NIGHT CRAWLER

The Night Crawler is a 3D printed, six-legged robot capable of detecting holes and obstacles in front of it thanks to its Infrared Sensor and correcting path in case of unexpected rotations thanks to its IMU. It also possesses a night vision camera and a wi fi connection ability, which in addition to the relatively high computing performance provided by the Raspberry pi zero W CPU leaves room for a variety of potential application. For example I'm in the process of designing a Convolutional Neural Network for image recognition from the robot's camera, which will give the Night Crawler the ability to recognize the objects in front of it and make more intelligent decisions on how to avoid (or just ignore) them.

COMPONENTS:

3D Printer model: Anet A3
Plastic used: PLA

Main Board: Raspberry Pi Zero W
IR sensor: Sharp GP2Y0A21YK0F
Servo Shield: PCA9685
IMU: MPU6050
Analog to I2C converter: ADS1015
Sevos: 6 x Tower Pro MG90S (declared torque 2.0 kg/cm)
4 x Emax ES08MA2 (declared torque: 1.8 kg/cm)
3 x Emax ES08A2 (declared torque: 1.6 kg/cm)
Main Battery: 2100 mAh 6 V
Secondary Battery: 1400 mAh 6 V

Total Mass: 0,76 kg

Main Movements Demonstrator Code:

```
import time
from random import randint
import math
import Adafruit_ADS1x15
from threading import Thread
import mpu6050
import Adafruit_PCA9685



#MOVEMENTS

bodyradius=6.0
legradius=6.5
tibia=5.5
femur=4.0
servo_max=620
servo_min=120
```

```
#servos real mid position,direction of rotation and channel
SMP=[30,10,50,-30,10,-10,10,16,10,25,0,10]
SD=[-1,-1,-1,1,1,1,-1,-1,-1,1,1,1]
SC=[0,1,2,3,4,5,7,6,11,12,13,14]


pwm=Adafruit_PCA9685.PCA9685()
pwm.__init__
pwm.set_pwm_freq(60)

servo=[0,0,0,0,0,0,0,0,0,0,0,0]
Gyro=0


def Proportion(x,in_min,in_max,out_min,out_max):

    return int((x-in_min)*(out_max-out_min)
            /(in_max-in_min)+out_min)


scont=[0,0,0,0,0,0,0,0,0,0,0,0]
scont[0]=Proportion(SMP[0],-90,90,servo_min,servo_max)
scont[1]=Proportion(SMP[1],-90,90,servo_min,servo_max)
scont[2]=Proportion(SMP[2],-90,90,servo_min,servo_max)
scont[3]=Proportion(SMP[3],-90,90,servo_min,servo_max)
scont[4]=Proportion(SMP[4],-90,90,servo_min,servo_max)
scont[5]=Proportion(SMP[5],-90,90,servo_min,servo_max)
scont[6]=Proportion(SMP[6],-90,90,servo_min,servo_max)
scont[7]=Proportion(SMP[7],-90,90,servo_min,servo_max)
scont[8]=Proportion(SMP[8],-90,90,servo_min,servo_max)
scont[9]=Proportion(SMP[9],-90,90,servo_min,servo_max)
scont[10]=Proportion(SMP[10],-90,90,servo_min,servo_max)
scont[11]=Proportion(SMP[11],-90,90,servo_min,servo_max)


def MovementIteration(servo,speed) :
    """
    moves each servo towards its target position by a small bit
    in order to have an apparent simoultaneous movement
    """


    for i in range(12):
        remainder=abs(servo[i]-scont[i])%speed
        if(remainder>0):
            if(scont[i]>servo[i]):
                scont[i]-=remainder
            else:
                scont[i]+=remainder
```

```
    finished=False

    while (finished==False):
      for i in range(12):
         if(scont[i]!=servo[i]):
            if(scont[i]>servo[i]):
               scont[i]-=speed
            else:
               scont[i]+=speed

         pwm.set_pwm(SC[i],0,scont[i])
         time.sleep(0.004)


      finished=True
      for i in range(12):
         if(scont[i]!=servo[i]):
            finished=False



def Walk(step,heigth,speed,path):
    print("walking...")


