G53ARS Notes

Craig Knott

Version 1.0
December 24, 2013

Contents

1 What is a robot?

The word "robot", originates from the 1921 play, "Rossum's Universal Robots" by playwright Karel Capek. It is derived from the Czeck words "robota", meaning "forced labour" and "robotnik", meaning "serf" (someone who needs to renders services to a lord or superior). In the 1942 novel "Runaround", by Isaac Asimov, three basic "laws of robotics" were defined. These were:

- A robot may not injure a human being or, through inaction, allow a human being to come to harm
- A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Seconds laws.

As a result of progress in the field of robotics, robot ethics has become a widely discussed issue.

According to M.J.Matarić, "A robot is an autonomous system which exists in the physical world, can sense its environment, and can act on it to achieve some goals".

1.1 Autonomy

According to the dictionary, "autonomous", means: actions that are undertaken without outside control. In robotics, this generally implies the necessity to carry on-board sensors, controllers and power supplies so that the robots can maintain themselves. Autonomous also means that the robot has the power of self-government, meaning it is able to adapt to changing environments, determine its course of action by its own reasoning process, have the ability to build internal representations of the world, and possess the ability to learn from experience and plan new actions.

1.2 Navigation Basics

Navigation of robots is broken down into four distinct categories:

- Perception
  The robot must interpret its sensors to extract meaningful information
- Localisation
  The robot must determine its position within the environment
- Cognition
  The robot must decided how to act to achieve its goals
- Control
  The robot must modulate its motor outputs to achieve the desired trajectory

2 The study of robotics

There are many different reasons for the study of robots, both theoretical (to investigate intelligent behaviour) and applied (to create robots to be used in hostile environements). We use intelligent agents within robots to control them. The word "agent" means, "to do", meaning that the agents within the robots are entities that produce some effect (on the robot). Consequently, "agent" is used to describe both software simulations and/or actual hardware implementations of robots (the robot being the physical machine, and the agent being both the physical machine, and the numerical computer model).

It is often debated whether or not we should use software agents as the primary mechanism to investigate robots, as this would be both cheap, and flexible. However, this come with the disadvantage that there would be subtle differences in transferring the software researched mechanics of the robots to physical ones, being that the simulated world is much different to the real world. Many influential people in the field of robotics (for instance, Brooks) feel that true intelligent behaviour can only emerge when a physical agent interacts with its environment. It is important at this point to understand what is meant by intelligence.

2.1 What is Intelligence?

"The extent to which we regard something as behaving in an intelligent manner is determined as much by our own state of mind and training as by the properties of the object under consideration".

"If we are able to explain and predict the behaviour of an object, or if there seems to be little underlying plan, we have a little temptation to imagine intelligence. With the same object, therefore, it is possible that one person would consider it as intelligent and another would not. The second would have found out the rules of its behaviour".

2.2 Components of robots

Robots are comprised of three main component classes:

1. Sensors
   A device that gives a signal for the detection or measurement of a physical property to which it responds. These provide the input to the robot, for instance, a light sensor.
2. Control Hardware & Software
   Programmed behaviours in the data/memory of the robot. Makes decisions for the robot.
3. Actuators
   A "thing" which moves to mechanical action, communicates motion to, or impels (an instrument, machine, or agent). This effects the outputs from the robot, for instance, a motor.

The robot, its tasks, and the environment all interact and influence each other - much like a spider in a bath.

2.3 General Purpose vs Purpose Built

It is not really possible to build a general purpose robot; just like there are no natural "general purpose living things". Humans are possibly the most intelligent living creatures, but we cannot fly or swim for instance. A robot's function and operation are defined by its own behaviour within a specific environment, taking into account a specific task. Only the simultaneous description of a robot, its task, and the environment describes the robot completely.

3 Environments

There are many different types of environments in which a robot many be required to operate. Environments are typically categorised by their degree of structure. Although there is no solidly accepted definition of structures, environments can be split into one of the following categories: structured, unstructured, partially structured.

1. Structured Environments
   A structured environment is one which has been specially designed for a robot to operate in, such as an artificial maze, where an exact description of the environment can be supplied to the robot during its design phase. This can mean that very little, or perhaps no, sensor data may be required. There are usually no unexpected or unplanned dynamic aspects to the environment: the robot has been told in advance of how and when the environment will change, and how to deal with it.

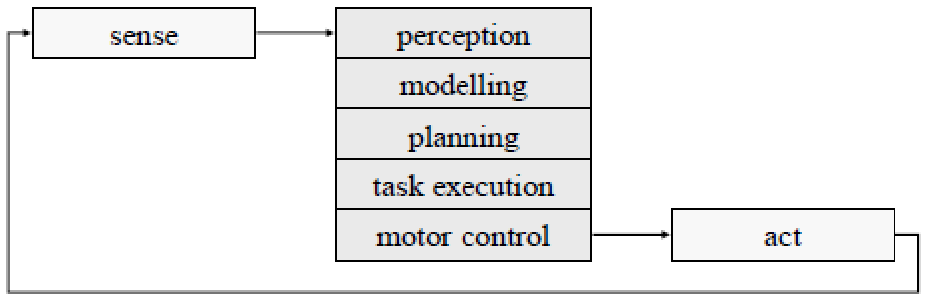
2. Unstructured Environments
   Complex environments for which no models or maps exist, or can even be accurately generated, the robots must therefore operate in response to real-time sensor data. Such environments usually have significant dynamic changes, like the natural world rather than simulated worlds. They also have many unknown attributes (deep-sea exploration for instance), or potentially completed unknown (planetary probes for instance).

3. Partially Structured
   Somewhere between the two previous extremes; this is an environment which may be modelled to a certain extent, but with insufficient model detail to fully support task completion. Possibly the static component of the environment has been modelled, but the dynamic changes are unpredictable and must be sensed, for example: a factory floor with in-built tracks to follow, but with unpredictable obstacles to avoid (humans, for instance).

4 Control Models

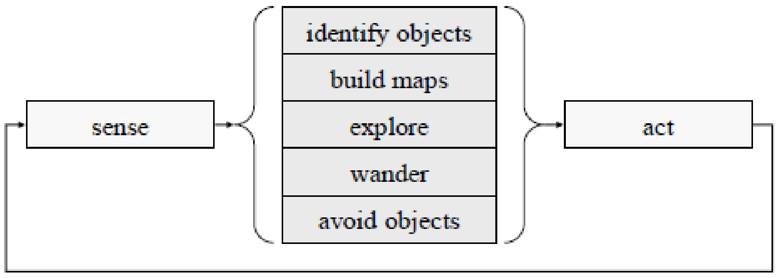
A fundamental methodology derived in the early days of robotics from engineering principles is the sense-think-act cycle. The principle is to continuously attempt to minimise the error between the actual state and the desired state, based on control theory. a variety of different approaches have been tried for implementing the sense-think-act control cycle. These approaches can be categorised as: model-based, reactive, or hybrid.

1. Model Based - Think hard first, then act
   A symbolic internal "world-model" is maintained. The sub-tasks are decomposed into functional layers, similar to a "classical" artificial intelligence approach. The problem with a model-based control method is that the model must be adequate, accurate, up-to-date and maintained at all times. This is very difficult in practice, and is very error prone (for instance, an unrecognised object appearing would cause havoc). A model-based system is very brittle, and if one of the functional layers fails, then the whole system will fail. It also requires significant processing power to maintain the model, which could cause slow response times.



2. Reactive Robotics - Don't think, react (The Zerg of robotics)
   In order to try to overcome the shortcomings of model-based robots, modern approaches have centred on simple reactive systems with minimal amounts of computation. Instead of maintaining some huge model, a small, simple, implicit model is maintain. The symbols do not use symbolic models but, for example, a rule-set which tells a robot how to react to a given situation. It implicitly encodes assumptions about the environment, but does not define an explicit symbolic model for a wall or corner, for instance. Limitations to this approach are that such robots, because they only look up actions for any sensory input, do not usually keep much information around, have no memory, no internal representations of the world around them, and no ability to learn over time.

3. Behaviour Based
   The control system is broken down into horizontal modules, or behaviours, that run in parallel. Each behaviour has direct access to sensor readings and can control the robot's motors directly. This supports *multiple goals* and is more efficient. There is no functional hierarchy between the layers (they do not call each other), and each layer can work on different goals, in parallel. These systems are easier to design, debug and extend, as each module can be designed and tested individually. These systems are also more robust, as one module failing will not affect any others. The limitations of this approach is that it is extremely difficult to implement plans, because behaviour based robots have no memory and are unable to follow an externally specified sequence of actions. It can also be very hard to predict how a large number of multiple behaviours may interact; the behaviour produced when all these systems interact is called "emergent", which is sometimes useful, and sometimes not. Robots can also get trapped in a "limit cycle", which is where robots become trapped in a dead-end, performing the same actions over and over.



4. Hybrid Approaches
   There are many different approaches that attempt to combine elements from reactive systems, and model-based systems. for instance, the SSS three-layer architecture (servo-subsumption-symbolic) combines Brook's architecture with a lower-level servo control level and a higher-level symbolic system.

   There is also a fuzzy logic and neural network controller system, that using fuzzy logic rule-bases and neural networks to take inputs from sensors and process data to generated outputs to actuators.

5 Behaviour Coordination

As seen in the previous section, there exist behaviour-based architectures that enable multiple behaviours in parallel, but this raised the issue of which behaviour should be executed first, and how can we coordinate multiple behaviours. We can apply weighting to certain behaviours, or apply scheduling algorithms like first-come, first-served, but it generally depends on the context. There are two main approaches to answer these questions: Behaviour Fusion, and Behaviour Arbitration.

Behaviour Fusion focuses on combining multiple behaviours into one single output behaviour, based on some form of aggregation. The challenge with this method is picking a suitable aggregation method, and designing all the behaviours in such a way that they are suitable for combination.

Behaviour Arbitration however, selects a single behaviour from the possibilities, this is then the designated output behaviour. This method is sometimes referred to as "competitive behaviour coordination", as the behaviours can said to be competing for the output. There are two main subtypes of this methods: fixed priority hierarchy, where priorities are fixed or known at run time; and dynamic hierarchy, where priorities can change at run time.

