MINI – PROJECT

Aim: Generate a PWM signals using RF controller

- **Components**

  1. Raspberry PI 3B+
  2. GPIO extension board
  3. RC Controller (FlySky FS - CT6B)
  4. LED (x4)
  5. Resistor (x10 / x4)
  6. Jumper Cables
  7. Oscilloscope
  8. Breadboard
  9. Micro – USB cable

- **Theory**

  - **Raspberry Pi 3B+**
    A single-board computer with built-in Wi-Fi, Bluetooth, and a 40-pin GPIO header, suitable for various IoT, robotics, and automation projects.

  - **GPIO Extension Board**
    A breakout board that expands and labels the GPIO pins of Raspberry Pi, making it easier to connect components without damaging the main board.

  - **RC Controller (FlySky FS - CT6B)**
    A 6-channel radio frequency controller commonly used for remote-controlled drones and robotic systems for wireless manual input.

  - **LED (x4)**
    Light-emitting diodes that emit light when powered, used as output indicators in electronic circuits.

  - **Resistor (x10 / x4)**
    10 resistors of 220 ohms used to limit current for LEDs, and 4 resistors of 2.2k ohms used for signal conditioning or voltage division in input lines.

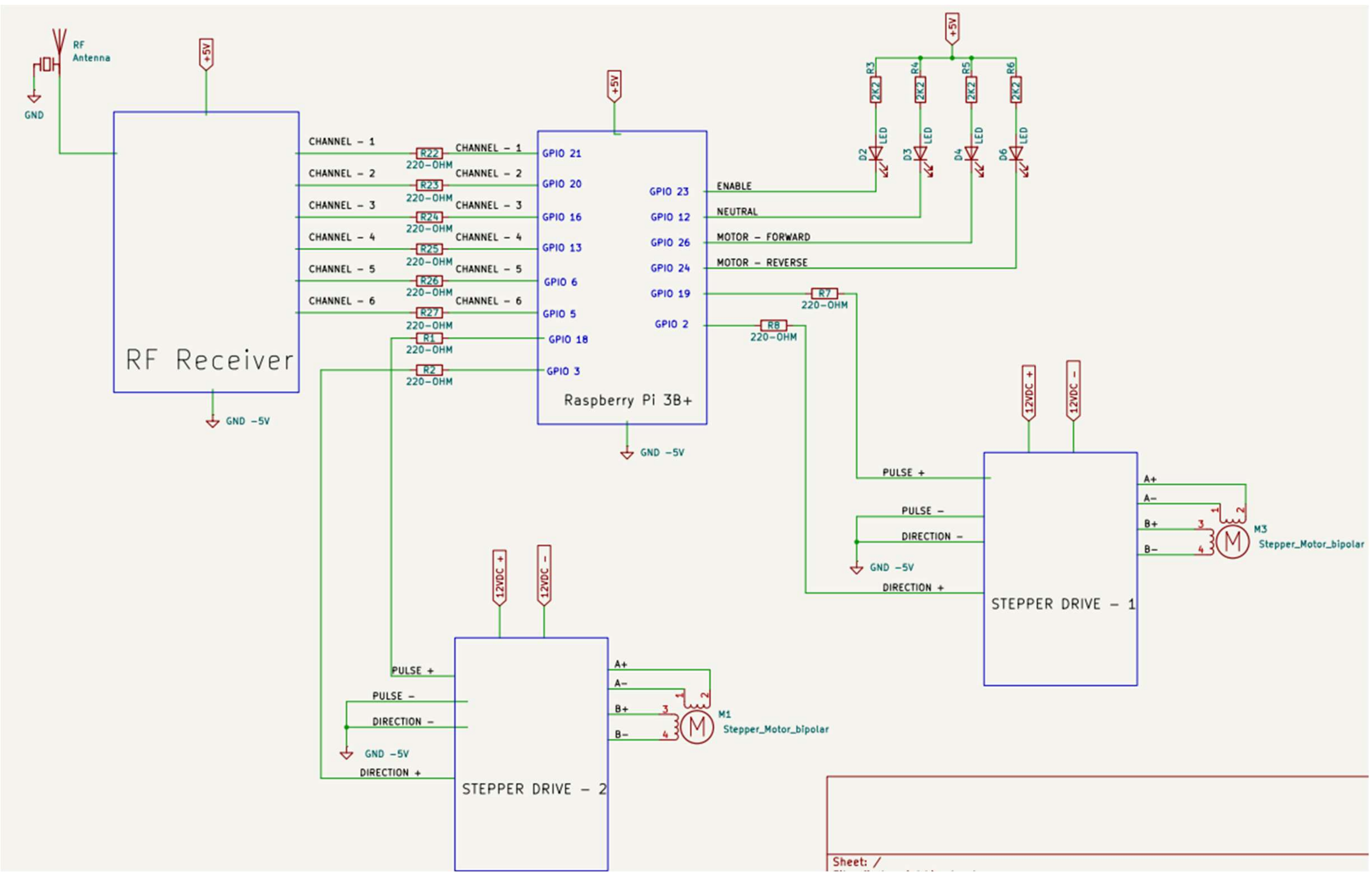
  - **Jumper Cables**
    Flexible wires with male/female connectors used to establish temporary electrical connections between components on a breadboard or GPIO.

