Title

Models Of Vehicular Collision: Development And Simulation With Emphasis On Safety IV: An Improved Algorithm For Detecting Contact Between Vehicles

Authors

O'Reilly, Oliver M.
Papadopoulos, Panayiotis
Lo, Gwo-jeng
et al.

Publication Date

1998-06-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

Models of Vehicular Collision: Development and Simulation with Emphasis on Safety IV: An Improved Algorithm for Detecting Contact Between Vehicles

Oliver M. O'Reilly, Panayiotis Papadopoulos, Gwo-Jeng Lo, Peter C. Varadi
University of California, Berkeley

California PATH Research Report
UCB-ITS-PRR-98-25

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

Report for MOU 309

June 1998

ISSN 1055-1425

Models of Vehicular Collision: Development and Simulation with Emphasis on Safety IV: An Improved Algorithm for Detecting Contact Between Vehicles

REPORT – June 1998

Submitted to: PATH (MOU 309)

Oliver M. O'Reilly (PI)
Panayiotis Papadopoulos (PI)
Gwo-Jeng Lo
Peter C. Varadi

Department of Mechanical Engineering
University of California, Berkeley

Abstract

In previous works, the authors developed a computational model for collision detection where each vehicle was modeled as an ellipsoid. This model does not accurately reflect the geometry of realistic vehicles, yet it possesses significant theoretical and computational advantages over other possible models. This report outlines a novel procedure for detecting the geometry of the contact interface between vehicles which employs a better approximation of their actual shape yet it also preserves the advantage of the ellipsoidal model.

Keywords: IVHS America, Vehicle Dynamics, Collision Dynamics, Safety, Computer Simulation, Animation and Simulation

Executive Summary

In previous works, the authors developed a computational model for collision detection where each vehicle was modeled as an ellipsoid. This model does not accurately reflect the geometry of realistic vehicles, yet it possesses significant theoretical and computational advantages over other possible models. This brief report outlines a novel procedure for detecting the geometry of the contact interface between vehicles which employs a better approximation of their actual shape yet it also preserves the advantage of the ellipsoidal model.

The procedure, known as BM, uses the vehicle ellipsoid to construct a box-like region representing the lateral surface of the vehicle. For two contacting vehicles, the corresponding ellipsoids are used to determine the unique contact point. However, in contrast to our earlier works, the unit normal at the contact point is determined using the box-like region.

Contents

1 Introduction

The model for the detection of vehicular collision used in MEDUSA is discussed in O'Reilly, Papadopoulos, Lo and Varadi [1]. We henceforth refer to this model as the ellipsoid model (EM). In that work, the positions of the contact points and the corresponding outward unit normal vectors were defined using the surfaces of ellipsoids. The approximation of the geometry of a vehicle as an ellipsoid is not only shape-preserving under the homogeneous deformation (see Truesdell and Toupin [4]), but is also numerically convenient for contact detection because of the convexity of an ellipsoid (see O'Reilly, Papadopoulos, Lo and Varadi [2]). However, the use of EM yields unrealistic solutions in simulations of offset head-on-tail collisions between two vehicles. The reason for this can be easily understood from Figure 1, where the vehicles in the simulation rotate counterclockwise due to the definition of the contact normal. Here, the normal to the ellipsoid is also used to specify the direction of the contact force between the contacting vehicles. Instead, when two vehicles collide under these conditions, they should realistically rotate clockwise as a result of the line of action of the resultant contact forces.

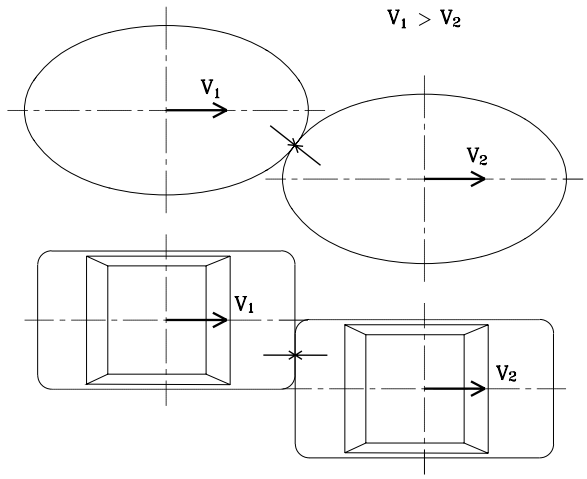

Figure 1: *The difference between the contact normal vectors using ellipsoids and the actual vehicles.*

In order to remedy this deficiency and further improve the geometric modeling of a vehicle, a new model for the low relative velocity vehicular

collision is proposed. It is based on a box model for the outer surface of the vehicle, and is referred to here as BM.

2 A New Model for Contact Detection

In order to retain the benefits of the ellipsoidal representation, while both fixing the error arising in the direction of the contact forces and improving the overall geometric modeling of a vehicle, the contact detection procedure still takes advantage of the ellipsoid. To this end, the positions of the contact points are defined on the surfaces of the ellipsoids as in O'Reilly, Papadopoulos, Lo and Varadi [2]. However, instead of using the corresponding outward unit normal vectors at the positions of the contact points, the contact normal vectors in the new vehicle model are defined as the outward unit normal vectors of the surface of a box with rounded corners as depicted in Figure 2, where R is the radius for all mollified corners.

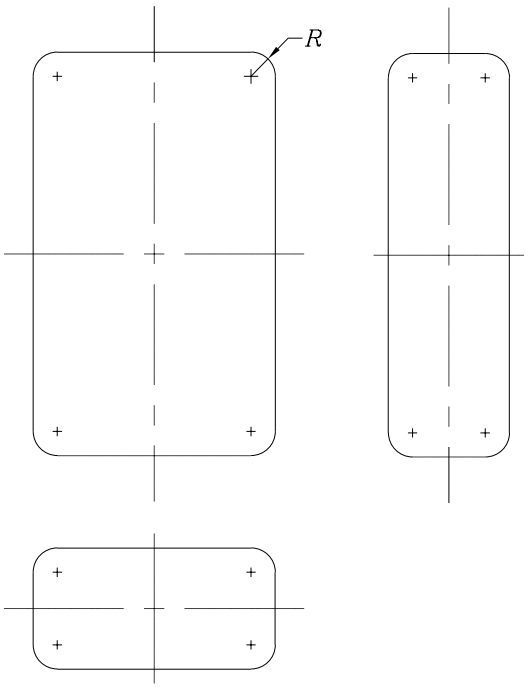

Figure 2: *The modified geometrical model of a vehicle.*

Since the ellipsoid in the current configuration is well-defined under the homogeneous deformation (for details, see O'Reilly, Papadopoulos, Lo and Varadi [1]), the corresponding three-dimensional box can be constructed using a unique mapping in which the ellipsoid and the box share the same volume, aspect ratios and principal directions. In addition, if the vehicle has a homogeneous mass density, the box and ellipsoid will have the same total mass. The reason for the existence of the rounded corners in the modified geometrical model is because the unit normal vector is not uniquely defined at sharp corners. However, it can be computed for smooth surfaces which include the rounded corners in Figure 2.

