Лабораторная работа 6 — Дискретный фильтр и фильтр частиц

1. **Дискретный фильтр**

   Предположим, что робот живет в мире, состоящем из 20 ячеек, расположенных подряд, и находится в 10-й ячейке. Поскольку мир робота ограничен, он не может выйти за пределы указанной области. На каждом временном шаге робот может выполнить команду движения либо вперед, либо назад. К сожалению, робот выполняет действия не с идеальной точностью. Известно, что при движении вперед может произойти следующее:

   i) С вероятностью 25 % робот не будет двигаться

   ii) С вероятностью 50 % робот переместится в следующую ячейку

   iii) С вероятностью 25 % робот переместится на две клетки вперед

   iv) Вероятность того, что робот будет двигаться не в том направлении или более чем на две клетки вперед, равна 0 %.

   Все то же самое справедливо при движении назад, только в противоположном направлении.

   Поскольку мир робота ограничен, вероятности движения в граничных ячейках будут другие, а именно:

   - Если робот, находясь в последней клетке, попытается двигаться вперед, то он останется в той же клетке с вероятностью 100 %
   - Если робот, находясь в предпоследней клетке, попытается двигаться вперед, то он останется в этой же клетке с вероятностью 25 %, а с вероятностью 75 % переместится в следующую клетку

   Все то же самое справедливо при движении назад из первой клетки.

   Реализуйте дискретный байесовский фильтр и, построив соответствующие графики, оцените положение робота после выполнения девяти (9) последовательных команд движения вперед и трех (3) последовательных команд движения назад.

   Подсказки:

   Начните с `bel = numpy.hstack((numpy.zeros(9), 1, numpy.zeros(10)))`.

   Вы можете проверить себя, — сумма результатов должна равняться единице (с очень небольшой погрешностью из-за ограниченной точности компьютерных вычислений).

   Будьте осторожны с границами мира!

2. **Фильтр частиц**

   Используйте готовый проект `pf_framework`, в котором уже описана необходимая для работы фильтра частиц последовательность действий и реализована визуализация.

   В архиве `pf_framework` содержатся следующие папки:

   `data` содержит файлы с данными о мире и показаниями датчиков.

   `code` содержит "заготовку" для написания фильтра частиц.

   Вы можете попробовать запустить фильтр частиц в терминале: `python particle_filter.py`. Однако он будет работать корректно только после того, как вы дополните код.

   (a) Допишите `sample_motion_model`, реализовав модель процесса на основе одометрии и осуществив сэмплинг. Функция генерирует новые положения, используя старые значения положений, измерения одометрии δ_{rot1}, δ_{trans} и δ_{rot2} и шумовые параметры модели процесса: $[\alpha_1, \alpha_2, \alpha_3, \alpha_4] =$ [0.1, 0.1, 0.05, 0.05]. После обновления процесса функция возвращает новый набор параметров.

   (b) Допишите функцию `eval_sensor_model`. Эта функция реализует модель измерений (используется датчик расстояния). В качестве входных данных принимаются положения и наблюдения ориентиров. Функция возвращает список весов для фильтра частиц. Вместо определения вероятности достаточно определить правдоподобие $p(z|x, l)$. Среднеквадратичное отклонение гауссовского шума измерений с нулевым средним значением составляет $\sigma_r = 0.2$.

(c) Допишите функцию `resample_particles`, осуществив отсев. Функция принимает в качестве входных данных набор частиц и соответствующие веса и возвращает набор частиц, прошедших отсев.

Подсказки:

Для считывания данных, полученных с датчиков, и данных об ориентирах используются словари. Словари обеспечивают более простой способ доступа к структурам данных на основе одного или нескольких ключей. Функции `read_sensor_data` и `read_world_data` в файле `read_data.py` считывают данные из файлов и создают соответствующий словарь с отметками времени в качестве первичных ключей.

Чтобы получить доступ к данным датчика из словаря `sensor_readings`, используйте:

```
sensor_readings[timestamp,'sensor']['id']
sensor_readings[timestamp,'sensor']['range']
sensor_readings[timestamp,'sensor']['bearing']
```

Чтобы получить доступ к данным одометрии, используйте:

```
sensor_readings[timestamp,'odometry']['r1']
sensor_readings[timestamp,'odometry']['t']
sensor_readings[timestamp,'odometry']['r2']
```

Чтобы получить доступ к положениям ориентиров из словаря `landmarks`, используйте:

```
position_x = landmarks[id][0]
position_y = landmarks[id][1]
```