PRÁCTICA DE INFORMÁTICA / EXPRESIÓN GRÁFICA

Aprendizaje basado en proyectos de ingeniería

1. Introducción

El sistema de recogida de residuos en cualquier localidad es una parte muy importante en el conjunto de sus infraestructuras.

A lo largo del tiempo, una tarea fundamentalmente manual se ha ido automatizando, de tal manera que se ha logrado aumentar la productividad mejorando los tiempos y disminuyendo los riesgos a los que se someten los trabajadores.

En la actualidad, una única persona es capaz de realizar toda la labor de recolección, conduciendo el camión de recogida y activando desde dentro de su misma cabina el mecanismo que se encarga de enganchar el contenedor de residuos, elevarlo y voltearlo sobre el área de recogida de la máquina.

2. Descripción

Aunque la mayoría de los sistemas son muy parecidos conceptualmente, tanto los contenedores como las máquinas recogedoras tienen que estar diseñados para interactuar, así como también tienen que estar habilitadas las zonas de recogida con respecto a unas medidas predeterminadas.

Una de las soluciones más común es la del empleo de contenedores multipropósito que cuentan con unos asideros laterales en forma de pequeñas barras laterales, que permiten que puedan ser elevados fácilmente por ellos. Además, la tapa del contenedor es una puerta deslizante que también expone otros pequeños asideros para permitir su apertura automática.

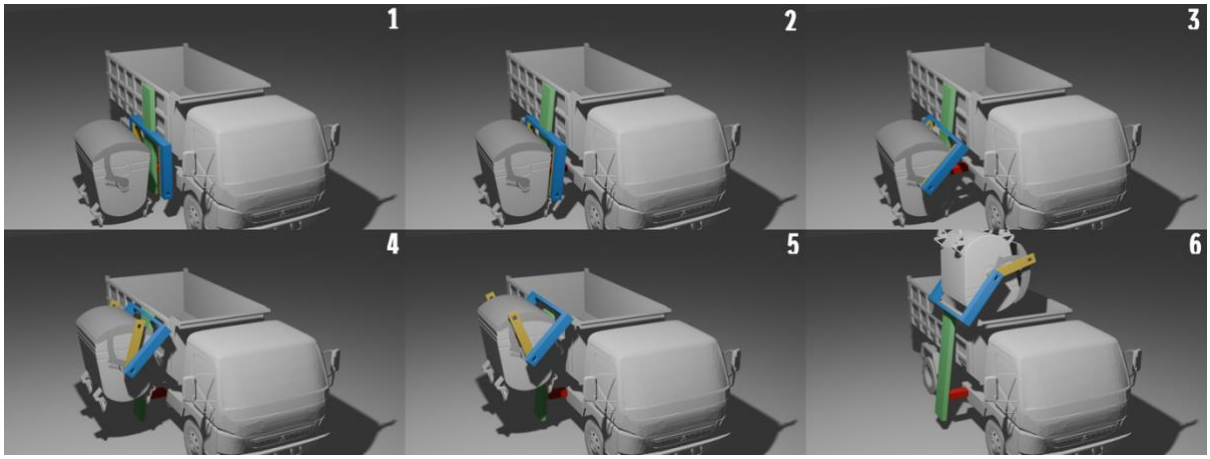
En https://youtu.be/_YFIBou6yJs?t=13 se puede observar el funcionamiento:

El mecanismo está compuesto principalmente por una horquilla cuya misión es rodear al contenedor, empujándolo por los asideros. La horquilla va montada sobre una deslizadera que inicialmente reposa verticalmente en el lateral del vehículo **(1)**.

Al iniciar la maniobra de recogida, la deslizadera se aproxima la horquilla mediante su propio pivotamiento con respecto a la parte superior **(2)**. A continuación, la horquilla se despliega y empuja al contenedor desde abajo por medio de los asideros **(3)**.

En la segunda fase, la deslizadera eleva a la horquilla, y con ello también al contenedor hasta el punto más alto, mientras el segundo brazo abre completamente la tapa **(4)**. En ese punto **(5),** la horquilla rota adicionalmente sobre sí misma posibilitando que el contenedor se vuelque completamente sobre el área de recogida, y el segundo brazo de la horquilla mantiene los asideros de la tapa para que se mantenga abierta completamente y no se vuelva a cerrar por acción de la gravedad **(6)**.

