AE 4803 Robotics and Autonomy Class Project: Implementing Differential Dynamic Programming, Model Based Control, and Barrier Functions on a Quadrotor's Dynamics

Ashkar Awal, Dalton Ford

December 6, 2022

1 Implementing Direct Differential Programming (DDP)

For a quadrotor with states and dynamics

$$X = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{pmatrix}, \dot{X} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ -\frac{\sin(\theta)\,(f_1+f_2+f_3+f_4)}{m} \\ \frac{\cos(\theta)\,\sin(\phi)\,(f_1+f_2+f_3+f_4)}{m} \\ \frac{f_1\,\cos(\phi)\,\cos(\theta)-g\,m+f_2\,\cos(\phi)\,\cos(\theta)+f_3\,\cos(\phi)\,\cos(\theta)+f_4\,\cos(\phi)\,\cos(\theta)}{m} \\ p + r\,\cos(\phi)\,\tan(\theta) + q\,\sin(\phi)\,\tan(\theta) \\ q\,\cos(\phi) - r\,\sin(\phi) \\ \frac{r\,\cos(\phi)}{\cos(\theta)} + \frac{q\,\sin(\phi)}{\cos(\theta)} \\ \frac{q\,r\,(I_{yy}-I_{zz})+\frac{\sqrt{2}\,l\,(f_1-f_2+f_3-f_4)}{2}}{I_{xx}} \\ -\frac{p\,r\,(I_{xx}-I_{zz})+\frac{\sqrt{2}\,l\,(f_1+f_2-f_3-f_4)}{2}}{I_{yy}} \\ \frac{k_t\,(f_1-f_2-f_3+f_4)}{I_{zz}} \end{pmatrix}, \quad (1)$$

going from an initial state of $(x, y, z) = [-3, -2, -1]$ to final state of $(x, y, z) = [5, 3, 2]$, we implement direct differential programming. We modify the cost function to include a running cost, so we have both Q and Q_f along with R for weighting.

$$Q = \text{diag}(1/10, 1/10, 10, 1, , 1350, 1, 1, 1, 70, 70, 70) \quad (2)$$

$$Q_f = \text{diag}(850, 850, 850, 100, 110, 350, 0, 0, 0, 0, 0, 0) \quad (3)$$

$$R = \text{diag}(.01, .01, .01, .01) \quad (4)$$

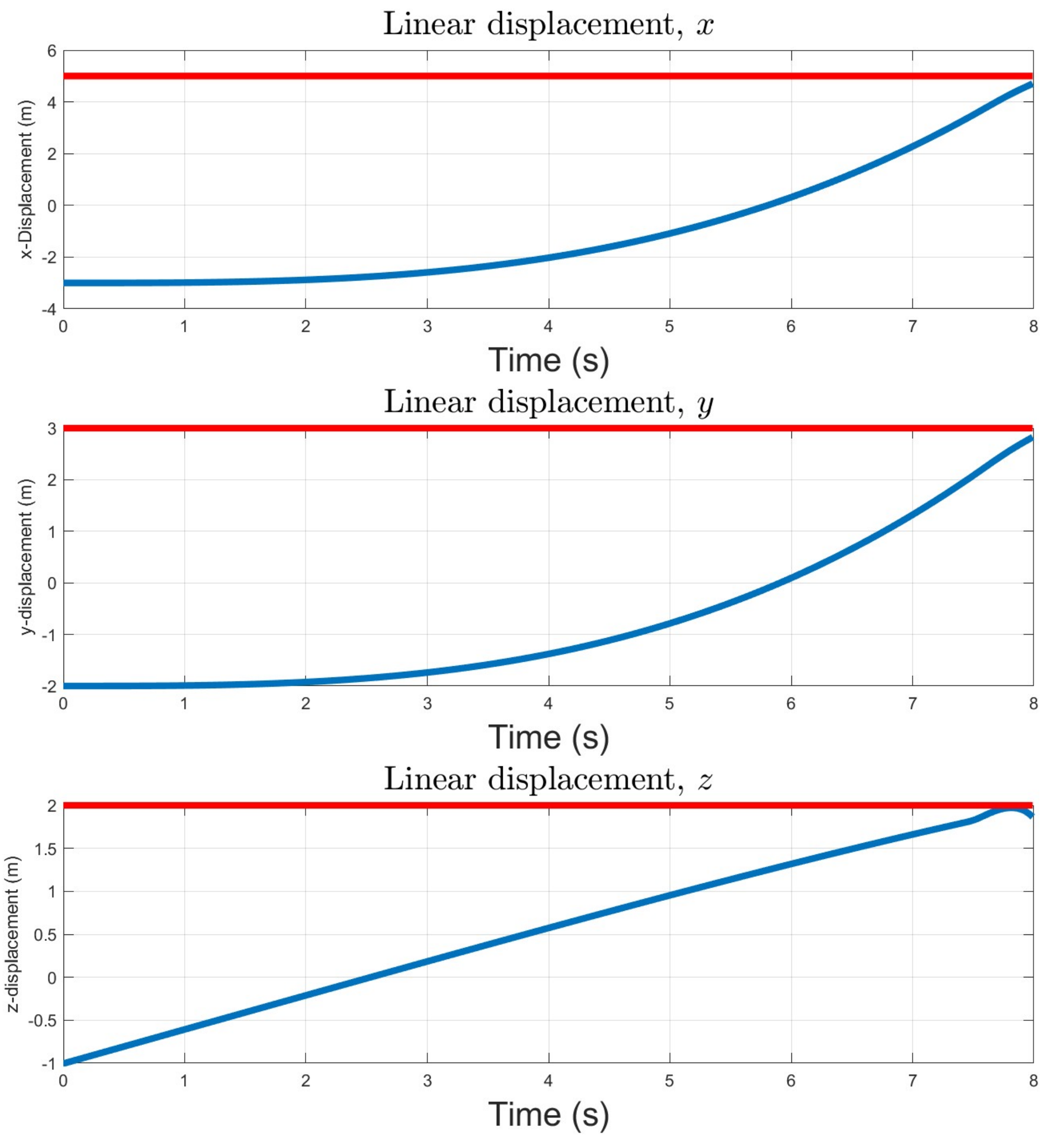

Figure 1: Position of the quad-copter over time.

With this weighting, the quad-copter takes the following path.
The quad-copter takes the following 3D path, as a result of these states and input.

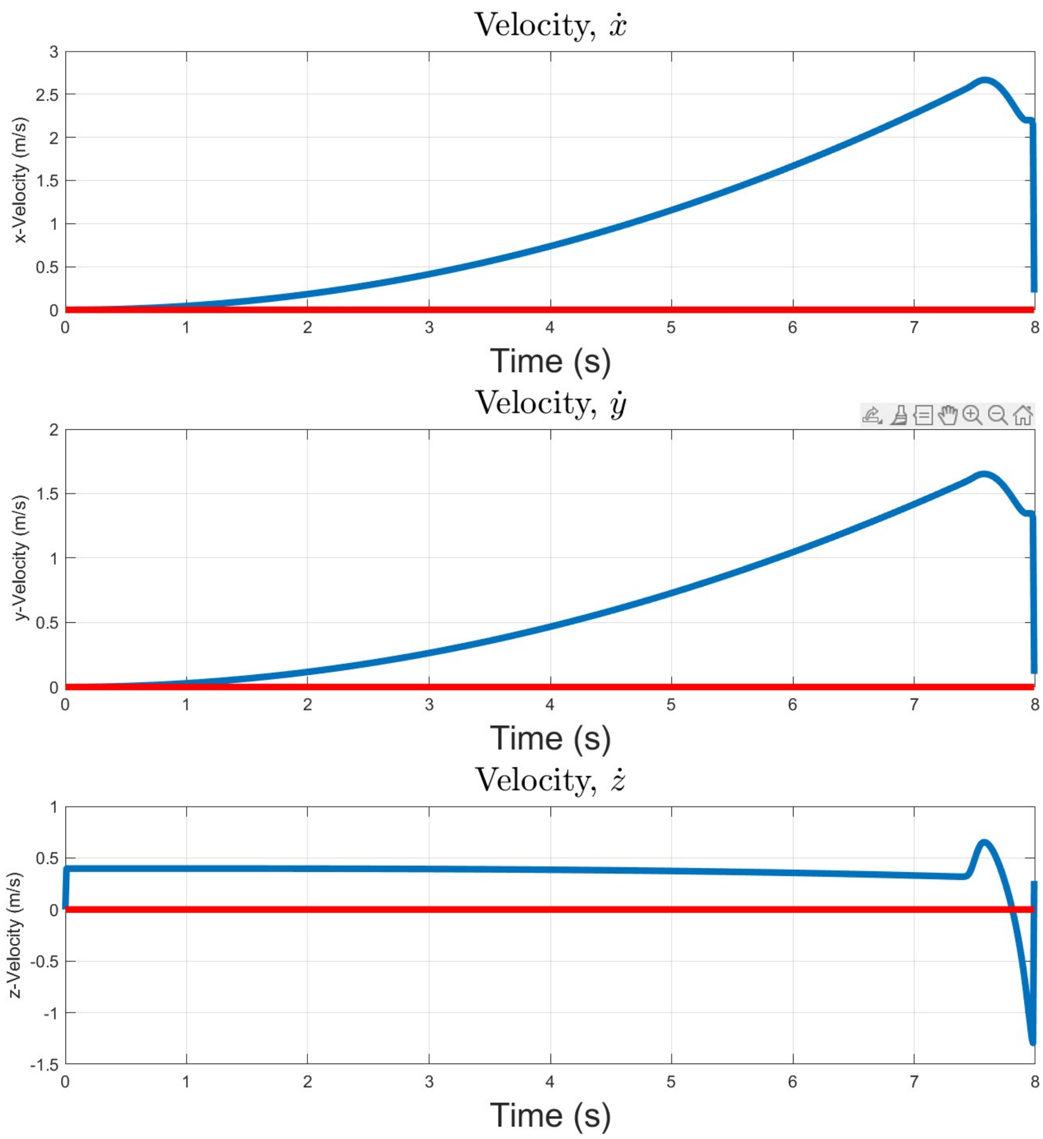

Figure 2: Velocity of the quad-copter over time.

