Model Predictive Control - EL2700

KTH ROYAL INSTITUTE OF TECHNOLOGY

Assignment 3: Model Predictive Control

Jonne van Haastregt - 20010713-3316

Miguel Garcia Naude - 19980512-3537

Due Date: September 30, 2022

Submitted: September 28, 2022

Question 1

We study the influence of the control horizon N, by computing the control invariant set with differing control horizon, namely $N = \{5, 10, 20\}$. The result plots in figure 1 to 3 from running our code are shown below. One can quite clearly see that for increased control horizons result in larger invariant sets. This is logical, since the control is constrained, far-away initial states need more control steps in order to reach the final state set. Furthermore, since the Y-position is constrained to [-0.1, 0.1], the set only grows in X direction to not break this constrain.

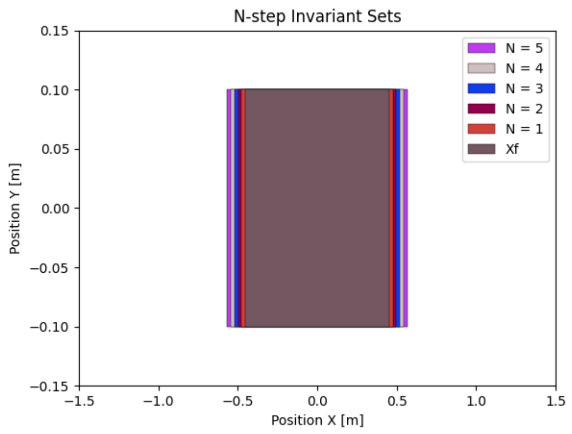


Figure 1: CIS resulting from 5-step

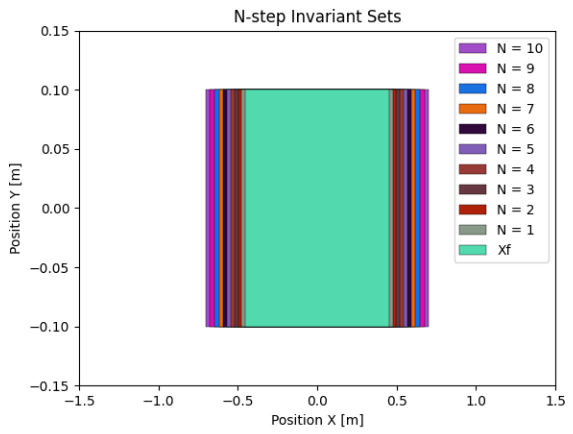


Figure 2: CIS resulting from 10-step.

We can also see from figures 4 to 6 that when we use the zero terminal constraint the control invariant set increases in size when the control horizon increases. However, this time the set grows in both X and Y because the final set is not yet at the boundaries of the constraints.

Question 2

Next, we investigated the influence of the control constraints on the control invariant set. We varied the original control constraints and held the control horizon constant at

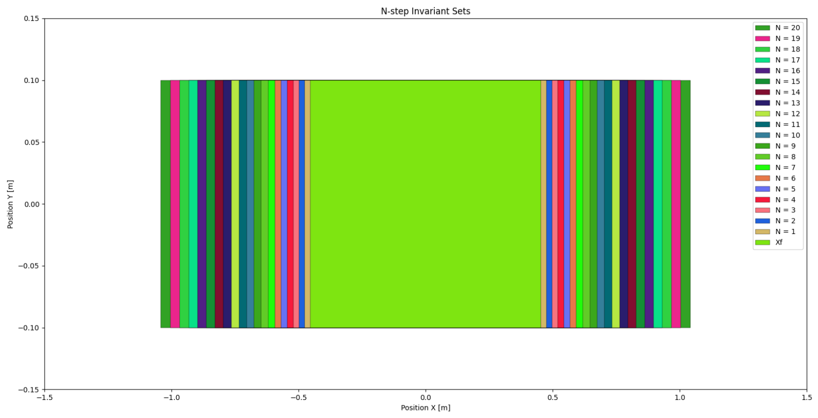


Figure 3: Translation Control Invariant Set (CIS) resulting from 20-step control horizon.

