Lab 4 Report: Soft Start/Stop via PWM and Proximity Control

Course: EECS3216
Date: 2025-03-11
Name: Syed Ali Raza Rizvi - //

Introduction

This lab focuses on implementing **soft start** and **soft stop** using Pulse Width Modulation (PWM) for motor/LED control and integrating proximity sensing. Soft start/stop mechanisms reduce mechanical stress, inrush current, and voltage spikes, critical in industrial applications. Part 1 demonstrates PWM-based gradual acceleration/deceleration using an LED, while Part 2 integrates a proximity sensor (HC-SR04 ultrasonic sensor) to dynamically control the LED brightness. The demonstration was deemed acceptable by the TAs since I am using the Raspberry Pi Pico 2W microcontroller.

Objectives

- Implement PWM-based soft start/stop using interrupts (RTI).
- Integrate a proximity sensor to trigger PWM changes.
- Ensure non-blocking code execution via IRQ-based timers.

Prelab

Submitted flowcharts during the lab.

Here is reference:

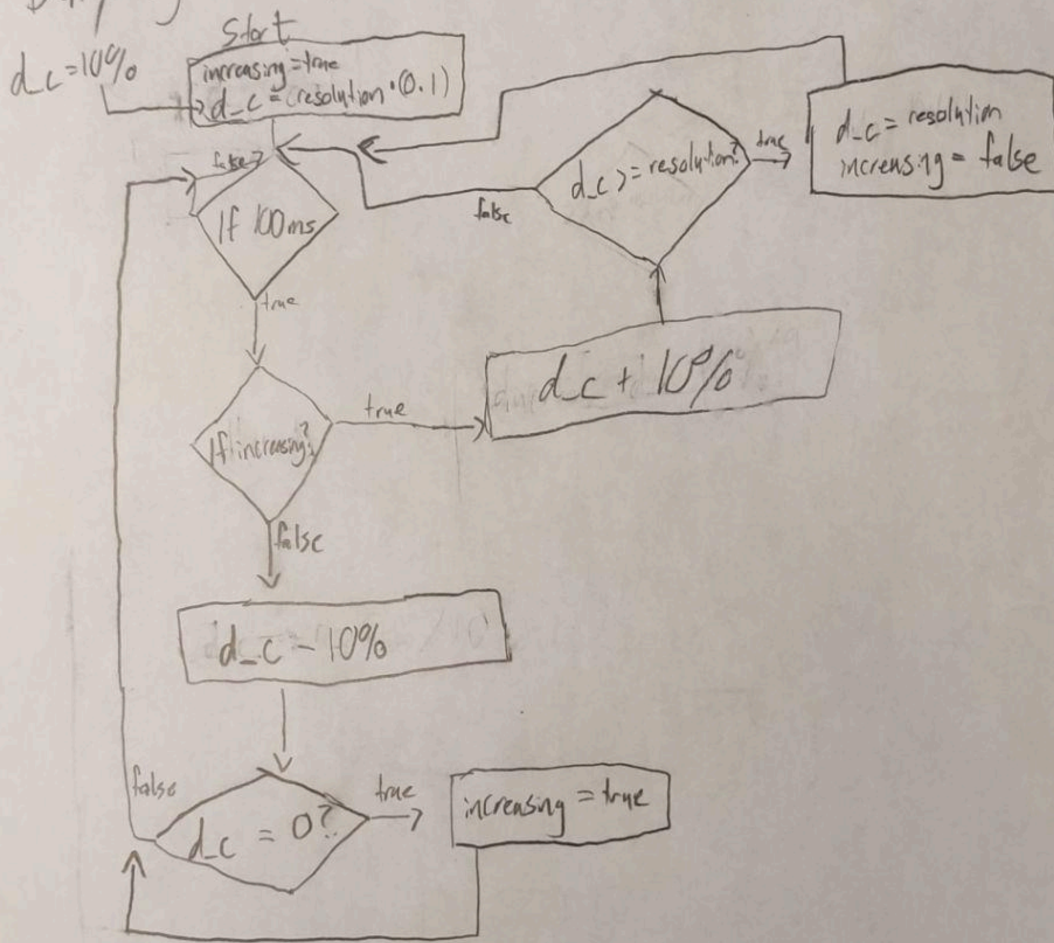
Lab-4 Prelab Flowchart

Syed Ali Raza Rizvi

Part 1.

