[ANEXO 1]
[Código del Módulo de Configuración y Calibración]

```
import os
import cv2
import scipy.io
import numpy as np
from configobj import ConfigObj


#######CLASE DE CONFIGURACION ######

class configuracion():

        def __init__(self):
                        self.pares = []
                        self.centrosMasa = []
                        self.puntosInteres = []
                        self.descriptoresInteres = []
                        self.banderaConfig = True
                        self.configHecho = False

        def inicio(self):
                self.pares = []
                self.centrosMasa = []
                self.puntosInteres = []
                self.descriptoresInteres = []
                areaMinimaPatron = 40000
                areaMaximaPatron = 60000
                areaMinima = 300
                indicePatron = 0
                indiceContorno = 0
                regionS = 40

                font = cv2.FONT_HERSHEY_SIMPLEX
                detector = cv2.ORB(500,2,8)
                configura = ConfigObj()

                # LECTURA DE MARCADOR PATRON
                img = cv2.imread('circulo.jpg',0)
                ret,umbral = cv2.threshold(img,umbralF,255,cv2.THRESH_BINARY)
                (contornoPatron,_)                                                              =
cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

                for indice in xrange(len(contornoPatron)):
                        area = cv2.contourArea(contornoPatron[indice])
                        if area > areaMinimaPatron and area < areaMaximaPatron:
                                indicePatron = indice

                # TOMA DE IMAGEN DE REFERENCIA
                cap = cv2.VideoCapture(0)

                # TIEMPO DE ADAPTACION DE CAMARA
                for indice in xrange(20):
                        ret,imagen = cap.read()

                if self.banderaConfig:
```

```
                        while(True):
                            # Captura cuadro
                            ret,imagen = cap.read()

                            # Guardar cuadro
                            if cv2.waitKey(1) & 0xFF == ord('q'):
                                    cv2.imwrite('contorno.jpg', imagen)
                                    break

                            #Mostrar Cuadro
                            cv2.imshow('frame',imagen)

                    else:
                        for indice in xrange(15):
                                ret,imagen = cap.read()

                        ret,imagen = cap.read()
                        cv2.imwrite('contorno.jpg', imagen)


                    # CERRAR CAMARA Y VENTANA
                    cap.release()
                    cv2.destroyAllWindows()
                    cv2.waitKey(1)
                    cv2.waitKey(1)
                    cv2.waitKey(1)
                    cv2.waitKey(1)

                    # PROCESAMIENTO DE IMAGEN DE REFERENCIA
                    dst = cv2.imread('contorno.jpg',1)

                    img = cv2.undistort(dst, distorsion['matrizoriginal'], distorsion['coeficientes'], None,
distorsion['matriznueva'])
                    x,y,w,h = distorsion['region']
                    img = img[y:y+h, x:x+w]

                    cv2.imwrite('contornoUD.jpg', img)
                    img = cv2.imread('contornoUD.jpg',0)

                    ret,umbral = cv2.threshold(img,umbralF,255,cv2.THRESH_BINARY)
                    (contorno,_) = cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
                    img = cv2.imread('contornoUD.jpg',1)

                    # IDENTIFICACION DE MARCADORES EN IMAGEN
                    for indice in xrange(len(contorno)):
                        area = cv2.contourArea(contorno[indice])
                        if area > areaMinima:
                                ret                                                                    =
cv2.matchShapes(contorno[indice],contornoPatron[indicePatron],1,0.0)
                                if ret < umbralSimil:
                                        momentos = cv2.moments(contorno[indice])
                                        cx = (momentos['m10']/momentos['m00'])
                                        cy = (momentos['m01']/momentos['m00'])
                                        self.centrosMasa.append((cx,cy,indice))


                    # ORDENAMIENTO SEGUN COORDENADA X
                    for indice in xrange(len(self.centrosMasa)):
```

```
                        for subindice in xrange (indice,len(self.centrosMasa)):
                                if self.centrosMasa[indice][0] > self.centrosMasa[subindice][0]:
                                        temp,temp2,temp3 = self.centrosMasa[indice]
                                        self.centrosMasa[indice] = self.centrosMasa[subindice]
                                        self.centrosMasa[subindice] = temp,temp2,temp3

                # ELIMINACION DE REPETICIONES Y ORDENAMIENTO SEGUN COORDENADA Y
                for indice in xrange(len(self.centrosMasa)):
                        try:
                                if abs(self.centrosMasa[indice][0] - self.centrosMasa[indice+1][0]) <
3:
                                        if abs(self.centrosMasa[indice][1] -
self.centrosMasa[indice+1][1]) < 3:
                                                self.centrosMasa.remove(self.centrosMasa[indice])
                                        elif self.centrosMasa[indice][1] >
self.centrosMasa[indice+1][1]:
                                                temp,temp2,temp3 = self.centrosMasa[indice]
                                                self.centrosMasa[indice] =
self.centrosMasa[indice+1]
                                                self.centrosMasa[indice+1] = temp,temp2,temp3
                        except IndexError:
                                pass

                for indice in xrange(len(self.centrosMasa)):
                        q = self.centrosMasa[indice][0]
                        w = self.centrosMasa[indice][1]
                        e = self.centrosMasa[indice][2]
                        preset.archivoConfiguracion['centroMasa_%i' %(indice)] = [q,w,e]

                for indice in xrange(len(self.centrosMasa)):
                        x,y,w,h = cv2.boundingRect(contorno[self.centrosMasa[indice][2]])
                        if indice == 4:
                                regionS = 10
                        preset.archivoConfiguracion['region_%i' %(indice)] = [x-regionS,y-
regionS,w+(regionS*2),h+(regionS*2),indice]

                preset.archivoConfiguracion['indiceRegiones'] = len(self.centrosMasa)

                if self.banderaConfig:

                        # RESALTAR EN FOTO MARCADORES RECONOCIDOS
                        for indice in xrange(len(self.centrosMasa)):

                                cv2.drawContours(img, contorno, self.centrosMasa[indice][2],
(0,0,255), 1)

        cv2.putText(img,'%i' %(indiceContorno),(int(self.centrosMasa[indice][0]),int(self.centrosMasa[in
dice][1])), font, 1,(0,0,255),1)
                                indiceContorno += 1

                        # MOSTRAR FOTO CON MARCADORES
                        cv2.imwrite('leyenda.jpg',img)
                        cv2.imshow('Contorno',img)
                        cv2.waitKey(500)

                        print "Ingrese los pares ordenados para el calculo de las distancias"
                        print "Ingrese 'fin' para finalizar"
                        indice = 0
```

```
            par = input("ej: (A,B):")
            preset.archivoConfiguracion['par_%i' %(indice)] = par

