PTC

Programación Técnica y Científica

Trabajo 2: Tkinter y robótica

Autor: Jesus Medina Taboada

Correo: jemeta@correo.ugr.es

Profesor: Eugenio Aguirre Molina

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Curso 2020 - 2021

Índice general

0.1. Introducción

En la Programación Técnica y Científica la tarea de captura de datos en tiempo real, su correcto almacenamiento y su posterior tratamientos es algo esencial y de gran importancia, por ello en esta práctica trataremos estos puntos. Trabajaremos en Python como venimos haciendo en toda la asignatura ya que nos brinda bibliotecas muy potentes y útiles para estas tareas, en concreto usaremos algunas como tkinter, matplotlib, json, os, glob .

Para generar los datos a tratar usaremos un simulador muy popular en el mundo de la robótica, denominado V-Rep (actualmente llamado Coppelia-Sim). En concreto usaremos la versión V-REP PRO EDU 3.6.2 gratuita y trabajaremos en sistema operativo Windows.

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.

Capítulo 1

Implementación de la interfaz gráfica con tkinter

Para empezar crearemos una interfaz gráfica(GUI) para controlar las funciones que queremos implementar con relación a los datos que el simulador genera. Para ello usaremos el módulo tkinter que nos basta para crear esta capa de abstracción y llevar a cabo de forma más intuitiva nuestro cometido.

La interfaz se divide en 3 parte, izquierda,central y derecha.
En la **parte izquierda** implementamos una serie de botones que activarán funciones sencillas y scripts que realizan procesos más complejos, la mayoría de ellos necesitan una serie de condiciones para activarse y al activarse cuentan con un despliegue de ventanas informativas,de aviso y de error si se dan los casos.

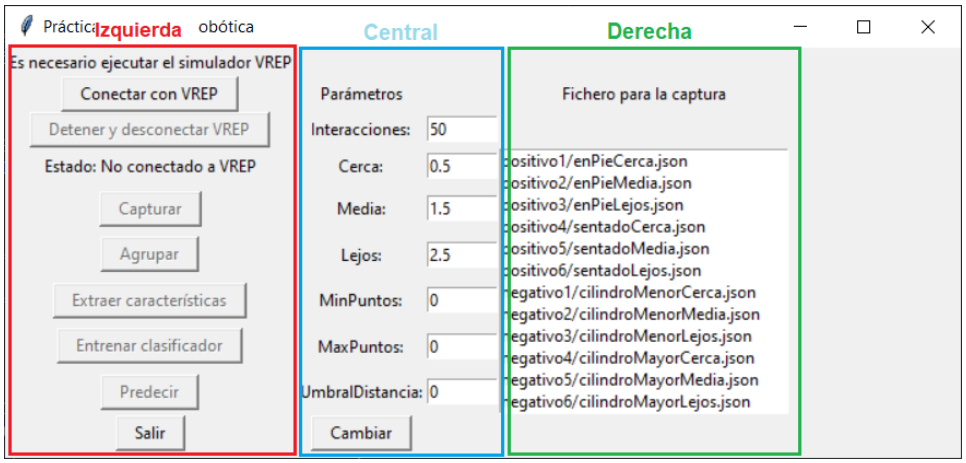
- **Conectar:** botón que nos conecta con el simulador VREP(el simulador debe estar ejecutándose).
- **Detener y desconectar:** detiene la simulación y nos desconecta del simulador.
- **Capturar:** activa función de capturar dado un fichero seleccionado de la parte derecha.
- **Agrupar:** activa función agrupar (el botón está disponible cuando se han capturado los 12 ficheros principales).
- **Extraer características:** activa función extracción de características(está disponible una vez se realiza la agrupación).

- **Entrenar clasificador:** activa el entrenamiento del clasificador(está disponible una vez se realiza la extracción de características).
- **Predecir:** activa la función de predicción(está disponible una vez se realiza el entrenamiento del clasificador).
- **Salir:** cierra la interfaz(debemos estar desconectados).

Más tarde profundizaremos en lo que realiza cada una de estas funciones.

En la **parte central** muestra 7 casillas de entrada de datos que son parámetros relacionados con el simulador, tienen unos valores por defecto pero podemos introducir otros y guardarlos usando el botón 'Cambiar' al en la parte inferior. Están relacionados con la función de capturar los ficheros.

En la **parte derecha** nos encontramos una ventana con un listado de rutas directorio/fichero.json, son con los ficheros que trabajaremos para tratar con los datos obtenidos, nos permite seleccionar de entre la lista un elemento para poderlo capturar con el botón homónimo.



IMPORTANTE: en la entrega se adjunta un archivo txt con las instrucciones para la reproducción de los resultados:
1. Se puede realizar todo el funcionamiento de la aplicación desde cero(teniendo que tomar nuevos datos manualmente).
2. Realizar el resto de funciones de la práctica partiendo de los datos que ya he tomado yo.

