1. ANEXO A: Código del microcontrolador

1.1. main.py

```
import pyb
import imu
from pyb import I2C
from pyb import UART
import aux
#-----
pyb.LED(1).off()
pyb.LED(2).off()
pyb.LED(3).off()
pyb.LED(4).off()
#-----
text=aux.creatext()+'\n'
blnk=pyb.LED(2)
bot=pyb.Switch()
text+='leds\tok\n'

#----- inicialitzacio busos -----#

text+='checking UART\t ..... '
uart=UART(3,9600)
uart.init(9600,bits=8,stop=1,timeout=10,parity=None)
text+='ok\n'+'checking I2C\t ..... '
i2c=imu.InvenSenseMPU(1,1,(0,1,2),(1,1,1))
text+='ok\n'

#guardar al fitxer la inicialitzacio
text+='\n#------------------#\n\n'
print(text)
f=open('report.txt','w')
f.write(text)

#----- programa principal -----#

error=False
itera=0
cua=''
totaltram=[0,0]
dades=(3.1254,-1.3568,-2.0000,310.32,-12.36,0.21)
```

```
while error==False:
    mode=itera%2
    text='numero de iteracio:  ' +str(itera)+'  mode:
'+str(mode)+'\n'
    #---
    if itera%50==0:
        blnk.toggle()
        text+='blinking\t ..... ok\n'
    #---
    # interrupccio del programa + escriure a la sd
    if bot()==True:
        f.write(text)
        break

    text,dades=aux.mpuI2C(i2c,text)
    text=aux.enviar(uart,mode,dades,text)
    cua,text=aux.rebre(uart,'',text)
    report=aux.procesa(cua)
    text+='\t\t'+str(report[0])+' trames ok\t '+str(report[1])+'
trames error\n'
    totaltram[1]+=report[1]
    totaltram[0]+=report[0]

    #-----
    text+='iteracio acabada\n'
    itera+=1
    f.write(text)


f.write('trames enviades [ok/error] = '+ str(totaltram))
f.close()

pyb.LED(1).off()
pyb.LED(2).off()
pyb.LED(3).off()
pyb.LED(4).off()
```

1.2. aux.py

```
def creatext():
    text=''
    text+='#-------------------------------------#\n'
    text+='#  registre creat automaticament per   #\n'
    text+='#  verificar la execucio del programa  #\n'
    text+='#-------------------------------------#\n'
    return text

def prepara(mode,dades):
    string='#'
    cksm=0
    if mode==0:
        string+='A'
        for i in dades[0:3]:
            if i>=0:
                string+='+'
            else:
                string+='-'
            string+=str(abs(int(i*10000)))
            for j in str(abs(int(i*10000))):
                cksm+=int(j)
    elif mode==1:
        string+='G'
        for i in dades[3:]:
            if i>=0:
                string+='+'
            else:
                string+='-'
                i=abs(i)
            num=(str(int(i/100))+str(int((i%100)/10))+str(int
(i%10))+str(int(abs((i-int(abs(i)))*100))))
            string+=num
            for j in num:
                cksm+=int(j)
    string+='&'+str(cksm%100)+'%'
    return string

def enviar(bus,mode,dades,text):
    text+='\t----- '+str(bus)+' -----\n'
    text+='   escrivint bus: \t'
    string=prepara(mode,dades)
    text+=string+'\n'
    bus.write(string)
    return text
```

```
def rebre(bus,cua,text):
    text+='\t----- '+str(bus)+' -----\n'
    text+='   llegint bus: \t'
    msg=b''
    while bus.any():
        char=bus.read()
        msg+=char
    msg=str(msg).split("'")[1]
    text+=msg+'\n'
    cua+=msg
    return cua,text

def procesa(cua):
    llista=[0,0]
    reports=cua.split()
    while len(reports)>0:
        elem=reports.pop(0)
        elem=elem.split('x')
        try:
            llista[0]+=int(elem[0])
            llista[1]+=int(elem[-1])
        except:
            pass
    return llista

def mpuI2C(bus,text):
    text+='\n'
    text+='\t----- '+'MPU6050 I2C-bus'+' -----\n'
    buff=(bus.accel.x,bus.accel.y,bus.accel.z,bus.gyro.x,
bus.gyro.y,bus.gyro.z)
    text+='accelerometre:\t\tgiroscopi:\n'
    text+=str(buff[0])+'\t\t\t'+str(buff[3])+'\n'
    text+=str(buff[1])+'\t\t\t'+str(buff[4])+'\n'
    text+=str(buff[2])+'\t\t\t'+str(buff[5])+'\n'
    return text,buff
```

1.3. imu.py

```
import pyb
from vector3d import Vector3d


class MPUException(OSError):
    '''
    Exception for MPU devices
    '''
    pass

def bytes_toint(msb, lsb):
    '''
    Convert two bytes to signed integer (big endian)
    for little endian reverse msb, lsb arguments
    Can be used in an interrupt handler
    '''
    if not msb & 0x80:
        return msb << 8 | lsb  # +ve
    return - (((msb ^ 255) << 8) | (lsb ^ 255) + 1)


class InvenSenseMPU(object):
    '''
    Module for InvenSense 9DOF IMUs. Base class implements features
common to MPU9150 and MPU9250.
