LISP PROGRAMMING OF THE "SHARP 1994" MOTORCYCLE MODEL

Simos Evangelou and David J.N. Limebeer

Department of Electrical and Electronic Engineering, Imperial College of Science, Technology and Medicine, Exhibition Road, London SW7 2BT, UK.
e-mail:d.limebeer@ic.ac.uk web page: http://www.ee.ic.ac.uk/control/motorcycles

Summary

Linear and nonlinear models are developed for the "Sharp 1994" motorcycle model [2] using the multi-body modelling package, Autosim [3]. The nonlinear part of the code uses Autosim to produce a FORTRAN program which solves the nonlinear equations of motion thereby producing time histories of the motorcycle behaviour as it evolves from an arbitrary, but given initial condition. We have studied the behaviour associated with small initial roll angles. The linear part of the code generates linearised equations of motion and produces a MATLAB^{TM} file that contains a state-space model in symbolic form. The MATLAB^{TM} code is used to generate root-locus stability plots with speed the varied parameter. The hands-on and hands-off cases are studied via the adjustment of a few parameter values.

1 Introduction

Motorcycles are multi-body systems that can be described by a large number of differential equations. The derivation of these equations is straightforward in principle, but labour intensive in practice and prone to error. Much of the work in generating these equations involves partial differentiation, dot and cross product calculations of complicated position, velocity and acceleration vectors and the manipulation of small matrices. Once these models increase in complexity, the manual derivation of the equations of motion becomes prohibitively time consuming and error prone. This is where Autosim shows its real strength, as it can be programmed to carry out algebraic manipulations effortlessly, at high speed and with minimal risk of error. In this way, the work of the dynamicist is reduced to providing the Autosim code with a correct description of the mechanism to be analyzed. The software handles all the routine computations and generates the equations of motion (nonlinear and linearised) in symbolic form.

2 Physical description of the model

The following assumptions are made regarding the vehicle under study [2]:

1. The motorcycle is represented as an assembly of rigid bodies as follows:
   - (a) Handlebars, front forks and front wheel.
   - (b) Rear frame containing the engine with components rotating about transverse axes (giving rise to gyroscopic moments), the rider's legs and lower body.
   - (c) The rear wheel assembly.
   - (d) The rider's upper body.
2. The bodies are joined together as follows: Each of bodies (a), (c) and (d) are joined to the rear frame (b) as shown in Figure 1 [2]. The joint between the front frame (a) and the rear frame (b) is the steer axis revolute joint. The damping and stiffness coefficients associated with this joint are used to represent the torques generated by the rider's arms. The housing

of the steering head bearings is connected to the rear frame by two flexible mechanisms. One allows relative lateral translation, while the other allows relative rotation about an axis perpendicular to the steering axis. Appropriate stiffnesses and damping coefficients are associated with these mechanisms.

3. The joint between the rear frame (b) and the rear wheel assembly (c) is an inclined hinge. There are stiffness and damping coefficients associated with this hinge. The upper body of the rider (d) is connected to the rear frame (b) by a longitudinal hinge at saddle height. The rider's muscular activity in remaining upright is represented by a spring-damper system.

4. The following degrees of freedom are allowed:

   - Forward and lateral motion of the reference point O, Figure 1.
   - Yaw of the rear frame.
   - Roll of the rear frame.
   - Lateral displacement of the steer axis relative to the rear frame.
   - Twist displacement of the steer axis relative to the rear frame.
   - Steering displacement of the front frame relative to the rear frame.
   - Twist displacement of the rear wheel assembly relative to the rear frame.
   - Roll displacement of the rider's upper body relative to the rear frame.

5. The tyre force and moment system is described as follows: side force and self aligning moments proportional to side-slip angle are generated. The constants of proportionality are functions of tyre load and these vary with speed since aerodynamic drag and lift forces and aerodynamic pitching moment influences are included in the model. The side force and aligning moment responses to side-slip are lagged, via a single time constant $\frac{\sigma}{u}$, in which σ is the tyre relaxation length. The relaxation length σ varies with tyre load, in accordance with measured data.

6. Side forces, aligning moments and overturning moment responses proportional to camber angle are introduced. Again, the constants of proportionality depend on tyre load, but in this case the camber force system responds instantly to changes in camber angle. The overturning moment response, also instantaneous, is calculated from the radius of curvature of the tyre cross-section. The normal reaction between the tyre and the ground moves around the cross-section as the camber angle changes. This effect is represented by a force and overturning moment at the theoretical centre of tyre/ground contact (as for an infinitely thin tyre). On this basis, the constant of proportionality between the overturning moment and the camber angle is proportional to load. These responses are not lagged, because they are geometrical in origin, rather than dependent on tyre distortions, that take time to build up.

The motorcycle is represented diagrammatically in Figure 1 [2].

3 Programming of the model

3.1 Body structure diagram

The multi-body system in Figure 1 is subdivided into its constituent bodies for the purpose of writing the Autosim code – the bodies are arranged in the parent-child relationship shown in Figure 2. The first body is the *Inertial Frame* which has the *Yaw Frame* as its only child. The *Yaw Frame* has the *Inertial frame* as its parent and the *Rear Frame* as its only child. The *Rear Frame* has the *Yaw Frame* as its parent and the *Rider Upper Body*, *Engine Flywheel*, the *Rear Wheel* and *Steering Head* assemblies as its children. The *Rear Wheel Assembly* has the *Rear Wheel* as its only child. The *Steering Head Frame* has the *Front Frame* as its only child and the *Front Frame* has the *Front Wheel* as its only child. The road wheels, *Rider Upper Body* and *Engine Flywheel* have no children.

