ECE 9513 Final Report - Analysis of PUMA 560 Manipulator with Various Control Algorithms

Chris Morley

Introduction

Throughout this report several different control algorithms will be introduced and demonstrated on a PUMA 560 model. The provided dynamic model of the PUMA 560 will be used in simulations of the more simplistic algorithms however a more realistic Simscape model will be used to test the more advanced algorithms which include interaction between the manipulator and the environment. The remainder of the report will be as follows (a) brief overview of the Simscape model and how it compares to the provided model, (b) some common nomenclature used throughout the report, (c) an overview of which algorithms will be covered during the report and lastly (d), an in-depth look at each algorithm including any definitions/justifications of chosen parameters, specific simulation scenario(s) and results.

Simscape Model

The Simscape Multi-body toolbox for MATLAB/Simulink was used to model the PUMA 560. The Simscape toolbox allows for CAD models to be imported in the Simulink environment where the CAD models can then be actuated at each joint in the model. The toolbox also allows for environment features to be created with which the robot can interact with. Interaction with the environment will be demonstrated with some of the later algorithms. Two slightly different models of the PUMA 560 were created, the first representing the geometry of the true PUMA 560 robot and the respective center of mass (COM) for each link. The second model which was created was representative of the provided dynamic model in which all the COMs are in the same plane. Some of the later algorithms showcase their usefulness when used to control the first model, where the parameters of the model in the controller do not exactly match that of the manipulator they are attempting to control. Both models of the PUMA 560 can be seen in Figure 1 & Figure 2. Note that the CAD itself looks the same but the COMs are in different locations. It can be assumed that if no mention of which PUMA 560 model is made for a given algorithm that the model with realistic COM locations is used.

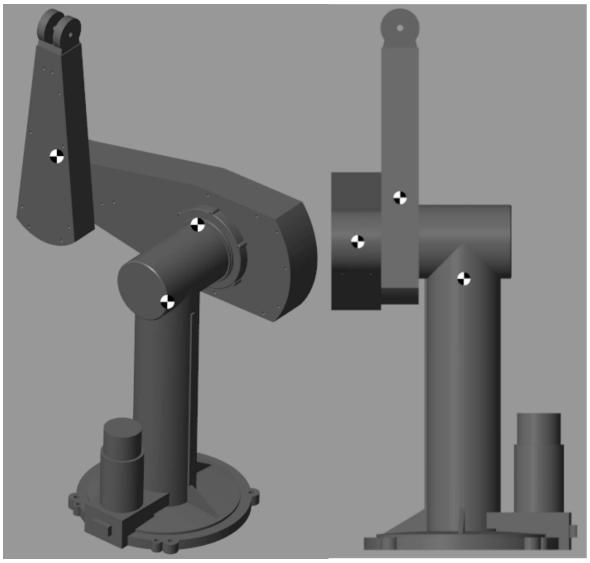

Figure 1 - PUMA 560 True Center of Mass

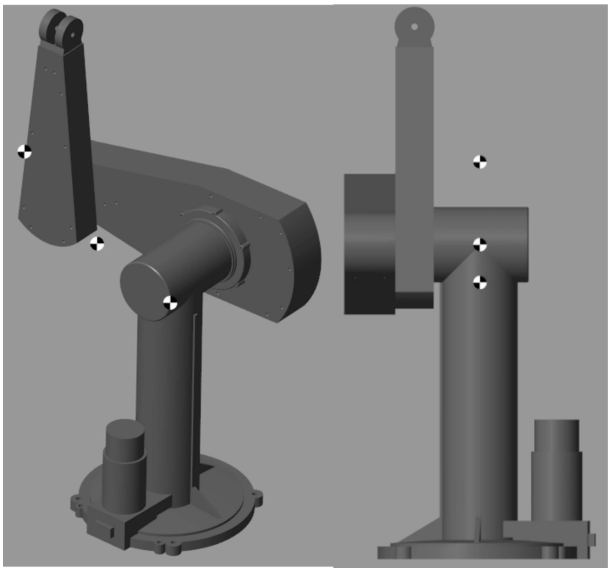

Figure 2 – PUMA 560 Planar Center of Mass

Nomenclature

The PUMA 560 manipulator can be described by the following equation:

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$

Where:

$$H(q) = Inertia\ Matrix$$
$$C(q,\dot{q}) = Coriolis\ \&\ Centrifugal\ terms$$
$$G(q) = Gravity\ Terms$$
$$\tau = Actuator\ Torque$$
$$q = Joint\ Positions$$

$$\dot{q} = Joint\ Velocity$$
$$\ddot{q} = Joint\ Acceleration$$

Throughout this report the closed-loop dynamics of the manipulator will be referenced. This refers to substituting the control algorithm in place of τ in the manipulator dynamics and bringing all terms in the equation to one side. Often doing so will cancel terms and simplify the equation. What is left describes the closed-loop dynamics of the manipulator in free space, with no external forces acting on it.

Other common terms which appear throughout the report are:

$$\tilde{q} = Joint\ Position\ Error$$
$$\dot{\tilde{q}} = Joint\ Velocity\ Error$$
$$\ddot{\tilde{q}} = Joint\ Acceleration\ Error$$
$$q_d = \text{Desired Joint Position}$$
$$\dot{q}_d = \text{Desired Joint Velocity}$$
$$\ddot{q}_d = \text{Desired Joint Acceleration}$$
$$K_d = Derivative\ Gain\ Matrix$$
$$K_p = Porportional\ Gain\ Matrix$$

Overview of Algorithms

This report will cover position control, interaction control and force control algorithms, in that order. Within each class of algorithms certain controllers have superior performance under certain conditions. The simulation results will highlight the advantages and disadvantages of each algorithm. The algorithms which will be examined for each category can be seen in Table 1.

Position Control	• Computed Torque • Sliding Adaptive Control • Sliding Robust Control
Interaction Control	• Task Space Impedance Control
Force Control	• Parallel Motion/Force Control • Hybrid Motion/Force Control

Table 1 - Studied Algorithms

Computed Torque

This algorithm forms the bases for many of the latter algorithms. The idea behind computed torque is to choose the actuator inputs by taking advantage of the mathematical model of the manipulator to cancel out the natural dynamics of the unactuated robot and then specify your desired dynamics. The process of canceling out the natural dynamics of the manipulator is often refered to inverse dynamics (ID) and the process of specifying your desired dynamics if often refered to as tracking control. The computed torque algorithm, shown below, cancels out the natural dynamics of the robot leaving us free to specify the acceleration of the manipulator under closed-loop conditions. There is a free parameter "u" in the control law. Once the control law is substituted into the manipulator dynamics to form the closed-loop dynamics it will be apparent how we go about choosing "u".

$$\tau = H(q)u + C(q,\dot{q})\dot{q} + G(q)$$

Substituting the control law into the manipulator dynamics can be seen below. The terms of matching colour cancel each other out.

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = H(q)u + C(q,\dot{q})\dot{q} + G(q)$$
$$H(q)\ddot{q} = H(q)u$$
$$\ddot{q} = u$$

Selecting "u" to be the desired acceleration is an obvious choice however this alone would does not incorporate any feedback and would not allow the control law to correct the manipulator if the manipulator drifts from it's desired trajectory. The following choice of "u" is made to correct for both position and velocity errors at each joint.

$$u = \ddot{q_d} - K_d * \dot{\tilde{q}} - K_p * \tilde{q}$$

Substituting "u" into the closed-loop dynamics yields the following result.

$$\ddot{\tilde{q}}_d + K_d * \dot{\tilde{q}} + K_p * \tilde{q} = 0$$

Because K_d & K_p are positive diagonal matrices the solution to the above differential equation is asymptotically stable.

This algorithm was run on the planar PUMA 560 model, the measured joint positions along with the desired joint positions can be see in Figure 3. Because the manipulator parameters in the controller perfectly match that of the manipulator being controlled the trajectory tracking is perfect in this scenario. Subsequently, the desired joint trajectories

perfectly coincide with the measured joint trajectories.

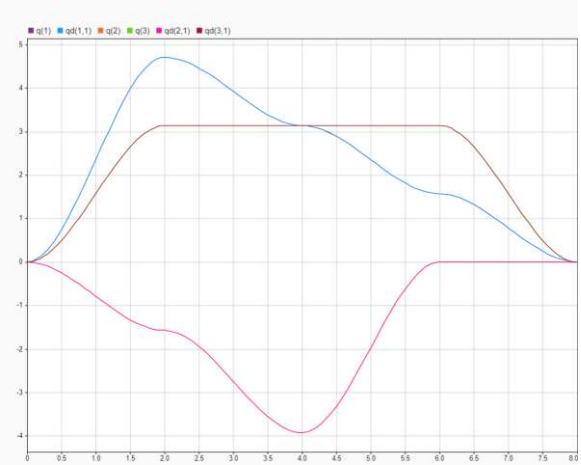

Figure 3 - Computed Torque Perfect Tracking

One large disadvantage with this control law however is that the parameters such as inertia, mass & link lengths need to be known exactly for perfect tracking. Knowing the exact parameters of a physical system is nearly impossible which makes this algorithm unpractical for many real-world applications. This downfall of computed torque can be seen in Figure 4 where the PUMA 560 Simscape model with real COM locations is used in place of the planar COM model. As a result, matrices H, C & G, in the control law do not perfectly represent the manipulator being controlled.