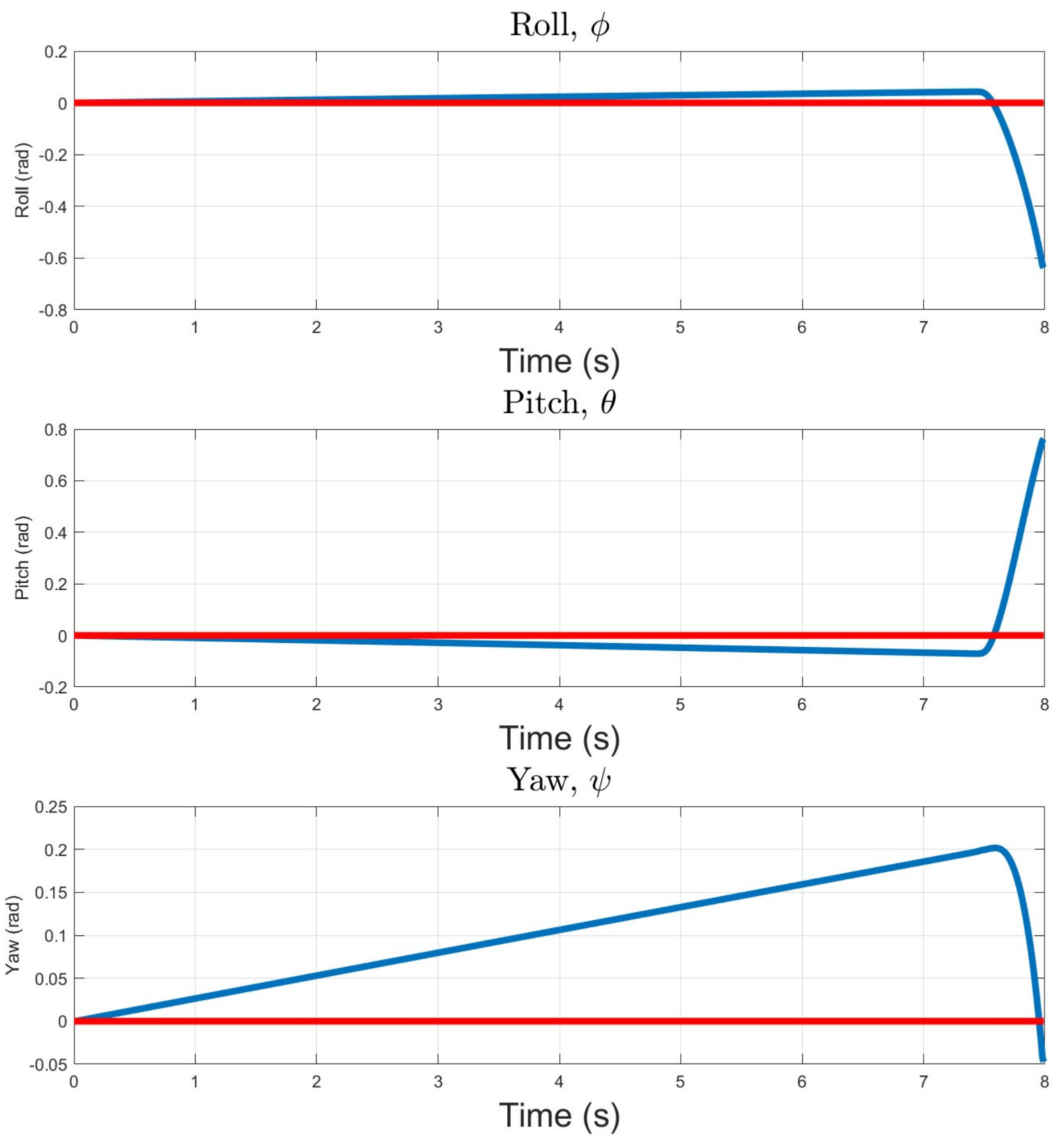

Figure 3: Euler angles of the quad-copter over time.

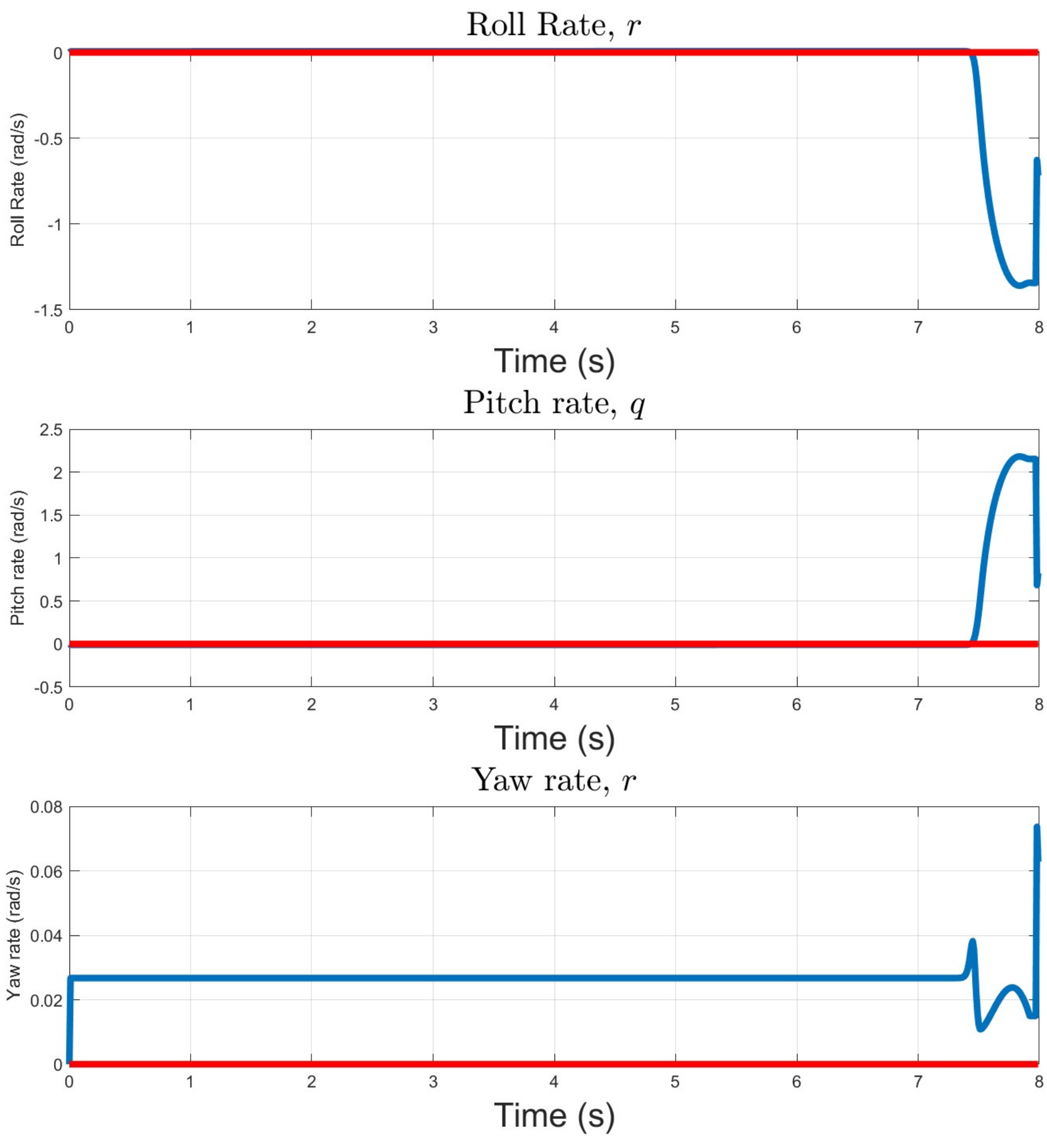

Figure 4: Body rates of the quad-copter over time.

