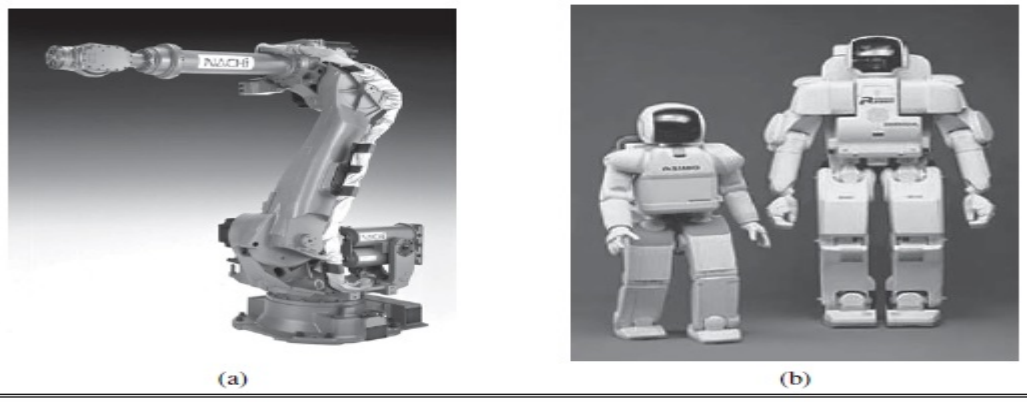
UNIT-V

Robotics: Introduction, Robot Hardware, Robotic Perception, planning to move, planning uncertain movements, Moving, Robotic software architectures, application domains

Philosophical foundations: Weak AI, Strong AI, Ethics and Risks of AI, Agent Components, Agent Architectures, are we going in the right direction, what if AI does succeed.

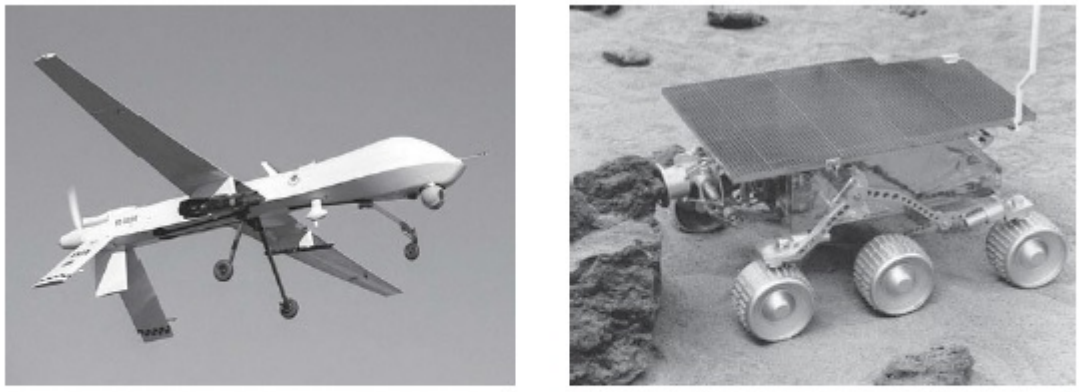
Introduction

- **Robots** are physical agents that perform tasks by manipulating the physical world. To do so they are equipped with **effectors** such as legs, wheels, joints, and grippers. Effectors have a single purpose: to assert physical forces on the environment. Robots are also equipped with **sensors**, which allow them to perceive their environment.
- Present day robotics employs a diverse set of sensors, including cameras and lasers to measure the environment, and gyroscopes and accelerometers to measure the robot's own motion.
- Most of today's robots fall into one of three primary categories. **Manipulators**, or robot arms, are physically anchored to their workplace, for example in a factory assembly line or on the International Space Station. Manipulator motion usually involves a chain of controllable joints, enabling such robots to place their effectors in any position within the workplace. Manipulators are by far the most common type of industrial robots, with approximately one million units installed worldwide. Some mobile manipulators are used in hospitals to assist surgeons.
- The second category is the **mobile robot**. Mobile robots move about their environment using wheels, legs, or similar mechanisms.
- They have been put to use delivering food in hospitals, moving containers at loading docks, and similar tasks. **Unmanned ground vehicles**, or UGVs, drive autonomously on streets, highways, and off-road.
- The **planetary rover** shown in Figure 25.2(b) explored Mars for a period of 3 months in 1997.
- Other types of mobile robots include **unmanned air vehicles** (UAVs), commonly used for surveillance, crop-spraying,



(a) (b)

Figure 25.1 (a) An industrial robotic manipulator for stacking bags on a pallet. Image courtesy of Nachi Robotic Systems. (b) Honda's P3 and Asimo humanoid robots.



(a) (b)

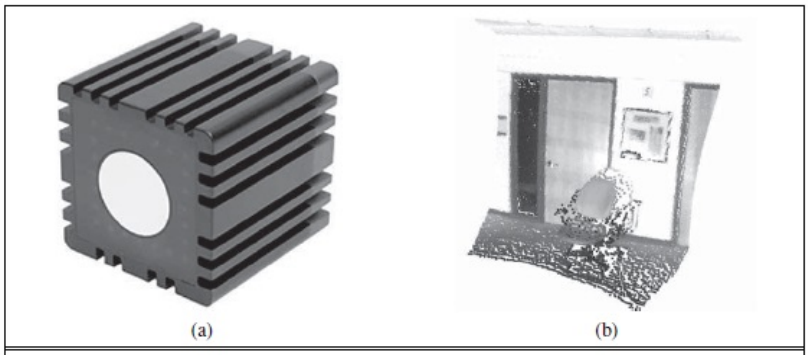
(a) Predator, an unmanned aerial vehicle (UAV) used by the U.S. Military. Image courtesy of General Atomics Aeronautical Systems. (b) NASA's Sojourner, a mobile robot that explored the surface of Mars in July 1997.

- UAV commonly used by the U.S. military. **Autonomous underwater vehicles** (AUVs) are used in deep sea exploration. Mobile robots deliver packages in the workplace and vacuum the floors at home.
- The third type of robot combines mobility with manipulation, and is often called a **mobile manipulator**. **Humanoid robots** mimic the human torso. shows two early humanoid robots, both manufactured by Honda Corp. in Japan. Mobile manipulators can apply their effectors further afield than anchored manipulators can, but their task is made harder because they don't have the rigidity that the anchor provides.

ROBOT HARDWARE

- **Sensors:** Sensors are the perceptual interface between robot and environment

- **Passive sensors**, such as cameras, are true observers of the environment: they capture signals that are generated by other sources in the environment.
- **Active sensors**, such as sonar, send energy into the environment. They rely on the fact that this energy is reflected back to the sensor. Active sensors tend to provide more information than passive sensors, but at the expense of increased power consumption and with a danger of interference when multiple active sensors are used at the same time. Whether active or passive, sensors can be divided into three types, depending on whether they sense the environment, the robot's location, or the robot's internal configuration.
- **Range finders** are sensors that measure the distance to nearby objects. In the early days of robotics, robots were commonly equipped with **sonar sensors**. Sonar sensors emit directional sound waves, which are reflected by objects, with some of the sound making it back into the sensor. The time and intensity of the returning signal indicates the distance to nearby objects. Sonar is the technology of choice for autonomous underwater vehicles



(a) Time of flight camera; image courtesy of Mesa Imaging GmbH. (b) 3D range image obtained with this camera. The range image makes it possible to detect obstacles and objects in a robot's vicinity.

- **Stereo vision** relies STEREO VISION on multiple cameras to image the environment from slightly different viewpoints, analysing the resulting parallax in these images to compute the range of surrounding objects. For mobile ground robots, sonar and stereo vision are now rarely used, because they are not reliably accurate.
- Most ground robots are now equipped with optical range finders. Just like sonar sensors, optical range sensors emit active signals (light) and measure the time until a reflection of this signal arrives back at the sensor. shows a **time of flight camera**. This camera acquires range images like the one at up to 60 frames per second.

