ANIMATION OF THE "SL2001" 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

The task of an animator is to generate a visual representation of the dynamic operation of complex multi-body systems such as road vehicles. This is done by generating a wire-frame image of the constituent parts of the mechanism and then "driving" it with the output of a simulation programme. The Autosim animator can be downloaded from http://www.trucksim.com/animator/index.html and is the one we will make use of in this report. The animator must be supplied with two files. The first is a parsfile (PAR) that contains keyword-based text data that defines all the mechanism parts together with other information such as programme settings. The second file is an ERD file generated by an Autosim simulation programme. It contains time history data associated with each of the vehicle bodies. The time histories comprise six degrees-of-freedom data associated with each body in global co-ordinates. The time histories in the ERD file are used to drive the various parts of the motorcycle. The aim of this report is to supplement, in a motorcycle specific context, the animator description that can be found in Chapter 6 of the CarsimEd manual. The manual can be downloaded from http://www.trucksim.com/carsimed/index.html. The vehicle used to demonstrate the animator is the "SL2001" motorcycle described in [3].

1 Introduction

Various motorcycle models have already been developed in the simulation code Autosim [4]. The simplest of these contains four bodies [1], while the most complex is the "SL2001" model [3] that contains seven bodies.

As the complexity of these motorcycle models increase, it becomes more and more difficult to visualise the overall dynamic behaviour of the vehicle. The purpose of the animator is to assist with the visualisation of the machine's dynamic behaviour via a properly scaled wire-frame image that is driven by simulator data. It is hoped this animation facility will prove to be a useful tool that will assist with the analysis of some of the more complex behaviours of the machine. The animation is essentially a sequence of images that are updated several time per second; this process is reminiscent of the images that can be seen with a video camera.

2 Description of the model

The motorcycle under study is represented diagrammatically in Figure 1. The body structure diagram of the vehicle that was used for the purpose of writing the Autosim code is taken from [3] and is shown in Figure 2. As is usual in our work, the bodies are arranged in a parent-child relationship.

3 Program codes

As shown in Figure 3, two files are required to run the animation: the parsfile and the simulation file. The first file is a keyword-based text file that contains the definitions of all the parts of the model, their shape information and other information such as program settings. Typically, this file has the extension PAR. The simulation file, which is an ERD file generated by Autosim, contains

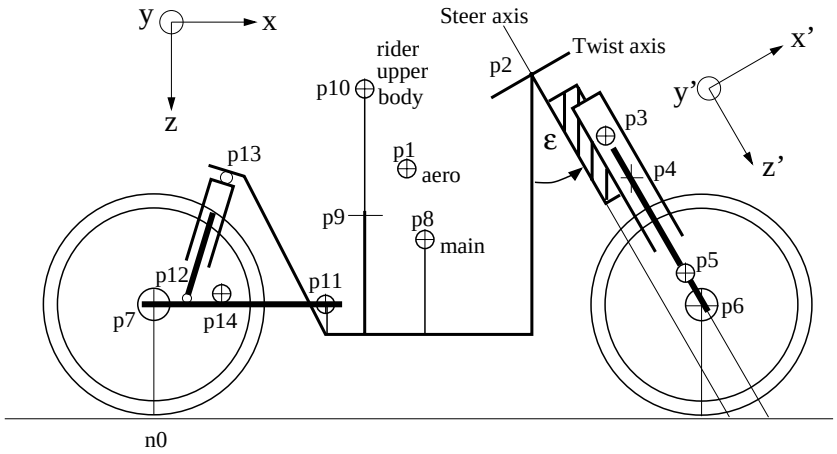

Figure 1: Diagrammatic representation of the motorcycle.

