# Synthetic Jet Actuator Project Overview

Nomi Wang

06/10/25

Presented at 2025 College of Engineering Research Open House

Undergraduate Departmental Award under Mechanical Engineering Division
& Third Place in People's Choice Award

## 1. Motivation

**Engineering Application:**

Synthetic jets (SJs) are widely utilized in various engineering domains due to their ability to generate controlled fluid motion without net mass injections. A notable application is in micro robotics for targeted drug delivery, where synthetic jets provide vortex-based propulsion. This enables microrobots to navigate efficiently within confined and complex biological environments, making them highly promising for biomedical interventions such as precise drug delivery (see Fig. 1).

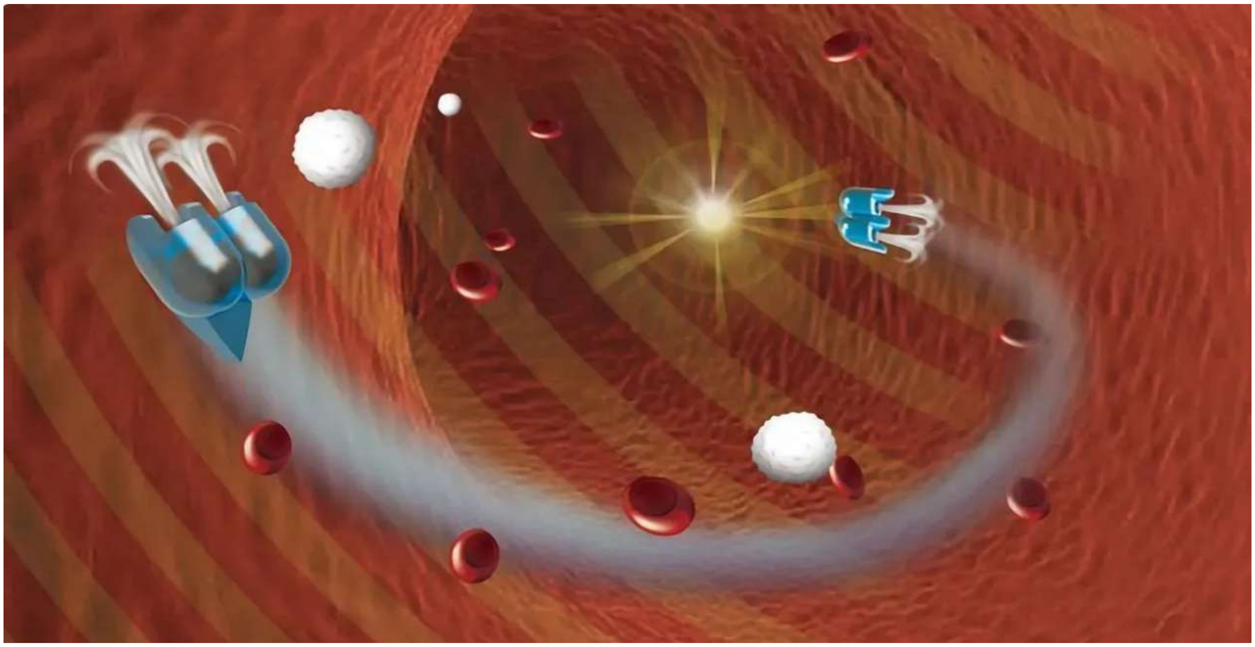


***Fig.1 Microrobot swimmers propelled through blood vessels via vortex-based propulsion, inspired by synthetic jet concept. Adapted from Technology Networks (https://www.technologynetworks.com/drug-discovery/news/micro-robot-swimmers-powered-by-ultrasound-353922).***

**Interactive Educational Tool and Scientific Outreach:**

This project is designed not only as a demonstration of synthetic jet behavior but also as a tool for scientific outreach. It enables users, especially students and the general public—to interactively explore the physics of synthetic jets and understand how their performance depends on actuation frequency.

## 2. System Design

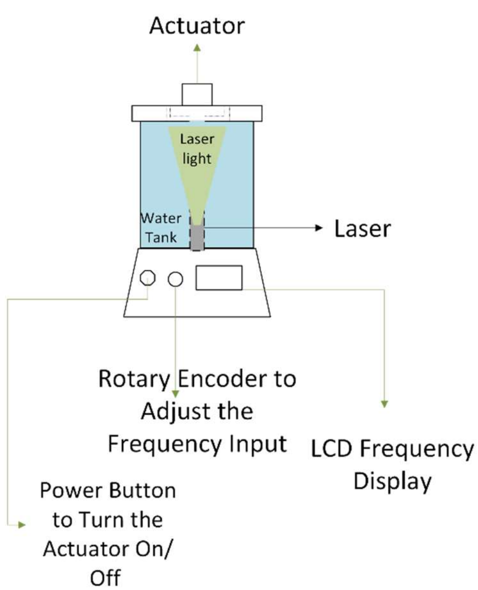
**Hardware Overview:**

ESP32 microcontroller, rotary encoder, LCD display, push button, amplifier, voice coil actuator, laser and cylindrical lens. (See Fig.2)

