MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL AND ELECTRONICS ENGINEERING

EE498 Term Project Report

Prepared by

Rafet KAVAK

2166783

Submission Date: *03/07/2020*

Table of Contents

1) Introduction 3
2) Part 1: Control System Design for Vehicle Platooning 4
2.1) System Model 4
2.2) Controller Design Methods 5
2.2.1) Bode Plot Design Method 5
2.2.2) Symmetric Optimum Controller Method 8
2.2.3) Pole Placement Method 9
2.2.4) Linear Quadratic Regulator (LQR) Method 9
2.2.5) Model Predictive Control (MPC) Method 10
2.3) Controller Design by Pole Placement 10
2.3.1) Parameter Decisions 15
2.3.2) Possible Limitations 15
2.3.3) Possible Disturbances 15
2.3) Simulations 16
3) Part 2: Simulation of The Trajectory of Apollo Satellite 22
3.1) Simulink Model Interpretation 22
3.1.1) Maximum Step Size 24
3.1.2) Relative Tolerance 25
3.1.3) Solver 26
3.1.4) Fixed Step Size 28
3.2) Explicit Runge-Kutta Method Implementation 29
3.2.1) Fixed Step Size 29
3.2.2) Variable Step Size 31
4) Conclusions 33
5) References 33
6) Appendices 34
6.1) Appendix A: MATLAB Script of the Part 1 34
6.2) Appendix B: Simulink Model of Apollo Satellite 38
6.3) Appendix C: MATLAB Script of the Part 2b – Fixed Step Size 39
6.3) Appendix D: MATLAB Script of the Part 2b – Variable Step Size 41

1) Introduction

Control Engineering is the branch of engineering that applies control theory to the design systems with desired behaviors. In practice, sensors and detectors are used to measure output of the system which is controlled and provide feedback to achieve desired performance. Control engineering is multi-disciplinary area and we came across its applications in many fields like automotive applications, process and production industry, manufacturing processes, home appliances, medical applications, robotics, etc. In order to develop such systems, EE498 course has provided great insight to us about control system design and simulation methods, and by this project, the course topics were covered one more time.

In the first part of the project, a controller design will be performed for a chosen system "Platooning". Firstly, the system model of the plant will be determined. Secondly, controller methods that were studied in the lecture will be classified as applicable or not applicable for the system. After choosing of the controller design method, controller will be realized and decision of the relevant parameters like steady state error, closed loop dynamics, etc. will be discussed. Also, possible limitations arising from actuator saturation, plant properties, etc. will be discussed. Then, the effects of the possible disturbances will be analyzed. Finally, the designed controller will be validated by simulations that are performed in Simulink.

In the second part of the project, the trajectory of the Apollo Satellite that orbiting around the earth in a plane will be simulated in Simulink. Firstly, after realization of the model in Simulink, numerical solutions for different parameters of the solver will be determined. Secondly, the Explicit Runge-Kutta method for $p = 4$ will be implemented. Then, this method will be tested for fixed and variable step sizes. Thirdly, the Runge-Kutta method will be extended by step-size adaptation and the error will be tried to be estimated. Lastly, my own solver will be tested, compared with the previous parts and interpreted according to change in the step sizes.

2) Part 1: Control System Design for Vehicle Platooning

Vehicle platooning, i.e., convey driving which can be seen in Figure 1, is a group of vehicles driving closely after each other autonomously. It improves fuel consumption, transport efficiency, traffic flow, etc. Due

Figure 1 An example of Truck Platooning

to safety reasons, the control mechanism of the system must be worked flawlessly because the system is expected to be autonomous. The vehicles in the platoon must automatically adjusts their speed and maintains a safe distance to the vehicle ahead. Therefore, Adaptive Cruise Control (ACC) system must be used for adjusting the desired inter vehicle distances.

2.1) System Model

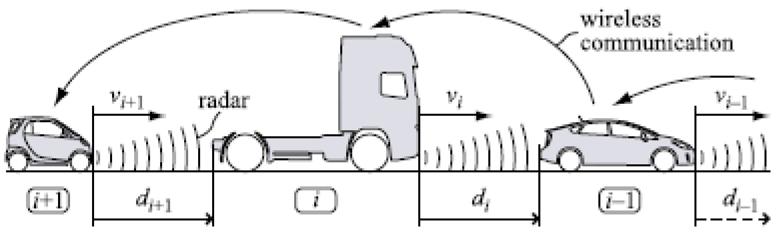


Figure 2 The Platoon model

In order to obtain a system model, let us consider a platoon of m vehicles as can be shown in Figure 2 where d_i is the distance between i^{th} vehicle and its preceding vehicle $i-1$, and v_i is the velocity of i^{th} vehicle. Then the objective of each vehicle is to follow the preceding vehicle at a desired distance $d_{r,i}$ as

$$d_{r,i}(t) = r_i + hv_i(t) \tag{1}$$

where h is the time headway, i.e., the time between successive vehicles passing a fixed point and r_i is the standstill distance, i.e., the distance between stopped vehicles. For the system model, state variables can be chosen as the distance d_i, the velocity v_i and the acceleration of vehicle i, a_i. By considering external input u_i and time constant τ which represents the engine dynamics, overall model can be defined as

$$\begin{bmatrix} \dot{d}_i(t) \\ \dot{v}_i(t) \\ \dot{a}_i(t) \end{bmatrix} = \begin{bmatrix} v_{i-1}(t) - v_i(t) \\ a_i(t) \\ -\frac{1}{\tau} a_i(t) + \frac{1}{\tau} u_i(t) \end{bmatrix} \tag{2}$$

Then, the frequency domain model of a platoon vehicle can be formulated as

$$G(s) = \frac{Q_i(s)}{U_i(s)} = \frac{1}{s^2(\tau s + 1)} \tag{3}$$

where q_i is the position of vehicle i according to

$$\dddot{q}_i = -\frac{1}{\tau}\ddot{q}_i(t) + \frac{1}{\tau}u_i(t) \tag{4}$$

Also, can be shown as

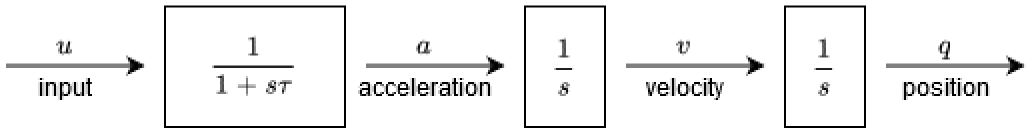


Figure 3 Block diagram of system model

2.2) Controller Design Methods

In EE498, several controller design methods have been studied and these can be listed as,

- P-PI-PD-PID Controller design by using Bode Plot Method
- Symmetric Optimum Controller Method
- Pole Placement Method
- Linear Quadratic Regulator (LQR) Method
- Model Predictive Control (MPC) Method

The main goal of the controller design is that satisfying the system requirements and providing a robust system. In our case, oscillation and overshoot is undesired conditions because the vehicle can cause an accident. Moreover, steady state error must be zero again for the safety conditions. Therefore, these conditions will be considered while designing the controller. Let us explain which of the controller design methods that we studied during the lecture are applicable or not applicable to the platooning system and MATLAB scripts of the design methods and the chosen method can be found in Appendix A.

2.2.1) Bode Plot Design Method

There are some specifications for bode plot design in terms of desired time domain behavior and open loop transfer function $G_0(s)$. The internal stability, small steady state error, sufficient damping/overshoot and fast closed loop response must be satisfied for the desired time domain behavior. Moreover, open loop transfer function must fulfill the stability criterion. Also, $G_0(s)$ should have large open loop gain at small frequencies and suitable phase margin.

Firstly, let us inspect the bode plot of the plant

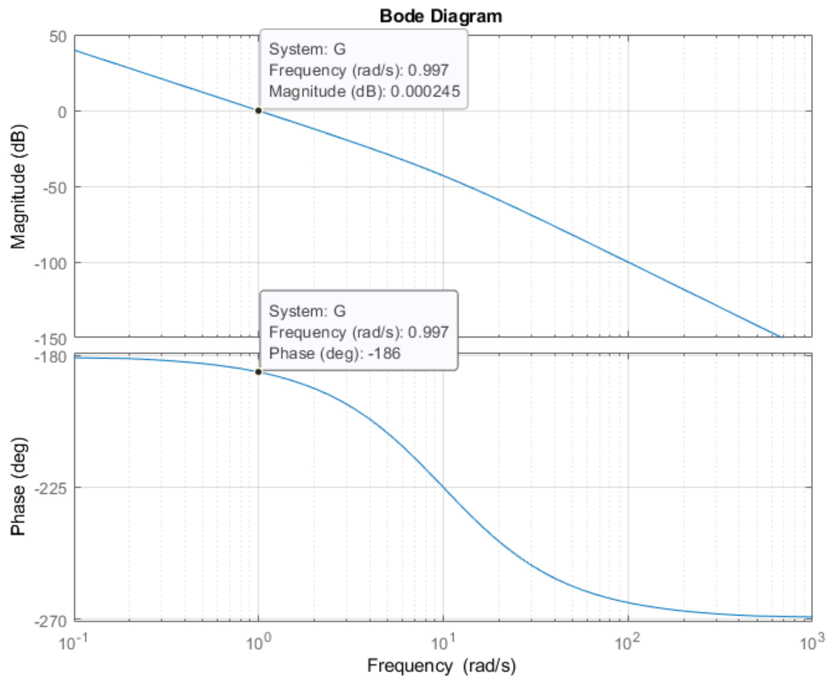


Figure 4 Bode plot characteristics of the plant

Plant is not internally stable because the phase margin is less than 0. It is expected because it has two integrators which two of poles of the plant are on the origin. Therefore, in the following sections, controller types for bode plot design will be examined whether they are applicable or not for the plant.

2.2.1.1) Proportional (P) Controller

For the following four controller type, firstly the controller transfer function will be given, then the open loop transfer function will be analyzed with bode plot if necessary.

$$C(s) = K_P \tag{5}$$

For the K_P values higher than 1, magnitude plot will be shifted up, otherwise shifted down and phase plot remains unchanged. As magnitude plot shifted up, phase margin will be decreased more, and the system will be still unstable. Moreover, if the magnitude plot shifted down, phase margin will be increased but it will never higher than zero. This is expected because open loop system has two integrators and it has no zeros. Therefore, phase margin always will be less than zero. The resulting bode plot can be seen in the Figure 5 below.

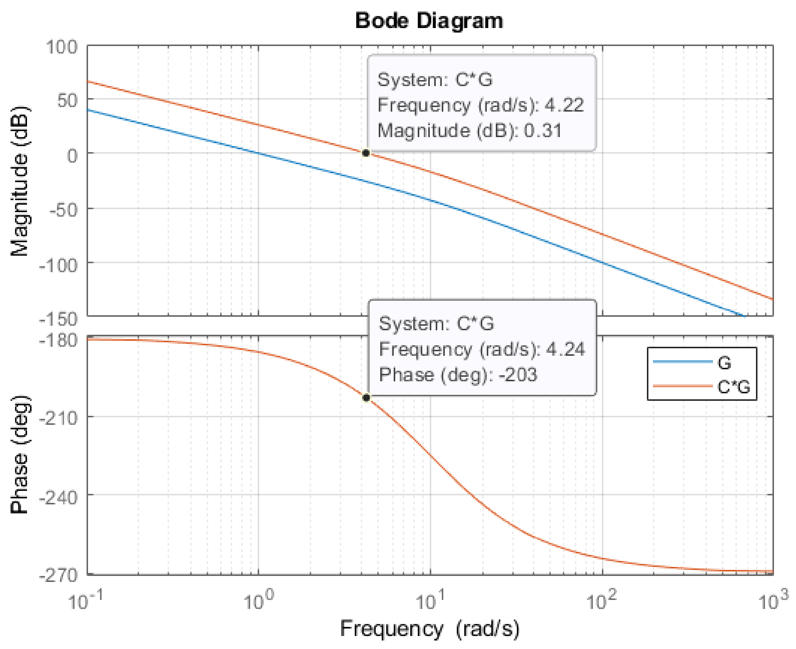


Figure 5 Bode Plot of the OLTF when P-Type controller used

2.2.1.2) Proportional-Derivative (PD) Controller

$$C(s) = K_P(1 + sT_D) \tag{6}$$

PD controller increases the phase plot by 90° at higher frequencies and does not change the phase plot at lower frequencies. Therefore, choosing the right parameters, open loop system can be stable for this case.

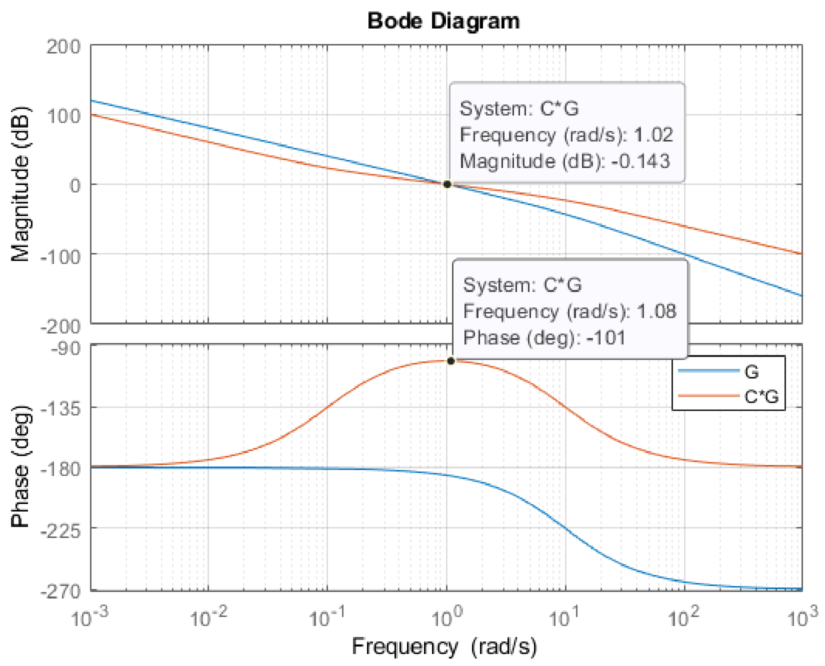


Figure 6 Bode Plot of the OLTF when PD-Type controller used

2.2.1.3) Proportional-Integral (PI) Controller

$$C(s) = K_P\left(1+\frac{1}{sT_I}\right) = K_P\frac{1+sT_I}{sT_I} \tag{7}$$

PI controller decrease the phase plot by 90° at lower frequencies and does not change the phase at high frequencies. Therefore, open loop system will never be stable. Consequently, PI controller is not applicable for controller design.

2.2.1.4) Proportional-Integral-Derivative (PID) Controller

PID controller also shifts the phase plot up at high frequencies, so it is applicable too as PD controller.