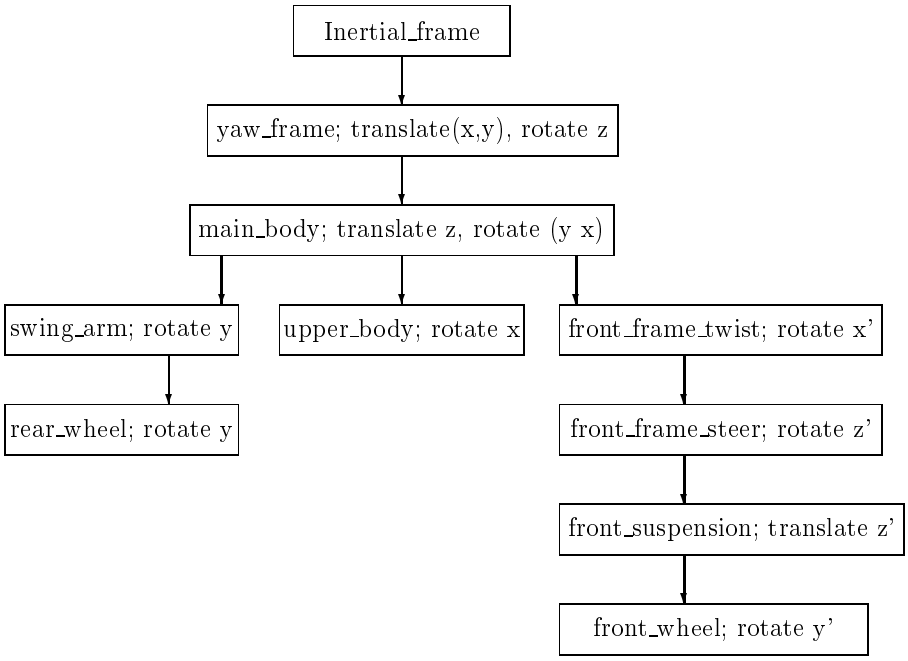

Figure 2: Body Structure Diagram of the "SL2001" motorcycle.

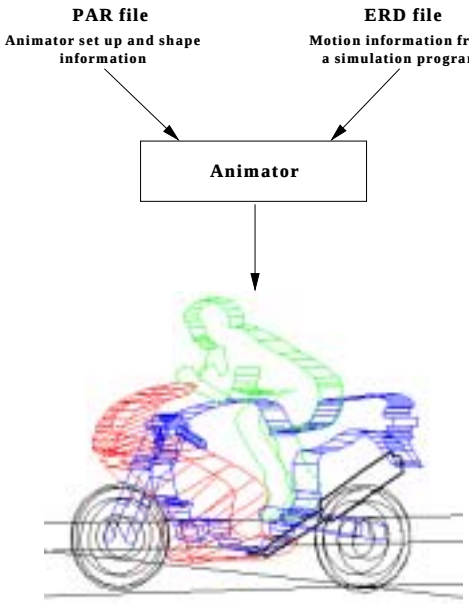

Figure 3: Animator input files.

the simulation responses in global coordinates. In general, the motion of each body requires three translational data sequences and three rotational data sequences. These responses are used by the animator to drive the various parts of the motorcycle as defined in the parsfile.

3.1 Parsfile

The animator creates images that are based on a set of visible objects that include a grid and wire-frame shapes that are defined via a sequence of connected lines. Some of the wire-frame objects are organized into groups that move together. A group of points and objects that maintain a fixed relationship to each other (i.e. that constitute a rigid body) is called a reference frame. Although a reference frame might move and rotate, the spatial relationships between the objects in the reference frame do not change relative to each other. In the animator all motions are associated with reference frames and their movement is defined by up to six variables from the ERD file (three translational freedoms and three rotations (Euler angles) in global coordinates). The reference frames that are used to define the motorcycle model are shown in Figure 4. Each reference frame has associated with it a group of shapes that are used to build up a detailed visual representation of the motorcycle. As the inputs to the animator are processed, each shape is moved along with its particular reference frame. As the animator is reading input data, the active reference frame and its associated shapes are moved with that frame.

A shape is a set of points connected by straight lines and each point is defined by a set of three coordinates (X-Y-Z). The animator starts with the first point and draws connecting lines to each of the following points in the list. All the coordinates are assumed to be in a local coordinate system that is associated with the active reference frame. Figure 5 shows the shapes associated with each reference frame.

The axis orientation used by the animator follows the ISO 8855 standard instead of SAE J670e used by Autosim. ISO 8855 has X pointing forwards, Z pointing upwards and Y pointing towards the left-hand side of the vehicle. In contrast, SAE J670e has Z pointing downwards, X pointing forwards and Y pointing towards the right-hand ride on the vehicle. All coordinates in the output file are in SI units if units have been set to "SI" in the Autosim LISP code. Both variable and static coordinates can be converted using scale factors. All animator Euler angles must be expressed in degrees and so scale factors are used to convert the Euler angles generated by Autosim to degrees, since the variables in the ERD file are in radians. If, however, `(si)` is replaced by `(mks)` in the Autosim code angles will be in degrees.

