BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

XE TỰ HÀNH

Môn Học: Hệ Thống Nhúng

GVHD: GV. Nguyễn Xuân Sâm

Mã môn học: ESYS431080

Nhóm sinh viên thực hiện:

1. Lê Trung Hậu 19110028
2. Nguyễn Duy Đạt 22110307

Thành phố Hồ Chí Minh, tháng 5, năm 2025

XE TỰ HÀNH

Lê Trung Hậu
19110028
Khoa Công nghệ thông tin
Trường ĐH Sư Phạm Kỹ Thuật
TP.HCM
19110028@student.hcmute.edu.vn

Nguyễn Duy Đạt
222110307
Khoa Công nghệ thông tin
Trường ĐH Sư Phạm Kỹ Thuật
TP.HCM
22110307@student.hcmute.edu.vn

Tóm tắt - Báo cáo trình bày quá trình phát triển mô hình xe tự hành, tập trung vào khả năng dò line và né vật cản sử dụng cảm biến và thuật toán điều khiển cơ bản. Nội dung bao gồm thiết kế phần cứng, lập trình phần mềm, xây dựng mô hình thực nghiệm và kiểm thử. Kết quả cho thấy xe có thể tự động di chuyển theo đường line, phát hiện và né vật cản. Báo cáo đề xuất các hướng phát triển để cải thiện hiệu suất và độ ổn định.

I. Giới thiệu

Trong bối cảnh công nghệ tự động hóa ngày càng phát triển, các hệ thống xe tự hành đóng vai trò quan trọng trong việc thử nghiệm và ứng dụng các giải pháp điều khiển tự động. Đề tài này được lựa chọn với mục tiêu khám phá khả năng xây dựng một mô hình xe tự hành đơn giản, có thể tự động dò line và né vật cản, từ đó đóng góp vào việc nghiên cứu các công nghệ điều khiển cơ bản cho xe tự hành trong tương lai.

Lý do chọn đề tài xuất phát từ nhu cầu thực tiễn trong việc phát triển các hệ thống tự động hóa nhỏ gọn, chi phí thấp, phù hợp cho việc nghiên cứu. Mô hình xe tự hành sử dụng vi điều khiển ESP32 giúp hiểu rõ các nguyên lý điều khiển mà còn tạo nền tảng cho các ứng dụng thực tế như robot di chuyển trong nhà máy hoặc phương tiện giao hàng tự động. Việc tích hợp giao diện web để giám sát và điều khiển từ xa cũng đáp ứng xu hướng kết nối IoT hiện nay.

II. Phân tích thiết kế hệ thống

2.1. Sơ đồ khối hệ thống

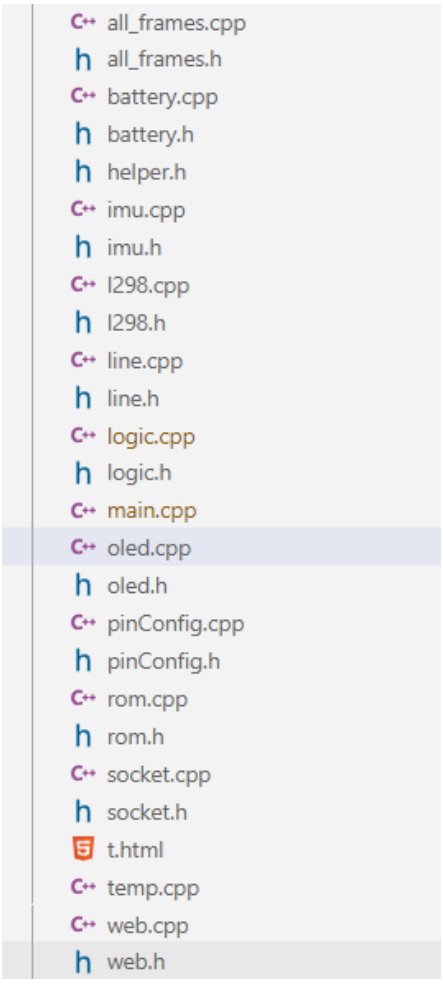
2.2. Các thành phần chính

Phần cứng:

- Vi điều khiển ESP32.
- Module dò line 5 led.
- Cảm biến siêu âm HC-SR04.
- Driver động cơ L298N.
- Màn hình OLED 128x64.
- Cảm biến IMU