Con esto termina la maniobra de descarga, y la finalización del proceso completo consiste simplemente en repetirla en sentido inverso.



3. Determinación de la secuencia de matrices de transformación (Álgebra)

Para la representación gráfica del movimiento, el ordenador necesita conocer la posición tridimensional de cada punto de la superficie de cada pieza, y de esta manera, graficarla. Las matrices de transformación constituyen una forma compacta de expresar las rotaciones y traslaciones de un sólido rígido en el espacio.

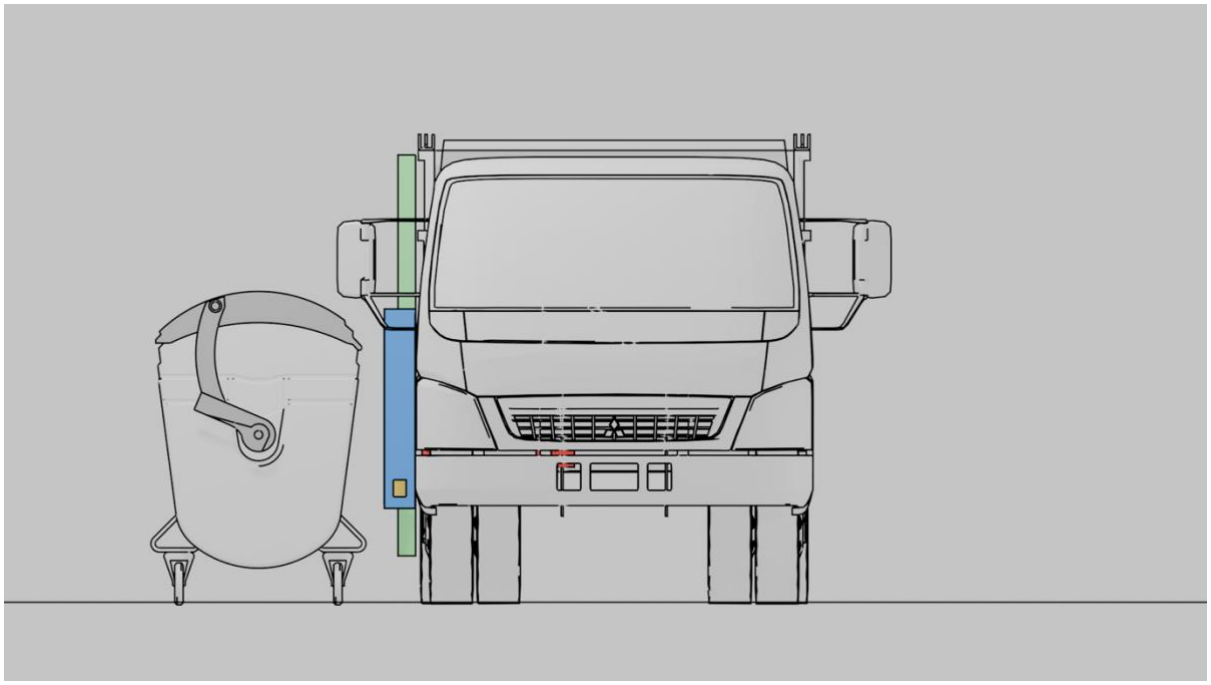
La máquina está constituida de diferentes piezas, que están unidas a las siguientes, y cada una de ellas tendrá un giro y/o una traslación con respecto a la anterior.

Se necesita determinar las rotaciones y la traslación o extensión de cada pieza en cada instante de tiempo que simule el movimiento de la máquina real. Para hallar la matriz que corresponde a cada elemento, hay que calcular la composición con los elementos anteriores, los cuales influyen en su posición.