**Actuator Mechanism:**

The actuator consists of a sealed cavity with an orifice on one side and a flexible membrane on the opposite side. Driven by the ESP32's sinusoidal output, the membrane vibrates, creating cyclic suction and expulsion of water through the orifice. At optimal frequencies, this forms distinct vortex rings characteristic of synthetic jets. (See Fig.3)

**Circuit Design:** See Fig 4.



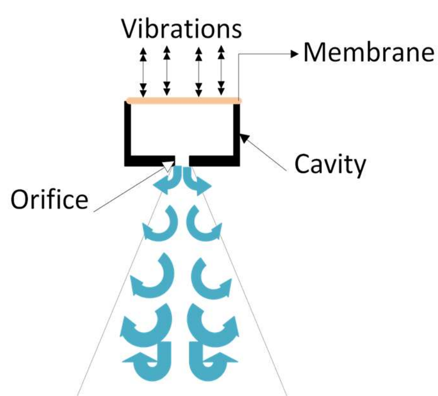




***Fig.2 (Left) System Overview Diagram of the Interactive Synthetic Jet Display***

***Fig.3(Right) Schematic Illustration of Synthetic Jet (SJ)***

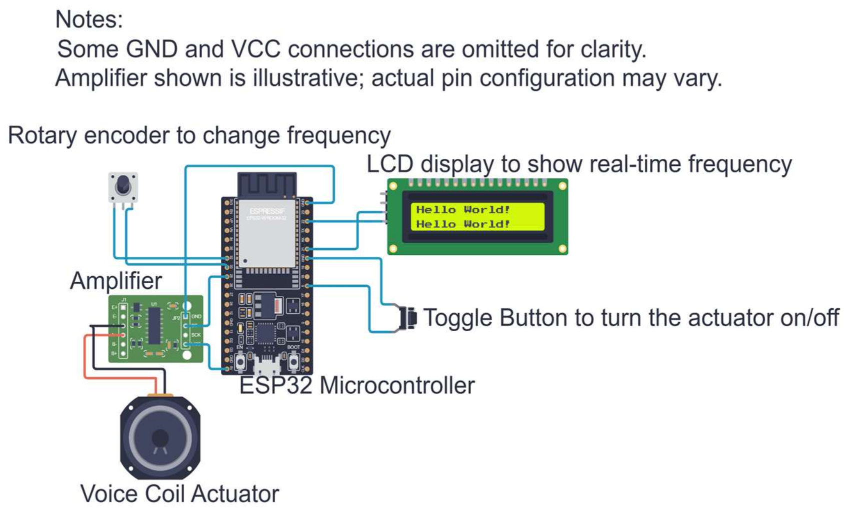



***Fig.4 Control Circuit Schematics for the Synthetic Jet Actuator System***

3. **Challenges:**

**<u>Timing synchronization of rotary encoder:</u>**

Early tests showed erratic frequency jumps because each encoder detent produces four fast edge transitions, with mechanical bounce adding unwanted signals. Our initial polling routine missed or duplicated pulses. **To fix this issue, we implemented quadrature state-machine** (See Fig.5)

**<u>Illumination for flow visualization:</u>**

Standard LEDs lacked sufficient intensity to visualize particle motion in water. We addressed this by employing a green laser paired with a cylindrical lens to generate a high-intensity laser sheet, effectively illuminating neutrally buoyant particles for improved visualization.