- Other range sensors use laser beams and special 1-pixel cameras that can be directed using complex arrangements of mirrors or rotating elements. These sensors are called **scanning lidars** (short for *light detection and ranging*). Scanning lidars tend to provide longer ranges than time of flight cameras, and tend to perform better in bright daylight.
- Other common range sensors include radar, which is often the sensor of choice for UAVs. Radar sensors can measure distances of multiple kilometres. On the other extreme end of range sensing are **tactile sensors** such as whiskers, bump panels, and touch-sensitive skin. These sensors measure range based on physical contact, and can be deployed only for sensing objects very close to the robot.
- A second important class of sensors is **location sensors**. Most location sensors use range sensing as a primary component to determine location. Outdoors, the **Global Positioning System** (GPS) is the most common solution to the localization problem. GPS measures the distance to satellites that emit pulsed signals. At present, there are 31 satellites in orbit, transmitting signals on multiple frequencies. GPS receivers can recover the distance to these satellites by analysing phase shifts. By triangulating signals from multiple satellites, GPS receivers can determine their absolute location on Earth to within a few meters.
- **Differential GPS** involves a second ground receiver with known location, providing millimetre accuracy under ideal conditions. Unfortunately, GPS does not work indoors or underwater. Indoors, localization is often achieved by attaching beacons in the environment at known locations. Many indoor environments are full of wireless base stations, which can help robots localize through the analysis of the wireless signal.
- The third important class is **proprioceptive sensors**, which inform the robot of its own motion. To measure the exact configuration of a robotic joint, motors are often equipped with **shaft decoders** that count the revolution of motors in small increments.
- On robot arms, shaft decoders can provide accurate information over any period of time. On mobile robots, shaft decoders that report wheel revolutions can be used for **odometry**—the measurement of distance travelled.
- **Inertial sensors**, such as gyroscopes, rely on the resistance of mass to the change of velocity. They can help reduce uncertainty.
- Other important aspects of robot state are measured by **force sensors** and **torque sensors**. These are indispensable when robots handle fragile objects or objects whose exact shape and location is unknown. Imagine a one-ton robotic manipulator screwing in a light bulb. It would be all too easy to

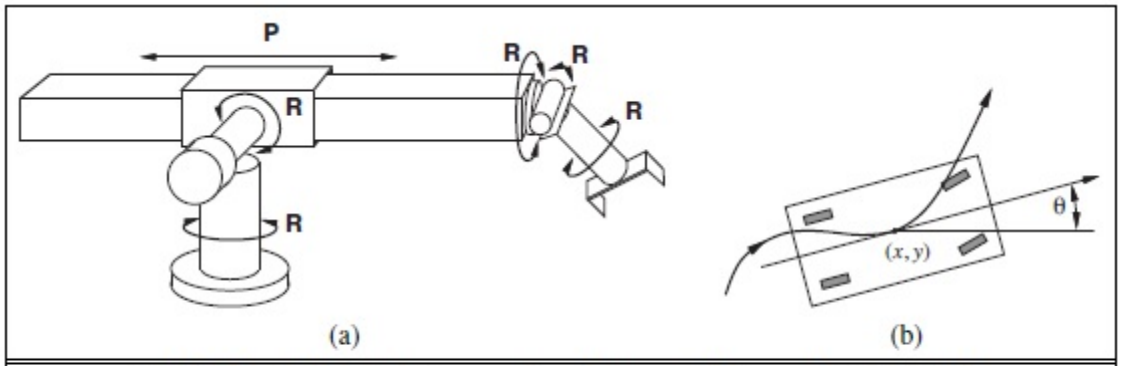
apply too much force and break the bulb. Force sensors allow the robot to sense how hard it is gripping the bulb, and torque sensors allow it to sense how hard it is turning. Good sensors can measure forces in all three translational and three rotational directions. They do this at a frequency of several hundred times a second, so that a robot can quickly detect unexpected forces and correct its actions before it breaks a light bulb.

25.2.2 EFFECTORS

Effectors are the means by which robots move and change the shape of their bodies. To understand the design of effectors, it will help to talk about motion and shape in the abstract, using the concept of a **degree of freedom** (DOF) We count one degree of freedom for each independent direction in which a robot, or one of its effectors, can move. For example, a rigid mobile robot such as an AUV has six degrees of freedom, three for its (x, y, z) location in space and three for its angular orientation, known as *yaw*, *roll*, and *pitch*. These six degrees define the **kinematic state**2 or **pose** of the robot. The **dynamic state** of a robot includes these six plus an additional six dimensions for the rate of change of each kinematic dimension, that is, their velocities.

For nonrigid bodies, there are additional degrees of freedom within the robot itself. For example, the elbow of a human arm possesses two degree of freedom. It can flex the upper arm towards or away, and can rotate right or left. The wrist has three degrees of freedom. It can move up and down, side to side, and can also rotate.

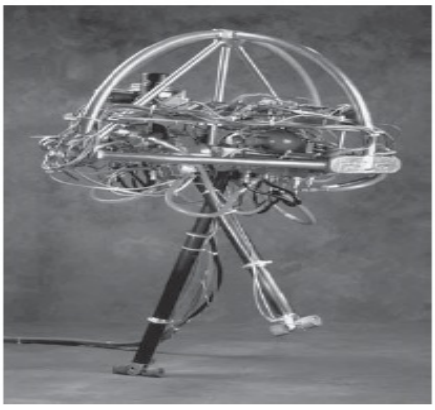
created by five **revolute joints** that generate rotational motion and one **prismatic joint** that generates sliding motion. You can verify that the human arm as a whole has more than six degrees of freedom by a simple experiment: put your hand on the table and notice that you still have the freedom to rotate your elbow without changing the configuration of your hand. Manipulators that have extra degrees of freedom are easier to control than robots with only the minimum number of DOFs. Many industrial manipulators therefore have seven DOFs, not six.



(a) The Stanford Manipulator, an early robot arm with five revolute joints (R) and one prismatic joint (P), for a total of six degrees of freedom. (b) Motion of a nonholonomic four-wheeled vehicle with front-wheel steering.

For mobile robots, the DOFs are not necessarily the same as the number of actuated elements. Consider, for example, your average car: it can move forward or backward, and it can turn, giving it two DOFs. In contrast, a car's kinematic configuration is three-dimensional:
on an open flat surface, one can easily maneuver a car to any (x, y) point, in any orientation. Thus, the car has three **effective degrees of freedom** but two **control label degrees of freedom**. We say a robot is **nonholonomic** if it has more effective DOFs than controllable DOFs and **holonomic** if the two numbers are the same. Holonomic robots are easier to control—it would be much easier to park a car that could move sideways as well as forward and backward—but holonomic robots are also mechanically more complex. Most robot arms are holonomic, and most mobile robots are nonholonomic. Mobile robots have a range of mechanisms for locomotion, including wheels, tracks, and legs. **Differential drive** robots possess two independently actuated wheels (or tracks), one on each side, as on a military tank. If both wheels move at the same velocity, the robot moves on a straight line. If they move in opposite directions, the robot turns on the spot. An alternative is the **synchro drive**, in which each wheel can move and turn around its own axis. To avoid chaos, the wheels are tightly coordinated. When moving straight, for example, all wheels point in the same direction and move at the same speed. Both differential and synchro drives are nonholonomic. Some more expensive robots use holonomic drives, which have three or more wheels that can be oriented and moved independently external forces.

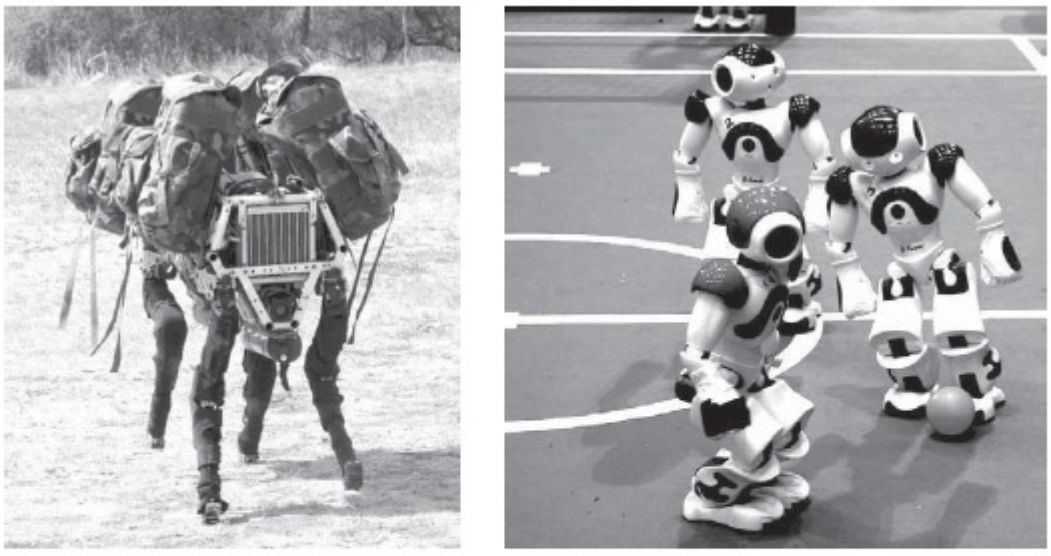
(a)



(b)

(a) Mobile manipulator plugging its charge cable into a wall outlet. Image courtesy of Willow Garage, © 2009. (b) One of Marc Raibert's legged robots in motion.

