Robotics

Lecture 2: Robot Motion

See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

Andrew Davison
Department of Computing
Imperial College London

Robot Motion

- A mobile robot can *move* and *sense*, and must *process information* to link these two. In this lecture we concentrate on robot movement, or locomotion.

What are the possible goals of a robot locomotion system?

- Speed and/or acceleration of movement.
- Precision of positioning (repeatability).
- Flexibility and robustness in different conditions.
- Efficiency (low power consumption)?

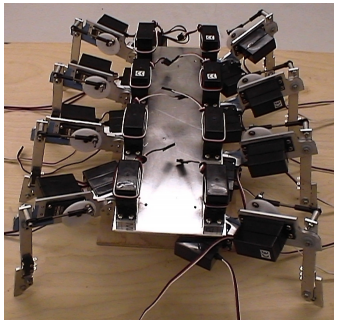
Locomotion

- Robots might want to move in water, in the air, on land, in space...?

AUV

Micro UAV

Zero-G Assistant

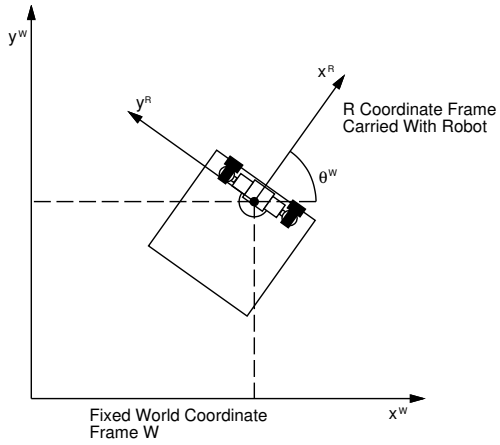


Spider

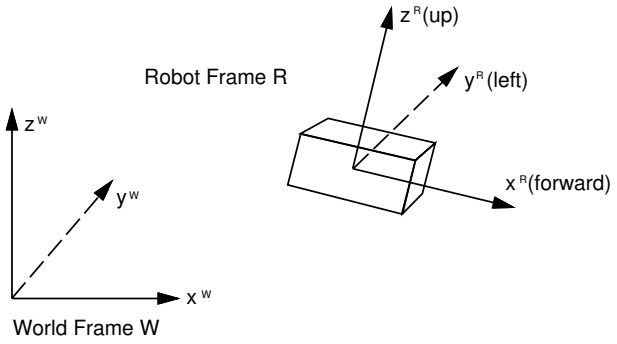
Humanoid

- In this course we will concentrate on wheeled robots which move on fairly flat surfaces.

Motion and Coordinate Frames



2D



3D

- We define two coordinate frames: a *world frame W* anchored in the world, and a *robot frame R* which is carried by and stays fixed relative to the robot at all times.
- Often we are interested in knowing the robot's *location*: i.e. what is the transformation between frames W and R?

Degrees of Motion Freedom

- A rigid body which translates and rotates along a 1D path has 1 degree of freedom (DOF): translational. Example: a train.
- A rigid body which translates and rotates on a 2D plane has 3 DOF: 2 translational, 1 rotational. Example: a ground robot.
- A rigid body which translates and rotates in a 3D volume has 6 DOF: 3 translational, 3 rotational. Example: a flying robot.

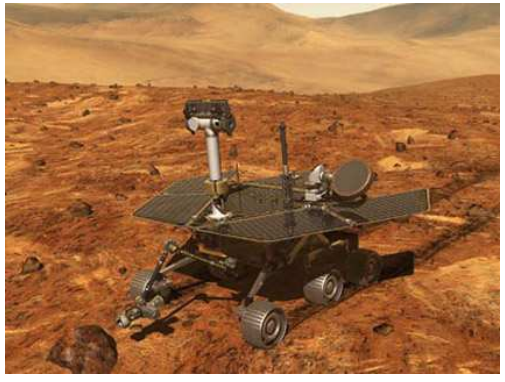
- A *holonomic robot* is one which is able to move instantaneously in any direction in the space of its degrees of freedom.

A Holonomic Ground Robot

- Holonomic robots do exist, but need many motors or unusual designs and are often impractical.
- Ground-based holonomic robots can be made using omnidirectional wheels; e.g.
  `http://www.youtube.com/watch?v=HkhGr7qfeT0`

Exotic Wheeled Robots

Segway RMP



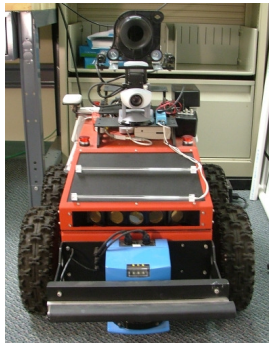
Mars Rover

- Segway platform with dynamic balance gives good height with small footprint and high acceleration.
- Mars Rover has wheels on stalks to tackle large obstacles.

Standard Wheel Configurations

Rack and Pinion

Differential Drive



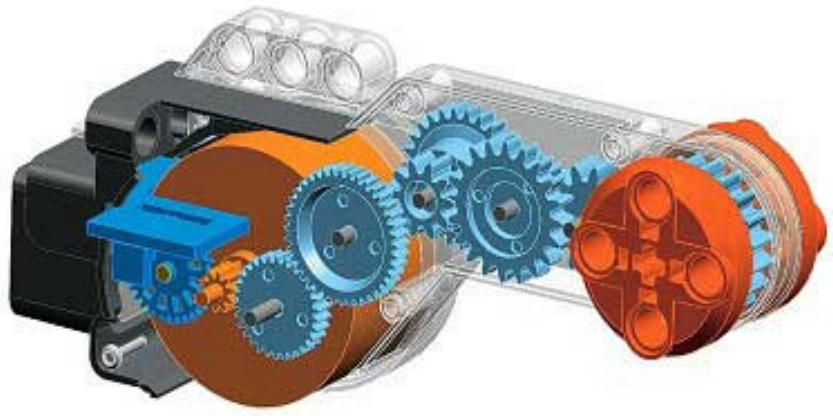
Skid-Steer

Synchro Drive

- Simple, reliable, robust mechanisms suitable for robots which essentially move in a plane.
- All of these robots are *non-holonomic* (each uses two motors, but three degrees of movement freedom).

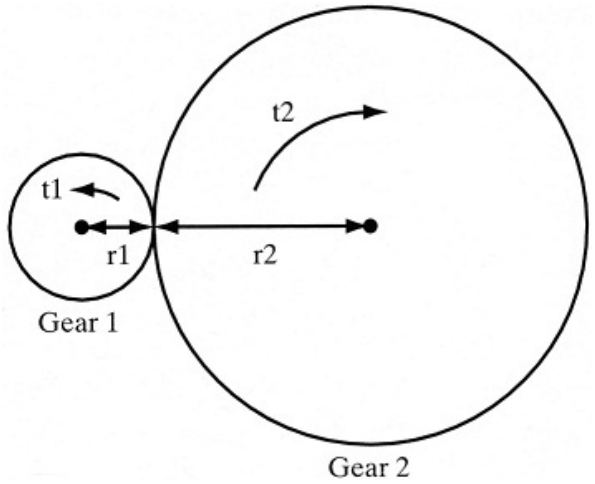
DC Motors

- Most common motors, available in all sizes and types.
- Simple control with voltage or pulse width modulation.
- For precision, encoders and feedback can be used for *servo* control (the NXT motors have built-in encoders).



Gearing

- DC motors tend to offer high speed and low torque, so gearing is nearly always required to drive a robot



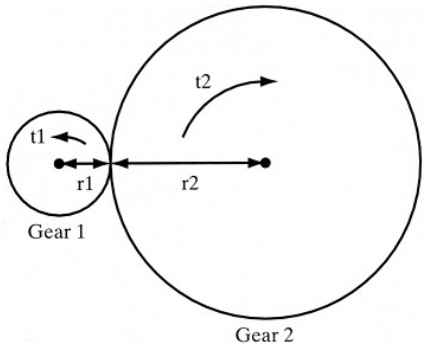
If Gear 1 is driven with torque t_1, it exerts tangential force:

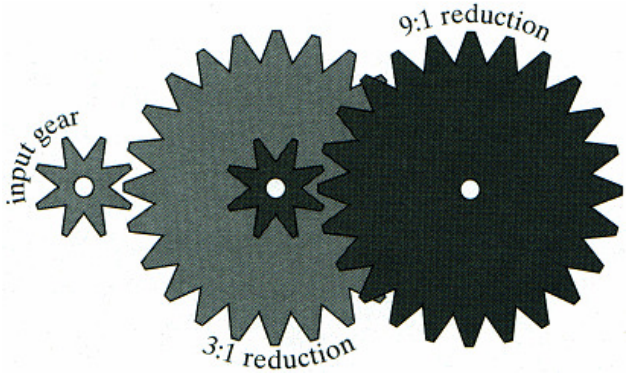
$$F = \frac{t_1}{r_1}$$

on Gear 2. The torque in Gear 2 is therefore:

$$t_2 = r_2 F = \frac{r_2}{r_1} t_1 \ .$$

Gearing



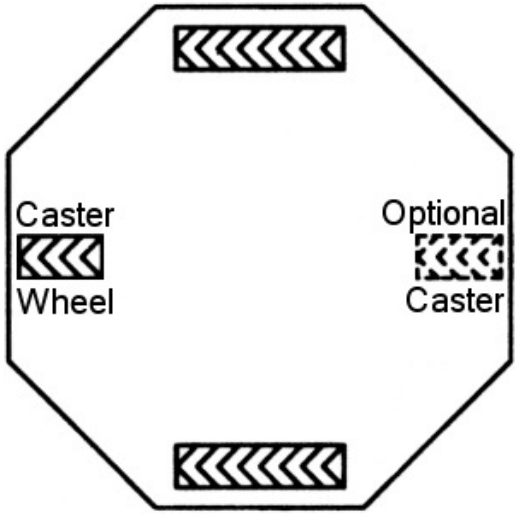


The change in angular velocity between Gear 1 and Gear 2 is calculated by considering velocity at the point where they meet:

$$\begin{aligned} v &= \omega_1 r_1 = \omega_2 r_2 \\ \Rightarrow \omega_2 &= \frac{r_1}{r_2}\omega_1 \end{aligned}$$

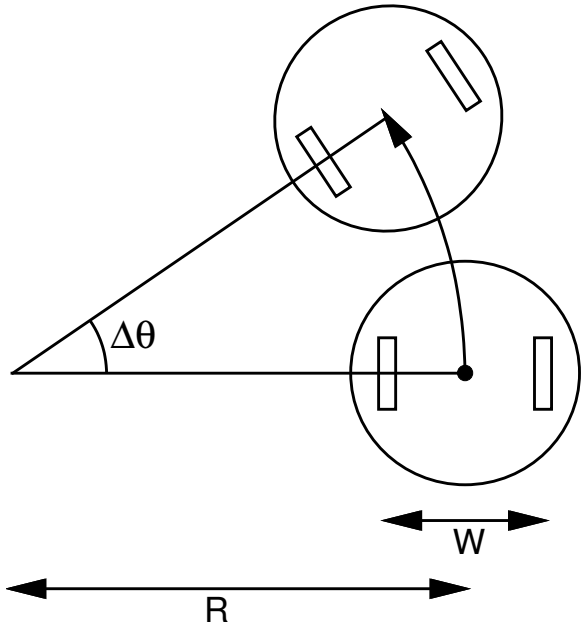
- When a small gear drives a bigger gear, the second gear has higher torque and lower angular velocity in proportion to the ratio of teeth.
- Gears can be chained together to achieve compound effects.

Differential Drive



- Two motors, one per wheel: steering achieved by setting different speeds.
- Wheels run at equal speeds for straight-line motion.
- Wheels run at equal and opposite speeds to turn on the spot.
- Other combinations of speeds lead to motion in a circular arc.

Circular Path of a Differential Drive Robot



If we define wheel velocities (the translational speed at which they cover ground) v_L and v_R, and width between wheels is W:

- Straight line motion if $v_L = v_R$
- Turns on the spot if $v_L = -v_R$
- More general case: moves in a circular arc.

Circular Path of a Differential Drive Robot

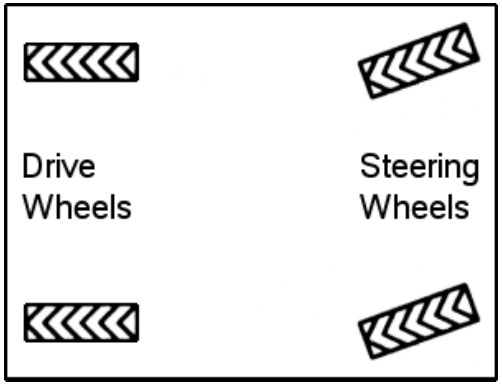
To find radius R of curved path: consider a period of motion Δt where the robot moves along a circular arc through angle $\Delta\theta$.

- Left wheel: distance moved $= v_L \Delta t$; radius of arc $= R - \frac{W}{2}$.
- Right wheel: distance moved $= v_R \Delta t$; radius of arc $= R + \frac{W}{2}$.
- Both wheel arcs subtend the same angle $\Delta\theta$ so:

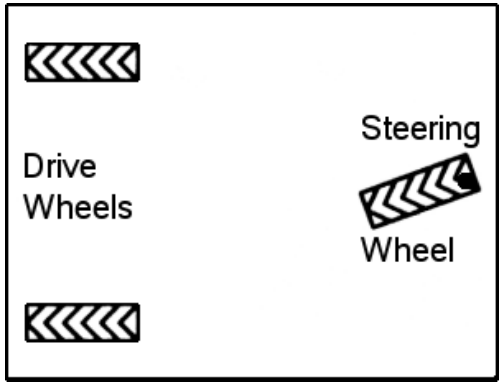
$$\begin{aligned} \Delta\theta &= \frac{v_L \Delta t}{R - \frac{W}{2}} = \frac{v_R \Delta t}{R + \frac{W}{2}} \\ \Rightarrow \frac{W}{2}(v_L + v_R) &= R(v_R - v_L) \end{aligned}$$

$$\Rightarrow R = \frac{W(v_R + v_L)}{2(v_R - v_L)} \qquad \Delta\theta = \frac{(v_R - v_L)\Delta t}{W}$$

Car/Tricycle/Rack and Pinion Drive



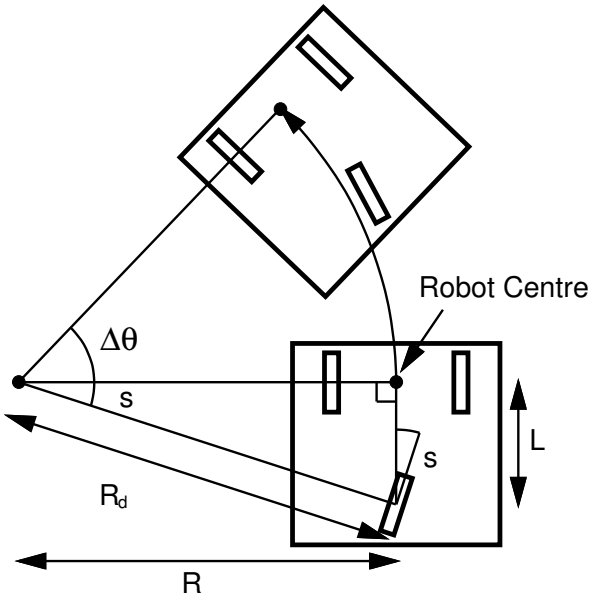
Car



Tricycle

- Two motors: one to drive, one to steer.
- Cannot normally turn on the spot.
- With a fixed speed and steering angle, it will follow a circular path.
- With four wheels, need rear differential and variable ('Ackerman') linkage for steering wheels.

Circular Path of a Car-Like Tricycle Robot



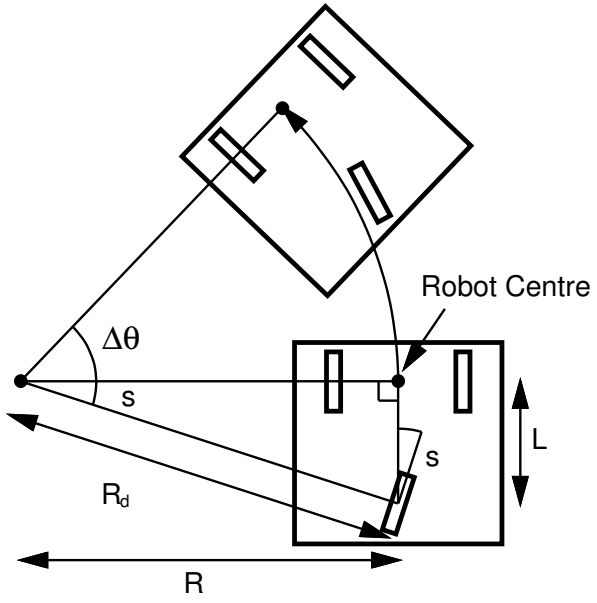
Assuming no sideways wheel slip, we intersect the axes of the front and back wheels to form a right-angle triangle, and obtain:

$$R = \frac{L}{\tan s} \ .$$

The radius of the path that the rear driving wheel moves in is:

$$R_d = \frac{L}{\sin s} \ .$$

Circular Path of a Car-Like Tricycle Robot



In time Δt the distance along its circular arc moved by the drive wheel is $v\Delta t$, so the angle $\Delta\theta$ through which the robot rotates is:

$$\Delta\theta = \frac{v\Delta t}{R_d} = \frac{v\Delta t \sin s}{L} .$$

$$R = \frac{L}{\tan s} \qquad \Delta\theta = \frac{v\Delta t \sin s}{L}$$

Estimating Motion from On-Board Sensors

- Very often, a robot will want to estimate its motion by monitoring its proprioceptive sensors, such as motor voltages or wheel encoders. This information is usually known as *odometry*.
- e.g. for very simple voltage/timing based estimation:

$$D = KV\Delta t$$

  Distance travelled is proportional to voltage and time. Here K is a constant to be calculated (from knowledge of electrics and geometry) or *calibrated*.
- Calibration involves experimental motion and comparison with ground truth.
- Encoders give more accurate, measured odometry of the number of wheel turns and this information can be converted in principle to linear distance by multiplying by a constant for wheel radius; but still we will normally calibrate for full accuracy.

Calibration

- When a calculation of expected motion is made, and then tested in the real world with a real robot motion, it can be compared with *ground truth* measurement of how the robot has actually moved.
- If there is a discrepancy over repeated trials, we can aim to improve matters by altering the values of the constants in our expressions (such as K above) and then iterating.

Motion and State on a 2D Plane

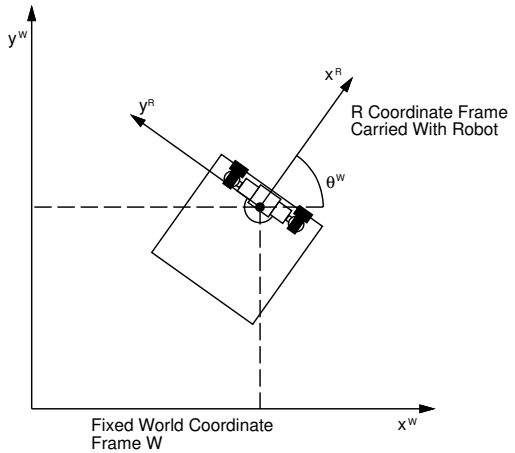
- If we assume that a robot is confined to moving on a plane, its location can be defined with a *state vector* **x** consisting of three parameters:

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

- x and y specify the location of the pre-defined 'robot centre' point in the world frame.
- θ specifies the rotation angle between the two coordinate frames (the angle between the x^W and x^R axes).
- The two coordinate frame coincide when the robot is at the origin, and $x = y = \theta = 0$.

Integrating Motion in 2D

- 2D motion on a plane: three degrees of positional freedom, represented by (x, y, θ) with $-\pi < \theta <= \pi$.
- Consider a robot which only drives ahead or turns on the spot:



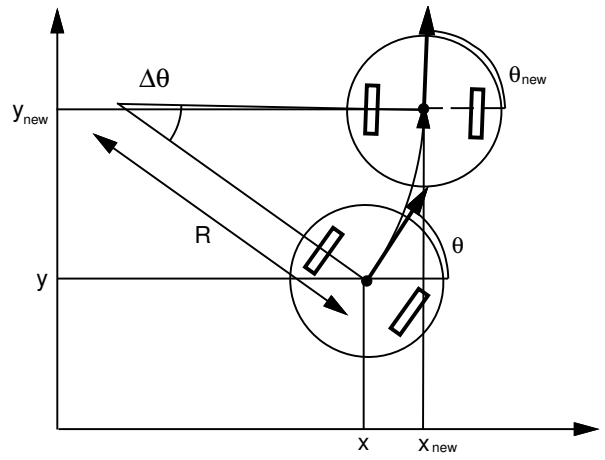
- During a straight-line period of motion of distance D:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + D\cos\theta \\ y + D\sin\theta \\ \theta \end{pmatrix}$$

- During a pure rotation of angle angle α:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha \end{pmatrix}$$

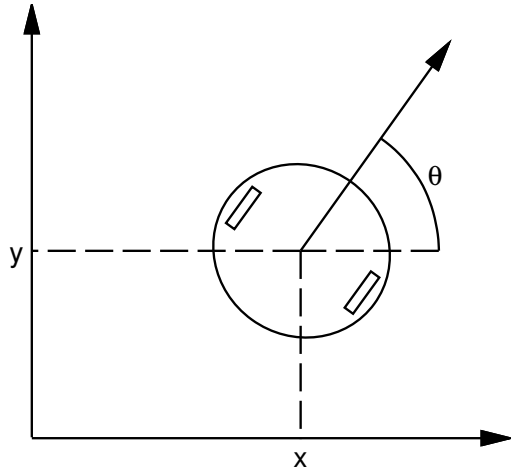
Integrating Circular Motion Estimates in 2D



In the cases of both differential drive and the tricycle robot, we were able to obtain expressions for R and $\Delta\theta$ for periods of constant circular motion. Given these:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + R(\sin(\Delta\theta + \theta) - \sin\theta) \\ y - R(\cos(\Delta\theta + \theta) - \cos\theta) \\ \theta + \Delta\theta \end{pmatrix}$$

Position-Based Path Planning



Assuming that a robot has *localisation*, and knows where it is relative to a fixed coordinate frame, then position-based path planning enables it to move in a precise way along a sequence of pre-defined waypoints. Paths of various curved shapes could be planned, aiming to optimise criteria such as overall time or power usage. Here we will consider the specific, simple case where we assume that:

- Our robot's movements are composed by straight-line segments separated by turns on the spot.
- The robot aims to minimise total distance travelled, so it always turns immediately to face the next waypoint and drives straight towards it.

Position-Based Path Planning

In one step of path planning, assume that the robot's current pose is (x, y, θ) and the next waypoint to travel to is at (W_x, W_y).

- It must first rotate to point towards the waypoint. The vector direction it must point in is:

$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} W_x - x \\ W_y - y \end{pmatrix}$$

  The absolute angular orientation α the robot must drive in is therefore given by:

$$\alpha = \tan^{-1} \frac{d_y}{d_x}$$

Care must be taken to make sure that α is in the correct quadrant of $-\pi < \alpha \leq \pi$. A standard $\tan^{-1}$ function will return a value in the range $-\pi/2 < \alpha <= \pi/2$. This can be also achieved directly with an `atan2(dy, dx)` function usually available in C (though not in RobotC!).

Position-Based Path Planning

- The angle the robot must rotate through is therefore $\beta = \alpha - \theta$. If the robot is to move as efficiently as possible, care should be taken to shift this angle by adding or subtracting 2π so make sure that $-\pi < \beta \leq \pi$.
- The robot should then drive forward in a straight line through distance $d = \sqrt{d_x^2 + d_y^2}$.

Practical Week 2

- Location: Teaching Lab Room 202
- Divide yourselves into groups; each group gets access to one Lego Mindstorms NXT kit located in numbered drawers to which I have the keys.
- I will note down the members of each group. One member of each team who will be specifically responsible for their kit, should make sure that it goes back in the drawer when it is not being used and looks after the keys.
- There are PCs in Room 202 with RobotC installed — these are indicated by 'RobotC' stickers on the monitors.

Lego Mindstorms NXT

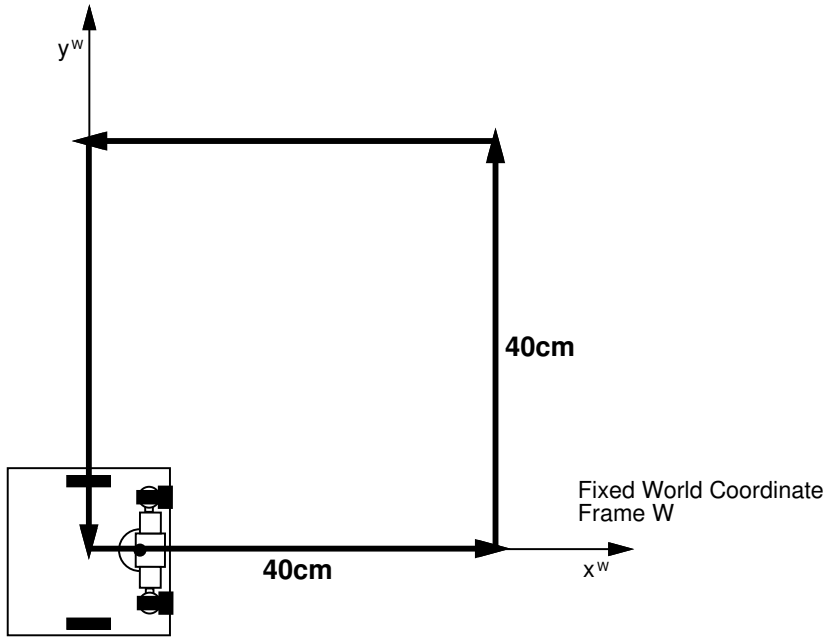
These are the important components; please look after them!

1 sonar (ultrasound) sensor	2 light sensors
1 Brick	1 battery
1 charger	USB cable
3 NXT motors	2 touch sensors
1 sound sensor	1 compass sensor

RobotC

- A comprehensive C programming environment for Mindstorms NXT development installed on labelled machines in 202.
- Run it from the desktop from Windows.
- Edit and compile programs on the PC then download to Brick via USB port.
- Run and monitor from RobotC with cable connected or disconnect and run on Brick independently.
- Many sample programs included.
- Extra documentation at `http://www.robotc.net/`
- Free trial version can be installed for 1 month your own PC.

Week 2 Practical: Locomotion, Calibration and Accurate Motion



- Today's practical is on accurate robot motion. How well is it really possible to estimate robot motion from wheel odometry?
- **Everyone should read the practical sheet fully!**
- This is an **ASSESSED** practical: we will assess your achievement next week.
- Please go straight to the lab now.

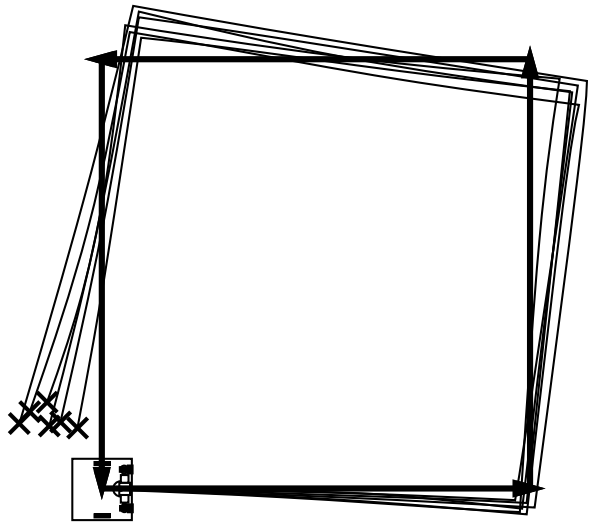
Robotics

Lecture 3: Sensors

See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

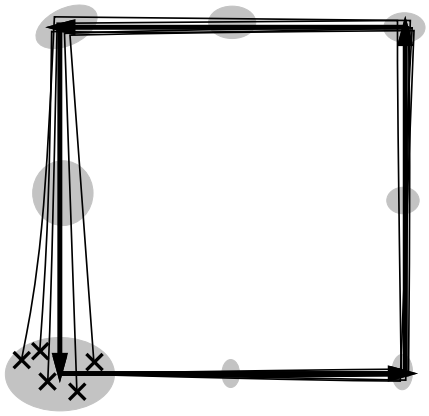
Andrew Davison
Department of Computing
Imperial College London

Review: Locomotion Practical from Lecture 2



- Gradual 'motion drift' from perfect square
- Causes: initial alignment errors; wheel slip; miscalibration; unequal left/right motors; others . . . ?
- Using the motor encoders and synchronizing the two motors improves matters but we will never achieve a perfect result every time in this experiment.

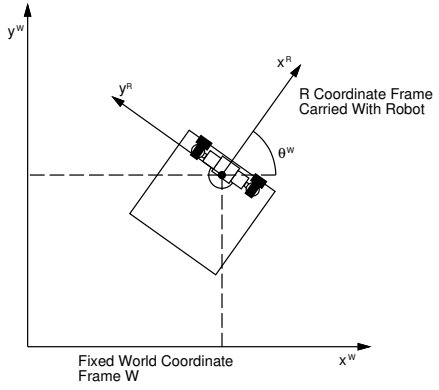
A Well-Calibrated Robot



- After careful calibration the robot should *on average* return to the desired location, but scatter will remain due to uncontrollable factors (variable wheel slip, surface, air currents!?...)
- *Systematic error* removed; what remains are *zero mean errors*.
- We can model the zero mean errors probabilistically: in most cases a Gaussian (normal) distribution is suitable.
- The size of the distribution of the errors will grow as the robot moves further around the square.

What does this mean for estimating motion?

- Perfect motion integration from odometry is not possible:



- During a straight-line period of motion of distance D:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + D\cos\theta \\ y + D\sin\theta \\ \theta \end{pmatrix}$$

- During a pure rotation of angle angle α:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha \end{pmatrix}$$

Uncertainty in Motion

- A better model acknowledges that this 'ideal' trajectory is affected by uncertain perturbations ('motion noise').
- During a straight-line period of motion of distance D:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + (D + e)\cos\theta \\ y + (D + e)\sin\theta \\ \theta + f \end{pmatrix}$$

- During a pure rotation of angle angle α:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$$

- Here f and g are 'noise' variables, with zero mean and a Gaussian distribution, describing how actual motion might deviate from the ideal trajectory.

