

طرح جامع پروژه ربات جوشکاری خودکار

عنوان پروژه

ربات جوشکاری خودکار اتصال لب به لب (Butt-Welding) برای لوله‌های با قطر ۳۰ سانتی‌متر مجهز به سامانه تشخیص فاصله، ترسیم نقشه شکاف (Gap Mapping) و تنظیم سرعت فیلر (Filler Control)

اهداف اصلی پروژه

ما یک سیستم جوشکاری (Butt-Welding) برای لوله در نظر گرفته‌ایم که از چند موتور پله‌ای (X, Y, F) و یک موتور R (برای چرخش لوله) تشکیل شده است. این سیستم باید به صورت غیرمسدودکننده (Non-blocking) کنترل شود تا محورهای مختلف بتوانند همزمان یا با هماهنگی مناسب حرکت کنند. در کنار آن، یک انکودر روی موتور DC نصب شده تا بتوان سرعت یا میزان چرخش لوله را پایش کرد (اما فعلاً کنترل سرعت خیلی پیچیده مدنظر نیست؛ فقط On/Off).

اهداف اصلی

- کنترل همزمان محورهای X، Y و F (فیلر) با روش غیرمسدودکننده.
- روشن/خاموش کردن موتور R برای چرخش لوله و خواندن انکودر آن به صورت ساده (Polling).
- ترسیم نقشه شکاف (Gap Mapping)
- انجام یک سیکل خودکار (Left-and-Right) روی محور X با تعداد تعیین شده.
- در پایان سیستم به خانه بازگردد و موتور R را خاموش کند.
- قابلیت توسعه برای اضافه کردن منطق بیشتر (مثل سنسور فاصله، دوربین و ...).

1. جوشکاری اتوماتیک دو لوله

- جوشکاری دو لوله (مثلاً استیل و آهن) با وزن تقریبی ۳۰ کیلوگرم و قطر ۳۰ سانتی‌متر.
- کنترل دقیق شکاف بین لوله‌ها، مقدار و سرعت سیم جوش (فیلر)، و فاصله نازل جوش از سطح لوله.

2. بینایی ماشین و پردازش تصویر

- استفاده از دوربین رزبری پای و OpenCV برای تشخیص عرض شکاف جوش.

- ساخت نقشه شکاف (Gap Map) در طول چرخش ۳۶۰ درجه لوله، جهت تنظیم پیش‌دستانه (Proactive) سرعت فیلر.

3. کنترل پیشرفته موتورهای پله‌ای

- استفاده از **Arduino Mega** برای مدیریت موتورهای پله‌ای:
 - **R**: چرخش لوله (Rotational) با انکودر برای فیدبک دقیق زوایا.
 - **X**: حرکت افقی مشعل جوشکاری (Torch) در راستای چپ/راست.
 - **Y**: تنظیم ارتفاع مشعل جوشکاری (Torch) با کمک سنسور فاصله **VL53L0X**.
 - **F**: موتور تغذیه سیم جوش با امکان تنظیم سرعت.

4. ایمنی و قابلیت اطمینان

- رد کردن کارهایی که شکاف < 6 میلی‌متر دارند.
- مجهز به شستی اضطراری (E-stop)، میکروسوییچ‌های محدودکننده (Limit Switch)، و مدیریت خطا.
- ثبت اطلاعات (Log) مربوط به زاویه و شکاف برای ردگیری و مستندسازی.

شرح دقیق سیستم

۱. بخش مکانیکی

1. نگهدارنده و چرخاننده لوله (Pipe Holder & Rotator)

- دو پایه یا فیکسچر مستحکم برای قرارگیری لوله‌ها (حدود ۳۰ کیلوگرم هر کدام).
- موتور **R** (مثلاً NEMA 17/موتور DC) برای چرخاندن لوله با یک تسمه یا گیربکس.
- انکودر **E6A2** برای خواندن سرعت چرخش و زاویه دقیق.