III. Cấu trúc mã nguồn

3.1. Cấu trúc thư mục



3.2. Giải thích các module chính

1. Module điều khiển chính (main.cpp):

- Khởi tạo và quản lý các thành phần hệ thống
- Xử lý các trạng thái hoạt động: dò line, điều khiển thủ công, hiệu chuẩn
- Tích hợp cảm biến siêu âm để tránh vật cản
- Quản lý kết nối WiFi và giao tiếp web

```
#include <WiFi.h>                    // TV kết nối WiFi cho ESP32
#include "web.h"                     // TV xử lý web server
#include "socket.h"                  // TV xử lý giao tiếp Socket
#include "pinConfig.h"               // Cấu hình chân (pin) phần cứng
#include "imu.h"                     // TV cảm biến IMU
#include "l298.h"                    // TV điều khiển động cơ thông qua L298
#include "line.h"                    // TV đọc line
#include "oled.h"                    // TV điều khiển màn hình OLED
#include <HCSR04.h>                  // TV đo khoảng cách bằng cảm biến siêu âm
#include <Wire.h>                    // TV giao tiếp I2C
#include "battery.h"                 // TV theo dõi pin
#include "logic.h"                   // TV xử lý logic điều khiển
#include "rom.h"                     // TV xử lý ROM để lưu dữ liệu hiệu chuẩn

// Khởi tạo các đối tượng
UltraSonicDistanceSensor        sonic(sonicTrig, sonicEcho); // Cảm biến siêu âm
Oled oled;                           // Màn hình OLED
Imu imu;                             // Cảm biến IMU
Logic lg;                           // Xử lý logic
L298 motor;                          // Động cơ sử dụng L298
Line line;                           // Cảm biến dò line
Web web;                            // Web server
Socket socket;                       // Giao tiếp Socket
Battery battery;                     // Theo dõi điện áp pin
Rom rom;                             // Lưu dữ liệu hiệu chuẩn
SocketState pevState = DISCONNECTED; // Trạng thái socket trước đó

// Hàm điều khiển robot dò line
void lineForward()
{
  float dist = sonic.measureDistanceCm(); // Đo khoảng cách bằng cảm biến siêu âm

  if (dist < 17.0) // Nếu vật cản cách dưới 17cm
  {
    bool ok = lg.avoidObstacle(&motor, &sonic, &oled, &line); // Thực hiện tránh vật cản
    if (!ok){
    }
    return;
  }

  float error = line.getLineError(); // Tính sai số lệch line
    int    *speeds    =    lg.computeFE(error, motor.getBaseSpeed()); // Tính tốc độ hai bánh dựa trên sai số
   motor.move(speeds[0],  speeds[1]);  //  Điều khiển động cơ
  oled.printMotor(speeds);           // Hiển thị tốc độ động cơ
  oled.printLineError(dist);         // Hiển thị khoảng cách

   socket.boardcastLine(String(dist));  //  Gửi khoảng cách lên server qua socket
}

void setup()
{
  Serial.begin(115200);             // Khởi động Serial
  while (!Serial) delay(10);       // Chờ Serial sẵn sàng

  rom.init();                       // Khởi tạo EEPROM
  Wire.begin(21, 22);               // Khởi tạo I2C với chân SDA=21, SCL=22
  oled.init();                      // Khởi tạo màn hình OLED
  oled.showBar(0);                  // Hiển thị tiến trình

  battery.init();                   // Khởi tạo pin
  oled.showBar(12);                 // Cập nhật tiến trình

  pinMode(buzzerPin, OUTPUT);       // Cấu hình chân còi
  digitalWrite(buzzerPin, 0);       // Tắt còi
  oled.showBar(24);

  imu.init(&rom);                   // Khởi tạo cảm biến IMU, đọc dữ liệu từ EEPROM
  imu.update();                     // Cập nhật dữ liệu từ IMU
  oled.showBar(36);

  line.init(&rom);                  // Khởi tạo cảm biến dò line, đọc dữ liệu hiệu chuẩn
                oled.setLine(line.getLowLine(), line.getHighLine()); // Hiển thị ngưỡng vạch đen/trắng
  oled.showBar(48);

   WiFi.softAP(wifiSsid, wifiPassword); // Tạo mạng WiFi AP cho robot
  delay(1000);                          // Chờ WiFi ổn định
  oled.showBar(60);

  lg.init();                        // Khởi tạo logic điều khiển

  socket.initSocket(&line, &motor); // Khởi tạo socket và liên kết với line và motor
  oled.showBar(80);

  web.begin();                      // Khởi động web server
  delay(1000);
  oled.showBar(100);
```

```
  oled.hello();                       // Hiển thị
màn hình chào
}

void loop()
{
  web.handle();                         // Xử lý
request từ web
  socket.soketLoop();                 // Xử lý dữ
liệu từ socket
  SocketState state = socket.getState(); // Lấy
trạng thái hiện tại
  oled.clear();                       // Xóa màn
hình OLED

  switch (state)
  {
  case LINE_FOLOW: // Trạng thái dò line
    if (pevState != state)
    {
      oled.debug("Start \n FOLOW "); // Hiển thị
trạng thái bắt đầu dò line
       lg.startLogic();                 // Khởi
động logic điều khiển
    }
    lineForward();                    // Gọi hàm
dò line
    break;

  case CONTROL: // Trạng thái điều khiển từ xa
     int *move = socket.getMove();     // Lấy dữ
liệu điều khiển từ socket
      motor.move(move[0],  move[1]);     // Điều
khiển động cơ
    oled.printMotor(move);            // Hiển thị
tốc độ động cơ
    break;

  case START_CALIBRATE: // Bắt đầu hiệu chuẩn
line
    if (pevState != state)
    {
      oled.debug("Start \n CALIBRATE"); // Hiển
thị trạng thái hiệu chuẩn
       line.calibrateLine(true);          // Bắt
đầu hiệu chuẩn lần đầu
    }
    else
    {
      line.calibrateLine(false);        // Tiếp
tục hiệu chuẩn
    }
    int *move = socket.getMove();       // Điều
khiển tay trong khi hiệu chuẩn
    motor.move(move[0], move[1]);
    oled.printMotor(move);
    break;

  case END_CALIBRATE: // Kết thúc hiệu chuẩn
    motor.move(0, 0);                   // Dừng
động cơ
     line.saveCalibration();              // Lưu
dữ liệu hiệu chuẩn vào EEPROM
     socket.emitLine();                   // Gửi
dữ liệu line lên server
                oled.setLine(line.getLowLine(),
line.getHighLine()); // Cập nhật OLED
    break;

  default:
    break;
  }

  pevState = state; // Cập nhật trạng thái trước
  oled.printState(state);                     //
Hiển thị trạng thái hiện tại
  oled.printBattery(battery.getBatteryVoltage()
); // Hiển thị pin
   oled.printLine(line.getRawLine());         //
Hiển thị dữ liệu raw của line
    oled.printWifiInfo(wifiSsid,   wifiPassword,
WiFi.softAPIP().toString()); // Hiển thị WiFi
  oled.displayUpdate();                   // Cập
nhật màn hình
}
```