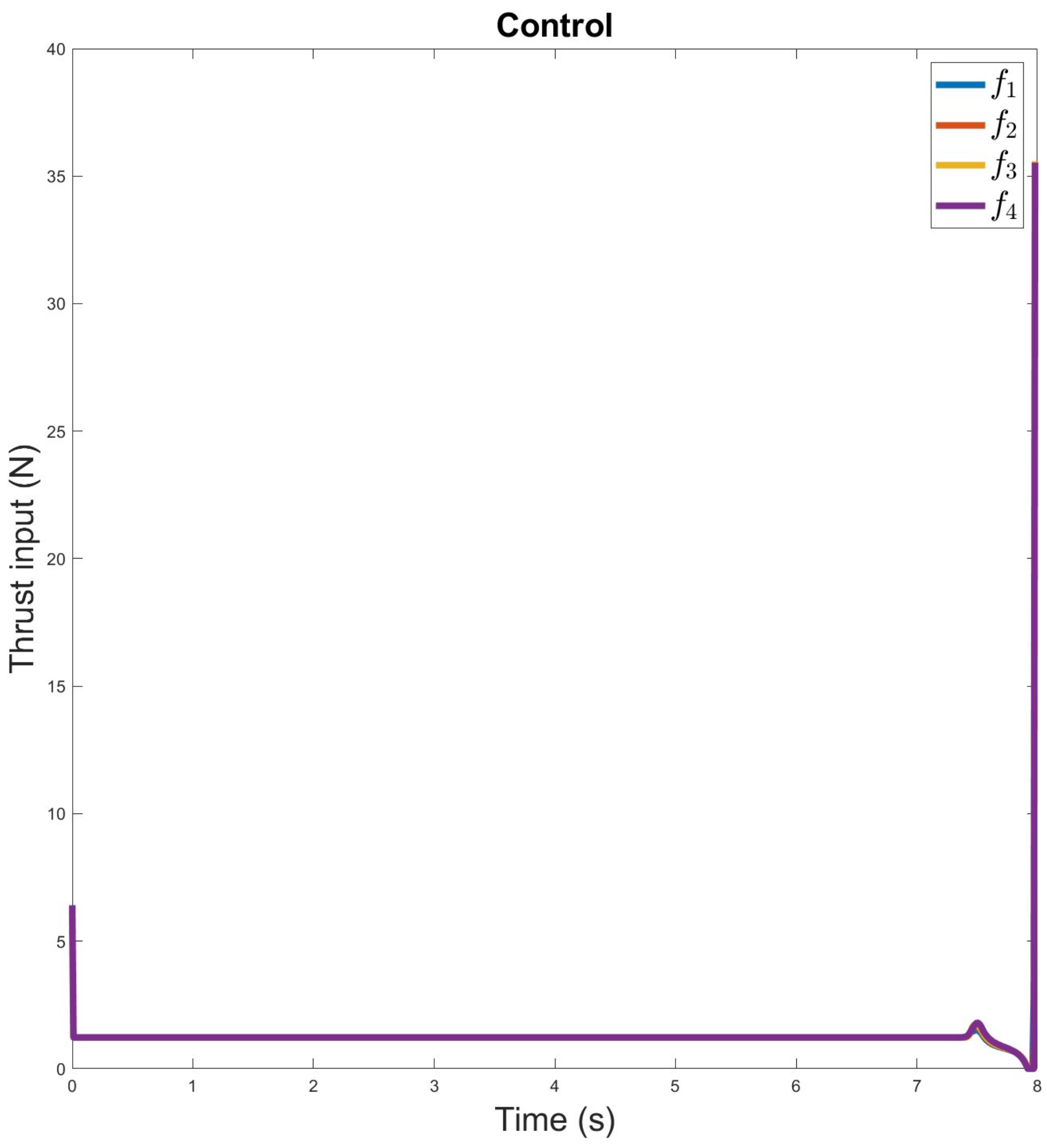

Figure 5: Control force used by the quad-copter over time.

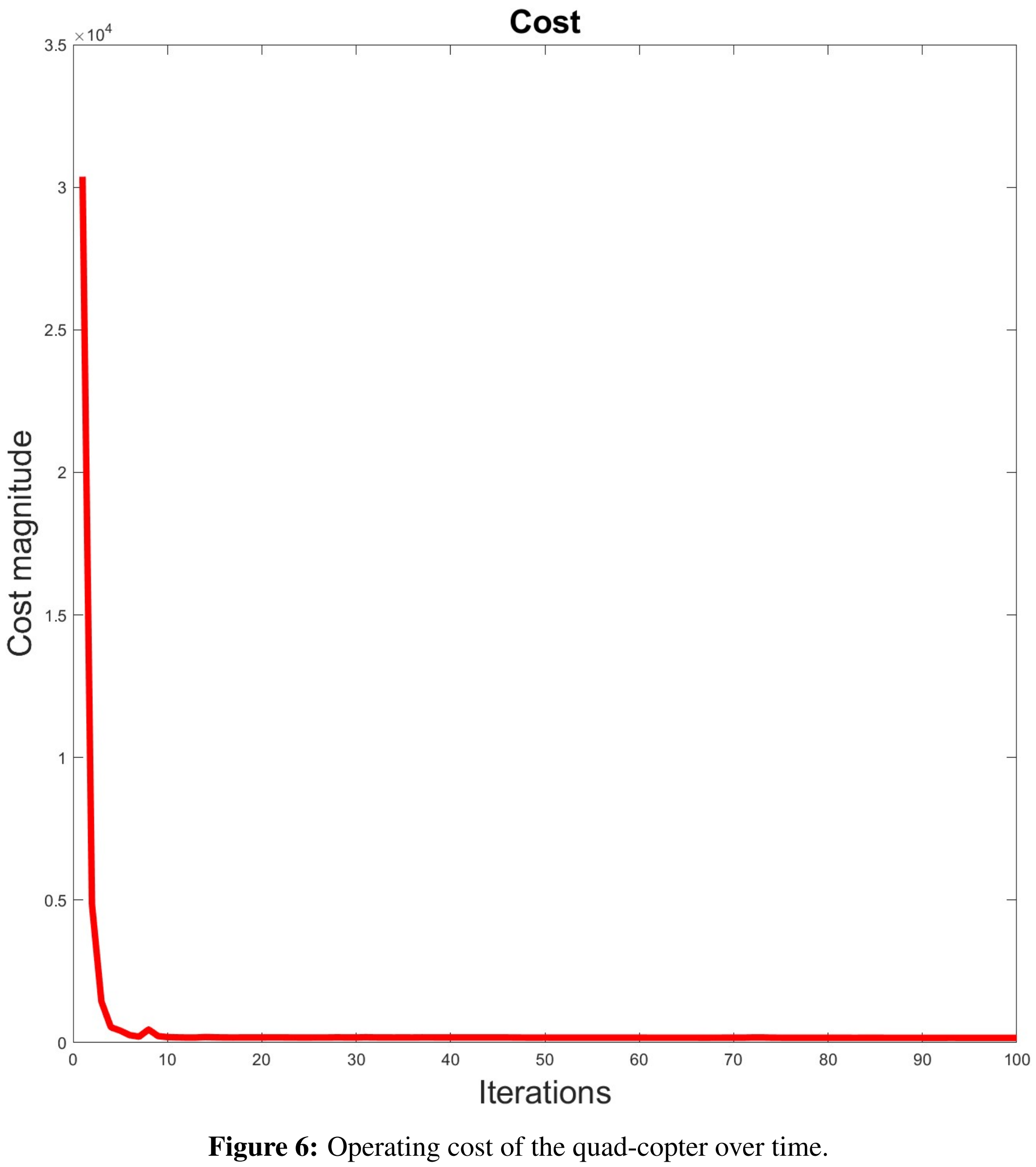

Figure 6: Operating cost of the quad-copter over time.

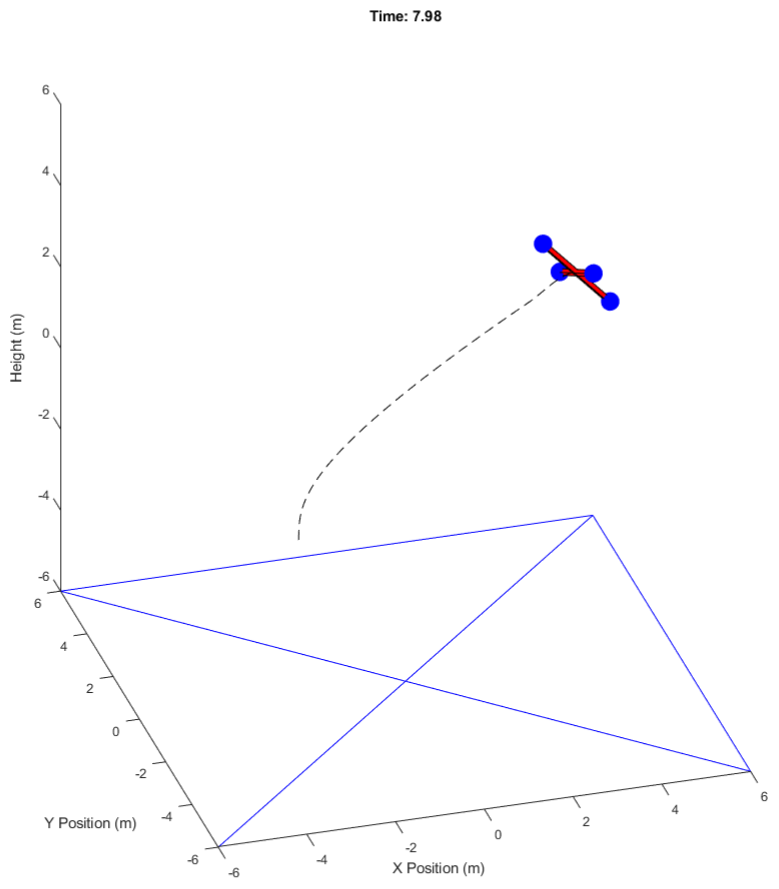

Figure 7: Caption

2 Implementing Model Predictive Control (MPC)

Direct Differential Programming gives a good result. but it can be improved by adding Model Predictive Control. This uses DDP to find paths for sections of the whole horizon, and then use that section to predict the next one. This allows it to use the system's real model to improve the feedback it creates.

