Review Probability

Before Sebastian teaches you about localizing a robot, in the next couple concepts, we'll review how to represent uncertainty in robot motion and sensors. The next couple sections will include multiple videos and quizzes to check your understanding of probability and probability distributions. These are meant to prepare you for this lesson, so make sure to read this material carefully to become familiar with terminology and mathematical representations of uncertainty!

Probability Review

Before learning more about uncertainty in robot localization, let's review how to mathematically represent uncertainty using probability!

A classic example of an event with some certainty associated with it is a coin flip. A typical coin has two sides: heads and tails. Without me telling you anything else, what chance do you think a coin like this has of flipping and landing heads up?

Heads and tails of a coin.

If you flip a coin with two sides (heads and tails), what is the probability, P(H), that the coin will land heads up? (1= 100%, 0 = 0% chance)

P(H) = 0.3

○

P(H) = 0.5

○

P(H) = 0.75

○

P(H) = 1.0

SOLUTION:P(H) = 0.5

When you flip a coin and it comes up *heads*, do you think it's more likely that the NEXT flip will come up *tails*? That is, does one flip affect the probability that another flip will be heads or tails?

[✓ Yes]

○

No

SOLUTION:

Formal Definition of Probability

The probability of an event, X, occurring is `P(X)`. The value of P(X) must fall in a range of 0 to 1.

- `0 <= P(X) <= 1`

An event, X, can have multiple outcomes which we might call X1, X2, .. and so on; the probabilities for all outcomes of X must add up to one. For example, say there are two possible outcomes, X1 and X2:

- If `P(X1) = 0.2` then `P(X2) = 0.8` because all possible outcomes must sum to 1.

Terminology

Independent Events

Events like coin flips are known as **independent events**; this means that the probability of a single flip does not affect the probability of another flip; P(H) will be 0.5 for each fair coin flip. When flipping a coin multiple times, each flip is an independent event because one flip does not affect the probability that another flip will land heads up or tails up.

Dependent Events

When two events are said to be *dependent*, the probability of one event occurring influences the likelihood of the other event. For example, say you are more likely to go outside if it's sunny weather. If the probability of sunny weather is low on a given day, the probability of you going outside will decrease as well, so the probability of going outside is dependent on the probability of sunny weather.

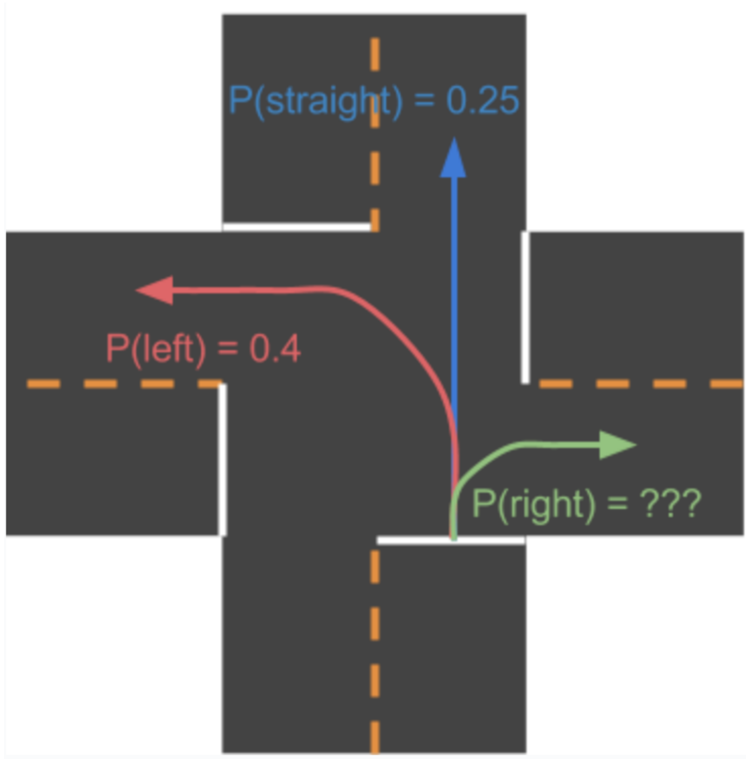
Joint Probability

The probability that two or more independent events will occur together (in the same time frame) is called a **joint probability**, and it is calculated by multiplying the probabilities of each independent event together. For example, the probability that you will flip a coin and it will lands heads up two times in a row can be calculated as follows:

- The probability of a coin flipping heads up, `P(H) = 0.5`
- The joint probability of two events (a coin landing heads up) happening in a row, is the probability of the first event times the probability of the second event: `P(H)*P(H) = (0.5)*(0.5) = 0.25`

Quantifying Certainty (and Uncertainty)

When we talk about being certain that a robot is at a certain location (x, y), moving a certain direction, or sensing a certain environment, we can quantify that certainty using probabilistic quantities. Sensor measurements and movement all have some uncertainty associated with them (ex. a speedometer that reads 50mph may be off by a few mph, depending on whether a car is moving up or down hill).



QUESTION:

At a particular intersection, cars can either:

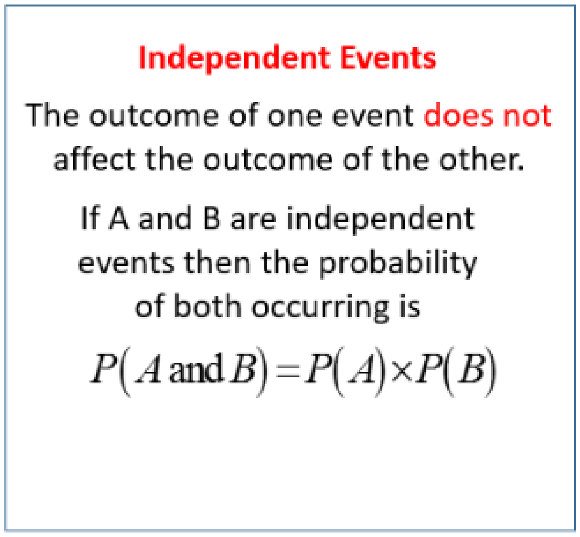
1. Turn left
2. Go straight
3. Turn right

Cars tend to turn left with a probability of 0.4 and go straight with a probability of 0.25. What is the probability that a car turns right?

Dependent Events

In this lesson, we will learn how to find the probability of dependent events. We will also learn the difference between the probability of dependent events and the probability of independent events.

Related Topics: More Lessons on Probability

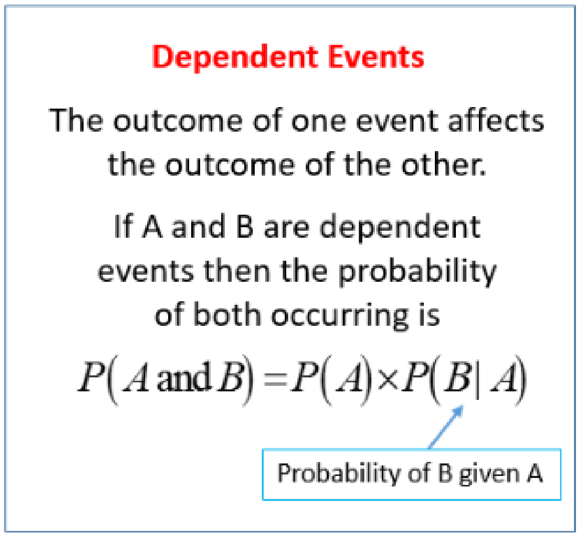


Independent Events

The outcome of one event does not affect the outcome of the other.

If A and B are independent events then the probability of both occurring is

$$P(A \text{ and } B) = P(A) \times P(B)$$



Dependent Events

The outcome of one event affects the outcome of the other.

If A and B are dependent events then the probability of both occurring is

$$P(A \text{ and } B) = P(A) \times P(B|A)$$

Probability of B given A

The following table gives the formulas for the probability of independent and dependent events. Scroll down the page for more examples and solutions.

Events are **dependent** if the outcome of one event affects the outcome of another. For example, if you draw two colored balls from a bag and the first ball is not replaced before you draw the second ball then the outcome of the second draw will be affected by the outcome of the first draw.

If *A* and *B* are dependent events, then the probability of *A* happening **AND** the probability of *B* happening, given *A*, is P(*A*) × P(*B* after *A*).

P(*A* and *B*) = P(*A*) × P(*B* after *A*)

P(*B* after *A*) can also be written as P(*B* | *A*)

then P(*A* and *B*) = P(*A*) × P(*B* | *A*)

Example:

A purse contains four \$5 bills, five \$10 bills and three \$20 bills. Two bills are selected without the first selection being replaced. Find P(\$5, then \$5)

Solution:

There are four \$5 bills.

There are a total of twelve bills.

P(\$5) = $\frac{4}{12}$

The result of the first draw affected the probability of the second draw.

There are three \$5 bills left.

There are a total of eleven bills left.

P(\$5 after \$5) = $\frac{3}{11}$

P(\$5, then \$5) = P(\$5) · P(\$5 after \$5) = $\frac{4}{12} \times \frac{3}{11} = \frac{1}{11}$

The probability of drawing a \$5 bill and then a \$5 bill is $\frac{1}{11}$

Example:

A bag contains 6 red, 5 blue and 4 yellow marbles. Two marbles are drawn, but the first marble drawn is not replaced.

a) Find P(red, then blue)

b) Find P(blue, then blue)

Solution:

a) There are 6 red marbles.

There are a total of 15 marbles.

P(red) $= \frac{6}{15}$

The result of the first draw affected the probability of the second draw.

There are 5 blue marbles.

There are a total of 14 marbles left.

P(blue after red) $= \frac{5}{14}$

P(red, then blue) = P(red) · P(blue after red) $= \frac{6}{15} \times \frac{5}{14} = \frac{1}{7}$

The probability of drawing a red marble and then a blue marble is $\frac{1}{7}$

b) There are 5 blue marbles.

There are a total of 15 marbles.

P(blue) $= \frac{5}{15}$

The result of the first draw affected the probability of the second draw.

There are 4 blue marbles left.

There are a total of 14 marbles left.

P(blue after blue) $= \frac{4}{14}$

P(blue, then blue) = P(blue) · P(blue after blue) $= \frac{5}{15} \times \frac{4}{14} = \frac{2}{21}$

The probability of drawing a red marble and then a blue marble is $\frac{2}{21}$

Examples of calculating the probability of dependent events

Example:

We have a box with 10 red marbles and 10 blue marbles. Find P(drawing two blue marbles).

- Show Step-by-step Solutions

Probability of Dependent Events

Example:

A club of 9 people wants to choose a board of 3 officers: President, Vice-President and Secretary. Assuming that the officers are chosen at random, what is the probability that the officers are Marsha for President, Sabita for Vice-President and Robert for Secretary?

- Show Step-by-step Solutions

Probability Dependent Events

How do you calculate probability of two dependent events?

Example:

A box contains 3 pens, 2 markers and 1 highlighter. Tara selects one item at random and does not return it to the box. She then selects a second item at random. What is the probability that Tara selects one pen and then one marker?

- Show Step-by-step Solutions

Example:

Andrea has 8 blue socks and 4 red socks in her drawer. She chooses one sock at random and puts it on. She then chooses another sock without looking. Find the probability of the following event P(red, then red).

- Show Step-by-step Solutions

Compare Dependent and Independent Events

Statistics - Dependent and Independent Events
This lesson teaches the distinction between Independent and Dependent Events, and how to calculate the probability of each.
The probability of two events is independent if what happens in the first event does **not** affect the probability of the second event. P(A + B) = P(A) × P(B)

The probability of two events is dependent if what happens in the first event does affect the probability the second event. P(A + B) = P(A) × P(B after A)

Example 1: If I roll a pair of dice, what is the probability that both dice land on a 6? Are these dependent or independent events?

Example 2: There are 4 puppies; two are male and two are female. If you randomly pick two puppies, what is the probability that they will both be female? Are these Independent or Dependent Events?

Example 3: You randomly choose one of the letter cubes. Without replacing it, you now choose a 2nd letter cube and place it to the right of the 1st letter cube. Then you pick a 3rd letter cube, and place it to the right of the 2nd letter cube. What is the probability that the letter cubes now spell "BIT"?

Let's talk a bit more about why uncertainty is so important in the field of robotics and self-driving cars.

We know that measurements like the

1- Speed
2- Direction
3- And the location

of a car are challenging to measure and we can't measure them perfectly.

There's some uncertainty in each of these measurements.

We also know that many of these measurements affect one another.

For example, if we are uncertain about the location of a car, we can reduce that uncertainty by collecting data about the car's surroundings and its movement.

