

Sistema de monitorización inercial del movimiento de las extremidades superiores

Daniel Fernández Villanueva

12 de julio de 2013

Índice general

I Memoria	5
1. ANTECEDENTES	6
2. OBJETIVO DEL PROYECTO	7
3. ESPECIFICACIONES DE DISEÑO	8
4. DISEÑO DEL SISTEMA	9
4.1. Estudio de soluciones	9
4.1.1. Obtención del estado del brazo	9
Encoders	9
Sensores ópticos	9
Sensores inerciales	9
Solución escogida: red de sensores inerciales xsens MTx	9
4.1.2. Comunicación entre los sensores y el PC	9
Driver para la comunicación xbus Master - PC	10
4.1.3. Comunicación entre programas	10
ROS	10
Herramientas proporcionadas por ROS	11
4.1.4. Sistema operativo	13
El sistema operativo Ubuntu	13
5. CÁLCULO DE LAS POSICIONES Y ORIENTACIONES DE LOS SEN- SORES	15
5.1. Formas de representar orientaciones espaciales	15
5.1.1. Ángulos de Euler	16
5.1.2. Matriz de rotación	16
5.1.3. Cuaternion	18
5.1.4. Solución elegida: cuaterniones	19
5.2. Utilización de cuaterniones para la representación de rotaciones de un sólido rígido	19
5.2.1. Rotación de un vector alrededor de un eje y un ángulo dados	21
5.2.2. Composición de rotaciones en coordenadas extrínsecas	23
5.2.3. Composición de rotaciones en coordenadas intrínsecas	24
5.2.4. Relación entre rotaciones intrínsecas y extrínsecas	24
5.2.5. Orientación relativa entre dos sólidos	27
5.3. Cálculo de la posición del brazo	28
5.4. Obtención de los ángulos de Euler a partir del cuaternion de orientación . .	29
5.4.1. Ampliación del intervalo de los ángulos obtenidos	32

Generación de una segunda solución a partir de la primera	32
Permutación de los ejes locales	33
6. IMPLEMENTACIÓN DEL SOFTWARE	34
6.1. Driver para la comunicación xbus Master/xsens - PC	34
6.1.1. Funcionamiento de la clase xsens::Driver	35
6.1.2. Programa principal: xsens_node	38
6.1.3. Cómo tomar datos del sensor desde otro programa	39
6.2. Librería para manejo de cuaterniones, vectores y matrices	40
6.3. Controlador del brazo robótico del robot Youbot real	40
6.3.1. El driver youbot_oodl	41
6.3.2. La clase Youbot	41
6.4. Otras aplicaciones de visualización y control	42
6.4.1. Visualizador de la posición del brazo	42
6.4.2. Controlador de un simulador del brazo robótico del robot Youbot .	42
7. IMPLEMENTACIÓN FÍSICA	45
7.1. Selección de componentes	45
7.2. Montaje	45
7.3. Ajuste	45
8. PROTOCOLO DE PRUEBAS. REDISEÑO	46
9. RESULTADOS OBTENIDOS	47
10. CONCLUSIONES	48
11. BIBLIOGRAFÍA	49
II Anexos	50
A. INSTALACIÓN Y PUESTA EN MARCHA DEL SOFTWARE	51
A.1. Instalación de ROS Fuerte	51
A.1.1. Configuración de los repositorios de Ubuntu	51
A.1.2. Configuración del archivo sources.list	51
A.1.3. Configuración de la keys	52
A.1.4. Descarga e instalación	52
A.1.5. Configuración del entorno	53
A.2. Instalación del simulador Gazebo	53
A.3. Instalación del simulador de brazo robótico del YouBot	53
A.4. Instalación del stack de ROS youbot-xsens-controller	53
A.4.1. Instalación desde el CD adjunto al proyecto	53
A.4.2. Instalación desde git	53
B. SOLUCIÓN DE PROBLEMAS	54
B.1. Error iniciando Gazebo	54

Índice de figuras

4.1.	Red de sensores xsens conectados a un máster xbus	10
4.2.	Logotipo de ROS	11
4.3.	Comando rostopic mostrando datos de un sensor	13
4.4.	Programa rxplot mostrando datos de un sensor	14
4.5.	Programa Gazebo con la simulación del brazo del Youbot	14
5.1.	Rotación de un vector mediante la operación $q_1 p q_1^*$	22
5.2.	Orientación relativa entre dos sólidos rígidos	27
5.3.	Posiciones del brazo inicial y genérica	28
5.4.	Sistema de coordenadas local sobre la esfera	30
5.5.	Tercera rotación en el sistema de coordenadas local sobre el punto P de la esfera	30
5.6.	Dos soluciones para θ (<i>pitch</i>) y ψ (<i>yaw</i>)	33
6.1.	Esquema simplificado de comunicaciones	43
6.2.	Articulaciones del brazo robot Youbot	44
A.1.	Ventana Software Sources	52

Índice de cuadros

5.1.	Propiedades de los cuaterniones	18
5.2.	Algoritmo de cálculo de los ángulos de Euler a partir del cuaternion de orientación.	32
6.1.	Topics publicados por el programa xsens_node	39
6.2.	Topics publicados por el programa xsens_node	39
6.3.	Topics a los que está suscrito el driver youbot_oodl	41

Parte I

Memoria

Capítulo 1

ANTECEDENTES

Capítulo 2

OBJETIVO DEL PROYECTO

Este proyecto tiene como objetivo la implementación de un sistema de monitorización en tiempo real del movimiento de las extremidades superiores del cuerpo humano. En concreto, este sistema deberá ser capaz de registrar el estado de orientación principalmente de un brazo humano, aunque su uso será extensible a otros sistemas articulados. De forma inherente este objetivo, aparecen unos problemas a los que es necesario proporcionar una solución:

1. Toma de datos de los sensores:

Realización de un sistema que permita obtener los datos que proporcionan los sensores utilizados para registrar el movimiento. Este sistema deberá de ser capaz de realizar la lectura en tiempo real, con mínima latencia y a una frecuencia aceptable.

