Identificando e Classificando Trechos de Pistas no Simulador de Corridas TORCS

Vinícius K. Daros, Flávio S. C. da Silva
Instituto de Matemática e Estatística - IME
Universidade de São Paulo - USP

Resumo

A criação de agentes capazes de navegar por ambientes sem o auxílio humano com eficiência tem sido foco de diversas pesquisas na área da Inteligência Artificial. Quando o ambiente em particular são pistas de corrida, existem aplicações diretas no desenvolvimento de jogos e simuladores automobilísticos. Um dos desafios envolvidos nessa tarefa é fazer bom uso dos dados disponíveis ao agente, transformando-os em informações mais condensadas e significativas. Tendo como plataforma de testes o simulador de corridas TORCS, mostramos neste trabalho um método capaz de construir um modelo de pista baseado apenas em leituras de um número limitado de sensores no carro de corrida. Além disso também apresentamos uma técnica de segmentação do circuito indicando o início e fim de trechos distintos, classificando-os por suas características - como reta, curva suave, curva fechada, entre outros. O resultado da segmentação e classificação dos trechos de cada pista pode ser um componente valioso na aplicação de técnicas de aprendizagem de máquina e/ou planejamento tanto para pilotos virtuais quanto para robôs autônomos. Nosso método se mostrou eficaz em segmentar e classificar as 18 pistas inclusas no TORCS.

Palavras-chave: Car racing, TORCS, videogame, segmentation, classification

Contato dos autores:

{vkdaros, fcs}@ime.usp.br

1 Introdução

Um dos desafios atuais na área de Inteligência Artificial é desenvolver controladores capazes de atingir uma boa performance no ambiente de simulação virtual de corridas automobilísticas sem o auxílio humano. Isto é, fazer com que um piloto virtual seja capaz de "aprender" sozinho quando acelerar, frear, fazer curvas corretamente, efetuar ultrapassagens dentre outras ações. Um primeiro passo para conseguir moldar o comportamento de um controlador é identificar os diferentes tipos de trechos das pistas, tais como reta longa, reta curta, curva fechada, entre outros, a fim de associar a cada um deles uma ação desejada. Para realizar essa tarefa, o piloto virtual deve ser capaz de montar um modelo do circuito como um todo, segmentá-lo em uma sequência de trechos e classificar cada uma dessas áreas de modo a conseguir reconhecer curvas ou retas semelhantes como sendo de uma mesma categoria. Nesse trabalho, construímos um piloto virtual para o simulador de corridas de código aberto *TORCS*[1]. Para se manter na pista, nosso controlador necessita monitorar apenas 2 sensores. Além disso, fazendo medições a cada 5 metros e armazenando dados referentes a apenas 4 outros sensores, é possível construir um modelo do circuito, segmentá-lo em trechos e classificar cada trecho.

2 Trabalhos Relacionados

Os jogos de corrida são de um gênero muito popular no mercado e também atraem o interesse de muitos pesquisadores. Vários autores tem usado o *TORCS* como base para explorar diferentes questões relacionadas a corridas, como formas eficientes de se encontrar o traçado ideal em uma pista [Braghin et al. 2008; Cardamone et al. 2010]. Há também tentativas de elaborar controladores capazes de aprender sozinhos como correr e alguns autores tem obtido êxito nessa tarefa [Onieva et al. 2009; Onieva et al. 2012; Butz and Lonneker 2009; Muñoz et al. 2010; Perez et al. 2009; Quadflieg et al. 2011]. Apesar disso, o desafio de se conseguir um piloto virtual que dirija um carro de corrida tão bem quanto um bom jogador humano ainda é uma questão sem solução fechada.

As abordagens adotadas no desenvolvimento de controladores autônomos são variadas, envolvendo redes neurais, algoritmos genéticos e lógica *fuzzy* por exemplo. Entretanto as estratégias mais recorrentes nas pesquisas geralmente envolvem algoritmos que buscam comportamentos ideais para responder a situações imediatas descritas pelos sensores do carro. Poucos são os estudos com foco em criar um modelo da pista e aproveitá-lo como parte do aprendizado do agente. Por exemplo, o trabalho de [Muñoz et al. 2010] faz um modelo da pista para que o controlador tenha ações parecidas com as que um humano teria em cada situação. Já em [Quadflieg et al. 2010] é apresentada uma técnica que usa vários sensores para construir um modelo do percurso com precisão considerável, essa informação é usada para planejar a velocidade e comportamento do agente em cada trecho.

