

CO333 - Robotics

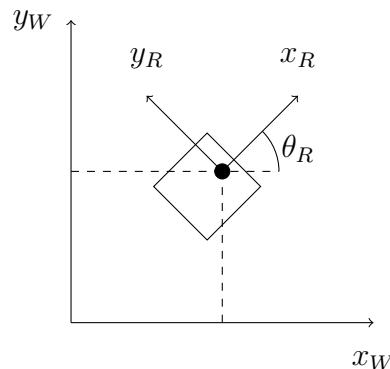
(60019)

Lecture 1 - Introduction to Robotics

Lecture 2 - Robot Motion

A definition of a robot is something that can **move** and **sense**, and uses some sort of information processing to link the two. Robots might want to move in the water, fly in the air, walk (legged) on land, or work in space. The course will focus on wheeled robots that work on generally flat surfaces.

Coordinate Frames



We will be mostly focused on 2D coordinates, on the flat (close to planar) ground. The world frame W is anchored in the world, and the robot frame R is anchored to the robot (consider one point and orientation as the centre of the robot) - consider a set of axis carried by the robot. Often we are interested in the robot's location; the transformation between the world frame W and the robot frame R .

Degrees of Motion Freedom

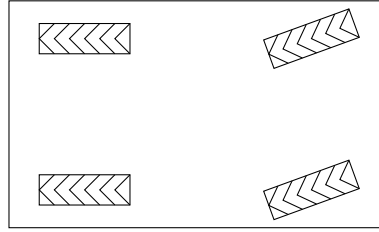
This is to do with how many parameters we need to specify the aforementioned transformation, generally related to the number of dimensions the robot is moving in. The simplest is a single degree of freedom, where a train moves along the x -axis - this position can be specified in one parameter. A rigid body which moves on a ground plane, such as an AV or robot vacuum cleaner has 3 DoF; two translational (x, y) and one rotational (typically θ). On the other hand, a rigid body which moves in 3D space has 6 degrees of freedom; three translational and three rotational.

A **holonomic robot** is able to move instantaneously in any direction in its space of DoF, otherwise it is **non-holonomic**. Most are non-holonomic, but some holonomic robots do exist; ground-based robots can be made with omnidirectional wheels.

Standard wheel configurations (both non-holonomic; each has two motors but has three degrees of movement freedom, the number of control inputs are lower than the DoF) include;

- **drive and steer** (car) - not implemented in this course

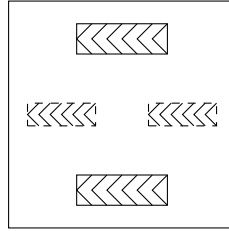
The combination of acceleration and braking determines how fast it moves forwards, and the orientation of wheel determines direction.



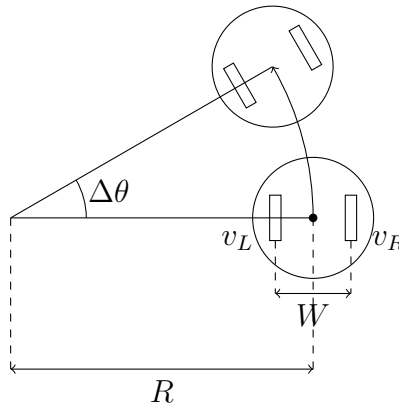
Rear wheels need a differential, variable (Ackerman) linkage for steering wheels.

- **differential drive** (robot vacuum)

Has two driving wheels, both pointing forwards, and maybe castors which keep it balanced. Robot moves in different ways depending on the different speeds the wheels are turning at.



The caster wheels (dashed) are passive, and simply support the robot. The active wheels have one motor each. If the active wheels are running at equal speeds, the robot moves in a straight line, and wheels running at equal and opposite speeds turn on the spot. On the other hand, in general, other combinations lead to motion in circular arcs / curves.



Consider the left wheel at speed v_L and the right wheel at v_R (linear velocities over the ground, hence $v_L = r_L \omega_L$, where r_L is the radius of the wheel and ω_L is the angular velocity). We also assume no slipping (the intersection is the centre of rotation).