Legs, unlike wheels, can handle rough terrain. However, legs are notoriously slow on flat surfaces, and they are mechanically difficult to build. Robotics researchers have tried designs ranging from one leg up to opens of legs. Legged robots have been made to walk, run, and even hop—as we see with the legged robot. This robot is **dynamically stable**, meaning that it can remain upright while hopping around. A robot that can remain upright without moving its legs is called **statically stable**. A robot is statically stable if its center of gravity is above the polygon spanned by its legs. The quadruped (four-legged) robot may appear statically stable. However, it walks by lifting multiple legs at the same time, which renders it dynamically stable. The robot can walk on snow and ice, and it will not fall over even if you kick it (as demonstrated in videos available online). Two-legged robots are dynamically stable.



(a) (b)

(a) Four-legged dynamically-stable robot "Big Dog." Image courtesy Boston Dynamics, © 2009. (b) 2009 RoboCup Standard Platform League competition, showing the winning team, B-Human, from the DFKI center at the University of Bremen. Throughout the match, B-Human outscored their opponents 64:1. Their success was built on probabilistic state estimation using particle filters and Kalman filters; on machine-learning models for gait optimization; and on dynamic kicking moves. Image courtesy DFKI, © 2009.

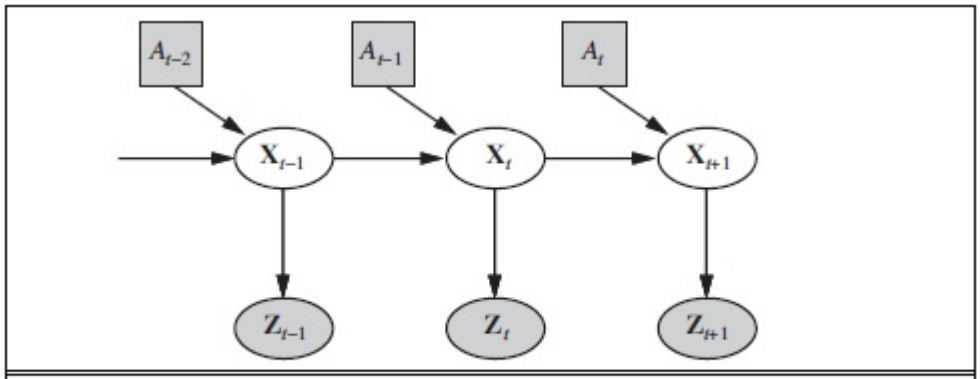
The **electric motor** is the most popular mechanism for both manipulator actuation and locomotion, but **pneumatic actuation** using compressed gas and **hydraulic actuation** using pressurized fluids also have their application niches.

ROBOTIC PERCEPTION

Perception is the process by which robots map sensor measurements into internal representations of the environment. Perception is difficult because sensors are noisy, and the environment is partially observable, unpredictable, and often dynamic. In other words, robots have all the problems of **state estimation** (or **filtering**). As a rule of thumb, good internal representations for robots have three properties: they contain enough information for the robot to make good decisions, they are structured so that they can be updated efficiently, and they are natural in the sense that internal variables correspond to natural state variables in the physical world.

we saw that Kalman filters, HMMs, and dynamic Bayes nets can represent the transition and sensor models of a partially observable environment, and we described both exact and approximate algorithms for updating the **belief state**—the posterior probability distribution over the environment state variables. Several dynamic Bayes net models. For robotics problems, we include the robot's own past actions as observed

variables in the model. the notation used in this chapter: **X**t is the state of the environment (including the robot) at time t, **Z**t is the observation received at time t, and At is the action taken after the observation is received.



Robot perception can be viewed as temporal inference from sequences of actions and measurements, as illustrated by this dynamic Bayes network.

We would like to compute the new belief state, **P**(**X**t+1 | **z**1:t+1, a1:t), from the current belief state **P**(**X**t | **z**1:t, a1:t−1) and the new observation **z**t+1. We did this in Section 15.2, but here there are two differences: we condition explicitly on the actions as well as the observations, and we deal with *continuous* rather than *discrete* variables.

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{z}_{1:t+1}, a_{1:t}) = \alpha \mathbf{P}(\mathbf{z}_{t+1} \mid \mathbf{X}_{t+1}) \int \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, a_t)\, P(\mathbf{x}_t \mid \mathbf{z}_{1:t}, a_{1:t-1})\, d\mathbf{x}_t \,.$$

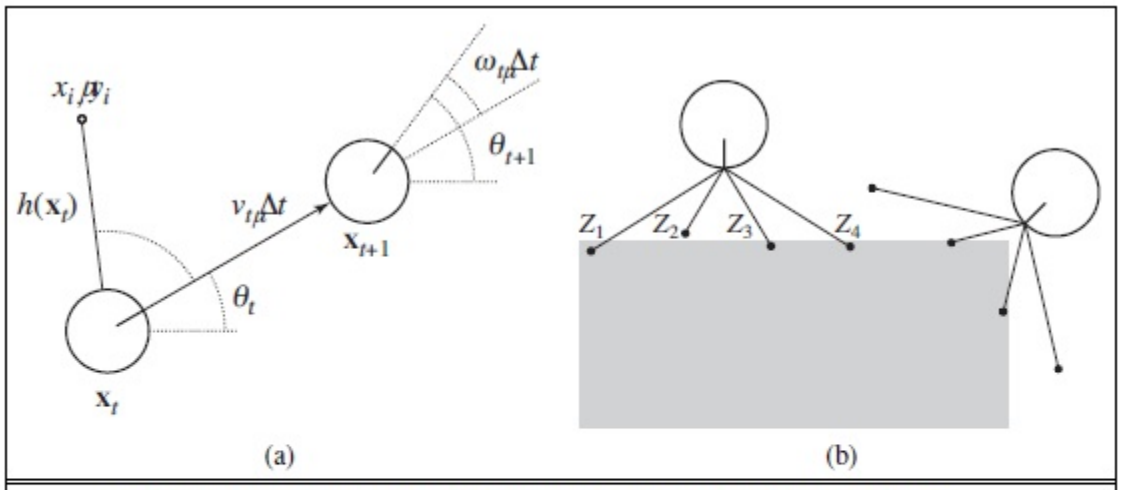
This equation states that the posterior over the state variables **X** at time t + 1 is calculated recursively from the corresponding estimate one-time step earlier. This calculation involves the previous action at and the current sensor measurement **z**t+1. For example, if our goal is to develop a soccer-playing robot, **X**t+1 might be the location of the soccer ball relative to the robot. The posterior **P**(**X**t | **z**1:t, a1:t−1) is a probability distribution over all states that captures what we know from past sensor measurements and controls. how to recursively estimate this location, by incrementally folding in sensor measurements (e.g., camera images) and robot motion commands. The probability **P**(**X**t+1 | **x**t, at) is called the **transition model** or **motion model**, and **P**(**z**t+1 | **X**t+1) is the **sensor model**.

25.3.1 LOCALIZATION AND MAPPING

Localization is the problem of finding out where things are—including the robot itself. Knowledge about where things are is at the core of any successful physical interaction with the environment. For

example, robot manipulators must know the location of objects they seek to manipulate; navigating robots must know where they are to find their way around.

To keep things simple, let us consider a mobile robot that moves slowly in a flat 2D world. Let us also assume the robot is given an exact map of the environment. (An example of such a map appears in Figure 25.10.) The pose of such a mobile robot is defined by its two Cartesian coordinates with values x and y and its heading with value θ, as illustrated. If we arrange those three values in a vector, then any particular state is given by **X**t =(xt, yt, θt)_.



(a) A simplified kinematic model of a mobile robot. The robot is shown as a circle with an interior line marking the forward direction. The state $\mathbf{x}_t$ consists of the (x_t, y_t) position (shown implicitly) and the orientation θ_t. The new state $\mathbf{x}_{t+1}$ is obtained by an update in position of $v_t\Delta_t$ and in orientation of $\omega_t\Delta_t$. Also shown is a landmark at (x_i, y_i) observed at time t. (b) The range-scan sensor model. Two possible robot poses are shown for a given range scan (z_1, z_2, z_3, z_4). It is much more likely that the pose on the left generated the range scan than the pose on the right.

In the kinematic approximation, each action consists of the "instantaneous" specification of two velocities—a translational velocity vt and a rotational velocity ωt. For small time intervals Δt, a crude deterministic model of the motion of such robots is given by

$$\hat{\mathbf{X}}_{t+1} = f(\mathbf{X}_t, \underbrace{v_t, \omega_t}_{a_t}) = \mathbf{X}_t + \begin{pmatrix} v_t\Delta t \cos\theta_t \\ v_t\Delta t \sin\theta_t \\ \omega_t\Delta t \end{pmatrix}$$

The notation ˆ**X** refers to a deterministic state prediction. Of course, physical robots are somewhat unpredictable. This is commonly modelled by a Gaussian distribution with mean f(**X**t, vt, ωt) and covariance **Σ**x. (See Appendix A for a mathematical definition.)