Capítulo 2

Captura de los datos del láser 2D en diferentes situaciones

Lo primero que necesitamos para empezar a trabajar es obtener un conjunto de datos adecuado.
Como hemos explicado el objetivo es conseguir diferenciar entre 2 clases: piernas y cilindros.
Para conseguir nuestro objetivo necesitamos obtener datos de ambas clases, datos positivos(piernas) y datos negativos(cilindros).
Para ello mandaremos ejecutar el script capturar, que hará uso del simulador Vrep con el que iremos generando una serie de escenas. De ellas obtendremos los puntos que identifican cada objeto e iremos almacenando estos datos.

En esencia usaremos 4 tipos de escenas, 2 positivas(con personas de pie y sentadas) y 2 negativas(con cilindros de circunferencia menos y mayor).
Los radios que he escogido para los cilindros son de 0.05 para el menor(un poco inferior al de las piernas) y 0.2 para los cilindros mayores, que es un poco mayor al de las piernas.

Aparte tomaremos 3 grupos de datos de cada posición moviendo el objeto en 3 rangos de distancia diferentes respecto del observador de la escena(robot). Estos rangos serán cerca(0.5-1.5 metros), media(1.5-2.5 metros) y lejos(2.5-3.5 metros).
Todo esto nos da un total de 12 conjuntos de datos que recopilaremos en un archivo con extensión '.json'.