3 *TORCS*

O ambiente de simulação usado para esse trabalho foi o *TORCS (The Open Racing Car Simulator)*, um simulador de corridas automobilísticas de código aberto desenvolvido com o intuito de permitir aos usuários a criação de módulos capazes de controlar o comportamento dos carros. Outras vantagens da escolha do *TORCS* como plataforma são: o simulador é também um jogo, permitindo o acompanhamento por um visualizador 3D e com possibilidade de um jogador humano participar das corridas; há um sofisticado mecanismo de física no qual a aderência dos pneus à pista, consumo de combustível, dano sofrido e diversos outros fatores influenciam o comportamento e desempenho do carro; o sistema é multiplataforma; e há grande variedade de conteúdo disponível, tal como pistas e modelos de carros. Por conta dos pontos já citados e muitos outros, o *TORCS* vem sendo usado em competições de inteligência artificial no meio acadêmico. Em particular, este trabalho segue os moldes do *SCRC (Simulated Car Racing Championship)*[2].

4 Construindo modelo da pista

O objetivo do piloto virtual desenvolvido nesse trabalho não é completar o percurso no menor tempo, mas sim coletar dados da pista para montar um modelo do circuito que possa ser usado para as etapas de segmentação e classificação. A seguir, descrevemos os passos que nosso controlador executa para criar modelo das pistas.

4.1 Piloto coletor de dados

Durante a volta de reconhecimento, o piloto virtual conta apenas com os dados fornecidos pelos sensores (Tabela 1) para guiar o carro e também adquirir informações sobre o traçado do circuito. Durante essa tarefa, o controlador executa três ações principais: *(i)* tentar manter o carro sempre sobre o eixo da pista (minimizando o valor de *trackPos*); *(ii)* tentar manter a velocidade constante por volta de 60 Km/h em trechos sem curvas (quando o sensor *track* frontal é maior que 50 metros) e 20 Km/h caso contrário; *(iii)* a cada 5 metros aproximadamente, armazenar os valores de todos os sensores. Quando se completa a volta, todos os dados armazenados são exportados para analise posterior e encerra-se o trabalho do piloto virtual.

A atividade de manter o carro sobre o eixo da pista demanda que se vire o volante proporcionalmente à defasagem de alinhamento

[1] http://torcs.sourceforge.net

[2] http://games.ws.dei.polimi.it/competitions/scr/

Tabela 1: *Sensores disponíveis*

Nome	Descrição
angle	Ângulo entre a direção do carro e a direção do eixo da pista. Valores negativos indicam que o carro está apontando para a direita do eixo da pista e valores negativos para a esquerda.
curLapTime	Tempo decorrido desde o início da volta atual.
damage	Quantidade de dano sofrido pelo carro.
distFromStart	Distância entre o carro e a linha de largada ao longo do eixo da pista.
distRaced	Distância percorrida pelo carro desde o início da corrida.
fuel	Indica o nível de combustível.
gear	Indica qual marcha está engatada.
lastLapTime	Tempo gasto para completar a volta anterior.
opponents	Vetor de 36 sensores que detecta a distância (de 0 a 100 metros) até um oponente. Cada sensor cobre um setor de 10°, de $-\pi$ a π, em torno do carro.
racePos	Posição na corrida com relação aos outros carros.
rpm	Número de rotações por minuto do motor.
speedX	Velocidade do carro ao longo do eixo longitudinal do carro.
speedY	Velocidade do carro ao longo do eixo transversal ao carro.
track	Vetor de 19 sensores de proximidade posicionados na frente: cada sensor indica a distância entre a frente do carro e a extremidade da pista. Cada sensor tem inclinação de 10° em relação ao anterior, indo de $-\pi/2$ a $+\pi/2$. A distância é dada em metros e o alcance dos sensores é de 100 metros. Quando o carro está fora da pista (ou seja, quando trackPos é menor que -1 ou maior que 1), esses sensores não são confiáveis.
trackPos	Distância entre o carro e o eixo da pista. O valor é normalizado com respeito à largura da pista: 0 quando o carro está sobre o eixo, -1 quando estiver sobre a borda esquerda da pista e 1 quando estiver sobre a borda direita. Valores menores que -1 e maiores que 1 indicam que o carro está fora da pista.
wheelSpinVel	Vetor de 4 sensores representando a rotação de cada roda.

