**COLEGIUL NAȚIONAL MIHAI EMINESCU PETROȘANI**

Proiect Olimpiada de Inovare și Creație Digitală – Infoeducație

Secțiumea Roboți

MICROBOT

Profesori coordonator:

GAIȚĂ-LUKACS IOANA

SZEDLACSEK ROLLAND Elevi:

GAIȚĂ-LUKACS ALEXANDRU GEORGE

SZEDLACSEK ANTONIU

2022

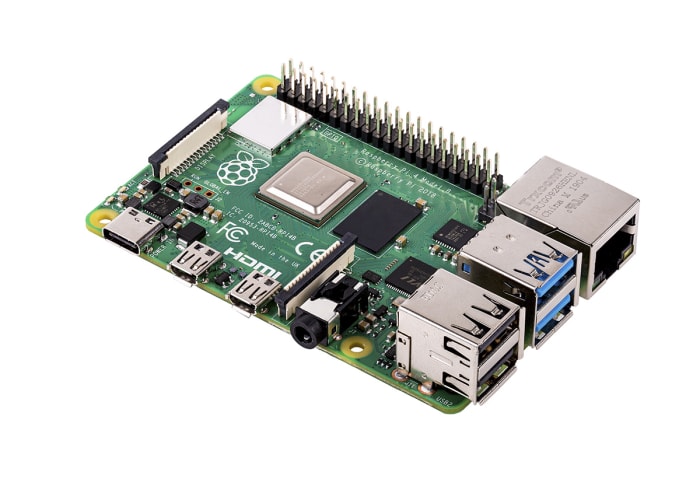
Capitolul I. Utilitate practică

O posibila utilitate practica ar putea fi de a fi plasat intr-un mediu necunoscut pentru a indeplini diverse sarcini cum ar fi: aflarea temperaturii din zona respectiva, luminii, si in viitor dorim sa ii adaugam un senzor de gaze, monoxid de carbon, umiditate si presiune atmosferica. De asemenea, cu ajutorul camerei web plasata pe robot putem sa implementam aplicatii de recunoastere faciala, recunoasterea gesturilor, culorilor, urmarirea unui obiect.

Capitolul II. Mecanica

Robotul este format dintr-o placa de baza din fibra de sticla epoxidica, care are rol de driver pentru cele 4 motoare cu reductie 1:48. Motoarele sunt alimentate la curent continuu si accepta o tensiune intre 3V si 6 V, avand in medie 200 de rotatii pe minut. Pe langa aceste 4 motoare mai avem 2 motoare de tip servo cu libertate in miscare de ~270 de grade care ajuta la orientarea unei camere web de 5Mp. Robotul este alimentat la o baterie de 12V - 6A care asigura o durata de functionare a robotului de aproximativ 4h, iar timpul de incarcare al bateriei este de aproximativ 2h de la un panou solar 12V la o putere de 375W. Microbit-ul foloseste un set de 2 baterii de zinc AAA reincarcabile pe care le conectam la un incarcator de baterii conectat la panoul solar.

Cap III. Electronica

****Robotul este controlat de o placa Raspberry Pi 4G avand un microprocesor Cortex A72 pe 64 biti, 1.5Ghz cu patru nuclee fizice, 4GB memorie RAM, conectivitate wi-fi si bluetooth, 2 conectoare micro HDMI si placa video OpenGLES 3.0, alimentata la o tensiune de 5V. Ca sistem de operare am folosit un sistem Debiant care este o distributie Linux, numita Raspbiant

O pereche de senzori ultrasonici pentru masurarea distantei cu ultrasunete care poate masura sau detecta obiectul in intervalul 2-450 cm. Acestia se folosesc pentru a evita obstacolele, cat si pentru masurarea distantei pana la un anumit obiect. 4 senzori cu infrarosu pentru urmarirea unei linii negre.

