|  |
| --- |
| **MINISTRY OF EDUCATION & TRAINING**  **UNIVERSITY OF TECHNOLOGY AND EDUCATION**  **FACULTY OF MECHANICAL ENGINEERING**  http://hcmute.edu.vn/Resources/Images/SubDomain/HomePage/skpt_banner_2.jpg  **GRADUATION PROJECT**  ***Topic:***  **DESIGN AND MANUFACTURE PICKLEBALL MACHINE**  Instructor: **TS. ĐỖ VĂN HIẾN**  Performer: **NGUYỄN QUANG GIAO MSSV: 19146121**  **BÙI TRƯƠNG VĨNH PHÚC MSSV: 20146518**  Class: **19146CLA1 ; 201461A**  Course: **2019 – 2023 ; 2020-2024**    **Ho Chi Minh, January 2025** |

|  |
| --- |
| **UNIVERSITY OF TECHNOLOGY AND EDUCATION**  **FACULTY OF MECHANICAL ENGINEERING**  **DEPARTMENT OF MECHATRONICS**    **GRADUATION PROJECT**  ***Topic:* DESIGN AND MANUFACTURE PICKLEBALL MACHINE**  Instructor: **TS. ĐỖ VĂN HIẾN**  Performer: **NGUYỄN QUANG GIAO MSSV: 19146121**  **BÙI TRƯƠNG VĨNH PHÚC MSSV: 20146518**  Class: **19165CLA1 ; 201461A**  Course: **2019- 2023 ; 2020-2024**    **Ho Chi Minh, January 2025** |

**Code của Arduino**

//-----L298N-----

#define enA 11

#define in1 8

#define in2 9

#define in3 12

#define in4 13

#define enB 10

#define en1A 3

#define in1\_1 2

#define in1\_2 4

bool flag = false;

unsigned long forwardDurationStartTime = 0;

bool lastForward = false;

char lastCommand = '0';

void DongCo() {

analogWrite(en1A, 50);

digitalWrite(in1\_1, HIGH);

digitalWrite(in1\_2, LOW);

}

void Forward() {

analogWrite(enA, 90);

analogWrite(enB, 90);

digitalWrite(in1, LOW);

digitalWrite(in2, HIGH);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

}

void Back() {

analogWrite(enA, 90);

analogWrite(enB, 90);

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

digitalWrite(in3, LOW);

digitalWrite(in4, HIGH);

}

void Left() {

analogWrite(enA, 240);

analogWrite(enB, 240);

digitalWrite(in1, HIGH);

digitalWrite(in2, LOW);

digitalWrite(in3, HIGH);

digitalWrite(in4, LOW);

}

void Right() {

analogWrite(enA, 240);

analogWrite(enB, 240);

digitalWrite(in1, LOW);

digitalWrite(in2, HIGH);

digitalWrite(in3, LOW);

digitalWrite(in4, HIGH);

}

void Stop() {

digitalWrite(in1, LOW);

digitalWrite(in2, LOW);

digitalWrite(in3, LOW);

digitalWrite(in4, LOW);

}

void setup() {

Serial.begin(9600);

pinMode(in1, OUTPUT);

pinMode(in2, OUTPUT);

pinMode(in3, OUTPUT);

pinMode(in4, OUTPUT);

pinMode(enA, OUTPUT);

pinMode(enB, OUTPUT);

pinMode(en1A, OUTPUT);

pinMode(in1\_1, OUTPUT);

pinMode(in1\_2, OUTPUT);

DongCo();

countTime = millis();

}

void loop()

{

unsigned long currentMillis = millis();

if (Serial.available() > 0) {

char receivedChar = Serial.read();

Serial.println(receivedChar);

if(receivedChar == 'A') //

{

flag = false;

}

else if(receivedChar == 'M')

{

flag = true;

}

if(!flag)

{

if(receivedChar == '1' || receivedChar == '2' )

{

if(lastForward)

{

if (currentMillis - forwardDurationStartTime >= 800 || forwardDurationStartTime == 0)

{

forwardDurationStartTime = millis();

Forward();

lastForward = false;

}

}

else // {

forwardDurationStartTime = millis();

Left();

}

}

else if (receivedChar == '4' || receivedChar == '3' || receivedChar == '5')

