João Luiz Bernardes Jr.
Ricardo Nakamura

FACT:
Ferramenta de Análise de Colisões de Trânsito

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para aprovação na disciplina de graduação PMC 581.

9.0 (nove)

São Paulo
1998

João Luiz Bernardes Jr.
Ricardo Nakamura

FACT:
Ferramenta de Análise de Colisões de Trânsito

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para aprovação na disciplina de graduação PMC 581.

Engenharia Mecânica - Automação e Sistemas

Orientador:
Clóvis de Arruda Martins

São Paulo
1998

Aos nossos pais.

AGRADECIMENTOS

Ao nosso orientador, Prof. Dr. Clóvis de Arruda Martins, pelo indispensável auxílio, apoio e compreensão.

Ao engenheiro Brian G. McHenry, pelo tempo e atenção a nós dispensados, sem os quais o desenvolvimento deste trabalho teria sido muito mais árduo.

Aos técnicos em reconstituição de acidentes da Delegacia Seccional Oeste do Município de São Paulo, pela presteza em nos fornecer informações quanto à prática da reconstituição de acidentes no Brasil.

Resumo

Este trabalho relata o procedimento utilizado para estudar a viabilidade e desenvolver um software de reconstrução de acidentes automobilísticos. Considera-se os efeitos de deformação e frenagem no modelo do acidente, que é simulado através de algoritmos numéricos.

Abstract

This work reports the procedure used to study the feasability and develop an automotive accident reconstruction software. The effects of crushing and braking are considered in the model, which is simulated through numerical algorithms.

Índice

1. Introdução

Estudos de colisões entre veículos são muito utilizadas nos tribunais de justiça (ao se julgarem os acidentes) e por companhias de seguro. Tais estudos são realizados visando a determinação das causas do acidente e da imputabilidade de culpa a qualquer das partes nele envolvidas.

1.1. Histórico

No cenário mundial, o maior levantamento de modelos analíticos, dados empíricos e ferramentas computacionais visando o estudo de acidentes de trânsito se deu nos Estados Unidos, a partir da década de 50. Isto se deu devido a uma intensa política governamental neste sentido. Tanto é, que a maioria das referências bibliográficas utilizadas neste trabalho provêm deste país.

Em 1952 foi implantado nos Estados unidos um programa de estudos quanto à segurança em auto-estradas, cujo principal objetivo era, com base em análises das causas dos ferimentos sofridos pelas vítimas, levar a um projeto de veículos mais seguros. Em meados dos anos 60, 31 dos 50 estados já participavam deste programa e já haviam fornecido mais de 50.000 casos para estudo. O principal critério de classificação da severidade dos acidentes era a comparação das fotos dos veículos neles envolvidos.

Também neste período, houve o surgimento dos grandes mainframes que, apesar de pouco desenvolvidos (ocupavam grandes espaços, tinham alto custo de operação e manutenção e dificuldade de armazenamento de dados), forneciam uma capacidade de processamento de dados até então não disponível.

Em setembro de 1966, o presidente Lyndon Johnson assinou o "National Traffic and Motor Vehicle Safety Act" e o "National Highway Safety Act", pois acidentes de tráfego eram a maior causa de mortes entre os americanos com menos de 35 anos de idade. Este ato criou a NHTSA (National Highway Traffic Safety Agency - originalmente NTSA) e o FMVSS (Federal Motor Vehicle Safety Standards). A

NHTSA foi a mola propulsora das pesquisas na área de reconstrução de acidentes, tendo a autoridade para financiar e encomendar projetos neste sentido, além de promover o intercâmbio de informações da área.

Neste contexto, o engenheiro Raymond McHenry, trabalhando no "Cornell Aeronautical Laboratory" (atualmente chamado de Calspan) desenvolveu um programa para demonstrar a viabilidade do desenvolvimento de um modelo matemático de colisões entre veículos. O objetivo disto era uniformizar a interpretação de evidências em acidentes de tráfego. Este programa, chamado SMAC (Simulation Model of Automobile Collisions), consistia num programa de simulação no tempo de um acidente automobilístico. Esta versão inicial do programa era bastante limitada na sua capacidade de integração da trajetória devido aos limites de natureza computacional presentes na época. Além disto, o custo de sua utilização era bastante elevado.

Para amenizar estes problemas, foi criado, pelo mesmo engenheiro, o programa CRASH, para fornecer uma estimativa inicial para o SMAC. Posteriormente o CRASH desenvolveu-se separadamente do SMAC, e inclusive o superou em termos de utilização, devido à sua maior simplicidade. O CRASH utiliza um modelo que relaciona linearmente a força média durante a deformação dos automóveis com o perfil desta deformação. Este modelo foi bastante aceito devido à sua simplicidade e por incorrer num erro aceitável e é utilizado amplamente até hoje. Foi criada uma extensa base de dados (baseados em "crash tests") para permitir o uso deste modelo.

Após este período houve o surgimento de diversas variantes destes programas, mas acompanhadas de poucas mudanças no equacionamento fundamental do problema, ou em sua solução computacional.

Na década de 80 houve uma retomada de interesse nesta área, causando uma revisão e o refinamento dos algoritmos existentes.

Atualmente, com o aumento notável do poder computacional dos computadores pessoais, e redução dos custos, tem havido um interesse de portabilizar as soluções existentes para estas plataformas, assim como de aproveitar este poder de processamento

para a utilização de métodos que forneçam respostas mais precisas e de novas interfaces e recursos de animação que permitam um uso mais simples e elucidativo dos programas

1.2. Situação Atual no Brasil

No Brasil, a reconstrução de acidentes é realizada, na maior parte dos casos, sem o auxílio de ferramentas computacionais. Como não existiu aqui o mesmo incentivo governamental verificado a partir da década de 60 nos Estados Unidos (apesar do número de acidentes ser da mesma ordem de grandeza), não se trabalhou no desenvolvimento destas ferramentas, nem tampouco na verificação da validade legal das mesmas.

Em geral, o procedimento aqui efetuado consiste em registrar o cenário do acidente, incluindo a posição final dos carros e eventuais marcas de pneus. Registram-se também a forma e posição das deformações causadas nos veículos. Com base nestes dados, determinam-se as velocidades dos veículos no momento da colisão, através de relações empíricas. Por exemplo, com auxílio de um instrumento semelhante a uma "régua de cálculo", determina-se pelas dimensões das marcas de frenagem uma estimativa da velocidade do veículo. Não se efetuam, normalmente, cálculos de conservação de momento ou energia.

1.3. Definição do Problema

O objetivo deste trabalho é fornecer uma ferramenta para auxiliar o técnico nessa análise, na forma de um programa. O problema que este programa deve resolver é o seguinte: partindo-se de condições iniciais de posição e velocidade dos veículos, encontrar a posição de repouso desses veículos após a colisão. Além disso, seria desejável que o programa fornecesse outras informações que pudessem ser comparadas com dados colhidos no local do acidente (como por exemplo, deformações dos veículos, marcas de pneu etc).

Apesar do objetivo final deste trabalho ser a apresentação do protótipo de um produto completo, por se tratar de um trabalho acadêmico de engenharia mecânica, não

será feito um estudo da validade legal do uso desta ferramenta (como seria necessário, no caso de um produto comercial).

O problema pode ser subdividido no tratamento das seguintes questões:

- interação entre os pneus e o solo;
- atuação do motorista sobre a posição das rodas dianteiras, através do volante;
- frenagem (parcial, total, ABS);
- resistência ao rolamento das rodas (freio motor);
- mudança do comportamento dinâmico do veículo devido às deformações;
- perda de energia por deformação e atrito durante o impacto;
- força atuante no impacto (módulo, ponto de aplicação e direção, determinada pelo ângulo denominado "ângulo de ataque");
- colisão com barreiras;
- obtenção das equações de movimento dos veículos envolvidos após o impacto;
- integração dessas equações;
- exibição dos resultados.

No diagrama de blocos abaixo, pode-se observar a estrutura proposta para resolver estes problemas.

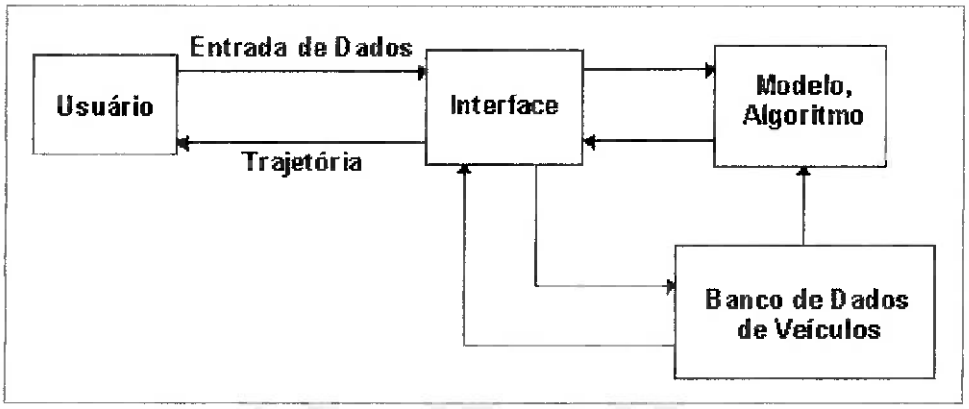


Figura 1 - Diagrama de blocos da solução proposta

2. Estudo de Viabilidade

2.1. Objetivos

Nesta parte do projeto, deseja-se fazer uma análise macroscópica do problema (conforme definido no item anterior), levantando diversas alternativas de solução e escolhendo a mais apropiada. O objetivo final desta parte do projeto é especificar completamente o produto a ser desenvolvido. Esse desenvolvimento será então realizado na fase posterior do projeto.

2.2. Estrutura do Estudo de Viabilidade

No próximo item, será dada uma definição detalhada do problema que se pretende resolver, de maneira que se possa prosseguir com a análise do mesmo de forma clara.

No item 3, apresenta-se um estudo feito sobre os possíveis meios de se solucionar o problema. Neste item, serão apresentados os fundamentos teóricos envolvidos nestas soluções, de forma superficial.

O item 4 consiste na escolha, dentre as soluções apresentadas no item 3, das que serão de fato utilizadas. Para fazer esta escolha, será utilizado o método da "Matriz de decisão", também explicado neste item.

O item 5 especifica a solução escolhida no item anterior num nível maior de detalhe que o apresentado até este ponto.

No item 6, por fim, apresenta-se a conclusão do trabalho.

No item 7 é mostrado o cronograma de atividades deste projeto, para fins de referência.

2.3. Estudo de Alternativas para a Solução

No desenvolvimento de um software existem vários parâmetros não diretamente relacionados com a solução do problema mas sim relacionados com o próprio

desenvolvimento. Por exemplo, a linguagem utilizada, a plataforma, o compilador, entre outros. Neste item, serão apresentadas estas alternativas e também aquelas diretamente relacionadas com a solução do problema (modelos, algoritmos, estruturas de dados etc).

2.3.1. Alternativas para Desenvolvimento do Software

2.3.1.1. Alternativas para Linguagem de Programação

Quanto à linguagem de programação a ser adotada, a princípio existe um grande número de alternativas. Entretanto, por questões de familiaridade e experiência anterior foram consideradas viáveis apenas quatro alternativas: Pascal, C, C++ e Fortran. Cada uma destas linguagens apresenta características distintas.

As linguagens C e Pascal, sendo linguagens estruturada para aplicações gerais, trazem como características a versatilidade e sua ampla utilização, fazendo com que seja possível encontrar com maior facilidade bibliotecas de funções e referências. O Pascal tem a vantagem de ser uma linguagem mais didática, que leva a programação de um código mais consistente. Justamente por isso, porém, é menos versátil que o C.

A linguagem Fortran apresenta como vantagem principal o fato de ser voltada para desenvolvimento de aplicações matemáticas ou físicas. Além disso, já existe uma grande variedade de rotinas e bibliotecas para essas aplicações. Desta forma, a implementação dos algoritmos ligados ao modelo e à solução numérica seria simplificada.

Finalmente, a linguagem C++ traz as características da linguagem C, e adicionalmente permite uma abordagem orientada a objetos.

2.3.1.2. Alternativas para Plataforma

Em relação à plataforma utilizada na implementação do programa, pode-se fazer duas observações:

Na implementação do código principal (algoritmos numéricos, simulação) não existem, a princípio, limitações; o código poderia ser compilado em qualquer plataforma na qual existisse um compilador para a linguagem escolhida. Se o algoritmo, após a

implementação, mostrar que exige muita memória ou mais velocidade para fornecer um desempenho satisfatório, será necessário escolher uma plataforma com mais recursos.

Existe, entretanto, outra parte do programa, constituída principalmente pela interface com o usuário, que em geral depende da plataforma escolhida. Tendo-se em vista esta limitação, as opções de plataforma consideradas foram aquelas às quais o acesso era maior. Essencialmente, micros padrão PC, com sistema operacional DOS ou Windows.

O sistema DOS permitiria, a princípio, a utilização do programa num maior número de computadores, inclusive de menor capacidade. Por outro lado, o desenvolvimento de interfaces amigáveis torna-se quase sempre um problema que toma muito tempo.

No sistema Windows, a implementação de uma interface gráfica torna-se mais simples, assim como o uso de recursos do computador como expansões de memória.

Além das plataformas analisadas, poderia-se utilizar outros sistemas, como "OS/2" e "System 7". Entretanto, no caso deste trabalho, a falta de familiaridade com estas plataformas (além de sua baixa disponibilidade) torna estas alternativas imediatamente inviáveis.

2.3.1.3. Alternativas para Compilador

Uma vez escolhida uma linguagem de programação e uma plataforma, existem vários compiladores disponíveis no mercado. Embora na escolha destes compiladores vários fatores sejam determinantes (como biblioteca de funções, recursos de debug, ambiente de edição...) o fator determinante da escolha é a familiaridade que já possa existir dos programadores com o compilador. Essa familiaridade também é determinante na escolha dos outros fatores.

2.3.2. Alternativas para Modelagem

2.3.2.1. Colisão

Uma primeira abordagem possível é não modelar a colisão. Neste caso, modela-se simplesmente um choque entre corpos rígidos. Como, no entanto, uma parcela significativa de energia é dissipada na colisão, esta modelagem apresenta erros não aceitáveis, exceto em colisões a baixas velocidades.

Outro modelo é o utilizado no programa CRASH (sendo referido, a partir deste ponto, como "modelo CRASH"). Este modelo propõe uma relação linear entre a força média atuante durante a colisão e a deformação plástica sofrida pelo veículo. Já foi comprovado que esta relação apresenta erros da ordem de 5% e existe uma extensa base de dados com as constantes requeridas por este modelo. Neste modelo, a força tem a seguinte expressão:

$$F = \int_0^L (A + B \cdot \delta) dx$$

onde F é o módulo da força atuante, A é a máxima força que não provoca deformação plástica, B é a inclinação da curva de força x deformação, δ é a deformação plástica medida no veículo e L é o comprimento da região deformada.

As constantes A e B podem ser obtidas a partir de resultados de "crash-tests" frontais. Nestes testes, a velocidade de colisão pode ser normalmente representada por uma relação linear com a deformação plástica medida, da seguinte forma:

$$V = B_0 + B_1.\delta$$

Desta maneira, pode-se obter as constantes A e B através de conservação de energia, chegando-se aos seguintes resultados:

$$A = M \cdot \frac{B_0 \cdot B_1}{L}, \qquad B = M \cdot \frac{B_1^2}{L}$$

onde M é a massa do veículo.

O modelo não fornece o ângulo desta força, que deve ser uma entrada para os cálculos. Estimativas deste ângulo, com base no coeficiente de atrito entre os veículos, se mostram pouco acuradas, de acordo com *McHenry* (ref. 2).

Normalmente utiliza-se este modelo tomando medidas da deformação em intervalos discretos.

Outra modelagem que também apresenta um erro aceitável é assumir que a força varia linearmente com a deformação total do veículo. Desta forma, pode-se construir um modelo onde a rigidez do veículo é modelada através de molas (por exemplo, ligando o centro de massa à periferia do veículo).

Neste caso, a força tem a seguinte expressão:

$$F = \int_0^L (K_V . \delta_T) dx$$

onde δ_T é a soma das deformações plástica e elástica e K_V é a constante da mola.

Finalmente, outra alternativa consistiria em modelar a deformação dos veículos por elementos finitos.

A análise por elementos finitos consiste num método de discretização de problemas contínuos. A aplicação deste método, normalmente se dá da seguinte maneira:

- O meio contínuo é dividido por linhas (ou superfícies) imaginárias em um número de elementos finitos.
- Assume-se que o número de pontos do contorno dos elementos que fazem a conexão entre eles (nós) é finito.
- Os estados dos nós (por exemplo, deformações, temperatura, posição, velocidade etc.) são as incógnitas do problema.

- É selecionado um conjunto de funções (funções de interpolação) de forma a definir unicamente o estado no interior de cada elemento, com base no estado dos nós deste elemento.
- Estas funções, juntamente com as condições iniciais do problema, descrevem o comportamento do sistema contínuo de forma discreta, permitindo que se obtenha os estados dos nós.

A determinação da forma de discretização (forma do elemento) não é um problema trivial, dependendo muito da experiência do autor do modelo. Além disto, a determinação das funções de interpolação depende da compatibilidade entre os elementos adjacentes. Normalmente é necessário impor um limite para esta compatibilidade (por exemplo, definir uma função contínua somente até a primeira derivada). Por isto, a análise somente nos nós não garante, necessariamente, uma resposta precisa localmente, no interior dos elementos.

Esta forma de análise apresenta normalmente (se os problemas acima forem resolvidos a contento) uma correlação elevada. Como pôde-se observar, porém, normalmente não só a modelagem, como também o "uso" do modelo obtido, é de alta complexidade. Além disto, devido à natureza deste método, observa-se que ele se presta melhor à uma análise mais minuciosa de problemas. Para análises mais macroscópicas, não é tão adequado.

2.3.2.2. Trajetória

Uma primeira aproximação para determinação da trajetória pode ser a seguinte: ignora-se o efeito do momento angular (ou do momento linear) e utiliza-se somente a equação de momento linear (ou angular) para obter uma solução fechada para a trajetória. Embora esta solução seja bastante simplificada e apresente erros elevados para alguns casos, é uma alternativa frequentemente considerada, pois trata-se do método utilizado pelos peritos para estimar trajetórias sem o auxílio do computador. Utiliza-se, normalmente, somente a conservação de momento linear com um coeficiente de atrito menor do que o nominal (entre 70 e 85%) para levar em conta a energia dispendida na rotação do veículo.

Utilizando a conservação de momento linear, após serem definidas as velocidades de separação dos veículos, para cada um deles pode-se obter a seguinte expressão para a distância percorrida:

$$S = \frac{V_s^2}{\mu \cdot g}$$

onde S é a distância percorrida, V_S é a velocidade de separação, μ é o coeficiente de atrito e g é a aceleração da gravidade.

Pode-se também obter uma relação semelhante ao se levar em conta somente a conservação de momento angular.

Ao se considerar somente o momento linear os maiores erros ocorrem em relação a trajetórias onde o ângulo total de que o veículo girou é elevado (maior do que 5 graus). O oposto ocorre ao se considerar somente o momento angular.

Uma outra modelagem que leva em conta a conservação tanto de momento linear quanto angular, mas ainda assim fornece uma solução fechada é a modelagem de Marquard.

Marquard observou que as variações de velocidade angular e linear se alternam no tempo, ou seja, quando uma varia, a outra mantém-se aproximadamente constante, como está ilustrado no gráfico (a) da figura 2. Baseado nisto, propôs a aproximação linear ilustrada no gráfico (b) da mesma figura e desvinculou estas variações de velocidade. Obteve assim, relações empíricas para balancear os efeitos dos momentos linear e angular. Por exemplo:

$$\dot{\theta} = -\frac{\mu \cdot g \cdot WB}{2 \cdot R^2}$$

onde $\dot{\theta}$ é a aceleração angular, WB é a distância entre as rodas e R é o raio de giração do veículo.

$$\ddot{S} = -0{,}85 \cdot \mu \cdot g$$

onde $\ddot{S}$ é a aceleração do veículo.

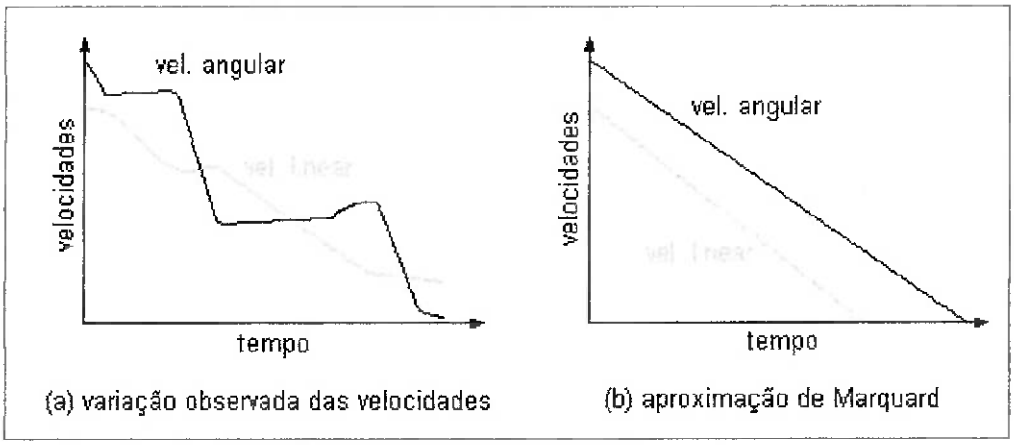


Figura 2 - modelagem de Marquard

Posteriormente, foram estudadas variações destas relações que se adequavam melhor a um certo conjunto de casos. No entanto, devido ao fato desta modelagem se basear em valores médios de vários casos de colisão, sempre existem casos que, ainda que atendendo às hipóteses do problema, se distanciam desta média, causando respostas com erro elevado.

Por fim, pode-se escrever as equações diferenciais que descrevem o movimento dos veículos, levando em conta as conservações de momento linear e angular e de energia. Integrando estas equações (de forma aberta) ao longo do tempo, obtém-se uma simulação da trajetória. Embora esta seja a alternativa mais direta e que apresenta o menor erro, a integração destas equações exige recursos computacionais que não eram amplamente disponíveis até pouco tempo atrás. Isto explica o desenvolvimento das relações explicadas anteriormente.

2.3.3. Alternativas para Estruturas de Dados

Uma lista ligada é uma coleção de elementos alocados dinamicamente e em lugares aleatórios da memória (ao contrário de um vetor, por exemplo, onde os elementos são alocados em uma ordem definida), e onde cada elemento guarda um ponteiro para pelo menos um outro elemento. Desse modo, guardando um elemento adequado da lista, pode-se alcançar qualquer outro elemento, independente de onde ele esteja guardado na memória. As listas são normalmente divididas em *simplesmente ligadas* ou *duplamente*

ligadas. A diferença é a seguinte: elementos de listas simplesmente ligadas só guardam um ponteiro para o próximo elemento. Desse modo só se pode percorrer a lista num sentido. Elementos de listas duplamente ligadas guardam dois ponteiros - para o próximo elemento e para o anterior - de modo que se pode percorrer a lista em dois sentidos. Listas ligadas não precisam necessariamente ser lineares. Uma matriz pode ser representada por uma lista onde cada elemento aponta para a direita e para baixo (simplesmente ligada) ou por uma onde cada elemento aponta para as quatro direções (duplamente ligada). A Figura 3 mostra um diagrama que busca facilitar o entendimento das diferenças entre vetores, listas ligadas e listas duplamente ligadas:

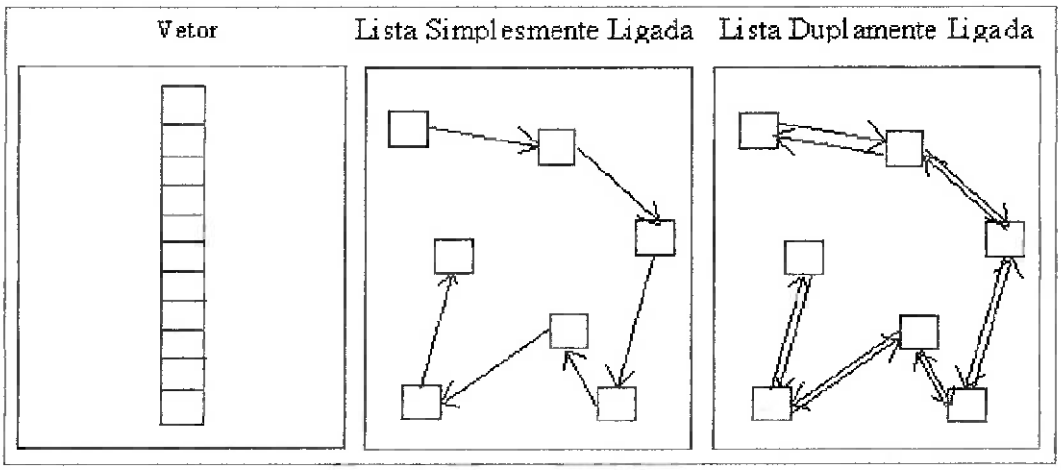


Figura 3 - Vetor e Listas Ligadas

Além do que já foi mencionado, as listas ligadas tem como grande vantagem a facilidade de inserção de elementos. Para inserir um elemento no meio de um vetor, por exemplo, é necessário mover todos os elementos abaixo do que será inserido uma posição para baixo, enquanto que para inserir numa lista, basta mudar um ponteiro do elemento anterior (ou também do posterior, nas duplamente ligadas). Sua grande desvantagem é a lentidão de acesso a um elemento qualquer. Para acessar um elemento no meio da lista é necessário percorrer todos os que a precedem, enquanto num vetor o acesso é imediato.

Vetores dinâmicos são essencialmente vetores, com as mesmas propriedades, A diferença é que, neste caso, seu tamanho pode ser definido no momento da execução do programa. Isto se dá porque a memória para estes vetores é alocada dinamicamente, por exemplo com a função "malloc" da linguagem C.

O CObArray é uma classe disponível na biblioteca "Microsoft Foundation Classes". Esta classe tem a mesma funcionalidade de um vetor e suas propriedades quanto a acesso das variáveis. No entanto, vai além da alocação dinâmica, pois permite o aumento do número de elementos alocados em qualquer momento da execução do programa. Sua grande desvantagem é que ocupa muito mais espaço do que o ocupado meramente pelos seus elementos devido às informações referentes à classe.

2.4. Escolha da Solução

Para se realizar a escolha entre as alternativas expostas no item 3, foi utilizada a metodologia de matrizes de decisão. Esta metodologia consiste em, dado um conjunto de critérios, atribuir a cada um deles um peso que indica quão importante é esse critério na decisão, em relação aos outros. Feito isso, para cada uma das alternativas, atribui-se uma nota referente a cada critério. Calcula-se então a nota total de cada alternativa, multiplicando-se a nota dada em um certo critério pelo seu peso, e somando as notas de todos os critérios. A alternativa que obtiver a maior nota é escolhida. No caso deste trabalho, decidiu-se atribuir notas entre 0 e 1. As notas totais poderiam ser normalizadas, mas neste caso não foram.

2.4.1. Escolhas para Desenvolvimento do Software

2.4.1.1. Linguagem de Programação

Foram empregados os critérios que já foram comentados no item 3.1.1 obtendo-se a seguinte matriz de decisão:

Tabela 1 - Matriz de decisão para a linguagem de programação

	Linguagem	C	C++	Pascal	Fortran
Critério	Peso				
Estruturada	3	1	1	1	0
Orientação a Objeto	2	0	1	0	0
Familiaridade	4	0,8	0,9	0,7	0,4
Recursos Matemáticos	2	0,6	0,6	0,4	1
Total	-	7,4	9,8	6,6	3,6

Com base nesta tabela, escolheu-se C++ como a alternativa para linguagem de programação.

2.4.1.2. Plataforma

No processo de decisão analisou-se somente os sistemas operacionais. A configuração de hardware a ser escolhida será a mínima adequada a este sistema a não ser que posteriormente comprove-se que o programa necessita de mais recursos.

Os critérios utilizados foram os seguintes:

- disponibilidade: é a facilidade, tanto para o programador, quanto para o universo de usuários, de ter acesso a uma plataforma utilizando o sistema. Nos sistemas DOS e Windows 16 bits, atribuiu-se uma disponibilidade maior, devido ao fato de que programas desenvolvidos para eles, normalmente podem ser rodados nos outros sistemas, ainda que não de maneira ótima;
- familiaridade: a familiaridade dos programadores quanto ao desenvolvimento de software para o sistema;
- interface: a facilidade do desenvolvimento de uma interface "user-friendly";
- segurança: a estabilidade do sistema contra falhas;

- recursos: o gerenciamento e acesso aos recursos de hardware;

Tabela 2 - Matriz de decisão de sistemas operacionais

	Sistema	DOS	Unix	Windows 16 bits	Windows 32 bits
Critério	Peso				
Disponibilidade	4	1	0,5	1	0,8
Familiaridade	4	0,8	0,6	1	0,8
Interface	2	0,4	0,7	1	1
Segurança	1	0,4	1	0,5	0,6
Recursos	2	0,4	1	0,7	1
Total	-	9,2	8,8	11,9	11

Da tabela, selecionou-se o sistema Windows de 16 bits (Windows 3.X, Windows for Workgroups). Observou-se ainda que há pouca diferença entre as notas dos sistemas Windows de 16 e 32 bits. Por causa disso, no item seguinte foram analisados também compiladores para 32 bits.

2.4.1.3. Compilador

Os critérios utilizados na escolha foram aqueles discutidos no item 3.1.3. Para todos estes compiladores, assumiu-se que a capacidade de "debug" e as bibliotecas de funções são equivalentes. Portanto, estes critérios não foram utilizados na análise.

Além disso, foi adicionado o critério 16/32 BITS. Este critério foi inserido pois, como foi visto no item anterior, considerou-se interessante analisar também compiladores para 32 bits. Neste caso, o compilador de 16 bits recebeu um peso maior que o destinado somente a 32 bits já que foi a plataforma escolhida no item anterior. Compiladores que oferecessem as duas opções receberam a nota máxima.

Tabela 3 - Matriz de decisão para compilador

	Compilador	Visual C++ 1.52	Visual C++ 5	Borland C++ 4	Borland Builder
Critério	Peso				
16/32 BITS	1	0,5	0	1	0
Disponibilidade	1	1	0,8	0,7	0,7
Familiaridade	1	1	0,7	0,6	0,6
Total	-	2,5	1,5	2,3	1,3

Com base nesta tabela, foi escolhido o compilador Visual C++ 1.52.

2.4.2. Escolha do Modelo

Para a avaliação do modelo, utilizaram-se os seguintes critérios:

- Dados: quantidade de dados necessários para proceder com a análise. Quanto maior o número de dados necessários, maior a dificuldade para a determinação dos mesmos pelo usuário do programa. Isso limita a aplicação do programa;

- Correlação: uma estimativa de quão bem o modelo reflete a realidade. Atribuiu-se a este critério um peso maior do que para abrangência visto que ele reflete os erros que ocorrem mesmo quando são respeitadas as hipóteses do modelo;

- Abrangência: uma estimativa da porcentagem de casos que atendem às hipóteses simplificadoras adotadas na modelagem;

- Simplicidade: simplicidade de obtenção e implementação computacional do modelo.

Para atribuir as notas aos modelos, baseou-se na comparação entre os resultados obtidos em programas que utilizam cada um deles, ou em artigos tratando sobre estes modelos, com testes reais.

2.4.2.1. Colisão

Tabela 4 - Matriz de decisão para modelos de colisão

	Modelo	CRASH	Molas	E.F.
Critério	Peso			
Dados	2	0,9	0,8	0,4
Correlação	2	0,7	0,8	1
Abrangência	1	0,7	0,8	1
Simplicidade	1	1	0,8	0,4
Total	-	4,9	4,8	4,2

Da tabela acima, escolheu-se o modelo CRASH. Entretanto, o modelo de molas obteve uma nota bastante próxima de forma que ainda existe a possibilidade de sua utilização se perceber-se que o modelo CRASH não apresenta uma resposta satisfatória.

2.4.2.2. Trajetória

Tabela 5 - Matriz de decisão para modelos de trajetória

	Modelo	M. Linear	M. Angular	Marquard	Integração
Critério	Peso				
Dados	2	1	1	1	0,8
Correlação	2	0,3	0,3	0,5	0,9
Abrangência	1	0,3	0,3	0,5	1
Simplicidade	1	1	1	0,8	0,7
Total	-	3,9	3,9	4,3	5,1

Escolheu-se, com base na tabela acima, o método de integração da trajetória.

2.4.3. Escolha da Estrutura de Dados

Os critérios adotados foram:

- Manipulação: facilidade de implementação e eficiência na inserção e remoção de elementos. Neste caso, este critério foi considerado o mais importante por se tratar de um banco de dados ou de uma lista de pontos, obtidos durante a simulação. Desse modo, será constantemente necessário inserir e remover elementos destas estruturas;

- Simplicidade: facilidade de implementação e uso da estrutura;

- Memória: uso eficiente da memória. No caso de vetores estáticos esta eficiência foi considerada baixa devido ao fato que, normalmente, ocupa-se uma quantidade de memória bem maior que a necessária, uma vez que o tamanho do vetor não pode ser aumentado durante o programa;

- Acesso: velocidade de acesso a um dado elemento.

Tabela 6 - Matriz de decisão para estruturas de dados

Critério / Estrutura / Peso	Lista	Vetor	Vetor Dinâmico	CObArray
Manipulação 3	1	0,5	0,6	0,7
Simplicidade 2	0,8	1	0,9	0,9
Memória 1	0,8	0,7	1	0,7
Acesso 1	0,6	1	1	0,8
Total -	6	5,2	5,6	5,4

A solução escolhida, considerando-se a tabela acima, foi a de lista ligada.

2.5. Especificação da Solução Escolhida

2.5.1. Modelo

O programa simulará somente colisões planas entre dois veículos. Terá como entradas as condições iniciais destes veículos, além das deformações por eles sofridas. A saída será uma simulação da trajetória de ambos os veículos após o impacto até sua posição de repouso.

Foi decidido que o momento inicial da simulação será o choque entre os veículos. Desta forma, as velocidades e posições iniciais (que serão entradas do programa) serão aquelas imediatamente antes do choque. Embora isto complique a entrada de dados, pois normalmente é difícil obter estimativas precisas destes parâmetros, esta alternativa ainda é mais simples do que simular desde um momento anterior à colisão, porque isto envolve tratar entradas do motorista (direção, frenagem), que são ainda mais difíceis de se obter com precisão.

Para modelar a força atuante durante a colisão, e a energia por ela dissipada, será usado o modelo CRASH, explicado no item 3.2.1. Para simulação da trajetória, será usada a integração das equações de movimento dos veículos, explicada no item 3.2.2. Estas equações de movimento não estão aqui descritas em detalhe porque ainda estão sendo testadas, utilizando o Simulink para integrá-las.

As hipóteses que estão sendo usadas até o momento, são:

- Os veículos atingem uma velocidade comum durante um momento da colisão.
- Os veículos colidem na altura do centro de gravidade.
- Só existem momentos em torno do eixo vertical. Esta hipótese e a anterior permitem o tratamento da colisão e posterior movimento dos veículos como problemas planos.
- Após a deformação, os veículos são considerados corpos rígidos.

- Após a colisão, os pneus se encontram travados (frenagem total). Para carros sem freio ABS, esta hipótese é bastante razoável, visto que na grande maioria dos casos, o motorista aciona o freio brusca e totalmente logo após a colisão.

- O módulo da força atuante durante a colisão varia linearmente com a deformação plástica sofrida pelos veículos. Esta é a hipótese utilizada na modelagem do CRASH.

- Não ocorrem mudanças do comportamento dinâmico do veículo devido às deformações, ou seja, não são considerados casos em que, devido à colisão, quebra-se um eixo, por exemplo, ou rasga-se um pneu.

Posteriormente, pretende-se acrescentar ao modelo a colisão com barreiras infinitamente rígidas, a frenagem ABS e o posicionamento angular das rodas do veículo (fixo durante a trajetória pós-impacto).

2.5.2. Estruturas de Dados

Para o banco de dados dos veículos, será utilizada uma lista ligada de estruturas contendo os parâmetros de rigidez, inércia e geometria de cada veículo. Uma ilustração da organização desta estrutura pode ser observada na figura 4.

Os parâmetros A,B e G de rigidez são vetores com componentes F, R e S correspondentes às regiões frontal, posterior e lateral do veículo.

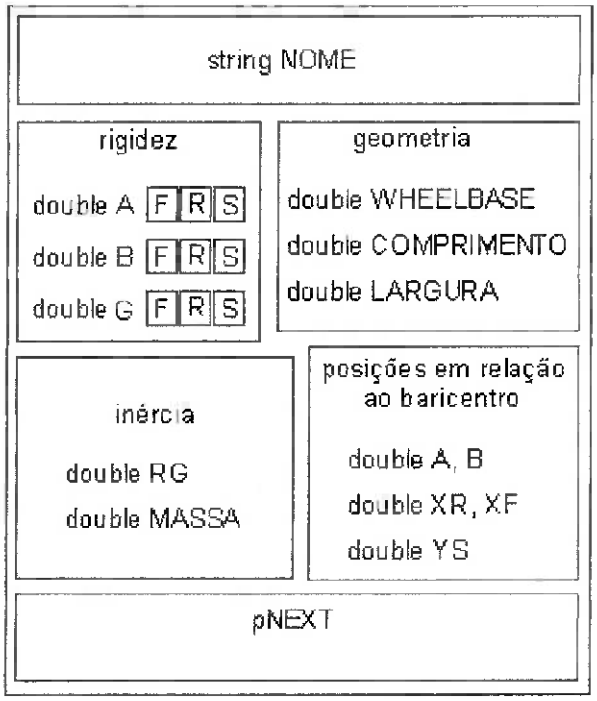


Figura 4 - Estrutura de dados para as informações dos veículos

2.5.3. Interface

Como já foi dito no item 4, foi escolhida a plataforma Windows para o desenvolvimento do programa. Além disso, existem muitos dados pertinentes ao problema. Para organizar melhor estes dados, resolveu-se agrupá-los na forma de "folders" como mostrado na figura 5, abaixo. Estes grupos são:

- veículo 1: entrada dos parâmetros (geométricos, de inércia e de rigidez) de um dos veículos envolvidos no acidente;
- veículo 2: o mesmo, para o outro veículo;
- deformação 1: descrição do perfil de deformação do veículo 1;
- deformação 2: o mesmo para o veículo 2;
- cenário: condições iniciais dos veículos envolvidos, posição e parâmetros das barreiras existentes;
- trajetória: saídas do programa.

Na figura 5 também pode ser visto um layout da entrada de dados dos veículos. Nesta entrada, além de escolher um dos veículos do banco de dados, o usuário também

pode modificar parâmetros referentes a este veículo, visto que os valores do banco de dados serão valores médios (por exemplo, o peso, que se refere a uma carga de somente dois passageiros de 70kg). Por fim, o usuário descreve as condições iniciais deste veículo.

A interação com o banco de dados também é feita por este grupo de entrada de dados, através dos botões para criar um novo veículo, adicionar e excluir veículos.

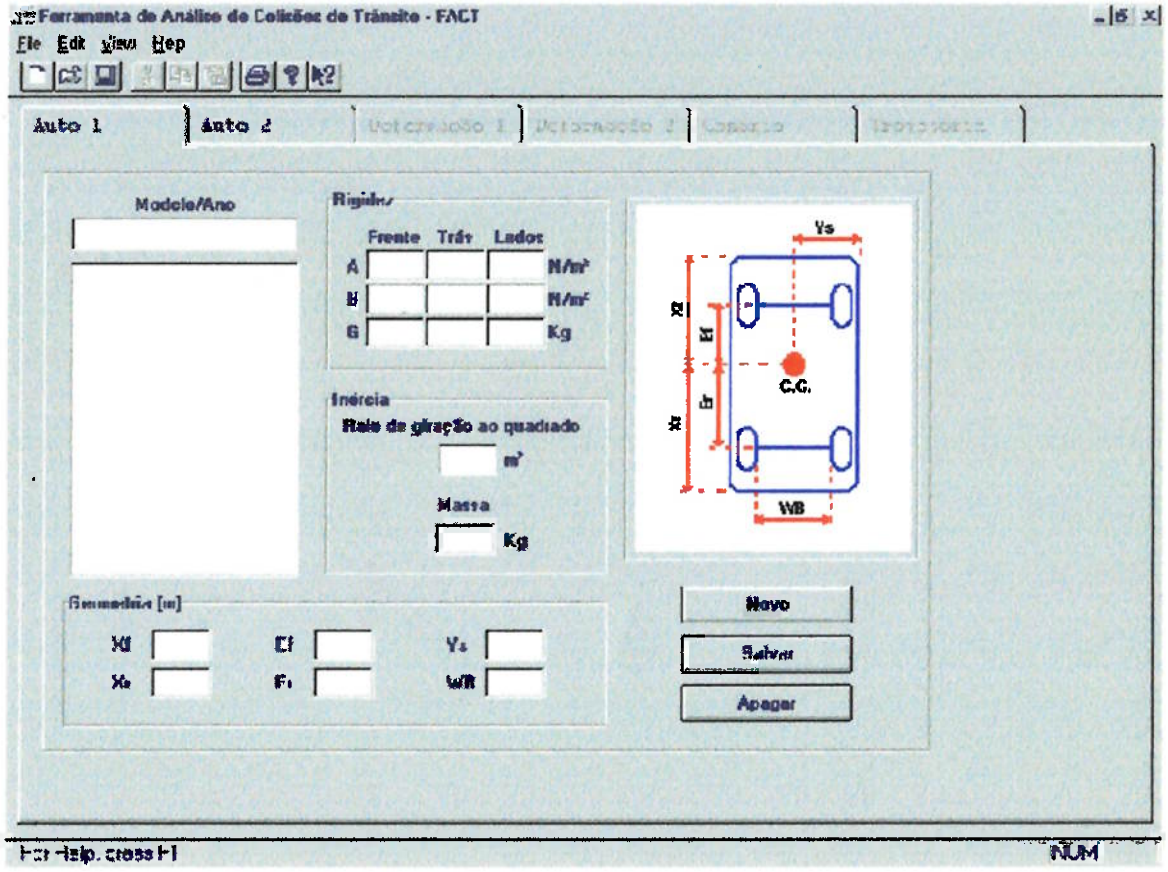


Figura 5 - Concepção geral da interface e layout dos dados do veículo

Na figura 6, abaixo, pode-se observar o layout proposto para a descrição do perfil de deformação de cada veículo. A deformação será descrita por meio de um vetor com profundidades de deformação em função da distância a um ponto. Este ponto, denominado origem, será entrado pelo usuário em relação ao baricentro. Além disso será apresentada uma figura ilustrando o perfil entrado pelo usuário.

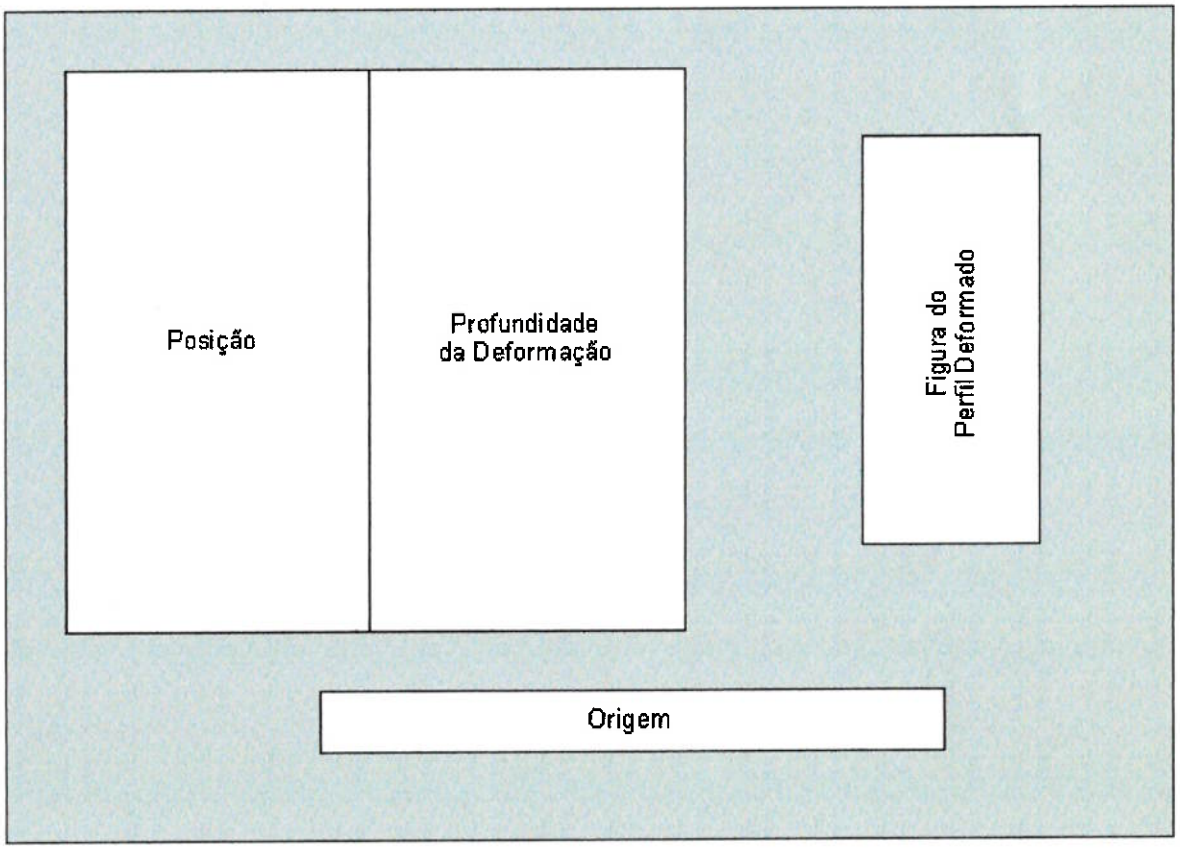


Figura 6 - Interface de entrada do perfil de deformação

Até este momento não se decidiu como serão modeladas as barreiras, de forma que não é possível especificar a parte da interface que trata da composição do cenário.

2.6. Conclusão

No primeiro semestre do projeto de formatura, a atividade principal foi a de pesquisa em busca de material para a definição clara do problema e alternativas para a forma de solução. Esta busca tomou bem mais tempo do que o esperado, devido à dificuldade de encontrar material bibliográfico sobre o assunto no Brasil. Por isso, o desenvolvimento do modelo ficou atrasado, mas em compensação, alguns dos dados

que precisariam ser levantados na segunda parte do projeto já foram obtidos e parte da codificação da interface e das estruturas de dados já foi feita.

3. Projeto Básico

3.1. Objetivos

Nesta fase do projeto, deseja-se aprofundar a análise da alternativa escolhida na fase anterior, de forma a implementar o projeto, conforme especificado no estudo de viabilidade.

3.2. Formulação do Modelo

Desde o início deste trabalho decidiu-se por não modelar o comportamento do veículo antes da colisão, pois os parâmetros de direção e aceleração, dependentes do motorista, são de natureza muito subjetiva, podendo prejudicar a análise posterior. Além disso, constatou-se que a polícia técnica tem, hoje em dia, boas condições para determinação das condições dos veículos no momento da colisão. Pode-se também utilizar o software para teste de versões, onde o que interessa é o intervalo entre a colisão e a parada dos veículos. Por isso, esta rformulação começa a partir da colisão, e utiliza as hipóteses definidas no item 2.5.1.

3.2.1. Força de Colisão

O modelo utilizado para a força de colisão, conforme já foi explicado no Estudo de Viabilidade, propõe uma relação linear entre a força média atuante durante a colisão e a deformação plástica sofrida pelo veículo. Essa expressão para a força é determinada igualando a energia dissipada numa colisão qualquer à dissipada numa colisão controlada (crash-test). Baseia-se, portanto, na conservação de energia. Já foi comprovado que esta relação apresenta erros da ordem de 5% e existe uma extensa base de dados com as constantes requeridas por este modelo. Neste modelo, a força tem a seguinte expressão:

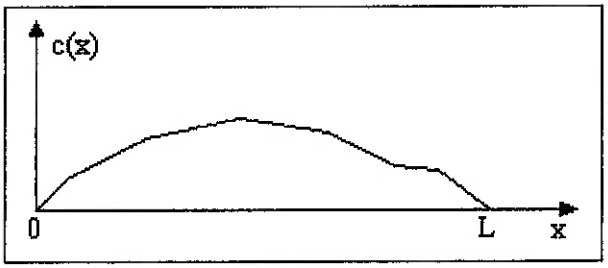


Figura 8 - perfil de deformação

$$F = \int_0^L (A + B \cdot c)dx$$

onde F é o módulo da força atuante, A é a máxima força que não provoca deformação plástica, B é a inclinação da curva de força x deformação, δ é a deformação plástica medida no veículo e L é o comprimento da região deformada.

As constantes A e B podem ser obtidas a partir de resultados de "crash-tests" frontais. Nestes testes, a velocidade de colisão pode ser normalmente representada por uma relação linear com a deformação plástica medida, da seguinte forma:

$$V = B_0 + B_1.c$$

Desta maneira, pode-se obter as constantes A e B através de conservação de energia, chegando-se aos seguintes resultados:

$$A = M \cdot \frac{B_0 \cdot B_1}{L}, \qquad B = M \cdot \frac{B_1^2}{L}$$

onde M é a massa do veículo.

O modelo não fornece o ângulo α desta força, que deve ser uma entrada para os cálculos. Estimativas deste ângulo, com base no coeficiente de atrito entre os veículos, se mostram pouco acuradas, de acordo com *McHenry* (ref. 2).

Normalmente utiliza-se este modelo tomando medidas da deformação em intervalos discretos.

Como a saída do modelo é a força média que atua durante a colisão, substitui-se a força impulsiva que atua por uma força constante, como na figura abaixo:

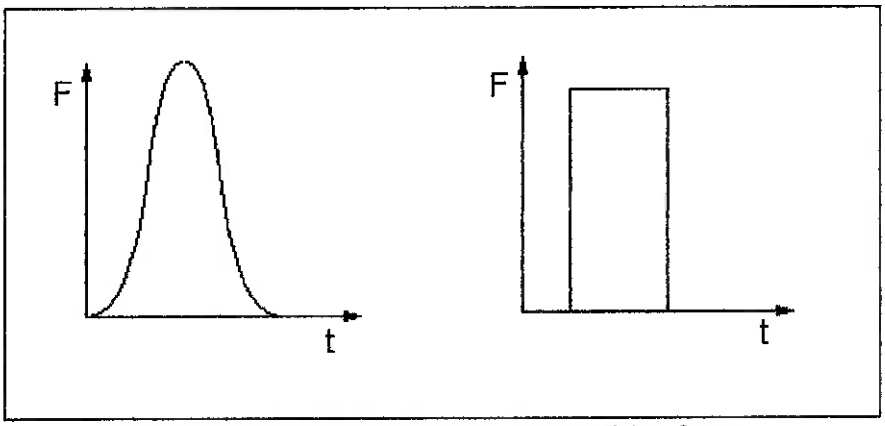


Figura 7 - Força de impulso (esquerda) e força adotada no modelo (direita)

Deve-se notar que, embora ambos os veículos tenham perfis de deformação diferentes, a força resultante da formulação deve ser a mesma para ambos, por ação e reação. Além disso, como o que se obtém é a força, e não o impulso atuante na colisão, é necessário determinar por quanto tempo ela age. Isso será discutido com mais detalhe no item 3.4.2, relativo aos algorítmos usados. Por fim, o ponto de aplicação da força (ponto de colisão, (xc, yc)) é definido com o baricentro da figura formada pelo perfil de amassamento.

Uma importante limitação deste modelo é a impossibilidade de distinguir entre amassamentos causados por colisões diferentes, o que não permitiu que fosse modelada a colisão com obstáculos após a colisão com outro veículo. Isso, além do fato de não se modelar o comportamento dos veículos antes da colisão, também impossibilitou a modelagem de colisão entre mais de dois veículos.

3.2.2. Esforços na Frenagem

Pode-se descrever esses esforços em função de cada pneu (responsável por parte da frenagem), ou em função dos esforços resultantes dos quatro pneus. A primeira análise é útil para o caso em que se considera o efeito do solo e da suspensão sobre os veículos. Nesse caso a normal em cada roda é diferente, tornando o cálculo da resultante mais complexo. Embora nesse programa esses efeitos não sejam

considerados, partiu-se desta formulação para ganhar um maior entendimento do mecanismo de frenagem, assim como para atingir a formulação simplificada. Ambas as formulações são indicadas a seguir. Nessa formulação utiliza-se a hipótese de que o peso do veículo sempre esta distribuído sobre os pneus da mesma forma, de acordo com a posição do baricentro. Ou seja, os pneus da direita e da esquerda sempre suportam a mesma parcela do peso e os dianteiros mais que os traseiros, pois estão mais próximos do baricentro. Por fim, a análise detalhada é feita na base do carro (ver Figura 9), para facilitar. A análise simplificada vale para ambas as bases (do carro e global). A base global é uma base fixa, inercial, que se relaciona com a base de cada carro pela sua posição angular θ e pela posição de seu baricentro. As bases dos carros estão transladadas de (x_G, y_g) e rotacionadas de θ em relação a base global.

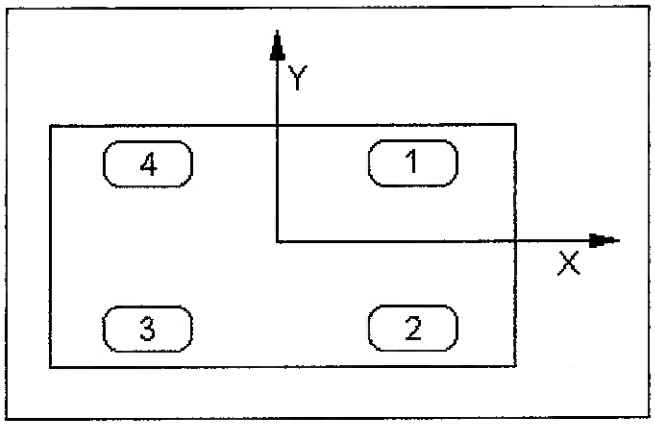


Figura 9 - Pneus e Sistema de Coordenadas Fixo no Veículo

3.2.2.1. Formulação Detalhada

Seja Pp a razão entre o peso suportado por um pneu qualquer e a distância, ao longo de x (na base do carro), entre o eixo oposto ao deste pneu e o baricentro do veículo.

$$Pp = \frac{Mg\mu}{2.(e_F + e_R)}$$

Onde:

M é a massa do veículo;

g é a aceleração da gravidade

μ é o coeficiente de atrito pneus-solo

e_F é a distância do eixo frontal ao baricentro

e_R é a distância do eixo traseiro ao baricentro

y_t é a distância das rodas ao baricentro,
ao longo de y (base do carro)

Para todos os pneus j, vale:

Vipj é a Velocidade no pneu, na direção i;
Vpj é a Velocidade total no pneu, se for
nula, não há força de atrito no pneu;
Fxj é a Força no pneu na direção x;
Fyj é a Força no pneu na direção y;
Tj é o Momento das forças no pneu,
em relação ao Baricentro.

Para calcular as velocidades dos pneus, utilizou-se a *equação de Poisson*, admitindo que os veículos se comportam como corpos rígidos, a menos da área amassada.

$$\vec{V}_P = \vec{V}_O + \vec{\omega} \wedge (P - O)$$

Pneu 1:

$$Vxp1 = Vx - w.y_T$$

$$Vyp1 = Vy + w.e_F$$

$$\vec{V}p1 = \vec{V}xp1 + \vec{V}yp1$$

$$Fx1 = -Pp.e_R.Vxp1/|Vp1|$$

$$Fy1 = -Pp.e_R.Vyp1/|Vp1|$$

$$T1 = -Fx1.\ y_T + Fy1.e_F$$

Pneu 2:

$$Vxp2 = Vx + w.\ y_T$$

$$Vyp2 = Vy + w.e_F$$

$$\vec{V}p2 = \vec{V}xp2 + \vec{V}yp2$$

$$Fx2 = -Pp.e_R.Vxp2/|Vp2|$$

$$Fy2 = -Pp.e_R.Vyp2/|Vp2|$$

$$T2 = +Fx2.\ y_T + Fy2.e_F$$

Pneu 3:

$$Vxp3 = Vx + w.\ y_T$$

$$Vyp1 = Vy - w.e_R$$

$$\vec{V}p3 = \vec{V}xp3 + \vec{V}yp3$$

$$Fx3 = -Pp.e_F.Vxp3/|Vp3|$$

$$Fy3 = -Pp.e_F.Vyp3/|Vp3|$$

$$T3 = -Fx3.\ y_T + Fy3.e_R$$

Pneu 4:

$$Vxp4 = Vx - w.\ y_T$$

$$Vyp4 = Vy - w.e_F$$

$$\vec{V}p4 = \vec{V}xp4 + \vec{V}yp4$$

$$Fx4 = -Pp.e_F.Vxp4/|Vp4|$$

$$Fy4 = -Pp.e_F.Vyp4/|Vp4|$$

$$T4 = -Fx4.\ y_T + Fy4.e_R$$

3.2.2.2. Formulação Simplificada

Calculando as resultantes (Fx, Fy e T) dos esforços nos quatro pneus obtém-se:

$$Fx = -Mg\mu\frac{Vx}{|V|} \qquad p/|V|>0$$

$$Fy = -Mg\mu\frac{Vy}{|V|} \qquad p/|V|>0$$

$$T = -Mg\mu.(e_F + e_r)\frac{\omega}{|\omega|} \qquad p/|\omega|>0$$

Caso a velocidade linear ou angular seja nula, o esforço correspondente também não existe, pois não há movimento ao qual o atrito se opõe..

3.2.3. Equações de Movimento dos Veículos

Como a interação entre os veículos está sendo modelada simplesmente como uma força constante (que se anula ao fim da colisão) sobre cada um, as equações do movimento de cada um estão desacopladas das do outro. Sendo assim, serão escritas somente as equações para um deles.

Teorema do movimento do baricentro:

$$\sum \vec{F} = m \cdot \dot{\vec{v}}_G$$

$$\ddot{x}_g = \frac{Fx + F\cos(\alpha)}{M}$$

$$\ddot{y}_g = \frac{Fy + F\mathrm{sen}(\alpha)}{M}$$

Teorema do momento angular (para problema plano, em relação ao baricentro):

$$\sum \vec{M}_G = J_G \dot{\vec{\omega}}$$

$$\ddot{\theta} = T - F\cos(\alpha).yc + Fsen(\alpha).xc$$

Esta formulação admite que, para cada carro, o ângulo α está definido de forma que a força sempre comprima o carro no ponto de colisão, nunca tracione.

3.3. Estruturas de Dados

Para as estruturas de dados, seguiu-se o modelo de orientação a objetos como no restante do programa. Desta forma, foram desenvolvidos objetos que armazenam conjuntos de dados relacionados e realizam o controle de acesso aos mesmos. As listagens da declaração e implementação de cada uma destas classes podem ser vistas no apêndice contendo as fontes do programa.

Na criação destes objetos, foram utilizadas como bases algumas classes presentes na MFC (Microsoft Foundation Classes) do compilador Visual C++ 1.52. Tais classes já apresentam implementações de estruturas como vetores de alocação dinâmica e listas ligadas, podendo ser utilizadas para reduzir o tempo de desenvolvimento.

As estruturas de dados mais importantes são as seguintes:

3.3.1. Dados dos Carros / Banco de Dados

Para armazenar os dados relativos a geometria, inércia e rigidez dos diferentes veículos, foi criada uma classe chamada CarData. Objetos desta classe podem ser adicionados então a um objeto CarDatabase, que implementa uma lista simplesmente ligada, sendo derivada da classe CObject da MFC (Microsoft Foundation Classes) do compilador Visual C++ 1.52.

A gravação dos dados dos veículos em arquivos é efetuada através do mecanismo de "serialização" implementada nos objetos CObject da MFC. Este mecanismo consiste no desenvolvimento de uma interface de funções através das quais

o objeto disponibiliza os dados que devem possuir armazenamento persistente a um objeto CArchive (tipicamente, um arquivo criado por outro objeto).

3.3.2. Dados de Deformação

O perfil de deformação foi implementado como um vetor de elementos alocados dinamicamente, utilizando-se a classe CObArray da MFC. Cada elemento armazena sua posição no sistema de coordenadas do veículo e a profundidade da deformação naquela posição.

O mecanismo de serialização, também presente nas classes CObArray, é utilizado para armazenar os dados dos perfis de deformação em arquivo.

