# Student's Handbook Guide to CMPUT 503 Experimental Mobile Robotics

A student's guide to the design and implemenation of mobile robotics

This course is an experimental inquisition of the fascinating field of mobile robotics. Students will specifically be exposed to basic concepts, models, and algorithms in autonomous navigation of a mobile robot operating in indoor environments. The instruction of the course will be through the Robot Operating System (ROS), which provides libraries and tools to help software developers quickly create robot applications. A mobile robot vehicle will be provided to the students, to conduct real experiments. It is equipped with a range sensor as well as a camera for the robot to perceive its environment and make navigational decisions. The course will be developed incrementally, through demos and competitions. The robot tasks that define the demos and competitions will begin with simple motion control and evolve toward a comprehensive term project involving robot mapping, localization, path planning, object detection and homing.

By slaying his assignment minions, defeating his midterm champion(s) and ending the final themselves. The heroic legacy of your achievement will be remembered on your academic transcript.

Disclaimer: This book is not responsible for any personal injuries caused by uncontrolled robotic behaviour and/or uprising.

# Contents

# Chapter 1: Robotics Architectures

## Reactive Architecture

Actions are directly triggered by sensors

- **No representations** of the environment
- **Predefined**, fixed response to the situations
- **Fast** response to changes in the environment

Basically, all actions are episodic

***Limitations.***

- Knowledge of the world is limited by the range of its sensors
- Unable to count
- Unable to recover from actions which fail silently
- Anything that requires memory
- Not possible to "plan ahead"
- Can not coordinate with other robots in a reasonable way
- ...

## Deliberative Architectures

These architectures are organized by decomposing the required system functionality into **concurrent modules** or components so things like:

- Map building
- Path planning
- Navigation
- ...

***Problems.*** These types of systems are overall more complex and hard to offer real-time guarantees on performance. Solving any given problem takes longer than an equivalent reactive implementations due to the overhead of such an architecture. Moreover, solving different problems takes different amounts of time.

## Temporal Decomposition

**Temporal Decomposition** distinguishes between processes that have varying real-time and non real-time demands

1. off-line - e.g., Things to do if you have spare compute cycles
2. strategic - e.g., Getting from from CCIS to ECHA for class
3. tactical - e.g., The path from CCIS to ECHA
4. real-time - e.g., Staying on the path from CCIS to ECHA
5. hard real-time - e.g., Avoiding students from CCIS to ECHA

The higher the number the shorter or more important to have a short response time, generally done in a local context. The lower the number the longer or less important to have a short response time, generally done in a global context.

## Subsumption Architecture

Formed using a collection of concurrent behaviours placed in layers. The higher-level behaviours always, if triggered, subsume the output of lower behaviours and therefore have overall control.

## ROS

ROS is a “meta” OS for robots, it has a collection of packages and software for building tools. It’s an architecture for distributed inter-process and inter-machine communication and configuration. There are development tools for system run time and data analysis. Language and platform agnostic. Really, ROS is a **peer-to-peer** robot middleware package/framework. It allows easier hardware abstraction and code reuse. In ROS, all major functionaliy is broken up into a number of chinks that communicate with each other using messages. Each chunk is called a node and is typically run as a separate process. Making matching between nodes is done by the **ROS Master**. Don’t try and use ROS as hard real-time architecture, there is too much overhead without specialization.

### ROS Nodes

**Node**
A process that performs some computation.

Typically we try to divide the entire software functionality into different modules. Each one is run over a single or multiple node. Nodes are combined together into a **graph** and communicate with one another using streaming **topics** and other ways. But generally using **topics**. These nodes are meant to operate at a fine-grained scale and generally a robot control system will usually comprise of **many nodes**.

### ROS Topics

**Topics**
Named buses over which nodes exchange messages.

Topics have anonymous publish/subscribe semantics. I.e., nodes do not care which node published the data it receives, or which one subscribes to the data it publishes. There can be **multiple** publishers and subscribers to a topic. Topics are strongly typed by the ROS message it transports. Transports is done using TCP or UDP.

## ROS Messages

Nodes communicate with each other by publishing messages to topics. A **message** is a simple data structure, comprising typed **fields**. You can take a look at some basic types:

- std_msgs/Bool
- std_msgs/Int32
- std_msgs/String
- std_msgs/Empty - This is useful for debugging and testing modules

Messages may also contain a special field called **header** which gives a timestamp and frame of reference.

# Chapter 2: Locomotion

**Locomotion** is the act or ability for the robot to move from place to place. There are many that are bio-inspired like walking, jumping, running, sliding, flying, rolling, swimming, etc There is one exception to this rule, and that is the **Powered wheel** which is a human invention. The reason why the powered wheel is so common is that it is extremely efficient in terms of complexity (or the lack of), and the amount of power needed for an attainable speed. It is extremely efficient with regards of using energy to move on a plane.

## Potential issue

***Stability.*** There are many factors that could affect the stability of a robotic system:

- number and geometry of contact points
- CoG
- Static and dynamic stability
- Inclination of terrain

***Characteristics of contact.***

- Contact point/path size and shape
- angle of contact
- friction

***Type of environment.***

- Structure
- Medium i.e., water, air, sand, mud, etc

## Legged Mobile Robots

These are robots that are characterized by a series of point contacts between the robot and the ground, usually have joints. The benefit of such a system is that they provide adaptability and manoeuvrability in rough terrains. The drawbacks are that they have much higher power requirements and mechanical complexity, high degrees of freedom, and complex control systems.

### Leg Configurations

**Definition 2.0.1** (Degrees of Freedom)**.** *Basically the number independent movements in a rotational or transnational direction.*

A minimum of 2 DOF is required to move a leg forward, one is lift and the other is swing motion. 3 DOF is generally required for each leg in most cases. A fourth DOF is needed for the ankle joint, this might improve walking but also adds complexity in both design and control. Arms often have 6 or 7 DOFs.