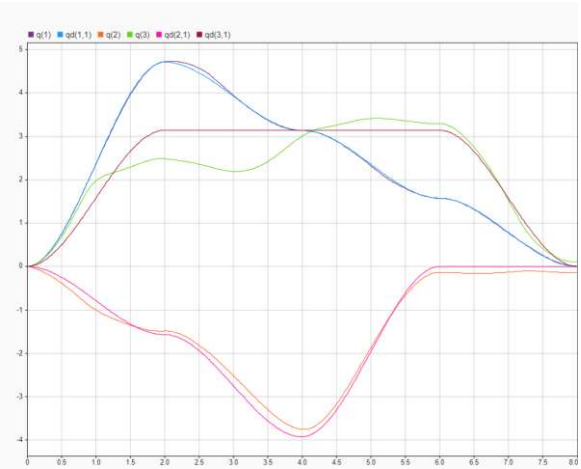

Figure 4 - Computed Torque, Non-Ideal Tracking

As it can be seen, the computed torque law fails to accurately track the desired joint trajectories when the manipulator parameters are not known exactly. The resultant poor tracking in cartesian space can be seen in Figure 5. Note that the green spline represents the desired trajectory and the red spline represents the true trajectory.

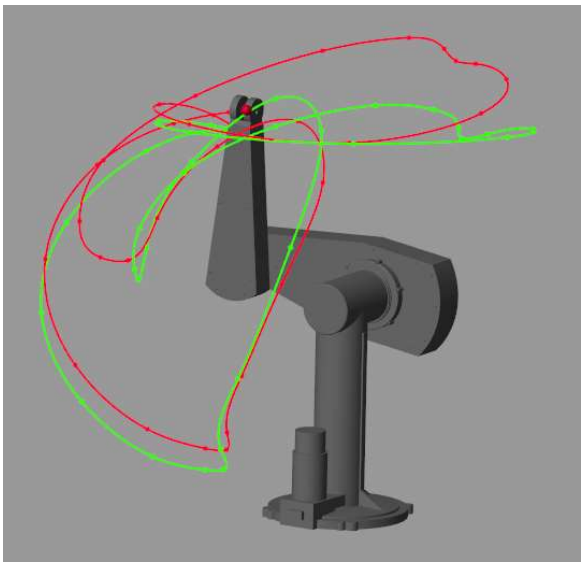

Figure 5 - Computed Torque Poor Cartesian Trajectory Tracking

Sliding Adaptive Control

Before discussing sliding adaptive control, the idea behind adaptive control is summarized below.

As shown with computed torque, uncertainty in system parameters can be detrimental to the performance of a system. This made researchers ask the question, can we develop a control law which can accurately track trajectories in the presence of parameter uncertainty? Beginning in the 1950s, one method put forward of dealing with this was adaptive control [1]. This section will demonstrate how the development of adaptive control offered a solution for when the parameters of a system are not known exactly.

Adaptive control is a control architecture comprising two parts, namely the adaptive law and the control law. The adaptive law is responsible for forcing the estimates of system parameters to converge to values which will allow the system to track the desired trajectory. Note the convergence of the estimates to the correct values is not guaranteed,

this will be discussed in more detail later. The control law uses the parameter estimates from the adaptive law, along with state measurements & desired trajectory information to calculate the required actuation for the system.

Before explaining sliding adaptive control it is advantageous to cover Sliding mode Control. Sliding mode control is a control method where the controller is designed so that the state of a given system will converge to a hyper plane and then stay on that hyper plane [2]. This hyperplane is usually denoted by "S" and often described by the following equation.

$$S = \dot{\tilde{q}} + \lambda * \tilde{q}$$

The hyperplane is the combination of the joint velocity errors and the joint position errors, scaled by lambda. Depending on the degrees of freedom in the system, the equation above can either be a scaler or matrix equation. Naturally it is desirable for error, of all kinds, to be zero. Setting S=0 the hyperplane, which we wish the state of our system to lie on, is defined within the entire state space. Sliding mode control has an advantage over it's counterparts because defining a hyperplane makes the control problem a first order one [2] in the sense that if S<0 you apply some positive actuation and if S>0 you apply some negative actuation. As it will be shown, once the state of your system reaches the hyperplane it is hypothetically guaranteed to remain on that hyperplane.

Sliding adaptive control combines both sliding mode control and adaptive control to achieve excellent trajectory tracking in the presence of parameter uncertainty. Below the formulation and proof of asymptotic stability will be shown.

To implement the adaptive control law, we must introduce what is known as the regressor matrix. The following equation will be used to define the regressor matrix.

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = Y * a$$

Y is the regressor matrix in the equation above. Y only contains terms consisting of state measurements and desired states. The vector, a, which is multiplying Y from the right, contains system parameters such as length, mass & inertia. The high-level idea is that the system dynamics can be expressed as a linear combination of the regressor matrix columns, where the coefficients of multiplicity are the rows of a. As it has been mentioned, we are dealing with the case where system parameters are not exactly known so as a result "a" is not known. Instead we introduce the following:

$$\hat{a} \coloneqq Estimate\ of\ System\ Parameters$$

Lastly before proving the asymptotic stability of the system with sliding adaptive control, one alternative representation of the hyperplane must be introduced.

$$S = \dot{q} - \dot{q}_r$$

Where:

$$\dot{q}_r = \dot{q}_d - \lambda * \tilde{q}$$

To prove asymptotic stability, the following Lyapunov function candidate is used. Note aTilda =aHat-a

$$V = \frac{1}{2} * H(q) * S^2 + \frac{1}{2} * \tilde{a}^T P^{-1} \tilde{a}$$

This function is positive definite and radially unbounded which meets the first criteria for asymptotic stability. Next, we need to ensure that the derivative is negative definite. Taking the derivative, we get the following result.

$$\dot{V} = S * H(q) * \dot{S} + \frac{1}{2} * \dot{\tilde{a}}^T P^{-1} \tilde{a} + \frac{1}{2} * \tilde{a}^T P^{-1} \dot{\tilde{a}}$$
$$\dot{V} = S * H(q) * \dot{S} + \dot{\tilde{a}}^T P^{-1} \tilde{a}$$
$$\dot{V} = S * H(q) * \dot{S} + \dot{\hat{a}}^T P^{-1} \tilde{a}$$

Aside: (Terms of identically colour in consecutive lines are substitutions)

$$H(q) * \dot{S} = H(q) * \frac{d}{dt}(\dot{q} - \dot{q}_r)$$
$$H(q) * \dot{S} = H(q) * (\ddot{q} - \ddot{q}_r)$$
$$H(q) * \dot{S} = H(q) * \ddot{q} - H(q) * \ddot{q}_r$$
$$H(q) * \dot{S} = \tau - G(q) - C(q,\dot{q})\dot{q} - H(q) * \ddot{q}_r$$
$$H(q) * \dot{S} = \tau - H(q) * \ddot{q}_r - G(q) - C(q,\dot{q})\dot{q}$$
$$H(q) * \dot{S} = \tau - Ya$$

Continuing from before:

$$\dot{V} = S * (\tau - Ya) + \dot{\hat{a}}^T P^{-1} \tilde{a}$$

We select the following control law where K is some positive definite diagonal matrix:

$$\tau = Y\hat{a} - KS$$

Substituting the control law into the derivative of the Lyapunov function candidate:

$$\dot{V} = S * (Y\hat{a} - KS - Ya) + \dot{\hat{a}}^T P^{-1}\tilde{a}$$
$$\dot{V} = -KS^2 + SY(\hat{a} - a) + \dot{\hat{a}}^T P^{-1}\tilde{a}$$
$$\dot{V} = -KS^2 + SY\tilde{a} + \dot{\hat{a}}^T P^{-1}\tilde{a}$$

The first term in red is negative definite however the two terms in blue have no guarantee of being negative definite. We need to do some manipulation to ensure that those two terms add to zero.

$$SY\tilde{a} + \dot{\hat{a}}^T P^{-1}\tilde{a} = 0$$
$$\left(SY + \dot{\hat{a}}^T P^{-1}\right)\tilde{a} = 0$$
$$\left(SY + \dot{\hat{a}}^T P^{-1}\right) = 0$$
$$\dot{\hat{a}} = -(PSY)^T$$
$$\dot{\hat{a}} = -PY^T S$$

With the condition above on the derivative of the estimate of "a", the derivative of our Lyapunov function candidate has only the first term in red remaining which is negative definite and thus our system is asymptotically stable. The condition above defines our adaptive law. The P matrix is often chosen to be a positive diagonal matrix. This is not necessary; however, it is required that P is invertible. P specifies the rate at which the estimated parameters converge.

The system is comprised of both the control law and the adaptive law. The adaptive law output is fed through an integrator which is then fed into the control law. A block diagram can be seen in Figure 6.