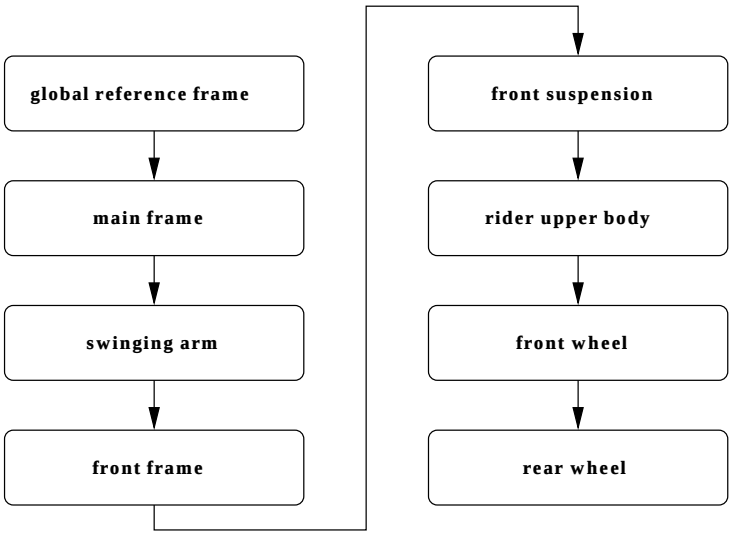

Figure 4: Reference Frames of the motorcycle.

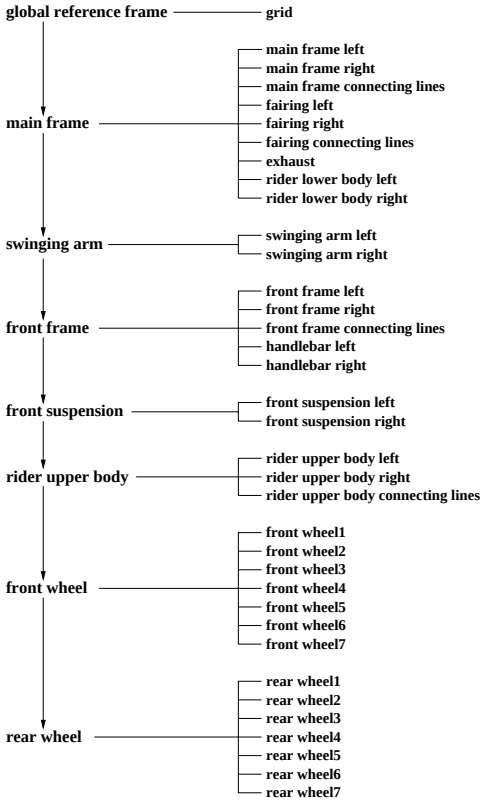

Figure 5: Groups of shapes.

The parsfile commands are described next and we recommend that the reader studies these in conjunction with the motorcycle parsfile `bk_sl2001.par`:

- Add the 2D ground-plane grid:

Keyword	Value	Description
`add_grid`	<none>	tells animator to draw a reference grid
`set_interval_x` `set_interval_y`	numbers	spacing used for drawing the grid lines
`set_color`	color name	color used for the grid lines

`set_min_x`
`set_max_x`
`set_min_y`
`set_max_y`
	numbers	size of the grid in the X and Y directions

Table 1: Keywords for describing the grid

A grid fixed in the global reference frame is drawn. Table 1 lists the keywords for describing the grid.

- Specify the camera settings:

  The camera point determines the location of the observer and the look point determines the point that the camera is aimed at. Both of these points are shown in Figure 6. At each time