Self-driving cars measure all of these things from a car's speed, to the scenery and objects that surround it, with sensors.

[And though these sensor measurements aren't perfect, when the information they provide is combined]

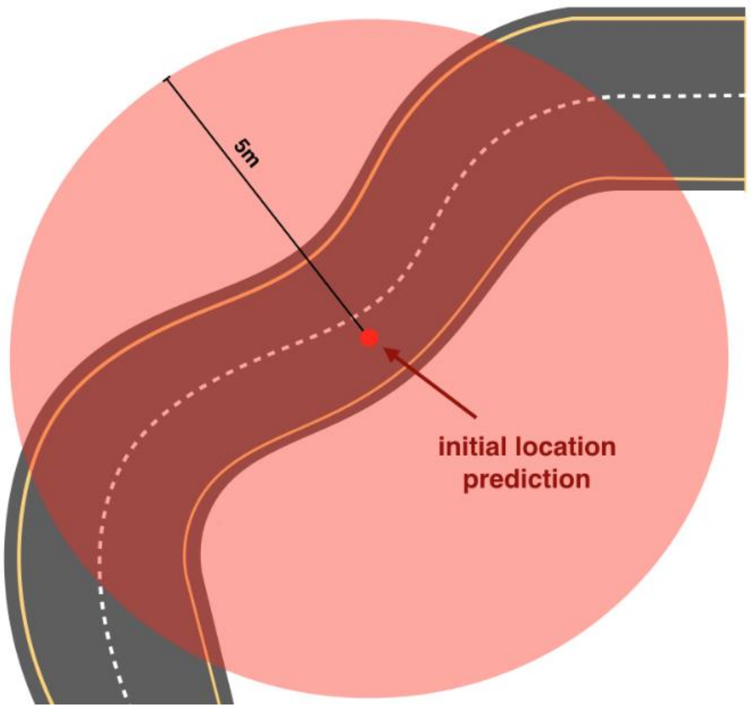
using conditional probability and something called Bayes rule, they can form a reliable representation of a car's position, its movement and its environment.

Let's take a look at Bayes Rule.

Bayes' Rule

Bayes' Rule is extremely important in robotics and it can be summarized in one sentence: given an initial prediction, if we gather additional data (data that our initial prediction depends on), we can improve that prediction!

Initial Scenario



Map of the road and the initial location prediction.

We know a little bit about the map of the road that a car is on (pictured above). We also have an initial GPS measurement; the GPS signal says the car is at the red dot. However, this GPS measurement is inaccurate up to about 5 meters. So, the vehicle could be located anywhere within a 5m radius circle around the dot.

Sensors

Then we gather data from the car's sensors. Self-driving cars mainly use three types of sensors to observe the world:

- **Camera**, which records video,
- **Lidar**, which is a light-based sensor, and
- **Radar**, which uses radio waves.

All of these sensors detect surrounding objects and scenery.

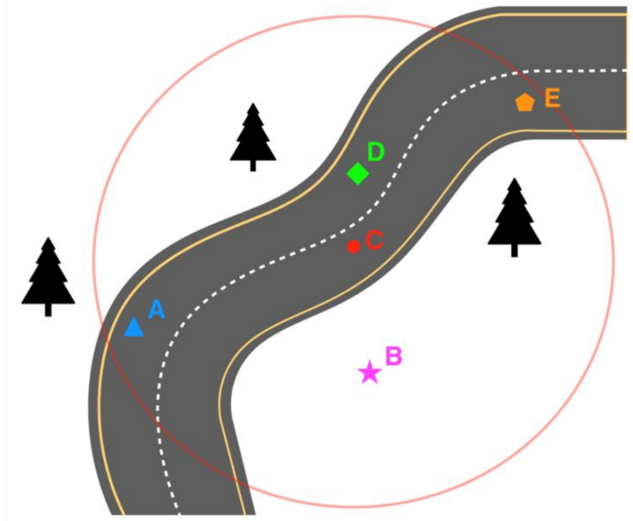
Autonomous cars also have lots of **internal sensors** that measure things like the speed and direction of the car's movement, the orientation of its wheels, and even the internal temperature of the car!

Sensor Measurements

Suppose that our sensors detect some details about the terrain and the way our car is moving, specifically:

- The car could be anywhere within the GPS 5m radius circle,
- The car is moving upwards on this road,
- There is a tree to the left of our car, and
- The car’s wheels are pointing to the right.

Knowing only these sensor measurements, examine the map below and answer the following quiz question.



Road map with additional sensor data

After considering the sensor measurements and the initial location prediction, which point on the above map is the best estimate for our car’s location? D

:

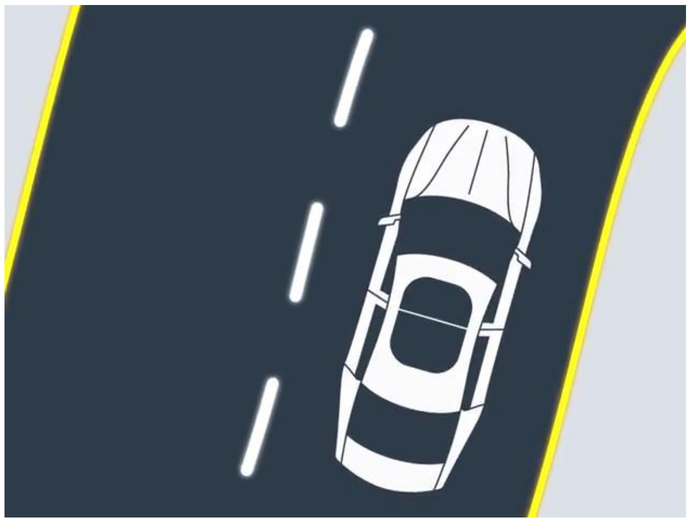
Lidar

Lidar is a method for measuring distances by illuminating the target with laser light and measuring the reflection with a sensor. Differences in laser return times and wavelengths can then be used to make digital 3-D representations of the target. It has terrestrial, airborne, and mobile applications.

Once we gather sensor data about the car's surroundings and its movement, we can then use this information to improve our initial location prediction.

For example, say we sense lane markers and specific terrain, and we say, hmm.

Actually, we know from previously collected data that if we sense landlines close to the sides of the car, the car is probably located in the center of the lane.



We also know that if we sense that our tires are pointing to the right, we're probably on a curved section of the road.

So this sensor data, combined with what we already know about the road and the car, gives us more **information about where our location is most likely** to be.

So using the sensor information, we can improve our initial prediction and better estimate our car's location.

Bayes rule gives us a mathematical way to correct our measurements, and let's us move from an uncertain prior belief to something more and more probable.

You'll see Bayes rule come up again and again in robotics. And in this lesson, you'll gain a greater understanding of Bayes rule.

You've seen how autonomous vehicles sense their environments and then use that information to **lower their uncertainty** and improve their predictions about **a car's movement and surroundings**.

But how do autonomous vehicles actually represent uncertainty in the real world? They use **probability and probability distributions**.

Probability distributions are a mathematical way to **represent uncertainty across all possible outcomes.**

For example, in the case of locating a vehicle, the possible outcomes are all the **places** where the vehicle could be.

When a vehicle first turns on the car has no idea where it is.

How do you think you could represent this initial uncertainty mathematically?

If you have no idea of the car located he probability distribution for its location will look completely flat.

The probability of it being anywhere say in San Francisco or in Tokyo will be the exact same.

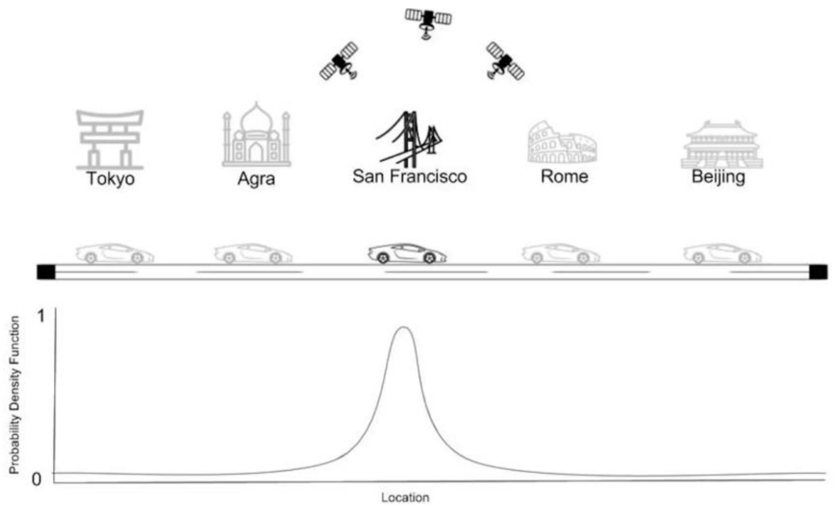
The probability that the car will be at any particular location will be a constant value that`s the same at every location.

So the probability distribution can be represented by a constant horizontal line on graph, that show the probability of all these outcomes.

But we as gather information, the distribution will change.

Say we get the A GPS sensor reading that tells us that we are a lot closer to San Franco than to Tokyo.

The Location probability goes up near the GPS measurement and the probability goes down in region far from the GPS measurement.



In even though the GPS signal isn`t perfect you have more information about the likely location of your car.

The vehicle is never 100 % sure of its location, yet through sensing the vehicle increases its correctly.

The Shape of probability distribution tells you the most likely locations and least likely of the vehicle.

Probability distributions are really useful ways to visualize and represent uncertainty, not just single vehicle localization but also tracking the locations of pedestrians, bicycles and other moving vehicles around a car.

These distributions are also used in representation uncertainty in sensor measurements.

Remember that autonomous vehicles are robots on wheels.

So everything from <u>sensing</u>, <u>measuring</u> and <u>moving</u> will involve some uncertainty.

You will learn draw and interpret probability distributions.

This will prepare you for vehicle localization and object tracking

What is a Probability Distribution?

Probability distributions allow you to represent the probability of an event using a mathematical equation. Like any mathematical equation:

- probability distributions **can be visualized** using a graph especially in **2-dimensional cases.**
- probability distributions **can be worked with using algebra, linear algebra and calculus**.

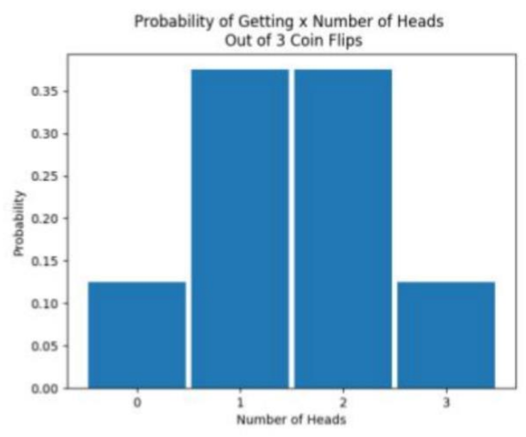
These distributions make it *much* easier to understand and summarize the probability of a system whether that system **be a coin flip experiment or the location of an autonomous vehicle.**

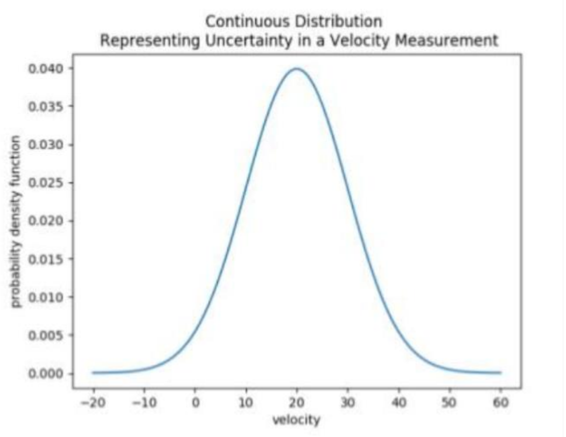
Types of Probability Distributions

Probability distributions are really helpful for understanding the probability of a system. Looking at the big pictures, there are two types of probability distributions:

- **Discrete probability distributions**
- **Continuous probability distributions**

Before we get into the details about what discrete and continuous mean, take a look at these two visualizations below. The first image shows a discrete probability distribution and the second a continuous probability distribution. What is similar and what is different about each visualization?





Discrete Distribution (left) and Continuous Distribution (right).

Based on the visualizations, which of the following are true about the discrete probability distribution versus the continuous probability distribution?

☑

The x-axis represents the main variable/event of interest for both visualizations.

