# Simulação Virtual de Carros em Jogos e Aplicações de I.A.

Fernando Santos Osório Gustavo Pessin Denis Fernando Wolf
Eduardo do Valle Simões Kalinka R L J Castelo Branco

USP - Universidade de São Paulo, ICMC – Depto. SSC, São Carlos, SP

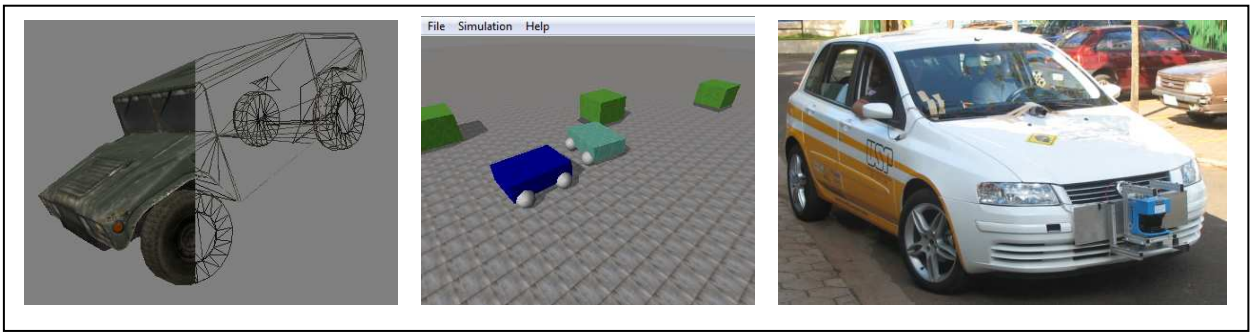



Figura 1: Simulação Virtual de Carros - Modelo virtual 3D (Esq.), Simulação Física (Centro), Estudo com Carro Real (Dir.)

## Resumo

Este tutorial visa apresentar conceitos, técnicas e ferramentas usadas na implementação de simuladores virtuais de veículos (corridas de carros). Inicialmente será apresentada uma introdução ao tema, contextualizando a apresentação e os conteúdos a serem abordados. Será discutida a questão da simulação virtual e da simulação de veículos, introduzindo os conceitos sobre os principais elementos envolvidos: cenários, carros (simulação física e controle), trajetórias, e comportamento autônomo/inteligente.

**Key-words**: Jogos de Corrida, Veículos, Simulação Virtual, Simulação Física, Agentes Inteligentes Autônomos, "*Car Racing Games*"

**E-mail de Contato dos Autores**:
{ fosorio, pessin, wolf }@ icmc. usp. br
{ simoes, kalinka } @ icmc. usp. br

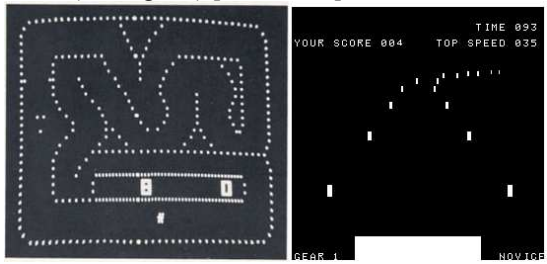
## 1. Introdução

Os jogos de simulação de corridas sempre foram uma das áreas de entretenimento digital bastante difundidas junto ao público de usuários e de desenvolvedores, a exemplo dos inúmeros títulos muito conhecidos do público: Top Gear, Daytona, SEGA GT, GTR, Rock N´Roll Racing, Super Mario Kart, BurnOut, Carmageddon, F1 Grand Prix, Gran Turismo, Nascar Racing, Colin McRae Rally, Project Gotham Racing, Need for Speed, e muitos outros jogos que podem ser citados [Car Racing Games 2009], inclusive jogos de simulação como o GTA (*Grand Theft Auto*) [GTA 2009] que incluem entre outros elementos os veículos. Como se pode constatar, o apelo dos jogos de simulação de carros e de corridas tem um grande público e por isso é de grande importância e relevância o estudo e o conhecimento dos elementos envolvidos no desenvolvimento destes jogos. Além disto, os jogos de corrida contêm elementos importantes de simulação física (cinemática e dinâmica), muito presentes em jogos mais modernos, e elementos de Inteligência Artificial (veículos autônomos/oponentes virtuais), sendo que tais conceitos são de interesse de desenvolvedores de diferentes tipos de jogos e não apenas de jogos de corrida.

Em função da relevância do estudo dos jogos de corrida, este tutorial se propõe a apresentar conceitos, técnicas e ferramentas que sirvam de suporte e referência aos interessados em desenvolver jogos e aos interessados em se aprofundar sobre o tema e realizar pesquisas na área de simulação virtual de veículos.

## 2. Realismo em Jogos

Os jogos digitais tem evoluído rapidamente em termos de suas características, recursos, modo de implementar e de se jogar. Se formos olhar a evolução dos jogos digitais, os primeiros como os jogos o Pong da Atari de 1972, eram em 2D com recursos gráficos bastante limitados. Um dos primeiros jogos de corrida para computadores foi o Grand Trak 10 lançado em 1974 pela Atari (ver Fig.2 a) e depois outros títulos também ficaram famosos como o Night Driver também da Atari de 1976 (ver Fig.2 b) [Klov 2009].




(a) (b)

Figura 2: Primeiros Video-Games de Corrida da Atari

Como pode ser constatado, nestes primeiros jogos 2D o realismo gráfico era muito limitado, a tal ponto que se abusava da imaginação do jogador criando-se assim jogos de corrida "noturnos", a fim de simplificar o cenário. Com a evolução do hardware gráfico, jogos com recursos gráficos mais sofisticados apareceram, passando do 2D ao "2D e meio" (uso de efeitos dando uma noção de perspectiva, porém com o processamento ainda feito em 2D/bitmaps (*sprites*), a exemplo dos jogos da SEGA como o Turbo de 1981 e do *Pole Position* da Atari/Namco de 1982, conforme ilustrado na Fig.3) [Klov 2009]. O primeiro jogo de corrida com polígonos em 3D foi o Hard Drivin (Atari, 1989).

Figura 3: Video-Game Pole Position – Atari/Namco 1982

A partir da década de 90, o desenvolvimento dos jogos de corrida foi realmente espetacular, acompanhando o rápido desenvolvimento de softwares gráficos e principalmente das modernas placas gráficas 3D. A Computação Gráfica evoluiu de tal forma que atualmente é difícil diferenciar uma imagem sintetizada pelo computador de uma imagem real[1]. Surgem jogos com alto realismo gráfico em 3D, com texturas, sombras, marcas de pneus na estrada, e muitos outros efeitos gráficos, dando uma sensação de imersão muito maior ao jogador, como mostrado na Fig.4.


Figura 4: Project Gotham Racing 4
http://www.bizarrecreations.com/games/pgr4/

**Realismo Gráfico:** De certo modo, podemos afirmar que a Computação Gráfica já atingiu um elevado grau de maturidade, fornecendo ferramentas de Hardware e de Software que permitem criar jogos (mais do que) realistas em termos visuais.

[1] Realismo 3D em Computação Gráfica: http://cg.tutsplus.com/articles/web-roundups/50-breathtaking-cg-images/

Entretanto a busca do realismo nos jogos não fica somente em termos de efeitos visuais, e mais recentemente os jogos vem experimentando uma grande melhoria em termos do realismo físico. Com o aumento do poder computacional dos PCs e das consoles de jogo, por meio da integração de GPUs (*Graphical Processing Units*) e de CPUs Multi-Core, atualmente já é possível realizar um tratamento mais adequado da física do jogo.

Da mesma forma como ocorreu com os recursos gráficos que evoluíram de 2D para 3D melhorando o realismo, a física nos jogos passou de um nível de "ajustes feitos a mão" para modelos de simulação da cinemática e da dinâmica dos veículos: incluindo aceleração/desaceleração, atrito, derrapagens, colisões e reação a colisões com um alto grau de realismo (baseadas em simulação de equações da física). Inclusive, mais recentemente aparecem no mercado as PPUs (*Physical Processing Units*) como a placa aceleradora da AGEIA PhysX, empresa que foi adquirida posteriormente pela NVidia [PhysX 2009]. Um outro elemento interessante dos jogos de corrida foi o aparecimento de dispositivos de *force-feedback* que aumentam ainda mais o realismo físico, mas no entanto a abordagem deste tipo de dispositivos está além do escopo deste trabalho.

**Realismo Físico:** Mais recentemente os jogos vêm atingindo um maior grau de maturidade no que diz respeito à simulação física, fornecendo ferramentas que permitem criar jogos realistas tanto em termos visuais como em relação ao comportamento físico de suas entidades.

A busca por um maior realismo em jogos não termina no realismo visual e de simulação física, tendo atualmente mais um nível de realismo que vem sendo intensamente estudado e buscado: o realismo comportamental. Os jogos de corrida e de ação incluem diversos componentes com comportamento próprio, inclusive os "oponentes virtuais", os chamados NPCs (*Non-Player Characters*). Os NPCs, ou Agentes Virtuais de um jogo, devem se comportar de modo adequado em relação ao que se espera do comportamento destes elementos junto ao mundo real.

Portanto, ao se buscar um comportamento realista, estamos também buscando a criação de "agentes inteligentes" capazes de se comportar de modo similar ao ser humano, como por exemplo, criando motoristas virtuais que atuem como uma pessoa. Estes "agentes autônomos" devem ter a capacidade de perceber informações sobre o seu estado e sobre o estado do ambiente em que estão inseridos, de modo a melhor decidir como devem agir. Sendo assim, passamos para um novo nível de realismo, onde os comportamentos não são todos pré-programados, mas sim determinados por cada agente que percebe a situação em que se encontra e reage de modo inteligente a esta situação, planejando, selecionando e realizando ações.

**Realismo Comportamental:** Passamos assim a integrar nos jogos 3 níveis de realismo: o realismo visual (computação gráfica), o realismo físico (simulação física) e o realismo comportamental (agentes autônomos inteligentes) [Osorio et al. 2006].

É importante salientar esta evolução histórica dos jogos, e mais especificamente dos jogos de corrida, que passaram por uma grande sofisticação e melhoria, sempre buscando um maior realismo:

**Realismo Visual ⟶ Físico ⟶ Comportamental**

Na realidade, o que vem ocorrendo com os jogos de corrida é a transformação de simples programas que imitavam elementos básicos visuais de corridas, para verdadeiras "recriações virtuais" que sejam o mais próximas possível da realidade. Estamos na realidade implementando *Ferramentas de Simulação Virtual em Tempo Real* de veículos. A busca pelo "realismo completo" nos leva a simular a realidade buscando aproximar o mais possível o virtual do real.

Esta é uma tendência clara que pode ser observada na evolução dos jogos digitais de corrida. Em função disto, iremos apresentar a seguir conceitos, ferramentas e métodos que nos permitem justamente avançar nesta direção: a criação de simuladores virtuais de veículos.

## 3. Veículos Virtuais

Os veículos virtuais podem ser simulados com diferentes graus de "sofisticação" (realismo visual, físico e comportamental). Portanto, o primeiro passo necessário para se projetar uma ferramenta de simulação virtual é a criação de um ***modelo de simulação*** do elemento simulado. No caso da simulação de veículos, busca-se então a criação de modelos que permitam simular a forma como este veículo irá se comportar junto ao ambiente em que está inserido. Nesta seção vamos focar no estudo dos modelos físico e comportamental, assumindo que os modelos visuais (modelagem 3D) serão providos por um *designer*, através do uso de ferramentas adequadas para a criação dos modelos gráficos dos veículos (e.g. 3D Studio Max, Maya, Blender, TrueSpace).

Um modelo de simulação é uma simplificação da realidade, que captura elementos essenciais para descrever um fenômeno. No caso de um veículo podemos simplificar o problema da simulação de um carro, considerando apenas a sua movimentação (cinemática do veículo), ou então, podemos considerar um modelo mais completo das forças que atuam no veículo influenciando seu comportamento (dinâmica do veículo), incluindo elementos como: aceleração, desaceleração, inércia, fricção, gravidade, etc.

Inicialmente vamos descrever modelos que são usados para simular o controle da cinemática de um corpo (veículo), fazendo o controle de sua trajetória.

### 3.1 Simulação do Movimento (Cinemática)

A simulação de movimento pode ser feita assumindo diferentes modelos, como por exemplo: um modelo de um veículo pontual no espaço 2D (modelo pontual 2D); um modelo de um veículo com comprimento, largura e uma barra de direção ainda em 2D (modelo Ackerman); um modelo de um veículo em 3D (modelo 3D), com volume e posicionado em um ponto do espaço tri-dimensional.

