Simulation of Engineering Systems 3

Turning control System for a Wheeled Rover Jagdeep Gill 2083986G

Introduction

A rover is robotic vehicle that is driven by motorised wheels which are designed to move across several different types of terrain. Rovers are commonly used machinery. One are in which it is used is in space exploration such as the Curiosity Rover the Mars Exploration Rover. A rover is also used within the bomb disposal field and for surveillance. For this given assignment, the movement of the river is determined by the speed at which the left and right wheel are moving at. In the case of turning, one of the wheels would turn slower than the other. The speeds of the left and right motor are dependent on the input voltage supplied by the system of the rover.

The purpose of this assignment was to create a simulation that would represent the turning control system of the rover’s wheels. This was achieved by creating a Matlab model that would use numerical integration and then be validated with the use of Simulink.

Part 1 – Creation of the Simulation Script and Simulation Model Script

Methodology

9 equations that defined the heading control system of the rover were taken from the ‘Problem Specification’ segment of the assignment.

These equations were:



This equation represents the control system which uses an error difference of angles Δψ, this being the difference in the yaw angle (ψref) and the rover’s actual yaw angle (ψ) to control the system. Gc and K1 are gains that are used to establish the performance of the control system. These are used to determine the voltage difference (ΔV) which will allow for a suitable heading to generated for the rover.



# Equations (2) and (3) shown above represent the two DC motors, one being the left and the other the right, which are used to drive the rover. In the equations, i is the current in the motor, L is the inductance, R is the resistance, Kt is the torque constant, Ke is the bake mf constant, bs is the damping coefficient and ω is the angular velocity of the wheel. Vin is the input of the motor which is described as VD ± ΔV differing when the left or right of the motor is being used. This circuit is shown in Figure 1.

# Figure 1. Rover Motor and Wheel Block Diagram

# 

The wheel can be regarded as the load on the motor’s shaft. This results in it having its own dynamics. This being represented in equation (4). ωw is the speed of rotation of the wheel and Jw is the moment of inertia of the wheel.

# 

# The current generates the force in each of the wheels. These forces are then processed into the next equations (7 and 8) which allow for the sway and yaw mechanics to be generated to give the rover the correct movement. This is show on Figure 1. Rw in the two equations represents the wheels radius.

# 

# Equations (7) and (8) represent the sway and yaw dynamics of the rover. RM in the equation is the moment arm for the motor relative to the centre of gravity of the rover. VT is the forward velocity of the rover.

# (9)

This final equation shows that the yaw rate, r, is equal to the rate of change of the yaw angle.

After collecting all of these equations, the next step was to create a space state model that would be used in the Matlab code to simulate the system. This procedure is carried out in four steps:

1. Reduced Form

The first step in creating the state space model is to put the equations from above into the reduced forms.

# States

# The second step in the state space model is to note down all the states in the system. (Note: subscript L denotes the left side and subscript R denotes the right side)

# Differentiation

# The third step in the state space model is to take the derivatives of the states.

# Substitution

# The final step in the state space model is to substitute the states and differentiated states into the reduced form equations.

# Once the state space model for the rover was derived, a Matlab equation was created. A set structure was in place for the Matlab equation simulation to follow. This consists of three stages, the Initial Segment which contains the initialisation of any variables and parameters. The second stage is the Dynamic Segment which contains the storage of data, differentiation and integration of the model. The final stage is the Terminal Segment which plots and displays the data. The Matlab code used in this simulation is shown in Appendix A and the states script is shown in Appendix B.

# For the numerical integration used within the simulation, it was decided that Runge-Kutta Fourth Order would be used in the Dynamic Segment. Runge-Kutta Fourth Order was chosen as it was found to give more accurate results at a smaller step size over the Euler integration method. The reason Runge-Kutta is more accurate is due to it taking the approximates of the first 5 terms of the Taylor’s Series compared to the Euler integration which only takes the first two terms. As accuracy of the turning controls of this system was of great significance, it seemed appropriate to choose this method.

# Results

# After running the simulation, several graphs were produced for each of the states against time using subplots. These are shown in Figure 2.

# 

# Figure 2. Simulation Plots of States against Time

# The results plotted by the simulation did not seem to be as expected with the initial values given. The simulation was unstable which was part due to there not being an integral term introduced into the Matlab model.

# Part 2 – Validation of Simulation response using Simulink

# Through the use of block diagrams in Simulink, the simulation from Part 1 was validated to measure its accuracy. The Simulink block diagram that was created is shown below in Figure 3.

# 

# Figure 3. Simulink Block Diagram for Turning Control System

# From Figure 3, it can be seen that there are 2 subsystems – one being the Turning Control and the other being the Rover & Wheel Motors. The Turning control subsystem contains Vin shown in Figure 4. There is another subsystem within Vin, that being ΔV, shown in Figure 5. The Rover & Wheel Motors contain 3 subsystems (Figure 6). One for each of the Left and Right Motor, shown in Figure 7 and Figure 8, and on for the Sway and Yaw dynamics, shown in Figure 9.