    '''

    _I2Cerror = "I2C failure when communicating with IMU"

    def __init__(self, side_str, device_addr, transposition,
scaling):

        self._accel = Vector3d(transposition, scaling, self._accel
_callback)
        self._gyro = Vector3d(transposition, scaling, self._gyro
_callback)
        self.buf1 = bytearray([0]*1)            # Pre-allocated
buffers for reads: allows reads to
        self.buf2 = bytearray([0]*2)            # be done in
interrupt handlers
        self.buf3 = bytearray([0]*3)
        self.buf6 = bytearray([0]*6)
        self.mpu_addr = 0x68
        self.timeout = 5                        # I2C tieout mS
```

```
        tim = pyb.millis()                          # Ensure PSU and
device have settled
        if tim < 200:
            pyb.delay(200-tim)

        self._mpu_i2c = pyb.I2C(1, pyb.I2C.MASTER)

        # Can communicate with chip. Set it up.
        self.wake()                                 # wake it up
        self.passthrough = True                     # Enable mag access
from main I2C bus
        self.accel_range = 0                        # default to highest
sensitivity
        self.gyro_range = 0                         # Likewise for gyro

    # read from device
    def _read(self, buf, memaddr, addr):        # addr = I2C device
address, memaddr = memory location within the I2C device
        '''
        Read bytes to pre-allocated buffer Caller traps OSError.
        '''
        self._mpu_i2c.mem_read(buf, addr, memaddr, timeout=self.
timeout)

    # write to device
    def _write(self, data, memaddr, addr):
        '''
        Perform a memory write. Caller should trap OSError.
        '''
        self._mpu_i2c.mem_write(data, addr, memaddr, timeout=
self.timeout)

    # wake
    def wake(self):
        '''
        Wakes the device.
        '''
        try:
            self._write(0x01, 0x6B, self.mpu_addr)  # Use best clock
source
        except OSError:
            raise MPUException(self._I2Cerror)
```

```
        return 'awake'

    # mode
    def sleep(self):
        '''
        Sets the device to sleep mode.
        '''
        try:
            self._write(0x40, 0x6B, self.mpu_addr)
        except OSError:
            raise MPUException(self._I2Cerror)
        return 'asleep'

    # passthrough
    @property
    def passthrough(self):
        '''
        Returns passthrough mode True or False
        '''
        try:
            self._read(self.buf1, 0x37, self.mpu_addr)
            return self.buf1[0] & 0x02 > 0
        except OSError:
            raise MPUException(self._I2Cerror)

    @passthrough.setter
    def passthrough(self, mode):
        '''
        Sets passthrough mode True or False
        '''
        if type(mode) is bool:
            val = 2 if mode else 0
            try:
                self._write(val, 0x37, self.mpu_addr)  # I think
this is right.
                self._write(0x00, 0x6A, self.mpu_addr)
            except OSError:
                raise MPUException(self._I2Cerror)
        else:
            raise ValueError('pass either True or False')

    # sample rate. Not sure why you'd ever want to reduce this from
the default.
    @property
    def sample_rate(self):
        '''
```

```
        Get sample rate as per Register Map document section 4.4
        SAMPLE_RATE= Internal_Sample_Rate / (1 + rate)
        default rate is zero i.e. sample at internal rate.
        '''
        try:
            self._read(self.buf1, 0x19, self.mpu_addr)
            return self.buf1[0]
        except OSError:
            raise MPUException(self._I2Cerror)

    @sample_rate.setter
    def sample_rate(self, rate):
        '''
        Set sample rate as per Register Map document section 4.4
        '''
        if rate < 0 or rate > 255:
            raise ValueError("Rate must be in range 0-255")
        try:
            self._write(rate, 0x19, self.mpu_addr)
        except OSError:
            raise MPUException(self._I2Cerror)

    # accelerometer range
    @property
    def accel_range(self):
        '''
        Accelerometer range
        Value:              0   1   2   3
        for range +/-:      2   4   8   16  g
        '''
        try:
            self._read(self.buf1, 0x1C, self.mpu_addr)
            ari = self.buf1[0]//8
        except OSError:
            raise MPUException(self._I2Cerror)
        return ari

    @accel_range.setter
    def accel_range(self, accel_range):
        '''
        Set accelerometer range
        Pass:               0   1   2   3
        for range +/-:      2   4   8   16  g
```

```
        '''
        ar_bytes = (0x00, 0x08, 0x10, 0x18)
        if accel_range in range(len(ar_bytes)):
            try:
                self._write(ar_bytes[accel_range], 0x1C,
self.mpu_addr)
            except OSError:
                raise MPUException(self._I2Cerror)
        else:
            raise ValueError('accel_range can only be 0, 1, 2 or 3')

