Outline

- What is a robot?
- Robotic perception
- Robotic planning
- Robotic movement
- Putting it together
What is a robot?

- “Robots are physical agents that perform tasks by manipulating the physical world”
- A combination of sensors and effectors
- Three main “types” of robot
  - Manipulators are stationary – robot arms
  - Mobile robots move – unmanned vehicles
  - Mobile manipulators combine the two – Asimo, killbots...
- Partially observable, stochastic, dynamic, continuous environments that can’t operate faster than real time
- We have left the wumpus world.
What is a robot? – Sensors

- Two categories of sensors
  - Passive sensors capture signals generated by the environment
  - Active sensors send energy into the environment
- Sense environment, robot location, or robot configuration
- Range finders find distance to objects (radar, whiskers...)
- Location sensors localize the robot (GPS, beacons...)
- Proprioceptive sensors tell the robot about itself
  - Lets massive robots do delicate tasks without crushing everything!
What is a robot? – Effectors

- Give robots the means to move and change shape
- Can be described by degrees of freedom (DOF)
- A static rigid mobile robot has 6DOF: x, y, z, yaw, pitch, roll
- Dynamic robots add another 6, the velocities of those
- Non-rigid robots add additional DOF within themselves
  - Wrists have 3DOF, elbows have 2DOF
What is a robot? – Effectors

- DOFs may not be controllable – think about a car
  - 3DOF - can be maneuvered to any x, y point in any orientation
  - Can only control 2DOF
- Cars have 3 effective DOF, but 2 controllable DOF
- This is problematic
Robotic Perception

- Partially observable environment, so Kalman filters, HMMs and dynamic Bayes nets work
- Include the robot’s past actions as observed variables
- We want to compute the new belief state from the previous belief state
  - \( P(X_{t+1} \mid z_{1:t+1}, a_{1:t}) \) from \( P(X_t \mid z_{1:t}, a_{1:t-1}) \)
- Similar to chapter 15, but we take actions into account and variables are continuous, so \( \int \) rather than \( \sum \)
  - \( P(X_{t+1} \mid z_{1:t+1}, a_{1:t}) = \alpha P(z_{t+1} \mid X_{t+1}) \int P(X_{t+1} \mid x_t, a_t) P(x_t \mid z_{1:t}, a_{1:t-1}) dx_t \)
- From this, we can recursively estimate state
Robotic Perception

Figure 25.7 Robot perception can be viewed as temporal inference from sequences of actions and measurements, as illustrated by this dynamic Bayes network.
“Localization is the problem of finding out where things are – including the robot itself”

Example: a flat 2D world, and the robot has a map

- Position \((x, y)\) and heading of \(\theta\), so \(X_t = (x_t, y_t, \theta_t)^T\)

An action is a translational velocity and a rotational velocity, so \(X_{t+1} = f(X_t, a_t) = f(X_t, v_t, \omega_t)\)

Robots are unpredictable so \(X_t\) is a gaussian distribution
Robotic Perception – Localization

- Assume our robot has sensors that detect stable, recognizable landmarks
- Assuming no noise, simple geometry can give us our robot’s position
- Alternatively, assume our robot has fixed sensors that produce a vector of range values
Robotic Perception – Localization

- Landmark localization works well when there are landmarks that will be consistently visible and identifiable. Otherwise, the robot knows nothing.
- Localization can be performed using particle filtering or a Kalman filter (chapter 15)
Robotic Perception – Localization
Monte Carlo Localization

Robot position
Robot Perception – Localization
Monte Carlo Localization

Robot position
Robot Perception – Localization
Monte Carlo Localization
Robot Perception – Localization

- Kalman filters are an alternative
- However, Gaussian beliefs are only closed under linear models, so use an extended Kalman filter
- Uncertainty grows as the robot moves
- BUT uncertainty decreases when the robot sees a landmark
- EKFs are useful when there are many landmarks that the robot can see
Robot Perception – Localization

- What if the robot doesn’t have a map?
- This problem set is known as simultaneous localization and mapping or SLAM
- EKFs work well for this if the state vector is changed to include the locations of landmarks
- Graph relaxations methods and expectation-maximization may also be used
Robotic Planning

- All robot plans decide how to move the robot’s effectors
  - Move from point A to point B
  - Screw in a light bulb
  - Push a box off a cliff
- Consider the configuration space of the robot
  - The specification of its degrees of freedom
- Two major methods for planning – cell decomposition and skeletonization
- Consider a robot arm with two DOF
Robotic Planning
Robotic Planning

- Translating from configuration space coordinates to workspace coordinates is easy, but the inverse is much more difficult.
- Our robot has 0-2 configuration space mappings for any workspace coordinate.
- Mapping the entire configuration space is silly, generate a configuration and then test it.
Robotic Planning – Cell Decomposition