soft start and soft stop

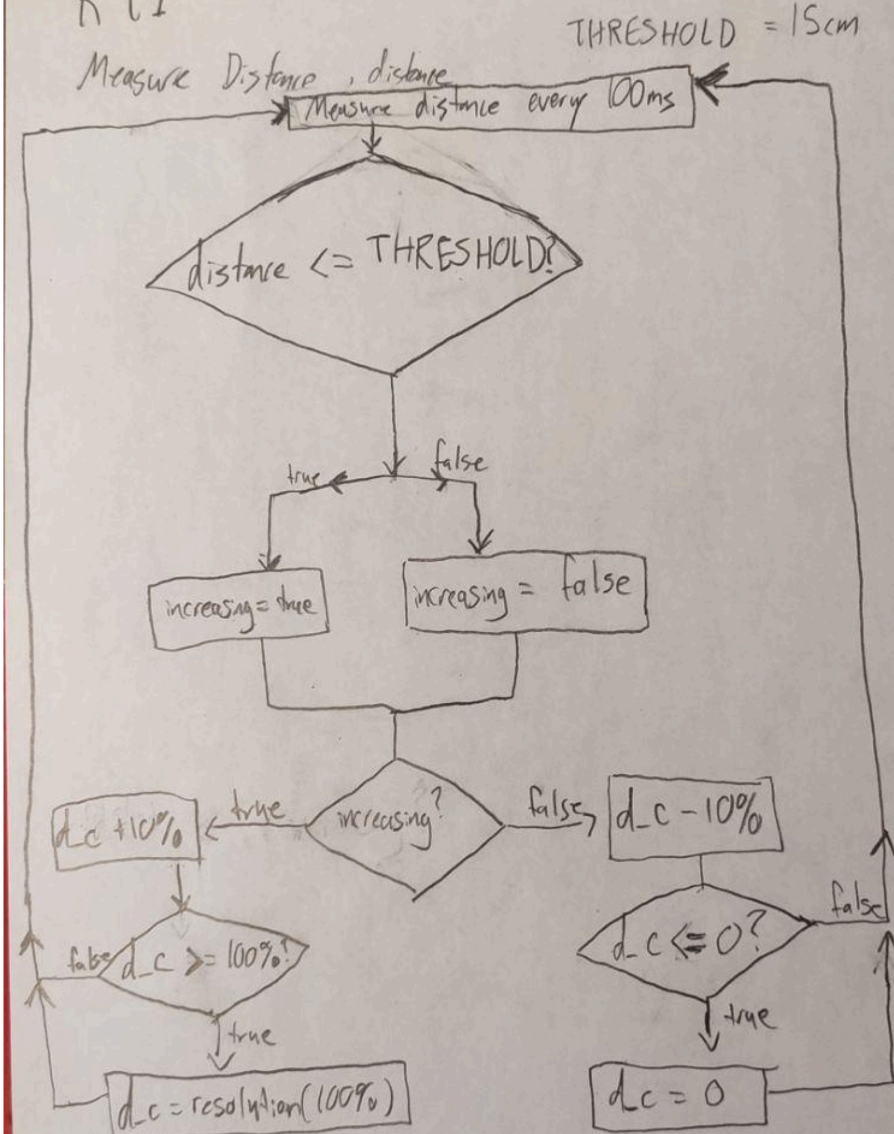
Duty cycle loop d_c^{*} = dutycycle



Part 2 Flowchart

15cm threshold for the ultrasonic sensor.

RTI



Methodology

Part 1: Soft Start/Stop with PWM

Components:

- Raspberry Pi Pico W
- LED (GPIO9)

Setup:

GPIO9 configured for PWM with 16-bit resolution, PWM_WRAP 65535.

The PWM for Part 1 uses RTI interrupts every 100ms to increment/decrement the duty cycle linearly: starting at 10%, it increases by 10% steps (soft start finished at 900ms) until reaching 100%, then decreases by 10% steps (soft stop finished at 1000ms) back to 0%, updating the LED brightness via pwm_set_chan_level.

The boolean (increasing) acts as a state flag:
increasing = true: Soft start mode — duty cycle increments by ~10% steps (ramp-up).
increasing = false: Soft stop mode — duty cycle decrements by 10% steps (ramp-down), with the flag toggling at PWM limits (100% or 0%) to reverse direction.

pwm_rti_handler() adjusts the duty cycle incrementally.