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, v_t, \omega_t) = N(\hat{\mathbf{X}}_{t+1}, \mathbf{\Sigma}_x) .$$

Next, we need a sensor model. We will consider two kinds of sensor model. The first assumes that the sensors detect stable, recognizable features of the environment called **landmarks**. For each landmark, the range and bearing are reported. Suppose the robot's state is **x**t =(xt, yt, θt)_ and it senses a landmark whose location is known to be (xi, yi)_. Without noise, the range and bearing can be calculated by simple geometry.

$$\hat{\mathbf{z}}_t = h(\mathbf{x}_t) = \begin{pmatrix} \sqrt{(x_t - x_i)^2 + (y_t - y_i)^2} \\ \arctan \frac{y_i - y_t}{x_i - x_t} - \theta_t \end{pmatrix}$$

A somewhat different sensor model is used for an array of range sensors, each of which has a fixed bearing relative to the robot. Such sensors produce a vector of range values **z**t = (z1, . . . , zM)_. Given a pose **x**t, let ˆzj be the exact range along the jth beam direction from **x**t to the nearest obstacle. As before, this will be corrupted by Gaussian noise. Typically, we assume that the errors for the different beam directions are independent and identically distributed, so we have

$$P(\mathbf{z}_t \mid \mathbf{x}_t) = \alpha \prod_{j=1}^{M} e^{-(z_j - \hat{z}_j)/2\sigma^2}.$$

function MONTE-CARLO-LOCALIZATION(a, z, N, $P(X'|X,\ v,\ \omega)$, $P(z|z^*)$, m) **returns** a set of samples for the next time step

inputs: a, robot velocities v and ω
z, range scan $z_1, \ldots, z_M$
$P(X'|X,\ v,\ \omega)$, motion model
$P(z|z^*)$, range sensor noise model
m, 2D map of the environment

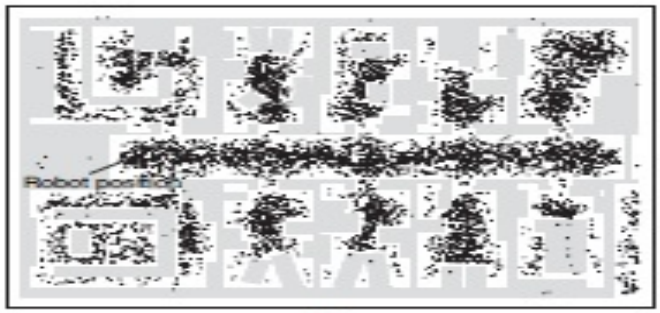
persistent: S, a vector of samples of size N

local variables: W, a vector of weights of size N
S', a temporary vector of particles of size N
W', a vector of weights of size N

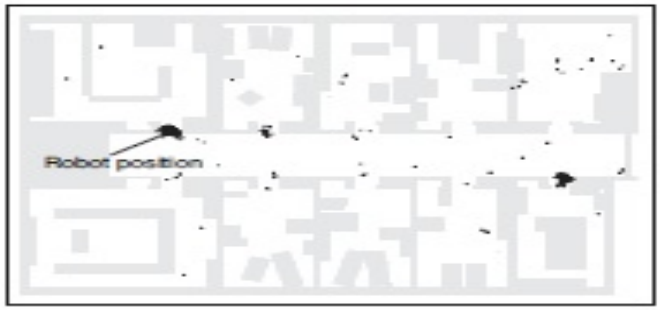
if S is empty **then** /* initialization phase */
  for $i = 1$ to N **do**
    $S[i] \leftarrow$ sample from $P(X_0)$
  for $i = 1$ to N **do** /* update cycle */
    $S'[i] \leftarrow$ sample from $P(X'|X = S[i], v, \omega)$
    $W'[i] \leftarrow 1$
    for $j = 1$ to M **do**
      $z^* \leftarrow$ RAYCAST($j, X = S'[i], m$)
      $W'[i] \leftarrow W'[i] \cdot P(z_j | z^*)$
  $S \leftarrow$ WEIGHTED-SAMPLE-WITH-REPLACEMENT(N, S', W')
return S

A Monte Carlo localization algorithm using a range-scan sensor model with independent noise.

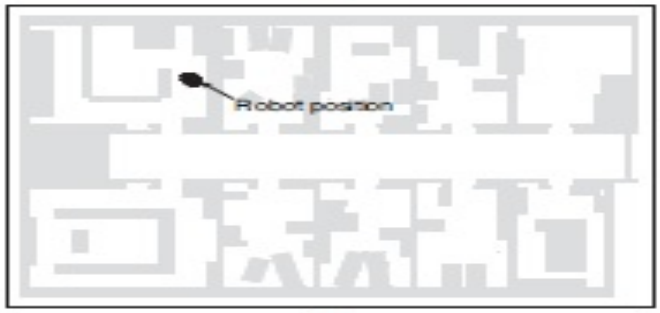
Localization using particle filtering MONTE CARLO is called **Monte Carlo localization**, or MCL. The MCL algorithm is an instance of the particle-filtering algorithm of All we need to do is supply the appropriate motion model and sensor model. shows one version using the range-scan model. The operation of the algorithm is illustrated in as the robot finds out where it is inside an office building. In the first image, the particles are uniformly distributed based on the prior, indicating global uncertainty about the robot's position. In the second image, the first set of measurements arrives and the particles form clusters in the areas of high posterior belief. In the third, enough measurements are available to push all the particles to a single location.



(a)

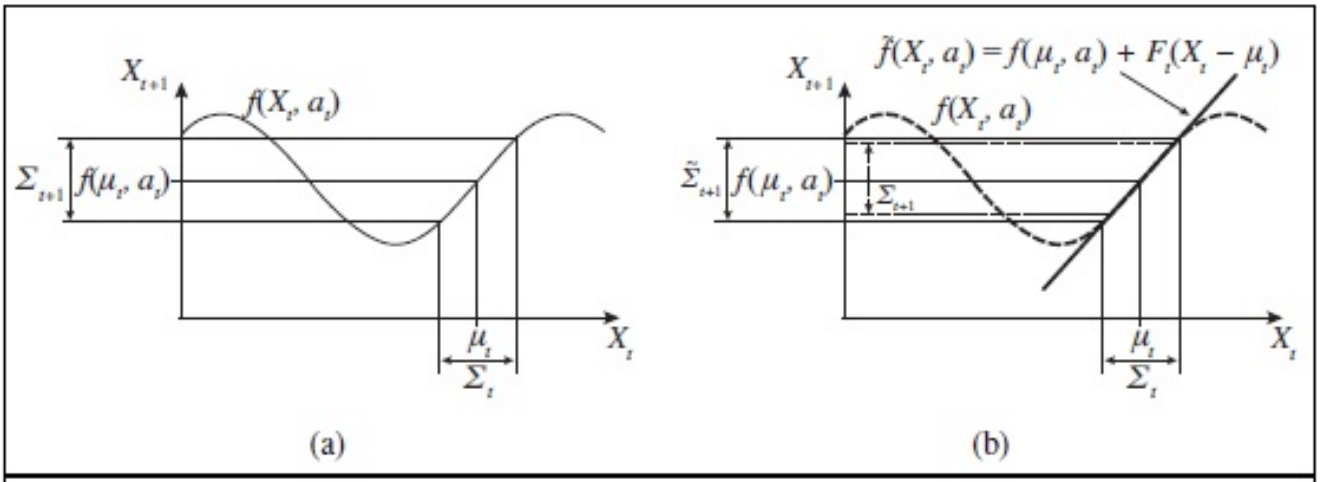


(b)

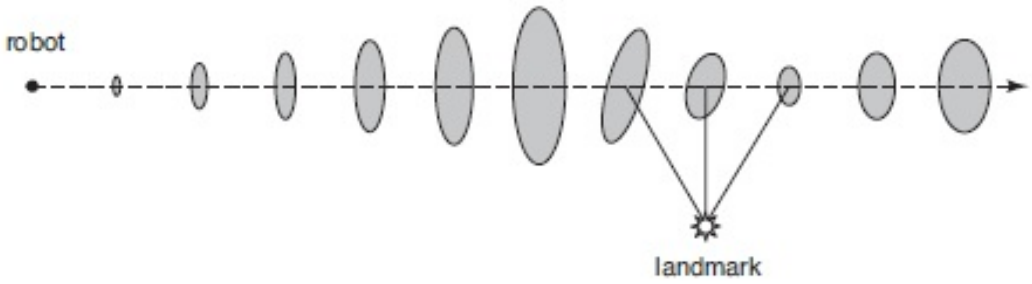


(c)

Monte Carlo localization, a particle filtering algorithm for mobile robot localization. (a) Initial, global uncertainty. (b) Approximately bimodal uncertainty after navigating in the (symmetric) corridor. (c) Unimodal uncertainty after entering a room and finding it to be distinctive.



