**崇新学堂**

**2022－2023学年第一学期**

实 验 报 告

课程名称： Experiments of Introduction to EECS

实验名称： Robots in Hallway

专 业 班 级 崇新21

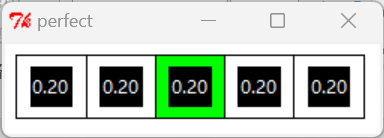
学 生 姓 名 余昊 葛明烨 崔宇鑫

实 验 时 间 2022年11月29日

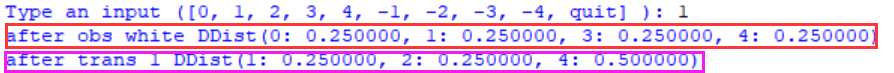
**Step1**

***Check Yourself 1.***

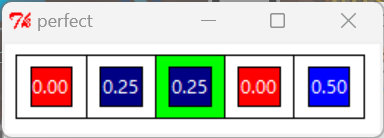
In the initial state, the robot does not know its position and does not observe the color of the room. In all rooms, the probability is equal and it is reflected as black.



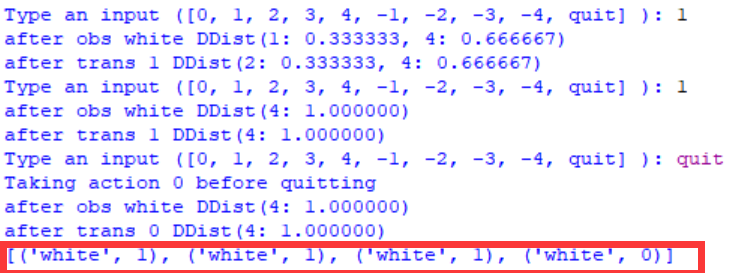
The robots are randomly distributed in a room. After we give the first command, the first thing the robot does is make an observation about the color of the room. In this perfect model, where there is no noise of any kind, the robot gives a belief state after observing the color of the room it is in. It should be noted that this belief state is based on the previous belief state, and then the robot calculates the new belief state after executing the instruction.

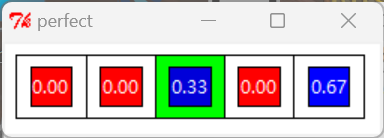
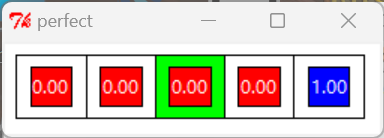


In the initial state, the robot observes that it is in a room with a true color of white, and gives its belief state, that is, all the white rooms are equally divided in probability, and then gives a new belief state based on the observed color after perfectly executing the instruction that we input to move one step to the right.



By parity of reasoning：



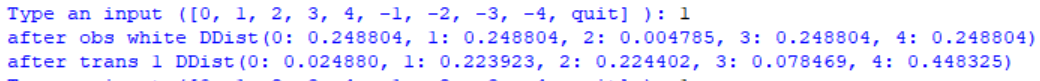
 

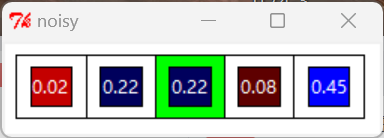
The quit command indicates that we expect the car to complete the last color observation to get the belief state, exit without action, and output each color observed by the robot itself and the motion instructions we input by a list.

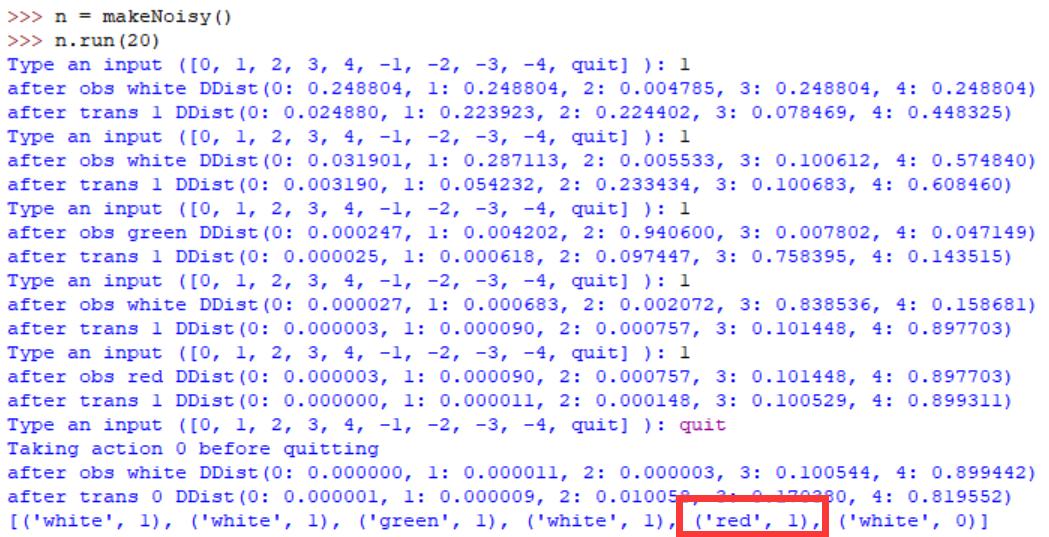
**Step2**

***Check Yourself 2.***

In a system with noise, the meaning of data printed by our python shell is exactly the same as that in the ideal model, except that there is noise in our observation model and motion model, and each corresponding state obtained is a probabilistic statistical model





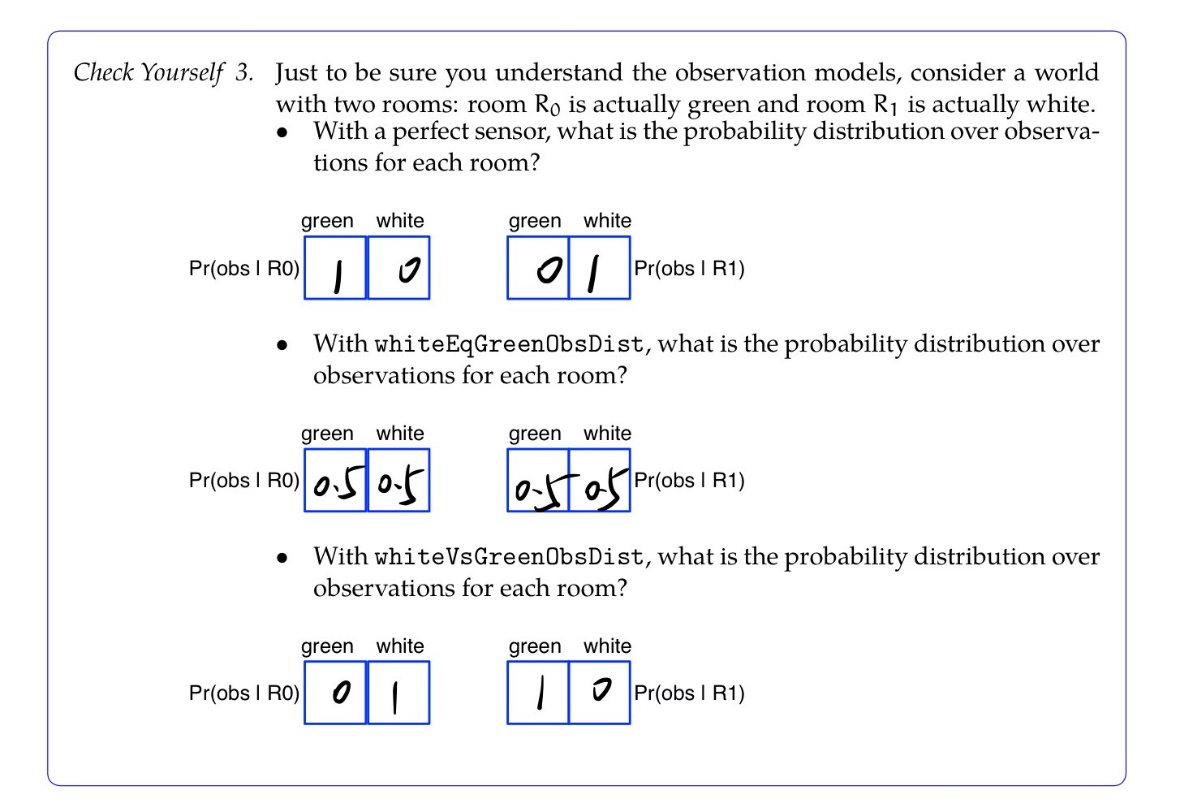
In the initial state, we let the robot move one unit to the right, and we could not get the probability that the percentage is at the leftmost end is 0, which proves that our trans model is not perfect 

