Control For Robotics - SS24 - Assigment 2

Student: Nicolas Helio Cunha Nafrink #: 03783400

Problem 2.1

inputs v and w → v: predefined $\bar{v}$ (desired); w: turning rate

$$\dot{y}(t) = v(t)\sin(h(t)) \qquad y \to y_{goal}$$
$$\dot{h}(t) = w(t) \qquad h \to 0$$

$$J(x_0) = \frac{1}{2}(x(T) - x_{goal})^T Q_t (x(T) - x_{goal}) + \int_0^T \frac{1}{2}\left((x(t) - x_{goal})^T Q_s (x(t) - x_{goal}) + u(t)^T R_s u(t)\right) dt$$

2.1-a) Equilibrium:

$x = x_{eq}$ $\quad u = u_{eq}$

$\dot{x} = 0$ $\quad \dot{u} = 0$

$\ddot{x} = 0$ $\quad \ddot{u} = 0$

Equilibrium Equations

$$\dot{y}(t) = v(t)\sin(h(t)) = 0$$
$$\dot{h}(t) = w(t) = 0 \rightarrow \underline{w(t) = 0}$$

$\bar{v}$ (desired) $= v(t)$

$$v(t)\sin(h(t)) = 0$$

$$x = \begin{bmatrix} y(t) \\ h(t) \end{bmatrix} \qquad u = w(t)$$

$$\sin(h(t)) = 0$$

since $x_{goal} = y_{goal}$ $\quad \underline{h(t) = 0}$

$$x_{eq} = \begin{bmatrix} y_{goal} \\ 0 \end{bmatrix} \qquad u_{eq} = 0$$

$y_{eq} = y_{goal}$ $\quad w_{eq} = 0$

$h_{eq} = 0$

Linearizing the system:

$$f_1(y, h, w) = \bar{v}\sin(h)$$
$$f_2(y, h, w) = w$$

$$L_1(y,h,w) = f_1(y_{eq}, h_{eq}, w_{eq}) + \frac{df}{dy}\Big|_{eq}\cdot(y - y_{eq}) + \frac{df}{dh}\Big|_{eq}(h - h_{eq}) + \frac{df}{dw}\Big|_{eq}(w - w_{eq})$$

($\frac{df}{dy}$: 0; $\frac{df}{dh}$: $\bar{v}\cos(h)$; $\frac{df}{dw}$: 0)

$$L_1(y,h,w) = \cancel{f(y_{eq}, h_{eq}, w_{eq})}^{0} + \cancel{f_{y_{eq}}}^{0}\cdot(y - y_{eq}) + f_{h_{eq}}(h - h_{eq}) + \cancel{f_{w_{eq}}}^{0}(w - w_{eq})$$

$$L_1(y,h,w) = \bar{v}\cdot\cos(h_{eq})\cdot h = \bar{v}\cdot(h - h_{eq})$$

$$L_2(y,h,w) = (w - w_{eq})$$

Linear system

$$\delta\dot{x} = \begin{bmatrix} 0 & \bar{v} \\ 0 & 0 \end{bmatrix} \delta x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \delta u \quad , \quad \delta x = \begin{bmatrix} \gamma \\ h \end{bmatrix}$$

b)

$$\delta u(x) = -K \delta x$$

$$u(x) = -K\,(x(t) - x_{goal})$$

$$u(x) = K x_{goal} - K x(t) = \theta_{FF} + \theta_{FB}\, x(t)$$

$$u(x) = [\theta_{FF} \;\; \theta_{FB}] \begin{bmatrix} 1 \\ x(t) \end{bmatrix}, \quad \theta_{FF} = K x_{goal} \quad \theta_{FB} = -K$$

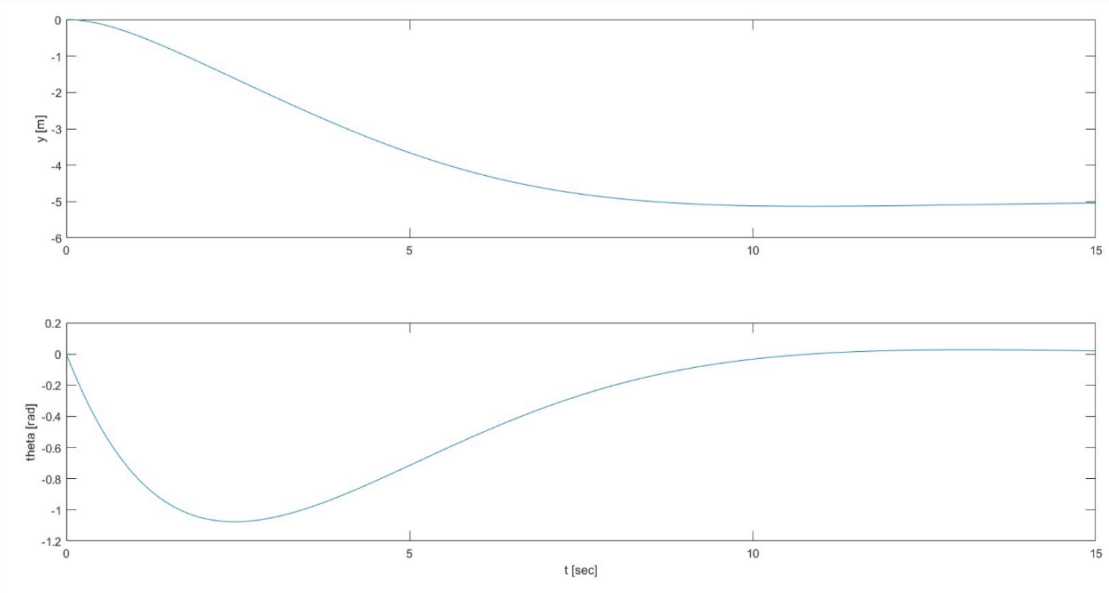
c)

i)

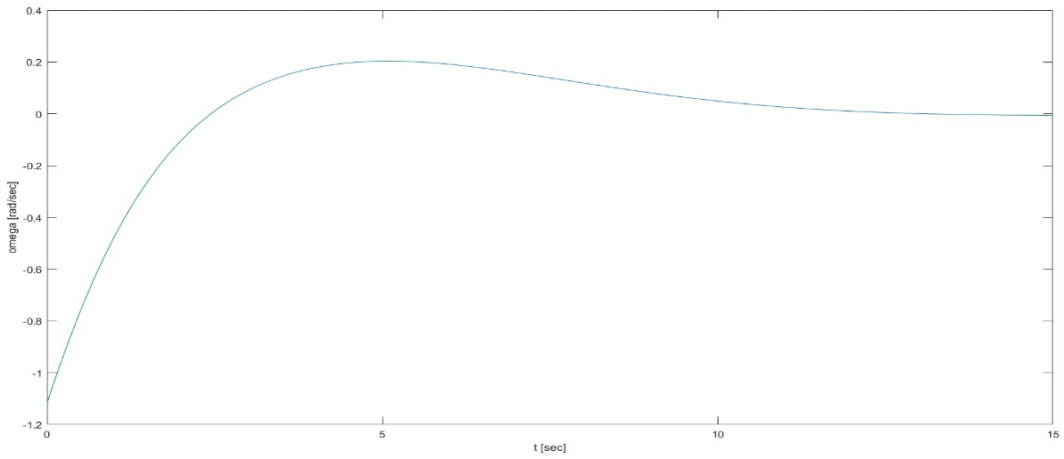
Increasing R places more emphasis on minimizing control effort, resulting in smoother and less agressive control activities, as we can see in the graph of omega when R is 200 and Q is kept constant. However, due to the penalization of control action, the reached state is farer from the goal, as also shown in the results.

Increasing Q, while mantaining R constant, emphasises minimizing the state error. Therefore, the col effort is bigger and more agressive, as shown in the results for Q equals diag(8,8).