We want to determine R , which is the radius of the circle formed by the robot's centre moving. Consider a small period of time Δt , and an angle of movement $\Delta \theta$;

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

By rearranging the above, and substituting back in, we have the following;

$$R = \frac{W(v_R + v_L)}{2(v_R - v_L)}$$

$$\Delta \theta = \frac{(v_R - v_L) \Delta t}{W}$$

Actuation (DC Motors)

DC motors are controlled by a power signal, using **PWM (pulse width modulation)** and a fixed voltage. A gearing system is typically used to **gear down** the end effector to be slower with higher torque (compared to the DC motor's rapid rotation, but low torque). Many motors have a built in encoders (connected to the motor), which can be read to measure angular position by counting steps. This is required as a running motor's rotations depend on many conditions (including the load it's driving); for example a heavier robot moving on rough ground will move slower. To measure the rate the motor is moving at, we can feed the rotation back. We can then use feedback control (servo control) to adjust the motor to make it do what we want. In principle, we want to determine where the motor is, and where it actually is. At a high rate, we want to send pulses to reduce the error between the target location and actual location.

PID (Proportional / Integral / Differential) Control

The PID expression sets the power as a function of error;

$$P(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de(t)}{dt}$$

In this, we have the following terms;

- $e(t)$ demand minus position (error)
For example, at each time step this is the angle we want the motor to be at minus the actual angle of the motor measured.
- k_p main term (high values give rapid response)
Continuing with the example above, $k_p e(t)$ will be higher if k_p is large. If there's a large error, send a large signal, and vice versa for small errors.
- k_i integral term (increased to reduce steady state error)
Imagine the motor is close to where it should be; this allows for the 'gap' to be closed.
- k_d differential term (reduced settling time / oscillation)

Wheel Rotation Speed to Velocity

Consider a wheel of radius r_w rotating at ω (in radians per second). The speed of the wheel, in theory, would be $v = r_w \omega$. However, in practice, there are factors such as uneven ground and tyre softness, the radius may be hard to measure accurately. Another consideration would be possible slipping; in general it will be better to **calibrate** this for the surface by considering some constant of proportionality.

State in 2D

If a robot is moving on a plane, we can define the robot with a state vector \mathbf{x} , with three parameters;

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \begin{array}{l} x \text{ component of robot centre point in world frame} \\ y \text{ component of robot centre point in world frame} \\ \text{rotation angle between coordinate frames (angle between } x_W \text{ and } x_R \text{ axes)} \end{array}$$

The two frames coincide when the robot is at the origin ($x = y = \theta = 0$). Note that $-\pi < \theta \leq \pi$. During a straight line period of motion of distance D , we have;

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + D \cos \theta \\ y + D \sin \theta \\ \theta \end{bmatrix}$$

On the other hand, during a pure rotation of angle α we have (rotation to the left is positive by convention);

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

Lecture 3 - Sensors (Behaviours)

Motivation

Recall the second practical; there was a gradual drift from the desired path. There are aspects of the world that the robot can never fully understand (such as the bumpiness of the floor) - even simulations cannot do exactly the same thing every time. After calibration, the robot should return to the desired location (on average), but some scatter will remain due to factors we cannot control.

Systematic errors are removed, however we still have **zero mean errors** - these errors occur incrementally (where every movement or rotation adds a small amount of potential error). The zero mean errors can be modelled probabilistically, often with a Gaussian distribution.

Uncertainty in Motion

A better model (compared to the previous lecture) adds uncertain perturbations / motion noise;

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + (D + e) \cos \theta \\ y + (D + e) \sin \theta \\ \theta + f \end{bmatrix}$$

We will likely also notice that rotations are not exact either;

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + \alpha + g \end{bmatrix}$$

Sensors

Proprioceptive are self-sensing (such as motor encoders / internal force sensors) and improve a robot's sense of its own internal state. In general, these measurements may depend on the previous states as (as well as the current state); for example wheel odometry gives a reading depending on a difference between the current and previous state, on the other hand, a gyro in an inertial measurement unit will report a reading depending on the current **rate** of rotation (not an instantaneous state).