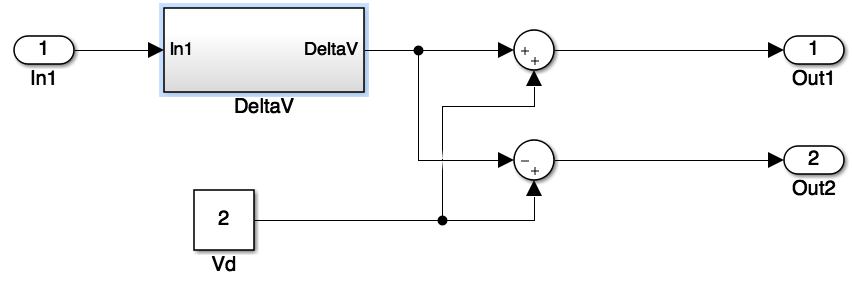


Figure 4. Turning Control Subsystem

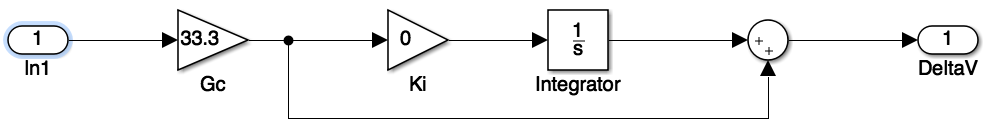


Figure 5. ΔV Subsystem

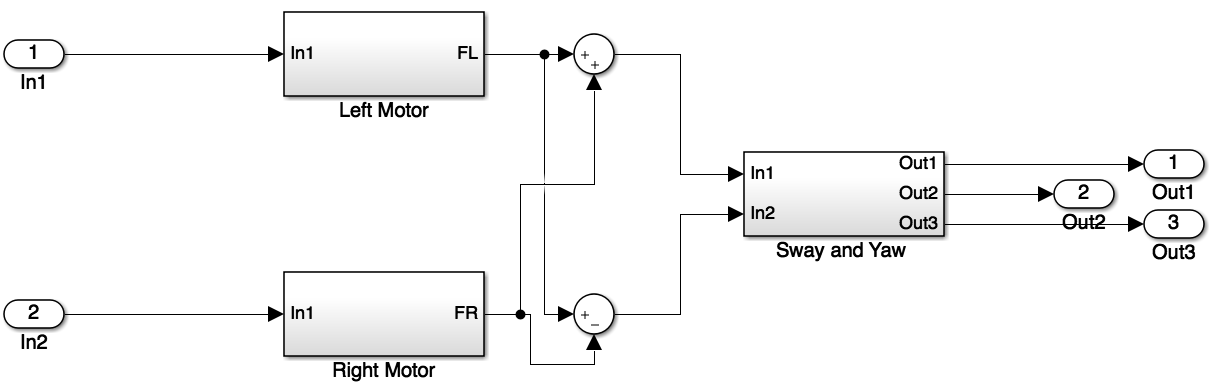


Figure 6. Rover & Wheel Motors Subsystem

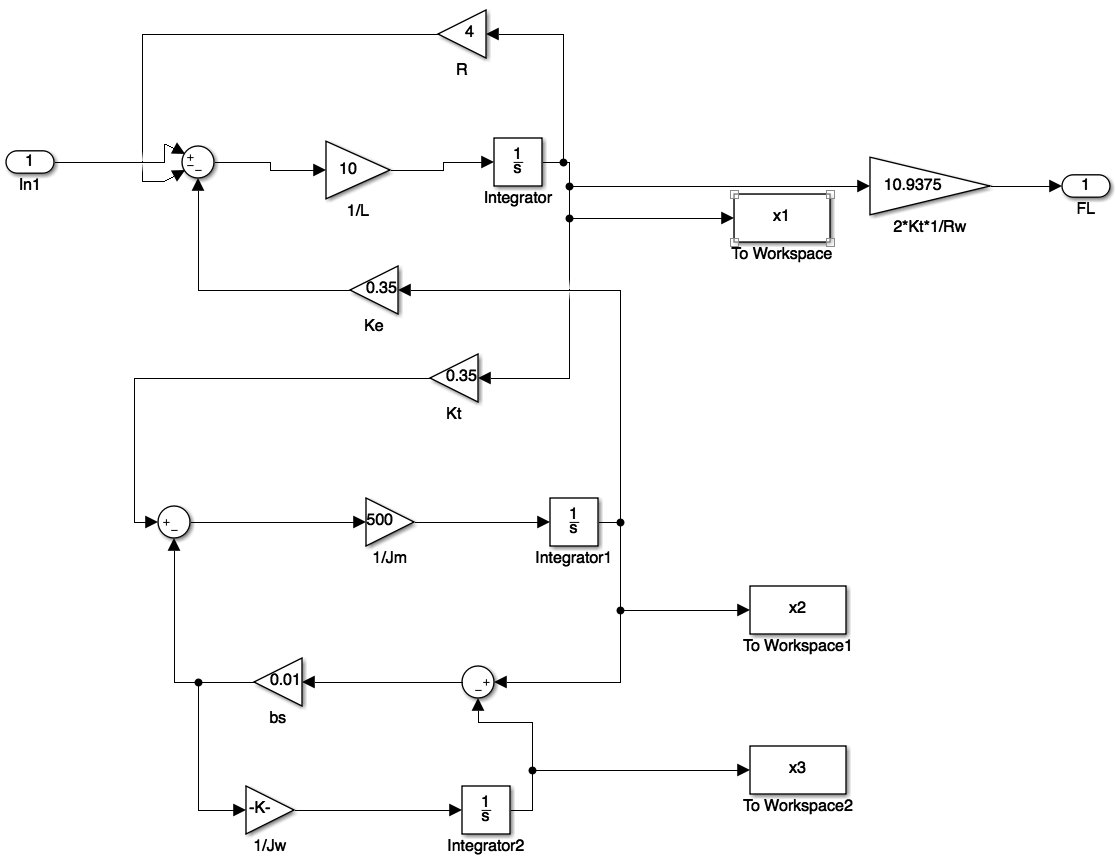


Figure 7. Left Motor Subsystem

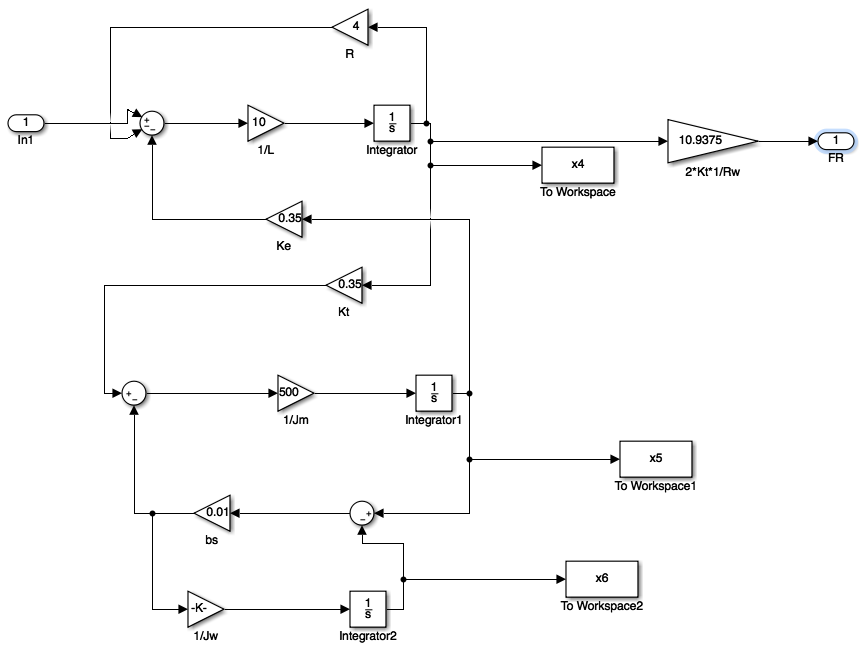


Figure 8. Right Motor Subsystem

