LISP PROGRAMMING OF THE "SHARP 1971" 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 1971" motorcycle model [1] using the multi-body modelling package, Autosim [2]. 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 a plot of the real parts of the eigenvalues of the motorcycle system for a sequence of forward speeds. By making minor changes to the MATLAB^{TM} code and using a hand-written plotting code, it is possible to present these same stability results in root-locus form with speed the varied parameter.

1 Introduction

Motorcycles are multi-body systems that are described by six 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 (3x3) 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 multi-body 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 representation of the vehicle [1]:

1. The vehicle consists of two rigid frames that are joined together via a conventional steering mechanism. This steering freedom is constrained by a linear steering damper.
2. The front frame consists of the front wheel, forks, handlebars and fittings.
3. The rear frame consists of the main structure, the engine-gearbox assembly, the petrol tank, seat, rear swinging arm, the rear wheel and a rigidly attached rider.
4. Each frame has a longitudinal plane of symmetry and the axis through the front frame mass centre parallel to the steering axis is a principal axis.
5. The road wheels are rigid discs each of which makes point contact with the road. They roll without longitudinal slip on a flat level road surface.

6. The axis of rotation of the engine flywheel is transverse.
7. The machine moves at constant forward speed with freedom to side slip, yaw, and roll; only small perturbations from straight running are considered.
8. The air through which the machine moves is stationary and the effects of aerodynamic side forces, yawing moments and rolling moments will be small compared with the tyre effects and are therefore neglected. The effects of drag, lift and pitching moment are to modify the vertical loading of the tyres and to make necessary a longitudinal force at the driving wheel sufficient to maintain the assumed constant forward speed. These effects are accounted for by variations in the coefficients relating tyre side forces to side-slip and camber angles.
9. Pneumatic trail of the tyres is not considered since, for the rear tyre, its effect will be very small, and for the front tyre it is small compared with the mechanical trail.
10. The drag force at the front tyre is small compared with the tyre side forces.

The motorcycle is represented diagrammatically in Figure 1 [1]:

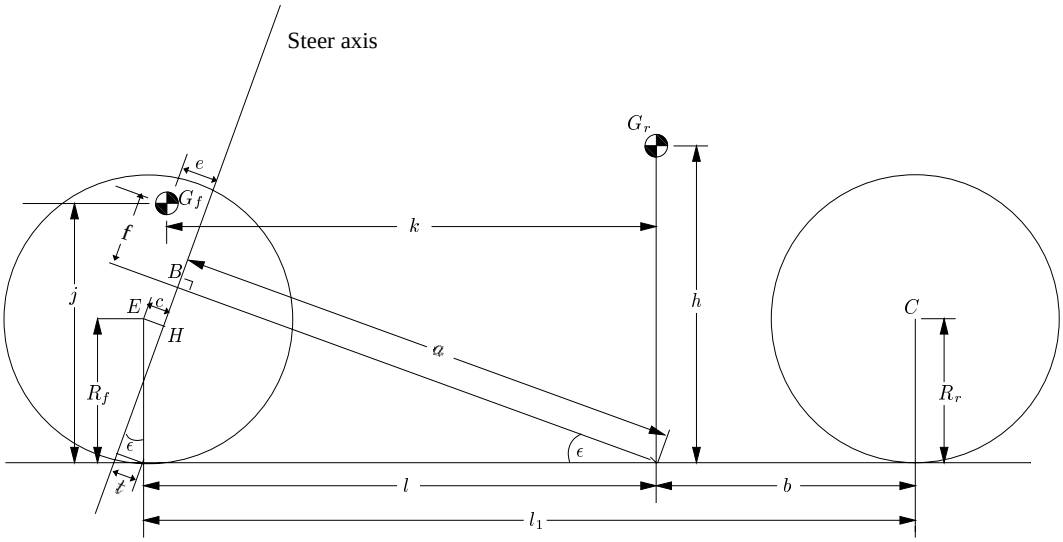

Figure 1: Diagrammatic representation of the motorcycle

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 a parent-child relationship as shown in Figure 2. The first body is the *Inertial Frame* and it 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 *Rear Wheel* and *Front Frame* as its children. The *Front Frame* has the *Front Wheel* as its only child. The road wheels have no children.