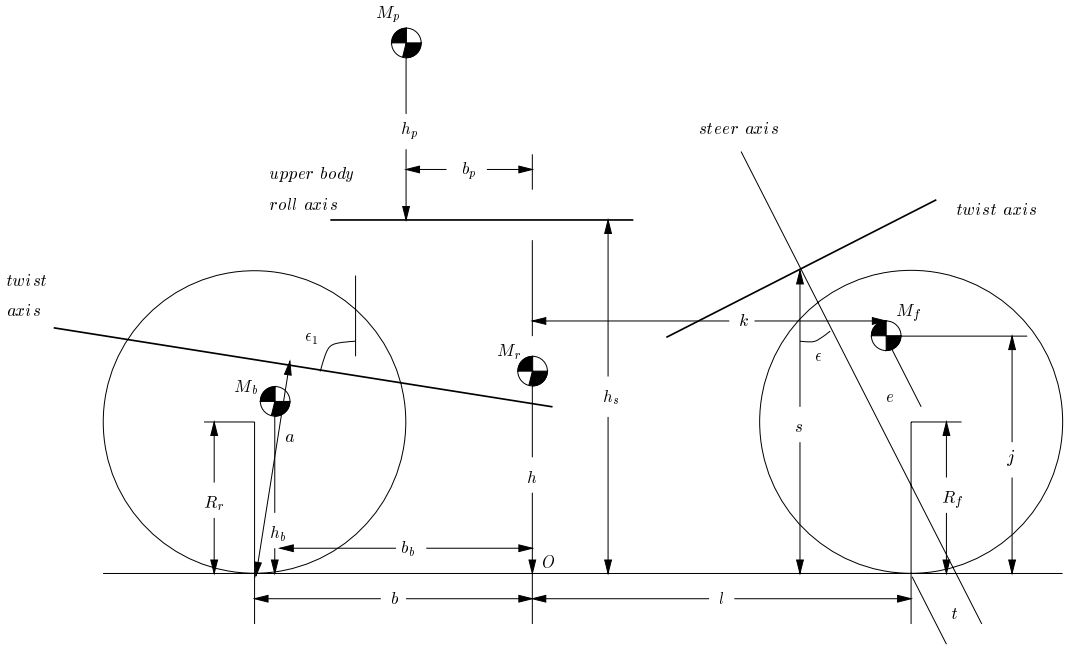

Figure 1: Diagrammatic representation of the motorcycle showing dimensions.

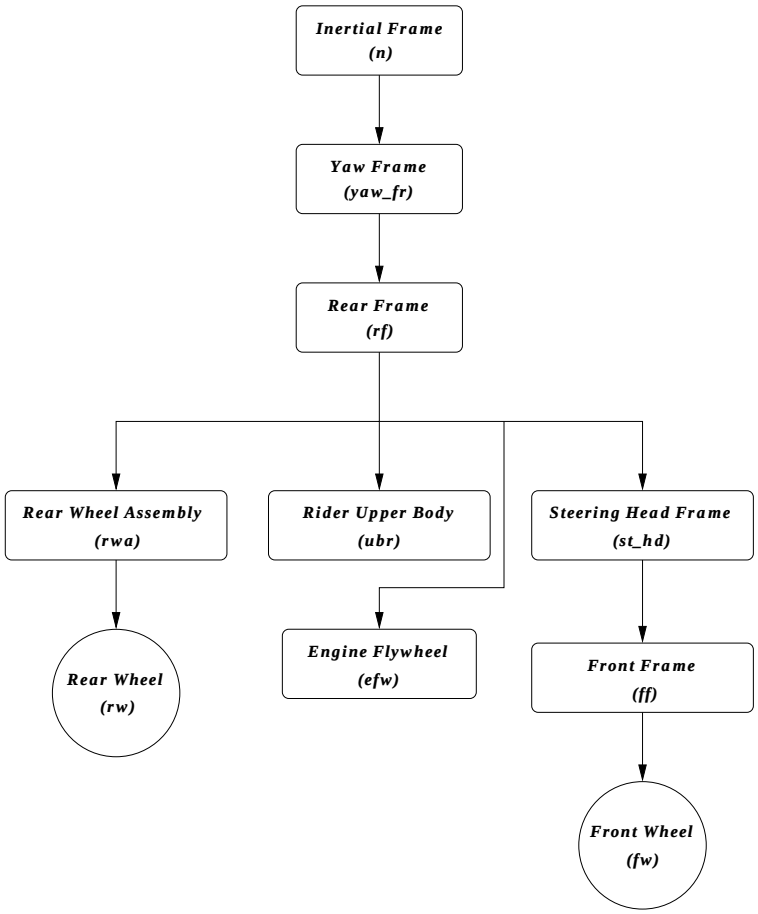

Figure 2: Body Structure Diagram of the motorcycle.

3.2 Program codes

The same Autosim code is used to generate the nonlinear and linearised models. A flag called `*linear*` is set at the beginning of the code to either true (`t`) or false (`nil`) and the appropriate parts of the code are selected or deselected so as to provide the nonlinear and linearised models. These parts of the code that are relevant to the nonlinear and linearised model building are separated via the use of the Lisp macros `unless` and `when`.

Autosim commands are used to describe the components of the motorcycle multi-body system via their parent-child relationships. The nonlinear version of the Autosim code is then used to generate the FORTRAN file that solves the nonlinear equations of motion, and the linear part is used to generate the symbolic representation of the linearised system matrices that are used to obtain root-locus plots. The programming details are described next:

- Set the flags:

```
(defvar *linear*)
(defvar *hands-on*)
(setf *linear* nil)
(setf *hands-on* nil)
```

  The `*linear*` flag is used to select nonlinear or linearised model building. The `*hands-on*` flag chooses between the hands-off and hands-on cases.

- A few preliminaries:

```
(reset)
(si)
(add-gravity)

(unless *linear* (setsym *multibody-system-name* "bk_94_ns"))
(when *linear* (setsym *multibody-system-name* "bk_94_ls"))
(setsym *double-precision* t)
```

  The `reset` line sets various global variables used by Autosim to store equations to their default values, `si` sets the units system to SI and `add-gravity` sets up a uniform gravitational field in the z-direction of the inertial frame. The next two lines name the system as `bk_94_ns` if the `*linear*` flag is set to `nil` and as `bk_94_ls` if the `*linear*` flag is set to `t`. The last line sets the `*double-precision*` variable to true so that all the FORTRAN floating-point declarations are made in double-precision.

