PIC18F46K22 Robotic Arm Controller Firmware

Written for the Microchip PICC (C18) compiler. Mirrors the Arduino sketch functionality: • Reads HC-SR04, MPX5700DP, LM35 & two potentiometers • Drives 3 × A4988 steppers via PID loops • Controls hydraulic valve via CCP1 PWM → 4–20 mA • Positions 3 × gripper servos via software PWM • Communicates over HC-05 (UART1) and ESP8266 (UART2) with MQTT telemetry

PIC18F46K22 Roboterarm-Controller-Firmware

Geschrieben für den Microchip PICC (C18)-Compiler. Spiegelt die Arduino-Sketch-Funktionalität wider: • Liest HC-SR04, MPX5700DP, LM35 und zwei Potentiometer • Steuert 3 × A4988-Schrittmotoren über PID-Regelkreise • Steuert Hydraulikventile über CCP1 PWM → 4–20 mA • Positioniert 3 × Greiferservos über Software-PWM • Kommuniziert über HC-05 (UART1) und ESP8266 (UART2) mit MQTT-Telemetrie

1. Configuration Bits & Definitions

1. Configuration Bits & Definitions

```
#include <p18f46k22.h>

#include <delays.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


// CONFIG1H
```

```
#pragma config FOSC = HS2PLL  // High-speed crystal w/PLL
#pragma config PLLDIV = 5     // (20 MHz / 5) = 4 MHz × 4 = 16 MHz internal
#pragma config CPUDIV = OSC1_PLL2
#pragma config USBDIV = 2     // USB clock from PLL/2

// CONFIG2L
#pragma config PWRT = ON      // Power-up Timer
#pragma config BOR = ON       // Brown-out Reset
#pragma config BORV = 3       // Brown-out Voltage

// CONFIG2H
#pragma config WDT = OFF      // Watchdog Timer
#pragma config WDTPS = 32768

// CONFIG3H
#pragma config CCP2MX = PORTC // CCP2 on RC1
#pragma config PBADEN = OFF   // PORTB<4:0> digital on RESET
#pragma config LPT1OSC = OFF

// CONFIG4L
#pragma config STVREN = ON    // Stack Overflow Reset
```

```
#pragma config LVP = OFF      // No Low-Voltage Programming
#pragma config XINST = OFF

// Timing definitions
#define _XTAL_FREQ 20000000UL

// Pin macros
#define STEP1_LAT LATDbits.LATD0
#define DIR1_LAT  LATDbits.LATD1
#define STEP2_LAT LATDbits.LATD2
#define DIR2_LAT  LATDbits.LATD3
#define STEP3_LAT LATDbits.LATD4
#define DIR3_LAT  LATDbits.LATD5

#define TRIG_LAT  LATCbits.LATC3
#define ECHO_PORT PORTCbits.RC4

#define SERVO1_LAT LATDbits.LATD6
#define SERVO2_LAT LATDbits.LATD7
#define SERVO3_LAT LATCbits.LATC0

// UART macros
```

```c
#define BT_TX_PIN  TRISCbits.TRISC6
#define BT_RX_PIN  TRISCbits.TRISC7
#define WIFI_TX_PIN TRISBbits.TRISB6
#define WIFI_RX_PIN TRISBbits.TRISB7

// ADC channels
#define CH_PRESSURE 0   // AN0, RA0
#define CH_TEMP     1   // AN1, RA1
#define CH_POT1     2   // AN2, RA2
#define CH_POT2     3   // AN3, RA3

// PID parameters
#define PID_KP  2.0
#define PID_KI  0.5
#define PID_KD  0.1

// Telemetry interval (ms)
#define TELEMETRY_INTERVAL 1000UL

// Wi-Fi / MQTT credentials
const rom char WIFI_SSID[]  = "YOUR_SSID";
const rom char WIFI_PASS[]  = "YOUR_PASS";
```

```
const rom char MQTT_BROKER[] = "broker.hivemq.com";
const unsigned int MQTT_PORT = 1883;
const rom char MQTT_TOPIC[] = "robot_arm/telemetry";
```

---

2. Global Variables & Structures

2. Globale Variablen und Strukturen

```
// Sensor readings
volatile float distance_cm;
volatile float pressure_kpa;
volatile float temp_c;

// Potentiometer setpoints
volatile float set_shoulder, set_elbow;

// PID state
typedef struct {
    float setpoint, input, output;
    float integral, last_error;
} PID_t;

PID_t pid1, pid2, pid3;
```

// Telemetry timer

volatile unsigned long millis_count = 0UL;

volatile unsigned long last_telemetry = 0UL;

