Università
degli Studi
di Padova

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

INFORMATION ENGINEERING DEPARTMENT

COMPUTER ENGINEERING - AI & ROBOTICS

Intelligent Robotics 2023/2024

Francesco Crisci 2076739

Introduction

Lecture 3

One possible exam question is the DOF of a synchro drive robot and why it is that. another question might be about robot exploration, mapping and localization. (Do not trust google to study or the exam).
Explain how tactile skin for a robot can be constructed, elements and how they are assembled. (This professor is basically an asshole about questions, so be as specific as possible and explain everything that is in the book and slides).
Some historical shit no-one cares about.

Automation and Autonomy

There is no intelligence without a body. we'll see it later on.
Autonomy means dealing with unknown problems. What is an intelligence robot? In Ai, a robot is intelligent if it can achieve is goal. AI typically is divided in 7 major area: knowledge representation, learning, learning, problem solving, natural language processing, inference, search and vision.
Autonomy: The quality or state of being self-governing, self-directing freedom and especially moral independence.We mean self-governing as the James Watt self-governor. It means taking action depending on the information from the outer world. Bounded rationality means that the capability of AI to have rational thinking is really limited. Strong AI can be applied to multiple domains, weak AI can be used to a single domain. Bounded rationality derives form the limits of cognitive agents. Robot cannot get drunk like us, so we win. Autonomy is the capability of generating action according to what happens in the environment. To take an action, or in general to be able to complete the goal, it is different from cognition. Autonomy is the capability of reasoning about a modelled event.
It is about a physically-situated agents who not only perform actions but can also adapt to the open world when the environment and tasks are not known a priori by generating new plans, monitoring and changing plans, and learning within the constraints of their bounded rationality.
Automation is a bout physically-situated tools performing highly repetitive, pre-planned actions for well-modeled tasks under the closed world assumption. It works in the close world assumption, i.e., we know every detail about that world.

Industrial robots

We have a mature technology, with strong basis of control theory, faster joint movement. They are high repeatable. Thou, they need fixturing. A fixture is a support, or reference, that you have to insert in the working space of the robot so that you ensure that an object can be manipulated.

a fixture keeps the item in the correct position at the right time. They are now adding sensors to reduce the fixtures, in order to add some flexibility in the case the fixture was not in the right place.

world assumption

The close world assumption implies tat everything in the robot environment is known or controlled. Everything is know a priori, this means you can use formal logic to. reason about the world, since we can substitute objects with symbols. Everything can be completely modeled and we can create exiting procedure if there are unexpected situations. We can also minimize or eliminate sensing. You can Engineer the world(He really likes this word so i'll use it in the exam), i.e, you change the world in order to simplify your problem/problem.

Lecture 4

We wan tot describe the difference between automation and autonomy; understand how this is difference has shaped different cultures within robotics; define...
Automation is usually related to industrial robots working engineered environments. Always repeating the same actions. There is some intelligence as sensors, segmentation, localizing algorithm and so on. we know everything about the world. For autonomy we usually can think about rovers, which they explore a non closed world, whee they do not know if the soil is solid or not, if there re martian, water and so on.We just know so many little things about this environment and the robot has to deal with the foreseen. An autonomous robot works in an open world. We cannot assume what is happening or what the robot encounters. These two assumption define close world and open world.

Close world

You know the environment, so you can create a sequence the action in which we can assume the success of all actions since we know what is gonna happen.. This means you can create a plan based on formal logic, based on sensors to use them as control signals or used them directly to trigger a machine. The focus is to create stable control loop to respond to all expected situation. This include also failures, not only the correct situations. This help in minimizing the sense of the robot, if not eliminating it. This assumption has many problems, like the toy problem called the monkey and the banana(frame problem, because you have your world and you know an area of your world, and you trying to model everything inside that frame, but you do not know what is happening outside of this frame). If we only manipulate symbols (mapping objects of the real worlds into symbols) we are missing the fact that there might be some configuration we aren't able to model or we didn't model. There are additional information that we cannot put in the modelling of the world. It comes from the fact that the real world is never a closed world. You never have the whole knowledge on it.

Open world

we do not make assumption that everything an be modelled, or that we can predict all the action. We assume from the beginning that we have a partial knowledge from the environment, and we assume that this knowledge is only partially correct. There might be some error, thing we cannot

perceive. We try to design our robot in order to be compliant to this world. We put enough sensing in our robot so that it can understand the world at the best. We need to have some system to dynamically to change the actions of the robot. We need t cope with actions and the world. You cannot model everything in the world. This is associated with being autonomous.
the difference between autonomy and autonomous affects the hardware design(think of vacuum cleaner and an industrial arm), the programming style and the kind of failures.

Why it matters?

It affects hardware design: autonomy requires rich sensing in order to monitor the key elements of the dynamic world, so a robot designed for automation or for teleoperation is not necessarily able to be used autonomously by adding software. Think of it as an ecology, what ecological niche is the robot going to fill.
It affects programming style: if you have a closed world, delegating for a small set of repetitious tasks, then we focus on formal, stable control loops. On the other hand, f you focus on open world, delegating for a variety of tasks while operating in dynamic environment, we focus to artificial intelligence. The first case is automation, the second is autonomy. For the first case we can think of military planes, such as drones and stuff. for the second case, you can think of robot puppies, which can do stupid things but are way cuter eheheh.
It affects how system breaks: which kind of failure you can expect from the system. a machine cannot substitute a person, because there are things that are not correctly modelled, and so if there is a person thinking about a situation he can recover from the unknown but a machine cannot. Thinking of devices that are controlled by humans and then you want a machine doing things by itself. This happened in the military, where they moved to almost autonomous flights or vehicles, controlled by remote operators.

Can you solve a problem with automation or autonomy?

Automation

- Execution: perhaps plan once, then repeat that plan forever
- deterministic: can model the system deterministically
- Closed world: the model contains everything
- Signals: control or decision making is at the signal level

Autonomy

- Generate: constantly generating new plans
- Non-deterministic: system is too complex to mode deterministically
- Open world: models will only be partial
- Symbols: control or decision-making is with symbols or labels.

The path to autonomy is never guaranteed, but sometimes it is necessary. There are some applications which mix the two of them, like which part of the problem can be solved with automation and which part with autonomy. The robot vacuum cleaner has the following slider: the plan slider

goes more towards the execution since the path is always the same and does not change through time. The action plans it goes more through the non-deterministic since it is always checking its own position. It continuously try to localize itself. The models goes towards more the close world assumption than through the open world. The knowledge representation goes towards the signals, but the most advanced ones use some symbols too.

Locomotion

There is no brain/intelligence without a body. You cannot have two different teams working separately on brain and body. It would not work.
Embodied intelligence: the intelligence is not only in the brain but is also in the body. You need some intelligence in the mechanical design.
When designing a robot you should put intelligence also in the body.

Locomotion from Class notes

We have 5 different type of locomotion:

- wheeled
- legged
- Snake
- Free-floating
- Swimming

Mobile robot kinematics is about models-maneuverability-motion control. We will focus on wheeled locomotion. Locomotion can be achieved with different mechanism and it involves different aspects: Actuators, properly designed, Forces control and the reactions to the forces, managed by control theory systems, dynamics, passive dynamics and animal locomotion to understand and get inspired for our robots. We can have hydraulic or pneumatic muscles for some types of robots to achieve movement. We need to consider the following aspects: Friction and direction, design and control, Actuation and sensing, animal and machine. In order to move in the environment we need Stability (number of contact points, center of gravity, static vs dynamic stabilization, inclination to terrain), Contact (contact point or area, angle of contact Friction) and Environment (structure and medium. You cannot project a swimming robot and put it in the desert.). A robot to be stable it need at least 3 contact points. The stability can be challenged from the environment, such as inclination of the terrain and so on. The number of contact points is not the only issue, we need to consider also the angle of contact of the contact points. There is an important table on the book, hopefully it is on the slides.(It is in. lesson 4).
The concept found in nature are difficulty to recreate artificially.

Legged locomotion

It is nature inspired. The movement of walking biped is close to rolling. The number of legs determines stability of locomotion. Bipedal locomotion is really hard to recreate, consider that a

human learns it in 2 years or so. Walking of a biped is not too far from real rolling. Rolling of a pol....
Legged systems can overcome many obstacles that are not reachable by wheeled systems. But it is quite hard to achieve this since many DOFs must be controlled in a coordinated way and the robot must see detailed elements of the terrain. The wheeled robots cannot overcome obstacles that have an height greater or equal than the radius of the wheel. Wheeled robots are worse than legged robots for stairs. Legged robots are more stable than wheeled ones. Wheeled have problems with slopes and discontinuities, while legged robots an overcome them. In some cases, like short discontinuities in the terrain, the legged robots can have some problems, like getting stuck or something. when designing, should we consider walking or rolling? You have to consider the following: How many actuators you need, the structural complexity, the control expense, the energy efficiency, thee terrain (flat ground, soft ground, climbing,...) and movement of the involved masses (walking/running includes up and down movement of COG(center of gravity); some extra losses). Crawling/sliding is requires more power compare to running, which is more consuming compared to walking, which is more consuming than railway wheels. But the top speed can be really different... Remember that when we are walking we are moving the center of mass, so we should consider exploiting this property. But how can we generate motions for walking? We have different ways:

- Manual motion design: classical slider based motion editors. You create different positions (like frames of an animation). We design different positions configurations and then we can achieve an almost fluent walk for the robot. Unfortunately, it takes a lot of time and it is trial and error procedure.
- Control based: ZMP(zero moments point: technique to make humanoid robots walking) and inverse pendulum based control.
- Learning algorithms: motion is specified in a parametric way(Central pattern generation); the parameters are determined by search algorithms(Genetic algorithms,...).
- Motions are obtained from human data: learning by watching/imitation; motion re-targeting. Usually the kinematic chain of a robot is different from the one of a human, that is why we use motion re-targeting, i.e., re-projecting the human kinematic chain(human motion) on the robot kinematic chain. It is used in game/movie industries; We use an actor and then a motion capture system to recreate some movements.

This is just about finding the position of the joints, we are not considering the dynamics and the structure of our robot, which is really important for the energy efficiency of our robots.
A passive walker is a robot with limbs that has no motors. You exploit gravity and structure of the legs that have springs in the structure which release energy during the walking. The energy comes from either the kinetic energy (like pushing the robot) converted in elastic energy either potential energy, like going downhill and it is transforming the potential energy in kinetic energy to move. Of course, it cannot move uphill. Asimo walks with the knees bent, which makes the leg extension process more easy since you do not move the center o gravity but you expend a lot of energy, so it is energy inefficient. We want to combine passive walking with dynamic stability. The study of passive walker....(Go to slides). We define a metric for the efficiency of a walker as

$$COT = c_{mt} = |mech.energy|/(weight \times distancetraveled)[JKg^{-1}km^{-1}]$$

a clever design o the legs can be really useful. Static walking can be represented as an inverse pendulum.

The fewer legs the more complicated becomes locomotion: stability, at least three legs are required for static stability: babies have to learn for qite a while until they are able to stand or even walk on their two legs. For static walking at least 6 legs are required. during walking some legs are lifted.

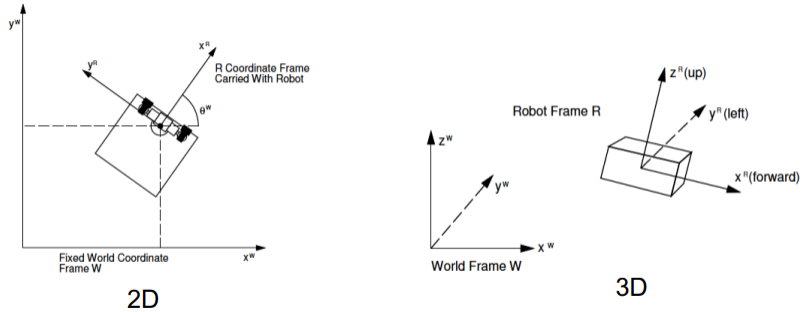
Wheeled locomotion

We have two types of wheel: Standard wheel with 2 DOF(It can go straight or i you have one wheel you can turn around in the same spot. It can turn around the point of contact. Think of an acrobat on the unicycle) or castor wheel with 3 DOF (It has two axis: one o spin the wheel and one to the point the wheel is attached on the chassis. We have one DOF for the spinning, one for rotating around the point of contact and one for the rotation about the contact point of the chassis). The standard wheel is fixed to the frame of the vehicle. Chassis of the ROBOT is the frame(or the box) composing the body of the robot and it is the structure to which we attach the wheel. We want the axe of the wheel to be fixed on the chassis of the robot. We have also other types of wheels: the Swedish wheel with 3 DOF(aka Meccanum wheel or Omnidirectional wheel. It can move also sideways. It can have rotation along the wheel axis, around the point of contact and it can move sideways. That is why it has 3 DOF. The first two are the same for the standard wheel) and spherical wheel with 3 DOF (You have a sphear incapsulated in a support attached to the chassis of the robot, like the one you have in old mouse for computers).

Locomotion again

It is the act of moving from place to place: we have seen synchro drive with 2 DOFs and omnidirectional robot with 3 DOFs. Synchro drive cannot generate torque. 4 wheel robot need a suspension system. We can also have 6 wheels arrangements. We can have platforms to help with the stability of the robot (platforms instead of wheels). Remember that a wheel, to overcome an obstacle, needs a radius that is bigger than the height of the obstacle itself. If you think of the Shrimp 3^{rd} passively climbs the obstacles that are higher than the wheel radius. It is a 6 wheel configuration robot. This structure is compliant(it can adapt) to the morphology of the terrain. Locomotion relies on the physical interaction between the vehicle and its environment. Locomotion is concerned with the interaction forces, along with the mechanism and actuators that generates them.

Motion and coordinate frames



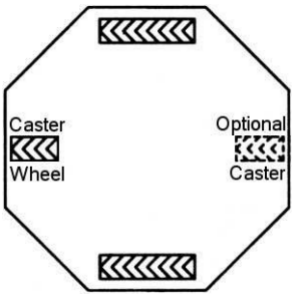
We can reduce the robot to a point and to a vector. Robot it's a vector i the 2D cartetian plane in the 2D case. We have 2 coordinate frames:

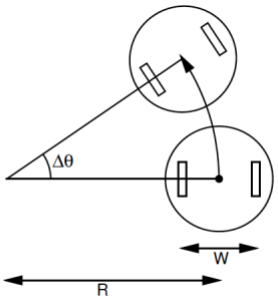
- world frame $W(x, y, \theta)$. We have 3 DOFs, aka 3 variables we cna change to manipulate the robot
- Robot's reference frame R, useful to move around and give them perception since they should be intelligent

In the 3D case we have the same things, W and R, but in 3D we have 6 DOFs of motion. In 3D perception and actuation are in the robot's reference frame. The goal is to find the correct transformation between W and R. If we have a robot with 1 DOF it means that it can move only in one direction; the robot can translate (or rotate) along 1 dimension(axle). We can think of a train as a 1 DOF machine. For 2 DOF we have a robot that can translate(or rotate) along 2 dimension; for example a telescope or the synchro drive robot.

Differential drive

Consider the following 2 figures: By selecting the speed of each wheel we can control the motion





of the robot: if the two wheels have the same speed, the robot will move straight; if the wheels have same speed but opposite direction, the robot will rotate on the spot; if one wheel goes slower than the other, the robot will turn on hte side of the slower wheel. You are setting a circular trajectory and an angular speed. LEt now consider the following:

- V_L and V_R as the speed of the left and right wheel respectively
- ω_L and ω_R as the angular speeds of the left and right wheels respectively
- W as the distance between the wheels
- L = R-W/2 and Ri= R+W/2 as the distances of the left and right wheels form the center of the trajectory respectively

We know that that $V_L = \omega_L r_l$ in case there is friction. In case of ice we have that the velocity and the radius are not zero, but the angular velocity of the wheel is zero. If the velocity of the left and right wheel are the same, we have straight motion; if they are the same but opposite in direction, the robot will turn on the spot. If they are different we'll have the following properties:

- Left wheel: in Δt the traveled distance is $V_L \Delta t$; the radius of arc is $R - W/2$
- Right wheel: in Δt the traveled distance is $V_R \Delta t$; the radius of arc is $R + W/2$

We then have: $\Delta\theta = \frac{V_L \Delta t}{R - W/2} = \frac{V_R \Delta t}{R + W/2} \Rightarrow \frac{W}{2}(V_L + V_R) = R(V_R - V_L)$. Then we have the following two equations:

$$R = \frac{W(V_R + V_L)}{2(V_R - V_L)} \qquad \Delta\theta = \frac{(V_R - V_L)\Delta t}{W}$$

Where W is a constant and Δt is the time elapsed. The angle is regulated by the difference of the two velocities. For the configurations we have seen the following will happen:

- if $V_R = V_L \Rightarrow \Delta\theta = 0, R = \infty$;
- if $V_R = -V_L, or, V_L = -V_L \Rightarrow \Delta\theta = \frac{2V_R \Delta t}{W}, R = 0$

Locomotion from Murphy

10 minutes on why you should take notes and why ghidoni is a chad.
For autonomous robots is important to move around. A mobile robot can move and sense, and must process information to link these two.Possible locomotion systems are speed and/or acceleration of movement; precision positioning,; flexibility and robustness in different conditions; efficiency. We need to be able to say if the motion of the robot has been accurate enough, since it might think it moved a meter but actually moved 80cm or 1.20m. You have a distribution of errors leading to an not precise positioning. We also want to cope with locomotion in different conditions, so we want it to be robust and flexible in different situations. We'd also like to program a robot that is efficient enough and with lower power consumption.
Locomotion is the act of moving from place to place. It relies on the physical interaction between the vehicle and its environment. It is concerned with the interaction forces, along with the mechanisms and actuators that generate them. A person has 3 degree of freedom: x,y and theta(the rotation angle to change the direction). To determine the position of a robot you need 3 degrees of freedom, x,y and the orientation angle. a robot can also have 3 degree of freedoms if they can move in the 3-dimensional space and can rotate on all 3 axes. So we have 3 DOF for traslation and 3 for rotation.

Mechanical locomotion

Many approaches i AI robotics assume that the robot is holonomic, which means that the device can be treated as a massless point capable of instantaneous turning in any direction. It allows the designer to ignore the complexity involved in mechanical control and we do not have to worry about the robot slowing down before making a turn or having parallel park. It simplifies the planning and localization. In reality robots are nonholonomic since robots have a mass, and no matter the design, there is always some skitter where the robot crabs to one side when it turns. Due to the fact that robots are nonhomonolic, the reactive or deliberative functions will need to know how fast the robot is moving and how far.

Steering

Ground robots are nonholonomic, so it is important to understand how they are steered. We have two designs that approximate holonomic turning for benign conditions:

- *Synchro-drive:* where the wheels must turn together, approximating turning in place, though there is some slippage of the wheels on the floor. The wheels are linked through either a drive belt or a set of gears. The wheels tend to be narrow to minimize contact with the ground and thus they skitter.
- *Omnidirectional wheels*, also called Mercanum wheels, are wheels with rollers that allow the robot to move perpendicular to the main wheel. By adjusting the relative speeds of the forward and lateral wheels, the robot can go sideways. Unfortunately, these wheels tend to be easily clogged with dirt and gravel in outdoor applications.

There are also two major styles of steering for nonholonomic vehicles, and these are use extensively in real-world applications. These two styles are:

- *Ackerman Steering* is how a car is steered; the wheels in front turn to guide the vehicle. Technically it refers to the mechanism that adjusts the wheels so that the wheel on the inside of the turn is rotated more than the wheel on the outside of the turn in order to compensate for slight difference between the inside and outside wheel diameters. It is truly nonholonomic, considering how hard it is to parallel park.
- *Skid steering*, also known as differential steering, is how a tank or bulldozer is steered. The tracks on each side can be controlled independently . If the robot is to turn left, the left track slows down while the right track speeds up. Skid steering is not restricted to tracks. It can be implemented on wheeled vehicles. It can allow the vehicle to approximate holonomicity but how closely depends on many variables, such as favorable surfaces and power of the platform.

Polymorphic Skid Systems

They are used prevalently in military robots. These robots have tracks that can change shape to adapt to terrain. Consider the following: the larger the area the tracks contact, the more traction and the more likely the track will be able to move and climb. However, the larger the contact area the more power it takes to turn as the robot pivots on the inner track and the inner track must overcome friction to act as a pivot. Thus it would be desirable to have a track system where the amount of contact area could be adjusted for the situation. Polymorphic skid systems allow tracks to have a variable geometry. One approach is to create a tracked system where the continuous track changes shape, from flat to triangular. The advantage is that it is highly adaptive and requires less power. The down side is that, as the track lift up to climb over a obstacle or turn, the lift creates a gap or opening. Object or debris may get into the gap causing the track to separate from the wheels moving the track, eventually disabling the robot. A second disadvantage is that the tracks tend to be flexible and loosely fitted so that they can adapt to different configurations, but the track may come off the rollers, or detrack.
Another approach to polymorphic skid-steering is to add tracked flippers. The robot relies on a pair of continuous flat tracks but can raise and lower tracked flippers to go over obstacles or gain greater traction or maintain balance. Other systems add flippers in the rear as well. Flippers eliminate the risk of an open track and enable the robot to right itself or crawl by using the flippers to push itself back over or pull itself forward.

Biomimetic Locomotion

The main concept is that there is a periodic, or repetitive, motion, either a vibration of the body or an oscillation of limbs. It can be divided into 5 main categories:

- *Crawling* occurs when the agent overcomes friction through longitudinal vibration or movement. An example in nature is a caterpillar which contracts along the length of its body to move forward, creating a slow sinusoidal wave of motion. The active Scope Camera (ASC)robot used to search the rubble at a building collapse in Jacksonville is an example of a crawling robot. Rather than duplicate the sinusoidal, the designers used vibrations to move the robot without complex mechanical deformation. The ASC is surrounded with a "skin" of VELCRO tilted at an angle similar to the "hairs" in a hairy caterpillar. As the robot vibrates throughout the axis, the VELCRO makes contact with surfaces and is thrusted slightly forward due to the angle of tilt. Crawling is the most energy-intensive category of movement, so it should be no surprise that the ASC is connected to a large power supply.

- *Sliding* occurs when the agent overcomes friction thorugh transverse vibrations or movements. An example in nature is a snake which moves in a sinusoidal side-to-side pattern using its scales to increase friction enabling pivoting. A snake robot uses transverse vibrations to move. It should be noted that most robots that are called "snakes" do not use transverse locomotion. They are actually replicating the flexibility of a snake or an elephant's trunk in order to move through tight spaces with high curvature or climb or grasp poles. The appeal of a snake or elephant trunk is that it is small but highly flexible, because it has many joints; in engineering this is called a hyper-redundant mechanism. The flexible robots often use a series of omnidirectional wheels to achieve the same mobility as a snake but without the frictional aspects.

- *Running* occurs when the agent overcomes kinetic energy with an oscillatory movement of a multi-link pendulum that leads to a predominately horizontal motion. A multi-link pendulum is an engineering representation of a leg and joints. Jumping is another way the agent can overcome kinetic energy with oscillatory movement of legs. In this case the oscillations produce a predominately vertical motion.

- *Walking*

- *Railway wheels*

Running, jumping and walking are integral to legged locomotion and form a separate field of study.

Legged Locomotion

Legs can provide versatile locomotion at the lower end energy cost for biological locomotion. In terms of AI robotics, legged locomotion has two components: computing the reference trajectory, or where the robot should go, and computing the gait(Andatura), or the specific type of oscillation to move along the reference trajectory. Computing the gait can be hard because there is a large set of possible motions for the legs, called a leg event, to create an oscillatory motion, and the oscillatory motions have to be adapted to the terrain so that the legs land on the right footfalls in order to maintain balance. Siegwart, Nourbakhsh and Scaramuzza note that the number of possible leg events, N, to achieve legged locomotion increases factorially. Given k as the number of legs, we have that N=(2k-1)! We use U and D to denote if a leg moves up or down respectively.

We use - if the leg does not move.The problem of control increases if the leg has joints. Now the control signal is not just up, down or not at all but it has to signal which parts, or linkages, of the leg move and by how much. The human legged is jointed such that there are 7 DOFs, or seven possible control directives to specify, just to move the leg to a desired footfall and this ignores placement of the foot and toes.
As a result of the high number of leg events, the approach is to bundle the movements into a small set of pre-computed coordinated movements called gaits.

Balance

An obvious challenge is to maintain balance. We can create static walkers,i.e., robots that are always statically balanced. The other is provide dynamic balance.

- *Static balance:* The concept of an agent is balanced while it walks relies on the notion of a support polygon. The support polygon is defined as the convex hull of the contact points with the ground. If the center of mass of the agent is in the support polygon, then the agent is statically balanced. For bipedal locomotion, the polygon degenerates to a line, requiring more precise poses in order for the agent to remain balanced. Static stability means when a robot is statically balance when they are moving and also when they are stationary. It means that at every moment i time, the robot is in static balance. It also means that the legs maintain balance without any passive correction, or that if the robot put down a set of legs, the legs were firmly planted on the ground and would not slip. Maintaining static stability while walking is called static walking. It is typically slow since speed is subordinated to safe movements.

- *Dynamic balance* is what most of the animals rely on. Consider normal human running: there is an instant in time when neither leg is on the ground as the human has pushed off of one foot and lifted it to start swinging it forward, but the other leg has not touched down yet. A challenge in dynamic balance is to make sure that when the agent lands on the forward leg, the leg do not slip out form under causing a fall, and the legs are positioned such that the agent can spring off of them. This is commonly modeled as placing the legs so that there is zero moment point on the leg. Think of the lg as an inverted pendulum where the leg can make contact with the ground within a range of angles. The zero momentum point is the angle where the horizontal forces of momentum and friction are balanced and thus the robot should not fall. The ZMP is used to compute where to place a leg, but it is also influenced by friction and the type of foot on the leg. Just like static walking, locomotion using ZMP principle to place a leg typically has to stomp to get the foot flat against the ground to ensure that it will not slip.

Gaits

Being able to compute the ZMP is one step in dynamic balance of legged locomotion, bet there is still the issue of how to reduce the number of legged events. A key concept i simplifying locomotion is organizing the oscillation of the legs into virtual gaits. The idea is that legs are collected into two sets, A and B, and all the legs in A move at the same time in the same way and then next all legs in B move in the same way at the same time. Doing this, we don't have a factorial computation but a fixed one, no matter how many legs an agent has. We have 3 basic gaits based on how legs are paired: Trot, where diagonal pairs of legs alternate moving; pace, where lateral pairs alternate moving; Bound, where front and back pairs alternate moving.

Legs with joints

Virtual gaits reduce the effort in planning the number of leg events fo dynamic balance because it reduces the number of legs to two virtual sets. However, gaits generally treat legs as inverted pendulum, ignoring the fact that legs often have joints. The joint in a leg give the agent a mechanism by which to spring up with kinetic energy and thus increase the range of motion and the ability to adapt to terrain. They also reintroduce computational complexity. Following the idea of virtual gaits, joints in a leg are generally moved according to a small set of oscillatory patterns, or macros. Whereas gaits have been studied for thousands of years in animal husbandry, the patterns for moving the joints on a single leg have not been well understood. There are two main approaches to determining the motion of legs with joints: use motion capture cameras to determine how a person or animal moves their legs for a particular situation and then duplicate that movement or develop a central pattern generators CPG for each joint.

CPG

In biology, there are neural structures called central pattern generators that produce oscillations which require synchronized movements. Once the CPG is activated, the oscillations occur without the need for sensing or other additional inputs. However, the default pattern can be modified, or tuned, with sensing. Ideally a CPG could be developed that creates the rolling polygon of leg motion up, forward, and down, and then the vertices on the polygon could be adjusted to adapt to sensed terrain. The most common implementation of a CPG is with a variant of the nonlinear Van der Pol equation. Motion capture and CPG are not mutually exclusive. We can use motion capture to document how a gait changes to adapt to a terrain which then can be incorporated as a tuning input to a CPG or a new CPG that is released as appropriate.

Action Selection

There exists a third way for locomotion: learning to assemble existing motor schemas so that an agent learns to locomotion. The learning approach reinforces the principles and merit of behaviors. Action selection connotes how an agent chooses which behavior to instantiate at a given time. Innate releasing mechanisms are technically a form of action selection, but those are pre-programmed and hardwired. The bigger question is how actions are selected and learned for new events.

