AER1517 Control for Robotics

Assignment 1: Model-Based Iterative Linear Quadratic Control

Prof. Angela Schoellig

Institute for Aerospace Studies
UNIVERSITY OF TORONTO

VECTOR INSTITUTE

DYNAMIC SYSTEMS LAB

General Information

- Download handout and script templates from Quercus
  - Problem 1.1 Mobile robot
  - Problem 1.2 Quadrotor
- Due on Feb. 17 (Monday) 23:59
- Submission through *Gradescope*
  - A single PDF with solutions and requested scripts
  - Both typed and scanned handwritten solutions are accepted
- Office hour: Feb. 13 (Thursday) 14:00 @ UTIAS

Problem 1.1 Mobile Robot | Overview

- Goal: Control a unicycle-type mobile robot to move along a straight line
- Two components:
  - LQR
  - ILQC

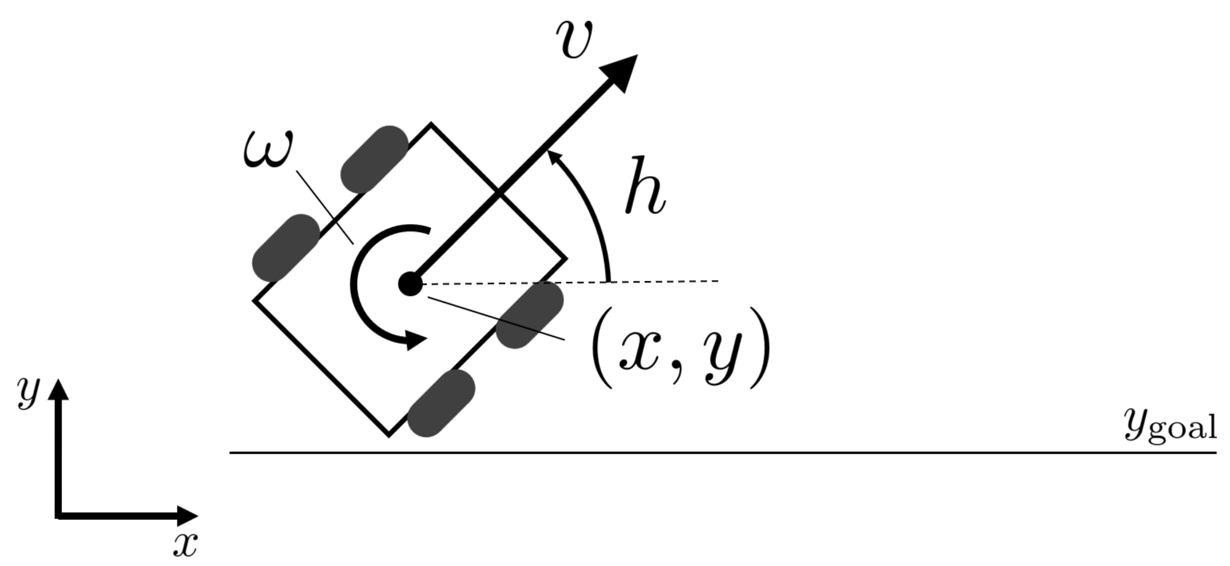
Problem 1.1 Mobile Robot| Model

- State: $\mathbf{x} = [y, h]^T \in \mathbb{R}^2$
- Input: $\mathbf{u} = \omega \in \mathbb{R}^1$

- System dynamics:

$$\dot{y}(t) = v(t)\ \sin\big(h(t)\big)$$
$$\dot{h}(t) = \omega(t)$$

(assume constant forward speed $v(t) = \bar{v}$)



Problem 1.1 Mobile Robot | Code Structure (LQR)

DYNAMIC SYSTEMS LAB

main_p1_lqr.m

- task_design()
- generate_model()
- LQR design
- mobile_robot_sim()
- plot_results()

