گزارش کار دوقلوی دیجیتال دستگاه HOT WIRE

محمد مهدی اسدی

امیرحسین سلیمانی

استاد مربوطه: عارف کریمی افشار

خرداد ۱۴۰۴

فهرست

چکیده

این گزارش به تشریح فرآیند طراحی، ساخت و پیاده‌سازی یک سیستم دوقلوی دیجیتال برای یک دستگاه برش هات‌وایر می‌پردازد. هدف اصلی این پروژه، ایجاد یک مدل مجازی و زنده از دستگاه فیزیکی است که بتواند وضعیت عملکردی آن، شامل موقعیت مکانی بازوهای متحرک و شرایط محیطی (دما، فشار و رطوبت)، را به‌صورت آنی (Real-time) نمایش دهد. برای این منظور، از یک معماری مبتنی بر اینترنت اشیاء استفاده شده است. داده‌های مکانی توسط چهار سنسور سونار و داده‌های محیطی توسط یک سنسور BMP280 جمع‌آوری شده و از طریق دو میکروکنترلر ESP8266 با استفاده از پروتکل MQTT به یک بروکر مرکزی ارسال می‌گردند. پلتفرم Node-RED وظیفه دریافت، پردازش اولیه و ارسال نهایی داده‌ها به یک شبیه‌ساز سه‌بعدی که در موتور بازی‌سازی Unity توسعه یافته است را بر عهده دارد. نتایج پروژه نشان‌دهنده موفقیت در همگام‌سازی مدل دیجیتال با دستگاه فیزیکی و ایجاد بستری برای نظارت از راه دور و تحلیل عملکرد آن است.

مقدمه

انقلاب صنعتی چهارم و اینترنت اشیاء

جهان در آستانه چهارمین انقلاب صنعتی (Industry 4.0) قرار دارد که مشخصه اصلی آن، ادغام دنیای فیزیکی و دیجیتال از طریق سیستم‌های سایبر-فیزیکی (Cyber-Physical Systems) است. در قلب این تحول، فناوری **اینترنت اشیاء (Internet of Things - IoT)** قرار دارد. اینترنت اشیاء به شبکه‌ای از اشیاء فیزیکی اطلاق می‌شود که به سنسورها، نرم‌افزارها و سایر فناوری‌ها مجهز شده‌اند تا بتوانند از طریق اینترنت با سایر دستگاه‌ها و سیستم‌ها ارتباط برقرار کرده و داده تبادل کنند. این فناوری امکان نظارت، کنترل و بهینه‌سازی فرآیندها را از راه دور فراهم می‌آورد و صنایع مختلف از تولید و بهداشت گرفته تا کشاورزی و شهرهای هوشمند را متحول کرده است.

مفهوم دوقلوی دیجیتال (Digital Twin) و کاربردهای آن

یکی از قدرتمندترین مفاهیم برآمده از اینترنت اشیاء، **دوقلوی دیجیتال** است. دوقلوی دیجیتال یک نمایش مجازی و پویا از یک شیء، فرآیند یا سیستم فیزیکی است. این مدل مجازی به‌صورت آنی با همتای فیزیکی خود از طریق جریان داده‌های ارسال‌شده از سنسورها همگام‌سازی می‌شود. برخلاف یک مدل سه‌بعدی ایستا، دوقلوی دیجیتال "زنده" است و وضعیت، عملکرد و شرایط محیطی موجودیت فیزیکی را در لحظه بازتاب می‌دهد.

کاربردهای دوقلوی دیجیتال بسیار گسترده است:

- **نظارت آنی :(Real-time Monitoring)** مدیران و اپراتورها می‌توانند عملکرد یک دارایی فیزیکی را بدون حضور فیزیکی در محل، مشاهده و بررسی کنند.
- **تعمیر و نگهداری پیش‌بینانه :(Predictive Maintenance)** با تحلیل داده‌های تاریخی و فعلی، می‌توان زمان احتمالی خرابی یک قطعه را پیش‌بینی و قبل از وقوع آن اقدام به تعمیر کرد.
- **شبیه‌سازی و تحلیل "چه می‌شود اگر :(What-if Analysis) "**می‌توان سناریوهای مختلف را روی مدل دیجیتال آزمایش کرد بدون آنکه ریسک یا هزینه‌ای متوجه دستگاه فیزیکی شود.
- **آموزش اپراتورها :**کارمندان جدید می‌توانند در یک محیط مجازی امن و کنترل‌شده با نحوه کار دستگاه آشنا شوند.