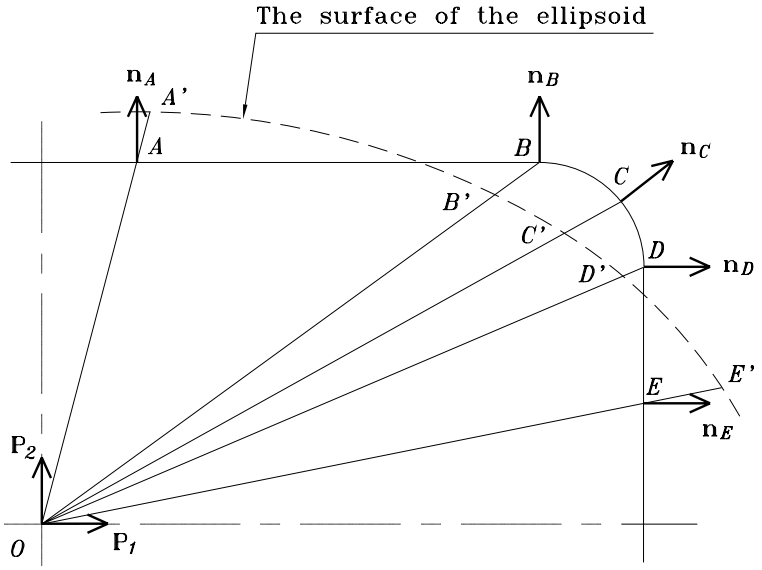

Figure 3: *The outward unit normal vectors on the surface of the modified vehicle model.*

Suppose the position of the contact point is detected as one of the following five points: A', B', C', D' and E' as in Figure 3. The corresponding five contact normal vectors $\mathbf{n}_A$, $\mathbf{n}_B$, $\mathbf{n}_C$, $\mathbf{n}_D$ and $\mathbf{n}_E$ are defined as the outward unit normal vectors at points A, B, C, D and E which are the intersection points of the surface of the box and the lines joining the original contact points and the center of mass of the ellipsoid. Note that in the event of a homogeneous mass distribution, this mass center coincides with

the geometrical center of the corresponding box. In Figure 3, the point O and the vectors $\mathbf{P}_1$ and $\mathbf{P}_2$ are the mass center and the normalized principal vectors of the ellipsoid, respectively. The third principal vector, $\mathbf{P}_3$, which is not shown in Figure 3, is normal to $\mathbf{P}_1$ and $\mathbf{P}_2$. These three principal vectors of the ellipsoid in the current configuration form a orthonormal basis to represent the contact normal vectors of the modified vehicle model. The intersection points A and E are located on the surfaces of planes. Therefore, the contact normal vectors are the plane normal vectors of the corresponding planes which are exactly the principal directions of the ellipsoid: $\mathbf{n}_A = \mathbf{P}_2$ and $\mathbf{n}_E = \mathbf{P}_1$. As for the points B, C and D, which are located in the corner region of the box, the contact normal vectors $\mathbf{n}_B$, $\mathbf{n}_C$ and $\mathbf{n}_D$ are the outward unit normal vectors on the surfaces of the corresponding cylinders or spheres. Notice that the tangent vectors at the points from the planes to the corner regions are continuous. This guarantees the continuity of the contact normal vectors along the surface of the modified model. Thus, $\mathbf{n}_A = \mathbf{n}_B$ and $\mathbf{n}_D = \mathbf{n}_E$ where the points B and D are at the interface of the mollified surfaces and the planes.

After the contact normal vector of an individual vehicle has been defined, the directions of the contact forces between impacting vehicles for several different scenarios can be defined as in Figure 4. Consider case I of Figure 4: this is the typical case which distinguishes between EM's and BM's determination of the directions of the contact forces. The contact points on each vehicle are located in the plane regions of the corresponding boxes and the three principal directions of the two vehicles are parallel, so that $\mathbf{n}_1$, the contact normal vector of the contact force acting on vehicle 2 from vehicle 1, is the corresponding principal direction, ${}_1\mathbf{P}_1$, of vehicle 1. The vector ${}_\alpha\mathbf{P}_1$ denotes the principal vector which corresponds to the maximum principal length of vehicle α. Notice that the contact normal vectors of the contact forces acting on vehicle 1 due to vehicle 2 for all cases are determined by Newton's third law; thus $\mathbf{n}_2 = -\mathbf{n}_1$. Case II is similar to case I, except that the principal directions of the two vehicles are not parallel. In this case, if the angle α_1 in Figure 4 is less than α_2, then the contact normal vector $\mathbf{n}_1$ will be the direction of the corresponding principal vector of vehicle 1, i.e., ${}_1\mathbf{P}_1$. Case III is a plane versus corner case in which the vector $\mathbf{n}_1$ is determined by the corresponding principal vector of the vehicle with its contact points projected in-plane. Thus, the contact normal vector $\mathbf{n}_1$ in case (III) of Figure 4 is equal to ${}_1\mathbf{P}_1$. The last case, IV, is a corner versus corner case where the outward unit normal vectors at the corners of vehicle 1 and 2 are $\hat{\mathbf{n}}_1$ and $\hat{\mathbf{n}}_2$, respectively. In such a case, the vector $\mathbf{n}_1$ is defined as $\mathbf{n}_1 = (\hat{\mathbf{n}}_1 - \hat{\mathbf{n}}_2)/||\hat{\mathbf{n}}_1 - \hat{\mathbf{n}}_2||$.

