UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

ESEIAAT

Grau en Enginyeria en Tecnologies Industrials

PROJECTE DE MILLORA DEL SISTEMA D'ENFOCAMENT D'UN MICROSCOPI DE BAIX COST

ANNEXES A LA MEMÒRIA

Autor: Oriol Rabasa Galan

Director: David González Diez

Curs: 2019-2020 Q2

Data lliurament: 30 de Juny del 2020

Sumari

Annex A

En aquest apartat es mostra tot el codi font en Python que controla el microscopi automàtic AIScope.

```
#-------------------------------------------------------------------------------------------------------------------#
#------AISCOPE WORKING CODE V3 -------- WITHOUT MOBILE PHONE CONNECTION --------#
#------Author: Oriol Rabasa Galan ----------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------------------------------------------#

import RPi.GPIO as gpio
import datetime
import numpy as np
import math
from picamera import PiCamera
import time
from time import sleep
import cv2

#-------------------------------------------------------------------------------------------------------------------#
#----------INPUT / OUTPUT PINS AND CAMERA CONFIGURATION----------------------#
#-------------------------------------------------------------------------------------------------------------------#

#PIN LAYOUT
xServo=15
yServo=18
zServo=14
LED=23
xSensor_0=7
xSensor_1=8
ySensor_0=1
ySensor_1=12
zSensor_0=16
zSensor_proximity=25

#ignore gpio initial warnings
gpio.setwarnings(False)

#set up the pin mode (this will determine the naming of the pins)
gpio.setmode(gpio.BCM)

#SERVOMOTORS OUTPUTS PINS set up
gpio.setup(xServo,gpio.OUT)
gpio.setup(yServo,gpio.OUT)
gpio.setup(zServo,gpio.OUT)

#set up the PWM frequency: 50 Hz
x=gpio.PWM(xServo,50)
y=gpio.PWM(yServo,50)
```

```
z=gpio.PWM(zServo,50)

#set up initial duty cycle of the PWM for the motors
x.start(0)
y.start(0)
z.start(0)

#LED OUTPUT PIN set up
gpio.setup(LED,gpio.OUT)

#SENSOR INPUTS PINS set up and pull-down resistors activation
gpio.setup(xSensor_0,gpio.IN,pull_up_down=gpio.PUD_DOWN)
gpio.setup(xSensor_1,gpio.IN,pull_up_down=gpio.PUD_DOWN)
gpio.setup(ySensor_0,gpio.IN,pull_up_down=gpio.PUD_DOWN)
gpio.setup(ySensor_1,gpio.IN,pull_up_down=gpio.PUD_DOWN)
gpio.setup(zSensor_0,gpio.IN,pull_up_down=gpio.PUD_DOWN)
gpio.setup(zSensor_proximity,gpio.IN,pull_up_down=gpio.PUD_DOWN)

#PI CAMERA RESOLUTION (1280x976 pixels)
focus_res_height=976
focus_res_width=1280

#PI CAMERA SETUP
camera = PiCamera()
camera.framerate = 60
camera.resolution = (focus_res_width, focus_res_height)


#-------------------------------------------------------------------------------------------------------------------#
#-------------------------FUNCTIONS-----------------------------------------------------------------------------------#
#-------------------------------------------------------------------------------------------------------------------#


#-----------------------------------------#
#--------FOCUS ALGORITHM--------#
#-----------------------------------------#
#Focus algorithm based on Sobel Derivatives

def Focus_Algorithm(img):

        #calculation of the sobel derivatives
        sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3)
        sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=3)
        sobel = np.abs(sobelx)+np.abs(sobely)

        #calculation of the variance of the sobel derivatives
        var1=np.var(sobel)

        #calculation of the modified variance of the sobel derivatives
        n=sobel.shape[0]*sobel.shape[1]
        mod1=(var1**2)*(n/(n-1))
```

```
        #returning the focus value, which is the square root of the modified variance
        return math.sqrt(mod1)

#-----------------------------------------#
#--------TAKE PICTURE-------------#
#-----------------------------------------#
#this function takes a picture and returns it

def Take_Picture():

        #create empty template for saving the picture
        picture = np.empty((focus_res_height, focus_res_width, 3), dtype=np.uint8)

        #take the picture
        camera.capture(picture, 'bgr')

        #return full picture
        return picture

#-----------------------------------------#
#--------FOCUS METHOD-----------#
#-----------------------------------------#

#this function focuses the sample and saves the most focused picture.

def Focus_Method(crop,number_pic):

        #initial parameters
        phase=1
        i=0
        i_max=0
        DT=0.0
        T=0.0
        focused=False

        #initialize lists to store the focus values
        SD_values=[]
        Increment_values=[]

        #set crop limits
        iy=crop[0]
        fy=crop[1]
        ix=crop[2]
        fx=crop[3]

        #image template to save the most focused picture
        best_picture = np.empty((focus_res_height, focus_res_width, 3), dtype=np.uint8)

        #approach sample to objective
        sample_approach(0.25)
```

```
while(focused==False and gpio.input(zSensor_proximity) == gpio.LOW and i<35):

        #move sample according to Duty Cycle(DT) and a delay Time(T)
        z.ChangeDutyCycle(DT)
        sleep(T)
        z.ChangeDutyCycle(0)

        #take picture (img) and turn it to gray-scale (img2)
        img = Take_Picture()
        img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        #analyze cropped image with focus algorithm