شرح مسئله: نیاز به نظارت و شبیه‌سازی دستگاه هات‌وایر

دستگاه‌های برش هات‌وایر (سیم داغ) در صنایع مختلفی مانند ماکت‌سازی، تولید بسته‌بندی‌های فومی (یونولیت) و نمونه‌سازی سریع کاربرد دارند. در این دستگاه‌ها، یک سیم فلزی داغ برای برش مواد نرم مانند فوم استفاده می‌شود. دقت حرکت بازوهایی که سیم را نگه داشته‌اند و همچنین شرایط محیطی مانند دما و رطوبت، تأثیر مستقیمی بر کیفیت برش نهایی دارد .در حالت سنتی، نظارت بر عملکرد این دستگاه‌ها نیازمند حضور فیزیکی اپراتور است. این امر امکان نظارت مستمر و تحلیل داده‌های عملکردی را محدود می‌کند. پروژه حاضر با هدف رفع این محدودیت، به دنبال ساخت یک دوقلوی دیجیتال برای یک دستگاه هات‌وایر است.

اهداف پروژه

اهداف اصلی این پروژه به شرح زیر تعریف شده‌اند:

1. **طراحی و ساخت یک سیستم جمع‌آوری داده :**ایجاد یک سیستم سخت‌افزاری برای اندازه‌گیری دقیق موقعیت بازوهای دستگاه و شرایط محیطی.
2. **انتقال بی‌سیم و مطمئن داده‌ها :**استفاده از پروتکل استاندارد اینترنت اشیاء (MQTT) برای ارسال داده‌های سنسورها به یک سرور مرکزی.
3. **پردازش و مدیریت داده‌ها :**استفاده از یک پلتفرم منعطف (Node-RED) برای دریافت، پردازش اولیه و مسیریابی داده‌ها.
4. **ایجاد یک مدل مجازی پویا :**توسعه یک شبیه‌ساز سه‌بعدی در محیط Unity که نمایانگر دستگاه فیزیکی باشد.
5. **همگام‌سازی آنی :**اطمینان از اینکه مدل مجازی به‌صورت لحظه‌ای با تغییرات در دستگاه فیزیکی به‌روزرسانی می‌شود.

معماری سیستم و اجزای مورد استفاده

نمای کلی معماری سیستم

معماری این پروژه از چهار لایه اصلی تشکیل شده است:

1. **لایه فیزیکی :(Perception Layer)** شامل دستگاه هات‌وایر و سنسورهای متصل به آن برای جمع‌آوری داده‌های خام از دنیای واقعی.
2. **لایه شبکه :(Network Layer)** وظیفه انتقال امن و مطمئن داده‌ها از میکروکنترلرها به سرور مرکزی را با استفاده از شبکه Wi-Fiو پروتکل MQTT بر عهده دارد.
3. **لایه پردازش :(Processing Layer)** در این لایه، پلتفرم Node-RED داده‌های خام را از بروکر MQTT دریافت کرده، آن‌ها را پردازش، تجمیع و برای لایه کاربرد آماده می‌کند.
4. **لایه کاربرد :(Application Layer)** این لایه شامل دوقلوی دیجیتال توسعه‌یافته در Unity است که داده‌های پردازش‌شده را دریافت و به‌صورت یک مدل سه‌بعدی پویا به کاربر نمایش می‌دهد.

اجزای سخت‌افزاری

۱ .میکروکنترلر :ESP8266 این میکروکنترلر به دلیل داشتن قابلیت اتصال به Wi-Fi داخلی، قیمت بسیار مناسب، مصرف انرژی پایین و جامعه کاربری بزرگ، برای این پروژه انتخاب شد. وجود دو عدد از این ماژول‌ها امکان تقسیم وظایف را فراهم می‌کند: یکی برای کنترل دو سنسور سونار اول و دیگری برای کنترل دو سنسور دوم به همراه سنسور محیطی. این تقسیم‌بندی به مدیریت بهتر کد و جلوگیری از تداخل در خواندن سنسورها کمک می‌کند.