EDITOR PUBLISH VIEW

main_p1_lqr.m main_p2_ilqc.m task_design.m

```matlab
% define task
task_lqr = task_design();
N = length(task_lqr.start_time:task_lqr.dt:task_lqr.end_time);

% add model
const_vel = 1; % desired forward speed
model = generate_model(const_vel);

% initialize controller
controller_lqr = zeros(3, N-1);

% save directory
save_dir = './results/';

% flags
plot_on = true;
save_on = true;

%% [Problem 1.1 (c)] LQR Controller
% ========================== [TODO] LQR Design ==========================
% Design a 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()'.
%
%
% =======================================================================

%% Simulation
sim_out_lqr = mobile_robot_sim(model, task_lqr, controller_lqr);
fprintf('--- LQR ---\n\n');
fprintf('trajectory cost: %.2f\n', sim_out_lqr.cost);
fprintf('target state [%.3f; %.3f]\n', task_lqr.goal_x);
fprintf('reached state [%.3f; %.3f]\n', sim_out_lqr.x(:,end));

%% Plots
if plot_on
    plot_results(sim_out_lqr);
end
```

Problem 1.1 Mobile Robot | (a)-(c) LQR Design

- Assume infinite horizon
- Linearize dynamics about one point $(\mathbf{x}_{\text{op}}, \mathbf{u}_{\text{op}})$

$$\dot{\mathbf{x}}(t) = \mathbf{F}(\mathbf{x}(t), \mathbf{u}(t)) \xrightarrow[\text{linearize}]{} \delta\dot{\mathbf{x}}(t) = \mathbf{A}_{\text{lin}}\delta\mathbf{x}(t) + \mathbf{B}_{\text{lin}}\delta\mathbf{u}(t)$$

where $\delta\mathbf{x}(t) = \mathbf{x}(t) - \mathbf{x}_{\text{op}}$ and $\delta\mathbf{u}(t) = \mathbf{u}(t) - \mathbf{u}_{\text{op}}$

- Given the (Q, R) matrices defined in task_design(), design a LQR controller with lqr() function in Matlab
  - Note the convention of lqr() (see documentation)

DYNAMIC SYSTEMS LAB

- Control law (static gains):

$$\mathbf{u}(t) = \mathbf{K}(\mathbf{x}(t) - \mathbf{x}_{\text{goal}}) = \underbrace{\begin{bmatrix} \boldsymbol{\theta}_{\text{ff}}^T & \boldsymbol{\theta}_{\text{fb}}^T \end{bmatrix}}_{\boldsymbol{\theta}^T} \begin{bmatrix} 1 \\ \mathbf{x}(t) \end{bmatrix} = \boldsymbol{\theta}_{\text{ff}}^T + \boldsymbol{\theta}_{\text{fb}}^T\, \mathbf{x}(t)$$

- 'Repmat' controller parameters for simulation: $\boldsymbol{\Theta} = \underbrace{[\boldsymbol{\theta}, \boldsymbol{\theta}, ..., \boldsymbol{\theta}]}_{(N-1)\text{ times}} \in \mathbb{R}^{3\times(N-1)}$
- Complete main_p1_lqr.m
- Run and observe
  - Effect of changing Q and R matrices
  - Performance for different initial states

Problem 1.1 Mobile Robot | Code Structure (ILQC)

DYNAMIC SYSTEMS LAB

main_p1_ilqc.m

- task_design()
- generate_model()
- ILQC design
- mobile_robot_sim()
- plot_results()

EDITOR PUBLISH VIEW

main_p1_lqr.m main_p2_ilqc.m task_design.m

```
% add task
task_ilqc = task_design();
N = length(task_ilqc.start_time:task_ilqc.dt:task_ilqc.end_time);

% add model
const_vel = 1; % assume constant forward speed
model = generate_model(const_vel);

% save directory
save_dir = './results/';

% initialize controller
load(strcat(save_dir, 'lqr_controller'));
controller_ilqc = controller_lqr;

% flags
plot_on = true;
save_on = true;

%% [Problem 1.1 (j)] Iterative Linear Quadratic Controller
% ========================== [TODO] ILQC Design ==========================
% Design a 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()'.
%
%
% ========================================================================

%% Simulation
sim_out_ilqc = mobile_robot_sim(model, task_ilqc, controller_ilqc);
fprintf('\n\ntarget state [%.3f; %.3f]\n', task_ilqc.goal_x);
fprintf('reached state [%.3f; %.3f]\n', sim_out_ilqc.x(:,end));

%% Plots
if plot_on
    plot_results(sim_out_ilqc);
end
```