3.3.3. Dados da Trajetória

O armazenamento dos dados de cada instante no tempo da solução das equações diferenciais é feito por objetos da classe TrajPoint. Esta classe contém as coordenadas X e Y, velocidades nestas direções, ângulo e velocidade angular de cada veículo. Estes objetos são por sua vez adicionados a um objeto da classe Trajetoria, que é baseado na classe CObArray. Desta forma, também se utiliza nesta classe a serialização para gravar os dados da trajetória (solução do problema) em arquivo.

3.4. Algoritmos para Solução do Problema

O principal algoritmo numérico utilizado na simulação é o algoritmo de integração do sistema de equações diferenciais. Junto dele, existe o algoritmo responsável pela verificação do término da atuação da força de colisão. Independente destes algoritmos, mas de grande importância foi o algoritmo de integração da força que atua durante a colisão, obtida a partir do perfil de deformação dos veículos.

Esta seção termina com um comentário a respeito dos problemas de precisão numérica que foram encontrados no desenvolvimento e teste dos algoritmos, e como eles foram solucionados.

Outros algoritmos desenvolvidos, como de escolha de escala para os gráficos dentre outros usados na interface não serão descritos, mas podem ser estudados no Apêndice III.

Serão apresentados os diagramas de Nassi-Schneidermann da integração das equações diferenciais, que compõe o cerne do programa. Os outors algoritmos serão somente descritos e sua implementação também pode ser encontrada nas listagens do Apêndice III.

3.4.1 Integração do perfil de deformação

Para calcular a força que age durante a colisão é preciso integrar a função A + By ao longo do perfil de deformação do veículo, onde A e B são parâmetros de rigidez e y é a profundidade de deformação no ponto. Para executar esta função, foi implementado um algoritmo de "regra do trapézio", utilizando diferenças para trás.

Tal escolha foi motivada pela simplicidade do método, assim como o fato da função integrada ser linear, de modo que a precisão obtida é suficiente mesmo com um número baixo de pontos descrevendo o perfil.

Na verdade, faz-se a integral da seguinte forma:

$$F = A(x_f - x_0) + B\int_{x_0}^{x_f} y dx$$

Assim, somente y, ao invés de F, é integrado ao longo do perfil, tornando o algoritmo mais rápido.

Além disso, essa integral, que representa a área A do perfil, é aproveitada para calcular as posições x e y do seu baricentro. Essas posições, também utilizando integração por trapézio, são dadas pelas expressões abaixo:

$$x_g = \frac{1}{A}\int_{x_0}^{x_f} xy dx$$

$$y_g = \frac{1}{A}\int_{x_0}^{x_f} y^2 dx$$

3.4.2. Detecção do Final da Colisão

Uma vez que foi utilizado um mesmo algoritmo integrando continuamente do início da colisão até a condição de parada da simulação, foi preciso efetuar uma verificação de quando a colisão propriamente dita terminava. Este dado é importante pois determina o instante em que a força devido à colisão deixa de atuar sobre os carros e estes passam a executar trajetórias influenciadas apenas pelo atrito pneus-solo.

Inicialmente propôs-se como condição de parada, com base na ref. 2, o fato de que, ao término da colisão, um ponto específico dos dois carros apresentava a mesma velocidade. Este ponto é o centro da colisão (aproximado pelo centro geométrico do perfil de deformação). Entretanto, efetuando-se alguns testes foi possível observar que em várias situações esta condição não era satisfeita, fazendo com que a força devida à colisão permanecesse agindo por muito mais tempo. É simples observar, por exemplo, que no caso em que um dos carros já tem velocidade perpendicular a do outro no instante da batida, essa condição nunca será satisfeita.

Para solucionar este problema, foi adicionada uma segunda condição, geométrica. Essencialmente, quando não existir mais contato entre os perímetros dos carros, não há mais colisão. No caso de acidentes em que há mais de uma colisão consecutiva entre os veículos envolvidos, essa hipótese não é válida. Esses casos, porém, não são tratados nesse programa. Para simplificar o algoritmo, tomou-se um perfil retangular para os veículos. A figura abaixo representa um caso em que há contato entre os veículos.

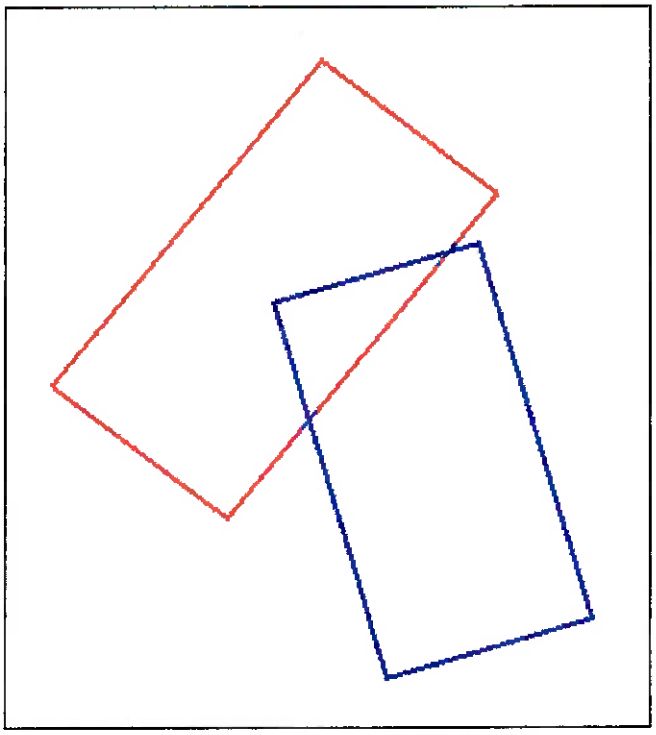


Figura 10 - Contato entre os Veículos

A primeira solução que se poderia pensar para este problema seria a verificação da interseção entre as retas que compõem os retângulos. Este problema, entretanto, envolve a solução de dezesseis sistemas lineares de duas incógnitas por duas equações. O fato dele ter de ser resolvido a cada passo de integração do algoritmo causa um impacto significativo no tempo de execução do programa.

Observou-se entretanto, que nos casos tratados, as interseções possíveis sempre apresentariam um ou mais vértices de um dos retângulos dentro do outro. A figura abaixo apresenta exemplos de casos possíveis e impossíveis na representação de colisões entre veículos.

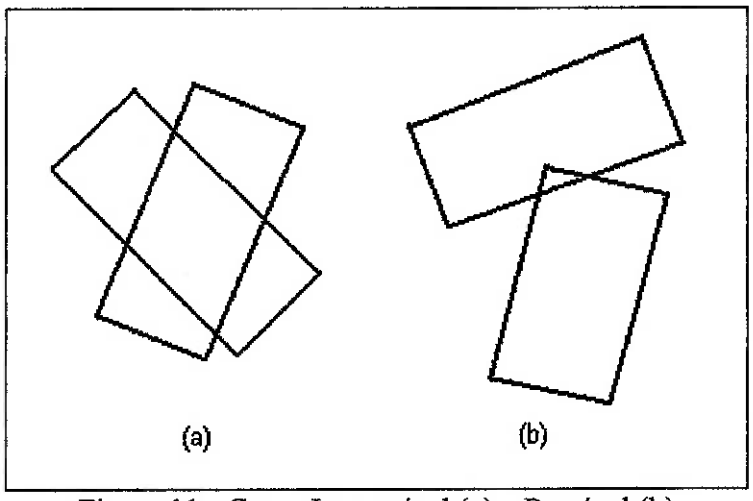


Figura 11 - Casos Impossível (a) e Possível (b)
de Colisão

Como se vê no caso (a), os veículos teriam de se sobrepor completamente, um caso que é pouco factível e de qualquer forma, não é tratado por esta versão do programa. Desta forma, utilizou-se um algoritmo alternativo. Este algoritmo se baseia no fato de que o produto escalar entre o vetor distância de um ponto a uma reta, e um vetor normal desta reta será positivo se o ponto estiver do mesmo "lado" para o qual aponta a normal. A figura na página seguinte ilustra esta propriedade.

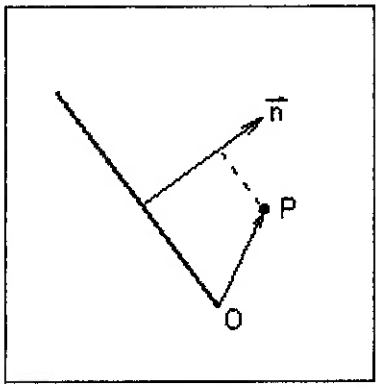


Figura 12 - Propiedade geométrica
usada no algoritmo

Sendo assim, tomando-se os vértices de um retângulo, e fazendo-se os produtos escalares da distância deste vértice até pontos arbitrários das retas que compõem o outro retângulo com as normais que "apontam para dentro do retângulo" das mesmas retas, tem-se que o vértice somente se encontra dentro do segundo retângulo se todos os produtos escalares forem positivos.

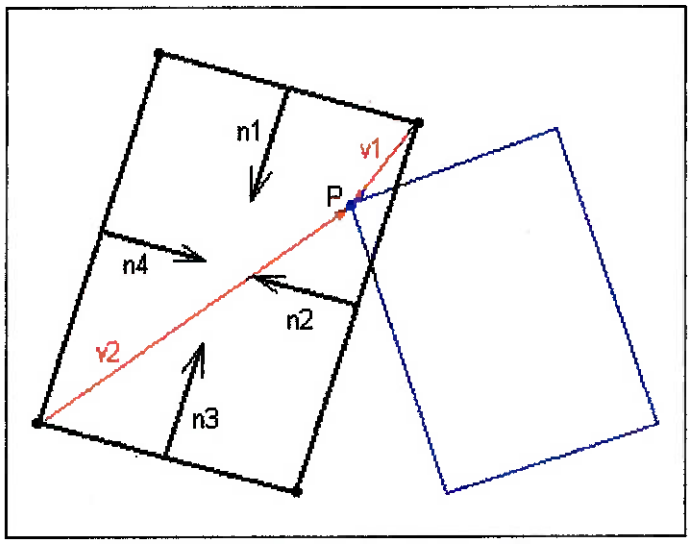


Figura 13 - Ilustração do Algoritmo

Por conveniência, os pontos arbitrários foram tomados como os vértices do segundo retângulo, de forma a reduzir ainda mais o número de cálculos necessários. Finalmente, dado que os lados do retângulo são perpendiculares dois a dois, não é necessário calcular efetivamente suas normais.

3.4.3. Integração do Sistema de Equações Diferenciais

A base do algoritmo de integração foi um método de Runge-Kutta de quarta ordem com passo adaptativo. As equações diferenciais dos dois veículos são resolvidas separadamente, mas simultaneamente e nos mesmos intervalos de tempo (passos de integração iguais), pois estas equações são desacopladas (entre os dois veículos, ver item 3.2).

O controle de passo é feito comparando-se o resultado de um passo de integração com métodos de Runge-Kutta de terceira e quarta ordens. Uma vez que o maior interesse é na reconstituição das trajetórias dos veículos, calcula-se um erro baseado nos desvios de posição pelos dois métodos. Se o erro calculado for inferior a uma tolerância pré-determinada, o passo é aumentado por um fator inversamente proporcional à raiz quadrada do erro.

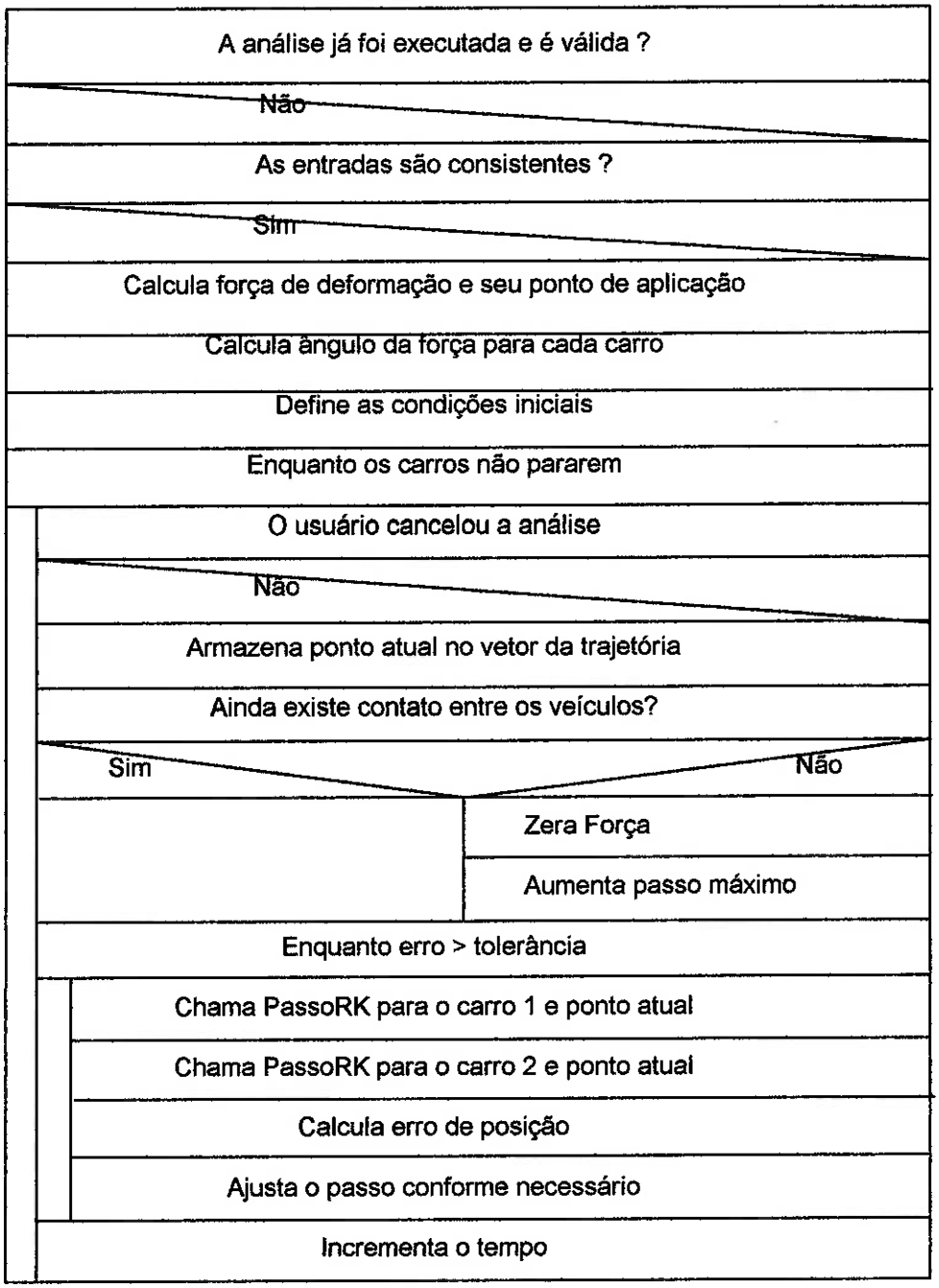
O valor máximo que o passo pode assumir também é limitado. Isto é necessário principalmente durante a fase em que a força de colisão atua, pois existem desacelerações muito bruscas, que levam a grandes imprecisões numéricas caso o passo varie muito. Uma vez que a força deixa de atuar, o limite para o passo máximo é aumentado.

Caso o erro obtido com o passo atual seja maior que o máximo tolerado, o passo é diminuido e repete-se a última integração. Caso contrário, procede-se à próxima integração.

A cada passo de integração, duas condições são verificadas. A primeira é a de término da colisão, quando a força de contato deixa de atuar. A segunda é o critério de parada da simulação, que consiste basicamente em verificar se as velocidades (angulares e lineares) dos dois veículos são nulas.

Os diagramas N-S nas páginas seguintes ilustram este algoritmo

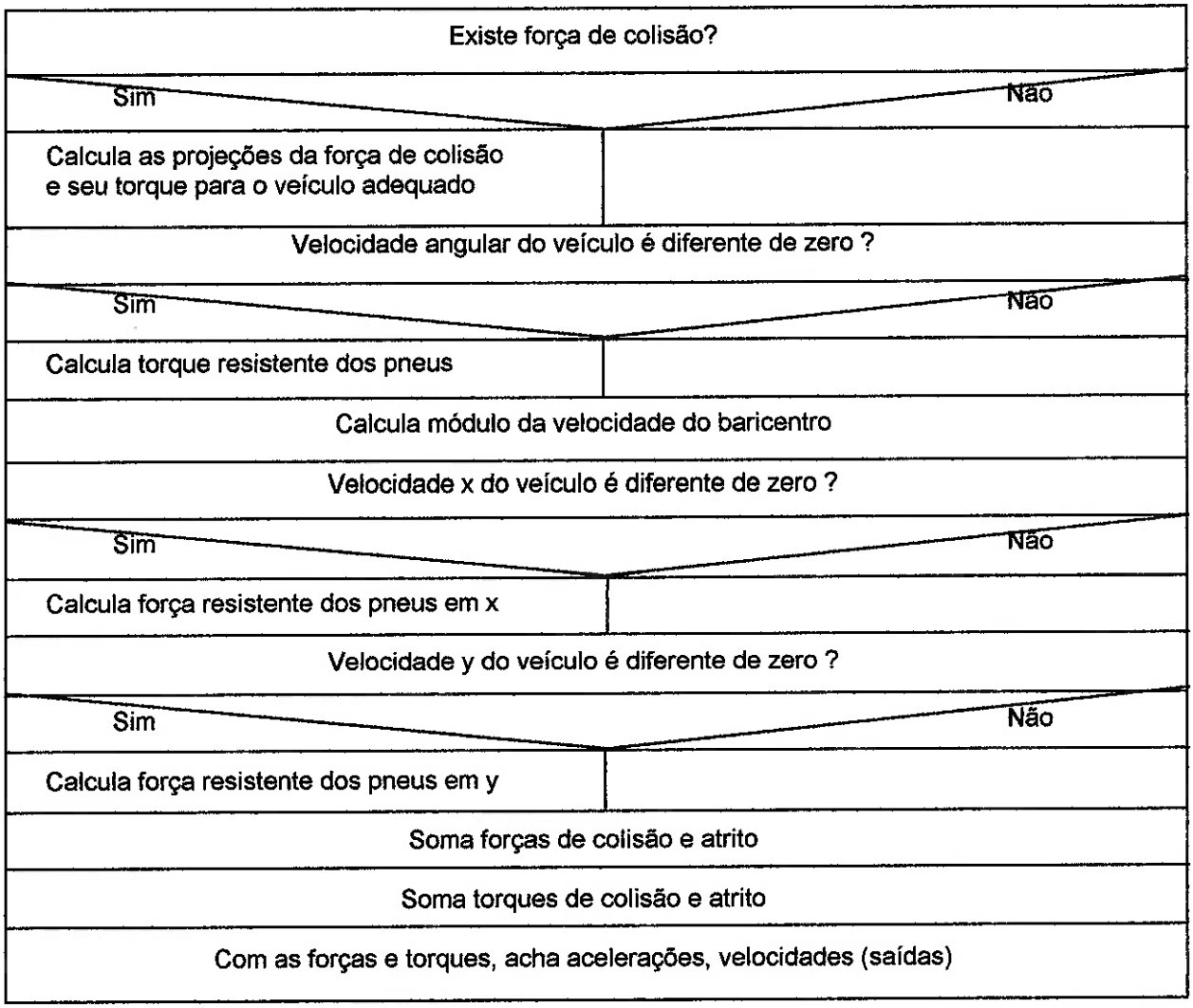
Algoritmo de integração:



PassoRK:

Calcula o vetor K1 chamando EqDifs para o ponto recebido
Para i de 1 a 6
ponto(i) = ponto recebido(i) + K1(i).passo/2
Calcula o vetor K2 chamando EqDifs para ponto
Para i de 1 a 6
ponto(i) = ponto recebido(i) + K2(i).passo/2
Calcula o vetor K3 chamando EqDifs para ponto
Para i de 1 a 6
ponto(i) = ponto recebido(i) + K3(i).passo/2
Calcula o vetor K4 chamando EqDifs para ponto
Para i de 1 a 6
ponto(i) = ponto recebido(i) + K2(i).passo.3/4
Calcula o vetor K3b chamando EqDifs para ponto
Para i de 1 a 6
Saída RK3(i) = ponto recebido(i) + passo.[2K1(i) + 3K2(i) + 4K3b(i)]/9
Saída RK4(i) = ponto recebido(i) + passo.[K1(i) + 2K2(i) + 2K3(i) + K4(i)]/6

EqDifs:


Existe força de colisão?
Sim
Não
Calcula as projeções da força de colisão e seu torque para o veículo adequado
Velocidade angular do veículo é diferente de zero ?
Sim
Não
Calcula torque resistente dos pneus
Calcula módulo da velocidade do baricentro
Velocidade x do veículo é diferente de zero ?
Sim
Não
Calcula força resistente dos pneus em x
Velocidade y do veículo é diferente de zero ?
Sim
Não
Calcula força resistente dos pneus em y
Soma forças de colisão e atrito
Soma torques de colisão e atrito
Com as forças e torques, acha acelerações, velocidades (saídas)

3.4.4. Precisão Numérica

Ao longo do processo de desenvolvimento e teste dos algoritmos aqui apresentados, ocorreram problemas de convergência devido a erros de precisão.

O primeiro tipo de problema consistiu na verificação da igualdade entre dois valores. Devido à imprecisão numérica, estes valores nem sempre resultavam exatamente iguais. Desta forma, foi preciso introduzir tolerâncias para que, se a diferença entre os valores comparados fosse menor que as mesmas, os valores seriam considerados iguais.

Outro problema encontrado surgiu na imprecisão das funções trigonométricas. Dado que estas funções são em geral implementadas como desenvolvimentos finitos de séries, existe uma imprecisão inerente a elas, que levava a erros principalmente em cálculos envolvendo projeções de vetores. Esta imprecisão tornava-se evidente em ângulos como 0 e 90 graus.

Para solucionar este problema, as funções trigonométricas como seno e cosseno foram encapsuladas, de modo que o seu resultado era comparado com uma tolerância e convertido para um valor adequado. Por exemplo, se um valor retornado pela função seno fosse inferior à uma tolerância pré-definida, este valor seria retornado como zero.

3.5. Desenvolvimento da Interface

Conforme decidido no estudo de viabilidade, a interface de usuário foi implementada com uso de "tabs" que agrupam conjuntos de dados que devem ser fornecidos pelo usuário. Os "tabs" desenvolvidos para esta interface foram:

- entrada de dados dos carros
- entrada do perfil de deformação e posição do mesmo
- entrada das condições iniciais de posição e velocidade dos veículos.

Adicionalmente, foi criado um "tab" para fornecer a resposta do problema.

De forma geral, os dados que são editados nos "tabs" são cópias locais daqueles que serão utilizados na análise numérica. Somente quando se troca de "tab" estes dados são efetivamente armazenados.

Deve-se ressaltar que, em ambiente Windows de 16 bits, não existe uma classe ou tipo de controle padronizado que implemente os "tabs". Desta forma, foi preciso desenvolver uma classe própria que tivesse esta funcionalidade. No caso, foi implementada uma classe que cria uma janela principal e contém dentro de si uma lista de caixas de diálogo que compõem os "tabs".

Nesta seção, serão comentados somente alguns aspectos peculiares da interface, sem a intenção de explicá-la ou descrevê-la como um todo. Como o programa é parte integrante deste trabalho, a melhor forma de observar a interface é utilizando-o, ou através do manual, no Apêndice IV.

3.5.1. Entrada de Dados dos Carros

Nos tabs de entrada de dados, o usuário pode escolher um veículo existente no banco de dados ou criar um novo. A listagem de carros existentes no banco de dados é copiada para uma caixa de listagem cada vez que o banco de dados é atualizado.

Quando os veículos são selecionados ou criados, os dados são copiados para objetos independentes que serão utilizados nos algoritmos de solução.

Quando um veículo novo é criado, é preciso utilizar o botão de Salvar para que o objeto contendo os dados novos seja copiado na lista ligada do banco de dados. O mesmo vale caso se deseje alterar permanentemente as características de um carro já existente no banco.

Os outros controles deste "tab" são controles comuns do Windows que permitem a edição dos dados.

3.5.2. Entrada de Dados da Deformação

Na entrada de dados da deformação, o usuário pode digitar o perfil de deformação que é colocado em uma caixa de listagem, sob forma de strings de texto. Estas strings são formatadas adequadamente de modo a formar uma tabela de duas colunas.

No mesmo "tab" existe um quadro no qual uma figura simplificada do veículo é mostrada, juntamente com o perfil de deformação atual. Quando o usuário executa um click do mouse sobre esta figura, o programa calcula a posição equivalente em coordenadas do carro, permitindo um posicionamento visual do perfil de deformação. Este posicionamento funciona através da captura de eventos de click do mouse.

O restante dos controles presentes têm comportamento comum.

3.5.3. Entrada de Dados do Cenário

No "tab" do cenário, optou-se por apresentar um quadro no qual pode-se observar figuras simplificadas dos veículos que refletem os dados de posição entrados como condições iniciais.

O usuário pode modificar estes dados através de caixas de edição. A cada mudança de valores nas caixas relacionadas à geometria (posição e ângulo) do problema, o programa emite uma mensagem do Windows que faz com que o quadro anteriormente descrito seja atualizado.

No processo de atualização dos dados do problema, os ângulos são armazenados em graus (conforme entrado pelo usuário) sendo convertidos para radianos apenas no momento da solução do problema. Assim, evita-se erros devido à precisão que ocorreriam caso esta conversão ocorresse sempre que o "tab" fosse atualizado.

3.5.4. Saída

Quando o usuário seleciona o tab de saída, o programa primeiramente efetua verificações de consistência dos dados entrados, e caso esta verificação tenha sucesso, procede com a execução dos algoritmos de solução.

Enquanto o algoritmo de solução é processado, uma caixa de diálogo permanece na tela informando sobre o andamento do processo, e permite que o mesmo seja cancelado. Para que isso seja possível, o algoritmo de solução faz sucessivas chamadas a funções do Windows para que o processamento dos comandos do usuário (como um click no botão Cancelar da caixa de diálogo) seja feito.

Quando o algoritmo termina, o tab é atualizado, mostrando a trajetória dos veículos. Existem controles que permitem ajustar alguns parâmetros de exibição desta trajetória. Existem também opções para a exibição de gráficos, como por exemplo velocidade x tempo, no lugar da trajetória.

O processamento destas saídas foi implementado sob forma de diversas funções independentes que permitem a exibição dos resultados em vários dispositivos. Isto foi feito assim para que se pudesse re-utilizar o código para enviar as saídas para a impressora.

3.5.5. Impressão

Conforme comentado no item sobre o tab de Saídas, a impressão do programa utiliza boa parte do código desenvolvida anteriormente. Adicionalmente foram criadas funções que efetuam a impressão dos dados dos veículos utilizados na simulação, assim como as condições iniciais.

Para manter o conceito da orientação a objetos, foi desenvolvida uma classe separada que trata de todos os aspectos da impressão, isto é, obtenção dos dados, configuração dos parâmetros de impressão e execução da impressão propriamente dita.

3.5.6. Help On-line

Foi implementado no programa tanto help on-line como context-sensitive help (isto é, ao se pressionar F1 com o cursor selecionando um controle ou parte do programa, obtém-se imediatamente help relacionado ao mesmo). O Visual C++ 1.52 disponibiliza mecanismos para simplificar esta implementação, através das classes da MFC.

Com estes mecanismos, a implementação do help se concentrou principalmente na composição do arquivo de help que é utilizado junto ao programa.

3.6. Testes

A etapa de testes do protótipo do programa foi prejudicada pela dificuldade em encontrar dados empíricos de colisões que pudessem ser comparados às saídas do programa. Isso ocorreu em decorrência da inexistência de tais dados no Brasil, e ao fato de que as condições dos testes nos Estados Unidos (de onde possuíamos resultados) são diferentes das tratadas neste protótipo. Além disso, os conjuntos de dados obtidos dos Estados Unidos eram incompletos, faltando, em geral, os perfis de deformação dos veículos.

Por exemplo, na grande maioria dos testes realizados nos Estados Unidos, os veículos possuem freios ABS e câmbio automático. Estes dois efeitos (frenagem parcial e freio motor) não são tratados por esta versão do programa. Assim, torna-se difícil uma comparação de resultados.

Levando-se em conta estas limitações, os testes que foram feitos consistiram em:

-verificar a consistência do algoritmo de integração utilizado, comparando-se com os resultados da integração do modelo matemático pelo MATLAB

-verificar os resultados do modelo em casos simples como colisões frontais e laterais, comparando os resultados com cálculos feitos manualmente.

Os quatro testes mais representativos efetuados são listados a seguir, com comentários sobre cada um. São eles:

- colisão frontal entre dois carros iguais;
- colisão traseira;
- colisão lateral sobre os baricentros dos carros;
- colisão lateral com os baricentros não alinhados.

Teste 1

Neste teste, sendo os veículos iguais e colidindo frontalmente, o resultado esperado seria uma colisão anelástica na qual os veículos simplesmente se aproximam de uma distância proporcional ao amassamento e param. Dadas as hipóteses utilizadas no programa, pode-se calcular o tempo de duração desta colisão manualmente:

$$t = V_0 \cdot \left[\frac{(2 \cdot A \cdot ys + 2 \cdot B \cdot ys \cdot c) + (\mu \cdot M \cdot g)}{M} \right]^{-1}$$

onde $V_0 = 28{,}8$ km/h
$M = 1383$ kg
$A = 45359$ N/m^2
$B = 296475$ N/m^2
$ys = 0.853$ m
$c = 0{,}25$ m
$\mu = 0{,}7$
$g = 9{,}81$ m/s^2

De onde se obtém $t = 0{,}0518$s

Também pode-se calcular a distância percorrida por um dos veículos:

$$\Delta s = \Delta V \cdot \Delta t / 2$$

$\Delta s = 0{,}207$ m

As figuras a seguir apresentam os dados utilizados neste teste e os resultados do mesmo. Os perfis de deformação utilizados foram retângulos ao longo de toda a extensão da frente dos veículos e com 0,25m de profundidade.

O tempo registrado para o final da colisão foi de 0,0518s. Como se pode observar, o modelo se comportou adequadamente neste teste.

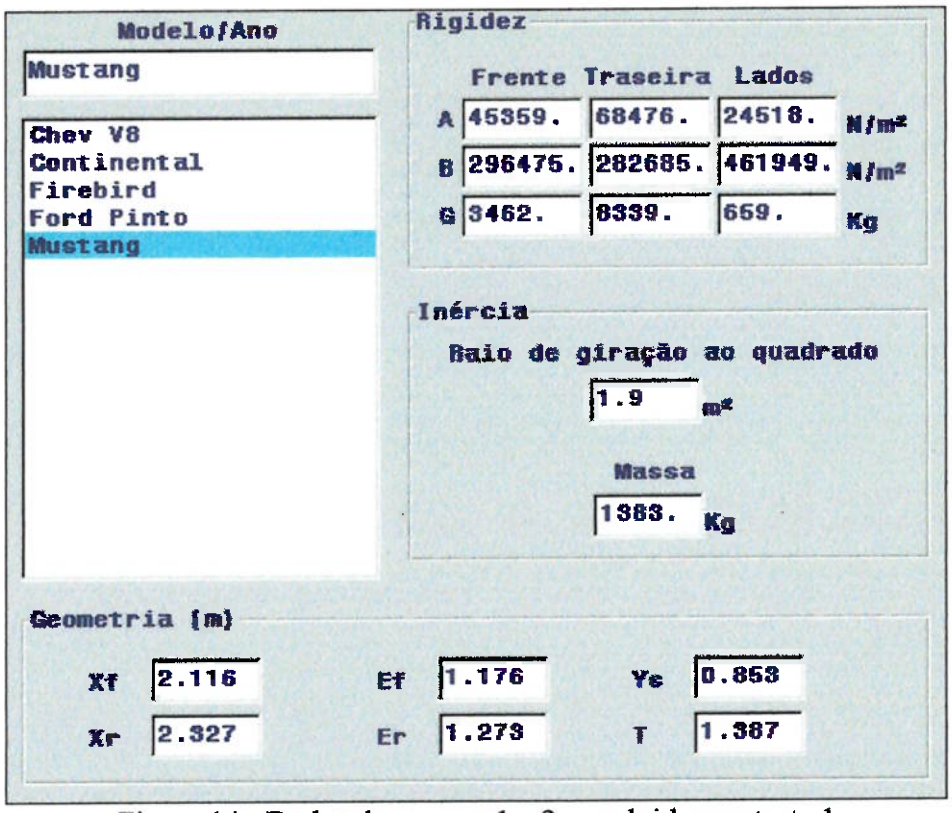


Figura 14 - Dados dos carros 1 e 2 envolvidos no teste 1

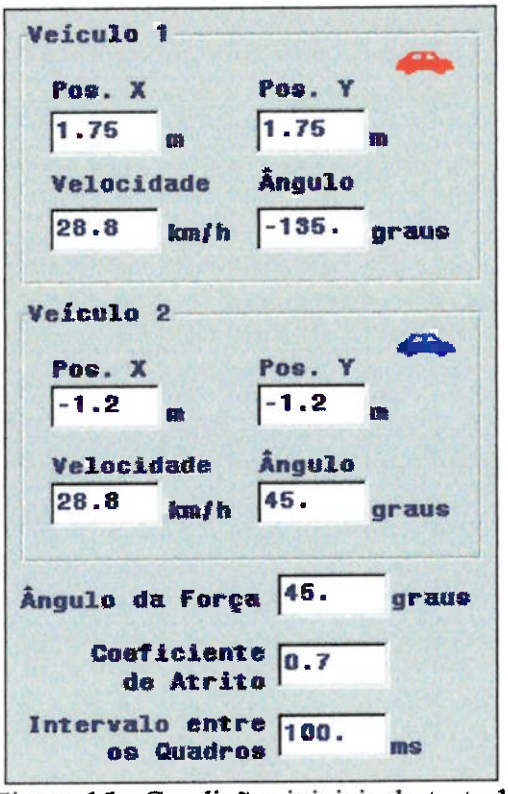


Figura 15 - Condições iniciais do teste 1

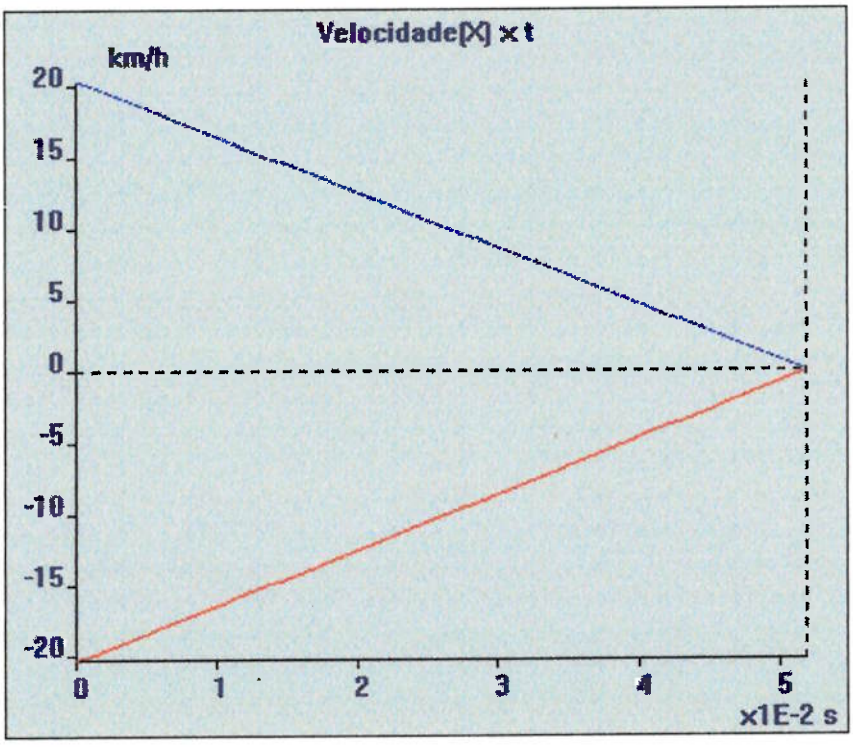


Figura 16 - Velocidades dos veículos durante a simulação

Teste 2

Neste teste, veículos diferentes colidem, estando um deles parados inicialmente. Assim, têm-se parâmetros de massa e rigidez diferentes. Ainda assim, neste caso como no anterior é possível calcular o tempo da colisão facilmente:

As acelerações dos veículos são dadas por:

$$a1 = \frac{(2 \cdot A1 \cdot ys1 + 2 \cdot B1 \cdot ys1 \cdot c1) + (\mu \cdot M1 \cdot g)}{M1}$$

$$a2 = \frac{(2 \cdot A2 \cdot ys2 + 2 \cdot B2 \cdot ys2 \cdot c2) + (\mu \cdot M2 \cdot g)}{M2}$$

O tempo de final da colisão (t_c) será, então:

$$t_c = \frac{V_{01}}{a1 + a2}$$

O tempo de final da simulação (t_f) será dado por:

$$t_f = t_c + \frac{a2 \cdot t_c}{\mu \cdot g}$$

onde A1 = 52889 N/m^2
B1 = 324054 N/m^2
A2 = 62521 N/m^2
B2 = 89632 N/m^2
M1 = 998 kg
M2 = 1923 kg
ys1 = 0,772 m
ys2 = 0,978
c1 = 0,1 m
c2 = 0,05 m
μ = 0,7
g = 9,81 m/s^2

Efetuando estas operações, obtém-se t_c = 0,0499s e t_f = 0,503s

A trajetória esperada, neste caso, seria uma linha reta nos dois veículos, com comprimento proporcional às massas dos veículos.

As figuras abaixo mostram os dados utilizados e os resultados. Os perfis de deformação utilizados foram retângulos com comprimentos iguais às larguras dos respectivos veículos. A profundidade do perfil do veículo 1 foi de 0,1m, e do perfil carro 2 foi 0,05m. O final da colisão ocorreu em 0,0499s. A simulação terminou em 0,5034s.

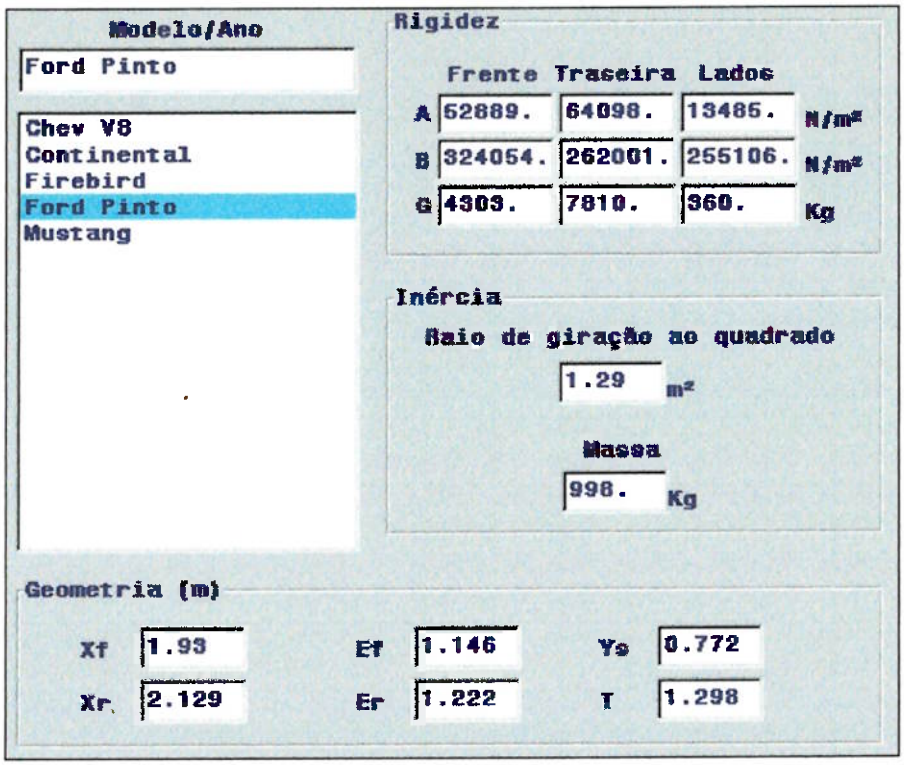


Figura 17 - Dados do veículo 1 do teste 2

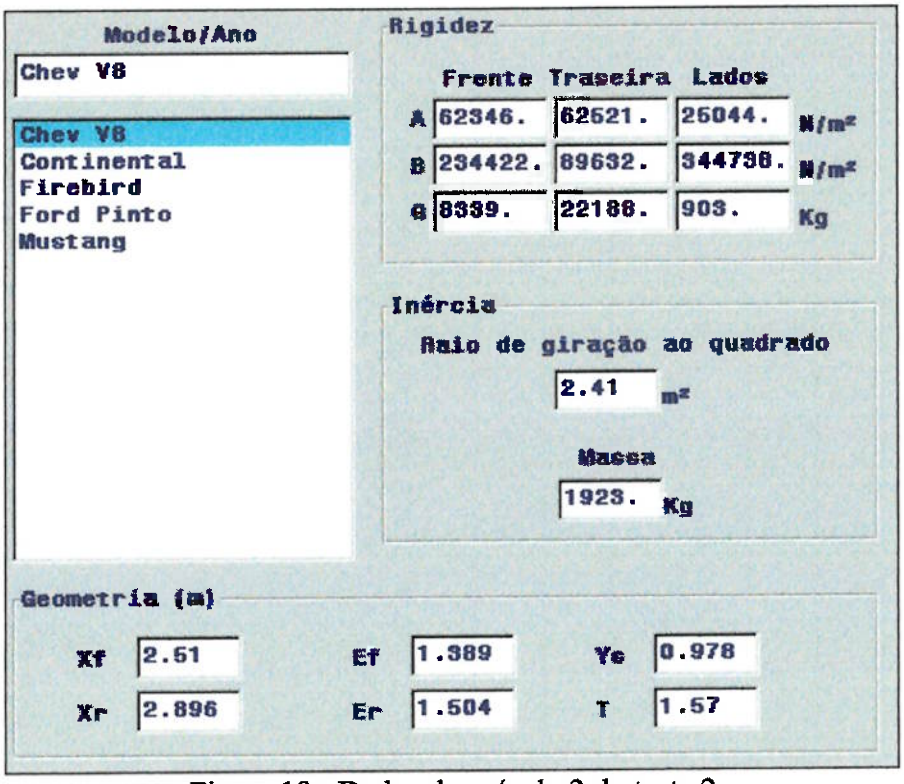


Figura 18 - Dados do veículo 2 do teste 2

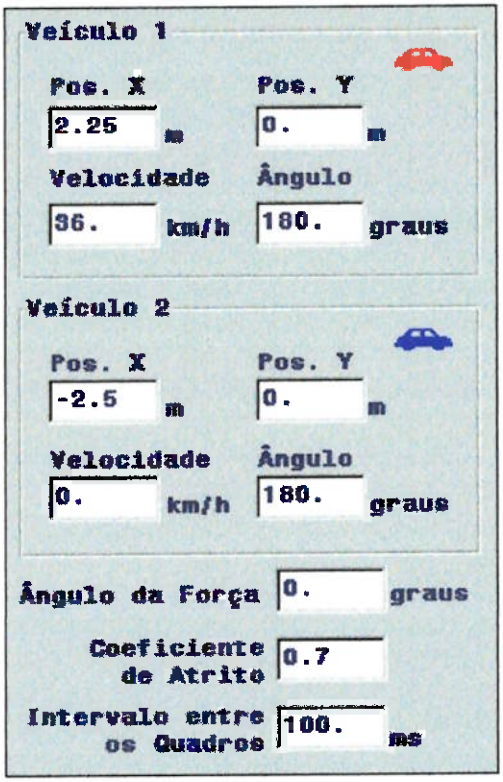


Figura 19 - Condições iniciais dos veículos para o teste 2

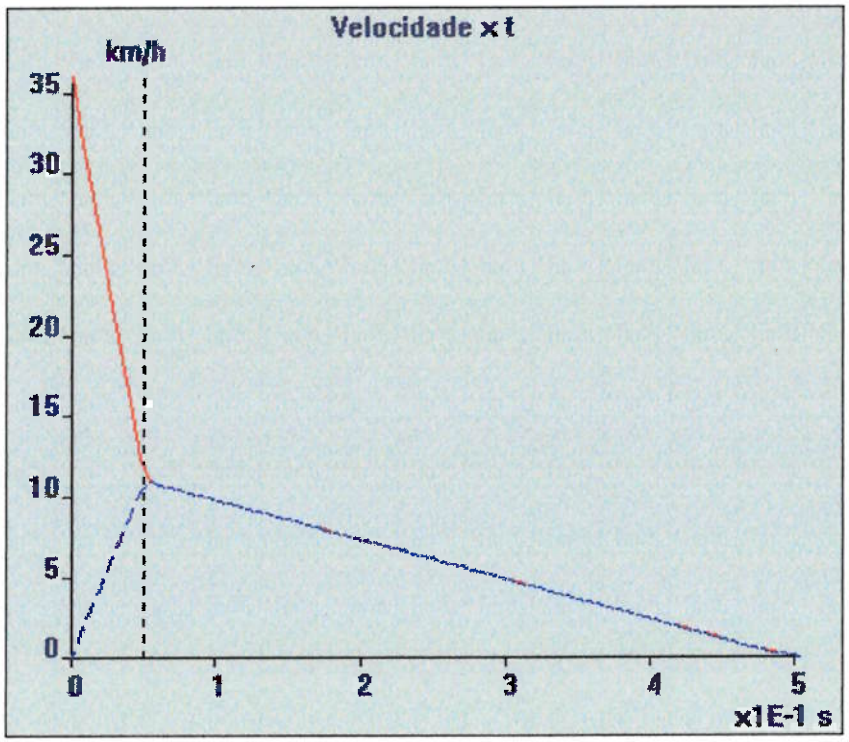


Figura 20 - Velocidades dos veículos no teste 2

Teste 3

Este teste foi utilizado para verificar as equações de momento utilizadas no modelo, especialmente após a adoção do modelo simplificado. Um veículo colide contra a lateral de outro, que está inicialmente parado. De acordo com o modelo, a trajetória dos veículos neste caso deveria ser reta pois a colisão se dá sobre o baricentro do veículo parado.

Como neste caso também existe um veículo em movimento colidindo com outro inicialmente estacionado, a dedução dos tempos é análoga ao do caso 2. A difierença significativa ocorre no cálculo da força de colisão no veículo 2, pois é preciso efetuar a integral sobre o perfil.

Os tempos calculados foram:

tempo de final da colisão: 0,0424s

tempo de final da simulação: 0,0596s

Os dados dos veículos são os mesmos do teste 2, de forma que não serão repetidos. O perfil de deformação do veículo 1 é um retângulo da largura do mesmo e com profundidade de 0,20m. O perfil do veículo 2 é dado a seguir, assim como os resultados do teste.

Os tempos obtidos na simulação foram: tempo de final da colisão = 0,0424s e tempo de final da simulação = 0,5959s

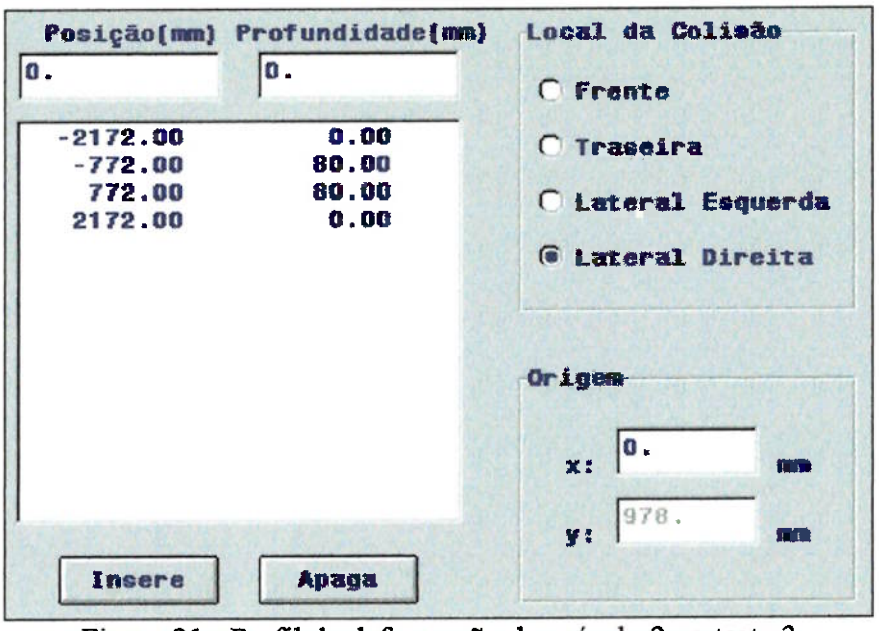


Figura 21 - Perfil de deformação do veículo 2 no teste 3

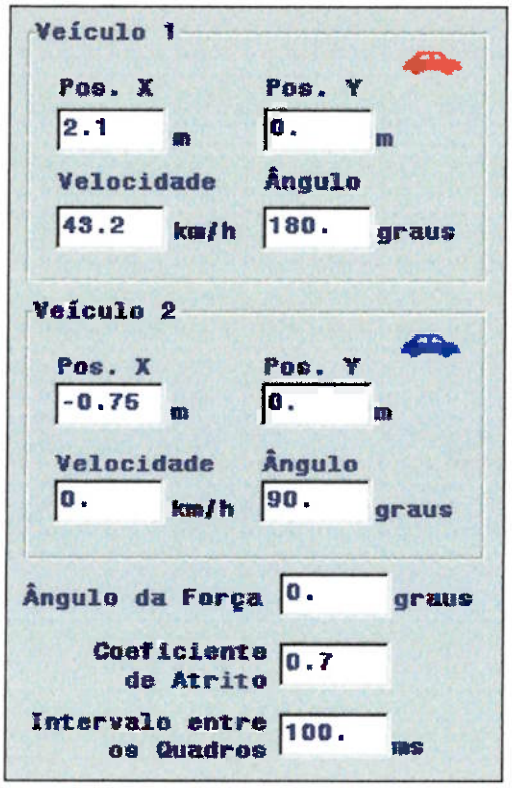


Figura 22 - Condições iniciais dos veículos no teste 3

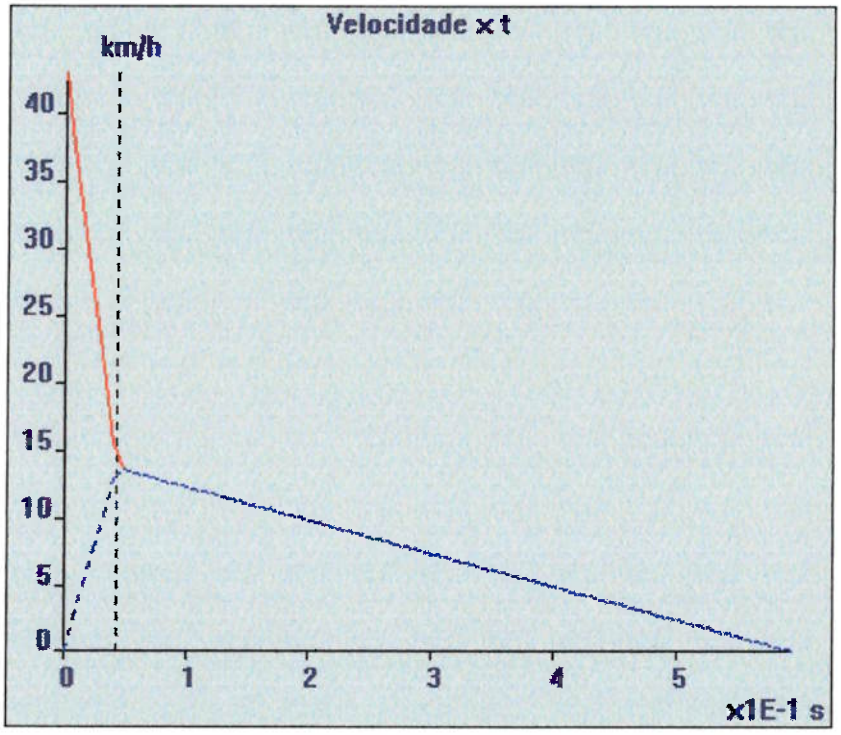


Figura 24 - Velocidades dos veículos no teste 3

Teste 4

O teste 4 é similar ao teste 3. Entretanto, os dois veículos têm velocidades iniciais e a colisão não ocorre sobre o baricentro do segundo veículo. Desta forma, surgem momentos sobre os dois veículos.

Neste teste não foi realizada nenhuma comparação com cálculos feitos manualmente, havendo mais interesse em verificar o comportamento do modelo em um caso com atuação de forças e momentos durante a colisão.

Os parâmetros deste teste, quanto aos veículos, são similares aos dos testes 2 e 3. Os perfis de deformação, condições iniciais e resultados são mostrados a seguir. Na figura 28, que apresenta a trajetória, os quadros representam intervalos de 100ms. A trajetória em vermelho corresponde ao veículo 1, e a trajetória em azul, ao veículo 2.

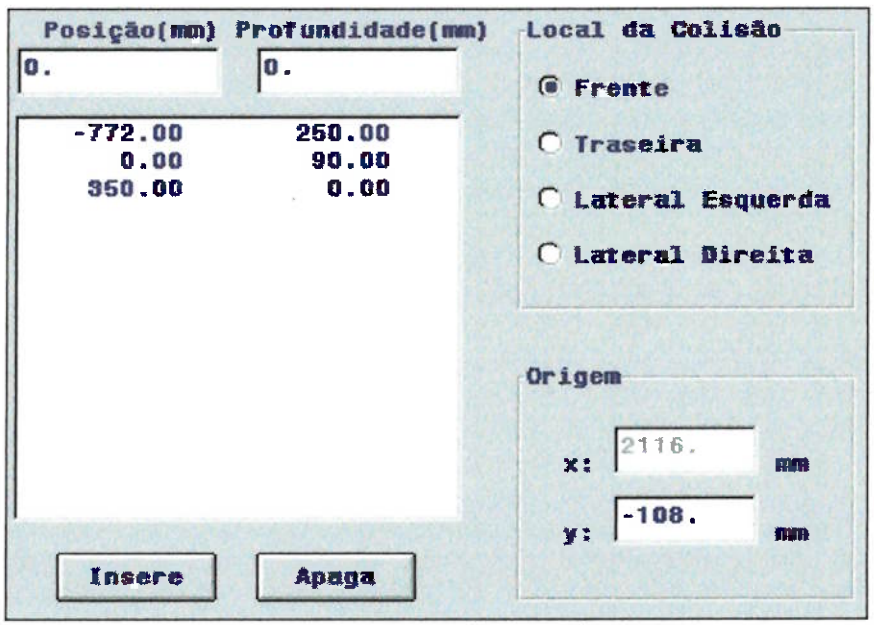


Figura 25 - Perfil de deformação do veículo 1 no teste 4

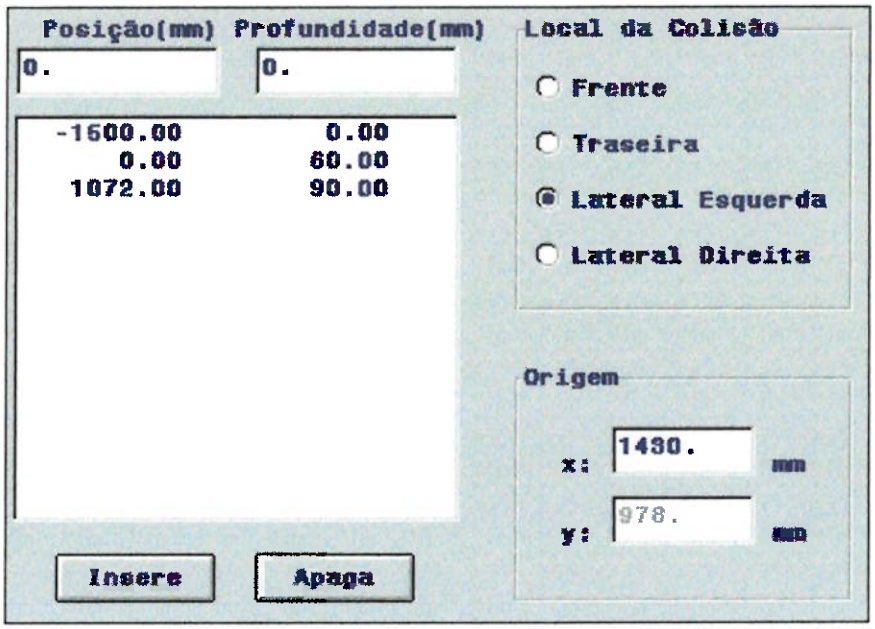


Figura 26 - Perfil de deformação do veículo 2 no teste 4

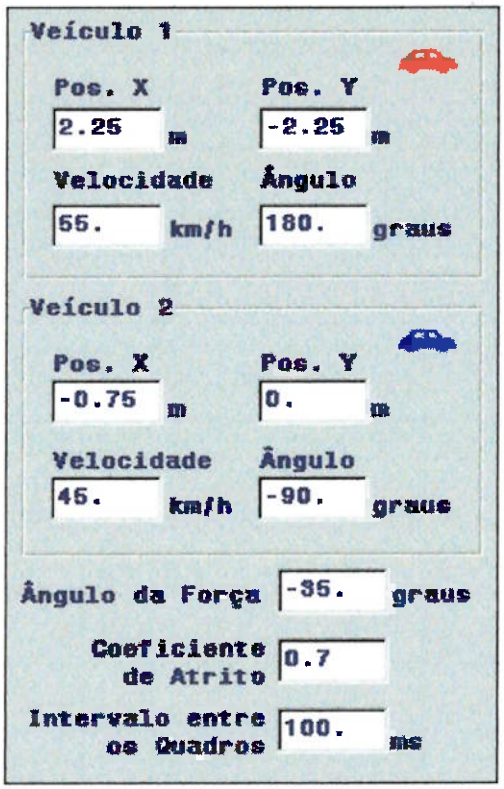


Figura 27 - Condições iniciais dos veículos no teste 4

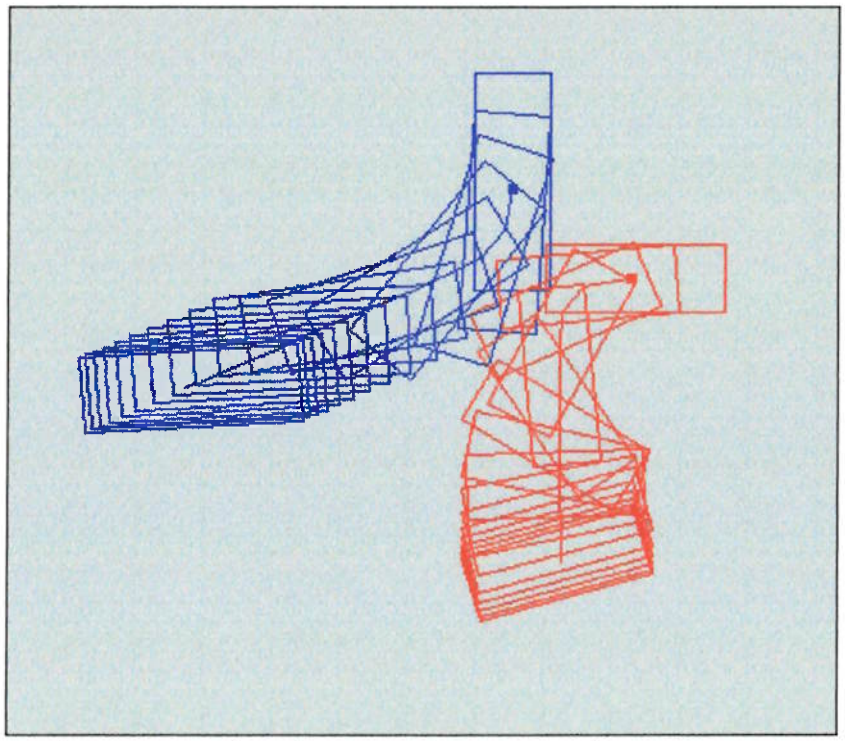


Figura 28 - Trajetórias percorridas pelos veículos no teste 4

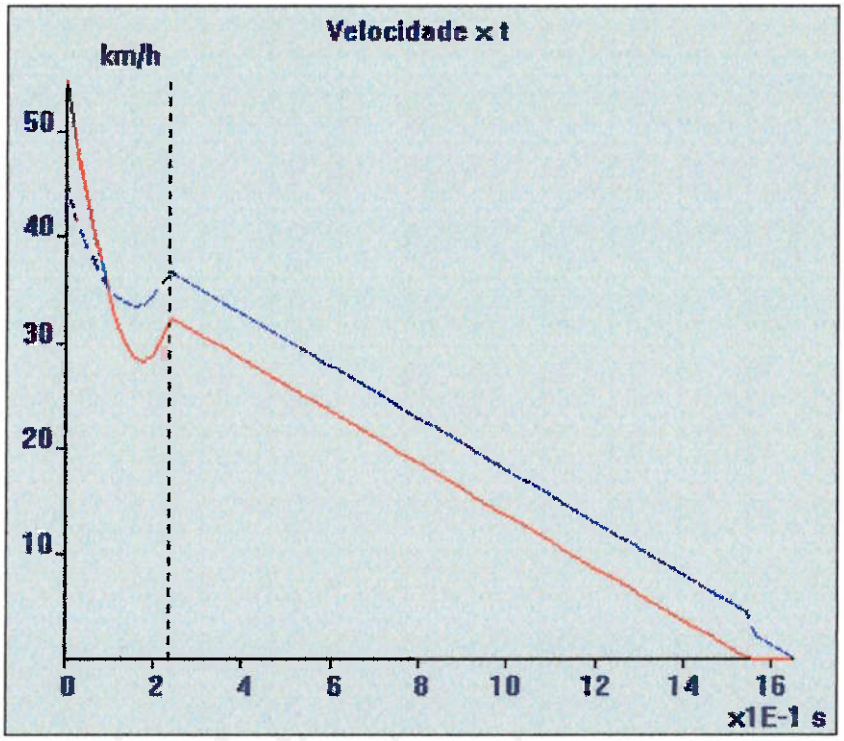


Figura 29 - Velocidades dos veículos no teste 4

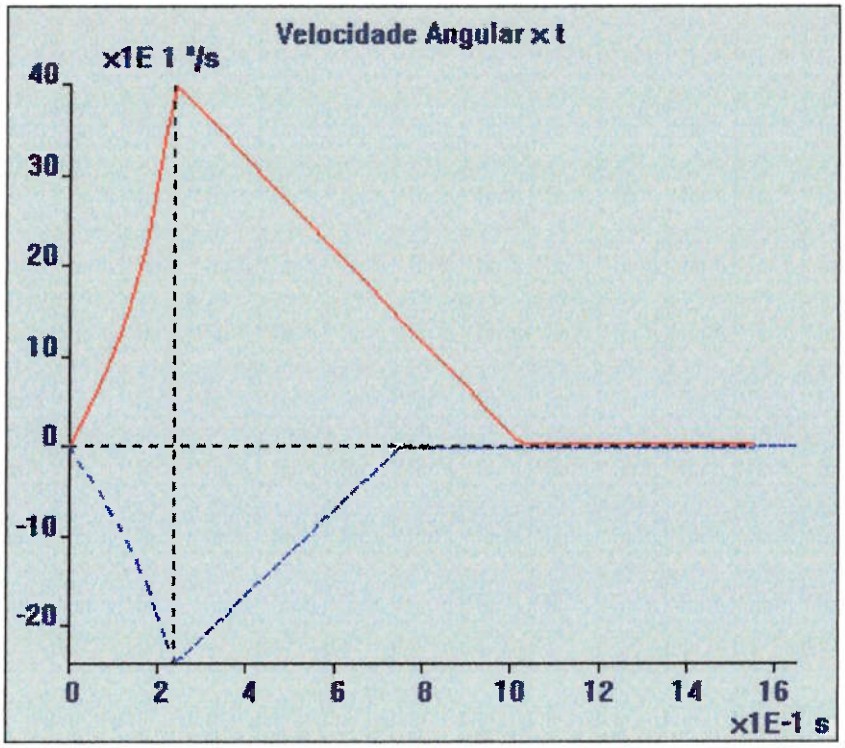


Figura 30 - Velocidades angulares dos veículos no teste 4

A partir destes testes, principalmente do teste 4, pode-se observar determinadas regiões dos gráficos de velocidades linear e angular que correspondem a situações físicas distintas. O gráfico na página seguinte ilustra este fato:

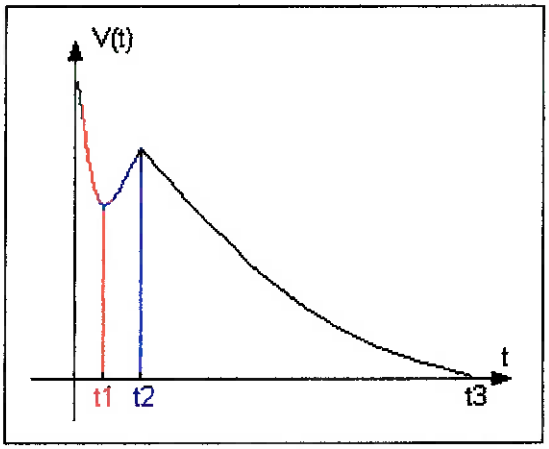


Figura 31 - Regiões da simulação

A região entre 0 e t1 corresponde à fase em que ocorre o amassamento na colisão. Nesta região, os centros de colisão estão se aproximando. A região entre t1 e t2

corresponde à separação dos veículos ainda sob influência da força de contato entre os veículos. E finalmente, na região de t2 a t3 cada veículo se move sujeito somente às forças de frenagem, até parar no instante t3.

