Tugas Supervisory Control Pengendali Pintu Otomatis Dengan Simulasi Hardware in The Loop

Oleh:
Sidartha Prastya. P - 13219033

INSTITUT TEKNOLOGI BANDUNG
2022

DAFTAR ISI

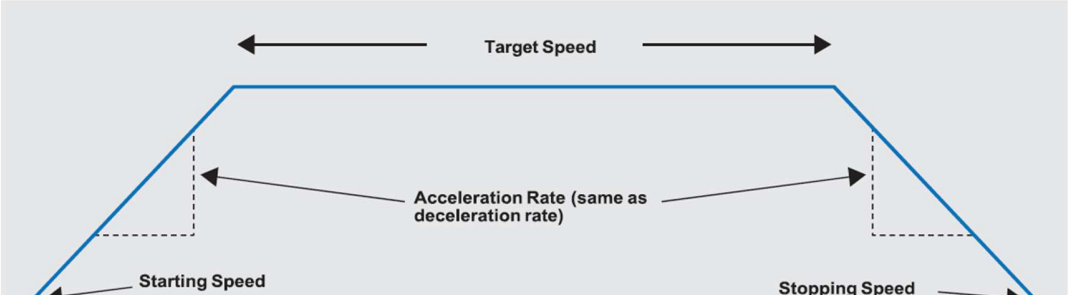
DAFTAR GAMBAR

1 Spesifikasi

Pada tugas "*Supervisory Control* Pengendali Pintu Otomatis" ini, terdapat beberapa spesifikasi yang diperlukan, yaitu:

- Time constant motor (kecepatan) = 1,2 detik
- Frekuensi sampling: 100 Hz (perioda = 10 ms)
- Terdapat sebuah tombol yang berperan sebagai tombol buka dan tutup sekaligus
- Apabila setelah 10 detik terbuka, maka pintu akan secara otomatis menutup.
- Ketika pintu sedang menutup, apabila tombol ditekan, maka pintu akan kembali terbuka.

Secara umum, target profil kecepatan dari pintu adalah seperti gambar berikut.



Gambar 1. Spesifikasi profil kecepatan

Pada gambar spesifikasi profil kecepatan, dapat disimpulkan bahwa terdapat 2 tipe kendali, yaitu kendali kecepatan dan percepatan. Kendali percepatan terjadi saat awal pintu bergerak dari diam menuju kecepatan puncak serta saat kecepatan puncak hingga diam kembali (perlambatan). Kendali kecepatan berada pada saat kecepatan konstan. Kemudian, gambar tersebut diperpanjang dari kecepatan positif, yaitu pada saat pintu bergerak membuka hingga pintu menutup kembali, yaitu terjadi pada saat kecepatan negatif. Dalam melakukan pergantian kontrol tersebut atau yang dapat disebut sebagai *supervisory control*, maka dibutuhkan FSM yang berfungsi untuk melihat keadaan di saat itu serta memberikan keluaran yang menentukan kondisi setelahnya.

2 Perhitungan Transformasi Bilinear

Dalam melakukan simulasi, maka perlu dibuat suatu persamaan yang mewakili sifat dari sebuah motor. Persamaan motor dapat dibuat dalam sebuah persamaan dalam domain frekuensi orde satu:

$$H(s) = \frac{K}{\tau s + 1}$$

Agar dapat diterapkan ke dalam simulasi, maka persamaan tersebut perlu untuk diubah ke dalam domain waktu diskrit. Oleh karena itu, digunakan sebuah Transformasi Bilinear yang mengubah persamaan dari domain s (frekuensi) menjadi domain z. Domain z kemudian dapat ditransformasi kembali menjadi domain waktu diskrit n.

Pada spesifikasi, diminta sebuah persamaan motor dengan *time constant* $\tau = 1{,}2\ detik$ dan *time sampling* $\mathrm{T} = 10$ ms. Maka, didapat sebuah persamaan mula-mula:

$$H(s) = \frac{1}{1{,}2s + 1}$$

Persamaan di atas dilakukan transformasi bilinear dengan melakukan substitusi:

$$s = \frac{2(z-1)}{T(z+1)} = \frac{200(z-1)}{z+1}$$

Maka, persamaan selanjutnya menjadi:

$$H(z) = \frac{0{,}83(z+1)}{200{,}83z - 199{,}17} = \frac{0{,}00413 + 0{,}00413z^{-1}}{1 - 0{,}992z^{-1}}$$

Dengan menguraikan $H(z) = \frac{Y(z)}{X(z)}$, maka persamaan di atas menjadi:

$$Y(z) - 0.992z^{-1}Y(z) = 0.00413X(z) + 0.00413z^{-1}X(z)$$

Persamaan dalam domain z di atas dapat diubah ke dalam domain n menjadi:

$$y(n) = 0.00413x(n) + 0.00413x(n-1) + 0.992y(n-1)$$

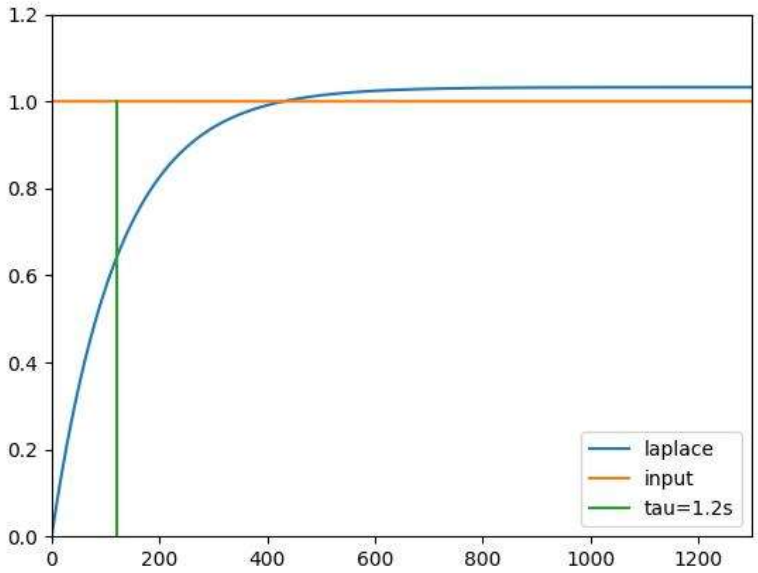
3 Simulasi Menentukan Time Constant

Dari persamaan di atas, kemudian dibuat sebuah program simulasi untuk melakukan verifikasi terhadap persamaan tersebut. Program dibuat dalam program python sebagai berikut.

```
import matplotlib.pyplot as plt

def motor(input, output):
    output[1] = output[0]
    # Insert to Laplace equation (1/1.2s+1) with time samp = 10 ms
    output[0] = 0.00413 * input[0] + 0.00413 * input[1] + 0.992 * output[1]
    return output

x = [0, 0]
y = [0, 0]
plt.axis([0, 1300, 0, 1.2])
x_axis = []
ydata = []
yin = []


for i in range(0, 1300): #sampling dari n = 0 hingga 1299
    x[1] = x[0]
    x[0] = 1
    y = motor(x, y)
    x_axis.append(i)
```

```
    ydata.append(y[0])
    yin.append(x[0])

tao_line = [0, 1]
tao_axis = [120, 120]
plt.plot(x_axis, ydata, label="laplace")
plt.plot(x_axis, yin, label="input")
plt.plot(tao_axis, tao_line, label="tau=1.2s")
plt.legend()
plt.savefig("time_const.jpg")
```

