Project3 汽车跟踪问题

注意事项:

本次任务包含书面部分和编程部分。

我们提供的代码和脚本可以在 car.zip 中下载。

您应该在 submission.py 中的 # BEGIN_YOUR_CODE 和 # END_YOUR_CODE 之间修改代码，但您也可以在此块之外添加其他帮助函数。请不要更改 submission.py 以外的文件。

您的代码将根据两种类型的测试用例进行评估：基本测试和隐藏测试，您可以在 grader.py 中找到。基本测试是完全提供给您的，不会使用大型输入或极端案例来测试您的代码。隐藏测试更加复杂，会对您的代码进行压力测试。隐藏测试的输入在 grader.py 中提供，但正确的输出不会提供。为了运行测试，您需要使代码和 grader.py 以及 graderUtil.py 在相同的目录下。然后，您可以通过输入以下命令运行所有测试:

```
python grader.py
```

这只会告诉您是否通过了基本测试。对于隐藏测试，脚本会在您的代码执行时间过长或崩溃时向您发出警报，但不会告知您是否获得了正确的输出。您也可以通过输入以下命令来运行单个测试（例如，3a-0-basic）

```
python grader.py 3a-0-basic
```

我们强烈建议您阅读并理解测试用例，创建自己的测试用例，而不仅仅是盲目运行 grader.py。

问题描述

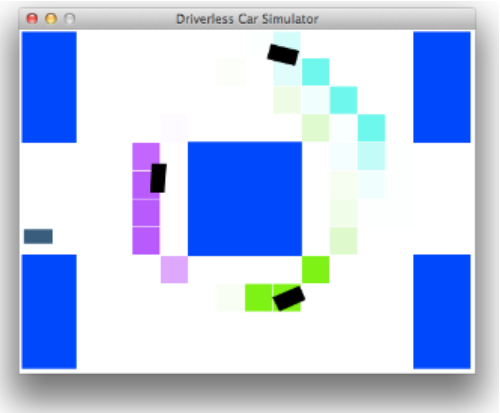
这个作业是由 Chris Piech 编写的 Driverless Car 作业的修改版本。

世界卫生组织的一项研究发现，全世界每年有124万人死于交通事故。因此，人们对开发自动驾驶技术非常感兴趣，这种技术可以提高驾驶精度，减少死亡人数。构建自动驾驶系统是一项极其复杂的工作。在本次作业中，您将重点关注传感系统，该系统允许我们根据噪声传感器读数来跟踪其他汽车。

准备：你将在这个作业中运行两个文件 grader.py 和 drive.py 。grader.py 在Python 3上运行，但 drive.py **在Python 2上运行。** drive.py 不用于任何评分目的，它只是用来可视化您将要编写的代码，并帮助您了解不同的方法如何导致不同的行为(并从中获得乐趣!)让我们从手动驾驶开始。*再次注意，您需要使用Python2运行 drive.py* 。

```
Python drive.py -l Lombard -i none
```

你可以使用方向键或 'w'， 'a'和'd'来控制方向。向上键和“w”使你的车加速前进，左键和 'a' 使方向盘转向左边，右键和 'd' 使方向盘转向右边。注意，你不能倒车或原地转弯。按 'q' 退出。你的目标是从起点一直开到终点(绿色方框)而不出事故。在不知道其他车辆位置的情况下，你能在狂风呼啸的伦巴第大街上做得多好？如果你出了车祸，不要担心;教学人员只有 4/10 次到达终点线。60%的事故率是非常糟糕的，这就是为什么我们要用人工智能来研究自动驾驶。



drive.py 中的标志：

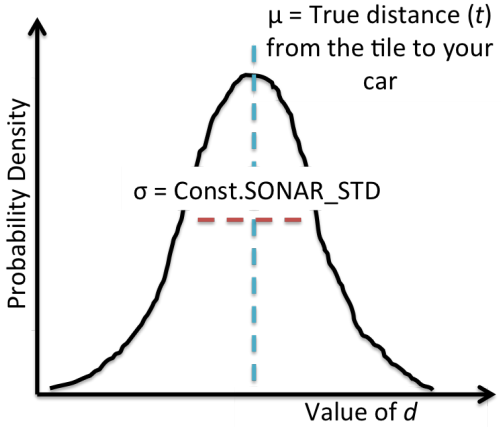
- -a：启用自动驾驶（与手动相对）。
- -i：使用 none、exactInference、particleFilter（近似）计算其他车辆位置的置信度分布。
- -l：使用此地图（例如 small 或 lombard ）。默认为 small。
- -d：通过在地图上显示所有汽车进行调试。
- -p：所有其他汽车保持停车状态（这样它们就不会移动）。