Sensors: Proprioceptive and Outward-Looking

- As we saw in the first lecture, sensors are either *proprioceptive* (literally self-sensing) or *outward-looking*.
- Proprioceptive sensors (such as motor encoders or internal force sensors) will improve a robot's sense of its own internal state and motion.
- But without outward-looking sensors a mobile robot is moving blindly. It needs them to:
  - Recognise landmarks and localise without drift.
  - Explore new places and build maps.
  - Avoid obstacles.
  - Interact with objects and people.
  - In general, be *aware* of its environment.

Sensor Measurements

- Sensors gather numerical readings or *measurements*
  - In the case of proprioceptive sensors, the value of the measurement $\mathbf{z}_p$ will depend on (be a function of) just the state of the robot $\mathbf{x}$:

$$\mathbf{z}_p = \mathbf{z}_p(\mathbf{x}) \ .$$

  - A measurement from an outward looking sensor will depend both on the state of the robot $\mathbf{x}$ and that of the world around it $\mathbf{y}$:

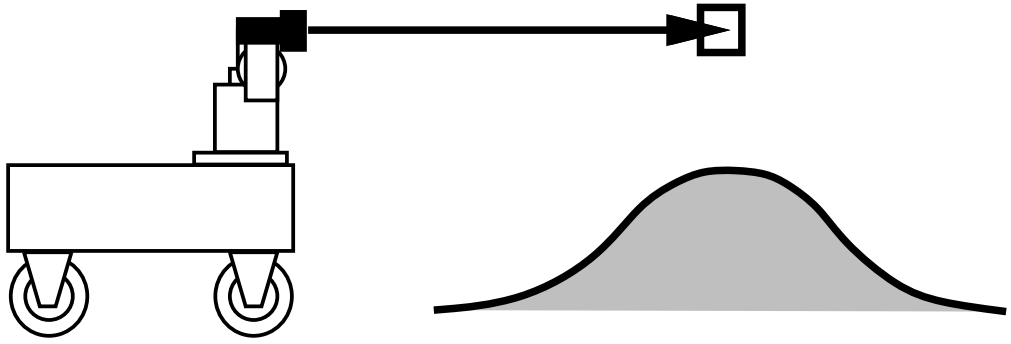
$$\mathbf{z}_o = \mathbf{z}_o(\mathbf{x}, \mathbf{y}) \ .$$

  - All of the quantities in these equations are in general vectors of multiple parameters rather than single scalar quantities.

Single and Multiple Value Sensors

- Touch, light and sonar sensors each return a *single value* within a given range.
- Sensors such as a camera or laser range-finder return an *array* of values. This can be achieved by scanning a single sensing element (as in a laser range-finder) or by having an array of sensing elements (such as the pixels of a camera's CCD chip).

Probabilistic Sensor Modelling



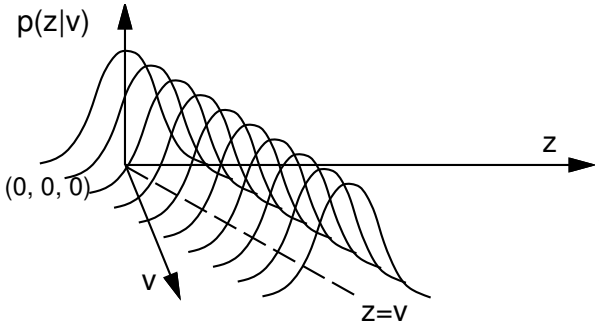
- Like robot motion, robot sensing is also fundamentally *uncertain*. Real sensors do not report the exact truth of the quantities they are measuring but a perturbed version.
- Having characterized (modelled; calibrated?) a sensor and understood the uncertainty in its measurements we can build a probabilistic measurement model for how it works. This will be a probability distribution (specifically a *likelihood function*) of the form:

$$p(\mathbf{z}_o|\mathbf{x},\mathbf{y}) \ .$$

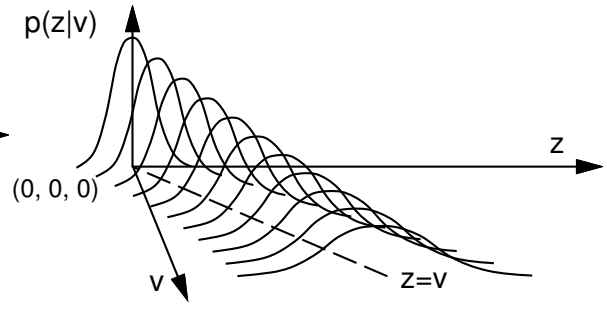
  Such a distribution will often have a Gaussian shape.

Likelihood Functions

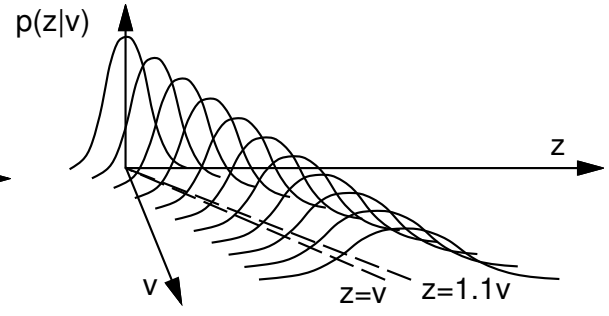
- A likelihood function fully describes a sensor's performance.
- $p(z|v)$ is a function of both measurement variables z and ground truth v and can be plotted as a probability surface. e.g. for a depth sensor:



Constant Uncertainty



Growing

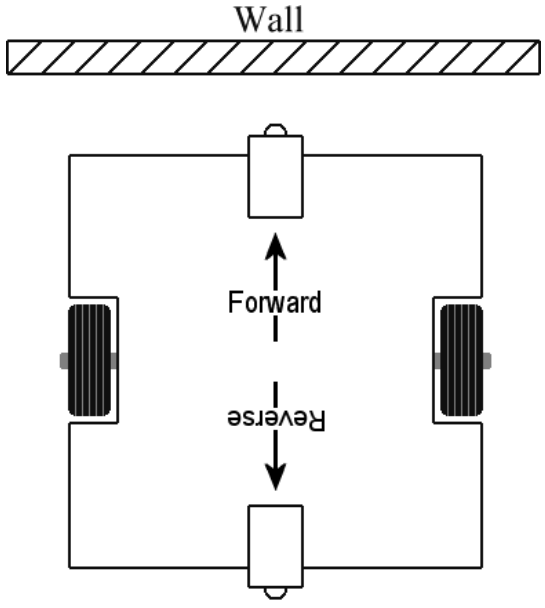


Systematic Error (Biased)

Touch Sensors

- Binary on/off state — no processing required.
- Switch open — no current flows.
- Switch closed — current flows (hit).

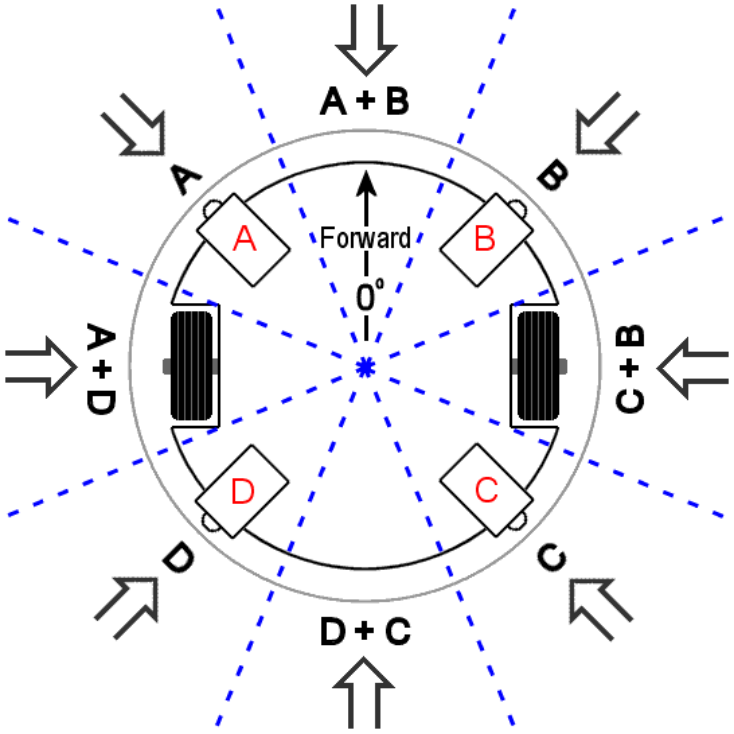
Touch Sensors for Bump Detection



- Sensor detects when an obstacle has been hit (last line of defence).
- Demands immediate reaction — evasive manoeuvre, or stop forward motion at least.
- Two bump sensors which should never be activated at the same time could be wired in parallel to the same input.

Multiple Touch Sensors — Where was I hit?

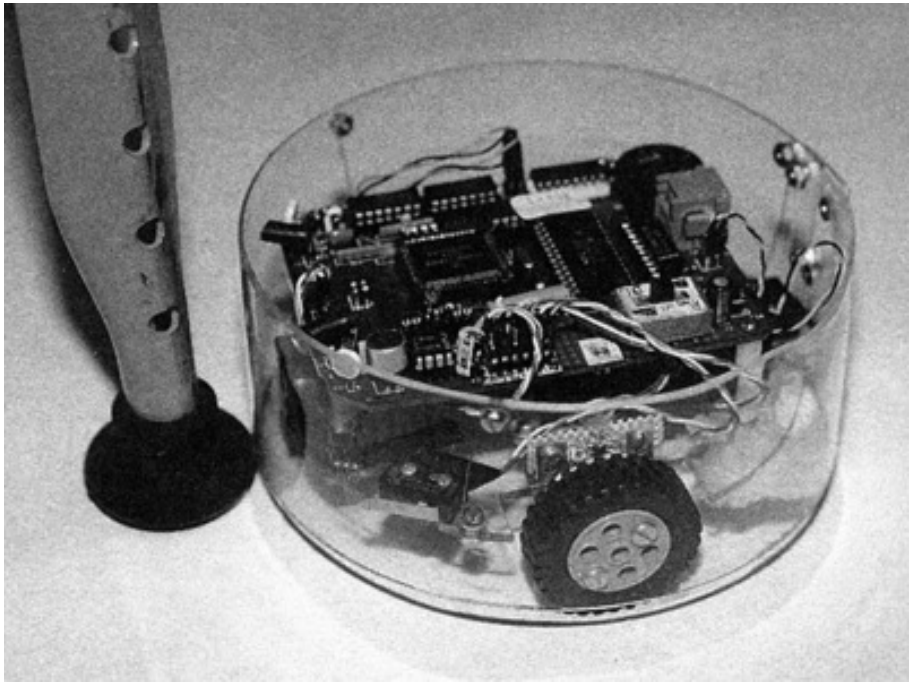
Touch sensors mounted inside 'floating skirt' around circular robot:



A	B	C	D	Sector	Centre
1	1	0	0	337.5° to 22.5°	0°
0	1	0	0	25.5° to 67.5°	45°
0	1	1	0	67.5° to 112.5°	90°
0	0	1	0	112.5° to 157.5°	135°
0	0	1	1	157.5° to 202.5°	180°
0	0	0	1	202.5° to 247.5°	225°
1	0	0	1	247.5° to 292.5°	270°
1	0	0	0	292.5° to 337.5°	315°

- Four sensors give the ability to measure eight bump directions.

Strategies After a Collision



- Explore — try to go around/along (turn to a fixed angle from hit and proceed).
- Moving object? Follow (turn towards hit, wait, then drive forward), or run away (turn away from hit and drive forward).

Light Sensors

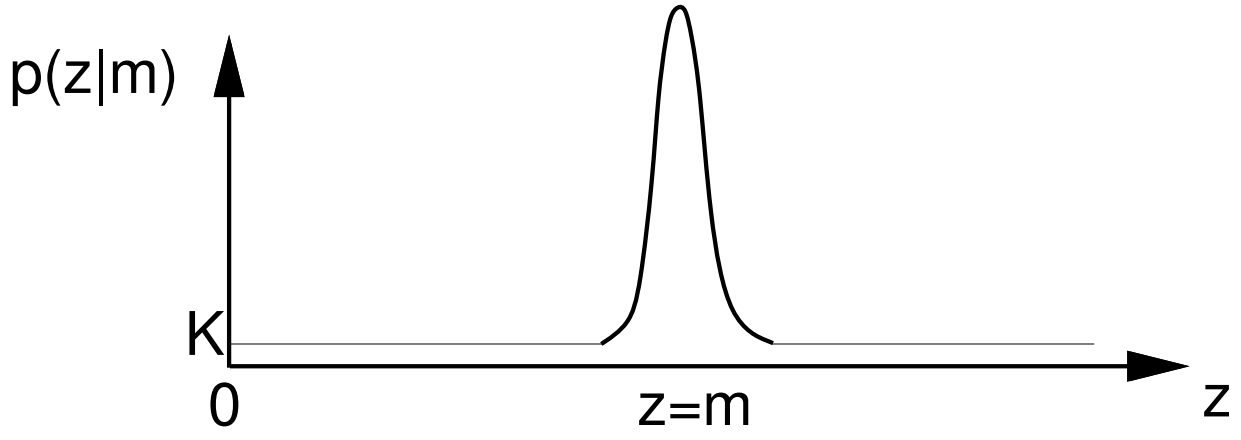
- Detect intensity of light incident from single forward direction.
- Multiple sensors pointing in different directions can guide steering behaviours.
- The Lego sensors also emit their own light, which will reflect off close targets and used for line-following (active sensing).

Sonar Sensors

- Measure depth by emitting an ultrasonic pulse and timing interval until echo returns.
- Fairly accurate depth measurement in one direction but can give 'noisy' measurements in the presence of complicated shapes.
- Robots often have a ring of sonar sensors.
- Comes into its own underwater where it is the only serious option.

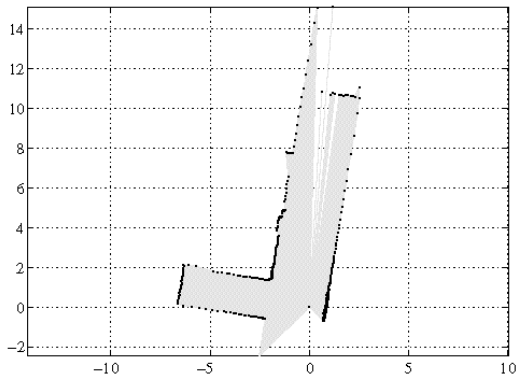
Robust Likelihood for Sonar Update

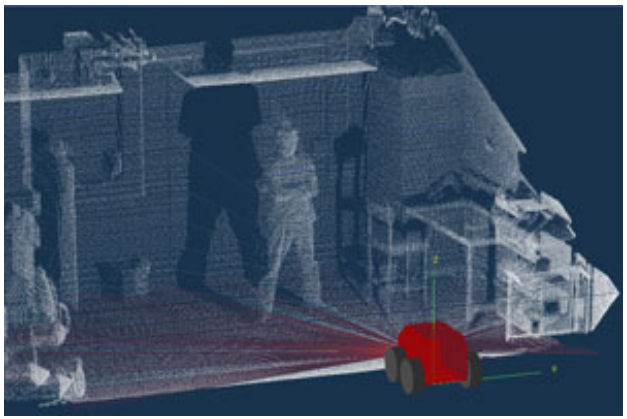
- We will return to this in later lectures on probabilistic robotics, but a suitable likelihood function for a sonar sensor is as follows: it says 'what is the probability of obtaining sensor measurement z given that the ground truth value I expect is m?
- This distribution has a narrow Gaussian band around the expected value, plus a constant additive band representing a fixed percentage of 'garbage' measurements.



$$p(z|m) \propto e^{\frac{-(z-m)^2}{2\sigma_s^2}} + K$$

External Sensing: Laser Range-Finder





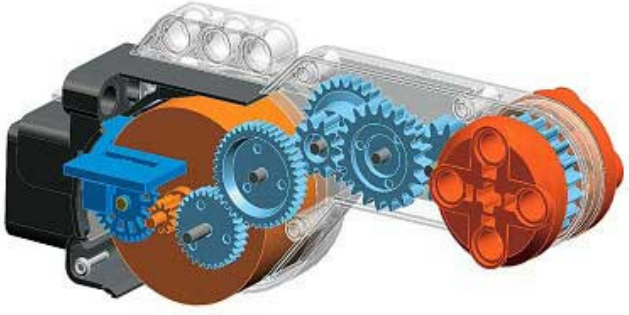
- Very accurate measurement of both depth and direction from time-of-flight measurement of scanning laser beam
- Normally scans in a 2D plane but 3D versions are also available
- Rather bulky (and expensive) for small robots

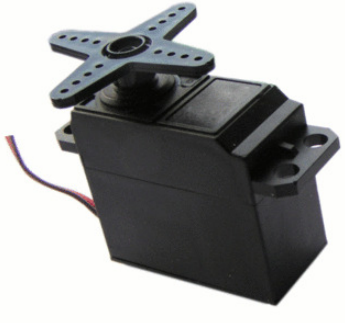
External Sensing: Vision

- A camera measures light intensity in many directions simultaneously by directing incident light onto a sensing chip with an array of light sensitive elements
- A single camera measures angles, but not depth information. 3D information comes from either multiple cameras or motion
- Vast research area: object recognition, location recognition, tracking, 3D reconstruction, etc.
- Huge motivation from biology
- Highly attractive for general purpose, low-cost robotics

Servoing

- Servoing is a robot control technique where control parameters (such as the desired speed of a motor) are coupled directly to a sensor reading and updated regularly in a *negative feedback loop*. It is also sometimes known as *closed loop control*.
- Servoing needs high frequency update of the sensor/control cycle or motion may oscillate.
- The Mindstorms NXT motors or the servos used in model radio-controlled cars and aircraft use internal position encoders to perform position control:



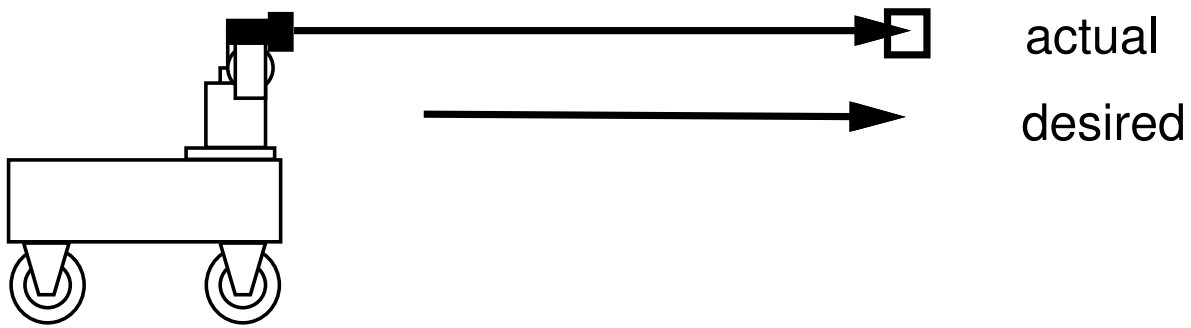


Servoing and Negative Feedback

- In servoing, a control demand is set which over time aims to bring the current value of a sensor reading into agreement with a desired value.
- Proportional control: set demand proportional to negative error (difference between desired sensor value and actual sensor value):

$$C = -\kappa(z_{desired} - z_{actual}) \ ,$$

where κ is a *gain* constant.



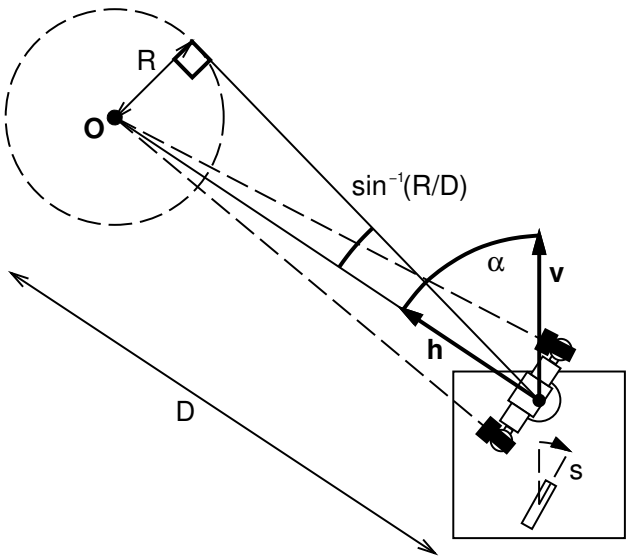
Visual Servoing to Control Steering

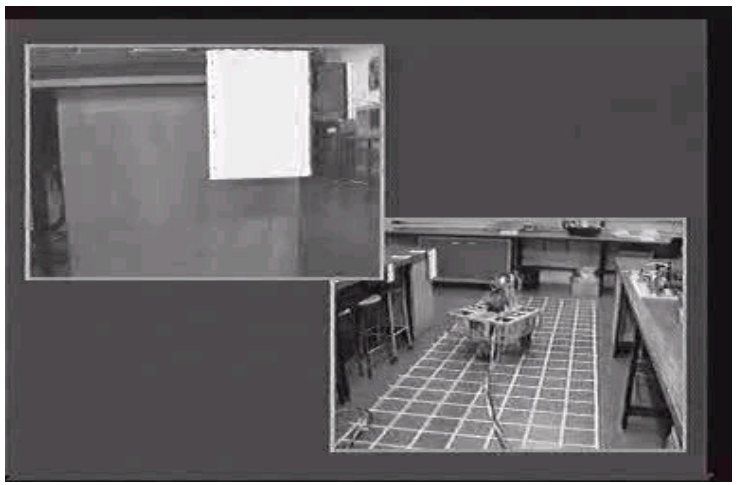
- For a robot with a tricycle or car-type wheel configuration.
- Simple steering law which will guide robot to collide with target:

$$s = \kappa\alpha$$

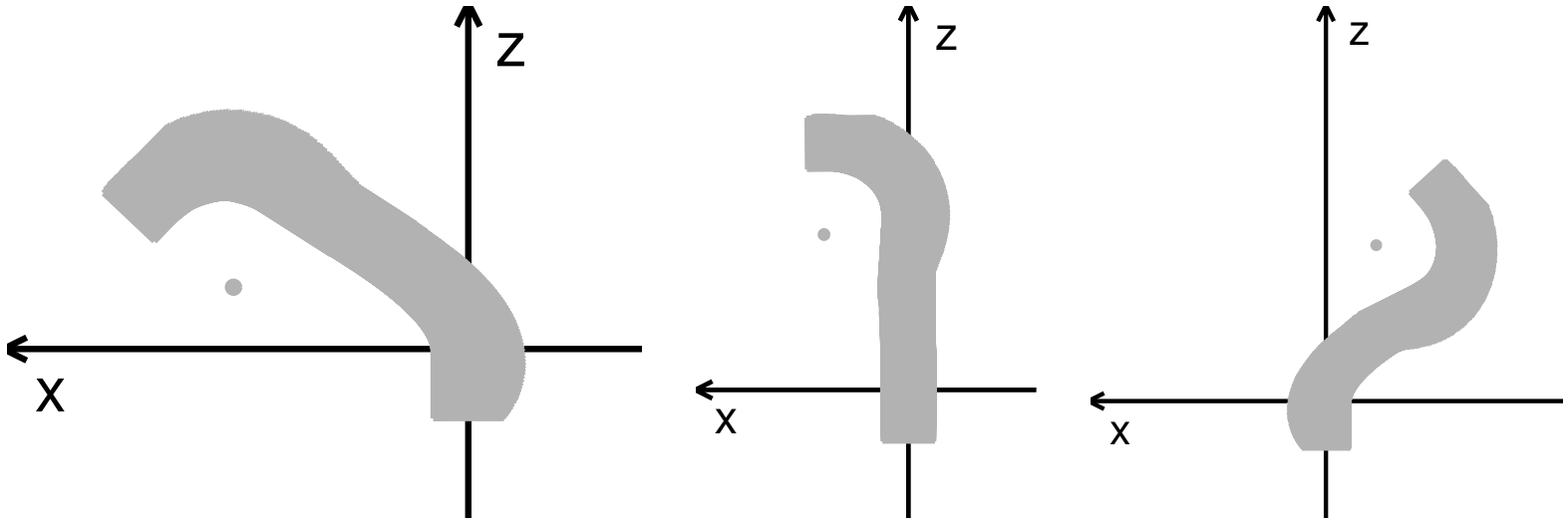
- Steering law which will guide robot to avoid obstace at a safe radius: subtract offset:

$$s = \kappa(\alpha - \sin^{-1}\frac{R}{D})$$





Visual Servoing Trajectories

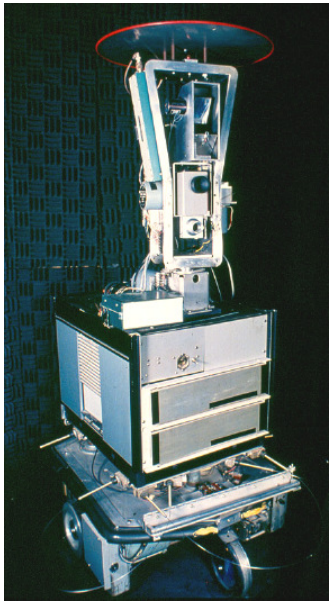


- Compare with simpler steering law which will guide robot to collide with target:

$$s = \kappa\alpha$$

Combining Sensors: World Model Approach

- Capture data; store and manipulate it using symbolic representations.
- *Plan* a sequence of actions to achieve a given goal
- Execute plan.
- If the world changes during execution, stop and re-plan.



Shakey

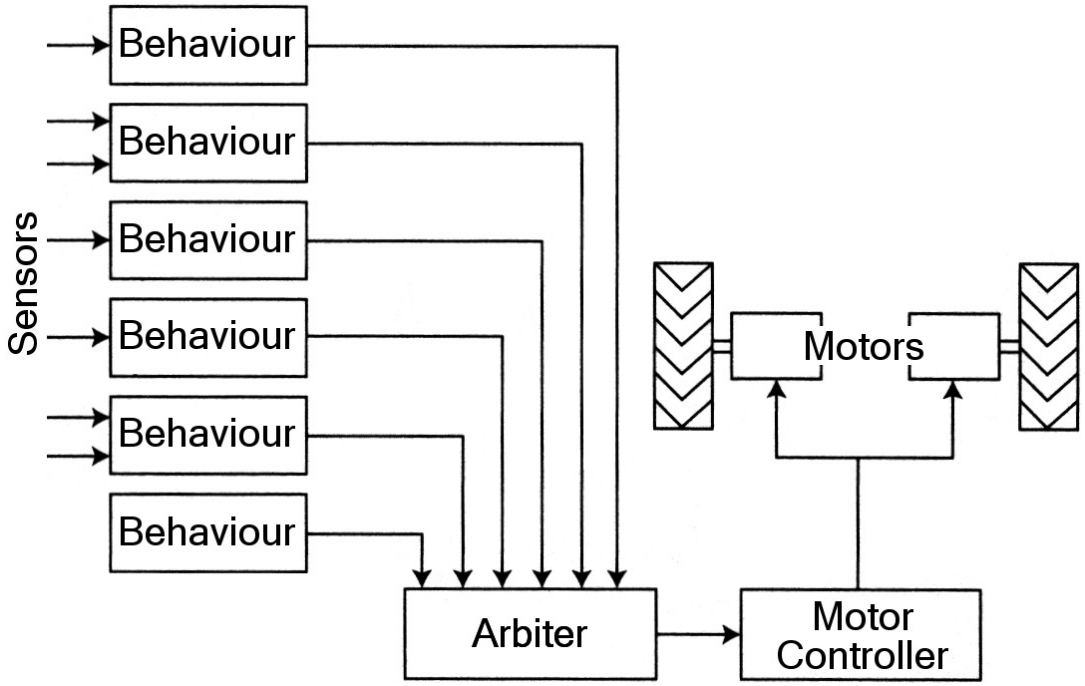
Sense $\rightarrow$ Model $\rightarrow$ Plan $\rightarrow$ Act

Combining: Sensing/Action Loops

- No modelling and planning! Consider each local 'servo'-like sensing-action loop as a **behaviour**.

Sense $\rightarrow$ Act

- The challenge is in combining many behaviours into useful overall activity: see TR Programs, Subsumption, Braitenberg vehicles in the next lecture.



Robotics

Lecture 4: Robot Behaviours

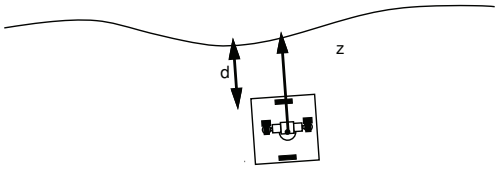
See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

Andrew Davison
Department of Computing
Imperial College London

Review: Sensors Practical from Lecture 3

- Touch sensor (returns yes/no state): use to detect collision and trigger avoidance action.
- Sonar sensor (returns depth value in cm): can be used for smooth *servoing* behaviour with proportional gain.
- Both are examples of negative feedback.

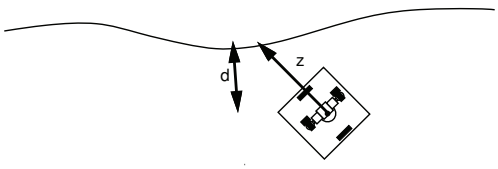
Review: Wall Following with Sonar



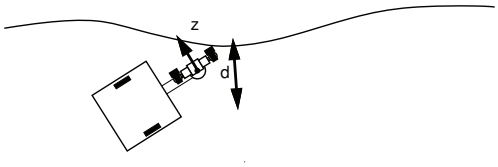
- Use sideways-looking sonar to measure distance z to wall.
- With the goal of maintaining a desired distance d, set difference between left and right wheel velocities proportional to difference between z and d:

$$v_R - v_L = \kappa(z - d)$$

Review: Wall Following with Sonar



- Problem if angle between robot's direction and wall gets too large because sonar doesn't measure the perpendicular distance.
- Solutions: ring of sonar sensors would be most straightforward. Clever combination of measurements from different times?
- Better result with sonar mounted forward of wheels.