One-dimensional illustration of a linearized motion model: (a) The function f, and the projection of a mean $\boldsymbol{\mu}_t$ and a covariance interval (based on $\boldsymbol{\Sigma}_t$) into time $t + 1$. (b) The linearized version is the tangent of f at $\boldsymbol{\mu}_t$. The projection of the mean $\boldsymbol{\mu}_t$ is correct. However, the projected covariance $\tilde{\boldsymbol{\Sigma}}_{t+1}$ differs from $\boldsymbol{\Sigma}_{t+1}$.



Example of localization using the extended Kalman filter. The robot moves on a straight line. As it progresses, its uncertainty increases gradually, as illustrated by the error ellipses. When it observes a landmark with known position, the uncertainty is reduced.

the state vector to include the locations of the landmarks in the environment. Luckily, the EKF update scales quadratically, so for small maps (e.g., a few hundred landmarks) the computation is quite feasible. Richer maps are often obtained using graph relaxation methods, similar to the Bayesian network inference techniques.

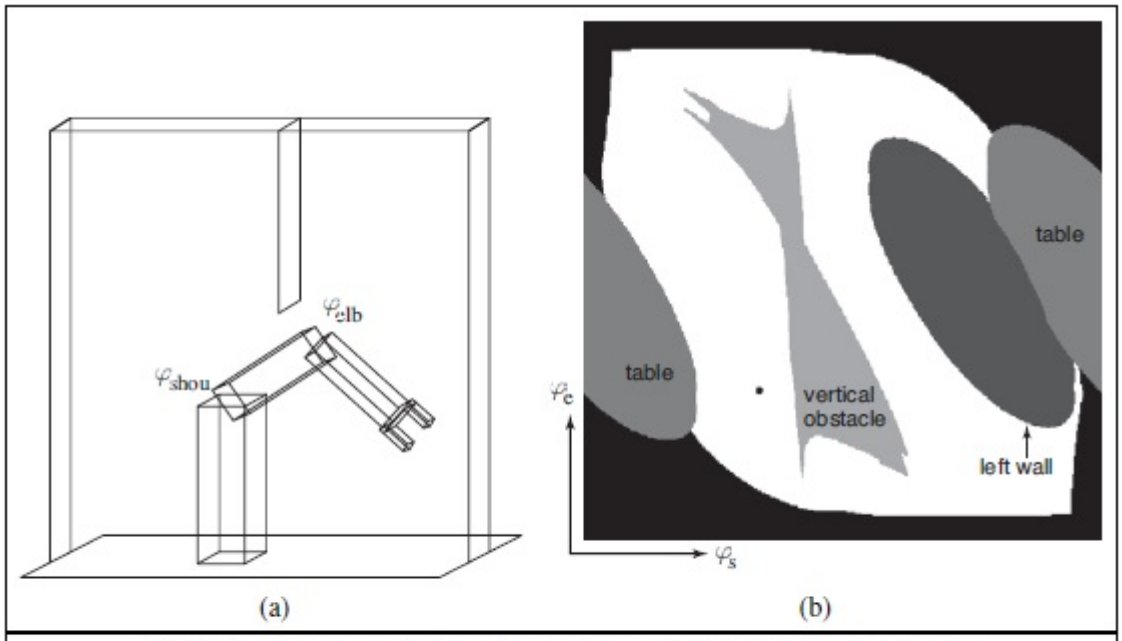
PLANNING TO MOVE

All of a robot's deliberations ultimately come down to deciding how to move effectors. The **point-to-point motion** problem is to deliver the robot or its end effector to a designated target location. A greater challenge is the **compliant motion** problem, in which a robot moves while being in physical contact with an

obstacle. An example of compliant motion is a robot manipulator that screws in a light bulb, or a robot that pushes a box across a table top. We begin by finding a suitable representation in which motion-planning problems can be described and solved. It turns out that the **configuration space**—the space of robot states defined by location, orientation, and joint angles—is a better place to work than the original 3D space. The **path lanning** problem is to find a path from one configuration to another in configuration space. We have already encountered various versions of the path-planning problem throughout this book; the complication added by robotics is that path planning involves continuous spaces. There are two main approaches: **cell decomposition** and **skeletonization**. Each reduces the continuous path-planning problem to a discrete graph-search problem. In this section, we assume that motion is deterministic and that localization of the robot is exact. Subsequent sections will relax these assumptions.

25.4.1 Configuration space

We will start with a simple representation for a simple robot motion problem. Consider the robot arm shown in Figure 25.14(a). It has two joints that move independently. Moving the joints alters the (x, y) coordinates of the elbow and the gripper. (The arm cannot move in the z direction.) This suggests that the robot's configuration can be described by a four-dimensional coordinate: (xe, ye) for the location of the elbow relative to the environment and (xg, yg) for the location of the gripper. Clearly, these four coordinates characterize the full state of the robot. They constitute what is known as **workspace representation**, since the coordinates of the robot are specified in the same coordinate system as the objects it seeks to manipulate (or to avoid). Workspace representations are well-suited for collision checking, especially if the robot and all objects are represented by simple polygonal models. The problem with the workspace representation is that not all workspace coordinates are actually attainable, even in the absence of obstacles. This is because of the **linkage constraints** on the space of attainable workspace coordinates. For example, the elbow position (xe, ye) and the gripper position (xg, yg) are always a fixed distance apart, because they are joined by a rigid forearm. A robot motion planner defined over workspace coordinates faces the challenge of generating paths that adhere to these constraints. This is particularly tricky



(a) Workspace representation of a robot arm with 2 DOFs. The workspace is a box with a flat obstacle hanging from the ceiling. (b) Configuration space of the same robot. Only white regions in the space are configurations that are free of collisions. The dot in this diagram corresponds to the configuration of the robot shown on the left.

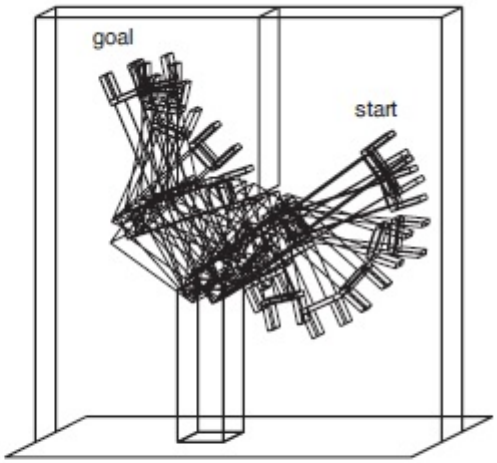
Unfortunately, configuration spaces have their own problems. The task of a robot is usually expressed in workspace coordinates, not in configuration space coordinates. This raises the question of how to map between workspace coordinates and configuration space. Transforming configuration space coordinates into workspace coordinates is simple: it involves a series of straightforward coordinate transformations. These transformations are linear for prismatic joints and trigonometric for revolute joints. This chain of coordinate transformation is known as **kinematics**.

25.4.2 Cell decomposition methods

The first CELL approach to path planning uses **cell decomposition**—that is, it decomposes the free space into a finite number of contiguous regions, called cells. These regions have the important property that the path-planning problem within a single region can be solved by simple means (e.g., moving along a straight line). The path-planning problem then becomes a discrete graph-search problem, very much like the search problems introduced in Chapter 3. The simplest cell decomposition consists of a regularly spaced grid. shows a square grid decomposition of the space and a solution path that is optimal for this grid size. Grayscale shading indicates the *value* of each free-space grid cell—i.e., the cost of the shortest path from that cell to the goal. (These values can be computed by a deterministic form of the VALUE-ITERATION shows the corresponding workspace trajectory for the arm. Of course, we can also use the A

algorithm to find a shortest path. Such a decomposition has the advantage that it is extremely simple to implement, but it also suffers from three limitations. First, it is workable only for low-dimensional configuration spaces, because the number of grid cells increases exponentially with d, the number of dimensions. Sounds familiar? This is the curse! Dimensionality @of dimensionality. Second, there is the problem of what to do with cells that are “mixed”—that is, neither entirely within free space nor entirely within occupied space. A solution path that includes such a cell may not be a real solution, because there may be no way to cross the cell in the desired direction in a straight line. This would make the path planner *unsound*. On the other hand, if we insist that only completely free cells may be used, the planner will be *incomplete*, because it might

(a)



(b)

(a) Value function and path found for a discrete grid cell approximation of the configuration space. (b) The same path visualized in workspace coordinates. Notice how the robot bends its elbow to avoid a collision with the vertical obstacle.