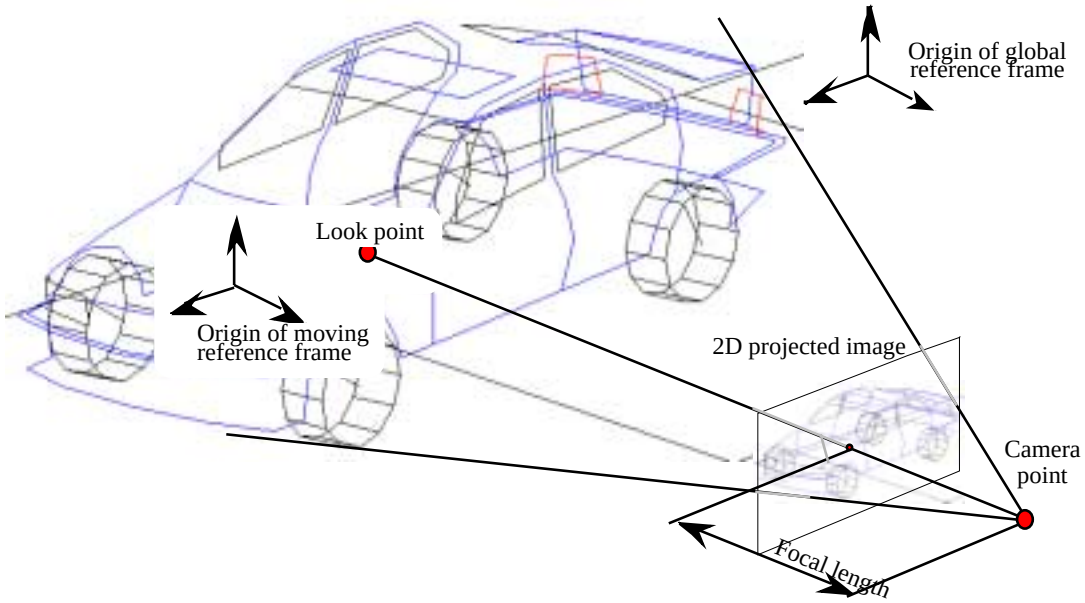

Figure 6: Geometry of the camera point and the look point.

instant, the animator generates a 2D image based on the relationships between the location and orientation of the simulated vehicle and the camera and look points. Table 2 lists the keywords that are used to specify the camera settings.

- All the reference frames with all the shapes associated with them as shown in Figure 5 are added by making use of the keywords in Table 3.

  The keyword `add_reference_frame` has three effects:

  1. It starts the scope of a new reference frame.

Keyword	Value	Description
set_camera_reference_frame	name of reference frame	reference frame in which the camera is situated
set_camera_x set_camera_y set_camera_z	numbers	coordinates of the camera location in its reference frame
set_lookpoint_reference_frame	name of reference frame	reference frame in which the look point is situated
set_lookpoint_x set_lookpoint_y set_lookpoint_z	numbers	coordinates of the look point in its reference frame
set_focal_length	number	focal length of camera (distance from point of viewer to 2D image on screen)
set_use_cpu_clock	on or off	option to slow animation down to real time by using the clock
set_superimpose	on or off	option to superimpose all images–don't erase between animation frames

Table 2: Keywords for the animator camera settings

Keyword	Value	Description
add_reference_frame	name of new reference frame	gives name to new reference frame and starts its scope
set_x_name set_y_name set_z_name	names of variables in ERD file	specifies the variables to be read from the ERD file and associated with X,Y,Z coordinates of the reference frame
set_pitch_name set_roll_name set_yaw_name	names of variables in ERD file	specifies the variables to be read from the ERD file and associated with Euler angles
set_scale_var_x set_scale_var_y set_scale_var_z set_scale_var_roll set_scale_var_pitch set_scale_var_yaw	numbers	scale factors for data read from the ERD file
set_offset_var_x set_offset_var_y set_offset_var_z set_offset_var_roll set_offset_var_pitch set_offset_var_yaw	numbers	offsets added to coordinates and Euler angles
set_euler_angles	yaw_pitch_roll or yaw_roll_pitch	sequence of rotation by Euler angles used to define orientation of the reference frame

Table 3: Keywords associated with reference frames

2. It ends the scope of the previous one.
3. It assigns a name to the new frame that can be used with the `set_camera_reference_frame` and `set_lookpoint_reference_frame` keywords. Each reference frame must have a unique name.

The scope of the reference frame begins when the keyword `add_reference_frame` is encountered and continues until this keyword is used again to start the scope of a different reference frame. All of the keywords shown in Table 3 are repeated several times in the parsile. Each time the value associated with the keyword is applied, the current reference frame is affected.

The position of a reference frame is defined by six variables: the three coordinates (X, Y, and Z), and three Euler angles. The animator reads the six variables from the output files generated by the simulation. The six keywords used to specify the ERD file short names determine how the three coordinates and the three Euler angles are defined. After reading the six variables, each coordinate and Euler angle is calculated via a relationship of the form:

$$\text{coordinate} = C_o + C^*SF_c$$

$$\text{angle} = A_o + A^*SF_a$$

where C and A are the translation and angle variables obtained from the ERD file. The constants C_o and A_o are offsets while SF_a and SF_c are scale factors (gains). The offsets and scale factors are specified by the keywords shown in Table 3. The scale factors are often used to convert angles from radians to degrees. The keyword `set_euler_angles` is used to specify the type of transformation used. There are two options: `yaw_roll_pitch` is used for rolling-wheel reference frames and `yaw_pitch_roll` is used for all the other vehicle reference frames.