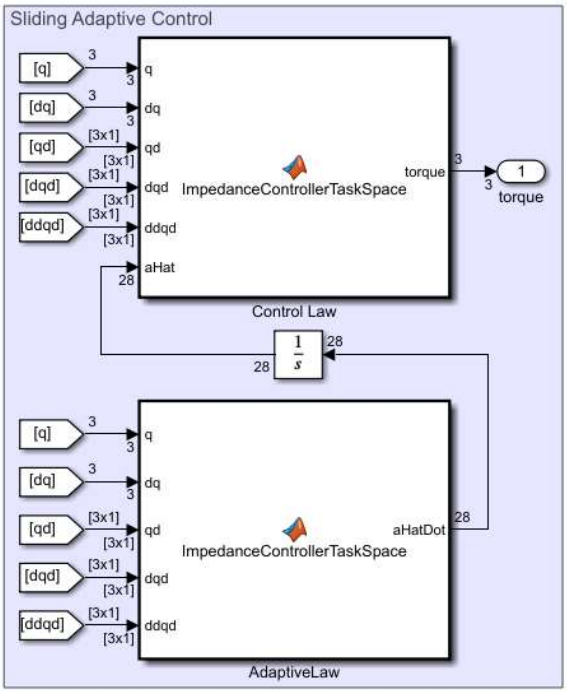

Figure 6 - Adaptive Control Architecture

The PUMA 560, using adaptive control, was commanded to follow the same trajectory that the computed torque algorithm attempted to follow in the presence of parameter uncertainty. The following parameters were chosen for the control law and adaptive law.

$$K = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix} \quad P = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 1000 \end{bmatrix}$$

The position trajectory tracking can be seen in Figure 7. As with Figure 3 for computed torque, the joint positions almost perfectly coincide with the desired joint positions so it appears as if there are only three signals plotted.

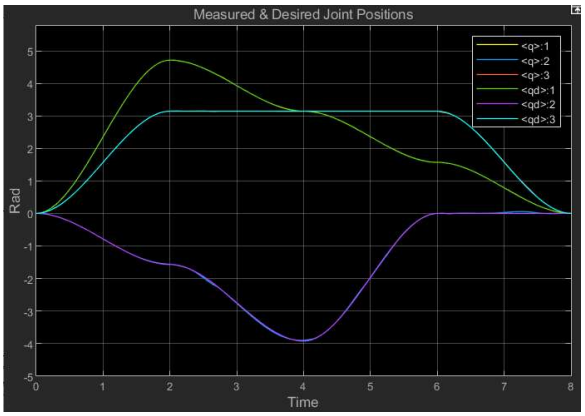

Figure 7 - Adaptive Control, Measured & Desired Joint Positions, P=100

To demonstrate the requirement of large adaptive gains for the parameter estimates to converge in a timely manor [3], the P matrix was chosen to be an identity matrix for the next scenario. It can be seen in Figure 8 that the tracking is poor when compared to the scenario with P=1000.

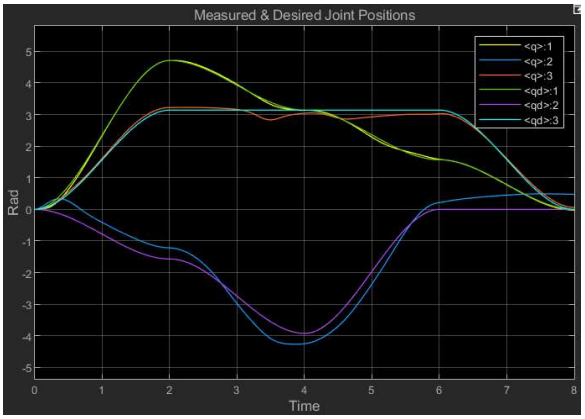

Figure 8 - Adaptive Control, Measured & Desired Joint Positions, P=1

One downfall with adaptive control/sliding adaptive control is that the parameters are estimated on a need to know basis, meaning that the estimate will only be good enough to allow for sufficient tracking of the desired trajectory. There is no guarantee that all the parameter estimates will converge to the true values. For all the parameters to converge to the correct value, the desired trajectory must have sufficient richness [4] or in other words must be so complicated that there is no way for the manipulator to follow the trajectory without knowing all the parameters exactly. This sufficient richness criteria is not a problem for those simply wanting to follow a desired trajectory however this algorithm should not be relied on to estimate parameters of an unknown load.

Sliding Robust Control

Adaptive control is good in situations where the parameters are not known exactly and/or slowly changing. Robust control on the other hand is a control technique that can guarantee trajectory tracking in the presence of uncertainty/disturbances where the upper bound on the uncertainty/disturbances is known [4]. Robust control is superior in dealing with sudden changes in system disturbance when compared to adaptive control [4].

Sliding robust control is a combination of sliding mode control and robust control, where again the objective is to design a controller that can guarantee that the state of a system converges to, and then remains on a hyperplane.

The first step of the design process is to determine the upper bound on system disturbance. To do this the manipulator dynamics first need to be rewritten as shown below.

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau$$
$$\ddot{q} = H(q)^{-1}\left(-C(q,\dot{q})\dot{q} - G(q)\right) + H(q)^{-1}\tau$$

We then define the following:

$$\hat{f} = H(q)^{-1}\left(-C(q,\dot{q})\dot{q} - G(q)\right)$$

To demonstrate the robustness of this controller, we will consider the case where there is some external force applied to the end-effector (EE) of the manipulator. The goal is to prove that the manipulator can follow the desired trajectory with some external force acting on the EE, which only the upper bound is known. Again, we will consider the manipulator dynamics, this time we will include an external force acting on the EE. Note that "d" is a 3x1 vector containing the x, y & z EE external forces acting on the EE in the EE frame of reference.

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau + J^T * d$$
$$\ddot{q} = H(q)^{-1}(-C(q,\dot{q})\dot{q} - G(q) + J^T * d) + H(q)^{-1}\tau$$

We then also define the following:

$$f = H(q)^{-1}(-C(q,\dot{q})\dot{q} - G(q) + J^T * d)$$

f is the true dynamics of the manipulator, whereas fHat is the ideal dynamics with no disturbances. We are interested in an upper bound for the absolute value of the difference of these two. The upper bound is denoted by "F" and defined as follows:

$$F \geq \left| f - \hat{f} \right|$$
$$F \geq \left| H(q)^{-1} * J^{T} * d \right|$$

Intuitively, F can be thought of as the maximum uncertainty in our model of the manipulator. To implement this algorithm, we need to find a vector, F, of either constants or functions which is the upper bound for model uncertainty. Note that F is a 3x1 vector, where row 'j' is the model uncertainty for joint 'j'. In our derivation of the system dynamics, model uncertainty is defined as angular acceleration due to unmodeled dynamics/disturbances. The disturbance/uncertainty in the system is related to the EE forces by the transpose of the jacobian and the inverse inertia matrix. This relationship will be used to determine only an approximation of system disturbance because the manipulator will be following a time-varying trajectory however the jacobian is used to relate *static* forces to joint torques. For this demonstration we will assume that we know a maximum of 200N can be applied to each the x, y & z axis of the EE frame. In the equation for F, d is constant however $H(q)^{-1}$ and J^{T} are dependant on joint positions. Intuitively this makes sense because the external disturbance will affect the manipulator dynamics differently depending on the manipulator configuration. There are two different options for defining F. The first option is to define F exactly as described above where F is a multi-variable function of joint positions. The other option is to define F as a constant vector which is an upper bound for the system disturbance. The latter option reduces the computational expense of the control algorithm while the earlier option allows for a smaller region of convergence around the hyperplane which results in a smaller state error.

In the case of the first option, F is a function of only joints 2 & 3. This makes sense because the external forces are being applied to the EE frame of reference. From the point of view of the EE frame, the manipulator does not appear to change configurations as the first joint changes positions. Because F only depends on two independent variables, the function in each row of F (disturbance at each joint) can be plotted as a surface againts joint 2 & 3 positions. This can be seen in Figure 9.

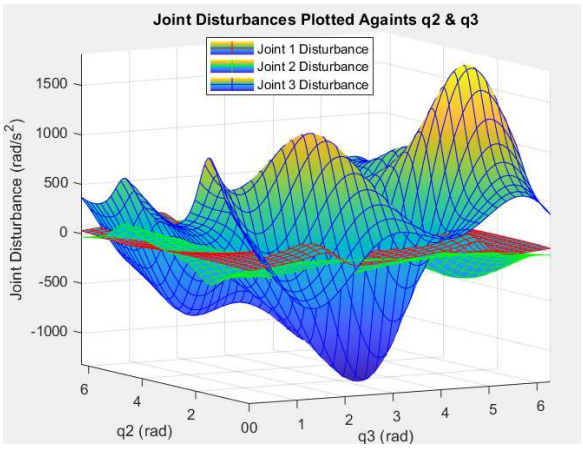

Figure 9 – Joint Disturbances, Plotted Againts Joint Positions

In the latter of the two options we are interested in a constant upper bound on disturbance for each joint. The constant values we are interested in are the maximum absolute values of each surface in Figure 9. These values were found using the fminsearch() function in MATLAB and are shown below (rounded to the nearest integer).

$$F = \begin{bmatrix} 254 \\ 341 \\ 1826 \end{bmatrix}$$

To design the controller we will mandate what is known as a sliding condition. A sliding condition places some requirement on the state of the system converging to the hyperplane described in sliding adaptive control. One choice would be:

$$\frac{d}{dt} S^2 < 0$$

This is sufficient however we can do better by placing time constraints on the convergence to the hyperplane with the following sliding condition:

$$\frac{d}{dt} \frac{1}{2} S^2 < \eta |S|$$

The LHS of the equation must be evaluated and substitutions must be made to arrive at a control law. The derivation can be seen below.

