VEHICLE DRIFT SIMULATOR

By Bright Soo

Senior Project

The Mathematics Department

California Polytechnic State University

December, 1997

TABLE OF CONTENTS

INTRODUCTION & APPLICATIONS **3**

ASSUMPTIONS AND JUSTIFICATIONS **6**

PRE-DEFINED ASSUMPTIONS 6
ASSUMPTIONS 6

BACKGROUND IN VEHICLE DYNAMICS **9**

FORCE-COUPLE SYSTEM 9
CONFIGURATION OF VEHICLE 9
TIRE FRICTIONAL FORCES 10
AN EXAMPLE 11

APPROACH TO CONSTRUCTING THE PROGRAM **13**

THE PROGRAM **15**

WRITING 15
KNOWN BUGS (READ ME) 16

ACKNOWLEDGEMENTS **18**

BIBLIOGRAPHY **19**

APPENDIX (SOURCE CODES) **20**

MODULE 20
FORMSPLASH 20
FORMGRAPH 21
FORMOPTION 37

ENDNOTES **39**

INTRODUCTION & APPLICATIONS

The purpose of the Vehicle Drift Simulator program is to allow repeatable scientific experimentation on the dynamics involved in a skidding vehicle with:

1) rear wheels locked
2) no braking applied to the front wheels, and
3) front wheels' steering angles locked.

Study of this subject has broad applications. For example, the movie industry frequently relies on such vehicle configurations to produce spectacular stunt driving: over-steering would occur easily if the stunt driver pulls up the handbrake in the middle of a turn (figure 1). Of more significance, study of such dynamics have already in the past enabled engineers to produce better automobiles such as mid-engine cars, which have better handling characteristics especially when skidding.

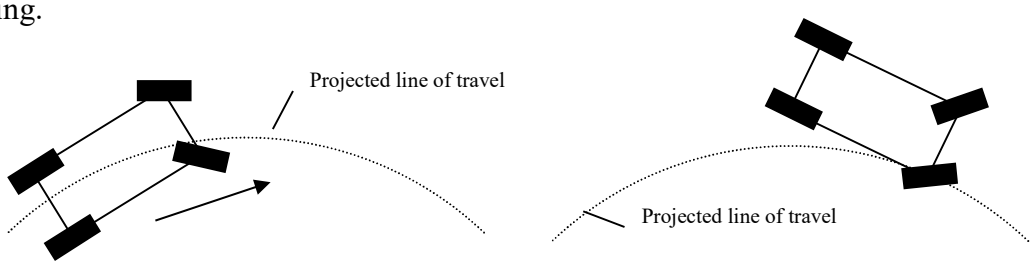


Figure 1. Over-steering.

While scientific predictions can easily be made on a vehicle in mostly any travel configuration at an instantaneous point in time, there is to my knowledge no publicly available programs that provides a top-view of the loci of a vehicle, and its wheels, skidding in the concerned fashion during a continuous frame of time.

Similar simulation algorithms of this sort no doubt exist in the many racing-games that are available. However, to my observation, they all lack scientific rigor[1], including but not limited to:

- Lack of accuracy. Many racing games incorporate artificial parameters to make the game more pleasurable and easy. For example, the vehicle travels faster if it is behind other players' cars. Also, many games have their own set of vehicle-skidding characteristics: some games over-steer easily, very few allows you to correct it by increased throttle, which in real life is a practical correctional maneuver.
- No handbrake control, with very few exceptions.
- Limited, if any, choice of vehicle configuration.

The Vehicle Drift Simulator overcomes all these limitations and more. All parameters are user-defined numerically, rather than graphically, giving very precise control. User can send the vehicle into a spin at any speed[2], manually set the rate of turn (angular velocity), and even adjust the gravity. Its calculations are scenario-independent: variables calculated for a car spinning along a curve are calculated the same way for a car sliding along a straight line, and the results are fed back into the motion equations the same way. To close this introduction, following is a list of all variables and parameters non-trivially fed into the program algorithms:

- Wheelbase length
- Wheelbase width
- Vehicle mass (weight)
- Weight distribution (front/rear) (not used for moment of inertia)
- Moment of inertia
- Tire coefficient of kinetic friction
- Speed and traveling direction of vehicle (center of mass)
- Angular velocity
- Linear and angular accelerations

- Frictional forces exerted on tires and the resultant force and torque exerted on center of mass
- Time
- Steer angle of front wheels
- Position of vehicle relative to its path of travel

ASSUMPTIONS AND JUSTIFICATIONS

Pre-Defined Assumptions

- *Rear wheels locked*. Handbrakes in real life apply almost invariably to rear wheels only. Locking the rear wheels is to simulate what happens once the handbrake is abruptly pulled up.
- *No braking applied to front wheels.* If the front wheels were also locked, the resulting frictional forces would act on the vehicle much the same way as those arising from the rear wheels do, which is not at all an interesting case.
- *Front wheels' steering angles locked.* If not, they may change or self-align during the course of skidding, and introduce too many variables. This change in steering angle depends primarily on what is called the pneumatic trail (Wong 29), and also non-trivially on the such parameters as tire pressure, camber angle (angle of wheel plane from vertical), inclination of turning axis (as the inclined fork of a bicycle), kingpin offset, resistance arising from steering linkages, etc.

Assumptions

- *Constant coefficient of kinetic friction.* In reality, the coefficient may drop slightly at increased speeds. However, this drop is usually small, and so is the rate of this drop. In addition, this drop is very difficult to quantify because it depends on such parameters as the temperatures of the contact areas, adhesion, etc. (Meriam and Kraige "Statics" 350) This assumption is a common practice in the engineering discipline in the solutions of a wide range of problems.
- *All four wheels skidding during motion.* Thus, only kinetic (sliding) friction will be considered. This assumption is considerably consistent with reality in that, by the time any

one tire regains traction, the vehicle would have come to a complete or almost complete stop, as evident from the fact that the resulting skid marks usually end where the tires are in their stopped position.

- *No weight shift.* In other words, only a two-dimensional model will be developed. Although weight shift is an important factor in no-slip braking (Senna 58), its contribution to the forces and the deceleration experienced by a skidding vehicle are comparatively small. It can also be observed that vehicles having their centers of mass at different heights with otherwise the same configurations behave very similarly. In any case, the forces exerted on the tires of a spinning vehicle usually dwarfs the effects of weight shift.
- *Front wheels experiencing lateral frictional forces only.* Because the front wheels are free to rotate on their lateral (running) axes, it is reasonable to assume that the longitudinal frictional forces exerted on these tires are negligible (figure 2). Dictated by the formula for kinetic friction, $F = \mu N$, where F is the frictional force, μ the coefficient of kinetic friction, and N the normal force (weight supported), the arising frictional force must still have its full magnitude (see assumption on μ above). Therefore, as long as the front wheel has any speed in the lateral direction, there will be a frictional force in the opposite direction with the full magnitude. Intuitively, this explains why vehicles go into spins so abruptly when the hand brakes are applied.

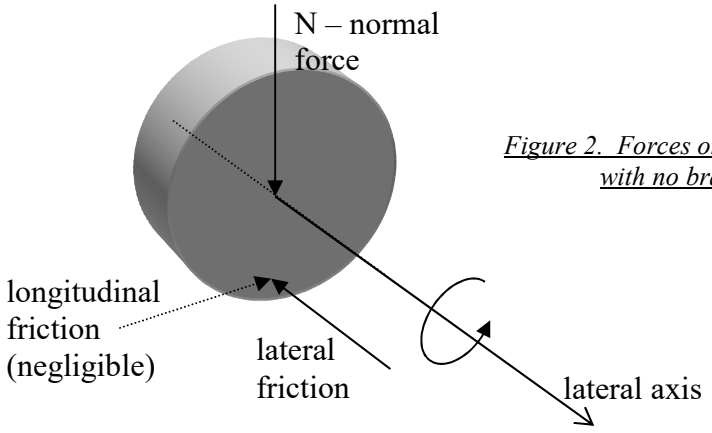


Figure 2. Forces on a front wheel with no braking.

