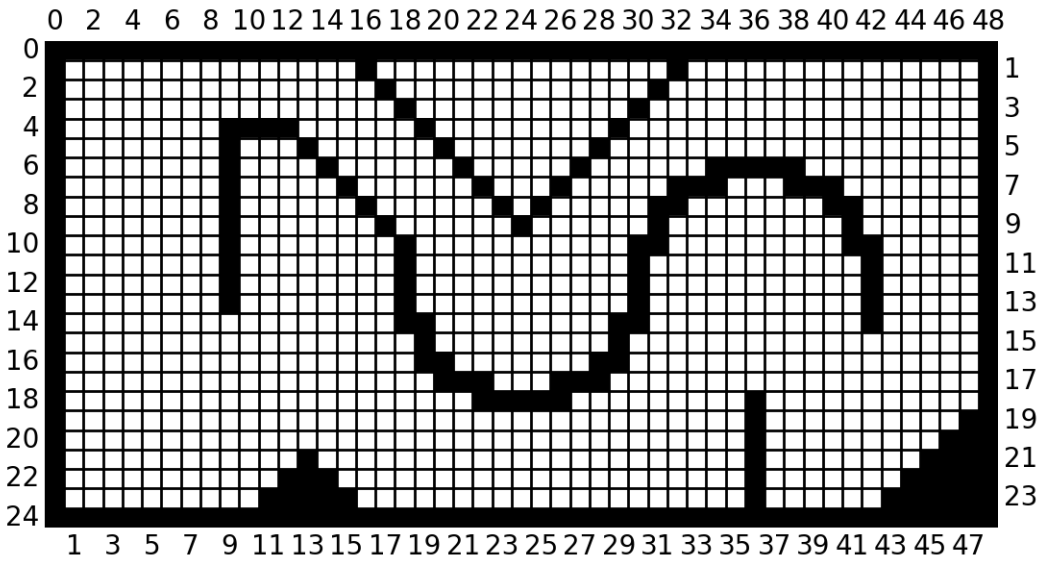
Homework #5
due Wednesday 2/7/24 11:59pm

This week, I would like to explore the localization. We'll ask you to program a grid-based approach and manually compute a Kalman filter challenge.

Please take a look (and work with) the demo code files `hw5_localize.py` and `hw5_utilities.py`. You will need to edit/complete the former (search for "FIXME"). The utilities should be good to go, though please feel free to dissect/change.

Problem 1-4 (Grid-Based Localization) - 68 total points:

Consider the challenge of localizing a robot in the following room/on the following grid



The robot is 2D and has four possible moves: one grid element up/down/left/right one grid element. It can not move diagonally. And we are *not* considering orientation, so the robot does not turn.

We have *four* proximity sensors, pointed up, down, left, right. Each sensor reports `True` or `False` to indicate it is looking at a wall. To start, assume the sensor range is exactly one grid element. That is, each sensor will report `True` next to a wall, `False` otherwise.

Because we are in a perfect 2D grid-world, we can also assume the four sensors are independent. So, for the purpose of localization, we can update our belief one sensor at a time, four times per command cycle.

In the following, we'll (1) get the code running, (2) add some randomness (noise) to the sensors, (3) add some randomness to the commands (not always executing faithfully), and (4) kidnap the robot.

Problem 1 (Basic Deterministic System) - 24 points:

Certainly, as a first step, we need to fully implement the algorithm. You'll notice that I used keyboard inputs for the motion commands (up/down/left/right) and hence to trigger each update. Please feel free to adjust as you best like.

(a) Getting the basic code running

You can run the code by commenting out the three "FIXME"s. It will *not* localize as is, but you can see the interface running.

The variables:

`prd`	Prediction (pre sensor update)	Prob. of robot being at element
`bel`	Belief (post sensor update)	Prob. of robot being at element
`prior`	Pre sensor update	Prob. of robot being at element
`post`	Post sensor update	Prob. of robot being at element
`probXYZ`	Pre-computed sensor XYZ probability	Prob. of sensor firing if at element
`probCmd`	Probability of robot actually moving according to command (single number)	
`probProximal`	List of probabilities of sensor firing at range of 1, 2, 3, ... elements from wall (probability vs distance)	

You will need to write/edit three functions (each marks with "FIXME"):

1) Predicting a new belief based on an issued command. Ignore any randomness for now (ignore/keep `probCmd=1.0`). So each command moves one grid space. But keep in mind that moving into a wall won't occur. The robot will instead simply stay where it is.