$$\frac{d}{dt} \frac{1}{2} S^2 < \eta |S|$$
$$S * \dot{S} < \eta |S|$$
$$S * \left(\ddot{q} + \lambda \dot{\tilde{q}} \right) < \eta |S|$$
$$S * \left(\ddot{q} - \ddot{q}_d + \lambda \dot{\tilde{q}} \right) < \eta |S|$$
$$S * \left(f + H(q)^{-1} \tau - \ddot{q}_d + \lambda \dot{\tilde{q}} \right) < \eta |S|$$

Now tau is chosen to eliminate everything in the brackets as best as possible while also adding a control term (in red) which will drive the state of the system to the hyperplane.

$$\tau = H(q) * (-\hat{f} + \ddot{q}_d - \lambda\dot{\tilde{q}} \text{-K*sign(S)})$$

Substituting the control law back into the sliding condition yields the following result.

$$S * \left(f - \hat{f}\right) - K * |S| < \eta|S|$$

Replacing the difference between f & fHat with F, found previously, and solving the expression for K we arrive at the following result.

$$K \geq F + \eta$$

One common problem with any variation of sliding mode control is what is known as chattering [4]. Examining the control law, it can be seen the mapping from S-> desired actuation is discontinuous at S=0 due to the sign() function. As the state of the system approaches the hyperplane from one side, it will actually move slightly past the hyperplane before the system has had a chance to respond. This is due to non-instantaneous actuators and computation time. Once the system has realized the state has overshot the plane, the system calls for actuation in the opposite direction. This is repeated at an incredibly high frequency and can cause problems in many types of systems [4]. To combat the issue of chattering, designers will specify a region around the hyperplane for the state of the manipulator to converge to instead of explicitly specifying that the state must converge to the hyperplane. This is done by replacing the sign() function with a saturation function. The region surrounding the hyperplane and the saturation function can be seen in Figure 10 & Figure 11 respectively.

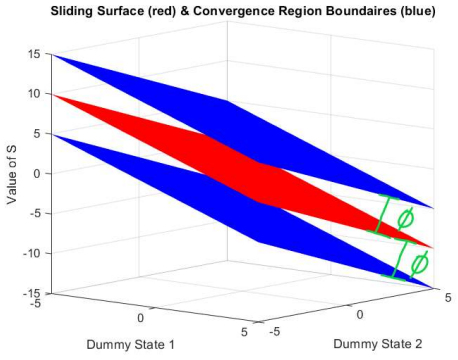

Figure 10 - Hyperplane Region

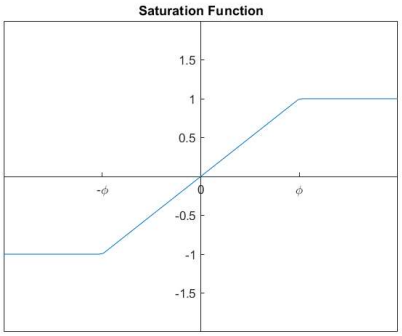

Figure 11 - Saturation Function

Phi defines the thickness of the region of convergence. Intuitively, Phi can be thought of as the acceptable error of the state. If F is chosen to be state dependant then Phi will also be state dependant. Phi can be continuously updated with the use of a filter. This is described below.

The behaviour of the closed-loop dynamics can be interpreted as two cascaded filters. The first filter accepts the disturbance as an input and outputs the value of S. The second filter accepts the value of S and outputs the position error. The filter setup can be seen in Figure 12.

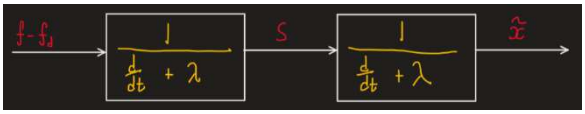

Figure 12 - Cascade Filter Interpretation

To guarantee the transfer functions seen in Figure 12, the following relationship between lambda and phi must hold true. Note that this is derived from the sliding condition.

$$\lambda = \frac{K - \dot{\phi}}{\phi}$$

K, in the expression above is set equal to the following:

$$K = F + \eta - \dot{\phi}$$

The first derivative of phi appears in the expression for K to ensure that the time constraint on the state converging to the hyperplane region is met while the hyperplane region boundaries are moving because of phi changing. Note that in the case where F is constant the derivative of phi is equal to zero. Rearranging the relationship between phi & lambda and then substituting the appropriate expression for K yields the following result:

$$\dot{\phi} + \lambda * \phi = F + \eta - \dot{\phi}$$

Phi can be computed, in real-time, with the filter in Figure 13.

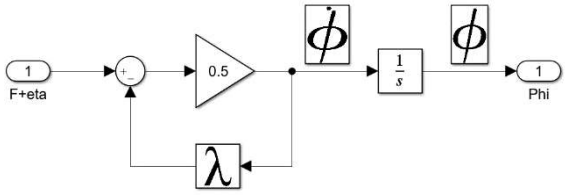

Figure 13 – Phi Filter

The remaining algorithm parameters are as follows. Note that in practice there is an upper bound on the diagonal entries in lambda so that the control law does not interfere with unmodeled dynamics [4] however for our simulations the actuator & computation dynamics are treated as ideal so the values of lambda can be chosen freely.

$$\lambda = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 50 \end{bmatrix}$$

$$\eta = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Recall the goal of this algorithm is to guarantee that the state of the system converges to the region defined around the hyperplane when there are disturbances bounded by some upper value. In our case the algorithm was designed to track a given trajectory while the EE is subject to a force of up to 200 N in all three axes of the EE frame. The disturbance to the system, due to the external forces, was approximated using the jacobian and the inverse inertia matrix. Recall this method is only an approximation because the jacobian is only valid for static forces.

To demonstrate the effectiveness of this algorithm, the manipulator will attempt to follow a given trajectory under two different external force conditions. The two different scenarios are when all EE axes are subjected to 250 N & 150 N respectively. This will demonstrate how the manipulator behaves while operating above and below the allowable disturbance. Each disturbance scenario will be run using the constant F vector found earlier. Following the initial simulations, the benefit of a state-dependant F vector will be demonstrated. The measured/desired joint positions for both initial scenarios can be seen in Figure 14 & Figure 16. The distance from the hyperplane and the region of convergence boundaries can be seen in Figure 15 & Figure 17.

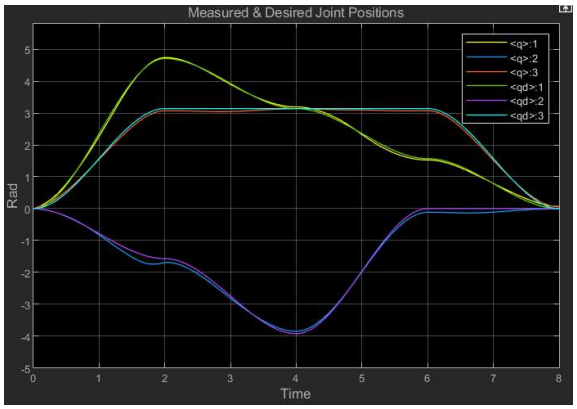

Figure 14 - Sliding Robust Control Measured & Desired Joint Positions, Disturbances Greater Than 200 N

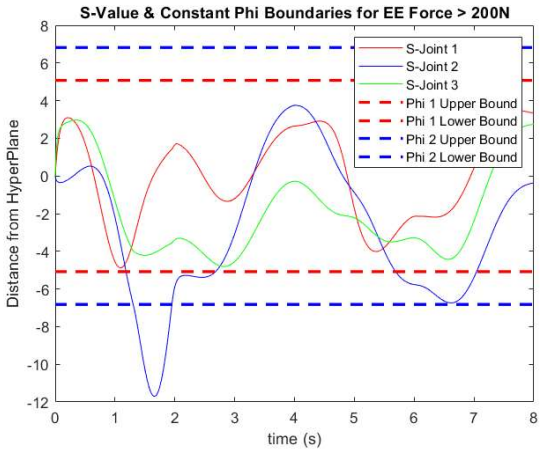

Figure 15 - Distance From Hyperplane & Phi Boundary, Each Joint, Disturbances Greater Then 200 N

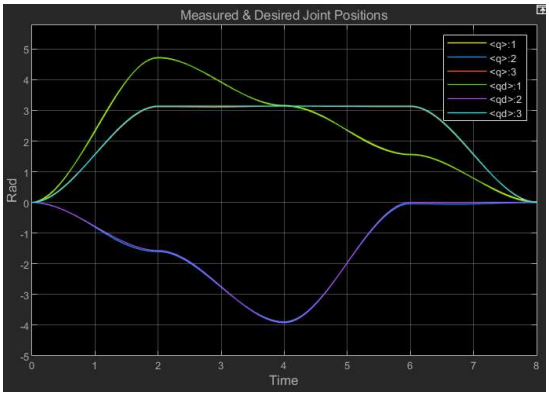

Figure 16 - Sliding Robust Control Measured & Desired Joint Positions, Disturbances Less Than 200 N

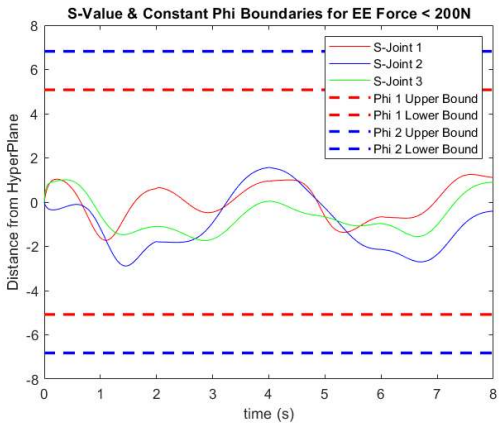

Figure 17 - Distance From Hyperplane & Phi Boundaries, Each Joint, Disturbances Less Then 200 N