- *Magnitude of friction arising from rear wheels independent of sliding direction.* In effect, we are assuming the rear tires are spherical at the ground contact patch. This is a reasonable assumption because, again, the kinetic friction formula asserts that the friction is independent of contact area.
- *Frictional forces arising only from road-tire interactions.* Thus, we ignore wind resistance, rolling resistance and the like. Although these kinds of resistance may be prominent at certain speeds, their effects are minimal in a skidding vehicle.

BACKGROUND IN VEHICLE DYNAMICS

Force-couple system

The most important concept employed in this project is the force-couple system. It asserts that when different forces act on a rigid body, their combined net effect is equivalent to that produced by a single resultant force and couple (torque) acting on any point of the body (Meriam and Kraige, "Statics" 47). Thus, for our purposes, the forces exerted on the four tires of the vehicle can be interpreted as a single force and couple acting on its center of mass.

The resultant force is merely the vector sum of all forces, and the resultant couple the vector sum of the moments about the center of mass produced by these forces. Under this system, then, linear and angular acceleration can be independently calculated with minimal efforts.

Configuration of vehicle

When making a turn, vehicle would travel along a circle with its center being collinear with the points of contact of the two rear tires. The distance of this center from the rear tires will, of course, depend on the curvature of the turn. In order to minimize slippage ("tire scrub"), the normals of the two front tires (i.e. lateral axes) will both point at this center of travel circle, as shown in figure 3, thus allowing them to be in pure rolling without lateral sliding(Wong 282-283). This steering geometry is called Ackerman steering geometry.

In the calculation of the initial conditions of the vehicle the moment before the rear wheels are locked, the linear and angular velocity of the center of mass will be calculated using a turning radius originating from this turning center that is common to all four wheels (and any point of the vehicle).

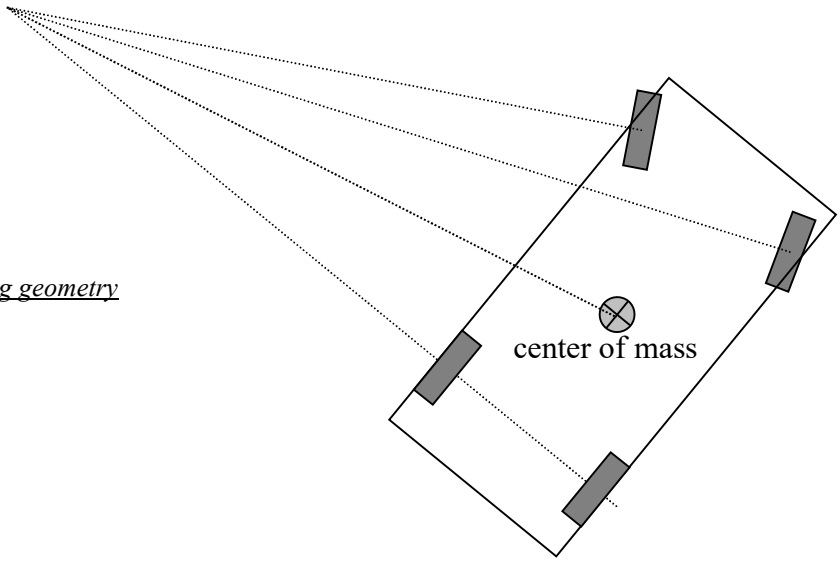


Figure 3. Ackerman steering geometry

Tire frictional forces

Calculation of friction arising from the sliding rear tires are straightforward. As noted in the *assumptions* section, it is merely the coefficient of kinetic friction multiplied by the normal force, or weight supported. This normal is in turn dependent on the location of center of mass, or *weight distribution*. For example, the rear tires of a vehicle with a 60% front / 40% rear weight distribution would each support a normal force of 20% of the vehicle weight. Note that this weight distribution corresponds to a center of mass positioned at 40% of the vehicle's length away from the *front*.

For the sliding front wheels, the *magnitude* of friction is identically calculated, with its direction perpendicular to the wheel plane and opposing the lateral sliding. This can be visualized by looking at the contact patch between the tire and the ground. As shown in (figure 4), when a front tire is both rolling forward and sliding laterally, the contact patch is sliding laterally while being rotated off this contact area. At the same time, new areas of the tire are being rotated into the contact patch. Recalling that we are ignoring rolling resistance and that no

brake is applied to the front wheels, we can see the net effect is that there is always an area of contact patch sliding laterally, but rolling off without resistance, thus generating a lateral kinetic friction equal to the locked, sliding rear tires.

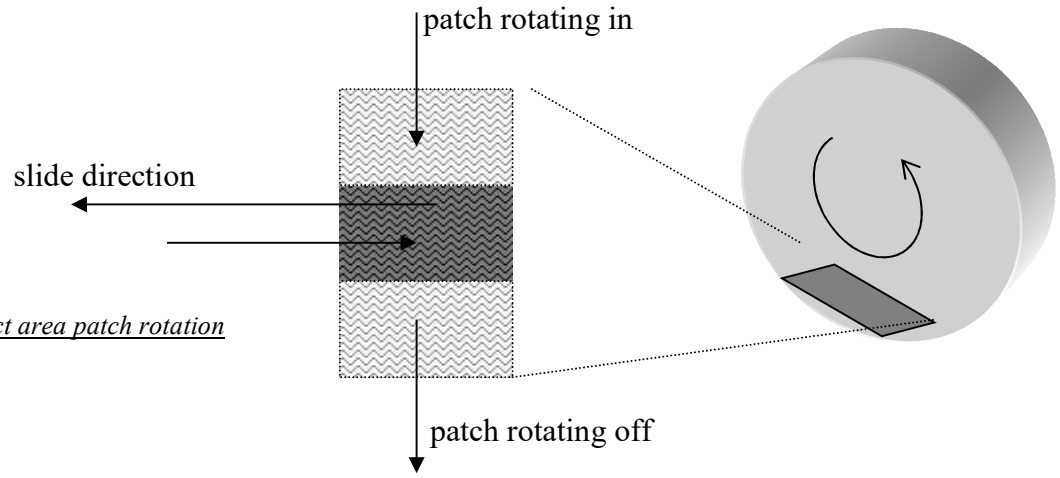


Figure 4. Contact area patch rotation

An example

Figure 5 shows an approximate diagram of a vehicle with front wheels at certain steering angles sliding to the right with a speed of **v**, while rotating counter-clockwise at an angular speed of **ω**. $\mathbf{v}_i$ is a vector representing the absolute velocity of the corresponding wheel, and is the vector sum of **v**, the vehicle velocity, and the velocity of the wheel with respect to the center of mass. Thus, because the vehicle is also rotating, the $\mathbf{v}_i$'s are not parallel.

