Departamento de Ingeniería de Sistemas y Automática

Universidad Politécnica de Valencia

Relaciones espaciales. Aplicación en el simulador *VRS*

Martin Mellado Arteche

Escuela Técnica
Superior de Ingeniería
Informática

Octubre, 2009

Relaciones espaciales. Aplicación en el simulador VRS

1. Introducción

En robótica se necesita representar localizaciones (posiciones y orientaciones) de objetos, lo que normalmente se realiza mediante la definición de sistemas de coordenadas (*frames*) y el uso de transformaciones que significan la relación entre estos sistemas. La fijación de sistemas de coordenadas solidarios con los objetos permite conocer la posición relativa de los primeros a partir del estudio de las situaciones de sus sistemas asociados. Se utiliza álgebra vectorial y matricial para desarrollar un método sistemático y generalizado para describir y representar la localización de los objetos con respecto a un sistema de coordenadas de referencia fijo.

El problema geométrico de conocer la posición y orientación de un objeto en el espacio se reduce a encontrar una matriz de transformación que relacione el sistema de coordenadas ligado al objeto con el sistema de coordenadas de referencia. Se utilizan vectores para describir posiciones y matrices de rotación 3x3 para describir las orientaciones de los objetos con respecto al sistema de referencia. Las matrices de transformación homogéneas 4x4 representan de una manera global posición y orientación, introduciendo el concepto de coordenadas homogéneas. La gran ventaja de la representación matricial es la universalidad algorítmica para desarrollar tanto los modelos geométricos como las ecuaciones cinemáticas y dinámicas de un brazo robot.

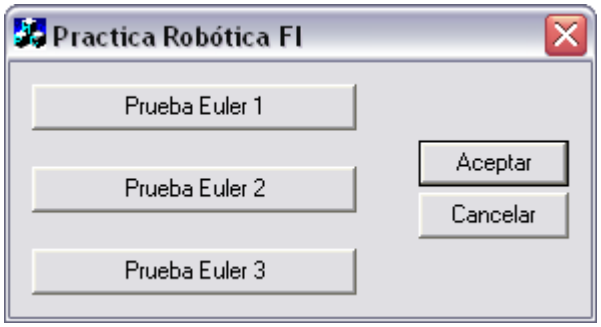
El objetivo de esta práctica es practicar con aspectos de las transformaciones homogéneas explicados en las clases de teoría, principalmente en sus conversiones a otras formas de representar localizaciones, como los ángulos de Euler. Para ello se utilizará la programación en Visual C++ mediante un proyecto ya desarrollado en el que se deben remplazar las rutinas suministradas por nuevas rutinas a implementar en la práctica. Finalmente, se verá el uso de las localizaciones en sus diferentes formas de representación en el simulador *VirtualRobot Simulator*[1].

2. Proyecto desarrollado

2.1. Ejecución del proyecto

Se parte de un proyecto Visual C++ denominado 'practica' (se recomienda tenerlo instalado en C:\Archivos de programa\VIRTUALROBOT\VRS\SourceCode\Users\PracticaVRTransf) y una serie de librerías (VRAux) que se suministran con VirtualRobot y cuya documentación se encuentra accesible desde VRDocs (Inicio->Todos los programas->Virtual Robot). Para comenzar el trabajo se debe abrir el fichero solución del proyecto 'practica.sln'.

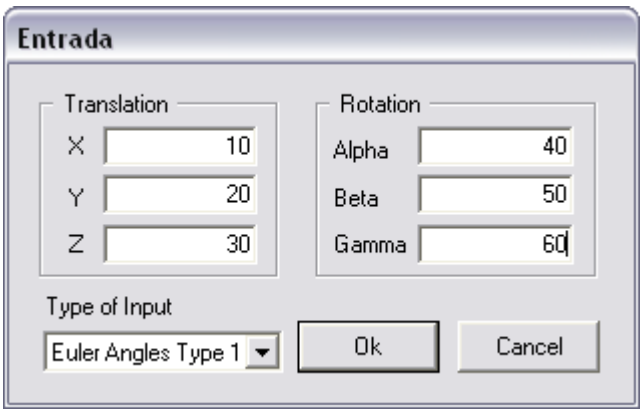
Compila y ejecuta el proyecto mediante la tecla F5, debiendo aparecer el siguiente diálogo:



