Udacity CS373: Programming a Robotic Car
Final Exam

Welcome to the Final Exam!

There are 19 questions and 1 programming exercise. Some of them are very easy, some of them are really hard and challenging, and these will be marked as such. If you don't want to be challenged too much, you can skip the hard questions.
We hope that you have enjoyed the course and wish you good luck in the final exam!

Please enter answers with a precision of 3 decimal places.

Final-1: Probabilities-1

You have a loaded coin that comes up **heads** with a **0.6** probability.
What is the probability that it will come up tails?

Final-2: Probabilities-2

You have a loaded coin that comes up **heads** with a **0.6** probability. Now you flip this coin twice.
What is the probability that it will never come up heads in these two flips?

Final-3: Probabilities-3

You have a loaded coin that comes up **heads** with a **0.6** probability. Now you add a fair coin (probability heads = 0.5). You now have two coins - one loaded and one fair. You select a coin at random with a probability of 0.5, you flip it twice and it comes up heads twice.
What is the probability that you picked the loaded coin? Obviously you have to use Bayes rule here.

Final-4: Markov Localization-1

Suppose you have a robot that lives in the following world. This holds true for the next 4 Markov Localizations questions.

Initially the robot has no clue where it is.
What initial probabilities will you assign for these 4 grid cells?

Final-5: Markov Localization-2

Now the robot senses RED, but has a measurement error probability of 0.2.
Please update the probabilities after sensing!

Final-6: Markov Localization-3

It now moves up north, but the world is not cyclic. If the robot hits wall, it will just not move.
Update the probabilities!

Final-7: Markov Localization-4

It now senses RED again.
Update the probabilities!

Final-8: 1-D Kalman filters-1

You are given two Gaussians -- Gaussian 1 and Gaussian 2.
Calculate for the two Gaussians the result of applying the Bayes rule. You can think of the Gaussian 1 as the prior and the Gaussian 2 as the measurement probability.

Gaussian 1		Gaussian 2		Result	
μ	σ^2	μ	?σ^2	μ	σ^2
1.0	**1.0**	**1.0**	**1.0**		

Final-9: 1-D Kalman filters-2

Now do the same for these Gaussian values:

Gaussian 1		Gaussian 2		Result	
μ	σ^2	μ	σ^2	μ	σ^2
1.0	**1.0**	**5.0**	**1.0**		

Final-10: 1-D Kalman filters-3

Now do the same for these Gaussian values:

Gaussian 1		Gaussian 2		Result	
μ	σ^2	μ	σ^2	μ	σ^2
1.0	**1.0**	**5.0**	**4.0**		

Final-11: 1-D Kalman filters-4

This question also will be about 1-D Kalman filters, but replacing the Bayesian measurement update with the **prediction step of Kalman filters**. Think of Gaussian 1 as the probability before motion and Gaussian 2 as the probability that characterizes the additive effect of robot motion.
While the values of Gaussians are same as before, the results are different.

Gaussian 1		Gaussian 2		Result	
μ	σ^2	μ	σ^2	μ	σ^2
1.0	**1.0**	**1.0**	**1.0**		

Final-12: 1-D Kalman filters-5

Now do the same for these Gaussian values:

Gaussian 1		Gaussian 2		Result	
μ	σ^2	μ	σ^2	μ	σ^2
1.0	**1.0**	**5.0**	**1.0**		

Final-13: 1-D Kalman filters-6

Now do the same for these Gaussian values:

Gaussian 1		Gaussian 2		Result	
μ	σ^2	μ	σ^2	μ	σ^2
1.0	**1.0**	**5.0**	**4.0**		

Final-14: Particle Filters-1 (challenge)

Suppose you have a robot that lives in the following non-cyclic world.

This is a challenging question. Suppose in a global localization you have 12 particles.
What is the probability that all cells have at least one particle, for that initial uniform sample?
Don't spend too much time on this question if you don't want to, this is actually beyond what you have learned at this class, but if you think about it for a while you may be able to figure it out.

Final-15: Particle Filters-2

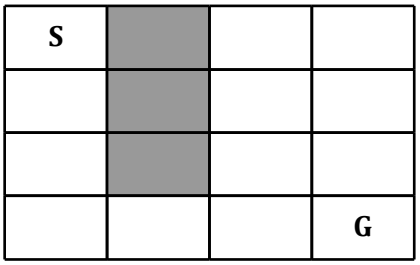
Assume each cell has 3 particles.
What is the normalized sum of all weights in each cell, if you observe RED and assume 0.2 measurement probability?

Final-16: Particle Filters-3

Assume each cell has 3 particles. You don't have to worry about resampling at all. The robot moves north, but the world is not cyclic (if the robot hits a wall, it will not move at all).
What is the number of particles in each cell after the motion? Assume we use each particle once, and assume motion is noise-free.

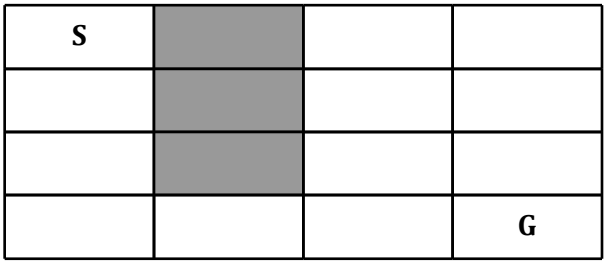
Final-17: A* planning-1

Given that the heuristic is admissible, **mark all the cells that A-star MUST expand** before it reaches the goal.
Assume only motions up/down/left/right, no diagonal motion.