    report=0


    #calculates the angle by which the mid body servos
    #must move from their rest position in order to reach
    #the step value
    Alfa1=math.degrees(math.asin(step/legradius))

    #calculates the angle by which the front and back servos
    #must move from their rest position in order to reach
    #the step value
    Alfa2=math.degrees(math.asin(step/legradius*
                               math.sin(math.radians(45))))

    #calculates the angle by which the heigth regulating
    #servos must move from the 0 degrees position to reach
    #the heigth value
    Beta=-math.degrees(math.asin((heigth-tibia)/femur))

    #checks if there is any obstacle that could
    #prevent the movement or even damage the robot
    if (IRcheck(heigth)=="clear" or step<0):

       #calculating the signal to be send to each servo
       #to achieve the first phase of the walk movement

       servo[0]=Proportion(SMP[0]+SD[0]*Alfa1,-90,90,servo_min,servo_max)
```

```
    servo[1]=Proportion(SMP[1],-90,90,servo_min,servo_max)
    servo[2]=Proportion(SMP[2]+SD[2]*Alfa1,-90,90,servo_min,servo_max)
    servo[3]=Proportion(SMP[3],-90,90,servo_min,servo_max)
    servo[4]=Proportion(SMP[4]+SD[4]*Alfa2,-90,90,servo_min,servo_max)
    servo[5]=Proportion(SMP[5],-90,90,servo_min,servo_max)
    servo[6]=Proportion(SMP[6]+SD[6]*20,-90,90,servo_min,servo_max)
    servo[7]=Proportion(SMP[7]+SD[7]*Beta,-90,90,servo_min,servo_max)
    servo[8]=Proportion(SMP[8]+SD[8]*20,-90,90,servo_min,servo_max)
    servo[9]=Proportion(SMP[9]+SD[9]*Beta,-90,90,servo_min,servo_max)
    servo[10]=Proportion(SMP[10]+SD[10]*20,-90,90,servo_min,servo_max)
    servo[11]=Proportion(SMP[11]+SD[11]*Beta,-90,90,servo_min,servo_max)

    MovementIteration(servo,speed)

    servo[6]=Proportion(SMP[6]+SD[6]*Beta,-90,90,servo_min,servo_max)
    servo[8]=Proportion(SMP[8]+SD[8]*Beta,-90,90,servo_min,servo_max)
    servo[10]=Proportion(SMP[10]+SD[10]*Beta,-90,90,servo_min,servo_max)

    MovementIteration(servo,speed)


else:
    report-=1


if (IRcheck(heigth)=="clear"or step<0):

    #swap target positions between servos form one
    #side to the other and recalculate the signal
    #to be send to each servo
    servo[0]=Proportion(SMP[0],-90,90,servo_min,servo_max)
    servo[1]=Proportion(SMP[1]+SD[1]*Alfa2,-90,90,servo_min,servo_max)
    servo[2]=Proportion(SMP[2],-90,90,servo_min,servo_max)
    servo[3]=Proportion(SMP[3]+SD[3]*Alfa1,-90,90,servo_min,servo_max)
    servo[4]=Proportion(SMP[4],-90,90,servo_min,servo_max)
    servo[5]=Proportion(SMP[5]+SD[5]*Alfa1,-90,90,servo_min,servo_max)
    servo[6]=Proportion(SMP[6]+SD[6]*Beta,-90,90,servo_min,servo_max)
    servo[7]=Proportion(SMP[7]+SD[7]*20,-90,90,servo_min,servo_max)
    servo[8]=Proportion(SMP[8]+SD[8]*Beta,-90,90,servo_min,servo_max)
    servo[9]=Proportion(SMP[9]+SD[9]*20,-90,90,servo_min,servo_max)
    servo[10]=Proportion(SMP[10]+SD[10]*Beta,-90,90,servo_min,servo_max)
    servo[11]=Proportion(SMP[11]+SD[11]*20,-90,90,servo_min,servo_max)

    MovementIteration(servo,speed)

    servo[7]=Proportion(SMP[7]+SD[7]*Beta,-90,90,servo_min,servo_max)
    servo[9]=Proportion(SMP[9]+SD[9]*Beta,-90,90,servo_min,servo_max)
    servo[11]=Proportion(SMP[11]+SD[11]*Beta,-90,90,servo_min,servo_max)

    MovementIteration(servo,speed)
```

```
    else:
        report-=1

    if(report<0 or step<0):

        #set the servos' target position to their rest position
        #and recalculate the signal to be send to each servo
