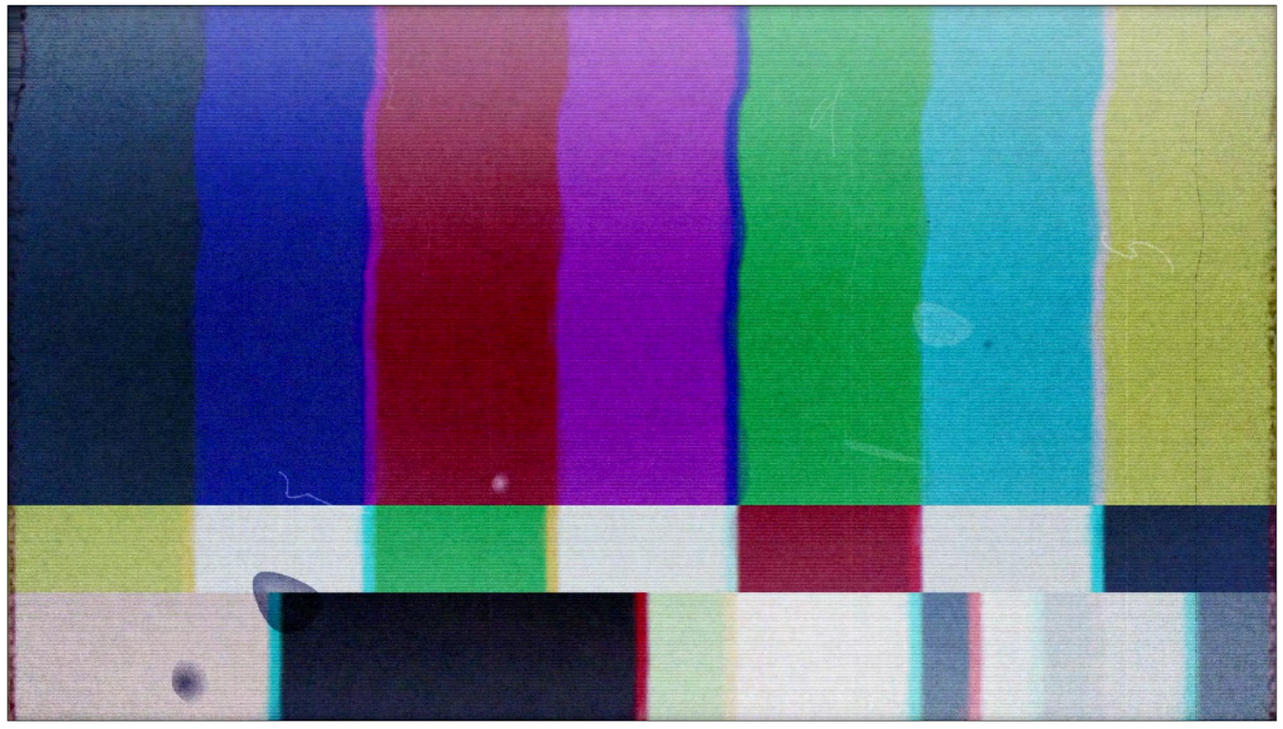
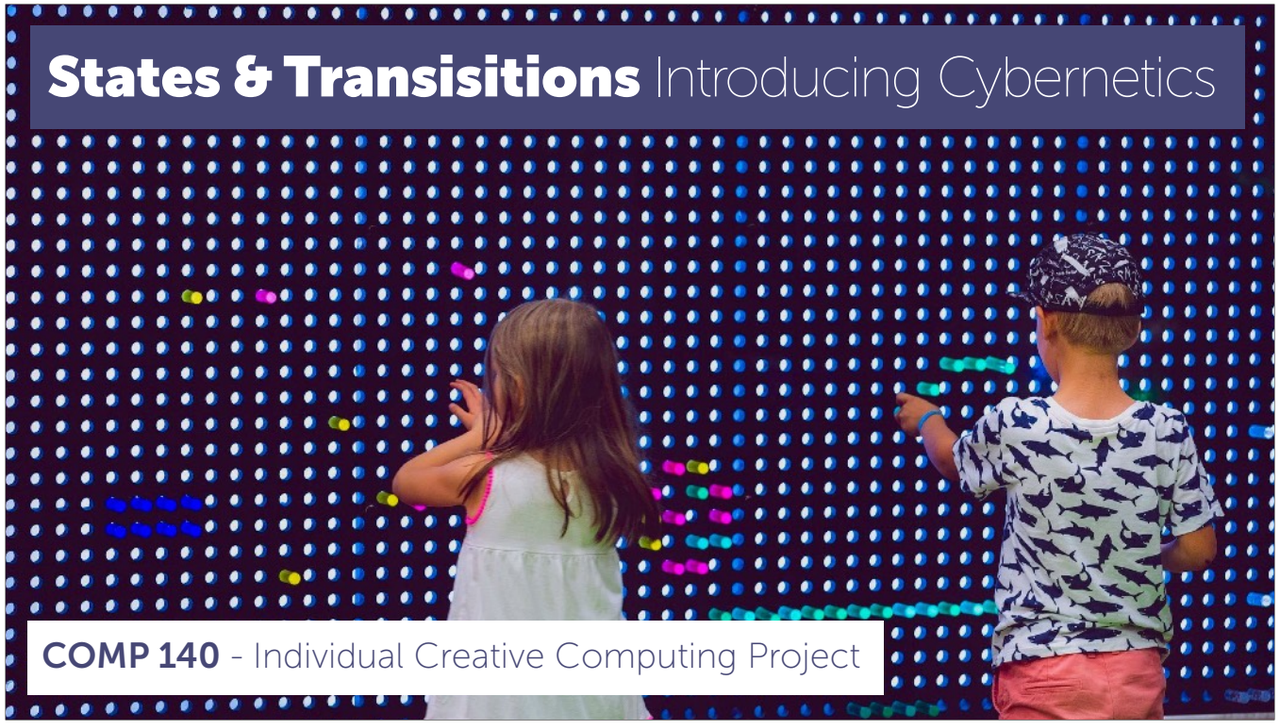
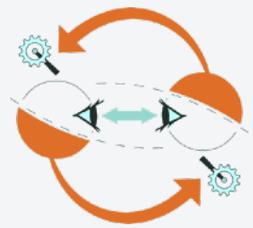


States & Transisitions Introducing Cybernetics



COMP 140 - Individual Creative Computing Project

Learning Outcomes



- Outline the meaning and application of **cybernetics**
- Explain and apply **transformations** between states using **Kinematics**
- Define the role of **transistors** as simple neurone in electronic control systems
- Identify uses of **signal processing** in embedded systems
- Apply cybernetics to electronics projects in the form of **finite state machines**

In this lecture I plan to outline some key theories but also their application in the field of Cybernetics. Cybernetics is commonly applied to robotics but its applications are relevant to all areas of computing and design.
This can be useful in thinking about both the physical and virtual components of your system but also the human interface with the experience you intend to develop

What is cybernetics?

"Cybernetics is the study of human/machine interaction guided by the principle that numerous different types of systems can be studied according to principles of feedback, control, and communications. The field has a quantitative component, inherited from feedback control and information theory, but is primarily a qualitative, analytical tool – one might even say a philosophy of technology."

David A Mindell - MIT

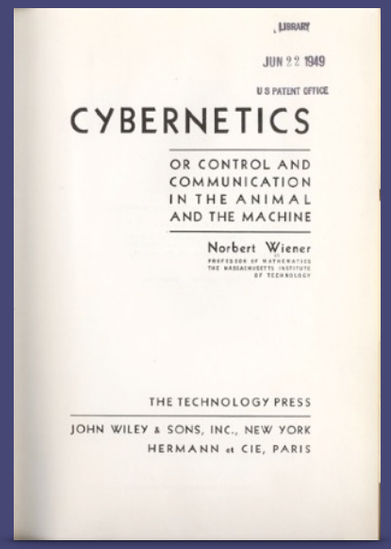
Photo by Will on Unsplash

History of Cybernetics

Cybernetics comes from the Greek word - **kybernetes** which means '*steersman*' and relates to the principle of controlling or directing of a system. The ancient Greeks dominance of the classical world was in part due to their mastery of machines. Specifically fast fighting ships, packed with Spartan warriors. Consequently the **kybernetes** who steered them throughout the Mediterranean sea had a very important role to play.