2. مجموعه مشعل جوش

- موتور پله‌ای **X**: وظیفه دارد مشعل را در راستای افقی (چپ/راست) حرکت دهد.
- موتور پله‌ای **Y**: وظیفه تنظیم ارتفاع مشعل را دارد.
- دریافت داده از سنسور **VL53L0X** و حفظ فاصله ثابت (مثلاً 1.5 میلی‌متر).

3. تغذیه سیم جوش (Filler Feed)

- موتور پله‌ای F با قابلیت تعیین سرعت توسط آردینو (یا فرمان مستقیم از رزبری‌پای).
-

۲. بخش الکترونیکی و کنترل

1. کنترل اصلی

- Raspberry Pi (مثلاً مدل ۵): مسئول پردازش تصویر، بینایی ماشین، و منطق تصمیم‌گیری سطح بالا.
- Arduino Mega: کنترل دقیق موتورها، دریافت و ارسال لحظه‌ای فیدبک سنسورها (انکودر و VL53L0X).

2. درایور موتور

- استفاده از شیلد CNC و درایورهای A4988 (یا درایورهای سازگار) برای موتورهای پله‌ای R, X, Y, F.
- تنظیم محدودکننده جریان برای جلوگیری از داغ شدن موتور یا از دست رفتن گام‌ها.

3. دوربین و نورپردازی

- ماژول دوربین رزبری‌پای (دوربین V3) یا مشابه.
- نور LED کافی یا رینگ لایت جهت حذف سایه و بازتاب‌های مزاحم روی فلز.

4. منبع تغذیه

- ۲۴ ولت DC (یا ۱۲ ولت، بسته به گشتاور لازم) برای موتورهای پله‌ای.
- ۵ ولت تنظیم‌شده برای رزبری‌پای و سنسورها.

5. سنسورها و ماژول‌های فرعی

- انکودر E6A2 برای محور R.
 - سنسور فاصله VL53L0X برای محور Y.
 - ماژول لیزر برای راهنمای جوشکاری.
 - رله برای روشن/خاموش کردن تورچ جوش.
-

۳. بخش نرم‌افزار و جریان داده (Data Flow)

1. نرم‌افزار آردوینو مگا

○ کنترل گام موتورها:

- دریافت فرمان از رزبری‌پای: مثلاً «چرخش لوله تا زاویه X»، «تنظیم سرعت موتور فیلر روی مقدار Y»، و غیره.
- تولید پالس‌های دقیق برای موتورهای پله‌ای (R, X, Y, F).

○ بازخورد (Feedback):

- موقعیت فعلی موتور X، زاویه محور R، و سرعت F را به رزبری‌پای گزارش می‌دهد.

○ رسیدگی وقفه‌ها:

- انکودر R برای تشخیص گردش لوله.
- سنسور فاصله برای تصحیح ارتفاع Y.
- شستی اضطراری و میکروسوییچ‌ها.

2. برنامه رزبری‌پای (Python/OpenCV)

○ ابتدای کار (Initialization):

- برقراری ارتباط سریال با آردوینو.
- کالیبراسیون دوربین برای تشخیص mm به پیکسل.

○ حلقه اصلی جوشکاری:

■ چرخش لوله و تصویربرداری

- صدور فرمان به آردوینو: «تا زاویه X برو» یا «با سرعت مشخص بچرخ».
- پس از پایدار شدن، گرفتن تصویر با دوربین.

■ پردازش تصویر (ROI ۵cm×۲cm)

- برش تصویر، تبدیل به خاکستری، آستانه‌گذاری (Threshold)، تشخیص لبه یا تحلیل پیکسل‌ها.

- محاسبه عرض شکاف بر حسب میلی‌متر با استفاده از ضریب کالیبراسیون.

■ نقشه‌برداری و تصمیم‌گیری

- اگر شکاف < 6 میلی‌متر، پروژه رد شود.
- در غیر این صورت، شکاف در جدول/آرایه‌ای ثبت می‌شود (Gap Map).