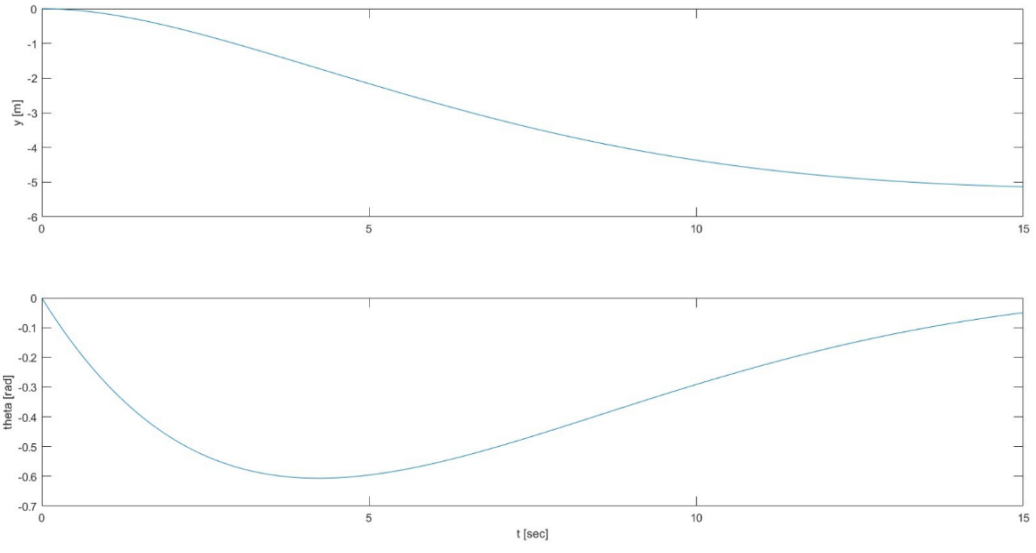
Graph For Default settings: R = 20 Q = diag(1,1) initial state = 0,0



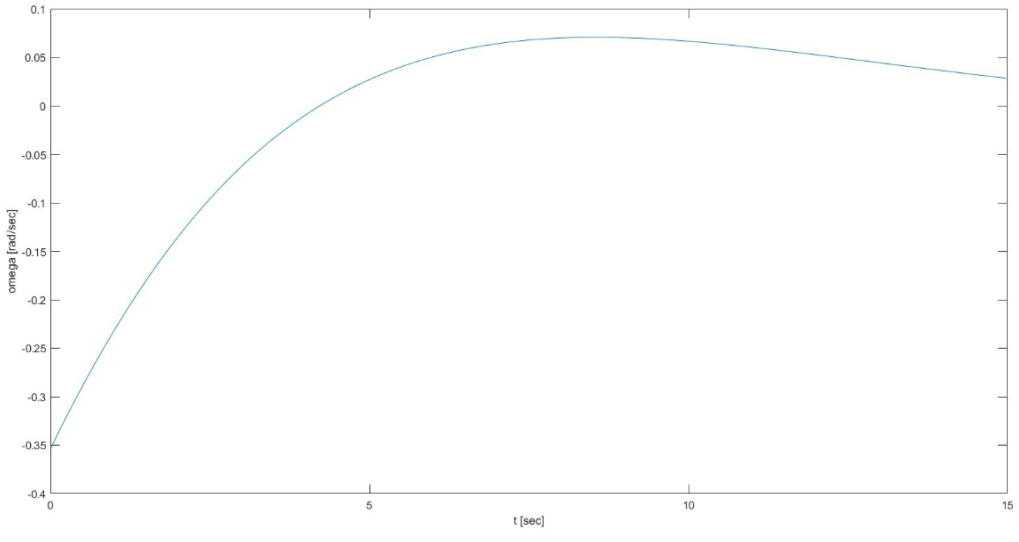

```
--- LQR ---

trajectory cost: 85.95
target state [-5.000; 0.000]
reached state [-5.040; 0.021]
>>
```



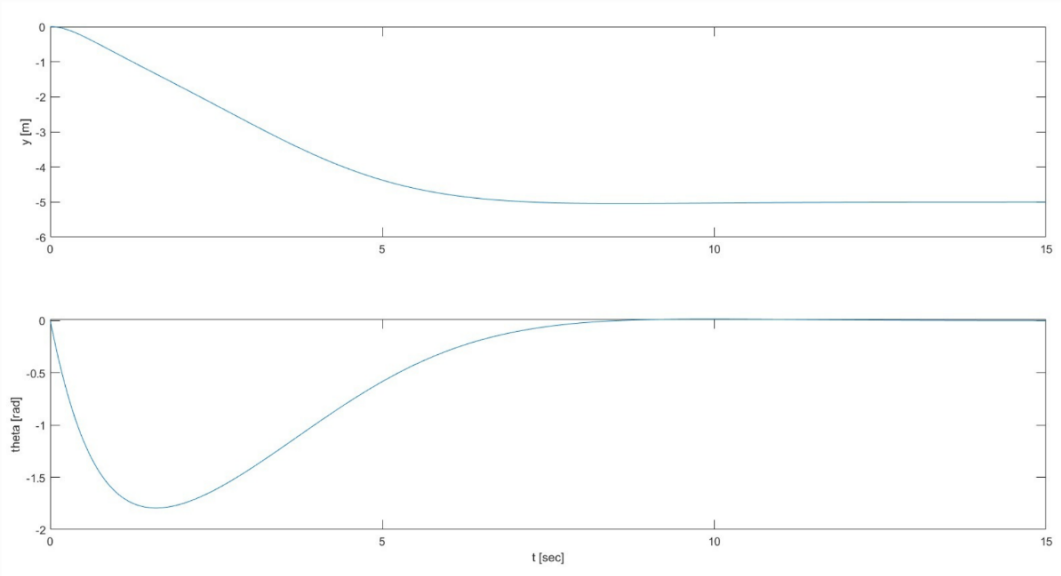
Dynamics with increased R: $R = 200$ $Q_{-s} = \text{diag}(1, 1)$



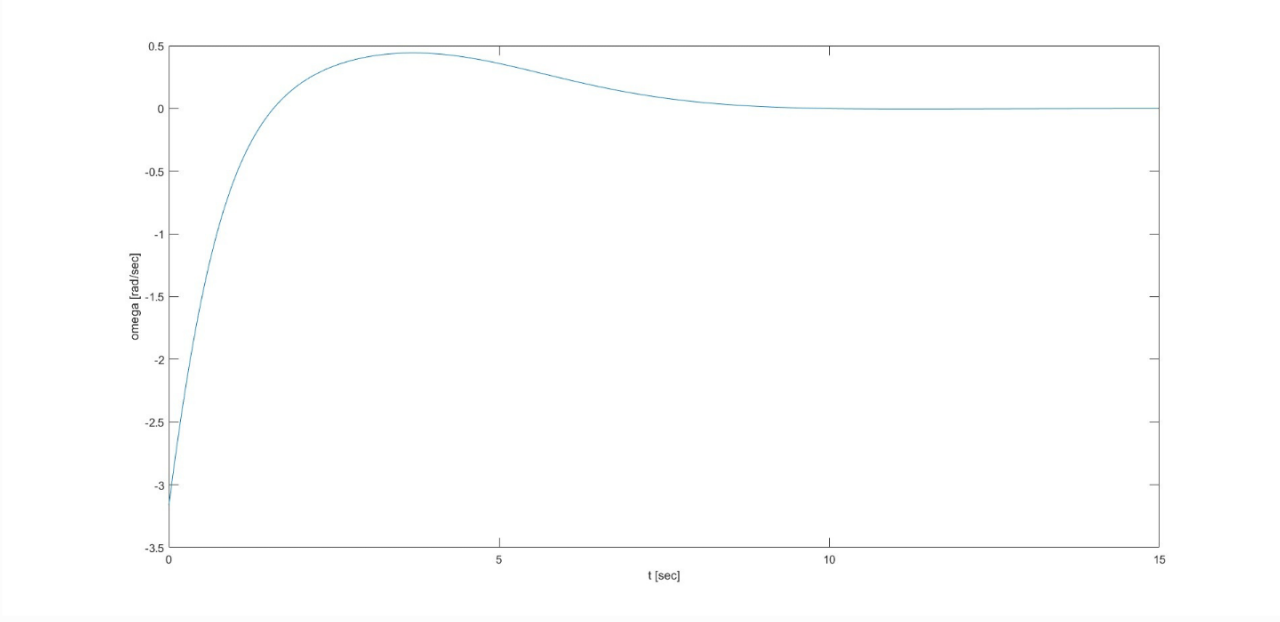

```
--- LQR ---

trajectory cost: 138.51
target state [-5.000; 0.000]
reached state [-5.133; -0.050]
```



Dynamics with increased Qs: R = 20 $Q_s = diag(8,8)$




```
--- LQR ---

trajectory cost: 541.50
target state [-5.000; 0.000]
reached state [-4.999; 0.000]
>>
```



ii)

The cost is smaller and the goal is reached faster if x_0 is closer to x_goal because it takes less control effort to reach the goal and also because the controller was developed on a linearization about x_eq = x_goal. Therefore, in the same way, for points farer from the goal the controll effort and the controller performance in general is worst.

```
%% [Problem 2.1 (c)] LQR Controller
% ========================== [TODO] LQR Design ==========================
% Design an LQR controller based on the linearization of the system about
% an equilibrium point (x_eq, u_eq). The cost function of the problem is
% specified in 'task_lqr.cost' via the method 'task_design()'.
%
A = [0 model.param.const_vel; 0 0];
B = [0;1];

[K,S,P] = lqr(A,B,task_lqr.cost.params.Q_s,task_lqr.cost.params.R_s);

theta_fb = -K;
theta_ff = K*task_lqr.goal_x;
theta = [theta_ff theta_fb]';
controller_lqr = repmat(theta, 1, N-1);

%
% =======================================================================
```

d)

$$x_{k+1} = \begin{bmatrix} y_{k+1} \\ h_{k+1} \end{bmatrix} = \begin{bmatrix} y_k \\ h_k \end{bmatrix} + \delta t \begin{bmatrix} \bar{v} \sin(h_k) \\ w_k \end{bmatrix}$$