Nos testes 1 a 3, não ocorre a separação dos veículos, e isso pode ser observado nos seus gráficos. Adicionalmente, no teste 1, o fim da colisão também corresponde ao fim da simulação, ou seja, não há a região de frenagem pura.

Além disso, no gráfico de velocidade angular pode-se verificar que a hipótese de Marquard (que a rotação termina sempre antes do deslizamento) é satisfeita no caso 4.

Finalmente, observou-se que as trajetórias percorridas pelos baricentros dos veículos após a colisão são sempre linhas retas, conforme deve-se esperar visto que somente o atrito age sobre os veículos.

3.7. Conclusão

Nesta fase do projeto de formatura, a principal atividade foi o estudo mais aprofundado das soluções escolhidas no estudo de viabilidade, permitindo a implementação de uma versão Beta do programa (protótipo).

Devido a algumas limitações do modelo, não foi possível implementar algumas das características desejadas no programa, como colisão com obstáculos. Isto está comentado na seção 3.2.

Além disso, não foram encontrados dados confiáveis para implementação de um modelo para frenagem ABS. Foi sugerido que um mero aumento nos coeficientes de atrito pneu-solo, seria suficiente para simular este efeito, mas não foi encontrado material que corroborasse esta afirmação. O desenvolvimento e testes de um modelo para frenagem ABS, partindo do zero, foge ao escopo deste trabalho.

Ao longo do último semestre houve novamente alguns problemas para obtenção de informações mais acuradas sobre o modelo de deformação, o que levou a um pequeno atraso com relação ao cronograma do Apêndice I. Este atraso, no entanto, foi

sanado a contento até meados de outubro. Não foram obtidos dados de teste adequados, de modo que só foi possível realizar testes mais simples, que podiam ser calculados manualmente. Por isso também, não foi possível testar a correlação do modelo de colisão.

Quanto à implementação dos algoritmos de solução do problema, surgiram as seguintes dificuldades (ver seção 3.4):

- Devido às altas acelerações a que são submetidos os veículos durante a colisão, durante este intervalo de tempo, foi necessário um cuidado especial com o algoritmo de integração das EDO's.

- Em função de problemas de precisão numérica, foi preciso definir tolerâncias para todas as comparações de variáveis, além de redefinir algumas funções matemáticas (por exemplo, a função "sin").

- A definição da condição de fim da colisão teve que ser revista, pois a condição recomendada por *McHenry* (ref. 2) não se aplicava a todos os casos testados

Apesar de todos estes problemas, a versão Beta produzida apresenta toda a funcionalidade inicialmente proposta para o trabalho, inclusive com características além das necessárias a um protótipo (como manual de usuário). O programa também atende o requisito de ser amigável e de fácil uso, visando atender a um público maior dentro da área de reconstrução de acidentes automobilísticos. Isso, aliado à falta de material sobre o assunto encontrados aqui no Brasil, leva à conclusão de que pode ser interessante continuar o desenvolvimento deste projeto.

Pode-se sugerir as seguintes áreas para futuro desenvolvimento:

- Obtenção de dados de acidentes reais para teste do modelo;

- Modelagem e implementação de colisão com obstáculos após a primeira colisão;

- Compilação de um banco de dados de veículos nacionais;
- Modelagem e implementação dos efeitos de suspensão e solo;
- Modelagem e implementação dos freios ABS;
- Heurística para determinação do ângulo da força de colisão;
- Determinação do perfil de deformação com base em imagens fotográficas;
- Estudo do impacto do acidente sobre os ocupantes do veículo.

4. Bibliografia

[1] MCHENRY, Raymond R. **The CRASH program - A simplified Collision Reconstruction Program.** Calspan Corporation, 1976.

[2] NATIONAL CENTER FOR STATISTICS AND ANALYSIS - ACCIDENT INVESTIGATION DIVISION. **CRASH3 User's Guide and Technical Manual.** Washington, D.C., U.S. Department of Transportation / National Highway Safety Admnistration, 1982.

[3] SOCIETY OF AUTOMOTIVE ENGINEERS. **Accident Reconstruction: Technology and Animation.** Vol. I. Society of Automotive Engineers, Inc., 1991.

[4] NISTROM, Gustav A.; STROTHER Charles E.; JAMES, Michael B. **Stiffness Parameters for Vehicle Collision Analysis.** Failure Analysis Associates, Inc.

[5] LIMPERT, Rudy; ANDREWS, Dennis F. **Linear and Rotational Momentum for Computing Impact Speeds in Two-car Collisions (LARM).** Prosource Software

[6] MCHENRY, Raymond R.; MCHENRY, Brian G. **CRASH97 Refinement of the Trajectory Analysis Procedure.** Society of Automotive Engineers, Inc., 1997.

[7] SOCIETY OF AUTOMOTIVE ENGINEERS. **Collision Deformation Classification - SAE J224:** SAE Recommended Practice. Society of Automotive Engineers, Inc., 1980.

[8] MCHENRY, Brian G. **SMAC-97 - Refinement of the Collision Algorithm.** Society of Automotive Engineers, Inc., 1997.

[9] UEYAMA, Masaru; MAKISHITA, Hiroshi; SAITO, Shuji. **Determination of Collision Configurations from Vehicle Deformation Patterns.** National Research Institute of Police Science.

[10] WEKEZER, Jerzy W.; OSKARD, Morton S.; LOGAN, Roger W.; ZYWICZ, Edward. **Vehicle Impact Simulation.** Journal of Transportation Engineering, 1993.

[11] ABE, Makoto; MORISAWA, Masaaki; SATO, Takeshi B. **Three-Dimensional Behavior of vehicle at perpendicular side collision: computer simulation using dynamic model.** Japan Society of Automotive Engineers, 1995.

[12] FISCHER, Raymond G.; HAERTIE, Joseph A. **Computer Modeling in New Vehicle Design.** Society of Automotive Engineers, Inc., 1985.

[13] CHEVA, Wichai; YASUKI, Tsuyoshi; GUPTA, Vikas; MENDES, Kolita. **Vehicle Development for Frontal/Offset Crash Using Lumped Parameter Modeling.** Society of Automotive Engineers, Inc.,1996.

[14] PRASAD, Priyaranjan; PDGAONKAR, Arvind J. **Static-to-Dynamic Amplification Factors for Use in Lumped-Mass Vehicle Crash Models.** Society of Automotive Engineers, Inc., 1981.

[15] GIACAGLIA, Giorgio E. O. **Mecânica Analítica.** Almeida Neves - Editores, Rio de Janeiro, LTDA., 1978.

[16] DIETMAROTTE, A. **Comparision and Realism of Crash Simulation Tests and Real Accident Situations for Biomechanical Movements in Car Colisions.** Accident Research Unit, Medical University of Hannover.

Adicionalmente, os seguintes *websites* foram utilizados como fonte de pesquisa:

McHenry Software

http://www.mchenrysoftware.com/

Engineering Dynamics

http://edccorp.com/

Society of Automotive Engineers - Special Publications

http://www.sae.org/PRODSERV/BOOKS/specpub.htm

Apêndice I. Cronograma de Atividades

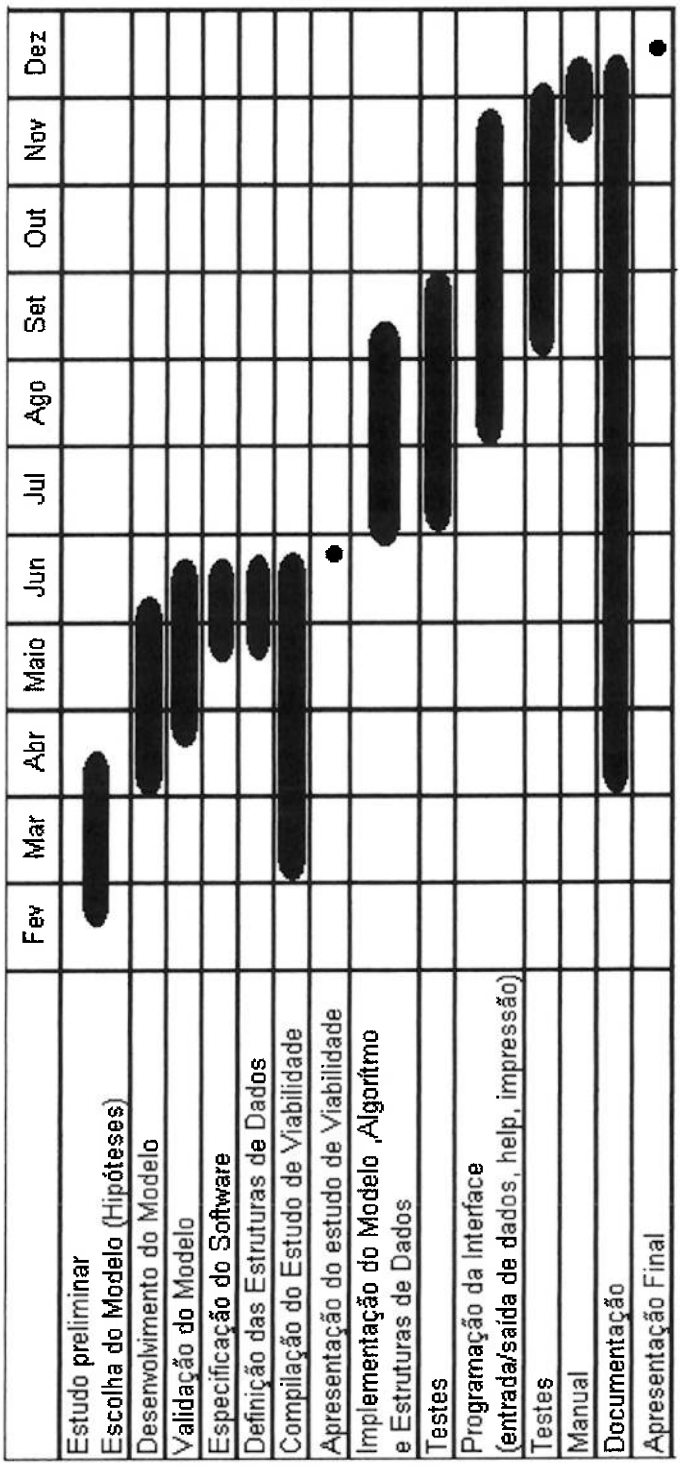


Figura A1.1 - Cronograma

Apêndice II. Testes do Modelo no MATLAB

1 - Introdução:

Uma vez que o modelo de colisão havia sido escolhido e detalhado, e antes de sua implementação definitiva no programa, houve uma etapa de testes intermediária, realizada utilizando o MatLab e o Simulink. Esta etapa de testes teve como objetivo analisar o modelo separadamente do programa final, sem interferências da interface de entrada e saída, do banco de dados, do algorítmo de integração utilizado. Dessa forma, foi possível implementar, observar e analisar os resultados do modelo antes de sua implementação definitiva. Também foi possível perceber algumas de sua deficiências que impediriam a implementação de algumas características inicialmente desejadas. Além disso, alguns problemas relacionados à precisão numérica foram detectados e solucionados, o que facilitou a implementação final. Em resumo, embora a implementação e testes do modelo no MatLab tenha sido trabalhosa e demorada, e não tenha acrescentado diretamente nada ao produto final, ela permitiu uma implementação final rápida e com menos problemas do modelo no programa.

Foram implementados dois modelos. O primeiro, mais detalhado, foi implementado utilizando o Simulink. Este modelo utiliza a formulação detalhada de atrito (levando em conta a influência de cada pneu, como visto em 3.2.2) e trata a colisão (tempo em que há força de colisão agindo sobre os carros) e a trajetória (tempo posterior à colisão, até a parada dos carros) separadamente, permitindo perceber peculiaridades de cada um desses momentos. Esse modelo será explicado inicialmente. Serão mostradas telas com a modelagem no Simulink, dos sistemas menos complexos aos mais complexos, para facilitar o entendimento. Também serão listados os "scripts" e funções utilizados.

O segundo modelo já é bem mais simples e parecido com o utilizado no programa. Não há uma separação clara entre os momentos de colisão e trajetória e é

utilizada a formulação simplificada de atrito. Esse modelo é composto somente por funções e scripts (sem modelagem no Simulink), que também serão listadas aqui.

Cada um dos testes listados na seção 3.6 também foi comparado com os resultados de ambas as modelos descritos acima, para testar não só a implementação do modelo no programa final, como também o algoritmo de integração e a interface de dados entre o modelo e o usuário.

Seguem abaixo os modelos detalhado e simplificado no Matlab, respectivamente.

2 - Modelo Detalhado (Simulink)

Como já foi mencionado, serão mostrados e explicados os blocos utilizados no Simulink na ordem de menor para maior complexidade, para facilitar o entendimento.

Bloco 1/módulo:

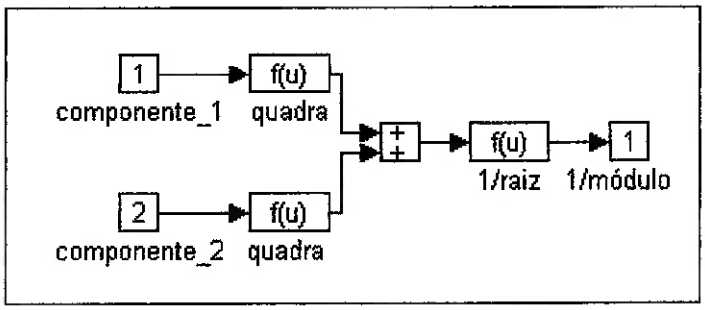


Figura A2.1 - Bloco 1/módulo

Este bloco simplesmente recebe as duas componentes de um vetor plano (entradas 1 e dois) e retorna o inverso do módulo deste vetor. A função quadra faz somente u(1)*u(1) e a função 1/ raiz faz 1/sqrt(u(1)). Este bloco é bastante utilizado adiante para calcular a divisão da força de atrito em x e y para cada pneu.

Blocos dos Pneus:

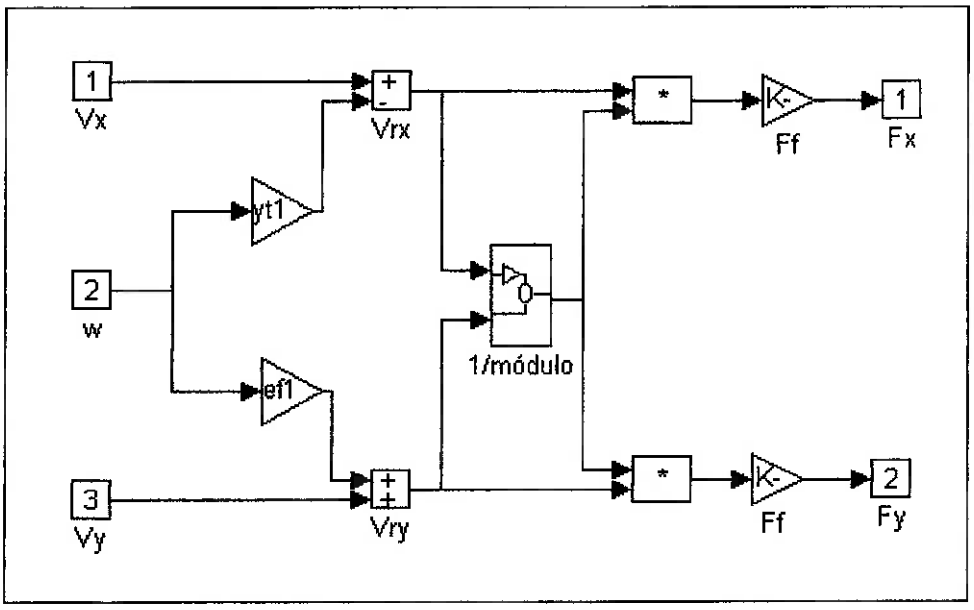


Figura A2.2 - Bloco Pneu 1

Este bloco calcula as componentes x e y (saídas 1 e 2, respectivamente) da força de atrito no pneu 1 (ver figura 9 no item 3.2.2) do veículo 1, com base nas velocidades x e y (entradas 1 e 3, respectivamente) do baricentro do veículo expressa na base solidária a ele (mostrada na figura 9) e na sua velocidade angular. Esse bloco nada mais faz que uma equação de Poisson para determinar a velocidade nos pneus, usando os braços e_F e y_T. O ganho Ff, para esse pneu, é dado por:

$$Ff = \frac{Mg\mu}{2.(e_F + e_R)} e_R$$

devido a ser um pneu dianteiro. Se fosse traseiro, seria usado Fr, dado por:

$$Fr = \frac{Mg\mu}{2.(e_F + e_R)} e_F$$

Esse bloco é repetido 4 vezes, uma para cada pneu, sendo mudados somente os braços (ver 3.2.2.1) e usando Ff ou Fr, conforme adequado. As figuras a seguir mostram os outros três pneus:

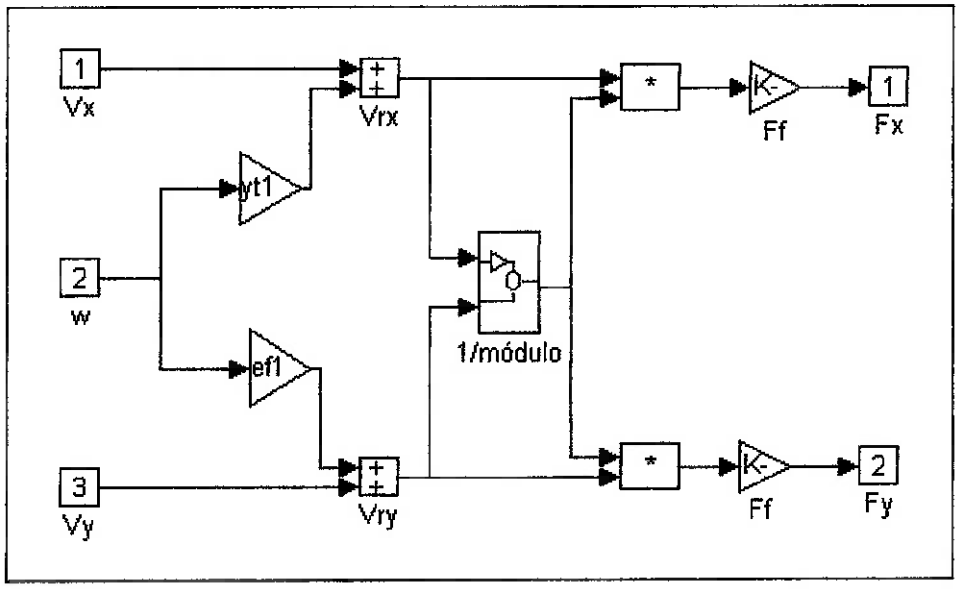


Figura A2.3 - Bloco Pneu 2

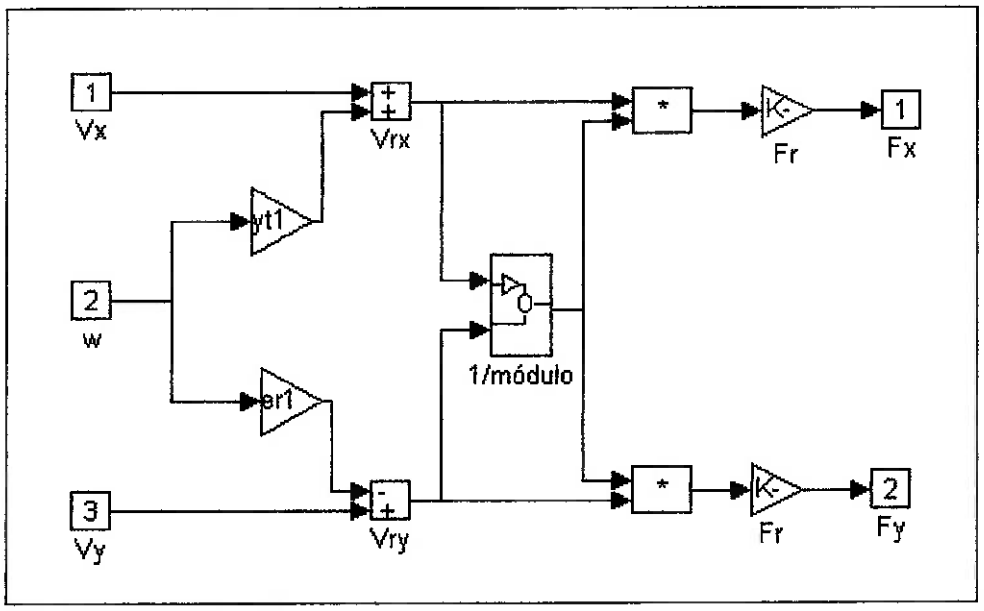


Figura A2.4 - Bloco Pneu 3

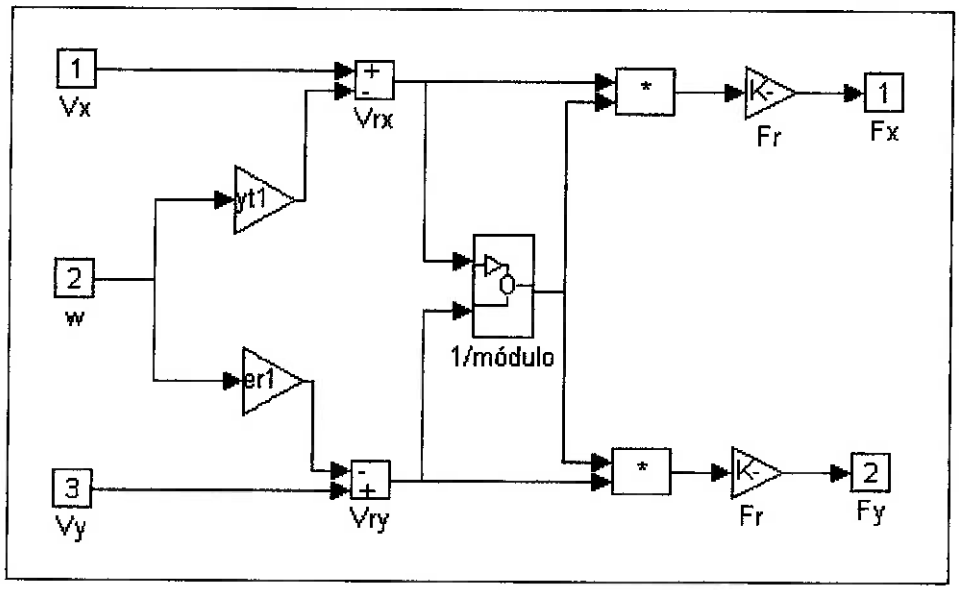


Figura A2.5 - Bloco Pneu 4

Bloco Frenagem:

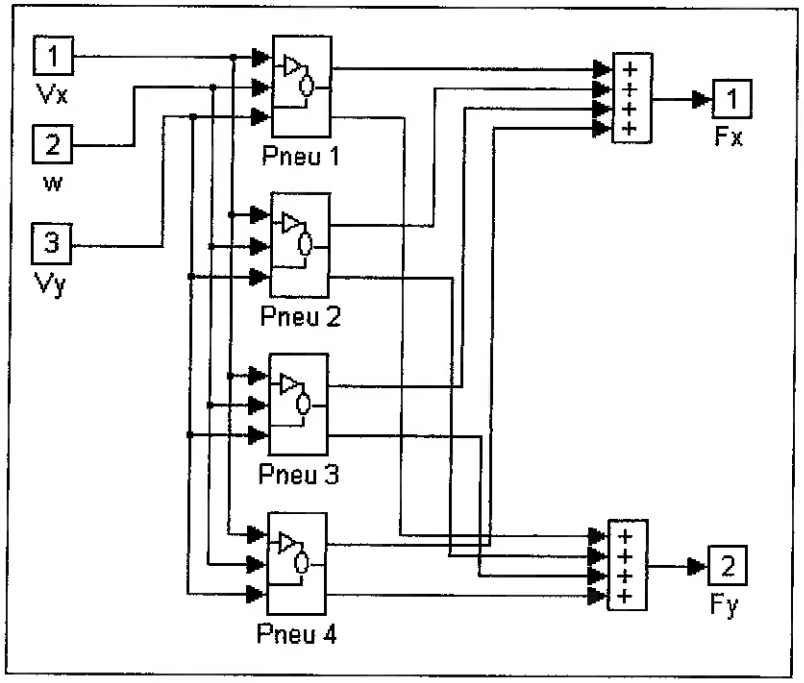


Figura A2.6 - Bloco Frenagem

Este bloco simplesmente calcula as forças x e y devidas a frenagem resultantes dos quatro pneus. As entradas são Vx, w e Vy e as saídas são Fx e Fy, como em cada bloco pneu.

Bloco Torque:

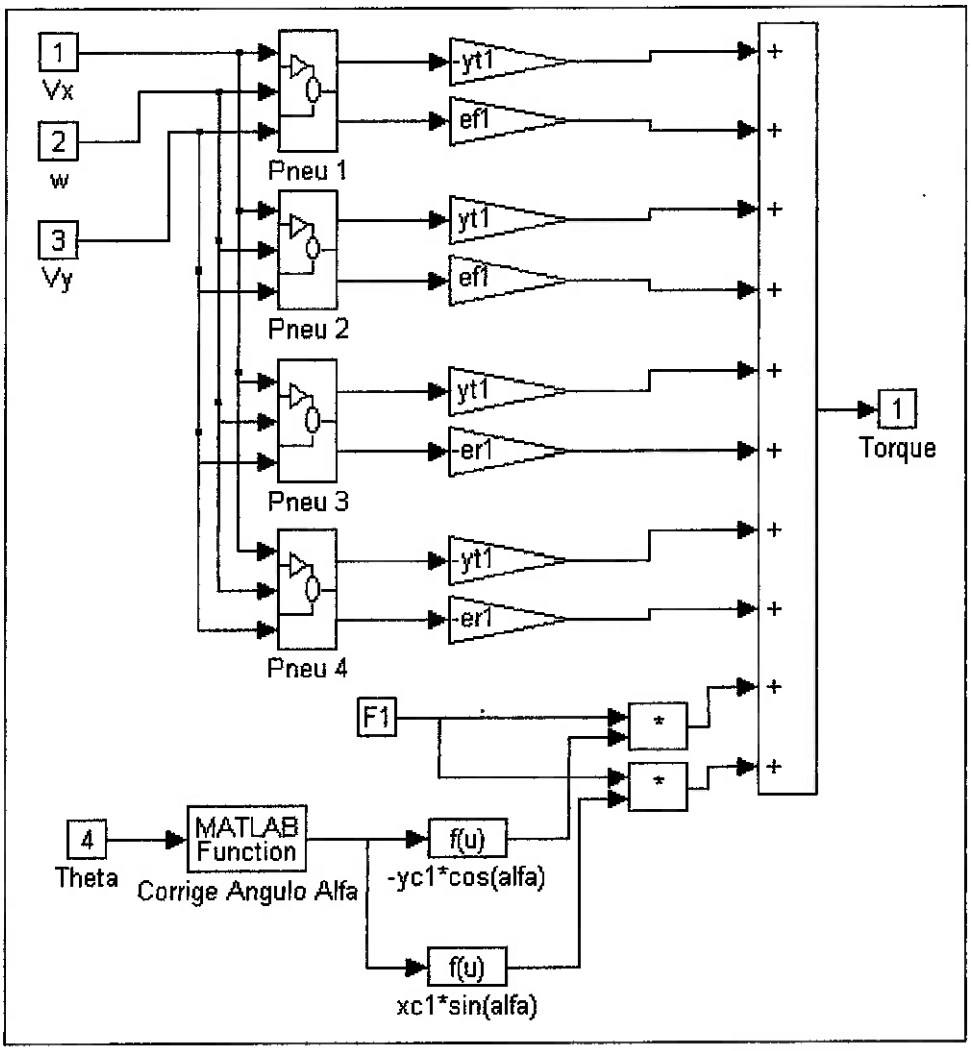


Figura A2.7 - Bloco Torque

Este bloco calcula qual o torque total agindo sobre o carro. Utilizando os blocos pneu e os braços e_R, e_F e y_T, calcula o torque total resultante da frenagem (ver 3.2.2.1). Utilizando o ângulo atual Theta, uma função que corrige o ângulo da força Alfa para adequá-lo a posição do carro (essa função será listada mais adiante) e a posição do centro de colisão (xc, yc) em relação ao baricentro do veículo, calcula também o torque

causado pela força de colisão F1. As entradas são Vx, w, Vy e Theta, e a saída é o Torque.

Bloco Forças na Base do Carro:

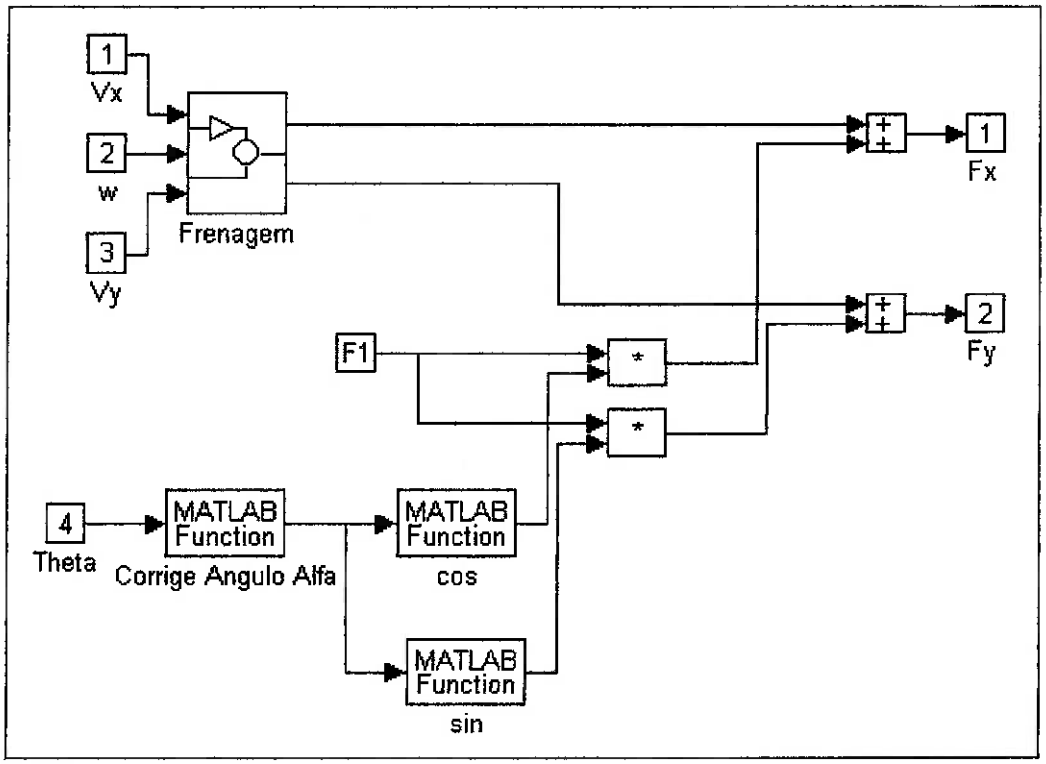


Figura A2.8 - Bloco Forças na Base do Carro

Esse bloco simplesmente soma as forças da frenagem com a força de colisão (decomposta conforme sua posição relativa com o carro). As entradas são como as do bloco anterior e as saídas são Fx e Fy, descritas na base solidária ao veículo.

Bloco Mudança de Base:

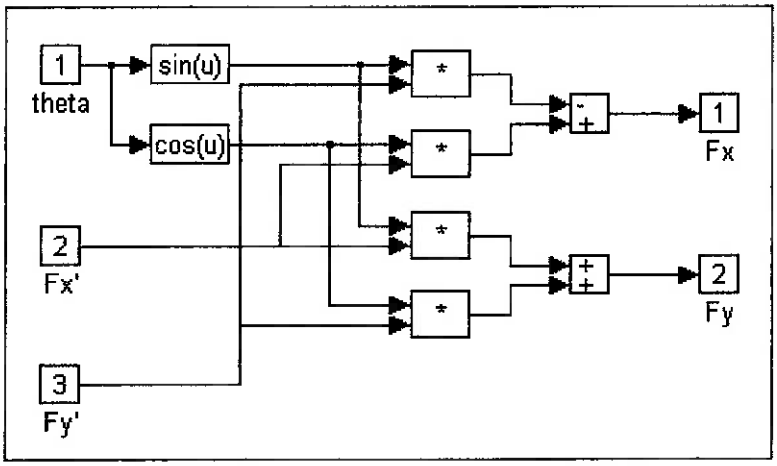


Figura A2.9 - Bloco Mudança de Base

Esse bloco recebe as força calculadas em Forças na Base do Carro Fx' e Fy' e passa essas forças para a base global, resultando Fx e Fy. Essa mudança de base nada mais é que uma rotação de -theta em torno da origem do sistema x'y'.

Bloco Carro 1:

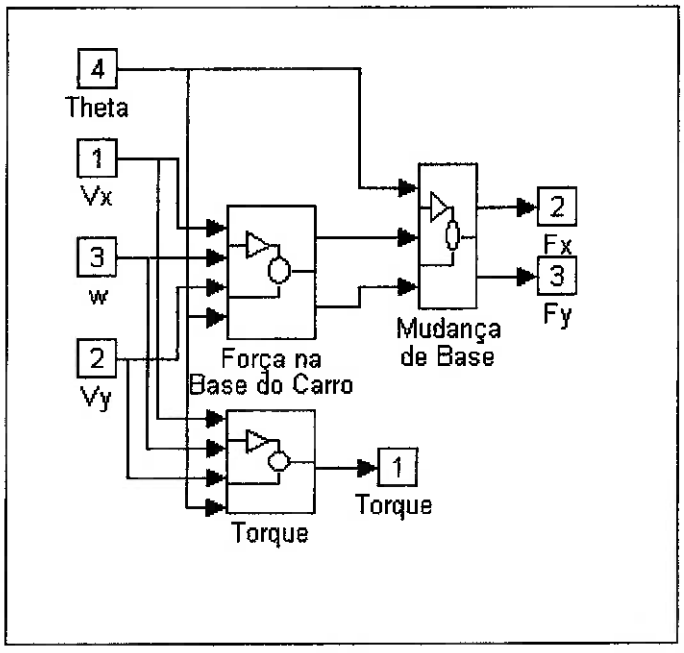


Figura A2.10 - Bloco Carro 1

Esse bloco simplesmente utiliza os três último blocos para determinar os esforços aplicados sobre o carro 1 descritos na base global.

Bloco Equações Diferenciais:

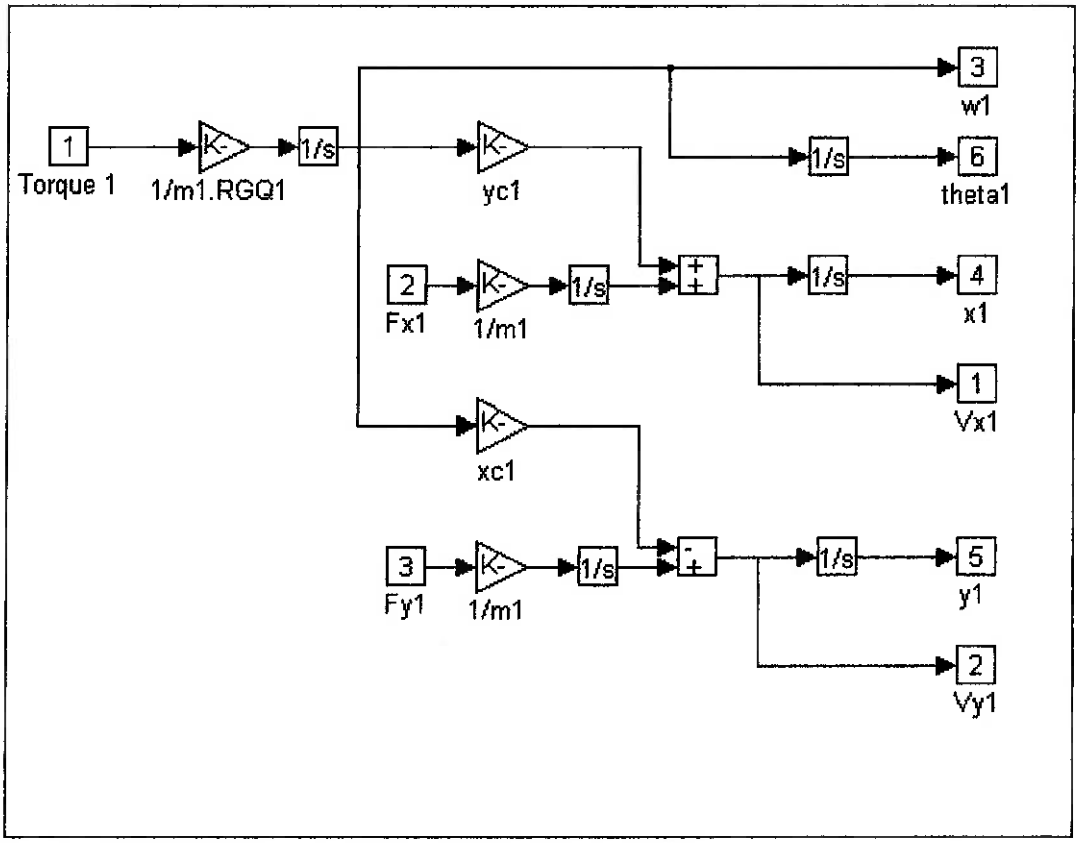


Figura A2.11 - Bloco Equações Diferenciais

Esse bloco recebe as saídas do bloco anterior, monta e integra as equações que descrevem o movimento do veículo (ver 3.2.3). Essa integração é feita usando os blocos 1/s e as condições iniciais X0, Y0, V0.cos(Theta0), V0.sin(Theta0) e Theta0. w0 é considerado sempre nulo. As condições iniciais são parâmetros dos blocos 1/s. As saídas são Vx, Vy, w, x, y e theta, respectivamente. Essas velocidades são a do ponto de contato entre os veículos e não as do baricentro, pois a velocidade desse ponto é de maior interesse na análise de colisão.

Bloco Mudança de Base (2):

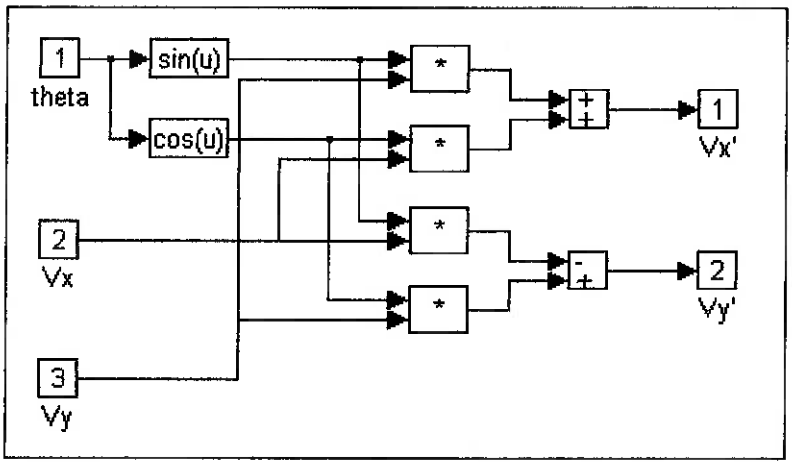


Figura A2.12 - Bloco Mudança de Base (2)

Embora com o mesmo nome do bloco utilizado em carro 1, esse bloco faz o oposto, leva um vetor da base global para a base solidária ao veículo, através de uma rotação de +theta. É utilizado para levar as velocidades obtidas da integração das equações de movimento (bloco anterior) para a base do carro, para ser usada no bloco carro 1.

Bloco Condição de Parada (Velocidades):

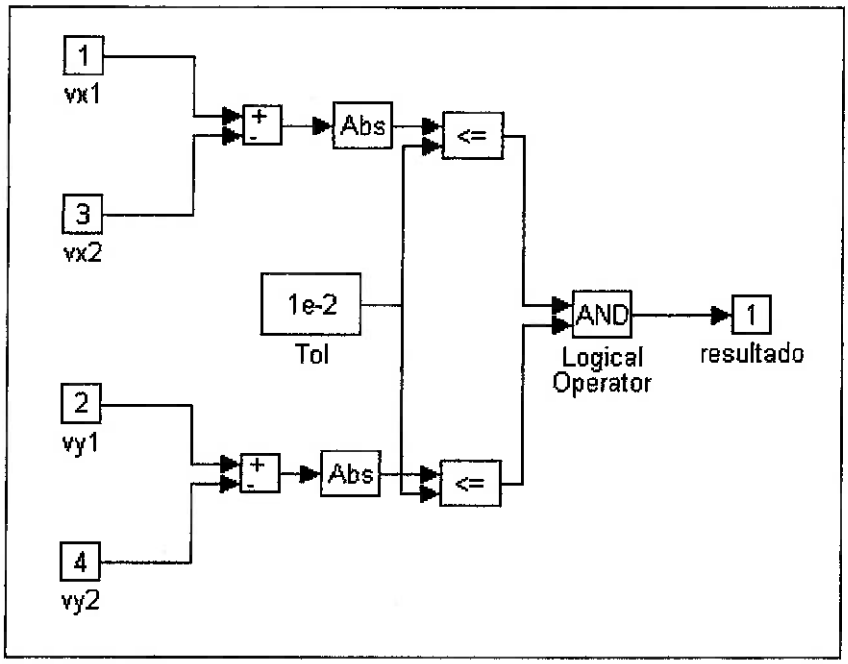


Figura A2.13 - Bloco Condição de Parada (Velocidades)

Esse bloco checa a cada passo de integração se a velocidade do ponto de contato em cada um dos carros atingiu um valor comum. Como foi explicada em 3.4.2 essa condição é suficiente para que não haja mais colisão. Deve-se observar neste bloco que a comparação é feita utilizando uma tolerância, para evitar os problemas discutidos em 3.4.4.

Bloco Condição de Parada (Posição):

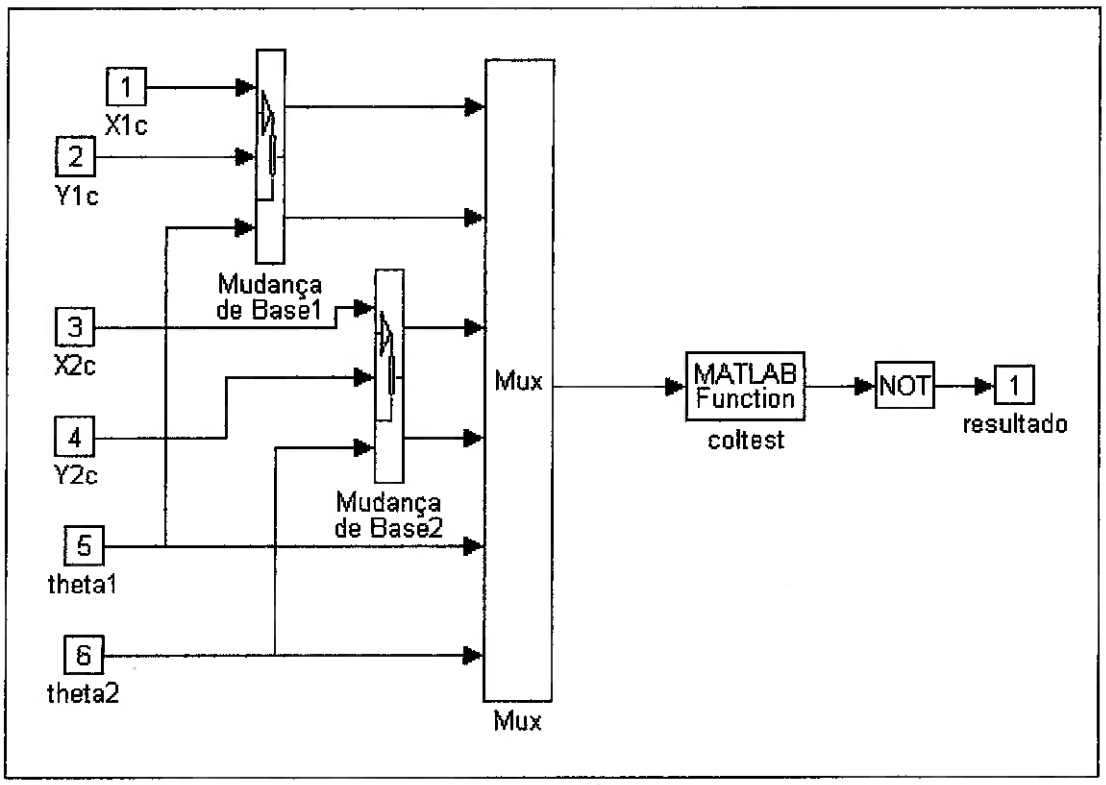


Figura A2.14 - Bloco Condição de Parada (Posição)

Este bloco recebe as posições (x,y,θ) de cada carro, faz a mudança de base necessária para descrever as velocidades do baricentro na base global e multiplexa esses valores, para que possam ser utilizados pela função coltest, que faz a detecção de contato entre os veículos, conforme explicado em 3.4.2. A versão para o MatLab dessa função pode ser encontrada mais adiante neste apêndice, enquanto a versão em C++ encontra-se no Apêndice III.

Bloco da Colisão:

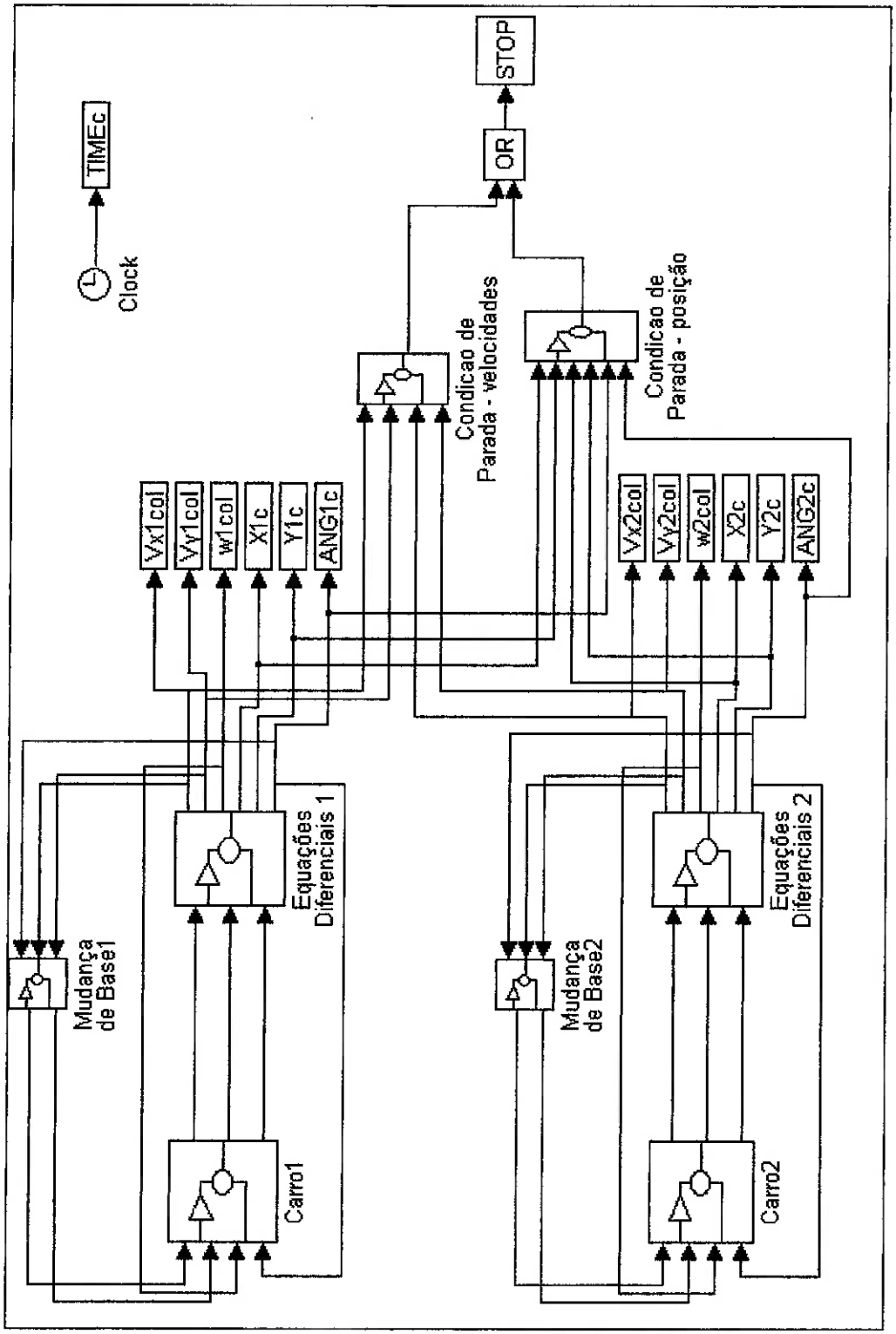


Figura A2.15 - Bloco da Colisão

Esse bloco utiliza todos os mencionados anteriormente para simular a colisão entre os dois veículos. O bloco carro 2, embora não tenha sido explicado, é igual ao carro 1, exceto que as variáveis (massas, braços, etc.) são as do carro 2. Isso faz com que todos os blocos mostrados, com excessão de 1/módulo, tenham que ser modificados, começando pelos pneus. Cabe aqui observar que embora o Simulink, com sua interface gráfica, seja um modo rápido e prático para teste de modelos, para sistemas mais complexos sua utilização torna-se bastante complicada. A descoberta de um erro nos blocos de pneus, por exemplo, fazia com que fosse necessário mudar os 32 pneus existentes nas análises de colisão e trajetória.

A análise é executada até que uma das condições de parada seja atingida, quando é executado o comando STOP do Simulink. As variáveis de posição, velocidade e tempo (medido com o bloco "Clock") são todas armazenadas em vetores do MatLab (usando os blocos "To Workspace") para serem analisadas posteriormente e também para serem utilizadas na análise de trajetória.

A análise de trajetória é muito parecida com a de colisão. Há somente duas grandes diferenças - a condição de parada e a inexistência de força de colisão durante a análise de trajetória. Dessa forma, de agora em diante serão mostrados os blocos da análise de trajetória, mas somente aqueles que são diferentes dos com mesmo nome usados na colisão.

Bloco Forças na Base do Carro:

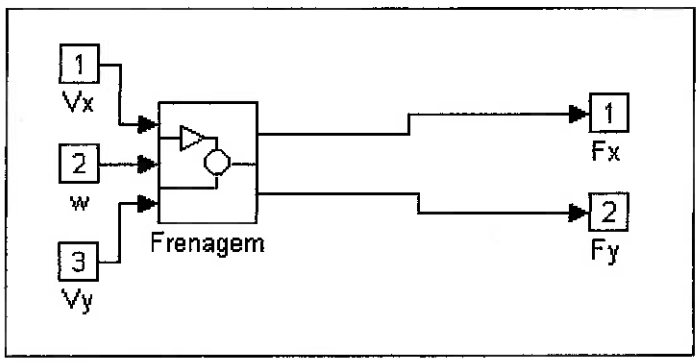


Figura A2.16 - Bloco Força na Base do Carro (trajetória)

Este bloco é igual ao da análise de colisão, exceto pela ausência da força de contato entre os veículos.

Bloco Torque:

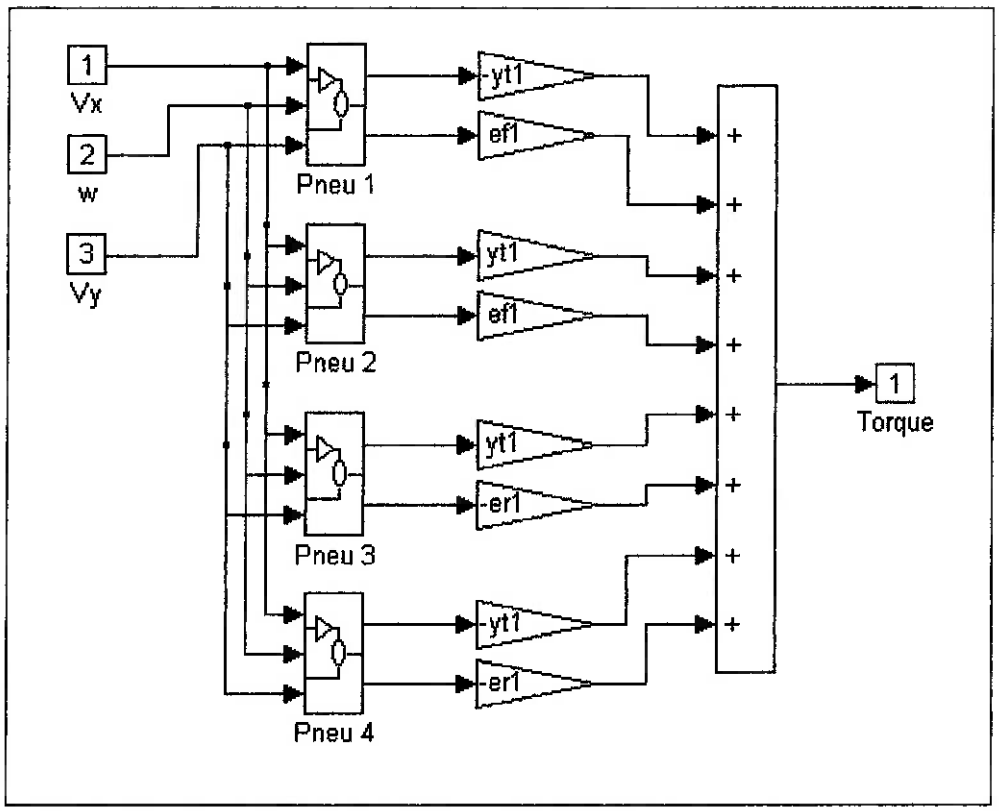


Figura A2.17 - Bloco Torque (trajetória)

Novamente, a única difereça entre este bloco e o da análise de colisão é a ausência da força de contato.

Bloco Equações Diferenciais:

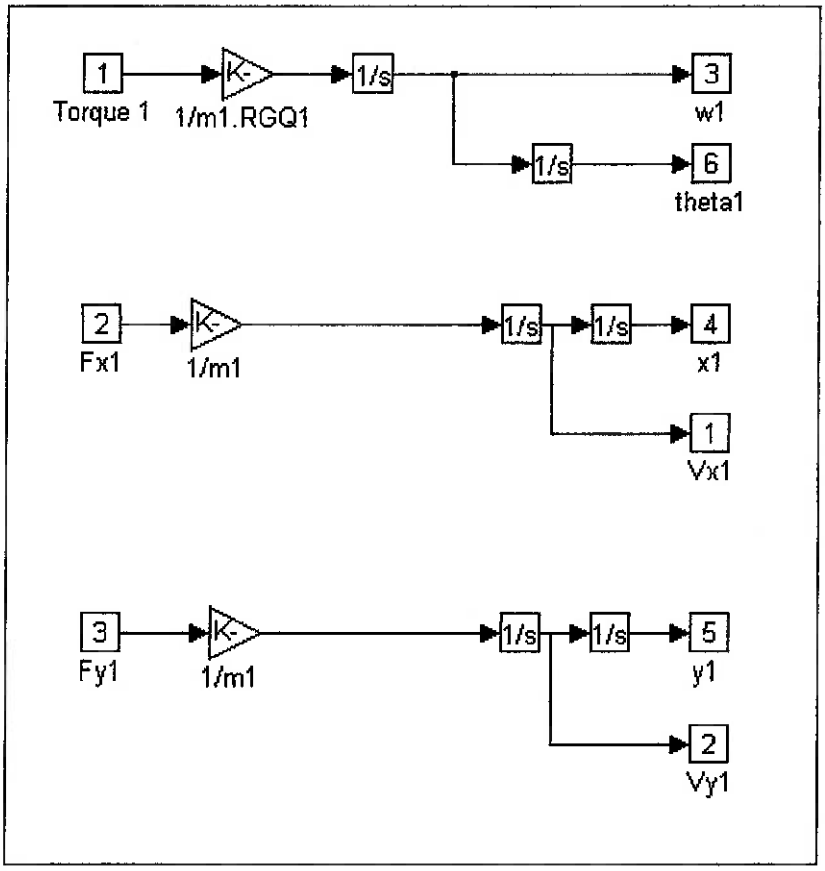


Figura A2.18 - Bloco Equações Diferenciais (trajetória)

Duas são as principais diferenças entre esse bloco e o da colisão. Uma é claramante visível - como as velocidades calculadas aqui são as do baricentro, e não do ponto de contato, elas não dependem diretamente de ω. Outra, invisível, são as condições iniciais dos integradores. Nesse caso, as condições iniciais dessa análise são iguais as finais da anterior, feitas as devidas correções (passa Vx do ponto de contato para o baricentro, por exemplo).

