變數不能大寫、有單引號(')、換行(\n)要設為脫逸字元、~~字串變數不用留雙引號~~

|  |  |
| --- | --- |
| \_01imi\_megacar\_init  iMi小車模組Mega初始化  UART通訊  循跡感測器  循跡感測器  貨斗伺服馬達  AI鏡頭伺服馬達  簽收鈕  L298N馬達腳位  L298N PWM腳位  \_02imi\_megacar\_init\_uart  RX TX  0  0  '//UART通訊(接收ESP32傳來的MQTT訊息)'  'SoftwareSerial ESP32Serial('+value\_rx+',' +value\_tx+');'  \_03imi\_megacar\_init\_track  左 前 後 右  0  0  0  0  '//4way循跡感測器腳位'  '#define TRACK\_LEFT\_PIN ' +value\_left +'//左邊 id1'  '#define TRACK\_FRONT\_PIN ' +value\_front +'//前面 id2'  '#define TRACK\_BACK\_PIN ' +value\_back +'//後面 id3'  '#define TRACK\_RIGHT\_PIN ' +value\_right +'//右邊 id4'  \_04imi\_megacar\_init\_trig  觸發值  1  '//觸發到白線回傳0或1'  '#define TRIGGLED ' +value\_trig  \_05imi\_megacar\_init\_carbox  腳位 打開角度 關閉角度  0  0  0  '//貨斗伺服馬達腳位,角度'  '#define SERVO\_CAR\_BOX\_PIN ' +value\_carbox  '#define ANGLE\_CAR\_BOX\_OPEN ' +value\_open + '//開啟貨斗'  '#define ANGLE\_CAR\_BOX\_CLOSE ' +value\_close + '//關閉貨斗'  \_06imi\_megacar\_init\_aicam  腳位 向前角度 向上角度  0  0  0  '//AI鏡頭伺服馬達腳位,角度'  '#define SERVO\_AI\_CAM\_PIN ' +value\_aicam  '#define ANGLE\_AI\_CAM\_FRONT ' +value\_front +'//鏡頭向前'  '#define ANGLE\_AI\_CAM\_UP ' +value\_up +'//鏡頭朝上'  \_07imi\_megacar\_init\_sign  腳位  0  '//簽收鈕'  '#define BUTTON\_SIGN\_PIN ' +value\_sign  \_08imi\_megacar\_init\_l298n  IN1~IN8  0  0  0  0  0  0  0  0  '//L298N腳位'  '#define L298N\_IN1 ' +value\_in1  '#define L298N\_IN2 ' +value\_in2  '#define L298N\_IN3 ' +value\_in3  '#define L298N\_IN4 ' +value\_in4  '#define L298N\_IN5 ' +value\_in5  '#define L298N\_IN6 ' +value\_in6  '#define L298N\_IN7 ' +value\_in7  '#define L298N\_IN8 ' +value\_in8  \_09imi\_megacar\_init\_l298n2  EN1~EN4  0  0  0  0  '#define L298N\_EN1 ' +value\_en1  '#define L298N\_EN2 ' +value\_en2  '#define L298N\_EN3 ' +value\_en3  '#define L298N\_EN4 ' +value\_en4 | #include <Arduino.h>  #include <SoftwareSerial.h>  #include <Servo.h>  #include "HUSKYLENS.h"  #include <Wire.h>  #include <Adafruit\_INA219.h>  //====插入變數Start====  Statements區塊  //====插入變數End====  //★★★車頭初始向下  char CAR\_INIT\_DIRECT = \'D\';  // 定義地圖大小  const int numRows = 4;  const int numCols = 6;  //地圖陣列，0表示障礙物，1表示可通行  int grid[numRows][numCols];  //位置  const char\* GOODS\_POINT = "goodsStation";  const char\* CHARGE\_POINT = "chargeStation";  const char\* Recipient\_POINT = "recipientHome";  //目前位置  String currentPoint = GOODS\_POINT;  //起點位置  String startPoint = GOODS\_POINT;  int startRow = 0;  int startCol = 0;  //終點位置  String endPoint = Recipient\_POINT;  int endRow = 3;  int endCol = 5;  //儲存座標結果  String pathXY[numRows \* numCols];  //int pathXYCount = 0;  //儲存地圖方向結果URDL  char pathMapDirect[numRows \* numCols];  char MAP\_DIRECT[4] = { \'U\', \'R\', \'D\', \'L\' };  int pathCount = 0;  //儲存車子(順時針)轉動角度  int pathCarDegree[numRows \* numCols];  //儲存車子移動方向  char\* pathCarMove[numRows \* numCols];  char\* CAR\_MOVE[4] = { "F,", "R,F,", "R,R,F,", "L,F," };  //與Esp32通訊  const char\* MAP\_SET = "mapSet";  const char\* GOODS\_LOAD = "goodsLoad";  const char\* LINE\_NOTIFY = "lineNotify";  const