        SD_values.append(Focus_Algorithm(img2[iy:fy, ix:fx]))

        #calculate % of increment in focus value between actual and last picture
        Increment_values.append(((SD_values[i]-SD_values[i-1])/SD_values[i-1])*100)

        #store the most focused picture, according to the focus value
        if(SD_values[i]>SD_values[i_max]):
                i_max = i
                best_picture = img

        #finish focusing when three consecutive increment values are negatives,  or
        #when it's phase 4 and the max value is far away or the focus value has
        #decremented more than a 20%
        if(i>2 and Increment_values[i-2]<-2 and Increment_values[i-1]<-2 and
        Increment_values[i]<-2):
                focused=True

        if(phase==4 and (Increment_values[i]<-20 or i-i_max>4)):
                focused=True

        #change to the next phase if any increment is higher than the following values
        if((phase==1 and Increment_values[i]>40) or (phase==2 and
        Increment_values[i]>40) or (phase==3 and Increment_values[i]>30)):

                #exceptionally, if Increment is very high and phase<3 return a step back
                if(Increment_values[i]>70 and (phase!=4)):
                        DT=6.3
                        T=0.03

                phase+=1

        #set Duty Cycle(DT) and Time(T) according to current phase
        if(phase==1 and Increment_values[i]<=70):
                DT=9
                T=0.03

        elif(phase==2 and Increment_values[i]<=70):
                DT=8
```

```
                T=0.03

            elif(phase==3 and Increment_values[i]<=70):
                DT=7.6
                T=0.03

            elif(phase==4):
                DT=7.3
                T=0.02

            i+=1

        #save the focused picture as Sample_x, where 'x' is the number of the picture
        cv2.imwrite('Sample_'+str(number_pic)+'.jpg', best_picture)

        #separate sample from objective
        sample_separation()

        return 0

#-----------------------------------------#
#----------IMAGE CROP---------------#
#-----------------------------------------#

#this function defines the crop limits for focus pictures
#the function searches for the circle of the objective by searching non-black pixels (a square
#arround the circle is formed). Then, gets the center of the circle, draws a square smaller
#finally, returns the limits of this square in order to be used for slicing the focus pictures.

def Image_Crop(img):

        #create list to store the crop values
        crop_values=[]

        #turn picture to gray-scale
        img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        #initialize max and min crop values
        miny=img2.shape[0]
        minx=img2.shape[1]
        maxy=0
        maxx=0

        #analyze all the picture (skipping 3 rows and 3 columns each iteration, to finish earlier)
        #search for non-black zone of the picture, in order to set minimum and maximum
        #values for x and y

        for x in range(0,img2.shape[1]-1,3):
            for y in range(0,img2.shape[0]-1,3):
                if (img2[y][x]>20):
                    if(x<minx):
```

```
                        minx = x
                if(y<miny):
                        miny = y
                if(x>maxx):
                        maxx = x
                if(y>maxy):
                        maxy = y

        #calculate distances between the minimum and maximum value for x and y axis
        total_x = maxx-minx
        total_y = maxy-miny

        #define the crop limits, which will make the image smaller than the ones found earlier,
        #and store them inside crop_values list
        crop_values.append(int(miny + total_y/4))
        crop_values.append(int(maxy - total_y/4))
        crop_values.append(int(minx + total_x/4))
        crop_values.append(int(maxx - total_x/4))

        #return crop values list
        return crop_values


#----------------------------------------#
#---------MAPPING PATH ----------#
#----------------------------------------#

#this function returns a list with the directions for sample mapping
#move=0 -> x+, move=1 -> y+, move=2 -> x-, move=3 -> y-

def mapping(sx,sy):

        #initialize parameters
        map_list=[]
        movement=0
        i=0

        #Sample_x and sample_y are the number of pictures in each row and column.
        #The picture diameter is 0.2mm, so there can be taken 25 (5x5) pictures in 1 mm2
        sample_x=sx*5
        sample_y=sy*5

        #Aspect ratio of the sample. This will be useful if the sample isnt square-shaped
        if(sample_x>=sample_y):
                Q=int(sx/sy)
                R=1
        elif(sample_x<sample_y):
                R=int(sy/sx)
                Q=1

        movements_x_counter=Q
```

```
        movements_y_counter=R

        #List of moves. Movements define a spiral, which starts at the center of the sample.
        while(len(map_list)<(sample_y*sample_y)):

                if(i%2==0):
                        movements_counter=movements_x_counter
                        movements_x_counter+=Q

                else:
                        movements_counter=movements_y_counter
                        movements_y_counter+=R

                for t in range(0,movements_counter):
                        map_list.append(movement)