3.1.1. Modelo Pontual 2D

O modelo de um veículo pode ser simplificado (de modo extremo), de forma a representar este apenas pela sua pose: definida por meio de sua coordenada no plano (X,Y) e sua orientação angular (Θ). Este tipo de modelo é muito usado em simuladores 2D ou 3D com corridas em uma pista plana. Note que o veículo possui apenas uma posição e orientação (pose 2D = X,Y, Θ), onde inicialmente não estamos tratando de sua altura e inclinação. Alguns jogos do tipo "*side-scrolling vídeo games*" ou "*platform games*" (Wikipédia), podem inclusive considerar apenas a posição do veículo, sem considerar sua orientação (visão lateral do carro, sempre alinhado horizontalmente na tela).

Para definir a pose de um modelo pontual (modelo de partícula), será adotado um ponto de referência em relação ao modelo visual (*bitmap*), sendo usualmente adotado como referência, o seu canto inferior esquerdo, ou seu centro de massa (ver Fig. 5). Este ponto é usado como sendo a referência da pose 2D do veículo.

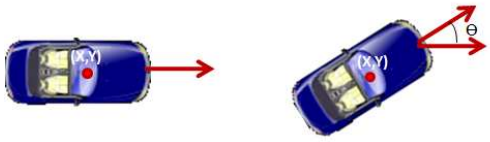
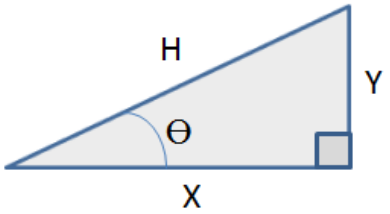



Figura 5: Pose 2D – Posição X,Y e Orientação Θ

Um veículo cuja cinemática permite assumir qualquer posição e orientação no plano 2D é definido como sendo um veículo holonômico (sem restrições de graus de liberdade), sendo estas questões de cinemática, deslocamento e restrições de movimento, bastante estudadas junto a robótica móvel [Dudek 2000, Siegwart 2004].

Do ponto de vista da implementação deste tipo de cinemática 2D sem restrições de liberdade, vamos usar como base conceitual elementos da trigonometria, onde uma ótima referência para estes conceitos (simples e bem apresentada) pode ser encontrada na obra de Holzner [Holzner 2005]. O teorema de Pitágoras é a base para o desenvolvimento dos cálculos que permitem estimar a posição e orientação de um veículo 2D que esteja sendo simulado, seja em um jogo feito em *Flash*, ou seja em um simulador de complexos robôs móveis holonômicos com cinemática diferencial (ver equações da Fig. 6).




H = Hipotenusa
X , Y = Catetos

Teorema de Pitágoras
$X^2 + Y^2 = H^2$
ou $H = \sqrt{X^2 + Y^2}$

Fórmulas Trigonométricas
X = H . Cosseno ( Θ )
Y= H . Seno ( Θ )

Figura 6: Revisão de Conceitos de Trigonometria [Holzner 2005, pg. 52]

A trajetória deste tipo de veículo com uma representação pontual será conforme apresentado na Figura 7, sempre tomando como referência a sua pose 2D (posição atual - ponto $X_0,Y_0$ e a orientação atual Θ), e se deslocando na direção apontada pela sua atual orientação.

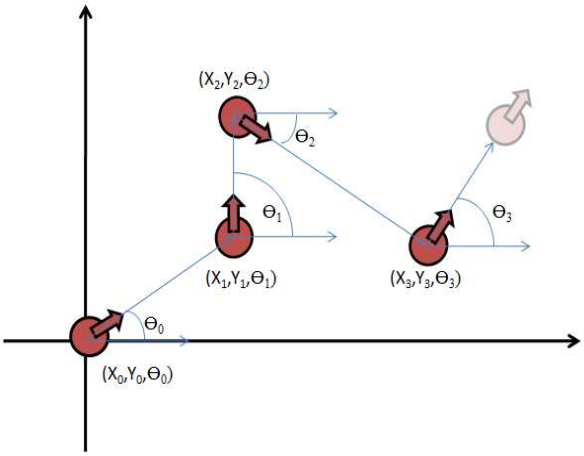



Figura 7: Trajetória baseada em um modelo pontual 2D

A implementação desta cinemática para um modelo pontual 2D é portanto baseada nas equações trigonométricas descritas anteriormente. Considerando o círculo de raio unitário (R=1), onde o R assume o valor de H (Hipotenusa = 1) do triângulo retângulo formado pelos catetos X e Y (coordenadas X e Y) e pela Hipotenusa H (Raio), conforme indicado na Figura 8.

O deslocamento deste veículo pontual se dará na direção apontada, a qual é definida pelo ângulo Θ. Para fazer avançar o veículo na direção desejada basta, portanto, aplicar as equações trigonométricas:

$X = H * \cos(\Theta)$ (Eq. 1)
$Y = H * \sin(\Theta)$ (Eq. 2)

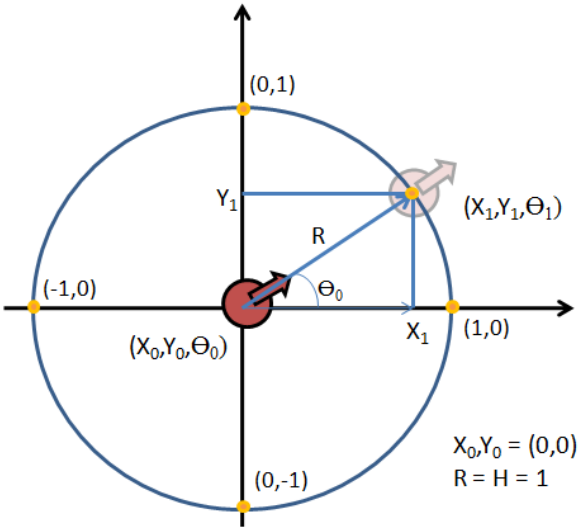



Figura 8: Circunferência de Raio Unitário

As Equações 1 e 2, dado um ângulo de 0 (zero) graus, irão obter o ponto (1,0) pois X = Cos (0) = 1 e Y = Sin(0) = 0, e para um ângulo de 90 graus, teremos o ponto (0,1) pois X = Cos(90) = 0 e Y = Sin(90) = 1. Conforme o ângulo fornecido será obtido assim um ponto sobre o perímetro da circunferência, uma vez que o raio é constante e igual a 1. Note que o ângulo inicia em 0 graus (ou 0 radianos) e vai aumentando até 360 graus (ou 2Π radianos[2]), fazendo com que seja executado um giro no sentido Anti-Horário.

Sendo assim, podemos mover um ponto localizado na origem (0,0) de uma unidade em qualquer direção definida pelo ângulo Θ.

Para que possamos realizar um deslocamento maior, de N unidades, basta que o raio seja aumentado, aplicando-se as equações 1 e 2 com um valor de R (ou H) maior. Por exemplo, para mover 10 unidades na direção dada pelo ângulo Θ, basta realizar o seguinte cálculo:

X1 = Raio * Cos (Θ) = 10 * Cos (Θ)
X2 = Raio * Sin (Θ) = 10 * Sin (Θ)

De um modo mais genérico, podemos ter nosso veículo localizado em qualquer ponto do plano 2D realizando um deslocamento de N unidades na direção para onde está orientado. Basta adaptar as Equações 1 e 2 de modo a incluir uma translação do nosso ponto de referência para a sua localização atual $X_0,Y_0$ no plano:

$X_1 = X_0 + Raio * \cos(\Theta)$ (Eq. 3)
$Y_1 = Y_0 + Raio * \sin(\Theta)$ (Eq. 4)

Portanto as Equações 3 e 4 permitem que se realize um movimento de uma posição qualquer inicial $X_0,Y_0$, deslocando-se na direção dada pelo ângulo Θ, e avançando um número de unidades R (Raio do deslocamento). Para se implementar uma trajetória

[2] Conversão Xg Graus em Yr Radianos: Yr = Xg * PI / 180
Nota: As funções Sin() e Cos() em "C" usam Radianos.

como a da Figura 7, basta realizar o deslocamento a partir de ($X_0$,$Y_0$, Θ) e a cada nova posição alcançada ($X_1$,$Y_1$, Θ) em que o ângulo Θ é alterado, assumir as coordenadas ($X_1$,$Y_1$) atuais como a nova coordenada de referência ($X_0$,$Y_0$) usada no cálculo do deslocamento. A distância do caminho percorrido, representada pelo incremento do valor do raio R, volta ao seu valor inicial R=0 e começa a ser incrementada novamente até que se alcance a próxima coordenada onde é feita uma alteração no ângulo de rotação.

Em um jogo, usualmente o usuário irá usar o teclado (ou um *joystick*) para comandar o avanço e giro do veículo. Baseado nas equações 3 e 4 podemos "traduzir" os comandos de avançar e girar da seguinte forma, cada vez que o usuário indicar que é para avançar, será feito um incremento da variável que armazena o raio (deslocamento) em relação ao ponto de referência de origem ($X_0$,$Y_0$). Cada vez que o usuário alterar a orientação por meio de um comando de giro, devemos: (i) atualizar o nosso ponto de origem ($X_0$,$Y_0$) com a coordenada atual onde se encontra o veículo ($X_1$,$Y_1$); (ii) alterar a variável que armazena o ângulo atual (Θ); (iii) retornar o valor do deslocamento (raio) para zero e recomeçar o processo: avança na direção Θ em relação ao ponto de referência $X_0$,$Y_0$.

É importante destacar que até o presente, foi abordada apenas a questão referente ao deslocamento que irá definir a trajetória do carro, onde outros conceitos físicos também devem ser incorporados no modelo de simulação. Os conceitos de velocidade e de aceleração devem ser também integrados no simulador. Um carro deve poder aumentar e diminuir sua velocidade, acelerando e desacelerando, de acordo com comandos que são dados pelo usuário.

Considerando o modelo matemático descrito anteriormente, a velocidade do veículo é na realidade o "tamanho do passo" (deslocamento) que é adicionado à posição atual do veículo. Mais especificamente, dentro do laço de execução do jogo, a velocidade é o valor do incremento do Raio a cada novo ciclo de execução (*game loop*). Por exemplo:

Raio = Raio + Velocidade (Eq. 5)

Então, quanto maior a velocidade, maior será o deslocamento do veículo, se afastando de seu ponto de referência ($X_0$,$Y_0$). Podemos adicionar uma dinâmica ao movimento do carro, fazendo com que o controle da velocidade sofra uma aceleração e desaceleração, onde neste caso, ao invés de aplicar uma Velocidade fica ao deslocamento (Raio) do carro, vamos aumentar esta velocidade com uma taxa constante (aceleração). Além disto, ao invés de também parar o carro bruscamente, simplesmente zerando sua velocidade, vamos então diminuir esta velocidade com uma taxa constante (desaceleração), que equivale a frenagem do carro. Esta aceleração/desaceleração pode ser implementada da seguinte forma:

```
Se Acionou_Acelerador
Então Velocidade = Velocidade + Taxa_Aceleração

Se Acionou_Freio
Então Velocidade = Velocidade - Taxa_Aceleração

Nova_Posição = Posição_Atual + Velocidade
```

Um exemplo de programa que realiza este deslocamento, baseado em um modelo pontual 2D definido através de sua pose 2D ($X_0$,$Y_0$, Θ) e sem restrições de graus de liberdade para o deslocamento, pode ser acessado no site criado pelos autores deste trabalho [Tutorial-Carros 2009].

### 3.1.2. Modelo Ackerman

O modelo de simulação de veículos representados por sua pose 2D e sem restrições referentes a sua trajetória não corresponde ao comportamento da maioria dos veículos que estamos habituados a dirigir. Um carro possui, basicamente, um controle de aceleração (pedal do acelerador - *gas*), um controle de frenagem (pedal do freio - *break*), um sistema de troca de marchas (câmbio - *gearbox*) e um sistema de direcionamento controlado pelo volante (direção - *steering wheel*). Uma vez que estamos interessados inicialmente na definição da trajetória do veículo, deixaremos de lado por enquanto as questões relativas a aceleração e desaceleração, e troca de marchas.

O modelo que descreve a cinemática de veículos com 4 rodas e uma barra de direção que comanda as rodas dianteiras, a exemplo da maioria dos veículos comerciais, corresponde ao modelo da cinemática Ackerman [Dudek 2000, Siegwart 2004, Steering 2009]. Neste modelo, mais realista, o carro descreve uma curva ao redor de um ponto central virtual (ICR) que é definido de acordo com o grau de giro da direção. As Figuras 9 e 10 apresentam um esquema representando o modelo da cinemática Ackerman.