Red, which does not exist in the actual true color, appears in the room color observation, which confirms that there are errors in our observation model

**Step3**

**Wk.11.1.1**

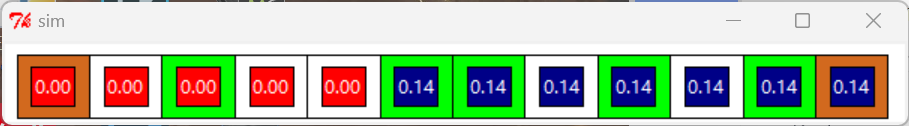
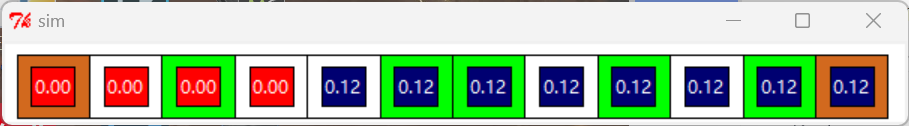
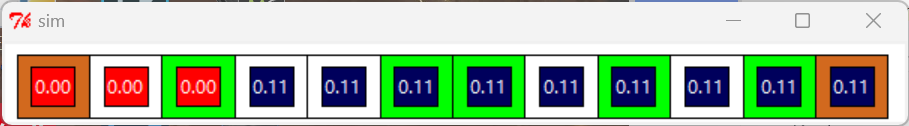
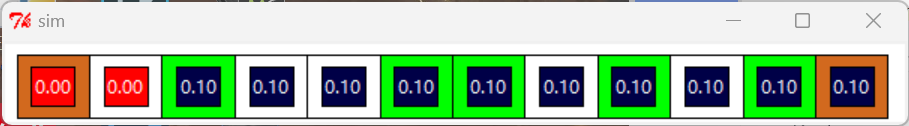
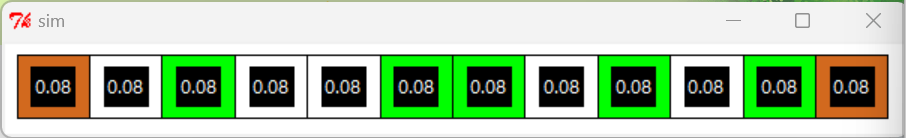
***Check Yourself 3.***

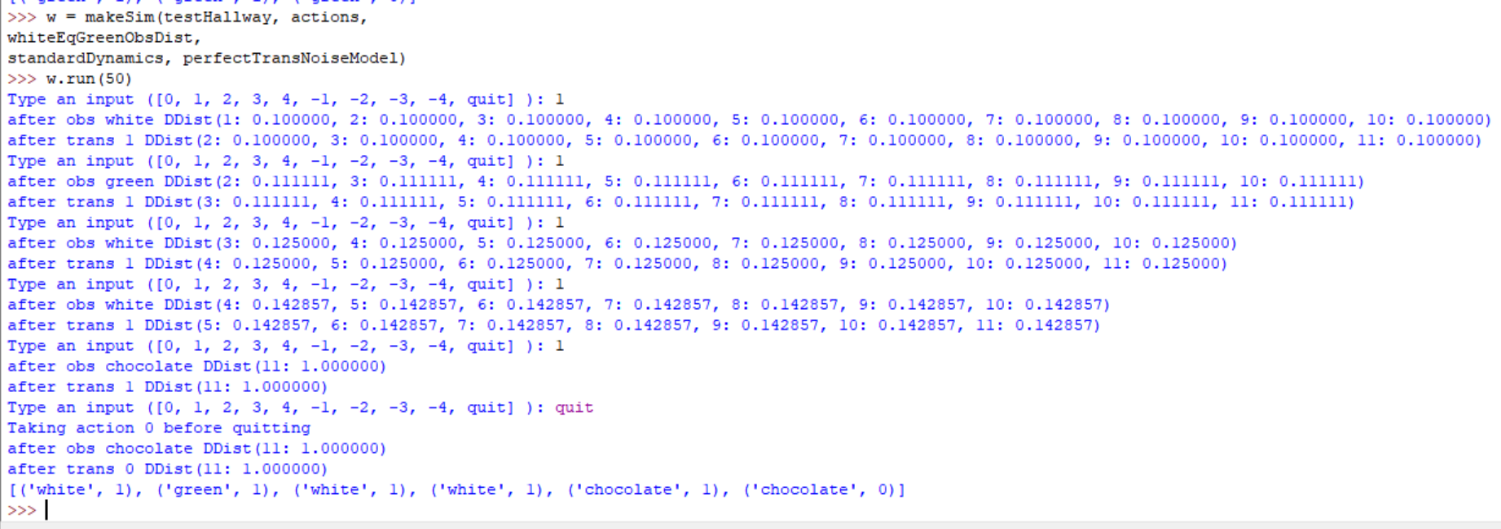


**Step4**

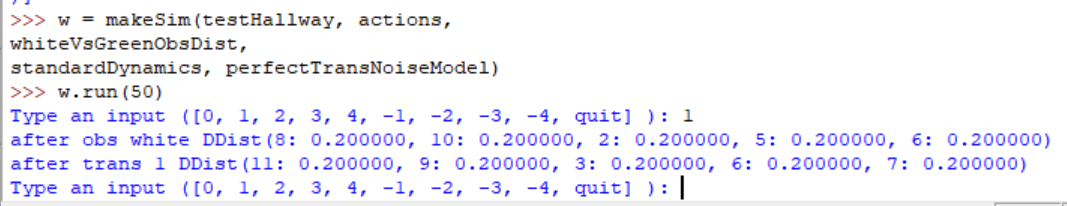
Our state trans model is perfect at this point

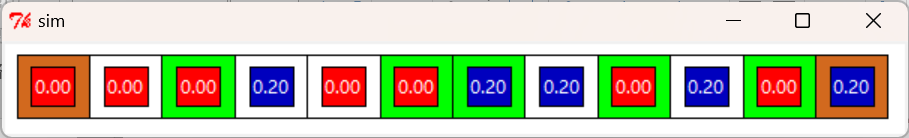
1. In using whiteEqGreenObsDist, we actually showed that the robot could not correctly distinguish the meaning of green and white. It can be imagined that for the robot, "green" and "white" are actually a separate color that does not belong to the. In the shell, it is obvious that all of our green rooms and white rooms are equally likely without moving to the defined 'chocolate' color end (where the robot can be located directly)





2. In using whiteVsGreenObsDist, we show that the robot will always perceive the true color "green" as white, and the corresponding true color "white" as green.