### Legged Robot Control

**Definition 2.0.2** (Gait Control)**.** *Leg coordination for locomotion*

The **gait** is the sequence of lift and release events for the individual legs. For a robot with $k$ legs, the total number of distinct event sequences $N$ is $N = (2k-1)!$.

For a 2 legged robot there are 6 distinct event sequences:

- DD, UD, DD
- DD, DU, DD
- DD, UU, DD
- UD, DU, UD, DU
- UD, UU, UD
- DU, UU, DU

But for 6 legged robots there are 11! or almost 40 million distinct event sequences.

#### Stotting

Also known as pronking or pronging is a unique gait that quadrupeds, particularly gazelles, where they spring into the air by lifting all four feet off the ground simultaneously. Some evidence shows that this is a honest signal to predators that the prey is not worth pursuing and tell the predator to pick another slower prey.

### Cost of transportation

- How much energy a robot uses to travel a certain distance
- Usually normalized by the robot weight
- Measured in J/N-m

There is a range of speed that is efficient for each type of gait. Generally the faster the speed the more of a galloping gait is required for efficient locomotion.

It is better to design systems to exploit the dynamics of walking or other natural phenomena:

- Natural oscillations of pendula and springs
- Dynamics of a double pendulum
- Springs and can be used to store energy
- Passive dynamics walkers

## Wheeled Mobile Robots

Wheel robots in contrasts are highly efficient and simple in terms of mechanical implementation. Therefore, they are the most popular form of locomotion for robotics. While different systems are required like suspensions to keep the wheels in contact with the ground on uneven terrain they do not usually exhibit balancing issues like legged robotics.

Wheel robotics are focused on

- Traction
- Stability
- Manoeuvrability
- Control

## Wheel designs

Most wheels like standard or castor wheels have 2 DOF. While more specialized wheels like the Swedish (omni) wheel and the ball or spherical wheel have 3 DOF. But these types of wheels have suspension issues due to their more complicated design.

## Stability

At least 3 wheels are required to guarantee stability so long as the CoG is within the triangle formed by the ground contact points of the wheels. Stability is improved by adding more wheels. The bigger the wheels allow the robot to overcome higher obstacles but requires more torque than smaller wheels. Most arrangements are **non-holonomic** and requires a high control effort. Combining actuation and steering on one wheel makes design complex and adds additional error for odometry.

Now you can do static stability using two wheels. Examples include Segways, and Hoverboards. This is done by ensuring the center of mass is below the wheel axis.

# Motion Control

To control any robot we first need to create the kinematics or dynamic model of the robot. This model for our case is the interaction between the wheel and the ground. Thus we require

- Speed control
- Position control

Control law that satisfies the requirements

## Mobile Robot Kinematics

**Kinematics** is the description of mechanical behavior of the robot based on actuator action. Similar to robot manipulator kinematics However, mobile robots can move unbound with respect to their environment:

- No direct way to measure robot's position
- Position must be integrated over time
- Lead to inaccuracies of the position (motion) estimate

## Definitions

**Definition 2.0.3** (Configuration). *Complete specification of the position of every point of the system, this includes the position and orientation. This is also called* ***pose****.*

**Definition 2.0.4** (Configuration Space). *The set or space of all possible configurations or poses.*

**Definition 2.0.5** (Workspace). *The 2D or 3D ambient space the robot is in.*

One could brute force the configuration space by sliding the robot around an obstacle and keeping track of the curve traced out by the reference point thus creating an area that is not in the configuration space. But that is not scalable for all obstacles and robot design. So we use math to describe how robots should move.

**Definition 2.0.6** (Inverse-kinematics). *The inverse of kinematics or how should be move the actuators to get the robot to do what we want to do.*

## Kinematics

The robot knows its movement relative to centre of rotation $P$. However, this is not the same as knowing how it moves in the world.

There are two **frame** of references:

- Initial Frame
- Robot Frame

Let the robot be at the initial frame: $x_I, y_I, \theta_I$. However, to move to a different location it cannot control $x_I, y_I, \theta_I$ directly.

The robot only knows:

- The speeds of the wheels: $\dot{\varphi}_1, \ldots, \dot{\varphi}_n$
- Steering angle of steerable wheels: $\beta_1, \ldots, \beta_m$
- Speed with which steering angles are changing: $\dot{\beta_1}, \ldots, \dot{\beta_m}$

These define the forward motion of the robot, or the **forward kinematics**:

$$f(\dot{\varphi_1}, \ldots, \dot{\varphi_n}, \beta_1, \ldots, \beta_m, \dot{\beta_1}, \ldots, \dot{\beta_m}) = [\dot{x_I}, \dot{y_I}, \dot{\theta_I}]^{\mathrm{T}}$$

But what we want are the **inverse kinematics**

$$[\dot{\varphi_1}, \ldots, \dot{\varphi_n}, \beta_1, \ldots, \beta_m, \dot{\beta_1}, \ldots, \dot{\beta_m}]^{\mathrm{T}} = f(\dot{x_I}, \dot{y_I}, \dot{\theta_I})$$

So we have this robot position

$$\xi_I = [x_I, y_I, \theta_I]^{\mathrm{T}} = \begin{bmatrix} x_I \\ y_I \\ \theta_I \end{bmatrix}$$

What we need is to map motion between frames

$$\dot{\xi_R} = \text{something } \dot{\xi_I}$$

Let that something be $R(\theta)$ and have that be:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Subbing that in the mapping equation gives us:

$$\begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\theta_R} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x_I} \\ \dot{y_I} \\ \dot{\theta_I} \end{bmatrix}$$

But we do not want this we want the inverse of this equation i.e., we want this: $\dot{\xi}_I = R(\theta)^{-1}\dot{\xi}_R$

So the inverse of $R(\theta)$ would be this:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Subbing that into the previous inverse equation gives us this:

$$\begin{bmatrix} \dot{x_I} \\ \dot{y_I} \\ \dot{\theta_I} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x_R} \\ \dot{y_R} \\ \dot{\theta_R} \end{bmatrix}$$

If we know the relative changes in $x$, $y$, and $\theta$, we can find the global position. But how do we know what these values are?