☑

In the discrete visualization, the x-axis variable can only take on certain values such as 1, 2 or 3.

☐

In the discrete visualization, the x-axis variable can take on any value such as 3.4 or 0.5.

☑

In the continuous visualization, the x-axis variable can take on any real number value from -infinity to +infinity.

SOLUTION:

More terminology

- **Prior** - a prior probability distribution of an uncertain quantity, such as the location of a self-driving car on a road. This is the probability distribution that would express one's beliefs about the car's location **before ** some sensor measurements or other evidence is taken into account.

- **Posterior** - the probability distribution of an uncertain quantity, **after** some evidence (like sensor measurements) have been taken into account.

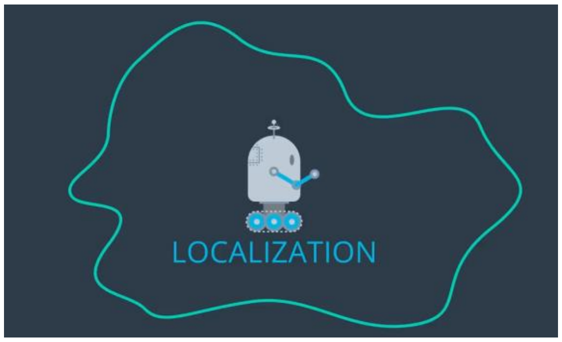
Localizations

The very first problem I'm trying to solve is called localization.

It involves a robot that's lost in space.

It could be a car, it could be a mobile robot.

So he has his environment and the poor robot has no clue where it is.

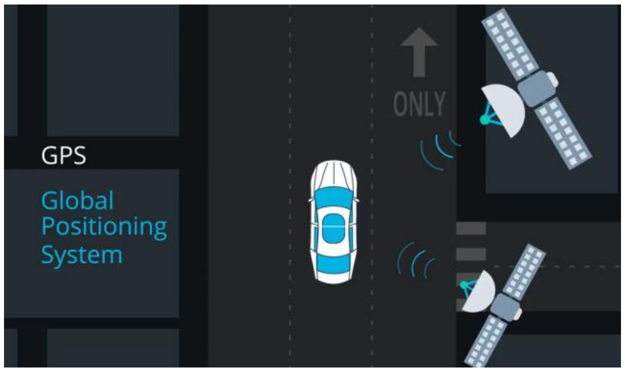


Similarly, we might have a car driving on a highway and this car would like to know where it is.

Is it inside the lane or is it crossing lane markers?

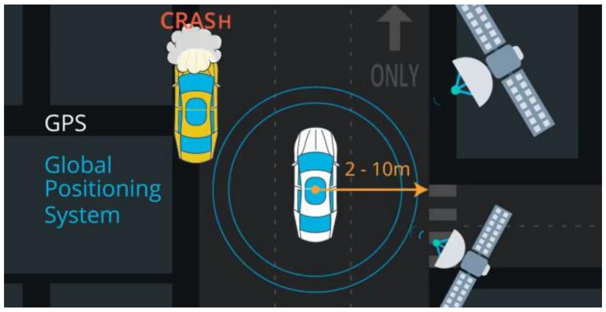
Now, the traditional way to solve this problem uses satellites, and these satellites emit signals that the car can perceive. That's known as GPS, short for Global Positioning System.

It's what you have in your dashboard if you have a car with GPS that shows you the maps and shows you where you are. Now unfortunately, the problem with GPS is it's really not very accurate.

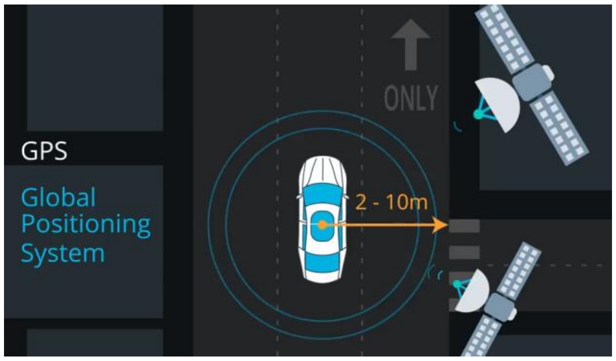


It's really common for a car to believe to be here but it has two, all the way up to 10 meters of error.

So you try to stay in the lane with 10 meters of error, you are far off and you're driving right over here and you **crash**.



So for our self-driving cars to be able to stay in lanes using **localization**, we need something like **2 to 10 centimeters** of error and then, we can drive with GPS in lanes.



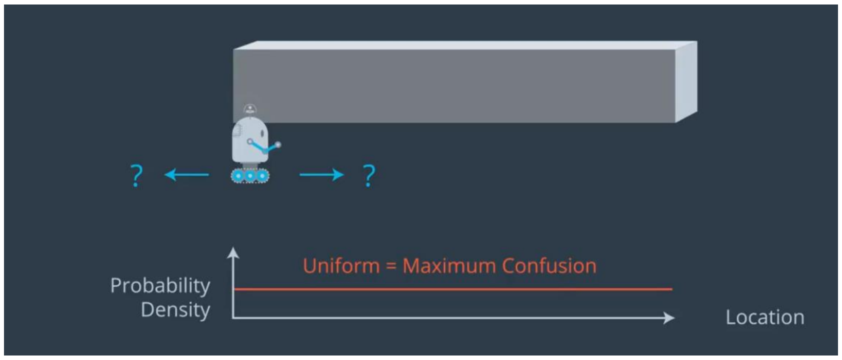
So the question is, how can we know where we are with 10-centimeter accuracy? That's the localization question.

Let me story in my world where a robot resides and let`s assume the robot has no clue where it is.

Then we will model this with a function,

I`m going to draw onto this diagram over here. Where the vertical axis is the probability for any location in this world and the horizontal axis corresponds to all the places in this one-dimension world.

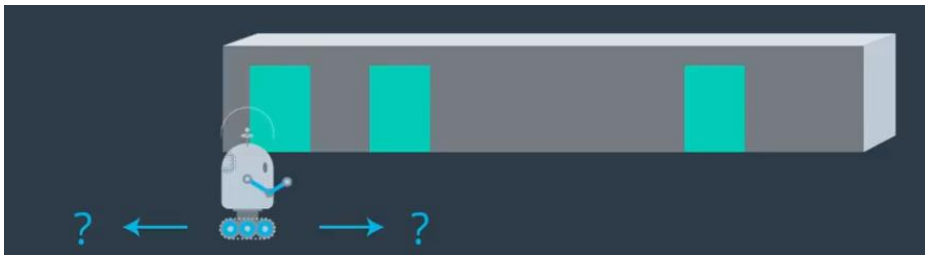
They way i`m going to model the robot`s current belief about where it might be, it`s confusion is by a uniform function that assigns equal weights to every possible place in the world.



There is a state of the maximum confusion.

Now to localize the bot has they have some distinctive features, let`s assume there`s three different landmarks in the world.

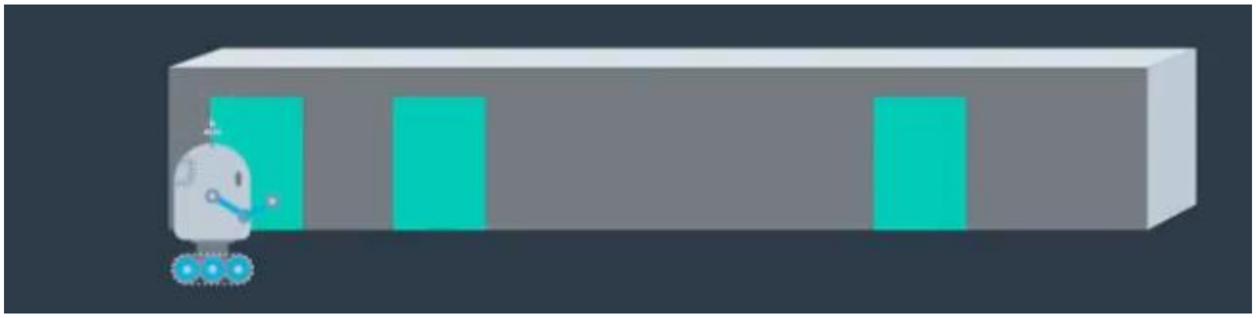
There`s a door over here, there `s a door over here and a third one way back here.



And for the sake of the argument let`s assume they are all look alike. من اجل الجدل دعونا نفتؤض انهم كلهم متشابهون

They are not distinguishable but you can distinguish the door from the none door area from the wall.

Now let`s see how the robot can localize itself  by assuming it senses,  and it senses that it`s standing right next to a door.



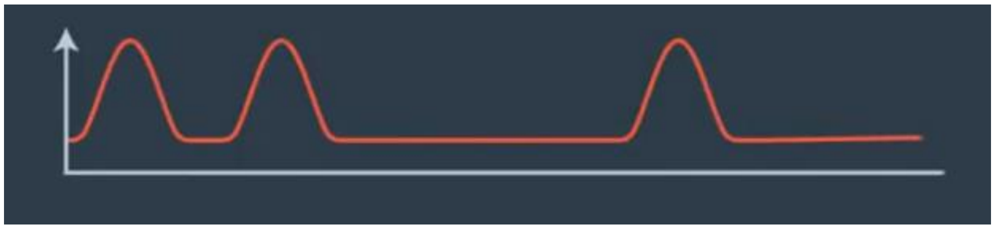
All it knowns now that is located likely next to a door

How will this affect our belief? (here is critical step for localization)

If you understand this step you understand localizations.

The measurements of a door transforms our belief function, defined over possible locations to a new function.

It looks very much like this.

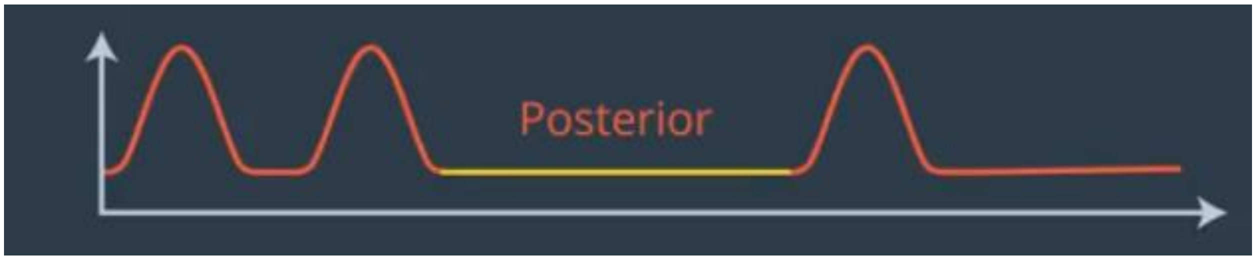


For the three locations adjacent to doors, we now have an increased belief of being there whereas all the other locations have a decreased belief.

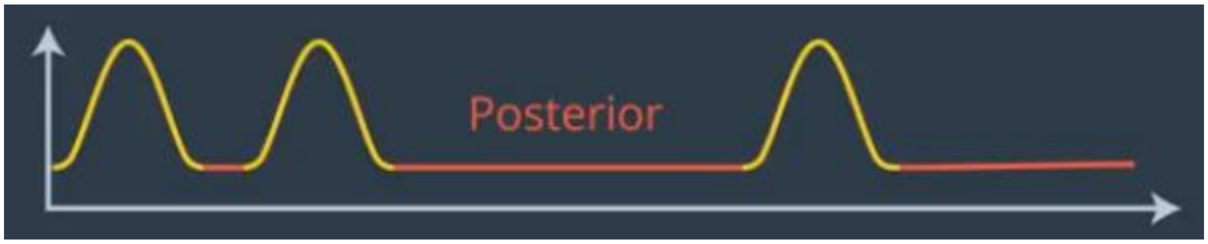
That is a probability distribution that assigns high probability for being next to door and it`s called the posterior belief with the word posterior means it`s after a measurement has been talking.