        servo[0]=Proportion(SMP[0],-90,90,servo_min,servo_max)
        servo[1]=Proportion(SMP[1],-90,90,servo_min,servo_max)
        servo[2]=Proportion(SMP[2],-90,90,servo_min,servo_max)
        servo[3]=Proportion(SMP[3],-90,90,servo_min,servo_max)
        servo[4]=Proportion(SMP[4],-90,90,servo_min,servo_max)
        servo[5]=Proportion(SMP[5],-90,90,servo_min,servo_max)
        servo[6]=Proportion(SMP[6]+SD[6]*20,-90,90,servo_min,servo_max)
        servo[7]=Proportion(SMP[7]+SD[7]*Beta,-90,90,servo_min,servo_max)
        servo[8]=Proportion(SMP[8]+SD[8]*20,-90,90,servo_min,servo_max)
        servo[9]=Proportion(SMP[9]+SD[9]*Beta,-90,90,servo_min,servo_max)
        servo[10]=Proportion(SMP[10]+SD[10]*20,-90,90,servo_min,servo_max)
        servo[11]=Proportion(SMP[11]+SD[11]*Beta,-90,90,servo_min,servo_max)

        MovementIteration(servo,speed)

        servo[6]=Proportion(SMP[6]+SD[6]*Beta,-90,90,servo_min,servo_max)
        servo[8]=Proportion(SMP[8]+SD[8]*Beta,-90,90,servo_min,servo_max)
        servo[10]=Proportion(SMP[10]+SD[10]*Beta,-90,90,servo_min,servo_max)

        MovementIteration(servo,speed)

    PathCorrection(path,heigth,speed)

    return report



def Turn(angle,heigth,speed):

    print("Turning...")

    #calculates the angle by which the heigth regulating
    #servos must move from the 0 degrees position to reach
    #the heigth value
    Beta=-math.degrees(math.asin((heigth-tibia)/femur))

    #calculates the angle by which the servos must move
    #from their rest position to reach the input angle value
    Alfa=angle+math.degrees(math.asin(bodyradius*
                        math.sin(math.radians(angle))/legradius))
```

```
#it splits the turning movement in two separate ones
#if the requested turning angle is too big
if(Alfa>40 or Alfa<-40):
    Turn(angle/2,heigth,speed)
    Turn(angle/2,heigth,speed)

else:
    #calculating the signal to be send to each servo
    #to achieve the first phase of the turn movement
    servo[0]=Proportion(SMP[0],-90,90,servo_min,servo_max)
    servo[1]=Proportion(SMP[1]-SD[1]*Alfa,-90,90,servo_min,servo_max)
    servo[2]=Proportion(SMP[2],-90,90,servo_min,servo_max)
    servo[3]=Proportion(SMP[3]+SD[3]*Alfa,-90,90,servo_min,servo_max)
    servo[4]=Proportion(SMP[4],-90,90,servo_min,servo_max)
    servo[5]=Proportion(SMP[5]+SD[5]*Alfa,-90,90,servo_min,servo_max)
    servo[6]=Proportion(SMP[6]+SD[6]*Beta,-90,90,servo_min,servo_max)
    servo[7]=Proportion(SMP[7]+SD[7]*20,-90,90,servo_min,servo_max)
    servo[8]=Proportion(SMP[8]+SD[8]*Beta,-90,90,servo_min,servo_max)
    servo[9]=Proportion(SMP[9]+SD[9]*20,-90,90,servo_min,servo_max)
    servo[10]=Proportion(SMP[10]+SD[10]*Beta,-90,90,servo_min,servo_max)
    servo[11]=Proportion(SMP[11]+SD[11]*20,-90,90,servo_min,servo_max)

    MovementIteration(servo,speed)

    servo[7]=Proportion(SMP[7]+SD[7]*Beta,-90,90,servo_min,servo_max)
    servo[9]=Proportion(SMP[9]+SD[9]*Beta,-90,90,servo_min,servo_max)
    servo[11]=Proportion(SMP[11]+SD[11]*Beta,-90,90,servo_min,servo_max)

    MovementIteration(servo,speed)

    #swap target positions between servos form one
    #side to the other and recalculate the signal
    #to be send to each servo
    servo[0]=Proportion(SMP[0]-SD[0]*Alfa,-90,90,servo_min,servo_max)
    servo[1]=Proportion(SMP[1],-90,90,servo_min,servo_max)
    servo[2]=Proportion(SMP[2]-SD[2]*Alfa,-90,90,servo_min,servo_max)
    servo[3]=Proportion(SMP[3],-90,90,servo_min,servo_max)