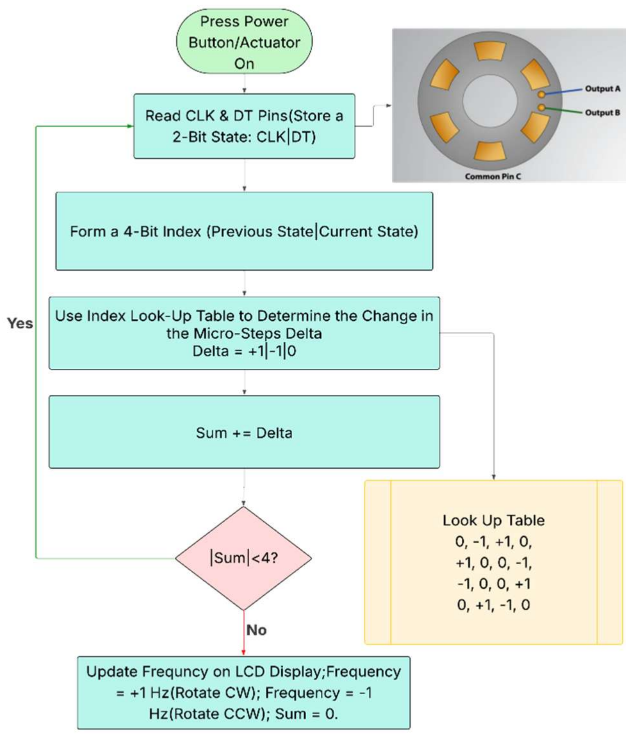



***Fig.5 A Simple Explanation of How Quadrature State-Machine Works.***

## 4. Conclusions & Results

**<u>Frequency-Dependent Flow Visualization:</u>**

Testing across frequencies from 1 to 20 Hz demonstrated clearer and more defined vortex-ring structures around 5Hz. Although the actuator's small scale limited visual clarity, we successfully demonstrated frequency-dependent behavior.

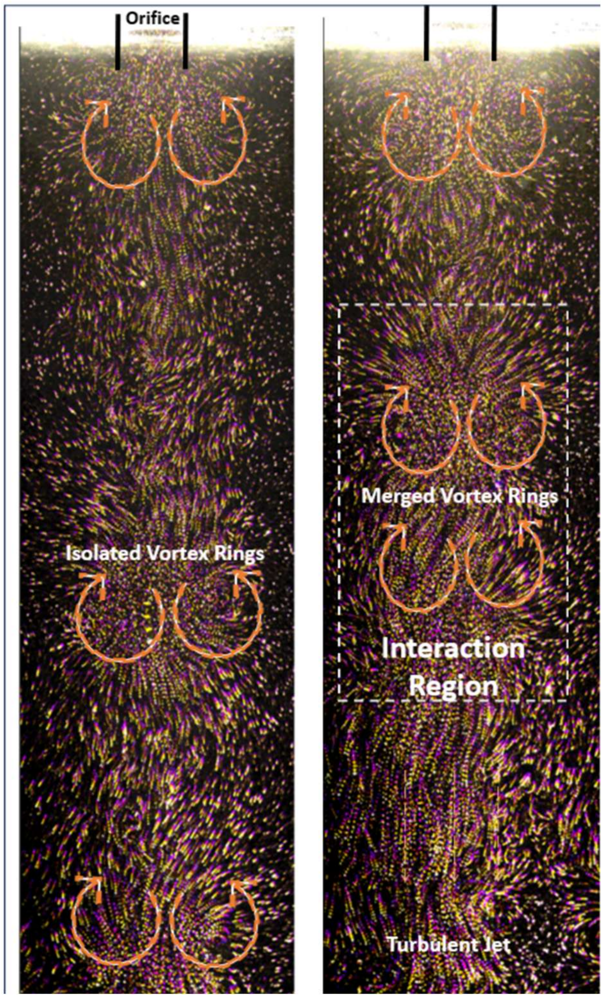


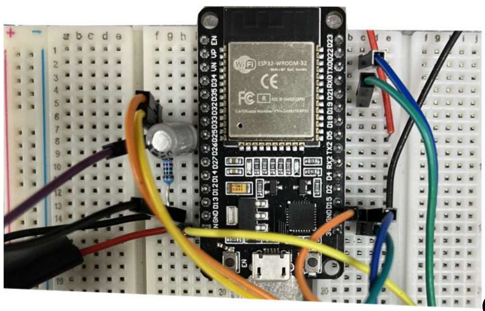

***Fig.6 Flow Pathline Visualization of Vortical Structures in Synthetic Jet (Left - 5Hz; Right - 10Hz)***

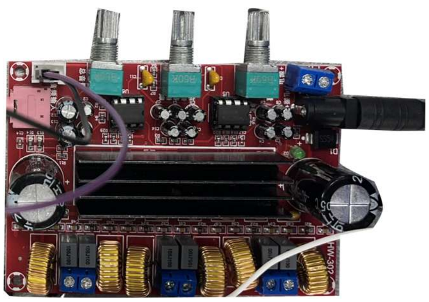
## 5. Acknowledgement

Special thanks to Professor Cong Wang, Stella Gerlock, Skinder A. Dar and Chukwudum Eluchie for their valuable support and contributions.

