Lecture 1: 9/30/19
Course introduction

Computational Robotics **ECE209AS – Fall 2019**
Prof. Ankur Mehta *Transcribed by: Ankur Mehta, Alexie Pogue*

1.1 About this course

1.1.1 About the prof

`https://uclalemur.com`

1.1.2 Class options

You can pick either of two tracks for this class:

- Track 1: Foundational
- Track 2: Research

The assignments are different for the two tracks, as seen in table 1.1.

Foundational	**Research**
30% labs (10% each)	10% project proposal
30% lit review presentation	40% project presentation
30% lit review writeup	40% project writeup
10% participation	10% participation

Table 1.1: The two tracks have different assignments and grade weightings.

More details are available in the lecture 1 notes on CCLE.

1.2 Robots and Robotics

What is a robot? See, for instance, figure 1.1.

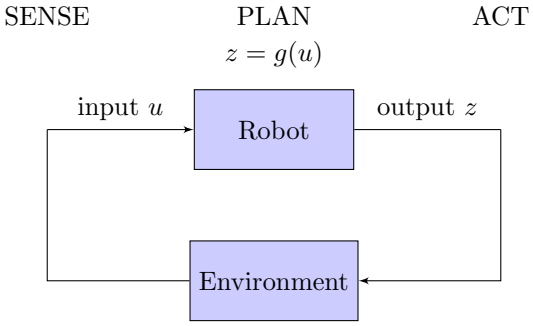
Figure 1.1: Robots are our friends.

Lecture 2: 10/02/19
Systems and MDPs

Computational Robotics **ECE209AS – Fall 2019**
Prof. Ankur Mehta *Transcribed by: Mandar Deshpande, Kenny Chen*

2.1 Robot

What is a "robot"? A machine that can **SENSE**, **PLAN**, and **ACT**.



2.2 System

Some input and output relationship.

$$z = g(u)$$

Types of systems:

- Linear vs. Non-Linear
  - Linear systems must satisfy the superposition principle

$$g(\alpha u_1 + \beta u_2) = \alpha g(u_1) + \beta g(u_2)$$

- Stable vs. Unstable / Chaotic vs. Non-Chaotic
- Causal vs Non-Causal
  - Causal: output only depends on past and present inputs
  - Non-Causal: output depends on past, present, and future inputs
- Memoryless vs Stateful
  - Memoryless if its output at a given time is dependent only on the input at that same time

- Deterministic vs. Stochastic
  - Deterministic: same input always generates the same output
  - Stochastic: same input may generate different outputs
- Discrete vs. Continuous
  - State space, time, or actions
- Time Dependent vs. Time Independent / Time Variant vs. Time Invariant
- Controllable / Observable
  - Controllability: can we achieve all states of the system?
  - Observability: does any state change generate a change in the system output?
- Scalar / Vector / Separable
  - Single/multi input/output
  - Separable: can a multivariate system be decomposed to many single systems?

In the first half of this class, we will be studying discrete systems. In the second half of the class, we will move and study continuous systems. Note that the underlined above are the assumptions we will make throughout the quarter.

2.3 States

2.3.1 Definition and Markov Property

States, in the context of robotics, are one or more variables which represent the "status" of the system.

Suppose we have a memoryless system of the form:

$$z(t) = g(u(t), t)$$

This function only considers the inputs at time t within its calculation. Now suppose we want to consider all inputs; this can be written in the form:

$$z(t) = g(u(0), u(1), ..., u(t), t)$$

This is called a "causal stateful system." Unfortunately, these systems are hard to deal with, partly because of the varying input size (i.e., the number of inputs differ at time $t = 1$ and $t = 2$). To simplify the problem, we can group all the inputs from time 0 to time $t - 1$ into the current state vector $x(t)$, with the assumption that the current state vector $x(t)$ captures the effects of the inputs up until time t. Thus, with this assumption, we can rewrite the system as:

$$z(t) = h(x(t), u(t), t)$$

and the state vector gets updated at each time step via:

$$x'(t) = f(x(t), u(t), t)$$

which captures the system dynamics, or how the system evolves over time. The assumption that we just made is called the *Markov Property*.

