Robot Control Final Exam

Submitted by: Purna Patel

WPI ID: 150062312

Robust Control:

Step c: Designing Robust Inverse Dynamics control law

- Initially A and B are defined as follows for designing v:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Control gains k are derived using place function and using eigenvalues $\{-3, -3, -4, -4\}$.
- Acl = A – B*k (A closed loop) is obtained.
- An appropriate positive definite diagonal Q matrix is selected.
- P is obtained using lyap function in matlab.
- A 2X2 diagonal rho matrix is defined.
- Boundary layer phi is selected.
- All these values are used as initial input for ode_robust_RRbot function.

Step d: Updating Ode function

- In the ode function ode_robust_RRbot, all the parameters derived in the above step are initialized.
- Desired states are derived using the equations of the trajectory obtained in the first step.
- Error matrix is defined.
- Then the robust control term vr is obtained using following commands (no boundary layer).

```
if norm(e'*P*B)> 0
    vr = -((e'*P*B)/norm(e'*P*B))*p;
else
    vr = 0;
end
```

- v is obtained as follows.

```
v = vd -k*(e)+vr';
```

- Now the Control inputs are derived as per the dynamics as in feedback linearized controller.

```
T = Mmat*v+Cmat*[dth1; dth2]+Gmat;
```

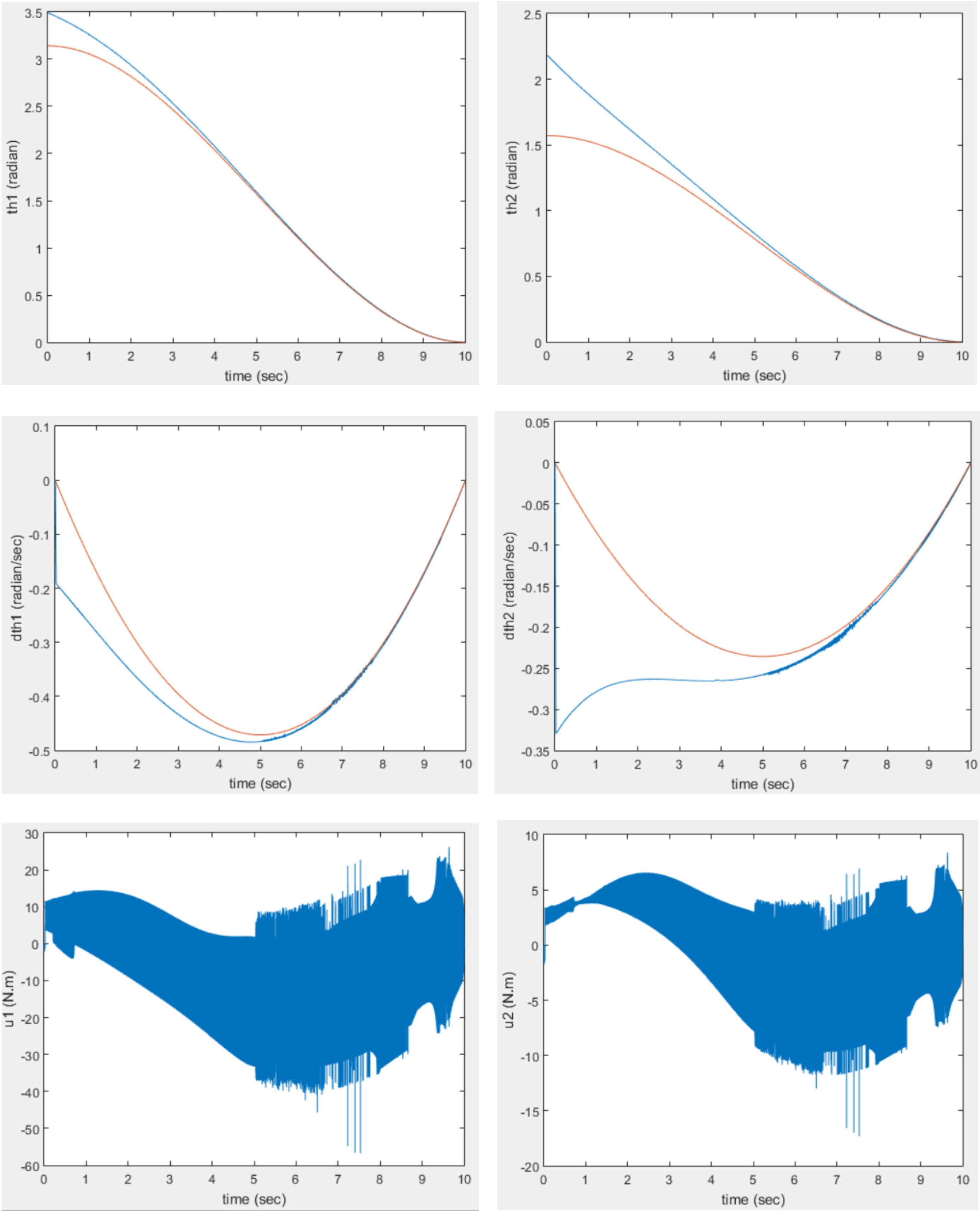
- The findings are simulated using the original values of the dynamics.

Step e: Plotting the output

- List of Parameters after tuning:
  - `lamda = [-3, -3, -4, -4];`
  - `k = place(A,B,lamda);`
  - `Acl = A-B*k;`
  - `Q = eye(4).*5;`
  - `P = lyap(Acl', Q);`

- `p = [10 0; 0 10]; %rho`
- `phi = 0;`

- Following outputs were obtained from the simulation.



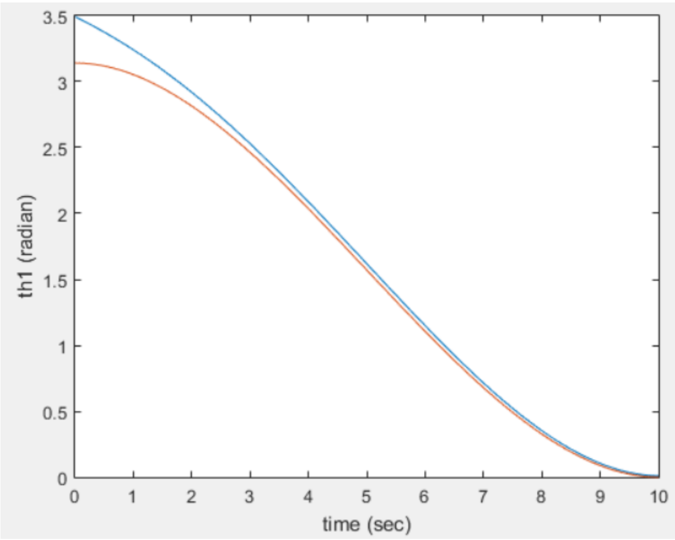
- As seen in the above plots, the system successfully converges to the trajectory.
- But huge amount of chattering was observed in the control efforts and as the control efforts frequently crosses the limits specified in the assignment, the controller frequently fails.
- To overcome this issue, a boundary layer phi is defined in the next step.