    # gyroscope range
    @property
    def gyro_range(self):
        '''
        Gyroscope range
        Value:              0   1   2    3
        for range +/-:      250 500 1000 2000  degrees/second
        '''
        # set range
        try:
            self._read(self.buf1, 0x1B, self.mpu_addr)
            gri = self.buf1[0]//8
        except OSError:
            raise MPUException(self._I2Cerror)
        return gri

    @gyro_range.setter
    def gyro_range(self, gyro_range):
        '''
        Set gyroscope range
        Pass:               0   1   2    3
        for range +/-:      250 500 1000 2000  degrees/second
        '''
        gr_bytes = (0x00, 0x08, 0x10, 0x18)
        if gyro_range in range(len(gr_bytes)):
            try:
                self._write(gr_bytes[gyro_range], 0x1B,
self.mpu_addr)  # Sets fchoice = b11 which enables filter
            except OSError:
                raise MPUException(self._I2Cerror)
        else:
            raise ValueError('gyro_range can only be 0, 1, 2 or 3')

    # Accelerometer
    @property
```

```
    def accel(self):
        '''
        Acceleremoter object
        '''
        return self._accel

    def _accel_callback(self):
        '''
        Update accelerometer Vector3d object
        '''
        try:
            self._read(self.buf6, 0x3B, self.mpu_addr)
        except OSError:
            raise MPUException(self._I2Cerror)
        self._accel._ivector[0] = bytes_toint(self.buf6[0],
self.buf6[1])
        self._accel._ivector[1] = bytes_toint(self.buf6[2],
self.buf6[3])
        self._accel._ivector[2] = bytes_toint(self.buf6[4],
self.buf6[5])
        scale = (16384, 8192, 4096, 2048)
        self._accel._vector[0] =
self._accel._ivector[0]/scale[self.accel_range]
        self._accel._vector[1] =
self._accel._ivector[1]/scale[self.accel_range]
        self._accel._vector[2] =
self._accel._ivector[2]/scale[self.accel_range]

    def get_accel_irq(self):
        '''
        For use in interrupt handlers. Sets self._accel._ivector[]
to signed
        unscaled integer accelerometer values
        '''
        self._read(self.buf6, 0x3B, self.mpu_addr)
        self._accel._ivector[0] = bytes_toint(self.buf6[0],
self.buf6[1])
        self._accel._ivector[1] = bytes_toint(self.buf6[2],
self.buf6[3])
        self._accel._ivector[2] = bytes_toint(self.buf6[4],
self.buf6[5])

    # Gyro
```

```
    @property
    def gyro(self):
        '''
        Gyroscope object
        '''
        return self._gyro

    def _gyro_callback(self):
        '''
        Update gyroscope Vector3d object
        '''
        try:
            self._read(self.buf6, 0x43, self.mpu_addr)
        except OSError:
            raise MPUException(self._I2Cerror)
        self._gyro._ivector[0] = bytes_toint(self.buf6[0],
self.buf6[1])
        self._gyro._ivector[1] = bytes_toint(self.buf6[2],
self.buf6[3])
        self._gyro._ivector[2] = bytes_toint(self.buf6[4],
self.buf6[5])
        scale = (131, 65.5, 32.8, 16.4)
        self._gyro._vector[0] =
self._gyro._ivector[0]/scale[self.gyro_range]
        self._gyro._vector[1] =
self._gyro._ivector[1]/scale[self.gyro_range]
        self._gyro._vector[2] =
self._gyro._ivector[2]/scale[self.gyro_range]

    def get_gyro_irq(self):
        '''
        For use in interrupt handlers. Sets self._gyro._ivector[] to
signed
        unscaled integer gyro values. Error trapping disallowed.
        '''
        self._read(self.buf6, 0x43, self.mpu_addr)
        self._gyro._ivector[0] = bytes_toint(self.buf6[0],
self.buf6[1])
        self._gyro._ivector[1] = bytes_toint(self.buf6[2],
self.buf6[3])
        self._gyro._ivector[2] = bytes_toint(self.buf6[4],
self.buf6[5])
```

1.4. vector3d.py

```
import pyb
from math import sqrt, degrees, acos, atan2


def default_wait():
    pyb.delay(50)


class Vector3d(object):
    '''
    Represents a vector in a 3D space using Cartesian coordinates.
    Internally uses sensor relative coordinates.
    Returns vehicle-relative x, y and z values.
    '''
    def __init__(self, transposition, scaling, update_function):
        self._vector = [0, 0, 0]
        self._ivector = [0, 0, 0]
        self.cal = (0, 0, 0)
        self.argcheck(transposition, "Transposition")
        self.argcheck(scaling, "Scaling")
        if set(transposition) != {0, 1, 2}:
            raise ValueError('Transpose indices must be unique and
in range 0-2')
        self._scale = scaling
        self._transpose = transposition
        self.update = update_function

    def argcheck(self, arg, name):
        '''
        checks if arguments are of correct length
        '''
        if len(arg) != 3 or not (type(arg) is list or type(arg) is
tuple):
            raise ValueError(name + ' must be a 3 element list or
tuple')

    def calibrate(self, stopfunc, waitfunc=default_wait):
        '''