■ کنترل فیلر

- براساس شکاف، سرعت فیلر تنظیم می‌شود (عادی یا سریع).
- گردش ۳۶۰ درجه
- با انکودر محور R، زوایای ۰ تا ۳۶۰ درجه را پیمایش می‌کنیم.

○ رابط کاربری (UI) و گزارش

- نمایش زاویه، شکاف، سرعت فیلر، و خطاها.
- ذخیره لاگ نهایی (زاویه-شکاف-سرعت).

3. پایان کار (Shutdown)

- توقف چرخش لوله، توقف موتور فیلر، خاموش کردن لیزر، بازگشت محورهای X و Y به خانه (Home).
 - نگهداری و بایگانی داده‌های لاگ.
-

منطق توالی رفتار سیستم (Behavior Logic)

1. راه اندازی (Startup)

- کاربر لوله ها را در نگهدارنده قرار می دهد.
- فعال کردن لیزر راهنما برای تراز اولیه.
- آردوینو محور R را به مبدأ (۰ درجه طبق انکودر) تنظیم می کند.
- شکاف در جدول/آرایه ای ثبت می شود (Gap Map).
- توقف در صورت Reject شدن کار

2. عملیات اصلی Welding

- حرکت X:
 - از خانه (۰) تا مرکز مثلاً ۱۰۰۰ گام.
 - انجام حرکت رفت و برگشتی ± 400 گام با سرعت تنظیم شده (مثلاً ۲ ثانیه برای یک سیکل کامل).
- تنظیم ارتفاع Y:
 - شروع از پوزیشن اولیه (مثلاً ۵۰۰ گام).
 - قرائت سنسور VL53L0X؛ اگر فاصله از 1.5 میلی متر بیشتر یا کمتر از حد مجاز باشد، موتور Y تنظیم می شود.
- تغذیه سیم (F):
 - فید پیوسته با سرعت پایه؛ اگر شکاف زیاد شود (اما هنوز ≥ 6 میلی متر)، سرعت افزایش می یابد.
 - چرخاندن لوله (R) با سرعت حدود ۱ دور در دقیقه (یا هر مقدار تعیین شده).
 - بینایی و بررسی شکاف:
 - در زوایای مختلف (با تکیه بر انکودر)، تصویربرداری و تحلیل.
 - اگر در بخشی شکاف $6 <$ میلی متر شد، کار متوقف یا اخطار داده می شود.
 - هماهنگی و همگام سازی:
 - حرکت X، تنظیم Y، و چرخش R تا تکمیل ۳۶۰ درجه یا پایان سیکل.

3. خاموشی و بازگشت (Shutdown)

- قطع فید سیم، توقف موتور R (برگشت به ۰ درجه).
- خاموش کردن لیزر.
- بازگشت محورهای X و Y به خانه.
- ذخیره همه داده ها و اعلام اتمام کار.

بهبودها و نکات کلیدی

1. انکودر برای موتور R

- استفاده از انکودر E6A2 برای مانیتور لحظه‌ای سرعت و زاویه لوله.
- اطمینان از موقعیت صفر (۰ درجه) و طی کردن کامل ۳۶۰ درجه.

2. کنترل پویا در محور Y

- سنسور فاصله VL53L0X به‌طور مداوم فاصله نازل جوش تا سطح لوله را اندازه‌گیری می‌کند (مثلاً 1.5 میلی‌متر).
- با هر انحراف خارج از بازه مجاز (مثلاً $0.5 \pm$ میلی‌متر)، محور Y به‌صورت خودکار تنظیم می‌شود.

3. منطق پیشرفته برای موتور فیلر (F)

- موتور پله‌ای F سرعت تغذیه را براساس شکاف جوش یا دستور دستی کاربر کم/زیاد می‌کند.
- در بازه‌های خاصی (مثلاً ۲ الی ۶ میلی‌متر) سرعت فیلر بالاتر می‌رود و اگر بالاتر از ۶ میلی‌متر باشد، پروژه رد می‌شود.