DYNAMIC SYSTEMS LAB

ILQC algorithm:

- Initial policy (LQR) - $\mathbf{\Theta}_{\text{lqr}}$
- Forward pass - $(\bar{\mathbf{x}}_0, \bar{\mathbf{u}}_0), (\bar{\mathbf{x}}_1, \bar{\mathbf{u}}_1), \ldots, (\bar{\mathbf{x}}_{N-1}, \bar{\mathbf{u}}_{N-1}), \bar{\mathbf{x}}_N$
- (Initializations)
- Backward pass

at each time step, approximate system dynamics and cost function about $(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$

- Linearizing system dynamics - $(\mathbf{A}_k, \mathbf{B}_k)$
- Quadratic approximation of costs - $(q_k, \mathbf{q}_k, \mathbf{Q}_k, \mathbf{r}_k, \mathbf{R}_k, \mathbf{P}_k)$
- Solve for optimal input at each time step and update policy - $\mathbf{u}_k = \bar{\mathbf{u}}_k + \delta\mathbf{u}_k^*$

repeat until convergence criteria met

Problem 1.1 Mobile Robot | (d)-(j) ILQC Design

DYNAMIC SYSTEMS LAB

- Complete main_p1_ilqc.m
- Run and observe
  - Differences between LQR and ILQC
- Note: Model used in controller design and simulation are identical, but two functions mobile_robot_sim.m and mobile_robot_sim_test.m in ./simulation can be used to explore model mismatch (see comments in code for details).

Problem 1.2 Quadrotor | Overview

- Goal: Control a quadrotor to reach a designated goal state and/or passing through a given via-point
- Four components (based on programming exercise from [1]):
  - LQR
  - LQR with via-point
  - ILQC
  - ILQC with via-point

Reference
[1] Jonas Buchli. *Optimal and Learning Control for Autonomous Robots*, ETH Zurich. Info: http://www.adrlab.org/doku.php/adrl:education:lecture:fs2015

Problem 1.2 Quadrotor | Model

- State: $\mathbf{x} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T \in \mathbb{R}^{12}$
- Input: $\mathbf{u} = [F_z, M_x, M_y, M_z]^T \in \mathbb{R}^4$

- System dynamics: $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}, \mathbf{u}) = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u}$
  - Nonlinear
  - Underactuated

AscTec Hummingbird [2]

References
[2] AscTec Hummingbird. Info: http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-hummingbird/#pane-0-0

Problem 1.2 Quadrotor | Code Structure (LQR)

DYNAMIC SYSTEMS LAB

main_p2_lqr.m

- Task_Design()
- Model
- Cost_Design()
- LQR_Design()
- Quad_Simulator()
- Visualize2()

EDITOR PUBLISH VIEW

main_p2_lqr.m | main_p2_ilqc.m | Task_Design.m | LQR_Design.m | ILQC_Design.m | Cost_Design.m

```
%% General
% add subdirectories
addpath(genpath(pwd));

% define task
Task = Task_Design();

% load the dynamic model of the quadcopter
load('Quadrotor_Model.mat', 'Model'); % save as structure "Model"

% define cost function
Task.cost = Cost_Design( Model.param.mQ, Task );

% save directory
save_dir = './Results/';

% flags
plot_on = true;
save_on = true;

%% Initial LQR Controller Design
% [Problem 1.2 (a)-(c)] Fill in the missing parts in ...
% LQR_Design(Model, Task)
[Initial_Controller, Cost_LQR] = LQR_Design(Model, Task);

%% Simulation
Sim_Out_LQR = Quad_Simulator(Model, Task, Initial_Controller);
disp('LQR controller performance:');
fprintf('Cost with LQR controller (metric: LQR cost function!): J* = %.3f \n', Cost_LQR);
fprintf('Start Quadcopter position: x = %.3f, y = %.3f, z = %.3f \n', Sim_Out_LQR.x(1:3,1));
fprintf('Final Quadcopter position: x = %.3f, y = %.3f, z = %.3f \n\n', Sim_Out_LQR.x(1:3,end));

%% Visualization of LQR controller
if plot_on
    Visualize2(Sim_Out_LQR, Model.param);
end
```