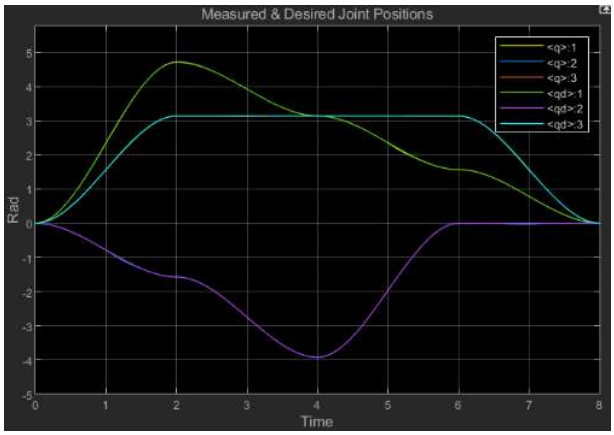

Figure 18 - Sliding Robust Control Measured & Desired Joint Positions with State-Varying F Vector, Disturbances Less Than 200 N

It can be seen by inspection of the joint position plots, that the manipulator does a superior job tracking the desired trajectory in the scenario with EE forces less than 200 N. It is shown in Figure 17 that all joints stay within the specified region around the hyperplane when the external forces are within the allowable limits. Conversely in Figure 15, the state of the second joint fails to stay within the region of the hyperplane when the disturbance is greater than the allowable limit.

Recall how the simulations above were run with the constant F vector. The values in this vector assumed worst case scenario of joint disturbances, regardless of the current manipulator configuration. Another scenario was run with the EE forces equal to 150 N however this time the state-varying F vector was used. The constant F vector can be unnecessarily conservative for a given region of the state space and as a result, the region of convergence around the hyperplane is unnecessarily large. Measured/desired joint positions and the state's distance from the hyperplane & region of convergence boundaries can be seen in Figure 18 & Figure 19 respectively.

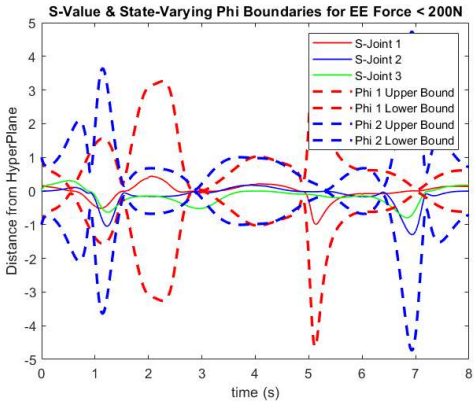

Figure 19 - Distance From Hyperplane & State-Varying Phi Boundaries, Each Joint, Disturbances Less Then 200 N

Table 2 presents the RMS distance, calculated over the entire duration of the trajectory, between the state & hyperplane for each joint in the manipulator. This was calculated for both the constant & state-varying F vector when the EE force is less than 200 N. The results indicate that the state-varying F vector results in a more accurate tracking of the trajectory, for all joints, when compared to the constant F vector simulation.

Joint #	Constant F	State-Varying F
1	0.7808	0.2562
2	1.4635	0.3003
3	0.9608	0.2737

Table 2 - Distances from Hyperplane

One last item to point out about the benefits of the state-varying F vector, is the ability to highlight a region of the system's state-space where the disturbance modeling is inaccurate. The region of convergence

boundaries, for the third joint, were intentionally left out in the previous plots because including them would scale the y-axis such that the plot was difficult to read. In Figure 20 the joint 3 boundaries are shown along with the distance from joint 3's state to the hyperplane. Note how at moments in time the Phi boundary is an order of magnitude larger than the state's distance to the hyperplane. This indicates that using the jacobian was not an accurate estimate for joint 3 disturbances due to the EE external forces.

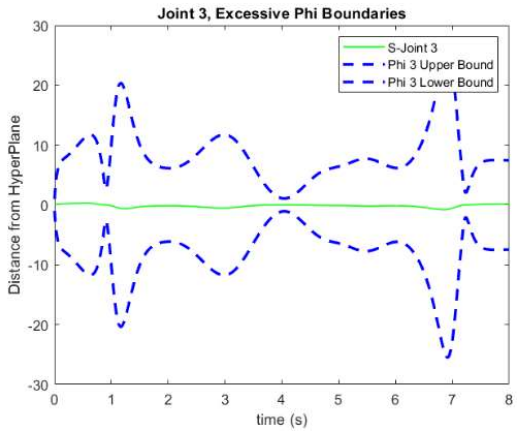

Figure 20 - Excessive Joint 3 Phi Boundaries

To compare sliding robust control with an algorithm not designed to handle system disturbances, the computed torque algorithm attempted to follow the same trajectory with the presence of external forces equal to 150 N. The same plots can be seen in Figure 21 & Figure 22 for the computed torque algorithm

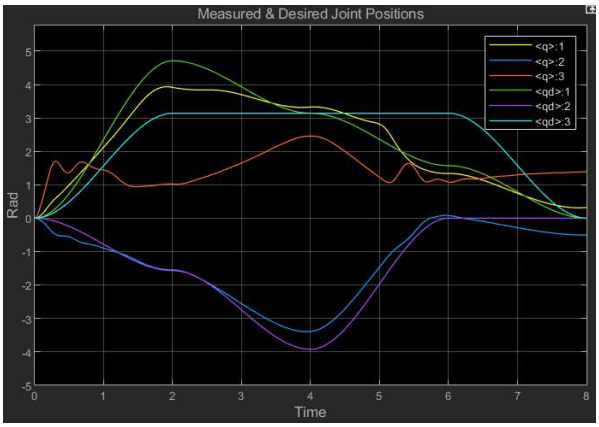

Figure 21 -Computed Torque Measured & Desired Joint Positions, Disturbances Less Than 200 N

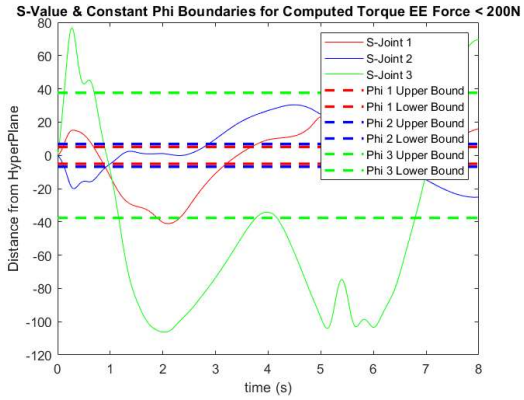

Figure 22 - Computed Torque Measured & Desired Joint Positions, Disturbances Less Than 200 N

All three joints fail to stay within their respective hyperplane region when computed torque was used. Note that the hyperplane does not appear within the computed torque algorithm, it is just used here to provide a constant metric for evaluation. It can be concluded that the robust controller outperforms the computed torque control method when dealing with external forces/disturbances. Further it can be concluded that using a state-varying F vector can even further improve performance and highlight states where the model of your system disturbance is inaccurate. In Figure 23 the desired EE trajectory can be seen in green along with the resulting trajectories of sliding robust control and computed torque in yellow and red respectively.

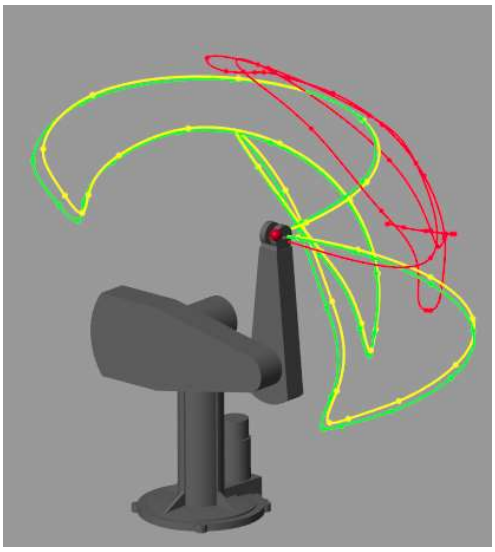

Figure 23 - Desired vs. Sliding Robust Control vs. Computed Torque Cartesian Trajectory in the Presence of 150 N Forces acting along EE Axes

Task Space Impedance Control

Impedance control is used when the operator of a manipulator is interested in controlling the interaction between the EE and environment. One common way of controlling the interaction is to use inverse dynamics to create closed-loop dynamics which resemble that of a mass-spring-damper. That is to say the EE acts as though it is connected to the reference point along the desired trajectory through a spring and damper. Such an implementation is desirable for manipulators interacting with their surroundings because they do not appear rigid to objects they may make contact with. Instead the EE appears as a mass-spring-damper, defined by a stiffness and damping matrix respectively.

To test the effectiveness of this algorithm the manipulator will have a constant reference trajectory (constant position, zero velocity & acceleration) while the EE is subjected to contact from a moving wall. The wall will apply a maximum, specified, force to the EE and the displacement of the EE along the normal direction of contact force will be measured to see if it moves as expected according to the linear relationship between force and displacement for a linear spring.