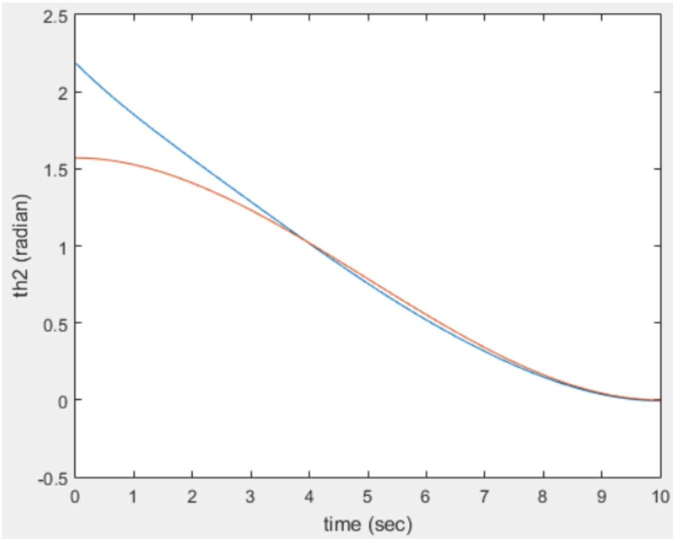
Step f: Defining Boundary layer

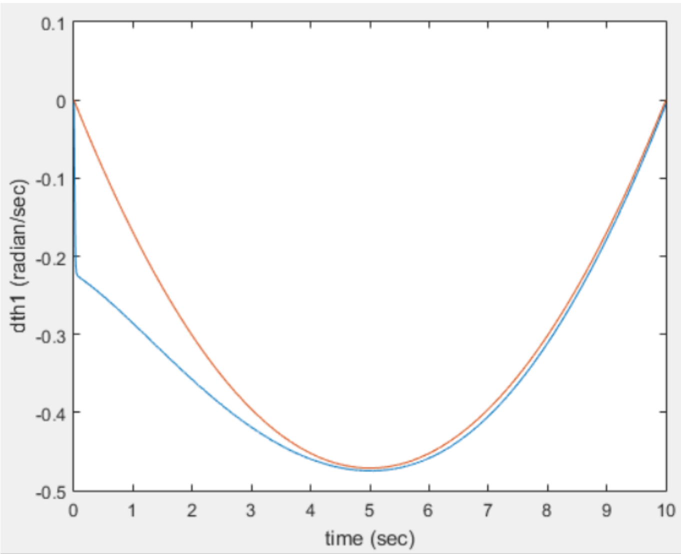
- Boundary layer phi is defined as follows.

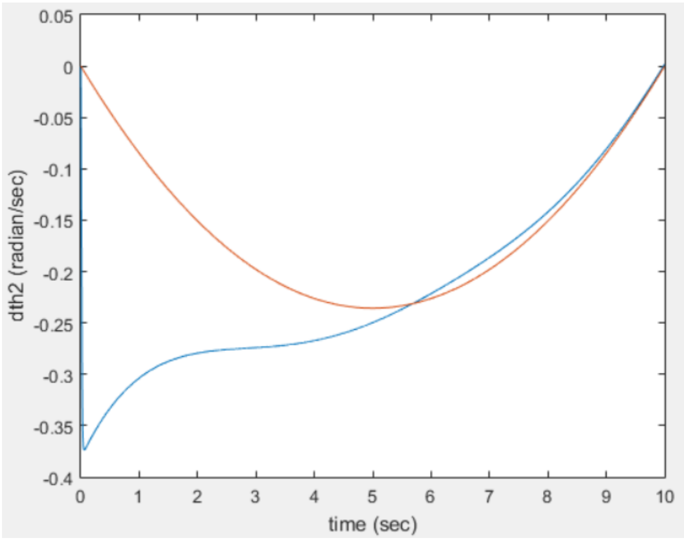
```
if norm(e'*P*B)> phi
    vr = -((e'*P*B)/norm(e'*P*B))*p;
else
    vr = -(e'*P*B)*p/phi;
end
```

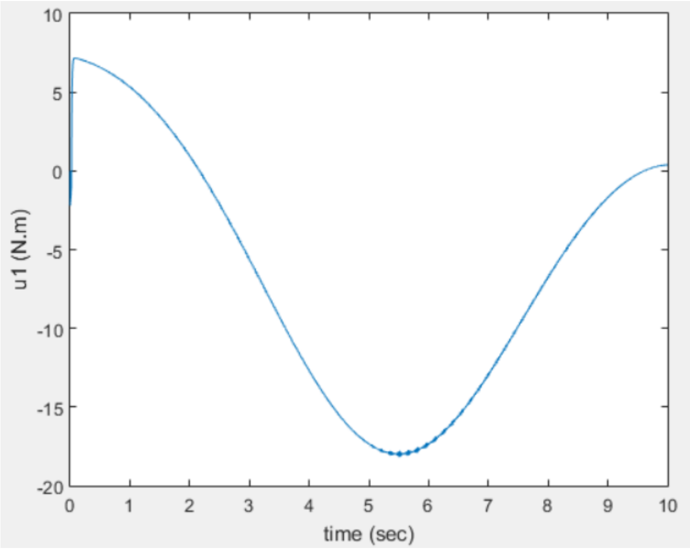
- All the parameters were kept unchanged. And phi = 0.025 was implemented.
- Following results were obtained by implementing this boundary layer.