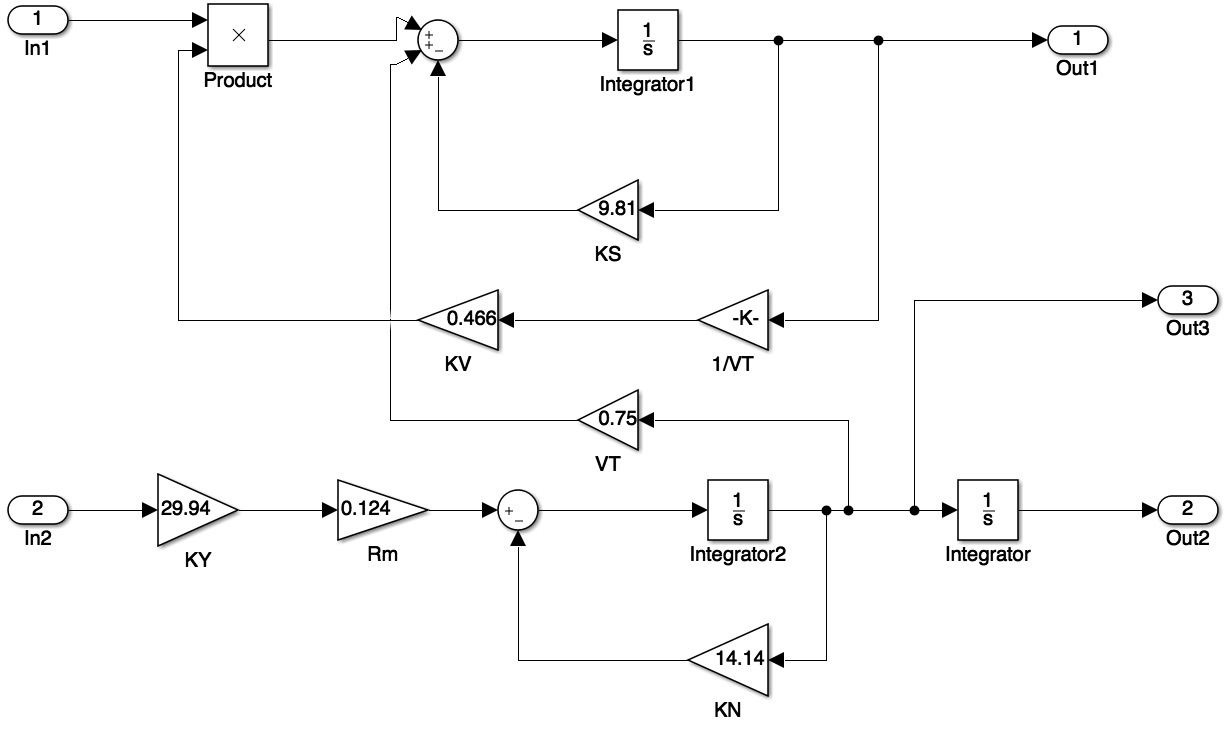


Figure 9. Sway and Yaw Dynamics Subsystem

Results

After the block diagrams were made in Simulink, the same values that were used in Part 1 were entered into the block diagram and was run. The plots that the block diagram made (shown in Figure 10) were very similar from the graphs in part 1 (Figure 2) when compared. This validated the accuracy of the simulation.

