Homework 5: Car Tracking

109550135 范恩宇

Part I. Implementation (20%) :

Part 1 : Emission probabilities

```python
    def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
        # BEGIN_YOUR_CODE (our solution is 9 lines of code, but don't worry if you deviate from this)
        for row in range(self.belief.getNumRows()):
            for col in range(self.belief.getNumCols()):
                post = self.belief.getProb(row, col)
                mean = math.sqrt((agentX - util.colToX(col)) ** 2 + (agentY - util.rowToY(row)) ** 2)
                state = util.pdf(mean, Const.SONAR_STD, observedDist)
                self.belief.setProb(row, col, post * state)
        self.belief.normalize()
```

In part 1, iterate over each tile and updates belief value (post*state) based on current (post*state) and the calculated probability density(state) .

During this process , use “mean” between agent and tile to calculate the probability density(state) , obtaining the former by the formula of calculating true distance .

Part 2 : Transition probabilities

```python
    def elapseTime(self) -> None:
        if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
            return
        # BEGIN_YOUR_CODE (our solution is 10 lines of code, but don't worry if you deviate from this)
        n_belief = util.Belief(self.belief.numRows, self.belief.numCols, value=0)

        for key, value in self.transProb.items():
            (oldTile, newTile) = key
            n_belief.addProb(newTile[0], newTile[1], self.belief.getProb(*oldTile) * value)
            #row,col,delta
        n_belief.normalize()
        self.belief = n_belief
        # END_YOUR_CODE
```

In part 2 , set a new belief distribution through a learned transition model .

First , declare a variable “n_belief” and initialize belief values in it as 0 , which means the suggested new belief .

Second , for each new location the car may exist (“newTile” in “self.transProb”) , update corresponding belief value in “n_belief” by product of current probability(“self.belief.getProb”(*oldTile”)) and the transition probability(“value” in “self.transProb.items()”) .

Finally , normalize “n_belief” and update “self.belief” with the former .

Part 3.1 : Particle filtering(observe)

```python
    def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
        # BEGIN_YOUR_CODE (our solution is 12 lines of code, but don't worry if you deviate from this)
        n_weight = dict()
        n_particles = dict()

        for key, value in self.particles.items():
            (row, col) = key
            mean = math.sqrt((agentX-util.colToX(col))**2 + (agentY-util.rowToY(row))**2)
            state = util.pdf(mean, Const.SONAR_STD, observedDist)
            n_weight[(row, col)] = value * state

        for i in range(self.NUM_PARTICLES):
            key = util.weightedRandomChoice(n_weight)
            if key in n_particles:
                n_particles[key] += 1
            else:
                n_particles[key] = 1

        self.particles = n_particles
        # END_YOUR_CODE
        self.updateBelief()
```

In part3-observe , re-weight particle distribution in each tile through the “mean” between agent and tile , then set particles based on the new weight .

First, declare “n_weight” and “n_particles” as the new weight and new sampled particles respectively .

Second , re-weight particle distribution in each tile through the “mean” between agent and tile . Weights of the tiles are updated by the product of current weight(value) and the probability density(state) .

Finally , sample “self.NUM_PARTICLES” times with the new weight . Then set the re-sampled distribution as the new one .

Part 3.2 : Particle filtering(elapseTime)

```python
    def elapseTime(self) -> None:
        # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
        n_particles = collections.defaultdict(int)
        for tile in self.particles:
            for i in range(self.particles[tile]):
                key = util.weightedRandomChoice(self.transProbDict[tile])
                n_particles[key] += 1
        self.particles = n_particles
        # END_YOUR_CODE
```

In part3-elapseTime , set a new belief distribution through a learned transition model . We can update the particle distribution by re-sample it through the learned transition model easily because the belief values are based on the particle distribution .

First , declare a “n_particles” as new particle distribution.

Second , iterate every tile in “self.particles”(number of particles in each tile), and re-sample “self.particles[tile]” times.

Third , get new weight (self.transProbDict[tile]) by the learned transition model for each particle sample , and randomly get a tile, “key” . Then record this particle in “n_particles”.

Finally , set the re-sampled distribution as the new “self.particles” .