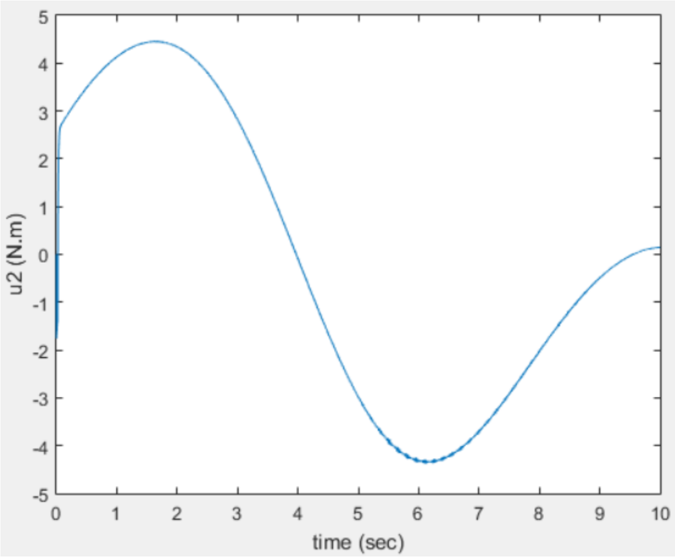








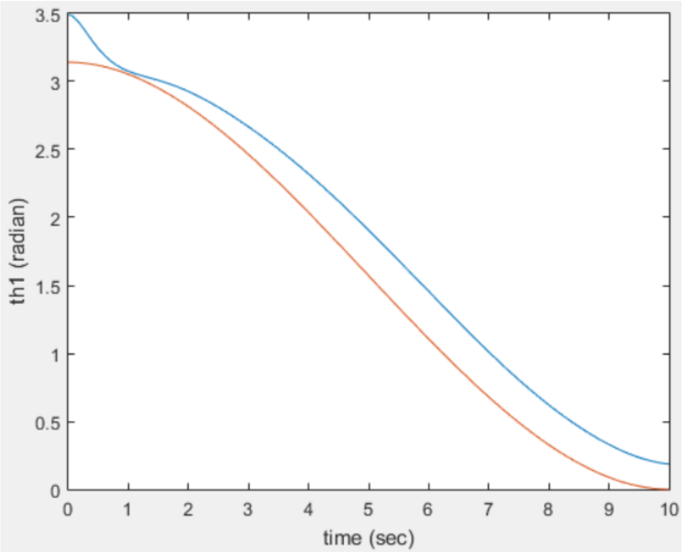


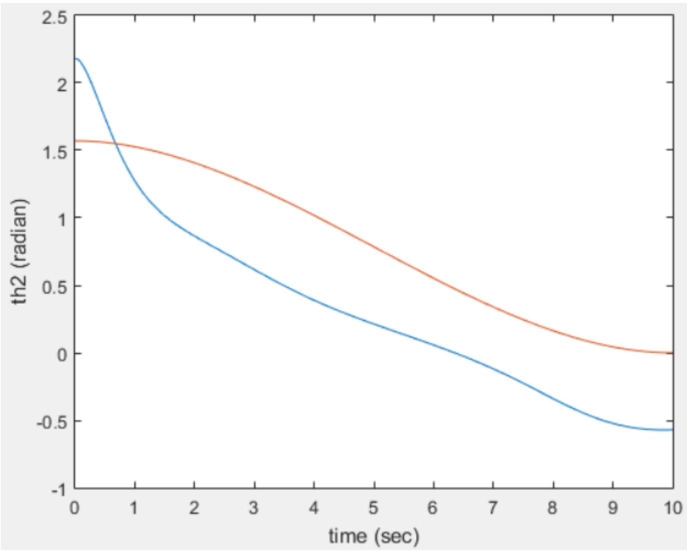


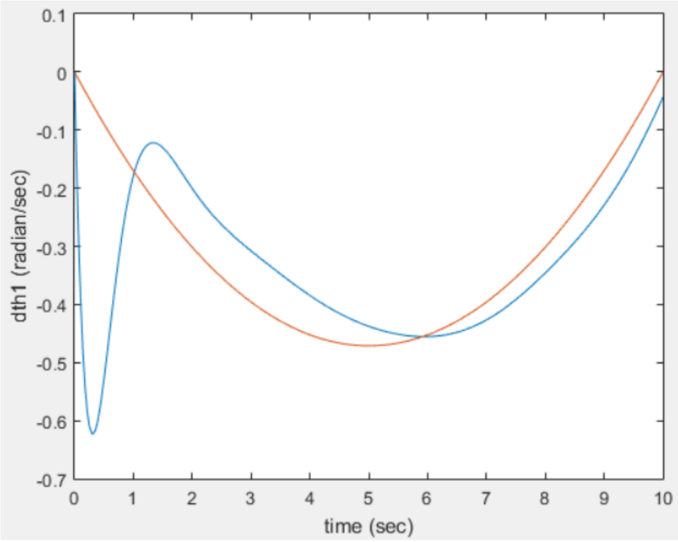
- ➢ As seen in the above plots, the system still converges to the trajectory.
- ➢ Chattering is greatly reduced.
- ➢ The control efforts also falls under the defined range.
- ➢ There is a slight error which remains through out the plot which is due to the implementation of the boundary layer.
- ➢ The overall output of the controller with boundary layer is more practical then that without boundary layer.

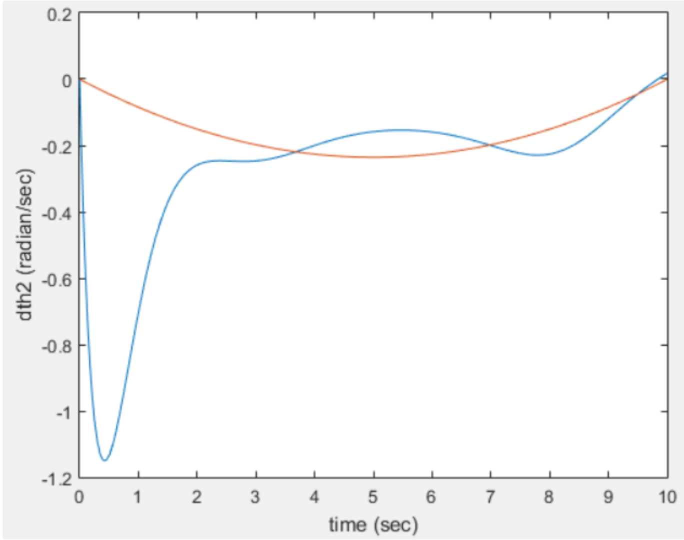
Step g: Comparing with non-robust controller

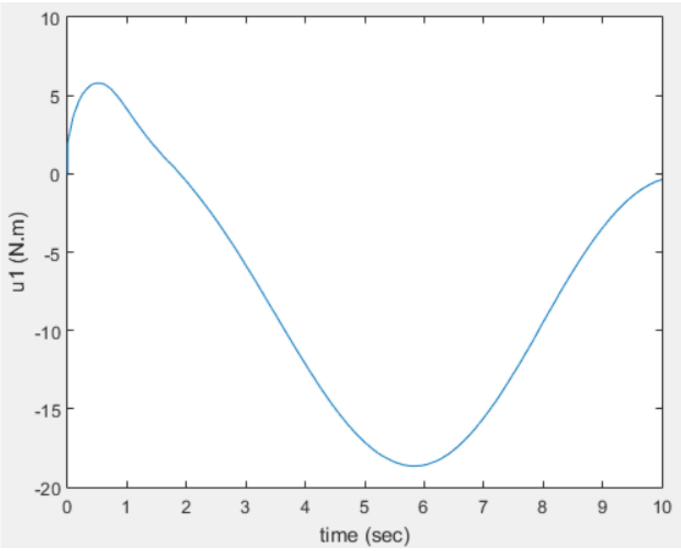
- ➢ Following plots were obtained after setting vr to zero.