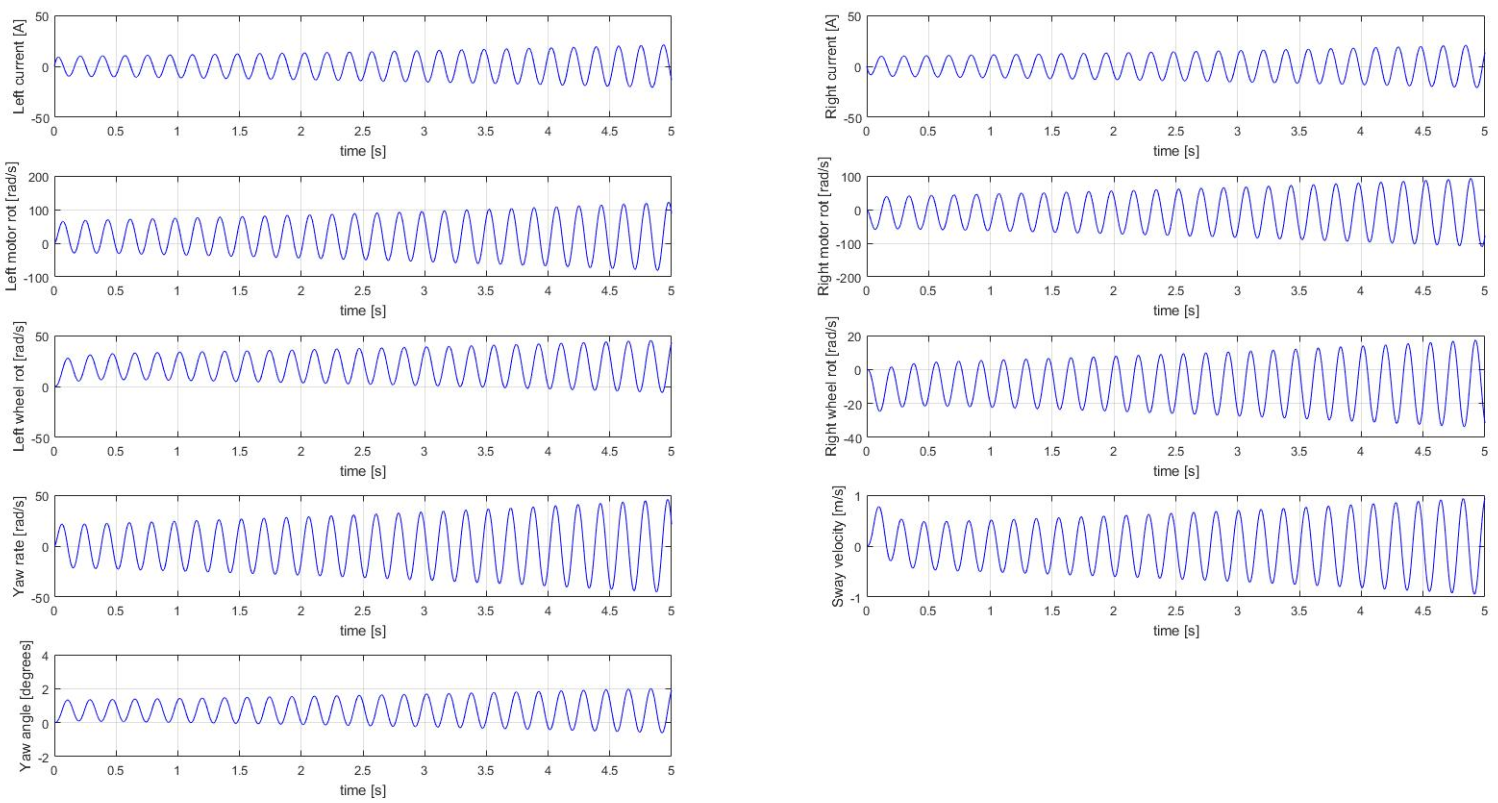


Figure 10. Simulation Plots of States against Time for Block Diagram

Part 3 – Finding the best Performance using values of Gc and KI

The purpose of part 3 was to introduce an integral term to investigate the effect on the Matlab simulation. The integral term would minimise the steady state error produced by the simulation.