- Some dimensions are computed here:

```
(setsym kk "ll+(ee+trail-jj*sin(epsilon))/cos(epsilon)")
(setsym whl_bs "bb+ll")
```

- Various points in the motorcycle nominal configuration within the coordinate system of body n are defined (shown diagrammatically in Figure 3):

```
(add-point p1  :name "Rear Frame centre of mass"
               :body n :coordinates (0 0 -hh))
(add-point p2  :name "Rear Wheel Assembly centre of mass"
               :body n :coordinates (-bb_b 0 -hh_b))
(add-point p3  :name "Front Frame centre of mass"
               :body n :coordinates (@kk 0 -jj))
(add-point p4  :name "Rider centre of mass"
               :body n :coordinates (-bb_p 0 "-hh_s-hh_p"))
(add-point p5  :name "Rear Wheel centre point"
               :body n :coordinates (-bb 0 -Rr))
(add-point p6  :name "Front Wheel centre point"
```

```
                :body n :coordinates (ll 0 -Rf))
(add-point p7   :name "Rear Wheel Assembly joint with Rear Frame"
                :body n :coordinates ("-bb+aa*cos(epsilon1)" 0
                                      "-aa*sin(epsilon1)"))
(add-point p8   :name "Front Twist Frame joint with Rear Frame"
                :body n :coordinates
                ("ll+trail/cos(epsilon)-ss*tan(epsilon)" 0 -ss))
(add-point p9   :name "Rider joint with Rear Frame"
                :body n :coordinates (-bb_p 0 -hh_s))
(add-point p10 :name "Rear Wheel ground contact point in n"
                :body n :coordinates (-bb 0 0))
(add-point p11 :name "Front Wheel ground contact point in n"
                :body n :coordinates (ll 0 0))
(add-point p12 :name "Centre of pressure"
                :body n :coordinates ("@whl_bs/2-bb" 0 -hh_cp))
```

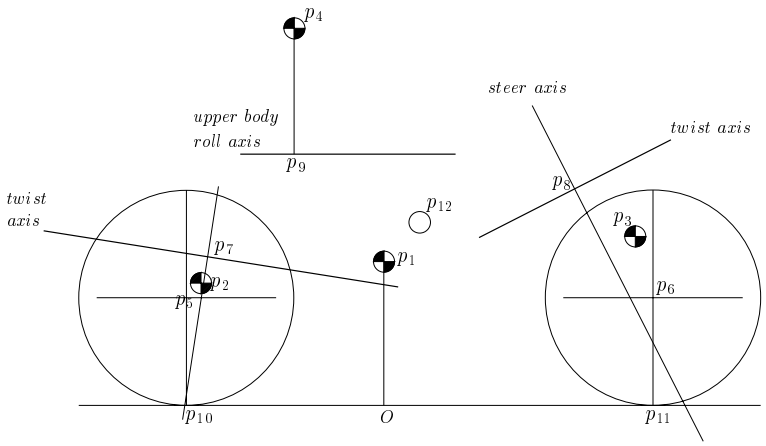

Figure 3: Diagrammatic representation of the motorcycle showing points.

Body n is the *Inertial Frame* in which all of the above points are defined. The coordinate system used to define the points is that associated with body n. The nominal configuration of the motorcycle is the upright position with zero roll, yaw, steer and twist angles and with zero forward speed. In the above, it is usual for us to use "`bb`" to represent the distance "b" in Figure 1, or "`bb_b`" to represent "b_b" and so on. The reason for this is that `t`, `g` etc. are reserved variables required by Autosim; `t` is time and `g` is acceleration due to gravity.

- The rear frame is introduced into the model next. This is done in two steps:

```
(add-body yaw_fr :name "Yaw Frame"
                 :translate (x y)
                 :body-rotation-axes z
                 :parent-rotation-axis z
                 :reference-axis x
                 :mass 0
                 :inertia-matrix 0)

(when *linear*
(add-speed-constraint "tu(yaw_fr,1) - vu" :u "tu(yaw_fr,1)")
)

(add-body rf    :name "Rear Frame"
                :parent yaw_fr
```

```
                :body-rotation-axes x
                :parent-rotation-axis x
                :reference-axis y
                :cm-coordinates p1
                :mass Mr
                :inertia-matrix ((Irx 0 Irxz)
                                 (0 0 0)
                                (Irxz 0 Irz)))
```

Firstly, the *Yaw Frame* is introduced as a massless body with translational freedoms in the x and y directions of body n (it is a child of n). Also, the *Yaw Frame* has a rotational degree of freedom in the z-direction (of body n) that describes the yawing motion of the motorcycle. The body named "Rear Frame" has the *Yaw Frame* as its parent and it has the mass and inertia properties[1] of the vehicle's entire rear frame assembly. It has one degree of rotational freedom around the *Yaw Frame's* x-axis and this freedom is used to model the rolling motion of the motorcycle. Unlike the "Sharp 1971" model [1], we do not use an `add-speed-constraint` command to constrain the forward velocity of the *Yaw Frame* unless the linear code has been selected. As will be explained later, the motorcycle speed is controlled using a speed controller in the nonlinear case.

- Add in the rider upper body:

```
(add-body ubr   :name "Rider Upper Body"
                :parent rf
                :body-rotation-axes x
                :parent-rotation-axis x
                :reference-axis y
                :joint-coordinates p9
                :cm-coordinates p4
                :mass Mp
                :inertia-matrix ((Ipx 0 Ipxz)
                                 (0 0 0)
                                (Ipxz 0 Ipz)))
```

The *Rider Upper Body* is a child of the *Rear Frame*. It has the freedom to roll relative to the *Rear Frame* and it has a mass and inertia-matrix associated with it.

- Add the rear wheel assembly:

```
(setsym rear_twist "sin(epsilon1)*[rfx] + cos(epsilon1)*[rfz]")

(add-body rwa   :name "Rear Wheel Assembly"
                :parent rf
                :body-rotation-axes x
                :parent-rotation-axis @rear_twist
                :reference-axis y
                :joint-coordinates p7
                :cm-coordinates p2
                :mass Mb
                :inertia-matrix 0)
```

Before the *Rear Wheel Assembly* is included in the code, the direction vector about which the *Rear Wheel Assembly* twists relative to the *Rear Frame* is defined via the vector (`rear_twist`). The *Rear Wheel Assembly* is also a child of the *Rear Frame* and it only has mass associated with it.

[1] Note that the inertia matrices of the *Rear Frame* and *Rider Upper Body* have been interchanged as compared with reference [2]. In addition, we should warn the reader that [2] and Autosim use different sign conventions for the products of inertia.

- Add in the rear wheel:

```
(add-body rw    :name "Rear Wheel"
                :parent rwa
                :body-rotation-axes y
                :parent-rotation-axis y
                :reference-axis z
                :joint-coordinates p5
                :mass 0
                :inertia-matrix ((irwx 0 0)
                                 (0 irwy 0)
                                 (0 0 irwx)))
```

The *Rear Wheel* is a child of the *Rear Wheel Assembly*. Its mass is set to zero, because it has already been included in the mass of the *Rear Wheel Assembly*; the *Rear Wheel* does however have inertia properties. The `:joint-coordinates p5` command line defines the coordinates of the wheel spin axis. The point `p5` has already been defined in body *n*.