Cell decomposition methods can be improved in a number of ways, to alleviate some of these problems. The first approach allows *further subdivision* of the mixed cells—perhaps using cells of half the original size. This can be continued recursively until a path is found that lies entirely within free cells. (Of course, the method only works if there is a way to decide if a given cell is a mixed cell, which is easy only if the configuration space boundaries have relatively simple mathematical descriptions.) This method is complete provided there is a bound on the smallest passageway through which a solution must pass. Although it focuses most of the computational effort on the tricky areas within the configuration space, it still fails to scale well to high-dimensional problems because each recursive splitting of a cell creates 2d smaller cells. A second way to obtain a complete algorithm is to insist on an **exact cell decomposition** of the free space. This

EXACT CELL method must allow cells to be irregularly shaped where they meet the boundaries of free space, but the shapes must still be “simple” in the sense that it should be easy to compute a traversal of any free cell. This technique requires some quite advanced geometric ideas, so we shall not pursue it further here.

The exact, continuous state that was attained with the cell was first expanded in the search. Assume further, that when propagating information to nearby grid cells, we use this continuous state as a basis, and apply the continuous robot motion model for jumping to nearby cells. In doing so, we can now guarantee that the resulting trajectory is smooth and can indeed be executed by the robot. One algorithm that implements this is **hybrid A***.

25.4.3 Modified cost functions

This problem can be solved by introducing a **potential field**. A potential field is a function defined over state space, whose value grows with the distance to the closest obstacle. shows such a potential field—the darker a configuration state, the closer it is to an obstacle. The potential field can be used as an additional cost term in the shortest-path calculation. This induces an interesting trade off. On the one hand, the robot seeks to minimize path length to the goal. On the other hand, it tries to stay away from obstacles by virtue of minimizing the potential function. With the appropriate weight balancing the two objectives, a resulting path may look like the one shown in Figure 25.17(b). This figure also displays the value function derived from the combined cost function, again calculated by value iteration. Clearly, the resulting path is longer, but it is also safer. There exist many other ways to modify the cost function. For example, it may be desirable to *smooth* the control parameters over time. For example, when driving a car, a smooth path is better than a jerky one. In general, such higher-order constraints are not easy to accommodate in the planning process, unless we make the most recent steering command a part of the state. However, it is often easy to smooth the resulting trajectory after planning, using conjugate gradient methods. Such post-planning smoothing is essential in many real world applications.

PLANNING UNCERTAIN MOVEMENTS

Most of today’s robots use deterministic algorithms for decision making, such as the path-planning algorithms of the previous section. To do so, it is common practice to extract MOST LIKELY STATE the **most likely state** from the probability distribution produced by the state estimation algorithm. The advantage of this approach is purely computational. Planning paths through configuration space is already a challenging problem; it would be worse if we had to work with a full probability distribution over states.

Ignoring uncertainty in this way works when the uncertainty is small. In fact, when the environment model changes over time as the result of incorporating sensor measurements, many robots plan paths online during plan execution. ONLINE REPLANNING This is the **online re-planning** technique.

The field of robotics has adopted a range of techniques for accommodating uncertainty. Some are derived from the algorithms given in for decision making under uncertainty. If the robot faces uncertainty only in its state transition, but its state is fully observable, the problem is best modeled as a Markov decision process (MDP). The solution of an MDP is an optimal **policy**, which tells the robot what to do in every possible state. In this way, it can handle all sorts of motion errors, whereas a single-path solution from a deterministic planner would be much less robust. In robotics, policies are called **navigation** NAVIGATION **functions**. The value FUNCTION

Just as in, partial observability makes the problem much harder. The resulting robot control problem is a partially observable MDP, or POMDP. In such situations, the robot maintains an internal belief state, like the ones discussed. The solution to a POMDP is a policy defined over the robot's belief state. Put differently, the input to the policy is an entire probability distribution. This enables the robot to base its decision not only on what it knows, but also on what it does not know. For example, if it is uncertain INFORMATION about a critical state variable, it can rationally invoke an **information gathering action**. This GATHERING ACTION is impossible in the MDP framework, since MDPs assume full observability. Unfortunately, techniques that solve POMDPs exactly are inapplicable to robotics—there are no known techniques for high-dimensional continuous spaces. Discretization produces POMDPs that are far too large to handle. One remedy is to make the minimization of uncertainty a control object COASTAL For example, the **coastal navigation** heuristic requires the robot to stay near known NAVIGATION landmarks to decrease its uncertainty. Another approach applies variants of the probabilistic roadmap planning method to the belief space representation. Such methods tend to scale better to large discrete POMDPs.

MOVING

So far, we have talked about how to *plan* motions, but not about how to *move*. Our plans particularly those produced by deterministic path planners—assume that the robot can simply follow any path that the algorithm produces. In the real world, of course, this is not the case. Robots have inertia and cannot execute arbitrary paths except at arbitrarily slow speeds. In most cases, the robot gets to exert forces rather than specify positions. This section discusses methods for calculating these forces.

25.6.1 Dynamics and control

Section 25.2 introduced the notion of **dynamic state**, which extends the kinematic state of a robot by its velocity. For example, in addition to the angle of a robot joint, the dynamic state also captures the rate of change of the angle, and possibly even its momentary acceleration. The transition model for a dynamic state representation includes the effect of forces on this rate of change. Such models are typically DIFFERENTIAL expressed via **differential equations**, which are equations that relate a quantity (e.g., a kinematic state) to the change of the quantity over time (e.g., velocity). In principle, we could have chosen to plan robot motion using dynamic models, instead of our kinematic models. Such a methodology would lead to superior robot performance, if we could generate the plans. However, the dynamic state has higher dimension than the kinematic space, and the curse of dimensionality would render many motion planning algorithms inapplicable for all but the most simple robots. For this reason, practical robot system often rely on simpler kinematic path planners. A common technique to compensate for the limitations of kinematic plans is to use a CONTROLLER separate mechanism, a **controller**, for keeping the robot on track. Controllers are techniques for generating robot controls in real time using feedback from the environment, so as to achieve a control objective. If the objective is to keep the robot on a preplanned path, it is often referred to as a **reference controller** and the path is called a **reference path**. Controllers that optimize a global cost function are known as **optimal controllers**. Optimal policies for continuous MDPs are, in effect, optimal controllers. On the surface, the problem of keeping a robot on a prespecified path appears to be relatively straightforward. In practice, however, even this seemingly simple problem has its pitfalls. illustrates what can go wrong; it shows the path of a robot that attempts to follow a kinematic path. Whenever a deviation occurs—whether due to noise or to constraints on the forces the robot can apply—the robot provides an opposing force whose magnitude is proportional to this deviation. Intuitively, this might appear plausible, since deviations should be compensated by a counterforce to keep the robot on track. However, illustrates, our controller causes the robot to vibrate rather violently. The vibration is the result of a natural inertia of the robot arm: once driven back to its reference position the robot then overshoots, which induces a symmetric error with opposite sign. Such overshooting may continue along an entire trajectory, and the resulting robot motion is far from desirable.

ROBOTIC SOFTWARE ARCHITECTURES

A methodology for structuring algorithms is SOFTWARE called a **software architecture**. An architecture

includes languages and tools for writing programs, as well as an overall philosophy for how programs can be brought together. Modern-day software architectures for robotics must decide how to combine reactive control and model-based deliberative planning. In many ways, reactive and deliberate techniques have orthogonal strengths and weaknesses. Reactive control is sensor-driven and appropriate for making low-level decisions in real time. However, it rarely yields a plausible solution at the global level, because global control decisions depend on information that cannot be sensed at the time of decision making. For such problems, deliberate planning is a more appropriate choice.

Consequently, most robot architectures use reactive techniques at the lower levels of control and deliberative techniques at the higher levels. We encountered such a combination in our discussion of PD controllers, where we combined a (reactive) PD controller with a (deliberate) path planner. Architectures that combine reactive and deliberate techniques are called **hybrid architectures**.