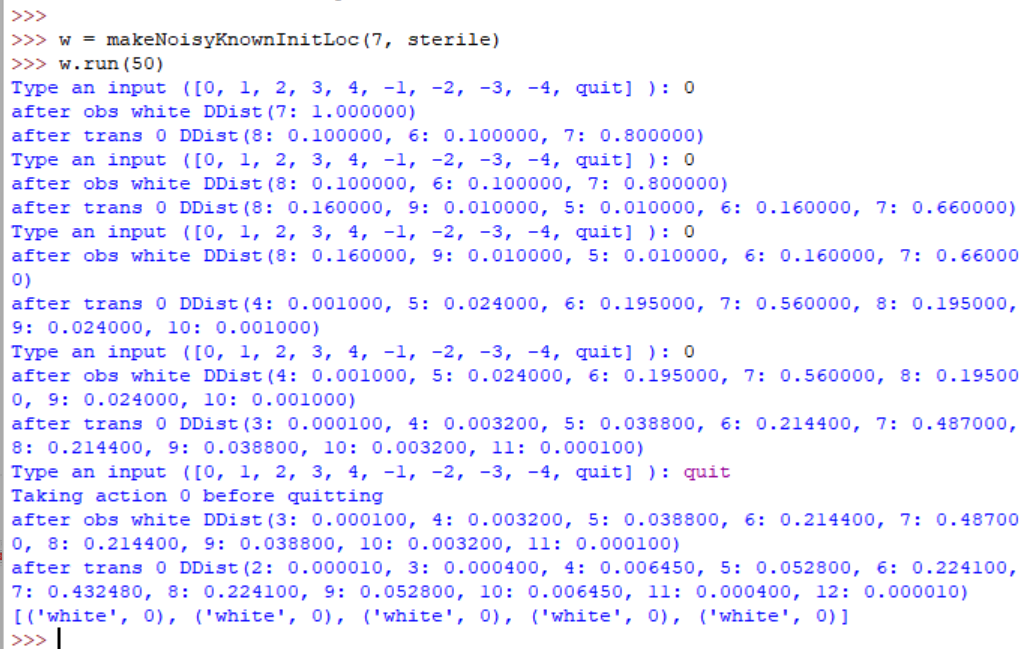
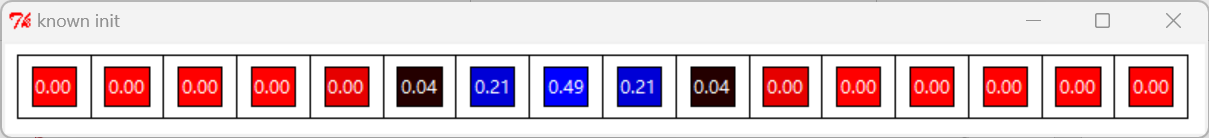
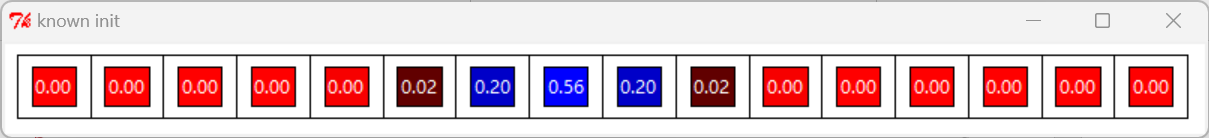
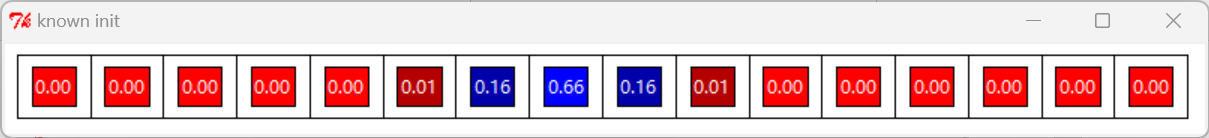
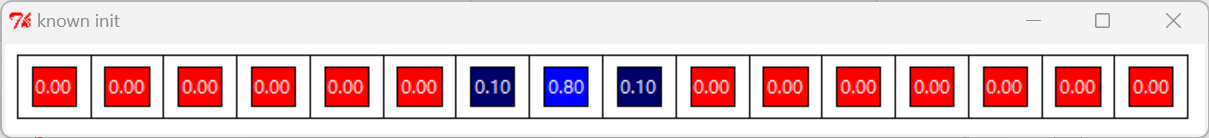
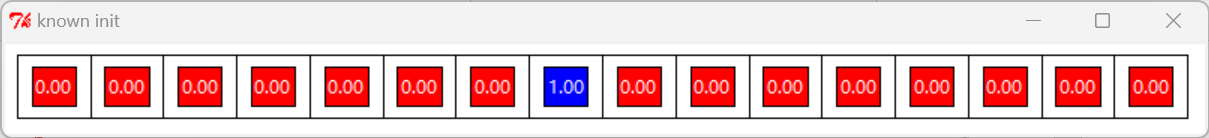
After the initialization of the analysis robot, it is observed that the room color is white. Then according to our logic, the initial state of the robot is actually a room with a true color of "green", corresponding to the probability positions of rooms 2, 5, 6, 8 and 10 are exactly the same as those printed 

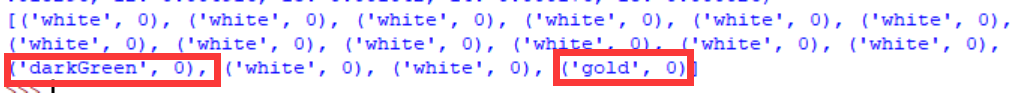
We believe that this does not affect the output and position judgment in this relatively symmetrical structure

**Step5**

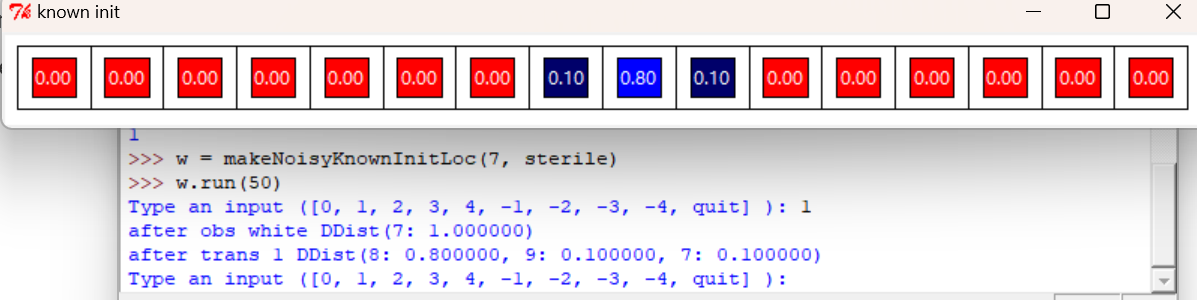
**Wk.11.1.2**

**Step6**

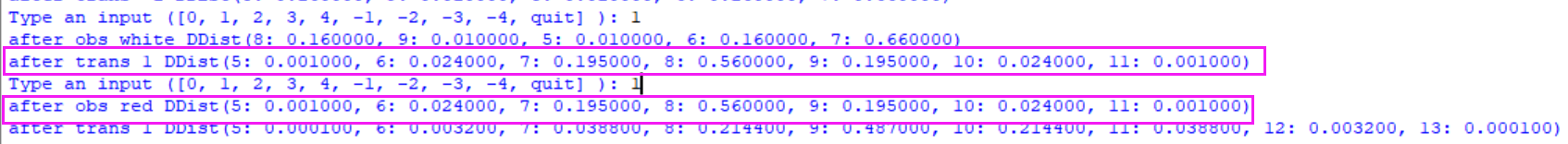




Since there is noise in our transformation model, even if the input "0" is given, our robot is still likely to drift left and right. After repeated accumulation, the robot will be distributed in space, rather than keeping precisely at the origin



A similar principle is followed when the robot moves from side to side



It's worth noting that our belief states didn't change if the robot read colors that weren't in the actual room because of the noise

***Checkoff 1.***

For A sensor that always reads ’black’ no matter what room it is in，This is completely unusable for us.