            while True:
                par = input("Par:")
                if par == 'fin':
                    preset.archivoConfiguracion['indicePares'] = indice + 1
                    break
                else:
                    indice += 1
                    preset.archivoConfiguracion['par_%i' %(indice)] = par

            cap.release()
            cv2.destroyAllWindows()
            cv2.waitKey(1)
            cv2.waitKey(1)
            cv2.waitKey(1)
            cv2.waitKey(1)

    else:
            self.pares = configPrevia['pares']
            vector = self.pares
            for x in xrange(len(vector)):
                vector[x] = eval(vector[x])

    img = cv2.imread('contorno.jpg',1)

    print "Ingrese numero de mediciones para devolver"
    variable = input("")
    preset.archivoConfiguracion['numeroMediciones'] = variable

    print "Ingrese tamano de banco de registro (numero de fotografias)"
    variable = input("")
    preset.archivoConfiguracion['bancoImagenes'] = variable + 1

    print "Ingrese aumento maximo admisible de las distancias"
    variable = input("")
    preset.archivoConfiguracion['distanciaDeriva'] = variable

    print "Ingrese numero de detecciones luego del cual debe reportarse deriva permanente"
    variable = input("")
    preset.archivoConfiguracion['deteccionesDeriva'] = variable

    print "Ingrese string para nombrar fotografias tomadas bajo demanda"
    variable = input("")
    preset.archivoConfiguracion['fotoUsuario'] = variable

    indice = int(preset.archivoConfiguracion['indicePares'])

    for i in xrange(indice):
            par = preset.archivoConfiguracion['par_%i' %(i)]

            #for j in xrange(2):
                #par[j] = int(par[j])

            self.pares.append(par)

    for indiceP in xrange(indice):
```

```
                        a = self.pares[indiceP][0]
                        b = self.pares[indiceP][1]

                        DX = self.centrosMasa[a][0] - self.centrosMasa[b][0]
                        if DX < 0:
                                DX = - DX

                        DY = self.centrosMasa[a][1] - self.centrosMasa[b][1]
                        if DY < 0:
                                DY = - DY

                        preset.archivoConfiguracion['referencia_%i'%(indiceP)]=(DX,DY)

                preset.archivoConfiguracion['segmento'] = umbralF
                preset.archivoConfiguracion['umbralSimil'] = umbralSimil

                print '# # # # # # # # # # # # # # # # \n'
                print "    Configuracion Completa    \n"
                print '# # # # # # # # # # # # # # # # \n'

                self.configHecho = True
                preset.archivoConfiguracion['configHecho'] = self.configHecho
                preset.archivoConfiguracion.write()


# # # # # # # # C L A S E  D E  C A L I B R A C I O N # # # # # # #

class calibracion():

        def __init__(self):
                self.banderaCalibracion = True
                self.calibHecho = False

        def calibrar(self):

                dx = 0
                dy = 0
                muestras = 0
                distanciaHamming = 30
                areaMinimaPatron = 40000
                areaMaximaPatron = 60000
                areaMinima = 300
                centrosMasa = []
                regiones = []
                puntosInteres = []
                descriptoresInteres = []
                detector = cv2.ORB(500,2,8)
                pareo = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
                font = cv2.FONT_HERSHEY_SIMPLEX

                # LECTURA DE MARCADOR PATRON
                img = cv2.imread('circulo.jpg',0)
                ret,umbral = cv2.threshold(img,umbralF,255,cv2.THRESH_BINARY)
                (contornoPatron,_)                                                              =
cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

                for indice in xrange(len(contornoPatron)):
                        area = cv2.contourArea(contornoPatron[indice])
```

```
                        if area > areaMinimaPatron and area < areaMaximaPatron:
                                indicePatron = indice

                # TOMAR FOTO Y ENCONTRAR PUNTOS

                print "Pulse q para registrar posicion"

                # TOMA DE IMAGEN DE REFERENCIA
                cap = cv2.VideoCapture(0)

                # TIEMPO DE ADAPTACION DE CAMARA
                for indice in xrange(20):
                        ret,imagen = cap.read()

                #if self.banderaConfig:
                while(True):
                  # Captura cuadro
                  ret,imagen = cap.read()

                  # Guardar cuadro
                  if cv2.waitKey(1) & 0xFF == ord('q'):
                                cv2.imwrite('calibracion01.jpg', imagen)
                                break

                  #Mostrar Cuadro
                  cv2.imshow('frame',imagen)

                # CERRAR CAMARA Y VENTANA
                cap.release()
                cv2.destroyAllWindows()
                cv2.waitKey(1)
                cv2.waitKey(1)
                cv2.waitKey(1)
                cv2.waitKey(1)

                # PROCESAMIENTO DE IMAGEN DE REFERENCIA
                dst = cv2.imread('calibracion01.jpg',1)

                img = cv2.undistort(dst, distorsion['matrizoriginal'], distorsion['coeficientes'], None,
distorsion['matriznueva'])
                x,y,w,h = distorsion['region']
                img = img[y:y+h, x:x+w]

                cv2.imwrite('calibracion01UD.jpg',img)
                img = cv2.imread('calibracion01UD.jpg',0)

                ret,umbral = cv2.threshold(img,umbralF,255,cv2.THRESH_BINARY)
                (contorno,_) = cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
                img = cv2.imread('calibracion01UD.jpg',1)

                # IDENTIFICACION DE MARCADORES EN IMAGEN
                for indice in xrange(len(contorno)):
                        area = cv2.contourArea(contorno[indice])
                        if area > areaMinima:
                                ret                                                                    =
cv2.matchShapes(contorno[indice],contornoPatron[indicePatron],1,0.0)
                                if ret < umbralSimil:
                                        momentos = cv2.moments(contorno[indice])
```

```
                                        cx = (momentos['m10']/momentos['m00'])
                                        cy = (momentos['m01']/momentos['m00'])
                                        centrosMasa.append((cx,cy,indice))
                                        break

                cv2.drawContours(img, contorno, centrosMasa[0][2], (0,0,255), 1)
                cv2.putText(img,'0',(int(centrosMasa[0][0]),int(centrosMasa[0][1])), font, 1,(0,0,255),1)

                # MOSTRAR FOTO CON MARCADORES
                cv2.imshow('Contorno',img)
                cv2.waitKey(0)
                cv2.destroyAllWindows()
                cv2.waitKey(1)
                cv2.waitKey(1)
                cv2.waitKey(1)
                cv2.waitKey(1)

                x,y,w,h = cv2.boundingRect(contorno[centrosMasa[0][2]])
                regiones.append((x,y,w,h,indice))

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

                if self.banderaCalibracion == True:

                        # LECTURA DE MARCADOR PATRON
                        img = cv2.imread('circulo.jpg',0)
                        ret,umbral = cv2.threshold(img,umbralF,255,cv2.THRESH_BINARY)