2. Tratamiento de los datos:

Una vez se tenga disponible un sistema para leer los datos de los sensores, el siguiente paso será procesar dichos datos para obtener las magnitudes necesarias para las distintas aplicaciones (Por ejemplo: orientación relativa entre sensores, ángulos de Euler, etc.)

3. Utilización de los datos:

En esta última fase se crearán los sistemas necesarios para la utilización de los datos con el objetivo deseado. En este proyecto se crearán tres aplicaciones concretas:

- a) Visualización de un esquema del brazo en un visualizador 3D.
- b) Control de un simulador de un brazo robótico.
- c) Control del movimiento de un robot real.

En este documento se realizará una descripción detallada de las soluciones que se han adoptado para cada problema, y como se han incorporado dentro del sistema total de monitorización, de forma que cada parte se comunique con las demás de forma eficaz.

Capítulo 3

ESPECIFICACIONES DE DISEÑO

En esta apartado se detallan las características esenciales que se pretenden implementar en el sistema de monitorización.

- **Flexibilidad:** que no sólo sea capaz de registrar la posición de un brazo humano, sino que sea aplicable a otro tipo de sistemas articulados, tanto humanos como de máquinas y robots. Por ello dicho sistema ha de ser flexible en cuanto a número de sensores y grados de libertad que se puedan monitorizar.
- **Pequeño y ligero:** el sistema no puede ser demasiado aparatoso, ya que en se pretende colocarlo en extremidades humanas. Además no debería de dificultar la movilidad de la persona en absoluto.
- **Fiabilidad.** Que sea lo más fidedigno posible en las medidas que proporciona. Esto puede tener mucha importancia en aplicaciones donde se requiera precisión.
- **Económico**
- **Versatilidad.** El sistema debería dejar la puerta abierta a poder ser incorporado en otras aplicaciones de forma sencilla, además de poder ser ampliado con mayor número de sensores o de distinto tipo sin mucho esfuerzo.

Capítulo 4

DISEÑO DEL SISTEMA

4.1. Estudio de soluciones

4.1.1. Obtención del estado del brazo

Encoders

Sensores ópticos

Sensores inerciales

Los sensores inerciales –normalmente incluidos dentro de una misma unidad llamada Unidad de Medición Inercial - Inertial Measurement Unit o IMU para abreviar– utilizan un conjunto de acelerómetros, giróscopos y en ocasiones magnetómetros para la obtención de la velocidad, orientación, fuerzas magnéticas entre otros posibles parámetros.

Una de las principales desventajas que presentan los IMUs es la deriva que presentan a lo largo del tiempo. Esto es debido a que para obtener el estado en cierto tiempo, hacen uso de las medidas del estado anterior y los errores presentes se van acumulando. Si embargo para la orientación es posible minimizar la deriva a un mínimo, incluso eliminarla, mediante un sistema llamado AHRS (*Attitude and Heading Reference System*), en el que se utilizan la medida de la aceleración de la gravedad y el norte magnético para compensar las derivas que se obtienen con los giróscopos.

Solución escogida: red de sensores inerciales xsens MTx

Los sensores xsens MTx implementan la corrección AHRS para el cálculo de la orientación, por lo que proporcionan una medida de alta precisión y sin deriva apreciable de la orientación que presenta el sensor. Además es posible utilizar varios de estos sensores a la vez formando una red conectándolos a un máster xbus, lo que permitirá tomar los datos de los distintos eslabones que forman el brazo al mismo tiempo.

4.1.2. Comunicación entre los sensores y el PC

La conexión del máster Xbus al PC se realizará mediante cable USB. Para la toma de datos del sensor será necesario un programa especial llamado *driver* que se encargue de gestionar las comunicaciones a través del puerto serie.



Figura 4.1: Red de sensores xsens conectados a un máster xbus.

Driver para la comunicación xbus Master - PC

Las funcionalidades que se pretenden implementar en el *driver* del Xsens son las siguientes:

- Capacidad de funcionar con varios sensores conectados
- Posibilidad de realizar la configuración de los sensores.
- Lectura a frecuencia aceptable.
- Integración en el sistema ROS.

Existen algunos drivers disponibles en internet preparados para trabajar en ROS con sensores Xsens. Sin embargo, lo que se han encontrado o bien sólo sirven para un sólo sensor, o trabajan en versiones bastante desfasadas de ROS y se encuentra poca o nula documentación sobre ellos.

La opción que se ha decidido seguir fue aprovechar que en la documentación de los sensores Xsens se proporcionan un conjunto de archivos con código en C++ para la comunicación a través del puerto serie con un sensor Xsens o un máster Xbus, y a partir de ellos construir el driver para la comunicación con los sensores y la publicación de los datos en ROS. Se realiza una descripción detallada de este programa en la sección *Driver para la comunicación Xbus Master/Xsens - PC* del capítulo *IMPLEMENTACIÓN DEL SOFTWARE*.