Dari program di atas, didapatkan hasil gambar sebagai berikut:



Gambar 2. Verifikasi time constant persamaan motor

Dari hasil simulasi persamaan motor dengan input berupa *step response* didapatkan bahwa garis τ berada di perpotongan pada saat garis motor (laplace) memiliki nilai ≈ 0.63. Setiap n pada sumbu x grafik memiliki skala 10 ms sesuai dengan *sampling time yang digunakan.* Pada grafik, garis τ berada di n = 120 atau pada saat $t = 1200\,ms$. Hal tersebut membuktikan bahwa persamaan yang digunakan telah sesuai dengan spesifikasi.

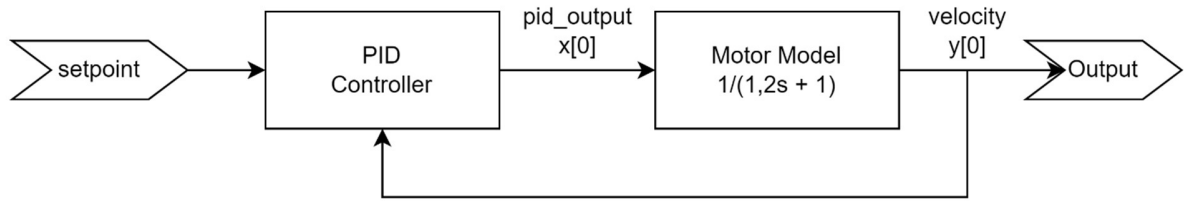
4 Simulasi Kendali Kecepatan Pintu di Desktop

Dari persamaan yang ada, kemudian dilakukan pengendalian kecepatan dengan menggunakan kontroler PID. Berikut adalah program simulasi yang dibuat:

```
import matplotlib.pyplot as plt
from simple_pid import PID
import time

#motor_laplace is actually a velocity function
def motor(input, output):
    output[1] = output[0]
    output[0] = 0.00413 * input[0] + 0.00413 * input[1] + 0.992 * output[1]
    return output

def pid(input, kp, ki, kd, setpoint, time_samp):
    global integral, last_error
    error = setpoint - input
    integral += error * time_samp
    deriv = (last_error - error) / time_samp
    last_error = error
    return (kp * error + ki * integral + kd * deriv)

KP = 50.0
KI = 15.0
KD = 0.1

setpoint = 0
time_samp = 0.01   # 10 ms
last_error = 0
integral = 0

x = [0, 0]
y = [0, 0]
plt.axis([0, 100, -2, 2])
x_axis = []
ydata = []
yin = []

for i in range (100):
    x[1] = x[0]
    # #-------------------------------
    # # Uncomment to select
    # #-------------------------------
    # # Unit Step
```

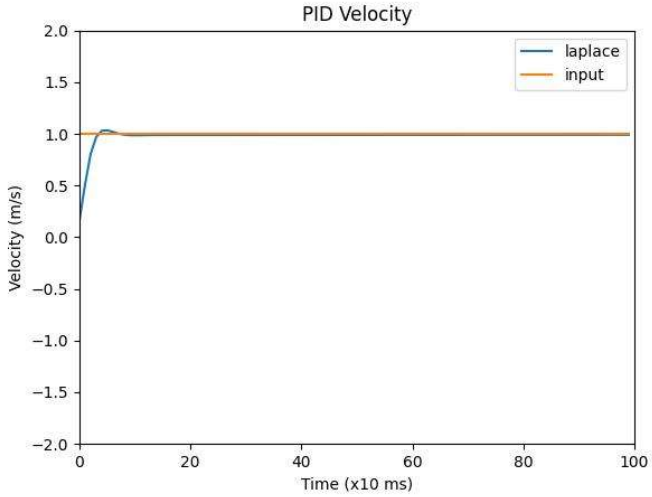
```
    # setpoint = 1
    # #-------------------------------
    # # Unit Ramp Up
    # if (i < 20):
    #     setpoint = 0
    # elif (i >= 20 and setpoint < 1):
    #     setpoint += 0.05
    # else:
    #     setpoint = 1
    # #-------------------------------
    # # Unit Ramp Down
    if (i < 20):
        setpoint = 0
    elif (i >= 20 and setpoint > -1):
        setpoint -= 0.05
    else:
        setpoint = -1
    # #-------------------------------

    x[0] = pid(y[0], KP, KI, KD, setpoint, time_samp)
    y = motor(x, y)
    x_axis.append(i)
    ydata.append(y[0])
    yin.append(setpoint)


plt.plot(x_axis, ydata, label="laplace")
plt.plot(x_axis, yin, label="input")
plt.title("PID Velocity")
plt.xlabel("Time (x10 ms)")
plt.ylabel("Velocity (m/s)")
plt.legend()
plt.savefig("velocity.jpg")
```

Program tersebut dapat direpresentasikan ke dalam diagram blok seperti berikut.