۲ .سنسورهای اولتراسونیک (سونار) :مدل رایج این سنسورها) مانند (HC-SR04 با ارسال یک پالس صوتی و اندازه‌گیری زمان بازگشت آن، فاصله تا یک مانع را محاسبه می‌کند. در این پروژه، چهار سنسور سونار برای تعیین موقعیت دقیق دو سر سیم داغ به کار رفته‌اند. با فرض اینکه هر سر سیم در یک محور) مثلاً X یا (Y حرکت می‌کند، دو سنسور می‌توانند موقعیت آن را با دقت بالایی اندازه‌گیری کنند (یکی فاصله از یک طرف و دیگری از طرف دیگر برای افزایش دقت و کاهش خطا).

۳ .ماژول سنسور محیطی :BMP280 این ماژول یک سنسور دقیق برای اندازه‌گیری **دما (Temperature)** و **فشار بارومتریک (Barometric Pressure)**است. (توجه: مدل BME280 علاوه بر این دو، **رطوبت (Humidity)** را نیز اندازه‌گیری می‌کند و احتمالاً منظور شما این مدل بوده است که در گزارش فرض بر آن گذاشته می‌شود). این داده‌ها برای دوقلوی دیجیتال ارزشمند هستند، زیرا تغییرات دما و رطوبت می‌تواند بر انبساط یا انقباض مواد و همچنین کیفیت فرآیند برش تأثیر بگذارد.

پروتکل‌ها و پلتفرم‌های نرم‌افزاری

۱ .پروتکل MQTT و بروکر Mosquitto: MQTT (Message Queuing Telemetry Transport) یک پروتکل پیام‌رسانی سبک مبتنی بر مدل ناشر-مشترک (Publish-Subscribe) است که برای دستگاه‌های با منابع محدود و شبکه‌های با پهنای باند کم طراحی شده است. در این پروژه:

- **کلاینت‌ها :(Clients)** دو ماژول ESP8266 به عنوان ناشر (Publisher) عمل کرده و داده‌های خود را منتشر می‌کنند . Node-REDو Unity (به‌طور غیرمستقیم) به عنوان مشترک (Subscriber) داده‌ها را دریافت می‌کنند.
- **بروکر Mosquitto :(Broker)** به عنوان بروکر MQTT انتخاب شده است. این نرم‌افزار متن‌باز، یک سرور سبک و قدرتمند است که وظیفه دریافت پیام‌ها از ناشران و توزیع آن‌ها بین مشترکین علاقه‌مند به هر "تاپیک (Topic) "را بر عهده دارد. تاپیک‌ها مانند کانال‌های ارتباطی عمل می‌کنند (مثلاً .(hotwire/motor1/distance

۲ .پلتفرم Node-RED: Node-RED یک ابزار برنامه‌نویسی بصری مبتنی بر جریان (Flow-based) است که برای اتصال دستگاه‌های سخت‌افزاری، APIها و سرویس‌های آنلاین به یکدیگر طراحی شده است. در این پروژه، Node-REDنقش یک "چسب دیجیتال" را ایفا می‌کند:

- **گره ورودی :MQTT** سه گره (Node) برای اشتراک در تاپیک‌های مربوط به فواصل موتور اول، فواصل موتور دوم و داده‌های محیطی تنظیم شده‌اند.
- **گره‌های پردازشی :**ممکن است نیاز باشد داده‌های خام سنسورها (مثلاً زمان پالس سونار) به واحد سانتی‌متر تبدیل شوند یا داده‌های مختلف در یک پیام واحد) مثلاً با فرمت (JSON بسته‌بندی شوند.
- **گره خروجی :TCP** پس از پردازش، داده‌ها در قالب یک پیام از طریق پروتکل TCP به آدرس IP و پورت مشخصی که اپلیکیشن Unityدر آن منتظر دریافت داده است، ارسال می‌شود.