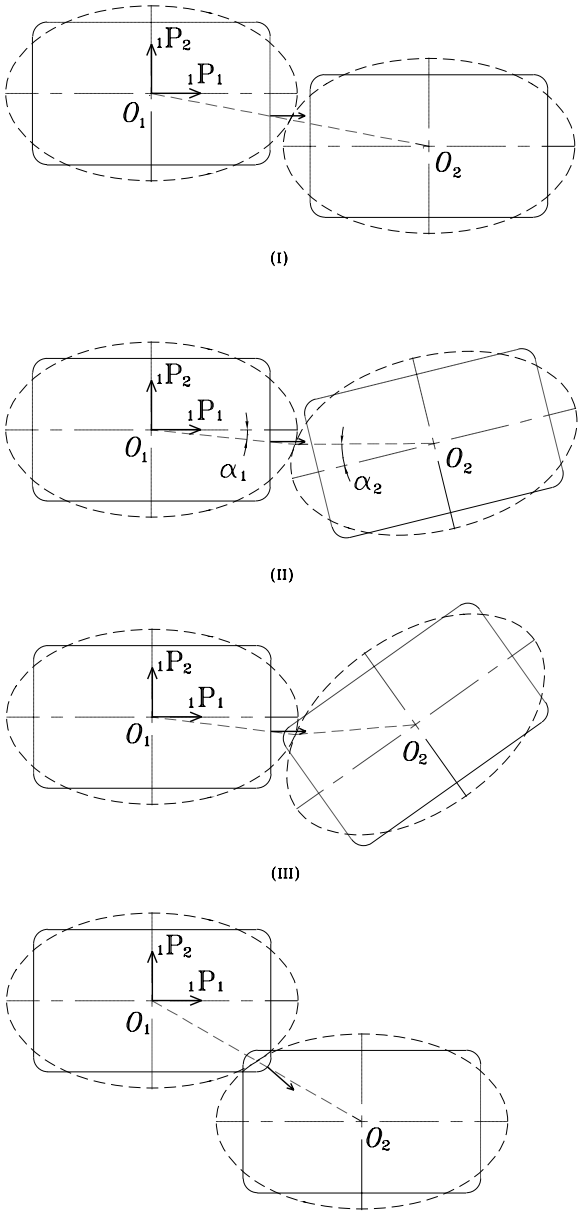

Figure 4: *Contact normal vectors determined by the box model (BM) for different contact scenarios: (i)* $\mathbf{n}_1 = {}_1\mathbf{P}_1$*, (ii)* $\mathbf{n}_1 = {}_1\mathbf{P}_1$ *when* $\alpha_1 < \alpha_2$*, (iii)* $\mathbf{n}_1 = {}_1\mathbf{P}_1$*, and (iv)* $\mathbf{n}_1$ *is an average.*

3 Computer Implementation of the Contact Algorithm

The changes to the computer code MEDUSA, where the contact algorithm based on BM is implemented, are restricted to the calculation of the normal vector to the contact surface of the colliding vehicles. The remainder of the program remains unaltered. Only the functions `info()` and `eig()` which are included in Appendix B, are added to `detect_contact`. The function `info()` calculates the matrix $\hat{\mathbf{K}}$ which contains the information pertaining to the principal directions and principal semi-axes of the ellipsoid in the current configuration (for details, see O'Reilly, Papadopoulos, Lo and Varadi [2]). The function `eig()` finds the eigenvalues and eigenvectors of the matrix $\hat{\mathbf{K}}$. The major modified part of the program is contained in the function `norm()` of the file *contact.c* which is included in this report as Appendix B.

4 Computer Simulation

Two illustrative examples of vehicular collision are presented in this section. Each ellipsoid in the following simulation represents a vehicle. In the first example, which is mentioned in Section 1, differences in the impact scenario using the current (BM) and previous (EM) versions of the contact algorithm are shown. The second example shows how the algorithm based on BM avoids incorrect contact detection since it still detects the positions of contact points on the surface of an ellipsoid. The vehicle parameters for both examples are listed in Appendix A. Further, the variables R, the radii of the corner regions of a vehicle, are set to be equal to 0.3 meters which are given in the function `norm()`.

The motivation for the development for the new model of contact detection comes from the first case where two vehicles are driving along the same direction at speeds of 65 miles per hour (mph) and 60 mph, respectively. The impact scenario shown in Figure 5, which uses EM to calculate the contact normal vectors. It results in an incorrect rotation of the vehicles which can be attributed to the geometry of the ellipsoids. The physical motion of the vehicles in this situation should result in a clockwise rotation as presented in Figure 6. This is predicted when the contact normal vectors are defined on the surfaces of the three-dimensional boxes, i.e., the BM is used.

In the second example, the two vehicles are still moving with the same direction and velocity as those in the first case, however, the distance be-

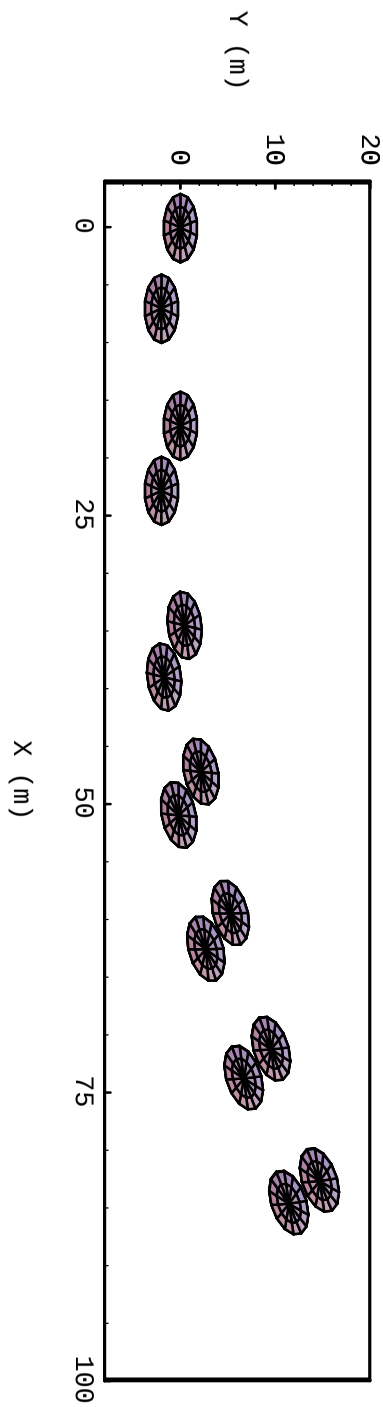

Figure 5: *Plan view of the pre-collision, collision, post-collision of two vehicles in an offset head-on-tail collision using the EM based contact detection algorithm. The non-realistic counterclockwise rotation should be noted.*