The whiteEqGreenObsDist is actually a slightly unreliable perfect sensor that combines two colors into one. The whiteVsGreenObsDist is a sensor that affects the operation of a system in an asymmetric structure. But both are available and even useful under certain conditions.

**Step7**

**Wk.11.1.4**

**Step8**

**Wk.11.1.5**

**Step9**

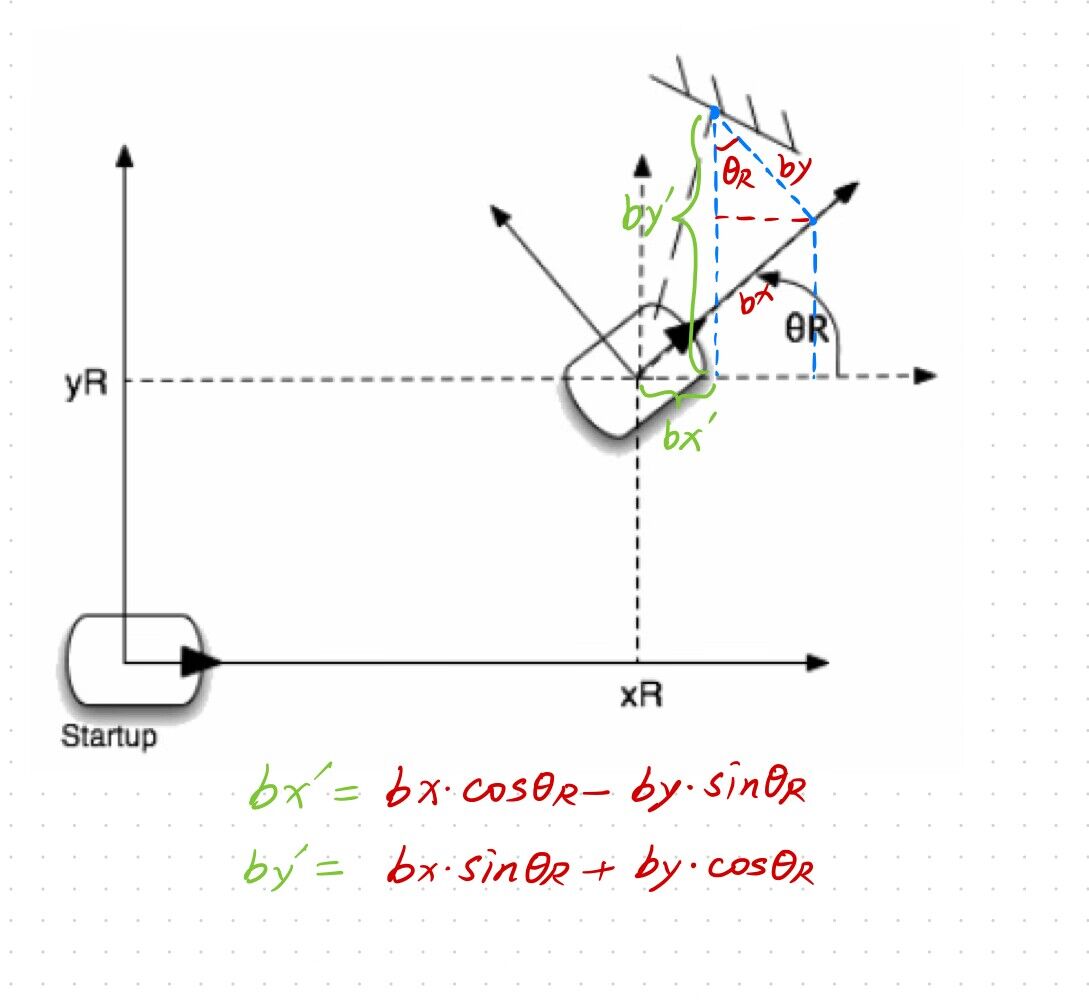
Preparing to localize

**Wk.11.1.6**

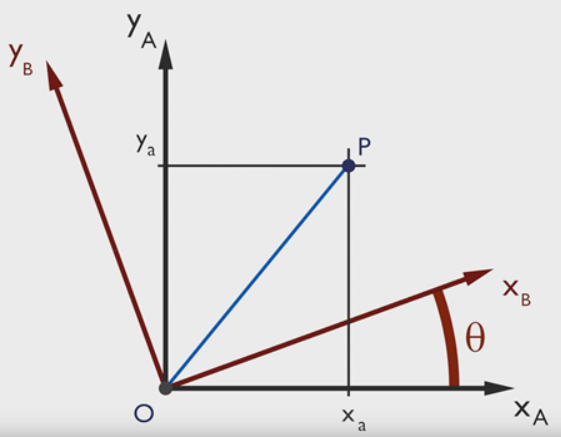
The target of this problem:Given a distance measurement from one of the sonars, return an instance of Point (look at the documentation of the util module), in the global odometry frame, the same coordinate frame that the robot's pose is measured, representing where the sonar beam bounced off an object.

**1.** Understand the coordinate change（The coordinate change formula has been given in WK.11.1.6）  

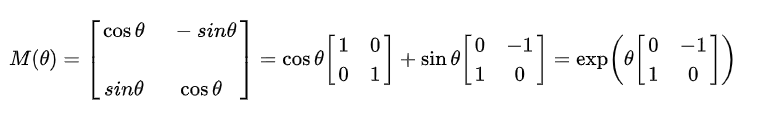

(1)Based on the geometry:

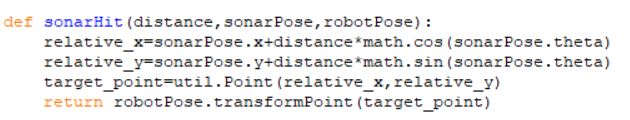


(2) Based on the rotation matrix



In two-dimensional space, rotation can be defined by a single angle. As a convention, a positive angle represents a clockwise rotation. The matrix that rotates the column vector of Cartesian coordinates counterclockwise with respect to the origin is:



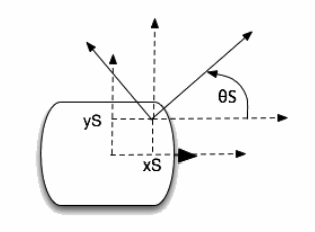
**2.** Write the function ‘sonarHit’:  


Explanation:

(1)

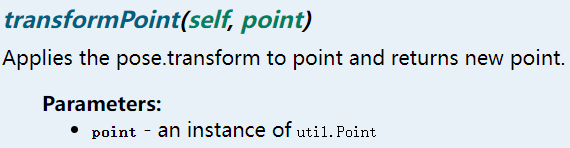
Relative\_x **->** bx Relative\_y **->** by.

sonarPose.x **->**xS sonarPose.y**->**yS



**,**

**(2)**

****

transformPoint(self,Point) is a method of class ‘Pose’.