Problem 1.2 Quadrotor | (a) LQR Controller Gain

- LQR: Linearize system dynamics around one linearization point $(\mathbf{x}_{\text{op}}, \mathbf{u}_{\text{op}})$

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{g}(\mathbf{x}(t))\mathbf{u}(t)$$

linearize

$$\delta\dot{\mathbf{x}}(t) = \mathbf{A}_{\text{lin}}\delta\mathbf{x}(t) + \mathbf{B}_{\text{lin}}\delta\mathbf{u}(t)$$

Useful functions

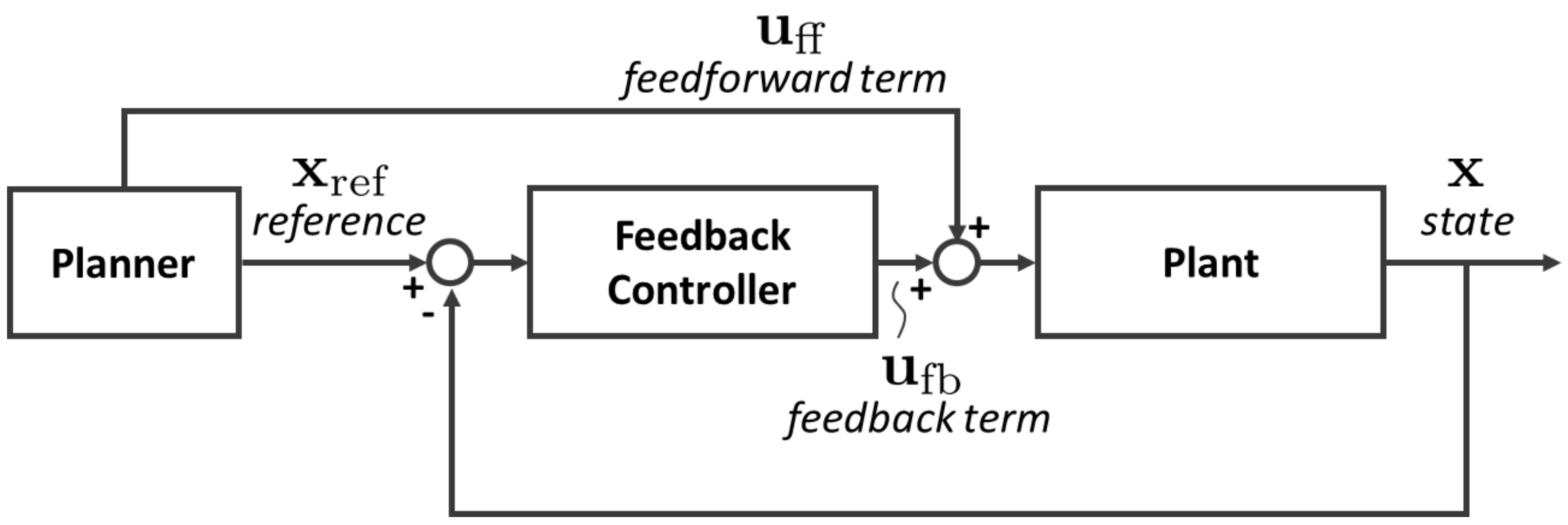
Model.Alin{1}(x_lin, u_lin, Model.param.syspar_vec)

Model.Blin{1}(x_lin, u_lin, Model.param.syspar_vec)

- Cost function (Q, R) matrices are specified in the struct Task.cost
  - Can be modified through Cost_Design()
- Matlab function lqr() for computing controller gain

Problem 1.2 Quadrotor | (b) LQR for Reaching Goal State

- Designing controller to reach goal state
- Controller has a feedforward component and a feedback component



Controller Architecture

$$
\begin{aligned}
\mathbf{u}_k &= \mathbf{u}_{\text{ff}} + \mathbf{u}_{k,\text{fb}} \\
&= \mathbf{u}_{\text{ff}} + \mathbf{K}(\mathbf{x}_k - \mathbf{x}_{\text{ref}}) \\
&= \begin{bmatrix} \mathbf{u}_{\text{ff}} - \mathbf{K}\mathbf{x}_{\text{ref}} & \mathbf{K} \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{x}_k \end{bmatrix} \\
&= \boldsymbol{\theta}_k^T \mathbf{x}_{k,\text{aug}}
\end{aligned}
$$