Robotul poate indeplini mai multe functii cum ar fi:

-Controlul deplasarii robotului:

-Evitarea obstacolelor

-Urmarirea unei linii continue negre

Pe placuta microbit avem un Procesor de aplicatii nRF51 unde rulează programele utilizatorului. O singură aplicație completă, care include codul utilizatorului, codul de rulare și stiva bluetooth este încărcată și rulată direct din memoria flash pe cip. Display-ul este alcatuit dintr-o https://tech.microbit.org/hardware/1-5-revision/ - nrf51-application-processor

A close up of a computer chip

Description automatically generated with low confidenceVarianta de 1.5 micro:biți are un cip combinat de accelerometru și magnetometru care oferă senzori pe 3 axe și detectarea intensității câmpului magnetic. Include, de asemenea, detectarea gesturilor la bord (cum ar fi detectarea căderii) în hardware și detectare suplimentară a gesturilor (de exemplu, logo-ul sus, sigla în jos, agitare) prin algoritmi software. Cu acesta am reusit sa programam compasul. Procesorul de aplicație nRF51 are un senzor de temperatură la bord pe care il folosim intr-un program pentru o estimare a temperaturii ambiante.

Cap IV. Software import smbus

import time

import math

class YB\_Pcb\_Car(object):

    def get\_i2c\_device(self, address, i2c\_bus):

        self.\_addr = address

        if i2c\_bus is None:

            return smbus.SMBus(1)

        else:

            return smbus.SMBus(i2c\_bus)

    def \_\_init\_\_(self):

        self.\_device = self.get\_i2c\_device(0x16, 1)

    def write\_u8(self, reg, data):

        try:

            self.\_device.write\_byte\_data(self.\_addr, reg, data)

        except:

            print ('write\_u8 I2C error')

    def write\_reg(self, reg):

        try:

            self.\_device.write\_byte(self.\_addr, reg)

        except:

            print ('write\_u8 I2C error')

    def write\_array(self, reg, data):

        try:

            self.\_device.write\_i2c\_block\_data(self.\_addr, reg, data)

        except:

            print ('write\_array I2C error')

    def Ctrl\_Car(self, l\_dir, l\_speed, r\_dir, r\_speed):

        try:

            reg = 0x01

            data = [l\_dir, l\_speed, r\_dir, r\_speed]

            self.write\_array(reg, data)

        except:

            print ('Ctrl\_Car I2C error')

    def Control\_Car(self, speed1, speed2):

        try:

            if speed1 < 0:

                dir1 = 0

            else:

                dir1 = 1

            if speed2 < 0:

                dir2 = 0

            else:

                dir2 = 1

            self.Ctrl\_Car(dir1, int(math.fabs(speed1)), dir2, int(math.fabs(speed2)))

        except:

            print ('Ctrl\_Car I2C error')

    def Car\_Run(self, speed1, speed2):

        try:

            self.Ctrl\_Car(1, speed1, 1, speed2)

        except:

            print ('Car\_Run I2C error')

    def Car\_Stop(self):

        try:

            reg = 0x02

            self.write\_u8(reg, 0x00)

        except:

            print ('Car\_Stop I2C error')

    def Car\_Back(self, speed1, speed2):

        try:

            self.Ctrl\_Car(0, speed1, 0, speed2)

        except:

            print ('Car\_Back I2C error')

    def Car\_Left(self, speed1, speed2):

        try:

            self.Ctrl\_Car(0, speed1, 1, speed2)

        except:

            print ('Car\_Spin\_Left I2C error')

    def Car\_Right(self, speed1, speed2):

        try:

            self.Ctrl\_Car(1, speed1, 0, speed2)

        except:

            print ('Car\_Spin\_Left I2C error')

    def Car\_Spin\_Left(self, speed1, speed2):

        try:

            self.Ctrl\_Car(0, speed1, 1, speed2)

        except:

            print ('Car\_Spin\_Left I2C error')

    def Car\_Spin\_Right(self, speed1, speed2):

        try:

            self.Ctrl\_Car(1, speed1, 0, speed2)

        except:

            print ('Car\_Spin\_Right I2C error')

    def Ctrl\_Servo(self, id, angle):

        try:

            reg = 0x03

            data = [id, angle]

            if angle < 0:

                angle = 0

            elif angle > 180:

                angle = 180

            self.write\_array(reg, data)

        except:

            print ('Ctrl\_Servo I2C error')

import RPi.GPIO as GPIO

import time

import YB\_Pcb\_Car

car = YB\_Pcb\_Car.YB\_Pcb\_Car()