Now the key aspects of this beliefs is that we still don`t know where we are,  there` three possible locations in facts, it might be that the sensors were erroneous and we accidentally saw a door there is none, ربما تكون اجهزة الاستشعار خاطئة عن طريق الخطا بابا لا يوجد

There`s still residual probability of being at these places over here,



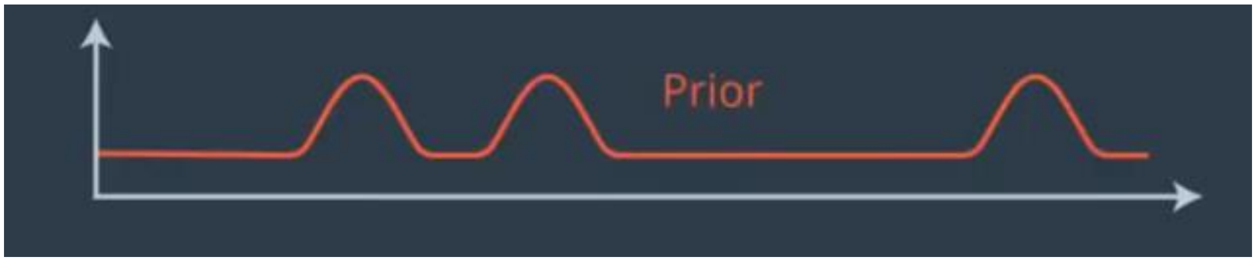
But there three bumps to get them really express our current best belief of where we are.



This representation is absolutely core to probability and to movable localization.

Now let`s assume the robot moves**. Say it moves to the right by a certain distance, then we can shift the belief according to the motion**.

Anyway this might look like is about like this.



This pump over here, made it to here.

This guy went over here and this guy over here.

Obviously this robot it knows it`s heading direction, it`s moving to the right in this example.

But it knows roughly how far it moved.

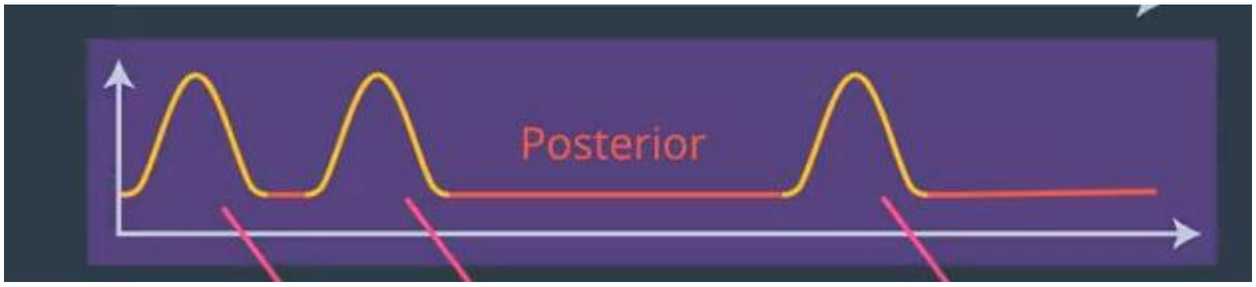
[now, robot motion is somewhat uncertain, we can never be certain where the robot moved.]

These things we will be a little bit flatter than guys over here.

<u>The process of moving those beliefs to right are essentially called a convolution, and let`s now assume the robot sense again and for the sake of the argument let`s assume it sees itself right next to a door</u>

Again, so the measurement is the same as before.

Now the most amazing thing happens. We end up multiplying our belief which is now prior to the second measurement, with a function looks very much like this one over here, which has a peek at each door and outcomes it believes it looks like the following.



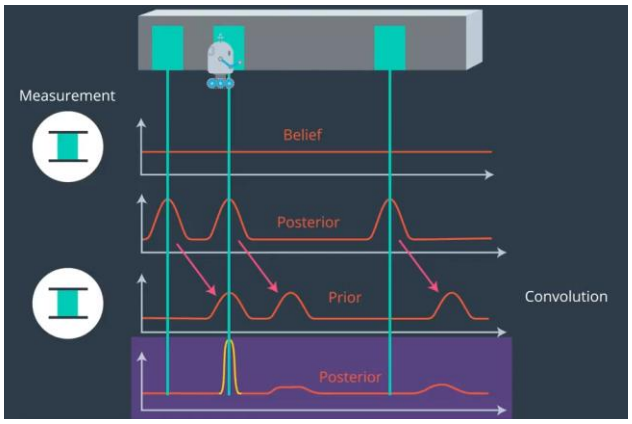
There`s a couple of minor bumps but the only really big bump is this one over here.

This one corresponds to this guy over here in the prior and it`s the only place in this prior that really corresponds to the measurements of a door whereas all the door places of doors have a low prior belief.

So as result this function is really interesting, it`s a distribution that focuses most of its weight under the correct hypothesis of the bot being on the second door and it provides very little belief to places for away from doors.

At this point our robot has localized itself.

If you understood this, you understand probability and you understand localization.



So let`s look the of this robot and measurement in his world with five different grid cells x_1 to x_5.

Let`s assume two of cells are coler the red whereas the other three are green.

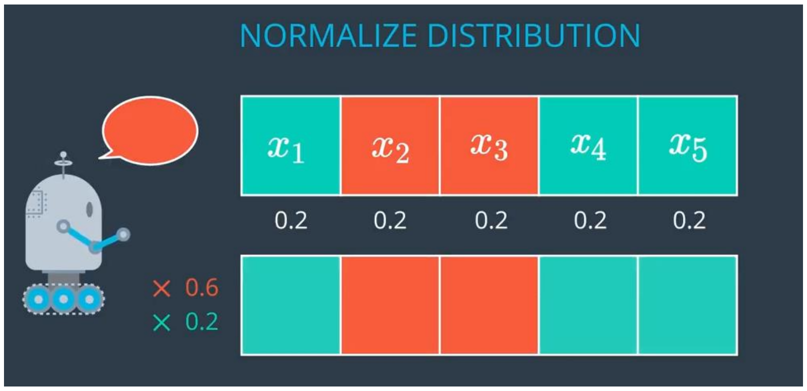
As we before we assign uniform probability to each cell of 0.2 and out robot is now allowed to sense, and what it sees is a red color.

How will this affect my belief over different places?

Obviously, the ones for x_2 and x_3 should go up and x_1 and x_3 and x_5 should go down.

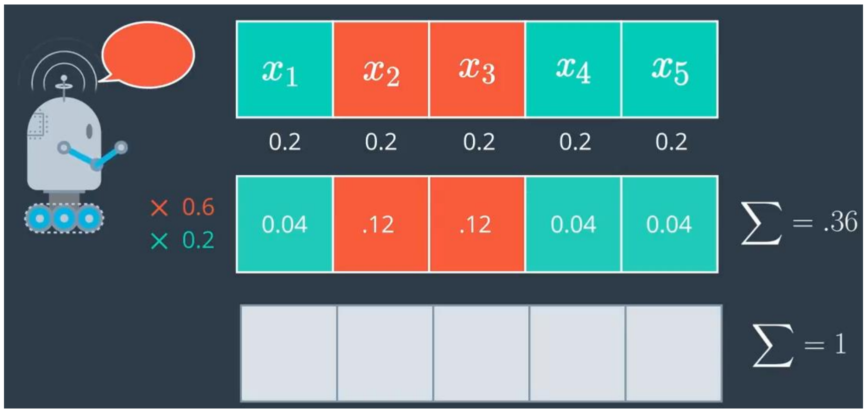
So, i`m going to tell you how to incorporate this measurement into our belief with a very simple rule a product into our belief with a very simple rule any cell where the color is correct, any of the red cells we multiply it with a relatively large number say 0.6 that feel small but as we will see later is actually a large number. Whereas all the green will be multiplied with 0.2 we look at the ratio of those that it seems about three times as likely to be in the red cell that it is to be in the green cell, because 0.6 is three times than 0.2

Let us do the multiplication. For each of five cell can you tell me what the result



We will now divide each of these number by 0.36 but differently we normalize.

Please fill the blank and check the sum of those truly adds up to 1.



or a non-normalized distribution `[0.04, 0.12, 0.12, 0.04, 0.04]`, what are the values for the **normalized** distribution?

○ `[0.04, 0.12, 0.12, 0.04, 0.04]`

○ `[0.1, 0.4, 0.4, 0.1, 0.1]`

○ `[0.233, 0.677, 0.677, 0.233, 0.233]`

[✓ `[0.111, 0.333, 0.333, 0.111, 0.111]`]

○ `[0.2, 0.2, 0.2, 0.2, 0.2]`

```
p = [0.2,0.2,0.2,0.2,0.2]

world = ['green' ,'red' , 'green' , 'green' , 'green']

Z = 'red'

pHit = 0.6

pMiss = 0.2


def sense(p,Z):
    q = []
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(p)):
        q[i] = q[i] / s
    return q
print(sense(p,Z))
```

Implement the absolute key of localization which is called the measurement update.

You can the test to see what amazing thing programmed

We had a uniform distribution over places, each place has a probability of 0.2.

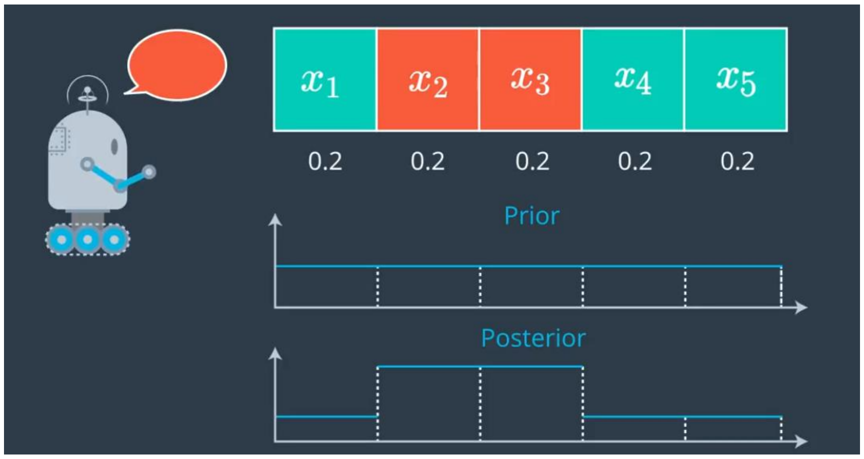
And then you we wrote a piece of code that use the measurement to turn this prior into a posterior, in which of the two red cells for the factor of three larger and the posterior of the green cells.

You have done exactly what I gave you intuitively in the beginning as the secret of localization.

You manipulate a probability distribution over places into a new one by incorporating the measurement.

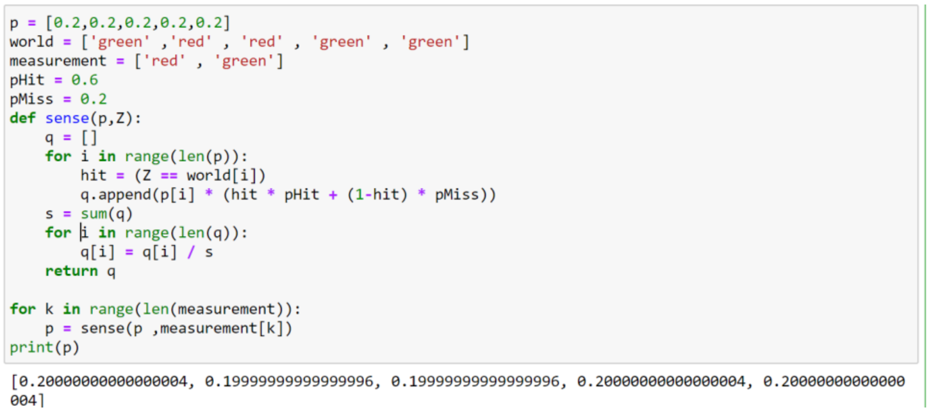
In fact, let`s go back to our code, and test in your code whether we get a good result when we replace our measurement red by green.

So, please type green into your measurement variable, and return your code to see if you get the correct result.



Change the code z to measurement.

```
p = [0.2,0.2,0.2,0.2,0.2]
world = ['green' ,'red' , 'red' , 'green' , 'green']
measurement = ['red' , 'green']
pHit = 0.6
pMiss = 0.2
def sense(p,Z):
    q = []
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
for k in range(len(measurement)):
    p = sense(p ,measurement[k])
print(p)
```

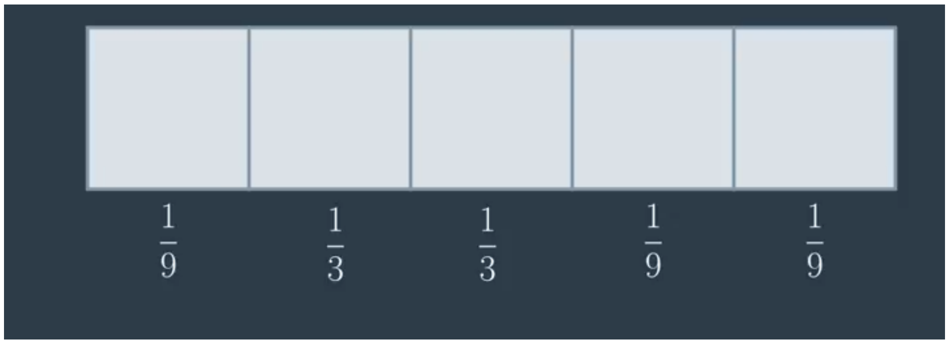


```
p = [0.2,0.2,0.2,0.2,0.2]
world = ['green' ,'red' , 'red' , 'green' , 'green']
measurement = ['red' , 'green']
pHit = 0.6
pMiss = 0.2
def sense(p,Z):
    q = []
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

for k in range(len(measurement)):
    p = sense(p ,measurement[k])
print(p)
```

```
[0.20000000000000004, 0.19999999999999996, 0.19999999999999996, 0.20000000000000004, 0.20000000000000
004]
```

I ’d like to talk about robot motion.