char\* CAR\_GPS = "carGps";  //收貨人  String recipient[6];  //4way循跡感測器陣列  int trackSensor[4];  #define TRACK\_LEFT 0  #define TRACK\_FRONT 1  #define TRACK\_BACK 2  #define TRACK\_RIGHT 3  //是否走完一格  bool isFrontArrive = false;  Servo servoCarBox;  Servo servoAiCam;  //HsukyLens AI鏡頭  HUSKYLENS huskylens;  int readData[5] = {};  byte dataType = 0;  byte idCount = 0;  bool detection\_now = 0;  int aiId = 0;  int aiX = 0; //中心點X座標  int aiY = 0; //中心點Y座標  int aiWidth = 0;  int aiHeight = 0;  const int PERSON\_ID = 1; //物體識別模式:人(透過AI鏡頭訓練)  const int CAR\_ID = 2; //物體識別模式:車子(透過AI鏡頭訓練)  const int FACE\_ID1 = 1; //人臉識別模式:車子(透過AI鏡頭訓練)  char\* CurrentAlgo = "ALGORITHM\_OBJECT\_RECOGNITION"; //目前的演算法  //測電壓電流  Adafruit\_INA219 ina219;  float busvoltage = 0; //電池電壓  float shuntvoltage = 0;  float loadvoltage = 0; //負載電壓  float current\_mA = 0; //負載電流  float power\_mW = 0; //負載功率  //轉速(120~255)  const int FSpeed = 120;  const int BSpeed = 120;  const int RSpeed = 120;  const int LSpeed = 120;  //轉動時間(毫秒)  const int FTimer = 2000;  const int BTimer = 1000;  const int RTimer = 1000;  const int LTimer = 1000;  const int STimer = 3000;  void setup() {  Serial.begin(9600);  //UART  ESP32Serial.begin(9600);  //按鈕I/O  pinMode(BUTTON\_SIGN\_PIN, INPUT);  //伺服馬達初始  servoCarBox.attach(SERVO\_CAR\_BOX\_PIN);  servoCarBox.write(ANGLE\_CAR\_BOX\_CLOSE);  delay(1000);  servoAiCam.attach(SERVO\_AI\_CAM\_PIN);  servoAiCam.write(ANGLE\_AI\_CAM\_FRONT);  delay(1000);  //HuskyLens鏡頭初始化  Wire.begin();  while (!huskylens.begin(Wire)) {  Serial.println(F("Begin failed!"));  Serial.println(F("1.Please recheck the Protocol Type in HUSKYLENS (General Settings>>Protocol Type>>I2C)"));  Serial.println(F("2.Please recheck the connection."));  delay(100);  }  huskylens.writeAlgorithm(ALGORITHM\_OBJECT\_RECOGNITION); //物體辨識模式  CurrentAlgo = "ALGORITHM\_OBJECT\_RECOGNITION";  //測電壓電流初始  if (!ina219.begin()) {  Serial.println("Failed to find INA219 chip");  while (1) { delay(10); }  }  //小車初始化  pinMode(L298N\_IN1, OUTPUT);  pinMode(L298N\_IN2, OUTPUT);  pinMode(L298N\_IN3, OUTPUT);  pinMode(L298N\_IN4, OUTPUT);  pinMode(L298N\_IN5, OUTPUT);  pinMode(L298N\_IN6, OUTPUT);  pinMode(L298N\_IN7, OUTPUT);  pinMode(L298N\_IN8, OUTPUT);  pinMode(L298N\_EN1, OUTPUT);  pinMode(L298N\_EN2, OUTPUT);  pinMode(L298N\_EN3, OUTPUT);  pinMode(L298N\_EN4, OUTPUT);  digitalWrite(L298N\_IN1, LOW);  digitalWrite(L298N\_IN2, LOW);  digitalWrite(L298N\_IN3, LOW);  digitalWrite(L298N\_IN4, LOW);  digitalWrite(L298N\_IN5, LOW);  digitalWrite(L298N\_IN6, LOW);  digitalWrite(L298N\_IN7, LOW);  digitalWrite(L298N\_IN8, LOW);  digitalWrite(L298N\_EN1, LOW);  digitalWrite(L298N\_EN2, LOW);  digitalWrite(L298N\_EN3, LOW);  digitalWrite(L298N\_EN4, LOW);  //循跡感測器初始化  pinMode(TRACK\_LEFT\_PIN, INPUT);  pinMode(TRACK\_FRONT\_PIN, INPUT);  pinMode(TRACK\_BACK\_PIN, INPUT);  pinMode(TRACK\_RIGHT\_PIN, INPUT);  stopCar();  delay(2000);  }//end setup  void diagonalRight() {  digitalWrite(L298N\_IN1, HIGH);  digitalWrite(L298N\_IN2, LOW);  