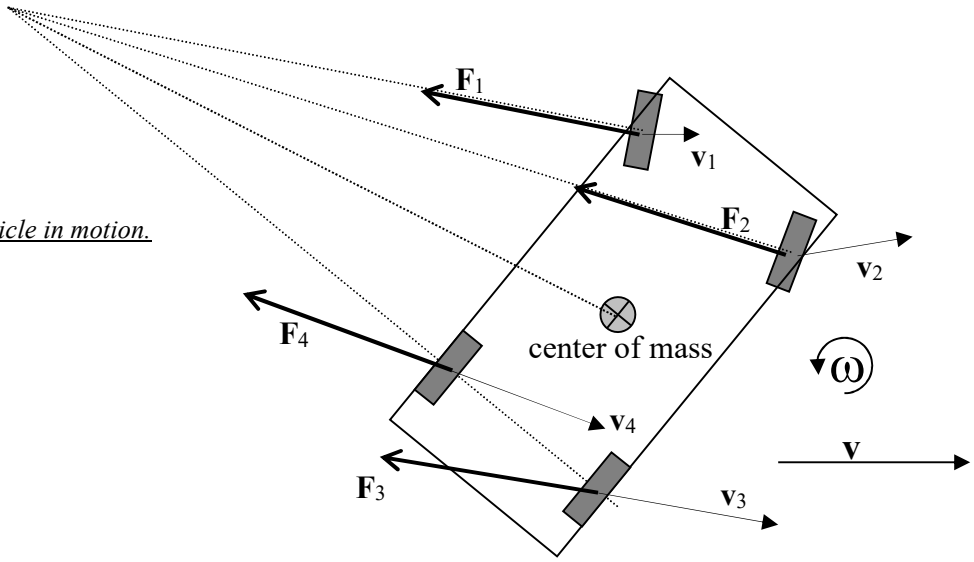


Figure 5. A sliding vehicle in motion.

Observe that the opposing frictional forces $\mathbf{F}_i$'s are of the same magnitude despite different wheel velocities. Furthermore, $\mathbf{F}_3$ and $\mathbf{F}_4$ are directly opposite the directions of $\mathbf{v}_3$ and $\mathbf{v}_4$, respectively, but the same is not true for $\mathbf{F}_1$ and $\mathbf{F}_2$, which are laterally perpendicular to the wheels. As far as linear acceleration is concerned, the resultant force is then the sum of all four force vectors placed at the center of mass. The resultant moment would be the moment sum of these forces about the center of mass.

In this diagram, it is apparent that the resulting linear acceleration will be to the left and slightly upward. The direction of angular acceleration, however, is not so clear in this diagram, although it seems to be clockwise at a small magnitude. The point is that with different vehicle configurations and different initial conditions, the resulting moment may either turn the vehicle into a spin, or it may slow down an already-spinning vehicle (as in this case).

APPROACH TO CONSTRUCTING THE PROGRAM

Microsoft's Visual Basic 5.0 Professional Edition is the programming software used to construct this program because of its powerful graphic capabilities, compatibility with large number of computers (Windows95), and ease of use (modularity).

The algorithm employs linear interpolation to approximate the path and position of the vehicle. Basically, in regular time intervals, it monitors the linear and angular velocities of the vehicle, then projects its position at the end of the time interval. Next it calculates forces acting on it at this moment, and finally calculates new acceleration parameters, to be used at the next cycle.

In deeper details, before the above cyclic algorithm is started, initial conditions, provided by the user, are input to calculate the initial position and velocities of the vehicle. The position is then plotted on screen, while the velocities are used to calculate the arising frictional forces, which are summarized into a force-couple system as described earlier. This force-couple equivalent enables the program to further calculate the linear and angular accelerations of the vehicle based on the values of vehicle mass and moment of inertia, respectively. Finally, constant-acceleration formulas are used to project the vehicle's position at a certain interval of

time t_{incr} later[3], and the new velocities and position at the end of this time interval are calculated and updated. The linear and angular constant-acceleration formulas used, respectively, in one-dimensional forms, are:

$$x_f = x_i + v_i t_{incr} + (\tfrac{1}{2}) a t_{incr}^2$$

$$\theta_f = \theta_i + \omega_i + (\tfrac{1}{2}) \alpha t_{incr}^2$$

The final velocities after t_{incr} are simply:

$$v_f = v_i + a t_{incr}$$

$$\omega_f = \omega_i + \alpha t_{incr}$$

At this point the program will repeat itself and go into a cycle.

THE PROGRAM

Writing

Not all parts of the program are written by me. In Visual Basic (VB), forms (windows) are visually designed via graphical user interface (GUI), so are all controls such as buttons, check boxes, scroll buttons, and so on. Most of the time, modifications to their appearances are also done through simple text entries via GUI. The entire program code, in text format, therefore contains statements defining the look of the program that are written by VB, while the rest are written by me.

The sections of codes that I wrote dictates how the code executes. They range from simple user-prompts to internal calculations, decision-making algorithms to the actually screen-drawing, and so on. In short, they are responsible for everything beyond the GUI scope of VB. Thus, VB and I comprise of the authorship of the program.

The VB programming elements can roughly be categorized into functions, sub-procedures, and modules. Functions are very similar to mathematical functions: they take in arguments and then return a value (numeric or textual). Sub-procedures are sections of codes that are executed when explicitly called by another procedure or event (such as mouse clicks) – these are the only ways they can be called. When a sub-procedure ends, the program goes into idle, and will not "flow" onto the next procedure. Finally, modules behave like a house for the sub-procedures and functions to reside in. In a module one can only declare global variables.

All the codes that have been written by me are in the appendix.

Known bugs (READ ME)

Unfortunately this program has not been designed to be fool-proof. The user must enter valid values in the text boxes for the program to run correctly, otherwise the program would crash, although such crashes does not seem to impact the Windows operating system. To assist the user in this matter, auto-popup ToolTips have been implemented for all input fields. These ToolTips will indicate valid ranges or limitations, if any, of the input fields.

During the testing of this program, it is found that the program is quite robust as long as the user follows all the instructions in the ToolTips. Thus far, the finished program has never crashed on any other causes.

One feature that have yet to be implemented is the ability to recognize when the vehicle has come to a stop. Currently, the program will continue indefinitely to make calculations on the vehicle even if all speeds have reduced to zero. Thus it is up to the user to stop the calculations, by pressing "Stop" or "Pause".

One negative side-effect of this lack of "auto-stop" capability is that, occasionally, after a vehicle has come to a near-complete stop, it may start to very slowly rotate about its center. This is due to the fact that the program uses time-interval linear interpolation techniques to approximate motion.

The forces that are bringing a spinning vehicle to a near-complete stop may, once interpolated beyond the moment of true stop, actually reverse the spinning direction. However minimal this reverse spinning rate is, the program recognizes the reversed spinning and therefore assert opposing forces which, again after interpolation, may reverse the spinning once again. And this cycle may repeat. Most of the time, the net effect is a stationary vehicle. Occasionally, however, this cycle of calculations may come to a steady state in which each successive cycle produces the same result, a result that spins the vehicle one way faster than the other. This is the causing of the mysterious spinning of the vehicle.

One attempted solution to this problem is the "Front Wheel Sensitivity" input field. It adjusts the sensitivity of the programs ability to recognize a slow spinning rate and apply frictional forces on the front wheels. Unfortunately, the scale of calculations that are causing the problem is apparently too small and random, so that this attempted solution has only limited success in this area.

A more practical use of this input field is to increase the stability or the self-aligning property of a vehicle. For example, if the simulation vehicle is set to skid directly straight ahead with no angular velocity, under zero front wheel sensitivity (most sensitive, zero allowance), the vehicle may go into a spin if the straight travel path is not vertical or horizontal, due to the intrinsic inaccuracies arising from the rounding-offs in internal calculations. In this case, increasing the input field value (increasing allowance) will delay application of lateral friction forces and thus produce a better simulation of a vehicle skidding on a straight path, where spinning would theoretically not occur.

ACKNOWLEDGEMENTS

I would like to thank Dr. Jim Mueller, my senior project advisor, for his patience, guidance, suggestions, kindness, and for being the math instructor in CalPoly that excels in his area the most. While I'd like to further elaborate, such an act would seem guilty of hypocrisy in trying to get a good evaluation.

I'd also like to thank my friend John Leung and my brother Armstrong Soo for beta-testing my program. Specifically, the zooming feature is his suggestion, and the scrolling feature a subsequent inspiration.

