# Table of Contents

# Table of Figures

# Table of Equations

# Introduction

An Inverted Pendulum (IP) is a pendulum that has its center of mass above its pivot point. It is naturally unstable. The IP is a common engineering challenge due to its application in real life. Many systems behave similar to the IP, such as missile guidance [1], robotic arm [2], and Segway [3].

The IP is a highly non-linear, unstable, and under-actuated Multiple Input and Multiple Outputs (MIMO) mechanical system [4]. Many types of controllers have been proposed to control the IP, such as PID controller [5], [6], [7], PD controller [8], [9], LQR controller [10], [11], feedback linearization [11], Sliding Mode Control (SMC) and its variants [11], [12], [13], [14], [15], [16], backstepping control [14], intermittent feedback control [17], and fuzzy logic controller [18], [12], [19].

The problem considered in this paper is a Cart Inverted Pendulum (CIP), a type of pendulum that is mounted on a motorized cart and has its movement limited only to one axis. Two types of controllers tested on the CIP in this paper is Sliding Mode Control (SMC) and Quasi-Sliding Mode Control (QSMC). The objective of both controllers is to keep the pendulum upright at all time.

# Problem Formulation

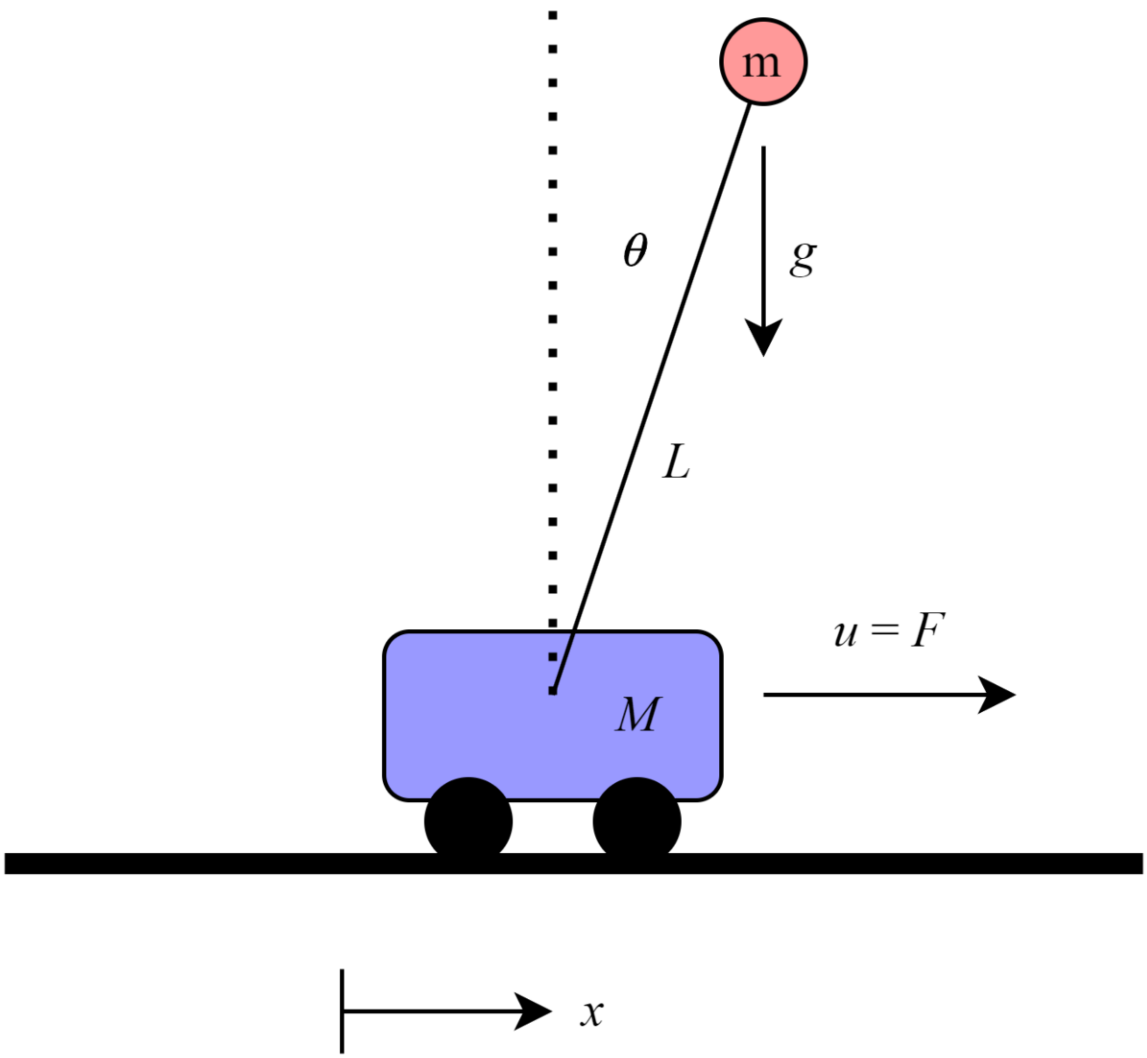


FIGURE 1: PHYSICAL MODEL OF CART INVERTED PENDULUM

TABLE 1: MEANING AND UNIT OF EACH SYSTEM STATES AND PARAMETERS

| Parameter | Meaning | Unit |
|---|---|---|
| $m$ | Mass of pendulum | kg |
| $M$ | Cart mass | kg |
| $L$ | Pendulum length | m |
| $g$ | Gravitational acceleration | m/s$^2$ |
| $x$ | Cart position | m |
| $v$ | Cart velocity | m/s |
| $\theta$ | Pendulum angle | rad |
| $\omega$ | Pendulum angular velocity | rad/s |
| $u$ | Control force | N |

TABLE 2: VALUE OF EACH SYSTEM PARAMETERS

| Parameter | Value | Unit |
|---|---|---|
| $m$ | 1 | kg |
| $M$ | 5 | kg |
| $L$ | 2 | m |
| $g$ | 9.8067 | m/s$^2$ |

Free Body Diagram (FBD) of the CIP can be seen on Figure 1, and the meaning and unit of each of its parameter can be seen on Table 1. Value of each of the parameter is shown on Table 2. The CIP consists of a moveable cart-rail system and a swing-able pole connected to the cart, as shown on Figure 1. Position of the cart is controlled by actuating the wheel.

To derive the full dynamics of the CIP in this paper, first let's consider the acting torque on the pendulum. The equation that represents the torque acting on the pendulum is shown as

$$F_x L\cos(\theta) - F_y L\sin(\theta) = mgL\sin(\theta) \tag{1}$$

where the force components in $x$-axis and $y$-axis, $F_x$ and $F_y$ are determined as

$$F_x = \quad m\left(\ddot{x} - L\sin(\theta)\,\dot{\theta}^2 + L\cos(\theta)\,\ddot{\theta}\right) \tag{2}$$

$$F_y = \quad -mL\left(\cos(\theta)\,\dot{\theta}^2 + \sin(\theta)\,\ddot{\theta}\right) \tag{3}$$

Substituting (2) and (3) into (1) will lead to

$$\ddot{x}\cos(\theta) + \ddot{\theta} = g\sin(\theta) \tag{4}$$

Force balance in the $x$-axis can be written as

$$(M+m)\ddot{x} - mL\sin(\theta)\,\dot{\theta}^2 + mL\cos(\theta)\,\ddot{\theta} = u \tag{5}$$

Taking the definition of $\ddot{x}$ from (4) and substituting it into (5) leads to

$$(M+m)(g\sin(\theta) - \theta) - mL\sin(\theta)\cos(\theta)\,\dot{\theta}^2 + mL\cos^2(\theta)\,\ddot{\theta} = u\cos(\theta) \tag{6}$$

which could be rewritten as

$$\left(mL\cos^2(\theta) - (M+m)\right)\ddot{\theta} = u\cos(\theta) - (M+m)g\sin(\theta) + mL\sin(\theta)\cos(\theta)\,\dot{\theta}^2 \tag{7}$$

Finally, dividing the lead coefficients of (7) leads to

$$\ddot{\theta} = \frac{-(M+m)g\sin(\theta) + mL\omega\sin(\theta)\cos(\theta) + u\cos(\theta)}{mL\cos^2(\theta) - (M+m)} \tag{8}$$

The selected state variables of the system are the angular position and the angular velocity, which is represented as $x_1 = \theta$ and $x_2 = \dot{\theta} = \omega$. The input variable of the system is the applied force $u$, and the output of the system is the angular position $y = x_1 = \theta$. The system's disturbance is represented as $d(x,t)$. The full nonlinear dynamics of the CIP is represented by

$$\dot{x}_1 = \quad \dot{\theta} = x_2 \tag{9}$$

$$\dot{x}_2 = \quad \frac{-(M+m)g\sin(x_1) + mLx_2\sin(x_1)\cos(x_1) + u\cos(x_1)}{mL\cos^2(x_1) - (M+m)} + d(x,t) \tag{10}$$

# Control Design

The objective of the control law is to make the pendulum stand upright.

## Sliding Mode Control

The first controller to be designed is a Sliding Mode Control (SMC). Let the sliding surface of the SMC $\sigma_{SMC}$ be defined as

$$\sigma_{SMC} = \dot{e} + c_{SMC}e \tag{11}$$

where $c_{SMC}$ is a positive constant and

$$e = x_{1,des} - x_1 \tag{12}$$

with $x_{1,des}$ representing the desired pendulum angle. Since the control objective is to always make the pendulum stand upright, this means

$$x_{1,des} = \dot{x}_{1,des} = \ddot{x}_{1,des} = 0 \tag{13}$$

and equation (12) can be simplified into

$$e = -x_1 \tag{14}$$

which leads to further simplification of (11) into

$$\sigma_{SMC} = -\dot{x}_1 - c_{SMC}x_1 \tag{15}$$

The derivative of $\sigma$ with respect to time is

$$\dot{\sigma}_{SMC} = -\ddot{x}_1 - c_{SMC}\dot{x}_1 \tag{16}$$

Further substitution of (10) into (16) leads to

$$\dot{\sigma}_{SMC} = -c_{SMC}x_2 + \frac{(M+m)g\sin(x_1) - mLx_2\sin(x_1)\cos(x_1) - u_{SMC}\cos(x_1)}{mL\cos^2(x_1) - (M+m)} - d(x,t) \tag{17}$$

Let the control signal be defined as

$$u_{SMC} = \begin{array}{l} (M+m)g\tan(x_1) - mLx_2\sin(x_1) + \\ \dfrac{(k_{SMC}\operatorname{sign}(\sigma_{SMC}) - c_{SMC}x_2)\big(mL\cos^2(x_1) - (M+m)\big)}{\cos(x_1)} \end{array} \tag{18}$$

where $k_{SMC}$ is a positive constant.