Review: Sonar Sensor Properties

1. Mininum/maximum range?
2. Maximum angle from perpendicular to wall?
3. Interference between two sonars?
4. Accuracy: any systematic errors?
5. Accuracy: size of zero-mean errors?

Combining Sensors: World Model Approach

- Capture data; store and manipulate it using symbolic representations.
- *Plan* a sequence of actions to achieve a given goal
- Execute plan.
- If the world changes during execution, stop and re-plan.


Shakey, SRI International 1966–1972

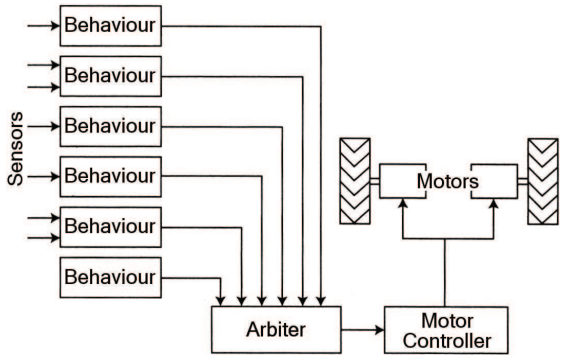
Sense $\rightarrow$ Model $\rightarrow$ Plan $\rightarrow$ Act

Combining: Sensing/Action Loops

- No modelling and planning!

Sense $\rightarrow$ Act

- A simple loop like this is often called a robot *behaviour*.
- The challenge is in combining many behaviours into useful overall activity.



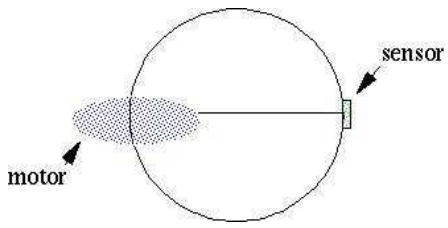
Imperial College London

Simple behaviours: Braitenberg vehicles

- Direct connections between sensors and motors
- Created in 1984 as thought experiments in study of intelligence in Braitenberg's book “Vehicles”.

Imperial College London

Variable speed – single direction



- Sensor controls speed of motor - always forward
- Sensor detects increasing heat or light increasing motor speed in response

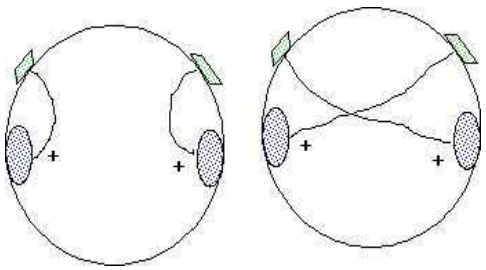
Imperial College London

Observed Anthropomorphic Behaviour

- With heat sensor robot appears to avoid warmth, seek cool areas.
- If uses a light sensor, appears to avoid light.
- If sensor has a threshold then motor will cut when sensed value is below a certain level.
- Light sensor robot appears to run towards shade - so exhibits **timid** behavour

Imperial College London

Two sensors and two motors

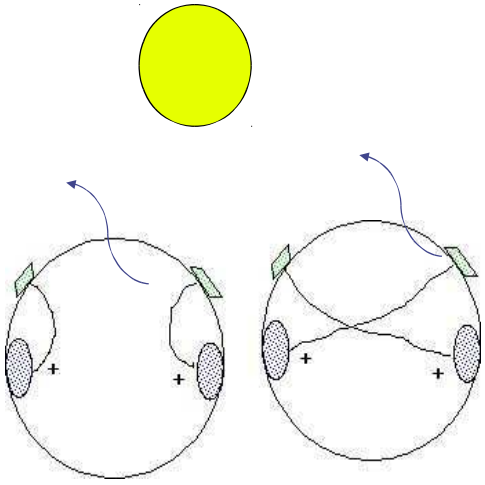


Sensors again control speed of motors

Increase in sensed value increases speed of motor

Imperial College London

Observed behaviours

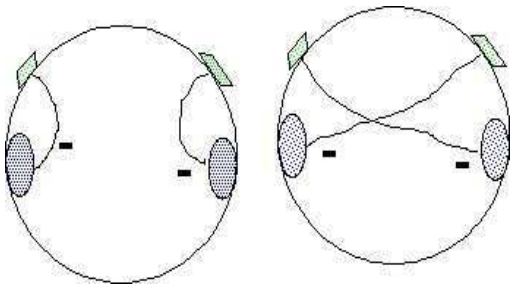


Light avoider

Light seeker

Imperial College London

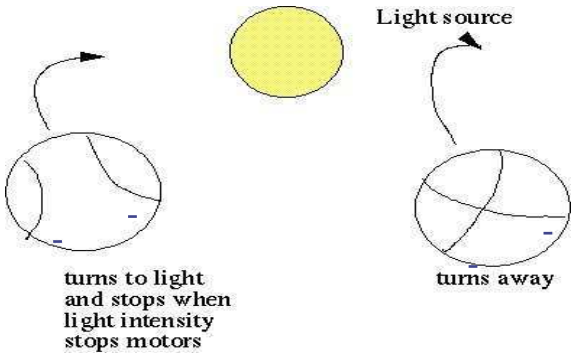
Two inhibiting sensors and two motors



Increase of sensed value decreases linked motor speed, and vice-versa.

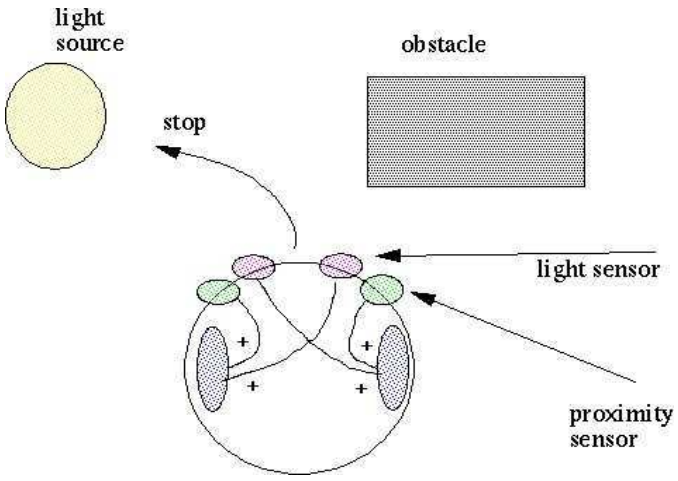
Imperial College London

Observed behaviour



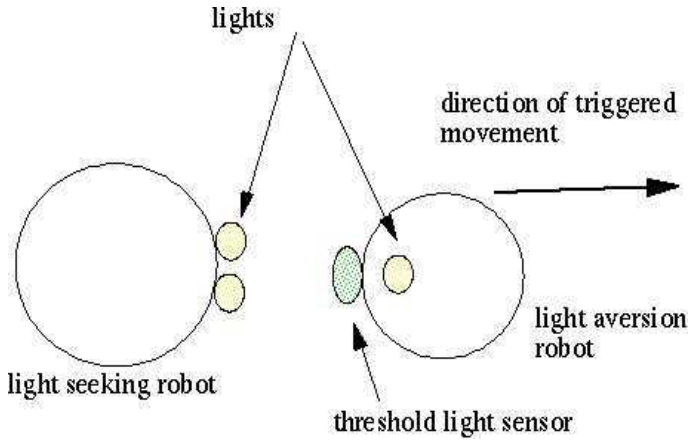
Imperial College London

Merged behaviours



Imperial College London

Following/leading behaviours



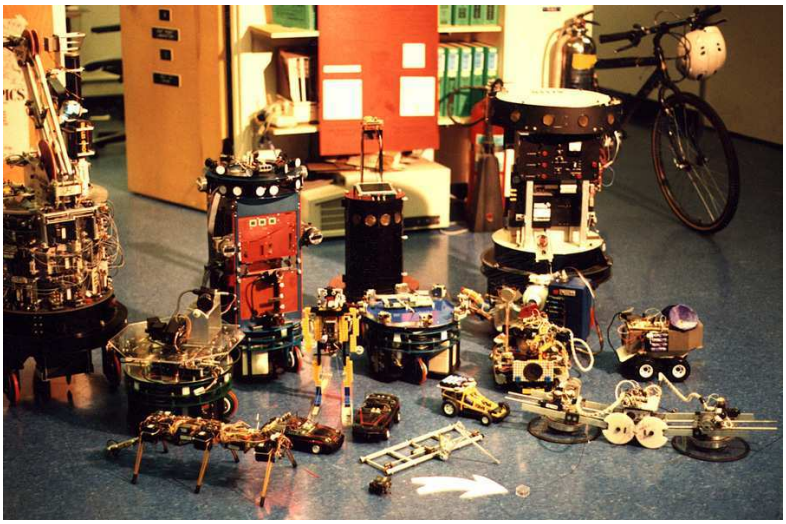
Imperial College London

Reactive control architectures

- Many of the successful robot control architectures are assemblies
  of concurrently executing micro-behaviours
- Two different ways of combining action outputs of the concurrently executing behaviours: subsumption and potential field (or vector field) summation.
- Subsumption similar to TR rule hierarchy, in which behaviours are ranked and highest ranked behaviour wanting to control the robot has control.
- Vector fields similar to the action merging of Braitenberg vehicles (light seeking + obstacle avoidance).

Imperial College London

Subsumption: Rodney Brooks, MIT AI Lab, 1980s



Imperial College London

Subsumption Philosophy

- Behaviours are grouped into *layers of competence*
- Behaviour at a higher lever *subsume* behaviors at a lower level.
- Higher behaviour can also use computed percepts and intermediary or final output values of any lower lever behaviour
- Layers of competence developed from bottom up
  - Starting with N=0, N layer behaviour developed and *debugged* before next layer is added
- Control using an N+1 layer behavour can *revert* to using the prior N layer behaviour when N+1st layer *cannot determine* the next action.

Example subsumption layered behaviour

Imperial College London

- Level 0 behaviour – run away from nearby objects
- Level 1 behaviour - wander avoiding objects
  - uses perception value computed by runaway
  - overrides simple runaway action except for its emergency stop
- Level 2 behaviour - travel down center of a corridor avoiding objects
  - Overrides wander behaviour except for its emergency stop
  - Reverts to wander when walls cannot be 'perceived'

Programming reactive robots - Teleo-reactive programs

Imperial College London

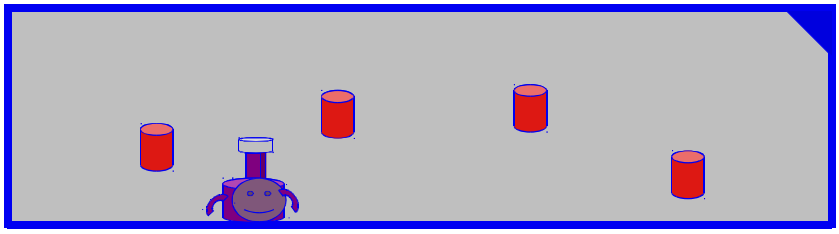
- Production rule notation for controlling behaviour.
- Inspired by control theory ideas of continuous monitoring and persistent responses - like the Braitenberg vehicles.
- But has parameterized procedures and recursion - so quite rich program structuring tools.
- Due to Nilsson, 1994.

`http://ai.stanford.edu/users/nilsson/trweb/tr.html`

Imperial College London

Motivating example: Can collecting robot

- Enclosed flat space with empty red soft drinks cans scattered about and a corner painted blue



- There are no other objects in the space.
- Robot is to locate, grab and deliver a red can to the blue corner.
- It then waits until can is removed by a human.
- Then repeats the task.
- New red cans can be placed in space at any time.

Imperial College London

TR Programs Example: Percepts and actions

- Robot has a forward camera, obstacle detecting sensors, and a gripper.

- It has a percept routine for the camera:

  see(D,C) **C** coloured blob at position D (**left**, **center** or **right**)

- By combining readings of obstacle sensors and camera it has percepts:

  at(blue) in the blue corner

  near(C) close to a **C** coloured thing

- Using its gripper sensors and obstacle detecting sensors it has percepts:

  touch touching something

  holding something between grippers

- It has action routines that enable it to:

  move(S) move forward at speed S

  turn(D,S) turn to D (left or right) at rotation speed S

  hold/drop close/open gripper

Teleo-Reactive Can Collector

Imperial College London

```
collect_cans..{
  at(blue), see(_,blue),see(_,red)  ⇒ nil.
  holding  ⇒ deliver_can.
  true ⇒ get_can.
}.
get_can ..{
   touching ⇒ hold.
   see(_,red),near(red) ⇒  approach(red,0.5,0.1).
   see(_,red) ⇒ approach(red,1.5,0.3).
   true ⇒ turn(right, 0.5).
}.
```

Can Collector (continued)

Imperial College London

```
deliver..{
   holding, at_blue ⇒ drop
   see(_,blue), near(blue)    ⇒ approach(blue,0.5,0.1).
   see(_,blue) ⇒ approach(blue,1.5,0.3).
   true ⇒ turn(right, 0.5).
}.

approach(C,FS,TS)..{
   see(center,C) ⇒ move(FS).
   see(Side,C)    ⇒ move(FS), turn(Side,TS).
   true ⇒ nil
}.
```

concurrent actions

General form of Teleo-Reactive procedure

Imperial College London

p(….)..{

$K_1 \Rightarrow A_1 \sim\sim U_1$.

$K_2 \Rightarrow A_2 \sim\sim U_2$.

…

$K_m \Rightarrow A_m \sim\sim U_m$.

}

Each K_i is a boolean test on beliefs (perceptual level or inferred)

Each Ai is a set of concurrent actions:

primitive actions,

calls to TR procedures (even a recursive call to p),

optional U_i memory update actions executed once each time rule becomes fired rule

Informal rules of evaluation

Imperial College London

- **First rule** with true condition is **fired**
- Actions are by default **durative** (eg move_forward)
  - Action of fired rule persists until another rule becomes first rule with a true condition and is fired.
- ***Rule conditions are continuously re-evaluated***
  - in practice re-evaluated only when a belief that might effect choice of rule to fire is changed.
- When action of fired rule is T-R procedure call
  - Rules of *calling* procedure **still tested.**
  - Called proc. **exited** as soon as calling rule no longer fired rule.
  - In a multi-threaded implementation each called TR procedure has its own thread of control.

Exit from called procedure

Imperial College London

p(….)..{
$K_1 \Rightarrow A_1$
$K_2 \Rightarrow q(\ldots)$
...
$K_m \Rightarrow A_m$
}

q exited as soon as:
K_1 becomes *true*
or K_2 becomes *false* and
some K_i, i>2 becomes true

q(….)..{
$K'_1 \Rightarrow A'_1$
$K'_2 \Rightarrow A'_2$
...
}

Desirable features of TR programs

Imperial College London

Regression property

for each rule $K_i \Rightarrow A_i$, i>1

the execution of A_i normally results in a state of the world in which the test of an earlier rule will be true

Completeness

In each procedure there is always one rule that can be fired

- can be guaranteed by having a last default rule with condition **true**

Special T-R programs

Imperial College London

- T-R rule sequence is *complete* if the disjunction of tests

  K_1 or K_2 or … K_m

  is a tautology (always true)

- T-R rule sequence is *universal* if

  It is complete and satisfies the regression property

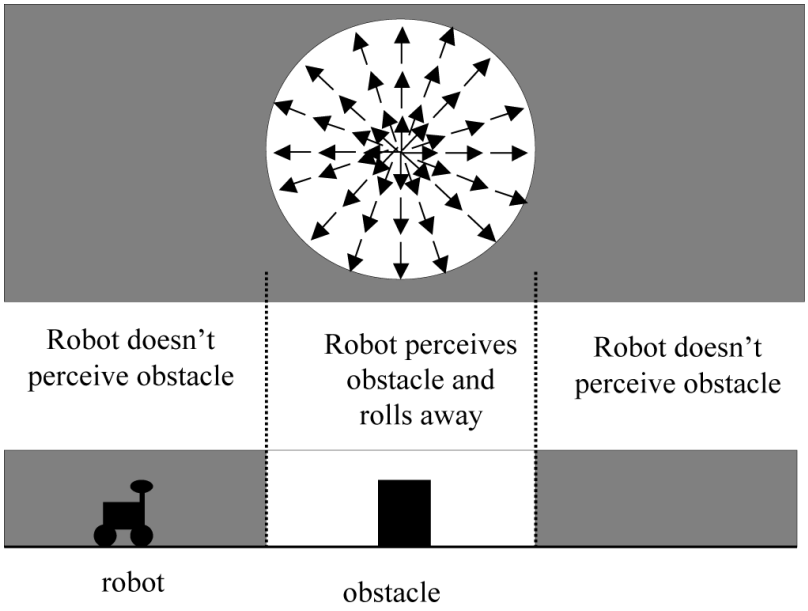
Merging Behaviours using Vectors

Imperial College London

- The motor schema component of a behavior (the action function) always represents its action as a vector.
- The action outputs of the different behaviors are then combined using vector summation to give a combined action response.
- From each behavior, the robot computes a vector
  - Magnitude related to detected strength of stimulus
  - Direction related to position of stimulus relative to the robot
- But we visualize the behaviour as a field, where every point in space has associated vector that is the response vector for the robot *if it were at that point*
- Magnitude of the vector usually interpreted as a speed indicator, and direction of the vector as the direction robot should be traveling at that point *for that behaviour*
- It is a *uniform representation* of actions allowing for easy *merging* of behaviours

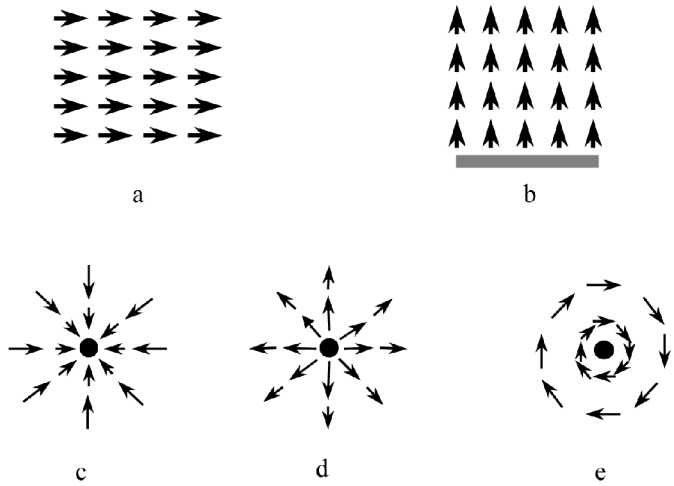
Imperial College London

Example: Run Away via Repulsion



Imperial College London

Some Primitive Potential Fields



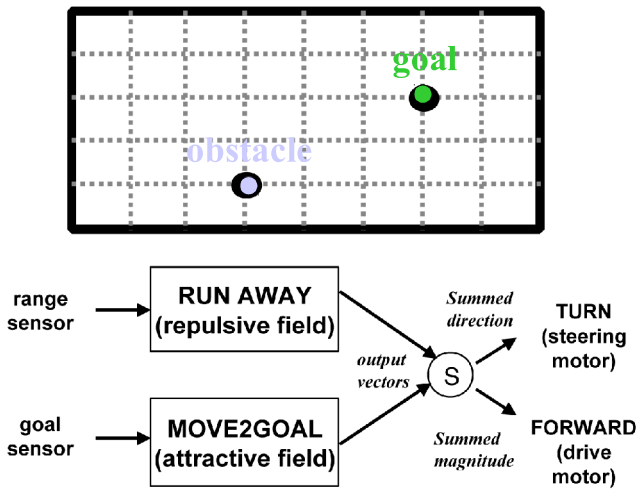
Imperial College London

Common fields in behaviors

- Uniform
  - Move in a particular direction, corridor following
- Repulsion
  - Runaway (obstacle avoidance)
- Attraction
  - Move to goal
- Perpendicular
  - Corridor following
- Tangential
  - Move through a door skirting one side of the door, explore an object

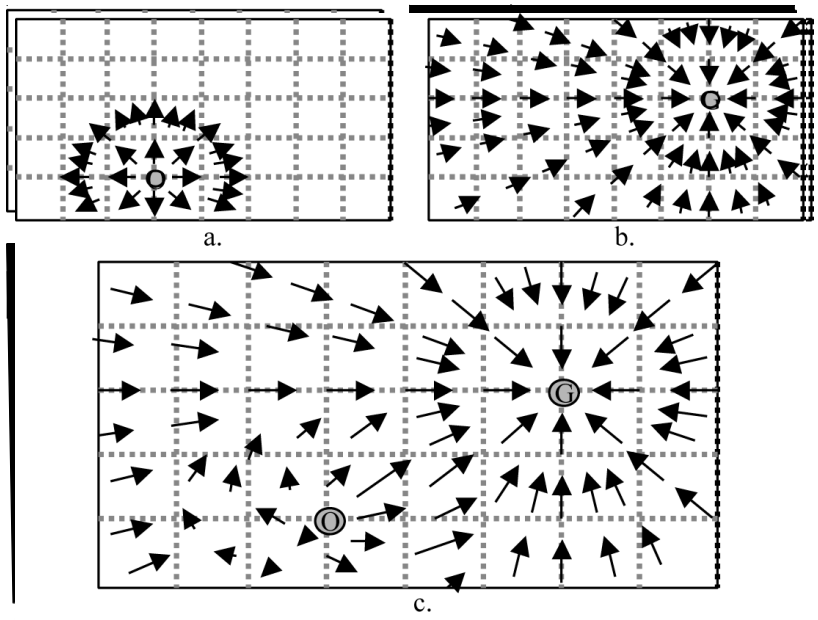
Imperial College London

Combining Fields for Emergent Behavior



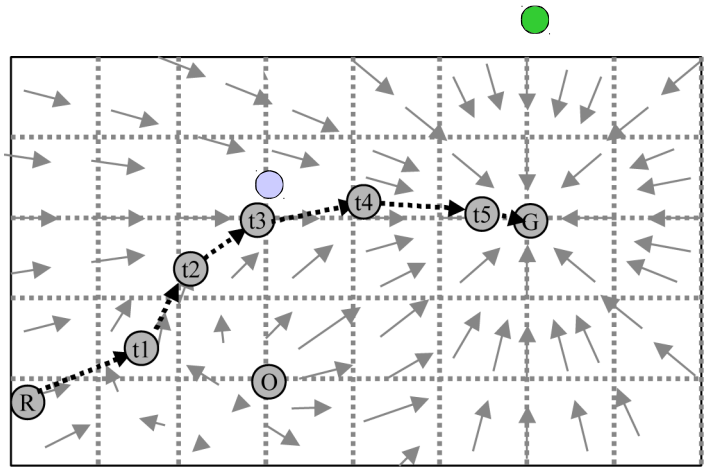
Imperial College London

Fields and Their Combination



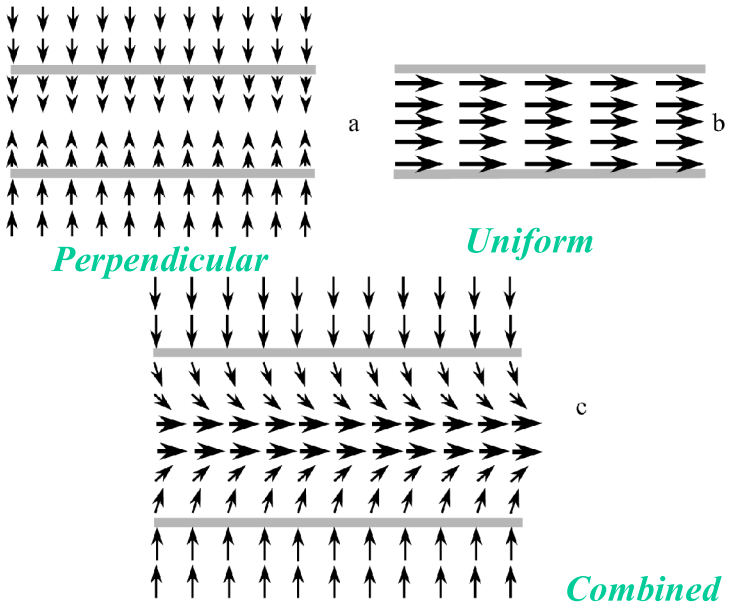
Imperial College London

Possible Path Taken



Example: follow-corridor or follow-sidewalk

Imperial College London



Vector fields

Imperial College London

- Advantages
  - Easy to visualize
  - Easy to build up software libraries
  - Combination mechanism is simple
  - Can be tweaked using weighted sums that can give higher weighting to the action vectors for certain behaviours
- Disadvantages
  - Local minima problem (sum to magnitude=0)
  - Jerky motion if not careful
  - Emergency stop behaviour has to be specially handled

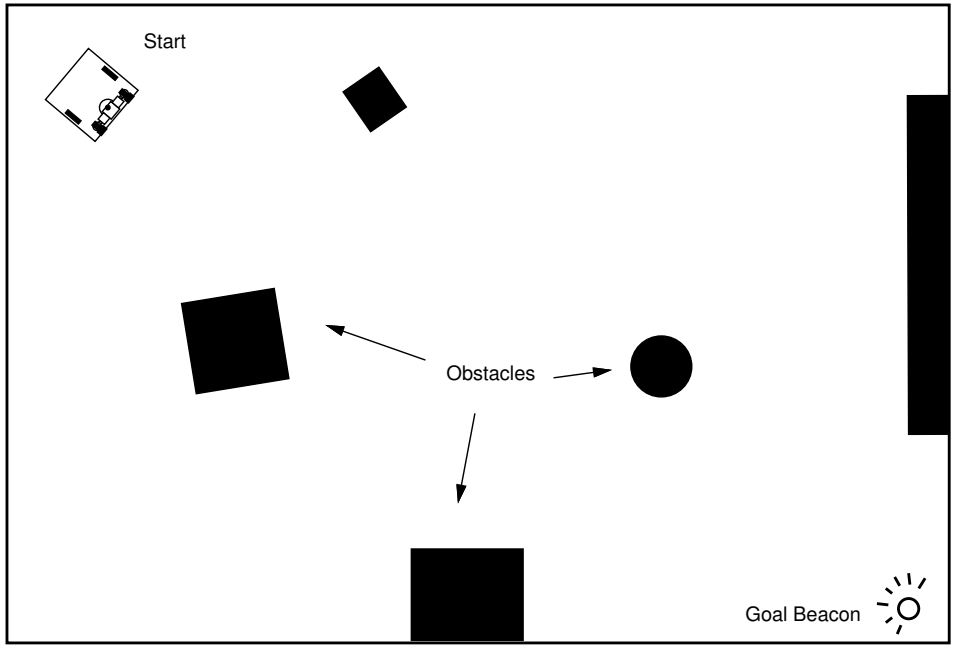
Robotics

Lecture 5: Probabilistic Robotics

See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

Andrew Davison
Department of Computing
Imperial College London

Review: Obstacle Course Practical from Last Week



Tackled using high-mounted light sensors, low touch sensors.

Review: Obstacle Course Practical

- A single TR program without procedures could implement a solution via its drop-through nature:

```
p {
        bump_left          ⇒back_up_right
        bump_right         ⇒back_up_left
        bump_both          ⇒back_up_random
        see_light_left     ⇒forward_and_right
        see_light_right    ⇒forward_and_left
        see_light_ahead    ⇒forward
        true               ⇒turn_on_spot
}
```

Review: Obstacle Course Practical

- Better to construct a proper TR program with the regression and completeness properties which implements subsumption using sub-procedures. For instance, it could have a top-level structure like this:

```
p {
        light_in_front      ⇒advance_to_light
        true                ⇒orient_towards_light
}

advance_to_light {
        no_bump             ⇒servo_towards_light
        true                ⇒avoid_obstacle
}
```

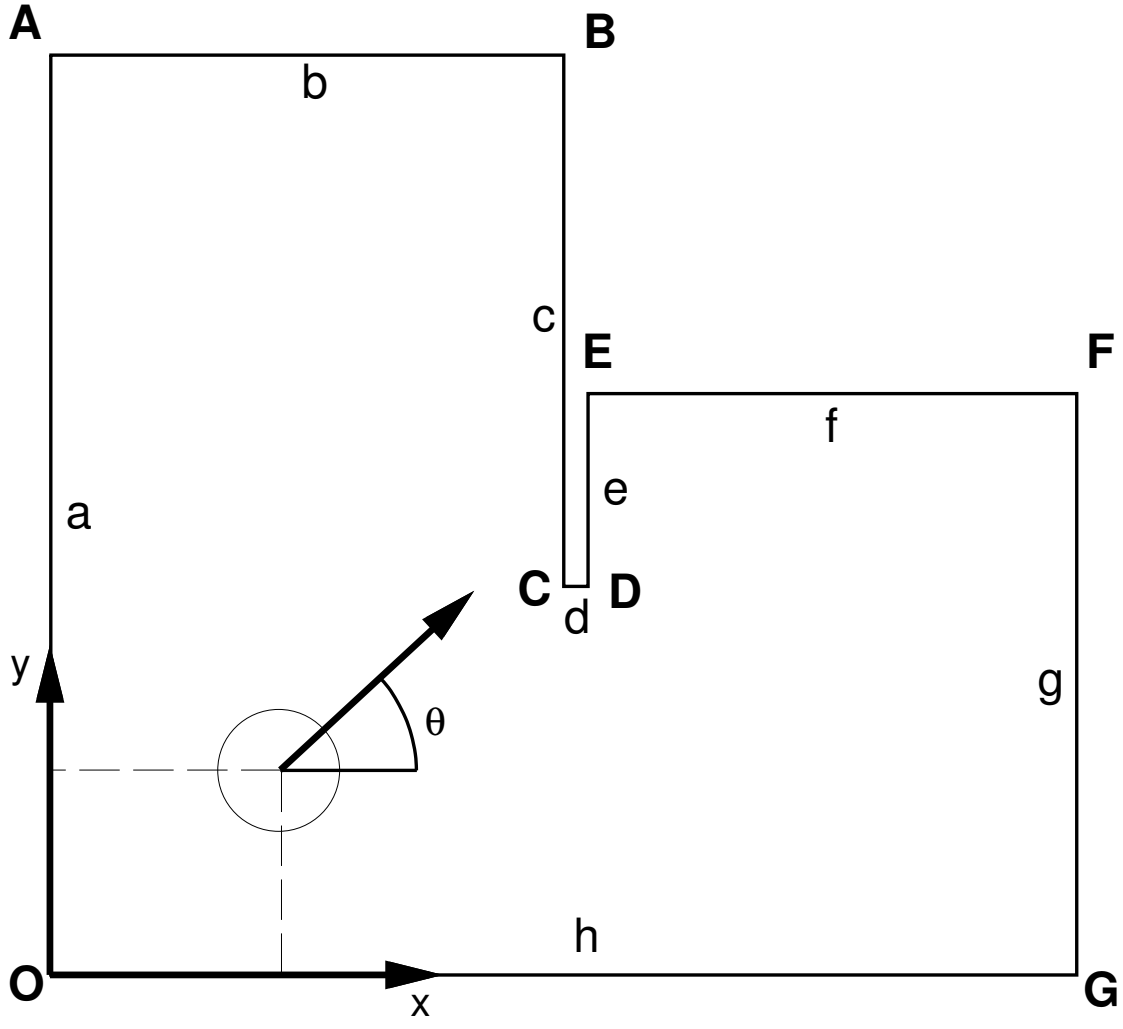
- p *subsumes* advance_to_light, which in turn subsumes avoid_obstacle and servo_towards_light, each themselves TR procedures.

