**Arduino Mega 2560 Full Firmware Sketch**

# Arduino Mega 2560 Vollständige Firmware-Skizze

```
/*
  Robotic Arm Controller
  • Reads: HC-SR04 distance, MPX5700DP pressure, LM35 temperature,
           potentiometers (shoulder/elbow)
  • Controls: 3 × A4988 steppers via PID loops,
              hydraulic valve via PWM (4–20 mA driver),
              3 × gripper servos
  • Interfaces: HC-05 Bluetooth (Serial1 @ 9600 bps)
                ESP8266 Wi-Fi (Serial2 @ 115200 bps) & MQTT
*/

#include <AccelStepper.h>
#include <PID_v1.h>
#include <NewPing.h>
#include <Servo.h>
```

```
//————————————————————————————————————————
// Pin Assignments
//————————————————————————————————————————
// Stepper drivers (A4988)
#define STEP1_PIN      2    // Motor 1 step
#define DIR1_PIN       3
#define STEP2_PIN      4    // Motor 2 step
#define DIR2_PIN       5
#define STEP3_PIN      6    // Motor 3 step
#define DIR3_PIN       7

// Hydraulic valve PWM → 4–20 mA converter
#define VALVE_PWM_PIN   9

// Ultrasonic sensor HC-SR04
#define ULTRA_TRIG_PIN  8
#define ULTRA_ECHO_PIN 10
#define MAX_DISTANCE   200  // cm max range

// Analog sensors
#define PRESSURE_PIN      A0  // MPX5700DP
```

```
#define TEMPERATURE_PIN   A1  // LM35
#define POT_SHOULDER_PIN  A2
#define POT_ELBOW_PIN     A3

// Gripper servos (3-pin headers)
#define GRIP_SERVO1_PIN  11
#define GRIP_SERVO2_PIN  12
#define GRIP_SERVO3_PIN  13

// Serial Interfaces
#define BT_SERIAL    Serial1   // HC-05 default 9600 bps
#define WIFI_SERIAL  Serial2   // ESP8266 AT → 115200 bps

//————————————————————————————————————————
// Wi-Fi / MQTT Settings (edit before uploading)
//————————————————————————————————————————
const char* WIFI_SSID     = "YOUR_SSID";
const char* WIFI_PASS     = "YOUR_PASS";
const char* MQTT_BROKER   = "broker.hivemq.com";
const uint16_t MQTT_PORT  = 1883;
const char* MQTT_TOPIC    = "robot_arm/telemetry";
```

```
//————————————————————————————————————————

// Module Objects

//————————————————————————————————————————

AccelStepper motor1(AccelStepper::DRIVER, STEP1_PIN,
DIR1_PIN);

AccelStepper motor2(AccelStepper::DRIVER, STEP2_PIN,
DIR2_PIN);

AccelStepper motor3(AccelStepper::DRIVER, STEP3_PIN,
DIR3_PIN);


NewPing sonar(ULTRA_TRIG_PIN, ULTRA_ECHO_PIN,
MAX_DISTANCE);


Servo gripper1, gripper2, gripper3;


// PID parameters & variables

double setpoint1, input1, output1;

double setpoint2, input2, output2;

double setpoint3, input3, output3;


// Tuning: adjust to your mechanics
```

```
double Kp = 2.0, Ki = 0.5, Kd = 0.1;
PID pid1(&input1, &output1, &setpoint1, Kp, Ki, Kd, DIRECT);
PID pid2(&input2, &output2, &setpoint2, Kp, Ki, Kd, DIRECT);
PID pid3(&input3, &output3, &setpoint3, Kp, Ki, Kd, DIRECT);

// Timing
unsigned long lastPublish = 0;
const unsigned long PUBLISH_INTERVAL = 1000;  // ms

//————————————————————————————————————————
// Helper: send AT commands to ESP8266 & wait for response
//————————————————————————————————————————
bool sendAT(const char* cmd, const char* ack, uint16_t timeout)
{
  WIFI_SERIAL.print(cmd);
  unsigned long t0 = millis();
  String resp;
  while (millis() - t0 < timeout) {
    if (WIFI_SERIAL.available()) {
      resp += char(WIFI_SERIAL.read());
      if (resp.indexOf(ack) != -1) return true;
```

```
    }
  }
  return false;
}

//————————————————————————————————————————
// Setup
//————————————————————————————————————————
void setup() {
  // Debug on USB-Serial
  Serial.begin(115200);
  while(!Serial);

  // Init hardware serials
  BT_SERIAL.begin(9600);
  WIFI_SERIAL.begin(115200);

  // Motor max speed & acceleration
  motor1.setMaxSpeed(1000);
  motor1.setAcceleration(500);
  motor2.setMaxSpeed(1000);
```

```
motor2.setAcceleration(500);
motor3.setMaxSpeed(1000);
motor3.setAcceleration(500);

// PID setup
pid1.SetMode(AUTOMATIC);
pid2.SetMode(AUTOMATIC);