Bloco Condição de Parada (Trajetória)

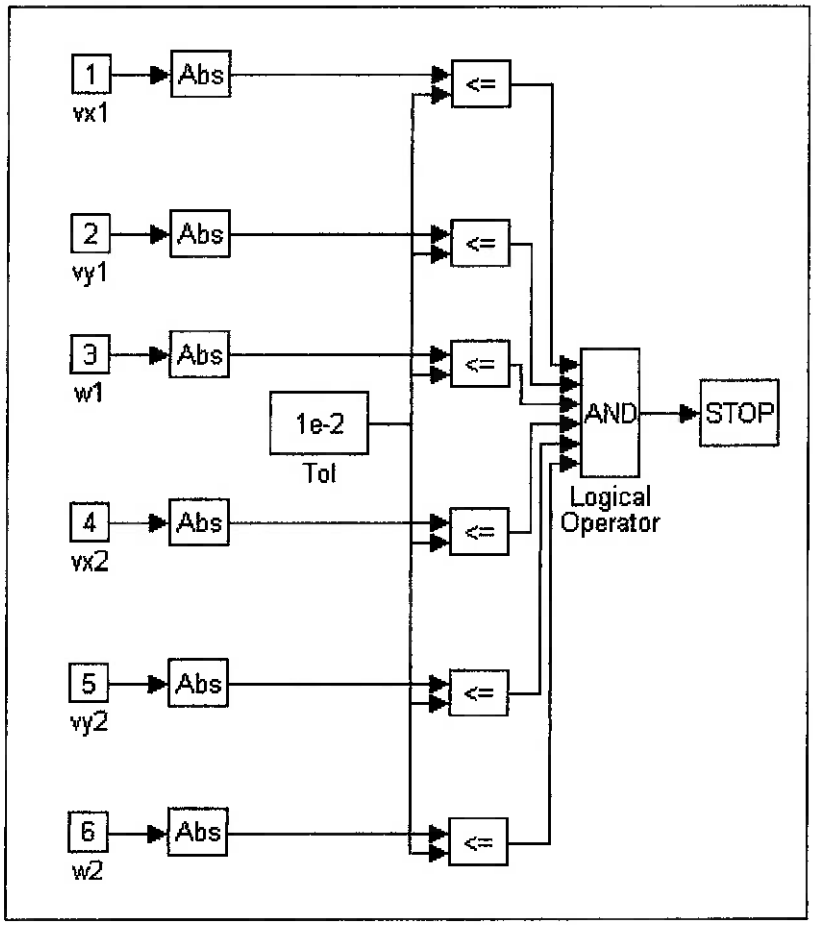


Figura A2.19 - Bloco Condição de Parada (trajetória)

Esse bloco simplesmente detecta quando os veículos param completamente (velocidades linear e angular). Mais uma vez é necessário usar uma tolerância, como visto em 3.4.4.

Bloco da Trajetória:

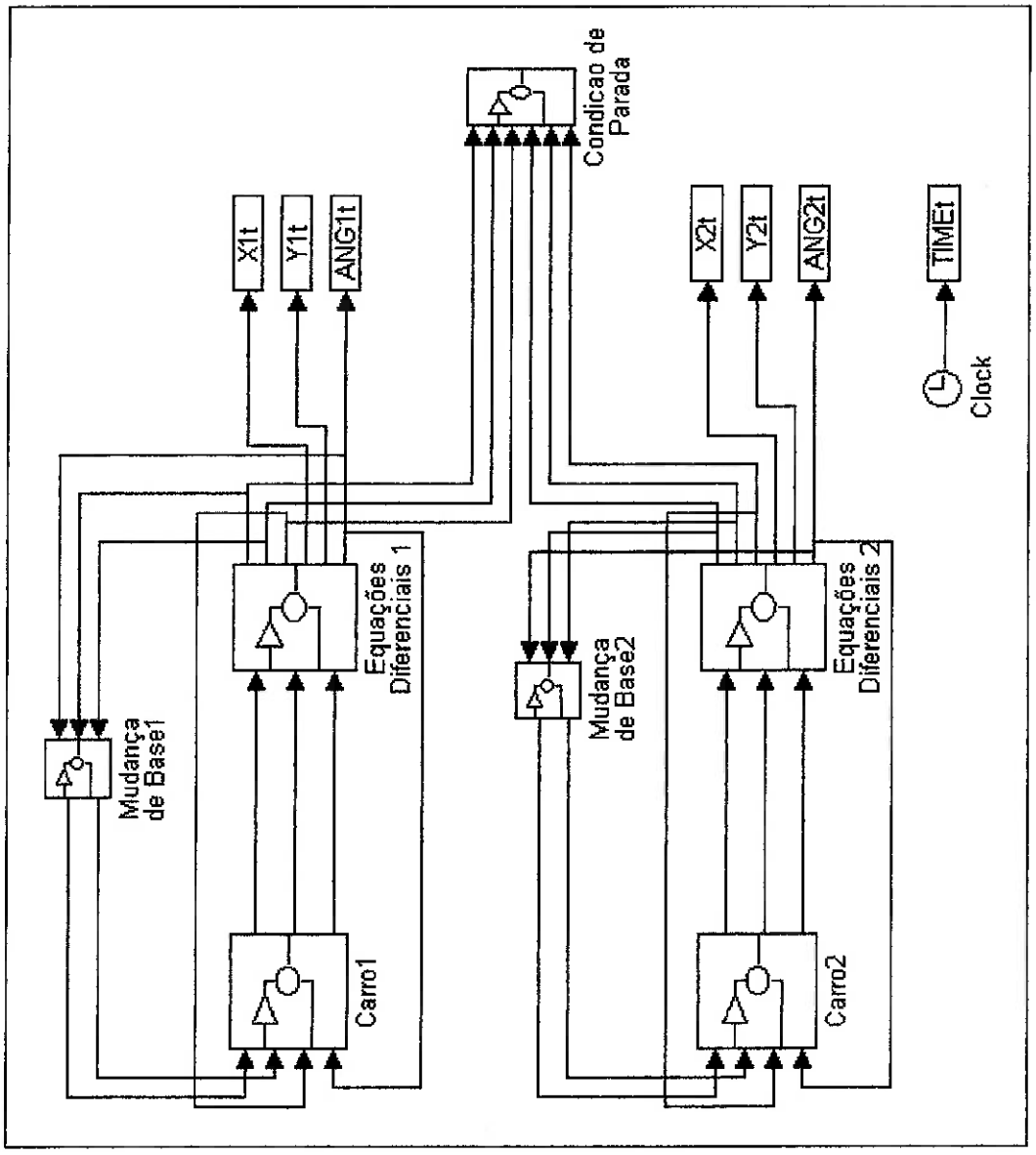


Figura A2.20 - Bloco da Trajetória

Novamente são usados os blocos anteriores para executar a análise de trajetória até que a condição de parada seja atingida. Dessa vez são registradas em vetores do MatLab somente as variáveis de posição dos veículos.

3 - Listagem das funções utilizadas (análise detalhada e simplificada)

Seguem as listagens das funções utilizadas em ambas as análises:

```
% Função crush
% Calcula a força de separação com base no perfil de deformações

function F = crush(A, B, perfil)

n = size(perfil, 1);
F = 0;
for i=2:n,
  F = F + (perfil(i, 2) + perfil(i-1, 2))*(perfil(i, 1) - perfil(i-1, 1));
end
F = A*(perfil(n,1)-perfil(1,1)) + B*F/2;


% Funcao crushcen
% calcula o baricentro da regiao de amassamento.
% p e uma matriz Nx2 contendo N pontos (X,Y) que definem o perfil

function [xg,yg] = crushcen(p)

N = size(p,1);
cr_area = 0;

% calcula a area por trapezio
for i = 1:(N-1),
  cr_area = cr_area + (p(i,2)+p(i+1,2))*(p(i+1,1)-p(i,1))/2;
end

% calcula o X do centro
xg1 = 0;
for i = 1:(N-1),
  xg1 = xg1 + (p(i,1)*p(i,2)+p(i+1,1)*p(i+1,2))*(p(i+1,1)-p(i,1))/2;
end
xg1 = xg1 / cr_area;

% calcula o Y do centro
yg1 = 0;
for i = 1:(N-1),
  yg1 = yg1 + (p(i,2)*p(i,2)+p(i+1,2)*p(i+1,2))*(p(i+1,1)-p(i,1))/2;
end
yg1 = yg1 / (2*cr_area);

xg = xg1;
yg = yg1;


% Funcao coltest
% verifica se ha contato entre os veiculos.
% recebe um vetor com as coordenadas multiplexadas

function [ret]=coltest(invec)

global xf1 xr1 ys1 xf2 xr2 ys2

% demultiplexa entradas
xg1 = invec(1);
yg1 = invec(2);
xg2 = invec(3);
yg2 = invec(4);
theta1 = invec(5);
theta2 = invec(6);

TolCol = 1e-4;

  % calcula os pontos do carro 1
```

```
st = sin(theta1);
ct = cos(theta1);
xf = xf1+TolCol;
xr = xr1+TolCol;
ys = ys1+TolCol;
x1(1) = xg1 + xf*ct - ys*st;
y1(1) = yg1 + xf*st + ys*ct;
x1(2) = xg1 - xr*ct + ys*st;
y1(2) = yg1 - xr*st - ys*ct;
x1(3) = xg1 + xf*ct + ys*st;
y1(3) = yg1 + xf*st - ys*ct;
x1(4) = xg1 - xr*ct - ys*st;
y1(4) = yg1 - xr*st + ys*ct;

% calcula as normais internas para o carro 1
nx1(1) = ct;
ny1(1) = st;
nx1(2) = -st;
ny1(2) = ct;
nx1(3) = -ct;
ny1(3) = -st;
nx1(4) = st;
ny1(4) = -ct;

% calcula os pontos do carro 2
st = sin(theta2);
ct = cos(theta2);
xf = xf2;
xr = xr2;
ys = ys2;
x2(1) = xg2 + xf*ct - ys*st;
y2(1) = yg2 + xf*st + ys*ct;
x2(2) = xg2 - xr*ct + ys*st;
y2(2) = yg2 - xr*st - ys*ct;
x2(3) = xg2 + xf*ct + ys*st;
y2(3) = yg2 + xf*st - ys*ct;
x2(4) = xg2 - xr*ct - ys*st;
y2(4) = yg2 - xr*st + ys*ct;

% calcula as normais internas para o carro 1
nx2(1) = ct;
ny2(1) = st;
nx2(2) = -st;
ny2(2) = ct;
nx2(3) = -ct;
ny2(3) = -st;
nx2(4) = st;
ny2(4) = -ct;

%% verifica colisao %

% carro 1 no carro 2
ret1 = 0;
for i=1:4,
  rr = 1;
  opx = x1(i) - x2(1);
  opy = y1(i) - y2(1);
  rr = rr & ((opx*nx2(3) + opy*ny2(3)) >= 0);
  rr = rr & ((opx*nx2(4) + opy*ny2(4)) >= 0);
  opx = x1(i) - x2(2);
  opy = y1(i) - y2(2);
  rr = rr & ((opx*nx2(1) + opy*ny2(1)) >= 0);
  rr = rr & ((opx*nx2(2) + opy*ny2(2)) >= 0);
  ret1 = ret1 | rr;
end

% carro 2 no carro 1
ret2 = 0;
for i=1:4,
  rr = 1;
  opx = x2(i) - x1(1);
```

```
    opy = y2(i) - y1(1);
    rr = rr & ((opx*nx1(3) + opy*ny1(3)) >= 0);
    rr = rr & ((opx*nx1(4) + opy*ny1(4)) >= 0);
    opx = x2(i) - x1(2);
    opy = y2(i) - y1(2);
    rr = rr & ((opx*nx1(1) + opy*ny1(1)) >= 0);
    rr = rr & ((opx*nx1(2) + opy*ny1(2)) >= 0);
    ret2 = ret2 | rr;
  end

  ret = ret1 | ret2;


% Funcao corang1
% corrige o angulo alfa com base na posicao do carro 1
% e do tipo de colisao.

function [ang] = corang1(teta)

global alfa tipo1

  ang = alfa - teta;

  % primeiro, converte ang para que fique entre 0 e 2*pi
  ang = rem(ang,2.*pi);
  if (ang<0)
    ang = 2.*pi+ang;
  end

  % agora, converte conforme o caso
  if (tipo1 == 'fr')
      if (ang<pi/2.)
        ang = ang + pi;
      elseif (ang>3.*pi/2.)
        ang = ang - pi;
      end
  end

  if (tipo1 == 'tr')
      if ((ang>pi/2.)&(ang<pi))
        ang = ang + pi;
      elseif ((ang>=pi)&(ang<3.*pi/2.))
        ang = ang - pi;
      end
  end


  if (tipo1 == 'le')
      if (ang<pi)
        ang = ang + pi;
      end
  end

  if (tipo1 == 'ld')
      if (ang>pi)
        ang = ang - pi;
      end
  end
```

A função corang2, usada para o carro dois, não será listada aqui, pois é igual a corang1, exceto por utilizar tipo2 ao invés de tipo1 para determinar o tipo de colisão.

```
% Script plotct
% plota resultados da colisao e trajetoria numa tela
% e corrige a escala para que seja proporcional

subplot(2, 1, 1)
plot(X1c(1), Y1c(1), 'y+', X2c(1), Y2c(1), 'b+')
```

```
hold
plot(X1c, Y1c, 'y', X2c, Y2c, 'b')
V=axis;
axis([min(V) max(V) min(V) max(V)])

subplot(2, 1, 2)
plot(X1t(1), Y1t(1), 'y*', X2t(1), Y2t(1), 'b*')
hold
plot(X1t, Y1t, 'y', X2t, Y2t, 'b')
V=axis;
axis([min(V) max(V) min(V) max(V)])
```

A partir deste ponto serão listadas os scripts que carregam as variáveis de entrada para os 4 testes realizados:

```
% Entradas do Algoritmo
% Batida de frente entre dois carros tipo 2

global alfa tipo1 tipo2 xf1 xr1 ys1 xf2 xr2 ys2 m1 m2 RGQ1 RGQ2 F mi g xc1 yc1
xc2 yc2 ef1 er1 yt1 ef2 er2 yt2

% Auto 1
ef1 = 1.176;
er1 = 1.273;
yt1 = 1.387/2.0;
m1 = 1383;
RGQ1 = 0.19;
A1 = 45359;
B1 = 296475;
xf1 = 2.116;
xr1 = 2.327;
ys1 = 0.853;

% Auto 2
ef2 = 1.176;
er2 = 1.273;
yt2 = 1.387/2.0;
m2 = 1383;
RGQ2 = 0.19;
A2 = 45359;
B2 = 296475;
xf2 = 2.116;
xr2 = 2.327;
ys2 = 0.853;

% Globais
mi = 0.7;
g = 9.81;
alfa = pi/4;

% Condições Iniciais 1
x10 = 1.7;
y10 = 1.7;
theta10 = 5*pi/4;
v10 = 8;

% Condições Iniciais 2
x20 = -1.2;
y20 = -1.2;
theta20 = pi/4;
v20 = 8;

% Colisão
perfil1 = [[-ys1 0];[-ys1 0.25];[0 0.25];[ys1 0.25];[ys1 0]];
F1 = crush(A1, B1, perfil1);
tipo1 = 'fr';
xc1 = xf1-0.125;
```

```
yc1 = 0;
perfil2 = [[-ys2 0];[-ys2 0.25];[0 0.25];[ys2 0.25];[ys2 0]];
F2 = crush(A2, B2, perfil2);
tipo2 = 'fr';
xc2 = xf2-0.125;
yc2 = 0;
F1 = (F1+F2)/2;
F2 = F1;
F = F1;


% Entradas do Algoritmo
% Batida da frente de um carro do tipo 1
% com a traseira de um tipo 4

global alfa tipo1 tipo2 xf1 xr1 ys1 xf2 xr2 ys2 m1 m2 RGQ1 RGQ2 F mi g xc1 yc1
xc2 yc2 ef1 er1 yt1 ef2 er2 yt2

% Auto 1
ef1 = 1.146;
er1 = 1.222;
yt1 = 1.298/2.0;
m1 = 998;
RGQ1 = 0.129;
A1 = 52889;
B1 = 324054;
xf1 = 1.93;
xr1 = 2.129;
ys1 = 0.772;

% Auto 2
ef2 = 1.389;
er2 = 1.504;
yt2 = 1.570/2.0;
m2 = 1923;
RGQ2 = 0.241;
A2 = 62521;
B2 = 89632;
xf2 = 2.510;
xr2 = 2.896;
ys2 = 0.978;

% Globais
mi = 0.7;
g = 9.81;
alfa = 0.0;

% Condições Iniciais 1
x10 = xf1;
y10 = 0;
theta10 = pi;
v10 = 12;

% Condições Iniciais 2
x20 = -xr2;
y20 = 0;
theta20 = pi;
v20 = 0;

% Colisão
perfil1 = [[-ys1 0];[-ys1 0.25];[0 0.25];[ys1 0.25];[ys1 0]];
F1 = crush(A1, B1, perfil1);
tipo1 = 'fr';
xc1 = xf1-0.125;
yc1 = 0;
perfil2 = [[-ys2 0];[-ys2 0.05];[0 0.05];[ys2 0.05];[ys2 0]];
F2 = crush(A2, B2, perfil2);
tipo2 = 'tr';
xc2 = -xr2+0.025;
yc2 = 0;
F1 = (F1+F2)/2;
```

```
F2 = F1;
F = F1;


% Entradas do Algoritmo
% Batida da frente de um carro do tipo 1
% com a lateral de um tipo 4

global alfa tipo1 tipo2 xf1 xr1 ys1 xf2 xr2 ys2 m1 m2 RGQ1 RGQ2 F mi g xc1 yc1
xc2 yc2 ef1 er1 yt1 ef2 er2 yt2

% Auto 1
ef1 = 1.146;
er1 = 1.222;
yt1 = 1.298/2.0;
m1 = 998;
RGQ1 = 0.129;
A1 = 52889;
B1 = 324054;
xf1 = 1.93;
xr1 = 2.129;
ys1 = 0.772;

% Auto 2
ef2 = 1.389;
er2 = 1.504;
yt2 = 1.570/2.0;
m2 = 1923;
RGQ2 = 0.241;
A2 = 62521;
B2 = 89632;
xf2 = 2.510;
xr2 = 2.896;
ys2 = 0.978;

% Globais
mi = 0.7;
g = 9.81;
alfa = 0.0;

% Condições Iniciais 1
x10 = xf1;
y10 = 0;
theta10 = pi;
v10 = 12;

% Condições Iniciais 2
x20 = -ys2;
y20 = 0;
theta20 = pi/2;
v20 = 0;

% Colisão
perfil1 = [[-ys1 0];[-ys1 0.35];[0 0.35];[ys1 0.35];[ys1 0]];
F1 = crush(A1, B1, perfil1);
tipo1 = 'fr';
xc1 = xf1-0.175;
yc1 = 0;
perfil2 = [[-xr2*3/4 0];[-ys1 0.03];[0 0.03];[ys1 0.03];[xr2*3/4 0]];
F2 = crush(A2, B2, perfil2);
tipo2 = 'ld';
xc2 = 0;
yc2 = -ys2+0.015;
F1 = (F1+F2)/2;
F2 = F1;
F = F1;


% Entradas do Algoritmo
% Batida da frente de um carro do tipo 1
% com a lateral de um tipo 4,
```

```
% fora de centro

global alfa tipo1 tipo2 xf1 xr1 ys1 xf2 xr2 ys2 m1 m2 RGQ1 RGQ2 F mi g xc1 yc1
xc2 yc2 ef1 er1 yt1 ef2 er2 yt2

% Auto 1
ef1 = 1.146;
er1 = 1.222;
yt1 = 1.298/2.0;
m1 = 998;
RGQ1 = 0.129;
A1 = 52889;
B1 = 324054;
xf1 = 1.93;
xr1 = 2.129;
ys1 = 0.772;

% Auto 2
ef2 = 1.389;
er2 = 1.504;
yt2 = 1.570/2.0;
m2 = 1923;
RGQ2 = 0.241;
A2 = 62521;
B2 = 89632;
xf2 = 2.510;
xr2 = 2.896;
ys2 = 0.978;

% Globais
mi = 0.7;
g = 9.81;
alfa = pi/6.0;

% Condições Iniciais 1
x10 = xf1+ys2-0.15;
y10 = xf2/4.0;
theta10 = pi;
v10 = 12;

% Condições Iniciais 2
x20 = 0;
y20 = 0;
theta20 = pi/2;
v20 = 8;

% Colisão
perfil1 = [[-ys1 0];[0 0.15];[ys1 0.30]];
F1 = crush(A1, B1, perfil1);
tipo1 = 'fr';
xc1 = xf1-0.1;
yc1 = -ys1/3;
perfil2 = [[0 0];[xf2/2 0.1];[xf2 0.2]];
F2 = crush(A2, B2, perfil2);
tipo2 = 'ld';
xc2 = 2*xf2/3;
yc2 = -ys2+(0.2/3);
if (abs(F1-F2)/F1 > 0.2)
  'As forcas divergem por mais de 20%'
end
F1 = (F1+F2)/2;
F2 = F1;
F = F1;
```

Por fim serão listadas as funções usadas somente na análise simplificada:

```
% Resolve.m - executa ODE45 convenientemente

start = [v10*cos(theta10) v10*sin(theta10) x10 y10 0 theta10 v20*cos(theta20)
v20*sin(theta20) x20 y20 0 theta20];
```

```
[T, Y] = ode45('eqdifs',0,5,start,1e-4);

plot(Y(:,3),Y(:,4),'r+',Y(:,9),Y(:,10),'b+')


% EqDifs.m - implementacao, no MATLAB, da funcao EqDifs do programa
% representa o sistema de equacoes diferenciais do problema

% ordem dos parametros y
% y(1) = vx1
% y(2) = vy1
% y(3) = x1
% y(4) = y1
% y(5) = w1
% y(6) = theta1
% y(7) = vx2
% y(8) = vy2
% y(9) = x2
% y(10) = y2
% y(11) = w2
% y(12) = theta2

function yp = eqdifs(t,y)

global alfa tipo1 tipo2 xf1 xr1 ys1 xf2 xr2 ys2 m1 m2 RGQ1 RGQ2 F mi g xc1 yc1
xc2 yc2 ef1 er1 yt1 ef2 er2 yt2

TolVel = 1e-2;

Izz1 = m1*RGQ1;
Izz2 = m2*RGQ2;

sint1 = sin(y(6));
cost1 = cos(y(6));
sint2 = sin(y(12));
cost2 = cos(y(12));

% verifica se ainda deve haver forca de colisao
if (F ~= 0)
  bContato = coltest([y(3) y(4) y(9) y(10) y(6) y(12)]);
  bColisao = veltest([y(1) y(2) y(7) y(8)]);

  if (~bContato | ~bColisao)
    F = 0;
  end
end

Fx1 = 0;
Fy1 = 0;
Fx2 = 0;
Fy2 = 0;
Tz1 = 0;
Tz2 = 0;

% forca devido a colisao
if (F ~= 0)
  % carro 1
  tempalfa = corang1(y(6));
  tempalfa = tempalfa + y(6);
  Fx1 = F*cos(tempalfa);
  Fy1 = F*sin(tempalfa);
  Tz1 = -Fx1*(xc1*sint1 + yc1*cost1) + Fy1*(xc1*cost1 - yc1*sint1);
  % carro 2
  tempalfa = corang2(y(12));
  tempalfa = tempalfa + y(12);
  Fx2 = F*cos(tempalfa);
  Fy2 = F*sin(tempalfa);
  Tz2 = -Fx2*(xc2*sint2 + yc2*cost2) + Fy2*(xc2*cost2 - yc2*sint2);
end
```

```
% forca devido a frenagem
% carro 1
modv = sqrt(y(1)*y(1) + y(2)*y(2));
if (abs(y(1)) > TolVel)
  Fx1 = Fx1 - g*m1*y(1)/modv;
end
if (abs(y(2)) > TolVel)
  Fy1 = Fy1 - g*m1*y(2)/modv;
end
% carro 2
modv = sqrt(y(7)*y(7) + y(8)*y(8));
if (abs(y(7)) > TolVel)
  Fx2 = Fx2 - g*m2*y(7)/modv;
end
if (abs(y(8)) > TolVel)
  Fy2 = Fy2 - g*m2*y(8)/modv;
end

% torque devido a frenagem
% carro 1
if (abs(y(5)) > TolVel)
  Taux = -g*m1*mi*(ef1+er1)*y(5)/abs(y(5));
  if ((abs(Taux)>abs(Tz1)) & (F ~= 0))
    Tz1 = 0;
  else
    Tz1 = Tz1 + Taux;
  end
end
% carro 2
if (abs(y(11)) > TolVel)
  Taux = -g*m2*mi*(ef2+er2)*y(11)/abs(y(11));
  if ((abs(Taux)>abs(Tz2)) & (F ~= 0))
    Tz2 = 0;
  else
    Tz2 = Tz2 + Taux;
  end
end

% transfere os valores para as saidas

yp(1) = Fx1/m1;
yp(2) = Fy1/m1;
yp(3) = y(1);
yp(4) = y(2);
yp(5) = Tz1/Izz1;
yp(6) = y(5);
yp(7) = Fx2/m2;
yp(8) = Fy2/m2;
yp(9) = y(7);
yp(10) = y(8);
yp(11) = Tz2/Izz2;
yp(12) = y(11);

% Funcao veltest
% criterio de parada da velocidade

function ret = veltest(invec)

vx1 = invec(1);
vy1 = invec(2);
vx2 = invec(3);
vy2 = invec(4);

TolVel = 1e-2;

b1 = (abs(vx1 - vx2) > TolVel);
b2 = (abs(vy1 - vy2) > TolVel);

ret = (b1 | b2);
```

Apêndice III. Fontes do Programa

Neste apêndice encontram-se as listagens dos principais arquivos utilizados neste projeto. Não estão listados aqui todos os arquivos necessários para a compilação do executável. Faltam o aqruivo de projeto (FACT.MAK), de definições do programa (FACT.DEF), os recursos gráficos (FACT.RC e RESOURCE.H) e os arquivos de Help (AFXCORE.RTF, AFXPRINT.RTF, FACT.HPJ, FACT.HM). Todos estes arquivos podem ser encontrados no disquete anexo.

Dentre os arquivos aqui listados, os mais importantes são:

- factdoc.h e factdoc.cpp - contém o documento, classe do programa responsável pela armazenagem, tratatamento e serialização das variáveis da análise. Esta classe pode ser considerada a mais importante do programa, pois contém todos os algoritmos responsáveis pela solução do problema. Por isso, é a que está mais extensivamente documentada.
- factview.h e factview.cpp - contém a classe de gerenciamento da interface.
- autodlg.h, autodlg.cpp, crushdlg,h, crushdlg.cpp, inipdlg.h, inipdlg.cpp, trajdlg.h, trajdlg.cpp - classes de implementação dos tabs auto1 e auto2, deformação 1 e deformação 2, cenário e trajetória, respectivamente. Destaca-se o tratamento dos gráficos feito em trajdlg.
- cardata.h, cardata.cpp, crush.h, crush.cpp, trajeto.h, trajeto.cpp - arquivos de implementação das classes discutidas na seção 3.3.

