Programación Técnica y Científica. Grado Ingeniería Informática (UGR)

Segunda práctica evaluable (1 punto)

Objetivo: Es habitual en la Programación Técnica y Científica la tarea de captura de datos en tiempo real, su correcto archivo y su tratamiento posterior. Para ello vamos a utilizar algunos módulos de Python como tkinter, matplotlib, json, os, glob junto a un simulador muy popular en el mundo de la robótica, denominado V-Rep (actualmente llamado CoppeliaSim). Este simulador nos suministrará una serie de datos de un sensor laser 2D simulado situado encima de un robot móvil también simulado. Con esos datos y usando los módulos tkinter y scikit-learn de pyhton programaremos una interfaz gráfica y utilizaremos un algoritmo de Machine Learning para entrenar un clasificador que nos permita identificar las piernas de las personas en los datos laser 2D. Finalmente, usando una escena de test, el robot detectará a las personas utilizando el sensor laser 2D y aplicando el clasificador previamente entrenado.

Instrucciones para la realización de la práctica

1. Preparación del entorno de trabajo

1.1. Módulos de python

Necesitamos tener instalado Python versión 3 con los módulos scikit-learn, tkinter y opencv. Los módulos scikit-learn y tkinter viene por defecto en la distribución Anaconda que ya instalamos al principio. En el caso de opencv existen varias posibilidades pues podemos instalarlo dentro de Anaconda o fuera. En nuestro caso vamos a instalarlo fuera de Anaconda, es decir NO usaremos la orden “conda install”. Lo instalamos en el sistema ya sea en Windows o en Linux. En el caso de Windows abrimos un prompt de Anaconda y ejecutamos `pip install opencv-python` (esto instalará la versión 4 de opencv).

Para instalarlo en entornos Linux hacer lo siguiente:

Abrir un terminal y ejecutar: `pip install --upgrade pip`

Y después: `pip install opencv-python`

Para comprobar que funciona en Prado tenéis en la carpeta “**Práctica evaluable 2. Tkinter y robótica**” el fichero ejemploOpenCV.py y el fichero leon.jpg. Este script muestra la imagen en una ventana hasta que presionamos la tecla escape.

1.2. Instalación del simulador Coppelia Sim (anteriormente VREP).

El simulador se puede descargar de la página de la empresa propietaria (http://www.coppeliarobotics.com/) que distribuye una versión “free” para uso educativo

Para esta práctica todos vamos a descargar la versión CoppeliaSim_Edu_V4_4_0_rev0_Ubuntu20_04

En sistemas linux realmente no se descarga un instalador como tal, sino que se baja un fichero comprimido que hay que descomprimir. Una vez descomprimido hay que entrar en la carpeta que contiene los ficheros y ejecutar `./coppeliaSim.sh`

1.3. Prueba de funcionamiento del simulador

Para lanzar el simulador en Linux ir a la carpeta donde está instalado, abrir una terminal y lanzar: `./coppeliaSim.sh`

Desde el menú File -> open scene se pueden abrir ejemplos de escenarios que hay dentro de la carpeta scenes. Por ejemplo cargar el fichero youBotAndHTower.ttt, para que empiece la simulación pulsar con el ratón el botón de “start” situado arriba a la derecha en la zona de la barra superior de iconos. La simulación se detiene con el botón de “stop”.

2. Ejemplos para controlar un robot y leer datos de los sensores desde Python

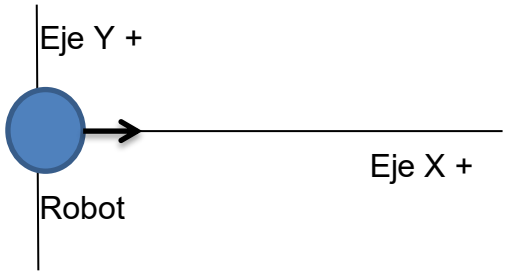
Hay que tener en cuenta que este simulador tiene un lenguaje propio de programación denominado **Lua** pero nosotros vamos a utilizar un lenguaje externo como es **Python**. Para ello, necesitamos los archivos `vrep.py`, `vrepConst.py` y `remoteApi.so`.

Vamos a trabajar con la escena "escenaLaser.ttt" para ver cómo funcionan los ejemplos y usaremos "escenaTest.ttt" para probar si hemos conseguido detectar bien las piernas de las personas con nuestro clasificador.

Todos los archivos necesarios están en Prado. Además tenemos los scripts:
"ejemploLaser_to_JSON.py" lee datos del laser del robot simulado y los almacena en formato json. Muestra lo que ve el robot con openCV. Salva los datos a un directorio diferente cada vez para no perder datos previos. Hay que resaltar que este script también muestra cómo poder mover y girar los objetos de la escena para conseguir diferentes configuraciones de distancia y ángulo de la posición de las personas o de las parejas de cilindros.
"ejemploLeerFicheroJSON.py" lee del fichero JSON para recuperar los datos del laser.
Toda estos archivos los descargamos de Prado y los almacenamos en una carpeta que se llame, por ejemplo, "practica2".