Figura 2.1: De pie

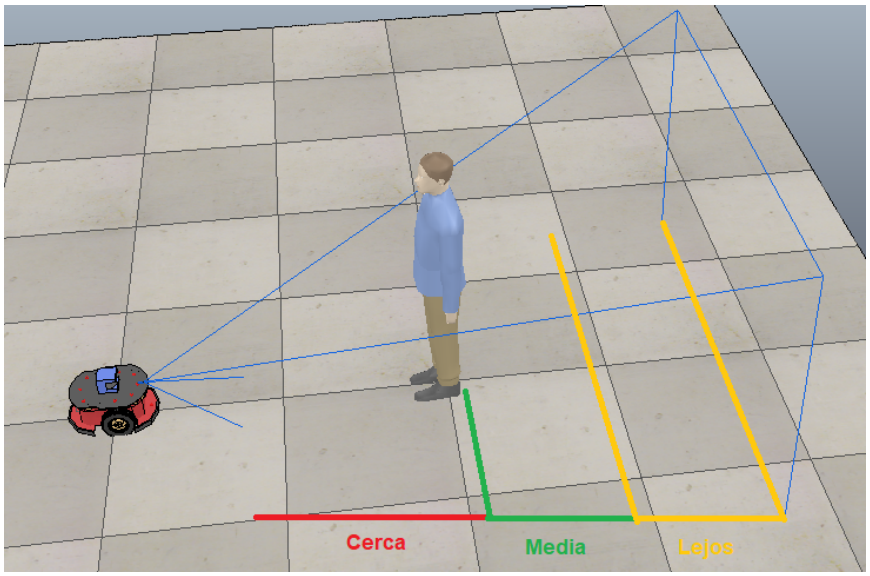


Figura 2.2: Sentado

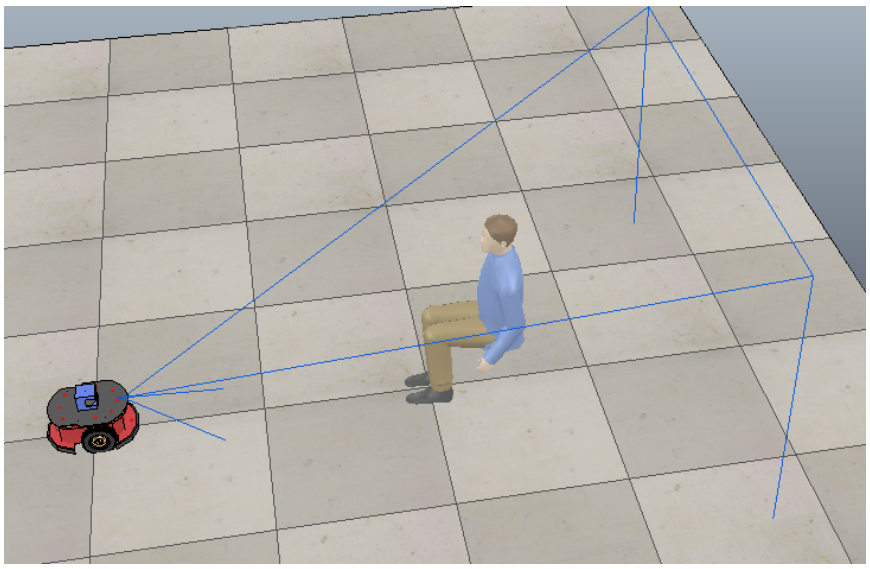


Figura 2.3: Cilindro Menor

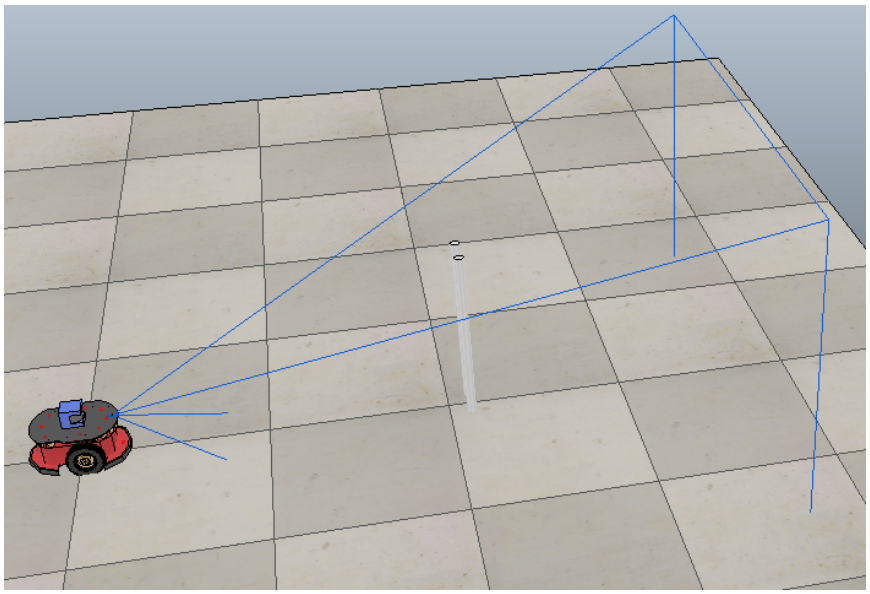
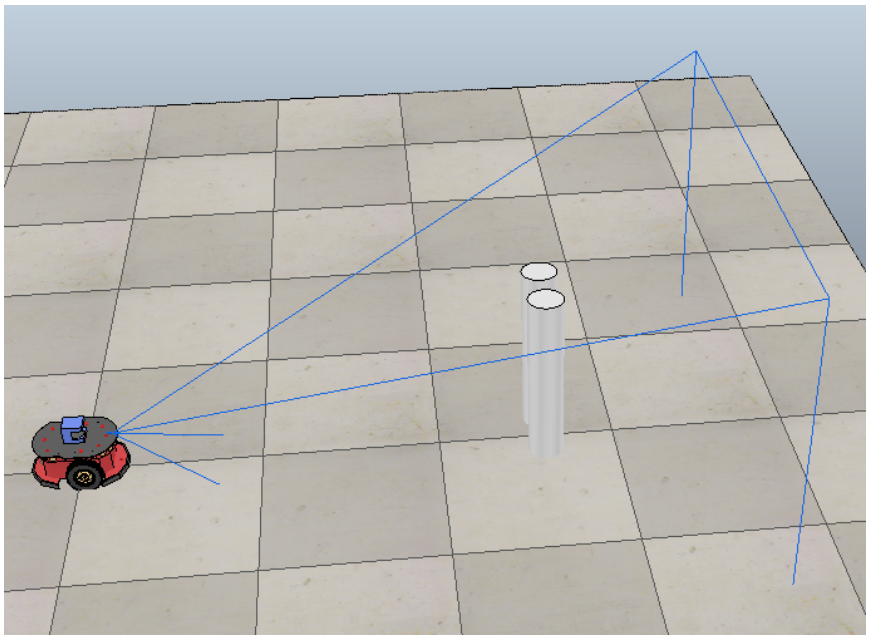


Figura 2.4: Cilindro Mayor



El procedimiento para realizar las capturas es el siguiente:

1. Elegimos la escena adecuada(depie,sentada,cilindromenor,cilindromayor) en el simulador y colocamos el objeto en el rango deseado(cerca,media,lejos).
2. Iniciamos la simulación y realizamos la conexión desde la interfaz.
3. Seleccionamos en la lista de la izquierda de la interfaz el fichero que queremos capturar.
4. Pulsamos botón capturar y vamos moviendo la figura en el rango de distancia deseado y rotándolo sobre su eje para obtener la mayor información que se pueda.

El tema de la rotación es muy importante porque realmente es el punto clave que nos ayudará a diferenciar luego entre la clase cilindro y pierna, ya que la forma de las piernas varía según la rotemos mientras que los cilindros permanecen iguales.

De esta forma iremos capturando los datos de las 12 escenas que hemos planteado, se crearán 12 directorios en los que se guardarán los datos obtenidos en un fichero json y 2 imágenes, la primera toma de datos y la última.