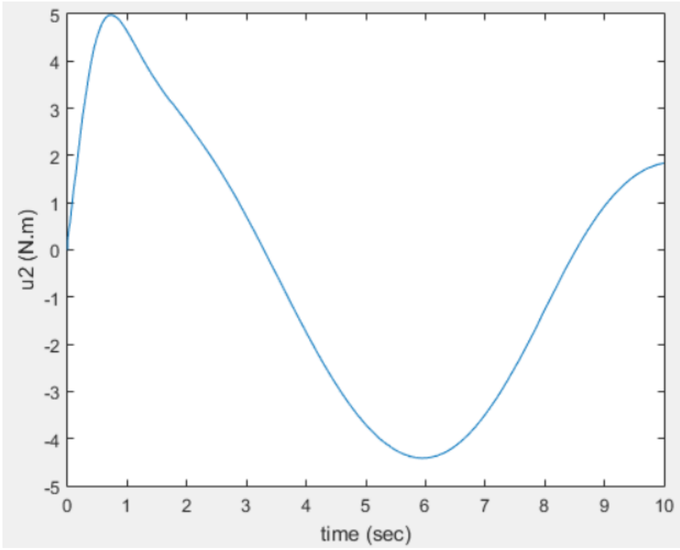








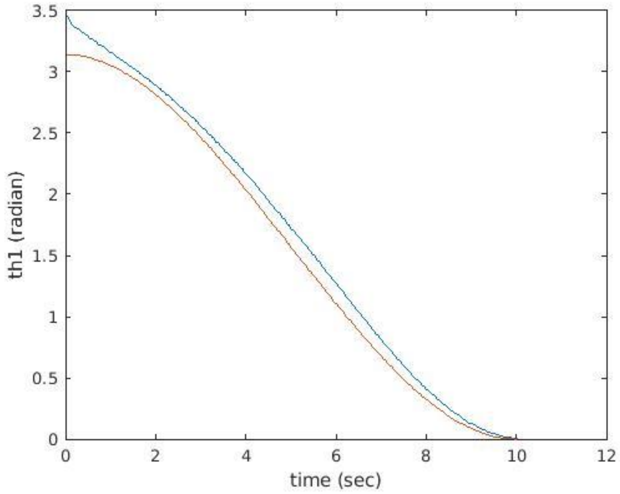


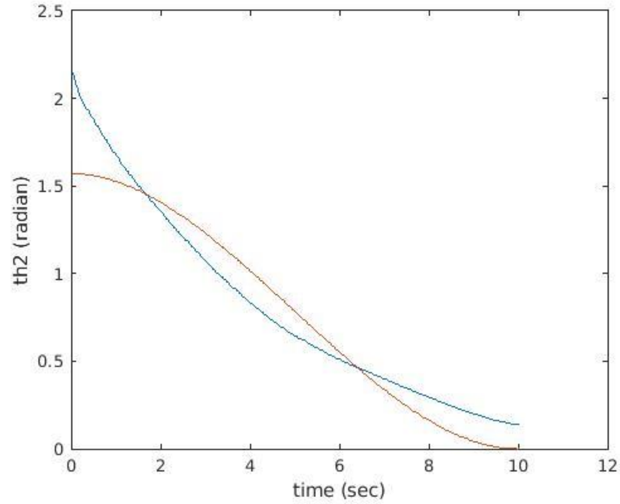


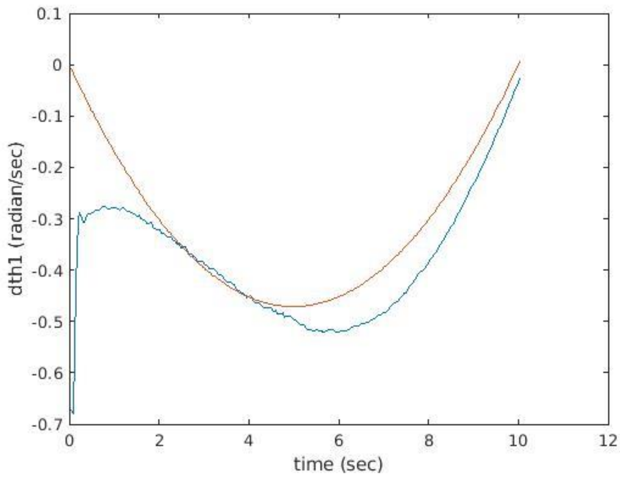
- As seen in the above plots, when vr is set to zero, the system never converges to the trajectory.
- There is always a steady state error between the system and the desired trajectory.
- It is clearly seen that the robust controller is able to control the system more accurately despite the error in the nominal dynamics.

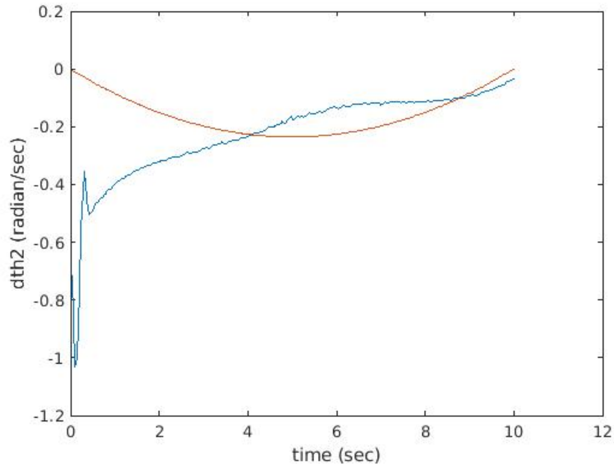
Step h: Gazebo Implementation

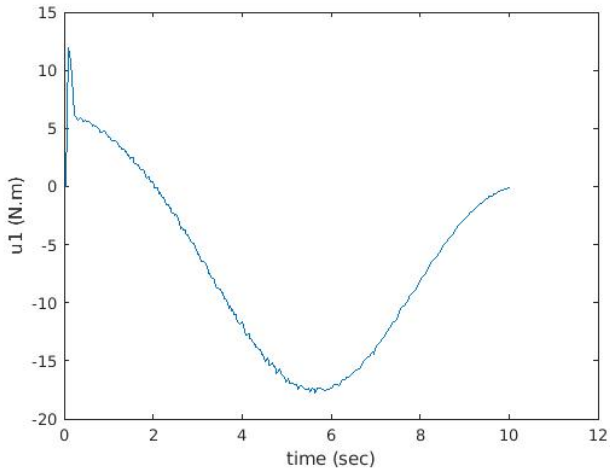
- The same controller discussed above was implemented for the gazebo simulation code rrbot_robust_control.
- Initially some chattering were observed but it was removed by tuning the parameters of the robust controller.
- List of parameters after tuning in Gazebo:
  - `lamda = [-3, -3, -4, -4];`
  - `k = place(A,B,lamda);`
  - `Acl = A-B*k;`
  - `Q = eye(4);`
  - `P = lyap(Acl', Q);`
  - `p = [10 0; 0 10]; %rho`
  - `phi = 0.05;`
- Following results were obtained.

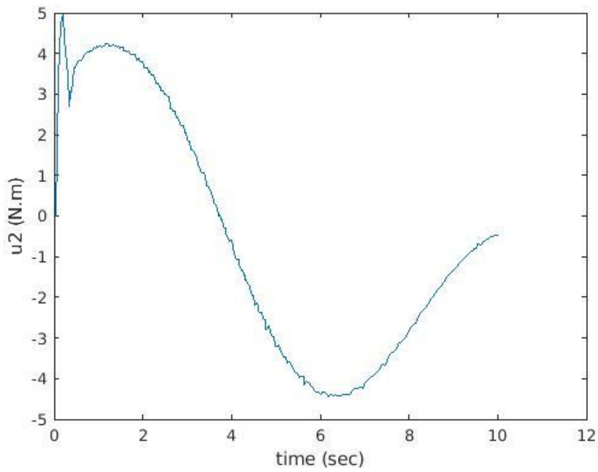












- As seen in the plots, the system almost converges to the trajectory.
- To eliminate the slight observed errors, the parameters were increased but it induced the phenomenon of chattering.
- This was the best optimum output obtained after tuning the parameters.
- The control efforts also remains in the defined range.
- It can be seen that the error in between the system and the trajectory is constantly decreasing.