2.3.2 Common Robotics Problems

- Modeling and System Identification
  - Given u and z, determine f and h (if we have the inputs and outputs of a system, can we reverse engineer the system functions?)
- State Estimation / Perception
  - Determine x given u and z (the inputs and outputs)
- Control
  - Given z, determine u (determine the inputs necessary to accomplish the outputs)
- Planning
  - Given the task, determine $z \Rightarrow u$
- Design
  - Given the task, determine f and h

2.3.3 Space vs. Time

	Time		
Space		*Discrete*	*Continuous*
	Discrete	Markov Processes	Hybrid Systems
	Continuous	Discrete-time Dynamical Systems	Continous-time Dynamical Systems

2.3.4 Markov Models

	Controllability		
Observability		*Full*	*Partial*
	Full	MDP	Markov Chain
	Partial	POMDP	Hidden Markov Models

2.4 Markov Decision Processes (MDPs)

$\mathcal{S}$: state space $\{s_i\}$, $||\mathcal{S}|| = N_s$

Lecture 3: 10/02/19
Markov Decision Process

Computational Robotics **ECE209AS – Fall 2019**
Prof. Ankur Mehta *Transcribed by: Meet Taraviya*

Markov Decision Processes (MDPs) model stateful systems with a discrete state space, a discrete action space, full observability, randomness and a discrete notion of time. Formally, an MDP consists of :-

- **States**, S

  S is the set of all states of the system, and it is discrete and finite. We use N_S to denote $|S|$.

- **Actions**, A

  A is the set of all states of the system, and it is also discrete and finite. We use N_A to denote $|A|$.

- **Transition probabilities**, $P : S \times A \times S \to \mathbb{R}_{\geq 0}$

  In an MDP, performing the same action a at the same state a may lead the agent to different states at different times. $P_{sa}(s') = P(s, a, s')$ gives the probability that the agent goes to the state s', on taking the action a from the state s'. Axioms of probabiity require that $\sum_{s' \in S} P_{sa}(s') = 1$. This stochastic component models uncertain/noisy behaviour of actuators.

These 3 components are enough to specify the *control* problem on MDPs. The control problem asks for the action a that takes the agent from a state s to another state s'. Because of randomness, it may not be possible to guarantee reaching a particular state s', so instead we find the action that maximises the probability of reaching s' in the next step. So, $a^* = \text{argmax}_{a \in A} P_{sa}(s')$ solves the control problem.
If we're interested in the *planning* problem, we also need three more components:-

- **Reward function**, $R : S \times A \times S \to \mathbb{R}$

  The agent also receives some rewards for his actions. Rewards provide a mechanism for him to identify "good" actions. $R(s, a, s')$ gives the reward the agent receives when he takes the action a at the state s and goes to the state s'. The agent should behave as to maximise "cumulative reward".

- **Horizon**, $H \in \mathbb{N} \cup \{\infty\}$

  H denotes the number of steps the agent has to accrue reward. When we are interested in modelling long term interactions between the agent and the environment, we may also take $H = \infty$.

- **Discount factor**, $0 \leq \gamma \leq 1$

  In many real word applications, immediate rewards are more useful than rewards in distant future. This is captured by the concept of discounting. Rewards after t steps are discounted by a factor of γ^t. $\gamma = 0$ means that the agent only cares about the reward in the next step. Whereas $\gamma \sim 1$ means that future rewards are almost as important as immediate rewards.