                        (contornoPatron,_)                                                              =
cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

                        for indice in xrange(len(contornoPatron)):
                                area = cv2.contourArea(contornoPatron[indice])
                                if area > areaMinimaPatron and area < areaMaximaPatron:
                                        indicePatron = indice

                        # PEDIR DESPLAZMAIENTO DE MARCADOR
                        print "Ingrese en cm el desplazamiento del marcador"
                        desplazamiento = input("")

                        # TOMAR FOTO
                        cap = cv2.VideoCapture(0)

                        #if self.banderaConfig:
                        while(True):
                           # Captura cuadro
                           ret,imagen = cap.read()

                           # Guardar cuadro
                           if cv2.waitKey(1) & 0xFF == ord('q'):
                                        cv2.imwrite('calibracion02.jpg', imagen)
                                        break

                           #Mostrar Cuadro
                           cv2.imshow('frame',imagen)

                        # CERRAR CAMARA Y VENTANA
                        cap.release()
                        cv2.destroyAllWindows()
```

```
                        cv2.waitKey(1)
                        cv2.waitKey(1)
                        cv2.waitKey(1)
                        cv2.waitKey(1)

                        # PROCESAMIENTO DE IMAGEN DE REFERENCIA
                        imag = cv2.imread('calibracion02.jpg',0)
                        imag2 = imag[y-25:y+h+50,x-25:x+w+50]
                        ret,umbral = cv2.threshold(imag2,umbralF,255,cv2.THRESH_BINARY)
                        (contorno,_) =
cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
                        img = cv2.imread('calibracion01.jpg',1)

                        # IDENTIFICACION DE MARCADORES EN IMAGEN
                        for indice in xrange(len(contorno)):
                                area = cv2.contourArea(contorno[indice])
                                if area > areaMinima:
                                        ret =
cv2.matchShapes(contorno[indice],contornoPatron[indicePatron],1,0.0)
                                        if ret < umbralSimil:
                                                momentos = cv2.moments(contorno[indice])
                                                cx = (momentos['m10']/momentos['m00']) + (x -
25)
                                                cy = (momentos['m01']/momentos['m00']) + (y -
25)
                                                centrosMasa.append((cx,cy,indice))
                                                break

                        print centrosMasa

                        # CALCULAR FACTOR DE CONVERSION
                        factorConversion = desplazamiento / (centrosMasa[0][0] - centrosMasa[1][0])
                        if factorConversion < 0:
                                factorConversion = - factorConversion

                        deltafactor = factorConversion * ((0.1/desplazamiento)**2 +
(1/(centrosMasa[0][0] - centrosMasa[1][0]))**2)**0.5

                else:

                        # PEDIR TAMANO DE MARCADOR
                        print "Ingrese en cm el diametro del marcador"
                        tamano = input("")

                        factorConversion = (tamano*1.0) / (regiones[0][2]-2)
                        deltafactor = factorConversion * ((0.1/tamano)**2 +
(1/(regiones[0][2])**2))**0.5

                contribdeltafactor = (deltafactor / factorConversion) ** 2

                preset.archivoConfiguracion['contribdeltafactor'] = contribdeltafactor
                preset.archivoConfiguracion['factor'] = factorConversion
                preset.archivoConfiguracion['deltafactor'] = deltafactor

                print '# # # # # # # # # # # # # # # # \n'
                print "     Calibracion Completa     \n"
                print "    factor = %f cm/px "%(factorConversion)
                print '# # # # # # # # # # # # # # # # \n'
```

```
                    self.calibHecho = True
                    preset.archivoConfiguracion['calibHecho'] = self.calibHecho
                    preset.archivoConfiguracion.write()

# # # # # # # # # # C L A S E  D E  A R C H I V O # # # # # # # # #

class archivo():
        def __init__(self):
                self.archivoConfiguracion = ConfigObj('configuracion.ini')

# # # # # # # # P R O G R A M A  P R I N C I P A L # # # # # # # # # #

preset = archivo()
calibracion = calibracion()
configuracion = configuracion()
umbralF = 170
umbralSimil = 0.030000
distorsion = scipy.io.loadmat('distorsion.mat')

try:
        calibracion.calibHecho = preset.archivoConfiguracion['calibHecho']
except KeyError:
        pass

try:
        configuracion.configHecho = preset.archivoConfiguracion['configHecho']
except KeyError:
        pass

if calibracion.calibHecho or configuracion.configHecho:
        print "Ya existe un archivo de configuracion. Desea eliminarlo?"
        eliminar = input("")

        if eliminar == 'si':
                os.remove('configuracion.ini')
                calibracion.calibHecho = False
                configuracion.configHecho = False

print '# # # # # # # # # # # # # # # # # \n'
print '  Calibracion y Configuracion   \n'
print '# # # # # # # # # # # # # # # # # \n'

while True:

        print 'Ingrese una de las opciones:\n'
        print 'Para calibracion.....1'
        print 'Para configuracion.....2'
        print 'Para finalizar.....3\n'
        eleccion = input("")

        if eleccion == 1:

                while True:
                        print 'Ingrese una de las opciones:\n'
                        print 'Para calibrar mediante desplazamientos controlados...1'
                        print 'Para calibrar mediante tamano de marcador.....2\n'
                        modoCalibracion = input("")
```

```
                        if modoCalibracion == 1:
                                # CALIBRACION POR DESPLAZAMIENTOS
                                calibracion.banderaCalibracion = True
                                break

                        elif modoCalibracion == 2:
                                # CALIBRACION POR TAMANO
                                calibracion.banderaCalibracion = False
                                break
                        else:
                                print "Incorrecto. Ingrese una de las opciones\n"

                calibracion.calibrar()

        elif eleccion == 2:

                configuracion.inicio()

        elif eleccion == 3:

                if not configuracion.configHecho or not calibracion.calibHecho:
                        if not configuracion.configHecho:
                                print "La configuracion no se ha hecho. Seguro que desea terminar?"
                                seguridad = input("")
                                if seguridad == 'si':
                                        break
                        else:
                                if not calibracion.calibHecho:
                                        print "La calibracion no se ha hecho. Seguro que desea
terminar?"
                                        seguridad = input("")
                                        if seguridad == 'si':
                                                break
                else:
                        break

        else:
                print "Incorrecto"

print 'FIN!'
```

