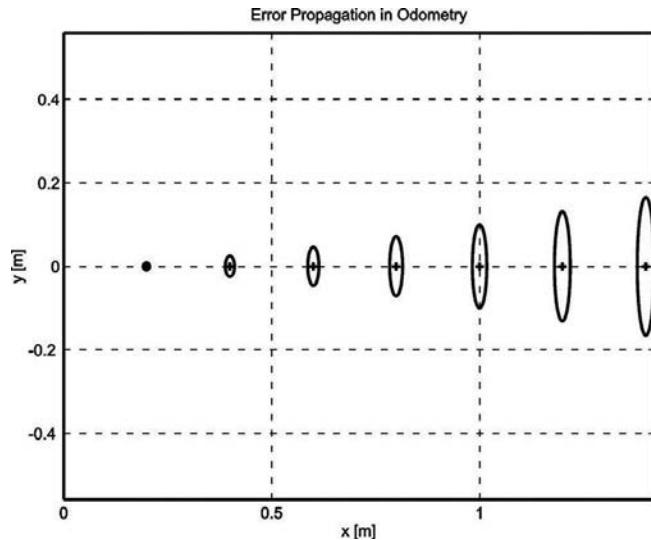
- Variation in the contact point
- Unequal floor contact of the wheel (slippage)

- Limited resolution during integration

- Time increments, measurement resolution

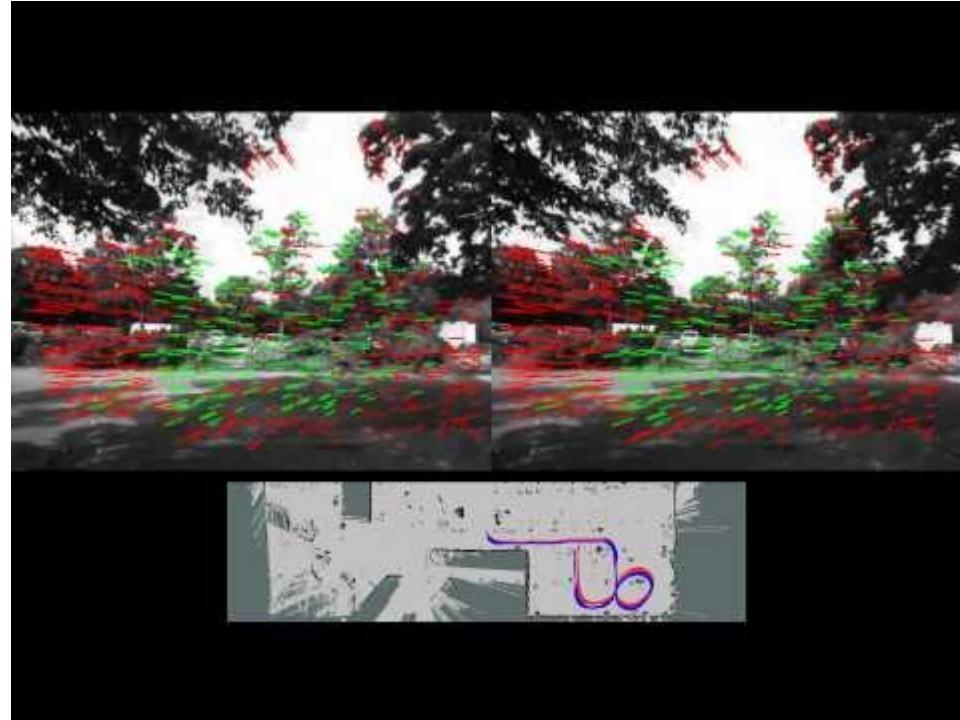


Error propagation in odometry



Visual odometry

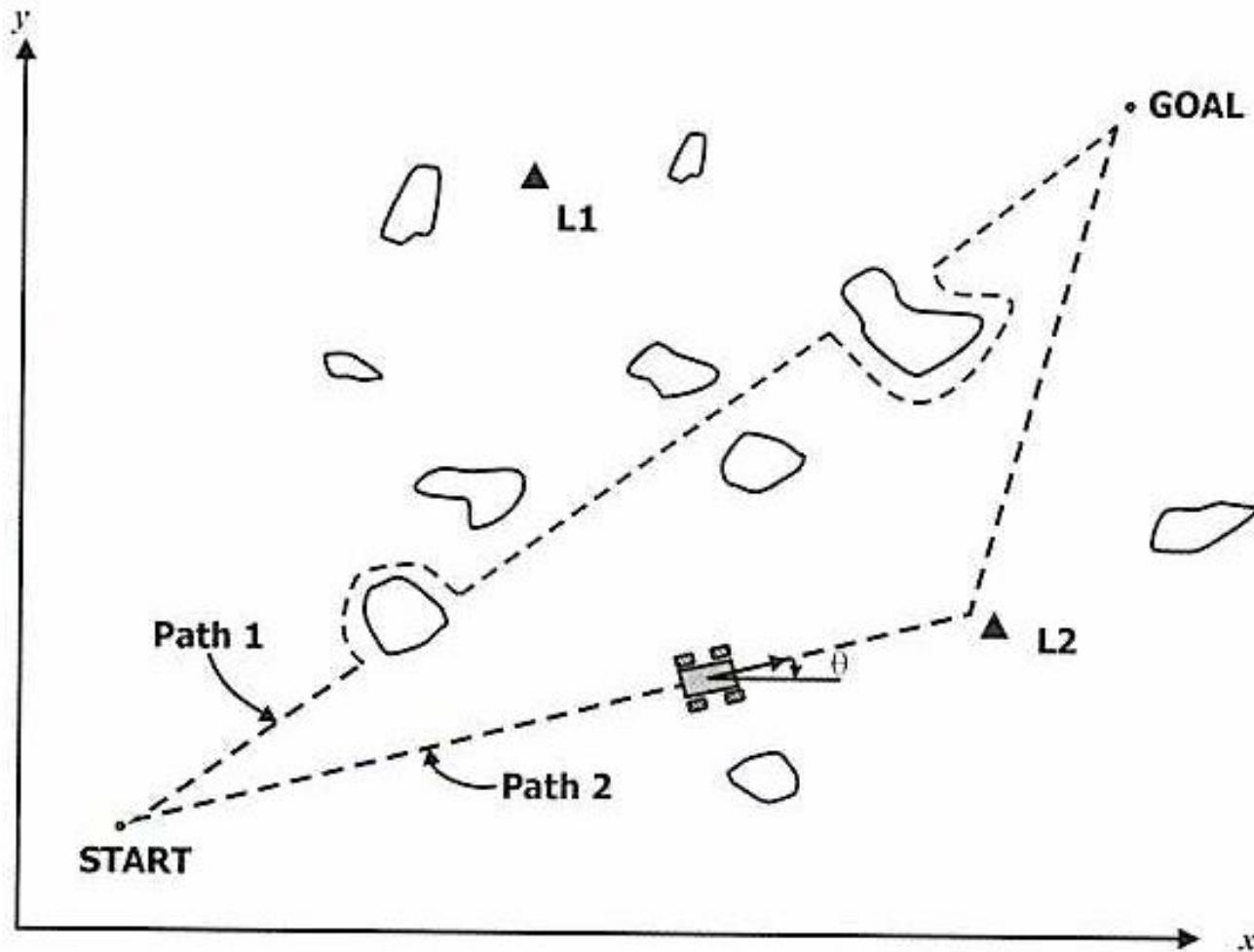
- Odometry is not limited to wheel encoders
- Consecutive camera images can be used to estimate velocity



Odometry summary

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

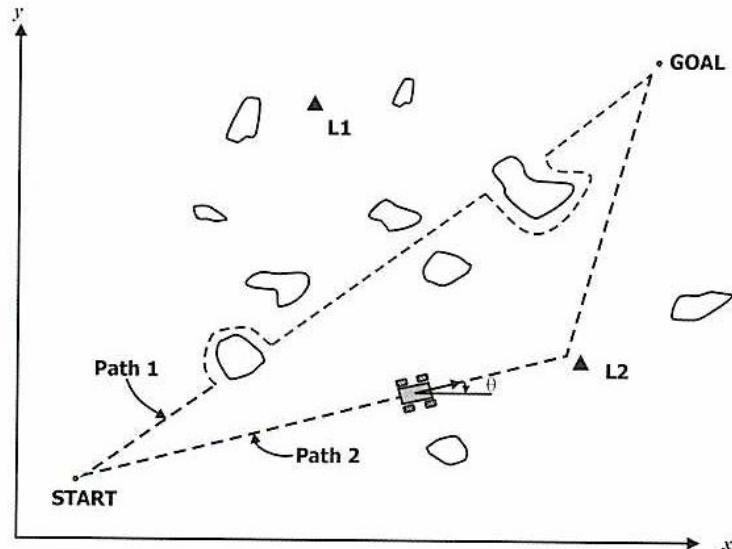
Navigation strategies



Navigation by vision and compass

Path 1

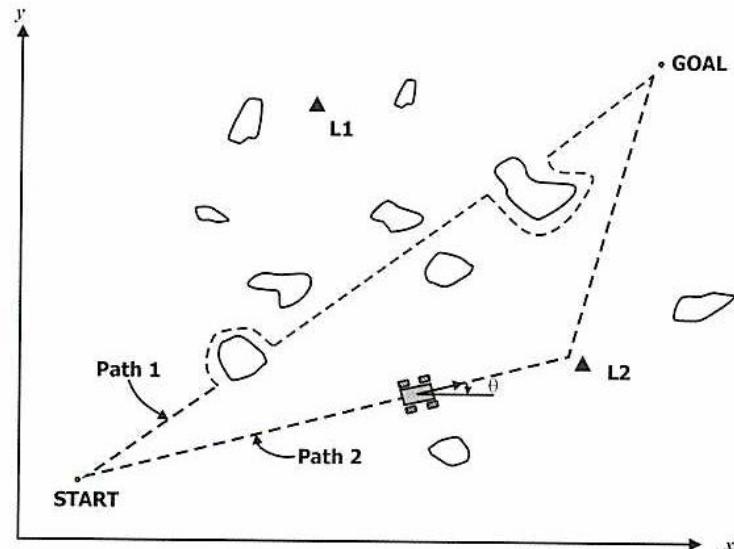
- Assuming visible goal, unknown distance
- Use vision to recognise goal
- Use compass to maintain heading towards goal
- When obstacle encountered (goal no longer visible):
 - Remember current compass reading
 - Travel around obstacle until the original compass direction is sensed again and goal is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



Navigation using landmarks as beacons

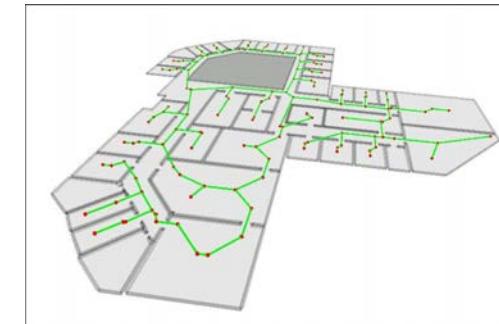
Path 2

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



Navigation by position tracking

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



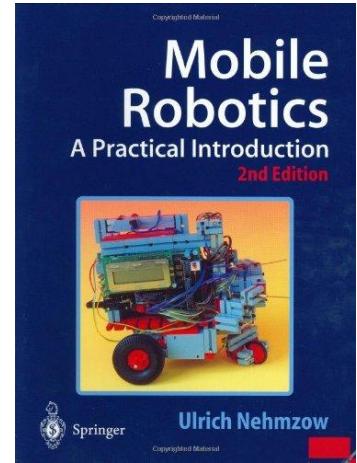
Navigation strategies

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

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

Recommended reading

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



CMP3103M

Autonomous Mobile Robotics

Lecture 7: Localisation

Dr Alan Millard

amillard@lincoln.ac.uk

Today's lecture

- Types of maps
- Map-based localisation
- Metric localisation
- Topological localisation

Mapping

What is a map?

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

Why maps?

Useful for a wide variety of robotic activities:

- Localisation, planning, mobile manipulation, human-robot interaction

Mapping is highly labour intensive

- Exploration (global coverage)
- Measurement (local coverage)
- Validity (correctness, error bounds)
- Currency (freshness)

Types of maps

- **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

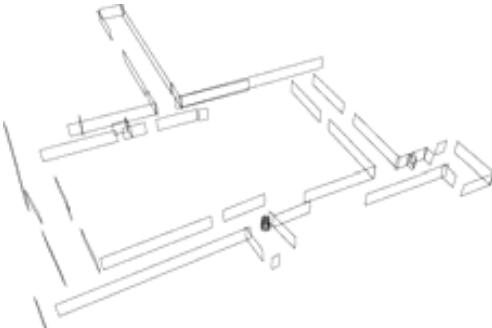
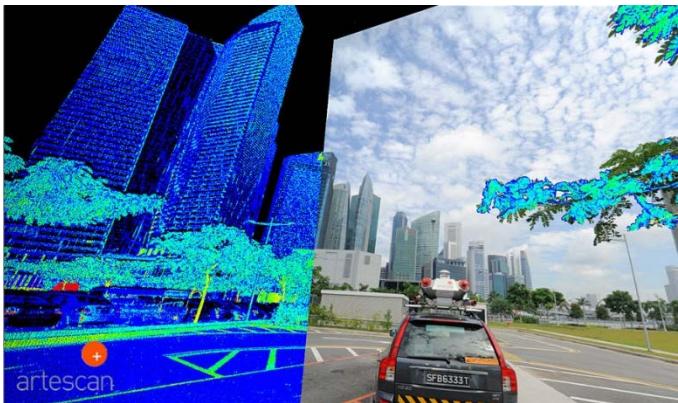


Record the location of objects in an absolute coordinate system

Metric maps

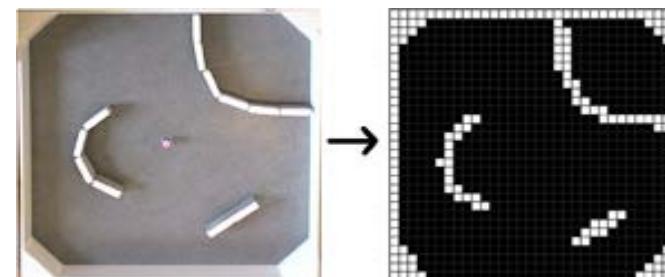
Continuous / “vector” format

- Points, linear/curved segments, surface patches



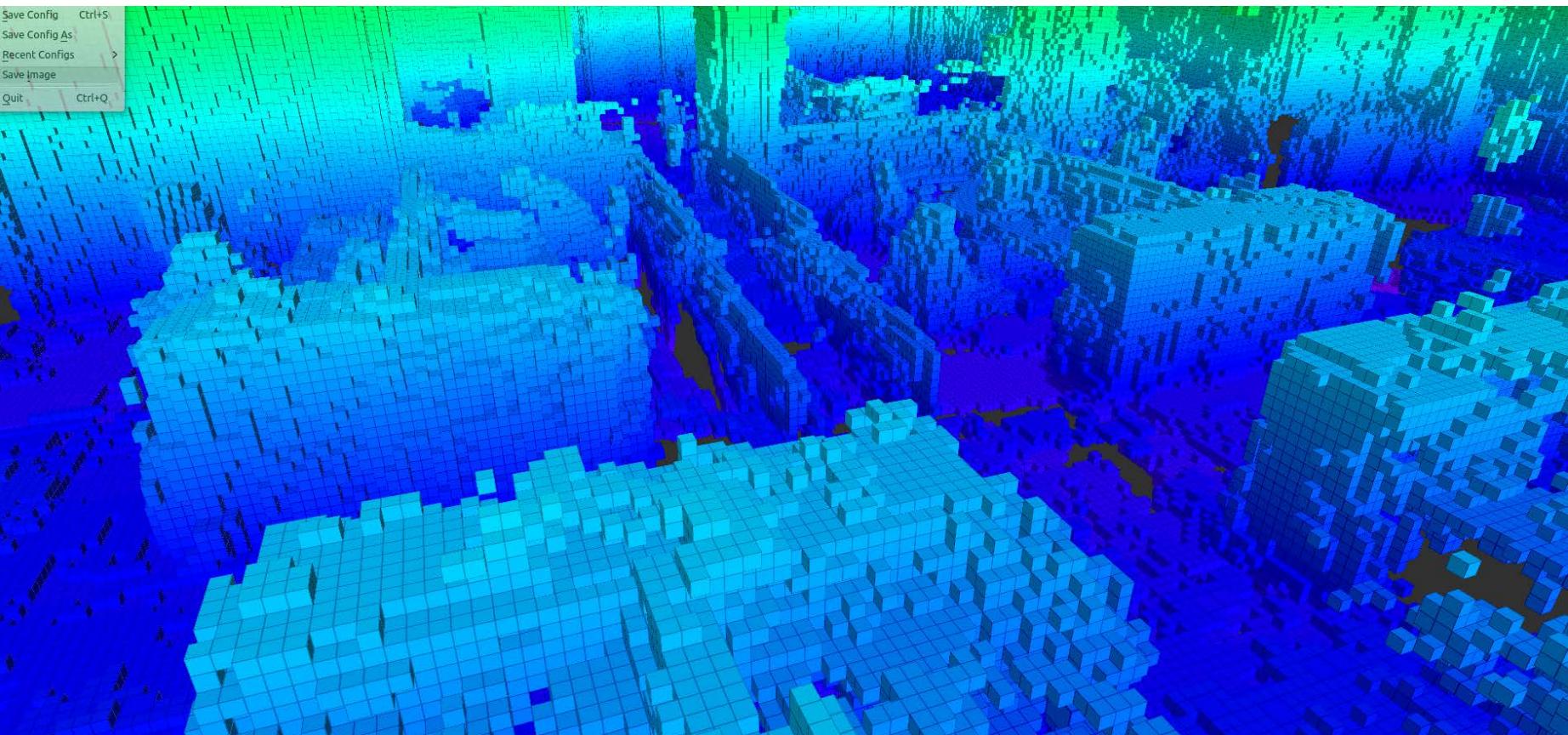
Discrete / “raster” format

- Occupancy grids

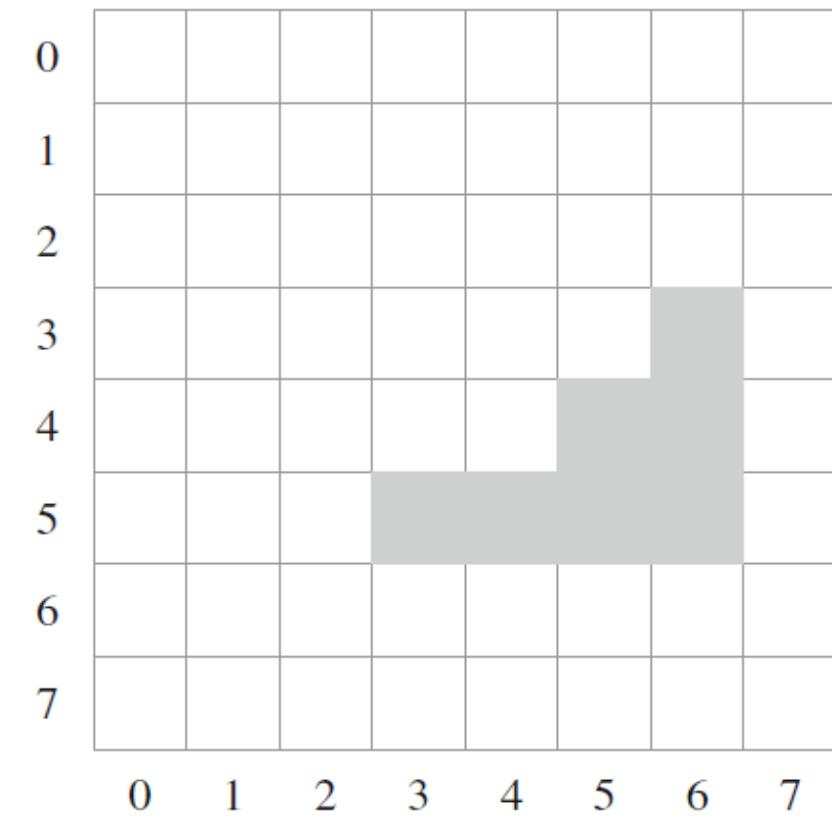
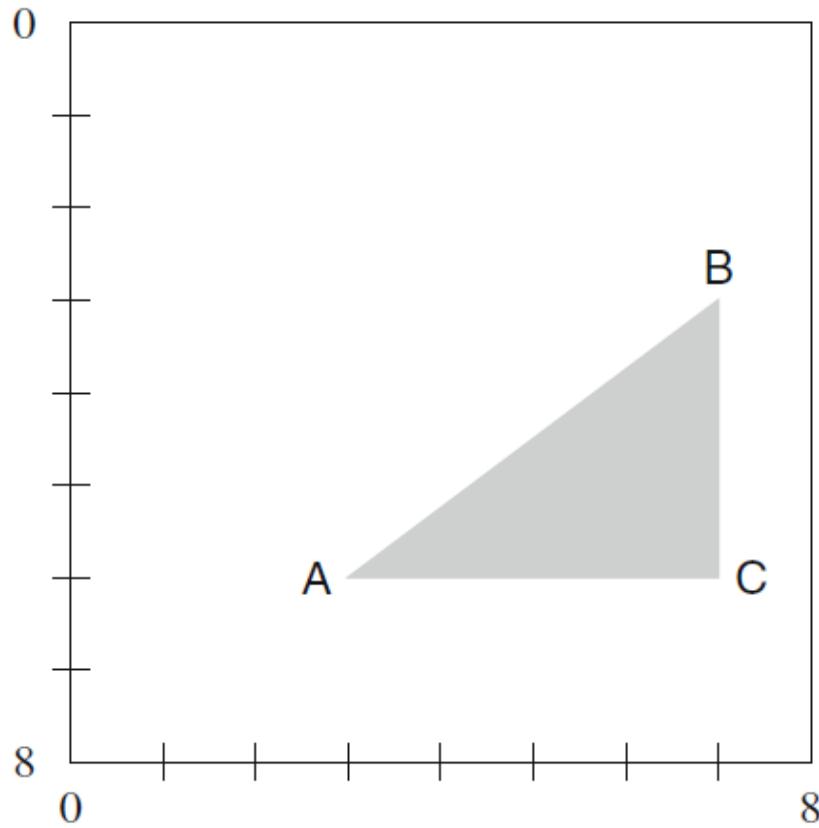


Linda's voxel map

http://wiki.ros.org/voxel_grid

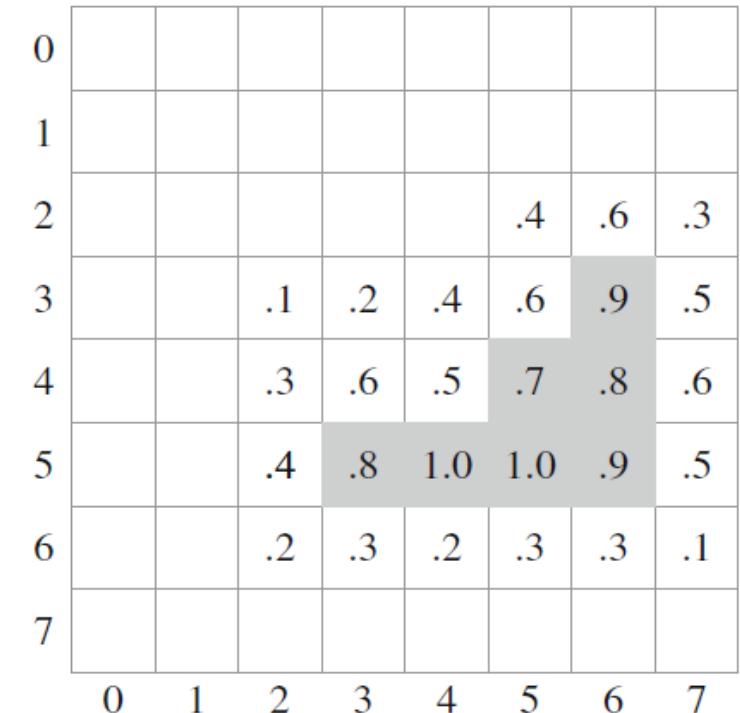


Continuous vs discrete maps



Metric maps – occupancy grids

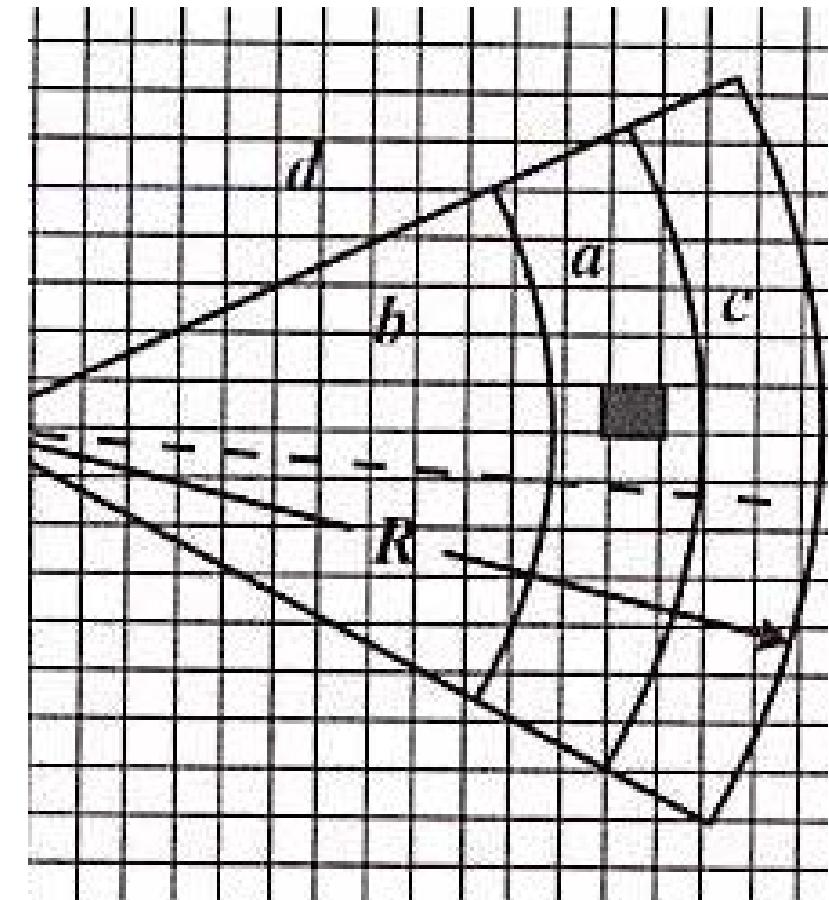
- Robot doesn't have complete and accurate prior knowledge about obstacles
- Each cell is associated with a probability that the cell is occupied, $P(\text{occ}_{x,y})$
- Updated using current robot pose (x, y, θ) and depth measurements from range-finder sensors, e.g. sonar, laser, stereo-vision



Model of a sonar beam

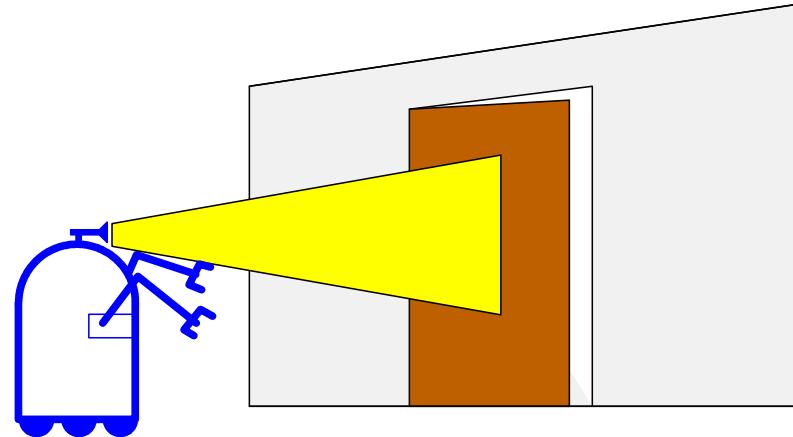
We need a model of the range sensor, e.g. for sonar we would define the following regions for a range measurement R :

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



Probabilistic robotics

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



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

Reasoning with probabilities

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

Bayes' rule

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

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

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

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

Bayes' rule

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

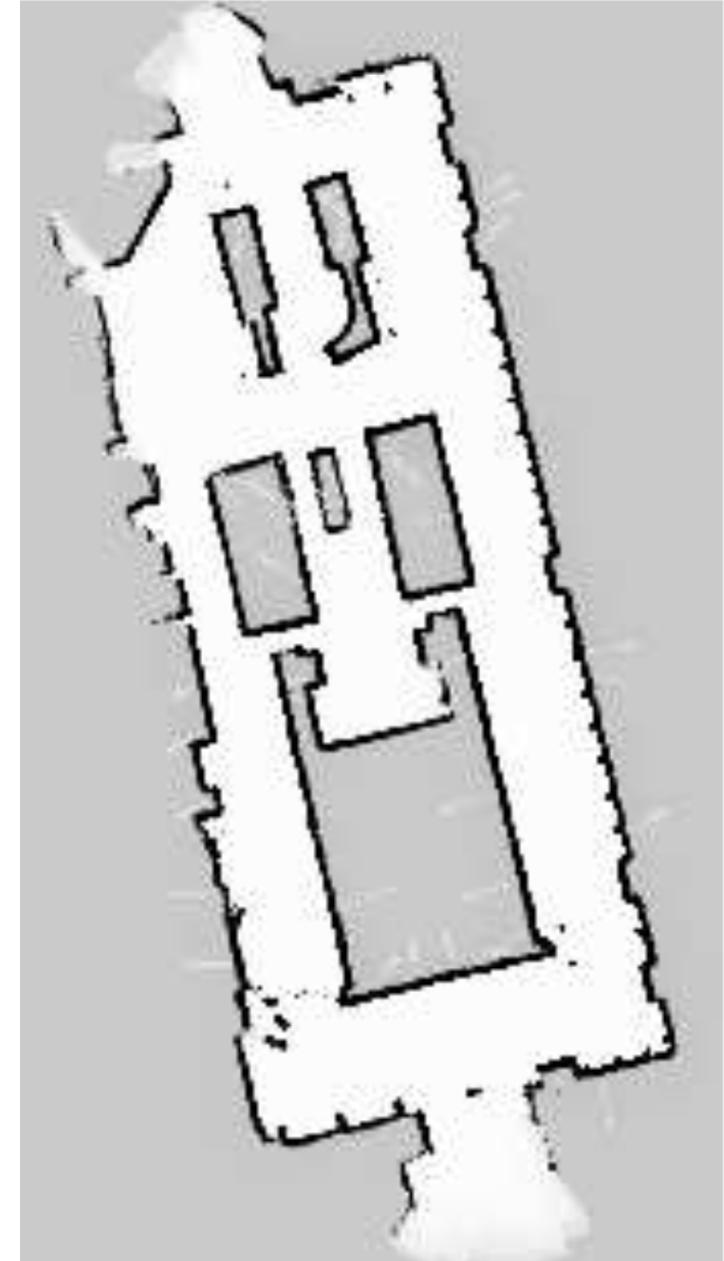
$$P(A | B) = \frac{1}{C} P(B | A)P(A)$$

Diagram illustrating the components of Bayes' rule:

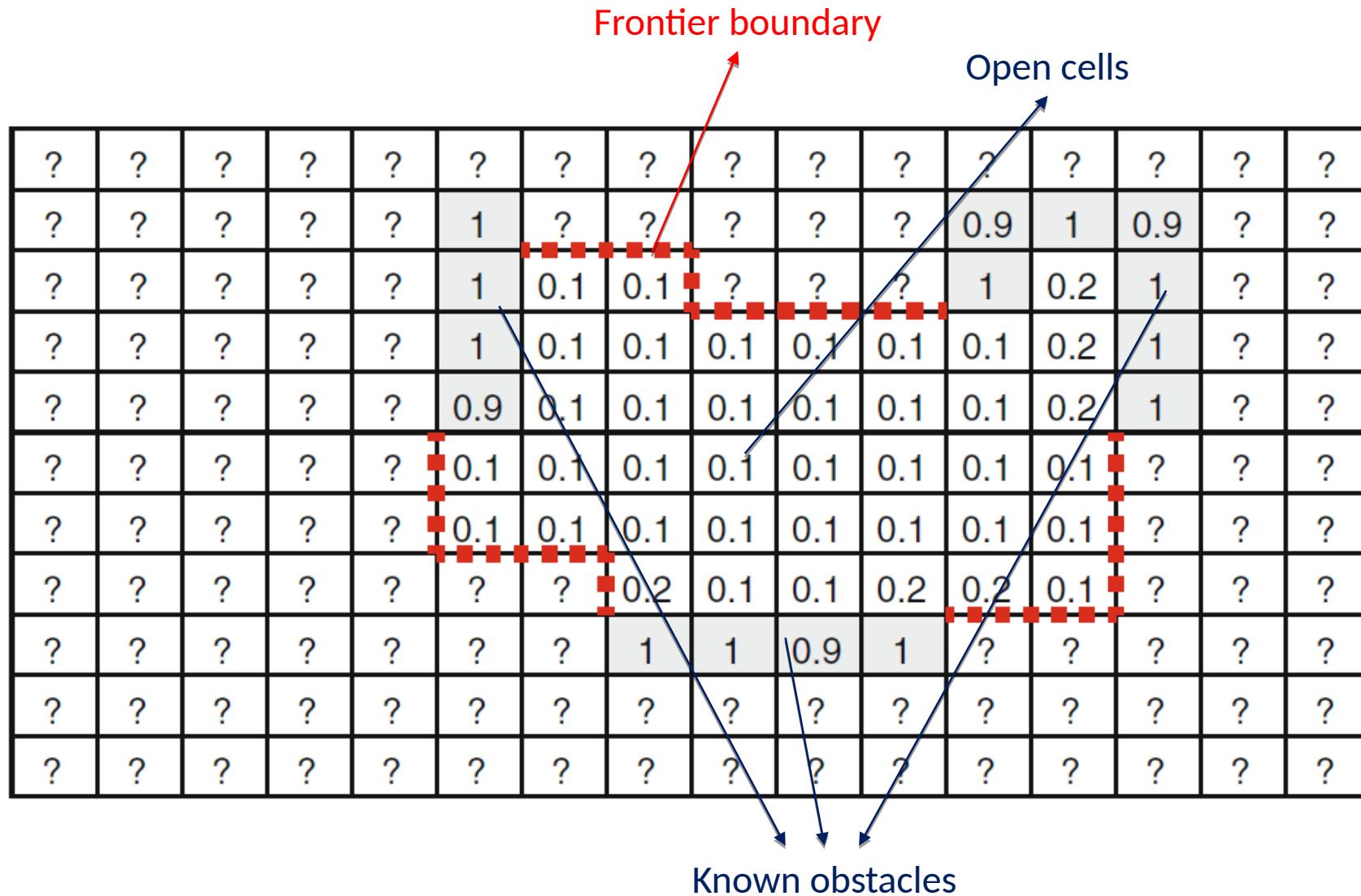
- posterior probability: points to $P(A | B)$
- normalisation factor: points to C
- likelihood: points to $P(B | A)$
- prior probability: points to $P(A)$

Example occupancy grid

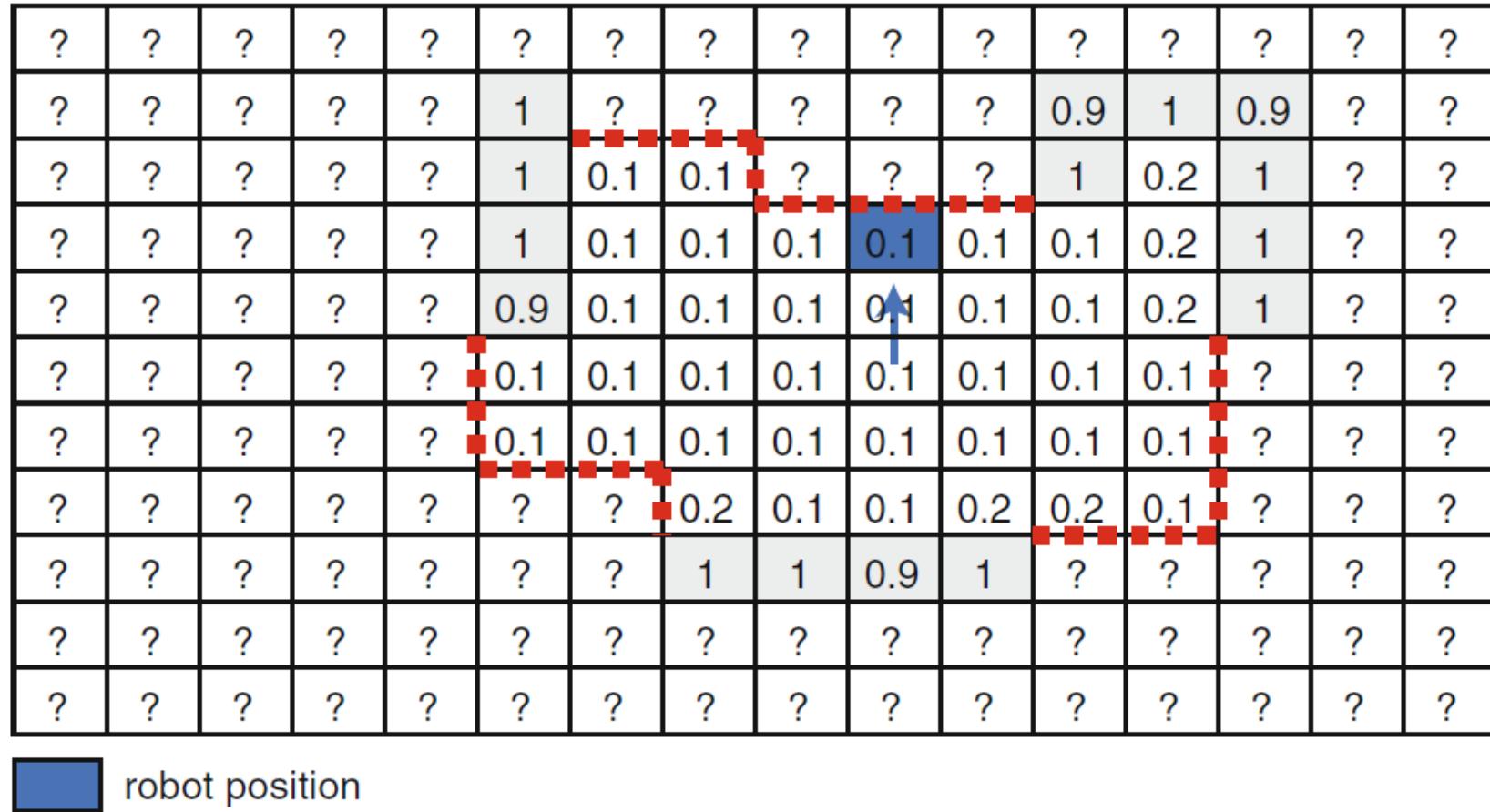
- 0 indicates that the cell has not been hit by any ranging measurements (free space)
- 1 indicates that the cell has been hit one or multiple times by ranging measurements (occupied space)
- Can change over time (e.g. dynamic obstacles)



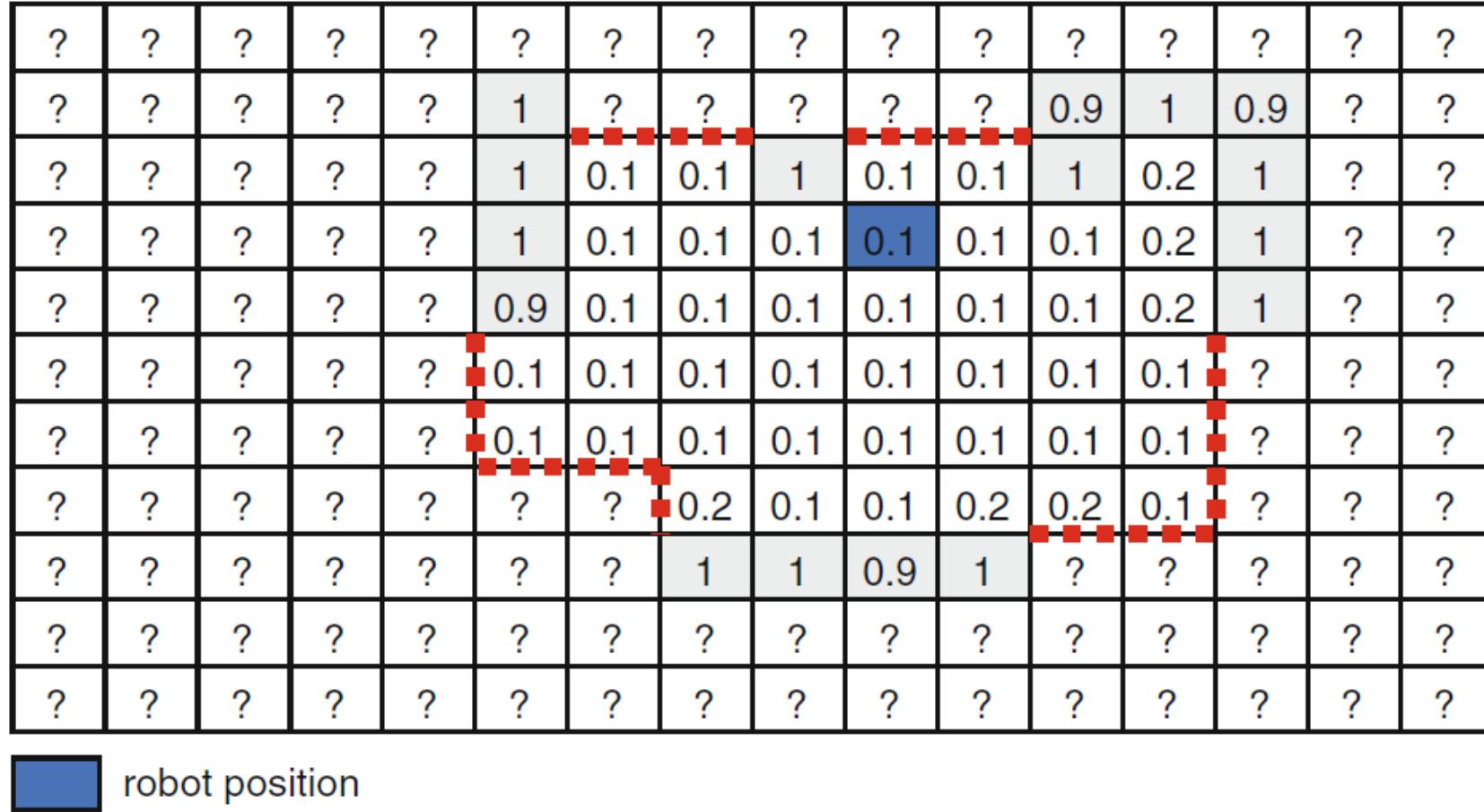
Frontier algorithm



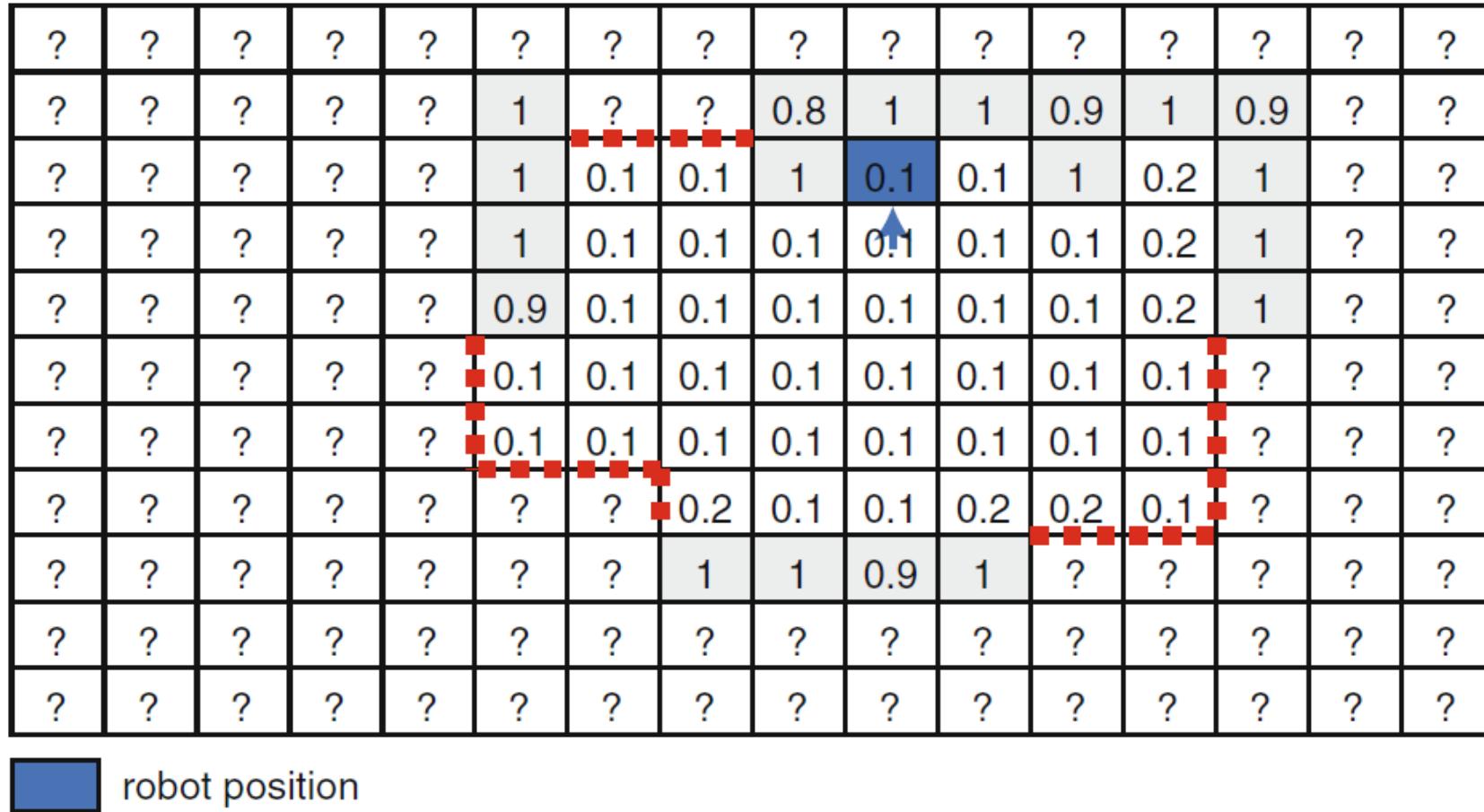
Frontier algorithm



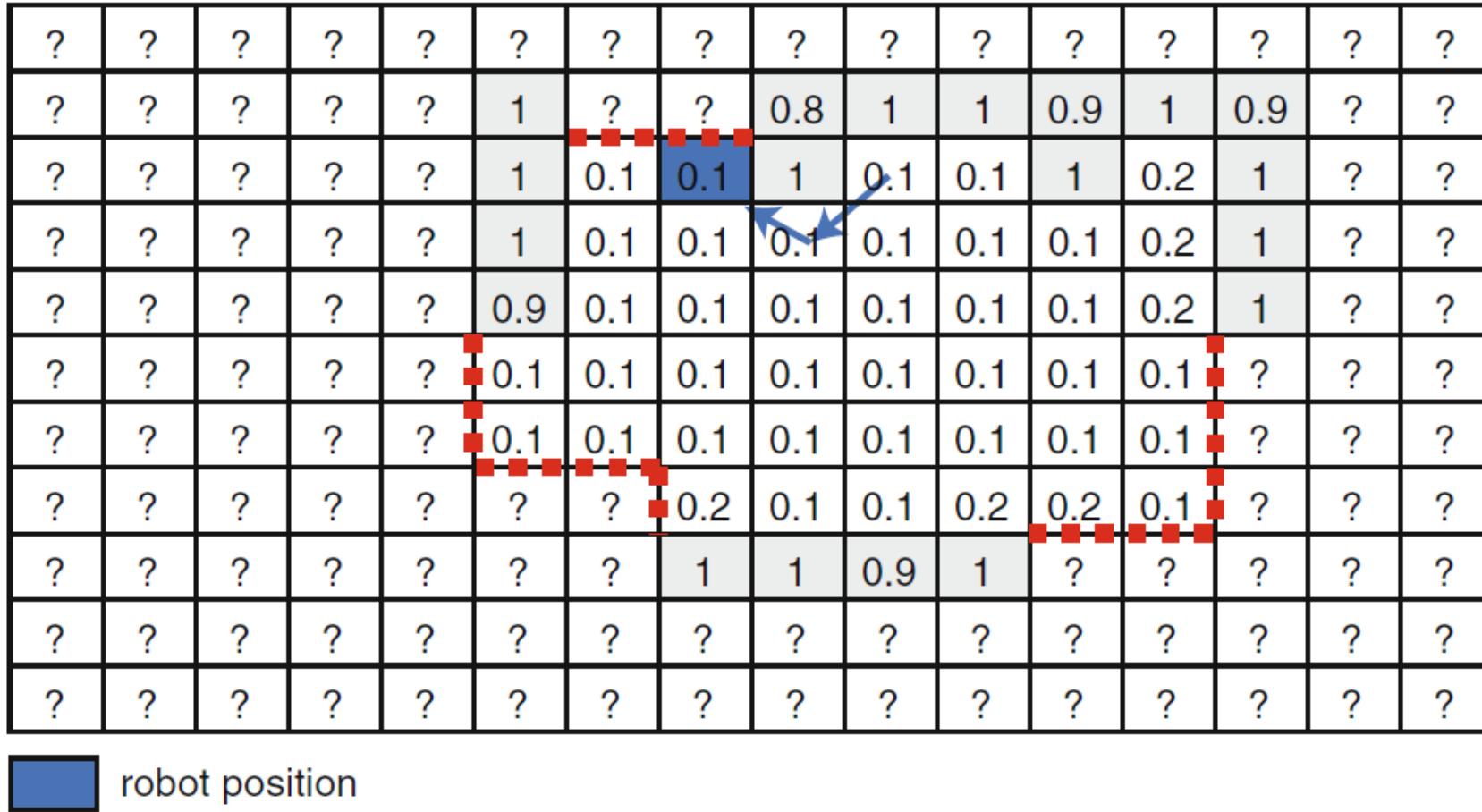
Frontier algorithm



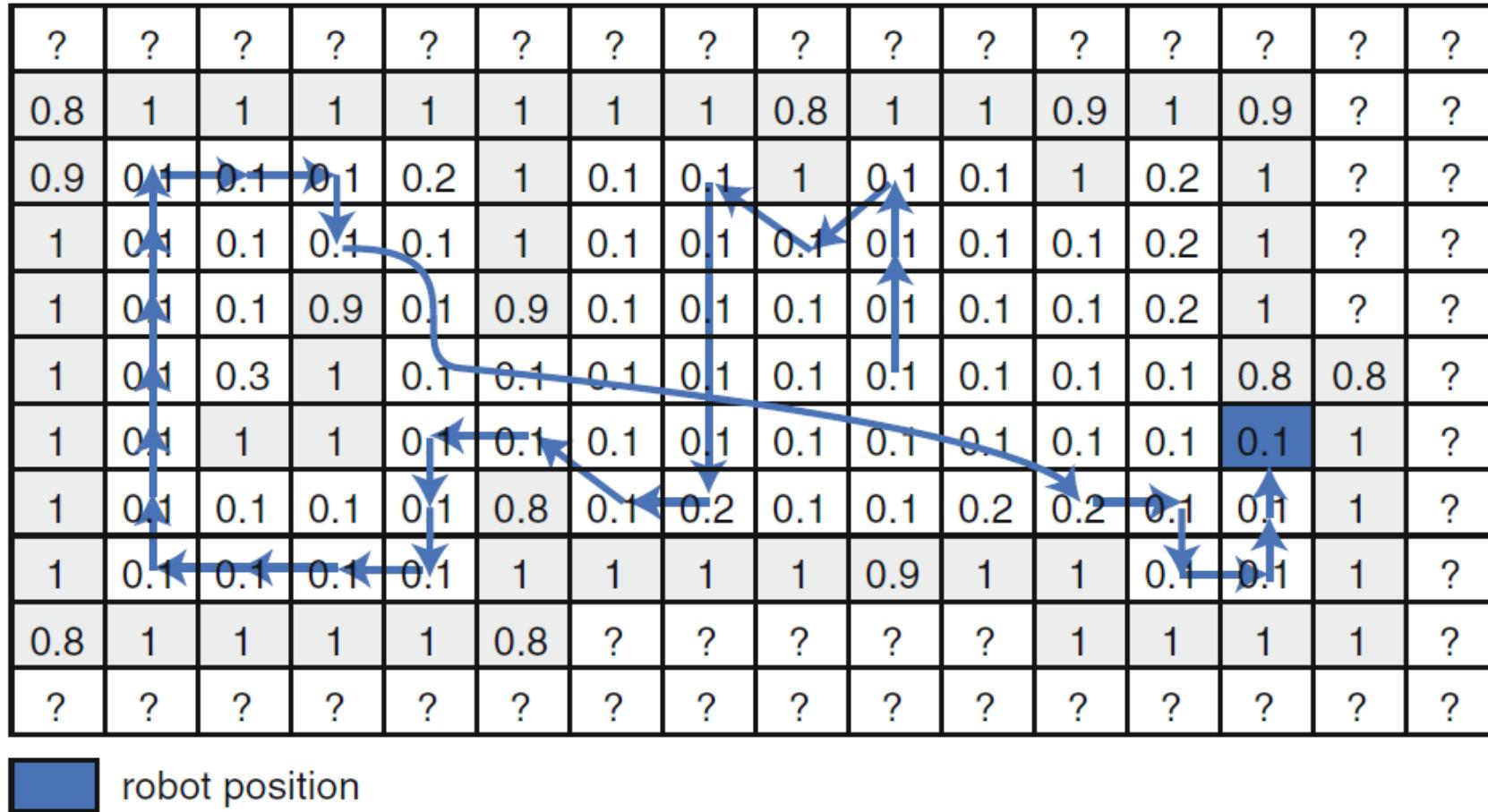
Frontier algorithm



Frontier algorithm



Frontier algorithm



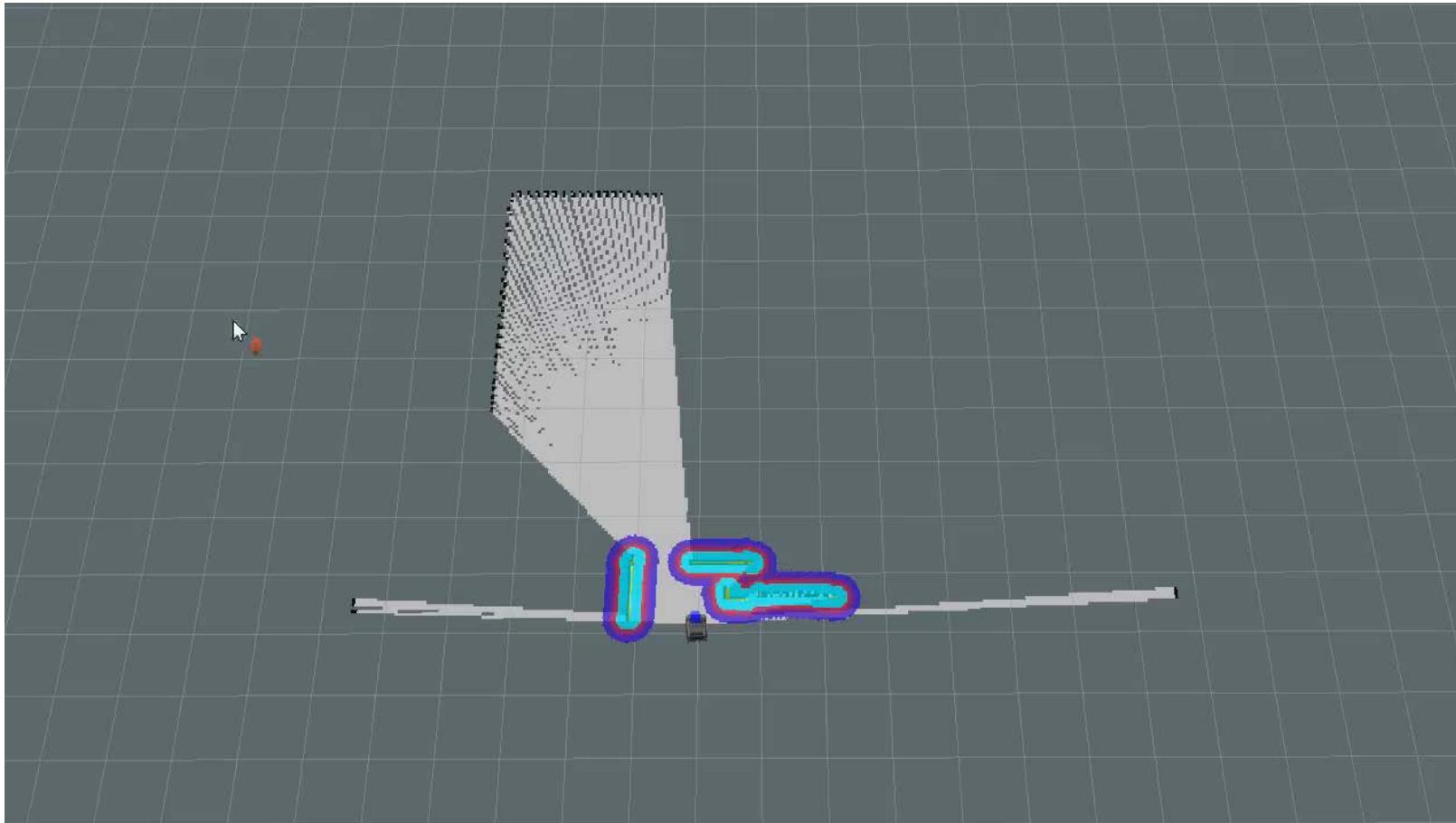
Priority of a frontier cell

Based on:

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

Frontier exploration

http://wiki.ros.org/frontier_exploration

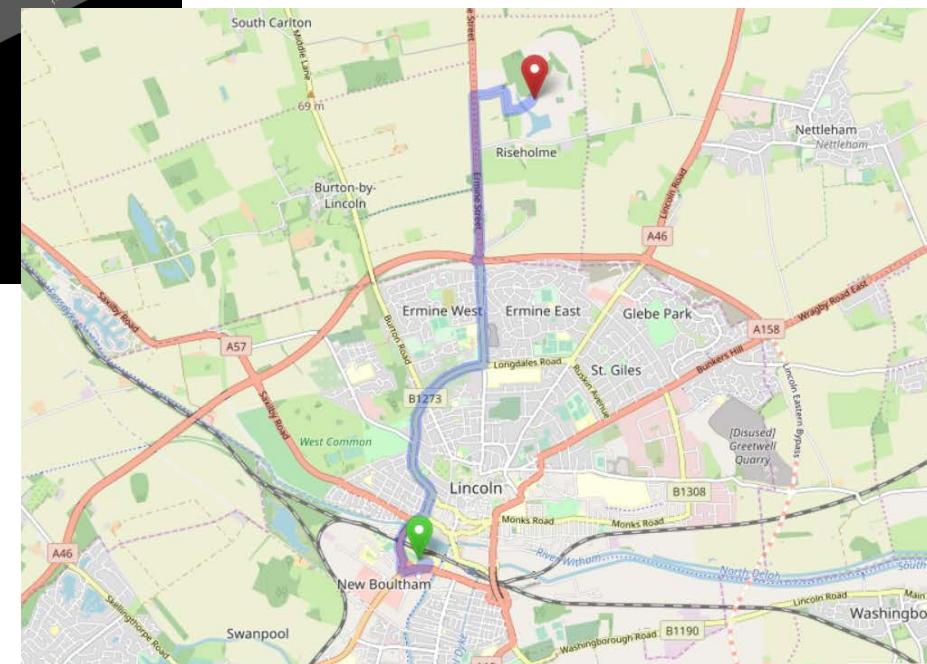
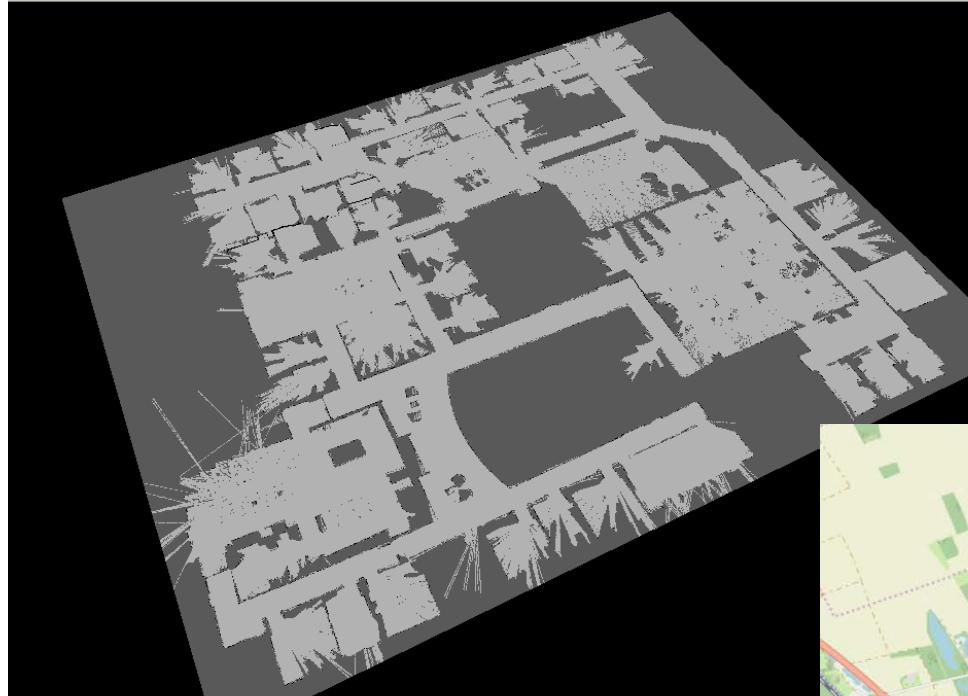


Topological maps



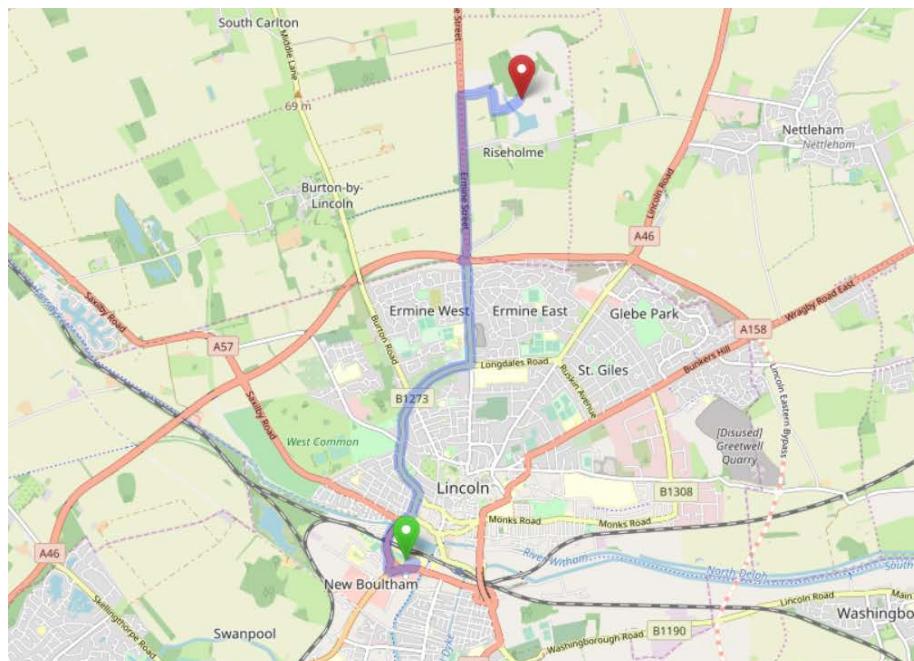
Represent known locations and connections between them as a set of nodes and edges in a graph

Complex, large, structured environments?



