

Controlador de IA para o SegWay

Trabalho 1 – INF01048

Utilização do simulador e implementação dos métodos de aprendizado

Professor: Bruno Castro da Silva – bsilva@inf.ufrgs.br

Monitor: Henrique Lopes – hplopes@inf.ufrgs.br

1. Introdução

Este documento apresenta instruções de como instalar o ambiente de desenvolvimento e o software de base necessários para desenvolver o Trabalho #1 da disciplina de INF01048. Você precisará: 1) instalar a linguagem Python; 2) obter uma cópia do código fonte básico utilizado para controle do SegWay, o qual você irá estender com suas funções de aprendizado; e 3) desenvolver as funções que implementam o seu método de aprendizado para controle do SegWay.

2. Como instalar o Python em seu sistema

Nota: O simulador SegWay não funciona na versão 3.5 do Python, nem no Windows nem no Linux. As instruções a seguir descrevem como instalar a versão correta do Python.

2.1 Instalando Python no Linux

Para rodar código Python em um sistema Linux, existem dois tipos de instalação: a virtual e a global. Cada uma delas requer passos diferentes, e cada uma destas alternativas oferece graus de liberdade e segurança diferentes, conforme discutido abaixo:

Instalação global: com passos mais simples, a instalação global é aquela que fica disponível em *todo* o sistema; ou seja, diferentes projetos que estejam em diferentes partes do seu sistema de arquivos têm acesso à mesma versão do Python, e aos mesmos pacotes instalados. Os passos de instalação, nesse caso, são mais simples, e podem ser uma opção para aqueles que não trabalham extensivamente com a linguagem. Em contrapartida, esse tipo de instalação não é recomendável para aqueles que trabalham com múltiplos projetos ao mesmo tempo, tendo em vista que os projetos podem requerer versões diferente do Python, e os pacotes necessários a um projeto podem gerar conflitos com os pacotes necessários a outros projetos.

Passos para instalação: versões mais recentes de Debian e Ubuntu já vêm com o Python instalado nativamente. Para verificar se a linguagem consta no sistema, rode o seguinte comando no terminal:

```
$ python --version
```

Se a linguagem Python já estiver instalada, o comando retornará sua versão. Caso contrário, é possível instalá-la com os seguintes comandos:

```
$ sudo apt-get install Python2.7 (distribuições Debian/Ubuntu)
```

```
$ sudo yum install python (distribuições Red Hat/RHEL/CentOS)
```

Se a instalação tiver sucesso, o comando que verifica a versão da linguagem retornará algo como:

```
$ python --version
```

```
$ python 2.7.6
```

Instalação virtual: na instalação virtual se utiliza um programa chamado *virtualenv* para criar, na pasta do projeto, uma pasta contendo os binários de uma versão específica do Python (a ser especificada pelo usuário). Essa instalação só ficará disponível para uso no projeto da disciplina. A instalação do *virtualenv* também é possível via apt-get/yum:

```
$ sudo apt-get install python-virtualenv
```

```
$ sudo yum install python-virtualenv
```

Após instalar o *virtualenv*, a criação de um ambiente virtual Python é feita através do seguinte comando:

```
path/to/project:$ virtualenv <nome do ambiente>
```

onde "path/to/project" descreve a pasta na qual o seu projeto está sendo desenvolvido. Por exemplo, o comando:

```
$ virtualenv env
```

cria uma pasta chamada "env" na pasta local onde o projeto está localizado. Para **ativar** o ambiente (e, assim, dizer para o sistema onde pacotes e bibliotecas Python devem ser instalados), basta executar o comando:

```
$ source env/bin/activate
```

Após isso, todos pacotes e bibliotecas instalados serão colocados na pasta "env" e estarão disponíveis para uso cada vez que o ambiente criado anteriormente for ativado através do comando *activate*. Note que o comando de ativação deve ser repetido cada vez que um terminal for aberto.