[ANEXO 2]
[Código de Módulo de Medición y Comunicación]

```
import cv2
import time
import rpyc
import numpy as np
from matplotlib import pyplot as plt
import threading
import scipy.io
from configobj import ConfigObj


#####CLASE DE SERVICIOS REMOTOS#####

class peticionService(rpyc.Service):

        def on_connect(self):

    # code that runs when a connection is created
    # (to init the serivce, if needed)
                print "Conexion establecida"

        def on_disconnect(self):

    # code that runs when the connection has already closed
    # (to finalize the service, if needed)
                Atencion.close()


        def exposed_inicio(self):
                Hilo.ciclo = cv2.VideoCapture(0)

                while not Hilo.ciclo.isOpened():
                        pass

                Hilo.start()
                print "Ciclo de mediciones iniciado"

        def exposed_mediciones(self):
                for i in xrange(len(Hilo.DXM)):
                        for j in xrange(len(Hilo.DXM[0])):
                                Hilo.medicionesR[i][j] = (Hilo.DXM[i][j],Hilo.DYM[i][j])

                return Hilo.medicionesR

        def exposed_terminar(self):
                Hilo.banderaCierre = True

                while Hilo.ciclo.isOpened():
                        pass

                Hilo.banderaTermino = True

                print "Ciclo detenido"

        def exposed_tomarFoto(self):
```

```
                Hilo.banderaUsuario = True

        def exposed_mat(self):
                M = scipy.io.loadmat('derivas.mat')
                return M


#C L A S E  D E  A D Q U I S I C  Y  P R O C  D E  I M A G E N E S

class Medicion(threading.Thread):

        def __init__(self):

                threading.Thread.__init__(self)

                # INIC. DE VARIABLES
                self.regiones = []
                self.centrosMasa = []
                self.pares = []
                self.referencia = []
                self.banderaCierre = False
                self.banderaUsuario = False
                self.banderaInicioVar = True
                self.banderaTermino = False
                self.banderaConfig = True
                self.banderaCalib = True
                self.contadorDeriva = []
                self.criterioObstaculoPermanente = 50
                self.errorX = -999
                self.errorY = -999
                self.areaMinima = 100
                self.configuracion = ConfigObj('configuracion.ini')
                self.areaMinimaPatron = 40000
                self.areaMaximaPatron = 60000
                self.deltaDX = 0
                self.deltaDY = 0

                # LECTURA DE MARCADOR PATRON
                img = cv2.imread('circulo.jpg',0)
                ret,umbral = cv2.threshold(img,80,255,cv2.THRESH_BINARY)
                (self.contornoPatron,_) =
cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)

                for indice in xrange(len(self.contornoPatron)):
                        area = cv2.contourArea(self.contornoPatron[indice])
                        if area > self.areaMinimaPatron and area < self.areaMaximaPatron:
                                self.indicePatron = indice

                # VERIFIC. DE PROCESOS DE CONFIG Y CALIB
                try:
                        self.indicePares = int(self.configuracion['indicePares'])
                except KeyError:
                        self.banderaConfig = False

                try:
                        self.factorConversion = float(self.configuracion['factor'])
                        self.contribdeltafactor = float(self.configuracion['contribdeltafactor'])
                except KeyError:
```

```
                self.banderaCalib = False


        # LECTURA DE INFO EN ARCHIVO DE CONFIG
        if self.banderaConfig and self.banderaCalib:
                self.bancoImagenes = int(self.configuracion['bancoImagenes'])
                self.mediciones = int(self.configuracion['numeroMediciones'])
                self.deteccionesDeriva = int(self.configuracion['deteccionesDeriva'])
                self.distanciaDeriva = float(self.configuracion['distanciaDeriva'])
                self.indiceRegiones = int(self.configuracion['indiceRegiones'])
                self.stringFotoUsuario = self.configuracion['fotoUsuario']
                self.umbralSimil = float(self.configuracion['umbralSimil'])
                self.umbralF = int(self.configuracion['segmento'])

                for i in xrange(self.indicePares):
                        par = self.configuracion['par_%i' %(i)]

                        for j in xrange(2):
                                par[j] = int(par[j])

                        self.pares.append(par)
                        self.contadorDeriva.append([0,0])

                for i in xrange(self.indicePares):
                        ref = self.configuracion['referencia_%i' %(i)]

                        for j in xrange(2):
                                ref[j] = float(ref[j]) * self.factorConversion

                        self.referencia.append(ref)

                for i in xrange(self.indiceRegiones):
                        region = self.configuracion['region_%i' %(i)]
                        centro = self.configuracion['centroMasa_%i' %(i)]

                        for j in xrange(5):
                                region[j] = int(region[j])

                        for k in xrange(3):
                                centro[k] = float(centro[k])

                        self.regiones.append(region)
                        self.centrosMasa.append(centro)

                self.DXO = [0 for indice in xrange(len(self.pares))]
                self.DYO = [0 for indice in xrange(len(self.pares))]
                self.referenciasR = [0 for indice in xrange(len(self.pares))]
                self.DXM = [[0 for subindice in xrange(self.mediciones)] for indice in
xrange(len(self.pares))]
                self.DYM = [[0 for subindice in xrange(self.mediciones)] for indice in
xrange(len(self.pares))]
                self.medicionesR = [[0 for subindice in xrange(self.mediciones)] for indice in
xrange(len(self.pares))]
                self.distorsion = scipy.io.loadmat('distorsion.mat')

        else:
                self.banderaTermino = True
```

```
def run(self):

        while not self.banderaTermino:

            if self.banderaInicioVar:

                img = cv2.imread('contorno.jpg',0)

                banderaReinicioDeriva = False
                bandera = True
                indice = 0
                indiceMediciones = 0
                indiceRegiones = []
                banderaIndiceDeriva = False
                indiceDerivas = 0
                indiceCriterioD = 0
                derivasX = []
                derivasY = []
