Foundations of Robotics, Fall 2018

Coding Project 5: Uncertainty
(CS 4750: 95 points, CS 5750: 100 points)

Due Friday, Dec. 7 at 3:00 PM

Objective:

Demonstrate understanding of methods of handling uncertainty in robotics applications.

Collaboration Policy:

This assignment can be discussed as a group of arbitrary size. You may work with up to one partner on this assignment, but each student is responsible for writing and submitting a separate solution, and **no student should see another student's solution**. Include in a comment at the top of each file your name and the names of all classmates you discussed any part of the project with.

Using the Simulator Tool:

Use the simulator tool to run the provided code for the assignment. **For this assignment, you will always need to run `./simulator-tool run p5 main` to run the framework code.** We have made the simulation start automatically, so you will no longer need to press the play button.

If you encounter difficulty using the simulator tool, please contact the course staff. Note that you can also run `./simulator-tool --help` to get a listing of commands and `./simulator-tool <command> --help` to get more information on the usage of a specific command.

Debugging Tips:

If you add `output="screen"` to your launch file after you specify `type="something.py"`, the print statements from that Python file will print to your terminal. We recommend using `rostopic echo` to see what messages are being published. Refer to the ROS documentation for more information.

We will have a FAQ and clarifications post for this assignment pinned on Piazza, so please check that post regularly for updates and information.

Assignment:

The goal of this assignment is to enable a KUKA youBot to find which of five plastic open-bottom blocks conceals a steel bearing ball, just as in the shell game (see Figure 1). While the blocks were being scrambled, the youBot was not paying close enough attention and completely lost track of the ball. For all it knows, each block has an equal probability of concealing the ball. Fortunately, the youBot is equipped with a metal detector that it can use to reduce its uncertainty about the ball's location. Just like any real sensor, the metal detector is not completely reliable and may return false positives (detection of nonexistent metal) or false negatives (failure to detect metal). To complicate the matter further, the youBot's tendency to not pay close enough attention applies to everything that it does. For any action it is commanded to carry out, there is a nonzero chance it will carry out a different action. You must account for this as well as the noisiness of the youBot's metal detector when writing code to locate and move to the block that conceals the ball.

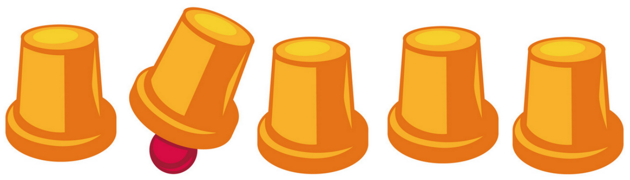

Figure 1: In the shell game, a ball is hidden under one of several identical containers.

When you have finished the assignment, compress your complete ROS package with the command `./simulator-tool package p5` and upload the compressed package (named `p5_solutions.tar.gz`) to CMS. Note that you can upload to CMS as many times as you'd like up until the deadline; the most recent submission will override all the previous ones. Keep this in mind and don't wait until the last minute to submit! We recommend uploading at least a half hour before the deadline to ensure that your solutions are submitted on time.

Remember to run `./simulator-tool check 5` prior to submitting, which verifies that your submission is runnable and can be graded correctly. If this command returns errors or warnings, you will lose points on the assignment. Bear in mind that a successful return from the `check` command does not guarantee full points on this assignment.

1 Using the Metal Detector (30 points)

Since its metal detector is unreliable, it is unlikely that the youBot will be able to confidently select a block based on a single reading. In this problem, you will use **Bayesian inference** to combine the youBot's individual beliefs about the blocks in the workspace into a probability distribution that represents its belief about the location of the ball (see Figure 2).

Let x_R be the position of the robot, and let x_i be the location of the i-th block. Define B_i to be a Bernoulli random variable such that

$$B_i = \begin{cases} 1 & \text{if the } i\text{-th block conceals the ball,} \\ 0 & \text{if the } i\text{-th block does not conceal the ball.} \end{cases} \tag{1}$$

Finally, define Z_i to be the Boolean measurement of whether the i-th block conceals the ball. The accuracy of the measurement is negatively affected by a larger distance between the robot and the block. To use the metal detector on a specific block, call the service `/vrep/youbot/detect_metal`, which will return Z_i as well as the distance from the youBot to the block (for the service definition, see `p5/DetectMetal.srv`). You may assume that consecutive sensor readings for a given block are conditionally independent.