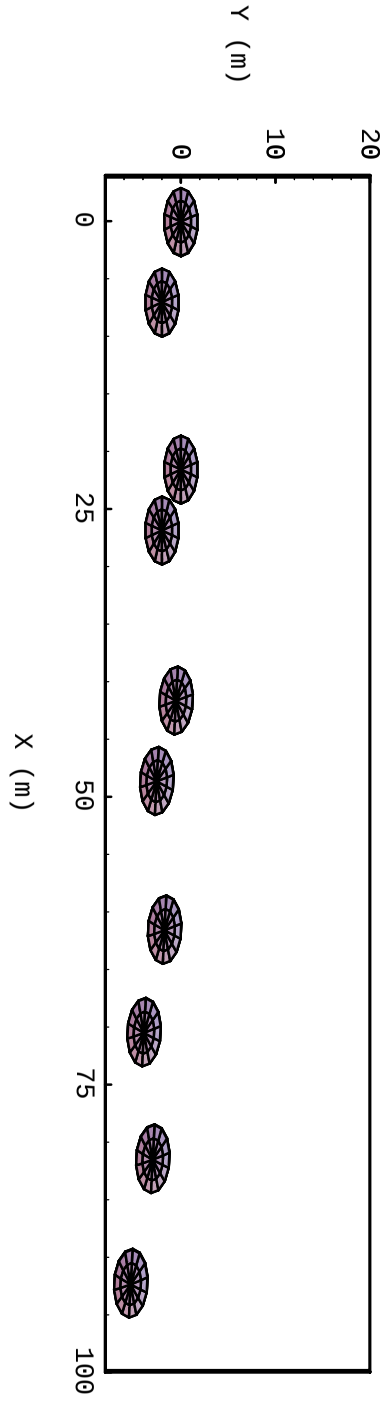

Figure 6: *Plan view of the pre-collision, collision, post-collision of two vehicles in an offset head-on-tail collision using the BM based contact detection algorithm.*

tween their mass centers is such that two ellipsoids intersect, however the corresponding boxes don't (cf. Figure 7).

To avoid inaccurate contact detection, the new algorithm, which uses the BM, detects no contact if

$$ {}_1\mathbf{P}_2 \text{ is parallel to } {}_2\mathbf{P}_2 \quad \text{and} \quad \mathrm{d} > \frac{1}{2}(\mathrm{B}_1 + \mathrm{B}_2) , \tag{1} $$

where ${}_1\mathbf{P}_2$, ${}_2\mathbf{P}_2$, B_1 and B_2 are the principal vectors and the widths of the two vehicles in the current configuration, respectively. The variable d is the component of the distance between the mass centers of the two vehicles in the ${}_1\mathbf{P}_2$ direction.

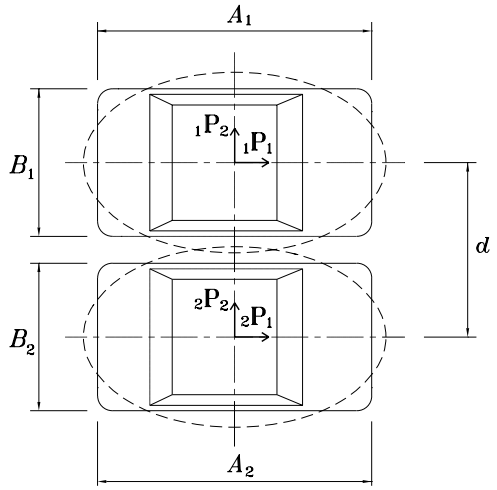

Figure 7: *A difference between the contact detection based on the ellipsoidal model and the box model.*

5 Conclusions

This work describes a new model which represents the realistic three dimensional shapes of vehicles while retaining the advantage of parameterizing their motions and deformations using ellipsoids and uniform deformation fields, respectively. The representative simulations reveal that the new model predicts physically plausible post-collision vehicular motions.

References

[1] O. M. O'Reilly, P. Papadopoulos, G.-J. Lo and P. C. Varadi. *Models of Vehicular Collision: Development and Simulation with Emphasis on Safety. II: On the Modeling of Collision between Vehicles in a Platoon System.* California PATH Research Report UCB-ITS-PRR 97-34, 1997.

[2] O. M. O'Reilly, P. Papadopoulos, G.-J. Lo and P. C. Varadi. *Models of Vehicular Collision: Development and Simulation with Emphasis on Safety. III: Computer Code, Programmer's Guide and User's Manual for MEDUSA.* California PATH Research Report UCB-ITS-PRR 98-10, 1998.

[3] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. *Numerical Recipes in C: the Art of Scientific Computing.* 2nd ed., Cambridge University Press, Cambridge, 1992.

[4] C. Truesdell and R. A. Toupin. The Classical Field Theories, in *Handbuch der Physik.* Vol. III/1, pp. 226-858, edited by S. Flügge, Springer-Verlag, Berlin, 1960.

A Vehicle Parameters

The following are the input data files *model.dat*, *platoon1.dat* and *platoon2.dat* for the computer simulation examples described in Section 4. The two examples share the same file *model.dat*. The files *platoon1.dat* and *platoon2.dat* in Sections A.2 and A.3 are used in the first and second examples, respectively.

A.1 *model.dat*

```
%
% Physical parameters of vehicle model for the examples in Section 4
%
NUMBER_OF_MODELS 1

% description of Model 1 starts here
MODEL 1
 COSSERAT_POINT
  MASS 1573.0  % mass of car [kg]
  Ix    479.6  % moments of inertia along principal axes [kg m^2]
  Iy   2594.6
  Iz   2782.0
  E   200.0e6  % Young's modulus [N/m^2]
  nu  0.30     % Poisson's ratio
  volume 0.42  % assumed volume of the Chassis [m^3]
 SUSPENSION
  L1 1.034     % distance from cg to front axle [m]
  L2 1.491     % distance from cg to rear axle [m]
  B  0.725     % track of axle [m]
  H1 0.0       % vertical distance from cg to front assembly pts.
  H2 0.0       % and to rear assembly points [m] (assumed)
  spring_ref 0.15  % reference length of spring [m]
  C1 40000.0   % spring constant for front wheel suspension [N/m]
  C2 40000.0   % spring constant for rear wheel suspension [N/m]
  D1 1500.0    % damping coeff. for front wheel suspension [Ns/m]
  D2 1200.0    % damping coeff. for rear wheel suspension [Ns/m]
 TYRE 0.0016   % lag parameter for tyre model [s]
 CONTACT
  A1 5.0       % dimensions of a vehicle: length [m]
  A2 3.0       % width [m]
  A3 2.9       % height [m]
 EQUILIBRIUM
  R3  5.039617e-02  % vertical position of vehicle's center of mass [m]
  D11  9.999151e-01   D12  0.0   D13  -1.382584e-02 % director 1 [.]
  D21  0.0            D22  1.0   D23  0.0           % director 2 [.]
  D31  1.382916e-02   D32  0.0   D33  9.999048e-01  % director 3 [.]
```

```
% description of model 1 ends here
```