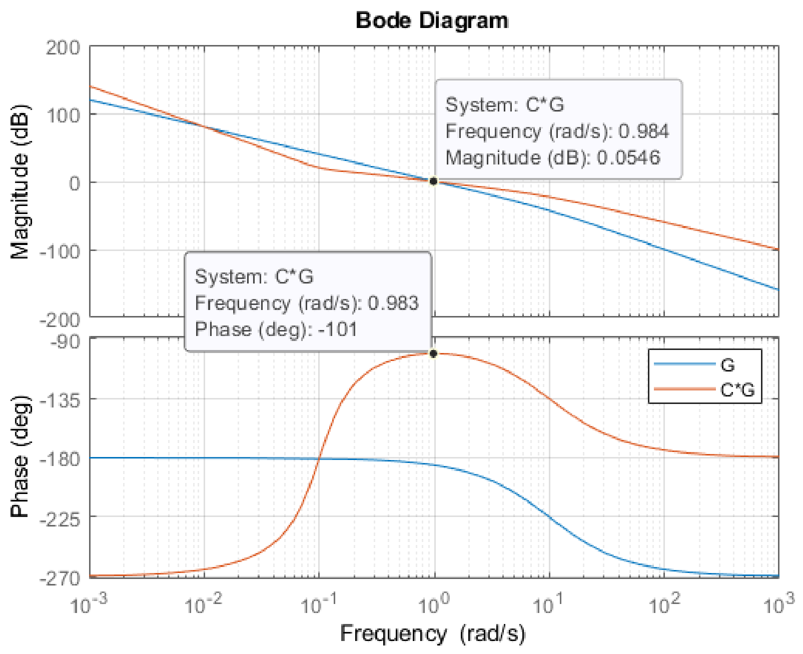


Figure 7 Bode Plot of the OLTF when PID-Type controller used

2.2.2) Symmetric Optimum Controller Method

In order to be use Symmetric Optimum Controller Method, there are some prerequisites. First of all, plant must be in the form of

$$G(s) = \frac{K}{(1+s\sigma)(1+sT_1)\ldots(1+sT_q)(sT_{q+1})\ldots(sT_n)} \tag{8}$$

where

- $\sigma\ is\ a\ small\ time\ constant$
- $T_1, \ldots T_q\ are$ large $time\ constants\ such\ that\ T_i \geq 10\sigma$
- $n+1\ is\ the\ system\ order$

Our system is third order so we need to design a symmetric optimum controller with n=2, but it seems to me quite complicated when n=2 and I could not verify whether this method is applicable or not.

2.2.3) Pole Placement Method

The main goal of pole placement method is placing the closed-loop poles to the pre-specified pole locations and the design problem can be considered as for a given plant G(s) and desired closed-loop denominator polynomial R(s), someone must find a controller transfer function C(s) such that

$$G(s) = \frac{B(s)}{A(s)}, \qquad C(s) = \frac{P(s)}{L(s)}, \qquad A(s)L(s) + B(s)P(s) = R(s) \tag{9}$$

In our case, if we choose our desired closed-loop pole locations coherent, pole placement method can be applicable for the controller design.

There are some properties of the pole placement method. For example, we can adjust the speed of the system according to the desired pole locations. Also, these locations determine the damping and the overshoot. Furthermore, by placing the poles on the real axis, someone can avoid from the oscillations. In order to be use pole placement, the system must be controllable, and all states must be available to us. The drawback of this method is that the closed loop zeros can cause overshoot or undershoot but this issue can be fixed by using pre-filter.

2.2.4) Linear Quadratic Regulator (LQR) Method

Linear Quadratic Regulator Method is an optimal control method that tries to solve an optimal control problem in Equation 10.

$$\min_{u_0,\dots,u_{N-1}} \left\{ J = X_N^T Q_f x_N + \sum_{k=0}^{N-1} \left(x_k^T Q x_k + u_k^T R u_k \right) \right\} \tag{10}$$

subject to

$$x_{k+1} = Ax_k + Bu_k \; for \; k = 0, \dots, N-1 \; and \; x_0 \; given \tag{11}$$

LQR tries to minimize the cost function by determining the values of the feedback gain matrix K, so LQR is also can be considered as pole placement method but there is a fundamental difference between pole placement method and LQR method. In the pole placement method, we can choose the pole locations but in LQR method, closed loop poles are located according to the cost function that makes the cost function is minimum. LQR method cannot be said applicable directly like pole placement method because sometimes it can fail while preventing the oscillations due to costs of the states. In order to avoid from this situation, model predictive controller method can be used because future signals can be considered to determine the input sequence.

2.2.5) Model Predictive Control (MPC) Method

Model Predictive Control (MPC) method solves the finite horizon optimal control problem (Eq. 12) for some future steps N by taking account of the current and future constraints (Eq. 14-17). To achieve this purpose, the current state must be measured, and a finite horizon optimum control problem must be solved online while considering the current and the future constraints.

$$\min_{u_{0|k},\dots,u_{N-1|k}} \left\{ J = \sum_{j=0}^{N-1} \left(x_{j|k}^T Q x_{j|k} + u_{j|k}^T R u_{j|k} \right) + x_{N|k}^T P x_{N|k} \right\} \tag{12}$$

$$x_{j+1|k} = A x_{j|k} + B x_{u|k} \text{ } where\ j = 0, 1, \dots, N-1\ and\ x_{0|k}\ given \tag{13}$$

$$State\ constraint: x_{min} \leq x_{j|k} \leq x_{max}\ where\ j = 0, 1, \dots, N-1 \tag{14}$$

$$Input\ constraint: u_{min} \leq u_{j|k} \leq u_{max}\ where\ j = 0, 1, \dots, N-1 \tag{15}$$

$$f_x^T x_{j|k} \leq v_x\ where\ j = 0, 1, \dots, N-1 \tag{16}$$

$$f_u^T u_{j|k} \leq v_u\ where\ j = 0, 1, \dots, N-1 \tag{17}$$

Actually, MPC method is an advanced method, so it is the most effective controller method among all these methods. The other methods do not include the future values of the inputs therefore they can deviate from the expected performance especially away from the equilibrium point, but MPC can compensate the undesired effects and provides more robust system. Therefore, it is applicable for our system. However, it is hard to implement compared with the previous methods.

2.3) Controller Design by Pole Placement

Among all these models, pole placement seems more appropriate because it is easier to implement within all the design procedures.

<u>Design Procedure</u>

- General Plant:

$$G(s) = \frac{B(s)}{A(s)} = \frac{b_0 + \dots + b_n s^n}{a_0 + \dots + a_n s^n} \tag{18}$$

$$G(s) = \frac{1}{s^2(\tau s + 1)}, \qquad n = 3 \tag{19}$$

- Controller: Because of the plant that has two integrators, we do not need to an integrator in the controller TF. Therefore, we can use controller transfer function of order $m = n - 1$.

$$C(s) = \frac{P(s)}{L(s)} = \frac{p_0 + \dots + p_m s^m}{l_0 + \dots + l_m s^m} \tag{20}$$

$$m = n - 1 = 3 - 1 = 2 \tag{21}$$

$$C(s) = \frac{p_0 + p_1 s + p_2 s^2}{l_0 + l_1 s + l_2 s^2} \tag{22}$$

- Write down the desired equation

$$R(s) = A(s)L(s) + B(s)P(s) = r_0 + \cdots + r_p s^p \tag{23}$$

$$R(s) = s^2(\tau s + 1)(l_0 + l_1 s + l_2 s^2) + (p_0 + p_1 s + p_2 s^2) \tag{24}$$

$$R(s) = \tau l_2 s^5 + (\tau l_1 + l_2)s^4 + (\tau l_0 + l_1)s^3 + (l_0 + p_2)s^2 + p_1 s + p_0 \tag{25}$$

- Let us place all the closed loop poles to the $s = -1$ where $p = m + n$. Then our desired closed loop polynomial will become

$$R(s) = (s+1)^5 = s^5 + 5s^4 + 10s^3 + 10s^2 + 5s + 1 \tag{26}$$

- After insertion of τ compute the free parameters by comparison of coefficients.

$$0.1l_2 = 1 \quad \Longrightarrow \quad l_2 = 10$$

$$0.1l_1 + l_2 = 5 \quad \Longrightarrow \quad l_1 = -50$$

$$0.1l_0 + l_1 = 10 \quad \Longrightarrow \quad l_0 = 600$$

$$l_0 + p_2 = 10 \quad \Longrightarrow \quad p_2 = -590$$

$$p_1 = 5$$

$$p_0 = 1$$

- Then the controller becomes

$$C(s) = \frac{1 + 5s - 590s^2}{600 - 50s + 10s^2} \tag{27}$$

After designing the controller, step response of the system was obtained by MATLAB and it can be seen in Figure 8.

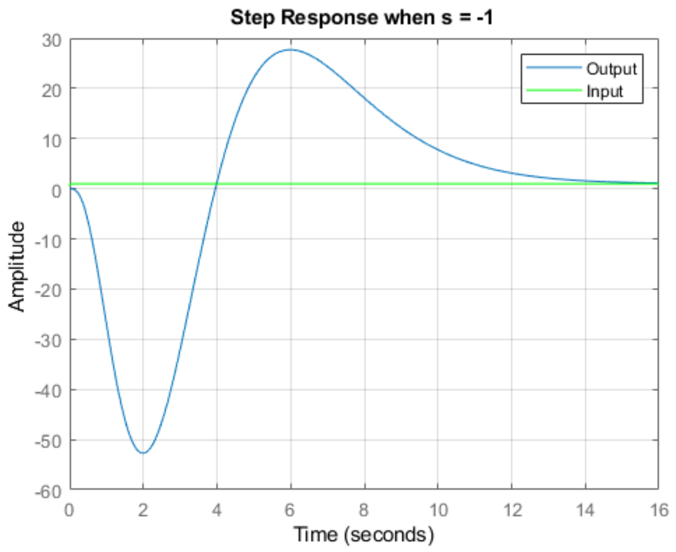


Figure 8 Step response of the system when desired closed loop poles at s = -1

There is no steady state error in the system as expected. However, values of the undershoot and overshoot is high, and the system has slow dynamics. At this step, let us chance our desired closed loop pole locations and try several pole locations.

- When desired closed loop poles at $s = -2$,

$$C(s) = \frac{32 + 80s + 400s^2}{-390 + 49s + 10s^2} \tag{28}$$

- When desired closed loop poles at $s = -3$,

$$C(s) = \frac{243 + 405s - 130s^2}{400 + 50s + 10s^2} \tag{29}$$

- When desired closed loop poles at $s = -4$,

$$C(s) = \frac{1024 + 1280s + 40s^2}{600 + 100s + 10s^2} \tag{30}$$

- When desired closed loop poles at $s = -5$,

$$C(s) = \frac{3125 + 3125s + 250s^2}{1000 + 150s + 10s^2} \tag{31}$$

- When desired closed loop poles at $s = -6$,

$$C(s) = \frac{7776 + 6480s + 560s^2}{1600 + 200s + 10s^2} \tag{32}$$

Then, let us see the step responses on the same graph for the controllers above

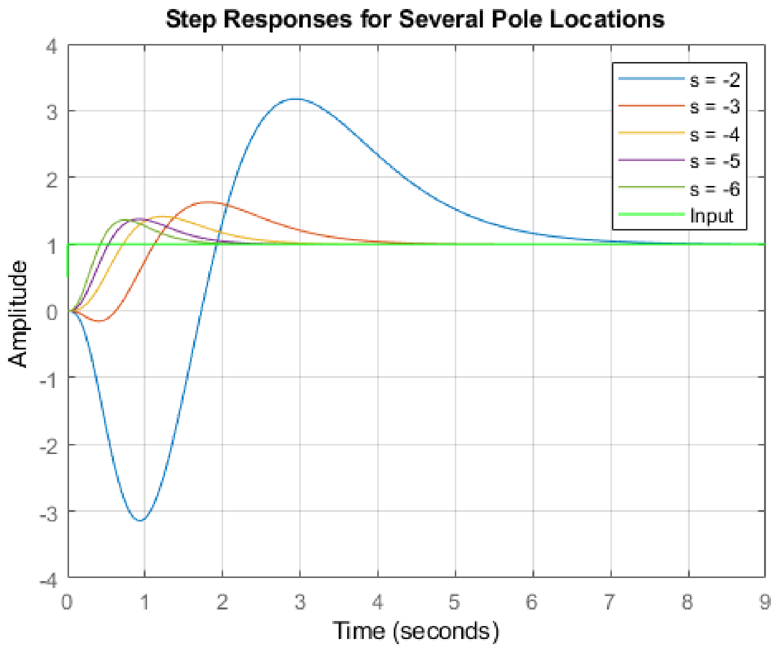


Figure 9 Step responses of the system for several desired pole locations

Also, exclude $s = -2$ condition for better resolution

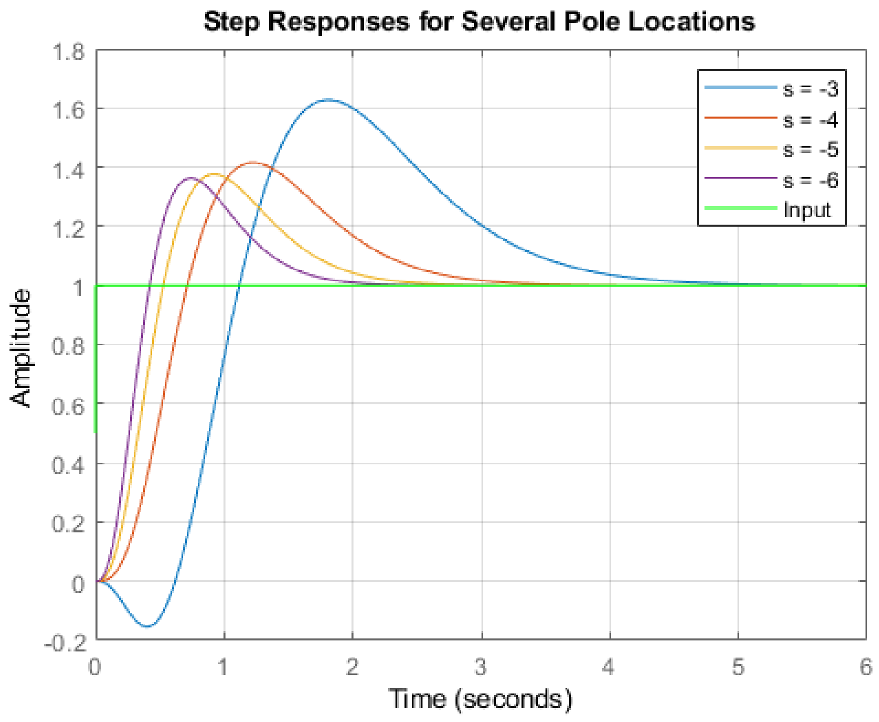


Figure 10 Step responses of the system for several desired pole locations (s=-2 excluded)

After $s = -3$, there are no undershoots for the desired closed pole locations $s = -4,5,6$. As the pole locations move left, the system has better response, but controller parameters also increases as the magnitude of the poles increases. Therefore, at this point, $s = -5$ was chosen for the desired pole locations because of good response and relatively low controller parameters.