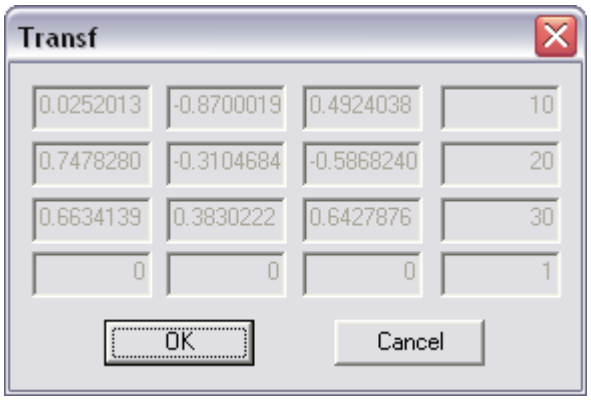
[1] Se supone un conocimiento previo de este simulador.

Como se puede observar, el programa permite, con 3 botones, verificar las conversiones entre transformaciones homogéneas y localizaciones usando ángulos de Euler de los tipos 1, 2 y 3.

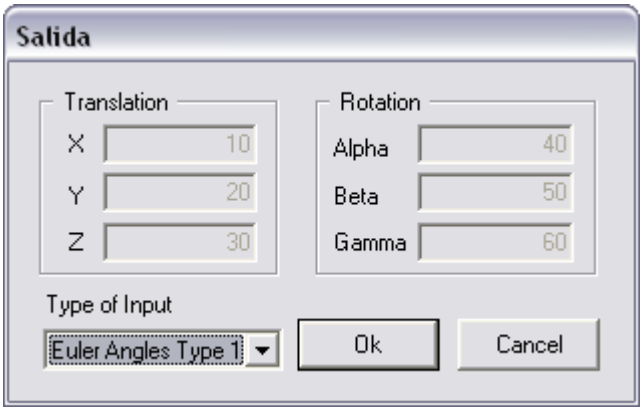
Pulsando en uno de estos botones, por ejemplo Prueba Euler 1, se entrará en el diálogo:



En este diálogo se puede introducir una localización mediante los valores de translación (X,Y,Z) y de rotación (alfa, beta, gama). Introduce unos valores, como por ejemplo (10,20,30) y (40,50,60), tal como aparece en la figura, pulsando Ok. Se obtendrá la matriz de transformación equivalente a dicha localización, siendo la siguiente:



Al pulsar OK, el programa generará de nuevo la localización con los ángulos de Euler correspondientes, que dará los mismos valores que los de la entrada inicial:



Prueba que el funcionamiento es correcto para los tres tipos de ángulos de Euler.

Nota:
Observa que al usuario del programa se le muestran los ángulos en grados, no en radianes.

2.2. Explicación del proyecto

Este proyecto se implementa en el fichero **practicaDlg.cpp** que se encarga de gestionar las funciones asociadas a cada uno de los tres botones existentes en la aplicación. El código asociado a uno de los botones es el siguiente:

```
void CPracticaDlg::OnButtonEuler1()
{
    // TODO: Add your control notification handler code here
    double location[6]={0,0,0,0,0,0};
    double Transformation[4][4];
    double Transf[4][4];
    if (DialogLocation(INPUT_OUTPUT_TYPE,"Entrada",location,EULER_ANGLES_TYPE_1) == INPUT_OK)
    {
        Euler1ToTransformation(location,Transformation);
        DialogTransformation(OUTPUT_TYPE,"Transf",Transformation);
        if (TransformationToEuler1(Transformation,location) == RET_OK)
        {
            Euler1ToTransformation(location,Transf);
            if (TransformationsEqual(Transformation,Transf) != TRUE)
                AfxMessageBox((CString)"La conversión no es correcta");
            DialogLocation(OUTPUT_TYPE,"Salida",location,EULER_ANGLES_TYPE_1);
        }
        else
            AfxMessageBox((CString)"Error en la conversión");
    }
}
```

Básicamente, este código realiza lo siguiente:

- Solicitar una localización de entrada con un tipo concreto de ángulos de Euler
- Convertirla a transformación homogénea y mostrarla
- Convertirla en localización con el mismo tipo de ángulos de Euler
- Volver a generar una nueva transformación homogénea a partir de esta localización
- Comparar las transformaciones homogéneas
- Si todo es correcto, mostrar la localización obtenida

Nota:
Para comparar dos localizaciones se deben comparar las transformaciones homogéneas (con cierta tolerancia), pero no los ángulos de Euler que pueden diferir para la misma localización. Esta comparación se realiza con TransformationsEqual.