The pseudo-code for the algorithm is provided below:

```
iterDDP = 200;
ddpHorizon = 75;
iterMPC = 400;
for i=1: iterMPC
    u_k = fnDDP(iterDDP, ddpHorizon, current_states, x_initial)
    next_state = fnDynamics(current_state,u_k(:,1))
    x_traj = next_state
    current_state = [current_state, next_state]
end MPC loop
```

The following gains were used:

$$\begin{aligned} Q &= \text{diag}([6,\ 6,\ 150,\ 2,\ 2,\ 10,\ 1,\ 1,\ 1,\ 70,\ 70,\ 70]) \\ Q_f &= \text{diag}([500,\ 500,\ 100,\ 10,\ 10,\ 50,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1]) \\ R &= 0.01 * \text{eye}(4) \end{aligned}$$

The function fnDDP was iterated over 200 times, with a mere 75 Horizon for DDP. Running MPC for 4 seconds total, we were able to reach the desires states. The repsonse plots are provided below:

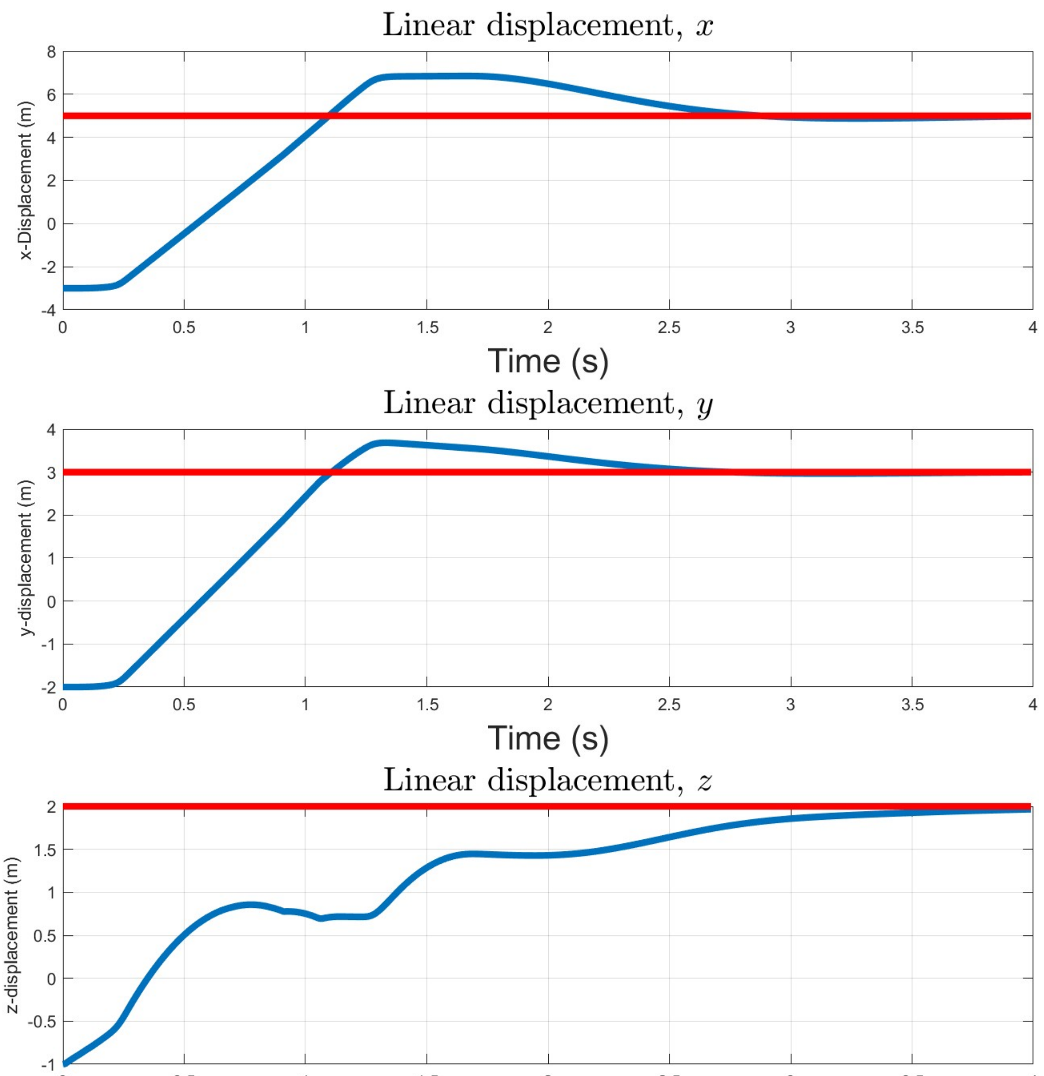

Figure 8: Position of the quadcopter over time, with MBC and DDP.

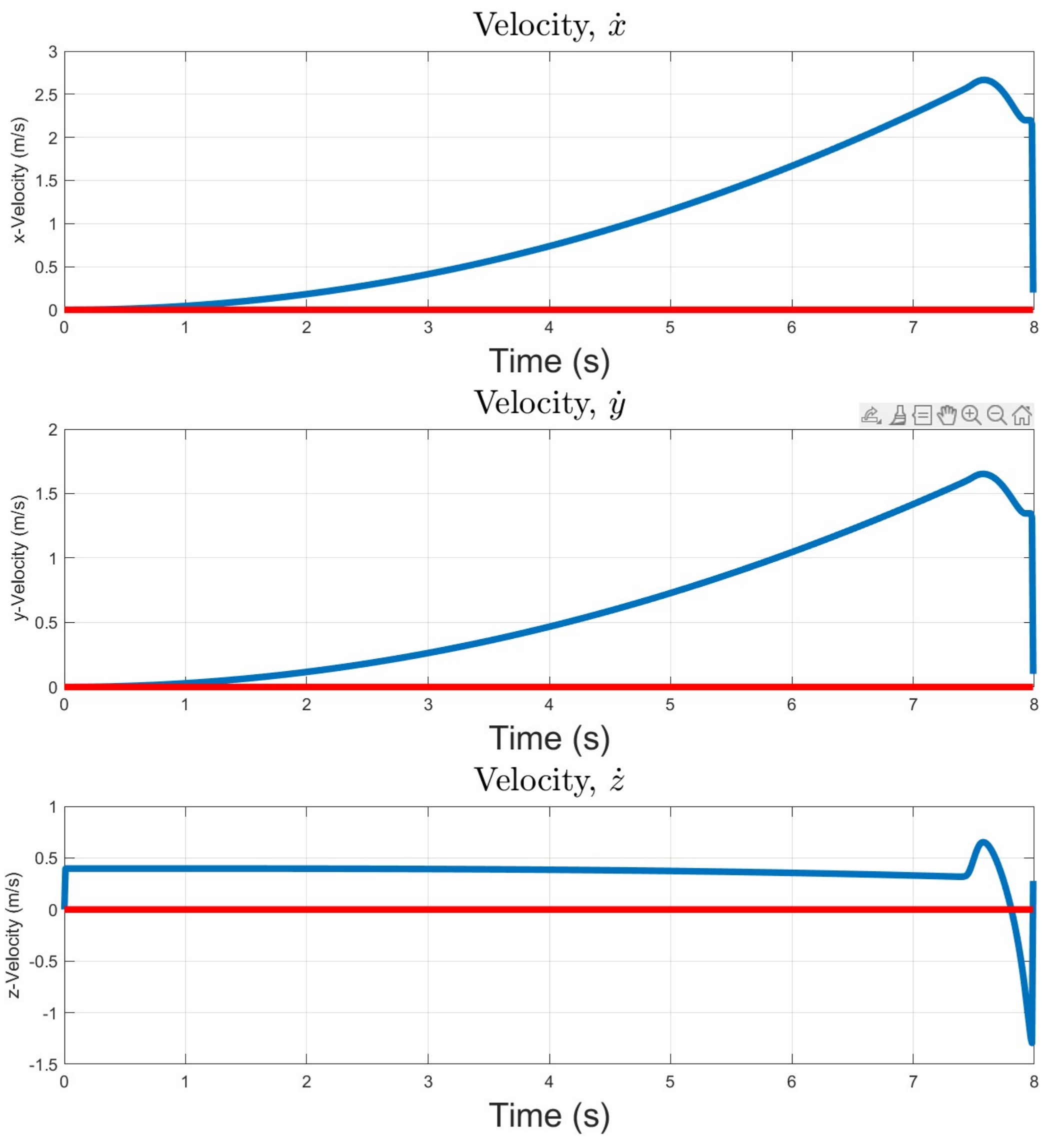

Figure 9: Velocity of the quadcopter over time, with MBC and DDP.

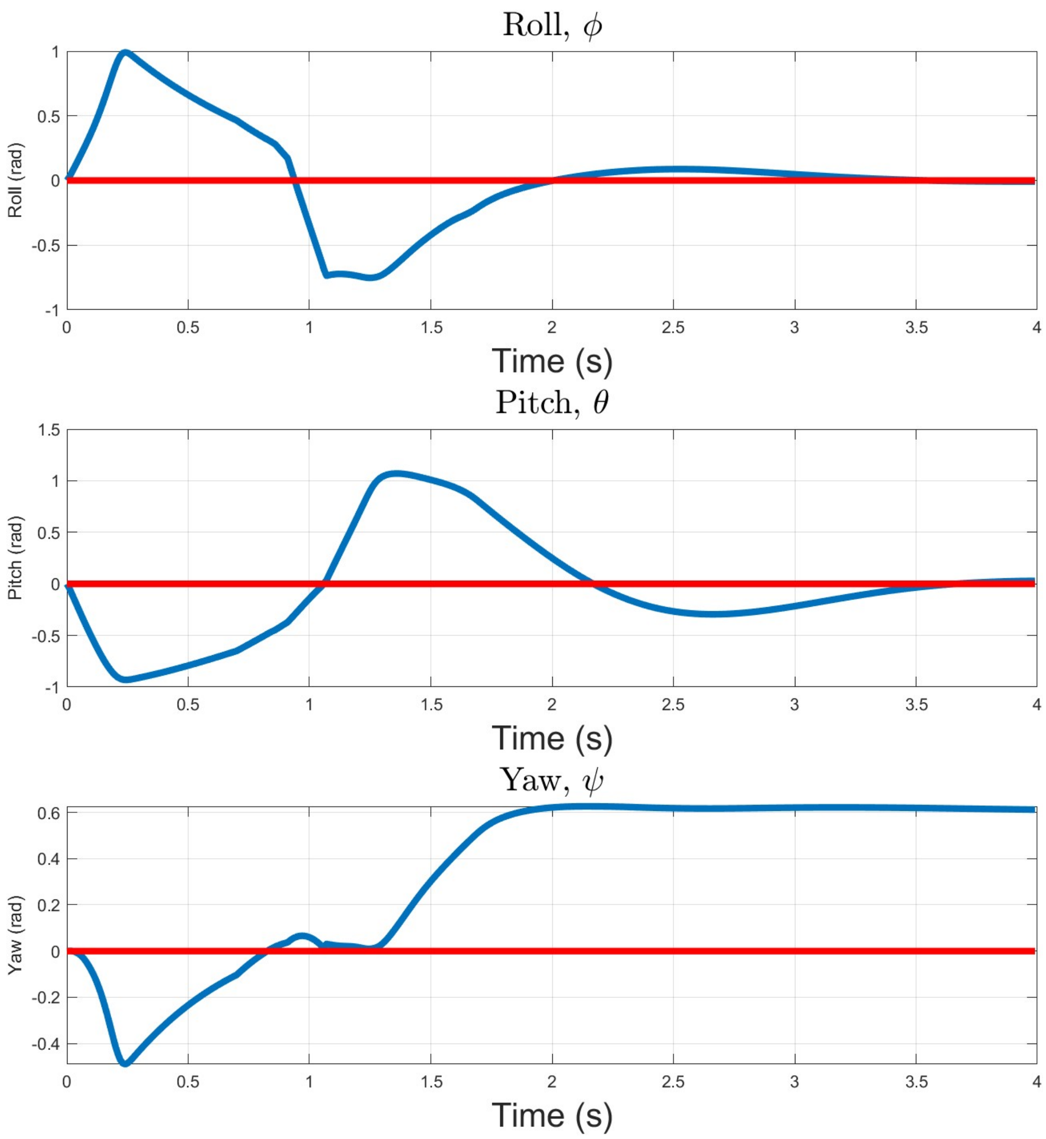

Figure 10: Euler angles of the quadcopter over time, with MBC and DDP.