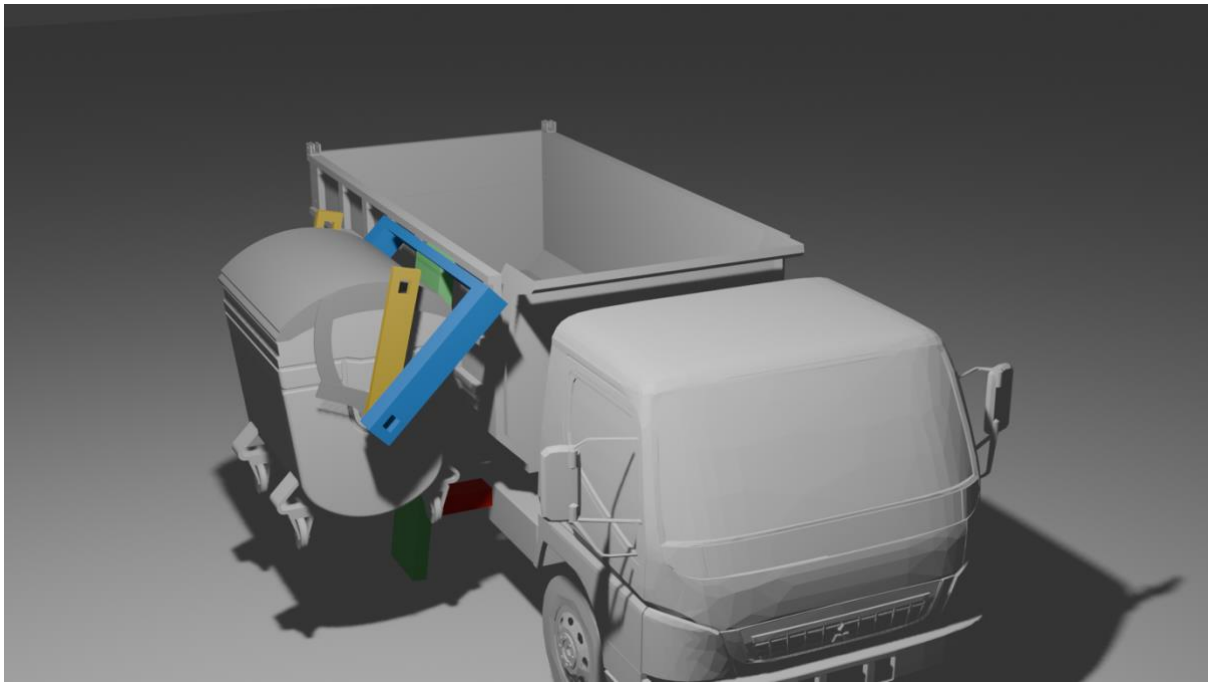
Por ejemplo, la orientación de la horquilla está determinada por la de la deslizadera, y la orientación del contenedor y su tapa también estarán determinadas a su vez por la rotación que la horquilla pueda tener adicionalmente con respecto a la deslizadera.

A continuación, se facilitan las coordenadas necesarias para ensamblar el camión de ejemplo usando las piezas disponibles en el Campus Online.

Elemento	**Traslación (x,y,z)**	**Origen de la traslación**
Camión	[0.0, 0.0, 0.0]	Origen de coordenadas
Empujador	[-0.075283, 0.73129, -0.19924]	Camión
Deslizadera	[-0.2416, 2.2309, 0.19993]	Camión
Brazo 1	[0.0, -0.8463, 0.07907]	Deslizadera
Brazo 2	[0.0, -0.81879, 0.0]	Brazo 1
Contenedor	[-0.17888, 0.83539, 0.98371086]	Origen de coordenadas
Tapa	[-0.03811, 0.0, 0.0]	Contenedor



4. Creación de la geometría 3D mediante CAD (Expresión Gráfica)



Mediante el programa de CAD SolidWorks se puede modelizar cada una de las piezas a color en la figura expuesta, es decir:

1. Deslizadera
2. Horquilla de elevación de contenedor.
3. Brazo de apertura de la tapa del contenedor

No hace falta modelar ni simular los actuadores hidráulicos, ni otro tipo de motores.

Se suministrarán los ficheros correspondientes a las partes recuadradas en blanco en la figura, es decir, todo el resto de la máquina que no sean las 3 piezas pedidas.

Para la modelización de las piezas de la máquina, se deben buscar referencias de máquinas reales como las de la tipología de la figura. Estas referencias visuales (imágenes, vídeos) se deben adjuntar en la entrega final. No es necesario modelizar con total detalle cada uno de los brazos de la grúa, pero se valorará la aproximación al original.

Además de las referencias que se han empleado para la modelización de las piezas (imágenes, enlaces a páginas de fabricantes, vídeos de funcionamiento...), se necesita entregar los ficheros de SolidWorks de cada una de ellas (.SLDPRT) y los planos de cada una de ellas, en el formato de papel (A3, A4) que se considere oportuno (.SLDDRW).