Thus, starting from $i = 0$, the agent takes an action a_i whenever he is at a state s_i; the MDP dynamics take him to some state s' and gives him a reward of r_t. This is represented in the form of a trajectory: $s_0 a_0 r_0 s_1 a_1 r_1 ....$ The agent is interested in maximising the cumulative reward $r_{tot} = \sum_{i=0}^{H-1} R(s_t, a_t, s_{t+1})$. He has control over what actions to take at each state, in the form of a *policy* π which maps each state to some action. Since there are N_S states and the agent has to choose from N_A actions at each of this state, there are ${N_A}^{N_S}$ policies in total. We use Π to denote the set of all policies. The *value* of a state s under some policy π, $V^\pi(s)$ is defined as the expectation of the cumulative reward the agent gets, starting from s and following the policy π and the MDP dynamics.
In the *planning* problem the agent wants to find the policy that maximises the expectation of cumulative reward:-

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} V^\pi \text{ such that } \forall t \geq 0 : a_t = \pi(s_t),\ s_{t+1} \sim p_{s_t a_t} \text{ and } r_t = R(s_t, a_t, s_{t+1})$$

Here, we are saying $V^\pi \geq V^{\pi'}$ when $V^\pi(s) \geq V^{\pi'}(s) \forall s$. π^* is known as an optimal policy for the MDP and the corresponding value function $V^\star$ as the optimal value function. It is known that an optimal policy always exists.
From this point we'll assume $H = \infty$, which will be useful in simplifying the math. Because of the infinite horizon, we can relate cumulative rewards starting from two consecutive steps in the form of Bellman back-up equations:-

$$V^\pi(s) = \sum_{s'} P(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$$

These are N_S linear equations in N_S variables - $\{V^\pi(s) | s \in S\}$. So they can be easily solved to find V^π given π.
When using an optimal policy, the agent should not be able to benefit by deviating from the policy at some step, as that would contradict the optimality of the policy. This idea is captured using the Bellman optimality conditions which V^* must satisfy:-

$$V^*(s) = \max_{a \in A} \big[\sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^*(s')] \big]$$

If we solve these equations, we get the optimal value function. To get an optimal policy, we use the same equations with "argmax" operators instead of "max" operators :-

$$\pi^*(s) = \underset{a \in A}{\operatorname{argmax}} \big[\sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^*(s')] \big]$$

3.1 Value Iteration

Value iteration is an iterative approach to find an optimal policy by first finding the optimal value function using the Bellman optimality conditions. We assume that the next's state current value is optimal and use that to get a more accurate estimate of V^*. It can be proved that the algorithm converges if $\gamma < 1$.

Algorithm 1: Value Iteration Algorithm

Result: Optimal Value function V^*
Initialize $V_0(s) = 0\ \forall s \in S$
do
  $V_{i+1}(s) = \max_{a \in A} \big[\sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V_i(s')] \big]$
until $V converges$;

3.2 Policy Iteration

Policy iteration starts with some initial policy - prespecified or selected randomly. Then it repeatedly evaluates the current policy, identifies the optimal actions with respect to the current value function and

updates the policy. The new policy obtained this way is known to be better than the previous policy. Since there are only ${N_A}^{N_S}$ policies, this algorithm converges in at most ${N_A}^{N_S}$ iterations.

Algorithm 2: Policy Iteration Algorithm

Result: Optimal policy π^*
Initialize $\pi_0 \in \Pi$
do
  Evaluate the value function V_i of policy π_i
  $\pi_{i+1}(s) = \text{argmax}_{a \in A} \left[\sum_{s'} P(s,a,s')[R(s,a,s') + \gamma V_i(s')] \right]$
until π *converges*;

Lecture 4: 10/09/19
Fully and Partially Observable MDPs

Computational Robotics **ECE209AS – Fall 2019**
Prof. Ankur Mehta *Transcribed by: Joshua Hannan, Hamza Khan, and Zach Harris*

4.1 Q-Learning

Previously, we defined the optimal value function $V^*(s)$, as follows:

$$V^*(s) = \max_{a \in A} \left\{ \mathbb{E}_{s' \in S} \left[\sum_t \gamma^t r_t \right] \right\}, \quad \forall s \in S$$