In this case, the value of the measurement z_p should be a function of the state of the state of the robot;

$$z_p = z_p(\mathbf{x})$$

On the other hand, exteroceptive are outward-looking. These allow the robot to generally be **aware** of its environment, letting it localise with respect to a map, recognise objects / locations, map out free space, avoid obstacles, and interact with objects.

In this case, we need both the state of the robot \mathbf{x} and the state of the world \mathbf{y} (relative to the robot);

$$z_o = z_o(\mathbf{x}, \mathbf{y})$$

The state of the world may be parameterised (such as a list of geometric coordinates describing landmarks). This state may be uncertain.

Some sensors (such as touch, light, and sonar) will return a **single value** in a range. On the other hand, some sensors such as a camera or laser range-finder will return an **array** of values, with the former having an array of sensing elements (pixels of a camera's CCD chip), or the latter performing scanning.

Some common sensors are as follows;

- **touch sensor**

This is a binary on / off state; an open switch will have no current flow, and a closed switch will have current flow (hit).

- **light sensor**

This detects intensity of passive light incident from a single forward direction (some range of angle sensitivity); this is a continuous value. Multiple sensors can guide steering behaviours (such as driving towards a brighter light). In *Lego*, there is also a mode where the sensor can emit light, where reflections from close surfaces can be measured.

- **sonar (ultrasonic) sensors**

Sonar measures distance by emitting ultrasonic pulses (tiny pulses of sound at a high frequency) - these beams typically have an angular width of $10^\circ - 20^\circ$. The sensors measure the time for a pulse to bounce back (which the sensor uses to calculate the distance). These are fairly accurate within a few centimetres in one direction (for simple shapes) - but can be noisy in the presence of complicated shapes. The maximum range is a few metres.

- **laser range-finder**

The principle of this is similar to sonar, however it uses infrared laser beams instead of sound. This is also reflected, detected, and timed to calculate a distance. These are very accurate (sub-millimetre), and work on most surfaces. They can normally scan in a 2D plane (rotating on one axis), but can also scan in 3D. Previously, these were bulky and expensive, however nowadays they are present in modern smartphones.

Vision

This is the generalisation of a light sensor. A camera can be thought of as a large array of light sensors (or a light sensor is a single pixel of) which returns a large, rectangular array of measurements. A single camera measures light intensity (rather than direct information about geometry).

Bump Detection (Touch Sensor)

We can either perform this with a touch sensor, or by thresholding sonar (such as if the distance falls below some range).

Multiple touch sensors mounted inside a floating skirt can allow for detection of where an obstacle was hit. For example, if a robot was hit at the front, it may attempt to drive around it by going backwards, moving around, and continuing. Another simple strategy is to rotate through a random angle and go forward until the next collision (random bounce).

Servoing

Servoing is a control technique where control parameters (e.g. the desired speed of a motor) are coupled directly to a sensor reading and updated regularly in a **negative feedback loop** - this is also known as **closed loop control**. This requires high frequency updates of both the sensor and the update, otherwise the motion can oscillate.

Proportional control sets the demand proportional to negative error - for example in velocity (where we look at the difference between the desired sensor value and actual sensor value);

$$v = -k_p(z_{\text{desired}} - z_{\text{actual}})$$

This proportional case is a specific case of PID control.

A simple steering law to guide the robot to collide with the target would be as follows (for example - if the obstacle is straight ahead ($\alpha = 0$), then don't steer);

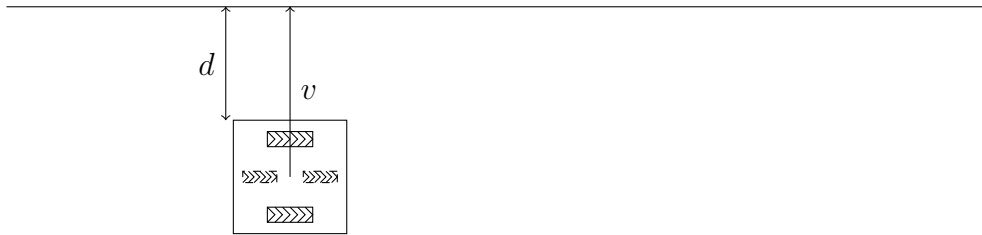
$$s = k_p \alpha$$

On the other hand, to guide the robot to avoid the obstacle at a safe distance (R) would subtract an offset (D is the distance from the robot to the object, and α is the angle between the robot's rotation and the direction to the object);