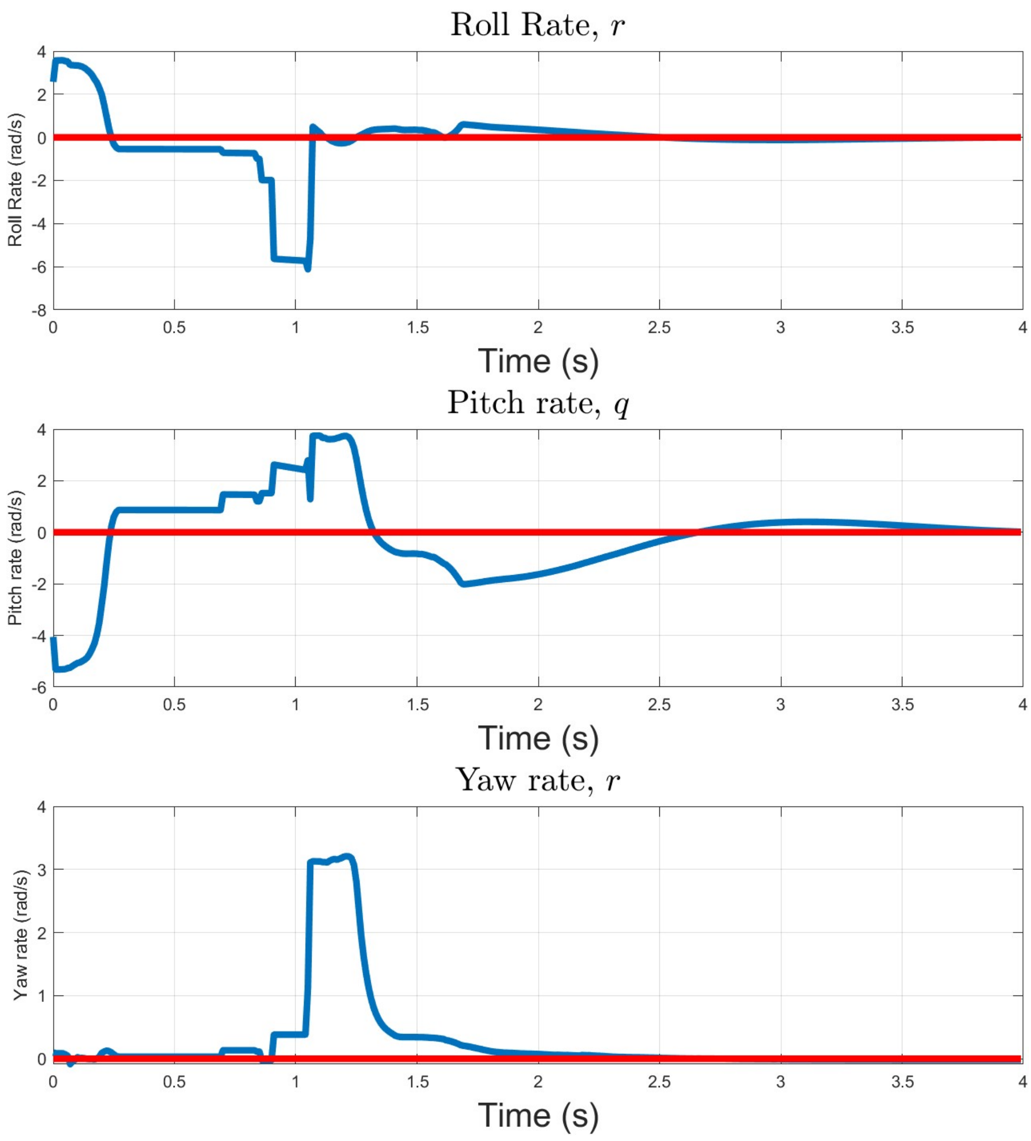

Figure 11: Body rates of the quadcopter over time, with MBC and DDP.

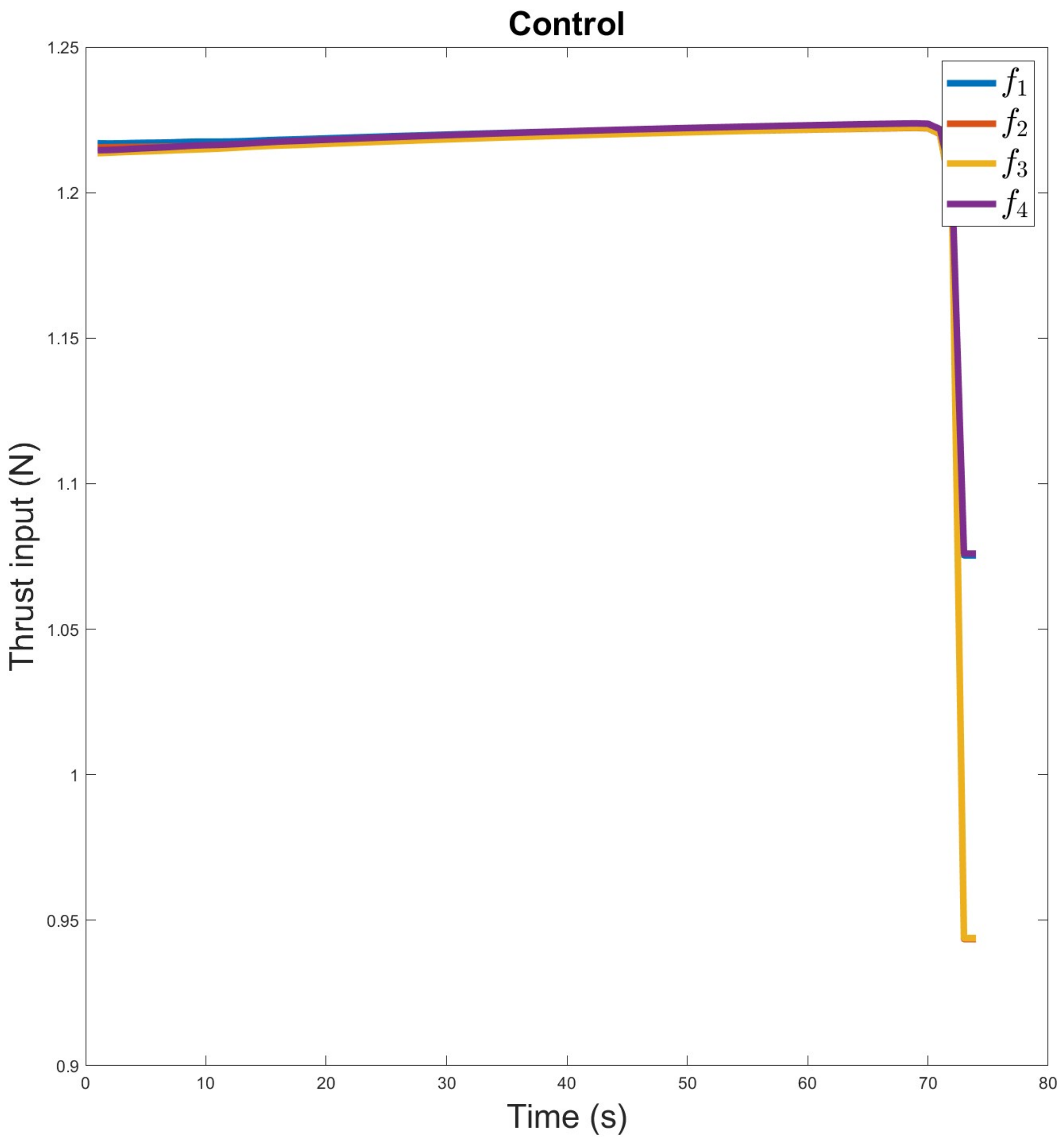

Figure 12: Control used by the quadcopter over time, with MBC and DDP.

3 Implementing Control Barrier Functions (CBF)

Our DDP algorithm will be augmented with a safety-embedded dynamical system. The following changes are added to the nominal DDP algorithm as described below:

1. There will be an additional state added to the initial state vector. We will call this w, called the discrete barrier state. Hence the new state vector will be 13x1 and

$$\mathbf{X} = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r, w]'$$

2. The barrier states are defined in main as follows:

```
h1(x,y,z) = (x-2.2)^2 + (y-2.2)^2 + (z-1)^2 - 1
h2(x,y,z) = (x)^2 + (y+0.2)^2 + (z)^2 - 1
h3(x,y,z) = (x-3)^2 + (y)^2 + (z-0.5)^2 - 1
h1_x = jacobian(h1,X);
h2_x = jacobian(h2,X);
h3_x = jacobian(h3,X);

h10 = double(h1(0,0,0));
h20 = double(h2(0,0,0));
h30 = double(h3(0,0,0));

h1_x_0 = double(h1_x(0,0,0));
h2_x_0 = double(h2_x(0,0,0));
h3_x_0 = double(h3_x(0,0,0));
```

   These zero-conditions for the barrier functions and their jacobians are fed as inputs into the `fnState_And_Control_Transition_Matrices` function.