It was established that by changing the values of Gc and KI, the rate at which the rover would turn was altered and this therefore had an impact on the rover’s performance. Now based on the application, the values of Gc and KI would have to be altered to give the best possible performance. An example being a space rover would be required to manoeuvre slowly in order to prevent any loss of traction on the terrain that it is exploring. Where as a rover that would be used for surveillance may be require to turn much faster. By examining the values of Gc, it was found that the rover would turn slower when a low value was inputted and would turn faster when a high value was inputted. Similarly, by increasing the value of KI, the turning would become more unstable. By entering a lower value of KI, this made the simulation much more stable with less fluctuations.

To ensure fast turning, values of Gc = 5 and KI = 0.01 were inputted. This resulted in the rover being able to turn from -5° to 45° in approximately 1 second shown in Figure 11.

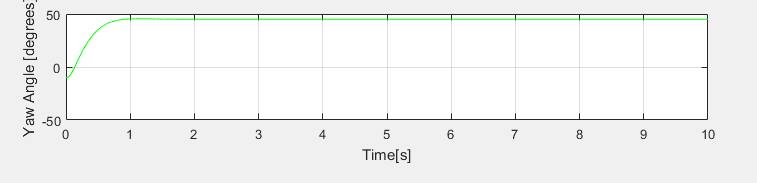


Figure 11. Fast turning Rover

To ensure slow turning, values of Gc = 0.01 and KI = 0.0001 were inputted. This resulted in the rover being able to turn from -5° to 45° in approximately 10 seconds shown in Figure 12.

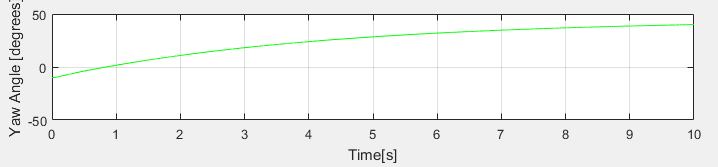


Figure 12. Slow turning Rover

Part 4 – Interpolation

For the part of the assignment, variations in the longitudinal dynamics were inserted into the simulation by adding a non-constant resultant forward velocity, VT, instead of having just one constant forward velocity. The data that was given is shown in Figure 13.

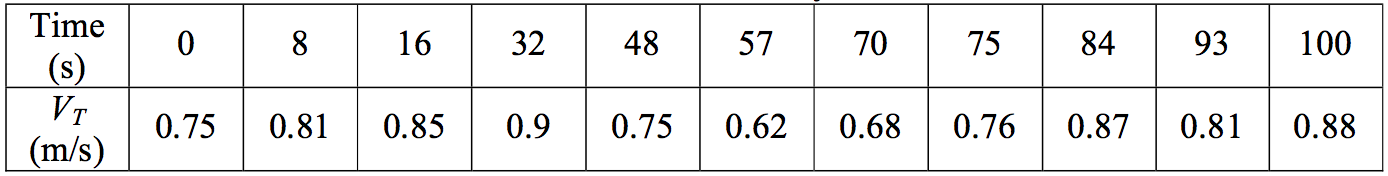


Figure 13. Velocity data against time

An interpolation routine was introduced in order to accommodate the changes in speed between the set data points. A quadratic spline was chosen for the interpolation. This works by having an equation for each of the quadratic spline defined within two interpolation points. For the case given, there is 11 data points and therefore, this would require 10 splines to be created using this from:

In order to use this equation correctly, the values of Z for each of the splines would have to be calculated. This was done using:

The calculated values of Z were then stored in an array by using the data shown in Figure 11 and then having a loop to generate the data. This code was inserted into the Initial Segment of the simulation shown in Figure 14 and in Appendix A.

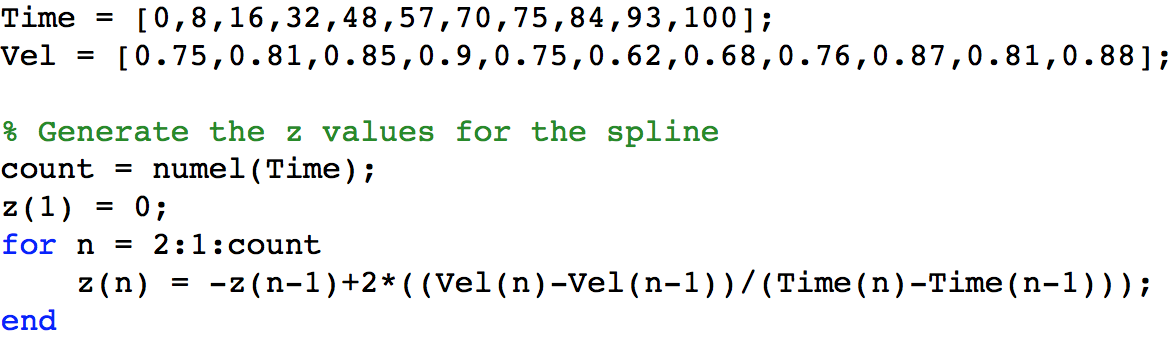


Figure 14. Matlab code for generating Z values

After all the Z values were calculated and stored, a loop and if statement was used to find the interpolating data. This was added to the Dynamic Segment of the simulation shown in Figure 15 and in Appendix A.