Como de costumbre, además del resultado final en las piezas se busca que se hayan definido todos los croquis 2D empleados. En los planos, que se emplee el número adecuado de vistas, se cumplan las normas de representación y se indiquen todas las medidas necesarias para la fabricación del elemento representado.

Es conveniente diseñar cada una de las piezas correctamente con respecto a los sistemas de coordenadas para que el ensamblaje sea más sencillo. También habrá que anotar las coordenadas de cada articulación con respecto al punto homólogo de la pieza siguiente para poder realizar el cálculo de las matrices de posicionamiento. Para ello se empleará el módulo de ensamblaje del programa. Se puede seguir el tutorial de creación y comprobación de ensamblajes en https://web.microsoftstream.com/video/681f8db3-1b8e-41ca-ae6a-505cc8c5aa79

Posteriormente se puede exportar cada fichero de pieza al formato STL para su carga en el programa de Python. Se debe prestar especial atención a que las dimensiones de las piezas están en **mm**, mientras que al exportarlas se tienen que pasar a **m**, unidades en las que está la simulación. Este proceso está descrito en https://web.microsoftstream.com/video/54ead193-d188-483c-95d8-2a5e0542c83b

5. Programa principal de control (Informática)

El resultado final de la tarea se podrá visualizar mediante un programa escrito en el lenguaje Python. En este lenguaje hay una cantidad ingente de librerías que permiten realizar prácticamente cualquier tipo de tarea con relativa facilidad. En este caso se emplearán las siguientes:

- NumPy => Cálculo con arrays n-dimensionales, álgebra, procesado de señal…
- numpy-stl => Procesado de STL (carga y almacenamiento de ficheros, manipulación…)
- vtkplotlib => Representación simple de gráficos 3D basándose en VTK, NumPy y Matplotlib. Adicionalmente, en nuestro caso, usará también numpy-stl y PyQt6.

Como paso previo a la realización del programa que se entregará, será necesario tomar un par de decisiones:

1. Cuáles serán las posiciones clave de las piezas de la máquina y cuáles son las rotaciones y traslaciones necesarias para llegar a ellas. Por “posiciones clave” se entiende aquellas posiciones por las cuáles una pieza debe de pasar necesariamente para cumplir con el objetivo de la máquina (las posiciones intermedias entre las posiciones clave las deberá de calcular

automáticamente el programa interpolando linealmente). Esto implicará "jugar" con las piezas para ver dónde quedan mejor.

2. Determinar en cuántos pasos de tiempo queremos dividir la animación. A mayor número de pasos, la animación será más suave pero más lenta, con lo que el valor óptimo dependerá de la potencia del ordenador. Se puede empezar con 100 pasos y ajustar cuando el programa esté finalizado.

El pseudocódigo de la aplicación, a grandes rasgos, es el siguiente:

1. Cargar los ficheros STL necesarios.
2. Crear la ventana de visualización.
3. Para cada posición clave:
    a. Para cada paso de tiempo mientras la máquina no llegue a la posición final:
        i. Hacer una copia de la máquina (para no tocar la original).
        ii. Para cada pieza de la (copia de la) máquina:
            1. Determinar su rotación y traslación respecto al origen de coordenadas teniendo en cuenta que le aplicaremos también las de la pieza anterior si la hubiese.
            2. Calcular las matrices de rotación y traslación para esos valores.
            3. Calcular la matriz de transformación correspondiente.
            4. Calcular la matriz de transformación final como producto de la suya y la de la pieza anterior.
            5. Aplicar la matriz de transformación.
        iii. Pintar las piezas en la ventana.