5.1 Brook's Subsumption Architecture

There are several assumptions that must be made whilst using Brook's subsumption architecture, these include. Appendix A has a mnemonic for remembering these:

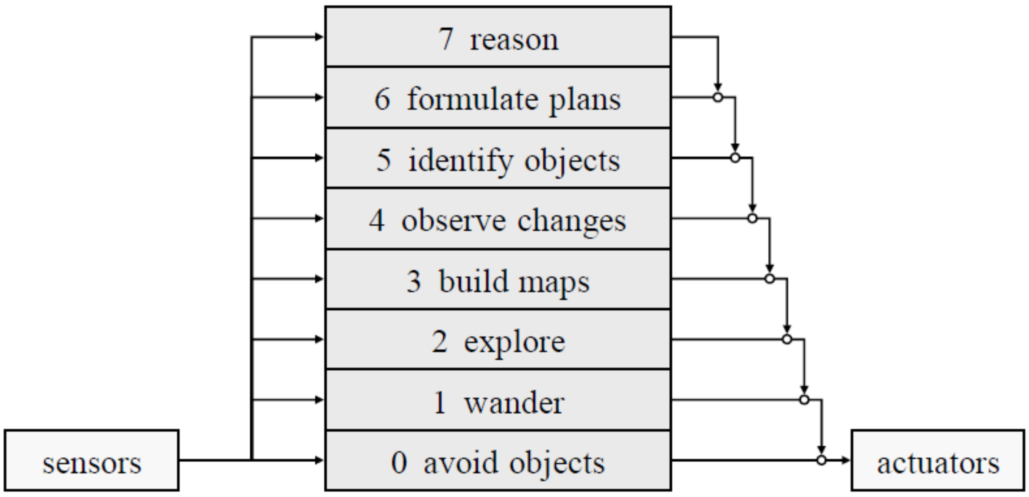
1. **Complex** and useful **behaviour** need not necessarily be a product of an extremely complex control system
2. Individual components should be **simple** and not ill-conditioned[1] (if a single module is growing too large, rethink the design)
3. Ability to wander in real environments is vital - **map making** is crucial
4. The real world is **three dimensional**
5. Absolute coordination systems are a source of error - **relative mapping** is much better
6. The real world is not constructed of exact shapes, thus, **no artificial environments** will be constructed for the robot
7. Just because data is easy to collect, does not mean it leads to rich descriptions of the world. **Visual data** is better than sonar
8. For robustness, the robot must be able to function when one or more of its **sensors fail**
9. Robots should be autonomous and **self-sustaining**

Brook's architecture is a robust layered control system for mobile robots that specifies levels of competence, to help break down the complexity of autonomous robotics. Brooks chose to decompose problems vertically; it is sliced up on the basis of desired external manifestation of the control system. A number of levels of competence are defined, which are informal specifications of a desired class of behaviours for a robot over all environments it will encounter. A higher level of competence implies a more specific desired class of behaviours. An example of a set of competences is shown below (Note that each level of competence includes a subset of each earlier level, they can also provide additional constraints on lower levels):

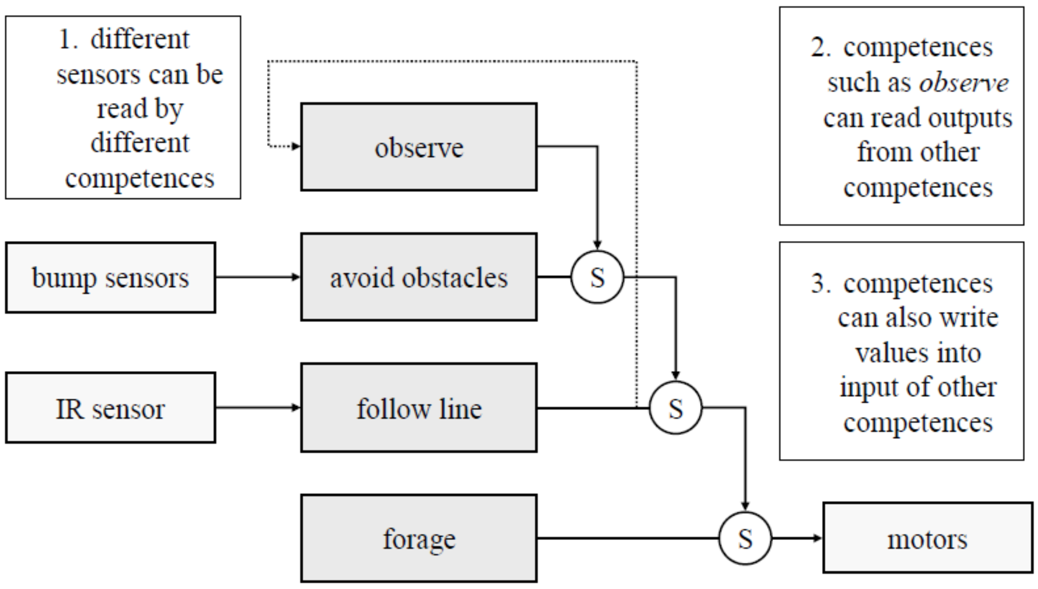
Reason about the behaviour of objects in the world and modify plans accordingly
Formulate and execute plans that involve changing the state of the world in some desirable way.
Reason about the world in terms of identifiable objects and perform tasks related to certain objects
Notice/Observe changes in the "static" environment
Build a map of the environment and plan routes from one place to another
Explore the world by seeing places in the distance that look reachable and heading for them
Wander aimlessly around without hitting things
Avoid contact with (stationary or moving) objects

[1]Not thought through/working well

The key idea of the levels of competence is that layers of a control system can be built to correspond to each level. Control systems can be built to achieve level zero competence, this can be programmed, debugged and then fixed in operation. Another layer can then be added, which can use data from the level zero system, and can then inject output, or suppress the level zero output. The level zero system is unaware of the suppression and continues to run. This is the main idea behind Brook's subsumption architecture. We say that the higher level subsume the roles of the lower level layers when they wish to take control. This system can be partitioned at any level, and the layer below form a complete operational system. This architecture allows for multiple goals, use of multiple sensors, robustness, and extensibility.



Another advantage of this architecture is that not all desired perceptions need to be processed by each level of competence, meaning different decompositions can be used for different sensor-set task-set pairs. Layers can also be built on a set of small processors, each one sending a small message to the other, meaning there is no need for central control. There is also much room for extension.


1. different sensors can be read by different competences
2. competences such as *observe* can read outputs from other competences
3. competences can also write values into input of other competences
observe
bump sensors
avoid obstacles
IR sensor
follow line
forage
S
S
S
motors

5.2 Finite State Machines

Similar to Deterministic Finite Automata, a Finite State Machine can be use to describe the logic and functioning of modules within an autonomous robotic system. These are defined as:

$$D = (Q, \Sigma, \delta, q_0, F)$$

where

$$\begin{aligned} Q &= Set\ of\ states \\ \Sigma &= Input\ symbols \\ \delta &= Transition\ function \\ q_0 &= Initial\ state \\ F &= Set\ of\ final\ states \end{aligned}$$

6 Robotics Hardware

6.1 Sensors

Sensors are physical devices that measure physical quantities, for example:

Sensing Technology	Physical Quantity
Bump sensors, switches	Contact
Ultrasound, Infra-red, Laser Range Finder	Distance
Photocells, Cameras	Light Level
Thermal, Infra-red	Temperature
Microphones	Sound Levels

As physical devices are operating in the real world, sensors are subject to noise and errors. Even two two sensors made in the same place could be different. As a result of this, robots cannot be certain about the properties of the environment or even its own state in respect to this environment. There are many sources of uncertainty in robotics, including: sensor noise, errors and limitations; effector and actuator noise; and, changing/dynamic environments.

Sensors vary in the amount of information they produce. For example, a bump sensor produces one bit of information (touching, or not touching), whereas a camera produces a huge amount of information (a VGA camera of 640x480 resolution will still produce 307,200 pixels of data per second of video - and that's without considering colours). Generally you would think that the more information, the better. However, in robotics, that is not always the case, because it can often mean higher processing requirements.

There is another problem that sensors face, characterised by the "signal-to-symbol" problem. "Symbols" are useful blocks of information that define an object, for instance, a person "Tom". Unfortunately, sensors provide data, and not symbols, and the translations between data and symbols is very difficult and requires a lot of computation. Often this means that symbols are not used in robotics.

6.2 Sensor Types

There are two main types of sensors, active and passive. Active sensors generate an active signal and measure the interaction of this signal with the environment (light sonar or light sensors). Passive sensors (also known as in-active) measure a physical property of the environment (such as thermal or sound sensors).

6.2.1 Switches

Switches are a type of sensor that are either open or closed. As mentioned before, they have a single bit of information, and are either on or off. Many types of switches exist in robotics: bump switches, a simple on/off sensor that detects touch; limit sensors, which detect *how much* a particular sensor is being touched; and whisker sensors, which work like cat whiskers in the way that there are multiple sensors feeling at once that can detect where objects are.

6.2.2 Light Sensors

Light sensors are based on photocells, and convert light into voltage. The voltage (an analogue measurement) is converted to a digital medium by an Analogue To Digital converter. Many different types of light sensors exist, measuring different properties of lights, including intensity, differential intensity, and break-beam sensors. Light sensors can be either active or inactive, depending on what reading they wish to take.

Due to the differing light levels that exist in different environment, it is important to calibrate light sensors. This is generally achieved by taking limits of light and dark.

6.2.3 Distance Sensors

Most distance sensors work by measuring *Time of Flight*, of a signal travelling at a known speed, S. The distance can then be determined in accordance to the formula:

$$Distance = (ToF * S)/2$$

For instance, knowing sound travels at 340.29m/s (in generic conditions), and in some case, the ToF is 0.005s, we can determine the distance between the sensor and the object is 0.85m.

$$\begin{array}{ll} & (0.005 * 340.29)/2 \\ = & \{ \quad \text{applying multiplication} \quad \} \\ & 1.70145/2 \\ = & \{ \quad \text{applying division} \quad \} \\ & 0.850725 \end{array}$$

There are, however, some problems with sonar distance detections, and these are that irregular shapes can cause the waves to reflect in unexpected ways, and angled shapes could cause only the closest distance to be returned, instead of the actual. Other distance sensors include laser range finders, and infra-red distance sensors.

6.3 Actuators

An effector is a device that has some effect on the environment, such as a gripper, or wheels. An actuator, on the other hand, is the mechanism that drives these effectors, for instance motors or hydraulics. There are two types of actuation: passive, where the energy for the effector/actuator is delivered through interaction of the effector with the environment; and active, where the energy for the effector is generated by the actuator.

6.3.1 Motors

Motors have a maximum speed (called the no-load speed) and a maximum torque (the stall torque). The no-load speed of a motor is the speed a motor will run when there is no weight to carry, and the stall torque of a motor is the torque produced when the rotational speed is zero (It may also mean the torque load that causes the output rotational speed of a device to become zero).

DC Motors

The most common type of actuator in robotics are DC motors (standing for direct current). These motors produce a continuous rotational motion, however in robotics, a more controlled motion is generally diserable, which is where servo-motors come in.

In DC motors, the motor speed ω is proportional to the induced voltage (the energy carried by the charge), V, in the following way:

$$\omega = k_v * V$$

And the torque (rotational force), t, is proportional to the applied current (rate of flow of charge), I, in the following way:

$$t = k_I * I$$

The speed of a DC motor is generally controlled using PWN (Pulse Width Modulation), and the direction of rotation is controlled using an H-bridge (implemented in electronics).