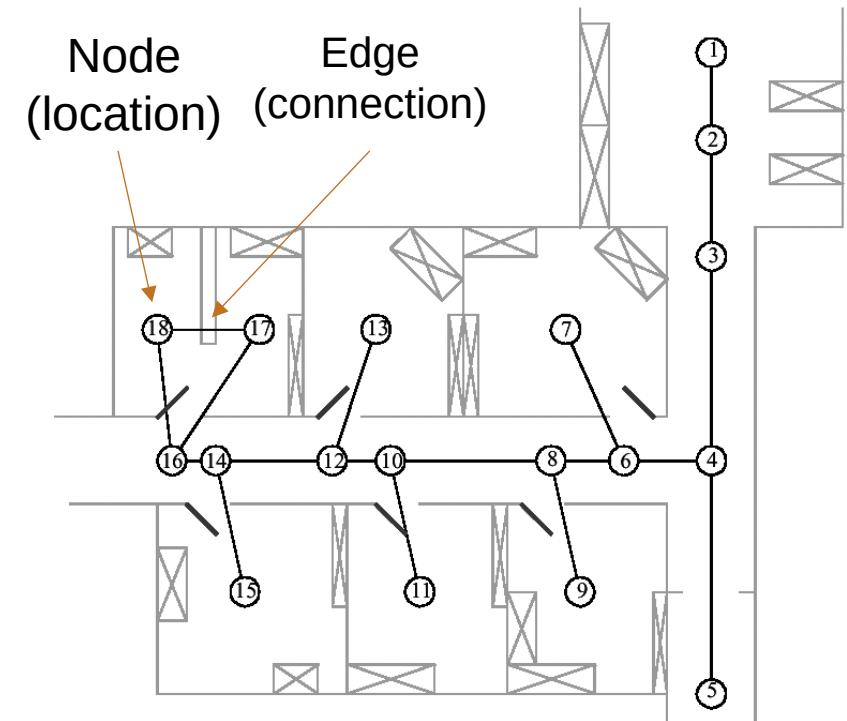
Complex, large, structured environments?

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



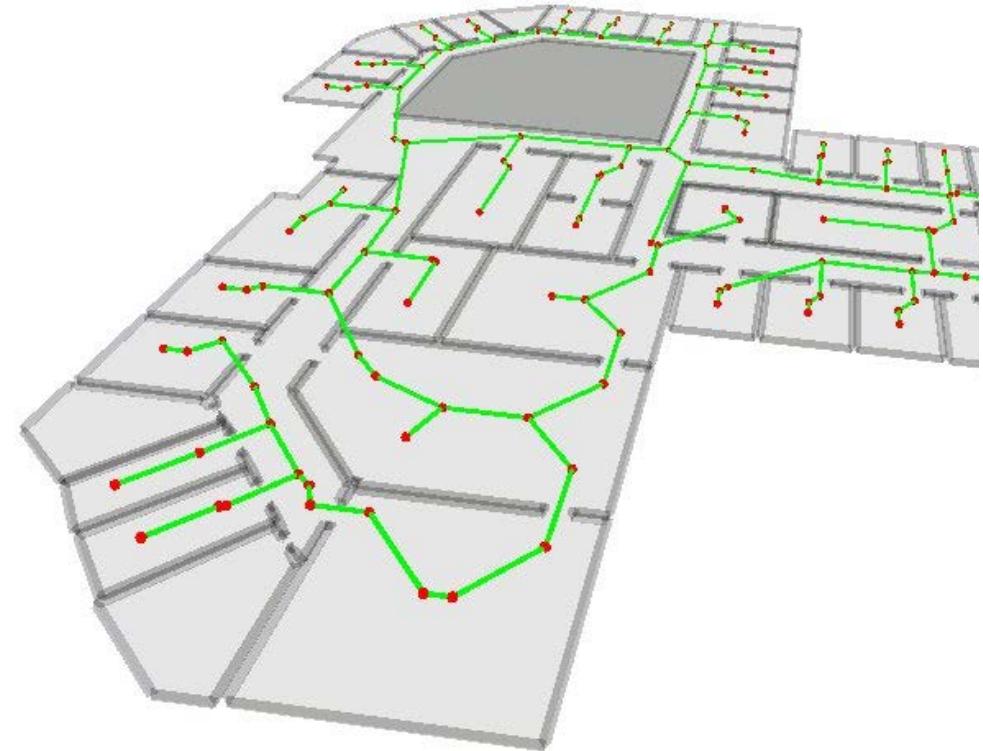
Topological maps

- 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



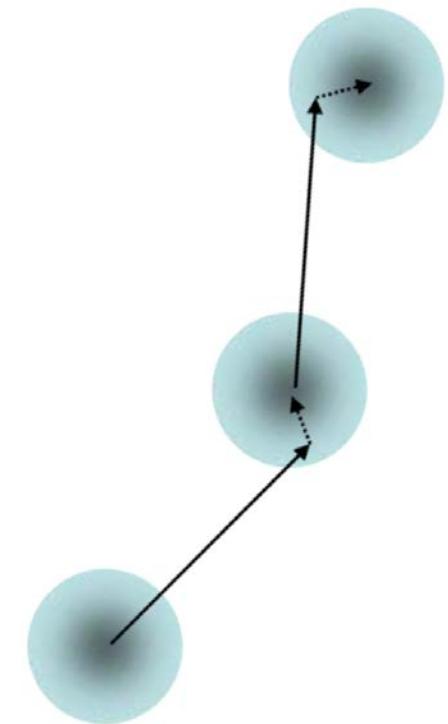
Topological maps

- Represent known locations and the connections between them as nodes and edges in a graph
- Edges could represent actions needed to get from one node to the next, or direction and distance
- Can determine a route via standard graph search methods (A*, Dijkstra) if distances (or costs) are added to the edges



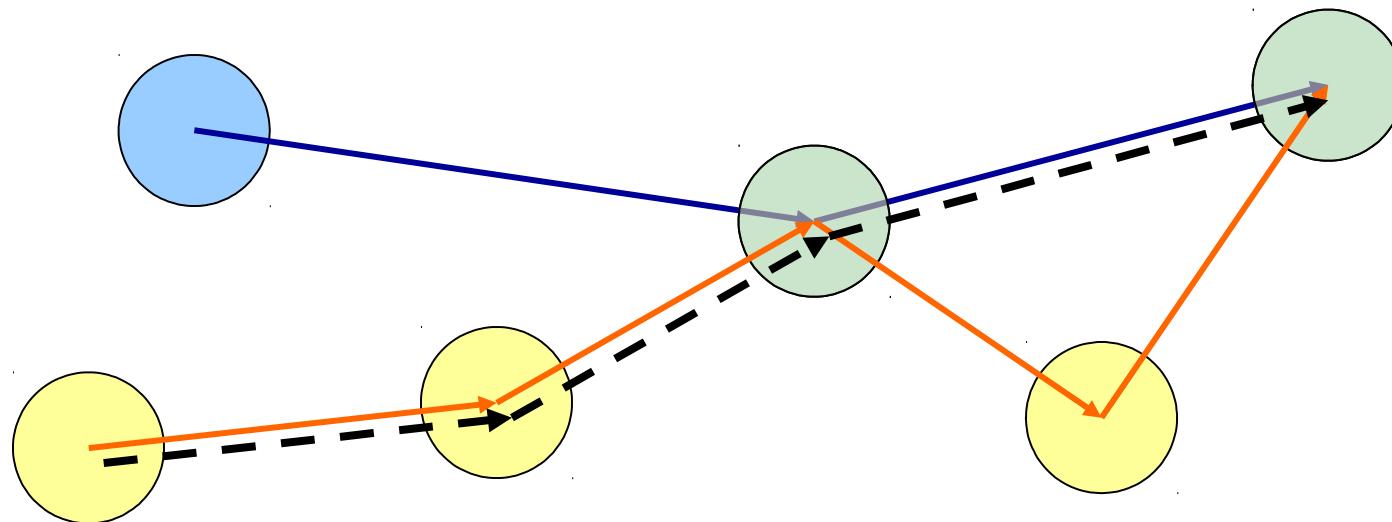
From local strategies to routes...

- Local strategies
 - Target location can be found from surrounding area using memory of 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



...to topological maps

- If we can combine routes by recognising overlapping locations from different routes, we have information we can use for finding novel routes
 - Often considered as a defining property of maps vs. routes



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

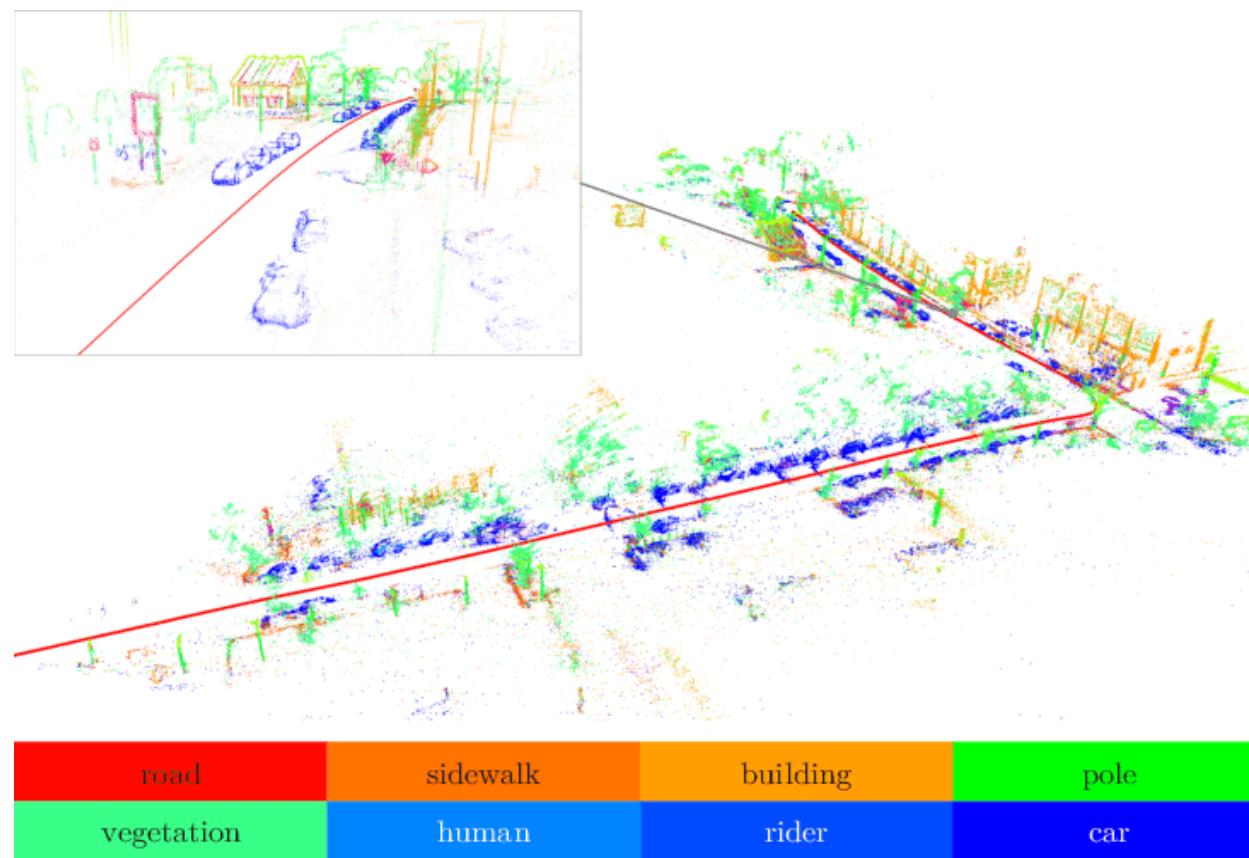
Metric maps

- 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

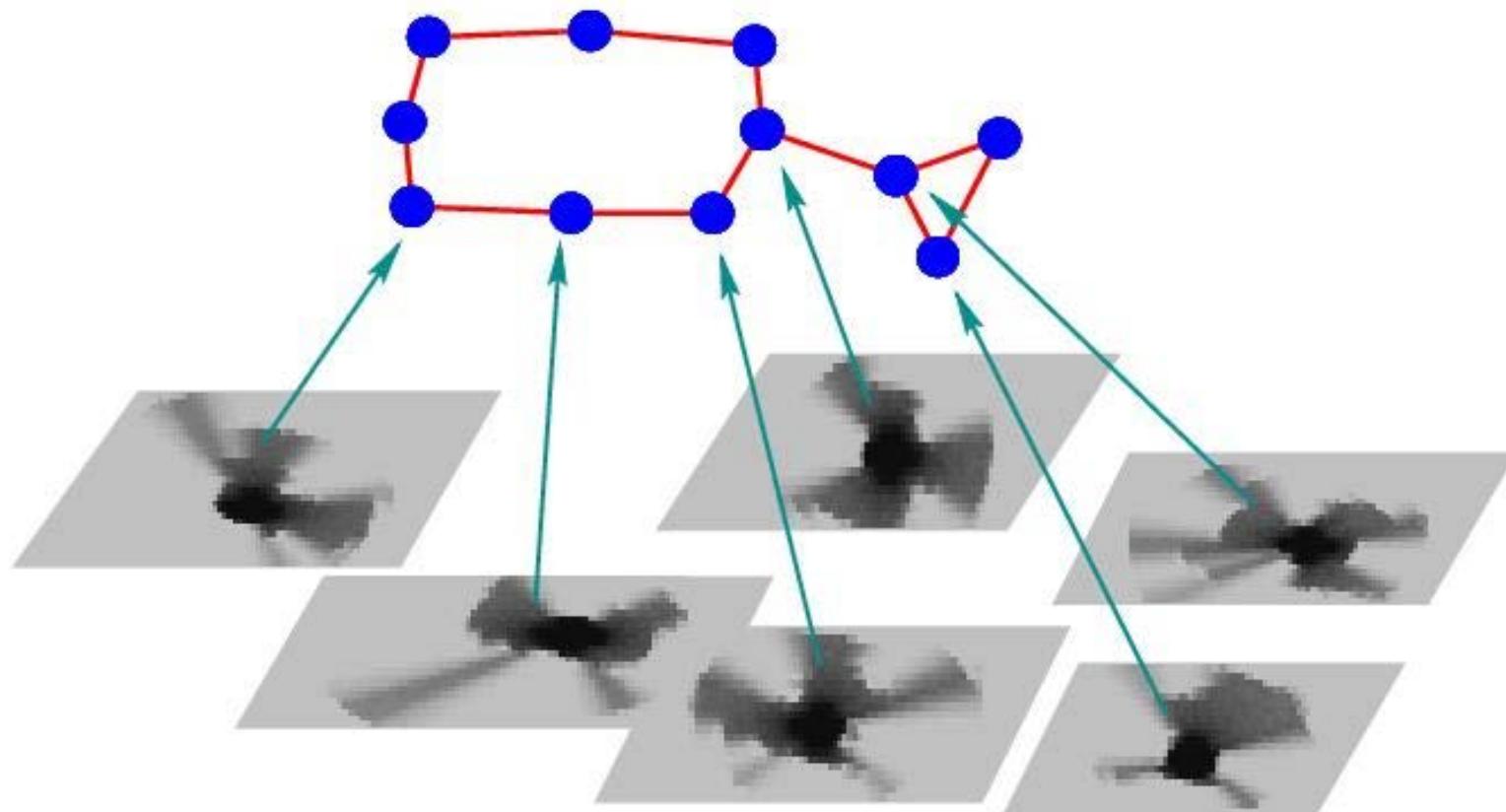
- Abstract, qualitative, “symbolic” representation
- May be more persistent/robust to environment dynamics
- Low computational cost - efficient path planning, scale better to large environments
- Require accurate place recognition - problem of perceptual aliasing (what if 2 or more places look alike?)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

Semantic maps



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

Hybrid maps

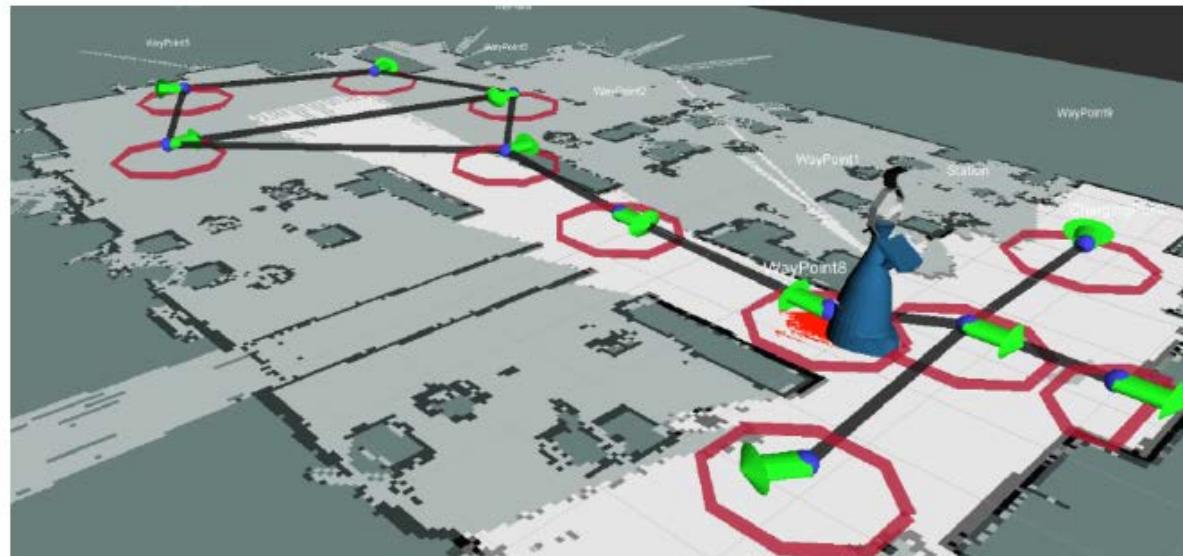


Combine complementary strengths of different representations (metric, topological, semantic maps)

Hybrid maps

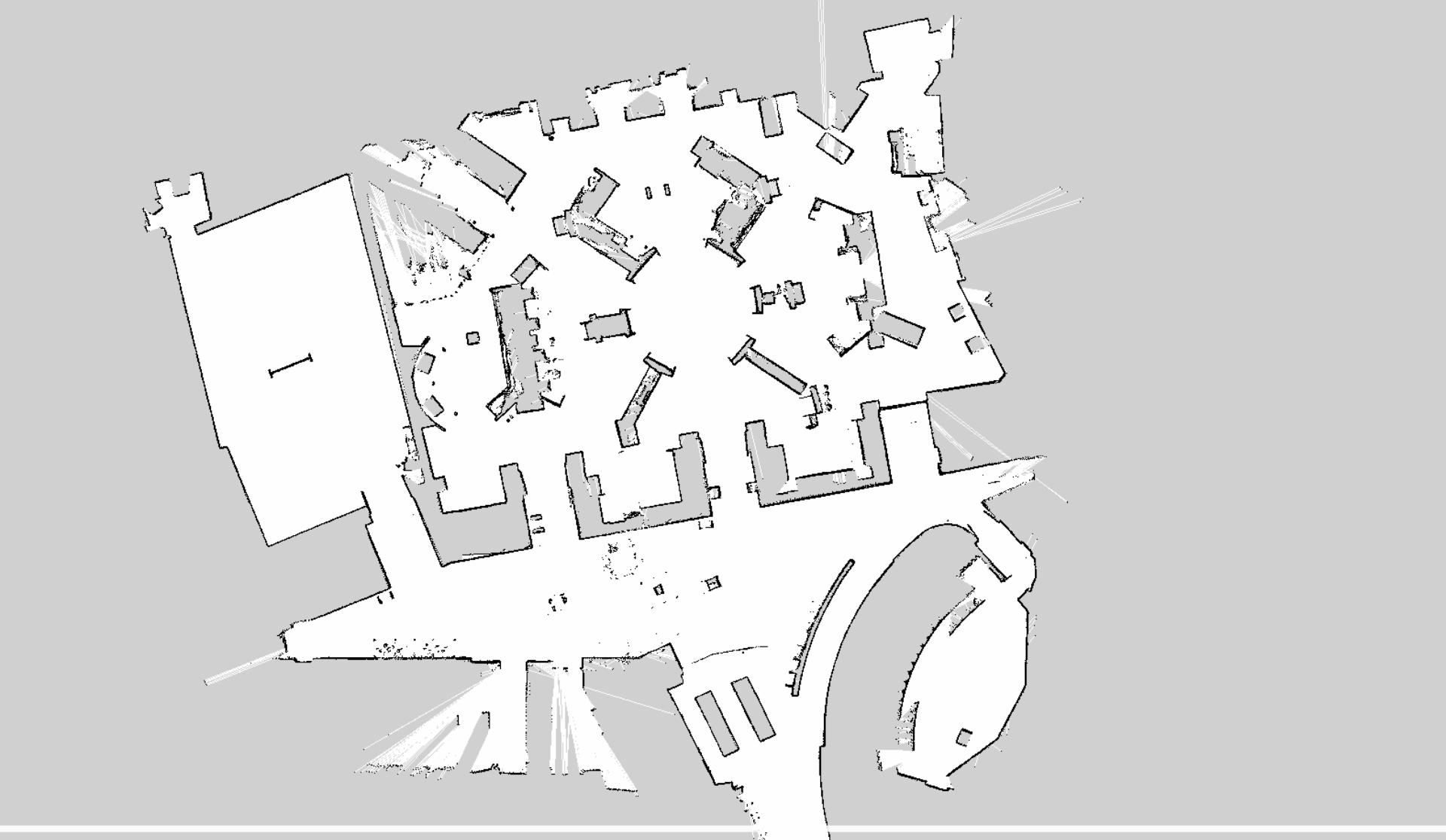
Example of a hybrid metric-topological map

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

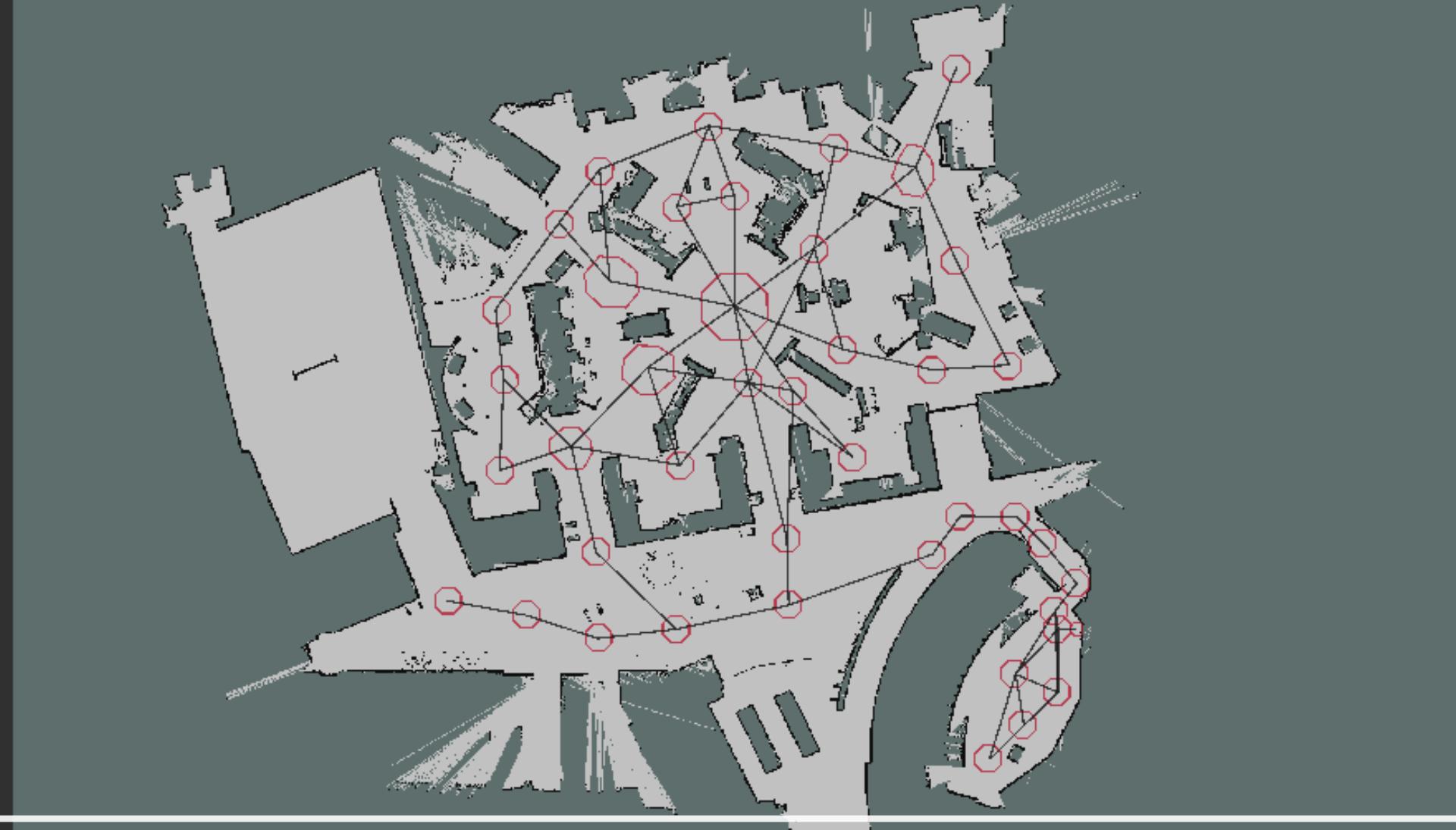


Lindsey at The Collection museum





Lindsey's obstacle map at The Collection

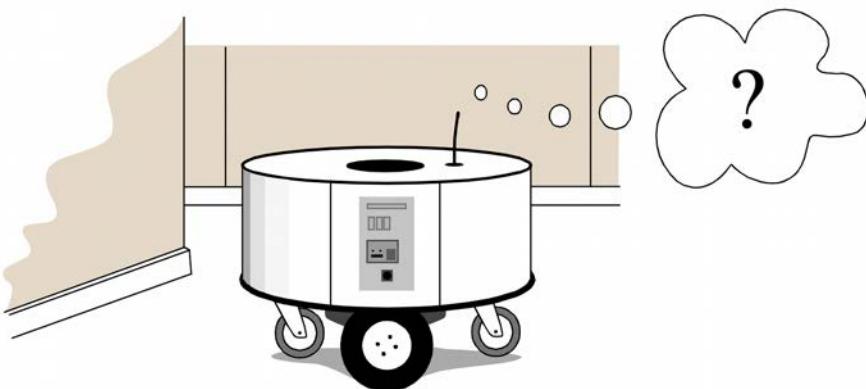


Lindsey's navigation map at The Collection

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



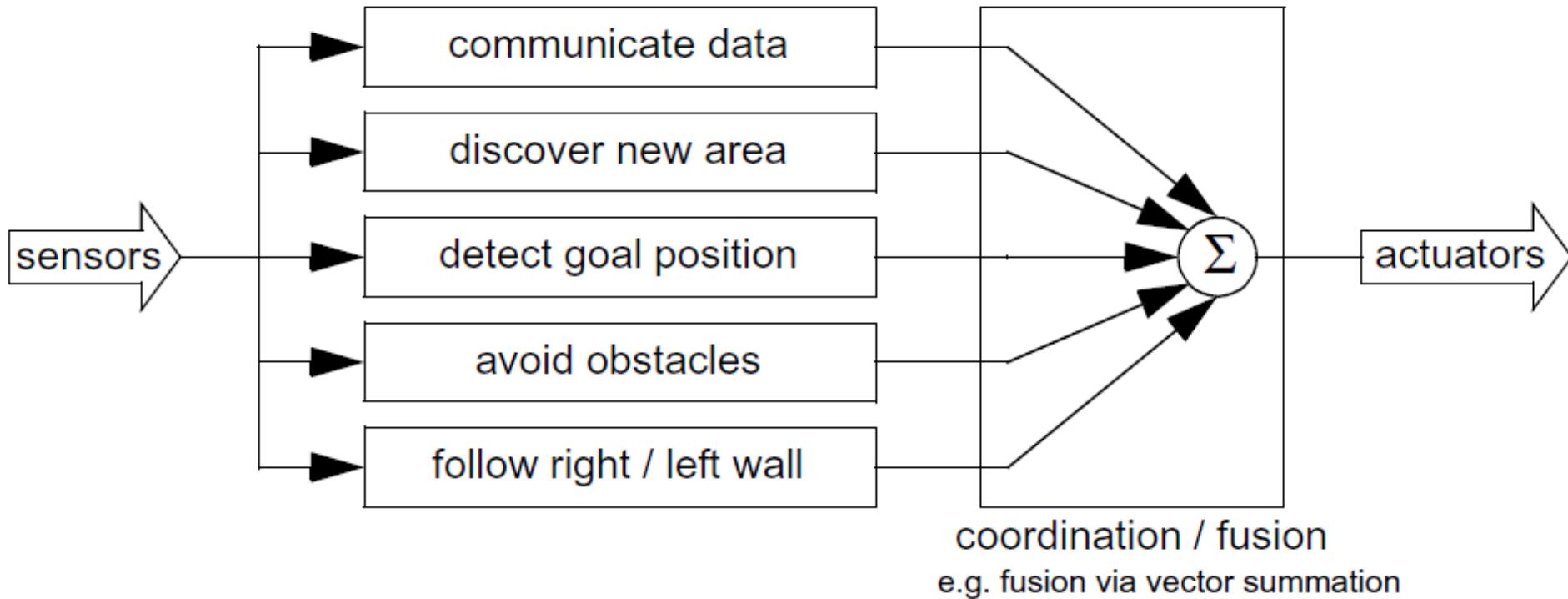
Map-based

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

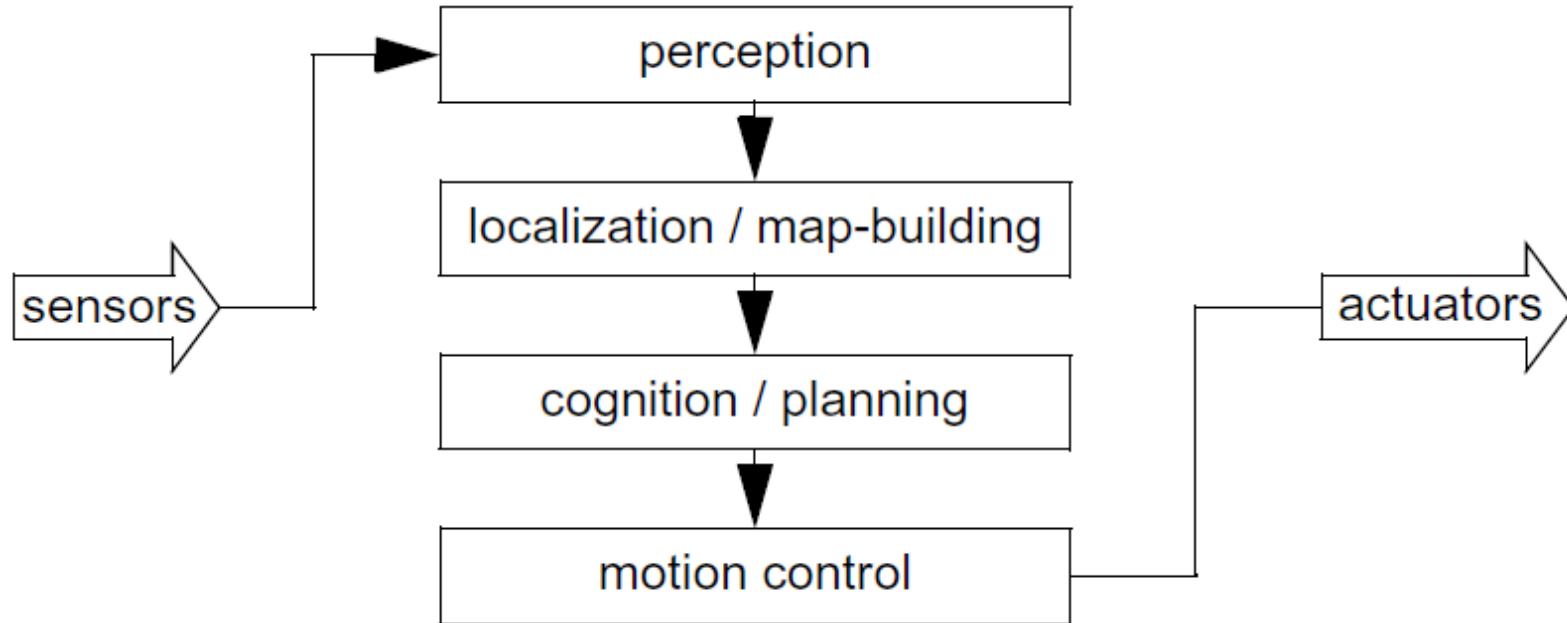
To localise or not to localise

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

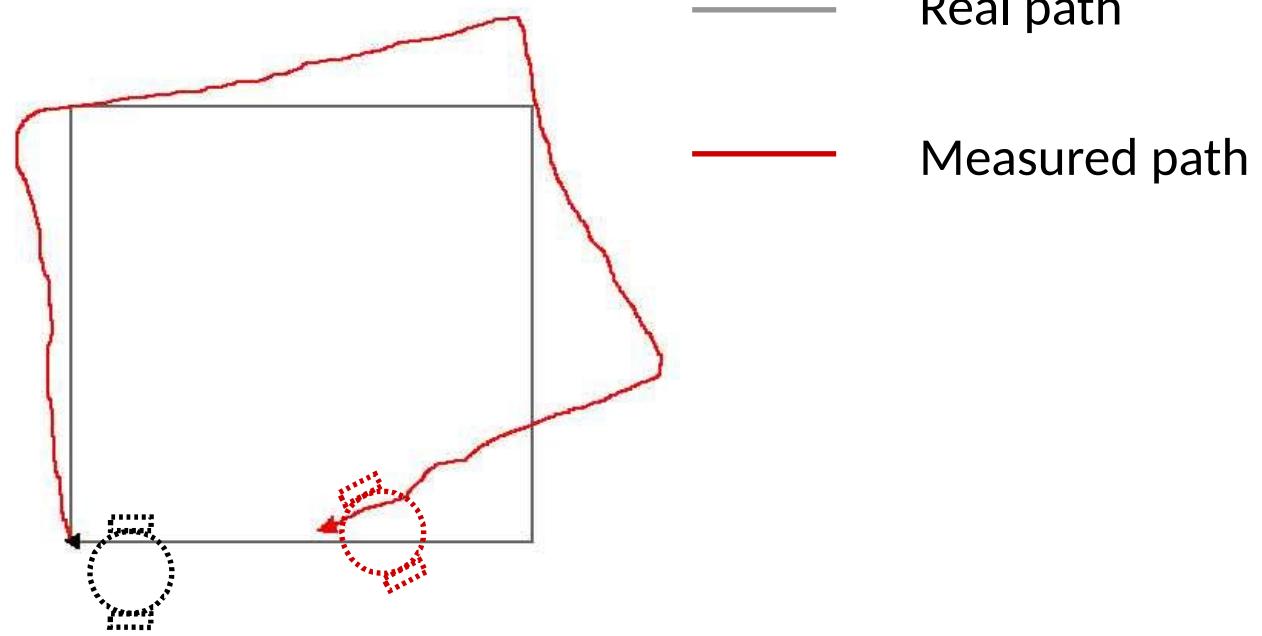


Kidnapped robot problem



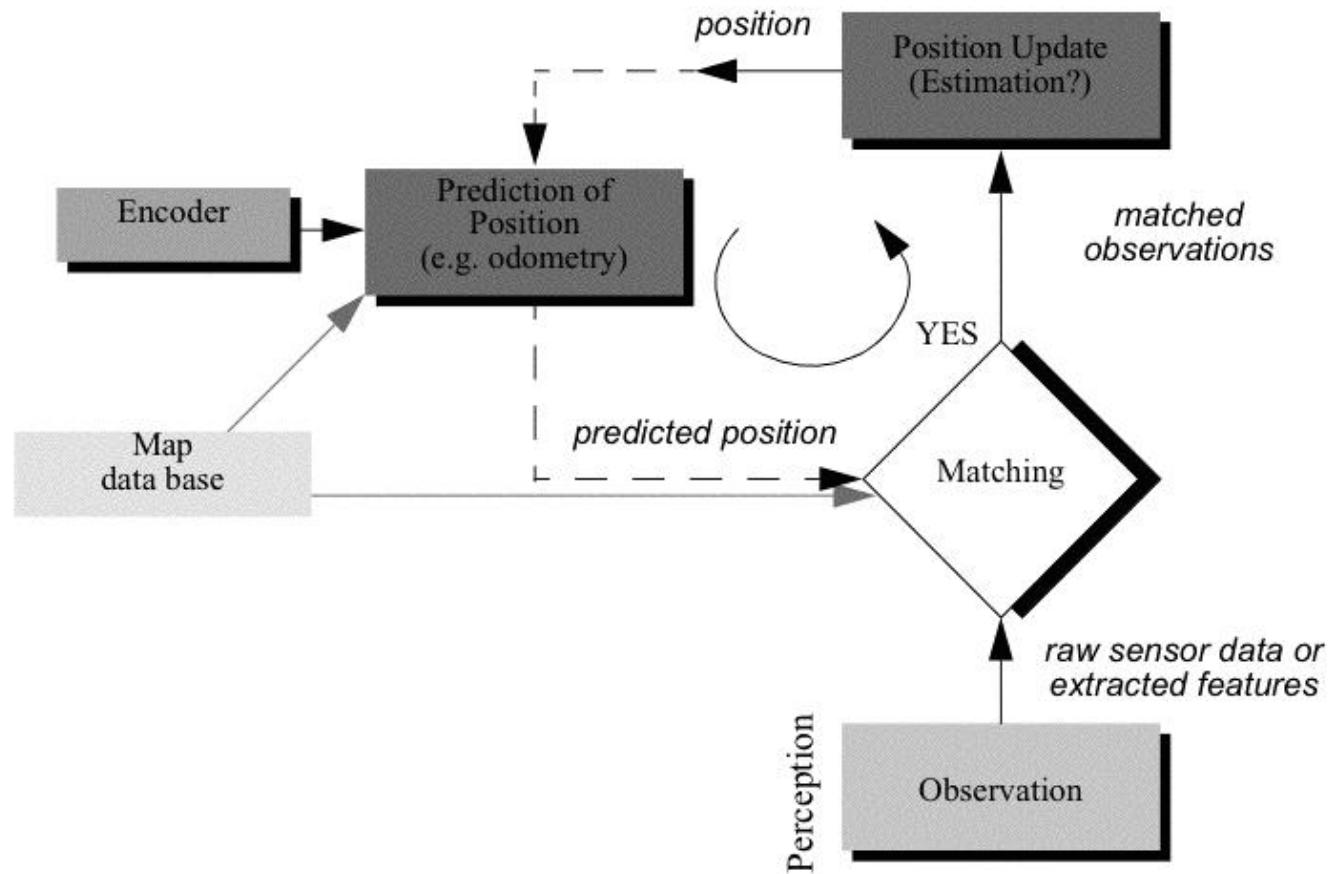
What's wrong with odometry?

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



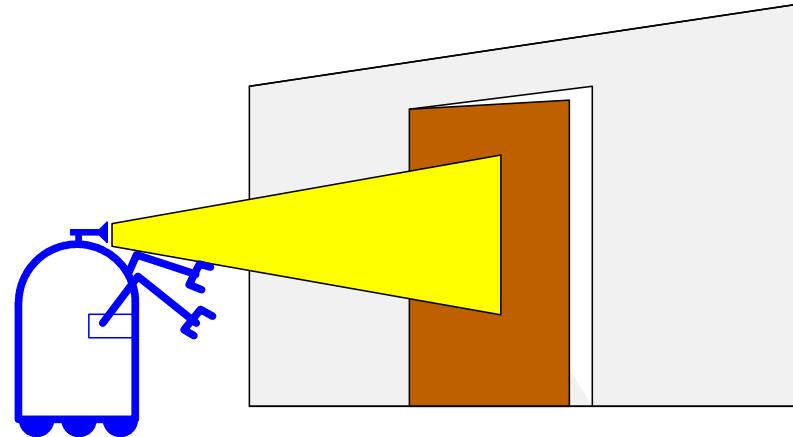
Metric localisation

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



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

Monte Carlo localisation

Perception update

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

Prediction update

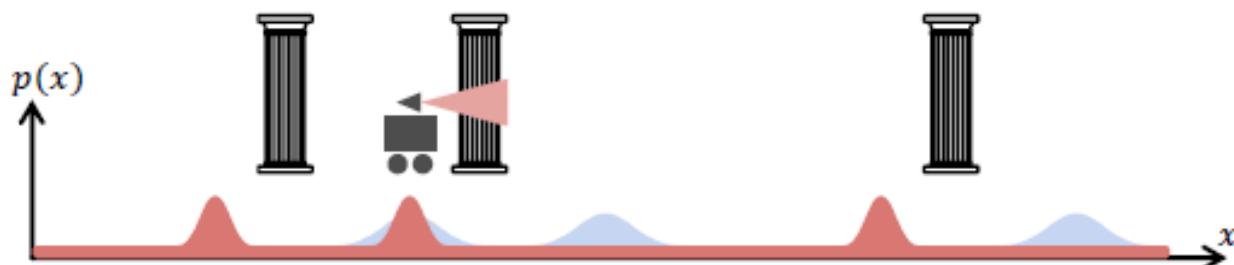
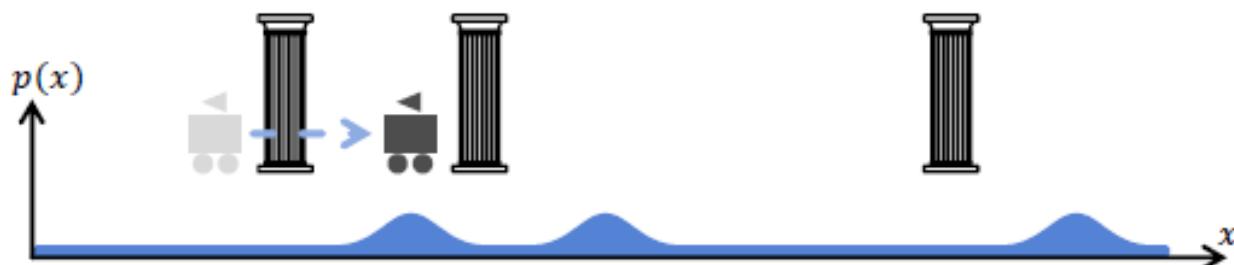
- Robot moves one metre forward
- Motion estimated by wheel encoder – accumulation of uncertainty

Perception update

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

Belief update

- Information fusion



What is a particle?

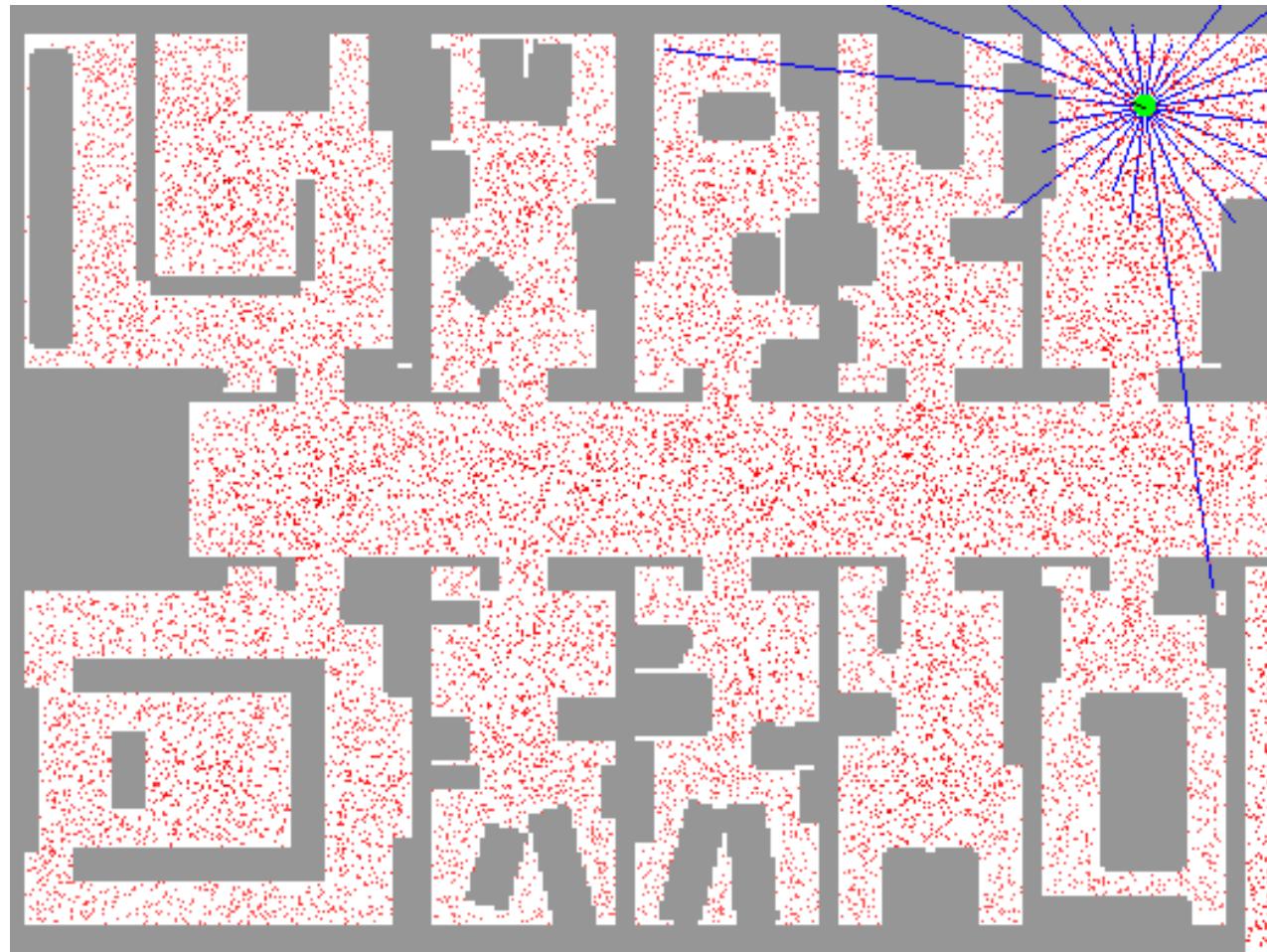
An individual state estimate, defined by:

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

Particle filters use many particles to represent the belief state

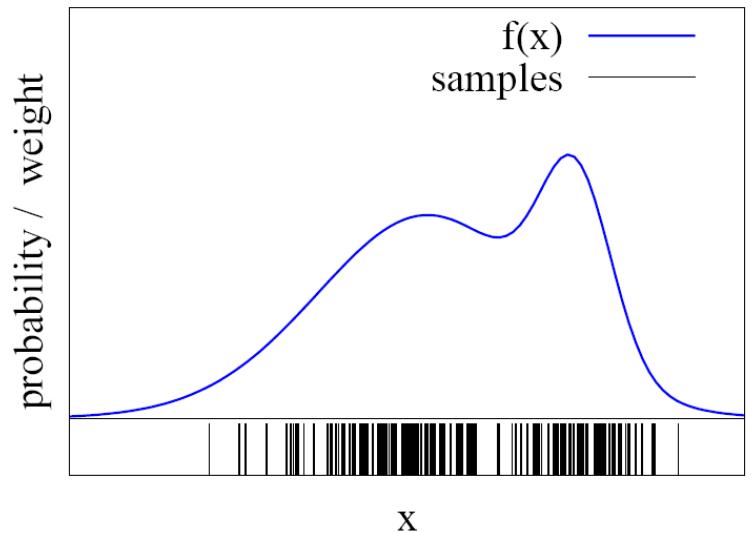
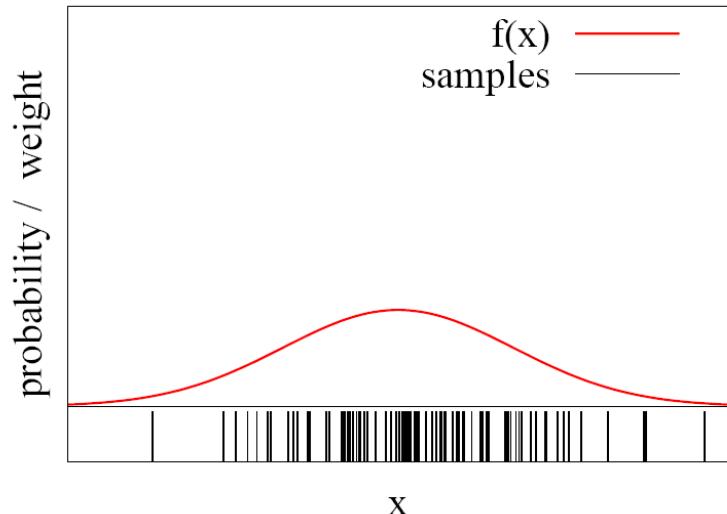
Randomised sampling

2D Monte Carlo localisation



Function approximation

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



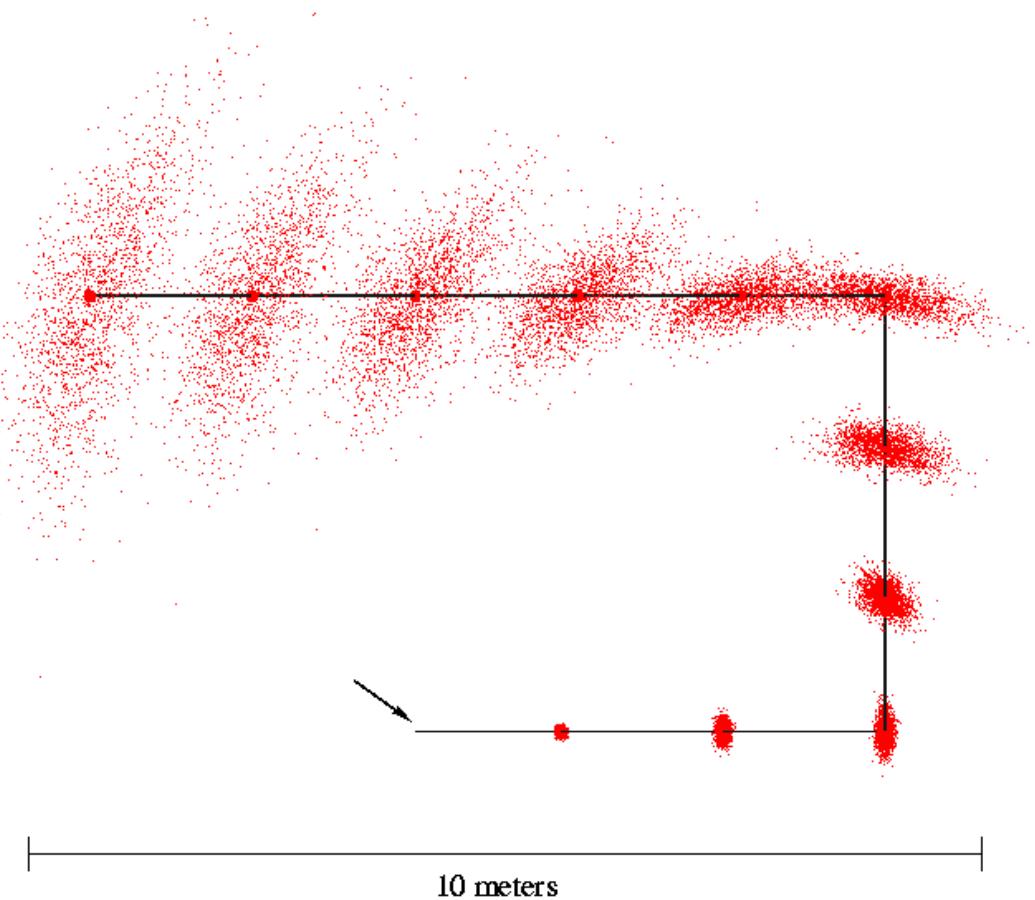
Monte Carlo Localisation (MCL)

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

Prediction step

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



Motion model

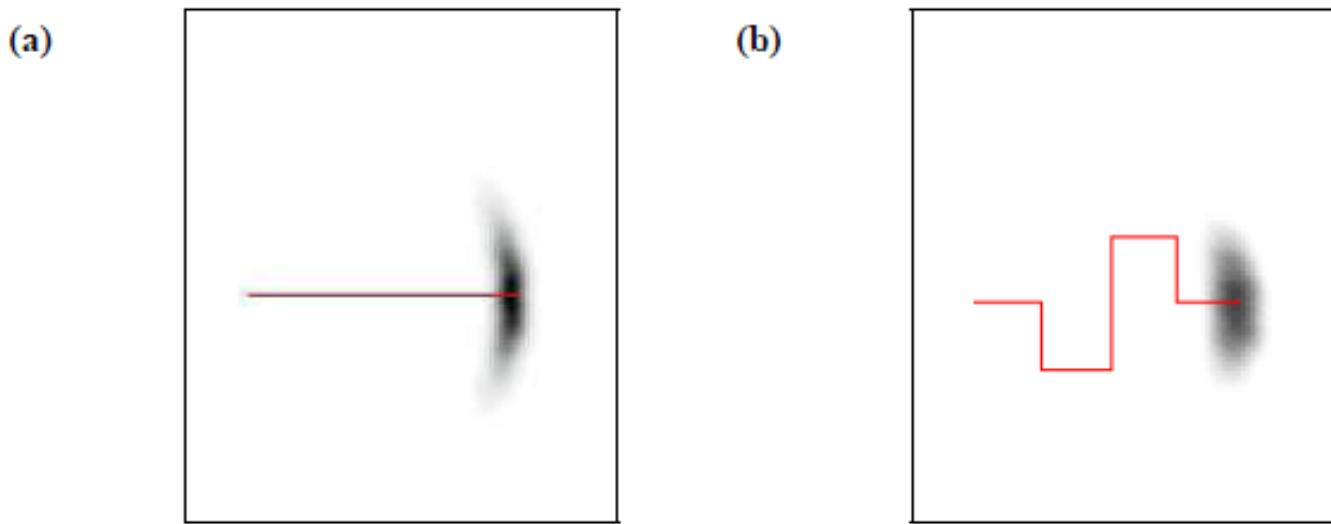
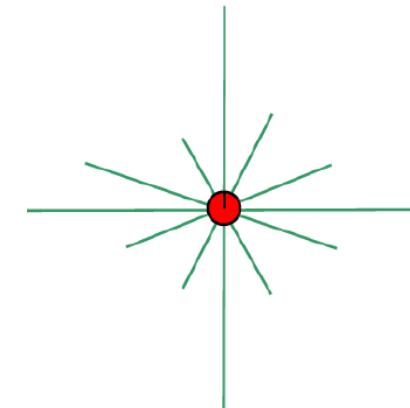


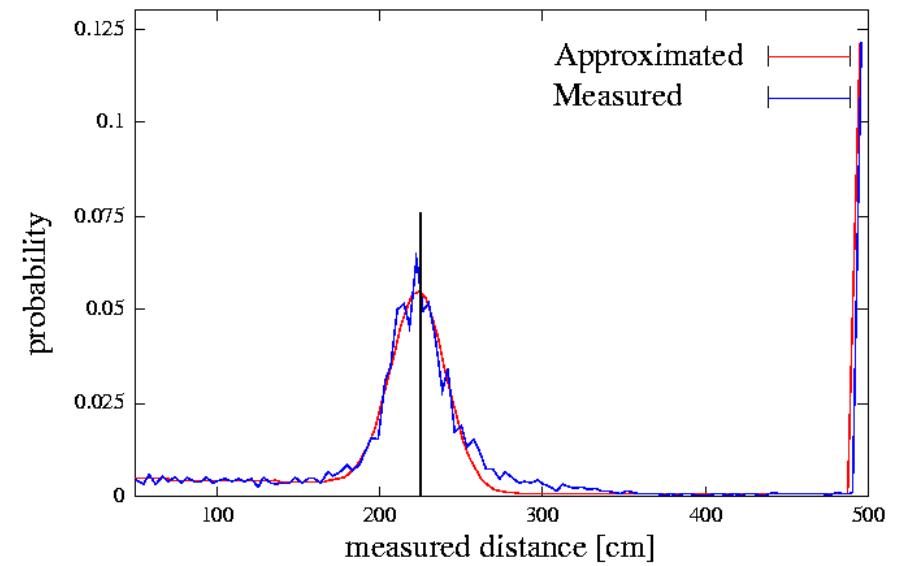
Figure 5.2 The motion model: Posterior distributions of the robot's pose upon executing the motion command illustrated by the solid line. The darker a location, the more likely it is. This plot has been projected into 2D. The original density is three-dimensional, taking the robot's heading direction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

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

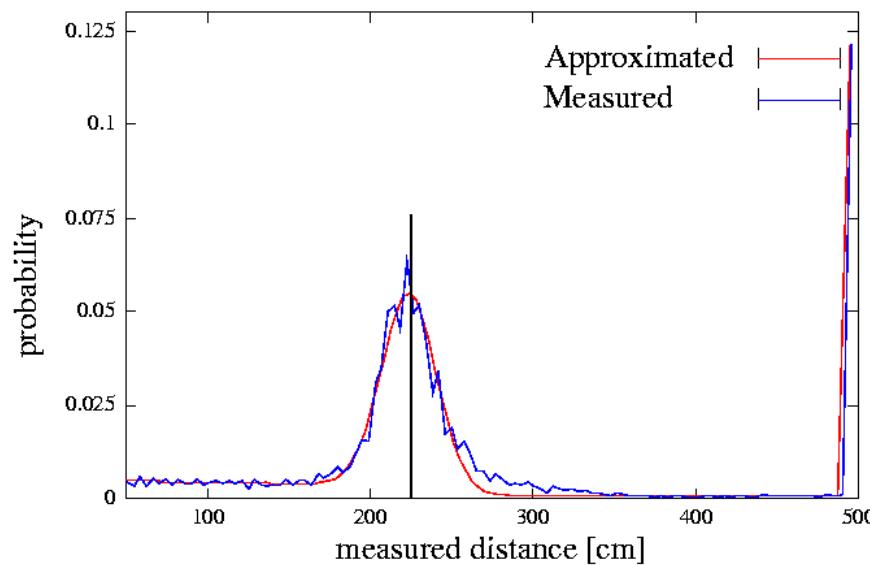


Laser sensor

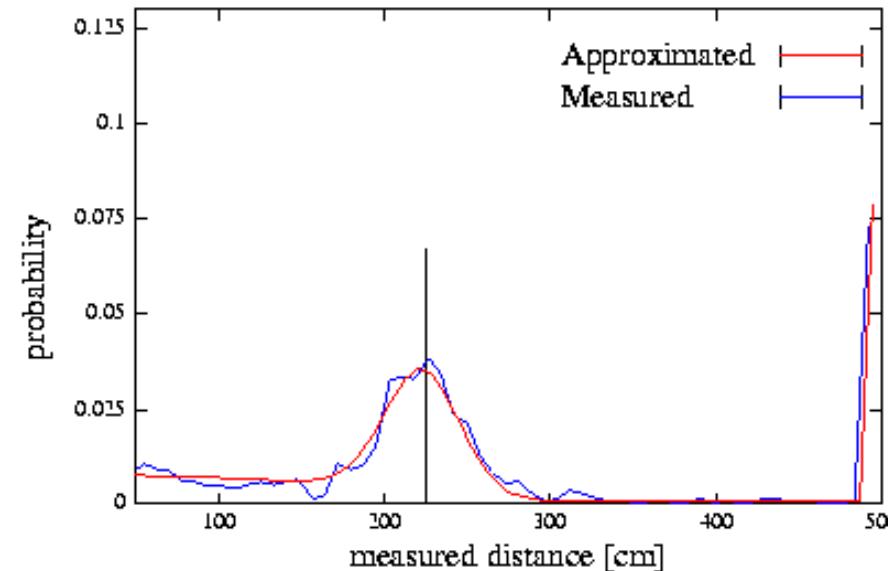


Sensor model

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



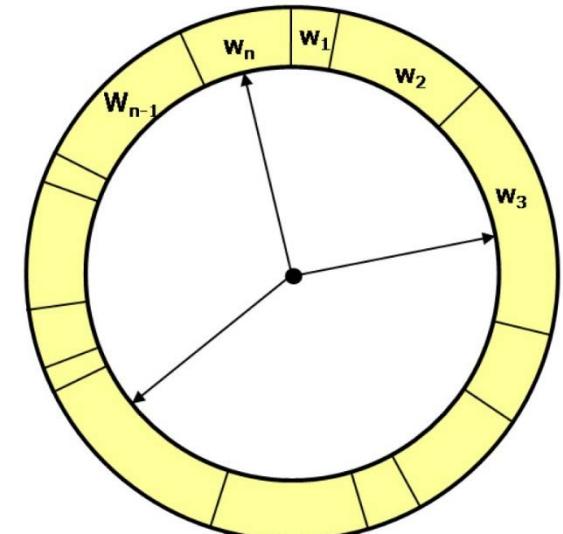
Laser sensor



Sonar sensor

Resample step

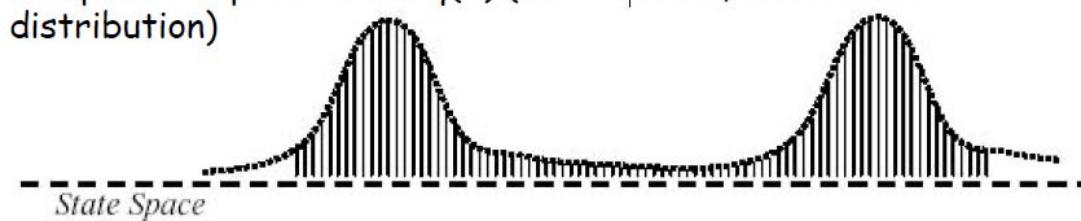
- New set of particles chosen
- “Survival of the fittest”
 - Each particle survives in proportion to its weight
 - Replace unlikely samples by more likely ones
- “Roulette wheel” resampling



Resampling



Sample from prior belief $q(x)$ (for instance, the uniform distribution)