Adaptive Control:

Step j: Designing Adaptive Control Law

- Initially A and B are defined as follows for designing v:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Control gains k are derived using place function and using eigenvalues $\{-3, -3, -4, -4\}$.
- Acl = A – B*k (A closed loop) is obtained.
- An appropriate positive definite diagonal Q matrix is selected.
- P is obtained using lyap function in matlab.
- Now initial nominal values are defined and the initial value of alpha_hat is obtained.
- Now a 5X5 diagonal matrix gamma is defined.
- All these values are used as initial input for ode_adaptive_RRbot function.

Step k: Updating ode function

- In the ode function ode_adaptive_RRbot, all the parameters derived in the above step are initialized.
- Desired states are derived using the equations of the trajectory obtained in the first step.
- Error matrix is defined.
- Using this error matrix, and desired acceleration, v is calculated.
- A Y_out is defined by replacing all ddth1 and ddth2 with v(1) and v(2) so that it can be directly used to obtain control efforts by multiplying it with alpha_hat.

```
Y_out = [v(1), ...
    cos(th2)*(2*v(1) + v(2)) - 2*sin(th2)*dth1*dth2 - sin(th2)*dth2^2, ...
    v(2), ...
    -sin(th1)*g, ...
    -sin(th1 + th2)*g; ...
    0, ...
    sin(th2)*dth1^2 + cos(th2)*v(1), ...
    v(1) + v(2), ...
    0, ...
    -sin(th1+th2)*g];

T = Y_out*alpha_hat;
```

- This control effort is then substituted in the original dynamics to simulate and also the obtained values are used as a feedback to update the adaption law.
- Now using the obtained values of dth1, dth2, ddth1 and ddth2, Y matrix is obtained.
- M_hat matrix and hence phi matrix are obtained as follows.

```
M1 = [1, 2*cos(th2), 0, 0, 0;
      0, cos(th2), 1, 0, 0];
M2 = [0, cos(th2), 1, 0, 0;
      0, 0, 1, 0, 0];
M_hat = [M1*alpha_hat, M2*alpha_hat];
phi = M_hat\Y;
```

- Here M1 and M2 are obtained from Y matrix.
- From these matrices d_alpha_hat is calculated.

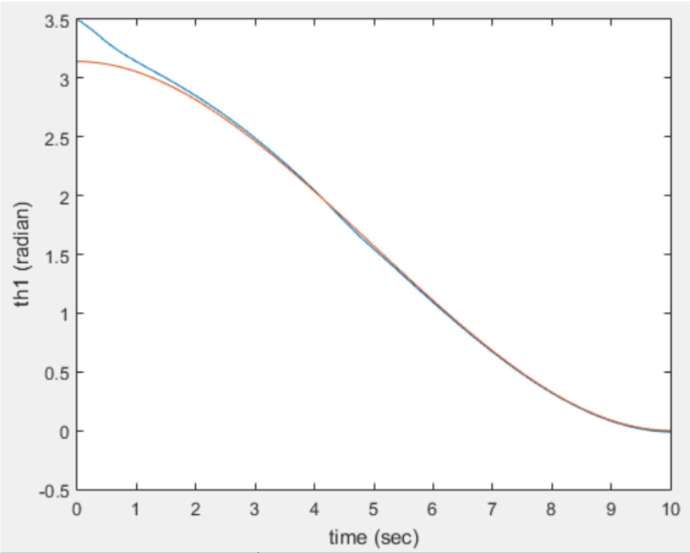
```
d_alpha_hat = -gamma\(phi'*B'*P*e);
```

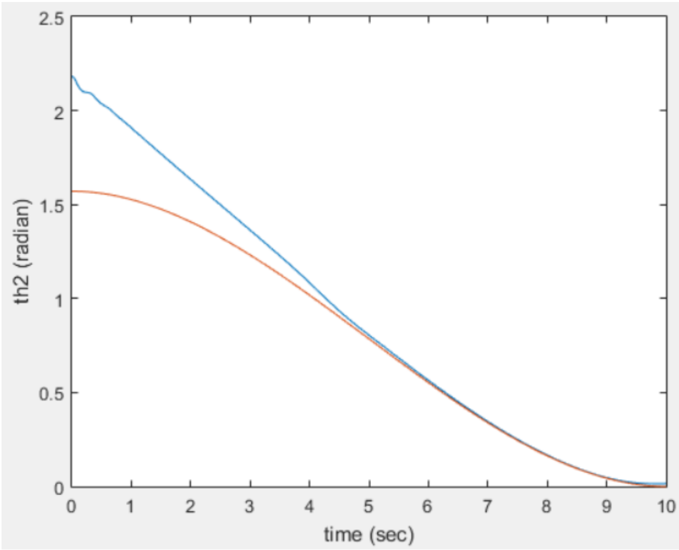
- This is then integrated to update the value of alpha_hat.

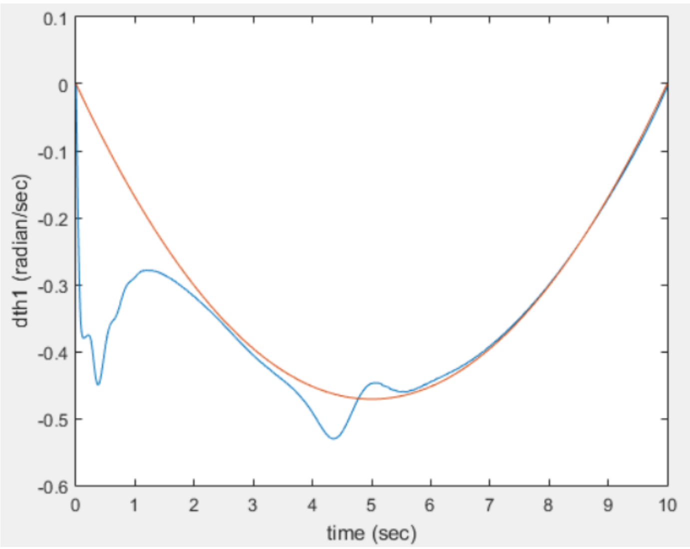
- This updated value is now used as updated dynamics for the next iteration.

