Programación Técnica y Científica. Grado Ingeniería Informática (UGR)

Segunda práctica evaluable (2 puntos)

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 matplotlib, json, os, glob junto a un simulador muy popular en el mundo de la robótica, denominado V-Rep. 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 el módulo scikit-learn de pyhton utilizaremos un algoritmo de Machine Learning para entrenar un clasificador que nos permita identificar, en los datos laser 2D, las piernas de las personas. Finalmente 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 el módulo scikit-learn y el módulo opencv. El módulo scikit-learn 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 decsai tenéis en la carpeta “ayudas” 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 V-Rep

El simulador se puede descargar de la página de la empresa propietaria (http://www.coppeliarobotics.com/) que distribuye una versión “free” para uso educativo llamada “V-Rep Pro Edu". Actualmente la última versión es la 3.6 pero desde esa página, al final, se da la opción de bajarse versiones anteriores de V-Rep. Para esta práctica todos vamos a descargar la versión **V-REP PRO EDU 3.4.0** para Linux o Windows, según el caso. En Windows seguir las instrucciones del instalador.

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 `./vrep.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: `./vrep.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 khepera3.ttf, 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. Tutorial para crear un robot y controlarlo 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, primero crearemos una carpeta de trabajo en Documentos, o donde queramos, que llamaremos `practica2`. Ahora necesitamos localizar la subcarpeta del simulador “`programming->remoteApiBindings->python->python`” y copiar todos los archivos a `practica2`.También se necesita otro fichero. Vamos otra vez al directorio donde tengáis instalado `Vrep -> programming -> remoteApiBindings ->lib ->lib` y veréis dos carpetas: una que pone 64 bits y otra 32 bits. Seleccionad la carpeta adecuada a los bits de tu sistema operativo y dentro veréis un fichero llamado `remoteApi.so` si estáis en Linux o bien `remoteApi.dll` si habéis descargado la versión Windows. Ese fichero también hay que copiarlo a nuestra carpeta de trabajo `practica2.`

Todo está explicado en este tutorial que debemos completar paso a paso para entender el funcionamiento del simulador.

Completad este tutorial:

https://robologs.net/2016/07/07/tutorial-de-vrep-y-opencv-python/

Nota importante: Para cerrar correctamente el script de Python, hay que situarse en las ventanas de las imágenes y pulsar escape.

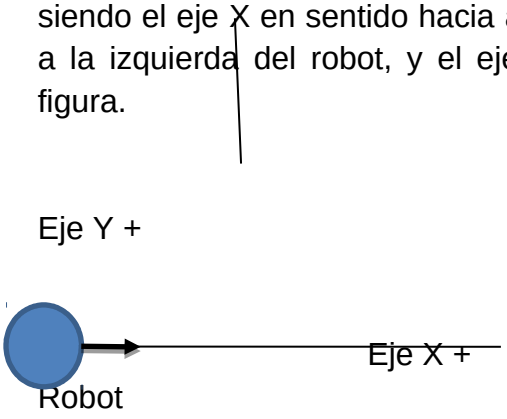
3. Incorporación de un sensor Laser 2D al robot simulado

Para poder realizar la práctica se ha incorporado un sensor Laser 2D al robot simulado del tutorial anterior y además se ha incluido un obstáculo en forma de cilindro y dos modelos de persona junto al cubo verde. Este escenario está disponible en decsai con el nombre “escenaLaser1-3.ttt”. También se ha modificado el script de Python para poder capturar los datos del sensor Laser 2D, salvarlos a un fichero JSON y pintarlos en un eje de coordenadas cartesianas. Este script está en decsai y se llama “ejemploLaser1_JSON.py”. Para cerrar correctamente el script de Python, hay que situarse en las ventanas de las

imágenes y pulsar escape o esperar a que se alcance un número máximo de iteraciones. El fichero JSON creado se guarda en una carpeta diferente llamada “dirLecturaN” para conservar todos los datos de ejecuciones diferentes. También se salva la última imagen captada por el robot en cada caso. Para leer los datos salvados a disco se puede usar el script “ejemploLeerLaser_JSON.py” situado en decsai.

Para realizar la práctica la forma de trabajar será la siguiente:

1. En un terminal ejecutamos el simulador V-Rep y cargamos el escenario “escenaLaser1-3.ttt”.
2. Desde la carpeta practica2, donde están los scripts de la API de python, y el ejemplo “ejemploLaser1_JSON.py” ejecutamos “spyder” (o bien otro editor de Python) y abrimos el script “ejemploLaser1_JSON.py”
3. Iniciamos la simulación con el botón “start” de V-Rep. Esto hace que los objetos del escenario empiecen a publicar sus datos.
4. Desde el spyder ejecutamos el script de Python “ejemploLaser1_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 laser dejando un tiempo de X segundos entre cada iteración hasta un número máximo de iteraciones. Al final se salva también la última imagen.
5. Si desde el simulador, con la simulación activada, movemos el cubo verde o/y los otros objetos se puede observar como van cambiando los datos recibidos por el script de Python.
6. Tened en cuenta que el robot gira para centrar el cubo en la imagen y que los datos del laser 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 ha probado escenaLaser1-3.ttt y “ejemploLaser1_JSON.py” podemos examinar el fichero “datosLaser.json” que se ha creado y leerlo con el script “ejemploLeerLaser_JSON.py”.