El programa usa las siguientes 6 funciones:

- Euler1ToTransformation y TransformationToEuler1
- Euler2ToTransformation y TransformationToEuler2
- Euler3ToTransformation y TransformationToEuler3

Estas funciones, cuya implementación se encuentra en el fichero **trabajo.cpp**, realizan una llamada a funciones de nombre casi similar, disponibles en la librería VRTransf, que resuelven las conversiones en ambos sentidos. Por ejemplo, las funciones para el caso de ángulos de Euler de tipo 1 son:

```
void Euler1ToTransformation(double location[6],double Transformation[4][4])
{
    // Elimina la siguiente línea:
    // Delete the next line:
    LocationEulerToTransformation(EULER_ANGLES_TYPE_1,location,Transformation);

    // HACER: Implementa la función para convertir de localización con ángulos de Euler 1 a Transformación Homogénea
    // TO DO: Program the function to convert from location with Euler angles 1 to Homogeneous Transformation
}
void Euler2ToTransformation(double location[6],double Transformation[4][4])
{
    // Elimina la siguiente línea:
    // Delete the next line:
    LocationEulerToTransformation(EULER_ANGLES_TYPE_2,location,Transformation);

    // HACER: Implementa la función para convertir de localización con ángulos de Euler 2 a Transformación Homogénea
    // TO DO: Program the function to convert from location with Euler angles 2 to Homogeneous Transformation

}
```

El objetivo de la práctica es re-implementar estas funciones con conversiones de matrices de transformación a localizaciones con ángulos de Euler de los 3 tipos y en sentido opuesto.

Nota:
La función TransformationToEuler1 (ó 2, 3) debe devolver RET_OK o RET_ERROR, no así la función Euler1ToTransformation (ó 2, 3).

3. Implementación de conversiones entre localización y transformación

3.1. Conversión entre localización con ángulos de Euler 2 y transformación homogénea

Ejercicio 1: Realiza las siguientes tareas:

- Según el algoritmo de conversión visto en clase de teoría, implementa la función **Euler2ToTransformation**
- Verifica el correcto funcionamiento de dicha rutina (para los mismos datos de entrada en ángulos de Euler de tipo 2 debe generar la misma transformación que anteriormente).
- Según el algoritmo de conversión visto en clase de teoría, implementa la función **TransformationToEuler2**
- Verifica el correcto funcionamiento de dicha rutina (para unos datos de entrada en ángulos de Euler de tipo 2 se deben volver a generar estos mismos datos).

Nota:
- No olvides eliminar la línea correspondiente tal como sugiere el comentario del código.

Sugerencias:

- Normalmente se tendrán que usar funciones trigonométricas como `sin`, `cos` y `tan`, exportadas por la librería estándar de C math.h que ya está incluida en el proyecto.
- Estas funciones trabajan en radianes. Para convertir de grados a radianes y viceversa, se puede usar la librería de VirtualRobot VRMaths (también incluida en el proyecto) que tiene definidas las siguientes constantes:

```
/* Definitions ---------------------------------------------------------*/
#define EPSILON      1E-12
#define PI           3.14159265358979323846264338327950288419716939937511
#define HALF_PI      (PI/2.0)
#define PI2          (PI*2.0)
#define DEG_TO_RAD   (PI/180.0)
#define RAD_TO_DEG   (180.0/PI)
#define ROOT_OF_2    1.41421356237309504880168872420969
#define ROOT_OF_3    1.73205080756887729352744634150587
```

  De esta forma, sen(45º) se obtiene con `sin(45.0 * DEG_TO_RAD)`

- La librería VRMaths también suministra las siguientes funciones trigonométricas:

```
// Trigonometric Functions

extern "C" __declspec( dllimport )double vrAtan2(double y, double x);
// this function computes arctan(y/x) according to quadrant
// the value is returned in radians

extern "C" __declspec( dllimport )int vrInvSin(double sinx, double *angle);
// this function computes positive solution of inverse Sin
// the angle is considered in radians
// the function returns RET_OK or RET_ERROR (when |sinx|>1.0)

extern "C" __declspec( dllimport )int vrInvCos(double cosx, double *angle);
// this function computes positive solution of inverse Cos
// the angle is considered in radians
// the function returns RET_OK or RET_ERROR (when |cosx|>1.0)
```

Nota que estas dos últimas funciones tienen dos parámetros (uno de entrada y otro de salida) y que pueden devolver error si el valor de entrada no es el adecuado. Por tanto, si queremos programar beta=arcos(az) se realizaría de la siguiente forma:

```
int err=vrInvCos(az,&beta);
```

La variable err valdrá RET_OK o RET_ERROR en función del valor de az.