Para comprobar los ejemplos la forma de trabajar será la siguiente:

1. En un terminal ejecutamos el simulador y cargamos el escenario "escenaLaser.ttt".
2. Desde la carpeta practica2, donde están los scripts de la API de python, y el ejemplo "ejemploLaser_to_JSON.py" ejecutamos "spyder" (o bien otro editor de Python) y abrimos el script "ejemploLaser_to_JSON.py"
3. Iniciamos la simulación con el botón "start" del simulador. Esto hace que los objetos del escenario empiecen a publicar sus datos.
4. Desde el spyder ejecutamos el script de Python "ejemploLaser_to_JSON.py" que es capaz de leer tanto la imagen como los datos 2D del laser. Dicho script te muestra la imagen en una ventana de openCV y te pinta los datos del láser en un eje de coordenadas en la salida del terminal del Ipython. Se van salvando los datos del láser dejando un tiempo de X segundos entre cada iteración hasta un número máximo de iteraciones. Se salvan también las imágenes y los plots.
5. Si desde el simulador, con la simulación activada, movemos las personas o bien otros objetos se puede observar cómo van cambiando los datos recibidos por el script de Python.
6. Tened en cuenta que los datos del láser son puntos en coordenadas (x, y) centradas en el robot (0,0) siendo el eje X en sentido hacia adelante en la marcha, y el eje Y positivo queda a la izquierda del robot, y el eje Y negativo a su derecha, según la siguiente figura.



Una vez que se han probado los scripts de ejemplo podemos empezar a realizar la práctica del siguiente modo.

4. Realización de la práctica (Cambiar VREP por CoppeliaSim)

4.1. Implementación de la interfaz gráfica con tkinter (script mainInterfaz.py)

Para poder manejar todos los scripts que se comentan a continuación primero debemos implementar un entorno gráfico usando tkinter que tenga el aspecto de la Figura 1.

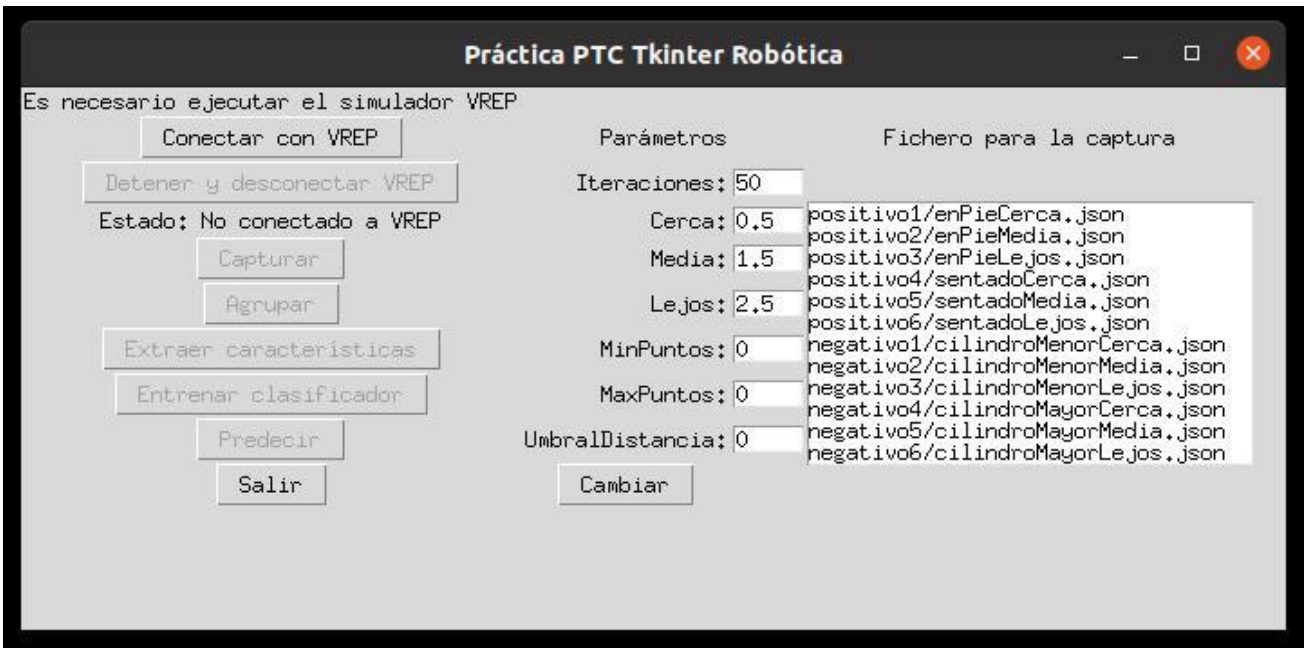


Figura 1

Para ello hay que utilizar el módulo tkinter y tener en cuenta:

`root.geometry("700x300")`. Columnas usadas en grid de 0 a 3. Filas usadas en grid de 0 a 9. Se inicia con los parámetros a unos valores por defecto. Se deben listar los 12 ficheros a capturar en un `Listbox(root, width=35, height=12)`. Inicialmente solo aparecen habilitados los botones como en la Figura 1. La etiqueta "Estado" debe reflejar en cada momento el estado de la conexión con el simulador. Para visualizar mejor las filas y columnas donde debe ir cada elemento se muestra la Figura 2.