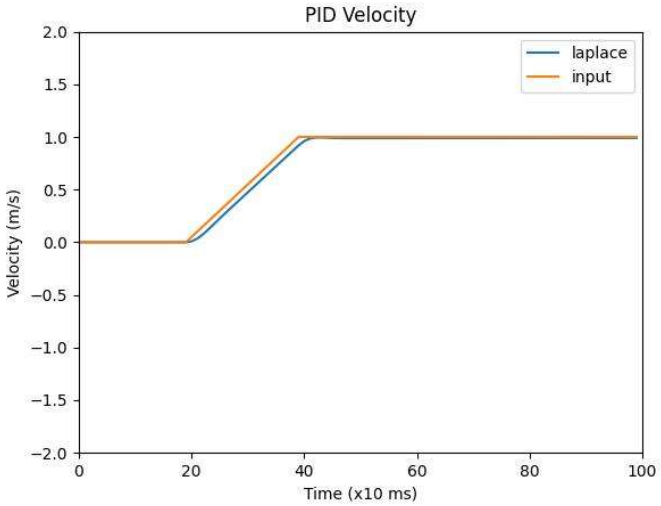


Gambar 3. Diagram blok sistem kendali kecepatan

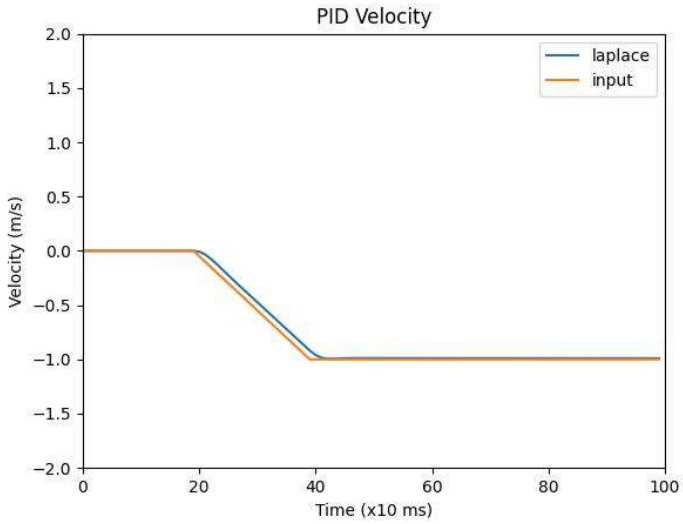
Program di atas dapat menghasilkan 3 buah grafik, yaitu grafik dengan input *step response, ramp* naik, dan *ramp* turun:



Gambar 4. Simulasi PID kecepatan dengan step response



Gambar 5. Simulasi PID kecepatan dengan input ramp response naik



Gambar 6. Simulasi PID kecepatan dengan input ramp response turun

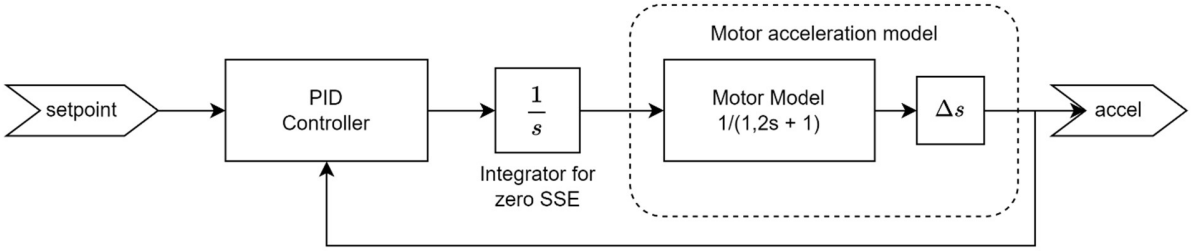
5 Simulasi Kendali Percepatan Pintu di Desktop

Dari model motor sebelumnya, juga dilakukan simulasi kendali percepatan dengan PID. Berikut adalah program yang digunakan pada kendali percepatan.

```
import matplotlib.pyplot as plt
from simple_pid import PID
import time

#motor_laplace is actually a velocity function
def motor(input, output):
    output[1] = output[0]
    # Insert to Laplace equation (10/s+10) with time samp = 10 ms
    # output[0] = 0.0476 * input[0] + 0.0476 * input[1] + 0.9048 * output[1]
    output[0] = 0.00413 * input[0] + 0.00413 * input[1] + 0.992 * output[1]
    return output

def pid(input, kp, ki, kd, setpoint, time_samp):
    global integral, last_error
    error = setpoint - input
    integral += error * time_samp
    deriv = (last_error - error) / time_samp
    last_error = error
    return (kp * error + ki * integral + kd * deriv)

KP = 0.8
```

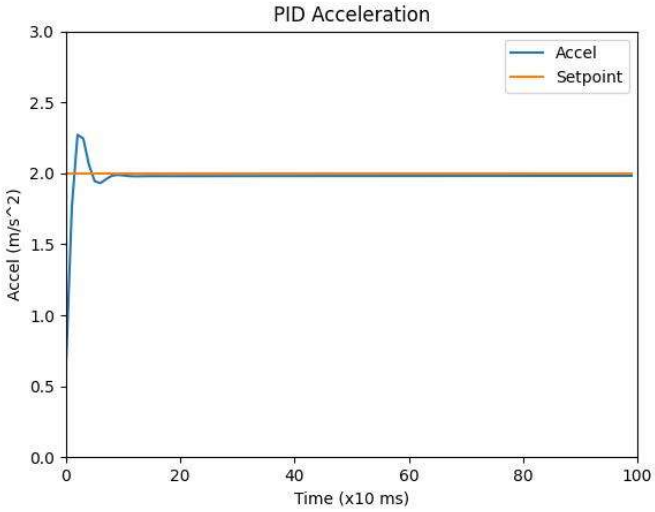
```
KI = 0.1
KD = 0.0

setpoint = 2
time_samp = 0.01   # 10 ms
last_error = 0
integral = 0

curr_accel = 0

x = [0, 0]
y = [0, 0]
plt.axis([0, 100, -2, 2])
x_axis = []
ydata = []
yin = []

for i in range(100):
    x[1] = x[0]
    if (i < 20 or i > 70):
        setpoint = 0
    else:
        setpoint = 1

    x[0] = pid(curr_accel, KP, KI, KD, setpoint, time_samp)

    # Add integrator to change the equation from vel to accel
    x[0] += x[1]

    y = motor(x, y)
    curr_accel = (y[0] - y[1])/0.01
    x_axis.append(i)
    ydata.append(curr_accel)
    yin.append(setpoint)


plt.plot(x_axis, ydata, label="Accel")
plt.plot(x_axis, yin, label="Setpoint")
plt.title("PID Acceleration")
plt.xlabel("Time (x10 ms)")
plt.ylabel("Accel (m/s^2)")
plt.legend()
# plt.show()
plt.savefig("accel.jpg")
```

Program di atas dapat direpresentasikan sebagai diagram blok berikut.