Compute importance weights, $w(x) = p(x) / q(x)$



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

Particle filter algorithm

$x_{t+1} = \emptyset$

for $j = 1$ to m

// apply the transition model

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

// apply the sensor model

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

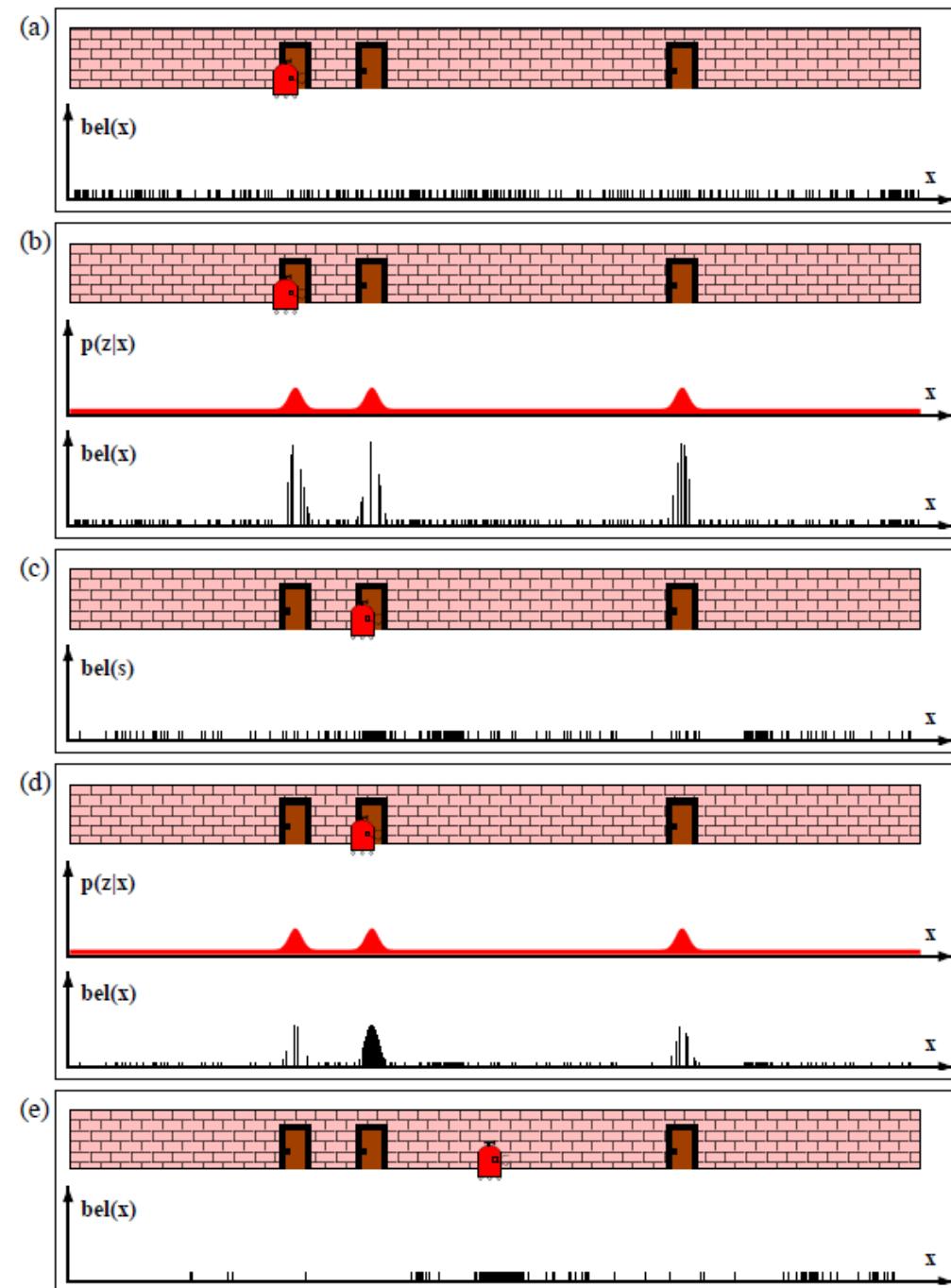
// pick points randomly but biased by their weight

for $j = 1$ to m

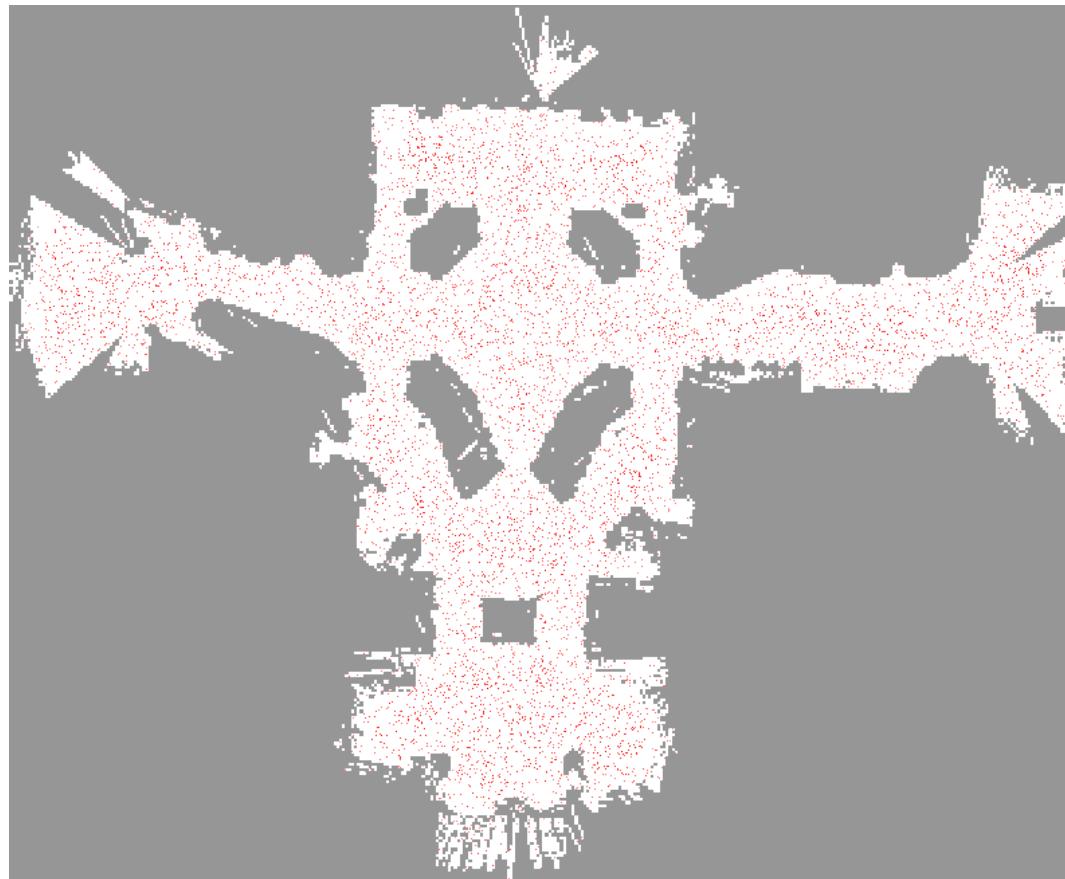
pick a random $x_{t+1}^{(i)}$ from x_{t+1} according to $w_{t+1}^{(1)}, \dots, w_{t+1}^{(m)}$

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

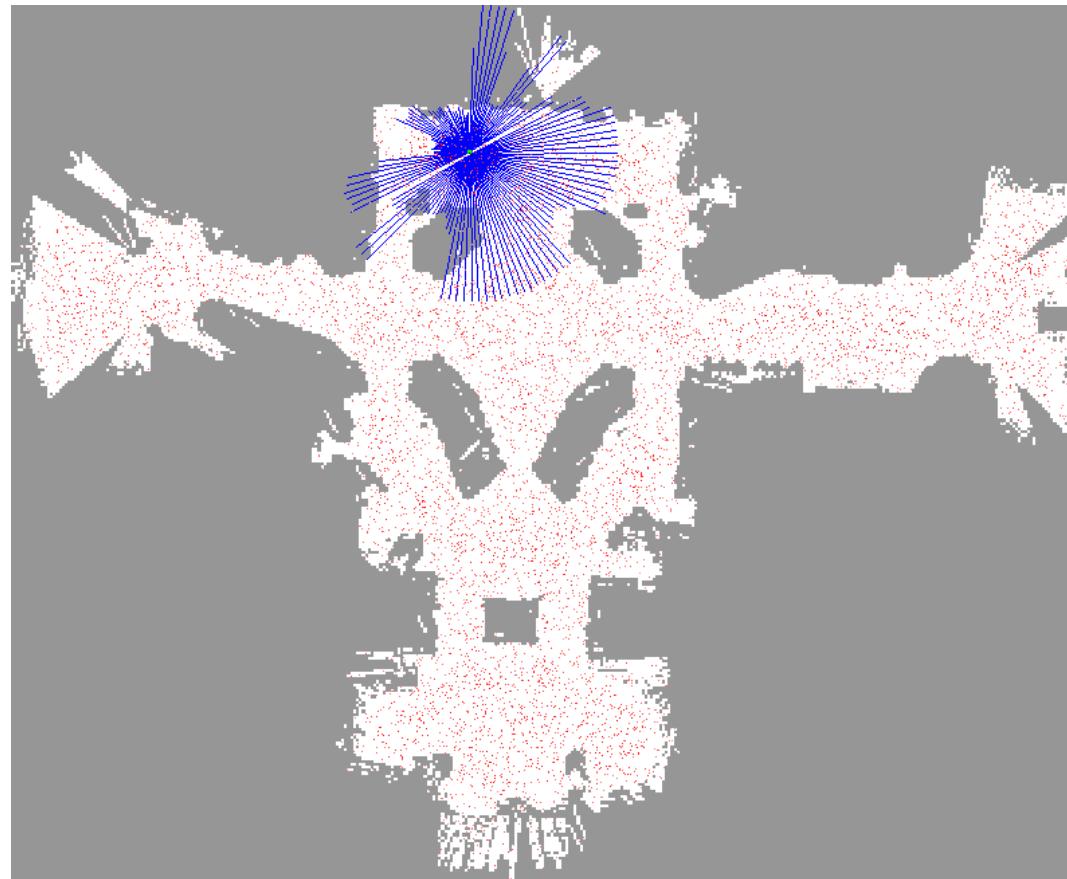
return x_{t+1}



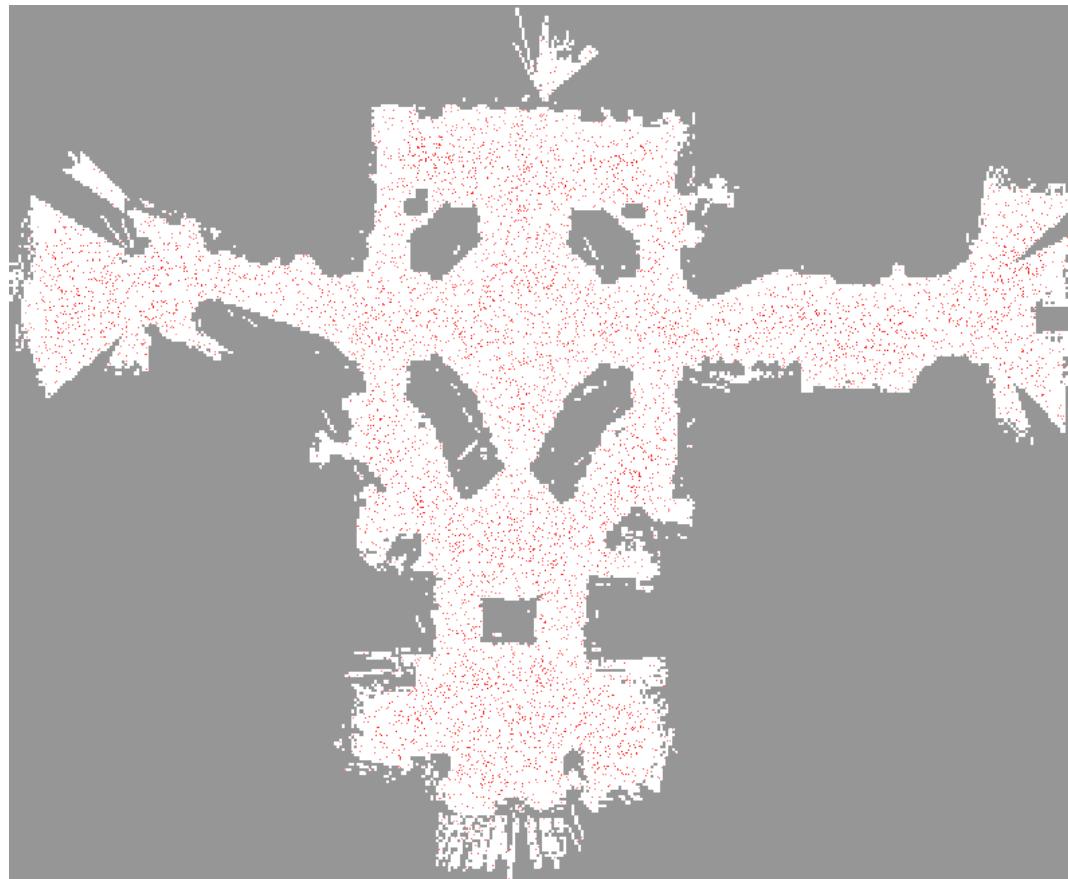
Initialisation



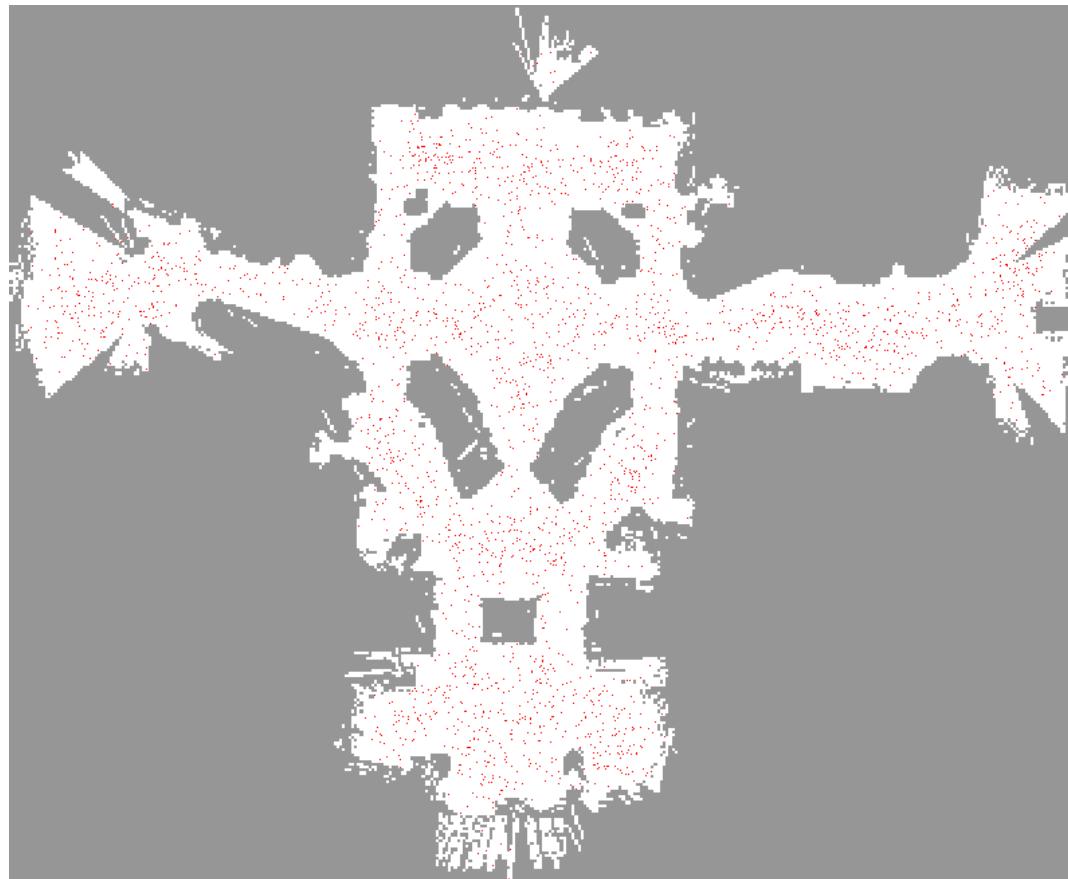
Measurement



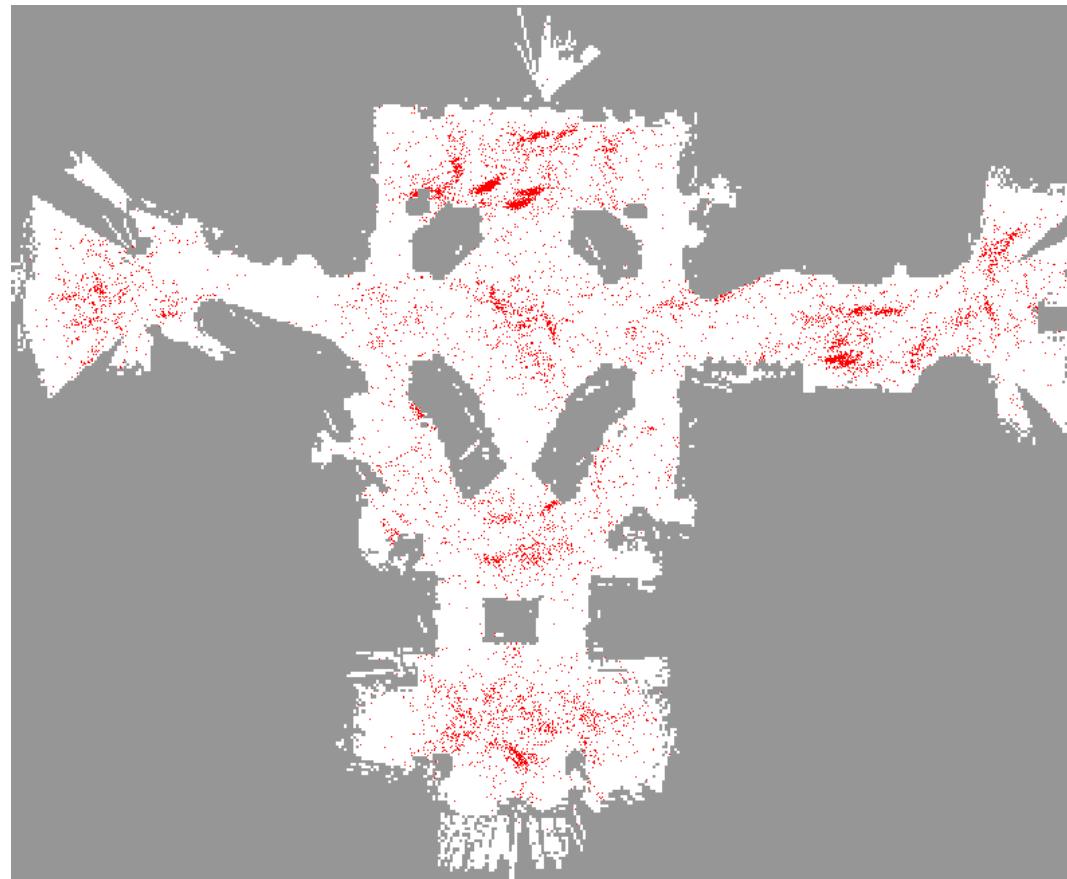
Weight update



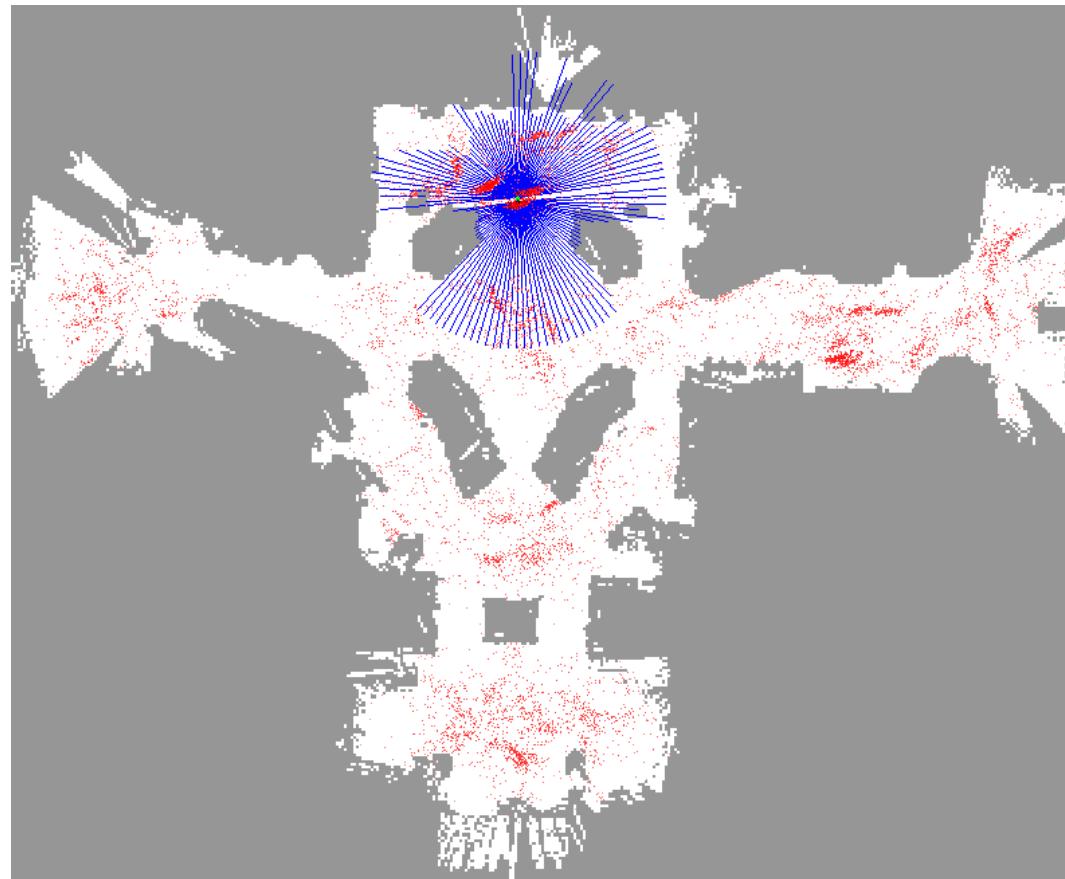
Resampling



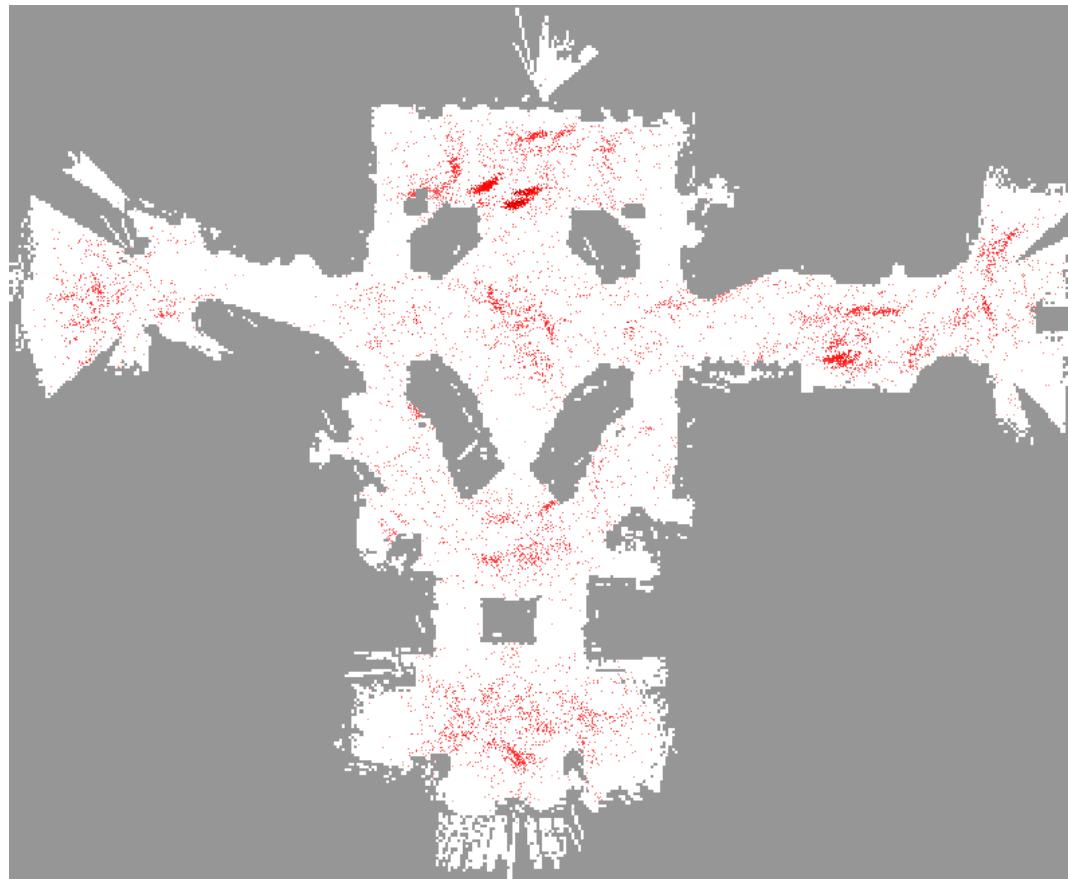
Motion update



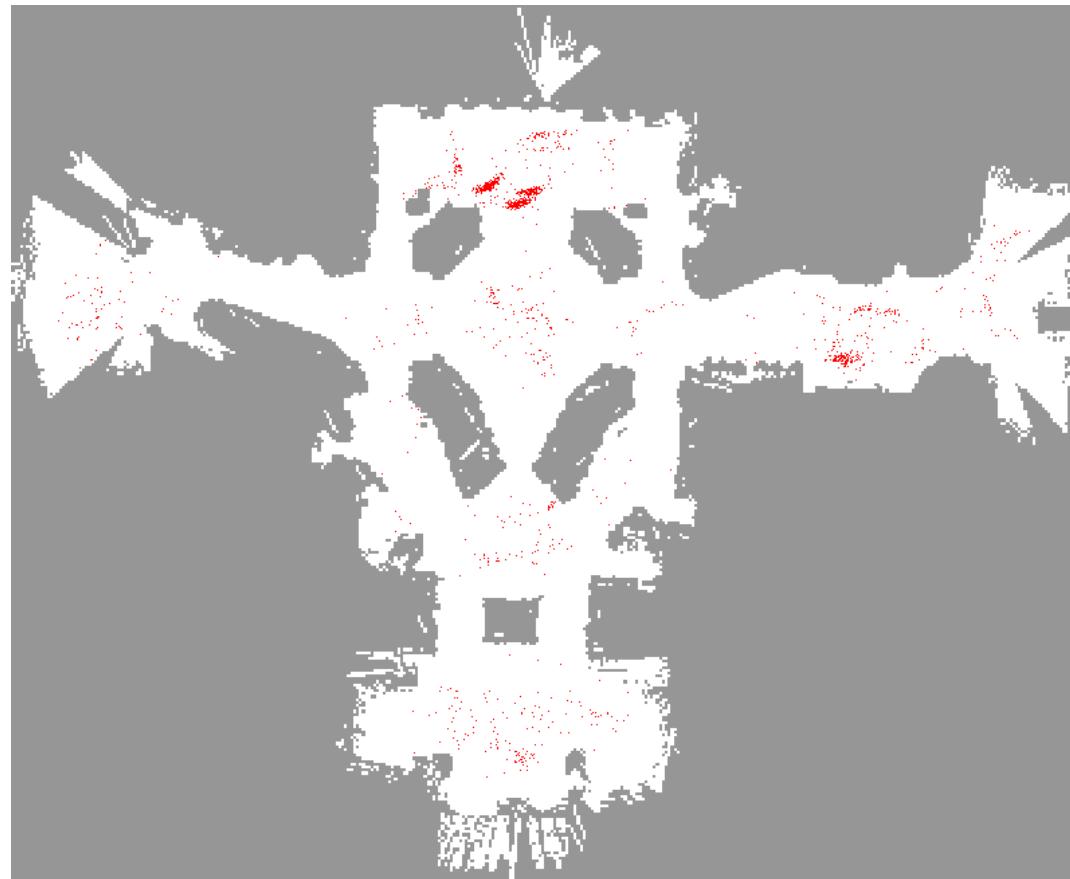
Measurement



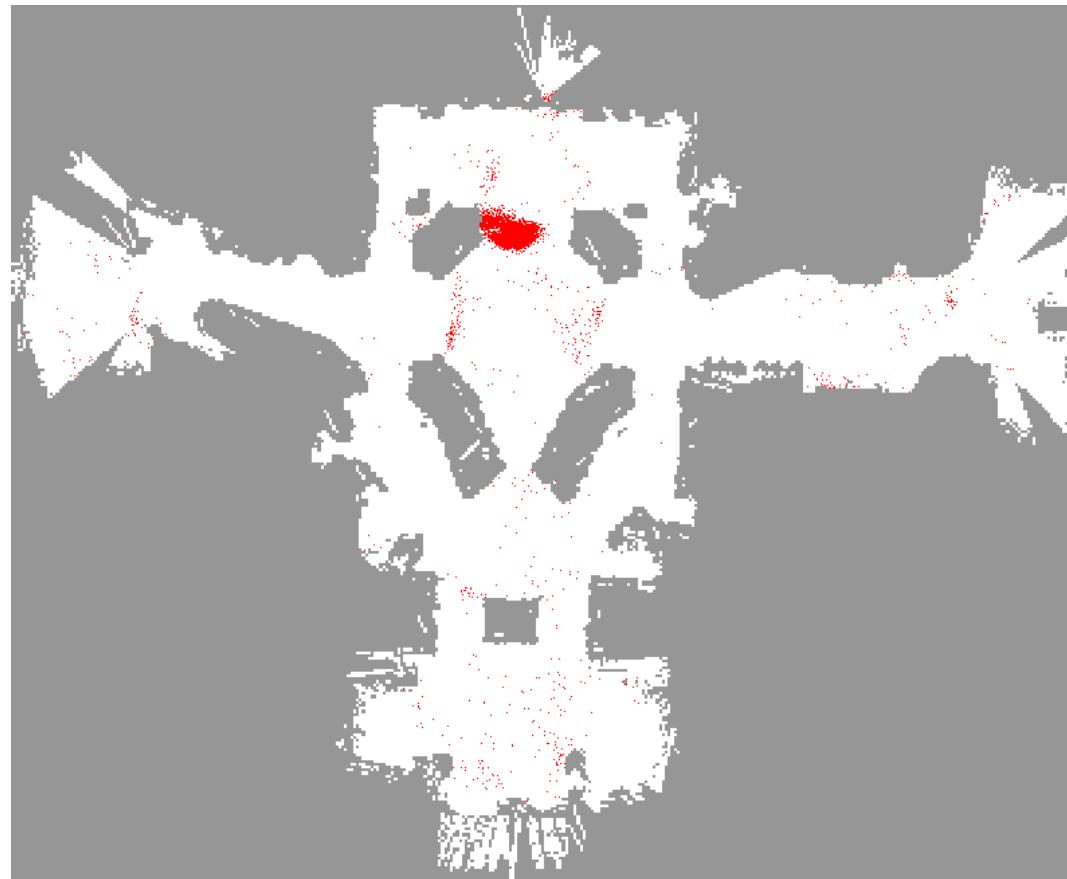
Weight update



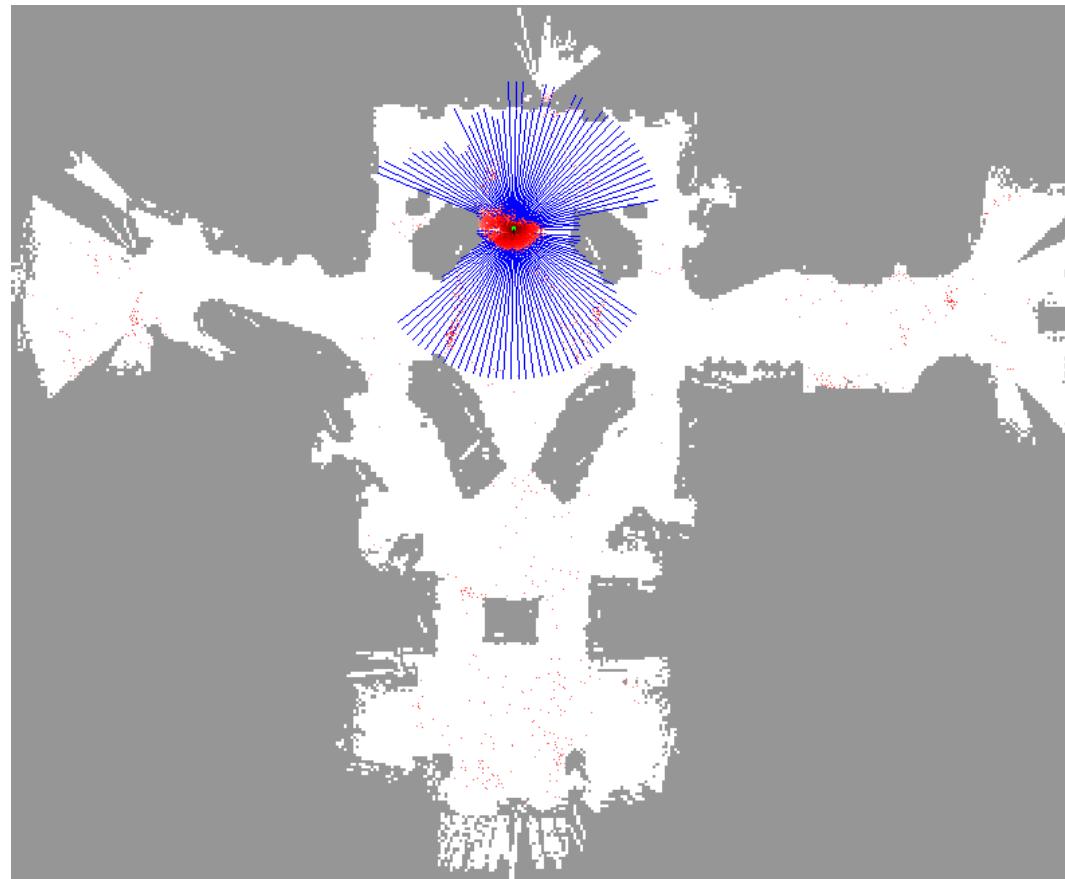
Resampling



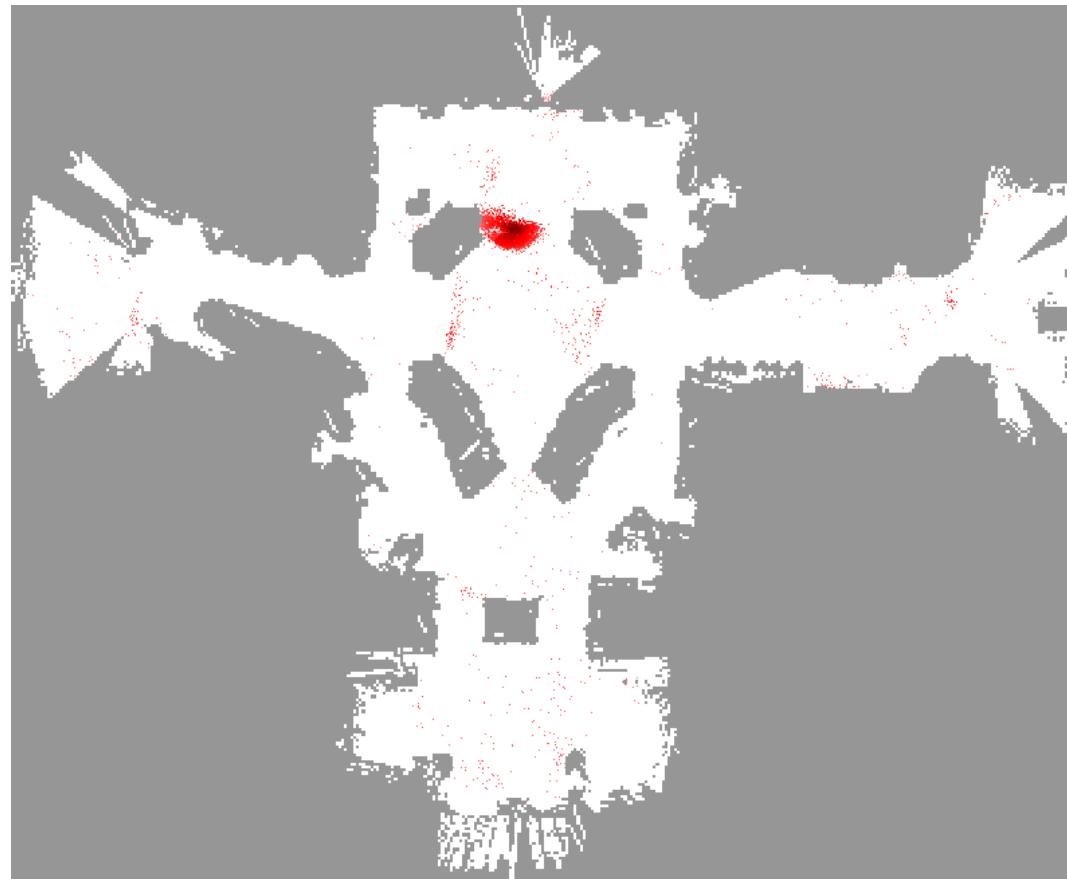
Motion update



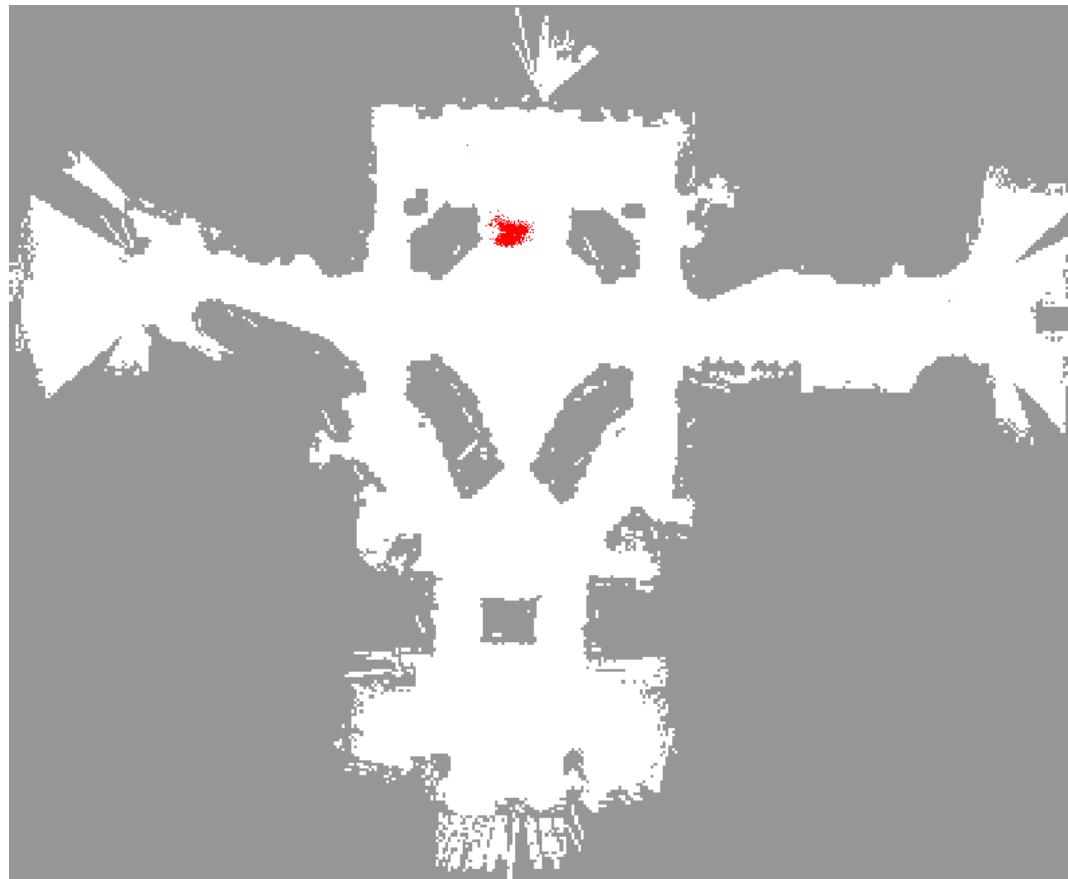
Measurement



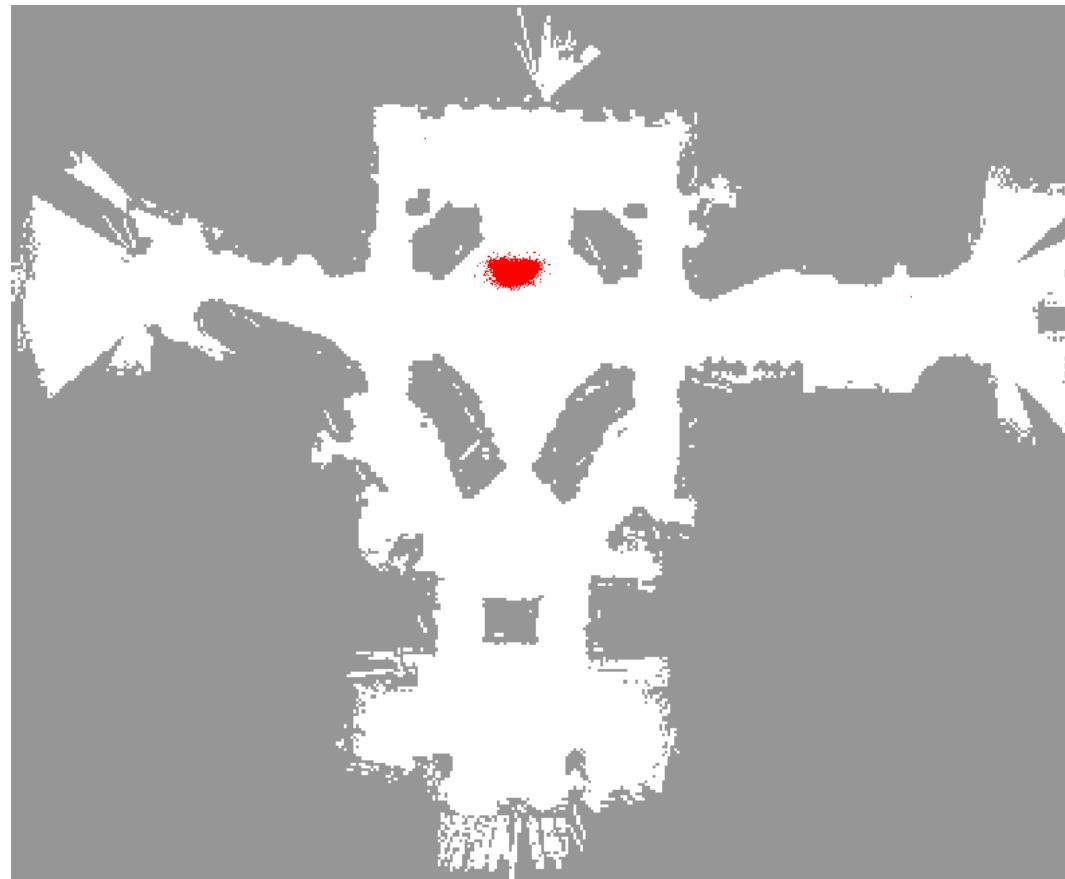
Weight update



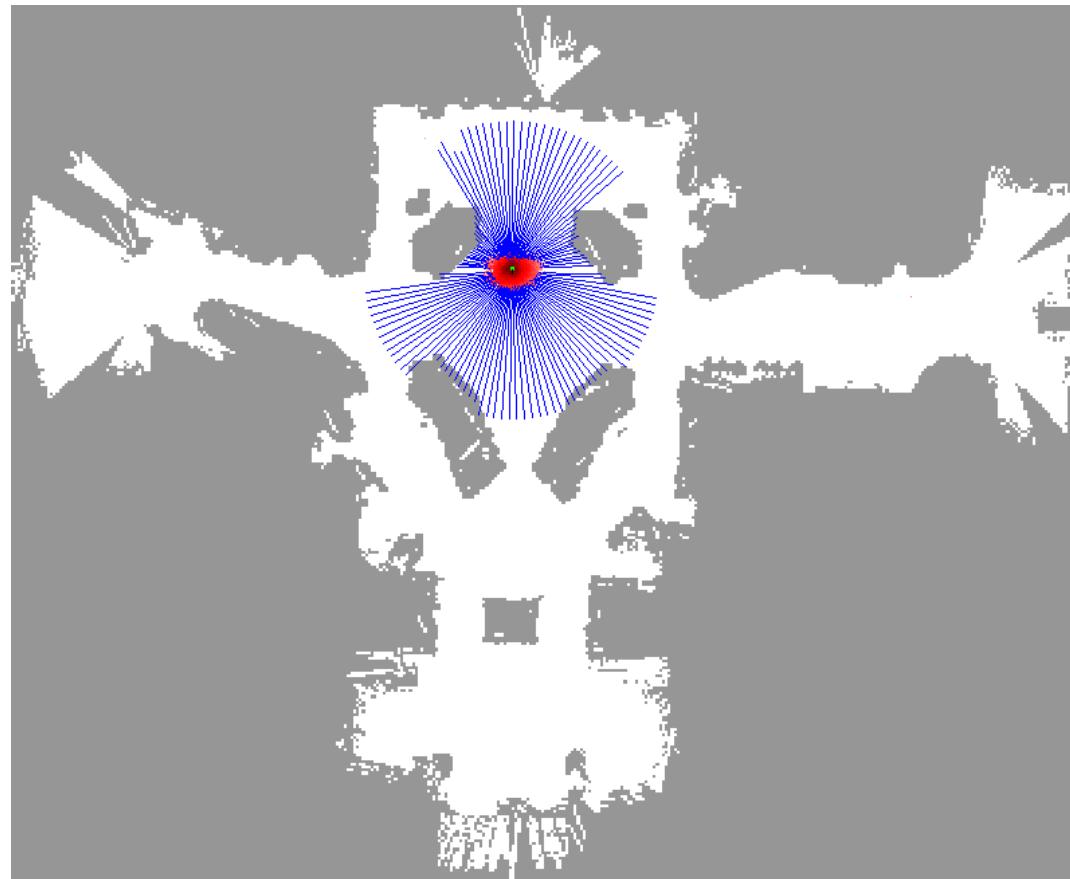
Resampling



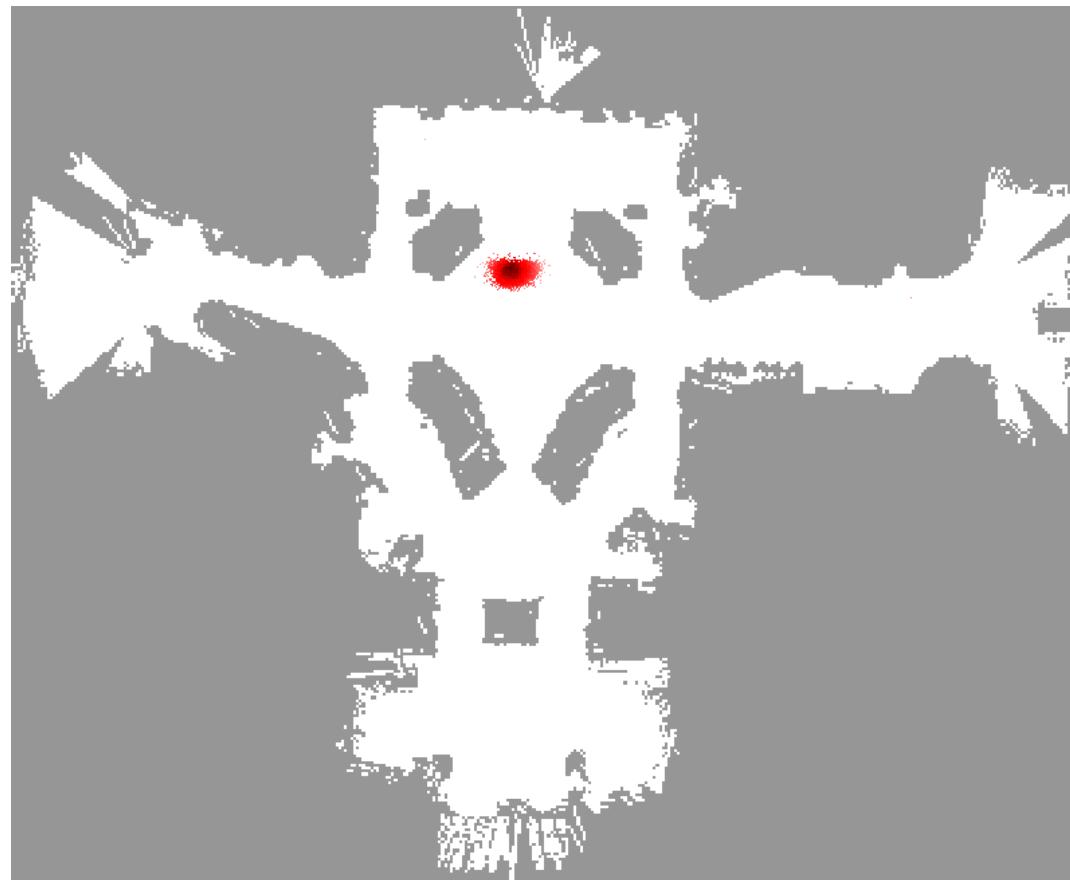
Motion update



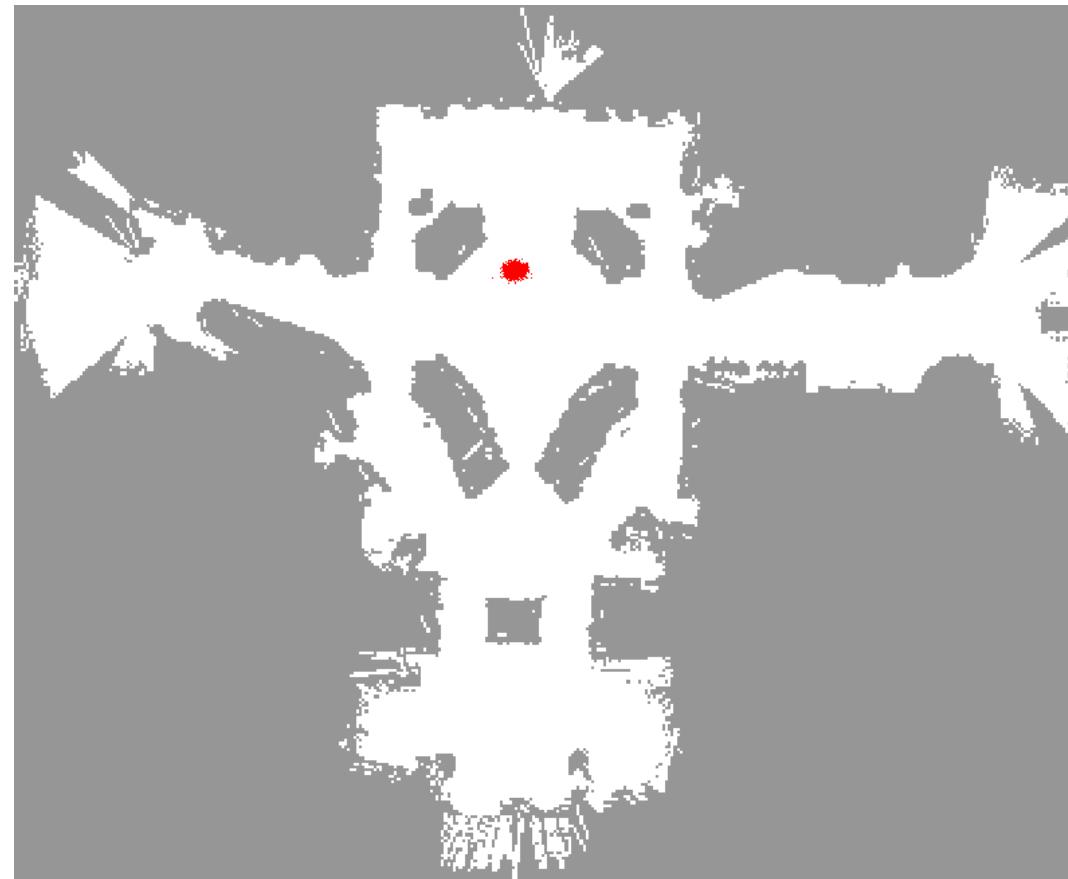
Measurement



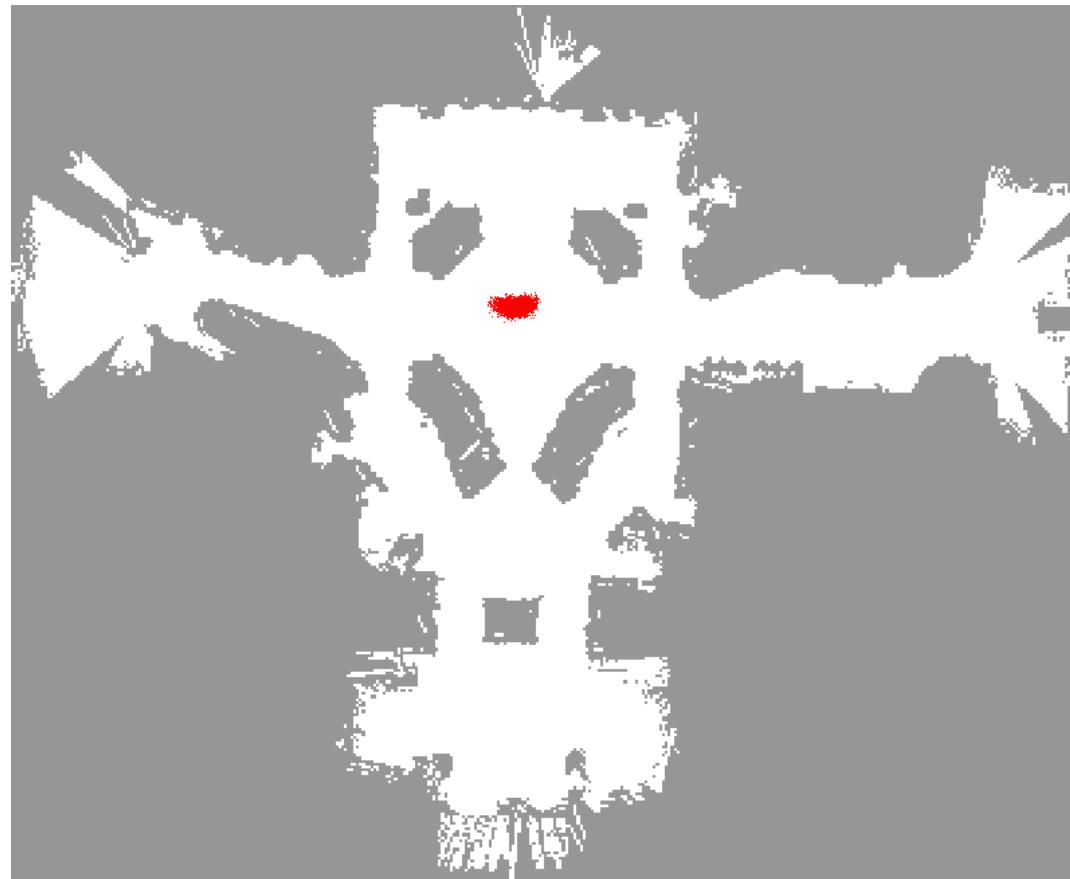
Weight update



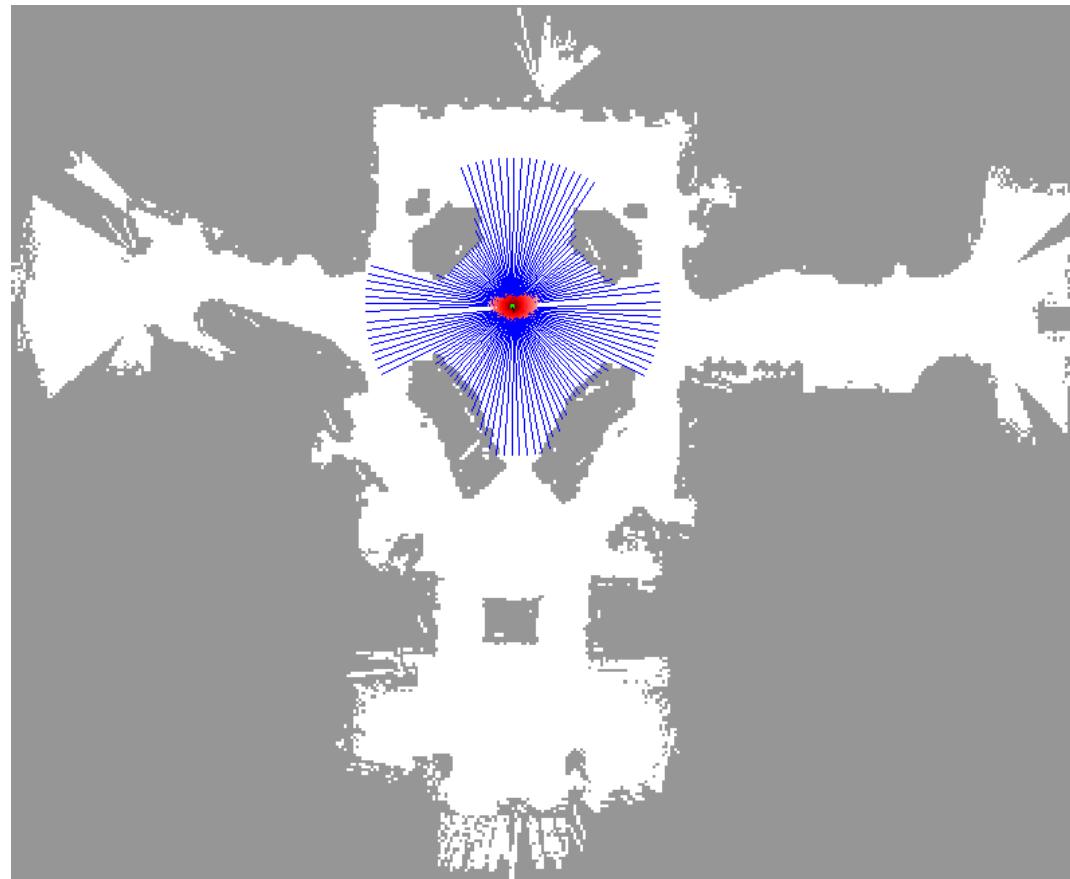
Resampling



Motion update



Measurement



Monte Carlo localisation summary

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

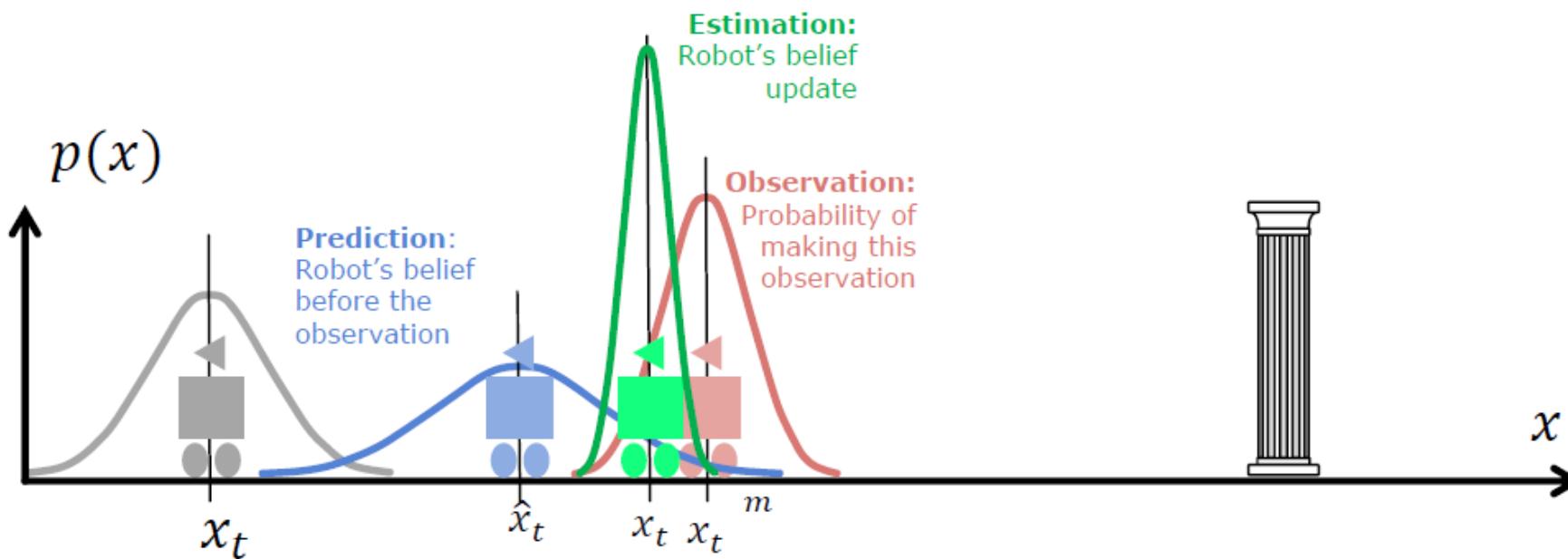


Kalman Filter (KF) localisation

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

Kalman Filter (KF) localisation

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



Kalman Filter localisation pros / cons

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

Pros:

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

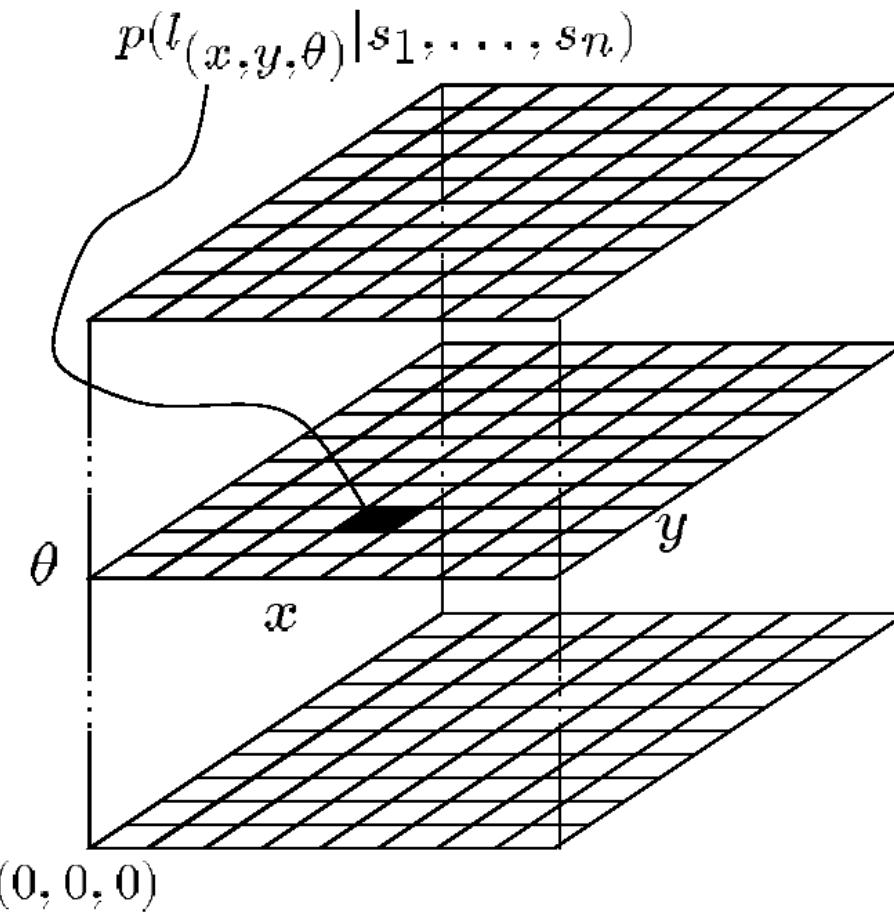
Cons:

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

Markov localisation

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

Markov localisation



Markov localisation pros / cons

Uses an explicitly specified probability distribution across all possible robot positions

Pros:

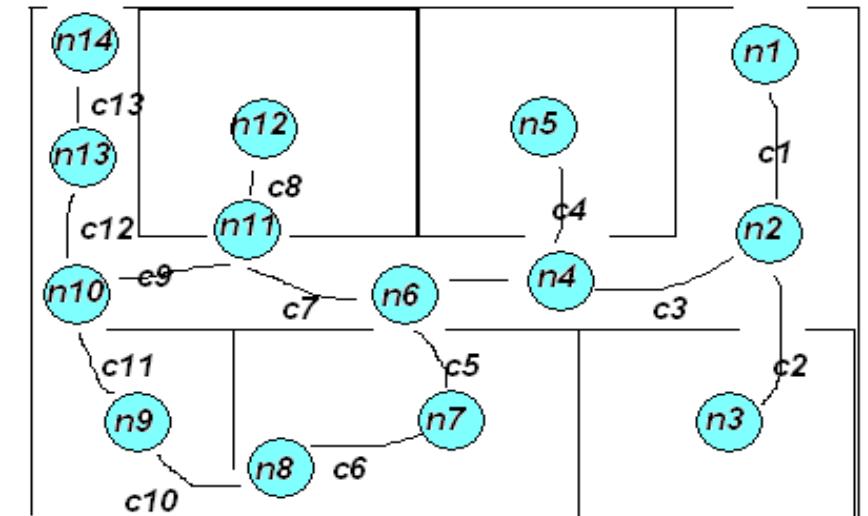
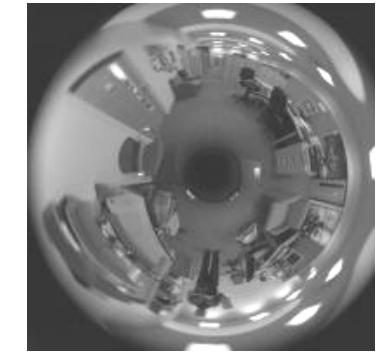
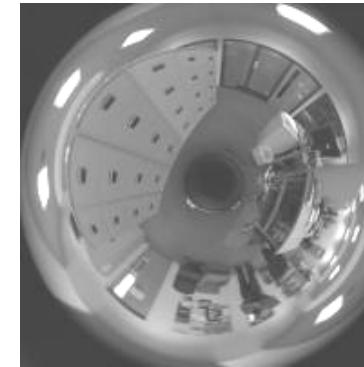
- Allows for localisation starting from any unknown position
- Can recover from ambiguous situations

Cons:

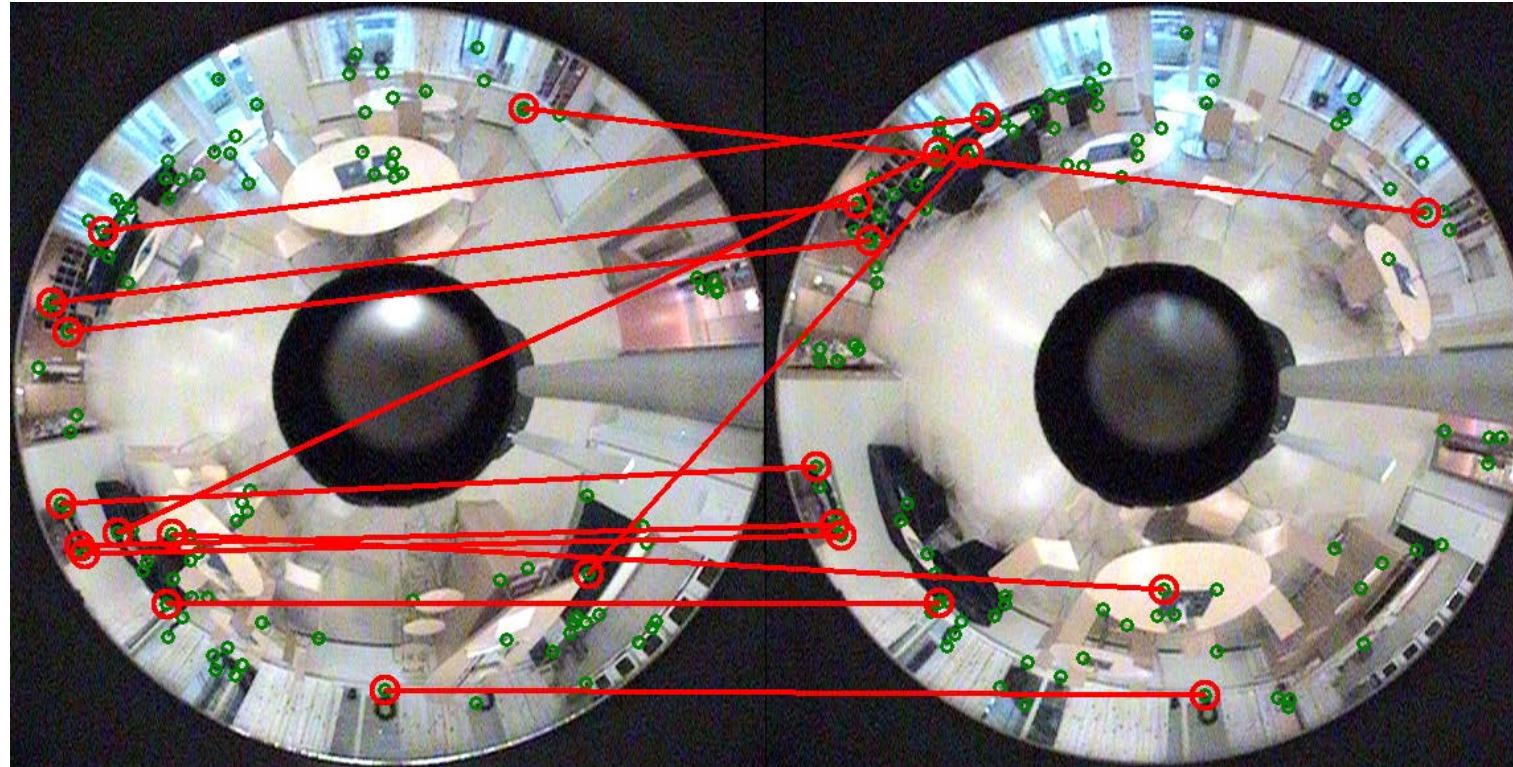
- Requires a discrete representation of the space, such as a geometric grid or a topological graph
- Consumes significant memory and computational resources

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

ROS Navigation stack

<http://wiki.ros.org/navigation> - taking information from odometry, sensors, and a goal pose, safely issues velocity commands.

http://wiki.ros.org/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.