                movement+=1

                if(movement==4):
                        movement=0

                i+=1

        #return movements list
        return map_list


#-----------------------------------------#
#----------POSITION 0-------------------#
#-----------------------------------------#

#this function moves the sample to X0, Y0 and Z0 position (detected by xSensor_0, ySensor_0
#and zSensor_0 sensors)

def position_0():

        #first move Z axis to protect objective and sample
        while(gpio.input(zSensor_0)== gpio.LOW):
                z.ChangeDutyCycle(5)
        z.ChangeDutyCycle(0)

        while(gpio.input(xSensor_0)== gpio.LOW):
                x.ChangeDutyCycle(5)
        x.ChangeDutyCycle(0)

        while(gpio.input(ySensor_0)== gpio.LOW):
                y.ChangeDutyCycle(10)
        y.ChangeDutyCycle(0)

        return 0
```

```
#-----------------------------------------------#
#-----SAMPLE APPROACH Z AXIS-----#
#-----------------------------------------------#
#this function approaches the sample to the objective (when zSensor_proximity detects)

def sample_approach(delay):

        #approach sample to initial Z position
        while(gpio.input(zSensor_proximity)==gpio.HIGH):
                z.ChangeDutyCycle(9)
        sleep(delay) #this delay does more approach for the sample
        z.ChangeDutyCycle(0)

        return 0


#-----------------------------------------------#
#----SAMPLE SEPARATION Z AXIS----#
#-----------------------------------------------#
#this function separates the sample from objective (when zSensor_proximity doesnt detect)

def sample_separation():

        #move sample back to initial Z position
        while(gpio.input(zSensor_proximity)==gpio.LOW):
                z.ChangeDutyCycle(5)
        sleep(0.02)
        z.ChangeDutyCycle(0)

        return 0


#-----------------------------------------------#
#---CENTER SAMPLE X-Y PLANE-----#
#-----------------------------------------------#

#centers the sample at the objective.

def center_sample(time_x, time_y):

        #Center sample for X - Y axis
        x.ChangeDutyCycle(9)
        sleep(time_x)
        x.ChangeDutyCycle(0)

        y.ChangeDutyCycle(6)
        sleep(time_y)
        y.ChangeDutyCycle(0)

        return 0
```

```
#-----------------------------------------------#
#---------SAMPLE MOVEMENTS-------#
#-----------------------------------------------#

#this function moves the sample horizontally according to an input number
#depending on the input number, the sample will move to a certain X or Y direction
# // '0' -> x+ // '1' -> y+ // '2' -> x- // '3' -> y- //

def New_Movement(move):

        if(move==0 or move==1):
                DT=9
        else:
                DT=5

        if(move==0 or move==2):
                x.ChangeDutyCycle(DT)
                sleep(0.2)
                x.ChangeDutyCycle(0)

        else:
                y.ChangeDutyCycle(DT)
                sleep(0.2)
                y.ChangeDutyCycle(0)

        return 0

#-------------------------------------------------------------------------------------------------------------#
#-----------------------------------MAIN AISCOPE PROGRAM-------------------------------------#
#-------------------------------------------------------------------------------------------------------------#

#-----------------------------------------------------------------#
#-------INITIAL POSITIONING OF THE SAMPLE-------#
#-----------------------------------------------------------------#

#Move the sample to point (X0, Y0 , Z0)
position_0()

#Center sample X & Y
center_sample(8, 8)

#approach sample to the objective
sample_approach(0.25)

#-----------------------------------------------------------------#
#-----------------FOCUSING THE SAMPLE----------------#
#-----------------------------------------------------------------#

#turning the LED ON
gpio.output(LED,gpio.HIGH)
```

```
#initial parameters
number_pic=1
limit_switch_detected=False
time_focus=[]
iterations_yes=[]

#sample dimensions in mm.
mm_x=2
mm_y=2

#define the list of movements to analyze all the sample
movements_list = mapping(mm_x,mm_y)

#define the crop limits for the focus pictures
crop_picture=Take_Picture()
crop_limits=Image_Crop(crop_picture)

#the process will stop when all the pictures have been taken or any x/y limit switch is activated
while (number_pic<len(movements_list) and limit_switch_detected==False):

        #approach, focus image, save focused picture, separate sample from the objective
        Focus_Method(crop_limits,number_pic)

        #move sample to new position
        New_Movement(movements_list[number_pic])

        number_pic+=1

        #check if any limit switch is activated
        limit_switch_detected = gpio.input(ySensor_0)==gpio.HIGH or
        gpio.input(ySensor_1)==gpio.HIGH or gpio.input(xSensor_0)==gpio.HIGH or
        gpio.input(xSensor_1)==gpio.HIGH


#-------------------------------------------------------------#
#--------------------END OF THE PROCESS----------------#
#-------------------------------------------------------------#

#Move the sample to point (X0, Y0 , Z0)
position_0()

#turning LED off
gpio.output(LED,gpio.HIGH)

#deactivate servomotors
x.stop(0)
y.stop(0)
z.stop(0)

#clean INPUT/OUTPUT pins
gpio.cleanup()
```