۳ .موتور بازی‌سازی Unity: Unity یک پلتفرم قدرتمند برای ساخت محتوای سه‌بعدی و تعاملی است. انتخاب آن برای ساخت دوقلوی دیجیتال به دلایل زیر بوده است:

- **رندرینگ باکیفیت :**قابلیت ایجاد مدل‌های سه‌بعدی واقع‌گرایانه.
- **اسکریپت‌نویسی با :C#** زبان برنامه‌نویسی قدرتمند برای پیاده‌سازی منطق سیستم.
- **پشتیبانی از شبکه :**امکان ایجاد کلاینت TCP برای دریافت داده از.Node-RED

۴ .پروتکل ارتباطی TCP/IP: برای ارتباط نهایی بین Node-RED و Unity از TCP استفاده شده است TCP .یک پروتکل اتصال‌گرا (Connection-oriented)و قابل اطمینان است که تضمین می‌کند داده‌ها به ترتیب و بدون خطا به مقصد می‌رسند. این ویژگی برای ارسال دستورات حرکتی و داده‌های حیاتی به شبیه‌ساز ضروری است.

جزئیات پیاده‌سازی

پیکربندی سخت‌افزار و اتصالات

برد اول :دو سنسور سونار به پین‌های GPIO در ESP8266 اول متصل شدند. هر سنسور به دو پین نیاز دارد Trig :(برای ارسال پالس) و Echo(برای دریافت بازگشت). تغذیه VCC و GND سنسورها نیز به ترتیب به پین‌های 3.3 Vو GND بورد متصل شد .

برد دوم :دو سنسور سونار دیگر به همان روش به ESP8266 دوم متصل شدند. سنسور BMP280 نیز از طریق پروتکل ارتباطی I2C به پین‌های SDA (D2) و SCL (D1) این بورد متصل گردید.

برنامه‌نویسی میکروکنترلرها(Firmware)

کد نوشته‌شده برای ماژول‌های ESP8266 با استفاده از محیط Arduino IDE توسعه یافت. منطق اصلی کد به شرح زیر است:

1. **اتصال به شبکه :**در بخش ,()setupکد ابتدا به شبکه Wi-Fi مشخص‌شده متصل می‌شود.
2. **اتصال به بروکر :MQTT** پس از اتصال بهWi-Fi ، کلاینت MQTT برای اتصال به آدرس IP و پورت بروکر Mosquitto پیکربندی می‌شود.
3. **حلقه اصلی :(Loop)**