We can define the expectation function in this equation as follows:

$$Q^*(s,a) = \mathbb{E}\left[r(s,a,s') + \gamma V^*(s') \right], \quad \forall s \in S \tag{4.1}$$

Equation (4.1) represents the expected discounted reward ("quality") for executing action a at state s and following the policy thereafter. We can obtain the following relationships from Q

$$V^*(s) = \max_{a \in A} Q^*(s,a), \quad \forall s \in S \tag{4.2}$$

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} Q^*(s,a), \quad \forall s \in S \tag{4.3}$$

4.2 Bellman Operator

We define Bellman Operators, T, as operators that transform a function to a function: $T : f \to f$. As is shown shortly, Bellman operators are linear mappings. A Bellman Operator operating on a value function is given by the following

$$TV(s) = \mathbb{E}\left[r(s,a,a') + \gamma V(s') \right] \tag{4.4}$$

4.3 Fixed Point Relationship

Next, we define the fixed point relationship: If $a = \pi(s) \implies T^\pi$ and $a = \pi^*(s) \implies T^*$, we conclude that $Q^*(s,a) = T^\pi V^*, \pi(s) = a$. Thus, we obtain $V^* = TV^*$, which is known as the "fixed point relationship." There are two ways to find the fixed point—value iteration and policy iteration.

4.3.1 Value Iteration

Value iteration works "backwards" by iterating over the horizon H in the following algorithm

Algorithm 3: Value Iteration Algorithm

Result: Fixed Point V^*
Initialize $H = 0 : {}^{H}V^*(s) = 0 \quad \forall s \in S$;
do
  $\quad {}^{H+1}V = f({}^{H}V) = T^* \, {}^{H}V$
until *Convergence*;

Argument for convergence

Starting with any ${}^{H=0}V$, repeatedly applying T^* will cause V to monotonically increase. Furthermore, since MDPs are discrete by definition, V has a maximum value, V^*. Thus, $\lim_{H\to\infty} T^* \, {}^{H}V = V^*$. Note: a more rigorous proof is provided at http://www.cs.cmu.edu/afs/cs/academic/class/15780-s16/www/slides/mdps.pdf, but is beyond the scope of the class.

4.3.2 Policy Iteration

Policy iteration repeatedly performs two steps, policy evaluation and policy refinement, to iterate over policy in the following algorithm

Algorithm 4: Policy Iteration

Result: Fixed Point V^*
Initialize Arbitrary ${}^{0}\pi$;
do
  1. Policy Evaluation: Compute ${}^{i}V^{\pi}$ under policy i;
    ${}^{i}V^{\pi} = {}^{i}T^{\pi} \, {}^{i}V^{\pi}$;
  2. Policy Refinement: Compute the new policy ${}^{i+1}\pi$;
    ${}^{i+1}\pi = \text{argmax}_a \, T \, {}^{i}V^{\pi}$
until *Value Doesn't Change*;

Linearity of Bellman Operator

We make the following observations

$$V : S \to R \implies V \in \mathbb{R}^{N_S} \tag{4.5}$$

$$P, R : S \times A \times S \to \mathbb{R} \overset{\pi}{\implies} P, R \in \mathbb{R}^{N_s \times N_s} \tag{4.6}$$

From (4.4), (4.5), and (4.6), we conclude that

$$\begin{aligned} V = TV &= \sum_{s'} P(R + \gamma V) \\ &= \text{diag}(PR^T) + \gamma PV \\ \implies V &= (I - \gamma P)^{-1} PR \end{aligned} \tag{4.7}$$

Since (4.7) is a linear system of equations, we conclude that the Bellman Operator is linear.