2. Module dò line (line.cpp)

- Đọc và xử lý tín hiệu từ cảm biến dò line
- Tính toán sai số vị trí so với line
- Hỗ trợ hiệu chuẩn cảm biến

```
#include "line.h"

Line::Line()
{
}

void Line::getLine()
{
  // Đọc giá trị từ 5 cảm biến line và lưu kết
quả là true/false dựa trên ngưỡng đã hiệu chỉnh
  for (int i = 0; i < 5; i++)
  {
    int value = analogRead(_pin[i]); // Đọc giá
trị analog từ chân cảm biến
    _line[i] = value > _calibrateLine[i] ? true
: false; // So sánh với ngưỡng để xác định có
line hay không
  }
}

int *Line::getRawLine()
{
  // Trả về mảng chứa giá trị thô từ 5 cảm biến
line
  static int rawLine[5];
  for (int i = 0; i < 5; i++)
  {
    rawLine[i] = analogRead(_pin[i]); // Đọc giá
trị analog từ từng cảm biến
  }
  return rawLine;
}

int *Line::getLowLine()
{
  // Trả về mảng giá trị thấp nhất của cảm biến
(dùng trong hiệu chuẩn)
  return _lowLine;
}

int *Line::getHighLine()
{
  // Trả về mảng giá trị cao nhất của cảm biến
(dùng trong hiệu chuẩn)
  return _highLine;
}

void Line::updateLowLine(int lowLine[5])
{
  // Cập nhật giá trị thấp nhất nếu giá trị mới
nhỏ hơn giá trị cũ
  for (int i = 0; i < 5; i++)
```

```
  {
    _lowLine[i] = min(_lowLine[i], lowLine[i]);
  }
}

void Line::updateHighLine(int highLine[5])
{
  // Cập nhật giá trị cao nhất nếu giá trị mới
lớn hơn giá trị cũ
  for (int i = 0; i < 5; i++)
  {
        _highLine[i]    =    max(_highLine[i],
highLine[i]);
  }
}

void Line::updateCalibrateLine()
{
  // Cập nhật giá trị ngưỡng hiệu chuẩn bằng
trung bình giữa low và high
  for (int i = 0; i < 5; i++)
  {
       _calibrateLine[i]   =   (_lowLine[i]   +
_highLine[i]) / 2;
  }
}

float Line::getLineError()
{
  // Tính sai số (error) theo trọng số để điều
khiển robot bám line
  getLine();
  const float weights[5] = {-2.0, -1.0, 0.0, 1.0,
2.0}; // Trọng số từ trái sang phải
  float error = 0.0;
  int sum = 0;

  for (int i = 0; i < 5; i++)
  {
    if (_line[i])
    {
      error += weights[i]; // Cộng dồn trọng số
      sum++;
    }
  }

  if (sum == 0)
  {
    _lineState = LINE_LOST; // Mất line
    return 5.0; // Sai số lớn để xử lý ngoại lệ
  }

  if (sum == 5)
  {
    _lineState = LINE_JUNCTION; // Giao điểm (tất
cả cảm biến đều thấy line)
    return 0.0;
  }

  float finalError = error / sum; // Tính sai số
trung bình có trọng số

  // Cập nhật trạng thái line dựa trên sai số
  _lineState = finalError < -0.5 ? LINE_LEFT :
(finalError > 0.5 ? LINE_RIGHT : LINE_CENTERED);
  return finalError;
}

LineState Line::getLineState()
{
  // Trả về trạng thái line hiện tại (trái, phải,
giữa, mất line,...)
  return _lineState;
}

void Line::calibrateLine(bool first)
{
  // Hiệu chuẩn cảm biến line: nếu là lần đầu
thì khởi tạo, còn lại thì cập nhật
  if (first)
  {
    int *raw = this->getRawLine();
    for (int i = 0; i < 5; i++)
    {
      _lowLine[i] = raw[i];  // Khởi tạo giá trị
thấp
     _highLine[i] = raw[i]; // Khởi tạo giá trị
cao
    }
  }
  else
  {
    int *raw = this->getRawLine();
    this->updateLowLine(raw);  // Cập nhật giá
trị thấp
    this->updateHighLine(raw); // Cập nhật giá
trị cao
  }
}

void Line::saveCalibration()
{
  // Lưu giá trị hiệu chuẩn vào ROM
  _rom->setLowLine(_lowLine);
  _rom->setHighLine(_highLine);
  _rom->save();
}

void Line::init(Rom *rom)
{
  // Khởi tạo đối tượng Line và đọc giá trị hiệu
chuẩn từ ROM
  _rom = rom;

  // Gán chân cảm biến line
  _pin[0] = line1Pin;
  _pin[1] = line2Pin;
  _pin[2] = line3Pin;
  _pin[3] = line4Pin;
  _pin[4] = line5Pin;

  int *lowLineInit = _rom->getLowLine();     //
Đọc giá trị lowLine từ ROM
  int *highLineInit = _rom->getHighLine();   //
Đọc giá trị highLine từ ROM

  for (int i = 0; i < 5; i++)
  {
    pinMode(_pin[i], INPUT_PULLUP);          //
Cấu hình chân đọc cảm biến là input pull-up
    _lowLine[i] = lowLineInit[i];            //
Gán giá trị low
    _highLine[i] = highLineInit[i];          //
Gán giá trị high
    _calibrateLine[i] = 512;                 //
Mặc định ngưỡng là 512
    _line[i] = false;                        //
Gán mặc định chưa phát hiện line
  }

   this->updateCalibrateLine();  //  Cập  nhật
ngưỡng hiệu chuẩn ban đầu
  this->printInfo();            // In thông tin
ra Serial
}

void Line::printInfo()
{
```

```
  // In ra Serial toàn bộ thông tin cảm biến line
    Serial.println("=====   Line   Sensor   Info =====");

  Serial.print("Pins: ");
  for (int i = 0; i < 5; i++)
  {
    Serial.print(_pin[i]);
    Serial.print(" ");
  }
  Serial.println();

  Serial.print("Raw: ");
  for (int i = 0; i < 5; i++)
  {
    Serial.print(_line[i]);
    Serial.print(" ");
  }
  Serial.println();

  Serial.print("Low: ");
  for (int i = 0; i < 5; i++)
  {
    Serial.print(_lowLine[i]);
    Serial.print(" ");
  }
  Serial.println();

  Serial.print("High: ");
  for (int i = 0; i < 5; i++)
  {
    Serial.print(_highLine[i]);
    Serial.print(" ");
  }
  Serial.println();

  Serial.print("Calibrated: ");
  for (int i = 0; i < 5; i++)
  {
    Serial.print(_calibrateLine[i]);
    Serial.print(" ");
  }
  Serial.println();

  Serial.print("Line State: ");
  switch (_lineState)
  {
  case LINE_LEFT:
    Serial.println("LEFT"); // Phát hiện line ở bên trái
    break;
  case LINE_RIGHT:
    Serial.println("RIGHT"); // Phát hiện line ở bên phải
    break;
  case LINE_CENTERED:
    Serial.println("CENTERED"); // Ở giữa line
    break;
  case LINE_LOST:
    Serial.println("LOST"); // Mất line
    break;
  default:
    Serial.println("UNKNOWN");
    break;
  }

  Serial.print("Line Error: ");
  Serial.println(getLineError(), 2); // In sai số line
}

int *Line::getPin()
{
  // Trả về mảng các chân kết nối cảm biến line
  return _pin;
}

Line::~Line()
{
}
```