After obtaining a measurement Z_i for the i-th block from a distance of $d_i = ||x_i - x_R||$, you can use Bayes' rule to model the belief about whether the block conceals the ball as

$$bel(B_i) = \mathrm{P}(B_i|Z_i, d_i) = \frac{\mathrm{P}(Z_i|B_i, d_i)\mathrm{P}(B_i|d_i)}{\mathrm{P}(Z_i|d_i)}. \tag{2}$$

The event that the i-th block conceals the ball is independent of the distance from which a sensor reading occurs and therefore $\mathrm{P}(B_i|d_i) = \mathrm{P}(B_i)$. Thus, we may rewrite (2) as

$$bel(B_i) = \frac{\mathrm{P}(Z_i|B_i, d_i)\mathrm{P}(B_i)}{\mathrm{P}(Z_i|d_i)}, \tag{3}$$

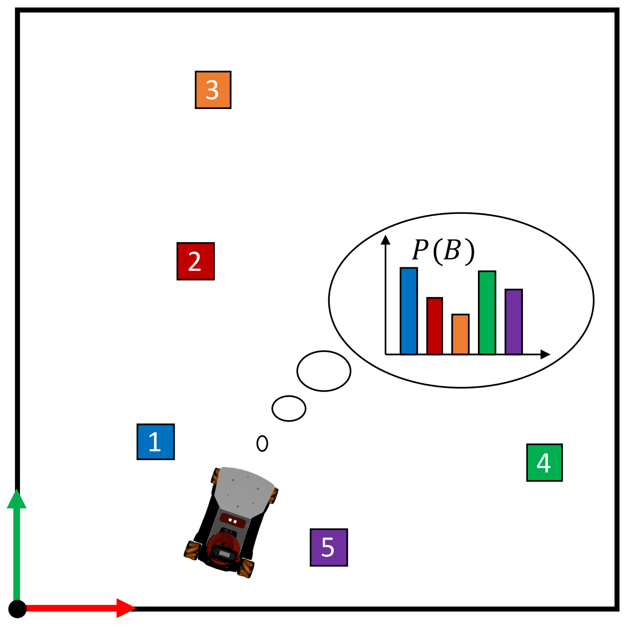

Figure 2: The youBot's belief about which block conceals the ball is represented as a probability distribution over the five blocks.

where $\mathrm{P}(B_i)$ is the prior probability of of the i-th block concealing the ball. Recall that before taking any readings, the youBot believes that each block has an equal probability of concealing the ball. Use this information to to define your initial prior probability distribution.

Next, by the law of total probability, we can rewrite the denominator of (3) as follows:

$$\mathrm{P}(Z_i|d_i) = \mathrm{P}(Z_i|B_i = 1, d_i)\mathrm{P}(B_i = 1) + \mathrm{P}(Z_i|B_i = 0, d_i)\mathrm{P}(B_i = 0). \tag{4}$$

In order to use (3) and (4) to infer the location of the ball, you need a model of the metal detector. The following model incorporates the sensor's dependency on the distance to the measured block:

$$\mathrm{P}(Z_i = 1|B_i = 1, d_i) = \frac{4 - \tanh(3(d_i - 1.5)) - \tanh(3)}{4}, \tag{5}$$

$$\mathrm{P}(Z_i = 1|B_i = 0, d_i) = \frac{\tanh(3(d_i - 1.5)) + \tanh(3)}{4}. \tag{6}$$

Equation (5) represents the case where the i-th block conceals the ball and the sensor correctly detects the ball in the i-th block. (To determine the probability where the i-th block conceals the ball and the sensor incorrectly does *not* detect the ball, simply compute $1 - \mathrm{P}(Z_i = 1|B_i = 1, d_i)$). As d_i decreases, the measurement Z_i will be 1 more consistently, indicating higher confidence that the block conceals the ball. On the other hand, if d_i is large enough, the sensor reading of a block is no better than flipping a coin, and the probability approaches 0.5. Equation (6) follows a similar logic but for the case that the i-th block does not conceal the ball. Try plotting both of these functions to better understand their behavior.

Once you have computed an updated belief about each block, you need to renormalize them, as

$$\mathrm{P}(B_i) = \frac{bel(B_i)}{\sum_{j=1}^{5} bel(B_j)}. \tag{7}$$

Implementation Instructions. Create a service called `problem1` of type `p5/SenseBall` that takes in the prior probability distribution over the five blocks and returns the updated distribution given by (7). All code for this problem should go in `src/problem1.py`.