Suppose we have a distribution over these cells, such as this



And even though we don`t know where the robot is, the robot moves, and it moves to the right.

In fact, the way we are going to program is that we assume the world cyclic So if it drops off the rightmost cell, and finds itself on the leftmost cell.

Suppose we know for a fact the robot moved exactly one grid cell to the right including the cyclic motion.

Can you tell me for all these five values what the posterior probability is after that motion?

Define function probability distribution and motion number U

Where U is the number of grid cells that the robot is moving to the right or to the left and I want you to program a function that return the new distribution Q after move where

1- if U equal zero, Q is the same as p
2- if U equal one , all the values are cyclically shifted to the right by 1
3- if you equals 3 they are cyclically shifted to the right by 3.
4- If equal -1 they are cyclically shifted to the left.

I will use a very simple p, that has a 1 at the second position and zeros elsewhere.

Otherwise if we were to use the uniform p, we couldn’t even see the effect of the motion whether that programmed correctly or not

Probabilistic convolution

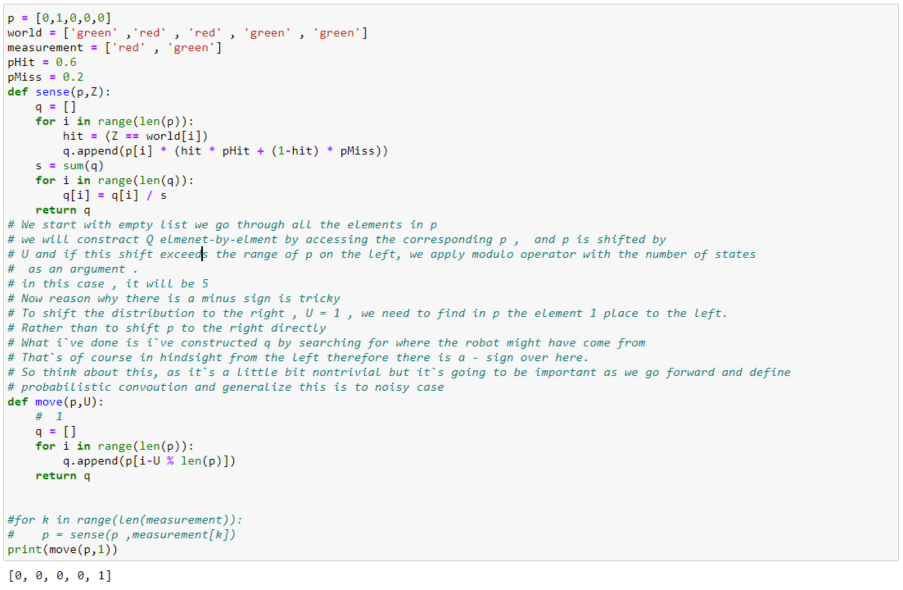
الالتفاف أو **الالتواء** أو **الطيّ** (بالإنجليزية: Convolution) هو مؤثر رياضي يستعمل في التحليل الدوالي ويقوم بتحوير دالة مخرجة من دالتين مدخلتين بحيث تكون قيمة الخرج عند أي لحظة زمنية متأثرة بكل قيم الدخل السابقة وتستخدم هذه الأداة الرياضية في عدة تطبيقات مثل:

The **convolution of probability distributions** arises in probability theory and statistics as the operation in terms of probability distributions that corresponds to the addition of independent random variables and, by extension, to forming linear combinations of random variables. The operation here is a special case of convolution in the context of probability distributions.

```
p = [0,1,0,0,0]

world = ['green' ,'red' , 'red' , 'green' , 'green']

measurement = ['red' , 'green']

pHit = 0.6

pMiss = 0.2

def sense(p,Z):

   q = []

   for i in range(len(p)):

      hit = (Z == world[i])

      q.append(p[i] * (hit * pHit + (1-hit) * pMiss))

   s = sum(q)

   for i in range(len(q)):

      q[i] = q[i] / s

   return q

# We start with empty list we go through all the elements in p

# we will constract Q elmenet-by-elment by accessing the corresponding p ,  and p is shifted by

# U and if this shift exceeds the range of p on the left, we apply modulo operator with the number of states

#  as an argument .
# in this case , it will be 5

# Now reason why there is a minus sign is tricky

# To shift the distribution to the right , U = 1 , we need to find in p the element 1 place to the left.

# Rather than to shift p to the right directly

# What i`ve done is i`ve constructed q by searching for where the robot might have come from

# That`s of course in hindsight from the left therefore there is a - sign over here.

# So think about this, as it`s a little bit nontrivial but it`s going to be important as we go forward and define

# probabilistic convoution and generalize this is to noisy case

def move(p,U):

    #  1

    q = []

    for i in range(len(p)):

        q.append(p[i-U % len(p)])

    return q

#for k in range(len(measurement)):

#    p = sense(p ,measurement[k])

print(move(p,1))
```

<u>Summary</u>:

• Motion that has a **<u>repeating pattern</u>** is called cyclic motion (or periodic motion).
• The time it takes to complete one full cycle, such as one rotation, is called the period.
• Frequency is the number of cycles per second.
• The amplitude indicates the size of the motion, for example, the stride length in a walk cycle.
• For rotation, the amplitude is the circumference, which depends on the radius of rotation.
• The speed of the motion increases with both amplitude and frequency.



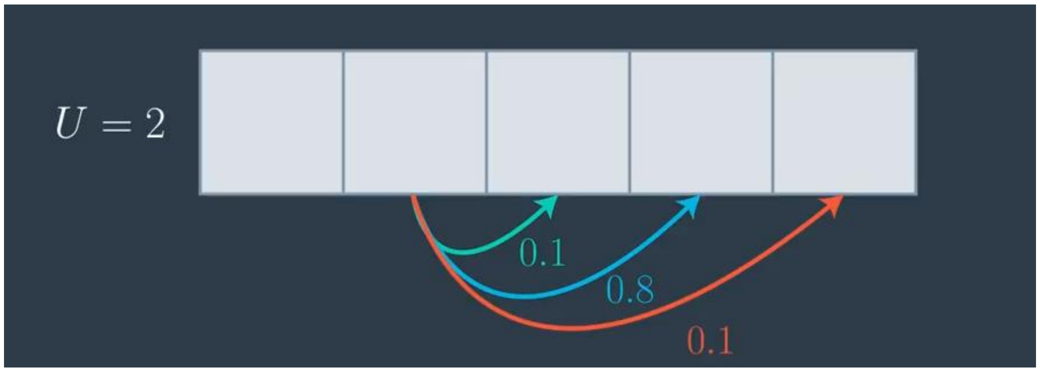
```
p = [0,1,0,0,0]
world = ['green' ,'red' , 'red' , 'green' , 'green']
measurement = ['red' , 'green']
pHit = 0.6
pMiss = 0.2
def sense(p,Z):
    q = []
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q
# We start with empty list we go through all the elements in p
# we will constract Q elmenet-by-elment by accessing the corresponding p ,  and p is shifted by
# U and if this shift exceeds the range of p on the left, we apply modulo operator with the number of states
#  as an argument .
# in this case , it will be 5
# Now reason why there is a minus sign is tricky
# To shift the distribution to the right , U = 1 , we need to find in p the element 1 place to the left.
# Rather than to shift p to the right directly
# What i`ve done is i`ve constructed q by searching for where the robot might have come from
# That`s of course in hindsight from the left therefore there is a - sign over here.
# So think about this, as it`s a little bit nontrivial but it`s going to be important as we go forward and define
# probabilistic convoution and generalize this is to noisy case
def move(p,U):
    #  1
    q = []
    for i in range(len(p)):
        q.append(p[i-U % len(p)])
    return q


#for k in range(len(measurement)):
#    p = sense(p ,measurement[k])
print(move(p,1))
```

```
[0, 0, 0, 0, 1]
```

Inaccurate robot motion

We have five grid cells and let`s assume a robot executes its action with high probability correct, say 0.80/ 0.1 chance.

It find itself short of the intended action and get another 1.1 probability it finds itself overshooting its target.



You can define the same for other U values, say U equals one then with chance 0.8 and 0.1 it says in the same element in 0.1 it hops two elements ahead.

Now, this is model a inaccurate robot motion.

The robot attempts to go U grid cells but occasionally falls short of its goal or overshoots and that`s a more common case robots  as they move accrue uncertainty and it`s really important to model this because this is the primary reason why localization is hard because robots are not very accurate.

We now are going to look into this first, for the mathematical side.

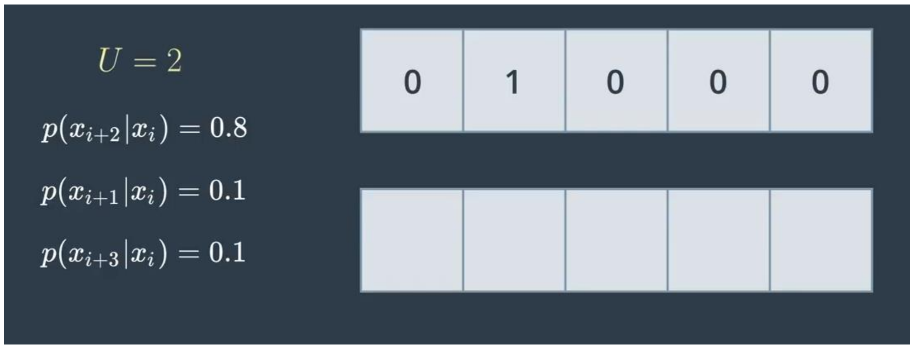
I will be giving you a prior distribution and we are going to be using the value U equals two and for the motion model that shifts the robots exactly two steps we believe there`s a 0.8 chance if we assign a 0.1 to the cases where the robot over undershoots by exactly one.

That`s kind of written by this formula over here where the two gets a 0.8 probability the one and three end up with 0.1 probability.

What I ask you now for the initial distribution that I’m writing up here,

Can you give me the distribution after the motion?

Inaccurate robot motion0020



Initialize belief of the robot

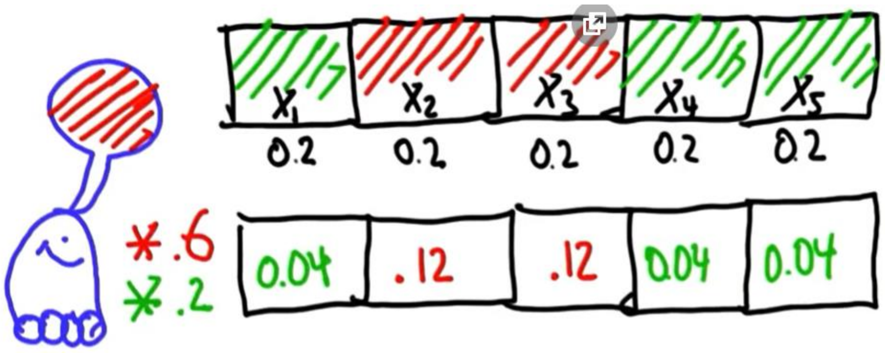
```
# initialize belief of the robot in the world over here
# Using Uniform Distrubtion
# n refer to Length
pro=[]
n=10
for i in range(n):
    pro.append(1./n)

print(pro)
```

```
[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]
```

Look at the measurement of robot in the world with 5 different grid cells.

When using measurement ? how will this affect my belief over different places ?