3.2 Program code

The same Autosim code is used to generate the nonlinear and linearised models. The linear and nonlinear parts of the code are separated using a "linear" flag and the Lisp macros `unless` and `when`. The flag called `*linear*` is set to be true (`t`) or false (`nil`) at the beginning of the code thereby separating the linear and nonlinear parts of the code. The nonlinear part of the Autosim

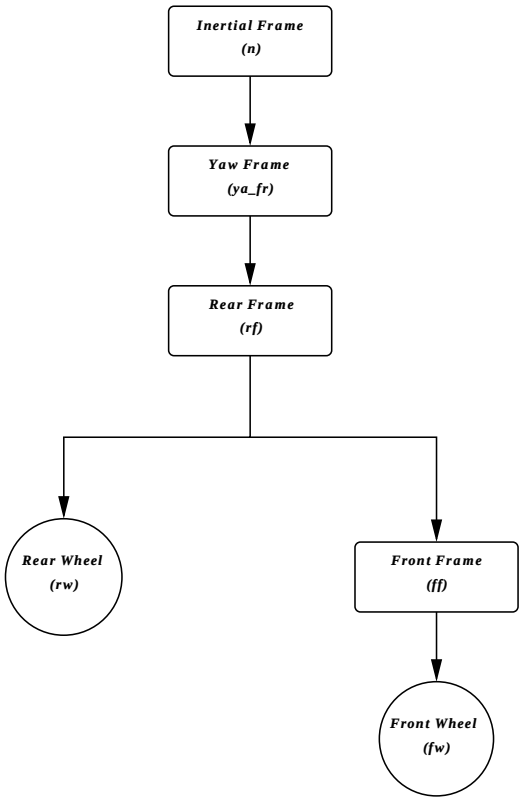

Figure 2: Body Structure Diagram of the motorcycle

code is then used to generate the FORTRAN file that is used to solve the nonlinear equations of motion, and the linear part is used to generate the symbolic representation of the linearised system matrices which are used to obtain root-locus plots.

Autosim commands are used to describe the components of the motorcycle multi-body system in their parent-child relationship. The programming details are described next:

- Set the flag for linear or nonlinear:

```
(defvar *linear*)
(setf *linear* t)
```

- A few preliminaries:

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

(unless *linear* (setsym *multibody-system-name* "bk_71_ns"))
(when *linear* (setsym *multibody-system-name* "bk_71_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_71_ns` if the `*linear*` flag is set to `nil` and as `bk_71_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.

- Various points in the motorcycle nominal configuration within the coordinate system of body n are defined:

```
(add-point Gr        :name "Rear frame centre of mass"
                     :body n :coordinates (0 0 -h))
(add-point rw_centre :name "Rear wheel centre point"
                     :body n :coordinates (-bb 0 -Rr))
(add-point ff_joint  :name "Front frame joint point with rear frame"
                     :body n :coordinates ("aa*cos(epsilon)"
                      0 "-aa*sin(epsilon)"))
(add-point Gf        :name "Front frame centre of mass"
                     :body n :coordinates (kk 0 -jj))
(add-point fw_centre :name "Front wheel centre point"
                     :body n :coordinates (ll 0 -Rf))
(add-point rwcpn     :name "Rear wheel ground contact point in n"
                     :body n :coordinates (-bb 0 0))
(add-point fwcpn     :name "Front wheel ground contact point in n"
                     :body n :coordinates (ll 0 0))
```

  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 and steer 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 and so on. The reason for this is that `t`, `g` etc. are reserved variables required by Autosim; `t` is time and `g` is gravitational acceleration constant.