Capítulo 3

Agrupar los puntos x,y en clústeres

A continuación y una vez hemos recolectado todos los datos correctamente pasaremos a agrupar estos datos en clústers, para ello clicaremos el botón agrupar que ya estará habilitado.
Definiremos un clúster como un conjunto de puntos cercanos que capta el sensor y pueden o no representar una pierna.
Para esto necesitamos establecer 3 parámetros: **MinPuntos**, **MaxPuntos** y **umbralDistancia**.
Los estableceremos de forma empírica y teniendo en cuenta que si una persona está más lejos en la escena el clúster estará formado por menos puntos.

Para hacer la agrupación en clústers usaremos el algoritmo de agrupación por distancia de salto. Este consiste en crear una lista de puntos metiendo un primer punto para después analizará 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 asi sucesivamente.

Hay 2 criterios de parada en la agrupación de un clúster, que el punto analizado está más lejos del umbral de distancia o que se ha superado el punto máximo de puntos (MaxPuntos) que puede tener un cluster.
Por lo tanto para obtener un clúster válido tendrá que tener más de MinPuntos y no más de MaxPuntos.

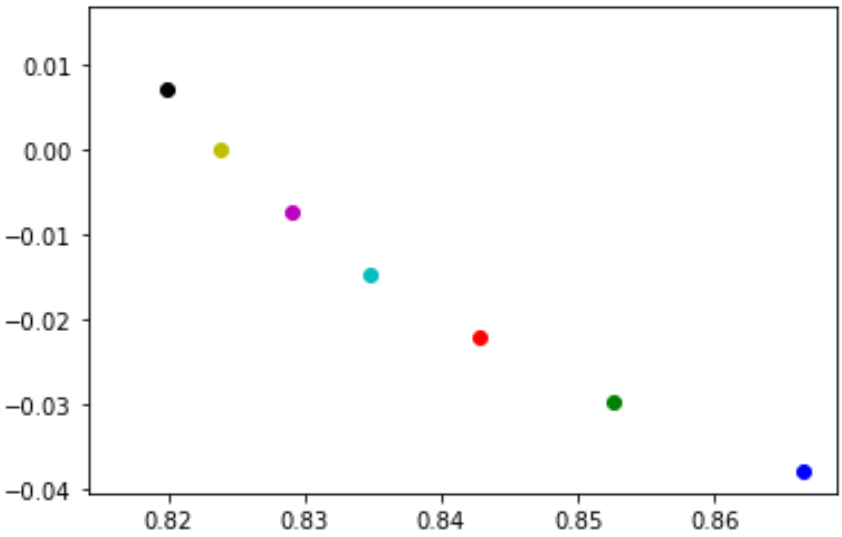
PARÁMETROS

- **MinPuntos:** tras varias pruebas he puesto un número de puntos mínimo de **3**, ya que en la escena de cilindros menores, no hay apenas superficie donde el láser tome puntos y si cogemos el valor 4 no se agrupa ningún clúster en esta escena.
- **MaxPuntos:** he optado por fijar el valor a **50** ya que las escenas de cilindros grandes muestran muchos puntos y si cogemos número inferiores a veces dividen un cilindro mayor en dos clústers diferentes lo que nos perjudica. En cuanto a si puede ser mayor este valor la respuesta es si. Ya que no influye apenas ya tenemos el criterio de parada del umbral que además es mucho más eficiente.
- **umbralDistancia:** este es a mi parecer el parámetro más importante y el que más me ha costado fijar con precisión. Probé valores entre 0.1 y 1 y opté por darle un **0.04**, es el valor que me da mejores clústers obteniendo buenos resultado en todas las escenas.

Se puede descomentar la visualización de los clústers que se van creando para verlos, pero no son representativos y son muchos, en el apartado siguiente si trabajaremos con ellos y empezaremos a ver patrones representativos.

Un ejemplo de los clústers que estamos formando seria el siguiente:

Figura 3.1: Clúster positivo



Estos parámetros los introduciremos a través de los campos de nuestra interfaz gráfica, por defecto vienen los 3 dados con el valor 0, por lo que si queremos cambiarlos, necesitamos cambiar los valores y muy importante pulsar el botón **'Cambiar'**.