fornecida pelo sensor *angle*, levando em conta também o sensor *trackPos*, pois, mesmo com o carro alinhado ($angle = 0$), deve-se mudar de direção caso não se esteja sobre centro da pista. Pela Equação (1) é possível determinar o valor $steer \in [-1, 1]$ de quanto se deve virar o volante, sendo *steerLock* a constante que define o ângulo máximo ao qual o volante pode chegar.

$$steer = \frac{angle - trackPos * steerLock}{steerLock} \quad (1)$$

$$steer' = max(min(3 * steer, 1), -1) \quad (2)$$

Com essa fórmula, o controlador é capaz de posicionar o carro ao centro da pista. Entretanto notamos pelos testes que, em curvas mais fechadas, o carro tendia a se afastar do caminho ideal. Uma vez que toda a análise dos dados se baseia na crença de que eles foram coletados com o carro sobre o eixo da pista (embora saibamos que essa é uma situação ideal e não reflete a realidade em vários momentos), buscamos adaptações que possibilitassem um melhor alinhamento em curvas mais fechadas. Dessa forma, adotamos a Equação (2), com a qual o piloto vira o carro menos suavemente mas se afasta menos do eixo da pista sem correr riscos de derrapar estando em baixas velocidades.

4.2 Curvaturas e segmentos

Coletados todos os dados referentes a uma volta completa pela pista, podemos usá-los para montar uma representação do percurso. Cada um dos dados em questão é o conjunto dos valores de todos os sensores no momento da medição, que ocorre aproximadamente a cada 5 metros. Portanto, tomando-os aos pares, é possível extrair informações para descrever pequenas fatias da pista, as quais chamaremos de **segmentos**. Primeiramente, temos que a diferença de valores em *distFromStart* revela o comprimento exato do segmento. Em seguida, é preciso descobrir se esse segmento é parte de uma reta ou se o carro mudou de direção entre as duas medições.

Para fazer essa distinção, pode-se verificar o quanto o volante estava virado na leitura inicial e final consultando o campo *steer*. Se ao menos um dos valores não for nulo, então o carro realizou uma curva no intervalo entre as medições. Nesse caso, é importante descobrir o quanto o carro virou, ou seja, a **curvatura** do segmento. Essa informação não está contida diretamente em nenhum dos sensores disponíveis e para consegui-la é preciso realizar uma série de operações.

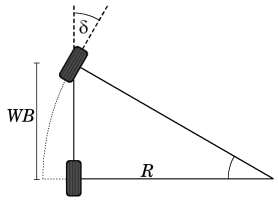


Figura 1: *Modelo de bicicleta usado para simplificar o carro.*

Simplificamos o carro pelo modelo da bicicleta [Gillespie 1992] ilustrado na Figura 1, onde *WB (wheelbase)* é a distância entre os eixos das rodas, δ é o ângulo do volante e *R* é o raio da curva. Por esse modelo, considerando-se pequenos valores para δ, é possível aproximar o raio da curva pela Equação (3).

$$R = \frac{WB}{tg(\delta)} \simeq \frac{WB}{sen(\delta)} \quad (3)$$

Tendo o raio da curva, a Equação (4) nos fornece o ângulo α do arco coberto pelo carro ao andar por esse segmento, sendo r a constante do raio das rodas dianteiras, w o valor do sensor *wheelSpinVel* de velocidade angular da roda externa à curva, Δ_t o intervalo entre as medições do início e fim do segmento e T a constante *TRACK* com o valor da distância entre as rodas do mesmo eixo.