   The new belief should implicitly still add to 100% if summed over all elements - the robot has to be somewhere.

2) Updating the belief based on the sensor readings - once for each of the four directions. Again the resulting belief should add to 100%, though this time via explicit normalization.

3) Pre-computing a sensor probability grid (map). I.e. for each grid element, what is the probability that the sensor would trigger in that grid element?

   Ignore the sensor noise/randomness (ignore/keep `probProximal=[1.0]`), assuming the sensor has a range of exactly 1 space and is 100% reliable. So the sensor always reports `True` next to a wall, `False` elsewhere.

You'll notice the main loop calls these three functions. I left the default settings to start the robot in a random location. Play with the up/down/left/right commands to watch the prediction and beliefs update.

For example, when moving up, the bottom row should turn into a zero probability (shown as yellow) - the robot can't be there any more. Also when you come next to a wall, the belief should be that you are next to a wall (somewhere). Zero probability (exactly) is color-coded yellow. Nonzero probability is color-coded increasing from light to dark, from white to pink to purple to blue.

Please submit the code for these three functions. Also, please submit a screen shot of the sensor's probability map for the up and right sensors.

(b) Demoing the basic code

With the working code, initialize the robot to start at (`row=12, col=26`). Though the algorithm has no knowledge thereof.

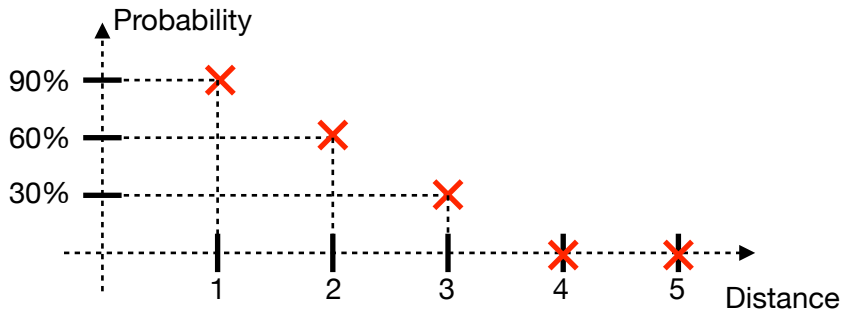
Move up three times (decreasing row numbers). Please submit a screenshot of the resulting belief grid.

What minimal set of *additional* actions should/can you take to completely localize the robot, i.e. have a single grid element show a belief of 100% while the other elements drop to 0%? Please submit a screenshot at the final location too.

Problem 2 (Noisy Extended Sensor) - 12 points:

In reality we'd want sensors to have a greater range. But they also exhibit noise. For proximity sensors, they may not trigger because the objects are glass or not appropriately reflective.

Assume each proximity sensor returns `True` with the following probability versus range.



Update the robot initialization call to include these probabilities. Also update your pre-computation of the sensor probability maps. Please again submit a screenshot of the maps for the up and right sensors.

And, as in Problem 1(b), initialize the robot to (`row=12, col=26`) and repeat the same extended movement sequence. Please submit a screenshot thereof. Why does this not completely converge to a single cell?

Problem 3 (Uncertain Command Execution) - 18 points:

In reality, robots don't always move exactly as commanded. Wheels can slip, the wheel radius might be slightly unknown, they might be slightly misaligned, or even hair or debris can get caught in the transmissions. These issues may lead to robots not moving perfectly.

Assume the robot only advances one space with 80% probability.

(a) Without Updating the Algorithm

Tell the robot initialization to model the robot as such, keeping the above sensor noise. But for the moment, don't tell your algorithm (don't "fix" your code). I would expect issues to arise - the algorithm will be overconfident.

Initialize the robot to (`row=15, col=47`) and command 12 movements to the left, followed by 6 movements down. Your visualization should show both the belief and the actual robot location - I would expect these not to be in agreement.

Please submit a screen shot after the 18 commands.

For fun: can you move the robot until the entire belief is zero, which clearly should never happen, i.e. the algorithm is very, very off?

(b) Fixing the Algorithm for Command Uncertainty

Please fix you algorithm by updating the prediction step, i.e. the predicted believe (probability map). Consider that the command is only executed with 80% probability. Repeat the movement from part (d) and submit the result.

For fun: can you command movements that lead to a complete convergence of the belief to a single grid element? It is possible to reach such a state, though additional moves will dilute the belief again. Unlike Problem 1, this is a unusual situation in general.