There is also the way that to eliminate the overshoot and this could be achieved by using a pre-filter. A pre-filter can be designed by firstly decomposing the closed loop transfer function numerator and cancelling the stable zeros of the T(s).

$$T(s) = \frac{C(s)G(s)}{1 + C(s)G(s)} = \frac{C(s)G(s)}{R(s)} = \frac{250s^2 + 3125s + 3125}{s^5 + 25s^4 + 250s^3 + 1250s^2 + 3125s + 3125} \quad (33)$$

$$T(s) = \frac{V^+(s)V^-(s)}{R(s)} = \frac{250(s + 1.09611)(s + 11.40388)}{R(s)} \quad (34)$$

where

$$V^+(s): Polynomial\ with\ instable\ zeros\ of\ T(s)\ \ (RHP)$$

$$V^-(s): Polynomial\ with\ stable\ zeros\ of\ T(s)\ \ (ORHP)$$

Therefore,

$$V^+(s) = 250, \qquad V^-(s) = (s + 1.09611)(s + 11.40388) \tag{35}$$

Then the pre-filter transfer function is

$$F(s) = \frac{R(0)}{V^-(s)V^+(0)} = \frac{3125}{250(s + 1.09611)(s + 11.40388)} = \frac{12.5}{(s + 1.09611)(s + 11.40388)} \tag{36}$$

$$F(s) = \frac{12.5}{s^2 + 12.5s + 12.5} \tag{37}$$

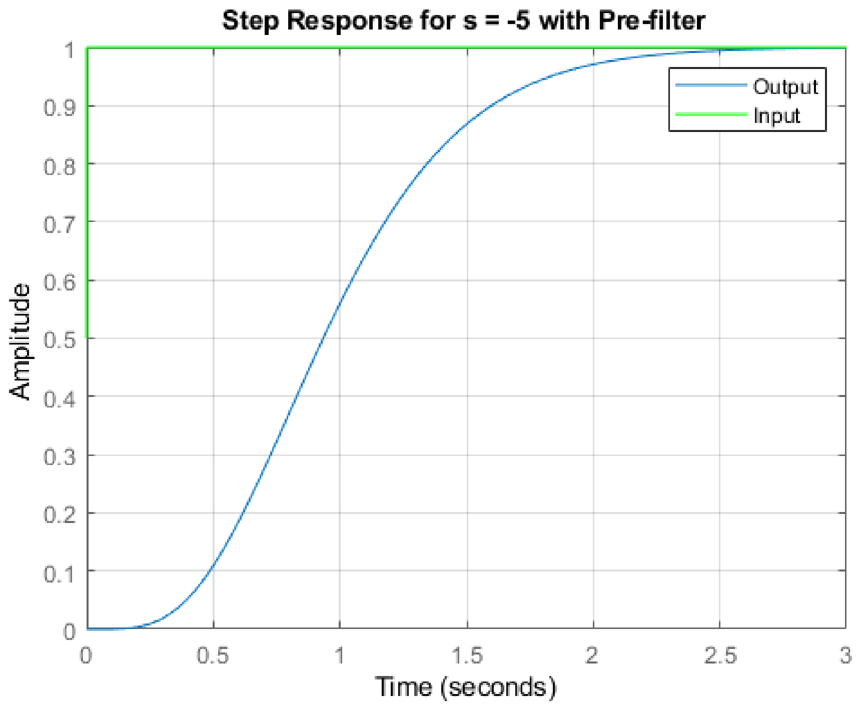


Figure 11 Step response of the system with pre-filter

As can be seen in the Figure 11, the overshoot on the output was eliminated by using a pre-filter. Note that these calculations are for the one vehicle. In order to see how the complete platooning system, the simulations will be performed by using Simulink.

2.3.1) Parameter Decisions

Steady state error is the strongly undesirable situation in the automotive area because even if small deviation can cause big problems such that cost, health, etc. Therefore, zero steady state error is one of the main goals. Moreover, we expect robust and fast system that responds the input quickly, but there must be no overshoot in the system. For these purposes, pole placement method gives quite satisfying results especially with the pre-filter.

The parameters of the controller were calculated above and according to the pole locations, their values are changing correspondingly. The optimization of the pole location can be done detailed for the professional purposes.

Also, the value of the time constant is chosen as 0.1 sec as in the Ploeg's Article. However, the delay was not considered in this project for the sake of simplicity.

2.3.2) Possible Limitations

In reality, some assumptions can be lost their validity and some limitations will arise due to practical reasons. For example, actuators might not supply arbitrary control input values to the plant all the time. Because of the temperature or some excessive voltage/power values, some assumptions can be lost their linearities. Moreover, rate of change of the actuator movement, i.e., slew rate is limited in practice.

In our system, we can use the bounds for acceleration and velocity for the controller design in order to do not exceed the safety bounds. Also overshoot and undershoot were eliminated in the previous parts.

There also other possible limitations as

- Traction force limit(acceleration)
- Quality of the sensor measurements
- The distance should stay positive and nearly constant for the standard case
- Difference of the vehicles (different characteristics, different time constants)
- Control Sensitivity

2.3.3) Possible Disturbances

There are many disturbances for the platooning system, and these can be listed as

- Different characteristics of speed up and breaking
- Wind and weather conditions especially for the heavy duty vehicles
- Curved roads
- Vehicle interrupt because of the other drivers
- Slope of the road
- Measurement errors due to environmental conditions
- Obstacles on the roadway (Like animals, rocks, etc.)

Also, the input disturbance and output disturbance as well as the noise will be simulated in the following part.

2.3) Simulations

At this point, I have struggled a bit because I tried to simulate a platooning system as a whole, i.e., I added two vehicles in series, but because of the I have only one input, I could not control the system entirely because other parameters like initial/actual velocities and initial/actual accelerations were needed in order to control the Platooning system. Instead, I have performed the simulations for one vehicle that takes the input from the car in front of it. For example, if the leading car stopped at the red light, then the following car received 0 input or if the leading car accelerating or decelerating, this condition is simulated as the following car receives sinusoidal input. To show other cases like disturbance rejection, limitations and initial conditions, step input was used.

The Simulink model of the system can be seen in the Figure 12&13 below. The following simulations will be performed by manipulating the disturbances, initial conditions and limitations.

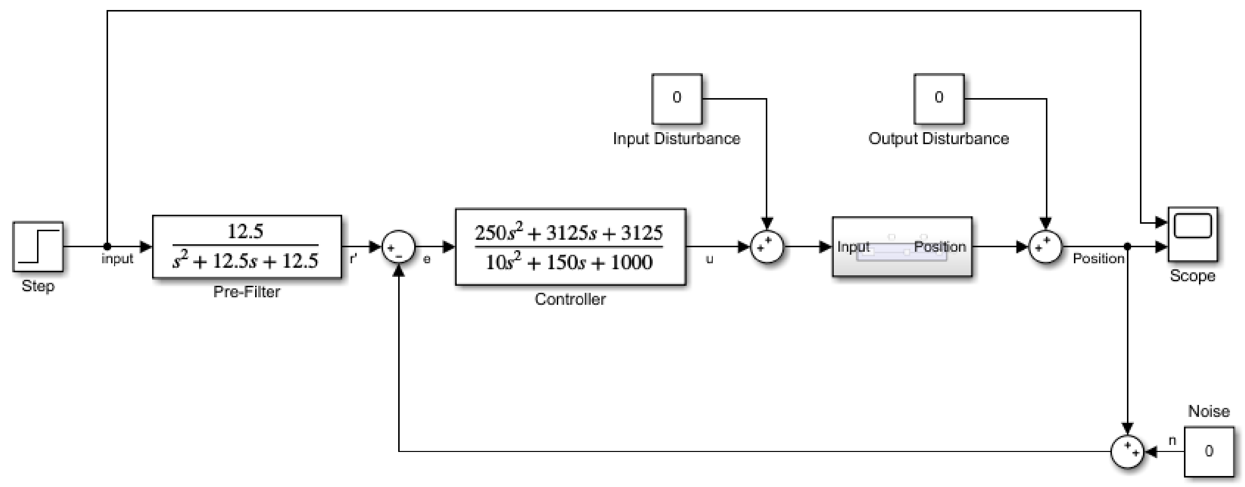


Figure 12 Simulink model realization of the system (one vehicle)

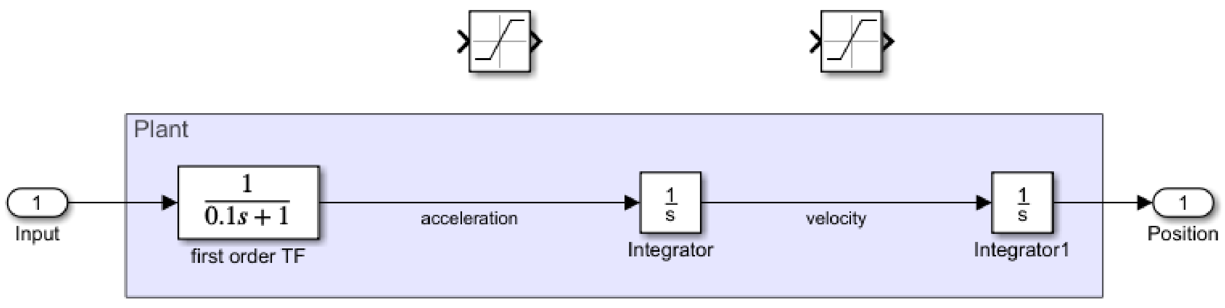


Figure 13 Subsystem of the plant with not connected saturation blocks

- Simulations without disturbances, initial conditions, saturation blocks. (The basic form)

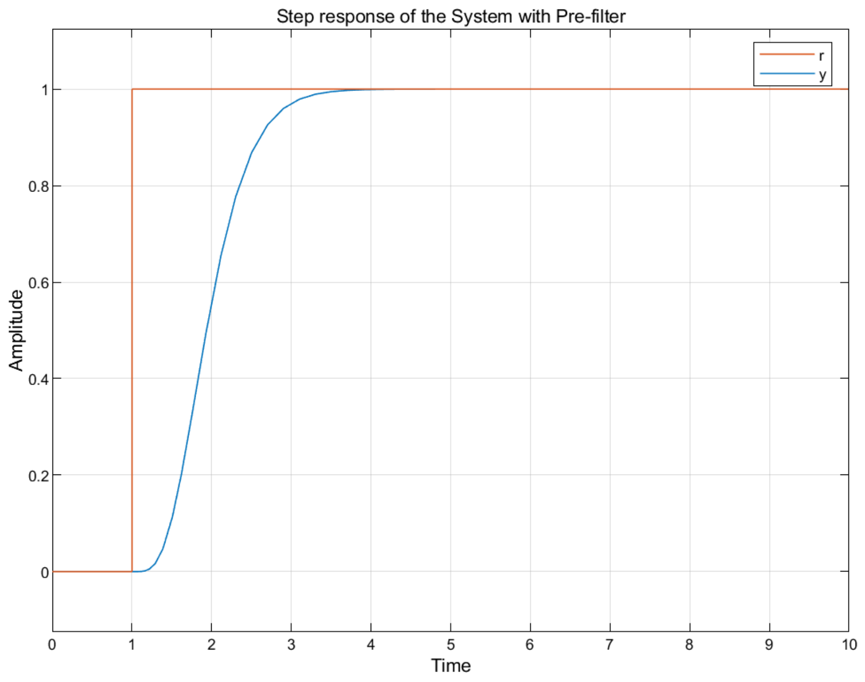


Figure 14 Step response of the system (one vehicle) where r is the input and y is the output (position)

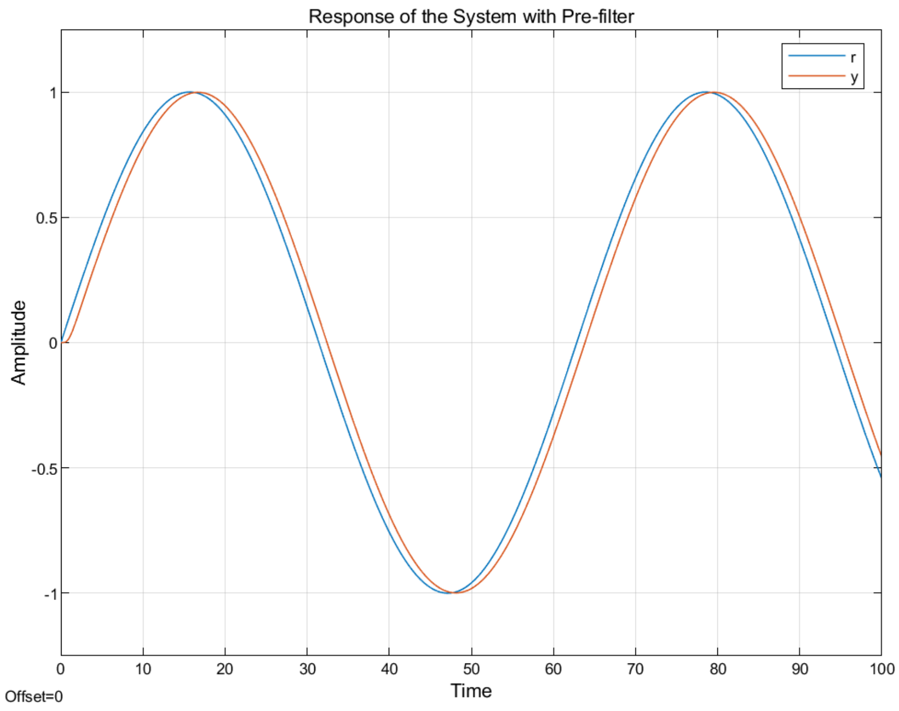


Figure 15 Response of the system to sinusoidal input with period 1 0secs.

As can be seen in the Figure 14-15, the system responds the input quite good in the case of basic form.

- Simulations without disturbances, initial conditions and with the saturation blocks.