- خواندن داده از سنسورهای سونار با ارسال یک پالس کوتاه به پین Trig و اندازه‌گیری مدت زمان HIGH بودن پین .Echoاین زمان سپس به فاصله (سانتی‌متر) تبدیل می‌شود.
- در بورد دوم، داده‌های دما، فشار و رطوبت از سنسور BMP280 با استفاده از کتابخانه مربوطه) مثلاً Adafruit BMP280 Library)خوانده می‌شود.
- داده‌ها در قالب یک رشته یا فرمت JSON آماده می‌شوند.
- با استفاده از تابع,()mqttClient.publish داده‌ها به تاپیک‌های مشخص‌شده روی بروکر (مثلاً hotwire/motor1/pos, hotwire/environment)منتشر می‌شوند.
- یک تأخیر کوتاه (delay) برای جلوگیری از ارسال بیش از حد داده‌ها در حلقه قرار داده می‌شود.

```ino
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>  // Use BMP280 instead of BME280

#define SEALEVELPRESSURE_HPA (1015.5) // example local sea level pressure

// I2C Pins
#define BMP_SDA 5  // D1 (GPIO5)
#define BMP_SCL 4  // D2 (GPIO4)

// Sonar Pins
const int trigPin_1 = 14;  // D5 (GPIO14)
const int echoPin_1 = 12;  // D6 (GPIO12)
const int trigPin_2 = 13;  // D7 (GPIO13)
const int echoPin_2 = 15;  // D8 (GPIO15)

Adafruit_BMP280 bmp;  // BMP280 object

// Timing Variables
unsigned long lastSend = 0;
const long sendInterval = 100;  // 100ms for all sensors

// WiFi & MQTT
const char* ssid = "testwifi";
const char* password = "naminami";
const char* mqtt_server = "192.168.78.177";

// MQTT Topics
const char* clientID = "ESP8266_Cube_2";
const char* sonarTopic = "cube/2/data";
const char* tempTopic = "cube/3/data";
const char* willTopic = "cube/2/data";
const char* willMessage = "offline";

WiFiClient espClient;
PubSubClient mqttClient(espClient);
void setup() {
  Serial.begin(9600);
  Serial.println("\nBooting Cube Sensor Board 2...");

  // Initialize sonar pins
  pinMode(trigPin_1, OUTPUT);
  pinMode(echoPin_1, INPUT);
  pinMode(trigPin_2, OUTPUT);
  pinMode(echoPin_2, INPUT);
```

```
  // Initialize I2C
  Wire.begin(BMP_SDA, BMP_SCL);
  delay(100);

  // --- I2C Scan for all devices ---
  Serial.println("Scanning I2C bus...");
  for (byte addr = 1; addr < 127; addr++) {
    Wire.beginTransmission(addr);
    if (Wire.endTransmission() == 0) {
      Serial.print("Found device at 0x");
      if (addr < 16) Serial.print("0");
      Serial.println(addr, HEX);
    }
  }

  // --- Try BMP280 Initialization ---
  if (!initializeBMP280()) {
    Serial.println("BMP280 initialization failed. Halting.");
    while (1);
  }

  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,
                  Adafruit_BMP280::SAMPLING_X2,
                  Adafruit_BMP280::SAMPLING_X16,
                  Adafruit_BMP280::FILTER_X16,
                  Adafruit_BMP280::STANDBY_MS_500);


  Serial.println("BMP280 initialization successful.");

  setup_wifi();
  mqttClient.setServer(mqtt_server, 1883);
  mqttClient.setCallback(callback);
}

// ... rest of code unchanged ...

bool initializeBMP280() {
  Serial.println("Initializing BMP280...");

  // Try default address (0x76)
  if (bmp.begin(0x76)) {
    Serial.println("Found BMP280 at 0x76");
    return true;
  }
```

```
  // Try alternative address (0x77)
  if (bmp.begin(0x77)) {
    Serial.println("Found BMP280 at 0x77");
    return true;
  }

  // Scan I2C if init failed
  Serial.println("Scanning I2C bus...");
  for (byte addr = 1; addr < 127; addr++) {
    Wire.beginTransmission(addr);
    if (Wire.endTransmission() == 0) {
      Serial.print("Found device at 0x");
      if (addr < 16) Serial.print("0");
      Serial.println(addr, HEX);
    }
  }

  return false;
}
void loop() {
  if (!mqttClient.connected()) {
    reconnect();
  }
  mqttClient.loop();

  unsigned long currentMillis = millis();

  if (currentMillis - lastSend >= sendInterval) {
    // Read sonar distances
    long dist1 = getSonarDistance(trigPin_1, echoPin_1);
    long dist2 = getSonarDistance(trigPin_2, echoPin_2);

    // Read BMP280 data
    float temp = bmp.readTemperature();
    float presRaw = bmp.readPressure();            // Pa
    float alt = bmp.readAltitude(SEALEVELPRESSURE_HPA); // meters

    // Check for NaNs
    if (isnan(temp)) Serial.println("Temperature is NaN");
    if (isnan(presRaw)) Serial.println("Pressure is NaN");
    if (isnan(alt)) Serial.println("Altitude is NaN");

    // Convert pressure to hPa
    float pres = presRaw / 100.0;
```

```
    // Debug prints
    Serial.print("Temperature (°C): "); Serial.println(temp);
    Serial.print("Raw Pressure (Pa): "); Serial.println(presRaw);
    Serial.print("Converted Pressure (hPa): "); Serial.println(pres);
    Serial.print("Altitude (m): "); Serial.println(alt);

    // Publish sonar data
    String sonarPayload = "{\"board\":2,\"sensors\":[";
    sonarPayload += "{\"id\":1,\"value\":" + String(dist1) + "},";
    sonarPayload += "{\"id\":2,\"value\":" + String(dist2) + "}";
    sonarPayload += "]}";
    mqttClient.publish(sonarTopic, sonarPayload.c_str());

    // Publish temperature, pressure, and altitude data
    String tempPayload = "{\"temp\":" + String(temp, 1) +
                         ",\"pressure\":" + String(pres, 1) +
                         ",\"altitude\":" + String(alt, 1) + "}";
    mqttClient.publish(tempTopic, tempPayload.c_str());

    // Print JSON payloads
    Serial.println("Sonar: " + sonarPayload);
    Serial.println("BMP280: " + tempPayload);

    lastSend = currentMillis;
  }

  delay(10);
}

long getSonarDistance(int trigPin, int echoPin) {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  return pulseIn(echoPin, HIGH) * 0.034 / 2;
}
```