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

Wall Following

Consider the following scenario, where a differential drive robot desires to travel along a wall, at some distance d . This uses a **sideways-looking sonar** to measure some distance z . In the following diagram, the wall can be curved; I just don't know how to do that in TikZ.



Note that we are moving left to right (along the diagram), and we want to steer towards the wall (since we'd ideally have $d = v$). To do this, the left wheel must turn slower than the right wheel. Note that we can also achieve symmetric behaviour with a constant offset v_C (standard speed for a straight line). We can then set the left and right wheel velocities as follows;

$$\begin{aligned} v_R - v_L &= k_p(z - d) \\ v_R &= v_C + \frac{1}{2}k_p(z - d) \\ v_L &= v_C - \frac{1}{2}k_p(z - d) \end{aligned}$$

Combining

More complex robots may have multiple servos, and we can combine sensing-action loops. We can instead consider each local servo-like sensing-action loop as a **behaviour**. These behaviours can then be combined in an **arbiter**, which feeds the instructions into a motor controller.

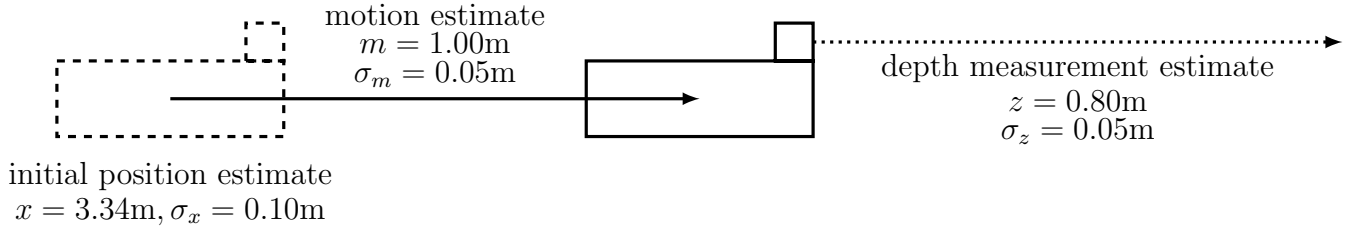
To go further, we need to go beyond a coupling between sensors and actions, and instead build a model of the world. Here we plan a sequence of actions to achieve a goal, and then execute the plan. However, if the world changes during the execution, we need to re-plan the actions.

Lecture 4 - Probabilistic Robotics

From the previous lab, we notice an issue when the robot is at a large angle to the wall; we realistically want the perpendicular distance between the robot and the wall, but this gets much larger if the robot is at a large angle. One solution is to mount the camera in front of the robot, as this couples rotation with distance.