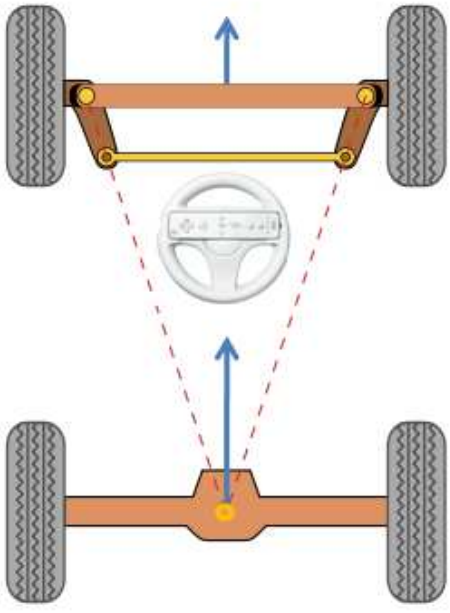


Figura 9: Cinemática Ackerman – Deslocamento Linear

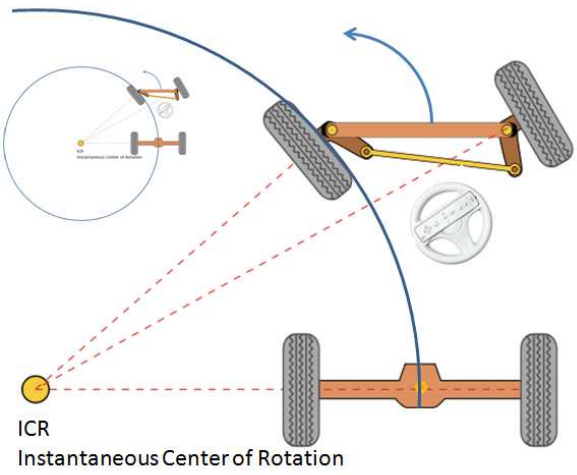



Figura 10: Cinemática Ackerman – Deslocamento Angular

Portanto, um carro com a cinemática Ackerman, ao manobrar, realiza uma rotação em torno de um ponto virtual de rotação (ICR), onde este ponto é definido de acordo com o grau de giro da direção. É importante destacar que, ao contrário do modelo pontual, o modelo de Ackerman não permite que o veículo realize um giro ao redor do próprio eixo (ou centro de massa). Sendo assim, ao conduzir um carro com a cinemática Ackerman é necessário manobrar este de modo a ajustar sua trajetória, respeitando as limitações impostas pelo seu modelo cinemático.

Pode-se afirmar que é justamente devido ao modelo de Ackerman que muitas vezes a tarefa de estacionar um carro torna-se uma tarefa complicada, que pode levar a uma manobra com diversos ajustes até que se consiga entrar na vaga.

Existe uma farta documentação na Internet descrevendo este modelo [Car_Racing 2009, Ackerman 2009], inclusive com simuladores demonstrando exemplos de como implementar veículos deste tipo [AI_Car 2009], onde neste caso em particular o autor busca simular um carro de controle-remoto (*RC car*). O site [Tutorial-Carros 2009] também apresenta um exemplo deste tipo de modelo cinemático.

A implementação da cinemática do modelo de Ackerman pode ser feita com base nas equações previamente apresentadas onde, no entanto, aplicamos a rotação considerando um ponto de referência externo ao veículo (ICR: *Instantaneous Center of Rotation*), e um raio (TR: *Turning Radius*) que pode ser determinado de acordo com os parâmetros do giro da barra da direção (SWA: *Steering Wheel Angle*) e o comprimento do carro (L: *Length*).

```
ICR = (X0,Y0)
R  = TR = L / Tan(SWA)      (Eq. 6)
Θ: 1) Incrementado de modo a girar no
   sentido anti-horário. Realiza uma
   curva para a esquerda.
2)Decrementado de modo a girar no sentido
  horário. Realiza uma curva para a
  direita.
```

A Figura 11 ilustra o modelo da cinemática Ackerman e seus componentes. A partir deste modelo podemos constatar que a implementação da cinemática Ackerman é feita considerando as fórmulas do modelo pontual onde o ponto de referência da rotação é o ICR. O valor de ICR é obtido traçando uma linha contígua ao eixo do veículo de comprimento igual a TR. As coordenadas do ponto ICR, irão definir o ponto $(X_0,Y_0)$ do modelo pontual:

Aplicar a Eq. 6 para determinar o TR:

```
Se SWA < > 0
   Então TR = L / Tan(SWA)
   Senão TR é infinito. Se SWA for 0 (zero)
   o carro deve se deslocar em linha reta
   [ Usar o modelo pontual 2D, considerando o
     centro do eixo traseiro como referência ]
   Obs.: SWA assume um intervalo angular
         limitado.

Xcet = Coordenada X do centro do eixo traseiro
Ycet = Coordenada Y do centro do eixo traseiro
X0: Se SWA > 0   [Giro para a esquerda]
    Então X0 = Desloca_Eixo(Xcet,TR,Esquerda)
    Se SWA < 0   [Giro para a direita]
    Então X0 = Desloca_Eixo(Xcet, TR, Direita)
Y0:  Repete os passos de X0 usando Y0 e Ycet.
Θ = Ângulo de giro do veículo relativo ao
    ponto de referência ICR (deslocamento).
```

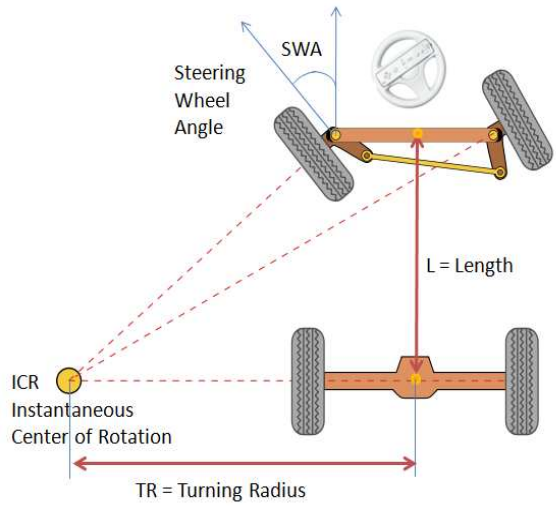



Figura 11: Modelo e Componentes da Cinemática Ackerman

O modelo apresentado na Figura 11 permite definir a trajetória do veículo de modo similar a trajetória de um carro que utiliza uma barra de direção. A velocidade do carro será fornecida pelo incremento da variável Θ, ou seja, enquanto no modelo pontual o raio R é usado para indicar o quanto o veículo se deslocou, no modelo Ackerman o ângulo Θ irá indicar o quanto da trajetória circular em torno de ICR foi percorrida pelo veículo. Então, os podemos implementar um controle de velocidade, inclusive adicionando parâmetros de aceleração e desaceleração, a exemplo do que foi feito no modelo pontual usando o valor de R, mas neste caso, usando o valor do ângulo Θ.

3.1.3. Modelo 3D

A simulação em um plano 2D é mais simples, onde podemos usar os conceitos apresentados no modelo pontual e no modelo Ackerman, inclusive em implementações em ambientes 3D, mas onde a pista é plana. Entretanto, em jogos de corrida, nem sempre os carros irão ficar limitados apenas ao uso de pistas planas, a exemplo de corridas em ambientes externos do tipo *off-road* (Fig. 12).

Figura 12: Corrida Off-Road – 1nsane from CodeMasters

A simulação em 3D requer um modelo mais completo do veículo, onde a pose 3D será definida por uma coordenada (X,Y,Z) e sua orientação no espaço dada pelos ângulos (Θx, Θy, Θz). A pose 3D inclui a elevação do veículo no terreno, assim como o seu ângulo de rotação em relação aos 3 eixos, a saber:

- Rotação XZ (Θy): Rotação ao redor do eixo Y, no plano XZ (considerada análoga a rotação 2D).
- Rotação XY (Θz): Rotação ao redor do eixo Z, no plano XY.
- Rotação YZ (Θx): Rotação ao redor do eixo X, no plano YZ.

Combinando as 3 rotações podemos fazer com que um veículo seja orientado em qualquer direção. Cabe destacar que podemos usar as equações de rotação no plano 2D (XY), apenas alternando para os demais planos (XZ e YZ). A Figura 13 apresenta uma representação gráfica das rotações nos 3 eixos.


Figura 13: Rotação Θx (Pitch), Θy (Yaw), Θz (Roll)

O controle da trajetória de um veículo em um ambiente 3D torna-se um pouco mais sofisticado, uma vez que agora é necessário controlar sua posição, orientação e sentido de deslocamento, considerando sua pose 3D. A pose 3D irá indicar a direção do seu deslocamento.

De forma análoga ao modelo pontual 2D, podemos considerar um carro como sendo um modelo pontual 3D que possui uma posição e orientação espacial, e a partir de sua pose 3D, deslocar o veículo considerando o referencial de sua pose 3D. Este tipo de implementação é bastante usual em jogos 3D, mas no entanto, o modelo mais completo inclui o tratamento da cinemática Ackerman em um espaço 3D [Heinen 2007].

A simulação de um veículo se deslocando em um terreno tridimensional irregular faz com que seja necessário o ajuste de sua inclinação em relação ao terreno, onde a pose do veículo não irá depender apenas da sua posição e orientação, a fim de determinar a direção de deslocamento. Será necessário considerar forças adicionais, como a gravidade, puxando o veículo em direção do solo, e fazendo com que muitas vezes a combinação de direção de deslocamento e alteração na inclinação do terreno produza como resultado um salto.

Desta forma, os modelos de simulação vêm sendo sofisticados de modo a incluir elementos referentes à trajetória do veículo, mas também referentes à sua dinâmica. O simulador deve passar a controlar diferentes forças e elementos que atuam sobre o veículo, como por exemplo: velocidade linear, velocidade angular, massa, centro de massa, peso, dimensões, atrito/aerodinâmica, fricção/derrapagem, gravidade, inclinação do terreno e do veículo, e colisões (com reação as colisões) com outros corpos. Desta forma o simulador irá incorporar diversos elementos da física, da cinemática e da dinâmica de corpos rígidos. Aconselhamos aos leitores as seguintes obras [Bourg 2001, Conger 2004, Palmer 2009], caso queiram se aprofundar no desenvolvimento de uma implementação própria de rotinas de simulação física.

Por outro lado, se nosso objetivo é o de implementar mais rapidamente um simulador de veículos de tempo real e com simulação física, é possível fazer uso de bibliotecas e *engines* físicas [Physics_Engine 2009] prontas de simulação física de corpos rígicos, como por exemplo: Havok [Havok 2009], Chipmunk 2D [Chipmunk 2009], OpenSteer [OpenSteer 2009], Newton Game Dynamics [Newton 2009], Tokamak [Tokamak 2009], PhysX [PhysX 2009] e ODE [ODE 2009].

Uma vez que não é possível tratar aqui das diversas *engines* de simulação física, optou-se por focar este trabalho na ODE (*Open Dynamics Engine*) [ODE 2009], onde esta é uma *engine* de código aberto – *Open Source*, que vem sendo amplamente utilizada junto a diversos ambientes de desenvolvimento de jogos e de simulação de veículos (e robôs móveis) em tempo real.

# 4. Simulação Física de Corpos Rígidos

A simulação física de corpos rígidos provê um elevado grau de realismo na simulação de veículos. Através do uso de *engines* física como a ODE, podemos integrar na simulação modelos cinemáticos e dinâmicos que irão definir o comportamento físico de corpos rígidos [Osorio 2006]. O programa de simulação de veículos passa a controlar o carro através da aplicação de forças (aceleração e frenagem) e de direcionamento (giro da direção). Enquanto isso, a *engine* física fica responsável por controlar e determinar a pose e trajetória do veículo, resultante da interação de diversas forças que atuam no carro. Além disto a engine física também irá controlar a interação dos diversos corpos presentes no ambiente de simulação (colisões e reação a colisões).

## 4.1 ODE – *Open Dynamics Engine*

A ODE (*Open Dynamics Engine*) [ODE 2009] é uma biblioteca de código livre para a simulação dinâmica de corpos rígidos articulados, desenvolvida por Russell Smith. A biblioteca ODE pode ser usada em ambientes Linux e Windows, usualmente através de programas escritos em C/C++, onde existem também diversas interfaces para com outras linguagens, como o Python (PyODE), por exemplo.

### 4.1.1 Componentes da ODE

O uso da ODE permite criar estruturas articuladas, onde estas são criadas ao se conectar vários elementos (corpos rígidos) de diferentes formas, unidos através de juntas de diferentes tipos. Assim é possível, por exemplo, unir partes móveis como rodas junto a um eixo e carroceria. A ODE provê a simulação física de forças que atuam nos corpos e juntas, obtendo resultados bastante realistas, como o apresentado na Figura 14.