Next, thanks go to my family who have gone through the ordeal of not watching television most of the time during the last two weeks of this project so I can concentrate.

Last but not least, I'd like to thank my girlfriend Charlotty Wong for giving me so much spiritual support, as she did in times past, all the while when I could not spend time with her during these two weeks.

BIBLIOGRAPHY

Meriam, J. L. , and L. G. Kraige. Engineering Mechanics: STATICS 4th Ed. New York: John Wiley & Sons, 1997

Meriam, J. L. , and L. G. Kraige. Engineering Mechanics: DYNAMICS Vol. 2, 3rd Ed. New York: John Wiley & Sons, 1992

Senna, Ayrton. Principles of Race Driving Surrey, United Kingdom, 1993

Wong, J. Y.. Theory of Ground Vehicles 2nd Ed. New York: John Wiley & Sons, 1993

APPENDIX (SOURCE CODES)

Module

```
Option Explicit
DefDbl A-Z

'Vehicle Variables
    'Wheelbase Length & Width; Mass;
    'CenterRear=0.3 means 7/3 F/R; Moment of Inertia
Global carLength, carWidth, Mass, CenterRear, MmtInertia

'Driving Variables
Global HandBrake As Boolean
Global Vx0, Vy0, massRadius
Global LR As String    'inmost radius; Steer direction ("L" or "R")

'Vehicle Performance Variables
Global CoeffKin            'Coefficient of Kinetic Friction of one tire

'Calculation Variables
Global incT, SnapInterval, dotSize, Pi As Double
Global Sensitivity

'Graphing Variables
Global xBegin, xEnd, yBegin, yEnd, incX, incY, Res
Global x0, y0
    ' 0-CEnter, 1-FL, 2-FR, 3-RR, 4-RL
Global Connect(4) As Boolean, Color(4) As Long
Global Gyrate As Boolean
```

FormSplash

```
Option Explicit

Private Sub Form_KeyPress(KeyAscii As Integer)
    Unload Me
End Sub

Private Sub Form_Load()
    lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." &
App.Revision
    lblProductName.Caption = App.Title
End Sub

Private Sub Frame1_Click()
    Unload Me
End Sub
```