4. ساختار ماژولار نرم‌افزاری

- رزبری پای پردازش‌های سطح بالا (بینایی و تصمیم‌گیری) را انجام می‌دهد.
- آردوینو مگا دستورات موتور، پالس‌ها و کنترل آنی سنسورها را مدیریت می‌کند.

5. لیزر راهنما

- یک ماژول لیزر برای بررسی موقعیت جوش یا هم‌راستا کردن لوله‌ها.
- می‌تواند توسط رزبری پای به‌صورت نرم‌افزاری روشن/خاموش شود.

پارامترهای قابل تنظیم

1. محور X

- فاصله تا مرکز (مثلاً ۱۰۰۰ گام)
- دامنه حرکت رفت‌وبرگشتی (مثلاً $400 \pm$ گام)
- سرعت (زمان برای هر رفت‌وبرگشت؛ مثلاً 1 ثانیه)
- تعداد سیکل‌ها یا ارتباط آن با چرخش ۳۶۰ درجه محور R

2. محور Y

- نقطه شروع (مثلاً ۵۰۰ گام)
- فاصله هدف از سطح لوله (مثلاً 1.5 میلی‌متر)
- بازه مجاز خطا (مثلاً $1 \pm$ میلی‌متر)
- میزان حرکت در هر تصحیح (مثلاً ۱۰ گام)
- سرعت تصحیح (تاخیر در میکروثانیه یا میزان گام بر ثانیه)

3. موتور فیلر (F)

- سرعت پایه فید (تعیین فاصله زمانی بین پالس‌های فید)
- سرعت بالاتر در صورت بزرگ شدن شکاف
- توقف در صورت Reject شدن کار

4. موتور R (چرخش لوله)

- سرعت (مثلاً ۱ دور در دقیقه یا RPM‌های پایین/بالا)
- اتصال به انکودر برای تشخیص دقیق زاویه و بازگشت به نقطه صفر

5. تورچ جوش (روشن/خاموش)

- رله یا ماژول خروجی دیجیتال؛ فرمان HIGH یا LOW

6. لیزر راهنما

- پین کنترل برای روشن/خاموش
 - در مرحله تنظیم روشن باشد؛ هنگام جوش می‌توان خاموش کرد (برای کاهش رفلکس و ایمنی)
-

نمونه شبه‌کد (Pseudocode) آردوینو

```
/******
 * Non-Blocking Multi-Stepper and DC Motor Control Example
 *
 * Author: ChatGPT (modified based on your requests)
 * Description:
 *   - Demonstrates how to move X, Y, and F (filler) stepper
 *     motors in a non-blocking manner using micros() timing.
 *   - Controls a DC motor for pipe rotation (on/off).
 *   - Reads a basic encoder for DC motor (polling method, no ISR).
 *   - Provides a simple "back-and-forth" cycle on X axis.
 *
 * NOTE: You must adjust step intervals, directions, and
 *       mechanical setup to match your actual rig.
 *       Also consider adding limit switch logic and
 *       advanced safety features (emergency stops, etc.).
 *****/

/******
 * User Configuration Area
 *
 * --> THESE ARE THINGS YOU NEED TO SET/CONFIRM <--
 *
 * 1) Pin assignments that match your hardware wiring.
 * 2) Step intervals or microstepping settings.
 * 3) The direction (HIGH/LOW) that moves each motor
 *    forward/backward in your physical setup.
 * 4) RPM or speed control logic for the DC motor
 *    if needed (currently just ON/OFF).
 * 5) Possibly incorporate sensor feedback or
 *    advanced logic as needed.
 *****/

// --- Pin Assignments ---

// DC motor (pipe rotation) and encoder
int DC_PIN      = 12;    // Pin controlling DC motor relay/driver (LOW =
ON, HIGH = OFF)
int ENCODER_A   = 9;     // Encoder A pin
int ENCODER_B   = 10;    // Encoder B pin
```

```

// Enable pin for stepper drivers (most CNC shields use one pin to
enable/disable all steppers)
int EN_PIN      = 8;    // LOW = enabled, HIGH = disabled

// X axis stepper
int X_STEP      = 2;
int X_DIR       = 5;

// Y axis stepper
int Y_STEP      = 3;
int Y_DIR       = 6;

// F axis stepper (filler) - previously "Z"
int F_STEP      = 4;
int F_DIR       = 7;

/*****
 * Motion Parameters
 * -> Tweak these values to suit your machine.
 *****/

// X axis travel
int X_CENTER_STEPS = 1000; // Steps for X to reach "center" position
from home
int X_TRAVEL_STEPS = 400;   // Steps for back-and-forth travel from
center
int X_CYCLES      = 26;    // How many back-and-forth cycles to do

// Y axis target position (from home)
int Y_TARGET_STEPS = 500;

// Non-blocking step intervals (in microseconds)
// -> Lower intervals = faster speed
unsigned long X_STEP_INTERVAL_US = 500;
unsigned long Y_STEP_INTERVAL_US = 500;
unsigned long F_STEP_INTERVAL_US = 800; // For filler feed motor

// DC motor on/off control
// - This code only toggles ON/OFF, no PID speed control here.
// - If you need adjustable RPM, add separate logic for PWM or driver
with speed feedback.
bool dcMotorRunning = false;

// If you'd like the filler motor to step in ratio to X moves, set
something like:

```

```
int F_STEP_RATIO = 10;    // e.g. every 10 X steps we do 1 F step (used
in more advanced logic if needed)
```

```
// Simple encoder reading (polling) variables
// -> If you need high-speed encoder reading, consider using
interrupts or a library
volatile long encoderCount = 0;
int lastEncAState = 0;
int lastEncBState = 0;
```

```
/******
```

```
 * Variables for Non-Blocking Motion
```

```
******/
```

```
// X axis
```

```
unsigned long lastXStepTime    = 0; // last time (micros) we stepped X
long currentXPosition          = 0; // our running "position" for X in
steps
long targetXPosition           = 0; // where we want X to go (in steps)
bool  movingForward            = true; // direction state for X axis
cycles
int   xCycleCount              = 0;   // how many forward-back cycles
completed
```

```
// Y axis
```

```
unsigned long lastYStepTime    = 0;
long currentYPosition          = 0;
long targetYPosition           = 0;
```

```
// F axis (filler)
```

```
unsigned long lastFStepTime    = 0;
long currentFPosition          = 0;
long targetFPosition           = 0;
```

```
// System states
```

```
bool systemRunning = false;    // Whether the system is active (X
cycles, etc.)
```

```
/******
```

```
 *          setup()
 * - Initializes all pins
 * - Enables stepper drivers (EN_PIN)
 * - Turns on DC motor (if needed)
 * - Sets initial target for X and Y
 * - Preps for encoder read
```

```
******/
```

```

void setup() {
    Serial.begin(115200);

    // --- Pin Modes ---
    pinMode(DC_PIN,      OUTPUT);
    pinMode(ENCODER_A,   INPUT_PULLUP);
    pinMode(ENCODER_B,   INPUT_PULLUP);
    pinMode(EN_PIN,      OUTPUT);
    pinMode(X_STEP,      OUTPUT);
    pinMode(X_DIR,       OUTPUT);
    pinMode(Y_STEP,      OUTPUT);
    pinMode(Y_DIR,       OUTPUT);
    pinMode(F_STEP,      OUTPUT);
    pinMode(F_DIR,       OUTPUT);

    // Enable all steppers (active LOW)
    digitalWrite(EN_PIN, LOW);

    // Turn ON DC motor (for pipe rotation) -> LOW = ON (depends on your
    relay/driver)
    stopDCMotor(false);
    dcMotorRunning = true;

    // Move X to center
    // -> Set a target for X, we do not block/wait here,
    //     actual motion is handled in loop() by updateX()
    setXTarget(X_CENTER_STEPS, true);

    // Move Y to target
    setYTarget(Y_TARGET_STEPS, true);

    // Mark system as running so we can start doing X cycles once it
    arrives
    systemRunning = true;

    // Initialize encoder variables
    lastEncAState = digitalRead(ENCODER_A);
    lastEncBState = digitalRead(ENCODER_B);
    encoderCount  = 0;
}

/*****
*
*           loop()
* - Continuously updates the motion of X, Y, F
*   in a non-blocking manner using micros() timing
*****/