A.2 *platoon1.dat*

```
%
% Initialization of a two vehicle platoon with collision
%
NUMBER_OF_VEHICLES 2

% Vehicle 1
VEHICLE_HAS_MODEL 1 INITIALLY_WITH
  X 0.0               % X coordinate of center of mass [m]
  Y 0.0               % Y coordinate of center of mass [m]
  ORIENTATION 0.0     % heading angle [.] (cw:"-", ccw:"+")
  SPEED 29.1          % forward speed [m/s]
  STEERING 0.0        % steer angle [rad]

% Vehicle 2
VEHICLE_HAS_MODEL 1 INITIALLY_WITH
  X 7.0               % X coordinate of center of mass [m]
  Y -2.0               % Y coordinate of center of mass [m]
  ORIENTATION 0.0     % heading angle [.] (cw:"-", ccw:"+")
  SPEED 26.8          % forward speed [m/s]
  STEERING 0.0        % steer angle [rad]
```

A.3 *platoon2.dat*

```
%
% Initialization of a two vehicle platoon without collision
%
NUMBER_OF_VEHICLES 2

% Vehicle 1
VEHICLE_HAS_MODEL 1 INITIALLY_WITH
  X 0.0               % X coordinate of center of mass [m]
  Y 0.0               % Y coordinate of center of mass [m]
  ORIENTATION 0.0     % heading angle [.] (cw:"-", ccw:"+")
```

```
  SPEED 29.1          % forward speed [m/s]
  STEERING 0.0        % steer angle [rad]

% Vehicle 2
VEHICLE_HAS_MODEL 1 INITIALLY_WITH
  X 6.0               % X coordinate of center of mass [m]
  Y 3.6               % Y coordinate of center of mass [m]
  ORIENTATION 0.0     % heading angle [.] (cw:"-", ccw:"+")
  SPEED 26.8          % forward speed [m/s]
  STEERING 0.0        % steer angle [rad]
```

B Modified Source Code for the New Contact Detection Algorithm

This appendix includes three functions: `info()`, `eig()` and `norm()`. The purpose of adding the functions `info()` and `eig()` is discussed in Section 3. The function `eig()` consists of two subfunctions: `jacobi()` and `eigsrt()` which are adapted from *Numerical Recipes in C* [3]. The first function `jacobi()` calculates the eigenvalues and eigenvectors of a matrix. The function `eigsrt()` sorts the eigenvalues in descending order and rearranges the eigenvectors accordingly.

