Controlador de IA para o SegWay
Trabalho 1 — INF01048

Dicas para formulação matemática do problema de aprendizado

Prof. Bruno Castro da Silva
bsilva@inf.ufrgs.br

1 Objetivo

O objetivo deste trabalho é utilizar técnicas de Inteligência Artificial para desenvolver um controlador capaz de equilibrar uma versão simulada do SegWay. O programa de IA a ser desenvolvido deverá ser capaz: *1)* de manter o equilíbrio do SegWay em diversas condições (diversos ângulos de inclinação, por exemplo); *2)* dinamicamente recuperar o seu equilíbrio caso empurrado ou perturbado; e *3)* se manter de pé em ambientes com níveis crescentes de dificuldade.

As técnicas de IA a serem utilizadas para resolução deste trabalho ficam a critério de cada dupla, mas deverão envolver os métodos de Busca Local estudados na disciplina. Por exemplo, Hill Climbing, Simulated Annealing, Método do Gradiente Ascendente, Busca em Feixe Local, Algoritmos Genéticos, etc. Podem ser utilizadas variantes destes métodos, caso os alunos queiram pesquisar por conta própria e achem apropriado.

O controlador a ser desenvolvido **não** pode ser manualmente codificado; isto é, não é permitido que se programe manualmente um software que verifica a condição atual do SegWay e determina a melhor ação a ser executada. O objetivo é que tal controlador seja *aprendido* de forma automática utilizando-se algum dos métodos de aprendizado estudados na disciplina.

2 O SegWay Real

O SegWay consiste em um veículo de duas rodas capaz de transportar uma pessoa e de se auto-equilibrar (Figura 1). Ele é movido a bateria e é atualmente produzido pela empresa SegWay Inc. O SegWay possui 5 giroscópios e 2 acelerômetros, responsáveis por medir informações relativas à posição do corpo de seu condutor, assim como a velocidade do equipamento e condições gerais do terreno. Com base nestas informações, um controlador embarcado do SegWay determina os comandos que devem ser enviados para cada roda, a fim de que o SegWay se mantenha equilibrado e mova-se na direção indicada pelo condutor. Para que o Segway avance, o condutor apenas precisa de se inclinar para a frente, e para que recue, é necessária apenas a inclinação para trás.

Figure 1: O SegWay sendo demonstrado por seu inventor, Dean Kamen.

3 O SegWay Simulado

Neste trabalho iramos utilizar uma versão simplificada do SegWay como plataforma de estudo para as técnicas estudadas nesta disciplina. O modelo simplificado que utilizaremos (Figura 2) consiste em um equipamento bidimensional sobre o qual um personagem de peso constante deseja se equilibrar. É possível enviar três tipos de comando para a roda: comando para que a roda mova-se para a esquerda (E), mova-se para a direita (D), ou não se mova ($\varnothing$). O controlador do SegWay envia 60 comandos por segundo para as rodas. Dado que o objetivo é permitir que o SegWay mantenha-se equilibrado, um controlador eficaz leva em conta o estado atual do SegWay a fim de determinar quais ações/comandos são mais apropriados. Se o SegWay estiver caindo para a direita, por exemplo, deve ser aplicado um comando de mover a roda para a direita, a fim de que a base do SegWay se mova na direção da queda e a evite (veja Figura 3 para um exemplo). Neste trabalho, assumimos que o estado s_t do SegWay no tempo t consiste em uma tupla de números $s_t = [\alpha, \dot{\alpha}, x]$ onde α corresponde ao ângulo do SegWay em relação ao chão, $\dot{\alpha}$ corresponde à velocidade angular o

SegWay (a taxa de mudança de seu ângulo), e x corresponde a sua localização horizontal no chão. Note que, ao contrário do que ocorre em um SegWay real, o objetivo aqui não envolve a locomoção em uma direção indicada pelo condutor; apenas desejamos controlar o equipamento de forma a permitir que o condutor permaneça equilibrado.

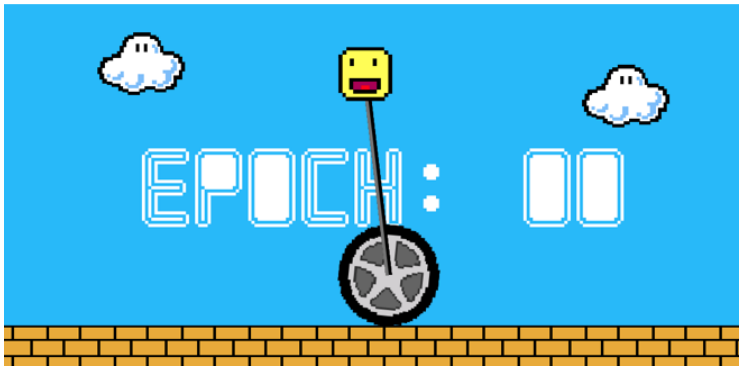

Figure 2: O SegWay simulado.