FormGraph

```
Dim loopflag As Boolean, StopNow As Boolean, _
    Pause As Boolean
Dim t As Double ', tEND As double
Dim massX As Double, massY As Double
Dim WhNormal(1 To 4, 1 To 2) As Double
Dim FLxy(1 To 4, 1 To 3) As Double

Dim Vx As Double, Vy As Double
Dim AccX As Double, AccY As Double
Dim theta As Double, omega As Double, alpha As Double

Dim F(1 To 4) As Double, g As Double    'Scalar
Dim totForce(1 To 2) As Double, totTorq As Double
Dim crnF(1 To 4, 1 To 3) As Double
Dim crnV(1 To 4, 1 To 3) As Double

Dim FL(0 To 4) As String        'as constants
Dim memMtx(1 To 3) As Double
'Dim In_xBegin As double, In_xEnd As double, _
'    In_yBegin As double, In_yEnd As double
Dim ChangeCar As Boolean, timeCount As Single, tick As Boolean
Private Sub Form_Load()
    Pi = 4 * Atn(1)
    g = 9.8     '9.8 m/s^2
    'Default McLaren Settings in SI Units
'    carLength = 4.29   'meters, 4.29x0.82
'    carWidth = 1.82     '1.82
'    Mass = 1139    '1139 kg
'    CenterRear = 0.412  '.412
    'MmtInertia = 2061   '2061 kg-m^2
'    CoeffKin = 10 / 11.16

    'Traveling Settings
    'Vx0 = 91.7 * 1000 / 3600     ' 25.47 m/sec
    HandBrake = True
    Vx0 = 0: Vy0 = 20      '6.7 m/s = 15 mph
    massRadius = 17: LR = "L" '6.34 massRadius
    incT = 0.01     'sec
    Sensitivity = 0.0001
    dotSize = 1

    'Default Screen Graph settings
    xBegin = -20: xEnd = 7.5
    yBegin = -15: yEnd = 35
    incX = 4: incY = 4
  '  Res = 0.06
    x0 = 0: y0 = 0
    Connect(0) = 0
    Connect(1) = 0: Connect(2) = 0
    Connect(3) = 0: Connect(4) = 0
    Color(0) = vbBlack
    Color(1) = vbRed: Color(2) = vbMagenta
```

```
    Color(3) = vbBlue: Color(4) = vbGreen

    'Default internal calculation settings
    FL(0) = "CE": FL(1) = "FL": FL(2) = "FR": _
    FL(3) = "RR": FL(4) = "RL"
    Gyrate = False
'    initToolTip

    carMcLaren
    theta = 0 ': torq = -11420
    omega = iniOmega
    calcMmtInertia
    InitBoxes (True)
End Sub
Private Sub cmdClear_Click()
    Cls
    DrawScale
End Sub
Private Sub cmdDraw_Click()
    cmdDraw.Caption = "Draw"
    cmdPause.Caption = "Pause"
    cmdDraw.Enabled = False
    Cls
    t = 0: 'tEND = 999
'    If CheckBoxes() = False Then Exit Sub
    If chkNormal = 1 Then cmdNormal_Click
    If chkInertia = 1 Then cmdInertia_click
    ReadBoxes
    massX = x0: massY = y0: Vx = Vx0: Vy = Vy0   ': t = 0
    AutoRedraw = True
    loopflag = False
    StopNow = False
    Pause = False

'    theta = 0 ': torq = -11420
'    omega = iniOmega

    DrawScale
    calcForce
    Main
End Sub
Private Sub cmdPause_Click()
    If Pause = False Then
        Pause = True
        StopNow = True
        cmdPause.Caption = "Resume"
        cmdDraw.Caption = "ReDraw"
        'cmdDraw.Enabled = True
    ElseIf Pause = True Then
        Pause = False
        StopNow = False
        cmdPause.Caption = "Pause"
        cmdDraw.Caption = "Draw"
        'cmdDraw.Enabled = False
'        AutoRedraw = True
        Main
    End If
```

```
    loopflag = True
End Sub
Private Sub cmdSnap_Click()
    loopflag = False
End Sub
Private Sub cmdStop_Click()
    StopNow = True
    cmdPause.Caption = "Resume"
    Pause = True
End Sub
Function Maximum(a, b)
    If Val(a) >= Val(b) Then
        Maximum = a
    Else
        Maximum = b
    End If
End Function
Public Sub BeforeMain()
    cmdDraw.Enabled = False
    cmdSnap.Enabled = True
    cmdPause.Enabled = True
    cmdStop.Enabled = True
    cmdClear.Enabled = False
    cmdSnap.SetFocus
End Sub
Public Sub Main()
    BeforeMain
    DrawWidth = dotSize

    PSet (massX, massY)
    If chkRealTime.Value = 1 Then
        Timer2.Interval = incT * 1000
        Timer2.Enabled = True
    End If
    Do Until t < 0
        If chkSnap.Value = 1 And t > SnapInterval Then
            cmdSnap_Click
            SnapInterval = t + SnapInterval
        End If
        cornerpos
        DrawLocus

'        PSet (t * 7, totForce(1) / 2000), vbRed
'        PSet (t * 7, totForce(2) / 2000), vbGreen
'        PSet (t * 7, Sqr(totForce(1) ^ 2 + totForce(2) ^ 2) / 1000)
'        AutoRedraw = False
'        CurrentX = 1: CurrentY = 5
'        Print t
'        DrawWidth = 5
'        Line (0, 0)-(totForce(1) / 1000, totForce(2) / 5000)
'        AutoRedraw = True
'        PSet (25, 25)
'        DrawWidth = 1

        DrawBox
        FWheelNormal
'        If Mag(Vx, Vy) < 0.1 And omega < 0.5 Then
```

```
'            Exit Do
'        End If
        If chkRealTime.Value = 0 Then
            t = t + incT
        Else
            If tick = True Then
                incT = Timer2.Interval / 1000
                t = t + incT
                tick = False
            Else
                incT = 0
            End If
        End If
        totForce(1) = 0: totForce(2) = 0
        totTorq = 0
        'F(0)=0:  TotForce   TotTorq    'instantaneous
        CornerVelocity
        FrontForce
        RearForce
        sumTorq
        AccX = totForce(1) / Mass
        AccY = totForce(2) / Mass
        alpha = totTorq / MmtInertia

        massX = massX + (Vx * incT) + (0.5 * AccX * (incT ^ 2)) 'uses old Vx
        massY = massY + (Vy * incT) + (0.5 * AccY * (incT ^ 2))
        Vx = Vx + AccX * incT               'updates Vx
        Vy = Vy + AccY * incT               'updates Vy

        theta = theta + (omega * incT) + (0.5 * alpha * (incT ^ 2))
        omega = omega + alpha * incT        'updates omega

        loopflag = True
'        If t * 10 Mod 10 = 0 Then
            DoEvents
            If StopNow = True Then Exit Do
'        End If
    Loop
    Timer2.Enabled = False
    AfterMain
    AutoRedraw = True
    DrawWidth = 1
End Sub
Public Sub AfterMain()
    cmdDraw.Enabled = True
'    If NaturalEnd = True Then
'        cmdSnap.Enabled = False
'        cmdPause.Enabled = False
'        cmdStop.Enabled = False
'    End If
    cmdClear.Enabled = True
End Sub
Public Sub cornerpos()
    Dim i As Integer
'        Case "FL"
            FLxy(1, 1) = -carWidth / 2
            FLxy(1, 2) = carLength * (1 - CenterRear)
```

```
'        Case "FR"
            FLxy(2, 1) = carWidth / 2
            FLxy(2, 2) = carLength * (1 - CenterRear)
'        Case "RR"
            FLxy(3, 1) = carWidth / 2
            FLxy(3, 2) = -carLength * CenterRear
'        Case "RL"
            FLxy(4, 1) = -carWidth / 2
            FLxy(4, 2) = -carLength * CenterRear

    For i = 1 To 4
'        tmpX = FLxy(i, 1)
'        tmpY = FLxy(i, 2)
        Rotate FLxy(i, 1), FLxy(i, 2), theta
'        FLxy(i, 1) = tmpX * Cos(theta) - tmpY * Sin(theta)
'        FLxy(i, 2) = tmpX * Sin(theta) + tmpY * Cos(theta)
    Next
End Sub
Public Sub Rotate(X As Double, Y As Double, theta As Double)
    Dim tmpX As Double, tmpY As Double
    tmpX = X
    tmpY = Y
    X = tmpX * Cos(theta) - tmpY * Sin(theta)
    Y = tmpX * Sin(theta) + tmpY * Cos(theta)
End Sub
Public Sub MyPset _
(X As Double, Y As Double, FLindex As Integer)

    Static tmpX(4) As Double, tmpY(4) As Double
'    Dim i As Integer

'    i = IndexDeterm(FL)
    If Connect(FLindex) = False Then
        PSet (X, Y), Color(FLindex)
    ElseIf Connect(FLindex) = True Then
        If loopflag = False Then
            tmpX(FLindex) = X: tmpY(FLindex) = Y
            Connect(FLindex) = False
            MyPset X, Y, FLindex
            Connect(FLindex) = True
        Else
            Line (X, Y)-(tmpX(FLindex), tmpY(FLindex)), Color(FLindex)
            tmpX(FLindex) = X: tmpY(FLindex) = Y
        End If
    End If
End Sub
Public Function IndexDeterm(FL As String) As Integer
    If FL = "CE" Then
            IndexDeterm = 0
        ElseIf FL = "FL" Then
            IndexDeterm = 1
        ElseIf FL = "FR" Then
            IndexDeterm = 2
        ElseIf FL = "RR" Then
            IndexDeterm = 3
        ElseIf FL = "RL" Then
            IndexDeterm = 4
```

```
    End If
End Function
Public Sub DrawLocus()
    Dim i As Integer

    MyPset massX, massY, 0
    For i = 1 To 4
        MyPset FLxy(i, 1) + massX, FLxy(i, 2) + massY, i
    Next
'    FLx = cornerpos("FL", "X")
'    FRx = cornerpos("FR", "X")
'    RLx = cornerpos("RL", "X")
'    RRx = cornerpos("RR", "X")
'
'    FLy = cornerpos("FL", "Y")
'    FRy = cornerpos("FR", "Y")
'    RLy = cornerpos("RL", "Y")
'    RRy = cornerpos("RR", "Y")
'
'    MyPset FLx, FLy, "FL"
'    MyPset FRx, FRy, "FR"
'    MyPset RLx, RLy, "RL"
'    MyPset RRx, RRy, "RR"
End Sub
Public Sub DrawBox()
    Dim absXY(1 To 4, 1 To 2) As Double, i As Integer
    For i = 1 To 4
        absXY(i, 1) = FLxy(i, 1) + massX
        absXY(i, 2) = FLxy(i, 2) + massY
    Next

    If loopflag = True Then AutoRedraw = False
'    If snap = True Then
'        AutoRedraw = True
'        snap = False
'    End If

    Line (absXY(1, 1), absXY(1, 2))-(absXY(2, 1), absXY(2, 2))
    Line (absXY(2, 1), absXY(2, 2))-(absXY(3, 1), absXY(3, 2))
    Line (absXY(1, 1), absXY(1, 2))-(absXY(4, 1), absXY(4, 2))
    DrawWidth = 1 * dotSize
    Line (absXY(3, 1), absXY(3, 2))-(absXY(4, 1), absXY(4, 2)), vbGreen +
      vbBlue