3. Module điều khiển động cơ (l298.cpp)

- Điều khiển 2 động cơ DC thông qua driver L298N
- Hỗ trợ điều chỉnh tốc độ PWM
- Các chế độ di chuyển: tiến, lùi, rẽ trái, rẽ phải

```
#include "l298.h"

// Kênh PWM cho ESP32 (ESP32 có 16 kênh PWM: 0-15)
#define PWM_CHANNEL_A 0
#define PWM_CHANNEL_B 1
#define PWM_FREQ 1000    // Tần số PWM 1kHz (tùy chọn)
#define PWM_RESOLUTION 8 // Độ phân giải 8 bit → giá trị 0-255

// Hàm khởi tạo lớp L298
L298::L298()
{
    _battery.init();        // Khởi tạo cảm biến pin
    _t45 = time45;          // Thời gian xoay 45 độ
    _t90 = time90;          // Thời gian xoay 90 độ
    _timeForward = timeForward; // Thời gian tiến lên
    _baseVolt = baseVolt;   // Điện áp cơ bản
    _enableA = enableAPin;  // Chân PWM động cơ A
    _enableB = enableBPin;  // Chân PWM động cơ B
    _inputA1 = inputA1Pin;  // Chân điều khiển động cơ A1
    _inputA2 = inputA2Pin;  // Chân điều khiển động cơ A2
    _inputB1 = inputB1Pin;  // Chân điều khiển động cơ B1
    _inputB2 = inputB2Pin;  // Chân điều khiển động cơ B2

    // Thiết lập các chân output
    pinMode(_inputA1, OUTPUT); // Chân A1 là output
    pinMode(_inputA2, OUTPUT); // Chân A2 là output
    pinMode(_inputB1, OUTPUT); // Chân B1 là output
    pinMode(_inputB2, OUTPUT); // Chân B2 là output

    // Thiết lập PWM cho ESP32
        ledcSetup(PWM_CHANNEL_A,    PWM_FREQ, PWM_RESOLUTION); // Cấu hình kênh PWM A
        ledcSetup(PWM_CHANNEL_B,    PWM_FREQ, PWM_RESOLUTION); // Cấu hình kênh PWM B
    ledcAttachPin(_enableA, PWM_CHANNEL_A); // Gắn chân enable A vào kênh PWM A
    ledcAttachPin(_enableB, PWM_CHANNEL_B); // Gắn chân enable B vào kênh PWM B
}

// Thiết lập các giá trị thời gian và điện áp
```

```
void L298::setValue(int t45, int t90, int TF,
float baseV)
{
    _t45 = t45;               // Cập nhật thời
gian xoay 45 độ
    _t90 = t90;               // Cập nhật thời
gian xoay 90 độ
    _timeForward = TF;        // Cập nhật thời
gian tiến lên
    _baseVolt = baseV;       // Cập nhật điện áp
cơ bản
}

// Lấy tốc độ cơ bản dựa trên điện áp pin
int L298::getBaseSpeed()
{
    float desiredVolt = _baseVolt; // Điện áp
mong muốn
       int   speed   =   (int)((desiredVolt   /
_battery.getBatteryVoltage())  *  386.3636);  //
Tính tốc độ
    speed = constrain(speed, 0, maxSpeed); //
Giới hạn tốc độ
    return speed;            // Trả về tốc độ cơ
bản
}

// Lấy thời gian xoay 90 độ
int L298::getT90()
{
    return _t90;             // Trả về thời gian
xoay 90 độ
}

// Tiến lên để tránh chướng ngại vật
void L298::forwardAvoid(bool sub)
{
   this->forward(this->getBaseSpeed()); // Tiến
lên với tốc độ cơ bản
    if (sub)
    {
        delay(_timeForward * 0.6); // Chờ thời
gian ngắn hơn (60%)
    }
    else
    {
        delay(_timeForward);   // Chờ thời gian
đầy đủ
    }
    this->stop();             // Dừng robot
}

// Dừng động cơ
void L298::stop()
{
   ledcWrite(PWM_CHANNEL_A, 0); // Tắt PWM động
cơ A
    ledcWrite(PWM_CHANNEL_B, 0); // Tắt PWM động
cơ B
}

// Tiến lên
void L298::forward(int speed)
{
    this->move(speed, speed); // Di chuyển cả
hai động cơ cùng tốc độ
}

// Lùi lại
void L298::backward(int speed)
{
    digitalWrite(_inputA1, LOW);  // Đặt chiều
quay ngược cho động cơ A
    digitalWrite(_inputA2, HIGH);
    digitalWrite(_inputB1, LOW);  // Đặt chiều
quay ngược cho động cơ B
    digitalWrite(_inputB2, HIGH);
    ledcWrite(PWM_CHANNEL_A, speed); // Đặt tốc
độ cho động cơ A
    ledcWrite(PWM_CHANNEL_B, speed); // Đặt tốc
độ cho động cơ B
}

// Xoay trái
void L298::left(int speed)
{
    this->move(-speed, speed); // Động cơ trái
quay ngược, phải quay thuận
}

// Xoay phải
void L298::right(int speed)
{
    this->move(speed, -speed); // Động cơ trái
quay thuận, phải quay ngược
}

// Điều khiển tốc độ hai động cơ
void L298::move(int leftSpeed, int rightSpeed)
{
    // Thiết lập chiều quay
    digitalWrite(_inputA1, leftSpeed > 0); //
Động cơ A quay thuận nếu tốc độ dương
    digitalWrite(_inputA2, leftSpeed < 0); //
Động cơ A quay ngược nếu tốc độ âm
    digitalWrite(_inputB1, rightSpeed > 0); //
Động cơ B quay thuận nếu tốc độ dương
    digitalWrite(_inputB2, rightSpeed < 0); //
Động cơ B quay ngược nếu tốc độ âm

    // Đặt tốc độ PWM, dừng nếu tốc độ nhỏ hơn
30
   ledcWrite(PWM_CHANNEL_A, abs(leftSpeed) > 30
? abs(leftSpeed) : 0);
    ledcWrite(PWM_CHANNEL_B, abs(rightSpeed) >
30 ? abs(rightSpeed) : 0);
}

// Xoay trái 90 độ
void L298::turnLeft90(bool lock)
{
                  float       curVolt       =
_battery.getBatteryVoltage(); // Lấy điện áp pin
hiện tại
    float desiredVolt = 3.4; // Điện áp mong
muốn
    int speed = (int)((desiredVolt / curVolt) *
386.3636); // Tính tốc độ
    speed = constrain(speed, 0, maxSpeed); //
Giới hạn tốc độ

    if (lock)
    {
        this->move(0, speed); // Chỉ quay động
cơ phải
        delay(_t90 * 2);       // Chờ gấp đôi
thời gian
    }
    else
    {
        this->move(-speed, speed); // Động cơ
trái quay ngược, phải quay thuận
        delay(_t90);           // Chờ thời gian
xoay 90 độ
    }
    this->stop();             // Dừng robot
}
```

```
// Xoay trái 45 độ
void L298::turnLeft45()
{
                    float        curVolt        =
_battery.getBatteryVoltage(); // Lấy điện áp pin
hiện tại
     float desiredVolt = 3.4; // Điện áp mong
muốn
    int speed = (int)((desiredVolt / curVolt) *
386.3636); // Tính tốc độ
     speed = constrain(speed, 0, maxSpeed); //
Giới hạn tốc độ
    this->move(-speed, speed); // Động cơ trái
quay ngược, phải quay thuận
    delay(_t45);                // Chờ thời gian
xoay 45 độ
    this->stop();              // Dừng robot
}

// Xoay phải 90 độ
void L298::turnRight90(bool lock)
{
                    float        curVolt        =
_battery.getBatteryVoltage(); // Lấy điện áp pin
hiện tại
     float desiredVolt = 3.4; // Điện áp mong
muốn
    int speed = (int)((desiredVolt / curVolt) *
386.3636); // Tính tốc độ
     speed = constrain(speed, 0, maxSpeed); //
Giới hạn tốc độ
    if (lock)
    {
         this->move(speed, 0); // Chỉ quay động
cơ trái
          delay(_t90 * 2);       // Chờ gấp đôi
thời gian
    }
    else
    {
          this->move(speed, -speed); // Động cơ
trái quay thuận, phải quay ngược
         delay(_t90);           // Chờ thời gian
xoay 90 độ
    }
    this->stop();              // Dừng robot
}

// Xoay phải 45 độ
void L298::turnRight45()
{
                    float        curVolt        =
_battery.getBatteryVoltage(); // Lấy điện áp pin
hiện tại
     float desiredVolt = 3.4; // Điện áp mong
muốn
    int speed = (int)((desiredVolt / curVolt) *
386.3636); // Tính tốc độ
     speed = constrain(speed, 0, maxSpeed); //
Giới hạn tốc độ
    this->move(speed, -speed); // Động cơ trái
quay thuận, phải quay ngược
    delay(_t45);                // Chờ thời gian
xoay 45 độ
    this->stop();              // Dừng robot
}

// Kiểm tra động cơ
void L298::test(Oled *oled)
{
    for (int i = 500; i < 1200; i += 100) // Thử
các khoảng thời gian khác nhau
    {
       oled->debug("left " + String(i)); // Hiển
thị thông báo xoay trái
       this->testLeft(i);                // Thử
xoay trái
        delay(3000);                     // Chờ
3 giây
         oled->debug("right " + String(i)); //
Hiển thị thông báo xoay phải
        this->testRigh(i);                // Thử
xoay phải
        delay(3000);                     // Chờ
3 giây
    }
}

// Thử xoay trái với thời gian xác định
void L298::testLeft(int time)
{
                    float        curVolt        =
_battery.getBatteryVoltage(); // Lấy điện áp pin
hiện tại
     float desiredVolt = 3.4; // Điện áp mong
muốn
    int speed = (int)((desiredVolt / curVolt) *
386.3636); // Tính tốc độ
     speed = constrain(speed, 0, maxSpeed); //
Giới hạn tốc độ
    this->move(-speed, speed); // Động cơ trái
quay ngược, phải quay thuận
    delay(time);               // Chờ thời gian
xác định
    this->stop();              // Dừng robot
}

// Thử xoay phải với thời gian xác định
void L298::testRigh(int time)
{
                    float        curVolt        =
_battery.getBatteryVoltage(); // Lấy điện áp pin
hiện tại
     float desiredVolt = 3.4; // Điện áp mong
muốn
    int speed = (int)((desiredVolt / curVolt) *
386.3636); // Tính tốc độ
     speed = constrain(speed, 0, maxSpeed); //
Giới hạn tốc độ
    this->move(speed, -speed); // Động cơ trái
quay thuận, phải quay ngược
    delay(time);               // Chờ thời gian
xác định
    this->stop();              // Dừng robot
}

// Hàm hủy lớp L298
L298::~L298()
{
}
```

