

# Relatório MAC0318 EP5

Renato Lui Geh — NUSP: 8536030

## 1 Definições e notação

Vamos chamar de  $X$  a variável aleatória discreta e finita que representa em que pedaço do espaço unidimensional discretizado o robô está. Então  $P(X = x)$  é a probabilidade do robô estar na “célula”  $x$ . Vou chamar de  $m$  o número de células total. Suponha que o espaço unidimensional mede  $d$ . Vou chamar de  $\bar{d}$  o tamanho de cada célula, ou seja,  $\bar{d} = \lfloor d/m \rfloor$ .

A variável aleatória discreta e finita  $Z$  representa o sonar. Como dito no enunciado,  $Z \sim \mathcal{N}$ . As gaussianas representando os erros e ruídos do sonar quando captando as caixas e buracos serão denotadas por  $\mathcal{N}_b$  e  $\mathcal{N}_g$  respectivamente.

Na probabilidade de ação  $P(X' = x' | X = x, u)$ ,  $X'$  é a variável aleatória da posição do robô na célula  $x'$  após a ação  $u$ .

## 2 Execução

Para rodar o EP5, é só rodar com `python3` sem argumentos.

```
python3 localization.py
```

Assim que o código terminar de pré-computar as matrizes de probabilidade, o programa vai abrir uma janela com um histograma da distribuição de probabilidade  $P(X)$ , onde cada barra é uma célula. Neste histograma, células vermelhas representam caixas e amarelas representam os buracos. Uma célula azul indica a posição verdadeira do robô.

Após aberto o histograma, o programa continuará rodando a espera de *input*. A tela mostrará algo semelhante a:

```
Starting simulation at position: 5
Ready. Press ? for help message.
```

Os comandos para o programa funcionam igual ao **NORMAL** mode do Vim. Um comando é representado por um caractere. O caractere pode conter um quantificador (um número) que altera o comportamento do comando. Abaixo segue a lista de comandos dada pelo comando de ajuda `?`:

```
-----
This controller works very much like vim. Available commands are:
  h - Go left and apply correction and prediction.
  j - No-op. Dont move, but apply correction and prediction.
  l - Go right and apply correction and prediction.
  k - Force correction only, with no prediction.
  c - Shows the current models constraints and localization settings.
  q - Quit.
  ? - Show this help message
Capitalized equivalents apply only prediction with no correction:
  H - Go left and apply only prediction.
  J - No-op. Dont move, but apply prediction only.
  K - No-op. Applies correction. The same as k.
  L - Go right and apply only prediction.
Every command can be quantified (just like vim!). A number before a
  command means the command should be repeated that many times. For
  example:
  2l - Go right two units and then apply correction and prediction.
  10H - Go left ten units and then apply only prediction.
  j - Compute prediction and correction values and dont move.
  5k - Compute correction values five times.
When omitting a quantifier, the command assumes the quantifier is 1.
-----
```

Os comandos são lidos em tempo-real, então logo que o programa receber um comando válido, ele o executará. Assim não é preciso ficar apertando Enter para validar o input. Se o comando não for válido, o programa descartará o input anterior e pedirá uma nova sequência de caracteres para input.

Quando aberto, o histograma é posto em foco. Esta será a única vez que a janela será dada foco. É possível manter a janela aberta de lado enquanto são dados os comandos que isso não fará com que o foco seja alterado. Em vários Desktop Environments, colocar a janela em “Always on top” pode facilitar a vida.

Como a mensagem de ajuda mostra, o quantificador modifica o comportamento do comando. Omitir este quantificador é a mesma coisa que se o quantificador fosse 1.

Após cada comando de movimento ou computação, uma mensagem indicará qual comando foi feito e quais são as posições reais do robô antes e depois do movimento.

O programa não lida casos em que o robô fosse sair do espaço (quando a posição do robô é maior ou igual a  $m$  ou menor que 0).

O comando `c` imprime informações úteis do mapa, gaussianas e do `Range` (que será co-

mentado posteriormente). O robô não pode andar mais do que o que estiver determinado em “Bounds for number of steps” de uma só vez. Por exemplo:

```
Map Properties:
  Sensor Gaussians:
    Means: 10.0 | Variance: 1.0
    Means: 15.0 | Variance: 1.0
  Discretization bin size: 100
  True size of each bin: 1.7
  Size of precomputed probability matrices:
    P(Z|X)      (sensor probability distribution): (20, 100)
    P(X|X,u)    (action probability distribution): 49 x (100, 100)
  Bot attached? False
Range Properties:
  Unique commands available: [-1, 0, 1]
  Bounds for number of steps: [-25, 0, 25]
  Pivot: 24
```

Neste caso, o comando 251 não deve ser executado. No entanto, 241 pode. A razão por trás disso é que este valor é o que foi pré-computado na matriz de probabilidade para até 24 passos de uma só vez. É possível aumentar este valor, como será visto na próxima seção.

### 3 Configurações

Para mudar a configuração do espaço, das gaussianas, matrizes de probabilidade ou discretização, é preciso mudar o próprio código. Para isso, foram criadas algumas funções.

A função `new_config` cria uma nova configuração de mapa. Ela toma como argumentos (em ordem):

1.  $\mu_b$ : Média da gaussiana do sonar para as caixas.
2.  $\sigma_b$ : Variância da gaussiana do sonar para as caixas.
3.  $\mu_g$ : Média da gaussiana do sonar para os buracos.
4.  $\sigma_g$ : Variância da gaussiana do sonar para os buracos.
5.  $C$ : Vetor de distâncias.
6. `starts_with` ∈ {"gap", "box"}: Estado inicial.
7.  $b$ : Número de bins para discretização.