  - **Oscilloscope**
    An instrument used to visualize and analyze voltage waveforms over time, helpful for debugging PWM signals or analog input.

- **Breadboard**
  A solderless platform for prototyping circuits, allowing quick assembly and testing of electronic connections.

- **Micro–USB Cable**
  Used to power the Raspberry Pi 3B+ and also for data transfer or communication in some setups.

- **Circuit Diagram**



Procedure to Demonstrate PWM Signal Control for Solar Panel Cleaner Robot using Raspberry Pi and RF Controller

Step 1: Add Components to the Workspace

1. Place the Raspberry Pi 3B+ on the workspace or mounting base.
2. Connect the GPIO extension board to the Raspberry Pi for easier access to pins.
3. Position a breadboard near the GPIO extension.
4. Insert 4 LEDs into the breadboard to act as indicators.
5. Connect 10 resistors (220 ohms) in series with each LED.
6. Attach 2.2k ohm resistors (x4) where required for voltage division or signal protection.
7. Connect the FlySky FS - CT6B RC controller receiver module to the GPIO pins using jumper cables.
8. Wire the oscilloscope probes to the PWM output pin coming from the RC controller's receiver.
9. Connect the stepper motors (used in the real robot) to driver modules (e.g., ULN2003 or A4988), and connect driver inputs to Raspberry Pi.
10. Use jumper cables to connect GPIO pins to the appropriate control pins of the stepper driver modules.
11. Power the Raspberry Pi using a micro–USB cable connected to a 5V/3A power source.

Step 2: Build the Circuit Connections

Power Supply Connections:

1. Connect 5V and GND from Raspberry Pi to the power rails on the breadboard.
2. Connect the RC receiver VCC and GND to 5V and GND from Raspberry Pi.
3. Connect stepper motor driver module power inputs to the Raspberry Pi's 5V and GND rails.

Signal Connections:

1. Connect RC receiver PWM signal pins (usually CH1 to CH6) to Raspberry Pi GPIO pins (like GPIO18, GPIO19 etc.).
2. Connect the oscilloscope probes to one of the PWM GPIO pins to visualize the signal.

3. Connect the Raspberry Pi GPIO pins to the stepper motor driver inputs (IN1, IN2, IN3, IN4).

LED Connections:

1. Connect each LED anode to a GPIO pin via a 220Ω resistor.
2. Connect all LED cathodes to GND rail.

Step 3: Simulate the Circuit

1. Power on the Raspberry Pi using the micro–USB cable.
2. Turn ON the FlySky FS - CT6B controller and ensure the receiver is paired.
3. Move the throttle or channel sticks on the controller.
4. Observe the PWM signal pattern changing live on the oscilloscope screen connected to the selected GPIO pin.
5. Observe the stepper motors responding to the PWM signals by rotating (or simulate this if motors aren't connected for demo).
6. Use the LEDs to indicate signal activity or logic status during demonstration.