The task space impedance control algorithm is shown below. Note that this algorithm does not require contact force measurements. Not requiring force measurements comes at the cost of not being able to specify the desired apparent inertia matrix.

$$\tau = J^T * (H_p * \ddot{p}_d + D_m * (\dot{p}_d - \dot{p}) + K_m * (p_d - p) + C_p(q, \dot{q}) * \dot{p} + G_p)$$

Where D_m & K_m are the damping and stiffness matrices respectively. The subscript 'p' indicates the matrices are in the task space frame of reference. This algorithm yields the following closed-loop dynamics, where f is the vector of forces acting on the EE, in the task space coordinate frame.

$$H_p * \ddot{\tilde{p}} + D_m * \dot{\tilde{p}} + K_m * \tilde{p} = f$$

The scenario described was performed with the following stiffness and damping matrices:

$$K_m = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 1000 \end{bmatrix} \& \; D_m = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 50 \end{bmatrix}$$

Figure 24 shows manipulator in it's initial & final position after the force of the wall has moved the EE as far as it will go. The wall, and subsequently the EE, was driven by a force ramped up from 0 to 20 N over the interval 2 to 6 seconds.

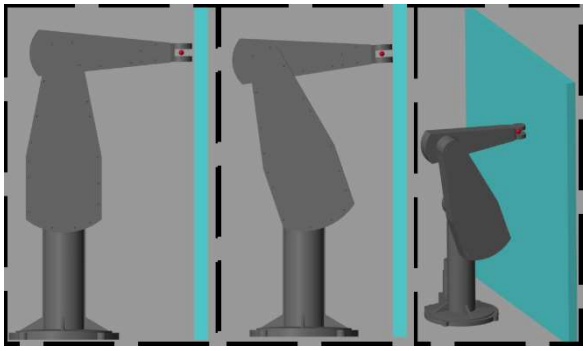

Figure 24 - Impedance Control, Different End-Effector Positions due to Moving Wall

The x-position of the EE, in the task space frame, can be seen in Figure 25 plotted againts time during the simulation. Highlighted in the red box is the x-axis displacement. In this simulation the displacement was 196.5 mm whereas the expected displacement was 200 mm. The small discrepancy in displacement is likely due to the true COM PUMA being used for the simulation. The simulation was re-run with the planar PUMA model. The results from the planar PUMA simulation showed that the x-axis displacement was exactly 200mm which matches the expected value. The plot from the second simulation can be seen in Figure 26.

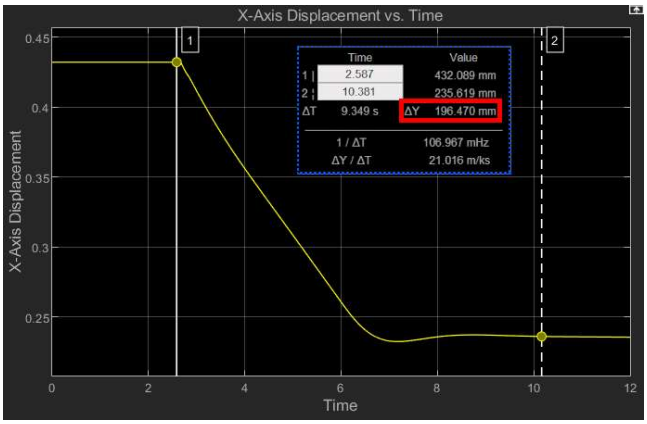

Figure 25 - Impedance Control, True COM PUMA, x-axis Displacement (mm) vs Time

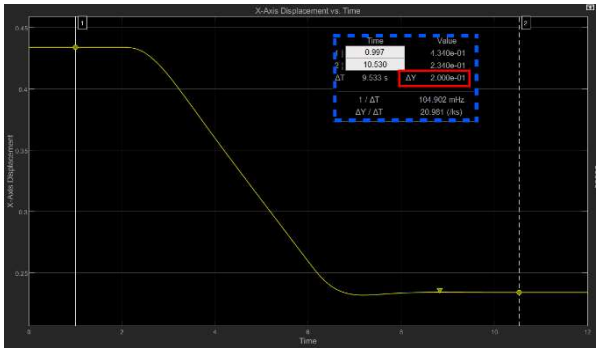

Figure 26 - Impedance Control, Planar COM, x-axis Displacement (m) vs. Time

To verify the damping properties of the manipulator are as specified, a transfer function for the x-axis motion was derived from the closed-loop dynamics so that a prediction could be made about the expected percent overshoot in response to a step input. The transfer function can be seen below.

$$\frac{X(S)}{F(S)} = \frac{1}{{H_p}^{1,1} * \ddot{x} + 50 * \dot{x} + 100 * x}$$

Note that $H_p^{1,1}$ is dependant on the joint positions. When computing the transfer function, $H_p^{1,1}$ was assumed to be constant and given the value corresponding to the manipulator configuration at t=0. That value of $H_p^{1,1}$ was 13.42. The damping ratio was determined to be 0.6829 & the expected percent overshoot was determined to be 5.13%. A simulation was run using the true COM PUMA 560 model where a step input force was applied to the EE along the task space x-axis direction. The step function had a final value of 20 N, just as with the final force in the wall scenario. The simulation showed that the manipulator had a percent overshoot of 9.48%. The discrepancy can be attributed to two sources, (a) the value of $H_p^{1,1}$ which was assumed to be constant and (b) the mathematical model used in the control law not being perfectly representative of the PUMA 560 model used. The simulation was run again except this time the planar PUMA 560 model was used. With the planar model, the percent overshoot was 4.98%. That result is very closed to the expected outcome so the small discrepancy be can attributed to the $H_p^{1,1}$ value being assumed to be constant. Figure 27 demonstrates the difference in percent overshoot when using the two different PUMA models.

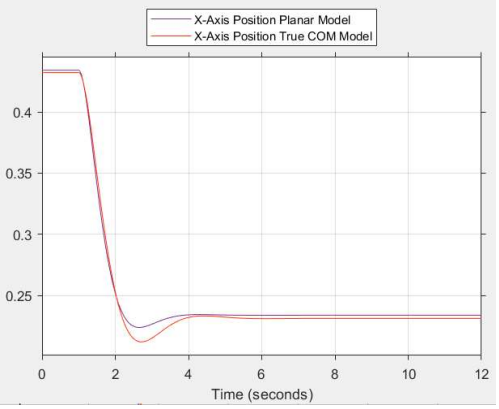

Figure 27 - Impedance Control Damping Comparison between different Puma Models

It was proven that the impedance control algorithm can control the given manipulator in such a way that it resembles a mass-spring-damper *provided* that the manipulator parameters are known exactly. When the parameters are not known exactly, the manipulator will not behave as an ideal mass-spring-damper and can demonstrate unexpected overshoot & displacement.

Parallel Motion/Force Control

This algorithm was designed in such a way that the manipulator can maintain a desired force between the EE and the environment while also attempting to track a desired position trajectory. This algorithm is very similar to that of impedance control, although more similar to the version not covered earlier which requires a force measurement. The difference between the two algorithms is that the desired trajectory in parallel motion/force control is made up of two components, namely the position trajectory and the force trajectory. The position trajectory is generated before-hand. The force trajectory is calculated in real-time using a double integrator filter. The trajectory generated by the filter, alters the position trajectory so that the desired contact force between the EE & environment is achieved.

The parallel motion/force algorithm can be seen below.

$$\tau = J^T * \left(H_p * u_0 + C_p(q, \dot{q}) * \dot{p} + G_p(q) - f\right)$$

Where:

$$u_0 = \ddot{p}_r + H_m^{-1} * (D_m * (\dot{p}_r - \dot{p}) + K_m * (p_r - p))$$

Where the following choices were made:

$$H_m = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{bmatrix}, D_m = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 50 \end{bmatrix} \&$$
$$K_m = \begin{bmatrix} 500 & 0 & 0 \\ 0 & 500 & 0 \\ 0 & 0 & 500 \end{bmatrix}$$

P_r denotes the overall desired trajectory. It is the sum of the task space position trajectory and the force trajectory. The force trajectory comes from the solution of the following differential equation:

$$K_A \ddot{P}_f + K_V \dot{P}_f = \Delta_f$$

Note that the RHS of the equation above is the force error and K_A & K_V are positive definite, diagonal matrices. The equation above is realized using the filter shown in Figure 28:

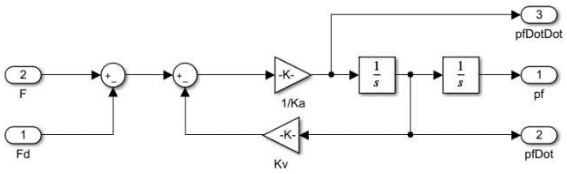

Figure 28 - Parallel Control, Force Trajectory Filter

The filter was implemented with the following constant matrices:

$$K_V = \begin{bmatrix} 95 & 0 & 0 \\ 0 & 95 & 0 \\ 0 & 0 & 95 \end{bmatrix}, \quad K_A = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{bmatrix}$$

Because the force trajectory is generated in real-time using a double integrator, force control is the dominant contributor to the overall desired EE trajectory. That is to say, the manipulator will prioritize ensuring the correct contact force is made over ensuring the correct Cartesian trajectory is followed. This can be beneficial or detrimental depending on your application. For example, if you are concerned with controlling force along one axis with little regard for your orientation/position in other axes then this algorithm will work really well. If however you need to control force in a certain direction while also controlling the orientation and position of the EE, you could encounter issues where the force component of the trajectory dominates the position component and your EE drifts away from the desired position.