## Speed of a wheel

***Assumptions.***

- Movement on a horizontal plane
- Point contact of wheels
- Wheels are not deformable
- Pure rolling: no slip
- No friction from rotation
- Steering axes orthogonal to surface
- Wheels connected by rigid frame

Therefore the translational velocity of the wheel is:

$$v = \dot{\varphi} \cdot r$$

Where:

| Notation | Definition |
|---|---|
| $v$ | wheel translational velocity |
| $\dot{\varphi}$ | wheel rotational velocity in rads/sec |
| $r$ | radius of the wheel |

## Differential Drive

A robot that uses differential drive uses two independently controlled motors on each side for forward and backward drive as well as for rotation. If wheels rotate at $\dot{\vartheta}$ then each wheel contributes $\frac{r\dot{\varphi}}{2}$ to motion of centre of rotation.

**Speed of a Differential Drive Robot**

Is the sum of two wheels

**Rotation of a Differential Drive Robot**

Is the difference between the translational velocity of the right wheel and the translational velocity of the left wheel after the distance between the wheel and the center of rotation has been accounted for.

$$\dot{\theta_R} = \frac{r\dot{\varphi}_r}{2l} - \frac{r\dot{\varphi}_l}{2l}$$

Where:

| Notation | Definition |
|---|---|
| $\dot{\varphi}_l$ | The rotational speed of the left wheel |
| $\dot{\varphi}_r$ | The rotational speed of the right wheel |
| $l$ | The distance between the wheel and center of rotation |
| $r$ | The radius of the wheel |

Combing these components you get

$$\begin{bmatrix} \dot{x}_R \\ \dot{y}_R \\ \dot{\theta}_R \end{bmatrix} = \begin{bmatrix} \frac{r\dot{\varphi}_r}{2} + \frac{r\dot{\varphi}_l}{2} \\ 0 \\ \frac{r\dot{\varphi}_r}{2\cdot l} - \frac{r\dot{\varphi}_l}{2\cdot l} \end{bmatrix}$$

## Sliding Constraint

Standard wheel has no lateral motion. Therefore, if the robot moves in a circle it rotates at the center point which is the intercept of all the wheels' axes. This is called the **Instantaneous Center of Rotation**.

There are more complex parameters with caster wheel than with steered standard wheels but the idea is still the same.

## Degree of Mobility

**Definition 2.0.7** (Degree of Mobility)**.** *The number of DoFs of the robot chassis that can be immediately manipulated through changes in wheel velocity.*

## Degree of Steerability

**Definition 2.0.8** (Degree of Steerability)**.** *The nubmer of centered orientable wheels that can be steered independently in order to steer the robot. This indirectly impacts the robot chassis's pose.*

## Degree of Manoeuvrability

**Definition 2.0.9** (Degree of Manoeuvrability)**.** *The overall degrees of freedom that a robot can manipulate, it is the sum of the* ***degrees of mobility*** *and* ***degrees of steerability***

$$\delta_M = \delta_m + \delta_s$$

There is no ideal drive configuration that simultaneously maximizes stability, manoeuvrability, and controllability. Everything is a compromise. Typically there is an inverse correlation between controllability and manoeuvrability .

# Holonomic Robots

Holonomic kinematic constraint can be expressed as explicit function of position variables only. Non-holonomic constraint requires additional information like heading, etc. Fixed and steered standard wheels impose non-holonomic constraints.

**Definition 2.0.10** (Holonomic)**.** *If the* ***controllable*** *DoF is* ***equal*** *to the* ***total*** *DoF then the robot is said to be* ***holonomic***

**Definition 2.0.11** (Non-holonomic)**.** *If the* ***controllable*** *DoF is* ***less*** *than the total DoF then it is* ***non-holonomic***

A robot is considered to be **redundant** if it has more controllable DoF than DoF in its task space.

## Kinematic model for a car-like robot

Let's say we have a car whose parameters are as follows:

$$\{x, y, \theta, \phi\}$$
$$\{u_1, u_2\}$$

Where:

| Notation | What it means |
|---|---|
| $x$ | x position in world |
| $y$ | y position in world |
| $\theta$ | Car angle in the world |
| $\phi$ | Steering angle |
| $u_1$ | forward velocity |
| $u_2$ | Steering velocity |

Then the kinematic for this model is:

$$\dot{x} = u_1 \cos\theta$$
$$\dot{y} = u_1 \sin\theta$$
$$\dot{\theta} = \frac{u_1}{l} \tan\phi$$
$$\dot{\phi} = u_2$$

The non-holonomic constraints being $\dot{x}\sin\theta - \dot{y}\cos\theta = 0$

# Chapter 3: Manipulators

**Definition 3.0.1** (Manipulators). *They manipulate something the world by physicall altering the world through contact. This is their primary goal, but they do not control their own position.*

These robots are desirable because they are much better at working in dangerous work spaces like space, foundries, etc. Dirty, dull, boring, repetitive, unpleasant work or any human-intractable work spaces. Either because it's too small/big, or too much precision is needed.

Examples include:

- Industrial application like welding, drilling, attaching, painting, etc.
- Surgery
- Space exploration
- Chores
- Patient care
- Delivery

## Terminology

**Definition 3.0.2** (Actuator). *Generates motion or force, usually a motor*

**Definition 3.0.3** (DoF). *Degrees of Freedom, same meaning as before in the previous chapter*

**Definition 3.0.4** (End Effector). *Device at the end of an arm that interacts with the environment*

**Definition 3.0.5** (Gripper). *Self-explanatory, a type of manipulator*

## Manipulator Robot

Modelled as a chain of rigid links connected by joints. **Links** are unjointed length of robot. **Joints** have translational or rotational movement. **Manipulator / End Effectors** these can have grippers or tools and can have sensor.