Locomotion from Sigwart

Nature favors legged locomotion, since locomotion systems in nature must operate on rough an unstructured terrain.

Key issues for locomotion

Locomotion is the complement of manipulation. Locomotion and manipulation thus share the same core issues of stability, contact characteristics and environmental type:

- stability: number and geometry of contact points; center of gravity; static/dynamic stability; inclination of terrain
- characteristics of contact: contact point/path size and shape; angle of contact; friction
- type of environment: structure; medium.

Legged locomotion

It is characterize by a series of point contacts between the robot and the ground. Because only a set of point contacts is required, the quality of the ground between those points does not matter as long as the robot can maintain adequate ground clearance. A walking robot is capable of crossing a hole or chasm so long as its reach exceeds the width of the hole. A final advantage is the potential to manipulate objects i the environment with great skill. The disadvantages include power and mechanical complexity. The leg, must be capable of sustaining part of the robot's total weight, and in many robots it must be capable of lifting and lowering the robot. High maneuverability will only be achieved if the legs have a sufficient number of DOFs to impart forces in a number if different directions.
The higher the umber of legs, the higher the complexity of the active control to maintain balance. In contrast, a creature with three legs can exhibit a static, stable pose provided that it ca ensure that its center of gravity is within the tripod of ground contact. Static stability, demonstrated by a three-legged stool, means that balance is maintained with no need for motion. In order to achieve static walking, a robot must have at least four legs, moving one of it at a time.
Compared to human legs, which need 7 DOFs, a robot leg requires a minimum of 2 DOFs to move a leg forward by lifting the leg and swinging it forward. In recent robots, they added a fourth DOFs to the ankle for bipedal walking, which enables the robot to shift the resulting force vector of the ground contact by actuating the pose of the sole of the foot. In general, adding degrees of freedom to a robot leg increase the maneuverability of the robot, both augmenting the range of terrains on which it can travel and the ability of the robot to travel with a variety of gaits. The disadvantage, though, is energy, control and mass, further increasing power and load requirements on existing actuators.

Consideration of dynamics

The cost of transportation expresses how much energy a robot uses to travel a certain distance. To better compare differently sized systems, this value is usually normalized bu the robot's weight and expressed in J/(Nm).When a robot moves with constant speed on a level surface, its potential and kinetic energy remain constant. In theory, no physical work is necessary to keep it moving, which makes it possible to get from one place to another with zero cost of transportation. In reality, some energy is always dissipated, and robots have to be equipped with actuators and batteries to compensate for the losses. Friction is present in the joints of legged systems and energy is dissipated by the foot-ground interaction. However, these effects cannot explain why legged systems usually consume considerably more energy than their wheeled counterparts. The bulk part of energy loss actually originates in the fact that legs are not performing a continuous motion, but are periodically moving back and forth. Joints have to undergo alternating phases of acceleration and deceleration, and energy is irrecoverably lost in the process. Because of the segmented structure of the legs, it can even happen that energy that is fed into one joint is simultaneously dissipated by another joint without creating any net work at the feet. Therefore, actuators are working against each other.
A solution to this problem is a better exploitation of the dynamics of the mechanical structure. The natural oscillations of pendula and springs can automatically create the required periodic motions. During running, these inverted pendulum dynamics are additionally enhanced by springs, which store energy during the ground phase and allow the main body to take off for the subsequent flight phase. With this approach it is possible to build legged robots that do not have actuation of any kind. Such passive dynamic walkers walk down a shallow incline but, because no actuators

are present, no negative work is performed and energetic losses due to braking are eliminated.
When the locomotion speed changes, characteristic properties such as stride length or stride frequency change as well, and more and more actuator effort is needed to force the joints to follow their required trajectories.
Changing the gait allows us to use a different set of natural dynamics, which better matches the stride frequency and step length that are needed for higher velocities.

Wheeled mobile robots

The wheel has been by far the most popular locomotion mechanism in mobile robotics and in man-made vehicles in general. It can achieve good efficiency and it does so with relatively simple mechanical implementation. Due to the fact that wheeled robots are almost always designed so that all the wheels are in ground contact at all times, balance is not usually a research problem in the robot design. Three wheels are sufficient to guarantee stable balance but also two-wheels robot can be stable. When we have more than 3 wheels, a suspension system is required to allow all wheels to maintain ground contact when the robot encounters uneven terrain. For wheeled robots we care more about traction, stability, maneuverability and control than balance.
There are 4 major wheel classes, which differ in their kinematics. The standard wheel and the castor wheel have a primary axis of rotation and are thus highly directional .To move in different direction, the wheel must be steer first along a vertical axis. The key difference between the two is that the standard wheel can accomplish this steering with no side effects, since the center of rotation passes through the contact patch with the ground, whereas the castor wheel rotates around an offset axis, causing a force to be imparted to the robot chassis during steering.
The Swedish wheel and the spherical wheel are both designs that are less constrained by directionality than the conventional standard wheel. The Swedish wheel functions as a normal wheel but provides low resistance in another direction as well, sometimes perpendicular to the conventional direction, and sometimes at an intermediate angle. The small rollers attached around the circumference of the wheel are passive and the wheel's primary axis serves as the only actively powered joint. The key advantage of this design is that, although the wheel rotation is powered only along the one principal axis, the wheel can kinematically move with very little friction along many possible trajectories, not just forward and backwards.
The spherical wheel is a truly omnidirectional wheel, often designed so that it may be actively powered to spin along any direction. One mechanism for implementing this spherical design imitates the computer mouse, providing actively powered rollers that rest against the top surface of the sphere and impart rotational forces.
The choice of wheel types for mobile robots is strongly linked to the choice of wheel arrangement, or wheel geometry. Mobile robots are designed for applications in a wide variety of situations. The minimum number of wheels required for static stability is two. A two-wheeled differential-drive robot can achieve static stability if the center of mass is below the wheel axle. However, under ordinary circumstances, such a solution requires wheel diameters that are impractically large. Static stability requires a minimum of three wheels, with the additional caveat that the center of gravity must be contained within the triangle formed by the ground contact points of the wheels.
Some robots are omnidirectional, meaning that they can move at any time in any direction along the ground plane regardless the orientation of the robot around its vertical axis. This level of maneuverability requires Swedish or spherical wheels. In general, the ground clearance of robots with Swedish or spherical wheels is somewhat limited due to the mechanical constraints of constructing omnidirectional wheels. A interesting modern solution is the four-castor wheel configuration in which each castor wheel is actively steered and actively translated. In this configuration the robot

is truly omidirectional.
There is generally an inverse correlation between controllability and maneuverability. For example, the omnidirectional designs such as the four-castor wheel configuration require significant processing to convert desired rotational and translational velocities to individual wheel commands. Such omnidirectional designs often have greater DOFs at the wheel. These DOFs cause an accumulation of slippage, tend to reduce dead-reckoning accuracy and increase the design complexity. Controlling an omnidirectional robot for a specific direction of travel is also more difficult and often less accurate when compared to less maneuverable designs. In summary, there is no ideal drive configuration that simultaneously maximizes stability, maneuverability and controllability. Each mobile robot application places unique constraints on the robot design problem, and the designer's task is to choose the most appropriate drive configuration possible among this space of compromises.

Synchro drive

Is a popular arrangement of wheels in indoor mobile robot applications. There are three driven and steered wheels but only two motors are used in total. The one translation motor sets the speed of all three wheels together, and the one steering motor spins all the wheels together about each of their individual steering axes. Not that the wheels are being steered with respect to the robot chassis, and therefore there is no direct way of orienting the robot chassis. In fact, the chassis orientation does drift over time due to uneven tire slippage, causing rotational dead-reckoning error.
It is particularly advantageous in cases where omnidirectionality is sought. So long as each vertical steering axis is aligned with the contact path of each ire, the robot can always orient its wheels and move along a new trajectory without changing its footprint. Of course, if the robot chassis has directionality and the designers intended to reorient the chassis purposefully, then synchro drive is appropriate only when combined with an independently rotating turret that attaches to the wheel chassis. In terms of dead reckoning, synchro drive systems are generally superior to true omnidirectional configurations but inferior to differential-drive and Ackerman steering systems(AKA cars). There are two main reasons for this: the translation motor generally drives the three wheels using a single belts. Because of slop and backlash in the drive train, whenever the drive motor engages, the closest wheel begins spinning before the furthest wheel, causing small change in the orientation of the chassis. With additional changes in motor speed, these small angular shifts accumulate to create a large error in orientation during dead reckoning. Second, the mobile robot has no direct control over the orientation of the chassis. Depending on the orientation of the chassis, the wheel thrust can be highly asymmetric, with two wheels on one side and the third wheel alone. or symmetric, with one wheel on each side and one wheel straight ahead or behind. The asymmetric case result in a variety of errors when tire-ground slippage can occur, again causing errors in dead reckoning of robot orientation.

Omnidirectional drive

Omnidirectional robots are able to move in any direction (x, y, θ) at any time are also holonomic. They can be realized by using spherical, castor or Swedish wheels:

- **3 spherical wheels**: each wheels is actuated by one motor. In the design presented by the book, the spherical wheels are suspended by three contact points, two given by spherical bearings and one by a wheel connected to the motor axle. This concept provides excellent maneuverability and is simple in design. However, it is limited to flat surfaces and small loads, and it is quite difficult to find round wheels with high friction coefficients.

- **4 Swedish wheels(Carnegie Mellon Uranus):** The configuration in the book uses 4 Swedish 45-degree wheels, each driven by a separate motor. By varying the direction of the rotation and relative speed of the four wheels, the robot can be moved along any trajectory in the plane and can simultaneously spin around its vertical axis. This arrangement is not minimal in terms of control motors. Because there are only 3 DOFs in the plane, one can build a 3 wheel omnidirectional robot chassis using 3 Swedish 90-degrees wheels.
- **4 castor wheels and 8 motors(Nomad XR4000)**

Tracked slip/skid locomotion

In the previous configurations we have made the assumption that wheels are not allowed to skid against the surface. An alternative form of steering, termed slip/skid, may be used to reorient the robot by spinning wheels that are facing the same direction at different speeds or in opposite directions. Robots that make use of trad have much larger ground contact patches, and this can be significantly improve their maneuverability in loose terrain compared to conventional wheel designs. However, due to this large ground contact patch, changing the orientation of the robot usually requires a skidding turn, wherein a large portion of the track must slide against the terrain. The disadvantage of such configurations is coupled to the slip/skid steering. Because of the large amount of skidding during a turn, the exact center of rotation of the robot is hard to predict and the exact change in position and orientation is also subject to variations depending on the ground friction. Therefore, dead-reckoning on such robots is highly inaccurate. Furthermore, a slip/skid approach on a high-friction surface can quickly overcome the torque capabilities of the motors being used. In terms of power efficiency, this approach is reasonably efficient on loose terrain but extremely inefficient otherwise.

Walking wheels

Walking robots might offer the best maneuverability in rough terrain. However, they are inefficient on flat grounds and need sophisticated control. Hybrid solutions combining the adaptability of legs with the efficiency of the wheels, offer an interesting compromise.

Kinematics

Kinematics from notes

trycicles

The circular path of a car-like tricycle robot is described by the following equations

$$R = \frac{L}{tans}$$

$$R_d = \frac{L}{sins}$$

Where R_d is the radius of the path that the rear driving wheel moves, while R is the distance from the robot center and the center of the circular path described by the robot motion. While s is the steering angle, i.e., the angle between the robot main axis and the axis of the wheel. L is the distance of the steering wheel from the center of the robot. By changing the steering angle, we change the radius of the circular path. If s is 0, the robot moves in a straight line. If we think about a time Δt, the angle spanned by the robot $\Delta\theta$ is the following:

$$\Delta\theta = \frac{v\Delta t}{R_d} = \frac{v\Delta t sins}{L}$$

We obtain the following two equations.

$$R = \frac{L}{tans}, \Delta\theta = \frac{v\Delta t sins}{L}$$

by controlling the velocity and the steering angle we can control the motion of the robot on the plane.
In real life, we use electric motors, but they have low torque and high speed. So to augment the torque we use gears of different size. The point of doing so is that by applying the same speed to a different gear we can increase the torque generated by the electric motor. The relation that governs the gears is the following:

$$\omega_2 = \frac{r_1}{r_2}\omega_1$$

It derives from the fact that the two gears have the same speed. When a small gear drives a bigger gear, the second gear has an higher torque and lower angular velocity in proportion to the ratio of teeth. Gears can be chained together to achieve compound effect. The angular sped is not the only changing factor, also the torques changes with this mechanism:

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

which has the reciprocate constant factor compared to the angular speed equation.
The robot we deal with have no static reference from which we can derive everything as in the robot manipulator case. We talk about forward and inverse kinematics in the sense that by knowing some information about the state of the robot we can derive the final positing of the robot. For a mobile robot the position of the motors do not tell the whole story, because there is no direct way to measure the robot position in the environment instantaneously. We can integrate position overtime, we can estimate the robot pose by looking at how the wheels moved through time in order to find the position of the robot, but this is prone to inaccurate measurements, which accumulates and leads to a drift(not sure about this last word). Understanding mobile robot motion starts from understanding wheel's constraints placed on robot mobility. So, a manipulator, by having a fixed position compared to the global frame, in order to calculate the position of the end-effector is enough to know the structure of the robot and the position of the joints, which can be calculated by encoders placed inside the joints. In the case of our robot, it is not enough to have encoders in the wheels to determine the final position of the robot since we don't have a fixed transformation in the reference system. With non-holonomic systems it is not possible to calculate the final position by looking at the position of the encoders, but we need to look at the differential equations (aka speeds) to find the final position of the robot. The traveled distance in the space is not enough to determine the position of the robot.
We define forward kinematic as a transform from the joint space to the physical space while the inverse kinematic is a transformation from the physical space to the joint space. The latest is the one required for motion control cause most of the time i want to bring my robot to a certain configuration on the plane, so i want to actuate the motors to reach this position. since our systems are are subject to non-holonomic constraint we need to use inverse kinematics(differential kinematics) so it means that we need to deal with velocities, so how the position changes in time, and not the absolute change in position. So, to control our robot we use the block that is in SLIDE 12, LOOK AT THE PICTURE BITCH. Since the robot does have mass , it cannot change it speeds from 0 to v or viceversa, so the inverse kinematics does not take into account the mass of the robot itself(we assume that the robot has no mass, it's a cheat code or so). So, inverse kinematics does not take into account forces and accelerations. So dynamic is another story. We represent the robot in the wold reference frame and in the robot reference frame. We want to be able to express the robot position as $\psi = [\dot{x}\ \dot{y}\ \dot{\theta}]^T$, so we would like to have a mapping between the two frames $\dot{\psi}_R = R(\theta)\dot{\psi}_W$, where R is the rotational matrix(see Segwart notes). CASE OF ROBOT ALIGNED WITH Y, LOOK AT GIANNINI?S NOTES.
The goal of the kinematic model is to be able to determine the robot speed as a function of the speeds of the wheels,the angle of the steering(if there), and the speed at which we steer the wheel. We want to describe, more generally, the robot speed $\dot{\psi}$ as a function of wheel speed $\dot{\phi}_i$, where i notes the wheel, steering angles β_i, and steering speeds $\dot{\beta}_i$. All this variables are connected by the geometrical parameters of the robot chassis(the constants of our model).
When we speak of the forward kinematics we need the following:

$$\dot{\psi} = [\dot{x}\ \dot{y}\ \dot{\theta}] = f(\dot{\phi}_1, ..., \dot{\phi}_n, \beta_1, ..., \beta_m, \dot{\beta}_1, ..., \dot{\beta}_m)$$

The function f describes the geometrical meaning of the robot. The inverse kinematic instead is described as follows:

$$[\dot{\phi}_1, ..., \dot{\phi}_n, \beta_1, ..., \beta_m, \dot{\beta}_1, ..., \dot{\beta}_m] = g(\dot{x}, \dot{y}, \dot{\theta})$$

How can we write all this in the simple case of the differential drive robot? given l as the semi-distance between the wheels and r as the radius of the wheel and θ is the heading of the robot., we can write some fancy equations that are on the slide, but i'm to lazy to write(SLIDE 3).

Degree of manueverability

The degree of maneuverability it's the motion capabilities that our platform has. The are the motion variables that we can control in order to move our robot. To decide the next pose of our robot. In other words, they are the overall DOFs that the robot can manipulate. It is obtained by summing the degree of mobility and degree of steerability. The degree of mobility it's δ_m, while the degree of steerability it's δ_s. So $\delta_M = \delta_m + \delta_s$. The degree of mobility are the DOFs of the chassis that can be control by changing the speed of the wheels. The degrees of steerability are the DOFs allowed by the change in orientation of the steerable wheels. The latter is the particular change at any instant of a steerable wheel imposes a (different) kinematic constraint.
Some examples are:

- *Unicycle:* has 1 single fixed standard wheel. $\delta_s = 0, \delta_m = 1$, so $\delta_M = 1$
- *Differential drive:* has 2 fixed standard wheels with two different arrangements(wheels on different axis or wheels on the same axis). $\delta_s = 0, \delta_m = 2$, so $\delta_M = 2$ in the first case, $\delta_s = 0, \delta_m = 1$, so $\delta_M =$ in the second case.
- *Omnidirectional robot:* with 3 Swedish wheels, each one of them driven by a different motor, here $\delta_m = 3$
- *bicycle:* 1 fixed standard wheel and 1 steerable standard wheel has $\delta_s = 1, \delta_m = 1, \delta_M = 2$
- *Tricycle*: 1 steerable wheel and 2 fixed standard wheel has $\delta_s = 1, \delta_m = 2$(if the two fixed are connected by a rod)
- *two-steer vehicle:* 2 steerable wheels and 1 spherical wheel for support has $\delta_s = 2, \delta_m = 1$

Two robots with the same degree of maneuverability δ_M are not equal nor move in the same way. For any robot with $\delta_M = 2$ the ICR is always constrained to lie on a line. for any robot with $\delta_M = 3$, the ICR is not constrained and can be set anywhere on the plane. Castor wheels do not add any constraint in steerability. They are useless.
We have five basic types of 3 wheel configurations:(PICTURE on the slides)

- *Omnidirectional drive:* we have 3 sweedish wheels $\delta_s = 0, \delta_m = 3, \delta_M = 3$
- *Differential drive:* $\delta_s = 0, \delta_m = 2, \delta_M = 2$
- *Omnisteer:* 2 sweedish wheel plus a steerable $\delta_s = 1, \delta_m = 2, \delta_M = 3$
- *tricycle:* $\delta_s = 1, \delta_m = 1, \delta_M = 2$
- *two-steer drive:* two steerable wheel with one motors and one steerable wheel as support. It has $\delta_s = 2, \delta_m = 1, \delta_M = 3$

The synchro drive has one motor to steer the steering belt and one robot to power the wheels, so $\delta_m = 1, \delta_s = 1, \delta_M = 2$. It has 2 DOF in the environment (x,y) and not θ. The ICR it's always at infinity. In fact this robot it's never rotating but always traslating. This is another reason why this robot can never change its heading.
The DOFs of the motion of a robot is the robot's ability to achieve various poses in the environment or in its working space. (Pose its position plus orientation). For example the car has $\delta_M = 2$ but the car has 3 DOFs in the environment. The environment DOFs tells you how much constrained is the degrees of mobility of your robot.

$\delta_m \Rightarrow$ robot's independently achievable velocities, which is the number of dimensions of the velocity space, which is the differentiable degrees of freedom DDOF. If DDOF=3 the robot can manipulate the 3 variable for the pose(x,y,theta with the dot on top).
DOFs are the robot ability to achieve various poses. DDOFs are the robot's ability to achieve various velocities in the velocity space,i.e., the ability to achieve various trajectories. The ruleis $DDOF \leq \delta_m \leq DOF$. Only robot with zero non-holonomic constraints are omnidirectional robot, and they have DDOF=DOF.

Mobile robot workspace

We need to introduce some concepts first:

- **Position:** the location of the robot in the 3D space; for example in the 2D space, with a mobile robot on a plane, you would use x and y to determine the position of the robot. In the case of a flying robot you use x,y and z;
- **Pose:** It captures both the position and the orientation of the robot. It is the value of the state variables of the robot in the space of its DOFs. In the case of a mobile robot on the plane we have that the pose is represented by x, y, θ. In the case of a flying robot we would have 6 variables to. determine the pose.
- **Path:** It's the sequence of poses in the space of the robot's DOFs.
- **Trajectory:** when the robot was in a specific position. It adds time to the path. It is the path + the time. You add another variable, which is the time stamp at which the robot was in that position.

The robots can have same paths but different trajectories.

Open loop control

A kinematic controller can compute the motion commands for a robot platform to follow a given trajectory described by poses vs. time. The trajectory is divided i motion segments of clearly defined shape: straight lines and segments of a circle. The control problem is. to pre-compute a smooth trajectory based on line and circle segments using the kinematics of the mobile platform. The disadvantages: are the following:

- it is not at all an easy task to pre-compute a feasible trajectory
- limitations and constraints of the robot's velocities and accelerations
- does not adapt or correct the trajectory if dynamical changes in the environment occur
- the resulting trajectories are usually not smooth
- the real position of the robot is not fed back to the controller.

Closed loop control

A feedback controller is the solution to control a mobile platform to move from point A to point B. The robot's path-planning is reducd to setting intermediate positions laying on the requested path. We need to find a control matrix K, if exists:

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \end{bmatrix} k_{ij} = k(t, e)$$

such that the control of v(t) and $\omega(t)$ brings the error to zero: e(t).

$$\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = Ke = K \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

$$\lim_{t \to \infty} e(t) = 0$$

To move the robot from one place to the other we need the Trajectory following(and obstacle Avoidance), using sensors and robot's current positions as inputs of the module. Before following the trajectory we need a higher level module to plan a trajectory, the Trajectory Planning module, which takes as input the map of the environment and the goal position, and the current position. The trajectory planning module has a lower frequency, while the trajectory following module has a higher frequency. On the slides there is a cute image of the trajectory following control loop. To lazy to add it now.

Kinematics from Segwart

We need a clear mapping between global and local reference frames, since the forces and constraints of each wheel must be expressed with respect to a clear and consistent reference frame.
Throughout the analysis we model the robot as a rigid body on wheels, operating on a horizontal plane. The total dimensionality of this robot chassis on the plane is 3, two for position in the plane and one for orientation along the vertical axis, which is orthogonal to the plane. Of course, there are additional DOFs and flexibility due to the wheel axles, wheel steering joints and wheel castor joints. However, by robot chassis we refer only to the rigid body of the robot, ignoring the joints and DOFs internal to the robot and its wheels.
In order to specify the position of the robot on the plane we establish a relationship between the global reference frame of the plane and the local reference frame of the robot. The axes X_I, Y_I define an arbitrarily inertial basis on the plane as the global reference frame from some origin $O : \{X_I, Y_I\}$. To specify the position of the robot, choose a point P on the robot chassis as its position reference point. The basis $\{X_R, Y_R\}$ defines two axes relative to P on the robot chassis and is thus the robot's local reference frame. The position of P in the global reference frame is specified by coordinates x and y, and the angular difference between the global and local reference frames is given by θ. We can describe the pose of the robot as a vector with these elements. Note the use of the subscript I to clarify the basis of this pose as the global reference frame:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

To describe robot motion in terms of component motions, it will be necessary to map motion along the axes of the global reference frame to motion along the axes of the robot's local reference frame.

Of course, the mapping function is a function of the current pose of the robot. This mapping is accomplished using the orthogonal rotation matrix:

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

This matrix can be used to map motion in the global reference frame $\{X_i, Y_i\}$ to motion in terms of the local reference frame $\{X_R, Y_R\}$. This operation is denoted by $R(\theta)\xi_I$ because the computation of this operation depens on the value of θ:

$$\dot{\xi}_R = R(\frac{\pi}{2})\dot{\xi}_i$$

Forward kinematics model

In the simplest case, the mapping showed previously is sufficient to generate a formula that captures the forward kinematics of the mobile robot.
Now, consider a differential drive robot with two wheels, each with diameter r. Given a point P centered between the two drive wheels, each wheel is a distance l from P. Given r,l, θ, and the spinning speed of each wheel $\dot{\phi}_1$ and $\dot{\phi}_2$, a forward kinematic model would predict the robot's overall speed in the global reference frame:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \dot{\phi}_1, \dot{\phi}_2)$$

We know that we can compute the robot's motion in the global reference frame from motion in its local reference frame: $\dot{\xi}_I = R(\theta)^{-1}\dot{\xi}_R$. Therefore, the strategy will be to first compute the contribution of each of the two wheels in the local reference frame, ξ_R. For this example of a differential-drive chassis, this problem is particularly straightforward.
Suppose that the robot's local reference frame is aligned such that the robot moves forward along $+X_R$. First consider the contribution of each wheel's spinning speed to the translation speed at P in the direction of $+X_R$. If one of the wheel spins while the other wheel contributes nothing and is stationary, since P is halfway between the two wheels, it will move instantaneously with half the speed: $\dot{x_{r_1}} = (1/2)r\dot{\phi}_1$ and $\dot{x_{r_2}} = (1/2)r\dot{\phi}_2$. In a differential drive robot, these two contributions can simply be added to calculate x_R component of $\dot{\xi}_R$.
Consider now a differential drive robot in which each wheel spins with equal speed but in opposite direction. The result is a stationary spinning robot. As expected, $\dot{x_R}$ will be zero as $\dot{y_R}$. Finally, we must compute the rotational component $\dot{\theta_R}$ of $\dot{\xi_R}$. Once again, the contributions of each wheel can be computed independently and just adds. Consider the right wheel(wheel 1.) Forward spin of this wheel results in counterclockwise rotation at point P. The rotational velocity ω_1 at P can be computed because the wheel is instantaneously moving along the arc of a circle of radius 2l: $\omega_1 = \frac{r\dot{\phi}_1}{2l}$. The same calculation applies to wheel 2, with the exception that forward spin results in clockwise rotation at point P: $\omega_2 = \frac{-r\dot{\phi}_2}{2l}$. Combining these two individual formulas yields a kinematic model for the differential-drive example robot:

$$\dot{\xi}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\phi}_1}{2} + \frac{r\dot{\phi}_2}{2} \\ 0 \\ \frac{r\dot{\phi}_1}{2l} + \frac{-r\dot{\phi}_2}{2l} \end{bmatrix}$$

We can now use this kinematic model in an example. However, we must first compute $R(\theta)^{-1}$. In general, calculating the inverse of a matrix may be challenging. In this case, however, it is easy because it is simply a transform form $\dot{\xi_R}$ to $\dot{\xi_I}$ rather than vice versa:

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

Wheel kinematic constraints

We assume that the plane of the wheel always remains vertical and that there is in all cases one single point of contact between the wheel and the ground plane. We assume that there is no sliding at this single point of contact. That is, the wheel undergoes only under conditions of pure rolling and rotation about the vertical axis through the contact point. We have two constraints for every wheel type: the first one is that the wheel must roll when motion takes place in the appropriate direction, the second is that the wheel must not slide orthogonal on the wheel plane.

fixed standard wheel

It has no vertical axis of rotation for steering, so its angle to the chassis is thus fixed, and it is limited to motion back and forth along the wheel plane and rotation around its contact point with the ground plane. Let us consider the following case, where the fixed standard wheel A is expressed in polar coordinates by distance l and angle α. The angle of the wheel plane relative to the chassis is denote by β, which is fixed since the standard wheel is not steerable. The wheel, which has radius r, can spin over time, and so its rotational position around its horizontal axle is a function of time t: $\phi(t)$. The rolling constraint for this wheel enforces that all motion along the direction of the wheel plane must be accompanied by the appropriate amount of wheel spin so that there is pure rolling at the contact point:

$$[sin(\alpha+\beta) - cos(\alpha+\beta)(-l)cos\beta]R(\theta)\dot{\xi}_I - r\dot{\phi} = 0$$

The firs term of the sum denotes the total motion along the wheel plane. The three elements of the vector on the left represent mapping from each $\dot{x}, \dot{y}, \dot{\theta}$ to their contributions for motion along the wheel plane.The sliding constraint for this wheel enforces that the component of the wheel's motion orthogonal to the wheel plane must be zero:

$$[cos(\alpha+\beta)sin(\alpha+\beta)lsin\beta]R(\theta)\dot{\xi}_I = 0$$

Steered standard wheel

They differ from the fixed standard wheel only in that there is an additional degree of freedom: the wheel may rotate around a vertical axis passing through the center of the wheel and the ground contact point. The equations of position for the steered standard wheel are identical to that of the fixed standard wheel with one exception: the orientation of the wheel to the robot chassis is no longer a single fixed value ,β, but instead varies as a function of time: $\beta(t)$. The rolling and sliding constraints are:

$$[sin(\alpha+\beta) - cos(\alpha-\beta)(-l)cos\beta]R(\theta)\dot{\xi}_I - r\dot{\phi} = 0$$

$$[cos(\alpha+\beta)sin(\alpha+\beta)lsin\beta]R(\theta)\dot{\xi}_I = 0$$

These constraints are identical to those of the fixed standard wheel because, unlike $\dot{\phi}$, $\dot{\beta}$ does not have a direct impact on the instantaneous motion constraints of a robot.