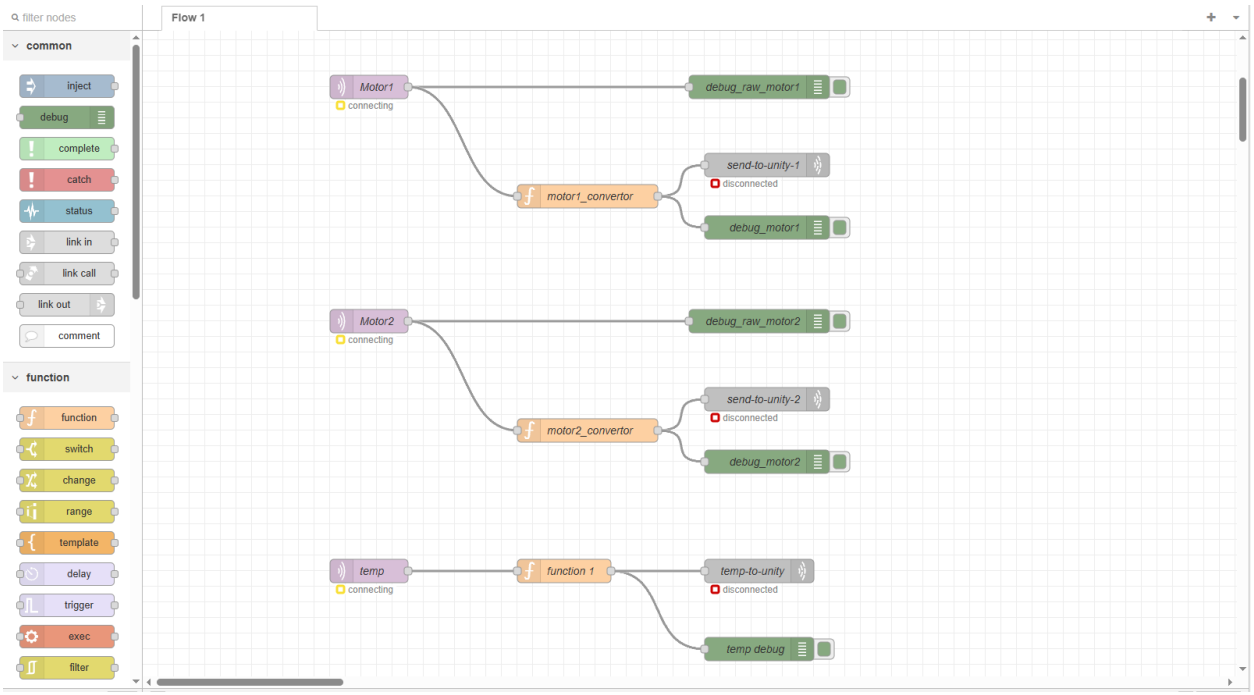
```
178  void setup_wifi() {
179    delay(10);
180    Serial.print("Connecting to ");
181    Serial.println(ssid);
182
183    WiFi.begin(ssid, password);
184    while (WiFi.status() != WL_CONNECTED) {
185      delay(500);
186      Serial.print(".");
187    }
188    Serial.println("\nWiFi connected");
189    Serial.print("IP: ");
190    Serial.println(WiFi.localIP());
191  }
192
193  void callback(char* topic, byte* payload, unsigned int length) {
194    // Handle incoming MQTT messages if needed
195  }
196
197  void reconnect() {
198    while (!mqttClient.connected()) {
199      Serial.print("Attempting MQTT connection...");
200      if (mqttClient.connect(clientID, willTopic, 1, true, willMessage)) {
201        Serial.println("connected");
202        mqttClient.publish(willTopic, "online", true);
203      } else {
204        Serial.print("failed, rc=");
205        Serial.print(mqttClient.state());
206        Serial.println(" retrying in 5s");
207        delay(5000);
208      }
209    }
210  }
```

طراحی فلو (Flow) در Node-RED

فلو طراحی‌شده در Node-RED شامل سه مسیر اصلی است:

1. **مسیر داده‌های موتور اول :**یک گره mqtt inبه تاپیک مربوط به فواصل موتور اول مشترک می‌شود. داده دریافتی به یک گره functionارسال می‌شود تا در صورت نیاز پردازش شود.
2. **مسیر داده‌های موتور دوم :**مشابه مسیر اول، اما برای تاپیک موتور دوم.
3. **مسیر داده‌های محیطی :**یک گره mqtt inبه تاپیک داده‌های محیطی مشترک می‌شود.