Cybernetics

or Control and Communication
in the Animal and the Machine



The seminal text on the subject is: READ
By Norbert Weiner who is considered the father of cybernetics

Learning from Nature

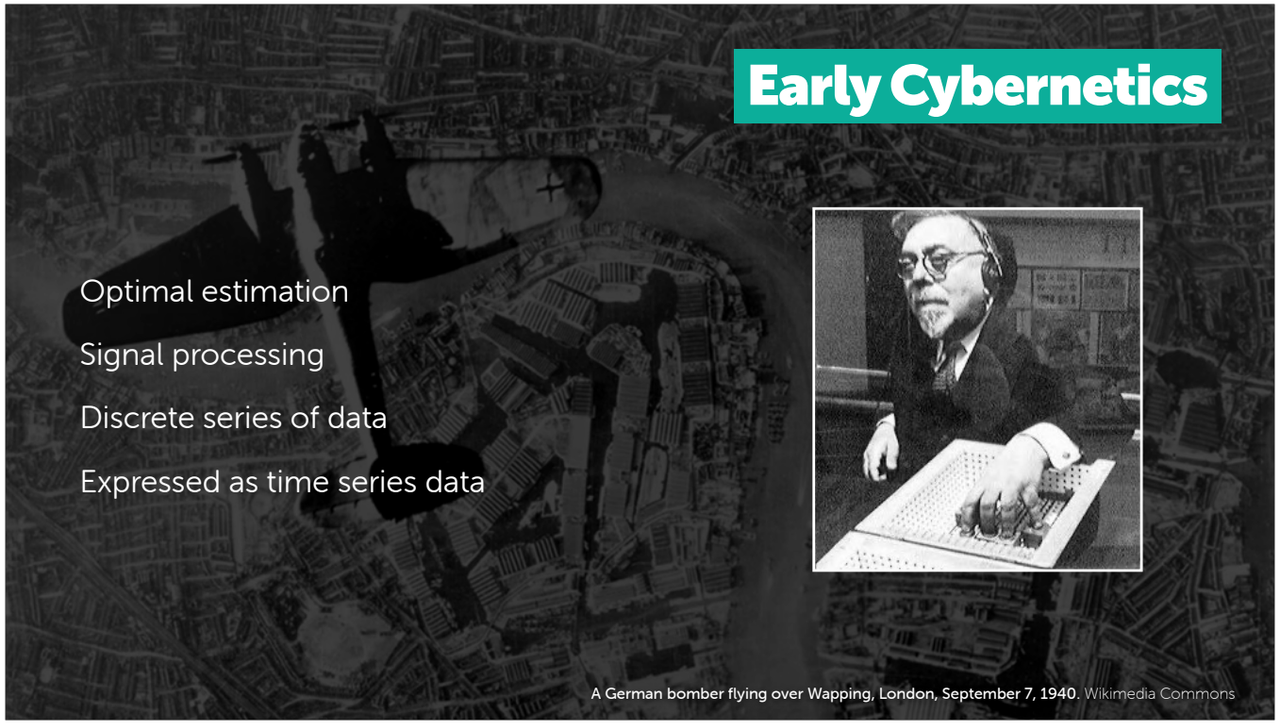
Lemur motion in Madagascar

'Salto' Jumping Robot | USC Berkeley

"Control and communication in the animal and the machine" - Norbert Wiener

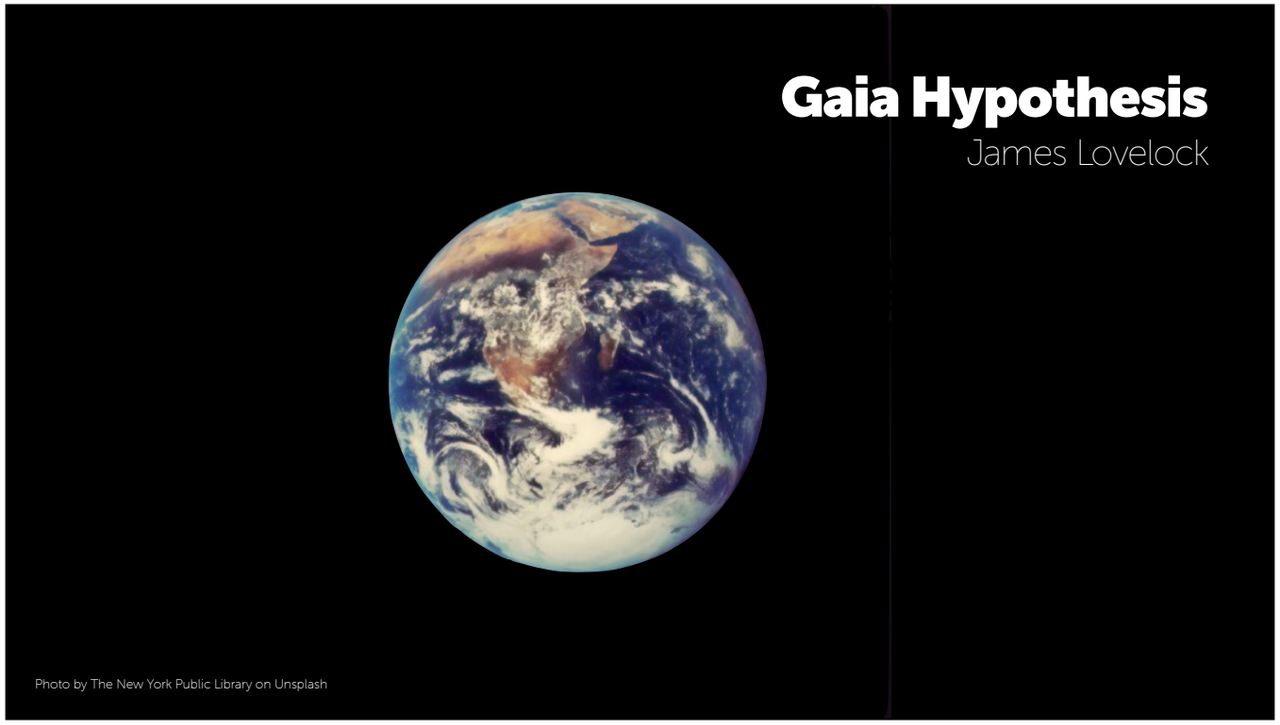
Wiener described the field like this: CLICK READ

Cybernetics is also about learning processes from animals and plants and using robotic simulations to gain a better understanding of complex natural systems.
We are used to associating robots with the imitation of humanoid behaviour but sometimes a simpler form of perambulation like, that we see in lemurs (CLICK) overcomes some of the complexity of attempting to move through an environment. If a robot can constantly jump it resolves the need to have complex feet or wheels, springs and differentials to adapt to changing terrain. When it hits any surface it jumps, if it is suitably well balanced it can stay mobile. This example 'Salto' CLICK by USC Berkeley's robotics lab shows how study of an animal jumping process can help to develop better robotic locomotion and navigation. We will return to this idea of mimicry later, but first let's address some fundamentals



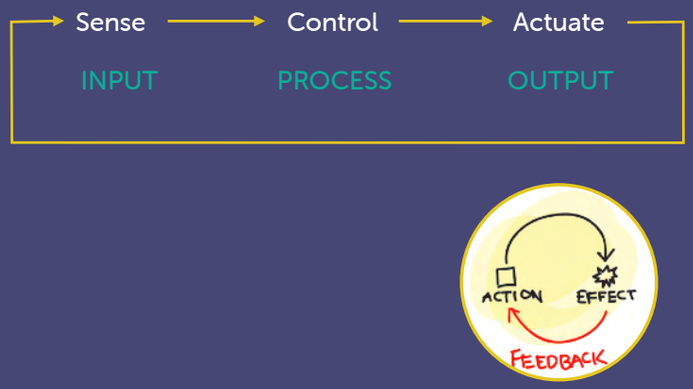
A German bomber flying over Wapping, London, September 7, 1940. Wikimedia Commons

In its early years in the 1920s and 30s cybernetics was concerned with the problem of tracking targets (optically and then with radar), predicting their future positions, calculating ballistics, and directing guns to fire to destroy the targets. CLICK Norbert Wiener, a brilliant but eccentric MIT mathematician, already had a successful career in which he made numerous contributions to mathematics particularly to fields like harmonic (Fourier) analysis and stochastic processes. However Wiener's work proved to have little application to wartime problems (it generated ponderous, complex solutions) but it did lead him to produce an important paper that paved the way for the modern theories of optimal estimation and signal processing CLICK. He created a general theory of smoothing and predicting any problem expressed as a discrete series of data CLICK. This generalization, from a specific human/machine problem to any aspect of the world that can be expressed as time-series data CLICK, presented an early glimpse of the strategies that would define cybernetics.