Obstacle Course: 2007 Competition

- Winning robot by Philip Stennett, Nicholas Ball, Maurice Adler and Wei Chieh Soon.

Probabilistic Localisation

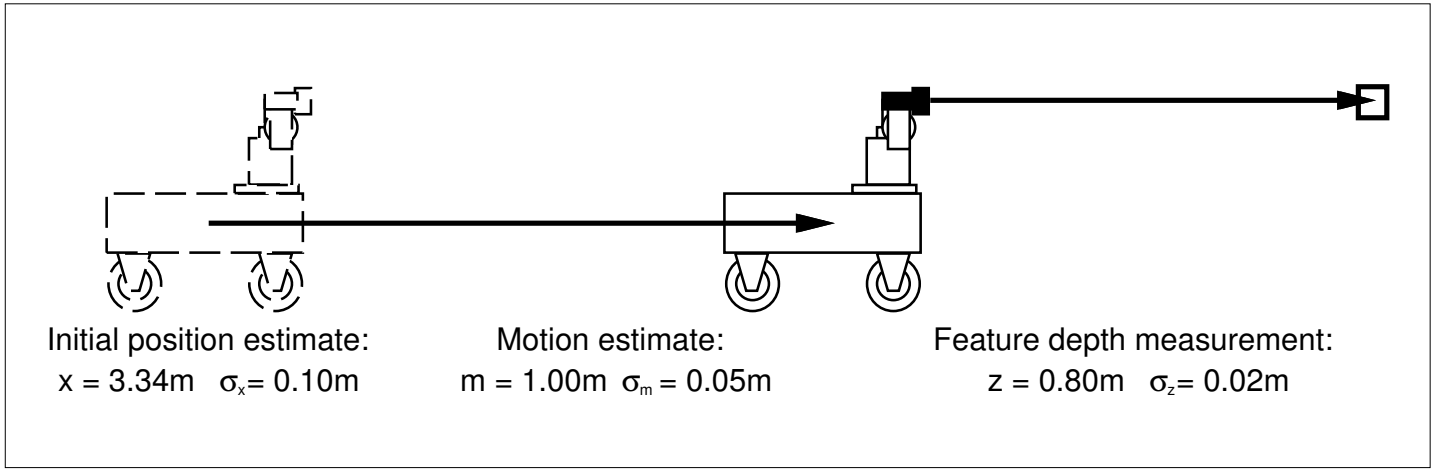
Over the next two weeks we will aim at much more reliable navigation: possible if the robot actually *knows where it is* relative to a map.



Probabilistic Robotics

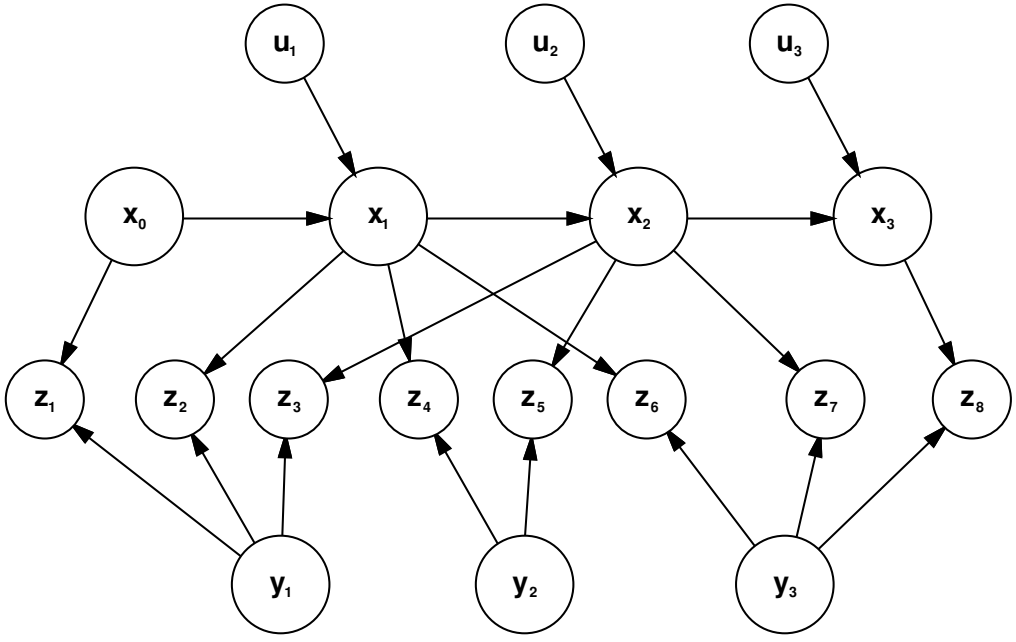
- Problem: simple sensing/action procedures can be locally effective but are limited in complicated problems in the real-world.
- 'Classical AI' approaches based on logical reasoning about true/false statements fall down when presented with real-world data.
- Why?
  - Advanced sensors don't lend themselves to straightforward analysis like bump and light sensors.
  - All information which a robot receives is uncertain.
- A probabilistic approach acknowledges uncertainty and uses models to abstract useful information from data.

Uncertainty in Robotics



- Every robot action is uncertain.
- Every sensor measurement is uncertain.
- Every state estimate is uncertain.

Probabilistic Inference



- What is my state and that of the world around me?
- Prior knowledge is combined with new measurements.
- A series of weighted combinations of old and new information.
- Sensor fusion: the general process of combining data from many different sources into useful estimates.
- This composite state estimate can then be used to decide on the robot's next action.

Bayesian Probabilistic Inference

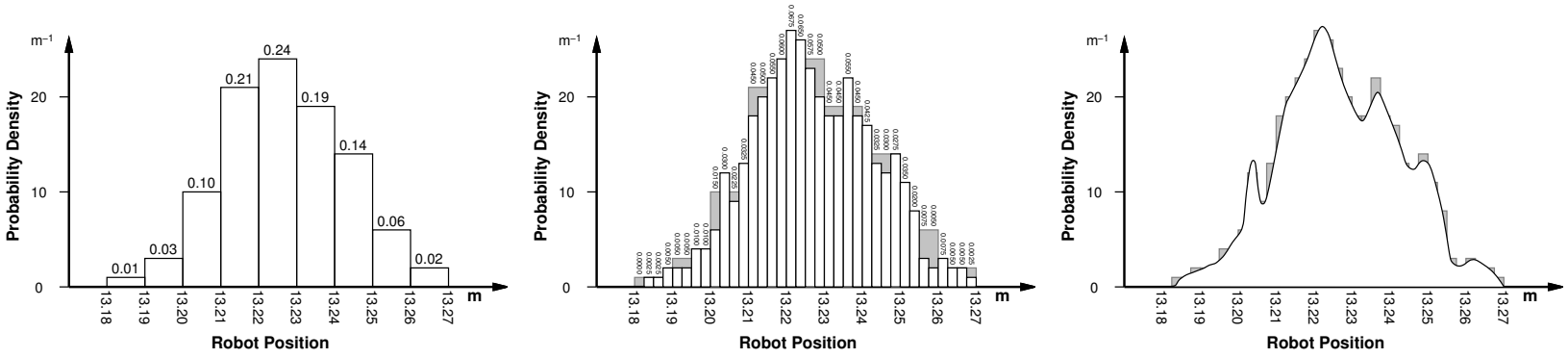
- 'Bayesian' has come to be known as a certain view of the meaning of probability theory as a measure of subjective belief.
- Probabilities describe our state of knowledge — nothing to do with randomness in the world.
- The absolute framework for managing uncertainty.
- Bayes' Rule relating probabilities of discrete statements:

$$\begin{aligned} P(\mathbf{XZ}) &= P(\mathbf{Z}|\mathbf{X})P(\mathbf{X}) = P(\mathbf{X}|\mathbf{Z})P(\mathbf{Z}) \\ \Rightarrow P(\mathbf{X}|\mathbf{Z}) &= \frac{P(\mathbf{Z}|\mathbf{X})P(\mathbf{X})}{P(\mathbf{Z})} \end{aligned}$$

- Here $P(\mathbf{X})$ is the prior; $P(\mathbf{Z}|\mathbf{X})$ the likelihood; $P(\mathbf{X}|\mathbf{Z})$ the posterior; $P(\mathbf{Z})$ sometimes called marginal likelihood.

Probability Distributions: Discrete and Continuous

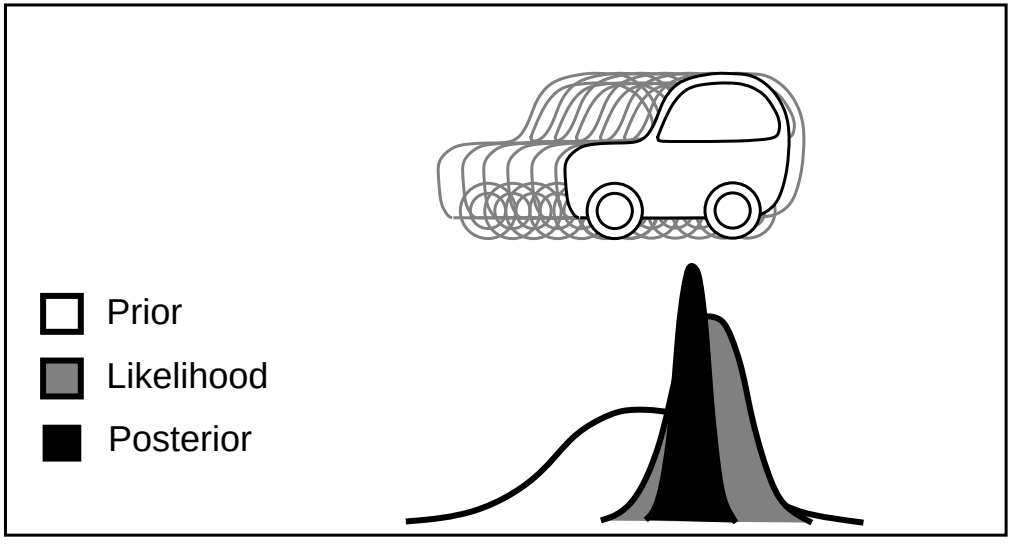
- A continuous Probability Density Function $p(x)$ is the limiting case as the widths of bins in a discrete histogram tend to zero.



- The probability that a continuous parameter lies in the range a to b is given by the area under the curve:

$$P_{a \to b} \int_a^b p(x)dx$$

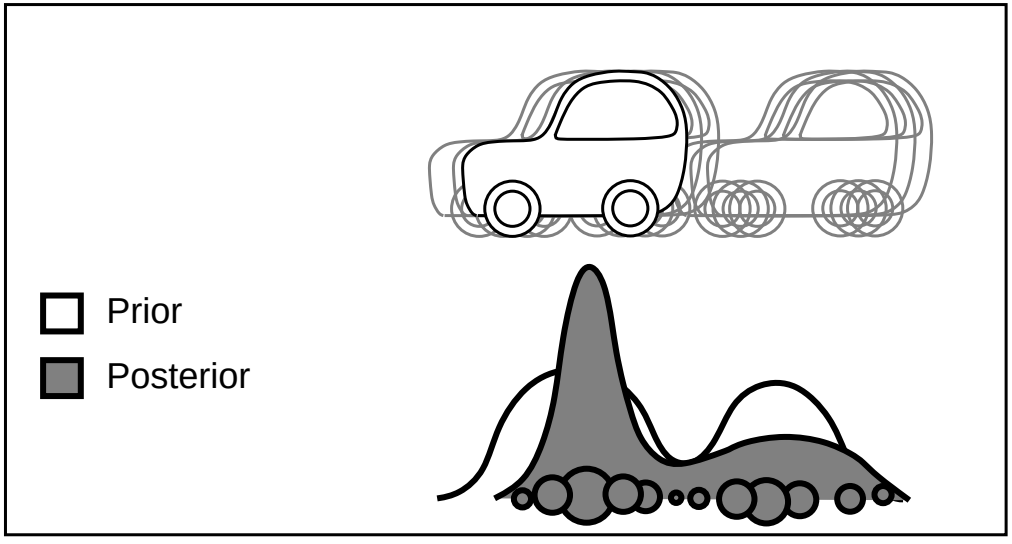
Probability Representations: Gaussians



$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Explicit Gaussian (or normal) distributions are often represent the uncertainty in sensor measurements very well.
- Wide Gaussian *prior* multiplied by *likelihood* curve to produce a *posterior* which is tighter than either.

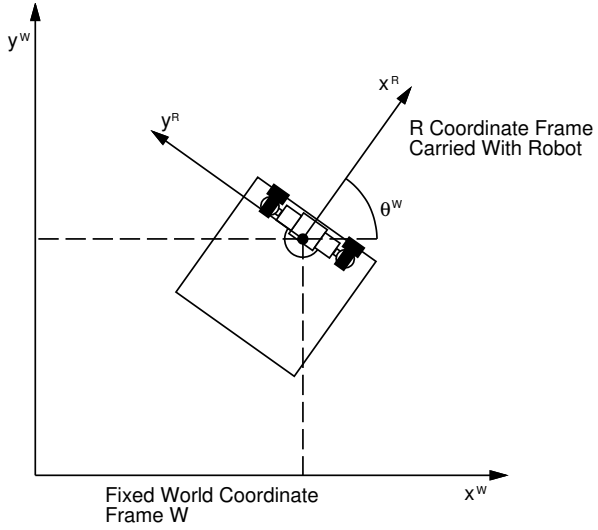
Probability Representations: Particles

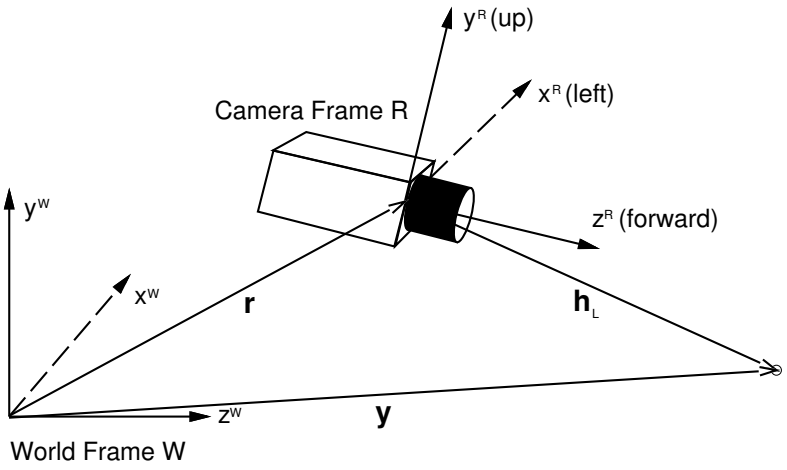


- Here a probability distribution is represented by a finite set of *weighted samples* of the state $\{\mathbf{x}_i, w_i\}$, where $\sum_i w_i = 1$.
- Big advantage is the ability to represent *multi-modal* distributions (with more than one peak) in ambigiuous situations.
- Poor ability to represent detailed shape of distribution when number of particles is low.

Probabilistic Localisation

- The robot has a map of its environment in advance.
- The only uncertain thing is the position of the robot.

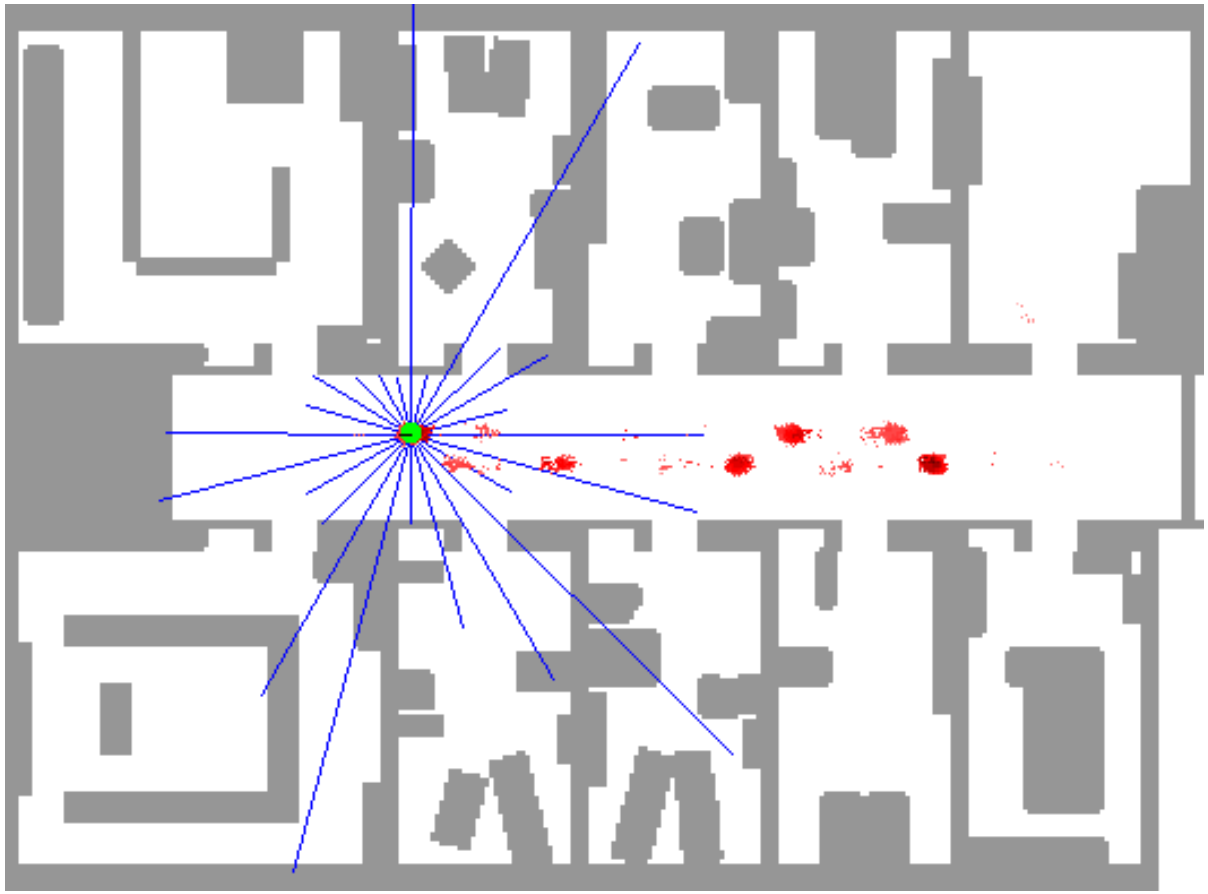




- The robot stores and updates a *probability distribution* representing its uncertain position estimate.

Monte Carlo Localisation (Particle Filter)

- Cloud of particles represent uncertain robot state: more particles = more probability.



(Dieter Fox *et al.*1999, using sonar)

The Particle Distribution

- A particle is a point estimate $\mathbf{x}_i$ of the state (position) of the robot with a weight w_i.

$$\mathbf{x}_i = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix}$$

- The full particle set is:

$$\{\mathbf{x}_i, w_i\} \ ,$$

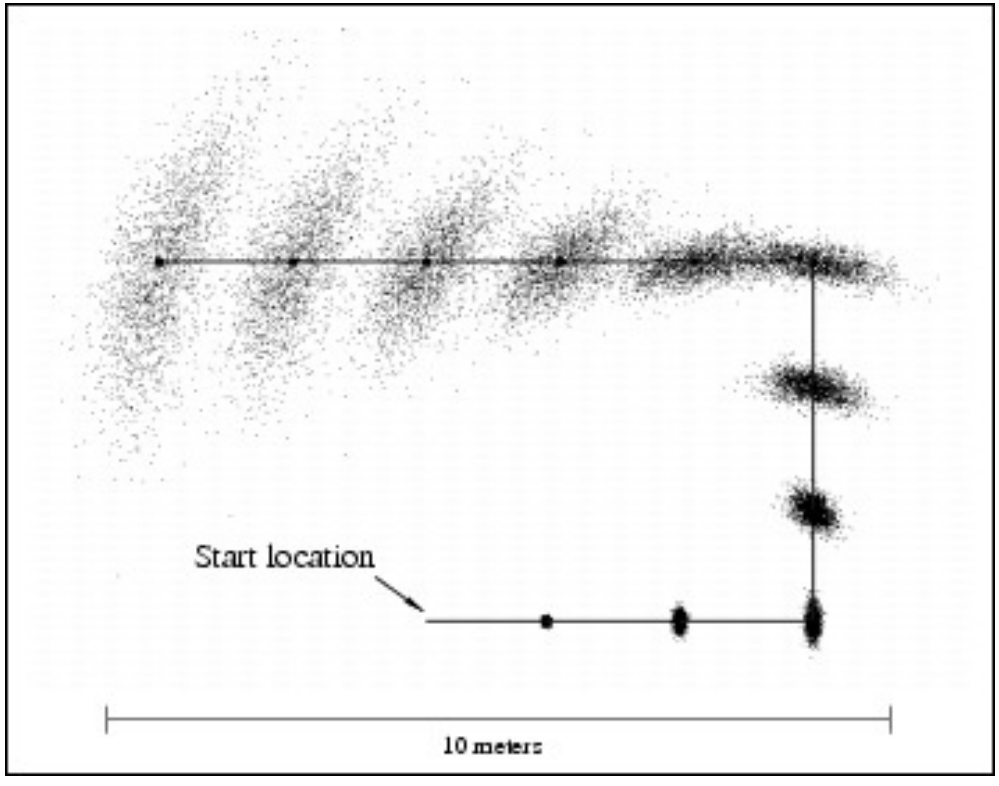
  for $i = 1$ to N. A typical value might be $N = 100$.

- All weights should add up to 1. If so, the distribution is said to be *normalised*:

$$\sum_{i=1}^{N} w_i = 1 \ .$$

Displaying a Particle Set

- We can visualise the particle set by plotting the x and y coordinates as a set of dots; more difficult to visualise the θ angular distribution (perhaps with arrows?) — but we can get the main idea just from the linear components.

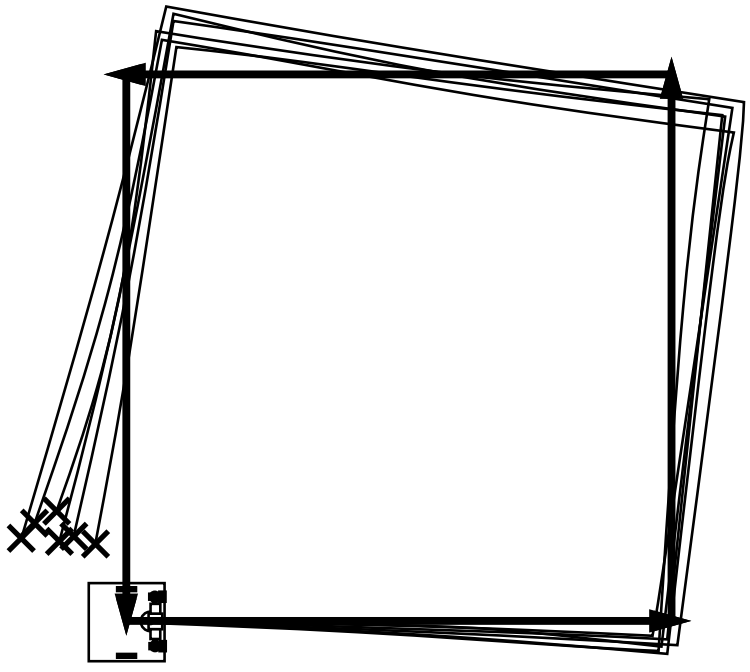


Steps in Particle Filtering

These steps are repeated every time the robot moves a little and makes measurements:

1. Motion Prediction based on Proprioceptive Sensors.
2. Measurement Update based on Outward-Looking Sensors.
3. Normalisation.
4. Resampling.

Motion Prediction



- We know that uncertainty grows during blind motion.
- So when the robot makes a movement, the particle distribution needs to shift its mean position but also spread out.
- We achieve this by passing the state part of each particle through a function which has a deterministic component and a random component.

Motion Prediction

- During a straight-line period of motion of distance D:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + (D + e)\cos\theta \\ y + (D + e)\sin\theta \\ \theta + f \end{pmatrix}$$

- During a pure rotation of angle angle α:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$$

- Here e, f and g are zero mean *noise* terms — i.e. random numbers typically with a Gaussian distribution.
- Watch out for angular wrap-around — i.e. make sure that θ values are always in the range 0° to 360°.

Measurement Updates

- A measurement update consists of applying Bayes Rule to each particle; remember:

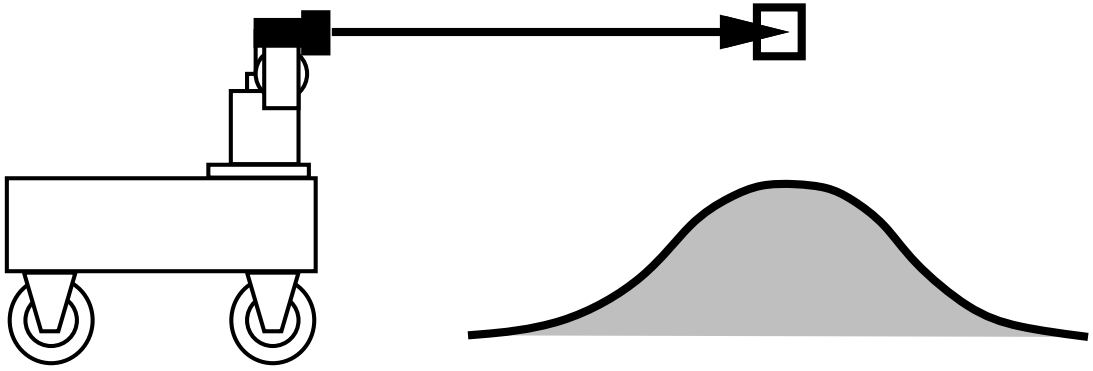
$$P(\mathbf{X}|\mathbf{Z}) = \frac{P(\mathbf{Z}|\mathbf{X})P(\mathbf{X})}{P(\mathbf{Z})}$$

- So when we achieve a measurement z, we update the weight of each particle as follows:

$$w_{i(new)} = P(z|\mathbf{x}_i) \times w_i \ ,$$

  remembering that the denominator in Bayes' rule is a constant factor which we do not need to calculate because it will later be removed by normalisation.
- $P(z|\mathbf{x}_i)$ is the *likelihood* of particle i; the probability of getting measurement z given that it represents the true state.

Likelihood Function



- The form of a likelihood function comes from a probabilistic model of the outward-looking sensor.
- Having calibrated a sensor and understood the uncertainty in its measurements we can build a probabilistic measurement model for how it works. This will be a probability distribution (specifically a likelihood function) of the form:

$$P(z|\mathbf{x}_i)$$

Such a distribution will often have a Gaussian shape.

Robotics

Lecture 6: Monte Carlo Localisation

See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

Andrew Davison
Department of Computing
Imperial College London

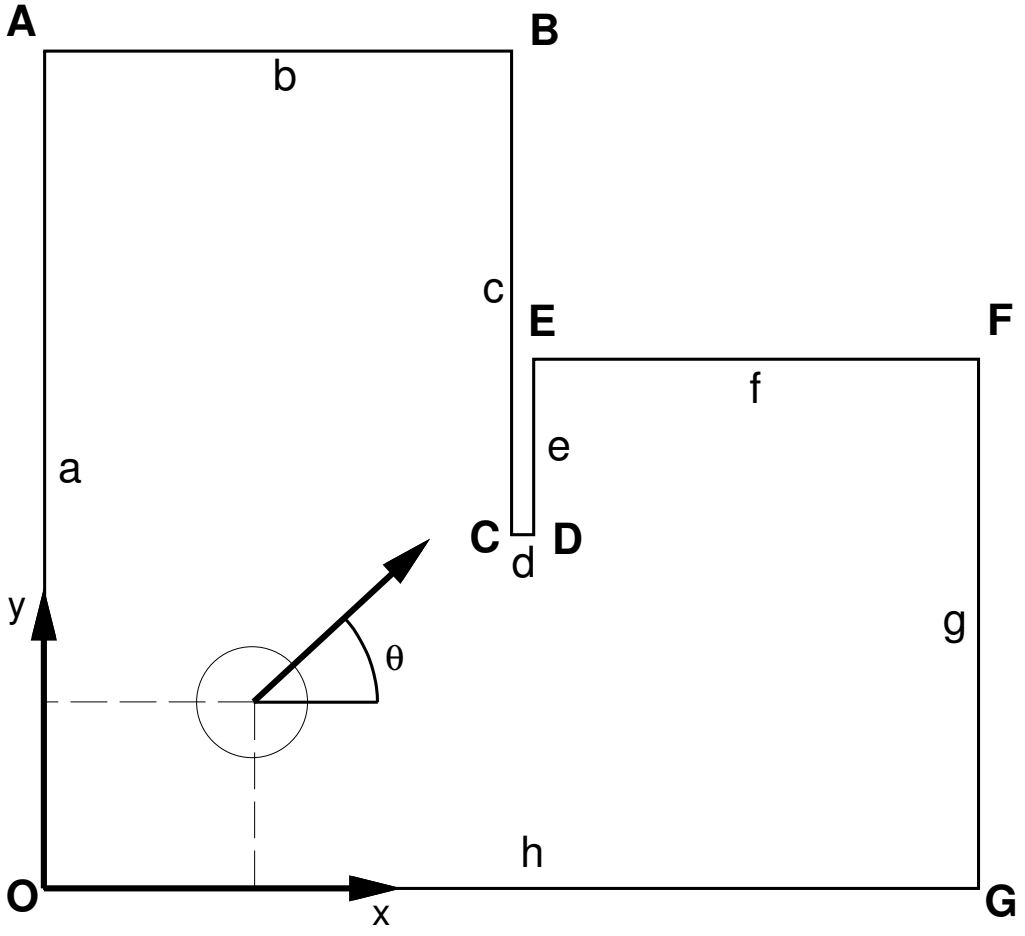
Review: Probabilistic Motion and Sensing Practical 5

The preliminaries for Monte Carlo Localisation:

- Updating a probability distribution, using particles, to represent motion uncertainty.
- Understanding probabilistically the performance of the sonar sensor.
- (Waypoint-based navigation).