<http://wiki.ros.org/gmapping> - 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.

<http://wiki.ros.org/amcl> - implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.

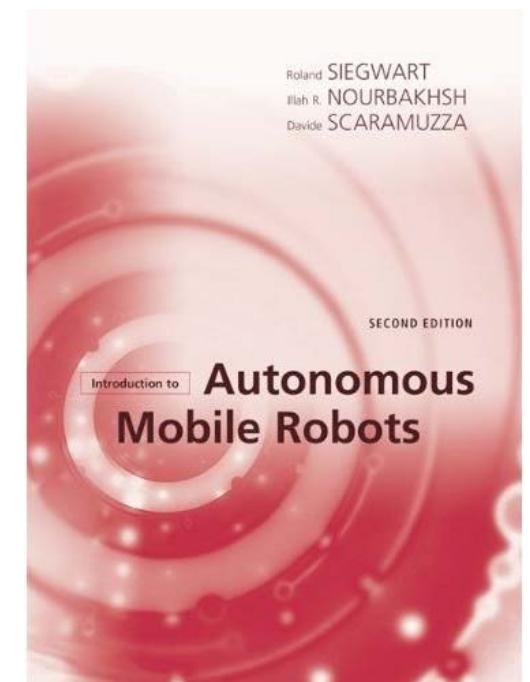
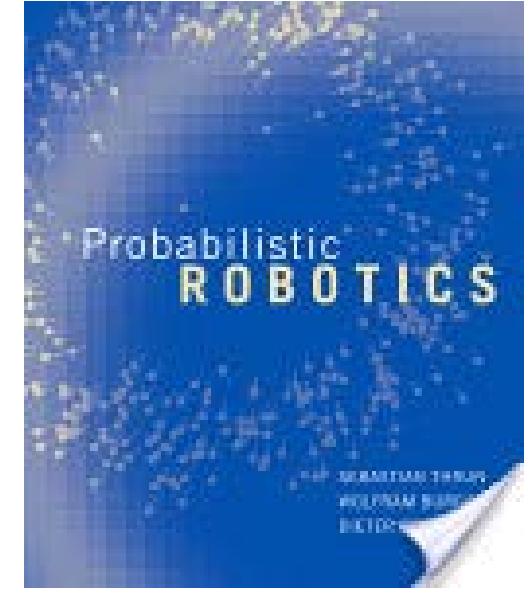
http://wiki.ros.org/global_planner - provides an implementation of a fast, interpolated global planner for navigation. Dijkstra/A*

http://wiki.ros.org/base_local_planner - provides implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane.

http://wiki.ros.org/move_base - links together a global and local planner to accomplish its global navigation task.

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

apolydoros@lincoln.ac.uk

Based on the slides of Dr. Alan Millard

END OF MODULE EVALUATION

Please take part in this short Module Evaluation survey.

It only takes five minutes to complete, and your feedback will help us to do more of what you like, as well as improve areas where you think we need to develop.

To take part please use the QR code or web address
<https://lncn.ac/sembmodval> to access the survey

YOUR MODULE
CODE IS
CMP3103M



Today's lecture

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

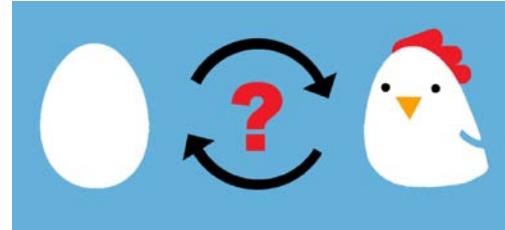
Simultaneous Localisation and Mapping (SLAM)

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?

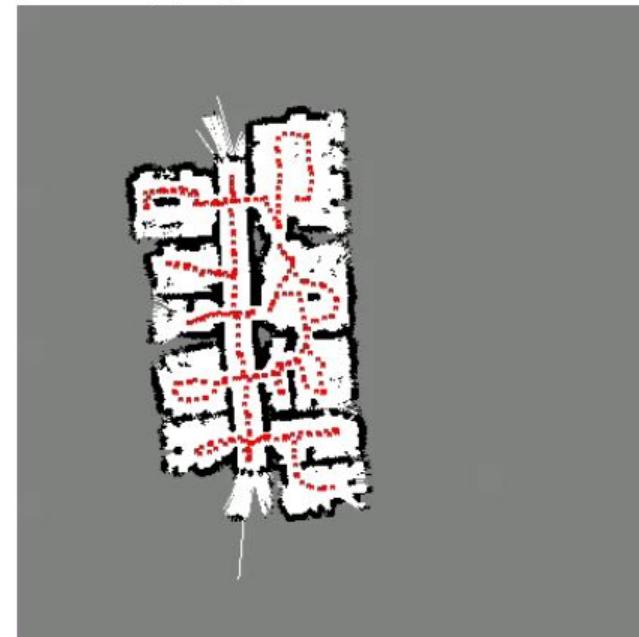
- When a robot must be truly autonomous (no human input)
- When there is no prior knowledge about the environment
- When we cannot rely exclusively on external positioning systems (e.g. GPS)
- When the robot needs to know where it is

Why is self-localisation needed?

Example of mapping using odometry



Example of simultaneous localisation and mapping



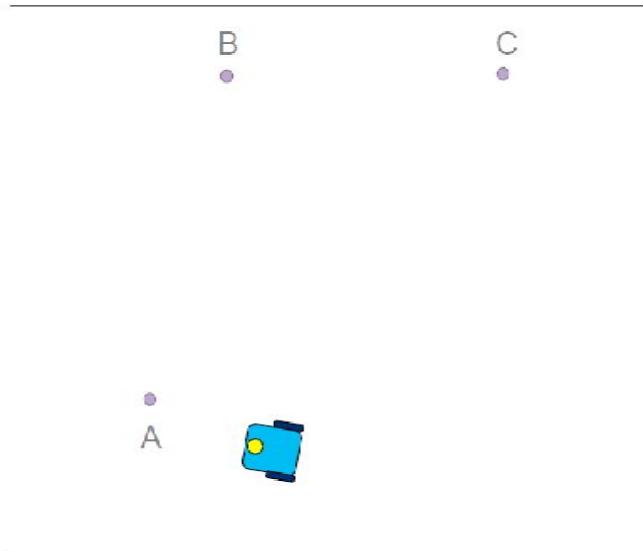
SLAM with a Gaussian filter

Use internal representations for:

- The positions of landmarks
- The camera parameters

Assumption:

- Robot's uncertainty at starting point is zero

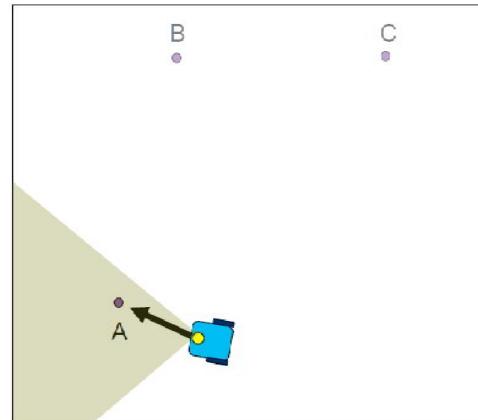


Start: robot has zero uncertainty

SLAM with Gaussian filter

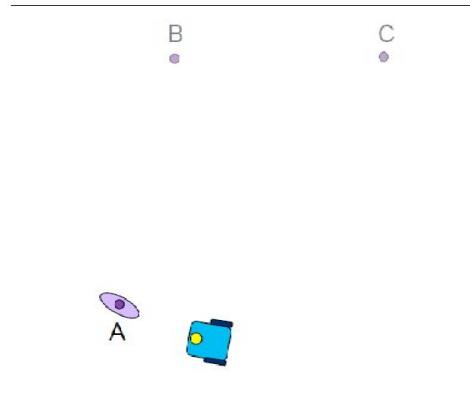
On every frame:

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



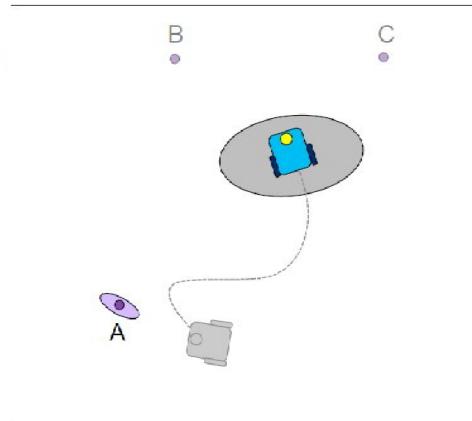
SLAM with Gaussian filter

- 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

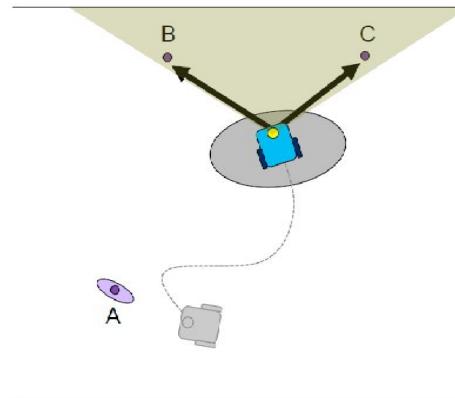
- As the robot moves, its pose uncertainty increases
 - Obeying the robot's motion model
- e.g. control commands: turn right, drive on for 1m
 - Uncertainty is added due to wheel slippage and other imprecisions



Robot moves forwards: uncertainty grows

SLAM with Gaussian filter

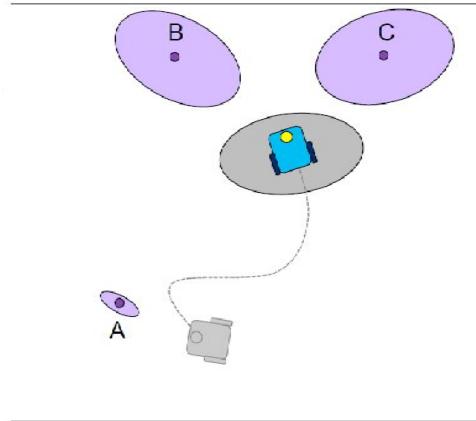
- Robot observes two new features
 - B and C



Robot makes first measurements of B & C

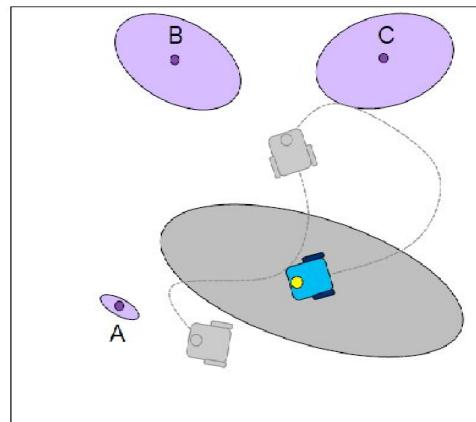
SLAM with Gaussian filter

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



SLAM with Gaussian filter

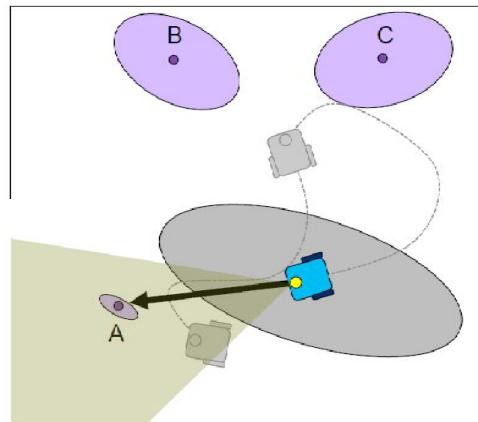
- Robot moves again
- Its uncertainty increases
 - Based on motion model



Robot moves again: uncertainty grows more

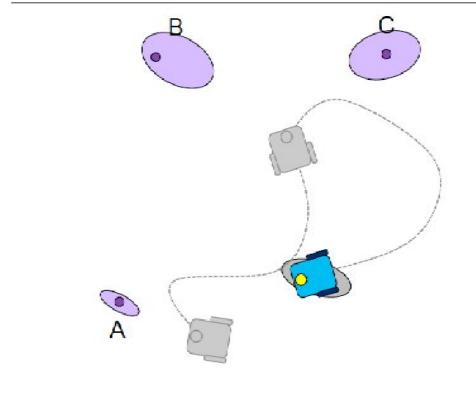
SLAM with Gaussian filter

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



SLAM with Gaussian filter

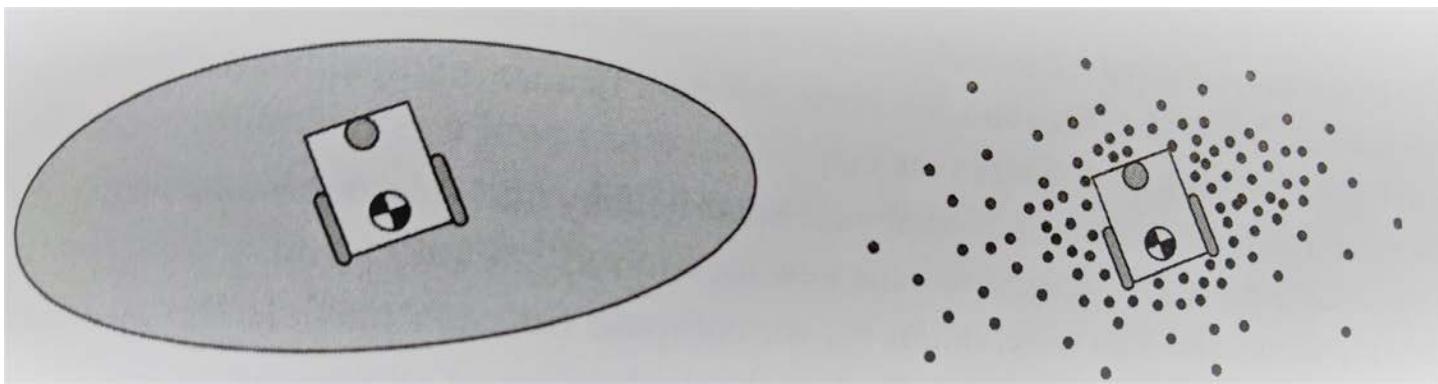
- Robot updates its position
 - Resulting pose estimate becomes correlated with the feature location estimates
- Robot's uncertainty shrinks
 - So does uncertainty in the rest of the map



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

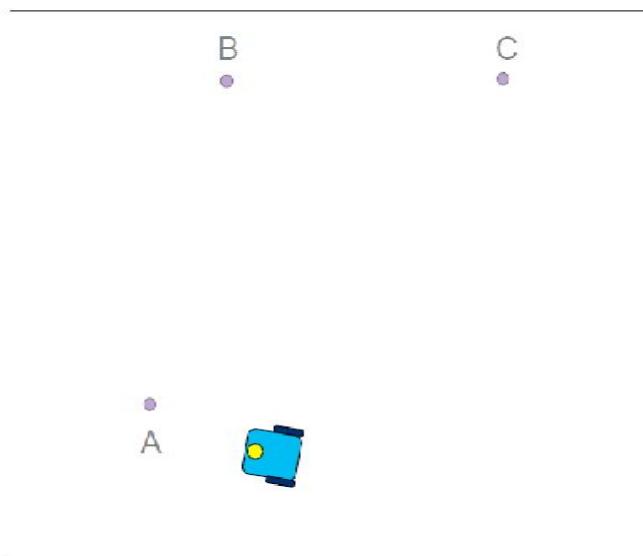
Use internal representations for:

- The positions of landmarks
- The camera parameters

Assumption:

- Robot's uncertainty at starting point is zero

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

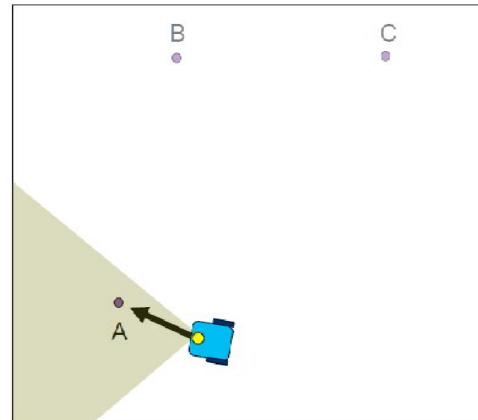


Start: robot has zero uncertainty

SLAM with particle filter

On every frame:

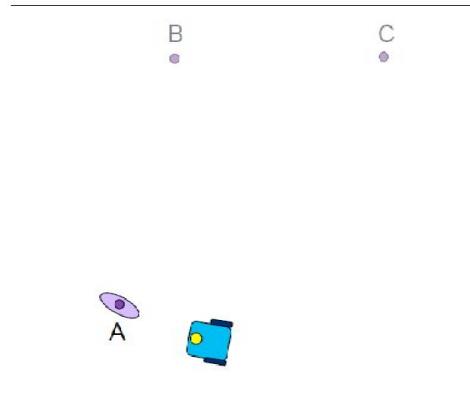
- **Predict** how the robot has moved
- **Measure** the world through sensors
- **Update** the internal representation



First measurement of feature A

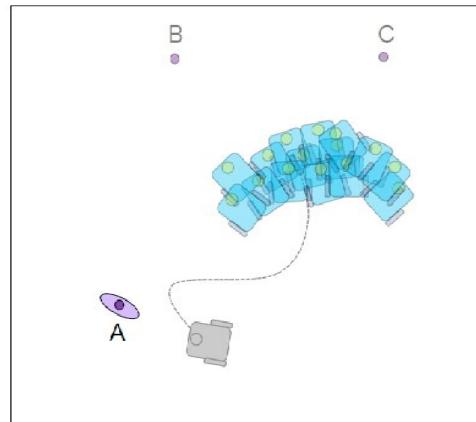
SLAM with particle filter

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



SLAM with particle filter

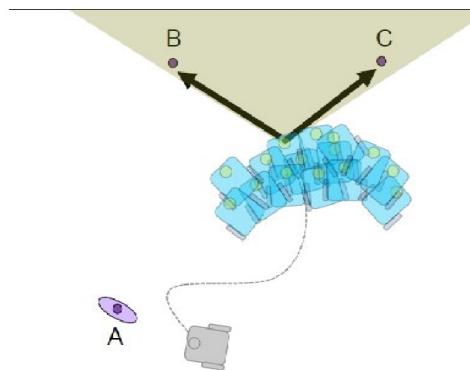
- As the robot moves, pose uncertainty increases
- Apply motion model to each particle



Robot moves forwards: uncertainty grows

SLAM with particle filter

- Robot observes two new features
 - B and C

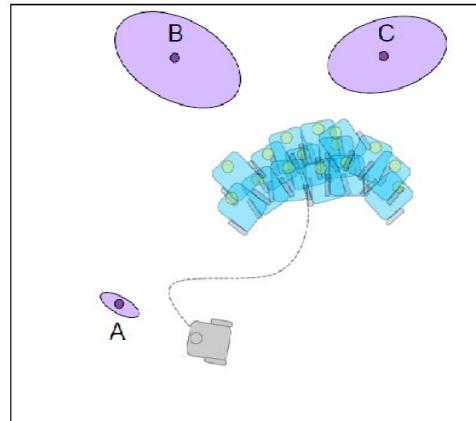


Robot makes first measurements of B & C

SLAM with particle filter

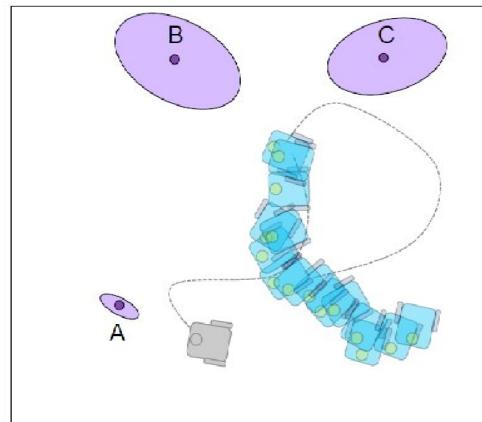
Position uncertainty encoded for each particle individually:

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



SLAM with particle filter

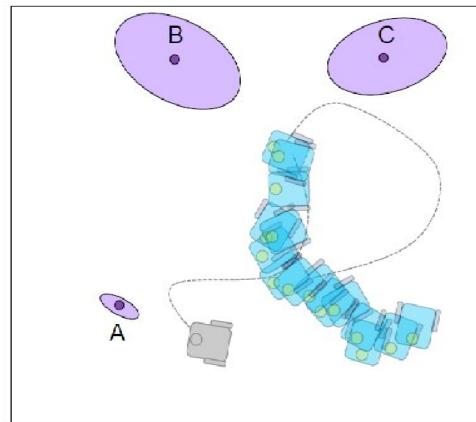
- Robot moves again and its uncertainty increases
- Apply motion model to each particle



Robot moves again: uncertainty grows more

SLAM with particle filter

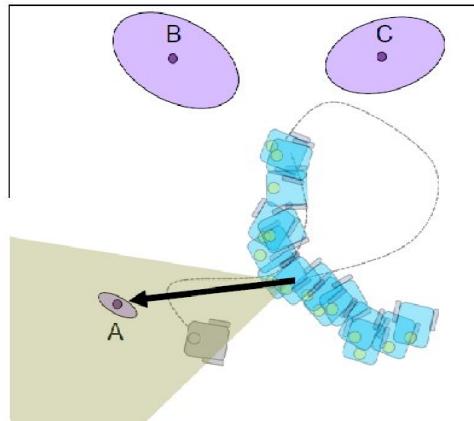
- Robot moves again and its uncertainty increases
- Apply motion model to each particle



Robot moves again: uncertainty grows more

SLAM with particle filter

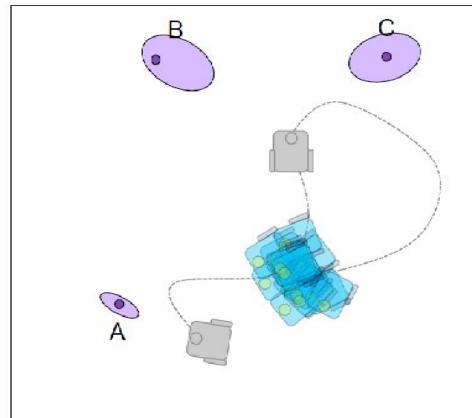
- Robot re-observes an old feature (A)
- **Loop closure** detection



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

SLAM with particle filter

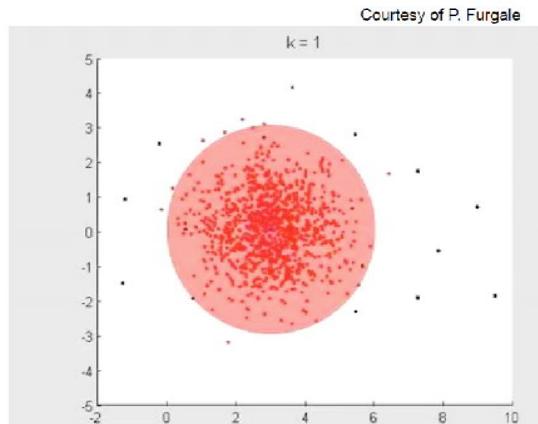
- Represents belief with a series of samples
- Each particle denotes a hypothesis of the state with an associated weight
- Predict/measure/update

Pros:

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

Cons:

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



Distribution in the robot's position estimate:

- red dots – particle filtering
- red ellipse – EKF filtering

Open source SLAM software

- <http://www.openslam.org>
 - Comprehensive list of SLAM software
- <http://wiki.ros.org/gmapping>
 - ROS wrapper for OpenSLAM's GMapping

Motion planning

ROS Navigation stack

<http://wiki.ros.org/navigation> - taking information from odometry, sensors, and a goal pose, safely issues velocity commands.

http://wiki.ros.org/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.

<http://wiki.ros.org/gmapping> - 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.

<http://wiki.ros.org/amcl> - implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.

http://wiki.ros.org/global_planner - provides an implementation of a fast, interpolated global planner for navigation. Dijkstra/A*

http://wiki.ros.org/base_local_planner - provides implementations of the Trajectory Rollout and Dynamic Window approaches to local robot navigation on a plane.

http://wiki.ros.org/move_base - links together a global and local planner to accomplish its global navigation task.

Localization and autonomous planning with TIAGo

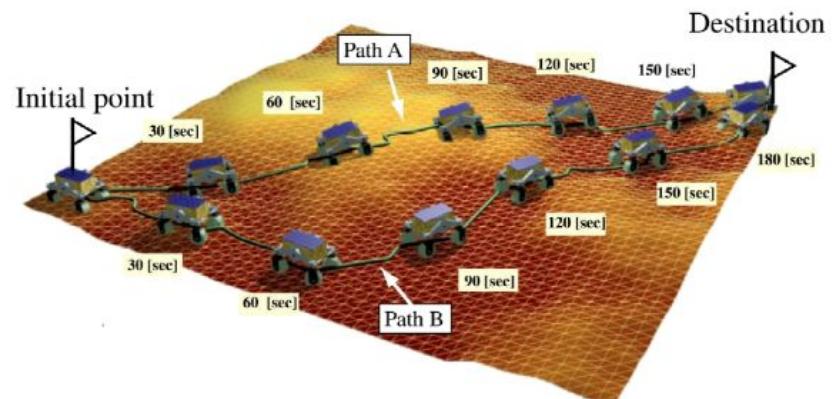
Robot motion planning

- Representing planning problems
 - Configuration spaces
- 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?

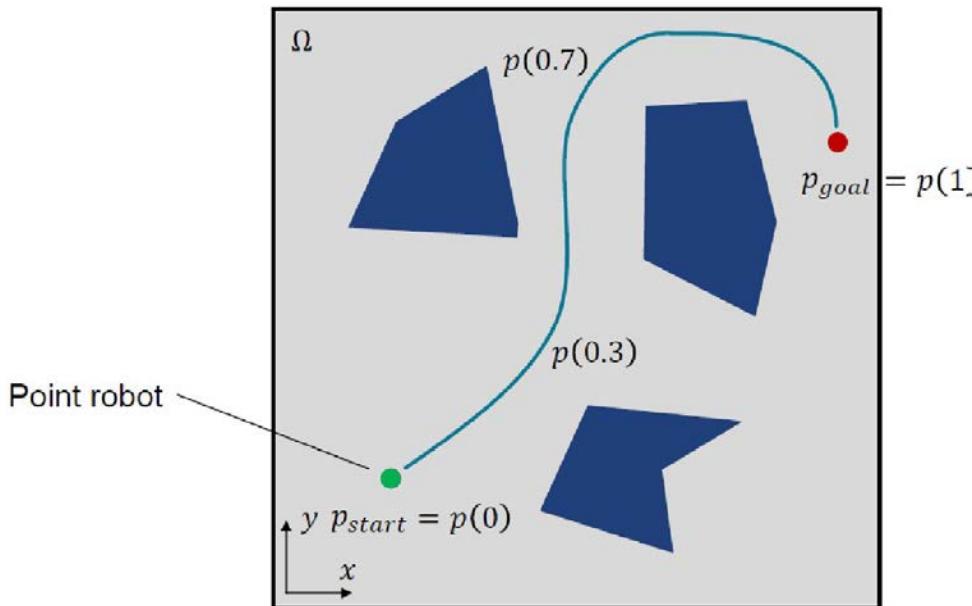
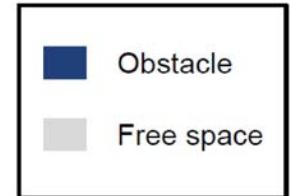
Assumptions:

- 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

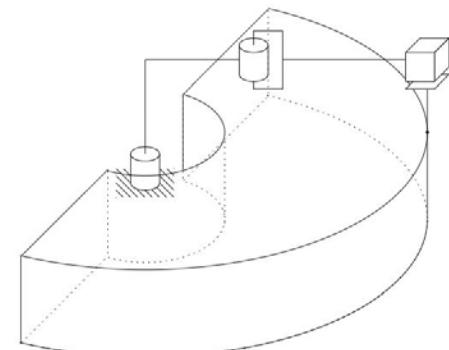
- **Workspace**

- Reachable space within the environment



- **Configuration space:**

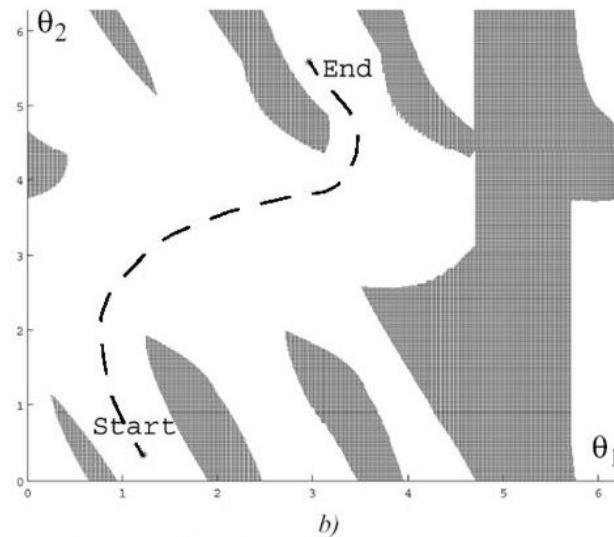
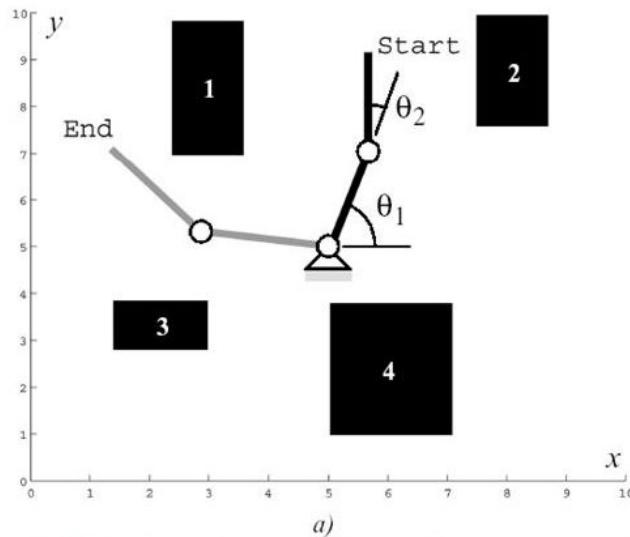
- Full state of the robot in the environment



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

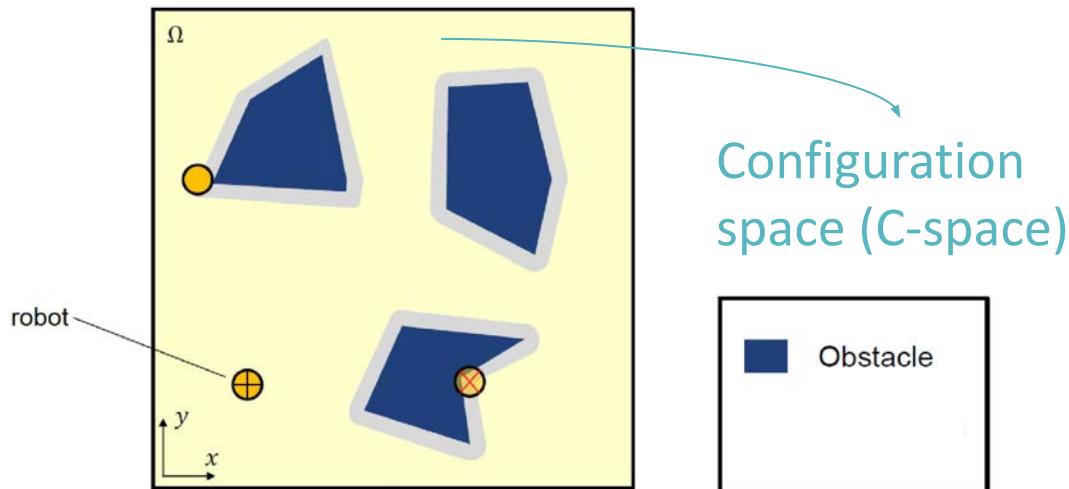


Path planning can be easier in configuration space

Configuration (C-)Space

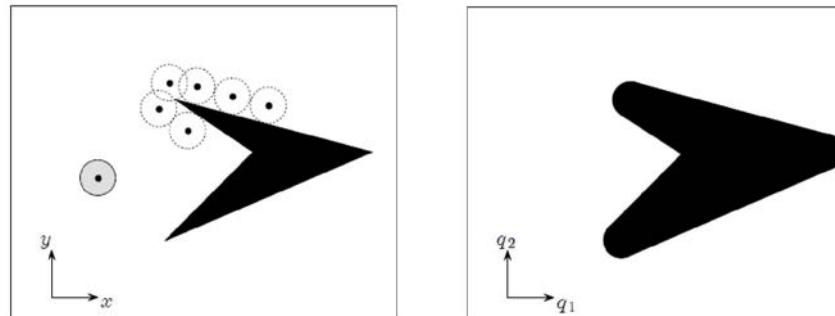
- For holonomic mobile robots, the configuration space is just the pose: (x, y, θ)
- We often assume the robot is holonomic
- Not a bad assumption for differential drive robots

Configuration (C-)Space



Configuration (C-)Space

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

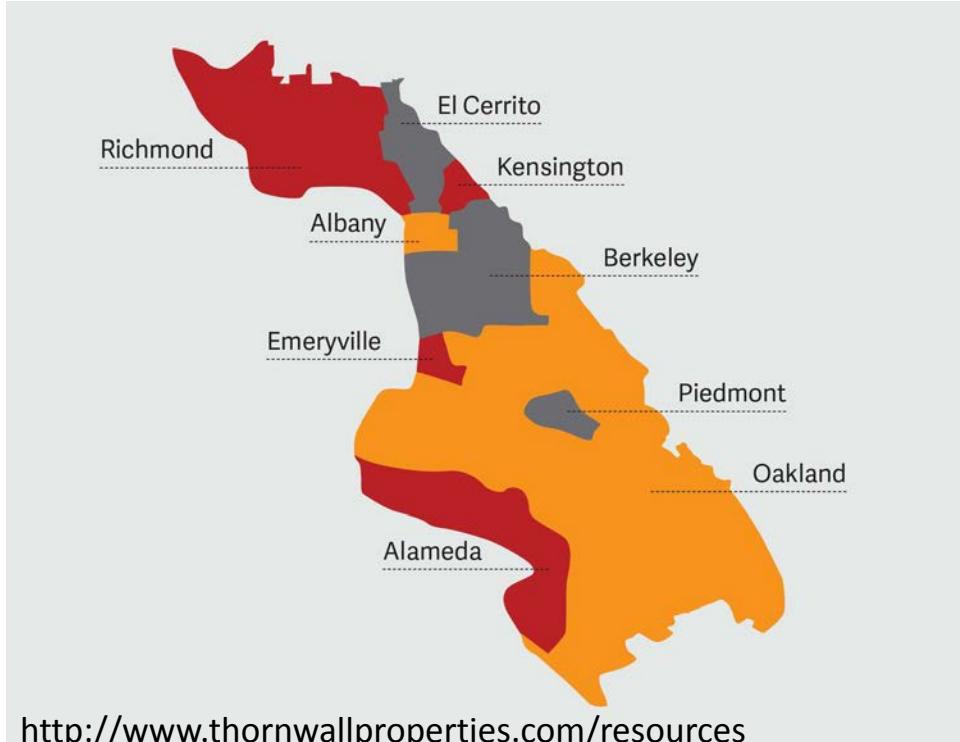


Continuous vs discrete state space representations

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

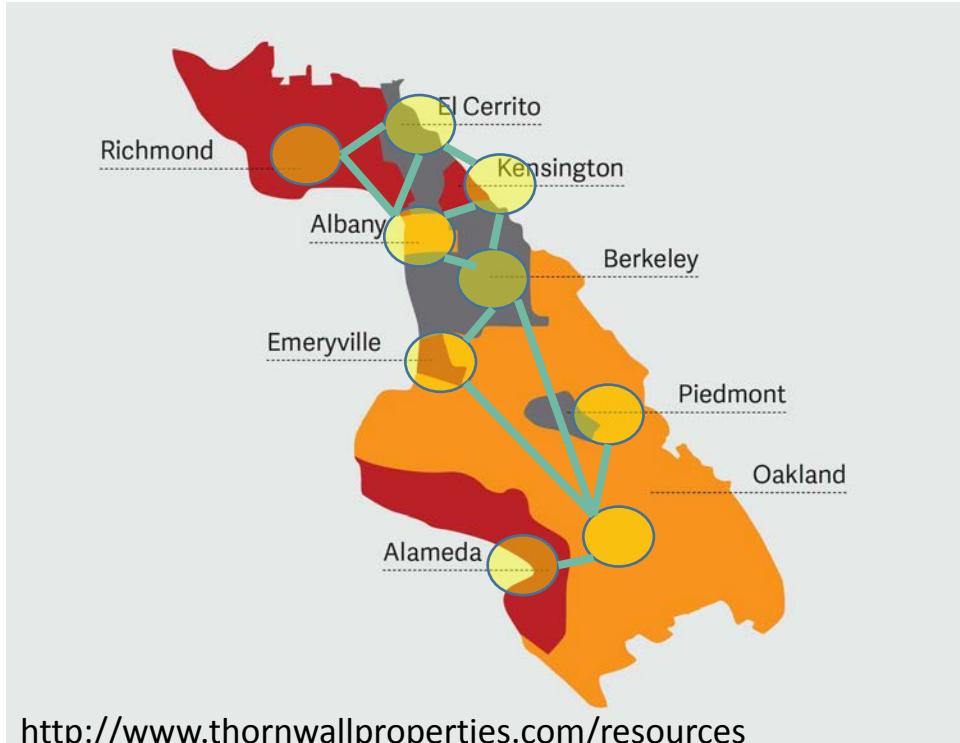
Example Graph

Neighborhood in the East Bay



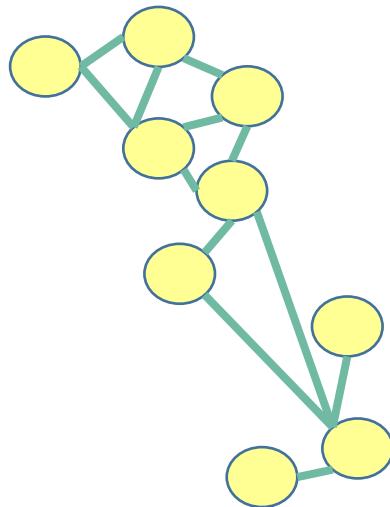
Example Graph

Neighborhood in the East Bay

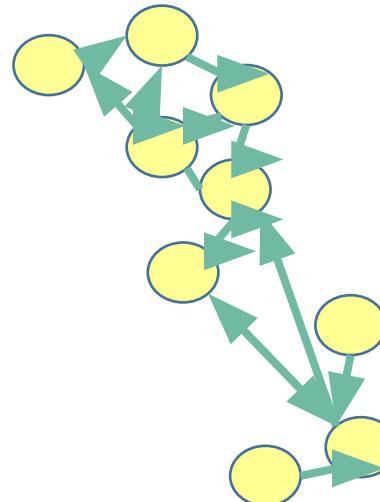


Types of graphs

Undirected Graph

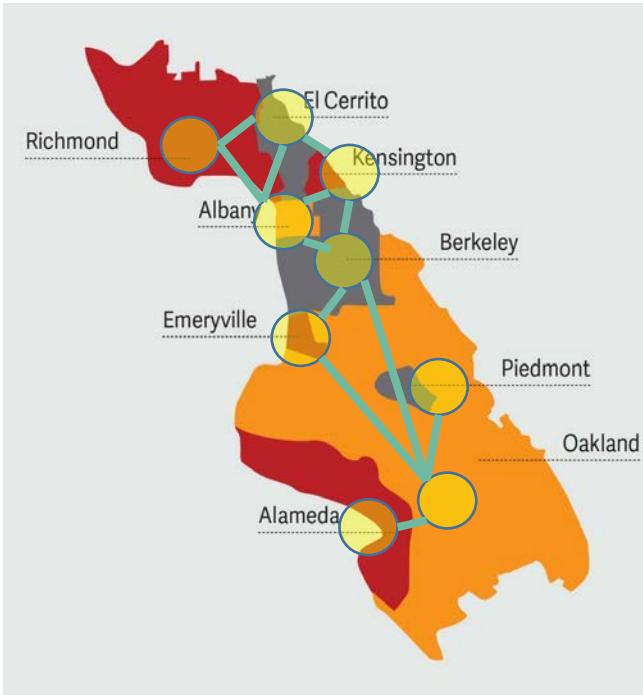


Directed Graph (Digraph)



Types of graphs

Unweighted Graph

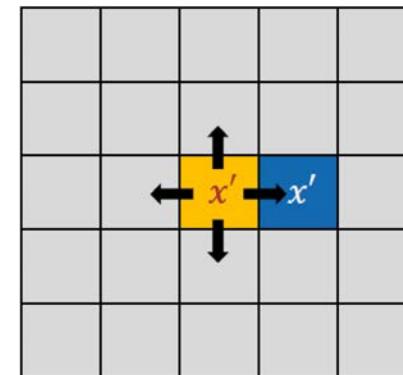


Weighted Graph



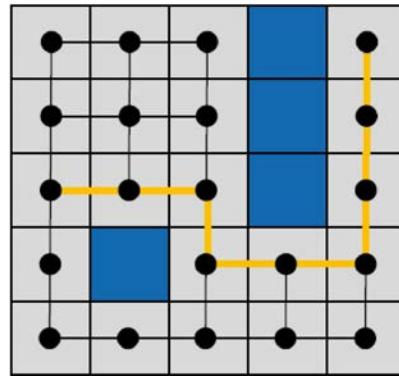
Discrete state space representation

- Reduce continuous state space to a finite set of discrete states (discrete state space representation)
 - $x \in X$
- Define feasible actions from each state
 - $A(x) = \{a0, a1, \dots, an\}$
- And an associated transition function
 - $f(x, a) = x'$



Grid to graph

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



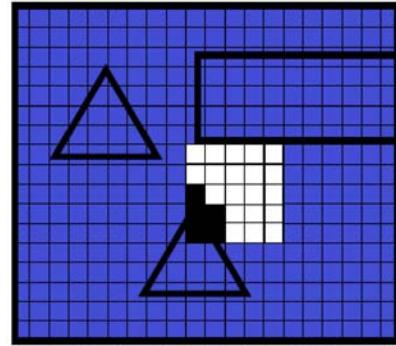
Turning a polygonal C-space into a grid

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

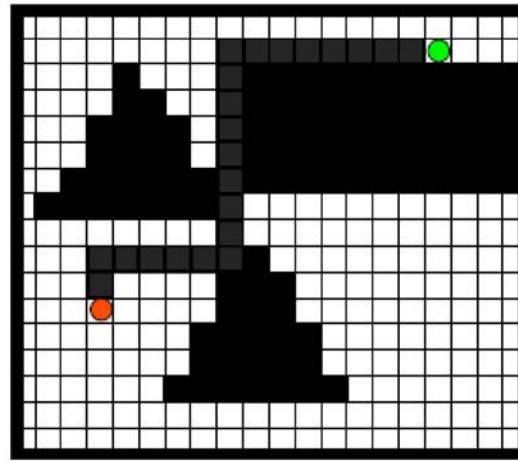
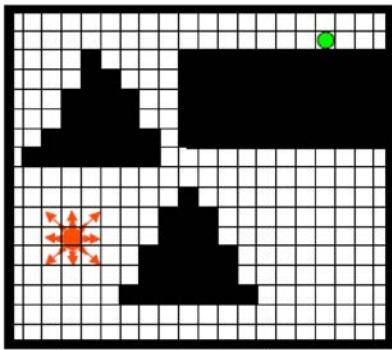
- Not inside an obstacle
- Further than the radius of the robot from all obstacle edges

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”

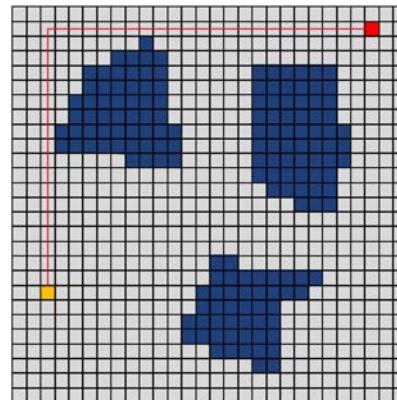


Perform search



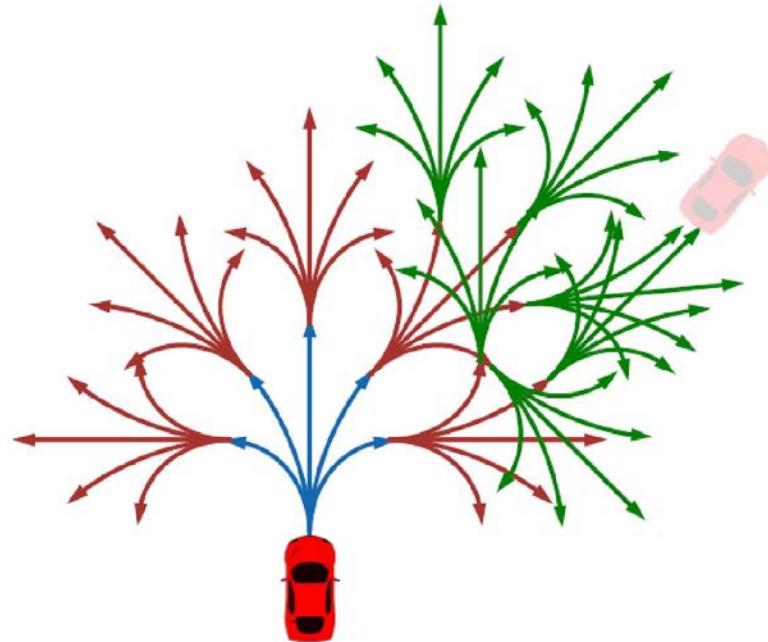
Issues with grid-based representations

- 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

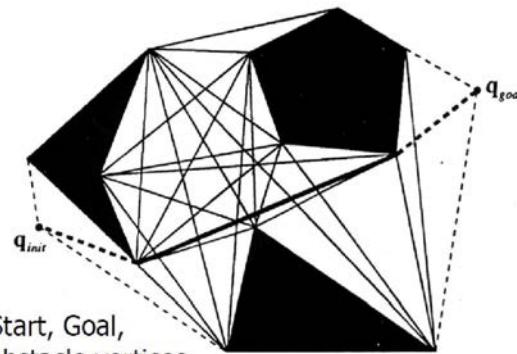
Create edges between all pairs of mutually visible vertices,
and search resulting graph

Pros:

- Optimal plan
- Good in sparse environments

Cons:

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



Vertices: Start, Goal,
obstacle vertices

Edges: all combinations (v_i, v_j) that do not intersect any obstacle

Graph construction: Voronoi diagram

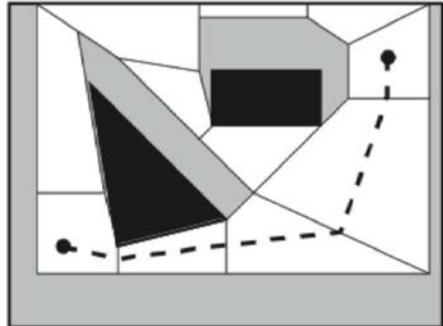
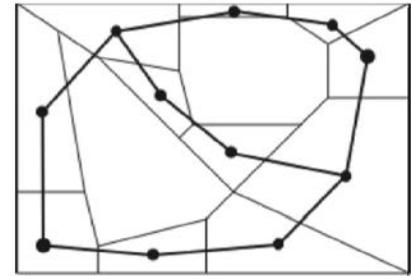
- Maximise the distance between the robot and the obstacles
- Draw equidistant lines
- Search resulting graph

Pros:

- Complete
- Executability

Cons:

- Not optimal
- Need long-range sensing



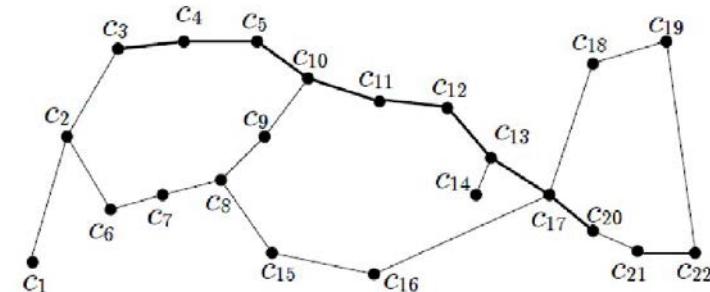
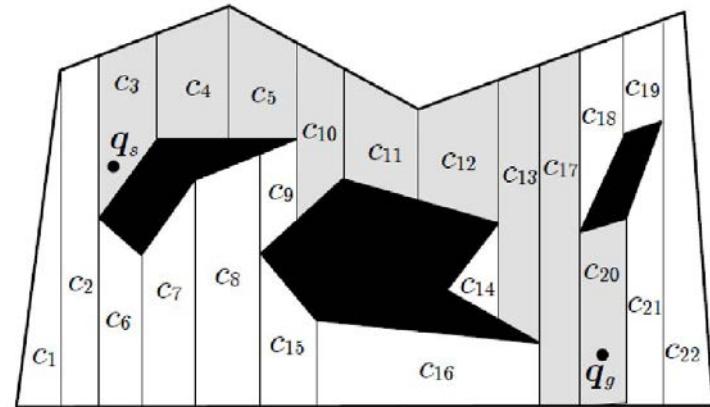
Graph construction: Exact cell decomposition

Pros:

- Complete

Cons:

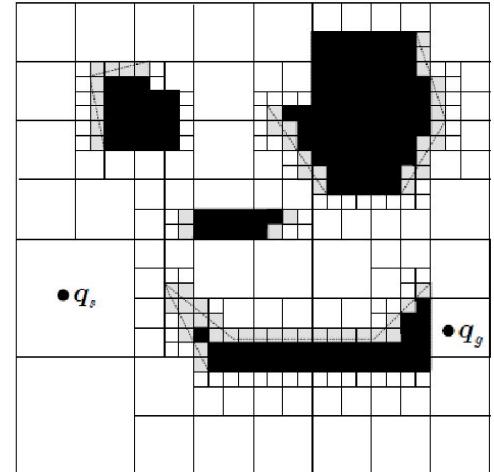
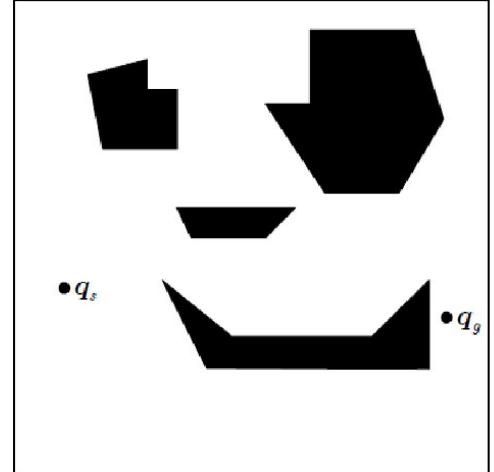
- Only good in extremely sparse environments



Graph construction:

Approximate cell decomposition

- Recursively decompose area into smaller rectangles
- Low computational complexity

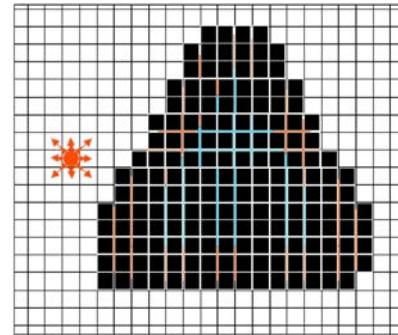


Planning as search

- 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

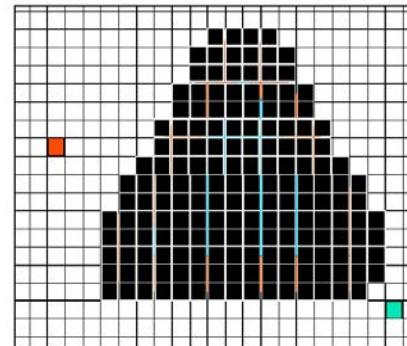
Setting up the state space

- Real space
- Configuration space
- State space

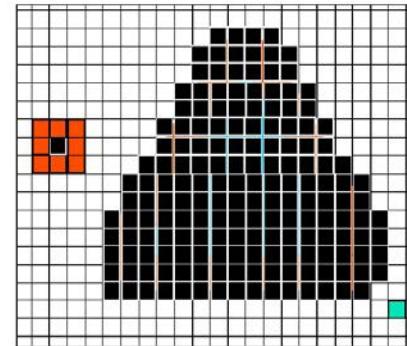
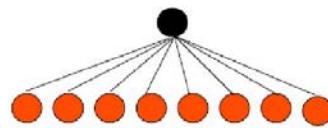


- Actions get you from one state to another
- Objective is to find a path from the start to the goal

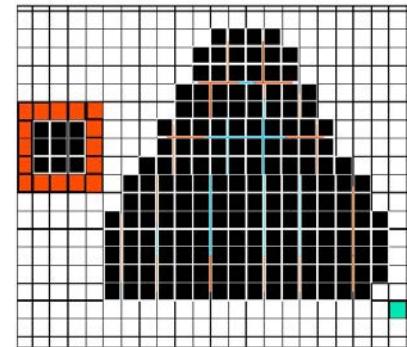
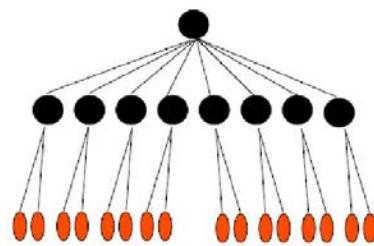
Tree search



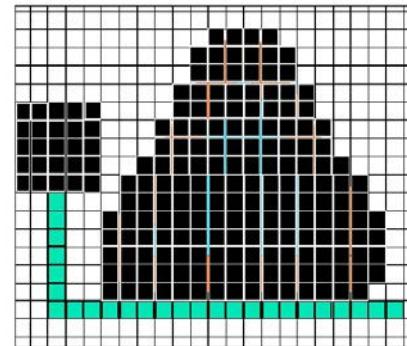
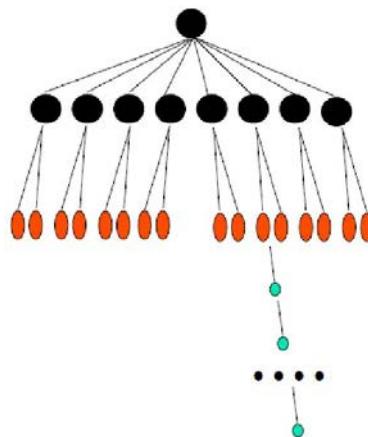
Tree search



Tree search

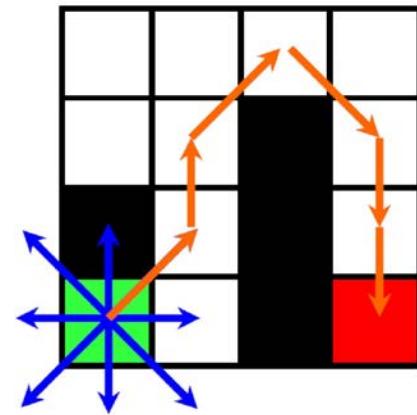


Tree search



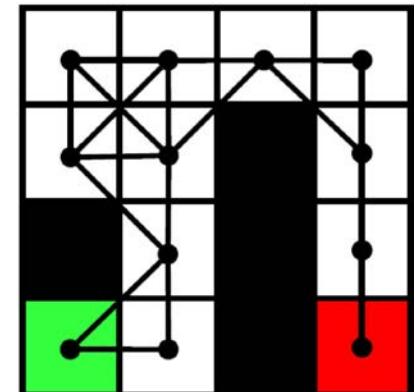
Setting up the state space

- 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



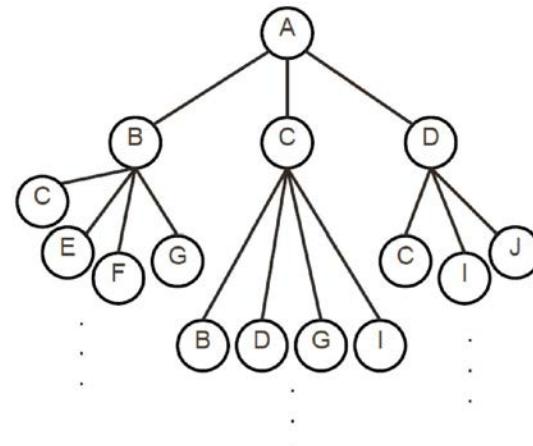
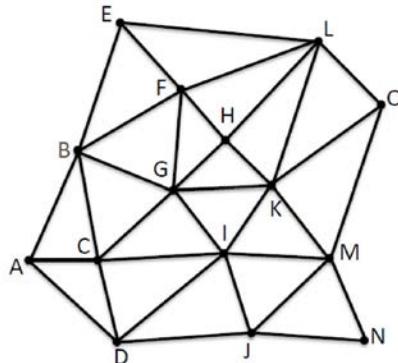
Setting up the state space

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



Search trees

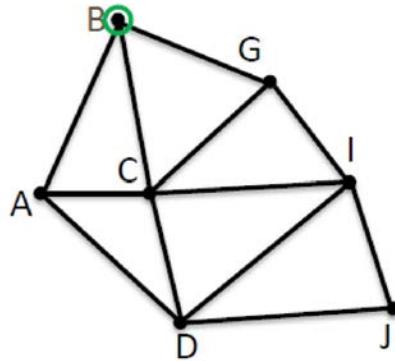
Construct a “tree” to search for optimal paths through the environment



Forward search – node expansion

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

Breadth-first search (BFS)



Open (Q):

{B}

Closed:

{}

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

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

Figure 2.4: A general template for forward search.