```

```

* - Polls the encoder for DC motor
* - Manages the X-axis back-and-forth cycles
*****/
void loop() {
    // 1) Simple polling for the DC motor encoder
    updateEncoder();

    // 2) Update each axis in a non-blocking manner
    updateX();
    updateY();
    updateF();

    // 3) Manage the cycle for X (back-and-forth motion)
    if (systemRunning) {
        manageXCycle();
    }

    // Additional logic can be placed here if needed,
    // e.g., adjusting filler feed speed based on gap measurement,
    // checking limit switches, or reading distance sensors, etc.
}

/*****
*
*          DC Motor (ON/OFF) Control
* - stopDCMotor(true) => turns DC motor OFF
* - stopDCMotor(false) => turns DC motor ON
* NOTE: This is a simple relay-based control, no speed control.
*****/
void stopDCMotor(bool stop) {
    // If your relay logic is reversed, adjust HIGH/LOW
    digitalWrite(DC_PIN, stop ? HIGH : LOW);
    dcMotorRunning = !stop;
}

/*****
*
*          setXTarget()
* - Sets the new target position for the X axis.
* - 'steps' is how many steps from the currentXPosition,
*   not from absolute zero.
* - 'forward' sets the DIR pin (HIGH or LOW).
*****/
void setXTarget(long steps, bool forward) {
    digitalWrite(X_DIR, forward ? HIGH : LOW);
    // If forward is true, we want currentXPosition + steps
    // If forward is false, we want currentXPosition - steps

```

```

    targetXPosition = (forward)
                        ? (currentXPosition + steps)
                        : (currentXPosition - steps);
}

/*****
 *          setYTarget()
 *   - Same logic as X but for Y axis
 *****/
void setYTarget(long steps, bool forward) {
    digitalWrite(Y_DIR, forward ? HIGH : LOW);
    targetYPosition = (forward)
                        ? (currentYPosition + steps)
                        : (currentYPosition - steps);
}

/*****
 *          setFTarget()
 *   - Same logic as X but for filler axis (F)
 *****/
void setFTarget(long steps, bool forward) {
    digitalWrite(F_DIR, forward ? HIGH : LOW);
    targetFPosition = (forward)
                        ? (currentFPosition + steps)
                        : (currentFPosition - steps);
}

/*****
 *          updateX(), updateY(), updateF()
 *   - Non-blocking step functions.
 *   - Each checks if enough time (micros) has passed
 *     since the last step to issue the next one.
 *   - Increments or decrements 'currentPosition'.
 *****/
void updateX() {
    unsigned long now = micros();
    bool forward = (targetXPosition > currentXPosition);

    // If X still hasn't reached its target
    if (currentXPosition != targetXPosition) {
        if (now - lastXStepTime >= X_STEP_INTERVAL_US) {
            lastXStepTime = now;

            // Step pin HIGH -> delay -> LOW
            digitalWrite(X_STEP, HIGH);

```

```

        delayMicroseconds(5); // short pulse
        digitalWrite(X_STEP, LOW);

        // Update currentXPosition
        currentXPosition += (forward ? 1 : -1);
    }
}

void updateY() {
    unsigned long now = micros();
    bool forward = (targetYPosition > currentYPosition);

    if (currentYPosition != targetYPosition) {
        if (now - lastYStepTime >= Y_STEP_INTERVAL_US) {
            lastYStepTime = now;

            digitalWrite(Y_STEP, HIGH);
            delayMicroseconds(5);
            digitalWrite(Y_STEP, LOW);

            currentYPosition += (forward ? 1 : -1);
        }
    }
}

void updateF() {
    unsigned long now = micros();
    bool forward = (targetFPosition > currentFPosition);

    if (currentFPosition != targetFPosition) {
        if (now - lastFStepTime >= F_STEP_INTERVAL_US) {
            lastFStepTime = now;

            digitalWrite(F_STEP, HIGH);
            delayMicroseconds(5);
            digitalWrite(F_STEP, LOW);

            currentFPosition += (forward ? 1 : -1);
        }
    }
}

/*****
*
*          manageXCycle()