Within the scope of a particular moving frame, the associated parts (shapes) are specified. A part is a set of points connected by straight lines. Each point is defined by a set of three coordinates (X-Y-Z). The animator starts with the first point and draws connecting lines to each of the following points in the list. All the coordinates are assumed to be in a local coordinate system associated with the active reference frame. The keyword `add_part` has the effect of starting the scope of a new object. It also has the effect of ending the scope of the previous object. However, it does not affect the scope of the current moving reference frame. All the keywords relevant to shapes are defined in Table 4. The list of coordinates

Keyword	**Value**	**Description**
`add_part`	name of part	starts scope for new part
`set_color`	color name	color used for lines drawn to connect the points in this part
`set_line_width`	integer	sets thickness of lines drawn for this part
`set_coordinates` `end_coordinates`	list of coordinates: 3 numbers per line	coordinates of this points making up the shape

`set_scale_x`
`set_scale_y`
`set_scale_z`
	numbers	scale factors applied to all coordinates in the part

`set_offset_x`
`set_offset_y`
`set_offset_z`
	numbers	offsets added to all points in the part

Table 4: Keywords for describing parts

begins with a line containing the keyword `set_coordinates`. Each following line contains an X, Y, and Z coordinate, separated by white space, until the list ends with a line containing the keyword `end_coordinates`. The listed coordinates for the part are transformed by the equations:

$$x_{new} = x_o + s_x\ x$$

$$y_{new} = y_o + s_y\ y$$
$$z_{new} = z_o + s_z\ z$$

where x_o, y_o, and z_o are offsets and s_x, s_y, and s_z are scale factors specified with the keywords `set_offset_x`, `set_offset_y`, `set_offset_z`, `set_scale_x`, `set_scale_y` and `set_scale_z`.

3.2 Example reference frame

The purpose of this section is to show how each motorcycle component is inserted one-by-one into the animation. We use the front wheel to explain the procedure. The remaining components are added in a similar manner using the ideas of reference frames, shapes and global coordinates.

To begin, it should be noted that each of the wheels is made up of a number of parts that are assembled using the `add_part` keyword. As indicated in Figure 5, the wheels are made up of seven parallel circles of different diameters. The diameters are chosen to correctly represent the tyre cross-sectional profiling. In the parsfile given here, the first part is the central circle. Copies of this circle are then scaled and shifted to generate the other six. These seven parts can then be grouped together under the front wheel reference frame to form the front wheel. The origin of this reference frame is at the centre of the detached wheel as shown in Figure 7. The shapes

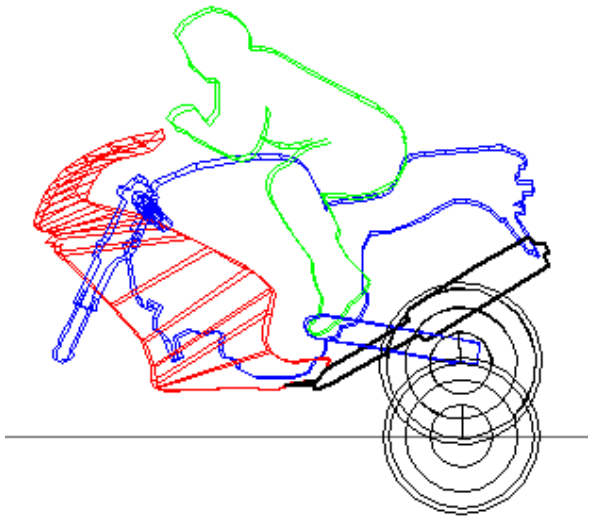

Figure 7: Front wheel example.