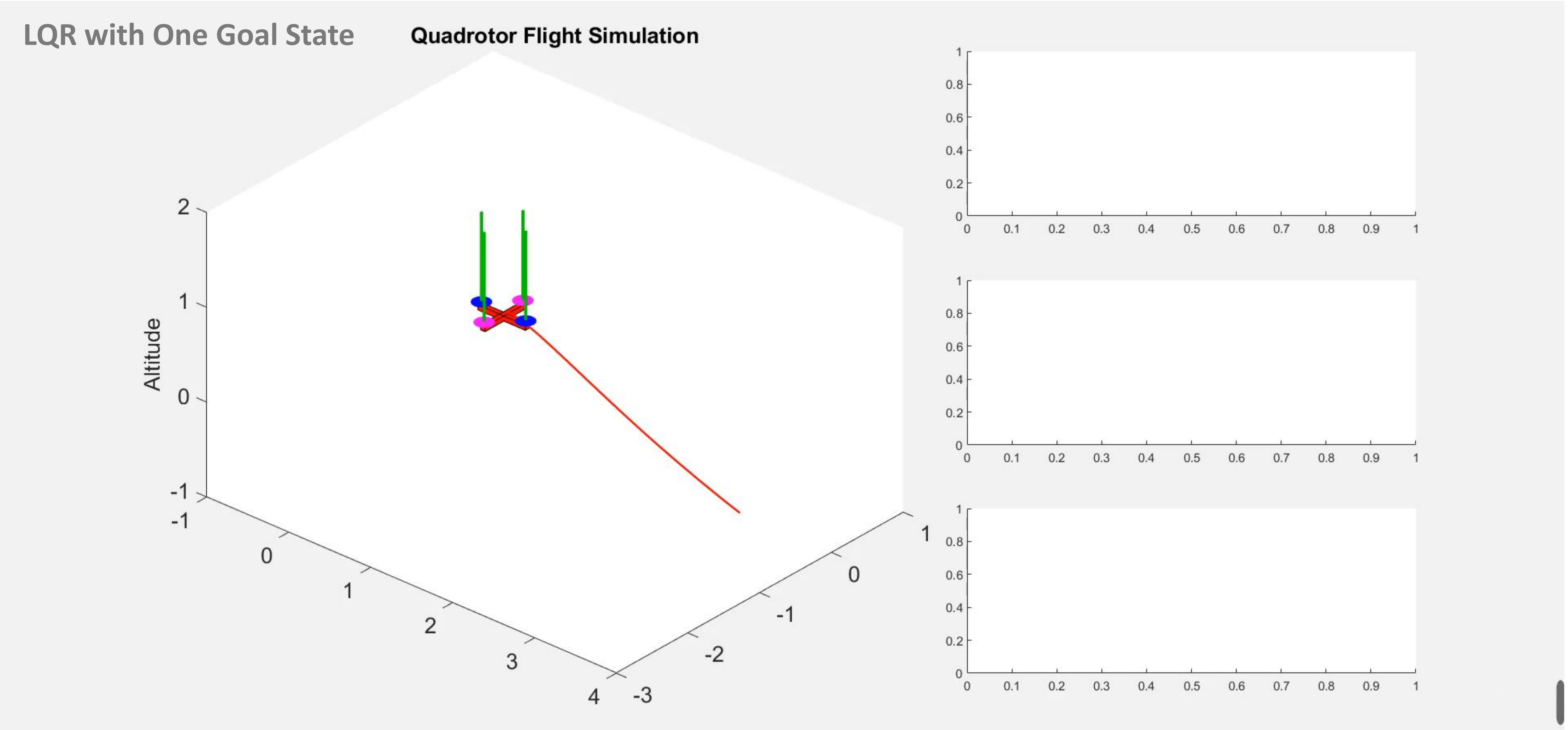
- Dimensions: $\boldsymbol{\theta}_k \in \mathbb{R}^{13\times 4}$ (time step k), $\underbrace{\boldsymbol{\Theta} \in \mathbb{R}^{13\times 4\times (N-1)}}_{\text{LQR\_Controller.theta}}$ (all time steps)