## Characterization

There are different types:

- by drive
- actuation - tendons, direct servoing, under actuation
- by motions - Prismatic (linear), revolute (rotational)
- by characteristics - Payload, working area

## Joints

A joint represents a connection between two links. Denotation of relative displacement between links:

- $\theta$ for revolute joint
- $d$ for prismatic joint

Denotation of axis of motion $z_i$ beteween link $i$ and link $i+1$:

- Axis of rotation of a revolute joint
- Axis of translation of prismatic joint

**Definition 3.0.6** (Prismatic). *Anything that is sliding or translational. Usually denoted as $P$.*

**Definition 3.0.7** (Revolute). *Anything that rotate. Usually denoted as $R$.*

## Actuator

There are many kinds

### Hydraulic and pneumatic

Use in heavy load, and high speed application. Sometimes to hard to control especially pneumatic. Doesn't produce sparks.

### Electro-mechanical

These are the most common types of actuators.
***DC Motor.*** Doesn't know anything about precise rotation it, spins and doesn't spin, open loop.
***Stepper motor.*** subdivides a rotational into # increments but are open loop they have no feedback for correction. They also consume power continuously as they need the power to fix the rotor.
***Servo motor.*** Subdivides a rotation arbitrarily and are closed loop so they can provide feedback for correction. Doesn't require power if they are not moving.

## Configuration Space

**Definition 3.0.8** (Configuration). *Specification of location of every point on a manipulator.*

How can we specify something in the configuration? Well Links are rigid and we assume that the base is fixed. Therefore, so long as we know the values for the joint variables, i.e., the angle for R and the offsets for P joints then we can add them together to get the location of the end effector.

The set of all possible conjurations is called the **configuration space**.

## DoF

A system has $n$ DoF if exactly $n$ parameters are required to completely specify the configuration. Rigid objects in 3D space has six parameters, 3 for position (x,y,z), and 3 for orientation (roll, pitch, and yaw angle). If the DoFs is less than 6 then the arm cannot reach every point in the workspace with arbitrary

orientation. Sometimes you need more DoFs if you are dealing with obstacles. Having more than 6 DoF is kinematically redundant.

# Workspace

Configuration only provides geometry

**Definition 3.0.9** (Workspace)**.** *Set of all possible position of the end effector.*

**Definition 3.0.10** (Dexterous workspace)**.** *Set of points where the end effect can be in any orientation. It is a subset of the* ***workspace****.*

# Grippers

There are 4 kinds

- Impactive - Jaws or claws which physically grasp by direct impact upon the object
- Ingressive - Pins or needles penetrate surface, handling textile, carbon and fiberglass.
- Astrictive - Suction forces applied to surface, things like vacuum, electro-magnetic, etc
- Kontugutive/Contigutive - Requiring direct contact for adhesion, things like glue, surface tension, or freezing.

## Grasps

A set of contact points on an object's surface. The goal is to constrain the object's movement. They can vary by:

- Hand (gripper)
- Object being grasped
- Type of motion desired

For each hand or hand/object pair:

- Where to grasp it?
- How hard?
- Then what?

  Additional constraints i.e., don't spill the stuff.

## The Grasping Problem

Grasps are not obvious (easy to calculate). Any given object has arbitrary contact points. Hand has geometry constraints, etc. Synthesized trial-and-error or a given hard/object pair. Different grasp types planned and analyzed. You can also do trial and error in real-life.

## Grasp Planning

**Definition 3.0.11** (Grasp synthesis)**.** *Find suitable set of contacts, given the object model and constrains on allowable contacts*

**Definition 3.0.12** (Grasp points)**.** *Are determined mostly assume point contacts, larger areas usually discretized. Contact model defines the force the manipulator exerts on contact area.*

**Definition 3.0.13** (Grasp Analysis)**.** *Is that grasp sable?*

# Chapter 4: Vision

As a given example how to find a red ball using a camera? What if it's moving?

## HSV, HSL, HSV

**Definition 4.0.1** (HSV). *Hue, Saturation, Value*

**Definition 4.0.2** (HSL). *Like HSV but replace value with lightness.*

Generally HSL and HSV are deaiser to define and are closer to human vision than RGB.

## Perception

**Vision** is our most powerful sense in aiding our perception of the 3D world around us. Our retina is around 1000mm$^2$ and contains millions of **photoreceptors**. They provide an enormous amount of information (approx 3 GBytes/sec) and a large proportion of our brain power is dedicated to processing the signal from our eyes.

## Vision-based Sensors: Hardware

**Definition 4.0.3** (CCD). *Charged-coupled device, it a light-sensitive, discharging capacitors of 5 to 25 microns*

**Definition 4.0.4** (CMOS). *Complementary Metal Oxide Semiconductor. Has an active pixel senors and are cheaper, lower power but traditionally have lower quality.*

### Bayer Filter Array

A grid of different colour sensitive sensors arranged in such a way similar to our eyes where there are more green sensors than red or blue. This is because our eyes are more sensitive to green light.

## OpenCV

Its an open source, multi-language, multi-platform Computer Vision library that we are going to use in this course. It contains modules and algorithms like Canny edge detection for detecting edges, face tracking, etc

### Face Tracking

Before deep learning there was **SIFT** or (Scale Invariant Feature Transform) features. It detects objects despite changes to scale, noise, orientation, illumination. You can also use Deep Learning.

### Stereo Vision

Idealized camera geometry for stereo vision. We can take the disparity between two images to compute the depth of the object. It works because

1. Distance is inversely proportional to disparity. However, closer objects can be measured more accurately.
2. Disparity is proportional to b, horizontal distance between lenses. For a given disparity error, the accuracy of the depth estimate increases with increasing baseline b. However, as b increased, some objects may appear in one camera, but not in the other.
3. A point visible from both cameras produce a conjugate pair. The conjugate pairs lie on epipolar line (parallel to the x-axis for the arrangement in the figure)

# Chapter 5: Sensors

## Classification of Sensors

- Proprioceptive, measures values internally to the system, things like motor speed, wheel load, heading of the robot, battery status.
- Exteroceptive, information from the robot's environment, things like distances to object, intensity of the ambient light, unique features.
- Passive, energy coming from the environment.
- Active, emit their proper energy and measures the reaction, better performance, but some influence on environment.