Monte Carlo Localisation (MCL)

In Practical 6 we will aim at repeatable localisation within an environment like this for which I will give you a map:



Monte Carlo Localisation (MCL)

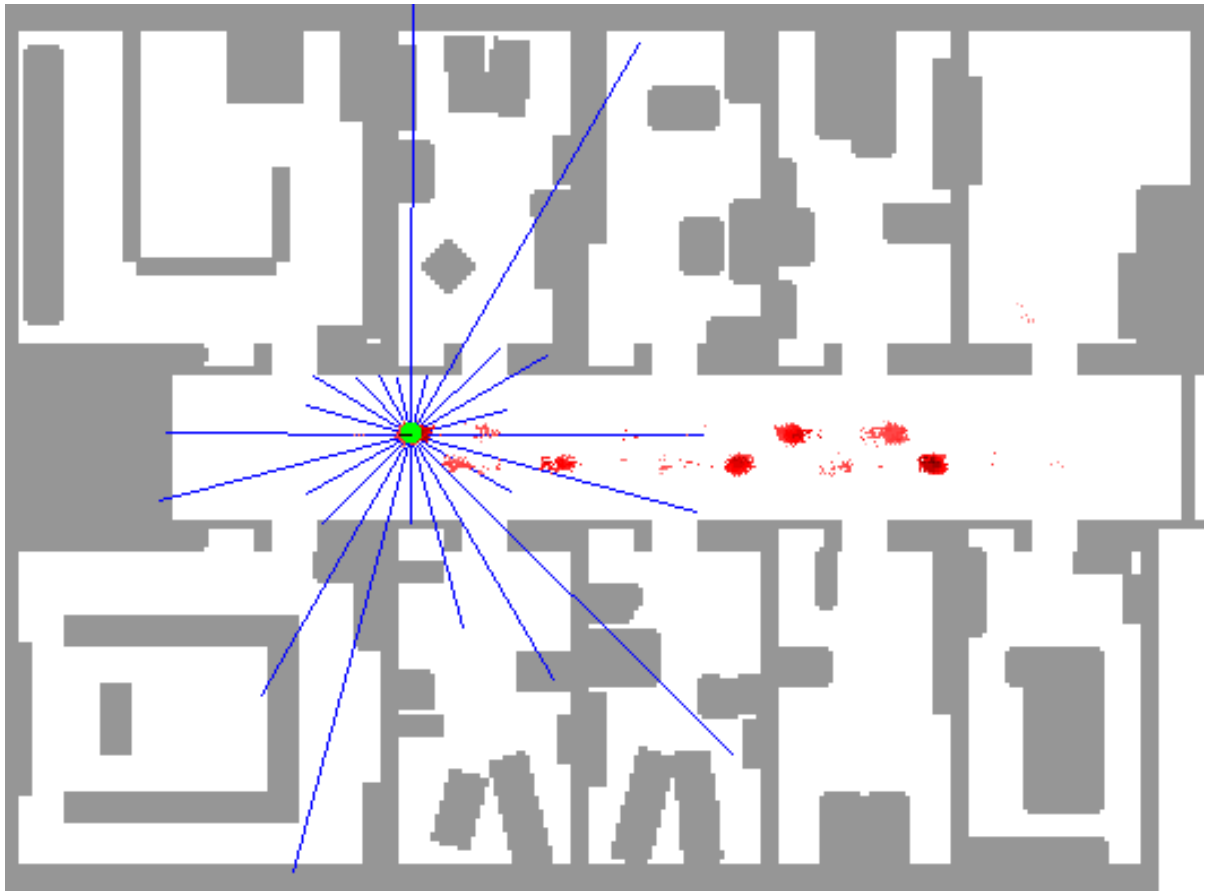
- In MCL, a cloud of weighted particles represents the uncertain position of a robot.

Two ways of thinking about how MCL works:

- A Bayesian probabilistic filter.
- 'Survival of the fittest', a type of genetic algorithm.

MCL Successful Large-Scale Implementation

- (Dieter Fox *et al.*1999, using ring of sonar sensors)



The Particle Distribution

- A particle is a point estimate $\mathbf{x}_i$ of the state (position) of the robot with a weight w_i.

$$\mathbf{x}_i = \begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix}$$

- The full particle set is:

$$\{\mathbf{x}_i, w_i\} \ ,$$

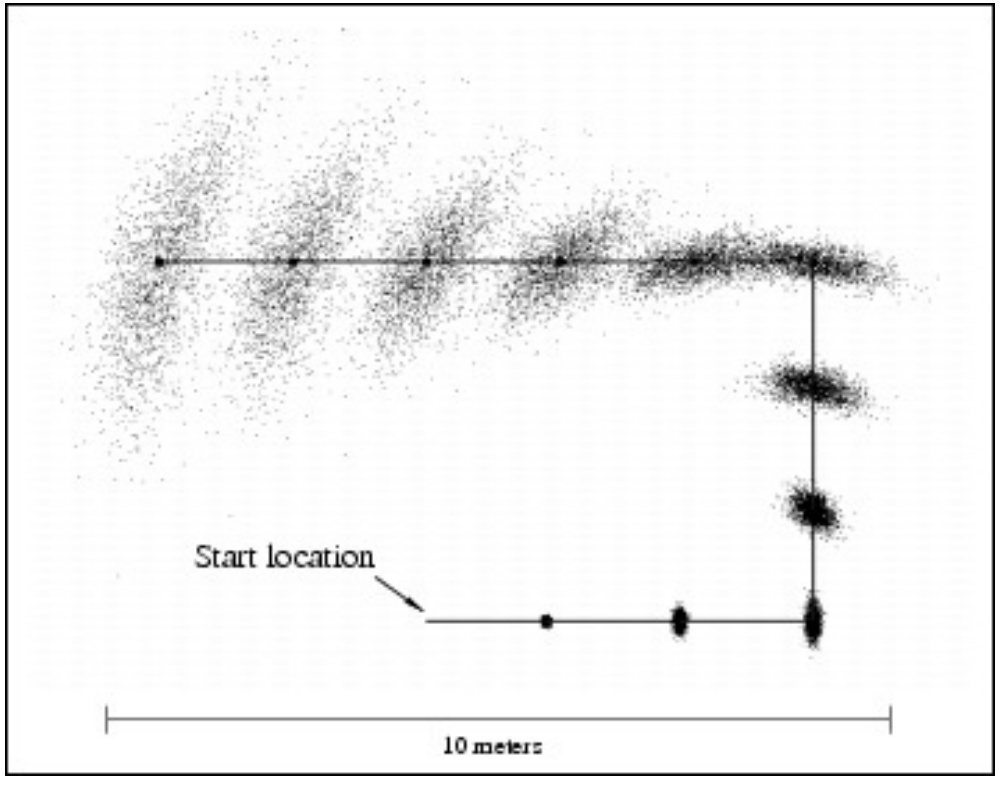
  for $i = 1$ to N. A typical value might be $N = 100$.

- All weights should add up to 1. If so, the distribution is said to be *normalised*:

$$\sum_{i=1}^{N} w_i = 1 \ .$$

Displaying a Particle Set

- We can visualise the particle set by plotting the x and y coordinates as a set of dots; more difficult to visualise the θ angular distribution (perhaps with arrows?) — but we can get the main idea just from the linear components.

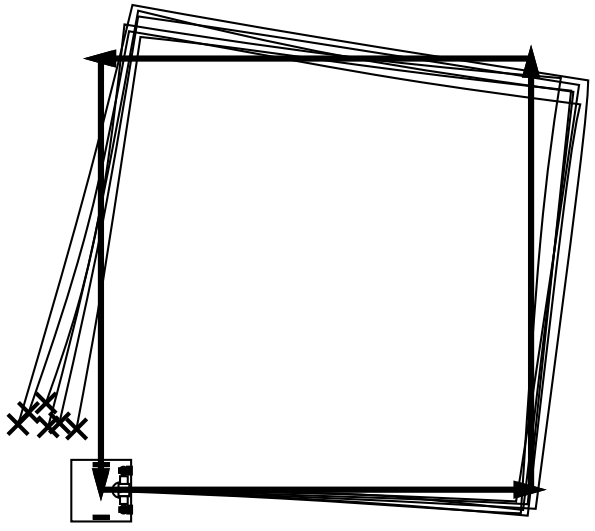


Steps in Particle Filtering

These steps are repeated every time the robot moves a little and makes measurements:

1. Motion Prediction based on Odometry
2. Measurement Update based on Outward Looking Sensors (Sonar)
3. Normalisation
4. Resampling

Motion Prediction



- We know that uncertainty grows during blind motion.
- So when the robot makes a movement, the particle distribution needs to shift its mean position but also spread out.
- We achieve this by passing the state part of each particle through a function which has a deterministic component and a random component.

Motion Prediction

- During a straight-line period of motion of distance D:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x + (D + e)\cos\theta \\ y + (D + e)\sin\theta \\ \theta + f \end{pmatrix}$$

- During a pure rotation of angle angle α:

$$\begin{pmatrix} x_{new} \\ y_{new} \\ \theta_{new} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta + \alpha + g \end{pmatrix}$$

- Here e, f and g are zero mean *noise* terms — i.e. random numbers *sampled from* with a Gaussian distribution.
- The spreading out comes from sampling a different set of random numbers for each particle.

Motion Prediction

- Watch out for angular wrap-around — i.e. make sure that θ values are always in the range 0° to 360°.
- Solve this by simply adding or subtracting 360° to any particles which go out of this range.

Measurement Updates

- A measurement update consists of applying Bayes Rule to each particle; remember:

$$P(\mathbf{X}|\mathbf{Z}) = \frac{P(\mathbf{Z}|\mathbf{X})P(\mathbf{X})}{P(\mathbf{Z})}$$

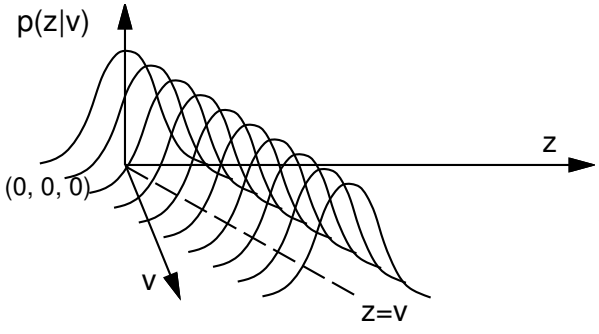
- So when we achieve a measurement z, we update the weight of each particle as follows:

$$w_{i(new)} = P(z|\mathbf{x}_i) \times w_i \ ,$$

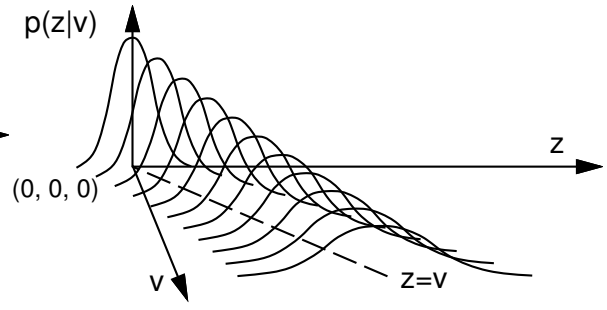
  remembering that the denominator in Bayes' rule is a constant factor which we do not need to calculate because it will later be removed by normalisation.
- $P(z|\mathbf{x}_i)$ is the *likelihood* of particle i; the probability of getting measurement z given that $\mathbf{x}_i$ represents the true state.

Likelihood Functions

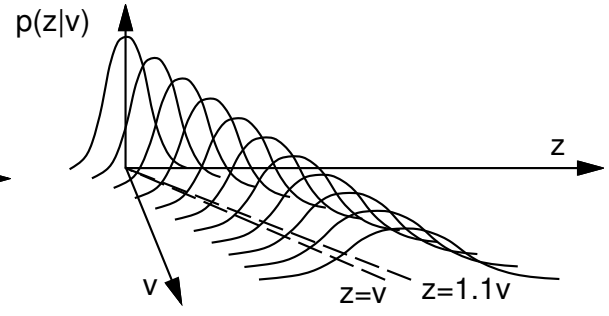
- A likelihood function fully describes a sensor's performance.
- $p(z|v)$ is a function of both measurement variables z and ground truth v and can be plotted as a probability surface. e.g. for a depth sensor:



Constant Uncertainty

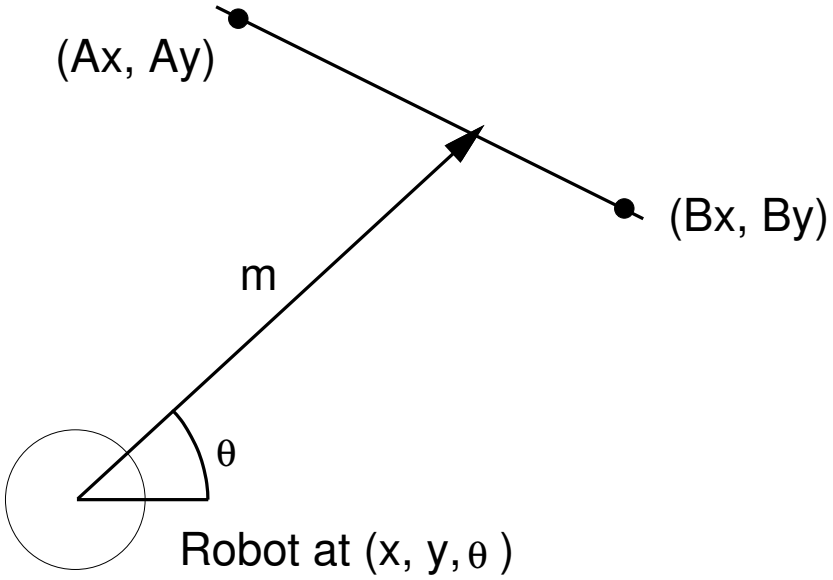


Growing



Systematic Error (Biased)

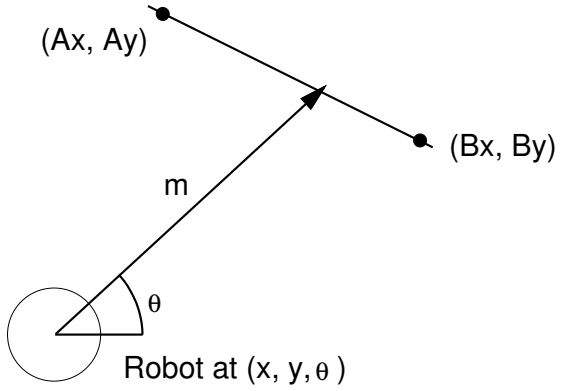
Measurement Update: Sonar



- If robot is at pose (x, y, θ) then its forward distance to an infinite wall passing through (A_x, A_y) and (B_x, B_y) is:

$$m = \frac{(B_y - A_y)(A_x - x) - (B_x - A_x)(A_y - y)}{(B_y - A_y)\cos\theta - (B_x - A_x)\sin\theta} \ .$$

Measurement Update: Sonar



- The world coordinates at which the forward vector from the robot will meet the wall are:

$$\begin{pmatrix} x + m\cos\theta \\ y + m\sin\theta \end{pmatrix} .$$

  Using this you can check if the sonar should hit between the endpoint limits of the wall.
- If the sonar would independently hit several of these walls, obviously the closest is the one it will actually respond to.

Likelihood for Sonar Update

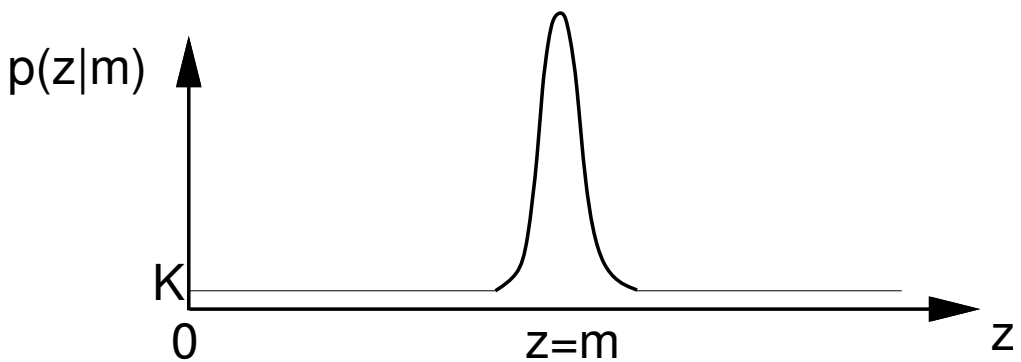
- Again, the likelihood should depend on the difference $z - m$: if this is small, then the measurements validates a particle; if it is big it does not and weakens the particle.
- A Gaussian function is a suitable form for this with standard deviation depending on sonar uncertainty.

$$p(z|m) \propto e^{\frac{-(z-m)^2}{2\sigma_s^2}}$$

- Note the difference between this and the motion prediction step. There we *sampled* randomly from a Gaussian distribution to move each particle by a slightly different amount. Here in the measurement update we just *read off* a value from a Gaussian function to obtain a likelihood for each particle.

Robust Likelihood for Sonar Update

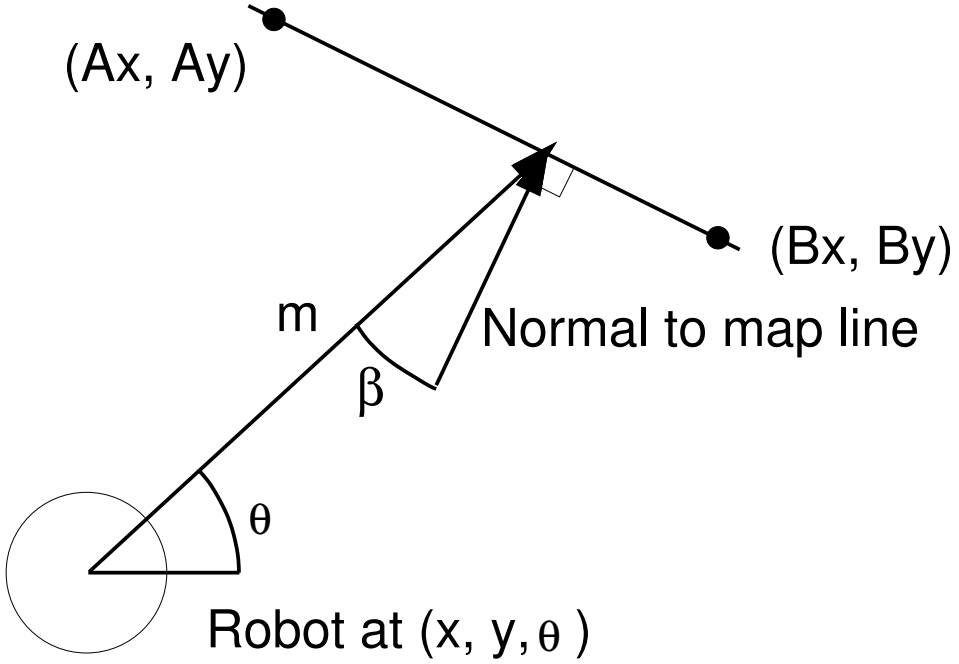
- A *robust* likehood function models the fact that real sensors sometimes report 'garbage' values which are not close to ground truth. Robust functions have 'heavy tails'. This can be achieved most simply by adding a constant to the likelihood function. The meaning of this is that there is some constant probability that the sensor will return a garbage value, uniformly distributed across the range of the sensor.



$$p(z|m) \propto e^{\frac{-(z-m)^2}{2\sigma_s^2}} + K$$

- The effect of a robust likelihood function in MCL is that the filter is less aggressive in 'killing off' particles which are far from agreeing with measurements. An occasional garbage measurement will not lead to the sudden death of all of the particles in good positions.

Likelihood for Sonar Update



- Also relevant may be the angle between the sonar direction and the normal to the wall. If this is too great, probably the sonar will not give a sensible reading and you should ignore it.

$$\beta = cos^{-1}\left(\frac{\cos\theta(A_y - B_y) + \sin\theta(B_x - A_x)}{\sqrt{(A_y - B_y)^2 + (B_x - A_x)^2}}\right)$$

Normalisation

- The weights of all particles should be scaled so that they again add up to 1.
- Calculate $\sum_{i=1}^{N} w_i$ and divide all existing weights by this:

$$w_{i(new)} = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

Resampling

- Resampling consists of generating a new set of N particles which all have equal weights $1/N$, but whose spatial distribution now reflects the probability density.
- To generate each of the N new particles, we copy the state of one of the previous set of particles with probability according to that particle's weight.
- This is best achieved by generating the *cumulative probability distribution* of the particles, generating a random number between 0 and 1 and then picking the particle whose cumulative probability this intersects.
- Note that for efficiency it is possible to skip the normalisation step and resample directly from an unnormalised distribution.

Another Measurement Possibility: Compass Sensor

A digital compass gives a robot the ability to estimate its rotation *without drift*.

- Having a digital compass and a single sonar sensor puts us in a position close to that of having a full sonar ring.
- In principle, the robot could stop and rotate on the spot every so often to simulate a sonar ring.
- We could simply monitor the compass as the robot continuously moves and feed this into our filter.
- But we won't try it today, because the NXT compasses don't work very well in the lab.

Measurement Update: Compass Sensor

- Compass measures bearing β relative to north. What is $P(\beta|\mathbf{x}_i)$?
- Clearly the likelihood only depends on the θ part of $\mathbf{x}_i$.
- If the compass is working perfectly, then:

$$\beta = \gamma - \theta \ ,$$

  where γ is the magnetic bearing of the x coordiante axis of frame W.
- So we should assess the uncertainty in the compass, and set a likelihood which depends on the difference between β and $\gamma - \theta$; e.g. Gaussian $e^{\frac{-(\beta-(\gamma-\theta))^2}{2\sigma_c^2}}$
- Particles far from the right orientation will get low weights.

Continuous vs. Global Localisation

- Continuous localisation is a tracking problem: given a good estimate of where the robot was at the last time-step and some new measurements, estimate its new position.
- Global localisation is often called the 'kidnapped robot problem': the environment is known, but the robot's position within it is completely uncertain.

Using MCL, we tackle both of these problems in the same way but initialise the particle set in different ways.

Continuous vs. Global Localisation

- Initialising continuous localisation from a known robot position: set the state of all particles to the same value. Set all weights to be equal. The result is a 'spike', completely certain point estimate of the robot's location.

$$\mathbf{x}_1 = \mathbf{x}_2 = \ldots = \mathbf{x}_N = \mathbf{x}_{init} \quad ; \quad w_1 = w_2 = \ldots w_N = 1/N$$

- Initialising global localisation: the state of each particle should be sampled randomly from all of the possible positions within the environment. Set all weights to be equal.

$$\mathbf{x}_i = Random \quad ; \quad w_1 = w_2 = \ldots w_N = 1/N$$

- With only one sonar sensor global localisation will be very difficult.

Inferring an Estimate and Position-Based Navigation

- We can make a point estimate of the current position and orientation of the robot by taking the mean of all of the particles:

$$\bar{\mathbf{x}} = \sum_{i=1}^{N} w_i \mathbf{x}_i \ .$$

- i.e. the means of the x, y and θ components are all calculated individually and stored in $\bar{\mathbf{x}}$.
- The robot can now plan A $\rightarrow$ B type positional navigation.

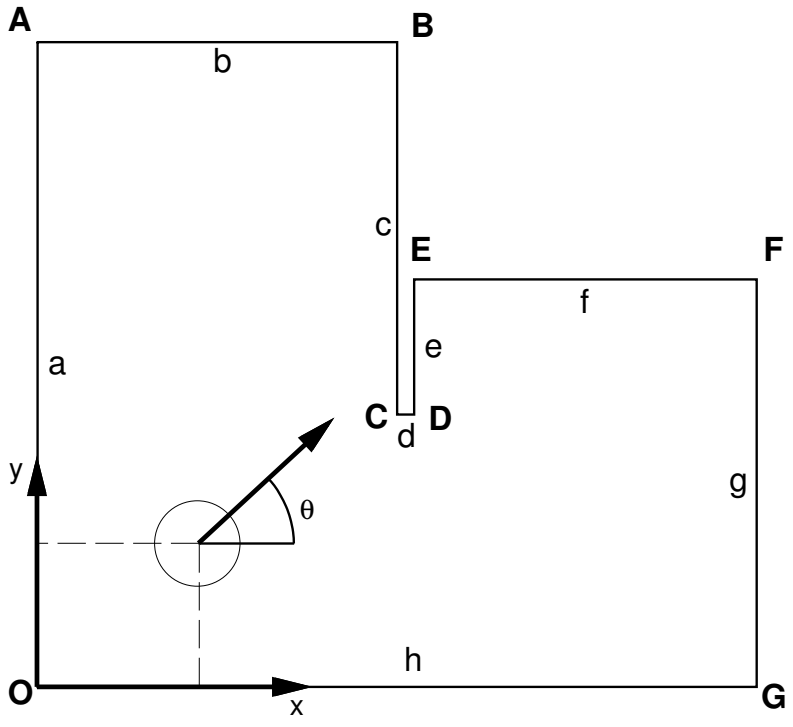
Robotics

Lecture 7: Advanced Sensing: Place Recognition and Occupancy Mapping

See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

Andrew Davison
Department of Computing
Imperial College London

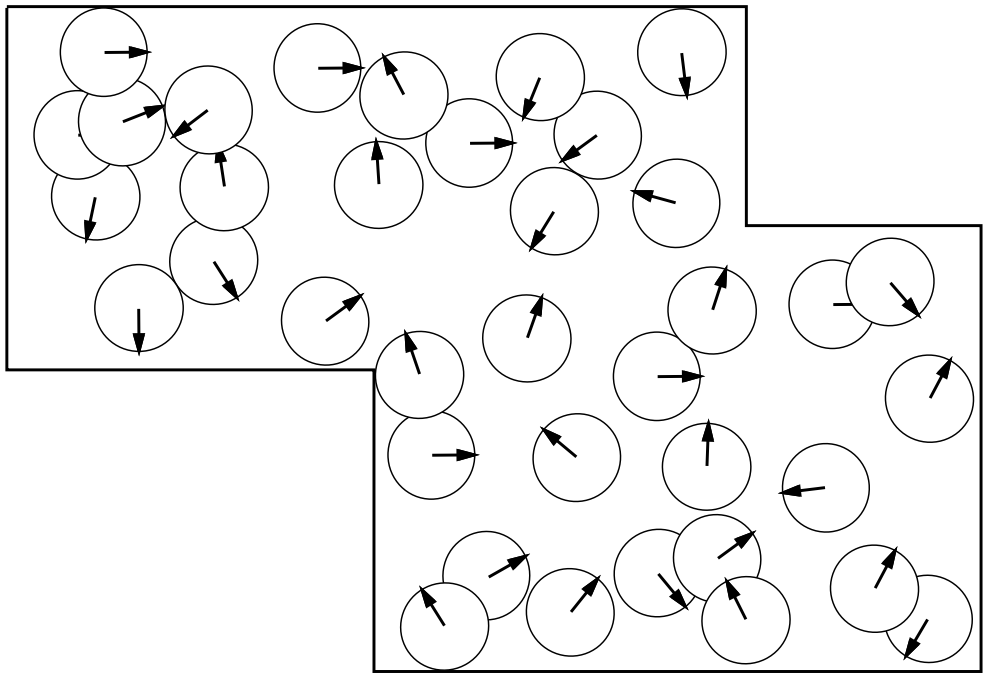
Review: Monte Carlo Localisation Practical 6



- Most challenging part is getting the likelihood function right to correctly reweight particles after measurements.

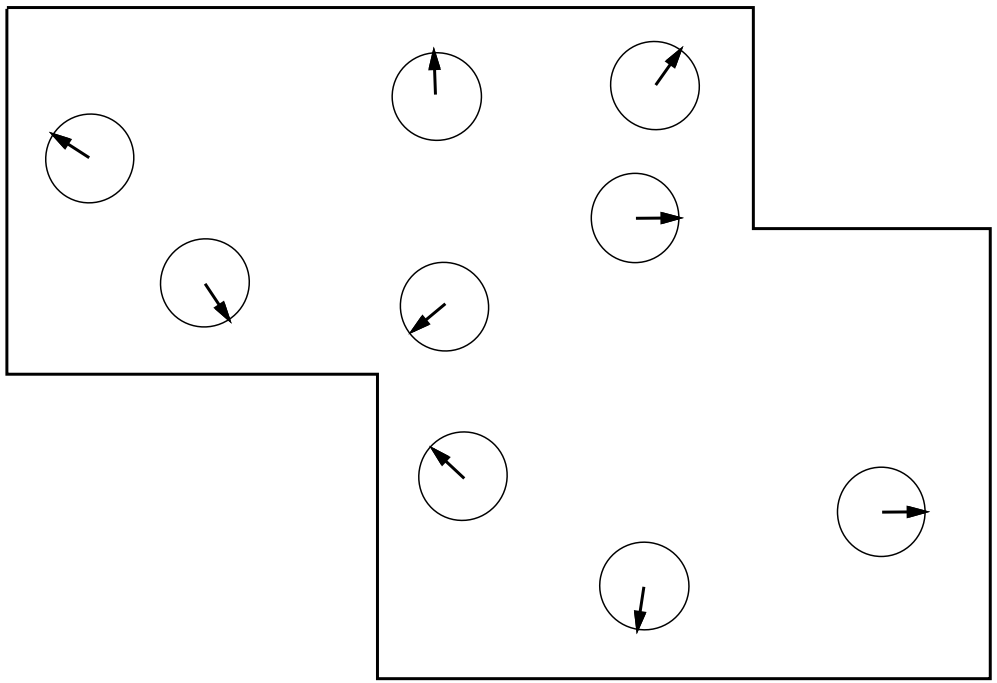
Global Localisation ('Kidnapped Robot') with Sonar

- In MCL, global localisation can be attempted by initialising a large number of particles randomly spread through the environment, and then running the filter normally. However, this requires many particles (computational expensive) and it may take many movements and measurements to find the right location.
- We would expect better performance with more powerful sensing.