2.2 Instalando Python no Windows

Instalação global: Baixe a versão 2.7 do Python no site <https://www.python.org/downloads/> e adicione o local de instalação (por exemplo, C:\Python27) à variável PATH. Para fazer isso, vá em **[[Painel de Controle > Sistema > Alterar as variáveis de ambiente > Variáveis de ambiente]]** e troque o valor da variável PATH para:

```
valor_antigo_da_PATH;C:\Python27;C:\Python27\Scripts
```

O exemplo acima assume que os arquivos referentes à linguagem foram instalados em C:\Python27.

2.3 Instalando os pacotes necessários para o simulador

Após instalar o Python, precisamos instalar alguns pacotes e bibliotecas necessárias para a execução do simulador do SegWay. A ferramenta de instalação de pacotes Python é chamada *pip*. O *pip* já vem incluso na instalação do Python desde a versão 2.7. Caso ele não esteja instalado, entretanto, rode os seguintes comandos (no Linux):

```
$ sudo apt-get install python-pip
$ sudo yum install python-pip
```

O simulador do SegWay necessita de três pacotes/bibliotecas Python:

- 1) **pyglet** (responsável pela interface do jogo);
- 2) **pymunk** (simulador de física, necessário para calcular o impulso/deslocamento da roda e da vareta);
- 3) **numpy** (para operações de álgebra linear, tal como a multiplicação de vetores possivelmente necessária para implementar o método `Controller.take_action()`).

Todos os três pacotes podem ser instalados através do comando

```
pip install <nome_do_pacote>.
```

3. Copiando o repositório do simulador do SegWay

Antes de mais nada, crie uma conta no site do Bitbucket (bitbucket.org), caso ainda não possua uma :-)

O código base que todos os alunos utilizarão para desenvolver o trabalho está disponível em um repositório nos servidores do Bitbucket. Um repositório nada mais é do que um local no qual um código fonte encontra-se salvo, e que armazena o histórico completo de todas modificações feitas ao código. Como o histórico de todas as modificações feitas ao código é salvo pelo sistema, pode-se ter um controle maior do estado atual do projeto (comparado com versões anteriores), assim como controle de quais mudanças e atualizações ao código foram feitas por qual desenvolvedor. O fato de que o repositório guarda o histórico completo de desenvolvimento também permite que se analise versões antigas do programa, e que se desfça eventuais modificações que introduziram bugs. Este tipo de sistema de gerência de código fonte se chama *controle de versões*. O Bitbucket é um sistema online que armazena repositórios com estas funcionalidades. O repositório contendo o código básico do simulador, e que será estendido pelas duplas através do desenvolvimento de métodos de aprendizado, se encontra no Bitbucket.

Para que cada grupo possa ter sua própria **cópia privada** do código do simulador SegWay disponibilizado no Bitbucket (e que será modificada durante a implementação do método de aprendizado da dupla) é preciso que cada grupo copie, no Bitbucket, o repositório principal do simulador através da opção "*fork*". A opção de *fork* está visível no canto superior da tela da página do repositório no site Bitbucket: <https://bitbucket.org/henriqdp/segway>.

No momento desta cópia, cada dupla poderá dar um nome a sua própria cópia do projeto, e poderá também modificar as suas permissões de acesso. Ao executarem o *fork*, **marquem a *checkbox* que torna o repositório privado (isso é importante para evitar que outros grupos copiem a sua solução!)**:



The screenshot shows the Bitbucket 'Fork' interface for the repository 'henriqdp / segway'. It includes a 'Name' field with the value 'segway', an 'Access level' section with a checked checkbox for 'This is a private repository', a link for 'Advanced settings', and two buttons at the bottom: 'Fork repository' and 'Cancel'.

Fork henriqdp / segway

Name* segway

Access level ☒ This is a private repository

> Advanced settings

Fork repository Cancel

Após feito o *fork*, cada dupla terá, no Bitbucket, uma cópia particular do repositório contendo o código base do simulador. Depois de criada esta cópia, por favor adicionem os usuários *henriqdp* (o monitor) e *bsilvapo* (o professor) como administradores do seu repositório (através da opção "*invite users to this repo*"). Isso permitirá com que o monitor e o professor da disciplina analisem o progresso sendo feito pela dupla durante o desenvolvimento do trabalho.