4.1.3. Comunicación entre programas

Dada la multi-modularidad de la que se pretende dotar al sistema de monitorización, será necesario un sistema en el que distintos programas se puedan comunicar entre sí. Sería posible resolver las comunicaciones partiendo desde cero mediante sockets TCP/IP. Sin embargo es bastante más práctico utilizar algún tipo de plataforma que resuelva por ella misma las comunicaciones. En este proyecto se usará la plataforma ROS, que además de solucionar el tema de las comunicaciones, posee una gran cantidad de herramientas que van a ser de mucha utilidad.

ROS

ROS (del inglés *Robot Operating System* - Sistema Operativo Robótico) es una plataforma de desarrollo de software que incluye conjunto de utilidades centradas en ayudar al desarrollador en la creación de programas para el control de robots. Esta herramienta incorpora abstracción del hardware, drivers para dispositivos, bibliotecas, visualizadores, utilidades para el intercambio de mensajes entre programas y administradores de paquetes de software, entre otras muchas cosas. ROS es además software abierto, bajo una licencia BSD, por lo que cualquier persona puede ver su código fuente y modificarlo.



Figura 4.2: Logotipo de ROS.

Herramientas proporcionadas por ROS

La plataforma ROS proporciona solución a diversos problemas que vienen dados inherentemente al objetivo de este proyecto:

- **Creación y compilación de programas:**

ROS proporciona bibliotecas en los lenguajes C++, python y lisp en las que se implementan las diversas herramientas que pone a disposición, como intercambio de mensajes entre programas, servidor de parámetros, gestión del tiempo, entre muchas otras. En el presente proyecto se utilizará la implementación en C++ –**rosCPP**–, que es la biblioteca más ampliamente difundida en la comunidad de ROS, además de estar diseñada para ser la biblioteca más actualizada y con mayor número de herramientas.

Además ROS pone a disposición un gestor de paquetes de software con el fin de facilitar la reutilización de código. Un **paquete** es la forma más básica de organización del software en ROS. Un paquete puede contener programas –llamados **nodos**–, bibliotecas, o cualquier otra cosa que posea funcionalidad. La herramienta *roscreate-pkg* permite la creación de un paquete de forma automática.

En un **stack** de ROS se pueden incluir varios paquetes. Los *stacks* tienen por fin hacer más sencillo compartir conjuntos de paquetes entre distintos ordenadores. Todos los paquetes creados en el presente proyecto se incluirán dentro de un mismo *stack*.

- **Comunicación entre programas:** Uno de los puntos más fuertes de ROS son las herramientas que incluye para la comunicación entre programas, las cuales hacen posible crear un sistema totalmente distribuido, incluso entre varios ordenadores, en los que cada programa puede comunicarse con los demás mediante un interfaz sencilla y con mínima latencia. El sistema de intercomunicación de ROS se basa en los siguientes conceptos:

- **Máster:** El máster de ROS es el programa que se encarga de gestionar las comunicaciones entre los distintos nodos (programas de ROS). Permite que distintos nodos se encuentren entre si, lleva un registro de los *publishers* y *subscribers* de los distintos *topics* y de los *servicios*, además de proporcionar un *servidor de parámetros*.

- **Topics:** Los *topics* son los canales de comunicación que utilizan los distintos *nodos* para comunicarse entre sí. Esta comunicación es llevada a cabo mediante *mensajes*.
- **Mensajes:** Un *topic* acepta una determinada clase de mensaje. Los mensajes están compuestos por un conjunto de uno o varios tipos de datos –floats, strings, arrays, etc.–, o de otros mensajes en una estructura anidada. Existen unas definiciones de mensajes estándar, aunque es posible crear nuevas definiciones de estos mensajes. En este proyecto se han tratado de utilizar mensajes estándar en la medida de lo posible.
- **Servicios:** Los *topics* proporcionan una comunicación entre nodos de tipo publicador-subscriptor muy flexible, pero en ocasiones se requiere un sistema de tipo solicitud-respuesta, en el que un cliente envía un mensaje de solicitud y el servidor contesta con un mensaje de respuesta. Este tipo de comunicación se implementa en los *servicios* de ROS. Los servicios están pensados para utilizarse en la solicitud de realización de ciertas tareas de vez en cuando. Para la comunicación de datos en *streaming* –por ejemplo para el envío de los datos de los sensores– es preferible el uso de *topics*.
- **Servidor de parámetros:** El servidor de parámetros de ROS es una especie de diccionario que puede ser accedido por los distintos nodos para crear, leer o modificar los valores de los parámetros almacenados en él. Se usará el servidor de parámetros para almacenar los valores de configuración de los sensores.
- **Bags:** Los *bags* de ROS son archivos que permiten almacenar los datos de uno o varios mensajes para su posterior reproducción. Permiten, por ejemplo, grabar los datos de los sensores para poder usarlos en otro momento o lugar en los que no estén disponibles los propios sensores.