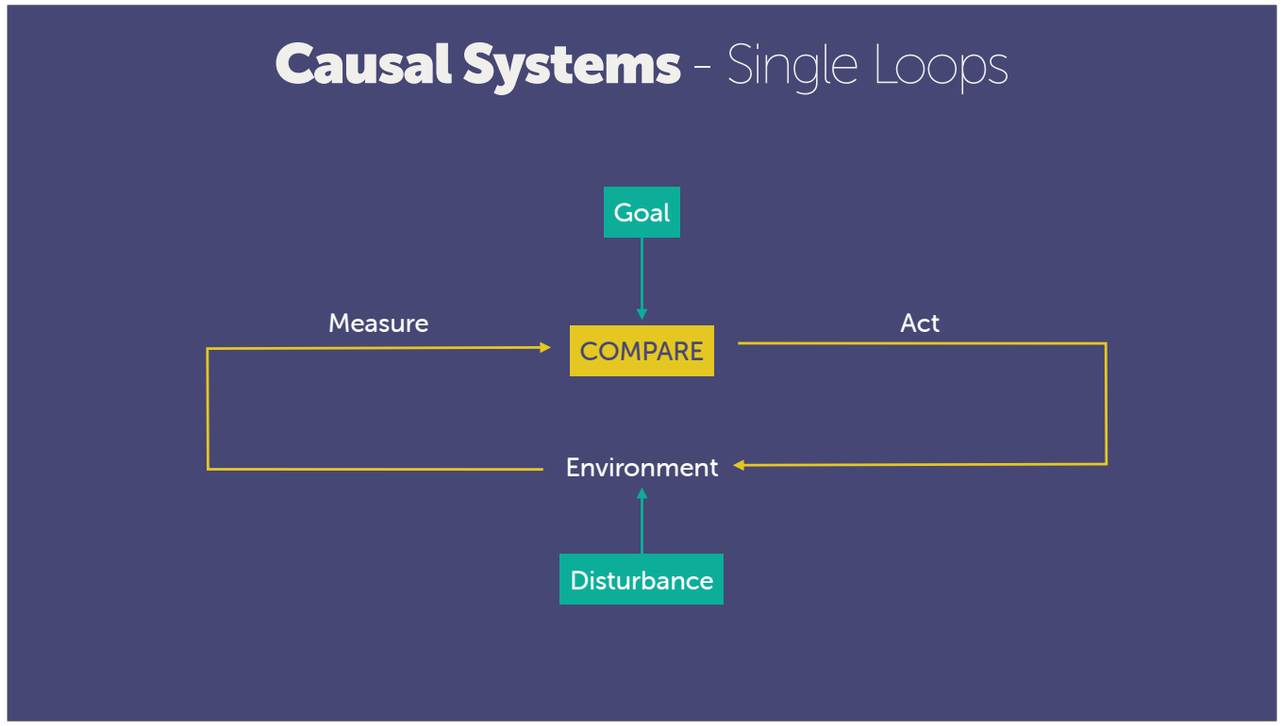


Weiner’s work on measuring natural systems with time series data lead the scientist James Lovelock to propose The Gaia hypothesis which posits that the Earth is a self-regulating complex system involving the biosphere, the atmosphere, the hydrospheres and the pedosphere, tightly coupled as an evolving system. The hypothesis contends that this system as a whole, called Gaia, seeks a physical and chemical environment optimal for contemporary life. Cybernetics can be used to measure all life as we know it.

The Cybernetics Loop

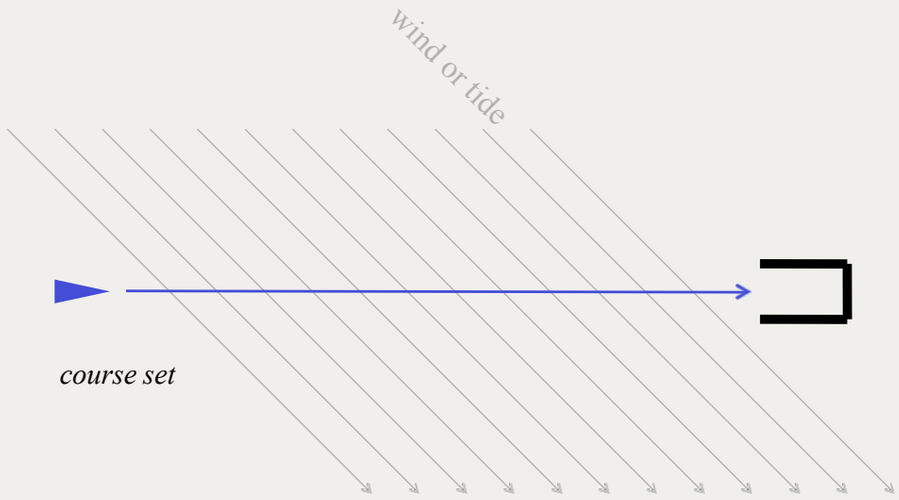


CLICK Sensing, Controlling, Actuating. Any computational system that is effected by the real world must have a way to sense it in some way CLICK, maybe it's a button, a camera, a microphone, or a light sensor. The data from the sensor is then measured and processed by the controller this is usually a computer but it can be simpler (as we well address later) and depending on the how the data is interpreted an actuator is started. CLICK This could be a motor turning a wheel, a pump or even moving a virtual object in a game or storing data point in a database. This process is a continuous feedback loop, CLICK The actuation may well effect the state of the sensor data, and so the process returns to the start to begin all over again. CLICK Cybernetics is also highly relevant to thinking about control scheme's and player feedback in games.CLICK In games design one of the fundamental principles is the game feedback loop which is derived from Cybernetic principles.

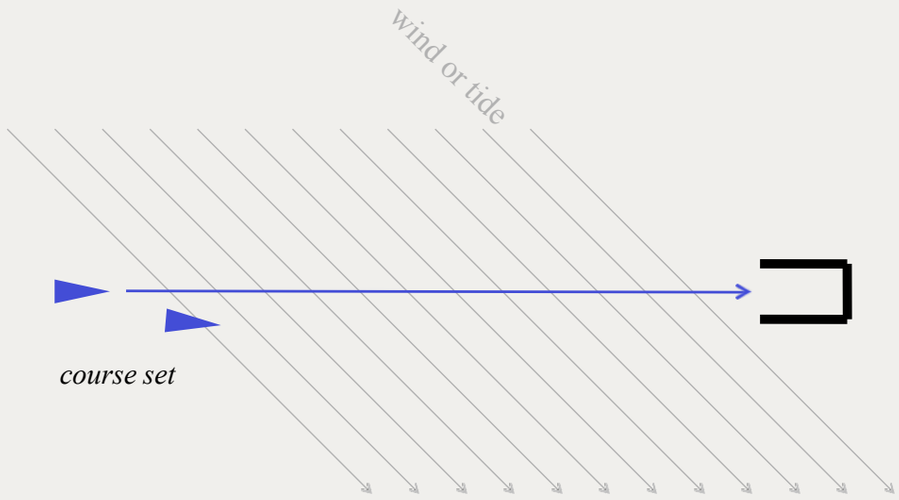


cybernetics explains how circular causal systems work or what we call single loops. So as well as defining the inputs CLICK and outputs we also need to factor in the goal CLICK of the actor and how factors in the environment can cause disturbances CLICK to the achievement of that goal.