- Introduce an unspun ground contact point for the *Rear Wheel*:

```
(add-point rwcp :name "Rear wheel contact point"
                :body rwa :coordinates p10)
```

This point is fixed in the *Rear Frame* and is introduced to assist with the calculation of the `vur` variable and the rear wheel side-slip angle. It is also used as the point of application of the rear tyre forces.

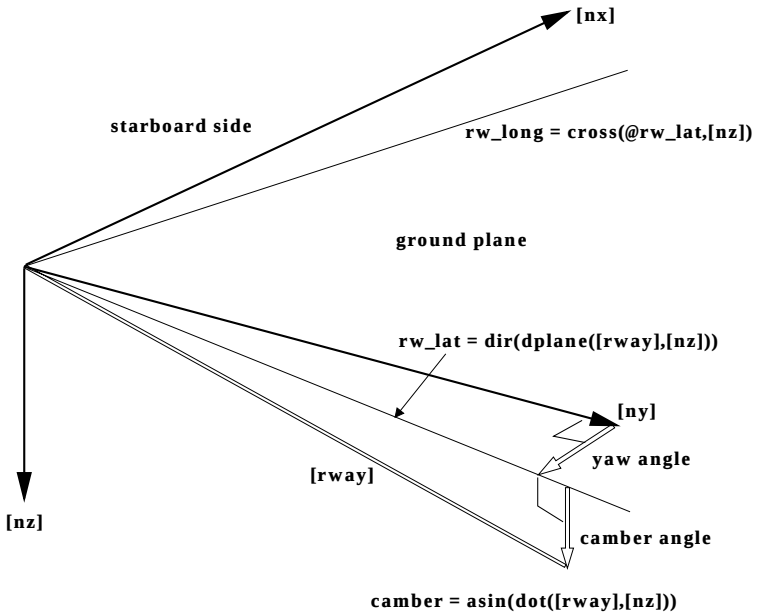

Figure 4: Wheel camber and yaw angles.

- Define the velocity component of `rwcp` along the line of intersection of the *Rear Wheel* plane and the ground plane. This will be used to compute the angular velocity of the *Rear Wheel* as follows:

```
(setsym rw_lat "dir(dplane([rway],[nz]))")
(setsym rw_long "cross(@rw_lat,[nz])")
(setsym vur "dot(vel(rwcp),@rw_long)")
```

The second line, which makes use of the first, defines the direction of the line of intersection of the *Rear Wheel* plane and the ground plane. The last line finds the velocity component of `rwcp` in the direction of `rw_long`. We refer the reader to Figure 4 for a diagrammatic representation of the various vector quantities being used.

- Assume no longitudinal slip for the *Rear Wheel*:

```
(add-speed-constraint "ru(rw)*Rr + @vur" :u "ru(rw)")
```

The rotational speed of the *Rear Wheel* is constrained to be `-vur/Rr` which means that the wheel is not allowed to slip longitudinally. The effect of this (nonholonomic) constraint is to remove the rotational speed of the rear wheel from the equations of motion.

- Define the front twist axis for the *Front Frame*:

```
(setsym front_twist "cos(epsilon)*[rfx] - sin(epsilon)*[rfz]")
```

This axis is used to assist with the addition of the front frame assembly, which is introduced into the model in two steps.

- Add in the front frame assembly:

```
(add-body st_hd :name "Steering Head Frame"
                :parent rf
                :translate y
                :body-rotation-axes x
                :parent-rotation-axis @front_twist
                :reference-axis y
                :joint-coordinates p8
                :mass 0
                :inertia-matrix 0)


(add-body ff    :name "Front Frame"
                :parent st_hd
                :body-rotation-axes z
                :parent-rotation-axis z
                :reference-axis x
                :cm-coordinates p3
                :mass Mf
                :inertia-matrix ((Ifx 0 Ifxz)
                                 (0 0 0)
                                (Ifxz 0 Ifz)))
```

To begin, the *Steering Head Frame* is used to represent lateral displacements and rotational twist freedoms between the *Front Frame* and the *Rear Frame*. The *Front Frame* is then added as a child of the *Steering Head Frame*. The steering freedom, the mass and the inertia-matrix of the front frame assembly are also included at this point.

- Add in the *Front Wheel*:

```
(add-body fw   :name "Front Wheel"
               :parent ff
               :body-rotation-axes y
               :parent-rotation-axis y
               :reference-axis z
               :joint-coordinates p6
               :mass 0
               :inertia-matrix (0 ifwy 0))
```

This body has the *Front Frame* as its parent. Its mass and x and z inertias are zero since these have been included in the *Front Frame* description. The spin inertia of the *Front Wheel* is included so that angular momentum (gyroscopic) effects are correctly represented.

- Introduce an unspun ground contact point for the *Front Wheel*:

```
(add-point fwcp :name "Front wheel contact point"
                :body ff :coordinates p11)
```

This point is fixed in the *Front Frame*. It is introduced to assist with the calculation of the `vuf` variable and the *Front Wheel* side-slip angle. It is also used as the point of application of the front tyre forces.

- Define the velocity component of `fwcp` along the line of intersection of the *Front Wheel* plane and the ground plane:

```
(setsym fw_lat "dir(dplane([ffy],[nz]))")
(setsym fw_long "cross(@fw_lat,[nz])")
(setsym vuf "dot(vel(fwcp),@fw_long)")
```

The second line, which makes use of the first, defines the direction of the line of intersection of the *Front Wheel* plane and the ground plane. The last line finds the velocity component of `fwcp` in the direction of `fw_long`.

- No longitudinal slip on the *Front Wheel*:

```
(add-speed-constraint "ru(fw)*Rf + @vuf" :u "ru(fw)")
```

As with the *Rear Wheel*, it is assumed that the *Front Wheel* undergoes no longitudinal slip. Consequently, its angular velocity is set to `-vuf/Rf` and the rotational speed of the wheel is eliminated from the equations of motion by Autosim.

- Add in the engine flywheel:

```
(add-body efw   :name "Engine flywheel"
                :parent rf
                :body-rotation-axes y
                :parent-rotation-axis y
                :reference-axis z
                :mass 0
                :inertia-matrix (0 iry 0))
```

The *Engine Flywheel* is a child of the *Rear Frame* and is located at its origin with freedom to rotate about the `y`-axis of the *Rear Frame*. The *Engine Flywheel* has a spin inertia associated with it so that the associated angular momentum effects associated with the spinning engine can be reproduced in the model.