■ Visualización de datos

- **rostopic:** El comando *rostopic* permite conocer en tiempo real los topics que se encuentran publicados en ROS, y obtener los valores que se están publicando en ellos. Para obtener una lista de los topics se ejecutará el siguiente comando:

```
$ rostopic list
```

Si se quisiera conocer los valores publicados se ejecutará el siguiente comando:

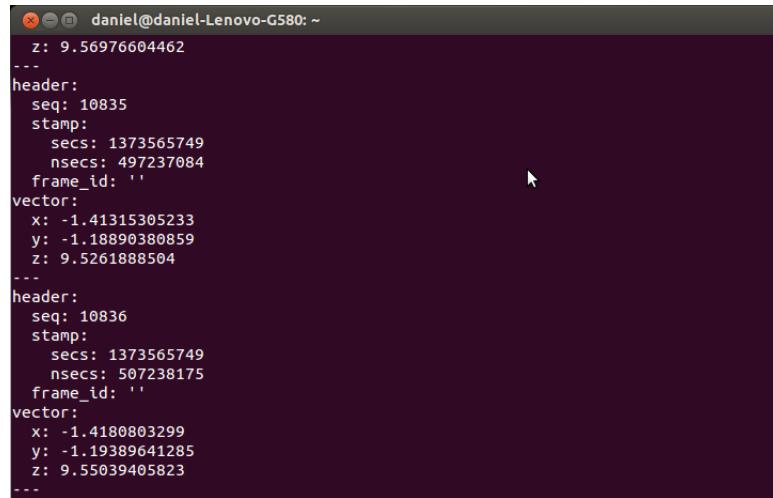
```
$ rostopic echo <nombre_del_mensaje>
```

Por ejemplo, en la figura 4.3 se muestran los datos de los acelerómetros de un sensor xsens conectado al PC. Es el resultado de ejecutar el siguiente comando:

```
$ rostopic echo /xsens_node/sensor0/acc
```

- **rxplot:** La herramienta rxplot permite la representación gráfica de uno o varios valores a la vez, en tiempo real. Un ejemplo se muestra en la figura 4.4, que se ha obtenido con el comando:

```
$ rxplot -r 10 /xsens_node/sensor0/acc/vector/x:y:z
```



```
daniel@daniel-Lenovo-G580: ~
z: 9.56976604462
---
header:
  seq: 10835
  stamp:
    secs: 1373565749
    nsecs: 497237084
  frame_id: ''
vector:
  x: -1.41315305233
  y: -1.18890380859
  z: 9.5261888504
---
header:
  seq: 10836
  stamp:
    secs: 1373565749
    nsecs: 507238175
  frame_id: ''
vector:
  x: -1.4180803299
  y: -1.19389641285
  z: 9.55039405823
---
```

Figura 4.3: Comando rostopic mostrando datos de un sensor en tiempo real.

■ Visualización 3D

ROS incluye varios paquetes para la visualización y simulación 3D de objetos. Entre ellos, los más importantes son:

- **rviz:** El paquete rviz incluido en ROS es un paquete que permite la lectura de topics de ROS desde el propio programa, y asignar los valores a parámetros de la visualización. No es un simulador de física de objetos 3D, por lo que no permite la simulación de robots propiamente dicha.
- **gazebo:** El programa Gazebo es un simulador de física de objetos rígidos. Permite la simulación de un robot o de un conjunto de robots, y su interacción entre ellos y con el ambiente. Además es posible obtener una simulación de la respuesta que tendrían ciertos tipos de sensores en el mundo real.

En la documentación del robot YouBot viene incluída una simulación del brazo del robot en Gazebo con la que se puede interactuar mediante ciertos topics de ROS.

4.1.4. Sistema operativo

El sistema operativo Ubuntu

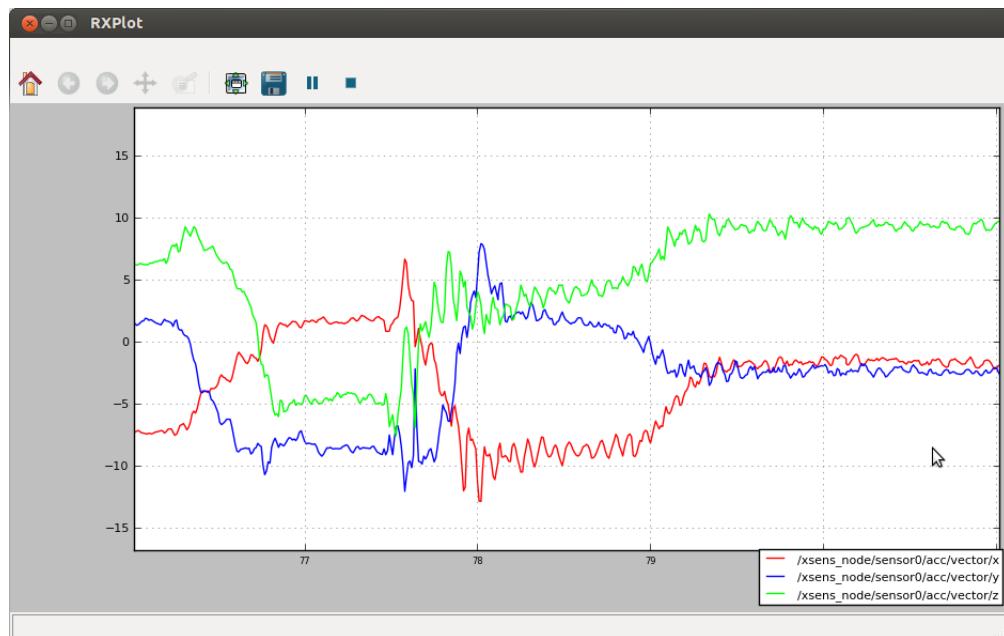


Figura 4.4: Programa rxplot mostrando una gráfica con los datos de los acelerómetros de un sensor en tiempo real