3. The state matrices are updated with the barrier function as follows:

```
XXX = (1/h10^2)*h1_x_0 +
      (1/h20^2)*h2_x_0 +
      (1/h30^2)*h3_x_0;
Az1 = -(XXX)*A - gamma1*(XXX);
Az2 = -gamma1;

Bz = -XXX*B;

Abar = double([A, zeros(12,1); Az1 Az2]);
Bbar = double([B; Bz]);

A = Abar;
B = Bbar;
```

This effectively embeds the safety conditions into the states.

4. Now we must update the `fnsimulate` function so the system is accurately propagated to avoid the obstacles in its path. Let our barrier function B be $B(h) = 1/h$, so it is simply the inverse of whatever the barrier state we are considering. We will also use the following rule for including multiple barrier states:

$$\sum_i^3 B(h) = \sum_i^3 1/h_i = \frac{1}{h_1} + \frac{1}{h_2} + \frac{1}{h_3}$$

Let the dynamics be $x_{k+1} = f(x_k, u_k)$. Let $w_k = \beta(x_k) - \beta^d$, where $\beta^d = \beta(x^d)$ is for the desired state to be tracked (in our code, $x^d =$ origin of our system).

$$\beta^d = \frac{1}{h_{1,d}} + \frac{1}{h_{2,d}} + \frac{1}{h_{3,d}}$$

The barrier state functions h_1, h_2, h_3 will be recalculated at every simulation step k to be h_k, and at every next time step $k+1$ to be h_{k+1}.

The barrier state will be calculated as follows:

$$w_k = \beta_k - \beta^3$$

For the dynamics function Fx, the barrier state derivative will be calculated as follows:

$$w_{k+1} = B(h(f(x_k, u_k))\beta^d = \beta_{k+1} - \beta^d - \gamma * (w_k + \beta_d - (\beta_k))$$

This was appended to the derivative function and the system was propagated as follows:

$$x_{k+1} = x_k + [Fx; w_{k+1}]dt$$

The quad-copter takes the following 3D path, as a result of these states and input.

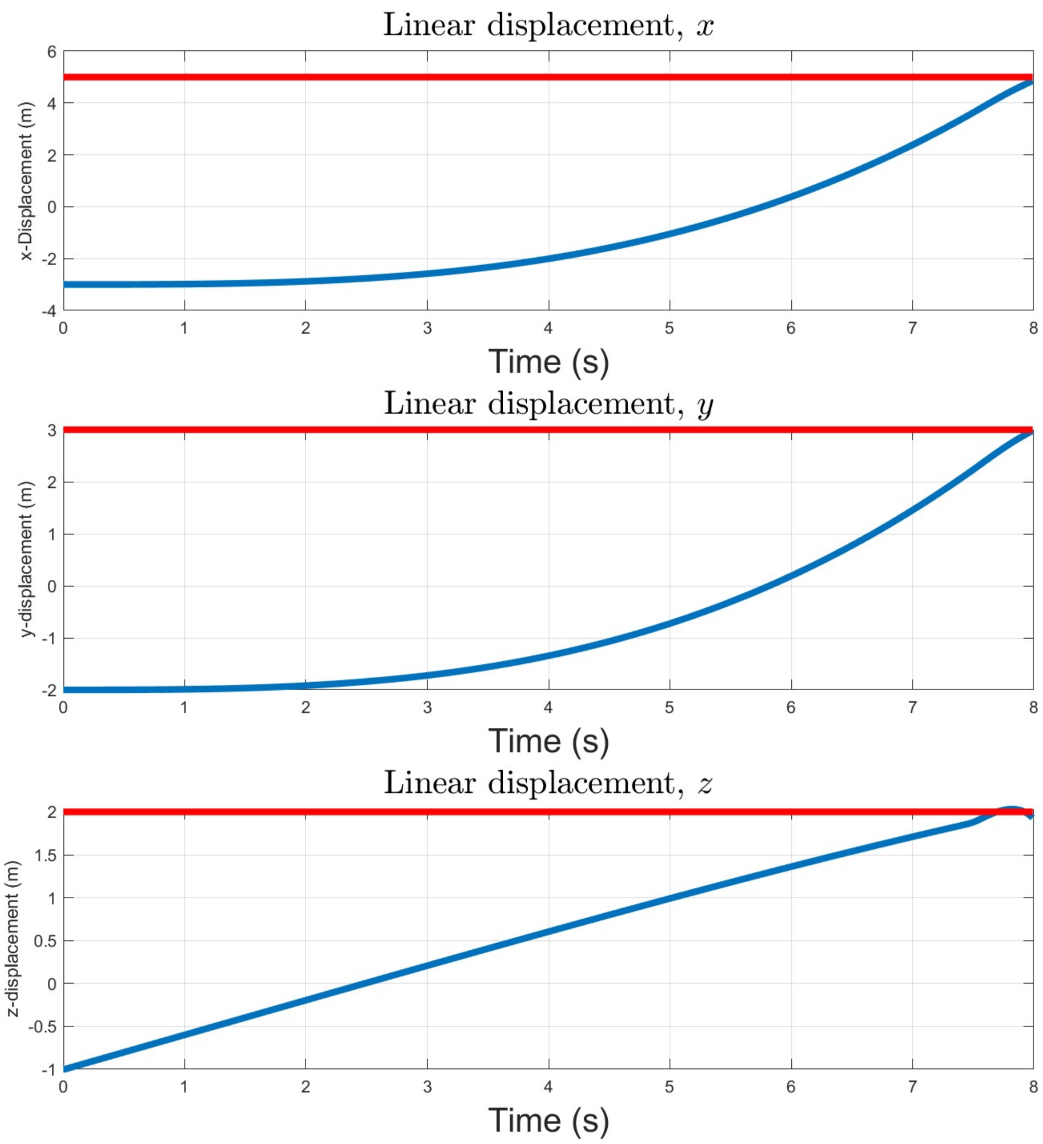

Figure 13: Position of the quad-copter over time using discrete barrier states.

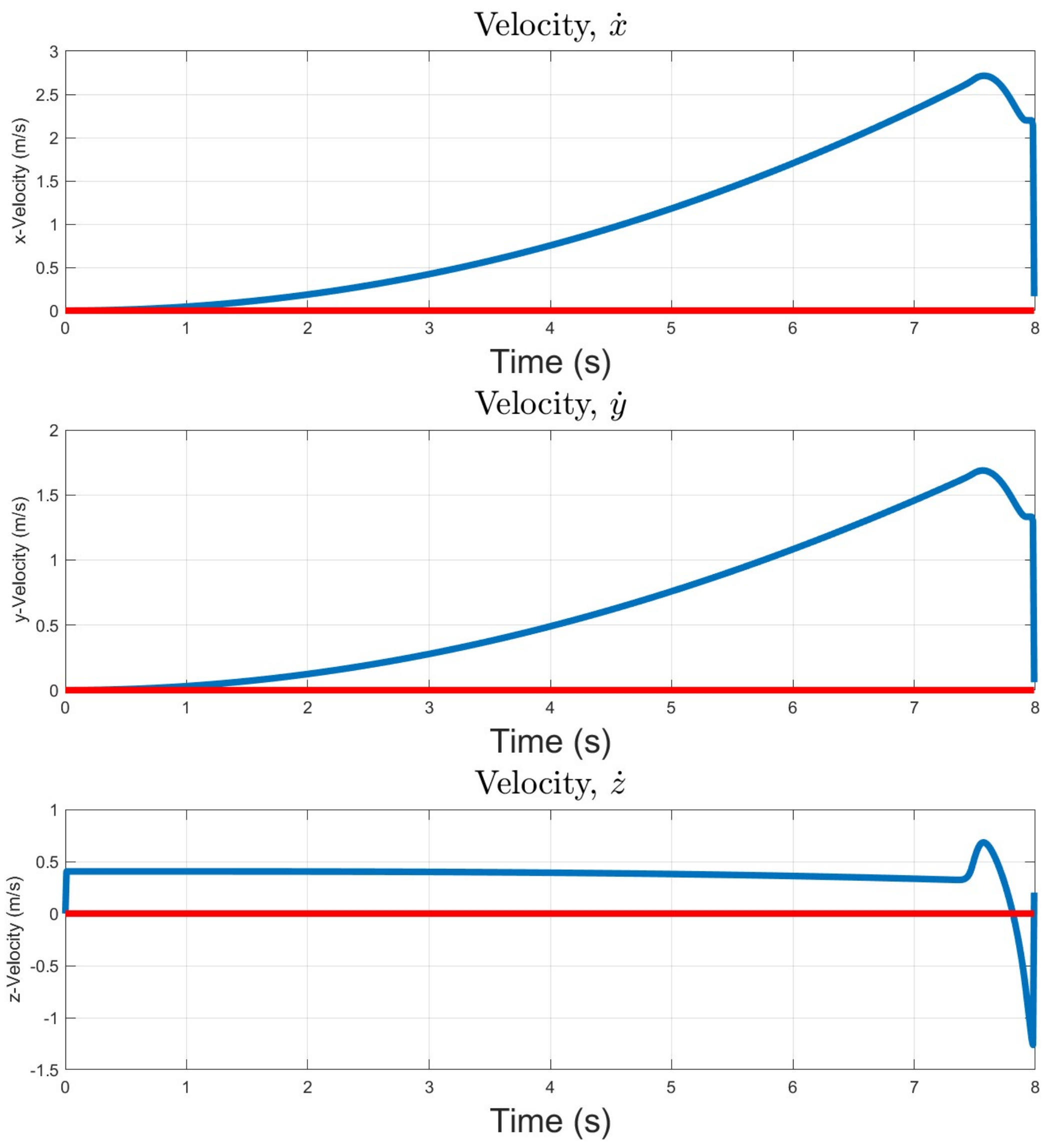

Figure 14: Velocity of the quad-copter over time using discrete barrier states.