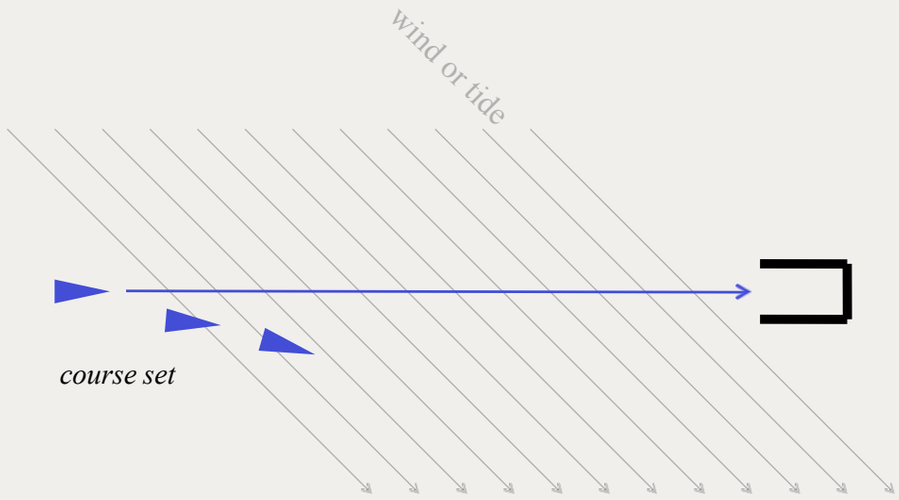
Art of Steering



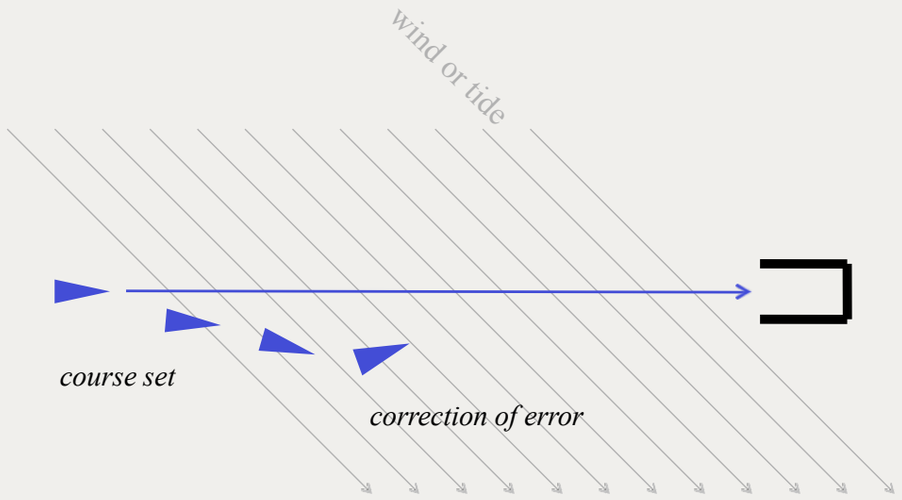
Art of Steering



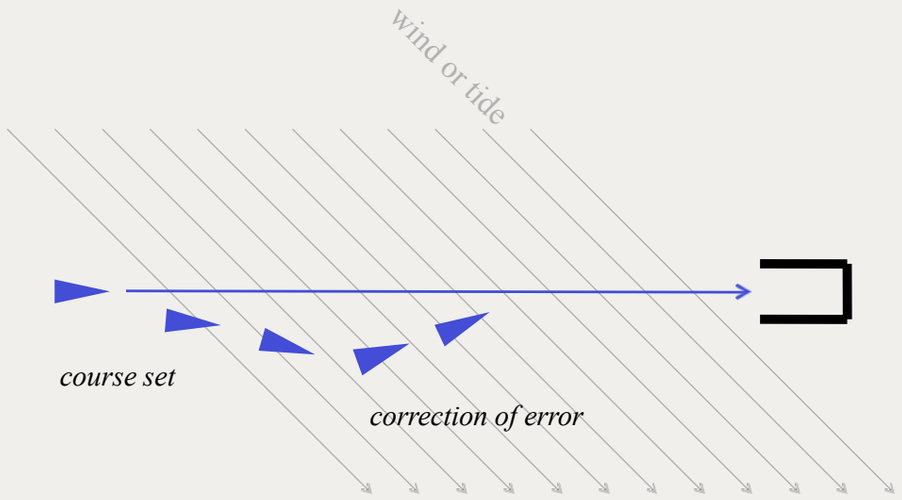
Art of Steering



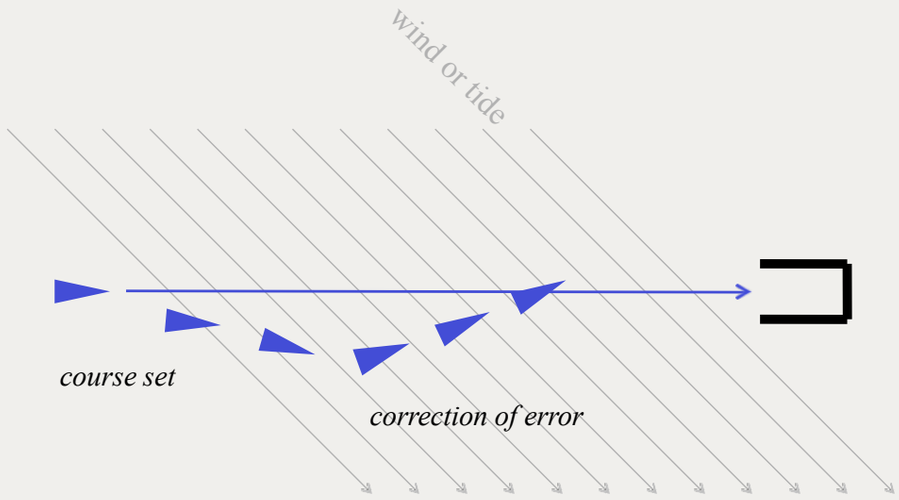
Art of Steering



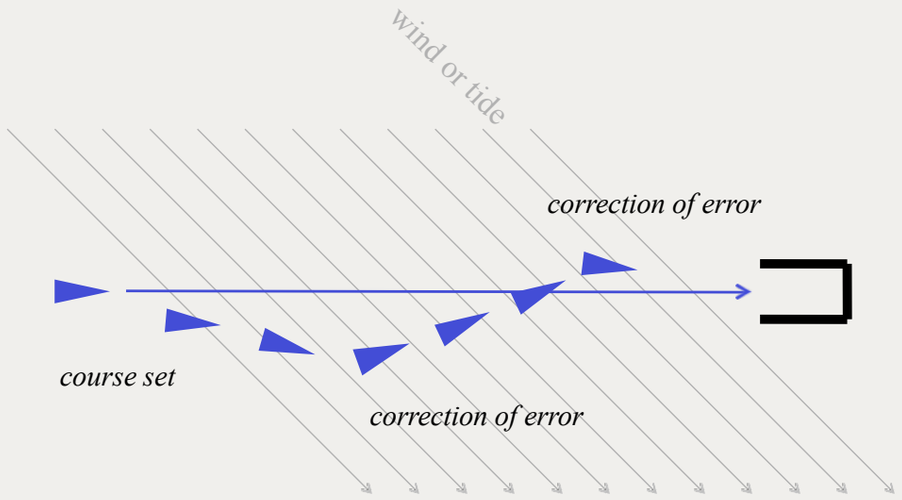
Art of Steering



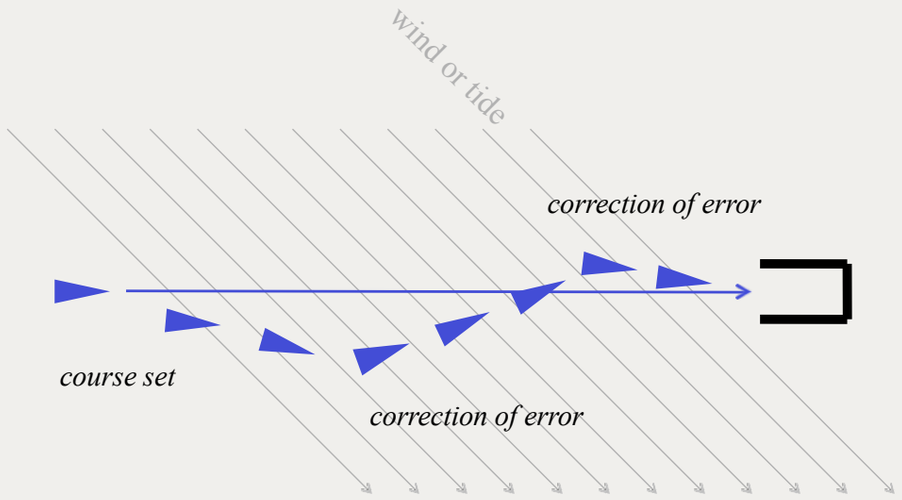
Art of Steering



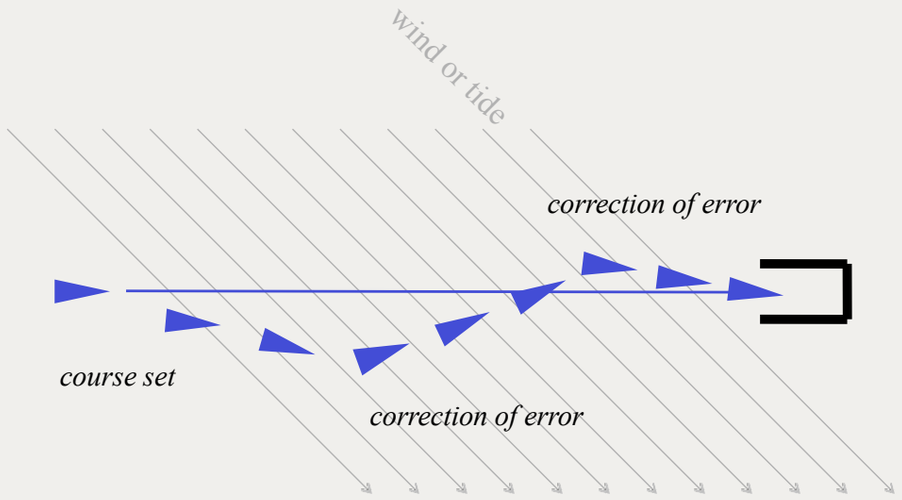
Art of Steering



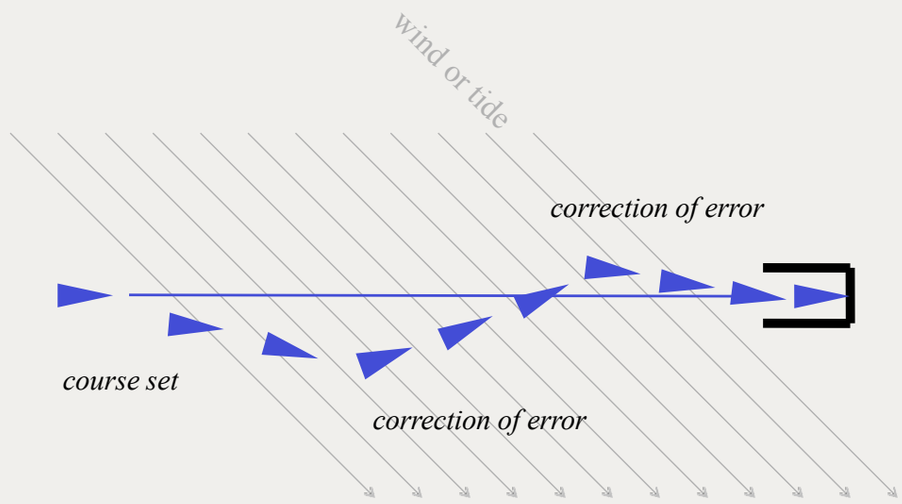
Art of Steering



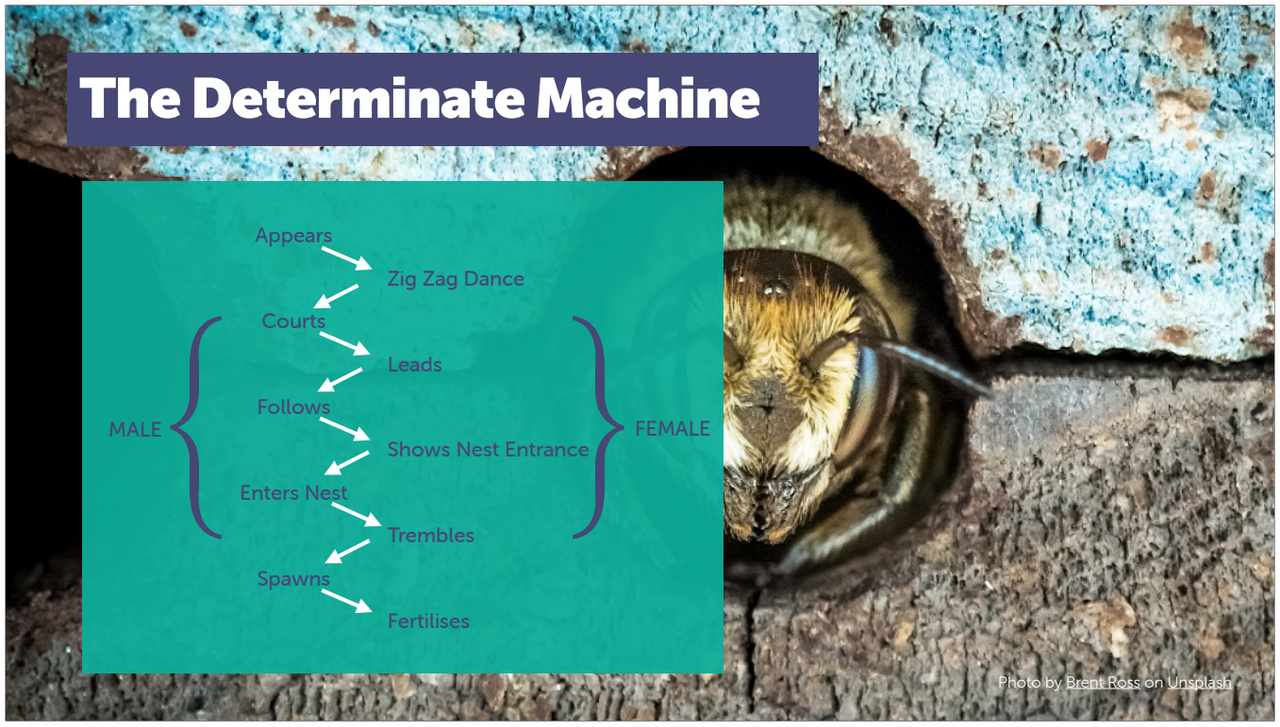
Art of Steering



Art of Steering



If we return to our steersman in Ancient Greece. We can see that the goal is to reach a safe port, and the input of rowing and wind in the sails should push the boat along its course. But the tide or wind of a real system like the sea can disturb the smooth procession of the vessel. The steersman has to measure the transformation and apply a correction to overcome the error created by the environment.



Cybernetics is concerned with the concept of determinate machines. Maths and physics is chiefly concerned with systems that are continuous and linear, but most natural systems are not linear and often not continuous and sometimes not even metrical ie not expressible in numbers. Therefore in order to understand these complex systems we have to define their states and the transformations and operands that occur between these states. A determinate machine is a simplification of a natural systems.

In this example of a study of insect mating behaviour one state proceeds to the next. Although the insect behaviour vary slightly. We can work on the basis that each state is relatively stable and will proceed in some fashion to the next.

J.C.R. Licklider, an MIT
psychologist who attended Weiners - Macy Conferences. In 1960, he wrote a seminal paper, “ManComputer Symbiosis” that employed cybernetic ideas to lay a roadmap for the future development of interactive computing

Calculating Transformations

A culture medium is inoculated with a thousand bacteria. Their number doubles every 30 minutes.

How do we express corresponding transformations?

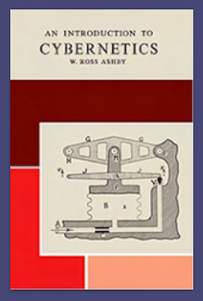
$$n' = 2n$$

It is an important function of cybernetics to calculate the transformations in states.
