Planning and control

IAR Lecture 13
Barbara Webb
The planning/control problem

• What should our robot do next?
  – N.B. could refer to short or long time horizon

• How can we bring about a desired state of the robot and/or world?
  – Complete a task, probably against disturbances.

• What control policy will satisfy the robot’s goals within the robot and world constraints?
The planning/control problem

Some typical examples:

- Get robot from A to B, within certain time
- Complete a mission within power constraints
- Map an area to a given level of accuracy
- Decide between alternative routes, e.g., uncertain shortcut vs. well-known path
- Stay on the road and don’t collide with anything
Consider problem of steering a car on a racetrack. Might have:

- Input: distance from edge, $y$
- (Internal) state: heading, $x$
- Output: steering angle, $u$
- Disturbances: undulating track

Want to determine a policy: $u = \pi(x, y)$

Multiple possible approaches e.g.:

- Open loop: pre-programmed sequence of actions
- Feedback: Turn wheel based on distance from edge
- Feedforward: Make corrections based on upcoming turn
The planning/control problem

Planning and control essentially refer to the same thing - deciding what the robot will do.

- Deliberative reasoning
- Typically offline
- Far time horizon
- Sensing only if plan fails

- Trajectory planning
- Look ahead but modify
- Near future
- Sensing used to monitor

- Low level control
- Online
- Immediate
- Sensing used directly

May use offline planning to construct an executable controller.
Have already covered a number of forms of control:

- **Reactive:** problem of designing direct sensorimotor connections to obtain robust behaviour
- **Lecture 3:** Sensors that extract property needed for action
- **Lecture 4:** Simple wiring in Grey Walter’s ‘tortoise’
Have already covered a number of forms of control:

- Lecture 5: designing the right intrinsic dynamics

\[
\dot{\theta}_i = 2\pi v_i + \sum_j (w_{ij} \sin(\theta_j - \theta_i - \phi_{ij}))
\]

\[
\ddot{r}_i = a_i \left( \frac{a_i}{4} (R_i - r_i) - \dot{r}_i \right)
\]

\[
x_i = r_i (1 + \cos(\theta_i))
\]

An isolated oscillator converges to:

\[
x_i^\infty(t) = R_i (1 + \cos(2\pi v_i t + \theta_0))
\]

Setpoints:

\[
\phi_i = x_i - x_{N+i}
\]
Have already covered a number of forms of control:

- Lecture 6: Behaviour based approaches
- Different solutions to problem of combining behaviours
  - Hierarchical
  - Weighted combination
  - Weighted selection
Have already covered a number of forms of control:

- Feedback control: problem of obtaining/maintaining desired set-point, e.g. PID (from IVR lectures)

\[ T(t) = K_p (\theta_d - \theta(t)) - K_d \dot{\theta}(t) + K_i \int_{t_0}^{t} (\theta_d - \theta(t))dt \]
Have already covered a number of forms of control:

- Using maps: problem of finding a feasible path
Path planning

The generalisation of a map is a configuration space:

• Given the kinematic description of the robot (e.g. holonomic or not, 3-d motion, articulation)
• Describes the possible state space of the robot (each point in the space is a possible pose or configuration of the robot)
• Can map kinematic obstacles into the C-space
• And thus represent the possible connected trajectories that can be taken

N.B. Here would need a corresponding C-space for every orientation of the robot to represent pose
Path planning

• Simple but reliable method – the bug algorithm

1. Draw a line AB from Start A to Goal B
2. Move along the line till you hit an obstacle
3. At an obstacle, circumnavigate it’s perimeter till line AB is met again
4. Continue till goal

• Can lead to rather convoluted paths
  • Can you think of a smarter Step 3 ??
• Guaranteed to get to the goal
Path planning

Typically want to find the shortest path:
• Convert configuration space to a graph
• E.g. Voronoi method or Grid method

• Apply search methods
Path planning

• For a graph with nodes connected by edges

\[ f(n) = g(n) + \epsilon h(n) \]

• \( f(n) \) is the “goodness” of the path via node \( n \)
• \( g(n) \) is the “cost” of going from the Start to node \( n \)
• \( h(n) \) is the cost of going from \( n \) to the Goal
• \( c(n, n') \) is cost from node \( n \) to adjacent node \( n' \)
Breadth-first search