- The *Engine Flywheel* is assumed to rotate at the same speed as the *Rear Wheel* – Its inertia is adjusted to make this accurate:

```
(add-speed-constraint "ru(efw)*Rr + @vur" :u "ru(efw)")
```

It is assumed that the *Engine Flywheel* rotates with an angular speed of `-vur/Rr` and consequently this freedom can be eliminated from the equations of motion.

- Define the camber and side-slip angles:

```
(setsym phir "asin(dot([nz],[rwy]))")
(setsym phif "asin(dot([nz],[fwy]))")
(setsym alphar "asin(dot(@rw_lat,dir(vel(rwcp))))")
(setsym alphaf "asin(dot(@fw_lat,dir(vel(fwcp))))")
```

These angles are needed in the calculation of the tyre side forces and moments. The first line defines the *Rear Wheel* camber angle, the second line defines the *Front Wheel* camber angle and the third defines the *Rear Wheel* side-slip angle by making use of the point `rwcp` defined above. The last line defines the *Front Wheel* side-slip angle via the point `fwcp`. Note that for the side-slip angles, positive lateral velocities give positive slip values and negative forces.

- Add the driving torque:

```
(unless *linear*

(add-state-variable D_tqi D_tqip "F*l")
(set-aux-state-deriv D_tqip "vu-tu(yaw_fr,1)")
(setsym D_tq "kp*(vu-tu(yaw_fr,1))+ki*D_tqi")

(add-moment Dr_mom        :name "Rear wheel drive torque"
                          :body1 rw     :body2 rf
                          :direction [rwy]
                          :magnitude -@D_tq)
)
```

Unlike the "Sharp 1971" model [1], this code uses a PI control loop to maintain a constant forward velocity for the *Yaw Frame* for the nonlinear case. This controller is shown in Figure 5. The first two lines define the integral part of the control loop. All the contributions

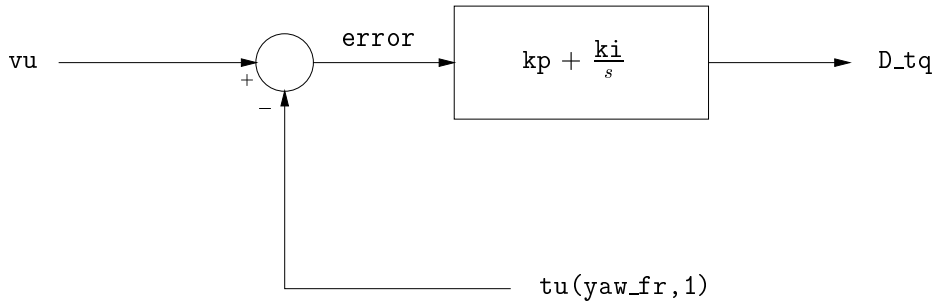

Figure 5: Control loop on the forward speed.

are added on the last line. The drive torque is then applied to the *Rear Wheel* with the reaction coming from the *Rear Frame*. If the `*linear*` flag is set to `t` then no driving torque is added and the `add-speed-constraint` command discussed earlier is used. This change was introduced to obviate the problem of tracking the initial value of `D_tq` as a function of speed.

- Introduce various damping and stiffness forces and moments:

```
(add-moment ubr_tq      :name "Rider Upper Body Damping and Stiffness"
                        :body1 ubr      :body2 rf
                        :direction [ubrx]
                        :magnitude "-k_zita*rq(ubr) - D_zita*ru(ubr)")

(add-moment rwa_tq      :name "Rear Twist Damping and Stiffness"
                        :body1 rwa      :body2 rf
                        :direction [rwax]
                        :magnitude "-k_lamda*rq(rwa) - D_lamda*ru(rwa)")

(add-line-force st_hd_f :name "Steering Head Lateral Stiffness"
```

```
                        :direction [st_hdy]
                        :point1 st_hd0   :point2 st_hdj
                        :magnitude "-k_v*tq(st_hd) - D_v*tu(st_hd)")

(add-moment st_hd_tq    :name "Front Twist Torque"
                        :body1 st_hd     :body2 rf
                        :direction [st_hdx]
                        :magnitude "-k_gamma*rq(st_hd) - D_gamma*ru(st_hd)")

(add-moment steer_torq  :name "Steering Damping and Stiffness"
                        :body1 ff       :body2 st_hd
                        :direction [ffz]
                        :magnitude "-k_steer*rq(ff)-D_steer*ru(ff)+rid_tq")
```

All the moments inserted here involve moments generated by torsional springs and dampers except from the last one which includes an external torque input from the rider – this defaults to zero. The first one is the moment between the *Rider Upper Body* and the *Rear Frame*. The second, between the *Rear Wheel Assembly* and the *Rear Frame*. The third moment is between the *Steering Head Frame* and the *Rear Frame* and the final one is between the *Front Frame* and the *Steering Head Frame*. The `add-line-force` command introduces a lateral force between the *Steering Head Frame* and the *Rear Frame* via a spring and damper. Appropriate values for the spring and damper constants are given at the end of the program listing.

- Introduce the aerodynamic drag and lift forces:

```
(add-point p13 :name "Centre of Pressure in rf" :body rf :coordinates p12)

(add-line-force drag    :name "Aerodynamic Drag"
                        :direction [rfx]
                        :point1 p13
                        :magnitude "-Dc*(tu(yaw_fr,1)**2)")

(add-line-force lift    :name "Aerodynamic Lift"
                        :direction [rfz]
                        :point1 p13
                        :magnitude "-Lc*(tu(yaw_fr,1)**2)")
```

The point `p13` is used to define the centre of pressure which is a point attached on the *Rear Frame*. The aerodynamic drag and lift forces are both applied here. These forces are proportional to the square of the forward speed (`tu(yaw_fr,1)**2`).

- Work out the normal tyre loads:

```
(setsym Zf "((Mf*(bb+@kk)+Mr*bb+Mb*(bb-bb_b)+Mp*(bb-bb_p))*G
-(vu**2)*(Dc*hh_cp+Lc*@whl_bs/2))/@whl_bs")
(setsym Zr "(Mf+Mr+Mp+Mb)*G-Lc*vu**2-@Zf")
```

The purpose of the above is to compute the normal tyre loads under a steady-state speed condition. This is done by taking moments about the rear wheel ground contact point.