4. Module xử lý logic (logic.cpp)

- Thuật toán để điều khiển xe đi theo line
- Xử lý tránh vật cản
- Tối ưu hóa chuyển động

```
#include "logic.h"
#include <Arduino.h>

// Hàm khởi tạo lớp Logic
Logic::Logic()
{
}

// Khởi tạo các thông số PID và đặt lại các biến
void Logic::init()
```

```
{
  _Kp = Kp;           // Thiết lập hệ số tỉ lệ
  _Ki = Ki;           // Thiết lập hệ số tích phân
  _Kd = Kd;           // Thiết lập hệ số vi phân
  _integral = 0;      // Đặt lại giá trị tích phân
  _lastError = 0;     // Đặt lại lỗi trước đó
  _lineIsGlitch = 0;  // Đặt lại bộ đếm lỗi tín hiệu
}

// Tính toán tốc độ động cơ bằng điều khiển PID để theo đường
int   *Logic::compute(float   lineError,   int baseSpeed)
{
  int speed = baseSpeed;
  static int motorSpeeds[2];

  // Xử lý lỗi phát hiện đường
  if (lineError >= 99.0)
  {
    // Cảnh báo: Có thể xảy ra lỗi phát hiện đường
    _lineIsGlitch++;
    if (_lineIsGlitch > 5)
    {
      motorSpeeds[0] = 0;  // Dừng động cơ trái
      motorSpeeds[1] = 0;  // Dừng động cơ phải
      return motorSpeeds;
    }
    else
    {
      motorSpeeds[0] = speed;  // Giữ tốc độ cơ bản
      motorSpeeds[1] = speed;  // Giữ tốc độ cơ bản
      return motorSpeeds;
    }
  }
  else
  {
    _lineIsGlitch = 0;  // Đặt lại bộ đếm lỗi
  }

  // Tính toán PID
  _integral += lineError;  // Cộng dồn lỗi để tính tích phân
  float derivative = lineError - _lastError;  // Tính vi phân
  _lastError = lineError;  // Lưu lỗi hiện tại
  float correction = _Kp * lineError + _Ki * _integral + _Kd * derivative;  // Tính giá trị điều chỉnh
  int motorLeft = speed - (correction > 0 ? correction * 3.5 : correction * 0.7);  // Tính tốc độ động cơ trái
  int motorRight = speed + (correction < 0 ? correction * 3.5 : correction * 0.7); // Tính tốc độ động cơ phải
  motorLeft = constrain(motorLeft, -maxSpeed, maxSpeed);  // Giới hạn tốc độ động cơ trái
  motorRight = constrain(motorRight, -maxSpeed, maxSpeed);  // Giới hạn tốc độ động cơ phải

  motorSpeeds[0] = motorLeft;   // Gán tốc độ động cơ trái
  motorSpeeds[1] = motorRight;  // Gán tốc độ động cơ phải

  return motorSpeeds;
}

// Tính toán tốc độ động cơ theo phương pháp cố định (Fixed Error)
int   *Logic::computeFE(float   lineError,   int baseSpeed)
{
  int speed = baseSpeed;
  static int motorSpeeds[2];

  // Xử lý lỗi phát hiện đường
  if (lineError >= 99.0)
  {
    // Cảnh báo: Có thể xảy ra lỗi phát hiện đường
    _lineIsGlitch++;
    if (_lineIsGlitch > 5)
    {
      motorSpeeds[0] = 0;  // Dừng động cơ trái
      motorSpeeds[1] = 0;  // Dừng động cơ phải
      return motorSpeeds;
    }
    else
    {
      motorSpeeds[0] = speed;  // Giữ tốc độ cơ bản
      motorSpeeds[1] = speed;  // Giữ tốc độ cơ bản
      return motorSpeeds;
    }
  }
  else
  {
    _lineIsGlitch = 0;  // Đặt lại bộ đếm lỗi
  }

  // Giới hạn lỗi tối đa
  if (abs(lineError) > maxError)
  {
    lineError = _lastError;  // Sử dụng lỗi trước đó nếu lỗi vượt quá giới hạn
  }

  _lastError = lineError;  // Lưu lỗi hiện tại
  int motorLeft = speed;   // Khởi tạo tốc độ động cơ trái
  int motorRight = speed;  // Khởi tạo tốc độ động cơ phải

  // Điều chỉnh tốc độ dựa trên giá trị lỗi
  if (lineError == 0.5)
  {
    motorLeft = speed / 1.5;  // Giảm tốc độ trái
    motorRight = speed;       // Giữ tốc độ phải
  }
  else if (lineError == -0.5)
  {
    motorLeft = speed;        // Giữ tốc độ trái
    motorRight = speed / 1.5; // Giảm tốc độ phải
  }
  else if (lineError == 1)
  {
    motorLeft = 0;            // Dừng động cơ trái
    motorRight = speed;       // Giữ tốc độ phải
  }
  else if (lineError == -1)
  {
    motorLeft = speed;        // Giữ tốc độ trái
    motorRight = 0;           // Dừng động cơ phải
  }
  else if (lineError == 1.5)
  {
```