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

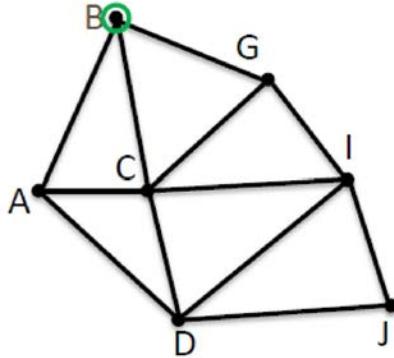
Breadth-first search (BFS)

- 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)



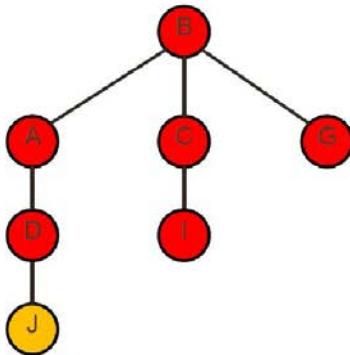
Open (Q): Closed:
{B} {}

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

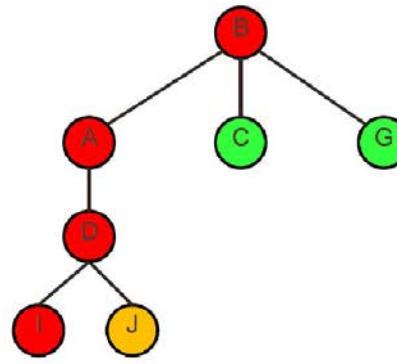
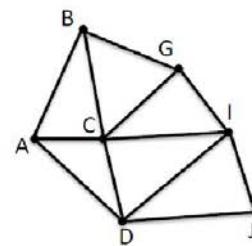
```
FORWARD_SEARCH
1    $Q.Insert(x_I)$  and mark  $x_I$  as visited
2   while  $Q$  not empty do
3        $x \leftarrow Q.GetFirst()$ 
4       if  $x \in X_G$ 
5           return SUCCESS
6       forall  $u \in U(x)$ 
7            $x' \leftarrow f(x, u)$ 
8           if  $x'$  not visited
9         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

Figure 2.4: A general template for forward search.

BFS vs DFS



BFS



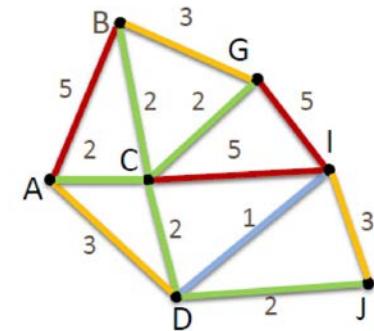
DFS

BFS vs DFS

- DFS not complete for infinite 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 $O(|V|+|E|)$

Dijkstra's shortest-path algorithm

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



Dijkstra's shortest-path algorithm

Open (Q):

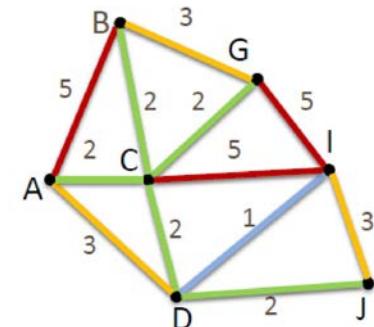
{B(0)}

Closed:

{B(0)}

Our Dijkstra queue will be ordered by cost to arrive:

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



Dijkstra's shortest-path algorithm

- 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
 - Can we incorporate our knowledge of the goal to expand nodes closer to the goal earlier?
 - Can we do it without breaking the condition that a node is only accepted with its lowest cost of arrival?

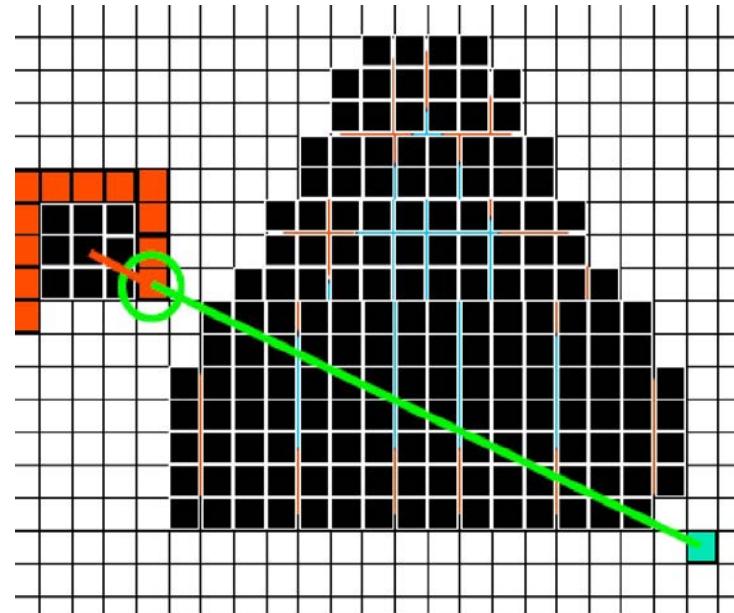
Informed Search – A*

Use domain knowledge to bias the search

Favour actions that might get closer to the goal

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

Choose the state with best f



Informed Search – A*

Use domain knowledge to bias the search

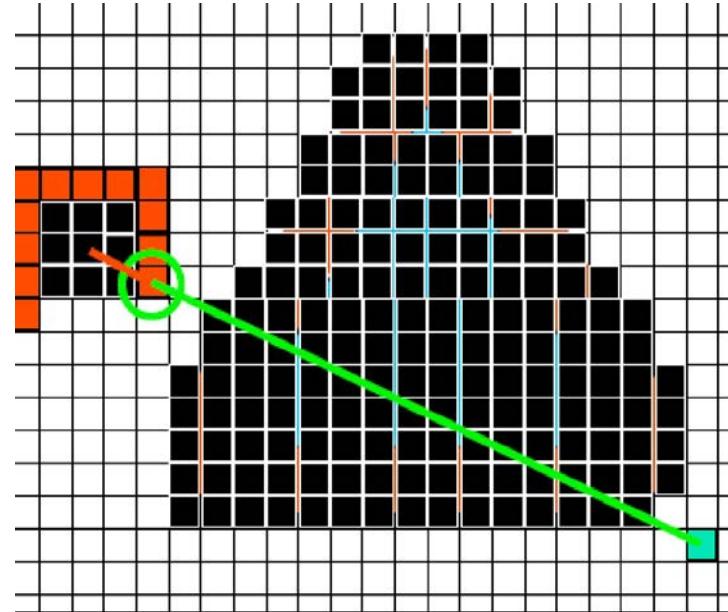
Favour actions that might get closer to the goal

Each state gets a value

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

Cost incurred so far, from the start state

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



Informed Search – A*

Use domain knowledge to bias the search

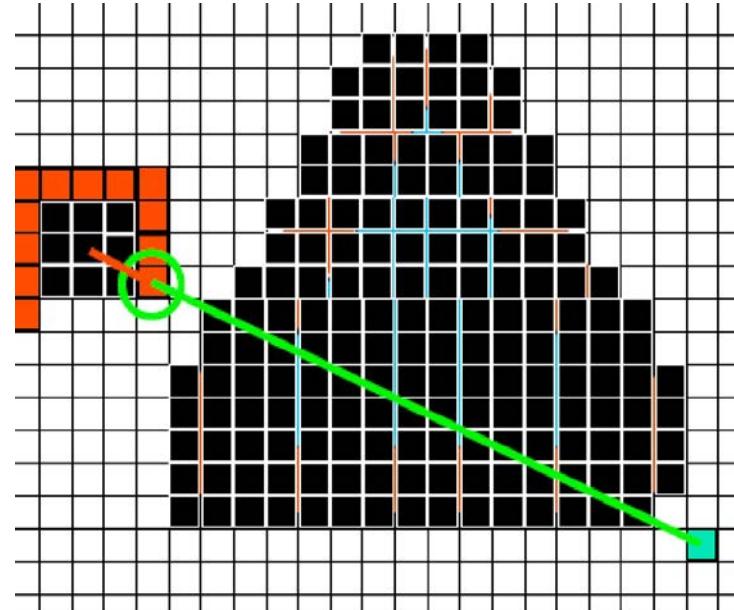
Favour actions that might get closer to the goal

Each state gets a value

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

Cost incurred so far, from the start state

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



Example:

$$g(x) = 3$$

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

$$f(x) = 22.7$$

Informed Search – A*

Use domain knowledge to bias the search

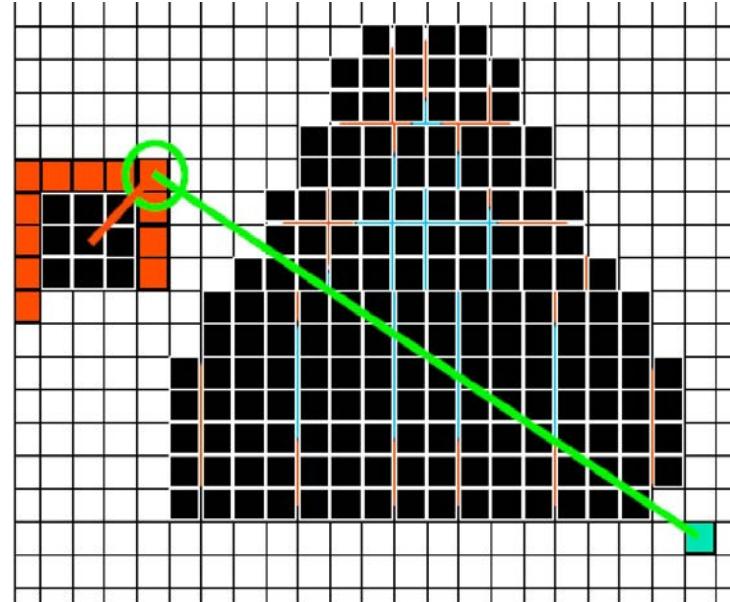
Favour actions that might get closer to the goal

Each state gets a value

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

Cost incurred so far, from the start state

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



Example:

$$g(x) = 4$$

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

$$f(x) = 25.1$$

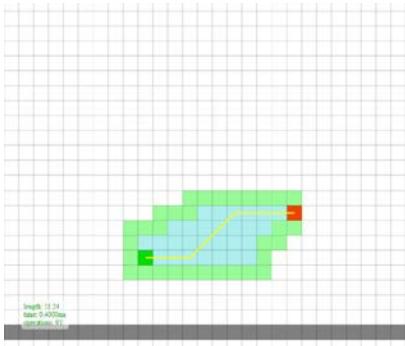
How to choose heuristics?

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

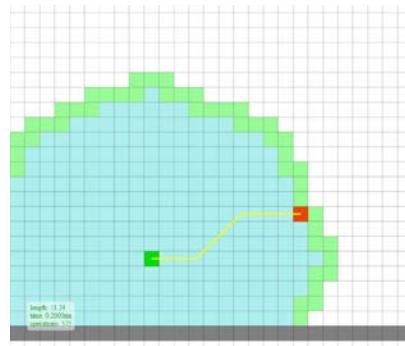
Decisions, decisions...

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

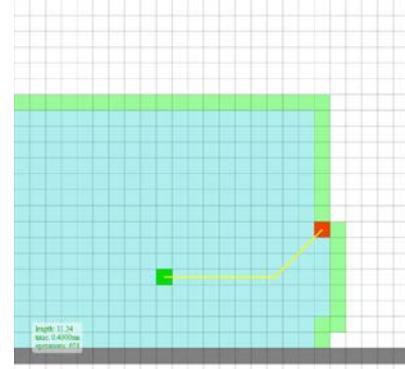
Comparison



A*



Dijkstra



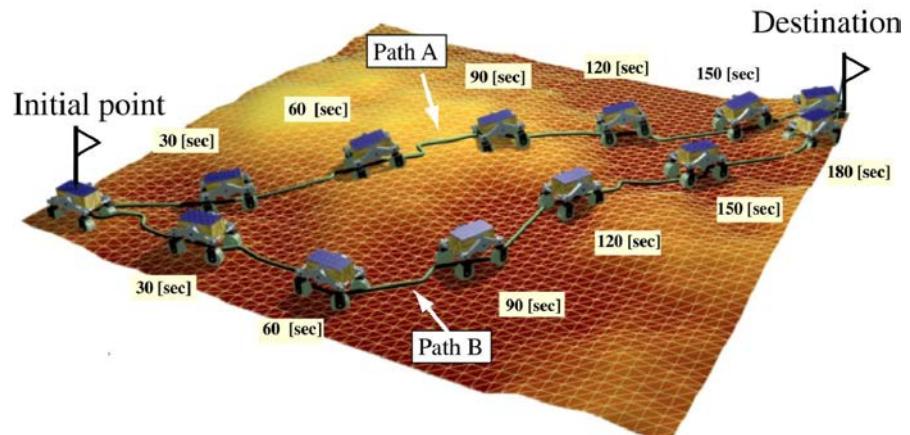
BFS

Randomised graph search

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

Number of nodes in a:

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



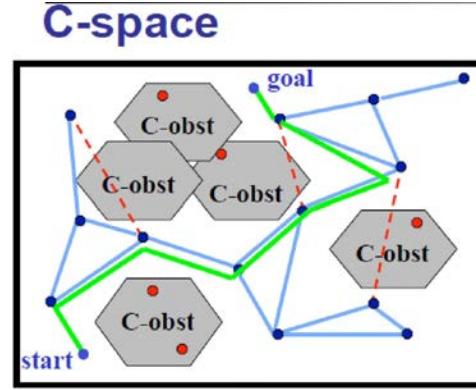
Randomised graph search

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

Probabilistic road maps (PRM)

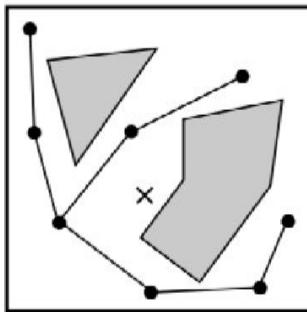
(Kavraki et al. 1996)

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

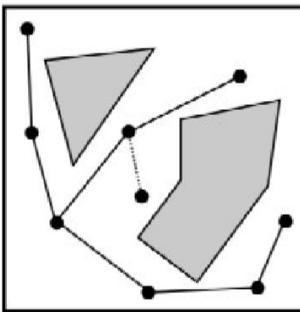


PRM algorithm

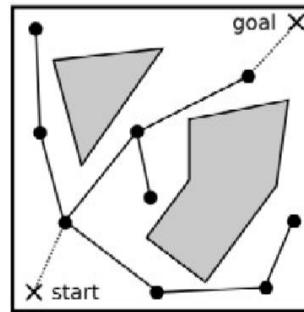
(1) PRM Algorithm



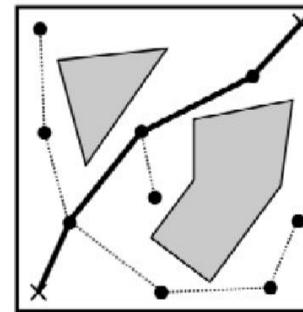
(a) The *learning* phase: a random sample, denoted by \times , is generated



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



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

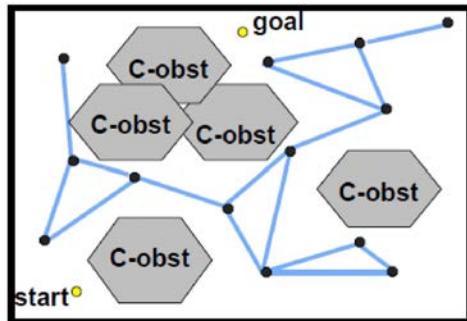


(d) A graph search algorithm is used to connect the start and goal through the roadmap.

PRMs

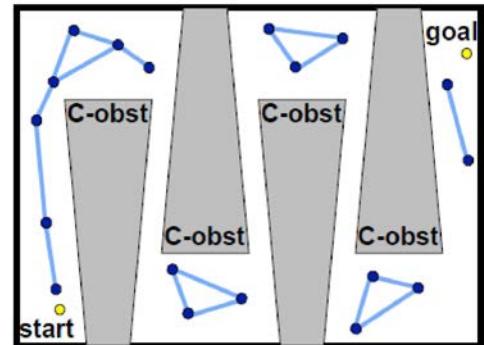
Pros

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



Cons

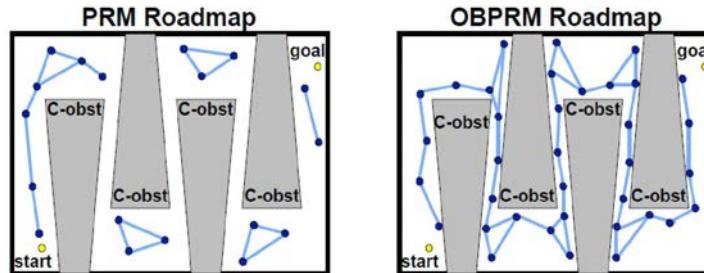
- Don't work as well for some problems:
 - Unlikely to sample nodes in *narrow passages*
 - Only *probabilistically complete*



Sampling around obstacles

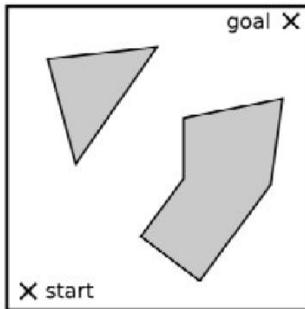
(Amato et al. 1998)

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

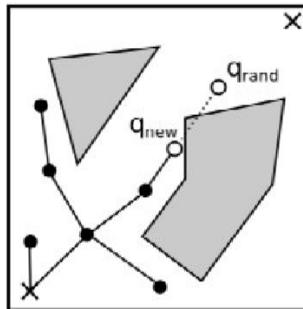


RRT algorithm

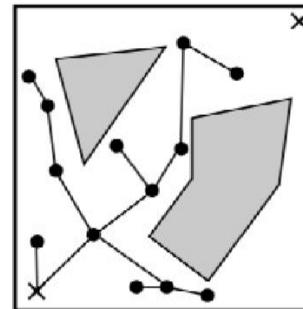
(2) RRT Algorithm



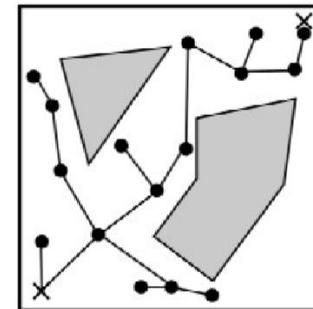
(a) A tree is grown from the start configuration towards the goal.



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

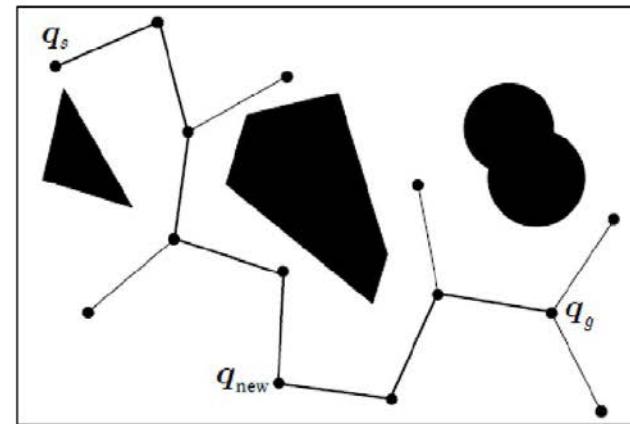
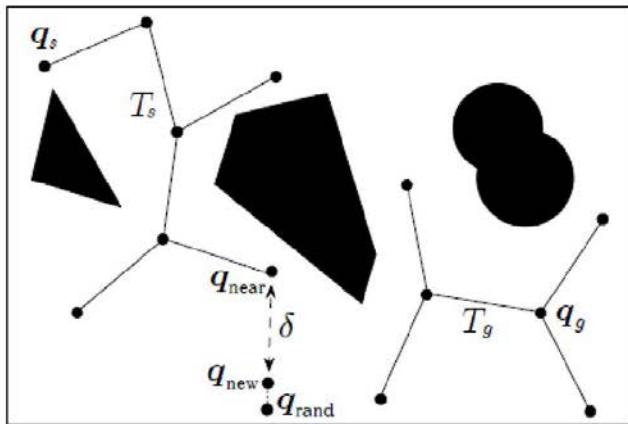


(c) The tree rapidly explores the free space.



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

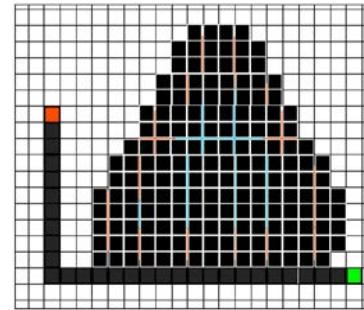
Bidirectional Rapidly-exploring Random Tree



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

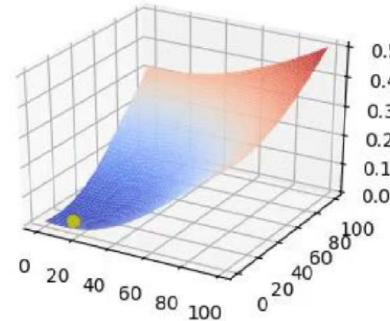
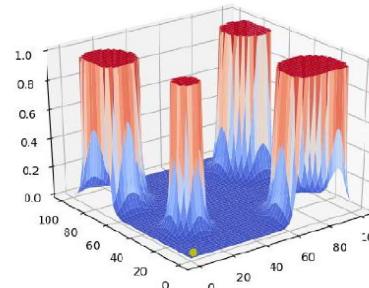
The potential of each obstacle generates a repulsive force

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

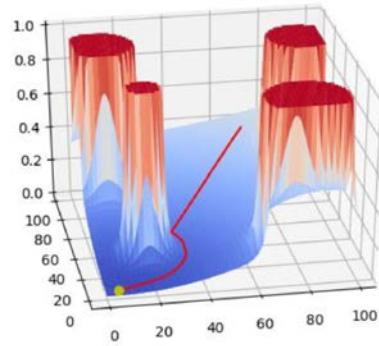
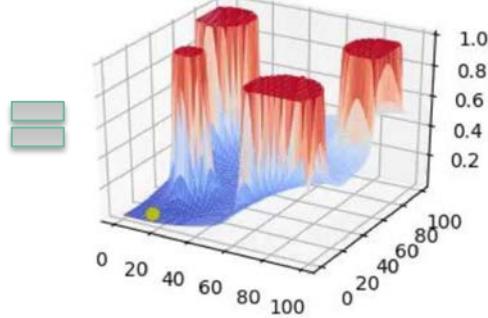
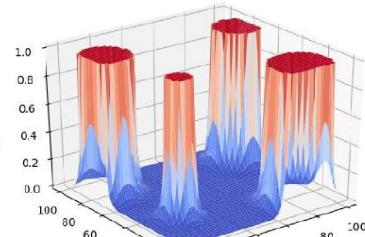
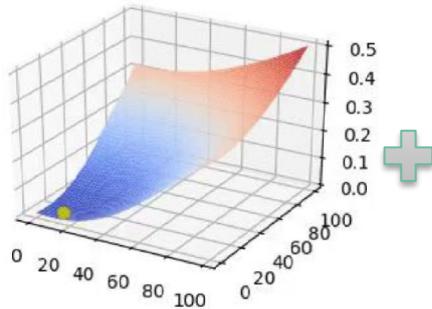
and the potential of the goal generates an attractive force

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

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**

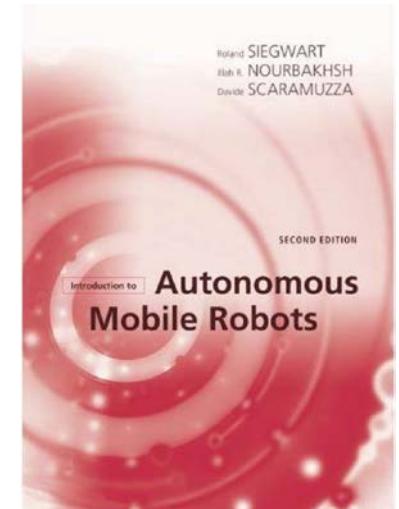
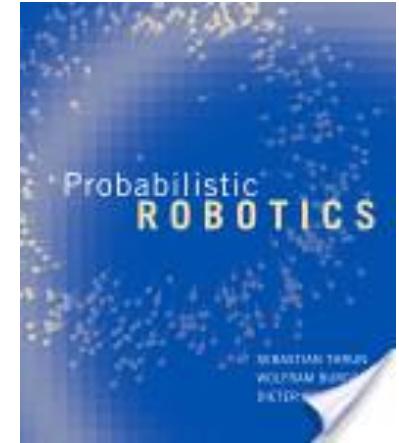
- Continuously run to adapt the planned global path to changes
- Avoids the need to compute the entire global path

- **Reactive**

- For collision avoidance in case of fast dynamic objects

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.



CMP3101M AMR – Week 9

Control Architectures

Dr. Athansios Polydoros

Based on the slides of Dr. Paul Baxter

Problems with robot control...

Complex Environment

Noisy / Unpredictable
Environment

Actions don't always lead
to intended consequences

Misleading sensors

Responses are too slow

Why Control Architectures?

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

Control Architecture Paradigms

- Robot control paradigm:

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

R. Murphy, 2000, p5

- Three main paradigms:

1. Deliberative
2. Reactive
3. Hybrid

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

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

Sensing, Planning, Acting

SENSE

Sense the Environment

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

PLAN

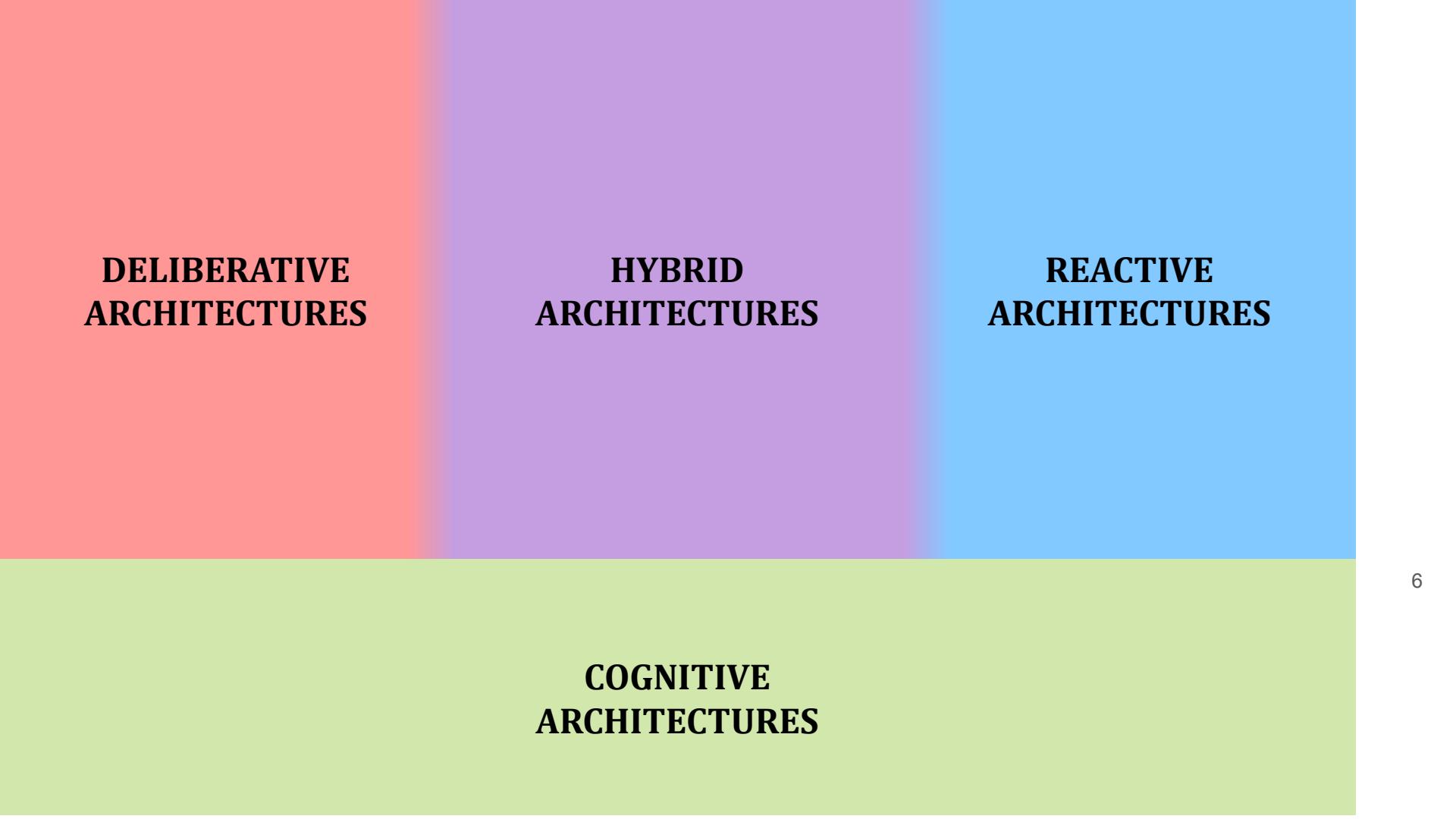
Decide what to do (using some model of the world)

- In: sensory information
- Out: directives

ACT

Act on the Environment

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



**DELIBERATIVE
ARCHITECTURES**

**HYBRID
ARCHITECTURES**

**REACTIVE
ARCHITECTURES**

**COGNITIVE
ARCHITECTURES**

DELIBERATIVE ARCHITECTURES

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

- Symbolic AI

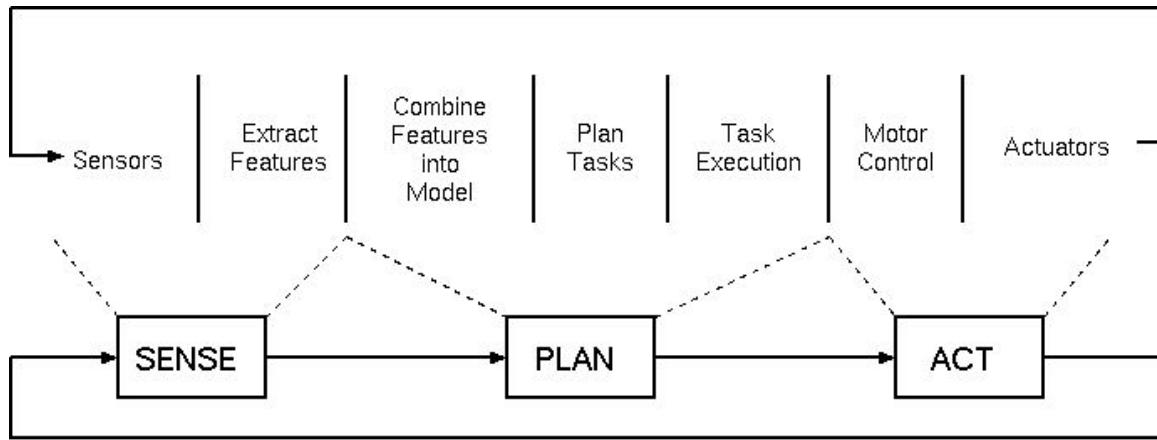
Emphasis on this ‘top-down’ (hierarchical) planning process

- Partly inspired by human introspection

Basic process:

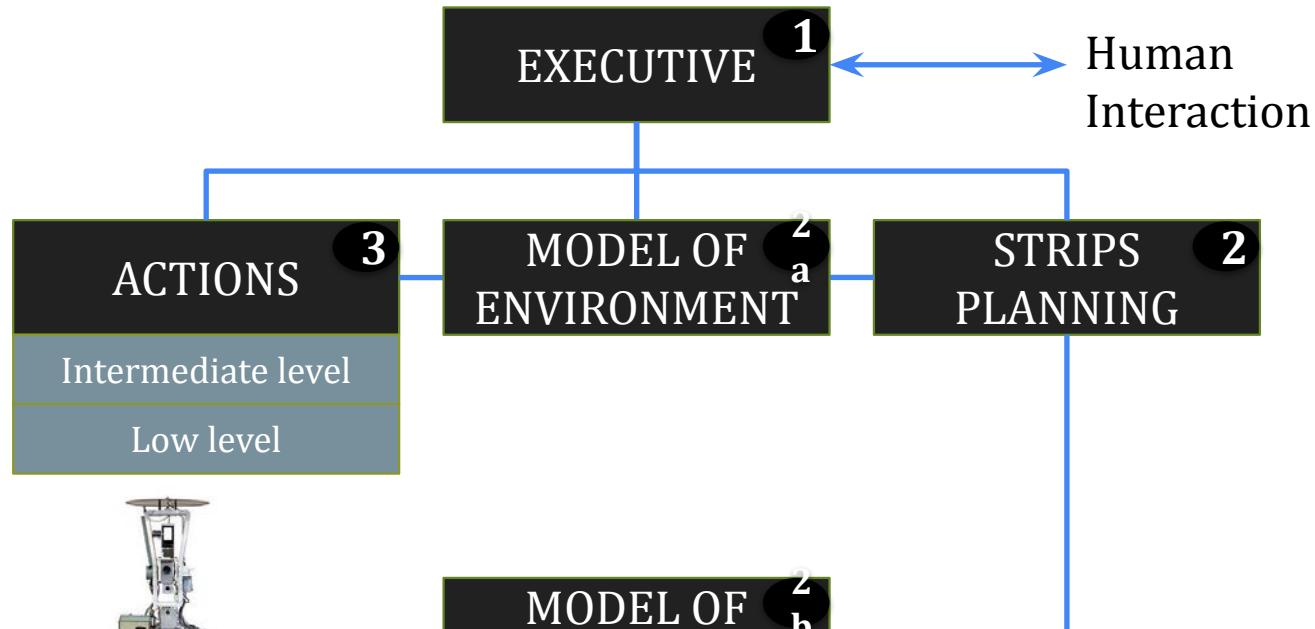
1. Gather currently available information, integrate into world model
2. Plan what to do
3. Execute the plan; return to step 1

Horizontal decomposition



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

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?)

Open ... / Close ...
Open door dx.
OPEN(dx)

Preconditions: NEXTTO(ROBOT, dx), TYPE(dx,DOOR), STATUS(dx,CLOSED)
Deletions: STATUS(dx,CLOSED)
Additions: *STATUS(dx,OPEN)

Close door dx.
CLOSE(dx)

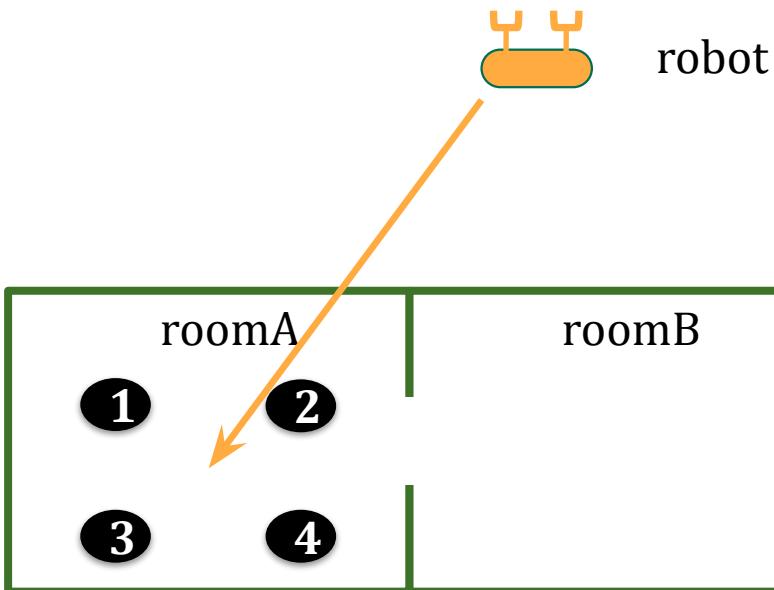
Preconditions: NEXTTO(ROBOT,dx), TYPE(dx,DOOR), STATUS(dx,OPEN)
Deletions: STATUS(dx,OPEN)
Additions: *STATUS(dx,CLOSED)

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-robbby rooma)
  (free left)
  (free right)
  (at ball4 rooma)
  (at ball3 rooma)
  (at ball2 rooma)
  (at ball1 rooma)
  (gripper left)
  (gripper right))
```



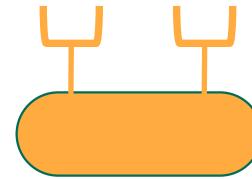
A brief PDDL example: Robot

```
define (domain gripper-strips)
:predicates (room ?r)
  (ball ?b)
  (gripper ?g)
  (at-roddy ?r)
  (at ?b ?r)
  (free ?g)
  (carry ?o ?g))

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

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



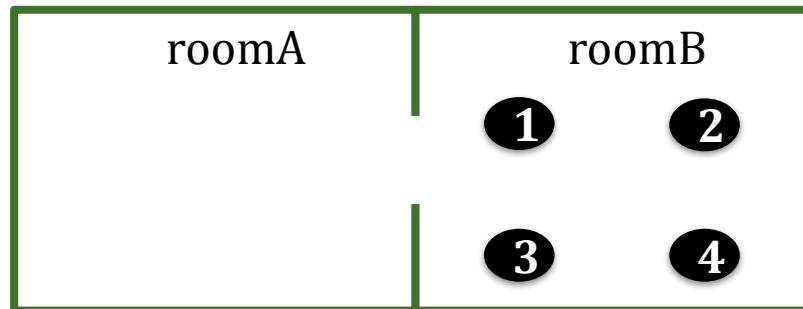
Robot can:

- Move
- Pick
- Drop

With left and
right grippers

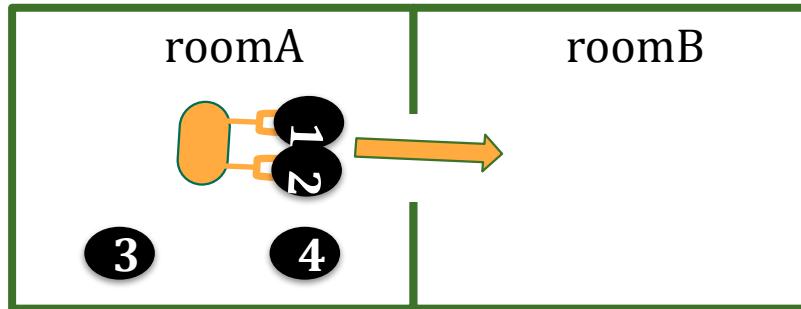
A brief PDDL example: Goal

```
(: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 affected by the agent (wrong model?)
- Uncertainty problem
 - How should this be handled in a symbolic planner that assumes crisp knowledge, and true/false conditionals?
- Computational load
 - High load leads to slow reactivity

Direct reaction against deliberative models

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

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

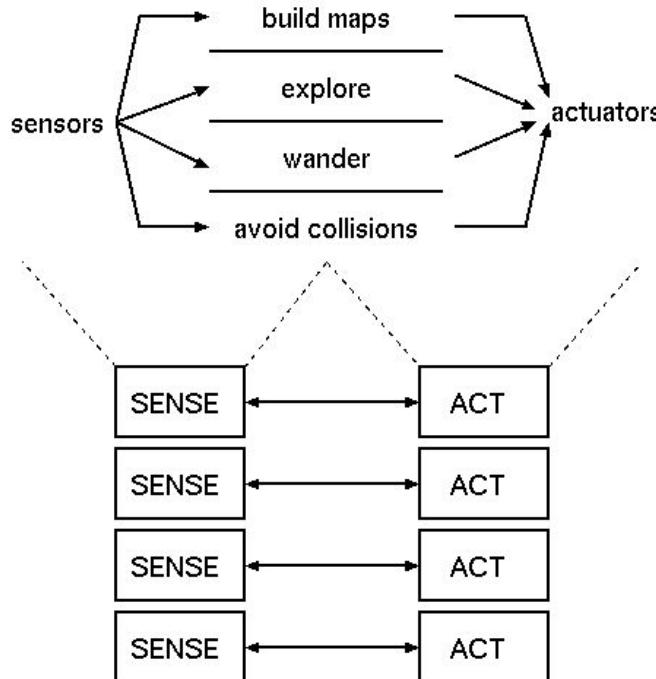
Basic process:

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

REACTIVE ARCHITECTURES



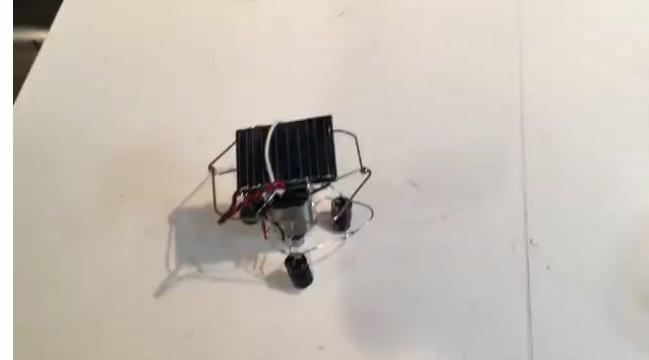
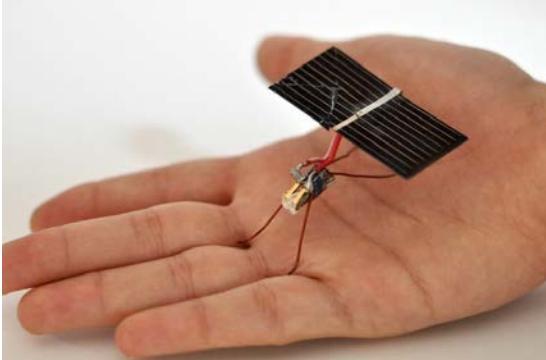
Vertical decomposition



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

Benefits of Reactive Architectures

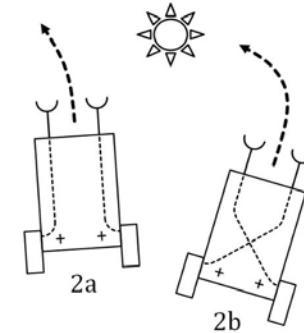
- No internal world model needed
 - “the world is its own best model”
 - Cheap and fast!
- Real-time behavioural control
- Can have emergence of complex behaviour with little design effort



<https://www.youtube.com/watch?v=k05V29ayVNU>

Behaviour-based Robotics

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



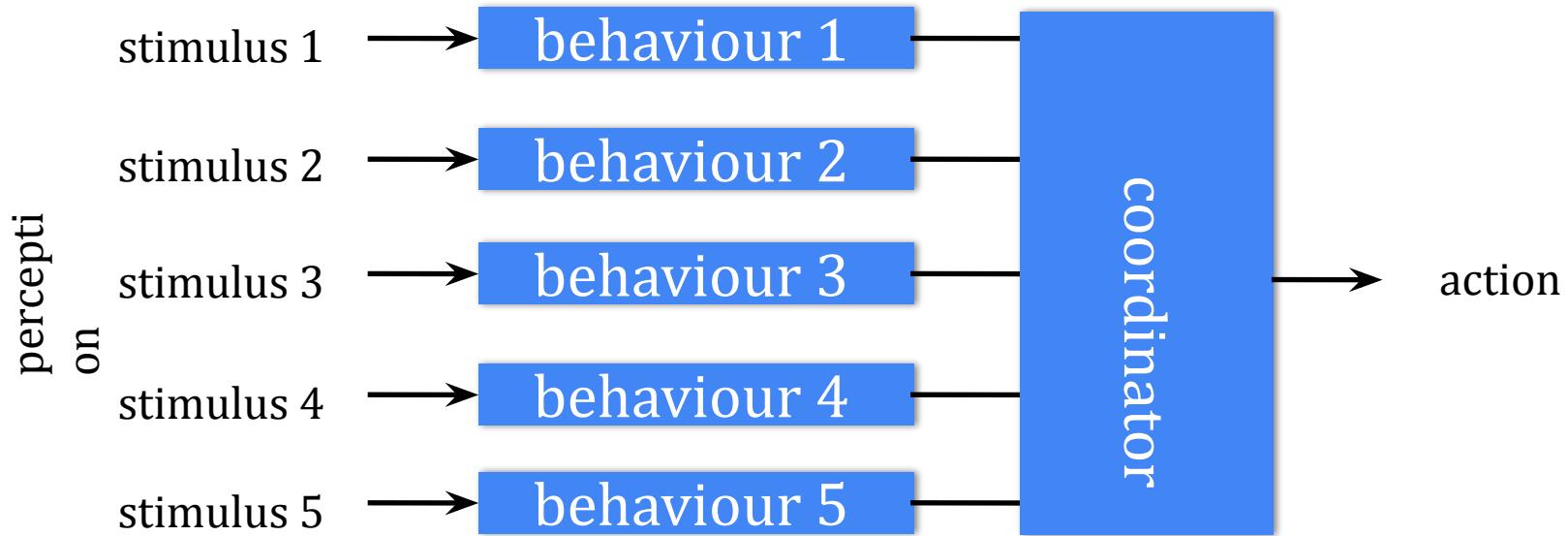
No World Model...

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



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

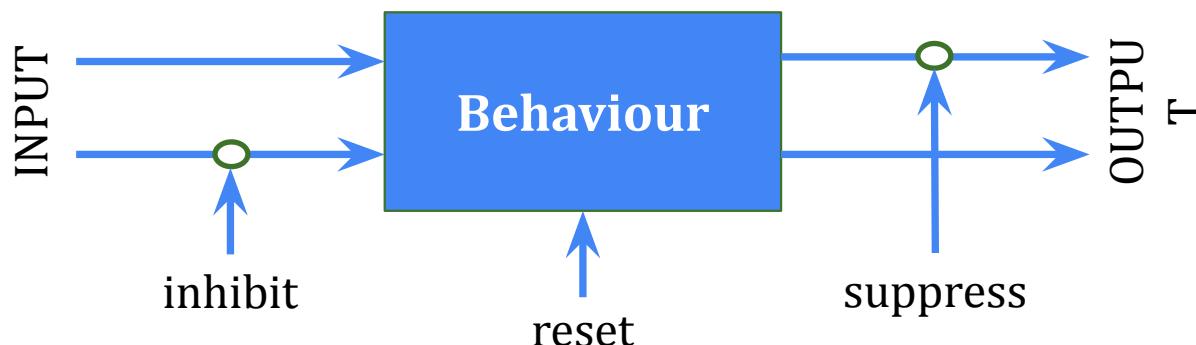
Multiple Behaviours (*recap*)



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

Subsumption Architecture

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



Example: Genghis

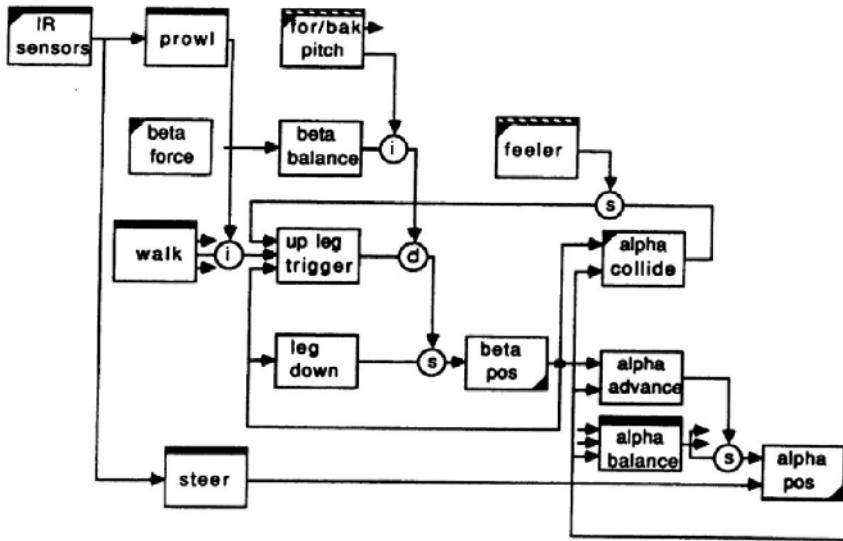
- Example architecture of Genghis

- Blocks:

- Sensor input
- Behaviours
- ...

- Note the relative complexity of the inter-connections

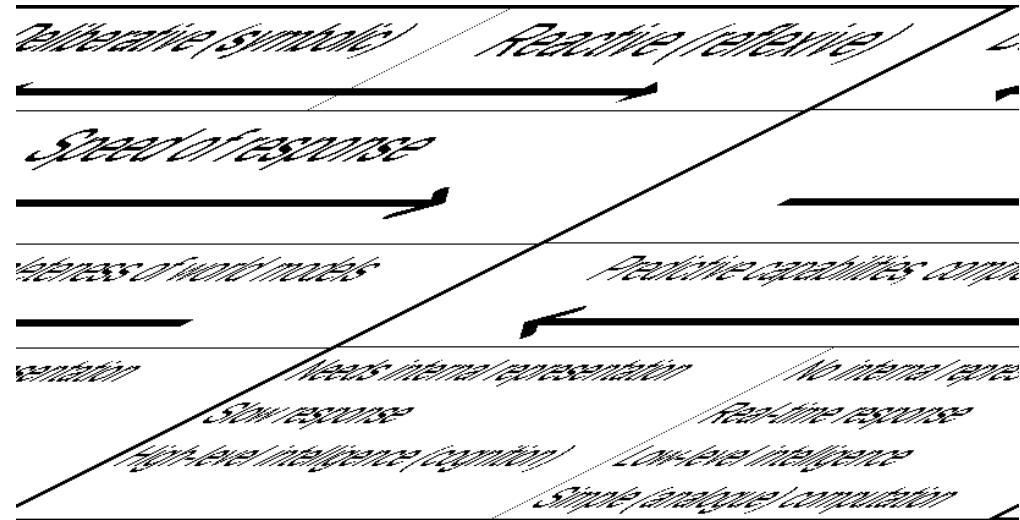
- Hand-designed and tuned behaviours



Reactive Architecture Limitations

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

Story so far...

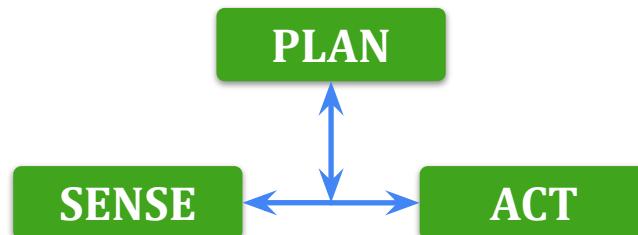


Trying to get the best of both Deliberative and Reactive paradigms

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

Multiple levels of control, each focused on a different aspect

HYBRID ARCHITECTURES



Linking Deliberative with Reactive

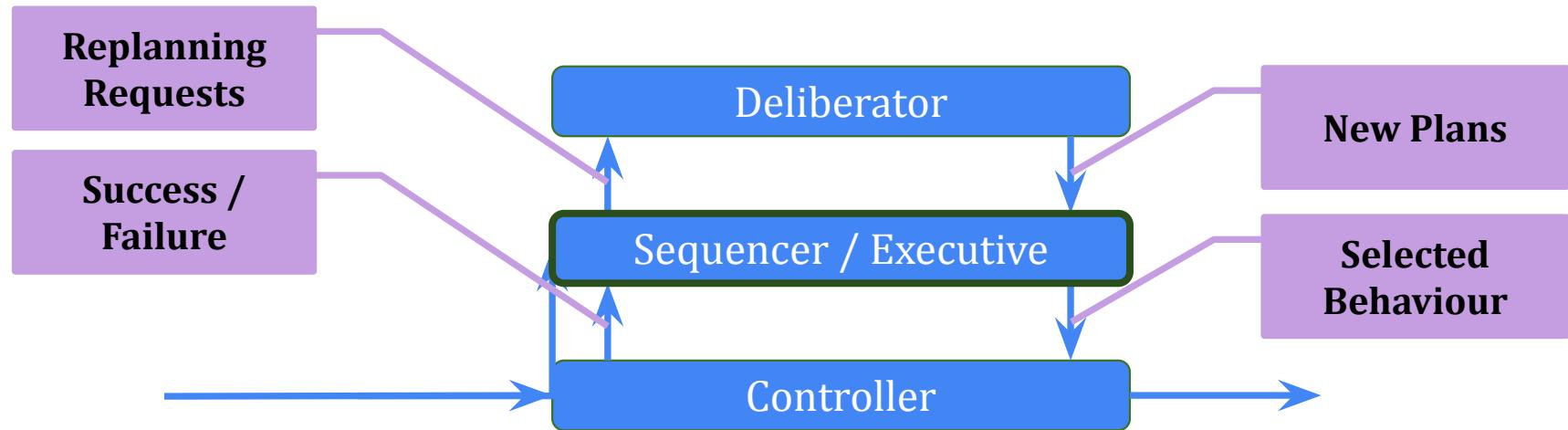


- To get best of both, use both...
 - Symbolic processing and world models/maps for planning
 - Reactive behaviours for fast, responsive action
- How best to link the two together?

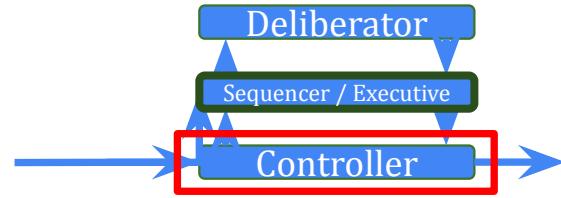
Temporal decomposition

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

Three Tier (3T) Architectures

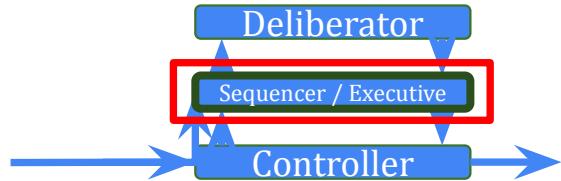


Controller



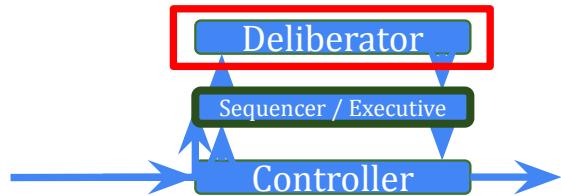
- Library of Behaviours
 - May be handcrafted
 - E.g. the behaviour-based approach
- Must be fast:
 - Avoiding internal state (except to estimate state), planning, etc
 - Stable closed-loop control
- Must be able to detect failure
 - This allows the Sequencer/Executive to call another behaviour, or call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

Sequencer / Executive



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

Deliberator



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

Performing a sample task

- I want coffee...
 - And I want it now...
- What do I need to take into account?

static ○ Where is it?

static ○ How to get there?

dynamic ○ Walk and Open doors... (physical conventions)

dynamic ○ Don't walk into people... (personal conventions) **Both planning and**

dynamic ○ Queue... (cultural conventions)

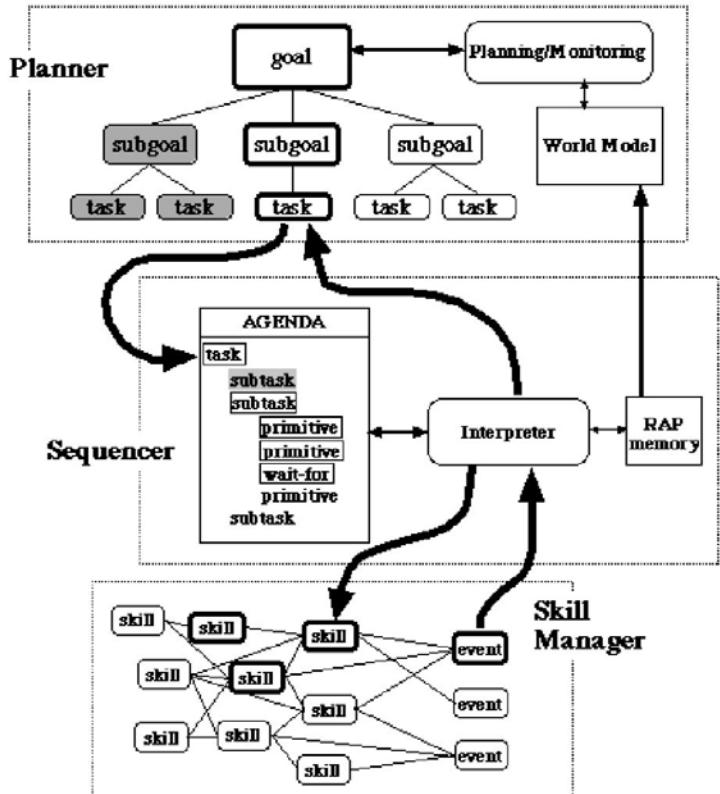
reactive components required to solve the task

dynamic ○ Pay... (legal conventions)

target! ○ Refreshment



Example Systems: NASA



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

Advantages

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

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

“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”

Hybrid Architecture Issues

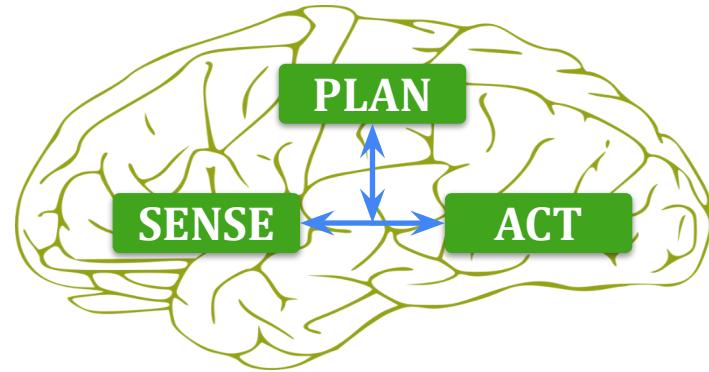
- The central role of the Sequencer / Executive
 - Clearly of central importance in the 3T approach
 - Thus needs particular attention
- “Symbol grounding”
 - Relevance of the representations (e.g. symbolic) to the instantiation/actions of the robot system it is planning for
 - Circumvented by appropriate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

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

- Not introspection, but evidence...

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

Overlaps with the previous paradigms, particularly hybrid



COGNITIVE ARCHITECTURES

What is a Cognitive Architecture?

Cognitive Architecture...

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

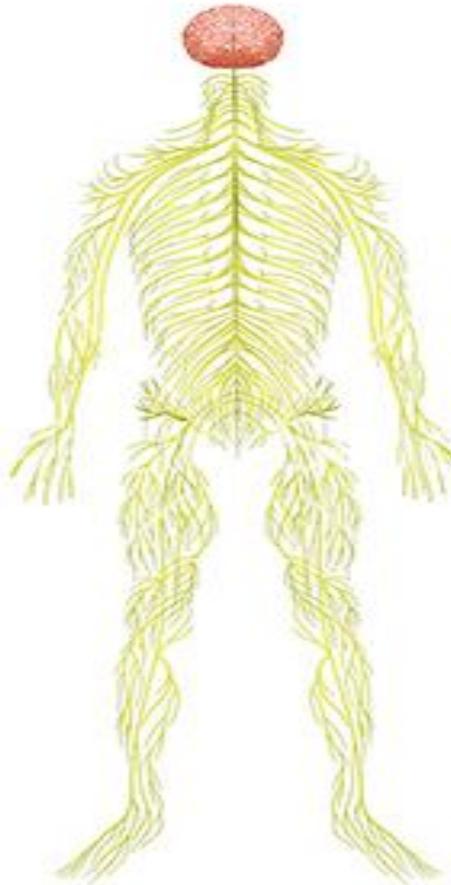
(Sun, 2004)

Unpacking the definition

- Cognition and Behaviour
 - Both an account of the internal workings, and also how this generates 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

Levels of control

- Brain
 - Cognitive processing
 - Memory, etc
- Joint between the two...
 - Embodied cognition (influence of the body)
 - Influence of drives (e.g. hunger)
 - Sensory information
- Body
 - Reflexes
 - Reactions
 - Independent of brain...



Why take inspiration from humans?

- Humans demonstrate the best example of highly complex intelligence, and so could form useful design guides
 - To solve the difficult problem of general-purpose intelligence, start somewhere...
- If robots are to interact with humans in human environments, then having them endowed with some human-like cognitive features could be useful
 - To understand humans and their behaviour, to facilitate interaction
- We will return to this in the coming weeks...



Two main approaches (and others besides...)

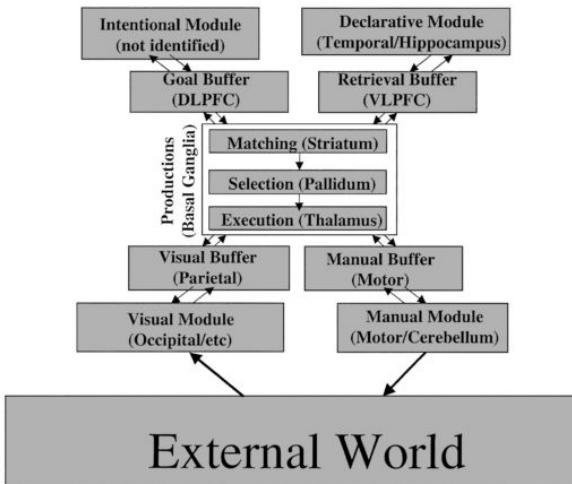
1. Derive set of mechanisms to use from human behavioural or other data
 - A “top-down” perspective
 - Can use this as a model of human behaviour
 - Typically based on data from psychology (overt behaviour)
2. Try to model fundamental principles of organisation of cognition, and implement these
 - A “bottom-up” perspective
 - Try to match to certain aspects of (human) behaviour
 - Typically derived from data closer to biology

Note...

There are many, many cognitive architectures, these are only two examples...