$N = 5$, such that the new control constraints could be expressed as: $x \cdot u_{max} = -x \cdot u_{min}$, where $x \in \{2, 3\}$. The results can be seen in figures 7 and 8. Once again, it makes sense that the CIS is growing, because with looser control constraints there can be more control performed per time step, and thus the final state set can be reached more easily. However as can be seen, the constraint of x is reached and thus the CIS becomes the same as the sets satisfying the positional constraints.

Question 3

How the state constraints are set

The state constraints are inequalities. To add them, a vector is created for the left-hand side of the equation, as well as two vectors for the bounds (and upper bound and a lower bound). The constraints are appended to these vectors one by one. The left-hand side of the equations are used to build the NLP solver, and the bounds are used as argument for solving.

```
### Adding the constraints using the three vectors, done for each time step
```

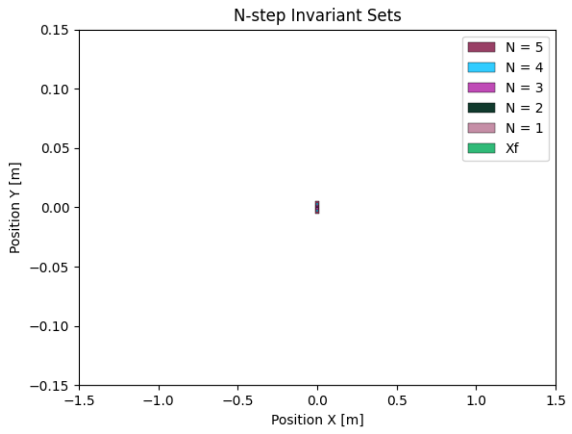


Figure 4: CIS resulting from 5-step

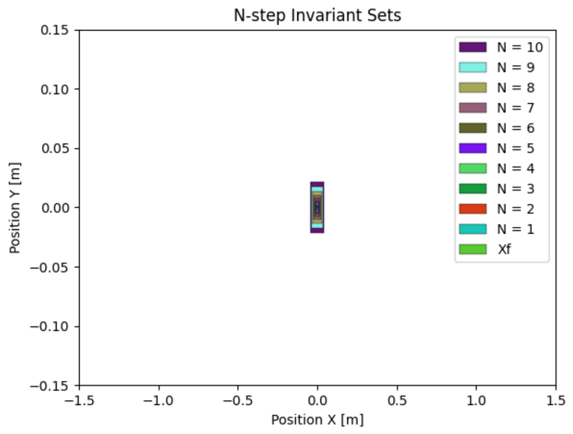


Figure 5: CIS resulting from 10-step.

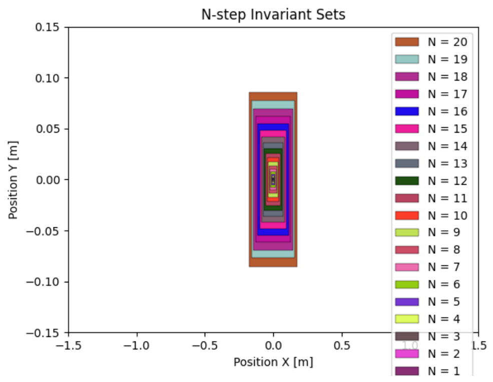


Figure 6: Translation Control Invariant Set resulting from 20-step, SET_TYPE = ”zero”

```
if xub is not None:
        con_ineq.append(x_t)
        con_ineq_ub.append(xub)
        con_ineq_lb.append(np.full((self.Nx,), -ca.inf))
if xlb is not None:
        con_ineq.append(x_t)
        con_ineq_ub.append(np.full((self.Nx,), ca.inf))
        con_ineq_lb.append(xlb)
```