- The rear frame is built into the model next:

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

(add-speed-constraint "tu(ya_fr,1) - vu" :u "tu(ya_fr,1)")

(add-body rf :name "Rear Frame"
             :parent ya_fr
             :body-rotation-axes x  :parent-rotation-axis x
             :reference-axis y  :cm-coordinates Gr
             :mass Mr
             :inertia-matrix ((Irx 0 Crxz)
                              (0 Iry 0)
                             (Crxz 0 Irz)))
```

  This is done in two steps. Firstly, the *Yaw Frame* is introduced as a massless body with translational degrees of freedom along the x and y directions of body n (it is a child of n). The *Yaw Frame* has a further rotational degree of freedom in the z direction that describes the yawing motion of the motorcycle. The second body is the child of the *Yaw Frame* and is called the *Rear Frame*. This body possesses the mass and moments of inertia of the whole rear frame assembly and also a rotational degree of freedom in the x-direction of the *Yaw Frame* which is used to describe the rolling motion of the motorcycle. The `add-speed-constraint` command constrains the forward velocity of the *Yaw Frame*, and therefore of the motorcycle, to be equal to `vu`, which is the forward speed parameter defined at the end of the program.

- Add in the rear wheel:

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

The *Rear Wheel* is a child of the *Rear Frame*. Its mass is set to zero, because it is included in the mass of the *Rear Frame*, but its inertia matrix is inserted here. The `:joint-coordinates rw_centre` command line defines the coordinates of the joint of the rear wheel and the rear frame using the coordinates of a point defined above in *n*.

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

```
(add-point rwcp :name "Rear wheel contact point"
                :body rf :coordinates rwcpn)
```

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

- 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*:

```
(setsym vur "dot(vel(rwcp),[ya_frx])")
```

- 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 steering and reference axis for the *Front Frame*:

```
(setsym steer_axis "sin(epsilon)*[rfx] + cos(epsilon)*[rfz]")
(setsym fw_reference "cos(epsilon)*[rfx] - sin(epsilon)*[rfz]")
```

These two axes are defined to assist the addition of the front frame assembly.

- Add in the *Front Frame*:

```
(add-body ff :name "Front Frame"
             :parent rf
             :body-rotation-axes z
             :parent-rotation-axis @steer_axis
             :reference-axis @fw_reference
             :joint-coordinates ff_joint
             :cm-coordinates Gf
             :mass Mf
             :inertia-matrix (Ifx Ify Ifz))
```

The *Front Frame* is a child of the *Rear Frame*. It has one degree of freedom, that is a steering freedom about the steering axis (`steer_axis`). The reference axis (`fw_reference`) is used to define the nominal configuration of the *Front Frame*.

- 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 fw_centre
             :mass 0
             :inertia-matrix (ifwx ifwy ifwx))
```

This body has the *Front Frame* as parent. Its mass is zero since this has been included in the mass of the *Front Frame*, but its moments of inertia are not and so they are inserted here.

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

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

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.

- 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 this wheel is eliminated from the equations of motion by Autosim.

- Define the camber and side-slip angles:

```
(setsym phir "asin(dot([nz],[rfy]))")

(setsym phif "asin(dot([nz],[ffy]))")

(setsym alphar "asin(dot([ya_fry],dir(vel(rwcp))))")

(setsym alphaf "asin(dot(@fw_lat,dir(vel(fwcp))))")
```

These angles are needed in the calculation of the side forces. The first line defines the *Rear Wheel* camber angle, the second line defines the *Front Wheel* camber angle and the third line 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. As a consequence the signs of the side-slip angles in the side force expressions below are opposite to those in the original paper [1].

- Introduce a steering head damping torque:

```
(add-moment sd :name "Steering Damping"
               :body1 ff :body2 rf
               :direction [ffz]
               :magnitude "-K*ru(ff)+st_tq")
```

This torque acts on the *Front Frame* with a positive magnitude and on the *Rear Frame* with negative magnitude in the z-direction of the *Front Frame*. Its magnitude is proportional to the rotation speed of the *Front Frame* relative to the *Rear Frame*. There is also a contribution from the rider (`st_tq`) which defaults to zero.