$$\alpha = \frac{r * w * \Delta_t}{R + \frac{T}{2}} \quad (4)$$

Assim, esse processo é repetido para todos os pares de dados adjacentes e cada fatia da pista passa a ser descrita por um segmento composto por: *comprimento*, próximo a 5 metros; e *curvatura*, referente ao ângulo coberto pelo arco feito pelo carro. Contudo, ao longo das etapas descritas há arredondamentos e imprecisões tanto em decorrência do intervalo entre medições quanto pelo mal posicionamento do carro no instante da coleta. Por esse motivo, a concatenação desses segmentos não forma um circuito fechado e é preciso aplicar um fator de correção em todas as curvaturas. Para isso, verificamos a proporção entre a soma S_c das curvaturas encontradas e 2π. Em seguida, definimos o fator de correção $f = \frac{S_c}{2\pi}$. Ao multiplicarmos por f o valor da curvatura de cada segmento, garantimos que a representação da pista tenha uma volta completa e conseguimos um modelo que se aproxima da pista real. É possível ver na Figura 2 os segmentos que compõem a pista CG1 e como a parte final nem sempre se encontra exatamente com a inicial. Porém, para os objetivos deste trabalho, essa disparidade é aceitável.

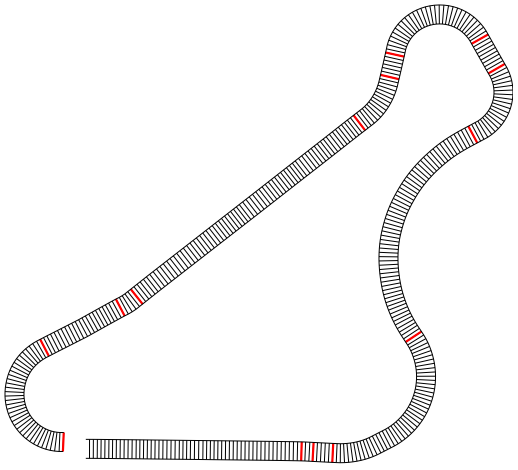


Figura 2: *Divisão da pista CG1 em segmentos de aproximadamente 5 metros. Em vermelho estão indicações dos pontos que o segmentador identificou a divisão de trechos.*

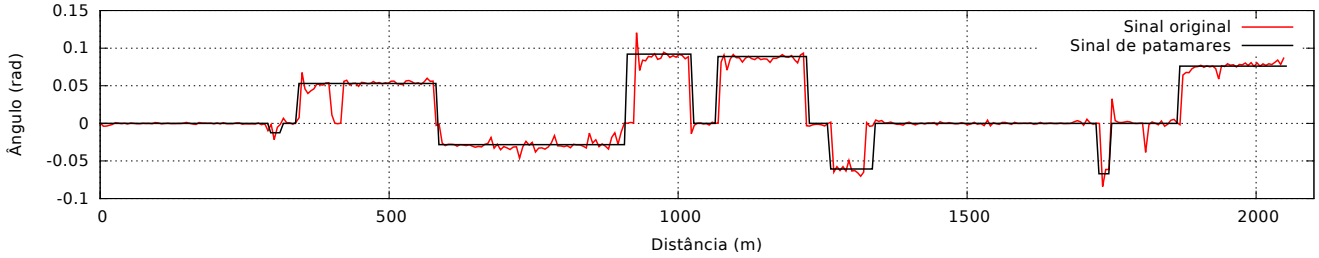


Figura 3: *Variação de ângulos da pista CG1 (Figura 2) vista como sinal.*

5 Segmentação das pistas

O modelo formado pelos segmentos de 5 metros remonta razoavelmente bem a pista, entretanto a informação contida em cada um desses segmentos é pouco expressiva e não há vantagem alguma em lidar com uma representação tão fragmentada. Ao ver o desenho de um circuito, é fácil para uma pessoa distinguir onde começa e termina cada reta, quão fechada é cada curva e quando um trecho é longo ou curto. Para um piloto, esse entendimento é importante, pois a informação do tipo de trecho em que se está e do(s) trecho(s) seguinte(s) tem influência direta na decisão de qual ação executar. Dessa forma, aglutinar os segmentos em trechos relativamente uniformes traz as seguintes vantagens:

- Simplifica-se o modelo. Por exemplo, a pista CG1 vista na Figura 2 pode ser representada por apenas 14 trechos em vez de 400 segmentos;
- A descrição por trechos é mais eficiente, transmitindo informações de forma concisa e significativa. Por exemplo, o fato dos primeiros 40 segmentos formarem uma reta com 200 metros é uma informação simples e valiosa;
- Facilita-se a leitura, compreensão e interpretação dos dados.

```
1  FiltroDePatamar(distâncias, ângulos, n)
2    ângulos ← FiltroDeMédia(ângulos)
3    limiar ← ESCALA * DesvioPadrão(ângulos)
4    base ← 0
5    distFiltrada.adiciona(0)
6    angFiltrada.adiciona(base)
7    para i ← 1 até n − 2 faça
8      se éSalto(ângulos[i], base, limiar) então
9        distTrás ← distâncias[i − 1]
10       distFrente ← distâncias[i]
11       angTrás ← base
12       enquanto i < n − 3 e não ePatamar(ângulos, i) faça
13         ▷ Se houve mudança de sinal, ajusta a distância.
14         se ângulos[i] * ângulos[i − 1] >= 0 então
15           distTrás ← distâncias[i − 1]
16           distFrente ← distâncias[i]
17         fim
18         i ← i + 1
19       fim
20       angFrente ← ângulos[i]
21       se |angFrente| < MAX_RETA então
22         angFrente ← 0
23       fim
24       ▷ Ponto do início do salto.
25       distFiltrada.adiciona(distTrás)
26       angFiltrada.adiciona(angTrás)
27       ▷ Ponto do fim do salto.
28       distFiltrada.adiciona(distFrente)
29       angFiltrada.adiciona(angFrente)
30       base ← angFrente
31     fim
32   fim
33   distFiltrada.adiciona(distâncias[n])
34   angFiltrada.adiciona(base)
35   devolva distFiltrada, angFiltrada
```

Algoritmo 1: Método de segmentação, onde n é o tamanho dos vetores *distâncias* e *ângulos*.

Para realizar a segmentação, isto é, determinar os pontos nos quais um trecho termina e outro começa, propomos um método que avalia as variações das curvaturas dos segmentos como oscilações de um sinal. Na Figura 3, a linha em vermelho ilustra o "sinal" referente à pista da Figura 2. No gráfico, o eixo das abscissas indica a distância do segmento até a linha de largada e o eixo das ordenadas se refere aos ângulos de curvatura. Vale notar que ângulos negativos indicam curvas para a direita e curvas para a esquerda tem ângulos positivos. Comparando as duas figuras, é fácil observar que os trechos de curva na pista tem paralelo com patamares no gráfico. Porém também é possível perceber que há regiões aparentemente homogêneas na pista que possuem oscilações consideráveis no sinal - como na primeira grande curva à direita por volta da distância de $600m$.

Nosso objetivo é filtrar o sinal original de modo a encontrar os **patamares** que representam trechos homogêneos na pista mesmo com a existência de oscilações. O método que desenvolvemos para tratar esses sinais é descrito pelo Algoritmo 1, o qual recebe os vetores *distâncias* e *ângulos* de tamanho n (que representam o sinal da pista) e devolve duas novas listas com as distâncias e ângulos representando o sinal contendo apenas os pontos referentes aos patamares encontrados. Ou seja, tomando dois pontos consecutivos desse novo sinal, ou eles estão alinhados (indicando um patamar) ou há uma diferença significativa de ângulo entre eles (indicando um "salto"). Na Figura 3 vemos em preto os pontos referentes ao sinal já filtrado.