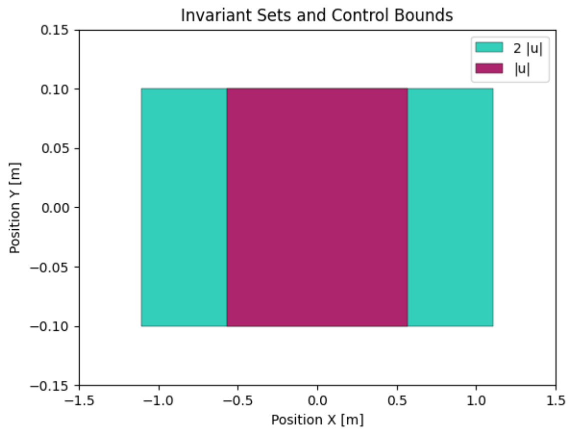


Figure 7: CIS for $2 \cdot u_{max} = -2 \cdot u_{min}$

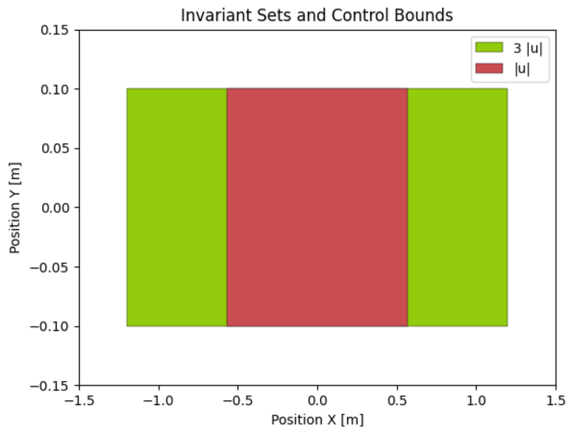


Figure 8: CIS for $3 \cdot u_{max} = -3 \cdot u_{min}$

```
### Using the constraints for setting up the NLP
    con = ca.vertcat(*con_eq, *con_ineq)
    self.con_lb = ca.vertcat(con_eq_lb, *con_ineq_lb)
    self.con_ub = ca.vertcat(con_eq_ub, *con_ineq_ub)

    # Build NLP Solver (can also solve QP)
    nlp = dict(x=opt_var, f=obj, g=con, p=param_s)

### Using the bounds in the solver
    args = dict(x0=self.optvar_init,
                lbx=self.optvar_lb,
                ubx=self.optvar_ub,
                lbg=self.con_lb,
                ubg=self.con_ub,
                p=param)

    sol = self.solver(**args)
```

How the object function is formulated

The object function is taken using the following formulas for the cost functions:

```
self.running_cost = ca.Function('Jstage', [x, Q, u, R],
                                [x.T @ Q @ x + u.T @ R @ u])


self.terminal_cost = ca.Function('Jtogo', [x, P],
                                 [x.T @ P @ x])
```

Then for each time step, the objective variable adds the running cost of that time step until it is at the end of the horizon. Then the terminal cost is added. Since this is all symbolic math, in the end there will be one symbolic function that can compute the cost.

How the terminal constraint is set

The terminal constraint is also added to the same inequality vectors as for the state constraints. Note that the terminal constraint is a polytope, so we can write this as an inequality as follows:

```
if terminal_constraint is not None:
        H_N = terminal_constraint.A
        H_b = terminal_constraint.b
        con_ineq.append(H_N @ opt_var['x', self.Nt])
        con_ineq_lb.append(-ca.inf * ca.DM.ones(H_N.shape[0], 1))
        con_ineq_ub.append(H_b)
```

What the variable param_s holds

The variable param_s holds the initial parameters (the 's' is probably for 'start'). This includes the initial state, the reference at t0, and the control input at t0.

Why we only select the first element of the control prediction horizon in the mpc_controller method

This has to do with how MPC works. It is a receding horizon control, so the point is to keep the horizon shifting forward in time. Then, each time step you would solve the full control sequence, but just the first element is needed, and then the control sequence can be solved again for a new horizon that is shifted a timestep foreward as well.