داده‌های خروجی از این سه مسیر در یک گره joinتجمیع می‌شوند تا یک پیام واحد و کامل ایجاد کنند. این پیام سپس به فرمت مورد نظر Unityمثلاً یک رشته CSV مانند ("pos1,pos2,temp,pressure,humidity")تبدیل شده و در نهایت توسط یک گره tcp out به کلاینت Unity ارسال می‌گردد.



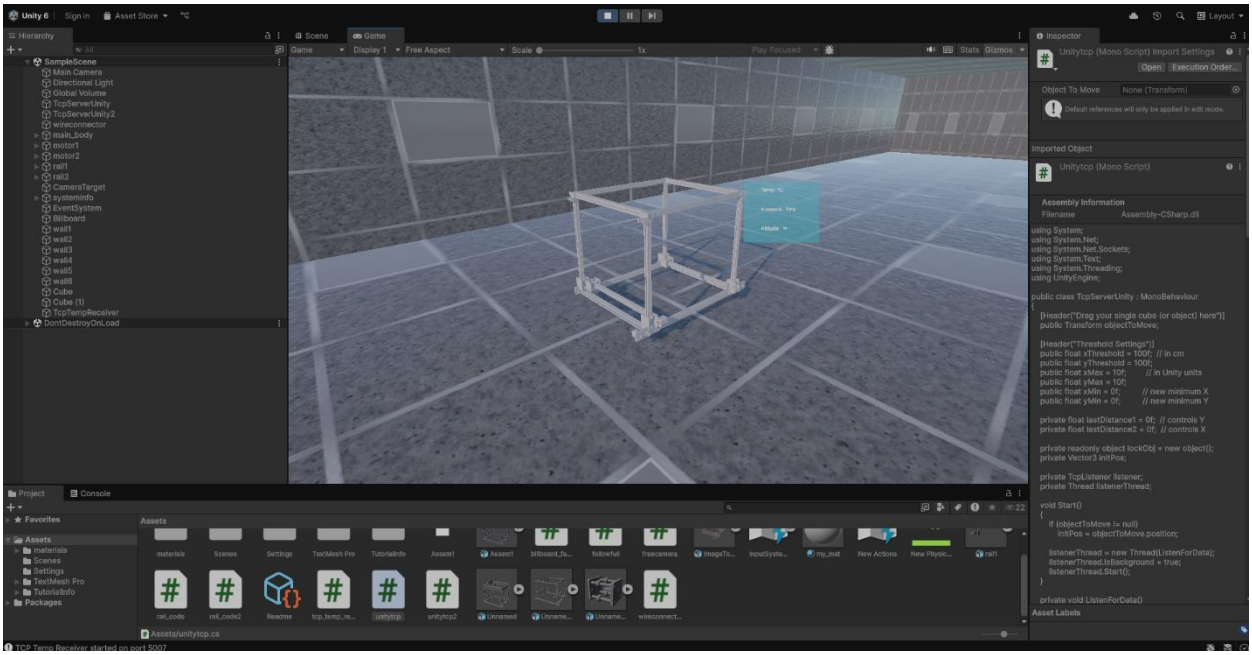
توسعه شبیه‌ساز در Unity

طراحی صحنه :(Scene) یک مدل سه‌بعدی ساده از دستگاه هات‌وایر، شامل پایه، بازوها و سیم برش، در Unity طراحی یا وارد شد .

اسکریپت ارتباطی :(TCP Client) یک اسکریپت C# نوشته شد که در متد ()Start یک کلاینت TCP ایجاد کرده و به آدرس IP و پورت Node-RED متصل می‌شود. این اسکریپت در یک حلقه ()Update به‌طور مداوم منتظر دریافت داده می‌ماند .

پردازش داده و به‌روزرسانی مدل : پس از دریافت پیام از Node-RED ، اسکریپت آن را تجزیه (Parse) می‌کند. مقادیر عددی استخراج‌شده برای به‌روزرسانی استفاده می‌شوند :

- مقادیر فاصله، موقعیت (Transform.position) اشیاء سه‌بعدی نماینده بازوها را تغییر می‌دهند.
- مقادیر دما، فشار و رطوبت در المان‌های رابط کاربری (UI Text) روی صفحه نمایش داده می‌شوند.