Step 4: Code

```
import pigpio
import time
import threading
import sys
import traceback
import os
PULSE_MIN = 1100
PULSE_NEUTRAL_LOW = 1450
PULSE_NEUTRAL_HIGH = 1550
PULSE_MAX = 1900
MAX_FORWARD_SPS = 1500
MAX_PIVOT_SPS = 500
STEERING_SENSITIVITY = 1.0
SPS_STEP = 25
RAMP_DOWN_DELAY = 0.012
UPDATE_DELAY = 0.025
SPS_RAMP_THRESHOLD = SPS_STEP / 2
STEP_DUTY_CYCLE = 500000
INITIAL_WAIT_TIMEOUT = 5.0
PRINT_INTERVAL = 20
SIGNAL_TIMEOUT = 1.5
PIN_THROTTLE_INPUT = 16
```

```
PIN_STEERING_INPUT = 21
PIN_MOTOR_L_STEP = 19
PIN_MOTOR_L_DIR  = 2
PIN_MOTOR_R_STEP = 18
PIN_MOTOR_R_DIR  = 3
PIN_MOTOR_ENABLE = 4
ENABLE_PIN_LOGIC_HIGH_IS_DISABLED = True
ENABLED_LED = 23
NEUTRAL_LED = 12
FORWARD_LED = 26
REVERSE_LED = 24
pi = None
terminate_flag = threading.Event()
last_signal_time = time.time()
drivers_enabled_state = False
last_read_ch3_throttle_us = None
last_read_ch1_steering_us = None
_ch3_rise_tick = None
_ch1_rise_tick = None
def _pulse_callback(gpio, level, tick, channel_name, rise_tick_global, pulse_us_global):
    global last_signal_time, last_read_ch3_throttle_us, last_read_ch1_steering_us
    rise_tick = globals().get(rise_tick_global)
    current_time = time.time()
    if level == 1:
        globals()[rise_tick_global] = tick
    elif level == 0:
        if rise_tick is not None:
            width = pigpio.tickDiff(rise_tick, tick)
            if PULSE_MIN * 0.8 <= width <= PULSE_MAX * 1.2:
                alpha = 0.5
                current_val = globals().get(pulse_us_global)
                smoothed_width = width if current_val is None else int(alpha * width + (1 - alpha) *
current_val)
                globals()[pulse_us_global] = smoothed_width
                last_signal_time = current_time
            globals()[rise_tick_global] = None
    elif level == 2:
        print(f"[Callback Timeout] Pulses stopped on {channel_name} (GPIO {gpio})")
        globals()[rise_tick_global] = None
def _callback_ch3(gpio, level, tick):
    _pulse_callback(gpio, level, tick, "CH3", "_ch3_rise_tick", "last_read_ch3_throttle_us")
def _callback_ch1(gpio, level, tick):
    _pulse_callback(gpio, level, tick, "CH1", "_ch1_rise_tick", "last_read_ch1_steering_us")
def watchdog_thread():
    global pi
    print("[Watchdog] Thread started.")
    pin_list_for_cleanup = [ PIN_MOTOR_L_DIR, PIN_MOTOR_R_DIR, ENABLED_LED,
```

```
NEUTRAL_LED, FORWARD_LED, REVERSE_LED ]
    if PIN_MOTOR_ENABLE is not None:
        pin_list_for_cleanup.append(PIN_MOTOR_ENABLE)
    while not terminate_flag.is_set():
        time_since_last_signal = time.time() - last_signal_time
        if time_since_last_signal > SIGNAL_TIMEOUT:
            print(f"\n[!! WATCHDOG TRIGGERED !!] No RC signal for {time_since_last_signal:.2f}s.
Shutting down.")
            terminate_flag.set()
            if pi and pi.connected:
                try:
                    print("[Watchdog] Stopping STEP pulse generation...")
                    pi.hardware_PWM(PIN_MOTOR_L_STEP, 0, 0)
                    pi.hardware_PWM(PIN_MOTOR_R_STEP, 0, 0)
                    print("[Watchdog] STEP pulse generation stopped.")
                    if PIN_MOTOR_ENABLE is not None:
                        disable_level = 1 if ENABLE_PIN_LOGIC_HIGH_IS_DISABLED else 0
                        print(f"[Watchdog] Disabling motor drivers via ENA Pin ({PIN_MOTOR_ENABLE})...")
                        try: pi.write(PIN_MOTOR_ENABLE, disable_level)
                        except Exception: pass
                    print("[Watchdog] Setting DIR/LED Outputs LOW...")