Todo esto lo harémos a partir de los archivos generados en las carpetas positivas y negativas obteniendo finalmente solo 2 nuevos archivos clusters-Piernas.json y 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]}

Capítulo 4

Convertir los clústeres en características geométricas

En este apartado nuestro objetivo es extraer carácterísitcas de nuestros clústers para posteriormente entrenar un clasificador.
Ejecutaremos el script caracteristicas usando el botón 'Extraer características'. Se leen los ficheros clustersPiernas.json y clustersNoPiernas.json y de cada clúster formado por n puntos p1,p2, . . . , pn tomaremos 3 características geométricas:

- **Perímetro:** suma de las distancias D1 a Dn-1, es decir, sumando la distancia euclídea de todos los puntos del clúster.
- **Profundidad:** máximo de las distancias de la recta P1Pn a los puntos P1 a Pn, que es el máximo de las distancias de los puntos del clúster a la recta que une el primer y último punto.
- **Anchura:** distancia de P1 a Pn,lo que es igual a la distancia euclídea entre el primer y último punto del clúster. Para ello calcularemos la recta en forma general Ax+By+C=0 que pasa por el primer y último punto del clúster, luego se calcula la distancia de cada punto del clúster a esa recta y finalmente nos quedamos con el máximo de esas distancias.

Finalmente generamos 2 ficheros nuevos (caracteristicasPiernas y caracteristicasNoPiernas) con las carácterísticas obtenidas.
Lo ficheros siguen el formato:
{"numero_cluster":i, "perímetro":p,"profundidad":d,"anchura":a, "esPierna":(1=true, 0=false)}

En el primer fichero tendremos características de clusters que eran piernas(positivos) y en el segundo de los clusters no-piernas(negativos) por lo que ya tenemos características que nos ayuden a diferencias entre las dos clases.

Finalmente a partir de estos 2 últimos ficheros generaremos el fichero "piernasDataset.csv" que contendrá primero todas los ejemplos de clase 0, (negativos) y luego todos lo de clase 1 (positivos).
El formato será:
perímetro, profundidad, anchura, clase (0 - 1)

Capítulo 5

Entrenar un clasificador binario utilizando Support Vector Machine (SVM) y Scikit-Learn

Ahora entrenaremos un clasificador binario Support Vector Machine haciendo uso del módulo Scikit-learn. Una vez obtenido evaluaremos el clasificador sobre un conjunto de prueba y generando la matriz de confusión y los valores asociados al rendimiento como en el ejemplo que se nos facilitaba.

Para ser exactos una vez entrenado nuestro clasificador SVM lo probaremos con diferentes kernels (linear,poly y rbf) para comparar y determinar si existen diferencias notorias para este problema.

Lanzo pues el script pulsando el botón 'Entrenar clasificador', el cual usa las características geométricas de los clústers para entrenar un SVM que decida si un clúster corresponde a una pierna o no.
Primero, cargo el dataset piernasDataset.csv usando la función np.genfromtxt. Después, separo los datos en sus atributos X (características geométricas) y etiquetas y (clase de cada clúster).

Acto seguido, uso la función de scikit learn train_test_split para dividir aleatoriamente los datos (X, Y) en el conjunto de entrenamiento y en el conjunto de test (con un 20 % de los datos). Para que ambos conjuntos de train y test tengan más o menos el mismo porcentaje de clases (pierna y no_pierna), uso el parámetro stratify.

Tras dividir el dataset en train y test, pruebo distintos hiperparámetros de SVM para decidir cuál funciona mejor. Los distintos kernels probados son: lineal, polinómico (poly) y de base radial (rbf). Para cada tipo de kernel, uso la clase de scikit_learn GridSearchCV para elegir, de entre varios posibles valores de hiperparámetros establecidos a mano, qué combinación funciona mejor para cada tipo de kernel.

Para los tres tipos de kernels usé esta técnica para escoger entre varios valores de C, que es el parámetro de regularización L2. Los valores probados han sido: 0.1, 1, 10, 100, 1000, 10000. Además, para el kernel poly también probé varios posibles grados del polinomio (parámetro degree): 2, 3, 4 y 5. De forma análoga, para el kernel rbf probé varios valores del hiperparámetro gamma: 0.001, 0.005, 0.01, 0.1. Este parámetro controla "cuánta influencia un solo ejemplo de entrenamiento tiene", según aparece en la documentación de scikit_learn.

Siguiendo este esquema, voy entrenando una SVM diferente para cada tipo de kernel. Primero uso GridSearchCV sobre el conjunto de entrenamiento para escoger los valores óptimos de hiperparámetros según el tipo de kernel. Esta clase entrena una SVM del kernel correspondiente para cada combinación posible de los valores de hiperparámetros ya comentados sobre el conjunto de training y, de todos ellos, se queda con el conjunto de valores para los que la SVM tiene una mayor media de accuracy.

Una vez obtenida la mejor combinación de hiperparámetros para ese tipo de kernel, creo una SVM con esos hiperparámetros y la entreno sobre el conjunto de training completo (X_train). Tras ello, uso la SVM entrenada para predecir las clases y_pred de los clústeres del conjunto de test (X_test).