---

3. Function Prototypes

3. Funktionsprototypen

void initHardware(void);

void initADC(void);

void initPWM(void);

void initUARTs(void);

void initTimers(void);

unsigned int readADC(unsigned char channel);

float readDistance(void);

void computePID(PID_t* pid);

void updateSteppers(void);

void updateServos(void);

void sendAT(const char* cmd, const char* ack, unsigned long timeout);

```
void publishTelemetry(void);
```

4. Initialization

4. Initialisierung

```
void main(void) {
   initHardware();

   // Initialize PID structures
   pid1.setpoint = pid2.setpoint = pid3.setpoint = 0.0f;
   pid1.integral = pid2.integral = pid3.integral = 0.0f;
   pid1.last_error = pid2.last_error = pid3.last_error = 0.0f;

   // ESP8266 AT init
   sendAT("AT\r\n", "OK", 2000);
   sendAT("AT+CWMODE=1\r\n", "OK", 2000);
   char buf[64];
   sprintf(buf, "AT+CWJAP=\"%s\",\"%s\"\r\n", WIFI_SSID,
WIFI_PASS);
   sendAT(buf, "OK", 8000);
   sprintf(buf, "AT+CIPSTART=\"TCP\",\"%s\",%u\r\n",
MQTT_BROKER, MQTT_PORT);
   sendAT(buf, "OK", 5000);

   while (1) {
      // 1) Read sensors
      temp_c      = (readADC(CH_TEMP) * (5.0f/1023.0f)) *
100.0f;
      pressure_kpa= (readADC(CH_PRESSURE)*(5.0f/1023.0f)
- 0.2f) * (700.0f/(4.7f-0.2f));
```

```
    set_shoulder= (readADC(CH_POT1)   * 180.0f) / 1023.0f;
    set_elbow   = (readADC(CH_POT2)   * 180.0f) / 1023.0f;
    distance_cm = readDistance();

    // 2) PID compute
    pid1.setpoint = set_shoulder;
    pid1.input    = 0.0f; // replace with actual feedback
sensor
    computePID(&pid1);

    pid2.setpoint = set_elbow;
    pid2.input    = 0.0f; // replace with actual feedback
sensor
    computePID(&pid2);

    // 3) Apply outputs
    updateSteppers();
    updateServos();

    // 4) Valve PWM (map pressure → duty cycle)
    CCPR1L = (unsigned char)((readADC(CH_PRESSURE) *
255) / 1023);

    // 5) Bluetooth command handling (optional)
    if (PIR3bits.RC1IF) {
      char c = RCREG1;
      // parse “S1:45\n” etc.
    }

    // 6) Periodic MQTT telemetry
```

```
        if ((millis_count - last_telemetry) >=
TELEMETRY_INTERVAL) {
            publishTelemetry();
            last_telemetry = millis_count;
        }
    }
}
```

5. Hardware Setup Routines

5. Hardware-Setup-Routinen

```
void initHardware(void) {
    // I/O directions
    TRISD = 0b10000000;  // RD6-7 outputs for servos; RD0-5
outputs for steppers
    TRISC = 0b10010000;  // RC4 input (echo), RC3 output (trig),
RC6-7 UART1
    TRISB = 0b11000000;  // RB6-7 UART2
    TRISA = 0xFF;        // RA0-RA3 analog
    LATD = 0; LATC = 0; LATB = 0;

    initADC();
    initPWM();
    initUARTs();
    initTimers();
```

```
}

void initADC(void) {
    ADCON0 = 0x01;      // ADC ON, channel 0 default
    ADCON1 = 0x0E;      // RA0-RA3 analog, others digital
    ADCON2 = 0xA9;      // Right justified, 4Tad, Fosc/8
}

void initPWM(void) {
    // CCP1 → RC2
    TRISCbits.TRISC2 = 0;
    PR2 = 0xFF;         // PWM period
    CCP1CON = 0x0C;     // PWM mode
    T2CON = 0x04;       // Timer2 on, prescale 1:1
}

void initUARTs(void) {
    // UART1 → HC-05 @ 9600
    TRISC6 = 1; TRISC7 = 1;
    RCSTA1 = 0x90;      // SPEN, CREN
    TXSTA1 = 0x24;      // BRGH=1, TX enable
    SPBRG1 = (_XTAL_FREQ/16/9600)-1;
```

```
    // UART2 → ESP8266 @115200
    TRISB6 = 1; TRISB7 = 1;
    RCSTA2 = 0x90;
    TXSTA2 = 0x24;
    SPBRG2 = (_XTAL_FREQ/16/115200)-1;
}

void initTimers(void) {
    // Timer0 → 1 ms tick for millis_count
    T0CON = 0x88;       // 16-bit, prescale 1:16
    INTCON2bits.T0IP = 1;
    INTCONbits.TMR0IE = 1;
    TMR0H = 0xF0; TMR0L = 0x18;  // preload for ~1 ms
    INTCONbits.GIE = 1; INTCONbits.PEIE = 1;
}
```