{

Forward();

lastForward = true;

}

else if (receivedChar == '6' || receivedChar == '7'|| receivedChar == '0')

{

if (lastForward)

{

if (currentMillis - forwardDurationStartTime >= 800 || forwardDurationStartTime == 0)

{

forwardDurationStartTime = millis();

Forward();

lastForward = false;

}

}

else

{

forwardDurationStartTime = millis();

Right();

}

}

}

else if(flag)

{

if(receivedChar == 'F')

{

Forward();

}

else if(receivedChar == 'L')

{

Left();

}

else if(receivedChar == 'B')

{

Back();

}

else if(receivedChar == 'R')

{

Right();

}

else if(receivedChar == 'S')

{

Stop();

}

}

}

}

**Code xử lý ảnh**

import cv2

import numpy as np

import threading

from blynk\_function import \*

import serial

import time

try:

cap = cv2.VideoCapture(0)

except:

cap = cv2.VideoCapture(1)

greenlow = (30, 70, 90)

greenup = (45, 140, 255)

stop\_event = threading.Event()

position = '0'

# Define positions

positions = {

"1": (0, 1),#LEFT3

"2": (1, 2),#Left2

"3": (2, 3),#Left1

"4": (3, 4),#Forward

"5": (4, 5),#RIGHT1

"6": (5, 6),#RIGHT2

"7": (6, 7)#RIGHT3

}

def extract\_green(frame, greenlow, greenup):

blur = cv2.GaussianBlur(frame, (21, 21), 0)

hsv = cv2.cvtColor(blur, cv2.COLOR\_BGR2HSV)

mask = cv2.inRange(hsv, greenlow, greenup)

mask = cv2.dilate(mask, None, iterations=4)

return mask

def contour\_ext(mask, img):

conts, \_ = cv2.findContours(mask.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

center = None

position = None

if len(conts) > 0:

c = max(conts, key=cv2.contourArea)

perimeter = cv2.arcLength(c, True)

approx = cv2.approxPolyDP(c, .03 \* cv2.arcLength(c, True), True)

area = cv2.contourArea(c)

cv2.imshow('Disp Frame', mask)

#print(len(approx))

if len(approx)>1 and area / (perimeter \* perimeter) > 0.05:

#cv2.drawContours(img, [c], 0, (220, 152, 91), -1)

((x, y), radius) = cv2.minEnclosingCircle(c)

M = cv2.moments(c)

center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

if radius > 20:

cv2.circle(img, (int(x), int(y)), int(radius), (126, 255, 60), 2)

cv2.circle(img, center, 2, (75, 54, 255), 2)

position = get\_position(center, img.shape[1])

return img, position

def get\_position(center, img\_width):

x\_center = center[0]

segment\_width = img\_width / 7

segment\_id = int(x\_center // segment\_width) + 1

position = '0'

for key, value in positions.items():

if segment\_id in value:

position = key

break

return position

def thread\_function\_1():

global position, controlMode

while True:

\_, frame = cap.read()

frame=cv2.resize(frame,(640,480))

frame\_height, frame\_width, \_ = frame.shape

for i in range(1, 7):

x\_coordinate = frame\_width // 7 \* i

cv2.line(frame, (x\_coordinate, 0), (x\_coordinate, frame\_height), (255, 255, 255), 1, lineType=cv2.LINE\_AA)

disp = extract\_green(frame, greenlow, greenup)

frame, position = contour\_ext(disp, frame)

cv2.putText(frame, f'Ball Position: {position}', (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE\_AA)

cv2.imshow('Original Frame', frame)

if cv2.waitKey(20) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

def send\_data(position1):

global controlMode

if position1 == None:

position1 = '0'

if not controlMode:

ser.write(position1.encode())

time.sleep(0.5)

def thread\_function\_2():

global controlMode

while True: #not stop\_event.is\_set() and

send\_data(position)

print(f"Ball Position: {position}")

if \_\_name\_\_ == "\_\_main\_\_":

thread1 = threading.Thread(target=thread\_function\_1)

thread2 = threading.Thread(target=thread\_function\_2)

thread3 = threading.Thread(target=blynk\_activate)

thread1.start()

thread2.start()

thread3.start()