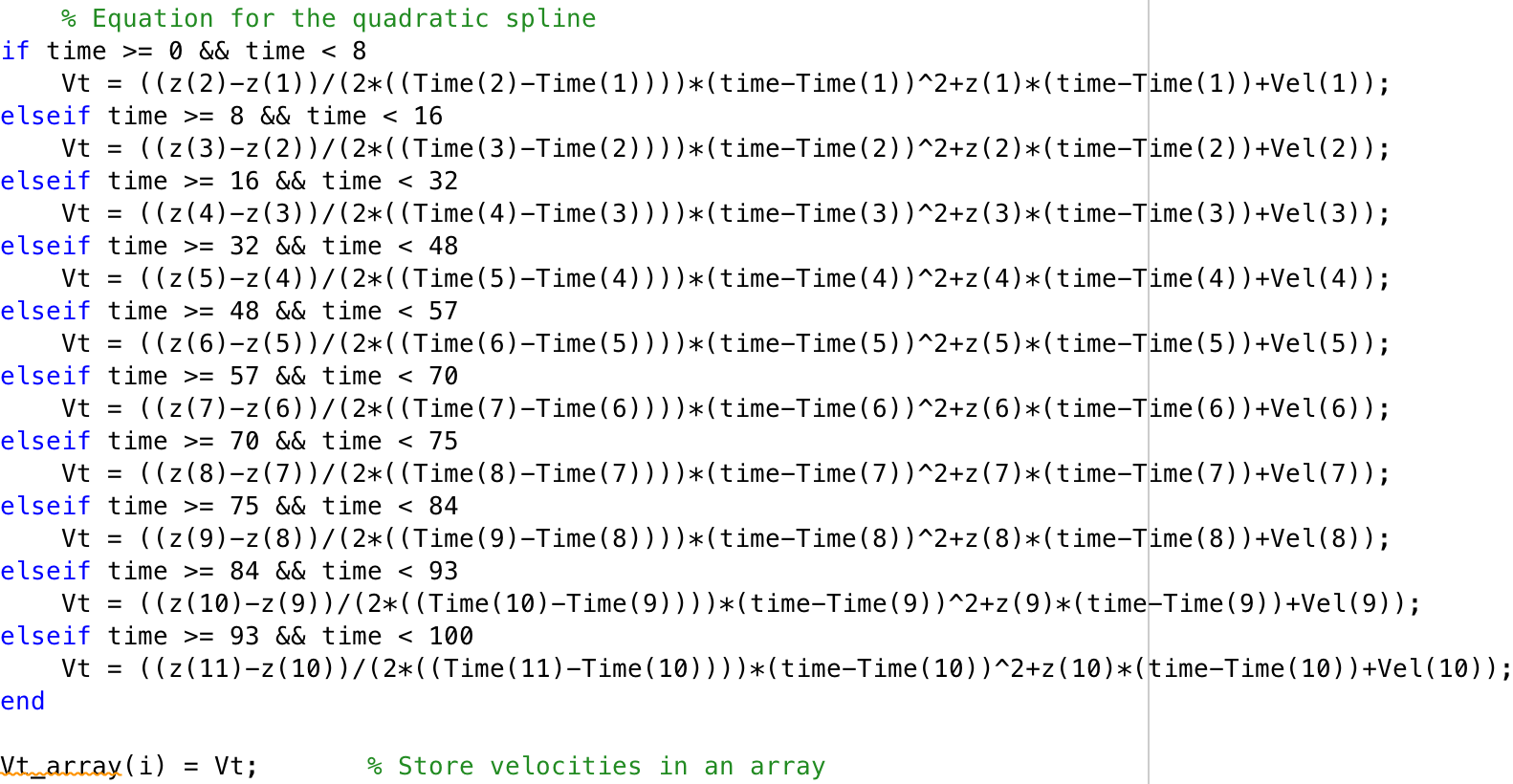


Figure 15. Matlab code for quadratic spline

Using the data from Figure 10, a different quadratic spline was used to calculate the velocity data. This was achieved through the use of elseif statements, shown in Figure 15. The velocity data generated was then plotted against time to obtain the graph of the quadratic spline. This is shown in Figure 16. The Matlab code for plotting Figure 16 is in Appendix C.