6. Utility & ISR

```
#pragma code high_vector=0x08
void interrupt_at_high_vector(void){ _asm goto isr _endasm }
#pragma code
```

```
#pragma interrupt isr
void isr(void) {
  // Timer0 overflow → ~1 ms
  if (INTCONbits.TMR0IF) {
    TMR0H = 0xF0; TMR0L = 0x18;
    INTCONbits.TMR0IF = 0;
    millis_count++;
  }
}

unsigned int readADC(unsigned char channel) {
  ADCON0 = (channel<<2) | 0x01; // select channel & turn on
  Delay10TCYx(5);          // acquisition time
  ADCON0bits.GO = 1;
  while (ADCON0bits.GO);
  return ((ADRESH<<8) | ADRESL);
}

float readDistance(void) {
  unsigned int t;
  // Trigger 10 µs pulse
  TRIG_LAT = 1; Delay10TCYx(2); TRIG_LAT = 0;
```

```
    // Wait echo high
    t = 0;
    while (!ECHO_PORT && t<60000) { t++; }
    TMR1H = TMR1L = 0; T1CON = 0x01; // start Timer1, prescale=1
    while (ECHO_PORT && TMR1L < 0xFF) {}
    T1CON = 0; // stop
    unsigned long cnt = ((unsigned int)TMR1H<<8)|TMR1L;
    // Timer1 increments at Fosc/4 = 5 MHz → 0.2 µs tick
    return (cnt * 0.0002f) / 2.0f; // round-trip
}
```

---

7. PID & Actuator Updates

7. PID- und Aktuator-Updates

```
void computePID(PID_t* pid) {
    float error = pid->setpoint - pid->input;
    pid->integral += PID_KI * error;
    float derivative = error - pid->last_error;
    pid->output = PID_KP*error + pid->integral +
PID_KD*derivative;
    // clamp output to safe range
    if (pid->output > 400) pid->output = 400;
    if (pid->output < -400) pid->output = -400;
    pid->last_error = error;
```

```
}

void updateSteppers(void) {
    // motor1
    DIR1_LAT = (pid1.output >= 0);
    // toggle STEP1 at frequency ∝ |output|
    // implement timer-based pulse generation or software delay
    // ...
    // motor2 similarly
    DIR2_LAT = (pid2.output >= 0);
    // ...
    // motor3 open-loop or command-driven
}

void updateServos(void) {
    // crude software PWM for servos (1 ms–2 ms in 20 ms)
    static unsigned long last_pwm = 0;
    static unsigned char phase = 0;
    if (millis_count - last_pwm < 20) return;
    last_pwm = millis_count;
    // generate pulses on RD6,7, RC0 according to desired angle
(90° = 1.5 ms)
```

```
    // ...
}

void sendAT(const char* cmd, const char* ack, unsigned long
timeout) {
    unsigned long start = millis_count;
    TXREG2 = 0;           // flush
    while (*cmd) {
        while (!PIR3bits.TX2IF);
        TXREG2 = *cmd++;
    }
    // wait for ack
    char buf[64]; unsigned char idx = 0;
    while ((millis_count - start) < timeout) {
        if (PIR3bits.RC2IF) {
            buf[idx++] = RCREG2;
            buf[idx] = 0;
            if (strstr(buf, ack)) return;
        }
    }
}
```

```
void publishTelemetry(void) {
   char payload[128];
   sprintf(payload, "{\"dist\":%.1f,\"press\":%.1f,\"temp\":%.1f}",
         distance_cm, pressure_kpa, temp_c);
   unsigned int topicLen = strlen(MQTT_TOPIC);
   unsigned int dataLen  = strlen(payload);
   unsigned int pktLen   = 2 + topicLen + 2 + dataLen;
   char cmd[32];
   sprintf(cmd, "AT+CIPSEND=%u\r\n", pktLen+2);
   sendAT(cmd, ">", 2000);

   // build & send MQTT packet
   unsigned char hdr[] = {0x30, pktLen,
                  (topicLen>>8)&0xFF, topicLen&0xFF};
   for (unsigned int i=0; i<sizeof(hdr); i++) {
      while (!PIR3bits.TX2IF);
      TXREG2 = hdr[i];
   }
   for (unsigned int i=0; i<topicLen; i++) {
      while (!PIR3bits.TX2IF);
      TXREG2 = MQTT_TOPIC[i];
   }
```

```
    unsigned char lenBytes[2] = {(dataLen>>8)&0xFF,
dataLen&0xFF};
    for (int i=0; i<2; i++) {
        while (!PIR3bits.TX2IF);
        TXREG2 = lenBytes[i];
    }
    for (unsigned int i=0; i<dataLen; i++) {
        while (!PIR3bits.TX2IF);
        TXREG2 = payload[i];
    }
    sendAT("\r\n", "SEND OK", 3000);
}
```