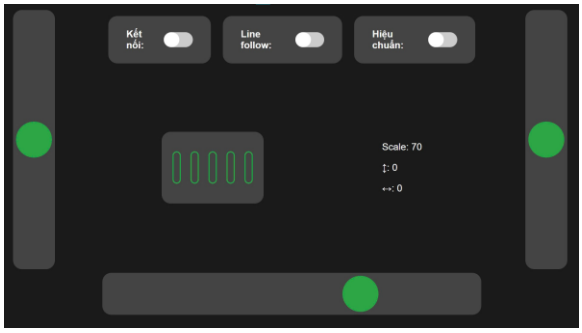
```
    motorLeft = -speed / 2;  // Quay ngược động
cơ trái
    motorRight = speed;      // Giữ tốc độ phải
  }
  else if (lineError == -1.5)
  {
    motorLeft = speed;       // Giữ tốc độ trái
    motorRight = -speed / 2; // Quay ngược động
cơ phải
  }
  else if (lineError == 2)
  {
    motorLeft = -speed * 1.3; // Quay ngược động
cơ trái mạnh hơn
    motorRight = speed;       // Giữ tốc độ phải
  }
  else if (lineError == -2)
  {
    motorLeft = speed;        // Giữ tốc độ trái
    motorRight = -speed * 1.3; // Quay ngược
động cơ phải mạnh hơn
  }

  motorSpeeds[0] = motorLeft;    // Gán tốc độ
động cơ trái
  motorSpeeds[1] = motorRight;  // Gán tốc độ
động cơ phải

  return motorSpeeds;
}

// Bắt đầu logic điều khiển, đặt lại các biến
void Logic::startLogic()
{
  _integral = 0;      // Đặt lại tích phân
  _lastError = 0;     // Đặt lại lỗi trước đó
}

// Đo khoảng cách bên trái
float    Logic::getLeftDistance(L298    *motor,
UltraSonicDistanceSensor *sonic)
{
  motor->turnLeft45();  // Xoay robot 45 độ sang
trái
  delay(200);          // Chờ động cơ hoàn thành
   float d = sonic->measureDistanceCm(); // Đo
khoảng cách
  motor->turnRight45(); // Xoay trở lại vị trí
ban đầu
  delay(200);          // Chờ động cơ hoàn thành
  return d;             // Trả về khoảng cách
}

// Đo khoảng cách bên phải
float    Logic::getRightDistance(L298    *motor,
UltraSonicDistanceSensor *sonic)
{
  motor->turnRight45(); // Xoay robot 45 độ sang
phải
  delay(200);          // Chờ động cơ hoàn thành
   float d = sonic->measureDistanceCm(); // Đo
khoảng cách
  motor->turnLeft45();  // Xoay trở lại vị trí
ban đầu
  delay(200);          // Chờ động cơ hoàn thành
  return d;             // Trả về khoảng cách
}

// Tách khỏi đường khi gặp chướng ngại vật
void   Logic::splitLine(bool   direction,   L298
*motor, Oled *oled)
{
  if (direction)
  {
    oled->debug(  // Hiển thị thông báo trên màn
hình OLED
        String("Obstacle\n") +
            String("                    \n") +
            String("                    \n") +
            String("                    \n") +
            String("        ##          \n") +
            String("        ##          \n") +
            String("                    \n") +
            String("        #######>    \n"),
        1);
    motor->turnRight90(); // Xoay phải 90 độ
  }
  else
  {
    oled->debug(  // Hiển thị thông báo trên màn
hình OLED
        String("Obstacle\n") +
            String("                    \n") +
            String("                    \n") +
            String("                    \n") +
            String("        ##          \n") +
            String("        ##          \n") +
            String("                    \n") +
            String("  <#######          \n"),
        1);
    motor->turnLeft90();  // Xoay trái 90 độ
  }
  delay(1000);              // Chờ 1 giây
   motor->forwardAvoid(true);  // Tiến lên để
tránh chướng ngại
  delay(1000);              // Chờ 1 giây
}

// Di chuyển song song với chướng ngại vật
void  Logic::parallelLine(bool  direction,  L298
*motor, Oled *oled)
{
  oled->debug("Obstacle\n\ndi song song", 1); //
Hiển thị thông báo
  if (direction)
  {
    oled->debug(  // Hiển thị hướng di chuyển
song song
        String("Obstacle\n") +
            String("                    \n") +
            String("                ^   \n") +
            String("                #   \n") +
            String("        ##      #   \n") +
            String("        ##      #   \n") +
            String("                #   \n") +
            String("        ########    \n"),
        1);
    motor->turnLeft90();  // Xoay trái 90 độ
  }
  else
  {
    oled->debug(  // Hiển thị hướng di chuyển
song song
        String("Obstacle\n") +
            String("                    \n") +
            String("  ^                 \n") +
            String("  #                 \n") +
            String("  #     ##          \n") +
            String("  #     ##          \n") +
            String("  #                 \n") +
            String("  ########          \n"),
        1);
    motor->turnRight90(); // Xoay phải 90 độ
  }
  delay(1000);          // Chờ 1 giây
  motor->forwardAvoid(); // Tiến lên để tránh
chướng ngại
  delay(1000);          // Chờ 1 giây
```

```
}

// Tìm và nhập lại đường
bool   Logic::mergeLine(bool   direction,   L298
*motor, Oled *oled, Line *line)
{
   oled->debug("Obstacle\n\ntim  line",  1);  //
Hiển thị thông báo
  if (direction)
  {
    oled->debug(  // Hiển thị hướng tìm đường
        String("Obstacle\n") +
            String("                    \n") +
            String("        <#######    \n") +
            String("               #    \n") +
            String("        ##     #    \n") +
            String("        ##     #    \n") +
            String("               #    \n") +
            String("        ########    \n"),
        1);
    motor->turnLeft90();  // Xoay trái 90 độ
  }
  else
  {
    oled->debug(  // Hiển thị hướng tìm đường
        String("Obstacle\n") +
            String("                    \n") +
            String("  #######>          \n") +
            String("  #                 \n") +
            String("  #     ##          \n") +
            String("  #     ##          \n") +
            String("  #                 \n") +
            String("  ########          \n"),
        1);
    motor->turnRight90(); // Xoay phải 90 độ
  }
  int attempts = 0;
  motor->stop();        // Dừng robot
  delay(1000);          // Chờ 1 giây
  const int maxAttempts = 800; // Số lần thử tối
đa
  while (attempts < maxAttempts)
  {
     motor->forward(motor->getBaseSpeed());  //
Tiến lên với tốc độ cơ bản
    float error = line->getLineError();   // Lấy
lỗi đường
    LineState linestate = line->getLineState();
// Lấy trạng thái đường

    if (linestate != LINE_LOST) // Nếu tìm thấy
đường
    {
      motor->stop();       // Dừng robot
      delay(1000);         // Chờ 1 giây
      break;
    }
    attempts++;            // Tăng số lần thử
  }

   oled->debug("Obstacle\n\nnhap  line", 1);  //
Hiển thị thông báo nhập lại đường
  if (attempts < maxAttempts) // Nếu tìm thấy
đường
  {
    if (direction)
    {
      oled->debug(  // Hiển thị hướng nhập lại
đường
          String("Obstacle\n") +
              String("        ^           \n") +
              String("        ########    \n") +
              String("               #    \n") +
              String("        ##     #    \n") +
              String("        ##     #    \n") +
              String("               #    \n") +
              String("        ########    \n"),
          1);
      motor->turnRight90(true); // Xoay phải 90
độ
    }
    else
    {
      oled->debug(  // Hiển thị hướng nhập lại
đường
          String("Obstacle\n") +
              String("        ^           \n") +
              String("  ########          \n") +
              String("  #                 \n") +
              String("  #     ##          \n") +
              String("  #     ##          \n") +
              String("  #                 \n") +
              String("  ########          \n"),
          1);
      motor->turnLeft90(true);  // Xoay trái 90
độ
    }
  }
  else
  {
    motor->stop(); // Dừng robot
    oled->debug(    // Hiển thị thông báo thất
bại
        String("Obstacle\n") +
        String("                    \n") +
        String("      #    #        \n") +
        String("       #  #         \n") +
        String("        ##          \n") +
        String("        ##          \n") +
        String("       #  #         \n") +
        String("      #    #        \n"));
    return false;  // Trả về thất bại
  }
  motor->stop();   // Dừng robot
  delay(100);      // Chờ 0.1 giây
  return true;     // Trả về thành công
}

// Tránh chướng ngại vật
bool      Logic::avoidObstacle(L298      *motor,
UltraSonicDistanceSensor  *sonic,  Oled  *oled,
Line *line)
{
  // Hiển thị thông báo chướng ngại vật
  oled->debug("Obstacle", 1);
  motor->stop();   // Dừng robot
  delay(800);      // Chờ 0.8 giây

  // Đo khoảng cách bên trái
   float  leftDist  =  getLeftDistance(motor,
sonic);
  oled->debug(     // Hiển thị khoảng cách bên
trái
      String("Obstacle\n") +
          String("                    \n") +
          String("                    \n") +
          String("                    \n") +
          String("        ##          \n") +
          String(leftDist) + String("\n") +
          String("                    \n") +
          String("                    \n"),
      1);

  // Đo khoảng cách bên phải
   float  rightDist  =  getRightDistance(motor,
sonic);
  oled->debug(      // Hiển thị cả hai khoảng
cách
```

```
      String("Obstacle\n") +
      String("                    \n") +
      String("                    \n") +
      String("                    \n") +
      String("        ##          \n") +
       String(leftDist) + String("            ")
+ String(rightDist) + String("\n") +
      String("                    \n") +
      String("                    \n"));

  // Kiểm tra khoảng cách an toàn
  const float minSafeDist = 20.0;
  bool canGoLeft = leftDist > minSafeDist;   //
Kiểm tra bên trái
  bool canGoRight = rightDist > minSafeDist; //
Kiểm tra bên phải

  // Nếu không có hướng nào an toàn
  if (!canGoLeft && !canGoRight)
  {
    motor->stop(); // Dừng robot
     oled->debug(    // Hiển thị thông báo thất
bại
        String("Obstacle\n") +
        String("                    \n") +
        String("      #    #        \n") +
        String("       #  #         \n") +
        String("        ##          \n") +
        String("        ##          \n") +
        String("       #  #         \n") +
        String("      #    #        \n"));
    return false;  // Trả về thất bại
  }

  // Chọn hướng di chuyển
  bool turnRight;
  if (canGoRight && !canGoLeft)
    turnRight = true;    // Chỉ có thể đi bên
phải
  else if (!canGoRight && canGoLeft)
    turnRight = false;  // Chỉ có thể đi bên
trái
  else
    turnRight = rightDist > leftDist; // Chọn
hướng có khoảng cách lớn hơn

  // Thực hiện các bước tránh chướng ngại
  this->splitLine(turnRight, motor, oled);    //
Tách khỏi đường
  this->parallelLine(turnRight, motor, oled); //
Di chuyển song song
  return this->mergeLine(turnRight, motor, oled,
line); // Nhập lại đường
}

// Hàm hủy lớp Logic
Logic::~Logic()
{
}
```