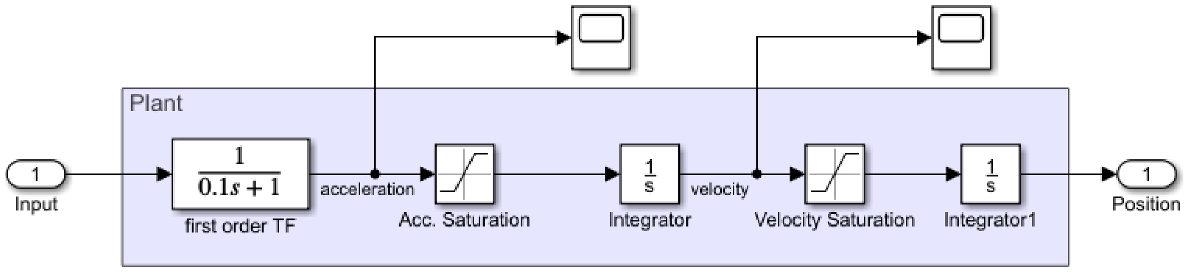
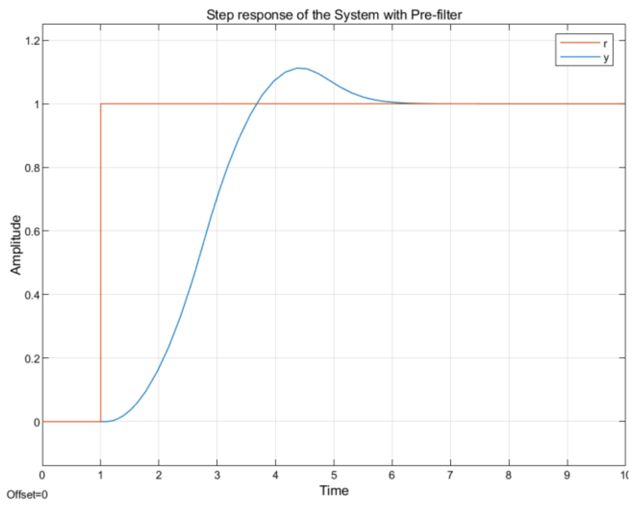


Figure 16 Plant model with saturation blocks (Scopes: to obtain the limits)



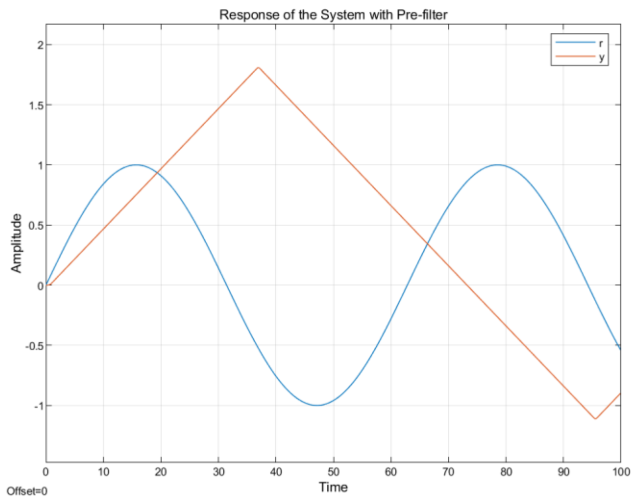


Figure 17 Response of the system to unit step

Figure 18 Response of the system to sinusoidal input

When the saturation blocks were added, behavior of the system get worse. In this case, I chose the limits a little bit strict to show the deformation, but in practice, the boundaries of the saturation blocks can be chosen optimally for both performance and safety.

- Simulations without disturbances, saturation blocks and with the initial conditions.

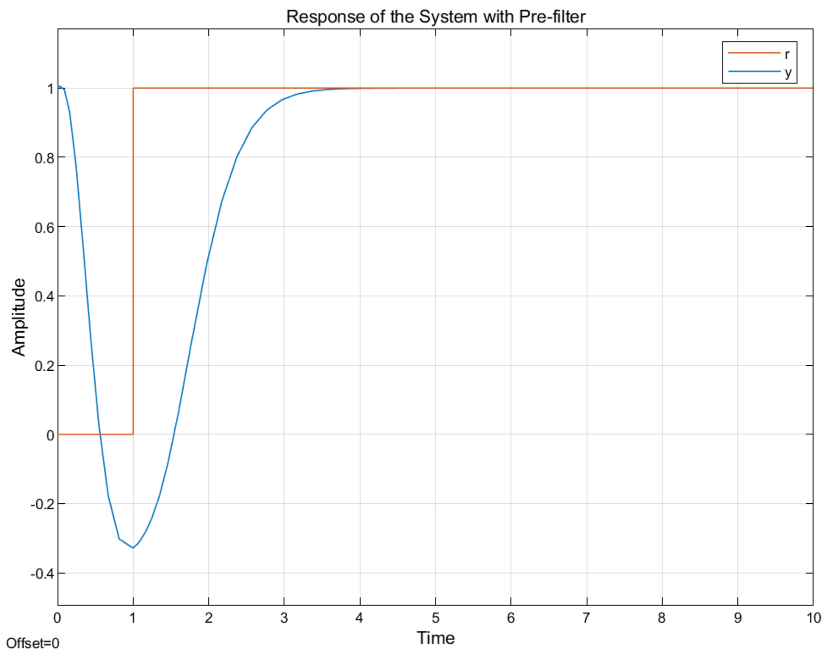


Figure 19 Response of the system to unit step

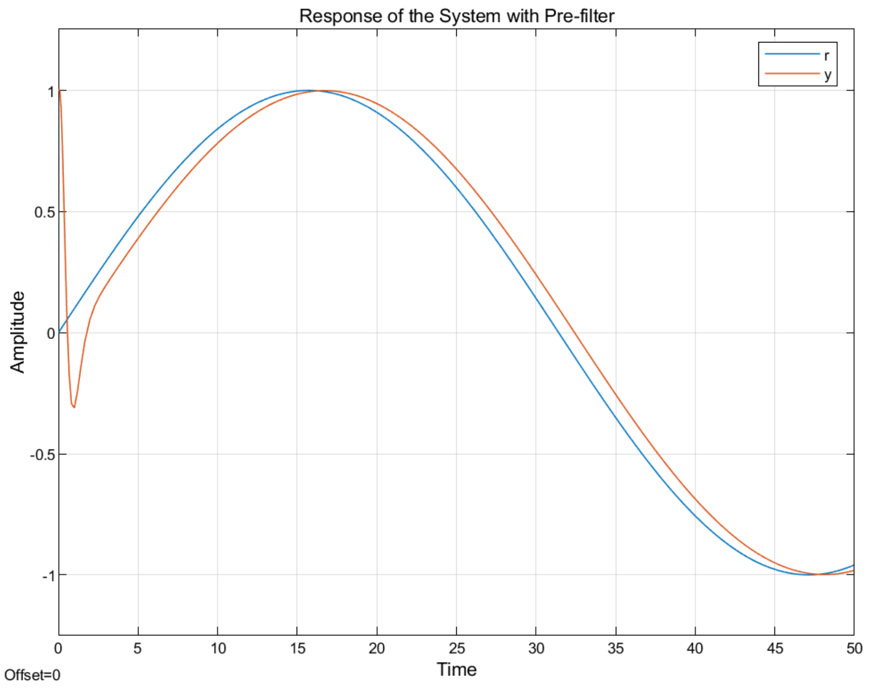


Figure 20 Response of the system to sinusoidal input

The system follows the reference input after some amount of time, i.e., the system can reject the initial conditions.

- Simulations without initial conditions, saturation blocks and with the disturbances for unit step

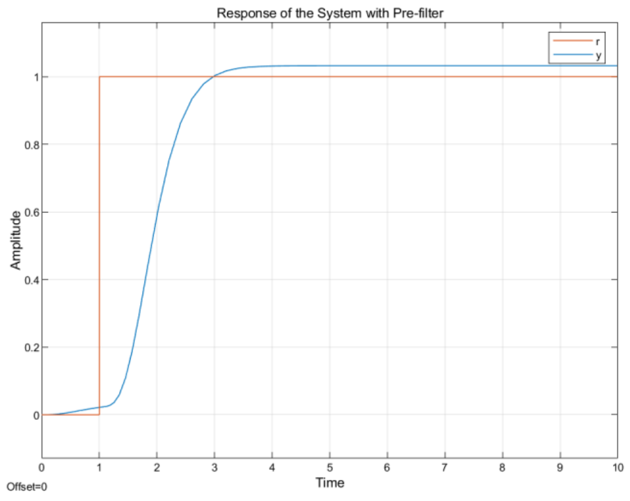


Figure 21 Response when the input disturbance is 0.1

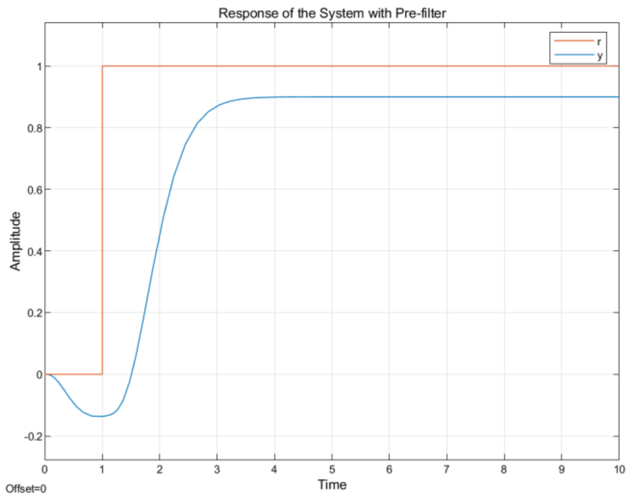


Figure 23 Response when the noise is 0.1

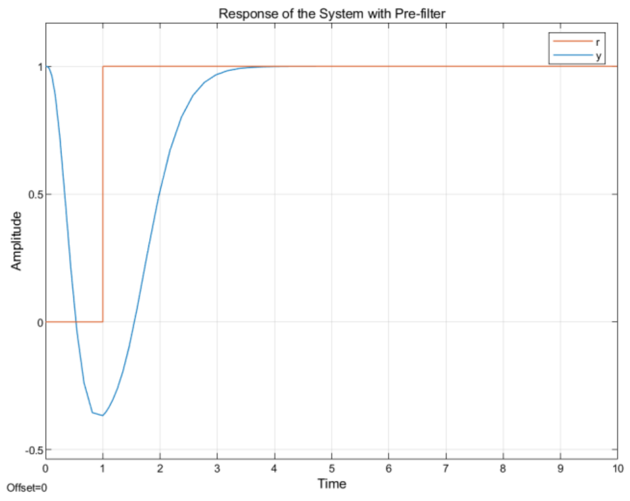


Figure 22 Response when the output disturbance is 1

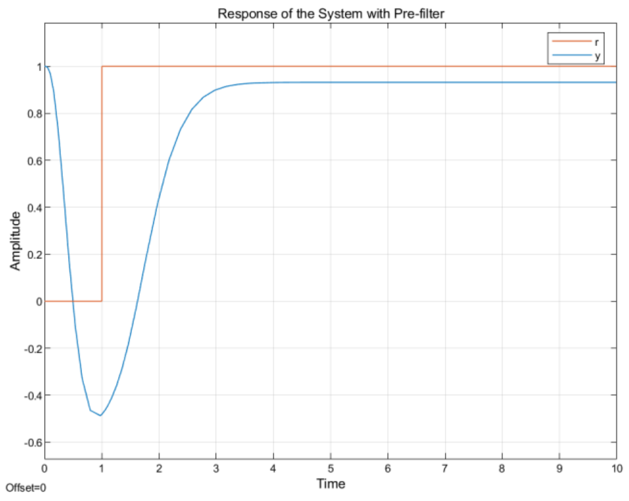


Figure 24 Response when all the disturbances are included

The system rejected the output disturbance well but because of the controller does not have an integrator, the system could not reject the input disturbance. Also, the system is sensitive to the noise and this can be explained from the sensitivity bode plot of the system.

- Simulations without initial conditions, saturation blocks and with the disturbances for sinusoidal input

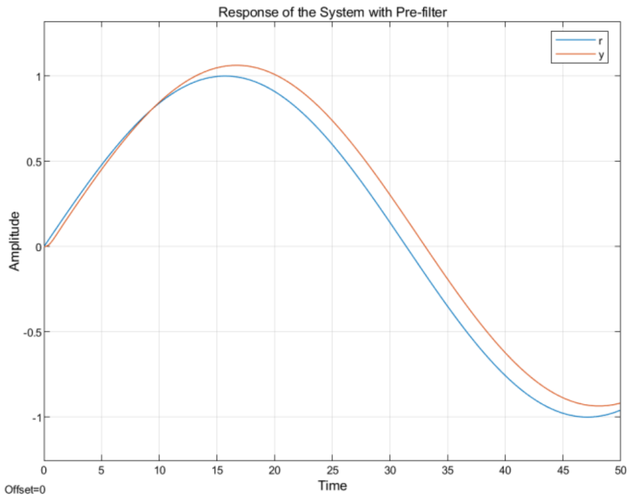


Figure 25 Response when the input disturbance is 0.2

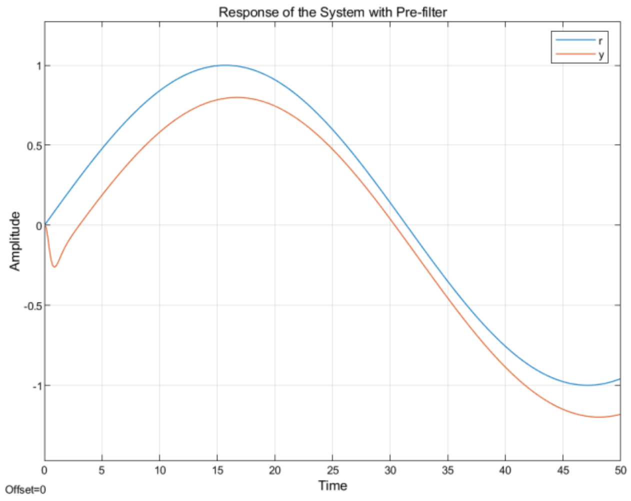


Figure 27 Response when the noise is 0.2

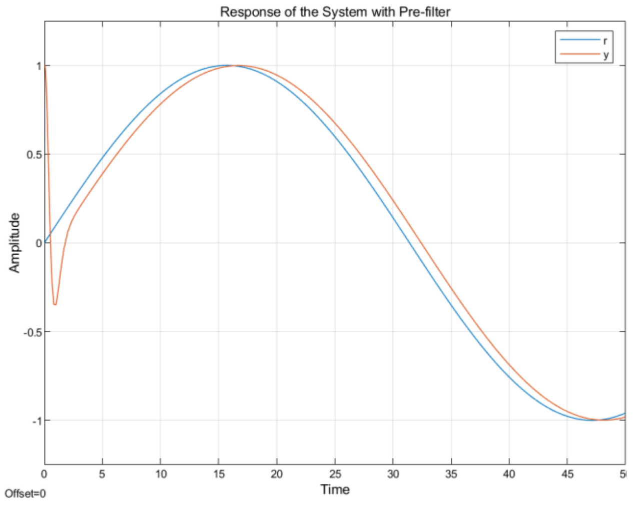


Figure 26 Response when the output disturbance is 1

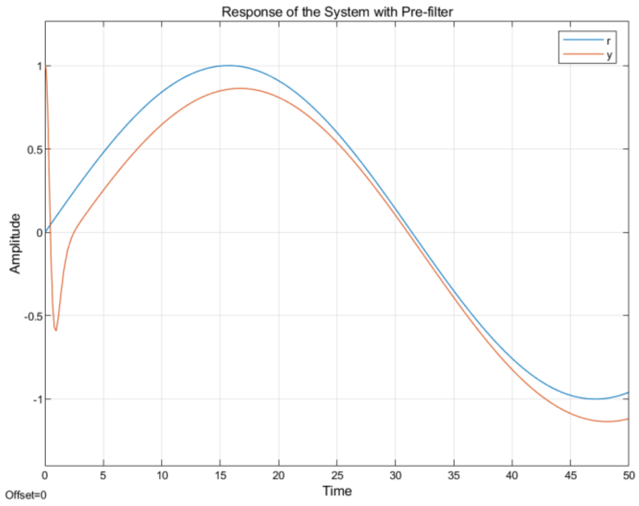


Figure 28 Response when all the disturbances are included

The same results are valid as in the case of the step input.