Final-18: A* planning-2 (challenge)

This is a very challenging question.
Which of the cells will NEVER be expanded, as long as the heuristic is admissible.
Assume only motions up/down/left/right, no diagonal motion.



Final-19: PID control

Lets say you are driving your car in a circular trajectory and you are experiencing difficulties in staying on or near the trajectory, either by veering off, or going in circles outside, or controlling like crazy, which, if you know how to drive, has happened to you either as part of a homework assignment or even in a real car.

The question is -- what modification COULD make it easier to stay near the reference trajectory. The question is not what would guarantee it, but things that sometimes make it easier to stay near the reference trajectory.

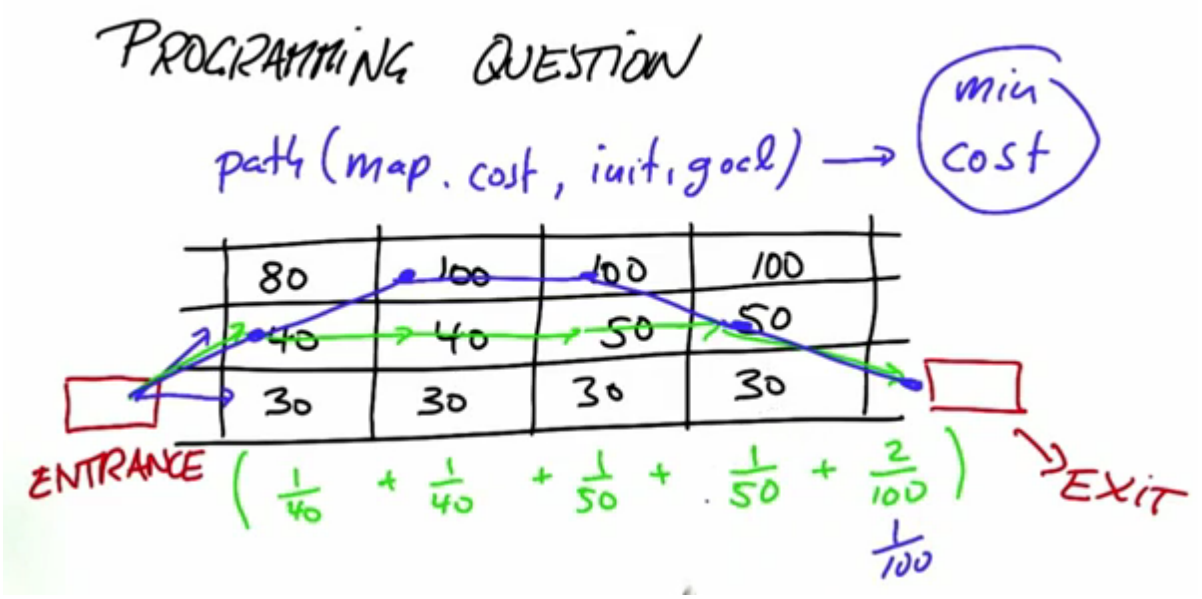
Please check any or all of these choices that could help:

- increase length of car
- decrease length of car
- increase maximum steering angle
- increase speed of the car
- remove P term from your controller
- increase diameter of the circle
- none of above

Final-20: Programming exercise

The programming exercise is about simulating a self-driving car on a multi-line highway.
Your car starts on the left side (ENTRANCE), after entering the highway, and you wish to go to the right side(EXIT), as fast as possible. One solution would be to just stay on the right lane, but that's often suboptimal, because it often moves slower than the left lanes.

The way the task is specified is by cutting the road into segments, and providing you, or your software, with the speeds in these segments. For example the speeds could be 30 on the right lane, between 40 and 50 in the second lane and, let's go crazy -- 100 miles per hour on the left lane.
The robot can do one of two actions - it can go forward or it can do a lane change. On the left lane there is only one lane change direction possible, but on the middle lane it can go either left or right.
A path will look like a sequence of lane shifts and straight motions that eventually lead to a goal.
The cost of this path is sum of (1/speed of the line segment), and to make this a little bit more challenging, also assume that there is a lane change cost. So every time you change lanes, you are going to add a small cost for the lane change (for this example it will be 1/100). The total cost for this path will be the sum specified under the road drawing.



Your task is to write a function **`plan`** that takes as arguments road, lane change cost, initial and goal positions and finds the optimal path, in this case that would be marked with the blue line. The function should return the total **`cost`** of the optimal path.

Important! You can only shift lanes one at a time, you can not do double lane shifts. And you always have to to move forward (towards the goal), you can not go back, and that wouldn't be optimal anyways.

In the code you will find a specification of what the road looks like. You will find plenty of user instructions. Read those! You will find parameters into the **`plan`** function -- **`road, lane_change_cost, init, goal`**. And there is plenty of test code at the end. There is a function that lets you build random roads, so that you can test as much as you want. There is a solution checker that you can use to test your solution against known roads and several test cases to test if your program is correct.

You have to build a planner and you can use whatever method you like. You have to implement the **`plan`** function and that function has to return at the end just one value - the **`cost`** that the optimal path obtains, and you can check whether your solution is correct, by checking against the provided test cases and the true cost value that is given there.