Importamos las librerias para manejar los motores y cámara del robot así como otras que nos ayudarán a trabajar con las imágenes.

In [ ]:
```
import RPi.GPIO as GPIO
import time
import cv2
import numpy as np
from queue import Queue
from threading import Thread
```

Creamos unas funciones que nos serán útiles para mover el robot y obtener imágenes de la cámara.

In [2]:
```
# Número de pines que usaremos para controlar los motores
hMotorPins = [21, 20, 16, 12]
vMotorPins = [26, 19, 13, 6]

# Secuencia que habrá que enviar para efectuar un ciclo de medios pasos en
motorSequence = [
    [1, 0, 0, 1],
    [1, 0, 0, 0],
    [1, 1, 0, 0],
    [0, 1, 0, 0],
    [0, 1, 1, 0],
    [0, 0, 1, 0],
    [0, 0, 1, 1],
    [0, 0, 0, 1]]


def start():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM) # Definimos la forma en que vamos a referenciar
    time.sleep(0.1)
    # Ponemos todos los pines como out y low
    for pin in hMotorPins + vMotorPins:
        GPIO.setup(pin, GPIO.OUT)
        GPIO.output(pin, 0)
    time.sleep(0.1)

def stop():
    GPIO.cleanup()
    time.sleep(0.1)


### Definimos los distintos movimientos de los motores ###

# 512 ciclos harían una revolución completa de motor => 512 / 360 * ángulo
# 1 ciclo = 8 medios pasos
# En cada medio paso debemos dar valores a 4 pines
# Después de dar esos valores, debemos esperar un tiempo mínimo antes del

def moveRight(cycles):
    global curPos
    start()
    for i in range(cycles):
```

```
        for halfstep in range(8):
            for pin in range(4):
                GPIO.output(hMotorPins[pin], motorSequence[halfstep][pin])
            time.sleep(0.001)
        curPos += 1
    stop()

def moveLeft(cycles):
    global curPos
    start()
    for i in range(cycles):
        for halfstep in reversed(range(8)):
            for pin in reversed(range(4)):
                GPIO.output(hMotorPins[pin], motorSequence[halfstep][pin])
            time.sleep(0.001)
        curPos -= 1
    stop()


def moveUp(cycles):
    start()
    for i in range(cycles):
        for halfstep in reversed(range(8)):
            for pin in reversed(range(4)):
                GPIO.output(vMotorPins[pin], motorSequence[halfstep][pin])
            time.sleep(0.001)
    stop()


def moveDown(cycles):
    start()
    for i in range(cycles):
        for halfstep in range(8):
            for pin in range(4):
                GPIO.output(vMotorPins[pin], motorSequence[halfstep][pin])
            time.sleep(0.001)
    stop()


# Tomaremos 60 fotos dentro de los rangos definidos para cada nodo
def get60Images(nodePos, path):
    global curPos, frame, takingPhotos
    takingPhotos = True
    moveLeft(curPos)
    for np in range(len(nodePos)):
        imNo = 0
        pos = nodePos[np]
        moveRight(pos[0] - curPos)