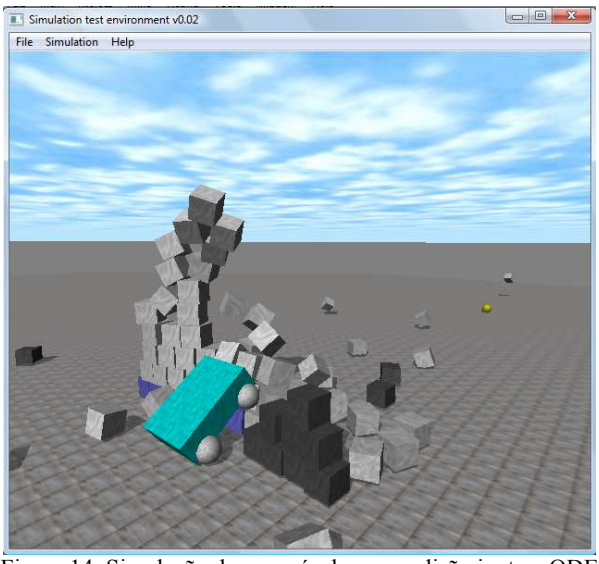



Figura 14: Simulação de um veículo com colisão junto a ODE

A ODE foi projetada para o uso interativo ou em simulações de tempo-real, sendo amplamente usada em jogos que usam simulação física. Esta ferramenta é particularmente muito boa para a simulação de objetos móveis em ambientes dinâmicos, sendo bastante rápida, robusta e estável. Além disto, o usuário tem uma completa liberdade para adaptar as estruturas (corpos e juntas) durante a própria simulação. A ODE usa um integrador de primeira ordem altamente estável, de modo que os erros de simulação não cresçam de modo ilimitado e de maneira fora do controle (um problema de certa forma usual em simulações físicas). Isto significa que o sistema não deve perder o controle da simulação, devido a uma propagação dos erros de cálculos.

A simulação é baseada em um método de integração que permite derivar os parâmetros de controle de força, posição, movimento e velocidades (angular e linear), usando multiplicadores de Lagrange, baseado no modelo de Trinkle & Stewart e Anitescu & Potra [Baraff 97, Smith 2006]. A ODE possui contatos rígidos, o que significa que restrições de não interpenetração são aplicadas quando dois corpos colidem. Esta ferramenta oferece um sistema de detecção de colisões com objetos do tipo esferas, caixas, cilindros abertos e planos. Além disto, a ODE também permite definir a distribuição de massa de corpos rígidos, e implementa modelos de contato e fricção, baseados no método *Dantzig LCP solver* [Baraff 97].

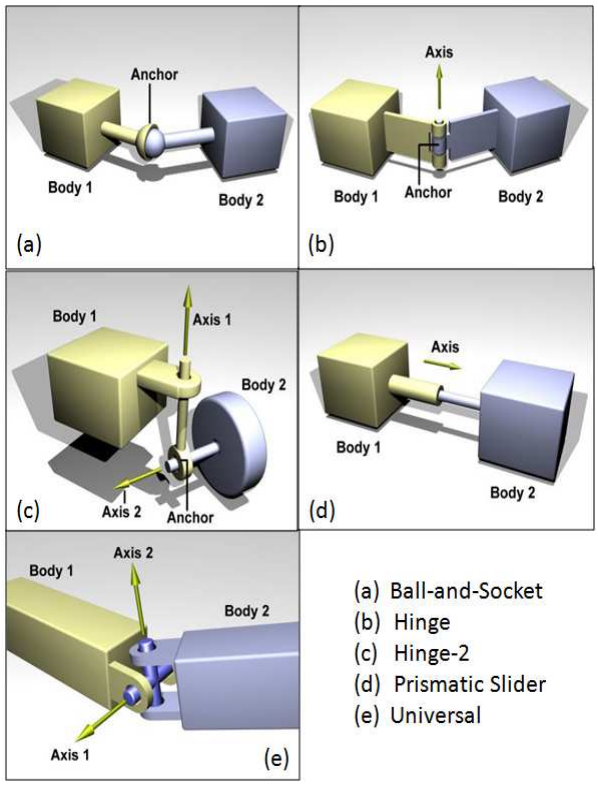



Figura 15: Juntas da ODE [Smith 2006]

Os tipo de juntas implementados na ODE são: *ball-and-socket* (articulação do tipo do ombro humano), *hinge* (dobradiça), *hinge-2* (duas juntas ligadas), *prismatic universal, fixed* (conexão fixa), *prismatic slider* (pistão) e *angular motor* (eixo com giro). No caso dos veículos as juntas do tipo *hinge-2* são usadas para criar as rodas do veículo e conectar estas ao restante do carro. A Figura 15 apresenta alguns exemplos de juntas da ODE [Smith 2004, Ode_Manual 2009].

Do ponto de vista da física, um corpo articulado é simplesmente uma composição de diversos corpos rígidos conectados através de juntas. Cada um destes corpos pode interagir com seus vizinhos, evitando colisões (e interpenetração): uma força ou um torque aplicado em um corpo irá afetar também seus vizinhos aos quais este corpo está conectado, onde todos os corpos devem ser capazes de se chocar uns com os outros (e sofrer uma reação em relação a este choque). Por fim, todos os corpos também são afetados por uma força de gravitação. Isto garante que se não houverem forças aplicadas sobre um corpo articulado, ele irá cair. Por outro lado, seu deslocamento também será a resultante da combinação destas diversas forças (forças aplicadas nos corpo e força gravitacional). Aplicando-se uma força adequada em diferentes elementos e articulações (motores), é possível assim evitar a queda de um corpo articulado, mantendo seu equilíbrio, de modo a controlar seu estado global, posição e inclinação em relação aos diferentes eixos. Em outras palavras, esta o uso desta ferramenta para simular corpos rígidos articulados irá garantir um maior realismo no comportamento físico dos objetos presentes em um ambiente virtual.

4.1.2 Simulação

Uma simulação típica baseada na ODE deverá ser realizada através dos seguintes passos [Smith 2006]:

1. Criar um mundo dinâmico.
2. Criar os corpos dentro do mundo dinâmico.
3. Ajustar o estado (posição, etc) de todos os corpos.
4. Criar as juntas dentro do mundo dinâmico.
5. Fixar as juntas aos corpos rígidos.
6. Ajustar os parâmetros de todas as juntas.
7. Criar o mundo de colisões e a geometria de colisões dos objetos, de acordo com o necessário.
8. Criar um grupo de juntas para manter as juntas de contato.
9. Laço de Simulação:
   a. Aplicar as forças aos corpos (se necessário)
   b. Ajustar os parâmetros das juntas (se necessário)
   c. Executar a chamada da detecção de colisão
   d. Criar uma junta de contato para cada ponto de colisão, e colocar ela no grupo de juntas de contato.
   e. Executar um passo de simulação
   f. Remover todas as juntas do grupo de juntas de contato.
10. Destruir o mundo dinâmico e de colisões.

O desenvolvimento de simulações usando a ODE pode ser feito através do uso das ferramentas de simulação e visualização (simples) oferecidas pela própria ODE, e de acordo com os passos descritos anteriormente. A ODE oferece uma ferramenta de visualização bastante simples que permite exibir em tempo-real o ambiente virtual e a evolução da simulação. O visualizador da ODE é denominado de "*drawstuff*".

Entretanto, em muitas aplicações de simulação, de realidade virtual e em jogos, o usuário está interessado em usar os recursos de simulação física da ODE, porém deseja implementar o seu próprio visualizador, com recursos mais avançados: inclusão de modelos 3D complexos, uso de sombras, efeitos de luz e partículas, adição de texturas, terrenos com uma modelagem mais complexa, além de cenários e paisagens mais realísticas do que as providas pelo *drawstuff*.

A ODE foi criada voltada também para este tipo de aplicações, onde o usuário pode desabilitar o *drawstuff*, aplicar a simulação física de corpos rígidos provida pela ODE e realizar a visualização através de suas próprias ferramentas (e.g. OpenGL, DirectX, OSG, DarkGDK, OGRE3D, Panda3D, Delta3D) [Game_Engines 2009].

Diversas aplicações, comerciais ou não, também fazem uso da ODE a fim de implementar suas simulações, como por exemplo na área da robótica temos os seguintes simuladores baseados na ODE: Player-Stage-Gazebo (Open Source), Microsoft Robotics Studio (MRS) e Webots da Cyberbotics [Robot_Simul 2009].

A integração da ODE com ambientes de simulação virtual é feita da seguinte forma:

1) Primeiramente é necessário criar 2 mundos duplicados, ou seja, um mundo de objetos do ambiente virtual (aplicação - VR) e um mundo de objetos no espaço da ODE (simulador - ODE);
2) Cada objeto do mundo virtual (VR) deve ser posicionado e orientado exatamente de acordo com o objeto correspondente a ele definido no mundo físico (ODE);
3) Executa-se o laço de simulação da seguinte forma:
   a) Inicia-se aplicando forças sobre os objetos, passando esta informação para o espaço da ODE;
   b) Então, a ODE irá integrar todos os dados numéricos e gerar novas posições e orientações de cada objeto como resultado da simulação física (aplicação de forças, torques, colisões, entre outros), atualizando as informações sobre cada objeto;
   c) Usando as informações atualizadas de posição e orientação de cada objeto no mundo físico (ODE), é possível então atualizar as informações de posição e orientação dos objetos do mundo virtual (VR).

As leis da física são aplicadas no espaço da ODE (e.g. colisão, cinemática, fricção, gravidade, entre outros), afetando os objetos, que depois são reposicionados no mundo gráfico, cujo foco será na visualização gráfica (e.g. texturas, sombras, luzes e formas).

4.1.3 Corpos e Juntas: Elementos e Propriedades

O texto acima descreveu as principais características incluídas na ODE. De modo a ter uma descrição mais completa desta ferramenta, vamos apresentar aqui uma descrição mais detalhada dos dois componentes principais da ODE: os corpos rígidos e as juntas.

**Corpos Rígidos**

Os corpos rígidos em uma simulação física da ODE possuem as seguintes propriedades:

- Um vetor de posição que corresponde ao centro de massa do corpo;
- Uma velocidade linear;
- Uma orientação espacial do corpo;
- Um vetor de velocidade angular, que descreve como a orientação muda através do tempo;

Além destas propriedades, que são constantemente atualizadas durante a simulação, existem outros valores que são definidos pelo usuário e não são mais alterados (constantes):

- A massa do corpo;
- A posição do centro de massa (relativa ao corpo);
- Uma matriz inercial que descreve como a massa do corpo está distribuída ao redor de seu centro de massa;
- Força da gravidade (pré-definida pelo sistema);

Baseado nestas informações, e nas forças e torques aplicados nos corpos, é que a ODE realiza a simulação do movimento dos objetos no mundo físico simulado.

**Juntas**

Os corpos rígidos são definidos em conjunto com os componentes que os unem: as juntas. As juntas podem ser de diferentes tipos, por exemplo, dobradiças (*hinge*), encaixes (*ball-and-socket*), eixos (*hinge2*) e juntas universais. A conexão entre os objetos é respeitada ("mantida") nas simulações, logo, quando um objeto que está conectado a outro é deslocado, este "arrasta consigo" os demais objetos aos quais está conectado. Isto permite que sejam criados corpos articulados, para fins de simulação de criaturas caminhantes, mas também de máquinas complexas, como por exemplo, empilhadeiras, guindastes e braços robóticos articulados. A simulação dos corpos articulados é simples do ponto de vista do implementador, que precisa apenas "unir" os corpos entre si, deixando para a ODE que esta garanta a coesão entre as partes, garantindo também que não ocorrerão colisões com interpenetração entre suas partes. A ODE também permite que sejam configurados parâmetros específicos das articulações e juntas, como por exemplo, definindo limites operacionais (ângulos limites) para as juntas. Além disto, o usuário pode adicionar motores nas juntas.

Conforme pôde ser visto nesta seção, através de uma ferramenta de simulação física como a ODE, podemos dotar um veículo de um comportamento físico. O comportamento físico do veículo está relacionado à sua movimentação no ambiente, através da aplicação de modelos cinemáticos e dinâmicos da física. Cabe destacar que o veículo irá "reagir" de modo "burro", porém realista, as forças que nele são aplicadas, sejam estas forças geradas externamente (e.g. gravidade) ou internamente (e.g. aceleração e frenagem). Portanto um veículo será capaz de se deslocar ou parar, dependendo das ações de um sistema de controle, que irá definir a forma de atuação sobre este veículo. Este sistema de controle usualmente é representado pelo usuário que controla o acelerador, freio e giro da direção do veículo, fazendo com que o carro se comporte da maneira desejada.