- Calculate tyre parameters:

```
(setsym Cfvr "-92.9+23.129*@Zr-(4.663/1000)*@Zr**2-(6.457*10**(-7))*@Zr**3
+(1.887*10**(-10))*@Zr**4")
(setsym Cfvf "-300+28.577*@Zf-0.0143*@Zf**2+(1.431*10**(-6))*@Zf**3
+(3.347*10**(-10))*@Zf**4")
(setsym Cmvr "9+0.3573*@Zr + (3.378*10**(-5))*@Zr**2")
(setsym Cmvf "-0.281+0.2442*@Zf+(8.575*10**(-5))*@Zf**2")
```

```
(setsym Cr1 "27.38+0.9727*@Zr-(4*10**(-6))*@Zr**2")
(setsym Cf1 "-13.25+1.302*@Zf-(1.39*10**(-4))*@Zf**2")
(setsym Cr2 "2.056+0.01282*@Zr+(4.928*10**(-6))*@Zr**2")
(setsym Cf2 "2.788+0.0165*@Zf+(3.9*10**(-6))*@Zf**2")
(setsym Cr3 "-0.07*@Zr")
(setsym Cf3 "-0.06*@Zf")
(setsym sigmar "0.03594+(1.941*10**(-4))*@Zr-(5.667*10**(-8))*@Zr**2
+(5.728*10**(-12))*@Zr**3")
(setsym sigmaf "0.1012+(1.297*10**(-4))*@Zf-(3.267*10**(-8))*@Zf**2")
```

Since aerodynamics effects are included, the tyre loads will vary with forward speed as will the various tyre parameters.

- Introduce a simple tyre relaxation model:

```
(add-state-variable Yr Yr_dot F)
(set-aux-state-deriv Yr_dot "(@vur/@sigmar)*( -@Cfvr*@alphar - Yr)")

(add-state-variable Yf Yf_dot F)
(set-aux-state-deriv Yf_dot "(@vuf/@sigmaf)*( -@Cfvf*@alphaf - Yf)")

(add-state-variable Mzr Mzr_dot "F*l")
(set-aux-state-deriv Mzr_dot "(@vur/@sigmar)*( @Cmvr*@alphar - Mzr)")

(add-state-variable Mzf Mzf_dot "F*l")
(set-aux-state-deriv Mzf_dot "(@vuf/@sigmaf)*( @Cmvf*@alphaf - Mzf)")
```

The `add-state-variable` commands introduce four state variables, one for each of the tyre force expressions and one for each of the tyre moment expressions. These states are used to describe the tyre relaxation properties. The force and moment equations are defined with the `set-aux-state-deriv` expressions and use the values of the side-slip, camber angle and tyre parameters defined above. The `set-aux-state-deriv` commands replace the earlier `add-equation` commands. Notice the minus sign on the `Cr1*@alphar` and `Cf1*@alphaf` terms.

- Introduce the tyre side forces and moments:

```
(add-line-force Yrt     :name "Total Rear Lateral Force"
                        :direction @rw_lat
                        :point1 rwcp
                        :magnitude "@Cr1*@phir + Yr")

(add-line-force Yft     :name "Total Front Lateral Force"
                        :direction @fw_lat
                        :point1 fwcp
                        :magnitude "@Cf1*@phif + Yf")

(add-moment Mzrt        :name "Total Rear Aligning Moment"
                        :body1 rwa
                        :direction [nz]
                        :magnitude "@Cr2*@phir + Mzr")

(add-moment Mzft        :name "Total Front Aligning Moment"
                        :body1 ff
                        :direction [nz]
                        :magnitude "@Cf2*@phif + Mzf")

(add-moment Mxr         :name "Rear Righting Moment"
```

```
                        :body1 rwa
                        :direction @rw_long
                        :magnitude "@Cr3*@phir")

(add-moment Mxf         :name "Front Righting Moment"
                        :body1 ff
                        :direction @fw_long
                        :magnitude "@Cf3*@phif")
```

The two forces are the tyre side forces and their directions are in the ground plane and normal to the line of intersection of the ground plane and the wheel plane. The tyre models include side forces due to the rolling and are introduced without relaxation, because these forces are produced by geometric effects. Next, the aligning moments are introduced with terms due to side-slip and terms due to camber. As with the side forces, relaxation effects are only associated with the side-slip components. The direction of these moments is the z-axis of the inertial frame. Finally, we introduce the front and rear tyre overturning moments. These moments are applied in the `fw_long` and `rw_long` direction respectively. Since these moments are purely geometric in character, they do not have relaxation effects associated with them.

- Incorporate the longitudinal and normal tyre loads:

```
(setsym Xr "-(Crr1 + Crr2*@vur**2)*@Zr")
(setsym Xf "-(Crr1 + Crr2*@vuf**2)*@Zf")

(add-line-force Xrr     :name "Rear Longitudinal Force"
                        :direction @rw_long
                        :point1 rwcp
                        :magnitude @Xr)

(add-line-force Xff     :name "Front Longitudinal Force"
                        :direction @fw_long
                        :point1 fwcp
                        :magnitude @Xf)

(add-line-force Wr      :name "Rear Vertical Force"
                        :direction [nz]
                        :point1 rwcp
                        :magnitude "-@Zr")

(add-line-force Wf      :name "Front Vertical Force"
                        :direction [nz]
                        :point1 fwcp
                        :magnitude "-@Zf")
```

The two longitudinal forces represent the rolling resistance to the forward motion of the motorcycle wheels and have a magnitude that depends on the forward speed of the vehicle. These forces are proportional to the normal tyre loads as defined in the two `setsym` expressions. The normal tyre forces are introduced into the vehicle equations of motion by the last two commands.

- Derive the equations of motion of the system or the linearised equations:

```
(unless *linear* (dynamics))
(when *linear*
(add-variables dyvars real rid_tq)
(linear :u rid_tq)
)
```

If the `*linear*` flag is set to `nil` the full equations of motion are derived, otherwise the linearised equations are computed with `rid_tq` as the input. `rid_tq` is defined as a real variable that is used as a steering torque input from the rider, and therefore the corresponding B-Matrix is also computed in the linearised equations.