In this example we have a single operand - doubling and this behaves in a linear procedural way.
But what if we have multiple operands?

N derivative = 2n

A Machine with Input

Multiple Operands



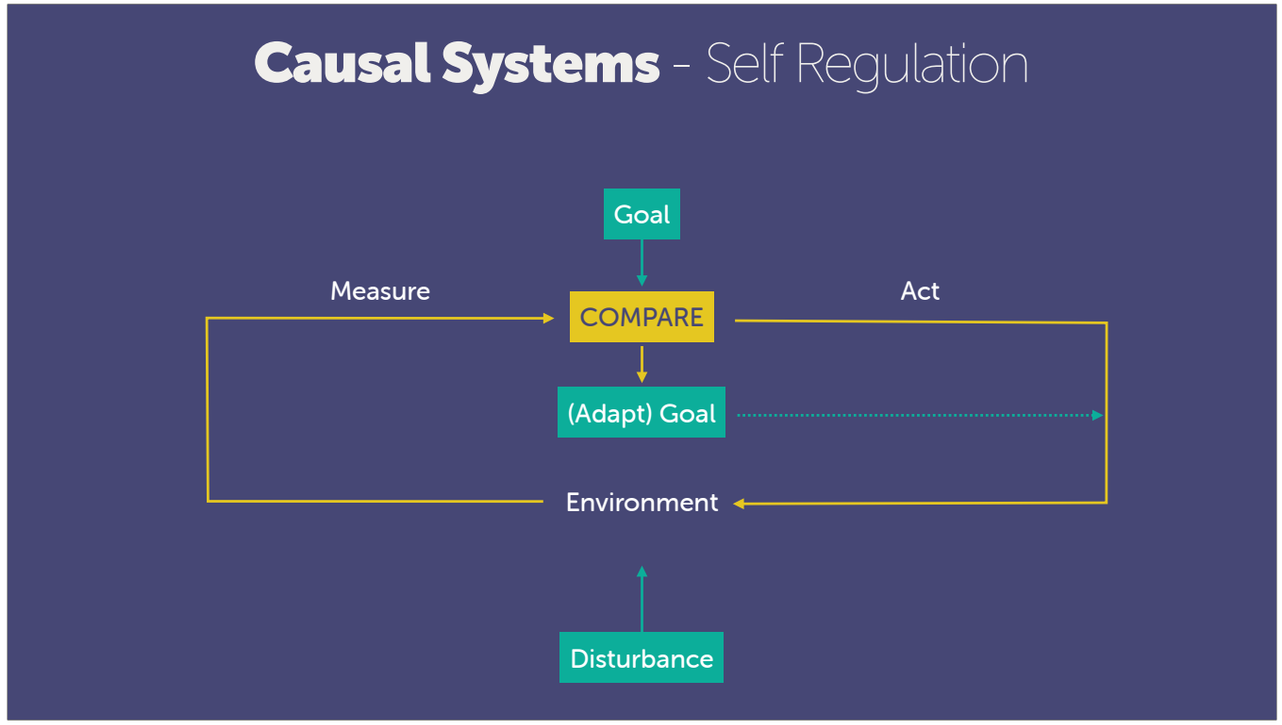
$\downarrow$	a	b	c
R_1	b	a	d
R_2	c	d	d
R_3	b	a	d

```
Vector3 movementMonster = new Vector3(-4, 9, 0);
monster1.transform.Translate(movementMonster);
monster1.transform.Translate(movementMonster * Time.deltaTime);
```

W. R. Ashby, Introduction to Cybernetics. London: Chapman & Hall, 1956.

Ross Ashby another key figure in Cybernetics whose book CLICK - An introduction to Cybernetics in 1956 described the machine with input and how multiple operands can change states. So he abstracted states and their transisitions CLICK we can have transformations R1, R2 and R3 and they change the states of a machine from a to b or c or d they can be measured in a table like this. This logical thinking underpins the way we manage transitions in matrices in computer generated imagery. Vector transforms are a good example of this thinking. CLICK

Inverse Kinematics



Cybernetics explains how circular causal systems work—
even when they self-regulate CLICK and modify their goals. As we saw in the example of the greek ship. Self-regulation is key.

Pain | Deck Nine, San Diego Studio (2007)

In this example of procedural animation in the game Pain from 2007 multiple events are chained together to create an evolving situation. The system acts measures the response to the environment and changes. It is self regulating. The rag doll seen here is one of the key applications of the kinematics.

Kinematics
Animation in Virtual & Real World Settings

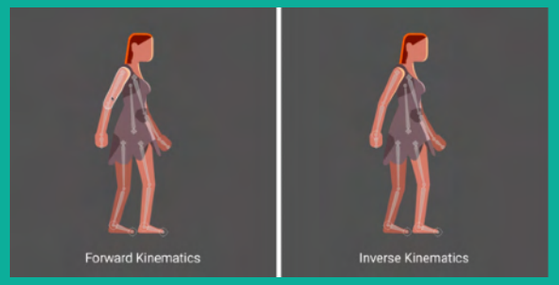
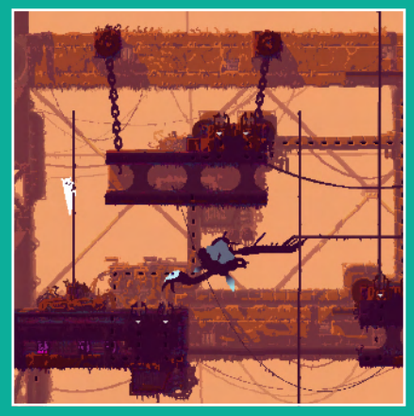


Image Attribution - Rive Software IK Help Centre



Rain World | Videocult (2017)

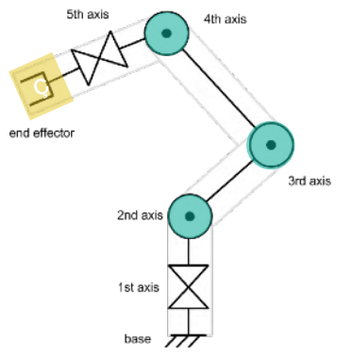
When we want to transform from one state to the next we need to apply some maths and physics to make things move effectively. Many robots and a lot of games are trying to mimic the movement of real world creatures. Most animals have joints and their movement is based on the rotation of numerous bones around various joints.