- Decompose the free space in the configuration space into cells
- Turns the path-planning problem into a graph search
- However, this only works for low numbers of DOF
- Also, what to do about partially-filled cells?
  - Subdivide further
  - Allow irregular cells
- Steer clear of obstacles by using a potential field
Robotic Planning – Skeletonization

- Reduce the robot’s free space to a lower-dimensional representation
- Voronoi graph – the robot moves to a point on the graph using a straight-line motion, moves to the point nearest the target, then moves to the target using a straight-line motion
- Does not give the shortest path especially in large free spaces
- Difficult to apply to higher-dimensional spaces
- Computing it can be expensive, especially with irregular objects
Robotic Planning - Skeletonization

- Probabilistic roadmaps offer more possible routes
- Randomly a large number of configurations, discarding those that don’t fall into free space
- Join nodes that are easy to transition between
- Add start and end configurations and the problem is a discrete graph search
- Could be incomplete due to a bad set of random points
- Possible to bound the probability of failure by generating more points and using geometry
- Can also direct points towards “good” areas
- Scales fairly well to higher dimensions
Robotic Planning - Skeletonization
Robotic Planning - Uncertainty

- Effectors always have some margin of error
- Planning paths is already challenging without working with full a probability distribution over each state
- Extract the most likely state and go with it
  - Works well when there is little uncertainty
- Many robots use online replanning (11.3.3)
- Markov decision processes work well for fully observable environments
- Must use a partially observable MDP in partially observable environments – bases decisions on what the robot does not know as well as what it knows
  - No known techniques to solve in high-dimensional continuous spaces, so minimize uncertainty or use a probabilistic roadmap
Robotic Planning – Uncertainty

- Robust methods are another approach
- Assumes a bounded amount of uncertainty without assigning probabilities.

**Figure 25.19** A two-dimensional environment, velocity uncertainty cone, and envelope of possible robot motions. The intended velocity is $v$, but with uncertainty the actual velocity could be anywhere in $C_v$, resulting in a final configuration somewhere in the motion envelope, which means we wouldn’t know if we hit the hole or not.

**Figure 25.20** The first motion command and the resulting envelope of possible robot motions. No matter what the error, we know the final configuration will be to the left of the hole.
Robot Movement

- Robots can’t necessarily follow all plans due to physical limitations, like inertia.
- We could plan using the robots dynamic state instead of its kinematic state, but that has much higher dimensionality.
- Use a separate mechanism to control the robot and keep it on track.
- Whenever a deviation occurs, the controller steers the robot back on track.
Robot Movement

- P-controllers exert force in negative proportion to the error: \( a_t = K_P(y(t) - x_t) \) where \( K_P \) is the gain parameter
  - This causes oscillation
- PD-controllers extend P-controllers with a differential:
  \[
  A_t = K_P(y(t) - x_t) + K_D(\frac{\delta(y(t) - x_t)}{\delta_t})
  \]

**Figure 25.22**  Robot arm control using (a) proportional control with gain factor 1.0, (b) proportional control with gain factor 0.1, and (c) PD (proportional derivative) control with gain factors 0.3 for the proportional component and 0.8 for the differential component. In all cases the robot arm tries to follow the path shown in gray.
Robot Movement

- PD-controllers fail to account for error that isn’t part of the model (wear and tear...)
- PID-controllers do! (Proportional Integral Derivative)
  \[ A_t = K_P (y(t) - x_t) + K_I \int (y(t) - x_t) dt + K_D \frac{\delta(y(t) - x_t)}{\delta_t} \]
Robot Movement

- Potential field control works very well (hill-climbing) but fails if a local minima traps the robot.

**Figure 25.23**  Potential field control. The robot ascends a potential field composed of repelling forces asserted from the obstacles and an attracting force that corresponds to the goal configuration. (a) Successful path. (b) Local optimum.
Robot Movement

- Reactive control uses environmental feedback
  - If a robot’s leg hits something while it’s moving, move it back, lift it higher and try again

- Reinforcement learning control works incredibly well, however the policy search needs an accurate domain model
  - Let the robot create its model based on the actions of a human expert – 4 minutes of learning was enough to teach a robot how to do a flip with a helicopter
Robotic Architecture

- Most robot architectures use reactive techniques at low levels and deliberate techniques at higher levels.
- Subsumption architectures use an augmented finite state machine.
- Three layer architectures are very robust.
  - Reactive layer provides low-level control (milliseconds).
  - Executive layer takes a plan and decides which reactive behaviors to invoke.
  - Deliberative layer generates global solutions using planning (minutes).
Robotic Architecture

- Pipeline architectures run in parallel, so very fast
  - Sensor layer passes sensor data to the
  - Perception layer, which updates the robot’s internal model and passes them to the
  - Planning and control layer, which adjusts the robot’s internal plans and instructs the
  - Vehicle interface layer, which performs the actions
Questions?