There are three small functions: `snorm()`, `cnorm()` and `choice()` in the function `norm()`. It determines the contact normal vector $\mathbf{n}_1$ which is calculated from the function `snorm()` if the intersection point defined in Section 2 is located in the spherical region of a vehicle. If the calculated intersection point is in the cylindrical region of a vehicle then the corresponding contact normal vector is computed by the function `cnorm()`. As for the function `choice()`, it determines which one of the principal directions is the contact normal vector if the intersection point is detected on a plane.

```
/*****************************************************************************/
/*   Compute the matrix KHAT=F^(-T)KF^(-1)                                   */
/*****************************************************************************/
void info(Vector SA, Matrix F, Matrix KHAT)
{
  Matrix FINV, FT, K, C;
  FINV=matrix(3,3); FT=matrix(3,3); K=matrix(3,3); C=matrix(3,3);

  matrix_inverse(F, 3, FINV);
  matrix_trans(FINV,FT,3,3);

  K[1][1]=1/(SA[1]*SA[1]); K[1][2]=0.; K[1][3]=0.;
  K[2][1]=0.; K[2][2]=1/(SA[2]*SA[2]); K[2][3]=0.;
  K[3][1]=0.; K[3][2]=0.; K[3][3]=1/(SA[3]*SA[3]);

  matrix_times_matrix(C,FT,K,3,3,3);
  matrix_times_matrix(KHAT,C,FINV,3,3,3);  /* KHAT=F^(-T)KF^(-1) */

  free_matrix(C);
  free_matrix(FT);
  free_matrix(K);
  free_matrix(FINV);
}

/*****************************************************************************/
/*  Compute and sort the eigenvalues and eigenvectors                        */
```

```
/*****************************************************************************/
void eig(Matrix a, int n, Vector d, Matrix v)
{
  int nrot;

  void jacobi(Matrix a, int n, Vector d, Matrix v, int *nrot);
  void eigsrt(Vector d, Matrix v, int n);

  jacobi(a, n, d, v, &nrot);
  eigsrt(d, v, n);
}

/*****************************************************************************/
/*  Compute the principal directions and principal values of the ellipsoids  */
/*  in the current configuration                                             */
/*****************************************************************************/
#define ROTATE(a,i,j,k,l) g=a[i][j]; h=a[k][l]; a[i][j]=g-s*(h+g*tau); a[k][l]=h+s*(g-h*tau);

void jacobi(Matrix a, int n, Vector d, Matrix v, int *nrot)
/* Compute all eigenvalues and eigenvectors of a real symmetric matrix a[1..n][1..n].
   On output, the elements of above the diagonal are destroyed. d[1..n] returns the
   eigenvalues of a. v[1..n][1..n] is a matrix whose columns contain, on output,
   the normalized eigenvectors of a. nrot returns the number jacobi rotates that
   were required. See Numerical Recipes on C pp.467-468.*/
{
  int j, iq, ip, i;
  double tresh, theta, tau, t, sm, s, h, g, c;
  Vector b, z;

  b=vector(n); z=vector(n);

  for(ip=1;ip<=n;ip++){
    for (iq=1;iq<=n;iq++) v[ip][iq]=0.0;
    v[ip][ip]=1.0;
  }
  for (ip=1;ip<=n;ip++){
    b[ip]=d[ip]=a[ip][ip];
    z[ip]=0.0;
  }
  *nrot=0;
  for (i=1;i<=50;i++){
    sm=0.0;
    for (ip=1;ip<=n-1;ip++){
      for (iq=ip+1;iq<=n;iq++)
sm += fabs(a[ip][iq]);
    }
    if(sm==0.0){
      free_vector(z);
      free_vector(b);
```

```
      return;
    }
    if (i<4)
      tresh=0.2*sm/(n*n);
    else tresh=0.0;
    for (ip=1;ip<=n-1;ip++){
      for (iq=ip+1;iq<=n;iq++){
g=100.*fabs(a[ip][iq]);
if (i>4 && (double)(fabs(d[ip])+g)==(double)fabs(d[ip])
    && (double)(fabs(d[iq])+g)==(double)fabs(d[ip]))
  a[ip][iq]=0.0;
else if (fabs(a[ip][iq])>tresh){
  h=d[iq]-d[ip];
  if ((double)(fabs(h)+g)==(double)fabs(h))
    t=(a[ip][iq])/h;
  else{
    theta=0.5*h/(a[ip][iq]);
    t=1.0/(fabs(theta)+sqrt(1.0+theta*theta));
    if (theta<0.0) t= -t;
  }
  c=1.0/sqrt(1+t*t);
  s=t*c;
  tau=s/(1.0+c);
  h=t*a[ip][iq];
  z[ip] -= h;
  z[iq] += h;
  d[ip] -= h;
  d[iq] += h;
  a[ip][iq]=0.0;

  for (j=1;j<=ip-1;j++) {
    ROTATE(a,j,ip,j,iq)
  }
  for (j=ip+1;j<=iq-1;j++) {
    ROTATE(a,ip,j,j,iq)
  }
  for (j=iq+1;j<=n;j++) {
    ROTATE(a,ip,j,iq,j)
  }
  for (j=1;j<=n;j++) {
    ROTATE(v,j,ip,j,iq)
  }
  ++(*nrot);
}
      }
    }
    for (ip=1;ip<=n;ip++){
      b[ip] += z[ip];
      d[ip]=b[ip];
```

```
      z[ip]=0.0;
    }
  }
  nrerror("Too many iterations in routine jacobi");
}

/*************************************************************************/
/*  Sorting the eigenvalues and eigenvectors                             */
/*************************************************************************/
void eigsrt(Vector d, Matrix v, int n)
/* Given the eigenvalues d[1..n] and the eigenvector in matrix v[1..n][1..n]
   as the output from Jacobi, this routine sorts the eigenvalues into decending
   order, adn rearranges the eigenvectors correspondingly. See Numerical Recipes
   in C on p.468. */
{
  int k, j, i;
  double p;

  for (i=1;i<n;i++){
    p=d[k=i];
    for (j=i+1;j<=n;j++)
      if (d[j] >= p) p=d[k=j];
    if (k != i){
      d[k]=d[i];
      d[i]=p;
      for (j=1;j<=n;j++){
p=v[j][i];
v[j][i]=v[j][k];
v[j][k]=p;
      }
    }
  }
}

/*************************************************************************/
/*  Compute the outward unit normal on the surface of the ellipsoids     */
/*************************************************************************/
void enorm(Vector x, Vector nvec1, Vector nvec2)
     /* x[1]=u1, x[2]=v1, x[3]=u2, x[4]=v2 */
{
  int i,k1,k2;
  double tang1[4], tang2[4], cp1[4], tang3[4], tang4[4], cp2[4];
  double norm1, norm2;
  double s_x1=sin(x[1]), s_x2=sin(x[2]), c_x1=cos(x[1]), c_x2=cos(x[2]);
  double s_x12=s_x1*s_x2, c_x12=c_x1*c_x2, cs_x21=c_x2*s_x1, cs_x12=c_x1*s_x2;
  double s_x3=sin(x[3]), s_x4=sin(x[4]), c_x3=cos(x[3]), c_x4=cos(x[4]);