4 Treinamento de um Controlador

O objetivo deste trabalho é utilizar técnicas de Busca Local estudadas em aula para aprender um controlador para o SegWay. O controlador a ser desenvolvido será uma função $\pi_\theta(s_t)$, a qual recebe como entrada o estado do SegWay no tempo t e devolve uma ação dentre $\{D, E, \varnothing\}$. Ou seja, a função π_θ deverá receber informação a respeito do ângulo, velocidade angular e posições atuais do SegWay e determinar uma ação apropriada para permitir com que ele se mantenha equilibrado.

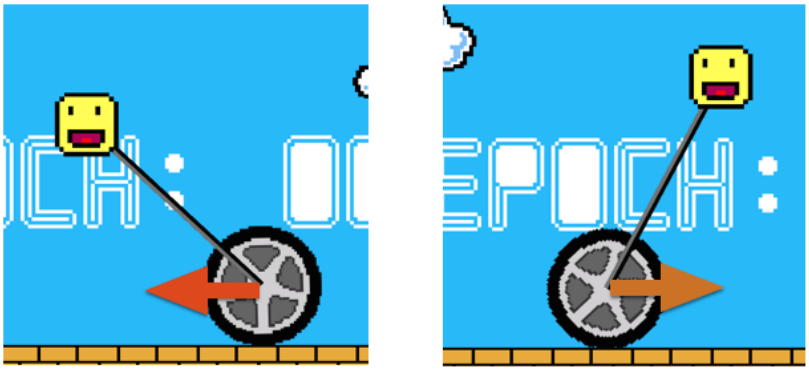

Figure 3: Ações apropriadas para permitir o equilíbrio dinâmico do robô.

Perceba que o controlador π_θ, descrito acima, depende de parâmetros θ. Neste trabalho, definimos θ como um vetor de N números reais: $\theta = [\theta_1, \ldots, \theta_N]$. Dependendo dos valores destes parâmetros, o controlador π_θ irá retornar diferentes ações para cada estado. O objetivo do trabalho é aprender os valores dos parâmetros que, quando utilizados pelo controlador π_θ, permitem com que o SegWay se mantenha equilibrado por mais tempo. Note que isso corresponde a um problema de otimização: desejamos maximizar uma função Valor(θ), a qual avalia a qualidade de um controlador π parametrizado por θ de forma que, quanto mais tempo o controlador π_θ conseguir equilibrar o SegWay, maior será Valor(θ). Portanto, qualquer processo de busca local que identifique os parâmetros θ que maximizam Valor(θ) pode ser utilizado.

A fim de permitir o aprendizado deste controlador, precisamos, primeiramente, decidir como a função π_θ será implementada computacionalmente. Uma maneira simples de implementá-la é a seguinte. Seja $Q_\theta(s_t, D)$ uma função que retorna um valor numérico denotando a *preferência* do SegWay por executar a ação D, quando este se encontra no estado s_t; quanto maior $Q_\theta(s_t, D)$, maior será a preferência do SegWay pela execução desta ação, ou seja, mais ele julga a ação como sendo apropriada para manter o equilíbrio no estado s_t. Se $s_t = [110, 20, 0]$, por exemplo, o SegWay encontra-se com uma inclinação de 110 graus e caindo para a direita a uma taxa de 20 graus por passo de simulação. A melhor ação neste caso seria mover a roda para a direita; isso significa que $Q_\theta(s_t, D)$ deverá ser um valor positivo grande. Podemos definir, de maneira semelhante, também funções de preferência para as ações E e $\varnothing$: $Q_\theta(s_t, E)$ e $Q(s_t, \varnothing)$, respectivamente. Note que se tivermos acesso aos valores de preferência corretos para cada uma das três possíveis ações, quando em um estado

s_t, o controlador $\pi_\theta(s_t)$ pode simplesmente escolher a ação com maior valor de preferência:

$$\pi_\theta(s_t) = \arg \max_{a \in \{D,E,\varnothing\}} Q_\theta(s_t, a). \tag{1}$$

uma vez que isso, por definição, significa que ele estaria escolhendo as ações mais apropriadas para cada estado s_t.