- All the motorcycle parameters are introduced with their names and default values[2]:

```
;;; Default values for masses:
(set-names      Mf      "Mass of Front Frame"
                Mr      "Mass of Rear Frame"
                Mb      "Mass of Rear Wheel Assembly"
                Mp      "Mass of Upper Body of Rider")
(set-defaults   Mf 40.59 Mr 170.3 Mb 25.0 Mp 50.0)

;;; Default values for moments of inertia:
(set-names      Ifx     "front frame inertia about x-axis"
                Ifz     "front frame inertia about z-axis"
                Ifxz    "front frame inertia product"
                Ipx     "rear frame inertia about x-axis"
                Ipz     "rear frame inertia about z-axis"
                Ipxz    "rear frame inertia product"
                Irx     "rider upper body inertia about x-axis"
                Irz     "rider upper body inertia about z-axis"
                Irxz    "rider upper body inertia product"
                irwx    "rear wheel inertia about x-axis"
                irwy    "rear wheel spin inertia"
                ifwy    "front wheel spin inertia"
                iry     "effective engine flywheel inertia")
(set-defaults   Ifx 3.97 Ifxz 0 Ipx 1.96 Ipz 0.55 Ipxz 0.26 Irx 7.43
                Irz 11.63 Irxz 7.4 irwx 0.4 irwy 0.65 ifwy 0.58 iry 0.41)

;;; Geometric parameters:
(set-names      Rr      "Rear wheel radius"
                Rf      "Front wheel radius"
                hh_cp   "Centre of pressure height")
(set-defaults   aa 0.527 bb 0.628 bb_b 0.62 bb_p 0.28 ee 0.049 hh 0.438
                hh_b 0.33 hh_p 0.4 hh_s 0.8 jj 0.527 ll 0.807
                Rf 0.336 Rr 0.321 ss 0.77 trail 0.094
                epsilon 0.47 epsilon1 1.435 hh_cp 0.33)

;;; Frame flexibility and rider parameters:
(set-names      k_v     "steering head lateral stifness coefficient"
                D_v     "steering head lateral damping coefficient"
                k_lamda "rear wheel assembly twist stiffness coefficient"
                D_lamda "rear wheel assembly twist damping coefficient"
                k_gamma "steering head torsional stiffness coefficient"
                D_gamma "steering head torsional damping coefficient"
                k_zita  "rider upper body restraint stiffness coefficient"
                D_zita  "rider upper body restraint damping coefficient"
                k_steer "steer stiffness coefficient"
                D_steer "steer damping coefficient")
(set-defaults   k_v "1.04*(10**6)" D_v 456 k_lamda 46000 D_lamda 17.7
                k_gamma 61200 D_gamma 44.1 k_zita 10000 D_zita 156)

(unless *hands-on* (set-defaults Ifz 0.71 k_steer 0.0 D_steer 1.0))
```

[2] It can be seen from the signs of `Ipxz` and `Irxz` that Autosim uses a negative sign convention, as compared with [2], for products of inertia.

```
(when *hands-on* (set-defaults Ifz 0.91 k_steer 50.0 D_steer 6.0))

;;; Tyre parameters:
(set-names      @Cfvf   "Front Tyre Cornering Stiffness"
                @Cfvr   "Rear Tyre Cornering Stiffness"
                @Cmvf   "Front Tyre Aligning Moment Stiffness"
                @Cmvr   "Rear Tyre Aligning Moment Stiffness"
                @Cf1    "Front Tyre Camber Stiffness"
                @Cr1    "Rear Tyre Camber Stiffness"
                @Cf2    "Front Tyre Aligning Moment Camber Coefficient"
                @Cr2    "Rear Tyre Aligning Moment Camber Coefficient"
                @Cf3    "Front Tyre Overturning Moment Coefficient"
                @Cr3    "Rear Tyre Overturning Moment Coefficient"
                Crr1    "Tyre rolling resistance coefficient 1"
                Crr2    "Tyre rolling resistance coefficient 2")
(set-defaults   Crr1 0.018 Crr2 "6.8*10**(-6)")

;;; Other parameters:
(set-names      Dc      "Aerodynamic Drag Coefficient"
                Lc      "Aerodynamic Lift Coefficient")
(set-defaults   Dc 0.377 Lc 0.05 vu 53.5)
(unless *linear* (set-defaults step 0.001 stopt 20 iprint 10 rid_tq 0
                      ki 300 kp 300 "tu(yaw_fr,1)" vu "rq(rf)" 0.005))
```

The `*hands-on*` flag is used to choose the appropriate values for the parameters corresponding to the hands-off and hands-on cases. Strictly speaking, for the hands-on case, the reaction from the steering restraint torque (`steer_tq`) comes from both the *Steering Head* (`st_hd`) and the *Rider Upper Body* (`ubr`) and not from the *Steering Head* as in the hands-off case. The last line sets some default values relevant only to the nonlinear model.

- Write up files:

```
(unless *linear*
(write-to-file write-sim "bk_94_ns.f")
(write-to-file print-default-positions "positions.txt")
(write-to-file print-default-directions "directions.txt")
(write-to-file print-parameters "parameters.txt")
)
(when *linear*
(write-to-file write-matlab "bk_94_ls.m")
)
```

The FORTRAN file is written to `bk_94_ns.f` and the files `positions.txt`, `directions.txt` and `parameters.txt` are used to store the default positions, directions and parameters if the `*linear*` flag is set to `nil`. This information is useful as a debugging aid. If the linearised model is asked then the MATLAB^{TM} file `bk_94_ls.m` is written.

4 Simulations and Results

The nonlinear code is used to generate the nonlinear equations of motion in the form of a FORTRAN code. This code can then be used to generate time histories of the various dynamic variables (positions, velocities, accelerations, forces and so on). A typical plot of a time history for the hands-off case is shown in Figure 6. In this case the forward speed is held constant at 53.5 $\frac{m}{s}$ and there is an initial non-zero roll angle of 0.005 *rad*. The z-rotational speed of `ff` relative to `st_hd` undergoes an initial transient and then settles to zero, because the system is stable in the straight running configuration for this value of forward speed. This transient has two frequency components, one being fast and the other slow. The slow mode corresponds to the weave mode