                    dir_led_pins = [ p for p in pin_list_for_cleanup if p != PIN_MOTOR_ENABLE ]
                    for pin in dir_led_pins:
                            try:
                                if pi.get_mode(pin) == pigpio.OUTPUT: pi.write(pin, 0)
                            except Exception: pass
                    print("[Watchdog] DIR/LED Outputs LOW.")
                except Exception as e_wd:
                        print(f"[Watchdog] Error during pigpio cleanup: {e_wd}")
            else:
                print("[Watchdog] pigpio not connected, cannot perform hardware cleanup.")
            print("[!! WATCHDOG] EXITING SCRIPT FORCEFULLY.")
            os._exit(1)
        time.sleep(0.1)
    print("[Watchdog] Thread exiting normally.")
def map_pulse_to_range(pulse, min_p, n_low, n_high, max_p, target_r):
    if pulse is None: return 0.0
    if n_low <= pulse <= n_high: return 0.0
    elif pulse < n_low: return -target_r * ((n_low - max(pulse, min_p)) / (n_low - min_p))
    else: return target_r * ((min(pulse, max_p) - n_high) / (max_p - n_high))
def get_target_values(pulse_for_throttle_logic, pulse_for_steering_logic):
    neutral_pulse = (PULSE_NEUTRAL_LOW + PULSE_NEUTRAL_HIGH) // 2
    if pulse_for_throttle_logic is None: pulse_for_throttle_logic = neutral_pulse
    if pulse_for_steering_logic is None: pulse_for_steering_logic = neutral_pulse
    throttle = map_pulse_to_range(pulse_for_throttle_logic, PULSE_MIN, PULSE_NEUTRAL_LOW,
PULSE_NEUTRAL_HIGH, PULSE_MAX, 1.0)
    steering = map_pulse_to_range(pulse_for_steering_logic, PULSE_MIN, PULSE_NEUTRAL_LOW,
```

```
PULSE_NEUTRAL_HIGH, PULSE_MAX, 1.0)
    base_sps = abs(throttle) * MAX_FORWARD_SPS
    base_dir = 1 if throttle > 0 else 0 if throttle < 0 else None
    target_l_sps, target_r_sps = 0, 0
    target_l_dir, target_r_dir = None, None
    if base_dir is not None:
        turn_factor_r = 1.0 - (steering * STEERING_SENSITIVITY)
        turn_factor_l = 1.0 + (steering * STEERING_SENSITIVITY)
        target_l_sps = max(0, min(base_sps * turn_factor_l, MAX_FORWARD_SPS))
        target_r_sps = max(0, min(base_sps * turn_factor_r, MAX_FORWARD_SPS))
        target_l_dir, target_r_dir = base_dir, base_dir
    elif steering != 0:
        pivot_sps = abs(steering) * MAX_PIVOT_SPS
        target_l_sps, target_r_sps = pivot_sps, pivot_sps
        if steering < 0: target_l_dir, target_r_dir = 0, 1
        else: target_l_dir, target_r_dir = 1, 0
    return int(target_l_sps), target_l_dir, int(target_r_sps), target_r_dir, base_dir
def update_single_motor(target_sps, target_dir, current_sps, current_dir_pin_state, dir_pin, step_pin):
    global pi
    new_sps = current_sps
    new_dir_state = current_dir_pin_state
    needs_dir_change = (target_dir is not None and target_dir != current_dir_pin_state)
    if needs_dir_change and target_sps > 0:
        if new_sps > 0:
            ramp_step = SPS_STEP * 1.5
            while new_sps > 0:
                new_sps = max(0, new_sps - ramp_step)
                try: pi.hardware_PWM(step_pin, int(new_sps), STEP_DUTY_CYCLE if new_sps > 0 else 0)
                except Exception as e:
                    print(f"WARN: PWM fail during dir ramp down {step_pin}: {e}")
                    try: pi.hardware_PWM(step_pin, 0, 0)
                    except Exception: pass
                    new_sps = 0; break
                time.sleep(RAMP_DOWN_DELAY / 4)
        try:
            pi.hardware_PWM(step_pin, 0, 0); new_sps = 0