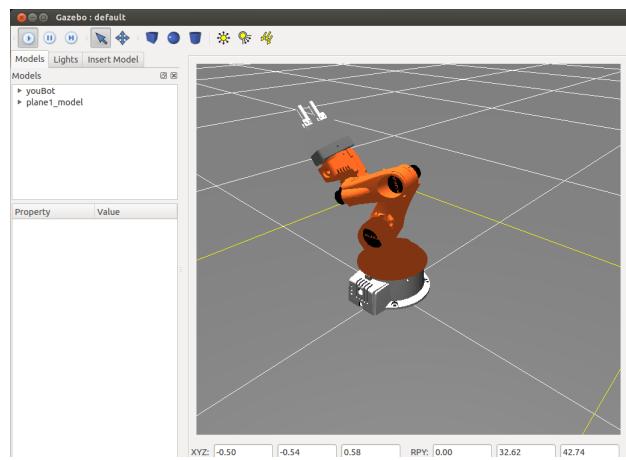


Figura 4.5: Programa Gazebo con la simulación del brazo del Youbot

Capítulo 5

CÁLCULO DE LAS POSICIONES Y ORIENTACIONES DE LOS SENSORES

Uno de los objetivos del proyecto es capturar el estado de orientación de los sensores para la utilizar los datos obtenidos en otros sistemas. Para ello es necesario utilizar un sistema de representación matemática de las orientaciones que permita la derivación de las magnitudes que se deseen (como ejes y ángulos de rotación) de forma rápida e inequívoca. En este capítulo se pretende dar una visión general de la derivación matemática y justificación previa de los algoritmos cuya implementación se mostrará más adelante.

5.1. Formas de representar orientaciones espaciales

Existen multitud de formas de representar la orientación de un sólido rígido:

- Ángulos de Euler
- Eje y ángulo de Euler
- Matriz de rotación
- Cuaternión
- Parámetros de Rodrigues
- Parámetros de Cayley-Klein
- ...

De todas estas formas de representación, los sensores xsens, al igual que muchos IMUs modernos, proporcionan la posibilidad de obtener los ángulos de Euler, la matriz de rotación y el cuaternion de rotación. En los siguientes apartados se analizarán las ventajas y desventajas de dichos sistemas y se escogerá la más conveniente.

5.1.1. Ángulos de Euler

Los ángulos de Euler son tres ángulos introducidos por Leonhard Euler para describir la orientación de un sólido rígido o de un sistema de referencia respecto a otro. Representan una secuencia de tres rotaciones elementales alrededor de los ejes de un sistema de coordenadas. Cualquier orientación puede describirse como la composición de tres rotaciones elementales, que pueden suceder alrededor de los ejes de un sistema de referencia fijo (rotaciones extrínsecas) o alrededor de los ejes de un sistema solidario al sólido rígido (rotaciones intrínsecas), lo que se denomina un sistema de referencia local.

Existen multitud de maneras distintas de expresar los ángulos de Euler, dependiendo del orden de los ejes sobre los cuales se realizan las rotaciones, y de si el sistema de referencia es local o global. La definición usada por los sensores xsens es la composición de rotaciones alrededor de los ejes XYZ en ese orden y un sistema de referencia global (fijo a la Tierra), en el que la ausencia de rotación equivale al vector Z paralelo a la línea que une la posición del sensor con el centro de la Tierra y sentido ascendente, y el vector X en dirección al norte magnético.

Los ángulos proporcionados por el sensor son:

- $\phi = roll$ = rotación alrededor del eje X $[-\frac{\pi}{2}, \frac{\pi}{2}]$
- $\theta = pitch$ = rotación alrededor del eje Y $[-\frac{\pi}{4}, \frac{\pi}{4}]$
- $\psi = yaw$ = rotación alrededor del eje Z $[-\frac{\pi}{2}, \frac{\pi}{2}]$

El uso de ángulos de Euler presenta un par de problemas:

- Si θ se extiende al intervalo $[-\frac{\pi}{2}, \frac{\pi}{2}]$ en vez de restringirse a $[-\frac{\pi}{4}, \frac{\pi}{4}]$, la descripción de la rotación no es única. En este caso existen dos posibles soluciones para cada orientación, por lo que los datos que proporciona el sensor no son suficientes para diferenciar entre un estado de rotación u otro.
- Cuando θ se acerca al valor $\pm\frac{\pi}{4}$ existe una singularidad matemática causada por la infinitud de valores que pueden adquirir ϕ y ψ en dicho caso. Esta situación es la que se conoce con el nombre de *gimbal lock*.

Estos problemas no están presentes en los demás modos de salida del sensor.

5.1.2. Matriz de rotación

Una matriz de rotación es una matriz usada para expresar una rotación en un espacio euclídeo. Si se supone un sistema generador S con un conjunto de vectores base B del espacio vectorial V :

$$B = \{\vec{i}, \vec{j}, \vec{k}\}, \quad \vec{i} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \vec{j} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \vec{k} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$V = S(\vec{i}, \vec{j}, \vec{k})$$