- La librería de VirtualRobot VRTransf (también incluida en el proyecto) de donde se han tomado las funciones de partida, puede tener otras funciones que sean útiles para realizar la solución. Se pueden usar libremente. En esta librería, hay que en cuenta que:
  - Cuando se utiliza el vector location[6] representa:
    location[0], location[1], location[2] es x,y,z
    location[3], location[4], location[5] es alfa,beta,gama
  - Cuando se utiliza la matriz T[4][4] representa:

T[0][0] es n_x	T[0][1] es s_x	T[0][2] es a_x	T[0][3] es x
T[1][0] es n_y	T[1][1] es s_y	T[1][2] es a_y	T[1][3] es y
T[2][0] es n_z	T[2][1] es s_z	T[2][2] es a_z	T[2][3] es z
T[3][0] = 0	T[3][1] = 0	T[3][2] = 0	T[3][3] = 1

- Un olvido habitual que no hay que hacer en la implementación es olvidarse de la posición.
- En el apéndice A están desarrolladas las matrices de rotación para los tres tipos de ángulos de Euler y un posible algoritmo para el tipo 2.

3.2. Conversión entre localización con ángulos de Euler 1 y transformación homogénea

Ejercicio 2: Realiza las siguientes tareas:

- Según el algoritmo de conversión visto en clase de teoría, implementa la función **Euler1ToTransformation**. Verifica el correcto funcionamiento de dicha rutina
- Según el algoritmo de conversión visto en clase de teoría, implementa la función **TransformationToEuler1**. Verifica el correcto funcionamiento de dicha rutina

3.3. Conversión entre localización con ángulos de Euler 3 y transformación homogénea

Ejercicio 3: Realiza las siguientes tareas:

- Según el algoritmo de conversión visto en clase de teoría, implementa la función **Euler3ToTransformation**. Verifica el correcto funcionamiento de dicha rutina
- Según el algoritmo de conversión visto en clase de teoría, implementa la función **TransformationToEuler3**. Verifica el correcto funcionamiento de dicha rutina

Ejercicios opcionales: implementa los algoritmos anteriores considerando tolerancias.

4. Utilización de localizaciones en el simulador *VRS*[2]

Para poder programar *VRS* mediante VREAL en este proyecto se deben realizar las siguientes operaciones:

- En el fichero practicaDlg.cpp se debe incluir VReal.h
- En el proyecto se debe incluir VReal.lib
- Se debe inicializar la comunicación con VRS mediante `alInitialize` en `OnInitDialog` y cerrar con `alClose` en `OnOk` y en `OnCancel`.
- Se debe arrancar *VRS*

4.1. Localización de figuras en *VRS*

Revisa la sección 19 de la documentación de VREAL. Comprueba que:

- con funciones del tipo alAddFrame, alAddPoint, … alAddBox, alAddCone, etc se pueden generar figuras en *VRS* (de la misma forma que hace el VRMEditor)
- con la función alSetFigureTransformation se puede aplicar una transformación a una figura.

[2] Para realizar esta sección se debe conocer cómo usar el simulador *VRS* y cómo programarlo

Ejercicio 4:
Modifica el programa para que:
- Cuando se pulse el botón de ángulos de Euler tipo 1, se cree un prisma[3] (elige las dimensiones y colores libremente) y se localice según los datos introducidos.
- Para los otros botones, aplica la transformación generada a otras dos figuras diferentes.
- Al cerrar el programa se deben borrar las figuras que haya en *VRS*.

4.2. Localización de objetos en *VRS*

Revisa la sección 17 de la documentación de VREAL. Comprueba que:
- con la función alSetObjectLocation se puede fijar la localización de un objeto.
- con la función alSetPartLocation se puede fijar la localización de una pieza.