  double s_x34=s_x3*s_x4, c_x34=c_x3*c_x4, cs_x43=c_x4*s_x3, cs_x34=c_x3*s_x4;

  k1=(int)(x[1]/(2*Pi));
```

```
  x[1]=x[1]-2*k1*Pi;

  k2=(int)(x[3]/(2*Pi));
  x[3]=x[3]-2*k2*Pi;

  for (i=1;i<=3;i++) {
    tang1[i]= -A1*cs_x21*F1[i][1]-A2*s_x12*F1[i][2]+A3*c_x1*F1[i][3];
    tang2[i]= -A1*cs_x12*F1[i][1]+A2*c_x12*F1[i][2];
    tang3[i]= -B1*cs_x43*F2[i][1]-B2*s_x34*F2[i][2]+B3*c_x3*F2[i][3];
    tang4[i]= -B1*cs_x34*F2[i][1]+B2*c_x34*F2[i][2];
  }
  /* Cross product of the two tangent vectors: */
  cp1[1]=tang2[2]*tang1[3]-tang1[2]*tang2[3];
  cp1[2]=tang2[3]*tang1[1]-tang2[1]*tang1[3];
  cp1[3]=tang2[1]*tang1[2]-tang2[2]*tang1[1];
  cp2[1]=tang4[2]*tang3[3]-tang3[2]*tang4[3];
  cp2[2]=tang4[3]*tang3[1]-tang4[1]*tang3[3];
  cp2[3]=tang4[1]*tang3[2]-tang4[2]*tang3[1];

  /* Compute unit normal vectors */
  norm1=sqrt(square(cp1[1])+square(cp1[2])+square(cp1[3]));
  norm2=sqrt(square(cp2[1])+square(cp2[2])+square(cp2[3]));
  if (fabs(x[1])<=Pi/2 || fabs(x[1])>=3*Pi/2)
    for (i=1;i<=3;i++) nvec1[i]=cp1[i]/norm1;
  else
    for (i=1;i<=3;i++) nvec1[i]= -cp1[i]/norm1;

  if (fabs(x[3])<=Pi/2 || fabs(x[3])>=3*Pi/2)
    for (i=1;i<=3;i++) nvec2[i]= cp2[i]/norm2;
  else
    for (i=1;i<=3;i++) nvec2[i]= -cp2[i]/norm2;
}

/*************************************************************************/
/*  Compute the direction of the contact force which is defined on the surface*/
/*  of a three-dimensional box with round corners                        */
/*************************************************************************/
/* Radius of the cylinders or spheres at the corners with respect to vehicle 1
   and 2 */
#define RAD1 0.3
#define RAD2 0.3

void norm(Vector x, Vector nvec1)
{
  int i;
  int corner1[4],corner[3];
  double e[4],pa1[4],pa2[4],pa3[4],pb1[4],pb2[4],pb3[4],nvec2[4];
  double alph[4],beta[4],drv[4],dev[4],ntri1[4],ntri2[4];
  double psi1,psi2,factor1,factor2,r1[4],r2[4],len1[4],len2[4];
```

```
double gama1[3],gama2[3],eta1[3],eta2[3],tau1[3],tau2[3];

double dot(Vector en1, Vector en2, int);
void pos(Vector state, Vector r1, Vector r2);
void snorm(Vector,Vector,Vector,Vector,Vector,double,Vector);
void cnorm(double,double,double,double,Vector,Vector,double,Vector);
void choice(Vector,Vector,Vector,Vector,Vector,Vector,Vector,Vector,Vector);

pos(x,r1,r2);
for (i=1;i<=3;i++) {
  /* the position vetors of the contact points on the ellipsoids 1 and 2
     relative to the mass centers are r1[] and e[], respectively */
  e[i]=r2[i]-xbar[i];

  /* The eigenvalues of KHAT are 1/(semi_axes)^2, therefore the acending sorting
     of eigenvalues in eigsrt() is actually decending sorting of the lengths of
     semi-axes. The vectors pa1,2,3 and pb1,2,3 are the principal directions of
     the ellipsoid 1 and 2, respectively, in the current configuration. */
  pa1[i]=v1[i][3]; pa2[i]=v1[i][2]; pa3[i]=v1[i][1];
  pb1[i]=v2[i][3]; pb2[i]=v2[i][2]; pb3[i]=v2[i][1];
}
drv[1]=dot(r1,pa1,3); drv[2]=dot(r1,pa2,3); drv[3]=dot(r1,pa3,3);
dev[1]=dot(e,pb1,3); dev[2]=dot(e,pb2,3); dev[3]=dot(e,pb3,3);

/* the angles between r1[] (e[]) and the principal directions */
alph[1]=atan(fabs(drv[2]/drv[1]));
alph[2]=atan(fabs(drv[3]/drv[1]));
alph[3]=atan(fabs(drv[3]/drv[2]));
beta[1]=atan(fabs(dev[2]/dev[1]));
beta[2]=atan(fabs(dev[3]/dev[1]));
beta[3]=atan(fabs(dev[3]/dev[2]));

/* Update the dimensions of the vehicles, len1[] & len2[] in the current
   configuration by the identity of the volumes of the ellipsoids and vehicles. */
factor1=pow(4*Pi/(3*sqrt(d1[1]*d1[2]*d1[3])*AX1*AX2*AX3),1./3);
factor2=pow(4*Pi/(3*sqrt(d2[1]*d2[2]*d2[3])*BX1*BX2*BX3),1./3);
len1[1]=factor1*AX1; len1[2]=factor1*AX2; len1[3]=factor1*AX3;
len2[1]=factor2*BX1; len2[2]=factor2*BX2; len2[3]=factor2*BX3;

if (2*RAD1 > len1[3] || 2*RAD2 > len2[3]) {
  nrerror("The radius of cylinder at the corner is too large!");
}

/* Define the angles of the ranges of corners located on the vehicles*/
gama1[1]=atan((len1[2]-2*RAD1)/len1[1]); gama1[2]=atan(len1[2]/(len1[1]-2*RAD1));
gama2[1]=atan((len2[2]-2*RAD2)/len2[1]); gama2[2]=atan(len2[2]/(len2[1]-2*RAD2));
eta1[1]=atan((len1[3]-2*RAD1)/len1[1]);  eta1[2]=atan(len1[3]/(len1[1]-2*RAD1));
eta2[1]=atan((len2[3]-2*RAD2)/len2[1]);  eta2[2]=atan(len2[3]/(len2[1]-2*RAD2));
tau1[1]=atan((len1[3]-2*RAD1)/len1[2]);  tau1[2]=atan(len1[3]/(len1[2]-2*RAD1));
```

```
tau2[1]=atan((len2[3]-2*RAD2)/len2[2]);  tau2[2]=atan(len2[3]/(len2[2]-2*RAD2));

/* The contact normal is defined on the vehicle 1. */
if (alph[1] >= gama1[1] && alph[1]<= gama1[2]) corner1[1]=TRUE;
else corner1[1]=FALSE;
if (alph[2] >= eta1[1] && alph[2] <= eta1[2]) corner1[2]=TRUE;
else corner1[2]=FALSE;
if (alph[3] >= tau1[1] && alph[3] <= tau1[2]) corner1[3]=TRUE;
else corner1[3]=FALSE;

if (corner1[1] || corner1[2] || corner1[3]) corner[1]=TRUE;
else corner[1]=FALSE;

if (beta[1] >= gama2[1] && beta[1] <= gama2[2]) corner[2]=TRUE;
else if (beta[2] >= eta2[1] && beta[2] <= eta2[2]) corner[2]=TRUE;
else if (beta[3] >= tau2[1] && beta[3] <= tau2[2]) corner[2]=TRUE;
else corner[2]=FALSE;

if (corner[1] && corner[2]){
  /* corner to corner contact */
  if (corner1[1] && corner1[2]) {
    if (!corner1[3]) nrerror("Errors occur in corner dectection.");
    snorm(len1,r1,pa1,pa2,pa3,RAD1,nvec1);
  }
  else if (corner1[1] && corner1[3]){
    if (!corner1[2]) nrerror("Errors occur in corner dectection.");
    snorm(len1,r1,pa1,pa2,pa3,RAD1,nvec1);
  }
  else if (corner1[2] && corner1[3]){
    if (!corner1[1]) nrerror("Errors occur in corner dectection.");
    snorm(len1,r1,pa1,pa2,pa3,RAD1,nvec1);
  }
  else if (corner1[1]) {
    cnorm(len1[1],len1[2],r1[1],r1[2],pa1,pa2,RAD1,nvec1);
  }
  else if (corner1[2]) {