建模汽车位置：我们假设世界是一个二维矩形网格，您的汽车和其他 K 辆汽车都驻留在上面。在每个时间步骤 t 中，您的汽车会对每辆汽车的距离得到一个带噪的估计。为简化起见，我们假设每个其他汽车都是独立移动的，并且每个汽车传感器读数的噪声也是独立的。因此，在接下来的讨论中，我们将独立地考虑每辆汽车（理论上，我们将假设只有一辆其他汽车）。

在每个时间步骤 t，令 $C_t \in R^2$ 为一对坐标，代表单个其他汽车的实际位置（未被观察到）。我们假设存在一个局部条件分布 $p(c_t \mid c_{t-1})$ 来控制汽车的移动。令 $a_t \in R^2$ 成为您的汽车位置，您观察并控制它。为了最小化成本，我们使用基于麦克风的简单传感系统。麦克风提供给我们 D_t，这是一个高斯随机变量，其均值等于你的车和另一辆车之间的真实距离，方差为 σ^2 （在代码中，σ是 Const.SONAR_STD，大约是一辆车的长度的三分之二）。用符号表示为:

$$D_t \sim \mathcal{N}\left(\|a_t - C_t\|, \sigma^2\right)$$

例如，如果您的车的位置在 $a_t = (1, 3)$ 而另一辆车的位置在 $C_t = (4, 7)$，那么实际距离为5，D_t 可能为4.6或5.2等。使用 util.pdf(mean, std, value) 计算给定均值 mean 和标准差 std 的高斯分布的概率密度函数（PDF），在值value处进行评估。请注意，评估概率密度函数并不返回概率 — — 密度可以超过1 — — 但是为了本任务的目的，您可以将其视为概率。以您与汽车之间的距离 $\mu = \|a_t - C_t\|$ 为中心，噪声距离观测 D_t 的高斯概率密度函数，如下图所示：



你的任务是实现一个汽车跟踪器，它(近似地)计算后验分布 $P(C_t \mid D_1 = d_1, \ldots, D_t = d_t)$ (你对另一辆车位置的置信)，并为每个 $t = 1, 2, \ldots$ 更新它。我们将使用这些信息来实际驾驶汽车(即，设置 a_t 来避免与 c_t 碰撞)，因此您不必担心这部分。

为了简化问题，我们将把世界离散化为由（row，col）表示的格子，其中 0 <= row < numRows，0 <= col < numCols 。对于每个格子，我们存储一个表示我们在该格子上相信有一辆汽车的概率。这些值可以通过 self.belief.getProb(row, col) 访问。要从一个格子转换为一个位置，请使用 util.rowToY(row) 和 util.colToX(col)。

以下是本次作业的组成部分概述：

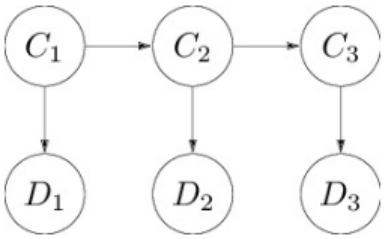
- 问题1（书面作答）将为您提供一些有关简单贝叶斯网络上概率推断的练习。
- 问题2和问题3（编程）中，您将实现 ExactInference，该算法计算出另一辆车在（row，col）位置的完整概率分布。
- 在问题4（编程）中，您将实现 ParticleFilter，它使用基于粒子的表示来处理这个相同分布。

在我们开始之前，有几个重要的注意事项：

- 代码部分的任务很短而且直接，总共不超过30行，但前提是你的概率理解清晰！如果不清晰，请参考前面的提示。
- 符号提醒：我们使用小写表达式 $p(x)$ 或 $p(x|y)$ 表示本地条件概率分布，这些分布由贝叶斯网络定义。我们使用大写表达式 $P(X = x)$ 或 $P(X = x|Y = y)$ 表示联合和后验概率分布，在贝叶斯网络中没有预定义，但可以通过概率推理计算。
- 正如作业开始时所提到的，记得使用Python 2运行 drive.py。请注意，drive.py 不用于评分。

问题1：贝叶斯网络基础（书面部分）

首先，让我们看一下汽车跟踪问题的简化版本。仅对于这个问题，设 $C_t \in \{0, 1\}$ 是我们希望在时间步 $t \in \{1, 2, 3\}$ 处观察到的汽车的实际位置。设 $D_t \in \{0, 1\}$ 是 t 时刻测量到的汽车位置的传感器读数。下面是贝叶斯网络(实际上是HMM)的样子:



初始车的分布是均匀的。即，对于每一个值 $c_1 \in \{0, 1\}$:

$$p(c_1) = 0.5$$

下面的局部条件分布决定了汽车的移动(汽车以概率 ϵ 移动)。对于每个 $t \in \{2, 3\}$:

$$p\left(c_t \mid c_{t-1}\right) = \begin{cases} \epsilon & \text{if } c_t \neq c_{t-1} \\ 1 - \epsilon & \text{if } c_t = c_{t-1} \end{cases}$$

下面的局部条件分布控制传感器读数中的噪声(传感器以 η 概率报告错误的位置)。对于每个 $t \in \{1, 2, 3\}$:

$$p\left(d_t \mid c_t\right)=\begin{cases}\eta & \text{if } d_t \neq c_t \\ 1-\eta & \text{if } d_t=c_t\end{cases}$$

下面，您将被要求在给定不同传感器读数的第二时间步(C_2)找到汽车位置的后验分布。

重要提示：对于下面的计算，请尝试遵循课堂中描述的一般策略(边缘化非祖先变量和条件变量，接着执行变量消除)。尽量将规范化延迟到最后。这样做会比试图处理大量方程式更容易获得洞察力。

问题：

1. (2') 假设我们有第二个时间步的传感器读数，$D_2=0$。计算后验分布 $\mathbb{P}(C_2=1 \mid D_2=0)$。
2. (2') 假设一个时间步长已经过去，我们得到另一个传感器读数 $D_3=1$，但我们仍然对 C_2 感兴趣。计算后验分布 $\mathbb{P}(C_2=1 \mid D_2=0, D_3=1)$。得到的表达式可能比较复杂。
3. (3') 假设 $\epsilon=0.1, \eta=0.2$
   - 计算 $\mathbb{P}(C_2=1 \mid D_2=0)$ 和 $\mathbb{P}(C_2=1 \mid D_2=0, D_3=1)$。保留四位有效数字。
   - 第二个传感器读数 $D_3=1$ 如何改变了结果？基于汽车的位置和相关的传感器观测，解释为什么这个变化是有意义的。
   - 保持 $\eta=0.2$，如何设置 ϵ 使得 $\mathbb{P}(C_2=1 \mid D_2=0)=\mathbb{P}(C_2=1 \mid D_2=0, D_3=1)$？基于汽车的位置和相关的传感器观测，解释这个结果。

问题2：发射概率（编程部分）

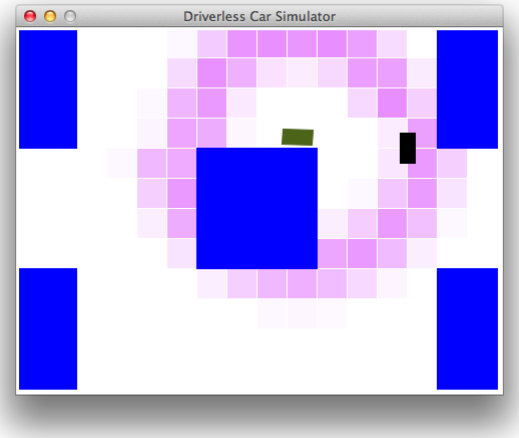
在这个问题中，我们假设另一辆车是静止的(例如，对于所有时间步长 $t, C_t=C_{t-1}$)。您将实现一个函数 observe，它基于新的距离测量 $D_t=d_t$，根据前一时刻的后验概率：

$$\mathbb{P}(C_t \mid D_1=d_1, \ldots, D_{t-1}=d_{t-1})$$

更新当前的后验概率：

$$\mathbb{P}\left(C_t \mid D_1=d_1, \ldots, D_t=d_t\right) \propto \mathbb{P}\left(C_t \mid D_1=d_1, \ldots, D_{t-1}=d_{t-1}\right) p\left(d_t \mid c_t\right)$$

其中，我们以及在发射概率中乘以之前描述过的 “建模汽车位置” 的概率 $p(d_t \mid c_t)$。当前的后验概率在 ExactInference 中的 self.belief 中储存。



问题：

(7') 在 submission.py 中的 ExactInference 类里，填充 observe 方法。这个方法应该就地修改 self.belief，以更新每个位置的后验概率，给定观测到的与另一辆车之间的带噪距离。完成后，您应该能够通过围绕着静止的车绕圈的方法找到静止的车（使用标志 -p 表示车辆不会移动）。

注意：

- 您可以立即开始使用精确推理进行驾驶。记得在运行drive.py时使用Python 2。

  ```
  python drive.py -a -p -d -k 1 -i exactInference
  ```