En clase, o en el propio Campus Online, se explicará esta práctica con el detalle que sea necesario / proveerá el material necesario para su comprensión, incluyendo el uso de numpy-stl y vtkplotlib. Para instalar esos paquetes es necesario teclear lo siguiente en el terminal integrado de Visual Studio Code o en cualquier terminal de Windows (cmd o PowerShell):

```
pip install vtkplotlib numpy-stl PyQt6
```

A continuación, se muestran algunos ejemplos que pueden resultar útiles, los cuales fueron probados con las siguientes versiones de software el día 25/11/2024:

- Windows 11 Pro 23H2. En principio, cualquier versión de Windows 10 o Windows 11 debería de valer. Por supuesto, la práctica también se puede realizar con macOS o Linux. Sea como sea, si hay dudas al respecto de los requisitos técnicos, contactad a los profesores.
- Python 3.12.7. En principio, cualquier versión de Python 3.10, 3.11 o 3.12 debería de valer.
- Visual Studio Code 1.95.3
- vtkplotlib 2.1.1. Al instalar este paquete, se instalará automáticamente NumPy 2.1.3, VTK 9.3.1 y matplotlib 3.9.2, salvo que ya esté instalada alguna versión previa.
- numpy-stl 3.1.2
- PyQt6 6.7.1

Si al probar el ejemplo que viene a continuación se produce un error que cuyo texto empieza por "ImportError: DLL load failed while importing", instalad la siguiente librería de Microsoft: https://aka.ms/vs/17/release/vc_redist.x64.exe

5.1. Ejemplo de cómo mostrar un STL en pantalla (numpy-stl y vtkplotlib).

Este ejemplo es útil para comprobar que todo está bien instalado y asume que nuestro fichero se llama “modelo.stl”, si se quiere usar otro, basta con cambiar el nombre.

```
import stl
import vtkplotlib as vpl

# Cargamos el modelo desde un fichero con nombre "modelo.stl".
modelo = stl.mesh.Mesh.from_file("modelo.stl")
# Crea la ventana (pero no la muestra).
vpl.QtFigure()
# Añadimos la malla del objeto a la ventana.
vpl.mesh_plot(modelo)
# Mostramos la ventana.
vpl.show()
```

5.2. Ejemplo de cómo crear una animación con ficheros STL (vtkplotlib).

Este ejemplo consiste en una función que podéis llamar desde vuestro programa para pintar cada paso de la animación. Es decir, para cada paso de tiempo, calcularéis las posiciones de las distintas piezas de la máquina y después llamaréis a esta función, pasándole una lista con las piezas, para que actualice lo que se ve en la ventana. Al hacer esto repetidamente se genera el efecto de la animación.

```
def paint(meshes):
    """
    Muestra los objetos en pantalla.

    meshes : lista con los objetos a visualizar.
    vpl_meshes : lista con los objetos que vamos pintando en pantalla y que se van a
        borrar después de pintarlos. Es necesaria porque vtkplotlib representa los
        objetos de una forma distinta a numpy-stl (guarda más información).
    """
    vpl_meshes = []
    for mesh in meshes:
        # Añadimos la malla del objeto a la ventana.
        vpl_mesh = vpl.mesh_plot(mesh)
        # Lo añadimos, en formato vpl, a otra lista para después poder borrarlo.
        vpl_meshes.append(vpl_mesh)
    # Obtenemos una referencia a la ventana actual.
    figure = vpl.gcf()
    # Recalculamos lo que se va a mostrar en la ventana.
    figure.update()
    # Lo mostramos, pero sin que el programa se detenga.
    figure.show(block=False)
    # Borramos de la ventana los objetos que acabamos de pintar, ya que la próxima vez que se
    # ejecute esta función volverán a ser pintados, probablemente con sus coordenadas
    # modificadas.
    for mesh in vpl_meshes:
        figure.remove_plot(mesh)
```

5.3. Funciones de NumPy que pueden ser útiles.

Ejemplos utilizando algunas funciones de NumPy que ***quizás*** os resulten útiles para la realización de la práctica.

```
import numpy

# Asigna el número 'pi' a la variable 'x'.
x = numpy.pi
# Pasa 60º a radianes.
angulo = numpy.radians(60)
# Calcula el coseno de 60º y lo guarda en 'coseno'.
coseno = numpy.cos(angulo)
# Calcula el seno de 60º y lo guarda en 'seno'.
seno = numpy.sin(angulo)
# Crea una matriz identidad de tamaño 4x4 y la guarda en 'm'.
Rz = numpy.identity(4)
# Junto con la sentencia anterior, crea una matriz de transformación
# en la que solo rota sobre el eje z, sin traslación alguna tampoco.
# https://es.wikipedia.org/wiki/Matriz_de_rotación#Rotaciones_básicas
Rz[0:3, 0:3] = [
    [numpy.cos(angle), -numpy.sin(angle), 0.0],
    [numpy.sin(angle), numpy.cos(angle), 0.0],
    [0.0, 0.0, 1.0],
]
# Suponiendo que se hayan calculado en otro sitio las otras matrices de
# rotación y la de traslación, la matriz de transformación se puede
# calcular así (recuerda que el operador @ multiplica matrices de NumPy):
Mt = T @ Rz @ Ry @ Rx
```