Problem 4 (Kidnapped Robots) - 14 points:

Finally, let's be a little mean to the robot, "teleporting" it to a random location and some random time. Can the algorithm re-localize?

(a) Without any Updates to the Algorithm

Tell the robot initialization to allow kidnapping, keeping the sensor noise and command uncertainty as above. Without adjusting the algorithm, I would expect the localization to get lost after the kidnapping.

Initialize the robot to (`row=7, col=12`), where the localization should be fairly good. Command 10-20 movements, allowing the belief to reduce to a pretty small area around the robot. At sometime after 10 movements, the robot will teleport.

You may get lucky and the robot teleports into a grid space that allows the algorithm to re-localize. If so, try again. Please submit a screen shot after the robot has teleported into a "yellow" grid showing zero probability - clearly inconsistent.

For fun: can you move the robot until the entire belief is zero, which clearly should never happen, i.e. the algorithm is very, very off?

(b) Fixing the Algorithm for Possible Kidnapping

Please fix. We'll leave the details up to you. Re-run from (`row=7, col=12`) and show the screen shot after the robot has teleported.

How many additional movements before you return to having a pretty good idea of the robot's location?

Problem 5 (Kalman Filter) - 28 points:

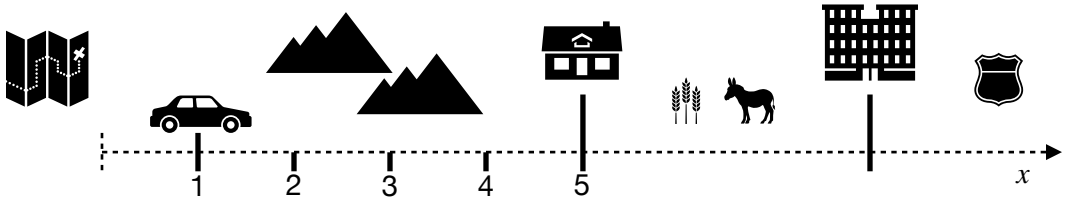
Assume you have decided to build an autonomous car in your spare time. Fortunately, your friend has implemented the lane keeping and you can leverage the built-in smart cruise control. So your challenge is to locate the 1D position of the vehicle along the road. Rather than coding this, let's compute this "by hand". I.e. you can simply calculate the answers using the necessary formulae.

Imagine you are driving to visit your grandmother who lives along a country road up in the hills. You start the final leg from a rest stop, which the map shows at kilometer-marker 1. But the map resolution is limited to approximately 100m, so you estimate your initial position is $\hat{x}_0 = 1{,}000$m with a *standard deviation* of $\sigma = 100$m.

The cruise control instantly accelerates you to your target speed of 37.5mph which works out to be 1000m every minute. To keep your samples and noise independent (nice move!) you decide to update your estimate once every minute. The cruise control does a decent job, but with the hilly conditions, it doesn't hold the speed perfectly and may not cover exactly 1000m every minute. So you expect a standard deviation of 30m, at the end of every minute.

At the end of each minute, you also try to take a GPS measurement. These are pretty accurate (the manufacturer claims standard deviation of 10m), but again the hills may interfere. That is, you don't always get the measurement.

You know your grandmother's house is near kilometer-marker 5, so you expect to get there in 4 minutes.



(a) What is your best estimate and *variance* of the vehicle position at t=0min, t=1min, t=2min, t=3min, and t=4min? In particular, assume you obtain the following measurements:

t=0min: The signal is blocked, no measurement.

t=1min: The signal is blocked, no measurement.

t=2min: GPS reports you at 3,100m.

t=3min: GPS reports you at 4,050m.

t=4min: The signal is blocked, no measurement.

To be redundant, please report both the *prior* and *posterior* values for both estimate and covariance, as well as the update gain for all five times.

(b) Your grandmother calls while you are still en route to tell you that she is staying at your cousin's apartment, a few kilometers farther down the road. So you keep driving. The hills end, turning into flat farm land, so that (i) the cruise control does a much better job, delivering a standard deviation of only 3m every minute. And (ii) the GPS has much better reception, ensuring a measurement every minute.

Assume the trip is long enough to reach a steady state condition. What is the best variance (immediately posterior to an update) and the worst variance (immediately prior to an update) that you expect to see?

Problem 6 (Time Spent) - 4 points:

As always, approximately how much time did you spend on this homework? Any particular bottlenecks? And any late submission information?