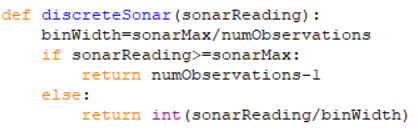
robotPose.transformPoint(p) returns an instance of the Point class, the value of Point.x and Point.y is equal to applying the unit rotation matrix to the x,y coordinates of p (the rotation angle is pose.theta), and translating the rotated coordinates (robotPose.x, robotPose.y) units in a two-dimensional coordinate system. Through this method, the coordinate value of the object detected by the sonar under the global odometry frame can be obtained.

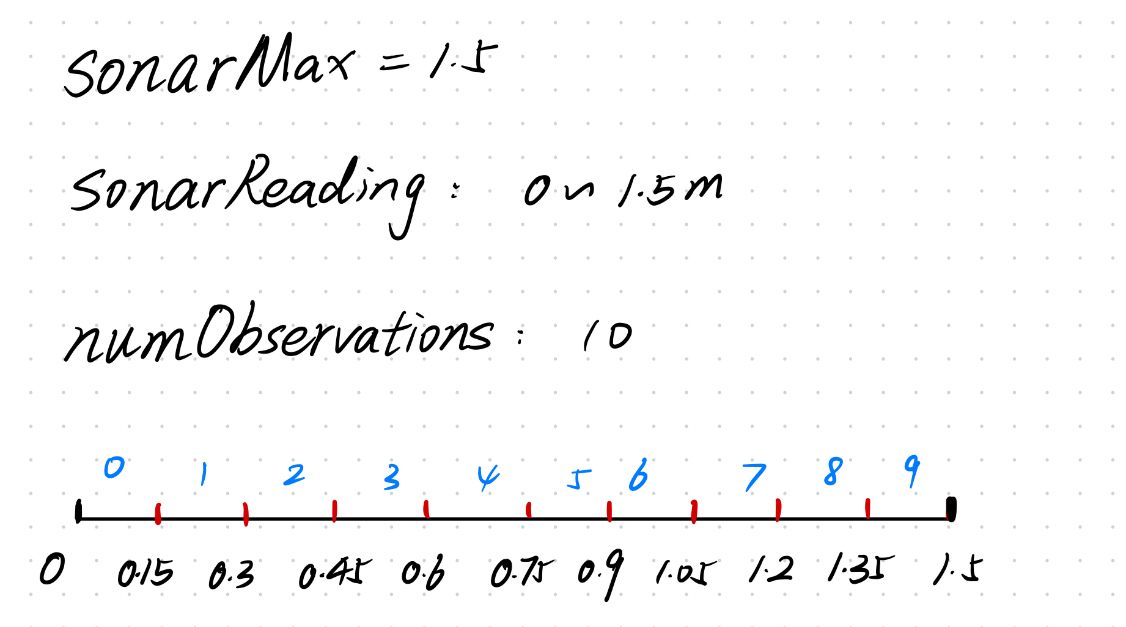
**Step10**

**Wk.11.1.7**

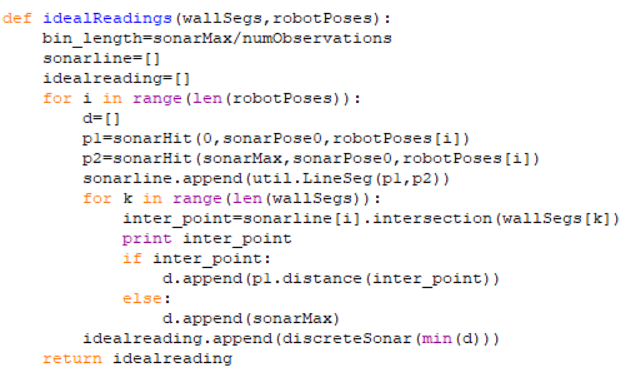
Ideal Sonar Readings：

1. discreteSonar:





Range:0~1.5 should be discretized into a fixed number, numObservations, of bins. And the bin indices are from 0 to numObservations-1(0~9). When the sonarReading is within 0~1.5m, it will fall into any bin with an indice from 0 to 9. “int(sonarReadings/binWidth)” can represent the index value corresponding to a sonar reading. For example, when sonarReadings=1.4, int(sonarReadings/binWidth)=9. So, sonarReadings=1.4 corresponds to the discreted ideal sonarReadings “9”

2.idealReadings:  


Explanation for my code:

bin\_length defines the width of the bin.

sonarLine=[], idealreadings=[] create two empty lists for later use.

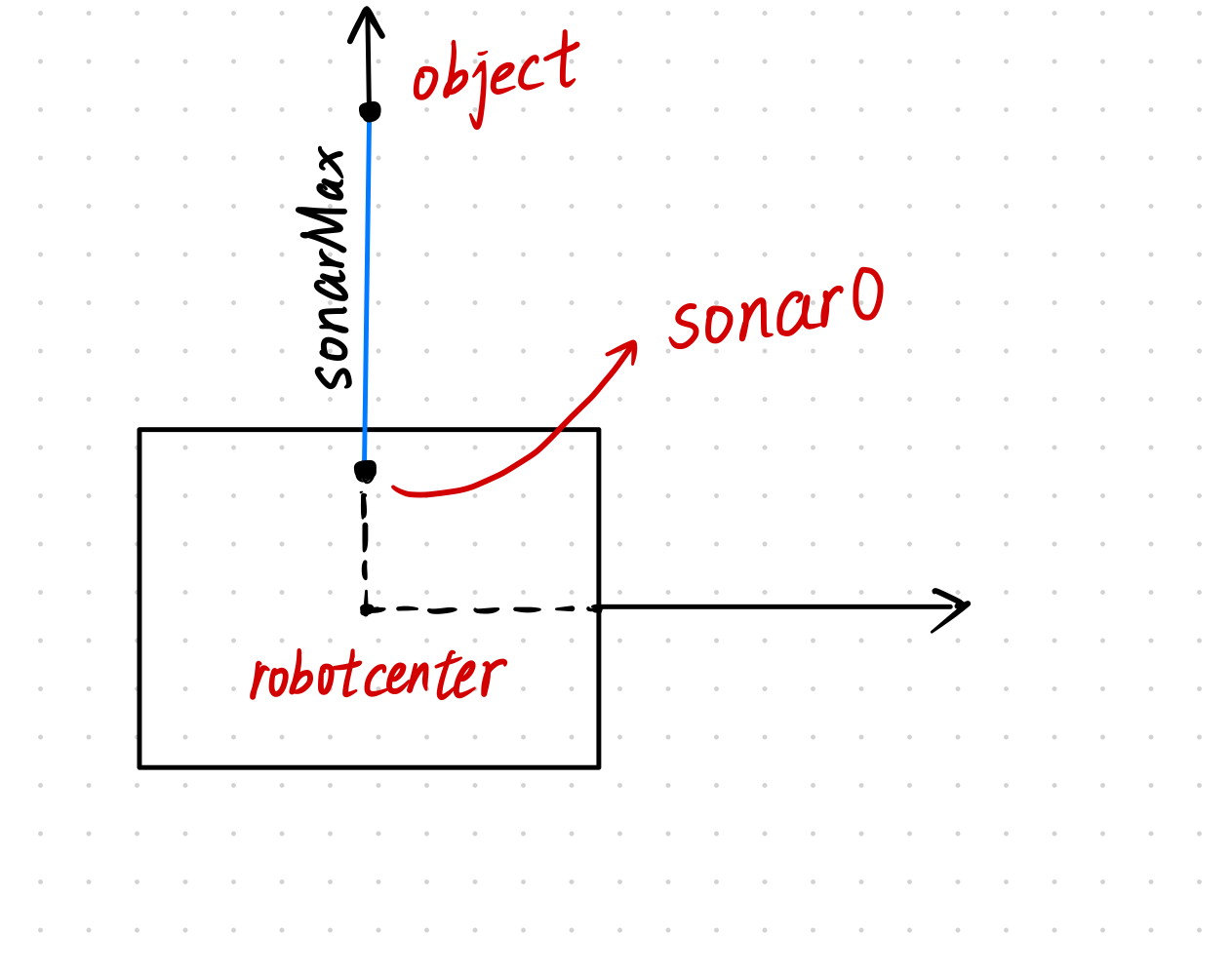
The purpose of the outer loop in the code is to read a gesture of robotPoses with each loop.