3) Part 2: Simulation of The Trajectory of Apollo Satellite

In the second part of the project, trajectory of the Apollo satellite was simulated in Simulink for different configuration parameters. The coordinates of the satellite, x and y, were described as the following state space model where $x_1 = x$ and $x_3 = y$. Moreover, it is assumed that the satellite is orbiting around the earth in a plane. The state space model of the satellite can be seen in Equation 38-41 below.

$$\dot{x}_1 = x_2 \tag{38}$$

$$\dot{x}_2 = 2x_4 + x_1 - \mu^* \frac{x_1 + \mu}{\left(\sqrt{(x_1 + \mu)^2 + x_3^2}\right)^3} - \mu \frac{x_1 - \mu^*}{\left(\sqrt{(x_1 - \mu^*)^2 + x_3^2}\right)^3} \tag{39}$$

$$\dot{x}_3 = x_4 \tag{40}$$

$$\dot{x}_4 = -2x_2 + x_3 - \mu^* \frac{x_3}{\left(\sqrt{(x_1 + \mu)^2 + x_3^2}\right)^3} - \mu \frac{x_3}{\left(\sqrt{(x_1 - \mu^*)^2 + x_3^2}\right)^3} \tag{41}$$

where

$$\mu = \frac{1}{82.45}, \qquad \mu^* = 1 - \mu$$

and

$$x_1(0) = 1.2, \qquad x_2(0) = x_3(0) = 0, \qquad x_4(0) = -1.0494$$

For the first part, numerical solutions for the Apollo Satellite will be obtained by built-in solver of the Simulink and for the different parameters of the solver, numerical solutions will be analyzed. For the second part, a solver based on the explicit Runge-Kutta method ($p = 4$) will be implemented for both fixed step sizes and variable step sizes.

3.1) Simulink Model Interpretation

The state space model of the satellite was constructed in Simulink and it can be seen in the Figure X. Also note that the initial conditions of the states were defined in the integrator blocks in Simulink. As a reference, Maximum step size: auto, Relative tolerance: 1e-5 and Solver: ode45 configuration was used. Moreover, the XY plot will be used in order to compare the different variations of the configuration parameters during the section.

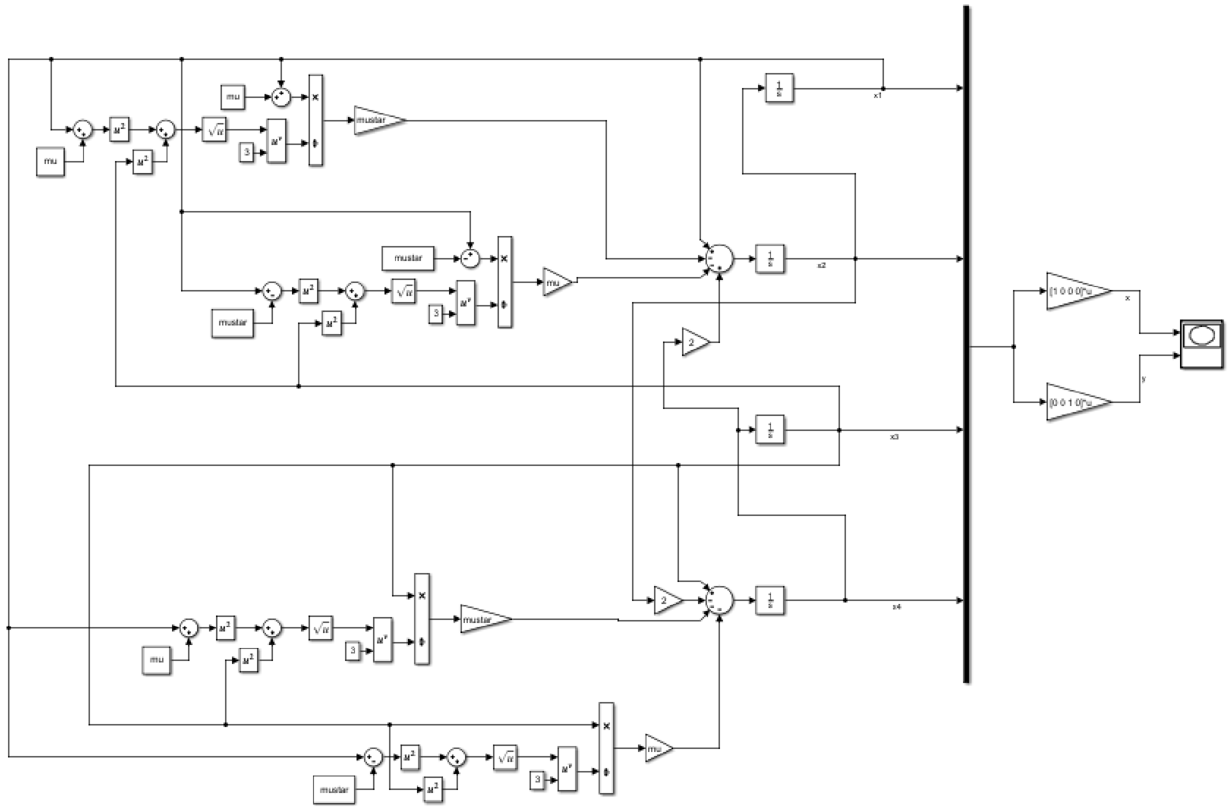


Figure 29 Simulink Model of the Apollo Satellite (can be found Appendix 2 more detailed)

The trajectory of the satellite for a reference configuration can be seen in the Figure 30 below for stop time 10 secs.

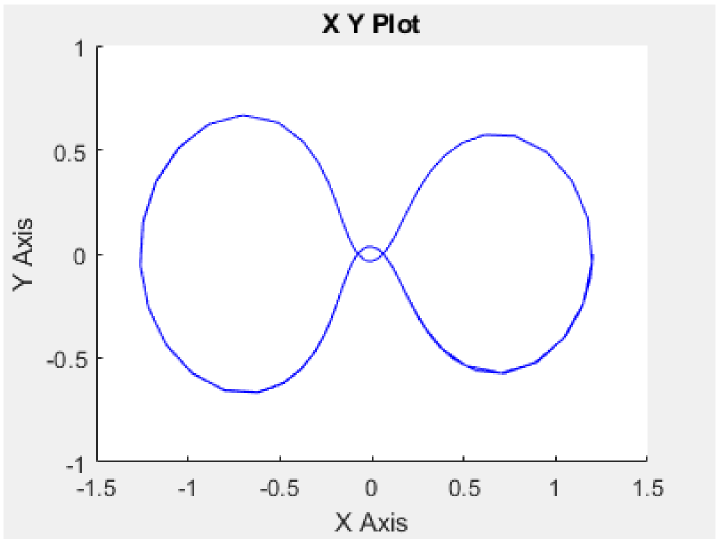


Figure 30 Trajectory of the Apollo Satellite for reference solver

3.1.1) Maximum Step Size

In this section, solver (ode45) and the relative tolerance (1e-5) will be kept constant and by changing the maximum step size, simulation results of the Apollo Satellite will be investigated.

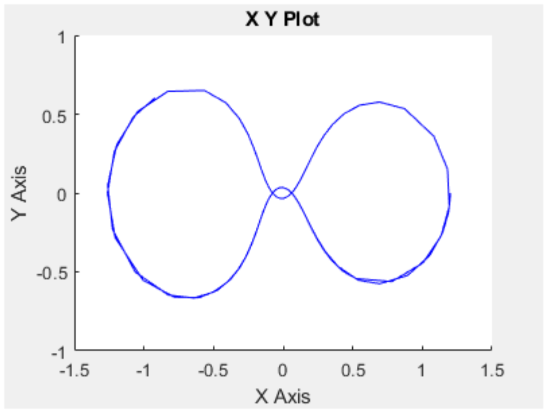


Figure 31 Trajectory when Maximum step size is 1

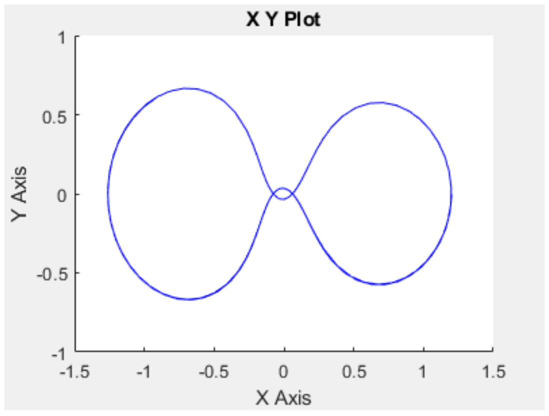


Figure 33 Trajectory when Maximum step size is 0.1

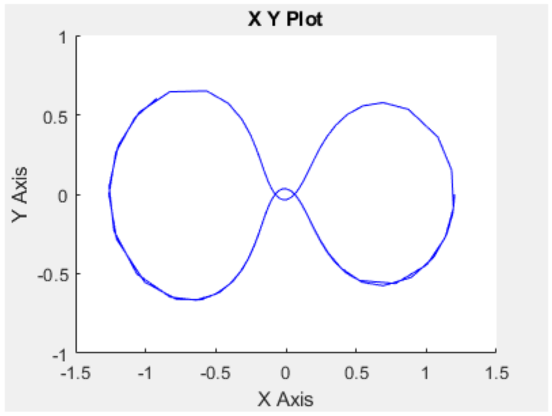


Figure 32 Trajectory when Maximum step size is 0.5

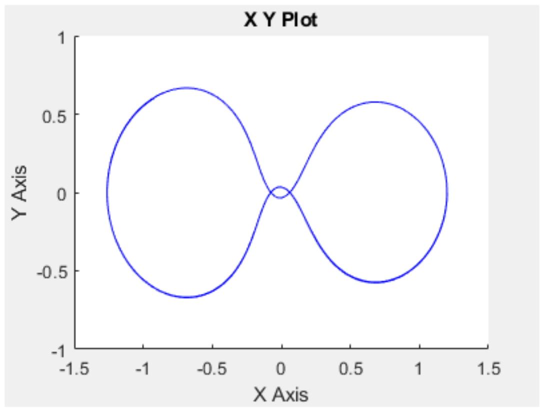


Figure 34 Trajectory when Maximum step size is 0.01

As the maximum step size decreases, quality of the simulation increases. However, simulation time also increases as a drawback. Stability of the simulation is independent from the maximum step size because relative tolerance is small enough. By the way, relative tolerance means that the largest acceptable solver error, relative to the size of each state during each time step. If the relative error exceeds this tolerance, the solver reduces the time step size. In terms of the simulation error, when the maximum step size is increased, solver increases its step size and number of break points decreases. Consequently, the simulation error increases

3.1.2) Relative Tolerance

In this section, solver (ode45) and the maximum step size (auto) will be kept constant and by changing the relative tolerance, simulation results of the Apollo Satellite will be investigated.

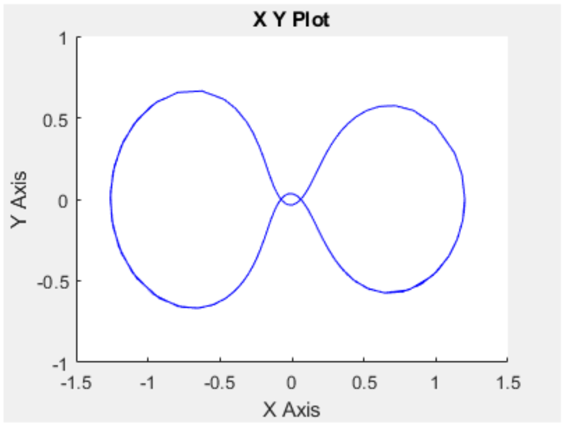


Figure 35 Trajectory when Relative tolerance is 1e-6

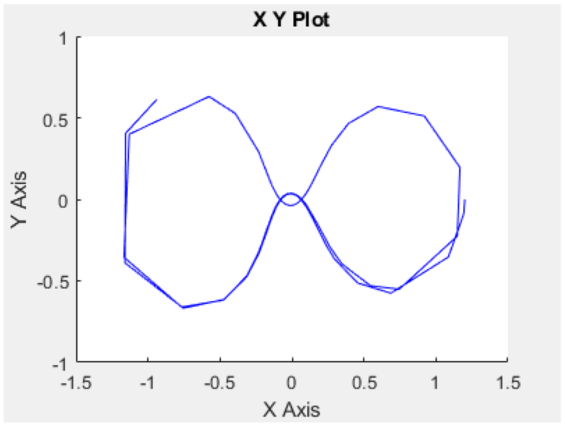


Figure 37 Trajectory when Relative tolerance is 1e-3

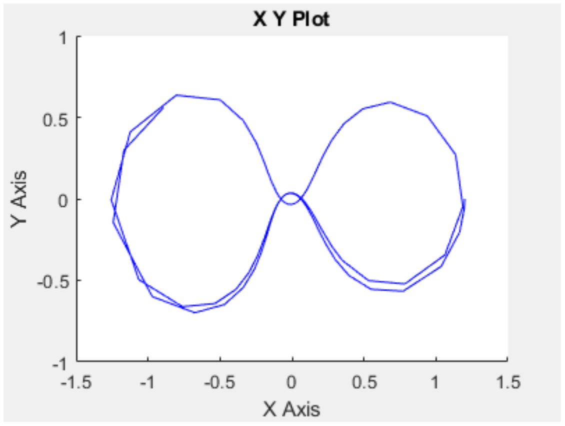


Figure 36 Trajectory when Relative tolerance is 1e-4

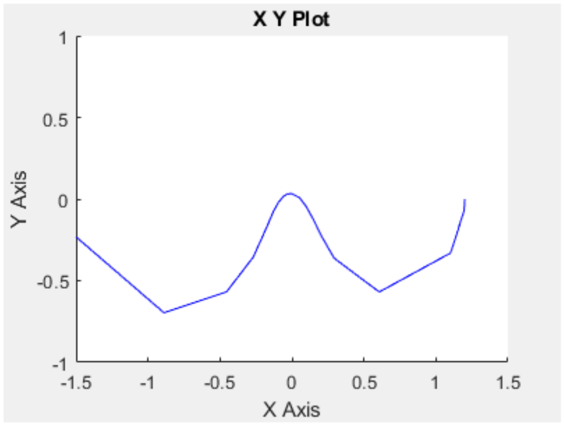


Figure 38 Trajectory when Relative tolerance is 1e-2

Because of the relative tolerance is the largest acceptable solver error, as the tolerance increases (in magnitude), number of break points decreases and after some point, simulation becomes unstable. Moreover, the simulation time increases as the relative tolerance decreases because of the increase of the number of break points.

3.1.3) Solver

In this section, end time (30 secs), maximum step size(auto) and the relative tolerance (5e-4 for stability, 3e-4 for simulation error) will be kept constant and by changing the solver type for variable-step size, simulation results of the Apollo Satellite will be investigated in terms of stability and simulation error.