that make up the front wheel are designed so that the centre of the wheel is at the centre of the reference frame. The reason for this is that all the Euler angles of the reference frame are defined as rotations about axes through the origin of the current reference frame. It is essential that this setup is made precisely compatible with the Autosim code that will be used to generate the data that drives the animation. For example, this means that if the front wheel is to be designed to rest on the ground plane in the nominal configuration, in the same way that the rear wheel does in Figure 7, then the pitch rotation will rotate the wheel around the ground contact point and not the wheel hub. Any yawing rotation must occur around the wheel centre. Once all the reference frame constituent parts have been designed with the same considerations in mind, appropriate output variables in the ERD file are linked to the reference frame in order that it is driven properly. In our case all the reference frames are constructed around the Autosim body centres, because this makes them easy to link to the driving variables that are stored in the ERD file. Obviously, the final aim is to create an image of the motorcycle that has all its components correctly dimensioned and correctly placed in relation to each other through the motion being studied. In the case of the front wheel, the output variables are `fw_xo`, `fw_yo` and `fw_zo` for the translational movements, while `fw_rol`, `fw_pit` and `fw_yaw` are used for the roll, pitch and yaw rotations respectively. These variables are calculated by the simulation programme that is derived from the Autosim lisp code.

The lisp commands that are used to find the animator driving variables in global coordinates are discussed next.

3.3 Lisp code

The following lisp instructions must be added to `bk_sl2001.lsp` code that describes the "SL2001" model. These commands are used to calculate the output variables needed by the animator:

- Main frame:

```
(setsym main_x "dot(pos(yaw_fr0,n0),[nx])")
(setsym main_y "dot(pos(yaw_fr0,n0),[ny])")
(setsym main_z "-dot(pos(main0,n0),[nz])")

(setsym main_yaw   "rq(yaw_fr)")
(setsym main_pit   "-rq(main,1)")
(setsym main_rol   "-rq(main,2)")

(add-out "@main_x" "main_x")
(add-out "@main_y" "main_y")
(add-out "@main_z" "main_z")

(add-out "@main_yaw"  "main_yaw")
(add-out "@main_pit"  "main_pit")
(add-out "@main_rol"  "main_rol")
```

  The `setsym` commands define the global variables required by the animator and the `add-out` commands add them to the output variables that are stored in the ERD file. The first three lines take the projections of the position vector between the origin of the `yaw_frame` and the origin of the inertial frame in the three standard directions. These projections are used to calculate the three translational coordinates in the global reference frame `n`. The next three lines define the global angles of rotation of the main frame. The z-component of position and two of the angles of rotation have minus signs, because the orientations of the axes in the animator and Autosim follow different standards. Note that some of the variables are already calculated by Autosim as standard outputs. However, these variables have been redefined for completeness. The remaining bodies are treated in much the same way.

- Swinging arm:

```
(setsym swg_arm_x "dot(pos(swg_arm0,n0),[nx])")
(setsym swg_arm_y "dot(pos(swg_arm0,n0),[ny])")
(setsym swg_arm_z "-dot(pos(swg_arm0,n0),[nz])")

(setsym swg_arm_yaw   "euler(swg_arm,1,n)")
(setsym swg_arm_pitch "-euler(swg_arm,2,n)")
(setsym swg_arm_roll  "-euler(swg_arm,3,n)")

(add-out "@swg_arm_x" "sa_x")
(add-out "@swg_arm_y" "sa_y")
(add-out "@swg_arm_z" "sa_z")

(add-out "@swg_arm_roll"  "sa_rol")
(add-out "@swg_arm_pitch" "sa_pit")
(add-out "@swg_arm_yaw"   "sa_yaw")
```

- Rider upper body:

```
(setsym ubr_x "dot(pos(ubr0,n0),[nx])")
```

```
(setsym ubr_y "dot(pos(ubr0,n0),[ny])")
(setsym ubr_z "-dot(pos(ubr0,n0),[nz])")

(setsym ubr_rol "-(rq(main,2)+rq(ubr))")

(add-out "@ubr_x" "ubr_x")
(add-out "@ubr_y" "ubr_y")
(add-out "@ubr_z" "ubr_z")

(add-out "@ubr_rol" "ubr_rol")
```

In the case of the rider's upper body it is only necessary to calculate the roll angle, because the pitch and yaw angles for this body are the same as those used for the main frame. We observe that the roll angle is the sum of the roll angle of the main frame and the roll angle of the rider's upper body with respect to the main frame. It is possible to simply sum up these angles, because the series of Euler angles used is `yaw_pitch_roll` and roll is the last rotation.