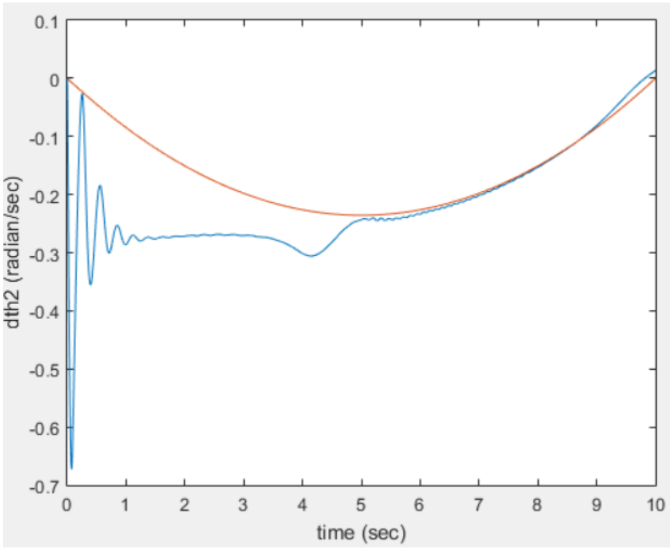
Step l: Plotting the output

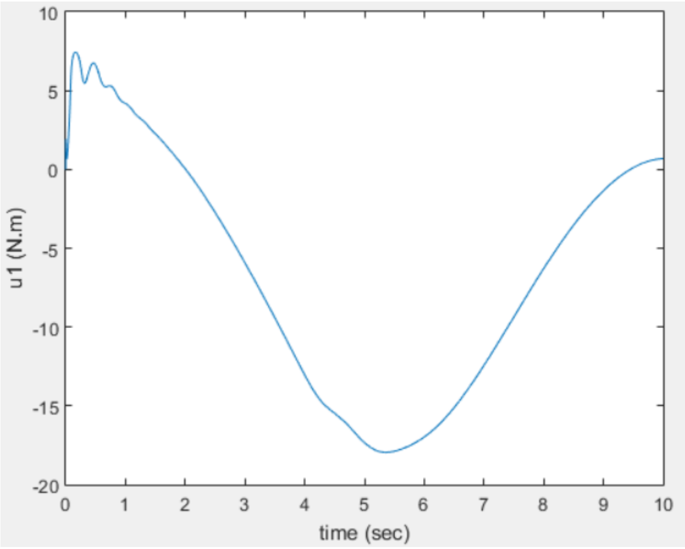
- List of Parameters after tuning:
  - `lamda = [-3, -3, -4, -4];`
  - `k = place(A,B,lamda);`
  - `Acl = A-B*k;`
  - `Q = eye(4);`
  - `P = lyap(Acl', Q);`
  - `gamma = eye(5).*0.1;`
- Following output was obtained after tuning the parameters.

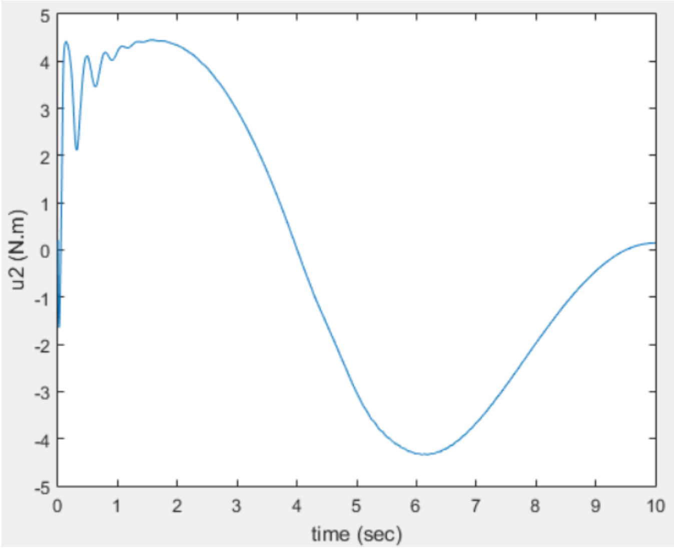




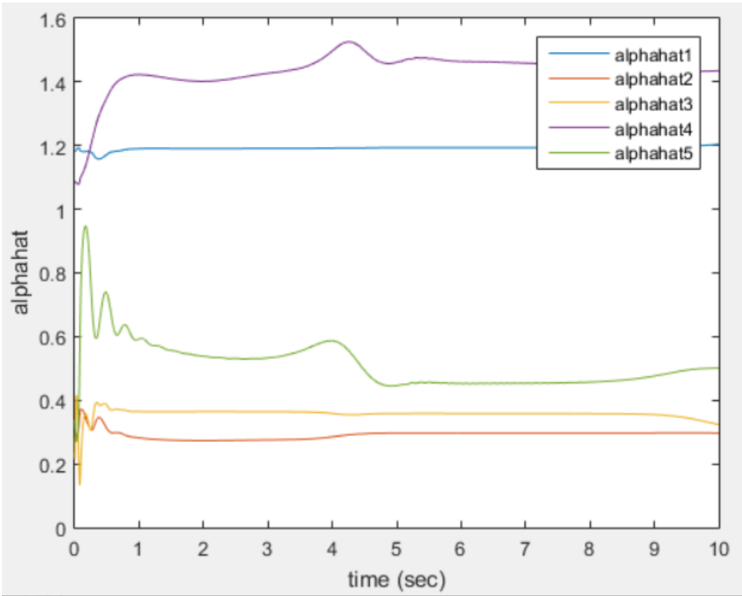








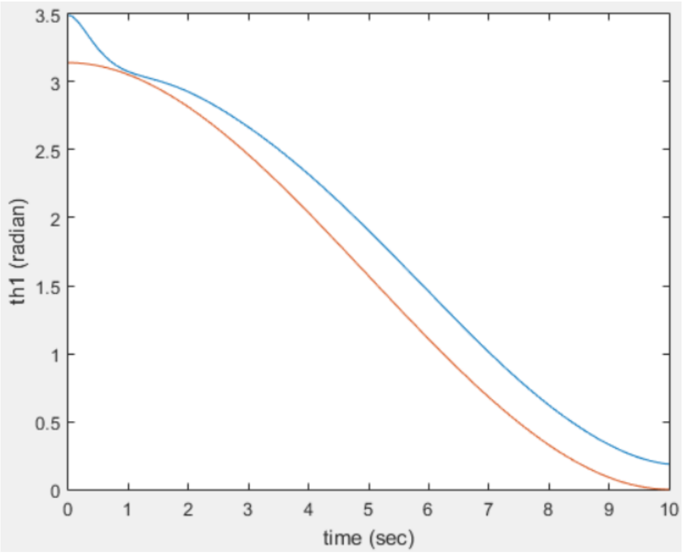
- As seen in the above plots, the system successfully converges to the trajectory.
- And the control efforts also falls under the defined range.
- There were slight oscillations observed initially in dth1 and dth2 and also in u1 and u2.
- But the system eventually stabilizes and converges to the trajectory.
- Following are the plots of alpha_hat vs time.