Ejercicio 5:
Modifica el programa para que:
- Cuando se arranque el programa se cargue un entorno en *VRS*
- Cuando se pulse el botón de ángulos de Euler tipo 1, se modifique la localización de uno de los objetos existentes en el entorno.
- Para otro botón, modifica la localización de una pieza.
- Al cerrar el programa se debe cerrar el entorno de *VRS*.

4.3. Movimiento de un robot en *VRS*

Tal como se define en VREAL (sección 14, options), la función alMoveRobot permite definir la localización de movimiento del robot mediante:
- 6 parámetros reales (x,y,z,alpha,beta,gamma)
- Un vector de 6 parámetros reales (x,y,z,alpha,beta,gamma)
- Una matriz 4x4 (transformación homogénea)

Ejercicio 6:
Modifica el programa anterior para que:
- Cuando se arranque el programa se cargue un robot en *VRS*
- Se indiquen movimientos al robot de estas 3 formas.

Ejercicio 7:
Modifica el programa para que, a partir de una localización introducida de cualquiera de estas formas, el robot realice una aproximación a esta localización (distancia 100 en su eje director) y posteriormente, en línea recta, se mueva a esta localización.

5. Ampliación de la práctica

5.1. Uso de la clase CTransformation

La librería VRTransf incluye también la definición de la clase CTransformation con atributos propios y métodos basados en las funciones disponibles en esta librería. Investiga cómo funciona la clase implementando alguno de los ejemplos anteriores con el uso de esta clase.

[3] En *VRS* se denomina Box

Apéndice A. Ángulos de Euler

A.1. Ángulos de Euler Tipo 1

La matriz de transformación con ángulos de Euler de tipo 1 es:

$$T_{x,y,z,\alpha,\beta,\gamma}=\begin{bmatrix} \cos\alpha\cos\gamma-\sin\alpha\cos\beta\sin\gamma & -\cos\alpha\sin\gamma-\sin\alpha\cos\beta\cos\gamma & \sin\alpha\sin\beta & x \\ \sin\alpha\cos\gamma+\cos\alpha\cos\beta\sin\gamma & \cos\alpha\cos\beta\cos\gamma-\sin\alpha\sin\gamma & -\cos\alpha\sin\beta & y \\ \sin\beta\sin\gamma & \sin\beta\cos\gamma & \cos\beta & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A.2. Ángulos de Euler Tipo 2

La matriz de transformación con ángulos de Euler de tipo 2 es:

$$T_{x,y,z,\alpha,\beta,\gamma}=\begin{bmatrix} \cos\alpha\cos\beta\cos\gamma-\sin\alpha\sin\gamma & -\cos\alpha\cos\beta\sin\gamma-\sin\alpha\cos\gamma & \cos\alpha\sin\beta & x \\ \sin\alpha\cos\beta\cos\gamma+\cos\alpha\sin\gamma & \cos\alpha\cos\gamma-\sin\alpha\cos\beta\sin\gamma & \sin\alpha\sin\beta & y \\ -\sin\beta\cos\gamma & \sin\beta\sin\gamma & \cos\beta & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A.3. Ángulos de Euler Tipo 3

La matriz de transformación con ángulos de Euler de tipo 3 es:

$$T_{x,y,z,\alpha,\beta,\gamma}=\begin{bmatrix} \cos\gamma\cos\beta & \cos\gamma\sin\beta\sin\alpha-\sin\gamma\cos\alpha & \cos\gamma\sin\beta\cos\alpha+\sin\gamma\sin\alpha & x \\ \sin\gamma\cos\beta & \sin\gamma\sin\beta\sin\alpha+\cos\gamma\cos\alpha & \sin\gamma\sin\beta\cos\alpha-\cos\gamma\sin\alpha & y \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A.4. Algoritmo de resolución para la transformación con ángulos de Euler Tipo 2

ALGORITHM VERSION 1: RotationToEulerAngles2
Input: nx,ny,nz sx,sy,sz ax,ay,az
Output: alfa,beta,gama (degrees)
Return: OK o ERROR

```
alpha=beta=gamma=0
if az=1
   gamma=atan2(ny,nx)*180/pi
else
   if az=-1
      beta=180
      gamma=atan2(ny,-nx)*180/pi
   else
      beta=invcos(az)*180/pi
      alpha=atan2(ay,ax)*180/pi
      gamma=atan2(sz,-nz)*180/pi
return OK
```