- Front frame:

```
(setsym ff_x "dot(pos(ff_str0,n0),[nx])")
(setsym ff_y "dot(pos(ff_str0,n0),[ny])")
(setsym ff_z "-dot(pos(ff_str0,n0),[nz])")

(setsym ff_rol "-euler(ff_str,3,n)")
(setsym ff_pit "-(euler(ff_str,2,n)-epsilon)")
(setsym ff_yaw "euler(ff_str,1,n)")

(add-out "@ff_x" "ff_x")
(add-out "@ff_y" "ff_y")
(add-out "@ff_z" "ff_z")

(add-out "@ff_rol" "ff_rol")
(add-out "@ff_pit" "ff_pit")
(add-out "@ff_yaw" "ff_yaw")
```

The front frame variables are calculated as before. Note that `epsilon` is subtracted from the pitch angle in line 5. This is because front frame is initially inclined at an angle of epsilon (ϵ) relative to the inertial reference frame. This adjustment could also be made in the parsfile, but this is a matter of taste.

- Rear wheel:

```
(setsym rw_xo "dot(pos(rw0,n0),[nx])")
(setsym rw_yo "dot(pos(rw0,n0),[ny])")
(setsym rw_zo "-dot(pos(rw0,n0),[nz])")

(setsym rwlong   "cross([swg_army],[nz])")
(setsym rw_pit_i "angle(@rwlong,[swg_armx])")
(setsym rw_rol   "-angle([swg_army],dplane([swg_army],[nz]))")
(setsym rw_pit   "-(-@rw_pit_i+rq(rw))")

(add-out "@rw_xo" "rw_xo")
(add-out "@rw_yo" "rw_yo")
(add-out "@rw_zo" "rw_zo")

(add-out "@rw_rol" "rw_rol")
(add-out "@rw_pit" "rw_pit")
```

It will be noted that the scheme here is different from the previous one used to calculate the Euler angles, since these now involve angles that are outside the range $\pm\pi$. In particular, the pitch angle of the wheel undergoes angular wind-up (it keeps rotating in one direction and consequently the pitch angle continues to grow). The series of rotations for the wheels is `yaw_roll_pitch`. It can be seen that roll angle is calculated from first principles. The quantity `-rw_pit_i` is the initial pitch angle of the wheel produced by the rotation of the swinging arm. The pitch angle of the wheel is consequently the sum of this angle and `rq(rw)`. The negative sign is required because Autosim and the animator use different standards. The yaw angle of the wheel need not be calculated because it is the same as that of the main frame.

- Front wheel:

```
(setsym fw_xo "dot(pos(fw0,n0),[nx])")
(setsym fw_yo "dot(pos(fw0,n0),[ny])")
(setsym fw_zo "-dot(pos(fw0,n0),[nz])")

(setsym fwlong   "cross([ff_susy],[nz])")
(setsym fw_pit_i "angle(@fwlong,[ff_susx])")
(setsym fw_rol   "-angle([ff_susy],dplane([ff_susy],[nz]))")
(setsym fw_pit   "-(-@fw_pit_i+rq(fw))")

(add-out "@fw_xo" "fw_xo")
(add-out "@fw_yo" "fw_yo")
(add-out "@fw_zo" "fw_zo")

(add-out "@fw_rol" "fw_rol")
(add-out "@fw_pit" "fw_pit")
```

3.4 Running the animator

To run the animator the following items are needed:

- The animator executable file `animator.exe` (this can be downloaded from http://www.trucksim.com/animator/index.html).
- The parsile `bk_sl2001.par` (this can be downloaded from our website http://www.ee.ic.ac.uk/control/motorcycles).
- The simulation files `example.erd` and `example.bin`. These files can be downloaded from our website, or they can be generated by patching the Lisp code `bk_sl2001.lsp`. The instructions that generate the required data in global coordinates should be placed just before the `(finish)` command. A new data file must be generated by loading the modified Autosim code and running the associated simulation file.

The animation can be run as follows:

- Start the animator by running the executable file `animator.exe`.
- Go to the *file* roll-down menu in the animator window, click on *Open Parsfile* and select the parsfile `bk_sl2001.par`.
- Find and select the ERD file (`example.erd`) in the same window.
- After all the files are loaded the animation can be started by going to the *Animation* roll-down menu and clicking on *Start From Beginning*.

References

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

[2] S. Evangelou and D.J.N. Limebeer, "Lisp programming of the "Sharp 1994" motorcycle model", 2000, *http://www.ee.ic.ac.uk/control/motorcycles*.

[3] R.S. Sharp and D.J.N. Limebeer, "A motorcycle model for stability and control analysis", Multibody System Dynamics, 2000.

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