Problem 1.2 Quadrotor | (b) LQR for Reaching Goal State

- Complete corresponding parts in LQR_Design()
- Run and observe
  - Effect of changing (Q, R) matrices
  - Effect of changing goal locations
  - …

- General note:
  - Private function ./Private/Design_functions/LQR_Design_Solution.p is available for checking implementation

Problem 1.2 Quadrotor | (b) LQR for Reaching Goal State



Link to video: https://drive.google.com/open?id=1b2wJco_63T5li3juw85mi8wMTr6PB1JL

Problem 1.2 Quadrotor | (c) LQR with Via-Point

DYNAMIC SYSTEMS LAB

- Modify the controller such that the quadrotor passes through a prescribed via-point
  - Controller varies over time
- Complete LQR_Design()
- Run and observe
  - Performance as compared to LQR alone

Problem 1.2 Quadrotor | Code Structure (ILQC)

DYNAMIC SYSTEMS LAB

main_p2_ilqc.m

- Task_Design()
- Model
- Cost_Design()
- ILQC_Design()
- Quad_Simulator()
- Visualize2()

EDITOR PUBLISH VIEW

main_p2_lqr.m | main_p2_ilqc.m | Task_Design.m | LQR_Design.m | ILQC_Design.m | Cost_Design.m

```matlab

% define task
Task = Task_Design();

% load the dynamic model of the quadcopter
load('Quadrotor_Model.mat', 'Model'); % save as structure "Model"

% define cost function
% [Problem 1.2 (e)] Fill in the missing parts under 'via_point' in ...
% Cost_Design( Model.param.mQ, Task )
Task.cost = Cost_Design( Model.param.mQ, Task );

% save directory
save_dir = './Results/';

% flags
plot_on = true;
save_on = true;
load_lqr = false;
...
%% Problem 2: ILQC controller design
% [Problem 1.2 (d)] Fill in the missing parts in ...
% ILQC_Design(Model,Task,Initial_Controller,@Quad_Simulator)
[ILQC_Controller, Cost] = ILQC_Design(Model,Task,Initial_Controller,@Quad_Simulator);

%% Simulation
t_cpu = cputime;
Sim_Out_ILQC = Quad_Simulator(Model,Task,ILQC_Controller);
t_cpu = cputime - t_cpu;
fprintf('The ILQC algorithm found a solution in %fs \n\n',t_cpu);
fprintf('Final Quadcopter Position: xQ = %.3f, yQ = %.3f, zQ = %.3f \n', Sim_Out_ILQC.x(1:3,end));
fprintf('Final Quadcopter Velocity: xQ = %.3f, yQ = %.3f, zQ = %.3f \n', Sim_Out_ILQC.x(7:9,end));

%% Visualization of ILQC controller
if plot_on
    Visualize2(Sim_Out_ILQC, Model.param);
end
```

LQR Controller

Problem 1.2 Quadrotor | (d) ILQC Controller Design

- ILQC: Linearize system dynamics around each discretized (state, input)
- Approximation to convert a continuous-time linearized system to a discrete-time linearized system:

$$\delta\dot{\mathbf{x}}(t) = \mathbf{A}_{\text{lin}}\delta\mathbf{x}(t) + \mathbf{B}_{\text{lin}}\delta\mathbf{u}(t)$$

approx.

$$\frac{\delta\mathbf{x}_{k+1} - \delta\mathbf{x}_k}{\delta t} = \mathbf{A}_{\text{lin}}\delta\mathbf{x}_k + \mathbf{B}_{\text{lin}}\delta\mathbf{u}_k$$

$$\delta\mathbf{x}_{k+1} = \underbrace{\left(\mathbf{I} + \mathbf{A}_{\text{lin}}\delta t\right)}_{\mathbf{A}_k}\delta\mathbf{x}_k + \underbrace{\left(\mathbf{B}_{\text{lin}}\delta t\right)}_{\mathbf{B}_k}\delta\mathbf{u}_k$$

```matlab
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 = ...;
    % Blin = ...;
    % A = ...;
    % B = ...;
    % ==================================================================

    % ==================================================================
    % [Todo] quadratize cost function
    % [Note] use function {q_fun, Qv_fun, Qm_fun, Rv_fun, Rm_fun,
    % Pm_fun} provided above.
    %
    % t0 = T0(:,k);
    % q = ...;
    % Qv = ...;
    % Qm = ...;
    % Rv = ...;
    % Rm = ...;
    % Pm = ...;
    % ==================================================================
```