Lyapunov's direct method are used to check the stability of the system. The Lyapunov candidate function is defined as

$$V_{SMC} = \frac{{\sigma_{SMC}}^2}{2} \tag{19}$$

The derivative of (19) is shown as

$$\dot{V}_{SMC} = \dot{\sigma}_{SMC}\sigma_{SMC} \tag{20}$$

Substituting (16) into (20) leads to

$$\dot{V}_{SMC} = \sigma_{SMC}\left(-c_{SMC}x_2 + \frac{(M+m)g\sin(x_1) - mLx_2\sin(x_1)\cos(x_1) - u_{SMC}\cos(x_1)}{mL\cos^2(x_1) - (M+m)} - d(x,t)\right) \tag{21}$$

Further substitution of (18) into (21) leads to

$$\dot{V}_{SMC} = -\left(k_{SMC}\sigma_{SMC}\text{sign}(\sigma_{SMC}) + \sigma_{SMC}d(x,t)\right) \quad (22)$$

$$\dot{V}_{SMC} = -\left(k_{SMC}|\sigma_{SMC}| + \sigma_{SMC}d(x,t)\right) \quad (23)$$

For the system to be stable, the condition $\dot{V} < 0$ must be fulfilled. For $\dot{V} < 0$ to be fulfilled, let

$$-\left(k|\sigma| + \sigma d(x,t)\right) < 0 \quad (24)$$

then,

$$k > -\frac{\sigma}{|\sigma|}d(x,t) \quad (25)$$

The maximum value of the right-hand side of (25) is achieved when $d(x,t)$ is at its lowest. This means that inequality (25) can be simplified into

$$k > \frac{\sigma}{|\sigma|}|d_{min}(x,t)| \quad (26)$$

where $d_{min}(x,t)$ denotes the minimal value of $d(x,t)$.

# Quasi-Sliding Mode Control

The second controller to be designed is a Quasi-Sliding Mode Control (QSMC). The QSMC works similarly to conventional SMC, but the sign function is replaced with a smoothly continuous function. Let the sliding surface of the QSMC $\sigma_{QSMC}$ be defined as

$$\sigma_{QSMC} = \dot{e} + c_{QSMC}e \quad (27)$$

where $c_{QSMC}$ is a positive constant.

Substitution of (10) into (27) leads to

$$\dot{\sigma}_{QSMC} = -c_{QSMC}x_2 + \frac{(M+m)g\sin(x_1) - mLx_2\sin(x_1)\cos(x_1) - u\cos(x_1)}{mL\cos^2(x_1) - (M+m)} - d(x,t) \quad (28)$$

Let the control signal be defined as

$$u_{QSMC} = (M+m)g\tan(x_1) - mLx_2\sin(x_1) + \frac{\left(k_{QSMC}\text{sign}\left(\sigma_{QSMC}\right) - c_{QSMC}x_2\right)\left(mL\cos^2(x_1) - (M+m)\right)}{\cos(x_1)} \quad (29)$$

where $k_{QSMC}$ is a positive constant.

Lyapunov's direct method are used to check the stability of the system. The Lyapunov candidate function is defined as

$$V_{QSMC} = \frac{{\sigma_{QSMC}}^2}{2} \quad (30)$$

The derivative of (30) is shown as

$$\dot{V}_{QSMC} = \dot{\sigma}_{QSMC}\sigma_{QSMC} \quad (31)$$

Substituting (27) into (31) leads to

$$\dot{V}_{QSMC} = \sigma_{QSMC}\left(-c_{QSMC}x_2 + \frac{(M+m)g\sin(x_1) - mLx_2\sin(x_1)\cos(x_1) - u_{SMC}\cos(x_1)}{mL\cos^2(x_1) - (M+m)} - d(x,t)\right) \quad (32)$$

Further substitution of (29) into (32) leads to

$$\dot{V}_{QSMC} = -\left(k_{QSMC}\sigma_{QSMC}\alpha\left(\sigma_{QSMC}\right) + \sigma_{QSMC}d(x,t)\right) \tag{33}$$

where $f\left(\sigma_{QSMC}\right)$ is the smooth function used to replace the sign function. Here, $f\left(\sigma_{QSMC}\right)$ is defined as

$$\alpha\left(\sigma_{QSMC}\right) = \frac{\sigma_{QSMC}}{\left|\sigma_{QSMC}\right| + \delta_{QSMC}} \tag{34}$$

where $\delta_{QSMC}$ is a positive constant. Then, equation (33) can be redefined as

$$\dot{V}_{QSMC} = -\left(\frac{k_{QSMC}{\sigma_{QSMC}}^2}{\left|\sigma_{QSMC}\right| + \delta_{QSMC}} + \sigma_{QSMC}d(x,t)\right) \tag{35}$$

For the system to be stable, the condition $\dot{V} < 0$ must be fulfilled. For $\dot{V} < 0$ to be fulfilled, let

$$-\left(\frac{k_{QSMC}{\sigma_{QSMC}}^2}{\left|\sigma_{QSMC}\right| + \delta_{QSMC}} + \sigma_{QSMC}d(x,t)\right) < 0 \tag{36}$$

then,

$$k_{QSMC} > -\frac{\left(\left|\sigma_{QSMC}\right| + \delta\right)d(x,t)}{\sigma_{QSMC}} \tag{37}$$

The maximum value of the right-hand side of (37) is achieved when $d(x,t)$ is at its lowest. This means that inequality (37) can be simplified into

$$k_{QSMC} > \frac{\left(\left|\sigma_{QSMC}\right| + \delta\right)|d_{min}(x,t)|}{\sigma_{QSMC}} \tag{38}$$

where $d_{min}(x,t)$ denotes the minimal value of $d(x,t)$.

# Simulation Results and Discussion

For simulation, the simulation-specific parameters are shown on Table 3.

TABLE 3: MEANING AND VALUE OF EACH SIMULATION-SPECIFIC PARAMETERS

| Parameter | Meaning | Value | Unit |
|---|---|---|---|
| t_step | Simulation time step | 0.001 | s |
| t_end | Simulation end time | 6 | s |

There are three types of disturbance that is simulated: sinusoidal disturbance, random disturbance, and gaussian disturbance. The parameter for each of these disturbance are shown on Table 4.

TABLE 4: MEANING AND VALUE OF DISTURBANCE PARAMETERS

| Parameter | Meaning | Value | Unit |
|---|---|---|---|
| sine_dist_freq | Frequency of sinusoidal disturbance | 100 | rad/s |
| sine_dist_amp | Amplitude of sinusoidal disturbance | 10 | rad/s |
| rand_dist_max | Maximum value of random noise | 10 | rad/s |
| rand_dist_min | Minimum value of random noise | -10 | rad/s |
| gauss_dist_max | Maximum value of gaussian noise | 10 | rad/s |
| gauss_dist_min | Minimum value of gaussian noise | -10 | rad/s |

For comparison between SMC and QMSC, the parameters shown on are Table 5 used.

TABLE 5: MEANING AND VALUE OF CONTROLLER PARAMETER USED

| Parameter | Meaning | Value |
|---|---|---|
| k_smc | Constant representing $k_{SMC}$ | 100 |
| c_smc | Constant representing $c_{SMC}$ | 1 |
| k_qsmc | Constant representing $k_{QSMC}$ | 100 |
| c_qsmc | Constant representing $c_{QSMC}$ | 1 |
| delta_qsmc | Constant representing $\delta_{QSMC}$ | 0.001 |

Side-by-side performance comparison is done, with the parameters used in the simulation shown by Table 4 and Table 5. Side-by-side comparison of both SMC and QSMC are shown by Figure 3, Figure 2, Figure 5, and Figure 4.

Between SMC and QSMC, SMC reaches the desired trajectory faster in a system without disturbance. This is because of QSMC has more “relaxed” boundary layer than SMC. Although this more “relaxed” boundary layer results in slower reaching time, it makes the chattering magnitude to become smaller. Value of the angle is discontinuous on the transition from reaching phase to sliding phase for SMC, but it is continuous for QSMC. This discontinuity results from the use of sign function, a discontinuous function. Side-by-side comparison of SMC and QSMC shows that both controllers have similar angle tracking performance in a noisy environment. This is because the structure of both controllers is similar. Both SMC and QSMC controller can make the angle of the pendulum to converge towards zero, but it can be observed that deviation of angle velocity of the CIP in a noisy environment for the QSMC has less magnitude than SMC. This means that QSMC is more robust than SMC.

It is more recommended to use QSMC instead of SMC in a noisy environment, because it has chattering with smaller magnitude, both controllers have similar angle tracking performance in a noisy environment, and QSMC has less angle velocity deviation in a noisy environment.

To show the effect of $\delta_{QSMC}$ on QSMC, $\delta_{QSMC}$ is varied on three different values: 0.001, 0.0001, and 0.00001. The parameter used while varying $\delta_{QSMC}$ is shown by Table 4 and Table 5. From Figure 6, Figure 7, Figure 8, and Figure 9, it can be shown that lower values of $\delta_{QSMC}$ makes the chattering amplitude bigger. It can also be shown that lower values of $\delta_{QSMC}$ also makes angle more discontinuous while the system is transitioning from reaching phase to sliding phase. All of these are caused by lower values of $\delta_{QSMC}$ makes QSMC behave more like SMC.