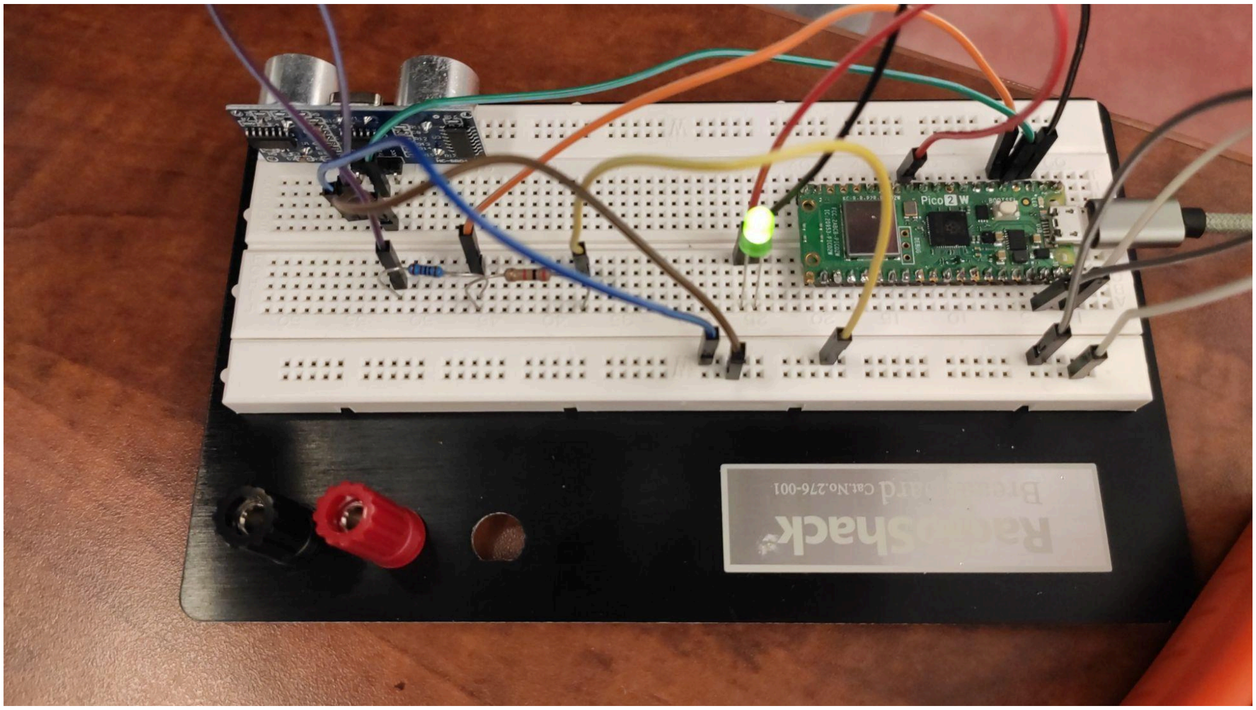
Non-blocking design using add_alarm_in_us() for scheduling, this is the timer used in Pico since it does not have an RTICTL like the HCS12 MCUs.

Part 2: Proximity-Based Control

Components:

- Raspberry Pi Pico W
- HC-SR04 Ultrasonic Sensor (Trig: GPIO2, Echo: GPIO3)
- LED (GPIO9)

Setup:



The HC-SR04 is powered through the 5V bus from the usb. The TRIG pins give 3.3V, which works with the Pico’s 3.3V logic, however the ECHO pin outputs 5V. This is problematic and can harm the Pico, so I set up a voltage divider circuit to get 3.3V. This works with R1 = 1K ohms, and R2 = 2K ohms. As seen in the picture above the orange wire takes the safe 3.3V output.

Distance is measured with a Trigger pulse sent via GPIO2. Echo pulse width measured using time_us_64().

Implemented dynamic PWM control. If distance ≤ 15 cm, soft start (LED brightens). If distance > 15 cm, soft stop (LED dims).

Results

- **Part 1:** LED brightness smoothly transitions between 10% and 100% duty cycle, confirming soft start/stop functionality.
- **Part 2:** LED brightness adjusts based on proximity:
    - Object within 15 cm → LED brightens.
    - Object beyond 15 cm → LED dims.

Discussion

- Sensor substitution, HC-SR04 replaced the lab proximity sensor, accepted by TAs
- LED soft start/stop implementation without the use of a servo was deemed acceptable by the TAs

Conclusion

This lab was done using the Pico Pi 2W using the C SDK. The lab successfully demonstrated PWM-based soft start/stop and proximity-triggered control. Key takeaways include interrupt-driven PWM design and sensor integration.