Linearization

$$\begin{bmatrix} y_{k+1} \\ h_{k+1} \end{bmatrix} \approx \begin{matrix} \bar{y}_k + \delta t\, \bar{v} \sin(\bar{h}_k) + \delta y + \delta t\, \bar{v} \cos(\bar{h}_k) \cdot \delta h \\ \bar{h}_k + \delta h_k + \delta t \bar{w}_k + \delta t\, \delta w \end{matrix}$$

$$\begin{bmatrix} \delta y_{k+1} \\ \delta h_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & \delta t\, \bar{v} \cos(\bar{h}_k) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \delta y \\ \delta h \end{bmatrix} + \begin{bmatrix} 0 \\ \delta t \end{bmatrix} \delta w$$

```
function [A_k,B_k] = mobile_robot_lin(xk,uk,dt,v)
    A_k = [1 dt*v*cos(xk(2)); 0 1];
    B_k = [0; dt];
end
```

e)

$$J(x_0) = \frac{1}{2} (x(T) - x_{goal})^T Q_t (x(T) - x_{goal}) + \int_0^T \frac{1}{2} \left((x(t) - x_{goal})^T Q_s (x(t) - x_{goal}) + u(t)^T R_s u(t) \right) dt$$

$$J(x_0) = \underbrace{\frac{1}{2} (x_N - x_{goal})^T Q_t (x_N - x_{goal})}_{g_N(x_N)} + \delta t \sum_{k=0}^{N-1} \underbrace{\frac{1}{2} \left((x_k - x_{goal})^T Q_s (x_k - x_{goal}) + u_k^T R_s u_k \right)}_{g_k(x_k, u_k)}$$

$$\tilde{g}_N(x_N) = \bar{g}_N + q_N^T \delta x_N + \frac{1}{2} \delta x_N^T Q_N \delta x_N$$

$$\bar{g}_N(x_N) = g_N(\bar{x}_N) = \frac{1}{2} (\bar{x}_N - x_{goal})^T Q_t (\bar{x}_N - x_{goal})$$

$$q_N = \nabla_{x_N} g_N(\bar{x}_N) = Q_t (\bar{x}_N - x_{goal})$$

$$Q_N = \nabla^2_{x_N} g_N(\bar{x}_N) = Q_t$$

$$\tilde{g}_k(x_k, u_k) = \bar{g}_k + q_k^T \delta x_k + r_k^T \delta u_k + \frac{1}{2} \delta x_k^T Q_k \delta x_k + \frac{1}{2} \delta u_k^T R_k \delta u_k + \delta u_k^T P_k \delta x_k$$

$$\bar{g}_k = g_k(\bar{x}_k, \bar{u}_k) = \frac{1}{2}\left((\bar{x}_k - x_{goal})^T Q_s (\bar{x}_k - x_{goal}) + \bar{u}_k^T R_s \bar{u}_k\right) \cdot \delta t$$

$$q_k = \nabla_{x_k} g_k(\bar{x}_k, \bar{u}_k) = Q_s \cdot (\bar{x}_k - x_{goal})$$

$$Q_k = \nabla^2_{x_k} g_k(\bar{x}_k, \bar{u}_k) = Q_s$$

$$r_k = \nabla_{u_k} g_k(\bar{x}_k, \bar{u}_k) = R_s \cdot \bar{u}_k$$

$$R_k = \nabla^2_{u_k} g_k(\bar{x}_k, \bar{u}_k) = R_s$$

$$P_k = 0 \quad \text{(cross term)}$$

```
function [g_k,q_k,Q_k,r_k,R_k, P_k] = stage_cost_quad(Q_s, R_s, x_goal, dt, x_k, u_k)
    g_k = (1/2 * (x_k - x_goal)' * Q_s * (x_k - x_goal) + 1/2*R_s*u_k^2 ) * dt;
    q_k = Q_s * (x_k - x_goal);
    Q_k = Q_s;
    r_k = R_s * u_k;
    R_k = R_s;
    P_k = 0;
end
```

```
function [g_N,q_N,Q_N] = terminal_cost_quad(Q_t,x_goal,x_N)
    g_N = 1/2 * (x_N - x_goal)' * Q_t * (x_N - x_goal);
    q_N = Q_t * (x_N - x_goal);
    Q_N = Q_t;
end
```

f) Substituting $\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k$ in (9) and (8) with the given $g_k(x_k, u_k)$, we can expand equation (8):

$$J_k^*(x_k) \approx \min_{\delta u_k} \left((l_k + G_k \delta x_k)^T \delta u_k + \frac{1}{2} \delta u_k^T H_k \delta u_k + c_k \right)$$

Where:

$$l_k = r_k + B_k^T s_{k+1}$$

$$G_k = B_k^T S_{k+1} A_k$$

$$H_k = R_k + B_k^T S_{k+1} B_k$$

$$c_k = g_k(\bar{x}_k, \bar{u}_k) + \bar{s}_{k+1} + (q_k^T + s_{k+1}^T A_k) \delta x_k + \frac{1}{2} \delta x_k^T (Q_k + A_k^T S_{k+1} A_k) \delta x_k$$

Than:

For $\min_{\delta u_k}$ $\rightarrow$ $\dfrac{d J_k^*(x_k)}{d \delta u} = (l_k + G_k \delta x_k) + H_k \delta u_k^* = 0$

$$\delta u_k^* = -H_k^{-1} l_k - H_k^{-1} G_k \cdot (x_k - \bar{x}_k)$$

$$u_k = \bar{u}_k + \delta u_k^*$$

$$u_k = \underbrace{\bar{u}_k - H_k^{-1} l_k + H_k^{-1} G_k \bar{x}_k}_{\theta_{k,FF}} + \underbrace{\left(- H_k^{-1} \cdot G_k \cdot x_k \right)}_{\theta_{k,FB}}$$

g)

Given:

$$J_k^*(x_k) \approx \min_{\delta u_k} \left((l_k + G_k \delta x_k)^T \delta u_k + \frac{1}{2} \delta u_k^T H_k \delta u_k \right)$$

We can substitute $\delta u_k = \delta u_k^*$, giving us:

$$J_k^*(\delta x_k) = \bar{s}_k + s_k^T \delta x_k + \frac{1}{2} \delta x_k^T S_k \delta x_k = (l_k + G_k \delta x_k)^T \delta u_k^* + \frac{1}{2} \delta u_k^{*T} H_k \delta u_k^* + c_k$$

Which leads to:

$$\bar{s}_k = \bar{g}_k + \bar{s}_{k+1} + \frac{1}{2} \delta u_{k,FF}^{*T} H_k \delta u_{k,FF}^* + \delta u_{k,FF}^{*T} l_k$$

$$s_k = q_k + A_k^T s_{k+1} + K_k^T H_k \delta u_{k,FF}^* + K_k^T l_k + G_k^T \delta u_{k,FF}^*$$

$$S_k = Q_k + A_k^T S_{k+1} A_k + K_k^T H_k K_k + K_k^T G_k + G_k^T K_k$$

where

$$K_k = -H_k^{-1} \cdot G_k \quad \text{and} \quad \delta u^*_{k,FF} = -H_k^{-1} l_k$$

With $\tilde{J}^*_N(x_N) = \tilde{g}_N(x_N)$ we obtain:

$$\bar{s}_N + s_N^T \delta x_N + \frac{1}{2} \delta x_N^T S_N \delta x_N = \bar{g}_N + q_N^T \delta x_N + \frac{1}{2} \delta x_N^T Q_N \delta x_N$$

by matching terms we obtain

$$\bar{s}_N = \bar{g}_N \qquad s_N = q_N \qquad S_N = Q_N$$

h)

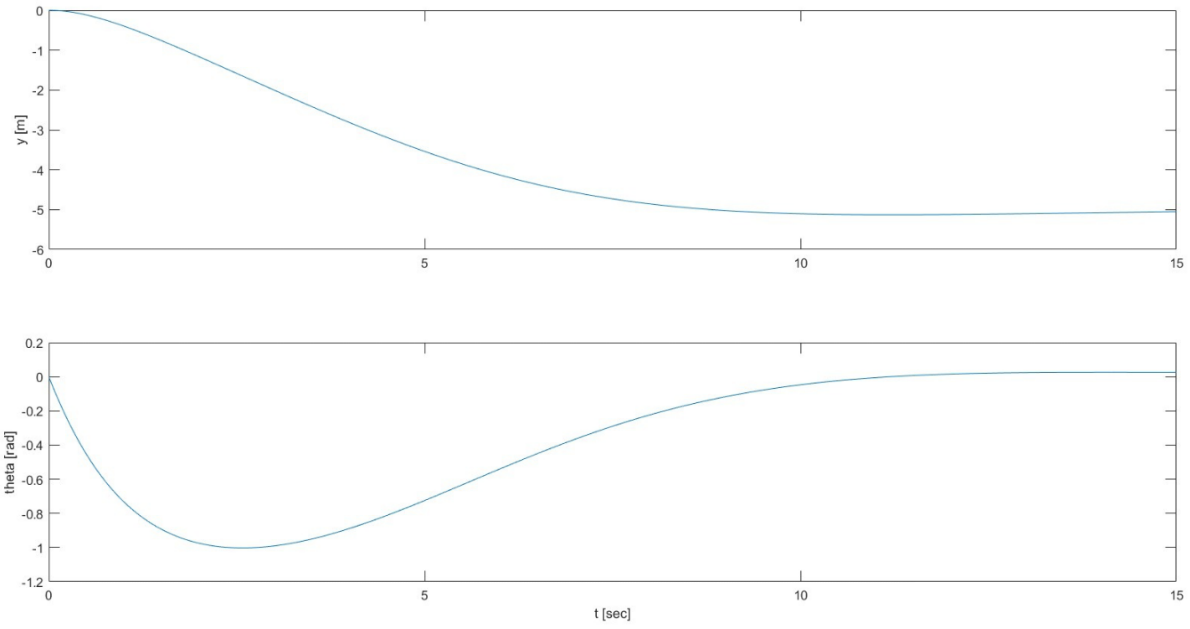
```
function [theta_kff,theta_kfb,s_k_,s_k,S_k] = update_policy(A_k,B_k,g_k,q_k,Q_k,r_k,R_k,P_k,s_k1_,s_k1,S_k1,x_k,u_k)
    l_k = r_k + B_k'*s_k1;
    G_k = B_k'*S_k1*A_k;
    H_k = R_k+B_k'*S_k1*B_k;

    K_k = -H_k\G_k;
    du_kff = -H_k\l_k;

    theta_kff = u_k+du_kff-K_k*x_k;
    theta_kfb = K_k;

    s_k_ = g_k + s_k1_ + 1/2 * (du_kff'*H_k*du_kff) + du_kff'*l_k;
    s_k = q_k + A_k'*s_k1 + K_k'*H_k*du_kff + K_k'*l_k + G_k'*du_kff;
    S_k = Q_k + A_k'*S_k1*A_k + K_k'*H_k*K_k + K_k'*G_k + G_k'*K_k;
end
```