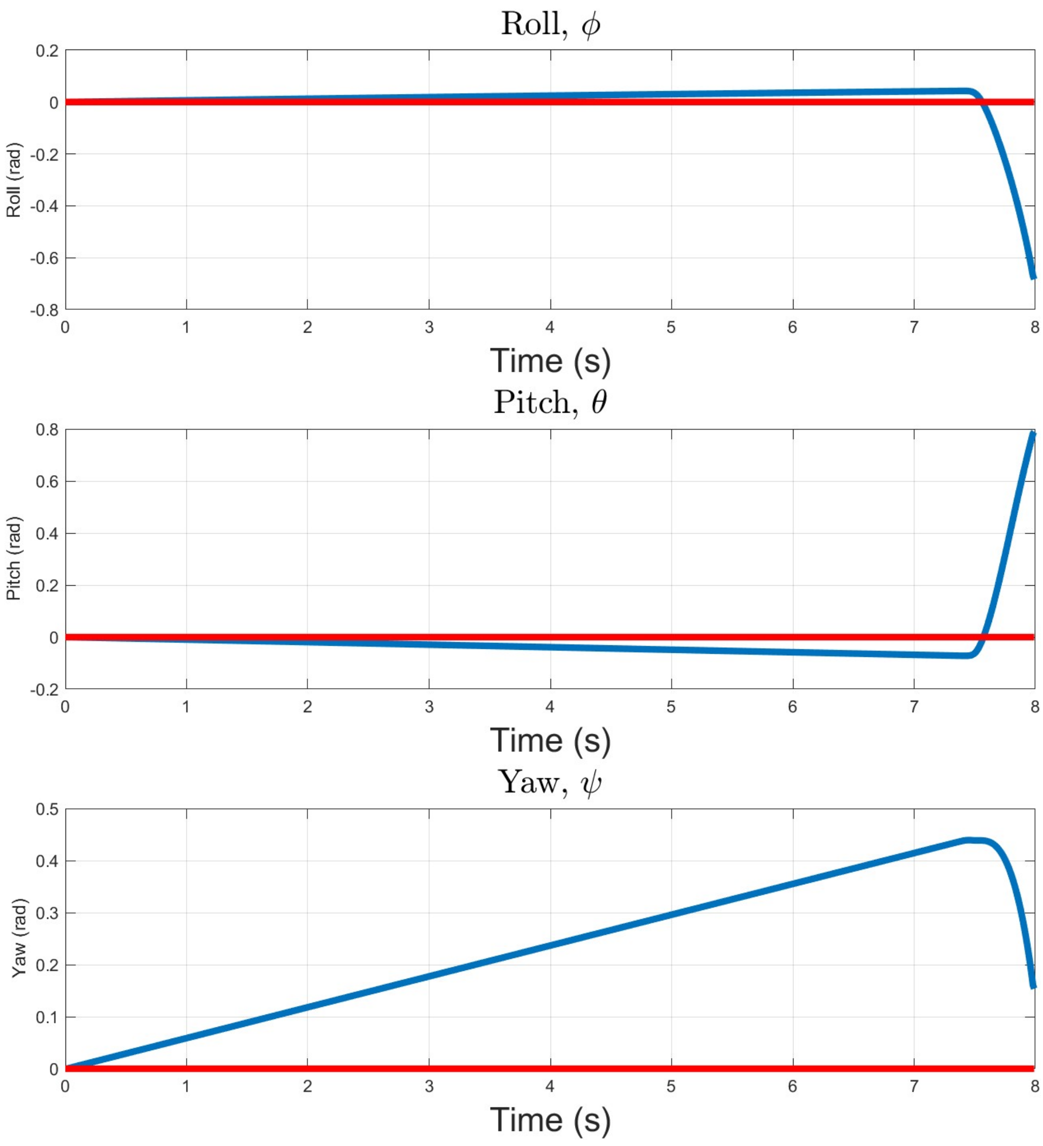

Figure 15: Euler angles of the quad-copter over time using discrete barrier states.

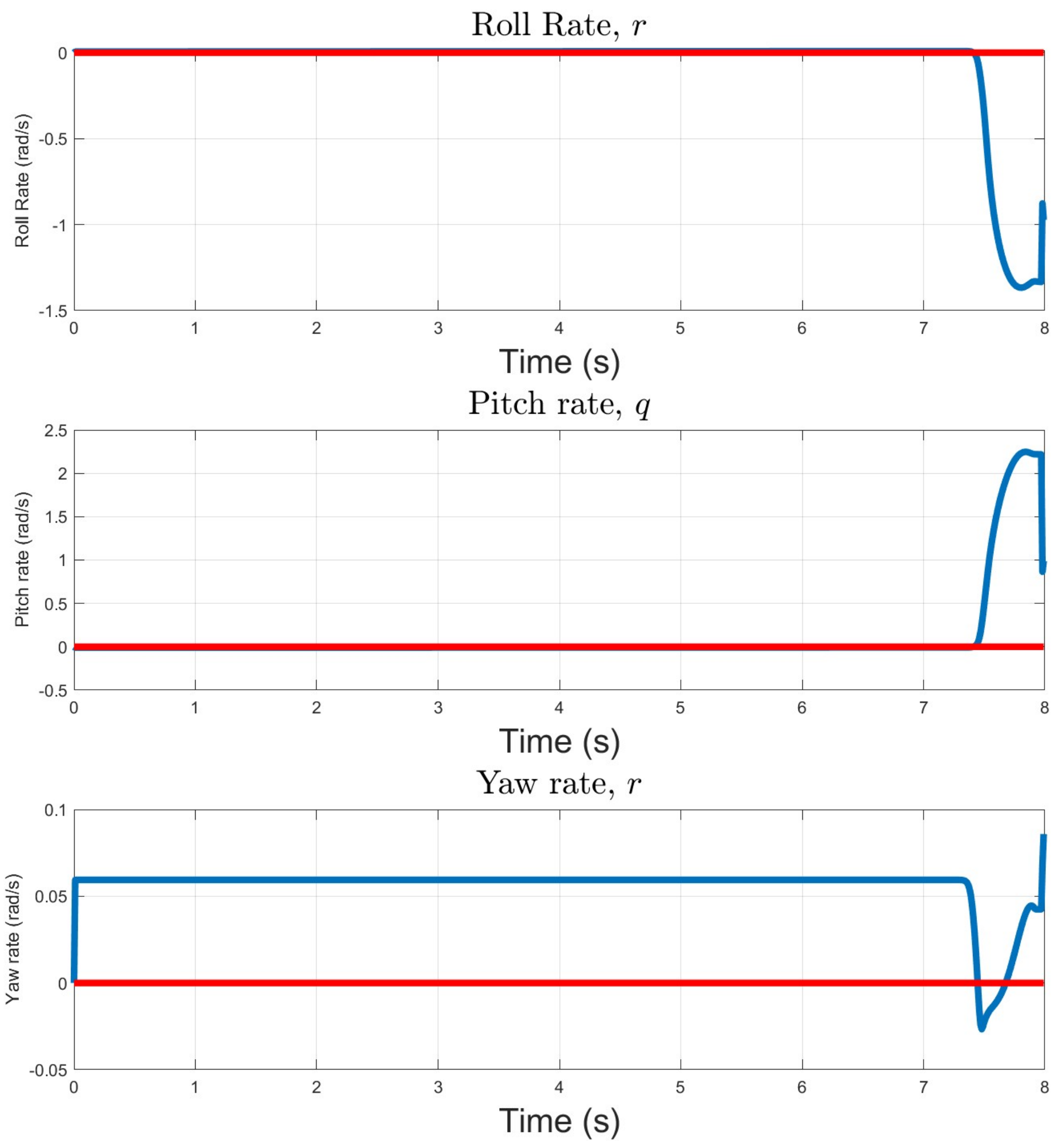

Figure 16: Body rates of the quad-copter over time using discrete barrier states.