        except Exception as e: print(f"WARN: PWM stop fail pin {step_pin} before dir change: {e}")
        time.sleep(RAMP_DOWN_DELAY)
        try:
            pi.write(dir_pin, target_dir); new_dir_state = target_dir
        except Exception as e: print(f"ERROR: Failed writing direction pin {dir_pin}: {e}")
        time.sleep(RAMP_DOWN_DELAY)
    sps_diff = target_sps - new_sps
    if abs(sps_diff) > SPS_RAMP_THRESHOLD:
        if target_sps > new_sps: new_sps = min(target_sps, new_sps + SPS_STEP)
        elif target_sps < new_sps: new_sps = max(0, new_sps - SPS_STEP)
    elif target_sps == 0 and new_sps > 0:
```

```
        new_sps = max(0, new_sps - SPS_STEP)
    active_sps = int(max(0, new_sps))
    active_duty = STEP_DUTY_CYCLE if active_sps > 0 else 0
    pwm_freq_to_set = active_sps if active_sps > 0 else 1
    try:
        pi.hardware_PWM(step_pin, pwm_freq_to_set, active_duty)
    except pigpio.error as e:
        if "GPIO not 12, 13, 18 or 19" in str(e):
            print(f"\nFATAL ERROR: Pin {step_pin} is NOT HW PWM!"); terminate_flag.set()
        else: print(f"ERROR: hardware_PWM set fail pin {step_pin}: {e}")
        try: pi.hardware_PWM(step_pin, 0, 0)
        except Exception: pass
        new_sps = 0
    return int(new_sps), new_dir_state
if _name_ == "_main_":
    cb_th = None; cb_st = None
    current_sps_left = 0; current_sps_right = 0
    current_dir_left_state = 0; current_dir_right_state = 0
    watchdog = None
    try:
        print("Connecting to pigpio daemon...")
        pi = pigpio.pi()
        if not pi.connected:
            print("ERROR: Failed to connect to pigpio daemon."); sys.exit(1)
        print("Connected to pigpiod.")
        print(f"Setting up Input Pins: {PIN_THROTTLE_INPUT}(Thr/CH3),
{PIN_STEERING_INPUT}(Ste/CH1)")
        pi.set_mode(PIN_THROTTLE_INPUT, pigpio.INPUT);
pi.set_pull_up_down(PIN_THROTTLE_INPUT, pigpio.PUD_DOWN)
        pi.set_mode(PIN_STEERING_INPUT, pigpio.INPUT);
pi.set_pull_up_down(PIN_STEERING_INPUT, pigpio.PUD_DOWN)
        print(f"Setting up Output Pins: DIRs({PIN_MOTOR_L_DIR},{PIN_MOTOR_R_DIR}),
LEDs(...)")
        dir_led_pins = [PIN_MOTOR_L_DIR, PIN_MOTOR_R_DIR, ENABLED_LED, NEUTRAL_LED,
FORWARD_LED, REVERSE_LED]
        for pin in dir_led_pins:
            try: pi.set_mode(pin, pigpio.OUTPUT); pi.write(pin, 0)
            except Exception as e: print(f"WARN: Failed setup output pin {pin}: {e}")
        if PIN_MOTOR_ENABLE is not None:
            print(f"Setting up Motor Enable Pin: {PIN_MOTOR_ENABLE}")
            try:
                pi.set_mode(PIN_MOTOR_ENABLE, pigpio.OUTPUT)
                disable_level = 1 if ENABLE_PIN_LOGIC_HIGH_IS_DISABLED else 0
                pi.write(PIN_MOTOR_ENABLE, disable_level)
                drivers_enabled_state = False
                print(f"  Drivers initially DISABLED (Pin {PIN_MOTOR_ENABLE} set to {disable_level})")
            except Exception as e:
```

```
            print(f"WARN: Failed to setup motor enable pin {PIN_MOTOR_ENABLE}: {e}");
PIN_MOTOR_ENABLE = None
        else: print("Motor Enable Pin control DISABLED.")
        print("\n" + "="*10 + " CRITICAL STEP PIN CONFIGURATION " + "="*10)
        print(f" STEP Left Pin (Motor 1): {PIN_MOTOR_L_STEP} (GPIO 19 - HW PWM OK)")
        print(f" STEP Right Pin (Motor 2): {PIN_MOTOR_R_STEP} (GPIO 18 - HW PWM OK)")
        print("="*50 + "\n")
        try:
            pi.set_mode(PIN_MOTOR_L_STEP, pigpio.OUTPUT); pi.write(PIN_MOTOR_L_STEP, 0)