## Characterizing Sensor Performance

Basic sensor response rating:

- Dynamic range - A ratio between the lower and upper limits, usually in decibels.
- Range - The upper limit
- Resolution - The minimum difference between two values, usually lower limit of dynamic range equal resolution. For digital sensors there is a analog-to-digital conversion
- Linearity - Variation of output signal as function of the input signal, could compensate if variations known.
- Bandwidth or Frequency - The speed with which a sensor can provide a stream of reading, usually there is an upper limit depending on the sensor and the sampling rate. Lower limit is also possible, e.g., acceleration sensors.

### In Situ Sensor Performance

Characteristics that are especially relevant for real world environment

- Sensitivity - Ratio of output changes to input changes, in real world, sensor often has sensitivity to other environmental changes like illumination.
- Cross-sensitivity - Sensitivity to environmental parameters that are orthogonal to the target parameters (e.e.,g compress responding to building material).
- Error / Accuracy - Difference between sensor's output and the true value.
- Systematic error - Deterministic errors caused by factors that can (in theory) be modelled and therefore be predictable.
- Random error - Non-deterministic error which cannot be predictable. However, they can be described probabilistically.
- Precision - Reproducibility of sensor results $prevision = \frac{range}{\sigma}$

## Multi-Modal Error Distributions

Behaviours of sensors modelled by probability distribution (random errors)

- Usually very little knowledge about causes of random errors
- Often probability distribution is assumed to be symmetric or even Gaussian
- However, these assumptions may be wrong
  - Sonar (Ultrasonic) sensor might overestimate the distance in real environment and is therefore not symmetric
  - Sonar sensor might be best modelled by two modes:
    1. The case that the signal returns directly
    2. The case that the signal returns after multi-path reflections
  - Stereo vision system might correlate to images incorrectly, thus causing results that make no sense at all.

## Wheel/Motor Encoder

Wheel or motor encoders measures the **position** or **speed** of the wheel. Integrating the wheel movements gets you an estimate of the robot's position and odometry. Optical encoders are proprioceptive sensors, position estimation in relation to a fixed reference frame is only valuable for short movements. Typical resolution of 2000 increments per revolution.

A simple rotating disk with a single laser shining through a code track will give you the speed but not the direction. To find the wheel direction you would need multiple lasers. At minimum you need at least two and using some phase comparisons you can find the direction that the wheel is spinning.

## Heading sensors

There are multiple kinds, proprioceptive like gyroscope, inclinometers or exteroceptive like a compass. These are used to determine the robot's orientation. Using the heading and velocity and integrating them over time gives you a position estimate (dead reckoning). Having the location and orientation gives you the **pose**.

### Compass

Ancient way of finding magnetic north, not feasible for indoor environments or any environment with a lot of magnetic interference.

### Gyrocompass

Patented in 1885 but the first practical use in 1906 in Germany. It finds true north as determined by Earth's rotation and it's not affected by a ship's composition, or magnetic interference.

### Gyroscope

Heading sensors that keeps the orientation to a fixed frame, i.e., absolute measure for the heading of mobile system. A mechanical gyroscope is a simple gyroscope, it has a drift of 0.1 degree in 6 hours. The spinning axis is aligned with north-south meridian, earth's rotation has no effect on the gyro's horizontal axis. If pointed east-west, the horizontal axis reads the earths rotation.

Optical gyroscopes has two beams in opposite directions around a circle. Their bandwidth is greater than 100 kHz and has a drift less than 0.0001 degrees/hr.

#### Mechanical Gyroscopes

The inertial properties of a fast-spinning rotor gives us gyroscopic precession, inertial navigation systems. The angular momentum associated with a spinning wheel keeps the axis of the gyroscope inertially stable. Reactive torque (tracking stability) proportional to spinning speed, precession speed and wheel's inertia. No torque can be transmitted from the outer pivot to the wheel axis, the spinning axis will therefore be stable. If the spinning axis is aligned with the north-south meridian then the earth's rotation has no effect on the gyro's horizontal axis. If it points east-west, then the horizontal axis reads the earth's rotation.

#### Rate Gyro

Same basic arrangement shown as regular mechanical gyros, but gimble(s) re strained by a torsional spring. It measure angular speeds instead of the orientations. Others, more simple gyroscopes, use Coriolis forces to measure changes in heading.

#### Optical Gyroscopes

Early 1980s, first installed in airplanes. Angular speed (heading) sensors using two monochromic light or laser beams from same source. One is travelling clockwise, the other counterclockwise. Laser beam travelling in direction of rotation. Slightly short path shows a higher frequency Difference in frequency $\Delta f$ of the two means is proportional to the angular velocity $\Omega$ of the cylinder. Solid-state optical gyroscopes based on the same principle. Much more accurate then a mechanical gyroscope.

## Ground-based Active and Passive Beacons

Beacons are signalling guiding devices with a precisely known positions. Beacon-based navigation is used single the humans started to travel.

- **Natural** beacons (landmarks) like stars, mountains, or the sun
- **Artificial** beacons like lighthouses

But GPS revolutionized modern navigation technology, this is a key sensor for outdoor mobile robotics. GPS is not applicable for indoor applications.

The major drawback with the use of beacons indoors are that it's expensive as you need to make environmental changes. It imposes additional limitations to its flexibility and adaptability to changing environments.

## GPS - Global Positioning System

Developed for military use, but now is commercial. Comprised of 24 satellites (including some spares). Orbits the earth every 12 hours at a height of 20,190 km. Location of GPS receivers determined through time-of-flight measurements.

Technical challenges include:

- **Time synchronization** between individual satellites and GPS receivers
- Real time update of the exact location of satellites.
- Precise measurement of the time of flight
- Interferences with other signal

### Time synchronization

Atomic clocks on each satellite, monitored from different ground stations. Electromagnetic radiation propagates at light speed around 0.3 meters per nanoseconds. Position accuracy proportional to precision of time measurement.