We have provided a visualization for you to help you determine whether your service is working properly. To run the visualizer, you will need to complete the launch file `visualize.launch` (note that it requires both Problem 1 and Problem 2 to be running). Once you have done that, run the main framework code with `./simulator_tool run p5 main`, and in a separate terminal window, run `roslaunch p5 visualize.launch` (do not forget to `source` your setup file). The robot will drive around randomly while the visualizer node calls your service and plots the result on a graph.

2 Computing Entropy (5 points)

Now that you have implemented a service that gives you the probability estimates of the ball being hidden under each of the blocks, you need a way of determining how uncertain the youBot is about the location of the ball based on those probability estimates. In other words, you need to be able to determine the **entropy** of the probability distribution. In this problem, you will write a service that takes in a probability distribution and returns its entropy, computed as

$$H_p(B_i) = -\sum_{i=1}^{5} p(B_i) \log_2 p(B_i). \tag{8}$$

Implementation Instructions. Create a service called `problem2` of type `p5/ComputeEntropy` that takes in a distribution over the five blocks and returns the entropy of the distribution as defined by (8). All code for this problem should go in `src/problem2.py`.

We have provided a visualization for you to help you determine whether your service is working properly. To run the visualizer, you will need to complete the launch file `visualize.launch` (note that it requires both Problem 1 and Problem 2 to be running). Once you have done that, run the main framework code with `./simulator_tool run p5 main`, and in a separate terminal window, run `roslaunch p5 visualize.launch` (do not forget to `source` your setup file). The robot will drive around randomly while the visualizer node calls your service and plots the result on a graph.

3 Finding a Policy (35 points)

Being able to compute the entropy of a probability distribution gives you a formal way to compare distributions and in doing so inform the decisions that the youBot makes. In this problem, you will find a **policy** for the robot to follow that allows it to reduce its entropy about the location of the ball over time and ultimately determine which block conceals the ball.

A policy π is a function that takes a state s as input and returns the action $\pi(s) = a$ that the robot should take when in state s. You will determine a policy for the robot to follow by implementing a solver for a **Markov Decision Process (MDP)**. An MDP is a stochastic process in which the robot has influence over the state transitions of the system by executing actions. Let the MDP be represented by a tuple (S, A, P_a, R_a, γ), where

- S is the finite set of states,

- A is the finite set of actions,
- $P_a(s, s') = \mathrm{P}(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability that executing action a in state s at time t will lead to state s' at time $t + 1$,
- $R_a(s, s')$ is the immediate reward received after transitioning from state s to state s' due to action a, and
- $\gamma \in [0, 1)$ is the discount factor on future rewards.

At each timestep t, the process is in some state s, and the youBot takes an action a selected from A. Taking this action makes the process transition to a new state s' and gives the robot the corresponding reward $R_a(s, s')$. State s' depends on the action a and the previous state s but is not deterministic since the MDP transitions between states stochastically.

The current state at timestep $t = i$ is a tuple $(x_{R_i}, \mathrm{P}(B_i))$, where x_{R_i} is the location of the robot and $\mathrm{P}(B_i)$ is the current probability distribution over the five blocks. The youBot starts in state $(x_{R_0}, \mathrm{P}(B_0))$, at an arbitrary location $x_{R_0} = \ell_0$ with $\mathrm{P}(B_0)$ a uniform distribution over the five blocks. Once the youBot leaves ℓ_0, it has no reason to go back there, so the only locations it can be commanded to go to are locations ℓ_1–ℓ_5 corresponding to the locations of the five blocks. Note that the state is fully observable: the current location is known, as is the current distribution. That is why the location of the ball, which is not observable, is not part of the state.

Actions fall into three categories, which are as follows:

- The `SENSE` action probes for the ball by calling your `problem1` service. `SENSE` updates the probability distribution that estimates the location of the ball, but it does not move the robot.
- The `MOVEn` action, where `n` $\in$ [`1`, `5`], attempts to move the robot to the location ℓ_n.
- The `STOP` action ends the search and declares that the ball is hidden under the adjacent block.

Bear in mind that the youBot does not pay close enough attention to its actions and therefore can end up at a different state than intended. For example, if the youBot is commanded to move to ℓ_1 from ℓ_2, it should take action `MOVE1`, but there is a chance (determined by $P_{\texttt{MOVE2}}(1, 3)$) that the robot could move to ℓ_3 instead of ℓ_1. If the youBot is commanded to `SENSE`, there is a chance (given by the transition probabilities) that instead of sensing it moves to a new location.