                stringFoto = 'foto'
                indiceFoto = 1
                indiceFotoUsuario = 1
                archivoMat = {}
                obstaculo = [0 for indice in xrange(len(self.regiones))]
                derivasX = [[0 for subindice in xrange(self.bancoImagenes-1)] for
indice in xrange(len(self.regiones))]
                derivasY = [[0 for subindice in xrange(self.bancoImagenes-1)] for
indice in xrange(len(self.regiones))]
                contadorDerivasX = [0 for indice in xrange(len(self.regiones))]
                contadorDerivasY = [0 for indice in xrange(len(self.regiones))]
                banderaDerivaPermanenteX = [False for indice in
xrange(len(self.regiones))]
                banderaDerivaPermanenteY = [False for indice in
xrange(len(self.regiones))]
                self.DXO = [0 for indice in xrange(len(self.pares))]
                self.DYO = [0 for indice in xrange(len(self.pares))]
                self.DXM = [[0 for subindice in xrange(self.mediciones)] for indice in
xrange(len(self.pares))]
                self.DYM = [[0 for subindice in xrange(self.mediciones)] for indice in
xrange(len(self.pares))]
                img2 = cv2.imread('contorno.jpg',0)
                txt=open("derivas.txt","w")
                txt.write("Derivas detectadas\n\n")

                for indiceP in xrange(len(self.pares)):
                    a = self.pares[indiceP][0]
                    b = self.pares[indiceP][1]
                    self.DXO[indiceP] = (self.centrosMasa[a][0]-
self.centrosMasa[b][0])
                    self.DYO[indiceP] = (self.centrosMasa[a][1]-
self.centrosMasa[b][1])

                for i in xrange(len(self.pares)):
                    self.referenciasR[i] = (float("{0:.1f}".format(self.DXO[i] *
self.factorConversion)),float("{0:.1f}".format(self.DYO[i] * self.factorConversion)))

                archivoMat['Referencia'] = self.referenciasR
                scipy.io.savemat('derivas.mat',archivoMat,oned_as='column')
```

```
                    self.banderaInicioVar = False

                if not self.banderaCierre:

                    while not self.ciclo.isOpened():
                        pass

                    # TIEMPO DE ADAPTACION DE CAMARA
                    for indice in xrange(20):
                        ret,imagen = self.ciclo.read()

                    #CICLO PRINCIPAL
                    while not self.banderaCierre:

                        # ADQUISICION DE CUADRO
                        ret,imagend = self.ciclo.read()

                        imagen = cv2.undistort(imagend,
self.distorsion['matrizoriginal'], self.distorsion['coeficientes'], None, self.distorsion['matriznueva'])
                        h,j,k,l = self.distorsion['region']
                        imagen = imagen[j:j+l, h:h+k]

                        # SALVADO DE CUADRO
                        cv2.imwrite('%s%i.jpg' %(stringFoto,indiceFoto), imagen)
                        if self.banderaUsuario :

            cv2.imwrite('%s%i.jpg' %(self.stringFotoUsuario,indiceFotoUsuario), imagen)
                            indiceFotoUsuario += 1
                            self.banderaUsuario = False

                        # LECTURA DE FOTO
                        self.foto =
cv2.imread('%s%i.jpg' %(stringFoto,indiceFoto),0)

                        # ESCRITURA EN ARCHIVO DE TEXTO
                        txt.write("
Foto %i: %s\n\n" %(indiceFoto,(time.strftime("%d-%m-%Y %H:%M:%S"))))


                        for indice in xrange(len(self.regiones)):
                            # SUBREGIONES DE BUSQUEDA
                            x  = self.regiones[indice][0]
                            x2 = self.regiones[indice][0] +
self.regiones[indice][2]
                            y  = self.regiones[indice][1]
                            y2 = self.regiones[indice][1] +
self.regiones[indice][3]
                            subregion = self.foto[y:y2,x:x2]
                            cx = 0
                            cy = 0
                            contador = 0

                            # EXTRACCION DE CONTORNOS
                            ret,umbral =
cv2.threshold(subregion,self.umbralF,255,cv2.THRESH_BINARY)
                            (contorno,_) =
cv2.findContours(umbral,cv2.RETR_LIST,cv2.CHAIN_APPROX_SIMPLE)
```

```
                                                    #RECONOC DE MARCADORES
                                                    for indiceC in xrange(len(contorno)):
                                                            area =
cv2.contourArea(contorno[indiceC])
                                                            if area > self.areaMinima:
                                                                    ret =
cv2.matchShapes(contorno[indiceC],self.contornoPatron[self.indicePatron],1,0.0)
                                                                    if ret < self.umbralSimil:
                                                                            momentos =
cv2.moments(contorno[indiceC])
                                                                            cx = cx +
(momentos['m10']/momentos['m00'])
                                                                            cy = cy +
(momentos['m01']/momentos['m00'])
                                                                            contador = contador +
1

                                                    try:
                                                            cx = cx / contador
                                                            cy = cy / contador

                                                            derivasX[indice][indiceFoto - 1] = cx + x
                                                            derivasY[indice][indiceFoto - 1] = cy + y


                                                    # NO HUBO DETECCION
                                                    except ZeroDivisionError:

                                                            obstaculo[indice] += 1

                                                            derivasX[indice][indiceFoto -1] =
self.errorX
                                                            derivasY[indice][indiceFoto -1] =
self.errorY

                                                    # CRITERIO DE OBSTACULO PERMANENTE
                                                            if obstaculo[indice] >=
self.criterioObstaculoPermanente:
                                                                    txt.write("  Recurrencia\n
Posible obstaculo permanente\n")



                                            # DISTANCIA ENTRE MARCADORES
                                            for indiceP in xrange(len(self.pares)):