Question 4

For the following parts we simulated the Astrobee with MPC. Setting the variables as instructed and with a receding horizon length of $N = 10$, we first compared the system performance with different input constraints. A table comparing the results as well as Figures 9 and 10 displaying results of the simulation are shown below for the original input constraints case as well as when the input constraints are set to 3 times the original magnitude.

Experiment	**Energy Used**	**Position Integral Error**	**Attitude Integral Error**
u$_{max} = u_{min} = u_{lim}$	156.197	3.94	0.902
u$_{max} = u_{min} = 3 \cdot u_{lim}$	160.512	3.557	0.882

Table 1: Table of results of two simulations with differing input constraints.

We can appreciate from the table that the results regarding the Astrobee performance are clearly better when the inputs are less constrained (smaller errors). However, as a consequence we can see that a higher amount of energy is used, likely due to larger control inputs being implemented, which in the case where energy wants to be conserved can be quite undesirable.

Question 5

We noticed that when we set the terminal set $\chi_f = \{0\}$, an infeasible problem was detected. When we set the terminal set $\chi_f = \chi_f^{LQR}$, the MPC solver is able to execute a solution to the feasible problem. Comparing these results with question 1, we can justify that the problem should be infeasible for when the terminal set is set to $\chi_f = \{0\}$ because the initial position is outside of the control invariant set for a control horizon of $N = 10$. This can be noted in Figure 5. As for the case where the terminal set is set to $\chi_f = \chi_f^{LQR}$, the initial position and attitude are both in the respective control invariant sets, and hence we have a feasible problem.

Question 6

Changing the controller horizon to $Horizon = 50$ and keeping $\chi_f = \{0\}$ the problem now became feasible. Running the simulation presented us with the following results shown in Table 2. These results are similar to those obtained for the case where $\chi_f = \chi_f^{LQR}$ and $Horizon = 10$, which are shown in Table 1. However, it was noticed that the overall computation time was significantly longer for the case where the MPC Horizon was set to 50, now at around 8ms for each time step. A longer computational time can be undesirable, because the computational time should fit in the sampling time. If the sampling time gets to big, this could pose other problems in the system dynamics such as stability.

Experiment	**Energy Used**	**Position Integral Error**	**Attitude Integral Error**
$\chi_f = \{0\}$, N = 50	156.204	3.941	0.902

Table 2: Results of simulation using $\chi_f = \{0\}$ and $N = 50$

Question 7

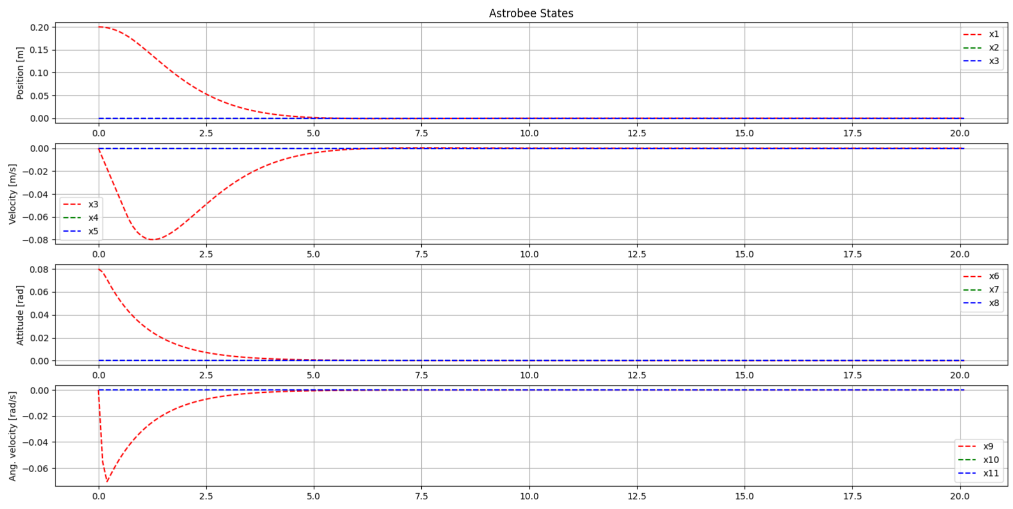
To Investigate the effect of the tuning parameters on the performance of the controller, we altered the Q and R matrix one at a time. A summary of the simulation results for each scenario is shown in Table 3.