A pergunta natural, agora, é como iremos representar as funções de preferência Q_θ. Uma possibilidade é defini-las simplesmente como uma função linear do ângulo do SegWay:

$$\begin{aligned} Q_\theta(s_t, D) \equiv Q_\theta(\alpha, \dot{\alpha}, x, D) &= (\theta_1 \times 1) + (\theta_2 \times \alpha) \\ Q_\theta(s_t, E) \equiv Q_\theta(\alpha, \dot{\alpha}, x, E) &= (\theta_3 \times 1) + (\theta_4 \times \alpha) \\ Q_\theta(s_t, \varnothing) \equiv Q_\theta(\alpha, \dot{\alpha}, x, \varnothing) &= (\theta_5 \times 1) + (\theta_6 \times \alpha) \end{aligned}$$

onde os coeficientes destas funções lineares correspondem ao parâmetros θ a serem aprendidos. Ao se ajustar os $N = 6$ parâmetros $\theta_1, \ldots, \theta_6$, define-se diferentes funções de preferência pelas ações D, E e $\varnothing$, o que implica que o controlador irá controlar o SegWay de maneiras diferentes. Algumas destas maneiras de reagir ao estado atual podem ser mais eficazes para permitir o equilíbrio do SegWay, outras nem tanto. Parâmetros que resultem em preferências corretas implicam que o SegWay irá escolher ações apropriadas para que, dado seu estado s_t atual, consiga manter o equilíbrio. Qualquer processo de busca local que maximize Valor(θ), portanto, corresponde ao aprendizado das preferências Q_θ de forma que as ações mais apropriadas em cada estado tenham maior preferência. Como exemplo, suponha que o método de busca está atualmente considerando o que aconteceria se $\theta_1 = 120$, $\theta_2 = 140$, $\theta_3 = -147$, $\theta_4 = -21$, $\theta_5 = 54$, $\theta_6 = 6$. Nesse caso, poderíamos plotar as funções Q resultantes e visualizar como as preferências por cada ação mudam conforme o ângulo α do SegWay:

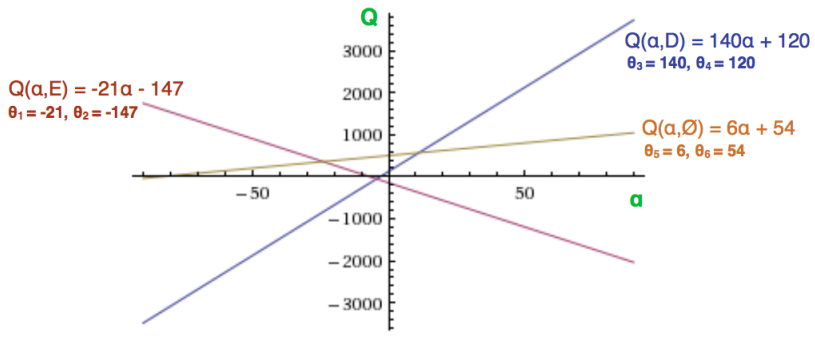

Figure 4: Funções de preferência Q implementadas como funções apenas do ângulo α do SegWay.

Na Figura 4, vemos que para ângulos entre -90 e aproximadamente -35, a ação com maior preferência é E. Para ângulos entre -35 e aproximadamente +15, a ação com maior preferência é $\varnothing$; e a partir de +15, a ação de maior preferência é D. Verifica-se, portanto, que a alteração dos parâmetros $\theta_1, \ldots, \theta_6$ gera diferentes funções lineares de preferência por E, $\varnothing$, e D, o que resulta em diferentes ações dependendo do ângulo do SegWay. Alterando-se tais parâmetros, portanto, podemos obter controladores $\pi_\theta(s_t)$ que tentam equilibrar o SegWay de maneiras diferentes—algumas com maior performance Valor(θ), outras com performance pior. O problema de busca local será justamente encontrar os parâmetros com maior performance.