Após a criação da cópia do projeto no site do Bitbucket, é necessário baixar, do Bitbucket, o código fonte para a máquina local na qual o trabalho será desenvolvido. Isso vai permitir com que um aluno altere o código fonte localmente, no seu computador, como de costuma durante o desenvolvimento de um programa. Permitirá também com que o código desenvolvido localmente por um dos alunos possa ser enviado de volta para o servidor Bitbucket, a fim de que possa ser acessado pelo outro aluno da dupla—o qual pode estar desenvolvendo *outras* partes do software em seu próprio computador. Esse tipo de sincronização das contribuições feitas por cada aluno através do site Bitbucket evita com que (p.ex.) o aluno 1 tenha que enviar um email para o aluno 2 contendo o novo código fonte, forçando o aluno 2 a manualmente identificar quais partes do código foram alteradas pelo aluno 1, e copiá-las para sua própria versão do código.

Para gerenciar o seu repositório do Bitbucket, utilizaremos o programa **GIT**. O GIT é usado para 1) baixar o código fonte do site Bitbucket para uma máquina local, na qual o aluno irá desenvolver seu código; 2) para salvar alterações feitas ao código no histórico de desenvolvimento do projeto; e 3) para enviar a versão do código sendo desenvolvida por um aluno, em seu computador, de volta ao servidor Bitbucket, a fim de que possa ser acessada pelo outro aluno e pelo professor e monitor.

Para instalar o GIT, siga as instruções em:

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>.

Os principais comandos GIT que vocês precisarão utilizar são os seguintes:

`$ git clone <endereço do repositório>` - faz uma cópia do código no repositório Bitbucket para a máquina de desenvolvimento. Esse passo é **executado apenas uma vez**, ao se baixar o código do Bitbucket pela primeira vez para um computador. Como o repositório de cada dupla (criado e nomeado através do processo de *fork*, descrito anteriormente) é privado, o usuário precisará se identificar para ter acesso através do *clone*. O endereço do repositório criado pela dupla durante o *fork* possui a seguinte forma:

`https://<user_no_bitbucket>@bitbucket.org/user_dono_do_repo/nome_do_repo.git`.

Alternativamente, vocês podem descobrir o endereço de seu repositório entrando no site do Bitbucket, acessando a sua cópia do repositório, e copiando o seu endereço no canto superior direito da dela; por exemplo:



Depois de clonar o repositório, vocês terão uma cópia completa do código fonte em sua máquina local, a qual poderão modificar normalmente. Sempre que forem adicionados *novos* arquivos Python ao seu código, você precisará adicioná-lo ao projeto. Isso é feito através do comando *git add*:

```
$ git add <arquivos_a_serem_adicionados_ao_projeto>.
```

Depois de se adicionar um ou mais arquivos ao projeto, será possível usar comandos GIT para salvar versões específicas daquele arquivo (ou do projeto completo) no histórico de desenvolvimento do repositório. Isso permitirá, conforme dito anteriormente, que seja feito o controle das diferentes versões do programa, conforme vai sendo desenvolvido. Para salvar a versão atual do código fonte no repositório local (criado através do comando *git clone*) utilize o comando *commit*:

```
$ git commit -m "mensagem"
```

 - esse comando salva as mudanças feitas ao código fonte no repositório local do projeto. A *mensagem* serve para explicar quais modificações foram feitas ao código; por exemplo, quais novas funcionalidades foram incluídas no projeto, ou quais bugs foram corrigidos.

```
$ git status
```

 - este comando compara o código fonte atual na pasta do projeto com a última versão gravada no repositório do projeto. Isto é, caso você altere arquivos depois de tê-los gravado no repositório (através do comando *commit*), o comando *status* irá informar exatamente quais arquivos foram modificados.

Todos os comandos acima atualizam apenas o repositório local do projeto, em um computador específico no qual o programa está sendo desenvolvido—por exemplo, no computador de um dos alunos da dupla. A fim de enviar estas modificações de volta para o servidor Bitbucket, de modo que o outro membro da dupla possa baixá-las no seu próprio computador (através do comando *pull*, descrito mais adiante), e também para que o monitor e professor possam acompanhar o progresso do desenvolvimento, é necessário utilizar o comando *git push*:

\$ `git push origin master` - este comando envia a versão atual do código (gravada no repositório local do projeto através de comandos *commit*) ao servidor do Bitbucket. Depois de feito um *push*, o histórico de todos os *commits* feitos poderá ser visualizado na página do repositório, na área de *commits*.

