# MQTT and PID

Annie Prasanna Manoharan

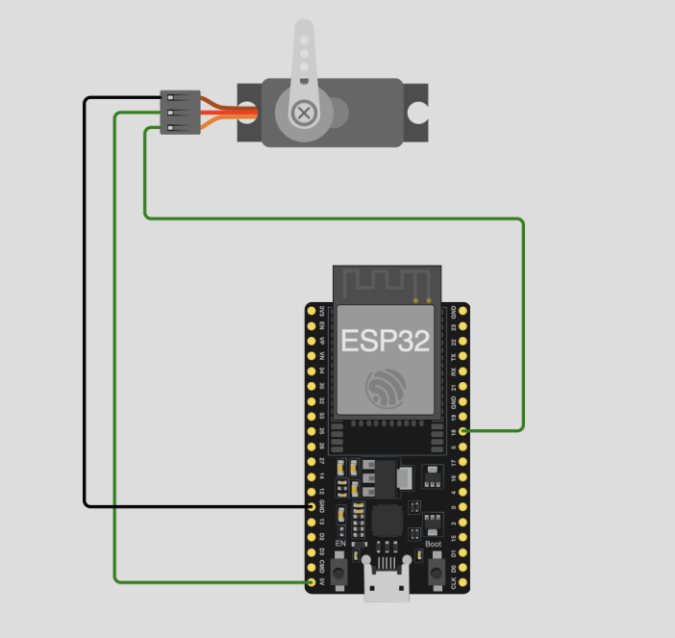
***Overview*— The MQTT and PID project implements a complete real-time control system on the ESP32 that uses FreeRTOS tasks, MQTT communication, and a PID controller to stabilize a servo actuator in the presence of random noise. The goal was to simulate a real engineering scenario where a mechanical system must be kept steady despite unpredictable disturbances. Multiple tasks operate in parallel, share data safely, and communicate sensor and actuator values to an MQTT broker for live monitoring.**

## I. Introduction

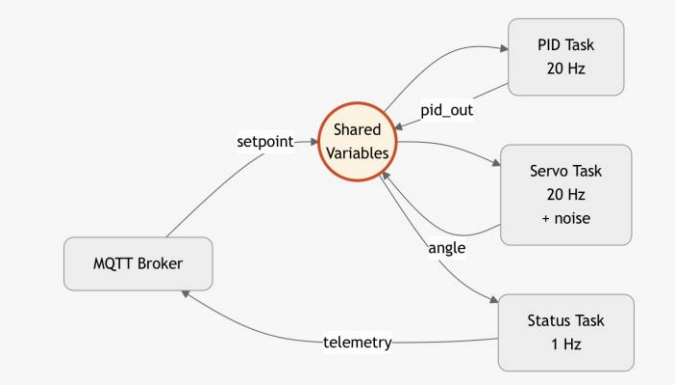
The purpose of this project was to develop a simulated PID-based actuator control system that operates in real time using the ESP32 microcontroller. The system uses FreeRTOS to manage multiple independent tasks, Wi-Fi and MQTT for remote data transmission, and a servo motor representing the actuator being stabilized. A stream of random noise simulates environmental disturbances, and the PID controller continuously adjusts the actuator to compensate for these fluctuations. The MQTT dashboard is used to visualize sensor readings, noise levels, and actuator output. The project emphasizes real-time multitasking, synchronized variable sharing, and reliable communication with a cloud MQTT broker.

## II. Application Design

The application is designed around several FreeRTOS tasks that run concurrently and share important variables through protected access. The Wi-Fi/MQTT task is responsible for establishing a stable network connection, maintaining the link with the MQTT broker, publishing sensor data, servo angles, noise values, and PID output, and receiving any possible user-set setpoints. A noise generation task continuously creates random disturbances that are stored in a shared variable. A sensor task reads the effective sensor value by combining the system setpoint with the generated noise. The PID controller task then processes that sensor value to compute the control signal using proportional, integral, and derivative gains. Its output determines the servo position, which is applied by the servo actuator task and simultaneously published to MQTT. Mutexes ensure that shared variables—such as noise, sensor readings, PID terms, and servo output—are accessed safely across tasks. The design ensures smooth coordination between tasks, prevents race conditions, and maintains a consistent real-time control loop.




The system operates as a networked closed-loop control simulation where the main Arduino loop manages MQTT connectivity to receive external setpoint commands into a shared global variable. The control logic is distributed across three concurrent FreeRTOS tasks: the **Servo Task** (20 Hz) simulates the physical "plant" by driving the servo motor based on the sum of the external MQTT setpoint, injected noise, and corrective PID efforts; the **PID Task** (20 Hz) functions as the controller, reading the current servo angle and calculating a pid_output to actively drive the system toward a fixed 90-degree target; and the **Status Task** (1 Hz) acts as a telemetry logger, reading the shared state variables and publishing the current angle and error metrics back to the MQTT broker for remote monitoring.




## III. Results

The system ran successfully on the ESP32 and performed well across all tests. The Wi-Fi and MQTT connections were stable,

and the device continuously published sensor data, noise, and servo outputs to HiveMQ in real time. With moderate noise, the PID controller stabilized the actuator quickly and accurately, showing minimal overshoot after tuning. Higher noise levels caused more initial fluctuation, but the controller still converged to the target value, demonstrating effective noise rejection. The derivative term played an important role in reducing sudden spikes, and limiting the integral term prevented wind-up. Wi-Fi dropout simulations were also tested, and the MQTT task successfully reconnected without interrupting the main control logic. Overall, the system functioned as expected and passed all example tests.

## IV. Conclusion

This project successfully integrated real-time operating system concepts, MQTT communication, and PID control into a single embedded application. It demonstrated how multiple tasks can work cooperatively on the ESP32 to simulate a realistic control system where noisy sensor inputs must be counteracted with continuous adjustments. The results confirmed that the PID controller-maintained stability under varying noise conditions and that MQTT provided reliable remote visualization of system behavior. The assignment strengthened understanding of real-time multitasking, shared-variable synchronization, and closed-loop control systems, and the final implementation met all requirements for functionality, stability, and performance.