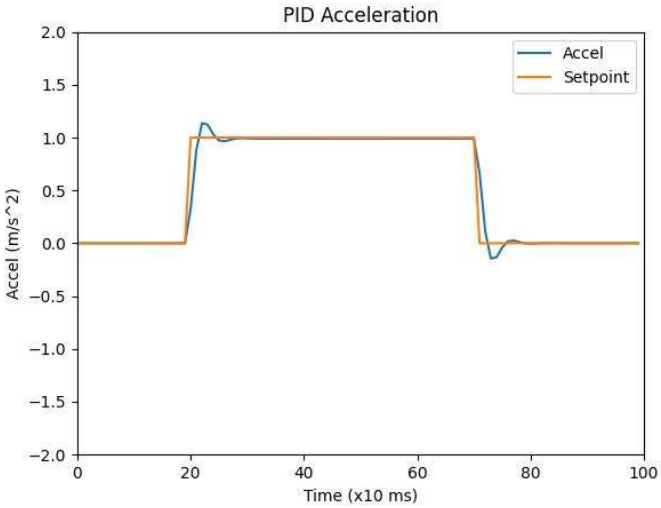


Gambar 7. Diagram blok sistem kendali percepatan

Simulasi kendali percepatan dilakukan dengan menggunakan *step response* saja karena pada kondisi yang digunakan pada *supervisory control* hanyalah kendali percepatan konstan. Berikut adalah hasil simulasi yang dilakukan.



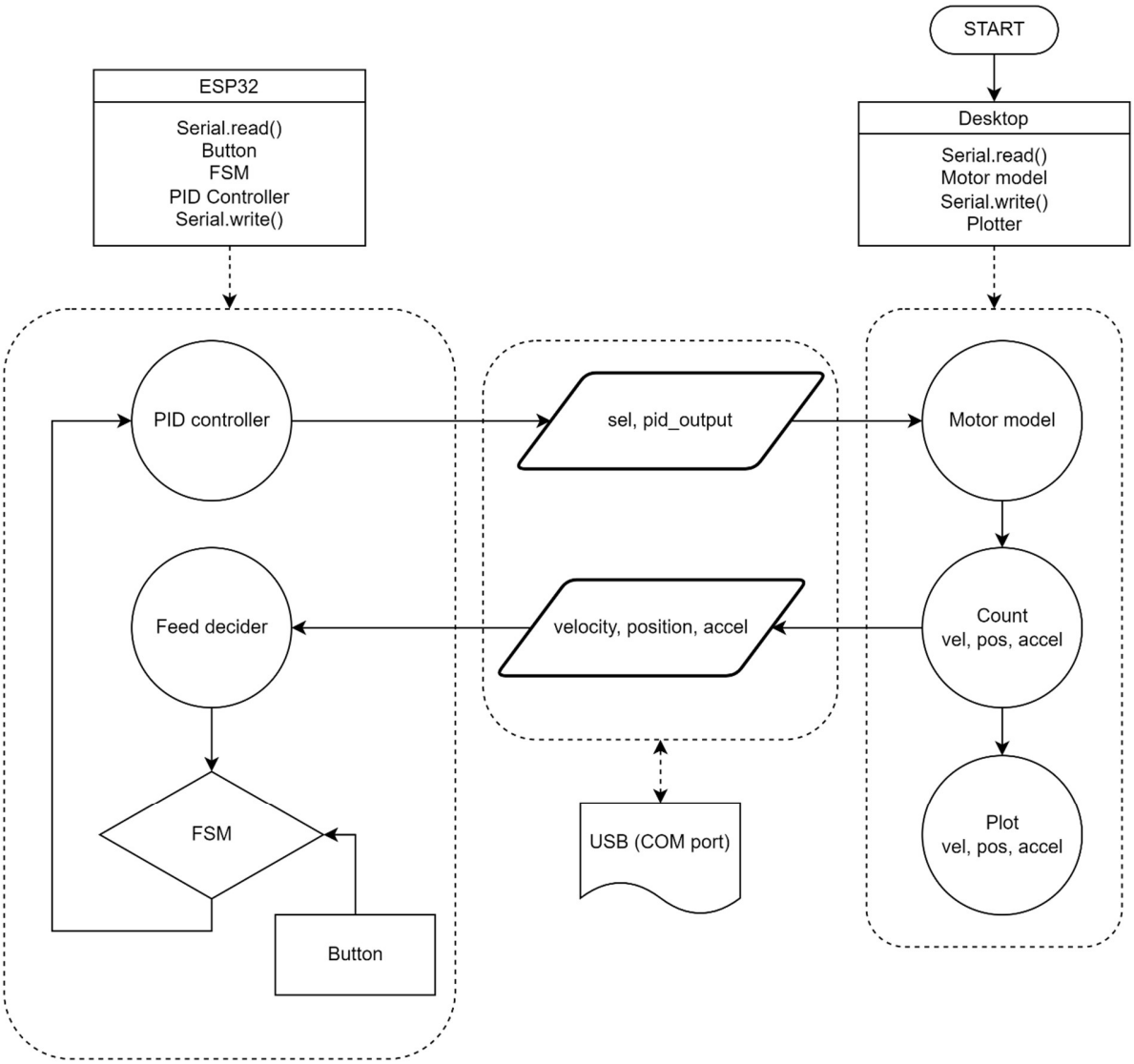
Gambar 8. Hasil kendali PID percepatan



Gambar 9. Hasil simulasi kendali percepatan

6 Simulasi HIL dengan desktop dan mikrokontroler

Pada simulasi HIL dengan desktop dan mikrokontroler, komponen terbagi menjadi 2 bagian, yaitu sistem kontrol pada mikrokontroler ESP32 serta model motor pada desktop. *Data flow diagram* pada sistem dapat diilustrasikan sebagai berikut.