The trade-off between force & position control will be demonstrated in the following example. In the following simulation the manipulator will follow a trajectory where the EE collides with a wall which is connected to the ground through a spring-damper. The wall can move linearly along the task space x-axis. There will be a desired contact force profile which is comprised of three step inputs with different step times. It will be demonstrated how the priority of force control causes the joint positions to veer away from their desired position trajectories. The visualization, force tracking & joint position tracking plots for this scenario can be seen in Figure 29, Figure 30 & Figure 31 respectively.

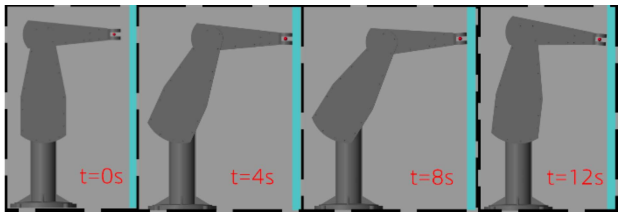

Figure 29 - Parallel Motion/Force Control Different End-Effector Positions

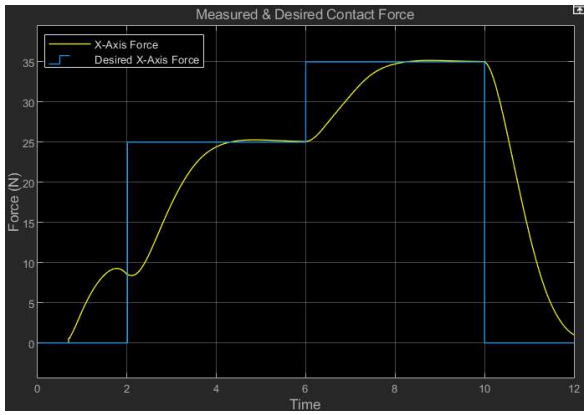

Figure 30 - Parallel Motion/Force Control x-axis Force Tracking

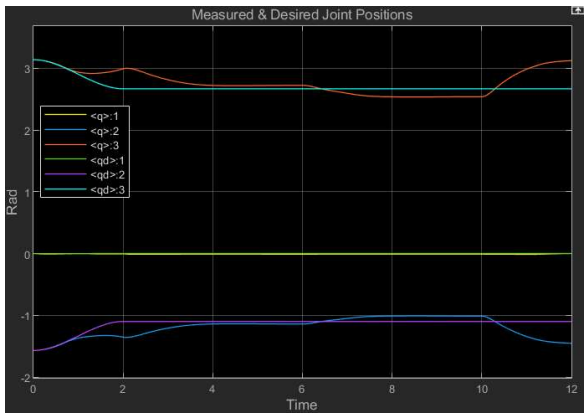

Figure 31 - Parallel Motion/Force Measured & Desired Joint Position

From 0-2 seconds there is a zero desired force so the overall trajectory is purely a positional one. The joint position error, which can be seen happening during this interval is due to the manipulator colliding with the wall. With only a position contribution towards the overall desired trajectory, the EE behaves as a mass-spring-damper between 0 & 2 seconds. At 2s the desired force changes to 25 N and the manipulator responds accordingly. The desired force can also be seen changing at 6 & 10 seconds. The settling time for force tracking is just over 2 seconds for all transients. The joint position errors are the largest after 10 seconds where the desired force between the EE and wall changes to zero and as a result the EE moves away from the wall despite the fact that the desired position trajectory has the EE in a location where it should be making contact with the wall. The force tracking trajectories, in task space coordinate frame, along the task space x-axis, can be seen in Figure 32. The behaviour of the double integrator can be seen in the plot as the acceleration and velocity initially grow much faster than position however they converge to zero whereas the output of the second integrator (position) remains some non-zero value which corrects the position tracking trajectories to ensure force tracking.

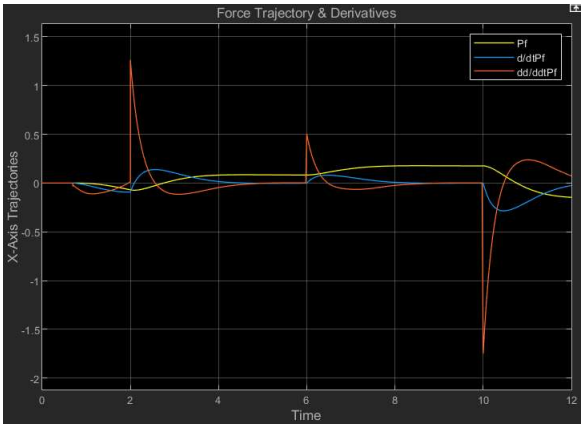

Figure 32 - Force Tracking Trajectories, x-axis

It was observed that the parallel motion/force tracking control behaved as expected, where the manipulator would attempt to ensure both position and force tracking were met however the algorithm would favour force tracking when the two objectives were conflicting. Again this is because the position tracking trajectories were computed before-hand whereas the force tracking trajectories were computed in real-time using a double integrator filter which causes the force trajectory to dominate the overall trajectory due to the nature of an integrator being used to drive the force error to zero.

Hybrid Motion/Force Control

This control algorithm allows for motion control in certain directions and force control in other directions. This is advantageous for certain applications over the parallel motion/force control because you can explicitly define which directions force/motion tracking is enforced without worrying about the force control dominating the desired trajectory and causing the EE to drift from the desired task space position.

To define which directions the manipulator should enforce position/force control we must first choose a frame of reference to define these directions. One of the easiest ways to do this is to attach a frame of reference to the environment which the manipulator will be interacting with, this frame will be denoted by “S”. The attached frame can be seen in Figure 33. Another alternative method would be choosing the EE reference frame as shown in Figure 34.

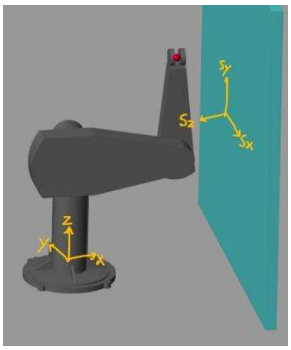

Figure 33 - Reference Frame attached to Environment

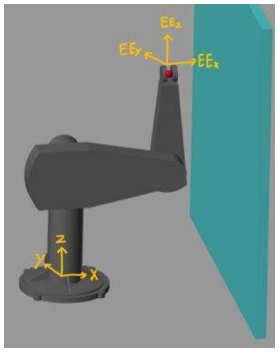

Figure 34 - End-Effector Reference Frame

With either choice in reference frame, it must be specified which axes, in the chosen reference, should obey

position/force tracking. To do this, the position and force specification matrices are introduced. These matrices are used to specify which directions the controller will enforce position/force tracking. An entry of 1 along the first, second and third diagonal entry of a given specification matrix implies enforcement of the corresponding tracking (position or force) along the x, y & z axes of the chosen reference frame.

To investigate this algorithm's effectiveness, we will be concerned with controlling the contact force normal to the same wall seen in the parallel motion/force control. If we choose the reference frame attached to the wall this would define the following position and force specification matrices.

$$S_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, S_f = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Because the control law for this algorithm is specified in the task space coordinate frame, the position/force specification matrices must be transformed into the task space coordinate frame through a change in basis. This can be done using the rotation matrix describing the S frame in the task space coordinate frame. This rotation matrix is as follows:

$$R = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Note that our implementation of the algorithm will follow the convention with the S-frame attached to the environment. If we were to choose the reference frame attached to the EE then the rotation matrix above would be replaced by the rotation matrix relating the EE frame to the base frame and would be dependant on joint positions.

The task space specification matrices were then calculated as follows:

$$\Omega_p = RS_pR^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\Omega_f = RS_fR^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The complete control law for hybrid motion/force control is given by:

$$\tau = J^T * \left(H_p * \Omega_p u_p + C_p(q, \dot{q}) * \dot{p} + G_p(q) + \Omega_f u_f\right)$$

Where u_P & u_f are the position and force tracking control laws. They are given as follows:

$$u_p = \ddot{p}_r + K_d * (\dot{p}_r - \dot{p}) + K_m * (p_r - p)$$
$$u_f = f_d + K_f * (f_d - f)$$

Where K_f is a positive definite diagonal matrix of appropriate gains.

This algorithm is harder to implement than the parallel motion/force control algorithm because you must have knowledge about when contact is made/to-be made with the environment. This is required because as the manipulator moves through free space there are natural force constraints acting along all three cartesian directions. If the force specification matrix has a 1 in the diagonal entry corresponding to a direction where there is a natural force constraint then the manipulator will unsuccessfully try to regulate force in that direction and become unstable. To avoid this issue the force specification matrix must only include a 1, in the appropriate location along the diagonal once the manipulator is close to contacting the environment. In the simulation performed for this algorithm, force tracking was only engaged once the manipulator was in contact with the wall. This simulation was run with the following parameters.

$$K_p = \begin{bmatrix} 500 & 0 & 0 \\ 0 & 500 & 0 \\ 0 & 0 & 500 \end{bmatrix}, K_d = \begin{bmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 0 & 50 \end{bmatrix},$$
$$K_f = \begin{bmatrix} 30 & 0 & 0 \\ 0 & 30 & 0 \\ 0 & 0 & 30 \end{bmatrix}$$

The force tracking performance can be seen in Figure 35.