نتایج و تحلیل

عملکرد سیستم و همگام‌سازی

سیستم نهایی با موفقیت پیاده‌سازی شد. حرکت بازوهای فیزیکی دستگاه هات‌وایر با تأخیر بسیار ناچیزی (کمتر از چند صد میلی‌ثانیه) در مدل دیجیتال در Unity منعکس می‌شد. داده‌های محیطی نیز به‌درستی بر روی صفحه نمایش شبیه‌ساز به‌روزرسانی می‌شدند. این نتایج نشان داد که معماری انتخاب‌شده و اجزای به کار رفته برای دستیابی به اهداف پروژه کاملاً مناسب بوده‌اند.

چالش‌های پیاده‌سازی

نویز سنسور سونار :در برخی موارد، سنسورهای سونار به دلیل بازتاب‌های ناخواسته از محیط، مقادیر نویزی یا پرت تولید می‌کردند. این مشکل با پیاده‌سازی یک فیلتر نرم‌افزاری ساده (مانند میانگین‌گیری از چند قرائت متوالی) در کد ESP8266 تا حد زیادی برطرف شد .

پایداری اتصال :Wi-Fi در نسخه‌های اولیه، قطعی‌های لحظه‌ای Wi-Fi باعث توقف ارسال داده می‌شد. با افزودن کدی برای بررسی وضعیت اتصال و تلاش برای اتصال مجدد در صورت قطعی، پایداری سیستم افزایش یافت .

فرمت‌بندی داده :هماهنگ کردن فرمت دقیق داده ارسالی از Node-RED و فرمت مورد انتظار در اسکریپت Unity نیازمند دقت و آزمون و خطای اولیه بود. استفاده از فرمت استاندارد JSON در نسخه‌های بعدی می‌تواند این فرآیند را ساده‌تر کند.

پیشنهادها برای توسعه آینده

ارتباط دوطرفه :می‌توان قابلیتی را به Unity اضافه کرد که کاربر بتواند از طریق مدل مجازی، دستورات حرکتی را به دستگاه فیزیکی ارسال کند .

ذخیره‌سازی داده‌های تاریخی :می‌توان Node-RED را به یک پایگاه داده سری زمانی (Time-series Database) مانند InfluxDB متصل کرد تا تمام داده‌های سنسورها برای تحلیل‌های بعدی ذخیره شوند .

هوش مصنوعی و یادگیری ماشین :با تحلیل داده‌های تاریخی، می‌توان مدل‌های یادگیری ماشین برای پیش‌بینی خرابی (Predictive Maintenance)یا بهینه‌سازی پارامترهای برش توسعه داد .

واقعیت افزوده :(AR) به جای نمایش روی صفحه، می‌توان دوقلوی دیجیتال را با استفاده از فناوری AR روی دستگاه فیزیکی واقعی نمایش داد.

نتیجه‌گیری

این پروژه با موفقیت نشان داد که چگونه می‌توان با استفاده از قطعات سخت‌افزاری ارزان‌قیمت و پلتفرم‌های نرم‌افزاری متن‌باز، یک سیستم دوقلوی دیجیتال کارآمد و پویا ایجاد کرد. معماری مبتنی برIoT ، از سنسورها تا پلتفرم شبیه‌سازی، انعطاف‌پذیری و قابلیت توسعه بالایی را فراهم می‌کند. دوقلوی دیجیتال ساخته‌شده نه تنها امکان نظارت آنی بر دستگاه هات‌وایر را فراهم می‌کند، بلکه بستری قدرتمند برای شبیه‌سازی، تحلیل عملکرد و پیاده‌سازی راهکارهای صنعتی هوشمند در آینده است. این پروژه نمونه‌ای عملی از قدرت فناوری‌های Industry 4.0 در بهینه‌سازی فرآیندهای تولیدی محسوب می‌شود.

منابع

- Datasheet for ESP8266EX
- Datasheet for HC-SR04 Ultrasonic Sensor
- Datasheet for Bosch BMP280 Sensor
- MQTT Version 3.1.1 Documentation (mqtt.org)
- Mosquitto: An Open Source MQTT Broker (mosquitto.org)
- Node-RED Documentation (nodered.org)
- Unity Real-Time Development Platform Documentation (unity.com)