        calibration routine, sets cal
        '''
        self.update()
```

```
        maxvec = self._vector[:]                # Initialise max and
min lists with current values
        minvec = self._vector[:]
        while not stopfunc():
            waitfunc()
            self.update()
            maxvec = list(map(max, maxvec, self._vector))
            minvec = list(map(min, minvec, self._vector))
        self.cal = tuple(map(lambda a, b: (a + b)/2, maxvec,
minvec))

    @property
    def _calvector(self):
        '''
        Vector adjusted for calibration offsets
        '''
        return list(map(lambda val, offset: val - offset,
self._vector, self.cal))

    @property
    def x(self):                                # Corrected, vehicle
relative floating point values
        self.update()
        return self._calvector[self._transpose[0]] * self._scale[0]

    @property
    def y(self):
        self.update()
        return self._calvector[self._transpose[1]] * self._scale[1]

    @property
    def z(self):
        self.update()
        return self._calvector[self._transpose[2]] * self._scale[2]

    @property
    def xyz(self):
        self.update()
        return (self._calvector[self._transpose[0]] *
self._scale[0],
                self._calvector[self._transpose[1]] *
self._scale[1],
                self._calvector[self._transpose[2]] *
self._scale[2])

    @property
```

```
    def magnitude(self):
        x, y, z = self.xyz  # All measurements must correspond to
the same instant
        return sqrt(x**2 + y**2 + z**2)

    @property
    def inclination(self):
        x, y, z = self.xyz
        return degrees(acos(z / sqrt(x**2 + y**2 + z**2)))

    @property
    def elevation(self):
        return 90 - self.inclination

    @property
    def azimuth(self):
        x, y, z = self.xyz
        return degrees(atan2(y, x))

    # Raw uncorrected integer values from sensor
    @property
    def ix(self):
        return self._ivector[0]

    @property
    def iy(self):
        return self._ivector[1]

    @property
    def iz(self):
        return self._ivector[2]

    @property
    def ixyz(self):
        return self._ivector

    @property
    def transpose(self):
        return tuple(self._transpose)

    @property
    def scale(self):
        return tuple(self._scale)
```

2. ANEXO B: Código de la Raspberry Pi

2.1. control.py

```
import serial as ps
import time

class control:
    def __init__(self):
        try:
            self.uart=ps.Serial("/dev/ttyUSB0",baudrate=9600,
timeout=5)
        except:
            self.uart=ps.Serial("/dev/ttyUSB1",baudrate=9600,
timeout=5)
        self.cua=''
        self.resultat=[0,0]
        self.dades=[0,0,0,0,0,0]

    def comunica(self):
        #self.dades=[accX,accY,accZ,roll,pitch,yaw]

        self.llegirUART()
        report=self.desxifra(self.desentrama())
        self.escriureUART(report)
        return self.dades


    def llegirUART(self):
        while self.uart.inWaiting():
            char=self.uart.read()
            self.cua+=char

    def escriureUART(self,data):
        self.uart.write(data)
#-----
    def desentrama(self):
        trames=[]

        if len(self.cua)>0:
            while self.cua[0]!='#':
                self.cua=self.cua[1:]
                if len(self.cua)==0:
                    return []
            cont=0
```

```
            final=False
            while final==False:
                if cont==len(self.cua):
                    final=True
                else:
                    if self.cua[cont]=='%':
                        trames.append(self.cua[0:cont+1])
                        self.cua=self.cua[cont+1:]
                        cont=0
                    else:
                        cont+=1
            return trames
        else:
            return []

    def desxifra(self,trames):
        ok=0
        err=0
        for trama in trames:
            mode=trama[1]
            try:
                cksum=int(trama[21:23])
                suma=0
                #-- cksm comprovacio
                for i in (trama[3:8]+trama[9:14]+trama[15:20]):
                    suma+=int(i)
                if suma%100==cksum:
                    ok+=1
                else:
                    errorexpresament
                #---
                if mode=='A':
                    if trama[2]=='+':
                        signe1=1
                    else:
                        signe1=-1
                    if trama[8]=='+':
                        signe2=1
                    else:
                        signe2=-1
                    if trama[14]=='+':
                        signe3=1
                    else:
```

```
                        signe3=-1

                    self.dades[0]=(int(trama[3])+int
(trama[4:8])/10000.)*signe1
                    self.dades[1]=(int(trama[9])+int
(trama[10:14])/10000.)*signe2