Incorporate this measurement into our belief with a very simple rule any cell with color correct



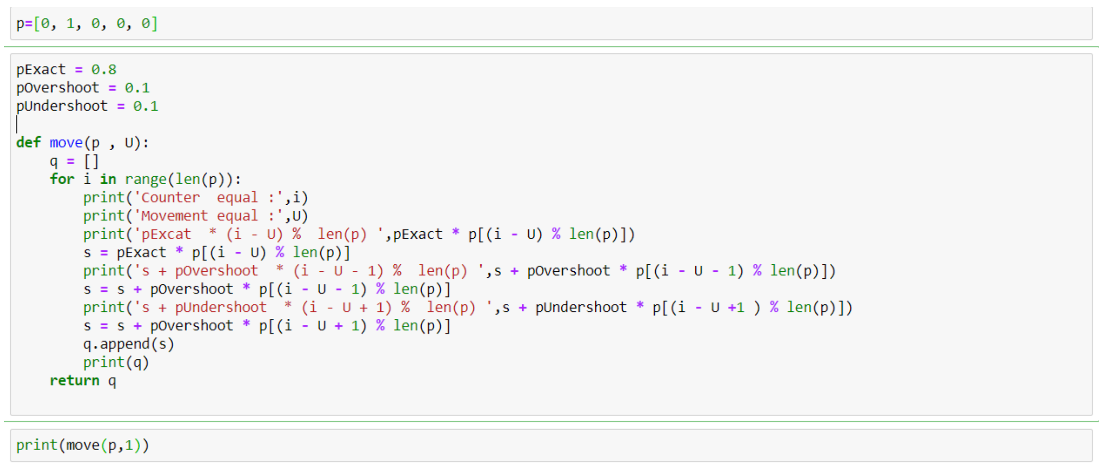
Whish is it isn`t a valid probability distribution.

The reason. why is probability distribution always have to add up to one which need to normalize.

The probability of each cell, I where I could range from 1 to 5, after we 've seen our measurement Z.

Formula is : P (X_i , Z) it is called Posterior distribution.

Nice work so far! You've learned a lot about combining probability with sensor measurements to localize yourself within an environment.

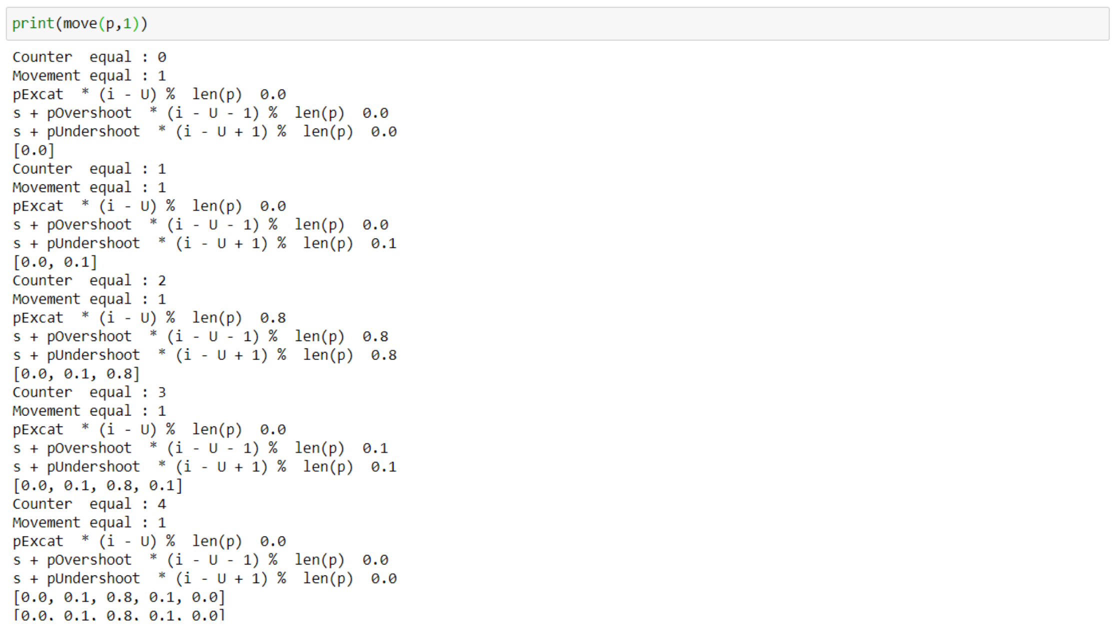


```
p=[0, 1, 0, 0, 0]
```

```
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def move(p , U):
    q = []
    for i in range(len(p)):
        print('Counter  equal :',i)
        print('Movement equal :',U)
        print('pExcat  * (i - U) %  len(p) ',pExact * p[(i - U) % len(p)])
        s = pExact * p[(i - U) % len(p)]
        print('s + pOvershoot  * (i - U - 1) %  len(p) ',s + pOvershoot * p[(i - U - 1) % len(p)])
        s = s + pOvershoot * p[(i - U - 1) % len(p)]
        print('s + pUndershoot  * (i - U + 1) %  len(p) ',s + pUndershoot * p[(i - U +1 ) % len(p)])
        s = s + pOvershoot * p[(i - U + 1) % len(p)]
        q.append(s)
        print(q)
    return q
```

```
print(move(p,1))
```

Trace code

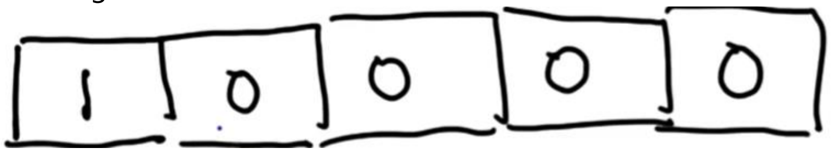


```
print(move(p,1))
```

```
Counter  equal : 0
Movement equal : 1
pExcat  * (i - U) %  len(p)  0.0
s + pOvershoot  * (i - U - 1) %  len(p)  0.0
s + pUndershoot  * (i - U + 1) %  len(p)  0.0
[0.0]
Counter  equal : 1
Movement equal : 1
pExcat  * (i - U) %  len(p)  0.0
s + pOvershoot  * (i - U - 1) %  len(p)  0.0
s + pUndershoot  * (i - U + 1) %  len(p)  0.1
[0.0, 0.1]
Counter  equal : 2
Movement equal : 1
pExcat  * (i - U) %  len(p)  0.8
s + pOvershoot  * (i - U - 1) %  len(p)  0.8
s + pUndershoot  * (i - U + 1) %  len(p)  0.8
[0.0, 0.1, 0.8]
Counter  equal : 3
Movement equal : 1
pExcat  * (i - U) %  len(p)  0.0
s + pOvershoot  * (i - U - 1) %  len(p)  0.1
s + pUndershoot  * (i - U + 1) %  len(p)  0.1
[0.0, 0.1, 0.8, 0.1]
Counter  equal : 4
Movement equal : 1
pExcat  * (i - U) %  len(p)  0.0
s + pOvershoot  * (i - U - 1) %  len(p)  0.0
s + pUndershoot  * (i - U + 1) %  len(p)  0.0
[0.0, 0.1, 0.8, 0.1, 0.0]
[0.0, 0.1, 0.8, 0.1, 0.0]
```

Here Q1 for u ?

1- Suppose we have a five grid cells as before with an initial distribution
2- Let`s assume U = 1 which means with 0.8 chance in each action we translation 1 to the right



3- With 0.1 chance we don`t move at all, and with 0.1 chance again skip and move 2 steps.
4- World is cyclic. So, every time I fall off on the right side, I find , myself back on the left side.

Q1 is suppose I run infinity many motion steps. Then I actually get what`s called '**limit distribution** 'what`s going to happen to my robot if it never sense but executes the action of going 2 to the right on our little cyclic environment forever. ?

What will be the so-called limit or stationary distribution be in the very end ?

1- U = 2
2- U = 2
3- U = 2
4- U = 2
5- .
6- .
7- .
8- .

As the robot becomes more and more uncertain about its position, what will happen to it's belief?

What is the MOST uncertain that the robot could be?

You might have guess it correctly.

It`s correctly. It`s the uniform distribution.

There`s an intuitive reasoning behind this.

Every time we move , we loss information.

That is, in the initial distribution we know exactly where we are.

One we step in we have a 0.8 chance, but the 0.8 will fall to something smaller as we move one – 0.64 and so on.

The distribution of the absolute least information is the uniform distribution. It`s no reference no preference whatsoever.

That is really the result of moving many, many ,many times.

There is a way to derive this mathematically and, I can prove a property that`s highly related , which is a balance property.

Say we take x4 and we would like to understand how x4 at some time sub t correspond to the previous time distribution over all these variables.

For this to be stationary , it has to be the same.

Put differently the probability of x4 must be the same as 0.8 p(x2) + 0.1 p(x1) + 0.1 p(x3)

This is exactly the same calculation we did before where we asked what`s the chance of being x4.

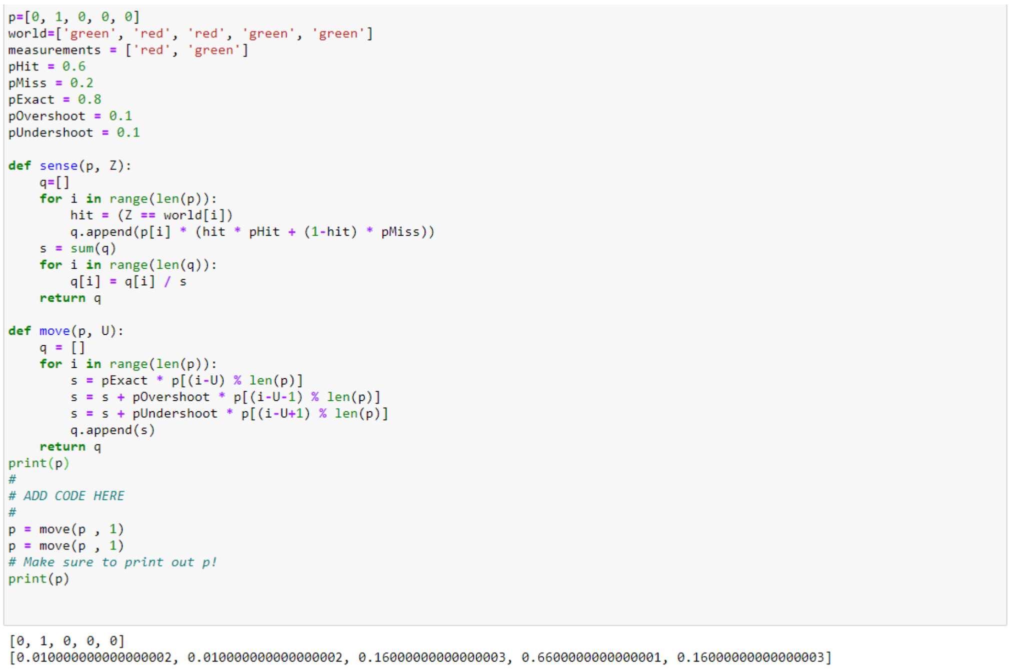
Well you might be coming from x2 , x1 or x3 , and there`s these probability are 0.8 , 0.1 and 0.1, they govern the likelihood you might have been coming from there.

Those together must hold true in the limit when things don`t move anymore.

You might think there are many different ways to solve this and the 0.2 is just one solution but it turns out 0.2 is the only solution

If you plug in 0.2 over here and 0.2 over here and 0.2 over here, you get x1 = 0.2 and that`s 0.2on the right side Clearly those over here meet the balance that is necessary to define a valid solution in the limit.

Move twice



```
p=[0, 1, 0, 0, 0]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q
print(p)
#
# ADD CODE HERE
#
p = move(p , 1)
p = move(p , 1)
# Make sure to print out p!
print(p)
```

```
[0, 1, 0, 0, 0]
[0.010000000000000002, 0.010000000000000002, 0.16000000000000003, 0.6600000000000001, 0.16000000000000003]
```

Move = 1000

For I in range(Move):

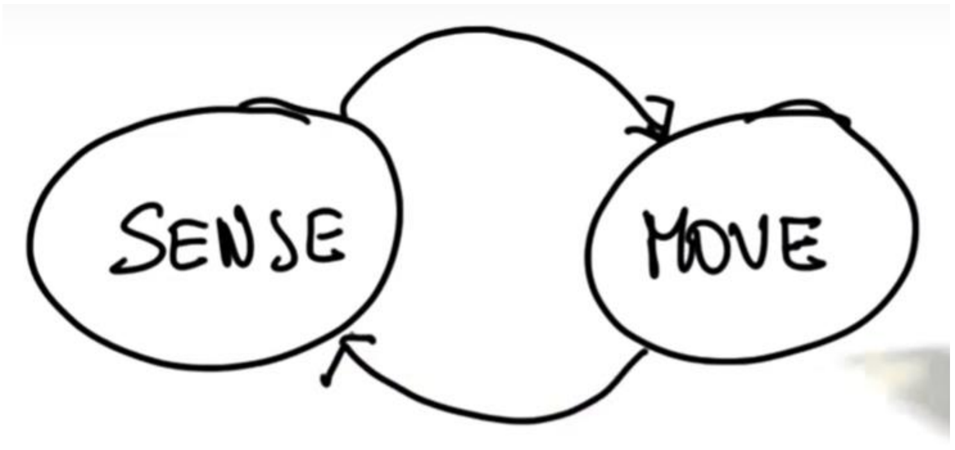
  P = move(p , i)

Sense and move

Review you are talking about

1- measurement updates
2- Motion
3- We called these routine 'Sense' and 'Move'

Now, localization is nothing else but the iteration of 'sense' and 'move'



There is initial belief that is tossed into this loop if you. If you sense first, if comes to the left side.