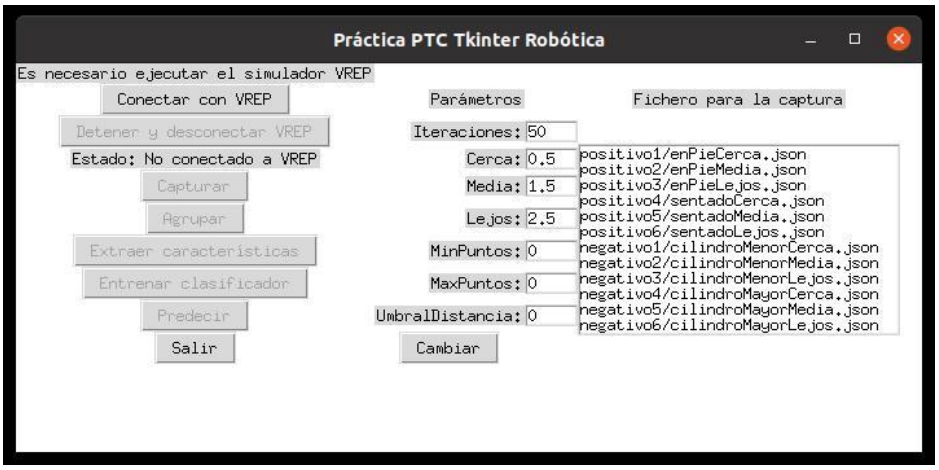


Figura 2

Creación de los directorios de trabajo: El script principal debe crear los directorios “positivo1” a “positivo6” y “negativo1” a “negativo6” más una carpeta llamada “prediccion”, todo al iniciarse, siempre y cuando no se hayan creado previamente.

Especificaciones de los diferentes elementos a utilizar

Botón “Conectar con VREP”: Intenta iniciar una conexión con el simulador. Si no es posible debe utilizar `messagebox.showerror()` para dar un mensaje de error como la Figura 3.

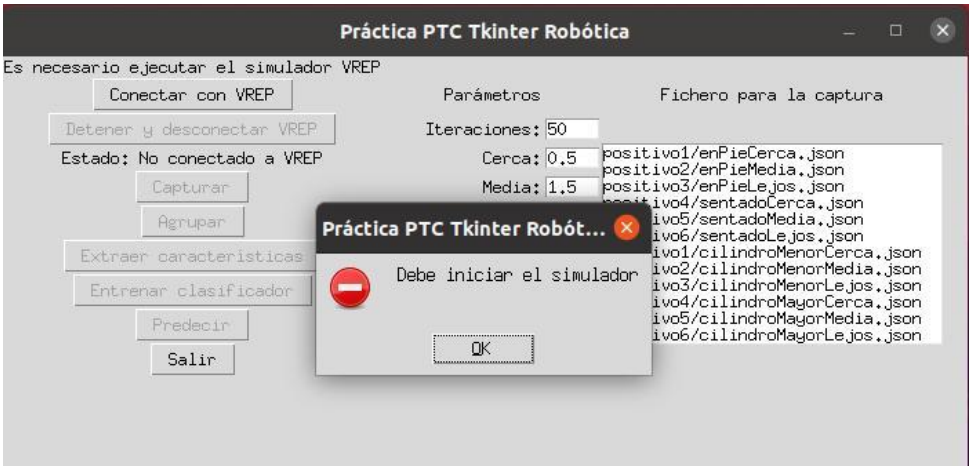


Figura 3

Si la conexión se establece correctamente debe utilizar `messagebox.showinfo()` para informar de que se ha establecido la conexión con el servidor. Ver Figura 4.

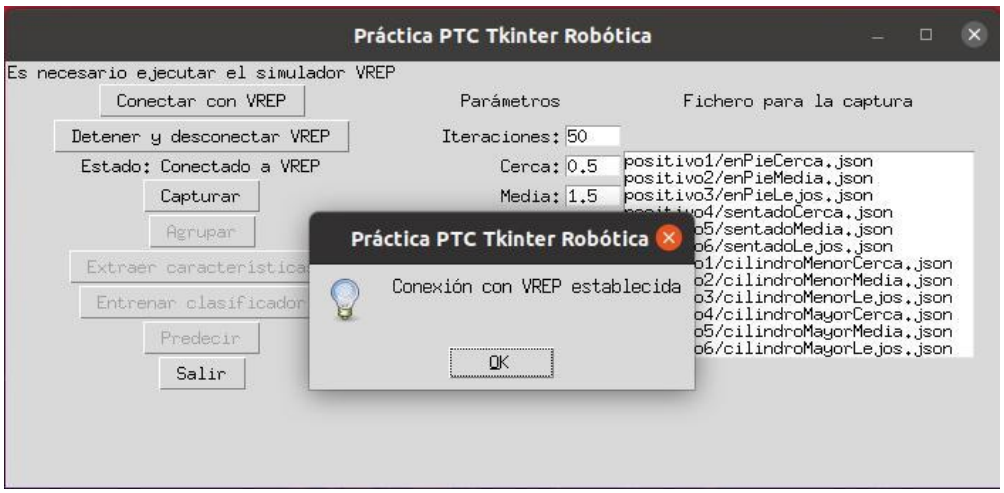


Figura 4

Además la etiqueta Estado pasa a valer “Conectado a VREP” y se activan los botones “Capturar” y “Desconectar” como se indica en la Figura 5.