3.1.3.1) Stability

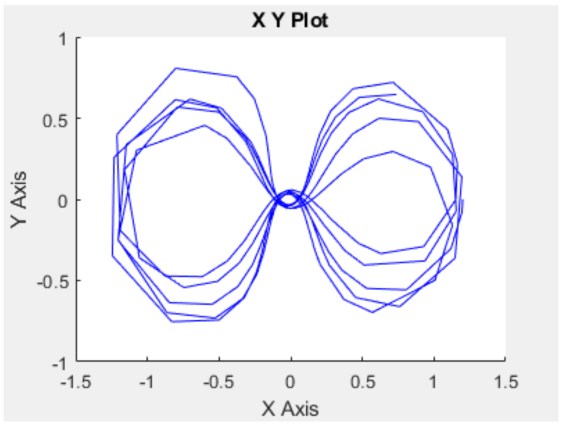


Figure 39 Trajectory when the solver is ode45

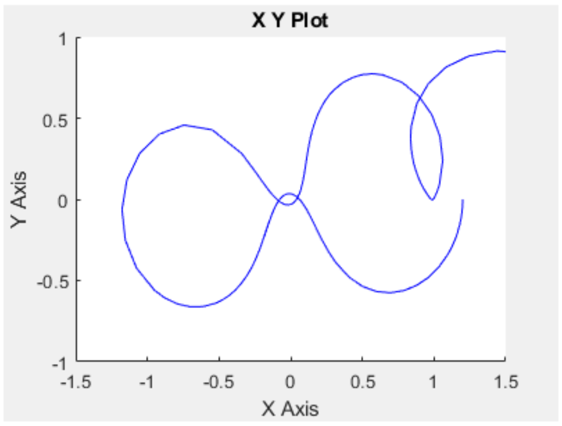


Figure 41 Trajectory when the solver is ode113

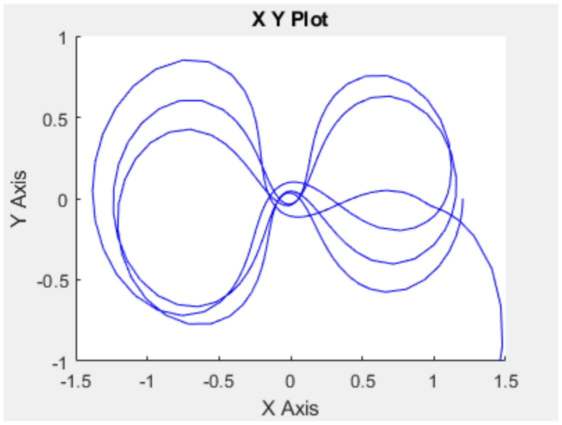


Figure 40 Trajectory when the solver is ode23

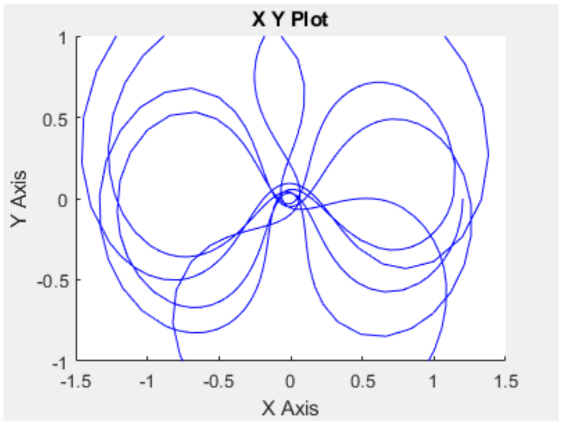


Figure 42 Trajectory when the solver is ode15s

In terms of the stability, ode45 gives the best result among the variable step size solvers. Actually, other solvers give unstable results for the determined relative tolerance. After decreasing relative tolerance, stability problem was disappeared.

3.1.3.2) Simulation Error

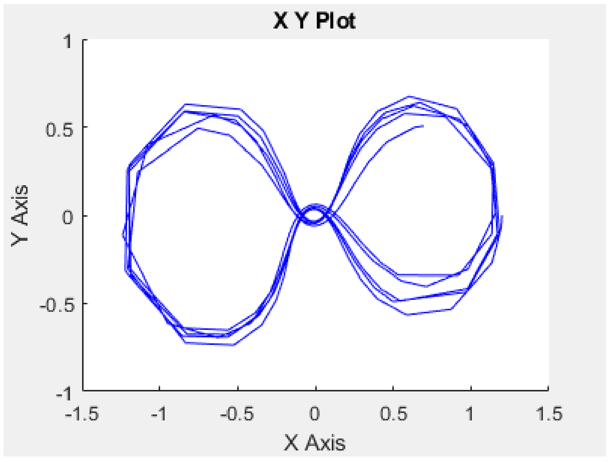


Figure 43 Trajectory when the solver is ode45

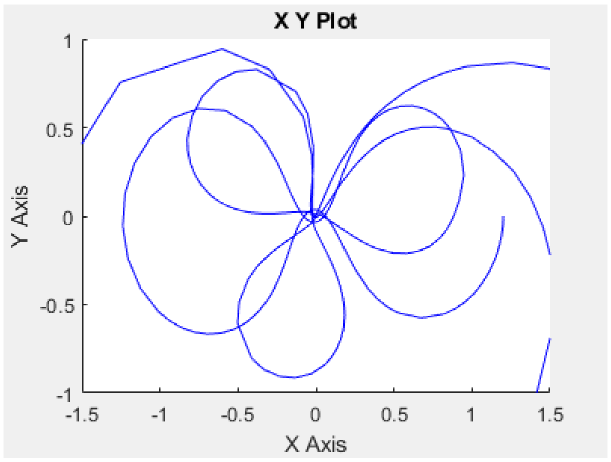


Figure 45 Trajectory when the solver is ode113

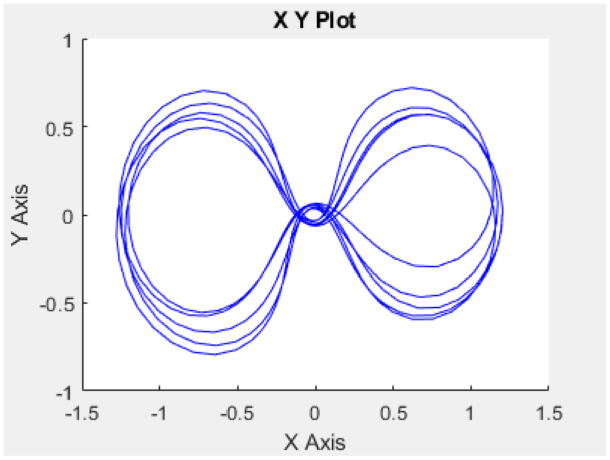


Figure 44 Trajectory when the solver is ode23

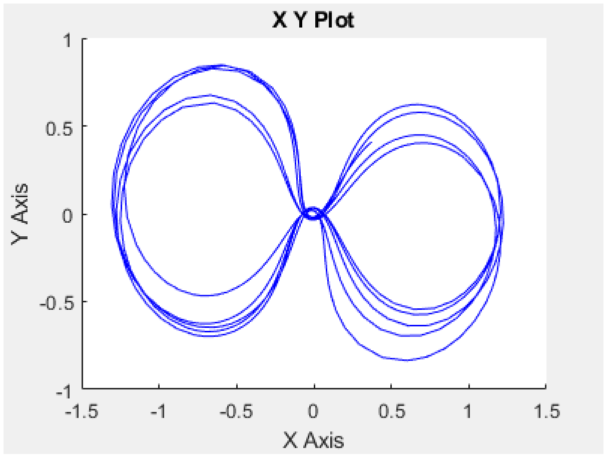


Figure 46 Trajectory when the solver is ode15s

In terms of simulation error, results of ode23 and ode15s quite good and ode45 follows them. The number of break points of ode45 is less than the other solvers, so the simulation error is obvious. On the other hand, ode113 gives the unstable result and we can conclude that this solver is not suitable for the Apollo Satellite.

3.1.4) Fixed Step Size

In this section, end time (15 secs) will be kept constant and by changing the step size for fixed-step size, simulation results of the Apollo Satellite will be investigated.

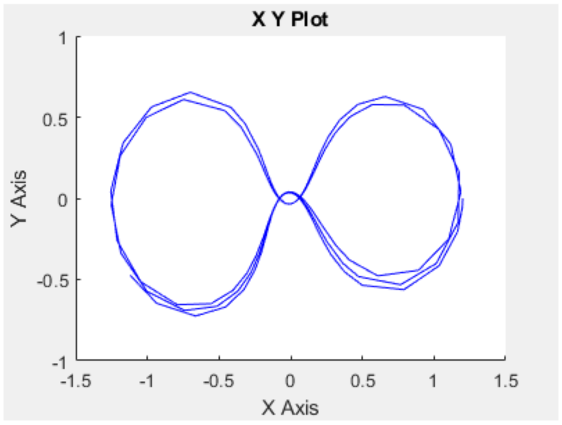


Figure 47 Trajectory when the solver is ode45 (variable step size) and relative tolerance is 1e-4

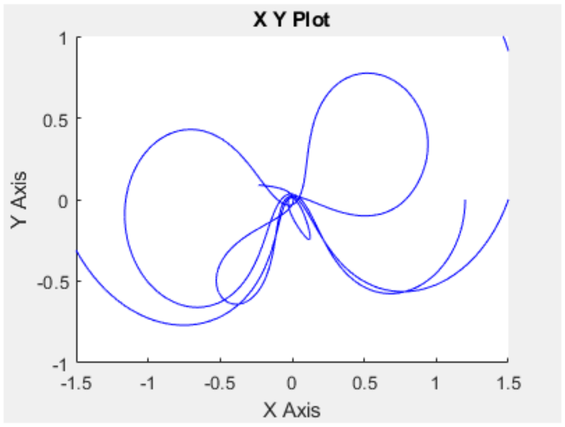


Figure 49 Trajectory for fixed step size solver when the step size is 2e-3

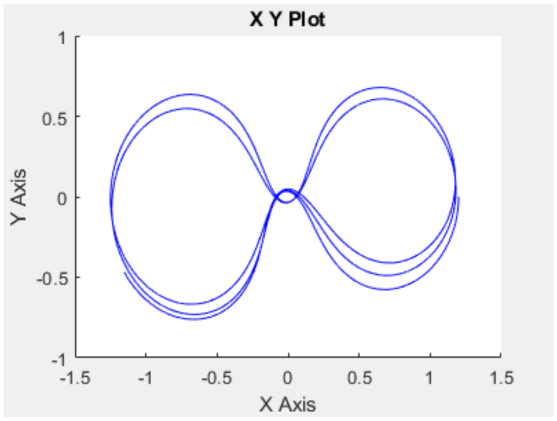


Figure 48 Trajectory for fixed step size solver when the step size is 1e-3

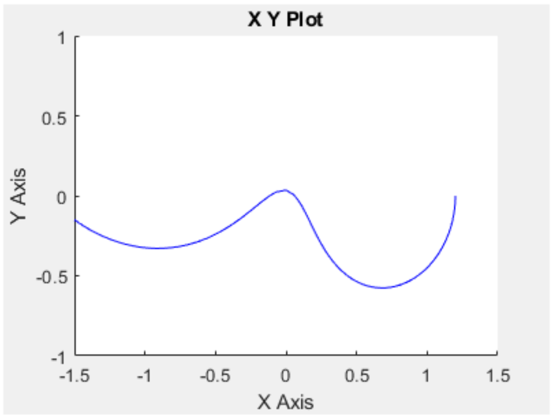


Figure 50 Trajectory for fixed step size solver when the step size is 1e-2

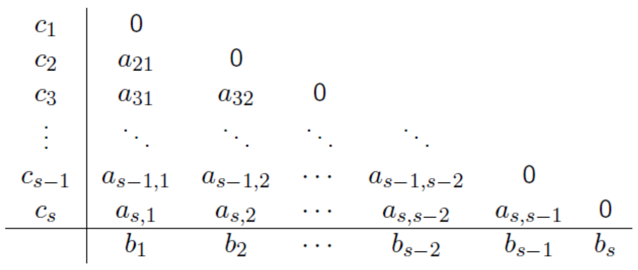
For the fixed step size, speed of the simulation was decreased significantly as expected because for the variable step size case, Simulink adjust the step sizes according to the solver type. For the important simulations, fixed step size can be used because it is very precise. However, using variable step size is more meaningful because suitable solver for the systems can be found by adjusting the configuration parameters and someone can save lots of time.

3.2) Explicit Runge-Kutta Method Implementation

In this part, a solver based on the explicit Runge-Kutta method (p=4) will be designed and implemented for both fixed and variable step sizes.

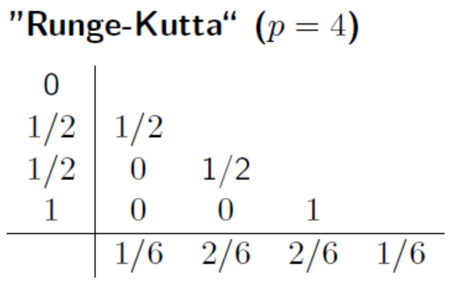
3.2.1) Fixed Step Size

In order to implement the Runge-Kutta method for fixed step size let us construct a function as in the last assignment of the course, but before this, we need to determine coefficients of the Butcher Tableau. The following figures are taken from the lecture notes and the coefficients will be determined in this manner.



$$\begin{array}{c|cccccc}
c_1 & 0 & & & & & \\
c_2 & a_{21} & 0 & & & & \\
c_3 & a_{31} & a_{32} & 0 & & & \\
\vdots & \ddots & \ddots & \ddots & \ddots & & \\
c_{s-1} & a_{s-1,1} & a_{s-1,2} & \cdots & a_{s-1,s-2} & 0 & \\
c_s & a_{s,1} & a_{s,2} & \cdots & a_{s,s-2} & a_{s,s-1} & 0 \\
\hline
 & b_1 & b_2 & \cdots & b_{s-2} & b_{s-1} & b_s
\end{array}$$

Figure 51 Coefficients of Explicit Runge-Kutta Method in a Table



"Runge-Kutta" $(p = 4)$

$$\begin{array}{c|cccc}
0 & & & & \\
1/2 & 1/2 & & & \\
1/2 & 0 & 1/2 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & 1/6 & 2/6 & 2/6 & 1/6
\end{array}$$

Figure 52 Coefficients of Explicit Runge-Kutta Method when p=4

Then the Runge-Kutta method can be written as

$$\Phi(t_i, x_i, h_i) = b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4 \tag{42}$$

$$k_1 = f(t_i, x_i) \tag{43}$$

$$k_2 = f(t_i + c_2 h_i, x_i + h_i a_{21} k_1) \tag{44}$$

$$k_3 = f(t_i + c_3 h_i, x_i + h_i a_{31} k_1 + h_i a_{32} k_2) \tag{45}$$

$$k_4 = f(t_i + c_4 h_i, x_i + h_i a_{41} k_1 + h_i a_{42} k_2 + h_i a_{43} k_3) \tag{46}$$

and the iteration can be written as

$$x_{i+1} = x_i + h_i \Phi(t_i, x_i, h_i) \tag{47}$$

After 8 nonlinear equations were solved, the resulting coefficients can be seen in Figure 52 above.