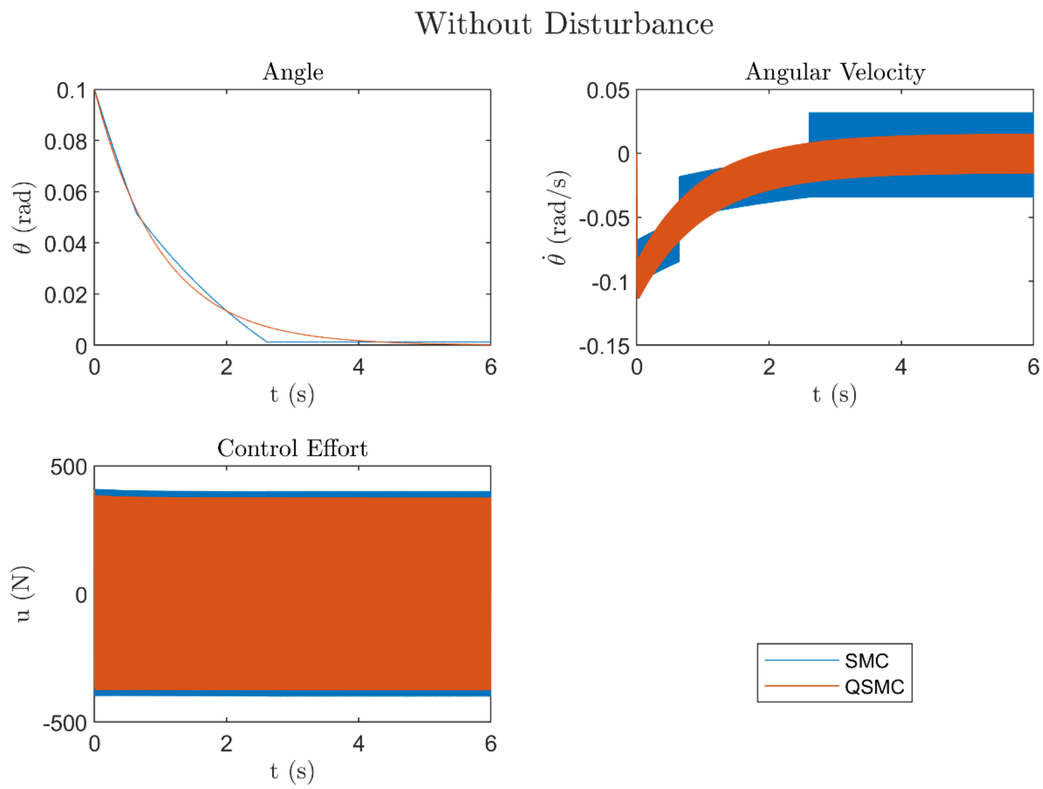


FIGURE 3: PERFORMANCE COMPARISON OF SMC AND QMSC WITHOUT DISTURBANCE

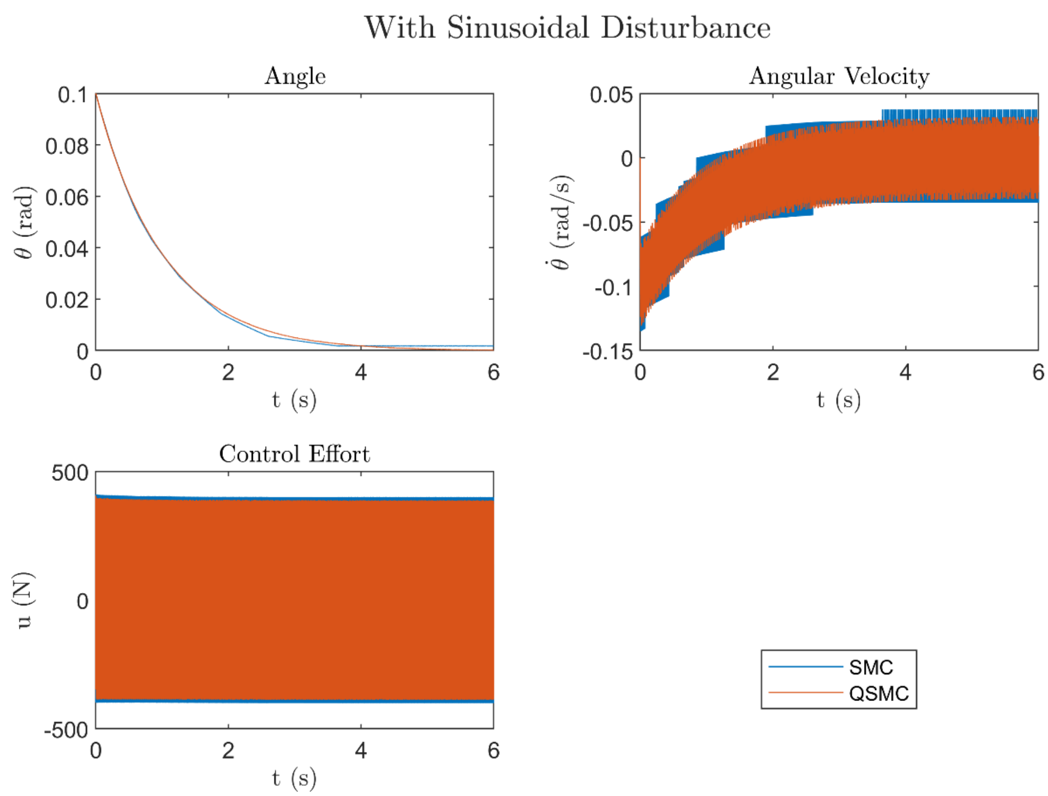


FIGURE 2: PERFORMANCE COMPARISON OF SMC AND QMSC WITH SINUSOIDAL DISTURBANCE

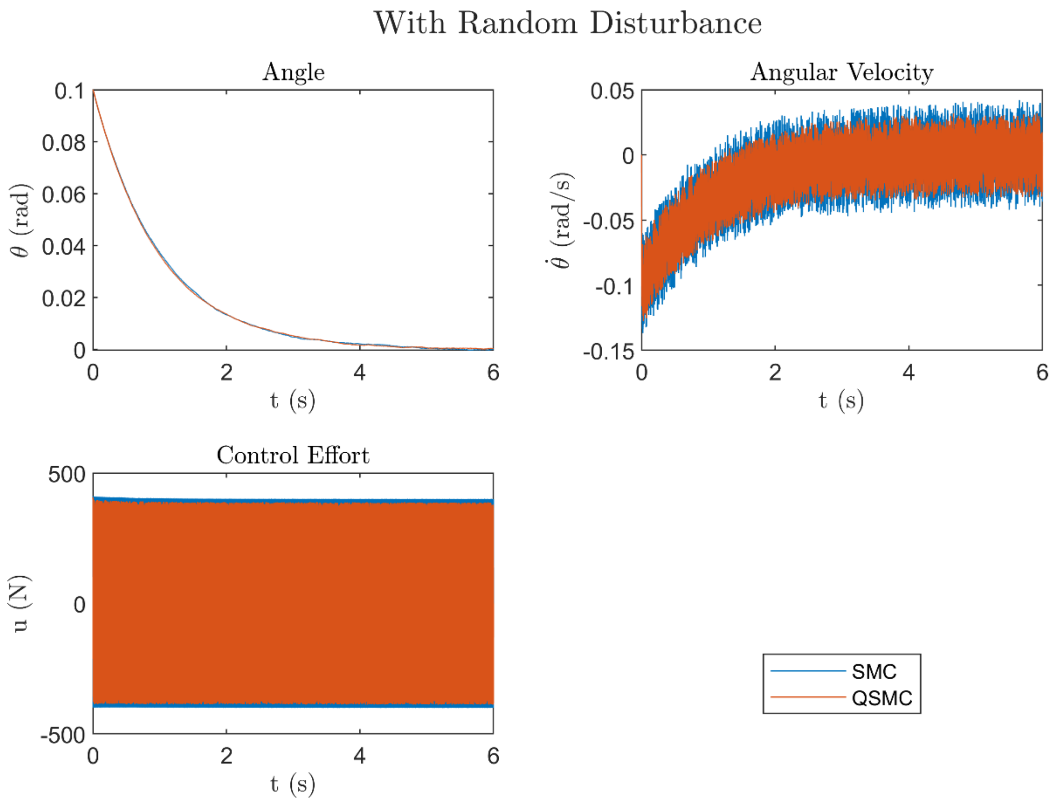


FIGURE 4: PERFORMANCE COMPARISON OF SMC AND QMSC WITH RANDOM DISTURBANCE

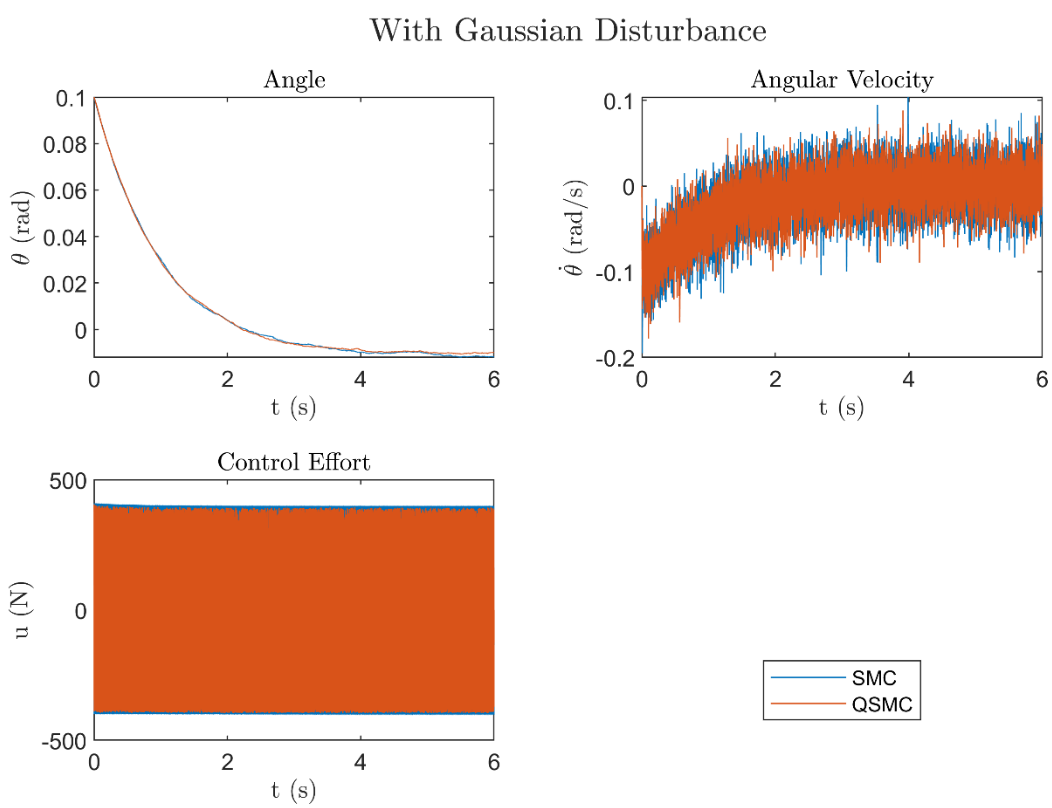


FIGURE 5: PERFORMANCE COMPARISON OF SMC AND QMSC WITH GAUSSIAN DISTURBANCE

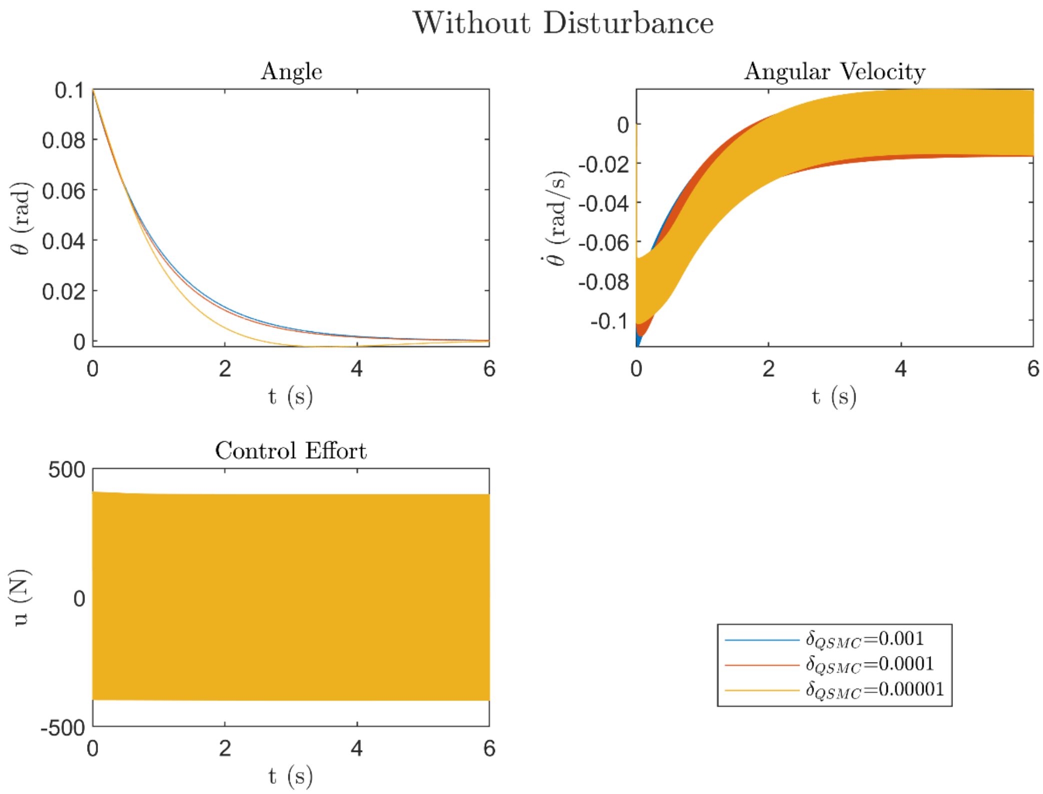


FIGURE 6: COMPARISON OF DIFFERENT $\delta_{QSMC}$ VALUES ON QSMC WITHOUT DISTURBANCE

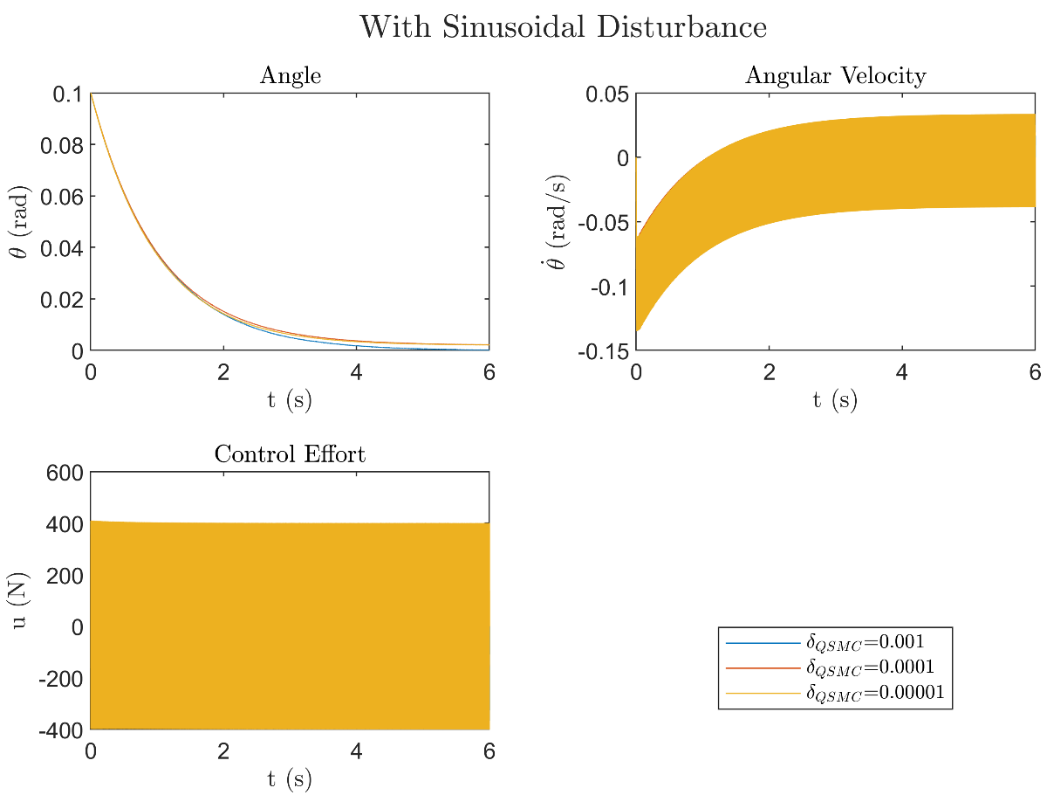


FIGURE 7: COMPARISON OF DIFFERENT $\delta_{QSMC}$ VALUES ON QSMC WITH SINUSOIDAL DISTURBANCE

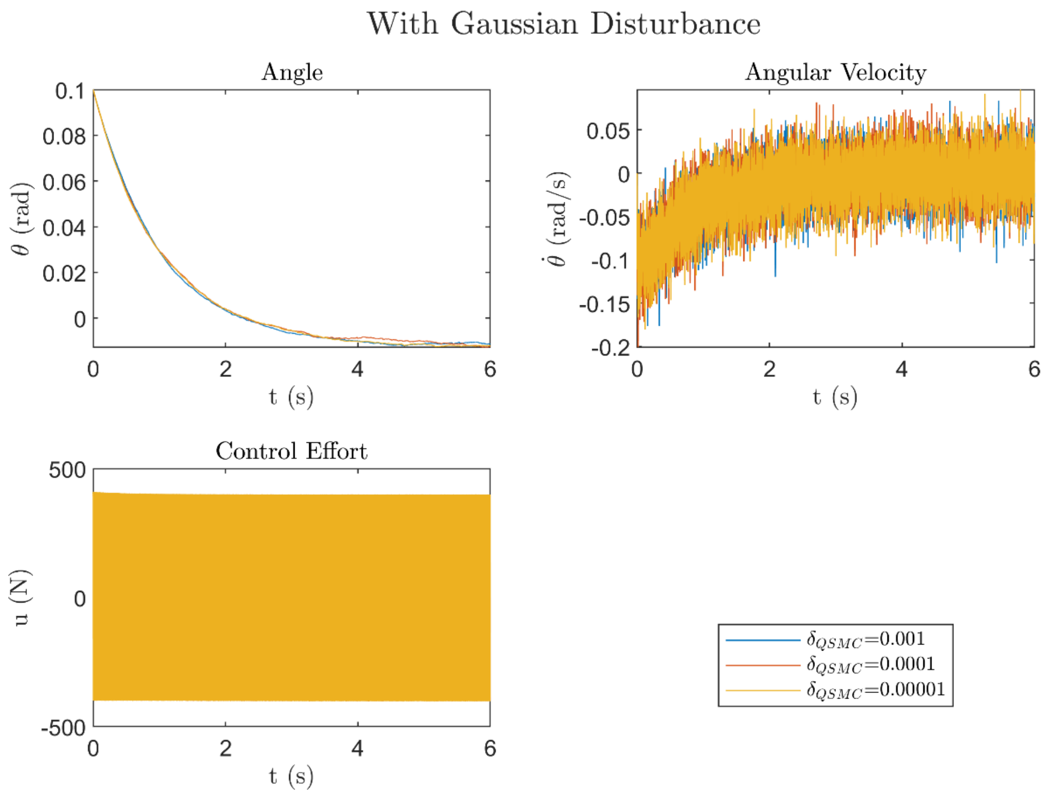


FIGURE 9: COMPARISON OF DIFFERENT $\delta_{QSMC}$ VALUES ON QSMC WITH GAUSSIAN DISTURBANCE

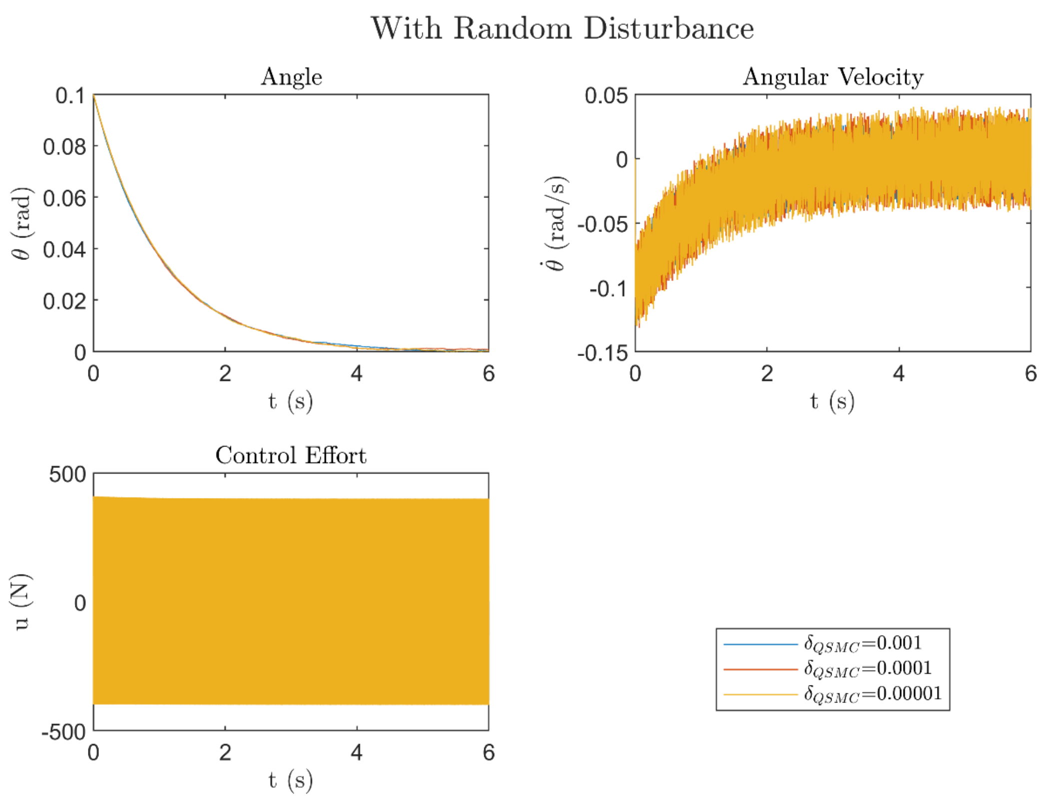


FIGURE 8: COMPARISON OF DIFFERENT $\delta_{QSMC}$ VALUES ON QSMC WITH RANDOM DISTURBANCE

# Conclusion

Both SMC and QSMC can be used to keep the pendulum of a CIP upright. The CIP that is controlled by both controller is proven to be able to be stable, using Lyapunov's direct method.

The amplitude of the chattering of the QSMC is less than the amplitude of the chattering of the SMC. Lower values of $\delta_{QSMC}$ on QSMC makes the chattering amplitude bigger, as it makes the QSMC approaches SMC.

QSMC is more recommended to be implemented over SMC in a noisy CIP system, as QSMC results in smaller chatter magnitude, both controllers have similar angle tracking performance in a noisy CIP system, and QSMC is more robust in a noisy CIP system. QSMC is also shown to be more robust against disturbances.

# References


[1] L. B. Prasad, B. Tyagi and H. O. Gupta, "Modelling and Simulation for Optimal Control of Nonlinear Inverted Pendulum Dynamical System Using PID Controller and LQR," in *2012 Sixth Asia Modelling Symposium*, Bali, 2012.