Seguem abaixo as listagens:

```
// factdoc.h : interface of the FACTDoc class
//
/////////////////////////////////////////////////////////////////////////////

class FACTDoc : public CDocument
{
// Classes utilizadas
class TrajPoint;
class CarDatabase;
class CarData;
class CrushArray;
class Trajetoria;

// Variaveis
protected:
  // Dados
  CarData* car1;                  // Carro 1
  CarData* car2;                  // Carro 2
  CrushArray* pCrushProf1;        // Deformacao 1
  CrushArray* pCrushProf2;        // Deformacao 2
```

```
  UINT Tipo1, Tipo2;              // Tipos de Colisao
  double Ocx1, Ocx2, Ocy1, Ocy2;  // Origens dos Ammassamentos
  // Cenario
  double X01, Y01, X02, Y02;      // Coordenadas Iniciais dos Baricentros
  double Theta01, Theta02;        // Angulos Iniciais (rad)
  double Theta01Deg, Theta02Deg;  // Angulos Iniciais (graus)
  double V01, V02;                // Velocidades Iniciais (direcao dada por Theta)
  double Alfa;                    // Angulo da Forca de Colisao (rad)
  double AlfaDeg;                 // Angulo da Forca de Colisao (graus)
  double Mi;                      // Coeficiente de Atrito Pneu-Solo
  // Banco de Dados de Veiculos
  CarDatabase* pCDB;
  // Parametros do Algoritmo de Solucao
  double PassoMax;                // Passo Maximo
  double PassoMin;                // Passo Minimo
  double TolPos;                  // Tolerancia para Posicao
  double TolVel;                  // Tolerancia para velocidade
  double TolCol;                  // Tolerancia para checagem de colisao
  // Deformacao (parametros calculados)
  double F;                       // Forca
  double xc1, yc1, xc2, yc2;      // Pontos de Aplicacao em cada carro
  double alfa1, alfa2;            // Angulo da forca para cada carro
  double TEndCol;                 // Tempo do fim da colisao
  // Solucao
  BOOL ExecFlag;                  // Valida a Solucao
  Trajetoria* pTrajeto1;          // Carro 1
  Trajetoria* pTrajeto2;          // Carro 2

// Construcao/Destruicao (protected pois so e construido na serializacao)
  FACTDoc();
  ~FACTDoc();
  DECLARE_DYNCREATE(FACTDoc)

// Serializacao
public:
  virtual void Serialize(CArchive& ar);

// Mapa de Mensagens
protected:
  //{{AFX_MSG(FACTDoc)
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

// Diagnostico
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

// Comandos
  virtual BOOL OnNewDocument();
  virtual BOOL OnOpenDocument(const char*);
  virtual void OnCloseDocument();
public:                  // Chamados pela View
  void SaveDatabase();
  void OnFileSaveAs();
  void OnFileSave();
  class StatusDlg;

// Algoritmo
  BOOL TrajetoRK4(StatusDlg* pS);  // Public porque dispara a solucao
protected:
  BOOL Consis();
  void PassoRK(CarData*, double, double, double*, double*);
  void EqDifs(CarData*, double, double*, double*);
  BOOL Parada(double, double, double, double, double, double, double*);
  BOOL Colisao(double, double, double, double, double, double, double, double, double*);
  BOOL Contato(double, double, double, double, double, double);
  double Deforma(CarData*, CrushArray*, double, double, double*, double*, UINT);
  double CorrigeAngulo(UINT, double);
// Matematicas
  double _sin(double);
  double _cos(double);

// Auxiliar - interpretacao de mensagens do Windows
BOOL DoEvents();

// Funcoes de acesso
public:
  // Variaveis
```

```
  void SetCar1(BOOL, CarData*);
  void GetCar1(CarData*);
  void SetCar2(BOOL, CarData*);
  void GetCar2(CarData*);
  void SetCrush1(BOOL, CrushArray*);
  void GetCrush1(CrushArray*);
  void SetCrush2(BOOL, CrushArray*);
  void GetCrush2(CrushArray*);
  void SetTipo1(UINT);
  UINT GetTipo1() const;
  void SetTipo2(UINT);
  UINT GetTipo2() const;
  void SetOcx1(double);
  double GetOcx1() const;
  void SetOcx2(double);
  double GetOcx2() const;
  void SetOcy1(double);
  double GetOcy1() const;
  void SetOcy2(double);
  double GetOcy2() const;
  void SetX01(double);
  double GetX01() const;
  void SetX02(double);
  double GetX02() const;
  void SetY01(double);
  double GetY01() const;
  void SetY02(double);
  double GetY02() const;
  void SetTheta01(double);
  double GetTheta01() const;
  void SetTheta02(double);
  double GetTheta02() const;
  void SetV01(double);
  double GetV01() const;
  void SetV02(double);
  double GetV02() const;
  void SetAlfa(double);
  double GetAlfa() const;
  void SetMi(double);
  double GetMi() const;
  // Banco de dados de veiculos
  CarDatabase* GetCDB();
  // Solucao
  BOOL GetExecFlag();
  double GetTEndCol() const;  // Tempo do fim da colisao
  double GetLastT();          // Tempo do fim da simulacao
  void GetMaxRect(double*, double*, double*, double*);  // coordenadas do retangulo
                                                        // que contem todos os pontos da
                                                        // trajetoria
  void GetMinMaxTraj(TrajPoint*, double*, TrajPoint*, double*, short);
  BOOL ResultadoEmT(double, short, TrajPoint*);
};

/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// factdoc.cpp : implementation of the FACTDoc class
//

#include "stdafx.h"
#include "fact.h"
#include "cardata.h"
#include "crush.h"
#include "trajeto.h"
#include "status.h"
#include "factdoc.h"
#include <math.h>

// Definicao de Constantes
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
#define RK_NUM_EQ 6
#define _GRAV_ 9.81
#ifndef DEGTORAD
  #define DEGTORAD    0.01745329
  #define RADTODEG   57.29578000
  #define PI          3.14159265358979
#endif  //DEGTORAD
```

```
////////////////////////////////////////////////////////////////////////////
// FACTDoc construction/destruction

IMPLEMENT_DYNCREATE(FACTDoc, CDocument)

/*************************************************************************
/ FactDoc()                                                              /
/ Construtor da Classe FactDoc - Cria e inicializa as variaveis membro   /
/ Parametros: -                                                          /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
FACTDoc::FACTDoc()
{
  car1 = new CarData;
  car2 = new CarData;

  pCrushProf1 = new CrushArray;
  pCrushProf2 = new CrushArray;
  Tipo1 = 0;
  Tipo2 = 0;
  Ocx1 = 0.;
  Ocy1 = 0.;
  Ocx2 = 0.;
  Ocy2 = 0.;

  X01 = 0.;
  X02 = 0.;
  Y01 = 0.;
  Y02 = 0.;
  Theta01 = 0.;
  Theta02 = 0.;
  Theta01Deg = 0.;
  Theta02Deg = 0.;
  V01 = 0.;
  V02 = 0.;
  Alfa = 0.;
  AlfaDeg = 0.;
  Mi = 0.7;

  PassoMax = 1e-4;
  PassoMin = 1e-6;
  TolPos = 1e-4;
  TolVel = 1e-2;
  TolCol = 1e-4;

  F = 0;
  xc1 = 0;
  yc1 = 0;
  xc2 = 0;
  yc2 = 0;

  pCDB = new CarDatabase;

  ExecFlag = FALSE;
  pTrajeto1 = new Trajetoria;
  pTrajeto2 = new Trajetoria;
}

/*************************************************************************
/ ~FactDoc()                                                             /
/ Destrutor da Classe FactDoc - Libera memoria ocupada pelas variaveis   /
/   membro                                                               /
/ Parametros: -                                                          /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
FACTDoc::~FACTDoc()
{
  delete pCDB;
  delete car1;
  delete car2;
  delete pCrushProf1;
  delete pCrushProf2;
  delete pTrajeto1;
  delete pTrajeto2;
}

////////////////////////////////////////////////////////////////////////////
// FACTDoc serialization
```

```
/*************************************************************************
/ Serialize()                                                            /
/ Serializacao da Classe FactDoc - Le ou escreve as variaveis membro     /
/   adequadas em arquivos                                                /
/ Parametros:                                                            /
/   ar e uma referencia para o CArchive associado ao arquivo que esta    /
/     sendo usado.                                                       /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTDoc::Serialize(CArchive& ar)
{
  if (ar.IsStoring())
  {
    ar<<car1;
    ar<<car2;

    ar<<pCrushProf1;
    ar<<pCrushProf2;
    ar<<(long)Tipo1;
    ar<<(long)Tipo2;
    ar<<Ocx1;
    ar<<Ocy1;
    ar<<Ocx2;
    ar<<Ocy2;

    ar<<X01;
    ar<<X02;
    ar<<Y01;
    ar<<Y02;
    ar<<Theta01Deg;
    ar<<Theta02Deg;
    ar<<V01;
    ar<<V02;
    ar<<AlfaDeg;
    ar<<Mi;

    ar<<(long)ExecFlag;
    ar<<pTrajeto1;
    ar<<pTrajeto2;
  }
  else
  {
    ar>>car1;
    ar>>car2;

    ar>>pCrushProf1;
    ar>>pCrushProf2;
    ar>>(long&)Tipo1;
    ar>>(long&)Tipo2;
    ar>>Ocx1;
    ar>>Ocy1;
    ar>>Ocx2;
    ar>>Ocy2;

    ar>>X01;
    ar>>X02;
    ar>>Y01;
    ar>>Y02;
    ar>>Theta01Deg;
    Theta01 = DEGTORAD*Theta01Deg;
    ar>>Theta02Deg;
    Theta02 = DEGTORAD*Theta02Deg;
    ar>>V01;
    ar>>V02;
    ar>>AlfaDeg;
    Alfa = DEGTORAD*AlfaDeg;
    ar>>Mi;

    ar>>(long&)ExecFlag;
    ar>>pTrajeto1;
    ar>>pTrajeto2;
  }
}

/*************************************************************************
/ SaveDatabase()                                                         /
/ Atualiza o arquivo cardata.cdb que contem o banco de dados de veiculos /
/ Parametros: -                                                          /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
```

```
void FACTDoc::SaveDatabase()
{
  CFile* pFile = new CFile();
  if (pFile->Open("cardata.cdb", CFile::modeCreate|CFile::modeWrite))
  {
    CArchive ar(pFile, CArchive::store);
    ar<<pCDB;
    ar.Close();
    pFile->Close();
  }
  delete pFile;
}

////////////////////////////////////////////////////////////////////////////
// Mapa de Mensagens

BEGIN_MESSAGE_MAP(FACTDoc, CDocument)
  //{{AFX_MSG_MAP(FACTDoc)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////////
// Diagnostico

#ifdef _DEBUG
void FACTDoc::AssertValid() const
{
  CDocument::AssertValid();
}

void FACTDoc::Dump(CDumpContext& dc) const
{
  CDocument::Dump(dc);
}
#endif //_DEBUG

////////////////////////////////////////////////////////////////////////////
// Comandos

/*****************************************************************************
/ OnNewDocument()                                                           /
/ Apaga e reinicializa as variaveis membro - chamada sempre que um novo     /
/   documento e aberto.                                                     /
/ Parametros: -                                                             /
/ Valor de Retorno: True se a operacao foi realizada com sucesso.           /
 *****************************************************************************/
BOOL FACTDoc::OnNewDocument()
{
  // Default
  BOOL ret = CDocument::OnNewDocument();

  // Se for preciso, salva banco de dados
  if(pCDB && pCDB->GetNumElements())
    SaveDatabase();

  //Abre o Banco de Dados
  CFile* pFile = new CFile();
  if(pFile->Open("cardata.cdb", CFile::modeRead))
  {
    // se o arquivo exisitir com zero bytes, o programa
    // morre horrivelmente com um ABNORMAL PROGRAM TERMINATION
    // (culpa da MFC mesmo)
    if (pFile->GetLength() > 0)
    {
      CArchive ar(pFile, CArchive::load);
      ar>>pCDB;
      ar.Close();
      car1->Copia(pCDB->GetAt(0));
      car2->Copia(pCDB->GetAt(0));
      Ocx1 = car1->GetXF();
      Ocx2 = car2->GetXF();
    }
    pFile->Close();
  }
  delete pFile;

  // Zera Variaveis
  pCrushProf1->DeleteAll();
  pCrushProf2->DeleteAll();
  Tipo1 = 0;
```

```
  Tipo2 = 0;
  Ocy1 = 0.;
  Ocy2 = 0.;

  X01 = 0.;
  X02 = 0.;
  Y01 = 0.;
  Y02 = 0.;
  Theta01 = 0.;
  Theta02 = 0.;
  Theta01Deg = 0.;
  Theta02Deg = 0.;
  V01 = 0.;
  V02 = 0.;
  Alfa = 0.;
  AlfaDeg = 0.;
  Mi = 0.7;

  PassoMax = 1e-4;
  PassoMin = 1e-6;
  TolPos = 1e-4;
  TolVel = 1e-2;
  TolCol = 1e-4;
  F = 0;
  xc1 = 0;
  yc1 = 0;
  xc2 = 0;
  yc2 = 0;

  ExecFlag = FALSE;
  pTrajeto1->DeleteAll();
  pTrajeto2->DeleteAll();

  return(ret);
}

/*************************************************************************
/ OnOpenDocument()                                                       /
/ Se for preciso, salva banco de dados - chamada sempre que um documento  /
/   e aberto.                                                            /
/ Parametros:                                                            /
/   pszPathName e uma string com a path do arquivo a ser aberto.         /
/ Valor de Retorno: True se a operacao foi realizada com sucesso.        /
 *************************************************************************/
BOOL FACTDoc::OnOpenDocument(const char* pszPathName)
{
  // Se for preciso, salva banco de dados
  if(pCDB && pCDB->GetNumElements())
    SaveDatabase();

  // Default
  BOOL ret = CDocument::OnOpenDocument(pszPathName);

  //Abre o Banco de Dados
  CFile* pFile = new CFile();
  if(pFile->Open("cardata.cdb", CFile::modeRead))
  {
    // se o arquivo exisitir com zero bytes, o programa
    // morre horrivelmente com um ABNORMAL PROGRAM TERMINATION
    // (culpa da MFC mesmo)
    if (pFile->GetLength() > 0)
    {
      CArchive ar(pFile, CArchive::load);
      ar>>pCDB;
      ar.Close();
    }
    pFile->Close();
  }
  delete pFile;

  return(ret);
}

/*************************************************************************
/ OnCloseDocument()                                                      /
/ Salva banco de dados - chamada sempre que um documento                 /
/   e fechado.                                                           /
/ Parametros: -                                                          /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
```

```
void FACTDoc::OnCloseDocument()
{
  // Salva o Banco de Dados
  SaveDatabase();

  CDocument::OnCloseDocument();
}

/*****************************************************************************
/ OnFileSaveAs() e  OnFileSave()                                              /
/ Chamam as versoes da classe base. Derivadas somente para virarem public.    /
/ Parametros: -                                                               /
/ Valor de Retorno: -                                                         /
 *****************************************************************************/
void FACTDoc::OnFileSaveAs()
{
  CDocument::OnFileSaveAs();
}

void FACTDoc::OnFileSave()
{
  CDocument::OnFileSave();
}

//////////////////////////////////////////////////////////////////////////////
// Algoritmo

/*****************************************************************************
/ TrajetoRK4()                                                                /
/ Dispara a simulacao, implementando-a atraves do metodo de Runge-Kutta       /
/   de ordem 4 com passo variavel. O controle de passo e feito comparando     /
/   a solucao com a de um metodo de ordem 3. Essa funcao tambem atualiza      /
/   a dialog de Status. Os pontos obtidos sao guardados nos vetores           /
    pTrajeto1 e pTrajeto2                                                     /
/ Parametros:                                                                 /
/   ps e um ponteiro para a dialog de Status                                  /
/ Valor de Retorno: TRUE se a analise foi realizada com sucesso               /
 *****************************************************************************/
BOOL FACTDoc::TrajetoRK4(StatusDlg* pS)
{
  // Auxiliares
  BOOL bContato, bColisao;
  short i;
  char buffer[100];
  double tt;
  double DeltaV, Vmax;
  TrajPoint* pTrajPoint;
  // carro 1:
  // Nos vetores, as variaveis estao na seguinte ordem:
  // vx, x, vy, y, w, theta
  double var41[RK_NUM_EQ]; // solucao no ponto
  double var31[RK_NUM_EQ]; // usado para calcular o erro
  // carro 2:
  double var42[RK_NUM_EQ]; // solucao no ponto
  double var32[RK_NUM_EQ]; // usado para calcular o erro
  // Tempo, controle de passo
  double desvio, erro, quoc;
  BOOL PassoOK;
  double t, amostra;
  double h = PassoMax;

  /* Verifica se a analise ja foi executada e e valida*/
  if(ExecFlag)
    return(TRUE);

  /* Apaga e invalida analises antigas */
  pTrajeto1->DeleteAll();
  pTrajeto2->DeleteAll();
  ExecFlag = FALSE;
  TEndCol = -1;

  /* Calcula parametros da colisao e checa entradas */
  if(!Consis())
    return(FALSE);

  /* Calcula angulo da forca para cada carro */
  alfa1 = CorrigeAngulo(Tipo1, Alfa-Theta01);
  alfa1 += Theta01;
  alfa2 = CorrigeAngulo(Tipo2, Alfa-Theta02);
  alfa2 += Theta02;
```

```
    // Dialog de Status
    pS->ShowWindow(SW_SHOW);
    pS->m_Title = "Colisão";
    pS->m_TLabel = "Tempo:";
    pS->m_Label2 = "Delta V:";

    /* ajusta as condições iniciais */
    // tempo
    t = 0;
    amostra = 0;
    // carro 1
    var41[0] = V01*_cos(Theta01);
    var41[1] = X01;
    var41[2] = V01*_sin(Theta01);
    var41[3] = Y01;
    var41[4] = 0;
    var41[5] = Theta01;
    // carro 2
    var42[0] = V02*_cos(Theta02);
    var42[1] = X02;
    var42[2] = V02*_sin(Theta02);
    var42[3] = Y02;
    var42[4] = 0;
    var42[5] = Theta02;
    // Copia para os vetores auxiliares
    for(i = 0; i < RK_NUM_EQ; i++)
    {
      var31[i] = var41[i];
      var32[i] = var42[i];
    }

    /* executa a simulacao ate que os carros parem */
    while (!Parada(var41[0],var41[2],var41[4],var42[0],var42[2],var42[4], &Vmax))
    {
      /* Antes de mais nada, permite que mensagens do Windows sejam processadas */
      /* Se for processado o Cancel, para a execucao */
      if(!DoEvents())
        return(TRUE);

      PassoOK = FALSE;

      /* Adiciona pontos nas trajetorias, comecando com as condicoes iniciais */
      /* Adiciona somente a cada 2ms durante a colisao */
      if(!F || (t >= amostra))
      {
        pTrajPoint = new TrajPoint(t, var41);
        pTrajeto1->Add(pTrajPoint);
        pTrajPoint = new TrajPoint(t, var42);
        pTrajeto2->Add(pTrajPoint);
        tt = t;
        amostra += 0.002;
      }

      sprintf(buffer, "%9.4lf", Vmax);

      /* Verifica se ainda ha força entre os carros, senao zera */
      if(F)  // Se a forca ja foi zerada, nao verifica mais
      {
        // Verifica se os carros estao em contato
        bContato = Contato(var41[1],var41[3],var42[1],var42[3],var41[5],var42[5]);
        // Verifica se suas velocidades sao diferentes (modulo e direcao)
        bColisao                                                                    =
Colisao(var41[5],var42[5],var41[0],var41[2],var42[0],var42[2],var41[4],var42[4],&DeltaV)
;
        if (!bColisao  || !bContato)  // Fim da Colisao
        {
          F = 0;
          TEndCol = t;
          PassoMax = 2e-2;            // Apos a colisao, pode aumentar o passo
          pS->m_Title = "Trajetória";
          pS->m_Label2 = "Vmax:";
        }

        sprintf(buffer, "%9.4lf", DeltaV);
      }

      // Dialog de Status
      pS->m_Value2 = buffer;
      sprintf(buffer, "%9.4lf", t);
```

```
  pS->m_Tempo = buffer;
  pS->UpdateData(FALSE);

  while (!PassoOK)
  {
    /* faz as integracoes */
    PassoRK(car1, t, h, var41, var31);  // carro 1
    PassoRK(car2, t, h, var42, var32);  // carro 2

    /* calcula os erros de posicao */
    erro = 0;
    // desvio de posicao X do carro 1
    desvio = fabs(var41[1] - var31[1]);
    erro = (desvio>erro)?desvio:erro;
    // desvio de posicao Y do carro 1
    desvio = fabs(var41[3] - var31[3]);
    erro = (desvio>erro)?desvio:erro;
    // desvio de posicao X do carro 2
    desvio = fabs(var42[1] - var32[1]);
    erro = (desvio>erro)?desvio:erro;
    // desvio de posicao Y do carro 2
    desvio = fabs(var42[3] - var32[3]);
    erro = (desvio>erro)?desvio:erro;

    /* verifica o erro e corrige o passo */
    quoc = erro/(TolPos*h);
    if(fabs(quoc) >  1e-4)  // Erro nao-nulo
    {
      if(quoc < 1)          // Aumenta Passo, Passo OK
      {
        t += h;
        PassoOK = TRUE;
        if(h < PassoMax)    // So se for menor que o maximo
        {
          h /= sqrt(quoc);
          if(h > PassoMax)  // Nao deixa passar do maximo
            h = PassoMax;
        }
      }
      else                  // Reduz Passo
        if(h > PassoMin)    // Se for maior que o minimo
        {
          h /= sqrt(quoc);  // Nao deixa passar do minimo
          if(h < PassoMin)
            h = PassoMin;
        }
    }
    else                    // Erro Nulo
    {
      t += h;               // Aumenta Passo, Passo OK
      PassoOK = TRUE;
      if(h < PassoMax)      // So se for menor que o maximo
      {
        h *= 3;
        if(h > PassoMax)    // Nao deixa passar do maximo
          h = PassoMax;
      }
    }
  }
}

// Garante que o ultimo ponto e salvo
if (pTrajeto1->GetAt(pTrajeto1->GetUpperBound())->GetT() != t)
{
  pTrajPoint = new TrajPoint(t, var41);
  pTrajeto1->Add(pTrajPoint);
  pTrajPoint = new TrajPoint(t, var42);
  pTrajeto2->Add(pTrajPoint);
  tt = t;
}

// Se o tempo continuar -1, o fim da colisao corresponde ao fim da simulacao
if(TEndCol == -1)
  TEndCol = tt;

// Simulacao OK, volta ao passo para colisao
PassoMax = 1e-4;
ExecFlag = TRUE;
SetModifiedFlag();
return(TRUE);
```

```
}

/*************************************************************************
/ Consis()                                                               /
/ Verifica se os dados sao consistentes, para que, caso contrario, a     /
/   analise possa ser cancelada                                          /
/ Parametros: -                                                          /
/ Valor de Retorno: TRUE se os dados sao consistentes                    /
 *************************************************************************/
BOOL FACTDoc::Consis()
{
  double temp;

  // Checa se ha deformacao
  if(!(pCrushProf1->GetSize()) || !(pCrushProf2->GetSize()))
  {
    MessageBox(NULL,
    "Os perfis de deformação não estão definidos. Análise cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa se as massas nao sao nulas
  if((car1->GetMassa() <= 0) || (car2->GetMassa() <= 0))
  {
    MessageBox(NULL,
    "A massa dos veículos deve ser estritamente positiva. Análise cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa se os raios de giracao nao sao nulos
  if((car1->GetRaioGirQ() <= 0) || (car2->GetRaioGirQ() <= 0))
  {
    MessageBox(NULL,
    "O raio de giração dos veículos deve ser estritamente positivo. Análise cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa se ha velocidade inicial relativa entre os veiculos
  if(!Colisao(Theta01, Theta02, V01*_cos(Theta01), V01*_sin(Theta01), V02*_cos(Theta02),
V02*_sin(Theta02), 0, 0, &temp))
  {
    MessageBox(NULL,
    "A velocidade relativa entre os veículos é nula na condição inicial. Análise
cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa se ha contato entre os veiculos inicialmente
  if(!Contato(X01, Y01, X02, Y02, Theta01, Theta02))
  {
    MessageBox(NULL,
    "Os veículos não estão em contato na condição inicial. Análise cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa se todos os angulos estao entre -180 e 180
  if((Theta01 < -PI || Theta01 > PI) ||
     (Theta02 < -PI || Theta02 > PI) ||
     (Alfa    < -PI || Alfa    > PI))
  {
    MessageBox(NULL,
    "Os ângulos devem ser definidos no intervalo [-180, 180]. Análise cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa consistencia do angulo da forca (alfa)
  if(!V01)
    if((Alfa > (Theta02 + PI/6.)  || Alfa < (Theta02 - PI/6.)) &&
       (Alfa > (Theta02 - 5*PI/6) || Alfa < (Theta02 - 7*PI/6)))
    {
      MessageBox(NULL,
      "O ângulo da força é inadequado. Análise cancelada.",
      "Aviso", MB_OK);
       return(FALSE);
```

```
    }
  if(!V02)
    if((Alfa > (Theta01 + PI/6.)  || Alfa < (Theta01 - PI/6.)) &&
       (Alfa > (Theta01 - 5*PI/6) || Alfa < (Theta01 - 7*PI/6)))
    {
      MessageBox(NULL,
      "O ângulo da força é inadequado. Análise cancelada.",
      "Aviso", MB_OK);
       return(FALSE);
    }
  double t01 = (Theta01<0)?Theta01+PI:Theta01;
  double t02 = (Theta02<0)?Theta02+PI:Theta02;
  t01 = (fabs(Theta01-PI)<1e-4)?0:t01;
  t02 = (fabs(Theta02-PI)<1e-4)?0:t02;
  double tmax = __max(t01, t02);
  double tmin = __min(t01, t02);
  if((Alfa < (tmax - PI) || Alfa > tmin) &&
     (Alfa < tmax) && (Alfa > (tmin - PI)))
  {
    MessageBox(NULL,
    "O ângulo da força é inadequado. Análise cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa se os modulos das forcas de deformacao sao proximos
  F = Deforma(car1, pCrushProf1, Ocx1, Ocy1, &xc1, &yc1, Tipo1);
  temp = Deforma(car2, pCrushProf2, Ocx2, Ocy2, &xc2, &yc2, Tipo2);
  if(fabs(F-temp)/F > 0.2)
    if(MessageBox(NULL,
    "Os perfis de deformação fornecem forças que diferem em mais de 20%. Continuar mesmo
assim?",
    "Aviso", MB_YESNO) == IDNO)
      return(FALSE);
  F = (F+temp)/2;

  // Checa se os perfis nao sao muito profundos
  int n = pCrushProf1->GetSize();
  double max1 = 0;
  for(int i = 0; i < n; i++)
  {
    temp = pCrushProf1->GetAt(i)->GetProf();
    if(temp > max1)
      max1 = temp;
  }
  n = pCrushProf2->GetSize();
  double max2 = 0;
  for(i = 0; i < n; i++)
  {
    temp = pCrushProf2->GetAt(i)->GetProf();
    if(temp > max2)
      max2 = temp;
  }
  max1/=1000.;
  max2/=1000.;
  if(max1>(car1->GetYS()) || max2>(car2->GetYS()))
  {
    MessageBox(NULL,
    "Um dos perfis de amassamento é muito profundo. Análise Cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  // Checa posicao do amassamento
  double xc1g = xc1*_cos(Theta01) - yc1*_sin(Theta01) + X01;
  double yc1g = xc1*_sin(Theta01) + yc1*_cos(Theta01) + Y01;
  double xc2g = xc2*_cos(Theta02) - yc2*_sin(Theta02) + X02;
  double yc2g = xc2*_sin(Theta02) + yc2*_cos(Theta02) + Y02;
  double dist = sqrt((xc1g-xc2g)*(xc1g-xc2g) + (yc1g-yc2g)*(yc1g-yc2g));
  if(2*dist >= __max(car1->GetYS(), car2->GetYS()))
  {
    MessageBox(NULL,
    "O perfil de amassamento não é compatível com o cenário. Análise Cancelada.",
    "Aviso", MB_OK);
     return(FALSE);
  }

  return(TRUE);
}
```

```
/*************************************************************************
/ PassoRK()                                                              /
/ Calcula um passo de integracao pelos metodos de Runge-Kutta de ordem 3 e /
/   4, para um valor de h conhecido                                      /
/ Parametros:                                                           /
/   pCar - Ponteiro para um objeto CarData com os parametros do carro    /
/     cujas equacoes de movimento estao sendo integradas                 /
/   t - tempo (atualmente nao e usado)                                   /
/   h - passo de integracao                                              /
/   var4 - comeca com as condicoes do passo anterior e recebe as condicoes /
/     do fim deste passo para o metodo de ordem 4                        /
/   var3 - comeca com as condicoes do passo anterior e recebe as condicoes /
/     do fim deste passo para o metodo de ordem 3                        /
/ Valor de Retorno: -                                                   /
 *************************************************************************/
void FACTDoc::PassoRK(CarData* pCar, double t, double h, double* var4, double* var3)
{
  // define os k's usados para calcular RK3 e RK4
  double k1[RK_NUM_EQ];
  double k2[RK_NUM_EQ];
  double k3[RK_NUM_EQ];
  double k3b[RK_NUM_EQ];
  double k4[RK_NUM_EQ];
  // variaveis auxiliares
  double kbuf[RK_NUM_EQ];
    short i;

  // calcula k1
  EqDifs(pCar, t,var4,k1);

  // calcula k2
  for (i=0; i<RK_NUM_EQ; i++)
    kbuf[i] = var4[i]+(h*k1[i]/2.0);
  EqDifs(pCar, t+(h/2.0), kbuf, k2);

  // calcula k3
  for (i=0; i<RK_NUM_EQ; i++)
    kbuf[i] = var4[i]+(h*k2[i]/2.0);
  EqDifs(pCar, t+(h/2.0), kbuf, k3);

  // calcula k4
  for (i=0; i<RK_NUM_EQ; i++)
    kbuf[i] = var4[i]+(h*k3[i]);
  EqDifs(pCar, t+h, kbuf, k4);

  // calcula k3b - para RK3
  for (i=0; i<RK_NUM_EQ; i++)
    kbuf[i] = var4[i]+(3.0*h*k2[i]/4.0);
  EqDifs(pCar, t+(3.0*h/4.0), kbuf, k3b);

  // realiza o passo de integração, tanto por RK3 como por RK4
  for (i=0; i<RK_NUM_EQ; i++)
  {
    var3[i] = var4[i] + (h*(2*k1[i]+3*k2[i]+4*k3b[i])/9.0);
    var4[i] = var4[i] + (h*(k1[i]+2*k2[i]+2*k3[i]+k4[i])/6.0);
  }
}

/*************************************************************************
/ EqDifs()                                                               /
/ Para cada uma das variaveis contidas no vetor in, calcula dx/dt = f(x)  /
/   e armazena o resultado dx/dt no vetor out                            /
/ Parametros:                                                           /
/   pCar - Ponteiro para um objeto CarData com os parametros do carro    /
/     cujas equacoes de movimento estao sendo integradas                 /
/   t - tempo (atualmente nao e usado)                                   /
/   in - vetor de entradas                                               /
/   out - vetor de saidas                                                /
/ Valor de Retorno: -                                                   /
 *************************************************************************/
void FACTDoc::EqDifs(CarData* pCar, double t, double* in, double* out)
{
  double Fx = 0, Fy = 0, Tz = 0;  // Esforcos
  double Taux;                    // esforco auxiliar
  double Izz;                     // Momento de Inercia do carro em torno de z
  double mod;                     // modulo da velocidade
  double ef, er, es;              // Bracos dos pneus
//double xc1, yc1, xc2, yc2       -> Bracos da forca de contato (Globais)
  // Angulos
  double cost = _cos(in[5]);
```

```
  double sint = _sin(in[5]);

  /* Posicao dos pneus */
  ef = pCar->GetEF();
  er = pCar->GetER();
  es = (pCar->GetTrack())/2;

  /* Calcula Izz */
  Izz = (pCar->GetMassa())*(pCar->GetRaioGirQ());

  // Colisao
  if(F)
  {
    if(pCar == car1)
    {
      Fx = F*_cos(alfa1);
      Fy = F*_sin(alfa1);
      Tz = -Fx*(xc1*sint + yc1*cost) + Fy*(xc1*cost - yc1*sint);
    }
    else
    {
      Fx = F*_cos(alfa2);
      Fy = F*_sin(alfa2);
      Tz = -Fx*(xc2*sint + yc2*cost) + Fy*(xc2*cost - yc2*sint);
    }
  }

  // Frenagem
  // torque
  if (fabs(in[4])>TolVel)
  {
    Taux = -_GRAV_*(pCar->GetMassa())*Mi*(ef+er)*in[4]/fabs(in[4]);
    if ((fabs(Taux)>fabs(Tz))&&(F))
      Tz = 0;
    else
      Tz += Taux;
  }
  // forca
  mod = sqrt(in[0]*in[0] + in[2]*in[2]);
  if (fabs(in[0])>TolVel)
  {
    Fx -= _GRAV_*(pCar->GetMassa())*Mi*in[0]/mod;
  }
  if (fabs(in[2])>TolVel)
  {
    Fy -= _GRAV_*(pCar->GetMassa())*Mi*in[2]/mod;
  }

  /* Preenche o vetor com as saidas */
  out[0] = Fx/(pCar->GetMassa());     // dvx/dt = Fx/m
  out[1] = in[0];                     // dx/dt = vx
  out[2] = Fy/(pCar->GetMassa());     // dvy/dt = Fy/m
  out[3] = in[2];                     // dy/dt = vy
  out[4] = Tz/Izz;                    // dw/dt = T/I
  out[5] = in[4];                     // dtheta/dt = w
}

/*************************************************************************
/ Parada()                                                               /
/ Verifica se ambos os carros estao parados                              /
/ Parametros:                                                            /
/   Vx1 - Velocidade x do veiculo 1                                      /
/   Vy1 - Velocidade y do veiculo 1                                      /
/   Vx2 - Velocidade x do veiculo 2                                      /
/   Vy2 - Velocidade y do veiculo 2                                      /
/   w1 - Velocidade angular do veiculo 1                                 /
/   w2 - Velocidade angular do veiculo 2                                 /
/   Vmax - recebe a velocidade máxima                                    /
/ Valor de Retorno: TRUE se todas as velocidades forem nulas             /
 *************************************************************************/
BOOL FACTDoc::Parada(double Vx1, double Vy1, double Vx2, double Vy2, double w1, double
w2, double* Vmax)
{
  BOOL ret;
  double temp;

  // ret so e verdadeiro se todas as velocidades forem nulas (menores que TolVel)
  temp = fabs(Vx1);
  *Vmax = temp;
  temp = fabs(Vy1);
```

```
  if(temp > *Vmax)
    *Vmax = temp;
  temp = fabs(Vx2);
  if(temp > *Vmax)
    *Vmax = temp;
  temp = fabs(Vy2);
  if(temp > *Vmax)
    *Vmax = temp;
  temp = fabs(w1);
  if(temp > *Vmax)
    *Vmax = temp;
  temp = fabs(w2);
  if(temp > *Vmax)
    *Vmax = temp;

  ret = (*Vmax < TolVel);

  return(ret);
}

/*************************************************************************
/ Colisao()                                                             /
/ Verifica se a colisao nao terminou devido ao pontos de contato dos    /
/   veiculos terem atingido a mesma velocidade                          /
/ Parametros:                                                           /
/   theta1 - angulo do carro 1                                          /
/   theta2 - angulo do carro 2                                          /
/   Vx1 - Velocidade x do veiculo 1                                     /
/   Vy1 - Velocidade y do veiculo 1                                     /
/   Vx2 - Velocidade x do veiculo 2                                     /
/   Vy2 - Velocidade y do veiculo 2                                     /
/   w1 - Velocidade angular do veiculo 1                                /
/   w2 - Velocidade angular do veiculo 2                                /
/   DeltaV - recebe a velocidade máxima relativa entre os veiculos      /
/ Valor de Retorno: TRUE se a colisao continua                          /
 *************************************************************************/
BOOL FACTDoc::Colisao(double theta1, double theta2, double Vx1, double Vy1, double Vx2,
double Vy2, double w1, double w2, double* DeltaV)
{
  BOOL ret;
  double temp;
  // Angulos
  double cost1 = _cos(theta1);
  double sint1 = _sin(theta1);
  double cost2 = _cos(theta2);
  double sint2 = _sin(theta2);

  // Calcula velocidade do ponto de colisao Vc para os 2 carros
  double Vxc1 = Vx1 + w1*(xc1*cost1 - yc1*sint1);
  double Vyc1 = Vy1 - w1*(xc1*sint1 + yc1*cost1);
  double Vxc2 = Vx2 + w2*(xc2*cost2 - yc2*sint2);
  double Vyc2 = Vy2 - w2*(xc2*sint2 + yc2*cost2);

  // ret so e verdadeiro a diferenca entre Vc1 e Vc2 for nula (menor que TolVel) em x e
y
  temp = fabs(Vxc1-Vxc2);
  *DeltaV = temp;
  temp = fabs(Vyc1-Vyc2);
  if(temp > *DeltaV)
    *DeltaV = temp;

  ret = (*DeltaV > TolVel);

  return(ret);
}

/*************************************************************************
/ Colisao()                                                             /
/ Verifica se ha contato entre os veiculos                              /
/ Parametros:                                                           /
/   xg1 - posicao x do baricentro do veiculo 1                          /
/   yg1 - posicao y do baricentro do veiculo 1                          /
/   xg2 - posicao x do baricentro do veiculo 2                          /
/   yg2 - posicao y do baricentro do veiculo 2                          /
/   theta1 - angulo do carro 1                                          /
/   theta2 - angulo do carro 2                                          /
/ Valor de Retorno: TRUE se houver contato                              /
 *************************************************************************/
BOOL FACTDoc::Contato(double xg1, double yg1, double xg2, double yg2, double theta1,
double theta2)
```

```
{
  // pontos do carro 1
  // (1o. indice = carro, 2o. indice = ponto)
  double x1[4], y1[4];
  // pontos do carro 2
  double x2[4], y2[4];
  // normais do carro 1
  double nx1[4], ny1[4];
  // normais do carro 2
  double nx2[4], ny2[4];
  // variaveis auxiliares
  double xf, xr, ys, st, ct;
  double opx, opy;
  short i;
  BOOL ret1, ret2, ret;

  // calcula os pontos do carro 1
  xf = car1->GetXF() + TolCol;
  xr = car1->GetXR() + TolCol;
  ys = car1->GetYS() + TolCol;
  st = _sin(theta1);
  ct = _cos(theta1);
  x1[0] = xg1 + xf*ct - ys*st; // frente, esquerda
  y1[0] = yg1 + xf*st + ys*ct;
  x1[1] = xg1 - xr*ct + ys*st; // tras, direita
  y1[1] = yg1 - xr*st - ys*ct;
  x1[2] = xg1 + xf*ct + ys*st; // frente, direita
  y1[2] = yg1 + xf*st - ys*ct;
  x1[3] = xg1 - xr*ct - ys*st; // tras, esquerda
  y1[3] = yg1 - xr*st + ys*ct;

  // calcula as normais do carro 1
  nx1[0] = ct;
  ny1[0] = st;
  nx1[1] = -st;
  ny1[1] = ct;
  nx1[2] = -ct;
  ny1[2] = -st;
  nx1[3] = st;
  ny1[3] = -ct;

  // calcula os pontos do carro 2
  xf = car2->GetXF();
  xr = car2->GetXR();
  ys = car2->GetYS();
  st = _sin(theta2);
  ct = _cos(theta2);
  x2[0] = xg2 + xf*ct - ys*st;
  y2[0] = yg2 + xf*st + ys*ct;
  x2[1] = xg2 - xr*ct + ys*st;
  y2[1] = yg2 - xr*st - ys*ct;
  x2[2] = xg2 + xf*ct + ys*st;
  y2[2] = yg2 + xf*st - ys*ct;
  x2[3] = xg2 - xr*ct - ys*st;
  y2[3] = yg2 - xr*st + ys*ct;

  // calcula as normais do carro 1
  nx2[0] = ct;
  ny2[0] = st;
  nx2[1] = -st;
  ny2[1] = ct;
  nx2[2] = -ct;
  ny2[2] = -st;
  nx2[3] = st;
  ny2[3] = -ct;

  /* verifica colisao */

  // carro 1 "dentro" do carro 2
  ret1 = 0;
  for (i=0; i<4; i++)
  {
    ret = 1;
    opx = x1[i] - x2[0];
    opy = y1[i] - y2[0];
    ret = ret && ((opx*nx2[2] + opy*ny2[2]) >= 0);
    ret = ret && ((opx*nx2[3] + opy*ny2[3]) >= 0);
    opx = x1[i] - x2[1];
    opy = y1[i] - y2[1];
    ret = ret && ((opx*nx2[0] + opy*ny2[0]) >= 0);
```

```
    ret = ret && ((opx*nx2[1] + opy*ny2[1]) >= 0);
    ret1 = ret1 || ret;
  }

  // carro 2 "dentro" do carro 1
  ret2 = 0;
  for (i=0; i<4; i++)
  {
    ret = 1;
    opx = x2[i] - x1[0];
    opy = y2[i] - y1[0];
    ret = ret && ((opx*nx1[2] + opy*ny1[2]) >= 0);
    ret = ret && ((opx*nx1[3] + opy*ny1[3]) >= 0);
    opx = x2[i] - x1[1];
    opy = y2[i] - y1[1];
    ret = ret && ((opx*nx1[0] + opy*ny1[0]) >= 0);
    ret = ret && ((opx*nx1[1] + opy*ny1[1]) >= 0);
    ret2 = ret2 || ret;
  }

  return (ret1 || ret2);
}

/*****************************************************************************
/ Deforma()                                                                  /
/ Dado um veiculo, um perfil de deformacao e o tipo de colisao calcula a     /
/  forca de colisao e a posicao do ponto de contato                          /
/ Parametros:                                                                /
/   pCar - Ponteiro para um objeto CarData com os parametros do carro        /
/     cujas equacoes de movimento estao sendo integradas                     /
/   pCrush - perfil de deformacao                                            /
/   Ocx, Ocy - origem da deformaca na base solidaria ao veiculo              /
/   xc, yc - recebem a posicao do ponto de colisao na base solidaria ao      /
/     veiculo                                                                /
/   tipo - tipo de colisao                                                   /
/ Valor de Retorno: Valor da forca de colisao                                /
 *****************************************************************************/
double FACTDoc::Deforma(CarData* pCar, CrushArray* pCrush, double Ocx, double Ocy,
double* xc, double* yc, UINT Tipo)
{
  short i;
  short n = pCrush->GetSize();
  double Area = 0;
  double tx, ty;
  double A, B;
  double F;
  Crush *pAnt, *pI;
  Crush *pCC, *pCF;
  BOOL bComeco, bFim;

  bComeco = FALSE;
  bFim = FALSE;

  if (pCrush->GetAt(0) != 0)
  {
    pCC = new Crush;
    pCC->SetPos(pCrush->GetAt(0)->GetPos());
    pCrush->InsertAt(0, pCC);
    n++;
    bComeco = TRUE;
  }

  if (pCrush->GetAt(n-1) != 0)
  {
    pCF = new Crush;
    pCF->SetPos(pCrush->GetAt(n-1)->GetPos());
    pCrush->Add(pCF);
    n++;
    bFim = TRUE;
  }

  // area do amassamento
  for(i=1; i < n; i++)
  {
    pI = pCrush->GetAt(i);
    pAnt = pCrush->GetAt(i-1);
    Area += (pI->GetProf() + pAnt->GetProf())*(pI->GetPos() - pAnt->GetPos());
  }

  Area /= 2.;
```

```
  // calcula o X do baricentro do amassamento
  *xc = 0;
  for(i=1; i < n; i++)
  {
    pI = pCrush->GetAt(i);
    pAnt = pCrush->GetAt(i-1);
    *xc = *xc + (pI->GetPos()*pI->GetProf() + pAnt->GetPos()*pAnt->GetProf())*(pI-
>GetPos() - pAnt->GetPos());
  }
  *xc /= Area*2000.;

  // calcula o Y do baricentro do amassamento
  *yc = 0;
  for(i=1; i < n; i++)
  {
    pI = pCrush->GetAt(i);
    pAnt = pCrush->GetAt(i-1);
    *yc = *yc + (pI->GetProf()*pI->GetProf() + pAnt->GetProf()*pAnt->GetProf())*(pI-
>GetPos() - pAnt->GetPos());
  }
  *yc /= Area*4000.;

  Area /= 1e6;

  /* De acordo com o tipo do carro, acha rigidez e centro de amassamento */
  switch(Tipo)
  {
    case 0:
      tx = Ocx - *yc;
      ty = Ocy + *xc;
      A = pCar->GetA(0);
      B = pCar->GetB(0);
    break;

    case 1:
      tx = Ocx + *yc;
      ty = Ocy + *xc;
      A = pCar->GetA(1);
      B = pCar->GetB(1);
    break;

    case 2:
      tx = Ocx + *xc;
      ty = Ocy - *yc;
      A = pCar->GetA(2);
      B = pCar->GetB(2);
    break;

    case 3:
      tx = Ocx + *xc;
      ty = Ocy + *yc;
      A = pCar->GetA(2);
      B = pCar->GetB(2);
    break;
  }

  *xc = tx;
  *yc = ty;

  // calcula forca
  double x1 = pCrush->GetAt(0)->GetPos();
  double x2 = pCrush->GetAt(n-1)->GetPos();
  F = A*(x2-x1)/1e3 + B*Area;

  if (bFim)
  {
    pCrush->RemoveAt(n-1);
    delete pCF;
  }

  if (bComeco)
  {
    pCrush->RemoveAt(0);
    delete pCC;
  }

  return(F);
}
```

```
/*************************************************************************
/ CorrigeAngulo()                                                        /
/ Corrige o angulo alfa da forca de colisao para garantir que a forca em  /
/   cada carro sempre atue para separa-los                               /
/ Parametros:                                                            /
/   Tipo - tipo de colisao                                               /
/   ang - angulo a ser corrigido                                         /
/ Valor de Retorno: valor do angulo corrigido                            /
 *************************************************************************/
double FACTDoc::CorrigeAngulo(UINT Tipo, double ang)
{
  // primeiro, converte ang para que fique entre 0 e 2*pi
  ang = fmod(ang,2.*PI);
  if (ang<0)
    ang = 2.*PI+ang;
  // agora, converte conforme o caso
  switch(Tipo)
  {
    // batida de frente - segundo e terceiro quadrantes
    case 0:
      if (ang<PI/2.)
        ang = ang + PI;
      else if (ang>3.*PI/2.)
        ang = ang - PI;
      break;
    // batida de tras - primeiro e quarto quadrantes
    case 1:
      if ((ang>PI/2.)&&(ang<PI))
        ang = ang + PI;
      else if ((ang>=PI)&&(ang<3.*PI/2.))
        ang = ang - PI;
      break;
    // batida do lado esquerdo - terceiro e quarto quadrantes
    case 2:
      if (ang<PI)
        ang = ang + PI;
      break;
    // batida do lado direito - primeiro e segundo quadrantes
    case 3:
      if (ang>PI)
        ang = ang - PI;
  }

  return ang;
}

////////////////////////////////////////////////////////////////////////////
// Funcoes Matematicas

/*************************************************************************
/ _sin() e _cos()                                                        /
/ seno e cosseno foram refeitas para reduzir erros de precisao numerica  /
/ Parametros:                                                            /
/  ang - angulo em radianos                                              /
/ Valor de Retorno: valor da funcao                                      /
 *************************************************************************/
double FACTDoc::_sin(double ang)
{
  double r = sin(ang);

  if(fabs(r) < 1e-4)
    r = 0;
  if(fabs(r-1) < 1e-4)
    r = 1;
  if(fabs(r+1) < 1e-4)
    r = -1;

  return(r);
}

double FACTDoc::_cos(double ang)
{
  double r = cos(ang);
  if(fabs(r) < 1e-4)
    r = 0;
  if(fabs(r-1) < 1e-4)
    r = 1;
  if(fabs(r+1) < 1e-4)
    r = -1;
```

```
  return(r);
}

////////////////////////////////////////////////////////////////////////////
// Funcoes de Acesso

/*************************************************************************
/ Funcoes Set[VARIAVEL]()                                                 /
/ Todas estas funcoes copiam o valor recebido em arg para uma variavel    /
/   membro do documento, e modificam as flags necessarias. As funcoes que /
/   modificam objetos mais complexos tambem recebem um parametro que      /
/   indica se o novo valor e diferente do antigo ou nao                   /
/ Parametros:                                                             /
/   arg ou pArg - parametro com o novo valor                              /
/   mudou - indica se o novo valor e diferente do antigo (presente so em  /
/     algumas funcoes                                                     /
/ Valor de Retorno: -                                                     /
 *************************************************************************/

/*************************************************************************
/ Funcoes Get[VARIAVEL]()                                                 /
/ Todas estas funcoes fornecem uma copia contendo o valor contido numa    /
/   variavel membro. A maioria retorna esse valor. Algumas recebem uma    /
/   variavel e copiam a variavel membro nesta variavel                    /
/ Parametros:                                                             /
/   pArg - presente somente nas funcoes sem valor de retorno, recebe o    /
/     valor da variavel acessada                                          /
/ Valor de Retorno: o valor da variavel acessada                          /
 *************************************************************************/

void FACTDoc::SetCar1(BOOL mudou, CarData* pArg)
{
  if(mudou)
  {
    car1->Copia(pArg);
    ExecFlag = FALSE;
    SetModifiedFlag();
  }
}

void FACTDoc::GetCar1(CarData* pArg)
{
  pArg->Copia(car1);
}

void FACTDoc::SetCar2(BOOL mudou, CarData* pArg)
{
  if(mudou)
  {
    car2->Copia(pArg);
    ExecFlag = FALSE;
    SetModifiedFlag();
  }
}

void FACTDoc::GetCar2(CarData* pArg)
{
  pArg->Copia(car2);
}

void FACTDoc::SetCrush1(BOOL mudou, CrushArray* pArg)
{
  if(mudou)
  {
    pCrushProf1->DeleteAll();
    int n = pArg->GetSize();
    for(int i = 0; i < n; i++)
      pCrushProf1->Add(new Crush(*(pArg->GetAt(i))));
    ExecFlag = FALSE;
    SetModifiedFlag();
  }
}

void FACTDoc::GetCrush1(CrushArray* pArg)
{
  pArg->DeleteAll();
  int n = pCrushProf1->GetSize();
  for(int i = 0; i < n; i++)
    pArg->Add(new Crush(*(pCrushProf1->GetAt(i))));
}
```

```
void FACTDoc::SetCrush2(BOOL mudou, CrushArray* pArg)
{
  if(mudou)
  {
    pCrushProf2->DeleteAll();
    int n = pArg->GetSize();
    for(int i = 0; i < n; i++)
      pCrushProf2->Add(new Crush(*(pArg->GetAt(i))));
    ExecFlag = FALSE;
    SetModifiedFlag();
  }
}

void FACTDoc::GetCrush2(CrushArray* pArg)
{
  pArg->DeleteAll();
  int n = pCrushProf2->GetSize();
  for(int i = 0; i < n; i++)
    pArg->Add(new Crush(*(pCrushProf2->GetAt(i))));
}

void FACTDoc::SetTipo1(UINT arg)
{
  if(Tipo1 != arg)
  {
    Tipo1 = arg;
    ExecFlag = FALSE;
  }
}

UINT FACTDoc::GetTipo1() const
{
  return(Tipo1);
}

void FACTDoc::SetTipo2(UINT arg)
{
  if(Tipo2 != arg)
  {
    Tipo2 = arg;
    ExecFlag = FALSE;
  }
}

UINT FACTDoc::GetTipo2() const
{
  return(Tipo2);
}

void FACTDoc::SetOcx1(double arg)
{
  if(Ocx1 != arg)
  {
    Ocx1 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetOcx1() const
{
  return(Ocx1);
}

void FACTDoc::SetOcx2(double arg)
{
  if(Ocx2 != arg)
  {
    Ocx2 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetOcx2() const
{
  return(Ocx2);
}

void FACTDoc::SetOcy1(double arg)
{
```

```
  if(Ocy1 != arg)
  {
    Ocy1 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetOcy1() const
{
  return(Ocy1);
}

void FACTDoc::SetOcy2(double arg)
{
  if(Ocy2 != arg)
  {
    Ocy2 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetOcy2() const
{
  return(Ocy2);
}

void FACTDoc::SetX01(double arg)
{
  if(X01 != arg)
  {
    X01 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetX01() const
{
  return(X01);
}

void FACTDoc::SetX02(double arg)
{
  if(X02 != arg)
  {
    X02 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetX02() const
{
  return(X02);
}

void FACTDoc::SetY01(double arg)
{
  if(Y01 != arg)
  {
    Y01 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetY01() const
{
  return(Y01);
}

void FACTDoc::SetY02(double arg)
{
  if(Y02 != arg)
  {
    Y02= arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetY02() const
{
  return(Y02);
```

```
}

void FACTDoc::SetTheta01(double arg)
{
  if(Theta01Deg != arg)
  {
    Theta01Deg = arg;
    Theta01 = Theta01Deg*DEGTORAD;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetTheta01() const
{
  return(Theta01Deg);
}

void FACTDoc::SetTheta02(double arg)
{
  if(Theta02Deg != arg)
  {
    Theta02Deg = arg;
    Theta02 = Theta02Deg*DEGTORAD;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetTheta02() const
{
  return(Theta02Deg);
}

void FACTDoc::SetV01(double arg)
{
  if(V01 != arg)
  {
    V01 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetV01() const
{
  return(V01);
}

void FACTDoc::SetV02(double arg)
{
  if(V02 != arg)
  {
    V02 = arg;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetV02() const
{
  return(V02);
}

void FACTDoc::SetAlfa(double arg)
{
  if(AlfaDeg != arg)
  {
    AlfaDeg = arg;
    Alfa = AlfaDeg*DEGTORAD;
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetAlfa() const
{
  return(AlfaDeg);
}

void FACTDoc::SetMi(double arg)
{
  if(Mi != arg)
  {
    Mi = arg;
```

```
    ExecFlag = FALSE;
  }
}

double FACTDoc::GetMi() const
{
  return(Mi);
}

CarDatabase* FACTDoc::GetCDB()
{
  return(pCDB);
}

BOOL FACTDoc::GetExecFlag()
{
  return ExecFlag;
}

double FACTDoc::GetTEndCol() const
{
  return(TEndCol);
}

double FACTDoc::GetLastT()
{
  short numPt;
  double T;
  numPt = pTrajeto1->GetSize();
  T = pTrajeto1->GetAt(numPt-1)->GetT();
  return T;
}

void FACTDoc::GetMaxRect(double* x0, double* y0, double* x1, double* y1)
{
  Trajetoria* pT;
  short numPt, cur, i;
  double x, y;
  numPt = pTrajeto1->GetSize();
  *x0 = 0;
  *y0 = 0;
  *x1 = 0;
  *y1 = 0;
  for (i=0;i<2;i++)
  {
    if (i==0)
      pT = pTrajeto1;
    else
      pT = pTrajeto2;
    for (cur=0;cur<numPt;cur++)
    {
      x = pT->GetAt(cur)->GetX();
      y = pT->GetAt(cur)->GetY();
      if (*x0 > x)
        *x0 = x;
      if (*x1 < x)
        *x1 = x;
      if (*y0 > y)
        *y0 = y;
      if (*y1 < y)
        *y1 = y;
    }
  }
}

/*************************************************************************
/ GetMinMaxTraj()                                                        /
/ Encontra os valores maximo e minimo de todas as variaveis da trajetoria /
/ Parametros:                                                            /
/   pTrajMin - ponto que recebe os valores minimos                       /
/   minV - minimo modulo da velocidade                                   /
/   pTrajMax - ponto que recebe os valores maximos                       /
/   maxV - maximo modulo da velocidade                                   /
/   flags - 1 pesquisa somente valores d pTrajeto1, 2 de pTrajeto2 e 3 de /
/     ambos                                                              /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTDoc::GetMinMaxTraj(TrajPoint* pTrajMin, double* minV, TrajPoint* pTrajMax,
double* maxV, short flags)
{
```

```
  Trajetoria* pT;
  TrajPoint*  pTraj;
  short numPt, cur, i;
  short ini, fim;
  double v;

  ini = (flags & 1)?0:1;
  fim = (flags & 2)?2:1;

  numPt = pTrajeto1->GetSize();

  pTrajMin->operator=(*(pTrajeto1->GetAt(0)));
  pTrajMax->operator=(*(pTrajeto1->GetAt(0)));
  v = sqrt(pTrajMin->GetVX(TRUE)*pTrajMin->GetVX(TRUE) + pTrajMin->GetVY(TRUE)*pTrajMin-
>GetVY(TRUE));
  *maxV = v;
  *minV = v;

  for (i=ini;i<fim;i++)
  {
    if (i==0)
      pT = pTrajeto1;
    else
      pT = pTrajeto2;
    for (cur=0;cur<numPt;cur++)
    {
      pTraj = pT->GetAt(cur);
      // V
      v   =   sqrt(pTraj->GetVX(TRUE)*pTraj->GetVX(TRUE)   +   pTraj->GetVY(TRUE)*pTraj-
>GetVY(TRUE));
      if (v > *maxV)
        *maxV = v;
      if (v < *minV)
        *minV = v;
      // X
      v = pTraj->GetX();
      if (v > pTrajMax->GetX())
        pTrajMax->SetX(v);
      if (v < pTrajMin->GetX())
        pTrajMin->SetX(v);
      // Y
      v = pTraj->GetY();
      if (v > pTrajMax->GetY())
        pTrajMax->SetY(v);
      if (v < pTrajMin->GetY())
        pTrajMin->SetY(v);
      // VX
      v = pTraj->GetVX();
      if (v > pTrajMax->GetVX())
        pTrajMax->SetVX(v);
      if (v < pTrajMin->GetVX())
        pTrajMin->SetVX(v);
      // VY
      v = pTraj->GetVY();
      if (v > pTrajMax->GetVY())
        pTrajMax->SetVY(v);
      if (v < pTrajMin->GetVY())
        pTrajMin->SetVY(v);
      // Theta
      v = pTraj->GetTheta();
      if (v > pTrajMax->GetTheta())
        pTrajMax->SetTheta(v);
      if (v < pTrajMin->GetTheta())
        pTrajMin->SetTheta(v);
      // W
      v = pTraj->GetW();
      if (v > pTrajMax->GetW())
        pTrajMax->SetW(v);
      if (v < pTrajMin->GetW())
        pTrajMin->SetW(v);
    }
  }
}

/*************************************************************************
/ ResultadoEmT()                                                        /
/ Interpola no tempo a trajetoria obtida na simulacao para obter o valor  /
/   no tempo pedido                                                     /
/ Parametros:                                                           /
/   t - tempo                                                           /
```

```
/   car - carro desejado (0 p/ 1 ou  1 p/ 2)                                  /
/   pOut - ponto que recebe o resultado                                       /
/ Valor de Retorno: False para tempo negativo ou maior que o maximo         /
 ****************************************************************************/
BOOL FACTDoc::ResultadoEmT(double t, short car, TrajPoint* pOut)
{
  short numPt, cur;
  double T, curT, fr, delta;
  Trajetoria* pT;
  TrajPoint *pTP1, *pTP2;

  if (car==0)
    pT = pTrajeto1;
  if (car==1)
    pT = pTrajeto2;

  // assumimos que o vetor de trajetoria contem os pontos em ordem de tempo
  // assim, o maior tempo esta no ultimo ponto e se o valor de t for maior,
  // nao podemos interpolar.
  // da mesma forma, o tempo comeca em zero, e se t for negativo, a funcao retorna
  numPt = pT->GetSize();
  T = pT->GetAt(numPt-1)->GetT();
  if ((t > T)||(t < 0))
    return FALSE;

  // se for o primeiro ponto, simplesmente copia
  if (t == 0.)
  {
    *pOut = *(pT->GetAt(0));
    return TRUE;
  }
  // senao, busca o intervalo de interpolacao adequado
  cur = 0;
  curT = pT->GetAt(0)->GetT();
  while ((curT<t)&&(cur<numPt))
  {
    cur++;
    curT = pT->GetAt(cur)->GetT();
  }
  // faz interpolacao com base no tempo
  pTP1 = pT->GetAt(cur-1);
  pTP2 = pT->GetAt(cur);
  T = pTP1->GetT();
  delta = curT-T;
  fr = (t-T)/delta;
  pOut->SetT(t);
  delta = pTP2->GetX() - pTP1->GetX();
  pOut->SetX(pTP1->GetX() + fr*delta);
  delta = pTP2->GetY() - pTP1->GetY();
  pOut->SetY(pTP1->GetY() + fr*delta);
  delta = pTP2->GetVX() - pTP1->GetVX();
  pOut->SetVX(pTP1->GetVX() + fr*delta);
  delta = pTP2->GetVY() - pTP1->GetVY();
  pOut->SetVY(pTP1->GetVY() + fr*delta);
  delta = pTP2->GetTheta() - pTP1->GetTheta();
  pOut->SetTheta(pTP1->GetTheta() + fr*delta);
  delta = pTP2->GetW() - pTP1->GetW();
  pOut->SetW(pTP1->GetW() + fr*delta);
  return TRUE;
}

BOOL FACTDoc::DoEvents()
{
  MSG msg;          // Mensagem
  BOOL ret = TRUE;  // Valor de retorno - Falso quando cancel é apertado

  while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
  {
    // detecta cancel
    if(msg.message == BN_CLICKED)
      ret = FALSE;

    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }

  return ret;
}

// factview.h : interface of the FACTView class
```

```
//
/////////////////////////////////////////////////////////////////////////////

#include "mtview.h"

#define PRINT_MAX_PAGES 15

class FACTView : public MyTabView
{
protected: // create from serialization only
  FACTView();
  DECLARE_DYNCREATE(FACTView)

// Attributes
public:
  FACTDoc* GetDocument();
protected:
  // controle da impressao
  BOOL bPrintCar[2];
  BOOL bPrintCrush[2];
  BOOL bPrintCen;
  BOOL bPrintTraj;
  BOOL bPrintGraph[7];
  short nPrintWhat[PRINT_MAX_PAGES];
  // auxiliares da impressao
  CPen *oldPen, *redPen, *bluePen, *dotPen;
  CBrush *redBrush, *blueBrush;
  CFont *oldFon;

// Operations
public:
  void OnChangeTab(UINT* cur, UINT* last);

// Implementation
public:
  virtual ~FACTView();
  virtual void OnDraw(CDC* pDC);  // overridden to draw this view
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

protected:
  // Printing support
  virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
  virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
  virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
  virtual void OnPrint( CDC* pDC, CPrintInfo* pInfo);

  // Funcoes auxiliares de impressao
  void ImprimeDados(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nCar);
  void ImprimePerfil(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nCar);
  void ImprimeCenario(FACTDoc* pDoc, CDC* pDC, CRect& figArea);
  void DesenhaTrajetorias(FACTDoc* pDoc, CDC* pDC, CRect& figArea, double step);
  void DesenhaGrafico(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nGraph);
  void ImprimeCabecalho(CDC* pDC, CRect& figArea, short nPage);
  void ImprimeLegenda(CDC* pDC, CRect& figArea);
  void CalcPontos(CPoint pontos[4], short ox, short oy, double xf, double xr, double ys,
double theta, double scale);
  void DesenhaCarro(CDC* dc, CPoint pontos[4]);
  void GraphPontos(FACTDoc* pDoc, short nGraph, double T, short nCar, double* x, double*
y);

// Generated message map functions
protected:
  //{{AFX_MSG(FACTView)
  afx_msg void OnFileSave();
  afx_msg void OnFileSaveAs();
  afx_msg void OnFilePrintSetup();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

public:
  virtual void OnInitialUpdate();

};

#ifndef _DEBUG  // debug version in factview.cpp
inline FACTDoc* FACTView::GetDocument()
   { return (FACTDoc*)m_pDocument; }
```

```
#endif

/////////////////////////////////////////////////////////////////////////////

// factview.cpp : implementation of the FACTView class
//

#include "stdafx.h"
#include "fact.h"
#include "cardata.h"
#include "crush.h"
#include "trajeto.h"

#include "factdoc.h"
#include "factview.h"

#include "autodlg.h"
#include "crushdlg.h"
#include "inipdlg.h"
#include "trjdlg.h"
#include "printdlg.h"

#include "status.h"

#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

#ifndef DEGTORAD
  #define DEGTORAD    0.01745329
  #define RADTODEG   57.29578000
  #define PI          3.14159265358979
#endif  //DEGTORAD

/////////////////////////////////////////////////////////////////////////////
// FACTView

IMPLEMENT_DYNCREATE(FACTView, MyTabView)

BEGIN_MESSAGE_MAP(FACTView, MyTabView)
  //{{AFX_MSG_MAP(FACTView)
  ON_COMMAND(ID_FILE_SAVE, OnFileSave)
  ON_COMMAND(ID_FILE_SAVE_AS, OnFileSaveAs)
  ON_COMMAND(ID_FILE_PRINT_SETUP, OnFilePrintSetup)
  //}}AFX_MSG_MAP
  // Standard printing commands
  ON_COMMAND(ID_FILE_PRINT, MyTabView::OnFilePrint)
  ON_COMMAND(ID_FILE_PRINT_PREVIEW, MyTabView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// FACTView construction/destruction

FACTView::FACTView()
{
  for (short i=0; i<2; i++)
  {
    bPrintCar[i] = TRUE;
    bPrintCrush[i] = TRUE;
  }
  bPrintCen = TRUE;
  bPrintTraj = TRUE;
  for (i=0; i<7; i++)
    bPrintGraph[i] = TRUE;
}

FACTView::~FACTView()
{
}

/////////////////////////////////////////////////////////////////////////////
// FACTView drawing

void FACTView::OnDraw(CDC* pDC)
{
  FACTDoc* pDoc = GetDocument();
  ASSERT_VALID(pDoc);
```

```
  MyTabView::OnDraw(pDC);
}

////////////////////////////////////////////////////////////////////////////
// FACTView printing

BOOL FACTView::OnPreparePrinting(CPrintInfo* pInfo)
{
  FACTDoc* pDoc = (FACTDoc*)GetDocument();
  short i, n;
  // verifica o que precisa ser impresso
  // obs: codigos de impressao:
  //   0 = dados carro 1
  //   1 = dados carro 2
  //   2 = crush carro 1
  //   3 = crush carro 2
  //   4 = cenario
  //   5 = trajetoria
  //   6 = grafico Velocidade
  //   7 = grafico Vx
  //   8 = grafico Vy
  //   9 = grafico PosX
  //  10 = grafico PosY
  //  11 = grafico W
  //  12 = grafico Theta

  for (i=0; i<PRINT_MAX_PAGES; i++)
    nPrintWhat[i]=0;
  n=0;
  for (i=0; i<2; i++)
    if (bPrintCar[i])
      nPrintWhat[n++] = i;
  for (i=0; i<2; i++)
    if (bPrintCrush[i])
      nPrintWhat[n++] = i+2;
  if (bPrintCen)
    nPrintWhat[n++] = 4;
  // so permite impressao das saidas caso
  // a simulacao ja tenha sido executada.
  if (pDoc->GetExecFlag())
  {
    if (bPrintTraj)
      nPrintWhat[n++] = 5;
    for (i=0; i<7; i++)
      nPrintWhat[n++] = i+6;
  }
  pInfo->SetMinPage(1);
  pInfo->SetMaxPage(n);

  return DoPreparePrinting(pInfo);
}

void FACTView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
  short nColors = pDC->GetDeviceCaps(NUMCOLORS);
  int pStyle = (nColors>2)?PS_SOLID:PS_DOT;

  // Verifica se estamos no print preview
  ((FACTApp*)AfxGetApp())->isppreview = pInfo->m_bPreview;

  // cria as pens para usar no desenho
  redPen = new CPen(PS_SOLID,1,RGB(255,0,0));
  bluePen = new CPen(pStyle,1,RGB(0,0,255));
  dotPen = new CPen(PS_DOT,1,RGB(0,0,0));
  // cria as brushes
  redBrush = new CBrush(RGB(255,0,0));
  blueBrush = new CBrush(RGB(0,0,255));
  // seleciona uma caneta padrao e guarda a antiga
  oldPen = (CPen*)pDC->SelectStockObject(BLACK_PEN);
  // faz o mesmo com a fonte
  oldFon = (CFont*)pDC->SelectStockObject(SYSTEM_FONT);
  pDC->SetMapMode(MM_TEXT);
  pDC->SetBkMode(TRANSPARENT);
}

void FACTView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
  // Volta o status para "nao estamos mais no print preview"
  ((FACTApp*)AfxGetApp())->isppreview = FALSE;
```

```
  pDC->SelectStockObject(BLACK_PEN);
  pDC->SelectStockObject(SYSTEM_FONT);
  delete redPen;
  delete bluePen;
  delete dotPen;
  delete redBrush;
  delete blueBrush;
}

void FACTView::OnPrint( CDC* pDC, CPrintInfo* pInfo )
{
  FACTDoc* pDoc = (FACTDoc*)GetDocument();
  TrajetoDlg* pAd6 = (TrajetoDlg*)(GetChild(5)->pSelf);
  double Passo = pAd6->Passo;
  short n = nPrintWhat[pInfo->m_nCurPage - 1];
  ImprimeCabecalho(pDC, pInfo->m_rectDraw, pInfo->m_nCurPage);
  switch (n)
  {
    case 0:
      ImprimeDados(pDoc, pDC, pInfo->m_rectDraw, 0);
      break;
    case 1:
      ImprimeDados(pDoc, pDC, pInfo->m_rectDraw, 1);
      break;
    case 2:
      ImprimePerfil(pDoc, pDC, pInfo->m_rectDraw, 0);
      break;
    case 3:
      ImprimePerfil(pDoc, pDC, pInfo->m_rectDraw, 1);
      break;
    case 4:
      ImprimeCenario(pDoc, pDC, pInfo->m_rectDraw);
      break;
    case 5:
      ImprimeLegenda(pDC, pInfo->m_rectDraw);
      DesenhaTrajetorias(pDoc, pDC, pInfo->m_rectDraw, Passo/1000);
      break;
    default:
      if ((n>5)&&(n<13))
      {
        ImprimeLegenda(pDC, pInfo->m_rectDraw);
        DesenhaGrafico(pDoc, pDC, pInfo->m_rectDraw, n-6);
      }
  }
}

// Funcoes auxiliares de impressao

void FACTView::ImprimeDados(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nCar)
{
  CString s;
  char buf[100];
  double temp;
  CSize sizeFon;
  short x, y, w, h;
  CarData* pCar = new CarData;

  y = figArea.top;
  x = figArea.left;
  w = figArea.Width();
  h = figArea.Height();
  s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  sizeFon = pDC->GetTextExtent(s, s.GetLength());
  sizeFon.cx /= s.GetLength();
  sizeFon.cy += 2;
  if (nCar == 0)
    pDoc->GetCar1(pCar);
  else
    pDoc->GetCar2(pCar);

  pDC->SelectStockObject(BLACK_PEN);
  // Titulo
  sprintf(buf,"%2d", nCar+1);
  s = "Dados do Veículo ";
  s = s + buf;
  pDC->SetTextAlign(TA_TOP|TA_CENTER);
  pDC->TextOut(x + w/2, y, s);
  y += 2*sizeFon.cy;
  // Nome do veículo
```

```
pDC->SetTextAlign(TA_TOP|TA_LEFT);
s = "Nome: ";
s += pCar->GetName();
pDC->TextOut(x+2*sizeFon.cx, y, s);
y += 2*sizeFon.cy;
// Parâmetros de Inércia
s = "Parâmetros de Inércia:";
pDC->TextOut(x, y, s);
y += 2*sizeFon.cy;
s = "Massa: ";
temp = pCar->GetMassa();
sprintf(buf, "%4.2lf", temp);
s += buf;
s += " kg";
pDC->TextOut(x+2*sizeFon.cx, y, s);
y += sizeFon.cy;
s = "Raio de Giração ao quadrado: ";
temp = pCar->GetRaioGirQ();
sprintf(buf, "%4.2lf", temp);
s += buf;
s += " m²";
pDC->TextOut(x+2*sizeFon.cx, y, s);
y += 2*sizeFon.cy;
// Parâmetros de Rigidez
s = "Parâmetros de Rigidez:";
pDC->TextOut(x, y, s);
y += 2*sizeFon.cy;
s = "Frontal";
pDC->TextOut(x+5*sizeFon.cx, y, s);
s = "Lateral";
pDC->TextOut(x+20*sizeFon.cx, y, s);
s = "Traseiro";
pDC->TextOut(x+35*sizeFon.cx, y, s);
// traça linhas para a tabela
for (short i=0; i<3; i++)
{
  pDC->MoveTo(x+(4 + i*15)*sizeFon.cx, y);
  pDC->LineTo(x+(4 + i*15)*sizeFon.cx, y+4*sizeFon.cy);
}
y += sizeFon.cy;
//// A
s = "A ";
pDC->TextOut(x+2*sizeFon.cx, y, s);
pDC->SetTextAlign(TA_TOP|TA_RIGHT);
for (i=0; i<3; i++)
{
  temp = pCar->GetA(i);
  sprintf(buf,"%6.2lf",temp);
  s = buf;
  pDC->TextOut(x+(18 + 15*i)*sizeFon.cx, y, s);
}
s = "N/m²";
pDC->SetTextAlign(TA_TOP|TA_LEFT);
pDC->TextOut(x+50*sizeFon.cx, y, s);
y += sizeFon.cy;
//// B
s = "B ";
pDC->TextOut(x+2*sizeFon.cx, y, s);
pDC->SetTextAlign(TA_TOP|TA_RIGHT);
for (i=0; i<3; i++)
{
  temp = pCar->GetB(i);
  sprintf(buf,"%6.2lf",temp);
  s = buf;
  pDC->TextOut(x+(18 + 15*i)*sizeFon.cx, y, s);
}
s = "N/m²";
pDC->SetTextAlign(TA_TOP|TA_LEFT);
pDC->TextOut(x+50*sizeFon.cx, y, s);
y += sizeFon.cy;
//// G
s = "G ";
pDC->TextOut(x+2*sizeFon.cx, y, s);
pDC->SetTextAlign(TA_TOP|TA_RIGHT);
for (i=0; i<3; i++)
{
  temp = pCar->GetG(i);
  sprintf(buf,"%6.2lf",temp);
  s = buf;
  pDC->TextOut(x+(18 + 15*i)*sizeFon.cx, y, s);
```

```
  }
  s = "kg";
  pDC->SetTextAlign(TA_TOP|TA_LEFT);
  pDC->TextOut(x+50*sizeFon.cx, y, s);
  y += 2*sizeFon.cy;
  // Parâmetros Geométricos
  pDC->SetTextAlign(TA_TOP|TA_LEFT);
  s = "Parâmetros Geométricos:";
  pDC->TextOut(x, y, s);
  y += 2*sizeFon.cy;
  s = "XF: ";
  temp = pCar->GetXF();
  sprintf(buf, "%4.2lf", temp);
  s += buf;
  s += " m";
  pDC->TextOut(x+2*sizeFon.cx, y, s);
  s = "XR: ";
  temp = pCar->GetXR();
  sprintf(buf, "%4.2lf", temp);
  s += buf;
  s += " m";
  pDC->TextOut(x+(22*sizeFon.cx), y, s);
  y += sizeFon.cy;
  s = "EF: ";
  temp = pCar->GetEF();
  sprintf(buf, "%4.2lf", temp);
  s += buf;
  s += " m";
  pDC->TextOut(x+2*sizeFon.cx, y, s);
  s = "ER: ";
  temp = pCar->GetER();
  sprintf(buf, "%4.2lf", temp);
  s += buf;
  s += " m";
  pDC->TextOut(x+(22*sizeFon.cx), y, s);
  y += sizeFon.cy;
  s = "YS: ";
  temp = pCar->GetYS();
  sprintf(buf, "%4.2lf", temp);
  s += buf;
  s += " m";
  pDC->TextOut(x+2*sizeFon.cx, y, s);
  s = "T:  ";
  temp = pCar->GetTrack();
  sprintf(buf, "%4.2lf", temp);
  s += buf;
  s += " m";
  pDC->TextOut(x+(22*sizeFon.cx), y, s);
  y += sizeFon.cy;

  delete pCar;
}

void FACTView::ImprimePerfil(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nCar)
{
  CString s;
  char buf[100];
  double temp;
  CSize sizeFon;
  short x, y, w, h, i;
  CrushArray* pCrush = new CrushArray;

  y = figArea.top;
  x = figArea.left;
  w = figArea.Width();
  h = figArea.Height();
  s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  sizeFon = pDC->GetTextExtent(s, s.GetLength());
  sizeFon.cx /= s.GetLength();
  sizeFon.cy += 2;
  if (nCar == 0)
    pDoc->GetCrush1(pCrush);
  else
    pDoc->GetCrush2(pCrush);

  pDC->SelectStockObject(BLACK_PEN);
  // Título
  sprintf(buf,"%2d", nCar+1);
  s = "Perfil de Deformação do Veículo ";
  s = s + buf;
```

```
    pDC->SetTextAlign(TA_TOP|TA_CENTER);
    pDC->TextOut(x + w/2, y, s);
    y += 2*sizeFon.cy;
    pDC->SetTextAlign(TA_TOP|TA_LEFT);
    // Tipo de colisao e origem
    short nTipo = (nCar==0)?(pDoc->GetTipo1()):(pDoc->GetTipo2());
    s = "Tipo de Colisão: ";
    switch(nTipo)
    {
      case 0:
        s += "Frontal";
        break;
      case 1:
        s += "Traseira";
        break;
      case 2:
        s += "Lateral Esquerda";
        break;
      case 3:
        s += "Lateral Direita";
        break;
    }
    pDC->TextOut(x, y, s);
    y += 2*sizeFon.cy;
    s = "Perfil de Deformação:";
    pDC->TextOut(x, y, s);
    y += 2*sizeFon.cy;
    temp = (nCar==0)?(pDoc->GetOcx1()):(pDoc->GetOcx2());
    temp *= 1000;
    s = "Origem X: ";
    sprintf(buf, "%4.2lf", temp);
    s += buf;
    s += " mm";
    pDC->TextOut(x + 2*sizeFon.cx, y, s);