    PSet (massX, massY), Color(0)
    For i = 1 To 4
        PSet (absXY(i, 1), absXY(i, 2)), Color(i)
    Next

    DrawWidth = dotSize
    AutoRedraw = True
End Sub
Public Function iniOmega() As Double
    'Based on Left Turn (positive omega)
    Dim circum As Double, period As Double

    If Vx0 = 0 And Vy0 = 0 Then
```

```
        iniOmega = 0
    Else
        circum = 2 * Pi * massRadius
        period = circum / Sqr(Vx0 ^ 2 + Vy0 ^ 2) 'in seconds
        iniOmega = 2 * Pi / period

        If LR = "R" Then iniOmega = -iniOmega
        If LR = "CE" Then iniOmega = 0
    End If
End Function
Public Sub CornerVelocity()
    Dim i As Integer
    For i = 1 To 4
        crnF(i, 3) = 0: crnV(i, 3) = 0
        'CornerVelocityVector= AngularVelcVector X PositionVector
        Cross 0, 0, omega, FLxy(i, 1), FLxy(i, 2), 0
        crnV(i, 1) = memMtx(1) + Vx
        crnV(i, 2) = memMtx(2) + Vy
    Next
End Sub
Public Sub FWheelNormal()
    Dim innRadius As Double, i As Integer, magFWN As Double
    innRadius = Sqr(massRadius ^ 2 _
        - (carLength * CenterRear) ^ 2) _
        - carWidth / 2

    If LR = "L" Then
        WhNormal(1, 1) = innRadius
        WhNormal(1, 2) = carLength
        WhNormal(2, 1) = innRadius + carWidth
        WhNormal(2, 2) = carLength
    ElseIf LR = "R" Then
        WhNormal(1, 1) = -innRadius - carWidth
        WhNormal(1, 2) = carLength
        WhNormal(2, 1) = -innRadius
        WhNormal(2, 2) = carLength
    ElseIf LR = "CE" Then
        WhNormal(1, 1) = 1
        WhNormal(1, 2) = 0
        WhNormal(2, 1) = 1
        WhNormal(2, 2) = 0
    End If

    For i = 1 To 2
        Rotate WhNormal(i, 1), WhNormal(i, 2), theta
        magFWN = Mag(WhNormal(i, 1), WhNormal(i, 2))
        WhNormal(i, 1) = WhNormal(i, 1) / magFWN
        WhNormal(i, 2) = WhNormal(i, 2) / magFWN
    Next

End Sub
Public Sub FrontForce()
Dim i As Integer, direction As Double, _
    mag1 As Single, mag2 As Single
'    FWheelNormal
    For i = 1 To 2
        direction = Dot(WhNormal(i, 1), WhNormal(i, 2), _
```

```
            crnV(i, 1), crnV(i, 2))
'         If direction > 0 Then
'             direction = 1
'         ElseIf direction < 0 Then
'             direction = -1
'         End If
'         direction = Sgn(direction)
          mag1 = Mag(WhNormal(i, 1), WhNormal(i, 2))
          mag2 = Mag(crnV(i, 1), crnV(i, 2))
          If Abs(direction) <= Sensitivity * mag1 * mag2 Then
              direction = 0
          Else
              direction = Sgn(direction)
          End If

          crnF(i, 1) = -direction * F(i) * WhNormal(i, 1)
          crnF(i, 2) = -direction * F(i) * WhNormal(i, 2)

          totForce(1) = totForce(1) + crnF(i, 1)
          totForce(2) = totForce(2) + crnF(i, 2)
      Next
End Sub
Public Sub RearForce()
      Dim msg As String, i As Integer
      Dim magV As Double, direction As Double, magRWN As Double


      For i = 3 To 4
          If HandBrake = True Then
              magV = Mag(crnV(i, 1), crnV(i, 2))
                  crnF(i, 1) = -F(i) * crnV(i, 1) / magV
                  crnF(i, 2) = -F(i) * crnV(i, 2) / magV
              totForce(1) = totForce(1) + crnF(i, 1)
              totForce(2) = totForce(2) + crnF(i, 2)
          Else
              WhNormal(i, 1) = 1: WhNormal(i, 2) = 0
              Rotate WhNormal(i, 1), WhNormal(i, 2), theta
              magRWN = Mag(WhNormal(i, 1), WhNormal(i, 2))
              WhNormal(i, 1) = WhNormal(i, 1) / magRWN
              WhNormal(i, 2) = WhNormal(i, 2) / magRWN

              direction = Dot(WhNormal(i, 1), WhNormal(i, 2), _
                  crnV(i, 1), crnV(i, 2))
              If direction > 0 Then
                  direction = 1
              ElseIf direction < 0 Then
                  direction = -1
              End If
              crnF(i, 1) = -direction * F(i) * WhNormal(i, 1)
              crnF(i, 2) = -direction * F(i) * WhNormal(i, 2)

              totForce(1) = totForce(1) + crnF(i, 1)
              totForce(2) = totForce(2) + crnF(i, 2)
          End If
      Next
End Sub
Public Sub sumTorq()
```

```
    Dim i As Integer, currTorq As Double

    totTorq = 0
    For i = 1 To 4
        Cross FLxy(i, 1), FLxy(i, 2), 0, crnF(i, 1), crnF(i, 2), 0
        currTorq = memMtx(3)
        totTorq = totTorq + currTorq
    Next

End Sub
Public Function Dot(Ax As Double, Ay As Double, _
Bx As Double, By As Double) As Double
    Dot = Ax * Bx + Ay * By
End Function
Public Sub Cross(Ax As Double, Ay As Double, Az As Double, _
Bx As Double, By As Double, Bz As Double)
    memMtx(1) = Ay * Bz - Az * By
    memMtx(2) = -Ax * Bz + Az * Bx
    memMtx(3) = Ax * By - Ay * Bx
End Sub
Public Function Mag(i As Double, j As Double, Optional k As Double = 0) As
      Double
    Mag = Sqr(i ^ 2 + j ^ 2 + k ^ 2)
End Function
Public Sub calcForce()
    F(1) = CoeffKin * (Mass * g) * (CenterRear)
    F(2) = F(1)
    F(3) = CoeffKin * (Mass * g) * (1 - CenterRear)
    F(4) = F(3)
End Sub
Public Sub calcMmtInertia()
    MmtInertia = Mass * (carWidth ^ 2 + carLength ^ 2) / 12
End Sub
Private Sub Form_Unload(Cancel As Integer)
    End
End Sub
Private Sub mnuOption_click()
    frmOption.Show
End Sub
Private Sub mnuReset_Click()
    Form_Load
End Sub
Private Sub mnuExit_Click()
    End
End Sub
Private Sub mnuAbout_Click()
    frmAbout.Show
End Sub
Private Sub cmdOK_Click()
    frmAbout.Hide
End Sub
Public Sub DrawScale()
Dim oriX As Double, oriY As Double, X As Double, Y As Double
Dim marginX As Double, marginY As Double, labelX As Double
Dim ratio, framePercent, scFrameWidth, twFrameWidth
Dim DPlace As Double, msg As String
Dim Aspect As Double, tmpAspect As Double
```

```
Dim DesiredWidth As Double, DesiredHeight As Double, _
    AddWidth As Double, AddHeight As Double
Dim ULx As Double, ULy As Double, _
    LRx As Double, LRy As Double
Dim tmpX0 As Double, tmpX1 As Double, _
    tmpY0 As Double, tmpY1 As Double

Cls
On Error GoTo errortrap

tmpX0 = xBegin: tmpX1 = xEnd: tmpY0 = yBegin: tmpY1 = yEnd

Scale
Aspect = frmGraph.Height / frmGraph.Width
twFrameWidth = fmeControl.Left + fmeControl.Width
framePercent = twFrameWidth / (frmGraph.Width - twFrameWidth)
scFrameWidth = (xEnd - xBegin) * framePercent

'Scale-label-scale structure for leaving space for labels _
 at right and bottom edges.

Scale (xBegin - scFrameWidth, yEnd)-(xEnd, yBegin)
labelX = Maximum(Abs(TextWidth(yBegin)), Abs(TextWidth(yEnd)))
marginX = Abs(TextWidth("X"))
marginY = Abs(TextHeight("X"))
ULx = xBegin - scFrameWidth - labelX - 5 * marginX
ULy = yEnd + 2 * marginY
LRx = xEnd + labelX + 3 * marginX
LRy = yBegin - 2 * marginY

tmpAspect = (ULy - LRy) / (LRx - ULx)
If tmpAspect > Aspect Then
    'CurrentH/DesiredW=Aspect
    DesiredWidth = (ULy - LRy) / Aspect
    AddWidth = (DesiredWidth - (LRx - ULx)) / 2
    ULx = ULx - AddWidth
    LRx = LRx + AddWidth

    DesiredWidth = (yEnd - yBegin) / Aspect
    AddWidth = (DesiredWidth - (xEnd - xBegin)) / 2
    xBegin = xBegin - AddWidth
    xEnd = xEnd + AddWidth

ElseIf tmpAspect < Aspect Then
    'DesiredH/CurrentW=Aspect
    DesiredHeight = (LRx - ULx) * Aspect
    AddHeight = (DesiredHeight - (ULy - LRy)) / 2
    ULy = ULy + AddHeight
    LRy = LRy - AddHeight

    DesiredWidth = (xEnd - xBegin) * Aspect
    AddHeight = (DesiredHeight - (yEnd - yBegin)) / 2
    yBegin = yBegin - AddHeight
    yEnd = yEnd + AddHeight
End If
Scale (ULx, ULy)-(LRx, LRy)
```

```
If (xBegin <= 0 And 0 <= xEnd) Then
    oriX = 0
Else
    oriX = xBegin
End If
If (yBegin <= 0 And 0 <= yEnd) Then
    oriY = 0
Else
    oriY = yBegin
End If

Line (xBegin, oriY)-(xEnd, oriY)
Line (oriX, yBegin)-(oriX, yEnd)

X = 0   'labeling starts at 0
Do
    If X <= xEnd Then
        Line (X, oriY - marginY / 2)-(X, oriY + marginY / 2)
            CurrentX = X - marginX / 1.5
            CurrentY = oriY - marginY / 2
                Print X
    End If
    If -X >= xBegin Then
        Line (-X, oriY - marginY / 2)-(-X, oriY + marginY / 2)
            CurrentX = -X - marginX / 1.5
            CurrentY = oriY - marginY / 2
                Print -X
    End If
    X = X + incX
   ' If X >= 1E+20 Then GoTo errortrap
Loop Until X > xEnd And -X < xBegin

Y = incY    'labeling starts at incY
Do
    DPlace = TextWidth(Str(Y))
    If Y <= yEnd Then
        Line (oriX - marginX / 2, Y)-(oriX + marginX / 2, Y)
            CurrentX = oriX - marginX - DPlace
            CurrentY = Y + marginY / 2
                Print Y
    End If
    If -Y >= yBegin Then
        Line (oriX - marginX / 2, -Y)-(oriX + marginX / 2, -Y)
            CurrentX = oriX - marginX - DPlace
            CurrentY = -Y + marginY / 2
                Print -Y
    End If
    Y = Y + incY
    'If Y >= 1E+20 Then GoTo errortrap
Loop Until Y > yEnd And -Y < yBegin

xBegin = tmpX0: xEnd = tmpX1: yBegin = tmpY0: yEnd = tmpY1
Exit Sub

errortrap:
    msg = "Something in SCALE went wrong, man!" & Chr$(13) & _
        "Visual Basic (VB) Error Number: " & Err.Number & Chr$(13) & _
```

```
        "VB Description: " & Err.Description