[2] N. Mellatshahi, S. Mozaffari, M. Saif and S. Alirezaee, "Inverted Pendulum Control with a Robotic Arm using Deep Reinforcement Learning," in *2021 International Symposium on Signals, Circuits and Systems (ISSCS)*, Iasi, 2021.

[3] W. Younis and M. Abdelati, "Design and Implementation of an Experimental Segway Model," in *AIP Conference Proceedings*, 2009.

[4] M. Bettayeb, C. Boussalem, R. Mansouri and M. U. Al-Saggaf, "Stabilization of an Inverted Pendulum-Cart System by Fractional PI-State Feedback," *ISA Transactions,* vol. 53, no. 2, pp. 508-516, 2014.

[5] M. Magdy, A. E. Marhomy and M. A. Attia, "Modeling of Inverted Pendulum System with Gravitational Search," *Ain Shams Engineering Journal,* vol. 10, pp. 129-149, 2019.

[6] M. Monir, "Analyzing and Designing Control System for an Inverted Pendulum on a Cart," *European Scientific Journal,* vol. 14, no. 6, 2018.

[7] H. Gao, X. Li, C. Gao and J. Wu, "Neural Network Supervision Control Strategy for Inverted Pendulum Tracking Control," *Discrete Dynamics in Nature and Society,* vol. 2021, 2021.

[8] F. Wei, G. Liangzhong, Y. Jin and B. Qingqing, "Inverted Pendulum Control System Based on GA Optimization," in *Workshop on Intelligent Information Technology Application (IITA 2007)*, 2007.

[9] V. Sukontanakam and M. Parnichkun, "Real-Time Optimal Control for Rotary Inverted Pendulum," *American Journal of Applied Sciences,* vol. 6, no. 6, pp. 1106-1115, 2009.

[10] J. Zhang, L. Zhang and J. Xie, "Application of Memetic Algorithm in Control of Linear Inverted Pendulum," in *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, 2011.

[11] S. Irfan, A. Mehmood, M. T. Razzaq and J. Iqbal, "Advanced Sliding Mode Control Techniques for Inverted Pendulum: Modelling and Simulation," *Engineering Science and Technology, an International Journal,* vol. 21, pp. 753-759, 2018.

[12] M. R. Dastranj, M. Moghaddas, Y. Ghezi and M. Rouhani, "Robust Control of Inverted Pendulum Using Fuzzy Sliding Mode Control and Genetic Algorithm," *International Journal of Information and Electronics Engineering,* vol. 2, no. 5, 2012.