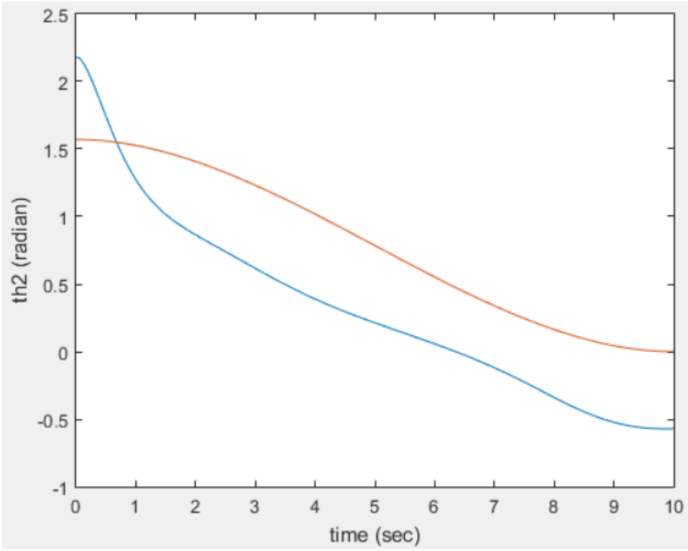


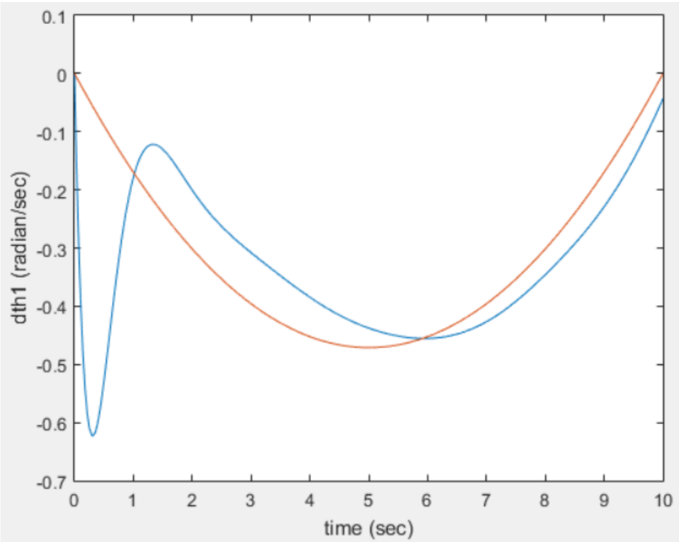
- It can be see from the plot that the parameters converges to a constant value as the time proceeds.
- The original value of alpha are {1.5730, 0.4500, 0.2865, 1.4500, 0.4500}.
- It can be observed that the values of alpha_hat do not converge to the original alpha values but they do converge to constant values.

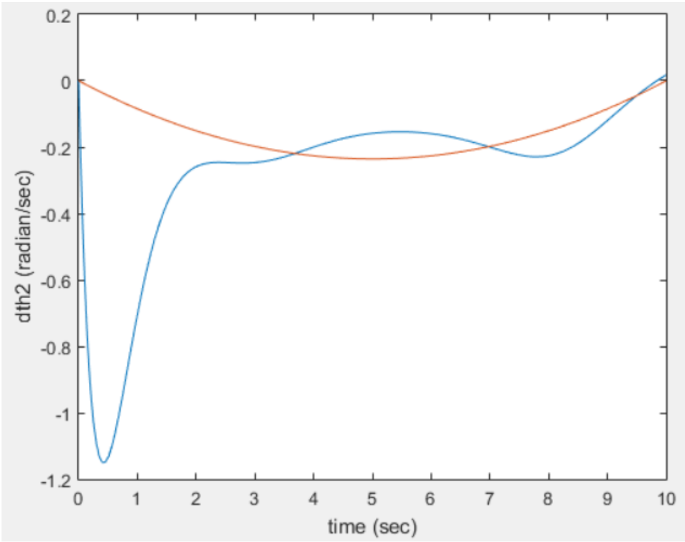
Step m: Comparing with non-adaptive controller

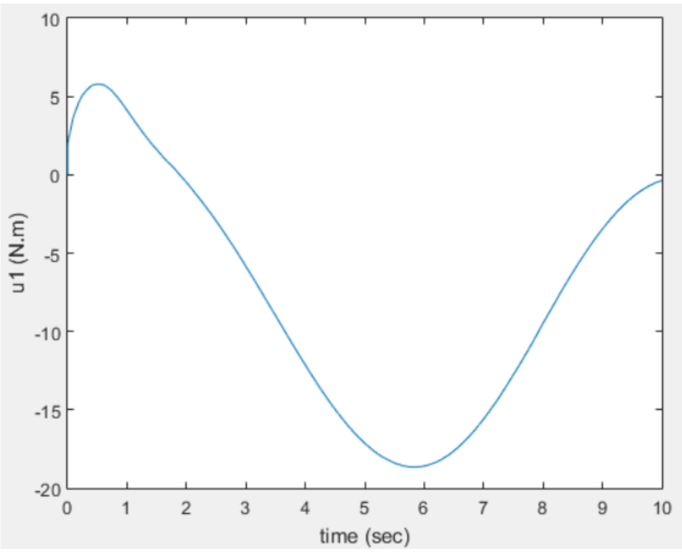
- Following plots were obtained after setting value of P to zero.

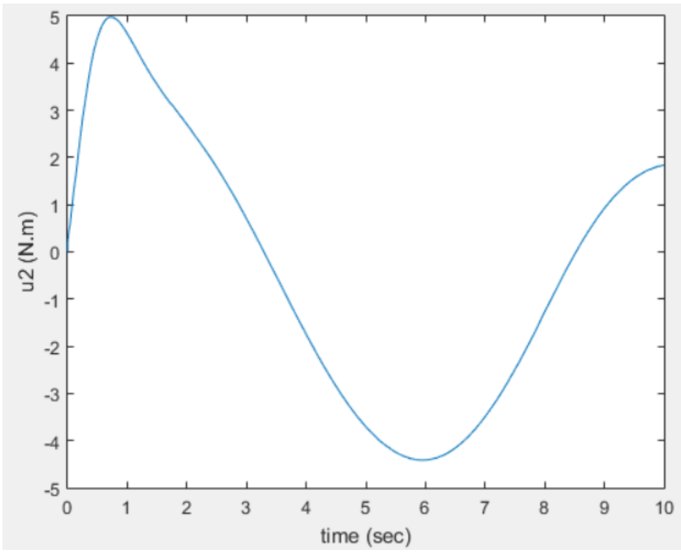












- As seen in the above plots, when P is set to zero, the system never converges to the trajectory.
- There is always a steady state error between the system and the desired trajectory.
- It is clearly seen that the adaptive controller is able to control the system more accurately despite the error in the nominal dynamics.