    y += sizeFon.cy;
    temp = (nCar==0)?(pDoc->GetOcy1()):(pDoc->GetOcy2());
    temp *= 1000;
    s = "Origem Y: ";
    sprintf(buf, "%4.2lf", temp);
    s += buf;
    s += " mm";
    pDC->TextOut(x + 2*sizeFon.cx, y, s);
    y += 2*sizeFon.cy;
    // Tabela do Perfil
    short nPt = pCrush->GetSize();
    s = "Pontos do Perfil:";
    pDC->TextOut(x + 2*sizeFon.cx, y, s);
    y += 2*sizeFon.cy;
    s = "Posição (mm)";
    pDC->TextOut(x+2*sizeFon.cx, y, s);
    s = "Profundidade (mm)";
    pDC->TextOut(x+20*sizeFon.cx, y, s);
    y += sizeFon.cy;
    // traça linhas para a tabela
    pDC->MoveTo(x+19*sizeFon.cx, y);
    pDC->LineTo(x+19*sizeFon.cx, y+nPt*sizeFon.cy);
    pDC->SetTextAlign(TA_TOP|TA_RIGHT);
    for (i = 0; i<nPt; i++)
    {
      temp = pCrush->GetAt(i)->GetPos();
      sprintf(buf,"%6.2lf",temp);
      s = buf;
      pDC->TextOut(x+18*sizeFon.cx, y, s);
      temp = pCrush->GetAt(i)->GetProf();
      sprintf(buf,"%6.2lf",temp);
      s = buf;
      pDC->TextOut(x+38*sizeFon.cx, y, s);
      y += sizeFon.cy;
    }
}

void FACTView::ImprimeCenario(FACTDoc* pDoc, CDC* pDC, CRect& figArea)
{
  CString s;
  char buf[100];
  double temp;
  CSize sizeFon;
  short x, y, w, h;

  y = figArea.top;
```

```
x = figArea.left;
w = figArea.Width();
h = figArea.Height();
s = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
sizeFon = pDC->GetTextExtent(s, s.GetLength());
sizeFon.cx /= s.GetLength();
sizeFon.cy += 2;

pDC->SelectStockObject(BLACK_PEN);
// Título
s = "Condições Iniciais dos Veículos (Cenário)";
pDC->SetTextAlign(TA_TOP|TA_CENTER);
pDC->TextOut(x + w/2, y, s);
y += 2*sizeFon.cy;
pDC->SetTextAlign(TA_TOP|TA_LEFT);
// Veiculo 1
s = "Veículo 1";
pDC->TextOut(x, y, s);
y += 2*sizeFon.cy;
//// posicao
temp = pDoc->GetX01();
sprintf(buf, "%4.2lf", temp);
s = "Posição X: ";
s += buf;
s += " m";
pDC->TextOut(x + 2*sizeFon.cx, y, s);
y += sizeFon.cy;
temp = pDoc->GetY01();
sprintf(buf, "%4.2lf", temp);
s = "Posição Y: ";
s += buf;
s += " m";
pDC->TextOut(x + 2*sizeFon.cx, y, s);
y += sizeFon.cy;
//// velocidade  e angulo
temp = pDoc->GetV01();
sprintf(buf, "%4.2lf", temp*3.6);
s = "Velocidade: ";
s += buf;
s += " km/h";
pDC->TextOut(x + 2*sizeFon.cx, y, s);
y += sizeFon.cy;
temp = pDoc->GetTheta01();
sprintf(buf, "%4.2lf", temp);
s = "Ângulo: ";
s += buf;
s += " graus";
pDC->TextOut(x + 2*sizeFon.cx, y, s);
y += 2*sizeFon.cy;
// Veiculo 2
s = "Veículo 2";
pDC->TextOut(x, y, s);
y += 2*sizeFon.cy;
//// posicao
temp = pDoc->GetX02();
sprintf(buf, "%4.2lf", temp);
s = "Posição X: ";
s += buf;
s += " m";
pDC->TextOut(x + 2*sizeFon.cx, y, s);
y += sizeFon.cy;
temp = pDoc->GetY02();
sprintf(buf, "%4.2lf", temp);
s = "Posição Y: ";
s += buf;
s += " m";
pDC->TextOut(x + 2*sizeFon.cx, y, s);
y += sizeFon.cy;
//// velocidade  e angulo
temp = pDoc->GetV02();
sprintf(buf, "%4.2lf", temp*3.6);
s = "Velocidade: ";
s += buf;
s += " km/h";
pDC->TextOut(x + 2*sizeFon.cx, y, s);
y += sizeFon.cy;
temp = pDoc->GetTheta02();
sprintf(buf, "%4.2lf", temp);
s = "Ângulo: ";
s += buf;
```

```
  s += " graus";
  pDC->TextOut(x + 2*sizeFon.cx, y, s);
  y += 2*sizeFon.cy;
  // Condicoes adicionais
  s = "Dados Adicionais";
  pDC->TextOut(x, y, s);
  y += 2*sizeFon.cy;
  temp = pDoc->GetAlfa();
  sprintf(buf, "%4.2lf", temp);
  s = "Ângulo da Força: ";
  s += buf;
  s += " graus";
  pDC->TextOut(x + 2*sizeFon.cx, y, s);
  y += sizeFon.cy;
  temp = pDoc->GetMi();
  sprintf(buf, "%4.2lf", temp);
  s = "Coeficiente de Atrito: ";
  s += buf;
  pDC->TextOut(x + 2*sizeFon.cx, y, s);
  y += sizeFon.cy;
}

/*************************************************************************
/ DesenhaTrajetorias()                                                   /
/ Esta funcao e usada para desenhar as trajetorias dos veiculos ao longo /
/ da simulacao, usando para isso dados obtidos do objeto FACTDoc.        /
/ Parametros:                                                            /
/   pDoc - ponteiro para o objeto FACTDoc que contem os dados da simulacao /
/   pDC - ponteiro para o device context onde se deve desenhar           /
/   figArea - referencia a um objeto CRect que delimita a area de desenho /
/   step - passo de tempo entre os quadros da trajetoria, em segundos    /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTView::DesenhaTrajetorias(FACTDoc* pDoc, CDC* pDC, CRect& figArea, double step)
{
  CarData* pCar = new CarData;
  TrajPoint* pTraj = new TrajPoint;
  double T, maxT;
  short ox, oy;
  short carOx1, carOy1;
  short carOx2, carOy2;
  CPoint pontos[4];
  double l1, l2, l1r, l2r, y1, y2;
  double w, h, dd, dd2;
  double xmax,ymax,xmin,ymin;
  double scale,sc2;

  // obtem os dados dos carros
  pDoc->GetCar1(pCar);
  l1r = pCar->GetXR();
  l1 =  pCar->GetXF();
  y1 =  pCar->GetYS();
  pDoc->GetCar2(pCar);
  l2r = pCar->GetXR();
  l2 =  pCar->GetXF();
  y2 =  pCar->GetYS();

  /* desenha um titulo (so na impressao) */
  pDC->SetTextAlign(TA_CENTER|TA_TOP);
  pDC->SelectStockObject(BLACK_PEN);
  CString s = "Trajetórias dos Veículos";
  CSize sizeFon = pDC->GetTextExtent(s, s.GetLength());
  pDC->TextOut(figArea.left + figArea.Width()/2, figArea.top, s);
  figArea.top += 2*sizeFon.cy;

  /* calcula uma escala correta */
  dd = sqrt((l1+l1r)*(l1+l1r)+y1*y1);
  dd2 = sqrt((l2+l2r)*(l2+l2r)+y2*y2);
  if (dd2>dd)
    dd=dd2;
  dd = dd*0.75;
  pDoc->GetMaxRect(&xmin,&ymin,&xmax,&ymax);
  xmin -= dd;
  ymin -= dd;
  xmax += dd;
  ymax += dd;
  w = xmax - xmin;
  h = ymax - ymin;
  scale = ((double)figArea.right-(double)figArea.left)/w;
  sc2 = ((double)figArea.bottom-(double)figArea.top)/h;
```

```
    if (scale>sc2)
      scale = sc2;

    maxT = pDoc->GetLastT();
    ox = figArea.left + (short)(fabs(xmin)*scale);
    oy = figArea.top + (short)(fabs(ymax)*scale);
    /* desenha as trajetorias dos carros 1 e 2 */
    T = 0;
    pDC->SelectObject(redPen);
    pDoc->ResultadoEmT(T,0,pTraj);
    carOx1 = ox + (short)(scale*pTraj->GetX());
    carOy1 = oy - (short)(scale*pTraj->GetY());
    pDC->FillRect(CRect(carOx1-2, carOy1-2, carOx1+3, carOy1+3), redBrush);
    // desenha o carro na posicao inicial...
    if (1 /*bDrawCar1*/)
    {
      CalcPontos(pontos, carOx1, carOy1, l1, l1r, y1, pTraj->GetTheta(), scale);
      DesenhaCarro(pDC, pontos);
    }
    /* desenha a trajetoria do carro 2 */
    T = 0;
    pDC->SelectObject(bluePen);
    pDoc->ResultadoEmT(T,1,pTraj);
    carOx2 = ox + (short)(scale*pTraj->GetX());
    carOy2 = oy - (short)(scale*pTraj->GetY());
    pDC->FillRect(CRect(carOx2-2, carOy2-2, carOx2+3, carOy2+3), blueBrush);
    // desenha o carro na posicao inicial...
    if (1 /*bDrawCar2*/)
    {
      CalcPontos(pontos, carOx2, carOy2, l2, l2r, y2, pTraj->GetTheta(), scale);
      DesenhaCarro(pDC, pontos);
    }
    while (T<maxT)
    {
      pDC->SelectObject(redPen);
      pDC->MoveTo(carOx1, carOy1);
      T += step;
      if (T>maxT)
        T = maxT;
      pDoc->ResultadoEmT(T,0,pTraj);
      carOx1 = ox + (short)(scale*pTraj->GetX());
      carOy1 = oy - (short)(scale*pTraj->GetY());
      pDC->LineTo(carOx1, carOy1);
      if (1/*bDrawCar1*/)
      {
        CalcPontos(pontos, carOx1, carOy1, l1, l1r, y1, pTraj->GetTheta(), scale);
        DesenhaCarro(pDC, pontos);
      }
      pDC->SelectObject(bluePen);
      pDC->MoveTo(carOx2, carOy2);
      pDoc->ResultadoEmT(T,1,pTraj);
      carOx2 = ox + (short)(scale*pTraj->GetX());
      carOy2 = oy - (short)(scale*pTraj->GetY());
      pDC->LineTo(carOx2, carOy2);
      if (1/*bDrawCar2*/)
      {
        CalcPontos(pontos, carOx2, carOy2, l2, l2r, y2, pTraj->GetTheta(), scale);
        DesenhaCarro(pDC, pontos);
      }
    }

    delete pTraj;
    delete pCar;
}

/*************************************************************************
/ DesenhaGrafico()                                                       /
/ Esta funcao e utilizada para plotar um dos graficos de saida na janela /
/ de trajetoria.                                                         /
/ Parametros:                                                            /
/   pDoc - ponteiro para o objeto FACTDoc que contem os dados da simulacao /
/   pDC - ponteiro para o device context onde se deve desenhar           /
/   figArea - referencia a um objeto CRect que delimita a area de desenho /
/   nGraph - inteiro entre 0 e 6, identifica qual o grafico a ser plotado /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTView::DesenhaGrafico(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nGraph)
{
// desenha um gráfico não-proporcional em que
// aparecem os dados de um ou dos dois carros
```

```
//
// os valores para nGraph são:
// 0 = V  x t
// 1 = Vx x t
// 2 = Vy x t
// 3 = X  x t
// 4 = Y  x t
// 5 = W  x t
// 6 = Theta x t
//
  TrajPoint cTraj, minTraj, maxTraj;
  double minV, maxV;
  double deltax, deltay, scalex, scaley;
  double ox, oy;
  double x1, y1, x2, y2;
  double T, maxT, step;
  double eX, eY;
  CString sPotX, sPotY;
  double stepX = 1, stepY = 1;
  short maxTicksX, maxTicksY;
  char buf[20];
  CString title;
  CSize fSize;
  short flagCars;

  // obtem o tempo final e calcula um passo de interpolacao
  maxT = pDoc->GetLastT();
  step = maxT/75.;

  // obtem os limites de todas as grandezas do grafico
  flagCars = 3 /*((bGraphCar1)?1:0) + ((bGraphCar2)?2:0)*/;
  minV = 0;
  maxV = 0;
  pDoc->GetMinMaxTraj(&minTraj, &minV, &maxTraj, &maxV, flagCars);

  // calcula os deltas para escala adequadamente, conforme o gráfico a ser plotado
  deltax = maxT;
  ox = 0;
  sPotX = "s";
  switch (nGraph)
  {
    case 0:
      title = "Velocidade x t";
      sPotY = "km/h";
      oy = minV;
      deltay = maxV - minV;
      break;
    case 1:
      title = "Velocidade(X) x t";
      sPotY = "km/h";
      oy = minTraj.GetVX(TRUE);
      deltay = maxTraj.GetVX(TRUE) - oy;
      break;
    case 2:
      title = "Velocidade(Y) x t";
      sPotY = "km/h";
      oy = minTraj.GetVY(TRUE);
      deltay = maxTraj.GetVY(TRUE) - oy;
      break;
    case 3:
      title = "Posição(X) x t";
      sPotY = "m";
      oy = minTraj.GetX();
      deltay = maxTraj.GetX() - oy;
      break;
    case 4:
      title = "Posição(Y) x t";
      sPotY = "m";
      oy = minTraj.GetY();
      deltay = maxTraj.GetY() - oy;
      break;
    case 5:
      title = "Velocidade Angular x t";
      sPotY = "°/s";
      oy = minTraj.GetW(TRUE);
      deltay = maxTraj.GetW(TRUE) - oy;
      break;
    case 6:
      title = "Ângulo x t";
      sPotY = "graus";
```

```
      oy = minTraj.GetTheta(TRUE);
      deltay = maxTraj.GetTheta(TRUE) - oy;
      break;
  }

  // evita divisao por zero
  if (deltax < 1e-6)
    deltax = 1;
  if (deltay < 1e-6)
    deltay = 1;

  // calcula a largura media dos caracteres, ponderando
  // um pouco para cima usando caracteres largos
  pDC->SelectStockObject(BLACK_PEN);
  fSize = pDC->GetOutputTextExtent("0123456789AOMW",14);
  fSize.cx/= 14;
  pDC->SetTextAlign(TA_CENTER|TA_TOP);
  pDC->TextOut(figArea.left+figArea.Width()/2,figArea.top,title);
  // reduz a area de plotagem descontando o espaco usado para
  // escala e titulo
  figArea.top += 2*fSize.cy;
  figArea.left += 4*fSize.cx;
  figArea.bottom -= 2*fSize.cy;
  figArea.right -= 2*fSize.cx;
  pDC->MoveTo(figArea.left, figArea.bottom);
  pDC->LineTo(figArea.right, figArea.bottom);
  pDC->MoveTo(figArea.left, figArea.bottom);
  pDC->LineTo(figArea.left, figArea.top);

  // calcula as escalas
  short w = figArea.Width();
  short h = figArea.Height();
  scalex = ((double)figArea.Width())/deltax;
  scaley = ((double)figArea.Height())/deltay;

  // calcula o numero maximo de ticks
  maxTicksX = figArea.Width()/(4*fSize.cx);
  maxTicksY = figArea.Height()/(2*fSize.cy);

  // calcula os expoentes dos eixos
  eX = (short)log10(__max(fabs(ox), fabs(ox+deltax))) - 1;
  eY = (short)log10(__max(fabs(oy), fabs(oy+deltay))) - 1;

  // calcula os deltas, que sao inicialmente usados como o numero de ticks.
  // se eles forem muito poucos, aumenta por um fator de 10
  deltax /= pow(10,eX);
  if (deltax <= 3)
  {
    deltax *= 10;
    eX--;
  }
  deltay /= pow(10,eY);
  if (deltay <= 3)
  {
    deltay *= 10;
    eY--;
  }

  // verifica a escala (1, 2, 5 e 10) que sera usada
  stepX = pow(10,eX);
  if (deltax > maxTicksX)
    if (deltax > 2*maxTicksX)
      if (deltax > 5*maxTicksX)
        stepX = pow(10,eX+1);
      else
        stepX = 5*pow(10,eX);
    else
      stepX = 2*pow(10,eX);
  stepY = pow(10,eY);
  if (deltay > maxTicksY)
    if (deltay > 2*maxTicksY)
      if (deltay > 5*maxTicksY)
        stepY = pow(10,eY+1);
      else
        stepY = 5*pow(10,eY);
    else
      stepY = 2*pow(10,eY);

  deltax *= pow(10,eX);
  deltay *= pow(10,eY);
```

```
// calcula os valores minimos a serem escritos nos ticks
double minVer, minHor;
if (ox <= 0)
  minHor = ox - fmod(ox,stepX);
else
  minHor = ox + stepX - fmod(ox,stepX);
if (oy <= 0)
  minVer = oy - fmod(oy,stepY);
else
  minVer = oy + stepY - fmod(oy,stepY);

// calcula as posicoes iniciais dos ticks
short X, Y;
X = (short)((minHor - ox)*scalex);
Y = (short)((minVer - oy)*scaley);

// plota os ticks horizontais
pDC->SetTextAlign(TA_CENTER|TA_TOP);
while (X <= figArea.Width())
{
  pDC->MoveTo(figArea.left+X, figArea.bottom);
  pDC->LineTo(figArea.left+X, figArea.bottom+5);
  sprintf(buf,"%2.0lf",minHor/pow(10,eX));
  pDC->TextOut(figArea.left+X, figArea.bottom+6,buf,lstrlen(buf));
  minHor += stepX;
  X = (short)((minHor - ox)*scalex);
}

// plota os ticks verticais
pDC->SetTextAlign(TA_RIGHT|TA_BASELINE);
while (Y <= figArea.Height())
{
  pDC->MoveTo(figArea.left, figArea.bottom-Y);
  pDC->LineTo(figArea.left-5, figArea.bottom-Y);
  sprintf(buf,"%2.0lf",minVer/pow(10,eY));
  pDC->TextOut(figArea.left-6, figArea.bottom-Y,buf,lstrlen(buf));
  minVer += stepY;
  Y = (short)((minVer - oy)*scaley);
}

// escreve as potencias de 10 e unidades
pDC->SetTextAlign(TA_RIGHT|TA_TOP);
if (eX)
{
  sprintf(buf,"x1E%2.0lf ",eX);
  sPotX = buf + sPotX;
}
pDC->TextOut(figArea.right + 2*fSize.cx, figArea.bottom + 6 + fSize.cy, sPotX);
pDC->SetTextAlign(TA_LEFT|TA_BOTTOM);
if (eY)
{
  sprintf(buf,"x1E%2.0lf ",eY);
  sPotY = buf + sPotY;
}
pDC->TextOut(figArea.left + 2*fSize.cx, figArea.top - 4, sPotY);

// desenha a linha do zero e a linha do final da colisao
pDC->SelectObject(dotPen);
double tend = pDoc->GetTEndCol();
pDC->MoveTo(figArea.left + (short)(scalex*tend), figArea.bottom);
pDC->LineTo(figArea.left + (short)(scalex*tend), figArea.top);
if ((oy<0)&&(oy+deltay>=0))
{
  pDC->MoveTo(figArea.left, figArea.bottom + (short)(scaley*oy));
  pDC->LineTo(figArea.right, figArea.bottom + (short)(scaley*oy));
}
pDC->SelectStockObject(BLACK_PEN);

T = 0;
if (1/*bGraphCar1*/)
{
  GraphPontos(pDoc, nGraph, T, 0, &x1, &y1);
  x1 = x1 - ox;
  y1 = y1 - oy;
}
if (1/*bGraphCar2*/)
{
  GraphPontos(pDoc, nGraph, T, 1, &x2, &y2);
  x2 = x2 - ox;
```

```
      y2 = y2 - oy;
    }
    while (T < maxT)
    {
      T += step;
      if (T>maxT)
        T = maxT;
      // obtem os dados para o carro 1
      if (1/*bGraphCar1*/)
      {
        pDC->MoveTo(figArea.left      +      (short)(scalex*x1),      figArea.bottom      -
(short)(scaley*y1));
        pDC->SelectObject(redPen);
        GraphPontos(pDoc, nGraph, T, 0, &x1, &y1);
        x1 = x1 - ox;
        y1 = y1 - oy;
        pDC->LineTo(figArea.left      +      (short)(scalex*x1),      figArea.bottom      -
(short)(scaley*y1));
      }
      // obtem os dados para o carro 2
      if (1/*bGraphCar2*/)
      {
        pDC->MoveTo(figArea.left      +      (short)(scalex*x2),      figArea.bottom      -
(short)(scaley*y2));
        pDoc->ResultadoEmT(T,1,&cTraj);
        pDC->SelectObject(bluePen);
        GraphPontos(pDoc, nGraph, T, 1, &x2, &y2);
        x2 = x2 - ox;
        y2 = y2 - oy;
        pDC->LineTo(figArea.left      +      (short)(scalex*x2),      figArea.bottom      -
(short)(scaley*y2));
      }
    }
}

void FACTView::ImprimeCabecalho(CDC* pDC, CRect& figArea, short nPage)
{
  CString s;
  char buf[20];
  CSize sizeFon;
  pDC->SetTextAlign(TA_TOP|TA_CENTER);
  s = "FACT - Ferramenta de Análise de Colisões de Trânsito";
  sizeFon = pDC->GetTextExtent(s, s.GetLength());
  sizeFon.cx /= s.GetLength();
  pDC->TextOut(figArea.left + figArea.Width()/2, figArea.top, s);
  s = "página ";
  sprintf(buf,"%2d", nPage);
  s = s + buf;
  pDC->SetTextAlign(TA_BOTTOM|TA_CENTER);
  pDC->TextOut(figArea.left + figArea.Width()/2, figArea.bottom, s);
  // reduz a área de impressão
  figArea.top += 2*sizeFon.cy;
  figArea.bottom -= 2*sizeFon.cy;
  figArea.left += 2*sizeFon.cx;
  figArea.right -= 2*sizeFon.cx;
}

void FACTView::ImprimeLegenda(CDC* pDC, CRect& figArea)
{
  CString s;
  CSize sizeFon;
  // obtem o tamanho do quadro de legenda
  s = "VEICULO XX ";
  sizeFon = pDC->GetTextExtent(s, s.GetLength());
  pDC->SelectStockObject(BLACK_PEN);
  pDC->Rectangle(figArea.right - 2*sizeFon.cx - 2, figArea.bottom - 5*sizeFon.cy - 2,
figArea.right, figArea.bottom);
  pDC->SetTextAlign(TA_TOP|TA_CENTER);
  // escreve o titulo e outros textos
  s = "LEGENDA:";
  pDC->TextOut(figArea.right - sizeFon.cx, figArea.bottom - 4*sizeFon.cy, s);
  s = "VEÍCULO 1 ";
  pDC->SetTextAlign(TA_TOP|TA_RIGHT);
  pDC->TextOut(figArea.right - sizeFon.cx, figArea.bottom - 3*sizeFon.cy, s);
  s = "VEÍCULO 2 ";
  pDC->TextOut(figArea.right - sizeFon.cx, figArea.bottom - 2*sizeFon.cy, s);
  // traca a linha do carro 1
  pDC->SelectObject(redPen);
  pDC->MoveTo(figArea.right - sizeFon.cx, figArea.bottom - (5*sizeFon.cy/2));
```

```
  pDC->LineTo(figArea.right     -     sizeFon.cx/s.GetLength(),     figArea.bottom     -
(5*sizeFon.cy/2));
  // traca a linha do carro 2
  pDC->SelectObject(bluePen);
  pDC->MoveTo(figArea.right - sizeFon.cx, figArea.bottom - (3*sizeFon.cy/2));
  pDC->LineTo(figArea.right     -     sizeFon.cx/s.GetLength(),     figArea.bottom     -
(3*sizeFon.cy/2));
  // desconta a area da legenda
  figArea.bottom -= 6*sizeFon.cy;
}

/*************************************************************************
/ CalcPontos()                                                           /
/ Esta e uma funcao auxiliar de DesenhaTrajetorias que calcula as coorde- /
/ nadas dos vertices do retangulo que representa um dos veiculos.        /
/ Parametros:                                                            /
/   pontos - vetor de objetos CPoint que sera usado para armazenar as    /
/  coordenadas dos vertices                                              /
/   ox, oy - coordenadas de origem da tela                               /
/   xf, xr, ys - dimensoes do veiculo                                    /
/   theta - orientacao angular do veiculo                                /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTView::CalcPontos(CPoint pontos[4], short ox, short oy, double xf, double xr,
double ys, double theta, double scale)
{
  pontos[0].x = ox + (int)((ys*sin(theta)+xf*cos(theta))*scale);
  pontos[0].y = oy - (int)((xf*sin(theta)-ys*cos(theta))*scale);
  pontos[1].x = ox + (int)((-xr*cos(theta)+ys*sin(theta))*scale);
  pontos[1].y = oy - (int)((-ys*cos(theta)-xr*sin(theta))*scale);
  pontos[2].x = ox + (int)((-xr*cos(theta)-ys*sin(theta))*scale);
  pontos[2].y = oy - (int)((-xr*sin(theta)+ys*cos(theta))*scale);
  pontos[3].x = ox + (int)((xf*cos(theta)-ys*sin(theta))*scale);
  pontos[3].y = oy - (int)((xf*sin(theta)+ys*cos(theta))*scale);
}

/*************************************************************************
/ DesenhaCarro()                                                         /
/ Esta e uma funcao auxiliar de DesenhaTrajetorias que recebe os pontos  /
/ dos vertices e desenha o veiculo.                                      /
/ Parametros:                                                            /
/   dc - ponteiro do device context onde se deve desenhar                /
/   pontos - vetor dos vertices do retangulo                             /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTView::DesenhaCarro(CDC* dc, CPoint pontos[4])
{
  dc->MoveTo(pontos[0]);
  dc->LineTo(pontos[1]);
  dc->LineTo(pontos[2]);
  dc->LineTo(pontos[3]);
  dc->LineTo(pontos[0]);
}

/*************************************************************************
/ GraphPontos()                                                          /
/ Esta e uma funcao auxiliar de DesenhaGrafico que retorna o ponto a ser  /
/ plotado conforme o grafico, usando uma funcao de interpolacao de       /
/ FACTDoc                                                                /
/ Parametros:                                                            /
/   pDoc - ponteiro para o objeto FACTDoc                                /
/   nGraph - tipo de grafico sendo plotado                               /
/   T - tempo no qual o ponto deve ser interpolado                       /
/   nCar - 0 para veiculo 1 e 1 para veiculo 2                           /
/   x, y - ponteiros para os valores de retorno do ponto interpolado     /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTView::GraphPontos(FACTDoc* pDoc, short nGraph, double T, short nCar, double* x,
double* y)
{
  TrajPoint cTraj;

  pDoc->ResultadoEmT(T,nCar,&cTraj);
  *x = T;
  switch (nGraph)
  {
    case 0:
      *y = sqrt(pow(cTraj.GetVX(TRUE),2) + pow(cTraj.GetVY(TRUE),2));
      break;
    case 1:
```

```
      *y = cTraj.GetVX(TRUE);
      break;
    case 2:
      *y = cTraj.GetVY(TRUE);
      break;
    case 3:
      *y = cTraj.GetX();
      break;
    case 4:
      *y = cTraj.GetY();
      break;
    case 5:
      *y = cTraj.GetW(TRUE);
      break;
    case 6:
      *y = cTraj.GetTheta(TRUE);
      break;
  }
}

/////////////////////////////////////////////////////////////////////////////
// FACTView diagnostics

#ifdef _DEBUG
void FACTView::AssertValid() const
{
  MyTabView::AssertValid();
}

void FACTView::Dump(CDumpContext& dc) const
{
  MyTabView::Dump(dc);
}

FACTDoc* FACTView::GetDocument() // non-debug version is inline
{
  ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(FACTDoc)));
  return (FACTDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// FACTView message handlers

/*************************************************************************
/ OnInitialUpdate()                                                      /
/ Chamada antes da view ser mostrada pela primeira vez. Cria os tabs e os  /
/   adiciona a lista de tabs                                             /
/ Parametros: -                                                          /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTView::OnInitialUpdate()
{
  FACTDoc* pFDoc = (FACTDoc*)GetDocument();
  CarData* pCar = new CarData();
  pFDoc->GetCar1(pCar);

  if(!HowMany)
  {
    AutoDlg* pAD = new AutoDlg(this);
    AddChild(pAD, "Auto 1", 0, 0, 0);
    AddChild(pAD, "Auto 2", 0, 0, 0);

    CrushDlg* pCD = new CrushDlg(this);
    AddChild(pCD, "Deformação 1", 0, 0, 0);
    AddChild(pCD, "Deformação 2", 0, 0, 0);

    InitPosDlg* pIPD = new InitPosDlg(this);
    AddChild(pIPD, "Cenário", 0, 0, 0);

    TrajetoDlg* pTD = new TrajetoDlg(this);
    AddChild(pTD, "Trajetória", 0, 0, 0);

    pAD->UpdateCarData(FALSE, pCar);
    pAD->UpdateData(FALSE);
    ((CListBox*)(pAD->GetDlgItem(IDC_LSTAUTOS)))->SelectString(0,pCar->GetName());

  }
  // Se as views ja estao criadas, so limpa
  else
```

```
  {
    GetChild(Cur)->pSelf->ShowWindow(SW_HIDE);
    Cur = 0;
    AutoDlg* pAd1 = (AutoDlg*)(GetChild(0)->pSelf);
    pAd1->UpdateCarData(FALSE, pCar);
    pAd1->UpdateData(FALSE);
    ((CListBox*)(pAd1->GetDlgItem(IDC_LSTAUTOS)))->SelectString(0,pCar->GetName());
    Invalidate();
  }

  delete pCar;
  ((AutoDlg*)(GetChild(0)->pSelf))->GetDlgItem(IDC_NOME)->SetFocus();
}

/*************************************************************************
/ OnChangeTab()                                                          /
/ Chamada sempre antes que haja uma troca do tab exibido. Atualiza os    /
/   dados dos tabs                                                       /
/ Parametros:                                                            /
/   last - indice da janela que estava sendo exibida                     /
/   cur - indice da janela que vai passar a ser exibida (pode ser mudado /
/     aqui, mudando a janela a ser exibida)                              /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void FACTView::OnChangeTab(UINT* cur, UINT* last)
{
  FACTDoc* pFDoc = (FACTDoc*)GetDocument();
  CarData* pCar = new CarData();
  CrushArray* pCrush = new CrushArray();
  Crush* teste;

  AutoDlg* pAd1 = (AutoDlg*)(GetChild(0)->pSelf);
  AutoDlg* pAd2 = (AutoDlg*)(GetChild(1)->pSelf);
  CrushDlg* pAd3 = (CrushDlg*)(GetChild(2)->pSelf);
  CrushDlg* pAd4 = (CrushDlg*)(GetChild(3)->pSelf);
  InitPosDlg* pAd5 = (InitPosDlg*)(GetChild(4)->pSelf);
  TrajetoDlg* pAd6 = (TrajetoDlg*)(GetChild(5)->pSelf);

  if(*last == 0)
  {
    pAd1->UpdateData(TRUE);
    pAd1->UpdateCarData(TRUE, pCar);
    pFDoc->SetCar1(pAd1->mudou, pCar);
    switch(pFDoc->GetTipo1())
    {
      case 0:
        pFDoc->SetOcx1(pCar->GetXF());
      break;
      case 1:
        pFDoc->SetOcx1(-(pCar->GetXR()));
      break;
      case 2:
        pFDoc->SetOcy1(pCar->GetYS());
      break;
      case 3:
        pFDoc->SetOcy1(-(pCar->GetYS()));
      break;
    }
  }

  if(*last == 1)
  {
    pAd2->UpdateData(TRUE);
    pAd2->UpdateCarData(TRUE, pCar);
    pFDoc->SetCar2(pAd2->mudou, pCar);
    switch(pFDoc->GetTipo2())
    {
      case 0:
        pFDoc->SetOcx2(pCar->GetXF());
      break;
      case 1:
        pFDoc->SetOcx2(-(pCar->GetXR()));
      break;
      case 2:
        pFDoc->SetOcy2(pCar->GetYS());
      break;
      case 3:
        pFDoc->SetOcy2(-(pCar->GetYS()));
      break;
    }
```

```
}

if(*last == 2)
{
  pAd3->UpdateData(TRUE);
  pFDoc->SetCrush1(pAd3->mudou, pAd3->pCrush);
  pFDoc->SetTipo1(pAd3->tipo);
  pFDoc->SetOcx1(pAd3->m_x0 /1000);
  pFDoc->SetOcy1(pAd3->m_y0 /1000);
}

if(*last == 3)
{
  pAd4->UpdateData(TRUE);
  pFDoc->SetCrush2(pAd4->mudou, pAd4->pCrush);
  pFDoc->SetTipo2(pAd4->tipo);
  pFDoc->SetOcx2(pAd4->m_x0 /1000);
  pFDoc->SetOcy2(pAd4->m_y0 /1000);
}

if(*last == 4)
{
  pAd5->UpdateData(TRUE);
  pFDoc->SetX01(pAd5->m_X01);
  pFDoc->SetX02(pAd5->m_X02);
  pFDoc->SetY01(pAd5->m_Y01);
  pFDoc->SetY02(pAd5->m_Y02);
  pFDoc->SetTheta01(pAd5->m_teta01);
  pFDoc->SetTheta02(pAd5->m_teta02);
  pFDoc->SetV01((pAd5->m_V01)/3.6);
  pFDoc->SetV02((pAd5->m_V02)/3.6);
  pFDoc->SetAlfa(pAd5->m_alfa);
  pFDoc->SetMi(pAd5->m_mi);
  pAd6->Passo = pAd5->m_Passo;
}

if(*cur == 0)
{
  pFDoc->GetCar1(pCar);
  pAd1->UpdateCarData(FALSE, pCar);
  pAd1->UpdateData(FALSE);
  pAd1->GetDlgItem(IDC_NOME)->SetFocus();
  pAd1->mudou = FALSE;
  ((CListBox*)(pAd1->GetDlgItem(IDC_LSTAUTOS)))->SelectString(0,pCar->GetName());
}

if(*cur == 1)
{
  pFDoc->GetCar2(pCar);
  pAd2->UpdateCarData(FALSE, pCar);
  pAd2->UpdateData(FALSE);
  pAd2->GetDlgItem(IDC_NOME)->SetFocus();
  pAd2->mudou = FALSE;
  ((CListBox*)(pAd2->GetDlgItem(IDC_LSTAUTOS)))->SelectString(0,pCar->GetName());
}

if(*cur == 2)
{
  pAd3->tipo = pFDoc->GetTipo1();
  if(pAd3->tipo < 2)
  {
    pAd3->GetDlgItem(IDC_C_X0)->EnableWindow(FALSE);
    pAd3->GetDlgItem(IDC_C_Y0)->EnableWindow(TRUE);
  }
  else
  {
    pAd3->GetDlgItem(IDC_C_X0)->EnableWindow(TRUE);
    pAd3->GetDlgItem(IDC_C_Y0)->EnableWindow(FALSE);
  }
  pAd3->mudou = FALSE;
  pFDoc->GetCrush1(pCrush);

  if(pCrush->GetSize())
    teste = pCrush->GetAt(0);

  pAd3->m_x0 = pFDoc->GetOcx1() *1000;
  pAd3->m_y0 = pFDoc->GetOcy1() *1000;
  pAd3->m_pos = 0;
  pAd3->m_prof = 0;
  pAd3->UpdateCrushProfile(pCrush);
```

```
      pAd3->pCrush->DeleteAll();
      int n = pCrush->GetSize();
      for(int i = 0; i < n; i++)
        pAd3->pCrush->Add(new Crush(*(pCrush->GetAt(i))));
      pAd3->UpdateTipo();
      pAd3->UpdateData(FALSE);

      pAd3->GetDlgItem(IDC_C_POS)->SetFocus();
    }

    if(*cur == 3)
    {
      pAd4->tipo = pFDoc->GetTipo2();
      if(pAd4->tipo < 2)
      {
        pAd4->GetDlgItem(IDC_C_X0)->EnableWindow(FALSE);
        pAd4->GetDlgItem(IDC_C_Y0)->EnableWindow(TRUE);
      }
      else
      {
        pAd4->GetDlgItem(IDC_C_X0)->EnableWindow(TRUE);
        pAd4->GetDlgItem(IDC_C_Y0)->EnableWindow(FALSE);
      }
      pAd4->mudou = FALSE;
      pFDoc->GetCrush2(pCrush);
      pAd4->m_x0 = pFDoc->GetOcx2() *1000;
      pAd4->m_y0 = pFDoc->GetOcy2() *1000;
      pAd4->m_pos = 0;
      pAd4->m_prof = 0;
      pAd4->UpdateCrushProfile(pCrush);
      pAd4->pCrush->DeleteAll();
      int n = pCrush->GetSize();
      for(int i = 0; i < n; i++)
        pAd4->pCrush->Add(new Crush(*(pCrush->GetAt(i))));
      pAd4->UpdateTipo();
      pAd4->UpdateData(FALSE);

      pAd4->GetDlgItem(IDC_C_POS)->SetFocus();
    }

    if(*cur == 4)
    {
      pAd5->m_X01 = pFDoc->GetX01();
      pAd5->m_X02 = pFDoc->GetX02();
      pAd5->m_Y01 = pFDoc->GetY01();
      pAd5->m_Y02 = pFDoc->GetY02();
      pAd5->m_teta01 = pFDoc->GetTheta01();
      pAd5->m_teta02 = pFDoc->GetTheta02();
      pAd5->m_V01 = 3.6*(pFDoc->GetV01());
      pAd5->m_V02 = 3.6*(pFDoc->GetV02());
      pAd5->m_alfa = pFDoc->GetAlfa();
      pAd5->m_mi = pFDoc->GetMi();
      pAd5->m_Passo = pAd6->Passo;
      pAd5->UpdateData(FALSE);

      pAd5->GetDlgItem(IDC_X01)->SetFocus();
    }

    if(*cur == 5)
    {
      StatusDlg* pS = new StatusDlg(this);
      GetParent()->EnableWindow(FALSE);

      if(!(pFDoc->TrajetoRK4(pS)))
        *cur = *last;

      GetParent()->EnableWindow(TRUE);
      pS->DestroyWindow();
      delete pS;

      pAd6->GetDlgItem(IDC_OPTTRAJETO)->SetFocus();
    }

    delete pCar;
    delete pCrush;
}

void FACTView::OnFileSave()
{
    FACTDoc* pFDoc = (FACTDoc*)GetDocument();
```

```
  UINT curTab = GetTabIndex();
  UINT cur = 6;

  OnChangeTab(&cur, &curTab);
  pFDoc->OnFileSave();
}

void FACTView::OnFileSaveAs()
{
  FACTDoc* pFDoc = (FACTDoc*)GetDocument();
  UINT curTab = GetTabIndex();
  UINT cur = 6;

  OnChangeTab(&cur, &curTab);
  pFDoc->OnFileSaveAs();
}

void FACTView::OnFilePrintSetup()
{
  MyPrintDlg *prndlg = new MyPrintDlg;
  prndlg->m_PrintCar1 = bPrintCar[0];
  prndlg->m_PrintCar2 = bPrintCar[1];
  prndlg->m_PrintCrush1 = bPrintCrush[0];
  prndlg->m_PrintCrush2 = bPrintCrush[1];
  prndlg->m_PrintCen = bPrintCen;
  prndlg->m_PrintTraj = bPrintTraj;
  prndlg->m_PrintVel = bPrintGraph[0];
  prndlg->m_PrintVX = bPrintGraph[1];
  prndlg->m_PrintVY = bPrintGraph[2];
  prndlg->m_PrintPosX = bPrintGraph[3];
  prndlg->m_PrintPosY = bPrintGraph[4];
  prndlg->m_PrintW = bPrintGraph[5];
  prndlg->m_PrintAng = bPrintGraph[6];

  short result = prndlg->DoModal();
  if (result == IDOK)
  {
    bPrintCar[0] = prndlg->m_PrintCar1;
    bPrintCar[1] = prndlg->m_PrintCar2;
    bPrintCrush[0] = prndlg->m_PrintCrush1;
    bPrintCrush[1] = prndlg->m_PrintCrush2;
    bPrintCen = prndlg->m_PrintCen;
    bPrintTraj = prndlg->m_PrintTraj;
    bPrintGraph[0] = prndlg->m_PrintVel;
    bPrintGraph[1] = prndlg->m_PrintVX;
    bPrintGraph[2] = prndlg->m_PrintVY;
    bPrintGraph[3] = prndlg->m_PrintPosX;
    bPrintGraph[4] = prndlg->m_PrintPosY;
    bPrintGraph[5] = prndlg->m_PrintW;
    bPrintGraph[6] = prndlg->m_PrintAng;
  }
  delete prndlg;
}

// autodlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// AutoDlg dialog

class AutoDlg : public CDialog
{
// definições de acesso
friend class NewAutoDlg;
friend class FACTView;

// Construction
public:
  AutoDlg(CWnd* pParent = NULL);
  ~AutoDlg();

  DECLARE_DYNAMIC(AutoDlg)

  BOOL mudou;

// Dialog Data
  //{{AFX_DATA(AutoDlg)
  enum { IDD = IDD_AUTO };
  double  m_A1;
  double  m_A2;
```

```
    double  m_A3;
    double  m_B1;
    double  m_B2;
    double  m_B3;
    double  m_EF;
    double  m_ER;
    double  m_G1;
    double  m_G2;
    double  m_G3;
    double  m_M;
    double  m_RGQ;
    double  m_XF;
    double  m_XR;
    double  m_YS;
    double  m_T;
    CString m_Nome;
    //}}AFX_DATA

// Implementation
public:
    BOOL PreTranslateMessage(MSG*);

protected:
    CBitmapButton m_cotas;

    class FACTDoc* pDoc;

    class CarData;
    void UpdateCarData(BOOL, CarData*);
    virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

    // Generated message map functions
    //{{AFX_MSG(AutoDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnAdd();
    afx_msg void OnErase();
    afx_msg void OnNew();
    afx_msg void OnSelchangeLstautos();
    afx_msg void OnChangeEditNome();
    afx_msg void OnChangeA1();
    afx_msg void OnChangeA2();
    afx_msg void OnChangeA3();
    afx_msg void OnChangeB1();
    afx_msg void OnChangeB2();
    afx_msg void OnChangeB3();
    afx_msg void OnChangeEf();
    afx_msg void OnChangeEr();
    afx_msg void OnChangeG1();
    afx_msg void OnChangeG2();
    afx_msg void OnChangeG3();
    afx_msg void OnChangeM();
    afx_msg void OnChangeRgq();
    afx_msg void OnChangeT();
    afx_msg void OnChangeXf();
    afx_msg void OnChangeXr();
    afx_msg void OnChangeYs();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
// NewAutoDlg dialog

class NewAutoDlg : public CDialog
{
// Construction
public:
    NewAutoDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(NewAutoDlg)
    enum { IDD = IDD_new };
    CString m_Nome;
    //}}AFX_DATA

// Implementation
protected:
    virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
    void Clear(void);
```

```
    CWnd* m_pParent;
    short Tipo;

    // Generated message map functions
    //{{AFX_MSG(NewAutoDlg)
    afx_msg void OnRadio1();
    afx_msg void OnRadio2();
    afx_msg void OnRadio3();
    afx_msg void OnRadio4();
    afx_msg void OnRadio5();
    virtual void OnOK();
    virtual BOOL OnInitDialog();
    afx_msg void OnRadio6();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// autodlg.cpp : implementation file
//

#include "stdafx.h"
#include "fact.h"
#include "cardata.h"
#include "crush.h"

#include "factdoc.h"
#include "factview.h"
#include "autodlg.h"
#include "cardata.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// AutoDlg dialog

IMPLEMENT_DYNAMIC(AutoDlg, CDialog)

AutoDlg::AutoDlg(CWnd* pParent /*=NULL*/)
    : CDialog()
{
    Create(AutoDlg::IDD, pParent);

    //{{AFX_DATA_INIT(AutoDlg)
    m_A1 = 0;
    m_A2 = 0;
    m_A3 = 0;
    m_B1 = 0;
    m_B2 = 0;
    m_B3 = 0;
    m_EF = 0;
    m_ER = 0;
    m_G1 = 0;
    m_G2 = 0;
    m_G3 = 0;
    m_M = 0;
    m_RGQ = 0;
    m_XF = 0;
    m_XR = 0;
    m_YS = 0;
    m_T = 0;
    m_Nome = "";
    //}}AFX_DATA_INIT

    UpdateData(FALSE);
    mudou = FALSE;
}

AutoDlg::~AutoDlg()
{
}

void AutoDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(AutoDlg)
    DDX_Text(pDX, IDC_A1, m_A1);
```

```
    DDX_Text(pDX, IDC_A2, m_A2);
    DDX_Text(pDX, IDC_A3, m_A3);
    DDX_Text(pDX, IDC_B1, m_B1);
    DDX_Text(pDX, IDC_B2, m_B2);
    DDX_Text(pDX, IDC_B3, m_B3);
    DDX_Text(pDX, IDC_EF, m_EF);
    DDX_Text(pDX, IDC_ER, m_ER);
    DDX_Text(pDX, IDC_G1, m_G1);
    DDX_Text(pDX, IDC_G2, m_G2);
    DDX_Text(pDX, IDC_G3, m_G3);
    DDX_Text(pDX, IDC_M, m_M);
    DDX_Text(pDX, IDC_RGQ, m_RGQ);
    DDX_Text(pDX, IDC_XF, m_XF);
    DDX_Text(pDX, IDC_XR, m_XR);
    DDX_Text(pDX, IDC_YS, m_YS);
    DDX_Text(pDX, IDC_T, m_T);
    DDX_Text(pDX, IDC_NOME, m_Nome);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(AutoDlg, CDialog)
    //{{AFX_MSG_MAP(AutoDlg)
    ON_BN_CLICKED(IDC_ADD, OnAdd)
    ON_BN_CLICKED(IDC_ERASE, OnErase)
    ON_BN_CLICKED(IDC_NEW, OnNew)
    ON_LBN_SELCHANGE(IDC_LSTAUTOS, OnSelchangeLstautos)
    ON_EN_CHANGE(IDC_NOME, OnChangeEditNome)
    ON_EN_CHANGE(IDC_A1, OnChangeA1)
    ON_EN_CHANGE(IDC_A2, OnChangeA2)
    ON_EN_CHANGE(IDC_A3, OnChangeA3)
    ON_EN_CHANGE(IDC_B1, OnChangeB1)
    ON_EN_CHANGE(IDC_B2, OnChangeB2)
    ON_EN_CHANGE(IDC_B3, OnChangeB3)
    ON_EN_CHANGE(IDC_EF, OnChangeEf)
    ON_EN_CHANGE(IDC_ER, OnChangeEr)
    ON_EN_CHANGE(IDC_G1, OnChangeG1)
    ON_EN_CHANGE(IDC_G2, OnChangeG2)
    ON_EN_CHANGE(IDC_G3, OnChangeG3)
    ON_EN_CHANGE(IDC_M, OnChangeM)
    ON_EN_CHANGE(IDC_RGQ, OnChangeRgq)
    ON_EN_CHANGE(IDC_T, OnChangeT)
    ON_EN_CHANGE(IDC_XF, OnChangeXf)
    ON_EN_CHANGE(IDC_XR, OnChangeXr)
    ON_EN_CHANGE(IDC_YS, OnChangeYs)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

void AutoDlg::UpdateCarData(BOOL bSave, CarData* pCar)
{
    if(bSave)
    {
        pCar->SetName(m_Nome);
        pCar->SetA(0, m_A1);
        pCar->SetA(1, m_A2);
        pCar->SetA(2, m_A3);
        pCar->SetB(0, m_B1);
        pCar->SetB(1, m_B2);
        pCar->SetB(2, m_B3);
        pCar->SetG(0, m_G1);
        pCar->SetG(1, m_G2);
        pCar->SetG(2, m_G3);
        pCar->SetTrack(m_T);
        pCar->SetEF(m_EF);
        pCar->SetER(m_ER);
        pCar->SetRaioGirQ(m_RGQ);
        pCar->SetMassa(m_M);
        pCar->SetXR(m_XR);
        pCar->SetXF(m_XF);
        pCar->SetYS(m_YS);
    }
    else
    {
        m_Nome = pCar->GetName();
        m_A1 = pCar->GetA(0);
        m_A2 = pCar->GetA(1);
        m_A3 = pCar->GetA(2);
        m_B1 = pCar->GetB(0);
        m_B2 = pCar->GetB(1);
        m_B3 = pCar->GetB(2);
        m_G1 = pCar->GetG(0);
```

```
    m_G2 = pCar->GetG(1);
    m_G3 = pCar->GetG(2);
    m_T = pCar->GetTrack();
    m_EF = pCar->GetEF();
    m_ER = pCar->GetER();
    m_RGQ = pCar->GetRaioGirQ();
    m_M = pCar->GetMassa();
    m_XR = pCar->GetXR();
    m_XF = pCar->GetXF();
    m_YS = pCar->GetYS();
  }
}

BOOL AutoDlg::PreTranslateMessage(MSG* pMsg)
{
  // na mensagem KEYDOWN, evita a criacao de um WM_CHAR quando a tecla
  // pressionada for TAB (sem CTRL)
  if (pMsg->message == WM_KEYDOWN)
  {
    if ((pMsg->wParam == VK_TAB)&&(!(GetKeyState(VK_CONTROL)&0xFF00)))
      return 1;
  }

  // na mensagem KEYUP, trata os casos corretos
  if (pMsg->message == WM_KEYUP)
  {
    // se for TAB
    if (pMsg->wParam == VK_TAB)
    {
      // e CTRL, manda para a janela parent
      if (GetKeyState(VK_CONTROL)&0xFF00)
        return 0;
      // senao, muda o foco para o proximo controle
      else
      {
        NextDlgCtrl();
        return 1;
      }
    }
  }

  return(CWnd::PreTranslateMessage(pMsg));
}

/////////////////////////////////////////////////////////////////////////////
// AutoDlg message handlers

BOOL AutoDlg::OnInitDialog()
{
  CDialog::OnInitDialog();
  BOOL ret = m_cotas.AutoLoad(IDC_COTAS, this);
  ASSERT(ret);

  pDoc = (FACTDoc*)((CView*)GetParent())->GetDocument();
  long n = pDoc->GetCDB()->GetNumElements();
  CListBox* pList = (CListBox*)GetDlgItem(IDC_LSTAUTOS);

  for(long i = 0; i < n; i++)
    pList->InsertString((int)i, pDoc->GetCDB()->GetAt(i)->GetName());

  return TRUE;
}

void AutoDlg::OnAdd()
{
  long index = (long)((CListBox*)GetDlgItem(IDC_LSTAUTOS))->GetCurSel();
  UpdateData(TRUE);
  UpdateCarData(TRUE, pDoc->GetCDB()->GetAt(index));
  pDoc->SaveDatabase();
}

void AutoDlg::OnErase()
{
  CListBox* pList = (CListBox*)GetDlgItem(IDC_LSTAUTOS);

  if(pList->GetCount() == 1)
  {
    MessageBox("O último item não pode ser apagado", "FACT", MB_OK|MB_ICONEXCLAMATION);
    return;
  }
```

```
  if(MessageBox("Tem certeza que deseja remover esse veículo do banco de dados?", "FACT
- Confirmação", MB_YESNO|MB_ICONQUESTION) == IDYES)
  {
    long index = (long)pList->GetCurSel();
    pList->DeleteString((int)index);
    pDoc->GetCDB()->DeleteCar(index);
    pList->SetCurSel(0);
    OnSelchangeLstautos();
  }
  pDoc->SaveDatabase();
}

void AutoDlg::OnNew()
{
  NewAutoDlg NAD(this);
  short Tipo = (short)NAD.DoModal();
  // Os tipos sao retornados negativos, de forma
  // a nao confundir com IDCANCEL que e 2.
  // Assim, se o Tipo for maior que zero, e invalido
  if (Tipo >0)
    return;

  // Caso contrario, reverte para que fique positivo...
  Tipo = -Tipo;

  switch(Tipo)
  {
    case 0:
      m_A1 = 0;
      m_A2 = 0;
      m_A3 = 0;
      m_B1 = 0;
      m_B2 = 0;
      m_B3 = 0;
      m_G1 = 0;
      m_G2 = 0;
      m_G3 = 0;
      m_T = 0;
      m_EF = 0;
      m_ER = 0;
      m_RGQ = 0;
      m_M = 0;
      m_XR = 0;
      m_XF = 0;
      m_YS = 0;
    break;

    case 1:
      m_A1 = 52889;
      m_A2 = 64098;
      m_A3 = 13485;
      m_B1 = 324054;
      m_B2 = 262001;
      m_B3 = 255106;
      m_G1 = 4303;
      m_G2 = 7810;
      m_G3 = 360;
      m_T = 1.298;
      m_EF = 1.146;
      m_ER = 1.222;
      m_RGQ = 1.29;
      m_M = 998;
      m_XR = 2.129;
      m_XF = 1.930;
      m_YS = 0.772;
    break;

    case 2:
      m_A1 = 45359;
      m_A2 = 68476;
      m_A3 = 24518;
      m_B1 = 296475;
      m_B2 = 282685;
      m_B3 = 461949;
      m_G1 = 3462;
      m_G2 = 8339;
      m_G3 = 659;
      m_T = 1.387;
      m_EF = 1.176;
      m_ER = 1.273;
```

```
        m_RGQ = 1.90;
        m_M = 1383;
        m_XR = 2.327;
        m_XF = 2.116;
        m_YS = 0.853;
      break;

      case 3:
        m_A1 = 55516;
        m_A2 = 71803;
        m_A3 = 30297;
        m_B1 = 386107;
        m_B2 = 303369;
        m_B3 = 393001;
        m_G1 = 4009;
        m_G2 = 8593;
        m_G3 = 1170;
        m_T = 1.496;
        m_EF = 1.303;
        m_ER = 1.410;
        m_RGQ = 2.14;
        m_M = 1607;
        m_XR = 2.703;
        m_XF = 2.281;
        m_YS = 0.922;
      break;

      case 4:
        m_A1 = 62346;
        m_A2 = 62521;
        m_A3 = 25044;
        m_B1 = 234422;
        m_B2 = 89632;
        m_B3 = 344738;
        m_G1 = 8339;
        m_G2 = 22188;
        m_G3 = 903;
        m_T = 1.570;
        m_EF = 1.389;
        m_ER = 1.504;
        m_RGQ = 2.41;
        m_M = 1923;
        m_XR = 2.896;
        m_XF = 2.510;
        m_YS = 0.978;
      break;

      case 5:
        m_A1 = 56917;
        m_A2 = 52014;
        m_A3 = 30998;
        m_B1 = 255106;
        m_B2 = 482633;
        m_B3 = 324054;
        m_G1 = 6359;
        m_G2 = 2759;
        m_G3 = 1473;
        m_T = 1.618;
        m_EF = 1.476;
        m_ER = 1.600;
        m_RGQ = 2.61;
        m_M = 2203;
        m_XR = 3.096;
        m_XF = 2.586;
        m_YS = 1.013;
      break;
    }

    CarData* pCar = new CarData;
    UpdateCarData(TRUE, pCar);
    long index = pDoc->GetCDB()->AddCar(pCar);
    CListBox* pList = (CListBox*)GetDlgItem(IDC_LSTAUTOS);
    pList->InsertString((int)index, m_Nome);
    pList->SetCurSel((int)index);
    UpdateData(FALSE);
    pDoc->SaveDatabase();
}

void AutoDlg::OnSelchangeLstautos()
{
```

```
  long index = (long)((CListBox*)GetDlgItem(IDC_LSTAUTOS))->GetCurSel();
  UpdateCarData(FALSE, pDoc->GetCDB()->GetAt(index));
  UpdateData(FALSE);
  mudou = TRUE;
}

void AutoDlg::OnChangeEditNome()
{
  CEdit* edCtrl = (CEdit*)GetDlgItem(IDC_NOME);
  // se a edit box nao tem o foco, nao interessa
  // o fato de seu conteudo ter mudado
  if ((CWnd*)edCtrl != GetFocus())
    return;
  // caso contrario, faz uma busca pela lista de carros
  CListBox* cLB = (CListBox*)GetDlgItem(IDC_LSTAUTOS);
  char buffer[100];
  short l = edCtrl->GetLine(0,buffer,99);
  // se a edit box esta vazia, nao tenta procurar na lista
  if (l==0)
    return;
  buffer[l] = 0;
  long index = cLB->GetCurSel();
  long indnew = cLB->SelectString(0,buffer);
  // se mudou a selecao, entao da um update
  if ((indnew != LB_ERR)&&(index != indnew))
  {
    UpdateCarData(FALSE, pDoc->GetCDB()->GetAt(indnew));
    UpdateData(FALSE);
    long nl = edCtrl->LineLength(0);
    if (l<nl)
      edCtrl->SetSel((int)l,(int)nl);
  }
}

void AutoDlg::OnChangeA1()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeA2()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeA3()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeB1()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeB2()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeB3()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeEf()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
```

```
  mudou = TRUE;
}

void AutoDlg::OnChangeEr()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeG1()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeG2()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeG3()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeM()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeRgq()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeT()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeXf()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeXr()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

void AutoDlg::OnChangeYs()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  pDoc->SetModifiedFlag(TRUE);
  mudou = TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// NewAutoDlg dialog


NewAutoDlg::NewAutoDlg(CWnd* pParent /*=NULL*/)
  : CDialog(NewAutoDlg::IDD, pParent)
{
```

```
  m_pParent = pParent;

  //{{AFX_DATA_INIT(NewAutoDlg)
  m_Nome = "";
  //}}AFX_DATA_INIT
}

void NewAutoDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(NewAutoDlg)
  DDX_Text(pDX, IDC_NOME, m_Nome);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(NewAutoDlg, CDialog)
  //{{AFX_MSG_MAP(NewAutoDlg)
  ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
  ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
  ON_BN_CLICKED(IDC_RADIO3, OnRadio3)
  ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
  ON_BN_CLICKED(IDC_RADIO5, OnRadio5)
  ON_BN_CLICKED(IDC_RADIO6, OnRadio6)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

void NewAutoDlg::Clear()
{
  ((CButton*)GetDlgItem(IDC_RADIO1))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_RADIO2))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_RADIO3))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_RADIO4))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_RADIO5))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_RADIO6))->SetCheck(0);
}

/////////////////////////////////////////////////////////////////////////////
// NewAutoDlg message handlers

void NewAutoDlg::OnRadio1()
{
  Clear();
  ((CButton*)GetDlgItem(IDC_RADIO1))->SetCheck(1);
  Tipo = 0;
}

void NewAutoDlg::OnRadio2()
{
  Clear();
  ((CButton*)GetDlgItem(IDC_RADIO2))->SetCheck(1);
  Tipo = 1;
}

void NewAutoDlg::OnRadio3()
{
  Clear();
  ((CButton*)GetDlgItem(IDC_RADIO3))->SetCheck(1);
  Tipo = 2;
}

void NewAutoDlg::OnRadio4()
{
  Clear();
  ((CButton*)GetDlgItem(IDC_RADIO4))->SetCheck(1);
  Tipo = 3;
}

void NewAutoDlg::OnRadio5()
{
  Clear();
  ((CButton*)GetDlgItem(IDC_RADIO5))->SetCheck(1);
  Tipo = 4;
}

void NewAutoDlg::OnRadio6()
{
  Clear();
  ((CButton*)GetDlgItem(IDC_RADIO6))->SetCheck(1);
  Tipo = 5;
}
```

```
void NewAutoDlg::OnOK()
{
  UpdateData(TRUE);
  ((AutoDlg*)m_pParent)->m_Nome = m_Nome;
  EndDialog((int)-Tipo);
}

BOOL NewAutoDlg::OnInitDialog()
{
  CDialog::OnInitDialog();

  Tipo = 0;
  ((CButton*)GetDlgItem(IDC_RADIO1))->SetCheck(1);
  GetDlgItem(IDC_NOME)->SetFocus();

  return FALSE;
}

// crushdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CrushDlg dialog

class CrushDlg : public CDialog
{
// Construction
public:
  CrushDlg(CWnd* pParent = NULL); // standard constructor
  ~CrushDlg();

  DECLARE_DYNAMIC(CrushDlg)

// Dialog Data
  //{{AFX_DATA(CrushDlg)
  enum { IDD = IDD_CRUSH };
  CStatic m_Figura;
  CListBox  m_Perfil;
  double  m_pos;
  double  m_prof;
  double  m_x0;
  double  m_y0;
  //}}AFX_DATA

public:
  UINT tipo;
  BOOL mudou;

  virtual  BOOL OnInitDialog( );
  void UpdateTipo();
  void UpdateCrushProfile(CrushArray* ca);

  double round(double);
  BOOL Consis(double c);

  CrushArray* pCrush;

// Implementation
public:
  BOOL PreTranslateMessage(MSG*);

protected:
  virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

  // Generated message map functions
  //{{AFX_MSG(CrushDlg)
  afx_msg void OnCDelete();
  afx_msg void OnCInsert();
  afx_msg void OnPaint();
  afx_msg void OnTipof();
  afx_msg void OnTipold();
  afx_msg void OnTipole();
  afx_msg void OnTipot();
  afx_msg void OnSelchangeCPerfil();
  afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
  afx_msg void OnChangeCX0();
  afx_msg void OnChangeCY0();
  afx_msg int OnVKeyToItem(UINT nKey, CListBox* pListBox, UINT nIndex);
  //}}AFX_MSG
```

```
  DECLARE_MESSAGE_MAP()
};

// crushdlg.cpp : implementation file
//

#include "stdafx.h"
#include "fact.h"
#include "cardata.h"
#include "crush.h"
#include "factdoc.h"
#include "factview.h"
#include "crushdlg.h"

#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CrushDlg dialog

IMPLEMENT_DYNAMIC(CrushDlg, CDialog)

CrushDlg::CrushDlg(CWnd* pParent /*=NULL*/)
  : CDialog()
{
  Create(CrushDlg::IDD, pParent);

  //{{AFX_DATA_INIT(CrushDlg)
  //}}AFX_DATA_INIT

  pCrush = new CrushArray;
  mudou = FALSE;
}

CrushDlg::~CrushDlg()
{
  delete pCrush;
}

void CrushDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CrushDlg)
  DDX_Control(pDX, IDC_FIGURA, m_Figura);
  DDX_Control(pDX, IDC_C_PERFIL, m_Perfil);
  DDX_Text(pDX, IDC_C_POS, m_pos);
  DDX_Text(pDX, IDC_C_PROF, m_prof);
  DDX_Text(pDX, IDC_C_X0, m_x0);
  DDX_Text(pDX, IDC_C_Y0, m_y0);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CrushDlg, CDialog)
  //{{AFX_MSG_MAP(CrushDlg)
  ON_BN_CLICKED(IDC_C_DELETE, OnCDelete)
  ON_BN_CLICKED(IDC_C_INSERT, OnCInsert)
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_TIPOF, OnTipof)
  ON_BN_CLICKED(IDC_TIPOLD, OnTipold)
  ON_BN_CLICKED(IDC_TIPOLE, OnTipole)
  ON_BN_CLICKED(IDC_TIPOT, OnTipot)
  ON_LBN_SELCHANGE(IDC_C_PERFIL, OnSelchangeCPerfil)
  ON_WM_LBUTTONUP()
  ON_EN_CHANGE(IDC_C_X0, OnChangeCX0)
  ON_EN_CHANGE(IDC_C_Y0, OnChangeCY0)
  ON_WM_VKEYTOITEM()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()


void CrushDlg::UpdateTipo()
{
  ((CButton*)GetDlgItem(IDC_TIPOF))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_TIPOT))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_TIPOLE))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_TIPOLD))->SetCheck(0);
```

```
    switch (tipo)
    {
      case 0:
        ((CButton*)GetDlgItem(IDC_TIPOF))->SetCheck(1);
      break;

      case 1:
        ((CButton*)GetDlgItem(IDC_TIPOT))->SetCheck(1);
      break;

      case 2:
        ((CButton*)GetDlgItem(IDC_TIPOLE))->SetCheck(1);
      break;

      case 3:
        ((CButton*)GetDlgItem(IDC_TIPOLD))->SetCheck(1);
      break;
    }
}

BOOL CrushDlg::PreTranslateMessage(MSG* pMsg)
{
  // na mensagem KEYDOWN, evita a criacao de um WM_CHAR quando a tecla
  // pressionada for TAB (sem CTRL)
  if (pMsg->message == WM_KEYDOWN)
  {
    if ((pMsg->wParam == VK_TAB)&&(!(GetKeyState(VK_CONTROL)&0xFF00)))
      return 1;
  }

  // na mensagem KEYUP, trata os casos corretos
  if (pMsg->message == WM_KEYUP)
  {
    // se for TAB
    if (pMsg->wParam == VK_TAB)
    {
      // e CTRL, manda para a janela parent
      if (GetKeyState(VK_CONTROL)&0xFF00)
        return 0;
      // senao, muda o foco para o proximo controle
      else
      {
        NextDlgCtrl();
        return 1;
      }
    }
  }

  return(CWnd::PreTranslateMessage(pMsg));
}

////////////////////////////////////////////////////////////////////////////
// CrushDlg message handlers

BOOL CrushDlg::OnInitDialog()
{
  m_pos = 0.;
  m_prof = 0.;
  m_x0 = 0.;
  m_y0 = 0.;

  tipo = 0;

  return CDialog::OnInitDialog();
}

void CrushDlg::OnCDelete()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  short dp;

  // pega o item selecionado; se nao houver, retorna.
  dp = m_Perfil.GetCurSel();
  if (dp == LB_ERR)
    return;

  pCrush->RemoveAt(dp);

  m_pos = 0;
```

```
  m_prof = 0;

  mudou = TRUE;
  pDoc->SetModifiedFlag(TRUE);
  UpdateCrushProfile(pCrush);
  RedrawWindow();
}

void CrushDlg::OnCInsert()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());

  UpdateData(TRUE);

  pCrush->InsertOrd(m_pos, m_prof);

  mudou = TRUE;
  pDoc->SetModifiedFlag(TRUE);
  UpdateCrushProfile(pCrush);
  RedrawWindow();
}

void CrushDlg::OnPaint()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  CarData* pCar = new CarData;
  Crush* pC;

  CRect figArea;
  short ox, oy;
  CPen *cp, *oldCp;
  double w, h, xf;
  double scale, sc2;

  short ct,i;

  CPaintDC dc(this); // device context for painting

  // Código específico: as crush dialogs sabem que
  // indice 2 = crush do carro 1
  // indice 3 = crush do carro 2
  UINT curTab = ((MyTabView*)GetParent())->GetTabIndex();

  if (curTab == 2)
  {
    pDoc->GetCar1(pCar);
    ct = pCrush->GetSize();
    w = (pCar->GetYS()*2);
    h = (pCar->GetXF() + pCar->GetXR());
    xf = pCar->GetXF();
  }
  else
  {
    pDoc->GetCar2(pCar);
    ct = pCrush->GetSize();
    w = (pCar->GetYS()*2);
    h = (pCar->GetXF() + pCar->GetXR());
    xf = pCar->GetXF();
  }

  m_Figura.GetWindowRect(&figArea);
  ScreenToClient(&figArea);
  figArea.left+=20;
  figArea.top+=20;
  figArea.right-=20;
  figArea.bottom-=20;
  scale = ((double)figArea.right-(double)figArea.left)/w;
  sc2 = ((double)figArea.bottom-(double)figArea.top)/h;
  if (scale>sc2)
    scale = sc2;
  // define o mapeamento
  dc.SetMapMode(MM_TEXT);
  ox = figArea.left + ((figArea.right-figArea.left)-(short)(w*scale))/2;
  oy = figArea.top + ((figArea.bottom-figArea.top)-(short)(h*scale))/2;

  dc.SelectStockObject(SYSTEM_FONT);
  dc.SetBkMode(TRANSPARENT);
  // desenha o carro
  dc.Rectangle(ox,oy,ox + 1 + (short)(w*scale),oy + 1 + (short)(h*scale));
  // eixo X
```

```
dc.MoveTo(ox+(short)((w/2)*scale),oy+(short)(xf*scale));
dc.LineTo(ox+(short)((w/2)*scale),oy-10);
// flecha X
dc.LineTo(ox+(short)((w/2)*scale)+2,oy-1);
dc.MoveTo(ox+(short)((w/2)*scale),oy-10);
dc.LineTo(ox+(short)((w/2)*scale)-2,oy-1);
// texto X
dc.TextOut(ox+(short)((w/2)*scale)+5,oy-15,"X",1);
// eixo Y
dc.MoveTo(ox+(short)((w/2)*scale),oy+(short)(xf*scale));
dc.LineTo(ox-10,oy+(short)(xf*scale));
// flecha Y
dc.LineTo(ox-1,oy+(short)(xf*scale)+2);
dc.MoveTo(ox-10,oy+(short)(xf*scale));
dc.LineTo(ox-1,oy+(short)(xf*scale)-2);
// texto Y
dc.TextOut(ox-15,oy+(short)(xf*scale)+5,"Y",1);

// atualmente, precisa de pelo menos tres pontos...
if (ct>=2)
{
  // traca linhas
  cp = new CPen(PS_SOLID, 1, RGB(0,0,255));
  oldCp = dc.SelectObject(cp);
  // seleciona conforme a posicao do crush
  for (i=0; i<ct; i++)
  {
    pC = pCrush->GetAt(i);
    switch (tipo)
    {
      case 0: // frente
        if (i==0)
        {
          ox = ox+(int)(scale*(w/2. - m_y0/1000.));
          scale = scale / 1000.;
          dc.MoveTo(ox-(int)(pC->GetPos()*scale),oy+(int)(pC->GetProf()*scale));
        }
        else
          dc.LineTo(ox-(int)(pC->GetPos()*scale),oy+(int)(pC->GetProf()*scale));
        break;
      case 1: // tras
        if (i==0)
        {
          ox = ox+(int)(scale*(w/2. - m_y0/1000.));
          oy = oy+(int)(scale*h);
          scale = scale / 1000.;
          dc.MoveTo(ox-(int)(pC->GetPos()*scale),oy-(int)(pC->GetProf()*scale));
        }
        else
          dc.LineTo(ox-(int)(pC->GetPos()*scale),oy-(int)(pC->GetProf()*scale));
        break;
      case 2: // esquerda
        if (i==0)
        {
          oy = oy+(int)(scale*(xf - m_x0/1000.));
          scale = scale / 1000.;
          dc.MoveTo(ox+(int)(pC->GetProf()*scale),oy-(int)(pC->GetPos()*scale));
        }
        else
          dc.LineTo(ox+(int)(pC->GetProf()*scale),oy-(int)(pC->GetPos()*scale));
        break;
      case 3: // direita
        if (i==0)
        {
          ox = ox+(int)(scale*w);
          oy = oy+(int)(scale*(xf - m_x0/1000.));
          scale = scale / 1000.;
          dc.MoveTo(ox-(int)(pC->GetProf()*scale),oy-(int)(pC->GetPos()*scale));
        }
        else
          dc.LineTo(ox-(int)(pC->GetProf()*scale),oy-(int)(pC->GetPos()*scale));
        break;
    }
    // Faz uma marca na origem da batida
    CBrush* pBrush = new CBrush(RGB(255,0,0));
    dc.FillRect(CRect(ox-2,oy-2,ox+3,oy+3),pBrush);
    delete pBrush;
  }
  dc.SelectObject(oldCp);
  delete cp;
```

```
    }
    else // sempre faz uma marca na origem
    {
      switch (tipo)
      {
        case 0:
          ox = ox+(int)(scale*(w/2. - m_y0/1000.));
          break;
        case 1:
          ox = ox+(int)(scale*(w/2. - m_y0/1000.));
          oy = oy+(int)(scale*h);
          break;
        case 2:
          oy = oy+(int)(scale*(xf - m_x0/1000.));
          break;
        case 3:
          ox = ox+(int)(scale*w);
          oy = oy+(int)(scale*(xf - m_x0/1000.));
          break;
      }
      CBrush* pBrush = new CBrush(RGB(255,0,0));
      dc.FillRect(CRect(ox-2,oy-2,ox+3,oy+3),pBrush);
      delete pBrush;
    }

    delete pCar;
}

void CrushDlg::UpdateCrushProfile(CrushArray* ca)
{
    short ct,i;
    CString newStr;
    char buf[100];
    Crush* cr;

    m_Perfil.ResetContent();
    ct = ca->GetSize();
    for (i=0; i<ct; i++)
    {
      cr = ca->GetAt(i);
      sprintf(buf,"%10.2lf",cr->GetPos());
      newStr = buf;
      sprintf(buf,"%10.2lf",cr->GetProf());
      newStr += "    ";
      newStr += buf;
      m_Perfil.AddString(newStr);
    }
}

void CrushDlg::OnTipof()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    UINT curTab = ((MyTabView*)GetParent())->GetTabIndex();
    CarData* pCar = new CarData;

    pDoc->SetModifiedFlag(TRUE);

    if(curTab == 2)
    {
      pDoc->GetCar1(pCar);
      m_x0 = pCar->GetXF()*1000;
      m_y0 = 0;
    }
    if(curTab == 3)
    {
      pDoc->GetCar2(pCar);
      m_x0 = pCar->GetXF()*1000;
      m_y0 = 0;
    }

    GetDlgItem(IDC_C_X0)->EnableWindow(FALSE);
    GetDlgItem(IDC_C_Y0)->EnableWindow(TRUE);
    UpdateData(FALSE);
    tipo = 0;
    RedrawWindow();

    delete pCar;
}

void CrushDlg::OnTipold()
```

```
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  UINT curTab = ((MyTabView*)GetParent())->GetTabIndex();
  CarData* pCar = new CarData;

  pDoc->SetModifiedFlag(TRUE);

  if(curTab == 2)
  {
    pDoc->GetCar1(pCar);
    m_x0 = 0;
    m_y0 = - pCar->GetYS()*1000;
  }
  if(curTab == 3)
  {
    pDoc->GetCar2(pCar);
    m_x0 = 0;
    m_y0 = - pCar->GetYS()*1000;
  }

  GetDlgItem(IDC_C_X0)->EnableWindow(TRUE);
  GetDlgItem(IDC_C_Y0)->EnableWindow(FALSE);
  UpdateData(FALSE);
  tipo = 3;
  RedrawWindow();

  delete pCar;
}

void CrushDlg::OnTipole()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  UINT curTab = ((MyTabView*)GetParent())->GetTabIndex();
  CarData* pCar = new CarData;

  pDoc->SetModifiedFlag(TRUE);

  if(curTab == 2)
  {
    pDoc->GetCar1(pCar);
    m_x0 = 0;
    m_y0 = pCar->GetYS()*1000;
  }
  if(curTab == 3)
  {
    pDoc->GetCar2(pCar);
    m_x0 = 0;
    m_y0 = pCar->GetYS()*1000;
  }

  GetDlgItem(IDC_C_X0)->EnableWindow(TRUE);
  GetDlgItem(IDC_C_Y0)->EnableWindow(FALSE);
  UpdateData(FALSE);
  tipo = 2;
  RedrawWindow();

  delete pCar;
}

void CrushDlg::OnTipot()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  UINT curTab = ((MyTabView*)GetParent())->GetTabIndex();
  CarData* pCar = new CarData;

  pDoc->SetModifiedFlag(TRUE);

  if(curTab == 2)
  {
    pDoc->GetCar1(pCar);
    m_x0 = - pCar->GetXR()*1000;
    m_y0 = 0;
  }
  if(curTab == 3)
  {
    pDoc->GetCar2(pCar);
    m_x0 = - pCar->GetXR()*1000;
    m_y0 = 0;
  }
```

```
  GetDlgItem(IDC_C_X0)->EnableWindow(FALSE);
  GetDlgItem(IDC_C_Y0)->EnableWindow(TRUE);
  UpdateData(FALSE);
  tipo = 1;
  RedrawWindow();

  delete pCar;
}

void CrushDlg::OnSelchangeCPerfil()
{
  int i = m_Perfil.GetCurSel();
  Crush* pC;

  pC = pCrush->GetAt((short)i);

  m_pos = pC->GetPos();
  m_prof = pC->GetProf();

  UpdateData(FALSE);
}

void CrushDlg::OnLButtonUp(UINT nFlags, CPoint point)
{
  FACTDoc* pDoc;
  CarData* pCar = new CarData;
  CRect figArea;
  short ox, oy, x, y, xf, xr, ys;
  double w, h;
  double scale, sc2;
  double temp;

  /* Verificacao rapida: se o click nao ocorreu dentro da area do
     carrinho, entao retorna */
  if (ChildWindowFromPoint(point) == (CWnd*)(&m_Figura))
  {
    // Código específico: as crush dialogs sabem que
    // indice 2 = crush do carro 1
    // indice 3 = crush do carro 2
    UINT curTab = ((MyTabView*)GetParent())->GetTabIndex();
    pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());

    if (curTab == 2)
    {
      pDoc->GetCar1(pCar);
      w = (pCar->GetYS()*2);
      h = (pCar->GetXF() + pCar->GetXR());
    }
    else
    {
      pDoc->GetCar2(pCar);
      w = (pCar->GetYS()*2);
      h = (pCar->GetXF() + pCar->GetXR());
    }

    m_Figura.GetWindowRect(&figArea);
    ScreenToClient(&figArea);
    figArea.left+=20;
    figArea.top+=20;
    figArea.right-=20;
    figArea.bottom-=20;

    scale = ((double)figArea.right-(double)figArea.left)/w;
    sc2 = ((double)figArea.bottom-(double)figArea.top)/h;
    if (scale>sc2)
      scale = sc2;

    if (curTab == 2)
    {
      pDoc->GetCar1(pCar);
      xr = (short)(scale*(pCar->GetXR()));
      xf = (short)(scale*(pCar->GetXF()));
      ys = (short)(scale*(pCar->GetYS()));
    }
    else
    {
      pDoc->GetCar2(pCar);
      xr = (short)(scale*(pCar->GetXR()));
      xf = (short)(scale*(pCar->GetXF()));
      ys = (short)(scale*(pCar->GetYS()));
```

```
    }

    /* acha a origem do carro em pixels */
    ox = figArea.left + ((figArea.right-figArea.left)-(short)(w*scale))/2;
    oy = figArea.top + ((figArea.bottom-figArea.top)-(short)(h*scale))/2;
    ox += ys;
    oy += xf;

    /* converte as coordenadas do ponto clicado */
    y = ox - point.x;
    x = oy - point.y;

    switch (tipo)
    {
      case 0: //frente
        if ((abs(x-xf)<=5)&&(abs(y)<=ys))
        {
          temp = round(1000.*(double)y/scale);
          if (Consis(temp))
          {
            m_y0 = temp;
            pDoc->SetModifiedFlag(TRUE);
            UpdateData(FALSE);
            Invalidate();
          }
        }
        break;
      case 1: //tras
        if ((abs(x+xr)<=5)&&(abs(y)<=ys))
        {
          temp = round(1000.*(double)y/scale);
          if (Consis(temp))
          {
            m_y0 = temp;
            pDoc->SetModifiedFlag(TRUE);
            UpdateData(FALSE);
            Invalidate();
          }
        }
        break;
      case 2: //esquerda
        if ((abs(y-ys)<=5)&&(x>=-xr)&&(x<=xf))
        {
          temp = round(1000.*(double)x/scale);
          if (Consis(temp))
          {
            m_x0 = temp;
            pDoc->SetModifiedFlag(TRUE);
            UpdateData(FALSE);
            Invalidate();
          }
        }
        break;
      case 3: //direita
        if ((abs(y+ys)<=5)&&(x>=-xr)&&(x<=xf))
        {
          temp = round(1000.*(double)x/scale);
          if (Consis(temp))
          {
            m_x0 = temp;
            pDoc->SetModifiedFlag(TRUE);
            UpdateData(FALSE);
            Invalidate();
          }
        }
        break;
    }
  }

  delete pCar;

  CDialog::OnLButtonUp(nFlags, point);
}

void CrushDlg::OnChangeCX0()
{
  CString s;
  ((CEdit*)GetDlgItem(IDC_C_X0))->GetWindowText(s);
  double n = atof((const char*)s);
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
```

```
  if (Consis(n))
  {
    pDoc->SetModifiedFlag(TRUE);
    UpdateData(TRUE);
    Invalidate();
  }
  else
  {
    MessageBeep(MB_ICONEXCLAMATION);
    UpdateData(FALSE);
  }
}

void CrushDlg::OnChangeCY0()
{
  CString s;
  ((CEdit*)GetDlgItem(IDC_C_Y0))->GetWindowText(s);
  double n = atof((const char*)s);
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  if (Consis(n))
  {
    pDoc->SetModifiedFlag(TRUE);
    UpdateData(TRUE);
    Invalidate();
  }
  else
  {
    MessageBeep(MB_ICONEXCLAMATION);
    UpdateData(FALSE);
  }
}

int CrushDlg::OnVKeyToItem(UINT nKey, CListBox* pListBox, UINT nIndex)
{
  if(nKey == VK_DELETE)
  {
    OnCDelete();
    return(-2);
  }

  return(-1);
}

// funcao auxiliar, talvez depois seja melhor mudar para outro objeto ou fazer global
double CrushDlg::round(double in)
{
  double t, out;
  t = modf(in,&out);
  t = (fabs(t)>=0.5)?0:1;
  out += (out<0)?(-t):t;
  return out;
}

BOOL CrushDlg::Consis(double c)
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  CarData* pCar = new CarData;

  UINT curTab = ((MyTabView*)GetParent())->GetTabIndex();
  short ct;
  double max = 0, min = 0;
  double ys, xf, xr;

  if (curTab == 2)
  {
    pDoc->GetCar1(pCar);
    ct = pCrush->GetSize();
    ys = pCar->GetYS()*1000.;
    xf = pCar->GetXF()*1000.;
    xr = pCar->GetXR()*1000.;
  }
  else
  {
    pDoc->GetCar2(pCar);
    ct = pCrush->GetSize();
    ys = pCar->GetYS()*1000.;
    xf = pCar->GetXF()*1000.;
    xr = pCar->GetXR()*1000.;
  }
```

```
  if (ct >0)
  {
    min = (pCrush->GetAt(0))->GetPos();
    max = (pCrush->GetAt(ct-1))->GetPos();
  }

  switch (tipo)
  {
    case 0:
      if (((c+max) > ys)||((c+min) < -ys))
        return FALSE;
    case 1:
      if (((c+max) > ys)||((c+min) < -ys))
        return FALSE;
    case 2:
      if (((c+max) > xf)||((c+min) < -xr))
        return FALSE;
    case 3:
      if (((c+max) > xf)||((c+min) < -xr))
        return FALSE;
  }
  return TRUE;

  delete pCar;
}

// inipdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// InitPosDlg dialog

class InitPosDlg : public CDialog
{
// Construction
public:
  InitPosDlg(CWnd* pParent = NULL); // standard constructor

  DECLARE_DYNAMIC(InitPosDlg)

// Dialog Data
  //{{AFX_DATA(InitPosDlg)
  enum { IDD = IDD_INIPOS };
  double  m_X01;
  double  m_X02;
  double  m_Y01;
  double  m_Y02;
  double  m_teta01;
  double  m_teta02;
  double  m_alfa;
  double  m_V01;
  double  m_V02;
  double  m_Passo;
  double  m_mi;
  //}}AFX_DATA


// Overrides
public:

protected:
  virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

// Implementation
public:
  BOOL PreTranslateMessage(MSG*);

protected:

  // Generated message map functions
  //{{AFX_MSG(InitPosDlg)
  afx_msg void OnPaint();
  afx_msg void OnChangeTeta01();
  afx_msg void OnChangeTeta02();
  afx_msg void OnChangeX01();
  afx_msg void OnChangeX02();
  afx_msg void OnChangeY01();
  afx_msg void OnChangeY02();
  afx_msg void OnChangeAlfa();
  afx_msg void OnChangeV01();
```

```
    afx_msg void OnChangeV02();
    afx_msg void OnChangeMi();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

// inipdlg.cpp : implementation file
//

#include "stdafx.h"
#include "fact.h"
#include "cardata.h"
#include "crush.h"
#include "factdoc.h"
#include "factview.h"
#include "inipdlg.h"

#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

#ifndef DEGTORAD
  #define DEGTORAD    0.01745329
  #define RADTODEG   57.29578000
#endif  //DEGTORAD

/////////////////////////////////////////////////////////////////////////////
// InitPosDlg dialog

IMPLEMENT_DYNAMIC(InitPosDlg, CDialog)

InitPosDlg::InitPosDlg(CWnd* pParent /*=NULL*/)
  : CDialog()
{
  Create(InitPosDlg::IDD, pParent);
  //{{AFX_DATA_INIT(InitPosDlg)
  m_X01 = 0.0;
  m_X02 = 0.0;
  m_Y01 = 0.0;
  m_Y02 = 0.0;
  m_teta01 = 0.0;
  m_teta02 = 0.0;
  m_alfa = 0;
  m_V01 = 0;
  m_V02 = 0;
  m_Passo = 0;
  m_mi = 0;
  //}}AFX_DATA_INIT
}


void InitPosDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(InitPosDlg)
  DDX_Text(pDX, IDC_X01, m_X01);
  DDX_Text(pDX, IDC_X02, m_X02);
  DDX_Text(pDX, IDC_Y01, m_Y01);
  DDX_Text(pDX, IDC_Y02, m_Y02);
  DDX_Text(pDX, IDC_TETA01, m_teta01);
  DDX_Text(pDX, IDC_TETA02, m_teta02);
  DDX_Text(pDX, IDC_ALFA, m_alfa);
  DDX_Text(pDX, IDC_V01, m_V01);
  DDX_Text(pDX, IDC_V02, m_V02);
  DDX_Text(pDX, IDC_PASSO, m_Passo);
  DDX_Text(pDX, IDC_MI, m_mi);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(InitPosDlg, CDialog)
  //{{AFX_MSG_MAP(InitPosDlg)
  ON_WM_PAINT()
  ON_EN_CHANGE(IDC_TETA01, OnChangeTeta01)
  ON_EN_CHANGE(IDC_TETA02, OnChangeTeta02)
  ON_EN_CHANGE(IDC_X01, OnChangeX01)
  ON_EN_CHANGE(IDC_X02, OnChangeX02)
  ON_EN_CHANGE(IDC_Y01, OnChangeY01)
```

```
  ON_EN_CHANGE(IDC_Y02, OnChangeY02)
  ON_EN_CHANGE(IDC_ALFA, OnChangeAlfa)
  ON_EN_CHANGE(IDC_V01, OnChangeV01)
  ON_EN_CHANGE(IDC_V02, OnChangeV02)
  ON_EN_CHANGE(IDC_MI, OnChangeMi)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL InitPosDlg::PreTranslateMessage(MSG* pMsg)
{
  // na mensagem KEYDOWN, evita a criacao de um WM_CHAR quando a tecla
  // pressionada for TAB (sem CTRL)
  if (pMsg->message == WM_KEYDOWN)
  {
    if ((pMsg->wParam == VK_TAB)&&(!(GetKeyState(VK_CONTROL)&0xFF00)))
      return 1;
  }

  // na mensagem KEYUP, trata os casos corretos
  if (pMsg->message == WM_KEYUP)
  {
    // se for TAB
    if (pMsg->wParam == VK_TAB)
    {
      // e CTRL, manda para a janela parent
      if (GetKeyState(VK_CONTROL)&0xFF00)
        return 0;
      // senao, muda o foco para o proximo controle
      else
      {
        NextDlgCtrl();
        return 1;
      }
    }
  }

  return(CWnd::PreTranslateMessage(pMsg));
}

/////////////////////////////////////////////////////////////////////////////
// InitPosDlg message handlers

void InitPosDlg::OnPaint()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