                    self.dades[2]=(int(trama[15])+int
(trama[16:20])/10000.)*signe3

                elif mode=='G':
                    if trama[2]=='+':
                        signe1=1
                    else:
                        signe1=-1
                    if trama[8]=='+':
                        signe2=1
                    else:
                        signe2=-1
                    if trama[14]=='+':
                        signe3=1
                    else:
                        signe3=-1

                    self.dades[3]=(int(trama[3:6])+int
(trama[6:8])/100.)*signe1
                    self.dades[4]=(int(trama[9:12])+int
(trama[12:14])/100.)*signe2
                    self.dades[5]=(int(trama[15:18])+int
(trama[18:20])/100.)*signe3
            except:
                err+=1
        report=str(ok)+'xxx'+str(err)+' '
        self.resultat[0]+=ok
        self.resultat[1]+=err
        return report
#------------
    def repeteix(self):
        while True:
            ara=self.comunica()
            print ara
            time.sleep(0.02)
```

3. ANEXO C: Interfaz gráfica

3.1. grafica.py

```
import pygame
import accel, giro, compas, control

def main():
    #-----
    pygame.init()
    midax=1300
    miday=700
    mides=(midax,miday)
    Fons=pygame.display.set_mode(mides)
    colorfons=(238,232,170)
    #-----
    #inicialitza classes
    accelx=accel.accel((670,290),(200,400),'eix
X',colorfons,(255,0,0))
    accely=accel.accel((880,290),(200,400),'eix
Y',colorfons,(0,255,0))
    accelz=accel.accel((1090,290),(200,400),'eix
Z',colorfons,(0,0,255))
    giroscopi=giro.giro((10,50),600,colorfons)
    comp=compas.compas((800,10),280,colorfons)
    #classe control
    cervell=control.control()
    #-----
    classes=pygame.sprite.OrderedUpdates()
    classes.add(accelx,accely,accelz,giroscopi,comp)

    crono=pygame.time.Clock()
    final=False
    while not final:
        for event in pygame.event.get():
            if event.type==pygame.QUIT:
                final=True
    #-----
        dades=cervell.comunica()
    #-----
        accelx.actualitza(dades[0])
        accely.actualitza(dades[1])
```

```
        accelz.actualitza(dades[2])
        #control del pitch
        if dades[4]>360:
            dades[4]-=360
        if dades[4]<-360:
            dades[4]+=360

        increm=dades[4]-giroscopi.pitch
        giroscopi.actualitza(dades[3],increm)
        comp.actualitza(dades[5])

    #-----
        classes.update()
        Fons.fill(colorfons)
        classes.draw(Fons)
        pygame.display.flip()
        crono.tick(24)
    pygame.quit()

main()
```

3.2. accel.py

```
#coding:utf-8
import pygame
from pygame.locals import *

class accel(pygame.sprite.Sprite):
    def __init__(self,pos,rec,text,color,colorbarra):
        pygame.sprite.Sprite.__init__(self)
        pygame.font.init()
        self.rec=rec
        self.espaillegenda=rec[0]*0.2
        self.espaititol=rec[1]*0.05
        self.dim=(self.rec[0]-self.espaillegenda,self.rec[1]-
self.espaititol)
        self.marge=self.dim[0]*0.05
        self.text=text
        self.valor=0
        self.color=color
        self.colorbarra=colorbarra

        self.fondo=self.creafondo()
        self.image=self.creafondo()
        self.rect=self.image.get_rect(left=pos[0],top=pos[1])
```

```
    def creafondo(self):
        fondo=pygame.Surface(self.rec)
        fondo.fill(self.color)
        #-----
        cuadrat=self.crearecuadre()
        self.poscuadrat=(0+2,self.espaititol-2)
        lleg=self.crealleg()
        poslleg=(self.dim[0],self.espaititol)
        text=self.creatext()
        postext=(0,0)
        #-----
        fondo.blit(cuadrat,self.poscuadrat)
        fondo.blit(text,postext)
        fondo.blit(lleg,poslleg)
        #-----
        pygame.draw.line(fondo,(50,50,50),(0,0),(0,self.rec[1]),3)
        pygame.draw.line(fondo,(50,50,50),(0,0),(self.rec[0],0),3)
        pygame.draw.line(fondo,(50,50,50),(self.rec[0],0),
(self.rec[0],self.rec[1]),5)
        pygame.draw.line(fondo,(50,50,50),(0,self.rec[1]),
(self.rec[0],self.rec[1]),5)

        #-----
        return fondo

    def creatext(self):

        fondo=pygame.Surface((self.rec[0],self.espaititol))
        fondo.fill(self.color)
        #-----
        lletra=pygame.font.Font('freesansbold.ttf',1)
        text=lletra.render(self.text,1,(0,0,0))
        text_mida=text.get_size()
        i=2
        while text_mida[0]<self.rec[0] and text_mida[1]<
self.espaititol:
            i=i+1
            lletra=pygame.font.Font('freesansbold.ttf',i)
            text=lletra.render(self.text,1,(0,0,0))
            text_mida=text.get_size()

        lletra=pygame.font.Font('freesansbold.ttf',i-1)
```

```
        text=lletra.render(self.text,1,(0,0,0))
        text_mida=text.get_size()
        fondo.blit(text,((self.rec[0]-text_mida[0])/2,
(self.espaititol-text_mida[1])/2))
        return fondo

    def crealleg(self):