There are multiple strategies for solving MDPs. In this assignment, you will use a strategy called **value iteration**, wherein the policy function π is never explicitly computed. Instead, the action $\pi(s)$ corresponding to state s is computed on an as-needed basis. To do this, you must calculate the **value** of state s, denoted $V(s)$, which is the expected sum of discounted rewards for state s. The action and value must be computed jointly, as

$$\pi(s) = \underset{a}{\operatorname{argmax}} \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V(s') \right) \right\},$$
$$V(s) = \sum_{s'} P_{\pi(s)}(s, s') \left(R_{\pi(s)}(s, s') + \gamma V(s') \right).$$

Note that computing the value of a state requires recursive calls to obtain the values of neighboring states. Since the value of any given state is potentially needed many times, it is stored in memory after being computed so that it can be looked up again without further computation. This process is called *memoizing* or *dynamic programming* and represents a trade-off between computer memory and computation time. In this case, computing the value of a state is expensive enough that the amount of computation time we save is well worth the memory required to store the values. In order to store computed values, use `set_stored_value`. To retrieve a previously stored value, call `get_stored_value`, which will return `None` if that state has not yet been seen.

Your goal in selecting actions for the youBot should be to reduce the entropy of $\mathrm{P}(B)$ until the youBot is confident about where the ball is located. Since the metal detector is more reliable with closer proximity, it will be more effective to move and then sense rather than staying in place and sensing. The rewards of the MDP will be set to enforce this policy. Once the entropy is sufficiently reduced, the actions that the robot needs to take are to move to the block it believes conceals the ball and then perform the STOP action.

Implementation Instructions. Create a service called `problem3` of type `p5/ChooseAction` that takes in a state, selects an action for the robot to take in that state, and returns that action. All code for this problem should go in `src/problem3.py`. Note that this service should not command the robot to do anything; it should only determine an action. For this reason, there is not a separate launch file for this problem. Your service will be used as part of the remaining three problems.

In your implementation, you will fill in two functions, `get_policy` and `value_iteration`, which compute the policy and value respectively for a given state. In order to implement value iteration as shown above, you will need to perform mutual recursion, meaning that each of these functions recursively calls the other one.

4 Commanding the Robot (10 points)

You now have all the pieces that you need to give the youBot commands and win the shell game. For this problem, you will implement a script that continuously calls your `/problem3` service and executes the actions returned by it until the service returns a `STOP` action.

Implementation Instructions. Create a service called `problem4` of type `p5/LocateBall` that takes in an initial State, commands the robot to perform actions as directed by the MDP you implemented in Problem 3, and returns the final State. Note that State is a message of type `p5/State`. In order to command the robot to perform actions, you should call the service `/command_robot`, which is provided to you. All code for this problem should go in `src/problem4.py`. You will also need to fill in the launch file `launch/problem4.launch`. Remember that you need to use the simulator tool to run the main framework code, and then use `roslaunch` to run the launch file. To see an example of how the robot should perform, you can launch `launch/problem4a_solution.launch`.

5 Tuning the Decision Process

By tweaking the rewards and the transition probabilities of the MDP, you will change the robot's behavior. In the reference solution provided to you, the transition probabilities are cautiously optimistic, meaning that if the robot senses where the ball appears to be, the probability that it is there will go up slightly. In this problem, you will try two different methods of tuning how the robot behaves in order to explore the effects of tuning on the performance of your MDP.

5.1 Optimistic Tuning (10 points)

The first tuning you will try is an optimistic tuning that leverages the robot's knowledge. For this tuning, the robot should assume that when it senses near the block that has the currently-highest estimated probability, it will find the ball under that block with very high probability.

Implementation Instructions. Create a service called `problem5a` of type `p5/ModelSenseAction` that takes in a State and returns a probability distribution over the blocks. All code for this problem should go in `src/problem5a.py`. All of the information you need is provided to you in the ModelSenseAction request, so you do not need to call any other services to solve this problem. You will also need to fill in the launch file `launch/problem5a.launch`. We have provided `launch/problem5a_solution.launch` for your reference.

5.2 Pessimistic Tuning (10 points)

The second tuning you will try is a pessimistic tuning that encourages the robot to explore. For this tuning, the robot should assume that when it performs a sensing action near a block, the ball will not be located under that block, thereby increasing the probability estimates for all other blocks.

Implementation Instructions. Create a service called `problem5b` of type `p5/ModelSenseAction` that takes in a State and returns a probability distribution over the blocks. All code for this problem should go in `src/problem5b.py`. All of the information you need is provided to you in the ModelSenseAction request, so you do not need to call any other services to solve this problem. You will also need to fill in the launch file `launch/problem5b.launch`. We have provided `launch/problem5b_solution.launch` for your reference.