Castor wheels

They are able to steer around a vertical axis, however, unlike the steered standard wheel, the vertical axis of rotation in a castor wheel does not pass through the ground contact point. The wheel contact point is now at position B, which is connected by a rigid rod AB of fixed length d to point A fixes the location of the vertical axis about which B steers, and this point A has a position specified in the robot's reference framed. We assume that the plane of the wheel is aligned with AB at all times. The castor wheel has two parameters that vary as a function of time. $\phi(t)$ represents the wheel spin over time as before. $\beta(t)$ denotes the steering angle and orientation of AB over time. The rolling constraint is identical to the one of the steered standard wheel since the offset axis plays no roll during motion that is aligned with the wheel plane:

$$[sin(\alpha+\beta) - cos(\alpha+\beta)(-l)cos\beta]R(\theta)\dot{\xi}_I - r\dot{\phi} = 0$$

The castor geometry does have significant impact on the sliding constraint. The critical issue is that the lateral force on the wheel occurs at point A because this is the attachment point of the wheel to the chassis. Because of the offset ground contact point relative to A, the constraint that there be zero lateral movement would be wrong. Instead, the constraint is much like a rolling constraint, in that appropriate rotation of the vertical axis must take place:

$$[cos(\alpha+\beta)sin(\alpha+\beta)d + lsin\beta]R(\theta)\dot{\xi}_I + d\dot{\beta} = 0$$

In this last constraint, any motion orthogonal to the wheel plane must be balanced by an equivalent and opposite amount of castor steering motion. This result is critical to the success of castor wheels because by setting the value of $\dot{\beta}$ any arbitrary lateral motion can be acceptable. In a steered standard wheel, the steering action does not by itself cause a movement of the robot chassis. But in a castor wheel the steering action itself moves the robot chassis because of the offset between the ground contact point and the vertical axis of rotation. More concisely, given any robot chassis motion $\dot{\xi}_I$, there exists some value for spin speed $\dot{\phi}$ and steering speed $\dot{\beta}$ such that the constraints are met. Therefore, a robot with only castor wheels can move with any velocity in the space of possible robot motions. We term such systems *omnidirectional*.

Swedish wheels

They have o vertical axis of rotation but they are still able to move omnidirectionally like castor wheels. Swedish wheels consist of a fixed standard wheel with rollers attached to the wheel perimeter with axes that are antiparallel to the main axis of the fixed wheel component. The exact angle γ between the roller aces and the main axis can vary. The pose of a Swedish wheel is expressed exactly as in a fixed standard wheel with the addition of a term γ representing the angle between the main wheel plane and the axis of rotation of the small circumferential rollers. The motion constraint is the following:

$$[sin(\alpha+\beta+\gamma) - cos(\alpha+\beta+\gamma)(-l)cos(\beta+\gamma)]R(\theta)\dot{\xi}_I - r\dot{\phi}cos\gamma = 0$$

Orthogonal to this direction the motion is not constrained because of the free rotation $\dot{\phi}_{sw}$ of the small rollers.

$$[cos(\alpha+\beta+\gamma)sin(\alpha+\beta+\gamma)lcos(\beta+\gamma)]R(\theta)\dot{\xi}_I - r\dot{\phi}sin\gamma - r_{sw}\dot{\phi}_{sw} = 0$$

The behavior of this constraint changes dramatically as the value γ varies.

Spherical wheels

It do not place direct constraints on motion. Such a mechanism has no principal axis of rotation, and therefore no appropriate rolling or sliding constraints exist. As with castor wheels and Swedish wheels, the spherical wheel is clearly omnidirectional and places no constraints on the robot chassis kinematics. Therefore, the following equation simply describes the roll rate of the ball in the direction of motion v_A of point A of the robot:

$$[sin(\alpha+\beta) - cos(\alpha+\beta)(-l)cos\beta]R(\theta)\dot{\xi}_I - r\dot{\phi} = 0$$

By definition the wheel rotation orthogonal to this direction is zero.

$$[cos(\alpha+\beta)sin(\alpha+\beta)lsin\beta]R(\theta)\dot{\xi}_I = 0$$

This equations are the same as the one for the standard fixed wheel.

Robot kinematic constraints

Given a mobile robot with M wheels we can now compute the kinematic constraints of the robot chassis. The key idea is that each wheel imposes zero or more constraints on the robot motion, and so the process is simply one of appropriately combining all of the kinematic constraints arising from all of the wheels based on the placement of whose wheels on the robot chassis.
Only fixed standard wheels and steerable standard wheels have impact on robot chassis kinematics and therefore require consideration when computing the robot's kinematic constraints. Suppose that the robot has a total of N standard wheels, comprising N_f fixed standard wheels and N_s steerable standard wheels. We use $\beta_s(t)$ to denote the variable steering angles of the N_s, while β_f to refer to the orientation of the N_f fixed standard wheels. In the case of wheel spin, both the fixed and steerable wheels have rotational positions around the horizontal axle that vary as a function of time. We denote the fixed and steerable cases separately as $\phi_f(t)$ and ϕ_s(t), and use $\phi(t)$ as an aggregate matrix that combines both values: $\phi(t) = \begin{bmatrix}\phi_f(t)\\ \phi_s(t)\end{bmatrix}$. The rolling constraints of all wheels can now be collected in a single expression:

$$J_1(\beta_s)R(\theta)\dot{\xi}_I - J_2\dot{\phi} = 0$$

J_2 is q constant diagonal NxN matrix whose entries are radii r of all standard wheels. $J_1(\beta_s)$ denotes a matrix with projections for all wheels to their motions along their individual wheel planes: $J_1(\beta_s) = \begin{bmatrix}J_{1f}\\ J_{1s}(\beta_s)\end{bmatrix}$. Note that $J_1(\beta_s)$ is only a function of β_s and not β_f. This is because the orientations of steerable standard wheels vary as a function of time, whereas the orientations of fixed standard wheels are constant. J_{1f} is therefore a constant matrix of projections for all fixed standard wheels.
We use the same technique to collect the sliding constraints of all standard wheels into a single expression:

$$C_1(\beta_s)R(\theta)\dot{\xi}_I = 0 \quad (3.26)$$

$$C_1(\beta_s) = \begin{bmatrix}C_{1f}\\ C_{1s}(\beta_s)\end{bmatrix}$$

This sliding constraint over all standard wheels has the most significant impact on defining the overall maneuverability of the robot chassis.

Maneuverability

The kinematic mobility of a robot chassis is its ability to directly move in the environment. The basic constraint limiting mobility is the rule that every wheel must satisfy its sliding constraint. In addition to instantaneous kinematic notion, a mobile robot is able to further manipulate its position by steering steerable wheels.

Equation 3.26 imposes the constraint that every wheel must avoid any lateral slip. Of course this holds separately for each and every wheel, and so it is possible to specify this constraint separately for fixed and for steerable standard wheels:

$$C_{1f}R(\theta)\dot{\xi}_I = 0$$

$$C_{1s}(\beta_s)R(\theta)\dot{\xi}_I = 0$$

For both of these constraints to be satisfied, the motion vector $R(\theta)\dot{\xi}_I$ must belong to the null space of the projection matrix $C_1(\beta_s)$, which is simply a combination of C_{1f} and C_{1s}. Mathematically, the null space of $C_1(\beta_s)$ is the space N such that for any vector n in N, $C_1(\beta_s)n = 0$. If the kinematic constraints are to be honored, then the motion of the robot must always be within this space N. Consider a single standard wheel. It is forced by the sliding constraint to have zero lateral motion. This can be shown geometrically by drawing a zero motion line through its horizontal axis, perpendicular to the wheel plane. At any given instant, wheel motion along the zero motion line must be zero. In other words, the wheel must be moving instantaneously along some circle of radius R such that the center og that circle is located on the zero motion line. This center point, called the instantaneous center of rotation, may lie anywhere along the zero motion line. When R is at infinity, the wheel moves in a straight line.

Robot chassis kinematics is a function of the set of independent constraints arising from all standard wheels. The mathematical interpretation of independence is related to the rank of a matrix. Recall that the rank of a matrix is the smallest number of independent rows or columns. Equation 3.26 represents all sliding constraints imposed by the wheels of the mobile robot. Therefore $rank[C_1(\beta_s)]$ is the number of independent constraints. The greater the number of independent constraints, the more constrained is the mobility of the robot. For example, in the case of a robot with a single fixed standard wheel we'll have the following relation:

$$C_1(\beta_s) = C_{1f} = [cos(\alpha+\beta)sin(\alpha+\beta)lsin\beta]$$

Now let us add additional fixed standard wheel to create a differential-drive robot by constraining the second wheel to be aligned with the same horizontal axis as the original wheel. Without loss of generality, we can place point P at the midpoint between the centers of the two wheels. Given α_1, β_1, l_1 for wheel w_1 and α_2, β_2, l_2 for wheel w_2, it hods geometrically that $\{l_1 = l_2), (\beta_1 = \beta_2 = 0), (\alpha_1 + \pi = \alpha_2)\}$. Therefore matrix $C_1(\beta_s)$ has two constraints but a rank of one:

$$C_1(\beta_s) = C_{1f} = \begin{bmatrix} cos(\alpha_1) & sin(\alpha_1) & 0 \\ cos(\alpha_1+\pi) & sin(\alpha_1+\pi) & 0 \end{bmatrix}$$

In general, if $rank[C_{1f}] > 1$ then the vehicle can only travel along a circle or along a straight line. This configuration means that the robot has two or more independent constraints due to fixed standard wheels that do not share the same horizontal axis of rotation.

In general, a robot will have zero or more fixed standard wheels and zero or more steerable standard wheels. We can therefore identify the possible range of rank values for any robot: $0 \leq rank[C_1(\beta_s)] \leq 3$. In case the rank is 0, there are neither fixed nor steerable standard wheels

attached tot he robot frame: $N_f = N_s = 0$. In case the rank is 3, the kinematic constraints are specified along 3 DOFs, therefore there cannot be more than 3 independent constraints since the robot is completely constrained in all directions and is degenerate since motion in the plane is totally impossible. Now we can formally define a robot's *degree of mobility* δ_m:

$$\delta_m = dim N[C_1(\beta_s)] = 3 - rank[C_1(\beta_s)]$$

The dimensionality of the null space of matrix $C_1(\beta_s)$ is a measure of the number of DOFs of the robot chassis that can be immediately manipulated through changes in wheel velocity. It is logical that δ_m must range between 0 and 3.

Degree of steerability

The degree of mobility defined above quantifies the degrees of controllable freedom based on changes to wheel velocity. Steering can also have an eventual impact on a robot chassis pose ξ, although the impact is indirect because after changing the angle of a steerable standard wheel, the robot must move for the change in steering angle to have impact on the pose. As with mobility, we care about the umber of independently controllable steering parameters when defining the *degree of steerability* δ_s:

$$\delta_s = rank[C_{1s}(\beta_s)]$$

Recall that in the case of mobility, an increase in the rank implied more kinematic constraints and thus a less mobile system. In the case of steerability, an increase in the rank implies more degrees of steering freedom and thus greater eventual maneuverability. Since $C_1(\beta_s)$ includes $C_{1s}(\beta_s)$, this means that a steered standard wheel can both decrease mobility and increase steerability. The range of δ_s can be specified: $0 \leq \delta_s \leq 2$. If it is 0, then the robot has no steerable standard wheels and $N_s = 0$. The case where is 1 is most common when a robot configuration includes one or more steerable standard wheels. In the case it is 2 occurs only when the robots has no fixed standard wheels: $N_f = 0$.

Robot maneuverability

The overall DOFs that a robot can manipulate, called the *degree of maneuverability* δ_M can be readily defined in terms of mobility and steerability:

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

Therefore maneuverability includes both the DOFs that the robot manipulates directly through wheel velocity and the DOFs that it indirectly manipulates by changing the steering configuration and moving. Note that two robots with the same δ_M are not necessarily equivalent. For example differential drive and tricycle geometry have equal $\delta_M = 2$. In the first case we have $\delta_m = 2$ and $\delta_s = 0$, while in the latter we have $\delta_m = 1$ and $\delta_s = 1$. For a robot with $\delta_M = 2$ the inner circle radius is always constrained to lie on a line, while for any robot with $\delta_M = 3$ it can be set on any point on the plane.

Mobile robot workspace

For a robot, maneuverability is equivalent to its control degrees of freedom. But the robot is situated in some environment, and the next question is to situate our analysis in the environment. Identifying a robot's space of possible configurations is important because surprisingly it can exceed δ_M. In addition to workspace, we care about how the robot is able to move between various configurations.

Degrees of freedom

In defining the workspace of a robot, it is useful to first examine its admissible velocity space. Give the kinematic constraints of the robot, its velocity space describes the independent components of robot motion that the robot can control. The number of dimensions in the velocity space of a robot is the number of independently achievable velocities. This is also called the *differentaible degrees of freedom*. A robot's DDOF is always equal to its degree of mobility δ_m. Clearly there is an inequality relation at work: $DDOF \leq \delta_M \leq DOF$.

Holonomic robots

The term holonomic in robotics, refers specifically to the kinematic constraints of the robot chassis. A holonomic robot is a robot thst has zero nonholonomic constraints of the robot chassis. Conversely, a nonholonomic robot is a robot with one or more nonholonomic kinematic constraints. A holonomic kinematic constraint can be expressed as an explicit function of position variables only. A nonholonomic kinematic constraint requires a differential relationship, such as the derivative of the position varaibles only. Because of this latter pov, nonholonomic systems are often called nonintegrable systems. Consider the fixed standard wheel sliding constraint:

$$[cos(\alpha + \beta)\ sin(\alpha + \beta)\ lsin\beta]R(\theta)\dot{\xi}_I = 0$$

This constraint must use robot motion $\dot{\xi}$ rather than pose ξ because the point is to constraint robot motion perpendicular to the wheel plane to be zero. The constraint is not integrable, depending explicitly on robot motion. Therefore, the sliding constraint is a nonholonomic constraint. The first type of holonomic robot is a robot where constraints do exist but are all holonomic kinematic constraints $\delta_M < 3$. The second type of holonomic robot exists when there are no kinematic constraint, that is $N_f = 0$ and $N_s = 0$. Since there are no kinematic constraints, there are also no nonholonomic kinematic constraints and so such robot is always holonomic.
An alternative way to describe holonomic robot is based on the relationship between the differential DOFs of a robot and the DOFs of its workspace: a robot is holonomic if and only if DDOF=DOF.

kinematic control

The objective of a kinematic controller is to follow a trajectory described by its position or velocity profile as a function of time. This is often done by dividing the trajectory in motion segments of clearly defined shape. The control problem is thus to precompute a smooth trajectory based on line and circle segments which drives the robot from the initial position to the final position. This approach can be regarded as open-loop motion control, because the measured robot position is not fed back for velocity or position control. It has several disadvantages:

- it is not at all an easy task to precompute a feasible trajectory if all limitations and constraints of the robot's velocities and accelerations have to be considered.
- The robot will not automatically adapt or correct the trajectory if dynamic changes of environment occur.
- The resulting trajectories are usually not smooth, because the transitions from on trajectory segment to another are, for most of the commonly used segments, not smooth. This means there is a discontinuity in the robot's acceleration.

Feedback control

A more appropriate in motion control of a mobile robot is reduced to setting intermediate positions lying on the requested path. Consider an arbitrary position and orientation of the robot and a predefined goal position and orientation. The actual pose error vector given in the robot reference frame $\{X_R, Y_R, \theta\}$ is $e =^R [x, y, \theta]^T$ with x,y, θ being the goal coordinates of the robot. The task of the controller layout is to find a control matrix K if it exists:

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \end{bmatrix} \text{ with } k_{ij} = k(t, e)$$

such that the control of v(t) and $\omega(t)$ can be expressed $\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = Ke = K^R \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$ drives the error e towards zero.

kinematic model

We assume that the goal is at the origin of the inertial frame. In the following the position vector $[x, y, \theta]^T$ is always represented in the inertial frame. The kinematics of a differential-drive mobile robot described in the inertial frame $\{X_I, Y_I, \theta\}$ is given by:

$$^I \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

where $\dot{x}, \dot{y}$ are the linear velocities in the direction of the X_I and Y_I of the inertial frame. Let α denote the angle between x_R axis of the robot's reference frame and the vector $\hat{x}$ connecting the center of the axle of the wheels with the final position. If $\alpha \in I_1$, where $I_1 = [-\pi/2, \pi/2]$, then consider the coordinate transformation into polar coordinates with its origin at the goal position:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2}, \alpha = -\theta + atan2(\Delta y, \Delta x), \beta = -\theta - \alpha$$

This yields a system description, in the new polar coordinates, using a matrix equation

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -cos\alpha & 0 \\ \frac{sin\alpha}{\rho} & -1 \\ \frac{-sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

where ρ is the distance between the center of the robot's wheel axle and the goal position, θ denotes the angle between the X_R axis of the robot reference frame, and the X_I axis associated with the final position v and ω are the tangent and angular velocities respectively. On the other hand, if $\alpha \in I_2$, where $I_2 = (-\pi, -\pi/2] \cup (\pi/2, \pi]$ redefining the forward direction of the robot by setting $v = -v$, we obtain a system described by a matrix equation of the form

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} cos\alpha & 0 \\ \frac{-sin\alpha}{\rho} & 1 \\ \frac{sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

The control law

The control signals v and ω must now be designed to drive the robot from its actually configuration to the goal position. It is obvious that there is a discontinuity for $\rho = 0$; thus the theorem of Brockett does not obstruct smooth stabilizability. If we consider now the linear control law:

$$v = k_\rho \rho, \omega = k_\alpha \alpha + k_\beta \beta$$

we get a closed-loop system described by

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho cos\alpha \\ k_\rho sin\alpha - k_\alpha \alpha - k_\beta \beta \\ -k_\rho sin\alpha \end{bmatrix}$$

This system does not have a singularity in $\rho = 0$ and has a unique equilibrium point at $(\rho, \alpha, \beta) = (0, 0, 0)$. Thus it will drive the robot to this point, which is the goal position.

Local stability issue

It can further be shown, that the closed-loop control system is locally exponentially stable if $k_\rho > 0; k_\beta < 0; k_\alpha - k_\rho > 0$. (I'm skipping the proof cause it is tedious, but it is a page 103(physically 88) of the book)

Architectures

Notes from classes

Ai has converged on a canonical operational architecture after years of implementation: historical development is of interest because it helps identify legacy code and to see barriers that led to the convergence. Operational architectures have advantages such as: The programmer is forced to apply the software engineering principles of abstraction and modularity; checklist of completeness; design specification and it can also reveal vulnerabilities.
Overall style of design or organization is called software architecture, which describes a set of architectural components and how they interact. It also provides a principled way of organizing a control system. However, in addition to providing structure, it imposes constraints on the way the control problem can be solved[Mataric].
We have different types of architectures:

- Operational architecture: What the system does at a high level and NOT HOW it does it
- Systems architecture: Describes how a system works in terms of major subsystems
- Technical architecture: It describes how a system works in terms of implementation details (even prescribing the programming language).

The last two blocks focuses on implementation ad software. We worry so much about organization because the 7 distinct areas of Artificial Intelligence have to be knitted together somehow. Software engineering is necessary for a successful software enterprise. Thinking about architectures is good software engineering due to the following aspects:

- **Abstraction:** Ignores details to permit focus for thinking about general organization of intelligence and is a semi-formal description
- **Modularity:** It gives as high cohesion, low coupling and supports unit testing and debugging
- **Anticipation of change, Incrementally:** How to adapt, support evolution
- **Generality:** Not re-invent the wheel each time.

The canonical robot's architecture is the following: Planner, sequencer and Reactive (or behavioral) Layer. The upper brain or cortex is used to reason over symbols about goals; the middle brain is converting sensor data into symbols; The spiral cord and lower brain are used for skills and responses. Unfortunately, in animals there is an emotional intelligence too. We can use the following to cope witht that:

- user interfaces: Display, transparency of what robot is doing and thinking; natural language and gestures.

- Working in teams: Explicit multi-agent coordination often relies on social rules
- Persona we present to others: Security as what can you at your security grade can see about me; Human-robot interaction as in affective responses and natural language.

The extended canonical architecture is composed of the following 4 elements: Interaction (Persona and teaming, Deliberative loop (Reasoning over symbols), Key function (converting sensor data into symbols, Reactive loop (skill and responses). The hard part is the following: form reactive to deliberate we have two types of perception (Direct and Recognition, which impacts computer vision), and we have two different time horizons: form present to present, past and future; impact sensing, storage, as wells as algorithm reasoning, projecting. It also needs a central structure to hold the symbols, history, knowledge but is tractable. From Reactive/Deliberate to interaction we need additional knowledge about "theory of mind"-beliefs,desires,intentions of the other agent and common ground.

Making things more tangible

The primitives for robot intelligence are the following: Sense, Plan, Act and Learn. The problem is that not everything intelligent requires deciding. Behavioral robotics was born with sets of SENSE-ACT couplings called behaviors that get turned on/off based on situations. There is no Plan. Hybrid architecture have can Pan and then Sense-Act, or Plan and then instantiate appropriate Sense-Act behaviors, until next step in plan. Plan requires a world model plus the actual planning algorithm. Control theory is lower level but doesn't necessarily capture it all. Reactive is tightly coupled with sensing, so very fast; many concurrent stimulus-response behaviors, strung together with simple scripting with FSA. the action is generated by sensed or internal stimulus. There is no awareness, no mission monitoring. Models are of the vehicle, not the larger world.
For each layer we have different programming languages: for the interaction Layer we need Procedural, Functional, Ontological languages such as OWL; for the Deliberative Layer we need Functional languages such as LISP; while for the behavioral layer we need Procedural languages such as C, C++ or Java.

Primitives for robot intelligence

We have three primitives that we can identify for our robot: SENSE, PLAN, ACT. The book introduced another primitive, which is the LEARN. Intelligent systems in nature, like animals or simple form of lives can learn from the environment and create some memory of what happened before in order to increase the survival of themselves. How do you implement or arrange these primitives? We have three different paradigms for organising intelligence: Hierarchical, Reactive or Deliberative/reactive hybrid. These days we might have also deep learning robotics. We have two ways to describe them through the relationships between the three primitives of robotics (SENSE,PLAN,ACT) and how sensory data is managed and distributed in the system.

Hierarchical robotics

The first solution was the classical AI approach aka the hierarchical approach: SENSE - PLAN - ACT and then start over again. We have a sequence of 3 primitives. The classical planning involves predicate Logic, logical actions and planning algorithms. We also use task policies and we integrate task and motion planning. What should we put into each of this modules? for SENSE we need sensors and feature extractions; for PLAN we need to combine features into model, plan

tasking and task execution; for ACT we need motor control and actuators. The hierarchical model has a centralized sensing. The world model is a fused global data structure. It combines a priori representation, sensed info and cognitive understanding. The SENSE takes in input sensor data and gives in output sensed information. The PLAN takes in input information (sensed and/or cognitive) and gives in output directives. The ACT takes in input sensed information or directives and gives in output the actuator commands.We. talked about Shakey(Hope it is not in the exam). While hierarchies have advantages, relying on a world model creates problems. we have problems on processing, particularly for control. The alternative is to create layers or hierarchies within the world model to match other subsystems. Another problem is that the world model requires extensive representation which leads to two major problems: it operates under the closed world assumption and frame problem. In practice, the implementations are planning-centric. The programmer must implement: a representation of the model of the world, a differentiable with operators, preconditions and postconditions, and different evaluation function. Strips(used for shakey) suffered from frame problem and assumes a closed world. But remember, not everything intelligent requires "deciding"(reflexes or muscle memory such as riding a bike or driving a car).

Reactive robotics

it raises from two reasons: dissatisfaction with the results of the hierarchical paradigm and the influence of ideas from Winer's ethology and cybernetics. We have direct relation between SENSE and ACT. We only consider the reactive intelligence of humans. Users loved it because it worked. AI people loved it but wanted to put PLAN back in. Control people hated it because couldn't rigorously prove it worked. The reactive approach didn't give a fixed models or a mathematical model to apply for different tasks. Cybernetics is a combination of control theory, information theory and biology. It tries to explain the principles of control in both animals and machines. Uses the math of feedback control systems to express natural behaviors. The emphasis is on the strong coupling between the organism and its environment. Wiener is the initiator of cybernetics. Now we are talking about turtle. The main behaviors of turtle are Thrifty, Exploration, Attraction, Aversion and Recharging. Complex behaviors arise form the combination of these simple actions.
Emerging intelligence: the capability to exhibit a new behavior that was not coded bu it arose as activation or combination of singular, simpler behaviors.
We can classify three broad categories of behaviors: Reflexive behaviors(stimulus-response. Usually hard-wired for fast response), Reactive behaviors (learned. Usually compiled down to be executed without conscious thought), Conscious behavior (requires deliberative thought).

Hybrid architectures

Neither completely deliberative nor completely reactive approaches are suitable for building agents: researchers concluded using hybrid systems, which attempt to combine hierarchical and reactive approaches. An obvious approach is to build agents out of tow (or more) subsystems:

- a deliberative one,containing a symbolic world model, which develops plans and makes decisions in the way proposed by symbolic AI;
- a reactive one, which is capable of reacting to events without complex reasoning

The combination of reactive and proactive behaviors leads to a class of architectures in which the various subsystems are arranged into a hierarchy of interacting layers. Hybrid architectures aims to mix the two approaches in order to get an overall better performance. The critical point is the choice of the degree of balancing. According to the environment and the task you have to

decide how much software you want to dedicate to reactivity and to deliberation. We want to Plan then Sense-Act. Plan requires a world model plus the actual planning algorithms. Reactive layer requires distributed sensing and reactive functions operating at different time scale from deliberative layer. The world model goes under the open world assumption in this case.
The planner(upper level) i for mission generation and monitoring. It needs past, present and future.
The sequencing(lower level) is a selection of behaviors to accomplish task and local monitoring. It need present and past.

Notes from Murphy

The term architecture is frequently used in robotics to refer to the overall style of design or organization. Architectures distill prior experience into templates for creating new intelligent robots. We have 3 types of architectures:

- *Operational architecture:* describes what the system does, or its functionality, at a high level, but not how it does it. An operational architecture specifies how the seven distinct areas of artificial intelligence, each with their own algorithms and data structures, are knitted together and can be used to determine if a design provides the intended functionality.

- *Systems architecture:* describes how a system is decomposed into major subsystems. The system architecture states the specific software systems that supply the desired intelligent functionality and describes how they are connected. It can be used to determine if a design meets good software engineering principles, especially modularity and extensibility.

- *Technical architecture:* describes how the implementation details of the system. It specifies the algorithms and even the languages that the modules are programmed in and it can be used to determine if a design is using the most appropriate algorithms. They are in constant evolution, reflecting advances in AI.

We want to use the four general principles of software engineering:

- **Abstraction**, where the operational and systems architecture portals allow designers to ignore details in order to focus on thinking about general organization of intelligence.

- **Modularity**, where systems and technical architectures encourage the designer to think in terms of object-oriented programming. Modules should have high cohesion, where each module or object does one thing well and unrelated functions are put elsewhere, and low coupling, where modules or objects are independent. High cohesion and low coupling support unit testing and debugging.