analogWrite(L298N\_EN1, FSpeed);  digitalWrite(L298N\_IN3, LOW);  digitalWrite(L298N\_IN4, LOW);  analogWrite(L298N\_EN2, FSpeed);  digitalWrite(L298N\_IN5, LOW);  digitalWrite(L298N\_IN6, LOW);  analogWrite(L298N\_EN3, FSpeed);  digitalWrite(L298N\_IN7, HIGH);  digitalWrite(L298N\_IN8, LOW);  analogWrite(L298N\_EN4, FSpeed);  }  void diagonalLeft() {  digitalWrite(L298N\_IN1, LOW);  digitalWrite(L298N\_IN2, LOW);  analogWrite(L298N\_EN1, FSpeed);  digitalWrite(L298N\_IN3, HIGH);  digitalWrite(L298N\_IN4, LOW);  analogWrite(L298N\_EN2, FSpeed);  digitalWrite(L298N\_IN5, HIGH);  digitalWrite(L298N\_IN6, LOW);  analogWrite(L298N\_EN3, FSpeed);  digitalWrite(L298N\_IN7, LOW);  digitalWrite(L298N\_IN8, LOW);  analogWrite(L298N\_EN4, FSpeed);  }  void forward() {  digitalWrite(L298N\_IN1, HIGH);  digitalWrite(L298N\_IN2, LOW);  analogWrite(L298N\_EN1, FSpeed);  digitalWrite(L298N\_IN3, HIGH);  digitalWrite(L298N\_IN4, LOW);  analogWrite(L298N\_EN2, FSpeed);  digitalWrite(L298N\_IN5, HIGH);  digitalWrite(L298N\_IN6, LOW);  analogWrite(L298N\_EN3, FSpeed);  digitalWrite(L298N\_IN7, HIGH);  digitalWrite(L298N\_IN8, LOW);  analogWrite(L298N\_EN4, FSpeed);  }  void backward() {  digitalWrite(L298N\_IN1, LOW);  digitalWrite(L298N\_IN2, HIGH);  analogWrite(L298N\_EN1, BSpeed);  digitalWrite(L298N\_IN3, LOW);  digitalWrite(L298N\_IN4, HIGH);  analogWrite(L298N\_EN2, BSpeed);  digitalWrite(L298N\_IN5, LOW);  digitalWrite(L298N\_IN6, HIGH);  analogWrite(L298N\_EN3, BSpeed);  digitalWrite(L298N\_IN7, LOW);  digitalWrite(L298N\_IN8, HIGH);  analogWrite(L298N\_EN4, BSpeed);  }  void turnRight() {  digitalWrite(L298N\_IN1, HIGH);  digitalWrite(L298N\_IN2, LOW);  analogWrite(L298N\_EN1, RSpeed);  digitalWrite(L298N\_IN3, HIGH);  digitalWrite(L298N\_IN4, LOW);  analogWrite(L298N\_EN2, RSpeed);  digitalWrite(L298N\_IN5, LOW);  digitalWrite(L298N\_IN6, HIGH);  analogWrite(L298N\_EN3, RSpeed);  digitalWrite(L298N\_IN7, LOW);  digitalWrite(L298N\_IN8, HIGH);  analogWrite(L298N\_EN4, RSpeed);  }  void turnLeft() {  digitalWrite(L298N\_IN1, LOW);  digitalWrite(L298N\_IN2, HIGH);  analogWrite(L298N\_EN1, LSpeed);  digitalWrite(L298N\_IN3, LOW);  digitalWrite(L298N\_IN4, HIGH);  analogWrite(L298N\_EN2, LSpeed);  digitalWrite(L298N\_IN5, HIGH);  digitalWrite(L298N\_IN6, LOW);  analogWrite(L298N\_EN3, LSpeed);  digitalWrite(L298N\_IN7, HIGH);  digitalWrite(L298N\_IN8, LOW);  analogWrite(L298N\_EN4, LSpeed);  }  void stopCar() {  digitalWrite(L298N\_IN1, LOW);  digitalWrite(L298N\_IN2, LOW);  analogWrite(L298N\_EN1, FSpeed);  digitalWrite(L298N\_IN3, LOW);  digitalWrite(L298N\_IN4, LOW);  analogWrite(L298N\_EN2, FSpeed);  digitalWrite(L298N\_IN5, LOW);  digitalWrite(L298N\_IN6, LOW);  analogWrite(L298N\_EN3, FSpeed);  digitalWrite(L298N\_IN7, LOW);  digitalWrite(L298N\_IN8, LOW);  analogWrite(L298N\_EN4, FSpeed);  }  void carHeadTurn(char targetDirect) {  //車頭轉向  int degree = mapDirectToCarDegree(CAR\_INIT\_DIRECT, targetDirect);  char\* CAR\_MOVE\_NO\_FRONT[4] = { "", "R", "RR", "L" };  int index = degree / 