Example of Top-down: ACT-R

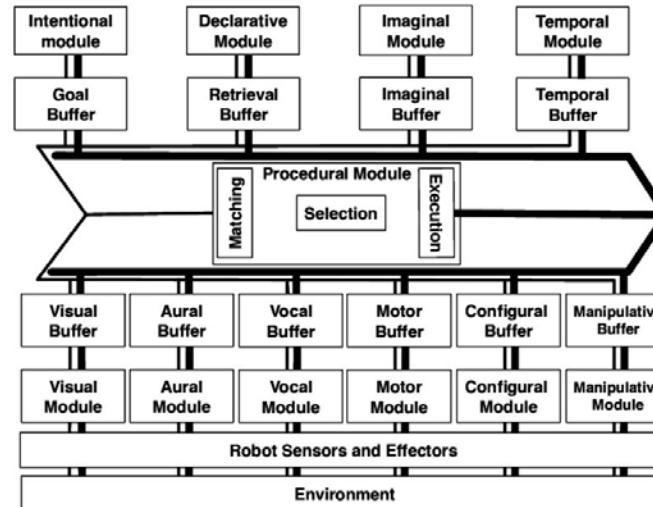
- ACT-R is an established cognitive architecture that has been applied to numerous models of human behaviour (reaction times, cognitive load, etc)
- Based on evidence from psychology, physiology, etc
- Hybrid Symbolic/Sub-symbolic processing
- <http://act-r.psy.cmu.edu/about/>



Example of Top-down: ACT-R

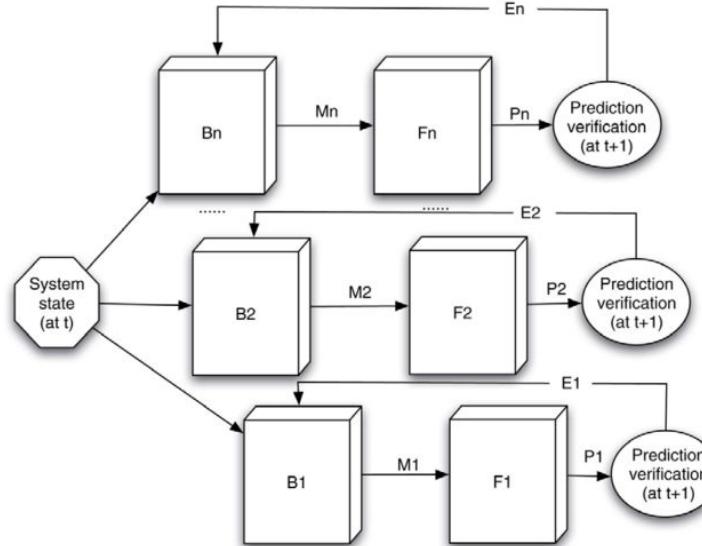


Images taken from (Trafton et al, 2013)

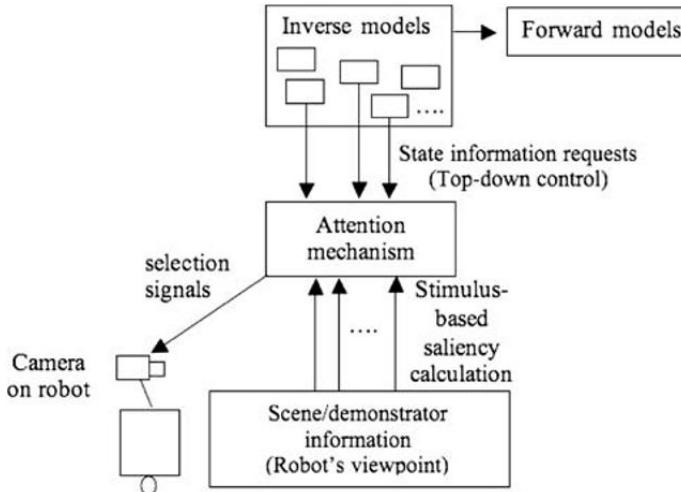


Example of Bottom-up: HAMMER

- Hierarchical, Attentive, Multiple Models for Execution and Recognition (HAMMER)
- Fundamentally based on Forward /Inverse model couplings, and applied to imitation, learning, assistance, etc
 - Strong theoretical foundations in psychology, neuroscience...
 - Comparing the different applications in a principled manner



Example of Bottom-up: HAMMER



Prediction of intentions

Cooperative
wheelchair control

See https://www.youtube.com/watch?v=CaH82_WzAeQ for a lecture by Prof. Yiannis Demiris on this, and other, work

Cognitive Architecture Issues

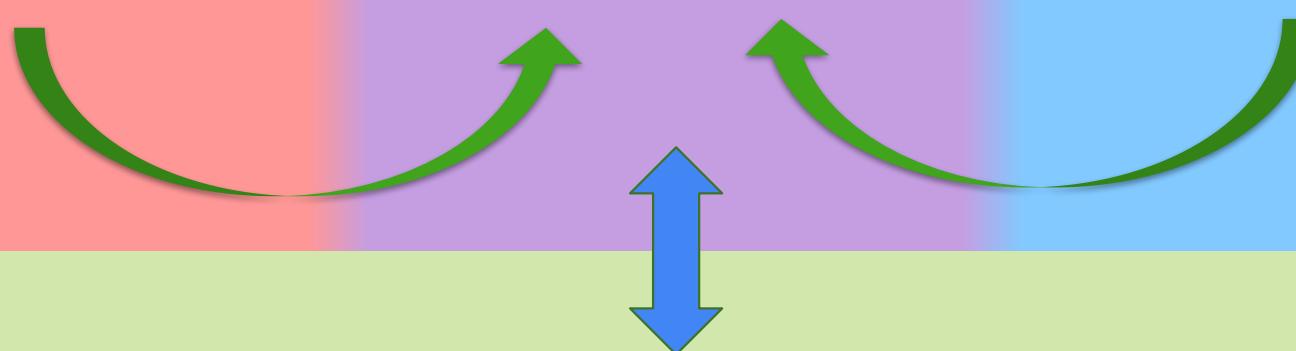
- How related to biology should it be?
 - Inspiration or constraints?
- Suitable level of abstraction?
 - Behaviour or mechanism?
- What is the purpose of using a Cognitive Architecture?
 - Functional or explanatory?

**DELIBERATIVE
ARCHITECTURES**

**HYBRID
ARCHITECTURES**

**REACTIVE
ARCHITECTURES**

**COGNITIVE
ARCHITECTURES**



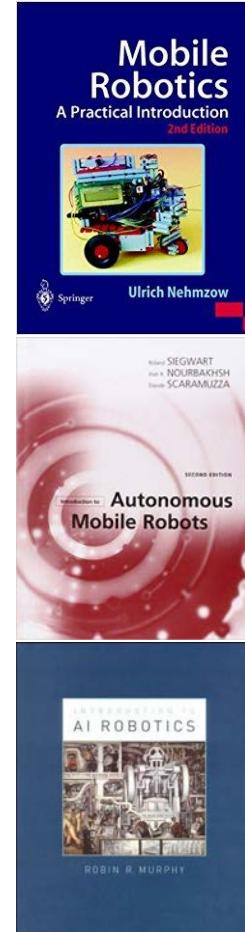
Tip of the Iceberg...

- 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...

E.g. <https://www.youtube.com/watch?v=G1t4M2XnIhI>

References / Reading

- Brooks, R.A., 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), pp.14–23.
- Brooks, R.A., 1990. Elephants don't play chess. *Robotics and Autonomous Systems*, 6, pp.3–15.
- Gat, E., 1998. "On Three-Layer Architectures." Artificial Intelligence and Mobile Robotics, in D. Kortenkamp, R. P. Bonnasso and R. Murphy (eds.), AAAI Press, pp.195-210.
- Kortenkamp, D. et al, 1998. Three NASA application domains for integrated planning, scheduling and execution. *AAAI AIPS Workshop*, pp.88-93
- Maes, P., 1989. How to do the right thing. *Connection Science*, 1(3), pp.291–232.
- Murphy, R., 2000. *Introduction to AI Robotics*, MIT Press.
- Sun, R., 2004. Desiderata for cognitive architectures. *Philosophical Psychology*, 17(3), pp.341–373.



Next week...

- Human-Robot Interaction
- Relating to the concepts from today, but also considering foundational concepts (and some applications)

CMP3101M AMR – Week 11

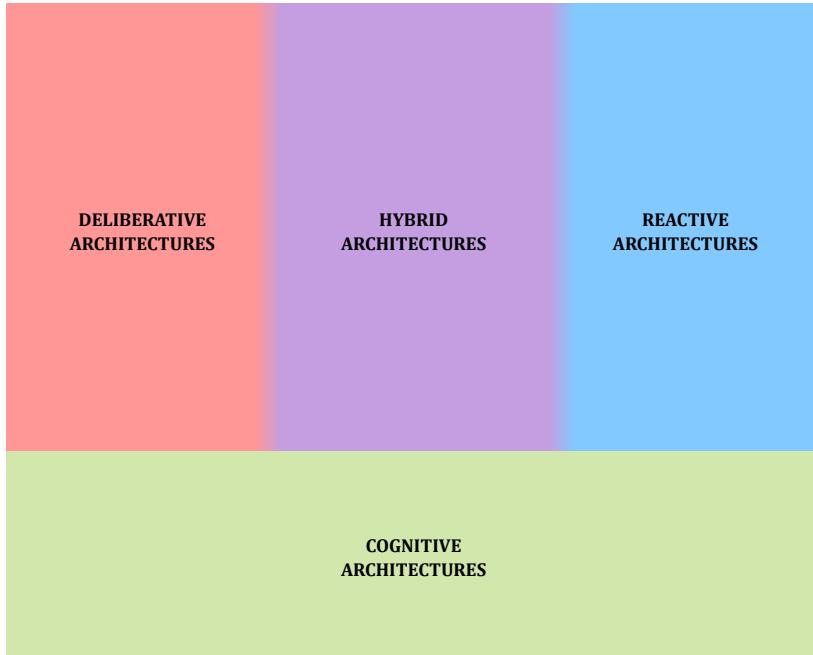
Human-Robot Interaction

part 1

Prof Marc Hanheide (with gratitude to Dr Paul Baxter)

mhanheide@lincoln.ac.uk

Last Week...



- Control Architectures for Autonomy
- Why they are necessary
- Three (+1) main paradigms

What is HRI?

Human-Robot Interaction (HRI) is:

... a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with humans. Interaction, by definition, requires communication between robots and humans.

or

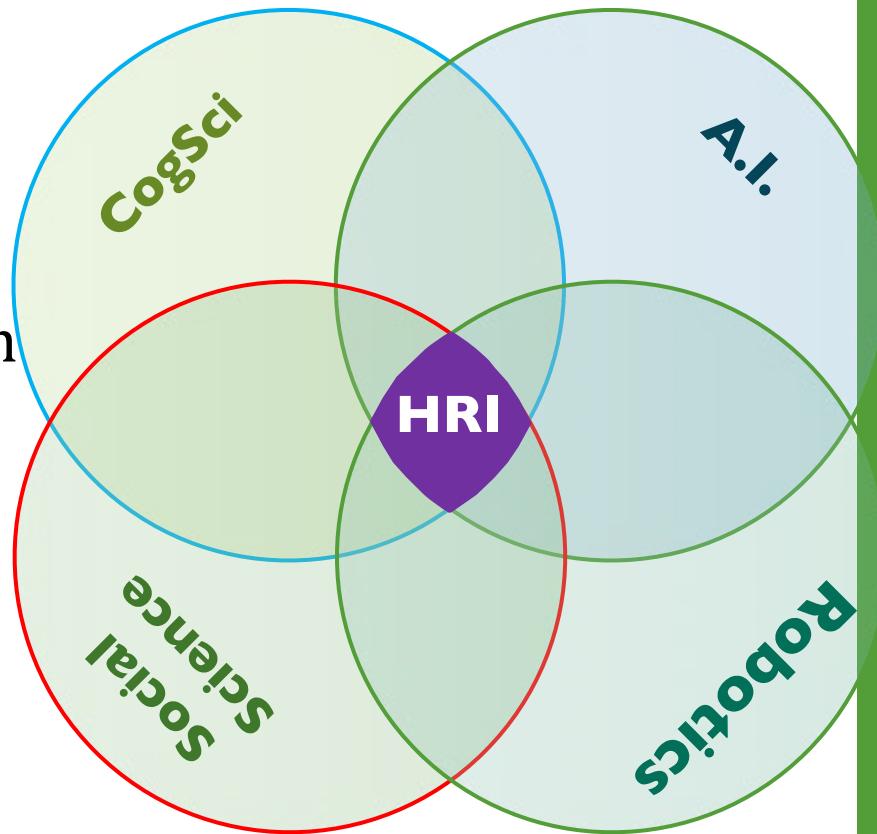
<http://humanrobotinteraction.org/1-introduction/>

Some overlaps with Human-Computer Interaction, so refresh your memory of last year's HCI module...

- In fact, origins of HRI lie in HCI
- Particularly in terms of development and evaluation methodologies

Aspects of HRI

- Psychology
 - Human Factors
-
- Sociology / Social Science
 - Human-Computer Interaction
-
- Computer Science
 - Mechatronics



From (Baxter et al, 2016)

Humans and Robots

Humans

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

Robots

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

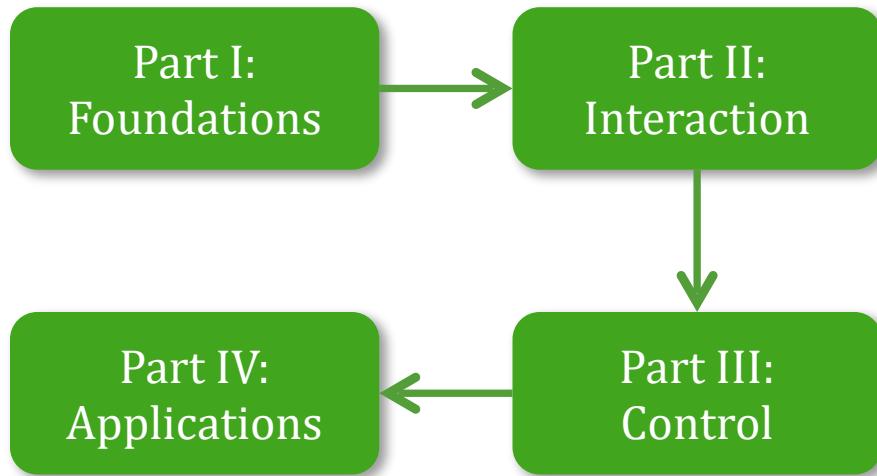


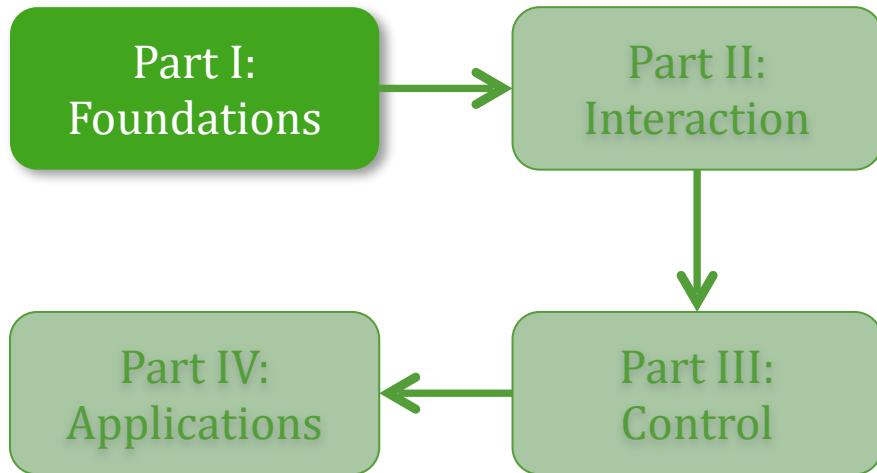
Why HRI in AMR?

- Purpose of robotics:
 - Automation: doing (boring/repetitive) jobs for people
 - Replacing people in risky/dangerous situations
 - ...
- Could also use for helping people:
 - Physical rehabilitation (exoskeletons, haptics, etc)
 - Social therapy, etc (e.g. socially-assistive robotics - SAR)
 - ...
- And as a tool for understanding people...?
 - The psychology/social science perspective

The interaction of humans and robots is pervasive in robotics
-> hence HRI...

Today...





Part I: Foundations

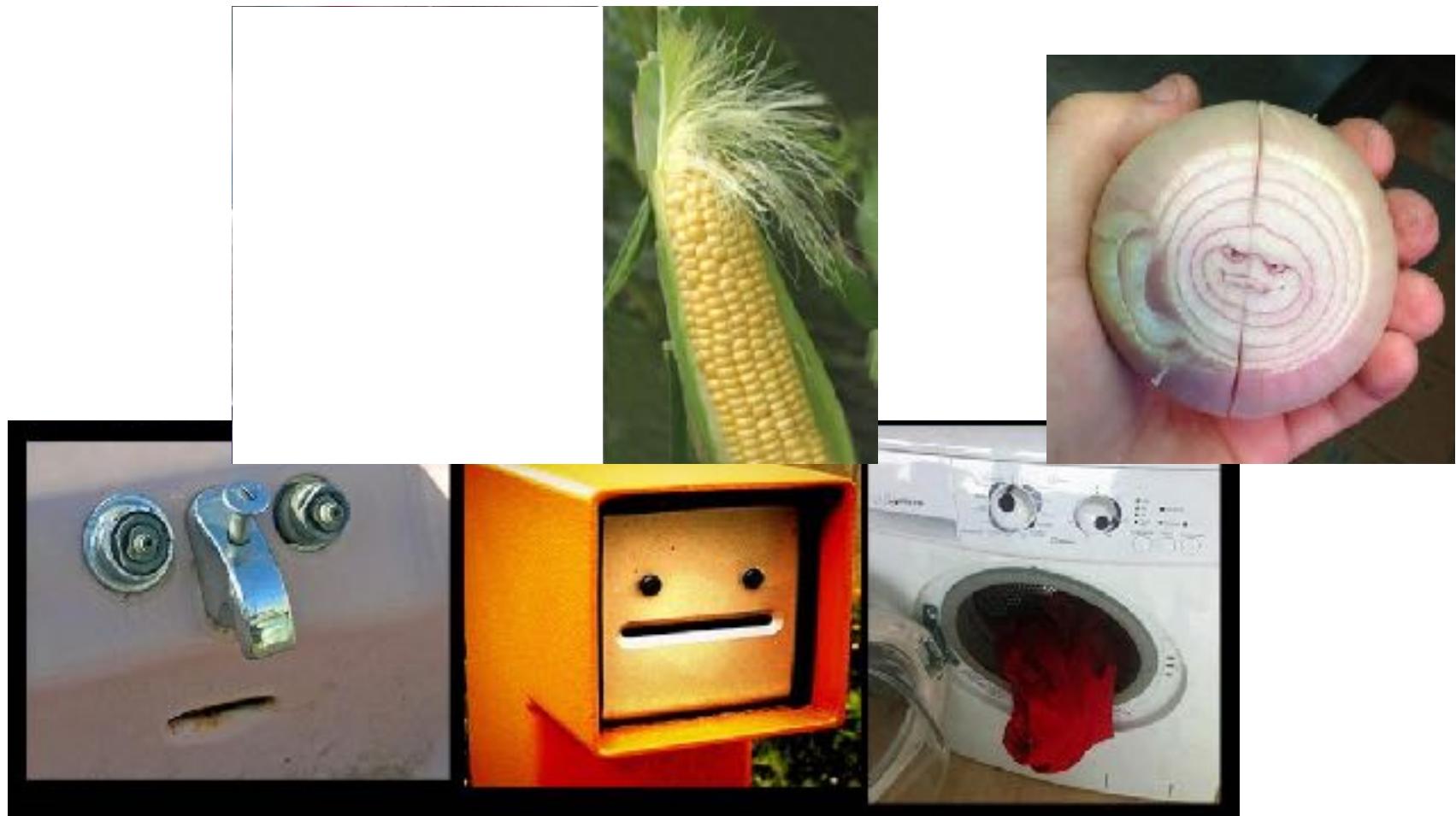
Assumptions and characteristics

Anthropomorphism

- Is the tendency to attribute human characteristics to inanimate objects, animals and others with a view to helping us rationalise their actions
 - (Duffy, 2003)
- Many examples in cartoons (Disney being particularly prolific)
- “the strategy of interpreting the behaviour of an entity (person, animal, artefact, whatever) by treating it as if it were a rational agent who governed its ‘choice’ of ‘action’ by a ‘consideration’ of its ‘beliefs’ and ‘desires’”
 - (Dennet, 1996): the intentional stance
- Embracing this concept in HRI
 - Taking advantage of it rather than trying to avoid it
 - Appearance and Behaviour
- Related concepts: active perception, and gestalt psychology from HCI – wanting to find and group information in ‘meaningful’ ways

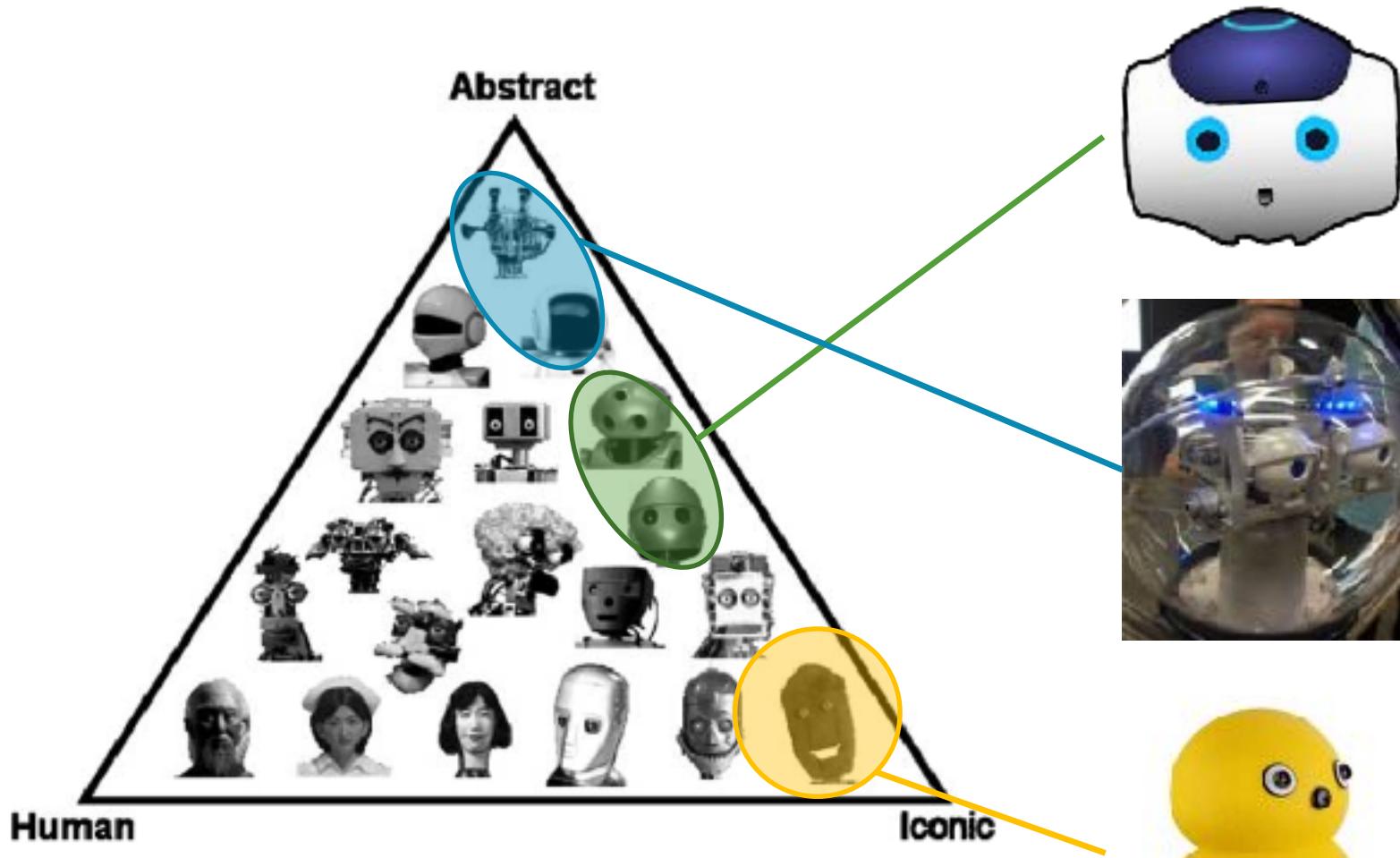


Anthropomorphism: Appearance



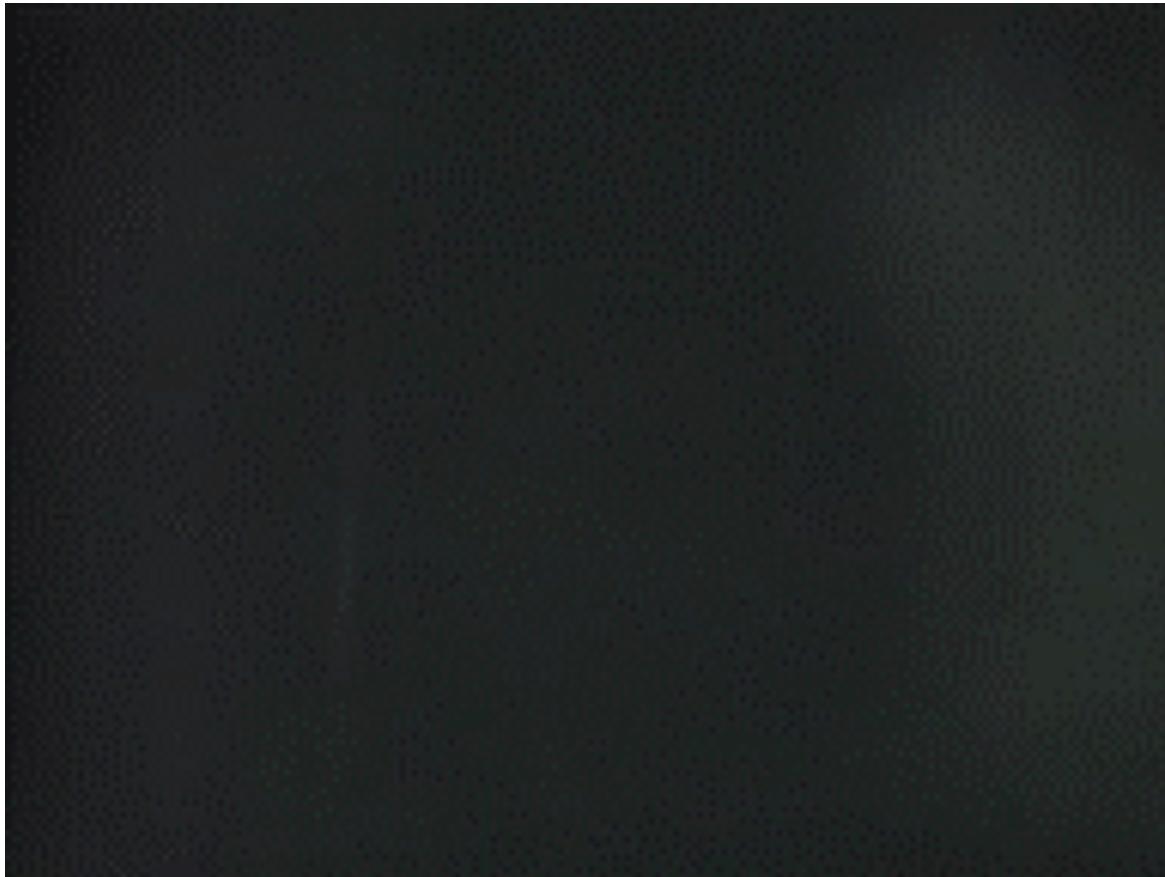
Pareidolia – perceiving a familiar pattern (typically face) where there is none

Anthropomorphism: Robot Faces



Design space for humanoid heads (Duffy, 2003)

Anthropomorphism: Behaviour



Apparent Behaviour (Heider & Simmel, 1944)

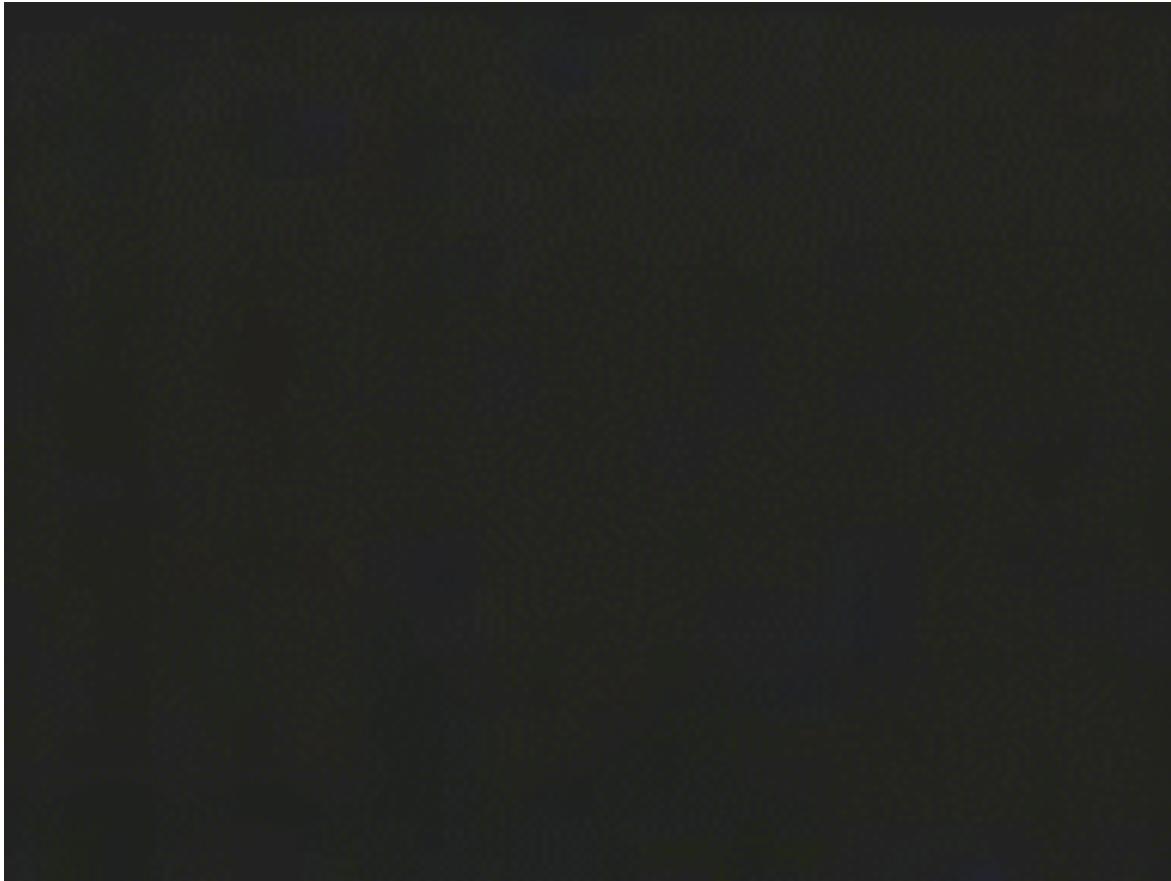
Anthropomorphism: Behaviour



While working on your assignment, how many of you:

- Swore at your robot?
- Complimented your robot?
- Referred to your robot as he/she?
- Gave your robot a name?

Anthropomorphism: Behaviour



Full video: <https://www.youtube.com/watch?v=VWeRC6j0fW4>

Also see: http://cosmo.nyu.edu/hogg/lego/braitenberg_vehicles.pdf

Anthropomorphism: Robots



Geminoids (Ishiguro et al)

The Uncanny Valley

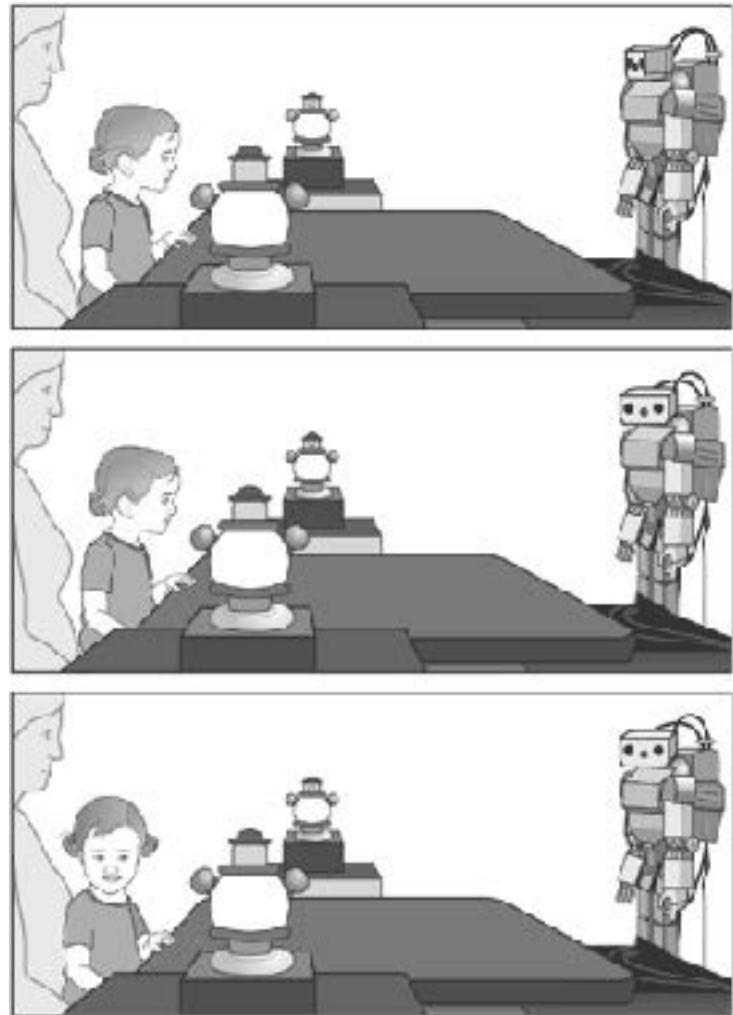


- One possible explanation: a conflict between cues (Moore, 2012)
 - Hence why the “moving” curve more pronounced
 - Greater mismatch between movement and appearance (see anthropomorphism notes above)

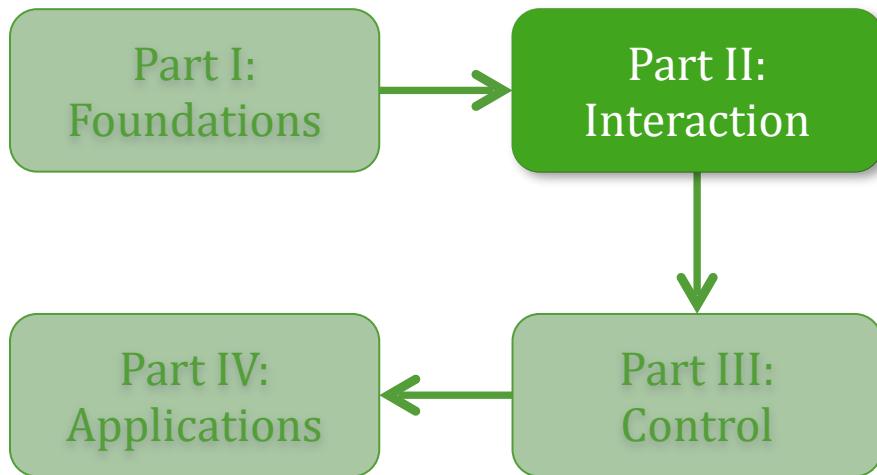
- The “Uncanny Valley”
 - Increasing human likeness increases familiarity...
 - ...however, at a certain point, a sharp drop in familiarity, resulting in revulsion, etc
- Refer to (Mathur et al, 2016) for an overview
- Reasons for this not fully understood

Attribution of Agency to Robots

- Particularly if certain characteristics are present, people will naturally attribute agency to an inanimate object
 - Though this illusion can be broken in the interaction
- This is (at least partly) learned from experience
- Seen in children with a robot for example
 - Meltzoff et al (2010)
 - Children watch adult interact with robot (joint attention)
 - These children then more likely to follow gaze when they interact with robot themselves



From Meltzoff et al (2010)



Part II: Interaction

Types and contexts

Fundamentally two modes...

Proximate

- The human and the robot are in the same space
- Physical and social interactions fall in this category
 - E.g. service robots



Remote

- The human and the robot are in different locations
 - Maybe even temporally removed
- E.g. teleoperation, supervised control, ...



Two types of Interaction

Explicit

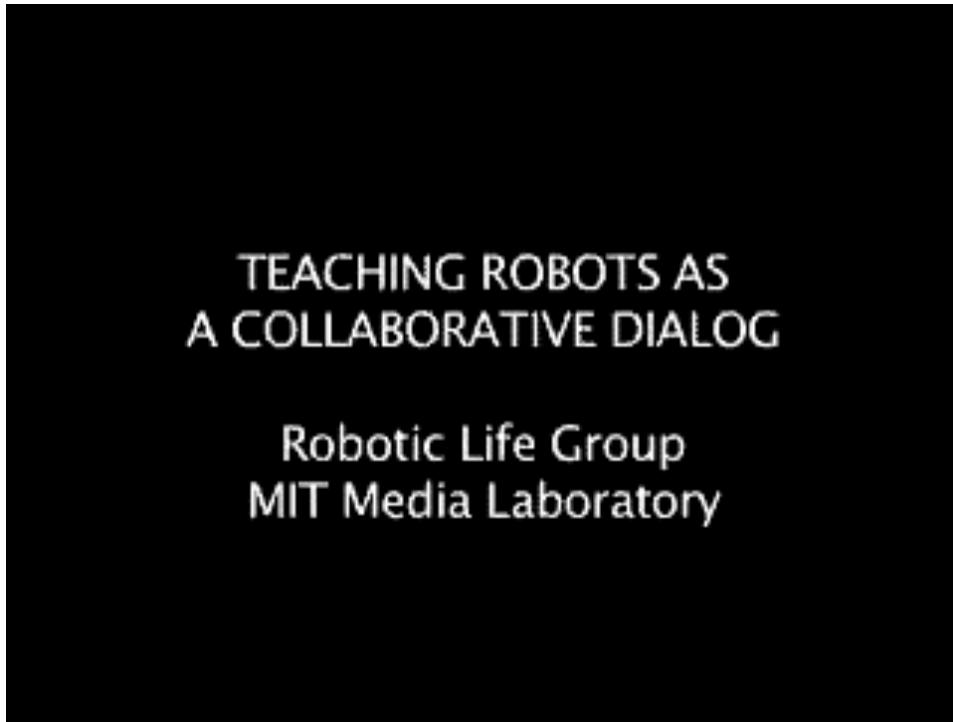
- An action performed with the purpose of eliciting a reaction
- Dialogue: e.g. asking a question
- Manipulation: e.g. handing over an object, or pointing

Implicit

- Actions are modified due to presence of an other, but no reaction is expected
- Navigation: e.g. avoiding humans in the way

Capacity for overlap between Explicit and Implicit: e.g. avoiding humans, but engaging in some strategies to encourage humans to move out of the way (next week!)

Explicit Interactions: example

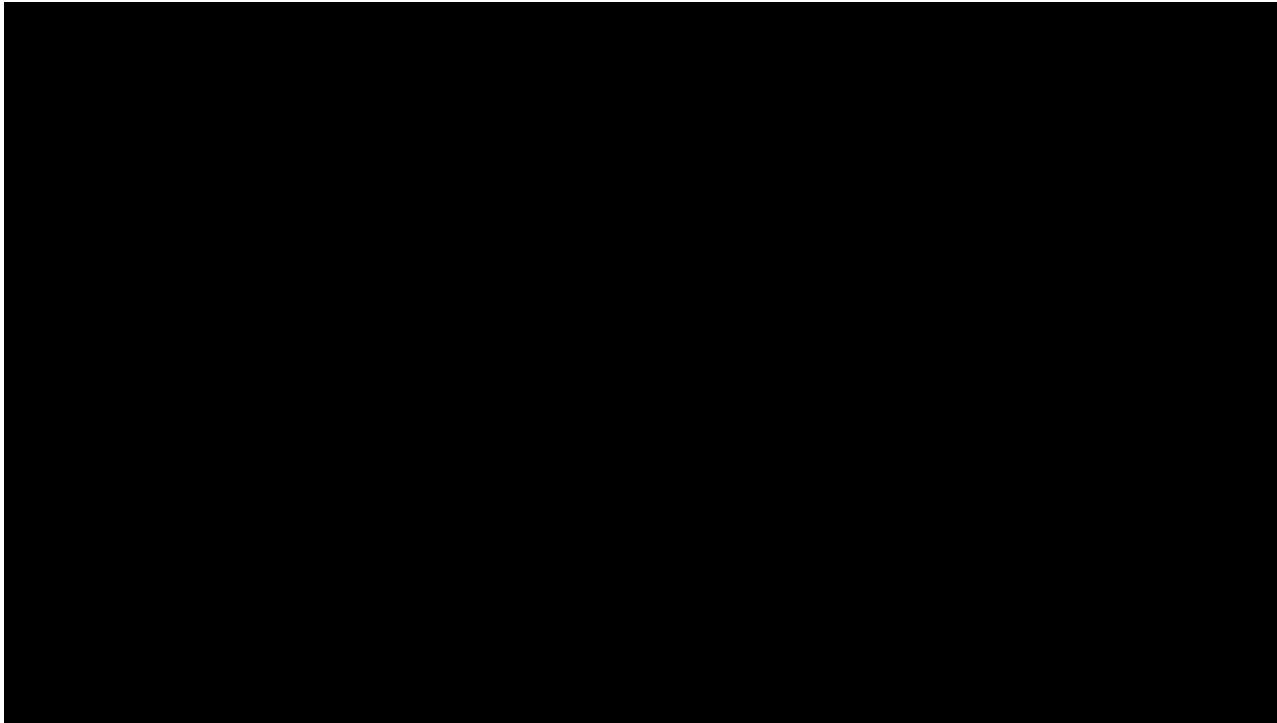


- Leonardo robot with Andrea Thomasz (MIT, 2006)
- Explicit interaction
 - Pointing from human
 - Gaze gestures from robot to indicate ready for next instruction
- See: <https://www.youtube.com/watch?v=GHllFrL7dKM>

Explicit Interactions: example

- Explicit interaction scenario
 - Robot and human facing each other
 - Using the same workspace
 - Human providing explicit instruction, with gestures
 - Robot performs actions, looks back at human
- Human teaching:
 - Names, attributes, affordances
- Robot learns from:
 - Speech, observation

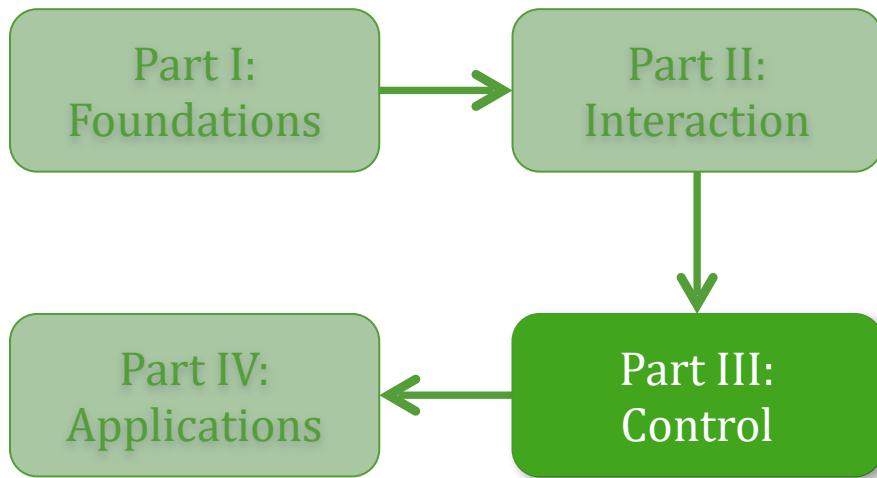
Implicit Interactions: example



- 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!

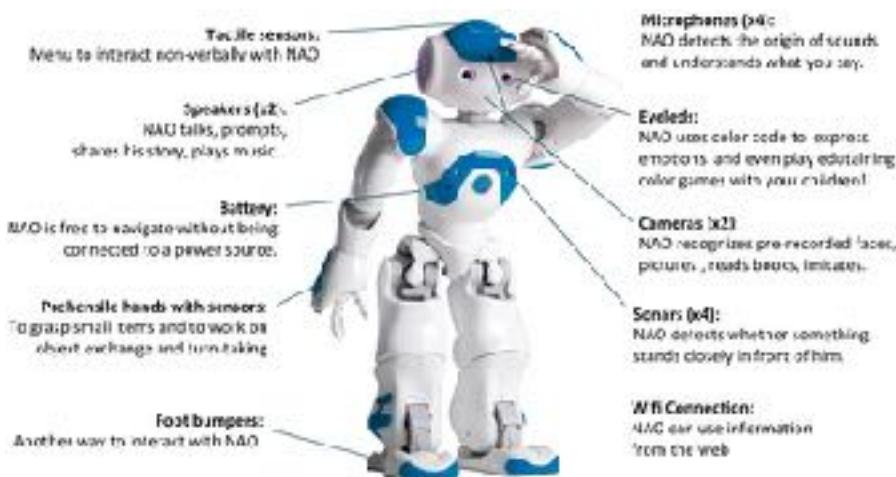
- Sensing
- Decision making
- Acting



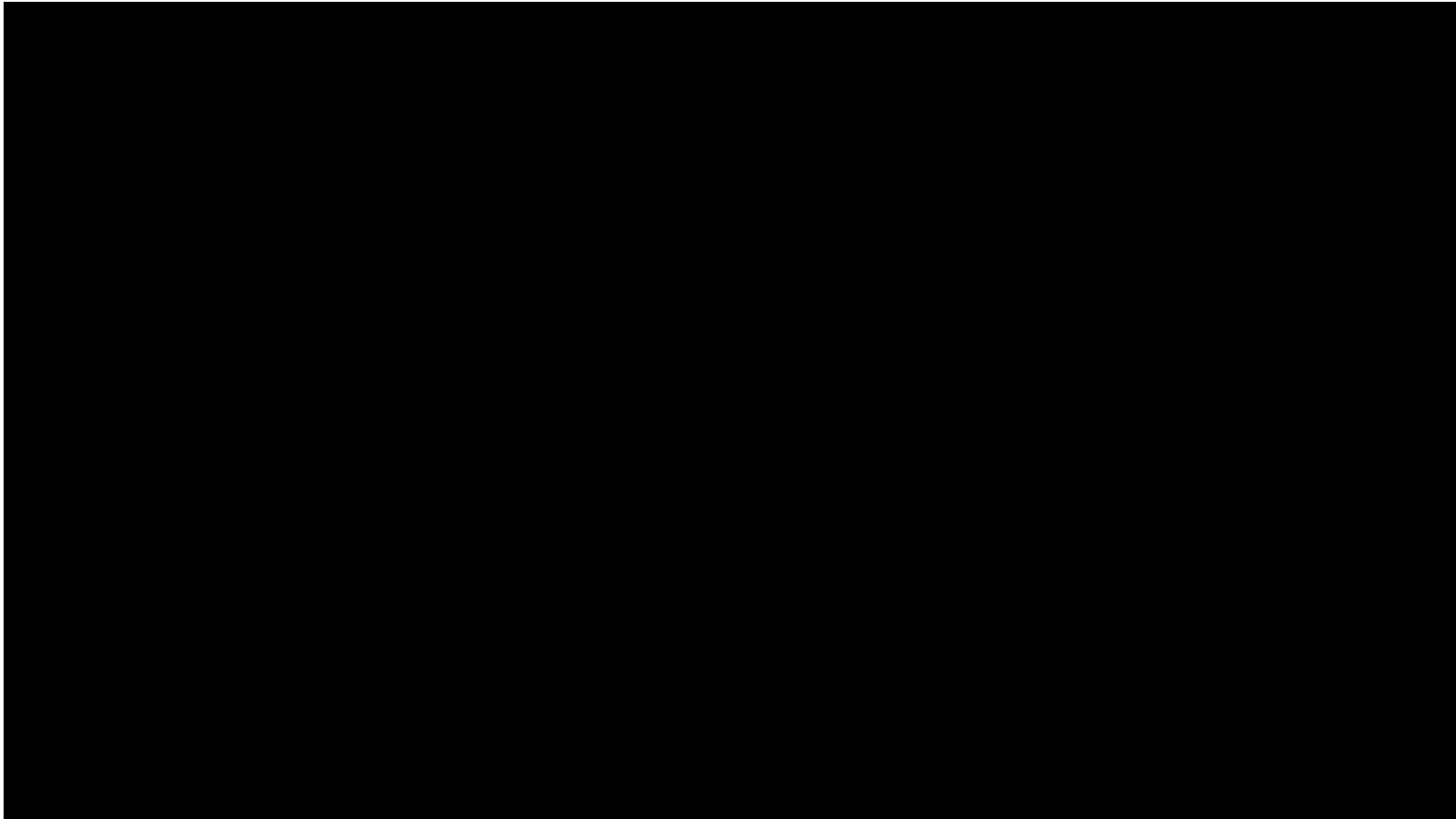
Sphero Robotics

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...

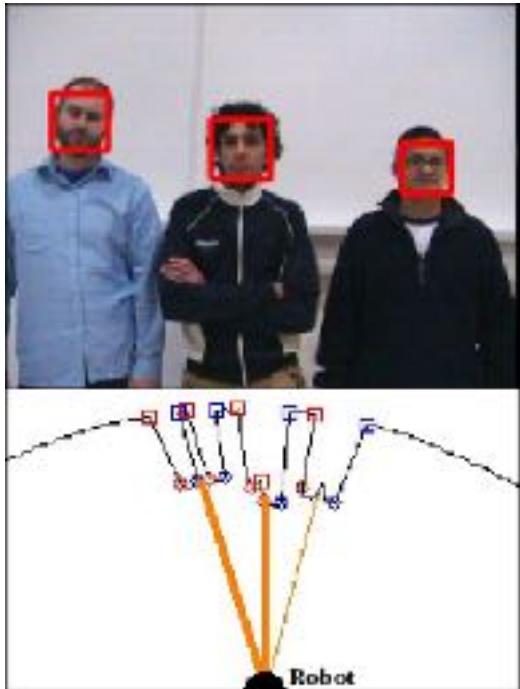


Sensing is difficult...

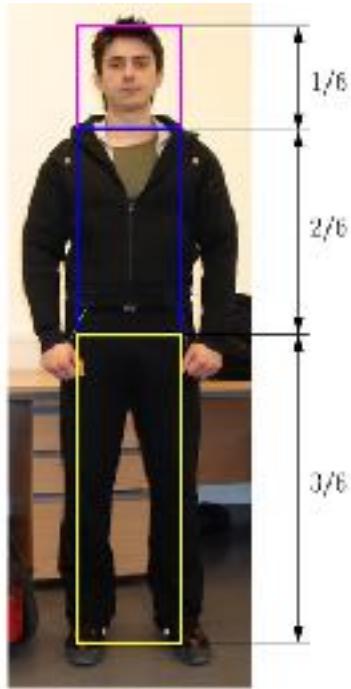


See paper and video (2012) at <http://dl.acm.org/citation.cfm?doid=2559636.2559650>

Human Detection

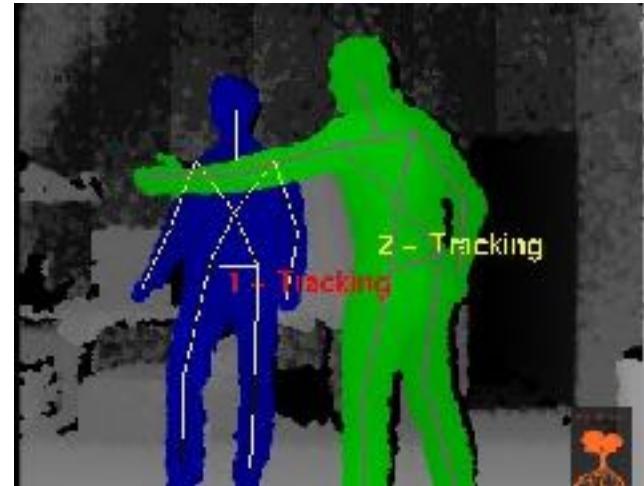


Person detection from
laser scans of legs



Person detection from
camera-based clothing
and face recognition

Both images from Dr. Bellotto homepage: <http://webpages.lincoln.ac.uk/nbellotto/software.html>



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!”

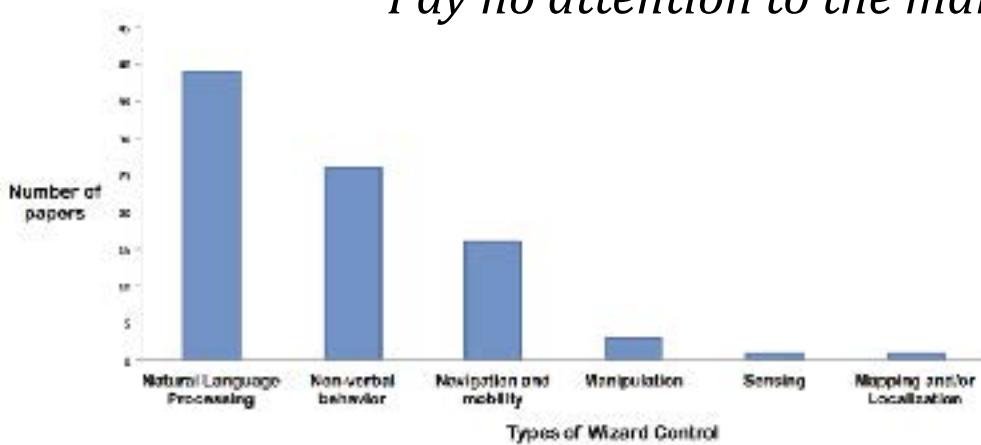
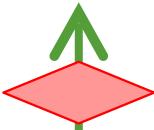


Figure 4. Chart depicting the types of Wizard controls employed in the included papers. Some papers described using more than one type of control.

From Riek (2012)

1. Wizard of Oz

Teleoperation

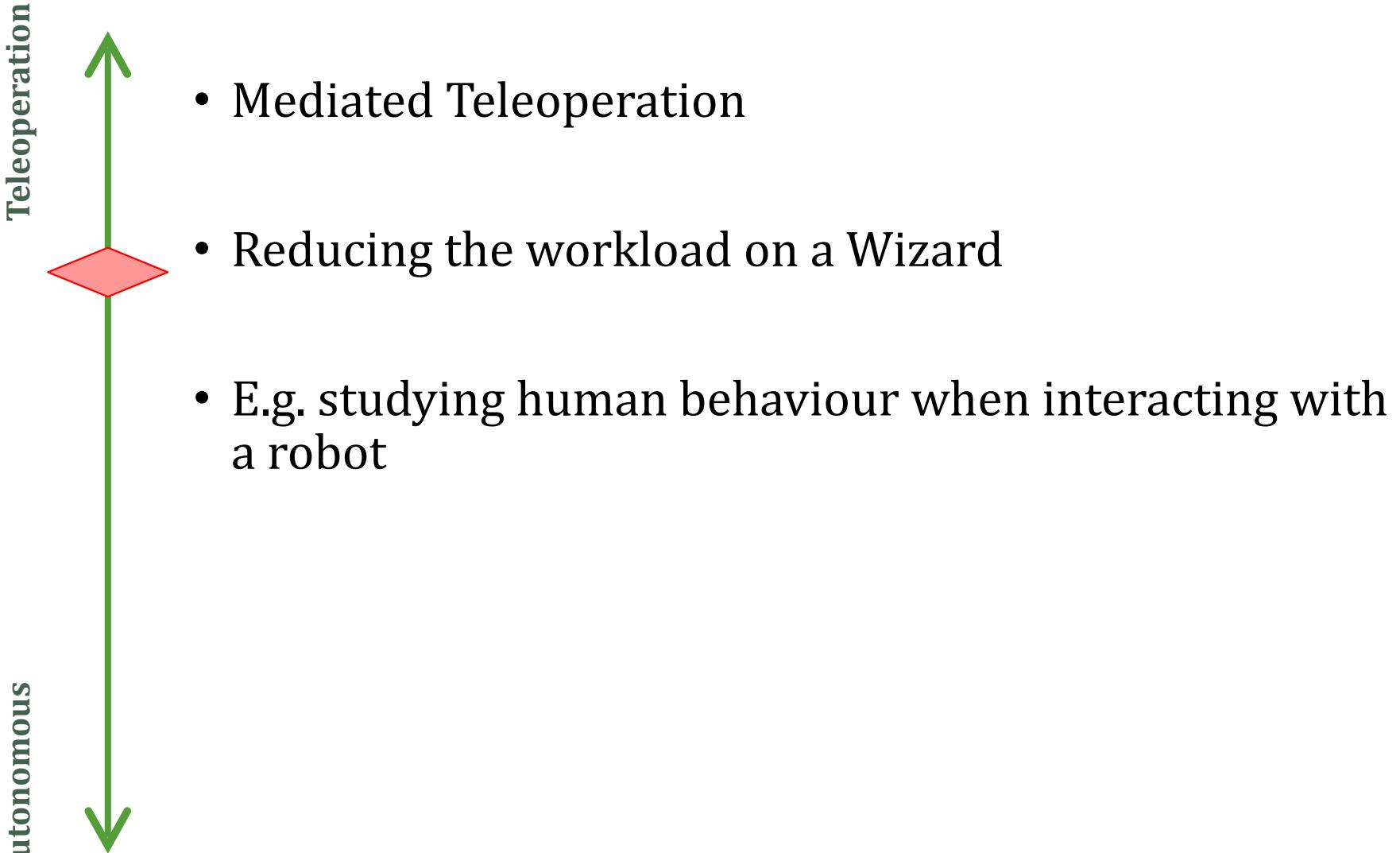


- Teleoperation
- Giving the impression of autonomy
- The Wizard is hidden from view, or not obviously associated with the control of the robot
 - Either way, the participant is unaware of the remote control.
- Complete WoZ entails full remote control of all aspects of behaviour

Autonomous

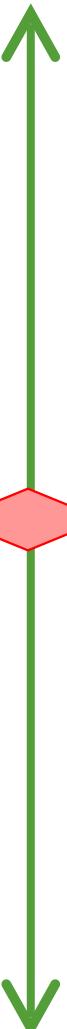


2. Scripted Robot Behaviour



3. Partial Autonomy

Teleoperation



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

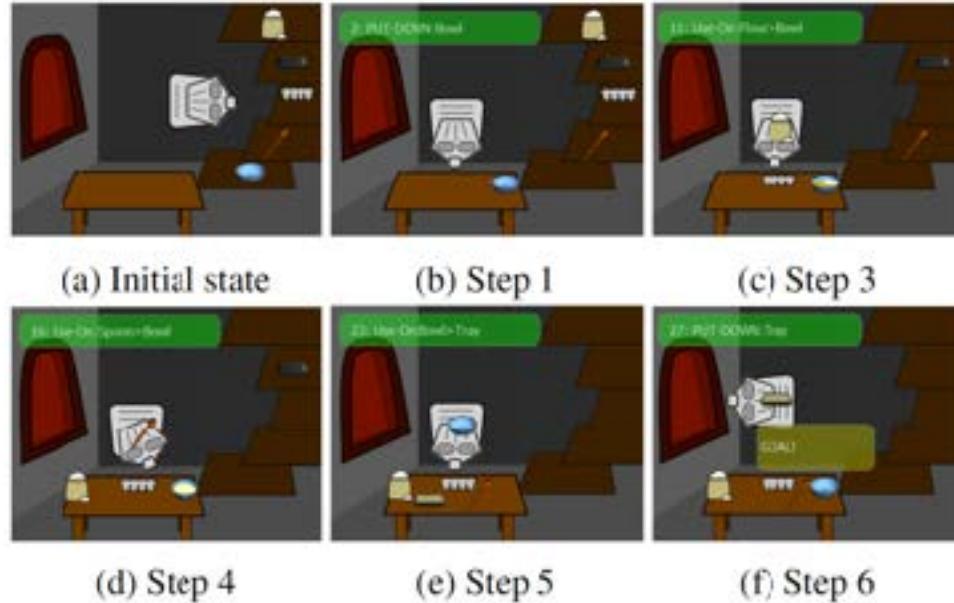


4. Supervised Autonomous Behaviour

Teleoperation



- 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



Autonomous

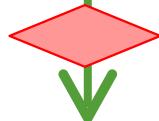
5. Full Autonomy

Teleoperation



- The typical goal for robot development
 - No requirement for human remote-controller present
 - Though environment/context may require someone close by for safety...
- Full autonomy implies capacity for adaptation
 - E.g. only using predefined behaviours or waypoints may be autonomously executed, but useless if something changes (e.g. obstacles)
 - Adaptation suggests learning...

Autonomous



Levels of Autonomy in Autonomous Driving

THE 6 LEVELS OF AUTONOMOUS DRIVING

DRIVER VEHICLE	L0 No Automation	L1 Driver Assistance	L2 Partial Automation	L3 Conditional Automation	L4 High Automation	L5 Full Automation
	 In charge of all the driving	 Must do all the driving but with some basic help in some situations	 Must stay fully alert even when vehicle assumes some basic driving tasks	 Must be always ready to take over within a specified period of time when the self-driving systems are unable to continue	 Can be a passenger who, with notice, can take over driving when the self-driving systems are unable to continue	 No human driver required steering wheel optional everyone can be a passenger in an L5 vehicle
	Responds only to inputs from the driver but can provide warnings about the environment	Can provide basic info, such as audio/wireless emergency braking or lane keep support	Can automatically steer, accelerate, and brake in limited situations	Can take full control over steering, acceleration, and braking under certain conditions	Can assume all driving tasks under nearly all conditions without driver attention	In charge of all the driving and can operate in all environments without need for human intervention

Contrasting WoZ and Autonomy

Social Sciences Point of View

WoZ

- Pros:
 - Remove uncertainty
 - Focus on evaluation of interaction rather than robot
 - Full control over robot behaviour
 - Repeatable experiment setup (??)
 - Easier to implement
- Cons:
 - Human-human interaction with a robot in the middle?
 - Not consistent...

Autonomous

- Pros:
 - Study with state-of-the-art robot instead of dummy
 - Testing system robustness
 - How to replicate human-like learning on robot
- Cons:
 - High uncertainty
 - Not necessarily repeatable
 - High maintenance
 - Can be slow...

Contrasting WoZ and Autonomy

Computer Science Point of View

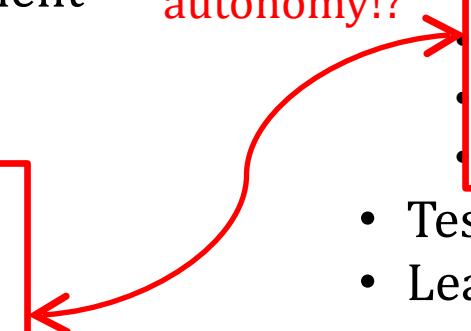
WoZ

- Pros:
 - Remove uncertainty
 - Evaluate robot design
 - Repeatable experiment
- Cons:
 - No evaluation of:
 - Perception
 - Reasoning
 - Learning
 - World Model
 - Action selection
 - Evaluates only robot design

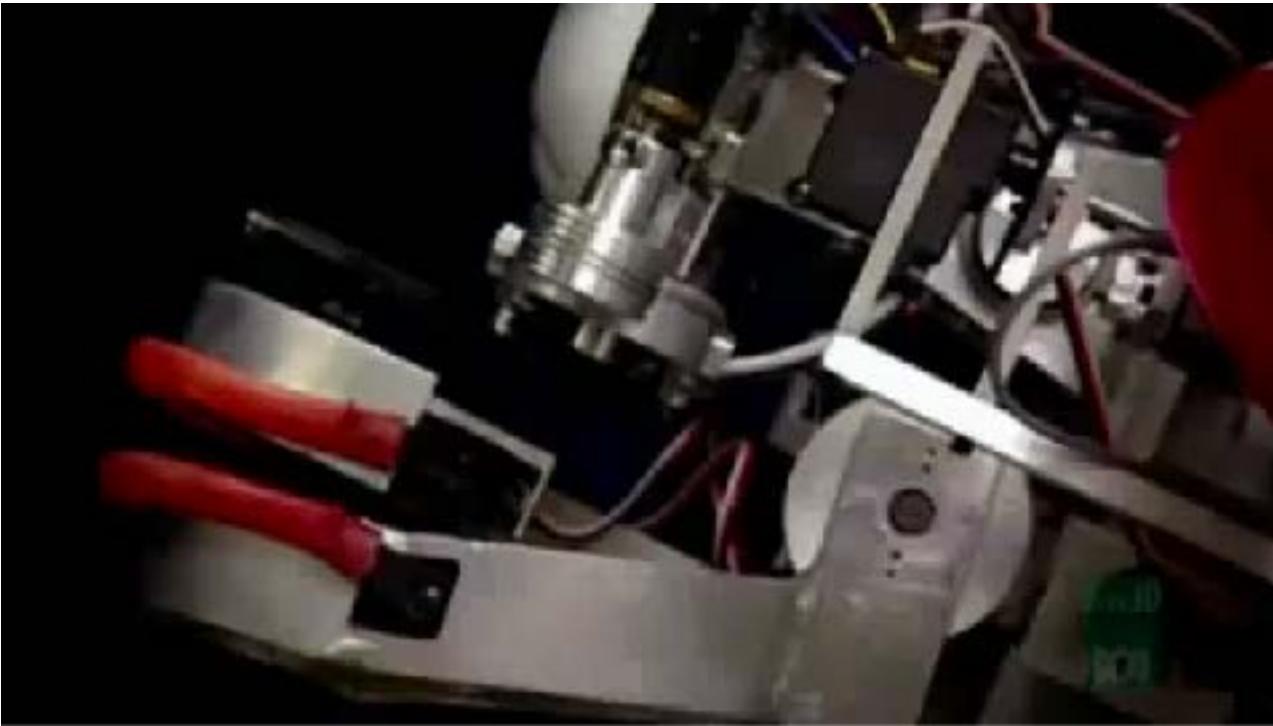
Autonomous

- Pros:
 - Evaluation of:
 - Perception
 - Reasoning
 - Learning
 - World Model
 - Action selection
 - Testing system robustness
 - Learning from interaction
- Cons:
 - High uncertainty
 - Not necessarily repeatable
 - Can be slow...

Level of
autonomy!?