4.3.3 Value vs Policy Iteration

Several differences exist between value and policy iteration. Specifically, value iteration has more steps with smaller gains per step, meaning that its iterations can stop before convergence and still remain fairly accurate. Contrarily, policy iteration takes fewer steps, each with a larger change. Furthermore, solving (4.7) can be computationally demanding for a large number of states, N_s. In general, however, value iteration and policy iteration can be used in conjunction to efficiently find the optimal value function / policy.

4.4 Example: MDP Definition

4.4.1 Reward Shaping

When defining rewards, we should not bias our outputs by setting various rewards along the expected path (**reward shaping**). This can lead to unexpected problems.

In the upcoming example, for example, we should give a positive reward on our goal point, give a negative reward for undesirable states, but we should not give a partial reward for areas near our goal state. If we do, we are biasing our results to a particular path, rather than letting the path planning determine the best path.

4.4.2 Problem Definition

Suppose we want to go and get ice cream. We want to find the optimal route to an ice cream store. We have two types of obstacles: first, we have buildings in our way that we cannot cross. Second, we have a road that we can cross, but doing so would be extremely undesirable (since it's so dangerous for pedestrians).

Specifically, suppose our world is modeled by Figure 4.1. The star represents our start point. The "X" shapes represent the buildings we cannot cross. The locations to the right with reward R = -100 represent the road (we set a negative reward to emphasize to the algorithm to avoid the location). Finally, we have two possible ice cream stores we can visit: Diddy Riese (less desirable, hence the reward R = 1), or Saffron and Rose (better, so it has reward R = 10).

The various arrows drawn, labeled A through D, show possible paths that will be further explained in the following section.

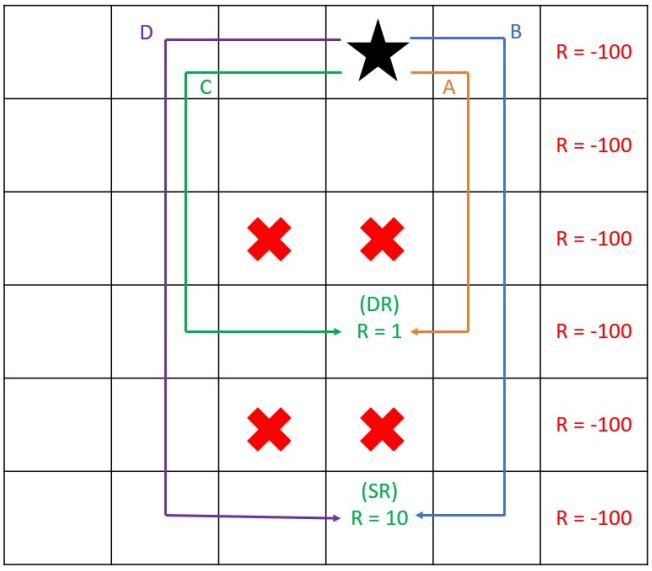


Figure 4.1: Example output routes given different parameters.

4.4.3 Possible Solutions

In this path planning problem, we have two main additional parameters we can set. First, we have the **discount factor** γ, which rewards getting to the location more quickly and penalizes slower routes. Second, we have p_e, which is the **Error Probability**. This comes into account because, as we move from one state to another, there is a probability that we end up in a different state than expected.

	$\gamma \to 0$	$\gamma \to 1$
$P_e \to 0$	A	B
$P_e \to 0.5$	C	D

Figure 4.2: Tradeoff between discount and error probability.

Figure 4.2 shows the various paths, as drawn in Figure 4.1, that would be taken with different parameter values for γ and p_e.

Let's first suppose that p_e is close to 0. This means that it is unlikely to make an error as we follow our desired route. Then, suppose we have γ close to 1, meaning we don't really care about how long it takes to get to the end goal. In this case, we will take our time and go to the location with higher reward, Saffron and Rose, using path B. However, if we have (for example) a tight schedule and need ice cream as quickly as possible, we would have a low γ closer to 0. Then, we would probably want to go to Diddy Riese since it is closer, despite the lower reward. We would take path A instead.