- **Anticipation of change with incrementality**, as the systems and technical architectures provide the designer with insights whether the code is engineered to support upgrading and algorithm with a newer one and adding new capabilities without requiring the designer to rewrite the code for existing modules and possibility to introduce bugs.

- **Generality**. The operational, systems and technical architectures provide the designer with frameworks to determine if the basic organization, subsystems and implementation will allow the code to be used for other applications. In robotics, this principle is often called portability.

In addition to the four general principles of software engineering, Arkin adds two more software engineering principles for AI robotics: niche targetability and robustness. The first one captures how well the robot works for the independent application, while robustness identifies where the system is vulnerable, and how the system design intrinsically reduces that vulnerability.

Canonical AI Robotics Operational Architecture

Organizationally, the operational architecture consists of three layers, reactive, deliberative and interactive. Biological intelligence can be thought of as consisting of four major functions, reaction, deliberation, conversion of signals into symbols and interaction of the robot with other external agents. This abstraction of biological intelligence leads to three distinct purpose and styles of computing. The 3 layers not only encapsulate philosophically different categories of functionality, they have 5 different attributes which influence computing: primitives, perceptual ability, planning horizon, time scale and use of models.

Primitives: Similar to function in the LOA and ACL architectures, AI robotics views autonomous capabilities as consisting of four primitives: SENSE, PLAN,ACT, and, LEARN. If a function is taking information from the robot's sensors and producing an output useful by other functions, then that function is into the SENSE category. If a function is taking in information and producing one or more tasks for the robot to perform, that function is in the PLAN category. Functions which produce output commands to motor actuators fall into ACT. Intelligence is generally viewe as a process of SENSE, PLAN and ACT. LEARN is an important mechanism by which an agent maximizes its chances for success, but it transcends the other 3 primitives; an agent can learn to sense or plan better, acquire more acts or skills, or even learn for a specific mission what to sense, what to plan for and how to act.

Perpetual ability: AI robotics view the perception needed for a particular capability as being either direct or requiring recognition. This divides capabilities into those that can work directly with incoming stimuli and those requiring a conversion of the stimulus into a symbol.

Planning horizon: AI robotics views a capability as also having a planning horizon of either present;past,present; or future,past,present. The planning horizon constrains the choice of data structures, as well algorithms.

Time-scale: AI robotics also considers the time-scale of a function,i.e., whether it need to be very fast, fast, or can be relatively slow. Time-scale helps determine asynchronous operation of the software components.

Reactive(behavioral) layer

Reactive functionality corresponds to the reactive loop in the central nervous system. This loop is associated with functionality that occurs in the spinal cord and lower brain, especially responses and skills based on motor memory. The responses and skills are patterns of action generally referred to as behaviors. Behavioral functionality is sufficient for animal intelligence. In robotics, it consists of functionality constructed from the SENSE and ACT primitives with no PLAN. SENSE and ACT are tightly coupled into constructs called behaviors which produce capabilities. The perceptual ability of the behaviors is called direct perception,. While a behavior does not create and maintain a global world model, a behavior may have a local world model, which serves as short term memory. There is no planning, and thus planning horizon is the present. The time-scale for the functions is very fast. Stimulus-response behaviors typically have a fast update cycle on the order of 15-30 cycles per second, matching control need and sensor update rates.

Deliberative Layer

Deliberative functionality corresponds to the cognitive loop associated with te cortex in the brain. The brain independently takes as input the same signals used by other ventral nervous system and adds additional sensor processing to make conscious decisions. The cognitive loop can modify the reactive loop, either instantiate new actions that would be carried out by reactive functionality or to modify them. Deliberative functionality is sufficient for the more sophisticated problem solving and reasoning aspects of intelligence as well as planning. The two loops differ in processing speed and the content of what is being processed. Reflexes and motor skills are very fast, while deliberating about a problem may be slow. The deliberative layer hosts the PLAN activities of the robot, has a Planner that generates a plan using a World Model and the instantiates appropriate SENSE-ACT behaviors in the reactive layer to execute the plan. These behaviors run until the next step of the plan is reached and a new set of behaviors is instantiated or the deliberative monitoring detects a problem with the plan. The perceptual ability of the behaviors is recognition, where the robot builds up global or persistent world models. The planing horizon uses information from the past and the present to project consequences of a plan into the future. The time it takes in the deliberative layer may update with a frequency ranging from on the order of 15 cycles per second to update the World model to several minutes to construct complex plans. The deliberative layer is divided into 2 sub-layers connected by the world model that contain the four deliberative functions:

- *Generating plans* which corresponds to planning, reasoning, and problem solving in AI
- *Selecting* specific resources to accomplish the plan, which corresponds to planning, resource allocation, and knowledge representation of capabilities in AI
- *Implementing* the plan, which corresponds to execution
- *Monitoring* the execution of the plan to determine if it is meeting the goal, learning what is normal, and anticipating potential failures, which corresponds to planning and reasoning in AI.

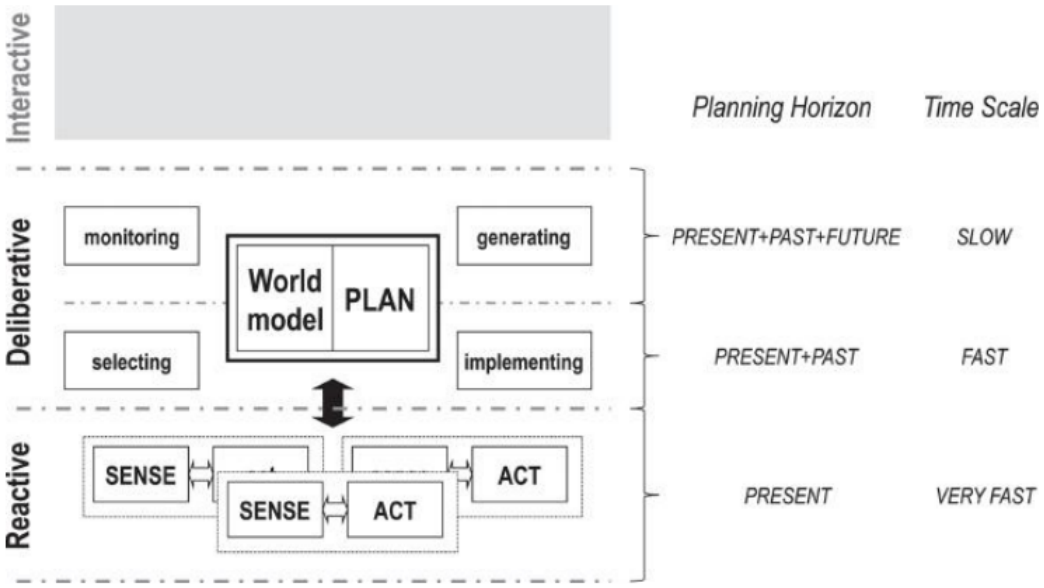
Interactive layer

Interactive functionality is need for interaction with other agents, either people, other robots, or software agents. Interaction may be regulated by reactive or cognitive loops. Displays of interactive functionality essentially surround the person and produce their persona. The interactive layer is essentially a wrapper on individual robot programming that enables the robot to work with other agents. The interactive layer may be programmed in procedural, functional, or ontological languages, such as OWL.

Other Operational Architectures

Levels of Automation (LOA)

- *Levels of Autonomy:* while biological intelligence provides insight into the general organization of intelligence, the human-machine systems community offers a different viewpoint. The human-machine system community has looked at automation for space exploration, control of processing plants, and autopilots in aircraft. The community has traditionally labeled the state of automation at any given time in terms of what functions a human has currently



delegated to the computer. This classification of the state in a system naturally leads to hierarchies of where the machine has responsibility for more of the functions, and thus would be considered more autonomous. These taxonomic hierarchies are often used as operational architectures or as de facto measure of how automated a system is, where level B is more automated than level A. The hierarchies are generally referred to as levels of automation or levels of autonomy, as automation and autonomy are often used as synonyms.

- *Traded, or Shared, Control:* A function can be performed by a human, a compute, both working together, or portion can be performed by either a human or computer and then the remaining portions by the other agent. In the first case we talk about shared control, while in the latter we talk about traded control. The state of automation for a process is defined by whether the human or the machine is performing the function. No particular levels off automation is accepted by all reserchers, with different experts arguing over the functions and the ordering of the hierarchy. However, the different variants of levels of automation all share the same tenet that there are a set of functions and the more functions delegated to a computer means higher automation. The resulting levels of automation approach is often used as an operational architecture, with the expecctation that an intelligent robot would be built incrementally in the layers. An unintended conssequencce of using levels of automation as an operational architecture is that it implies full automation as the design foal, rather than matching the appropriate level of automation for themission and robot's capabilities. The major advantage of using the levels of automation organization is that the four functions and levels offer more speccific modularity than the three layers in biological intelligence. The major disadvantage if this organization is the levels of autonomy were initially intended to be a set of definitions for labeling the current state of an active capability, not an operational architecture for coordinating all capabilities. There are other two disadvvantages: treating the levels of autonomy taxonomy as an operational architecture results in a system that considers only the four deliberative functions, ignoring reactive behaviors; and levels of autonomy taxonomy considers the mode of interaction with the human supervisor but does not consider interaction with other agents or nonsupervisory interactions.

Levels of Initiative

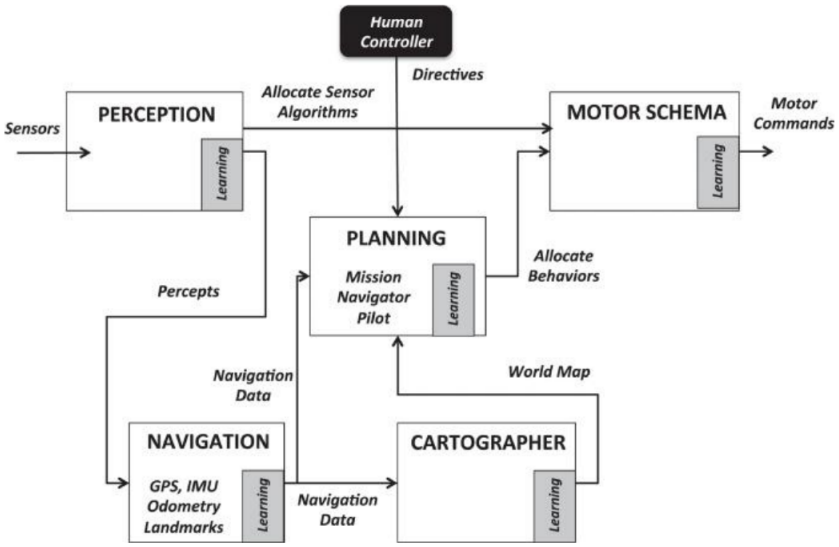
Another style of operational architecture is based on the amount of initiative that a robot is given to conduct a task. In this case, autonomy is based on the political connotation of autonomy as self-governing rather that the mechanical connotation we used previously. Unlike the other operational architecctures, initiative is conceptualized by roles rather than by functions. The five levels of initiative are:

- *No autonomy:* The robot follows rigid programmming to perform a task or achieve a goal. This level is similar to stimulus-response behaviors in a biological architecture and the 10 levels in the levels of automation architecture.
- *Process autonomy:* The robot can choose the algorithm or process to achieve its task goals. This choicce is similar to the selecting function in a levels of automation architecture.
- *System-state autonomy:* The robot can generate and choose between options to achieve its goal. This level is similar to the generating function in a levels of automation architecture but implies increased unpredictability and nondeterminism.
- *Intentional autonomy:* The robot can change its goals. This functionalty introudces the notion of being aware of others and explicitly ubteracting with them, similar to the interface layer in biologically-inspired architecture.
- *Constaint autonomy:* The robt can create its own roles and goals, which may involve relaxing constraints on its goals or how it accomplishes its goals. It is still subject to bounded rationality.

Five Subsystems in Systems Architectures

Operational architectures focus on the general style, while systems architectures focus on the general components. The software for an intelligent robot normally has at leasst fivve subsystems, which are encapsulated as object libraries or similar reusable programming repositories. System architectural design encourages designers to think in terms of creating libraries of algorithms and data structures for each subsystem. These libraries serve as general clearninghouses from which the designer can pick specific functions for a particualr robot in order to customize it for a new application. The subsystems are the followings:

- **Planning:** is associated with generting and monitoring overall mission objectives and passing along the geospatial component of those objectives to the navigation subsystem, selecting the motor and perceptual resources, implementing or instantiating those resources, and monitoring the performance of the mission. This subsystem is host to the numerous classical planning and resource allocation algorithms.
- **Cartographer:** also known as the world model or world map, is associated with the construction and maintenance of knowledge representations about the world. For navigation-oriented robots, this subsystem is the key data structure that enables generating, monitoring, selecting, and implementing actions. These knowledge representations are often geoospatial maps but can include more abstract symbolic information, such as beliefs about the state of the world or the intent of other agents. In more deliberative robots, this subsystem bridges the gap between signals and symbols. In interactive robotss, the subsystem also monitors the robot's mental models of itself and the beliefs, desires, and intents of other agents.



- **Motor Schema:** also known as Motor Schema Library, or more generally as the Behaviors subsystem, is associated with selecting the best motor routines and implementing actions. This subsystem contains the functions that connect the deliberative and reactive functions with the requisite guidance, navigation, and control algorithms for the actuators and effectors. This subsystem typically does not havve deiberative functionality but is rather about execution. Tee algorithms in the motor schema subsystem may be programmed to monitor their execution within a control loop.

- **Perception:** also knows as the Sensing Perceptual Shema, or Perceptual Schema Library, is associated with selecting the best sensors for the motor actions and implementing sensor processing algorithms. This subsystem contains the functions that control the sensors and extract the percept, that is, what is perceived. This includes algorithms that extract a special type of percept called affordances or those that perform object recognition and image understanding.

The subsystems are not independent and do not represent a sequence of programming actions. Instead, they are essentially the major object or classes in object-oriented programming that refelct software engineering decisions abut how to group similar functions and data. The list of subsystems is not necessarily complete. The five common subsystems highlight the historical focus within the robotics community on autonomous navigation of a platform and mapping. The five subsystems above have no clear component for interacting with other agents or for manipulation. An intelligent robot in healthcare, entertainment, or even driving in city traffic would likely have additional subsystems.

Three System Architecture Paradigms

Historically, system architectures for AI robotics fall into three categories caled paradigms: hierarchical, reactive, or hybrid deliberative/reactive. These paradigms arrange the flow of data and control within the robot. This data and control flow for each paradigm can be uniquely desccribed using two traits: the pattern of interaction between the three robot primitives of SENSE, PLAN and ACT, and the route of sensing.

Interaction between primitives

Intellligent robots use one of three cyclic patterns of interacction between the primitives. One is to SENSE, then PLAN, and finally ACT, at which point acting changes the state of the world which lead to a new iteration of SENSE, PLAN and ACT. We allso gave reactive animal intelligence, which uses a pattern of SENSE and ACT coupled into behavviors with no PLAN. Neurophysiological models favor a different pattern of intelligence where higher brain functions PLAN asynchronously while SENSE and ACCT behavviors both carry out plans and reacct to the environment.

Sensing Route

We have three routes:

- **Local sensing:** it goes directly from the sensor to the behavvior or function using the sensor data. The receiving function is responsible for any transformations or conversions of the raw data into a suitable data structure. The transformations or conversions stay local (within the programming scope of that function). Local sensing can be a one-to-one or a one-to-many mappping, where the data from one sensor can go to one or more independent functions.

- **Global sensing:** is how data originating from all the sensors is transmitted ot a function that transforms and fuses the data intoo a global world model or unified data structure. Global sensing can be thought of as a many-to-one mapping, where the data from many sensors go to one function.

- **Hybrid sensing:** is a ccombination, where the same data from a sensor may go to one or more functions that perform local transformations and to a global sensing function. Hybrid sensing can be thought of as a many-to-many mapping, where the data from many sensors can go to many independent functions.

Hierarchical System Architecture Paradigm

Hierarchical systems architectures organize the five subsystems to suppport a SENSE-PLAN-ACT pattern that relies on a global sensing route. Hierarchies are a natural way to organize functionality and they can be computationally efficient because they can reduce computation by specifying the frame and defining a closed world. They are used for implementations where the mission or application is well-understood and further additions of capabilities or major upgrades are not expected. They are often used where ther is little expecctation of reuse of code or later expansion of functionality. Thus the goal is to optimize the programming.
The SENSE phase collects data from all sensors using routines from the perception subsystem and then fuses sensor data into a World Model using routines from the cartographer subsystem. It fuses the current readings with older redings and any a priori knowledge, such as maps. The PLAN phase applies routines from the planner subsystem to the World Model to determine any changes to the missions or constraints and the pplans navigational routes and movements using routines from the navigational subsystem. The output is a set of acctions. The ACT phase executes the actions using routines from the motor schema subsystem. Action may also involve the perception subsystem if sensors have actuators to be moved. The SENSE-PLAN-ACT sequence executes in a continuous loop.

Reactive Systems Paradigm

The Reactive Systems use only two subsystems, Perception and Motor Schema, of the five canonical subsystems to support SENSE-ACT pattern tht relies on a local sensing route. Programming by behavior has a number of advantages, most of them consistent with good software engineering principles. Behaviors are inherently modular and easy to test in isolation from the system. Behaviors also support incremental expansion of the capabilities of a robot. A robot becomes more intelligent by having more behaviors. The behavioral decomposition results in an implementation tht works in real-time and is usually computationally inexpensive. Generally, the reaction speeds of a reactive robot are equivalent to stimulus-response times in animal, although if the technical architecture choices are implemented poorly, then a reactive implementation can be slow. Purely reactive robots are useful for simple autonomous systems, such as Roomba, robot mowers and so on. They can be used as well to add autonomous capabilities to an existing human-machine system where a rapid response is important.
The flow of a typical reacctive architecture uses multiple instances oof the Perception and Action subsystems in parallel. The SENSE phase collects data from all sensors using routines from the Perception subsystem and directly supplies them to the Motor Schemas which produce Actions. There may be multiple SENSE-ACT pairs and they run concurrently and independently. Each behavior (or SENSE-ACT pair) is a local loop and may have a different computation time, and thus the behaviors work asynchronously. Sensing is routed to each behavior without any filtering or processing and each behavior extracts the direcct perception that is unique to it. A sensor can supply the same sensor reading simultaneously to different behaviors, and then each behavior compute a different percept. The sensor processing is local to each behavior with no global model of the worls. There is no PLAN ohase as behaviors produce actions only if the sensing produces a reaction.
A robot constructed only with the reactive layer is often called either a reacctive or behavioral robot. A behavioral robot has at least 3 advantages: irecct perception is usually simple to program; the behaviors are highly modular; It can work welll in cooperation with a human in telesystems, because the rrobot can react to stimuli to trigger panic behaviors or guarded motion behavviors.
A reactive robot has 3 disadvantages: the inability to PLAN leads to the "fly-at-the-window" effect; the emergence of the overall behavvior from individual behaviors using direct perception makes it difficult to precisely predict what the robot will do. The behavior may not be optimal because it is based on local, instantaneous sensing rather than on a plan generated from a global model.

Hybrid Deliberative/Reactive Systems Paradigm

This kind of architecture organize the five subsystems to support a "PLAN, then SENSE-ACT" pattern coupled with hybrid sensing. The idea of this pattern evolved both from biomimicry and from two software engineering assumptions: planning covers a long time horizon and requires global knowledge; planning and sensor fusion algorithms are computationally expensive. The world model is constructed by the process independent of the behavior-specific sensing. Hybrid systems use hybrid sensing routes. The cartographer or model making processes can access not only the same sensors as the behaviors but also the results of local sensing. The cartographer can also have sensors that are dedicated to providing observations which are useful for world modeling but are not used for any active behavior. Hybrid systems are the most flexible and have beccome the default systems architecture for researchers and for applications where the robot's functionality is expected to be modified or extended in the future.

Execution Approval and Task Execution

The many facets of deliberation and reaction pose the concern as to whether we have algorithms and cooordination methods to actually build a fully autonomous robot that can work in the open world. One strategy is to build the robot but have a human surpervisor check the robot's planned actions before it executes tthem; this strategy is known as execution approval and task rehearsal. In the case of execution approval, the cognitive agent (human or computer) considers the next step and decides wheter it is safe. Taks rehearsal verifies the plan by considering an entire sequence of actions.

Behaviors

Even though it seems reasonable to explore biological and cognitive sciencens for insights into intelligence, how can we ccompare such different systems: carbon and silicon "life" forms? One powerful means of conceptualizing the different systems is to think of an abstract intelligent system as an agent. In object.oriented programming terms, the agent is the superclass, and the calsses of persona, animal and robot agents are derived from it. one helpuful way of seeing correspondences is to decide on the level at which these entitites have something in common. The set of levels of commonalities leads to a **computational theory.** The levels in a computational theory can be greatly simplified as:

- **Level 1: Existence proof of what can/should be done.** At Level 1, agents can share commonality of purpose or functionality.
- **Level 2: Decomposition of "what" into iputs, outputs, and transformations.** This level can be thought of as creating a fflow chart of black boxes to actually accomplish the "what" from Level 1. Each box represents a transformation from input to output. At Level 2, agents can exhibit common processes.
- **Level 3: How to implement the process.** This level of computational theory focuses on describing how eaach transformation is implemented, like a neural network or an algorithm. At Level 3, agents may have litlle or no commonality in the acctual physical implementation of the functionality specified in Level 1.

More precisely, Level 1 corresponds to a phenomenon that people can do. The second level refines the description of the phenomenon into a series of transformations that would describe what senso data enters the brain and what region of the brain are activated and produce outputs leading to the ultimate output. The third level describes the activity of each region of the brain in terms of the specific types oof neurons and how they are connected. it should be clear that Levels 1 and 2 are abstract enough to aplly to any agent. It is only Level 3 that the differences between a robotic agent and a biological agent really emerge.

Rana Computatrix

Starting with Level 1 of the computational theory approach, toads and frogs have visually directed behaviors: feed and flee. Level 2 requires expressing the feed and fflee behavviors as inputs, outputs, and transformations. From biology studies, frogs can be characterized as being able to detect only two visual inputs: small moving blobs on their visual field or large moving blobs. The small moving blobs are inputs to the feeding behavior, which produce the output. The output is that the frog orients itself towards the blob and then snaps at it. large moving bblobs are inputs to the fleeing behavior, causing the frog to hop away.
At Level 2, the stimulus output is called a **percept**.

Exploring the Level 3, they implemented the taxis behaviorr as a vecotr field: rana computatrix would literally feel an attractive force along the direction of the fly. The relative direction and intensity of the stimulus was represented as a vector. The direction indicated where the rana had to turn and the magnitude indicated the strength that should be applied to snapping. At Level 3 the perceptand the output are more precisely defined, though still not exppressed in a true software enginneering format. In the case of the rana computatrix implementation, if it gets more than an imput, it woudl wum the vectors, resulting in the generation of a new vector in-between of the original inputs.

Animal Behavior

Rana computatrix illustrates how a computational theory spans biological and artificial intelligence. A **Behavior** is a mapping of sensory inputs to a pattern of motor actions which then are used to achieve a task. We have diferrent kind of behaviors: **REFLEXIVE BEHAVIOR, STIMULUS-RESPONSE, REACTIVE BEHAVIOR, CONSCIOUS BEHAVIOR**. behaviors can then be divided into 3 broad categories: **Reactive behaviors, Reflexive behaviors and conscious behaviors**. Reflexive behaviors are stimulus-response. Reactive behaviors are learned and then consolidated to where they ccan be executed without conscious thought. Conscious behaviors are deliberative.

Reflexive Behaviors

They imply no need for ccognition: if you sense it, you do it. For a robot it means aa hardwired response, eliminating computation and guaranteed to be fast. Reflexive behaviors can then divvided into 3 categories:

- **Reflexes:** where the response lasts only as long as the stimulus, and the response is propotional to the intensity of the stimulus.
- **Taxes:** where the response is to move to a particular orientation
- **Fixed-action patterns:** where the response continues for a longer duration than the stimulus.

These categories are not mutually exclusive.

Behavioral Coordination

The overall response of robot ρ is a function of the behaviors B, the gain of each behavior G, and the **coordination function C**. The coordination function C takes the output of multiple behaviors and produces one response or output. More formally $\rho = C(G \times B(S))$. There are 3 categories of coordination functions. A set of behaviors can be either **concurrent**, that is, aall are acting at the same time, or they can be acting as a **sequence**. If the behaviors are concurrent, there are two broad categories of algorithms for coordinating the set of behaviors. One set is the **cooperating methods**, where the coordination function blends the output of the individual behaviors into a single output. The most common methods are vector summation. The other category of concurrent algorithms is **competing methods**, where the individual behaviors undergo arbitration. The most common methods of competiotion are subsumption and voting. A group of behaviors that form a sequence or macro can be grounded into an **abstract behavvior** that may use a finite state automata to explicitly coordinate behaviors while any other behaviors, such as avoiding obstacles, might use potential fields.

Potential Fields

The most common method for cooperating behavviors is to represent the output as a vecctor. Often the vector is produced by the motor schema using a potentail field representation of possible actions, typically referred to as a **potential field methodology**. They always use vectors to represent the action output of behaviors and vector summation to combine vectors from different behaviors to produce an emergent behavior.

Visualizing potential fields

We need to represent the motor action of a behavior as a potential field, which is an array, or field, of vectors. It has a magnitude (real number from 0.0 to 1, if normalized, otherwise any real number) and a direction. The array represent a region of the space, usually 2 dimensional. We can divide the map into squares, where each array represent a square of the space. Perceivable objects in the world would exert a force field on the surrounding space. The vector in each element represents the force, both the direction toturn and the magnitude or velocity to head in that direction. Potential fields are continuous because it does not matter how small the element is. The field represnet what the robot should do based on wheter the robot perceives an obstacle. The field is not concerned with how the robot came to be so close to the obstaccle; the robot feels the same force if it happens to move within range or if it was just sitting there and someone put a hand next to the robot. One way of thinking about potential fields is to imagine a force field acting on the robot.
There are five basical potential fields, or primitives, which can be combined to build more complex fields:

- **Uniform field:** the robot would feel the same force no matter where it was. No matter where it got set down and at what orientation, it would feel a need to align itself with the direction of the arrow and to move in that direction at a velocity proportional to the length of the arrow. (go to the direction n°).
- **Perpendicular field:** where the robot is oriented perpendicular to some object or wall or border. It can be pointed away or torward an object.
- **Attractive field:** the center of the field represents an object that is exerting an attraction to the robot. Wherever the robot is, the robot ffeels a force relative to the object. Attractive fields are to represent cases where the agent is attracted to light, food or goal.
- **Repulsive field:** opposite of an actractive field. Commonly associated with obstacles or things the agent should avoid. The closer the robot is to the object, the stronger the repuslive force pushes the robot 180 degrees away from it.
- **Tangential field:** the field is tangent to the object. Tangential field can spin either clockwise or counterclockwise. These fields are useful for directing a robot to go around an obstacle or having a robot investigate something.

Magnitude profiles

The magnitude profile is the way the magnitude of the vecotrs change. Mathematically the field can be eresented with polar coordinates, and the center of the field being the origin (0,0).. Magnitude profiles resolve the problem of constant magnitude. They also make it possible for a robot designer ro represent flexibility and to create interesting responses. Imagine the following scenario: if the robot is far away from the object, it will turn and move quickly towards it and the slow up to keep from overshooting and hitting the object. Mathematically, this is called a **linear drop off**, since the rate at which the magnitude of the vectors drops off can be plotted as a straight line. This profile tho shares the problem of the constant magnitude profile in the sharp transition to 0.0 velocity. Therefore, another profile might be used to capture the need for a stronger reaction but with mote of a taper. Such profile is an **exponential drop off** function, where the drop off magnitude is propotional to the square of the distance.

Potential fields and Perception

The force of the potential field at any given point was a function of both the relative distance between the robot and an bject and the magnitude profile. The strength of a potential field can be a function of the stimulus, regardless of distance.