## 5. Agentes Autônomos Inteligentes

Os jogos digitais de corrida usualmente oferecem a possibilidade de competir contra outros oponentes, sejam eles reais (jogos multi-jogador em rede), ou virtuais (controlados pelo próprio computador). Nesta seção iremos abordar as questões relativas a implementação de oponentes virtuais controlados pelo próprio simulador, ou seja, a implementação de NPCs (*Non-Player Characters*) que controlam os veículos de forma autônoma.

Os veículos autônomos devem ser controlados por agentes autônomos inteligentes, de modo a dotar os veículos não apenas de um comportamento realista do ponto de vista da física, mas de um comportamento autônomo realista do ponto de vista da Inteligência Artificial (I.A.) [Osorio 2006, Osorio 2007]. Portanto, passamos do estudo do comportamento físico para o estudo do comportamento inteligente.

Figura 16: Veículo Móvel Autônomo – Percepção e Ação

Nesta seção, serão apresentados conceitos que usualmente são estudados junto à robótica móvel (Fig. 16), pois esta área também visa o desenvolvimento de veículos móveis autônomos e inteligentes [Wolf 2009].

## 5.1 Comportamento Inteligente

Os veículos autônomos precisam obter informações sobre o ambiente em que estão inseridos, e assim, tomar decisões sobre como vão agir, para então atuar sobre o veículo controlando sua velocidade e trajetória. A Figura 16 apresenta um exemplo do problema de controle de veículos autônomos que envolve este laço:

**Percepção => Planejamento e Decisão => Ação**

O desenvolvimento de algoritmos que permitam obter um comportamento inteligente para o controle de agentes móveis e veículos autônomos, vem sendo amplamente estudado junto a área de jogos [Osorio 2007, Bourg 2004, Buckland 2002, Rabin 2002, 2001]. O conceito de agente inteligente é adotado para implementar NPCs (agentes virtuais/humanóides), mas também para implementar os programas que controlam de maneira autônoma os entes que atuam em um ambiente virtual, como é o caso dos veículos autônomos.

Por definição, um agente autônomo deve ser capaz:

(i) de **perceber** o ambiente através de seus sensores; (ii) de **planejar e decidir** sobre como deve agir para atingir seus objetivos, considerando o seu estado atual e a sua percepção do estado atual do ambientes; (iii) de **agir**, realizando ações através de seus atuadores.

Portanto, um comportamento inteligente é o resultado da integração deste conjunto de elementos, a capacidade de percepção, de processamento a capacidade de usar as informações disponíveis para a tomada de decisões (raciocínio), e a capacidade de atuar realizando ações junto ao ambiente em que se encontra inserido.

### 5.1.1 Percepção

Um veículo autônomo deve ser capaz de desviar dos obstáculos ao mesmo tempo em que segue uma trajetória. Por muito tempo a tarefa de seguir uma trajetória foi baseada em informações "privilegiadas", ou seja, o veículo autônomo conhecia a sua posição e orientação exatas junto ao ambiente, conhecia previamente todos os obstáculos existentes (previamente mapeados), e, além disto, tinha definido perfeitamente um caminho através de uma rota de pontos por onde deveria passar com precisão. Então, bastava realizar o deslocamento seguindo ponto-à-ponto esta trajetória pré-definida, de forma determinística, totalmente controlada e sendo incapaz de tratar eventos imprevistos. Isto nos remete aos *vídeo games* como o *Pole Position* apresentado na Figura 3.

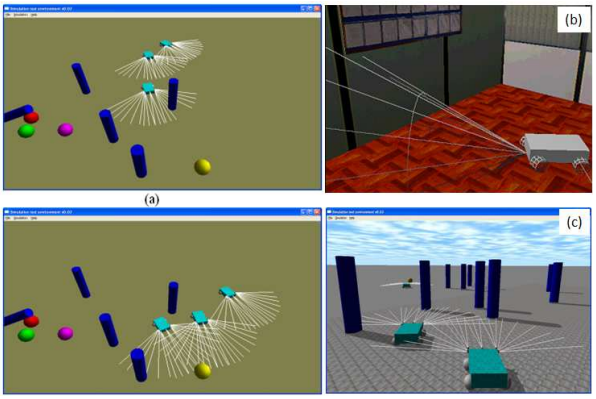
O problema desta abordagem é que ela não é realística: um ser humano, por maior que seja sua capacidade e inteligência, ao conduzir um veículo não dispõe da posição exata em que se encontra, não dispõe de uma previsão precisa da localização de todos os obstáculos do caminho, e nem é capaz de repetir uma trajetória previamente definida a ponto de seguir exatamente (sem erro algum) os pontos desta trajetória. Logo, se buscamos criar um agente autônomo capaz de conduzir um carro de modo similar a um piloto humano, devemos criar um agente inteligente que busque reproduzir o comportamento humano (podendo ser pior ou até melhor do que um humano).

O ser humano possui a capacidade de perceber o ambiente através de seu apurado sistema de visão, podendo estimar aproximadamente sua localização, detectar obstáculos a sua frente, e usando sua capacidade de memória e aprendizado, é capaz de aprender rotas e planejar caminhos. Quais são então as percepções que seriam mais adequadas para dar a um veículo a capacidade de se tornar autônomo? O ideal seria dotá-lo de um sistema de visão que tivesse uma performance tão boa quanto a do sistema de visão humano ao tratar informações visuais, no entanto, até o presente, os sistemas de visão computacionais ainda não conseguem realizar tarefas com um nível de complexidade, precisão, velocidade e sofisticação, comparado com que o cérebro humano é capaz de fazer. Sendo assim, podemos então adotar uma abordagem alternativa, a de buscar junto à robótica as técnicas e mecanismos que vem sendo usados para criar veículos móveis autônomos.

Os robôs móveis adotam sensores capazes de detectar obstáculos a sua frente, sendo usualmente equipados com sensores de proximidade infra-vermelhos, sonares e sensores a laser. Os sensores a laser são os mais precisos, onde por exemplo, o sensor SICK LMS 200 é capaz de medir com uma precisão de milímetros a distância até um obstáculos que esteja num raio de 80 metros ao seu redor. A Figura 17(b) apresenta uma foto deste equipamento. Além dos sensores de proximidade, um outro equipamento bastante usado é o GPS (*Global Positioning System*), que permite obter a posição aproximada e orientação (bússola virtual) do veículo, com um certo erro.


Figura 17: Sensores Robóticos

Portanto, ao invés de simplesmente obter a pose 3D exata do veículo, indicada nas variáveis que controlam sua posição e orientação na simulação, podemos obter as suas coordenadas e orientação de modo mais realista simulando dispositivos como o GPS e uma unidade inercial IMU. Além disto, ao invés de obter a posição dos demais veículos e obstáculos, seja porque temos acesso ao mapa do ambiente (preciso e exato), ou seja porque temos acesso as coordenadas dos demais veículos, podemos implementar um algoritmo que simule a percepção de obstáculos a nossa frente, simulando um sensor do tipo sonar ou laser (Fig. 18).




(a) e (c) Laser [Robombeiros] – (b) Sonar [SimRob3D]
Figura 18: Simulando Sensores Robóticos [Wolf 2009]

E quais são as vantagens desta abordagem? Primeiramente, o comportamento do agente não será um comportamento pré-programado e dependente de informações que ele não poderia possuir no mundo real. Em segundo lugar, esta abordagem busca um maior realismo comportamental, e para tanto, precisamos trabalhar mais com os dados sensoriais, extraindo destes dados as informações que permitam controlar melhor os veículos autônomos, assim como as pessoas fazem. É importante destacar e relembrar os conceitos que foram apresentados no início deste texto: a busca do realismo gráfico, depois do realismo físico e por fim do realismo comportamental. O busca por um elevado realismo comportamental passa pelo desenvolvimento de algoritmos e técnicas capazes de tratar informações sensoriais e traduzir estas informações em ações, sem no entanto recorrer as informações de "baixo nível" (usadas no controle da simulação, como a posição e orientação dos elementos do ambiente virtual).

Figura 19: Veículo real equipado com sensores e atuadores

Figura 20: Dados sensoriais coletados com o veículo real Laser + GPS + câmera

As Figuras 19 e 20 apresentam o veículo real do projeto SENA – Sistema Embarcado de Navegação Autônoma [SENA 2009], juntamente com dados reais adquiridos com o uso desta plataforma de experimentação. O veículo real permite desenvolver pesquisas na área de I.A., a fim de implementar agentes autônomos capazes de controlar um veículo virtual ou real, baseado em dados de sensores simulados ou reais. Assim, mesmo em um ambiente virtual, estamos trabalhando com alto grau de realismo e proximidade do modelo real concreto. Este exemplo demonstra uma interessante aproximação e troca de conhecimentos entre as áreas do desenvolvimento de jogos digitais, simulação virtual, inteligência artificial e robótica móvel autônoma.

### 5.1.2 Atuadores

Os atuadores são os motores que fazem com que o carro se desloque, sendo que um veículo deve ter atuadores para controlar a tração de suas rodas (acelerar – aumento da velocidade), atuadores para controlar a frenagem de suas rodas (desacelerar – diminuição da velocidade), e um atuador para controlar o giro da direção (direcionamento - *steering*), podendo opcionalmente também atuar na troca de marchas (câmbio manual/automático). Logo, as principais ações executadas pelo piloto são: acelerar, frear e controlar a direção.

Do ponto de vista de um agente autônomo, será necessário desenvolver um sistema capaz de, a partir dos dados sensoriais do ambiente e de uma estimativa do estado do veículo (e.g. pose 3D estimada), gerar comandos de aceleração/frenagem e de direcionamento do veículo. Este é o papel do controlador do veículo: controlar a trajetória e realizar a navegação de modo autônomo, a fim de executar uma tarefa pré-definida.

### 5.1.3 Controle Inteligente

O controle de um veículo autônomo pode ser implementado por meio de diferentes estratégias. Existem diversos métodos de controle descritos na literatura sobre agentes autônomos inteligentes, onde encontramos usualmente uma classificação das arquiteturas de controle em: (a) arquitetura de controle deliberativo, (b) arquitetura de controle reativa, (c) arquitetura de controle em camadas, (d) arquitetura de controle híbrida [Wolf 2009, Osorio 2007, Jung 2005].

Estas arquiteturas de controle são implementadas fazendo uso de diferentes métodos, como por exemplo, algoritmos capazes: de realizar um planejamento global de trajetórias (deliberativa); de navegar evitando obstáculos localmente (reativa); de priorizar e selecionar diferentes comportamentos previamente especificados conforme cada situação (camadas); de perceber o ambiente, mapear seus elementos, planejar e agir (camadas); de realizar o seguimento de trajetórias previamente definidas evitando os obstáculos imprevistos (híbrida). Algoritmos como estes citados acima são amplamente estudados junto às áreas de Inteligência Artificial e de Robótica [Mataric 2007, Siegwart 2004, Dudek 2000, Russel 1995].

Para que se possa estudar e desenvolver algoritmos de controle inteligente para veículos autônomos, alguns conceitos devem ser previamente discutidos. Primeiramente é importante diferenciar os algoritmos baseados puramente em informações locais e algoritmos baseados em informações globais.

**Algoritmos baseados em Informações Locais**

O uso apenas de informações locais permite que sejam implementados algoritmos reativos, cuja principal característica é sua simplicidade. Os algoritmos reativos, como diz seu nome, reagem diretamente as percepções de informações disponíveis localmente. Usualmente este tipo de algoritmos permite a implementação de comportamentos mais básicos, uma vez que um sistema reativo não possui a princípio uma visão global do ambiente e nem da tarefa que está realizando. Todo o conhecimento de que os algoritmos precisam para realizar o controle do veículo é baseado em informações locais.

Como exemplos de algoritmos reativos baseados em informações locais, temos os comportamentos de evitar/desviar de obstáculos, seguir uma parede (ou o meio-fio da calçada, ou a marcação da borda da pista – *line follow*), seguir em uma direção (indicada por uma bússola ou fonte de luz, por exemplo), acompanhar um outro agente ou veículo (*follow-me*), ou então, compor um comportamento um pouco mais complexo a partir de diferentes comportamentos básicos. Um exemplo de composição de comportamentos é o de seguir em uma direção, porém desviando de obstáculos.