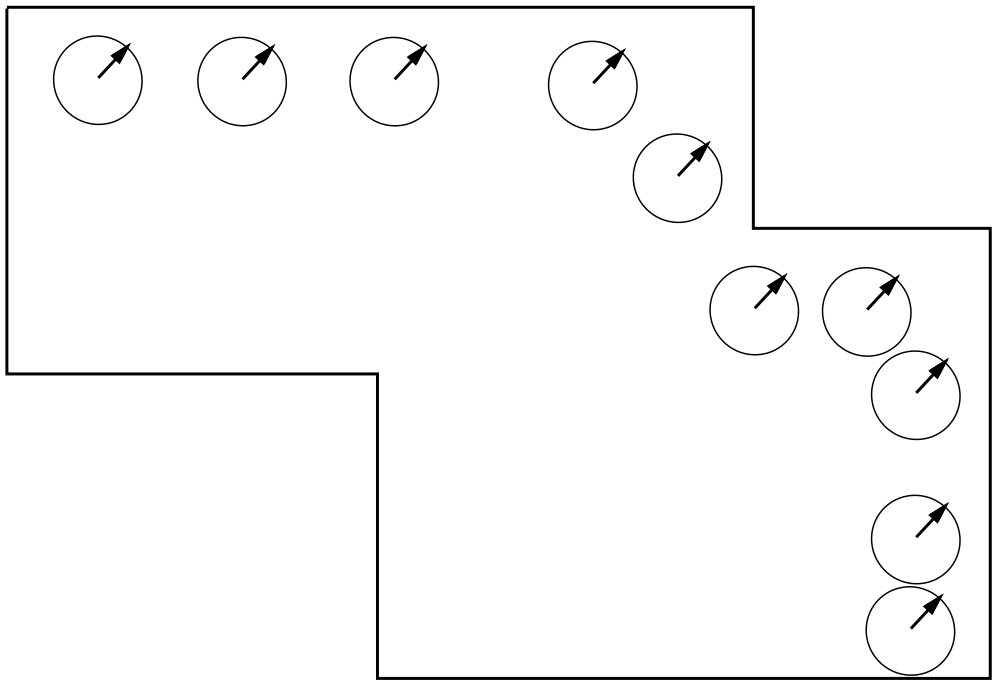
One Depth Measurement and Resampling

- After a measurement (e.g. sonar depth = 20cm), the weights of particles consistent with it will increase.
- Movement and further measurements are needed to lock down position, and ambiguities may still arise.



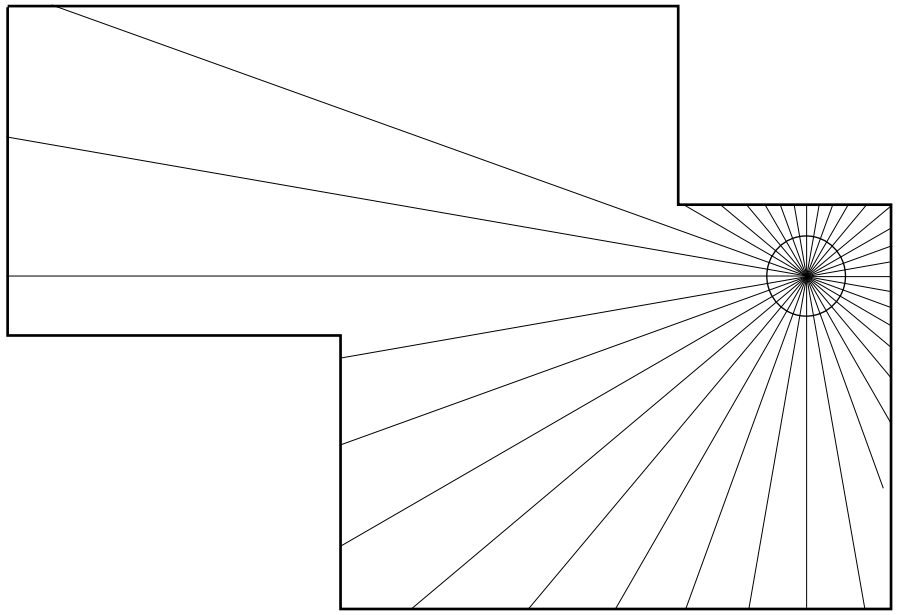
Using a Compass and Sonar Together

- Ambiguity is much reduced with extra measurements, such as from a compass.
- e.g. sonar depth = 20cm, compass bearing = 45°.



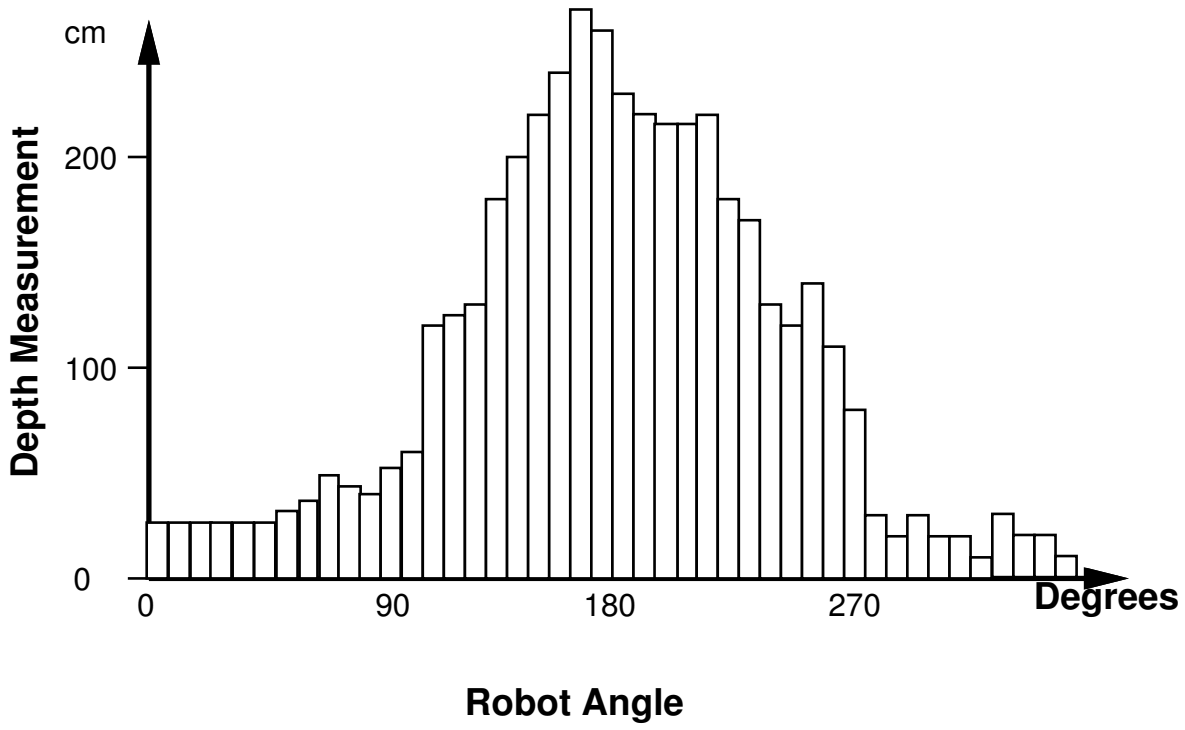
Global Localisation Via Recognition

- An alternative relocalisation technique involves making a lot of measurements at a number of chosen locations and *learning* their characteristics.
- This can be done without a prior map but needs *training*.
- The robot can only recognise the locations it has learned.
- For instance: at each location, spin the robot and take a regularly spaced set of sonar measurements (e.g. one per degree).



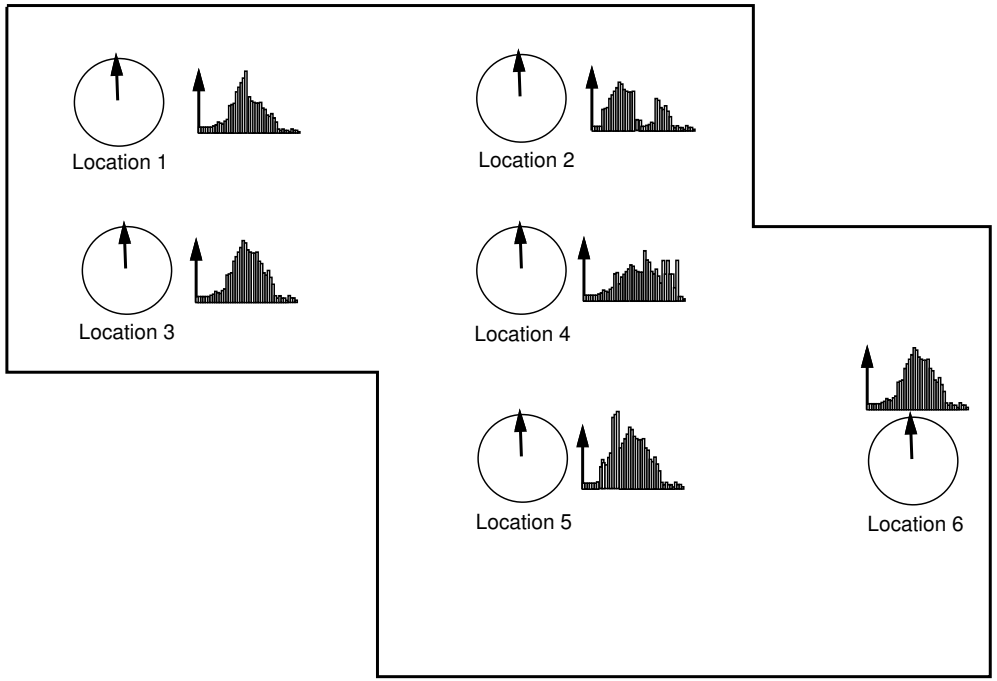
Measuring to Learn a Location

- First the robot must be placed in each target location to learn its appearance.
- The raw measurements are stored to describe the location: a place descriptor, or *signature*.



Place Recognition

- Afterwards, the robot is placed in one of the locations and it must take a set of measurements then decide which it is in.
- It must see which saved signature matches best to the new measurements by checking each in turn.



Place Recognition

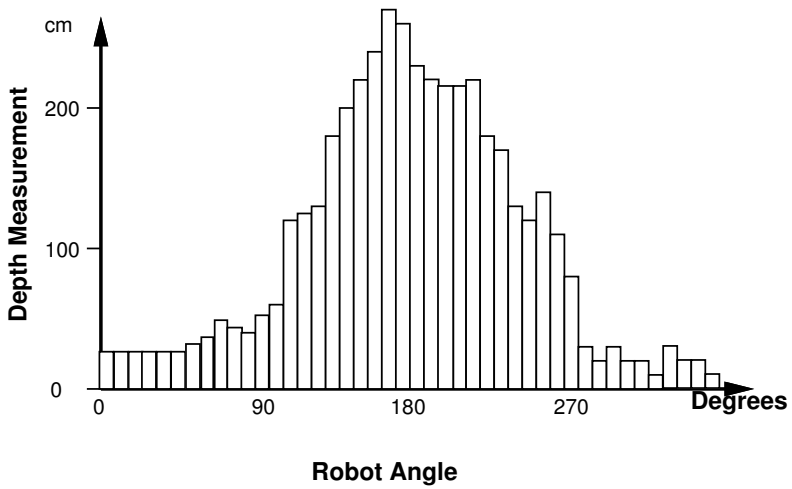
- Two histograms can be compared with a correlation test, measuring the sum of squared differences. Difference D_k between new measurement histogram $H_m(i)$ and saved signature histogram $H_k(i)$ is:

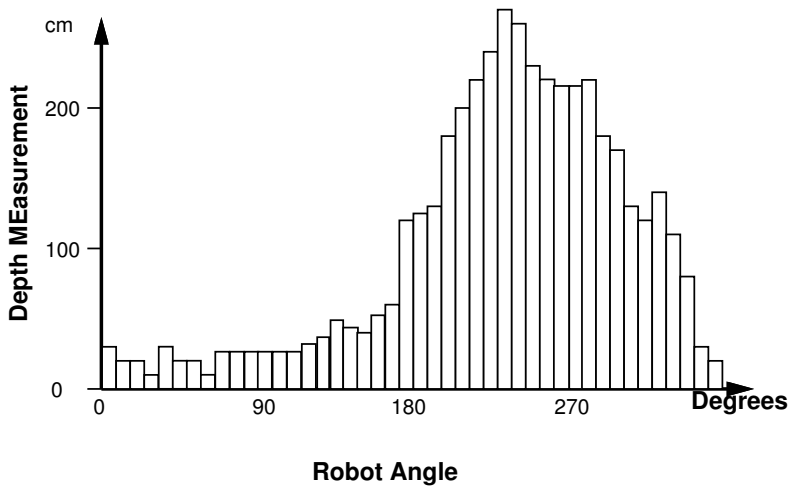
$$D_k = \sum_i \left(H_m(i) - H_k(i)\right)^2 \ .$$

- The saved location with lowest D_k is the most likely candidate, but perhaps we also check that D_k is below a threshold in case the robot is in none of the known locations.

Estimating Orientation

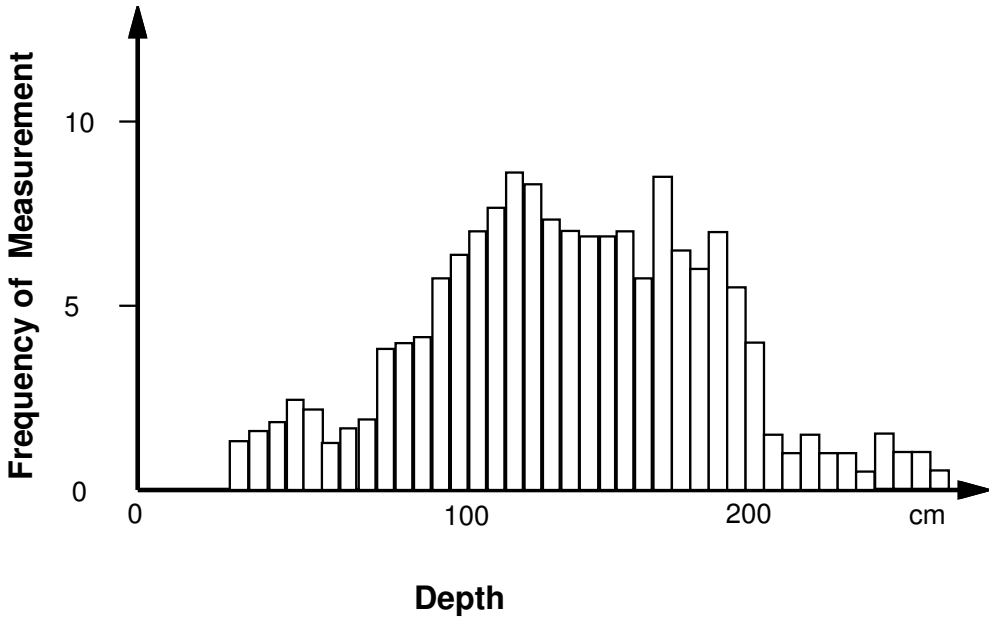
- If the test histogram and that from one of the saved locations can be brought into close agreement by only a shift, the robot is in the same place but rotated.
- The amount of shift to get the best agreement is a measurement of the rotation.



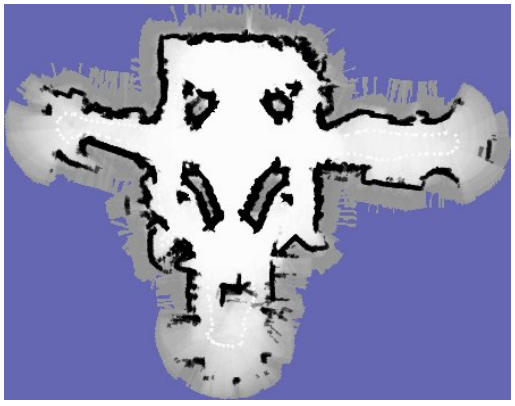


Depth Measurement Histogram

- Optionally (to save the computational cost of always trying every shift), we can build a signature which is *invariant* to robot rotation, such as a histogram of occurences of certain depth measurements.
- Matching tests can then be carried out on this directly.
- Once the correct location has been found, the shifting procedure to find the robot's orientation need only be carried out for that one location.

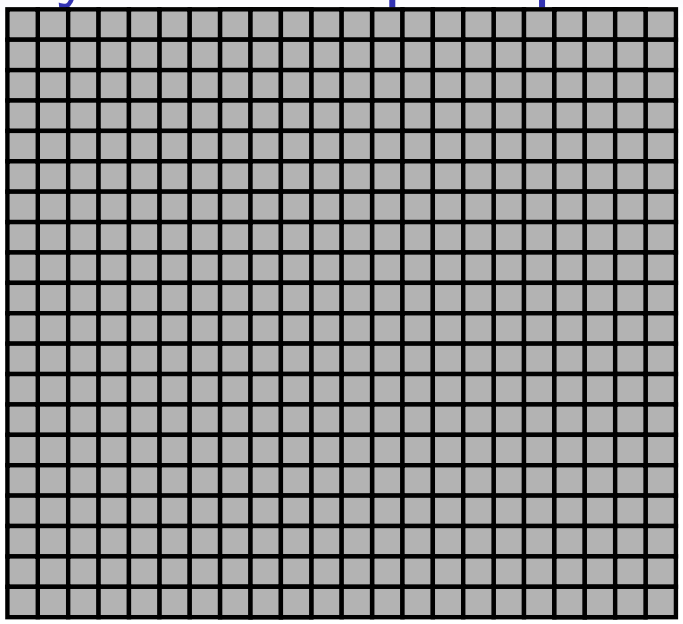


Probabilistic Occupancy Grid Mapping



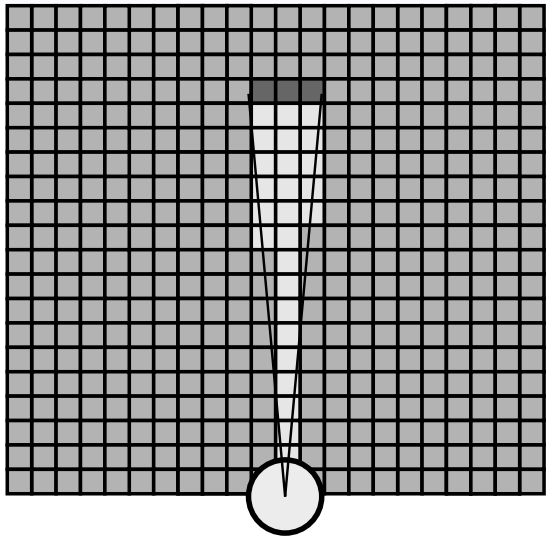
- The second method we will look at today is a probabilistic algorithm for *mapping* in the case that a robot's localisation is known. The goal is to infer which parts of the environment around a robot are navigable free-space, and which parts contain obstacles.
- In Occupancy Mapping, rather than building a parametric map of the positions of walls we use a regular grid representation.
- Occupancy grids accumulate the uncertain information from sensors like sonar to solidify towards precise maps.

Occupancy Grid Map Representation



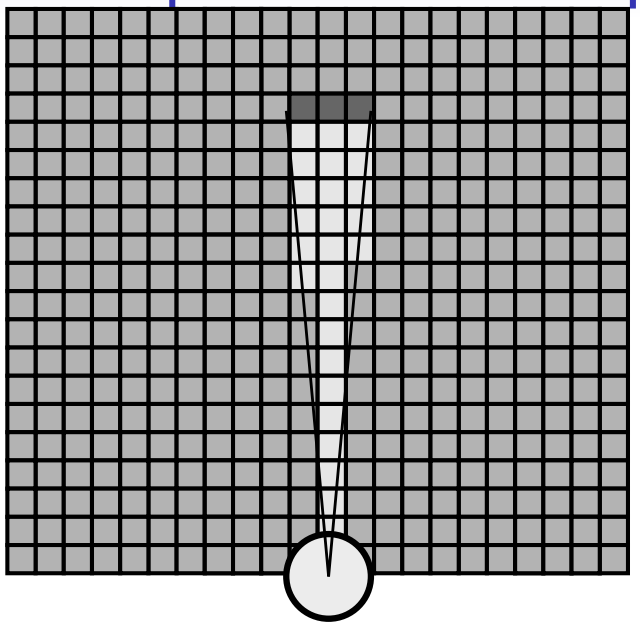
- We define an area on the ground we would like to map, and choose a square grid cell size.
- For each cell i, we store and update a *probability of occupancy* $P(O_i)$ that it is occupied by an obstacle.
- $P(E_i)$ is the corresponding probability that the cell is empty, where $P(O_i) + P(E_i) = 1$.
- We initialise the occupancy probabilities for unexplored space to a constant prior value; for instance 0.5
- Occupancy maps are often visualised with a greyscale value for each cell: from black for $P(O_i) = 1$ to white for $P(O_i) = 0$; intermediate

Update after Sonar Measurement



- For each cell we want to update the probability of occupancy to take account of a new sonar measurement Z.
- Suppose that the sonar reports a depth $Z = d$. This provides evidence that cells around distance d in front of the robot are more likely to be occupied. But also, that cells in front of the robot *at depths less than d* are more likely to be empty.
- A sonar beam is not a perfect ray but has a width (e.g. 10–15° as is spreads out and we can take account of this as shown.
- For each cell, we must test if it lies within the beam given the robot's position. We do not learn anything about cells beyond the beam width or beyond the measured depth.

Bayesian Update of Occupancy



Technically, for each cell, we apply Bayes rule to get a posterior probability for each cell.

$$P(O_i|Z) = \frac{P(Z|O_i)P(O_i)}{P(Z)}$$

As in MCL, we could avoid calculating $P(Z)$ by also calculating $P(E_i|Z)$ and normalising since we know $P(O_i|Z) + P(E_i|Z) = 1$.

$$P(E_i|Z) = \frac{P(Z|E_i)P(E_i)}{P(Z)}$$

Log Odds Representation

If we take the ratio of the two Bayes rule expressions on the previous slide:

$$\left(\frac{P(O_i|Z)}{P(E_i|Z)}\right) = \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) * \left(\frac{P(O_i)}{P(E_i)}\right)$$

We can use the odds notation: $o(A) = \frac{P(A)}{P(\bar{A})}$.

$$o(O_i|Z) = \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) * o(O_i)$$

Taking logs:

$$\ln o(O_i|Z) = \ln \left(\frac{P(Z|O_i)}{P(Z|E_i)}\right) + \ln o(O_i)$$

So in this form, for each cell we store $\ln o(O_i)$ and update it additively. Cells with probability 0.5 of occupancy will have log odds 0; positive log odds means probability > 0.5; negative log odds means probability < 0.5. Also we normally cap log odds at certain positive and negative limits.

Likelihood Model of Sensor

- We need models for the likelihood function which models sensor performance. We can consider directly the ratio of likelihoods we need to update log odds.
- Log odds update $U = \ln \frac{P(Z|O_i)}{P(Z|E_i)}$: for each cell $P(Z|O_i)$ is the probability of obtaining the sensor value we did given that the cell is occupied; $P(Z|E_i)$ is the probability of obtaining that value given that the cell is empty.
- For cells within the sonar beam but closer than the measured depth $Z = d$: $\frac{P(Z|O_i)}{P(Z|E_i)}$ is less than 1; we can choose a constant negative value for U.
- For cells within the sonar at around the measured depth $Z = d$: $\frac{P(Z|O_i)}{P(Z|E_i)}$ is greater than 1; we can choose a constant positive value for U.
- Note that we are somewhat oversimplifying in occupancy grids in assuming the the probabilities of occupancy for each cell are independent.

Occupancy Grid Mapping

- Over time and robot motion, measurements accumulate and the occupancy map converges towards white and black indicating definite knowledge.
- See an example at `http://www.youtube.com/watch?v=aEeS8hDnnYg`
- An occupancy map's probability values must be thresholded if a decision is to be made about where the robot can actually drive.
- Large scale occupancy maps are very memory intensive; and subject to drift due to localisation uncertainty. We will look at methods to resolve this next week.

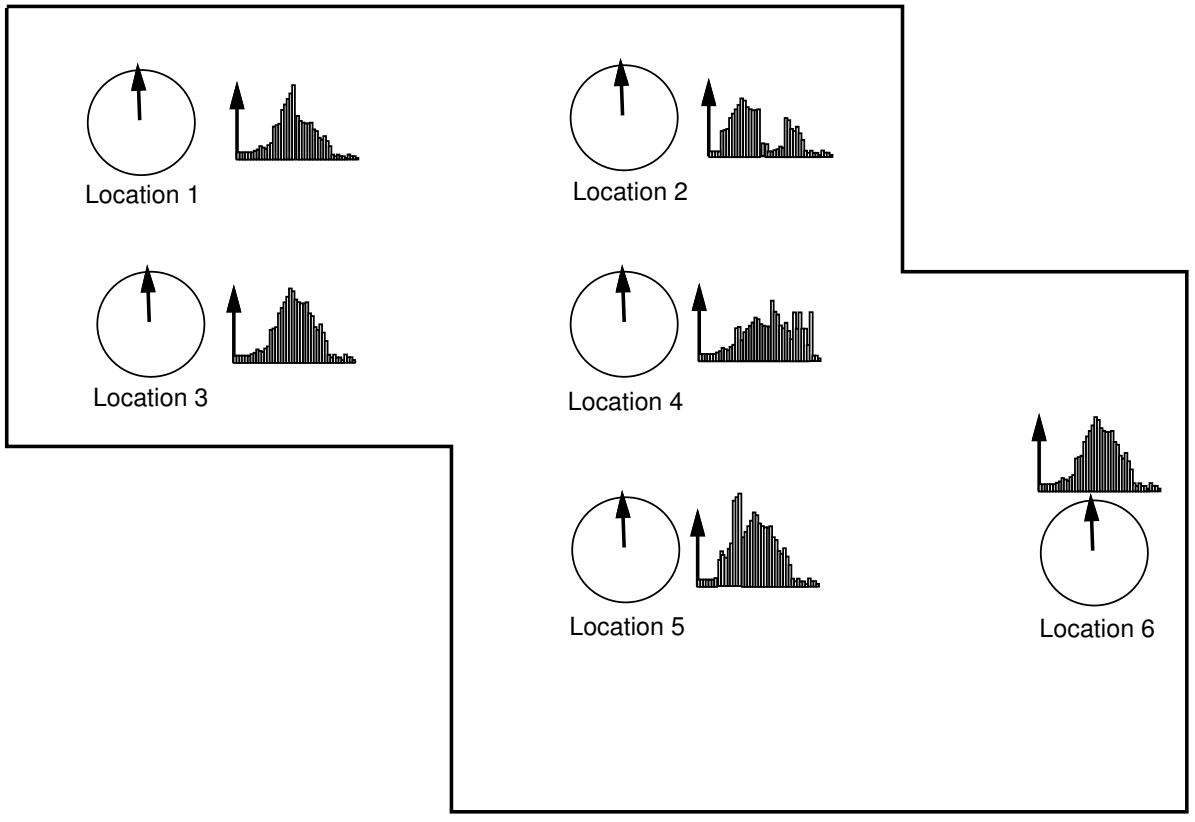
Robotics

Lecture 8: Simultaneous Localisation and Mapping (SLAM)

See course website
`http://www.doc.ic.ac.uk/~ajd/Robotics/` for up to date information.

Andrew Davison
Department of Computing
Imperial College London

Review: Practical 7



- Need repeatable spin and measurement.
- Recognising orientation too will be computationally costly without invariant descriptors.

Simultaneous Localisation and Mapping

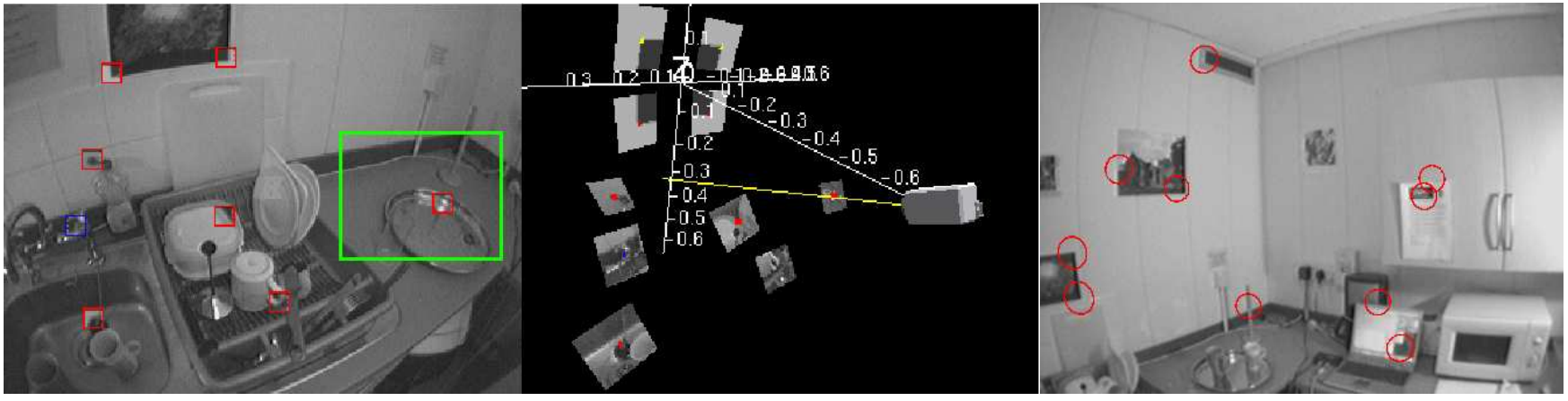
- One of the big successes of probabilistic robotics.