Gambar 10. Data flow diagram simulasi HIL Supervisory Control

6.1 Desain Software di Desktop

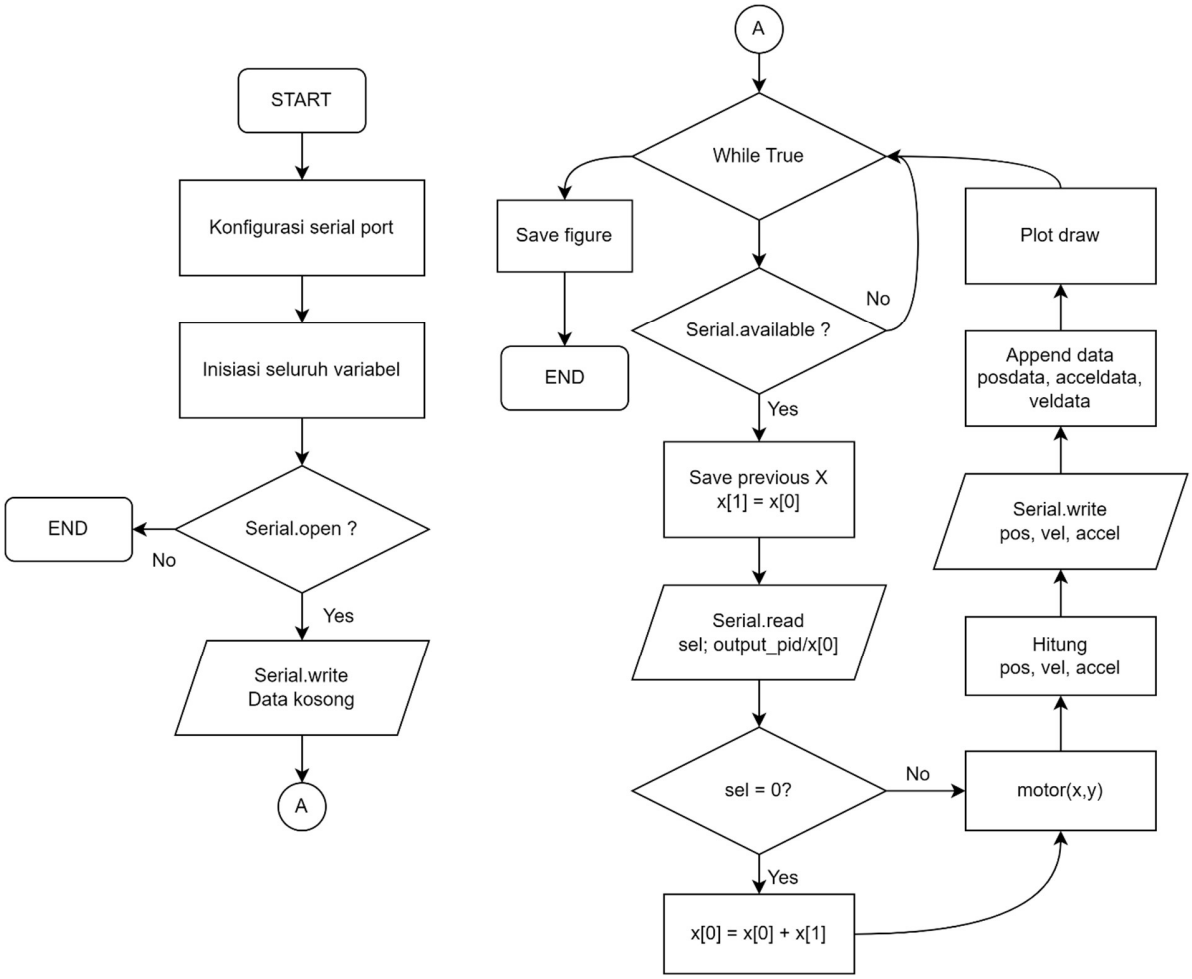
Pada desktop, dibuat suatu program bernama "motor_pc.py" yang berfungsi sebagai model dari motor yang digunakan, melakukan perhitungan terhadap posisi, kecepatan, dan percepatan motor, serta melakukan *plotting* terhadap hasil perhitungan. Program menerima input berupa selektor kontrol dan output dari pid. Lalu, setelah melakukan perhitungan, program desktop akan mengirimkan keluaran berupa hasil perhitungan posisi, kecepatan, dan percepatan. Komunikasi dilakukan dengan mikrokontroler secara dua arah menggunakan USB melalui *port COM*. Berikut adalah *source code* yang dibuat:

```
import serial
import time
```

```
import numpy as np
import matplotlib.pyplot as plt

def motor(input, output):
    output[1] = output[0]
    output[0] = 0.00413 * input[0] + 0.00413 * input[1] + 0.992 * output[1]
    return output

xmin = 0
ser = serial.Serial('COM5', 115200, bytesize=serial.EIGHTBITS,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        timeout=None)
plt.axis([0, 1000, -3, 3])
xdata = []
ydata = []
posdata = []
acceldata = []
x = [0.0,0.0]
y = [0.0,0.0]
i = 0
posisi = 0.0
if ser.is_open:
    # Melakukan write awal dengan nilai 0 agar program karena mikon hanya dapat
bekerja setelah menerima input
    ser.write(("%f;%f;%f\r\n" % (0.0, 0.0, 0.0)).encode("utf-8"))

    while (True):
        try:
            size = ser.inWaiting()
            if size:
                x[1] = x[0]
                [sel, x[0]] = [float(v) for v in (ser.readline().decode("utf-
8").split(";"))]
                if(sel == 0.0):
                    x[0] += x[1]

                y = motor(x, y)

                posisi += y[0] * 0.01
                accel = (y[0] - y[1]) / 0.01

                ser.write(("%f;%f;%f\r\n" % (y[0], posisi, accel)).encode("utf-8"))

                posdata.append(posisi)
```

```
                acceldata.append(accel)
                ydata.append(y[0])
                xdata.append(i)

                i += 1
                if (i % 10 == 0):
                    plt.clf()
                    if(i <= 1000):
                        plt.axis([0, 1000, -3, 3])

                    else:
                        plt.axis([0, i, -3, 3])

                    # Hanya melakukan plot kecepatan secara realtime agar tidak
berat
                    plt.plot(xdata, ydata, label="Profil kecepatan")
                    plt.draw()
                    plt.pause(0.05)

        except Exception as e:
            plt.clf()
            plt.axis([0, i, -3, 3])
            plt.plot(xdata, ydata, label="Profil kecepatan")
            plt.plot(xdata, posdata, label="Profil posisi")
            plt.plot(xdata, acceldata, label="Profil Akselerasi")
            plt.legend()
            plt.savefig("hasil.jpg") # Menyimpan gambar lengkap
            print(e)
            break
```

Program di atas dapat diilustrasikan melalui *flowchart* berikut:



Gambar 11. Flowchart software Supervisory HIL pada desktop

6.2 Desain Software di Mikrokontroler

Peran dari mikrokontroler pada simulasi ini adalah sebagai *supervisor* dan modul PID kontroler. Oleh karena itu, pada mikrokontroler terbagi menjadi 2 segmen, yaitu *controller* serta FSM yang dibuat melalui *source code* berikut:

```
#include <stdio.h>
#include "driver/gpio.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/timer.h"
#include "fsm.h"

#define GPIO_INPUT_PB    15
#define GPIO_INPUT_PIN_SEL  (1ULL<<GPIO_INPUT_PB)
```