[13] M. S. Mahmoud, R. A. A. Saleh and A. Ma'arif, "Stabilizing of Inverted Pendulum System Using Robust Sliding Mode Control," *International Journal of Robotics and Control Systems,* vol. 2, no. 2, pp. 230-239, 2022.

[14] A. Ma'arif, M. A. M. Vera, M. S. Mamoud, S. Ladaci, A. Cakan and J. N. Parada, "Backstepping Sliding Mode Control for Inverted Pendulum System with Disturbance and Parameter Uncertainty," *Journal of Robotics and Control (JRC),* vol. 3, no. 1, 2022.

[15] S. Babushanmugham, S. Srinivasan and E. Sivaraman, "Assessment of Optimisation Techniques for Sliding Mode Control of an Inverted Pendulum," *International Journal of Applied Engineering Research,* vol. 13, no. 14, pp. 11518-11524, 2018.

[16] A. K. Sharma and B. Bhushan, "Sliding Mode Control of Inverted Pendulum with Decoupling Algorithm," *International Journal of Computer Applications,* vol. 181, no. 27, 2018.

[17] P. Morasso, T. Nomura, Y. Suzuki and J. Zenzeri, "Stabilization of a Cart Inverted Pendulum: Improving the Intermittent Feedback Strategy to Match the Limits of Human Performance," *Frontiers in Computational Neuroscience,* vol. 13, no. 16, 2019.

[18] A. I. Roose, S. Yahya and H. Al-Rizzo, "Fuzzy-Logic Control of an Inverted Pendulum on a Cart," *Computers and Electrical Engineering,* vol. 61, pp. 31-47, 2017.

[19] I. Chawla, V. Chopra and A. Singla, "Robust LQR Based ANFIS Control of x-z Inverted Pendulum," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, 2019.

[20] S. A. Ajwad, A. Mehmood, M. I. Ullah and J. Iqbal, "Optimal v/s Robust Control: A Study and Comparison for Articulated Manipulator," *Journal of the Balkan Tribological Association ,* vol. 2, no. 3, pp. 2460-2466, 2016.

[21] V. I. Utkin and J. Shi, "Integral Sliding Mode in Systems Operating Under Uncertainty Conditions," in *Proceedings of 35th IEEE Conference on Decision and Control*, 1996.