Experiment	**Energy Used**	**Pos. Integral Error**	**Attitude Integral Error**
Q = eye(12), R = eye(6)	58.688	10.074	0.961
Q = eye(12), R = 10*eye(6)	30.900	16.846	1.165
Qvel += 100, R = eye(6)	27.309	19.077	7.033
Qpos += 100, R = eye(6)	183.361	3.700	0.446
Q = 101*eye(12), R = eye(6)	156.432	3.937	0.902

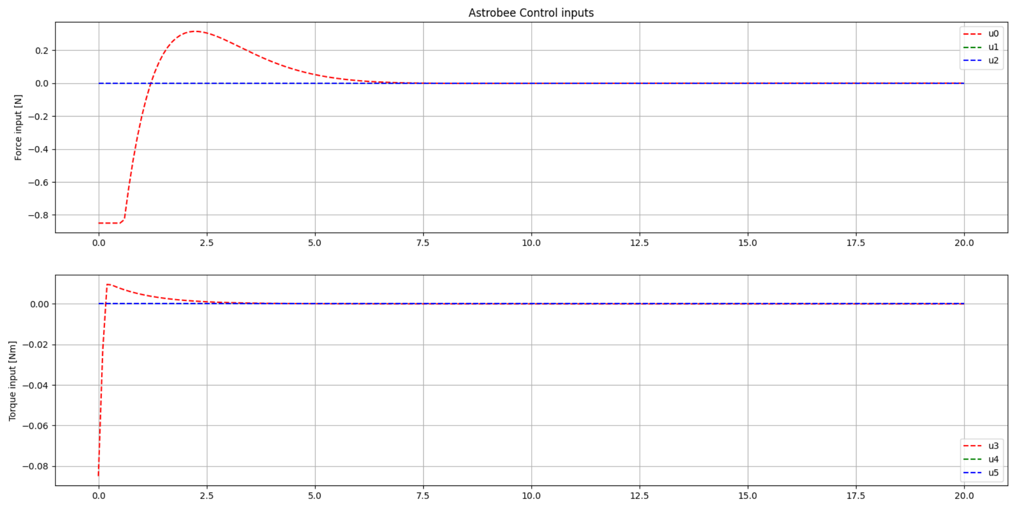
Table 3

The results are further illustrated by Figures 11 through 15. From the table above, we can immediately see that when we increase the R-matrix in the second experiment, we put more emphasis on reducing control input, the result being reduced energy usage at the expense of the final position and attitude integral errors. We can also see that the velocity peaks are smaller than those for the case where Q and R are simply the identity matrix (to be expected as the control inputs should be smaller). When we return R to the identity matrix and increase the values of the Q-matrix corresponding to the linear and angular velocities we see slower convergence of the states towards zero than the previous experiment and as a result higher position and attitude integral errors. The control inputs also converge toward zero much more quickly. For the fourth experiment where the Q-matrix values corresponding to the position states are increased, we see significantly more energy used, which the integral errors decrease to their lowest values. This is to be expected as we put emphasis on sending the position states to zero as quickly as possible. Lastly, we increase all the diagonal elements of the Q-matrix and hence try to put some emphasis on decreasing the cost due to the velocity states. We have slightly less energy used than the previous case and the velocity curves are smoother. The integral error results are still relatively small.

When comparing this to the LQR controller of previous week, we see that it is very similar in behaviour. This is logical, since we are essentially tuning the same weights.