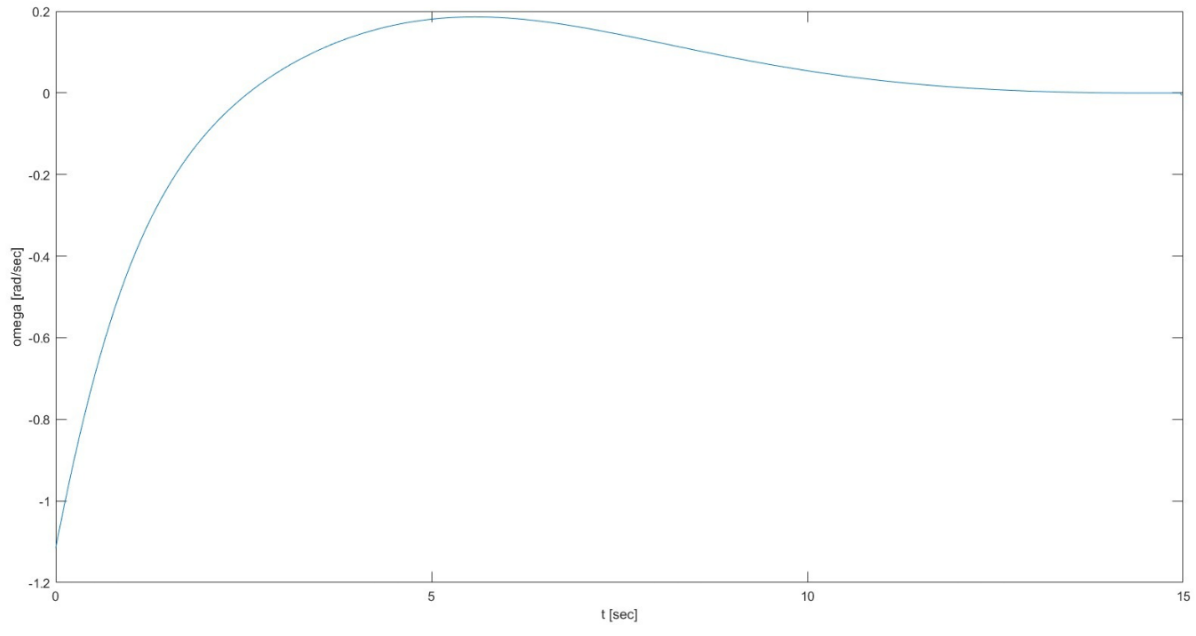
i)

For iterations do:
  forward pass = mobile_robot_sim(...,controller_ilqc)
  gn, qn, Qn = terminal_cost_quad(...)
  get sk_, sk, Sk

  starts backward pass
  for k = 1:1:N-2
    a_k,b_k = mobile_robot_lin(...)
    stage params = stage_cost_quad(...)
    feedback, feedforward and updated s's = update_policy(...)
    controller_ilqc(:,(end-k)) = feedback and feedforward gains
  End
End

j) i) ILQC Simulation : Initial state: [0,0]



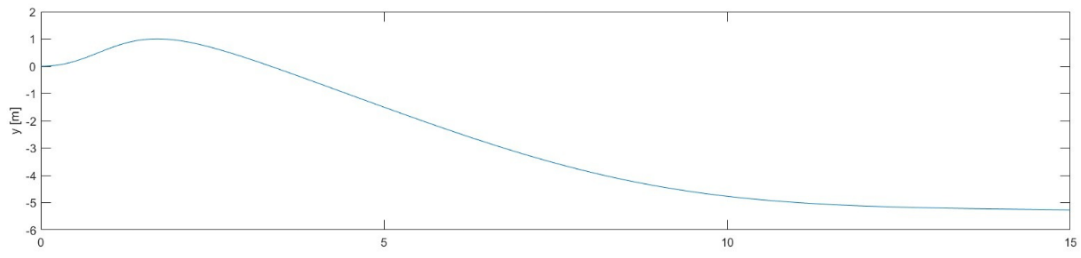


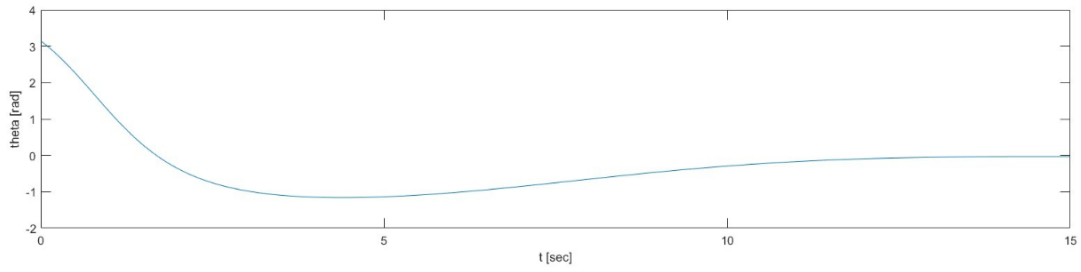
Generally, the LQR conttroller is ideal for simpler, linear problems while the ILQC is used for more complex, nonlinear systems, providing iterative improvements. In this case, where the start point was set to [0,0], the performance of the LQR controller was already good enough for the problem, and, therefore, both controllers performed similiarly.

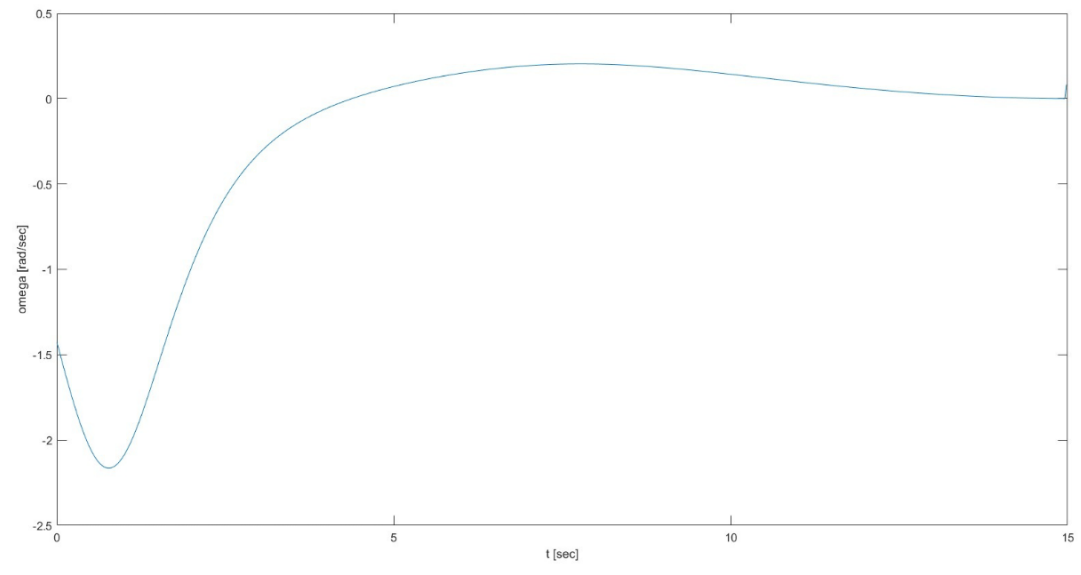
.

```
--- LQR ---

trajectory cost: 85.95
target state [-5.000; 0.000]
reached state [-5.040; 0.021]
initial state [0.000; 0.000]
>> main_p1_ilqc
--- ILQC ---

trajectory cost: 85.63
target state [-5.000; 0.000]
reached state [-5.052; 0.026]
initial state [0.000; 0.000]
```

ILQC Dynamics : Initial state [0, pi]