        time.sleep(0.2)
        start()
        photoAt = (8 * (pos[1] - pos[0])) / 60
        for i in range(pos[1] - pos[0]):
            for halfstep in range(8):
                for pin in range(4):
                    GPIO.output(hMotorPins[pin], motorSequence[halfstep][p
                if int((i * 8 + halfstep) % photoAt) == 0:
                    time.sleep(0.4)
```

```
                    cv2.imwrite(path + 'nodo_' + str(np + 1) + '_img_' + st
                    imNo += 1
                else:
                    time.sleep(0.001)
            curPos += 1
        stop()
    takingPhotos = False
    moveLeft(curPos)


# Esta función nos muestra, con algunas anotaciones (Nodo y posiciones que
# y ciclos que giraremos) lo que se está viendo a través de la cámara
def showImage(cycles, left, frame, node, nodePos, vl):
    global takingPhotos

    if not takingPhotos:
        text = 'Node ' + str(node) + ': ['

        if len(nodePos) >= node:
            if nodePos[node - 1][0] != -1:
                text += str(nodePos[node - 1][0]) + ', '
            else:
                text += ' - , '
            if nodePos[node - 1][1] != -1:
                text += str(nodePos[node - 1][1]) + '] '
            else:
                text += '  - ] '
        else:
            text += ' - ,  - ] '

        if cycles != 0:
            if left:
                text += ' Move: -' + str(cycles)
            else:
                text += ' Move: +' + str(cycles)

        if vl:
            cv2.line(frame, pt1 = (frame.shape[1] // 2, 0), pt2 = (frame.sh

        cv2.imshow("Frame", cv2.putText(frame, text, (50, 50), cv2.FONT_HE
    else:
        cv2.imshow("Frame", frame)


# Esta función nos muestra, con algunas anotaciones (Ciclos que giraremos y
# lo que se está viendo a través de la cámara
def showImageClassified(cycles, left, frame, lab, trueLab, nodeLabels):
    global rec
    recNames = ['1-NN-d+HistBW', '1-NN-d+HistRBG', '1-NN-d+Mat']
    if cycles != 0:
        if left:
            cv2.putText(frame, ' Move: -' + str(cycles), (50, 30), cv2.FONT
        else:
            cv2.putText(frame, ' Move: +' + str(cycles), (50, 30), cv2.FONT

    cv2.putText(frame, 'True Node: ' + str(trueLab), (50, 60), cv2.FONT_HE
```

```
    # Anotamos los resultados en la imagen (verde si coincide, rojo si son
    if lab != None:
        if lab == trueLab:
            cv2.putText(frame, recNames[rec] + ': ' + nodeLabels[lab - 1],
        else:
            cv2.putText(frame, recNames[rec] + ': ' + nodeLabels[lab - 1],
    else:
        if trueLab == 0:
            cv2.putText(frame, recNames[rec] + ': Unknown', (50, 90), cv2.I
        else:
            cv2.putText(frame, recNames[rec] + ': Unknown', (50, 90), cv2.I

    cv2.imshow("Frame", frame)
```