Appendix: Code

Part 1:

```
#include "pico/stdlib.h"

#include "hardware/pwm.h"

#include "hardware/timer.h"


#define LED_PIN 9         // GPIO9 for PWM output

#define PWM_WRAP 65535    // 16-bit resolution

#define RTI_INTERVAL_US 100000  // RTI interval in microseconds (100ms)


// Global Variables

volatile int duty_cycle = PWM_WRAP * 0.1;  // Start at 10% duty cycle as said in manual

volatile bool increasing = true;  // while true increase dutycycle(brightness)


// RTI Interrupt Handler

int64_t pwm_rti_handler(alarm_id_t id, void *user_data) {
```

```
    if (increasing) {

        duty_cycle += (PWM_WRAP * 0.9) / 9;  // Increment duty cycle

        if (duty_cycle >= PWM_WRAP) {

            duty_cycle = PWM_WRAP;

            increasing = false;  // Switch to soft stop mode

        }

    } else {

        duty_cycle -= PWM_WRAP / 10;  // Decrement duty cycle

        if (duty_cycle <= 0) {

            duty_cycle = 0;

            increasing = true;  // Switch back to soft start mode

        }

    }



    // Update PWM level

    pwm_set_chan_level(pwm_gpio_to_slice_num(LED_PIN),
pwm_gpio_to_channel(LED_PIN), duty_cycle);



    // Schedule next interrupt

    return RTI_INTERVAL_US;  // fix the RTI for the next cycle

}



//timer used in Pico, Pico does not have an RTICTL

void setup_rti() {
```

```
    add_alarm_in_us(RTI_INTERVAL_US, pwm_rti_handler, NULL, true);  // Schedule the first
RTI
}

int main() {
    stdio_init_all();

    // Configure LED pin for PWM
    gpio_set_function(LED_PIN, GPIO_FUNC_PWM);
    uint slice_num = pwm_gpio_to_slice_num(LED_PIN);
    uint channel = pwm_gpio_to_channel(LED_PIN);

    // Configure PWM
    pwm_config config = pwm_get_default_config();
    pwm_config_set_clkdiv(&config, 4.0);  // Adjust clock divider
    pwm_config_set_wrap(&config, PWM_WRAP);
    pwm_init(slice_num, &config, true);

    // Set up the Real-Time Interrupt (RTI)
    setup_rti();

    while (1) {
        sleep_ms(1000);
    }
```

```
}
```

Part 2:

```
#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "hardware/timer.h"

#define TRIG_PIN 2  // Trigger Pin for Ultrasonic Sensor
#define ECHO_PIN 3  // Echo Pin for Ultrasonic Sensor
#define LED_PIN 9   // PWM Output (For LED or Servo)
#define PWM_WRAP 65535
#define DISTANCE_THRESHOLD 15  // 15cm threshold

// Function to measure distance
float get_distance() {
    uint64_t start_time, end_time;

    // Send a 10micro second pulse to trigger the sensor
    gpio_put(TRIG_PIN, 1);
    sleep_us(10);
    gpio_put(TRIG_PIN, 0);

    // Wait for ECHO to go HIGH
```

```c
    while (gpio_get(ECHO_PIN) == 0);
    start_time = time_us_64();

    // Wait for ECHO to go LOW
    while (gpio_get(ECHO_PIN) == 1);
    end_time = time_us_64();

    // Calculate pulse duration and convert to distance (cm)
    float distance = (end_time - start_time) * 0.0343 / 2;
    return distance;
}

volatile int duty_cycle = PWM_WRAP * 0.1;  // Start at 10% duty cycle
volatile bool increasing = false;

// RTI Interrupt Handler for PWM Soft Start/Stop
int64_t pwm_rti_handler(alarm_id_t id, void *user_data) {
    float distance = get_distance();

    if (distance <= DISTANCE_THRESHOLD) {
        increasing = true;  // Object in range --> Soft Start
    } else {
        increasing = false;  // Object out of range --> Soft Stop
```

```
    }

    if (increasing) {
        duty_cycle += (PWM_WRAP * 0.9) / 9;
        if (duty_cycle >= PWM_WRAP) duty_cycle = PWM_WRAP;
    } else {
        duty_cycle -= PWM_WRAP / 10;
        if (duty_cycle <= 0) duty_cycle = 0;
    }

    pwm_set_chan_level(pwm_gpio_to_slice_num(LED_PIN),
pwm_gpio_to_channel(LED_PIN), duty_cycle);
    return 100000;  // Re-trigger RTI every 100ms
}

// Function to Initialize RTI
void setup_rti() {
    add_alarm_in_us(100000, pwm_rti_handler, NULL, true);
}

int main() {
    stdio_init_all();

    // Configure Ultrasonic Sensor Pins
```

```
    gpio_init(TRIG_PIN);
    gpio_set_dir(TRIG_PIN, GPIO_OUT);
    gpio_init(ECHO_PIN);
    gpio_set_dir(ECHO_PIN, GPIO_IN);

    // Configure LED/Servo for PWM
    gpio_set_function(LED_PIN, GPIO_FUNC_PWM);
    uint slice_num = pwm_gpio_to_slice_num(LED_PIN);
    uint channel = pwm_gpio_to_channel(LED_PIN);

    //configure pwm
    pwm_config config = pwm_get_default_config();
    pwm_config_set_clkdiv(&config, 4.0);
    pwm_config_set_wrap(&config, PWM_WRAP);
    pwm_init(slice_num, &config, true);

    // Start RTI for soft start/stop
    setup_rti();

    while (1) {
        sleep_ms(1000);  // Keep the program running
    }
}
```