If $\varepsilon=0$, and $c(n,n')$ is constant for all $n$ (e.g. in grid) then breadth first search will find optimal route.
A* Heuristic Function

If $\varepsilon = 1$, and $c(n,n')$ is not constant, A* is a more efficient search. Like breadth-first except always expand the ‘best’ (least cost) node first (note same method with $\varepsilon = 0$ is Dijkstra’s algorithm)

$$f^*(n) = g^*(n) + h^*(n)$$

- $g^*(n)$ is easy: just sum up the path costs to $n$
- $h^*(n)$ is tricky
  - But if began with metric map, may know the *direct* distance between any two nodes, even if not what path is needed to get between them.
  - Thus a minimal estimate of the remaining cost we can use for $h^*(n)$ is the direct distance between $n$ and Goal
Example: A to E

- But since you’re starting at A and can only look 1 node ahead, this is what you see:
• Two choices for n: B, D
• Do both
  – \( f^*(B) = 1 + 2.24 = 3.24 \)
  – \( f^*(D) = 1.4 + 1.4 = 2.8 \)
• Expand the most plausible path first => A-D-?-E
• A-D-?-E
  – “stand on D”
  – Can see 2 new nodes: F, E
  – \( f^*(F) = (1.4+1)+1 = 3.4 \)
  – \( f^*(E) = (1.4+1.4)+0 = 2.8 \)

• Three paths
  – A-B-?-E >= 3.24
  – A-D-E = 2.8
  – A-D-F-?-E >=3.4

• A-D-E is the winner!
  – Don’t have to look farther because expanded the shortest first, others couldn’t possibly do better without having negative distances, violations of laws of geometry…
Planning as optimisation

• The problem of finding a control policy or plan can be formulated as an optimisation problem:
  – Robot has goals, but actions have costs
  – Express both as a single ‘pay-off’ function, e.g.,

\[
 r(x,u) = \begin{cases} 
 +100 & \text{If reach the desired state} \\
 -1 & \text{Otherwise, i.e., cost for each time step} 
\end{cases}
\]

Aim is to maximise the cumulative expected pay-off:

\[
 R_T = E \left[ \sum_{\tau=1}^{T} \gamma^{\tau} r_{T+\tau} \right] 
\]

Where \(0 < \gamma < 1\) is a discount factor, making distant reward less attractive.
Optimal Policy

- 1-step optimal policy:
  \[ \pi_1(x) = \arg\max_u r(x,u) \]

- Value function of 1-step optimal policy:
  \[ V_1(x) = \gamma \max_u r(x,u) \]

- T-step optimal policy is defined recursively:
  \[ \pi_T(x) = \arg\max_u \left[ r(x,u) + \sum_{i=1}^N V_{T-1}(x_i)p(x_i | u, x) \right] \]

- In theory this can sometimes be solved; in practice usually obtain the value function by iteration till the approximation converges

\[ N \text{ is number of possible states} \]

\[ \text{Value of the next state} \]

\[ \text{State transition probabilities} \]
What about measurements?

- Previous method assumes that we know what state we are in (uncertainty only in probability of transition from $x$ to $x'$ for control action $u$): hence is a Markov Decision Process (MDP).
- Extension needed to deal with (realistic) situation that state is uncertain, i.e., we have a belief about the state, which can be expressed as a probability distribution.
- Measurements are now introduced: making a measurement may reduce the uncertainty about the state, allowing a better estimate of the values, and hence better policy choice.
- This is a Partially Observable Markov Decision Process (POMDP): beyond the scope of this course but see Reinforcement Learning for details.
Summary

• Classic planning methods often assume noise-free, deterministic worlds.
• Even probabilistic approaches need to limit the state space / action space to be feasible, e.g., compute value function of robot co-ordinates but not velocities.
• Can we use low-level reactive/feedback control to deal with the uncertainty or state variables that we cannot plan for?
• This is the basic idea of hybrid control, see next lecture.

References:
Robin Murphy, “Introduction to AI Robotics”
Roland Siegwart & Illah Nourbakhshsh “Introduction to Autonomous Robotics”