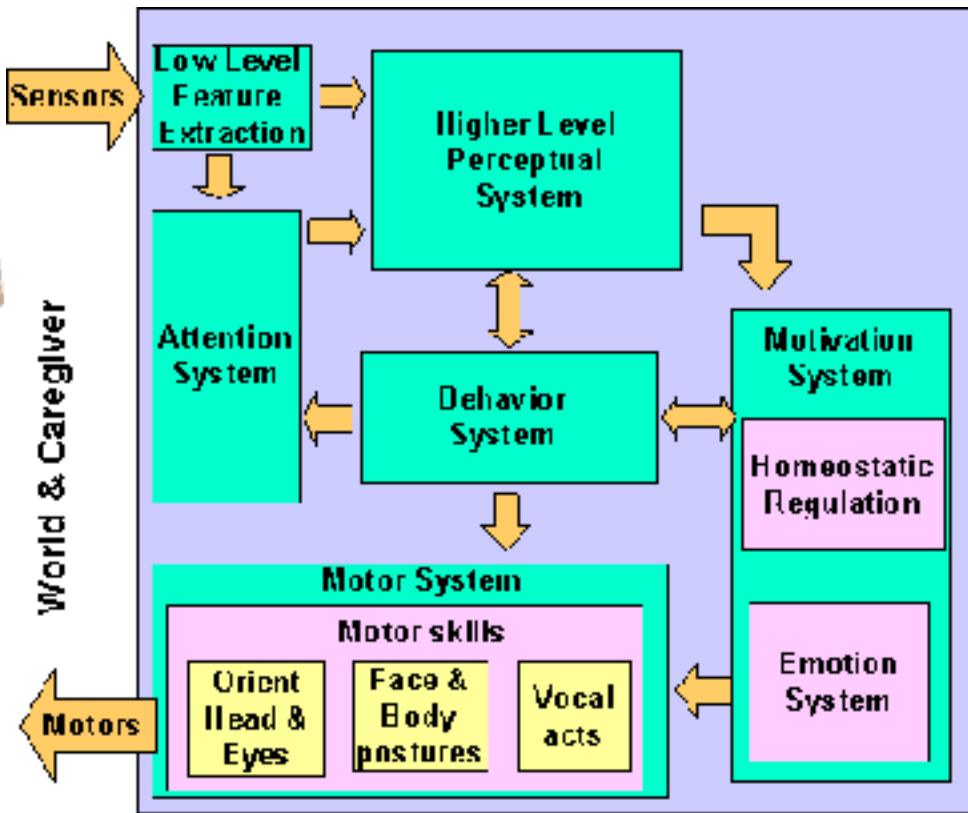
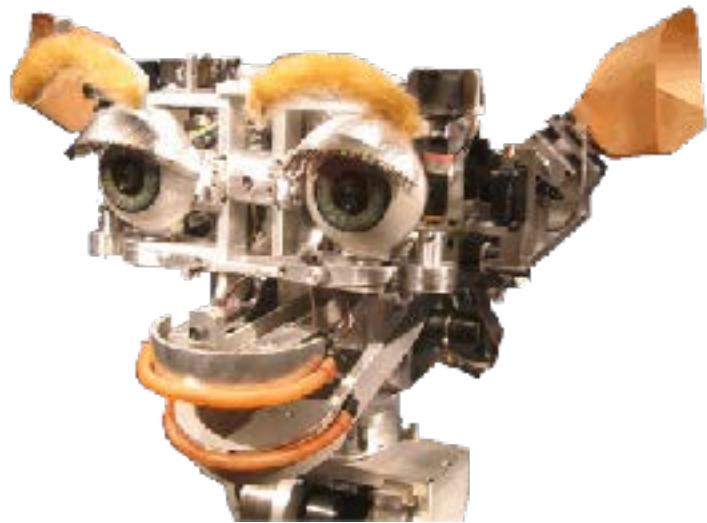


Control Architectures: Reactive



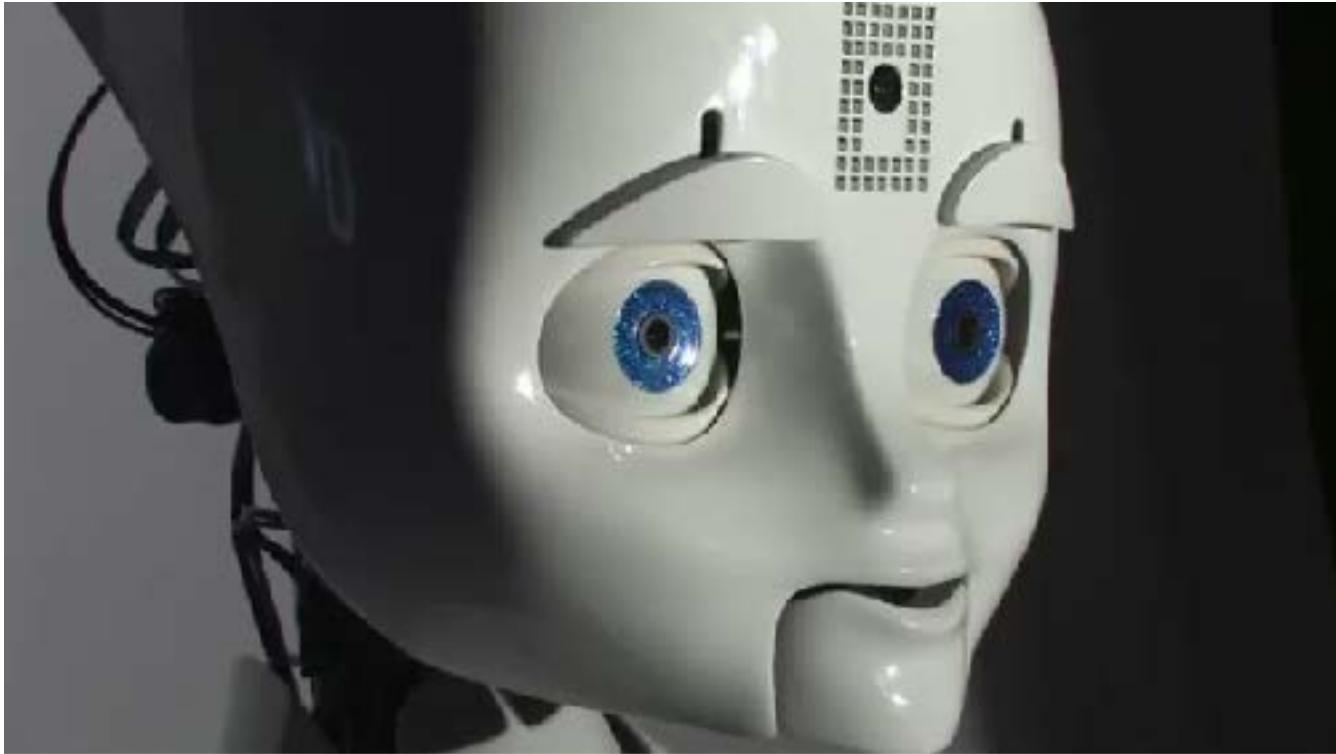
- Kismet (MIT, 1990's)
- See video at: <https://www.youtube.com/watch?v=8KRZX5KL4fA>
- Note the brief discussion of anthropomorphisation...

Control Architectures: Reactive



- Kismet control architecture
- See <http://www.ai.mit.edu/projects/sociable/kismet.html>

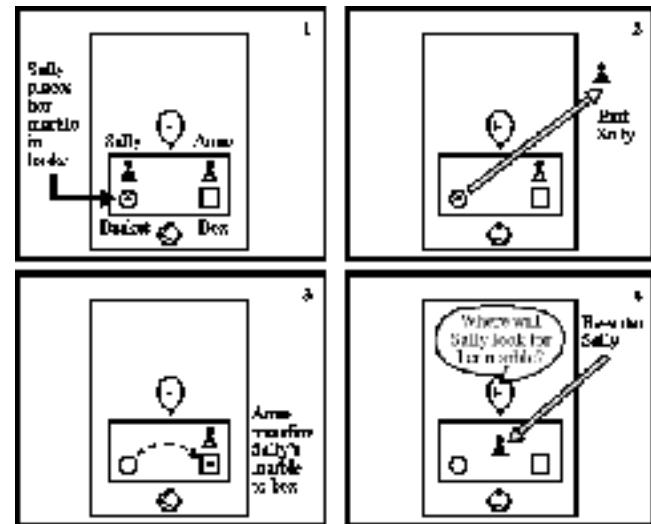
Control Architectures: Cognitive



- MDS – (Octavia @ NRL)
- Video: <https://www.youtube.com/watch?v=aQS2zxmrrrA>
- Controlled with cognitive architecture ACT-R/E (Trafton, 2013)

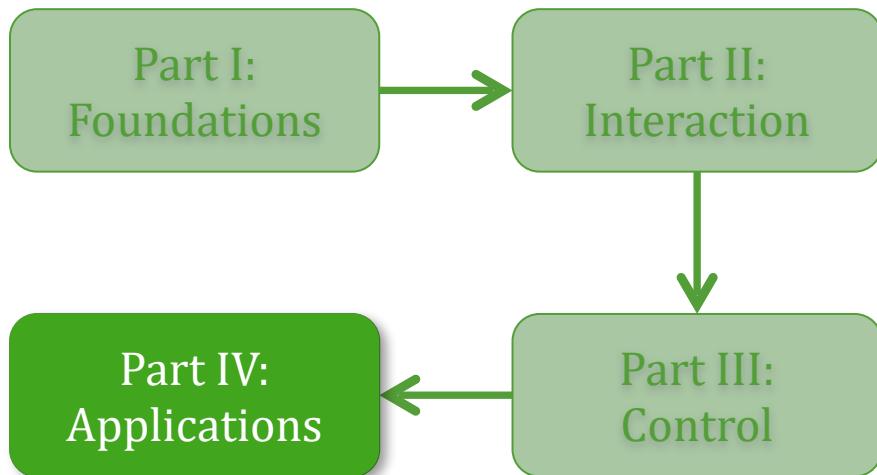
Control Architectures: Cognitive

- Octavia case study: Theory of Mind
 - The capacity of humans to attribute mental states to others and to oneself, and to understand that others may have different states than oneself



<http://www.holah.karoo.net/baronstudy.htm>

- Further relevant concepts to CogArch:
 - Belief-Desire-Intention architectures, e.g. <http://www.aaai.org/Papers/ICMAS/1995/ICMAS95-042.pdf>
 - Affective Computing related to HRI, e.g. <http://cseweb.ucsd.edu/~lriek/papers/riek-acd-aisb09.pdf>



Part IV: Applications

Robot roles and ethics

Robot Roles: Learning Peer



ROBOTICS
WITH
PLYMOUTH
UNIVERSITY

alize
Autism, Mental Health, Social Skills, Early Intervention

The 'Sandtray'

Mediating Social Human-Robot Interaction

Plymouth University, U.K.

- Robot as a learning partner for a child: <http://ieeexplore.ieee.org/abstract/document/6249477/>
- Robot and child co-located, facing each other over a touchscreen, and can both interact with the touchscreen
- Robot plays the role of a peer: not perfect performance, makes mistakes, and adapts its behaviour

Robot Roles: Therapy Aid



- Probo robot: a green elephant-like cuddly robot
- Used in autism therapy for children, under the supervision of a therapist
- Used in Robot-Enhanced Therapy, see <http://www.dream2020.eu/>

Other Robot Roles

Have seen:

- Robot as learning partner
- Robot as assistant
- Robot as therapy tool

Could be:

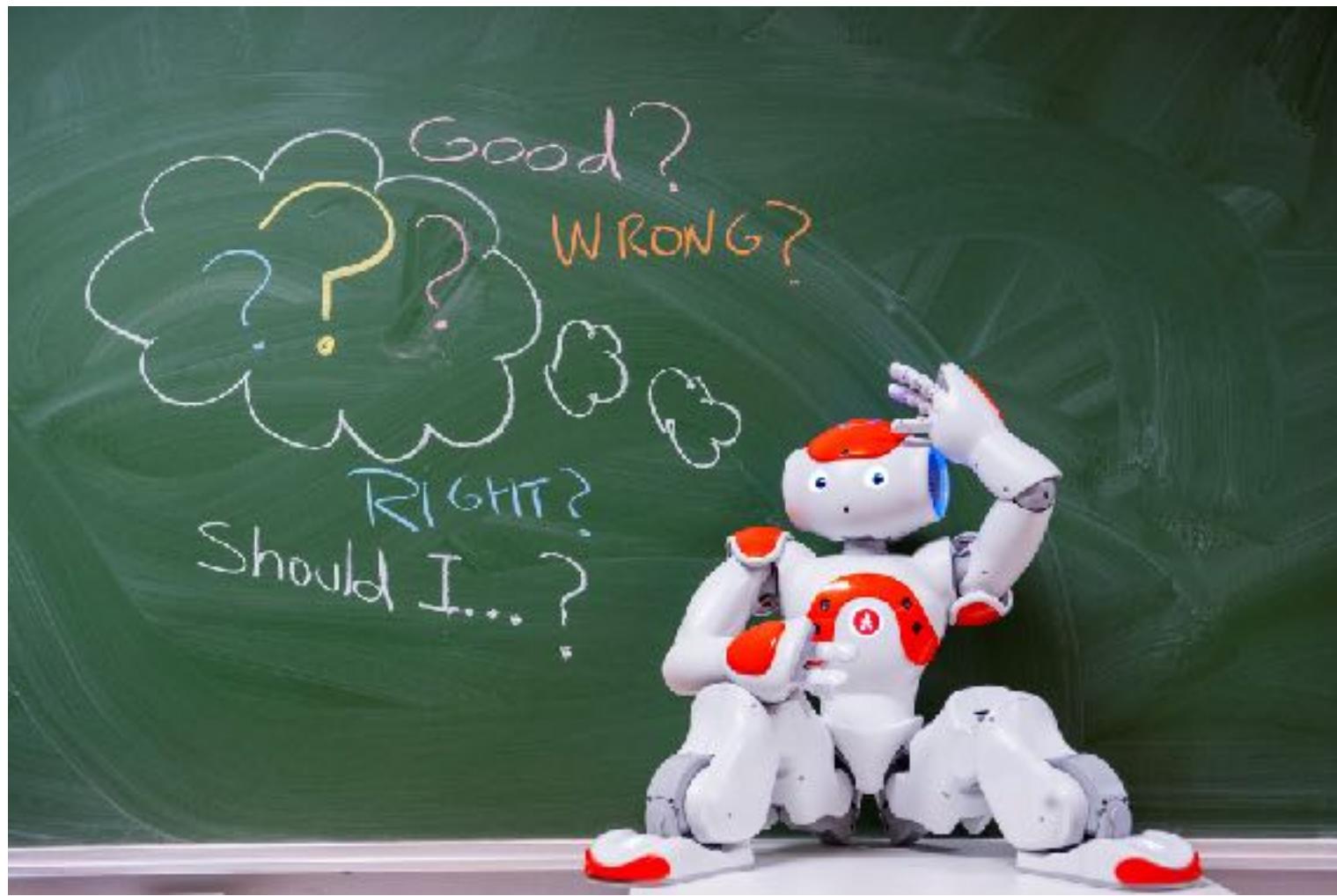
- Robot as teacher/tutor
- Robot as tool
- Robot as therapist
- Robot as team member

(Autonomous) Tool

Social Agent

- Above, level, below in social hierarchy

Ethical Issues

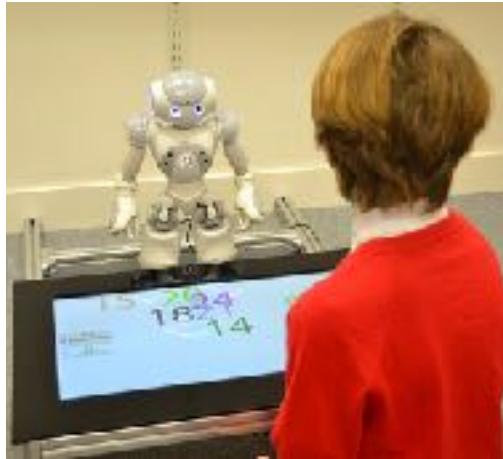
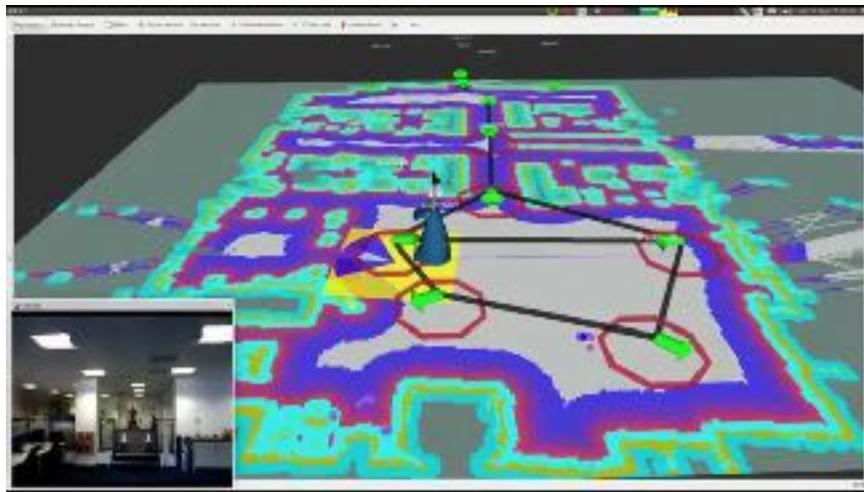


Ethical Issues

- Is it right to use robots as social agents in this way?
 - Issue of deception?
 - Are we replacing social contact with other humans if we use these devices?
 - In the case of child therapy, are the problems made worse (e.g. getting used to interacting with robot rather than people)?
 - Attachment to robots rather than humans...
- Technical/Legal concerns:
 - Data protection, and the role of data recording/capture
 - Memory of prior interactions and privacy of this information

Next Week...

- Engagement in HRI
- Navigation in HRI
 - Human-aware planning
- Methods for HRI Evaluation



References / Reading

- Baxter, P. et al., 2016. From Characterising Three Years of HRI to Methodology and Reporting Recommendations. In *HRI 2016*. Christchurch, New Zealand: ACM Press, pp. 391–398.
- Duffy, B.R., 2003. Anthropomorphism and the social robot. *Robotics and Autonomous Systems*, 42(3–4), pp.177–190.
- Heider, F., Simmel, M., 1944. An experimental study of apparent behavior. *The American Journal of Psychology*, Vol 57, 243-259
- Mathur, M.B. & Reichling, D.B., 2016. Navigating a social world with robot partners: A quantitative cartography of the Uncanny Valley. *Cognition*, 146, pp.22–32.
- Meltzoff, A.N. et al., 2010. “Social” robots are psychological agents for infants: a test of gaze following. *Neural networks : the official journal of the International Neural Network Society*, 23(8–9), pp.966–72.
- Moore, R.K., 2012. A Bayesian explanation of the “Uncanny Valley” effect and related psychological phenomena. *Scientific Reports*, 2, p.864.
- Riek, L., 2012. Wizard of Oz Studies in HRI: A Systematic Review and New Reporting Guidelines. *Journal of Human-Robot Interaction*, 1(1), pp.119–136.
- Senft, E., Baxter, P., Kennedy, J., Lemaignan, S., & Belpaeme, T., 2017. Supervised Autonomy for Online Learning in Human-Robot Interaction. *Pattern Recognition Letters*, 99, p77–86.
- Trafton, J.G. et al., 2013. ACT-R/E : An Embodied Cognitive Architecture for Human-Robot Interaction. *Journal of Human-Robot Interaction*, 2(1), pp.30–54.

CMP3101M AMR – Week 12

Human-Robot Interaction

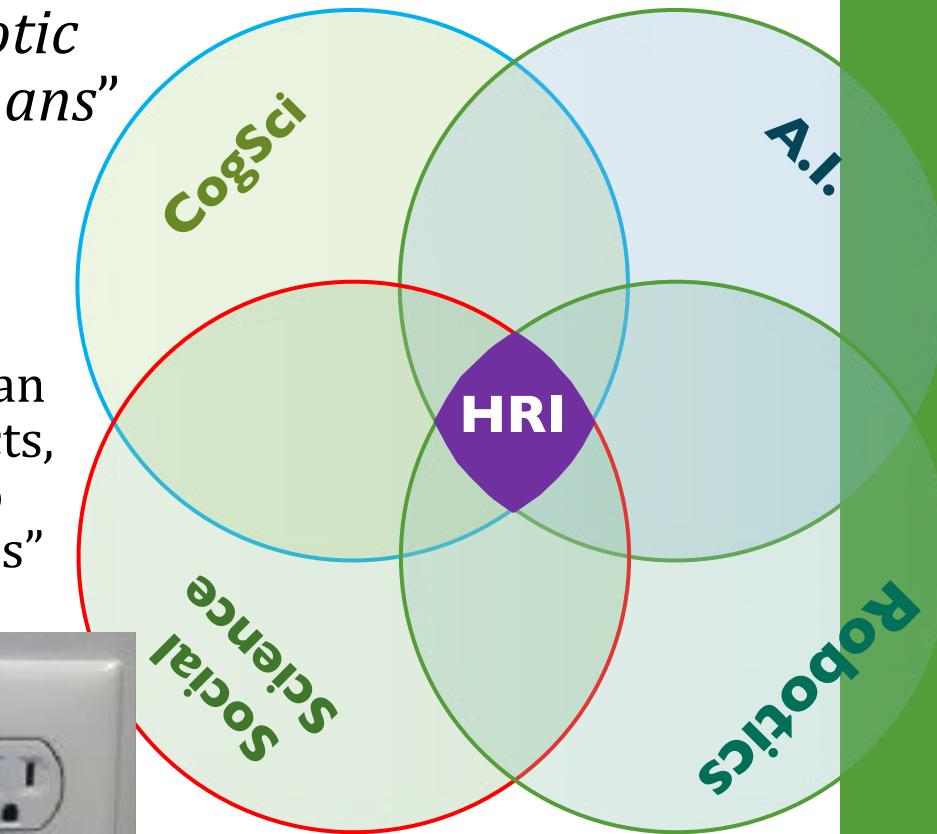
part 2

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

mhanheide@lincoln.ac.uk

last Week: Intro to HRI

- HRI is “*dedicated to understanding, designing, and evaluating robotic systems for use by or with humans*”
- Anthropomorphism as an important factor:
Is the “tendency to attribute human characteristics to inanimate objects, animals and others with a view to helping us rationalise their actions”
(Duffy, 2003)
- The Uncanny Valley



From (Baxter et al, 2016)

Modes of Interaction

Proximate

- The human and the robot are in the same space
- Physical and social interactions fall in this category
 - E.g. service robots



Remote

- The human and the robot are in different locations
 - Maybe even temporally removed
- E.g. teleoperation, supervised control, ...



Two types of Interaction

Explicit

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

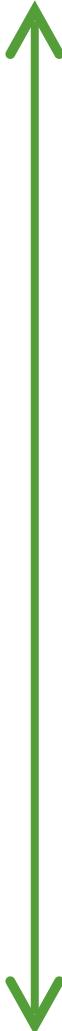
Implicit

- Actions are modified due to presence of an other, but no reaction is expected
- Navigation: e.g. avoiding humans in the way

Capacity for overlap between Explicit and Implicit: e.g. avoiding humans, but engaging in some strategies to encourage humans to move out of the way (more today!)

Levels of Autonomy

Teleoperation



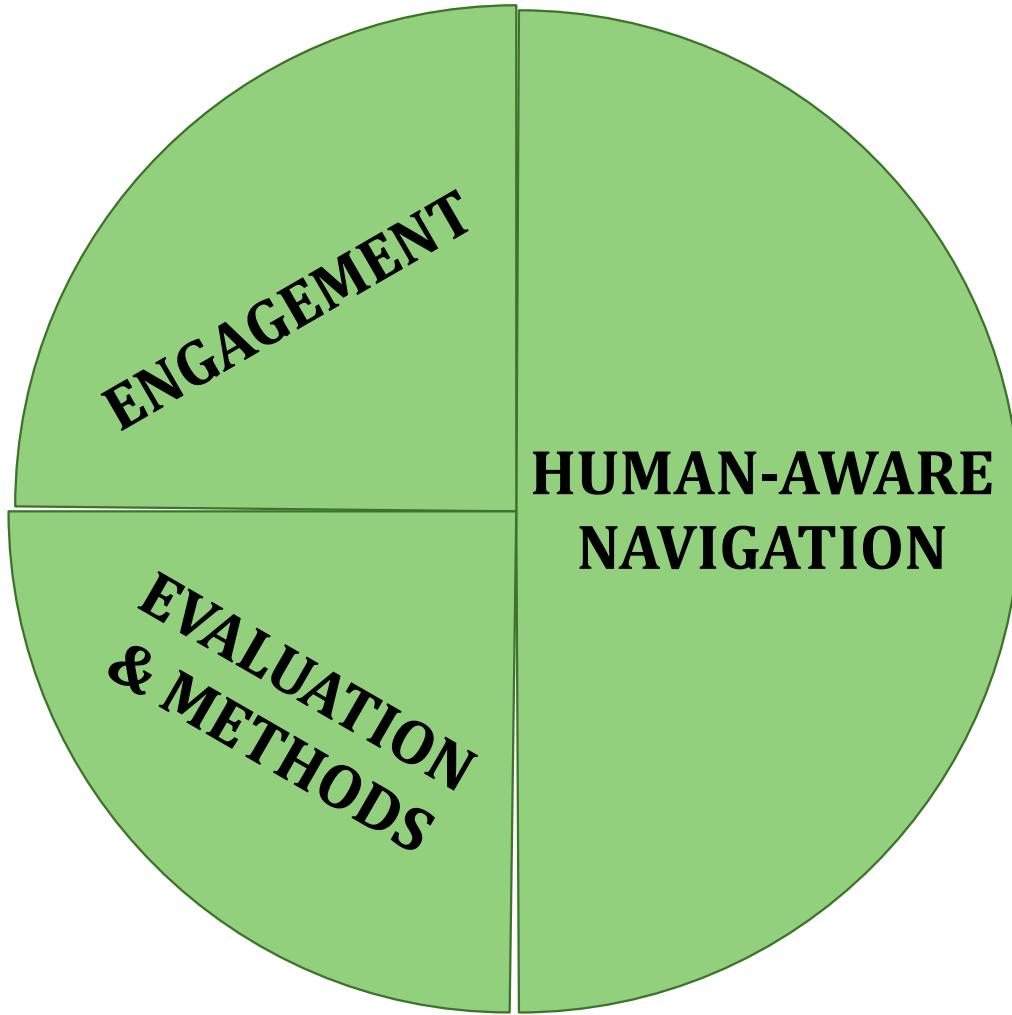
1. Wizard of Oz
 - Teleoperation
2. Scripted Robot Behaviour
 - Operator runs pre-scripted behaviours
3. Partial Autonomy
 - Hybrid autonomous/controlled system
4. Supervised Autonomous Behaviour
 - System acts autonomously, but is supervised, with human take-over in uncertainty
5. Full Autonomy
 - No human intervention required

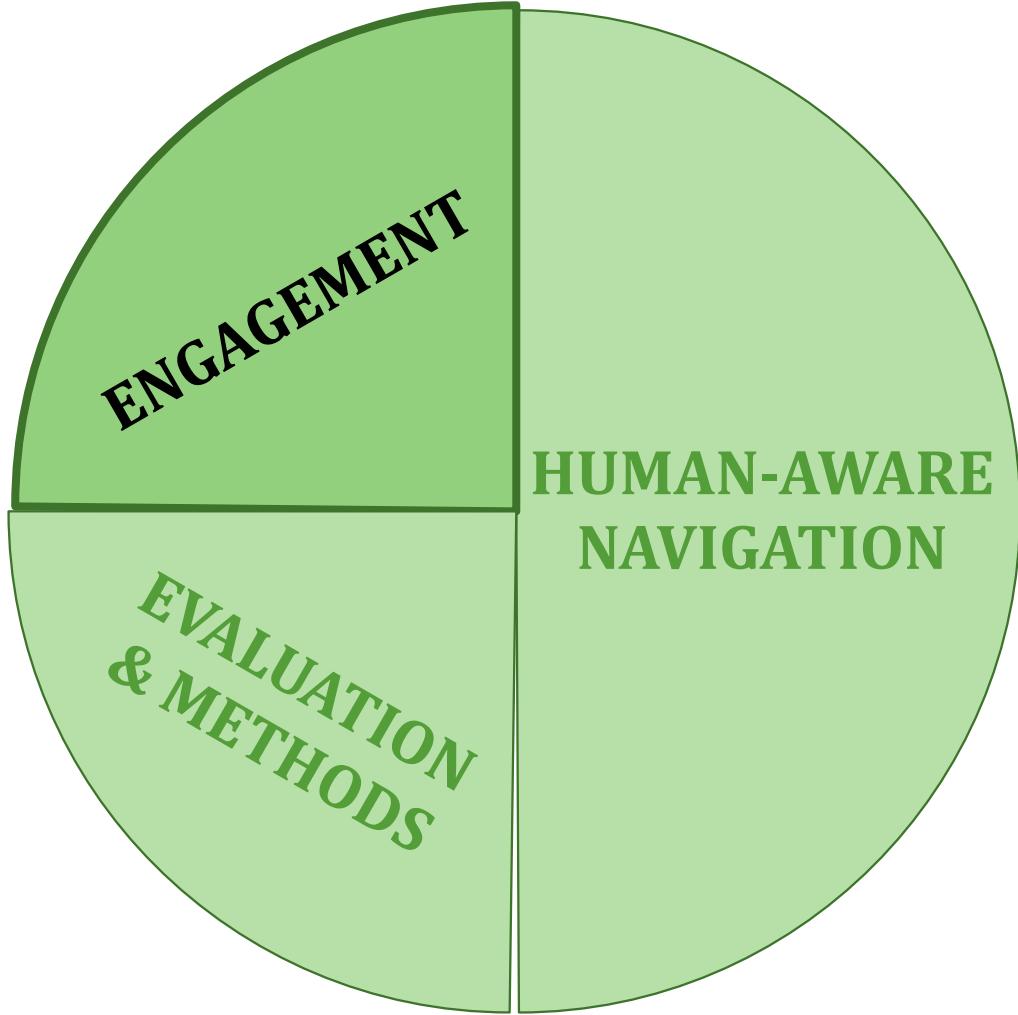
Robot Roles and Ethics

- Robot could take on a number of roles in an interaction, both social and non-social:
 - Learning peer / tutor
 - Therapy assistant / tool
 - Remote presence
 - Team member
 - ...
- Ethical issues can arise in each of these:
 - Deception
 - Attachment (particularly, but not just, with children)
 - There are consequences for technical implementation: e.g. data protection and deletion



Today



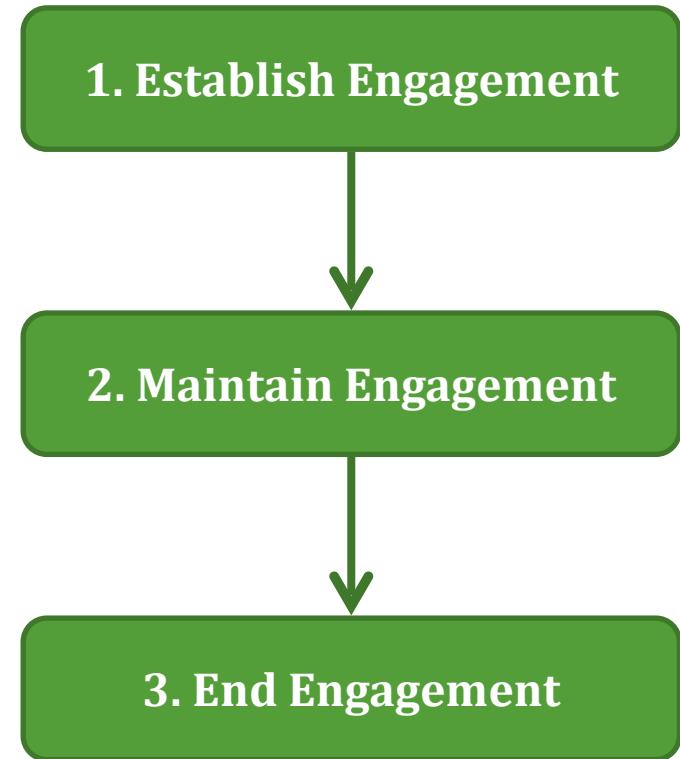


What is engagement?

- Many different definitions...
See e.g. Glas & Pelachaud, 2015 for an overview of these
- Sidner et al 2005:
“...the process by which two (or more) participants establish, maintain and end their perceived connection during interactions they jointly undertake.”
- It encompasses all types of behaviour:
 - Implicit/Explicit
 - Verbal/Nonverbal



Stages in Engagement



1. Establish Engagement

- Attracting attention
- Drawing into an (ongoing?) interaction



1. Establish Engagement

Social Strategies

- Speech
 - Saying something to someone
- Gesture
 - Only useful if have limbs...
 - Waving, etc
- Behaviour
 - Could engage in attention seeking behaviour
- Physical Interaction
 - Equivalent of a tap on the shoulder... (clearly safety implications)

Non-Social Strategies

- Sound
 - Pre-recorded speech
 - Alarm / beeps / motor noise
 - White noise
- Vision
 - Flashing lights
- Behaviour
 - Bumping into human (not advisable...)



Video from:

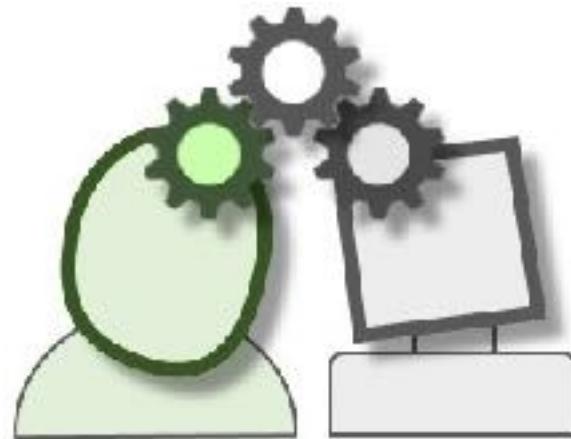
<https://www.youtube.com/watch?v=asHaWo04mIo>

2. Maintain Engagement

- Recall that we are at least partially relying on:
 - Anthropomorphism: appearance and behaviour
 - Attribution of agency

This can only get you so far!

- The necessity for going beyond this with “deeper” complexity and adaptivity of behaviour
 - Refer to Control Architectures Lecture, Week 9
 - This also underlies part of the motivation for incorporating Cognitive Architectures into HRI systems



See: <https://sites.google.com/site/cogarch4socialhri2016/>

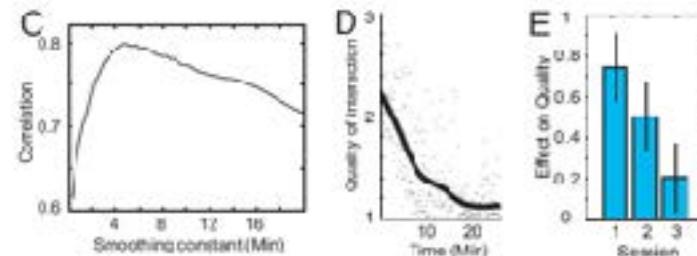
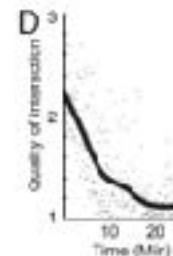
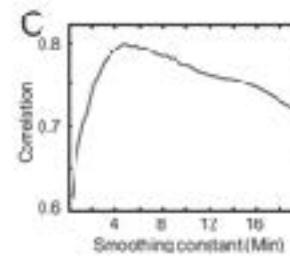
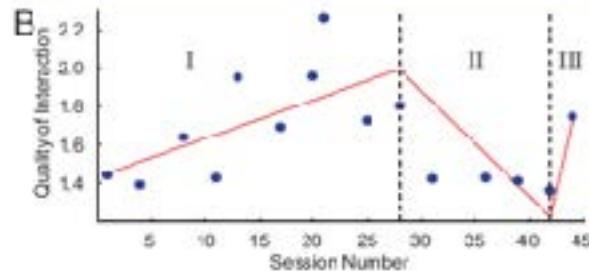
3. End Engagement

- Ending the interaction, and hence any engagement in the interaction
 - Possibly to be resumed at a later time
- Task related:
 - Joint task has completed successfully/unsuccessfully
 - Task no longer relevant
- Personal related
 - Illusion of agency has gone
 - Boredom!



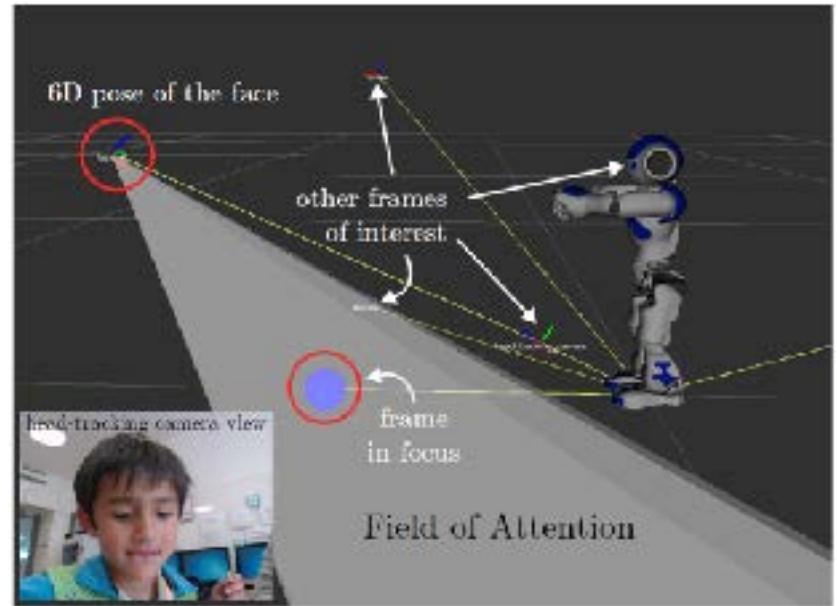
How to detect Engagement?

- How to tell whether someone is engaged in an interaction?
 - And, ideally, how to do so automatically?
 - What should you examine?
- This is a difficult task!
- Humans have an intuitive sense of engagement, based on extensive experience:
 - Developed from baby onwards
 - Can take advantage of this
 - E.g. Tanaka et al (2007): rating using a 'slider'



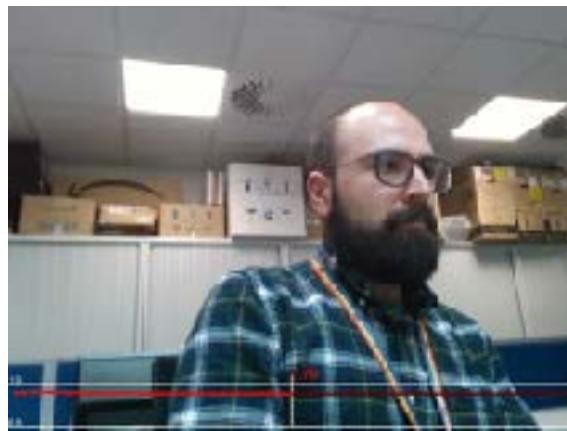
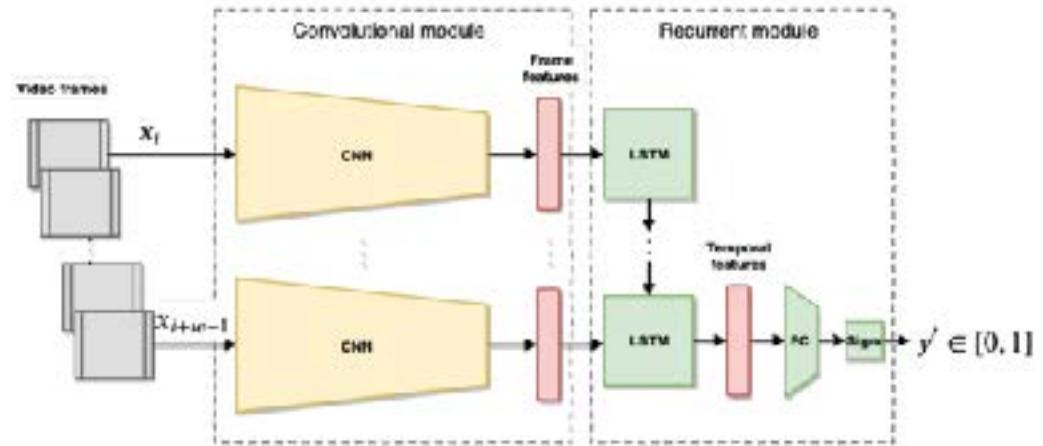
How to detect Engagement?

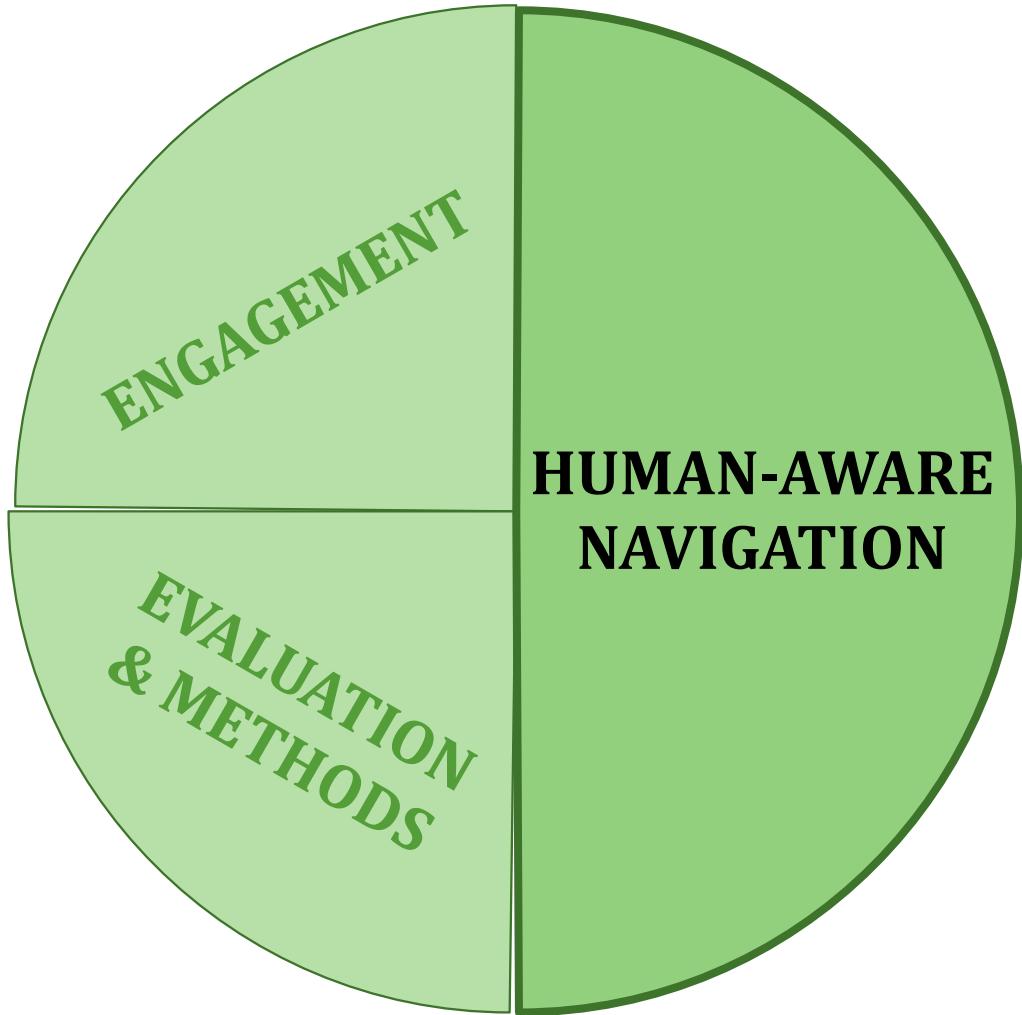
- Gaze is often used as an indicator of engagement
 - E.g. Baxter et al, 2014
- The problem is that this is often post hoc analysis
 - I.e. not in real-time, but only afterwards
- Recent developments trying to achieve this in real-time, using robot sensors:
 - E.g. the GAZR gaze estimator, which can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



How to detect Engagement?

- Rather than using a model (gaze), use machine (deep) learning
 - E.g. del Duchetto et al, 2020
- Engagement is modelled as a continuous value between 0 and 1





Will I bother here?

- A robot anticipating its influence
on pedestrian walking comfort -

“Will I bother here? - A robot anticipating its influence on pedestrian walking comfort”, HRI 2013, Tokyo, Japan

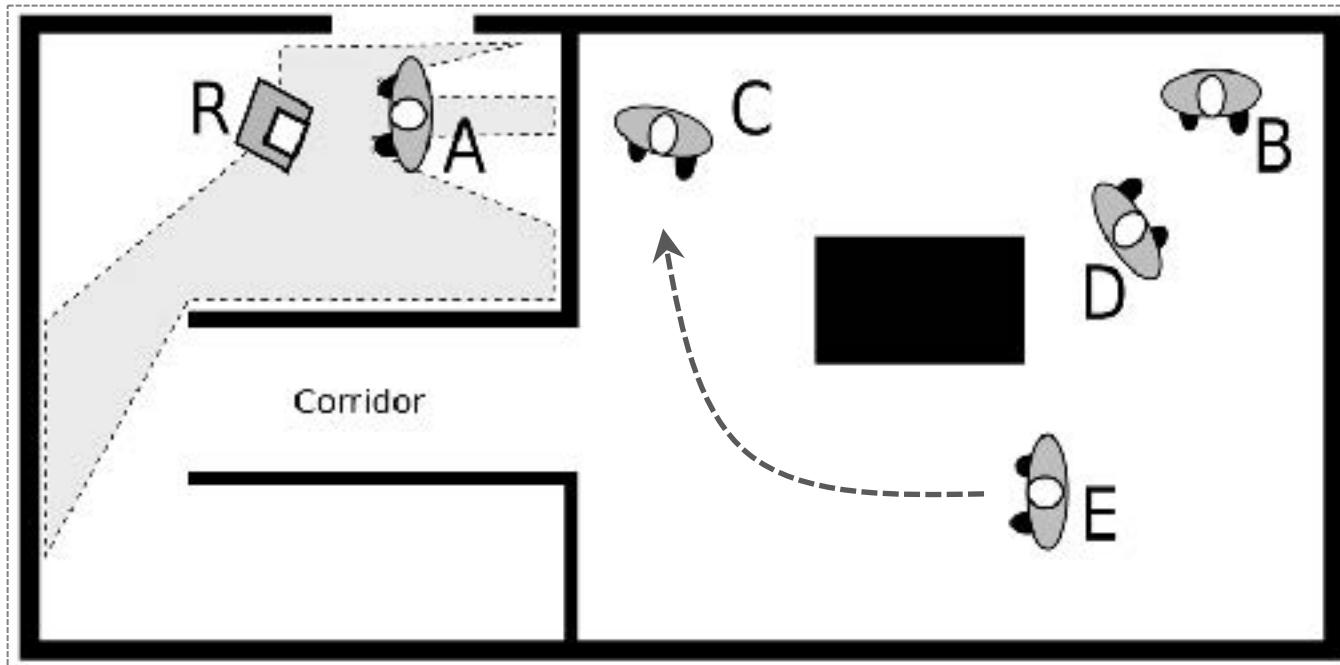
Paper: <http://ieeexplore.ieee.org/abstract/document/6483597/>

Human-Aware Navigation

- Autonomy for mobile robots requires navigation capabilities
- Obstacle avoidance clearly required
 - Preventing robot damage
 - Human safety!
- Goals (Kruse et al, 2013):
 1. Comfort: absence of annoyance and stress for humans
 2. Naturalness: similarity of robot behaviour to humans
 3. Sociability: adherence to high-level cultural constraints
- May be necessary to reduce efficiency (in terms of speed/distance to goal) in the service of these goals

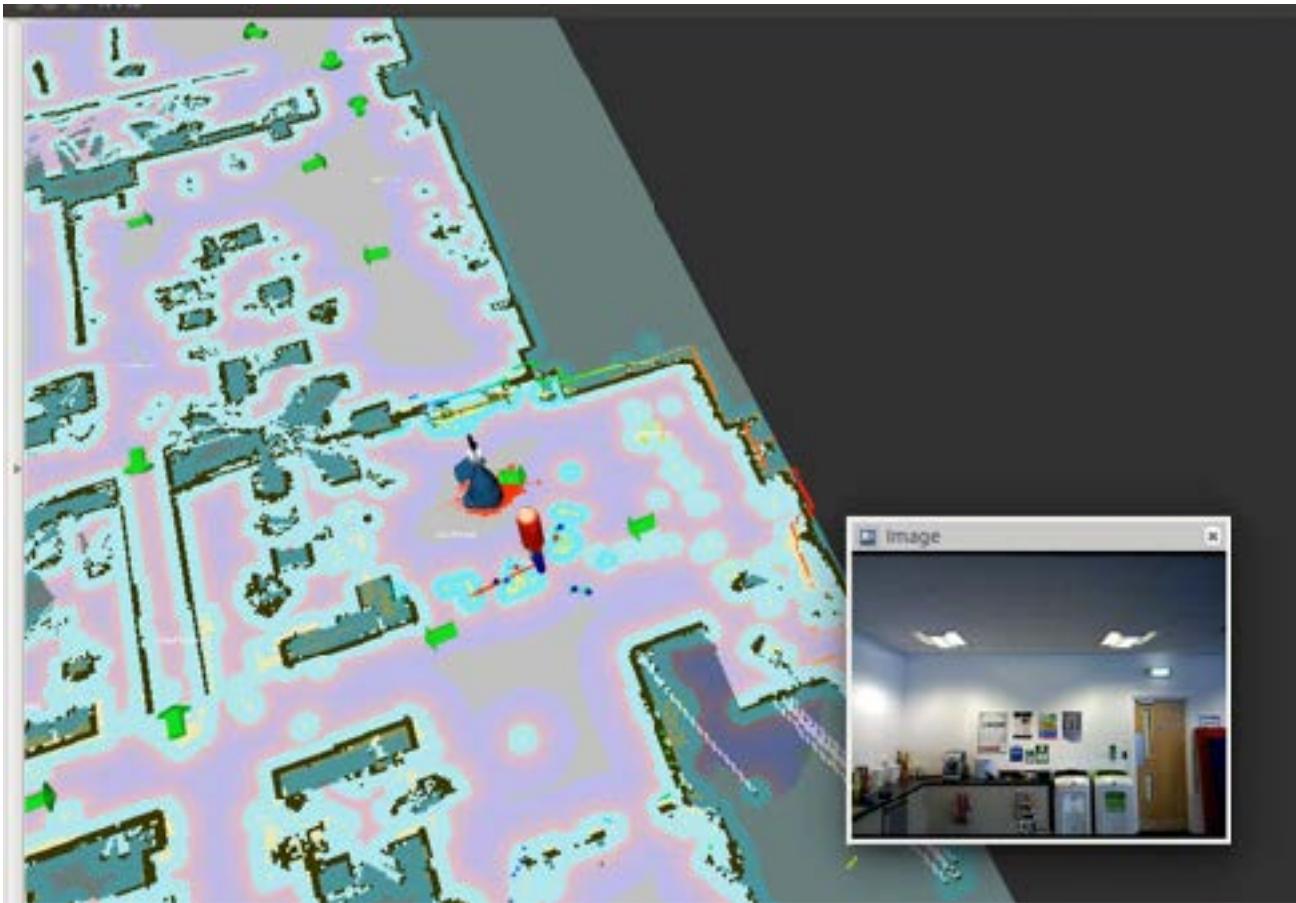
An example

From Kruse et al, 2013:



How should the robot guide person A to person B?

Humans are Awkward Obstacles...



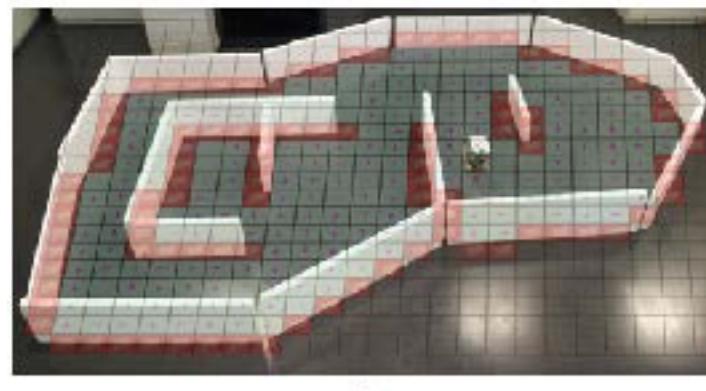
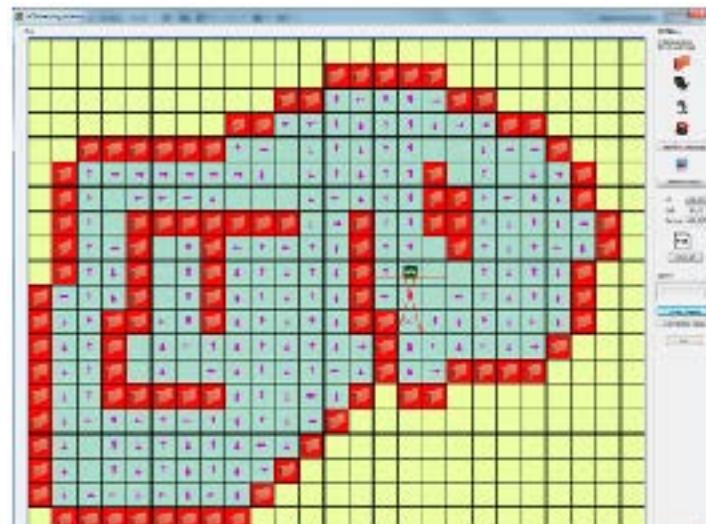
Also see this L-CAS video: <https://www.youtube.com/watch?v=zdnyhQU1YNo>



(a) Navigation and Costmaps

Recap: path planning

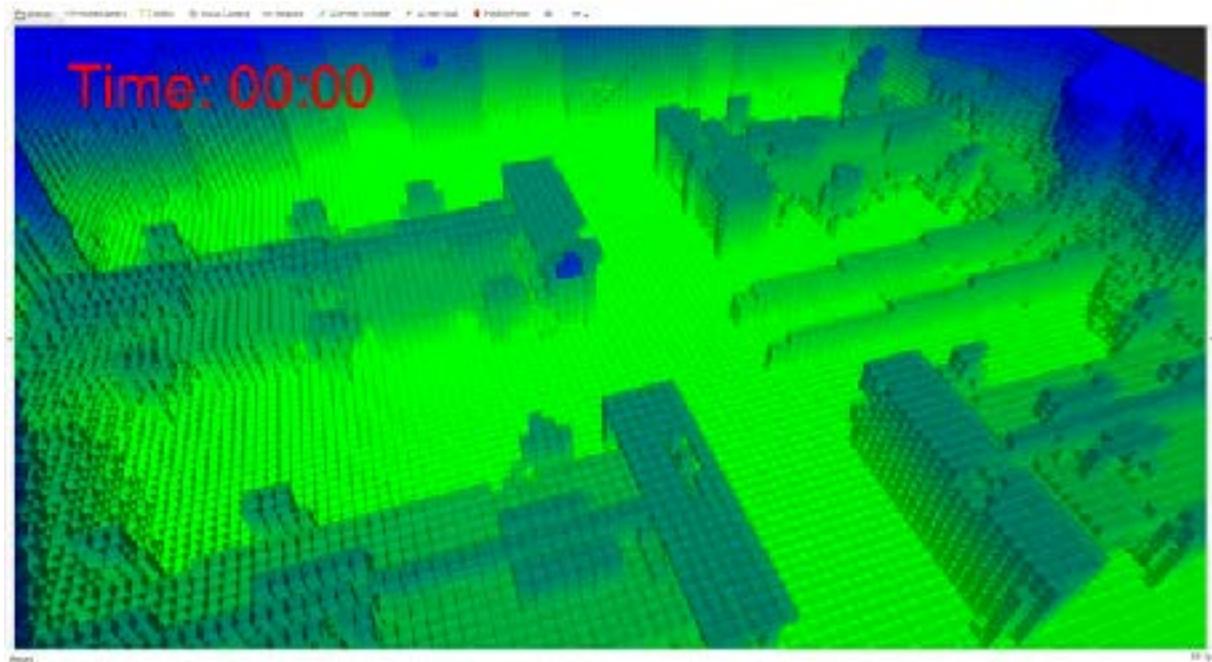
- Refresh your memory of Lecture Week 7: Navigation 1
 - Dijkstra and A*
 - Include movement cost into node
- Application to discrete states, in this example a network of nodes
- Equally applies to a 2D space discretised in a grid
 - “Occupancy” grid
 - Image from (Gonzalez et al, 2013)



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

An aside: 3D discretisation (voxels)

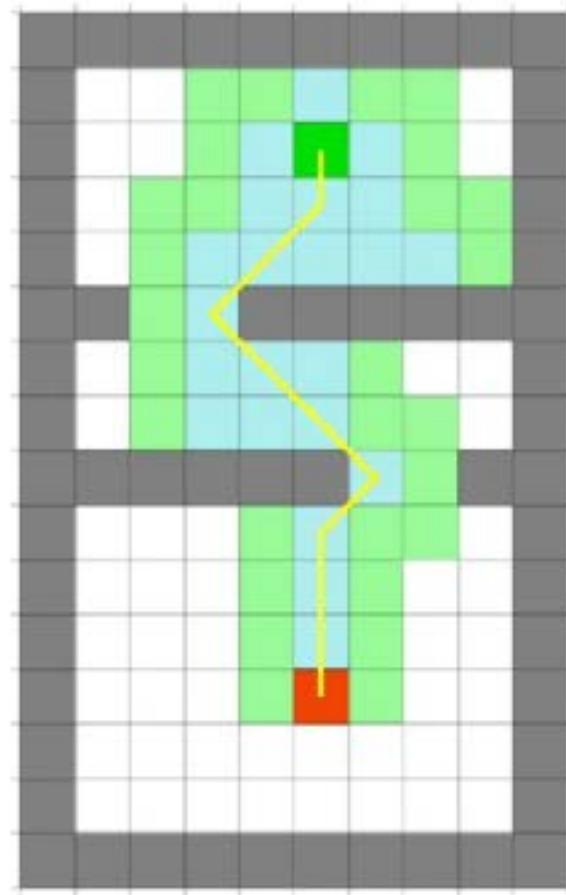
- Regular grid in 3D space
- Can use to discretise a 3D space in a similar way as done for 2D spaces



Video: https://www.youtube.com/watch?v=CUT3t-ico5Y&list=PLnS6TQ_QsDUxCrv1fbzPHn0LoTwc_n8NF&index=2

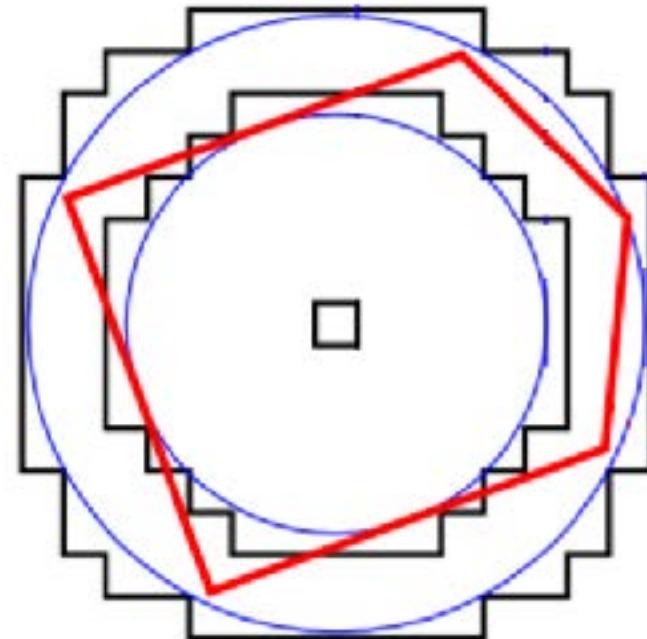
Path Planning

- Have what we need to plan path
 - E.g. Dijkstra
 - However, Dijkstra not ideal:
 - Path close to obstacles
 - Next to walls
 - Not good for robots!
 - Want to keep our robot safe:
 - If in corridor, drive in middle
 - Keep distance from obstacles



Robot Distances and Costs

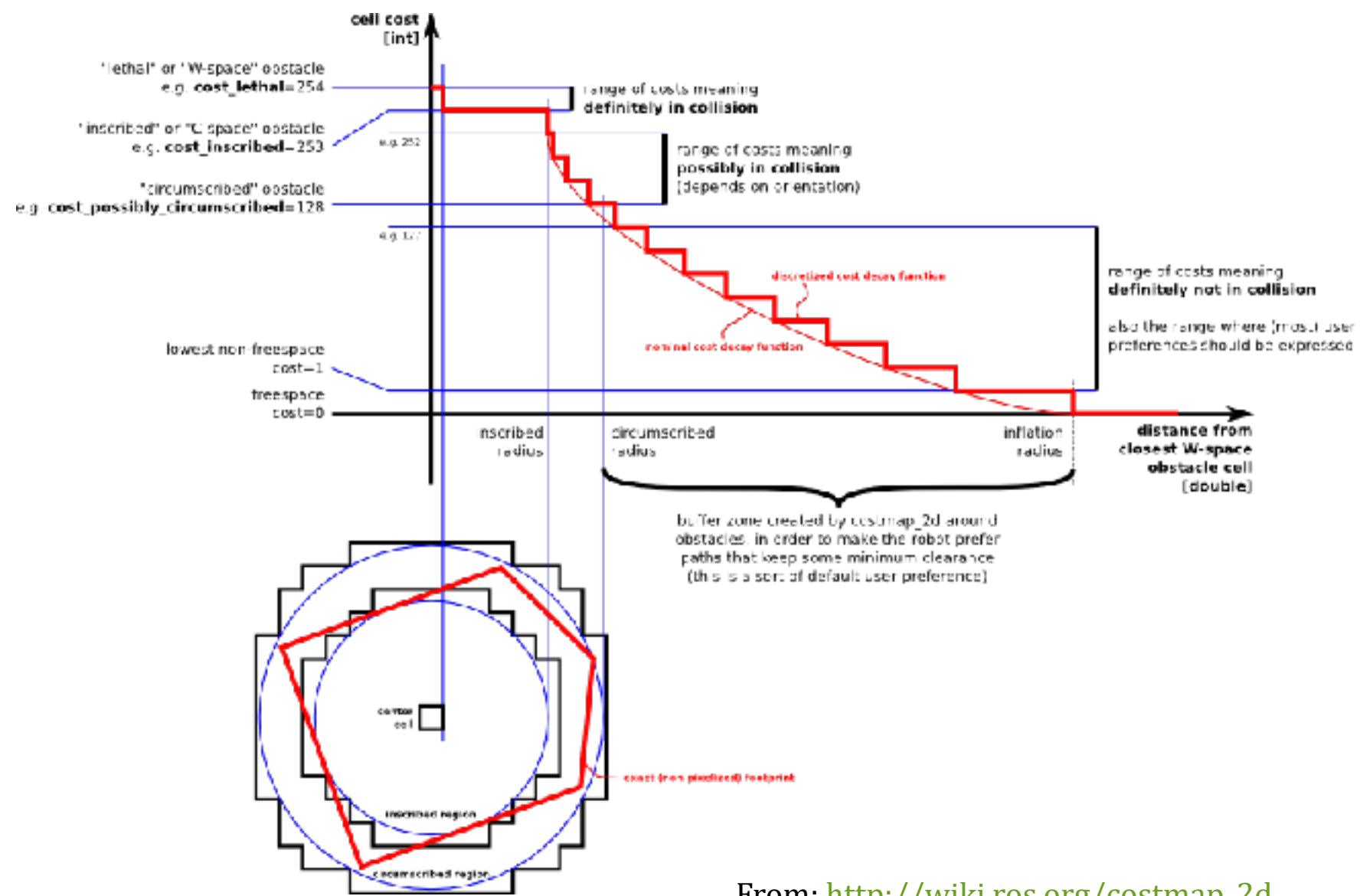
- Robot centre point assumed as point of rotation
 - E.g. the turtlebots
- Obstacles are “lethal”
 - In map
 - Sensed by laser
- Lethal obstacles “inflated” based on the robot size
 - Helps determine distance from obstacles



RED – actual robot footprint

OUTER CIRCLE –circumscribed region

INNER CIRCLE – inscribed region

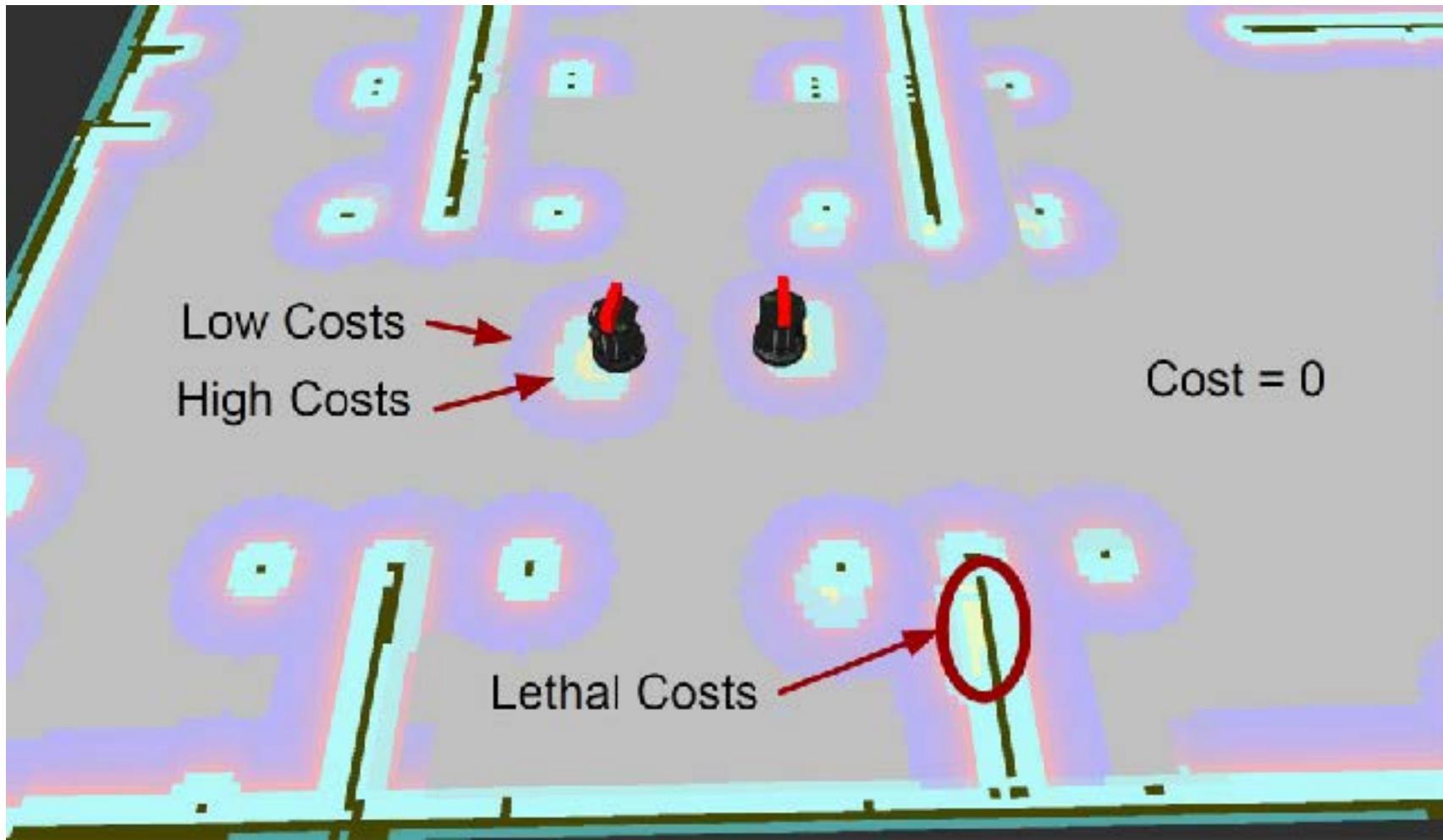


From: http://wiki.ros.org/costmap_2d

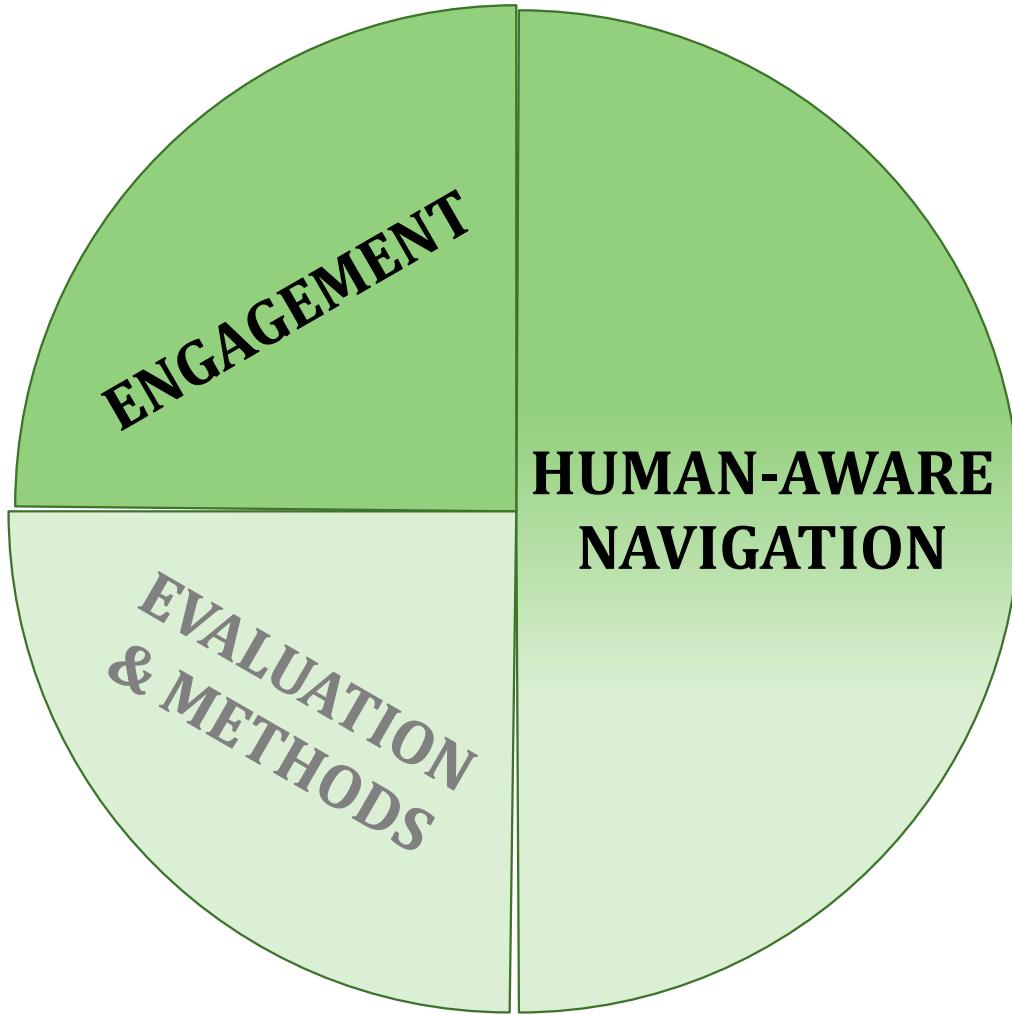
Using these costs

- Costs encoded into the cost-map, for each obstacle
- In Dijkstra example:
 - Using movement as part of the cost (1 for straight, $\sqrt{2} = 1.414$ for diagonal)
 - Now also use the cost of the relevant cells in the costmap to calculate the next neighbour
- Result:
 - For navigation, the robot can stay clear of obstacles, even when using Dijkstra for planning

Costmaps



Before the break...