# Appendix

Simulation code:

```
clc;
clearvars;

%% Flag
DEBUG = false;
SHOW_FIGURE = true;
SAVE_FIGURE = true;
SAVE_DATA_TO_MAT = true;

%% File names
SBS_WITHOUT_DIST_FILENAME = 'sbs_without_dist'; % filename for side-by-side without
disturbance
SBS_WITH_SINE_DIST_FILENAME = 'sbs_with_sine_dist'; % filename for side-by-side with
sinusoidal disturbance
SBS_WITH_RAND_DIST_FILENAME = 'sbs_with_rand_dist'; % filename for side-by-side with
random disturbance
SBS_WITH_GAUSS_DIST_FILENAME = 'sbs_with_gauss_dist'; % filename for side-by-side
with gauss disturbance

DELTA_WITHOUT_DIST_FILENAME = 'delta_without_dist'; % filename for delta without
disturbance
DELTA_WITH_SINE_DIST_FILENAME = 'delta_with_sine_dist'; % filename for delta with
sinusoidal disturbance
DELTA_WITH_RAND_DIST_FILENAME = 'delta_with_rand_dist'; % filename for delta with
random disturbance
DELTA_WITH_GAUSS_DIST_FILENAME = 'delta_with_gauss_dist'; % filename for delta with
gauss disturbance

IMG_RES = "500"; % image resolution

%% Simulation parameters
t_step = 0.001; % simulation time step size, s
t_end = 6; % simulation end time, s
t_span = 0: t_step: t_end;
x_0 = [0.1; % initial pendulum angle
       0]; % initial pendulum angular velocity

%% Physical system parameters
m = 1; % pendulum mass, kg
M = 5; % cart mass, kg
L = 2; % pendulum length, m
g = 9.8067; % gravitation acceleration constant, m/s^2

%% SIDE BY SIDE PERFORMANCE COMPARISON

%% ----------------------------------------------------------------------

%% SMC parameter
k_smc = 100;
c_smc = 1;
```

```
%% QSMC parameter
k_qsmc = 100;
c_qsmc = 1;
delta_qsmc = 0.001;

%% Sinusoidal disturbance parameter
sine_dist_freq = 100; % frequency of sinusoidal disturbance, Hz
sine_dist_amp = 10; % amplitude of sinusoidal disturbance, rad/s

%% Random disturbance parameter
rand_dist_max = 10; % max value of random disturbance, rad/s
rand_dist_min = -10; % min value of random disturbance, rad/s

%% Gaussian disturbance parameter
gauss_dist_max = 10; % max value of random disturbance, rad/s
gauss_dist_min = -10; % min value of random disturbance, rad/s

%% Initialize simulation data
% SMC
sbs_x_smc_clean = zeros(2, 1+t_end/t_step);
sbs_x_smc_clean(:, 1) = x_0;
sbs_u_smc_clean = zeros(1, 1+t_end/t_step);

sbs_x_smc_dirty_sine = zeros(2, 1+t_end/t_step);
sbs_x_smc_dirty_sine(:, 1) = x_0;
sbs_u_smc_dirty_sine = zeros(1, 1+t_end/t_step);

sbs_x_smc_dirty_rand = zeros(2, 1+t_end/t_step);
sbs_x_smc_dirty_rand(:, 1) = x_0;
sbs_u_smc_dirty_rand = zeros(1, 1+t_end/t_step);

sbs_x_smc_dirty_gauss = zeros(2, 1+t_end/t_step);
sbs_x_smc_dirty_gauss(:, 1) = x_0;
sbs_u_smc_dirty_gauss = zeros(1, 1+t_end/t_step);

% Quasi SMC
sbs_x_qsmc_clean = zeros(2, 1+t_end/t_step);
sbs_x_qsmc_clean(:, 1) = x_0;
sbs_u_qsmc_clean = zeros(1, 1+t_end/t_step);

sbs_x_qsmc_dirty_sine = zeros(2, 1+t_end/t_step);
sbs_x_qsmc_dirty_sine(:, 1) = x_0;
sbs_u_qsmc_dirty_sine = zeros(1, 1+t_end/t_step);

sbs_x_qsmc_dirty_rand = zeros(2, 1+t_end/t_step);
sbs_x_qsmc_dirty_rand(:, 1) = x_0;
sbs_u_qsmc_dirty_rand = zeros(1, 1+t_end/t_step);

sbs_x_qsmc_dirty_gauss = zeros(2, 1+t_end/t_step);
sbs_x_qsmc_dirty_gauss(:, 1) = x_0;
sbs_u_qsmc_dirty_gauss = zeros(1, 1+t_end/t_step);

%% Parse plotting flags
if SHOW_FIGURE == true
```

```
    set(0, 'DefaultFigureVisible', 'on');
else
    set(0, 'DefaultFigureVisible', 'off');
end

%% SIMULATION

%% SMC without disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = SMCCtrlEff(sbs_x_smc_clean(:, count), m, M, L, ...
                   k_smc, c_smc, g); % calculate control effort
    sbs_u_smc_clean(:, count) = u;

    d = 0; % no disturbance

    dx = pendCart(sbs_x_smc_clean(:, count), m, M, L, g, d, u); % calculate change of
states

    sbs_x_smc_clean(:, count+1) = sbs_x_smc_clean(:, count) + dx*t_step; % state
update
end

%% QSMC without disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(sbs_x_qsmc_clean(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc); % calculate control effort
    sbs_u_qsmc_clean(:, count) = u;

    d = 0; % no disturbance

    dx = pendCart(sbs_x_qsmc_clean(:, count), m, M, L, g, d, u); % calculate change
of states

    sbs_x_qsmc_clean(:, count+1) = sbs_x_qsmc_clean(:, count) + dx*t_step; % state
update
end

%% SMC with sinusoidal disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = SMCCtrlEff(sbs_x_smc_dirty_sine(:, count), m, M, L, ...
                   k_smc, c_smc, g); % calculate control effort
    sbs_u_smc_dirty_sine(:, count) = u;

    d = sineDist(t, sine_dist_freq, sine_dist_amp); % calculate disturbance value

    dx = pendCart(sbs_x_smc_dirty_sine(:, count), m, M, L, g, d, u); % calculate
change of states
```

```
    sbs_x_smc_dirty_sine(:, count+1) = sbs_x_smc_dirty_sine(:, count) + dx*t_step; %
state update
end

%% QSMC with sinusoidal disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(sbs_x_qsmc_dirty_sine(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc); % calculate control effort
    sbs_u_qsmc_dirty_sine(:, count) = u;

    d = sineDist(t, sine_dist_freq, sine_dist_amp); % calculate disturbance value

    dx = pendCart(sbs_x_qsmc_dirty_sine(:, count), m, M, L, g, d, u); % calculate
change of states

    sbs_x_qsmc_dirty_sine(:, count+1) = sbs_x_qsmc_dirty_sine(:, count) +
dx*t_step; % state update
end

%% SMC with random disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = SMCCtrlEff(sbs_x_smc_dirty_rand(:, count), m, M, L, ...
                   k_smc, c_smc, g); % calculate control effort
    sbs_u_smc_dirty_rand(:, count) = u;

    d = randDist(rand_dist_min, rand_dist_max); % calculate disturbance value

    dx = pendCart(sbs_x_smc_dirty_rand(:, count), m, M, L, g, d, u); % calculate
change of states

    sbs_x_smc_dirty_rand(:, count+1) = sbs_x_smc_dirty_rand(:, count) + dx*t_step; %
state update
end

%% QSMC with random disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(sbs_x_qsmc_dirty_rand(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc); % calculate control effort
    sbs_u_qsmc_dirty_rand(:, count) = u;

    d = randDist(rand_dist_min, rand_dist_max); % calculate disturbance value

    dx = pendCart(sbs_x_qsmc_dirty_rand(:, count), m, M, L, g, d, u); % calculate
change of states

    sbs_x_qsmc_dirty_rand(:, count+1) = sbs_x_qsmc_dirty_rand(:, count) +
dx*t_step; % state update
end
```

```
%% SMC with gaussian disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = SMCCtrlEff(sbs_x_smc_dirty_gauss(:, count), m, M, L, ...
                   k_smc, c_smc, g); % calculate control effort
    sbs_u_smc_dirty_gauss(:, count) = u;

    d = gaussDist(gauss_dist_min, gauss_dist_max); % calculate disturbance value

    dx = pendCart(sbs_x_smc_dirty_gauss(:, count), m, M, L, g, d, u); % calculate
change of states

    sbs_x_smc_dirty_gauss(:, count+1) = sbs_x_smc_dirty_gauss(:, count) +
dx*t_step; % state update
end

%% QSMC with gaussian disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(sbs_x_qsmc_dirty_gauss(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc); % calculate control effort
    sbs_u_qsmc_dirty_gauss(:, count) = u;

    d = gaussDist(gauss_dist_min, gauss_dist_max); % calculate disturbance value

    dx = pendCart(sbs_x_qsmc_dirty_gauss(:, count), m, M, L, g, d, u); % calculate
change of states

    sbs_x_qsmc_dirty_gauss(:, count+1) = sbs_x_qsmc_dirty_gauss(:, count) +
dx*t_step; % state update
end

%% DATA PLOTTING

%% Plot and save figure data for system without disturbance
sbs_fig_clean = tiledlayout(2, 2, "Visible", "on");
title(sbs_fig_clean, "Without Disturbance", 'Interpreter', 'latex');

% Plot theta data
nexttile();
plot(t_span, sbs_x_smc_clean(1, :), ...
     t_span, sbs_x_qsmc_clean(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, sbs_x_smc_clean(2, :), ...
     t_span, sbs_x_qsmc_clean(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');
```

```matlab
% Plot control effort data
nexttile();
plot(t_span, sbs_u_smc_clean, ...
     t_span, sbs_u_qsmc_clean);
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'SMC', 'QSMC'}, 'Location', 'south');
axis off;

% Remove spaces between subplots
sbs_fig_clean.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(sbs_fig_clean, append(SBS_WITHOUT_DIST_FILENAME, '.fig'));
    exportgraphics(sbs_fig_clean, ...
                   append(SBS_WITHOUT_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(SBS_WITHOUT_DIST_FILENAME, '.mat'), ...
         'sbs_x_smc_clean', 'sbs_x_qsmc_clean', ...
         'sbs_u_smc_clean', 'sbs_u_qsmc_clean', ...
         't_span');
end

%% Plot and save figure data for system with sinusoidal disturbance
sbs_fig_dirty_sine = tiledlayout(2, 2, "Visible", "on");
title(sbs_fig_dirty_sine, "With Sinusoidal Disturbance", 'Interpreter', 'latex');

% Plot theta
nexttile();
plot(t_span, sbs_x_smc_dirty_sine(1, :), ...
     t_span, sbs_x_qsmc_dirty_sine(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, sbs_x_smc_dirty_sine(2, :), ...
     t_span, sbs_x_qsmc_dirty_sine(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');
```

```
% Plot control effort data
nexttile();
plot(t_span, sbs_u_smc_dirty_sine, ...
     t_span, sbs_u_qsmc_dirty_sine);
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'SMC', 'QSMC'}, 'Location', 'south');
axis off;

% Remove spaces between subplots
sbs_fig_dirty_sine.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(sbs_fig_dirty_sine, append(SBS_WITH_SINE_DIST_FILENAME, '.fig'));
    exportgraphics(sbs_fig_dirty_sine, ...
                   append(SBS_WITH_SINE_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(SBS_WITH_SINE_DIST_FILENAME, '.mat'), ...
         'sbs_x_smc_dirty_sine', 'sbs_x_qsmc_dirty_sine', ...