With the help of last assignment of the course, the necessary MATLAB code was created, and it can be seen in the Appendix 3. The execution time of the code was 0.3114 sec (Actually changes every time according to the condition of the computer), and the resulting satellite trajectory can be seen in the Figure 53 below. Simulation parameters also have been chosen as in the last assignment which are tend=10 sec (final time) and h=0.001 (suitable value for the step size).

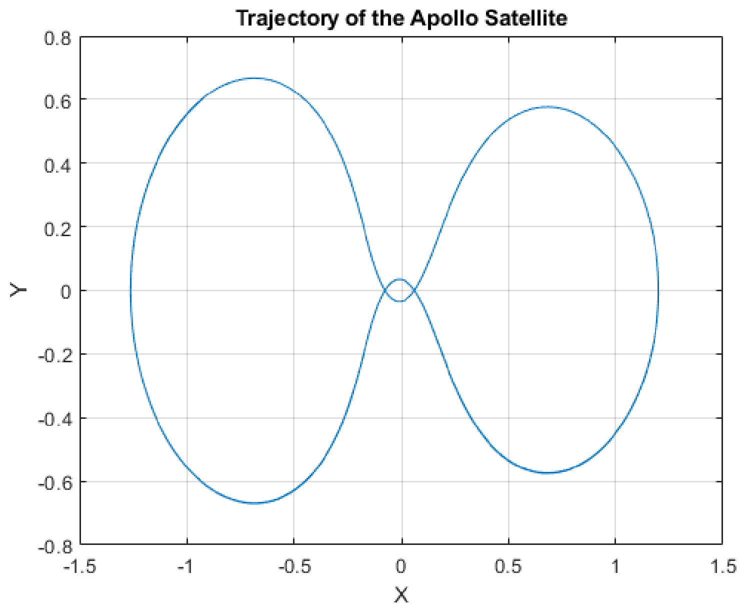


Figure 53 Trajectory for explicit Runge-Kutta method (p=4) when fixed step size is used

Also, it is good to mention about that when the step size increased, the simulation gives an unstable result as in the Figure 54.

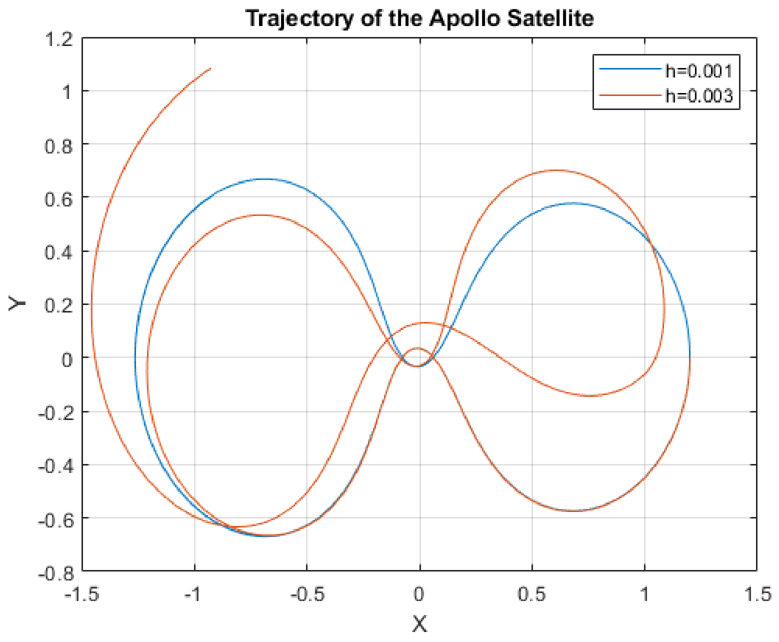


Figure 54 Comparison of the step sizes

3.2.2) Variable Step Size

In order to construct the Runge-Kutta method for variable step size, the key point is that the local error must satisfy the desired tolerance step at each step. If the local error becomes higher than the specified tolerance, we need to repeat the process with a smaller step size so that the local error tolerance is not violated.

To construct an algorithm, the inputs are can be considered as the initial time t_i, final time t_f, the initial condition $y(t_i)$, initial step size h_0, absolute error tolerance $atol \epsilon \mathbb{R}^n$, relative error tolerance $rtol \epsilon \mathbb{R}^n$, step size adjustment parameters $0 < \beta < 1$, $fac_1 > fac_0 > 0$, the minimum allowable step size h_{min}, and the maximum allowable number of iterations $MAX\_ITER > 0$. Moreover, the output can be considered as $y(t_f)$.

The main implementation steps that needed for a solver with variable step sizes are can be listed as

1. Make sure the system is at the initial step
2. Determine and assign the initial conditions for t and h as $t = t_i$ and $h = h_0$
3. Assign the states
4. If $t + h > t_f$, then assign $h = t_f - t$
5. Start the iteration from 0 to MAX_ITER
    a. Calculate the next step by the `ExplicitRungeKuttaFixedSize`
    b. Estimate the error by using suitable explicit method of order 3 error estimator
    c. If the local error is higher than the relative error tolerance
        i. Increase the current step size and continue
    d. If the local error lower than the relative error tolerance
        i. End the iteration and forward the value of h and states
6. Forward the step size and states

According to this algorithm, the necessary MATLAB code was created, and it can be seen in the Appendix 4. Notice that I could not find an error estimator which is order 3. Instead I have used sixth order error estimator, I hope it does not constitute a problem.

After testing the solver, resulting trajectory can be seen in the Figure 55 below. Computation time was computed for a single period and for the variable step size, it is the 0.127 secs, which was previously 0.3114 secs. In this case, the computation time was decreased as expected for the numerical solution because in the variable step size case, the code scales the step size according to the error during the execution time. For this case, break points are obvious because number of break points were decreased.

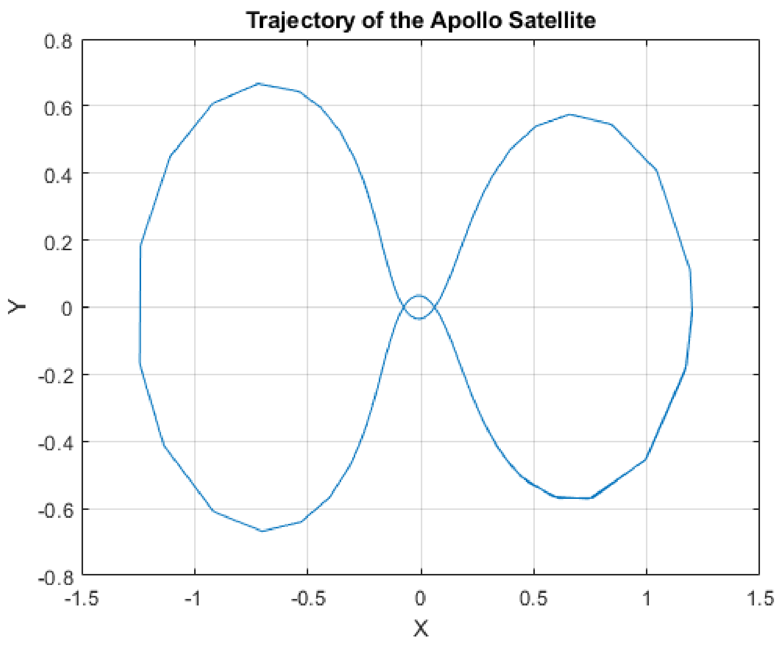


Figure 55 Trajectory for explicit Runge-Kutta method (p=4) when variable step size is used

To observe the step sizes, let us plot for the variable step size case. The resulting plot can be seen in the Figure 56 below.

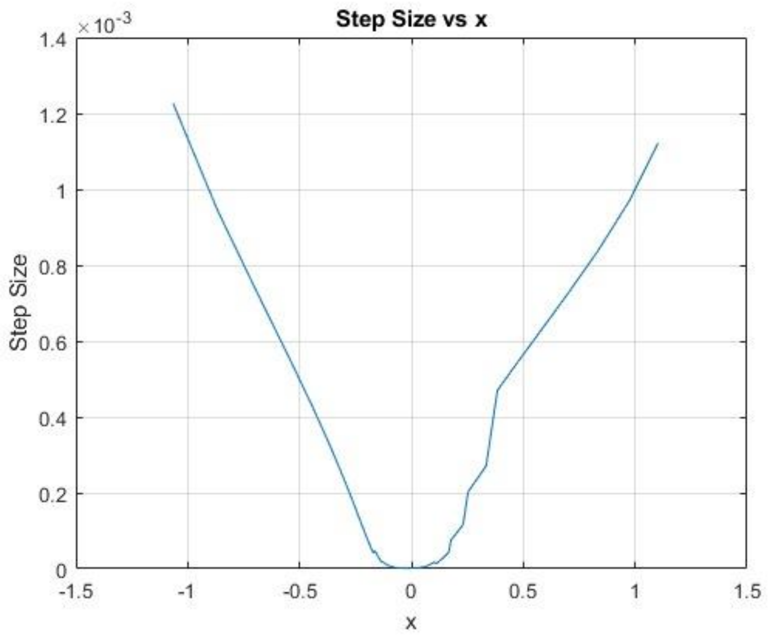


Figure 56 Plot of the step size vs x

The step size varies during the simulation and when the error was higher, magnitude of the step size was decreased to make the error less than the relative tolerance. According to the Figure 56, step size increases after getting further from origin because the error was decreased on the boundaries of the trajectory.

4) Conclusions

At the end of the project, I have gained lots of experience about understanding fundamental limitations for the design of control systems and application of the controller design methods. Moreover, under favor of the second part of the project, I had the chance of perform one of the famous simulation methods which is Runge-Kutta method.

In the first part of the project, platooning technology has expand my horizon very much about the control system engineering and I have seen many modern control system articles about this subject. Because of it is very new technology, researching of this subject was very enjoyable for me. Furthermore, simulation of the platooning system helped me to understand the concept better.

For the second part of the project, I can say that it was much harder than it seemed. Especially in the variable step size part, I had lots of difficulty, but all in all, I understand the working principle of Explicit Runge-Kutta method better.

I have to say that, I made great use of the lecture notes and they helped me a lot. Also, previous assignments of the course helped me particularly in the second part of the project.

Lastly, I would like to thank the course assistant Mehmet Çetinkaya for the patience and feedbacks to the homework and specially I would like to thank Klaus Verner Schmidt for his support and guidance.

5) References

- Al-Jhayyish, A. M. H., Schmidt, K.W: Feedforward Strategies for Cooperative Adaptive Cruise Control in Heterogeneous Vehicle Strings, IEEE Transactions on Intelligent Transportation Systems, Special Issue on "Applications and Systems for Collaborative Driving", vol. 19, no. 1, pp. 113-122, 2018.
- Deng, Q. (2016). Heavy-Duty Vehicle Platooning : Modeling and Analysis (Licentiate dissertation). Stockholm. Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-181934
- J. Ploeg, B. T. M. Scheepers, E. van Nunen, N. van de Wouw, and H. Nijmeijer, "Design and experimental evaluation of cooperative adaptive cruise control," in Proc. 14th Int. IEEE Conf. Intell. Transp. Syst., Oct. 2011, pp. 260–265.
- J. Ploeg, N. van de Wouw, and H. Nijmeijer, "Lp string stability of cascaded systems: Application to vehicle platooning," IEEE Trans. Control Syst. Technol., vol. 22, no. 2, pp. 786–793, Mar. 2014.
- Roshanghias, D. (2017). Evaluation and Implementation of a Longitudinal Control in a Platoon of Radio Controlled Vehicles (Dissertation). Retrieved from http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-212564
- R. Rajamani, Vehicle Dynamics and Control (Mechanical Engineering Series), 2nd ed. New York, NY, USA: Springer, 2012.

6) Appendices

6.1) Appendix A: MATLAB Script of the Part 1

```
%{
Rafet KAVAK - 2166783
EE498 - Control System Design and Simulation
Term Project - Part 1
%}
clc, clear, close all;
```

a.

```
tau = 0.1; % sec

s = tf('s');
G = 1/(s^2*(tau*s+1));
% Necessary parts can be uncommented.
% figure
% bode(G);
% grid on;
```

b.

Kp = 0.1; Ti = 10; Td = 10;

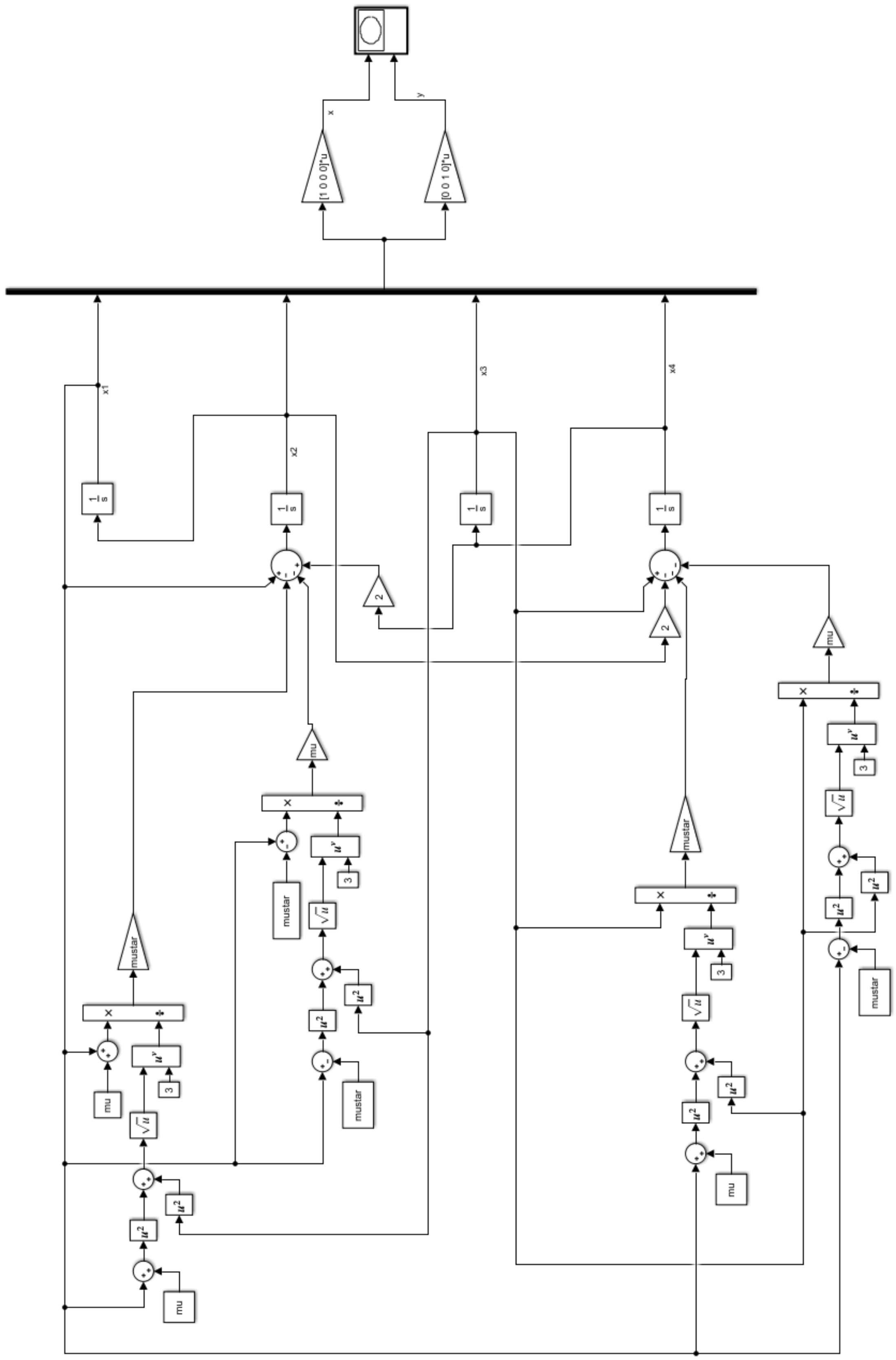
```
% P
% C = Kp
%
% figure
% bode(G)
% grid on;
% hold on
% bode(C*G);
% legend('G','C*G')



% PI
% C = Kp*(1+1/(s*Ti));
%
% figure
% bode(G)
% grid on;
% hold on
% bode(C*G);
% legend('G','C*G')



% PD
% C = Kp*(1+s*Td);
%
%
% figure
% bode(G)
% grid on;
% hold on
% bode(C*G);
% legend('G','C*G')

% PID

% C = Kp*(1+1/(s*Ti)+s*Td);
%
%
% figure
% bode(G)
% grid on;
% hold on
% bode(C*G);
% legend('G','C*G')
```