### Real time updates of exact location of satellites

Satellites are monitored from several widely distributed ground stations. A master station analyzes all measurements and transmit actual positions to each satellites

Position accuracy down to about 2 meters, probably less. Improvements include differential GPS and you can have the accuracy increase to 10 cm. But you need a fixed and known location.

### Indoor GPS

What about indoor GPS? One could use sounds and transmit them to the robot so they can do the triangulation that way. Another could be using Wi-Fi antenna and using RSSI, fingerprinting, angle of arrival, or even time of flight to do triangulation.

## Range Sensor

Used to measure large range distance. These range information is a key element for localization and

environmental modelling.

There are two kinds:

1. Ultrasonic sensors using sound
2. Laser sensors using EM waves

Regardless of the medium they use the propagation speed and the round-trip time of flight to determine distance. The propagation speed for sound is about 0.3 m/ms, while the propagation speed for the EM waves is about 0.3 m/ns or about a million times faster. Due to it's speed finding the ToF using EM waves is not an easy task. Laser range sensors are expensive and well as being delicate.

The quality of time-of-flight range sensor manly depend on:

- Uncertainties about the exact time of arrival of the reflected signal
- Inaccuracies in the ToF measure w.r.t. laser range sensors
- Opening angle of transmitted beam w.r.t. ultrasonic range sensors
- Interaction wit the target (surface, specular reflection)
- Variation of propagation speed
- Speed of mobile robot and target if it is not standing still

## Ultrasonic Sensor

Transit packet of ultrasonic pressure waves. Distance $d$ of the echoing object can be calculated based on the propagation speed of sound $c$ and the time of flight $t$. The speed of sound $c$ (340 m/s) at NTP. This value will change based on the air pressure and temperature.

Typical, the range of frequency is 40 - 180 kHz The generation of sound waves is generated by a piezoelectric transducer. The sound beam propagates in a cone-like manner with an opening angle of around 20-40 degrees. Soft surface like carpet will absorb most of the sound energy thus having very little coming back to the receiver. Therefore, it will overestimate the distance to a surface. Surfaces far from perpendicular to the direction of sound will also overestimate distances as the reflected waves aren't pointed back to the sensor due to specular reflection. We use Ultrasonic sensors because they are cheap and easy to work/program with.

## Laser Range Finder

Transmitted and received beam coaxial. The transmitter illuminates a target with a collocated beam. The receiver detects the time needed for the round-trip A mechanical mechanism with mirror sweeps across the environment getting 2 or 3D measurements.

Pulsed laser will send out pulses of light and measure the elapsed time of the pulse to get the distance directly. However, this is very difficult to measure.

Phase-Shift measurement is an easier methods that measures the phase shift of the reflected signal and determine the distance that way. However, because you can have multiple distances that will produce the same phase shift in the reflected signal the actual distance is a multiple of the computed distance.

### Triangulation Ranging

Geometric properties of image establish a distance measurement. Project a well-defined light pattern (something like a point or a line) onto the environment. The reflected light is captured by a photo-sensitive line or matrix (camera) sensor device and using triangulation establish a distance.

### Structured Light

Eliminates corresponding problem by projecting structured light on the scene. Slits of light or a laser is emitted by a rotating mirror. The light is perceived by the camera. The range to an illuminated point can then be determined from simple geometry.

## Doppler Effect

If the transmitted object is moving then the frequency of the transmitted object will be higher if the object is moving towards you, and lower if the object is moving away from you. The amount of shift is proportional the speed at which it is approaching or moving away from you.

# Chapter 6: Planning

# Chapter 7: Searching and Planning

Mobile robot path planning: identifying a trajectory that, when executed, will enable the robot to reach the goal location.

**Definition 7.0.1** (Representation). • *State (state space)*
- *Actions (operators)*
- *Initial and goal states*

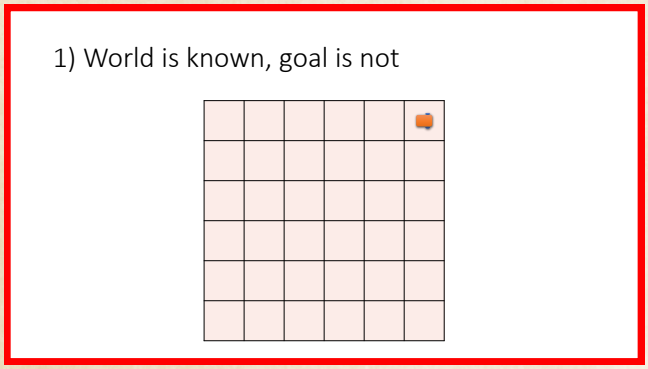
**Definition 7.0.2** (Plan). *Sequence of actions or state that achieve desired goal state*

- Model of the world
- Compute path
- Smooth it and satisfy differential constraints
- Design a trajectory (velocity function) along the path
- Design a feedback control law that tracks the trajectory
- execute

## Fundamental Questions

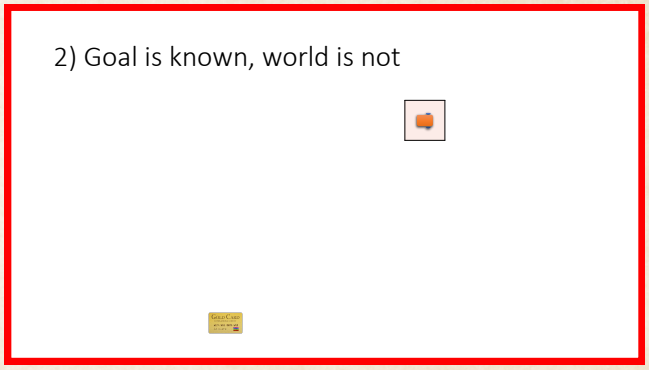
- How is the plan represented?
- How is a plan computed?
- What does the plan achieve?
- How do we evaluate a plan's quality?