Parte 1: Obtención del dataset de entrenamiento

El siguiente código nos ayudará a recoger alrededor de 60 imágenes por nodo.

- 't' (turn): Gira la cámara según el número de pasos definidos por el usuario.
- 'h' (home): Vuelve a la posición inicial.
- 'p' (photos): Gira según el ángulo definido hacia la derecha tomando alrededor de 60 fotos y vuelve a la posición original.

La manera de conseguir nuestro dataset será primero orientar la cámara mediante comandos 't' de manera que vea la imagen de más a la izquierda de un nodo y guardar esta posición asociándola a un nodo (l), movernos hacia la derecha mediante comandos 't' hasta encontrar la posición de la imagen de más a la derecha del nodo y una vez hecho esto para todos los nodos usaremos el comando 'p' que volverá a la posición inicial y hará un giro de 360 tomando 60 fotos dentro del rango de posiciones de cada nodo.

In [6]:
```
# Inicializamos variables
cycles = 0
left = False
vl = False
node = 1
path = 'Dataset/'
curPos = 0
nodePos = [[15, 48], [51, 93], [96, 156], [159, 199], [202, 287], [317, 36
takingPhotos = False

cap = cv2.VideoCapture(0)

# Definimos la resolución de la cámara
cap.set(3, 640)
cap.set(4, 480)

while(True):

    # Leer y mostrar (anotado) un frame de la cámara
    ret, frame = cap.read()
    showImage(cycles, left, frame, node, nodePos, vl)
```

```
        # Comprobar el input del usuario
        key = cv2.waitKey(1) & 0xFF

        # q: salir
        if key == ord("q"):
            break

        # h: vuelve a la posición 0
        elif key == ord("h"):
            Thread(target = moveLeft, args =(curPos, )).start()

        # t: girar en una dirección u otra los ciclos que haya definido el usu
        elif key == ord("t"):
            if left:
                if curPos - cycles < 0:
                    cycles = curPos
                Thread(target = moveLeft, args =(cycles, )).start()
            else:
                if curPos + cycles > 512:
                    cycles = 512 - curPos
                Thread(target = moveRight, args =(cycles, )).start()

        # p: tomar 60 fotos mientras se barre el ángulo que haya definido el u
        elif key == ord('p'):
            if min([p for pos in nodePos for p in pos]) >= 0: # Comprobamos qu
                Thread(target = get60Images, args =(nodePos, path, )).start()

        # 0-9: nos permite ir escribiendo el número de ciclos
        elif key >= 48 and key <= 57:
            cycles = cycles * 10 + key - 48
            if cycles > 512:
                cycles = 512

        # -: cambia el signo del movimiento (- izquierda, + derecha)
        elif key == ord("-"):
            left = not left

        # DEL: borrar el último dígito de los ciclos que haya definido el usua
        elif key == 8:
            cycles = int(cycles // 10)
            if cycles <= 0:
                cycles = 0
                left = False

        # m: incrementar el contador de nodo
        elif key == ord("m"):
            node += 1

        # n: decrementar el contador de nodo
        elif key == ord('n'):
            if node > 1:
                node -=1

        # l: guarda la posición de la imagen de más a la izquierda de un nodo
        elif key == ord("l"):
            if len(nodePos) < node:
```

```
            nodes = len(nodePos)
            for i in range(node - nodes):
                nodePos.append([-1, -1])
        nodePos[node - 1][0] = curPos

    # r: guarda la posición de la imagen de más a la derecha de un nodo
    elif key == ord("r"):
        if len(nodePos) < node:
            nodes = len(nodePos)
            for i in range(node - nodes):
                nodePos.append([-1, -1])
        nodePos[node - 1][1] = curPos

    # z: dibuja una línea vertical en el centro de la imagen (Nos sirve pa
    elif key == ord("z"):
        vl = not vl

moveLeft(curPos) # Volvemos a la posición inicial

for i in range(len(nodePos)):
    print('Node ' + str(i + 1) + ': [' + str(nodePos[i][0]) + ', ' + str(n

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

```
Node 1: [15, 48]
Node 2: [51, 93]
Node 3: [96, 156]
Node 4: [159, 199]
Node 5: [202, 287]
Node 6: [317, 367]
Node 7: [382, 407]
Node 8: [410, 490]
```

Parte 2: Reconocedores

In [2]:
```
import cv2
import numpy as np
```

1) El reconocedor de base de este trabajo práctico es el 1-Nearest Neighbor con umbral ajustable + histograma de grises normalizado, debido a su simplicidad conceptual y eficiencia estática y dinámica.

In [3]:
```
def getNormalizedHistogramBW(img):
    return cv2.calcHist([cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)], [0], None,

def r1NNdHistBW(img, trainFeatures, trainLabels, threshold):
    d = np.linalg.norm(trainFeatures - getNormalizedHistogramBW(img), axis
    idxDmin = d.argmin()
    if d[idxDmin] <= threshold:
        return trainLabels[idxDmin]
    else:
        return None
```

2) Adicionalmente, se propone trabajar con el 1-NN de umbral ajustable + histograma RGB normalizado y vectorizado(se encadenan en un supervector 256+256+256 elementos) los histogramas RGB normalizados de las imágenes.

In [4]:
```
def getNormalizedHistogramRBG(img):
    return np.vstack(tuple(cv2.calcHist([img], [i], None, [256], [0,256]) :