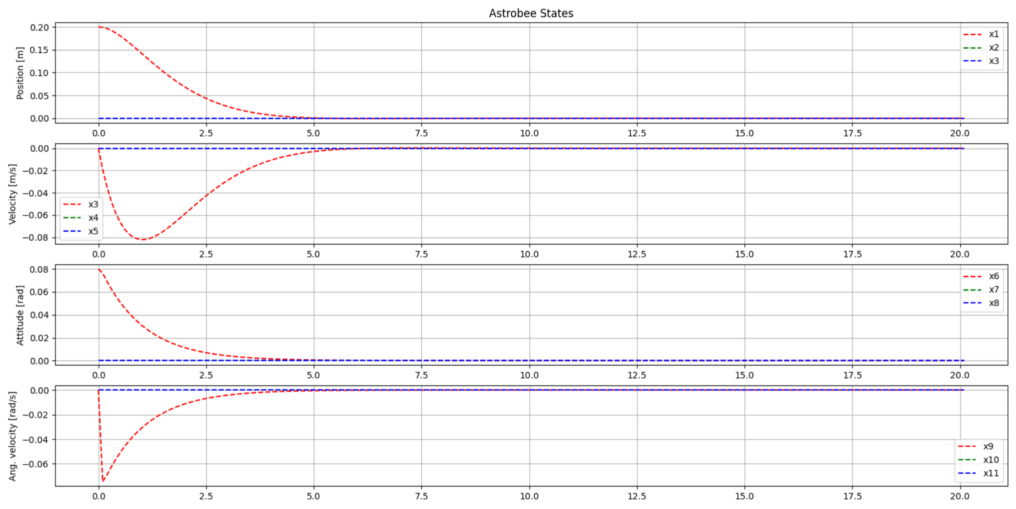


(a) State simulation with original control input constraints

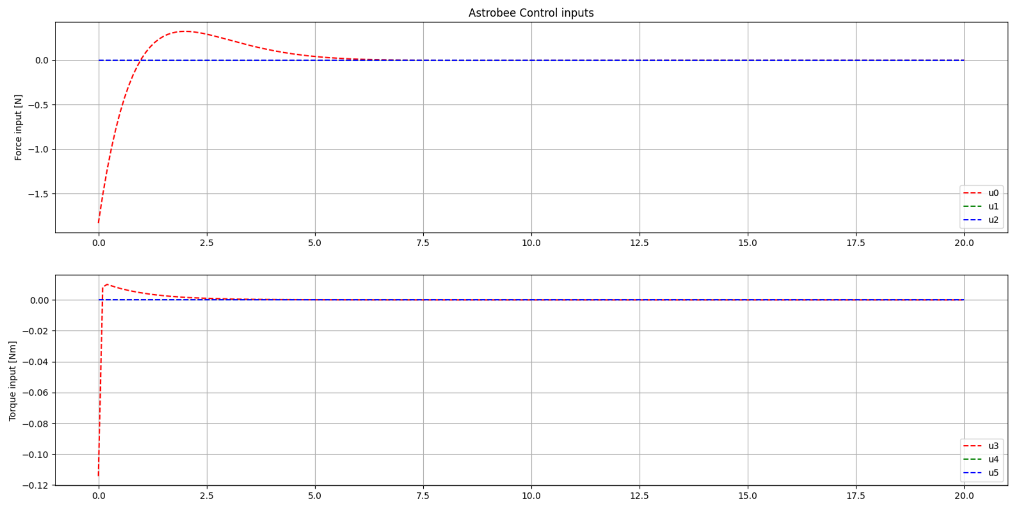


(b) Input simulation with original control input constraints

Figure 9: Original Control Input Constraints

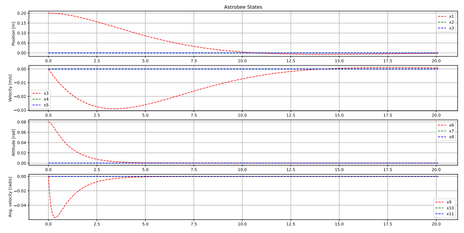


(a) State simulation with control constraints multiplied by 3.