        lletra=pygame.font.Font('freesansbold.ttf',15)
        fondo=pygame.Surface((self.espaillegenda,self.dim[1]))
        fondo.fill(self.color)

        increm=self.dim[1]/8.

        num=lletra.render('0g',1,(0,0,0))
        num_mida=num.get_size()
        fondo.blit(num,((self.espaillegenda-num_mida[0])/2,
(self.dim[1]-num_mida[1])/2))
        for i in range(1,5):
            num1=lletra.render(str(i)+'g',1,(0,0,0))
            num_mida1=num.get_size()
            num2=lletra.render(str(-i)+'g',1,(0,0,0))
            num_mida2=num.get_size()
            fondo.blit(num1,((self.espaillegenda-num_mida1[0])
/2,self.dim[1]/2-i*increm))
            fondo.blit(num2,((self.espaillegenda-num_mida2[0])
/2,self.dim[1]/2-num_mida2[1]+i*increm-3))
        return fondo

    def crearecuadre(self):
        fondo=pygame.Surface(self.dim)
        fondo.fill((0,0,0))
        #-----
        pygame.draw.line(fondo,(200,200,200),(self.marge/4-1,0),
(self.marge/4-1,self.dim[1]),int(self.marge/2))
        pygame.draw.line(fondo,(200,200,200),(self.dim[0]-self.marge
/4-1,0),(self.dim[0]-self.marge/4-1,self.dim[1]),int(self.marge/2))
        pygame.draw.line(fondo,(200,200,200),(0,self.marge/4-
1),(self.dim[0],self.marge/4-1),int(self.marge/2))
        pygame.draw.line(fondo,(200,200,200),(0,self.dim[1]-
self.marge/4-1),(self.dim[0],self.dim[1]-self.marge/4-
1),int(self.marge/2))
        #-----
        pygame.draw.line(fondo,(150,150,150),(self.marge*0.75-
1,self.marge/2),(self.marge*0.75-1,self.dim[1]-self.marge/2-
1),int(self.marge/2))
        pygame.draw.line(fondo,(150,150,150),(self.dim[0]-
```

```
self.marge*0.75-1,self.marge/2),(self.dim[0]-self.marge*0.75-
1,self.dim[1]-self.marge/2-1),int(self.marge/2))
        pygame.draw.line(fondo,(150,150,150),(self.marge/2,
self.marge/2) ,(self.dim[0]-self.marge/2-1, self.marge
/2) ,int(self.marge/2))
        pygame.draw.line(fondo, (150,150,150), (self.marge*0.75,
self.dim[1] -self.marge*0.75-1) ,(self.dim[0]-self.marge*0.75 ,
self.dim[1 ]-self.marge*0.75-1),int(self.marge/2))
        #-----
        pygame.draw.line(fondo,(150,150,150), (self.marge/2,
self.dim[1] /2+self.marge/4), (self.dim[0]-self.marge/2-1,
self.dim[1] /2+self.marge/4),int(self.marge/2))
        pygame.draw.line(fondo, (150,150,150), (self.marge
/2 ,self.dim[1] /2-self.marge/4-1),(self.dim[0]-self.marge/2-
1,self.dim[1]/2-self.marge/4-1),int(self.marge/2))
        pygame.draw.line(fondo, (200,200,200) , (0,self.dim[1]
/2) ,(self.dim[0],self.dim[1]/2),int(self.marge/4))
        return fondo

    def barres(self,valor):
        color=(10,10,10)
        valor1=abs(valor)
        mides=(self.dim[0]-self.marge*2,self.dim[1]/2-self.marge
*1.3+1)
        increm=mides[1]/4
        base=(self.dim[0]-self.marge)*0.75
        #-----
        barra1=pygame.Surface(mides)
        barra2=pygame.Surface(mides)
        barra1.fill(color)
        barra2.fill(color)
        pygame.draw.line(barra1,self.colorbarra,(mides[0]/2,
mides[1]),(mides[0]/2,mides[1]-increm*valor1),int(base))
        #----
        for i in (barra1,barra2):
            for j in range(1,4):
                pygame.draw.line(i,(250,250,250),(0,mides[1]-
j*increm),(mides[0],mides[1]-j*increm),3)
        #-----
        if valor<0:
            barra1=pygame.transform.rotate(barra1,180)
            return (barra2,barra1)
        else:
```

```
            return (barra1,barra2)

    def actualitza(self,valor):
        self.valor=valor

    def update(self):
        valor=self.valor
        cuadrat=self.crearecuadre()
        barres=self.barres(valor)
        cuadrat.blit(barres[0],(self.marge,self.marge-2))
        cuadrat.blit(barres[1],(self.marge,self.marge*1.9+
barres[0].get_size()[1]))

        fondo=self.image
        fondo.blit(cuadrat,self.poscuadrat)
        self.image=fondo
```

3.3. giro.py

```
#coding:utf-8
import pygame
import math
from pygame.locals import *

class giro(pygame.sprite.Sprite):
    def __init__(self,pos,dim,color):
        pygame.sprite.Sprite.__init__(self)
        pygame.font.init()
        self.transparent=(255,0,255)

        self.roll=0
        self.pitch=0
        self.dim=dim
        self.pos=pos
        self.color=color
        self.radi=(0.5*self.dim)-5
        self.gruix=self.dim/150

        self.T=self.marcador()
        self.fondo=self.creafondo()
        self.sky=self.creasky()
        self.lletra=pygame.font.Font('freesansbold.ttf',20)
```

```
        self.image=pygame.Surface((self.dim,self.dim))
        self.rect=self.image.get_rect(left=pos[0],top=pos[1])

    def creafondo(self):
        fondo=pygame.Surface((self.dim,self.dim))
        fondo.fill(self.color)
        return fondo

    def creasky(self):
        lletra=pygame.font.Font('freesansbold.ttf',15)
        gruix=self.gruix
        radi=self.radi
        dim=0.5*self.dim
        x=self.dim
        y=720/35.*self.radi
        increm=y/48.