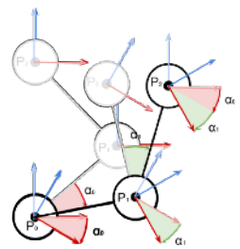
CLICK This process of moving around a joint is called Forward Kinematics when it is directly moving a joint to a position or inverse kinematics when the joint is being moved based on it’s relationship to the end effector. IN the example above the end effector is the characters hand.

In this example of a game Rainworld. CLICK The creators use procedural animation that uses Kinematics and more specifically Inverse Kinematics to achieve flowing interactions of objects. Having joints and limbs weighted by physics creates a life-like effect to the characters and their movement. As the characters move the environment impacts and alters their bodies as it might in real life.

Forward Kinematics

Transformations in space

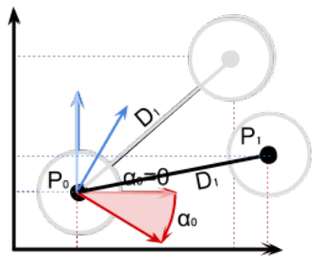




I am going to briefly touch on the maths and some code we apply to machines in order to make them move realistically.
A robot arm may have 3 joints CLICK and an end effector CLICK (this is the name used to describe the claw or gripper on the end.
If the arm moves around it’s joints we have to calculate the movement and rotations between each bone and it’s joint. CLICK, CLICK, CLICK

Measuring a single Tranformation

Maths



$$\alpha_0 = 0$$

$$P_1 = P_0 + D_1$$

$$P_1 = P_0 + rotate\left(D_1, P_0, \alpha_0\right)$$

$$P_2 = P_1 + rotate\left(D_2, P_1, \alpha_0 + \alpha_1\right)$$

$$P_i = P_{i-1} + rotate\left(D_i, P_{i-1}, \sum_{k=0}^{i-1} \alpha_k\right)$$

From the previous diagrams it should be clear to solve the problem of forward kinematics, we need to be able to calculate the position of nested objects due to their rotation.
Let’s see how this is calculated with just two joints. Once solved for two, we can just replicate it in sequence to solve chains of any length.
In this example we will start with the easy case, the one in which the first joint is in its starting position. This means that CLICK \alpha0=0, like in the following diagram:
This means that, simply: CLICK

\[P_1 = P_0 + D_1\]

When \alpha_0 is not zero, what we have to do is rotate the distance vector at rest D_1 around P_0 by \alpha_0 degrees:
Mathematically we can write this as:CLICK

\[P_1 = P_0 + rotate\left(D_1, P_0, \alpha_0\right)\]

By replicating the same logic, we can derive the equation for P_2:

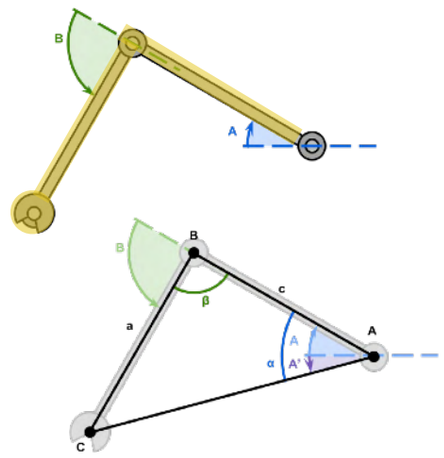
\[P_2 = P_1 + rotate\left(D_2, P_1, \alpha_0 + \alpha_1\right)\]

And finally, the general equation:

\[P_{i} = P_{i-1} + rotate\left(D_i, P_{i-1}, \sum_{k=0}^{i-1}\alpha_k\right)\]

Robotics - Inverse Kinematics in 2D

Two Degrees of Freedom



$$A = \underbrace{\cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2bc}\right)}_{\alpha} + \underbrace{\tan^{-1}\left(\frac{C_Y - A_Y}{C_X - A_X}\right)}_{A'}$$

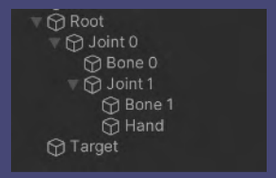
$$B = \underbrace{\pi - \cos^{-1}\left(\frac{a^2 + c^2 - b^2}{2ac}\right)}_{\beta}$$

In this scenario, we have a robot arm with 2 degrees of freedom.
The length of the arms, lower case c and a, is a known.CLICK If the point we have to reach is C, CLICK then the configuration becomes a triangle in which all sides are known. CLICK
We have then derived the equations for the angles A and B, which controls the rotation of the robotic arms' joints. CLICK

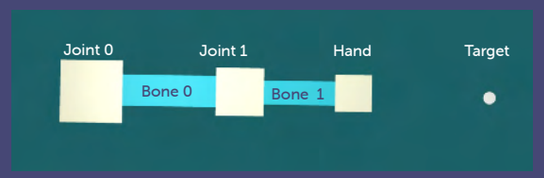
A Simple IK Example in Unity

Set-Up

Hierarchy



Scene



We can model this in Unity. The concept of “joints” is not something that Unity comes with. However, the parenting system offered by the engine can be exploited to create a hierarchy of components that will behave exactly like a robotic arm.
I have modelled an arm using various cubes. You can see how I have named them and parented them in the scene in Unity. In the first part of the script we assign the joints, hands and target as variables that take the transforms of the game objects

Unity Example

Lets take a look at how my simple IK system works and then we will explore it’s use in code.

A Simple IK Example in Unity

Inverse Kinematics Application

```
public class SimpleIK2D : MonoBehaviour
{
    struct IKResult
    {
        public float Angle0;
        public float Angle1;
    }

    [Header("Joints")]
    public Transform Joint0;
    public Transform Joint1;
    public Transform Hand;

    Vector3 nextPoint = prevPoint + rotation * Joints[i].StartOffset;

    [Header("Target")]
    public Transform Target;
```

$$A = \underbrace{\cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2bc}\right)}_{\alpha} + \underbrace{\tan^{-1}\left(\frac{C_Y - A_Y}{C_X - A_X}\right)}_{A'}$$

$$B = \underbrace{\pi - \cos^{-1}\left(\frac{a^2 + c^2 - b^2}{2ac}\right)}_{\beta}$$

In the first part of the script we assign the joints, hands and target as variables that take the transforms of the game objects. We also create a struct to contain the angle values of the 2 joints. CLICK

The joints are rotated by accessing the localEulerAngles property of the joints' Transform component. Unfortunately, it is not possible to change the z angle directly, so the vector needs to be copied, edited and replaced.CLICK

The equations derived from knowing the length of the first two bones (called c and a, respectively). Since the length of the bones is not supposed to change, it can be calculated in the IK bool in the floats *length* CLICK

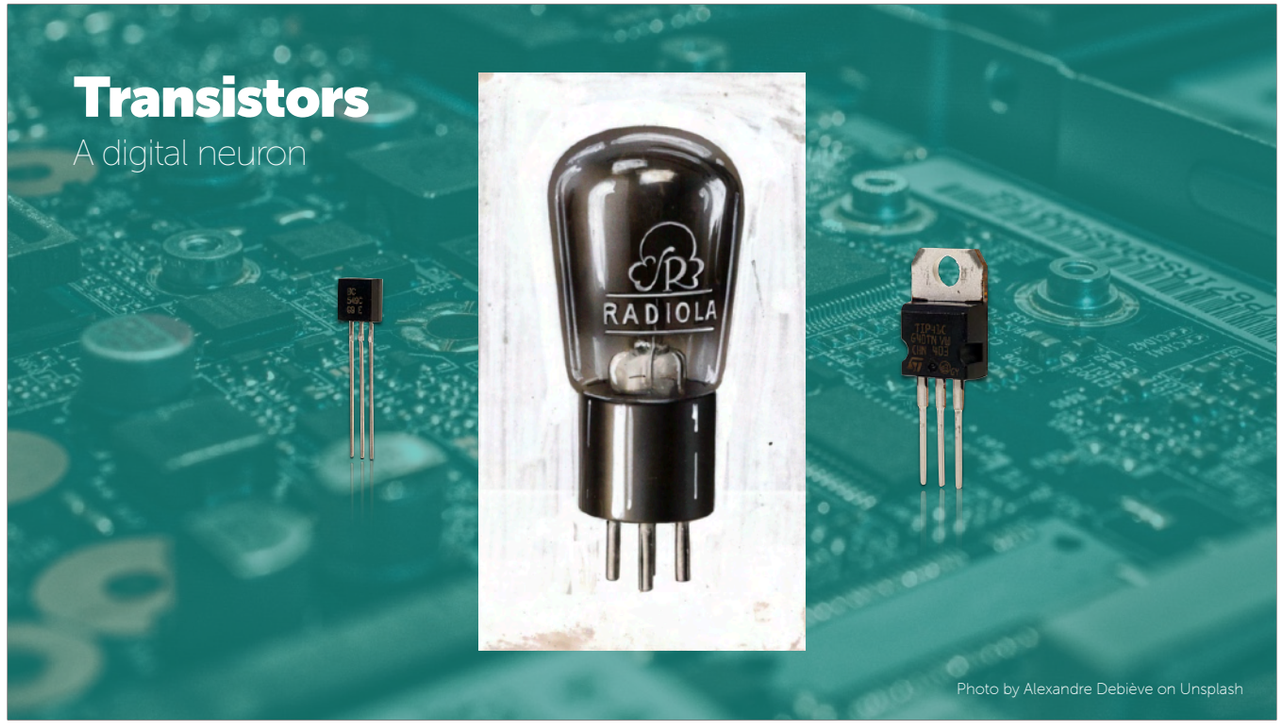
What happens if the target is unreachable? The solution is to fully stretch the arm in the direction of the target. Such a behaviour is consistent with the reaching movement that we are trying to simulate. The code detects if the target is out of reach by checking if the distance from the root is greater than that the total length of the arm. CLICK