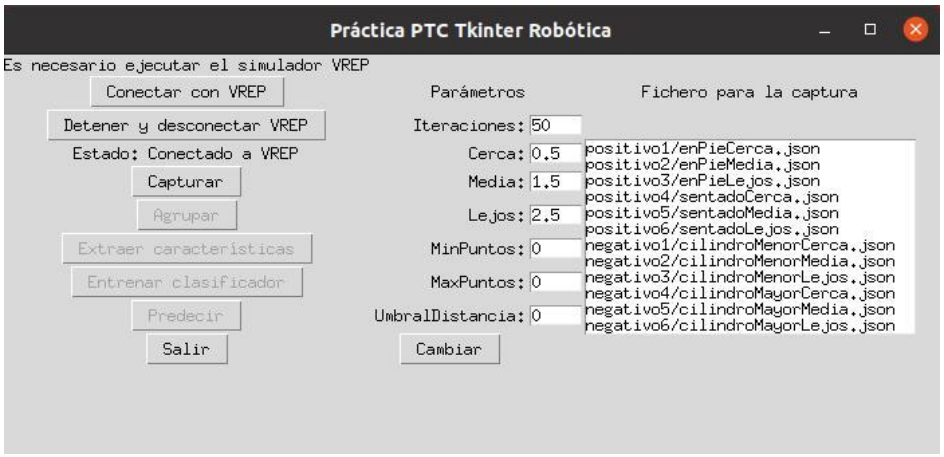


Figura 5

Botón Cambiar: Este botón permite que los parámetros adquieran los valores de las Cajas de Entrada. Para los parámetros podéis usar variables globales declaradas en un fichero parametros.py que debe ser importado por el resto de scripts que los necesite (dejo opcional usar class). Al pulsar “Cambiar” enseñaremos los valores nuevos de los parámetros por consola con print (dejo opcional enseñarlos por la interfaz de ventanas).

Botón Capturar: Debe comprobar que se ha elegido un fichero de la lista. Si no se ha elegido ningún fichero debe dar un aviso al usuario como en la Figura 5 con `messagebox.showwarning()`.

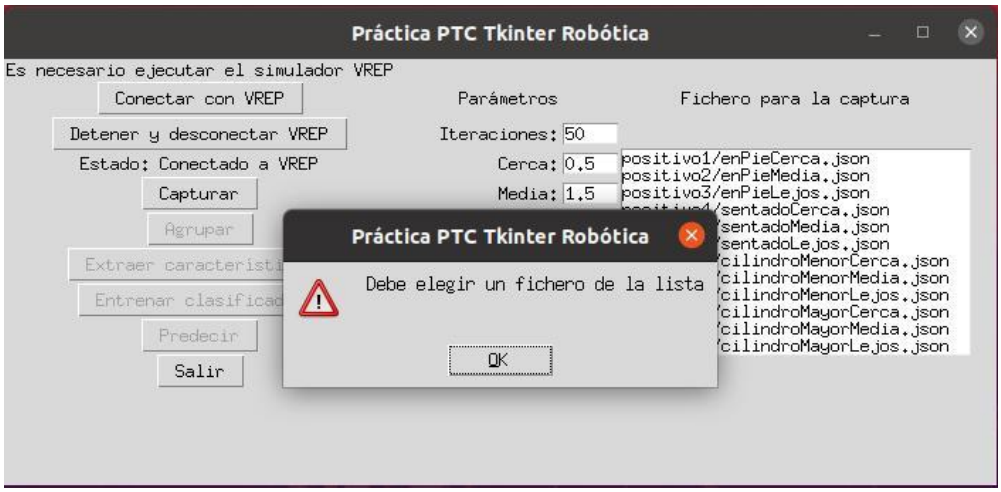


Figura 6

Si se ha elegido un fichero debe comprobar si existe ya previamente o se va a crear nuevo. En caso de que sea nuevo debe pedir confirmación al usuario como en la Figura 7 con `messagebox.askyesno()`.

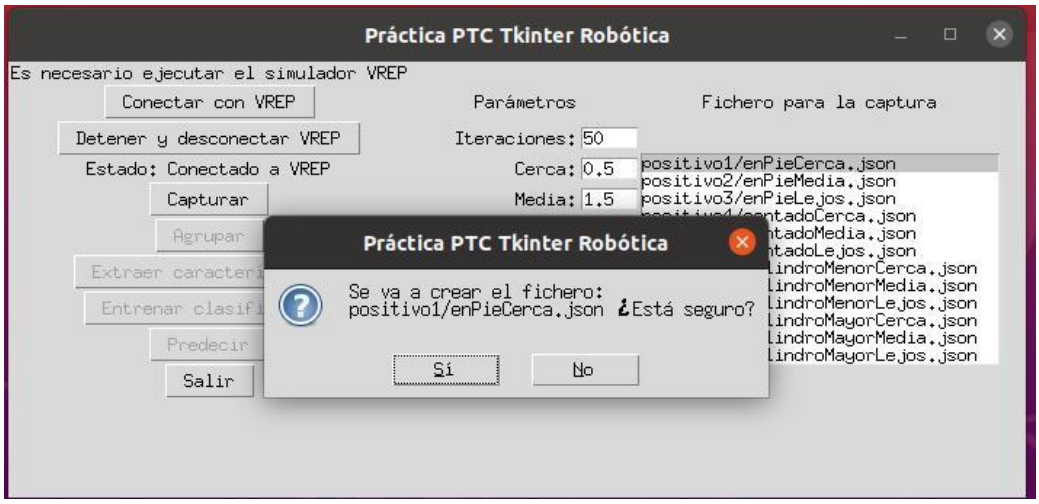


Figura 7

Si ya existe debe pedir confirmación al usuario para crearlo nuevamente lo que sobre-escribirá la versión anterior del fichero. Para ello usad `messagebox.askyesno()` como en la Figura 8.