Programming a Single Potential Field

They are actually easy to program, especially since the fields are egocentric to the robot. The robot computes the effect of the potential field, ussually as a straight line, at every update, with no memory of where it was previously or where it has moved from.
A primitive potential field is usuallly represented by a single function. The vector impacting the robot is computed at each update. Considering the case off a robot with a single tnage sensor facing forward. The designer has decided that a repulsive gield with a linear drop off is appropirat.

The formula is:

$$V_{direction} = -180, \qquad V_{magnitude} = \begin{cases} \frac{(D-d)}{D} & \text{for } d \leq D \\ 0 & \text{otherwise} \end{cases}$$

where D is the maximum range off the field's effecct, or the maximum distance at which the robot can detect the obstacle, and d is the distance off the robot to the obstacle.

Combination of Fields and Behaviors

The first attribute of a true potential field methodology is that it requires all behaviors to be implemented as potential fields. The second attribute is that it combines behaviors by **vector summation**. A robot will generally have forces from multiple behaviors acting on it concurrently. One problem is that the sum may be 0.0, leading to the **local minima problem**, i.e. the potential field has a minima, or valley, that traps the robot. (If you want more details look at the Murphy book).

Davantages and Disadvantages

The potential field is a continuous representetaion that is easy to visualize over a large reggion of space. As a result, it is eassier for the designer to visualize the robot's overall behavior. The potential fields can be parametrized: their range of influence can be limited, and any continuous function can express the change in amgnitude over distance. Furthermore, a ttwo-dimensional field can usually be extended to a 3-dimensional field, and so behaviros developed for 2D will work for 3D.
Buildin ga reactive system with potential fields is now withut disadvantages. The most commonly cited problem with fields is that multiple fields can sum to a vector with 0 magnitude. There are many elegant solutions to this problem: one of the earliest was to always have a motor schema producing vectors with a small magnitude from **random noise**. Another solution is the **navigation templates**. The avoid behavior receives, as input, the vector summed from the other behaviorss. This strategic behavior represents the direction the robot would take if there were no obstacles nearby. If the robot has a strategic vector, that vector gives a clue as to whether an obstacle whould be passed on the right or left.
A third solution to the local minima problem is to express the fields as **harmonic functions**. They are guaranteed not to have local minima of 0. This is more expensive as the entire field has to be computed for large areas, not just for the vectors acting on the robot.

Competing Methods: Subsumption

Competing methods use some form of arbitration, usually either subsumption or vvoting. In subsumption, higher levels of behaviors subsume or inhibit lower behaviors. In voting, behaviors cast votes leading to a winner-take-all scenario. The term behavior in the subsumption architecture has a less precise meaning than in schema theory. A behavior is a network of sensing and acting modules which accomplish a task. The modules are augmented finite state machines, or finite state machines which have registers, timers, and other enhancements to permit them to interface with other modules. In terms of schema theory, a subsumption behavior is actually a collecction of one or more behavvioral schemas which this book calls **abstract behavior**. Subsumption behavios are released in a timulus-response way, without an external program explicitly coordinating and

controlling them. There are four interesting aspects off subsumption in terms of releasing and control:

- **Layers of Competence.** Modules are grouped into layers of competence. The layers reflect a hierarchy of intelligence or competence. Lower layers encapsulate basic survival functions, such as avoiding collisions, while higher levels create more goal-directed actions, such as mapping. Each of the layers can be viewed as an abstract behavior for a partcular task.
- **Layers can Subsume Lower Layers.** Modules in a higher layer can overrride, or subsume, the output from behaviors in the next lower layer. The behavioral layers operate concurrently and independently, so there needs to be a mechanism to handle potential conflicts. The solution is subsumption is a type of winner-take-all, where the winner is always the higher layer.
- **No Internal State.** The use off internal state is avoided. internal state in this cae means any type of local, persistent representation which represents the state of the world, or a model. Because the robot is a situated agent, most of its information should come directly from the world. If the robot depends on an internal representations, what it believes may begin to diverge dangerously from reality. Some internal state is needed ffor releasing behaviors like being scared or hungry, but good behavioral designss minimize this need.
- **Activate Task by Activating Appropriate Layer.** A task is accomplished by acctivating the appropriate layer, which then activates the layer below it, and so on. However, in praactice, subsumption style systems are not easily taskable, that is, they cannot be ordered to do another task without the behaviors being reorganized.

There is an example on the book, but i'm gonna skip it and write just few notions i've found :)
Inhibition: the output of the subsuming module is connencted to the output of another module. If the output of the subsuming module is on or has any value, the outut of the subsumed module is blocked or turned off. Inhibition acts like a faucet, turning an output stream on and off.
Suppresion: the output of the subsuming module is connected to the input of another module. If the output of the subsuming module is on, it replaces the normal input to the subsumed module. Suppresion acts like a switch, swapping one input stream for another.

sequences: Finite State Automata

Potential fields and subsumption are good for handling behaviors that are independent. However, some behaviors form a larger abstract behavior where the intersections may be more nuanced. In these cases, a finite state automata or a script or a related construc tcalled skills, might be a good choice; the three are equivalent nut reflects different programming styles. A **Finite State Automata** is a set of popular mechanisms for specifying what a program should be ding at a given time or circumstance. The FSA is generally written as a table accompanied by a **state diagram**, giving the designer a visual representation. There are many variants of FSA, but each works about the same way. There are 2 examples on the boook: a follow the road FSA and a Pick Up the Trash FSA. Not gonna put them here.

Sequences: Scripts

Abstract behaviors often use scropts, or a related construct called skill, to create generic templates of assemblages of behaviors. Scripts provide different way of generating the logic for an assemblage o behaviors. They encourage the designer to think that the robot and the task in terms of

a screenplay. Scripts were originally developed for natural language processing to help the audience understand actors. Scripts can be use more litterally, where the actors are robots reading the script. The script has more room for imrpovisation, thoughh, should the robot encounter an unexpected condition, the robot begins following a **sub-script.**
The main sequence of events is called **casual chain**, which is critical because itt embodies the coordination control program logic just as a FSA does.
When programmin grobots, people often like to abbreviate the routine portions of control and concentrate on representing and debugging the important sequences of events. FSA force the designer to consider and enumerate every possible transition, while scripts simplify the specification. The concepts of **indexing** and **focus-to-attention** are extremely valuable for coordinating behaviors in robots in an eficient and intuitive manner.
The resulting script for an abstract behavior to accomplish a task is usually the same as the programming logic derived from an FSA.

Perception

Sensors from notes

We want to extract information from the sensor for the perception. We have a pipeline for the perception: From raw data we extract features with navigation, then we obtain objects with interaction and we get places/situations with servicing/reasoning. The pipeline can work also going backward. To go from one level to the other you can use different techniques and/or models. Senors provide the raw data. Sensing is the combination of the data you acquire and the processing, and it produces percepts. Sensor fusion is the mechanism which allows to merge the information coming from different sensors to produce percept and world model (like merging laser scan data with image data).
We have 3 types of Perception:

- **Proprioceptive:** sensing stimuli that are produced and perceived within an organism, especially those connected with the position and movement of the body;(inside robot)
- **Exteroceptive:** sensing stimuli that are external to an organism;(outside robot)
- **Exproprioceptive:** the sense of the position of external objects relative to parts of the body.(camera for selfies).

We can divide sensors again into passive and active sensors. The passive sensors grab and collect the energy that is in the environment(like a thermometer or a camera). Active sensors emit energy in the environment and then the sensors collect back what the objects in the environment are reflecting back. We can classify sensors also on the type of output they gave back. Each sensor has their own characteristics: range, resolution and linearity. The perceptive horizon is the range i have plus the occlusions, so what i can actually see compared to the theoretical area i can cover. Probabilistic robotics uses mathematical formulation to extract information from the noise or to cope with it. We have different errors: offset error is how far are you from the true value; linearity error is a non-constant error in our measurement; the resolution error are triggered only if the change is of a specific value.

Sensor model

To build a model we need to understand how sensor are build and how they work.

- **Encoders:** sensor that can tell us the values of how many degrees an axle has to turn.
- **Heading:** can be proprioceptive or exteroceptive. Used to determine the robots orientation and inclination(e.g., gyroscope, acceleration, compass, inclinometer).
- **Compass**

- **Gyroscopes**
- **Accelerometers**
- **IMU (Inertia Measurement Unit)**: composed by 3 gyroscopes and 3 accelerometers, which can give us the displacement and the orientation of the robot.

Dead reckoning is the process of calculating one's position, especially at sea, by estimating the direction and distance travelled rather than by using landmarks or astronomical observations. We also have contact senors. They are passive and they measure the contact with objects. They are cheap but have poor sensitivity, coverage and localization.

Sensors from Siegwart

A wide variety of sensors is used in mobile robots. Some sensors are used o measure simple values such as the internal temprerature of a robot's electronics or the rotational speed of the motors. Other more sophisticated sensors can be used to acquire information about the robot's ennvironment or even to measure directly a robot's global position.

Sensor classification

We classify sensors using two important functional axes: **Proprioceptive/exteroceptive** and **passive/active**. Proprioceptive sensors measure values internal to the system, like motor speed, wheel load, robot arm joint angles and battery voltage. Exteroceptive sensrs acquire information from the robot0s environment, like distance measurements, light intensity and sound amplitude. Hence exteroceptive sensor's measurements are interpreted by the robot in order to extracct meaningful environmental features.
Passive sensors measure ambient environmental energy entering the sensors, like CCD or CMOS cameras. Active sensors emit energy into the environment, then measure the environmental reaction. Activve sensing introduces several risks: the otubound energy may affect the very characteristics that the sensor is atttempting to measure. Furthermore, an active sensor may suffer from interference between its siggnal and those beyond its control. The following image provides a classification of the most useful sensors fro mobile robot applications.

Characterizing sensor performance

In order to quantify sensors performance characteristics, first we formally define the sensor performance tterminology that will be valuable trhought the rest of this chapter. A number of sensor characteristics can be rated quantitatively in a laboratory setting. Such performance rating will necessarily be best-case scenarios when the sensor is placed on a real world robot, but are nevertheless useful.
Dynamic range is used to measure the spread between the lower and upper limits o input values to the sensor while maintaining normal sensor operation. Formally, the dynamic range is the ratio of the maximum input value to the minimum measurable input value. This ratio is usually measured in **decibels**, which are computed as ten times the common logarithm of the dynamic range ($10 \log[\frac{\text{upper bound}}{\text{lower bound}}]$).
Range is aslo an important rating in mobile robot applications because often robot sensors operate in environments wherre they are frequently exposed to input values beyond their working range.

General classification (typical use)	Sensor Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers	EC	P
	Optical barriers	EC	A
	Noncontact proximity sensors	EC	A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A/P
Acceleration sensor	Accelerometer	PC	P
Ground beacons (localization in a fixed reference frame)	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensor	EC	A
	Laser rangefinder	EC	A
	Optical triangulation (1D)	EC	A
	Structured light (2D)	EC	A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s)	EC	P
	Visual ranging packages		
	Object tracking packages		

In succh cases, it is criitical to understand how the sensor will respond.

Resolution is the minimum difference between two values that can be detected by a sensor. Usually, the lower limit of a dynamic range of a sensor is equal to its resolution. However, in the case of digital sensors, this is not necessarily so.

Linearity is an important measure governing the behavior of the sensor's utput siignal as the input signal varies. A linear response idicates that if two inputs x and y result in the two output f(x) and f(y), then for any values a and b, $f(ax + by) = af(x) + bf(y)$. This means that a plot of the sensor's input/output response is simply a straight line.

Bandiwth or frequency is used to measure the speed with which a sensor can provide a stream of readings. Formally, the number of measurements per second is defined as the sensorr's frequency in hertz.

The sensor characteristics can be reasonibly measured in a laboratory environment with confident extrapolation to performance in real-world deployment.

Sensitivvity is a measure of the degree to wich an incremental change in the target input signal changes the outpput signal. formally, sensitivity is the rtio of output changes to input changes. Unfortunately, however, the sensitivity off exteroceptiive sensors is often confounded by undesirable sensitivvity and performance coupling to other envvironmental parameters.

Cross-sensitivity is the technical term for sensitivity to environmental parameters that are orthogonal to the target paraameters for the sensors.

Error of a sensor is defined as the difference between the sensor's output measurements and the true values being measures, within some specific operating context. Given a true value v and a measured value m, we can define error as $errror = m - v$.

Accuracy is defined as the degree of comformity between the sensor's measurement and the true

value, and is often expressed as a proportion of the true value. Thus small error corresponds to high accuracy and viceversa:

$$(accuracy = 1 - \frac{|error|}{v})$$

Of course, obtaining the ground truth v can be difficult or impossible, and soo establishing a confident characterization of sensor accuracy can be problematic. Furthermore, it is important to distinguish between two different source of error:

Systematic errors are caused by facctors or processes that can in theory be modeled. These errors are therefore deterministics. Poor calibration of a laser, an unmodeled slop, or a damaged sensor due to a collision are causes of systematic sensor errors.

Raandom errors cannot be predictd using a sophisticated model; neither can they be mitigated by more precise sensor machinery. These errors can only be described in probabilistic terms. Instability in a color camera, spurious rangefinding errors, and black level noise in a camera are example of random errors.

Precision is often confused with accuracy and now we have the tools to clearly distinguish the two terms. High precision relates to reproducibility of the sensor results. Precision does not have any bearing on the accuraacy of the sensor's output with respect to the true value being measured. Supppose that the random error of a sensor is characterized by some mean value μ and a standard deviation σ. The formal definition of precision is the ratio of the sensor's output range to the standard deviation: $precision = \frac{range}{\sigma}$. Note that only σ and not μ has an impact on precision. In contrast, mean error μ is directly proportional to overall sensor error and inversely proportional to sensor accuracy.

Mobile robots depend heavily on exteroceptive sensors. Many of these sensors concentrate on a central task for the robot: acquiring information on objects in the robot's immediate vicinity so that it may interpret the state of its surroundings. We are now gonna desccribe how dramatically the sensor error of a mobile robot disagrees with the ideal picture drawn previously:

- **Blurring of systematic and random error:** Active ranging sensors tend to have failure modes that are triggered largely by specific relative positions of the sensor and environment targets. For exammple, a sonar sensor will produce specular reflections, producing grossly inacccurate measurements of range, at specific angles to a smooth sheetrock wall. During motion of the robot, such relative angles occur at stochastic intervals. The fundamental mechanism at work here is the cross-sensitivity of mobile rrobot sensors to robot pose and robot-environmentt dynamics. The models for such cross-sensitivity are not, in an underlying sense, truly random. The important point is to realize that, while systematic and random errors are well defined in a controlled settings, the mobile robot can exhibit error charateristics that bridege the gap between deterministic and stochastic error mechanisms.

- **Multimodal error distributions:**It is common to characterize the behavior of a sensor's random error in terms of a probability distribution over various otput values. In general, one knows very little about the causes of random error, and therefore several simplifying assumptions are commonly used. For example, we can assume that the error is zero-mean in that it symmetrically generates both positive and negative measurements error. We can go even further and assume that the probability density curve is a Gaussian. It is imprtant to recognize that one frequently assumes **symmetry and unimodular distribution**. This means that measuring the correct value is most probable, and any measurement that is fartehr away from the correct value is less likely than any measurements that is closer to the correcct value. These assumpito, even tho they are really useful, they are ususally wrong.

Representing uncertainty

Sensoors are imperfect devices with errors of both systematic and randomm nature. The latter in particular cannot be corrected, and so they represent atomic levels of sensor uncertainty. We have already defined error as the difference between a sensor measurement and the true value. from a statistical point of view, we wish to characterize the error of a sensor, not for one specific measurement but for any measurement. Let us formulate the problem of sensing as an estimation problem. the sensor has taken a set of n measurements with values ρ_i. The goal is to characterize the estimate of the true value $\mathbb{E}[X]$ given these measurements: $\mathbb{E}[X] = g(\rho_1, ..., \rho_n)$. from this perspective, the true value is represented by a random variable X. We use a **probability density function** to characterize the statistical properties of the value X. Recall of probability notions:

$$\int_{-\infty}^{+\infty} f(x)dx = 1 \qquad p[a < X \leq b] = \int_a^b f(x)dx$$

The probability density function is a useful way to characterize the possible values of X because it captures not only the range of X but also the comparative probability of different values for X. Using f(x) we can quantitatively define mean, variaccne, and standatd deviation as follows:
The mean value μ is quivvalent to the expected value $\mu = \mathbb{E}[X] = \int_{-\infty}^{+\infty} xf(x)dx$. This eqaution is identical to the wwighted average of all possible values of x. In contrast, **the mean square value** is simply the weighted average of the squares of all values of x: $\mathbb{E}[X^2] = \int_{-\infty}^{+\infty} x^2 f(x)dx$. Characterization of the wiht of the width of the possible values of X is a key statistical measure, and this requires first defining the variance $\sigma^2 = Var(X) = \int_{-\infty}^{+\infty}(x-\mu)^2 f(x)dx$. Fianlly the standard deviation σ is simply the square root of the variance, and the variance will play important roles in our characterization of the error of a single sensor a well as the errror of a model generated by combining multiple sensor readings. Important formulas: if the events are independent then:

$$\mathbb{E}[X_1 X_2] = \mathbb{E}[X_1]\mathbb{E}[X_2] \qquad Var(X_1 + X_2) = Var(X_1) + Var(X_2)$$

There are also the notions of the Gaussian probability density function, but since we have seen in so many times i'm gonna skip it :).
These probability mechanism may be used to describe errors associated with a single sensor's attempts to measure a real-world value. But in mobile robotics, one often uses a series of measurements, all of them uncertain, to extract a single environmental measure. Consider a system where X_i are n input signal with a known probability distribution Y_i are m outputs. What can we say about the probability distribbution of the output signals Y_i if they depend with known functions f_i upon the input signals? The generral solution can be generated using the first-order Taylor expansion of f_i. The output covarance matrix C_Y is given by the error propagation law:

$$C_Y = F_X C_X F_X^T$$

Where C_X and C_Y are the covaraince matrix representing the input uncertainties and the propagated uncertainties for the uìoutputs respectively. F_x iss the jacobian defined as follows:

$$F_X = \nabla f = \begin{bmatrix} \frac{\partial f_1}{\partial X_1} & \cdots & \frac{\partial f_1}{\partial X_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial X_1} & \cdots & \frac{\partial f_m}{\partial X_n} \end{bmatrix}$$

Wheel/motor sensors

They are devices used to measure the internal state and dynamics o a mobile robot. These sensors have vast applications outside of mobile robotics and mobile robotics has enjoyed the benefits of

high-quality, low-cost wheel and motor sensors that offer excellent resolution.
Optical encoders have become the most popular device for measuring angular speed and position within a motor drive or at the shaft of a heel or steering mechanism. In mobile robotics, encoders are used to control the position or speed of wheels and other motor-driven joints. Because these sensors are **proprioceptive**, their estimate of position is best in the reference frame of the rrobot and, when applied to the problem of robot localization, significat corrections are required. An optical encoder is basically a mechanical light chopper that produces a certine number of sine or square wae pulses ffor each shaft evvolution. It consists of an illumination source, a fixed grating that masks the light, a rotor disc with a a fine optical grid that rotates with the shaft, and fixed optical detectors. As the rotor moves, the amount of light striking the optical detectors varies based on the alignment of the fixed and moving gratings. In robotics, the resulting sine wave is transformed into a disscrete square wave using a treshold to choose between **light** and **dark** states. Resolution is measured in **cycles per revolution (CPR)**. The minimum angular resolution can be readly computed from an encoder's CPR rating. A typical encoder in mobile robotics may have 2000 CPR, while the optical encoder industry can readily manifacture encoders with 10'000 CPR. Usually in mobile robotics the **quadrature encoder** is used. In this case, a second illumination and detector pair is placed 90 degrees shifted w.r.t. the original in terms of the rotor disc. The resulting twin square waves provide significantly more information. The ordering of which square wave produces a rising edge first identifies the direction of rotation. Furthermore, the four detectably different states imrpove the resolution by a factor of four with no change to the rotto disc. As with most proprioceptive sensors, encoders are generally in the controlled environment of a mobile robot's internal structure, and so systematic error and cross-sensitivity can be engineered away.

Heading sensors

Heading sensors can be proprioceptive or exteroceptive. They are used to determin the robot's orientation and inclination. They allow us to integrate the mvement to a position estimate. This procedure, which has its roots in vessel and ship navigation, is called **dead reckoning**.

- **Compasses:**the two most common modern sensors for measuring the direction of a magnetic field are the Hall efffect and flux gate compasses. The **Hall effect** describes the behavior of electric potential in a semiconductor when in the presence of a magnetic field. When a costant current is applied across the length of a semiconductor, there will be a bvoltage difference in the perpendicular direction, across the semiconductor's width, based on the relative orientation of the semiconductor to magnetic flux lines. In adddition, the sign of the voltage potential identifies the direction of the magnetic field. **The flux gate compass** operates on a different principle. Two small coils are wound on ferrite cores and are fixed perpendicular to one another. When alternating current is activated in both coils, the magnetic field causes shifts in the phase depending on its relative alignment with eacch coil. By measuring both phase shifts, the direction of the magnetic field in two dimensions can be computed. The flux gate compass can accurately measure the strength of a magnetic field and has improved resolution and accuracy; however it is both larger and more expensive than Hall effecct compass. However, compass are disturbet by other magnetic fields, so it is better to use them in local environment.

- **Gyroscopes:** are heading sensors that preserve their orientation in relation to a fixed reference frame. They provide an absolute measure for the heading of a mobile system. We have

two classes of gyroscopes: **Mechanical** and **Optical**. Mechanical gyroscope relies on the inertial properties of a fast-spinning rotor. The property of interest is known as the gyroscopic precession. If you try to rotate a fast-spinning wheel around is vertical axis, you will feel a harsh reaction in the horizontal axis. This is due to the angular momentum associated with a spinning wheel and will keep the axis of the gyroscope inertially stable. The reactive troque τ and thus the tracking stability with the inertial frame are proportional to the spinning speed ω, the precession speed Ω and the wheel's inertia I: $\tau = I\omega\Omega$. Opticcal gyroscoper work on the principle that the speed of light remains unchanged and geometric change can cause ligth to take a varying amount of tiime to reach its destination. One laser beam is sent traveling clockwise though an optical fiiber while the other travels couterclockwise. Because the laser traveling in the direction of rotation has a slightly shorter path, it will have a higher frequency. This principle is known as the **Sagnac effect**. The difference in frequency Δf of the two beams is proportional to the angular velocity Ω of the cylinder.

Accelerometers

An accelerometer is a devicce used to measure all external forces acting upon it, including gravity. Accelerommeters belong to the proprioceptive sensors class. Concpetually, it is a spring-mass-damper system in wich the three-dimensional position of the proof mass relative to the accelerometer casing can be measured with some mechanism. Assume that an external force is applied on the sensor casing and that we have an ideal springg with a force proportional to its displacement. Then we can write:

$$F_{applied} = F_{inertial} + F_{dampling} + F_{spring} = m\ddot{x} + c\dot{x} + kx$$

where m is the proof mass, c is the damping coefficient, k is the spring constant, and x is the equilibrium case relative position. By choosing appropriately the damping material and the mass, the system can be made to converge very quickly to a stable value under the effect of a static force. When the stable value is reached, then $\ddot{x} = 0$ and the applied acceleration can be obtained as $a_{applied} = \frac{kx}{m}$. Noticce that each accelerometer measures acceleration along a single axis. By mounting three accelerometers orthogonally to one another, an omnidirectional accelerometer can be obtained. we can further distinguish accelerometers in two classes: **low-pass accelerometers**, which can measure accelerations from 0 to 500 Hz. The second cathegory of accelerometers are used for measuring accelerations of vibrating objects or accelerations during crashes. The bandwidth ranges between few up to 50 KHz.

Inertial measurement unit

Is a device that uses gyroscopes and accelerometers to estimate the relative position, velocity, and acceleration of a moving vehicle. It estimates the six-degree-of-freedom pose of the vehicle: position (x,y,z) and orientation (roll, pitch, yaw). It is composed by 3 orthogonal acccelerometers and 3 orthogonal gyroscopes.

Groound beacons

One elegant approach to solving the localization problem in mobile robotics is to use active or passive beacons. Using the interaction of on-board sensors and the envvironmentatl beacons, the robot can identify its position precisely. Although the general intuition is identical to that of early human navigation beacons, modern technology has enabled sensors to localize an outdoor robot

with acccuracies of better than 5cm within areas that are kilometers in size. now we are gonna look at the **Global Positioning System (GPS)**, initially developed for military use. There are at least 24 operational GPS satellites at all times. 4 satellites are located in each of six planes inclined 55 degrees w.r.t. the plane of the earth's equator. Eacch satellite continuously transimts data that idicate its location and the current time. therefore, GPs receivers are completely passive but exteroceptive sensors. The GPS satellites synchronize their transmission so that their signals are sent at the same time. When a GPS receiver reads the transmission of two or more satellites, the arrival time differences inform the receiver as to its relative distance to each satellite. By combining the information regarding the arrival time and instantaenous location of 4 satellites, the receiver can infer its own position.

Active ranging

Acctive ranging sensors continue to be the most popular sensors in mobile robotics. Many ranging sensors have a low price point and they provide easily interpreted outputs: direc measurements of distance from the robot to objects in its vicinity. For obstacles detection and avoidance, most mobile robots rely heavily on active ranging sensors.
Time-of-Flight ranging makes use of the propagation speed of sound or an elecctromagnetic wave. In general, the travel distance of a sound of electromagnetic wave is given by d=ct, where d is the distance traveled (usually round-trip), c the speed of wave propagation and t the time of flight. The quality of time-of-flight range sensors depends mainly on: uncertainties in determining the exact time of arrival of the reflected signal; inaccuracies in the time-of-flight measurements; the dispersal cone of the transmitted beam; interaction with the target; variation of propagation speed; the speed of the mobile robot and target.
Ultrasonic sensors transmits a packet of pressure waves and measures the time it takes for this wave packet to reflect and return to the receiver. The distance d of the object causing the reflection can be calculated based on the propagation speed of sound c and the time of flight t: $d = \frac{ct}{2}$. The speed of sound c is given by $c = \sqrt{\gamma RT}$, where gamma is the ratio of specific heats, R the gas constant and T the temperature in Kelvin.
Laser rangeffinder It is a time-of-flight sensor that acchieves significant improvements over the ultrasonic range sensor owning to the use of laser light instead of sound (electromagnetic wave). This type of sensor consists of a transmitter that illuminates a target with a collimated beam, and a receiver capable of detecting the component of light, which is essentially ccoaxial with the transmitted beam. Oftten referre to as optical radar or **lidar**, thede devices produce a range estimate based on the time needed for the light beam to reach the tarhet and return. A mechanical mechanism with a mirror sweeps the light beam to cover the required scene in a plane or even in 3 dimension, using a rotating, nodding mirror.
Phase-shift measurement. Near-infrared ligtht is collimated and traansmitted from the transmitter and hits a point P in the environment. For surfaces having a roughnesss greater than the wavelenght of the incident liht, diffuse reflection will occur, meaning that the light is reflected almost isotropically. The wavelength of the infrared light emitted is 824 nm, and so most surfaces, with the exception of only highly polished reflecting objects, will be diffuse reflectors. The component of the infrared light that falls within the receiving aperture of the sensor will return almost parallel to the transmitted beam for distant objects. The sensor transmits 100% amplitude-modulated light at a known frequency and measure the phase shift between the transmitted and reflected signals. The wavelength of the modulating signal obeys the equation $c = f\lambda$, where c is the speed of light and f the modulating frequency. For f=5MHz, $\lambda = 60m$, the total distance D' covered by the emitted light is $D' = L + 2D = L + \frac{\theta}{2\pi}\lambda$, where D is the distance from the target

and L is the distanc ebetween the transmitter and the phase measurement sensor. The required distance D between th ebeam splitter and the target is therefore given by $D = \frac{\lambda}{4\pi}\theta$, where θ is the electronically measured phase difference between the transmitted and reflected light beams, and λ the knwon modulating wavelength.
3D laser rangefinders is a laser scanneer that acquires scan data in more than a single plane. Custom-made 3D scanners are typically built by nodding or rotating a 2D scanner in a stepwise orr ccontinnuous manner around an axis parallel to the scanning plane.
Time-of-Flight camera works similarly to a lidar with the advantage that the whole 3D scene is captured at the same time and that there are no moving parts. This device uses a modulated infrared lihting source to determine the distance for each pixel of a Photonicc Mixer Device (PMD) ssensor. As the illumination source is placced just next to the lens, the whole system is very compact compared to lidars, stereo vision, or triangular sensors. In the presence of background light, the image sensor receives an additional illumination signal which disturbs the distance measurement. To eliminate the background part of the signal, the acquisition is ddone a second time with the illumination switched off. As the scene is captured in one shot, the camera reaches up to 100 frames per second and is therefore ideally suited for real-time applications.