5.4. Ejemplo final.

Ejemplo utilizando algunas de las funciones mencionadas anteriormente que carga un modelo, lo rota 90º sobre el eje Z y lo muestra por pantalla.

```
import numpy
import stl
import vtkplotlib as vpl


def rotate_z(angle):
    """Create a transformation matrix for rotation over z axis."""
    matrix = numpy.identity(4)
    matrix[0:3, 0:3] = [
        [numpy.cos(angle), -numpy.sin(angle), 0.0],
        [numpy.sin(angle), numpy.cos(angle), 0.0],
        [0.0, 0.0, 1.0],
    ]
    return matrix


def main():
    # Crea la ventana
    vpl.QtFigure()
    # Carga el modelo 'pieza.stl' y lo guarda en 'pieza'.
    pieza = stl.mesh.Mesh.from_file("pieza.stl")
    # Rota 'pieza' 90º sobre el eje 'z'.
    T = numpy.identity(4)
```

```
    Rz = rotate_z(numpy.radians(90.0))
    Ry = numpy.identity(4)
    Rx = numpy.identity(4)
    Mt = T @ Rz @ Ry @ Rx
    pieza.transform(Mt)
    # Muestra la pieza
    vpl.mesh_plot(pieza)
    vpl.show()


if __name__ == "__main__":
    main()
```

5.5. Entrega

La práctica debe realizarse de forma individual.

El plazo de entrega finaliza el domingo 22 de diciembre de 2024 a las 23:55 horas (nota: no se resolverán dudas los fines de semana).

Es necesario enviar antes de ese momento el o los fichero(s) .py que contiene(n) el código fuente de la práctica a través del Campus Online de la UDC (https://udconline.udc.gal/), en la sección *Entrega de la práctica*, utilizando el enlace que allí aparece.

Dicho fichero deberá ser renombrado antes de subirlo a la página web de la asignatura de forma que su nombre sea *apellido1_apellido2_nombre.py* donde *apellido1*, *apellido2* y *nombre* son los apellidos y el nombre del alumno que entrega la práctica.

Las notas se publicarán en el Campus Online como muy tarde el 5 de enero.

Se realizará una **defensa de la práctica entre los días 8 y 10 de enero**. Para ello, se habilitará en la página web de la asignatura un procedimiento para concertar una cita con los profesores de la asignatura. La cita debe concertarla cada alumno con su profesor de prácticas correspondiente. **El resultado de esta defensa podrá bajar la nota de la práctica**.

La defensa consistirá en responder al profesor una serie de preguntas al respecto de la práctica. **¡Una práctica perfecta puede ocasionar un 0 si las respuestas no son adecuadas! Si no se realiza la defensa la práctica no puntuará en la nota final.**

Si se detectan varias prácticas iguales o muy similares todos los alumnos implicados tendrán un 0 como nota, incluido el autor original si estamos ante una práctica realizada por un alumno y copiada por otros. A este efecto se usará un detector de plagios y **todas las prácticas con un porcentaje de parecido superior al 50% se considerarán suspensas con un 0**.

5.6. Evaluación

Para puntuar la práctica se valorará lo siguiente:

- Que el programa funcione correctamente.

- La eficiencia en la implementación.
- La organización ***correcta*** del programa en funciones (30% de la nota).
- El uso ***adecuado*** de listas (15% de la nota) y diccionarios (15% de la nota). Por ejemplo, se podría usar una lista para las piezas, donde cada pieza fuese un diccionario con campos para la malla, el nombre de la pieza y el nombre de la pieza a la que está conectada.
- El uso de técnicas adecuadas de programación tales como: ***no*** utilización de variables globales, uso de nombres significativos para las variables y las funciones, existencia de comentarios explicativos, etc.
- La adecuación y rapidez en las respuestas a las preguntas del profesor en la defensa de la práctica por parte del alumno.