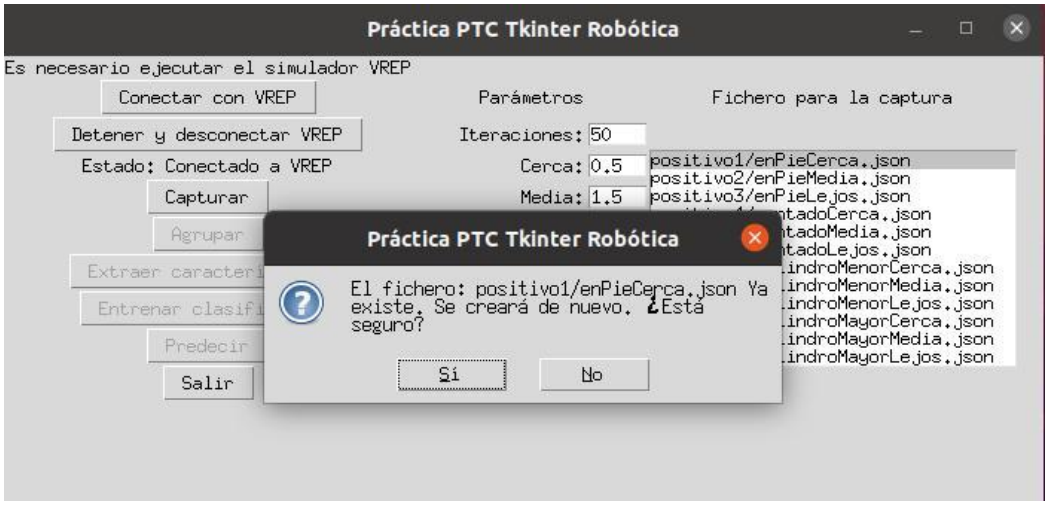


Figura 8

Usando los valores de los parámetros, se debe llamar al script Capturar.py (explicado más adelante) pasándole el nombre del fichero como parámetro. Antes de capturar el usuario debe haber cargado el escenario correspondiente en el simulador según se quiera capturar un ejemplo positivo o negativo y con las situaciones de distancia cerca, media o lejos que se comentan en el siguiente apartado. Al finalizar la captura de los 12 ficheros se puede habilitar el botón “Agrupar” que sería el siguiente paso.

Botón Agrupar: Usando los parámetros establecidos debe llamar al script Agrupar.py explicado más adelante. Genera los ficheros con los clústeres positivos y negativos tomando los ejemplos positivos y negativos de Capturar. Al finalizar habilita el botón Extraer características.

De manera semejante los siguientes botones llaman a los scripts del mismo nombre habilitando cada uno el botón siguiente al finalizar.

Botón Detener y desconectar VREP: Detiene la simulación y cierra la conexión con el simulador. La etiqueta Estado pasa a "No conectado a VREP". Los botones Capturar y Desconectar se deshabilitan y se informa al usuario con `messagebox.showinfo()` como en la Figura 9.

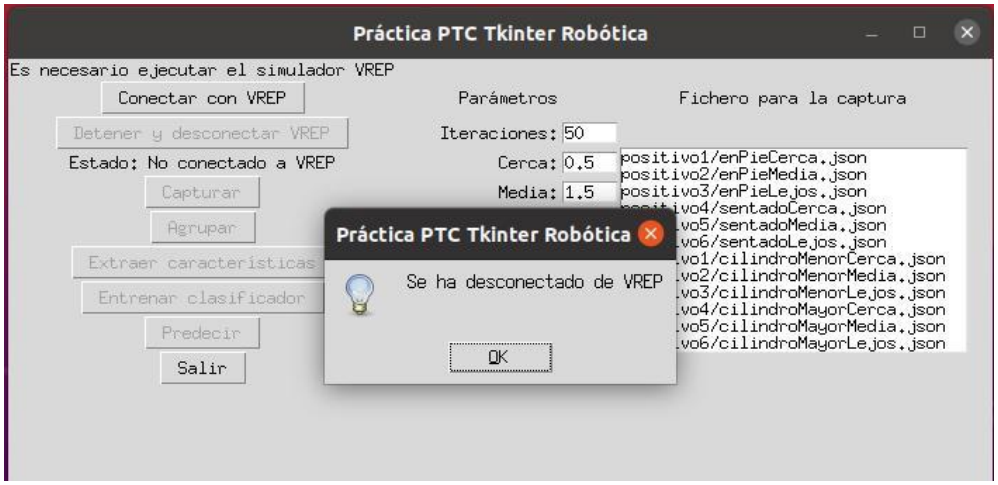


Figura 9

Botón Salir: Debe comprobar primero el estado de la conexión. Si no se ha desconectado del simulador entonces debe usar `messagebox.showwarning()` para advertir que se debe desconectar primero del simulador como en la Figura 10.