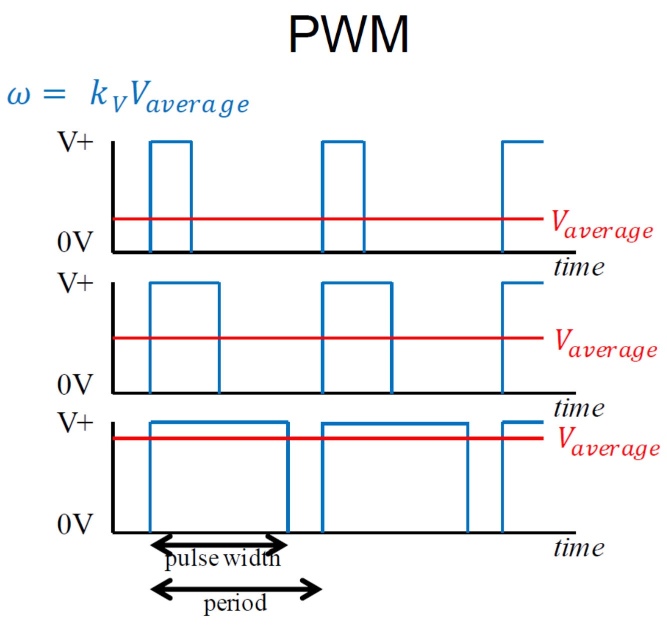
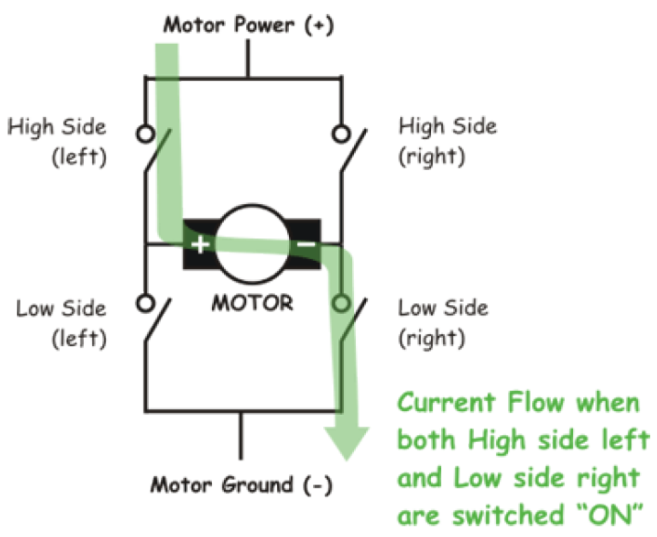


Figure 1: Pulse Width Modulation Example



Top Left	Top Right	Bottom Left	Bottom Right	Description
On	Off	Off	On	Motor goes clockwise
Off	On	On	Off	Motor goes anti-clockwise
On	On	Off	Off	Motor brakes, decelerates
Off	Off	On	On	Motor brakes, decelerates

Figure 2: H-Bridge picture and truth table for output

Servo Motors
Servo motors are used to move the drive-shaft of the motor to a specific position, allowing for much more precise movement. Servo motors *are* Dc motors, but with additional components (gears, position/shaft encoders[2] and electronics to control the motor). They allow for accurate position control, but makes the actuators very stiff.

6.3.2 Motor Gearing

The figure below shows the speed to torque relationship.

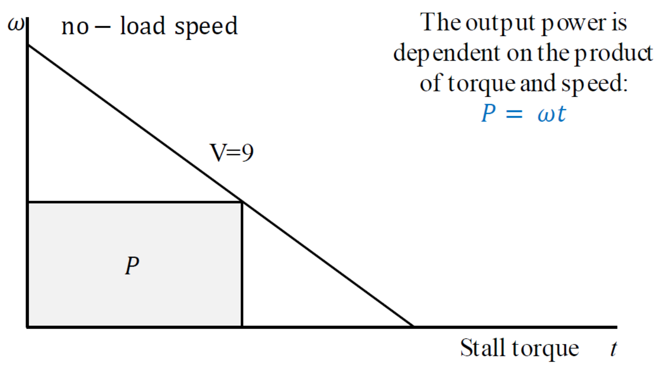


Figure 3: Speed to Torque Relationship

[2]A shaft encoder is a device that detects the rotation of a shaft by counting the number of times an optical switch is closed and opened. They can be used to measure rotation too, if given a second sensor and a second row of holes

However, in robotics we generally need low speed and high torque, this means we require *gearing* to alter the relationship between torque and speed, as shown below.

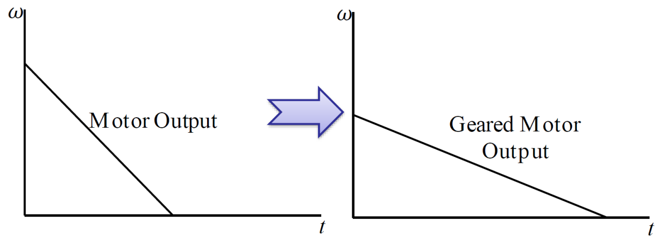


Figure 4: Speed to Torque Relationship - Geared

The linear speed, v, at the edge of a gear with radius, r, can be equated to:

$$v = \omega r$$

The Force, F, at the edge of a gear with radius, r, can be equated to:

$$F = \frac{t}{r}$$

Two meshed gears have the *same* linear speed, it is only the rotational speed that differs between them. Knowing this fact, we can work out the output speed of one gear, entirely through knowing the output speed of a connected gear, using the formula:

$$\omega_2 = (\frac{r_1}{r_2}) * \omega_1$$

Similarly, we can determine the output torque of a gear, using the torque of a connected gear, using the formula:

$$t_2 = (\frac{r_2}{r_1}) * t_1$$

The expression $(\frac{r_2}{r_1})$ is known as the *Gear Ratio.* Using this ratio, different combinations of speed and torque can be achieved using the same motor, using gears. The process of *gearing up*, where the input gear is twice the size of the output gear will half the torque and double the speed. On the other hand, *gearing down*, where the input gear is half the size of the output gear, will half the speed, and double the torque.

6.3.3 Wheels

In the field of robotics, there are four main varieties of wheel: standard, caster, Swedish (Mecanum), and Spherical. The standard wheel and castor wheels have two degrees of freedom: rotation around the motorized axle, and around the point of contact. Both of these wheels have a primary axis of rotation, and are thus highly directional. To move in a different direction (left or right), the wheels must be first steered along a vertical axis (for instance, turning left requires the powering of the right hand motor forwards). The standard wheel can accomplish this steering with no side effects, as the centre of rotation passes through the contact path with the ground, whereas the castor wheel rotates around an offset axis, causing a force to be imparting to the robot chassis during steering. The Swedish wheel and spherical wheel are less constrained by directionality than the other two wheels. The Swedish wheel functions as a normal wheel, but provides low resistance in another direction as well. The spherical wheel is entirely omnidirectional, for instance, a computer mouse wheel.

The choice of wheel greatly affects the robot in three ways: stability, controllability, and manoeuvrability.

7 Degrees of Freedom

In mechanical terms, the degrees of freedom represent the number of variables that determine the state of a physical system. Informally, it can be thought of as the directions in which a system can move. In three dimensional space, an object generally will have six degrees of freedom, split into two groups: translation (x, y, and z positions), and rotation (roll, yaw, and pitch). Effectors can have one or more degrees of freedom.

Degrees of freedom are split into two categories, controllable and uncontrollable, which have rather self explanatory names. For instance, if we think of a driving car, we have a forward degree, x, a left and right degree, y, and a rotational degree, θ. It is obvious that we have control over both x and θ, as we can drive the car forward, and we can turn the car with the steering. However, we can directly control the car to move along the right and left planes, y (otherwise parallel parking would be much easier). This means a car has two controllable degrees of freedom, and one uncontrollable degree of freedom.

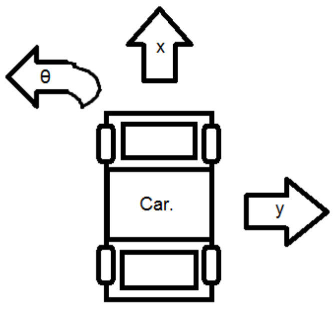


Figure 5: Fantastic image of a car showing the degrees of freedom it has

There are three possibilities of relation between the total degrees of freedom, and controllable degrees of freedom. These are:

1. Holonomic - CDOF = TDOF
   For instance a helicopter, which allows control of all six degrees of freedom
2. Nonholonomic - CDOF < TDOF
   For instance a car, like in the example above
3. Redundant - CDOF > TDOF
   For instance the human arm, which has several degrees that achieve the same goal (there are different ways to move your arm that result in the same position in 3D space).

8 Proportional-Integral-Derivative Controllers (PID)

The "problem" in robotics is that generally we want to do very specific things, and thus, we need to design a system that allows us to do these things. For instance, to actually move our robots, we need to have a Control system that directs enough power to the motors to turn the wheels the correct amount. On a straight, smooth surface, this isn't generally very difficult. But if the robot suddenly begins going up an incline, the amount of power to the motors to travel will be too small, and the robot will not be able to travel up the incline. A control system is used to determine how much the robot should be powered depending on its environment, and how this environment changes. The purpose of the control system is to determine *how* to achieve a specific goal.

An example of a much better control system is the Proportional-Integral-Derivative Controller, shortened to PID. It follows the same sense-think-act cycle that most control systems do; it repeatedly senses the current state, and then reduce the difference between the current state and the goal state, until the current state equals the goal state.

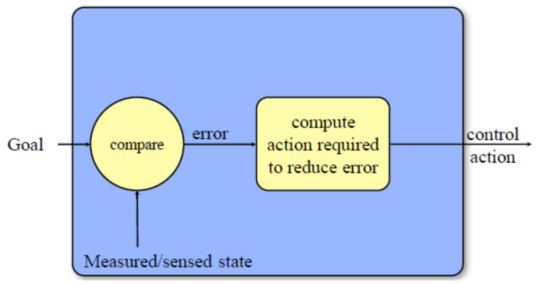


Figure 6: The sense-think-act cycle

When talking about PID, there are three terms that are necessary to be aware of:

1. Present
   The current error, this is the *proportional*. If the current error is very big, you need to make a very big change to what you are currently doing. You want to do something about the problem, depending on the magnitude of the problem at the current time.

2. Past
   The sum of errors up to the present time, this is the *integral*. The integral will, over time, take care of the differences between the desired state and the current state.

3. Future
   The rate of change of the error, this is the *derivative*.

To actual obtain the control action, the PID controller combines these three terms in a weighted sum. Each term has a constant, that is adjusted to change the action of the robot. The formula for the weighting can be seen below:

$$c(t) = K_p e(t) + K_i \int_o^f e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

Some applications, however, do not require the use of all three terms. In these cases, the parameters can be set to zero, in order to remove unused terms. This gives way to a large number of controllers: PI controllers, PD controllers, P controllers, or even I controllers. PI is the most common of these, as the integral term is required in order to remove steady-state error, and the derivative term is often very sensitive to measurement noise. Steady-state error is the error that persists over term, whilst you're in a steady state. This is because, often, there is a very small difference between what you want to achieve, and what you actually achieve.

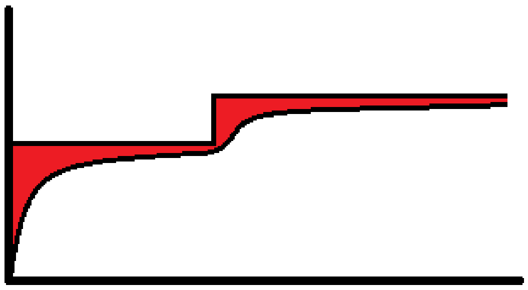


Figure 7: Highlighted area is what is known as the steady-state error

8.1 Proportional Term

$$P_{out}(t) = K_p e(t)$$

A high proportional gain results in a large change in the output for a given error. If the proportional gain is too high, the system can become unstable. If the proportional gain is too low, the system will very slow at responding to change. Proportional term is often the dominant term when calculation the weighted sum of a PID system, but proportional control on its own is not sufficient control. It is difficult to define a good P value, because you both want to achieve the goal state quickly, but you do not want to overshoot too far. The following images show how the editing of the proportional term can have drastic effects on the system.
Figures 16 and 17 are bad for several reasons: firstly, they don't actually follow the desired output very closely, your robot will be fluctuating in movement, and the strain on the electronics can be high, due to the constant shifting from very high to very low output.

8.2 Integral Term

$$I_{out}(t) = K_i \int_0^t e(\tau)d\tau$$

The contribution of the integral term is proportional to both the *magnitude* and *duration* of the error: it looks at the sum of the instantaneous errors over time. The integral term accelerates the process towards the set point (the point at which the actual state and desired state cross on the graph), and helps to eliminate the steady-state error. Unfortunately, it can easily cause overshoot, as the actual value crosses over the set-point and creates an error in the opposite direction. Similarly with the modification of the proportional value, you it is difficult to find a perfect integral value.