            pi.set_mode(PIN_MOTOR_R_STEP, pigpio.OUTPUT); pi.write(PIN_MOTOR_R_STEP, 0)
        except Exception as e: print(f"FATAL ERROR: Could not set initial state for STEP pins: {e}");
sys.exit(1)
        print("GPIO Setup Complete.")
        print("Setting up input callbacks...")
        last_signal_time = time.time()
        cb_th = pi.callback(PIN_THROTTLE_INPUT, pigpio.EITHER_EDGE, _callback_ch3)
        cb_st = pi.callback(PIN_STEERING_INPUT, pigpio.EITHER_EDGE, _callback_ch1)
        pi.set_watchdog(PIN_THROTTLE_INPUT, 500); pi.set_watchdog(PIN_STEERING_INPUT, 500)
        print("Callbacks registered.")
        print(f"\nWaiting for initial RC signals ({INITIAL_WAIT_TIMEOUT}s)...")
        timeout_start = time.time()
        while (last_read_ch3_throttle_us is None or last_read_ch1_steering_us is None) and \
            (time.time() - timeout_start < INITIAL_WAIT_TIMEOUT):
            if terminate_flag.is_set(): break
            time.sleep(0.05)
        print("\nInitial Signal Check:")
        neutral_pulse = (PULSE_NEUTRAL_LOW + PULSE_NEUTRAL_HIGH) // 2
        if last_read_ch3_throttle_us is None: last_read_ch3_throttle_us = neutral_pulse; print(f"  WARN:
No CH3 signal!")
        if last_read_ch1_steering_us is None: last_read_ch1_steering_us = neutral_pulse; print(f"  WARN:
No CH1 signal!")
        print(f"  Using Initial Pulses: CH3={last_read_ch3_throttle_us} us,
CH1={last_read_ch1_steering_us} us")
        print("\nCalculating Initial Motor States...")
        init_sps_l, init_dir_l, init_sps_r, init_dir_r, _ = get_target_values(last_read_ch3_throttle_us,
last_read_ch1_steering_us)
        current_dir_left_state = init_dir_l if init_dir_l is not None else 0
        pi.write(PIN_MOTOR_L_DIR, current_dir_left_state)
        current_dir_right_state = init_dir_r if init_dir_r is not None else 0
        pi.write(PIN_MOTOR_R_DIR, current_dir_right_state)
        print(f"  Initial Dirs: L={current_dir_left_state}, R={current_dir_right_state}")
        if PIN_MOTOR_ENABLE is None:
            print("Initializing Step Pulses (Enable Pin not used)...")
            current_sps_left = init_sps_l if init_sps_l > 0 else 0
            initial_duty_l = STEP_DUTY_CYCLE if current_sps_left > 0 else 0; initial_freq_l =
current_sps_left if current_sps_left > 0 else 1
            pi.hardware_PWM(PIN_MOTOR_L_STEP, initial_freq_l, initial_duty_l)
```

```
            current_sps_right = init_sps_r if init_sps_r > 0 else 0
            initial_duty_r = STEP_DUTY_CYCLE if current_sps_right > 0 else 0; initial_freq_r =
current_sps_right if current_sps_right > 0 else 1
            pi.hardware_PWM(PIN_MOTOR_R_STEP, initial_freq_r, initial_duty_r)
            print(f"  Initial SPS: L={current_sps_left}, R={current_sps_right}")
            drivers_enabled_state = (current_sps_left > 0 or current_sps_right > 0)
        else:
            print("Step Pulses will initialize on first move command (Enable Pin is used).")
            current_sps_left = 0; current_sps_right = 0
            pi.hardware_PWM(PIN_MOTOR_L_STEP, 0, 0); pi.hardware_PWM(PIN_MOTOR_R_STEP, 0,
0)
        print("\nStarting Watchdog Thread...");
        watchdog = threading.Thread(target=watchdog_thread, daemon=True); watchdog.start()
        print("\n" + "="*30 + "\nInit Complete. Entering Main Loop...\n" + "="*30 + "\n")
        last_update_time = time.time(); loop_count = 0
        while not terminate_flag.is_set():
            loop_start_time = time.time()
            current_ch3 = last_read_ch3_throttle_us; current_ch1 = last_read_ch1_steering_us