def r1NNdHistRBG(img, trainFeatures, trainLabels, threshold):
    d = np.linalg.norm(trainFeatures - getNormalizedHistogramRBG(img), axis
    idxDmin = d.argmin()
    if d[idxDmin] <= threshold:
        return trainLabels[idxDmin]
    else:
        return None
```

3) Adicionalmente, se propone también trabajar con el algoritmo 1-NN de umbral ajustable basado en el concepto de distancia o diferencia de las imágenes matriciales. En este caso se puede experimentar con diversos procedimientos heurísticos de suavizado de las imágenes matriciales(por rango Imax - Imin o aplicando la función de normalización minmax de opencv).

In [5]:
```
def r1NNdMat(img, trainFeatures, trainLabels, threshold):
    d = np.square(trainFeatures - cv2.normalize(cv2.cvtColor(img, cv2.COLOR
    idxDmin = d.argmin()
    if d[idxDmin] <= threshold * 500:
        return trainLabels[idxDmin]
    else:
        return None
```

Parte 3: Pruebas y Testeo del reconocedor de landmarks

1.- Creación del Conjunto de Muestras Etiquetadas CME o data sets de entrenamiento: se deben captar al menos 60 tomas para cada uno de los landmarks del mapa-grafo a reconocer, por lo que el data set de entrenamiento y test deberá contar con P datos, con P = (nº landmarks del mapa) x 60:

DS = {(x1, α1) ..... (xP ,αP) }; siendo x el vector de variables discriminantes (histograma de grises normalizado) y α es la etiqueta(landmark) del correspondiente dato

In [15]:
```
nNodes = 8
nImgPerNode = 60

trainFeatures1NNdHistBW = np.array([getNormalizedHistogramBW(cv2.imread('Da
trainFeatures1NNdHistRBG = np.array([getNormalizedHistogramRBG(cv2.imread(
trainFeatures1NNdMat = np.array([cv2.normalize(cv2.imread('Dataset/nodo_' +
trainFeatures1NNdHistBWNight = np.array([getNormalizedHistogramBW(cv2.imrea
trainFeatures1NNdHistRBGNight = np.array([getNormalizedHistogramRBG(cv2.im
trainFeatures1NNdMatNight = np.array([cv2.normalize(cv2.imread('NightDatase

trainLabels = np.array([range(0, nNodes)]).repeat(nImgPerNode)
```

2.-Testeo estático de los reconocedores:

In [101…
```
def LooCrossValidation(fun, features, labels, threshold):
    global nImgPerNode

    P = features.shape[0]
    nErrors = 0
    ind = np.arange(P) # Row indices

    for step in range(P):
        label = fun(cv2.imread('Dataset/nodo_' + str((step // nImgPerNode)
        if label != labels[step]:
            nErrors += 1

    print('True error rate % = ' + str((nErrors / P) * 100))
```

- **a)Testeo estático del reconocedor básico:** se estimarán las tasas de errores por el método de validación cruzada "leave-one-out" del reconocedor básico (1-NN con umbral ajustable + histograma normalizado de grises).

In [94]:
```
LooCrossValidation(r1NNdHistBW, trainFeatures1NNdHistBW, trainLabels, 0.1)
```

```
True error rate % = 0.20833333333333334
```

- **b)Testeo estático del reconocedor 1-NN de umbral ajustable + histograma RGB normalizado y vectorizado:** se estimarán las tasas de errores por el método de validación cruzada "leave-one-out" del reconocedor 1-NN de umbral ajustable + histograma RGB normalizado y vectorizado.

In [8]:
```
LooCrossValidation(r1NNdHistRBG, trainFeatures1NNdHistRBG, trainLabels, 0.
```

```
True error rate % = 0.20833333333333334
```

- **c)Testeo estático del reconocedor 1-NN de umbral ajustable basado en el concepto de distancia o diferencia de las imágenes matriciales:** se estimarán las tasas de errores por el método de validación cruzada "leave-one-out" del reconocedor 1-NN de umbral ajustable basado en el concepto de distancia o diferencia de las imágenes matriciales.

In [102…
```
LooCrossValidation(r1NNdMat, trainFeatures1NNdMat, trainLabels, 0.1)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-102-d08032ff5106> in <module>
----> 1 LooCrossValidation(r1NNdMat, trainFeatures1NNdMat, trainLabels, 0.1
)