\$ `git pull` - suponha que o aluno 1 do grupo esteja desenvolvendo parte do programa em seu computador, e que tenha enviado sua nova versão do código para o servidor Bitbucket através do comando *push*. O aluno 2 poderá então baixar estas alterações e automaticamente integrá-las a sua versão do código, gravada em sua máquina local. O comando *pull* baixa do Bitbucket o código enviado pelo aluno 1, compara-o com o código local do aluno 2, verifica quais são as novidades, e integra todas modificações feitas pelo aluno 1 no código do aluno 2, de forma que ambos os códigos fiquem sincronizados e que aluno 2 possa ter acesso às novas contribuições do aluno 1.

3.1 Exemplo de uso

Um exemplo de uso do GIT seria o seguinte: imaginem que cada aluno da dupla baixou o código fonte do Bitbucket utilizando o comando *git clone*. Cada aluno, portanto, tem agora uma cópia do projeto em sua própria máquina local. Imagine que o aluno 1 fez mudanças na função **compute_features()** (no arquivo *AI_controller/controller.py*), e que tais mudanças são necessárias para que o aluno 2 possa prosseguir com sua parte do trabalho. O primeiro aluno executaria os seguintes comandos:

```
$ git add AI_controller/controller.py
```

```
$ git commit -m "Realizadas modificações na função compute_features pelo motivo blablabla etc etc"
```

```
$ git push origin master
```

Após a execução destes comandos, todas as modificações feitas ao arquivo *AI_controller/controller.py* pelo aluno 1 seriam gravadas no repositório local (comando *commit*) e depois enviadas ao site do Bitbucket (comando *push*). O aluno 2, para ter acesso a estas modificações em sua própria máquina, executaria então (em seu próprio computador) o comando:

```
$ git pull
```

Este comando iria baixar todas as alterações feitas pelo aluno 1 (e enviadas ao Bitbucket) e integrá-las ao código local do aluno 2, sincronizando o código de ambos alunos.

O método de desenvolvimento descrito acima pode parecer complicado a primeira vista, mas sistemas de controle de versão deste tipo (tal como o GIT) são amplamente tanto em empresas quanto academicamente, a fim de permitir o controle do estado do software em diferentes momentos de seu desenvolvimento, e também para facilitar o compartilhamento de funções desenvolvidas por um programador com todo o resto do time de desenvolvimento. Para mais informações, um bom ponto de partida para estudar o sistema GIT é o guia já citado anteriormente, e também o seguinte manual:

<https://www.atlassian.com/git/tutorials/what-is-version-control/benefits-of-version-control>.