            target_sps_l, target_dir_l, target_sps_r, target_dir_r, base_dir = get_target_values(current_ch3,
current_ch1)
            should_be_enabled = (target_sps_l > 0 or target_sps_r > 0)
            if PIN_MOTOR_ENABLE is not None:
                if should_be_enabled and not drivers_enabled_state:
                    enable_level = 0 if ENABLE_PIN_LOGIC_HIGH_IS_DISABLED else 1
                    pi.write(PIN_MOTOR_ENABLE, enable_level)
                    drivers_enabled_state = True
                    time.sleep(0.002)
                elif not should_be_enabled and drivers_enabled_state:
                    disable_level = 1 if ENABLE_PIN_LOGIC_HIGH_IS_DISABLED else 0
                    pi.write(PIN_MOTOR_ENABLE, disable_level)
                    drivers_enabled_state = False
                    try: pi.hardware_PWM(PIN_MOTOR_L_STEP, 0, 0)
                    except Exception: pass
                    try: pi.hardware_PWM(PIN_MOTOR_R_STEP, 0, 0)
                    except Exception: pass
                    current_sps_left = 0; current_sps_right = 0
            current_time = time.time()
            if (current_time - last_update_time >= UPDATE_DELAY) and \
            (PIN_MOTOR_ENABLE is None or drivers_enabled_state or should_be_enabled):
                if PIN_MOTOR_ENABLE is not None and should_be_enabled and not drivers_enabled_state:
                    current_sps_left = 0; current_sps_right = 0

                current_sps_left, current_dir_left_state = update_single_motor(
                    target_sps_l, target_dir_l, current_sps_left, current_dir_left_state,
                    PIN_MOTOR_L_DIR, PIN_MOTOR_L_STEP)
                current_sps_right, current_dir_right_state = update_single_motor(
                    target_sps_r, target_dir_r, current_sps_right, current_dir_right_state,
```

```
                PIN_MOTOR_R_DIR, PIN_MOTOR_R_STEP)
            last_update_time = current_time
            is_moving = current_sps_left > 0 or current_sps_right > 0
            pi.write(ENABLED_LED, 1 if is_moving else 0)
            pi.write(NEUTRAL_LED, 1 if not is_moving else 0)
            pi.write(FORWARD_LED, 1 if (is_moving and base_dir == 1) else 0)
            pi.write(REVERSE_LED, 1 if (is_moving and base_dir == 0) else 0)
        elif PIN_MOTOR_ENABLE is not None and not drivers_enabled_state:
            pi.write(ENABLED_LED, 0); pi.write(NEUTRAL_LED, 1)
            pi.write(FORWARD_LED, 0); pi.write(REVERSE_LED, 0)
        if loop_count % PRINT_INTERVAL == 0:
            dl='F'if current_dir_left_state==1 else'R'; dr='F'if current_dir_right_state==1 else'R'
            tl='F'if target_dir_l==1 else'R'if target_dir_l==0 else'-'; tr='F'if target_dir_r==1 else'R'if
target_dir_r==0 else'-'
            tp_ch3_str = f"{current_ch3:<5}" if current_ch3 is not None else "N/A  "
            sp_ch1_str = f"{current_ch1:<5}" if current_ch1 is not None else "N/A  "
            ts_str = f"{time.time()-last_signal_time:.2f}s" if last_signal_time else "N/A"
            ena_str = f"ENA:{'On ' if drivers_enabled_state else 'Off'}" if PIN_MOTOR_ENABLE is not
None else "ENA:N/A"

            print(f"--- {time.strftime('%H:%M:%S.%f')[:-3]} (Sig Age: {ts_str}) {ena_str} V11.4 ---")
            print(f" Pulse Read: Thr({PIN_THROTTLE_INPUT})={tp_ch3_str} |
Ste({PIN_STEERING_INPUT})={sp_ch1_str}")
            print(f" Target SPS: L({PIN_MOTOR_L_STEP})={target_sps_l:<5} ({tl}) |
R({PIN_MOTOR_R_STEP})={target_sps_r:<5} ({tr})")