O primeiro passo realizado pelo algoritmo é passar o vetor *ângulos* por um filtro de média para atenuar os picos do sinal original. Assim, os novos valores a'_i do vetor são tais que:

$$a'_i = \begin{cases} \dfrac{a_{i-1} + 2a_i + a_{i+1}}{4} & , i \in [1, n-2] \\ a_i & , cc. \end{cases}$$

Na sequência, é definido o *limiar* acima do qual a diferença entre dois ângulos é considerada significativa, um "salto". Esse limiar é proporcional ao desvio padrão das diferenças entre ângulos consecutivos. Para nossos testes, o fator *ESCALA* usado foi 4. Em seguida, o ponto inicial $(0, 0)$ é adicionado ao novo sinal e inicia-se a iteração sobre os pontos do sinal original. A função *éSalto* identifica se a diferença entre o ângulo atual e o ângulo *base* é suficientemente grande, se esses dois ângulos estão em curvas de direções opostas ou se um dos ângulos faz parte de uma reta e o outro de uma curva. Nesses casos, *éSalto* devolve *verdade*. Quando um salto é encontrado, *distTrás* e *angTrás* indicam o ponto do início do salto. Já *distFrente* e *angFrente* marcarão o fim do salto e início de um patamar. A função *éPatamar* devolve *verdade* quando os ângulos imediatamente à frente de i tem valores próximos. Durante a busca ao fim do salto, há a possibilidade de dois ângulos consecutivos terem sinais diferentes (indicando duas curvas para direções opostas em sequência). Quando isso acontece, as distâncias de início e fim do salto são reajustadas para que o salto cruze o eixo das abscissas no ponto onde há a inversão das curvas. Caso o valor absoluto de *angFrente* seja muito pequeno o arredondamos para zero, indicando que o patamar encontrado é o início de uma reta. O valor de *MAX_RETA* usado em nossos experimentos foi 0.008.

Ao término da execução do algoritmo, cada salto precede um patamar, o qual é uma região uniforme da pista e, portanto, pode ser considerado como um trecho. Assim, temos a segmentação da pista em trechos, como exemplificado pela Figura 2, onde as marcações em vermelho mostram os saltos encontrados.

6 Classificação

Depois da segmentação, cada trecho pode ser descrito por dois atributos: (i) comprimento; (ii) e raio, que determina o quanto uma curva é fechada. Como o comprimento de um trecho de curva está diretamente relacionado ao ângulo da mudança de direção do trajeto, optamos por usar esse ângulo para a classificação. Além disso, em trechos de reta, o raio deveria ser infinito, porém usaremos valores negativos para representar esse caso.

Nosso classificador agrupa trechos em 11 classes distintas. Embora não haja necessidade de rotular cada categoria, atribuímos a cada uma delas um nome de modo a facilitar a compreensão da classificação. As retas foram divididas em *curta*, *média* e *longa*. As curvas, da mais fechada para a mais suave, são *hairpin (grampo)*, *cotovelo*, *acentuada*, *moderada* e *leve*, sendo que as três últimas categorias ainda foram subdivididas em *curta* ou *longa*. As regras para a classificação dos trechos estão descritas no Algoritmo 2.

```
1  Classificador (ângulo, raio, comprimento)
2    ▷ Retas tem raio infinito.
3    se raio < 0 então
4      se comprimento <= 50 então
5        devolva Reta curta
6      se comprimento <= 150 então
7        devolva Reta média
8      devolva Reta longa
9    senão
10     se raio <= 60 então
11       se ângulo >= 120° então
12         devolva Hairpin
13       se ângulo >= 50° então
14         devolva Cotovelo
15     se raio <= 127 então
16       se ângulo <= 50° então
17         devolva Curva acentuada curta
18       devolva Curva acentuada longa
19     se raio <= 185 então
20       se ângulo <= 50° então
21         devolva Curva moderada curta
22       devolva Curva moderada longa
23     senão
24       se ângulo <= 50° então
25         devolva Curva leve curta
26       devolva Curva leve longa
```