Se somete al sistema de coordenadas a una rotación R . Los vectores base sufren una transformación que se puede expresar con respecto al sistema de referencia original de la siguiente forma:

$$\vec{i}' = \alpha_{11}\vec{i} + \alpha_{21}\vec{j} + \alpha_{31}\vec{k} = \begin{bmatrix} \alpha_{11} \\ \alpha_{21} \\ \alpha_{31} \end{bmatrix}$$

$$\vec{j}' = \alpha_{12}\vec{i} + \alpha_{22}\vec{j} + \alpha_{32}\vec{k} = \begin{bmatrix} \alpha_{12} \\ \alpha_{22} \\ \alpha_{32} \end{bmatrix}$$

$$\vec{k}' = \alpha_{13}\vec{i} + \alpha_{23}\vec{j} + \alpha_{33}\vec{k} = \begin{bmatrix} \alpha_{13} \\ \alpha_{23} \\ \alpha_{33} \end{bmatrix}$$

donde α_{ij} son las componentes de la base rotada expresadas con respecto a la base original. Se tiene un vector cualquiera en la base original:

$$\vec{v} = a\vec{i} + b\vec{j} + c\vec{k} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Tras la rotación las componentes del vector expresadas conforme a la nueva base no variarán debido a que se mueve solidario al sistema de referencia. Por lo tanto el vector rotado es el siguiente:

$$\vec{v}' = a\vec{i}' + b\vec{j}' + c\vec{k}'$$

Sustituyendo los vectores $\vec{i}', \vec{j}', \vec{k}'$ por sus expresiones respecto a la base original:

$$\vec{v}' = a \begin{bmatrix} \alpha_{11} \\ \alpha_{21} \\ \alpha_{31} \end{bmatrix} + b \begin{bmatrix} \alpha_{12} \\ \alpha_{22} \\ \alpha_{32} \end{bmatrix} + c \begin{bmatrix} \alpha_{13} \\ \alpha_{23} \\ \alpha_{33} \end{bmatrix} = \begin{bmatrix} a\alpha_{11} + b\alpha_{12} + c\alpha_{13} \\ a\alpha_{21} + b\alpha_{22} + c\alpha_{23} \\ a\alpha_{31} + b\alpha_{32} + c\alpha_{33} \end{bmatrix}$$

Este nuevo vector puede expresarse como el producto de una matriz por un vector:

$$\vec{v}' = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = R\vec{v}$$

Dicha matriz R es la denominada matriz de rotación. Se comprueba que puede interpretarse como una matriz cuyos elementos son las componentes de los vectores base del sistema de coordenadas rotado expresados con respecto al sistema de coordenadas original.

El uso de matrices de rotación evita los problemas que presentan los ángulos de Euler; Proporcionan una descripción biunívoca del estado de rotación del sensor, y además no presentan el problema del *gimbal lock*.

5.1.3. Cuaternion

Los cuaterniones son una extensión de los números complejos ideada por el matemático irlandés William Rowand Hamilton con el objetivo de poder utilizar el análisis complejo en un espacio de 3 dimensiones. Un cuaternion es la combinación lineal de cuatro cantidades $\{1, i, j, k\}$, donde cada una de las cantidades $\{i, j, k\}$ es la raíz cuadrada de -1 , de tal forma que se cumplen las siguientes propiedades:

$$i^2 = j^2 = k^2 = ijk = -1$$

La expresión general de un cuaternion es la siguiente:

$$q = w + xi + yj + zk$$

donde w, x, y y z son números reales. Los cuaterniones satisfacen las leyes conmutativa y asociativa de la suma, la ley asociativa de la multiplicación, las leyes distributivas de la multiplicación respecto a la suma y la existencia de los elementos neutros para la suma y la multiplicación. Una propiedad importante de los cuaterniones es que no satisfacen la propiedad conmutativa de la multiplicación.

Propiedades de los cuaterniones	
Comutativa respecto a la suma	$q_1 + q_2 = q_2 + q_1$
Asociativa respecto a la suma	$q_1 + (q_2 + q_3) = (q_1 + q_2) + q_3$
Asociativa respecto a la multiplicación	$q_1(q_2q_3) = (q_1q_2)q_3$
Distributiva de la multiplicación respecto a la suma	$q_1(q_2 + q_3) = q_1q_2 + q_1q_3$ $(q_1 + q_2)q_3 = q_1q_3 + q_2q_3$
Elemento neutro de la suma	$q + 0 = q$
Elemento neutro de la multiplicación	$q1 = 1q = q$
No conmutatividad de la multiplicación	$q_1q_2 \neq q_2q_1$

Cuadro 5.1: Propiedades de los cuaterniones

Los cuaterniones presentan las mismas ventajas que las matrices de rotación en cuanto a unicidad y ausencia de *gimbal lock*. Pero además son superiores a las matrices en los siguientes aspectos:

- Su representación en memoria es más compacta que la de las matrices (4 números frente a 9 necesarios para la matriz).
- Se puede construir fácilmente un cuaternion a partir de un eje y un ángulo, y viceversa. Estas operaciones son más complejas para matrices de rotación y ángulos de Euler.
- Mayor estabilidad numérica de los cuaterniones. Tras la composición de varias rotaciones en un ordenador necesariamente se van a acumular errores de redondeo. Para que los cuaterniones y matrices representen una rotación deben ciertas propiedades. En concreto, los cuaterniones tienen que ser unitarios y la matriz de rotación tiene que ser ortogonal. Es bastante más fácil normalizar un cuaternion para que vuelva a representar una rotación que recomponer una matriz para que vuelva a ser ortogonal.

- Además con los cuaterniones es sencillo componer una interpolación esférica (llamada *slerp - spherical linear interpolation*) para producir una rotación suave a lo largo del tiempo.

5.1.4. Solución elegida: cuaterniones

Por la gran cantidad de ventajas que presentan los cuaterniones con respecto a las demás formas de representar una rotación, éstos serán los escogidos para registrar el estado de orientación de los sensores y realizar los cálculos matemáticos.

5.2. Utilización de cuaterniones para la representación de rotaciones de un sólido rígido

En esta sección se verá cómo describir rotaciones simples y composiciones de rotaciones de un sólido rígido mediante el uso de cuaterniones.