Then localization cycles through these move , sense and move , sense nd move , sense cycle

Every time the robot move it loses information as to where it is.

That`s because robot motion inaccurate.

Every time it senses it gains information.

That is manifest by the fact after motion., the probability distribution is a little bit flatter and a bit more spread out. And after sense it`s focused a little bit more.

In fact, as a foot note there is <u>a measure of information called 'entropy'</u>

<u>ENTROPY</u>

$$-\sum p(X_i) \log p(X_i)$$

as the expected log likelihood of the probability of each grid cell without going into detail, this is a measurement of the information the distribution has and it can be shown that the update step the motion make the entropy go down, and the measurement step makes it go up.

You really losing and gaining information.

In addition to this measurements we had before red and green

I`m going to give you 2 motion 1 and 1 which means the robot moves right and right again

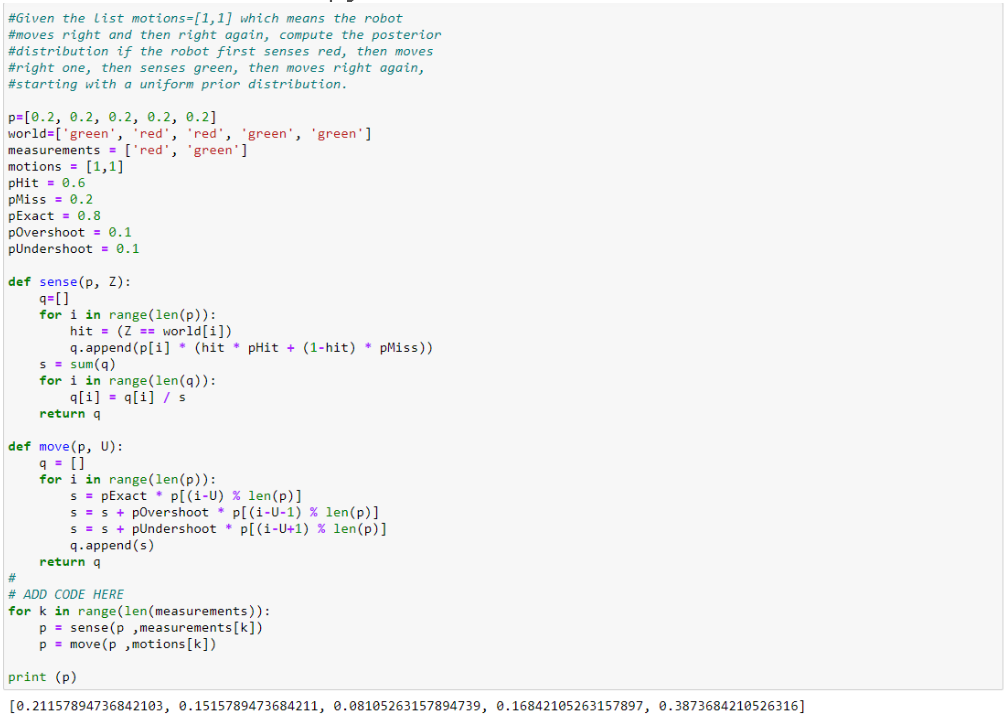
Can you compute the posterior distribution if the robot first sense red, then moves right by 1 , then senses green , then moves right again ?

Let`s start with uniform prior distribution

The video mentions that entropy will decrease after the motion update step and that entropy will increase after measurement step. What is meant is that entropy will decrease after the measurement update (sense) step and that entropy will increase after the movement step (move).

In general, entropy represents the amount of uncertainty in a system. Since the measurement update step decreases uncertainty, entropy will decrease. The movement step increases uncertainty, so entropy will increase after this step.

Let's look at our current example where the robot could be at one of five different positions. The maximum uncertainty occurs when all positions have equal probabilities [0.2, 0.2, 0.2, 0.2, 0.2][0.2,0.2,0.2,0.2,0.2]
Following the formula Entropy = \Sigma (-p \times log(p))$Entropy=\Sigma(-p\times log(p))$, we get -5 \times (.2)\times log(0.2) = 0.699$-5\times(.2)\times log(0.2)=0.699$.
Taking a measurement will decrease uncertainty and entropy. Let's say after taking a measurement, the probabilities become [0.05, 0.05, 0.05, 0.8, 0.05][0.05,0.05,0.05,0.8,0.05]. Now we have a more certain guess as to where the robot is located and our entropy has decreased to 0.338.



```
#Given the list motions=[1,1] which means the robot
#moves right and then right again, compute the posterior
#distribution if the robot first senses red, then moves
#right one, then senses green, then moves right again,
#starting with a uniform prior distribution.

p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'green']
motions = [1,1]
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q
#
# ADD CODE HERE
for k in range(len(measurements)):
    p = sense(p ,measurements[k])
    p = move(p ,motions[k])

print (p)
```

```
[0.21157894736842103, 0.1515789473684211, 0.08105263157894739, 0.16842105263157897, 0.3873684210526316]
```

The world has a green, a red , a red , a green and green field.

And robot saw red, followed by a right motion and green That suggests that ir probably started with the highest likelihood in grid cell number 3 which is the right most of the two red cells.

It saw red correctly it then moved to the right by 1.

It saw green correctly moved right again it now finds itself most likely in the right most cell.

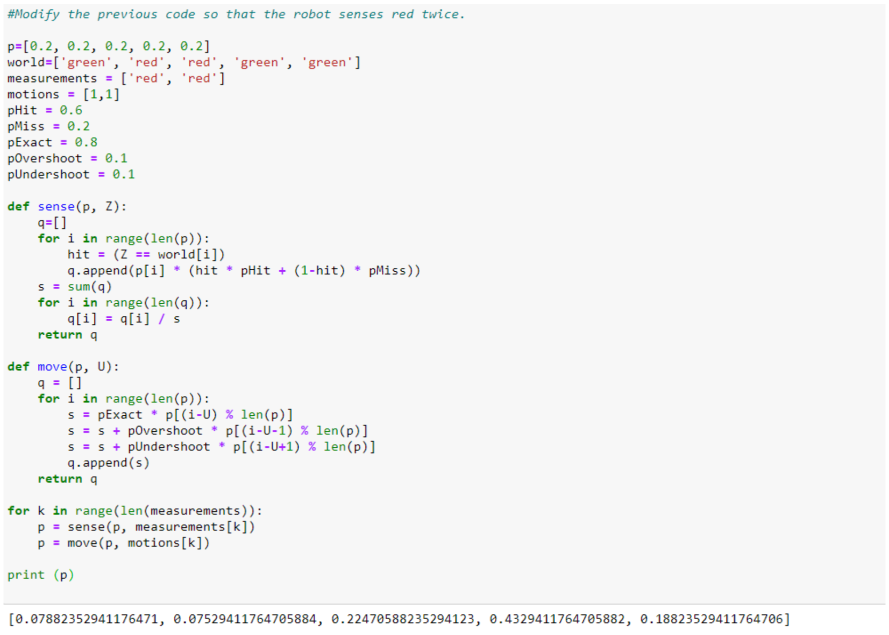
This is just looking at these values over here without any probabilistic math and any code limitation.

Let`s look at the output it`s very correctly 0.387368……..158 then it would most likely assign this position to the right-most cell اخر كانت هى اللcell

As should be given the sequence of observation over here.

Let`s assume the robot saw red twice.

It sense red it moves it sense red it moves again what is the most likely cell ?



```
#Modify the previous code so that the robot senses red twice.

p=[0.2, 0.2, 0.2, 0.2, 0.2]
world=['green', 'red', 'red', 'green', 'green']
measurements = ['red', 'red']
motions = [1,1]
pHit = 0.6
pMiss = 0.2
pExact = 0.8
pOvershoot = 0.1
pUndershoot = 0.1

def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q

for k in range(len(measurements)):
    p = sense(p, measurements[k])
    p = move(p, motions[k])

print (p)
```

```
[0.07882352941176471, 0.07529411764705884, 0.22470588235294123, 0.4329411764705882, 0.18823529411764706]
```

The most likely cell is four cell and make sense because the best match of red , red to the world is red over here.

And after sense the 2nd red , the robot still moved 1 to the right and finds itself in the 4th cell as show over here.

Now I want to celebrate with you code that just you wrote which is a piece of software that of implementation the essence of google`s  self driving car`s localization approach.

It`s absolutely crucial that the car knows exactly where it is relative to the map of its road.

While the road isn`t painted green and red the road has **lane markers** instead of those green and red cells over here. We plug in the color of the lane markings relative to the color of the pavement رصيف

It isn`t just one observation time step it`s ***an entire field of observation***, an entire camera image , but we can do the same with a camera image as long as you can correspond a camera in your model with a camera model is a measurement

These a piece of code not much more difficult than what you coded yourself is responsible for localizing the Google self-driving car.

You just implemented a major , major function that makes Google`s car drive itself

Now why on earth did it take Google that long to build a product that drives itself.

Well, the truth is the situation is a little more difficult.

Sometimes the road get paved over and changed and we are working on this but what you have implemented is the core of Google`s self-driving car localization idea.

We learned that localization maintains a function over all possible places where a road might be where each cell has an associated probability value.

Belief = probability

The measurement update function or sense is nothing else but a product in which take those probability values and multiply them up or down depending on the exact measurement.

Because the product might violate the fact that probabilities add up to 1 there was a product followed by normalization.

Motion was a convolution.

This word itself might sound cryptic , but what it really means is **for each possible location after the motion we reverse engineered the situation** and guessed where the world might have come from and then collected we added corresponding probabilities something as simple as multiplication and addition solves all of localizations and it the foundation for autonomous driving

Formal definition :

We define a probability function to be p(x), and it`s value that is bounded below and above by 0 and 1

$$0 \leq P(X) \leq 1$$

X can take a multiple value we had the case of 5 grid cell.

Suppose it can take 2 values there`s only grid cells x1 , x2

1- If the probability for x1 is 0.2 what would be the probability for x2 = 0.8

Let`s me ask a second question and I know that`s not particularly difficult.

What is p(x1) = 0

```
If x1 has zero probability, all of the probability must be with
                              x2.
```

For our world with 5 different grid cells, suppose we know that the first 4 of them have a 0.1 probability

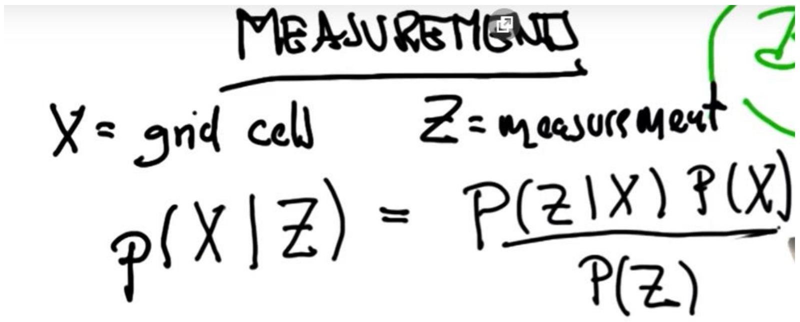
```
0.1 + 0.1 + 0.1 + 0.1 + 0.6 = 1
```

Let`s look into measurements and they will lead to something called Bayes Rule.

Bayes Rule it`s most fundamental consideration in probabilistic inference but the basic rule is really very simple .

Suppose X is my grid cell and Z is my measurement then the measurement update seeks to calculate a belief over my location after seeing the measurement.

It really easy to compute in our localizations example Now I'm going to make it a little bit more formal



MEASUREMENT

X = grid cell   Z = measurement

$$p(X \mid Z) = \frac{P(Z \mid X)\, P(X)}{P(Z)}$$

It what it does is it take my prior distribution P(x) and multiplies in the chances of seeing a red or green tile for every possible location and out comes if you just look at the denominator here.