    servo[4]=Proportion(SMP[4]+SD[4]*Alfa,-90,90,servo_min,servo_max)
    servo[5]=Proportion(SMP[5],-90,90,servo_min,servo_max)
    servo[6]=Proportion(SMP[6]+SD[6]*20,-90,90,servo_min,servo_max)
    servo[7]=Proportion(SMP[7]+SD[7]*Beta,-90,90,servo_min,servo_max)
    servo[8]=Proportion(SMP[8]+SD[8]*20,-90,90,servo_min,servo_max)
    servo[9]=Proportion(SMP[9]+SD[9]*Beta,-90,90,servo_min,servo_max)
    servo[10]=Proportion(SMP[10]+SD[10]*20,-90,90,servo_min,servo_max)
    servo[11]=Proportion(SMP[11]+SD[11]*Beta,-90,90,servo_min,servo_max)

    MovementIteration(servo,speed)

    servo[6]=Proportion(SMP[6]+SD[6]*Beta,-90,90,servo_min,servo_max)
    servo[8]=Proportion(SMP[8]+SD[8]*Beta,-90,90,servo_min,servo_max)
    servo[10]=Proportion(SMP[10]+SD[10]*Beta,-90,90,servo_min,servo_max)
```

```
        MovementIteration(servo,speed)

        #set the servos' target position to their rest position
        #and recalculate the signal to be send to each servo
        servo[0]=Proportion(SMP[0],-90,90,servo_min,servo_max)
        servo[1]=Proportion(SMP[1],-90,90,servo_min,servo_max)
        servo[2]=Proportion(SMP[2],-90,90,servo_min,servo_max)
        servo[3]=Proportion(SMP[3],-90,90,servo_min,servo_max)
        servo[4]=Proportion(SMP[4],-90,90,servo_min,servo_max)
        servo[5]=Proportion(SMP[5],-90,90,servo_min,servo_max)
        servo[7]=Proportion(SMP[7]+SD[7]*20,-90,90,servo_min,servo_max)
        servo[9]=Proportion(SMP[9]+SD[9]*20,-90,90,servo_min,servo_max)
        servo[11]=Proportion(SMP[11]+SD[11]*20,-90,90,servo_min,servo_max)

        MovementIteration(servo,speed)

        servo[7]=Proportion(SMP[7]+SD[7]*Beta,-90,90,servo_min,servo_max)
        servo[9]=Proportion(SMP[9]+SD[9]*Beta,-90,90,servo_min,servo_max)
        servo[11]=Proportion(SMP[11]+SD[11]*Beta,-90,90,servo_min,servo_max)

        MovementIteration(servo,speed)


def PathCorrection(path,heigth,speed):
    global Gyro
    actualpath=Gyro
    if(abs(path-actualpath)>10):
        print("correcting: "+str(path-actualpath))
        print(actualpath)
        Turn((path-actualpath),heigth,speed)


#MANAGER

sensor=Adafruit_ADS1x15.ADS1x15.ADS1015()
sensor.__init__

sensor2=mpu6050.mpu6050(0x68)
sensor2.__init__
sensor2.set_gyro_range(0x08)
print(sensor2.read_gyro_range())

def Update():
    path=0
    while True:
        if(Walk(4,5.5,20,path)<0):
            print("obstacle")
            Walk(-4,5.5,20,path)
            direction=randint(0,1)
            if direction==0:
                direction=-1
            turnangle=direction*90
```

```
            print (turnangle)
            path+=turnangle
            Turn(-turnangle,5.5,20)
            PathCorrection(path,5.5,20)

def Calibration():
    total=0
    for i in range(2000):
        total+=sensor2.get_gyro_data()['z']
    offset=total/2000
    return offset


def GyroUpdate():
    global Gyro
    while(True):
        Gyro+=int((sensor2.get_gyro_data()['z']-offset)*0,004*2.5)
        time.sleep(0.004)


def IRcheck(heigth):
    expecteddistance=(heigth)/math.sin(math.radians(35))
    somma=0
    for i in range(50):
        somma+=float(sensor.read_adc(0))
    distance= 4080/(somma/50)
    if(distance>expecteddistance+3):
        return "hole"
    elif(distance<expecteddistance-3):
        return "wall"
    else:
        return "clear"

offset=Calibration()
P1=Thread(target=Update)
P2=Thread(target=GyroUpdate)
P1.start()
P2.start()
P1.join()
P2.join()
```