A representação acima possui uma limitação importante: o controlador do SegWay leva em conta apenas o ângulo α do SegWay a fim de determinar a melhor ação, quando em um estado s_t. Podemos imaginar, intuitivamente, que a ação ótima provavelmente também dependerá de $\dot{\alpha}$ e x. Poderíamos considerar então a seguinte representação (mais completa) das funções de preferência, as quais levam em conta todas as três features básicas do estado: α, $\dot{\alpha}$ e x. Especificamente, a representação mais completa poderia consistir na soma ponderada das três *features* de estado do SegWay:

$$\begin{aligned} Q_\theta(s_t, D) \equiv Q_\theta(\alpha, \dot{\alpha}, x, D) &= (\theta_1 \times 1) + (\theta_2 \times \alpha) + (\theta_3 \times \dot{\alpha}) + (\theta_4 \times x) \\ Q_\theta(s_t, E) \equiv Q_\theta(\alpha, \dot{\alpha}, x, E) &= (\theta_5 \times 1) + (\theta_6 \times \alpha) + (\theta_7 \times \dot{\alpha}) + (\theta_8 \times x) \\ Q_\theta(s_t, \varnothing) \equiv Q_\theta(\alpha, \dot{\alpha}, x, \varnothing) &= (\theta_9 \times 1) + (\theta_{10} \times \alpha) + (\theta_{11} \times \dot{\alpha}) + (\theta_{12} \times x) \end{aligned}$$

Como antes, ao se ajustar os $N = 12$ parâmetros $\theta_1, \ldots, \theta_{12}$, define-se diferentes funções de preferência pelas ações D, E e $\varnothing$, o que implica que o controlador irá controlar o SegWay de maneiras diferentes; algumas destas maneiras de reagir ao estado atual podem ser mais eficazes para permitir o equilíbrio do SegWay, outras nem tanto. Parâmetros que resultem em preferências corretas implicam que o SegWay irá escolher ações apropriadas para que, dado seu estado s_t atual, consiga manter o equilíbrio. Qualquer processo de busca local que maximize Valor(θ), portanto, corresponde ao aprendizado das preferências Q_θ de forma que as ações mais apropriadas em cada estado tenham maior preferência.

Embora a representação acima seja possível, note que cada função Q_θ é **linear** em relação as *features* de estado, α, $\dot{\alpha}$ e x (i.e., cada função de preferência é uma soma ponderada direta das *features* de estado s_t). Funções lineares deste tipo permitem a representação de preferências simples; especificamente, apenas de funções de preferência que são "linhas retas"[1], conforme visto na Figura 4. Em alguns casos, um controlador eficaz pode ter preferências **não-lineares** em função do estado: a preferência pela ação de desligar o motor, $\varnothing$, por exemplo, pode ser baixa para ângulos α negativos, crescer quando o ângulo se aproxima se zero, e depois novamente decrescer quando o ângulo se torna positivo. Isso significa que a preferência $Q(s_t, \varnothing)$ não é uma "linha reta", que apenas cresce ou apenas decresce conforme o ângulo α muda, e sim uma parábola. A fim de permitir a representação de funções de preferência mais complexas, não necessariamente lineares em relação às features básicas de estado do SegWay, frequentemente se define as funções Q_θ em relação a um **conjunto expandido**

[1] Ou, nesse caso, planos no espaço tridimensional cujos eixos são α, $\dot{\alpha}$, e x.

$f(s_t)$ de *features* de estado. Um possível conjunto expandido de *features* $f(s_t)$ poderia ser, por exemplo, dado por polinômios de grau 3 das *features* de estado: $f(s_t) \equiv f(\alpha, \dot{\alpha}, x) = [1, \alpha, \alpha^2, \alpha^3, \dot{\alpha}, \dot{\alpha}^2, \dot{\alpha}^3, x, x^2, x^3]$. Nesse caso, teríamos $N = 10$ features usadas na definição de cada uma das preferências:

$$\begin{aligned}
Q_\theta(s_t, D) &= (\theta_1 \times 1) + (\theta_2 \times \alpha) + (\theta_3 \times \alpha^2) + (\theta_4 \times \alpha^3) + (\theta_5 \times \dot{\alpha}) + \\
&\quad (\theta_6 \times \dot{\alpha}^2) + (\theta_7 \times \dot{\alpha}^3) + (\theta_8 \times x) + (\theta_9 \times x^2) + (\theta_{10} \times x^3) \\
Q_\theta(s_t, E) &= (\theta_{11} \times 1) + (\theta_{12} \times \alpha) + (\theta_{13} \times \alpha^2) + (\theta_{14} \times \alpha^3) + (\theta_{15} \times \dot{\alpha}) + \\
&\quad (\theta_{16} \times \dot{\alpha}^2) + (\theta_{17} \times \dot{\alpha}^3) + (\theta_{18} \times x) + (\theta_{19} \times x^2) + (\theta_{20} \times x^3) \\
Q_\theta(s_t, \varnothing) &= (\theta_{21} \times 1) + (\theta_{22} \times \alpha) + (\theta_{23} \times \alpha^2) + (\theta_{24} \times \alpha^3) + (\theta_{25} \times \dot{\alpha}) + \\
&\quad (\theta_{26} \times \dot{\alpha}^2) + (\theta_{27} \times \dot{\alpha}^3) + (\theta_{28} \times x) + (\theta_{29} \times x^2) + (\theta_{30} \times x^3)
\end{aligned}$$