pid3.SetMode(AUTOMATIC);
pid1.SetOutputLimits(-400, 400);
pid2.SetOutputLimits(-400, 400);
pid3.SetOutputLimits(-400, 400);

// Valve PWM
pinMode(VALVE_PWM_PIN, OUTPUT);
analogWrite(VALVE_PWM_PIN, 0);

// Servos
gripper1.attach(GRIP_SERVO1_PIN);
gripper2.attach(GRIP_SERVO2_PIN);
gripper3.attach(GRIP_SERVO3_PIN);

// Wi-Fi → MQTT
```

```
  Serial.println("Initializing ESP8266...");
  if (!sendAT("AT\r\n", "OK", 2000)) {
    Serial.println("ESP8266 not responding");
  }
  sendAT("AT+CWMODE=1\r\n", "OK", 2000);
  char cmd[64];
  snprintf(cmd, sizeof(cmd), "AT+CWJAP=\"%s\",\"%s\"\r\n",
WIFI_SSID, WIFI_PASS);
  if (!sendAT(cmd, "OK", 8000)) {
    Serial.println("Wi-Fi join failed");
  }
  snprintf(cmd, sizeof(cmd),
          "AT+CIPSTART=\"TCP\",\"%s\",%u\r\n",
          MQTT_BROKER, MQTT_PORT);
  sendAT(cmd, "OK", 5000);
}

//————————————————————————————————————————
// Telemetry publisher (simple MQTT PUBLISH via AT)
//————————————————————————————————————————
void publishTelemetry(float dist, float pres, float temp) {
```

```
char payload[128];
// JSON style
snprintf(payload, sizeof(payload),
        "{\"dist\":%.1f,\"press\":%.1f,\"temp\":%.1f}",
        dist, pres, temp);

// Build MQTT PUBLISH packet (fixed header + topic + payload)
// Simplified: length = topicLen + payloadLen + 10
uint16_t topicLen = strlen(MQTT_TOPIC);
uint16_t dataLen  = strlen(payload);
uint16_t pktLen   = 2 + topicLen + 2 + dataLen;

// Send AT+CIPSEND
char cip[32];
snprintf(cip, sizeof(cip), "AT+CIPSEND=%u\r\n", pktLen + 2);
if (!sendAT(cip, ">", 2000)) return;

// MQTT header
WIFI_SERIAL.write(0x30);         // PUBLISH, QoS0
WIFI_SERIAL.write(pktLen);       // Remaining length
WIFI_SERIAL.write((topicLen >> 8) & 0xFF);
WIFI_SERIAL.write(topicLen & 0xFF);
```

```
  WIFI_SERIAL.print(MQTT_TOPIC);
  WIFI_SERIAL.write((dataLen >> 8) & 0xFF);
  WIFI_SERIAL.write(dataLen & 0xFF);
  WIFI_SERIAL.print(payload);
  sendAT("\r\n", "SEND OK", 3000);
}

//————————————————————————————————————————
// Loop
//————————————————————————————————————————
void loop() {
  // 1) Read analog sensors
  float voltage, temperatureC, pressureKpa;
  // Temperature LM35: 10 mV/°C
  voltage     = analogRead(TEMPERATURE_PIN) * (5.0 / 1023.0);
  temperatureC= voltage * 100.0;

  // Pressure MPX5700DP: 0.2 V @ 0 kPa → 4.7 V @ 700 kPa
  voltage      = analogRead(PRESSURE_PIN) * (5.0 / 1023.0);
  pressureKpa  = (voltage - 0.2) * (700.0 / (4.7 - 0.2));
```

```
  // Distance
  float distanceCm = sonar.ping_cm();


  // Potentiometers → angle setpoints
  setpoint1 = map(analogRead(POT_SHOULDER_PIN), 0, 1023, 0, 180);
  setpoint2 = map(analogRead(POT_ELBOW_PIN),    0, 1023, 0, 180);
  // For motor3 without pot feedback, you could parse a command via Bluetooth:
  // input3 = current position (if sensor exists), else leave PID off.


  // 2) PID compute & stepper update
  input1 = /* read actual shoulder angle, e.g. via separate sensor */;
  pid1.Compute();
  motor1.setSpeed(output1);
  motor1.runSpeed();


  input2 = /* read actual elbow angle */;
  pid2.Compute();
  motor2.setSpeed(output2);
  motor2.runSpeed();
```

```
// Third motor open-loop example:
// motor3.setSpeed(0); // or driven from commands
// motor3.runSpeed();

// 3) Gripper example (static position):
gripper1.write(90);
gripper2.write(90);
gripper3.write(90);

// 4) Valve control (example: map pressure → valve)
int pwmVal = map(analogRead(PRESSURE_PIN), 0, 1023, 0, 255);
analogWrite(VALVE_PWM_PIN, pwmVal);

// 5) Bluetooth command handling (optional)
if (BT_SERIAL.available()) {
  String cmd = BT_SERIAL.readStringUntil('\n');
  // Parse commands like "S1:45" → setpoint1=45, etc.
}

// 6) Periodic MQTT telemetry
if (millis() - lastPublish > PUBLISH_INTERVAL) {
```

```
    publishTelemetry(distanceCm, pressureKpa, temperatureC);
    lastPublish = millis();
  }
}
```