Después de haber obtenido las predicciones y_pred, las comparo con los valores correctos de las clases (y) para obtener diversas métricas en el conjunto de test y, así, poder decidir cuál de los 3 kernels es el mejor para este problema.
Las métricas obtenidas son:
-**Accuracy sobre test** porcentaje de ejemplos (clústeres) correctamente clasificados.
-**Métricas dadas por clasification_report** que son precision (TP / (TP + FP)), recall (TP /(TP + FN)) y el f1-score (2 * (precision * recall) / (precision + recall)).
-**Matriz de confusión** que muestra el número de TP, TN, FN y FP en test.

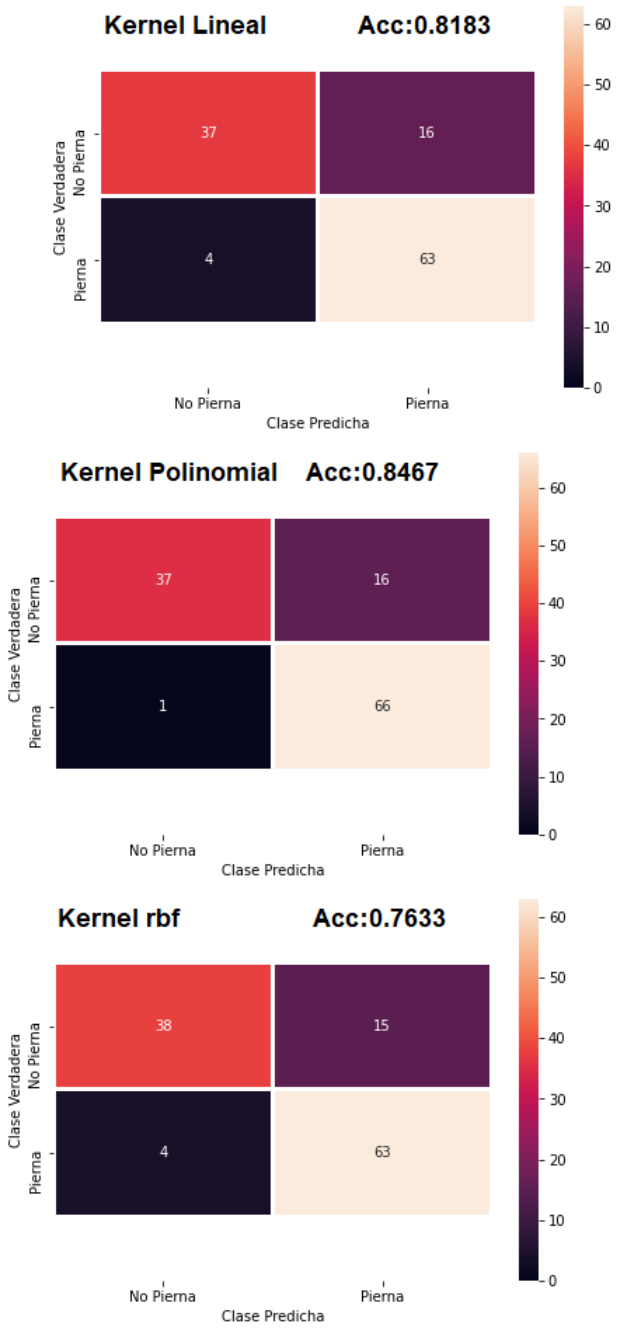
Además, para asegurarme de que el conjunto de test usado para obtener todas estas métricas es representantivo, uso la función cross_val_score que realiza validación cruzada sobre todo el conjunto de datos (X) y, así, obtengo la media de las test accuracy de 5 distintas parejas train, test que me sirve para obtener un intervalo de confianza (al 95 %) de la media del test accuracy.

Así, si el accuracy del conjunto de test usado para obtener todas las métricas previamente mencionadas se encuentra dentro de este intervalo, puedo estar bastante seguro de que el conjunto de test usado (y, por tanto, las métricas) es representativo y que, por ello, puedo decidir cuál es la mejor SVM en función de estas métricas.
Esto se cumple para las 6 SVM entrenadas por lo que, en todos los casos, puedo fiarme de las métricas obtenidas sobre el conjunto de test usado.
Además de realizar todo este proceso con los 3 distintos tipos de kernels, repetí el mismo proceso pero normalizando antes el conjunto de datos X restándole a cada una de las 3 características su media y dividiendo entre su desviación típica.