90;  if (CAR\_MOVE\_NO\_FRONT[index] == "R") {  turnRight();  delay(RTimer);  stopCar();  delay(RTimer);  Serial.println("右轉,");  } else if (CAR\_MOVE\_NO\_FRONT[index] == "RR") {  for (int i = 0; i < 2; i++) {  turnRight();  delay(RTimer);  stopCar();  delay(RTimer);  Serial.println("右轉,");  }  } else if (CAR\_MOVE\_NO\_FRONT[index] == "L") {  turnLeft();  delay(LTimer);  stopCar();  delay(LTimer);  Serial.println("左轉,");  }  //紀錄車頭方向  CAR\_INIT\_DIRECT = targetDirect;  }  //取得循跡感測器值  void getTracks() {  //格式：左前後右(ex:0101)  trackSensor[TRACK\_LEFT] = digitalRead(TRACK\_LEFT\_PIN);  trackSensor[TRACK\_FRONT] = digitalRead(TRACK\_FRONT\_PIN);  trackSensor[TRACK\_BACK] = digitalRead(TRACK\_BACK\_PIN);  trackSensor[TRACK\_RIGHT] = digitalRead(TRACK\_RIGHT\_PIN);  }  void AiDetect() {  bool isDetectPerson = false;  bool isDetectCar = false;  //直到沒有偵側到人or車才離開迴圈  do {  //開始AI識別物體  startDetectObject();  //在範圍內偵測到人,車才算  if ((aiX >= 100 && aiX <= 220) && (aiY >= 60 && aiY <= 180)) {  if (aiId == PERSON\_ID) {  isDetectPerson = true;  Serial.println("人");  } else if (aiId == CAR\_ID) {  isDetectCar = true;  Serial.println("車");  }  }  delay(500);  } while (isDetectPerson || isDetectCar);  }  void startDetectObject() {  if (!huskylens.request()) {  Serial.println(F("Fail to request data from HUSKYLENS, recheck the connection!"));  } else {  if (huskylens.available()) {  detection\_now = true;  HUSKYLENSResult result = huskylens.read();  idCount = huskylens.countLearned();  if (result.command == COMMAND\_RETURN\_BLOCK) {  dataType = 0;  readData[0] = result.xCenter;  readData[1] = result.yCenter;  readData[2] = result.width;  readData[3] = result.height;  readData[4] = result.ID;  } else if (result.command == COMMAND\_RETURN\_ARROW) {  dataType = 1;  readData[0] = result.xOrigin;  readData[1] = result.yOrigin;  readData[2] = result.xTarget;  readData[3] = result.yTarget;  readData[4] = result.ID;  } else {  for (byte i = 0; i < 5; i++) {  readData[i] = 0;  }  }  } else {  detection\_now = false;  }  }  if (detection\_now) {  aiId = readData[4];  aiX = readData[0];  aiY = readData[1];  aiWidth = readData[2];  aiHeight = readData[3];  //id=0表示有辨識到物體，但該物體沒有被學習  if (aiId > 0) {  Serial.print(aiId);  }  } else {  aiId = 0;  aiX = 0;  aiY = 0;  aiWidth = 0;  aiHeight = 0;  }  }  int mapDirectToIndex(char direct) {  for (int i = 0; i < 4; i++) {  if (MAP\_DIRECT[i] == direct) {  return i;  };  }  return 0;  }  char\* carDegreeToMove(int degree) {  int index = degree / 90;  return CAR\_MOVE[index];  }  int mapDirectToCarDegree(char firstDirect, char NextDirect) {  int firstIndex = mapDirectToIndex(firstDirect);  int NextIndex = mapDirectToIndex(NextDirect);  if (NextIndex < firstIndex) {  NextIndex += 4;  };  int degree = (NextIndex - firstIndex) \* 90;  return degree;  }  void convertXyToCarMove() {  //反轉陣列(原本A\*輸出結果相反)  reverseStringArray(pathXY, pathCount + 1);  reverseCharArray(pathMapDirect, pathCount + 1);  //將地圖方向URDL轉成順時針角度90°,180°,270°  for (int i = 0; i < pathCount + 1; i++) {  