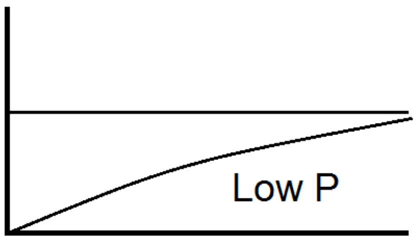


Figure 8: Very low proportional gain, slow reactions

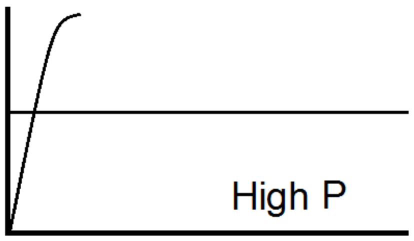


Figure 9: Very high proportional gain, much overshooting

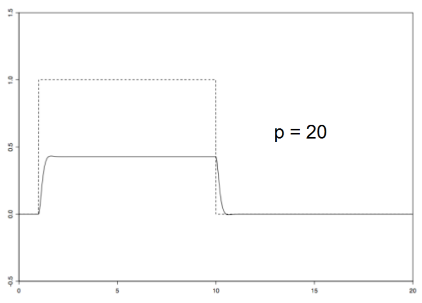


Figure 10: A small P value produces a huge steady state error

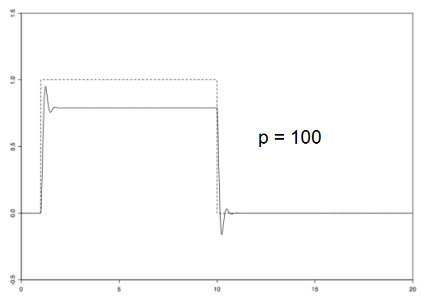


Figure 11: As P increases, oscillations begin to appear, but the steady state error persists

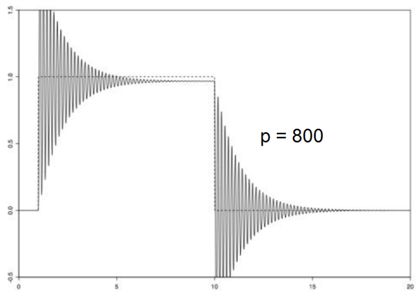


Figure 12: P increases further, the oscillations become dominant

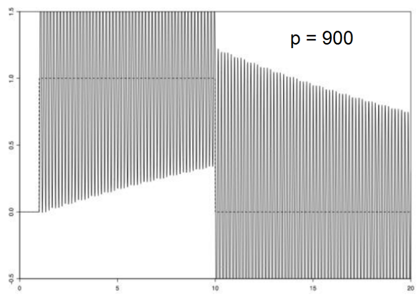


Figure 13: Very large P produce very unstable behaviour

In the figure below you can see how much closer the actual output is to the desired output, now that the integral term has been introduced. You can also see how, if there is too much integral term, much overshooting takes place.

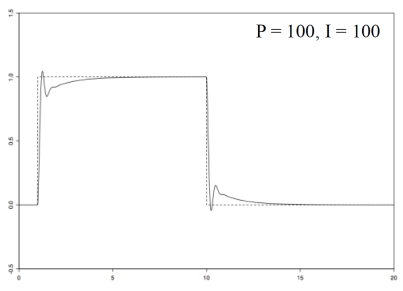


Figure 14: Steady state error is reduced

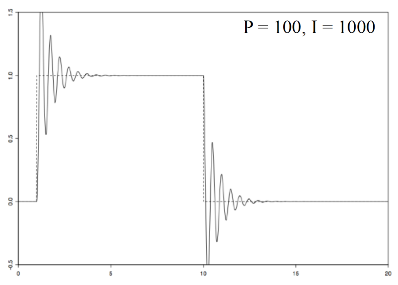


Figure 15: Rise time improves, but overshoots are significant

The one term that really helps you to negate the overshoot, is the derivative term.

8.3 Derivative Term

$$D_{out}(t) = K_d \frac{de(t)}{dt}$$

The contribution is proportional to the rate of change of the error over time. At a given point, it is the first derivative of the instantaneous error. It is used to slow the rate of change of the error, and this effect is most noticeable near to the set point. It helps reduce overshoot caused by the integral term, and can be seen as a “dampening” effect. Unfortunately, differentiation amplifies signal noise and so derivative can cause instability with noise. In the figures below, you can see how the effect of modifying P, I and D terms can increase the quality of control within the system.

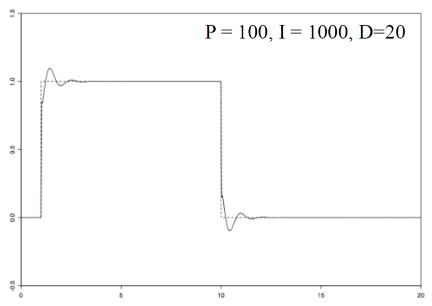


Figure 16: D introducted, oscillations are dampened

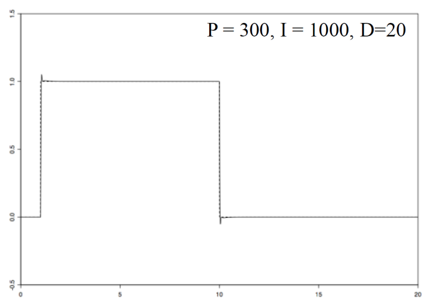


Figure 17: PID tuning, exceedingly good control

8.4 PID Tuning

If PID parameters are chosen incorrectly, the process may be unstable, and its output may diverge (with or without oscillation). The weights must be adjusted to achieve the desired behaviour for a given application, this is known as PID parameter tuning. Naturally, the optimum behaviour is application dependent, as different applications have different conditions (rise-time must be less than specified time, overshoot may not be allowed, minimisation of energy required to reach set point, or even oscillations may not be permitted).
A summary of the tuning of the different parameters in a PID controller can be seen below (the table depicts the effects of increasing the various parameters).

Parameter	**Rise-Time**	**Overshoot**	**Settling Time**	**Steady-State Error**
K_p	Decrease	Increase	Small Change	Decrease
K_i	Decrease	Increase	Increase	Eliminates
K_d	Small Change	Decrease	Decrease	None

8.5 Implementation of PID

The pseudo-code for a PID controller would be something like:

```
previous_error = setpoint − measured_feedback;
integral = 0;
while ( true ) {
        wait ( dt );
        error = setpoint − measured_feedback;
        integral = integral + error ∗ dt;
        derivated = (error − previous_error) / dt;
        previous_error = error;
        output = Kp ∗ error + Ki ∗ integral + Kd ∗ derivative;
}
```

9 Fuzzy Controllers

Fuzzy controllers are another method of controller robots action, that encompass aspects of both cognition, and control. But like PID, fuzzy control systems sense the current state, and then continually attempt to reduce the difference between the current state and the goal state, until they are equal.

Fuzzy control is a methodology built on the framework of fuzzy logic and fuzzy sets. Instead of exact values, we use imprecise values, as they are much more natural for humans to communicate in. In classic logic, we have values that are either TRUE (1) or FALSE (0). In fuzzy logic, however, we have values between true and false, such as "partially" true, and "somewhat" false.

These additional terms are known as "hedges". In classic set theory, we say that an element either belongs to a set or it does not. In fuzzy set theory, an element can be a member to any set, to a differing degree. For instance, in the set below, detailing all real numbers less than twenty, there is a clear cut off point between when this is true, and when it is false, and that point is at exactly 20.

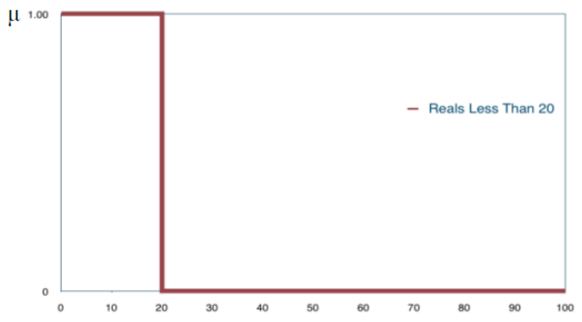


Figure 18: Crisp set membership of real numbers less than 20

However, if we were to specify a value that was more "fuzzy" such as how young some one was, that would be more difficult, and not fit for a crisp set. You cannot determine a single boundary at which a person changes from being young to being not young (imagine being 19 years old, and having your twentieth birthday, and suddenly being considered old). This is why we use a fuzzy system, as this allows us to define membership to a certain set to a certain degree, at all points. The graph below plots young, middle aged and old against each other, and you can see that for any given age, there will be a membership to each of these categories to a certain degree.

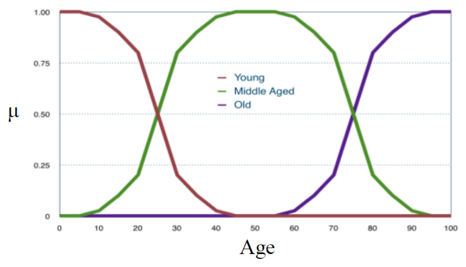


Figure 19: Fuzzy sets showing the relation of young, to middle aged to old

When working with fuzzy sets, we need to define some operators that allow us to manipulate them. For instance, how would we work out the notion of "young *and* old". In fuzzy set theory, we use t-Norms and t-Conorms to define how to operate on the sets. In general, we use the minimum fuzzy sets to determine their *and* value, and we use the maximum of fuzzy sets to determine their *or* value, as shown in the diagram below.

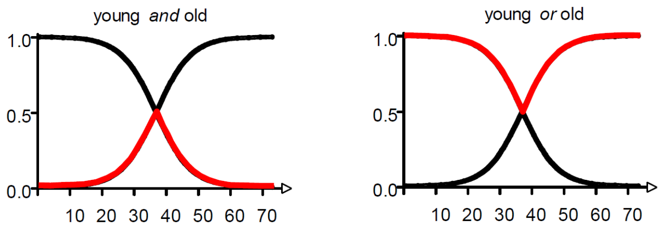


Figure 20: Left, AND operator. Right, OR operator

To actually control the robot, we need to use inputs, outputs and rules of fuzzy system, in order to determine a single action to inact. This process is called fuzzy inference. It takes the crisp inputs, passes them through the fuzzy sets, it then uses the rules we define to map these inputs, returning a fuzzy set, which is then defuzzified into a single crisp output. As shown in the diagram below.

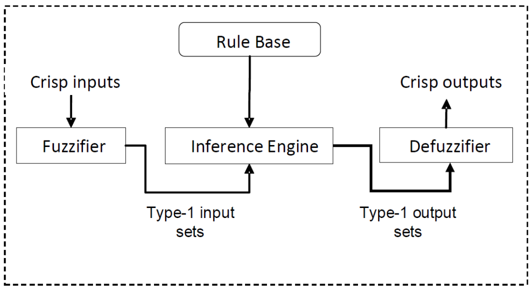


Figure 21: The fuzzy inferencing process

The rules of the system determine how the input functions are manipulated. If you refer to the example, you can see that when two functions are combined with the AND operator, we take the set with the minimum x value. We then only use the set up to this x value, and we combine this with all the other taken sets. As you can see, in the second rule, the x value of a_2 is smaller than that are b_2. In this case we use the values of a_2, but only up to the value of x. We then merge then with the results of the first rule, and we get our output set.

