**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**

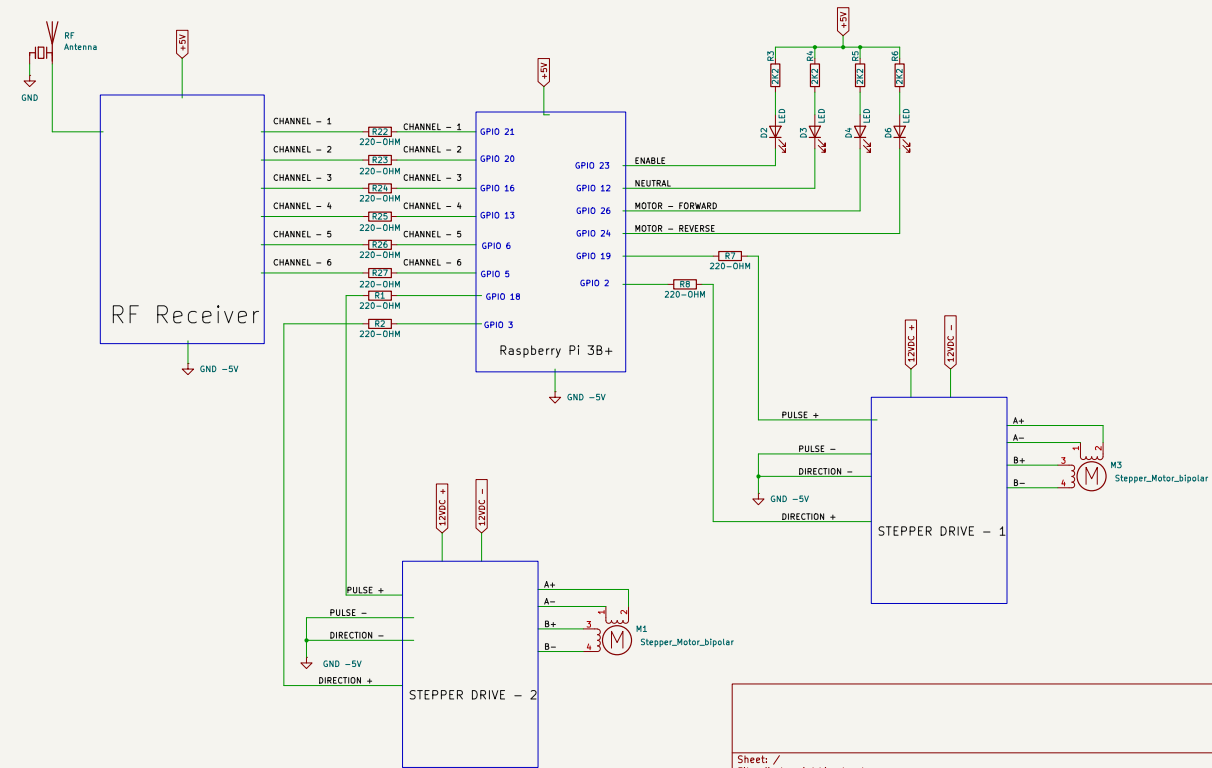
An instrument used to visualize and analyze voltage waveforms over time, helpful for debugging PWM signals or analog input.

* + **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.