Tais comportamentos podem ser implementados de diversas maneiras, seja através de um simples conjunto de regras (*If-Then*) que ajustam a trajetória de acordo com as percepções externas, ou baseados em uma referência externa (*setpoint*) e no ajuste e correção da trajetória através de um controlador de malha fechada (e.g. controladores P, PI e PID [PID 2009]), ou até mesmo através do uso de sofisticados algoritmos de Aprendizado de Máquina (*Machine Learning*) [Mitchell 97, Rezende 2003, Buckland 2002, Osorio 2007].

Alguns algoritmos usados para implementar comportamentos reativos, e que são bastante conhecidos são: Algoritmos Reativos para Seguir Linhas (*Reactive Line Folllowing*) [Jones 2003, Cook 2002], Campos Potenciais [Siegwart 2004, Heinen 2002], Algoritmos de Direcionamento de Grupos (*Reynolds´ Steering Behaviors* [OpenSteer 2009]), Algoritmos de composição de direcionamento e desvio de obstáculos reativo [Dudek 2000, Pessin 2008]. Todos estes algoritmos podem ser implementados com robôs móveis reais, mas também através de ambientes de simulação, como é o caso dos jogos de corrida.

Um dos algoritmos mais conhecidos usados para o controle de carros é através do ajuste da trajetória do veículo baseada em uma trajetória virtual. Esta trajetória virtual é denominada de "*way-point*", ou seja, uma seqüência de pontos que definem o caminho que deveria ser seguido (aproximadamente). Encontramos na literatura sobre jogos (e de robótica) uma extensa bibliografia descrevendo técnicas baseadas em *way-points*, a exemplo das citadas nos livros da série "*A.I. game programming wisdom*" [Rabin 2001, 2002].

Esta trajetória virtual, definindo um caminho em uma pista de corrida, tem um paralelo junto ao mundo real: a captura de pontos de GPS (*Global Positioning System*) que permitem também definir uma trajetória sobre um mapa (Fig. 20). Usualmente a maior diferença entre um *way-point* em um jogo digital e um *way-point* capturado no mundo real (GPS) é relativa a precisão da trajetória. Em um jogo podemos usar pontos exatos, simplesmente listando coordenadas precisamente escolhidas que definem uma trajetória, entretanto no mundo real, as coordenadas de um sistema de GPS são aproximadas e usualmente temos um erro associado na ordem de alguns metros (2,5 a 5 metros de imprecisão).

Figura 20: Trajetória real de um veículo com pontos GPS

Note que, se o objetivo é reproduzir as características do mundo real, o mundo simulado deve obrigatoriamente considerar esta imprecisão nas coordenadas, tanto da trajetória, quanto da posição atual do veículo obtida através de um dispositivo como o GPS. Um erro de apenas 2,5 metros em uma via de duas mãos (ver Fig. 20) já é suficiente para provocar uma colisão!

O método dos *way-points* é um método que pode ser classificado como local, se considerarmos que o veículo somente recebe o próximo ponto e se dirige em direção a ele (*setpoint*). No entanto, muitas vezes, é definido o trajeto como um todo e este é conhecido pelo sistema de controle do veículo, neste caso, passamos de um processamento de informações locais (e navegação reativa) para algoritmos baseados em rotas previamente definidas e conhecidas, adotando algoritmos baseados em informações globais.

**Algoritmos baseados em Informações Globais**

Dois conceitos muito importantes de navegação com planos globais são referentes à localização e ao uso de mapas. Inicialmente é necessária a obtenção de um mapa do local onde o veículo irá se deslocar, no caso de corridas, usualmente a descrição da pista é o elemento que fornece as informações sobre a trajetória a ser seguida. De posse de um mapa, e sabendo a localização atual do veículo (pose 2D ou 3D), é possível definir uma trajetória da posição inicial atual até uma determinada posição destino, navegando através dos elementos descritos no mapa.

O planejamento prévio de trajetórias, com o uso de um mapa que permita que se conheça antecipadamente as posições de todos os elementos presentes no ambiente, é uma característica dos algoritmos deliberativos. O algoritmo, de posse da posição inicial do veículo e de um mapa, é capaz de deliberar, planejando previamente a seqüência de ações que lhe permitem navegar da posição atual até uma posição de destino. Note que este tipo de algoritmos exige um conhecimento com informações globais sobre o estado do veículo e do ambiente.

Alguns dos algoritmos de planejamento de trajetórias mais famosos são o A* (A Star) e seus derivados (D Star), e os algoritmos baseados no espaço de configuração e obtenção do menor caminho (Djikstra). Estes algoritmos foram extensivamente apresentados e discutidos em tutoriais apresentados nas edições anteriores do SBGames [Osorio 2007]. É importante que se destaque que tais algoritmos possuem certas restrições, notadamente, a necessidade de ter um conhecimento global, estático e preciso do ambiente (qualquer mudança implica em um re-planejamento da trajetória), além de ser necessário que se trabalhe com um posicionamento preciso dos elementos envolvidos: carros, pista e obstáculos. Um erro de posicionamento pode levar a um desastre, pois as ações já foram previamente definidas!

**Algoritmos de Controle Híbridos**

Uma vez que as informações locais muitas vezes são bastante limitadas, e que as informações globais são sujeitas as imprecisões locais... uma das estratégias mais usuais é a integração de um plano "macro" de navegação (deliberativo), baseado em informações globais, juntamente a um processamento local que permite reagir aos obstáculos imprevistos (reativo), criando-as assim um sistema híbrido.

Os sistemas híbridos costumam tirar proveito das vantagens de diferentes técnicas, de modo a que sua integração permita que as vantagens de uma compensem as desvantagens das outras. É exatamente isto que é feito ao se integrar estratégias locais baseadas em leituras dos sensores (e reação a obstáculos imprevistos), juntamente com estratégias globais baseadas em mapas que permitem criar trajetórias mais adequadas e inteligentes.

Um exemplo de arquitetura híbrida de robótica que foi proposta com este objetivo, é a arquitetura COHBRA (Controle HíBrido de Robôs Autônomos) [Heinen 2002, 2002a]. A arquitetura Cohbra propõe a integração de mapas com uso de planejamento do tipo A* juntamente com comportamentos reativos do tipo campos potenciais, que permitem evitar colisões. A arquitetura Cohbra também provê recursos para manter o controle da localização do robô móvel (estimativa da pose do robô), baseada nas percepções lidas por meio de seus sensores (auto-localização).

É importante destacar que a arquitetura híbrida acima citada é apenas mais uma proposta de controle de robôs móveis autônomos. Existem diversos algoritmos e técnicas usadas para realizar o controle, tratando questões que vão da localização e mapeamento simultâneo do ambiente até o uso de informações visuais para a estimativa da posição e deslocamento dos robôs. Qual é a melhor técnica? Não é possível afirmar atualmente que exista uma melhor técnica, visto que os desafios são muitos e ainda existe muito a ser desenvolvido nesta área.

**Desafios na Implementação de Algoritmos de Controle Inteligente**

Os principais desafios da implementação de algoritmos inteligentes de controle de veículos autônomos dizem respeito as características do mundo real: imprecisão e erros de leitura dos sensores, imprecisão na localização do veículo (manter controle da pose exata do veículo), imprecisão do mapa, incapacidade de planejar todas as ações antecipadamente devido a ocorrência de eventos imprevistos, necessidade de reagir de modo rápido a diversos eventos, necessidade de interagir com os demais elementos do ambiente. *Um simulador realista deve, portanto, reproduzir no mundo virtual estas características do mundo real, vinculadas a imprecisão, incerteza, tolerância a falhas e reação a eventos imprevistos.*

Em resumo, um algoritmo de controle inteligente deve ser robusto para que mesmo diante de um ambiente parcialmente conhecido, que é percebido com muita imprecisão, ainda assim agir de forma a atingir corretamente todos os seus objetivos, ou pelos menos desempenhar suas tarefas com uma performance adequada.

## 6. Desafios da I.A. em Jogos de Corrida

Um dos grandes desafios da I.A. em jogos digitais é a criação de agentes autônomos inteligentes, onde estes personagens (NPCs) devem ser capazes de interagir com o ambiente de "modo inteligente" [Osorio 2007]. Em particular nos jogos de corrida, os agentes autônomos inteligentes (Fig.21) irão assumir o papel de piloto dos carros controlados de forma autônoma. Um piloto virtual deve reproduzir (simular) o comportamento de um piloto humano, ou seja, por meio de suas percepções ele deve ser capaz de tomar as decisões e agir da melhor forma possível para controlar a trajetória do veículo e sua dinâmica.

Os jogos de corrida se tornam um grande desafio para a I.A., pois cada vez mais aproximamos os jogos digitais de uma simulação bastante realista de uma corrida real. Nos simuladores são inseridos elementos inspirados no mundo real (e.g. direção, acelerador, troca de marchas) e em componentes do comportamento físico real dos veículos - cinemática e dinâmica (e.g. inércia, derrapagem, aceleração, frenagem).

Portanto, o desenvolvimento de um sistema de Inteligência Artificial robusto e de bom desempenho torna-se uma tarefa extremamente desafiadora e interessante do ponto de vista da pesquisa científica nesta área. Este desafio tem motivado diversas iniciativas que buscam promover o desenvolvimento de algoritmos e técnicas nesta área.

Figura 21: Agentes Inteligentes em competições de Futebol de Robôs (Reais e Simulados) – SBIA 2008

### 6.1 Competições de I.A.: Agentes e Veículos Autônomos

Existem diversas competições que vem sendo propostas com o intuito de desenvolver a área de I.A., e mais especificamente os agentes autônomos. Uma destas competições que se tornou bastante famosa é a RoboCup [Robocub 2009] que visa desenvolver robôs capazes de jogar uma partida de futebol, sendo que atualmente existem ligas da RoboCup com o uso de robôs reais (com rodas e com pernas) e de robôs simulados. A RoboCup é mais um exemplo de competição que vem sendo desenvolvida em dois níveis, através de simuladores e através do uso de robôs reais. Os robôs da RoboCup são também dotados de sensores, da capacidade de decisão e ação, de modo a executar sua missão: jogar uma partida de futebol. A proposta da RoboCup é de até o ano de 2050 termos uma disputa de uma partida de futebol (de igual nível) entre humanos e robôs.

Uma outra competição muito conhecida dos pesquisadores de I.A. e de robótica é o *DARPA Challenge* [Gibbs 2006]. Este desafio visa desenvolver veículos capazes de seguir uma trajetória de um modo autônomo e robusto. O *DARPA Challenge* ficou conhecido inicialmente pelo primeiro desafio o *DARPA Grand Challenge* [Thrun 2006], realizado no deserto onde ocorreu um grande *rally* de veículos autônomos (Fig. 22). Nesta competição diversas equipes desenvolveram veículos dotados de sensores e capazes de seguir uma trajetória virtual definida por um *way-point* (pontos de GPS). O sucesso desta iniciativa estimulou a criação 2 anos mais tarde de uma nova competição, o *DARPA Urban Challenge*, onde os veículos deveriam trafegar em um ambiente urbano [Urmson 2008], respeitando as regras de tráfego deste tipo de ambiente.


Figura 22: Agentes Inteligentes em competições de veículos autônomos – Vencedor do DARPA Grand Challenge 2005

O desafio nestas competições é o de controlar um veículo dotado de sensores (como os sensores laser que aparecem na Fig. 22) e capazes de realizar uma tarefa de navegação autônoma por meio de uma rota pré-definida. Apesar da rota ser pré-definida era necessário evitar obstáculos, evitando colisões com elementos do ambiente e mesmo contra os demais veículos que participavam da corrida. O DARPA Challenge é considerado um dos grandes desafios da I.A. moderna.

Como era de se esperar, começaram a surgir diversas competições para o desenvolvimento da I.A. de veículos autônomos, sejam eles reais ou simulados. Em 2007 foi proposto junto ao Simpósio CIG (*Computational Intelligence in Games*) [CIG 2007] uma competição baseada na simulação de corridas (Fig. 23), e a partir desta competição, em 2008 e 2009 diversas outras competições de simulação de corridas foram sendo propostas em eventos como o CIG, CEC e GECCO.

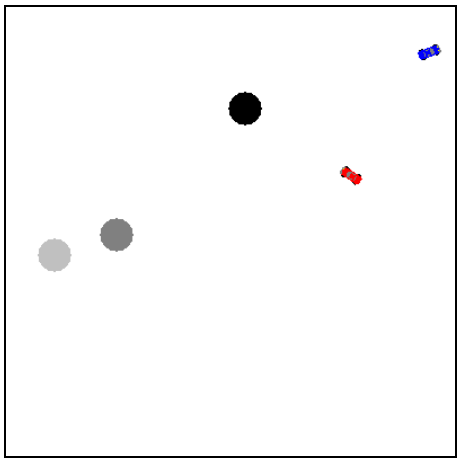


Figura 23: CIG 2007 Car Racing Competition [CIG 2007]

Os simuladores de corrida evoluíram bastante nestes últimos anos, e atualmente as competições de agentes inteligentes (I.A.) para o controle de veículos em corridas simuladas, adotam simuladores bem mais realistas (Fig. 24). Os simuladores agora incluem elementos de simulação física (cinemática e dinâmica) dos veículos, uma interface gráfica bem mais realista, e o que é mais importante, reproduzem dados de sensores mais próximos aos dados adquiridos por sensores reais usados em sistemas robóticos.