  CarData* pCar = new CarData;

  CRect figArea;
  short ox, oy;
  short carOx, carOy;
  CPoint pontos[4];
  CPen *cp1, *cp2, *oldCp;
  double ang1, ang2;
  double l1, l2, y1, y2;
  double l1r, l2r;
  double w, h, dd, dd2;
  double scale,sc2;
  CStatic* pCen;

  CPaintDC dc(this); // device context for painting

  pCen = (CStatic*)(GetDlgItem(IDC_CENARIO));
  pCen->GetWindowRect(&figArea);
  ScreenToClient(&figArea);
  figArea.left+=10;
  figArea.top+=30;
  figArea.right-=10;
  figArea.bottom-=10;

  UpdateData(TRUE);

  // obtem os dados dos carros
  pDoc->GetCar1(pCar);
  ang1 = DEGTORAD*m_teta01;
  l1r = pCar->GetXR();
  l1 =  pCar->GetXF();
  y1 =  pCar->GetYS();
  pDoc->GetCar2(pCar);
  ang2 = DEGTORAD*m_teta02;
  l2r = pCar->GetXR();
```

```
l2 =  pCar->GetXF();
y2 =  pCar->GetYS();

/* calcula uma escala correta */
dd = sqrt((l1+l1r)*(l1+l1r)+y1*y1);
dd2 = sqrt((l2+l2r)*(l2+l2r)+y2*y2);
if (dd2>dd)
  dd=dd2;
w = 2*dd;
h = 2*dd;

/* reutiliza dd e dd2 para armazenar as origens dos carros */
dd = (m_X01 + m_X02)/2.;
dd2 = (m_Y01 + m_Y02)/2.;

scale = ((double)figArea.right-(double)figArea.left)/w;
sc2 = ((double)figArea.bottom-(double)figArea.top)/h;
if (scale>sc2)
  scale = sc2;

// desenha os carros
cp1 = new CPen(PS_SOLID,2,RGB(255,0,0));
cp2 = new CPen(PS_SOLID,2,RGB(0,0,255));
ox = figArea.left + (int)(scale * w/2.);
oy = figArea.top + (int)(scale * h/2.);
// carro 1
carOx = ox+(int)((m_X01-dd)*scale);
carOy = oy-(int)((m_Y01-dd2)*scale);
oldCp = dc.SelectObject(cp1);
pontos[0].x = carOx + (int)((y1*sin(ang1)+l1*cos(ang1))*scale);
pontos[0].y = carOy - (int)((l1*sin(ang1)-y1*cos(ang1))*scale);
pontos[1].x = carOx + (int)((-l1r*cos(ang1)+y1*sin(ang1))*scale);
pontos[1].y = carOy - (int)((-y1*cos(ang1)-l1r*sin(ang1))*scale);
pontos[2].x = carOx + (int)((-l1r*cos(ang1)-y1*sin(ang1))*scale);
pontos[2].y = carOy - (int)((-l1r*sin(ang1)+y1*cos(ang1))*scale);
pontos[3].x = carOx + (int)((l1*cos(ang1)-y1*sin(ang1))*scale);
pontos[3].y = carOy - (int)((l1*sin(ang1)+y1*cos(ang1))*scale);
dc.MoveTo(pontos[0]);
dc.LineTo(pontos[1]);
dc.LineTo(pontos[2]);
dc.LineTo(pontos[3]);
dc.LineTo(pontos[0]);
// desenha o baricentro e identifica a frente do carro com uma flecha
pontos[0].x = carOx;
pontos[0].y = carOy;
pontos[1].x = carOx + (int)(0.5*cos(ang1)*scale);
pontos[1].y = carOy - (int)(0.5*sin(ang1)*scale);
pontos[2].x = carOx + (int)((0.1*sin(ang1)+0.35*cos(ang1))*scale);
pontos[2].y = carOy - (int)((0.35*sin(ang1)-0.1*cos(ang1))*scale);
pontos[3].x = carOx + (int)((0.35*cos(ang1)-0.1*sin(ang1))*scale);
pontos[3].y = carOy - (int)((0.35*sin(ang1)+0.1*cos(ang1))*scale);
dc.MoveTo(pontos[0]);
dc.LineTo(pontos[1]);
dc.LineTo(pontos[2]);
dc.MoveTo(pontos[1]);
dc.LineTo(pontos[3]);
// carro 2
carOx = ox+(int)((m_X02-dd)*scale);
carOy = oy-(int)((m_Y02-dd2)*scale);
dc.SelectObject(cp2);
pontos[0].x = carOx + (int)((y2*sin(ang2)+l2*cos(ang2))*scale);
pontos[0].y = carOy - (int)((l2*sin(ang2)-y2*cos(ang2))*scale);
pontos[1].x = carOx + (int)((-l2r*cos(ang2)+y2*sin(ang2))*scale);
pontos[1].y = carOy - (int)((-y2*cos(ang2)-l2r*sin(ang2))*scale);
pontos[2].x = carOx + (int)((-l2r*cos(ang2)-y2*sin(ang2))*scale);
pontos[2].y = carOy - (int)((-l2r*sin(ang2)+y2*cos(ang2))*scale);
pontos[3].x = carOx + (int)((l2*cos(ang2)-y2*sin(ang2))*scale);
pontos[3].y = carOy - (int)((l2*sin(ang2)+y2*cos(ang2))*scale);
dc.MoveTo(pontos[0]);
dc.LineTo(pontos[1]);
dc.LineTo(pontos[2]);
dc.LineTo(pontos[3]);
dc.LineTo(pontos[0]);
// desenha o baricentro e identifica a frente do carro com uma flecha
pontos[0].x = carOx;
pontos[0].y = carOy;
pontos[1].x = carOx + (int)(0.5*cos(ang2)*scale);
pontos[1].y = carOy - (int)(0.5*sin(ang2)*scale);
pontos[2].x = carOx + (int)((0.1*sin(ang2)+0.35*cos(ang2))*scale);
pontos[2].y = carOy - (int)((0.35*sin(ang2)-0.1*cos(ang2))*scale);
```

```
    pontos[3].x = carOx + (int)((0.35*cos(ang2)-0.1*sin(ang2))*scale);
    pontos[3].y = carOy - (int)((0.35*sin(ang2)+0.1*cos(ang2))*scale);
    dc.MoveTo(pontos[0]);
    dc.LineTo(pontos[1]);
    dc.LineTo(pontos[2]);
    dc.MoveTo(pontos[1]);
    dc.LineTo(pontos[3]);
    // volta os objetos, etc.
    dc.SelectObject(oldCp);
    delete cp1;
    delete cp2;
    delete pCar;
}

void InitPosDlg::OnChangeTeta01()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
    Invalidate();
}

void InitPosDlg::OnChangeTeta02()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
    Invalidate();
}

void InitPosDlg::OnChangeX01()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
    Invalidate();
}

void InitPosDlg::OnChangeX02()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
    Invalidate();
}

void InitPosDlg::OnChangeY01()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
    Invalidate();
}

void InitPosDlg::OnChangeY02()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
    Invalidate();
}

void InitPosDlg::OnChangeAlfa()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
}

void InitPosDlg::OnChangeV01()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
}

void InitPosDlg::OnChangeV02()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
}

void InitPosDlg::OnChangeMi()
{
    FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());
    pDoc->SetModifiedFlag(TRUE);
}
```

```
// trjdlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// TrajetoDlg dialog

class TrajetoDlg : public CDialog
{
// Construction
public:
  TrajetoDlg(CWnd* pParent = NULL); // standard constructor

  DECLARE_DYNAMIC(TrajetoDlg)

// Dialog Data
  //{{AFX_DATA(TrajetoDlg)
  enum { IDD = IDD_TRAJETO };
  //}}AFX_DATA

protected:
  BOOL bDrawCar1, bDrawCar2;
  BOOL bGraphCar1, bGraphCar2;
  short nTipoOutput;
  short nCodGraph;
  double PassoAnim;
// Implementation
public:
  double Passo;
  BOOL PreTranslateMessage(MSG*);

protected:
  void DesenhaTrajetorias(FACTDoc* pDoc, CDC* pDC, CRect& figArea, double step);
  void DesenhaGrafico(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nGraph);
  void CalcPontos(CPoint pontos[4], short ox, short oy, double xf, double xr, double ys,
double theta, double scale);
  void DesenhaCarro(CDC* dc, CPoint pontos[4]);
  void GraphPontos(FACTDoc* pDoc, short nGraph, double T, short nCar, double* x, double*
y);

  void UpdateAllRadio();
  void MyInvalidate();
protected:
  virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

  // Generated message map functions
  //{{AFX_MSG(TrajetoDlg)
  afx_msg void OnPaint();
  afx_msg void OnContornoCarro1();
  afx_msg void OnContornoCarro2();
  virtual BOOL OnInitDialog();
  afx_msg void OnExecutarAnim();
  afx_msg void OnOptgrafico();
  afx_msg void OnOpttrajeto();
  afx_msg void OnOptveic1();
  afx_msg void OnOptambos();
  afx_msg void OnSelendokTipograf();
  afx_msg void OnOptveic2();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

// trjdlg.cpp : implementation file
//

#include "stdafx.h"
#include "fact.h"
#include "cardata.h"
#include "crush.h"
#include "trajeto.h"

#include "factdoc.h"
#include "factview.h"
#include "trjdlg.h"

#include <math.h>
#include <time.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
```

```
#endif

/////////////////////////////////////////////////////////////////////////////
// TrajetoDlg dialog

IMPLEMENT_DYNAMIC(TrajetoDlg, CDialog)

TrajetoDlg::TrajetoDlg(CWnd* pParent /*=NULL*/)
  : CDialog()
{
  Create(TrajetoDlg::IDD, pParent);
  bDrawCar1 = TRUE;
  bDrawCar2 = TRUE;
  bGraphCar1 = TRUE;
  bGraphCar2 = TRUE;
  PassoAnim = 0;
  Passo = 100;
  nTipoOutput = 0;
  nCodGraph = 0;
  //{{AFX_DATA_INIT(TrajetoDlg)
  //}}AFX_DATA_INIT
}

void TrajetoDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(TrajetoDlg)
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(TrajetoDlg, CDialog)
  //{{AFX_MSG_MAP(TrajetoDlg)
  ON_WM_PAINT()
  ON_BN_CLICKED(IDC_CARRO1, OnContornoCarro1)
  ON_BN_CLICKED(IDC_CARRO2, OnContornoCarro2)
  ON_BN_CLICKED(IDC_ANIM, OnExecutarAnim)
  ON_BN_CLICKED(IDC_OPTGRAFICO, OnOptgrafico)
  ON_BN_CLICKED(IDC_OPTTRAJETO, OnOpttrajeto)
  ON_BN_CLICKED(IDC_OPTVEIC1, OnOptveic1)
  ON_BN_CLICKED(IDC_OPTAMBOS, OnOptambos)
  ON_CBN_SELENDOK(IDC_TIPOGRAF, OnSelendokTipograf)
  ON_BN_CLICKED(IDC_OPTVEIC2, OnOptveic2)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL TrajetoDlg::PreTranslateMessage(MSG* pMsg)
{
  // na mensagem KEYDOWN, evita a criacao de um WM_CHAR quando a tecla
  // pressionada for TAB (sem CTRL)
  if (pMsg->message == WM_KEYDOWN)
  {
    if ((pMsg->wParam == VK_TAB)&&(!(GetKeyState(VK_CONTROL)&0xFF00)))
      return 1;
  }

  // na mensagem KEYUP, trata os casos corretos
  if (pMsg->message == WM_KEYUP)
  {
    // se for TAB
    if (pMsg->wParam == VK_TAB)
    {
      // e CTRL, manda para a janela parent
      if (GetKeyState(VK_CONTROL)&0xFF00)
        return 0;
      // senao, muda o foco para o proximo controle
      else
      {
        NextDlgCtrl();
        return 1;
      }
    }
  }

  return(CWnd::PreTranslateMessage(pMsg));
}

/////////////////////////////////////////////////////////////////////////////
// TrajetoDlg message handlers

/*****************************************************************************
```

```
/ OnPaint()                                                                  /
/ Funcao responsavel por executar o desenho do conteudo da tab de           /
/ trajetoria. Esta funcao inicialmente escreve na tela os tempos de final   /
/ da colisao e simulacao e em seguida chama a funcao responsavel por        /
/ desenhar o grafico ou trajetoria, conforme o caso                         /
/ Parametros: -                                                             /
/ Valor de Retorno: -                                                       /
 ****************************************************************************/
void TrajetoDlg::OnPaint()
{
  FACTDoc* pDoc = (FACTDoc*)(((FACTView*)GetParent())->GetDocument());

  char buffer[100];
  double tendcol = pDoc->GetTEndCol();
  CRect figArea;
  CStatic* pCen;

  CPaintDC dc(this); // device context for painting
  dc.SetMapMode(MM_TEXT);
  dc.SetBkMode(TRANSPARENT);

  if (tendcol != -1)
    sprintf(buffer, "%7.4lf", tendcol);
  else
    sprintf(buffer, "---");
  GetDlgItem(IDC_TENDCOL)->SetWindowText(buffer);
  sprintf(buffer, "%7.4lf", pDoc->GetLastT());
  GetDlgItem(IDC_TENDSIM)->SetWindowText(buffer);

  pCen = (CStatic*)(GetDlgItem(IDC_TRAJETO));
  pCen->GetWindowRect(&figArea);
  ScreenToClient(&figArea);
  figArea.left+=10;
  figArea.top+=15;
  figArea.right-=10;
  figArea.bottom-=10;

  UpdateData(TRUE);

  if (nTipoOutput == 0)
  {
    GetDlgItem(IDC_ANIM)->EnableWindow(FALSE);
    DesenhaTrajetorias(pDoc, (CDC*)&dc, figArea, Passo/1000.);
    GetDlgItem(IDC_ANIM)->EnableWindow(TRUE);
  }
  else
    DesenhaGrafico(pDoc, (CDC*)&dc, figArea, nCodGraph);
}

/****************************************************************************
/ DesenhaTrajetorias()                                                       /
/ Esta funcao e usada para desenhar as trajetorias dos veiculos ao longo    /
/ da simulacao, usando para isso dados obtidos do objeto FACTDoc.           /
/ Parametros:                                                               /
/   pDoc - ponteiro para o objeto FACTDoc que contem os dados da simulacao  /
/   pDC - ponteiro para o device context onde se deve desenhar              /
/   figArea - referencia a um objeto CRect que delimita a area de desenho   /
/   step - passo de tempo entre os quadros da trajetoria, em segundos       /
/ Valor de Retorno: -                                                       /
 ****************************************************************************/
void TrajetoDlg::DesenhaTrajetorias(FACTDoc* pDoc, CDC* pDC, CRect& figArea, double
step)
{
  CarData* pCar = new CarData;
  TrajPoint* pTraj = new TrajPoint;
  double T, maxT;
  short ox, oy;
  short carOx1, carOy1;
  short carOx2, carOy2;
  CPoint pontos[4];
  CPen *cp1, *cp2, *oldCp;
  double l1, l2, l1r, l2r, y1, y2;
  double w, h, dd, dd2;
  double xmax,ymax,xmin,ymin;
  double scale,sc2;
  clock_t start, delta = (clock_t)(PassoAnim*CLOCKS_PER_SEC);

  // obtem os dados dos carros
  pDoc->GetCar1(pCar);
  l1r = pCar->GetXR();
```

```
l1 =  pCar->GetXF();
y1 =  pCar->GetYS();
pDoc->GetCar2(pCar);
l2r = pCar->GetXR();
l2 =  pCar->GetXF();
y2 =  pCar->GetYS();

/* calcula uma escala correta */
dd = sqrt((l1+l1r)*(l1+l1r)+y1*y1);
dd2 = sqrt((l2+l2r)*(l2+l2r)+y2*y2);
if (dd2>dd)
  dd=dd2;
dd = dd*0.75;
pDoc->GetMaxRect(&xmin,&ymin,&xmax,&ymax);
xmin -= dd;
ymin -= dd;
xmax += dd;
ymax += dd;
w = xmax - xmin;
h = ymax - ymin;
scale = ((double)figArea.right-(double)figArea.left)/w;
sc2 = ((double)figArea.bottom-(double)figArea.top)/h;
if (scale>sc2)
  scale = sc2;

// cria as pens para usar no desenho
cp1 = new CPen(PS_SOLID,1,RGB(255,0,0));
cp2 = new CPen(PS_SOLID,1,RGB(0,0,255));

maxT = pDoc->GetLastT();
ox = figArea.left + (short)(fabs(xmin)*scale);
oy = figArea.top + (short)(fabs(ymax)*scale);
/* desenha as trajetorias dos carros 1 e 2 */
start = clock();
T = 0;
oldCp = pDC->SelectObject(cp1);
pDoc->ResultadoEmT(T,0,pTraj);
carOx1 = ox + (short)(scale*pTraj->GetX());
carOy1 = oy - (short)(scale*pTraj->GetY());
CBrush *pBrush = new CBrush(RGB(255,0,0));
pDC->FillRect(CRect(carOx1-2, carOy1-2, carOx1+3, carOy1+3), pBrush);
delete pBrush;
// desenha o carro na posicao inicial...
if (bDrawCar1)
{
  CalcPontos(pontos, carOx1, carOy1, l1, l1r, y1, pTraj->GetTheta(), scale);
  DesenhaCarro(pDC, pontos);
}
/* desenha a trajetoria do carro 2 */
T = 0;
pDC->SelectObject(cp2);
pDoc->ResultadoEmT(T,1,pTraj);
carOx2 = ox + (short)(scale*pTraj->GetX());
carOy2 = oy - (short)(scale*pTraj->GetY());
pBrush = new CBrush(RGB(0,0,255));
pDC->FillRect(CRect(carOx2-2, carOy2-2, carOx2+3, carOy2+3), pBrush);
delete pBrush;
// desenha o carro na posicao inicial...
if (bDrawCar2)
{
  CalcPontos(pontos, carOx2, carOy2, l2, l2r, y2, pTraj->GetTheta(), scale);
  DesenhaCarro(pDC, pontos);
}
while (clock()-start < delta);
while (T<maxT)
{
  start = clock();
  pDC->SelectObject(cp1);
  pDC->MoveTo(carOx1, carOy1);
  T += step;
  if (T>maxT)
    T = maxT;
  pDoc->ResultadoEmT(T,0,pTraj);
  carOx1 = ox + (short)(scale*pTraj->GetX());
  carOy1 = oy - (short)(scale*pTraj->GetY());
  pDC->LineTo(carOx1, carOy1);
  if (bDrawCar1)
  {
    CalcPontos(pontos, carOx1, carOy1, l1, l1r, y1, pTraj->GetTheta(), scale);
    DesenhaCarro(pDC, pontos);
```

```
    }
    pDC->SelectObject(cp2);
    pDC->MoveTo(carOx2, carOy2);
    pDoc->ResultadoEmT(T,1,pTraj);
    carOx2 = ox + (short)(scale*pTraj->GetX());
    carOy2 = oy - (short)(scale*pTraj->GetY());
    pDC->LineTo(carOx2, carOy2);
    if (bDrawCar2)
    {
      CalcPontos(pontos, carOx2, carOy2, l2, l2r, y2, pTraj->GetTheta(), scale);
      DesenhaCarro(pDC, pontos);
    }
    while (clock()-start < delta);
  }

  PassoAnim = 0;

  pDC->SelectObject(oldCp);
  delete cp1;
  delete cp2;
  delete pTraj;
  delete pCar;
}

/*************************************************************************
/ DesenhaGrafico()                                                       /
/ Esta funcao e utilizada para plotar um dos graficos de saida na janela /
/ de trajetoria.                                                         /
/ Parametros:                                                            /
/   pDoc - ponteiro para o objeto FACTDoc que contem os dados da simulacao /
/   pDC - ponteiro para o device context onde se deve desenhar           /
/   figArea - referencia a um objeto CRect que delimita a area de desenho /
/   nGraph - inteiro entre 0 e 6, identifica qual o grafico a ser plotado /
/ Valor de Retorno: -                                                    /
 *************************************************************************/
void TrajetoDlg::DesenhaGrafico(FACTDoc* pDoc, CDC* pDC, CRect& figArea, short nGraph)
{
// desenha um gráfico não-proporcional em que
// aparecem os dados de um ou dos dois carros
//
// os valores para nGraph são:
// 0 = V  x t
// 1 = Vx x t
// 2 = Vy x t
// 3 = X  x t
// 4 = Y  x t
// 5 = W  x t
// 6 = Theta x t
//
  CPen *cp1, *cp2, *oldCp;
  TrajPoint cTraj, minTraj, maxTraj;
  double minV, maxV;
  double deltax, deltay, scalex, scaley;
  double ox, oy;
  double x1, y1, x2, y2;
  double T, maxT, step;
  double eX, eY;
  CString sPotX, sPotY;
  double stepX = 1, stepY = 1;
  short maxTicksX, maxTicksY;
  char buf[20];
  CString title;
  CSize fSize;
  CFont* oldFon;
  short flagCars;

  // obtem o tempo final e calcula um passo de interpolacao
  maxT = pDoc->GetLastT();
  step = maxT/75.;

  // obtem os limites de todas as grandezas do grafico
  flagCars = ((bGraphCar1)?1:0) + ((bGraphCar2)?2:0);
  minV = 0;
  maxV = 0;
  pDoc->GetMinMaxTraj(&minTraj, &minV, &maxTraj, &maxV, flagCars);

  // calcula os deltas para escala adequadamente, conforme o gráfico a ser plotado
  deltax = maxT;
  ox = 0;
  sPotX = "s";
```

```
switch (nGraph)
{
  case 0:
    title = "Velocidade x t";
    sPotY = "km/h";
    oy = minV;
    deltay = maxV - minV;
    break;
  case 1:
    title = "Velocidade(X) x t";
    sPotY = "km/h";
    oy = minTraj.GetVX(TRUE);
    deltay = maxTraj.GetVX(TRUE) - oy;
    break;
  case 2:
    title = "Velocidade(Y) x t";
    sPotY = "km/h";
    oy = minTraj.GetVY(TRUE);
    deltay = maxTraj.GetVY(TRUE) - oy;
    break;
  case 3:
    title = "Posição(X) x t";
    sPotY = "m";
    oy = minTraj.GetX();
    deltay = maxTraj.GetX() - oy;
    break;
  case 4:
    title = "Posição(Y) x t";
    sPotY = "m";
    oy = minTraj.GetY();
    deltay = maxTraj.GetY() - oy;
    break;
  case 5:
    title = "Velocidade Angular x t";
    sPotY = "°/s";
    oy = minTraj.GetW(TRUE);
    deltay = maxTraj.GetW(TRUE) - oy;
    break;
  case 6:
    title = "Ângulo x t";
    sPotY = "graus";
    oy = minTraj.GetTheta(TRUE);
    deltay = maxTraj.GetTheta(TRUE) - oy;
    break;
}

// evita divisao por zero
if (deltax < 1e-6)
  deltax = 1;
if (deltay < 1e-6)
  deltay = 1;

// desenha os eixos e outros acessorios
oldCp = (CPen*)pDC->SelectStockObject(BLACK_PEN);
oldFon = (CFont*)pDC->SelectStockObject(SYSTEM_FONT);
// calcula a largura media dos caracteres, ponderando
// um pouco para cima usando caracteres largos
fSize = pDC->GetOutputTextExtent("0123456789AOMW",14);
fSize.cx/= 14;
pDC->SetTextAlign(TA_CENTER|TA_TOP);
pDC->TextOut(figArea.left+figArea.Width()/2,figArea.top,title);
// reduz a area de plotagem descontando o espaco usado para
// escala e titulo
figArea.top += 2*fSize.cy;
figArea.left += 4*fSize.cx;
figArea.bottom -= 2*fSize.cy;
figArea.right -= 2*fSize.cx;
pDC->MoveTo(figArea.left, figArea.bottom);
pDC->LineTo(figArea.right, figArea.bottom);
pDC->MoveTo(figArea.left, figArea.bottom);
pDC->LineTo(figArea.left, figArea.top);

// calcula as escalas
short w = figArea.Width();
short h = figArea.Height();
scalex = ((double)figArea.Width())/deltax;
scaley = ((double)figArea.Height())/deltay;

// calcula o numero maximo de ticks
maxTicksX = figArea.Width()/(4*fSize.cx);
```

```
maxTicksY = figArea.Height()/(2*fSize.cy);

// calcula os expoentes dos eixos
eX = (short)log10(__max(fabs(ox), fabs(ox+deltax))) - 1;
eY = (short)log10(__max(fabs(oy), fabs(oy+deltay))) - 1;

// calcula os deltas, que sao inicialmente usados como o numero de ticks.
// se eles forem muito poucos, aumenta por um fator de 10
deltax /= pow(10,eX);
if (deltax <= 3)
{
  deltax *= 10;
  eX--;
}
deltay /= pow(10,eY);
if (deltay <= 3)
{
  deltay *= 10;
  eY--;
}

// verifica a escala (1, 2, 5 e 10) que sera usada
stepX = pow(10,eX);
if (deltax > maxTicksX)
  if (deltax > 2*maxTicksX)
    if (deltax > 5*maxTicksX)
      stepX = pow(10,eX+1);
    else
      stepX = 5*pow(10,eX);
  else
    stepX = 2*pow(10,eX);
stepY = pow(10,eY);
if (deltay > maxTicksY)
  if (deltay > 2*maxTicksY)
    if (deltay > 5*maxTicksY)
      stepY = pow(10,eY+1);
    else
      stepY = 5*pow(10,eY);
  else
    stepY = 2*pow(10,eY);

deltax *= pow(10,eX);
deltay *= pow(10,eY);

// calcula os valores minimos a serem escritos nos ticks
double minVer, minHor;
if (ox <= 0)
  minHor = ox - fmod(ox,stepX);
else
  minHor = ox + stepX - fmod(ox,stepX);
if (oy <= 0)
  minVer = oy - fmod(oy,stepY);
else
  minVer = oy + stepY - fmod(oy,stepY);

// calcula as posicoes iniciais dos ticks
short X, Y;
X = (short)((minHor - ox)*scalex);
Y = (short)((minVer - oy)*scaley);

// plota os ticks horizontais
pDC->SetTextAlign(TA_CENTER|TA_TOP);
while (X <= figArea.Width())
{
  pDC->MoveTo(figArea.left+X, figArea.bottom);
  pDC->LineTo(figArea.left+X, figArea.bottom+5);
  sprintf(buf,"%2.0lf",minHor/pow(10,eX));
  pDC->TextOut(figArea.left+X, figArea.bottom+6,buf,lstrlen(buf));
  minHor += stepX;
  X = (short)((minHor - ox)*scalex);
}

// plota os ticks verticais
pDC->SetTextAlign(TA_RIGHT|TA_BASELINE);
while (Y <= figArea.Height())
{
  pDC->MoveTo(figArea.left, figArea.bottom-Y);
  pDC->LineTo(figArea.left-5, figArea.bottom-Y);
  sprintf(buf,"%2.0lf",minVer/pow(10,eY));
  pDC->TextOut(figArea.left-6, figArea.bottom-Y,buf,lstrlen(buf));
```

```
    minVer += stepY;
    Y = (short)((minVer - oy)*scaley);
  }

  // escreve as potencias de 10 e unidades
  pDC->SetTextAlign(TA_RIGHT|TA_TOP);
  if (eX)
  {
    sprintf(buf,"x1E%2.0lf ",eX);
    sPotX = buf + sPotX;
  }
  pDC->TextOut(figArea.right + 2*fSize.cx, figArea.bottom + 6 + fSize.cy, sPotX);
  pDC->SetTextAlign(TA_LEFT|TA_BOTTOM);
  if (eY)
  {
    sprintf(buf,"x1E%2.0lf ",eY);
    sPotY = buf + sPotY;
  }
  pDC->TextOut(figArea.left + 2*fSize.cx, figArea.top - 4, sPotY);

  // desenha a linha do zero e a linha do final da colisao
  cp1 = new CPen(PS_DOT,1,RGB(0,0,0));
  pDC->SelectObject(cp1);
  double tend = pDoc->GetTEndCol();
  pDC->MoveTo(figArea.left + (short)(scalex*tend), figArea.bottom);
  pDC->LineTo(figArea.left + (short)(scalex*tend), figArea.top);
  if ((oy<0)&&(oy+deltay>=0))
  {
    pDC->MoveTo(figArea.left, figArea.bottom + (short)(scaley*oy));
    pDC->LineTo(figArea.right, figArea.bottom + (short)(scaley*oy));
  }
  pDC->SelectStockObject(BLACK_PEN);
  delete cp1;

  // cria as pens para usar no desenho
  cp1 = new CPen(PS_SOLID,1,RGB(255,0,0));
  cp2 = new CPen(PS_DOT,1,RGB(0,0,255));

  T = 0;
  if (bGraphCar1)
  {
    GraphPontos(pDoc, nGraph, T, 0, &x1, &y1);
    x1 = x1 - ox;
    y1 = y1 - oy;
  }
  if (bGraphCar2)
  {
    GraphPontos(pDoc, nGraph, T, 1, &x2, &y2);
    x2 = x2 - ox;
    y2 = y2 - oy;
  }
  while (T < maxT)
  {
    T += step;
    if (T>maxT)
      T = maxT;
    // obtem os dados para o carro 1
    if (bGraphCar1)
    {
      pDC->MoveTo(figArea.left      +      (short)(scalex*x1),      figArea.bottom      -
(short)(scaley*y1));
      pDC->SelectObject(cp1);
      GraphPontos(pDoc, nGraph, T, 0, &x1, &y1);
      x1 = x1 - ox;
      y1 = y1 - oy;
      pDC->LineTo(figArea.left      +      (short)(scalex*x1),      figArea.bottom      -
(short)(scaley*y1));
    }
    // obtem os dados para o carro 2
    if (bGraphCar2)
    {
      pDC->MoveTo(figArea.left      +      (short)(scalex*x2),      figArea.bottom      -
(short)(scaley*y2));
      pDoc->ResultadoEmT(T,1,&cTraj);
      pDC->SelectObject(cp2);
      GraphPontos(pDoc, nGraph, T, 1, &x2, &y2);
      x2 = x2 - ox;
      y2 = y2 - oy;
      pDC->LineTo(figArea.left      +      (short)(scalex*x2),      figArea.bottom      -
(short)(scaley*y2));
```

```
    }
  }

  pDC->SelectObject(oldCp);
  delete cp1;
  delete cp2;
}

/*************************************************************************
/ CalcPontos()                                                          /
/ Esta e uma funcao auxiliar de DesenhaTrajetorias que calcula as coorde- /
/ nadas dos vertices do retangulo que representa um dos veiculos.        /
/ Parametros:                                                           /
/   pontos - vetor de objetos CPoint que sera usado para armazenar as    /
/  coordenadas dos vertices                                             /
/   ox, oy - coordenadas de origem da tela                               /
/   xf, xr, ys - dimensoes do veiculo                                    /
/   theta - orientacao angular do veiculo                                /
/ Valor de Retorno: -                                                   /
 *************************************************************************/
void TrajetoDlg::CalcPontos(CPoint pontos[4], short ox, short oy, double xf, double xr,
double ys, double theta, double scale)
{
  pontos[0].x = ox + (int)((ys*sin(theta)+xf*cos(theta))*scale);
  pontos[0].y = oy - (int)((xf*sin(theta)-ys*cos(theta))*scale);
  pontos[1].x = ox + (int)((-xr*cos(theta)+ys*sin(theta))*scale);
  pontos[1].y = oy - (int)((-ys*cos(theta)-xr*sin(theta))*scale);
  pontos[2].x = ox + (int)((-xr*cos(theta)-ys*sin(theta))*scale);
  pontos[2].y = oy - (int)((-xr*sin(theta)+ys*cos(theta))*scale);
  pontos[3].x = ox + (int)((xf*cos(theta)-ys*sin(theta))*scale);
  pontos[3].y = oy - (int)((xf*sin(theta)+ys*cos(theta))*scale);
}

/*************************************************************************
/ DesenhaCarro()                                                        /
/ Esta e uma funcao auxiliar de DesenhaTrajetorias que recebe os pontos  /
/ dos vertices e desenha o veiculo.                                      /
/ Parametros:                                                           /
/   dc - ponteiro do device context onde se deve desenhar                /
/   pontos - vetor dos vertices do retangulo                             /
/ Valor de Retorno: -                                                   /
 *************************************************************************/
void TrajetoDlg::DesenhaCarro(CDC* dc, CPoint pontos[4])
{
  dc->MoveTo(pontos[0]);
  dc->LineTo(pontos[1]);
  dc->LineTo(pontos[2]);
  dc->LineTo(pontos[3]);
  dc->LineTo(pontos[0]);
}

/*************************************************************************
/ GraphPontos()                                                         /
/ Esta e uma funcao auxiliar de DesenhaGrafico que retorna o ponto a ser /
/ plotado conforme o grafico, usando uma funcao de interpolacao de      /
/ FACTDoc                                                               /
/ Parametros:                                                           /
/   pDoc - ponteiro para o objeto FACTDoc                               /
/   nGraph - tipo de grafico sendo plotado                              /
/   T - tempo no qual o ponto deve ser interpolado                      /
/   nCar - 0 para veiculo 1 e 1 para veiculo 2                          /
/   x, y - ponteiros para os valores de retorno do ponto interpolado    /
/ Valor de Retorno: -                                                   /
 *************************************************************************/
void TrajetoDlg::GraphPontos(FACTDoc* pDoc, short nGraph, double T, short nCar, double*
x, double* y)
{
  TrajPoint cTraj;

  pDoc->ResultadoEmT(T,nCar,&cTraj);
  *x = T;
  switch (nGraph)
  {
    case 0:
      *y = sqrt(pow(cTraj.GetVX(TRUE),2) + pow(cTraj.GetVY(TRUE),2));
      break;
    case 1:
      *y = cTraj.GetVX(TRUE);
      break;
    case 2:
```

```
      *y = cTraj.GetVY(TRUE);
      break;
    case 3:
      *y = cTraj.GetX();
      break;
    case 4:
      *y = cTraj.GetY();
      break;
    case 5:
      *y = cTraj.GetW(TRUE);
      break;
    case 6:
      *y = cTraj.GetTheta(TRUE);
      break;
  }
}

void TrajetoDlg::OnContornoCarro1()
{
  bDrawCar1 = !bDrawCar1;
  ((CButton*)GetDlgItem(IDC_CARRO1))->SetCheck(bDrawCar1);
  if (nTipoOutput == 0)
    MyInvalidate();
}

void TrajetoDlg::OnContornoCarro2()
{
  bDrawCar2 = !bDrawCar2;
  ((CButton*)GetDlgItem(IDC_CARRO2))->SetCheck(bDrawCar2);
  if (nTipoOutput == 0)
    MyInvalidate();
}

BOOL TrajetoDlg::OnInitDialog()
{
  CDialog::OnInitDialog();

  bDrawCar1 = TRUE;
  bDrawCar2 = TRUE;
  bGraphCar1 = TRUE;
  bGraphCar2 = TRUE;
  PassoAnim = 0;
  Passo = 100;
  nTipoOutput = 0;
  nCodGraph = 0;

  ((CButton*)GetDlgItem(IDC_OPTTRAJETO))->SetCheck(1);
  ((CButton*)GetDlgItem(IDC_CARRO1))->SetCheck(bDrawCar1);
  ((CButton*)GetDlgItem(IDC_CARRO2))->SetCheck(bDrawCar2);
  ((CButton*)GetDlgItem(IDC_OPTGRAFICO))->SetCheck(0);
  ((CComboBox*)GetDlgItem(IDC_TIPOGRAF))->SetCurSel(0);
  ((CButton*)GetDlgItem(IDC_OPTAMBOS))->SetCheck(1);
  ((CButton*)GetDlgItem(IDC_OPTVEIC1))->SetCheck(0);
  ((CButton*)GetDlgItem(IDC_OPTVEIC2))->SetCheck(0);
  return TRUE;  // return TRUE  unless you set the focus to a control
}

void TrajetoDlg::OnExecutarAnim()
{
  if (nTipoOutput == 0)
  {
    PassoAnim = 0.5;
    MyInvalidate();
  }
}

void TrajetoDlg::OnOptgrafico()
{
  nTipoOutput = 1;
  UpdateAllRadio();
  MyInvalidate();
}

void TrajetoDlg::OnOpttrajeto()
{
  nTipoOutput = 0;
  UpdateAllRadio();
  MyInvalidate();
}
```

```
void TrajetoDlg::OnOptveic1()
{
  bGraphCar1 = TRUE;
  bGraphCar2 = FALSE;
  UpdateAllRadio();
  if (nTipoOutput == 1)
    MyInvalidate();
}

void TrajetoDlg::OnOptambos()
{
  bGraphCar1 = TRUE;
  bGraphCar2 = TRUE;
  UpdateAllRadio();
  if (nTipoOutput == 1)
    MyInvalidate();
}

void TrajetoDlg::OnSelendokTipograf()
{
  CComboBox* pList = (CComboBox*)GetDlgItem(IDC_TIPOGRAF);
  nCodGraph = pList->GetCurSel();
  if (nTipoOutput == 1)
    MyInvalidate();
}

void TrajetoDlg::OnOptveic2()
{
  bGraphCar1 = FALSE;
  bGraphCar2 = TRUE;
  UpdateAllRadio();
  if (nTipoOutput == 1)
    MyInvalidate();
}

void TrajetoDlg::UpdateAllRadio()
{
  ((CButton*)GetDlgItem(IDC_OPTTRAJETO))->SetCheck(((nTipoOutput==0)?1:0));
  ((CButton*)GetDlgItem(IDC_OPTGRAFICO))->SetCheck(((nTipoOutput==1)?1:0));
  ((CButton*)GetDlgItem(IDC_OPTVEIC1))->SetCheck(((bGraphCar1 && !bGraphCar2)?1:0));
  ((CButton*)GetDlgItem(IDC_OPTVEIC2))->SetCheck(((bGraphCar2 && !bGraphCar1)?1:0));
  ((CButton*)GetDlgItem(IDC_OPTAMBOS))->SetCheck(((bGraphCar1 && bGraphCar2)?1:0));
}

void TrajetoDlg::MyInvalidate()
{
  CRect figArea;
  (GetDlgItem(IDC_TRAJETO))->GetWindowRect(&figArea);
  ScreenToClient(&figArea);
  figArea.left+=5;
  figArea.top+=5;
  figArea.right-=5;
  figArea.bottom-=5;
  RedrawWindow(&figArea, NULL, RDW_INVALIDATE|RDW_ERASE|RDW_UPDATENOW);
}

// cardata.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CarData Class

#ifndef CARDATADEFS
#define CARDATADEFS

class CarData : public CObject
{
// definições de acesso
friend class CarDatabase;

// dados
public:

  DECLARE_SERIAL(CarData)

  // constantes
  static const short front, rear, side;
protected:
  // nome
  CString name;
```

```
  // rigidez
  double A[3], B[3], G[3];
  // geometria
  double track, ef, er;
  // inércia
  double rgq, massa;
  // posições em relação ao baricentro
  double xr, xf, ys;
  // ponteiro para a lista
  CarData* pNext;

// construtores e destrutores
public:
  CarData();
  virtual ~CarData();

// funções de acesso
public:

  virtual void Serialize( CArchive& ar );
  void Copia(CarData*);
  CString GetName();
  BOOL SetName(const char* n);
  double GetA(short pos);
  BOOL SetA(short pos, double val);
  double GetB(short pos);
  BOOL SetB(short pos, double val);
  double GetG(short pos);
  BOOL SetG(short pos, double val);
  double GetTrack();
  BOOL SetTrack(double val);
  double GetEF();
  BOOL SetEF(double val);
  double GetER();
  BOOL SetER(double val);
  double GetRaioGirQ();
  BOOL SetRaioGirQ(double val);
  double GetMassa();
  BOOL SetMassa(double val);
  double GetXR();
  BOOL SetXR(double val);
  double GetXF();
  BOOL SetXF(double val);
  double GetYS();
  BOOL SetYS(double val);

// funções para o ponteiro da lista
public:
  CarData* GetNext();
protected:
  CarData* SetNext(CarData* pCar);
};

/////////////////////////////////////////////////////////////////////////////
// CarDatabase Class

class CarDatabase : public CObject
{
// dados
protected:
  // manutenção da lista
  CarData* top;
  CarData* cursor;
  long curpos;
  long numCars;

// construtores e destrutores
public:

  DECLARE_SERIAL(CarDatabase)

  CarDatabase();
  ~CarDatabase();

// funções de acesso
public:
  void Flush();
  virtual void Serialize( CArchive& ar );
  long GetNumElements();
  long AddCar(CarData* pCar);
```

```
  BOOL DeleteCar(long pos);
  CarData* GetAt(long pos);
};

#endif

// cardata.cpp : implementation file
//

#include "stdafx.h"
#include "cardata.h"

/////////////////////////////////////////////////////////////////////////////
// CarData
//

const short CarData::front = 0;
const short CarData::rear = 1;
const short CarData::side = 2;

IMPLEMENT_SERIAL(CarData,CObject,1)

CString CarData::GetName()
{
  return name;
}

BOOL CarData::SetName(const char* n)
{
  name = n;
  return TRUE;
}

CarData::CarData()
{
  name = "";
  for (short i=0; i<3; i++)
  {
    A[i]=0;
    B[i]=0;
    G[i]=0;
  }
  track=0;
  ef=0;
  er=0;
  rgq=0;
  massa=0;
  xr=0;
  xf=0;
  ys=0;
  pNext=NULL;
}

CarData::~CarData()
{
}

void CarData::Serialize( CArchive& ar )
{
  if (ar.IsStoring())
  {
    ar<<name;
    for (short i=0; i<3; i++)
    {
      ar<<A[i];
      ar<<B[i];
      ar<<G[i];
    }
    ar<<track;
    ar<<ef;
    ar<<er;
    ar<<rgq;
    ar<<massa;
    ar<<xr;
    ar<<xf;
    ar<<ys;
  }
  else
  {
    ar>>name;
```

```
    for (short i=0; i<3; i++)
    {
      ar>>A[i];
      ar>>B[i];
      ar>>G[i];
    }
    ar>>track;
    ar>>ef;
    ar>>er;
    ar>>rgq;
    ar>>massa;
    ar>>xr;
    ar>>xf;
    ar>>ys;
    pNext=NULL;
  }
}

void CarData::Copia(CarData* origem)
{
  name = origem->name;
  for (short i=0; i<3; i++)
  {
    A[i] = origem->A[i];
    B[i] = origem->B[i];
    G[i] = origem->G[i];
  }
  track = origem->track;
  ef = origem->ef;
  er = origem->er;
  rgq = origem->rgq;
  massa = origem->massa;
  xr = origem->xr;
  xf = origem->xf;
  ys = origem->ys;
  pNext=NULL;
}

double CarData::GetA(short pos)
{
  if ((pos>=0)&&(pos<3))
    return A[pos];
  else
    return -1.0;
}

BOOL CarData::SetA(short pos, double val)
{
  if ((pos>=0)&&(pos<3)&&(val>=0))
  {
    A[pos]=val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetB(short pos)
{
  if ((pos>=0)&&(pos<3))
    return B[pos];
  else
    return -1.0;
}

BOOL CarData::SetB(short pos, double val)
{
  if ((pos>=0)&&(pos<3)&&(val>=0))
  {
    B[pos]=val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetG(short pos)
{
  if ((pos>=0)&&(pos<3))
    return G[pos];
```

```
  else
    return -1.0;
}

BOOL CarData::SetG(short pos, double val)
{
  if ((pos>=0)&&(pos<3)&&(val>=0))
  {
    G[pos]=val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetTrack()
{
  return track;
}

BOOL CarData::SetTrack(double val)
{
  if (val>=0)
  {
    track = val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetEF()
{
  return ef;
}

BOOL CarData::SetEF(double val)
{
  if (val>=0)
  {
    ef = val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetER()
{
  return er;
}

BOOL CarData::SetER(double val)
{
  if (val>=0)
  {
    er = val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetRaioGirQ()
{
  return rgq;
}

BOOL CarData::SetRaioGirQ(double val)
{
  if (val>=0)
  {
    rgq = val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetMassa()
```

```
{
  return massa;
}

BOOL CarData::SetMassa(double val)
{
  if (val>=0)
  {
    massa = val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetXR()
{
  return xr;
}

BOOL CarData::SetXR(double val)
{
  if (val>0)
  {
    xr = val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetXF()
{
  return xf;
}

BOOL CarData::SetXF(double val)
{
  if (val>0)
  {
    xf = val;
    return TRUE;
  }
  else
    return FALSE;
}

double CarData::GetYS()
{
  return ys;
}

BOOL CarData::SetYS(double val)
{
  if (val>0)
  {
    ys = val;
    return TRUE;
  }
  else
    return FALSE;
}


CarData* CarData::GetNext()
{
  return pNext;
}

CarData* CarData::SetNext(CarData* pCar)
{
  CarData* pOld = pNext;
  pNext = pCar;
  return pOld;
}


////////////////////////////////////////////////////////////////////////////
// CarDatabase
//
```

```
IMPLEMENT_SERIAL(CarDatabase,CObject,1)

CarDatabase::CarDatabase()
{
  top = NULL;
  cursor = NULL;
  curpos = 0;
  numCars = 0;
}

CarDatabase::~CarDatabase()
{
  Flush();
}

void CarDatabase::Flush()
{
  CarData* tp;
  while (numCars > 0)
  {
    tp = top->GetNext();
    delete top;
    top = tp;
    numCars--;
  }
  top = NULL;
  cursor = NULL;
  curpos = 0;
  numCars = 0;
}

void CarDatabase::Serialize( CArchive& ar )
{
  long i,j;
  CarData *pCar;

  if (ar.IsStoring())
  {
    ar<<numCars;
    for (i=0; i<numCars; i++)
      ar<<GetAt(i);
  }
  else
  {
    Flush();
    // nao trazemos de volta o numero de carros diretamente
    // para numCars porque a funcao AddCar ja cuida disso
    // no loop a seguir.
    ar>>j;
    for (i=0; i<j; i++)
    {
      pCar = new CarData();
      ar>>pCar;
      AddCar(pCar);
    }
  }
}

long CarDatabase::GetNumElements()
{
  return numCars;
}

// insere no banco de dados, em ordem alfabetica
// (obs: a posicao do cursor de busca da lista e movida para
//  aquela em que o novo carro foi inserido)
long CarDatabase::AddCar(CarData* pCar)
{
  // em qualquer caso, o numero de carros e incrementado
  numCars++;
  // caso 1: a lista esta vazia
  if (top == NULL)
  {
    top = pCar;
    cursor = top;
    curpos = 0;
    return(curpos);
  }
  // caso 2: verifica se o novo carro deve ser o
```

```
  // primeiro da lista
  if (top->GetName() > pCar->GetName())
  {
    pCar->SetNext(top);
    top = pCar;
    cursor = top;
    curpos = 0;
    return(curpos);
  }
  // caso 3: percorre o restante da lista buscando
  // o lugar certo.
  CarData* tp;
  cursor = top;
  curpos = 0;
  tp = cursor->GetNext();
  while (tp != NULL)
  {
    if (tp->GetName() > pCar->GetName())
    {
      pCar->SetNext(tp);
      cursor->SetNext(pCar);
      cursor = pCar;
      curpos++;
      return(curpos);
    }
    cursor = cursor->GetNext();
    tp = cursor->GetNext();
    curpos++;
  }
  cursor->SetNext(pCar);
  cursor = pCar;
  curpos++;
  return(curpos);
}

BOOL CarDatabase::DeleteCar(long pos)
{
  if ((pos>=numCars)||(numCars==0))
    return FALSE;
  // a nao ser no caso que em que o elemento nao existe
  // (tratado acima), o numero total de elementos sempre
  // e subtraido de 1.
  numCars--;
  // se a posicao buscada esta antes do cursor,
  // posiciona o mesmo no inicio da lista;
  if (pos <= curpos)
  {
    cursor = top;
    curpos = 0;
  }
  CarData* tp;
  // caso 1: a posicao e o topo
  if (pos == 0)
  {
    tp = top;
    top = top->GetNext();
    delete tp;
    cursor = top;
    return TRUE;
  }
  // busca a posicao
  while (curpos < (pos-1))
  {
    cursor = cursor->GetNext();
    curpos++;
  }
  // apaga o elemento
  tp = cursor->GetNext();
  cursor->SetNext(tp->GetNext());
  delete tp;
  return TRUE;
}

CarData* CarDatabase::GetAt(long pos)
{
  if ((pos>=numCars)||(numCars==0))
    return NULL;
  // se a posicao buscada esta antes do cursor,
  // posiciona o mesmo no inicio da lista;
  if (pos < curpos)
```

```
  {
    cursor = top;
    curpos = 0;
  }
  // busca a posicao
  while (curpos < pos)
  {
    cursor = cursor->GetNext();
    curpos++;
  }
  return cursor;
}

// Crush.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// Crush Class

#ifndef CRUSHDEFS
#define CRUSHDEFS

class Crush : public CObject
{
  protected:
    double pos;
    double prof;

  public:
    Crush();
    Crush( const Crush&);
    DECLARE_SERIAL(Crush)
    void Serialize(CArchive& ar);

    void SetPos(double);
    void SetProf(double);
    double GetPos(void) const;
    double GetProf(void) const;
    void    operator = (Crush Other);
};

/////////////////////////////////////////////////////////////////////////////
// CrushArray Class

class CrushArray : public CObArray
{
  public:
    CrushArray(void);
    DECLARE_SERIAL(CrushArray)

    void  DeleteAll(void);

    Crush* GetAt(short);
    short  InsertOrd(Crush);
    short  InsertOrd(double, double);
};

#endif //CRUSHDEFS

// crush.cpp : implementation file
//

#include "stdafx.h"
#include "crush.h"

IMPLEMENT_SERIAL( Crush, CObject, 1 )

/////////////////////////////////////////////////////////////////////////////
// Crush Class

Crush::Crush()
{
  pos = 0;
  prof = 0;
}

Crush::Crush( const Crush& Other)
{
  pos = Other.pos;
  prof = Other.prof;
```

```
}

void Crush::SetPos(double p)
{
  pos = p;
}

void Crush::SetProf(double p)
{
  prof = p;
}

double Crush::GetPos(void) const
{
  return(pos);
}

double Crush::GetProf(void) const
{
  return(prof);
}

void Crush::operator = (Crush Other)
{
  pos = Other.pos;
  prof = Other.prof;
}

void Crush::Serialize(CArchive& ar)
{
  CObject::Serialize(ar);
  if(ar.IsStoring())
  {
    ar << pos;
    ar << prof;
  }
  else
  {
    ar >> pos;
    ar >> prof;
  }
}


IMPLEMENT_SERIAL(CrushArray, CObArray, 1)

////////////////////////////////////////////////////////////////////////////
// Crush Class

CrushArray::CrushArray(void)
{
}

void CrushArray::DeleteAll(void)
{
  short i,j;

  j=GetUpperBound();

  for(i=0;i<=j;i++)
  {
    delete(GetAt(0));
    RemoveAt(0);
  }

}

Crush* CrushArray::GetAt(short i)
{
  return((Crush*)(CObArray::GetAt(i)));
}

short CrushArray::InsertOrd(Crush NewCrush)
{
  Crush  posCrush;
  Crush* pCrush = new Crush;
  short   i;

  pCrush->SetPos(NewCrush.GetPos());
  pCrush->SetProf(NewCrush.GetProf());
```

```
   for (i = 0; i<=GetSize(); i++)
     {
     if(i < GetSize())
       {
       posCrush = *(GetAt(i));
       if( NewCrush.GetPos() < posCrush.GetPos() )
         {
         InsertAt(i,pCrush);
         break;
         }
       if( NewCrush.GetPos() == posCrush.GetPos() )
         {
         delete (GetAt(i));
         CObArray::RemoveAt(i);
         InsertAt(i,pCrush);
         break;
         }
       }
     else
       {
       InsertAt(i,pCrush);
       break;
       }
     }
     return i;
}

short CrushArray::InsertOrd(double pos, double prof)
{
  Crush  posCrush;
  Crush* pCrush = new Crush;
  short   i;

  pCrush->SetPos(pos);
  pCrush->SetProf(prof);

  for (i = 0; i<=GetSize(); i++)
    {
    if(i < GetSize())
      {
      posCrush = *(GetAt(i));
      if( pos < posCrush.GetPos() )
        {
        InsertAt(i,pCrush);
        break;
        }
      if( pos == posCrush.GetPos() )
        {
        delete (GetAt(i));
        CObArray::RemoveAt(i);
        InsertAt(i,pCrush);
        break;
        }
      }
    else
      {
      InsertAt(i,pCrush);
      break;
      }
    }
    return i;
}

// trajeto.h : header file
//

#ifndef TRAJETODEF
#define TRAJETODEF

/////////////////////////////////////////////////////////////////////////////
// TrajPoint Class

class TrajPoint : public CObject
{
  protected:
    double t;
    double x, vx;
    double y, vy;
    double theta, w;
```

```
  public:
    TrajPoint();                          // Construtor Default
    TrajPoint(const TrajPoint&);          // Construtor de Copia
    TrajPoint(double tempo, double* v); // Construtor que recebe
                                          // o vetor de solucao
    DECLARE_SERIAL(TrajPoint)
    void Serialize(CArchive& ar);

    void SetT(double);
    void SetX(double);
    void SetVX(double);
    void SetY(double);
    void SetVY(double);
    void SetTheta(double);
    void SetW(double);
    double GetT(void) const;
    double GetX(void) const;
    double GetVX(BOOL kmh=FALSE) const;
    double GetY(void) const;
    double GetVY(BOOL kmh=FALSE) const;
    double GetTheta(BOOL deg=FALSE) const;
    double GetW(BOOL deg=FALSE) const;
    void    operator = (TrajPoint Other);
};

////////////////////////////////////////////////////////////////////////////
// Trajetoria Class

class Trajetoria : public CObArray
{
  public:
    Trajetoria(void);
    DECLARE_SERIAL(Trajetoria)

    void  DeleteAll(void);

    TrajPoint* GetAt(short);
    int Add(TrajPoint*);
};

#endif //TRAJETODEF

// trajeto.cpp : implementation file
//

#include "stdafx.h"
#include "trajeto.h"

#ifndef DEGTORAD
  #define DEGTORAD    0.01745329
  #define RADTODEG   57.29578000
  #define PI          3.14159265358979
#endif  //DEGTORAD

IMPLEMENT_SERIAL( TrajPoint, CObject, 1 )

////////////////////////////////////////////////////////////////////////////
// TrajPoint Class

TrajPoint::TrajPoint()
{
  t = 0;
  x = 0;
  y = 0;
  vx = 0;
  vy = 0;
  theta = 0;
  w = 0;
}

TrajPoint::TrajPoint(const TrajPoint& Other)
{
  t = Other.t;
  x = Other.x;
  y = Other.y;
  vx = Other.vx;
  vy = Other.vy;
  theta = Other.theta;
  w = Other.w;
```

```
}

TrajPoint::TrajPoint(double tempo, double* v)
{
  t = tempo;
  x = v[1];
  y = v[3];
  vx = v[0];
  vy = v[2];
  theta = v[5];
  w = v[4];
}

void TrajPoint::SetT(double p)
{
  t = p;
}

void TrajPoint::SetX(double p)
{
  x = p;
}

void TrajPoint::SetY(double p)
{
  y = p;
}

void TrajPoint::SetVX(double p)
{
  vx = p;
}

void TrajPoint::SetVY(double p)
{
  vy = p;
}

void TrajPoint::SetTheta(double p)
{
  theta = p;
}

void TrajPoint::SetW(double p)
{
  w = p;
}

double TrajPoint::GetT(void) const
{
  return(t);
}

double TrajPoint::GetX(void) const
{
  return(x);
}

double TrajPoint::GetY(void) const
{
  return(y);
}

double TrajPoint::GetVX(BOOL kmh) const
{
  if (kmh)
    return(vx*3.6);
  return(vx);
}

double TrajPoint::GetVY(BOOL kmh) const
{
  if (kmh)
    return(vy*3.6);
  return(vy);

}

double TrajPoint::GetTheta(BOOL deg) const
{
```

```
  if (deg)
    return(theta*RADTODEG);
  return(theta);
}

double TrajPoint::GetW(BOOL deg) const
{
  if (deg)
    return(w*RADTODEG);
  return(w);
}

void TrajPoint::operator = (TrajPoint Other)
{
  t = Other.t;
  x = Other.x;
  y = Other.y;
  vx = Other.vx;
  vy = Other.vy;
  theta = Other.theta;
  w = Other.w;
}

void TrajPoint::Serialize(CArchive& ar)
{
  CObject::Serialize(ar);
  if(ar.IsStoring())
  {
    ar << t;
    ar << x;
    ar << y;
    ar << vx;
    ar << vy;
    ar << theta;
    ar << w;
  }
  else
  {
    ar >> t;
    ar >> x;
    ar >> y;
    ar >> vx;
    ar >> vy;
    ar >> theta;
    ar >> w;
  }
}

IMPLEMENT_SERIAL(Trajetoria, CObArray, 1)

////////////////////////////////////////////////////////////////////////////
// Trajetoria Class

Trajetoria::Trajetoria(void)
{
}

void Trajetoria::DeleteAll(void)
{
  short i,j;

  j=GetUpperBound();

  for(i=0;i<=j;i++)
  {
    delete(GetAt(0));
    RemoveAt(0);
  }
}

TrajPoint* Trajetoria::GetAt(short i)
{
  return((TrajPoint*)(CObArray::GetAt(i)));
}

int Trajetoria::Add(TrajPoint* pTJ)
{
  return(CObArray::Add((CObject*)pTJ));
}
```

```
// PrintDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// MyPrintDlg dialog

class MyPrintDlg : public CDialog
{
// Construction
public:
  MyPrintDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
  //{{AFX_DATA(MyPrintDlg)
  enum { IDD = IDD_PRINTDLG };
  BOOL  m_PrintAng;
  BOOL  m_PrintCar1;
  BOOL  m_PrintCar2;
  BOOL  m_PrintCen;
  BOOL  m_PrintCrush1;
  BOOL  m_PrintCrush2;
  BOOL  m_PrintPosX;
  BOOL  m_PrintPosY;
  BOOL  m_PrintTraj;
  BOOL  m_PrintVel;
  BOOL  m_PrintVX;
  BOOL  m_PrintVY;
  BOOL  m_PrintW;
  //}}AFX_DATA


// Overrides
public:

protected:
  virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

// Implementation
protected:

  // Generated message map functions
  //{{AFX_MSG(MyPrintDlg)
    // NOTE: the ClassWizard will add member functions here
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

// printdlg.cpp : implementation file
//

#include "stdafx.h"
#include "fact.h"
#include "printdlg.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// MyPrintDlg dialog


MyPrintDlg::MyPrintDlg(CWnd* pParent /*=NULL*/)
  : CDialog(MyPrintDlg::IDD, pParent)
{
  //{{AFX_DATA_INIT(MyPrintDlg)
  m_PrintAng = FALSE;
  m_PrintCar1 = FALSE;
  m_PrintCar2 = FALSE;
  m_PrintCen = FALSE;
  m_PrintCrush1 = FALSE;
  m_PrintCrush2 = FALSE;
  m_PrintPosX = FALSE;
  m_PrintPosY = FALSE;
  m_PrintTraj = FALSE;
  m_PrintVel = FALSE;
  m_PrintVX = FALSE;
  m_PrintVY = FALSE;
  m_PrintW = FALSE;
```

```
  //}}AFX_DATA_INIT
}


void MyPrintDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(MyPrintDlg)
  DDX_Check(pDX, IDC_PRINTANG, m_PrintAng);
  DDX_Check(pDX, IDC_PRINTCAR1, m_PrintCar1);
  DDX_Check(pDX, IDC_PRINTCAR2, m_PrintCar2);
  DDX_Check(pDX, IDC_PRINTCEN, m_PrintCen);
  DDX_Check(pDX, IDC_PRINTCRUSH1, m_PrintCrush1);
  DDX_Check(pDX, IDC_PRINTCRUSH2, m_PrintCrush2);
  DDX_Check(pDX, IDC_PRINTPX, m_PrintPosX);
  DDX_Check(pDX, IDC_PRINTPY, m_PrintPosY);
  DDX_Check(pDX, IDC_PRINTTRAJ, m_PrintTraj);
  DDX_Check(pDX, IDC_PRINTVEL, m_PrintVel);
  DDX_Check(pDX, IDC_PRINTVX, m_PrintVX);
  DDX_Check(pDX, IDC_PRINTVY, m_PrintVY);
  DDX_Check(pDX, IDC_PRINTW, m_PrintW);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(MyPrintDlg, CDialog)
  //{{AFX_MSG_MAP(MyPrintDlg)
    // NOTE: the ClassWizard will add message map macros here
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// MyPrintDlg message handlers

// status.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// StatusDlg dialog

class StatusDlg : public CDialog
{
// Construction
public:
  StatusDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
  //{{AFX_DATA(StatusDlg)
  enum { IDD = IDD_STATUS };
  CString m_Label2;
  CString m_Title;
  CString m_Tempo;
  CString m_Value2;
  CString m_TLabel;
  //}}AFX_DATA

// Implementation
protected:
  virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

  // Generated message map functions
  //{{AFX_MSG(StatusDlg)
  virtual void OnCancel();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

// status.cpp : implementation file
//

#include "stdafx.h"
#include "fact.h"
#include "status.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
```

```
// StatusDlg dialog


StatusDlg::StatusDlg(CWnd* pParent /*=NULL*/)
  : CDialog()
{
  Create(StatusDlg::IDD, pParent);

  //{{AFX_DATA_INIT(StatusDlg)
  m_Label2 = "";
  m_Title = "";
  m_Tempo = "";
  m_Value2 = "";
  m_TLabel = "";
  //}}AFX_DATA_INIT
}

void StatusDlg::DoDataExchange(CDataExchange* pDX)
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(StatusDlg)
  DDX_Text(pDX, IDC_LABEL2, m_Label2);
  DDX_Text(pDX, IDC_TITLE, m_Title);
  DDX_Text(pDX, IDC_TEMPO, m_Tempo);
  DDX_Text(pDX, IDC_VALUE2, m_Value2);
  DDX_Text(pDX, IDC_TLABEL, m_TLabel);
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(StatusDlg, CDialog)
  //{{AFX_MSG_MAP(StatusDlg)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()


/////////////////////////////////////////////////////////////////////////////
// StatusDlg message handlers

void StatusDlg::OnCancel()
{
  CDialog::OnCancel();
}

/////////////////////////////////////////////////////////////////////////////
// mtview.h : interface of the MyTabView class

#ifndef MTVIEW
#define MTVIEW

#include "winlist.h"

class MyTabView : public CView
{
protected: // create from serialization only
  MyTabView();
  DECLARE_DYNCREATE(MyTabView)

// Attributes
private:
  BOOL m_first;

protected:
  int TabSize, x1, y1;
  MyWinList* pChildHead;
  UINT Cur;

// Operations
public:
  UINT HowMany;

  void GetClientRect(LPRECT);
  UINT GetTabIndex();

// Implementation
public:
  virtual ~MyTabView();
  virtual void OnDraw(CDC* pDC);

protected:
    BOOL PreCreateWindow(CREATESTRUCT&);
```

```
    void AddChild(CWnd*, CString, short, UINT, UINT);
    MyWinList* GetChild(UINT);
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    virtual void OnChangeTab(UINT*, UINT*);

// Generated message map functions
protected:
  //{{AFX_MSG(MyTabView)
  afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

#endif  // MTVIEW

// mtview.cpp : implementation of the MyTabView class
//

#include "stdafx.h"
#include "mtview.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// MyTabView

IMPLEMENT_DYNCREATE(MyTabView, CView)

BEGIN_MESSAGE_MAP(MyTabView, CView)
  //{{AFX_MSG_MAP(MyTabView)
  ON_WM_LBUTTONUP()
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// MyTabView construction/destruction

MyTabView::MyTabView()
{
  m_first = TRUE;
  pChildHead = new MyWinList(NULL, "", 0, 0, 0);
  TabSize = 4;
  HowMany = 0;
  Cur = 0;  // Dessa forma, pelo menos o PRIMEIRO NAO pode ser gray !!!
}

MyTabView::~MyTabView()
{
}

BOOL MyTabView::PreCreateWindow(CREATESTRUCT& cs)
{
  cs.lpszClass      =      AfxRegisterWndClass(0,      LoadCursor(NULL,      IDC_ARROW),
(HBRUSH)(COLOR_BTNFACE+1), 0);
  return(TRUE);
}

void MyTabView::GetClientRect(LPRECT lpRect)
{
  CWnd::GetClientRect(lpRect);
  lpRect->left += 32;
  lpRect->top += 60;
  lpRect->right -= 32;
  lpRect->bottom -= 32;
}

/////////////////////////////////////////////////////////////////////////////
// MyTabView drawing

void MyTabView::OnDraw(CDC* pDC)
{
  if(m_first)
  {
    // Nessa funcao, nas classes derivadas, devem ser criados e inicializados todos os
    // ponteiros para as ChildView.  Isso nao acarretara problemas de memoria, pois a
    // janela do Windows so e criada ao se chamar a funcao BeBorn.  Isso tambem traz a
    // vantagem de nao se perderem os dados na janela. DEPOIS, DEVE-SE CHAMAR A FUNCAO
```

```
  // DA CLASSE BASE, OU SEJA, ESSA! A funcao AddChild deve ser chamada para cada um.

  MyWinList* cursor = pChildHead;
  int temp;

  while(cursor->pNext != NULL)
  {
    temp = cursor->pNext->MyName.GetLength();
    if(temp > TabSize)
      TabSize = temp;
    cursor = cursor->pNext;
  }
  TabSize += 2;
  m_first = FALSE;
}

CPen* pPen1 = new CPen(PS_SOLID, 1, GetSysColor(COLOR_BTNSHADOW));
CPen* pPen2 = new CPen(PS_SOLID, 1, GetSysColor(COLOR_BTNFACE));
CBrush* pBrush = new CBrush(GetSysColor(COLOR_BTNFACE));
TEXTMETRIC text;
RECT Client;
int x, y, x2, y2, ox, oy;

pDC->SetMapMode(MM_TEXT); // Define algumas cotas uteis
pDC->SetBkMode(TRANSPARENT);
pDC->SelectStockObject(ANSI_FIXED_FONT);
pDC->GetTextMetrics(&text);
CWnd::GetClientRect(&Client);
x1 = text.tmAveCharWidth;
y1 = text.tmHeight;
x2 = Client.right;
y2 = Client.bottom;
TabSize *= x1;
ox = 3*x1;
oy = 4*y1;

pDC->SelectStockObject(WHITE_PEN);  // Desenha borda 3D da janela
pDC->MoveTo(x1+1, int(2.5*y1));
pDC->LineTo(x1+1, y2-y1-1);
pDC->MoveTo(x1, int(2.5*y1));
pDC->LineTo(x1, y2-y1);
pDC->SelectObject(pPen1);
pDC->LineTo(x2-x1, y2-y1);
pDC->LineTo(x2-x1, int(2.5*y1)+2);
pDC->LineTo(x2-x1-2, int(2.5*y1));
pDC->MoveTo(x1+1, y2-y1-1);
pDC->LineTo(x2-x1, y2-y1-1);
pDC->LineTo(x2-x1, int(2.5*y1)+2);
pDC->SelectStockObject(BLACK_PEN);
pDC->MoveTo(x1, y2-y1+1);
pDC->LineTo(x2-x1+1, y2-y1+1);
pDC->LineTo(x2-x1+1, int(2.5*y1)+2);
pDC->LineTo(x2-x1-1, int(2.5*y1));
pDC->SelectStockObject(WHITE_PEN);
pDC->MoveTo(x2-x1-2, int(2.5*y1));
pDC->LineTo(x1, int(2.5*y1));
pDC->MoveTo(x2-x1-2, int(2.5*y1)+1);
pDC->LineTo(x1, int(2.5*y1)+1);

x = x1;
for(UINT i = 0; i < HowMany; i++)  // Desenha borda 3D dos tabs e seu texto
{
  pDC->SetTextColor(GetSysColor(COLOR_BTNTEXT));

  pDC->SelectStockObject(WHITE_PEN);
  pDC->MoveTo(x, int(2.5*y1));
  pDC->LineTo(x, y1/2+2);
  pDC->LineTo(x+2, y1/2);
  pDC->LineTo(x+TabSize, y1/2);
  pDC->SelectObject(pPen1);
  pDC->LineTo(x+TabSize+2, y1/2+2);
  pDC->LineTo(x+TabSize+2, int(2.5*y1));
  pDC->SelectStockObject(BLACK_PEN);
  pDC->MoveTo(x+TabSize+1, y1/2+1);
  pDC->LineTo(x+TabSize+3, y1/2+3);
  pDC->LineTo(x+TabSize+3, int(2.5*y1));

  if(GetChild(i)->grayed)  // se for gray, desenha texto assim
    pDC->SetTextColor(GetSysColor(COLOR_GRAYTEXT));
```

```
    pDC->TextOut(x+x1, y1, GetChild(i)->MyName);

    x += TabSize+4;
  }

  x = x1 + Cur*(TabSize+4); // Destaca tab ativo
  pDC->SelectObject(pPen2);
  pDC->MoveTo(x+1, int(2.5*y1));
  pDC->LineTo(x+TabSize+1, int(2.5*y1));
  pDC->MoveTo(x+2, int(2.5*y1+1));
  pDC->LineTo(x+TabSize+1, int(2.5*y1)+1);
  pDC->SelectObject(pPen1);
  pDC->LineTo(x+TabSize+1, y1/2+2);
  pDC->LineTo(x+TabSize-1, y1/2);
  pDC->SelectStockObject(WHITE_PEN);
  pDC->MoveTo(x+1, int(2.5*y1));
  pDC->LineTo(x+1, y1/2+2);
  pDC->LineTo(x+2, y1/2+1);
  pDC->LineTo(x+TabSize, y1/2+1);

  TabSize/=x1;

  MyWinList* pCur = GetChild(Cur);  // acerta a posicao da janela filha
  x2 -= (2*ox+2);
  y2 -= (3*ox+2);
  x = pCur->b;
  y = pCur->h;
  if(pCur->CropVarFix == -1)
    pCur->CropVarFix = 1;
  if(pCur->CropVarFix == 1 && (x > x2-ox || y > y2-oy))
  {
    if(x > x2)
      x = x2;
    if(y > y2)
      y = y2;

    pCur->CropVarFix = -1;
  }
  if(pCur->CropVarFix == 0)
  {
    x = x2;
    y = y2;
  }
  pCur->pSelf->MoveWindow(ox, oy, x, y);
  pCur->pSelf->ShowWindow(SW_SHOW);

  delete pPen1;  // limpeza
  delete pPen2;
  delete pBrush;
}

////////////////////////////////////////////////////////////////////////////
// List Functions

void MyTabView::AddChild(CWnd* pChild, CString Name, short cvf, UINT b, UINT h)
{
  MyWinList* cursor = pChildHead;
  while(cursor->pNext != NULL)
    cursor = cursor->pNext;
  cursor->pNext = new MyWinList(pChild, Name, cvf, b, h);
  HowMany++;
}

MyWinList* MyTabView::GetChild(UINT n)
{
  ASSERT(n<HowMany);

  MyWinList* cursor = pChildHead;
  for(UINT i = 0; i <= n; i++)
    cursor = cursor->pNext;

  return(cursor);
}

////////////////////////////////////////////////////////////////////////////
// MyTabView message handlers

BOOL MyTabView::PreTranslateMessage(MSG* pMsg)
{
  UINT cur = Cur, last;
```

```
  if (pMsg->message == 256 && pMsg->wParam == 9 && GetKeyState(VK_CONTROL) < 0)
  {  // aceita ctrl+tab pra andar
    last = Cur;

    cur++;
    while(cur < HowMany && GetChild(cur)->grayed)  // se for gray nao entra
      cur++;
    if(cur==HowMany)
      cur = 0;

    OnChangeTab(&cur, &last);
    Cur = cur;
    GetChild(last)->pSelf->ShowWindow(SW_HIDE);
    Invalidate();

    return(TRUE);
  }
  return(CView::PreTranslateMessage(pMsg));
}

void MyTabView::OnLButtonUp(UINT nFlags, CPoint point)
{  // move com o mouse
  UINT cur, last;
  UINT temp = point.x/(TabSize*x1+4);

  if(point.y  <  2.5*y1  &&  point.y  >  y1/2  &&  point.x  >  x1  &&  point.x  <
int(HowMany*(TabSize*x1+4)) && temp != Cur && !(GetChild(temp)->grayed))
  {
    last = Cur;
    cur = temp;

    OnChangeTab(&cur, &last);
    Cur = cur;
    GetChild(last)->pSelf->ShowWindow(SW_HIDE);
    Invalidate();
  }

  CView::OnLButtonUp(nFlags, point);
}

void MyTabView::OnChangeTab(UINT* Cur, UINT* Last)
{
}

UINT MyTabView::GetTabIndex()
{
  return(Cur);
}

// winlist.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// MyWinList Class

class MyWinList
{
  friend class MyTabView;

  public:
    MyWinList(CWnd*, CString, short, UINT, UINT);
    CWnd* pSelf;

  protected:
    MyWinList* pNext;
    CString MyName;
    short CropVarFix;
    UINT b, h;
    BOOL grayed;

  public:
    void SetAsGray(BOOL);
};

// winlist.cpp : implementation file
//

#include "stdafx.h"
#include "winlist.h"
```

```
////////////////////////////////////////////////////////////////////////////
// MyWinList Class

MyWinList::MyWinList(CWnd* self, CString Name, short cvf, UINT l, UINT a)
{
  pSelf = self;
  MyName = Name;
  pNext = NULL;
  CropVarFix = cvf;
  b = l;
  h = a;
  grayed = 0;
}

void MyWinList::SetAsGray(BOOL gray)
{
  grayed = gray;
  (pSelf->GetParent())->Invalidate();
}

////////////////////////////////////////////////////////////////////////////
// MyFrame frame

#include "mainfrm.h"

class MyFrame : public CMainFrame
{
  DECLARE_DYNCREATE(MyFrame)
protected:
  MyFrame();      // protected constructor used by dynamic creation

// Attributes
public:
// Operations
public:

// Implementation
protected:
  virtual ~MyFrame();
  void ActivateFrame(int nCmdShow = -1);

  // Generated message map functions
  //{{AFX_MSG(MyFrame)
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////////

// myframe.h : header file
//

#include "stdafx.h"
#include "myframe.h"

IMPLEMENT_DYNCREATE(MyFrame, CMainFrame)

////////////////////////////////////////////////////////////////////////////
// MyFrame Class

MyFrame::MyFrame()
{
}

MyFrame::~MyFrame()
{
}

void MyFrame::ActivateFrame(int nCmdShow)
{
  CMainFrame::ActivateFrame(SW_SHOWMAXIMIZED);
}

BEGIN_MESSAGE_MAP(MyFrame, CMainFrame)
  //{{AFX_MSG_MAP(MyFrame)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// fact.h : main header file for the FACT application
```

```
//

#ifndef __AFXWIN_H__
  #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

/////////////////////////////////////////////////////////////////////////////
// FACTApp:
// See fact.cpp for the implementation of this class
//

class FACTApp : public CWinApp
{
public:
  FACTApp();

  BOOL isppreview;
  void WinHelp(DWORD dwData, UINT nCmd = HELP_CONTEXT );

// Overrides
  virtual BOOL InitInstance();

// Implementation

  //{{AFX_MSG(FACTApp)
  afx_msg void OnAppAbout();
  afx_msg void OnFileNew();
  afx_msg void OnFileOpen();
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

// fact.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "fact.h"
#include "factdoc.h"
#include "factview.h"
#include "myframe.h"

#include "autodlg.h"
#include "crushdlg.h"
#include "inipdlg.h"
#include "trjdlg.h"

#include <ctl3d.h>

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// FACTApp

BEGIN_MESSAGE_MAP(FACTApp, CWinApp)
  //{{AFX_MSG_MAP(FACTApp)
  ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
  ON_COMMAND(ID_FILE_NEW, OnFileNew)
  ON_COMMAND(ID_FILE_OPEN, OnFileOpen)
  //}}AFX_MSG_MAP
  // Standard print setup command
  ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// FACTApp construction

FACTApp::FACTApp()
{
  isppreview = FALSE;
}

/////////////////////////////////////////////////////////////////////////////
// The one and only FACTApp object
```

```
FACTApp NEAR theApp;

/////////////////////////////////////////////////////////////////////////////
// FACTApp initialization

BOOL FACTApp::InitInstance()
{
  // Standard initialization
  // If you are not using these features and wish to reduce the size
  //  of your final executable, you should remove from the following
  //  the specific initialization routines you do not need.

  Ctl3dRegister(AfxGetInstanceHandle());
  Ctl3dAutoSubclass(AfxGetInstanceHandle());

//  SetDialogBkColor();        // Set dialog background color to gray
  LoadStdProfileSettings();  // Load standard INI file options (including MRU)

  // Register the application's document templates.  Document templates
  //  serve as the connection between documents, frame windows and views.

  CSingleDocTemplate* pDocTemplate;
  pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(FACTDoc),
    RUNTIME_CLASS(MyFrame),     // my SDI frame window
    RUNTIME_CLASS(FACTView));
  AddDocTemplate(pDocTemplate);

  // enable file manager drag/drop and DDE Execute open
  EnableShellOpen();
  RegisterShellFileTypes();

  // simple command line parsing
  if (m_lpCmdLine[0] == '\0')
  {
    // create a new (empty) document
    OnFileNew();
  }
  else
  {
    // open an existing document
    OpenDocumentFile(m_lpCmdLine);
  }

  m_pMainWnd->DragAcceptFiles();

  return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
  CAboutDlg();

// Dialog Data
  //{{AFX_DATA(CAboutDlg)
  enum { IDD = IDD_ABOUTBOX };
  //}}AFX_DATA

// Implementation
protected:
  virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
  //{{AFX_MSG(CAboutDlg)
    // No message handlers
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
  //{{AFX_DATA_INIT(CAboutDlg)
  //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
  CDialog::DoDataExchange(pDX);
  //{{AFX_DATA_MAP(CAboutDlg)
  //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
  //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void FACTApp::OnAppAbout()
{
  CAboutDlg aboutDlg;
  aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// FACTApp commands

void FACTApp::OnFileNew()
{
  UINT cur = 6;

  if (m_pMainWnd)
  {
    FACTView* pFV = (FACTView*)m_pMainWnd->GetWindow(GW_CHILD);

    if(pFV)
    {
      UINT curTab = pFV->GetTabIndex();
      pFV->OnChangeTab(&cur, &curTab);
    }
  }
  CWinApp::OnFileNew();
}

void FACTApp::OnFileOpen()
{
  UINT cur = 6;

  if (m_pMainWnd)
  {
    FACTView* pFV = (FACTView*)m_pMainWnd->GetWindow(GW_CHILD);

    if(pFV)
    {
      UINT curTab = pFV->GetTabIndex();
      pFV->OnChangeTab(&cur, &curTab);
    }
  }

  CWinApp::OnFileOpen();
}

// os defines desta parte (para a funcao WinHelp) foram obtidos a partir dos
// arquivos:
//   msvc\mfc\include\afxhelp.hm
//   hlp\fact.hm
//
#ifndef __WINHELP_DEFINES__
#define __WINHELP_DEFINES__
#define FIRST_MFC_MENUID 98308
#define LAST_MFC_MENUID 98353
#define FIRST_NONCLIENTID 0x40000
#define LAST_NONCLIENTID 0x40012
#define PPREVIEW_CODE 387074
#define __HIDD_AUTO 0x20066
#define __HIDD_CRUSH 0x2006B
#define __HIDD_INIPOS 0x2006C
#define __HIDD_TRAJETO 0x2006F
#endif

void FACTApp::WinHelp(DWORD dwData, UINT nCmd /* = HELP_CONTEXT */)
{
  BOOL nonwindow = 0;

  if ((dwData >= FIRST_MFC_MENUID && dwData <= LAST_MFC_MENUID) ||
```

```
      (dwData >= FIRST_NONCLIENTID && dwData <= LAST_NONCLIENTID))
    nonwindow = 1;

  if (isppreview)
    dwData = PPREVIEW_CODE;

  if (!isppreview && !m_bHelpMode  && !nonwindow)
  {
    CWnd* pHelpWnd = m_pMainWnd->GetFocus();
    CWnd* pPWnd    = pHelpWnd->GetParent();
    if(pHelpWnd->IsKindOf(RUNTIME_CLASS(AutoDlg))                ||                pPWnd-
>IsKindOf(RUNTIME_CLASS(AutoDlg)))
      dwData = __HIDD_AUTO;
    if(pHelpWnd->IsKindOf(RUNTIME_CLASS(CrushDlg))               ||                pPWnd-
>IsKindOf(RUNTIME_CLASS(CrushDlg)))
      dwData = __HIDD_CRUSH;
    if(pHelpWnd->IsKindOf(RUNTIME_CLASS(InitPosDlg))             ||                pPWnd-
>IsKindOf(RUNTIME_CLASS(InitPosDlg)))
      dwData = __HIDD_INIPOS;
    if(pHelpWnd->IsKindOf(RUNTIME_CLASS(TrajetoDlg))             ||                pPWnd-
>IsKindOf(RUNTIME_CLASS(TrajetoDlg)))
      dwData = __HIDD_TRAJETO;
  }

  CWinApp::WinHelp(dwData, nCmd);
}

// mainfrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
  CMainFrame();
  DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Implementation
public:
  virtual ~CMainFrame();
#ifdef _DEBUG
  virtual void AssertValid() const;
  virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
  CStatusBar  m_wndStatusBar;
  CToolBar    m_wndToolBar;

// Generated message map functions
protected:
  //{{AFX_MSG(CMainFrame)
  afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    // NOTE - the ClassWizard will add and remove member functions here.
    //    DO NOT EDIT what you see in these blocks of generated code!
  //}}AFX_MSG
  DECLARE_MESSAGE_MAP()

void OnSysColorChange();

};
/////////////////////////////////////////////////////////////////////////////

// mainfrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "fact.h"

#include "mainfrm.h"

#include <ctl3d.h>

#ifdef _DEBUG
```

```
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
  //{{AFX_MSG_MAP(CMainFrame)
    // NOTE - the ClassWizard will add and remove mapping macros here.
    //    DO NOT EDIT what you see in these blocks of generated code !
  ON_WM_CREATE()
  //}}AFX_MSG_MAP
  // Global help commands
  ON_COMMAND(ID_HELP_INDEX, CFrameWnd::OnHelpIndex)
  ON_COMMAND(ID_HELP_USING, CFrameWnd::OnHelpUsing)
  ON_COMMAND(ID_HELP, CFrameWnd::OnHelp)
  ON_COMMAND(ID_CONTEXT_HELP, CFrameWnd::OnContextHelp)
  ON_COMMAND(ID_DEFAULT_HELP, CFrameWnd::OnHelpIndex)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// arrays of IDs used to initialize control bars

// toolbar buttons - IDs are command buttons
static UINT BASED_CODE buttons[] =
{
  // same order as in the bitmap 'toolbar.bmp'
  ID_FILE_NEW,
  ID_FILE_OPEN,
  ID_FILE_SAVE,
    ID_SEPARATOR,
  ID_FILE_PRINT,
  ID_APP_ABOUT,
  ID_CONTEXT_HELP,
};

static UINT BASED_CODE indicators[] =
{
  ID_SEPARATOR,           // status line indicator
  ID_INDICATOR_CAPS,
  ID_INDICATOR_NUM,
  ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
  // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
  if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
    return -1;

  if (!m_wndToolBar.Create(this) ||
    !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
    !m_wndToolBar.SetButtons(buttons,
      sizeof(buttons)/sizeof(UINT)))
  {
    TRACE("Failed to create toolbar\n");
    return -1;      // fail to create
  }

  if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
      sizeof(indicators)/sizeof(UINT)))
  {
    TRACE("Failed to create status bar\n");
    return -1;      // fail to create
  }
```

```
  return 0;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
  CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
  CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers

void CMainFrame::OnSysColorChange()
{
  CFrameWnd::OnSysColorChange();

  Ctl3dColorChange();
}
```