## Appendix A – Major Hardware Used in this Project


*(a) Oscillating Actuator*


*(b) ESP32 Microcontroller (Built in Digital-Analog Converter)*


*(c) Power Amplifier*


*(d) LCD Display for Frequency Feedback*

# Appendix B – Full Project Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <math.h>

// === LCD Setup ===
LiquidCrystal_I2C lcd(0x27, 16, 2);

// === Pin Assignments ===
const int DAC_OUTPUT = 25;
const int BTN_TOGGLE = 5;
const int ENCODER_CLK = 32;  // CLK wired to GPIO32
const int ENCODER_DT  = 33;  // DT wired to GPIO33

// === Frequency Settings ===
const int freqMin = 1;
const int freqMax = 20;
const int freqStep = 1;
int frequency = freqMin;

// === Sine Wave Parameters ===
const int amplitude       = 127;
const int offset          = 128;
const int samplesPerCycle = 300;
```

```
// === State Variables ===
bool     actuatorOn      = false;
int      lastCLKState    = HIGH;
unsigned long lastTogglePress = 0;
const int    debounceDelay    = 200;

// === Quadrature Accumulator ===
static int8_t  lastEncState   = 0;   // holds previous (CLK,DT) state
static int16_t encAccumulator = 0;   // sums +1/-1 per transition

// lookup table: (prev<<2|curr) → +1, -1 or 0
const int8_t enc_states[16] = {
   0, -1, +1,  0,   //0000, 0001, 0010, 0011
  +1,  0,  0, -1,   //0100, 0101, 0110, 0111
  -1,  0,  0, +1,   //1000, 1001, 1010, 1011
   0, +1, -1,  0    //1100, 1101, 1110, 1111
};

void setup() {
  Wire.begin(21, 22);        // I2C: SDA, SCL
  lcd.init();
  lcd.backlight();
```

```
 pinMode(BTN_TOGGLE, INPUT_PULLUP);
 pinMode(ENCODER_CLK, INPUT_PULLUP);
 pinMode(ENCODER_DT,  INPUT_PULLUP);

 // initialize quadrature state
 int msb = digitalRead(ENCODER_CLK);
 int lsb = digitalRead(ENCODER_DT);
 lastEncState = (msb << 1) | lsb;
 lastCLKState = msb;

 Serial.begin(115200);
 updateDisplay();


oid loop() {
 handleToggleButton();
 handleEncoder();

 if (actuatorOn) {
   generateSineWave(frequency);
 } else {
   dacWrite(DAC_OUTPUT, offset);
   delay(10);
```

```
void handleToggleButton() {
  static bool lastButtonState = HIGH;
  bool current = digitalRead(BTN_TOGGLE);

  if (current == LOW && lastButtonState == HIGH) {
    if (millis() - lastTogglePress > debounceDelay) {
      actuatorOn = !actuatorOn;

      //
      if (actuatorOn) {
        frequency = freqMin;       // reset to 1 Hz
        encAccumulator = 0;         // clear any encoder residue
      }
      //

      Serial.println(actuatorOn ? "Actuator ON" : "Actuator OFF");
      updateDisplay();
      lastTogglePress = millis();
    }
  }

  lastButtonState = current;
}
```

```
void handleEncoder() {
  int msb = digitalRead(ENCODER_CLK);
  int lsb = digitalRead(ENCODER_DT);
  int currEncState = (msb << 1) | lsb;
  int index = (lastEncState << 2) | currEncState;
  int8_t delta = enc_states[index];
  lastEncState = currEncState;

  if (delta != 0) {
    encAccumulator += delta;
    // full detent = 4 micro-steps
    if (encAccumulator >= 4) {
      frequency = constrain(frequency + freqStep, freqMin, freqMax);
      Serial.print("Full CW → freq = ");
      Serial.println(frequency);
      updateDisplay();
      encAccumulator = 0;
    }
    else if (encAccumulator <= -4) {
      frequency = constrain(frequency - freqStep, freqMin, freqMax);
      Serial.print("Full CCW → freq = ");
      Serial.println(frequency);
      updateDisplay();
```

```
      encAccumulator = 0;
    }
  }
}

void generateSineWave(int freq) {
  unsigned long startTime = micros();
  unsigned long period = 1000000UL / freq;

  for (int i = 0; i < samplesPerCycle; i++) {
    float angle = (2.0 * PI * i) / samplesPerCycle;
    int value = offset + amplitude * sin(angle);
    dacWrite(DAC_OUTPUT, value);

    unsigned long target = startTime + (period * i) / samplesPerCycle;
    while (micros() < target) {
      // keep button & encoder responsive even during waveform
      handleToggleButton();
      handleEncoder();
    }
  }
}
```

```
void updateDisplay() {
  lcd.clear();
  lcd.setCursor(0, 0);
  if (actuatorOn) {
    lcd.print("Frequency:");
    lcd.setCursor(0, 1);
    lcd.print(frequency);
    lcd.print(" Hz");
  } else {
    lcd.print("Actuator OFF");
  }
}
```