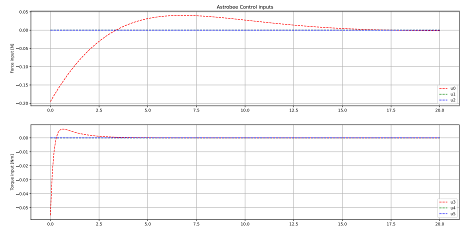


(b) Input simulation with control constraints multiplied by 3.

Figure 10: Control Input Constraints Multiplied by 3

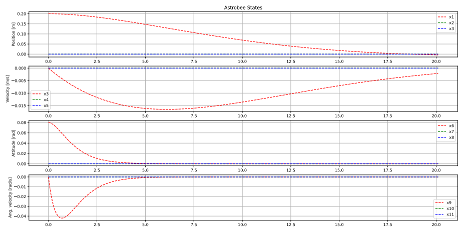


(a) States

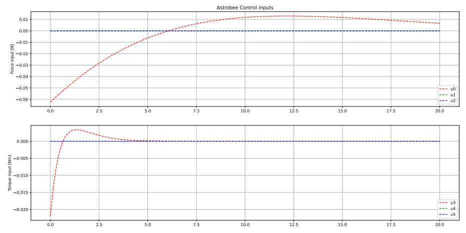


(b) Inputs

Figure 11: Q = eye(12), R = eye(6)

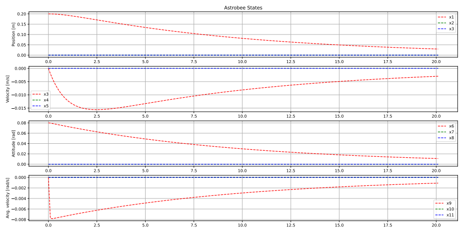


(a) States

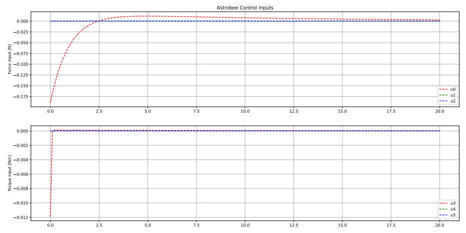


(b) Inputs

Figure 12: Q = eye(12), R = 10*eye(6)

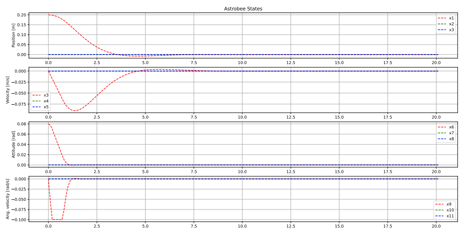


(a) States

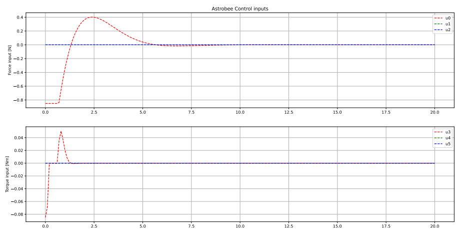


(b) Inputs

Figure 13: Qvel += 100, R = eye(6)

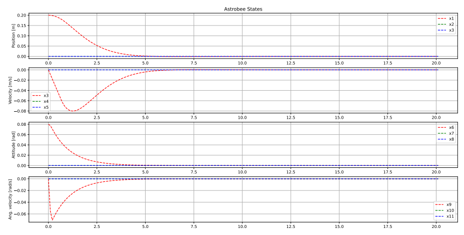


(a) States

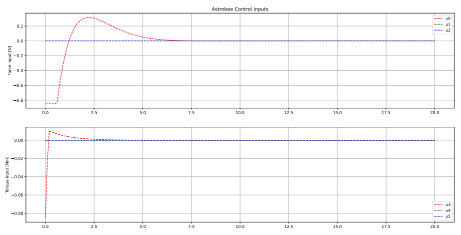


(b) Inputs

Figure 14: Qpos += 100, R = eye(6)



(a) States



(b) Inputs

Figure 15: Q = 101*eye(12), R = eye(6)