(b) The Human Obstacle

Humans are not just obstacles...

- Human-Robot Spatial Interaction

The study of joint movement of robots and humans through space and the social signals governing these interactions

- Movement of robots and humans
- Focus on social signals

- Human-Aware Navigation

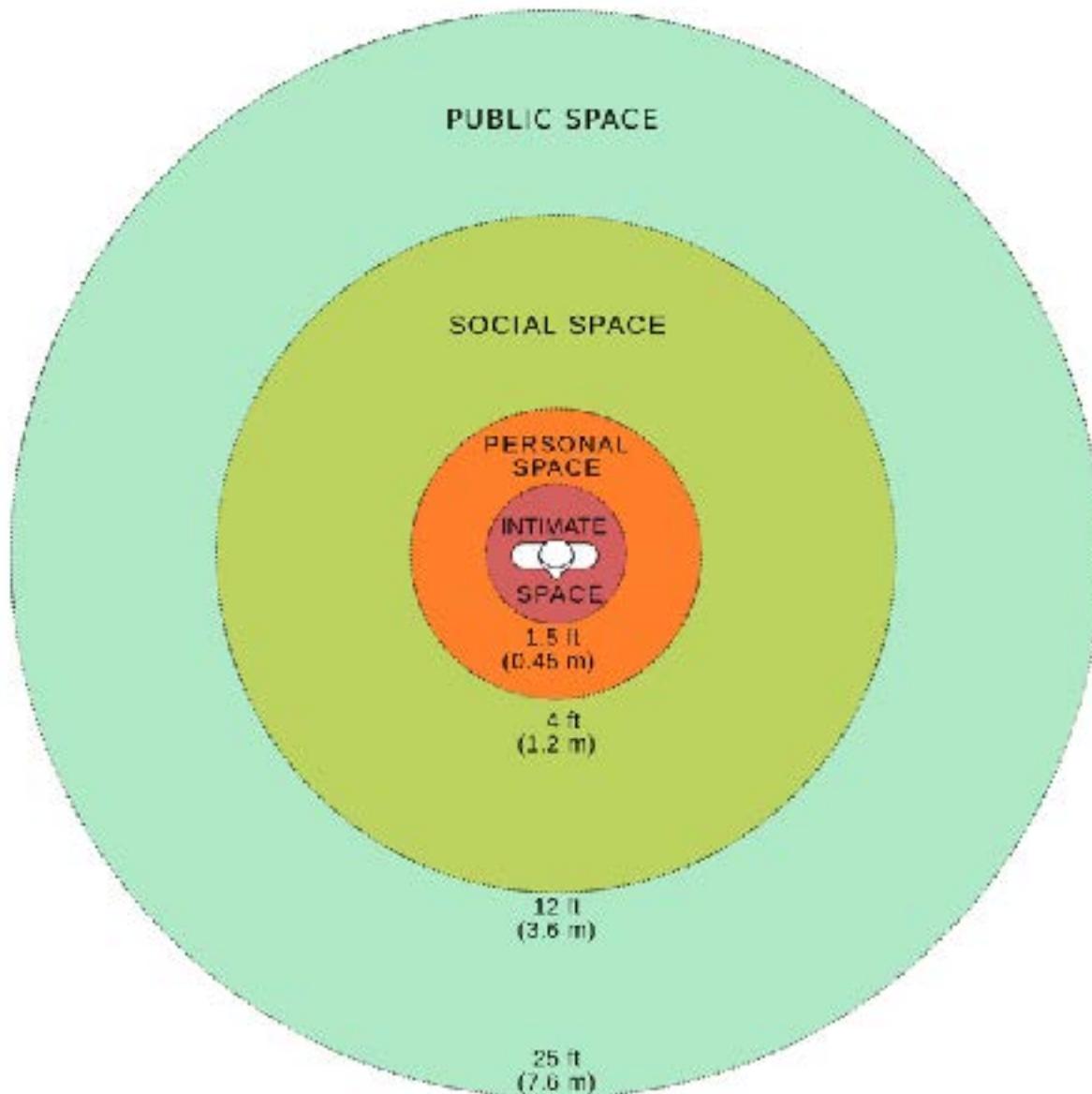
- Specifically taking the human into account
- Recall the three goals: comfort, naturalness, and sociability
- Two main methods:
 1. Stop-and-wait: human does all the hard work
 2. Cost functions based on principles of Proxemics

Proxemics

- A virtual personal space around an individual
 - Edward Hall, 1966
- Divided into four main zones
 - Each zone at a different distance, and with different interaction characteristics
 - Also dependent on relationship
 - Two 'phases' per zone

The four zones:

1. Intimate
2. Personal
3. Social
4. Public



Intimate Space

- 0 – 45cm: Intimacy
- Close Phase (0-15cm)
 - Intimacy, comforting
 - Vision blurred, vocalisations whispered
 - Senses of smell and radiant heat effective
 - Arms can encircle
- Far Phase (15-45cm)
 - Hand can reach and grasp extremities
 - Heads, thighs, and pelvis are not easily brought into contact
 - Able to focus the eye easily, peripheral vision includes the outline of the head and shoulders
 - Heat and odour of the other person's breath might be detected

Personal Space

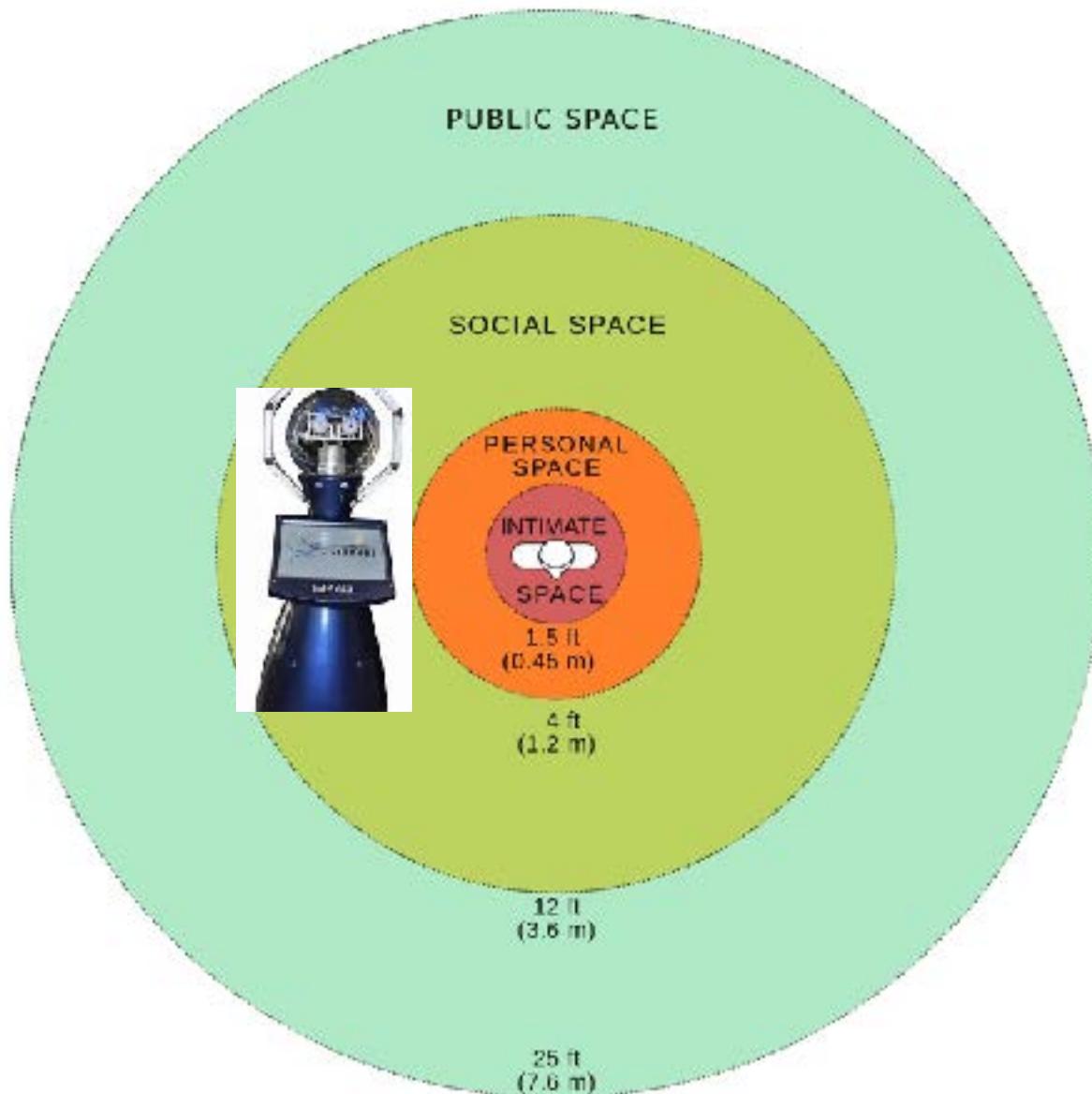
- 45 – 120cm: Family and good friends
- Close Phase (45-75cm)
 - Can grasp other person: peripersonal space
 - No visual distortion of other's features
 - Perception of 3-dimensional qualities of objects
- Far Phase (75-120cm)
 - Outside of grasping distance: at arm's length
 - Other's features clearly visible
 - Moderate voice volume
 - No perception of body heat
 - Lower levels of olfaction

Social Space

- 1.2m – 3.6m: Interactions with acquaintances/strangers
- Close Phase (1.2-2.1m)
 - No touching without special effort
 - Normal voice volume – can be heard from a moderate distance
 - Visual focus extends to nose and parts of both eyes, or, nose, mouth and one eye
- Far Phase (2.1-3.6m)
 - Fine details of face are lost
 - Skin texture, hair, teeth, and condition of clothes readily visible
 - Odour not detectable

Public Space

- > 3.6m: public speaking
- Close Phase (3.7-7.6m)
 - Can take evasive actions
 - Voice loud but not full volume
 - Can see whole face, fine details not visible
 - Only whites of eyes visible
- Far Phase (>7.6m)
 - Subtleties of meaning in voice is lost, as are details of facial expressions
 - Vocal, facial, and bodily expression must be exaggerated
 - Foveal vision takes in increasingly more of the other person



Caveats

- Dependent on culture (Hall's studies were in US)
- Equal spacing around individual
 - No difference between front and back
 - No accounting for movement (e.g. warping of space when walking)
- Do not take into account environmental conditions
 - E.g. dim lighting, loud environments, etc
- Enforced violations?
 - E.g. public transport
 - Changes behaviour
- These zones may be applicable to humans, but do they apply to robots?





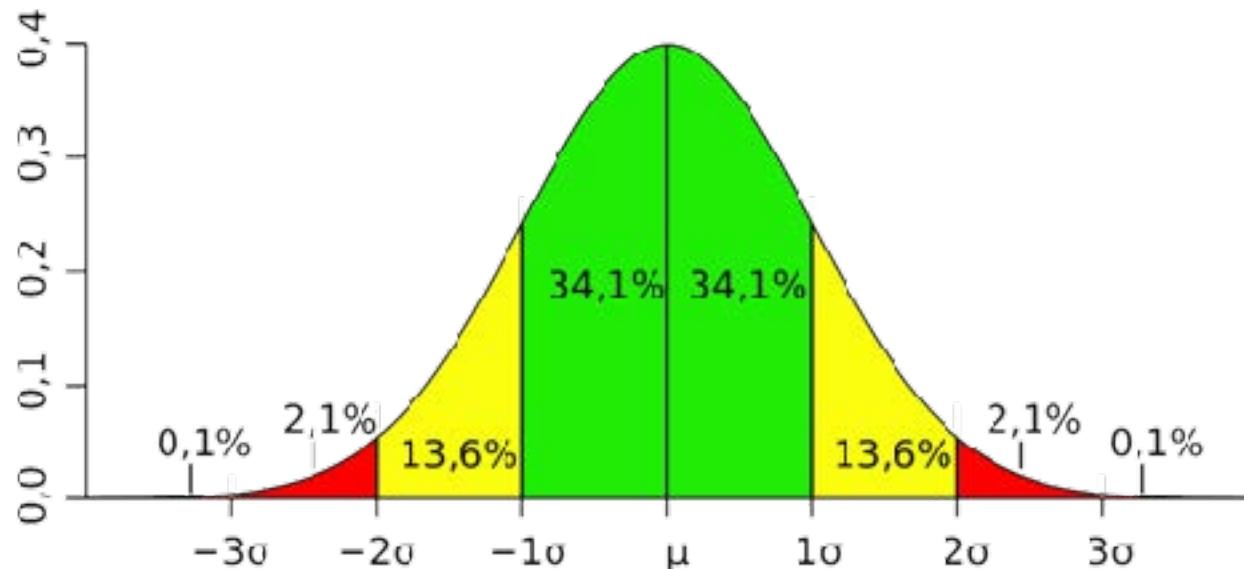
(c) Bringing These Together

Combining Costmaps and Proxemics

- Including costmaps as part of the human representation in the map:
 - Including proxemics as part of this
 - What is the benefit?
- Keeping a greater distance to the human
 - Perceived as safer
 - Reduced stress
 - Even though less “efficient”
- Not necessarily either of the other two goals
 - Doesn’t guarantee more natural behaviour
 - Doesn’t incorporate societal/cultural norms (e.g. drive on the left or the right?)
- How to put Proxemics into Costmap?

Gaussian Distribution and Proxemics

- Sigma – standard deviation (mean of zero)
 - Mean could be position in one dimension (x or y)
- Theoretically infinite, so cut-off at 3-sigma
- Set sigma to size of the Intimate Space
 - This is in only one dimension...
 - For quick visualisation: <http://homepage.stat.uiowa.edu/~mbognar/applets/normal.html>

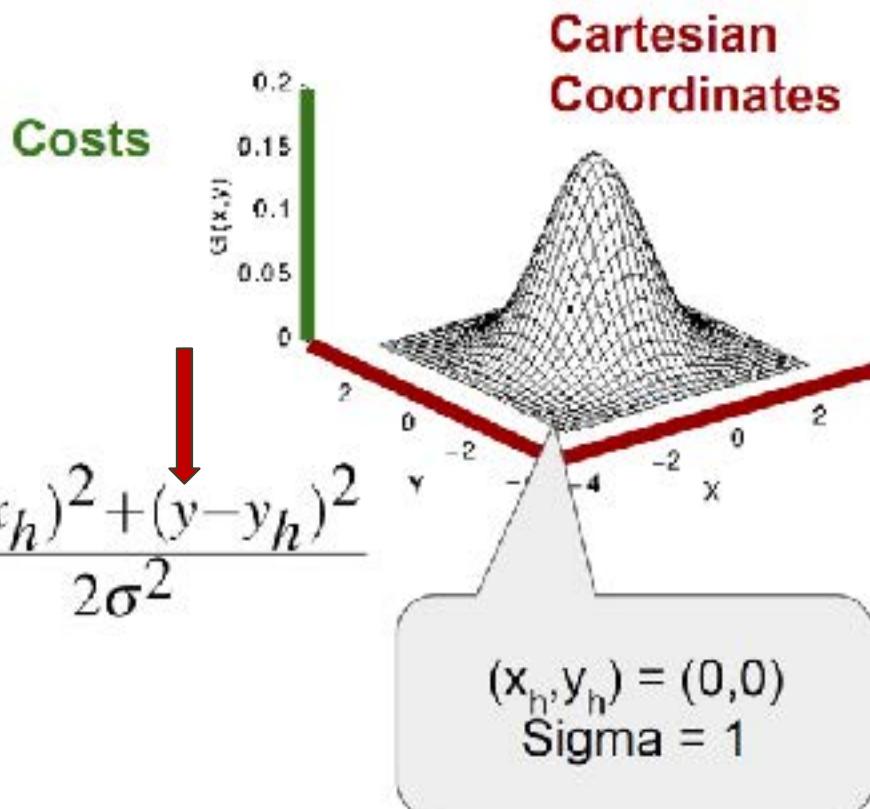


Extending to 2D...

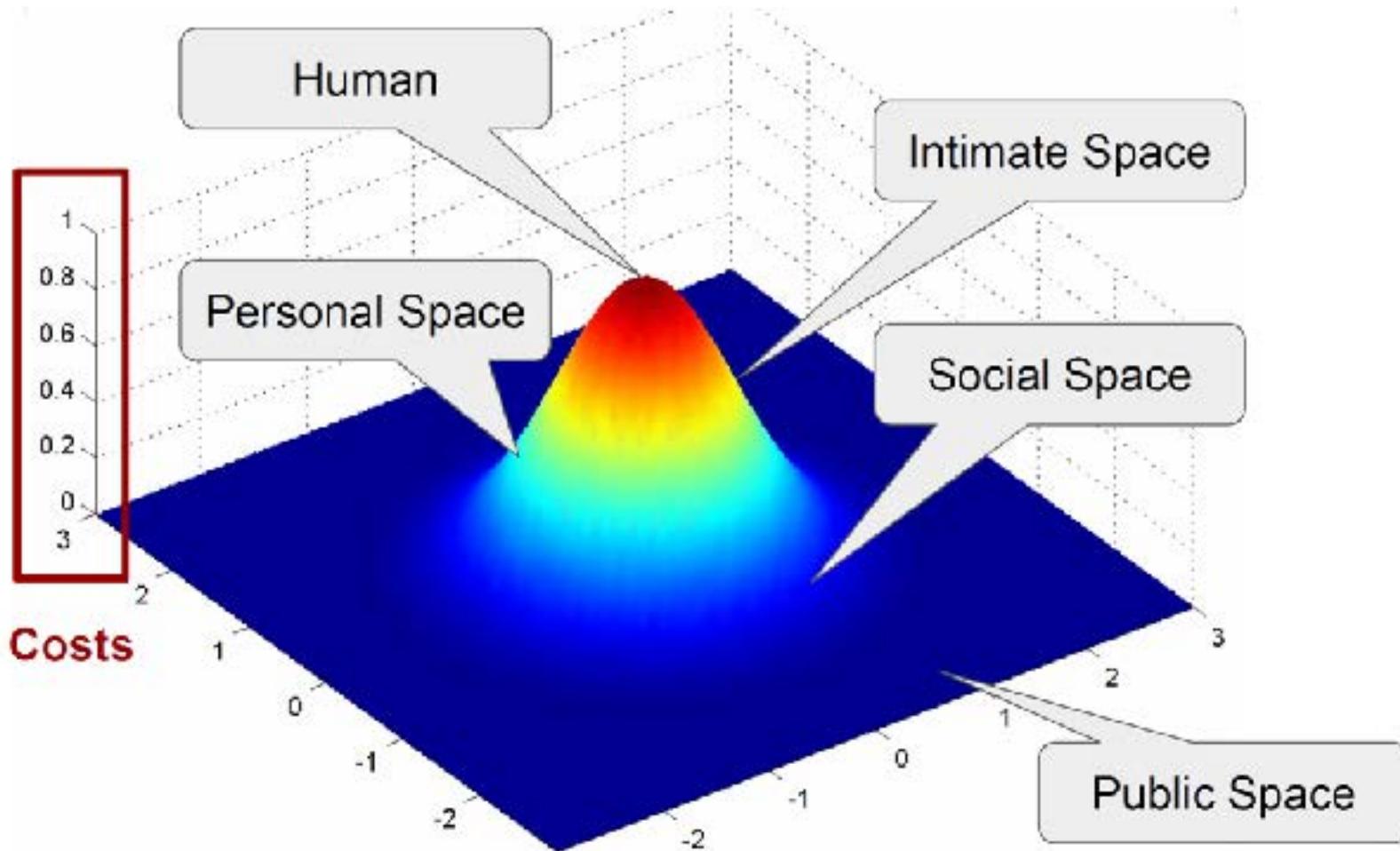
- 2D Gaussian equation – example parameters shown
- Cartesian coordinates centred on the position of the human

Height of the peak

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

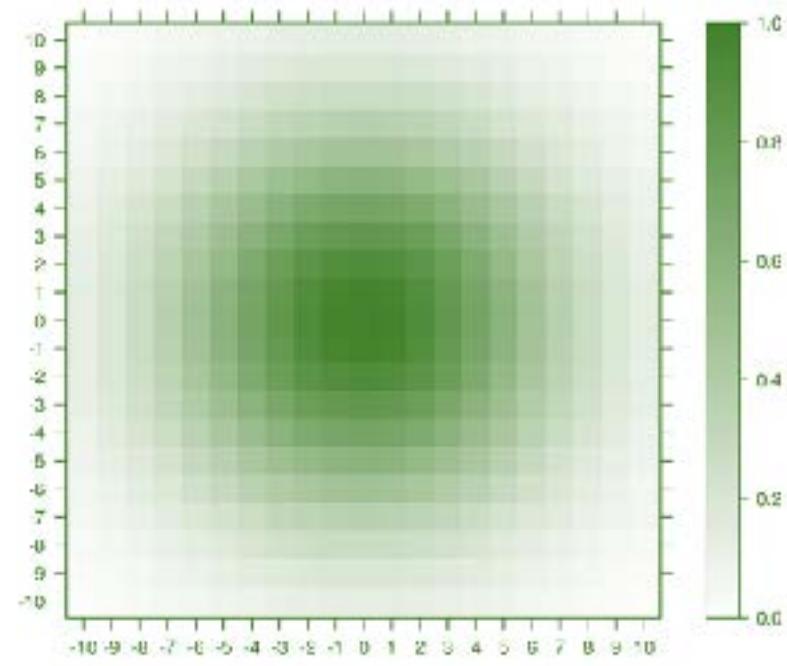
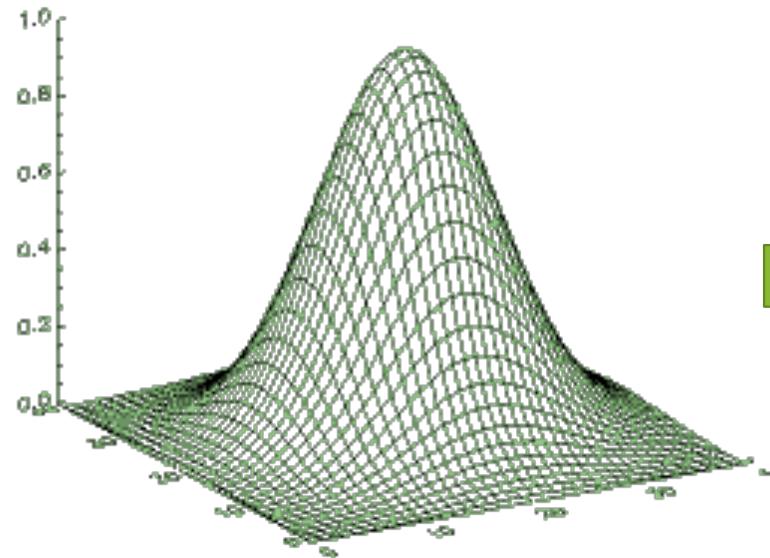


Proxemic Gaussian



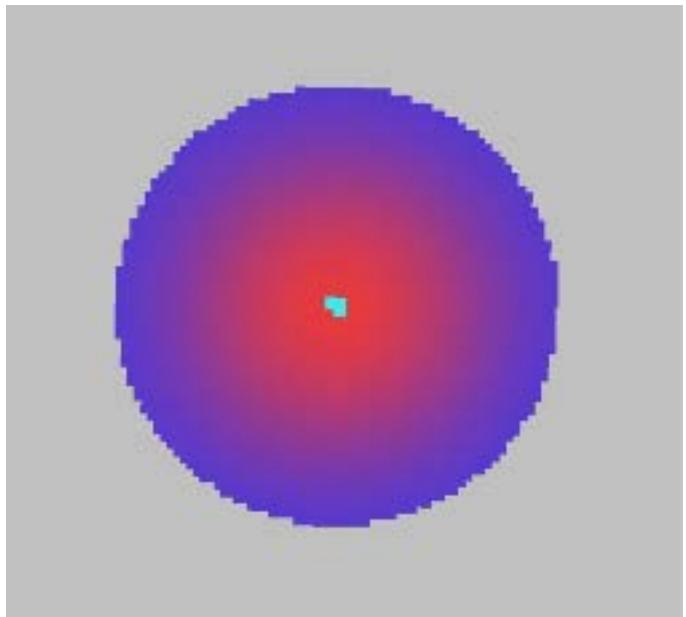
Discretisation

- Have a continuous function, need it to be discretised

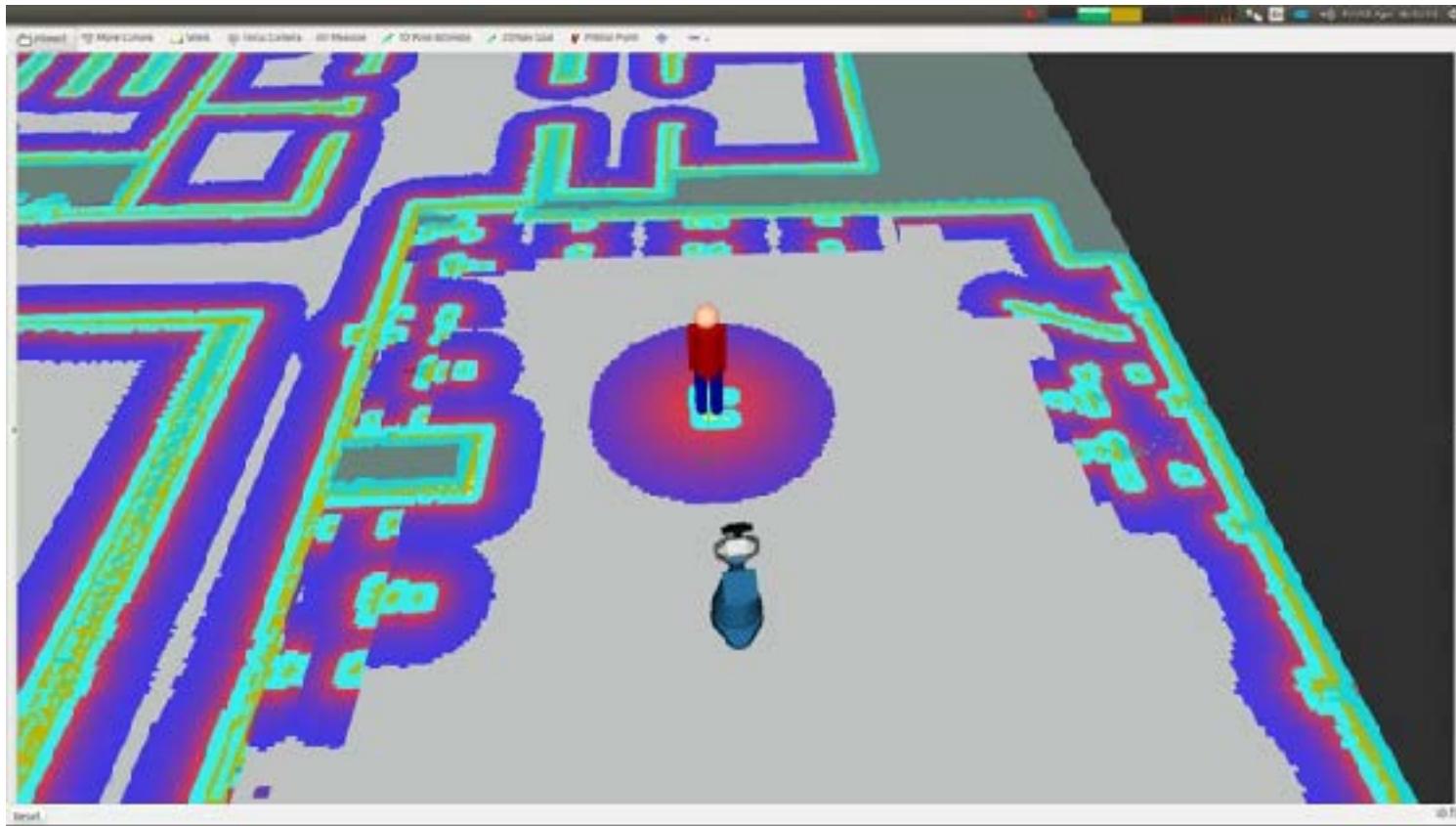


And back to Costmaps...

- This information can now be added to the overall costmap, with the other obstacles
 - Or added to a separate layer of the costmap (Lu et al, 2014)
- Path planning in this space as described before
 - Dijkstra

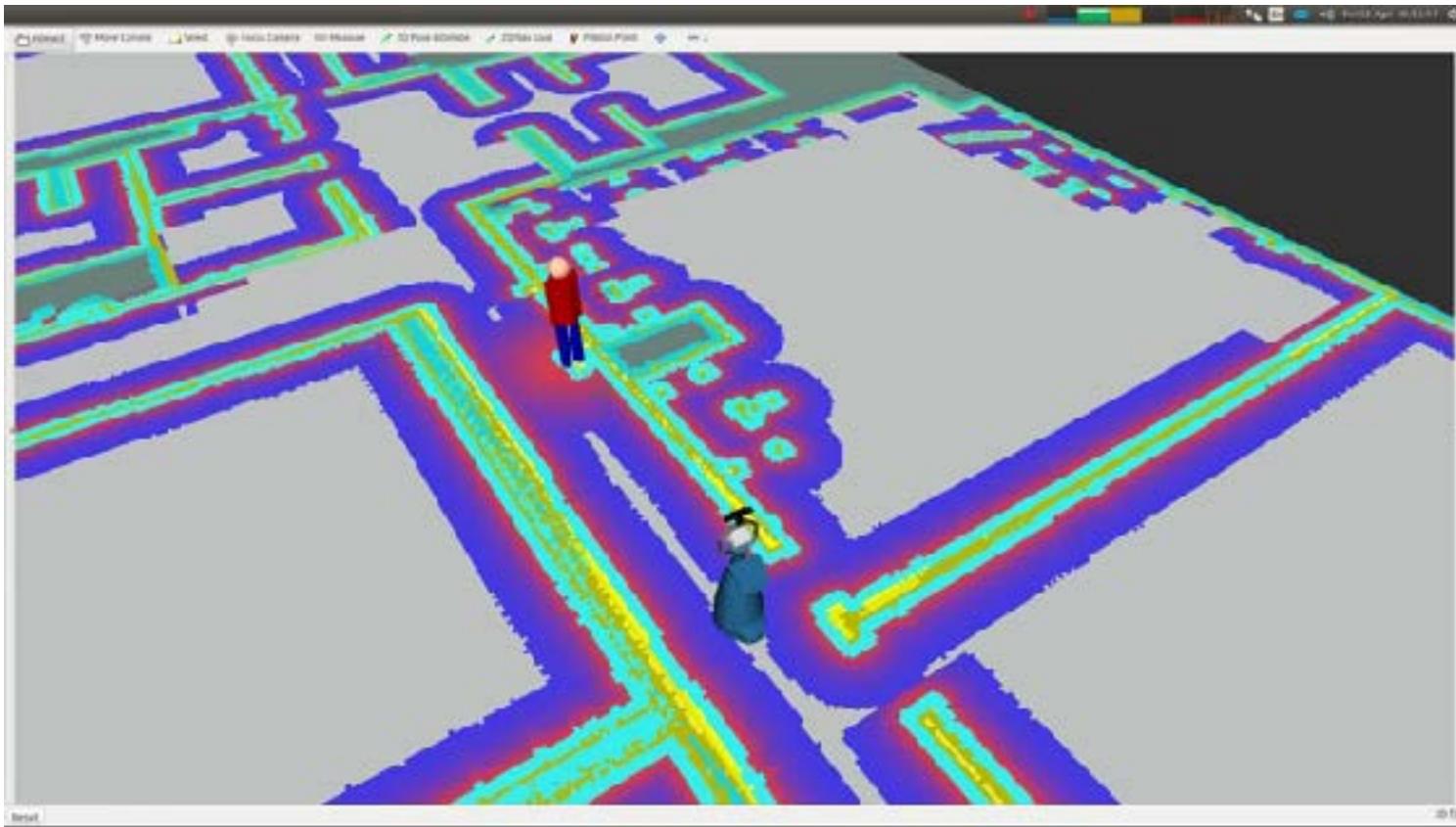


Navigation in Open Space



Full video: <https://www.youtube.com/watch?v=pg7g7qv80MU>

Navigation in Corridor



Full video: <https://www.youtube.com/watch?v=pg7g7qv80MU>

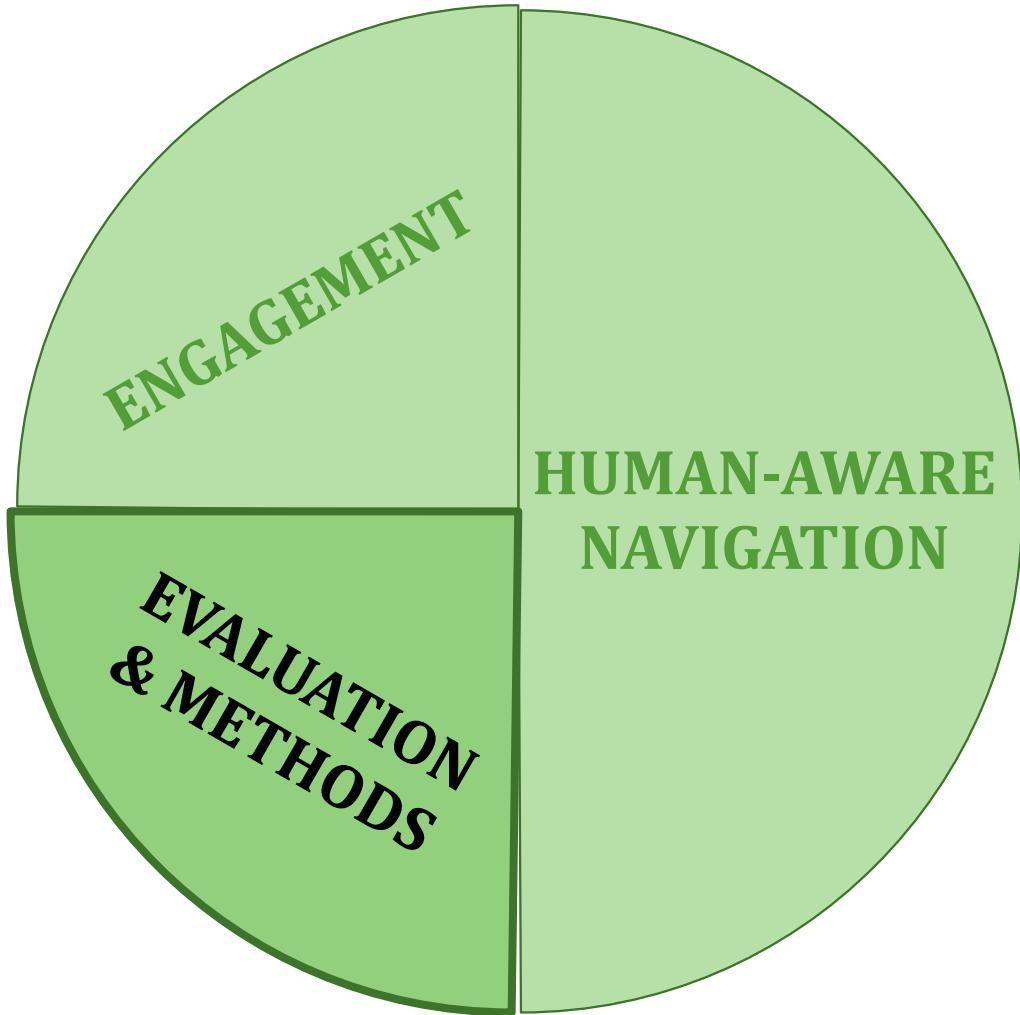
Gaussian Cost functions

Positives

- Straightforward implementation
- Is relevant for all environments
- Takes into account proxemics
- Ensures interaction is safe, and perceived to be as such

Negatives

- Only influences distances
- Sociability and naturalness not guaranteed
- Does not take into account social context
 - Only based on proxemics
 - Could drive through groups?
- Works with Dijkstra
 - Slow/inefficient



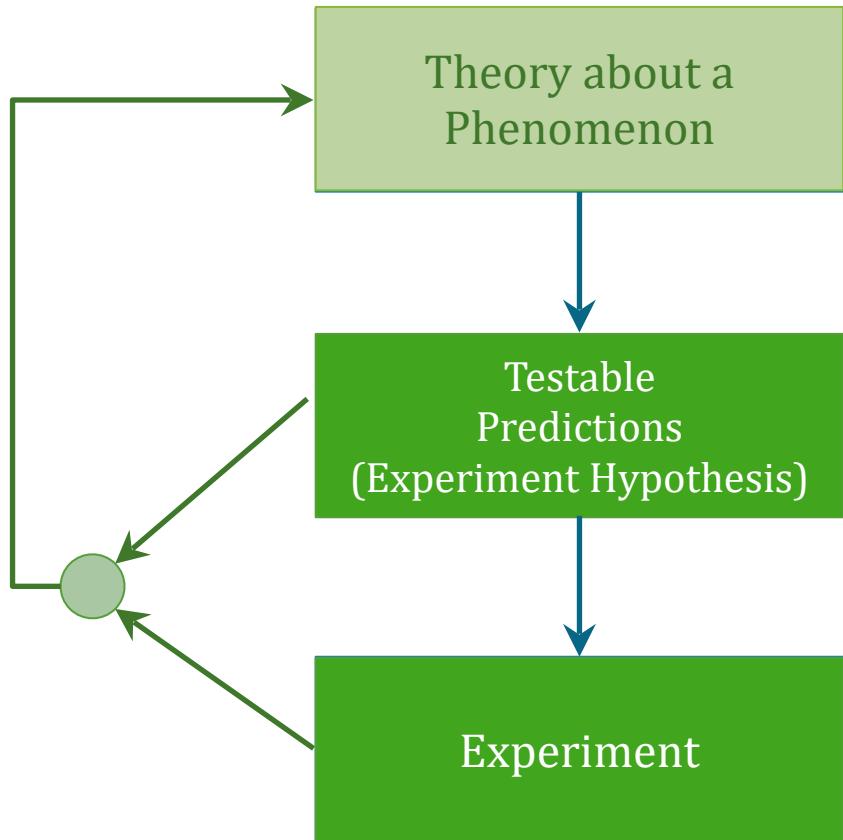
Is my Robot Successful?

- You have implemented a complete robotic system – how do you know if it is successful?
 - Performance metrics (remember your assignment!)
 - Test with real robots/simulation, etc
- But what if it is an HRI system?
 - Humans are problems...
 - Non-predictable, they come with prior expectations/experience
- Need to evaluate your system with people...
 - Running experiments
 - To verify that the robot has an effect (or is perceived) in the way you (as a designer) intended

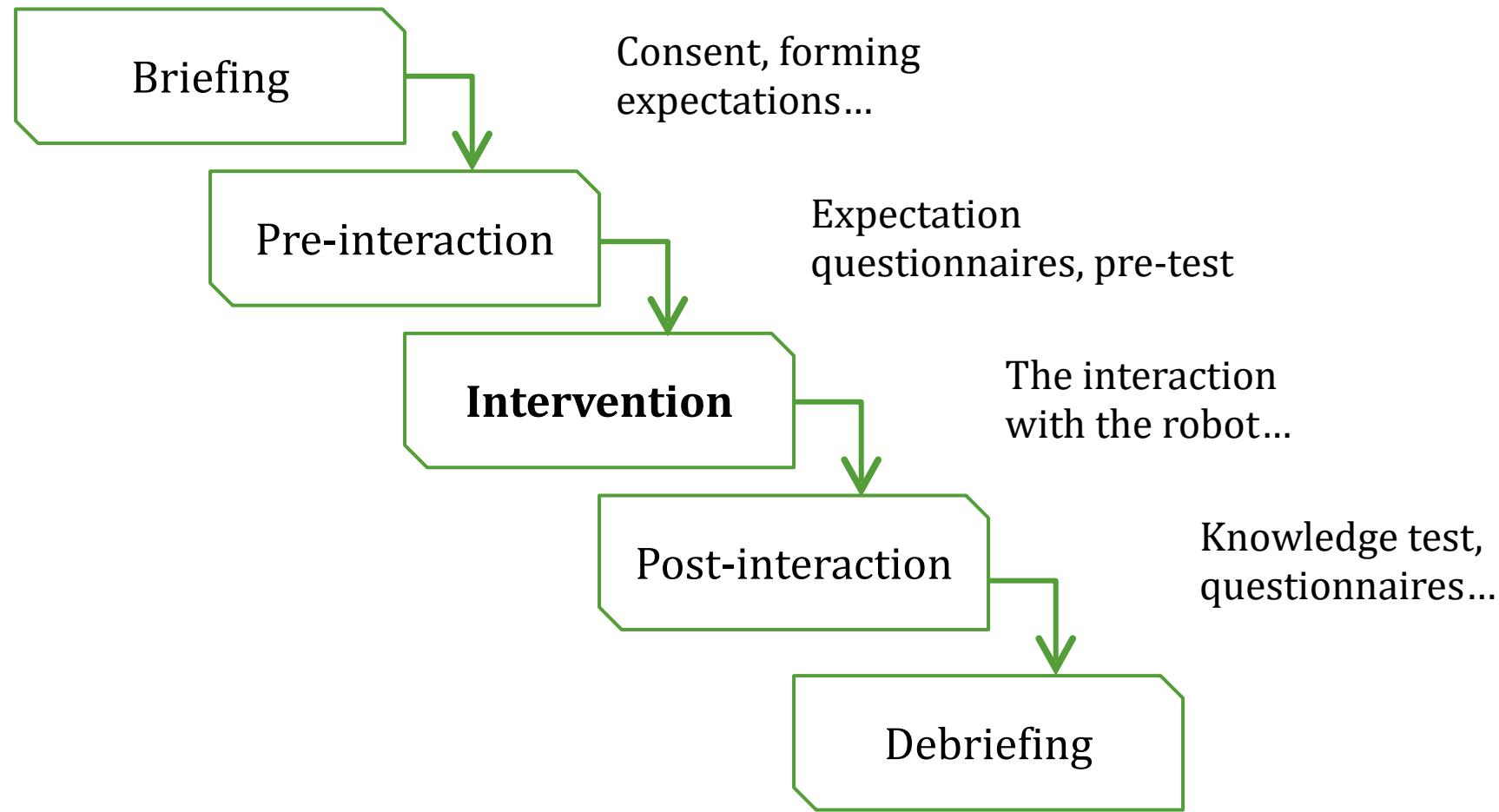


The Scientific Method

- The “classic” view of the scientific method
 - Generate testable predictions from a theory
 - Perform experiment specifically generated to address the hypothesis
 - Assess the experimental hypotheses: observations in context of the theory
 - Update/refine the founding theory as necessary
- Does not deny role for exploratory studies (later)

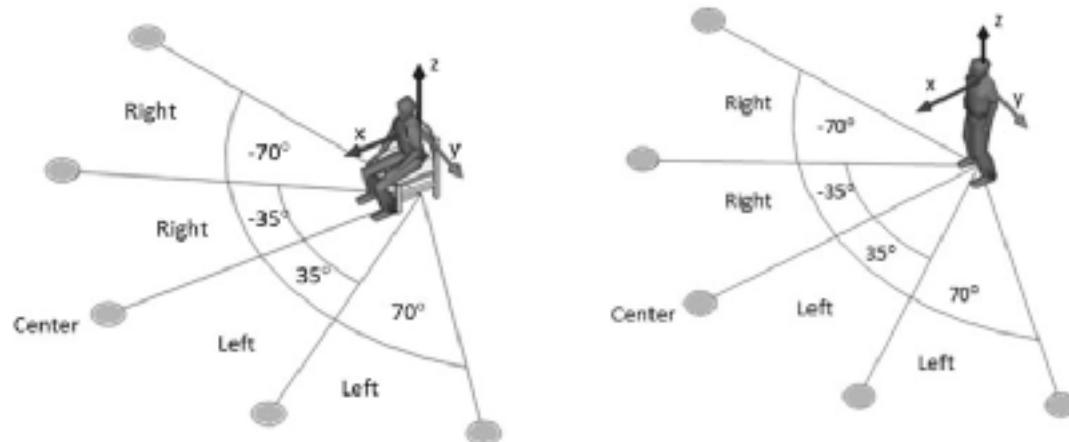


The structure of an experiment...



Example HRI Experiment (1)

- Study by (Torta et al, 2013): Design of a parametric model of personal space for robotic social navigation
- Small humanoid (Nao) approached human participants in one of two conditions: sitting or standing
 - Approach from a range of angles: humans would stop it when they wanted to
- Indication that the HRI distance is longer than would be expected in HHI
 - Also see (Walters et al, 2009)
- Effect of approach angle, and whether standing or sitting



Example HRI Experiment (2)

- Study by (Beck et al, 2010): exploring how body posture of a Nao robot relates to the interpretation of emotion
 - Displaying “key poses” and assessing interpretation
- Research question:
Is the interpretation of the key poses displayed consistent with their positions in the Affect Space?
- Participants better than chance at interpreting the five key poses
- Some blending was also possible, though this made it more difficult to identify

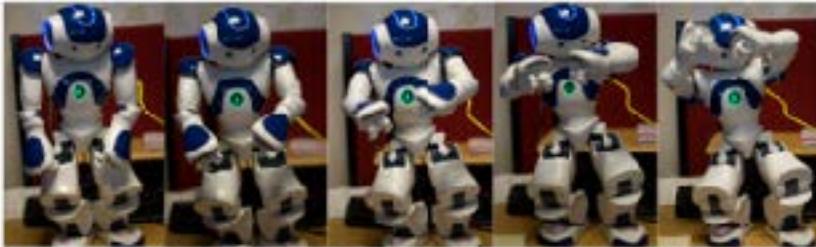


Figure 3: Five Key poses generated by the system (A: 100% Sadness, B: 70% Sadness 30% Fear, C: 50% Sadness 50% Fear, D: 30% Sadness 70% Fear, E: 100% Fear).

Table 2: Postures and their main interpretations.

“None” indicates that the question was left unanswered.

Posture	Most common Primary Interpretation (PI)	Most common Secondary Interpretation (SI)
100% Sadness	Sadness (74%)	None (70%)
50% Sadness 50% Neutral	Neutral (52%)	None (70%)
70% Sadness 30% Pride	Sadness (61%)	None (70%)
50% Sadness 50% Pride	Neutral (52%)	None (52%)

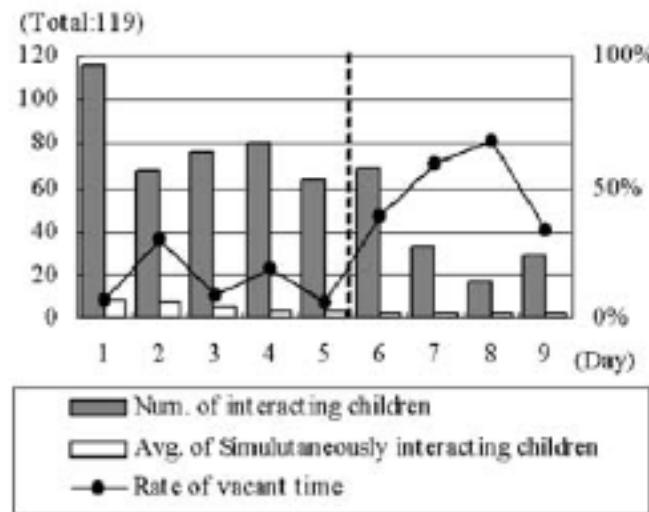
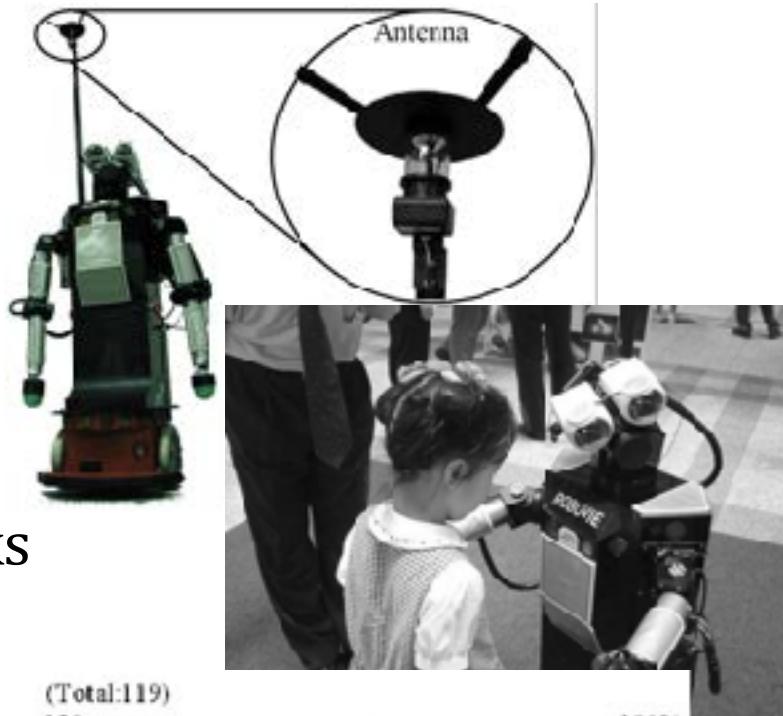
Pilot studies

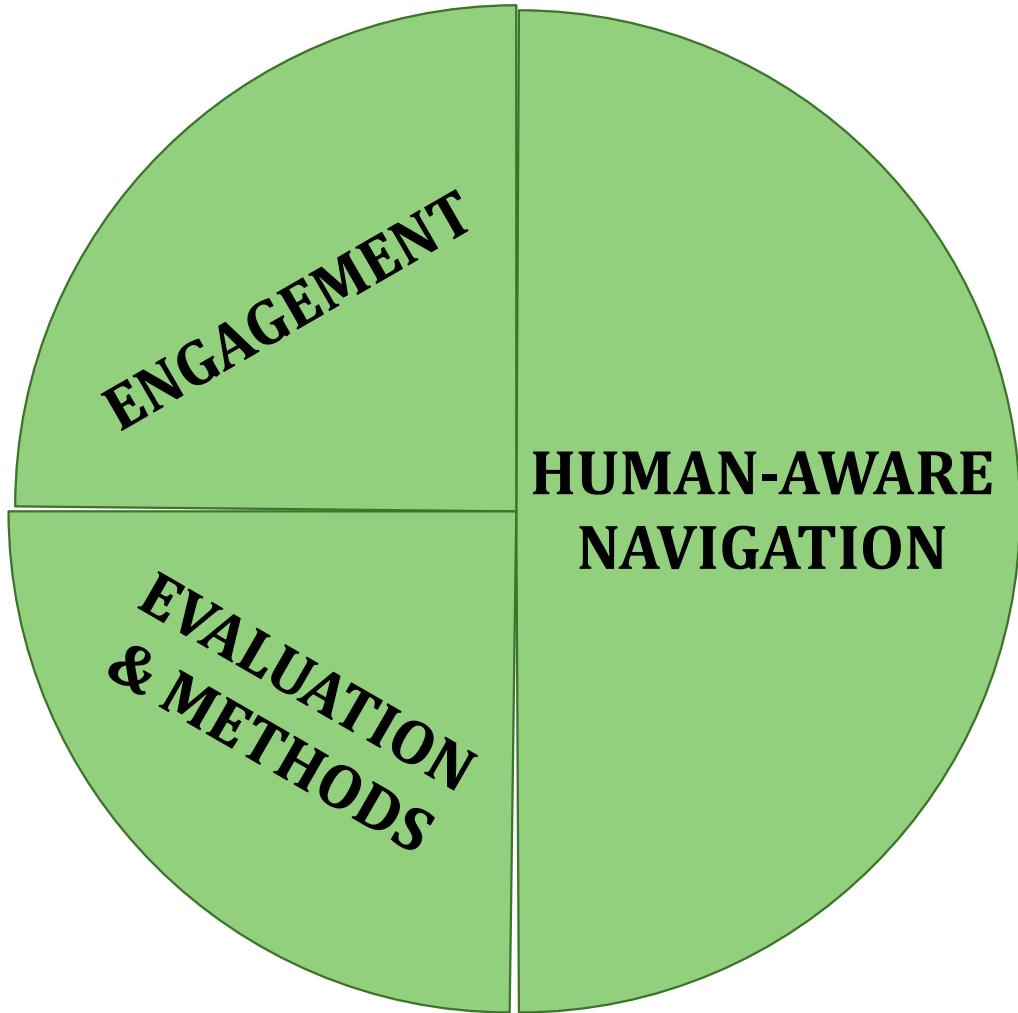
... or *Exploratory Studies*

- Can learn from studies that don't formally fit within the scientific method outlined:
 - Studies with no hypothesis (though would still have some expectations of what will/could happen – if only for ethics)
 - Perhaps a hypothesis, but only a single experimental condition
- Some degree of control and rigour are still required to draw meaningful observations (e.g. Kanda et al, 2007)
 - Reduce the incidence of confounds, or even discover what the confounds are
 - Finding data from which hypotheses can be formed

Example Pilot Study

- Field trial:
 - (Kanda et al, 2004)
 - Two weeks in a corridor speaking English with Japanese children
 - Watch to see what happens
- Various pre/post expt checks
 - E.g. English level etc
- No explicit hypothesis, one experimental condition
 - Could examine rates of interaction over time
 - And a look at learning...





References / Reading

- Baxter, P. et al., 2014. Tracking gaze over time in HRI as a proxy for engagement and attribution of social agency. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14*. Bielefeld, Germany: ACM Press, pp. 126–127.
- Beck, A., Hiolle, A., Mazel, A., & Cañamero, L. (2010). Interpretation of emotional body language displayed by robots. *Proceedings of the 3rd International Workshop on Affective Interaction in Natural Environments - AFFINE '10*, 37.
- Glas, N. & Pelachaud, C., 2015. Definitions of Engagement in Human-Agent Interaction. In *International Conference on Affective Computing and Intelligent Interaction (ACII)*. Xi'an, China: IEEE Press, pp. 944–949.
- Gonzalez-Arjona, D. et al., 2013. Simplified Occupancy Grid Indoor Mapping Optimized for Low-Cost Robots. *ISPRS Int. J. Geo-Information*, 2, pp.959–977.
- Hall, E., 1966. The Hidden Dimension, Anchor Books
- Kanda, T. et al., 2004. Interactive Robots as Social Partners and Peer Tutors for Children: A Field Trial. *Human-Computer Interaction*, 19(1), pp.61–84.
- Kruse, T. et al., 2013. Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, 61(12), pp.1726–1743.
- Lemaignan, S. et al., 2016. From real-time attention assessment to “with-me-ness” in human-robot interaction. In *ACM/IEEE International Conference on Human-Robot Interaction*. Christchurch, New Zealand.
- Lu, D. V. Hershberger, D. & Smart, W.D., 2014. Layered Costmaps for Context-Sensitive Navigation. In *IROS 2014*. Chicago, USA: IEEE, pp. 709–715.
- Sidner, C.L. et al., 2005. Explorations in engagement for humans and robots. *Artificial Intelligence*, 166(1–2), pp.140–164.
- Tanaka, F., Cicourel, A., & Movellan, J. R. (2007), “Socialization between toddlers and robots at an early childhood education center”, *PNAS*, 102(46), 17954–17958.
- Torta, E., Cuijpers, R.H. & Juola, J.F., 2013. Design of a Parametric Model of Personal Space for Robotic Social Navigation. *International Journal of Social Robotics*, 5(3), pp.357–365.
- Walters , M L , et al, 2009 , 'An empirical framework for human-robot proxemics' in *Procs of New Frontiers in Human-Robot Interaction : symposium at the AISB09 convention* . pp. 144-149 .