PIN = 36;

buzzer = 32;

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

ir\_repeat\_cnt = 0

def init():

    GPIO.setup(PIN,GPIO.IN,GPIO.PUD\_UP)

    GPIO.setup(buzzer,GPIO.OUT)

    print("IR control start...")

#whistle

def whistle():

    p = GPIO.PWM(buzzer, 440)

    p.start(50)

    time.sleep(0.5)

    p.stop()

def exec\_cmd(key\_val):

    if key\_val==0x45:  #Power button

        car.Ctrl\_Servo(1, 90)

        car.Ctrl\_Servo(2, 90)

        car.Car\_Stop()

    elif key\_val==0x40:   #+ button

        car.Car\_Run(100, 100)   #car advance

    elif key\_val==0x15:   #Stop button

        car.Car\_Stop()

    elif key\_val==0x07:   #Left button

        car.Car\_Left(100, 100)

    elif key\_val==0x47:   #MENU button

        whistle()         #buzzer whistle

    elif key\_val==0x09:   #Right button

        car.Car\_Right(100, 100)

    elif key\_val==0x16:   #0 button

        car.Car\_Spin\_Left(100, 100)

    elif key\_val==0x19:   #- button

        car.Car\_Back(100, 100)

    elif key\_val==0x0d:   #C button

        car.Car\_Spin\_Right(100, 100)

    elif key\_val==0x0c:   #1 button

        car.Ctrl\_Servo(1, 0)

    elif key\_val==0x18:   #2 button

        car.Ctrl\_Servo(1, 90)

    elif key\_val==0x5e:   #3 button

        car.Ctrl\_Servo(1, 180)

    elif key\_val==0x08:   #4 button

        car.Ctrl\_Servo(2, 0)

    elif key\_val==0x1c:   #5 button

        car.Ctrl\_Servo(2, 90)

    elif key\_val==0x5a:   #6 button

        car.Ctrl\_Servo(2, 180)

    else:

        print(key\_val)

        print("no cmd")

try:

    init()

    while True:

        if GPIO.input(PIN) == 0:

            ir\_repeat\_cnt = 0;

            count = 0

            while GPIO.input(PIN) == 0 and count < 200:

                count += 1

                time.sleep(0.00006)

            count = 0

            while GPIO.input(PIN) == 1 and count < 80:

                count += 1

                time.sleep(0.00006)

            idx = 0

            cnt = 0

            data = [0,0,0,0]

            for i in range(0,32):

                count = 0

                while GPIO.input(PIN) == 0 and count < 15:

                    count += 1

                    time.sleep(0.00006)

                count = 0

                while GPIO.input(PIN) == 1 and count < 40:

                    count += 1

                    time.sleep(0.00006)

                if count > 9:

                    data[idx] |= 1<<cnt

                if cnt == 7:

                    cnt = 0

                    idx += 1

                else:

                    cnt += 1

            if data[0]+data[1] == 0xFF and data[2]+data[3] == 0xFF:

                print("Get the key: 0x%02x" %data[2])

                exec\_cmd(data[2])

        else:

            if ir\_repeat\_cnt > 110:

                ir\_repeat\_cnt = 0

                car.Car\_Stop()

            else:

                time.sleep(0.001)

                ir\_repeat\_cnt += 1

except KeyboardInterrupt:

    pass

print("Ending")

GPIO.cleanup()