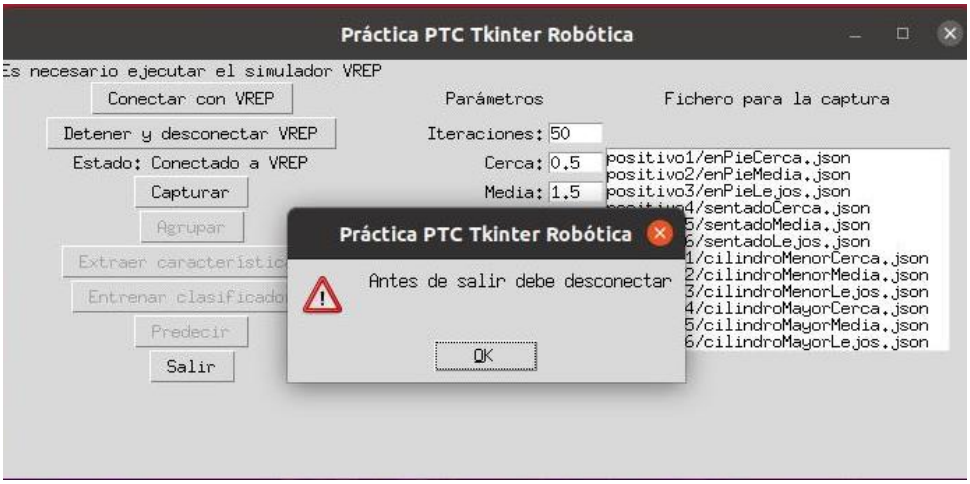


Figura 10

Si el Estado es desconectado, usad `messagebox.askyesno()` para pedir confirmación de que el usuario quiere salir realmente de la aplicación como en la Figura 11.

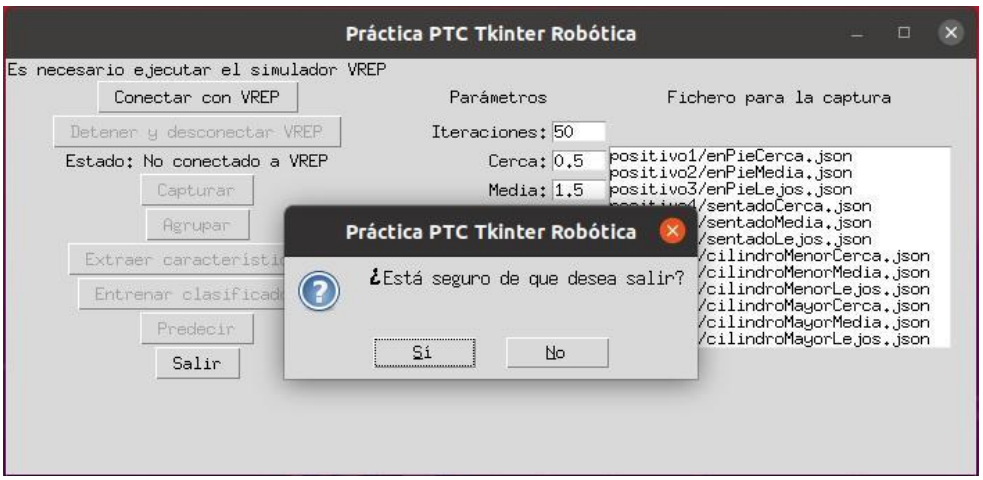


Figura 11

4.2. Captura de los datos del laser 2D en diferentes situaciones (script capturar.py)

Este script recibe el nombre del fichero (que incluye el directorio) donde se deben almacenar los valores de la captura. Luego cambia el directorio de trabajo al directorio creado y usa el nombre de fichero proporcionado para crear el fichero JSON correspondiente.

Si queremos reutilizar los ficheros de ejemplo de Prado hay que hacer lo siguiente.

1. Como ejemplos positivos usaremos personas y como negativos parejas de cilindros.
   Hay que hacer un script de Python llamado "**capturar.py**" para la captura de datos del láser en 6 situaciones para ejemplos positivos, donde solo haya personas, y otras 6 situaciones para ejemplos negativos, donde solo haya cilindros. Las 6 situaciones positivas se almacenarán en las carpetas positivos1 a positivos6 y las 6 situaciones negativas las carpetas negativos1 a negativos6. El número de lecturas en cada situación está controlado por el parámetro "**Iteraciones**=50". Entre cada iteración dejar 0.5 segundos de tiempo de espera. Hay que salvar en su carpeta correspondiente, al menos, la imagen de la primera iteración y de la última como se comenta más adelante. También es importante salvar la escena concreta (fichero .ttt) de la que estamos capturando los datos en la carpeta "positivoX" o "negativoX" correspondiente.