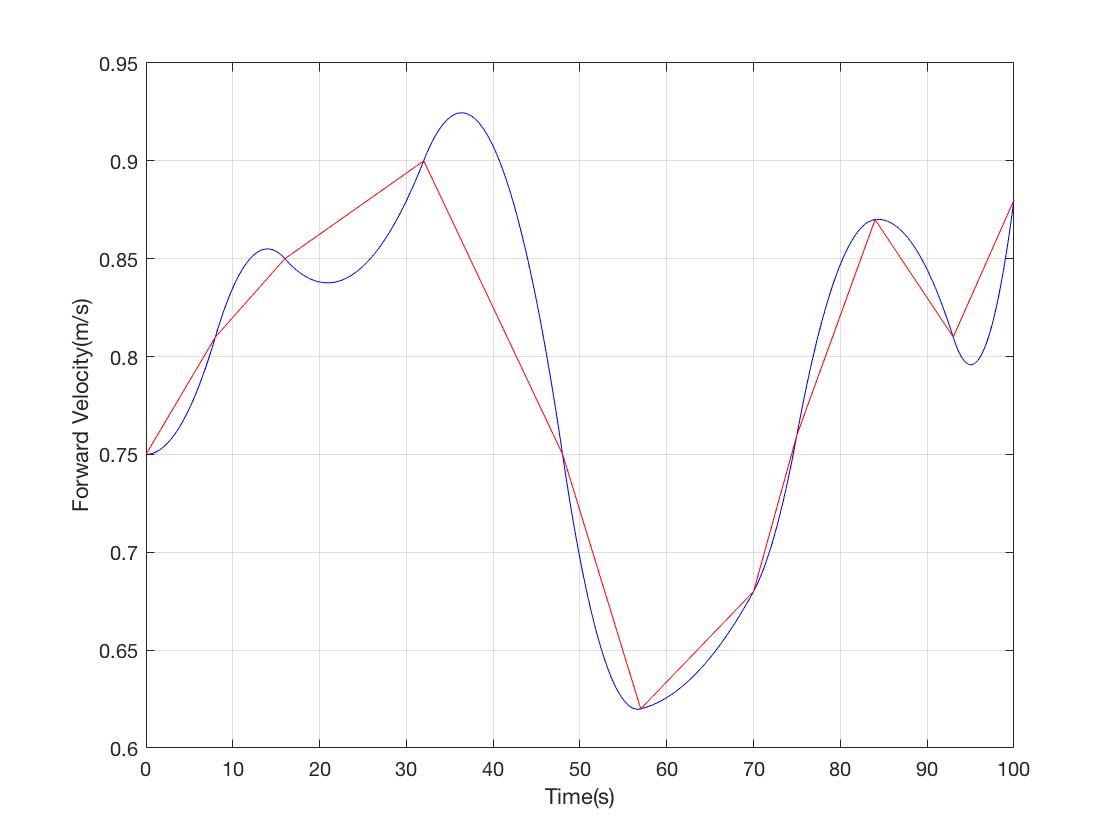


Figure 16. Quadratic Spline of Forward Velocity against Time

In Figure 16, the blue line represents the quadratic spline for the velocity data and the line connecting the interpolating points is in red. This figure shows that the code in Matlab did plot an accurate quadratic spline with the given data.

Conclusion

In conclusion, an equation based simulation of the head control system of a four wheeled rover was created using Matlab. Each of the states was then approximated through the use of Runge-Kutta as the numerical integration.

A block diagram was then created in Simulink which was to validate the Matlab model and simulation response produced before.

By incorporating an integral term into the Matlab model, the steady state error response was minimised. Depending on the application of the rover, the performance could be changed by altering the values of Gc and KI. Higher values of Gc would result in the rover turning faster and lower values would result in the rover turning slowly. Similarly, by increasing the value of KI, the turning would become more unstable. By entering a lower value of KI, this made the simulation much more stable with less fluctuations.

Lastly, with the use of quadratic splines, the varying velocities given could be interpolated and put into the Matlab model. This method allowed for any velocity values that feel below the data point to be determined.

References

<http://mars.nasa.gov/msl/mission/rover/>

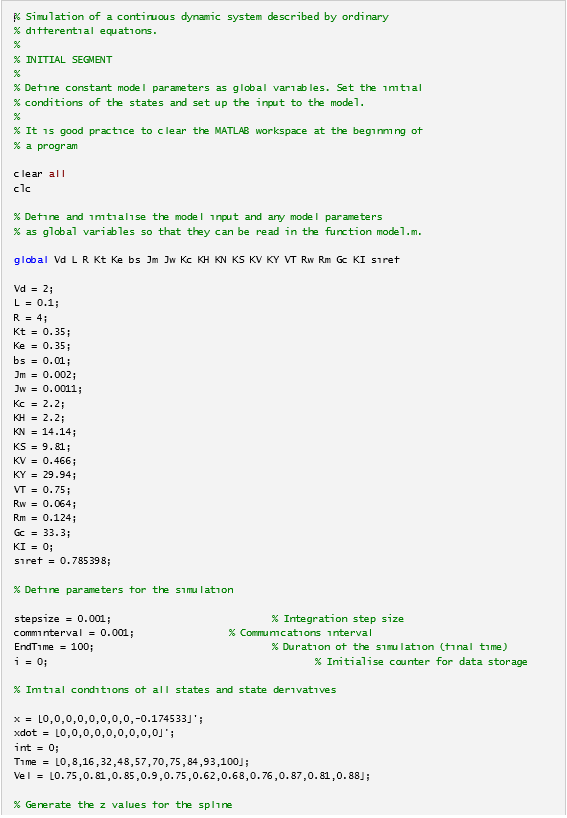
<https://en.wikipedia.org/wiki/Curiosity_(rover)>