                                                    a = self.pares[indiceP][0]
                                                    b = self.pares[indiceP][1]
                                                    c = self.referencia[indiceP][0]
                                                    d = self.referencia[indiceP][1]
                                                    txt.write("  Distancia (%i,%i)\n" %(a,b))

                                                    if derivasX[a][indiceFoto - 1] != self.errorX and
derivasX[b][indiceFoto - 1] != self.errorX:
                                                            DX = (derivasX[a][indiceFoto - 1] -
(derivasX[b][indiceFoto - 1])) * self.factorConversion

                                                            if DX < 0:
                                                                    DX = - DX
```

```
                                                    self.deltaDX = DX *
(((derivasX[a][indiceFoto - 1] - derivasX[b][indiceFoto - 1]) ** -2 + self.contribdeltafactor ) ** 0.5)


                                                    if DX >= c + self.distanciaDeriva or DX <=
c - self.distanciaDeriva:
                                                        txt.write("  Distancia Maxima
admisible superada\n")
                                                        self.contadorDeriva[indiceP][0]
+= 1
                                                    else:
                                                        if
self.contadorDeriva[indiceP][0] != 0:
            self.contadorDeriva[indiceP][0] -= 1

                                                    self.DXM[indiceP][indiceMediciones] =
float("{0:.1f}".format(DX))
                                                    txt.write("
X:%f\n" %(float( "{0:.1f}".format(self.DXM[indiceP][indiceMediciones]))))
                                                    txt.write("
dX:%f\n" %(float( "{0:.5f}".format(self.deltaDX))))

                                                    if self.contadorDeriva[indiceP][0] >=
self.deteccionesDeriva:
                                                        txt.write("  Detectado %i
veces\n" %(self.contadorDeriva[indiceP][0]))

                                                else:
                                                    self.DXM[indiceP][indiceMediciones] =
self.errorX
                                                    txt.write("  X: --Obstaculo\n")

                                                if derivasY[a][indiceFoto - 1] != self.errorY and
derivasY[b][indiceFoto - 1] != self.errorY:
                                                    DY = (derivasY[a][indiceFoto - 1] -
(derivasY[b][indiceFoto - 1])) * self.factorConversion

                                                    if DY < 0:
                                                        DY = - DY

                                                    self.deltaDY = DY *
(((derivasY[a][indiceFoto - 1] - derivasY[b][indiceFoto - 1]) ** -2 + self.contribdeltafactor ) ** 0.5)


                                                    if DY >= d + self.distanciaDeriva or DY <=
d - self.distanciaDeriva:
                                                        txt.write("  Distancia Maxima
admisible superada\n")
                                                        self.contadorDeriva[indiceP][1]
+= 1
                                                    else:
                                                        if
self.contadorDeriva[indiceP][1] != 0:
            self.contadorDeriva[indiceP][1] -=1
```

```
                                                        self.DYM[indiceP][indiceMediciones] =
float("{0:.1f}".format((DY)))
                                                        txt.write("
Y:%f\n" %(float("{0:.1f}".format(self.DYM[indiceP][indiceMediciones]))))
                                                        txt.write("
dY:%f\n\n" %(float( "{0:.5f}".format(self.deltaDY))))

                                                        if self.contadorDeriva[indiceP][1] >=
self.deteccionesDeriva:
                                                                txt.write("  Detectado %i
veces\n\n" %(self.contadorDeriva[indiceP][1]))

                                                else:
                                                        txt.write("  Y: --Obstaculo\n\n")

                                        # CONTROL INDICE FOTO
                                        indiceMediciones += 1
                                        if indiceMediciones == self.mediciones:
                                                indiceMediciones = 0
                                        indiceFoto += 1
                                        if indiceFoto == self.bancoImagenes:
                                                indiceFoto = 1

                                                for i in xrange(len(Hilo.DXM)):
                                                        for j in xrange(len(Hilo.DXM[0])):
                                                                self.medicionesR[i][j] =
(self.DXM[i][j],self.DYM[i][j])

                                                archivoMat['Mediciones'] = self.medicionesR


        scipy.io.savemat('derivas.mat',archivoMat,oned_as='column')
                                                txt.write("N U E V O   C I C L O\n")

                                        # CIERRE DE CAMARA
                                        txt.close()
                                        self.ciclo.release()
                                        while self.ciclo.isOpened():
                                                pass

                                else:
                                        pass


# # # # # # # # P R O G R A M A   P R I N C I P A L # # # # # # # # # # #

if __name__ == "__main__":

   # Instancia de clase de medicion
        Hilo = Medicion()

        from rpyc.utils.server import ThreadedServer
        Atencion = ThreadedServer(peticionService, port = 18861)

        if not Hilo.banderaTermino:

                #Servicio remoto
                print "Esperando conexion"
```

```
                Atencion.start()

        else:
                if not Hilo.banderaConfig and not Hilo.banderaCalib:
                        print "\nNo hay archivo de configuracion\n"
                        print "Programa terminado\n"

                else:
                        if not Hilo.banderaConfig:
                                print "\nNo se ha realizado configuracion\n"
                                print "Programa terminado\n"
                        else:
                                print "\nNo se ha realizado calibracion\n"
                                print "Programa terminado\n"

print "Programa terminado"
```