Step n: Gazebo Implementation

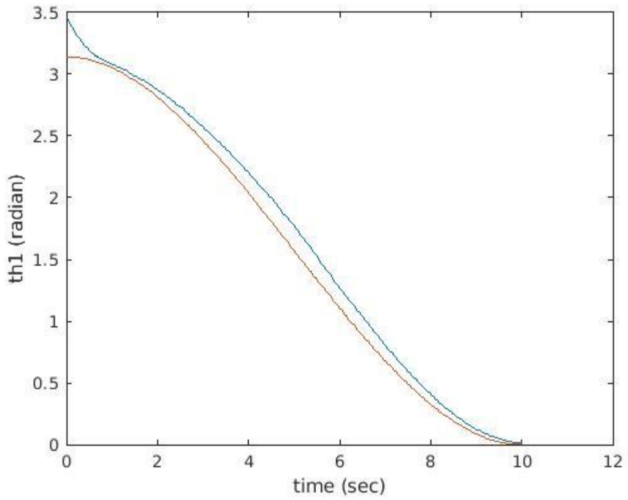
- The same controller discussed above was implemented for the gazebo simulation code rrbot_adaptive_control.
- To integrate the d_alpha_hat and find the alpha_hat, following iterative formula was used.

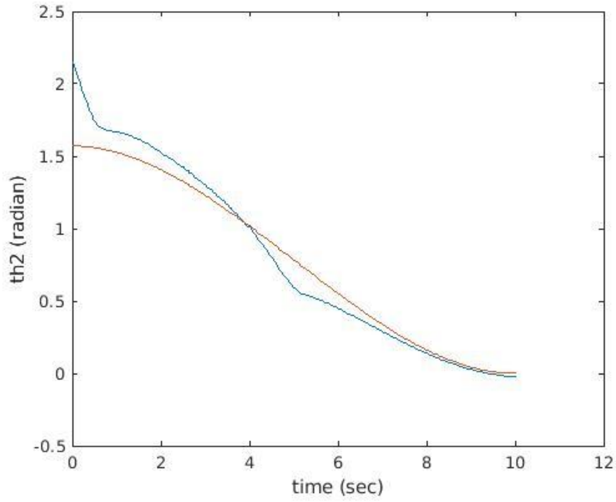
```
alpha_hat = alpha_hat + d_alpha_hat*(t-t_prev);
```

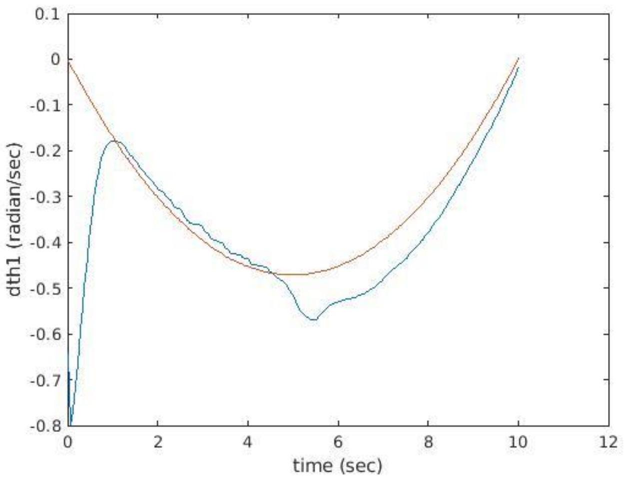
- Other than this ode45 was also used in some simulations to cross verify the results.
- ode45 was implemented as follows.

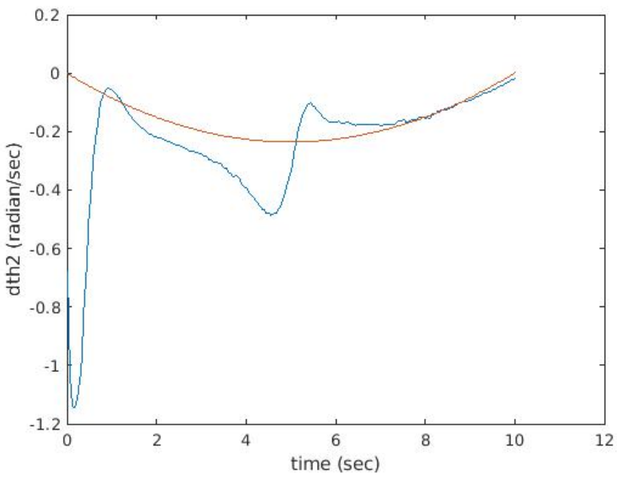
```
[tm,y] = ode45(@(tm,y) d_alpha_hat, [t_prev t], alpha_hat);
b = size(y);
alpha_hat = y(b(1), :)';
```

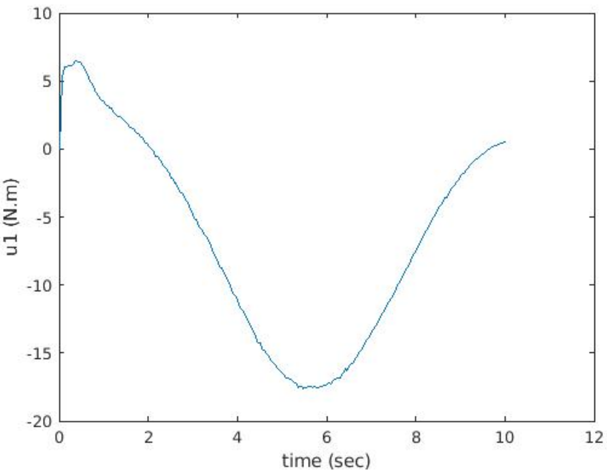
- There was no difference in the results observed by implementing above two formulas seperately.
- But it was observed that using the ode45 was slight computationally heavy as it did around 50 iterations each time and gave a list of values, from which only last value was used.
- List of parameters after tuning in Gazebo:
    - `lamda = [-3, -3, -4, -4];`
    - `k = place(A,B,lamda);`
    - `Acl = A-B*k;`
    - `Q = eye(4);`
    - `P = lyap(Acl', Q);`
    - `gamma = eye(5).*9;`
- After critically tuning the parameters, following result were obtained.

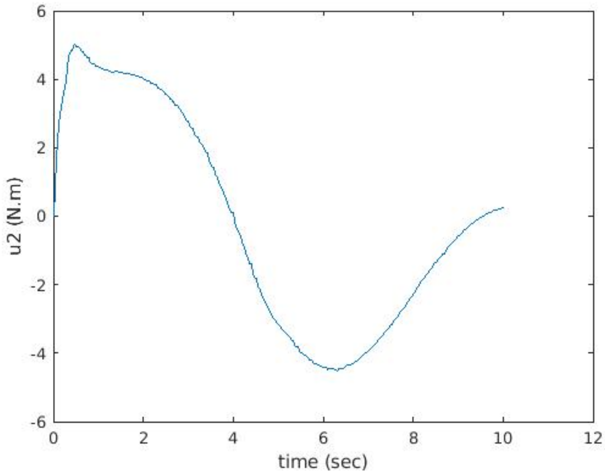












- As seen in the plots, the system successfully converges to the trajectory.
- The control efforts also remains in the defined range.
- It can be seen that the error in between the system and the trajectory is constantly decreasing.
- It also can be observed from the plots that adaptive controller performed overall better than the robust controller in the gazebo environment.
- Adaptive controller was able to converge system to the trajectory more accurately than the robust controller without any chattering.
- But the parameters of Adaptive controller were very critically tuned, even a small change in the parameters can make the system to explode.