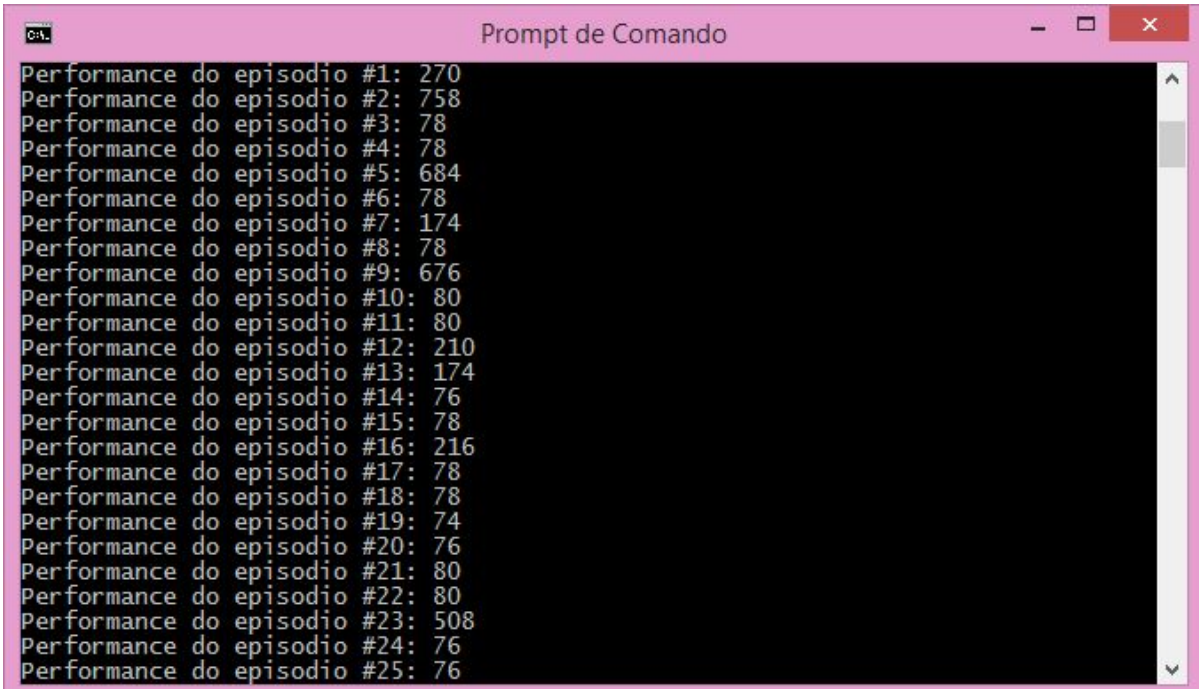
4. Modo de usar o simulador do SegWay

Há 2 modos diferentes de utilizar o simulador do SegWay: o modo **learn** e o modo **evaluate**.

O modo **learn** é usado para executar o método de aprendizado implementado pela dupla, a fim de que se aprenda um controlador eficaz para o SegWay. Esse modo não aciona a visualização gráfica do SegWay sendo controlado, a fim de tornar a execução do aprendizado mais rápida. No modo **learn**, o SegWay é inicializado em uma configuração aleatória (uma posição e inclinação aleatória), e então utiliza-se o controlador desenvolvido pela dupla para tentar equilibrá-lo. A performance do controlador é medida (isto é, mede-se por quanto tempo ele conseguiu manter o SegWay equilibrado), e tal informação é passada ao método de aprendizado implementado pela dupla (na função **Controller.update**; mais detalhes adiante). Este método poderá, então, alterar os parâmetros $\theta_1 \dots \theta_N$ do controlador a fim de atualizá-lo e melhorar sua performance. Os parâmetros iniciais a serem utilizados pelo controlador, no início do processo de aprendizado, podem ser informados através de um arquivo de texto; caso nenhum arquivo seja informado, pesos iniciais serão gerados automaticamente. Exemplo de uso:

```
$ python __init__.py learn pesosiniciais.txt
```

O código acima produzirá uma saída parecida com essa:



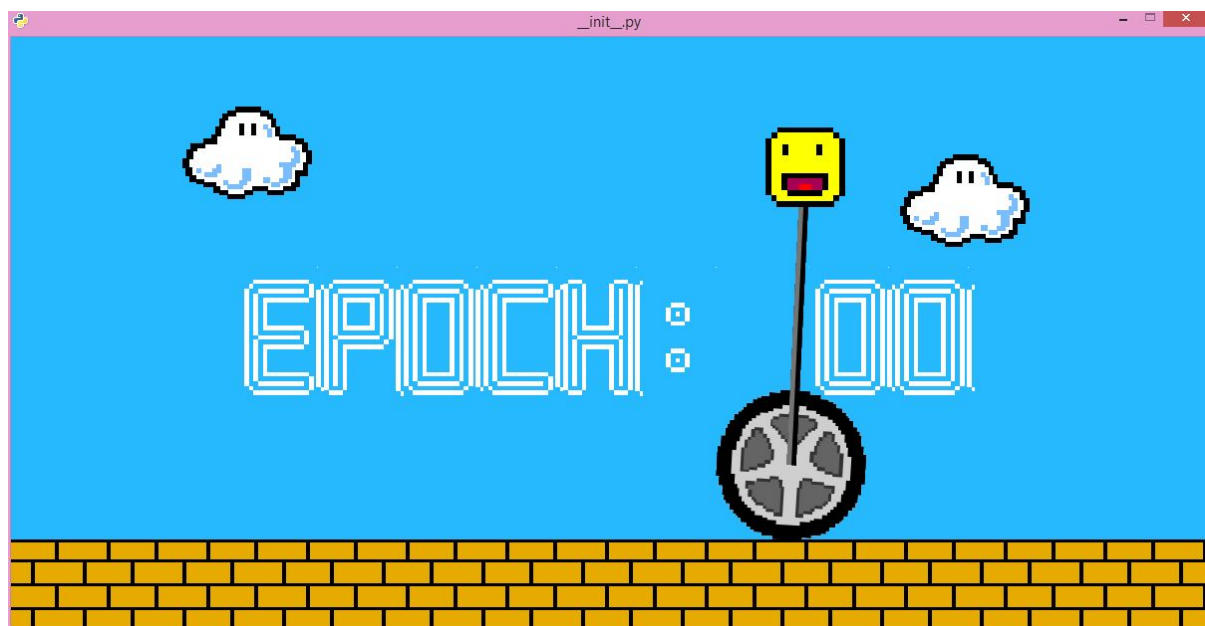
```
Performance do episódio #1: 270
Performance do episódio #2: 758
Performance do episódio #3: 78
Performance do episódio #4: 78
Performance do episódio #5: 684
Performance do episódio #6: 78
Performance do episódio #7: 174
Performance do episódio #8: 78
Performance do episódio #9: 676
Performance do episódio #10: 80
Performance do episódio #11: 80
Performance do episódio #12: 210
Performance do episódio #13: 174
Performance do episódio #14: 76
Performance do episódio #15: 78
Performance do episódio #16: 216
Performance do episódio #17: 78
Performance do episódio #18: 78
Performance do episódio #19: 74
Performance do episódio #20: 76
Performance do episódio #21: 80
Performance do episódio #22: 80
Performance do episódio #23: 508
Performance do episódio #24: 76
Performance do episódio #25: 76
```

O processo de aprendizado utiliza o conceito de um **episódio de treinamento**: em um episódio, o SegWay é inicializado em uma configuração aleatória, e o controlador atual (com parâmetros $\theta_1 \dots \theta_N$) é utilizado para controlá-lo até que ele caia ou saia da tela. Quando isso ocorrer, sua performance (tempo até a falha) é calculado e informado para a função de aprendizado, a qual poderá atualizar os parâmetros. Neste momento, diz-se que o episódio de treinamento acabou. A seguir, um novo episódio de treinamento é iniciado, e o processo se repete. O modo **learn** executa episódios de treinamento indefinidamente, até que o usuário interrompa o processo de aprendizado com Ctrl-C. A cada dez episódios, os parâmetros atuais aprendidos do controlador são gravados na pasta “params/”.

O modo **evaluate**, por sua vez, permite que o aluno teste e visualize o que acontece quando o SegWay é controlado através de um determinado conjunto de parâmetros, descobertos durante o processo de aprendizado. Nesse modo, o programa roda a interface gráfica do simulador, mostrando o SegWay tomando as ações de equilíbrio. Como esse modo é utilizado para verificar o comportamento do SegWay resultante de um conjunto de parâmetros, um arquivo texto com os parâmetros que se deseja testar deve ser informado. Exemplo de uso:

```
$ python __init__.py evaluate pesos.txt
```

O programa, nesse caso, vai alimentar o controlador do SegWay com os parâmetros informados e rodar/exibir o simulador, conforme o exemplo abaixo:



Note que no modo **evaluate**, os parâmetros do controlador *não* são atualizados ao final de cada episódio. Este modo apenas avalia um controlador aprendido pelo modo **learn**. A fim de testar a performance do seu controlador frente a perturbações externas, você pode também manualmente "empurrar" a cabeça do usuário do SegWay para a direita ou esquerda (utilizando, para isso, as teclas A e D). Você pode também testar a performance do seu controlador em um cenário com vento (i.e., com uma força constante sendo aplicada em uma direção aleatória). O vento pode ser ligado/desligado através da tecla W. A cada vez que se reinicia o vento, ele é alterado para uma nova direção aleatória.