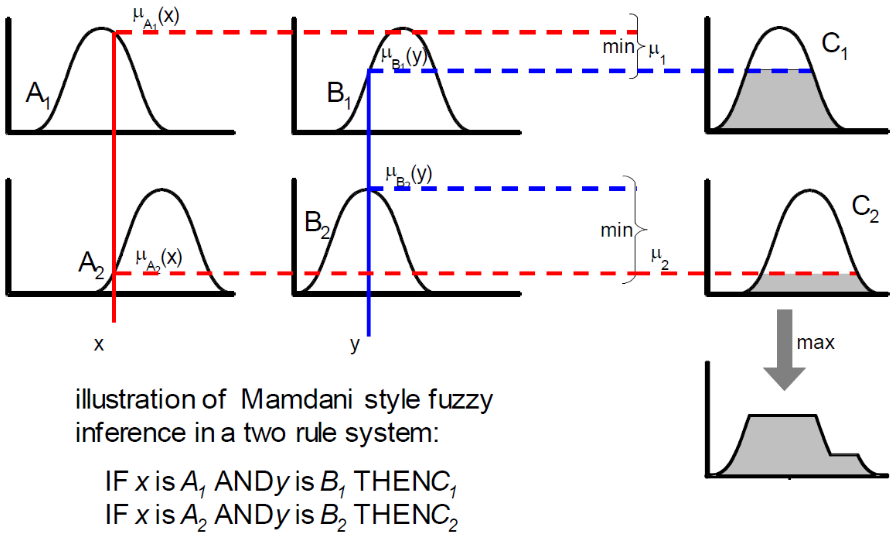

$\mu_{A_1}(x)$
A_1
B_1
$\mu_{B_1}(y)$
min
μ_1
C_1
A_2
$\mu_{A_2}(x)$
B_2
$\mu_{B_2}(y)$
min
μ_2
C_2
x
y
max
illustration of Mamdani style fuzzy inference in a two rule system:
IF x is A_1 AND y is B_1 THEN C_1
IF x is A_2 AND y is B_2 THEN C_2

Figure 22: The fuzzy inferencing process of rules

This final graph must then be "defuzzified", which can be done with a number of different methods, including the mean of the maximum set, or even the centre of gravity.

9.1 Fuzzy Architecture for DC motor control

In a fuzzy implementation of a motor control, we use the error and rate of change of error as our inputs, and will output the control action. The diagram below shows this process, as well as how fuzzy logic is involved.

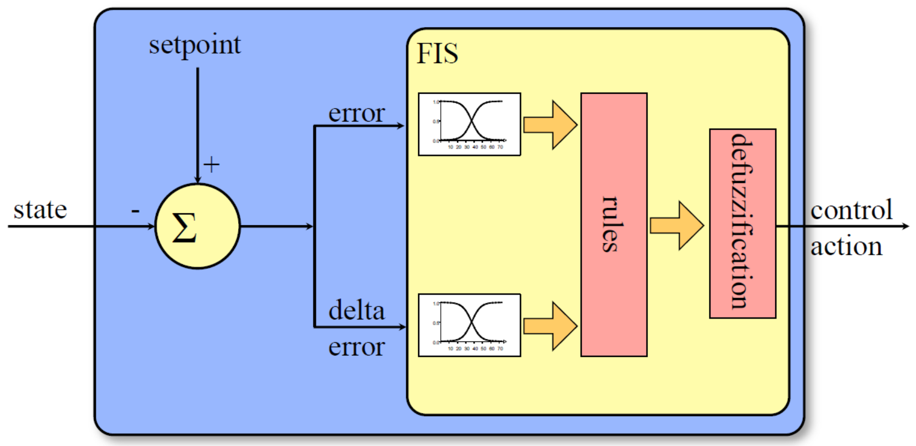


Figure 23: How fuzzy logic is used in a DC motor control system

Each of the inputs has two membership functions, positive and negative, and the output has three membership function, negative, zero and position. These are used to represent whether the current error is positive or negative, whether the rate of change of error is positive or negative, and how the output should be adjusted accordingly, either positively, negatively, or not at all. We then apply our rules to this system, to tells the control what to do with the values it receives. In our example, our rules are:

$$IF\ error\ IS\ negative\ AND\ \Delta error\ IS\ negative\ THEN\ output\ IS\ negative$$

$$IF\ error\ IS\ negative\ AND\ \Delta error\ IS\ positive\ THEN\ output\ IS\ zero$$

$$IF\ error\ IS\ positive\ AND\ \Delta error\ IS\ negative\ THEN\ output\ IS\ zero$$

$$IF\ error\ IS\ positive\ AND\ \Delta error\ IS\ positive\ THEN\ output\ IS\ positive$$

As an example, say our set point is 10rps. If our last reading [t-1] was 6rps, our last error [t-1] was 4rps. If our current reading [t] is 7rps, then our error is 3rps. Our rate of change of error is, therefore, 1rps. In this case, our error is positive, our rate of change or error is positive, and thus our output is positive. This looks something like this:

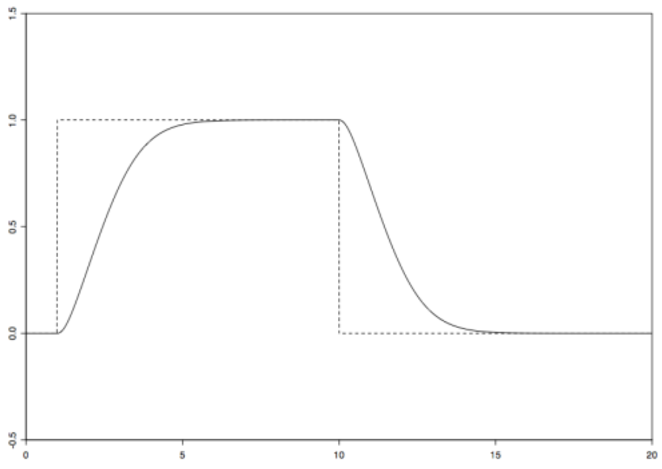


Figure 24: Results of our first fuzzy control system

This is decent, but is not great. The way we can improve upon this, is by increasing the number of membership functions we have, and the number of rules we have to govern these.

9.2 FLC Design Choices

As mentioned above, there are very many design choices in a fuzzy inference system, including:

1. Variables
   Choice and number of input variables
   Number and shape of membership functions in the input and output variables

2. Rules
   Number and the form of the rules

3. Inferencing Method
   Mamdani or TSK inferencing
   Alternative operators for the t-Norm and t-Conorm

4. Defuzzification Methods

10 Localisation

Localisation falls within the "localisation" category of the Autonomous Mobile Robot navigation cycle. This is the part of the robot's thinking process in which it determines its own position within an environment. Several major problems exist at the current time with localisation: the most prominent is that "sensors suck".

The optimal localisation would be some sort of GPS system on the robot that would provide the precise and completely accurate location of the robot. However, this still would not equate for obstacles in the environment. The true optimal localisation set up would be a GPS system on the robot, and on all obstacles in the environment - a set up that is very impractical.

Below you can see the general flow of information and control when designing localisation algorithms. The term "extracted features" means raw data that has had some processing applied to help encapsulate the information more (for instance, looking through raw data to determine a "red" "building").

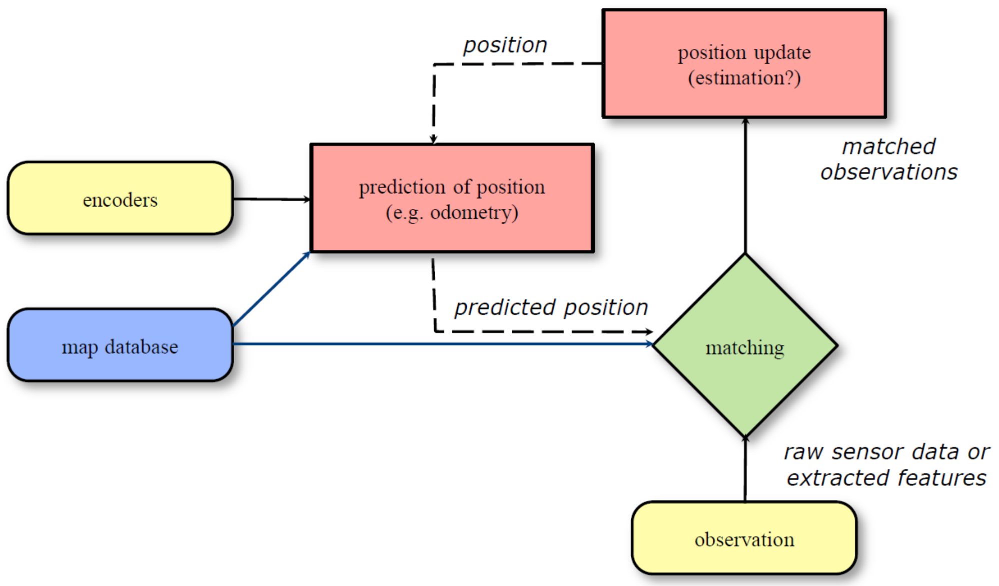

position
position update (estimation?)
matched observations
encoders
prediction of position (e.g. odometry)
map database
predicted position
matching
raw sensor data or extracted features
observation

Figure 25: General Schematic for Localisation

10.1 Localisation noise

10.1.1 Sensor Noise

Sensor noises induces a limitation on the consistency of sensor readings obtained from some given environmental state (as in, there is a limit on the number of useful bits of information). Often, the source of sensor noise is that some environmental features are not captured by the robot's representation, for instance, CCD color cameras detecting hue values are dependent on lighting conditions, and sonar range-measurement are dependent on the angle of incidence and the surface texture.

10.1.2 Effector Noise

Effector noise is also a fundamental physical limitation on localisation. This is the notion that a single action has many different possible results, even if the initial state is fully known. This means that uncertainty is introduced in future states (the robot thinks it has moved some distance, x, but in reality has moved some other distance y).
The diagram below shows how this uncertainty grows exponentially with time. The black lines depict where the robot is expected to go, the red show the potential positions the robot could be in.

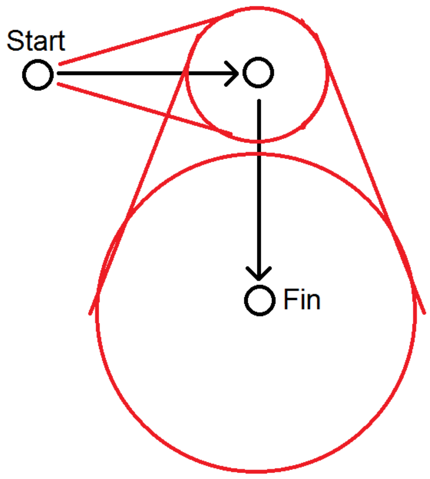


Figure 26: Showing how effector noise can significantly reduce the accuracy of the robot's position

10.2 Sensor Aliasing

Sensor aliasing is the concept of the same information being returned by a sensor for two different local states. This is rarely the case for the human visual system, because every place looks different. The affect can still be achieved if a significant reduction of visuals is encountered (for instance a dark cave), or in other special environments (mazes).

Robot sensors are much less rich that those of humans, and thus aliasing is the norm. For instance, a sonar will only return the distance to a point. If you were to move the robot forward, but still keep the sonar distance to some object the same, the robot would have no way of knowing that it had been moved.

10.3 Odometry and Dead Reckoning

Odometry is the use of the wheel sensors, and Dead Reckoning is the use of additional sensors, in the context of localisation. As robot movement is integrated over time, the positional error accumulates.