pathCarDegree[i] = mapDirectToCarDegree(pathMapDirect[i], pathMapDirect[i + 1]);  }  //將角度轉換成車子移動F,RF,RRF,LF  for (int i = 0; i < pathCount; i++) {  pathCarMove[i] = carDegreeToMove(pathCarDegree[i]);  }  }  void printAStarResult() {  //印出地圖座標  for (int i = 0; i < pathCount + 1; i++) {  Serial.print(pathXY[i]);  Serial.print("→");  }  Serial.println("");  //印出地圖上下左右方向  for (int i = 0; i < pathCount + 1; i++) {  Serial.print(pathMapDirect[i]);  Serial.print(\',\');  }  Serial.println("");  //印出角度  for (int i = 0; i < pathCount; i++) {  Serial.print(pathCarDegree[i]);  Serial.print(\',\');  }  Serial.println("");  //印出車子移動指令  for (int i = 0; i < pathCount; i++) {  Serial.print(pathCarMove[i]);  }  Serial.println("");  }  void reverseCharArray(char arr[], int length) {  int start = 0;  int end = length - 1;  while (start < end) {  // 交換陣列中的元素  char temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  // 移動指標以繼續反轉  start++;  end--;  }  }  void reverseStringArray(String arr[], int length) {  int start = 0;  int end = length - 1;  while (start < end) {  // 交換陣列中的元素  String temp = arr[start];  arr[start] = arr[end];  arr[end] = temp;  // 移動指標以繼續反轉  start++;  end--;  }  }  //設起訖點  void setStartEndPoint(String start, String end) {  startPoint = start;  endPoint = end;  if (startPoint == GOODS\_POINT) {  startRow = 0;  startCol = 0;  } else if (startPoint == CHARGE\_POINT) {  startRow = 3;  startCol = 0;  } else if (startPoint == Recipient\_POINT) {  startRow = 3;  startCol = 5;  }  if (endPoint == GOODS\_POINT) {  endRow = 0;  endCol = 0;  } else if (endPoint == CHARGE\_POINT) {  endRow = 3;  endCol = 0;  } else if (endPoint == Recipient\_POINT) {  endRow = 3;  endCol = 5;  }  }  bool starAStarPlan(char\* target) {  setStartEndPoint(currentPoint, target); //設起訖點  bool isFindPath = aStar(grid, startRow, startCol, endRow, endCol); //A\*演算  return isFindPath;  }  void starNav(bool isFindPath, char\* target) {  if (isFindPath) {  Serial.println("找到路徑!");  //座標起點  pathXY[pathCount] = String(startRow) + "," + String(startCol);  //車頭初始方向  pathMapDirect[pathCount] = CAR\_INIT\_DIRECT;  //座標轉換成車子移動指令  convertXyToCarMove();  //印出結果  printAStarResult();  //開始移動實際車子(含雲端平台GPS模擬)  goCar();  //紀錄  CAR\_INIT\_DIRECT = pathMapDirect[pathCount]; //最後車頭方向,當成下次導航車頭起始方向  currentPoint = target; //目前車子位置  } else {  Serial.println("未找到路徑.");  }  }  void standByAiCam() {  //A鏡頭上  servoAiCam.write(ANGLE\_AI\_CAM\_UP);  delay(1000);  //Hsukylens人臉識別模式  huskylens.writeAlgorithm(ALGORITHM\_FACE\_RECOGNITION);  CurrentAlgo = "ALGORITHM\_FACE\_RECOGNITION";  }  float getBusPowerPercent() {  const float minVolt = 6.3; //以7.4v鋰電池為準  const float maxVolt = 8.2; //以7.4v鋰電池為準  //測電壓電流  shuntvoltage = ina219.getShuntVoltage\_mV();  busvoltage = ina219.getBusVoltage\_V();  current\_mA = ina219.getCurrent\_mA();  power\_mW = ina219.getPower\_mW();  loadvoltage = busvoltage + (shuntvoltage / 1000);  return ((busvoltage - minVolt) / (maxVolt - minVolt));  }  void startCharge() {  //導航至充電站  bool isFindPath = starAStarPlan(CHARGE\_POINT);  starNav(isFindPath, CHARGE\_POINT);  //進充電站前，車子原地轉向，車頭要朝左L  carHeadTurn(\'L\');  //接著倒車進入充電站  backward();  delay(2000); //倒車2秒  stopCar();  //充電中……  float percent = 0.0;  while (percent < 1) { //100%  percent = getBusPowerPercent();  delay(10000);  }  //充電完成，車子往前開出充電站  forward();  delay(2000); //往前2秒  stopCar();  //完成充電，導航回物流站  isFindPath = starAStarPlan(GOODS\_POINT);  starNav(isFindPath, GOODS\_POINT);  } |