Problem 1.2 Quadrotor | (d) ILQC Controller Design

DYNAMIC SYSTEMS LAB

- ILQC: Linearize system dynamics around each discretized (state, input)
- Approximation to convert a continuous-time linearized system to a discrete-time linearized system:

$$\delta\dot{\mathbf{x}}(t) = \mathbf{A}_{\text{lin}}\delta\mathbf{x}(t) + \mathbf{B}_{\text{lin}}\delta\mathbf{u}(t)$$

$$\Big\downarrow \text{approx.}$$

$$\delta\mathbf{x}_{k+1} = \underbrace{(\mathbf{I} + \mathbf{A}_{\text{lin}}\delta t)}_{\mathbf{A}_k}\,\delta\mathbf{x}_k + \underbrace{(\mathbf{B}_{\text{lin}}\delta t)}_{\mathbf{B}_k}\,\delta\mathbf{u}_k$$

- Cost approximation: Jacobian and Hessian of the cost functions are provided in the script

```
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 (xO,uO). See exercise sheet Eqn. (18) for
    % details.
    %
    % Alin = ...;
    % Blin = ...;
    % A = ...;
    % B = ...;
    % =================================================================

    % =================================================================
    % [Todo] quadratize cost function
    % [Note] use function {q_fun, Qv_fun, Qm_fun, Rv_fun, Rm_fun,
    % Pm_fun} provided above.
    %
    % t0 = T0(:,k);
    % q = ...;
    % Qv = ...;
    % Qm = ...;
    % Rv = ...;
    % Rm = ...;
    % Pm = ...;
    % =================================================================
```

Problem 1.2 Quadrotor | (d) ILQC Controller Design

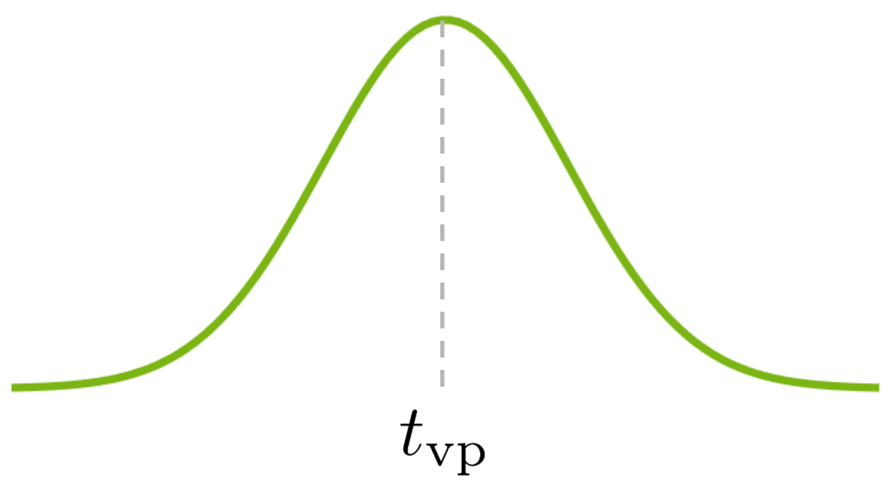
- Similar to Problem 1.1, complete ILQC_Design()
  - Private function ./Private/Design_functions/ILQC_Design_Solution.p is available for checking implementation
- Run and observe
  - Compare with LQR controller