IV. Giao diện web và kết nối

- Giao diện web responsive, tối ưu cho điều khiển trên điện thoại
- Các thành phần chính:
  + Thanh trượt điều khiển tốc độ
  + Thanh trượt điều khiển hướng
  + Hiển thị trạng thái cảm biến dò line
  + Nút chuyển đổi chế độ hoạt động



Hình 1: Giao diện điều khiển

- ESP32 hoạt động như Access Point WiFi
- Giao thức WebSocket cho truyền dữ liệu thời gian thực
- API điều khiển thông qua HTTP server

V. CÁC TÍNH NĂNG CHÍNH

1. Dò line và điều khiển động cơ:

- Sử dụng cảm biến dò line để phát hiện đường đi
- Điều khiển động cơ thông qua driver L298N
- Thuật toán PID để điều chỉnh tốc độ và hướng đi

2. Né vật cản

- Sử dụng cảm biến siêu âm để phát hiện vật cản phía trước xe
- Tự động chuyển hướng hoặc dừng lại khi gặp vật cản để tránh va chạm

3. Giao diện web:

- Hiển thị trạng thái hoạt động của xe
- Điều khiển xe từ xa
- Cấu hình các thông số hệ thống

4. Giám sát hệ thống:

- Hiển thị thông tin trên màn hình OLED
- Theo dõi trạng thái pin
- Cảnh báo lỗi và trạng thái bất thường

5. Các chế độ hoạt động:

a) Chế độ dò line (LINE_FOLLOW):

- Tự động dò và đi theo line
- Tránh vật cản khi phát hiện (khoảng cách < 17cm)
- Hiển thị thông tin về sai số và tốc độ động cơ

b) Chế độ điều khiển (CONTROL):

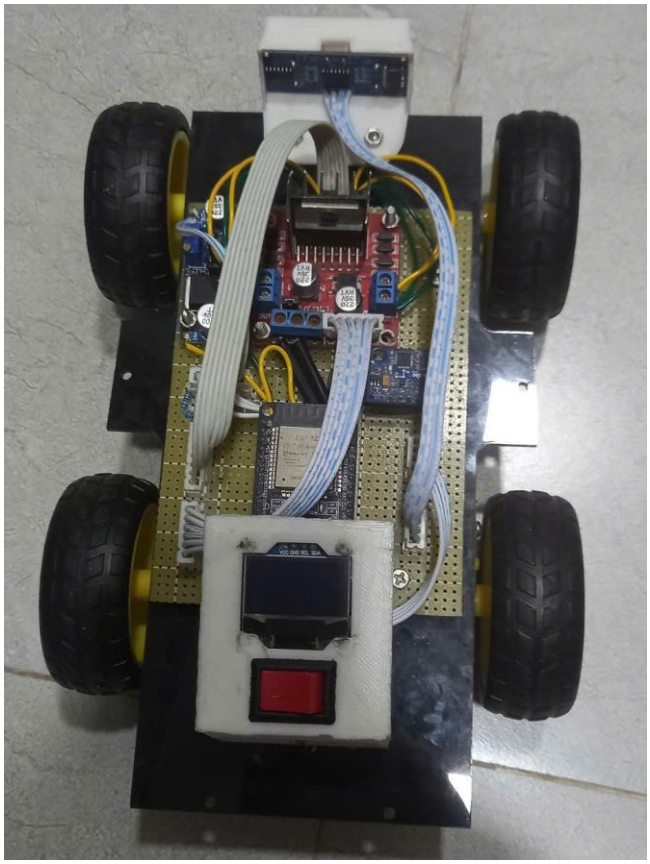
- Điều khiển xe thủ công thông qua giao diện web
- Điều chỉnh tốc độ từng động cơ

c) Chế độ hiệu chuẩn (CALIBRATE):