2. Situaciones positivas: Vamos a contemplar 3 situaciones con una persona en pie según la distancia de la persona al robot, cerca, media o lejos. Estas distancias en metros son controladas por los parámetros "**Cerca**=0.5", "**Media**=1.5" y "**Lejos**=2.5". Estando siempre el robot parado en el punto (0,0) las 3 situaciones para la persona en pie son:

Carpeta positivo1. Fichero: “enPieCerca.json”. En este caso mover el modelo de persona en un radio del intervalo “**Cerca**” <= radio < “**Media**” realizando rotaciones. Salvar también las imágenes primera y última de esta situación a su carpeta con los nombres “enPieCercaX.jpg” siendo X la iteración 1 y la última.

Carpeta positivo2. Fichero: “enPieMedia.json”. En este caso mover el modelo de persona en un radio del intervalo “**Media**” <= radio < “**Lejos**” realizando rotaciones. Salvar también las imágenes primera y última de esta situación a su carpeta con los nombres “enPieMedioX.jpg” siendo X la iteración 1 y la última.

Carpeta positivo3. Fichero: “enPieLejos.json”. En este caso mover el modelo de persona en un radio del intervalo “**Lejos**” <= radio < “**Lejos” + 1** metros realizando rotaciones. Salvar también las imágenes primera y última de esta situación a su carpeta con los nombres “enPieLejosX.jpg” siendo X la iteración 1 y la última.

Realizar otras tres situaciones positivas (4,5,6) con la misma filosofía para el modelo de persona sentada debiéndose llamar los ficheros “sentadoCerca.json” “sentadoMedia.json” y “sentadoLejos.json”. Esto ha tenido que generar las carpetas positivo1 a positivo6 y los tomaremos como datos de ejemplos positivos de piernas de personas.

3. Ahora tenemos que tomar datos de ejemplos negativos que se deben almacenar en carpetas llamadas “negativoN” (N de 1 a 6) que contendrán los ficheros json y las imágenes a semejanza de los ejemplos positivos. Para crear los ejemplos negativos, eliminamos todos los objetos de la escena de prueba y utilizaremos dos tipos de cilindros. En un caso creamos un par de cilindros de menor radio que el que tienen las piernas de nuestros modelos de personas y en un segundo caso crearemos un par de cilindros con un radio mayor. Así tenemos que generar los ficheros cilindroMenorCerca.json, cilindroMenorMedia.json y cilindroMenorLejos.json para el par de cilindros menores y los ficheros cilindroMayorCerca.json, cilindroMayorMedia.json y cilindroMayorLejos.json para el par de cilindros mayores usando las 3 zonas de distancias anteriormente definidas.

   Esto ha tenido que generar las carpetas negativo1 a negativo6 y los tomaremos como datos de ejemplos negativos de piernas de personas.

4.3. Agrupar los puntos x,y en clústeres (script agrupar.py)

Vamos a definir un clúster como un conjunto de puntos cercanos y que pueden ser candidatos a formar una única pierna. Primero hay que establecer el número mínimo y máximo de puntos que puede tener un clúster. Hay que definirlos como parámetros llamados “**MinPuntos**” y “**MaxPuntos**” y según los datos de la experimentación, establecer los más apropiados. Tened en cuenta que cuando las personas están más lejos, los

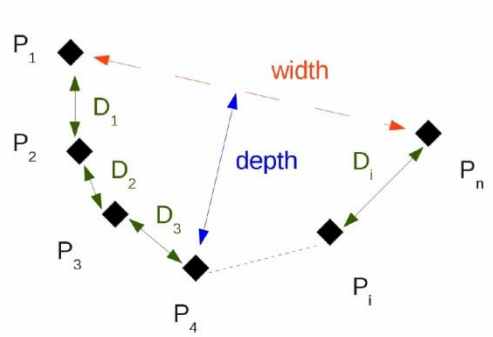
clústeres pueden estar formados por menos puntos. Para agrupar los puntos de los ficheros anteriores en clústeres hay que utilizar el algoritmo de "agrupación por distancia de salto". Se crea un clúster como una lista de puntos, se mete el primer punto, ahora se analiza el segundo punto para ver si se incorpora al clúster actual. Si la distancia del punto anterior al que se está analizando es menor a un umbral (parámetro "**umbralDistancia"**), entonces se incorpora el punto analizado al clúster y se pasa al siguiente. El cluster actual puede finalizar por dos motivos: que el punto analizado está más lejos del umbral de distancia o bien que se ha superado el punto máximo de puntos (MaxPuntos) que puede tener un cluster. Para aceptar un grupo de puntos como clúster válido tened en cuenta que debe tener un número mínimo de puntos (MinPuntos) además de cumplir con el valor máximo (MaxPuntos).

A partir de las carpetas de ejemplos positivos y negativos se deben generar solo dos ficheros, uno que llamaremos clustersPiernas.json y otro clustersNoPiernas.json con el siguiente formato de cada línea usando diccionarios y el módulo json:

{"numero_cluster":i, "numero_puntos":j, "puntosX":[lista],"puntosY":[lista]}

4.4. Convertir los clústeres en características geométricas (script caracteristicas.py)

De cada clúster formado por n puntos {p1,p2, …, pn} vamos a tomar tres características de tipo geométrico, que son las siguientes: **perímetro** es la suma de las distancias D1 a Dn-1, **profundidad** (depth) es el máximo de las distancias de la recta P1Pn a los puntos P1 a Pn y **anchura** (width) es la distancia de P1 a Pn. Estos cálculos se hacen usando las fórmulas de geometría básica, de cálculo de distancia euclídea entre dos puntos, del cálculo de la recta que pasa por dos puntos y de la distancia de un punto a una recta. Ver la siguiente figura para mayor claridad.