        cel=pygame.Surface((x,y))
        pygame.draw.rect(cel,(50,50,255),((0,0),(x,y)))
        pygame.draw.line(cel,(0,0,0),(0,0.25*y),(x,0.25*y))
        pygame.draw.rect(cel,(160,82,45),((0,0),(x,0.25*y)))
        pygame.draw.rect(cel,(160,82,45),((0,0.5*y),(x,0.75*y)))
        pygame.draw.rect(cel,(50,50,255),((0,0.75*y),(x,y)))
        pygame.draw.line(cel,(0,0,0),(0,0.5*y),(x,0.5*y))
        pygame.draw.line(cel,(0,0,0),(0,0.25*y),(x,0.25*y))
        pygame.draw.line(cel,(0,0,0),(0,0.75*y),(x,0.75*y))
        color=(255,255,255)

        for i in range(25):
            pos11=([dim-0.3*radi-0.2*radi,0.5*y-i*increm],[dim-
0.3*radi,0.5*y-i*increm])
            pos12=([dim-0.3*radi,0.5*y-i*increm-gruix*2],[dim-
0.3*radi,0.5*y-i*increm+gruix*2])
            pos21=([dim+0.3*radi+0.2*radi,0.5*y-
i*increm],[dim+0.3*radi,0.5*y-i*increm])
            pos22=([dim+0.3*radi,0.5*y-i*increm-
gruix*2],[dim+0.3*radi,0.5*y-i*increm+gruix*2])

            num=lletra.render(str(i*15),1,color)
            num_mida=num.get_size()
            posn1=(pos11[0][0]-1.2*num_mida[0],pos11[0][1]-
num_mida[1]*0.5)
            posn2=(pos21[0][0]+0.2*num_mida[0],pos21[0][1]-
num_mida[1]*0.5)
```

```
            cel.blit(num,posn1)
            cel.blit(num,posn2)
            #dibuixar numeros
            pos=(pos11,pos12,pos21,pos22)
            #-----
            for i in pos:
                pygame.draw.line(cel,color,i[0],i[1],gruix)
        for i in range(24):
            pos11=([dim-0.3*radi-0.2*radi,y-i*increm],[dim-
0.3*radi,y-i*increm])
            pos12=([dim-0.3*radi,y-i*increm-gruix*2],[dim-
0.3*radi,y-i*increm+gruix*2])
            pos21=([dim+0.3*radi+0.2*radi,y-
i*increm],[dim+0.3*radi,y-i*increm])
            pos22=([dim+0.3*radi,y-i*increm-
gruix*2],[dim+0.3*radi,y-i*increm+gruix*2])

            num=lletra.render(str(i*15),1,color)
            num_mida=num.get_size()
            posn1=(pos11[0][0]-1.2*num_mida[0],pos11[0][1]-
num_mida[1]*0.5)
            posn2=(pos21[0][0]+0.2*num_mida[0],pos21[0][1]-
num_mida[1]*0.5)
            cel.blit(num,posn1)
            cel.blit(num,posn2)
            pos=(pos11,pos12,pos21,pos22)
            #-----
            for i in pos:
                pygame.draw.line(cel,color,i[0],i[1],gruix)
        return cel

    def marcador(self):
        color=(0,200,0)
        creueta=pygame.Surface((2*self.radi,2*self.radi))
        creueta.set_colorkey(self.transparent)
        creueta.fill(self.transparent)
        gruix=self.gruix*2
        #-----
        pygame.draw.line(creueta,color, (0.2*
self.radi ,self.radi) ,(0.8*self.radi,self.radi),gruix)
        pygame.draw.line(creueta,color, (0.2*
self.radi ,0.9*self.radi) ,(0.2*self.radi,self.radi+0.5*gruix),gruix
)
        pygame.draw.line(creueta,color,(0.8*self.radi,self.radi-
0.5*gruix+1),(0.8*self.radi,1.15*self.radi),gruix)
        #------
```

```
         pygame.draw.line(creueta,color, (1.2*
self.radi ,self.radi) ,(1.8*self.radi,self.radi),gruix)
        pygame.draw.line(creueta,color, (1.8*self.radi,
0.9*self.radi) ,(1.8*self.radi,self.radi+0.5*gruix),gruix)
        pygame.draw.line(creueta,color,(1.2*self.radi,self.radi-
0.5*gruix+1),(1.2*self.radi,1.15*self.radi),gruix)
        #-----
        pygame.draw.line(creueta,color,
(self.radi,1.1*self.radi) ,(self.radi,1.8*self.radi),gruix)
        #-----

pygame.draw.polygon(creueta,color,((self.radi,0.2*self.radi) ,(0.975
*self.radi,0.3*self.radi),(1.03*self.radi,0.3*self.radi)),0)
        #-----
        return creueta

    def anglesroll(self):

        color=(255,255,255)
        fondo=pygame.Surface((2*self.radi,2*self.radi),
flags=pygame.SRCALPHA)
        fondo.set_colorkey((0,0,0))
        fondo.fill((0,0,0,0))
        gruix=self.gruix
        lletra=pygame.font.Font('freesansbold.ttf',15)

        for i in range(13):
            angle=i*15
            posicions=(0.8*self.radi,0.82*self.radi,0.93*self.radi,
0.95*self.radi)
            for j in range(2):
                for k in [angle,-angle]:
                    pygame.draw.line(fondo,(255,255,255), (posicions
[2*j] *math.sin(math.radians(k)) +self.radi,posicions[2*j]* math.cos
(math.radians(k))+self.radi),(posicions[2*j+1]*math.sin(math.radians
(k))+self.radi,posicions[2*j+1]*math.cos(math.radians(k))+self.radi)
,gruix*4/3)

                    num=lletra.render(str(abs(angle)),0,color)
                    num=pygame.transform.rotate(num,k)
                    num_mida=num.get_size()
                    fondo.blit(num,(0.88*self.radi*math.sin
(math.radians(180+k))+self.radi  -num_mida[0]/2          ,0.88*
```

```
self.radi * math.cos(math.radians (180+k)) +self.radi -
num_mida[1]/2            ))

        return fondo

    def actualitza(self,roll,pitch):
        self.roll=roll
        self.pitch+=pitch
        if self.pitch>325:
            desfase=325-self.pitch
            self.pitch=-35-desfase
        if self.pitch<-325:
            desfase=-325-self.pitch
            self.pitch=35-desfase

    def update(self):
        roll=self.roll
        pitch=self.pitch
        increm=self.sky.get_size()[1]/720.