Next, suppose that we have a high probability of error in either direction, indicated by p_e close to 0.5. Then, if we take either path A or B, we have a high probability of accidentally moving one space over and going into the road, with reward -100. As a result, we want to actually avoid the regions close to the road on the right side. Instead, we would want to approach the ice cream stores from the left. Then, if we had urgency (low γ), we would take Path C to Diddy Riese. If we had plenty of time, we would instead take Path D to Saffron and Rose.

4.4.4 Key Takeaway

This example illustrates the reason we don't want to bias our results by rewarding being close to the end goal. If we had done so in the above example, we would have been biased towards particular routes and neglected other routes. However, as we can see through the example, different parameters (such as discount factor or error probability) can cause different optimal routes; we should simply reward a favorable end state and let the algorithm decide the best route(s) to get there.

4.5 Partially Observable MDPs (POMDPs)

In the previous example MDPs, we assumed that we knew exactly what state we are in (full observability). However, what happens if we don't know which state we are in? This is called a **Partially Observable MDP**.

Previously, we defined our MDPs as having the following two known functions:

$$x' = f(x, u)$$
$$z = h(x, u)$$

The first equation contains $\{S, A, P\}$ (our state, actions, and probabilities). The second equation contains $\{O\}$ (our observations). In this case, we don't have direct access to x' or our state, but we do have access to our observations from z.

Instead of having a known state, we instead create a **Belief State**, $Bel(s)$, which maps each state in S to a probability from 0 to 1. The sum of probabilities $Bel(s)$ for each s in S should sum up to 1

($\sum_{s \in S} Bel(S) = 1$). In this formulation, $Bel(S)$ is actually a function of time which we update every time we have a new observation.

The formal definition of the Belief State can be stated as follows:

$$Bel(s_t) = f(a_{0\ to\ t-1}, o_{0\ to\ t}, s_0) \tag{4.8}$$

In Equation 4.8, our Belief State is based off of all of our actions taken up to this point ($a_{0\ to\ t-1}$), all observations ($o_{0\ to\ t}$), and possibly also our initial state s_0, if it is known.

4.5.1 Computing Belief State

In order for us to understand how the Belief State changes with respect to our observations and actions, we will look at the order of the events that occur and adapt the Belief State each time. In Figure 4.3, all actions, a_t, and all observations, o_t, are known. The states, s_t, are unknown.

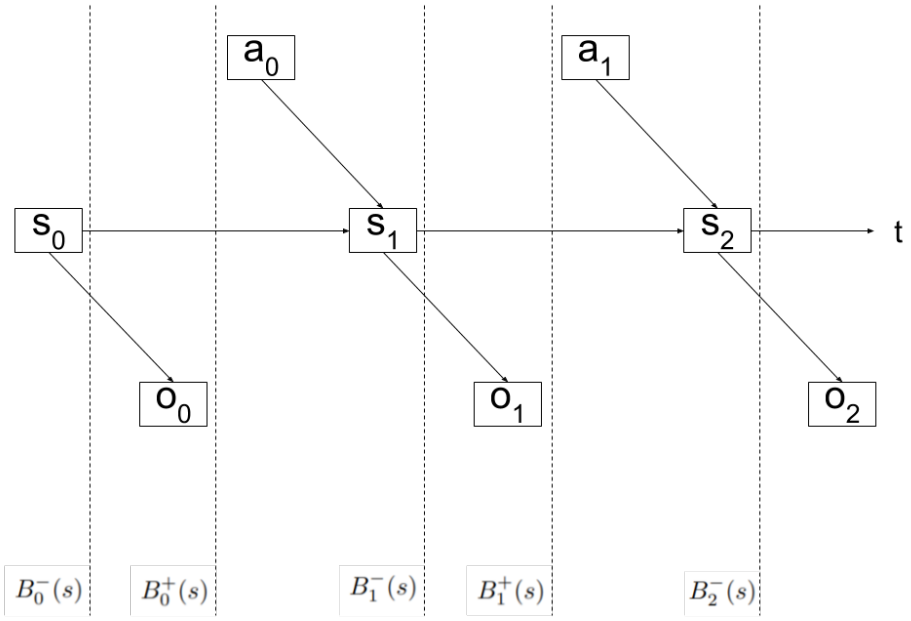


Figure 4.3: Changes in the Belief State over time, t.