Sources of this error can be: limited resolution during integration, misalignment of wheels, uncertainty in wheel diameters, variation in contact point of the wheel or even unequal floor contact. These errors are categorised into two areas: deterministic, which can be eliminated by calibration (if you know one tire is deflated, you know to power that motor more); and non-deterministic, which are random errors.

These errors can be classified into three further types:

1. Range Error (Distance)
   The sum of the wheel motions, which dictates the path length of the robot's movement
2. Turn Error (Angle) The difference of wheel motions, which dictations the amount the robot has turned
3. Drift Error (Displacement)
   The difference in the error of the wheels can lead to an error in the robot's angular orientation (this can basically be equated to the y axis in the degrees of freedom of a car).

Overtime, turn and draft far outweigh range errors. This is intuitively obvious; turning too much, or displacing too much can cause huge differences in expected output and actual output. Whereas an error in the range will only mean you have overshot/undershot travelling forward. The formulae associated with odometry are the following, and they will be given in context in the next section.

$$\Delta\theta = \frac{\Delta S_r - \Delta S_l}{b}$$

Where $\Delta\theta$ is the Variation in angle, ΔS_r is the distance travelled by the right wheel, ΔS_l is the distance travelled by the left wheel, and b is the distance between the two wheels.

$$\Delta S = \frac{\Delta S_r + \Delta S_l}{2}$$

Where ΔS is the Variation in distance travelled, ΔS_r is the distance travelled by the right wheel and ΔS_l is the distance travelled by the left wheel.

$$p' = p + \begin{vmatrix} \Delta S\ cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta S\ sin(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{vmatrix}$$

In this final formula, p' is the new x, y, z matrix depicting the robots position, p is the previous x, y, z matrix depicting the robots position, and all other values come from the two formulae specified above.

10.4 Odometry Example

10.4.1 Perfect Actuation

We will now go through a quick example, to show how the formulae above are applied. Imagine the following scenario: we have a robot with the following specifications:

$$p = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} and\ b = 12cm$$

If the robot travels $50cm$ forward, with perfect actuation (no noise), what will p' equal?

We must start by working out the values of $\Delta\theta$ and ΔS. It is known that ΔS_r and ΔS_l are both 50, because of the perfect actuation.

$$\Delta\theta = \frac{\Delta S_r - \Delta S_l}{b}$$

$= \quad \{ \quad$ substituting known values $\quad \}$

$$\Delta\theta = \frac{50 - 50}{12}$$

$= \quad \{ \quad$ arithmetic $\quad \}$

$$\Delta\theta = \frac{0}{12}$$

$= \quad \{ \quad$ arithmetic $\quad \}$

$$\Delta\theta = 0$$

Now for ΔS:

$$\Delta S = \frac{\Delta S_r + \Delta S_l}{2}$$

$= \quad \{ \quad$ substituting known values $\quad \}$

$$\Delta S = \frac{50 + 50}{2}$$

$= \quad \{ \quad$ arithmetic $\quad \}$

$$\Delta S = \frac{100}{2}$$

$= \quad \{ \quad$ arithmetic $\quad \}$

$$\Delta S = 50$$

Using the values of $\Delta\theta$ and ΔS, we can complete the rest of the equation.

$$p' = p + \begin{vmatrix} \Delta S \; cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta S \; sin(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{vmatrix}$$

$= \quad \{ \quad$ substituting known values $\quad \}$

$$p' = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 50 \; cos(0 + \frac{0}{2}) \\ 50 \; sin(0 + \frac{0}{2}) \\ 0 \end{vmatrix}$$

$= \quad \{ \quad$ arithmetic $\quad \}$

$$p' = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 50 \; cos(0) \\ 50 \; sin(0) \\ 0 \end{vmatrix}$$

$= \quad \{ \quad$ applying sin and cos $\quad \}$

$$p' = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 50 \cdot 1 \\ 50 \cdot 0 \\ 0 \end{vmatrix}$$

$= \quad \{ \quad$ arithmetic $\quad \}$

$$p' = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 50 \\ 0 \\ 0 \end{vmatrix}$$

$= \quad \{ \quad$ matrix addition $\quad \}$

$$p' = \begin{vmatrix} 50 \\ 0 \\ 0 \end{vmatrix}$$

In this case our robot has moved forward in the x plane $50cm$, which is what you would expect.

10.4.2 Environmental Hazards

In that last example, no external factors affecting the travelling of the robot; making calculations very simple. However, in the real world, the chance of a robot being completely unaffected by it's environment is very slim, and this next example will have some external factors affecting the robot.

Imagine the following scenario: we have a robot with the following specifications:

$$p = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} and\ b = 12cm$$

We wish for this robot to travel $50cm$ forward, as before. However, some one has spilt some oil on the track, and the left wheel of the robot slips slightly whilst travelling. This results in the effective distance travelled by the wheels as: $\Delta S_l = 46cm$ and $\Delta S_r = 50cm$. As a result of this, what would p' be? Once again, we would start by determining the values of $\Delta\theta$ and ΔS.

$$\begin{aligned} &\Delta\theta = \frac{\Delta S_r - \Delta S_l}{b} \\ = &\quad \{ \quad \text{substituting known values} \quad \} \\ &\Delta\theta = \frac{50-46}{12} \\ = &\quad \{ \quad \text{arithmetic} \quad \} \\ &\Delta\theta = \frac{4}{12} \\ = &\quad \{ \quad \text{arithmetic} \quad \} \\ &\Delta\theta = 0.333 \end{aligned}$$

$$\begin{aligned} &\Delta S = \frac{\Delta S_r + \Delta S_l}{2} \\ = &\quad \{ \quad \text{substituting known values} \quad \} \\ &\Delta S = \frac{50+46}{2} \\ = &\quad \{ \quad \text{arithmetic} \quad \} \\ &\Delta S = \frac{96}{2} \\ = &\quad \{ \quad \text{arithmetic} \quad \} \\ &\Delta S = 48 \end{aligned}$$

Which allows us to determine p'.

$$\begin{aligned} &p' = p + \begin{vmatrix} \Delta S\ cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta S\ sin(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{vmatrix} \\ = &\quad \{ \quad \text{substituting known values} \quad \} \\ &p' = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 48\ cos(0 + \frac{0.333}{2}) \\ 48\ sin(0 + \frac{0.333}{2}) \\ \frac{1}{3} \end{vmatrix} \\ = &\quad \{ \quad \text{arithmetic} \quad \} \\ &p' = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 48\ cos(\frac{1}{6}) \\ 48\ sin(\frac{1}{6}) \\ \frac{1}{3} \end{vmatrix} \\ = &\quad \{ \quad \text{applying cos and sin, and arithmetic} \quad \} \\ &p' = \begin{vmatrix} 0 \\ 0 \\ 0 \end{vmatrix} + \begin{vmatrix} 47.9998 \\ 0.1386 \\ 0.3333 \end{vmatrix} \end{aligned}$$

$$= \quad \{ \quad \text{Matrix addition} \quad \}$$

$$p' = \begin{vmatrix} 47.9998 \\ 0.1386 \\ 0.3333 \end{vmatrix}$$

Now, what problems does this identify? Our robot thinks it has moved $50cm$ forward, but in reality, it has only moved $48cm$, and to a slight angle.

10.5 To localise or not to localise

Whether or not we should implement localisation is largely problem dependant.For some problems, behaviour based architectures excel, and allowing achieving robust solutions using a very simple set of behaviour. However, behaviours alone can often make it very hard to solve specific problems. For instance, in mobile robotics, robots often must reason about where they are, whether they have been there before, and what is the optimal route to get there again. To answer these questions; mapping is required.

11 Mapping

Maps provide a basic tool of ordering spatial information - such as what the robot has encountered, and where that encounter took place. The map-based concept of the robot's position makes the robot's belief about position visible to the human (we can see where the robot thinks it is, in relation to where it actually is), maps also allow a medium of communication between humans and a robot (a human can tell the robot to go to some specific point on the map). Maps can also be a useful output if they are, say, generally automatically by the robot (imagine mapping the bottom of the ocean, or the surface of another planet).

11.1 Representations of maps

There are several different methods of representing maps, and these are each explored below.

11.1.1 Exact Cell Decomposition

Space is tessellated into areas of free space. Each area is represented by a single node, which means some assumptions must be mode: the actual position of the robot within the area is not important, but the potential to traverse from one area of free space to another is important.

The information for ECD is not always easily available.

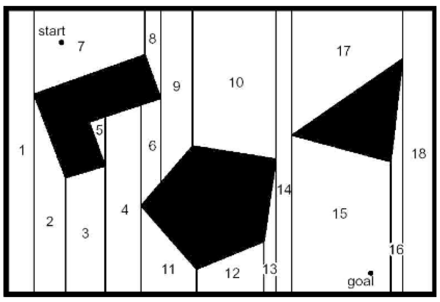


Figure 27: Diagrammatic representation of an Exact-Cell Decomposed map

11.1.2 Fixed Cell Occupancy

In this method, the real world is tessellated - continuous space is transformed into a discrete representation; an artificial grid is imposes on the real world. FCO is highly dependent of cell-size, the smaller the cell, the higher the precision, but the higher the cost. This is demonstrated in the example image: the grid size is so large that some information is lost.

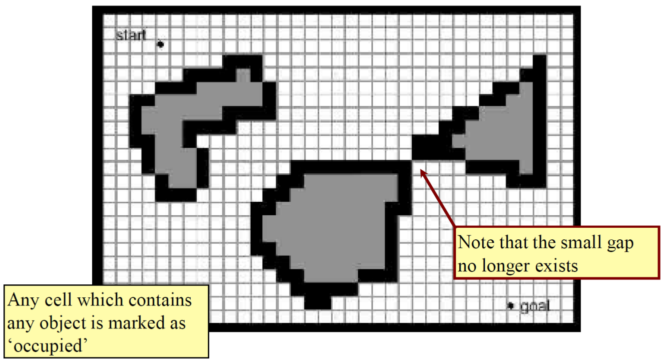

start
goal
Note that the small gap no longer exists
Any cell which contains any object is marked as 'occupied'

Figure 28: Diagrammatic representation of a Fixed-Cell Occupancy map

11.1.3 Continuous Polygons

Generally employed when an environment is simple and well known, however it less applicable for mapping of the real world.

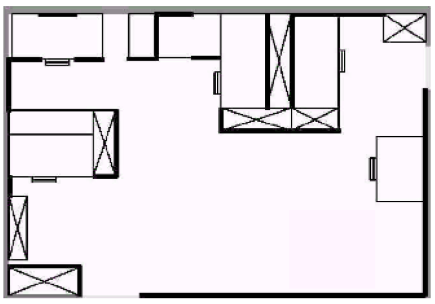


Figure 29: Diagrammatic representation of a Continuous Polygon Map

11.1.4 Continuous Line-Based

Abstraction of the real world based on continuous lines, which can be generated on the fly using a laser range finder. Very good for indorr environments, because human built environments are very linear in design.

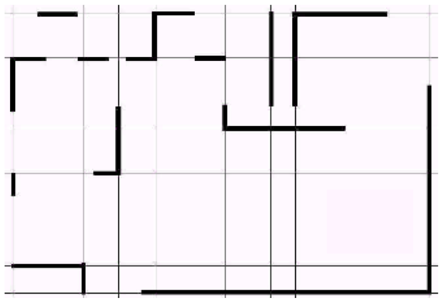


Figure 30: Diagrammatic representation of a Continuous Based Map

11.1.5 Topological Maps

A topological map is an abstraction of a traditional map. Instead of accurately depicting the location of places on the map, it only depicts the route between places; it is only concern with nodes and their connectivity. These maps rely on the robot to employ any of its sensors to recognise and exploit the connectivity between the nodes, and be physically capable of doing so. However, as topological maps are not necessarily based on the geometry of space, the potential for them to describe the robot's precise position is very limited.