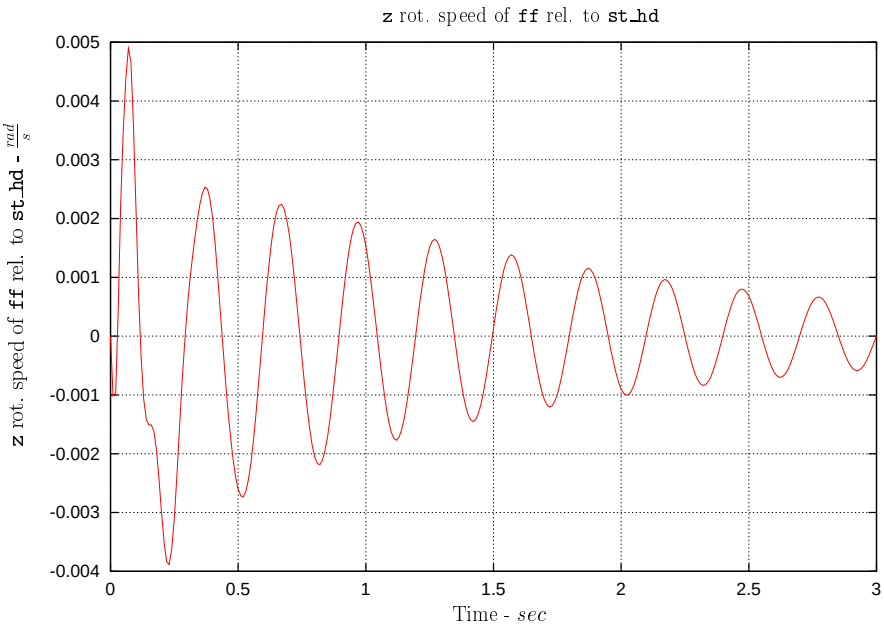

Figure 6: z rot. speed of `ff` rel. to `st_hd`.

of the motorcycle and has a frequency of about 22.85 $\frac{rad}{s}$, while the high frequency mode is the so called wobble mode which has a frequency of about 59.14 $\frac{rad}{s}$. We refer to Appendix A in the report [4] for a detailed discussion of these modes.

The linear code is used to generate the linearised equations of motion about straight running equilibrium conditions. In the equilibrium state, the motorcycle is moving with constant forward speed and with zero roll, yaw, steer and twist angles. Figure 7 shows root-locus plots in which speed is the varied parameter. The first part of Figure 7 is a plot for the hands-off case, while the second part of Figure 7 is a root-locus plot for the hands-on case. These figures agree with Figures 3 and 4 in the original paper [2].

5 Conclusions

The aim of this work is to demonstrate that the results presented in [2] can be reproduced by the multi-body modelling code Autosim. As is the case with many nonlinear systems, local stability is investigated via the eigenvalues of linearised models that are associated with equilibrium points. In our case the linearisations were taken about constant-speed straight running conditions. Autosim can be used to generate time histories from the nonlinear equations of motion, and most usefully, it can be used to generate linearised state-space models in symbolic form. The linearised models can be imported into MATLAB^{TM} for evaluation. A typical local stability study will require time histories from the nonlinear model and the symbolic linearised equations of motion generated by the linear Autosim code. The nonlinear equations are stored in the FORTRAN file `bk_94_ns.f` and the linearised equations of motion are stored in the MATLAB^{TM} file `bk_94_ls.m`. In order to construct the root-loci in Figure 7, two lines have to be removed from this code: the first line which is a `clear` statement and second a line containing the `VU=53.5` statement. The modified version of the MATLAB^{TM} file is stored in `bk_94_ls.m` and the diagrams in Figure 7 may be generated using this file and the hand-written plotting code `bk_94_rootlocus.m`.

A Errata in the "Sharp 1994" motorcycle model

The following errors appear in the original paper [2]:

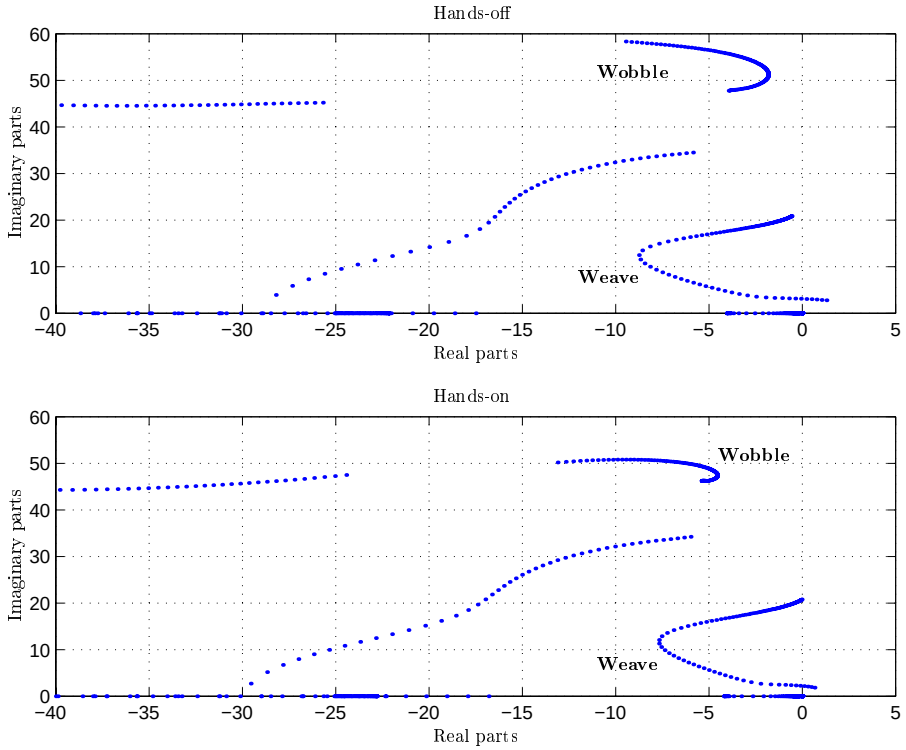

Figure 7: Root-loci for the weave and wobble modes of baseline machine and rider for the speed range 5 - 53.5 $\frac{m}{s}$.

- The *Rear Frame* moments and products of inertia must be interchanged with those of the *Rider Upper Body*.
- The tyre model polynomials are corrected in this report so that they fit the graphs given in [2].

More errors and misprints appear in [2] but they are not included here since they are irrelevant to our case.

References

[1] R. S. Sharp, "The stability and control of motorcycles", Jour. Mech. Eng. Sci., Vol. 13, No. 5, 1971, pp. 316-329.

[2] R. S. Sharp, "Vibrational modes of motorcycles and their design parameter sensitivities", Vehicle NVH and Refinement, Mech. Eng. Publ., London, 1994, pp. 107-121.

[3] Mechanical Simulation Corporation, "Autosim 2.5+ Reference Manual", 1998, *http://www.trucksim.com.*

[4] S. Evangelou and D.J.N. Limebeer, "Lisp programming of the "Sharp 1971" motorcycle model", 2000, *http://www.ee.ic.ac.uk/control/motorcycles.*