        ara=self.sky.get_size()[1]/2-increm*pitch

        nova=pygame.Surface((self.dim,self.dim))
        nova.set_colorkey(self.transparent)

        nova.blit(self.sky,(0,0),(0,ara-self.radi-
5,self.dim ,ara+self.dim-self.radi-5))
        #-----
        cercle=pygame.Surface((self.dim,self.dim))
        cercle.fill(self.color)
        cercle.set_colorkey(self.transparent)
        pygame.draw.circle(cercle,self.transparent,(self.dim/2,
self.dim/2),int(self.radi)+1,0)
        pygame.draw.circle(cercle,(20,20,20),(self.dim/2,
self.dim/2),int(self.radi)+2,3)
        nova.blit(cercle,(0,0))
        nova.blit(self.anglesroll(),((self.dim-
2*self.radi)/2 ,(self.dim-2*self.radi)/2))
        #-----
        nova=pygame.transform.rotate(nova,roll)
        nova1=pygame.Surface((self.dim,self.dim))
        nova1.fill(self.color)
        nova1.set_colorkey(self.transparent)
        marge=(nova.get_size()[0]-self.dim)/2
        nova1.blit(nova,(0,0),(marge,marge,marge+self.dim,
marge+self.dim))
        #-----
```

```
        posT=((self.dim-2*self.radi)/2,(self.dim-2*self.radi)/2)
        nova1.blit(self.T,posT)
        #-----
        self.image=nova1
```

3.4. compass.py

```
#coding:utf-8
import math
import pygame
from pygame.locals import *

class compas(pygame.sprite.Sprite):
    def __init__(self,pos,dim,colorfons):
        pygame.sprite.Sprite.__init__(self)
        pygame.font.init()

        self.dim=dim
        self.yaw=0
        self.colorfons=colorfons
        self.radi=(0.5*self.dim)-5
        self.gruix=self.dim/150
        self.transparent=(255,0,255)

        self.T=self.marcador()
        self.fondo=self.creafondo()
        self.image=self.creafondo()
        self.rect=self.image.get_rect(left=pos[0],top=pos[1])

    def creafondo(self):
        fondo=pygame.Surface((self.dim,self.dim))
        fondo.fill((30,30,30))
        fondo.set_colorkey(self.transparent)
        color=pygame.Surface((self.dim,self.dim))
        color.set_colorkey(self.transparent)
        color.fill(self.colorfons)
        pygame.draw.circle(color,self.transparent,
(self.dim/2 ,self.dim/2),int(self.radi),0)
        angles=self.anglesyaw()
        fondo.blit(color,(0,0))
        fondo.blit(angles,((self.dim-2*self.radi)/2,(self.dim-
2*self.radi)/2))
```

```
        return fondo

    def anglesyaw(self):
        color=(255,255,255)
        fondo=pygame.Surface((2*self.radi,2*self.radi) ,flags
=pygame.SRCALPHA)
        fondo.fill((0,0,0,0))
        fondo.set_colorkey(self.transparent)
        gruix=self.gruix
        lletra=pygame.font.Font('freesansbold.ttf',12)

        for i in range(13):
            angle=i*15
            posicions=(0.55*self.radi,0.72*self.radi ,0.87*self.
radi,0.95*self.radi)
            for j in range(2):
                for k in [angle,-angle]:
                    pygame.draw.line(fondo,(255,255,255), (posicions
[2*j]* math.sin(math.radians(k))+self.radi ,posicions [2*j]
*math.cos (math.radians(k))+self.radi), (posicions [2*j+1] *math.sin
(math.radians(k))+ self.radi, posicions [2*j+1] *math.cos
(math.radians(k))+ self.radi),gruix*4/3)

                    num=lletra.render(str(abs(angle)),0,color)
                    num=pygame.transform.rotate(num,k)
                    num_mida=num.get_size()
                    fondo.blit(num,(0.8*self.radi*math.sin
(math.radians(180+k))+self.radi  -
num_mida[0]/2          ,0.8*self.radi *math.cos
(math.radians(180+k)) +self.radi -num_mida[1]/2           ))

        return fondo

    def marcador(self):
        color=(0,255,0)
        creueta=pygame.Surface((2*self.radi,2*self.radi))
        creueta.set_colorkey(self.transparent)
        creueta.fill(self.transparent)

pygame.draw.polygon(creueta,color,( (self.radi,0.5*self.radi),
(0.8*self.radi,1.2*self.radi), (1.2*self.radi,1.2*self.radi)),0)
        return creueta

    def actualitza(self,valor):
        self.yaw=valor
```

```
    def update(self):
        yaw=self.yaw
        nova=self.fondo

        nova=pygame.transform.rotate(nova,yaw)
        marge=(nova.get_size()[0]-self.dim)/2

        nova1=pygame.Surface((self.dim,self.dim))
        nova1.set_colorkey(self.transparent)
        nova1.blit(nova,(0,0),(marge ,marge,marge+self.dim,
marge+self.dim))

        posT=((self.dim-2*self.radi)/2,(self.dim-2*self.radi)/2)
        nova1.blit(self.T,posT)
        cercle=pygame.Surface((self.dim,self.dim))
        cercle.fill(self.colorfons)
        cercle.set_colorkey(self.transparent)
        pygame.draw.circle(cercle,self.transparent,
(self.dim/2 ,self.dim/2), int(self.radi),0)
        nova1.blit(cercle,(0,0))

        self.image=nova1
```