### World is known, goal is not



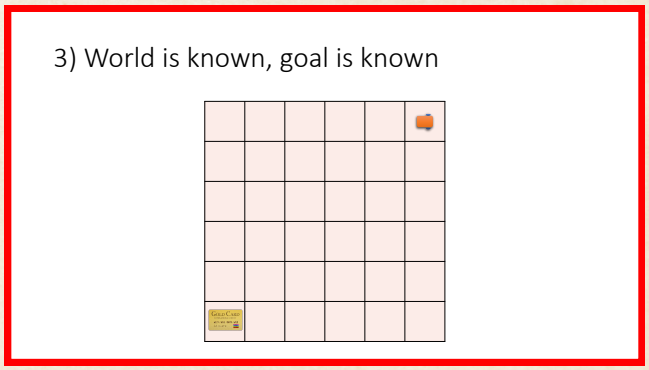

Here the entire world is known, but we don't know the goal, one could search all squares to find it but's that is not feasible sometimes.

### Goal is known, world is not



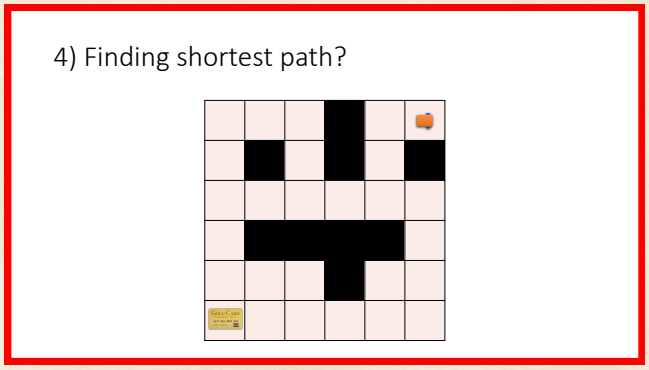

Here the goal is known, but the world is not. We need to drive around to learn about our environment so we can drive to the goal.

### World is known, goal is known




In this scenario both the world and the goal is known. Here we just plan to get the shortest route to get from our current position to our destination.

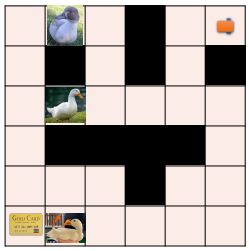
### Finding the Shortest Path




Same as before but with obstacles.

## Finding the shortest path with cost

5) Finding shortest path with costs



Same as before but different obstacles have different difficulties of traversal.

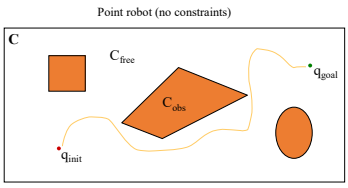
## The world consists of

**Definition 7.0.3** (Obstacles). *Places where the robot can't (shouldn't) go*

**Definition 7.0.4** (Free Space). *Unoccupied space within the world. Robot "might" be able to go here, there many be unknown obstacles, and the state may be unreachable.*

**Definition 7.0.5** (Configuration Space). *Is the space that contains all the free and constrained points that the robot can and cannot be.*

Configuration Space




For a point robot moving in 2-D plane, C-space is $R^2$

## Path Planning in Configuration Space

Typical simplifying assumptions for indoor mobile robots:

- 2 DOF for representation
- Robot is round, so that orientation doesn't matter
- Robot is holonomic, can move in any direction

# General Tree Search

## Depth-First Search

Expand deepest unexpanded node. Implementation: fringe = LIFO queue or a stack

## Breath First Search

Expand the shallowest node first. Uses a FIFO or a queue to keep track of which nodes to search first.

## Iterative Deepening

Iterative deepening uses DFS as a subroutine,

## Uniform Cost

Expand the cheapest node first, fringe is a priority queue. For graphs with actions of different cost, this is the general case of BFS if all costs are equal. Expand least "total cost" unexpaneded node

## Best First Search

## A*

### Admissible Heuristic

**A heuristic $h$ is *admissible***
if $h(n) \leq h^*(n)$ if

Creating admissible heuristic, most of the work in solving hard search problem optimally.

# Big Picture

Occupancy grids perform exhaustive search across the state space. It will represent and evaluate a far greater number of world states than the root would actually need to traverse.

What can we do differently? Make the grids larger, use line segments and coordinates instead of grid.

# Roadmap Algorithms

**Idea**

- Avoid searching the entire space
- Pre-compute a *hopefully) small graph like a roadmap such that staying on the "road" will avoid obstacles
- Find a path from $q_{\text{start}}$ to the closest point on roadmap
- Use the roadmap to find a path to $q_{\text{goal}}$

Sweep algorithm

visibility graph, gives the shortest path but:

- Stay close to the obstacle as close as possible

Voronoi Diagram: have the robot stay away from the obstacles as far as it can.

Difficult to compute in higher dimensions. Staying away from obstacles is not necessarily the best heuristic:

- Can lead to paths that are much too conservative
- Can be unstable: small changes in obstacles configuration can lead to large changes in the diagram.

The **Fortune's Algorithm** creates this voronoi diagram in $O(n \log(n))$ time where $n$ is the number of points.

Exact Cell Decomposition:

# Chapter 8: Localization

Two types of approaches:

- **Iconic**: use raw sensor data directly. Match current sensor readings with that was observed in the past
- **Feature-based**: extract features of the environment, such as corners and doorways. Match current observations

Localization

Two types of approaches:

- Iconic: use raw sensor data directly. Match current sensor readings with what was observed in the past
- Feature-based: extract features of the environment, such as corners and doorways. Match current observations

## Matching and Registration

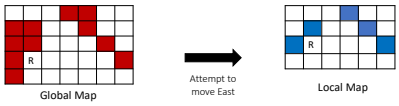
Collect **sensor** readings and create a local map. Estimate **poses** that the robot is likely to be in given distance travelled from last map update.

- In theory $k$ is infinite
- Discretize the space of possible positions (e.g., consider errors in increments of 5 degrees)
- Try to model the likely behaviour of your robot and try to account for systematic errors (e.g., robot tends to drive to one side)

For each pose $k$, **score** how well the local map matches the global map at this position. **Choose** the pose with the best score. Update the position of the robot to the corresponding $(x, y, \theta)$.