For the case where the start point was set to [0,pi], the diference between the two controllers was a bit more perceptible. The cost of the trajectory of the ILQC controller was smaller than for the LQR controller. However the ILQC controller presented some instability, the reached state was farer from the goal than with the LQR controller. This instability could be because fo numerical factors, or because the iterative approximations

```
--- LQR ---

trajectory cost: 317.28
target state [-5.000; 0.000]
reached state [-5.096; 0.030]
initial state [0.000; 3.142]
>> main_p1_ilqc
--- ILQC ---

trajectory cost: 303.90
target state [-5.000; 0.000]
reached state [-5.266; -0.030]
initial state [0.000; 3.142]
```

j) ii)

Both controllers would suffer with the inacuracies of the model because both rely on the assumption that the dynamics is correct. However, they boh have some robustness in comparisson to any open-loop controller due to the feedback action. The ILQC would even perform better than the LQC due to the iterative approach.

```
%% [Problem 2.1 (j)] Iterative Linear Quadratic Controller
% ========================= [TODO] ILQC Design ========================
% Design an ILQC controller based on the linearized dynamics and
% quadratized costs. The cost function of the problem is specified in
% 'task_ilqc.cost' via the method 'task_design()'
%

for i = 1:1:task_ilqc.max_iteration
    sim_out_it = mobile_robot_sim(model,task_ilqc,controller_ilqc); %Initial Forward Pass
    [g_n_,q_n,Q_n] = terminal_cost_quad(task_ilqc.cost.params.Q_t, task_ilqc.goal_x, sim_out_it.x(:,end));
    s_k_ = g_n_;
    s_k = q_n;
    S_k = Q_n;

    for k = 1:1:N-2 %Backward Pass
        [A_k,B_k] = mobile_robot_lin(sim_out_it.x(:,end-k),sim_out_it.u(:,end-k),task_ilqc.dt,model.param.const_vel);
        [g_k_,q_k,Q_k,r_k,R_k,P_k] = stage_cost_quad(task_ilqc.cost.params.Q_s,task_ilqc.cost.params.R_s,task_ilqc.goal_x,task_ilqc.dt,sim_out_it.x(:,end-k),sim_out_it.u(:,end-k));
        [theta_kff,theta_kfb,s_k_,s_k,S_k] = update_policy(A_k,B_k,g_k_,q_k,Q_k,r_k,R_k,P_k,s_k_,s_k,S_k,sim_out_it.x(:,end-k),sim_out_it.u(:,end-k));
        controller_ilqc(:,(end-k)) = [theta_kff theta_kfb]';
    end
end

%
% ====================================================================
```

Problem 2.2

a)

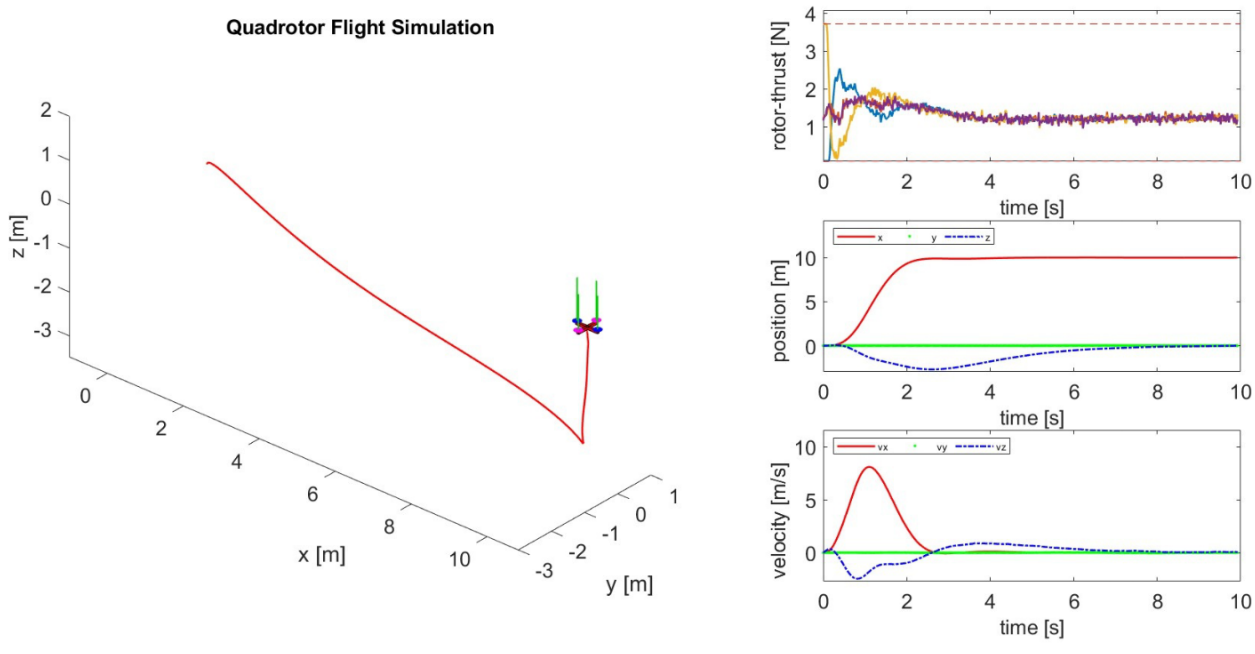
```
%% [Problem 2.2 (a)] Find optimal feedback gains according to an LQR controller
% Make use of the elements in Task.cost for linearization points and cost
% functions.

% ====================================================================
% [Todo] Define the state input around which the system dynamics should be
% linearized assuming that most of the time the quadrotor flies similar to
% hovering (Fz != 0)
%
% x_lin = ...;
% u_lin = ...;
x_lin = Task.cost.x_eq;
u_lin = Task.cost.u_eq;

% [Todo] The linearized system dynamics matrices A_lin B_lin describe the
% dynamics of the system around the equilibrium state. Use Model.Alin{1}
% and Model.Blin{1}.
%
% A_lin = ...;
% B_lin = ...;
A_lin = Model.Alin{1}(x_lin,u_lin,Model.param.syspar_vec);
B_lin = Model.Blin{1}(x_lin,u_lin,Model.param.syspar_vec);

% [Todo] Compute optimal LQR gain (see command 'lqr')
% Quadratic cost defined as Task.cost.Q_lqr,Task.cost.R_lqr
%
% K = ...;
[K,S,P] = lqr(A_lin,B_lin,Task.cost.Q_lqr,Task.cost.R_lqr);
% ====================================================================
```

b)

```
%% [Problem 2.2 (b)] Drive system to Task.goal_x with LQR Gain + feedforward
        % ========================================================================
        % [Todo] Define equilibrium point x_eq correctly and use it to
        % generate feedforward torques (see handout Eqn.(12) and notes
        % above).
        %
        % x_ref = ...; % reference point the controller tries to reach
        % theta = ...; % dimensions (13 x 4)
        x_ref = Task.goal_x;
        u_ref = u_lin;
        theta_ff = [u_ref + K*x_ref];
        theta_fb = -K;
        theta = [theta_ff theta_fb]';

        % ========================================================================
```

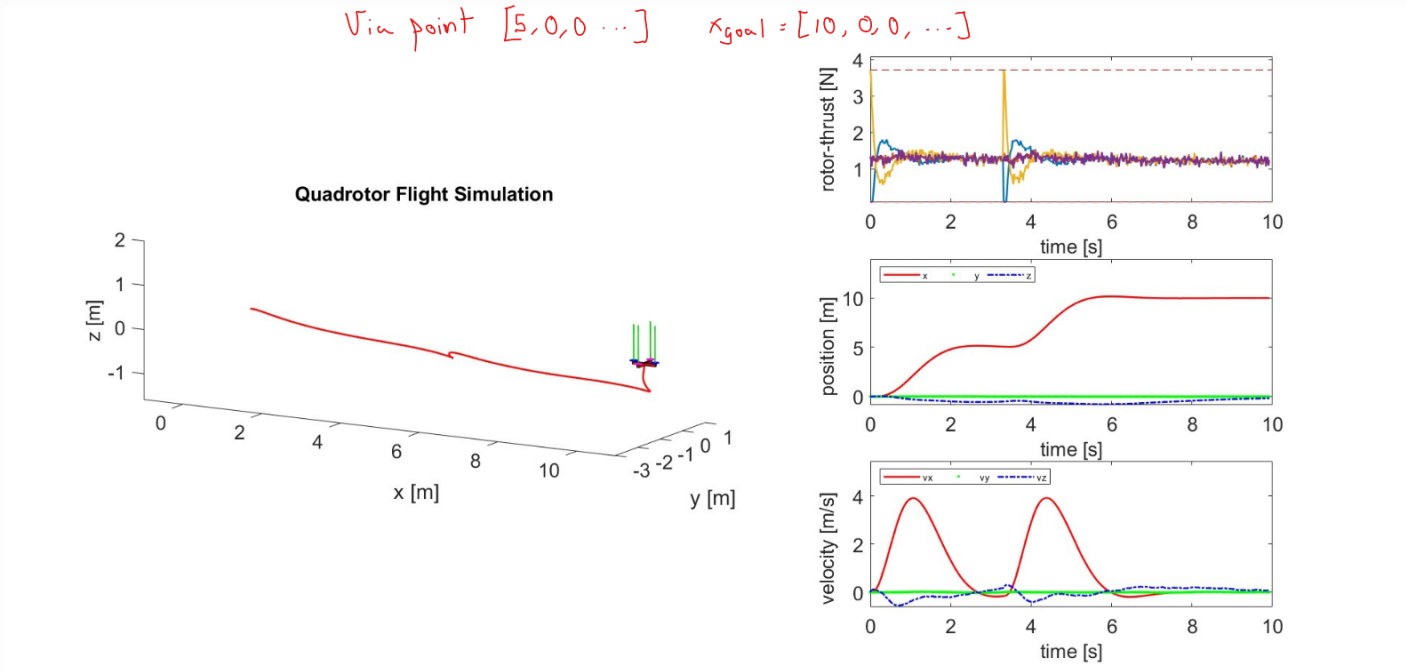
i)