    MsgBox msg, vbMsgBoxHelpButton, "Error", Err.HelpFile, Err.HelpContext

xBegin = tmpX0: xEnd = tmpX1: yBegin = tmpY0: yEnd = tmpY1
Exit Sub
End Sub
Public Sub InitBoxes _
(Optional Init As Boolean = False)
Dim i As Integer

    txtInput(0).Text = carLength
    txtInput(1).Text = carWidth
    txtInput(2).Text = Mass
    txtInput(3).Text = (1 - CenterRear) * 100
    txtInput(4).Text = CoeffKin
    If Init = True Then
    txtInput(5).Text = MmtInertia
        txtInput(6).Text = Sqr(Vx0 ^ 2 + Vy0 ^ 2)
        txtInput(7).Text = 0
        With txtInput(8)
            Select Case LR
            Case "L"
                .Text = "Left"
            Case "R"
                .Text = "Right"
            Case "CE"
                .Text = "Center"
            End Select
        End With
        txtInput(9).Text = massRadius
        txtInput(10).Text = g
        txtInput(11).Text = theta
        txtInput(12).Text = -omega / (2 * Pi)
        txtInput(13).Text = incT * 1000
        txtInput(14).Text = incT * 40000
        txtInput(15).Text = dotSize
        txtInput(16).Text = x0
        txtInput(17).Text = y0
        txtInput(18).Text = Sensitivity
        For i = 0 To 4
            If Connect(i) = True Then
                chkWheel(i) = 1
            Else
                chkWheel(i) = 0
            End If
        Next
    End If
    ChangeCar = False
End Sub
Public Sub ReadBoxes()
Dim Radian(1 To 2) As Double, i As Integer, factor As Single
factor = 1E+20

    carLength = txtInput(0).Text
    carWidth = txtInput(1).Text
    Mass = txtInput(2).Text
    CenterRear = 1 - txtInput(3).Text / 100
```

```
    CoeffKin = txtInput(4).Text
    If Gyrate = False Then
        MmtInertia = txtInput(5).Text
    Else
        MmtInertia = txtInput(5).Text ^ 2 * Mass
    End If
    Radian(1) = txtInput(7).Text / 180 * Pi
        Vx0 = txtInput(6).Text * Sin(Radian(1)) 'Cos(Pi / 2 - Radian(1))
        Vy0 = txtInput(6).Text * Cos(Radian(1)) 'Sin(Pi / 2 - Radian(1))
'        Vx0 = Fix(Vx0 * factor) / factor
'        Vy0 = Fix(Vy0 * factor) / factor
'    MsgBox (Vx0)

'    Vx0 = txtInput(6).Text
'    Vy0 = txtInput(7).Text
''    If Vy0 = 0 Then
''        theta = Sgn(Vx0) * Pi / 2
''    elseif vy0
'    theta = -Atn(Vx0 / Vy0) + txtInput(11).Text

    Select Case txtInput(8).Text
    Case "Left"
        LR = "L"
    Case "Right"
        LR = "R"
    Case "Center"
        LR = "CE"
    End Select
    massRadius = txtInput(9).Text
    g = txtInput(10).Text
    Radian(2) = txtInput(11).Text / 180 * Pi
'        If Vx0 = 0 And Vy0 >= 0 Then
'            theta = 0
'        ElseIf Vx0 = 0 And Vy0 < 0 Then
'            theta = -Pi
'        Else
'            theta = Atn(Vy0 / Vx0)
'        End If
'        theta = theta + Radian + Pi / 2
        theta = -Radian(1) - Radian(2)
    omega = -txtInput(12) * 2 * Pi
    incT = txtInput(13).Text / 1000
    SnapInterval = txtInput(14).Text / 1000
    dotSize = txtInput(15).Text
    x0 = txtInput(16).Text
    y0 = txtInput(17).Text
    Sensitivity = txtInput(18).Text
    For i = 0 To 4
        If chkWheel(i) = 1 Then
            Connect(i) = True
        Else
            Connect(i) = False
        End If
    Next
End Sub
Private Sub cmdNormal_Click()
Dim tmpOmega As Double
```

```
    ReadBoxes
    tmpOmega = iniOmega
    txtInput(11).Text = 0
    txtInput(12).Text = -tmpOmega / (2 * Pi)
End Sub
Private Sub cmdInertia_click()
    ReadBoxes
    calcMmtInertia
    If Gyrate = False Then
        txtInput(5).Text = MmtInertia
    Else
        txtInput(5).Text = Sqr(MmtInertia / Mass)
    End If
End Sub
Private Sub scrX_Change(Index As Integer)
Dim xIncr As Double

    With scrX(Index)
    Select Case Index
    Case 0
        xIncr = (xEnd - xBegin) * 0.04  '25 clicks cycle
        xEnd = xEnd - Sgn(.Value) * xIncr
        xBegin = xBegin - Sgn(.Value) * xIncr
'    Case 1
'        incX = incX + 2 * (-Sgn(.Value))
    End Select
    .Value = 0
    End With

    DrawScale
End Sub
Private Sub scrY_Change(Index As Integer)
Dim yIncr As Double

    With scrY(Index)
    Select Case Index
    Case 0
        yIncr = (yEnd - yBegin) * 0.04  '25 clicks cycle
        yEnd = yEnd + Sgn(.Value) * yIncr
        yBegin = yBegin + Sgn(.Value) * yIncr
    End Select
    .Value = 0
    End With

    DrawScale
End Sub
Public Sub scrZoom_Change(Index As Integer)
Dim xIncr As Double, yIncr As Double, Incr As Single

    With scrZoom(Index)
    Select Case Index
    Case 0
        xIncr = (xEnd - xBegin) * 0.1  '10% zoom
        yIncr = (yEnd - yBegin) * 0.1  '10% zoom
            xEnd = xEnd - Sgn(.Value) * xIncr
            xBegin = xBegin + Sgn(.Value) * xIncr
            yEnd = yEnd - Sgn(.Value) * yIncr
```

```
            yBegin = yBegin + Sgn(.Value) * yIncr
    Case 1
        If incX = 1 And Sgn(.Value) = -1 Then
            .Value = 0
            Exit Sub
'        ElseIf incX < 0.2 Then
'            Incr = 0.001
'        ElseIf incX < 2 Then
'            Incr = 0.1
        ElseIf incX < 20 Then
            Incr = 1
        ElseIf incX < 100 Then
            Incr = 5
        ElseIf incX < 1000 Then
            Incr = 25
        Else
            Incr = 100
        End If

        incX = incX + Incr * Sgn(.Value)
        incY = incY + Incr * Sgn(.Value)
    End Select
    .Value = 0
    End With
    DrawScale
End Sub
Private Sub scrSelect_Change(Index As Integer)

    With txtInput(Index)
    If Index <> 8 And Index <> 18 Then 'Turning direction
        .Text = .Text + scrSelect(Index).Value / 100
    ElseIf Index = 8 Then
        If scrSelect(Index).Value > 0 Then
            If .Text = "Left" Then
                .Text = "Right"
            ElseIf .Text = "Right" Then
                .Text = "Center"
            ElseIf .Text = "Center" Then
                .Text = "Left"
            End If
        ElseIf scrSelect(Index).Value < 0 Then
            If .Text = "Left" Then
                .Text = "Center"
            ElseIf .Text = "Right" Then
                .Text = "Left"
            ElseIf .Text = "Center" Then
                .Text = "Right"
            End If
        End If
    ElseIf Index = 18 Then
        .Text = .Text + scrSelect(Index).Value / 10000
    End If
    End With
    scrSelect(Index).Value = 0
End Sub
Private Sub txtInput_Change(Index As Integer)
    If ChangeCar = False And Index <= 3 Then
```

```
        comVehicle.Text = "Custom"
    Else
        Select Case Index
'        Case 5
'            If Gyrate = True Then
'                txtInput(5).Text = Sqr(MmtInertia / Mass)
'            End If
        Case 8
            If txtInput(8).Text = "CE" Then
                txtInput(9).Enabled = False
                scrSelect(9).Enabled = False
                'lblCaption(9).Enabled = False
            Else
                scrSelect(9).Enabled = True
                txtInput(9).Enabled = True
                'lblCaption(9).Enabled = True
            End If
        End Select
    End If
End Sub
Private Sub comVehicle_Click()
    ChangeCar = True
    Select Case comVehicle.Text
        Case "Custom"
            ChangeCar = False
            Exit Sub
        Case "McLaren F1"
            carMcLaren
        Case "Imaginery Car"
            carImagine
    End Select
    InitBoxes
End Sub
Private Sub cmdsnap_KeyDown(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyEscape Then cmdPause_Click
End Sub
Private Sub cmdpause_KeyDown(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyEscape Then cmdPause_Click
End Sub
Private Sub cmdstop_KeyDown(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyEscape Then cmdPause_Click
End Sub
Private Sub cmdclear_KeyDown(KeyCode As Integer, Shift As Integer)
    If KeyCode = vbKeyEscape Then cmdPause_Click
End Sub
Public Sub carMcLaren()
    ChangeCar = True

    carLength = 4.29   'meters, 4.29x0.82
    carWidth = 1.82     '1.82
    Mass = 1139    '1139 kg
    CenterRear = 0.412  '.412
    CoeffKin = 10 / 11.16
End Sub
Public Sub carImagine()
    ChangeCar = True
```

```
    carLength = 7   'meters, 4.29x0.82
    carWidth = 2     '1.82
    Mass = 4000    '1139 kg
    CenterRear = 0.7  '.412
    CoeffKin = 0.8
End Sub
'Public Sub initToolTip()
'    lblDesc(18).ToolTipText = _
'        "0-most sensitive, 1-no sensitivity." & Chr$(13) & _
'        "0.0001 is the default, and represents a very rigidly locked " & _
'        "front-wheel steer-direction.  A value of 1 means that the front " &
 _
'        "wheels are free to turn, like a shopping cart, and therefore " & _
'        "will provide no friction."
'End Sub
Public Sub Timer1_Timer()
    timeCount = timeCount + 1
    If timeCount = 1 Then
'        frmGraph.Enabled = False
        frmSplash.Show
    ElseIf timeCount >= 25 Then
        Timer1.Enabled = False
        frmGraph.Show
'        frmGraph.Enabled = True
        frmSplash.Hide
        Unload frmSplash
    End If
End Sub
Public Sub Timer2_Timer()
    tick = True
'    t = t + incT
'    CurrentX = t / 1000: CurrentY = 1
'    'Print "."
'    MsgBox ("")
End Sub
```