Triangular active ranging

Triangular ranging sesnors use geometric properties manifest in their measuring strategy to establish distance readings to objects. The simplest class of triangulation rangers are active because they project knwon light pattern onto the environment. The reflecction f the knwon pattern is captured by a recceiver and, together with known geometric values, the system can use simple triangulation to establish range measurements.
Optical triangulation (1D sensor). The principle behind this sensor is straightforward: a collimated beam is transmitted toward the target. The reflected light is collecctd by a lens and projected onto a position-sensitive device or linear camera. the distance D is given by $D = f\frac{L}{x}$. The distance is proportional to 1/x; therefore the senor resolution is best for close objects and becomes poor at a distance. Sensor based on this principle are used in rrange sensing up to 1 or 2 m, but also in high-precision industrial measurements with resolutions far below $1\mu m$.
Structured ligth (2D sensor). If one replacces the linear camera or PSD of an opticcal tirrangulation sensor with a 2D recceivver succh as a CCD or CMOS camera, then one can recover distance to a large set of points instead of to only one point. The emitter must project a knwn patter, or **structured light**, onto the environment. Many systems exist which either projecct light textures or emit colimated light by means of a rotating mirror. Yet another popular alternative is to projecct a laser stripe by turning a laser beam into a plane using a prism. Regardless of how it is created, the projected light has a known structure, and therefore the image taken by the CCD or CMOS receiver can be filtered to identify the pattern's reflection. Note that the problem of recovering depth is in this case far simpler than the problem of passive image analysis. In passivve image analysis existing features in the environment must be used to perform **ccorrelation**, while the present method projeccts a known pattern upon the environment and thereby avoids the standard correlation problem altogether. Furthermore, te structured light senso is an active devvice so it will continue to work in dark environments as well as environments in which the objects are featureless. In contrast, stereovision would fail in such texture-free circumstances. I'm gonna skip the math behind this, cause personally we haven't evenn mensioned this sensor in class :)

Motion/speed sensors

Some sensors measure direcctly the relative motion between the robot and its environment. Since such motion sensors detect the relativve motion, so long as an object is moving relative to the robot's reference frame, it will be detected and its speed can be estiamted.
Doppler effect sensing (radar or sound). A transmitter emits an electromagnetic or sound wave with a frequency f_t. It is either received by a receiver or refleccted from ann object. The measured frequency f_r at the receiver is a function of the relative speed v between transmitter and receiver according to $f_r = f_t \frac{1}{1+v/c}$ if the transmitter is movig and $f_r = f_t(1 + v/c)$ if the receiver is moving. In the case of a reflected wave there is a facctor of 2 introduced, since any change x in relative separation affects the round-trip path length by 2x. Furthermore, in such situations it is generally more convenient to consider the change in frequency Δf, knwon as the **Doppler shift**, as opposed to the Doppler frequency notation above:

$$\Delta f f = f_r - f_t = \frac{2 f_t v \cos\theta}{c}, \qquad v = \frac{\Delta f c}{2 f_t \cos\theta}$$

where Δf is the Doppler frequenccy shift, θ the relative angle between direction of motion and beam axis. The Doppler effect applies to sound and magnetic waves.

Fundamentals of Computer Vision

The digital camera

Light falling on an image sensor is usually picked up by an active sensing area, integrated for the duration of the exposre and then passed to a seto os sense amplifiers. The two main kinds of sensors used in digital and video cameras today are CCD and CMOS:
CCD cameras are composed of CCD chip, an array of light-sensitive picture elements, or pixels, usually with between 20'000 and several milion pixels total. Each pixel can be thought of as a light-sensitive, discharging capacitator that is 5 to 25 μm in size. First the capacitors of all pixels are charged fully, then the integration period begins. As photons of light strike each pixel,they liberate elecctrons, which are captured by electric fields and retained at the pixel. Over time, each pixel acccumulates a varying level of charge based on the total number of photonss that have struck it. After the integration period is complete, the relative charges of all pixels need to be frozen and read. In a CCD, the reading process is performed at one corner of the CCD chip. The bootm row of pixel charges is transported to this corner and read, then the rows above shift down and the process is repeated. This means that each charge must be transported across the chip, and it is critical that the value be preserved. Thiss requires specialized control circuitry and custom fabrication techniques to ensure the stability of transported charges. The key disadvantages are primarly in the areas of inconsistency and dynamic range. Tehy also perform badly in case of bad illumination or very high illumination.
CMOS cameras have the complementary metal oxide semiconductor chip, which is a significant departure from the CCD. It has an array of pixels, but located along the side of each pixel are several transistors specific to that pixel. As in CCD chips, all of the pixels accumulate charge during the integration period. During the data collecct step, the CMOS takes a new approach: the pixel-specific circuitry next to every pixel measures and amplifies the pixel's signal, all in parallel ffor every pixel in the array. Using more traditional traces from general semiconductor chips, the resulting pixel values are all carried to their destinations.
CMOS have a number of advantages ovver CCD technologies. First an foremost, there is no need for specialized clock driver and circuitry required in the CCD to transfer each pixel's charge down

all pf the array columns and across all of its rows. It consumes less power and it is simpler.
Color Camera There are two common approaches for creating color images, which use a single cchip or three separate chips. The single chip technology usses the so-called **Bayer Filter**. The pixels on the chip are grouped into 2x2 sets of four, then red, green, and blue color filters are applied so that each individual pixel receives only light of one color. Normally, two pixels of each 2x2 block measure green while the remaining two pixels measure red and blue light intensity. The reason there are twice as many green filters as red and blue is that the luminance signal is mostly dominated by green values, and the visual system is much more sensitive to high frequency detail in luminance than in chrominance. The process of interpolating the missing color values so that we have a valid RGB value as all the pixels is known as **demosaicing**. The number of pixels in the system has been efffectively cut by a factor of four, and therefore the image resolution output by the camera will be sacrificed.
The **three-chip color camera** avoids these problems by splitting the incoming light into 3 complete copies. Three separate chips receive the light, with one red, green or blue filter over each entire chip. Thus, in parallel, each chip measures light intensity for one color, and the camera must combine the chips' outputs to create a joint color image. Resolution is preserved in this solution, although the three-chip color cameras are significantly more expensive and therefore more rarely used in mobile robotics.

Image formation

Before we can inteligently analyze and manipulate images, we need to understand the image formation process that produced a particular image. Once the light from the scene reaches the camera, it must still pass through the lens before reaching the sensor. The lens is thin and it is composed of a sinfle piece of glass with very low, equal curvature on both sides. According to the lens law the relationship between the distance to an object z and the distance behind the lens at which a focused image is formed e can be expressed as $\frac{1}{f} = \frac{1}{z} + \frac{1}{e}$, where e is the focal lenght. As you can perceive, this formula can also be used to estimate the distance to an object by knowing the focal length and the current distance of the image plane to the lens. This technique is called **depth from focus**. I the image plane is located at istance e from the lens, then for the specific object voxel depicted, all light will be focused at a single point on the image plane and the objecct voxel will be focused. However, when the image plane iis not at e, then the light from the object voxel will be cast on the image plane as a **blur circle**. To a first pproximation, the light is homogeneously distribbuted thoughout this blur circle, and the radius R of the circle can be characterized accprding to the equation $R = \frac{L\delta}{2e}$, where L is the diameter of the lens or apperture and δ is the displacement of the image plane from the focal point.
Pinhole camera model, or camera obscura, has no lens, but a single very small aperture. Light from the scene passes through this single point and project an inverted image on the oppposite side of the box.
perspective Projections. To decribe analitically the perspective projection operated by the camerra, we have to introduce some opportune reference system wherein we can express the 3D coordinates of the scene point P and the coordinates of its projection p on the image plane. The simplified model considers (x,y,z) as the **camera reference frame** with origin C and z-axis coincident with the optical axis. Assume also that the camera reference frame coincides with the world reference frame. This implies that the ccoordinates of the scene point P are already expressed in the camera frame. Let us also introduce a two-dimensional reference frame (u,v) for the image plane Π with origin O and the u and v axes aligne as x and y respectively. Fianlly, let P=(x,y,z) adn p=(u,v). By means of simple considerations on the similarity of triangle we can

write

$$\frac{f}{z} = \frac{u}{x} = \frac{v}{y} \Rightarrow u = \frac{fx}{z}, \qquad v = \frac{fy}{z}$$

This is the perspective projection. The mapping from 3D coordinates to 2D coordinates is clearly not linear. However, using **homogeneous coordinates** instead allows us to obtain linear equations. Let:

$$\tilde{p} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad \tilde{P} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The projection equation can be writtten as:

$$\begin{bmatrix} \lambda\, u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Where λ is equal to the third coordinate of P.
General model. A realistic camera model that desccribes the transformation from 3D coordinates to pixel coordinates must also take into account the pixelization adn the rrigid body transformation between the camera and the scene. The pixelization takes into account that the camera optical ccenter has pixel ccoordinates (u_0, v_0) w.r.t. the upper left corner of the image, which is ccommonly assumed as origin of the image coordinate system; the coordinates of a point on the image plane are measured in pixels, so we must introduce a scale facctor; the shape of the pixel is iin general assumed not perfectly squared and therefore we must use two different scale factors k_u, k_v along the horizontal and vertical directions respectively; the u and v axes might not be orthogonal but misaligned of an angle θ. The first three points are addressed by means of the translation of the optical center and the individual rescaling off the u and v axes:

$$u = k_u \frac{fx}{z} + u_0 \qquad u = k_v \frac{fy}{z} + v_0$$

After this update the perspective projection becomes:

$$\begin{bmatrix} \lambda\, u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} fk_u & 0 & u_0 & 0 \\ 0 & fk_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

We can then introduce a rrotation matrix and a translation factor, R and t respectively and obtain that:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + t$$

$$\begin{bmatrix} \lambda\, u \\ \lambda v \\ \lambda \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

$$\lambda \tilde{p} = A[R|t]\tilde{P}_w$$

where

$$A = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

is the **intrinsic parameter matrix.** and R and t are the **extrinsic parameters. Radial distortion.** The standard model of radial distortion is a transformation from the ideal coordinates to the observable coordinates. Depending on the type of radial distortion, the coordinates in the observed image are dispersed aqay or torward the image center. The amount of distortion of the coordinates of the observed image is a nonlinear function of their radial distance r. For most lenses, a simpe quadratic model of distortion produces good results:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = (1 + k_1 r^2) \begin{bmatrix} u - u_0 \\ v - v_0 \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \qquad r^2 = (u - u_0)^2 + (v - v_0)^2$$

and k_1 is the radial distortion parameter, which can be estimated by camera calibration. The radial distortion parameter is also an intrinsic parameter of the camera.
Camera calibration consists in measuring accurately the intrinsic and extrinsic parameters of the camera mdel. As these parameters govern the way the scene points are mapped to their corresponding image points, the idea is that by knowing the pixel coordinates of the image points $\tilde{p}$ and the 3D coordinates of the corresponding scene points $\tilde{P}$, it is possible to compite the unknown parameters A, R, t by solving the perspective projection equation.

Feature Extraction Basen on Range Data

Line Fitting

Geometric feature fitting is usually the process of comparing and matchinng measured sensor data against a predefined description, or template, of the expected feature. Usually, the system is overdetermined iin that the number of sensor measurements exceeds the number of feature parameters to be estimated. Since the sensor measurements all have some errr, there is no perfectlu consistent solution and the problem is one of optimization.
Probabilistic line fitting from uncertain range sensor data. Our goal is to fit a line to a set of sensor measurements. There is uncertainty associated with each of the noisy range sensor measurements, and so there is no single line that passes through the set. Instead, we wish to select the best possible match, given some optimization criterion.. More formally, suppose n ranging measurement points in polar coordinates $x_i = (\rho_i, \theta_i)$ are produce by the robot's sensors. We know that there is uncertainty associated with each measurement, andd so we can model each measurement using two random variables $X_i = (P_i, Q_i)$. In this anaysis we assume that the uncertainty with respect to the vaalue of P andd Q is independent. We can state this formally:

$$\mathbb{E}[P_i P_j] = \mathbb{E}[P_i]\mathbb{E}[P_j] \qquad \forall i, j = 1, ...n | i \neq j$$

$$\mathbb{E}[Q_i Q_j] = \mathbb{E}[Q_i]\mathbb{E}Q_j] \qquad \forall i, j = 1, ...n | i \neq j$$

$$\mathbb{E}[P_i Q_j] = \mathbb{E}[P_i]\mathbb{E}Q_j] \qquad \forall i, j = 1, ...n$$

furthermore, we assume that each random variable is subject to a Gaussian probability density curve, with a mean at the true value and with some specified variance:

$$P_i \sim N(\rho_i, \sigma^2_{\rho_i}) \qquad Q_i \sim N(\theta_i, \sigma^2_{\theta_i})$$

Givven some measurement point (ρ, θ), we can calculate the corresponding Euxlidean coordinates $x = \rho\cos\theta$ and $y = \rho\sin\theta$. If there were no error, we would want to find a line for which all measurements lie on that line:

$$\rho\cos\theta\cos\alpha + \rho\sin\theta\sin\alpha - r = \rho\cos(\theta - \alpha) - r = 0$$

Of course, there is measurement error, and so this quantity will not be zero. When it is nonzero, this is a measure of the error between the measurement point (ρ, θ) and the line, specifically in terms of the minimum orthogonal distance between the point and the line. It is always important to understand how the error that shall be minimized is being measured. For a specific (ρ_i, θ_i), we can write the orthogonal distance d_i between (ρ_i, θ_i) and the line as $\rho_i\cos(\theta_i - \alpha) - r = d_i$. If we consider eacch measurement to be equally uncertain, we can sum the square of all errors together, for all measurement points, to quantify an overall fit between the line and all of the measurements:

$$S = \sum_i d_i^2 = \sum_i (\rho_i\cos(\theta_i - \alpha) - r)^2$$

Our goal is to minimize S when selecting the line parameters (α, r). We can do so by solving the nonlinear equation system

$$\frac{\partial S}{\partial \alpha} = 0 \qquad \frac{\partial S}{\partial r} = 0$$

This formalism is conssiidered an **unweighted least-square** solution because no distinction is made from among the measurements. In reality, each sensor measurement may have its own, unique uncertainty based on the geometry of the robot and enviironment when the measurement was recorded.

Propagation of uncertainty during line fitting. We would like to understand how the uncertainties of specific range sensor measurements propagate to govern the uncertainty of the extraccted line. In other words, how uncertainty in ρ_i and θ_i propagate in the above equations and how they affect the uncertainty of r and α. We want to derive the 2x2 outpput covariance matrix

$$C_{AR} = \begin{bmatrix} \sigma_A^2 & \sigma_{AR} \\ \sigma_{AR} & \sigma_R^2 \end{bmatrix}$$

given the 2nx2n input covariance matrix

$$C_X = \begin{bmatrix} diag(\sigma_{\rho_i}^2) & \mathbf{0} \\ \mathbf{0} & diag(\sigma_{\theta_i}^2) \end{bmatrix}$$

Then by calculating the jacobian

$$F_{PQ} = \begin{bmatrix} \frac{\partial\alpha}{\partial P_1} \frac{\partial\alpha}{\partial P_2} \cdots \frac{\partial\alpha}{\partial P_n} \frac{\partial\alpha}{\partial Q_1} \frac{\partial\alpha}{\partial Q_2} \cdots \frac{\partial\alpha}{\partial Q_n} \\ \frac{\partial r}{\partial P_1} \frac{\partial r}{\partial P_2} \cdots \frac{\partial r}{\partial P_n} \frac{\partial r}{\partial Q_1} \frac{\partial r}{\partial Q_2} \cdots \frac{\partial r}{\partial Q_n} \end{bmatrix}$$

we can instantiate the uncertainty propagatio to yield $C_AR = F_{PQ}C_XF_{PQ}^T$.

Six line-extraction algorithms

The process of divinding up a seto of measurements into subsets that can be interpreted one by one is termed **segmentation** and is the most important step of line extraction.

Algorithm 1: *Split-and-Merge*

1. Initial: set s_1 consists of N points. Put s_1 in a list L
2. Fit a line to the next set s_i in L
3. Detect point P with maximum distance d_P to the line
4. If d_P is less than a threshold, continue (go to step 2)
5. Otherwise, split s_i at P into s_{i1} and s_{i2}, replace s_i in L by s_{i1} and s_{i2}, continue (go to 2)
6. When all sets (segments) in L have been checked, merge collinear segments.

I **Slit-and-merge.** The algortihm chan be modified on line 3 to make it more robust to noise. Indeed, sometimes the splitting position can be the result of a point which still belongs to the same line but which, because of noise, appears too far away from this line. In this case, we can scan for a splitting position where two adjacent points P_1 and P_2 are on the same side of the line and both have distances to the line greater than the threshold. If we find only such point, then we automatically discard it as a noisy point.

II **Line regression.** It uses a sliding window of size N_f. At every step, a line is fittend to

Algorithm 2: *Line-Regression*

1. Initialize sliding window size N_f
2. Fit a line to every N_f consecutive points
3. Compute a line fidelity array. Each element of the array contains the sum of Mahalanobis distances between every three adjacent windows
4. Construct line segments by scanning the fidelity array for consecutive elements having values less than a threshold
5. Merge overlapped line segments and recompute line parameters for each segment

the N_f points within the window. The window is then shifted one point forward, and the line fitting operation is repeated again. The goal is to find adjacent line segments and merge them together. To do this, at every step the **Mahalanobis** diistance between the last two windows is ccomputed and is stored in a **fidelity array**. When all the points have been analyzed, the fidelity array is scanned for consecutive similar elements. This is done by using an appropriate clustering algorithm. At the end, the clustered consecutive line segments are merged together using line regression. Notice tht the sliding window size N_f is very dependent on the environment and has a strong influence on the algorithm performance. Typically $N_f = 7$ is used.

III **Incremental.** At the beginning, the set consists of two points. Next, an extra point is added to this set and a line is constructed. If the line satisfies a predefined line condition, then a new point is added to the set and the procedure is repeated. If the line does not verify the line condition, the new point is put bacck and the line is computed from all previous visited points. At this point, the procedure starts again with the two new following points.

IV **RANSAC.** RANSAC (Random Sample Consensus) is an algorithm to estimate robustly the parameters of a model from a given data in the presence of **outliers**. Outliers are data that

Algorithm 3: *Incremental*

1. Start by the first 2 points, construct a line
2. Add the next point to the current line model
3. Recompute the line parameters by line fitting
4. If it satisfies the line condition, continue (go to step 2)
5. Otherwise, put back the last point, recompute the line parameters, return the line
6. Continue with the next two points, go to step 2

Algorithm 4: *RANSAC*

1. Initial: let A be a set of N points
2. **repeat**
3. Randomly select a sample of 2 points from A
4. Fit a line through the 2 points
5. Compute the distances of all other points to this line
6. Construct the inlier set (i.e. count the number of points with distance to the line $< d$)
7. Store these inliers
8. **until** Maximum number of iterations k reached
9. The set with the maximum number of inliers is chosen as a solution to the problem

do not fit the model. Such outliers can be due to high noise in the data, wrong measurements, or they can more simply be points which come from other objects for which our mathematical model does not apply. RANSAC is an iterative method and is nondeterministic in that the probability to find a line free of outliers increases as more iterations are used. RANSAC is not restricted to line extraction from laser data but it can be more generally applied to any problem where the goal is to identify the inliers which satisfy a predefined mathematical model.

The algorithm starts by randomly selectin a sample of two points from the dataset. Then a line is constructed from these two points and the distance off all other points to this line is computed. The inliers set comprises all the points whose distance to the line is within a predefined threshold d. The algorithm then stores the inliers set and starts again by selecting another minimal set of two points at random. The procedure is iterated until a set with a maximum number of inliers is found, which is chosen as a solution to the problem. Because we cannot know in advance if the observed set contains the maximum number of inliers, the ideal would e to check all possible combinations of 2 points in a dataset of N point, which makes it computationally unfeasible if N is tooo large. Using a probabilistic approach we do not need to check all combinations but just a subset of them if we have a rough estimate of the percentage of inliers in our dataset.

Let p be the probability of finding a set of points free of outliers. Let w be the probability of selectin an inlier from our dataset of N points. Hence, w expresses the fraction of inliers in the data, that is w=number of inliers/N. If we assume that the two points needed for estimating

a line arre selected independently, w^2 is the probability that both points are inliers and $1-w^2$ is the probability that at least one of these two points is an outlier. Now, let k be the number of RRANSACiterations executed so far, then $(1-w^2)^k$ will be the probability that RANSAC never selects two points that are both inliers. This probability must be equal to 1-p. Accordingly $1-p=(1-w^2)^k \Rightarrow k=\frac{\log(1-p)}{\log(1-w^2)}$. the problem of RANSAC is that after k iteration the soulution we found might not be the optimal one.

V **Hough Transform.**

Algorithm 5: *Hough Transform*

1. Initial: let A be a set of N points
2. Initialize the accumulator array by setting all elements to 0
3. Construct values for the array
4. Choose the element with max. votes V_{max}
5. If V_{max} is less than a threshold, terminate
6. Otherwise, determine the inliers
7. Fit a line through the inliers and store the line
8. Remove the inliers from the set, go to step 2

VI **Expectation maximization.** It is a probabilistic method commonly used in missing variable problems. There are some drawbacks: first, it can fall into local minima; second, it is difficult to choose a god initial value.

Algorithm 6: *Expectation Maximization*

1. Initial: let A be a set of N points
2. **repeat**
3. Randomly generate parameters for a line
4. Initialize weights for remaining points
5. **repeat**
6. *E-Step*: Compute the weights of the points from the line model
7. *M-Step*: Recompute the line model parameters
8. **until** Maximum number of steps reached or convergence
9. **until** Maximum number of trials reached or found a line
10. If found, store the line, remove the inliers, go to step 2
11 Otherwise, terminate

Implemetation details: we use **clustering** to agglomerate a few sparse points. We use **merging** since due to occlusions, a line may be observed and extracted as several segments. To decide if ttwo consecutive line segments have to be merged, the mahalanobiss distance between each pair of line segments is typically used. If the distance is less than a predefined threshold, then they are merged.

Comparison of the algorithms:

	Complexity	Speed [Hz]	False positives	Precision
Split-and-Merge	$N \cdot \log N$	1500	10%	+++
Incremental	$S \cdot N^2$	600	6%	+++
Line-Regression	$N \cdot N_f$	400	10%	+++
RANSAC	$S \cdot N \cdot N_{Trials}$	30	30%	++++
Hough-Transform	$S \cdot N \cdot N_C + S \cdot N_R \cdot N_C$	10	30%	++++
Expectation Maximization	$S \cdot N_1 \cdot N_2 \cdot N$	1	50%	++++

Range histogram features

A histogram is a simple way to combine characteristic elements of an image. An angle histogram plots the statistics of lines extracted by two adjacent range emasurements. First, a 360 degree scan of the room is taken with the range scanner, and the resulting hits are recorded in a map. Then the allgorithm measures the relative angle between any two adjacent hits. After compensating for noise in the readings the angle histogram can be built. The uniform direction of the main walls are clearly visible as peaks in the angle histogram. Detection of peaks yields only two main peaks: one for each pair of parallel walls. This algorithm is very robust with regard to opening in the walls, such as doors and windows, or even cabinets lining the walls.

Extracting other geomettic features

Line features are of particular value for mobile robotis operating in man-made envvironments. For indoor mobile robots, the line feature is cetainly a member of the optimal feature set. In addition, other geometric kernels consistently appear throughout the indoor manmade environment. **corner features** are defined as a point feature with an orientation. **Step discontinuities** are characterize by their form and step size. **Doorways** are characterized by their width.

Navigation, Localization and Mapping

Localization and Mapping from notes

Localization is the problem of estimating the robot's position given a map of the environment and a sequence of sensor readings. The problem classes are position tracking, global localization and kidnapped robot problem. **The SLAM problem** is the problem of computing the robot's pose and the map of the environment at the same time. **Mapping** means building a map giving the robot's location. Now we define the localization problem: Given the robot controls $u_{1:T} = \{u_1, ..., u_T\}$ and observations $z_{1:T} = \{z_1, ..., z_T\}$ we want to find a Path, or current position, of the robot $x_{0:T} = \{x_0, ..., x_T\}$.
In localization and SLAM, motion controls are used, like from controls sent to the actuators estimate the angular and translational velocity or when wheel encoders are available use odometry (that is actually an output) ad an input control.
Observaations include range scans, 3D scans or point clouds. Maps can be Land-mark based or volumetric (e.g. grid-based). to model the uncertainty in the robot's motions and aboservation we use the probaability theory to explicitly represent the uncertainty.