Note que como $f(s_t)$ mapeia o estado s_t para um conjunto expandido de 10 *features*, e como há 3 funções de preferência Q (uma para cada ação), teremos um total de $10 \times 3 = 30$ parâmetros θ, ao invés de apenas $N = 12$, como era o caso quando as funções de preferência eram definidas apenas em função das três *features* básicas de estado. Isso indica que, se quisermos representar funções de preferência mais complexas, pagaremos o preço através de um maior número de parâmetros que precisam ser aprendidos pelo processo de busca local, o que torno o aprendizado mais lento.

O conjunto expandido de features $f(s_t)$ descrito acima é apenas uma possibilidade, e não necessariamente resulta em boa performance. Vocês poderia considerar, também, outros tipos de *features*: $(\alpha \times \dot{\alpha})$, $(\alpha \times \dot{\alpha}^2)$, $(\alpha^2 \times x)$, etc. Nada impede, também, que se use *features* que não são polinômios: por exemplo, $\sin(\alpha \times x)$. Fica a critério de cada dupla determinar quais *features* expandidas tentar. A intuição por trás deste processo é a seguinte: a fim de que o controlador possa decidir a ação correta, possivelmente pode ter que levar em consideração *features* que **combinem** o ângulo do SegWay com sua posição x, uma vez que ambas informações são, de forma conjunta, relevantes à tomada de decisão. Isso poderia ser obtido através de uma *feature* do tipo $(\alpha \times x)$, ou também $(\alpha^2 \times x^3)$, etc. Esse tipo de combinação, quando necessária, indica que as informações de ângulo e de posição precisam ser levadas em conta *de forma conjunta* na hora de determinar uma ação. Caso a ação apropriada em um determinado valor de x **não** dependa fortemente da **combinação** específica de ângulo com posição, por outro lado, talvez não faça sentido utilizar *features* que combinem tais informações, visto que isto apenas tornaria o aprendizado mais lento.

Durante o processo de desenvolvimento, você poderá perceber que diferentes tipos de *features* podem auxiliar ou prejudicar o aprendizado. Um dos objetivos deste trabalho é experimentar com tipos diferentes de *features* expandidas e determinar quais permitem o aprendizado rápido de um controlador eficaz. Note que, por um lado, precisamos utilizar um número suficiente de *features* expandidas, de forma a permitir a representação de funções de preferência complexas. Por outro lado, não desejamos que o número de *features* seja alto a ponto de tornar o processo de aprendizado/busca local demasiadamente lento. Você deverá experimentar com diversos tipos de *features* $f(s_t)$ e descobrir quais resultam em aprendizado mais rápido de um controlador de alta performance.

5 Entrega e Apresentação de Resultados

Cada dupla deverá fazer *commit* de seu código, *de forma regular*, para o repositório Bitbucket compartilhado com o professor e com o monitor (vide documento técnico para mais detalhes). Isso permitirá com que avaliemos o progresso sendo feito pela dupla durante o desenvolvimento do trabalho. Ao final, a dupla deverá entregar a última versão do código fonte utilizado.

Cada dupla deverá preparar um relatório técnico descrevendo o projeto desenvolvido. Descreva quais algoritmos de busca foram implementados e avaliados. Idealmente cada dupla deverá implementar[2] e avaliar **ao menos dois algoritmos**. Discuta as razões que os levaram a tentar tais algoritmos, se eles funcionaram, o quão eficientes foram na descoberta de controladores eficazes, quais dificuldades foram encontradas em sua implementação, e desvantagens do uso daquele tipo de método de busca—p.ex., muito lento, não encontrou soluções boas, etc. Discuta também quais parâmetros do algoritmo de busca você precisou ajustar; tamanho da população, se utilizando um Algoritmo Genético; parâmetro de temperatura, caso utilizando Simulated Annealing; etc. Descreva *como* você descobriu valores razoáveis para tais parâmetros. Fale sobre quais *features* de estado você utilizou, e porquê. **Não** discuta com outros grupos, durante o desenvolvimento do trabalho, quais *features* sua dupla está utilizando ou avaliando (novamente, vide Seção 6 para mais informações sobre o que constitui plágio).