- Work out the tyre side forces and introduce a simple tyre relaxation model:

```
(add-state-variable Yr Yr_dot F)
(set-aux-state-deriv Yr_dot "(@vur/sigmar)*(-Cr1*@alphar+Cr2*@phir-Yr)")

(add-state-variable Yf Yf_dot F)
(set-aux-state-deriv Yf_dot "(@vuf/sigmaf)*(-Cf1*@alphaf+Cf2*@phif-Yf)")
```

The `add-state-variable` commands introduce two state variables, one for each of the two force expressions, that are used to describe the tyre relaxation property. The force equations are defined with the `set-aux-state-deriv` and use the values of the side-slip and camber angles defined above; the `set-aux-state-deriv` complements the `add-equation` command in earlier versions of Autosim, overcoming a difficulty arising previously with added variables in forming the linear model. Notice the minus sign on the `Cr1*@alphar` and `Cf1*@alphaf` terms.

- Introduce the tyre side forces:

```
(add-line-force Yr2 :name "Rear Wheel Lateral Tyre Force"
                    :direction [ya_fry]
                    :point1 rwcp
                    :magnitude Yr)

(add-line-force Yf2 :name "Front Wheel Lateral Tyre Force"
                    :direction @fw_lat
                    :point1 fwcp
                    :magnitude Yf)
```

The direction of the two tyre side forces is in the ground plane and normal to the line of intersection of the ground plane and the wheel plane.

- Incorporate the normal tyre loads:

```
(setsym ground_vector "pos(fwcpn,rwcpn)")
(setsym Gr_vector     "pos(Gr,rwcpn)")
(setsym Gf_vector     "pos(Gf,rwcpn)")


(setsym Zf "-G*(Mf*dot(@Gf_vector,[nx])+Mr*dot(@Gr_vector,[nx]))
            /dot(@ground_vector,[nx])")


(add-line-force Zff :name "Vertical load on front wheel"
                    :direction [nz]
                    :point1 fwcp
                    :magnitude @Zf)
```

The first three lines define three vectors from the rear wheel ground contact point to three points in the nominal configuration. These points are the front wheel ground contact point,

the rear frame centre of gravity and the front frame centre of gravity respectively. The next line calculates the magnitude of the normal force on the front wheel by projecting the three vectors onto the ground plane in the nominal configuration and taking moments about the rear wheel contact point. The `add-line-force` command introduces the normal force into the vehicle equations of motion. Note that only the front force is influential since the system has no heave freedom in body n and therefore the rear force is omitted.

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

```
(unless *linear* (dynamics))
(when *linear*
(add-variables dyvars real st_tq)
(linear)
)
```

If the `*linear*` flag is set to `nil` the full equations of motion are derived. Alternatively if the flag is set to `t` the linearised equations are derived with `st_tq` as the input. `st_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[1]:

```
;;Default values for masses
(set-names    Mf   "Mass of front frame"
              Mr   "Mass of rear frame")
(set-defaults Mf 30.6472 Mr 217.4492)

;;Default values for moments of inertia
(set-names    Ifx  "front frame inertia w.r.t. OX4 about mass centre"
              Ify  "front frame inertia w.r.t. OY4 about mass centre"
              Ifz  "front frame inertia w.r.t. OZ4 about mass centre"
              Irx  "rear frame inertia w.r.t. OX2 about mass centre"
              Iry  "rear frame inertia w.r.t. OY2 about mass centre"
              Irz  "rear frame inertia w.r.t. OZ2 about mass centre"
              Crxz "rear frame inertia product w.r.t. mass centre"
              ifwx "front wheel camber inertia"
              ifwy "front wheel polar moment of inertia"
              irwx "rear wheel camber inertia"
              irwy "rear wheel polar inertia")
(set-defaults Ifx 1.2338 Ify 0 Ifz 0.4420 Irx 31.1838 Iry 0 Irz
              21.0694 Crxz -1.7354 ifwx 0 ifwy 0.7186 irwx 0 irwy 1.0508)

;;Geometric parameters
(set-names Rr    "Rear wheel radius"
           Rf    "Front wheel radius")
(set-defaults aa .9485 bb .4798  h 0.615696 Rf 0.30480 Rr 0.30480
              epsilon 471.5e-3 jj 0.46716 kk 0.853855 ll 0.934658)

;;Tyre parameters
(set-names    sigmar "rear tyre relaxation length"
              sigmaf "front tyre relaxation length"
              Cf1    "front tyre cornering stiffness"
              Cf2    "front tyre camber stiffness"
              Cr1    "rear tyre cornering stiffness"
              Cr2    "rear tyre camber stiffness")
(set-defaults sigmar 0.24384 sigmaf 0.24 Cf1 11174.38 Cf2 938.6124 Cr1
```