Probabilistic Localisation

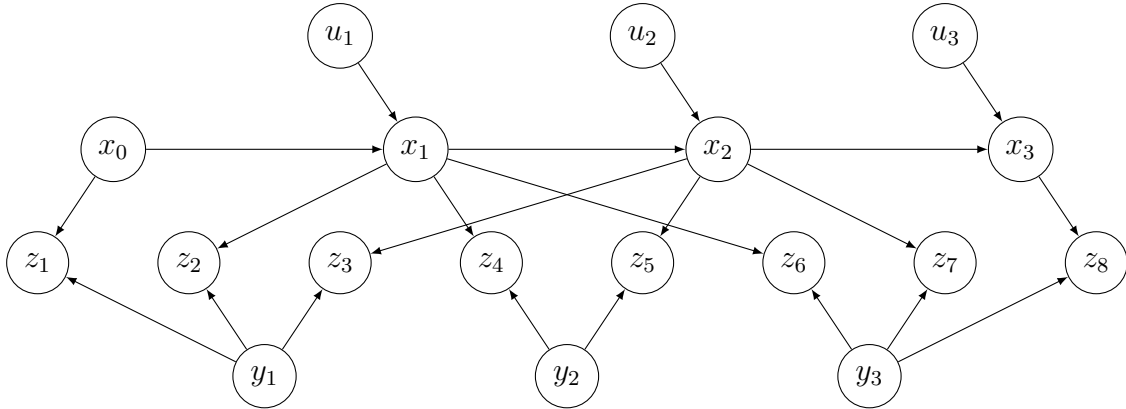
Notice from the previous labs that attempting to estimate the location of the robot can lead to errors adding up. We assume some knowledge of the scene, but we do not know the location of the robot relative to the map. Long-term estimation based on logical reasoning fail when presented with real data; advanced sensors cannot be easily analysed like bump and light sensors, and all information received is uncertain (every action as well as every sensor measurement). A probabilistic robot acknowledges the uncertainty (and uses models to abstract useful information).



When we start with an uncertain position, and make an uncertain motion, the robot will have an uncertain state measurement. Once the robot then performs some measurement, there will likely be some uncertainty in the result.

Probabilistic Inference

We think of what we want to estimate, and then their probabilistic relation to things we can actually observe. The following graph is one visualisation of a SLAM problem;



Note that in the above diagram, the nodes x_i denote the estimate of the location of the robot at some point in time. We are also interested in estimating the locations of landmarks in the world (denoted by y_i - fixed landmarks). The sensors (such as camera or sonar) will make some sort of measurement relative to the position robot (z_i) - for example, at x_1 the robot observes y_1, y_2, y_3 with measurements z_2, z_4, z_6 respectively. Finally, the control inputs (for the movement) are denoted by u_i .

The process of **sensor fusion** combines data from many different sources to create useful estimates. This state estimate can be used to decide the next action.

Probabilities describe our state of knowledge (not randomness, which is rare in the real world). For example, the physics for a coin flip are deterministic, however without this knowledge, we can only guess the outcome.

$$P(XZ) = P(Z | X)P(X) = P(X | Z)P(X) \Rightarrow \underbrace{P(X | Z)}_{(1)} = \frac{\overbrace{P(Z | X)}^{(2)} \overbrace{P(X)}^{(3)}}{\underbrace{P(Z)}_{(4)}}$$

- (1) posterior
- (2) likelihood
- (3) prior
- (4) marginal likelihood

In the state diagram, we have things we are interested in estimating (state variables) and things we can get from sensors (measurements / observations). Typically state variables are represented with X in the probabilities, with measurements being represented by Z . Baye's rule is used to incrementally digest new information; we have a prior estimate of X already - $P(X)$. From this, we want to get the

probability of X given Z - $P(X | Z)$. We want to obtain $P(Z | X)$, which is the likelihood of the measurement given the state variable.

Consider the following example;

kitchen	bathroom	living room
$P(K) = 0.3$	$P(B) = 0.2$	$P(L) = 0.5$

The robot starts completely lost, let the probability of it being in each room be proportional to the area (note that the probabilities are normalised). Assume we also have some light sensor (which gives a thresholded value of it either being light or dark), which has had experiments performed in each of the rooms to obtain probabilities. Assume the sensor also now reports it as being bright (hence we want to find the probability of it being in a room, **given** that it is bright).

$$P(b | K) = 0.8 \quad \text{bright 80\% of the time in the kitchen}$$

$$\begin{aligned} P(K | b) &= \frac{P(K)P(b | K)}{P(b)} \\ &= \frac{0.3 \times 0.8}{P(b)} \\ &= \frac{0.24}{P(b)} \\ &= 0.4 \end{aligned}$$

$$P(b | B) = 0.3 \quad \text{bright 30\% of the time in the bathroom}$$

$$\begin{aligned} P(B | b) &= \frac{P(B)P(b | B)}{P(b)} \\ &= \frac{0.2 \times 0.3}{P(b)} \\ &= \frac{0.06}{P(b)} \\ &= 0.1 \end{aligned}$$

$$P(b | L) = 0.6 \quad \text{bright 60\% of the time in the living room}$$

$$\begin{aligned} P(L | b) &= \frac{P(L)P(b | L)}{P(b)} \\ &= \frac{0.5 \times 0.6}{P(b)} \\ &= \frac{0.3}{P(b)} \\ &= 0.5 \end{aligned}$$