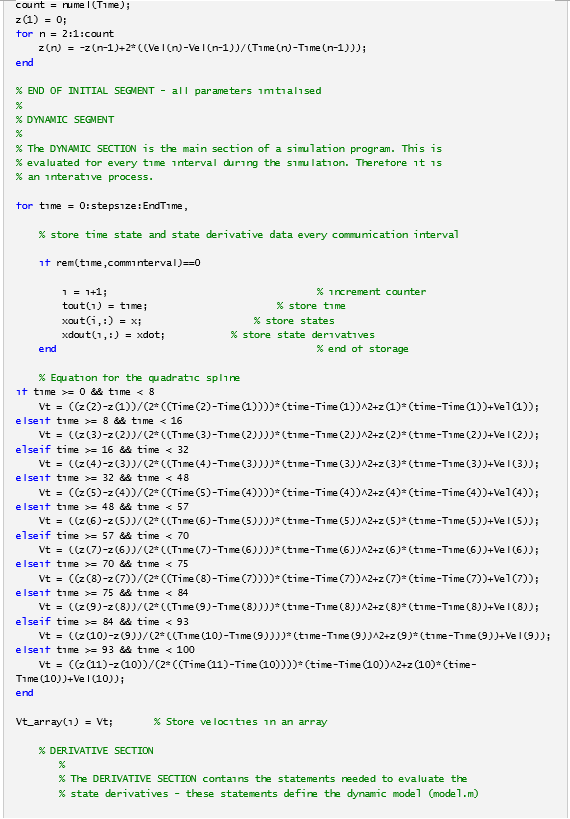
<https://en.wikipedia.org/wiki/MarkV-A1>

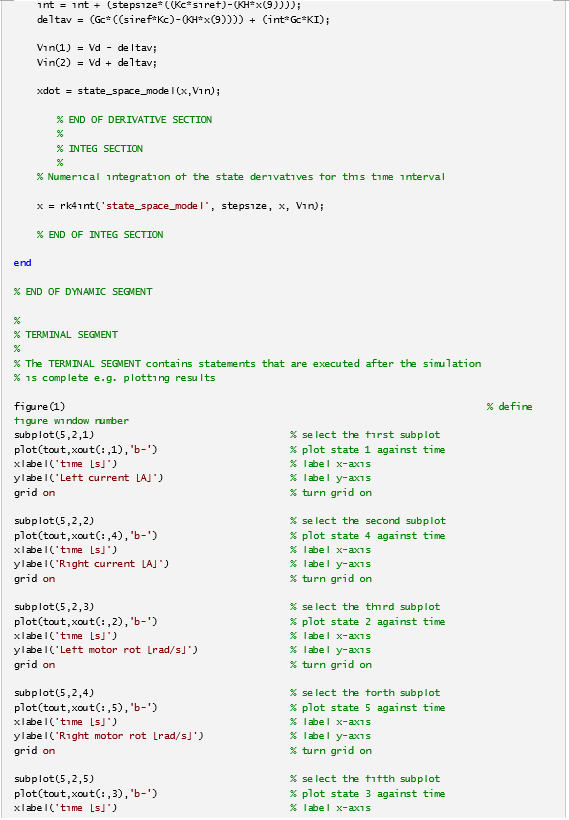
<https://airandspace.si.edu/exhibitions/exploring-the-planets/online/tools/rovers.cfm>

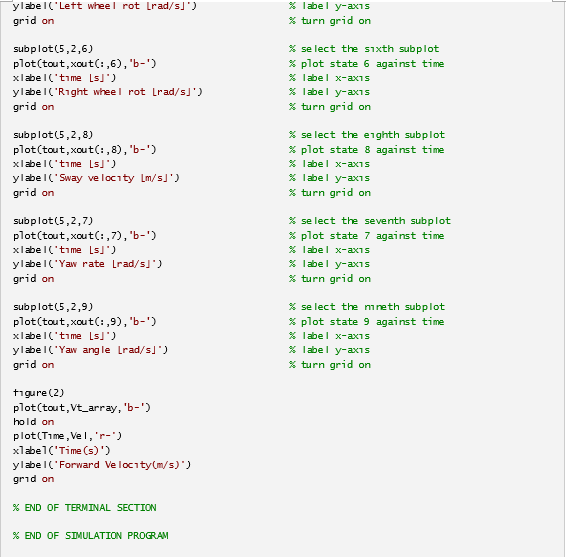
<http://www.bbc.co.uk/science/space/solarsystem/space_missions/mars_exploration_rover>

Appendix A: Matlab Code

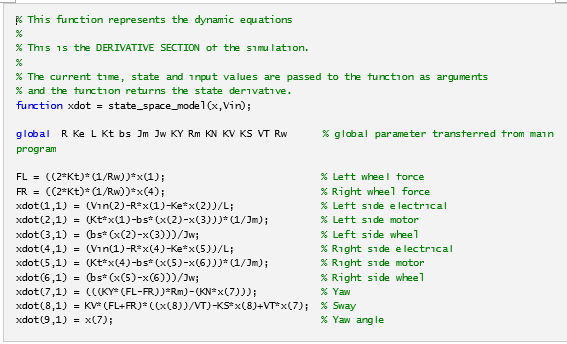








Appendix B: States Script



Appendix C: Interpolation Matlab Code