            print(f" Current SPS: L={current_sps_left:<5} ({dl}) | R={current_sps_right:<5} ({dr})")
        loop_count = (loop_count + 1) % 10000
        loop_duration = time.time() - loop_start_time
        sleep_time = max(0.001, UPDATE_DELAY - loop_duration)
        time.sleep(sleep_time)
    except KeyboardInterrupt: print("\n\n*** Ctrl+C Detected: Stopping... ***"); terminate_flag.set()
    except SystemExit: print("\n\n*** SystemExit Detected (Watchdog Force Exit?) ***")
    except Exception as e: print(f"\n\n*** FATAL ERROR in Main Loop: {e} ***");
traceback.print_exc(); terminate_flag.set()
    finally:
        print("\n" + "="*30 + "\n Initiating Final Cleanup...\n" + "="*30)
        terminate_flag.set()
        if pi and pi.connected:
            print("  Cancelling GPIO callbacks...")
             # *** SYNTAX FIX: Use try...except Exception: ***
            if cb_th:
                try: cb_th.cancel()
                except Exception: pass
             if cb_st:
                 try: cb_st.cancel()
                 except Exception: pass
             print("  Disabling callback watchdogs...")
```

```
            # *** SYNTAX FIX: Use try...except Exception: ***
            try: pi.set_watchdog(PIN_THROTTLE_INPUT, 0)
            except Exception: pass
            try: pi.set_watchdog(PIN_STEERING_INPUT, 0)
            except Exception: pass
            print("  Stopping STEP pulse generation...")
            # *** SYNTAX FIX: Use try...except Exception: ***
            try: pi.hardware_PWM(PIN_MOTOR_L_STEP, 0, 0)
            except Exception: pass
            try: pi.hardware_PWM(PIN_MOTOR_R_STEP, 0, 0)
            except Exception: pass
            if PIN_MOTOR_ENABLE is not None:
                disable_level = 1 if ENABLE_PIN_LOGIC_HIGH_IS_DISABLED else 0
                print(f"  Disabling motor drivers via ENA Pin ({PIN_MOTOR_ENABLE})...")
                # *** SYNTAX FIX: Use try...except Exception: ***
                try: pi.write(PIN_MOTOR_ENABLE, disable_level)
                except Exception as e: print(f"    WARN: Failed to set ENA pin {PIN_MOTOR_ENABLE} to
    disable state: {e}")
            time.sleep(0.05)
            print("  Setting DIR/LED/STEP output pins LOW...")
            all_output_pins = [ PIN_MOTOR_L_DIR, PIN_MOTOR_R_DIR, ENABLED_LED,
    NEUTRAL_LED, FORWARD_LED, REVERSE_LED, PIN_MOTOR_L_STEP,
    PIN_MOTOR_R_STEP ]
            for pin in all_output_pins:
               try:
                  if pi.get_mode(pin) == pigpio.OUTPUT: pi.write(pin, 0)
               except Exception: pass # Ignore final cleanup errors
            print("  Disconnecting from pigpio daemon...")
            # *** SYNTAX FIX: Use try...except Exception: ***
            try: pi.stop()
            except Exception: pass
        else:
           print("  pigpio connection not available for cleanup.")
        if watchdog and watchdog.is_alive():
           print("  Waiting for watchdog thread to exit..."); watchdog.join(1.0)
           if watchdog.is_alive(): print("    Watchdog did not exit cleanly.")
        print("\nCleanup complete.\n" + "="*30 + "\n Script Exited.\n" + "="*30 + "\n")
```

Conclusion

This setup demonstrates how a solar panel cleaner robot receives PWM signals via an RF controller, and how the Raspberry Pi processes those signals to control two stepper motors. The oscilloscope is used to visibly validate the PWM waveform, confirming successful signal transmission and GPIO-level control.