Sea un vector unitario:

$$\vec{e} = e_x i + e_y j + e_z k, \quad \|e\| = \sqrt{e_x^2 + e_y^2 + e_z^2} = 1 \quad (5.1)$$

Se definirá el cuaternion de rotación con ángulo θ sobre el eje definido por el vector \vec{e} :

$$q(\theta, \vec{e}) = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right) e_x i + \sin\left(\frac{\theta}{2}\right) e_y j + \sin\left(\frac{\theta}{2}\right) e_z k \quad (5.2)$$

Este cuaternion representa la orientación de un sólido tras sufrir una rotación de ángulo θ alrededor del eje definido por el vector \vec{e} . Se puede descomponer el cuaternion como suma de un número real y un cuaternion imaginario puro unitario multiplicado por otro número real:

$$q(\theta, \vec{e}) = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right) e \quad (5.3)$$

Donde e es el cuaternion imaginario puro (parte real nula) cuyas componentes se corresponden a las del vector unitario \vec{e} que define el eje de rotación.

Por cuestión de comodidad definimos las siguientes variables:

$$c_i = \cos\left(\frac{\theta_i}{2}\right) \quad (5.4)$$

$$s_i = \sin\left(\frac{\theta_i}{2}\right) \quad (5.5)$$

De tal forma que ahora el cuaternion se escribirá de la siguiente manera:

$$q(\theta_i, \vec{e}_i) = c_i + s_i e_i \quad (5.6)$$

Si se tienen en cuenta las propiedades del cuadro 5.1, el producto de dos cuaterniones se puede expresar de esta forma:

$$q_1 q_2 = (c_1 + s_1 e_1)(c_2 + s_2 e_2) = c_1 c_2 + c_1 s_2 e_2 + c_2 s_1 e_1 + s_1 s_2 e_1 e_2 \quad (5.7)$$

En donde:

$$\begin{aligned} e_1 e_2 &= (e_{1_x} i + e_{1_y} j + e_{1_z} k)(e_{2_x} i + e_{2_y} j + e_{2_z} k) = \\ &= e_{1_x} i(e_{2_x} i + e_{2_y} j + e_{2_z} k) + e_{1_y} j(e_{2_x} i + e_{2_y} j + e_{2_z} k) + e_{1_z} k(e_{2_x} i + e_{2_y} j + e_{2_z} k) = \\ &= -(e_{1_x} e_{2_x} + e_{1_y} e_{2_y} + e_{1_z} e_{2_z}) + i(e_{1_y} e_{2_z} + e_{1_z} e_{2_y}) + j(-e_{1_x} e_{2_z} + e_{1_z} e_{2_x}) + k(e_{1_x} e_{2_y} + e_{1_y} e_{2_x}) \end{aligned} \quad (5.8)$$

Se definirán las siguientes operaciones con cuaterniones imaginarios puros (parte real nula):

Definición 5.2.1. Sean dos cuaterniones imaginarios puros q_1 y q_2 tales que:

$$q_1 = x_1 i + y_1 j + z_1 k$$

$$q_2 = x_2 i + y_2 j + z_2 k$$

Se define el operador producto escalar (\cdot) como:

$$q_1 \cdot q_2 = x_1 x_2 + y_1 y_2 + z_1 z_2$$

Este operador presenta las mismas propiedades que el mismo operador para vectores de 3 dimensiones:

$$q_1 \cdot q_2 = q_2 \cdot q_1$$

$$q_1 \cdot q_1 = \|q_1\|^2$$

$$q_1 \cdot q_2 = 0 \iff q_1 \perp q_2$$

Definición 5.2.2. Sean dos cuaterniones imaginarios puros q_1 y q_2 tales que:

$$q_1 = x_1 i + y_1 j + z_1 k$$

$$q_2 = x_2 i + y_2 j + z_2 k$$

Se define el operador producto vectorial (\times) como:

$$q_1 \times q_2 = \begin{bmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{bmatrix} = (y_1 z_2 - z_1 y_2)i + (z_1 x_2 - x_1 z_2)j + (x_1 y_2 - y_1 x_2)k$$

Este operador presenta las mismas propiedades que el mismo operador para vectores de 3 dimensiones:

$$q_1 \times q_2 = -(q_2 \times q_1)$$

$$q_1 \times q_2 = 0 \iff q_1 \parallel q_2$$

$$q_1 \times q_2 = q_3 : q_3 \perp q_1 \wedge q_3 \perp q_2$$

Utilizando la definición de estos operadores se puede simplificar la ecuación (5.8):

$$e_1 e_2 = -e_1 \cdot e_2 + e_1 \times e_2 \quad (5.9)$$

Por lo tanto, sustituyendo (5.9) en la ecuación (5.7):

$$q_1 q_2 = c_1 c_2 + c_1 s_2 e_2 + c_2 s_1 e_1 + s_1 s_2 (-e_1 \cdot e_2 + e_1 \times e_2)$$

$$q_1 q_2 = c_1 c_2 - s_1 s_2 (e_1 \cdot e_2) + c_1 s_2 e_2 + c_2 s_1 e_1 + s_1 s_2 (e_1 \times e_2) \quad (5.10)$$