Finally we have to calculate the angles. If we translate equations (A) and (B) directly to code, we end up with something like this. The mathematical functions cos^{-1} and tan^{-1} are called Mathf.Acos and Mathf.Atan2 in Unity. Also, the final angles are converted to degrees using Mathf.Rad2Deg, since the Transform component accepts degrees, instead of radians.

———-

The principle of Inverse Kinematics is at the heart of both robotic movement but also virtual movement in games. We have explored it here to demonstrate how

The Transistor

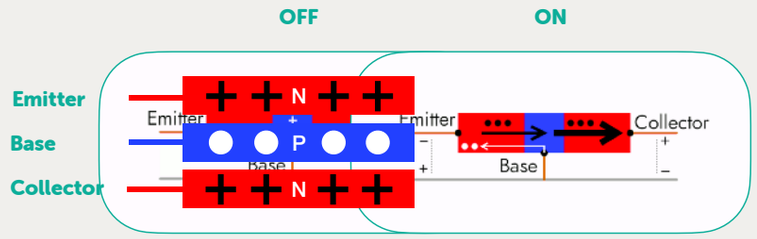


The digital revolution owes its success to one small component which is standard in all electronic devices - the transistor. CLICK The first Transistors were made in glass vacuum tubes in 1907(the above example is from the 1930s) however the first solid state transistor was not invented until 1947.
A transistor is a miniature electronic component that can do two different jobs. (CLICK) It can work either as an amplifier or a switch. As an amplifier, it takes in a tiny electric current at one end (an input current) and produces a much bigger electric current (an output current) at the other. In other words, it's a kind of current booster. That comes in really useful in things like radios where a small sound signal needs boosting. In the 50s and 60s radios were often referred to as ‘transistors’.

It’s other use is as a switch. A tiny electric current flowing through one part of a transistor can make a much bigger current flow through another part of it. In other words, the small current switches on the larger one. Another way of imagining this is to say we use electrical current rather than a finger to flick a switch. A modern memory chip contains hundreds of millions or even billions of transistors, each of which can be switched on or off individually. Since each transistor can be in two distinct states, it can store two different numbers, zero and one. If something can be in 2 states it is like a very small very primitive brain.

Transistors

How they work



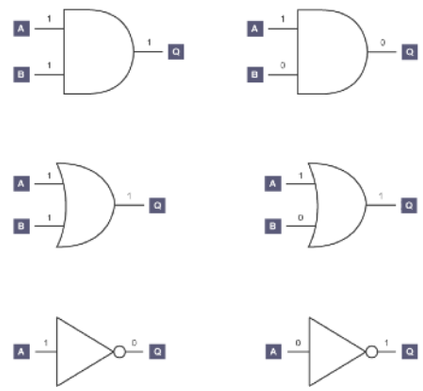
https://www.explainthatstuff.com/howtransistorswork.html

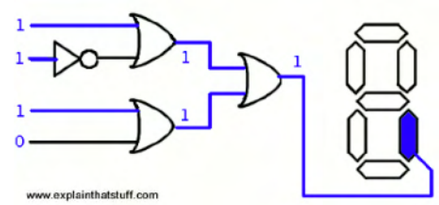
A transistor is comprised of 3 layers of silicon in a sandwich. CLICK The layers are comprised of either an n or a p type of silicon and so transistors are often called either NPN or PNP transistors, because of the order the materials are sandwiched. To understand the difference of the layers we can say that the n-type has a surplus of electrons, CLICK the p-type has holes where electrons should be. CLICK The layers have pins attached to them that are called the emitter, the base and the collector. CLICK Let's look at this arrangement from a different angle. Normally, the holes in the base act like a barrier, preventing any significant current flow from the emitter to the collector while the transistor is in its "off" state.

A transistor works when the electrons and the holes start moving across the two junctions between the n-type and p-type silicon.
If we connect the transistor up to some power. CLICK and we attach a small positive voltage to the base, make the emitter negatively charged, and make the collector positively charged. Electrons are pulled from the emitter into the base through these holes—and then from the base into the collector. And the transistor switches to its "on" state:

When there is no current to the base, little or no current flows between the collector and the emitter. Turn on the base current and a big current flows. So the base current switches the whole transistor on and off. So then we have a binary switch

How we make a bigger brain





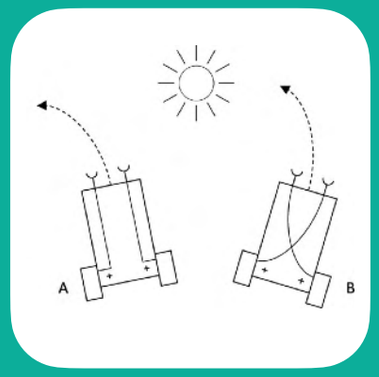
If a transistor acts as an electronic automated switch or a single neuron we can use them in concert with more transistors to create more states in the machine. Effectively by having a range of different gates - AND, OR and NOT we can control more complex states, like what part of seven segmented digital display is on simply by feeding a binary number sequence like 1,1,1,0.

I won’t go into great detail as you have already touched on the theory of this in COMP110 with the application of truth tables. Let’s look at a practical example in TinkerCAD.

Logic Gates in TinkerCad

Braitenburg Vehicles

The simplest form of mimicry



- Simple ‘eyes’ sensitive to light
- These are connected to neurons which respond to the eyes
- Which are connected to motors which drive the robot’s wheels

- The robot can be a ‘light seeker’
- Or a ‘light phobe’ (avoids light)
- Each behaviour can be considered ‘aggressive’ or ‘shy’

In the middle of the last century, Valentino Braitenberg proposed a form of robot with simple neurons. Born in Italy, Braitenberg was a neuroscientist and cyberneticist who became the director of the Max Planck Institute for Biological Cybernetics in Tübingen, Germany. He is most famous for his thought experiment the Braitenberg vehicle. A Braitenburg vehicle has:

CLICK
The vehicles are put in an environment with lights. The robot’s behaviour is set by how the neurons are connected.

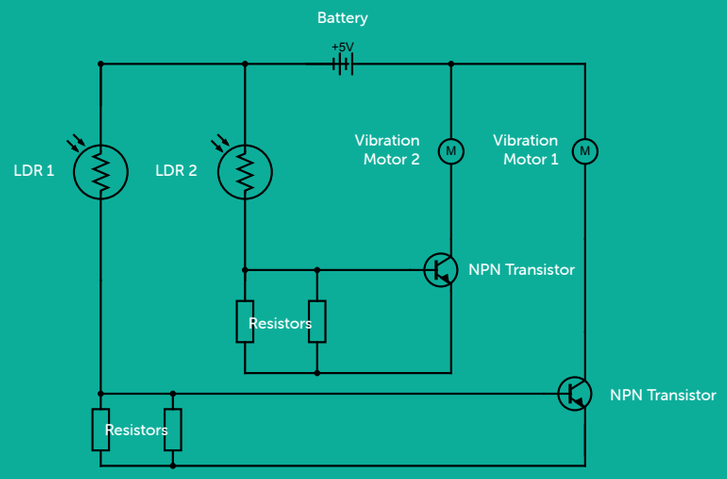
CLICK

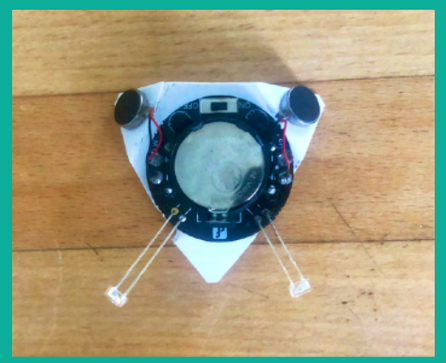
Let’s return to TinkerCAD and look at how we might build a simple robot like this. We’re going to look at the light seeker example.

Dimmer Switch in TinkerCad

To understand the principle of a Braitenburg vehicle it is important to understand how to build a self-regulating system using a transistor. Let’s take a look at how we might wire one up.