[1] As can be seen from the sign of `Crxz`, Autosim uses the opposite sign convention for products of inertia as compared with [1].

```
                15831.8556 Cr2 1326.6232)

;;Other parameters
(set-names    K "steering damping coefficient")
(set-defaults K 6.78 vu 6.1538)
(unless *linear* (set-defaults iprint 1 stopt 50 step 0.01 "rq(rf)"
                               0.005 epsdi 1e-7 st_tq 0))
```

The last line sets some default values relevant only to the nonlinear model.

- Write up files:

```
(unless *linear*
(write-to-file write-sim "bk_71_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_71_ls.m")
)
```

The FORTRAN file is written to `bk_71_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 a useful debugging aid. If the linearised model is asked for, then the MATLAB^{TM} file `bk_71_ls.m` is written to disc.

4 Simulations and Results

The nonlinear code is used to generate the nonlinear equations of motion in the form of a FORTRAN code. The Fortran file is compiled and executed to generate time histories of the various dynamic variables (positions, velocities, accelerations, forces and so on). A typical plot of a time history is shown in Figure 3. In this case the forward speed is held constant at 20 ft/s (6.1538 m/s) and there is an initial non-zero roll angle of 0.005 *rad*. The z rotational speed of `ff` relative to `rf` undergoes an initial transient and then settles to zero, because the system is stable in its straight running configuration. It is easy to see that this transient has two different frequency components, one being fast and the other slow. The slow mode corresponds to the weave mode of the motorcycle and has a frequency of about 2.24 $\frac{rad}{s}$. The high frequency mode is the so called wobble mode which has a frequency of 58.18 $\frac{rad}{s}$.

The Autosim code is used to generate the linearised equations of motion about straight running equilibrium conditions. In this equilibrium state the motorcycle is moving with constant forward speed and with zero roll, yaw and steer angles. The forward speed is varied in steps and the eigenvalues of the system for each equilibrium speed are calculated and plotted in Figure 4. The first part of Figure 4 is a plot of the real parts of the eigenvalues against forward speed - this agrees with Figure 5 in [1]. The second part of Figure 4 is the root-locus plot with forward speed the varied parameter. A detailed analysis of the weave, wobble and capsize modes is given in Appendix A.

5 Conclusions

The aim of this work is to demonstrate that the results presented in [1] 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 of the nonlinear system. 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 also be used to generate linearised state-space models in symbolic form. The linearised models can be imported into MATLAB^{TM} for evaluation. A