    cnorm(len1[1],len1[3],r1[1],r1[3],pa1,pa3,RAD1,nvec1);
  }
  else {
    cnorm(len1[2],len1[3],r1[2],r1[3],pa2,pa3,RAD1,nvec1);
  }
}
else if (corner[1] || corner[2]){
  /* corner to face contact */
  if (corner[1]){
    choice(beta,gama2,eta2,tau2,dev,pb1,pb2,pb3,nvec2);
    /* using the principal direction of ellipsoid 2 as the contact normal */
    for (i=1;i<=3;i++) nvec1[i]= -nvec2[i];
  }
```

```
    else {
      choice(alph,gama1,eta1,tau1,drv,pa1,pa2,pa3,nvec1);
      /* using the principal direction of ellipsoid 1 as the contact normal */
    }
  }
  else {
    /* face to face contact */
    choice(alph,gama1,eta1,tau1,drv,pa1,pa2,pa3,ntri1);
    choice(beta,gama2,eta2,tau2,dev,pb1,pb2,pb3,ntri2);

    /* compute the angles psi1,2 which are the angles between the position
       vector of contact point and the plane normal(ntri1,2), the principal
       direction, chosen in the function choice */
    if (fabs(dot(ntri1,pa1,3)) > 0.9) {
      psi1=alph[1]; /* the contact point is at the plane with normal pa1 */
    }
    else if (fabs(dot(ntri1,pa2,3)) > 0.9) {
      psi1= Pi/2-alph[1]; /* the contact point is at the  with normal pa2 */
    }
    else psi1= Pi/2-alph[2]; /* the contact point is at the plane with normal pa3 */

    if (fabs(dot(ntri2,pb1,3)) > 0.9) psi2=beta[1];
    else if (fabs(dot(ntri2,pb2,3)) > 0.9) psi2= Pi/2-beta[1];
    else psi2= Pi/2-beta[2];

    if (psi1 < psi2) {
      /* if the angle psi1 is less than psi2 then the plane normal on the
 vehicle 1 is used as the contact normal*/
      for (i=1;i<=3;i++) nvec1[i]= ntri1[i];
    }
    else {
      for (i=1;i<=3;i++) nvec1[i]= -ntri2[i];
    }
  }
}

/*************************************************************************/
/*  Compute the outward unit normal on the surface of spheres            */
/*************************************************************************/
void snorm(Vector L, Vector a, Vector p1, Vector p2, Vector p3, double R,
   Vector out)
/* Vector L[1..3] and a[1..3] are the dimensions of the vehicles and positions
   of contact points on the ellipsoid 1 w.r.t. the mass center in the current
   configuration. The principal directions of the ellipsoid 1 or 2 are p1,2,3.
   The variable R is  the radius of spheres at the corners of a vehicle and
   the output unit normal vector on the vehicle 1 is the Vector out. */
{
  int i;
  double g,h,q,temp,p,x[4],y[4],z[4],norm;
```

```
  g= a[1]*a[1]+a[2]*a[2]+a[3]*a[3];
  h= 2*R*(fabs(a[1])+fabs(a[2])+fabs(a[3]))-(fabs(a[1])*L[1]+fabs(a[2])*L[2]+fabs(a[3])*L[3]);
  q= 2*R*R+(L[1]*L[1]+L[2]*L[2]+L[3]*L[3])/4-(L[1]+L[2]+L[3])*R;
  temp= h*h-4*g*q;

  if (temp < 0) nrerror("Negative values in square root in function snorm.");
  p= (-h+sqrt(temp))/(2*g);

  /* x[1..3] is the position of point projected by the line from the mass center
     to the contact point on the surface of the sphere */
  for (i=1;i<=3;i++) x[i]= p*a[i];

  /* the distance measured from the mass center to the intersection point on the
     sphere must be larger than the distance from the mass center to the contact
     point */
  if (p < 1) nrerror("Errors occur in function snorm.");

  for (i=1;i<=3;i++) {
    y[i]= a[i]/fabs(a[i])*(L[i]/2-R); /* the position of the center of the sphere */
    z[i]= x[i]-y[i];
  }
  out[1]= z[1]*p1[1]+z[2]*p2[1]+z[3]*p3[1];
  out[2]= z[1]*p1[2]+z[2]*p2[2]+z[3]*p3[2];
  out[3]= z[1]*p1[3]+z[2]*p2[3]+z[3]*p3[3];

  norm= sqrt(out[1]*out[1]+out[2]*out[2]+out[3]*out[3]);
  for (i=1;i<=3;i++) out[i] /= norm;
}

/*****************************************************************************/
/*  Compute the outward unit normal on the surface of cylinders              */
/*****************************************************************************/
void cnorm(double L1, double L2, double ax, double bx, Vector p1, Vector p2,
   double R, Vector out)
{
  int i;
  double c,g,h,q,temp,a[3],L[3],x[3],y[3],z[3],norm;

  a[1]=ax; a[2]=bx;
  L[1]=L1; L[2]=L2;
  c= tan(a[2]/a[1]);

  /* x[1..2] is the position of point projected by the line from the mass
     center to the contact point on the surface of the cylinder */
  g= 1+c*c;
  h= (2*R-L[1])*a[1]/fabs(a[1])+(2*R-L[2])*c*a[2]/fabs(a[2]);
  q= R*R+(L[1]*L[1]+L[2]*L[2])/4-(L[1]+L[2])*R;
  temp= h*h-4*g*q;
```

```
  if (temp < 0) nrerror("Negative values in square rootin function cnorm.");

  x[1]= (-h+sqrt(temp))/(2*g);
  x[2]= c*x[1];
  for (i=1;i<=2;i++) {
    y[i]= a[i]/fabs(a[i])*(L[i]/2-R); /* the position of the center of the circle */
    z[i]= x[i]-y[i];
  }
  out[1]= z[1]*p1[1]+z[2]*p2[1];
  out[2]= z[1]*p1[2]+z[2]*p2[2];
  out[3]= z[1]*p1[3]+z[2]*p2[3];

  norm= sqrt(out[1]*out[1]+out[2]*out[2]+out[3]*out[3]);
  for (i=1;i<=3;i++) out[i] /= norm;
}

/*************************************************************************/
/*  Find the corresponding plane normal of the 3d box as the contact normal   */
/*************************************************************************/
void choice(Vector a, Vector g, Vector e, Vector t, Vector d, Vector p1,
    Vector p2, Vector p3, Vector out)
{
  int i;

  if (d[1]>0 && a[1]<g[1] && a[2]<e[1]) {
    for (i=1;i<=3;i++) {
      out[i]=p1[i];
    }
  }
  else if (d[1]<0 && a[1]<g[1] && a[2]<e[1]) {
    for (i=1;i<=3;i++) {
      out[i]= -p1[i];
    }
}
  else if (d[2]>0 && a[1]>g[2] && a[3]<t[1]) {
    for (i=1;i<=3;i++) {
      out[i]= p2[i];
    }
  }
  else if (d[2]<0 && a[1]>g[2] && a[3]<t[1]) {
    for (i=1;i<=3;i++) {
      out[i]= -p2[i];
    }
  }
  else if (d[3]>0 && a[3]>t[2] && a[2]>e[2]) {
    for (i=1;i<=3;i++) {
      out[i]= p3[i];
    }
  }
```

```
  else {
    for (i=1;i<=3;i++) {
      out[i]= -p3[i];
    }
  }
}
```