[ANEXO 3]
[Manual de Funcionamiento y Mantenimiento]

DISPOSITIVO CÁMARA-COMPUTADOR PARA MEDICIÓN DE DISTANCIAS MEDIANTE PROCESAMIENTO DE IMÁGENES.

Funcionamiento y Mantenimiento.

Funcionamiento.

La función del dispositivo es medir la distancia entre 2 puntos a partir de una imagen. El mismo se compone de una cámara web y computador embebido BeagleBoard-xM, la cámara se conecta a uno de los puertos USB host del computador. En el computador se ejecuta el sistema operativo Ubuntu 12.14 y el programa que procesa las imágenes ha sido desarrollado en el lenguaje Python y usa la librería OpenCV, de procesamiento de imágenes.

Para medir las distancias, el programa reconoce marcadores que deben estar presentes en la escena y según lo indicado por el usuario en el proceso de configuración, el programa mide la distancia entre un par de marcadores. El marcador es como el que se muestra en la siguiente figura.

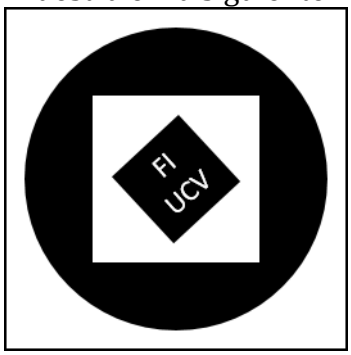

Figura 1. Marcador.

La orientación del dispositivo es medir las distancias que separan los elementos estructurales de una edificación para tener registro del estatus de la estructura y tomar medidas tempranas en caso de compromiso de la misma.

Instalación.

Se colocan sobre la estructura tantos marcadores como sean necesarios, tomando en cuenta que un par de marcadores delimita una distancia a medir y que un mismo marcador puede formar parte de varios pares. Es importante que todos los marcadores estén en el mismo plano. A continuación se muestra un ejemplo de instalación de marcadores.

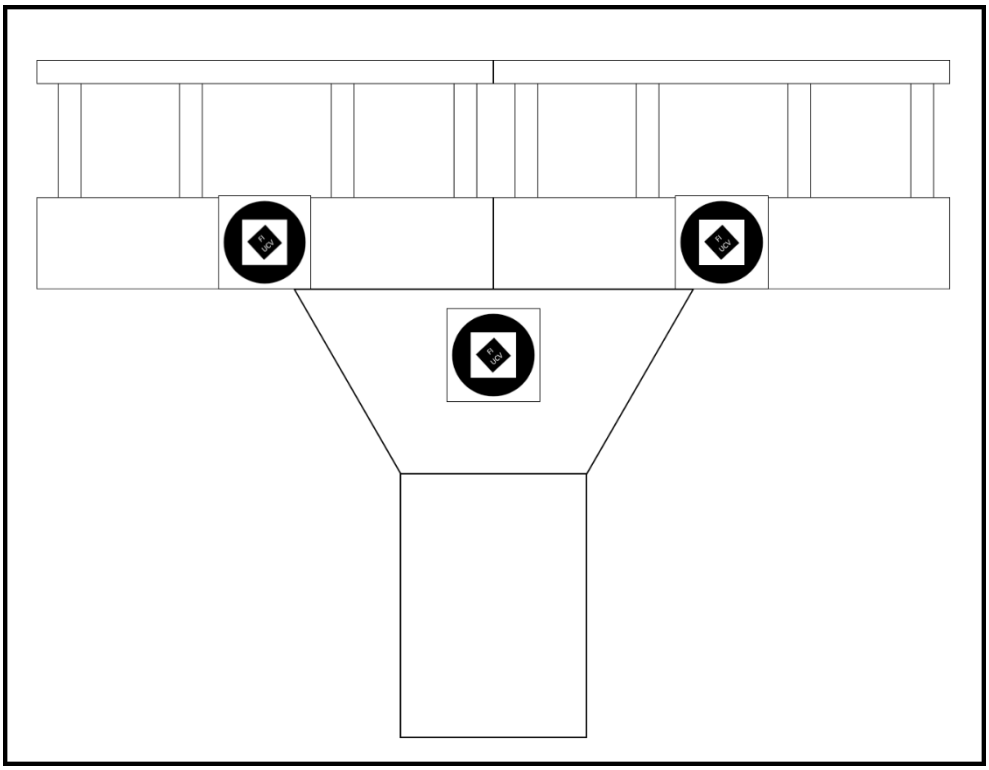

Figura 2. Instalación de marcadores.

La cámara debe ser situada de forma que todos los marcadores sean captados por ella. Es importante que el plano de los marcadores y el del lente de la cámara sean paralelos, en aras de obtener medidas confiables.

La cámara debe ser conectada a uno de los puertos USB del computador y éste al suministro de energía.

Calibración y configuración.

Para calibrar y configurar el dispositivo debe ejecutarse el programa configuración.py. Al ejecutarlo se presentará un menú que permite el proceso a realizar, como el siguiente:

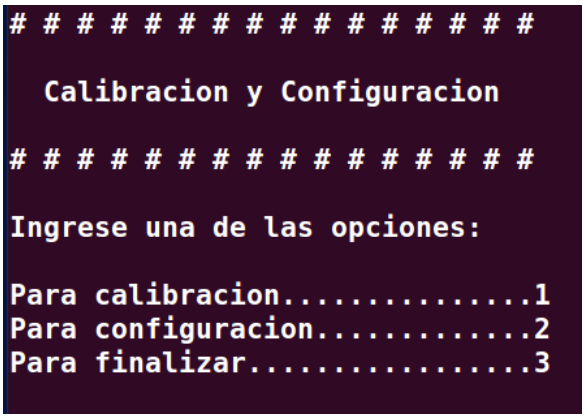

Figura 3. Menú principal del programa de calibración y configuración.

Es importante que ambos procesos sean realizados, de lo contrario el programa de medición no se ejecutará. Todos los parámetros fijados quedan a disposición del programa de medición mediante un archivo llamado configuración.ini

Calibración.