As gaussianas  $\mathcal{N}_b(\mu_b, \sigma_b)$  e  $\mathcal{N}_g(\mu_g, \sigma_g)$  são as distribuições de probabilidade para os erros e ruídos do sonar.

O vetor de distâncias  $C$  é um vetor onde cada entrada  $i$  representa a distância do  $i$ -ésimo objeto (caixa ou buraco) até o  $i + 1$ -ésimo objeto. Junto com `starts_with`, que indica se  $C$  começa com uma caixa ou buraco,  $C$  representa o ambiente unidimensional inteiro.

O argumento  $b$  é o número de bins para discretização. Supondo que  $C$  é dado em centímetros, então se  $b = \sum_{i=0}^{|C|-1} C_i$ , então cada bin terá tamanho de 1 centímetro.

A função retorna três objetos:

1.  $M$ : Mapa de tamanho  $b$ , com  $M_i = 0$  se a célula  $i$  é buraco e  $M_i = 1$  se é caixa.
2.  $(\mathcal{N}_g, \mathcal{N}_b)$ : Par ordenado com as gaussianas relevantes.
3.  $\bar{d}$ : Tamanho de cada célula na mesma unidade dada em  $C$ .

A função `new_config` então cria um mapa do espaço discretizado e cria as gaussianas para o sensor. No código original temos:

```
C, N, d = new_config(15, 1, 10, 1, [5, 10, 15, 10, 20, 10, 30, 10, 15, 10,
20, 10, 5], bin_size=100)
```

Após criado o mapa do espaço, devemos criar a distribuição de probabilidade  $P(X)$  inicial. Para isso, usamos a função `gen_init_pdist`, que toma como argumentos:

1.  $n$ : Número de valores possíveis para  $X$ , ou seja, número total de células.
2.  $\mu$ : Média para gaussiana.
3.  $\sigma$ : Variância para gaussiana.
4.  $s$ : Tamanho de amostras para gerar a distribuição.
5.  $u$ : Booleana que indica se a distribuição é uniforme.

A função gera ou uma uniforme de  $\mathcal{U}(0, n)$  ou uma gaussiana  $\mathcal{N}(\mu, \sigma)$ . Se  $u$  for `True`, então ignora-se todos os argumentos menos  $n$ . Caso contrário, serão geradas  $s$  amostras de uma gaussiana  $\mathcal{N}(\mu, \sigma)$  para formar a distribuição inicial.

O valor de retorno da função é um vetor  $P$  distribuição de probabilidade com  $n$  entradas. No código original temos:

```
# Uniform initial belief.
U = gen_init_pdist(len(C), uniform=True)
# Gaussian initial belief centered on second box.
G1 = gen_init_pdist(len(C), mu=20, sigma=math.sqrt(40))
# Gaussian initial belief centered on fourth box.
G2 = gen_init_pdist(len(C), mu=60, sigma=math.sqrt(40))
```

Finalmente, criamos um objeto da classe `Map`. O construtor toma os valores de retorno das funções acima e mais dois argumentos.

1.  $C$ : Mapa das células.
2.  $N$ : Par de gaussianas dos sensores.
3.  $d$ : Tamanho real de cada célula.
4.  $P$ : Distribuição de probabilidade inicial  $P(X)$ .
5.  $p$ : Precisão das matrizes pré-computadas.
6.  $R$ : Objeto da classe `Range`.

Quando pré-computamos as matrizes de probabilidade, consideramos a área das gaussianas entre  $[-p\sigma, +p\sigma]$ , onde  $\sigma$  é o desvio padrão. Então com  $p = 3$ , cobrimos 99.7% da distribuição.

A classe `Range` representa quais são os possíveis movimentos do robô. Além disso, também limita o número de passos pré-computados e define a variância da gaussiana para a probabilidade de ação  $P(X' = x' | X = x, u)$ . O construtor de `Range` recebe:

1.  $M$ : Possíveis movimentações.
2.  $B$ : Bounds para movimentação.
3.  $\sigma$ : Variância para gaussiana da probabilidade de ação.

Portanto, temos no código original:

```
M = Map(C, N, d, G1, 3, Range([-1, 0, 1], [-25, 0, 25], 4))
```

O último argumento modificável é a posição inicial do robô. A função `start` tem como terceiro argumento um número no intervalo  $[0, m[$ , onde  $m$  é o número total de células. Este número indica onde o robô começará.

## 4 Observações

Quando o robô se move por um comando, o simulador decide a movimentação por meio da gaussiana  $\mathcal{N}(\mu = x + d, \sigma)$ , onde  $x$  é a célula em que o robô está e  $d$  é o quanto que o usuário deseja que robô ande. A variância  $\sigma$  é dada pela variância descrita na classe `Range`. A movimentação real é uma amostragem desta gaussiana, e portanto aleatória. Não sabia se esta movimentação devia ser determinística, então deixei aleatória para ficar mais geral. A linha em que se computa essa amostragem é 187:

```
dp = int(round(stats.norm(self.pos+u*d, self.R.s).rvs()))
self.pos = min(max(0, dp), self.m-1)
```

Se a movimentação deveria ser determinística, é possível alterar as duas linhas na seguinte:

```
self.pos += u*d
```

A variância talvez esteja muito alta. Neste caso, para mudar a variância, basta mudar a variância do `Range`.

```
Range([-1, 0, 1], [-50, 0, 50], sigma)
```

Se  $\sigma$  é muito baixo, talvez seja necessário diminuir os bounds definidos no `Range`. Isso é por que, se a variância é muito baixa, os valores pré-computados ficam muito próximo de zero, e portanto pode ser que a matriz seja dividida por zero.