Braitenburg Vehicle Wiring





The vehicle is really a duplication of the previous wiring. That just incorporates a second sensor, motor, transistor and two more resistors.
I have used vibration motors the type found in mobile phones. CLICK These will vibrate the plastic triangular chassis to produce changes in direction.

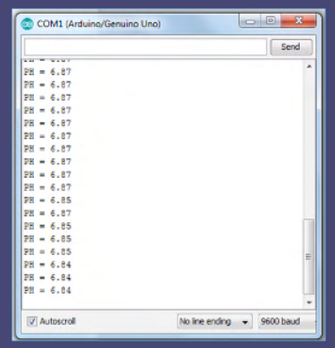
Braitenburg Example Robot

Let’s have a look at the robot in action. CLICK

What the Braitenburg vehicle demonstrates is that you can have an autonomous agent that can orient itself in space based on a simple set of sensors, a simple controller (2 transistors) that allows for variance in the output of it’s motors that translates directly from the input to it’s sensors. As humans we immediately assign animal like qualities to it’s behaviour. In this instance the robot is aggressive because it is always moving towards the light if we switch the polarity of the motors the robot will interpret less light to mean more power, this will produce a shy behaviour moving away from the light.

This demonstrates how simple sensing, controlling and actuating can produce complex intriguing behaviour.

Signal Processing



Fourier transforms
Finite Impulse Response

The Braitenburg example is a simple system and…

In most of your application for COMP140 you will probably have fairly predictable inputs from your analog sensors. CLICK Maybe measuring values that you can easily map to outputs in your game. CLICK
CLICK But in future you may want to pursue more complex methods of processing incoming signals and measuring more subtle change. This is where signal processing comes in. We can use various algorithmic approaches to detect the mean of a wave, or use Fourier transforms and Finite Impulse Response to work out patterns in a data set based on their change over time and the sum of their harmonic oscillations. Scientists classically have used this to measure the light from stars to estimate their distance from us.

In embedded systems one of the recent conundrums faced by the designers of Fitbits and similar health apps that make use of accelerometers is how to take the 3 dimensional vectors outputted by accelerometers CLICK and determine the states of activity being undertaken by it's user.
Are they cycling CLICK or running CLICK or standing still. This is where Signal Processing helps us to work out the complex variance in the signal produced. Robotics students will study this area in more detail next year.

Finite State Machines

The final part of our discussion about Cybernetics is to look at an aspect that is directly relevant to your goals in this module…

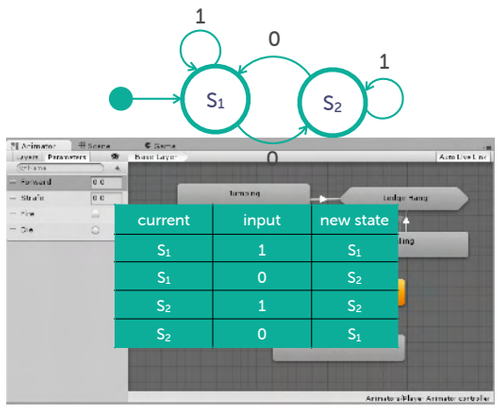
States and Transitions

The world is full of state machines from Traffic lights to thermostats to game controllers to turnstiles. CLICK
If we consider this example of a coin operated turnstile. A turnstile is locked (the first state) until the user introduces a coin it is then unlocked (the second state) and can pushed. Once it has been pushed it is then locked again and requires another coin.
As we have established cybernetics is concerned with states and the transformations between states - locked to unlocked and how we move between these states using transitions in this case a coin and a push.

Finite State Machine (FSM)

Modelling Logic

- An FSM consists of a **finite** number of states. At a given time only **one** such state is **active**.
- Each state defines which state it'll **transition** to as an **output**, based on the **sequence** of **inputs** it receives.
- The output state becomes the **new active state**. In other words, a **state transition** occurs.



Finite State Machines is in abstract method of modelling logic in a machine. It can be a real machine or a virtual one. A UML state diagram in its pure form identifies a numbered state and then assigns numbered transitions to establish the relationship between states. The table illustrates the inputs and transformations to the new states. CLICK You are probably familiar with seeing them in Unity to manage animations in games. The animation state machine manages states or animation cycles and their transition to others.

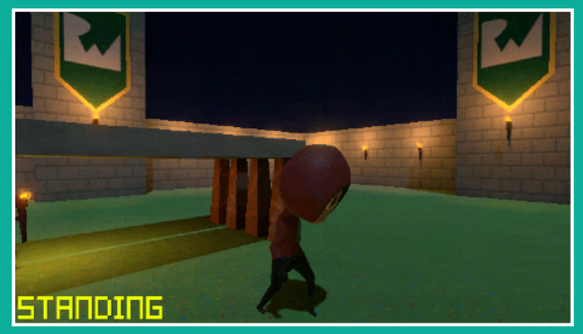
State Machines in Arduino

State machines are vital in embedded systems as we have to manage power time and the sequence of control to actuators like motors, servos and lights.
Let’s look at an example in Arduino.

OOP - Finite State Machines

State Design Pattern

A behavioural design pattern that lets an object alter its behaviour when its internal state changes.

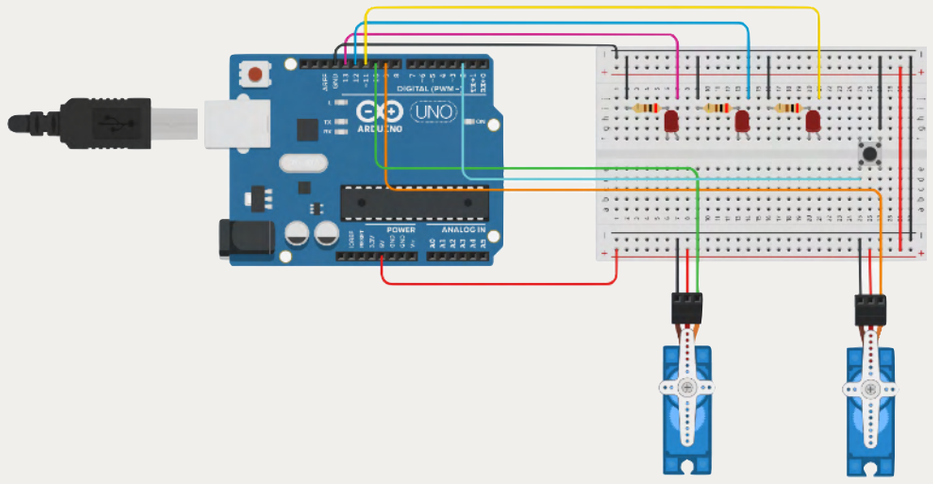


https://www.raywenderlich.com/%206034380-state-pattern-using-unity

https://refactoring.guru/design-patterns/state

We have covered Object Oriented Design patterns and their application. The state pattern - READ
This pattern is especially useful for defining state machines. Ray Wenderlich CLICK has a great article on using the pattern to control animation states in player characters in Unity. You can also explore the basic implementation of the state at refactoring guru.

UML for an **Arduino State Machine**

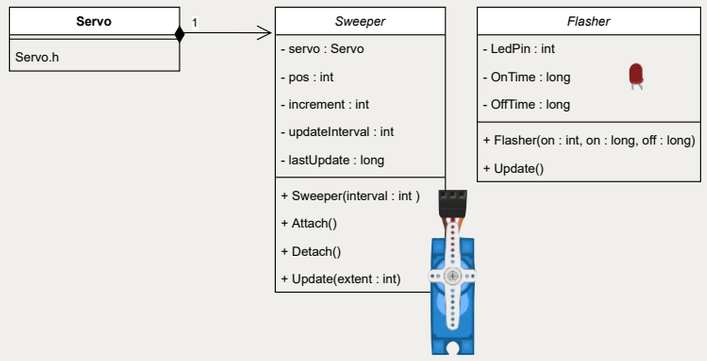


But we are going to focus on creating a Finite State Machine to handle multiple machines in Arduino. It is common in robotics or electronics projects to end up with numerous inputs and outputs and this is all power and resource hungry. We are going to work with this example layout with 3 Led’s and 2 servo’s and a button to create a changes in states.

GO to version 1

UML for an **Arduino State Machine**

Class Diagram



This is a clumsy way to build an effective system that can handle states so lets build an Finite state machine that is not tied to the loop but is running off the clock this will increase the performance of the system.

To do this we will use bit of OOP to create 2 classes - sweeper and flasher. We can then generate instances of these classes to create more state machines.

Note that we have variables to keep track of whether the LED is ON or OFF CLICK and method determine whether a servo is attached and running or not CLICK. And variables to keep track of when the last change happened. That is the State part of the State Machine.

We also have code that looks at the state and decides when and how it needs to change. CLICK That is the Machine part. Every time through the loop we ‘run the machine’ and the machine takes care of updating the state. Finally we have an interface to the Servo library which is essential for the servos functioning. This is a standard Arduino library.

Conclusions

1. We have addressed the context and history of **cybernetics**
2. We have looked at **causal loops** and how self regulating systems apply to procedural motion in games and robots with an example of **inverse kinematics**
3. We have explored the role of the **transistor** and how it can provide the simplest implementation of a robot control system.
4. We have touched on **signal processing** and its role in interpreting data from sensors.
5. Finally we looked at how we can use **finite state machines** to control a series of sensors and actuators that facilitate stable transitions and efficient management of memory and power resources.