Probability recap

A discrete random vriable X can take on a countable number of values, e.g. $\{x_1, ..., x_n\}$. $\mathbb{P}(X = x_1)$, or $\mathbb{P}(x_i)$, is the probability that the random variable X takes on value x_i, e.g. $\{0.1, 0.3, ..., 0.05\}$. $\mathbb{P}(.)$ is called probablity mass function, with $\sum_x \mathbb{P}(x) = 1$. A continuous random variable X can take values in a continuous sace. $p(X = x_i)$, or $p(x_i)$, is a probability density function:

$$\mathbb{P}(x \in (a, b)) = \int_a^b p(x)dx \qquad \int p(x)dx = 1$$

$\mathbb{P}$(X=x and Y=y)=$\mathbb{P}$(x,y). If X and Y are independent then $\mathbb{P}$(x,y)=$\mathbb{P}$(x)$\mathbb{P}$(y). $\mathbb{P}(x|y)$ is the probability of x given y: $\mathbb{P}(x|y) = \mathbb{P}(x, y)/\mathbb{P}(y) \Rightarrow \mathbb{P}(x, y) = \mathbb{P}(x|y)\mathbb{P}(y)$. If X and Y are independet then $\mathbb{P}(x|y) = \mathbb{P}(x)$. Marginalization just follows the properties we have shown:

$$\mathbb{P}(x) = \sum_y \mathbb{P}(x, y) = \sum_y \mathbb{P}(x|y)\mathbb{P}(y) \qquad p(x) = \int p(x, y)dy = \int p(x|y)p(y)dy$$

Bayes formula allos us to obtain an unknown target probability density in terms of other, possibly known, probability densities: $\mathbb{P}$(x,y)=$\mathbb{P}(x|y)\mathbb{P}(y) = \mathbb{P}(y|x)\mathbb{P}(x) \Rightarrow \mathbb{P}(x|y) = \frac{\mathbb{P}(y|x)\mathbb{P}(x)}{\mathbb{P}(y)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}$. To remove the evidence we can use normalization:

$$\mathbb{P}(x|y) = \frac{\mathbb{P}(y|x)\mathbb{P}(x)}{\mathbb{P}(y)} = \eta\mathbb{P}(y|x)\mathbb{P}(x)$$

$$\eta = \mathbb{P}(y)^{-1} = \frac{1}{\sum_x \mathbb{P}(y|x)\mathbb{P}(x)}$$

In case of more conditions we can still apply the whole thing:

$$\mathbb{P}(x|y,z) = \frac{\mathbb{P}(y|x,z)\mathbb{P}(x|z)}{\mathbb{P}(y|z)}$$

Recursive Bayesian updating: given a stream of observations z=$\{z_1, ..., z_t\}$, how can we estimate $\mathbb{P}(x|z_1, ..., z_t)$? We can use the Bayesian rule:

$$\mathbb{P}(x|z_1, ..., z_n) = \frac{\mathbb{P}(z_n|x, z_1, ..., z_{n-1})\mathbb{P}(x|z_1, ..., z_{n-1})}{\mathbb{P}(z_n|z_1, ..., z_{n-1})}$$

$$= \eta\mathbb{P}(z_n|x)\mathbb{P}(x|z_1, ..., z_{n-1}) = \eta_{1,...,n} \prod_{1=1,...,} \mathbb{P}(z_i|x)\mathbb{P}(x)$$

Actions: the robot turns its wheels to move, uses its manipualtor to grasp objeccts and so on. How can we incorporate such actions u=$\{u_1, ..., u_t\}$,i.e. $p(x|u)$? We define a new probability density, called **state transitions** $p(x|u, x')$ where x' is the prevvious state. For actions we can use marginaliation, so we can write:

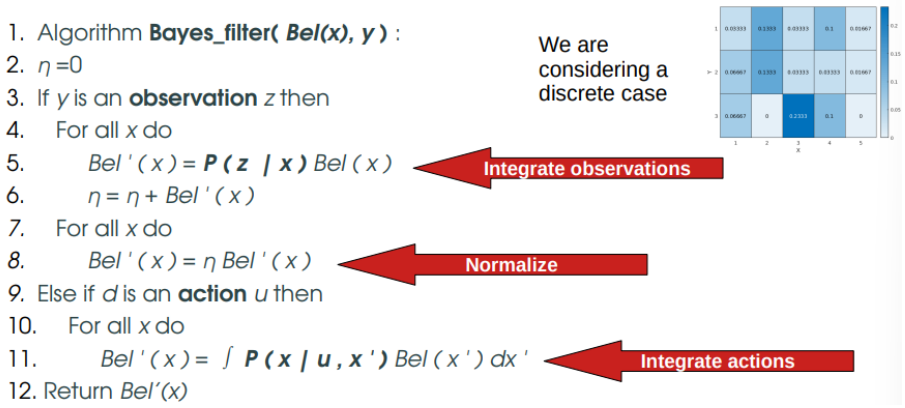
$$\mathbb{P}(x|u) = \int \mathbb{P}(x|u, x')\mathbb{P}(x')dx' \qquad \mathbb{P}(x|u) = \sum \mathbb{P}(x|u, x')\mathbb{P}(x')$$

Let us now define the **Probabilistic Location Problem:** given a stream of observations z and action data u: d_t=$\{u_1, z_1, ..., u_t, z_t\}$, we want to estimate the robot position X as $\mathbb{P}(x_t|u_1, z_1, ..., u_t, z_t)$. this posterior of state is also called **belief.**
A variable x_t depends only on its direct predecessor stte x_{t-1} and on the lates action u_t,i.e. $p(x_t|x_{1:t-1}, z_{1:t}, u_{1:t}) = p(x_t|x_{t-1}, u_t)$ and $p(z_t|x_{0:t}, z_{1:t}, u_{1:t}) = p(z_t|x_t)$. We also assume that the world is static.

$$\begin{aligned}
Bel(x_t) &= P(x_t \mid u_1, z_1 \ldots, u_t, z_t) \\
\text{Bayes} \quad &= \eta\ P(z_t \mid x_t, u_1, z_1, \ldots, u_t)\ P(x_t \mid u_1, z_1, \ldots, u_t) \\
\text{Markov} \quad &= \eta\ P(z_t \mid x_t)\ P(x_t \mid u_1, z_1, \ldots, u_t) \\
\text{Total prob.} \quad &= \eta\ P(z_t \mid x_t) \int P(x_t \mid u_1, z_1, \ldots, u_t, x_{t-1}) \\
&\qquad P(x_{t-1} \mid u_1, z_1, \ldots, u_t)\, dx_{t-1} \\
\text{Markov} \quad &= \eta\ P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\ P(x_{t-1} \mid u_1, z_1, \ldots, u_t)\, dx_{t-1} \\
\text{Markov} \quad &= \eta P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\, P(x_{t-1} \mid u_1, z_1, \ldots, z_{t-1})\, dx_{t-1} \\
&= \eta\ P(z_t \mid x_t) \int P(x_t \mid u_t, x_{t-1})\ Bel(x_{t-1})\, dx_{t-1}
\end{aligned}$$

Bayes Filter Aglorithm

In the bayes filter algorithm, we used two probabilities densities to update the belief: **Observation, or Sensor Model** $\mathbb{P}(z_t|x_t)$ and **action model** $\mathbb{P}(x_t|u_t, x_{t-1})$. When the action is the movement



of the robot, the action model is called **Motion Model.** The observation model models the uncertainty of the observations, i.e., the probability of a measurement z_t given that the robot is at position x_t (i.e. $p(z_t|x_t)$). An example on how to estimate the sensor model density is the **Beam-based Proximity Model:** put the sensor at several known distances from some obstacles and collect sensor measurements.
The motion Model is the model that models the uncertainty of the motion $p(x_t|x_{t-1}, u_t)$. To estimate the motion model density we have 3 steps:

I. Move the robot from position A to position B, and colect the relative motion from the odometry.

II. Measure and collect the actual travelled distancce with a meter, and repeat II for several trials.

III. Estimate the density parameters given the estimated and travelled distances.

How do we represent the state? so far we used a discrete way, or grid based, which is not parametric. But then, why not in an analytical way? We could define the Prediction as $\bar{\text{bel}}(x_t) = \int p(x_t|u_t, x_{t-1})\text{bel}(x_{t-1})dx_{t-1}$ and the correction as $\text{bel}(x_t) = \eta p(z_t|x_t)\bar{\text{bel}}(x_t)$.
Covariance Matrix: given a n-dimensional randdom vector X with Gaussian density, define μ_k the expected value(mean) of the k^{th} element of X, i.e., $\mu_k = \mathbb{E}[X_k]$, with $\mu = \mathbb{E}[X]$. The covariance matrix is defined as:

$$\sum = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^T] =$$

$$= \begin{bmatrix} \mathbb{E}[(X_1 - \mu_1)(X_1 - \mu_1)^T] & \mathbb{E}[(X_1 - \mu_1)(X_2 - \mu_2)^T] & \dots & \mathbb{E}[(X_1 - \mu_1)(X_n - \mu_n)^T] \\ \mathbb{E}[(X_2 - \mu_2)(X_1 - \mu_1)^T] & \mathbb{E}[(X_2 - \mu_2)(X_2 - \mu_2)^T] & \dots & \mathbb{E}[(X_2 - \mu_2)(X_n - \mu_n)^T] \\ \vdots & \vdots & \vdots & \vdots \\ \mathbb{E}[(X_n - \mu_n)(X_1 - \mu_1)^T] & \mathbb{E}[(X_n - \mu_n)(X_2 - \mu_2)^T] & \dots & \mathbb{E}[(X_n - \mu_n)(X_n - \mu_n)^T] \end{bmatrix}$$

The (i,i) element is the variance of X_i, wile the element (i,j) is the covariance between X_i and X_j: if positive, if X_i increases, also X_j tends to increase; if negative, if X_i increases, X_j tends to decrease and vice versa; if zero X_i and X_j are independent hence uncorrelated.
Properties of Gaussians: Linear combinations of Gaussian random variables are still Gaussians:

$$Y \sim N(A\mu + B, A\Sigma A^T) \Leftarrow \begin{cases} X \sim N(\mu, \Sigma) \\ Y = AX + B \end{cases}$$

products of Gaussian probability density functions are still Gaussian PDF:

$$p(X_1) \cdot p(X_2) \sim N\Big(\frac{\Sigma_2}{\Sigma_1+\Sigma_2}\mu_1 + \frac{\Sigma_1}{\Sigma_1+\Sigma_2}\mu_2, \frac{1}{\Sigma_1^{-1}+\Sigma_2^{-1}}\Big) \Leftarrow \begin{cases} X_1 \sim N(\mu_1, \Sigma_1) \\ X_2 \sim N(\mu_2, \Sigma_2) \end{cases}$$

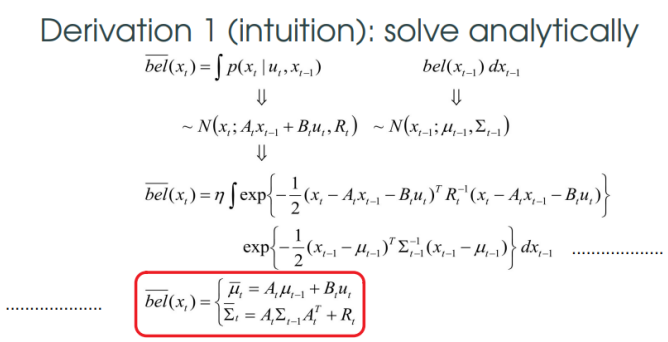
Chain Rule and linear combinations of Gaussian variables/vectors: the general chain rule is $\mathbb{P}(X,Y) = \mathbb{P}(X|Y)\mathbb{P}(Y)$. Now, given $p(x_a) = \mathcal{N}(x_a; \mu_a, \Sigma_a)$ and $p(x_b|x_a) = \mathcal{N}(x_b; Ax_a + c, \Sigma_{b|a})$ we want to compute $p(x_a, x_b) = \mathcal{N}(x_{a,b}; \mu_{a,b}, \Sigma_{a,b})$. The parameters are:

$$\mu_{a,b} = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} = \begin{pmatrix} \mu_a \\ A\mu_a + c \end{pmatrix}$$

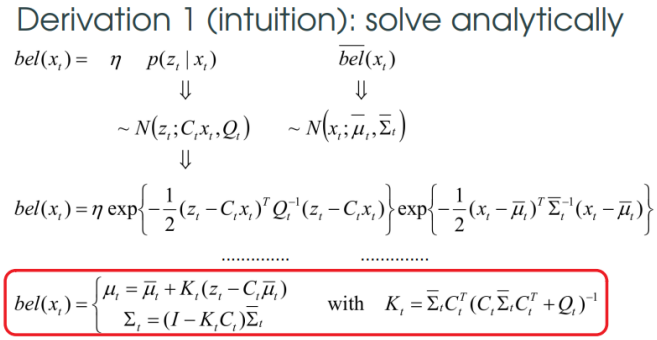
$$\Sigma_{a,b} = \begin{pmatrix} \Sigma_a & \Sigma_a A^T \\ A\Sigma_a & \Sigma_{b|a} + A\Sigma_a A^T \end{pmatrix}$$

Kalman Filter

It estimates the state x of a discrete-time controlled process that is governed by a linear (or linearized) stochasticc equation: $x_T = A_t x_{t-1}$, where A_t is a nxn matrix that describes how the state evvolves from t-1 to t without controls or noise, even tho in many cases is just an identity matrix. We can then itroduce another teerm to get $x_t = A_t x_{t-1} + B_t u_t$, where B_t is a nxl matrix that describes how the control u_t changes the state from t-1 to t. We then can introduce also another last term, getting $x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$, where ϵ_t is a normally distributed random variable representing the process noise with covariance R_t. It has stochastic measurements: "Given a landmark with known position, i want to calculate the measurement i would get from my sensor given the current robot position x_t": $z_t = C_t x_t + \sigma_t$, where C_t is a kxn matrix that describes how to map the state x_t to an observation z_t and δ_t is a normally distributed random variable representing the measurement noise with covariance Q_t. So, all the prevvious components become $p(x_t|u_t, x_{t-1}) = N(x_t; \underbrace{A_t X_{t-1} + B_t u_t}_{\mu_t}, R_t)$ and $p(z_t|x_t) = N(z_t; \underbrace{C_t x_t}_{mu_z}, Q_t)$. Given a Gaussian belief at time t-1, how to inccorporate actions and sensor measurements to obtain the belief at time t? The kalman filter algorithm then becomes:



(a) Prediction factor



(b) Correction factor

$Kalman_filter(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

Prediction:

$$\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$$
$$\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$$

Correction

$$K_t = \bar{\Sigma}_t C_t^T (C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$$
$$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$$

Return μ_t, Σ_t

There is an example on the slides which i'm gonna omit cause it is long and full of math content :) The major problem is that there is no linear relationship neither between x_{t-1}, u_t and x_t and neither between x_t and z_t. So, Given $x \in \mathbb{R}$ and a general multi-dimensional function f(x):$\mathbb{R}^n \to \mathbb{R}^m$, how do we linearize f(x)? We ccan use the Jacobiann matrix (the idea is to approximate f(x) with Jx). The kalman filter algorithm then becomes:

$Extended_Kalman_filter(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

Prediction:

$$\bar{\mu}_t = g(\mu_{t-1}, u_t)$$
$$\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + R_t$$

Correction

$$K_t = \bar{\Sigma}_t H_t^T (H_t\bar{\Sigma}_t H_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t + -h(\mu_t))$$
$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

Return μ_t, Σ_t

where $H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}$ and $G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}}$.

1. EKF_localization ($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, m$):

Prediction:

2. $G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} = \begin{pmatrix} \frac{\partial x'}{\partial \mu_{t-1,x}} & \frac{\partial x'}{\partial \mu_{t-1,y}} & \frac{\partial x'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial y'}{\partial \mu_{t-1,x}} & \frac{\partial y'}{\partial \mu_{t-1,y}} & \frac{\partial y'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial \theta'}{\partial \mu_{t-1,x}} & \frac{\partial \theta'}{\partial \mu_{t-1,y}} & \frac{\partial \theta'}{\partial \mu_{t-1,\theta}} \end{pmatrix}$ Jacobian of *g* w.r.t location

3. $V_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial u_t} = \begin{pmatrix} \frac{\partial x'}{\partial v_t} & \frac{\partial x'}{\partial \omega_t} \\ \frac{\partial y'}{\partial v_t} & \frac{\partial y'}{\partial \omega_t} \\ \frac{\partial \theta'}{\partial v_t} & \frac{\partial \theta'}{\partial \omega_t} \end{pmatrix}$ Jacobian of *g* w.r.t control

4. $M_t = \begin{pmatrix} (\alpha_1 |v_t| + \alpha_2 |\omega_t|)^2 & 0 \\ 0 & (\alpha_3 |v_t| + \alpha_4 |\omega_t|)^2 \end{pmatrix}$ Motion noise

5. $\bar{\mu}_t = g(u_t, \mu_{t-1})$ Predicted mean

6. $\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + V_t M_t V_t^T$ Predicted covariance

(a) Prediction factor of EKF_localization

1. EKF_localization ($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, m$):

Correction:

2. $\hat{z}_t = \begin{pmatrix} \sqrt{(m_x - \bar{\mu}_{t,x})^2 + (m_y - \bar{\mu}_{t,y})^2} \\ \text{atan2}(m_y - \bar{\mu}_{t,y}, m_x - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{pmatrix}$ Predicted measurement mean

3. $H_t = \frac{\partial h(\bar{\mu}_t, m)}{\partial x_t} = \begin{pmatrix} \frac{\partial r_t}{\partial \bar{\mu}_{t,x}} & \frac{\partial r_t}{\partial \bar{\mu}_{t,y}} & \frac{\partial r_t}{\partial \bar{\mu}_{t,\theta}} \\ \frac{\partial \varphi_t}{\partial \bar{\mu}_{t,x}} & \frac{\partial \varphi_t}{\partial \bar{\mu}_{t,y}} & \frac{\partial \varphi_t}{\partial \bar{\mu}_{t,\theta}} \end{pmatrix}$ Jacobian of *h* w.r.t location

4. $Q_t = \begin{pmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_r^2 \end{pmatrix}$

5. $S_t = H_t\bar{\Sigma}_t H_t^T + Q_t$ Pred. measurement covariance

6. $K_t = \bar{\Sigma}_t H_t^T S_t^{-1}$ Kalman gain

7. $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$ Updated mean

8. $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$ Updated covariance

(b) Correction factor of EKF_localization

Particle filters

Represent belief by random samples. Allows us to estimate non-Gaussians and nonlinear processes. It follows the following idea: **More the samples (particles) are dense in a neighborhood, more the probability is high in such neighborhood.**

Start drawing particles from an uniform distribution over the whole space domain. For each

particle, compute the **importance weight** directly from the sensor model $p(z_t|x_t)$. For each particle x(i), sample a new particle x(i)' by using the motion model $p(x_t|u_t, x_{t-1})$. Resample a new generation of particles in accordance with their importance weight: **particles with high weigths will be easily duplicated, vice versa will be easily deleted**. The Particle Filter Algorithm then becomes:

$$\text{Bel}(x_t) = \eta p(z_t|x_t) \int p(x_t|x_{t-1}, u_{t-1})\text{Bel}(x_{t-1})dx_{t-1}$$

Tje localization with Particle filters works as follows: (Particle are **pose hypotheses**)

I. Update each pose hypotheses with the more recent motion.

II. Weight each pose hypotheses with the more recent sensor reading.

III. Resample in accordance with their importance weigth.

IV. Repeat from I.

SLAM

We want to eastimate the robot's path and the map with $p(x_{0:T}, m|zz_{1:T}, u_{1:T})$. but why is SLAM a hard problem? The mapping between observations andd the map is unknown. Picking wrong data associations can have catastrophic consequences. The main paradigms for SLAM are the Kalman filter, Particle filter or graph-based.
SLAM with Kalman Filter: If we assume that all the involved PDFs are Gaussians, we can solvve the SLAM problem with the Kalman Filter. We can write the motion model from $x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$ and $p(x_t|u_t, x_{t-1})$, and we can write the Sensor model from $z_t = C_t x_T + \delta_t$ and $p(z_t|x_t)$. The Landmarrk-based online SLAM consists in adding landamrk position $I_i = (x_i, y_i)$ directly to the basic localization state, and update them. A map with N-landmarks has (3+2N)-dimensional Gaussian density and can handle only up to a few hundred landmarks. The state size is not fixed: everytime a new landmark j is detected, the state should be expanded to insert the new position I_j. More importantly **a perfect data association is essential.**
SLAM with Particle Filter: particles are full trajectories hypotheses. Update each trajectory with the more recent motion, and weight each trajectory with the more recent sensor reading. Trajectories with high weight will be easily duplicated, viceversa will be easily removed.
Graph-based SLAM: it is a full SLAM solution. Landmarks and robot locations as nodes of a graph. Pair of consecutive locations x_i and x_{i+1} are linked by an edge. If the landmark m_j is observed by location x_i, they are linked by an edge. Each edge represents a conditional dependency, i.e. a sor of constraint. An intuitive derivation of a basic landmark based graph SLAM solution is provided here:

$$\text{definition:} p(x_{1:t}) := p(x_1, ..., x_t) \qquad \text{SLAM target density} -> p(x_{1:t}, m|z_{1:n}, u_{1:t-1})$$

$$\textbf{Bayes rule} = \frac{p(z_{1:n}|x_{1:t}, m, u_{1:t-1})p(x_{1:t}, m|u_{1:t-1})}{p(z_{1:n}|u_{1:t-1})}$$

$$\textbf{Normalization} = \eta p(z_{1:n}|x_{1:t}, m, u_{1:t-1})p(x_{1:t}, m|u_{1:t-1})$$

$$\textbf{Markov} = p(z_1|x_{1:t}, m, u_{1:t-1})p(z_2|x_{1:t}, m, u_{1:t-1})...p(z_n|x_t, m, u_{1:t-1})$$

$$\textbf{Markov} = p(z_1|x_{i_{z_1}}, m)p(z_2|x_{i_{z_2}}, m)...p(z_n|x_{i_{z_n}}, m)$$

$$\textbf{Chain rule} = p(x_t|x_{1:t-1}, m, u_{1:t-1})p(x_{1:t-1}, m|u_{1:t-1})$$

$$\textbf{Markov} = p(x_t|x_{1:t-1}, u_{1:t-1})p(x_{1:t-1}, m|u_{1:t-2})$$
$$\textbf{Markov} = p(x_t|x_{t-1}, u_{t-1})p(x_{1:t-1}, m|u_{1:t-2})$$

By putting everything together, we obtain:

$$= p(x_t|x_{t-1}, u_{t-1})p(x_{t-1}|x_{t-2}, u_{t-2})p(x_{t-2}|x_{t-3}, u_{t-3})...\underbrace{p(x_0)}_{\text{prior}}$$

$$= \eta p(z_{1:n}|x_{1:t}, m, u_{1:t-1})p(x_{1:t}, m|u_{1:t-1})$$

$$= \eta p(z_1|x_{i_{z_1}}, m)p(z_2|x_{i_{z_2}}, m)...p(z_n|x_{i_{z_n}}, m)p(x_t|x_{t-1}, u_{t-1})p(x_{t-1}|x_{t-2}, u_{t-2})p(x_{t-2}|x_{t-3}, u_{t-3})...p(x_0)$$

In other word, solving a Graph-based SLAM problem is equivalent to a maximum likelihood approach to find:

$$x^*_{1:t}, m^* = \arg\max_{x_{1:t}, m} p(x_{1:t}, m|z_{1:n}, u_{1:t-1})$$

Which is equivalent to minimize the negative natural logarithm probability;

$$x^*_{1:t}, m^* = \arg\min_{x_{1:t}, m} (-ln(p(x_{1:t}, m|z_{1:n}, u_{1:t-1})))$$

After some tedious math procedure, you can formulate the SLAM problem into a least squares approach (squares on residuals are due to Gaussian assumption). It is a weigthed least squares problem: less noisy sensor will have more influence in computation of the cost function:

$$x^*_{1:t}, m^* = \frac{1}{2}\sum_{i=1}^{n}(z_i - h(x_{j_{z_i}}, m))^T \sum_{z}^{-1}(z_i - h(x_{j_{z_i}}, m)) + \frac{1}{2}\sum_{i=2}^{t}(x_i - g(x_{i-1}, u_{i-1}))^T \sum_{u}^{-1}(x_i - g(x_{i-1}, u_{i-1}))$$

If the map m is landmark based $m := (m_1, ..., m_k)$ just solve the previous least square problem by means of standard optimization method, like Gauss-Newton or the Levenberg-Marquardt algorithms, by inferring both robot positions and landmark locations. What can be done if the map m is volumetrix, e.g. grid-based, when the cardinality of the map is huge? Include in the state just the robot path, not the map. Solve SLAM by means of an optimization problem that takes into acccount consecutive transformations and loop closure detections given both observations and actions. We want to minimize

$$E(\mathbb{T}) = \sum_i f(T_i, T_{i+1}, R_i) + \sum_{i,j} f(T_i, T_j, T_{ij})$$

then, just render the map given the robot positions.

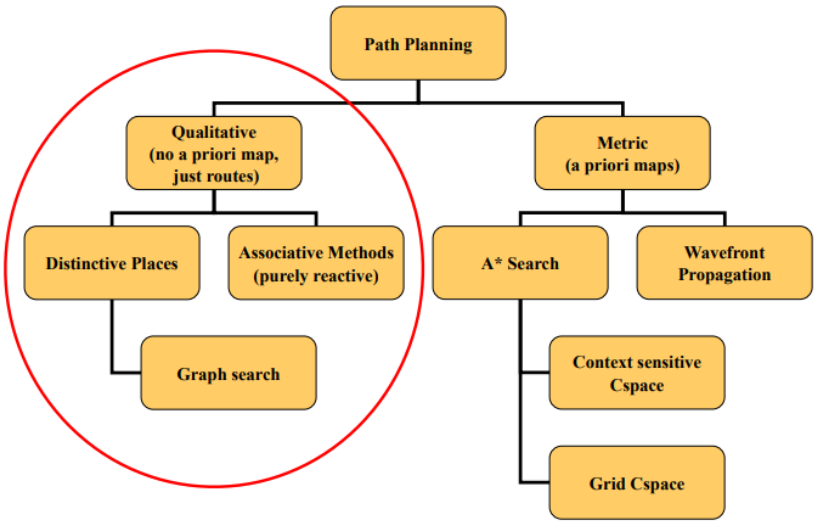
Pose Graph with Scan Matching: for robot with 2D or 3D range finders. As usual, use the odometry to get a prior of the robot between cconsecutive positions. Incrementally align two scans without revising the past/map. Use the iterative closest point (ICP) algorithm to align scans, hence to compute transformations between robot positions.

Visual Odometry VS Visual SLAM. First of all Visual SLAM is the process of incrementally estimating the pose of a camera and a map of the environment by examining the changes that motion induces on the acquired images. Visual Odometry focus on incremental/local consistency. Visual SLAM focus on globally consistent estimation and is composed by visual odometry, loop detection and graph optimization.

Conclusions: SLAM estimates the pose of a robot and the map of the environment at the same time. There are solid and established probabilistic solutions, in particular graph-based. There is still room for improvement, among other things, in terms of scalability and robustness against outliers.

Topological Navigation

Navigation requires spatial memory. The best way to get somewhere depends on the representation of the world. A robot's world representation and how it is maintained over time is its spatial memory: attention, reasoning, path planning and information collection. We have 3 major map models:**Grid-Based** uses a collection of discretized obstacle/free-space pixels; **Feature-Based(distinctivve Objects)** uses a collection of landmark locations and correlated uncertainty; **Topological(distinctive Properties)** uses a ccollection of nodes and their interconnections. The first one is the least abstract, the last one is the most abstract. Grid-Based has discrete localization the computational complexity depends on the grid size and resolution and uses Frontier-based exploration. Feature-Based uses Arbitrary localization, the computational complexity depends on the landmark covariance and has no inherent exploration. Topological localize the nodes, has minimal complexity and uses graph exploration.
Atlas Framework is a hybridd solution since it extracts local features from logical grid map. The local map frames are created at ccomplexity limit and the topology consist of connected local map frames.



Route, or Qualitative Navigation

There are two categories: **Rational** where the spatial memory is a relational graph, also known as a tpological map and uses graph theory to plan paths; or **Associative** where the spatial memry is a series of remembered viewpoints, where each viewpoint is labeled with a location, it is good for retracing step. It is called Quality Navigation becuse you don't have GPS indoors, you don't always have lasers and people don't have to accurately measure distances, just turn right at the end of the all. But qualitative navigation has become less important.
Landmarks: a landmark is one or more perceptually distinctive features of ineterest on an object or locale of interest. A **natural landmark** is a configuration of existing features that wasn't put in the environment to aid with the robot's navigation; An **artificial landmark** is a set of ffeatures added to the environment to support navigation. Ee usually avoid the latters since we do not want to engineer the environment. We want the landmarks to have the following charteristics:

I. Recognizable: Passive, perceivable over the entire range of where the robot might need to view it, and distinctive features should be globally unique, or at least locally unique.

II. Perceivable for the task

III. Be perceivable ffrom many different viewpoints.

Rational methods use graphs and landmarks: the best known relational method is distinctive places, which aare often gateways; local control strategies are behaviors.
Associative methods rememmber places as image signature or a vviewframe extracted from a signature: can't really plan a path, just retrace it; direcct stimulus-response couping by matching signature to current perception.

Representing Area/Volume in Path planning

In quantitative or metric maps we have many different ways to represent an area or volume of space, and the algorithms are graph or network algorithsm or wvefrom or graphics-derived algorithms.
Metric Maps Motivation for having a metric map is often path planning. To determine a path from one point to goal usually consist in find the best or the optimal path. Path planning assumes an a priori map of relevant aspects. A real mobile robot should not be modeled as a point, so to take into account its shape obstacles are enlarged. This might generate some issues and a trade-off is between memory requirements and performance.
World space: physical space robots and obstacles exist in. In order to use, generally need to know (x,y,z) plus Euler angles: 6DOF. The **Configuration Space** is the trnsform space into a representation suitable for robots, simplifying assumptions. Now let us define some things that will help us in the motion planning: The robot configuration R consists in the poses of all points that compose a robot in a certain instant according to a certain coordinate system. The configuration space C is the set off all possible cconfigurations. The ccollision space C_{obs} ares the colliding configurations. The Free space C_{free} are the collision-free configurations. For an accurate collision detection the Configuration space is used. A configuration of an object is a point $q = (q_1, ..., q_n)$ where point q is free if the robot in q does not collide. C-obstacle is the union of all q where the robot collides. C-free is the union of all free q. So, the Cspace is the union of C-free and C-obstacle. Planning can be performed in C-space. In a 2D C-Space a robot can translate in the plane and/or rotate. Obstacles should be expanded according to the robot orientation. Non holonomic constraints can't be C-space obstacles. The idea behind Cspace representation is to reduce physical space to a cspace representation which is more amenable for storage in computers and for rapid execution of algorithms. The major types are Meadow maps, Generalized Voronoi Graphs (GVG) or Regular grids, quadtrees.