Dynamics for $x_{goal} = [10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

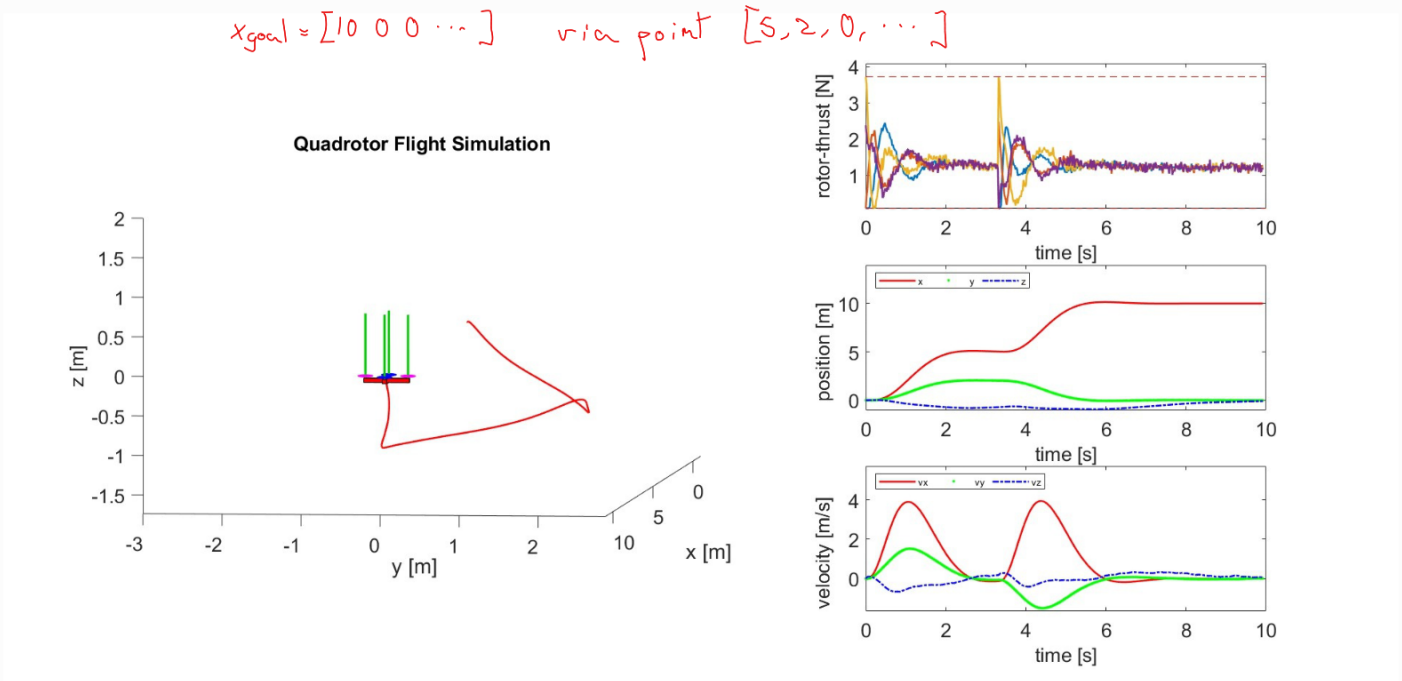


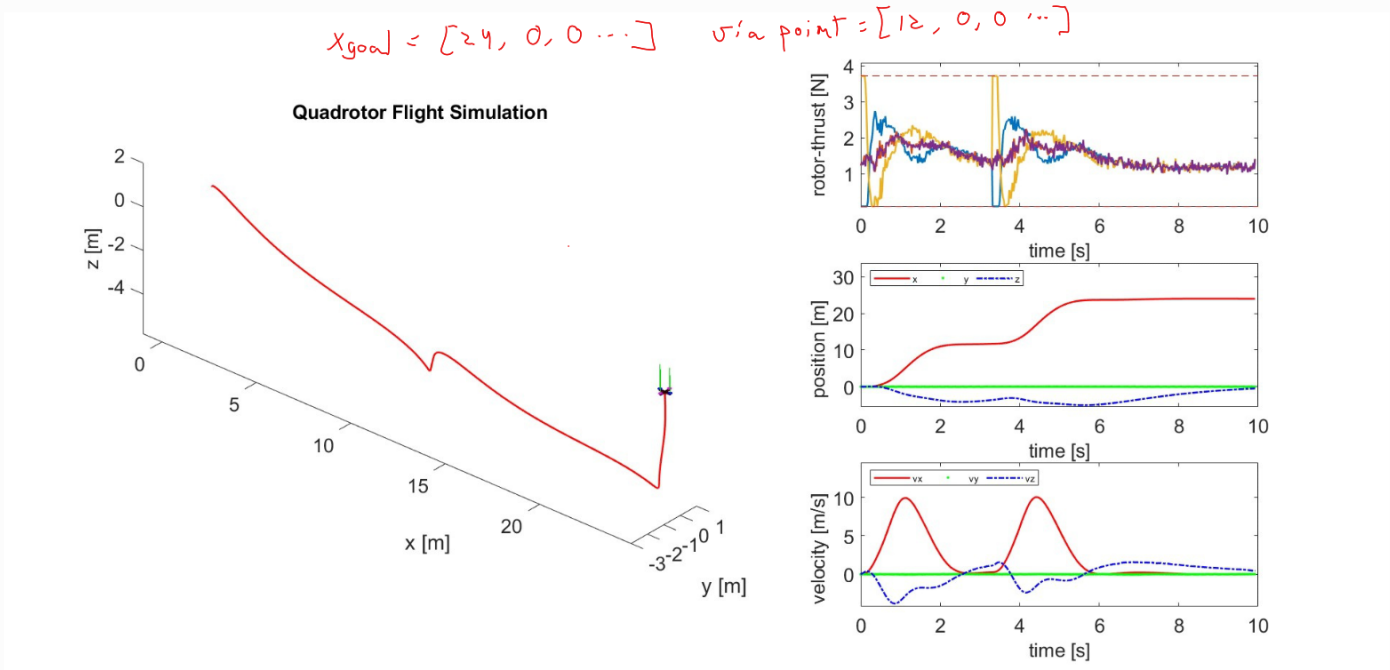
ii) The controller reaches the goal For xgoal = 0, but it can not go further than xgoal = 12 without being unstable. That limit is reasonable because the lqr uses only one point for linearization.

c)



i) With a via point in the middle of the way the trajectory is smoother, the drone doesnt loose altitude in the z direction and the cost is similar, 205.710 for the goal-state and 213.685 with the via point.





Ii) The via point adds a step to the system, this way it is as if the state goal was closer to the initial state, therefore the controller can perform better with a shorter "horizon".

iii) Looking at the graph it is clear that the drone had to slow down to reach the via point and accelerate back again, this is a performance disadvantage with via points, the transitions might not be optimal.

```
%% [Problem 2.2 (c)] Drive system to Task.goal_x through via-point p1.
        t1 = Task.vp_time;
        p1 = Task.vp1; % steer towards this input until t1

        for t=1:Nt-1
            % ================================================================
            % [Todo] x_ref at current time
            % ...
            %
            u_ref = u_lin;
            theta_fb = -K;

            if t*Task.dt < t1
             x_ref = p1;
             theta_ff = [u_ref + K*x_ref];

            else
             x_ref = Task.goal_x;
             theta_ff = [u_ref + K*x_ref];
            end
            % [Todo] time-varying theta matrices for every time step Nt
            % (see handout Eqn.(12)) (size: 13 x 4)
            % theta_k = ...;
            theta_k = [theta_ff theta_fb]';
            % ================================================================
```

d)