         'sbs_u_smc_dirty_sine', 'sbs_u_qsmc_dirty_sine', ...
         't_span');
end

%% Plot and save figure data for system with random disturbance
sbs_fig_dirty_rand = tiledlayout(2, 2, "Visible", "on");
title(sbs_fig_dirty_rand, "With Random Disturbance", 'Interpreter', 'latex');

% Plot theta data
nexttile();
plot(t_span, sbs_x_smc_dirty_rand(1, :), ...
     t_span, sbs_x_qsmc_dirty_rand(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, sbs_x_smc_dirty_rand(2, :), ...
     t_span, sbs_x_qsmc_dirty_rand(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');

% Plot control effort data
```

```
nexttile();
plot(t_span, sbs_u_smc_dirty_rand, ...
     t_span, sbs_u_qsmc_dirty_rand);
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'SMC', 'QSMC'}, 'Location', 'south');
axis off;

% Remove spaces between subplots
sbs_fig_dirty_rand.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(sbs_fig_dirty_rand, append(SBS_WITH_RAND_DIST_FILENAME, '.fig'));
    exportgraphics(sbs_fig_dirty_rand, ...
                   append(SBS_WITH_RAND_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(SBS_WITH_RAND_DIST_FILENAME, '.mat'), ...
         'sbs_x_smc_dirty_rand', 'sbs_x_qsmc_dirty_rand', ...
         'sbs_u_smc_dirty_rand', 'sbs_u_qsmc_dirty_rand', ...
         't_span');
end

%% Plot and save figure data for system with gaussian disturbance
sbs_fig_dirty_gauss = tiledlayout(2, 2, "Visible", "on");
title(sbs_fig_dirty_gauss, "With Gaussian Disturbance", 'Interpreter', 'latex');

% Plot theta data
nexttile();
plot(t_span, sbs_x_smc_dirty_gauss(1, :), ...
     t_span, sbs_x_qsmc_dirty_gauss(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, sbs_x_smc_dirty_gauss(2, :), ...
     t_span, sbs_x_qsmc_dirty_gauss(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');

% Plot control effort data
nexttile();
```

```
plot(t_span, sbs_u_smc_dirty_gauss, ...
     t_span, sbs_u_qsmc_dirty_gauss);
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'SMC', 'QSMC'}, 'Location', 'south');
axis off;

% Remove spaces between subplots
sbs_fig_dirty_gauss.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(sbs_fig_dirty_gauss, append(SBS_WITH_GAUSS_DIST_FILENAME, '.fig'));
    exportgraphics(sbs_fig_dirty_gauss, ...
                   append(SBS_WITH_GAUSS_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(SBS_WITH_GAUSS_DIST_FILENAME, '.mat'), ...
         'sbs_x_smc_dirty_gauss', 'sbs_x_qsmc_dirty_gauss', ...
         'sbs_u_smc_dirty_gauss', 'sbs_u_qsmc_dirty_gauss', ...
         't_span');
end

%% -----------------------------------------------------------------------

%% VARYING THE VALUE OF DELTA ON QSMC

%% -----------------------------------------------------------------------

%% QSMC parameter
delta_qsmc_1 = 0.001;
delta_qsmc_2 = 0.0001;
delta_qsmc_3 = 0.00001;

%% Initialize simulation data
% QSMC with delta_qsmc = 0.001
delta_x_qsmc_clean_1 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_clean_1(:, 1) = x_0;
delta_u_qsmc_clean_1 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_sine_1 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_sine_1(:, 1) = x_0;
delta_u_qsmc_dirty_sine_1 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_rand_1 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_rand_1(:, 1) = x_0;
```

```
delta_u_qsmc_dirty_rand_1 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_gauss_1 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_gauss_1(:, 1) = x_0;
delta_u_qsmc_dirty_gauss_1 = zeros(1, 1+t_end/t_step);

% QSMC with delta_qsmc = 0.0001
delta_x_qsmc_clean_2 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_clean_2(:, 1) = x_0;
delta_u_qsmc_clean_2 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_sine_2 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_sine_2(:, 1) = x_0;
delta_u_qsmc_dirty_sine_2 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_rand_2 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_rand_2(:, 1) = x_0;
delta_u_qsmc_dirty_rand_2 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_gauss_2 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_gauss_2(:, 1) = x_0;
delta_u_qsmc_dirty_gauss_2 = zeros(1, 1+t_end/t_step);

% QSMC with delta_qsmc = 0.00001
delta_x_qsmc_clean_3 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_clean_3(:, 1) = x_0;
delta_u_qsmc_clean_3 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_sine_3 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_sine_3(:, 1) = x_0;
delta_u_qsmc_dirty_sine_3 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_rand_3 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_rand_3(:, 1) = x_0;
delta_u_qsmc_dirty_rand_3 = zeros(1, 1+t_end/t_step);

delta_x_qsmc_dirty_gauss_3 = zeros(2, 1+t_end/t_step);
delta_x_qsmc_dirty_gauss_3(:, 1) = x_0;
delta_u_qsmc_dirty_gauss_3 = zeros(1, 1+t_end/t_step);

%% SIMULATION

%% QSMC with delta_qsmc = 0.001 and without disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_clean_1(:, count), m, M, L, ...
                k_qsmc, c_qsmc, g, delta_qsmc_1); % calculate control effort
    delta_u_qsmc_clean_1(:, count) = u;

    d = 0; % no disturbance

    dx = pendCart(delta_x_qsmc_clean_1(:, count), m, M, L, g, d, u); % calculate
change of states
```

```
    delta_x_qsmc_clean_1(:, count+1) = delta_x_qsmc_clean_1(:, count) + dx*t_step; %
state update
end

%% QSMC with delta_qsmc = 0.001 and sinusoidal disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_sine_1(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_1); % calculate control effort
    delta_u_qsmc_dirty_sine_1(:, count) = u;

    d = sineDist(t, sine_dist_freq, sine_dist_amp); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_sine_1(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_dirty_sine_1(:, count+1) = delta_x_qsmc_dirty_sine_1(:, count) +
dx*t_step; % state update
end

%% QSMC with delta_qsmc = 0.001 and random disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_rand_1(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_1); % calculate control effort
    delta_u_qsmc_dirty_rand_1(:, count) = u;

    d = randDist(rand_dist_min, rand_dist_max); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_rand_1(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_dirty_rand_1(:, count+1) = delta_x_qsmc_dirty_rand_1(:, count) +
dx*t_step; % state update
end

%% QSMC with delta_qsmc = 0.001 and gaussian disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_gauss_1(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_1); % calculate control effort
    delta_u_qsmc_dirty_gauss_1(:, count) = u;

    d = gaussDist(gauss_dist_min, gauss_dist_max); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_gauss_1(:, count), m, M, L, g, d, u); %
calculate change of states

    delta_x_qsmc_dirty_gauss_1(:, count+1) = delta_x_qsmc_dirty_gauss_1(:, count) +
dx*t_step; % state update
end
```

```
%% QSMC with delta_qsmc = 0.0001 and without disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_clean_2(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_2); % calculate control effort
    delta_u_qsmc_clean_2(:, count) = u;

    d = 0; % no disturbance

    dx = pendCart(delta_x_qsmc_clean_2(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_clean_2(:, count+1) = delta_x_qsmc_clean_2(:, count) + dx*t_step; %
state update
end

%% QSMC with delta_qsmc = 0.0001 and sinusoidal disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_sine_2(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_2); % calculate control effort
    delta_u_qsmc_dirty_sine_2(:, count) = u;

    d = sineDist(t, sine_dist_freq, sine_dist_amp); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_sine_2(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_dirty_sine_2(:, count+1) = delta_x_qsmc_dirty_sine_2(:, count) +
dx*t_step; % state update
end

%% QSMC with delta_qsmc = 0.0001 and random disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_rand_2(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_2); % calculate control effort
    delta_u_qsmc_dirty_rand_2(:, count) = u;

    d = randDist(rand_dist_min, rand_dist_max); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_rand_2(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_dirty_rand_2(:, count+1) = delta_x_qsmc_dirty_rand_2(:, count) +
dx*t_step; % state update
end

%% QSMC with delta_qsmc = 0.0001 and gaussian disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier
```

```
    u = QSMCCtrlEff(delta_x_qsmc_dirty_gauss_2(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_2); % calculate control effort
    delta_u_qsmc_dirty_gauss_2(:, count) = u;

    d = gaussDist(gauss_dist_min, gauss_dist_max); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_gauss_2(:, count), m, M, L, g, d, u); %
calculate change of states

    delta_x_qsmc_dirty_gauss_2(:, count+1) = delta_x_qsmc_dirty_gauss_2(:, count) +
dx*t_step; % state update
end

%% QSMC with delta_qsmc = 0.00001 and without disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_clean_3(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_3); % calculate control effort
    delta_u_qsmc_clean_3(:, count) = u;

    d = 0; % no disturbance

    dx = pendCart(delta_x_qsmc_clean_3(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_clean_3(:, count+1) = delta_x_qsmc_clean_3(:, count) + dx*t_step; %
state update
end

%% QSMC with delta_qsmc = 0.00001 and sinusoidal disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_sine_3(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_3); % calculate control effort
    delta_u_qsmc_dirty_sine_3(:, count) = u;