Object growing

Since we assume robot is round, we can grow objects by the width of the robot and then consider the robot to be a point. This asusmption greatly simplifies path planning. New representation of objects typically called configuration space object. After the region growing, we can now plan path of point trhough this space without dealing with shape of robot. It is important to point out that yu must make multipe configurations spaces corresponding to various degrees of rotations for moving objects. Then, generalize search to move from spacce to space.

Meadow Maps

We are gonna illustrate an example of the basicc procedure of transforming world space to cspace. Step 1(optional): grow obstacles as big as robot. Step 2: construct convex polygons as line segments between pairs of corners, edges. Step 3: represent convex polygons in a way suitable for path planning-convert to a relational graph.
This approach has some problems: we do not have a unique generation of polygons. The disadvantage of this approach is that the resulting path is jugged. To solve this we can use **path relaxation**, which is a technique for smoothing jagged paths resulting from any discretization of space. The approach is to imagine the path as a string, then imagine pulling on both ends of the string to tighten it. This is to remove mpst of kinks in path. Path relaxations can be used with any cspacce representation. The problem of the Meadow maps are the following: the technique to generate polygonss is computationally complex. It uses artifact of the map to determine polygon boundaries, rather than things that can be sense. It is unclear how to update or repair diagrams as robot discovers differences between a priori map and the real world.

Generalized Voronoi Diagrams (GVGs)

It is a popular mechanism for representing Cspace andd generating a graph. It can be constructed as robot enters new environment. The basic approach is the following: generate a Voronoi edge, which is equidistant from all points; Points where Voronoi edge meets are called Voronoi verteces; vertices often have physical correspondence to aspects of environment that can be sensed; if the robot follows Voronoi edge, it won't collide with any modeled obstacle, ttthen you don't need to grow obstacle boundaries.
Imagine a fire startin at the boundaries, creating a line where they intersect, intersections of lines are node. The result is a relational graph. The problems of this approach is that it is sensitive to sensor noise and the path execution requires the robot to be able to sense boundaries.

Regular Grids/Occupancy Grids

Superimposes a 2D cartesian grid on the world space. If there is any object in the area contained by a grid element, that element is marked as occupied. The center of each element in grid becomes a node, leading to highly connected graph. Grid nodes are connected to neighbors. The disadvantages are the Digitization bias: world doesn't alway line up on grids; leads to wasted space. Partial solution to wasted space are Quadtrees.

Quadtrees

The representation starts with large area. If object falls into part of the grid, nut not all, the space is subdivided into smaller grids. If the object doesn't fit into sub-element, continue reccursive subdivision. A 3D version of Quadtree are called Octree. Octrees are tree based data structure and they use recursive subdivision of the space into octand. The volume is allocated as needed. It gives full 3D model, it is probabilistic, inherently multi-resolution and memory efficient. Unfortunately the implementation can be really tricky.

Techniques for 3D mapping

We have different ways of representation:

- **Point clouds:** they need no discretization of data and the mapped area is not limited. But, you have unboundend memory usage and no direct representation of free or unknown space.
- **3D Voxel Grids:** it gives volumetric representation ,ffers constant access time and uses probabilistic update. But the complete map is allocated in memory, the extent of the map has to be known or guessed and there are discretization errors.
- **2.5D MAPS: Height Maps:** the average over all scan points that fall into a cell. It is memory efficient and has constant time acces. But it's non-probabilistic and there is no distintion between free and unknown space.
- **Elevation Maps:** 2D grid that stores an estimated height for each cell. Typically, the uncertainty increases with measured distance. We use Kalman update to estimate the elevation. You have a 2.5D representation ,constant time access andd probabilistic estimate abouth the height. But, no vertical objeccts and only one level is represented.
- **Extended Elevation Maps:** they identify cells that correspond to vertical structures and cells that contain gaps. They check whether the variance of the height of all data points is large ffor a cell. If so, check whether the corresponding point set contains a gap exceeding the height of the robot("gap cell"). It allows planning with underpasses possible but no paths passing under and crossing over bridges possible.
- **Multi-level Surace Map (MLS):** each 2D cell stores various patches consisting of the height mean μ, the height variance σ and the depth value d. A patch can have no depth and a cell can have one or many patches.

From Point Clouds to MLS maps: determine the cell for each 3D point. Compute vertical intervals. Classify into vertical(>10cm) and horizontal intervals. Apply Kalmna update to estimate the height based on all data points for the horizontal intervals. Take the mean and variance of the highest measurement for the verticcal intervals.

Localization and Mapping from Murphy

Navigation has been extensivvely explored in animals, and scientists see navigation as answering four quesstions necessary for survival:

- **Where am i going?** Addresed by mission planning in AI, where the agent sets goals through deliberation.
- **What is the best way there?**This is the problem of path finding, and is the area of navvviggation which has received the most attention. Path planning methods fall into two broad categories: qualitative andd quantitative.
- **Where have i been?** This is the purview of map making. As a robot expplores new envvironments, it may be part of uts mission to map the environment. Mapping the environment requires knowing where the robot is, or localization. The cathegory of algorithms are now known as Simultaneus Localization And Mapping (SLAM).
- **Where am I?** In order to ffollow a path or build a map, the robot has to know where it is; this is referred to as localization.

Spatial Memory

A robot's world representation and how it is maintained over time is its **spatial memory.** It's the heart of the Cartographer module, which should provide methods and data structures for processing and storin output froom current sensory inputs. Spatial memory shoud also be organized to support methods which can extracct the relevant expectations about navigational task. Spatial memory supports four basic functions: **Attention**(What features or landmarks should i look for next?), **Reasoning**(Can that surfacce support my weight?), **Path Planning**(WWhat is the best way through this space?) and **Information collection**(What does this place look like? Have I ever seen it before? What has changed since the last time i was here?)

Types of Path Planning

Spatial memory takees two formes: route, or qualitative, and layout, or metric or quantitative, representations. In general, path planning algorithms abstract the rbot as a holonomic vehicle and abstract the wrold as a graph. Topological path planning applies classic computer science algorithms for optimally searching for routes through a graph. Metric planning applies more specialized path planning algorithms that can efficiently handle planning through large empty spaces where each block of empty space may be a node on a graph.

Topological Navigation: Route, topological, or qualitative representations express space in terms of the connections between landmarks. The route navigation is perspective dependent. Landmarks that are easy for a human to see may not be visible to a small robot operating close to the floow. Route representations also tend to supplu orientatiion cues, which are egocentric, in that they assume the agent is following the directions at each step. Route representations also assume that there is physiccal symbol grounding. Naviggation using routes is commonly referred to as **topological navigation** because it is using topology to relate landmarks spatially.

Metric Navigation: Layout, metric or quatitative representations ae the opposite of route representations. They are often called metric representations because most maps have some apprroximate scale to estimate distances. The major difference from the route representations are viewpoint andd utility. A layout representation is not dependent of the perspective of the agent; the agent is assumed to be able to translate the ayout int features to be sensed. The layout is orientation and position independent. They can be used to generate route representation but this does not necessarily work the other way. Most maps contain extra information, such as cross strees. An agent can use this information to generate alternative routes if the desired route is blocked. Navigation using metric layouts of the world is sometimes referred to as quantitative navigation, but most often as metric navigation.

Landmarks andd Gateways

Topological navigation depensd on the presence of **landmarks**, which is one or more perceptually distinctive feature of interest on an object or locale. A landmark can be a grouping of objects as well. If a robot finds a landmark in the world and that landmark appears on a map, then the robot is localized with respect to the map. If the robot plans a path consisting of segments, landmarks are needed so the robot can tell when it has completed a segment and another should begin. If a obot finds new landmarks, they ccan be added to its spatial memory, creating or extending a map. A **gateway** is a special case of landmark. It is an opportunity for a robot to change its overall direction of navigation. Because gateways are navigational opportunities, recognizing gateways is critical for localization, path planning and map making.

Landmarks can still be classified in **Artificial**, which is a set of features added to an existing object or locale in order to support either recognition of the landmark or some other perceptual activity, or **Natural**, which is a configuration of existing features selected for recognition which were not expressly designed for the perceptual activity. Regardless of whether a landmark is artificial or natural, it must satisfy 3 criteria: **Be readily recognizable:** if the robot cannot find the landmark it is not useful; **Support the task dependent activity:** if the task dependent activity is simply an orietation cue, then being recognized is enough; **Be perceivable from many different viewpoints:** if the landmark is widely visible, the robot may never find it. A good landmark has many other desirable characteristics. It shuld be passive in order to be available despite a power failure. It should be perceivable over the entire range where the robot might need to see it. it should have distinctive feature and if possible unique features.

Relational Methods

Relational methods represent the world as a graph or network of nodes and edges. Nodes represent gateways, landmarks or goals. Edges represent a navigable path between two nodes, in effect, representing that two nodes have a spatial relationship. Paths can be computed between two points using standard graph algorithms, such as Dijkstra's single source shortest path algorithm.

Distinctive Places

A **distinctive place** is a landmark that the robot could detect from a nearby region called a neighborhood. Each node represents a distinctive place. Once in the neighborhood, the robot can position itself, using sensor readings, in a knwon spot relative to the landmark. The ide of a neighborhood was that the robot would move around in the neighborhood until it achieved a specified relative location to the landmark. Then the robot would be localized on the map.
Hill-Climb Algorithm: an arc or edge in the relational graph ewas called a local control strategy or lcs. The local control strategy is the procedure for getting from the current node to the next one. When the robot senses the landmark, it fills in values for a set of features. Tje robot uses a hill-climbing algorithm, which direct the robot around in the neighborhood until a measurement function indicates when the robot is at a position where the feature values are maximized. The point where the feature values are maximized is the distinctive plave. The robot always moves in the direcction whicch appears to cause the most positive change in the measurement function.
The distinctive concept eliminates any navigational errors at each node. The robot may drift off course because the hall is wide, but as soon as it reaches the neighborhood, it self-corrects and localizes itself. It should be noticed that a landmark must be unique to a node pair. There cannot be any corners in the real world that are not on the graph between the nodes, or else the robbot will localize itself incorrectly.

Associative methods

Associative methods for topological navvigation essentially create a behavior which converts sensor observation into the direction to go to reach a particular landmark. There are basic approaches: **visual homin**, where the robot recognizes visual patterns associated with locations and routes, and **QualNav**, where the robot infers its location based on how the relative position of landmarks change. The underlying assumption is that a location or landmark of interest for navigation usually has two attributes:**Perceptual stability:** views of the location that are close

together should look similar; **Perceptual Distinguishability:** view far away should look different. These principles are implicit in the idea of a neigghborhood around a distinctive place. If the robot is in the neighborhood, the view of the landmark look similar.

Motion Planning

Motion Planning from notes

Motion Planning is the ability for an agent to compute its own motion in order to achieve certain goals. All autonomous robots and digital actors should eventually have this ability. we use it to compute motion strategies like geometric paths, time-parameterized trajecctories or sequence of sensor-based motion commands. We use it to achieve high-level goals like go to A withouth colliding with obstacles, assemble product P, build map of environment E or to find an object O. We can define the Motion Planning problem as: **Finding a path from START to GOAL without collisions.** The formal definition is the following: Compute a collision-free path for a rrigid or articulated object among static obstaccles. The inputs are the geometry of moving object and obstacles, kinematics of moving objects and initial and goal configurations. The output is a continuous sequence of collision-free object configurations connecting the initial and goal configurations. More formally, we can define the Motion planning basic problme as:
Given a workspace W, where either $W = \mathbb{R}^2$ or $W = \mathbb{R}^3$, an obstacle region $O \in W$, a robot defined in W. Either a rigid body R or a collection of m links $A_1, ..., A_m$. The configuration space C (C_{obs} and C_{free} are then defined). An initial configuration $q_I \in C_{free}$, and a goal configuration $q_G \in C_{free}$. The initial and goal configurations are often called a query (q_I, q_G). We want to compute a continuous path $\tau : [0, 1] \to C_{free}$, **such that** $\tau(0) = q_I$ **and** $\tau(1) = q_G$.
Path planning refers to the purely geometric problem of computin a collision-free path for a robot among static obstacles. Motion planning is used for problems involving time, dynamic constraint, object coordination, sensory interaction, etc. A MP Solver has a Planner as a component inside its algorithmic structure.

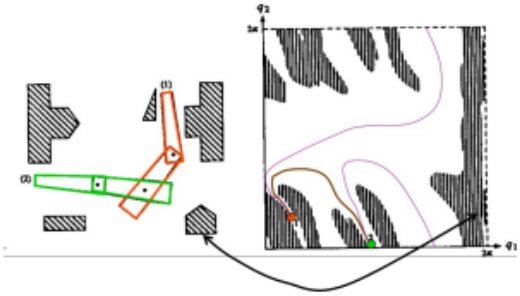
Translating Planar Rigid Bodies

the robot's configuration can be specified by a reference point (x,y) on the planar rigid body relative to some fixed coordinate frame. C-space is $\mathbb{R}^2$. The obstacle region is traced by sliding the robt around the workspace obstacle to find the constraint on all $q \in C$.
how can we define a better path? We need to go from a continuous space (coonfiguration space), to discretization to search.

Planar Arms(Example of configuration space)

The bases of both links are pinned and they have no joint limit, THEN the C-space: each joint angle θ_i ccorresponds to a point on a unit circle $\mathbb{S}^1$ annd the C-space is $\mathbb{S}^1 \times \mathbb{S}^1 = T^2$ (2 dimensional torus).
In the case of an higher number of links without joint limits then $C = \mathbb{S}^1 \times \mathbb{S}^1 \times ... \times \mathbb{S}^1$.
If a joint has limit, $\mathbb{S}^1$ is replaced with $\mathbb{R}$ (even though it is a finite interval). If the base of

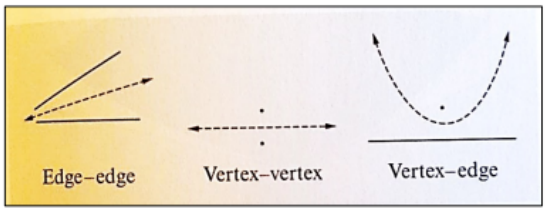
the planar arm is mobile and not pinned, then the additional translation parameters must also be considered in the arm's configuration: $C = \mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{S}^1 \times ... \times \mathbb{S}^1$. Ann so until reaching a configuration space of this type:



Discretization

Sampling-based planners

The key idea is to exploit advances in collision detection algorithms that compute whether a single configuration is collision free. Given this simple primitive, a planner samples different configurations to construct a data structure that stores 1-D C-space curves, which represent collision-free paths. Sampling-based planners do not access the C-space obstacles directly but only through the collision detector and the constructed data structure. Using thiss level of abstraction, the planners are applicable to a wide range of problems by tailoring the collision detector to specific robots and applications. Sampling-based planners provide a weaker, but still interestng, form of completeness: **If a solution path exists, the planner will eventually find it.**
Maximum Clearance Roadmap: method that constructs a roadmap that keeps paths as far from the obstacles as possible. Paths are cconstributed to the roadmap fro the following 3 cases, which correspond to all ways to pair together polygon features: The roadmap can be made naively



in time $O(n^4)$(best algorithm is O(nlogn)) by generating all aforementioned curves for all possible pairs, computing their intersections, and tracing out the roadmap.
Shortest-Path roadmap: paths may actually touch obstacles, which must be allowed for paths to be optimal. The roaddmap vertices are the reflex vertices of C_{obs}, which are vertices for which the interior angle is greater then π. An edge exists in the roadmap if and only if a pair of vertices is mutually visible and the line through them pokes into C_{free} when extended outward from each vertex (such lines are called **bitangents**). An $O(n^2 \log n)$ time construction algorithm can be formed by using a radial sweep algorithm from each reflex vertex. It can theoretically be computed in time $O(n^2 + m)$, in which m is the total number of edges in the roadmap.
Vertical Cell decomposition: the idea is to decompose C_{free} into cells that are trapezoids or triangles. Planning each cell is trivial because each cell is convex. A roadmap is made by placing a point in the center of each cell and on each boundary between cells. Any graph search alggorithm

can be used to find a ccollision-free path quickly. The cell decomposition can be constructed in $= (n \log n)$ time using the plane sweep principle.

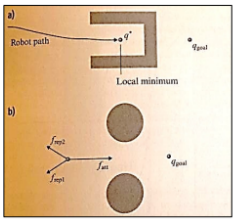
Potential Fields

Inspired from obstacle avoidance techniques. It does not explicitly cconstruct a roadmap, but instead constructs a differentiable real-vvalued function $U : \mathbb{R}^n \to \mathbb{R}$, called a potential function, that guides the motion of the moving objects. A potential is composed of two components: $U_a(q)$, which is an actractive component which pulls the robot towards the goal, and $U_r(q)$, which is a repulsive component which pushes the robot aeay from the obstaccles. The gradient of the potential function is the following:

$$\nabla U(q) = DU(q)^T = \left[\frac{\partial U}{\partial q_1}(q)_1, ..., \frac{\partial U}{\partial q_m}(q)\right]$$

Gradient Descent: after the definition of U, a path can be computed by starting from q_I and applying gradient descent: The gradient descent approach does not guarantee a solution to the

1. q(0) = q_i; i=0;
2. While $\nabla U(q(i)) \neq 0$ do
3. q(i+1) = q(i) + $\nabla U(q(i))$
4. i = i+1



problem. It can only reach a local minimum of U(q), which may not correspond to the goal state q_G.
Randomized Potential Planner: the idea is to combine potential functions with random walks by exploiting multiple planning modes. The modes are calculated by:

I. Gradient descent is applied until a local minimum is reached;

II. Random walks are used to try to escape local minima;

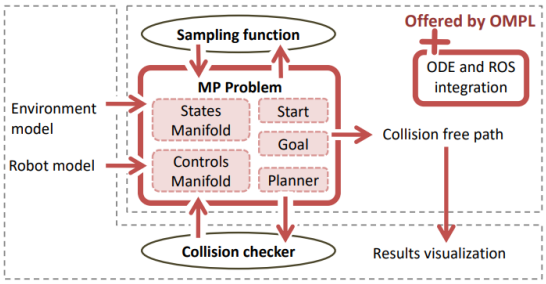
III. Backtracking is performed whenever several attempts to escape a local minimum have failed.

The approach is similar to a sampling-based planner but it guarantees completenes. The pro is that it has the benefits of potential functions but it avoids local minima.

Motion Planning Libraries

OMPL

Stands for Open Motion Planning Library. It is a motion planning library that implements Probabilistic sampling-based motion planning algorithms. A probabilistic planner has been developed since a complete planner is too slow and an heuristic planner is too unreliable. In 1991, a Randomized Planner (RRT) was introducced, which was able to solve complex path planning problems ffor many-dof robots by alternating down motions to track the negated gradient of a potential field and random motions to escape local minima. That means that a **Probabilistic Planner brings Probabilistic Completeness.** Probabilistic completeness means that if there is a solution path, the probability that the planner will find it is a function that goes to 1 as running time increases.



For **the states** we have rotation in SO2 and SO3, translations ($\subseteq \mathbb{R}^n$), Roto-translations in SE2 and SE3, but is also customizable. For **the controls**(only for the kinodinamic planning) wehave a subset of $\mathbb{R}^n$, e.g.: the control is the force F applied to the robot: $F = (F_x, F_y, F_z)$. The **sampling function** is a uniform distribution and obstacle based. The idea is that a state near an obstacle increases the probability to overcome narrow passages. The sampling function has two strategies: the simple strategy:

I. extract $q_1 \in Q_{free}$ according to a uniform distribution;

II. extract $q_2 \in Q_{obs}$ according to a uniform distribution;

III. Interpolate the path between q_1 and q_2;

IV. the selected state q_{rand} is the last valid state from the interpolation;

Gaussian strategy:

I. extract $q_1 \in Q_{free}$ according to a uniform distribution;

II. extract $q_2 \in Q_{obs}$ according to a Gaussian distribution: $\mu = q_1$ and $\sigma = a$;

III. if $(q_1, q_2 \in Q_{free})$ or $(q_1, q_2 \in Q_{obs})$ repeat the extraction;

IV. else if $q_1 \in Q_{free}$ then $q_{rand} = q_1$;

V. else if $q_2 \in Q_{free}$ then $q_{rand} = q_2$;

The **collision checker** is used to evaluate the states validity andd the validity of a path between 2 states. Implementation examples are collision in terms of geometric models, collision between the model of the robot and the objects of the environment perceived by the sensos, more and more, and **iCollide.**
iCollide is an interactive and exact collision detection library for large environments composed of convex polyhedra. It consider the features of all object of the space. It builds the bounding boxes around the objects; Considers all couples of bounding boxes and test their overlap with Sweep and Prune. Is absent the bodies do not collid and the output is the smallest distance that separates the 2 bodies; if it is present then the bodies can collied and we use the **Lin Canny test.** More accurately, the pipeline is the following: It check the overlaps between the bounding boxes. The functioning is: projects every box on the axes, it make the intervals, it maintain a list of intervals for each axis, then order the list according to the insertion sort algorithm, then: a couple of boxes overlaps if and only if their intervals overlap in all dimensions.
Planner: plan a robot's trajectory that satisfies a dynamic equation of motion of the form $s' = f(s, u)$, where s is the rrobot sttatte/configuration at time t and u is the control input at time t. The planner has to avoid collisions with moving obstacles and produces in output a piecewise constant control function u(t).

Basic Probabilistic Roadmap

The idea is building a roadmap that connects Start and goal. We have two phases:

I. **Learning:** the algorithm expands the roadmap: pick-up n configurations at random. Kepp thos that are in C_{free}(collision checking), Call V the set of all free configurations. For each pair of points of V: try to connect them by using a simple fast deterministic motion planner; each succesful connection becomes a part of the roadmap.

II. **Search:** the algorithm determines a solution through the roadmap: connect start ad goal configurations to the roadmap, search for a path between connection points of the roadmap.

When we expand the node we have a parameter ϵ which is the maximum extension of the new edge added by the local planner.
RRTConnect: two trees are built: one from Start and one from Goal; Trees are alternatively expanded; after the expansions of a tree, we expand the other tree with the aim of creating a single connected component.
BKPIECE: like RRTConnect but the trees are expanded using the KPIECE technique.

Motion Planning from Murphy

4 situations where topological navigation is noot sufficient

One situation occurs when the space between the starting point and the destination is not easily abstracted into labeled or perceptually distinct regions. A second situation arises whhen the choice of route impacts the control or energy costs of the vehicle. The third case can occure when the purpose of the path is to allow sensor coverage of an area. A fourt situation can arise when the pose of the robot is important, either while reaching a destination or during coverage of an area.
Waypoints: these four situations lead to a different fflavor of path planning, called metric path planning and motion planning. In metric methods, the robot has an a priori map of the environment and generates a plan of appropriate movements to reach the goal or degree of coverage. Metric pathss usually decompose the path into subgoals consisting of **waypoints**, which are most often fixed locations o coorrdinates or locations and poses.

Configuration space

The physical spacce robots and obstcacles exist in can be thought of as the world space. The configuration space, Cspace for short, is a data structure which allows the robot to specify the position of itself and any other objectss and the robot. A good Cspace representation reduces the number of dimensions that a planner has to contend with. For mobile robots the world is described in two dimensions and the robot can only move forward or backward and turn within the (x,y) plane. We will now see some Cspace representations.

Meadow Maps

Many early path planning algorithms developed for mobile robots assumed that the robot have a highly accurate map of the world in advance. This map could be digitized in some manner, and the the robot could apply various algorithms to convert the map to a suitable Cspace representation.
Meadow Maps transform ffree space into convex polygons, which have an important property:

if the robot starts on the perimeter and goes in a straight line to any other point on the perimeter, it will not go out of the polygon. The polygon represents a safe region for the robot to traverse. The path planning problem becomes a matter of picking the best series of polygons to transit through. The programming steps are straightforward: first the planner begins witha metric layout of the world space. The enxt step is to construct convex polygons by considering the line segments between pairs of interesting features. The meadow map is technically complete now, but it is not in a format that supports path planning. Some of the line segments foming the perimeter are not connected to another polygon, so theu should be off limits to the planning algorithm. Meadow maps have three problems which limit their usefulness: the technique to generate the polygons is computationally complex; it uses artifact of the map to determine the polygon boundaries rather than things which can be sensed; it is unclear how to update or repair the diagrams as the robot discovers discrepancies between the a priori map and the real world.

Generalized Voronoi Graphs

A GVG is a popular mechanism for representing Cspace and fgenerating a graph. Unlike meadow maps, they can be constructed as the robot enters a new environment, thereby creating a topological map. The basic idea of a GVG is to generate a line, called a **Voronoi edge**, equidistant from all points. The point where many Voronoi edges meet is known as a **Voronoi vertex**. Notice the vertices often have a physical correspondence to configurations that can be sensed in the environment. This makes it much easire for a robot to follow a path generated from a GVG, since there is an implicit local control strategy for staying equidistant from all obstacles. If the robot follows the Voronoi edge, it will not collide with any modeled obstacles because it is staying in the middle.

Regular Grids

Another method of partitioning the world space is a regularr grid, which superimposes a 2D cartesian grid on the world space. If there is any object in the area contained by a grid element, that element is marked occupied. Therefore, regular grids are often referre to as occupancy grids. Regular grids are straightforward to apply. The center of each element of the grid can become a node, transforming the grid into a highly connected graph. Grids are either considere 4-connected or 8-connected, depending on whether they permit an arc to be drawn diagonally between nodes. Unfortunately they introduce digitization bias, which means that if an object falls into even the smallest portion of a grid element, the whole element is marked occupied.

Metric Path Planning

Metric methods for destination or route planning rely on a priori map. Graph methods transform the a priori map into a graph-like Cspace, then use a graph search algorithm to find the shortest path in the graph between the start and goal nodes.

A* and Graph-Based Planners

We want to compute the path between the initial node and the goal node using search algorithm. The A* algorithm is the classic method for computing optimal paths for holonomic rrobots. It is derived from the A search algorithm. The evaluation function of the A* is the following: $f*(n) = g*(n) + h*(n)$, where the * means tthat the functions are estimates of the values that would hae been plugged into the A search evaluation. f(n) measures how good the move to node n is, g(n) measures the cost of getting node n from the initial node, h(n) represents the lowest cost

of getting from n to goal. In path planning, g(n) is the same as g*(n). h(n) is the real difference, we want h*(n) always to be samlller than h(n) (admissibility condition).

Wavefront-based Planners

They consider the Cspace to be a conductive material with heat radiating from the initial node to the goal noe. Eventually the heat will spread and reach the goal g there is a way. The optimal path from all grid elements to the goal can be computed as a side effect, and any cost of navigating throughh a terrain can be incorporated as a different conductivity. Wavefront propagation results in a map which looks like a potential field.

Executing a Planned Path

Subgoal Obsession

It occurs when the robot spends too much time and energy trying to reach the exact subgoal position, or more precisely, when the termination condistions are set with an unrealistic tolerance.

Replanning

There are two approachef to replanning (changging subgoal according to the necessity): one is to continuously replan, essentially imposing hierarchiclal Sense, Plan, Act cycle; the other is to replan where there is some event, exception, or indication that the plan execution is not working. An example of continous replanning is the **D+ search algorithm.** D* executes first A* search from each possible location to the goal in advance; this reformulation transforms A* from a single-source path algorithm into an all-paths algorithm. D* represents one extreme on the replanning scale: **continuous replanning.**
An event-driven replanning occurs when an event noticeable by the receiver system would trigger replanning.

Motion Planning

Motion planning stems from industria manipulator branch of robotics where the dynamics of robot cannot be abstracted away by assumptions of holonomicity and the shape of the robot. The class of applications where pose cannot be abstracted away is often referred to as the **piano mover's problem**, because moving a robot through an extremely cluttered 3D environment is analogous to a group of movers planning how to rotate and flip the piano at each step in order to get a piano through a dorr. The piano mover's roblem becomes tractable if the goal is simply to generate a collision-free plan, but not necessarily an optimal one.
Some things about the RRT algorithm(look at class notes.)

Criteria for Evaluating Path and Motion Planners

The minimum criteria for evaluating the suitability of a path or motion planner is:

I. **Complexity**

II. **Sufficiently represents the terrain**

III. **Sufficiently represents the physical limitations of the robot platform**

IV. **Compatible with the reactive layer**

V. **Supports corrections to the map and replanning**