(*robotPoses* is a list of elements in the list that are instances of the Pose class that represent the robot's posture at this moment, e.g. robotPoses[0] represents the robot's first pose.)

P1=sonarHit(0,sonarPose0,robotPoses[i]) utilizes the the function *sonarHit* completed in WK.11.1.6, which returns the coordinate point P1 of the center of the sonar0 (relative to the center of the robot) in the global coordinate system

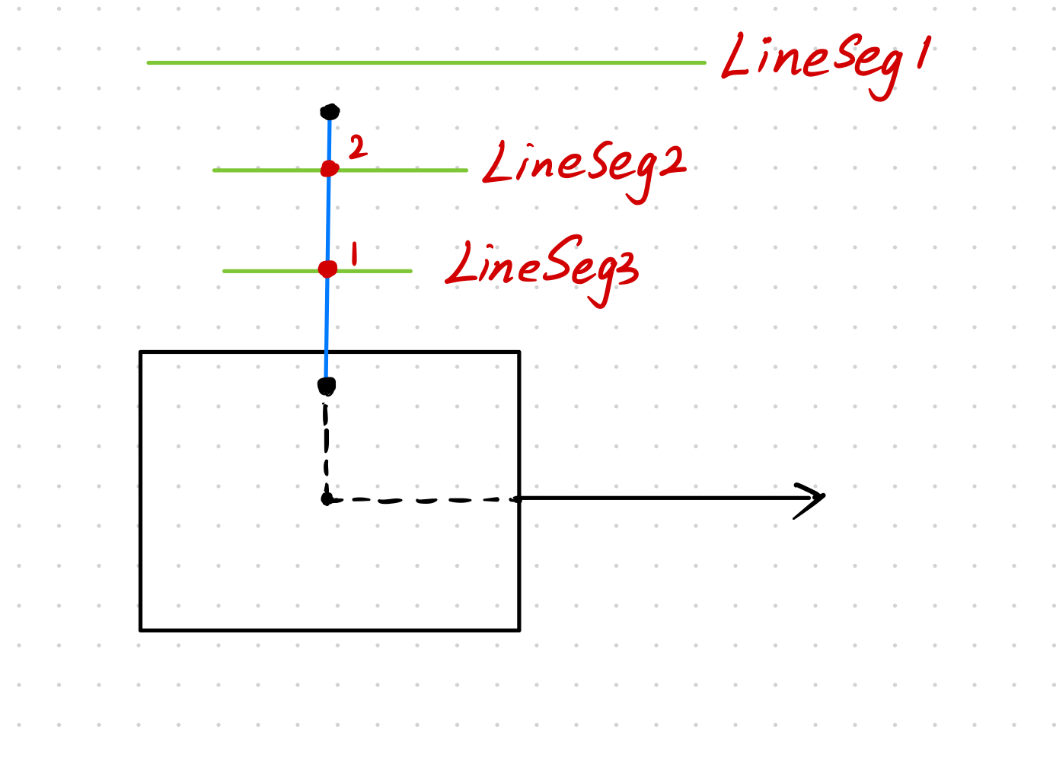
P2=sonarHit(sonarMax,sonarPose0,robotPose[i]) returns the position of the object in the global coordinate system at the detection distance of sonarMax (1.5m here) from sonar0

As shown in the following figure:



With the two points P1 and P2, we can generates a probe segment and the length of it is sonaMax. (the blue segment as shown in the figure). Each loop of the outer loop adds a probe line segment of length sonarMax to the list *sonarLine*. The inner loop reads an instance of util.LineSeg in the list *WallSeg* each time. The role of the inner loop is to find the nearest object detected by sonar0. The intersection method of class LineSeg can determine whether sonar0's probe segment intersects with a given segment in *WallSeg.*

As shown in the following figure:

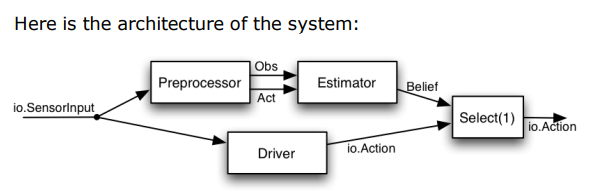


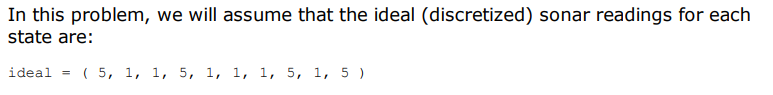
Suppose the *WallSeg* has only 3 line segments, namely LineSeg1, LineSeg2, and LineSeg3. As can be seen from the figure, the detection line segment (blue line segment) has an intersection point with LineSeg2 and LineSeg3, and no intersection with LineSeg1, through the distance method of the Point class, the distance between the point where the center of sonar0 is located and the intersection point (that is, the detection distance) can be calculated, if there is no intersection, we return the detection distance as sonarMax. We use list d, add the three detection distances to the list after the end of the inner loop, judge the minimum value in list d in each outer loop, take min(d) as an argument to the function named *discreteSonar,* and add the return value to the *idealreading.* Each time the outer loop, the discreted value added to the *idealreading* corresponds to the discreted ideal reading of sonar0 in each pose of the robot.