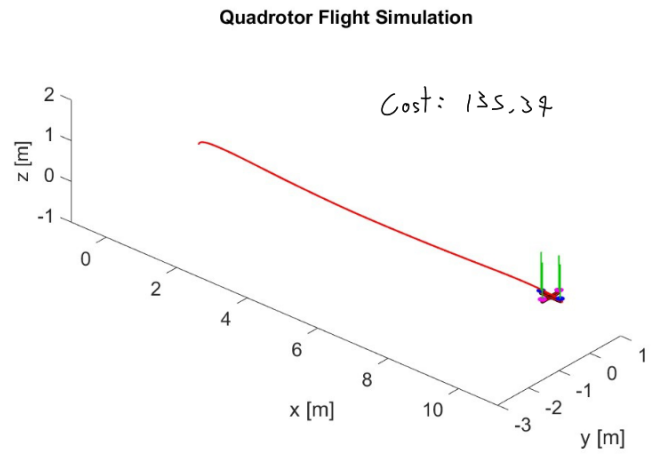
```
%% [Problem 2.2 (d)] Implementation of ILQC controller
% Each ILQC iteration approximates the cost function as quadratic around the
% current states and inputs and solves the problem using DP.
i  = 1;
while ( i <= Task.max_iteration && ( norm(squeeze(duff)) > 0.01 || i == 1 ))

    %% Forward pass / "rollout" of the current policy
    % =====================================================================
    % [Todo] rollout states and inputs
    sim_out = Quad_Simulator(Model,Task,Controller);
    % =====================================================================

    % pause if cost diverges
    cost(i) = Calculate_Cost(sim_out, gbar_fun, gbarN_fun);
    fprintf('Cost of Iteration %2d (metric: ILQC cost function!): %6.4f \n', i-1, cost(i));

    if ( i > 1 && cost(i) > 2*cost(i-1) )
        fprintf('It looks like the solution may be unstable. \n')
        fprintf('Press ctrl+c to interrupt iLQG, or any other key to continue. \n')
        pause
    end

    %% Solve Riccati-like equations backwards in time
    % =====================================================================
    % [Todo] define nominal state and control input trajectories (dim by
    % time steps). Note: sim_out contains state x, input u, and time t
    %
    X0 = sim_out.x; % x_bar
    U0 = sim_out.u; % u_bar
    T0 = sim_out.t; % t
    % =====================================================================

% "Backward pass": Calculate the coefficients (s,Sv,Sm) for the value
% functions at earlier times by proceeding backwards in time
% (DP-approach)
for k = (length(sim_out.t)-1):-1:1

    % state of system at time step n
    x0 = X0(:,k);
    u0 = U0(:,k);

    % =====================================================================
    % [Todo] Discretize and linearize continuous system dynamics Alin
    % around specific pair (x0,u0). See exercise sheet Eqn. (18) for
    % details.
    %
    Alin = Model.Alin{1}(x0,u0,Model.param.syspar_vec);
    Blin = Model.Blin{1}(x0,u0,Model.param.syspar_vec);
    A = Alin*Task.dt + eye([size(Alin*Task.dt)]);
    B = Blin*Task.dt;
    % =====================================================================
```

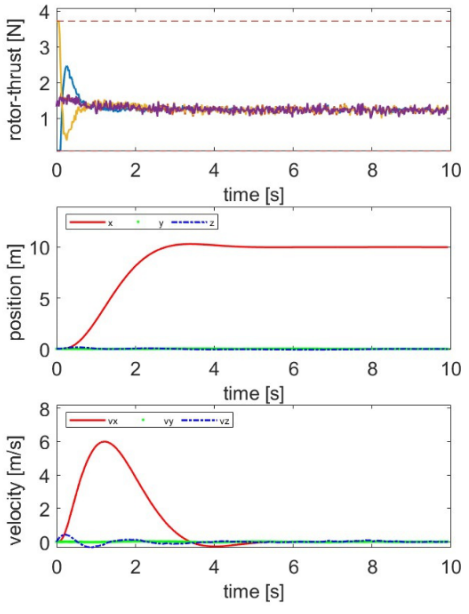
```
    % ================================================================
    % [Todo] quadratize cost function
    % [Note] use function {gbar_fun, gv_fun, Qm_fun, rv_fun, Rm_fun,
    % Pm_fun} provided above.
    %
    t0 = T0(:,k);
    gbar = gbar_fun(t0,x0,u0);
    Qv = gv_fun(t0,x0,u0);
    Qm = Qm_fun(t0,x0,u0);
    Rv = rv_fun(t0,x0,u0);
    Rm = Rm_fun(t0,x0,u0);
    Pm = Pm_fun(t0,x0,u0);
    % ================================================================

    % ================================================================
    % [Todo] control dependent terms of cost function (Problem 2.2 (f))
    %
    l = Rv + B'*Sv(:,k+1);  % linear control dependent
    G = Pm + B'*Sm(:,:,k+1)*A;  % control and state dependent
    H = Rm + B'*Sm(:,:,k+1)*B;  % quadratic control dependent
    %
    H = (H+H')/2; % ensuring H remains symmetric; do not delete!
    % ================================================================
```

```
% ================================================================
% [Todo] the optimal change of the input trajectory du = duff +
% K*dx (Problem 2.2 (f))
%

K(:,:,k) = -H\G;
duff(:,:,k) = -H\l;
% ================================================================

% ================================================================
% [Todo] Solve Riccati-like equations for current time step n
% (Problem 2.2 (g))
%
Sm(:,:,k) = Qm + A'*Sm(:,:,k+1)*A + K(:,:,k)'*H*K(:,:,k) + K(:,:,k)'*G + G'*K(:,:,k);
Sv(:,k) = Qv + A'*Sv(:,k+1) + K(:,:,k)'*H*duff(:,:,k) + K(:,:,k)'*l + G'*duff(:,:,k);
s(k) = gbar + s(k+1) + 0.5*duff(:,:,k)'*H*duff(:,:,k) + duff(:,:,k)'*l;
% ================================================================
```

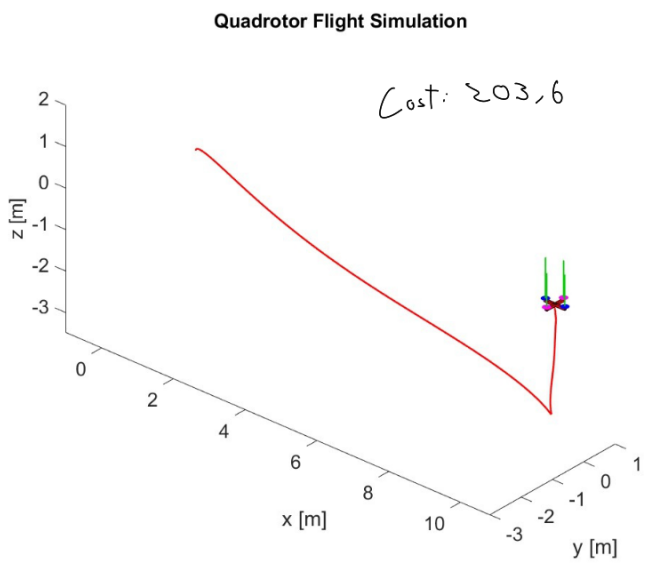
```
% ========================================================================
% [Todo] feedforward control input
%
U0 = permute(U0, [3, 1, 2]);
dUff = permute(dUff, [2, 1, 3]);
KX0 = zeros(1, m, N-1);
for i = 1:1:N-1
    KX0(:,:,i) = K(:,:,i)*X0(:,i);
end
%
theta_ff = U0 + dUff - KX0;
% ========================================================================
```