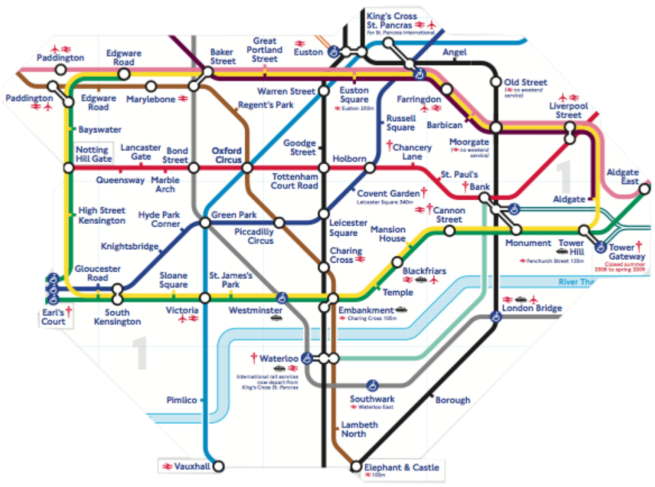


Figure 31: The London Underground map, a perfect example of a Topological Map

11.2 Map Representation

The method for representing the map is closely linked to the method used of the belief representation (where the robot thinks it is on the map). The precision of the map must appropriately match what the robot needs in order to achieve its goals, it must also have the necessary features and precision for the robots sensors to be used.

The complexity of the map also has a direction impact on the computational complexity of reasoning about mapping, localisation and navigation.

11.3 Belief Representation

Belief representation is where the robots thinks it is, which we as humans can then compare against where the robot actually is, to judge the performance of the robot, its localisation, and its mapping. There are two main categories of belief:

1. Single-Hypothesis Belief The robot postulates its unique position, and only keeps a track of the single position it believes itself to be in. If the robot is incorrect about this, it can get lost later. This uses Kalman Filter Localisation.

2. Multiple-Hypothesis Belief The robot describes the degree to which it is uncertain about its position, by postulating many possible alternatives, and ranking like in likelihood. This is much more computationally expensive, but can be much more accurate. This uses particle filter localisation.

The different belief representations can affect the map representations; which can either be continuous or discreet, as shown below.

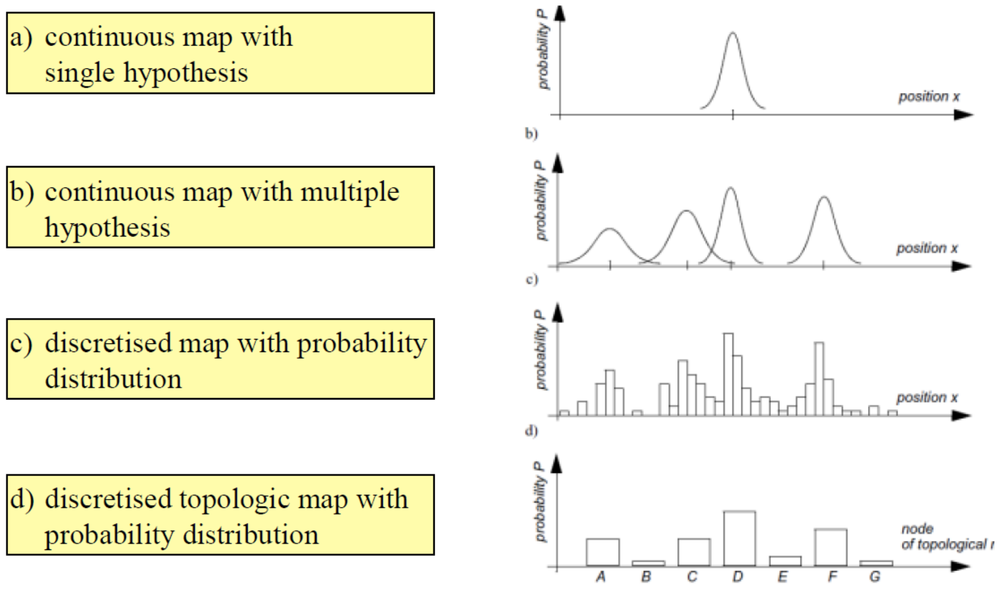


Figure 32: Multiple examples of representations of maps

Single vs Multiple Hypothesis

1. Single
   Typically reasonable in processing power, and good for simple decision making. However, robot is lost when diverging from hypothesized state.

2. Multiple
   Robot can maintain and reason about the level of uncertainty of its position. Uncertainty resulting from incomplete sensor information can be modelled through the updated of the multiple states. However, this is a great challenge for decision making, and can potentially be computationally expensive.

12 Kalman Filter

Kalman filter features under the perception and localisation segments of the navigation processing of the robot. It deals with determining the robots position, but also interpreting raw sensor data to extract useful information. The filter itself is a recursive system that estimates the state of a dynamic system, from a series of noisy and possibly incomplete observations, developed by Rudolf Kalman in 1960. It is used to provide accurate, regularly updated information about the position of an object.

Figure 33 shows three curves. The two dotted curves show two raw sensors estimations of the position of the robot, one of them being somewhat precise (the taller of the two), and the other being quite vague. The bold line is the Kalman Filtered position of the robot. As you can see, this takes weighted information from all of the readings, and uses this information to plot a more accurate position.

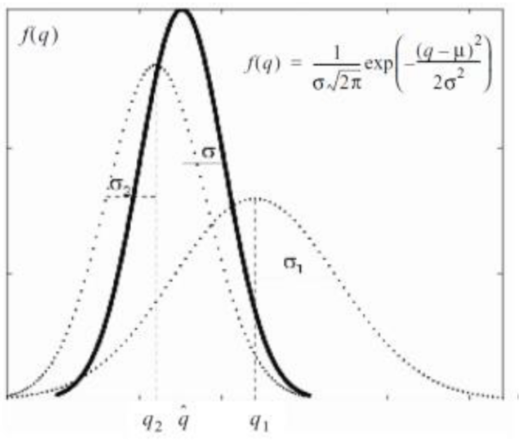


Figure 33: Three curves plotted: the bold being the Kalman filter fusion of the dotted two

The Kalman filter is a optimal sensor fusion technique, which means the problem itself needs to be expressed as a sensor fusion problem. In this context, we use "fusion" to mean the combination of two or more separate readings, that are combined to produce a weighted average of these.

12.1 Brief Overview

This section covers the Kalman filtering process in a small amount of detail, to be expanded upon later.

1. Predict the robot's current position
   In this step, we use the last recorded position of the robot, and the actions that the robot have taken to predict where the robot is. For instance, if the robot is at (0,0), and it travels forward for 50 units, we predict that the position will be (50,0).

2. Obtain sensor measurements
   We use the sensors on the robot to determine the actual position of the robot in the map.

3. Predict sensor measurements
   Using the predicted position, we tell the robot what we think the last step should have observed.

4. Match the actual sensor measurements to the predicted measurements
   We compare the actual sensor readings and the predicted sensor readings. This is then used to determine how accurate our predictions were. If the robot actual readings are nothing like the predicted ones, we have made a mistake and the robot is not where we think it should be. The readings that are closest to the actually recorded readings are known as the "validated" readings, and the other readings are discarded.

5. Estimate robot position
   Using the validated readings, we estimate the robots position on the map.

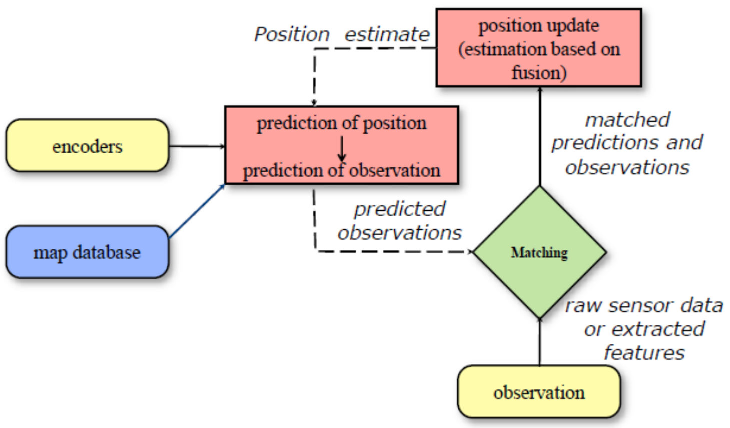

Position estimate
position update (estimation based on fusion)
encoders
prediction of position
prediction of observation
matched predictions and observations
map database
predicted observations
Matching
raw sensor data or extracted features
observation

Figure 34: General Schematic for Kalman Filter Localisation

12.2 Formal Definition

System Model
We begin with some state, x, such that $x \in \mathbb{R}^n$. We also have some discrete linear processes

$$x_k = A_k x_{k-1} + B_k u_k + W_k$$

Where x_k is your state, A_k and B_k relate the previous state to the current state (A in terms of position and B in terms of control signal), x_{k-1} was the previous state, u_k is the control signal applied, and w_k is the signal noise, which is normally distributed like so: $w_k \sim N(0, Q_k)$ (where 0 is the mean, and Q_k is the process noise covariance matrix)

Measurements
We have some measurement, z, such that $z \in \mathbb{R}^m$. The measurement for a given state is defined as:

$$z_k = H_k x_k + v_k$$

Where z_k is the feature you are calculating, H_k is the transformation to the sensor frame (as in, if your sonar sensor is 5cm behind the front of the robot, we should add 5cm to the value read in), x_k is the measurement taken in this instance, and v_k is the measurement noise (which is normally distributed, as per w_k in the last formula).

Predictions
Using the previous state and the control signal, we can predict the state that we believe we will be in. This is done with the following formula:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

In which, $\hat{x}_k^-$ represents a matrix of the prior estimate (ie, what we assumed before any correction are made), and all other variables are as described in the system model.
We also predict the error covariance that we expect to encounter ahead. This is a measure of the estimated accuracy of the estimated state, as is defined:

$$P_k^- = AP_{k-1}A^T + Q$$

Where, P_k^- is the prior error covariance (ie, before correction), A is the relation from the previous states, P_{k-1} is the previous error covariance, A^T is the transposed matrix and Q is noise.

Corrections
Firstly, we calculate the "Kalman gain"; which tells us how much to adjust our state, given the difference between our measurements, and our predicted measurements. It is defined as:

$$K_k = P_k^- H^T (H P_k^- H^T + R)^{-1}$$

In which, K_k is the Kalman gain of state k, P_k^- is the prior error covariance, H^T is the transposed transformation of the sensor frame, H is the transformation of the sensor frame, and R is the measurement noise. You can see that, is the measurement noise is large, the Kalman gain is small. This means that, if the measurements are noisy, they are less important, and thus do not skew the results as much.

After working out the Kalman gain, we use it to update our estimates with measurement z_k. Using the formula:

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

The $(z_k - H\hat{x}_k^-)$ segment is the difference between the actual measurement and the predicted measurement, as this is known as the *innovation* or *residual.*

After this, we update the error covariance:

$$P_k = (I - K_k H) P_k^-$$

Where I is an identity matrix of size n x n. (Note that, $\hat{x}_k$ and P_k are posterior values).