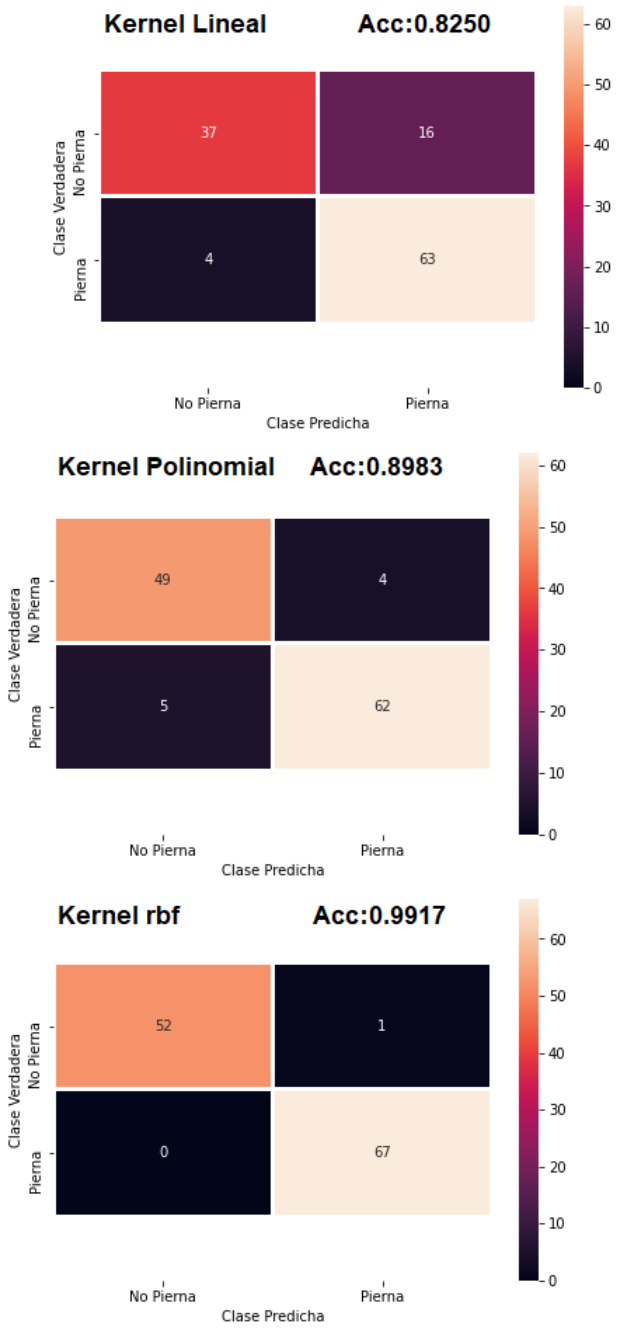
La razón de realizar esto es que, así, todas las características tienen media 0 y desviación típica 1, lo que suele facilitar el entrenamiento de los modelos de Machine Learning y, por tanto, hacer que los resultados obtenidos sean mejores. Así, al final, entrené y obtuve las métricas de 6 SVM diferentes: 3 con los datos sin normalizar y 3 con los datos normalizados (para comprobar si se obtenían mejores resultados al normalizar los datos) y, para cada tipo de datos, cada una de las 3 SVM con un kernel diferente: lineal, poly o rbf.

Como de cada una de las 6 SVM se obtienen muchas métricas, solo voy a mostrar para cada una de ellas la accuracy obtenida usando la función cross_val_score (la media de las accuracy del 5-CV sobre todo el dataset) y la matriz de confusión, obtenida usando la función heatmap del módulo seaborn, (el resto de métricas son impresas por pantalla por el script clasificarSVM.py)

Dataset sin normalizar (dataset original):



Dataset normalizado:



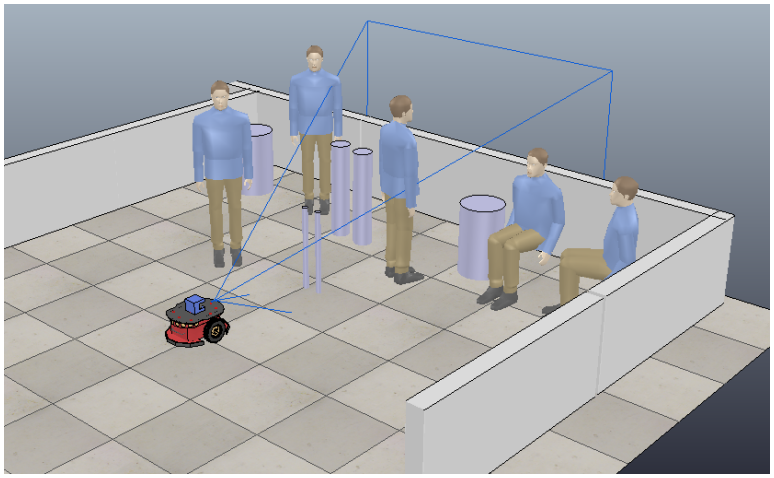
Para los datos sin normalizar, los 3 tipos de kernel obtienen más o menos el mismo resultado (alrededor de un 80 % de accuracy), que es bastante mediocre teniendo en cuenta que solo hay dos clases entre las que elegir.
Al normalizar los datos, para el kernel lineal se obtiene un resultado igual de malo. Sin embargo, para los otros dos kernels (poly y rbf), los resultados mejoran drásticamente: para poly se obtiene una accuracy de casi el 90 % y, para rbf, una accuracy del 0.9917 %, o lo que es lo mismo, casi el 100 %.