| \_10imi\_megacar\_astar  A\*演算法初始化 | // ===========A\*演算法 Start===========  // 表示地圖上的節點的類別  class Node {  public:  int row;  int col;  int f; // f = g + h，表示節點的總代價  int g; // 從起點到當前節點的實際代價  int h; // 從當前節點到目標節點的估算代價  Node\* parent;  Node(int r, int c)  : row(r), col(c), f(0), g(0), h(0), parent(nullptr) {}  // 計算當前節點到目標節點的估算代價（這裡使用曼哈頓距離）  void calculateH(int targetRow, int targetCol) {  h = abs(targetRow - row) + abs(targetCol - col);  }  };  bool aStar(int grid[numRows][numCols], int startRow, int startCol, int endRow, int endCol) {  Node\* openList[numRows \* numCols];  Node\* closedList[numRows \* numCols];  int openListCount = 0;  int closedListCount = 0;  Node\* startNode = new Node(startRow, startCol);  Node\* endNode = new Node(endRow, endCol);  startNode->calculateH(endRow, endCol);  startNode->f = startNode->g + startNode->h;  openList[openListCount++] = startNode;  while (openListCount > 0) {  // 找到f值最小的節點  int minFIndex = 0;  for (int i = 1; i < openListCount; i++) {  if (openList[i]->f < openList[minFIndex]->f) {  minFIndex = i;  }  }  Node\* currentNode = openList[minFIndex];  // 將當前節點移到關閉列表  for (int i = minFIndex; i < openListCount - 1; i++) {  openList[i] = openList[i + 1];  }  openListCount--;  closedList[closedListCount++] = currentNode;  // 如果當前節點是目標節點，找到路徑並返回  if (currentNode->row == endRow && currentNode->col == endCol) {  Node\* current = currentNode;  while (current->parent != nullptr) {  pathXY[pathCount] = String(current->row) + "," + String(current->col);  int deltaX = current->col - current->parent->col;  int deltaY = current->row - current->parent->row;  if (deltaX == 1) {  pathMapDirect[pathCount++] = \'R\';  } else if (deltaX == -1) {  pathMapDirect[pathCount++] = \'L\';  } else if (deltaY == 1) {  pathMapDirect[pathCount++] = \'D\';  } else if (deltaY == -1) {  pathMapDirect[pathCount++] = \'U\';  }  current = current->parent;  }  return true;  }  // 獲取當前節點的相鄰節點  int neighbors[4][2] = { { -1, 0 }, { 1, 0 }, { 0, -1 }, { 0, 1 } };  for (int i = 0; i < 4; i++) {  int newRow = currentNode->row + neighbors[i][0];  int newCol = currentNode->col + neighbors[i][1];  // 檢查新節點是否在地圖範圍內  if (newRow >= 0 && newRow < numRows && newCol >= 0 && newCol < numCols) {  // 檢查新節點是否是障礙物  if (grid[newRow][newCol] == 0) {  continue;  }  Node\* neighbor = new Node(newRow, newCol);  neighbor->calculateH(endRow, endCol);  neighbor->g = currentNode->g + 1; // 假設每個步驟的代價都是1  neighbor->f = neighbor->g + neighbor->h;  neighbor->parent = currentNode;  // 檢查新節點是否在關閉列表中  bool inClosedList = false;  for (int j = 0; j < closedListCount; j++) {  if (closedList[j]->row == neighbor->row && closedList[j]->col == neighbor->col) {  inClosedList = true;  break;  }  }  if (inClosedList) {  delete neighbor;  continue;  }  // 檢查新節點是否已經在開放列表中  bool inOpenList = false;  for (int j = 0; j < openListCount; j++) {  if (openList[j]->row == neighbor->row && openList[j]->col == neighbor->col) {  inOpenList = true;  if (neighbor->g < openList[j]->g) {  openList[j]->g = neighbor->g;  openList[j]->f = neighbor->f;  openList[j]->parent = neighbor->parent;  }  delete neighbor;  break;  }  }  if (!inOpenList) {  openList[openListCount++] = neighbor;  }  }  }  }  // 如果開放列表為空，表示沒有找到路徑  return false;  }  // ===========A\*演算法 End=========== |
|  |  |
| \_11imi\_megacar\_esp32  loop區(但要用code)  當接收ESP32訊息時 | //接收訊息：ESP32→Mega  while (ESP32Serial.available()) {  String str = ESP32Serial.readString();  Serial.println(str);  }//end while |
|  |  |
| \_12imi\_megacar\_isesp32msg  loop區(但要用code)  ESP32訊息包含 | 'str.indexOf('+dropdown\_msg+') != -1'  str.indexOf(GOODS\_LOAD) != -1 |
| \_13imi\_megacar\_servo  loop區(但要用code)  設定為 角度  貨斗伺服馬達  打開  貨斗伺服馬達 servoCarBox  AI鏡頭伺服馬達 servoAiCam  打開 ANGLE\_CAR\_BOX\_OPEN  關閉 ANGLE\_CAR\_BOX\_CLOSE  朝前 ANGLE\_AI\_CAM\_FRONT  向上 ANGLE\_AI\_CAM\_UP | dropdown\_servo+'.write('+dropdown\_angle+');'  delay(1000); |
| \_14imi\_megacar\_savemap  loop區(但要用code)  儲存地圖陣列 | //儲存地圖陣列(格式:MAP\_SET,4x6地圖陣列)  String tmpArray[25];  char\* token = strtok((char\*)str.c\_str(), ",");  int tokenLen = 0;  while (token != NULL && tokenLen < 25) {  tmpArray[tokenLen] = token;  token = strtok(NULL, ",");  tokenLen++;  }  int count = 1; //第0個MAP\_SET不用放入  for (int i = 0; i < 4; i++) {  for (int j = 0; j < 6; j++) {  grid[i][j] = tmpArray[count++].toInt();  }  } |
| \_15imi\_megacar\_savereci  loop區(但要用code)  儲存收件人陣列 | //儲存收件人陣列(格式:goodsLoad,姓名,商品,倉庫X,倉庫Y,收件人X,收件人Y)  String tmpArray[7];  char\* token = strtok((char\*)str.c\_str(), ",");  int tokenLen = 0;  while (token != NULL && tokenLen < 7) {  tmpArray[tokenLen] = token;  token = strtok(NULL, ",");  tokenLen++;  }  //第0個goodsLoad不用放入  int count = 0;  for (int i = 1; i < 7; i++) {  recipient[count++] = tmpArray[i];  } |
| \_16imi\_megacar\_startastar  loop區(但要用code)  開始A\*路徑規劃: 終點  收件人  物流站 GOODS\_POINT  充電站 CHARGE\_POINT  收件人 Recipient\_POINT | //A\*路徑歸劃  'bool isFindPath = starAStarPlan('+dropdown\_point+');'  //開始導航  'starNav(isFindPath,' +dropdown\_point+');' |
| \_17imi\_megacar\_standbyaicam  loop區(但要用code)  AI鏡頭朝向,準備人臉識別收貨人 | //抵達目的地,AI鏡頭朝向,準備人臉識別收貨人  if (isFindPath) {  standByAiCam();  } |
| \_18imi\_megacar\_esp32line  loop區(但要用code)  [傳送]通知至ESP32：LINE通知收件人 | //完成送貨，發送LINE通知收貨人(格式:LINE\_NOTIFY,姓名,商品)  if (isFindPath) {  ESP32Serial.print(String(LINE\_NOTIFY) + "," + recipient[0] + recipient[1]);  } |
| \_19imi\_megacar\_startdetectobject  loop區(但要用code)  等待AI人臉識別  statements\_msg  \_20imi\_megacar\_isfaceid  人臉ID=  **1**  'aiId == FACE\_ID1'  \_21imi\_megacar\_currentalgo  AI識別模式切換為:  物件識別  'currentAlgo ="'+ dropdown\_algo+'";'  value\_algo字串””  物件識別 ALGORITHM\_OBJECT\_RECOGNITION  人臉識別 ALGORITHM\_FACE\_RECOGNITION | //等待AI人臉識別  if (CurrentAlgo == "ALGORITHM\_FACE\_RECOGNITION") {  //開始AI人臉識別  startDetectObject();  Statements區塊  } |