We begin with an initial belief of the system state, $Bel_0^-(s)$. This belief can either be derived from an initial state if s_0 is provided, or by determining the most likely possible state of the system if s_0 is not provided. Although the Bell State is operating on s_0, we place the subscript on Bel to show that Bel is the function we care about changing. Equation 4.10 shows how we calculate this Bell state, generalizing to the current time interval t. $P(s_{t+1})$ is shorthand for writing $P(s(t+1) = s_{t+1})$. You will notice we use the state transition probabilities, $P(s, a, s')$, to make this this decision.

$$Bel_{t+1}^-(s) = P(s_{t+1} | a_{0\ to\ t}, o_{0\ to\ t-1}, s_0) \tag{4.9}$$

Once we have formed an initial Belief State, we observe the environment around us. This observation will change our Belief State as well as govern the action that follows. The act of performing our observation may also change our state, however for simplicity we do not include this case. To represent the updated Belief State, we introduce the $Bel_{t+1}^+(s)$ term. Equation 4.10 shows this term, and how we compute $Bel_{t+1}^+(s)$ given the transition probabilities. Notice that we now account for the last observation, o_t.

$$Bel_{t+1}^+(s) = P(s_t | a_{0\ to\ t}, o_{0\ to\ t}, s_0) \tag{4.10}$$

By taking action a_t, we will change the state of our system. This is why, after action a_t is applied, we will move to state s_{t+1} Our Belief State must also adapt accordingly. We therefore require a way of moving from $Bel_t^+(s)$ to $Bel_{t+1}^-(s)$.

However, the Belief State represents the probability of being in any state. Since there is no certainty of being in any particular state, we will show that it is necessary to compute over all previous likely states in order to achieve all present likely states. To show this, we will first write out $Bel_t^+(s)$ and $Bel_{t+1}^-(s)$ in a simpler manner. We let $A = a_{0\ to\ t}, o_{0\ to\ t}$ and $B = a_t$. Then, following this format, we get the set of equations in Equation 4.11.

$$\begin{aligned} Bel_{t+1}^+(s) &= P(s_t|A) \\ Bel_t^-(s) &= P(s_{t+1}|A,B) \end{aligned} \tag{4.11}$$

By the definition of conditional probability in Equation 4.12 and the definition of P_{sa} in Equation 4.13, we present the equation to produce $Bel_{t+1}^-(s)$ provided $Bel_t^+(s)$. This is shown in Equation 4.14.

$$\begin{aligned} &P(A,B) = P(B) * P(A|B) \\ &if P(B) \neq 0 \end{aligned} \tag{4.12}$$

$$P_{sa}(s') = p(s_{t+1}|s_t, a_t) \tag{4.13}$$

$$Bel_{t+1}^-(s) = \sum_{s_t} p(s_{t+1}|s_t, a_t) * Bel_t^+(s) \tag{4.14}$$

4.5.2 Concluding Remarks

We defined an MDP system which cannot know for certain which state the robot is in as a Partially Observable MDP. In order to select which action to perform, we defined an uncertain representation of our state as a Belief State. By looking at the Belief State life cycle we define two states, $Bel_t^+(s)$ and $Bel_{t+1}^-(s)$, that capture the periods of transition during our decision process. We defined a way of moving through the entire Belief State life cycle from Figure 4.3. In this manner, a robot can act on its environment without knowing for certain which state it is in. Although this solution achieves what we desire, it requires a large amount of computation. In the next lecture we will characterize the computation complexity and provide solutions for simplifying this complexity.