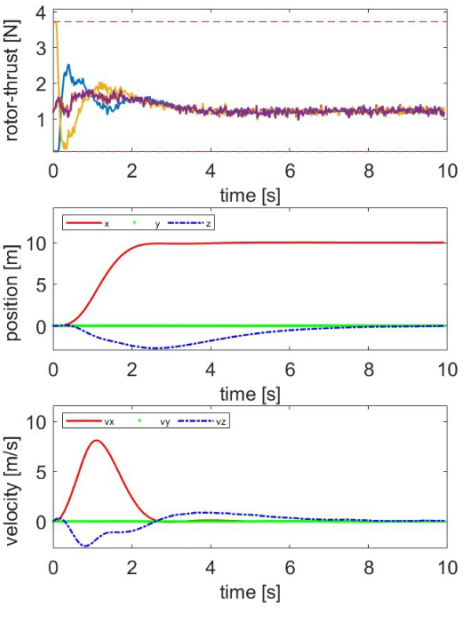
ILQC: Default settings





LQR: Default Settings





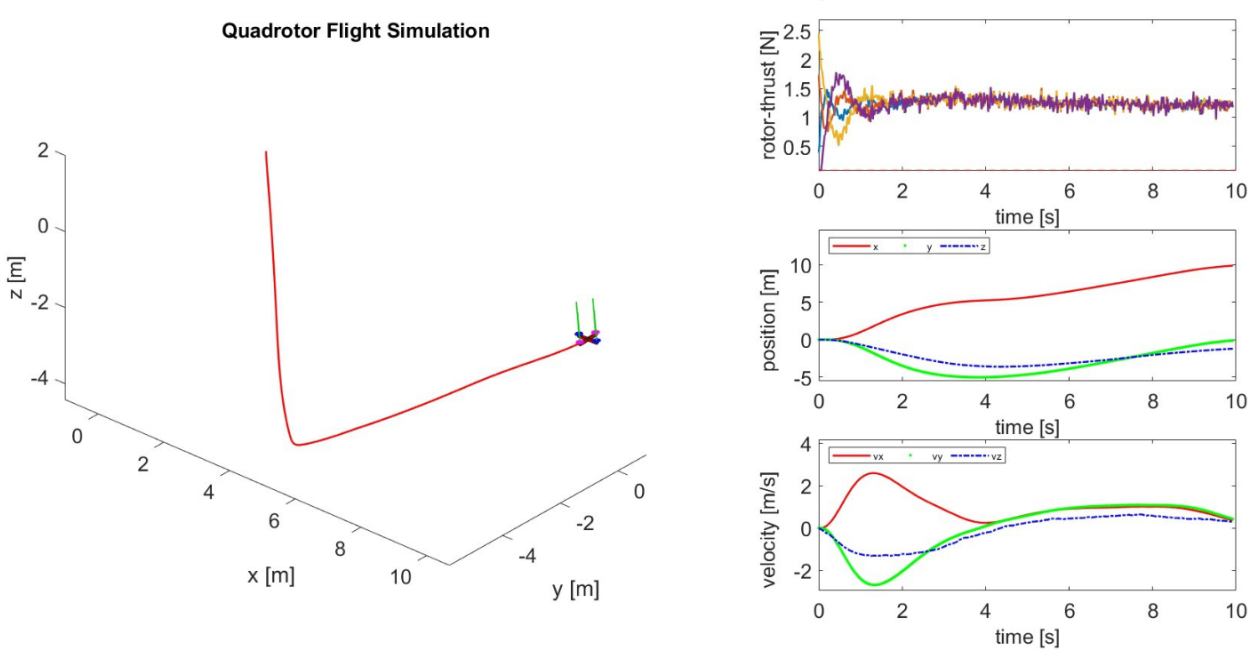
i) Looking at the graphs it is clear that ILQC performs better both in the trajectory, keeping the altitude, and in cost comparison. The cost for the ILQC was 135,37 against 203,6 for the LQR.

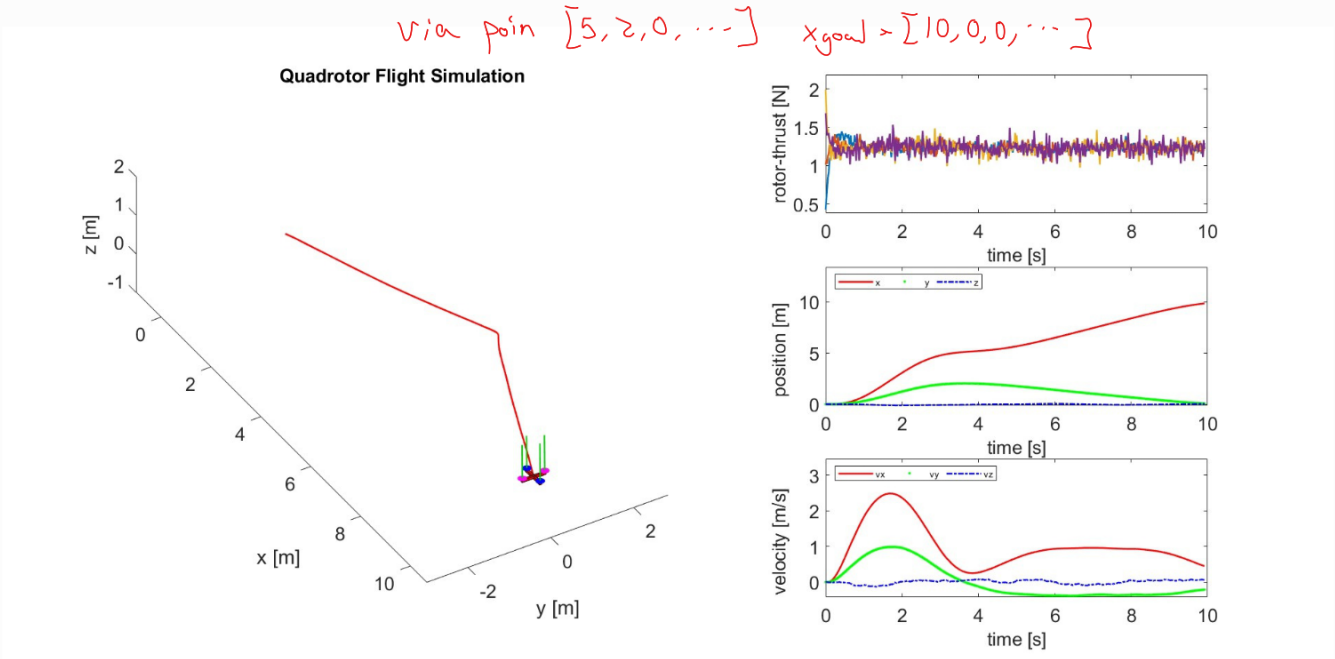
ii) ILQC uses a iterative approach to solve subproblems of the bigger and more complex problem. Therefore, when this type of controller can perform better for distant goals by breaking it into shorter distances.

e)

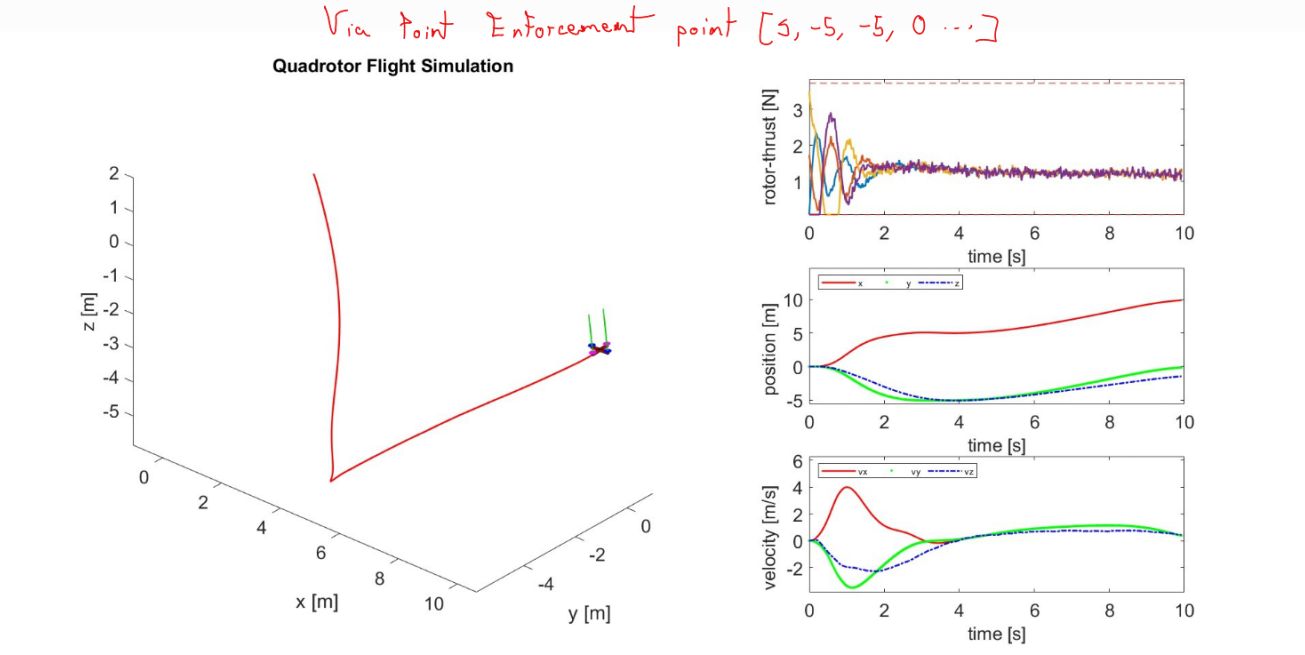
```
%% [Problem 2.2 (e)] Include via_point p1 in the ILQC cost function formulation
    p1 = Task.vp1;      % p1 = Task.vp2 also try this one
    t1 = Task.vp_time;

    % ====================================================================
    % [Todo] Define an appropriate weighting for way points (see
    % handout Eqn.(19)) Hint: Which weightings must be zero for the
    % algorithm to determine optimal values?
    %
    Q_vp = diag([3 3 3 0 0 0 0 0 0 0 0 0]);
    % ====================================================================

    % don't penalize position deviations, drive system with final cost
    Cost.Qm(1:3,1:3) = zeros(3);

    % ====================================================================
    % [Todo] Define symbolic cost function.
    % Note: Use function "viapoint(.)" below
    %
    viapoint_cost = viapoint(t1,p1,x,t_sym,Q_vp);
    Cost.h = simplify((x-x_goal)'*Cost.Qmf*(x-x_goal));
    Cost.l = simplify( (x-Cost.x_eq)'*Cost.Qm*(x-Cost.x_eq) + (u-Cost.u_eq)'*Cost.Rm*(u-Cost.u_eq)) + viapoint_cost;
    % ====================================================================
```

via point [5, -5, -5, ...] x_{goal} = [10, 0, 0, ...]





i) The weighting matrix Q_vp can be defined with non zero elements only for the position elements of the state, therefore keeping it zero the weight for the velocities. In this task Q_vp was attributed with the weight 3 for x, y and z, and zero weight for the other state elements.



ii) Exact passing through via point, as shown in the Figure above can be assured by increasing the weights in the Q_vp matrix. For this case it was used value 30 weights for the x,y and z states. A soft suggestion would be to keep those weights low, 1 for example.