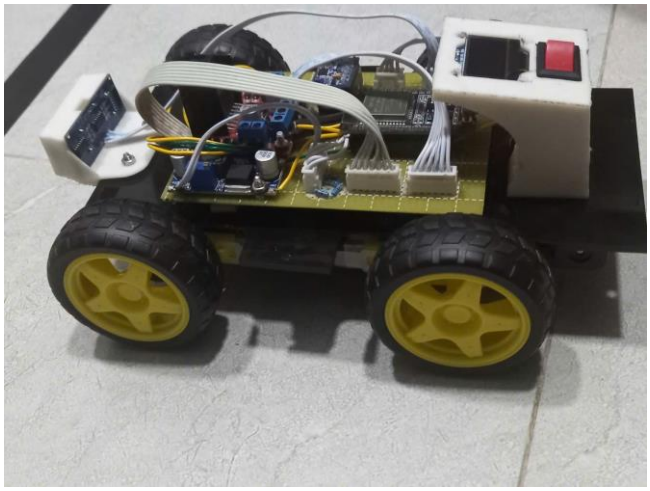
- Hiệu chuẩn cảm biến dò line
- Lưu các thông số vào bộ nhớ
- Cập nhật ngưỡng phát hiện line

V. Kết quả

Hình 1: Mô hình xe tự hành 1

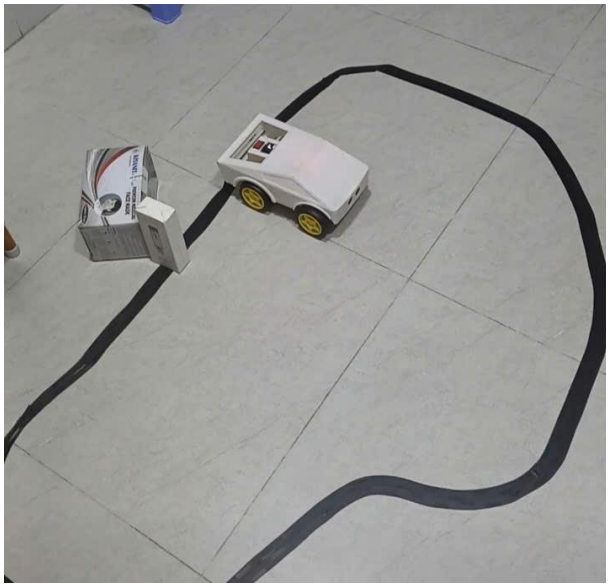


Hình 2: Mô hình xe tự hành 2

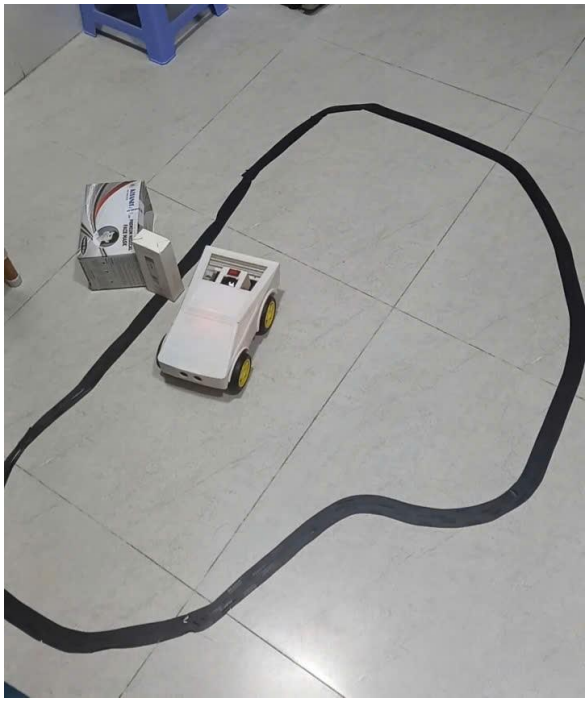


Hình 3: Mô hình xe tự hành 3

Hình 4: Xe có thể tự dò line



Hình 5: Xe nhận diện được vật cản và tránh né



Hình 6: Xe nhận diện được vật cản và tránh né né

VI. Kết luận

Qua quá trình nghiên cứu và thực hiện đề tài, nhóm đã xây dựng thành công một mô hình xe tự hành có khả năng dò line và né vật cản một cách ổn định, đáp ứng mục tiêu đề ra. Hệ thống còn được tích hợp giao diện web điều khiển từ xa hoạt động mượt mà, cùng với khả năng giám sát tình trạng xe một cách chính xác. Những kết quả này cho thấy tính khả thi và hiệu quả của mô hình khi áp dụng các kỹ thuật điều khiển cơ bản mà không cần sử dụng trí tuệ nhân tạo.

Tuy nhiên, bên cạnh những kết quả đạt được, hệ thống vẫn tồn tại một số hạn chế. Cụ thể, cảm biến dò line có thể bị ảnh hưởng bởi ánh sáng môi trường, gây sai số trong quá trình nhận dạng vạch kẻ. Ngoài ra, cảm biến siêu âm né vật cản đôi khi bị nhiễu bởi các bề mặt vật mềm, vật nhỏ hoặc tín hiệu siêu âm khác, dẫn đến việc xe kích hoạt chế độ né vật cản không cần thiết trong một số tình huống.

Những hạn chế trên sẽ là cơ sở để nhóm tiếp tục cải tiến hệ thống trong các giai đoạn sau, nhằm nâng cao độ ổn định và mở rộng khả năng hoạt động của xe trong môi trường thực tế phức tạp hơn.

VI. Tài liệu tham khảo

[1] http://arduino.vn/tutorial/1570-gioi-thieu-module-esp32-va-huong-dan-cai-trinh-bien-dich-tren-arduino-ide

[2] http://arduino.vn/bai-viet/893-cach-dung-module-dieu-khien-dong-co-l298n-cau-h-de-dieu-khien-dong-co-dc

[3] http://arduino.vn/bai-viet/233-su-dung-cam-bien-khoang-cach-hc-sr04#:~:text=C%E1%BA%A3m%20bi%E1%BA%BFn%20HC%2DSR04%20c%C3%B3,%2C%20Trig%2C%20Echo%2C%20GND