<ipython-input-101-45efe4fe5442> in LooCrossValidation(fun, features, label
s, threshold)
      7
      8     for step in range(P):
----> 9         label = fun(cv2.imread('Dataset/nodo_' + str((step //
nImgPerNode) + 1) + '_img_' + str(step % nImgPerNode) + '.jpg'), features[i
nd != step, :], labels[ind != step], threshold)
     10         if label != labels[step]:
     11             nErrors += 1

KeyboardInterrupt:
```

3.- Testeo dinámico del navegador: una vez testeado en estática el reconocedor del navegador, pasamos al testeo dinámico contra vídeos de recorridos del entorno de navegación para estimar la eficiencia del navegador en el reconocimiento de los landmarks del entorno.

In [21]:

```
recNames = ['1-NN-d+HistBW', '1-NN-d+HistRBG', '1-NN-d+Mat']

videoPos = [[170, 240], [245, 410], [415, 645], [650, 850], [855, 1300], [
nightVideoPos = [[110, 185], [190, 305], [310, 440], [445, 545], [550, 795
out = True
curFrame = 0

nErrors = 0

nodeLabels = ['Guitarras', 'Television', 'Cortina', 'Sofa', 'Mesa', 'Tecla

trueLab = 0
rec = 1

cap = cv2.VideoCapture('./video.avi')

fourcc = cv2.VideoWriter_fourcc('X','V','I','D')
videoWriter = cv2.VideoWriter('/Users/fer/Documents/UPM/Master_IA/RA/Recon

ret, frame = cap.read()

while(ret):

    #if curFrame % 50 == 0:
    #    print(curFrame)

    if out:
        if videoPos[trueLab][0] == curFrame:
            out = False
    else:
        if videoPos[trueLab][1] == curFrame:
            trueLab += 1
            out = True
            if trueLab >= len(videoPos):
```

```
                trueLab = 0

    # Hacemos la clasificación con los 3 reconocedores
    if rec == 0:
        lab = r1NNdHistBW(frame, trainFeatures1NNdHistBW, trainLabels, 0.1
    elif rec == 1:
        lab = r1NNdHistRBG(frame, trainFeatures1NNdHistRBG, trainLabels, 0
    elif rec == 2:
        lab = r1NNdMat(frame, trainFeatures1NNdMat, trainLabels, 0.1)

    if out:
        cv2.putText(frame, 'Unknown', (50, 60), cv2.FONT_HERSHEY_SIMPLEX, (
    else:
        cv2.putText(frame, nodeLabels[trueLab], (50, 60), cv2.FONT_HERSHEY_

    # Anotamos los resultados en la imagen (verde si coincide, rojo si son
    if lab != None:
        if not out and lab == trueLab:
            cv2.putText(frame, recNames[rec] + ': ' + nodeLabels[lab], (50,
        else:
            cv2.putText(frame, recNames[rec] + ': ' + nodeLabels[lab], (50,
            nErrors += 1
    else:
        if out:
            cv2.putText(frame, recNames[rec] + ': Unknown', (50, 90), cv2.I
        else:
            cv2.putText(frame, recNames[rec] + ': Unknown', (50, 90), cv2.I
            nErrors += 1

    videoWriter.write(frame)

    # Comprobar el input del usuario
    key = cv2.waitKey(1) & 0xFF

    # q: salir
    if key == ord("q"):
        break

    # m: siguiente reconocedor
    elif key == ord("m"):
        if rec < 2:
            rec += 1

    # n: reconocedor anterior
    elif key == ord("n"):
        if rec > 0:
            rec -= 1

    ret, frame = cap.read()
    curFrame += 1

print('True error rate % = ' + str((nErrors / (curFrame - 1)) * 100))

# When everything done, release the capture
cap.release()
videoWriter.release()
```

```
True error rate % = 30.550918196994992
```

Incluimos el código para hacer testeo dinámico en el robot

In [ ]:

```
nodeLabels = ['Guitarras', 'Television', 'Cortina', 'Sofa', 'Mesa', 'Teclad
nodePos = [[15, 48], [51, 93], [96, 156], [159, 199], [202, 287], [317, 36