  A body with quantitative sensors
  moves through a previously unknown,
  static environment, mapping it
  and calculating its egomotion.

- When do we need SLAM?
  - When a robot must be truly autonomous (no human input).
  - When nothing is known in advance about the environment.
  - When we can't place beacons (or even use GPS like indoors or underwater).
  - And when the robot actually needs to know where it is.

Features for SLAM

- Most SLAM algorithms make maps of natural scene *features*.
- Laser/sonar: line segments, 3D planes, corners, etc.
- Vision: salient point features, lines, textured surfaces.

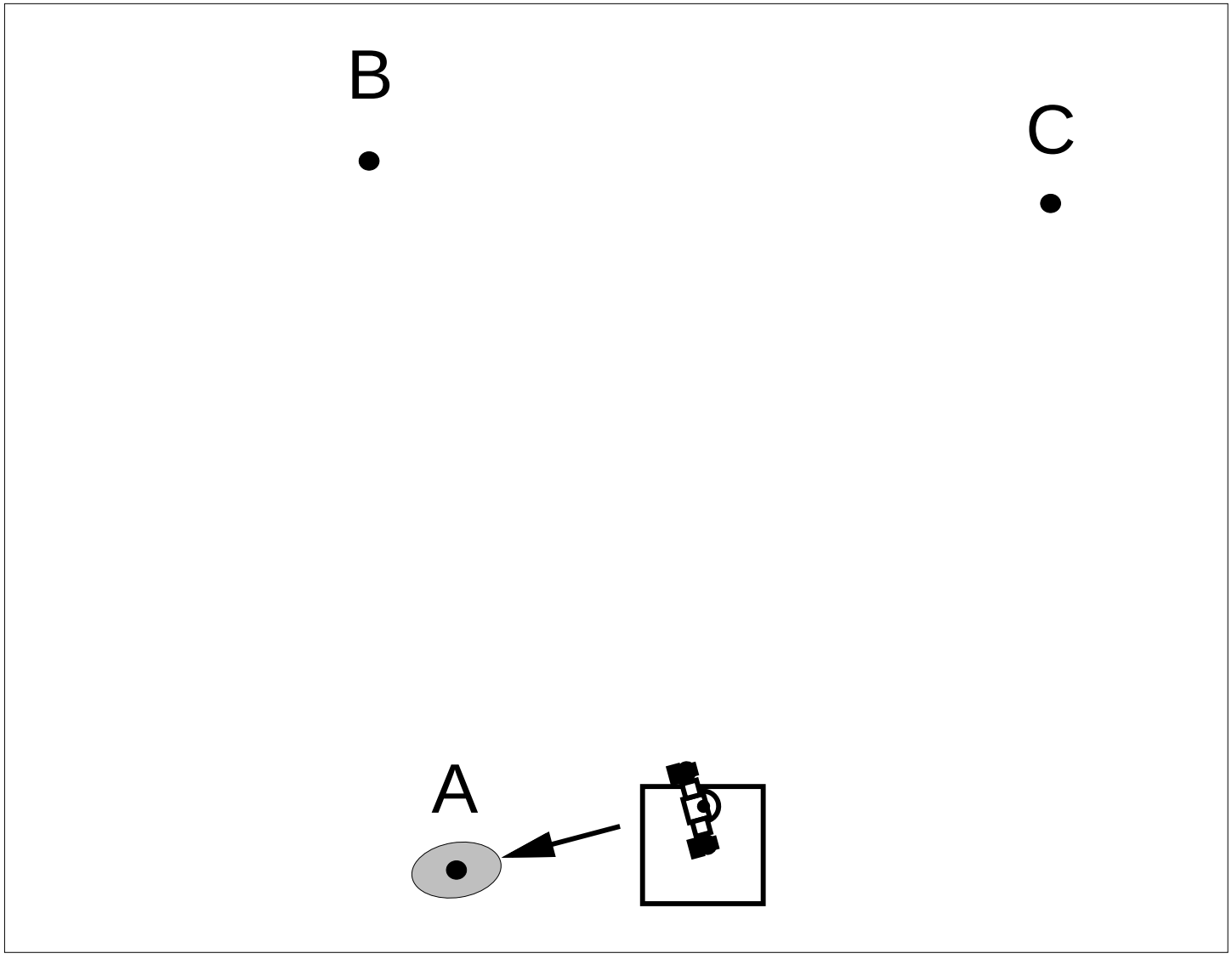


- Features should be distinctive; recognisable from different viewpoints (data association).

Propagating Uncertainty

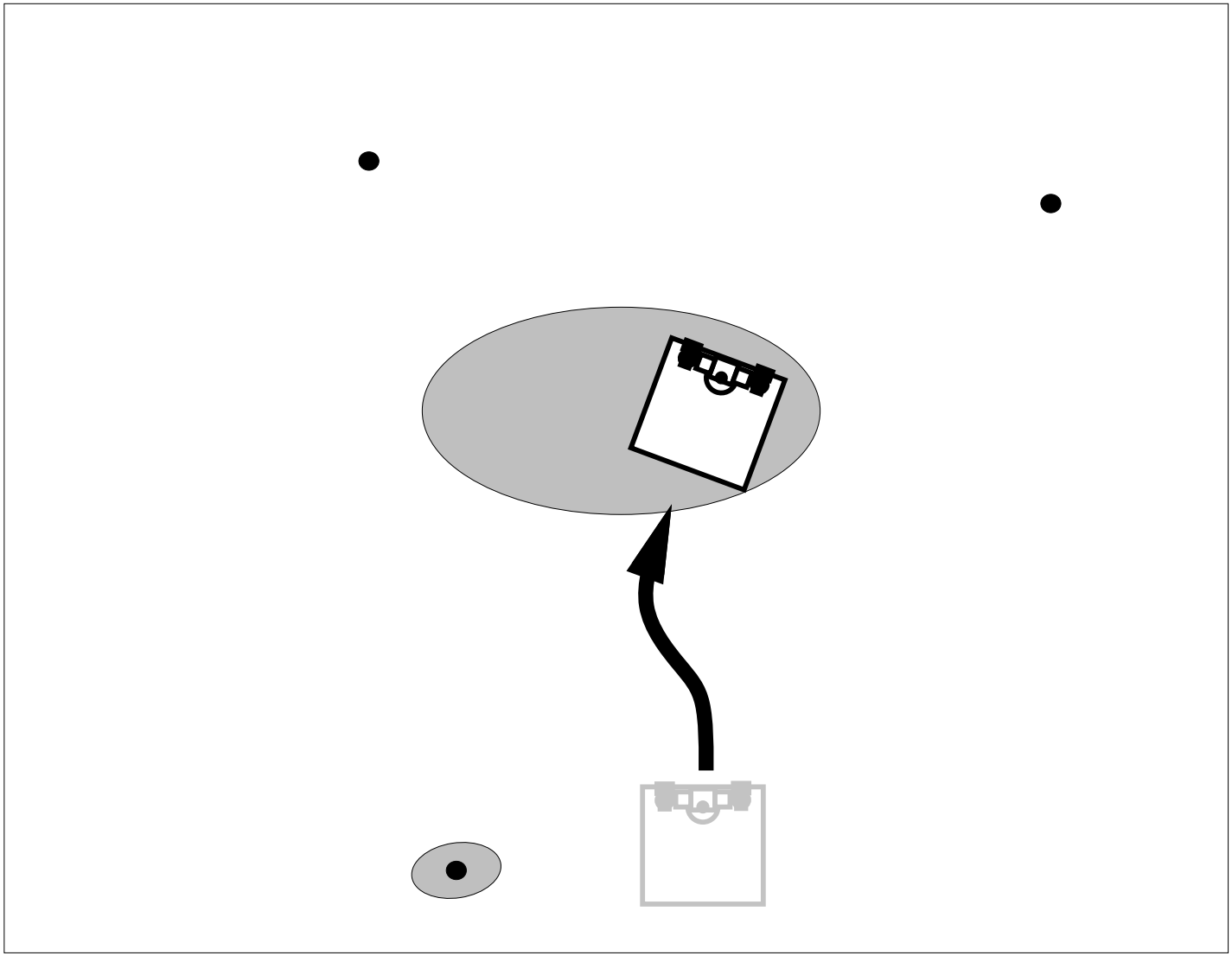
- SLAM seems like a chicken and egg problem — but we can make progress if we assume the robot is the only thing that moves.
- Main assumption: the world is static.

Simultaneous Localisation and Mapping



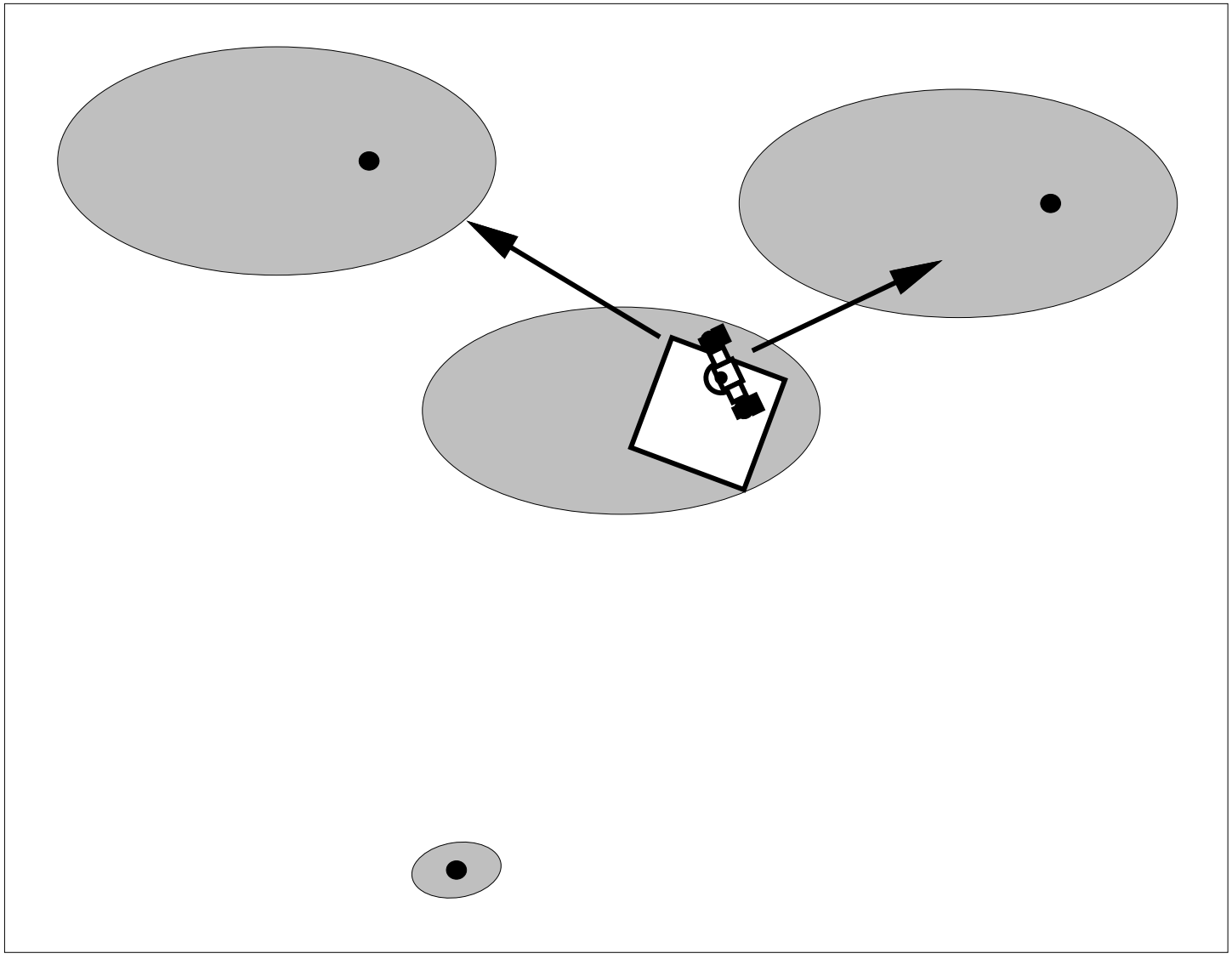
(a) Robot start (zero uncertainty); first measurement of feature A.

Simultaneous Localisation and Mapping



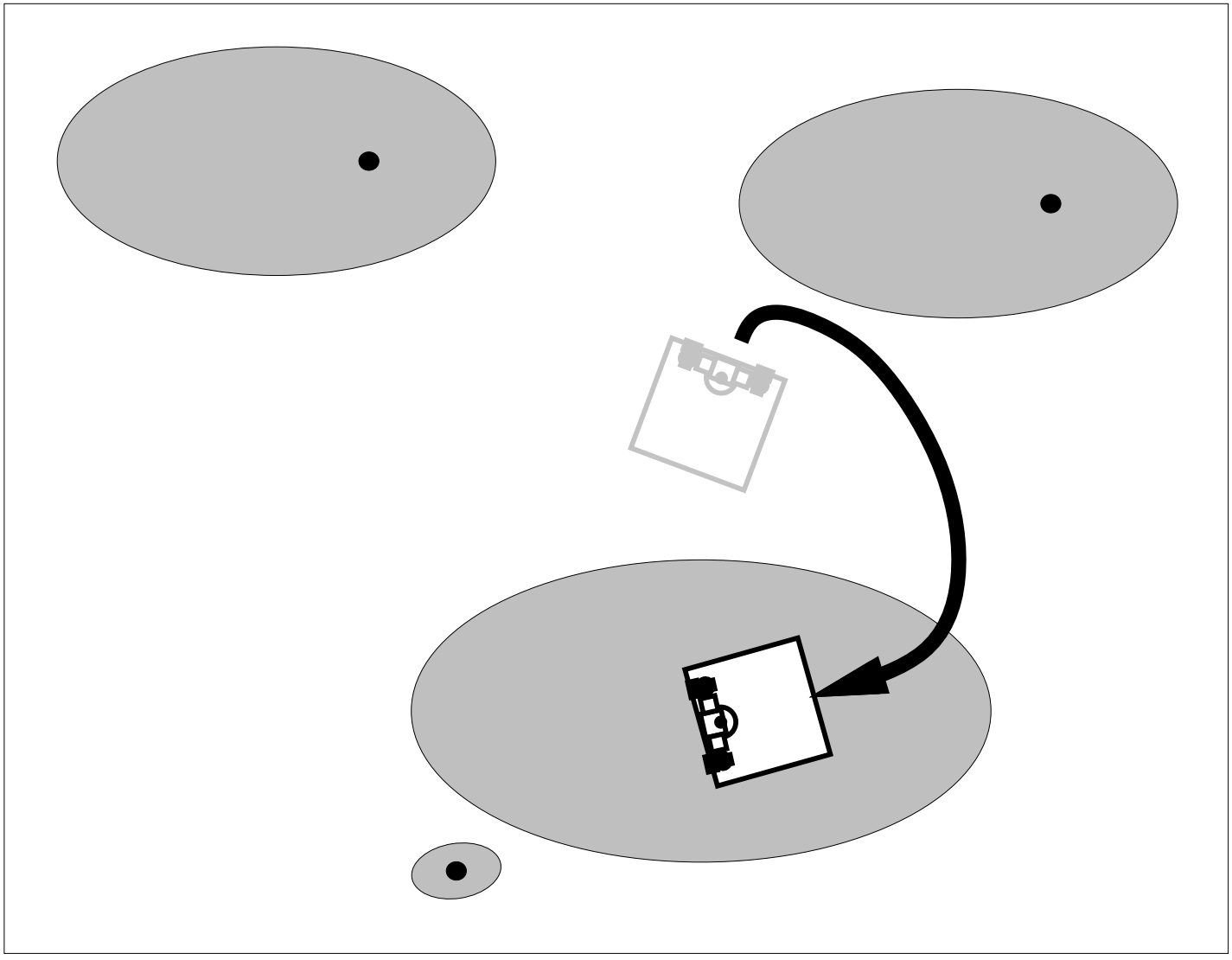
(b) Robot drives forwards (uncertainty grows).

Simultaneous Localisation and Mapping



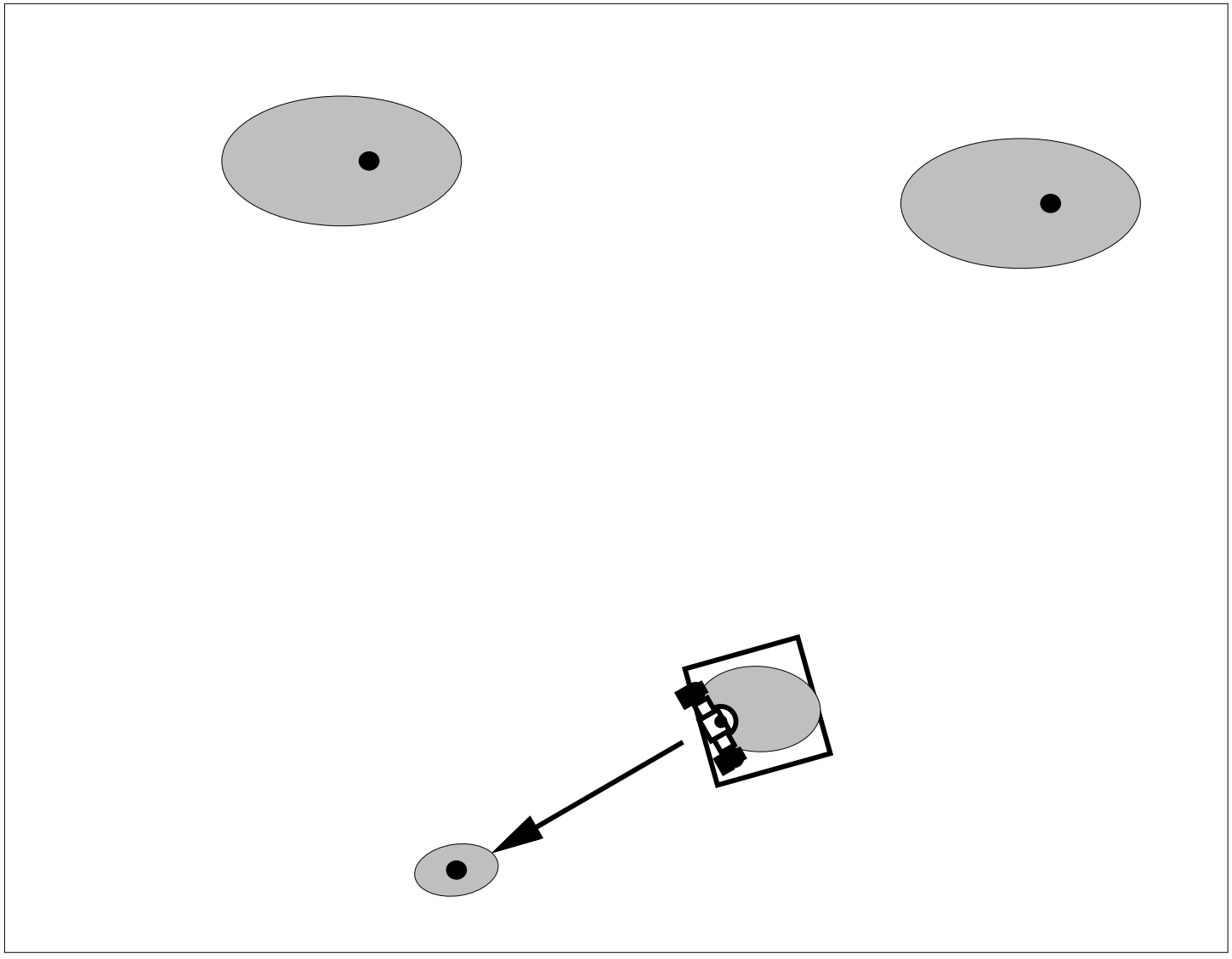
(c) Robot makes first measurements of B and C.

Simultaneous Localisation and Mapping



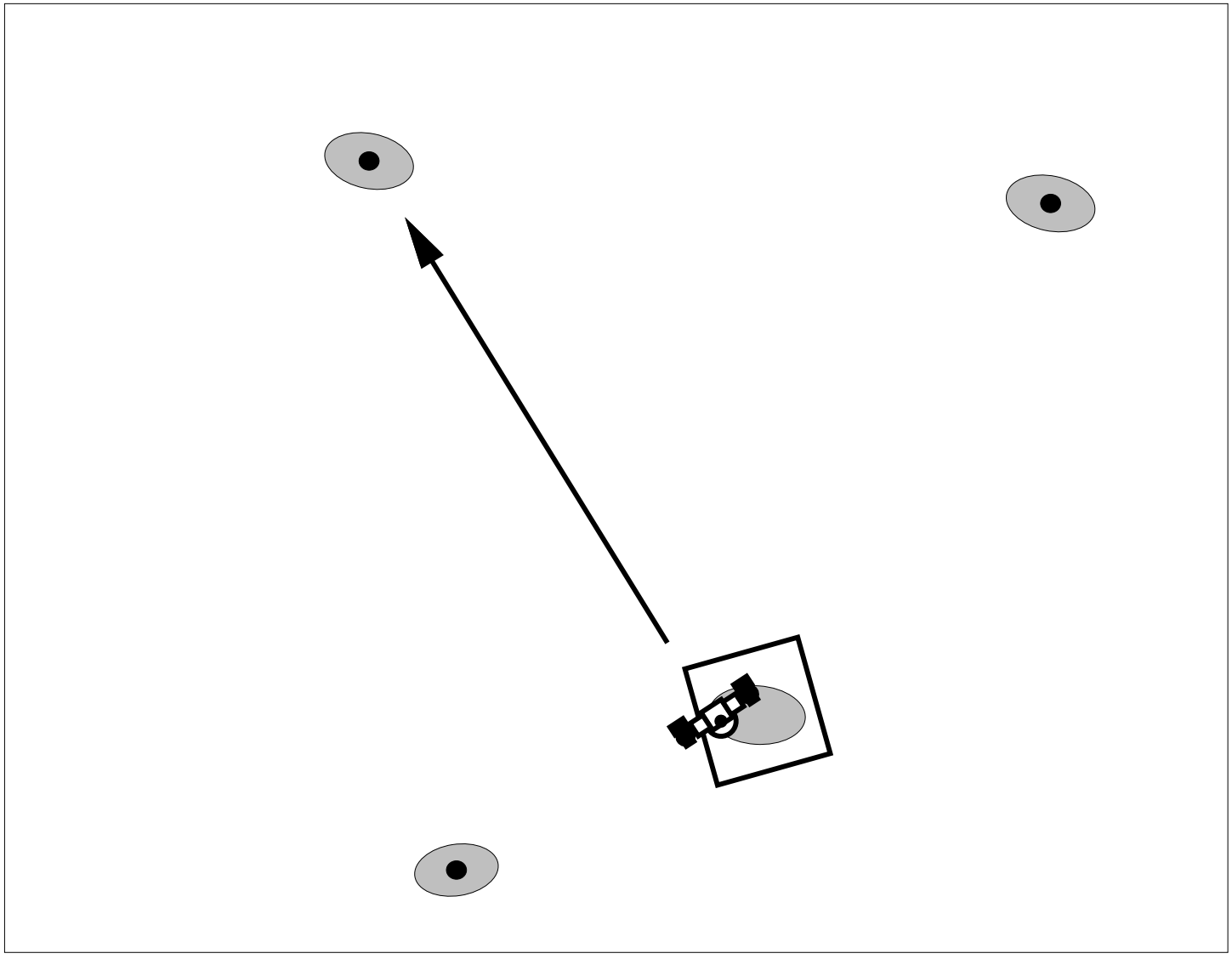
(d) Robot drives back towards start (uncertainty grows more)

Simultaneous Localisation and Mapping



(e) Robot re-measures A; *loop closure*! Uncertainty shrinks.

Simultaneous Localisation and Mapping



(f) Robot re-measures B; note that uncertainty of C also shrinks.

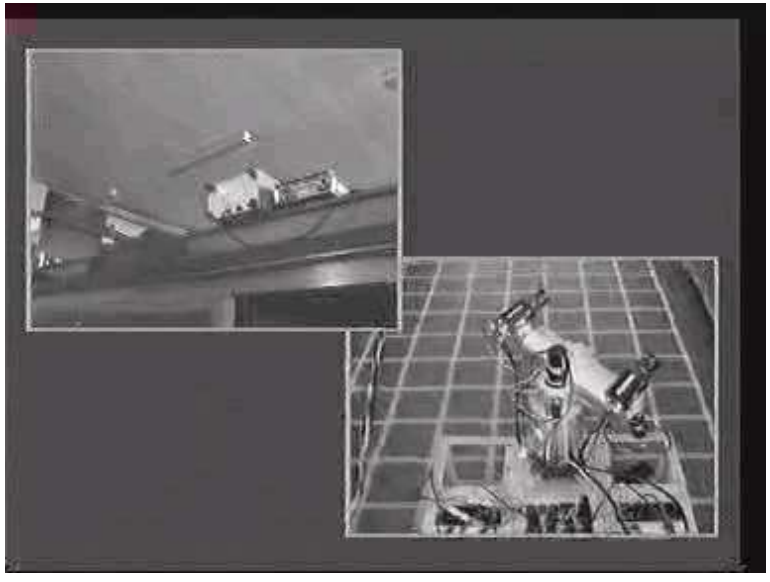
Simultaneous Localisation and Mapping

- First Order Uncertainty Propagation

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{\mathbf{x}}_v \\ \hat{\mathbf{y}}_1 \\ \hat{\mathbf{y}}_2 \\ \vdots \end{pmatrix} \quad , \quad \mathrm{P} = \begin{bmatrix} \mathrm{P}_{xx} & \mathrm{P}_{xy_1} & \mathrm{P}_{xy_2} & \cdots \\ \mathrm{P}_{y_1x} & \mathrm{P}_{y_1y_1} & \mathrm{P}_{y_1y_2} & \cdots \\ \mathrm{P}_{y_2x} & \mathrm{P}_{y_2y_1} & \mathrm{P}_{y_2y_2} & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

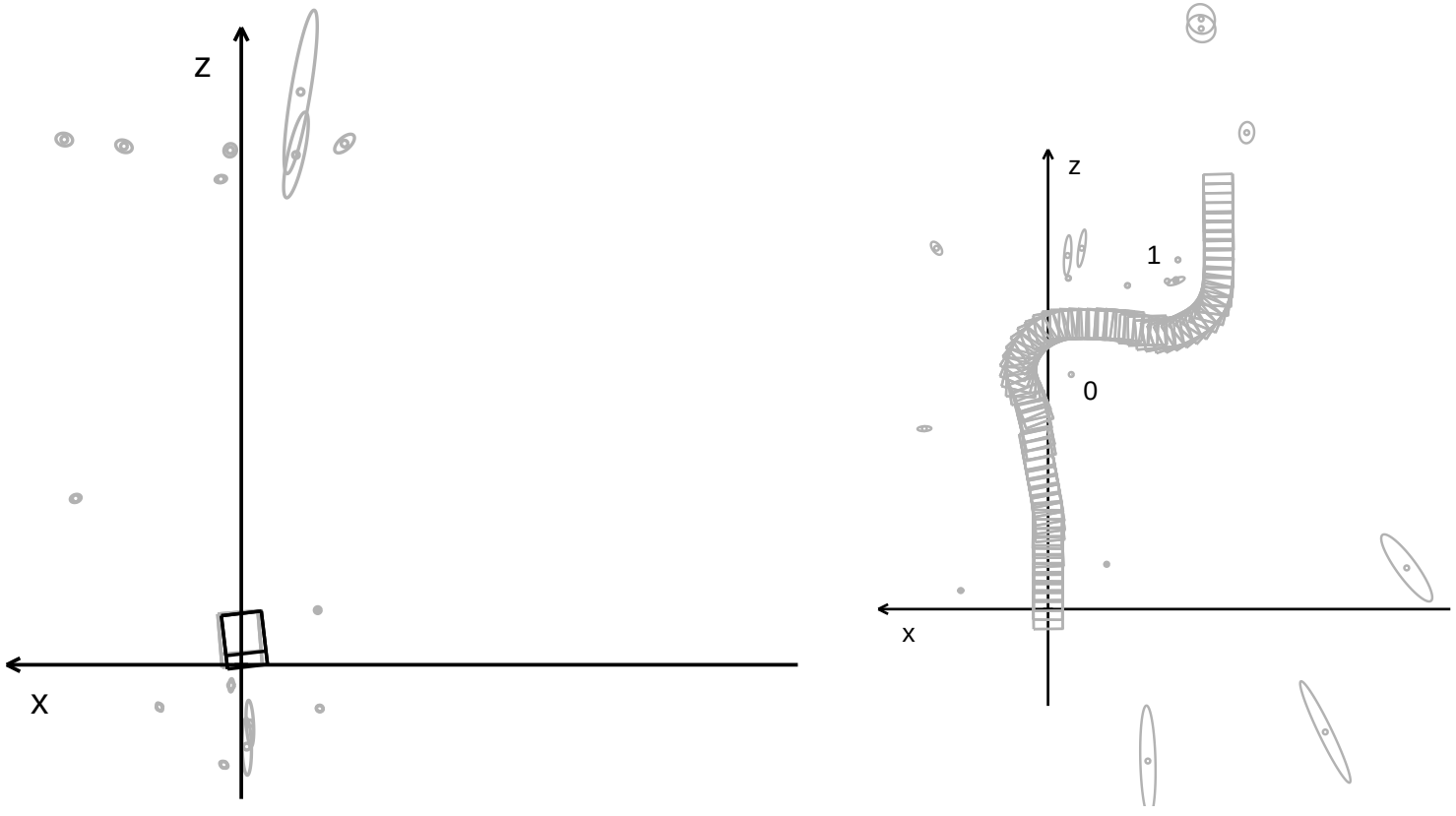
- $\mathbf{x}_v$ is robot state, e.g. (x, y, θ) in 2D; y_i is feature state, e.g. (X, Y) in 2D.
- PDF over robot and map parameters is modelled as a single multi-variate Gaussian and we can use the Extended Kalman Filter.
- PDF represented with state vector and covariance matrix.