Apêndice IV. Manual do Usuário

FACT

Manual do Usuário

Índice

1. Usando o FACT

O objetivo deste programa é fornecer uma ferramenta para auxiliar o técnico na análise de colisões de trânsito. O problema que este programa deve resolver é o seguinte: partindo-se de condições iniciais de posição e velocidade dos veículos, encontrar a posição de repouso desses veículos após a colisão.

O programa só é capaz de simular colisões planas (sem capotagem) de dois veículos, sem múltiplas colisões entre os veículos. Os veículos inicialmente não podem estar "derrapando" (velocidade angular inicial é nula). Além disso, assume-se que durante a colisão os pneus estão travados (frenagem total).

Para usar o programa, o usuário deve:

- escolher um veículo do banco de dados ou entrar dados de geometria, inércia e rigidez dos veículos envolvidos usando os tabs Auto 1 e Auto 2 (ver seção 2);
- definir os perfis de deformação de cada veículo com os tabs Deformação 1 e Deformação 2 (ver seção 3);
- especificar as condições iniciais da colisão no tab de Cenário (ver seção 4);
- executar a análise e ver os resultados, clicando no tab de Trajetória (ver seção 5);

Os dados e resultados também podem ser impressos (ver seção 6);

2. Entrando Dados dos Veículos Envolvidos e Editando o Banco de Dados de Veículos

A entrada de dados relacionados aos veículos é feita através dos tabs Auto 1 e Auto 2. Nestes tabs, pode-se escolher um dos veículos pré-existentes no banco de dados, alterar os dados do mesmo e salvar as alterações, ou ainda criar um novo veículo.

As variáveis de cada veículo são divididas em três categorias:

Geometria: são variáveis que descrevem a geometria do carro e a posição dos pneus em relação à posição do baricentro, como indicado na figura existente nestes tabs.

Inércia: massa e raio de giração ao quadrado do veículo.

Rigidez: nesta categoria, estão as variáveis relacionadas ao modelo de deformação utilizado. Para maiores detalhes desta formulação, ver o anexo 2 deste manual. É importante notar que os parâmetros de rigidez são diferentes para colisões frontais, laterais e traseiras.

Digitando-se um nome na caixa de edição de Modelo/Ano, o cursor da caixa de listagem abaixo desta é movido para o nome mais próximo. Assim, pode-se localizar rapidamente um veículo existente. Alternativamente, pode-se clicar diretamente um nome daquela caixa de listagem.

Através do botão Novo, pode-se inserir um veículo novo no banco de dados. O pressionamento deste botão abre a Caixa de Diálogo de Novo Veículo (ver seção 2.1) na qual digita-se o nome do novo veículo e o tipo-base do mesmo.

Os dados do veículo podem ser editados através das caixas de texto apropriadas.
Deve-se observar que as modificações se aplicam apenas ao arquivo de simulação atual. Para tornar as mudanças permanentes no banco de dados de veículos, pressione o botão Salvar.

Finalmente, o botão Apagar permite que o veículo atualmente selecionado seja permanentemente excluído do banco de dados.

Observação: quando é mudado o veículo, são automaticamente mudadas as dimensões utilizadas nos tabs de Perfil de Deformação (ver seção 3) e Cenário (seção 4).

Os dados dos veículos são utilizados para executa a simulação, que é disparada ao clicar no tab de Trajetória (ver seção 5).

2.1. Caixa de Diálogo Novo Veículo

Esta caixa de diálogo permite a criação de um novo veículo do banco de dados. Ela é exibida ao se pressionar o botão Novo dos tabs Auto 1 e Auto 2 (ver seção 2). Na caixa de texto deve-se digitar o nome que identifica o novo veículo.

Os botões de opção abaixo são usados para determinar qual o conjunto de parâmetros será usado como base para o novo veículo:

Nulo - deixa todos os parâmetros em branco

Tipo 1 a 5 - usa um dos padrões de veículos, de acordo com o tamanho

1 = sub-compacto,

2 = compacto,

3 = médio,

4 = sedan,

5 = grande

Esta classificação de tipos é a mesma utilizada pela NHTSA (National Highway Traffic Safety Agency) dos Estados Unidos.

3. Entrando Dados de Deformação

Os dados de deformação dos veículos são entrados através dos tabs Deformação 1 e 2, respectivamente para os veículos 1 e 2.

Na parte esquerda destes tabs existe uma tabela contendo os pontos do perfil de deformação. Para entrar um novo ponto, basta digitar a posição do ponto e a profundidade nas caixas de texto, e pressionar o botão Inserir.
Para apagar um ponto, inicialmente seleciona-se o ponto a ser apagado e, em seguida, pressiona-se a tecla DELETE ou o botão Apaga.

O posicionamento do perfil de deformação sobre o carro é feito em duas etapas:

Inicialmente, seleciona-se a área em que ocorreu a deformação através das opções de Local da Colisão. Automaticamente, as caixas de texto de Origem são preenchidas e apenas uma delas é habilitada para edição (conforme o local da colisão).

Em seguida, pode-se editar a origem (indicada por um ponto vermelho na figura da parte direita dos tabs) diretamente através das caixas de texto, ou clicando com o mouse na posição desejada.

Os dados dos perfis de deformação são utilizados para executa a simulação, que é disparada ao clicar no tab de Trajetória (ver seção 5).

4. Entrando Dados de Cenário

No tab de cenário são entrados os valores iniciais de posição, orientação e velocidade dos veículos, além de parâmetros globais como coeficiente de atrito.

A edição destes parâmetros é feita através das respectivas caixas de texto, sendo que a figura na parte direita do tab reflete as mudanças conforme estas são executadas.

Tanto neste tab como no tab de saída (trajetória e gráficos), segue-se a convenção:

- vermelho para o veículo 1
- azul para o veículo 2

Os ângulos entrados devem estar entre -180° e 180°, sendo o sentido anti-horário positivo. A origem (ângulo de 0°) é o eixo horizontal.

Um cuidado que se deve ter é que os veículos devem estar inicialmente em contato para que a simulação possa ser executada.

Os dados do cenário são utilizados para executa a simulação, que é disparada ao clicar no tab de Trajetória (ver seção 5).

5. Executando a Simulação e Vendo os Resultados

Uma vez que todos os dados foram inseridos nos tabs anteriores (dados dos veículos, perfis de deformação e cenário), basta clicar no tab de Trajetória para iniciar a simulação.

Várias checagens de consistência dos dados serão efetuadas, de modo que eventualmente uma mensagem de erro pode ser emitida, cancelando a execução da simulação. Estas mensagens são listadas abaixo, juntamente às suas prováveis causas:

"Os perfis de deformação não estão definidos" - um, ou os dois perfis de deformação não foram inseridos. Verifique os tabs de deformação.
"A massa dos veículos deve ser estritamente positiva" - um, ou ambos os veículos estão com massa nula ou negativa. Verifique os tabs Auto 1 e 2.
"Os raios de giração dos veículos devem ser estritamente positivos" - um ou ambos os veículos estão com raios de giração negativos ou nulos. Verifique os tabs Auto 1 e 2.
"A velocidade relativa entre os veículos é nula na condição inicial" - os veículos estão se movendo paralelamente ou parados, no instante inicial. Verifique o tab de cenário.
"Os veículos não estão em contato na condição inicial" - os veículos não estão se tocando no instante inicial. Verifique o tab de cenário.
"Os ângulos devem ser definidos no intervalo [-180, 180]" - os ângulos especificados no tab de cenário devem estar neste intervalo. Verifique o tab de cenário.
"O ângulo da força é inadequado" - o ângulo da força de colisão é inconsistente com as posições, velocidades e ângulos dos veículos conforme especificado no tab de cenário.
"Os perfis de deformação fornecem forças que diferem em mais de 20%" - os valores obtido da integração dos perfis de deformação de cada veículo apresentaram uma grande discrepância. Os resultados da análise podem, em decorrência disso, ser igualmente inconsistentes.
"Um dos perfis de amassamento é muito profundo" - a profundidade máxima dos perfis de amassamento é igual a metade da largura do veículo. Verifique os tabs de Deformação.
"O perfil de amassamento não é compatível com o cenário" - provavelmente, os Locais da Colisão, definidos nos tabs de Deformação, não coincidem com o ponto de contato determinado no tab de Cenário.

Quando a simulação é iniciada, abre-se uma caixa de diálogo de status, que permite o acompanhamento da simulação. O botão Parar permite interromper a simulação a qualquer instante, de modo a visualizar o resultado até aquele instante. Este botão também pode ser usado em casos em que a simulação parece não convergir.

Assim que a simulação termina, o programa alterna para o tab de Trajetória, onde podem ser vistos os resultados. Os controles de opção de Gráfico e Trajetória permitem alternar entre estas saídas. Os outros controles permitem ajustar a exibição das saídas adequadamente. O botão "quadro a quadro" executa a exibição da trajetória com um pequeno intervalo de tempo entre o desenho dos quadros, a fim de facilitar a visualização.

5.1. Caixa de Diálogo Status

A caixa de Diálogo Status é exibida enquanto a simulação é executada, ao se clicar no tab de Trajetória (ver seção 5), para que se possa acompanhar o andamento desta.
Na primeira linha é exibido o estágio em que a simulação se encontra: Colisão ou Trajetória. O estágio de Trajetória se inicia quando a força de colisão pára de atuar.
Na segunda linha desta caixa de diálogo, exibe-se o tempo atual da simulação.
Na terceira linha, apresenta-se uma informação que depende do estágio da simulação:

Durante a colisão, é exibido o ΔV, que é a velocidade relativa entre os pontos de contato dos veículos.

Durante a trajetória é exibida a velocidade máxima absoluta dentre os veículos.

O botão Parar pode ser usado para interromper a simulação a qualquer momento. Neste caso, será exibida a parte da simulação executada até então. No entanto, o programa ainda se comporta como se a simulação não tivesse sido completamente executada, ou seja, toda vez que for exibida o tab de trajetória a simulação será reiniciada.

6. Imprimindo Dados e Resultados

A impressão dos dados e resultados é feita através de comandos do menu Arquivo. O comando Configurar Impressão permite selecionar quais os grupos de dados e saídas devem ser impressos através da caixa de diálogo Configurar Impressão (ver seção 6.1). O comando Visualizar Impressão apresenta na tela um "preview" de como os dados e resultados serão impressos.

Finalmente, o comando Imprimir efetua a impressão propriamente dita. Este comando exibe uma caixa de diálogo onde se pode escolher opções como número de cópias, e qual a impressora será utilizada.

6.1. Caixa de Diálogo Configurar Impressão

As opções desta caixa de diálogo permitem selecionar quais os dados e saídas da simulação devem ser impressos.

Dados

Este grupo contém as opções dos dados de entrada. Selecione cada check box para imprimir um dos conjuntos de dados:

Dados dos Veículos 1 e 2 - correspondem aos dados entrados nos tabs Veículo 1 e 2.

Perfil de Deformação dos Veículos 1 e 2 - são os dados contidos nos tabs de Deformação. O perfil de deformação é impresso como uma tabela de pontos contendo posição e profundidade do amassamento.

Condições Iniciais - corresponde aos dados do tab Cenário.

Saídas

Selecione os check boxes deste grupo para especificar quais as saídas da simulação devem ser impressas:

Trajetórias - imprime as trajetórias dos veículos

Gráficos - cada check box corresponde a um dos gráficos existentes (Velocidades, Velocidades em X e Y, Posição em X e Y, Velocidade Angular e Ângulo)

O botão Cancel fecha a dialog box desfazendo as seleções efetuadas.

7. Comandos de Menu

Comandos do menu Arquivo

O menu Arquivo apresenta os seguintes comandos:

Novo	Cria um novo documento.
Abrir	Abre um documento pré-existente.
Salvar	Salva o documento aberto usando o mesmo nome de arquivo.
Salvar Como	Salva o documento aberto em um arquivo com outro nome.
Imprimir	Imprime dados do documento atualmente aberto..
Visualizar Impressão	Apresenta os dados na tela como se eles fossem impressos.
Configurar Impressão	Seleciona uma impressora a ser utilizada.
Sair	Sai do programa FACT.

Comandos do menu Visualizar

O menu Visualizar contém os seguintes comandos:

Barra de Ferramentas	Exibe ou oculta a barra de ferramentas.
Barra de Status	Controla a exibição da barra de status.

Comandos do menu Help

O menu Help possui os seguintes comandos, que fornecem auxílio no uso deste aplicativo:

Índice	Apresenta um índice de tópicos de ajuda.
Usando o Help	Fornece instruções gerais quanto ao uso da Ajuda.
Sobre	Exibe informações como a versão deste aplicativo.

Anexo 1: Determinação do Ângulo da Força

Para execução da simulação, o programa necessita do ângulo α (ângulo da força principal de colisão durante o impacto) como entrada. Como a obtenção deste ângulo não é imediata e tem grande influência sobre o resultado da simulação, este anexo apresenta algumas diretrizes para auxiliar a sua determinação com base nas evidências do acidente.

Em primeiro lugar, será apresentado um diagrama que mostra uma análise estatística (obtida do artigo "**Comparision and Realism of Crash Simulation Tests and Real Accident Situations for Biomechanical Movements in Car Colisions**" de DIETMAROTTE) mostrando a frequência de ocorrência de impactos em função do ângulo da força e do local da colisão.

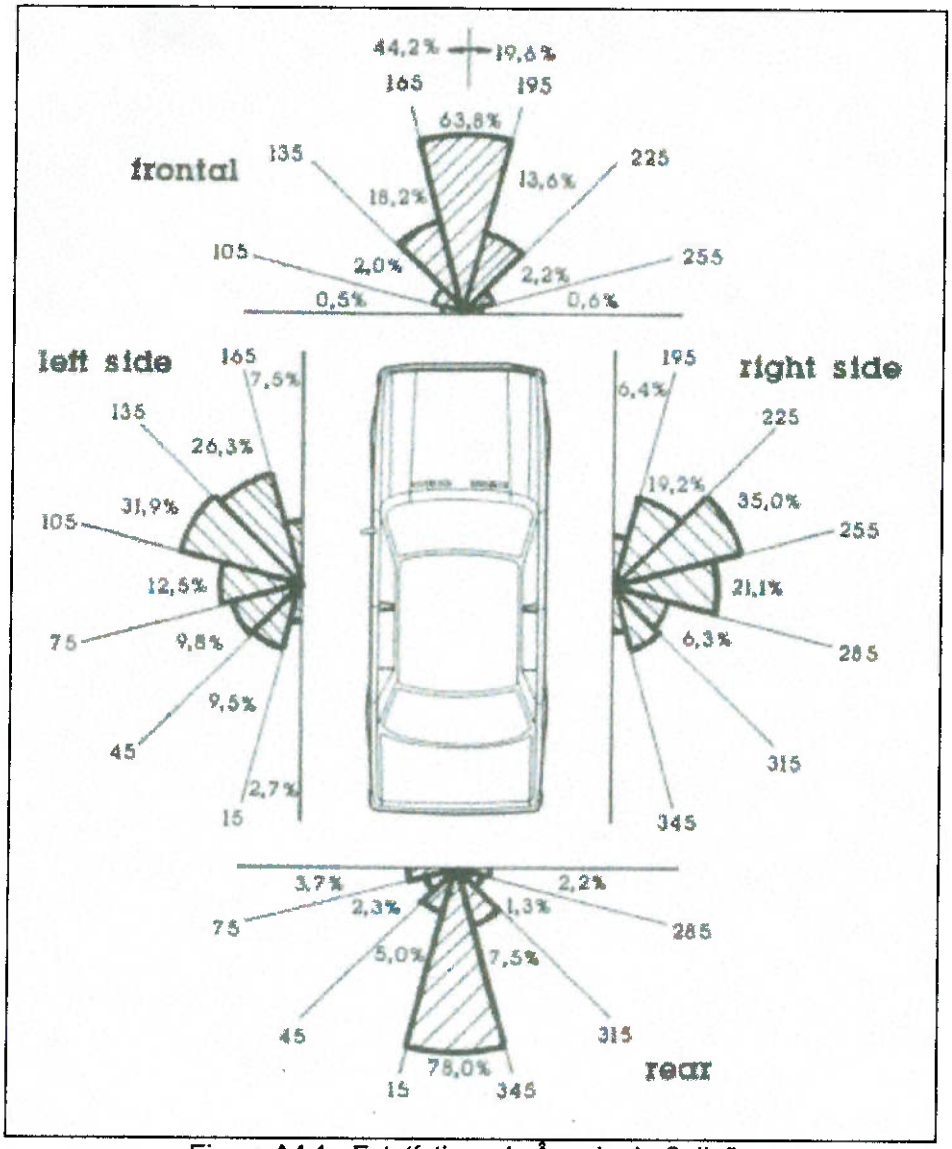


Figura A4.1 - Estatísticas de Ângulo da Colisão

No caso em que o usuário tem uma estimativa do ângulo, mas precisa de mais informações, o diagrama da figura 1 pode vir a ser útil.

De acordo com a NHTSA, a determinação do ângulo da força principal de colisão deve ser feita levando em consideração o maior número possível de evidências do acidente. Ela sugere a análise da trajetória percorrida pelos ocupantes do veículo, análise do ambiente e das trajetórias percorridas pelo veículo e do veículo em si. Esta Organização destaca enfaticamente a natureza vetorial da força de colisão, principalmente por tratar-se da resultante de várias forças envolvidas no impacto. Além disso, estas forças devem observar a terceira lei de Newton, ou seja, devem ser colineares, de sentido oposto e módulo igual. A NHTSA faz as seguintes recomendações com relação às evidências do acidente:

- Quanto à trajetória dos ocupantes do veículo: Se houverem dados, testemunhos ou evidência física (marcas de sangue, deformação da parte interna do veículo, etc.) indicando esta trajetória, a determinação da direção de ação da força é imediata, visto que os ocupantes do veículo respondem à aceleração a ele imposta pela resultante das forças. Embora a interpretação destes dados seja direta, a obtenção correta da trajetória dos ocupantes do veículo é um procedimento difícil e nem sempre possível.
- Quanto ao ambiente, destaca-se a importância da análise das marcas de pneus no solo quando existirem, e das trajetórias percorridas pelos veículos durante e imediatamente após o impacto. A NHTSA recomenda um cuidado especial na interpretação de dados do ambiente, em particular da trajetória e principalmente quando houver rotação significativa dos veículos. Existe um grande número de procedimentos empíricos para análise de marcas de pneu. Para mais informações, consultar o site da North Western University na Internet.
- Quanto à deformação do veículo, pode-se afirmar que este é o principal fator na determinação do ângulo de colisão. Em primeiro lugar, é necessário estabelecer a diferença entre deformação e distorção da estrutura do veículo. Distorção (shifting) da estrutura do veículo é definida pela NHTSA como mudança de direção de elementos da estrutura, em oposição à deformação ou esmagamento (crushing) da estrutura. Além disso, deve-se observar a diferença entre deformação de contato e deformação induzida. Deformação de contato é aquela que ocorre nas regiões onde houve contato entre os dois veículos durante a colisão. Deformação induzida é todo o resto. Nas análise com respeito ao ângulo, o tratamento da deformação e da distorção é distinto, mas ambas são utilizadas. Quanto ao tipo de deformação, no entanto, somente se utiliza deformação de contato na análise, pois dano induzido não fornece uma base sólida de comparação.

Adicionalmente, pode-se mencionar algumas conclusões obtidas por UEYAMA et. al. em seu artigo "**Determination of Collision Configurations from Vehicle Deformation Patterns**".

Após realizar uma série de experimentos com colisões controladas (staged collisions), monitoradas por câmeras de alta velocidade e acelerômetros nos veículos, UEYAMA chegou a uma importante conclusão: na maioria dos acidentes, onde o ângulo da força varia pouco ao longo da colisão e as rotações dos carros até sua separação é pequena, pode-se usar os perfis de deformação para definir o ângulo da colisão da seguinte forma:

De posse dos perfis de deformação de ambos os veículos, encontra-se a posição relativa entre os veículos que permite um melhor "encaixe" desses perfis, ou seja, a posição em que há maior área de contato entre os perfis. Tomando a bissetriz do ângulo formado pelas linhas de simetria longitudinais de ambos os veículos, pode-se tomar a direção da força como perpendicular à esta bissetriz. A figura abaixo ilustra este procedimento.

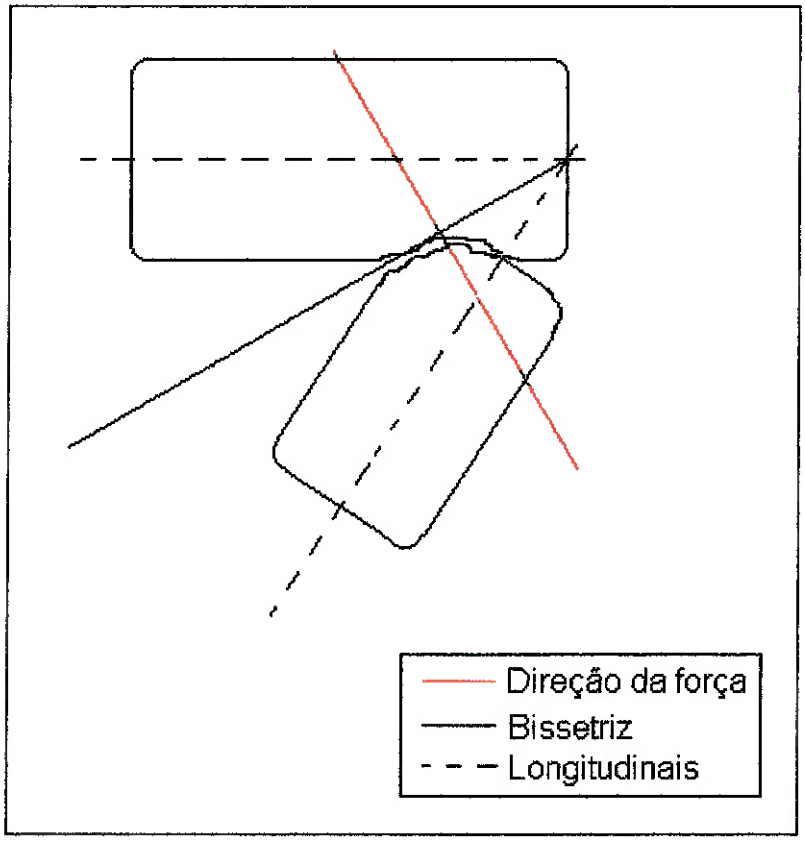


Figura A4.2 - Procedimento de Determinação do Ângulo

Anexo 2: Formulação do Problema

A força que age durante a colisão é calculada em função do perfil de deformação que é entrado nos tabs de Deformação. A relação empírica utilizada é a seguinte:

$$F = \int_0^L (A + B \cdot c)dx$$

onde A e B são os parâmetros de rigidez presentes nos tabs Auto 1 e Auto 2.

L é o comprimento do perfil de deformação,

c é a profundidade do perfil em cada ponto.

O parâmetro G, também presente nos tabs Auto 1 e Auto 2, é utilizado em análises baseadas na energia, e atualmente não é utilizado.

O ponto de contato entre os veículos durante a colisão é considerado como o centro geométrico do perfil de deformação. Em um sistema de coordenadas fixo no veículo, as coordenadas desse centro geométrico são dadas por:

$$x_{cg} = \frac{1}{A}\int_{x_0}^{x_f} xydx$$

$$y_{cg} = \frac{1}{A}\int_{x_0}^{x_f} y^2dx$$

As forças devido à frenagem são dadas pelas equações abaixo. Observa-se que a força de atrito nos pneus age em direção oposta à velocidade do veículo.

$$Fx = -Mg\mu\frac{Vx}{|V|} \qquad p / |V|>0$$

$$Fy = -Mg\mu\frac{Vy}{|V|} \qquad p / |V|>0$$

onde M é a massa do veículo,

g é a aceleração da gravidade

μ é o coeficiente de atrito

O torque que surge devido ao atrito dos pneus é obtido da fórmula abaixo:

$$T = -Mg\mu.(e_F + e_r)\frac{\omega}{|\omega|} \qquad p / |\omega|>0$$

onde e_F e e_r são as distâncias dos eixos frontal e traseiro até o baricentro do veículo,

ω é a velocidade angular do veículo,

T é o torque aplicado.

Desta forma, as equações de movimento do veículo são:

$$\ddot{x}_g = \frac{Fx + F\cos(\alpha)}{M}$$

$$\ddot{y}_g = \frac{Fy + Fsen(\alpha)}{M}$$

$$\ddot{\theta} = T - F\cos(\alpha).yc + Fsen(\alpha).xc$$

onde x_g, y_g são as coordenadas do baricentro do veículo em coordenadas de um referencial externo

θ é o ângulo de orientação do veículo em relação ao eixo horizontal do sistema externo.

Apêndice V. Conteúdo do Disquete em Anexo

Anexo a este trabalho segue um disquete contendo o seguinte:

- matlab.exe - arquivo tipo "zip" *self-extracting* (capaz de se descompactar ao ser executado. Contém os arquivos utilizados para testes no MatLab e Simulink (explicados no Apêndice III).

- release.exe - arquivo tipo "zip" *self-extracting* (capaz de se descompactar ao ser executado. Contém os arquivos necessários para a execução do FACT: O excutável, o arquivo de Help, o banco de dados de veículos (cardata.cdb), a fontes conos.ttf (que deve ser instalada manualmente antes da execução do programa) e 4 arquivos de teste (tipo FCT).

- fonte.exe - arquivo tipo "zip" *self-extracting* (capaz de se descompactar ao ser executado. Contém os arquivos fonte do programa e do Help, incluido os recursos gráficos utilizados. No início do Apêndice III pode-se encontrar uma descrição do conteúdo dos principais arquivos fonte.