### Example

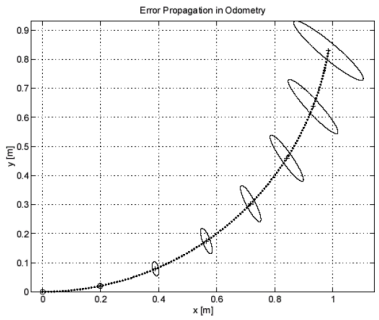
Example




Where is the robot?
What things would you need to know?

Now there will be some errors in the odometry and the odometry error will get worst the more complicated the kinematics becomes.

but now add in turning...




## Odometry and Mapping

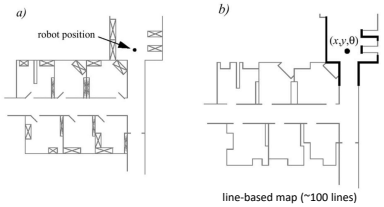
Creates a map as the robot is exploring the environment. Our odometry could be perfect, but there will be errors in our sensor measurements either caused by the environment or by the tolerances in the sensor itself. Therefore, we can't trust the odometry, and we must localize as we are mapping.

## Other representations

There are other ways that we can represent our map

- Line-based map
- Grid-based map
- Topological map

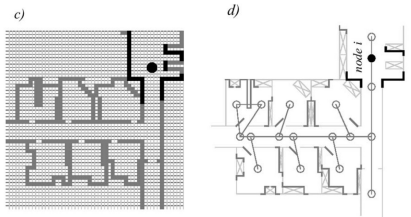
Other Representations




line-based map (~100 lines)

Representations

One location vs. location distribution




Grid-based map (3000 cells)

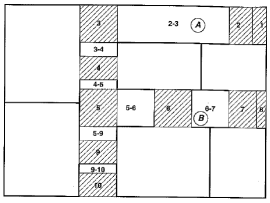
Topological map (50 features, 18 nodes)

## Feature-based Localization

Extract features such as doorways, corners, and intersections and either

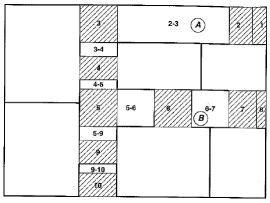
- Use continuous locaization to try and match features at each update
- Use Topological information to create a graph of the environment.

Topological Map of Office Building



The robot has identified 10 doorways, marked by single number
Hallways between doorways labeled by gateway-gateway pairing

Topological Map of Office Building

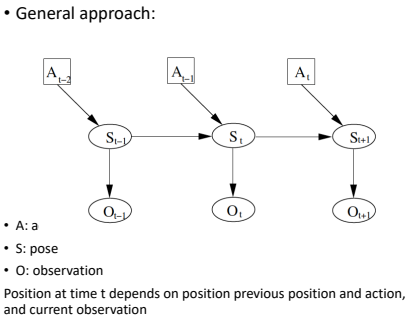


The robot has identified 10 doorways, marked by single number
Hallways between doorways labeled by gateway-gateway pairing
Robot is told it is at position A but it's actually at B --- how could it correct?

# Localization Problem(s)

- Position Tracking
- Global Localization
- Kidnapped Robot Problem
- Multi-robot Localization

## General approach




- A: Action
- S: Pose
- O: Observation

Position at time $t$ depends on position previous position and action, and current observation.

Pose at time $t$ determines the observation at time $t$. If we know the pose, we can say what the observation is. But, this is backward

# Bayes Filtering

If event a and b are independent, then $p(a, b) = p(a) \cdot p(b)$ If event a and b are not independent, then $p(a, b) = p(a) \cdot p(b \mid a) = p(b) \cdot p(a \mid b)$.

$$p(c \mid d) = \frac{p(d \mid c) \cdot p(c)}{p(d)}$$

## Modelling actions

$$P(x \mid u, x')$$

# Discrete Continuous

$$\mu' = \left( \frac{\mu r^2 + v\mu^2}{\mu^2 + r^2} \right)$$

$$\sigma^{2'} = \left( \frac{\sigma^2 r^2}{\sigma^2 + r^2} \right)$$

# Particles

```
1 def Gaussian(self, mu, sigma, x):
2   return exp(-((mu - x)**2) / (sigma**2) /
    ↪  2.0) ) / sqrt(2.0*pi*(sigma**2))
3
```

Listing 1: Particle Filtering Code

# Chapter 9: Neural Networks and Machine Learning

## Perceptron

**Perceptron** is a simple learning algorithm for. Originally introduced in the online learning scenario.

- Online learning model
- Perceptron algorithm
- Its guarantees a large margin

Example arrives **sequentially**. We need to make a prediction. Afterwards observe the outcome The goal is to minimize the number of mistakes. Analysis wise, make **no distributional** assumptions, consider a worst sequence of examples.

### Linear Separators

The Perceptron Algorithm

- Set $t = 1$ , start with all zero vector $w_1$.
- Give example $x$

**Definition 9.0.1** (Geometric Margin)**.** *The* ***margin*** *of example $x$ w.r.t. a linear separator $w$ is the distance from $x$ to the plane $w \cdot x = 0$ (or the negative if on the wrong side)*

**Definition 9.0.2** (Margin $\gamma_w$)**.** *of a set of all examples*

## Linear Regression

### Supervised Learning

Goal: Construct a predictor

**Definition 9.0.3** (Linear Regression)**.**

$$\hat{f}_n^L = argmin$$

## Machine Learning

# Credits

## General

- Created by: Jihoon Og, u/DnD_Notes
- Compiled on Wednesday 1st March, 2023 at 20:42
- Typesetting engine: LaTeX
- Dungeon and Dragon (5e) LaTeX Template
- CMPUT-503 lecture slides by Matthew E. Taylor

## Art

- Warforged for the cover art is from D&D Beyond
- Andy the D&D Ampersand is from Dungeon and Dragons
- Book logo is from Adobe Stock
- Cover art formatting and design done in Photoshop CC 2019

## Disclaimer