El objetivo de este proceso es determinar el factor que permite convertir una medida en unidades de píxeles a unidades de longitud. La calibración puede hacerse de dos maneras: mediante un desplazamiento controlado o mediante el tamaño del marcador. La forma de calibrar el dispositivo se elige en el menú que es como el que se presenta a continuación.

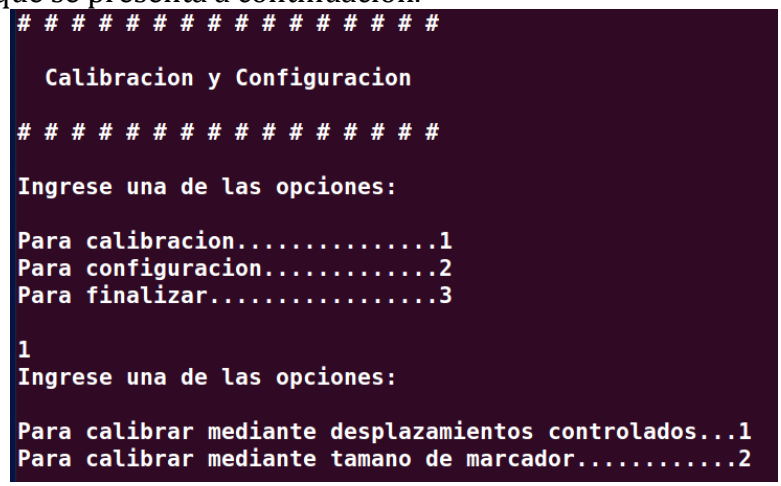

Figura 4. Menú para elegir el método luego de elegir calibración.

Al elegir calibración por desplazamiento controlado, el programa adquiere una imagen y reconoce un marcador, luego pide al usuario que lo desplace en una sola dimensión una distancia conocida, el programa vuelve a reconocer el marcador en la imagen y mide su desplazamiento en píxeles y pregunta al usuario qué distancia en centímetro fue movido, con ambos datos se calcula el factor de conversión.

Para calibrar por tamaño de marcador, el programa reconoce un marcador en la imagen y mide el diámetro de círculo negro en píxeles y pregunta al usuario cuánto mide el diámetro en centímetros y con ambos datos se calcula el factor de conversión. La ecuación con que se calcula es la siguiente:

$$factor\ de\ conversión = \frac{desplazamiento\ o\ diámetro\ (cm)}{desplazamiento\ o\ diámetro\ (píxeles)}$$

Configuración.

En esta etapa el usuario introduce los siguientes datos:

- Las distancias a medir, delimitadas por pares de marcadores.
- El número de fotos que constituyen el banco de registro de fotos.
- El número de mediciones recientes que serán servidos bajo petición externa.
- La distancia máxima admisible, en centímetros, al detectar que una de las piezas supera esta distancia e programa lo reporta en los resultados.
- El número de veces que debe ser detectada la distancia máxima admisible para ser considerada como deriva permanente.
- La cadena de caracteres que se usará para guardar las fotos que se toman bajo petición externa.

En la caso de las distancias a medir, el programa reconoce y numera todos los marcadores presentes en la escena y los presenta al usuario quien debe introducir qué distancias a medir mediante pares ordenados de la forma (x, y). La siguiente es una imagen ejemplo de la que el programa presenta al usuario luego de reconocer y numerar los marcadores.

Figura 5. El programa reconoce, numera los marcadores y los presenta al usuario.

El resto de los parámetros son datos numéricos o de texto.

Medición y Servicio de peticiones externas.

El programa de medición y servicio de peticiones tiene el nombre de hilo2.py, una vez que se ejecute chequea la presencia del archivo configuración.ini donde deben estar todos los parámetros resultantes de los procesos de calibración y configuración y queda a la espera de la conexión con un cliente externo, esta conexión se establece mediante la librería RPyC para el lenguaje Python; si falta algún parámetro el programa así lo advertirá y termina su ejecución. El proceso de mediciones es continuo y una vez que inicia queda ejecutándose en segundo plano, no ofrece ninguna interfaz al usuario. Este programa está diseñado para atender las siguientes peticiones externas: inicio y fin del ciclo de mediciones, desconexión del cliente externo, toma de foto bajo demanda, reportar las mediciones más recientes tanto en formato diccionario (.mat) para ser procesado por algún paquete matemático externo como en un arreglo matricial,

Todas las mediciones quedan documentadas en un archivo de texto con extensión .txt, en este archivo es donde quedan reportadas las detecciones de obstáculos y de derivas permanentes.

En el cliente externo debe ejecutarse el programa remoto.py para poder establecer la conexión y hacer las peticiones al programa de medición. Al ejecutar el programa remoto.py se presenta al usuario el siguiente menú.

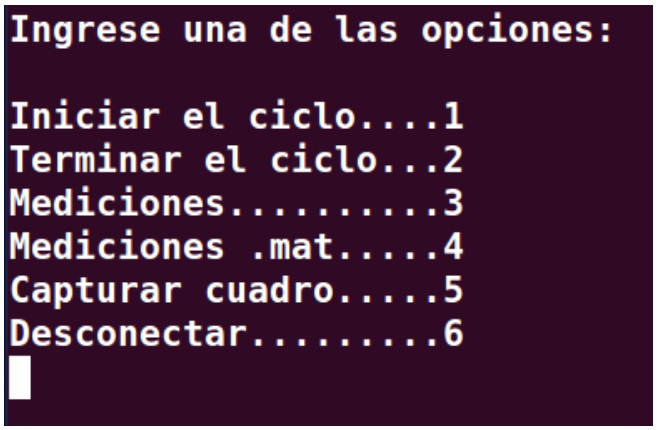

Figura 6. Menú de programa de cliente remoto.

Mantenimiento.

Marcadores.

Está previsto que lo marcadores sean impresos en vinil, material recomendado para la intemperie. Se recomienda realizar una limpieza cada 6 meses de los marcadores.

Cámara y computador.

Se recomienda el uso de una carcasa para el conjunto cámara-computador para proveerlo de resistencia ante la intemperie. Debe estar ubicado fuera del alcance del público general y se recomienda una limpieza externa de la carcasa y despeje de la zona circundante cada 6 meses.