However, note that the sum of the posteriors must also be equal to 1, hence;

$$P(K | b) + P(B | b) + P(L | b) = 1 \Rightarrow \frac{0.24 + 0.06 + 0.3}{P(b)} = 1 \Rightarrow P(b) = 0.6$$

Note that in the example above, we've digested information that suggests the robot being in the kitchen (due to the brightness), hence the probability of it being in the kitchen has gone up, whereas the probability of it being located in the bathroom has gone down.

Probability Distribution

Discrete interference generalises to more possible states as the bins get smaller and smaller (for example, if we were to divide the floorplan into more zones). As the width of the bins tend to 0, we have a continuous probability density function $p(x)$ at the limit. The probability that a continuous parameter lies in the range a to b is;

$$P_{a \rightarrow b} = \int_a^b p(x) \, dx$$

However, as the resolution becomes higher, it becomes more expensive in terms of memory and computation.

Gaussian distributions often represent the uncertainty in sensor measurements well, parametising a 1D distribution in two parameters (μ and σ);

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

A Gaussian prior multiplied by a likelihood will produce a posterior tighter than both. The product of two Gaussians is **always** another Gaussian. However, there is only one peak, which may not always fit.

Particle Filtering

Another representation is to use particles. A probability distribution here is represented with a finite set of weighted samples (typically $N = 100$) of the state;

$$\{\mathbf{x}_i, w_i\} = \left\{ \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix}, w_i \right\} \text{ where } \sum_{i=1}^N w_i = 1$$

Instead of storing a large table of numbers, we almost entirely ignore where the probability is low (close to 0), and focus on the areas where there is higher probability. We want to store the probability distribution within the degrees of freedom of the robot. The idea here is to test the hypothesis of where the robot is (the cloud of points) and narrow it down based on sensor observations - a good estimate is a tight clump of particles.

In general, these steps are repeated every time the robot moves and makes measurements;

1. motion prediction based on **proprioceptive** sensors

With blind motion, the particles should spread out more and more (due to a growing uncertainty). For this, we use the state changes with the zero mean noise terms (e, f, g) - these are generated randomly for **each** particle.

Also note that the robot may move through variable step sizes, and we will need to scale the variance accordingly.

2. measurement update based on outward-looking sensors
3. normalisation
4. resampling

Redistributes positions of particles - particles will spread out under motion, but will acquire different weights based on how well they agree with the sensor measurements, according to Bayes' rule (higher weights for more probable locations).

At any point in time, our uncertain estimate of the location of the robot is represented by the whole particle set; we can make a point estimate of the current state by taking the (weighted) mean of the particles;

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

This state can be used to plan waypoint navigation.

Position-Based Path Planning

Assume the current state of the robot is (x, y, θ) , and we want to travel to a waypoint set at (W_x, W_y) . We need to first rotate towards the waypoint, with the following direction vector;

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

This can be used to obtain an **absolute** angular orientation α , which the robot must drive in;

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

We need to make sure that α is in the correct quadrant of $(-\pi, \pi]$, as a standard \tan^{-1} function would give a value in the range $(-\frac{\pi}{2}, \frac{\pi}{2}]$. This can also be achieved with `atan2(dy, dx)` in many languages.

Using this, we can obtain the angle the robot must rotate through as β (and we should take care to shift this angle into the range $(-\pi, \pi]$ for efficient movement). The robot should then drive through a distance d in a straight line;

$$\beta = \alpha - \theta$$
$$d = \sqrt{d_x^2 + d_y^2}$$

Lecture 5 - Monte Carlo Localisation

Lecture 6 - Advanced Sonar Sensing

Lecture 7 - SLAM