Así, el mejor modelo es la SVM con los datos normalizados y kernel de base radial (rbf), ya que es el que tiene mejor accuracy y, basándonos en su matriz de confusión, vemos que es el modelo que tiene tanto menos falsos positivos (1 solo) como menos falsos negativos.
Además, los mejores valores de hiperparámetros obtenidos para esta SVM con GridSearchCV son: C=1000 y gamma=0,1. Así, de entre todos los modelos entrenados, guardo este modelo usando joblib.dump (tal y como recomiendan en la documentación de scikit—learn, en vez de usar pickle) en un archivo de nombre clasificador.pkl.

Capítulo 6

Utilizar el clasificador con datos nuevos a partir del simulador

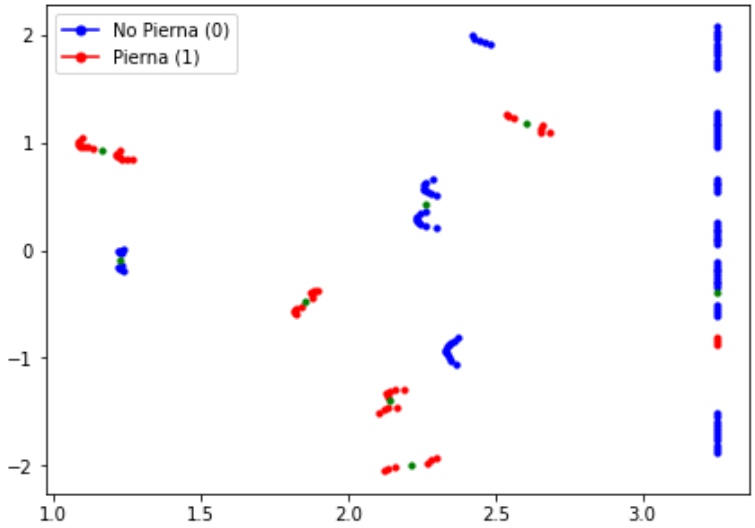
Por último usaremos nuestro clasificador ya entrenado 'clasificador.plk' para predecir un nuevo conjuntos de datos de una nueva escena, y discernir entre si los objetos que hay son piernas o no-piernas.
Lanzaremos el script predecir clicando en el botón 'Predecir', pero antes deberemos tener abierto el simulador VRep con la nueva escena llamada "escenaTest.ttt" en la que nos encontramos una persona de pie, otra sentada, un par de cilindros de mayor radio que las piernas y otro par de menor radio.



Utilizar el clasificador con datos nuevos a partir del simulador

Una vez esté todo listo el script realizará todas las taras que hemos hecho a la hora de entrenar el clasificador, reulitizando las funciones de los anteriores scripts, esto es, recibir los datos del láser(capturar),convertirlos en clústers(agrupar),extraer las características geométricas de los clústers(características) y utilizar el clasificador con estas nuevas características.

Finalmente dibujaremos en color rojo los clústers que sean piernas y en azul los clústers considerados como no-piernas.
Cuando se detectemos dos clústeres "cercanos" que pertenezcan a un mismo objeto o persona, calcularemos el centroide de ambos clústeres y pintamos un punto verde en el punto medio del segmento que une ambos centroides, entendiendo que ese punto representa la posición de la persona/objeto detectada/o, para acabar guardaremos este gráfico en la carpeta predicción.
En la práctica se pedía que se pintase el centroide del mismo color que la clase, es decir, o azul o rojo, pero al obtener los resultados he visto que no se distinguia el centroide y no se podía apreciar realmente. Por ello he decidido pintarlos de verde.



Comparando la gráfica con los objetos de la escena, se puede apreciar cómo el resultado obtenido es muy bueno, casi perfecto.
Conseguimos clasificar correctamente todos los objetos de la escena menos una pequeña confusión, un trozo de pared.

Si comparamos en la escena hay 5 personas con piernas y se detectan en la gráfica correctamente, lo mismo con los cilindros que se clasifican como no-personas.

Utilizar el clasificador con datos nuevos a partir del simulador

Además nuestro clasificador, que jamás se ha entrenado con paredes(un elemento nuevo en la escena), es capaz de clasificarlo como no-piernas.
A pesar de saber clasificar las paredes como no-piernas casi a la perfección, vemos que el único error en nuestra detección es un pequeño trozo de pared a la derecha que clasifica como 'pierna'.
Esto se podría solucionar añadiendo algunas paredes a las escenas que usamos para la toma de datos, de esta forma conseguiríamos un 100 % de acierto.

Salvando este pequeño detalle hemos conseguido un clasificador realmente bueno cercano al 100 % de accurate tanto en el train como en el test.