25.7.1 Sub Sumption architecture

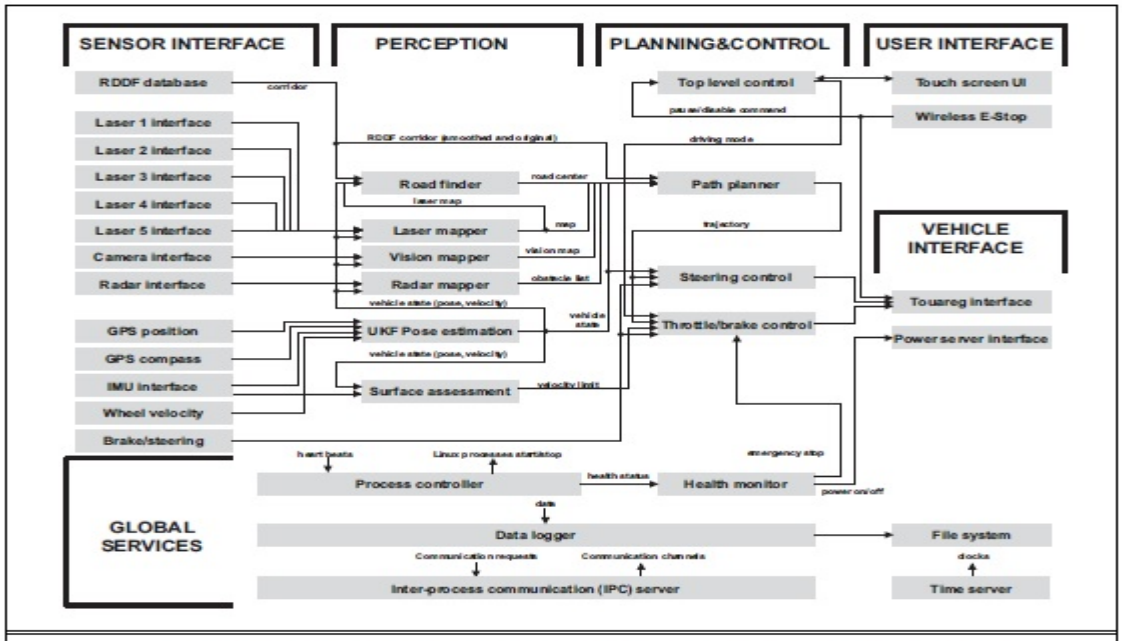
The **sub sumption architecture** (Brooks, 1986) is a framework for assembling reactive con-trollers out of finite state machines. Nodes in these machines may contain tests for certain sensor variables, in which case the execution trace of a finite state machine is conditioned on the outcome of such a test. Arcs can be tagged with messages that will be generated when traversing them, and that are sent to the robot's motors or to other finite state machines. Additionally, finite state machines possess internal timers (clocks) that control the time it takes to traverse an arc. The resulting machines are refereed to as **augmented finite state machines**, or AFSMs, where the augmentation refers to the use of clocks.

An example of a simple AFSM is the four-state machine which generates cyclic leg motion for a hexapod walker. This AFSM implements a cyclic controller, whose execution mostly does not rely on environmental feedback. The forward swing phase, however, does rely on sensor feedback. If the leg is stuck, meaning that it has failed to execute the forward swing, the robot retracts the leg, lifts it up a little higher, and attempts to execute the forward swing once again. Thus, the controller is able to *react* to contingencies arising from the interplay of the robot and its environment.

The sub sumption architecture offers additional primitives for synchronizing AFSMs, and for combining output values of multiple, possibly conflicting AFSMs. In this way, it enables the programmer to compose increasingly complex controllers in a bottom-up fashion.

25.7.2 Three-layer architecture

Hybrid architectures combine reaction with deliberation. The most popular hybrid architecture is the **three-layer architecture**, which THREE-LAYER consists of a reactive layer, an executive layer, and a deliberative layer. The **reactive layer** provides low-level control to the robot. It is characterized by a tight sensor–action loop. Its decision cycle is often on the order of milliseconds. The **executive layer** (or sequencing layer) serves as the glue between the reactive layer and the deliberative layer. It accepts directives by the deliberative layer, and sequences them for the reactive layer. For example, the executive layer might handle a set of via-points generated by a deliberative path planner, and make decisions as to which reactive behaviour to invoke. Decision cycles at the executive layer are usually in the order of a second. The executive layer is also responsible for integrating sensor information into an internal state representation. For example, it may host the robot's localization and online mapping routines.



Software architecture of a robot car. This software implements a data pipeline, in which all modules process data simultaneously.

25.7.3 Pipeline architecture

Another architecture for robots is known as the **pipeline architecture**. Just like the sub sumption architecture, the pipeline architecture executes multiple process in parallel. However, the specific modules in this architecture resemble those in the three-layer architecture. pipeline architecture, which is used to control an autonomous car. Data enters this pipeline at the **sensor interface layer**. The **perception layer**

APPLICATION DOMAINS

Here are some of the prime application domains for robotic technology. **Industry and Agriculture.** Traditionally, robots have been fielded in areas that require difficult human labor, yet are structured enough

to be amenable to robotic automation. The best example is the assembly line, where manipulators routinely perform tasks such as assembly, part placement, material handling, welding, and painting. In many of these tasks, robots have become more cost-effective than human workers. Outdoors, many of the heavy machines that we use to harvest, mine, or excavate earth have been turned into robots. For example, a project at Carnegie Mellon University has demonstrated that robots can strip paint off large ships about 50 times faster than people can, and with a much-reduced environmental impact. Prototypes of autonomous mining robots have been found to be faster and more precise than people in transporting ore in underground mines. Robots have been used to generate high-precision maps of abandoned mines and sewer systems. While many of these systems are still in their prototype stages, it is only a matter of time until robots will take over much of the semi mechanical work that is presently performed by people.

(a) (b)

(a) Robotic car BOSS, which won the DARPA Urban Challenge. Courtesy of Carnegie Mellon University. (b) Surgical robots in the operating room. Image courtesy of da Vinci Surgical Systems.

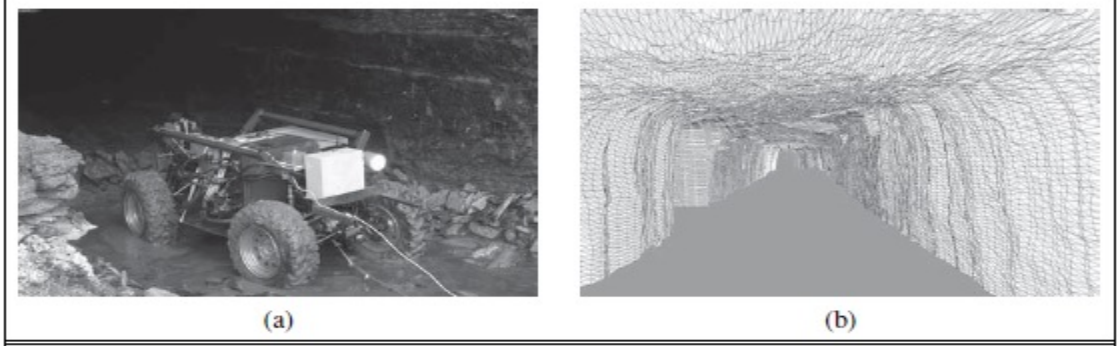
Transportation. Robotic transportation has many facets: from autonomous helicopters that deliver payloads to hard-to-reach locations, to automatic wheelchairs that transport people who are unable to control wheelchairs by themselves, to autonomous straddle carriers that outperform skilled human drivers when transporting containers from ships to trucks on loading docks. A prime example of indoor transportation robots, or gofers, is the Helpmate robot. This robot has been deployed in dozens of hospitals to transport food and other items. In factory settings, autonomous vehicles are now routinely deployed to transport goods in warehouses and between production lines. Many of these robots require environmental modifications for their operation. The most common modifications are localization aids such as inductive loops in the floor, active beacons, or barcode tags. An open challenge in robotics is the design of robots that can use natural cues, instead of artificial devices, to navigate, particularly in environments such as the deep ocean where GPS is unavailable.

Robotic cars. Most of use cars every day. Many of us make cell phone calls while driving. Some of us even text. The sad result: more than a million people die every year in traffic accidents. Robotic cars like BOSS and STANLEY offer hope: Not only will they make driving much safer, but they will also free us from the need to pay attention to the road during our daily commute.

Progress in robotic cars was stimulated by the DARPA Grand Challenge, a race over 100 miles of rehearsed desert terrain, which represented a much more challenging task than had ever been accomplished before. Stanford's STANLEY vehicle completed the course in less than seven hours in 2005, winning a $2 million prize and a place in the National Museum of American History. Figure 25.28(a) depicts BOSS, which in 2007 won the DARPA Urban Challenge, a complicated road race on city streets where robots faced other robots and had to obey traffic rules.

Health care. Robots are increasingly used to assist surgeons with instrument placement when operating on organs as intricate as brains, eyes, and hearts.. Robots have become indispensable tools in a range of surgical procedures, such as hip replacements, thanks to their high precision. In pilot studies, robotic devices have been found to reduce the danger of lesions when performing colonoscopy. Outside the operating room, researchers have begun to develop robotic aides for elderly and handicapped people, such as intelligent robotic walkers and intelligent toys that provide reminders to take medication and provide comfort. Researchers are also working on robotic devices for rehabilitation that aid people in performing certain exercises.