12.3 Filter Parameters and Tuning

A few things that are worth noting about the parameters of a Kalman filtering system:

1. The measurement noise covariance, R
   This depicts the size of the error in the sensors, and can be experimentally determined prior to operation
2. The process noise covariance, Q
   May also be determined experimentally, but it is often more difficult to directly observe the process. Q can sometimes be tuned off-line, through another Kalman filter performing system identification.
3. The error covariance, P
   If the starting conditions are known exactly, this is initialised with zeros, otherwise a large diagonal.

12.4 Optimality

For the Kalman filtering process to be optimal there are a few assumptions that have to be made: the process is linear, the process noise is a Gaussian with a zero mean, the observation model is linear, the observation noise is a Gaussian with a zero mean, and the process noise and observation noise are independent. Under its specific assumptions, however, the Kalman filter method is guaranteed to be optimal.

13 Particle Filtering

Just like the Kalman filter, particle filtering is entrusted with the tasks of perception, extracting useful information from the sensors, and localisation, determining the robots position in the environment. Particle Filtering is defined as follows:

A particle filter is a sample-based filter that sequentially estimates the state of a dynamic system from a series of noisy observations, and was introduced by Gordon, Salmond and Smith in 1993. Particle filtering is similar to Kalman filter in that it combines a prediction phase with an update phase, and operates sequentially over discrete time-steps. However, it differs in that it make no assumptions of process/measurement linearity, or noise character, and it maintains a population of estimates.

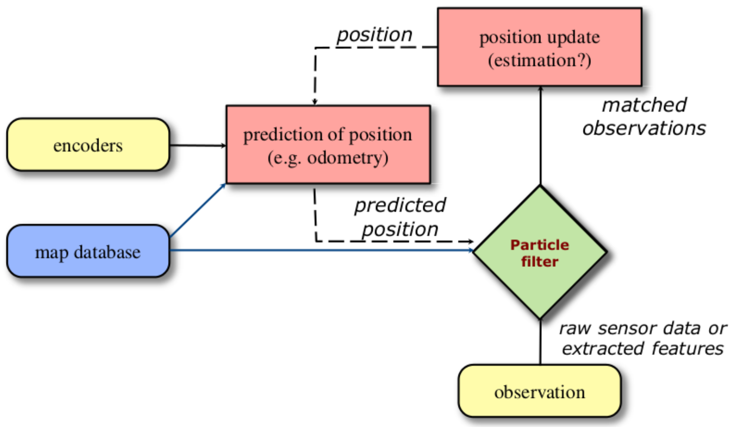

position
position update (estimation?)
matched observations
encoders
prediction of position (e.g. odometry)
predicted position
map database
Particle filter
raw sensor data or extracted features
observation

Figure 35: General Schematic for Particle Filter Localisation

In most real world cases, the assumptions of the Kalman filter are not met, but if they are, we should always use Kalman filtering. If the assumptions are close enough, we should also use Kalman filtering, and if the assumptions are violated, but only by a little bit, we should use the Extended Kalman Filter. If the assumptions are grossly violated, we use the particle filter instead. The reason for this is that the particle filter has a severe computational complexity, and the Kalman filter is better in most cases.

13.1 The System Model

Given some state, x, such that $x \in \mathbb{R}^n$, our position is determined by a linear process $x_k = f(x_{k-1}, u_k, w_k)$ which is a function of our previous state, the control signal and the process noise. In particle filtering, unlike Kalman filtering, we have a *non-linear* measurement model, defined $z_k = h(x_k, v_k)$, which is a function that takes the reading of our position, and the measurement noise.

Particle filtering is a multiple belief representation, and the key idea is that you have a large number of particles, each representing a belief. Think of it as a number of virtual robots and, as you move along, you pick whichever of these “makes the most sense” - for which one does its predicted position correspond best to what the robot can actually see in the environment.

Each particle, s^i, consists of a state (the robots possible position), x^i, and an associated weight (how likely it is that this position is true), ω^i, at each time k. This is represented by:

$$s_k^i = [x_k^i, \quad \omega_k^i]$$

And there is a set, S, of N particles. Obviously, the more elements in this set (and thus the more particles), the more potential positions you are evaluating at the same time - which will give you a better performance. However, the drawback of this is that the system overall with run slowly, as it is very computationally expensive.

13.2 The Algorithm

```
for k = 1 to t
    for each particle // predict particles
        calculate new state, based on process model
    end for

    for each particle // update weights
        calculate new weight, based on measurement model
        normalise weights (to sum to unity)
    end for

    calculate overall predicted position // determine position

    if particles are insufficiently diverse
        re-sample particles // re-sampling
    end if
end for
```

The two diagrams below show a different spread of particles. In the first (figure 36), we have a very diverse set of particles and we can thus calculate a diverse spread of possible positions, which is good. The second (figure 37), however, shows many particles close together, this is knows the particle convergence, and is very bad, because you lose the utility of particle filtering, because, at this point, it is basically the same as doing a Kalman filter. In the second diagram, we would do re-sampling, to re-diversify the particles. (Note, the red robots are representations of the particles, the blue squares are obstacles on the track).

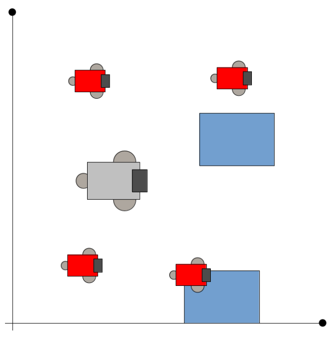


Figure 36: Diverse Spread of particles

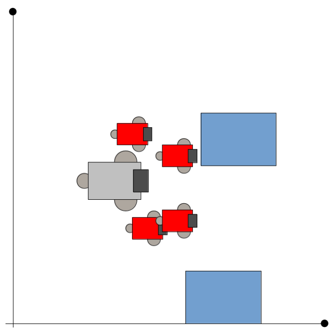


Figure 37: Converged set of particles

13.3 Predicting Particles

As your robot moves, you know (via the control signal) where it *should* be going. To represent this, you move each of your particles the same way (see figure 38). So if you move forward one unit, all of your particles are moved forward one unit, as shown in the diagram. To do this, we use the formula: $x_k^i = f(x_{k-1}^i, u_k, w_k^i)$.
It is important that we include the process noise (w_k) in our calculations, so that we account for external factors such as a bent axle, a wheel slipping, or even the power being drained. Our process noise, is a sample drawn from the known process noise distribution, which unlike in Kalman filter, need not necessarily be Gaussian. However, if the process noise distribution, Gaussian will be used.

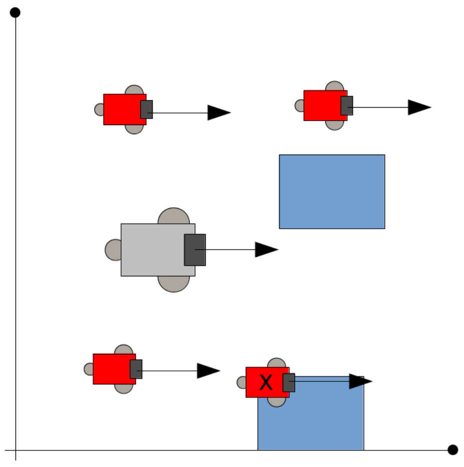


Figure 38: Prediction of particles

13.4 Updating Weights

We assume that each particle is the actual position, and then update the weight of these particles. Which is done by estimating the probably of the observed measurement. This is done with the formula: $\omega_k^i = \omega_k^i P(z_k \mid x_k^i)$. Note that the $\mid$ syntax means "such that" (in this case, "$\mid x_k^i$" simply assumes this particle is the actual position). From here, we calculate the residual (which you should recall from Kalman filtering is the difference between the predicted measurement, and the actual measurement), using the formula $z_k - h(x_k^i, 0)$. We then calculate the probability by considering the distribution of the measurement noise, if this is unknown, we assume a Gaussian distribution.

It is at this stage we also look for improbable states, such as particles that would have moved us to be inside of an obstacle, as this is obviously not possible. For example, robot x, in figure 38.

13.5 Determining Position

After calculating the weights, we determine the actual position of the robot. Which can be achieved in a multitude of ways.

1. Weighted Mean

$$\hat{x}_k = \sum_{i=1}^{N} \omega_k^i x_k^i$$

2. Best Particle
   Use the particle that has the highest probability of being correct (generally used when one particle is close to 1, and all others are much smaller).

$$\hat{x}_k = x_k^i \mid \omega_k^i = max\ \omega_k^i$$

3. Robust Mean
   Only use particles that are within a given distance of the best particle, and compute the mean of these.

$$\hat{x}_k = \sum_{i=1}^{N} \omega_k^i x_k^i :\mid x_k^i - x_k^{best} \mid \leq \varepsilon$$

13.6 Re-sampling

Generally at this point, we would be finished with the filtering, and would repeat the process. However, one problem that can occur is that all of the particles could converge on one place. At this point, we must go through the process of re-sampling. To determine when we should re-sample, we calculate the effective sample size (ESS, or N_{eff}), using the formula:

$$N_{eff} = \frac{1}{\sum\limits_{i=1}^{N} (\omega_k^i)^2}$$

Which is 1 over the sum of all weights squared. When your effective sample size falls below a given threshold (often $\frac{N}{5}$), then re-sampling should be performed. At this point we pick a random new populate, in proportion to the weights of the previous population.

13.6.1 (Systematic) Re-Sampling Algorithm

```
$S' = \emptyset$
$\Delta = \text{rand}\ (0,\ \text{N}^{-1})$
$c = \omega_k^1$
$i = 1$
**for** $j = 0 \rightarrow \text{N}-1$
    $u = \Delta + \frac{j}{N}$
    **while** $u > c$
        $i = i + 1$
        $c = c + \omega_k^i$
    **end while**
    $S' = S' \cup \{(x_k^i, \frac{1}{N})\}$
**end for**
```

13.6.2 Implementation of Particle Filter

As mentioned before, the more particles that are present, the better the approximation that can be made. However, the more particles present, the more computation that is required. Particle filter approach the Bayesian optimal estimate, but only if the number of particles is sufficiently large.

There is also a large number of variations of the particle filtering method, which are listed below (although the details of these are not overly important):

1. Bootstrap Particle Filter
2. Auxiliary Sampling Importance Re-Sampling
3. Regularised Particle Filter
4. Local Linearisation Particle Filter
5. Multiple-Model Particle Filter

13.7 Properties of the Particle Filter

While it's not entirely accurate to state that particle filters have no underlying assumptions, they *can* be used is very many situations. However, particle states must be predicted, noises must be added to the process and an estimate of measurement errors is required. However, particle filters are far more computationally expensive the Kalman filters (and even the extended Kalman filters). If the Kalman filter assumptions are valid, then they are better.

A Brook's Assumptions Mnemonic

Certain **B**ooks, **s**uch as **M**ajora's **M**ask **3D**, **r**eally **m**ade **n**aughty **a**nal **e**lves **v**isibly **d**istressed. **S**enses **f**ail **s**hould **s**top.

This gives us the letters: CB S MM 3D RM NAE VD SF SS, which then convert into brook's assumptions like so:

CB : Complex Behaviours

S : Simple

MM : Map Making

3D : 3D

RM : Relational maps

NAE : No artificial environments

VDD : Visual Data

SF : Sensors Fail

SS : Self Sustaining

Which converts to:

Complex Behaviours are not necessarily the result of a complex control system

Things should be **Simple**

Map Making is crucial

The real world is **3D**, and should be modelled so

Relational maps are best

No artificial environments will be built for the robot

Visual Data is better than sonar; just because it is easy to collect, does not mean it gives a rich understanding of the environment

The system should continue to function even after some **Sensors Fail**

The system should be autonomous and **Self Sustaining**