| \_22imi\_megacar\_issign  loop區(但要用code)  按下簽收鈕 | 'digitalRead(BUTTON\_SIGN\_PIN) == 1' |
| \_23imi\_megacar\_carheadturn  loop區(但要用code)  車頭旋轉  朝下  'U', 'R', 'D', 'L'  value\_direct字元’ ‘ | '//抵達物流站，車子原地轉向(車頭要朝下D，否則貨斗開啟方向會卡住)'  'carHeadTurn('+dropdown\_direct+');' |
| \_24imi\_megacar\_getpower  loop區(但要用code)  取得小車當前電量  \_25imi\_megacar\_islowpower  電量低於  **20%**  \_26imi\_megacar\_startcharge  導航至[充電站]開始充電 | 'float percent = getBusPowerPercent(); //取得電池當前電壓百分比(小數)\n'  'currentPoint == GOODS\_POINT && percent <=' +dropdown\_percent  'startCharge();\n' |
| \_27imi\_megacar\_track\_func  循跡感測函式  statements\_msg | void trackForward() {  //先往前300ms,越過前方白線  forward();  delay(300);  //開始前進,直循跡感測踫到前方白線  while (!isFrontArrive) {  //取得循跡感測值  getTracks();  Statements區塊  } //end while  }//end trackForward |
| \_28imi\_megacar\_tracksensor  trackForward()  循跡感測被觸發  前方  TRACK\_LEFT、TRACK\_FRONT、TRACK\_BACK、TRACK\_RIGHT | 'trackSensor['+dropdown\_direct+'] == TRIGGLED' |
| \_29imi\_megacar\_stopfront  trackForward()  車子執行  結束此格 | backward();  delay(50);  stopCar();  //結束此格行走  isFrontArrive = true;  break; |
| \_30imi\_megacar\_direct  trackForward()  車子執行  對角右進  清單值：  對角右進 right  對角左進 left  前進 forward   |  | | --- | | var direct ="";  if(dropdown\_direct==right){  direct = "diagonalRight();";  }else if(dropdown\_direct==left){  direct = "diagonalLeft();";  }else if(dropdown\_direct==forward){  direct = "forward();";  }  var code =direct; | | diagonalRight();  diagonalLeft();  forward(); |
| \_31imi\_megacar\_isfindpath  A\*演算法有找到路徑 | 'isFindPath' |
| \_32imi\_megacar\_gocar\_func  開始自動駕駛函式  statements\_msg  \_33imi\_megacar\_aidetect  AI鏡頭開始識別行人、車子避障  'AiDetect();'  \_34imi\_megacar\_trackforward  循跡感測器開始偵測  'trackForward();'  \_35imi\_megacar\_esp32gps  [傳送]通知至ESP32：模擬GPS座標  'ESP32Serial.print(String(CAR\_GPS) + "," + pathXY[countPathXY++]);' | void goCar() {  //1.先將移動指令轉成字串  String stringPathCarMotor = "";  for (int i = 0; i < pathCount; i++) {  stringPathCarMotor += pathCarMove[i];  }  //2.字串轉成陣列  //使用 strtok() 函数分割字符串  int cacheLength = 200; //太小會裝不下  String pathCarMotor[cacheLength];  char\* token = strtok((char\*)stringPathCarMotor.c\_str(), ",");  int tokenLen = 0;  //字串轉成陣列  while (token != NULL && tokenLen < cacheLength) {  pathCarMotor[tokenLen] = token;  token = strtok(NULL, ",");  tokenLen++;  }  //3.開始移動車子  int countPathXY = 1;  for (int i = 0; i < tokenLen; i++) {  if (pathCarMotor[i] == "F") {  //AI識別行人,車子  //循跡感測  //雲端平台模擬GPS  Statements區塊  isFrontArrive = false;  delay(FTimer);  Serial.println("前,");  } else if (pathCarMotor[i] == "R") {  turnRight();  delay(RTimer);  stopCar();  delay(RTimer);  Serial.println("右轉,");  } else if (pathCarMotor[i] == "L") {  turnLeft();  delay(LTimer);  stopCar();  delay(LTimer);  Serial.println("左轉,");  }  }//end for  }//end goCar |
|  |  |
|  |  |