A partir de los ficheros de clústeres anteriores se deben generar ahora otros dos ficheros llamados caracteristicasPiernas.dat y caracteristicasNoPiernas.dat con el siguiente formato por línea usando diccionarios y el módulo json:

{"numero_cluster":i, "perímetro":p,"profundidad":d,"anchura":a, "esPierna":(1=true, 0=false)}

En el primer fichero todos los clústeres serán pierna y en el segundo no. Por tanto ya tenemos ejemplos positivos y negativos para el clasificador teniendo en cuenta que la clase 0 significará no pierna y la clase 1 significará que hay pierna.

A partir de estos dos ficheros de características hay que generar un fichero "piernasDataset.csv" que contendrá primero todas los ejemplos de clase 0, (negativos) y luego todos lo de clase 1 (positivos). No debe tener cabecera y el formato será:

perímetro, profundidad, anchura, clase (0 | 1)

por ejemplo, una línea podría ser: 0.23, 0.12, 0.15, 0

(ver fichero iris.data de Prado como ejemplo)

4.5. Entrenar un clasificador binario utilizando Support Vector Machine (SVM) y Scikit-Learn (script clasificarSVM.py)

El módulo Scikit-learn se explica en clase de teoría y también se puede obtener información en https://scikit-learn.org/stable/

Para comprender mejor el funcionamiento del proceso de entrenamiento, predicción, evaluación del clasificador y validación cruzada tenéis disponible un ejemplo de clasificación, usando SVM, del problema de clasificar las tres especies de flor Iris. El script está en Prado: "ejemploSklearnSVM.py" junto al fichero "iris.data" que contiene los ejemplos.

Como resultado del proceso de entrenamiento se debe obtener un clasificador y evaluar dicho clasificador sobre el conjunto de prueba generando la matriz de confusión y los valores asociados de rendimiento (como tenéis en el ejemplo "ejemploSklearnSVM.py"). Probar con SVM con los kernels 'rbf' , 'linear' y 'poly' para ver si en este problema hay diferencias. Se deben mostrar los resultados de los tres kernels y comparar su funcionamiento para determinar cuál elegir. La manera de realizar la comparación así como la búsqueda de los mejores parámetros serán elementos a tener en cuenta en la evaluación de la práctica.

4.6. Utilizar el clasificador con datos nuevos a partir del simulador (script predecir.py)

Una vez que tenemos el clasificador entrenado, se puede usar como predictor para identificar las piernas de las personas en el entorno simulado. Para ello se debe utilizar un escenario simulado llamado "escenaTest.ttt" disponible en Prado que incluye una persona en pie, otra sentada, un par de cilindros de mayor radio que las piernas y otro par de menor radio. Es decir, al menos deben detectarse esos 4 objetos por parte del detector. Este

escenario de test y los resultados de este apartado deben almacenarse en la carpeta "predicción". Ahora hay que crear un script llamado "predecir.py" que:

- Reciba los datos de laser
- Los convierta en clústeres
- Genere las tres características de cada clúster
- Utilice el predictor para cada clúster a partir de sus características
- Dibuje en color rojo aquellos clústeres que sean piernas según el predictor y en azul aquellos que no.

Cuando se detecte dos clústeres "cercanos" que pertenezcan a un mismo objeto o persona, se debe calcular el centroide de ambos clústeres y pintar un punto rojo (o azul) en el punto medio del segmento que une ambos centroides, entendiendo que ese punto representa la posición de la persona/objeto detectada/o. Este gráfico se almacena en fichero .jpg y se salva en la carpeta "predicción".

Plazo de entrega en Prado hasta el 19 de diciembre.

Para la entrega todos los scripts y carpetas deben estar en un directorio llamado "practicaTkinterRobotica.zip" que debe ser comprimido en un fichero .zip o rar y entregado en Prado. Comprobad que dicha carpeta contiene: todos los ficheros necesarios para el cliente de vrep, las escenas ejemplo que hayáis usado para la captura de datos dentro de sus respectivas carpetas, junto a los 12 ficheros de positivos y negativos de los datos leídos y la carpeta predicción con el gráfico resultado y la escena de test. Que no falten vuestros scripts de mainInterfaz.py, capturar.py, agrupar.py, caracteristicas.py, clasificarSVM.py y predecir.py.

Además debe entregarse una memoria en PDF de la práctica con el nombre del estudiante, donde se diga cuantos apartados se han realizado, que explique de forma razonada qué parámetros se han elegido, incluya imágenes del simulador mostrando el funcionamiento del sistema de captura, de los resultados obtenidos por la SVM en fase de aprendizaje y en la fase posterior de predicción sobre la escena de test (incluir imágenes en la memoria).

IMPORTANTE: Añadir un botón DEBUG debajo del botón SALIR, de manera que al pulsar ese botón todos los botones queden activados por motivos de depuración del programa. Al pulsar de nuevo deben volver a su estado anterior.