**Code xử lý ảnh chọn vùng màu của banh**

import cv2

import numpy as np

def get\_hsv\_value(event, x, y, flags, param):

if event == cv2.EVENT\_LBUTTONDOWN: # Check if the left mouse button was clicked

hsv\_value = hsv\_frame[y, x]

print(f"HSV Value at ({x}, {y}): {hsv\_value}")

# Initialize webcam capture

cap = cv2.VideoCapture(0)

cv2.namedWindow("Frame")

cv2.setMouseCallback("Frame", get\_hsv\_value)

while True:

\_, frame = cap.read()

hsv\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

# Display the frame

cv2.imshow("Frame", frame)

key = cv2.waitKey(1)

if key == 27: # Esc key to break

break

# Release the webcam and close windows

cap.release()

cv2.destroyAllWindows()

**Code giao tiếp Blynk với Raspberry Pi**

import BlynkLib

import serial

import time

BLYNK\_AUTH\_TOKEN = 'fv1QDFbmcSUs9KNC2Jm9npL75On4p04k'

blynk = BlynkLib.Blynk(BLYNK\_AUTH\_TOKEN)

controlMode = 0 is\_enable = 0

my\_variable = 0

forward = 0

left = 0

right = 0

backward = 0

#from TennisDetection import thread\_function\_1, thread\_function\_2

try:

ser = serial.Serial('/dev/ttyACM0',9600)

#ser = serial.Serial('/dev/ttyUSB0',9600)

except:

ser = serial.Serial('/dev/ttyACM1',9600)

#ser = serial.Serial('/dev/ttyUSB1',9600)

#FORWARD

@blynk.on("V1")

def v1\_write\_handler(value):

global is\_enable, forward

if is\_enable:

if(value[0] == '1'):

blynk.virtual\_write(2, 0)

blynk.virtual\_write(3, 0)

blynk.virtual\_write(4, 0)

forward = 'F'

else:

forwar = 'S'

ser.write(forward.encode())

print(f'FORWARD:{forward}')

#TURN LEFT

@blynk.on("V2")

def v2\_write\_handler(value):

global is\_enable, left

if is\_enable:

if(value[0] == '1'):

blynk.virtual\_write(1, 0)

blynk.virtual\_write(3, 0)

blynk.virtual\_write(4, 0)

left ='L'

else:

left = 'S'

ser.write(left.encode())

print(f'LEFT:{left}')

#BACKWARD

@blynk.on("V3")

def v3\_write\_handler(value):

global is\_enable, backward

if is\_enable:

if(value[0] == '1'):

blynk.virtual\_write(1, 0)

blynk.virtual\_write(2, 0)

blynk.virtual\_write(4, 0)

backward = 'B'

else:

backward = 'S'

ser.write(backward.encode())

print(f'BACKWARD:{backward}')

#TURN RIGHT

@blynk.on("V4")

def v4\_write\_handler(value):

global is\_enable, right

if is\_enable:

if(value[0] == '1'):

blynk.virtual\_write(1, 0)

blynk.virtual\_write(3, 0)

blynk.virtual\_write(2, 0)

right = 'R'

else:

right = 'S'

ser.write(right.encode())

print(f'RIGHT:{right}')

#MODE

@blynk.on("V5")

def v5\_write\_handler(value):

global controlMode, is\_enable

controlMode = value[0]

print(f'controlMode: {controlMode}')

if controlMode != '0' and controlMode != 0:

blynk.virtual\_write(1, 0)

blynk.virtual\_write(2, 0)

blynk.virtual\_write(3, 0)

blynk.virtual\_write(4, 0)

ser.write('M'.encode())

is\_enable = 1 # Bat mode manual

print('Mode: Manual')

else:

blynk.virtual\_write(1, 0)

blynk.virtual\_write(2, 0)

blynk.virtual\_write(3, 0)

blynk.virtual\_write(4, 0)

is\_enable = 0# Bat mode auto

ser.write('A'.encode())

print(controlMode)

#CHECK\_CONNECTION

@blynk.on("connected")

def blynk\_connected():

print("Raspberry Pi Connected to New Blynk")

def blynk\_activate():

while True:

blynk.run()

time.sleep(0.5)