SLAM Using Active Vision



- Stereo active vision; 3-wheel robot base.
- Automatic fixated active mapping and measurement of arbitrary scene features.
- Sparse mapping.

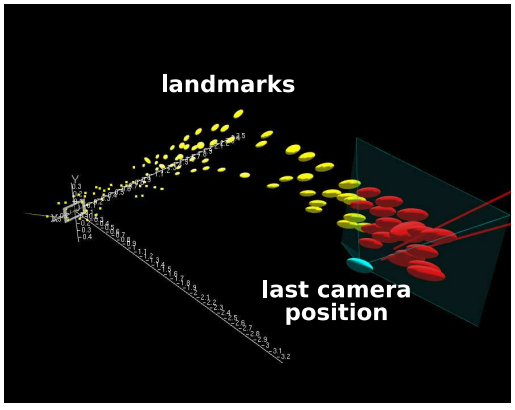
Limits of Metric SLAM

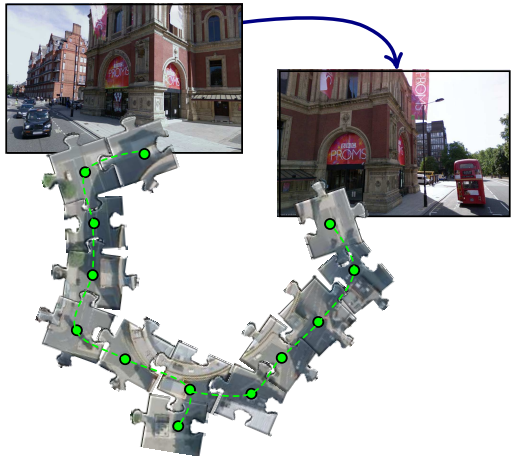


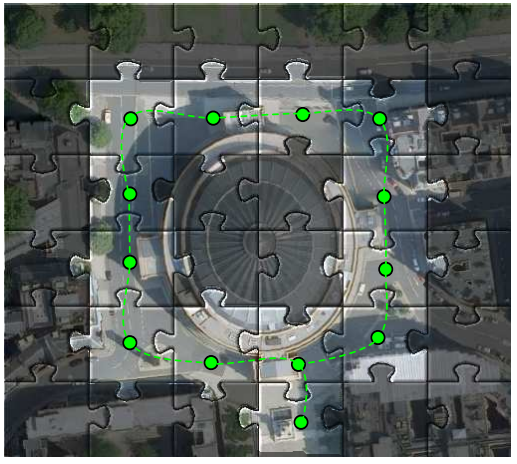
Purely metric probabilistic SLAM is limited to small domains due to:

- Poor computational scaling of probabilistic filters.
- Growth in uncertainty at large distances from map origin makes representation of uncertainty inaccurate.
- *Data Association* (matching features) gets hard at high uncertainty.

Large Scale Localisation and Mapping







Local Metric Place Recognition Global Optimisation

Practical modern solutions to large scale mapping follow a *metric/topological* approach. They need the following elements:

- Local metric mapping to estimate trajectory and possibly make local maps.
- Place recognition, to perform 'loop closure' or relocalise the robot when lost.
- Map optimisation/relaxation to optimise a map when loops are closed.

Global Topological: 'Loop Closure Detection'

- Angeli *et al.*, IEEE Transactions on Robotics 2008.

Pure Topological SLAM

- Graph-based representation.
- Segmentation of the environment into linked distinct places.
- Adapted to symbolic planning and navigation.

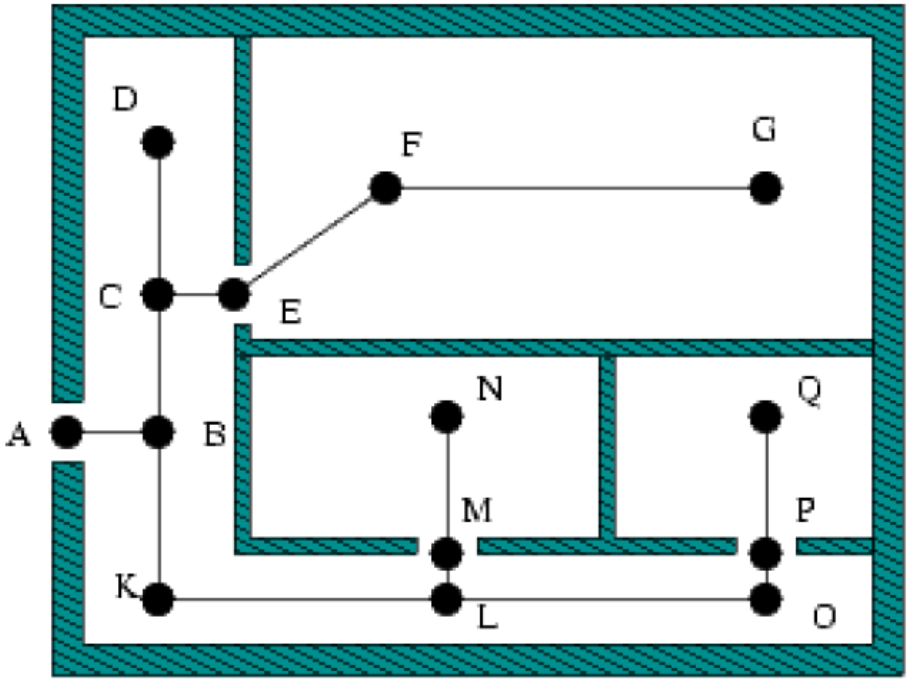
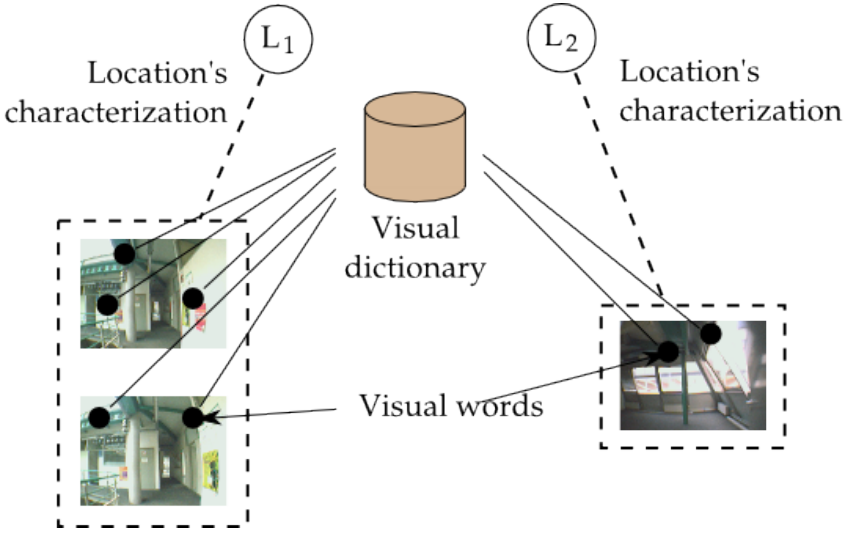
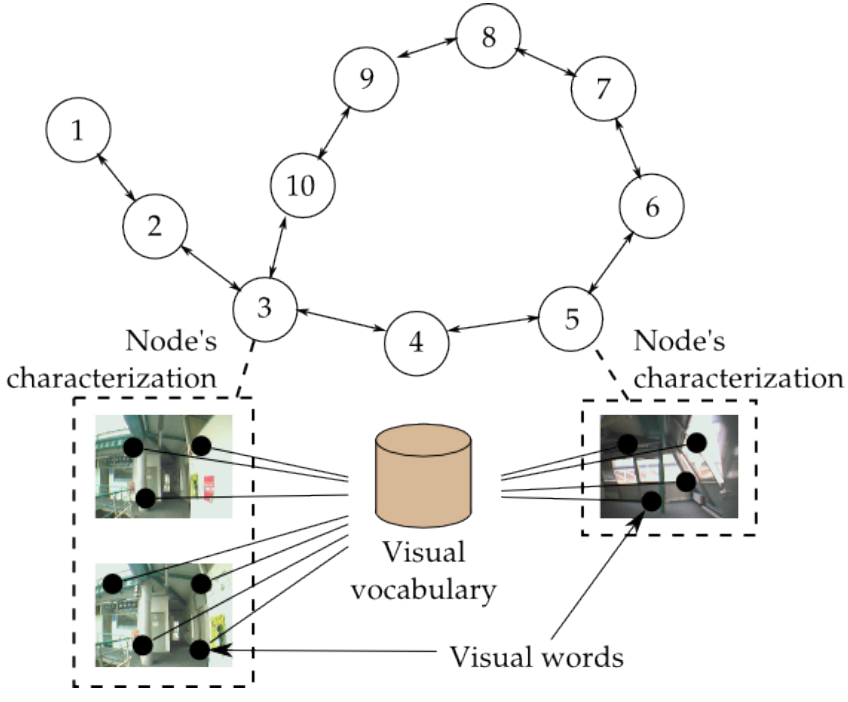


Figure: Topological representation

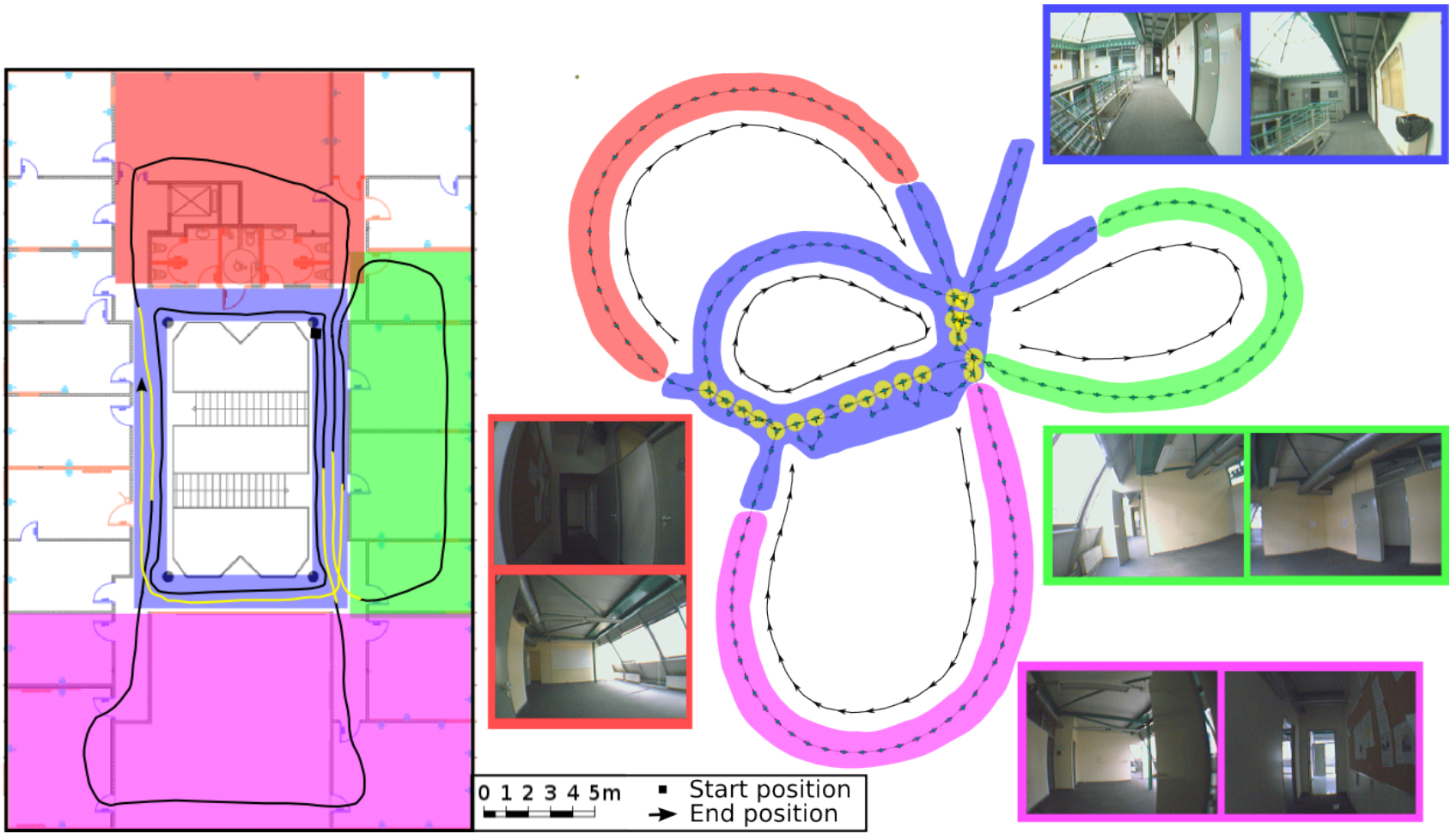
Environment Model

- Map defined as a graph of connected locations.
- Edges model relationships between locations (e.g. traversability, similarity).

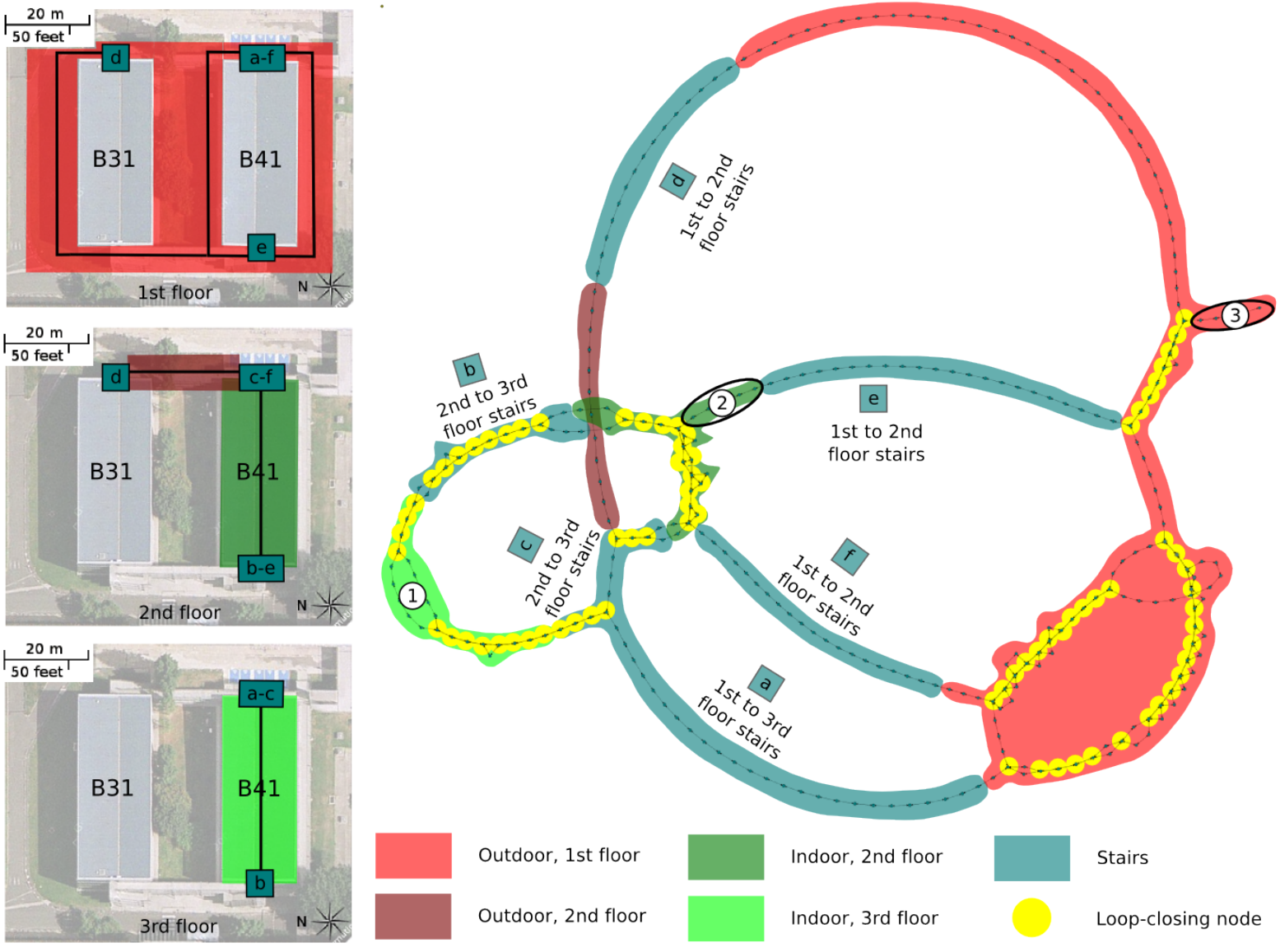




Indoor Topological Map

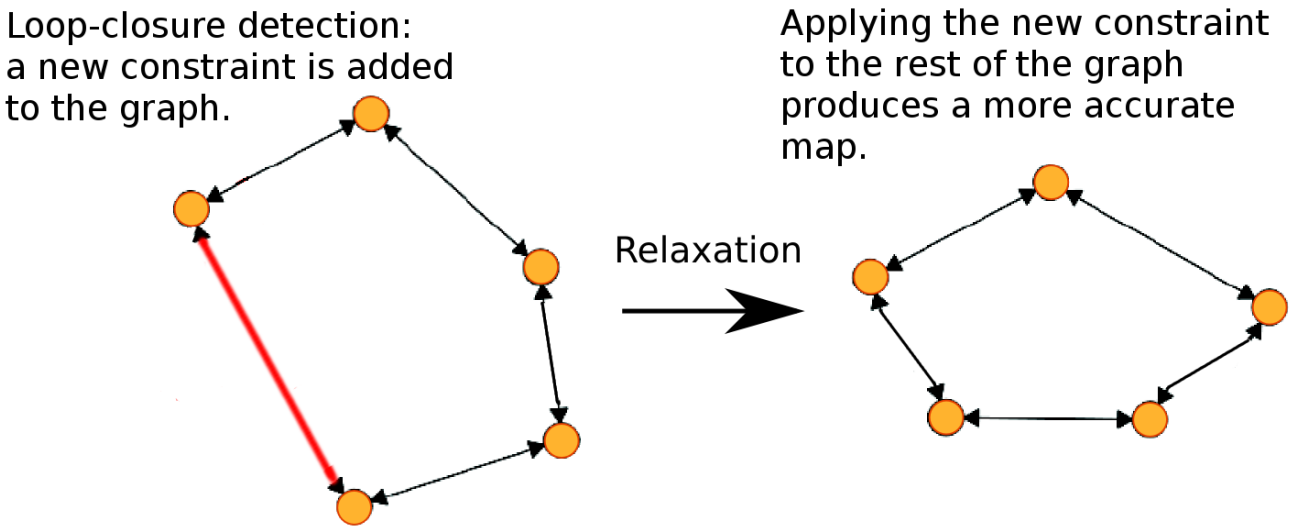


Mixed Indoor / Outdoor Topological Map, Several Levels



Adding Metric Information on the Edges

- Take advantage of odometry measurements from a wheeled robot to add relative displacement information between nodes.
- Apply simple graph-relaxation algorithm. to compute accurate 2D absolute positions for the nodes.



Relaxation Algorithm

1. Estimate position and variance of node i from each neighboring node j:

$$(x_i')_j = x_j + d_{ji}\cos(\theta_{ji} + \theta_j) \quad (y_i')_j = y_j + d_{ji}\sin(\theta_{ji} + \theta_j) \quad (\theta_i')_j = \theta_j + \varphi_{ji} \tag{1}$$

$$(v_i')_j = v_j + v_{ji} \tag{2}$$

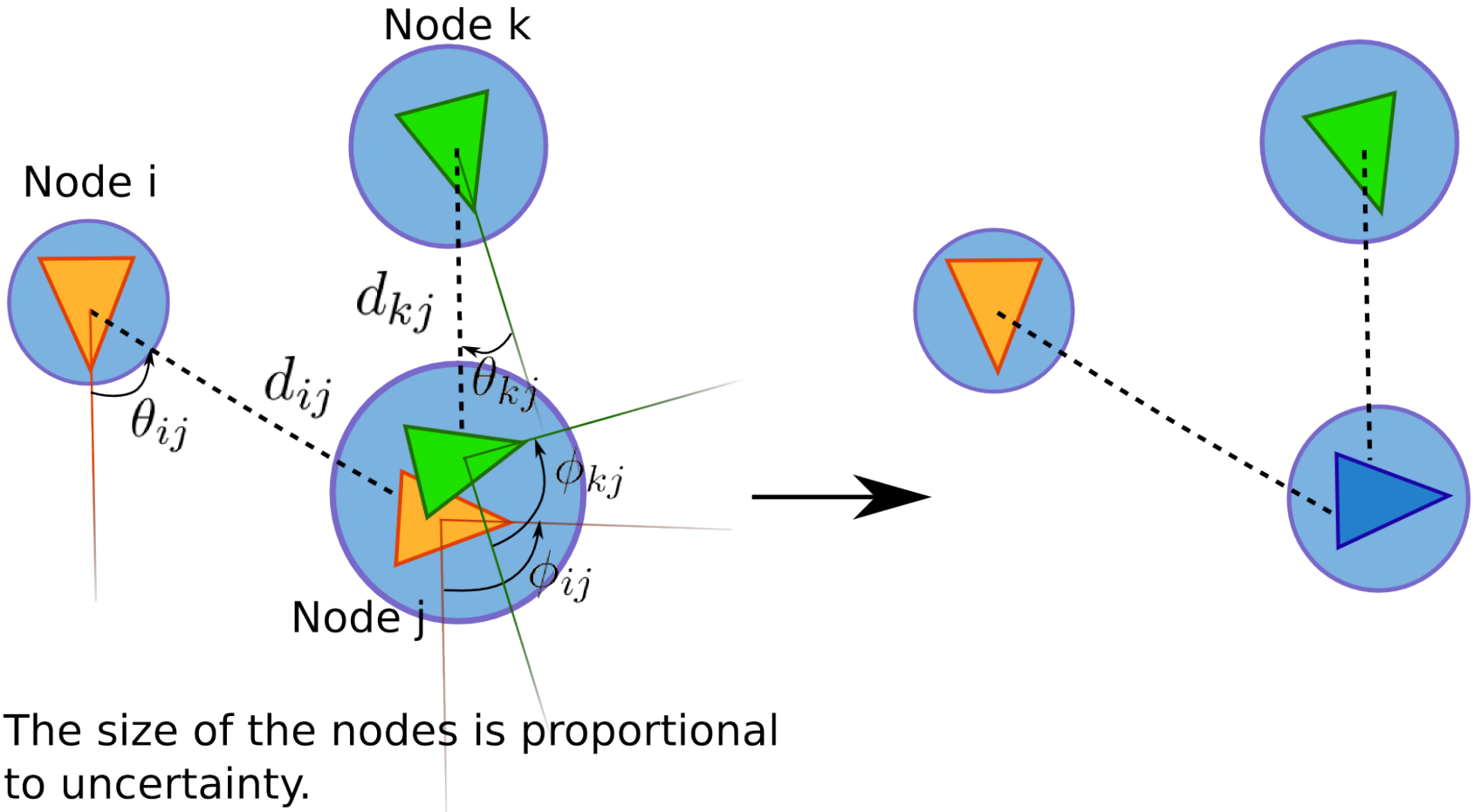
2. Estimate variance of node i using harmonic mean of estimates from neighbors (n_i = number of neighbors of node i):

$$v_i = \frac{n_i}{\sum_j \frac{1}{(v_i')_j}} \tag{3}$$

3. Estimate position of node i as the mean of the estimates from its neighbors:

$$x_i = \frac{1}{n_i}\sum_j \frac{(x_i')_j v_i}{(v_i')_j} \qquad y_i = \frac{1}{n_i}\sum_j \frac{(y_i')_j v_i}{(v_i')_j} \qquad \theta_i = \arctan\left(\frac{\sum_j \frac{\sin((\theta_i')_j)}{(v_i')_j}}{\sum_j \frac{\cos((\theta_i')_j)}{(v_i')_j}}\right) \tag{4}$$

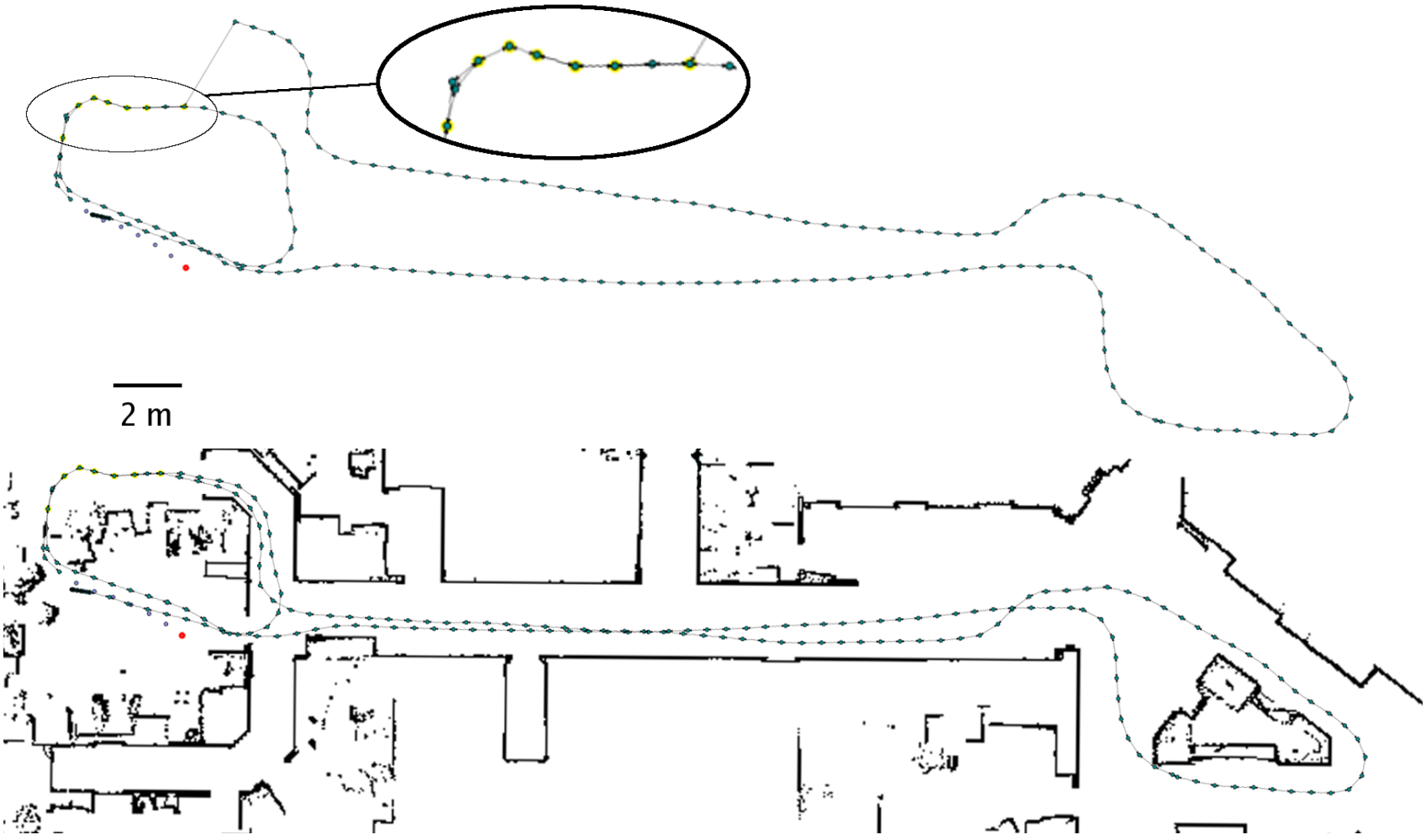
Relaxation Algorithm (Duckett, 2000): Illustration



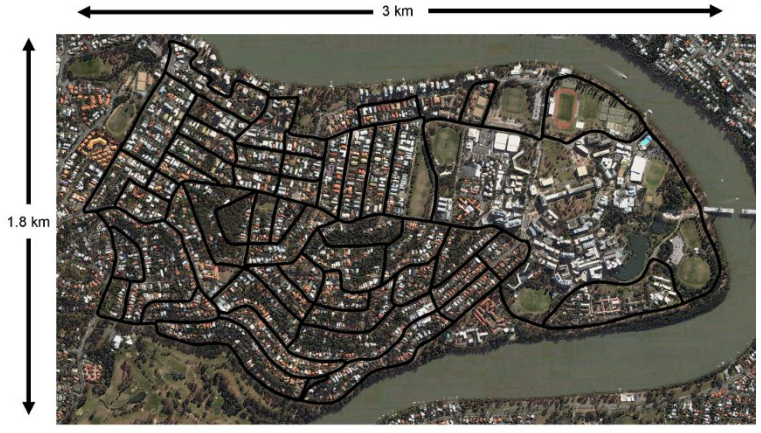
The size of the nodes is proportional to uncertainty.

The position and orientation of node j is obtained as the mean of the positions obtained from nodes i and k (i.e., by composing their positions with the corresponding relative displacements to node j).

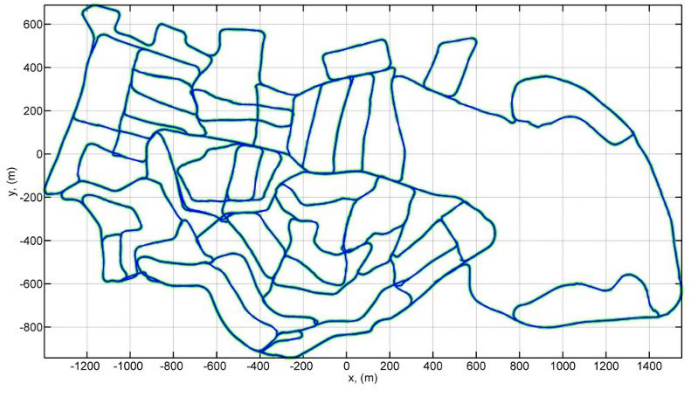
Map Relaxation: Good Odometry, One Loop Closure



Simple Large-Scale SLAM: RATSLAM



(a)



(b)

Milford and Wyeth, 2007.
`http://www.youtube.com/watch?v=-0XSUi69Yvs`

- Very simple 'visual odometry' gives rough trajectory.
- Simple visual place recognition provides *many* loop closures.
- Map relaxation/optimisation to build global map.