**Step11**

**Wk.12.2.3**

**Localization：**

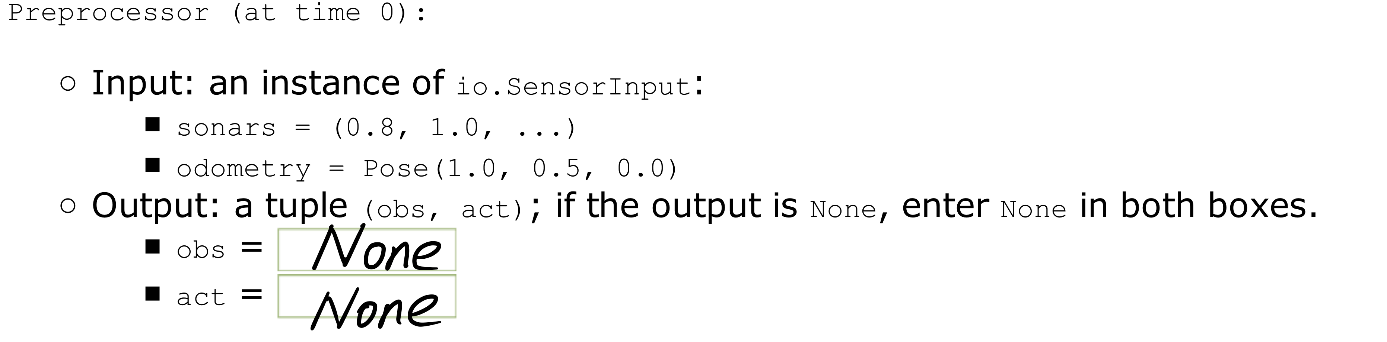
****

****

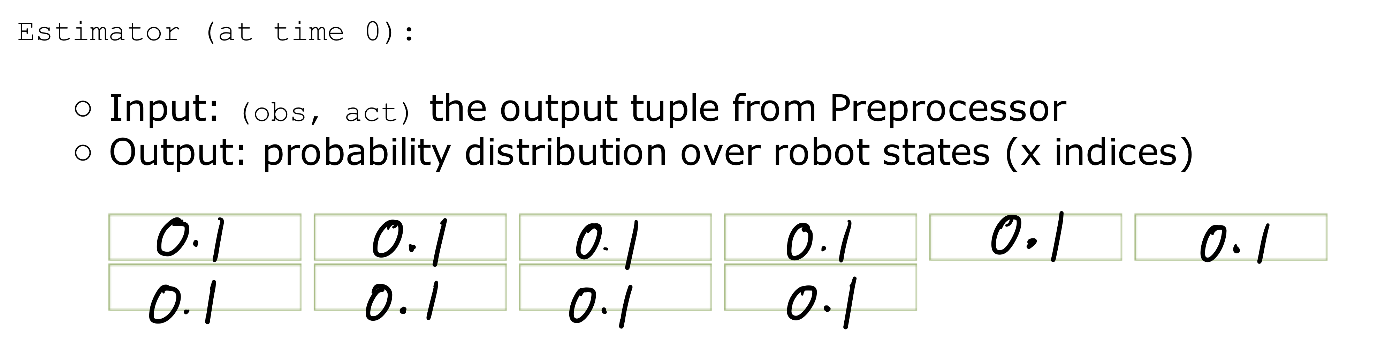
Preprocessor output an observation from step t-1 and the action that took place between steps t-1 and t. The input of the Estimator is the output of the Preprocessor, The Estimator is an instance of a state machine that acts as a state estimator.

1. **At time 0**

***Preprocessor：***

****

***Estimator：***

****

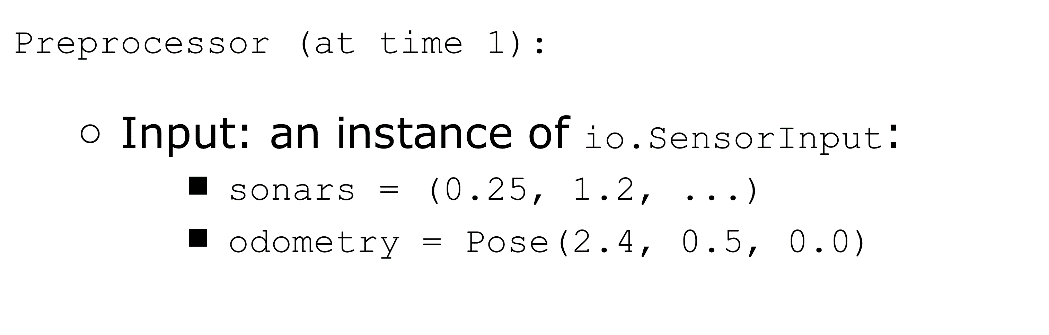
Explanation*：*

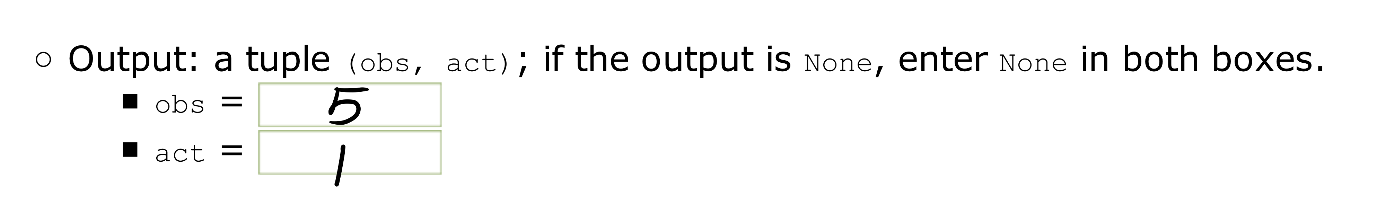
When the state machine is first started there will not be any previous observation or odometry available, so we will simply generate an output of None. We will make a special modification in our state estimator, so that if the input to the machine is None, then no state update will be made at all.

As for the Estimator,we can let the starting distribution be uniform.