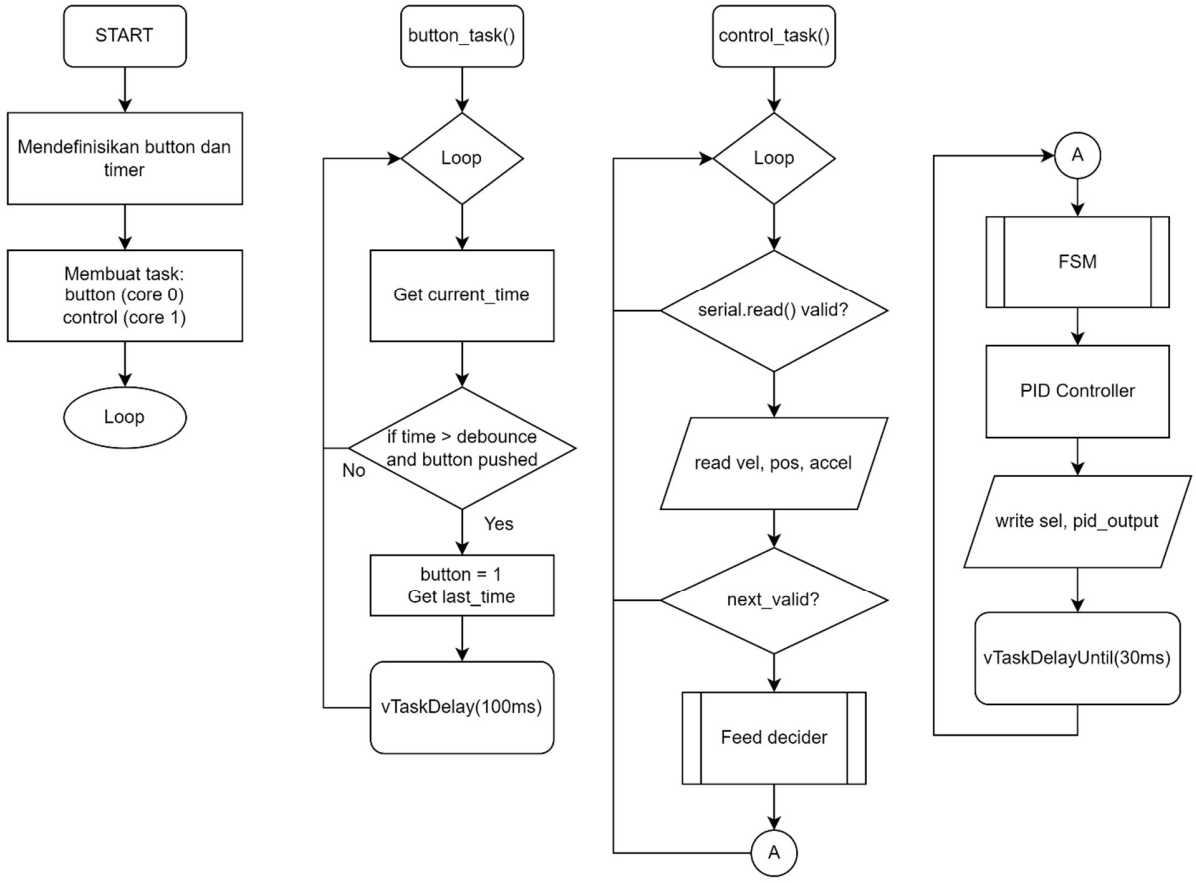
```
#define ESP_INTR_FLAG_DEFAULT 0
#define TIMER_DIVIDER         16
#define TIMER_SCALE           (TIMER_BASE_CLK /TIMER_DIVIDER)
#define DELAY_S               0.25
#define NUMBER_OF_LED         8
#define TIMER1_INTERVAL_SEC   (DELAY_S * NUMBER_OF_LED)

#define KP_vel  50.0
#define KI_vel  15.0
#define KD_vel  0.1

#define KP_acc  0.8
#define KI_acc  0.1
#define KD_acc  0.0

const TickType_t xDelay = 10 / portTICK_PERIOD_MS;
double current_time_sec = 0, last_time_sec = 0, last_reset_time = 0; // untuk
debounce sampling time

float vel = 0.0, pos = 0.0, accel = 0.0; //kecepatan dari motor
float output_pid = 0.0;
float setpoint = 0.0;
int next_valid = 1, button = 1, start_program = 0;
int counter, last_debounce, feed, state, sel;
float input;
float Kp, Ki, Kd;

float integral = 0;
float last_err = 0;

void pid(float input, float *output, float setpoint, float kp, float ki, float kd,
float time){

    float error = setpoint - input;
    integral += error * time;
    float deriv = (last_err - error)/time;
    last_err = error;
    *output = kp * error + ki * integral + kd * deriv;
}

void button_config(){
    gpio_config_t io_conf;
    io_conf.pin_bit_mask = GPIO_INPUT_PIN_SEL;
    io_conf.mode = GPIO_MODE_DEF_INPUT;
    io_conf.intr_type = GPIO_INTR_NEGEDGE; // Falling Edge
```

```
    io_conf.pull_up_en = 1; // enable interrupt
    io_conf.pull_down_en = 0;
    gpio_config(&io_conf);
}

void timer_config(){
    // TIMER DEBOUNCE**************************
    timer_config_t config = {
        .divider = TIMER_DIVIDER,
        .counter_dir = TIMER_COUNT_UP,
        .counter_en = TIMER_START,
        .alarm_en = TIMER_ALARM_EN,
        .auto_reload = TIMER_AUTORELOAD_DIS,
    };

    timer_init(0, 0, &config); //TIMERG0, hw_timer[0]
    timer_set_counter_value(0, 0, 0x00000000ULL);
    timer_start(0, 0);
}

void main_control(void *pvParam){
    while(1){
        TickType_t xLastWakeTime1 = xTaskGetTickCount();

        // Membaca data dari desktop agar bisa lanjut
        if(scanf("%f;%f;%f", &vel, &pos, &accel) == 3){
            next_valid = 1;
        }

        // Dijalankan setelah pembacaan berhasil
        if(next_valid == 1 ){
            // Melihat kondisi saat ini pada motor sebagai penentu kelanjutan state
            if(state == 1 && (vel >= 1.0 || pos >= 2.0)){
                feed = 1;
            }
            else if(state == 2 && pos >= 1.5 ){
                feed = 1;
            }
            else if(state == 3 && pos >= 2.0){
                feed = 1;
            }
            else if(state == 5 && vel <= -1.0){
                feed = 1;
            }
            else if(state == 6 && pos <= 0.5){
```

```
                feed = 1;
            }
            else if(state == 7 && pos <= 0.0){
                feed = 1;
            }

            fsm(&counter, &button, &feed, &state, &sel, &setpoint);
            if(sel == 0){
                Kp = KP_acc;
                Ki = KI_acc;
                Kd = KD_acc;
                input = accel;
            }
            else{
                Kp = KP_vel;
                Ki = KI_vel;
                Kd = KD_vel;
                input = vel;
            }

            pid(input, &output_pid, setpoint, Kp, Ki, Kd, 0.01);

            // Mengirimkan sel dan output_pid ke desktop
            printf("%d;%.8f\r\n", sel, output_pid);

            next_valid = 0;
        }
        vTaskDelayUntil(&xLastWakeTime1, 30/portTICK_PERIOD_MS);
    }
}

void button_task(void *pvParam){
    while(1){
        timer_get_counter_time_sec(0, 0, &current_time_sec);
            if (gpio_get_level(GPIO_INPUT_PB) == 0 && (current_time_sec -
last_time_sec > DELAY_S)) {
                button = 1;
                //start_program = 1;
                timer_get_counter_time_sec(0, 0, &last_time_sec);
            }
        vTaskDelay(100/portTICK_PERIOD_MS);
    }
}

void app_main()
```

```
{
    button_config();
    timer_config();

    // Task untuk button pada core 0
    xTaskCreatePinnedToCore(button_task, "Button Task", 2048, NULL, 1, NULL, 0);
    // Task untuk kontrol PID pada core 1
    xTaskCreatePinnedToCore(main_control, "Main Task", 2048, NULL, 1, NULL, 1);
}
```