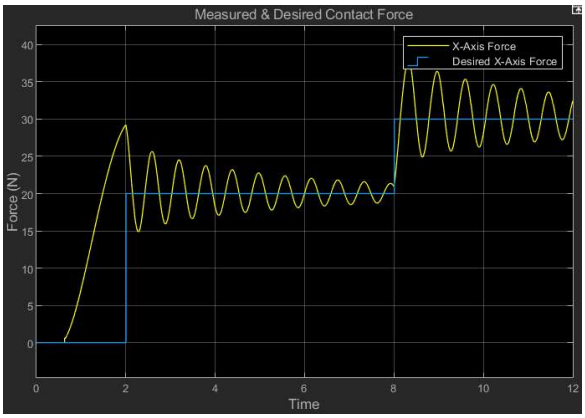

Figure 35 - Hybrid Motion/Force Control, Force Tracking

The heavy oscillation in the plot demonstrates a flaw in this control algorithm when there are more constraints on the desired motion than degrees of freedom in the manipulator. The PUMA 560 (link1->3 for this project) can not move the EE radially outwards from the task space origin independently of the z position *while maintaining a constant orientation*. The desired trajectory required the 3rd link of the manipulator to remain horizontal. The force tracking law is commanding the x-position of the EE to increase in order to satisfy the desired force, however this is simultaneously decreasing the z-position which contradicts the desired z-position. Because of this, there are two competing tracking laws. This is best demonstrated when plotting the x-force error & z-position error on the same graph. This can be seen in Figure 36.

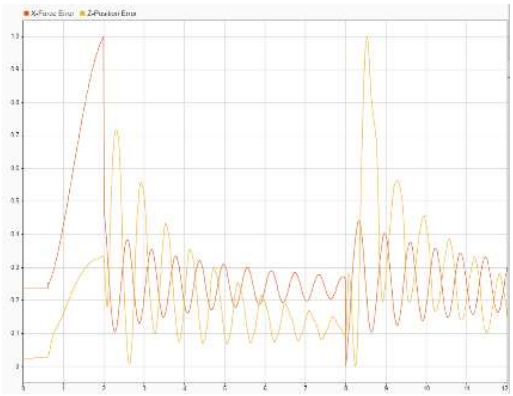

Figure 36 - Hybrid Motion/Force Control Normalized Position & Force Errors

The position and force error are nearly 180 degrees out of phase. Because of this, the competing tracking laws introduce oscillation into the system. To reduce the oscillation, a new force tracking law was proposed.

$$u_f = f_d + K_f * (f_d - f) - K_{d_f} * \dot{p}$$

This force tracking law is the same as before except with the addition of a damping term, porportional to the task space velocity. Two additional simulations were run with the new proposed force tracking law. Different values for the force damping matrix, K_{df}, were chosen to evaluate the value's effect on the overall system behaviour. The two different values chosen were:

$$K_{d_f} = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix} \& K_{d_f} = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 1000 \end{bmatrix}$$

The force tracking for both choices in the force damping matrix can be seen in Figure 37.

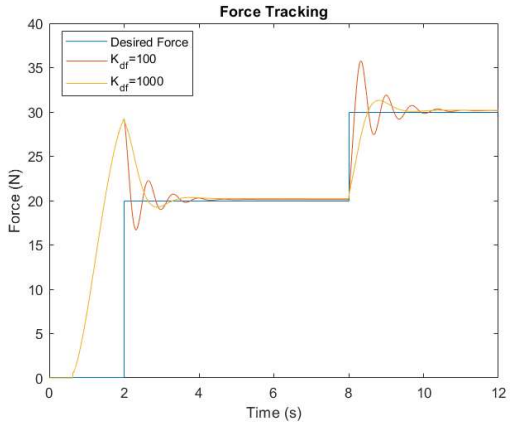

Figure 37 - Hybrid Motion/Force Control, New Proposed Force Tracking

The result of adding the damping term greatly improved the oscillations and made an improvement to the settling time, though to a lesser degree. It was also observed that the larger of the two K_{df} values resulted in a better reduction in oscillation. Through trial and error it was discovered that K_{df} could be increased to approximately 1400 before the oscillation began to grow unbounded.

When compared to parallel motion/force control there are a couple key differences. Hybrid control results in a faster response in the system though this comes at the cost of overshoot and oscillation. Additionally, because the force tracking in the hybrid control has no integral term, the force error is not driven exactly to zero as in the case with the parallel control scheme. When K_{df}=100 & 1000 the steady state errors were 0.1215 N & 0.2363 N respectively. While these errors are quite small it is worth mentioning as certain applications may require more precise force control. Lastly it was noted that depending on the manipulator's degrees of freedom, there may be situations where the position and force tracking laws are competing with each other.

Conclusion

This paper presented several different control algorithms which were used to control different variations of the PUMA 560 manipulator. It was shown that certain algorithms required exact model parameters for accurate tracking of position/force while other algorithms can handle model uncertainty. Comparisons were also made, when possible, between different algorithms and suggestions were made as to scenarios where one algorithm may be advantageous compared to another.

References

[1] K. Aström, “History of Adaptive Control,” in *Encyclopedia of Systems and Control*, 2014.

[2] F. M. Zaihidee, S. Mekhilef, and M. Mubin, “Robust speed control of pmsm using sliding mode control (smc)-a review,” *Energies*. 2019, doi: 10.3390/en12091669.

[3] P. A. Ioannou *et al.*, “L1-Adaptive control: Stability, robustness, and interpretations,” *IEEE Trans. Automat. Contr.*, 2014, doi: 10.1109/TAC.2014.2318871.

[4] W. Slotine, J.J.E., and Li, *Applied Nonlinear Control*. Prentice Hall, 1991.

Appendix

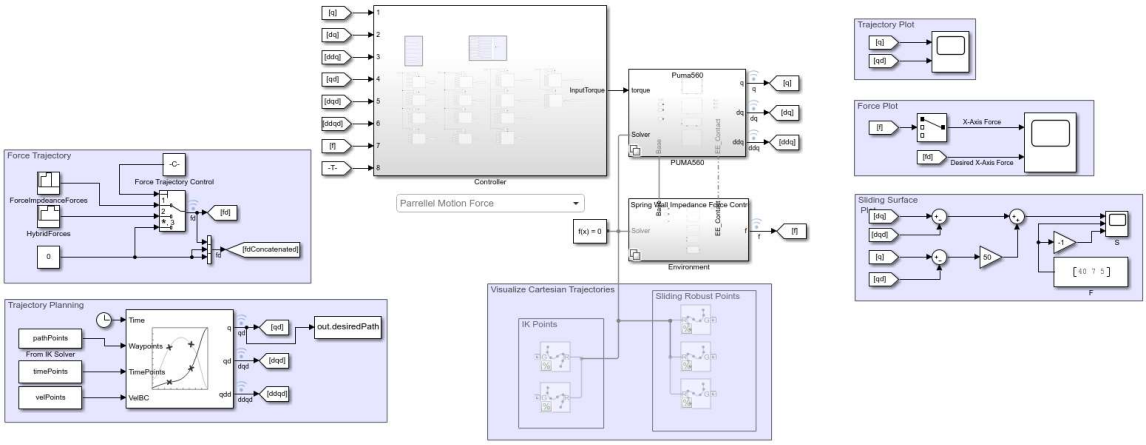

Figure 38 - High Level Simulink Block Diagram

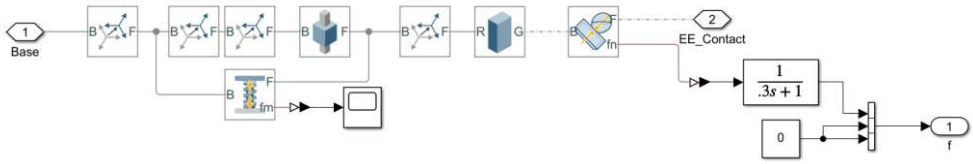

Figure 39 - Manipulator Environment Block Diagram

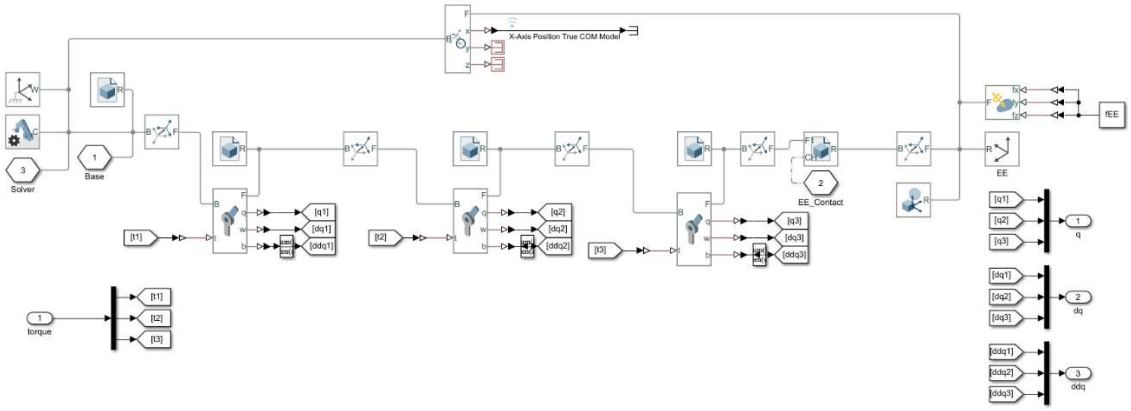

Figure 40 - PUMA 560 Simscape Multibody Block Diagram