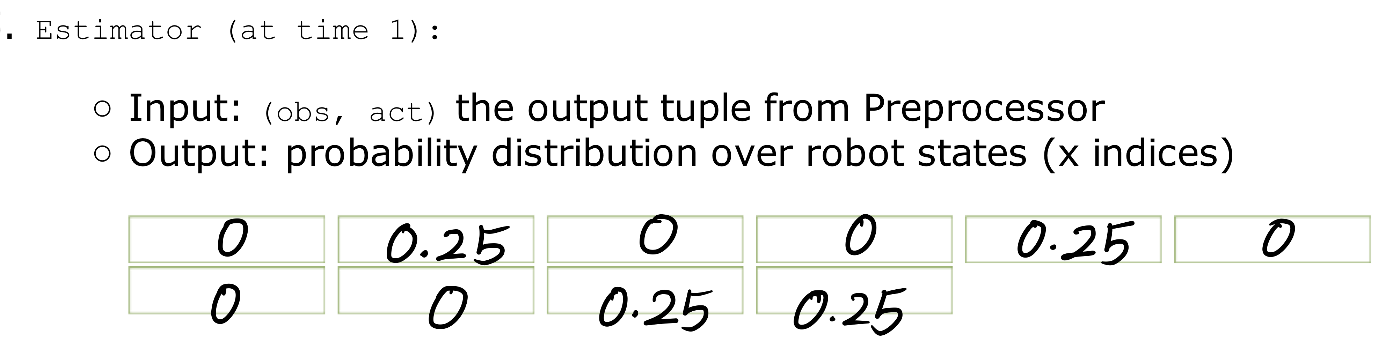
1. **At time 1**

***Preprocessor：***

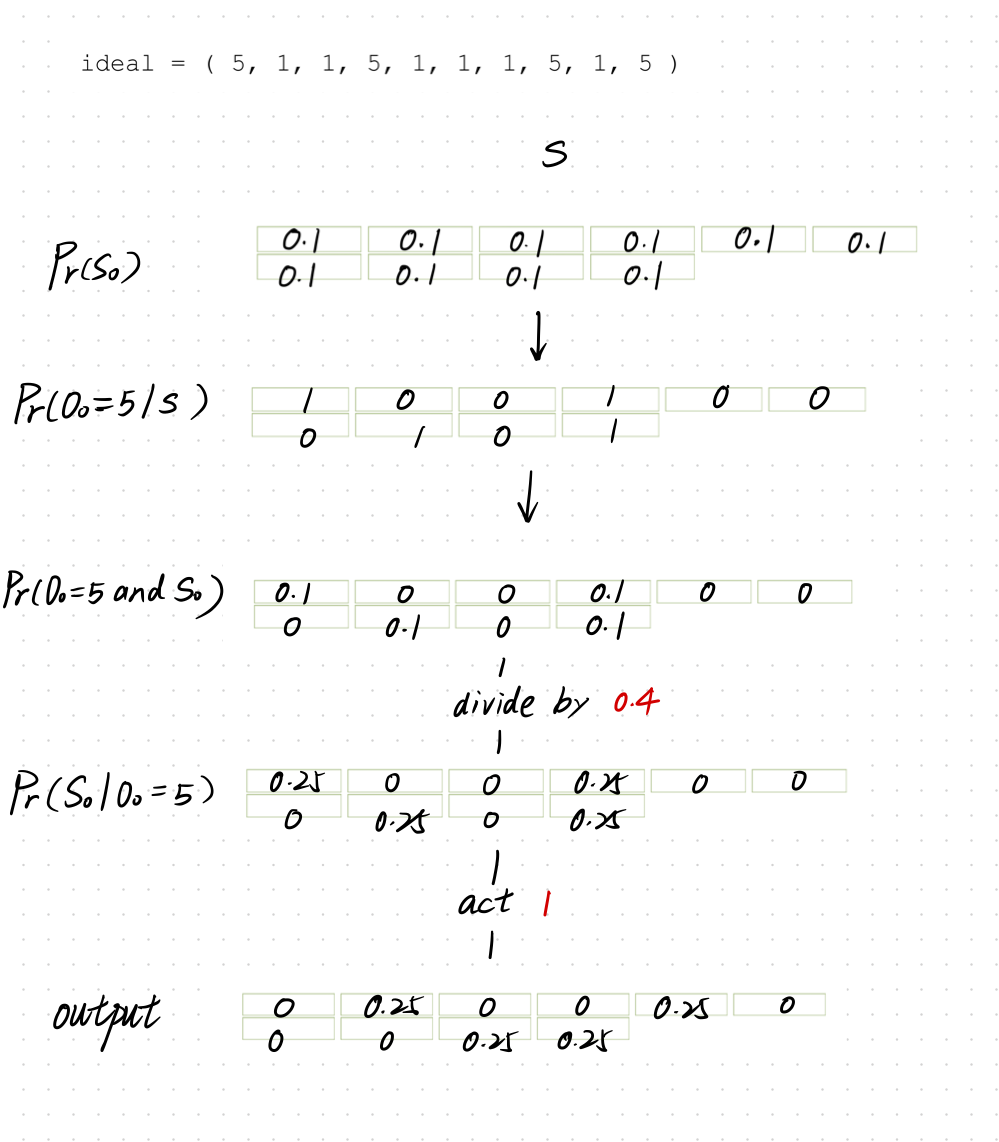
****

****

***Estimator:***

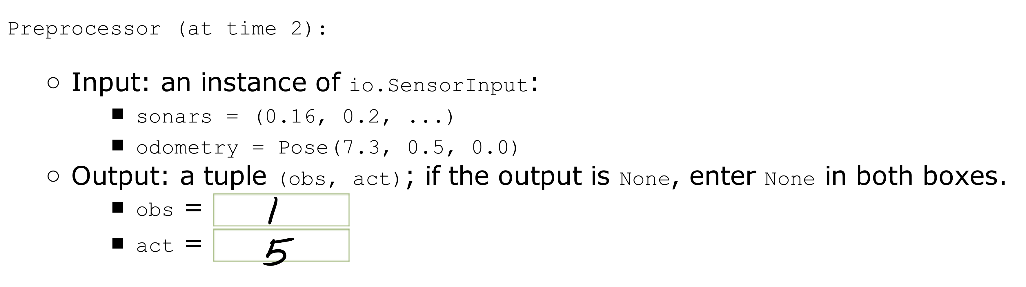
******

Explanation:

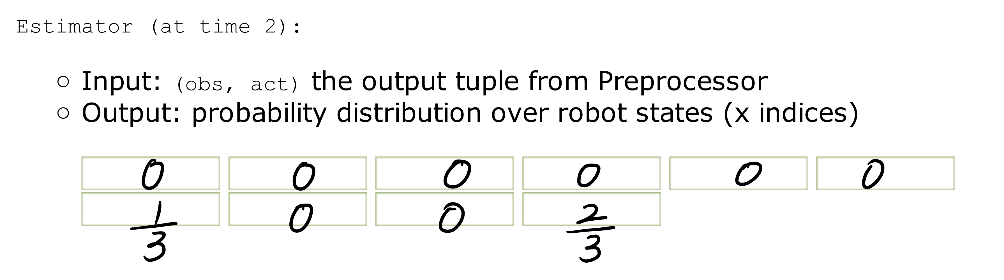
****

1. **At time 2**

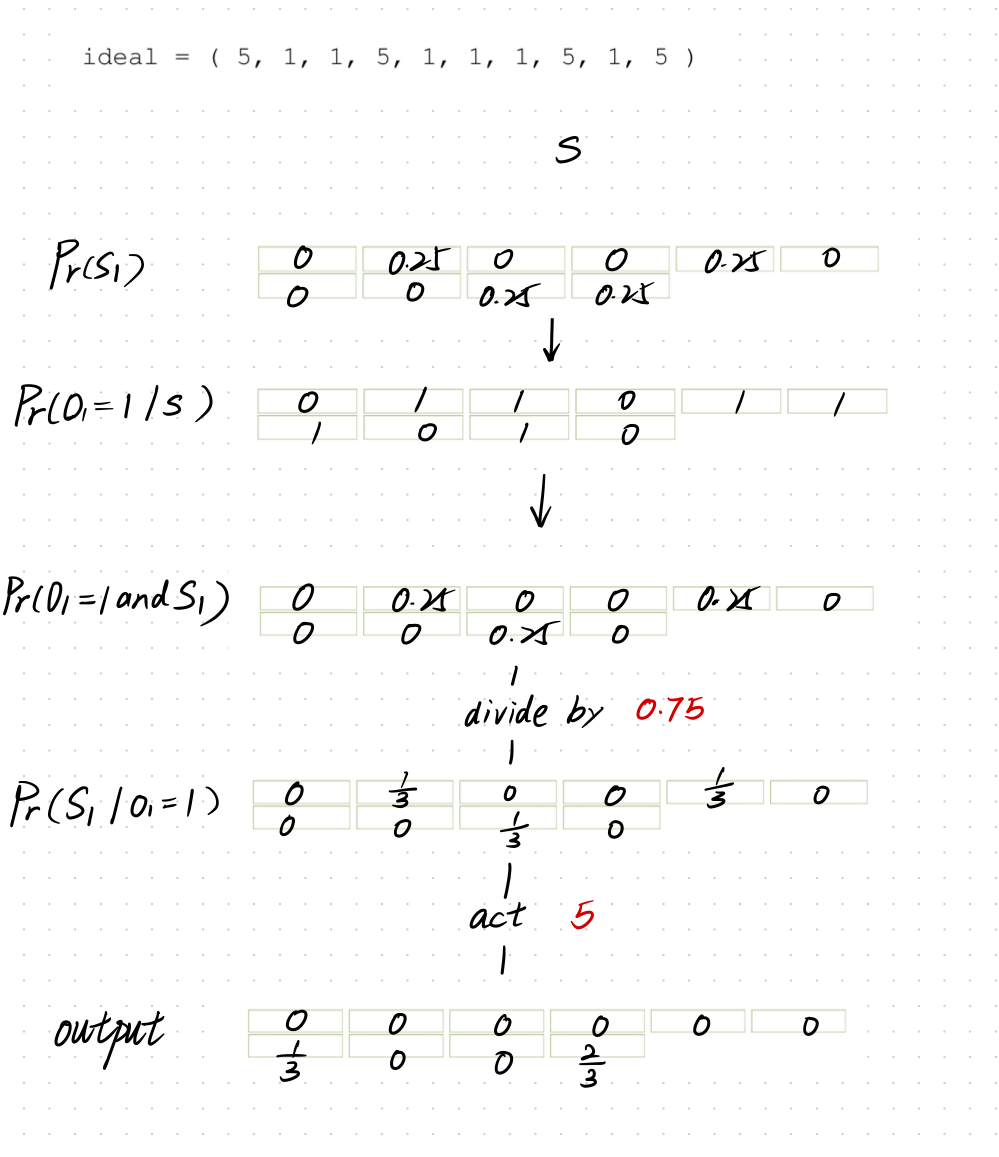
***Preprocessor：***

****

***Estimator:***

****

Explanation:

****

**Summary**