  您也可以关闭 -a 以手动驾驶。
- 通常最好在本地计算机上运行 drive.py ，但是如果您决定在 cardinal/rice 上运行它，请使用 -X 或 -Y 选项 ssh 到 farmshare 机器，以获取图形界面；否则，您将收到一些显示错误消息。注意：预期这个图形界面会有点慢..... drive.py 不用于评分，只是为您提供可视化和游戏乐趣！
- 在开始之前，请阅读 util.py 中的 Belief 类的代码... 您需要在本次作业的几个代码任务中使用此类。
- 记得在更新后对后验概率进行归一化。（在 util.py 中有一个有用的函数可供此用）
- 在小地图上，自动驾驶员有时会在前往目标区域之前在中间的区块周围盘旋。一般来说，不要太担心汽车的精确路径。相反，重点是您的车辆跟踪器是否正确推断出其他车辆的位置。
- 如果您的车偶尔发生事故，不要担心！无论您是人还是AI，事故都会发生。但是，即使发生了事故，您的驾驶员也应该意识到该区域存在另一辆车的高概率。

问题3：转移概率（编程部分）

现在，让我们考虑另一辆车按照转移概率 $p(c_{t+1} \mid c_t)$ 移动的情况。我们已经在 self.transProb 中为您提供了转移概率。具体来说，self.transProb[(oldTile, newTile)] 是指在时间步骤 t 时，另一辆车在 oldTile 中，时间步骤 $t+1$ 时在 newTile 中的概率。

在本部分中，您将实现一个函数 elapseTime，该函数将关于汽车在当前时间 t 位置的后验概率：

$$\mathbb{P}\left(C_t = c_t \mid D_1 = d_1, \ldots, D_t = d_t\right)$$

通过递归，更新为下一个时间步 $t+1$，基于相同条件的概率：

$$\mathbb{P}\left(C_{t+1} = c_{t+1} \mid D_1 = d_1, \ldots, D_t = d_t\right) \propto \sum_{ct} \mathbb{P}\left(C_t = c_t \mid D_1 = d_1, \ldots, D_t = d_t\right) p\left(c_{t+1} \mid c_t\right)$$

同样，后验概率储存在 ExactInference 类的 self.belief 中。

问题：

(7') 实现 elapseTime 方法以完成 ExactInference 。当所有这些都完成后，通过运行以下命令自动驾驶，您应该能够很好地跟踪一辆移动的汽车。记得使用 python2 运行 drive.py。

```
python drive.py -a -d -k 1 -i exactInference
```

注意：

- 你也可以在有多辆车的情况下自动驾驶:

  ```
  python drive.py -a -d -k 3 -i exactInference
  ```

- 你也可以开车去伦巴第:

  ```
  python drive.py -a -d -k 3 -i exactInference -l lombard
  ```

  记住使用Python 2来运行 drive.py。在 Lombard，自动驾驶可能会尝试在街道上来回行驶，然后才会驶向目标区域。同样，关注汽车跟踪组件，而不是实际驾驶。

问题4：粒子滤波（编程部分）

虽然精确推断在小地图上效果很好，但它浪费了大量精力来计算每个可用位置的概率，即使是那些不太可能有汽车的位置。我们可以用粒子滤波器来解决这个问题。粒子过滤器的更新复杂度与粒子数量成线性关系，而不是与位置数量成线性关系。

要了解粒子滤波如何工作的概念解释，请查看这段使用粒子滤波估计飞机高度的视频。

在这个问题中，您将在 submission.py 中为 ParticleFilter 类实现两个简短但重要的方法。完成后，您的代码应该能够像精确推断一样有效地跟踪汽车。

问题：

(18') 我们已经为您提供了部分代码。例如，粒子已经被随机初始化。您需要填写 observe 和 elapseTime 函数。self.particles 需要被修改，它是由(row, col) 到该位置粒子数的一个映射。每次您对粒子位置进行重采样时，需要更新self.belief 。

您应该使用与精确推断相同的转换概率。由粒子滤波器生成的信念分布与通过精确推断得到的信念分布相比，我们预期它看起来噪声更大。记得在Python 2中运行 drive.py。

如果您要调试，您可能需要从停放的汽车标志(-p)和显示汽车标志(-d)开始。

注意：注意: util.weightedRandomChoice 内部的随机数生成器在不同系统版本的Python(例如，Unix和Windows版本的Python)上的行为不同。

提交

请提交 zip 文件，包含 car.pdf 和 submission.py ，命名为：**pj3-id-name.zip** 并通过超星学习通提交。

截止日期：23:59 2023.12.24

关于本次作业的任何问题，请联系助教：叶爵达