Figura 24: CIG 2009 Car Racing Competition [CIG 2009]

## 6.2 CIG 2009 e TORCS

As diversas competições e desafios como o Darpa Challenge, a Robocup, e muitas outras, como por exemplo, "CiberRato e MicroRato" (Portugal), "*Trinity College Fire-Fighting Home Robot Contest*" (Trinity College – Hartford Connecticut), "*IEEE Latin American Robotics Contest for Students* – LARC" (SBIA 2008 – Salvador), "Olimpíada Brasileira de Robótica – OBR" (SBC, SBA), e as Competições de *LEGO Robotics*, tem servido de inspiração e alavancado o desenvolvimento na área da robótica.

Por outro lado, as competições e desafios de desenvolvimento de jogos digitais também tem atraído um grande número de entusiastas: Global Game Jam (IGDA), *GameJam Competition* (SIGGRAPH), *Imagine Cup* (XNA Games - Microsoft), 72 Hour GDC – *Game Development Competition*, *Ludum Dare* (Mike Kasprzak), *Mobile Games Innovation Challenge* (Nokia), entre muitas, e muitas outras.

Em meio a estas competições de robótica e de desenvolvimento de jogos, surge uma competição em particular, com uma proposta muito interessante: unir Jogos Digitais, Inteligência Artificial e Agentes Inteligentes e Autônomos, tudo junto em um grande desafio – ***The Simulated Car Racing Competition***.

Neste ano (2009), a competição "T*he Simulated Car Racing Competition*" irá ocorrer junto a grandes conferências (CEC, GECCO e CIG), onde destacamos o IEEE CIG. O simpósio IEEE CIG é o "*The IEEE Symposium on Computational Intelligence and Games 2009*" patrocinado pela *IEEE Computational Intelligence Society* (IEEE CIS). Este evento conta com diversas competições, destacando-se a competição de corridas simuladas.

O objetivo desta competição é desenvolver um controlador inteligente capaz de dirigir de modo autônomo um carro simulado, usando como base a ferramenta TORCS – *The Open Race Car Simulator*, que provê toda a base para a simulação das corridas.

### 6.2.1. TORCS

O TORCS é uma ferramenta multi-plataforma, altamente portável (Linix, Windows e MacOs), de simulação de corridas de carros. Esta ferramenta é utilizada como um jogo de corrida de carros sendo distribuída sob a licença GNU GPL (Código Aberto). O fato de ser um *software* aberto tem contribuído para o estudo e desenvolvimento de plataformas de simulação de corridas, mas também permitiu a adoção desta plataforma como uma ferramenta de pesquisa na área de I.A. para jogos de corrida.

O TORCS provê mais de 50 diferentes carros e mais de 20 pistas, contando com 50 diferentes oponentes contra quem podemos competir. Esta ferramenta permite o controle dos veículos por meio de um *joystick* ou com o uso de um volante para jogos, ou mesmo, por meio do uso de mouse e teclado. Os gráficos são realistas, incluindo efeitos de iluminação, fumaça, marcas de derrapagem e de travadas. A simulação inclui modelos de dano simples ao carro com colisões, definição de propriedades de pneus e rodas, de aerodinâmica e muitos outros parâmetros que afetam a performance dos carros. O TORCS implementa corridas do tipo "*single user*" e "*multi-user*", suportando até quatro jogadores humanos, em competições de corridas simples até campeonatos. A instalação é simples e o jogo pode depois ser fácilmente usado, como os demais similares jogos de corrida.

6.2.2. *Software* para a Competição

Uma vez instalado o TORCS, podemos instalar um "*patch*" de atualização que prepara o TORCS para receber adições usadas nas competições de corridas simuladas do CIG-CEC-GECCO. Este *patch* instala componentes que permitem que sejam desenvolvidos módulos externos de controle dos carros do TORCS. É adotada uma arquitetura do tipo cliente-servidor conforme apresentado na figura abaixo (Fig. 25).

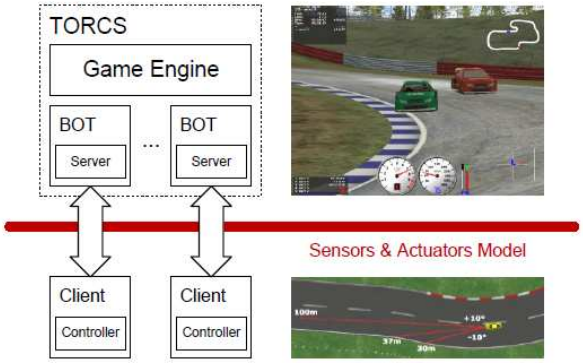



Figura 25: Arquitetura do Simulador
Simulated Car Racing Competition [CIG_Manual 2009]

Os bots (NPCs) são controlados por controladores desenvolvidos em C/C++ pelos usuários que participam da competição. Um controlador possui acesso a certo número de sensores, e pode enviar comandos de controle aos atuadores do carro. Portanto, um carro simulado no TORCS irá se comportar como um carro real sendo controlado por um Agente Inteligente. O cliente não possui informações privilegiadas sobre a pista ou localização do carro, onde deve controlar o veículo baseado apenas nas informações obtidas pelos seus sensores (e providas pelo servidor).

Os sensores disponíveis ao usuário são:

- **Angle**: ângulo entre a direção do carro e a direção do eixo da pista (atua como uma bússola associada a uma informação local de orientação da pista);
- **Curlaptime**: tempo decorrido na volta corrente;
- **Damage**: dano atual do carro (quanto maior o dano, maior é o valor desta variável);
- **DistFromStart**: distância percorrida pelo carro a partir da linha de partida;
- **DistRaced**: distância total percorrida desde o início da corrida;
- **Fuel**: nível atual de combustível;
- **Gear**: marcha corrente;
- **LastLapTime**: tempo da última volta;
- **Opponents**: vetor de 36 sensores que detecta os oponentes (simula um sensor laser que detecta especificamente os carros dos oponentes). Cada sensor cobre 10 graus com uma distância de até 100 metros, retornando a distância até o oponente mais próximo dentro de sua área perceptiva;
- **RacePos**: Posição na corrida em relação aos demais oponentes;
- **Rpm**: Número de rotações por minuto do motor;
- **SpeedX**: Velocidade do carro, considerando o sentido longitudinal do carro;
- **SpeedY**: Velocidade do carro, considerando o sentido transversal do carro;
- **Track**: vetor de 19 "*range finder sensors*" (sensores laser), capazes de detectar a distância até a borda da pista, com uma precisão de até 100 metros. Este sensor faz uma varredura de 10 em 10 graus com uma abertura de 0 a 180 graus (frontal).
- **TrackPos**: distância entre o carro e o centro do eixo da pista. A largura da pista é normalizada;
- **WheelSpinVel**: 4 sensores que medem de modo independente a velocidade de rotação das rodas.

Os atuadores disponíveis são:

- **Accel**: pedal do acelerador virtual, indicando o grau de aceleração (note que assim como o pedal do acelerador, este pedal tem um intervalo de atuação, de "não pressionado" até "pisando a fundo";
- **Brake**: pedal do freio virtual;
- **Gear**: Troca de marchas (são usadas 7 marchas, mais neutro e ré);
- **Steering**: Valor do giro da direção (ângulo de esterçamento da direção);
- **Meta**: Sinaliza para o servidor reinicializar a corrida.

O usuário/participante da competição deve escrever o código em C/C++ do programa cliente do TORCS, que irá ler os sensores, planejar e decidir as ações a serem tomadas, e enviar comandos para os atuadores do veículo simulado.

Constata-se que o TORCS permite que sejam realizados estudos relativos a implementação de agentes autônomos inteligentes capazes de controlar um veículo simulado. Os algoritmos de desenvolvidos junto ao TORCS permitem que sejam aperfeiçoadas técnicas de I.A., as quais podem em uma etapa posterior servir para o desenvolvimento de algoritmos de controle de veículos reais não tripulados.

Portanto a ferramenta TORCS é uma excelente plataforma, seja para o estudo e desenvolvimento de jogos digitais de corrida, seja para o estudo e pesquisa de novos algoritmos de I.A. usados no controle de veículos autônomos.

## 7. Conclusões

Este tutorial teve por objetivo apresentar conceitos relativos ao desenvolvimento de jogos de corrida, implementados como simuladores virtuais realísticos com processamento em tempo real. Também se buscou demonstrar a possibilidade da aplicação de jogos digitais e destes simuladores de corrida para o uso junto as pesquisa de I.A. e de Robótica.

Por fim, este trabalho visou demonstrar aos desenvolvedores de jogos a grande importância desta área de simulação de veículos e jogos de corrida, buscando motivar futuros desenvolvedores, e apresentando as perspectivas que se abrem no estudo e pesquisa junto a esta área.

## Agradecimentos

Os autores gostariam de agradecer pelas contribuições para este trabalho por parte dos membros do Projeto SENA - Sistema Embarcado de Navegação Autônoma (Profs. Marcelo Becker, Glauco Caurin, Valdir Grassi Jr. e Daniel Varela Magualhães), membros do GPVA – Grupo de Pesquisa em Veículos Autônomos (Profs. Farlei Heinen, Christian Kelber, Cláudio Jung e Milton Heinen), e do Grupo FOG (*Fellowship of the Game* – Grupo de alunos que trabalham com Jogos no ICMC-USP). Também gostaríamos de agradecer pelo apoio e suporte financeiro dos órgãos de fomento a pesquisa, CNPq e FAPESP, e pelo seu financiamento ao INCT-SEC, processos 573963/2008-8 e 08/57870-9 (Instituto Nacional de Ciência e Tecnologia em Sistemas Embarcados Críticos), e por fim ao LRM-ICMC-USP (Laboratório de Robótica Móvel do ICMC USP) e seus membros pelo apoio e recursos disponibilizados.

## Referências

BARAFF, B. AND WITKIN, A. (1997). *PHYSICALLY BASED MODELING: PRINCIPLES AND PRACTICE* (ONLINE) SIGGRAPH '97 COURSE NOTES, CARNEGIE MELLON UNIVERSITY (CMU), PITTSBURGH, CA. AVAILABLE AT: HTTP://WWW.CS.CMU.EDU/~BARAFF/SIGCOURSE/

BOURG, DAVID. *PHYSICS FOR GAME DEVELOPERS*. O'REILLY MEDIA, INC.; 1ST EDITION (NOVEMBER 15, 2001). 336P. ISBN-10: 0596000065.

BOURG, DAVID. *AI FOR GAME DEVELOPERS*. O'REILLY MEDIA, INC.; O'REILLY MEDIA, INC. (JULY 23, 2004). 390P. ISBN-10: 0596005555.

BUCKLAND, MAT. *AI TECHNIQUES FOR GAME PROGRAMMING*. PREMIER PRESS, GAME DEVELOPMENT SERIES. 2002. 481 P.

CONGER, DAVID. *PHYSICS MODELING FOR GAME PROGRAMMERS*. PREMIER PRESS - COURSE TECHNOLOGY PTR, 2004. ISBN-13: 978-1592000937.

COOK, DAVID. *ROBOT BUILDING FOR BEGINNERS*. APRESS; 1 EDITION (JANUARY 18, 2002). 600P. ISBN: 978-1893115446

DUDEK, G.; JENKIN, M. *COMPUTATIONAL PRINCIPLES OF MOBILE ROBOTICS*. CAMBRIDGE, LONDON, UK: THE MIT PRESS, 280 P. , 2000.

GIBBS, W. (2006) *INNOVATIONS FROM A ROBOT RALLY*, SCIENTIFIC AMERICAN. VOL.294, JANUARY 2006, P.64-71.

HEINEN, FARLEI (2002A). *SISTEMA DE CONTROLE HÍBRIDO PARA ROBÔS MÓVEIS AUTÔNOMOS*. UNISINOS – PIPCA – DISSERTAÇÃO DE MESTRADO EM COMPUTAÇÃO APLICADA. SÃO LEOPOLDO, RS. DISPONÍVEL ON-LINE EM: HTTP://BDTD.UNISINOS.BR/ OU SITE DO SIMROB3D: HTTP://NCG.UNISINOS.BR/ROBOTICA/ ACESSO: 08/02/2009.