Hazardous environments. Robots have assisted people in cleaning up nuclear waste, most notably in Chernobyl and Three Mile Island. Robots were present after the collapse of the World Trade Center, where they entered structures deemed too dangerous for human search and rescue crews. Some countries have used robots to transport ammunition and to defuse bombs—a notoriously dangerous task. A number of research projects are presently developing prototype robots for clearing minefields, on land and at sea. Most existing robots for these tasks are teleoperated—a human operates them by remote control. Providing such robots with autonomy is an important next step.



(a) (b)

(a) A robot mapping an abandoned coal mine. (b) A 3D map of the mine acquired by the robot.

Entertainment. Robots have begun to conquer the entertainment and toy industry. we see **robotic soccer**, a competitive game very much like human soccer, but played with autonomous mobile robots. Robot soccer provides great opportunities for research in AI, since it raises a range of problems relevant to many other, more serious robot applications. Annual robotic soccer competitions have attracted large numbers of AI researchers and added a lot of excitement to the field of robotics.

Human augmentation. A final application domain of robotic technology is that of human augmentation. Researchers have developed legged walking machines that can carry people around, very much like a wheelchair. Several research efforts presently focus on the development of devices that make it easier for people to walk or move their arms by providing additional forces through extra skeletal attachments.

What is strong AI?

Strong artificial intelligence (AI), also known as artificial general intelligence (AGI) or general AI, is a theoretical form of AI used to describe a certain mindset of AI development. If researchers are able to develop Strong AI, the machine would require an intelligence equal to humans; it would have a self-aware consciousness that has the ability to solve problems, learn, and plan for the future.

Strong AI aims to create intelligent machines that are indistinguishable from the human mind. But just like a child, the AI machine would have to learn through input and experiences, constantly progressing and advancing its abilities over time.

While AI researchers in both academia and private sectors are invested in the creation of artificial general intelligence (AGI), it only exists today as a theoretical concept versus a tangible reality. While some individuals, like Marvin Minsky, have been quoted as being overly optimistic in what we could accomplish in a few decades in the field of AI; others would say that Strong AI systems cannot even be developed. Until

the measures of success, such as intelligence and understanding, are explicitly defined, they are correct in this belief. For now, many use the Turing test to evaluate intelligence of an AI system.

Tests of Strong AI

Turing Test

Alan Turing developed the Turing Test in 1950 and discussed it in his paper, "Computing Machinery and Intelligence" (PDF, 566 KB) (link resides outside IBM). Originally known as the Imitation Game, the test evaluates if a machine's behavior can be distinguished from a human. In this test, there is a person known as the "interrogator" who seeks to identify a difference between computer-generated output and human-generated ones through a series of questions. If the interrogator cannot reliably discern the machines from human subjects, the machine passes the test. However, if the evaluator can identify the human responses correctly, then this eliminates the machine from being categorized as intelligent.

While there are no set evaluation guidelines for the Turing Test, Turing did specify that a human evaluator will only have a 70% chance of correctly predicting a human vs computer-generated conversation after 5 minutes. The Turing Test introduced general acceptance around the idea of machine intelligence. However, the original Turing Test only tests for one skill set — text output or chess as examples. Strong AI needs to perform a variety of tasks equally well, leading to the development of the Extended Turing Test. This test evaluates textual, visual, and auditory performance of the AI and compares it to human-generated output. This version is used in the famous Loebner Prize competition, where a human judge guesses whether the output was created by a human or a computer.

Chinese Room Argument (CRA)

The Chinese Room Argument was created by John Searle in 1980. In his paper, he discusses the definition of understanding and thinking, asserting that computers would never be able to do this. In this excerpt from his paper, from Stanford's website (link resides outside IBM), summarizes his argument well, "Computation is defined purely formally or syntactically, whereas minds have actual mental or semantic contents, and we cannot get from syntactical to the semantic just by having the syntactical operations and nothing else…A system, me, for example, would not acquire an understanding of Chinese just by going through the steps of a computer program that simulated the behavior of a Chinese speaker (p.17)."

The Chinese Room Argument proposes the following scenario:

Imagine a person, who does not speak Chinese, sits in a closed room. In the room, there is a book with Chinese language rules, phrases and instructions. Another person, who is fluent in Chinese, passes

notes written in Chinese into the room. With the help of the language phrasebook, the person inside the room can select the appropriate response and pass it back to the Chinese speaker.

While the person inside the room was able to provide the correct response using a language phrasebook, he or she still does not speak or understand Chinese; it was just a simulation of understanding through matching question or statements with appropriate responses. Searle argues that Strong AI would require an actual mind to have consciousness or understanding. The Chinese Room Argument illustrates the flaws in the Turing Test, demonstrating differences in definitions of artificial intelligence.

Strong AI vs. weak AI

Weak AI, also known as narrow AI, focuses on performing a specific task, such as answering questions based on user input or playing chess. It can perform one type of task, but not both, whereas Strong AI can perform a variety of functions, eventually teaching itself to solve for new problems. Weak AI relies on human interference to define the parameters of its learning algorithms and to provide the relevant training data to ensure accuracy. While human input accelerates the growth phase of Strong AI, it is not required, and over time, it develops a human-like consciousness instead of simulating it, like Weak AI. Self-driving cars and virtual assistants, like Siri, are examples of Weak AI.

Strong AI trends

While there are no clear examples of strong artificial intelligence, the field of AI is rapidly innovating. Another AI theory has emerged, known as artificial superintelligence (ASI), super intelligence, or Super AI. This type of AI surpasses strong AI in human intelligence and ability. However, Super AI is still purely speculative as we have yet to achieve examples of Strong AI.

With that said, there are fields where AI is playing a more important role, such as:

Cybersecurity: Artificial intelligence will take over more roles in organizations' cybersecurity measures, including breach detection, monitoring, threat intelligence, incident response, and risk analysis.

Entertainment and content creation: Computer science programs are already getting better and better at producing content, whether it is copywriting, poetry, video games, or even movies. OpenAI's GBT-3 text generation AI app is already creating content that is almost impossible to distinguish from copy that was written by humans.

Behavioral recognition and prediction: Prediction algorithms will make AI stronger, ranging from applications in weather and stock market predictions to, even more interesting, predictions of human

behavior. This also raises the questions around implicit biases and ethical AI. Some AI researchers in the AI community are pushing for a set of anti-discriminatory rules, which is often associated with the hashtag #responsibleAI.

Strong AI terms and definitions

The terms artificial intelligence, machine learning and deep learning are often used in the wrong context. These terms are frequently used in describing Strong AI, and so it's worth defining each term briefly:

Artificial intelligence defined by John McCarthy (PDF, 109 KB) (link resides outside IBM), is "the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable."

Machine learning is a sub-field of artificial intelligence. Classical (non-deep) machine learning models require more human intervention to segment data into categories (i.e. through feature learning).

Deep learning is also a sub-field of machine learning, which attempts to imitate the interconnectedness of the human brain using neural networks. Its artificial neural networks are made up layers of models, which identify patterns within a given dataset. They leverage a high volume of training data to learn accurately, which subsequently demands more powerful hardware, such as GPUs or TPUs. Deep learning algorithms are most strongly associated with human-level AI.

To read more about the nuanced differences between these technologies, read "AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?"

Deep learning applications

Deep learning can handle complex problems well, and as a result, it is utilized in many innovative and emerging technologies today. Deep learning algorithms have been applied in a variety of fields. Here are some examples:

Self-driving cars: Google and Elon Musk have shown us that self-driving cars are possible. However, self-driving cars require more training data and testing due to the various activities that it needs to account for, such as giving right of way or identifying debris on the road. As the technology matures, it'll then need to get over the human hurdle of adoption as polls indicate that many drivers are not willing to use one.

Speech recognition: Speech recognition, like AI chatbots and virtual agents, is a big part of natural language processing. Audio-input is much harder to process for an AI, as so many factors, such as background noise, dialects, speech impediments and other influences can make it much harder for the AI to convert the input into something the computer can work with.

Pattern recognition: The use of deep neural networks improves pattern recognition in various applications. By discovering patterns of useful data points, the AI can filter out irrelevant information, draw useful correlations and improve the efficiency of big data computation that may typically be overlooked by human beings.

Computer programming: Weak AI has seen some success in producing meaningful text, leading to advances within coding. Just recently, OpenAI released GPT-3, an open-source software that can actually write code and simple computer programs with very limited instructions, bringing automation to program development.

Image recognition: Categorizing images can be very time consuming when done manually. However, special adaptions of deep neural networks, such as DenseNet, which connects each layer to every other layer in the neural network, have made image recognition much more accurate.

Contextual recommendations: Deep learning apps can take much more context into consideration when making recommendations, including language understanding patterns and behavioral predictions.

Fact checking: The University of Waterloo recently released a tool that can detect fake news by verifying the information in articles by comparing it with other news sources.