c.

```
syms x

C1 = (1 + 5*s - 590*s^2)/(600 - 50*s + 10*s^2);

T1 = feedback(G*C1, 1);

% figure
% step(T1);
% title('Step Response when s = -1');
% grid on;
% hold on
% fplot(heaviside(x), 'g')
% legend('Output','Input')


C2 = (32 + 80*s - 320*s^2)/(400 + 0*s + 10*s^2);

T2 = feedback(G*C2, 1);

% figure
% step(T2);
% title('Step Response when s = -2');
% grid on;
% hold on
% fplot(heaviside(x), 'g')
% legend('Output','Input')
```

```
C3 = (243 + 405*s - 130*s^2)/(400 + 50*s + 10*s^2);

T3 = feedback(G*C3, 1);
%
% figure
% step(T3);
% title('Step Response when s = -3');
% grid on;


C4 = (1024 + 1280*s + 40*s^2)/(600 + 100*s + 10*s^2);

T4 = feedback(G*C4, 1);

% figure
% step(T4);
% title('Step Response when s = -4');
% grid on;
% hold on
% fplot(heaviside(x), 'g')
% legend('Output','Input')
% hold off

C5 = (3125 + 3125*s + 250*s^2)/(1000 + 150*s + 10*s^2);

T5 = feedback(G*C5, 1);

% figure
% step(T5);
% title('Step Response when s = -5');
% grid on;
% hold on
% fplot(heaviside(x), 'g')
% legend('Output','Input')
% hold off


C6 = (7776 + 6480*s + 560*s^2)/(1600 + 200*s + 10*s^2);

T6 = feedback(G*C6, 1);

% figure
% step(T6);
% title('Step Response when s = -6');
% grid on;
% hold on
% fplot(heaviside(x), 'g')
% legend('Output','Input')
% hold off

% figure
% step(T1)
% hold on
% step(T2)
% step(T3)
% step(T4)
% step(T5)
% step(T6)
% fplot(heaviside(x), 'g')
% hold off
% legend('s = -1', 's = -2', 's = -3', 's = -4', 's = -5', 's = -6', 'Input')
% grid on

% figure
% step(T2)
% hold on
% step(T3)
% step(T4)
% step(T5)
% step(T6)
% fplot(heaviside(x), 'g')
% hold off
% title('Step Responses for Several Pole Locations');
% legend('s = -2', 's = -3', 's = -4', 's = -5', 's = -6', 'Input')
% grid on


% figure
% step(T3)
% hold on
% step(T4)
% step(T5)
% step(T6)
% fplot(heaviside(x), 'g')
% hold off
% title('Step Responses for Several Pole Locations');
% legend('s = -3', 's = -4', 's = -5', 's = -6', 'Input')
% grid on

F = 12.5/((s+1.09611)*(s+11.40388)); % Pre-filter

% figure
% step(F*T5)
% hold on
% fplot(heaviside(x), 'g')
% hold off
% title('Step Response for s = -5 with Pre-filter');
% legend('Output', 'Input')
% grid on
```

6.2) Appendix B: Simulink Model of Apollo Satellite



6.3) Appendix C: MATLAB Script of the Part 2b – Fixed Step Size

```
%{
Rafet KAVAK - 2166783
EE498 - Control System Design and Simulation
Term Project - Part 2b.
Runge-Kutta for a fixed step size
%}
clc, clear, close all;

tStart = tic;

mu = 1/82.45;
mustar = 1 - mu;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicit Runge-Kutta method (p=4) for the Apollo Satellite
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f1 = @(t,x)(x(2));
f2 = @(t,x)(2*x(4) + x(1) - mustar*((x(1)+mu)/(sqrt((x(1)+mu)^2+x(3)^2))^3) - mu*((x(1)-
mustar)/(sqrt((x(1)-mustar)^2+x(3)^2))^3));
f3 = @(t,x)(x(4));
f4 = @(t,x)(-2*x(2) + x(3) - mustar*((x(3))/(sqrt((x(1)+mu)^2+x(3)^2))^3) -
mu*((x(3))/(sqrt((x(1)-mustar)^2+x(3)^2))^3));

f  = @(t,x)([f1(t,x);f2(t,x);f3(t,x);f4(t,x)]);
x0 = [1.2; 0; 0; -1.0494];

% =======================
% Simulation
% =======================
tend = 10; % final time

h = 0.001; % small step size; this is very close to the actual solution
x(:,1) = x0; % state values during the simulation
time(1) = 0; % simulation time

for kk = 1:tend/h
    x(:,kk+1) = ExplicitRungeKuttaFixedSize(time(kk),x(:,kk),h,f);
    time(kk+1) = kk*h;
end

figure;
plot(x(1,:),x(3,:));
xlabel('X')
ylabel('Y')
title('Trajectory of the Apollo Satellite')
grid on;
hold all;

% % Different step size
% h = 0.003; % larger step size
% clear x time;
% x(:,1) = x0; % state values during the simulation
% time(1) = 0; % simulation time
% for kk = 1:tend/h
%     x(:,kk+1) = ExplicitRungeKuttaFixedSize(time(kk),x(:,kk),h,f);
%     time(kk+1) = kk*h;
% end
%
% plot(x(1,:),x(3,:));
% xlabel('X')
% ylabel('Y')
% title('Trajectory of the Apollo Satellite')
% grid on;
% legend('h=0.001', 'h=0.003')

tEnd = toc(tStart)

function xNext = ExplicitRungeKuttaFixedSize(t,x,h,f)
% =============================
% Function computes one step
% Inputs
% - x: current state
% - h: step size
% - f: right hand side of the IVP to be simulated (given as function
% handle): https://www.mathworks.com/help/matlab/matlab_prog/pass-a-function-to-another-
function.html
% Output
% - xNext: state in the next time step

% Explicit Runge-Kutta Method with 4 Stages
c1 = 0;
c2 = 1/2;
c3 = 1/2;
c4 = 1;

b1 = 1/6;
b2 = 2/6;
b3 = 2/6;
b4 = 1/6;

a21 = 1/2;
a31 = 0;
a32 = 1/2;
a41 = 0;
a42 = 0;
a43 = 1;

k1 = f(t, x);
k2 = f(t + c2*h, x + h*a21*k1);
k3 = f(t + c3*h, x + h*a31*k1 + h*a32*k2);
k4 = f(t + c4*h, x + h*a41*k1 + h*a42*k2 + h*a43*k3);

xNext = x + h*(b1*k1 + b2*k2 + b3*k3 + b4*k4); % Iteration
end
```

6.3) Appendix D: MATLAB Script of the Part 2b – Variable Step Size

```
%{
Rafet KAVAK - 2166783
EE498 - Control System Design and Simulation
Term Project - Part 2b.
Runge-Kutta for a variable step size
%}
clc, clear, close all;

tStart = tic;

mu = 1/82.45;
mustar = 1-mu;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicit Runge-Kutta method (p=4) for the Apollo Satellite
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f1 = @(t,x)(x(2));
f2 = @(t,x)(2*x(4) + x(1) - mustar*((x(1)+mu)/(sqrt((x(1)+mu)^2+x(3)^2))^3) - mu*((x(1)-
mustar)/(sqrt((x(1)-mustar)^2+x(3)^2))^3));
f3 = @(t,x)(x(4));
f4 = @(t,x)(-2*x(2) + x(3) - mustar*((x(3))/(sqrt((x(1)+mu)^2+x(3)^2))^3) -
mu*((x(3))/(sqrt((x(1)-mustar)^2+x(3)^2))^3));

f = @(t,x)([f1(t,x);f2(t,x);f3(t,x);f4(t,x)]);

x0 = [1.2; 0; 0; -1.0494];

% =======================
% Simulation
% =======================
tend = 0.25; % final time

h = 0.001; % small step size; this is very close to the actual solution
hmax = 10;
MAX_ITER = 20;
rtol = 1e-7;
tspan = 0:h:tend;

clear x time;
t0 = 0;
alpha = 1e-2; % alpha < 1
beta = 1.5e0; % beta  > 1

[x time step] = ExplicitRungeKuttaVariableSize(t0,tend,x0,h,f,MAX_ITER,rtol,hmax,alpha,beta);

figure;
plot(x(1,:),x(3,:));
xlabel('X')
ylabel('Y')
title('Trajectory of the Apollo Satellite')
grid on;
```

```
hold all;

% % Different step size
% h = 0.003; % larger step size
% clear x time;
% t0 = 0;
% [x time] = ExplicitRungeKuttaVariableSize(t0,tend,x0,h,f,MAX_ITER,rtol,hmax);
%
% plot(x(1,:),x(3,:));
% xlabel('X')
% ylabel('Y')
% title('Trajectory of the Apollo Satellite')
% grid on;
% legend('h=0.001', 'h=0.003')

time(end)
tEnd = toc(tStart)

function [x time step] =
ExplicitRungeKuttaVariableSize(t0,tend,x0,h,f,MAX_ITER,rtol,hmax,alpha,beta)

x(:,1) = x0;
time(1) = t0;

for kk = 1:tend/h

    tf = time(kk) + hmax;
    xx  = x(:,kk);

    % Explicit Runge-Kutta Method with 4 Stages
    c1 = 0;
    c2 = 1/2;
    c3 = 1/2;
    c4 = 1;

    b1 = 1/6;
    b2 = 2/6;
    b3 = 2/6;
    b4 = 1/6;

    a21 = 1/2;
    a31 = 0;
    a32 = 1/2;
    a41 = 0;
    a42 = 0;
    a43 = 1;


    if h(kk)>hmax
        h(kk) = hmax;
    end
    hh(1) = h(kk);
    for i = 1:MAX_ITER
```

```
          k1 = f(time(kk), xx(:,i));
          k2 = f(time(kk) + c2*h(kk), xx(:,i) + h(kk)*a21*k1);
          k3 = f(time(kk) + c3*h(kk), xx(:,i) + h(kk)*a31*k1 + h(kk)*a32*k2);
          k4 = f(time(kk) + c4*h(kk), xx(:,i) + h(kk)*a41*k1 + h(kk)*a42*k2 + h(kk)*a43*k3);

          xx(:,i+1) = xx(:,i) + h(kk)*(b1*k1 + b2*k2 + b3*k3 + b4*k4); % Iteration

          er = Runge_Kutta(f, xx(:,i), time(kk), h(kk));
          err(i) = norm(er);
          if i == 1
              if err(i) > rtol
                  h(kk) = h(kk)*alpha;
                  hh(i+1) = h(kk);
              elseif  err(i) == rtol
                  hh(i+1) = h(kk);
                  hnext = hh(i+1);
                  xf = xx(:,i+1);
              else
                  h(kk)= h(kk)*beta;
                  hh(i+1)= h(kk);
              end
          else
              if err(i) > rtol
                  h(kk) = h(kk)*alpha;
                  hh(i+1) = h(kk);
                  if err(i-1) <= rtol
                      hnext = hh(i);
                      xf = xx(:,i);
                  end
              elseif  err(i) == rtol
                  hh(i+1) = h(kk);
                  hnext = hh(i+1);
                  xf = xx(:,i+1);
              else
                  h(kk) = h(kk)*beta;
                  hh(i+1) = h(kk);
                  if err(i-1) >= rtol
                      hnext = hh(i+1);
                      xf = xx(:,i+1);
                  end
              end
          end
      end


      h(kk+1) = hnext;
      step(kk) = hnext;
      x(:,kk+1) = xf;
      time(kk+1) = time(kk) + hnext;

      if time(kk)>=tend
          break;
      end
  end
```

```
end

function [y, out] = Runge_Kutta(func, y, x0, h)

a2 = 0.25;
a3 = 0.375;
a4 = 12/13;
a6 = 0.5;

b21 = 0.25;
b31 = 3/32;
b32 = 9/32;
b41 = 1932/2197;
b42 = -7200/2197;
b43 = 7296/2197;
b51 = 439/216;
b52 = -8;
b53 = 3680/513;
b54 = -845/4104;
b61 = -8/27;
b62 = 2;
b63 = -3544/2565;
b64 = 1859/4104;
b65 = -11/40;

c1 = 25/216;
c3 = 1408/2565;
c4 = 2197/4104;
c5 = -0.20;

d1 = 1/360;
d3 = -128/4275;
d4 = -2197/75240;
d5 = 0.02;
d6 = 2/55;

h2 = a2 * h; h3 = a3 * h; h4 = a4 * h; h6 = a6 * h;

k1 = func(x0, y);
k2 = func(x0+h2, y + h * b21 * k1);
k3 = func(x0+h3, y + h * ( b31*k1 + b32*k2) );
k4 = func(x0+h4, y + h * ( b41*k1 + b42*k2 + b43*k3) );
k5 = func(x0+h,  y + h * ( b51*k1 + b52*k2 + b53*k3 + b54*k4) );
k6 = func(x0+h6, y + h * ( b61*k1 + b62*k2 + b63*k3 + b64*k4 + b65*k5) );
y = y +  h * (c1*k1 + c3*k3 + c4*k4 + c5*k5);
out = d1*k1 + d3*k3 + d4*k4 + d5*k5 + d6*k6;

end
```

Published with MATLAB® R2019b