HEINEN, FARLEI ; OSÓRIO, FERNANDO S. (2002). HYCAR - A ROBUST HYBRID CONTROL ARCHITECTURE FOR AUTONOMOUS ROBOTS. IN: HIS 2002 - HYBRID INTELLIGENT SYSTEMS, 2002, SANTIAGO DO CHILE. SOFT COMPUTING SYSTEMS - DESIGN, MANAGEMENT AND APPLICATIONS. FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS SERIES.. AMSTERDAM : IOS PRESS, 2002. V. 87. P. 830-840. WEB: (USER: "USP" - PASSWORD: "GUEST") HTTP://OSORIO.WAIT4.ORG/PUBLICATIONS/PAPERS-OSORIO.HTM ACESSO: 08/02/2009.

HEINEN, MILTON ROBERTO ; OSÓRIO, FERNANDO S. ; HEINEN, FARLEI ; KELBER, CHRISTIAN . *SEVA3D: AUTONOMOUS VEHICLES PARKING SIMULATOR IN A THREE-DIMENSIONAL ENVIRONMENT*. INFOCOMP (UFLA), V. 6, P. 63-70, 2007.

HOLZNER, STEVE. *PHYSICS FOR DUMMIES*. WILEY PUBLISHING - FOR DUMMIES. 384P., 2005. ISBN: 978-0-7645-5433-9.

JONES, JOE; ROTH, DANIEL. *ROBOT PROGRAMMING : A PRACTICAL GUIDE TO BEHAVIOR-BASED ROBOTICS*. MCGRAW-HILL / TAB ELECTRONICS; 1ST. EDITION (DECEMBER 12, 2003). 288P. ISBN: 978-0071427784

JUNG, C. R.; OSÓRIO, F. S.; KELBER, C.; HEINEN, F. (2005) *COMPUTAÇÃO EMBARCADA: PROJETO E IMPLEMENTAÇÃO DE VEÍCULOS AUTÔNOMOS INTELIGENTES*, IN: ANAIS DO CSBC'05. XXIV JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA (JAI). SÃO LEOPOLDO, RS: SBC, V. 1, P. 1358–1406. (ACESSO EM 08/02/2009). WEB: HTTP://OSORIO.WAIT4.ORG/PALESTRAS/JAI2005.HTML

MATARIC, MAJA J. THE ROBOTICS PRIMER. THE MIT PRESS. SEPTEMBER 2007, 288 PP. ISBN: 978-0-262-63354-3.

MITCHELL, T. M. *MACHINE LEARNING*. NEW YORK: MCGRAW-HILL. SERIES IN COMPUTER SCIENCE, 1997. 414P.

PALMER, GRANT. *PHYSICS FOR GAME PROGRAMMERS*. APRESS; 1 EDITION (MAY 4, 2009). 472P. ISBN-10: 159059472X.

OSÓRIO, FERNANDO S. ; MUSSE, SORAIA RAUPP ; VIEIRA, RENATA ; HEINEN, M. R. ; PAIVA, D. C. . *INCREASING REALITY IN VIRTUAL REALITY APPLICATIONS THROUGH PHYSICAL AND BEHAVIOURAL SIMULATION*. IN: X. FISCHER. (ORG.). RESEARCH IN INTERACTIVE DESIGN - PROCEEDINGS OF THE VIRTUAL CONCEPT CONFERENCE 2006. 1 ED. BERLIM: ESTIA - VIRTUAL CONCEPT - SPRINGER VERLAG, 2006, V. 2, P. 1-45. AVAILABLE AT: HTTP://OSORIO.WAIT4.ORG/PUBLICATIONS/OSORIO-ET-AL-VIRTUAL-CONCEPT2006.PDF (USER: "USP" - PASSWORD: "GUEST") OU HTTP://OSORIO.WAIT4.ORG/OLDSITE/PALESTRAS/VR-PBSIM.HTML [ACESSO 01/09/2009]

OSÓRIO, FERNANDO; PESSIN, GUSTAVO; FERREIRA, SANDRO; NONNENMACHER, VINÍCIUS. *INTELIGÊNCIA ARTIFICIAL PARA JOGOS: AGENTES ESPECIAIS COM PERMISSÃO PARA MATAR... E RACIOCINAR!* TUTORIAL SBGAMES 2007 (COMPUTING TRACK). SÃO LEOPOLDO, RS. DISPONÍVEL EM: HTTP://WWW.INF.UNISINOS.BR/~SBGAMES/ANAIS/TUTORIALS/ [Acesso 01/09/2009]

PESSIN, GUSTAVO (2008). *EVOLUÇÃO DE ESTRATÉGIAS E CONTROLE INTELIGENTE EM SISTEMAS MULTI-ROBÓTICOS ROBUSTOS.* DISSERTAÇÃO DE MESTRADO (ORIENTADOR: F.OSÓRIO) – PPG EM COMPUTAÇÃO APLICADA DA UNISINOS: SÃO LEOPOLDO, RS. DISPONÍVEL ON-LINE: HTTP://BDTD.UNISINOS.BR/ OU HTTP://PESSIN.GOOGLEPAGES.COM/ (ACESSO EM 08/02/2009).

RABIN, STEVE (EDITOR). *AI GAME PROGRAMMING WISDOM.* SECTION 9: "RACING AND SPORTS AI". PUBLISHED BY CHARLES RIVER MEDIA; 2002, 672 P. (WEB: HTTP://WWW.AIWISDOM.COM/ ). ISBN-10: 1584500778.

RABIN, STEVE (EDITOR). *AI GAME PROGRAMMING WISDOM 2.* SECTION 2 "PATHFINDING AND MOVEMENT" AND SECTION 8 "RACING AND SPORTS AI". PUBLISHED BY CHARLES RIVER MEDIA; 2003, 732 P.WOODCOCK, S., 2001.

REZENDE, SOLANGE (ED.). *SISTEMAS INTELIGENTES: FUNDAMENTOS E APLICAÇÕES.* BARUERI: EDITORA MANOLE, 2003. 525 P.

RUSSEL, R.; NORVIG, P. *ARTIFICIAL INTELLIGENCE: A MODERN APPROACH. ENGLEWOOD CLIFFS:* PRENTICE HALL. 1995. 932P.

SIEGWART, ROLAND AND NOURBAKHSH, ILLAH R. *INTRODUCTION TO AUTONOMOUS MOBILE ROBOTS.* A BRADFORD BOOK, THE MIT PRESS: CAMBRIDGE, LONDON. 317P. 2004.

SMITH, R. *CONSTRAINTS IN RIGID BODY DYNAMICS.* IN: GAME PROGRAMMING GEMS 4 BY KIRMSE, ANDREW (ED.). CHARLES RIVER MEDIA. (2004). ISBN: 978-1584502951. SITE: HTTP://WWW.GAMEPROGRAMMINGGEMS.COM/ AND HTTP://WWW.ODE.ORG/JOINTS.PDF

SMITH, R. (2006). *OPEN DYNAMICS ENGINE V0.5 USER GUIDE* - HTTP://WWW.ODE.ORG/ODE-LATEST-USERGUIDE.HTML - LAST VISITED: SEPT. 2009.

THRUN, S. ET AL. (2006). *STANLEY: THE ROBOT THAT WON THE DARPA GRAND CHALLENGE*,. JOURNAL OF FIELD ROBOTICS, VOL. 23, NO. 9, JUNE 2006, P.661-692. HTTP://ROBOTS.STANFORD.EDU/PAPERS.HTML (ACESSO EM 08/02/2009).

URMSON, CHRIS ET AL. (2008). *AUTONOMOUS DRIVING IN URBAN ENVIRONMENTS: BOSS AND THE URBAN CHALLENGE.* IN: JOURNAL OF FIELD ROBOTICS. VOL. 25 , ISSUE 8 (AUGUST 2008). SPECIAL ISSUE ON THE 2007 DARPA URBAN CHALLENGE, PART I. PAGES 425-466.

WOLF, DENIS F. ; OSÓRIO, FERNANDO S. ; SIMÕES, EDUARDO ; TRINDADE JR., ONOFRE . *ROBÓTICA INTELIGENTE: DA SIMULAÇÃO ÀS APLICAÇÕES NO MUNDO REAL.* IN: ANDRÉ PONCE DE LEON F. DE CARVALHO; TOMASZ KOWALTOWSKI. (ORG.). JAI: JORNADA DE ATUALIZAÇÃO EM INFORMÁTICA DA SBC. RIO DE JANEIRO: SBC - EDITORA DA PUC RIO, 2009, V. 1, P. 279-330. DISPONÍVEL EM: HTTP://OSORIO.WAIT4.ORG/PALESTRAS/JAI2009.HTML

## Referências Web (On-Line)

Ackerman – Ackerman Steering Principle
Web: http://www.rctek.com/technical/handling/ackerman_steering_principle.html [Acesso 08/10/2009]

AI_Car - Vehicle Simulation Software [Acesso 08/10/2009]
Web: http://sites.google.com/site/xrobot17/project-1/ai-car

Car_Racing - 3D Theory: Car Racing Game
Web: [Acesso 08/10/2009]
http://www.euclideanspace.com/threed/games/examples/cars/

Car Racing Games [Acesso 01/09/2009]
Web: http://en.wikipedia.org/wiki/Racing_game

Chipmunk – Chipmunk 2D [Acesso 08/10/2009]
Web: http://code.google.com/p/chipmunk-physics/

CIG 2007 Competitions [Acesso 01/09/2009]
Web: http://julian.togelius.com/cig2007competition/

CIG 2009 Competitions [Acesso 01/09/2009]
Web: http://www.ieee-cig.org/competitions/#scr

CIG_Manual 2009 Competition [Acesso 01/09/2009]
Web: http://sourceforge.net/projects/cig/files/Championship 2009 Manual/

Delta3D Engine: OSG + ODE + CAL3D
Web: http://www.delta3d.org/ [Acesso 01/09/2009]

Game_Engines – List of Game Engines [Acesso 01/09/2009]
Web: http://en.wikipedia.org/wiki/List_of_game_engines

GTA – Grand Theft Auto [Acesso 01/09/2009]
Web: http://www.rockstargames.com/games/

Havok - Havok Physics [Acesso 01/09/2009]
Web: http://www.havok.com/index.php?page=havok-physics

KLOV – Killer list of Video Games
Web: http://www.klov.com/ [Acesso 01/09/2009]

LRM – Laboratório de Robótica Móvel do ICMC-USP
Web: http://www.icmc.usp.br/~lrm/ [Acesso 01/09/2009]

Newton Game Dynamics [Acesso 01/09/2009]
Web: http://newtondynamics.com/

ODE – Open Dynamics Engine
Web: http://www.ode.org/ [Acesso 01/09/2009]

ODE_Manual (Wiki) [Acesso 01/09/2009]
Web: http://www.ode.org/ode-latest-userguide.html
http://opende.sourceforge.net/wiki/index.php/Manual_(All)

OpenSteer - Steering Behaviors for Autonomous Characters.
Web: http://opensteer.sourceforge.net/ [Acesso 01/09/2009]
e http://www.red3d.com/cwr/steer/

PID Controller – Wikipedia [Acesso 01/09/2009]
Web: http://en.wikipedia.org/wiki/PID_controller

Physics_Engine – Wikipedia [Acesso 01/09/2009]
Web: http://en.wikipedia.org/wiki/Physics_engine

PhysX – Nvidia / AGEIA PhysX [Acesso 01/09/2009]
Web: http://www.nvidia.com/object/physx_new.html

RoboCup – Official site [Acesso 01/09/2009]
Web: http://www.robocup.org/

Robot_Simul – List of Robot Simulator Softwares (OGAI)
Web: http://ogai.org/software/ [Acesso 01/09/2009]

SENA – Projeto SENA – EESC/ICMC USP
Web: http://www.eesc.usp.br/sena/ [Acesso 01/09/2009]

Simulated Car Racing Competition at CIG 2009
(Computational Intelligence in Game) [Acesso 01/09/2009]
Web: http://www.ieee-cig.org/competitions/#scr

Steering – Ackermann Steering Geometry
Web: [Acesso 01/09/2009]
http://en.wikipedia.org/wiki/Ackermann_steering_geometry

Tokamak – Physics Engine [Acesso 01/09/2009]
Web: http://www.tokamakphysics.com/

TORCS, The Open Racing Car Simulator.
Web: http://torcs.sourceforge.net/ [Acesso 01/09/2009]

Tutorial-Carros: Site deste Tutorial do SBGames 2009
Web: http://osorio.wait4.org/palestras/sbgames09.html
[Acesso 08/10/2009]