Program di atas dapat diilustrasikan ke dalam *flowchart* seperti berikut.



Gambar 12. Flowchart Supervisory HIL pada ESP32

Adapun dibuat file “fsm.h” yang berisikan FSM/ESM yang digunakan pada program ini sebagai berikut:

```
#ifndef FSM_H
#define FSM_H

#include <stdio.h>
```

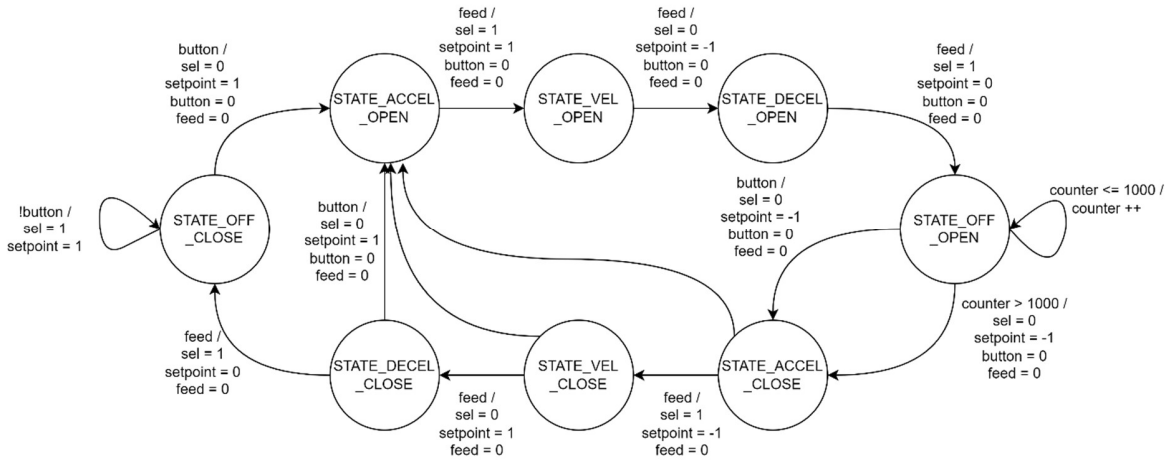
```c
#include <stdlib.h>
#include <time.h>

#define STATE_OFF_CLOSE     0
#define STATE_ACCEL_OPEN    1
#define STATE_VEL_OPEN      2
#define STATE_DECEL_OPEN    3
#define STATE_OFF_OPEN      4
#define STATE_ACCEL_CLOSE   5
#define STATE_VEL_CLOSE     6
#define STATE_DECEL_CLOSE   7


#define SEL_ACCEL   0
#define SEL_VEL     1

void fsm(int *counter, int *button, int *feed_motor, int *state, int *select, float
*setpoint){
    switch (*state){
        case STATE_OFF_CLOSE:
            if(*button == 1){                //tombol ditekan
                *state = STATE_ACCEL_OPEN;
                *select = 0;                 //accel
                *setpoint = 1;              //2 m/s^2
                *button = 0;                 //reset button
                *feed_motor = 0;             //reset feed motor
            }
            else{
                *setpoint = 0;               //setpoint kecepatan = 0
                *select = 1;
            }
            break;

        case STATE_ACCEL_OPEN:
            if(*feed_motor == 1){
                *state = STATE_VEL_OPEN;
                *select = 1;                 //velocity
                *setpoint = 1.0;            //10 m/s
                *feed_motor = 0;
                *button = 0;
            }
            else{

            }
            break;
```

```
        case STATE_VEL_OPEN:
            if(*feed_motor == 1){
                *state = STATE_DECEL_OPEN;
                *select = 0;
                *setpoint = -1;
                *feed_motor = 0;
                *button = 0;
            }
            else{

            }
            break;

        case STATE_DECEL_OPEN:
            if(*feed_motor == 1){
                *state = STATE_OFF_OPEN;
                *select = 1;
                *setpoint = 0.0;
                *feed_motor = 0;
                *button = 0;
            }
            else{

            }
            break;

        case STATE_OFF_OPEN:
            if(*button == 1){
                *state = STATE_ACCEL_CLOSE;
                *select = 0;
                *setpoint = -1;
                *button = 0;
                *feed_motor = 0;
            }
            else{
                *counter += 1;
                if(*counter > 1000){
                    *button = 1;
                    *counter = 0;
                }
            }
            break;

        case STATE_ACCEL_CLOSE:
```

```
            if(*feed_motor == 1){
                *state = STATE_VEL_CLOSE;
                *select = 1;
                *setpoint = -1.0;
                *feed_motor = 0;
            }
            else{
                if(*button == 1){
                    *state = STATE_ACCEL_OPEN;
                    *select = 0;
                    *setpoint = 1;
                    *feed_motor = 0;
                    *button = 0;
                }
            }
            break;

        case STATE_VEL_CLOSE:
            if(*feed_motor == 1){
                *state = STATE_DECEL_CLOSE;
                *select = 0;
                *setpoint = 1;
                *feed_motor = 0;

            }
            else{
                if(*button == 1){
                    *state = STATE_ACCEL_OPEN;
                    *select = 0;
                    *setpoint = 1;
                    *feed_motor = 0;
                    *button = 0;
                }
            }
            break;

        case STATE_DECEL_CLOSE:
            if(*feed_motor == 1){
                *state = STATE_OFF_CLOSE;
                *select = 1;
                *setpoint = 0.0;
                *feed_motor = 0;

            }
            else{
```

```
                if(*button == 1){
                    *state = STATE_ACCEL_OPEN;
                    *select = 0;
                    *setpoint = 1;
                    *feed_motor = 0;
                    *button = 0;
                }
            }
            break;

        default:
            break;
    }
}

#endif
```