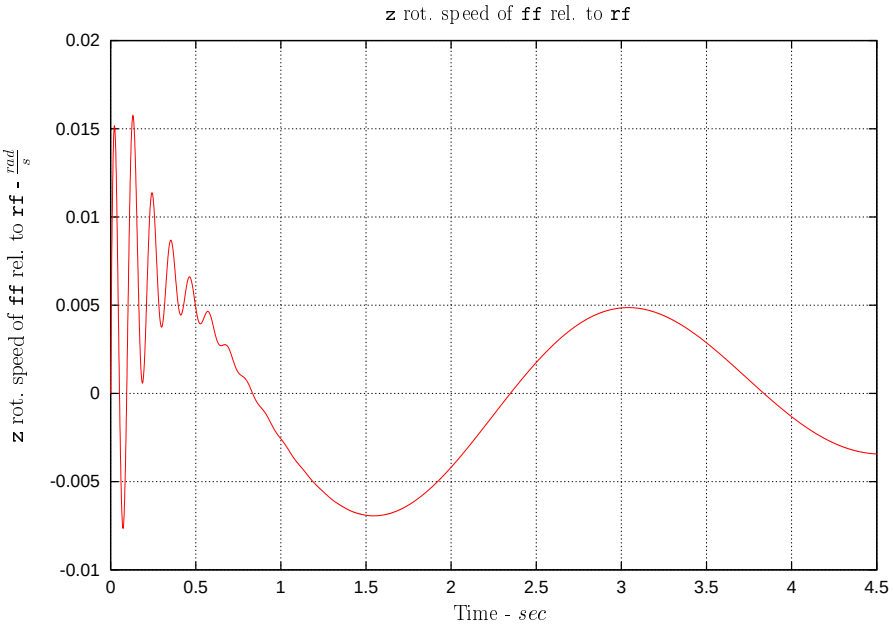

Figure 3: z rot. speed of `ff` rel. to `rf`

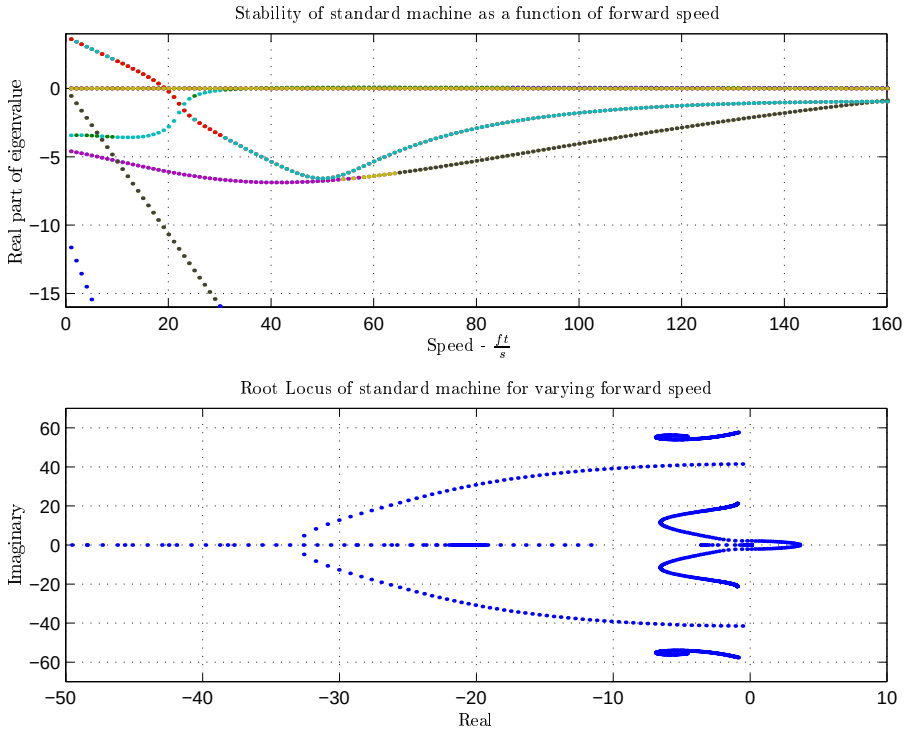

Figure 4: Stability and Root Locus plots

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_71_ns.f` and the linearised equations of motion are stored in the MATLAB^{TM} file `bk_71_ls.m`. In order to construct the root-loci in Figure 4, two lines have to be removed from this code: the first line, which is a `clear` statement, and the line containing the `VU=80/13` statement. The modified version of the MATLAB^{TM} file is stored in `bk_71_ls.m` and the diagrams in Figure 4 may be generated using this file and the hand-written plotting codes `bk_71_results.m` and `bk_71_rootlocus.m`.