FormOption

```
Option Explicit
Private Sub Form_Load()
'    MsgBox (Gyrate)
    frmGraph.Enabled = False
    If Gyrate = False Then
        optValue.Value = True
        optGyrate.Value = False
    Else
        optValue.Value = False
        optGyrate.Value = True
    End If
End Sub
Private Sub cmdOK_Click()
    If optValue.Value = True Then
        Gyrate = False
    Else
        Gyrate = True
    End If
```

```
'    MsgBox (Gyrate)
'    Form_Unload 1
    ToggleGyrate
    Unload Me
End Sub
Private Sub cmdCancel_Click()
'    Form_Unload 1
    Unload Me
End Sub
Private Sub Form_Unload(Cancel As Integer)
    frmGraph.Enabled = True
    frmGraph.Show
End Sub
Public Sub ToggleGyrate()
    With frmGraph
    If Gyrate = False Then
        .txtInput(5).Text = MmtInertia
        .lblCaption(5).Caption = _
            "Moment of Inertia (kg-m^2)"
        .lblCaption(5).ToolTipText = _
            "1000 lb-ft-sec^2 = 1355.8 kg-m^2    Must be non-zero.  Can be
      negative."
        .txtInput(5).ToolTipText = .lblCaption(5).ToolTipText
        .scrSelect(5).SmallChange = 5000
    Else
        .txtInput(5).Text = Sqr(MmtInertia / Mass)
        .lblCaption(5).Caption = _
            "Radius of Gyration (m)"
        .lblCaption(5).ToolTipText = _
            "10 ft = 3.048 m    Must be non-zero (negative OK).  Specify by
      assuming " & _
            "all mass concentrated this distance from mass center."
        .txtInput(5).ToolTipText = .lblCaption(5).ToolTipText
        .scrSelect(5).SmallChange = 10
    End If
    End With
End Sub
```

ENDNOTES

[1] The following two games are the only ones to my knowledge that appear to be scientifically accurate. Even then, however, their user-configurability is very limited, and in any case they are not useful for scientific study.

a) *Need for Speed II*, by Electronic Arts at www.nfs2.com, is a very realistic PC racing game. Thus far I have not been able to detect any artificial dynamics during game-play.

b) *Unknown name.* There is a very old (at least eight years) driving (not *racing*) simulation arcade game, which I happened to come upon recently. I was surprised to learn that it simulates very accurate dynamics.

[2] Zero speed is not allowed. However, it can be set to as small as 1e-40.

[3] The default value of the user-configurable parameter t_{incr} is 0.01 seconds, which gives reasonably accurate approximations over a wider range of speeds and configurations.