```

```

*   - Checks if X has reached its target.
*   - If so, decides whether to reverse direction or
*     if we've finished all cycles.
*   - When all cycles done, it may stop the DC motor
*     and return X/Y to home, etc.
*   - This logic is an example. Adjust to your needs.
*****/
void manageXCycle() {
    // Only proceed if X reached its target
    if (currentXPosition == targetXPosition) {
        if (movingForward) {
            // We just arrived at the forward limit
            setXTarget(X_TRAVEL_STEPS, false); // go back
            movingForward = false;
        } else {
            // We just arrived back at the backward limit
            xCycleCount++;
            if (xCycleCount < X_CYCLES) {
                // Start next forward move
                setXTarget(X_TRAVEL_STEPS, true);
                movingForward = true;
            } else {
                // All cycles completed
                // Example: turn off DC motor, bring X & Y home, etc.
                stopDCMotor(true); // Off
                setYTarget(Y_TARGET_STEPS, false); // Return Y to home
                setXTarget(X_CENTER_STEPS, false); // Return X to home
            }
        }
    }
}

/*****
*           updateEncoder()
*   - Simple polling approach to detect encoder changes
*     on DC motor.
*   - If your motor rotates quickly, consider using
*     external interrupt or dedicated library.
*****/
void updateEncoder() {
    int encA = digitalRead(ENCODER_A);
    int encB = digitalRead(ENCODER_B);

    // If A changed, that indicates a possible "tick"
    if (encA != lastEncAState) {

```



```

// Determine direction by reading B
if (encB == encA) {
    encoderCount++;
} else {
    encoderCount--;
}
}

// Save state
lastEncAState = encA;
lastEncBState = encB;
}

```

گام‌های آتی و نکات پایانی

1. آزمایش و تست زیرسیستم‌ها

- چرخاندن لوله و گرفتن تصاویر برای اطمینان از سرعت مناسب و بدون لرزش.
 - تست محور Y برای نوسان نداشتن یا عدم "لرزش" هنگام اصلاح فاصله.
2. بهبود الگوریتم بینایی ماشین

- تنظیم نور و Exposure دوربین.
 - کالیبره کردن فاصله پیکسلی به میلی‌متری با یک الگوی مرجع.
3. ایمنی و مهندسی صنعتی

- تعبیه شستی اضطراری در مسیر تغذیه برق موتور یا رله جداگانه.
- استفاده از میکروسوئیچ‌ها در محور X و Y برای جلوگیری از برخورد یا اضافه‌حرکت (Overtravel).

4. تنظیم دقیق پارامترها

- سرعت‌ها و شتاب موتورهای پله‌ای بسته به مکانیک دستگاه و گشتاور لازم تغییر می‌کنند.
- دامنه مجاز سنسور فاصله بسته به نوع نازل و هندسه لوله قابل تنظیم است.

5. کنترل جریان و ولتاژ جوش

- اگر منبع جوشکاری (Welding Power Supply) دارید، باید پارامترهایی نظیر ولتاژ و آمپر جوش هم به‌صورت هماهنگ‌کننده مدیریت شوند؛ این موضوع از طریق کنترلر جداگانه یا افزودن قابلیت تنظیم در نرم‌افزار ممکن است.

با این ساختار و توضیحات، می‌توانید پروژه ربات جوشکاری لوله را گام‌به‌گام پیش ببرید. مستندات که در بالا ارائه شد، هم نقشه کلی سیستم است و هم نمونه‌ای کاربردی از کدنویسی آردوینو برای کنترل موتورها و سنسورها. موفق باشید!