A The weave, wobble and capsize modes

A.1 Body capsize

When the motorcycle has zero forward velocity and the steering freedom is removed, it behaves like an inverted pendulum that is about to fall over. For small camber angles, one can balance inertial and gravitational moments to obtain:

$$(\sum I_i)\ddot{\phi} = g\phi(\sum m_i l_i) \tag{1}$$

in which ϕ is the camber angle and $\sum I_i$ is the total moment of inertia of the vehicle about the line joining the two wheel ground contact points in the nominal configuration as shown in Figure 5. The sum $g\phi \sum m_i l_i$ is the total torque generated by the gravitational forces. It is easy to see that the second order differential equation (1) has two real poles associated with it:

$$\alpha = \pm\sqrt{\frac{g(\sum m_i l_i)}{\sum I_i}}. \tag{2}$$

Figure 6 shows the right-half plane part of the root-locus corresponding to the "Sharp 1971"

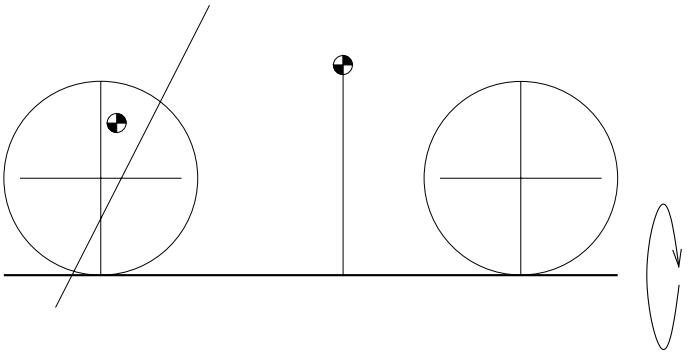

Figure 5: Motorcycle as an inverted pendulum.

model for low values of forward speed. As the machine speed increases, these poles meet, coalesce and become the complex pole pair associated with the weave motion of the machine. The pole with the largest initial value of about 4.27 corresponds to the positive solution of equation (2) which can be solved to yield 3.46 for the "Sharp 1971" model parameters. The reason for the discrepancy between these two values (4.27 and 3.46) can be traced to the vehicle's steering action. Indeed, when steering is inhibited, the "Sharp 1971" model has a positive real pole located at 3.494 rather than at 4.27 as shown in Figure (6). The reason for this increased capsize growth rate under steering is interesting. Suppose the machine begins to fall to the rider's right. In this case the motorcycle's steering geometry causes the machine to steer right thereby moving the front wheel ground contact point towards the rider's left. Consequently, the ground contact line that joins

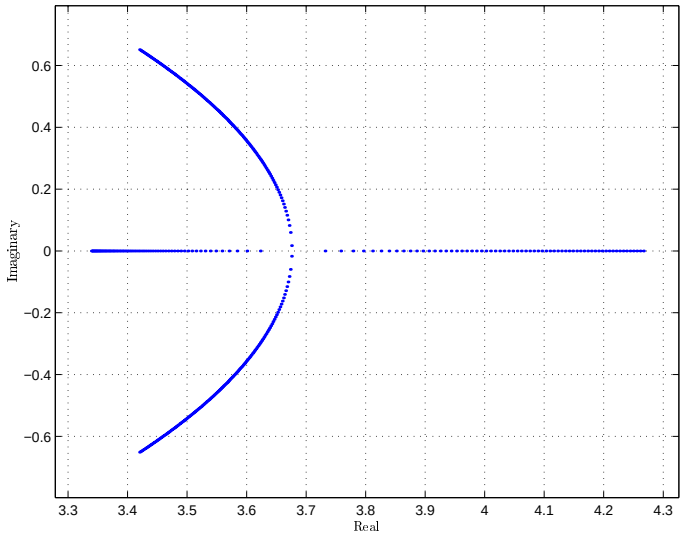

Figure 6: Capsize portion of the root-locus plot.

the front and rear wheel ground contact points rotates to the rider's left. This means that the gravitational torque produced by the $g\phi \sum m_i l_i$ terms increase and so the machine capsizes more quickly.