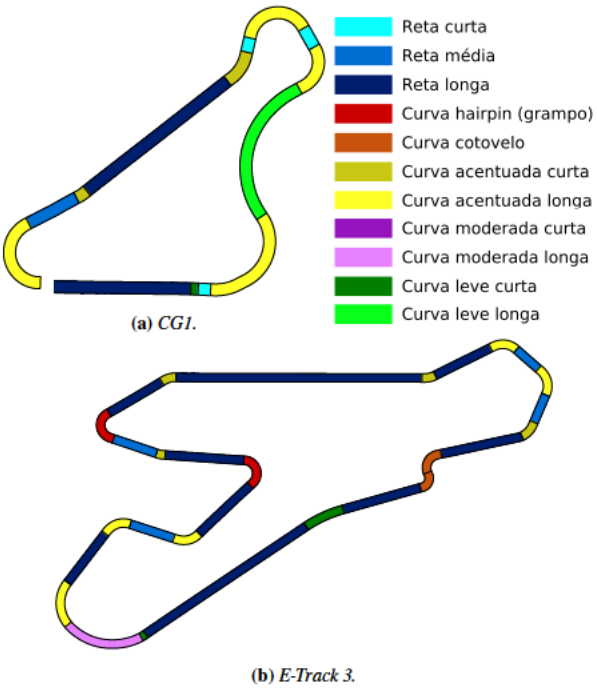
Algoritmo 2: Classificador de segmentos.

7 Resultados

Usando o *TORCS* como ambiente de testes, executamos baterias de experimentos nas 18 pistas de asfalto que acompanham o simulador. Em todas elas o resultado da segmentação e classificação foram satisfatórios, como exemplificado pela Figura 4. Embora algumas pistas tenham apresentado encaixe perfeito entre o início e fim do circuito como visto na Figura 4b, isso não ocorreu em todos os casos. Entretanto essa não é uma condição essencial para os contextos de uso que este trabalho visa atender.

8 Conclusão

Nesse trabalho, mostramos como construímos um piloto virtual para coletar dados das pistas do *TORCS* e como calcular a curvatura dos segmentos. Além disso, apresentamos um método de segmentação de pistas baseado na interpretação da variação das curvaturas como oscilações em um sinal. Por fim, propusemos um conjunto de regras para identificar e classificar trechos com características semelhantes. O ferramental aqui apresentado se mostrou eficaz e pode ser utilizado no desenvolvimento e aplicação de técnicas de aprendizagem de máquina em ambientes de simulação virtual e jogos de corrida. Também acreditamos que há potencial de uso em sistemas de robôs móveis ou qualquer outro método de tomada de decisão e planejamento que se baseie na interpretação dos tipos de trechos de um circuito fechado.



(a) *CG1.*

(b) *E-Track 3.*

Figura 4: *Exemplos de resultado da segmentação e classificação de trechos de pistas do TORCS.*

Referências

BRAGHIN, F., CHELI, F., MELZI, S., AND SABBIONI, E. 2008. Race driver model. *Comput. Struct. 86*, 13–14 (July), 1503–1516.

BUTZ, M., AND LONNEKER, T. 2009. Optimized sensory-motor couplings plus strategy extensions for the torcs car racing challenge. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 317–324.

CARDAMONE, L., LOIACONO, D., LANZI, P., AND BARDELLI, A. 2010. Searching for the optimal racing line using genetic algorithms. In *Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium on*, 388–394.

GILLESPIE, T. D. 1992. *Fundamentals of Vehicle Dynamics*. Warrendale, PA, Mar.

MUÑOZ, J., GUTIERREZ, G., AND SANCHIS, A. 2010. A human-like torcs controller for the simulated car racing championship. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 473–480.

ONIEVA, E., PELTA, D., ALONSO, J., MILANÉS, V., AND PÉREZ, J. 2009. A modular parametric architecture for the torcs racing engine. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 256–262.

ONIEVA, E., A., D. P., GODOY, J., MILANÉS, V., AND PÉREZ, J. 2012. An evolutionary tuned driving system for virtual car racing games: The autopia driver. *Int. J. Intell. Syst. 27*, 3 (Mar.), 217–241.

PEREZ, D., RECIO, G., SAEZ, Y., AND ISASI, P. 2009. Evolving a fuzzy controller for a car racing competition. In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, 263–270.

QUADFLIEG, J., PREUSS, M., KRAMER, O., AND RUDOLPH, G. 2010. Learning the track and planning ahead in a car racing controller. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 395–402.

QUADFLIEG, J., PREUSS, M., AND RUDOLPH, G. 2011. Driving faster than a human player. In *Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I*, EvoApplications'11, 143–152.