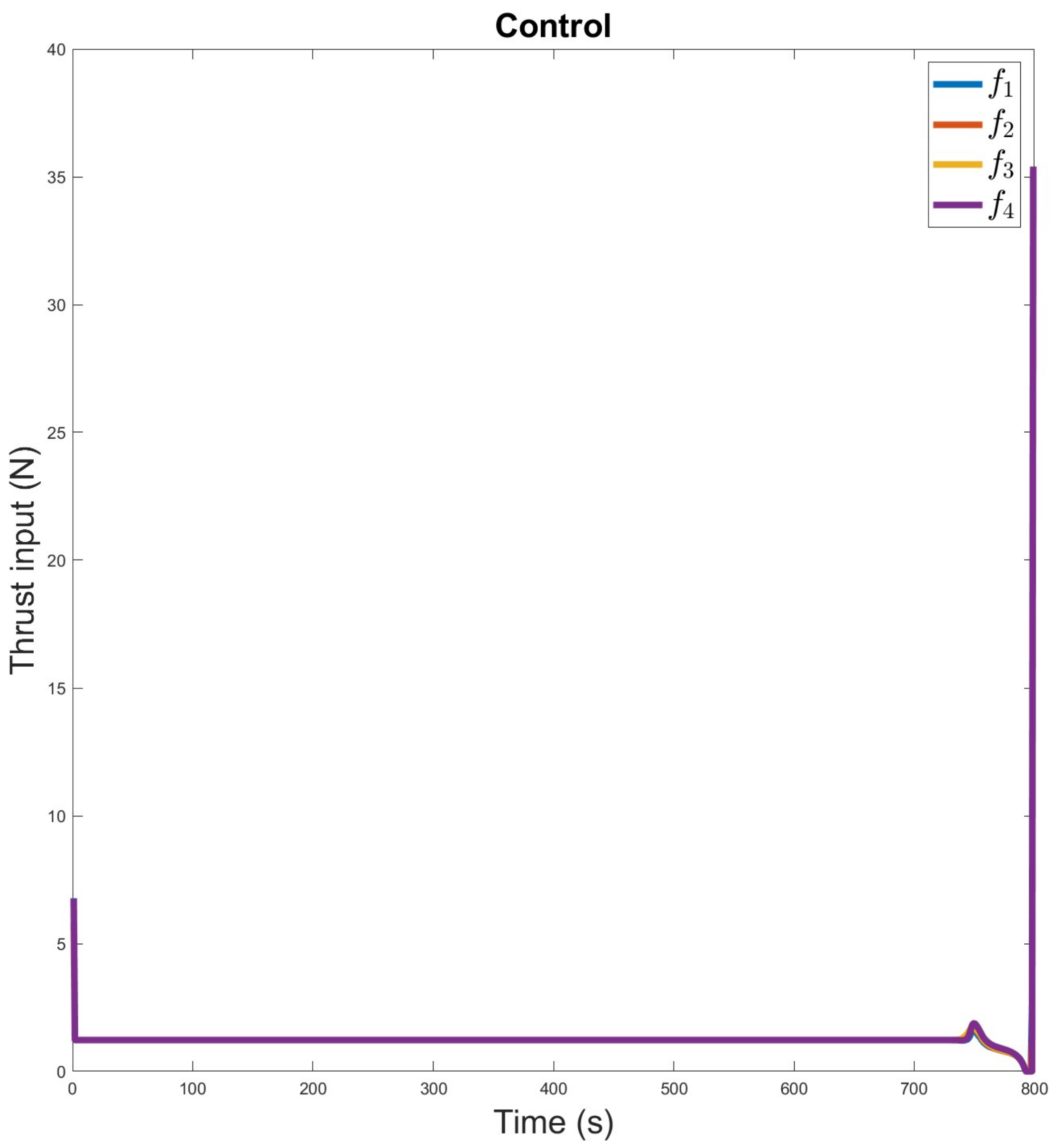

Figure 17: Control force used by the quad-copter over time using discrete barrier states.

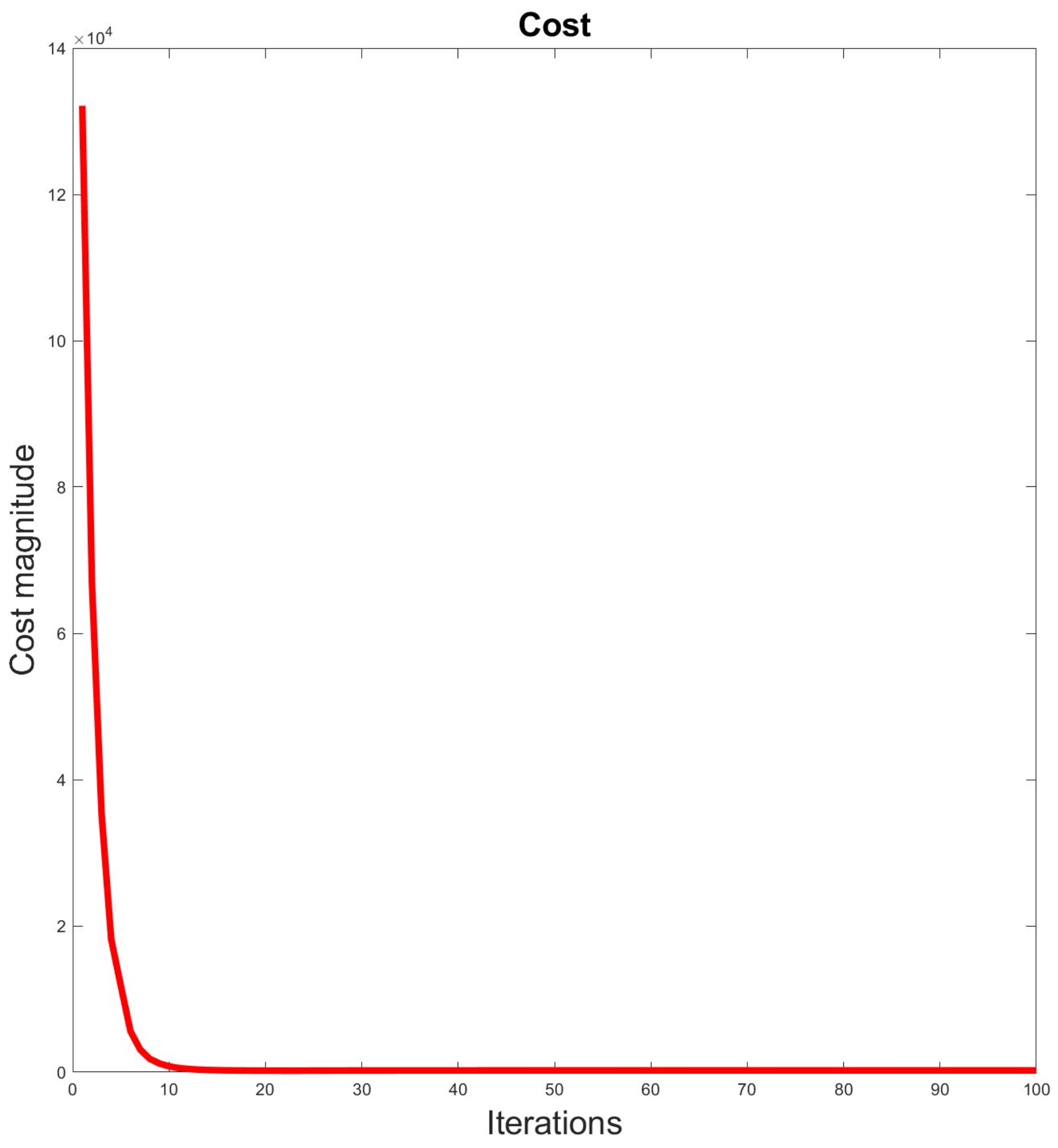

Figure 18: Operating cost of the quad-copter over time using discrete barrier states.

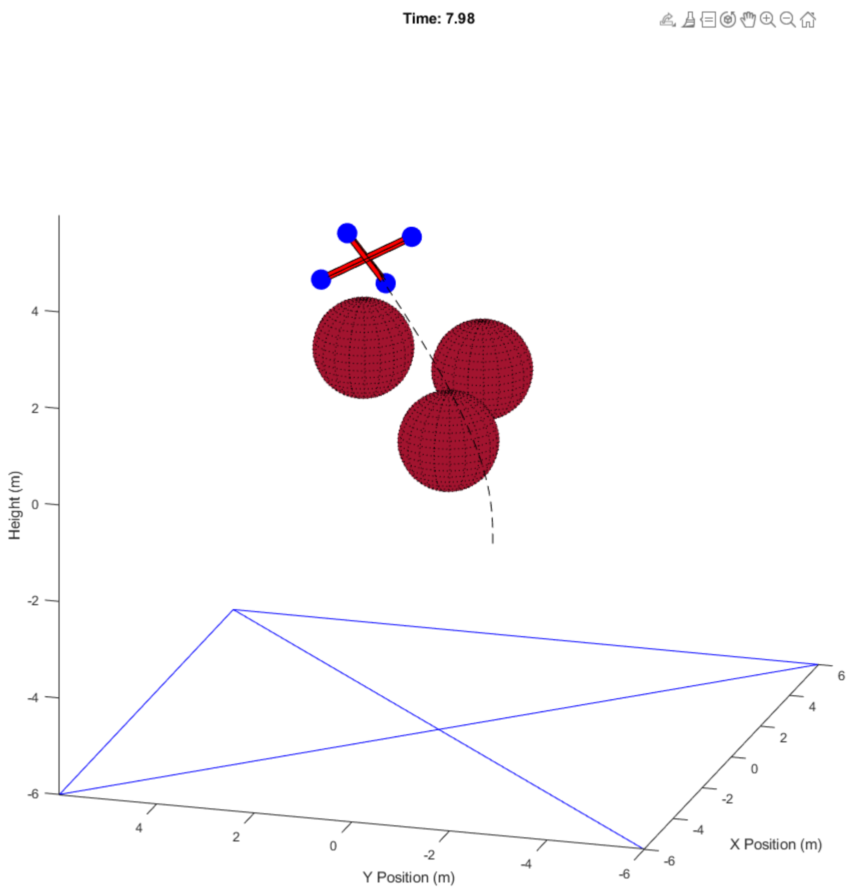

Figure 19: Caption

It can be seen that the safety-embedded trajectory and the nominal DDP trajectory of the rotor are very similar. The safe trajectory definitely does not cross the sphere barrier, but gets very close. Perhaps tuning the γ parameter might help it avoid the obstacles more conservatively.

In future, different sized constraints (rectangular, perhaps moving) might be considered, and the algorithm developed here can be tested.

Ackowledgements

Firstly we thank Manan Satish Gandhi and Hassan Abdullah Almubarak for their continuous help and support during our course. We would also like to thank the Robotarium for their quick responses to our requests, even though we did not get to use their facilities.

References

1 Safety Embedded Control of Nonlinear Systems via Barrier States - Hassan Almubarak, Nader Sadegh, Evangelos A. Theodorou https://doi.org/10.48550/arXiv.2102.10253

2 Safety Embedded Differential Dynamic Programming Using Discrete Barrier States - Hassan Almubarak,Kyle Stachowicz, Nader Sadegh Evangelos A. Theodorou