    d = sineDist(t, sine_dist_freq, sine_dist_amp); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_sine_3(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_dirty_sine_3(:, count+1) = delta_x_qsmc_dirty_sine_3(:, count) +
dx*t_step; % state update
end

%% QSMC with delta_qsmc = 0.00001 and random disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_rand_3(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_3); % calculate control effort
    delta_u_qsmc_dirty_rand_3(:, count) = u;
```

```
    d = randDist(rand_dist_min, rand_dist_max); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_rand_3(:, count), m, M, L, g, d, u); % calculate
change of states

    delta_x_qsmc_dirty_rand_3(:, count+1) = delta_x_qsmc_dirty_rand_3(:, count) +
dx*t_step; % state update
end

%% QSMC with delta_qsmc = 0.00001 and gaussian disturbance
for t = 0: t_step: t_end-t_step
    count = int64(1 + t/t_step); % to mak_smce array indexing easier

    u = QSMCCtrlEff(delta_x_qsmc_dirty_gauss_3(:, count), m, M, L, ...
                    k_qsmc, c_qsmc, g, delta_qsmc_3); % calculate control effort
    delta_u_qsmc_dirty_gauss_3(:, count) = u;

    d = gaussDist(gauss_dist_min, gauss_dist_max); % calculate disturbance value

    dx = pendCart(delta_x_qsmc_dirty_gauss_3(:, count), m, M, L, g, d, u); %
calculate change of states

    delta_x_qsmc_dirty_gauss_3(:, count+1) = delta_x_qsmc_dirty_gauss_3(:, count) +
dx*t_step; % state update
end

%% DATA PLOTTING

%% Plot and save figure data for system without disturbance
delta_fig_clean = tiledlayout(2, 2, "Visible", "on");
title(delta_fig_clean, "Without Disturbance", 'Interpreter', 'latex');

% Plot theta data
nexttile();
plot(t_span, delta_x_qsmc_clean_1(1, :), ...
     t_span, delta_x_qsmc_clean_2(1, :), ...
     t_span, delta_x_qsmc_clean_3(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, delta_x_qsmc_clean_1(2, :), ...
     t_span, delta_x_qsmc_clean_2(2, :), ...
     t_span, delta_x_qsmc_clean_3(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');

% Plot control effort data
nexttile();
plot(t_span, delta_u_qsmc_clean_1, ...
     t_span, delta_u_qsmc_clean_2, ...
     t_span, delta_u_qsmc_clean_3);
```

```
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'$\delta_{QSMC}$=0.001', '$\delta_{QSMC}$=0.0001',
'$\delta_{QSMC}$=0.00001'}, ...
       'Location', 'south', 'Interpreter', 'latex');
axis off;

% Remove spaces between subplots
delta_fig_clean.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(delta_fig_clean, append(DELTA_WITHOUT_DIST_FILENAME, '.fig'));
    exportgraphics(delta_fig_clean, ...
                   append(DELTA_WITHOUT_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(DELTA_WITHOUT_DIST_FILENAME, '.mat'), ...
         'delta_x_qsmc_clean_1', 'delta_x_qsmc_clean_2', 'delta_x_qsmc_clean_3', ...
         'delta_u_qsmc_clean_1', 'delta_u_qsmc_clean_2', 'delta_u_qsmc_clean_3', ...
         't_span');
end

%% Plot and save figure data for system with sinusoidal disturbance
delta_fig_dirty_sine = tiledlayout(2, 2, "Visible", "on");
title(delta_fig_dirty_sine, "With Sinusoidal Disturbance", 'Interpreter', 'latex');

% Plot theta
nexttile();
plot(t_span, delta_x_qsmc_dirty_sine_1(1, :), ...
     t_span, delta_x_qsmc_dirty_sine_2(1, :), ...
     t_span, delta_x_qsmc_dirty_sine_3(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, delta_x_qsmc_dirty_sine_1(2, :), ...
     t_span, delta_x_qsmc_dirty_sine_2(2, :), ...
     t_span, delta_x_qsmc_dirty_sine_3(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');

% Plot control effort data
```

```
nexttile();
plot(t_span, delta_u_qsmc_dirty_sine_1, ...
     t_span, delta_u_qsmc_dirty_sine_2, ...
     t_span, delta_u_qsmc_dirty_sine_3);
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'$\delta_{QSMC}$=0.001', '$\delta_{QSMC}$=0.0001',
'$\delta_{QSMC}$=0.00001'}, ...
       'Location', 'south', 'Interpreter', 'latex');
axis off;

% Remove spaces between subplots
delta_fig_dirty_sine.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(delta_fig_dirty_sine, append(DELTA_WITH_SINE_DIST_FILENAME, '.fig'));
    exportgraphics(delta_fig_dirty_sine, ...
                   append(DELTA_WITH_SINE_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(DELTA_WITH_SINE_DIST_FILENAME, '.mat'), ...
         'delta_x_qsmc_dirty_sine_1', 'delta_x_qsmc_dirty_sine_2',
'delta_x_qsmc_dirty_sine_3', ...
         'delta_u_qsmc_dirty_sine_1', 'delta_u_qsmc_dirty_sine_2',
'delta_u_qsmc_dirty_sine_3', ...
         't_span');
end

%% Plot and save figure data for system with random disturbance
delta_fig_dirty_rand = tiledlayout(2, 2, "Visible", "on");
title(delta_fig_dirty_rand, "With Random Disturbance", 'Interpreter', 'latex');

% Plot theta data
nexttile();
plot(t_span, delta_x_qsmc_dirty_rand_1(1, :), ...
     t_span, delta_x_qsmc_dirty_rand_2(1, :), ...
     t_span, delta_x_qsmc_dirty_rand_3(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, delta_x_qsmc_dirty_rand_1(2, :), ...
     t_span, delta_x_qsmc_dirty_rand_2(2, :), ...
```

```
     t_span, delta_x_qsmc_dirty_rand_3(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');

% Plot control effort data
nexttile();
plot(t_span, delta_u_qsmc_dirty_rand_1, ...
     t_span, delta_u_qsmc_dirty_rand_2, ...
     t_span, delta_u_qsmc_dirty_rand_3);
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'$\delta_{QSMC}$=0.001', '$\delta_{QSMC}$=0.0001',
'$\delta_{QSMC}$=0.00001'}, ...
       'Location', 'south', 'Interpreter', 'latex');
axis off;

% Remove spaces between subplots
delta_fig_dirty_rand.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(delta_fig_dirty_rand, append(DELTA_WITH_RAND_DIST_FILENAME, '.fig'));
    exportgraphics(delta_fig_dirty_rand, ...
                   append(DELTA_WITH_RAND_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(DELTA_WITH_RAND_DIST_FILENAME, '.mat'), ...
         'delta_x_qsmc_dirty_rand_1', 'delta_x_qsmc_dirty_rand_2',
'delta_x_qsmc_dirty_rand_3', ...
         'delta_u_qsmc_dirty_rand_1', 'delta_u_qsmc_dirty_rand_2',
'delta_u_qsmc_dirty_rand_3', ...
         't_span');
end

%% Plot and save figure data for system with gaussian disturbance
delta_fig_dirty_gauss = tiledlayout(2, 2, "Visible", "on");
title(delta_fig_dirty_gauss, "With Gaussian Disturbance", 'Interpreter', 'latex');

% Plot theta data
nexttile();
plot(t_span, delta_x_qsmc_dirty_gauss_1(1, :), ...
     t_span, delta_x_qsmc_dirty_gauss_2(1, :), ...
     t_span, delta_x_qsmc_dirty_gauss_3(1, :));
title("Angle", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
```

```
ylabel("$\theta$ (rad)", 'Interpreter', 'latex');

% Plot theta dot data
nexttile();
plot(t_span, delta_x_qsmc_dirty_gauss_1(2, :), ...
     t_span, delta_x_qsmc_dirty_gauss_2(2, :), ...
     t_span, delta_x_qsmc_dirty_gauss_3(2, :));
title("Angular Velocity", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("$\dot{\theta}$ (rad/s)", 'Interpreter', 'latex');

% Plot control effort data
nexttile();
plot(t_span, delta_u_qsmc_dirty_gauss_1, ...
     t_span, delta_u_qsmc_dirty_gauss_2, ...
     t_span, delta_u_qsmc_dirty_gauss_3);
title("Control Effort", 'Interpreter', 'latex');
xlabel("t (s)", 'Interpreter', 'latex');
ylabel("u (N)", 'Interpreter', 'latex');

% Add legend
dummy = linspace(0, 1, 100)';
nexttile();
plot(dummy, nan);
legend({'$\delta_{QSMC}$=0.001', '$\delta_{QSMC}$=0.0001',
'$\delta_{QSMC}$=0.00001'}, ...
       'Location', 'south', 'Interpreter', 'latex');
axis off;

% Remove spaces between subplots
delta_fig_dirty_gauss.Padding = 'none';

% Save figure
if SAVE_FIGURE == true
    saveas(delta_fig_dirty_gauss, append(DELTA_WITH_GAUSS_DIST_FILENAME, '.fig'));
    exportgraphics(delta_fig_dirty_gauss, ...
                   append(DELTA_WITH_GAUSS_DIST_FILENAME, '.png'), ...
                   "Resolution", IMG_RES);
end

% Save data to .mat file
if SAVE_DATA_TO_MAT == true
    save(append(DELTA_WITH_GAUSS_DIST_FILENAME, '.mat'), ...
         'delta_x_qsmc_dirty_gauss_1', 'delta_x_qsmc_dirty_gauss_2',
'delta_x_qsmc_dirty_gauss_3', ...
         'delta_u_qsmc_dirty_gauss_1', 'delta_u_qsmc_dirty_gauss_2',
'delta_u_qsmc_dirty_gauss_3', ...
         't_span');
end

%% ----------------------------------------------------------------------

%% HELPER FUNCTIONS

%% ----------------------------------------------------------------------
```

```
%% Control effort
function u = SMCCtrlEff(x, m, M, L, k_smc, c_smc, g) % SMC control effort
    Cx = cos(x(1));
    Sx = sin(x(1));
    Tx = tan(x(1));

    sigma = -(x(2) + c_smc*x(1));

    disc_term = k_smc*sign(sigma);

    u = (M+m)*g*Tx - m*L*x(2)*Sx + (disc_term-c_smc*x(2))*(m*L*Cx^2-(M+m))/Cx;
end

function u = QSMCCtrlEff(x, m, M, L, k, c, g, delta) % QSMC control effort
    Cx = cos(x(1));
    Sx = sin(x(1));
    Tx = tan(x(1));

    sigma = -(x(2) + c*x(1));

    disc_term = k*sigma/(norm(sigma)+delta);

    u = (M+m)*g*Tx - m*L*x(2)*Sx + (disc_term-c*x(2))*(m*L*Cx^2-(M+m))/Cx;
end

%% System dynamics
function dx = pendCart(x, m, M, L, g, d, u) % dynamics of inverted pendulum
    % x(1) = pendulum angle
    % x(2) = pendulum angle velocity

    Sx = sin(x(1));
    Cx = cos(x(1));
    denom = m*L*Cx^2* - (M+m);

    dx(1,1) = x(2);
    dx(2,1) = (-(M+m)*g*Sx + m*L*x(2)*Sx*Cx + u*Cx)/denom + d;
end

%% Disturbance

function val = sineDist(t_now, sine_dist_freq, sine_dist_amp) % to simulate
sinusoidal disturbance
    val = sine_dist_amp * sin(2 * pi * sine_dist_freq * t_now);
end

function val = randDist(min, max) % to simulate random disturbance
    val = rand*(max-min) + min;
end

function val = gaussDist(min, max) % to simulate random disturbance
    val = randn*(max-min) + min;
end

%% ------------------------------------------------------------------------
```