# Inicializamos variables
cycles = 0
left = False
curPos = 0
trueLab = 0
rec = 0

cap = cv2.VideoCapture(0)

# Definimos la resolución de la cámara
cap.set(3, 640)
cap.set(4, 480)

while(True):
    # Leer y mostrar (anotado) un frame de la cámara
    ret, frame = cap.read()

    # Hacemos la clasificación con los 3 reconocedores
    if rec == 0:
        lab = r1NNdHistBW(frame, trainFeatures1NNdHistBW, trainLabels, 0.1
    elif rec == 1:
        lab = r1NNdHistRBG(frame, trainFeatures1NNdHistRBG, trainLabels, 0
    elif rec == 2:
        lab = r1NNdMat(frame, trainFeatures1NNdMat, trainLabels, 0.1)
    trueLab = 0
    for i in range(len(nodePos)):
        if nodePos[i][0] <= curPos and nodePos[i][1] >= curPos:
            trueLab = i + 1

    showImageClassified(cycles, left, frame, lab, trueLab, nodeLabels)

    # Comprobar el input del usuario
    key = cv2.waitKey(1) & 0xFF

    # q: salir
    if key == ord("q"):
        break

    # h: vuelve a la posición 0
    elif key == ord("h"):
        Thread(target = moveLeft, args =(curPos, )).start()

    # t: girar en una dirección u otra los ciclos que haya definido el usu
    elif key == ord("t"):
        if left:
            if curPos - cycles < 0:
                cycles = curPos
            Thread(target = moveLeft, args =(cycles, )).start()
        else:
            if curPos + cycles > 512:
```

```
                cycles = 512 - curPos
            Thread(target = moveRight, args =(cycles, )).start()

    # 0-9: nos permite ir escribiendo el número de ciclos
    elif key >= 48 and key <= 57:
        cycles = cycles * 10 + key - 48
        if cycles > 512:
            cycles = 512

    # -: cambia el signo del movimiento (- izquierda, + derecha)
    elif key == ord("-"):
        left = not left

    # DEL: borrar el último dígito de los ciclos que haya definido el usua
    elif key == 8:
        cycles = int(cycles // 10)
        if cycles <= 0:
            cycles = 0
            left = False

    # m: siguiente reconocedor
    elif key == ord("m"):
        if rec < 2:
            rec += 1

    # n: reconocedor anterior
    elif key == ord("n"):
        if rec > 0:
            rec -= 1

moveLeft(curPos) # Volvemos a la posición inicial

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Incluimos también un código para grabar un video de un giro de 360 grados en el robot.

In [ ]:

```
import cv2

working = True
started = False

def tour():
    global working
    time.sleep(3)
    start()
    for i in range(512):
        for halfstep in range(8):
            for pin in range(4):
                GPIO.output(hMotorPins[pin], motorSequence[halfstep][pin])
            time.sleep(0.02)
    for i in range(512):
        for halfstep in reversed(range(8)):
            for pin in reversed(range(4)):
                GPIO.output(hMotorPins[pin], motorSequence[halfstep][pin])
            time.sleep(0.001)
```

```
    stop()
    working = False

cap = cv2.VideoCapture(0)

# Definimos la resolución de la cámara
cap.set(3, 640)
cap.set(4, 480)

fourcc = cv2.VideoWriter_fourcc('X','V','I','D')
videoWriter = cv2.VideoWriter('/home/pi/Desktop/video.avi', fourcc, 30.0,

while (working):

    ret, frame = cap.read()

    if ret:
        #cv2.imshow('video', frame)
        videoWriter.write(frame)

    if cv2.waitKey(1) == 27:
        break

    # Comprobar el input del usuario
    key = cv2.waitKey(1) & 0xFF

    # q: salir
    if key == ord("q"):
        break

    if not started:
        Thread(target = tour, args =( )).start()
        started = True

cap.release()
videoWriter.release()

cv2.destroyAllWindows()
```