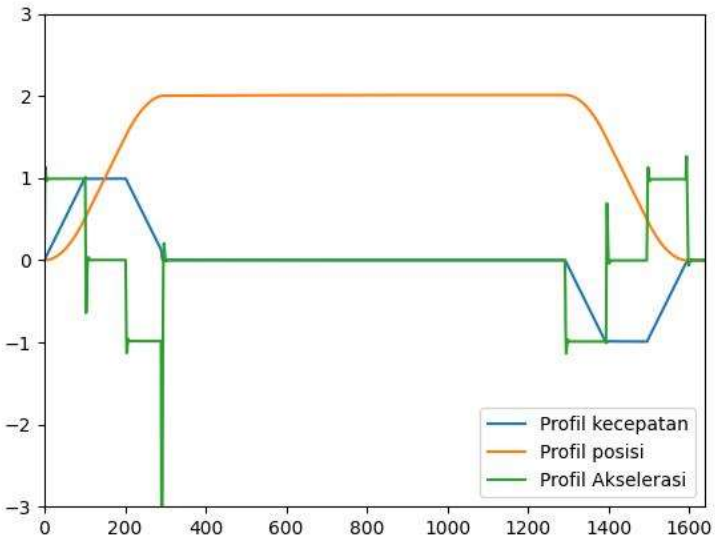
FSM di atas dapat diilustrasikan ke dalam diagram berikut:



Gambar 13. Diagram FSM Supervisory Control HIL

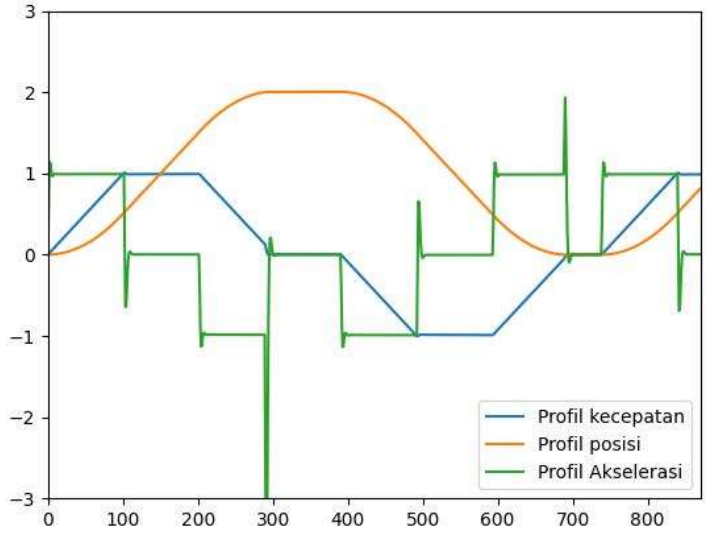
Dari simulasi gabungan antara mikrokontroler dan desktop, didapatkan grafik kecepatan, posisi, dan akselerasi. Pada simulasi, diasumsikan keluaran kecepatan memiliki satuan meter/detik, sehingga posisi yang didapatkan merupakan satuan meter. Pada simulasi, digunakan posisi maksimum 2 meter. Dalam satu perioda aksi (membuka/menutup), pintu membutuhkan waktu sekitar 3 detik. Terdapat 3 buah *use case* yang telah dibuat, yaitu:

- Pintu terbuka, menunggu 10 detik, lalu menutup kembali



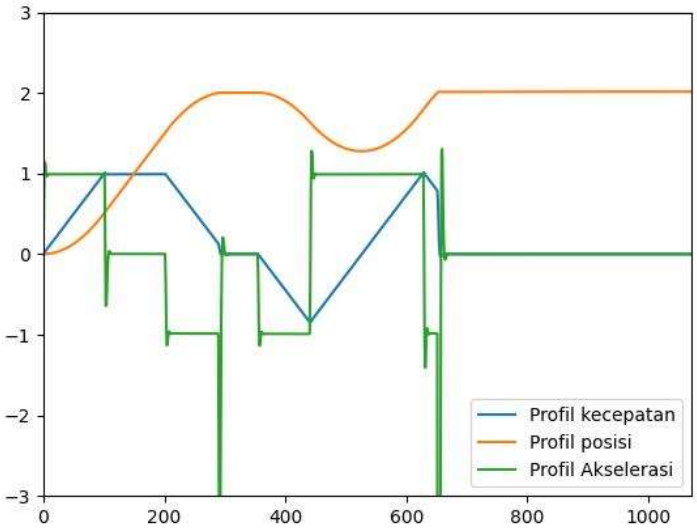
Gambar 14. Pintu terbuka, lalu menutup setelah 10 detik

- Pintu terbuka, belum 10 detik, tapi *user* menekan tombol untuk menutup sebelum 10 detik.



Gambar 15. User menekan tombol untuk menutup pintu sebelum 10 detik

- Pada saat pintu sedang menutup, *user* menekan tombol untuk membuka pintu sebelum pintu tertutup sempurna



Gambar 16. User menekan tombol untuk membuka pintu saat pintu sedang menutup

7 Kesimpulan

Dari simulasi yang dilakukan, didapatkan kesimpulan sebagai berikut:

- *State Machine* sistem kendali *supervisory* berhasil dibuat dengan menggunakan mikrokontroler sebagai FSM dan PID *controller* serta desktop sebagai model dari motor dan *plotter* grafik.
- Pada kendali kecepatan, PID berhasil digunakan dengan ditunjukkan pada hasil simulasi dengan 3 buah input berbeda, yaitu unit step, ramp naik, dan ramp turun.
- Pada kendali percepatan, PID berhasil dignakan dengan memanfaatkan integrator tambahan pada output PID agar menghasilkan *zero steady state error.* Sama seperti pada model kecepatan, hanya saja ditambahkan sebuah diferensiator berupa perhitungan percepatan sebagai input *feedback* untuk PID.
- Secara total sistem dapat berfungsi dengan baik, walaupun pada grafik hasil terlihat kurang halus. Akan tetapi, secara fungsional, setiap *use case* telah berhasil terpenuhi.

8 Lampiran

Seluruh *source code* pada simulasi ini dapat diakses melalui tautan *Github* berikut:

https://github.com/sidarthaprastya/Supervisory-Control-HIL