The non-normalized posterior distribution.

Recognize this, this was our prior this was our measurements probabilities

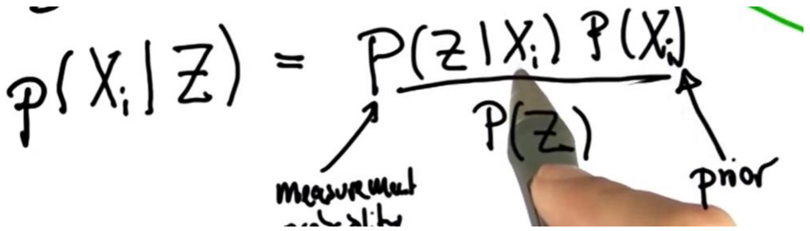
P(x) = Prior

P(x | z) = measurement probability

P(X | Z) if we do this all the grid cells , so we put a littile index I over here X_i

Then just product of the prior of the grid cell times the measurement probability, which was large if the measurement corresponded to the correct color and small if it corresponded a false color

That product gave us the non-normalized posterior distribution for the grid cell.



You programmed a product between the prior probability distribution and a number

The normalization is now the constant over here P(Z)

Technically , that is the probability of seeing a measurement devoid of any location information.

P(Z) this is function here that assigns to each grid cell a number and the P(Z) doesn`t have the grid cell as a index N matter what grid cell we consider , the P(z) is the same.

No matter what p(Z) is because the final posterior has to be a probability distribution by normalizing these non-normalized products over here.

We will exactly calculate p(Z)

Put Differently, P(Z) is sum over all I of just this product over here this make bayes rule very simple.

-it`s a product of our prior distribution with a measurement probability, which we know to be large if the color is correct and small otherwise.

$$p(Z) = \sum_i P(Z|X_i)\,P(X_i)$$

We do this and assign it is so called non-normalized probability which I will do bar of p then I will compute the normalize which I will call alpha is the sum of all these guys over here.

This is normalize my resulting probability will be 1 / alpha of the normalized probability.

X = grid cell $\quad$ Z = measurement

$$\bar{p}(X_i|Z) \leftarrow P(Z|X_i)\,P(X_i)$$

$$\alpha \leftarrow \sum_i \bar{p}(X_i|Z)$$

$$p(X_i|Z) \leftarrow \frac{1}{\alpha}\,\bar{p}(X_i|Z)$$

Bayes theorem reminder:

```
P(A|B) = P(B|A)*P(A) / P(B)
```

CANCER TEST

$p(C) = 0.001$

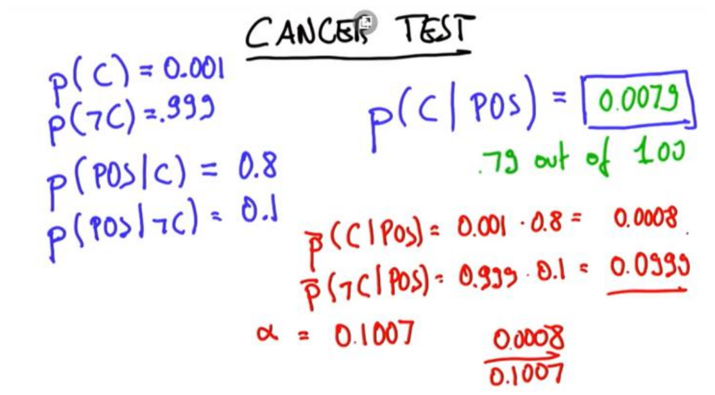
$p(\neg C) = .999$

$p(POS|C) = 0.8$

$p(POS|\neg C) = 0.1$

$$p(C|POS) = \boxed{8/1007}$$

Another way solution



Our normalizer is alpha is the sum of both of those which is 0.1007 so divided the non-normalized probability.

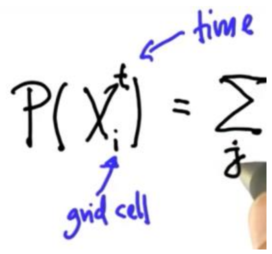
Let`s look at the motion, which will turn out to be something we will call total probability

You remember that we cared about grid cell ‘ X^t_i ’ and we asked what is the chance of being in X1 after the robot motion ?

To indicate the after and before let me add a time index . t is time

Index I which is the grid cell.

You might remember the way we computed this was by looking add all the cells the robot the robot could have come from on time step earlier indexed by here j.

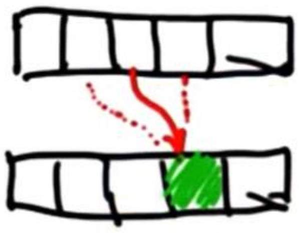


we looked at the prior probability of those prior probability of those grid cells t time t- 1 we multiply with the probability that our motion command would carry us form p (X_i | X_j)

$$P(X_i \mid X_j)$$

this is write as a condition distribution as follow there is exactly was our grid cell over here and we asked one time step later about specific grid cell over here.

We would combine 0.8 from over here 0.1 from over here and 0.1 from over here into the probability of this grid cell.

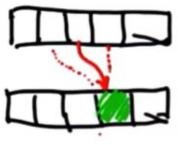


MOTION- TOTAL PROBABILITY

time

$$P(X_i^t) = \sum_j P(X_j^{t-1}) \cdot P(X_i | X_j)$$

grid cell



It is same formula here this is now Xi and the way we find the posterior probability for Xi is to go through all possible places from which we could have come , all different j'S look at the prior probabilities , multiply the probability that I translation from j to I given my motion command which in this case is go 1 to the right state side in probability term people often write it as follows.

$$P(A) = \sum_B P(A|B)\,P(B)$$

And you can see directly the correspondence of A as a place I of time and all difffernt B as the possible prior locations

That is often called the theorem of total probability

The operation of a weighted sum over other variable is often called convolution.

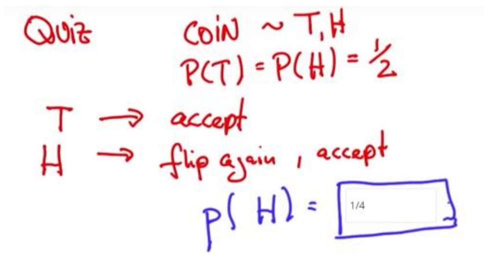
CONVOLUTION

$$\sum_B P(A|B)\,P(B)$$

Independence event

Coin = T ,H

P (H) * P (T) = 1 / 4



Quiz coin ~ T,H

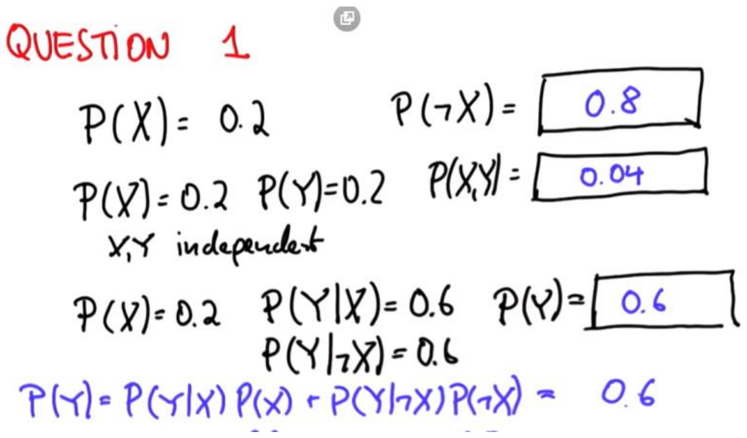
$P(T) = P(H) = \frac{1}{2}$

$T \rightarrow$ accept

$H \rightarrow$ flip again, accept

$p(H) = \boxed{1/4}$

Quiz



QUESTION 1

$P(X) = 0.2 \qquad P(\neg X) = \boxed{0.8}$

$P(X) = 0.2 \quad P(Y) = 0.2 \qquad P(X,Y) = \boxed{0.04}$

X,Y independent

$P(X) = 0.2 \quad P(Y|X) = 0.6 \qquad P(Y) = \boxed{0.6}$

$P(Y|\neg X) = 0.6$

$P(Y) = P(Y|X)P(X) + P(Y|\neg X)P(\neg X) = 0.6$

Quiz

3 rotation angle

1- Row
2- Pitch
3- Yaw

If you build localization system for fly robot higher dimensional state space

I wonder how the memory used will scaled for histogram localization based method

Does memory scale

1- Linearly
2- <u>Exponentially</u>
3- Quadratically
4- None of the above

In the number of state variable used localization ? so the number of state is three x and y and orientation.

If you were to look t a flying where you have x , y ,z , roll pitch and yaw you get six variable and I wonder how the memory use of the basic histogram localization scales.

Solution explain

Suppose we resolve each variable at a granularity of 20 different values

1- 20 for x
2- 20 for y
3- 20 for theta

Then joint table of all those will be $20^{\wedge N}$, where N is the number of state dimensions.

There is unfortunately no easy way around it The biggest disadvantage of the grid based localization method or the histogram method is that the scale of memory is exponential which mean it`s not applicable to even problems with 6 dimensions because you can`t allocate memory for 6 dimensions.

For the purpose of this homework assume that the robot can move only left, right, up, or down. It cannot move diagonally. Also, for this assignment, the robot will **never overshoot** its destination square; it will either make the movement or it will remain stationary.

Warning:

If you define any helper functions make sure they do not rely on globally defined variables and take all their state as parameters.

Reminder:

Reference 1D sense and move functions developed during the lesson:

```
def sense(p, Z):
    q=[]
    for i in range(len(p)):
        hit = (Z == world[i])
        q.append(p[i] * (hit * pHit + (1-hit) * pMiss))
    s = sum(q)
    for i in range(len(q)):
        q[i] = q[i] / s
    return q

def move(p, U):
    q = []
    for i in range(len(p)):
        s = pExact * p[(i-U) % len(p)]
        s = s + pOvershoot * p[(i-U-1) % len(p)]
        s = s + pUndershoot * p[(i-U+1) % len(p)]
        q.append(s)
    return q
```

Additional Test Cases

```
# test 1
colors = [['G', 'G', 'G'],
          ['G', 'R', 'G'],
          ['G', 'G', 'G']]
measurements = ['R']
motions = [[0,0]]
sensor_right = 1.0
p_move = 1.0
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.0, 0.0, 0.0],
     [0.0, 1.0, 0.0],
     [0.0, 0.0, 0.0]])

# test 2
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R']
motions = [[0,0]]
sensor_right = 1.0
p_move = 1.0
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.0, 0.0, 0.0],
     [0.0, 0.5, 0.5],
     [0.0, 0.0, 0.0]])

# test 3
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R']
motions = [[0,0]]
sensor_right = 0.8
p_move = 1.0
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.06666666666, 0.06666666666, 0.06666666666],
     [0.06666666666, 0.26666666666, 0.26666666666],
     [0.06666666666, 0.06666666666, 0.06666666666]])

# test 4
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
```

```
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 0.8
p_move = 1.0
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.03333333333, 0.03333333333, 0.03333333333],
     [0.13333333333, 0.13333333333, 0.53333333333],
     [0.03333333333, 0.03333333333, 0.03333333333]])

# test 5
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 1.0
p_move = 1.0
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.0, 0.0, 0.0],
     [0.0, 0.0, 1.0],
     [0.0, 0.0, 0.0]])

# test 6
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 0.8
p_move = 0.5
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.0289855072, 0.0289855072, 0.0289855072],
     [0.0724637681, 0.2898550724, 0.4637681159],
     [0.0289855072, 0.0289855072, 0.0289855072]])

# test 7
colors = [['G', 'G', 'G'],
          ['G', 'R', 'R'],
          ['G', 'G', 'G']]
measurements = ['R', 'R']
motions = [[0,0], [0,1]]
sensor_right = 1.0
p_move = 0.5
p = localize(colors,measurements,motions,sensor_right,p_move)
correct_answer = (
    [[0.0, 0.0, 0.0],
     [0.0, 0.33333333, 0.66666666],
     [0.0, 0.0, 0.0]])
```

The video shows sensor_wrong and p_stay defined globally and referenced from sense() and move() helper functions. This approach will not actually work when submitting the assignment. You must pass in all required state into your sense() and move() functions from inside your localize() routine: def sense(p, colors, measurements, sensor_wrong): ... def move(p, motion, p_stay): ... def localize(...): ... p = move(p, motion, p_stay) p = sense(p, colors, measurement, sensor_wrong) [You should NOT modify the function signature of localize()]