Problem 1.2 Quadrotor | (e) ILQC with Via-Point

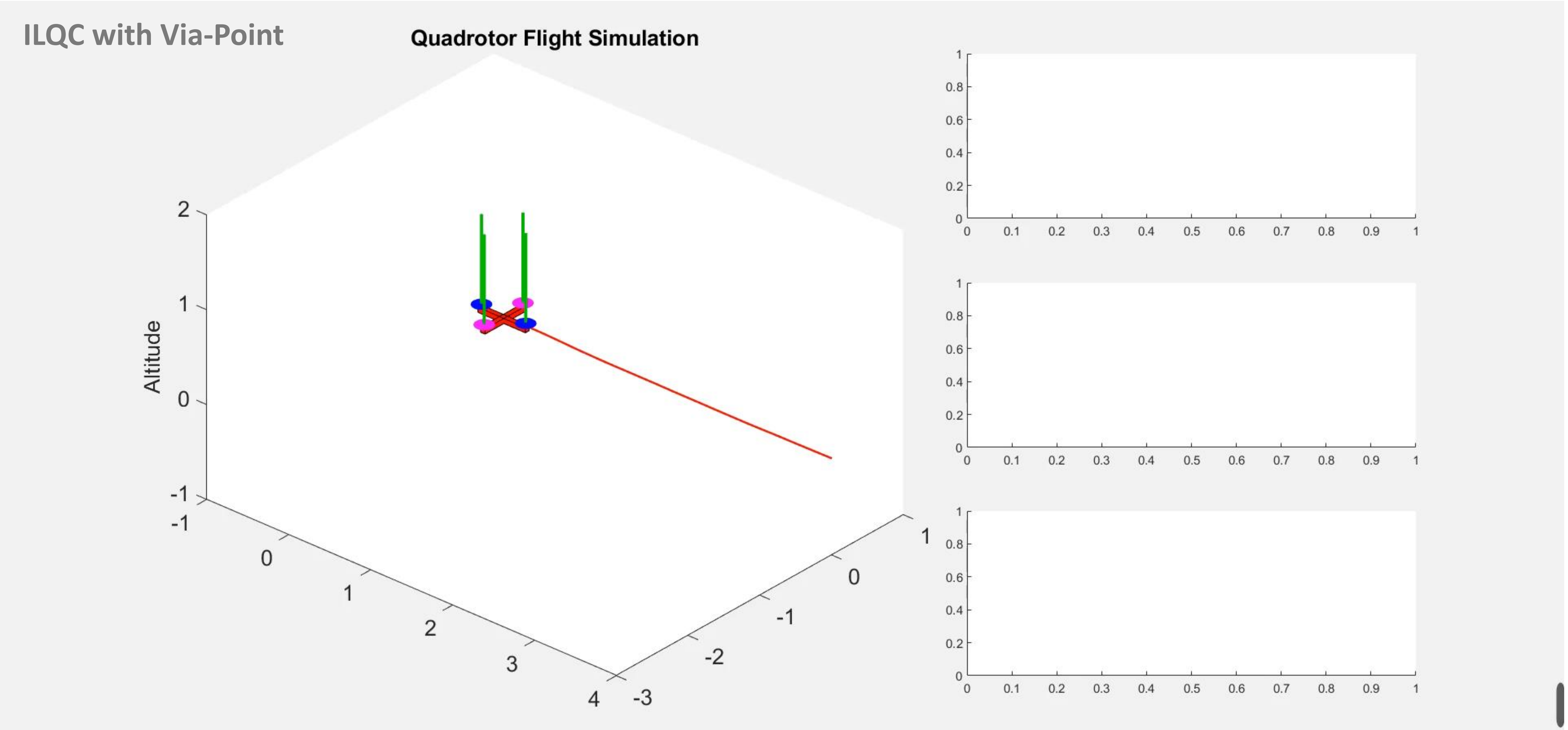
- Add to ILQC cost function a term that penalizes deviation from via-point $\mathbf{x}_{\mathrm{vp}}$ in proximity of time t_{vp}:

$$g_{\mathrm{vp}}(t) = (\mathbf{x} - \mathbf{x}_{\mathrm{vp}})^T \mathbf{Q}_{\mathrm{vp}} (\mathbf{x} - \mathbf{x}_{\mathrm{vp}}) \underbrace{\sqrt{\frac{\rho}{2\pi}} \exp\left(-\frac{\rho}{2}(t - t_{\mathrm{vp}})^2\right)}_{\textit{weighted by } \rho\textit{-steep bell curve}}$$

- Complete Cost_Design()
- Run and observe
  - Use different via-points
  - Compare with LQR



Problem 1.2 Quadrotor | (e) ILQC with Via-Point



Link to video: https://drive.google.com/open?id=1awkzzF6MyIcgP6hxMJwn7RcmvZobRFng