4. Realización de la práctica

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

Reutilizando los ficheros de ejemplo anteriores hay que hacer lo siguiente.

1. Eliminamos el cubo verde, el cilindro y la persona sentada. Dejamos el robot y la persona en pie. Hay que hacer un script de Python llamado "**capturar.py"** para la captura de datos del láser en varias situaciones diferentes (N situaciones). Para cada situación primero debe solicitar el nombre del fichero. Supongamos que le damos "enPieCerca". Debe crear una carpeta llamada "positivoN" (siendo N el número de situación capturada) y en ella se debe almacenar el fichero "enPieCerca.json" que contendrá los datos del laser como en el ejemplo "datosLaser.json". El script debe capturar al menos 50 ciclos de lecturas y usar un tiempo de espera entre ciclos de, al menos, 5 segundos. 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.

2. Vamos a contemplar 3 situaciones con una persona en pie según la distancia de la persona al robot, cerca, media o lejos. Estando siempre el robot parado en el punto (0,0) y usando este script, las 3 situaciones para la persona en pie son:

   Situación 1. Fichero: "enPieCerca.json". Mover el modelo de persona en un radio < 1,5 metros al robot y realizando rotaciones. La persona la tenemos que mover nosotros desde el entorno de VREP usando el ratón.

   Situación 2. Fichero: "enPieMedio.json". En este caso mover el modelo de persona a un 1,5 < radio < 2,5 metros y realizar rotaciones.

   Situación 3. Fichero: "enPieLejos.json". En este caso mover el modelo de persona en un 2,5 <radio < 3,5 metros.

   Realizar otras tres situaciones (4,5,6) con la misma filosofía para el modelo de persona sentada debiéndose llamar los ficheros "sentadoCerca.json" "sentadoMedio.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

cilindroMenor1.json, cilindroMenor2.json y cilindroMenor3.json para el par de cilindros menores y los ficheros cilindroMayor1.json, cilindroMayor2.json y cilindroMayor3.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.

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

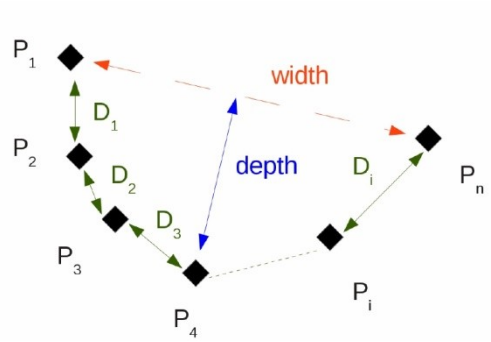
Vamos a definir un clúster como un conjunto de puntos cercanos y que pueden ser candidatos a formar una única pierna. Primero hay que establecer el número mínimo y máximo de puntos que puede tener un clúster. Hay que definirlos como parámetros y según los datos de la experimentación, establecer los más 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 cluster como una lista de puntos, se mete el primer punto, ahora se analiza el segundo punto para ver si se incorpora al cluster actual. Si la distancia del punto anterior al que se está analizando es menor a un umbral, entoces se incorpora el punto analizado al cluster 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 que puede tener un cluster. Para aceptar un grupo de puntos como cluster válido tened en cuenta que debe tener un número mínimo de puntos además de cumplir con el valor máximo.

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]}

4.3. 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 como ejemplo)

4.4. 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 decsai: “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 cual elegir.

4.5. 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 decsai 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. 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 cluster
- Utilice el predictor para cada cluster 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.

Apartado opcional (0.5 puntos, script detectar.py)

Una vez que se han detectado varios objetos, persona o cilindros en la escena vamos a mover la orientación del robot para que pueda captar con la cámara cada objeto detectado y tomar una imagen del mismo. Para girar el robot se puede usar la orden de girar el robot hacia el ángulo en que está el objeto. El ángulo se puede determinar pues conocemos las coordenadas cartesianas de cada objeto respecto al robot (0,0). Se pueden examinar los comandos que acepta el robot en:

http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsMatlab.htm

Para más información sobre cómo funciona el simulador con este robot se puede ver este tutorial para Matlab pues son los mimos comandos que para Python.

https://robologs.net/2017/07/04/interaccion-entre-v-rep-y-matlab/

Se debe generar una tabla en una página web “resultados.html” con las siguientes columnas.

Tipo de objeto	Valor de la predicción	Distancia al robot	Imagen del objeto

Pierna	0.85	2.56	Imagen de las piernas

Plazo de entrega 20 de diciembre.

Para la entrega todos los scripts y carpetas deben estar en un directorio llamado “practicaRobotica.zip” que debe ser comprimido en un fichero .zip o rar y entregado en decsai. Comprobad que dicha carpeta contiene: todos los ficheros necesarios para el cliente de vrep, la escena ejemplo que hayais usado para la captura de datos, junto a las carpetas positivos y negativos de los datos leidos, vuestros scripts de capturar.py, agrupar.py, caracteristicas.py, clasificarSVM.py, predecir.py y el script opcional detectar.py. También hay que incluir la escena, utilizada para la predicción y detección.

Además de los ficheros python generados debe entregarse una memoria explicativa de su funcionamiento que 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). Incluir la página web “resultados.html” si realiza el apartado opcional con imágenes del funcionamiento del simulador.