Por fim, você deverá apresentar e discutir a performance do seu controlador para cada um dos algoritmos avaliados. Isso deverá ser feito através de gráficos de *curva de aprendizado* para cada algoritmo. Uma curva deste tipo consiste em um gráfico onde o eixo horizontal representa o número do episódio de treinamento, e o eixo vertical mostra qual a performance do controlador naquele episódio[3]. Caso o método de aprendizado da dupla estiver conseguindo melhorar de performance do controlador de forma consistente, isso será refletido de forma clara no gráfico, que será uma curva crescente. Prepare este tipo de gráfico para os **diferentes algoritmos** avaliados e para os **diferentes conjuntos de *features*** com os quais você experimentou, a fim de demonstrar o impacto do algoritmo e da representação das funções de preferência na performance do processo de busca local. Apresente, por fim, o *tempo* de aprendizado (em segundos/minutos), para cada algoritmo e tipo de *features* utilizadas. Discuta a razão de possíveis acelerações no tempo de aprendizado em função destas escolhas.

No dia indicado no cronograma da disciplina, haverá um primeiro turno de competição entre os controladores implementados por cada dupla. Neste dia, cada grupo deverá levar, em um arquivo texto, o conjunto de parâmetros ótimos aprendidos por seu algoritmo de busca. Deverá, também, entregar a primeira versão de seu relatório. Haverá um segundo turno de competições, mais adiante no semestre, envolvendo cenários mais desafiadores para controle do SegWay. Ao final do curso, cada dupla deverá entregar seus relatórios finais (atualizados com as descobertas referentes ao segundo turno da competição) e apresentar oralmente os seus trabalhos e conclusões.

[2] i.e., desenvolver totalmente do zero! Vide política de plágio na seção 6.

[3] Tais informações são produzidas automaticamente pelo simulador, quando executado no modo **learn**.

6 Política de Plágio

Duplas poderão **apenas** discutir questões de *alto nível* relativas à resolução do problema em questão. Poderão discutir, por exemplo, quais técnicas de busca local estão considerando utilizar, quais suas vantagens e desvantagens, etc. **Não** é permitido com que as duplas utilizem quaisquer códigos fonte provido por outras duplas, ou encontrados na internet. Os alunos **poderão** fazer consultas na internet ou em livros **apenas** para estudar o modo de funcionamento das técnicas de IA, e para analisar o **pseudo-código** que as implementa. **Não** é permitida a análise ou cópia de implementações concretas (em quaisquer linguagens de programação) da técnica escolhida. **Não** discuta com outros grupos, durante o desenvolvimento do trabalho, quais *features* sua dupla está utilizando ou avaliando. O objetivo deste trabalho é justamente implementar a técnica do zero, e descobrir as dificuldades envolvidas na sua utilização para resolução de um problema de aprendizado. Toda e qualquer fonte consultada pela dupla (tanto para estudar os métodos a serem utilizadas, quanto para verificar a estruturação da técnica em termos de *pseudo-código*) **precisa obrigatoriamente** ser citada no relatório final. O professor e monitores utilizam rotineiramente um sistema anti-plágio que compara o código-fonte desenvolvido pelas duplas com soluções enviadas em edições passadas da disciplina, e também com implementações conhecidas e disponíveis online. *Qualquer* nível de plágio (ou seja, utilização de implementações que não tenham sido 100% desenvolvidas pela dupla) poderá resultar em nota zero no trabalho. Caso a cópia tenha sido feita de outra dupla da disciplina, *todos* os alunos envolvidos (não apenas os que copiaram) serão penalizados. Esta política de avaliação **não** é aberta a debate posterior. Se você tiver quaisquer dúvidas sobre se uma determinada prática pode ou não, ser considerada plágio, não **assuma** nada: pergunte ao professor e ao monitor.

Note que, considerando-se os pesos das avaliações desta disciplina (especificados e descritos no plano de ensino) percebe-se que nota zero em qualquer um dos trabalhos de implementação **obrigatoriamente** resulta em média inferior a 6.0 nos projetos práticos, o que impede com que o aluno faça prova de recuperação. Ou seja: caso seja detectado plágio, há o risco *direto* de reprovação. As duplas deverão desenvolver o trabalho **sozinhas**.