5. O que deve ser feito no trabalho

O simulador do SegWay funciona da seguinte maneira: a cada vez que a tela é redeseenhada (60 vezes em um segundo), uma função é chamada que verifica o estado do SegWay. O estado do SegWay consiste em três informações: sua coordenada x , ângulo do SegWay, e sua velocidade angular. O simulador então pede ao controlador que tome uma ação correspondente.. As ações possíveis são para aplicar uma força (na roda) para a esquerda, para a direita, ou nenhuma força. A função que decide qual ação executar, com base no estado do SegWay, se chama **take_action()**, e faz parte da classe **Learner** disponível no arquivo **AI_controller/learner.py**. Tal função deverá, para escolher a ação apropriada em um estado, usar um conjunto de *features*. Estas *features* deverão ser calculadas pela função **compute_features()**, presente no mesmo arquivo. Concretamente, tal função deve utilizar as informações básicas do estado atual do SegWay para calcular um conjunto de *features* expandidas de estado, as quais combinam uma ou mais das informações básicas de estado; por exemplo, um conjunto de *features* expandidas poderia ser o seguinte:

$$\{ (coord_x), (coord_x*angulo), (angulo), (vel_angular), (vel_angular*angulo^2) \}.$$

Quais *features* específicas serão utilizadas pelo controlador faz parte do que as duplas irão investigar. A função **take_action()** deverá utilizar os parâmetros $\theta_1 \dots \theta_N$ do controlador e as *features* calculadas por **compute_features()** para calcular a função de preferência Q para cada uma das três ações disponíveis. Deverá, então, verificar qual ação possui a maior preferência e retorná-la. Note que se a sua função **compute_features()** calcular K *features*, então seu controlador terá um total de $3*K$ *features*, uma vez que cada uma das três funções Q utiliza K *features* no cálculo da preferência. Mais especificamente:

- 1) O vetor de parâmetros utilizado pelo seu controlador deverá ter, exatamente, três vezes o tamanho do vetor retornado por **compute_features()**.
- 2) O vetor de parâmetros deverá dividido em três partes, cada uma com o tamanho exato do vetor retornado por **compute_features()**.
- 3) A primeira parte destas três será multiplicada pelo vetor de *features* calculadas com base no estado atual, e o valor resultante será referente à preferência pela ação de impulsionar a roda para a esquerda.
- 4) A segunda parte destas três será multiplicada pelo vetor de *features* calculadas com base no estado atual, e o valor resultante será referente à preferência pela ação de não impulsionar a roda.

- 5) A terceira parte destas três será multiplicada pelo vetor de *features* calculadas com base no estado atual, e o valor resultante será referente à preferência ação de impulsionar a roda para a direita.

Após verificar-se qual ação tem maior preferência, o programa deve:

- 1) retornar -1 caso se queira impulsionar a roda para a esquerda;
- 2) retornar 0 caso não se queira executar nenhum impulso;
- 3) retornar +1 caso se queira impulsionar a roda para a direita.

O simulador repetidamente solicitará à **take_action()** que informe a ação apropriada dado o estado atual do SegWay, até que o SegWay caia, saia da tela, ou um *timeout* (número máximo de passos) seja alcançado. Nesse momento, o episódio atual de treinamento acaba, e a performance daquele episódio é calculada e informada ao método **Controller.update**. Tal método poderá, então, alterar os parâmetros $\theta_1 \dots \theta_N$ do controlador a fim de atualizá-lo e melhorar sua performance.

Em resumo, você deverá **obrigatoriamente** implementar três funções:

- a) **compute_features()**;
- b) **take_action()**;
- c) **update()**.

Todas essas funções devem ser colocadas no arquivo **AI_controller/controller.py**.

6. Notas finais

- Lembre-se que o simulador do SegWay **não** funciona com a versão 3.5 do Python. Certifiquem-se de instalar a versão 2.7, conforme descrito nas instruções acima.
- Você pode a imagem do "rosto" do usuário conduzindo o SegWay. Para isso, basta substituir a imagem original (smiley.png, na pasta "resources/") pela imagem que se quer utilizar. Note que o programa não redimensiona a imagem; certifiquem-se de substituí-la por uma imagem de dimensões semelhantes.