A.2 Steering capsize

Consider the simplified situation in which the rear frame is fixed (in body n) and the *Front Frame* is free to steer (ground contact effects being ignored). This situation is shown in Figure 7. As before, balancing the gravitational and inertial torques gives:

$$I_{fz}\ddot{\delta} = (M_f g e sin\epsilon)\delta \tag{3}$$

in which all the symbols have their usual meaning. This second order system has the real poles:

$$\zeta = \pm\sqrt{\frac{M_f g e sin\epsilon}{I_{fz}}} \tag{4}$$

associated with it. Substituting the "Sharp 1971" model parameters into (4) gives the positive real pole a value of 2.742. As we will explain, this root is related to the smaller of the real roots in Figure 6. The initial agreement is not very good, because equation (4) predicts a growth rate of $\sim e^{2.742t}$, while the "Sharp 1971" model predicts a rate of about $\sim e^{3.33t}$. It turns out that this discrepancy is due to a combination of the steering damping, which is neglected in equation (4), and the front wheel tyre forces. In order to show this, one can multiply the steering damping factor and the front wheel tyre forces terms in the "Sharp 1971" Autosim code by a parameter λ, and then consider reducing the value of λ from $1 \rightarrow 0$. It turns out that the real pole corresponding to the steering capsize mode varies from $3.33 \rightarrow 2.69$. This latter value is much closer to the figure of 2.74 predicted by equation (4). If in addition the rolling motion of the rear frame is inhibited by setting `Irx` to some large value, the pole predicted by the Autosim code becomes even closer to that predicted by equation (4); we obtained agreement to three significant figures.

A.3 Wobble frequency

As we will show, the wobble frequency for small forward speeds can be calculated by considering the front frame and the front wheel tyre side force. The situation of interest is shown in Figure 8. Balancing the inertial torque with that generated by the side-slip tyre force gives:

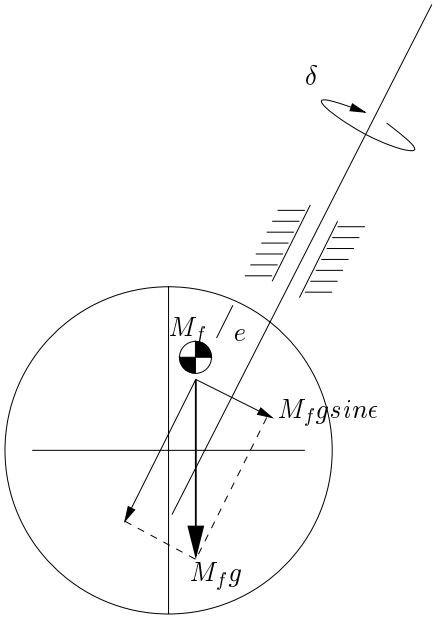

Figure 7: Steering mechanism as it relates to the steering capsize mode.

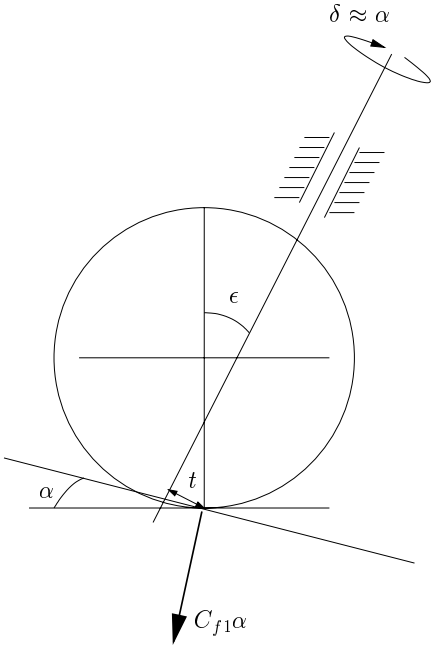

Figure 8: The steering system and the tyre forces associated with the wobble mode.

$$I_{fz}\ddot{\delta} = -tC_{f1}\alpha \tag{5}$$

and since $\alpha = \delta cos\epsilon$

$$I_{fz}\ddot{\delta} = -tC_{f1}\delta cos\epsilon. \tag{6}$$

This gives a predicted wobble frequency of:

$$\omega_{wobble} = \sqrt{\frac{tC_{f1}cos\epsilon}{I_{fz}}}. \tag{7}$$

The "Sharp 1971" model predicts a low-speed wobble frequency of 57.7, which is good agreement with the value of 51.1 computed from equation (7).

References

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

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