Definición 5.2.3. El conjugado de un cuaternión $q = c + se$ se define como el cuaternión resultado de negar la parte imaginaria. Se representará de la siguiente manera:

$$\text{conj}(q) = q^* = c - se \quad (5.11)$$

Dado a que se va a trabajar siempre con cuaterniones unitarios (también llamados *versores*), se podrá asumir lo siguiente:

$$\|q\| = 1, \quad q^{-1} = \frac{q^*}{\|q\|^2} = q^* \quad (5.12)$$

De tal forma que:

$$qq^* = q^*q = 1 \quad (5.13)$$

5.2.1. Rotación de un vector alrededor de un eje y un ángulo dados

Se puede realizar la rotación de un vector \vec{p} alrededor de un eje \vec{e} y un ángulo θ mediante la siguiente operación:

$$p' = qpq^*, \quad q = q(\theta, \vec{e}) \quad (5.14)$$

donde p es el cuaternión asociado al vector \vec{p} , que se define como:

$$\vec{p} = p_x \vec{i} + p_y \vec{j} + p_z \vec{k} \Rightarrow p = p_x i + p_y j + p_z k \quad (5.15)$$

Se comprueba que en efecto la ecuación (5.14) es cierta. Desarrollando la expresión para un cuaternión genérico q y un vector expresado como cuaternion Pp , donde P es el módulo del vector y p es el cuaternion asociado al vector normalizado:

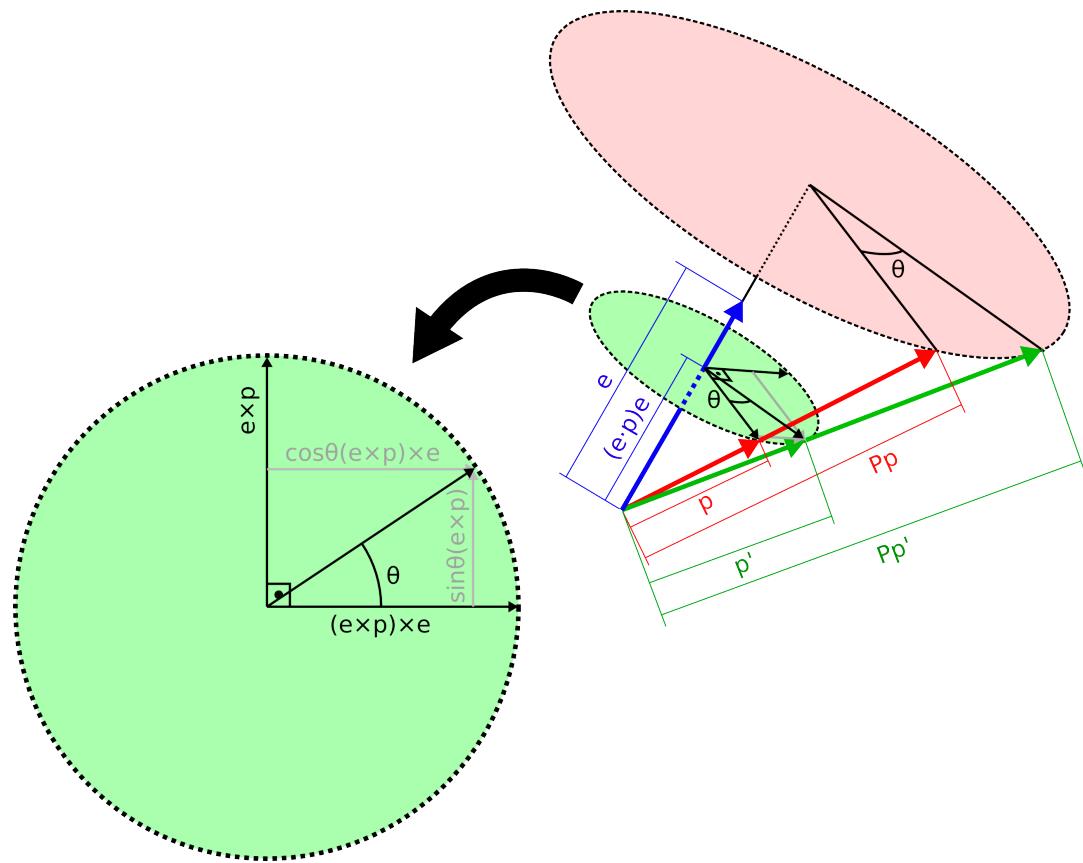
$$q(Pp)q^* = P(qpq^*) = P(c+se)p(c-se) = P(cp+sep)(c-se) = P(c^2p - cspe + csep - s^2ep)$$

Se aplican las propiedades de los cuaterniones unitarios vistas anteriormente:

$$q(Pp)q^* = P(c^2p + 2cs(e \times p) + s^2(e \cdot p)e - s^2(e \times p) \times e) \quad (5.16)$$

Se le puede asignar a cada término un significado geométrico:

- $(e \cdot p)e$ es la proyección de p sobre e

Figura 5.1: Rotación de un vector mediante la operación $q_1 p q_1^*$

- $(e \times p)$ es un vector perpendicular a e y a p . Formaría una supuesta coordenada y en el círculo de giro.
- $(e \times p) \times e$ es un vector perpendicular al anterior, cuyo origen podemos situar al final de $(e \cdot p)e$ y su final, en el mismo punto que p . Sería la coordenada x del círculo de giro.

En la figura 5.1 se muestra más claramente el significado de cada término.

Es posible descomponer así el término $c^2 p$ como suma de dos de estos vectores:

$$c^2 p = c^2(e \cdot p)e + c^2(e \times p) \times e$$

Sustituyendo el resultado en (5.16):

$$q(Pp)q^* = P(c^2(e \cdot p)e + c^2(e \times p) \times e + 2cs(e \times p) + s^2(e \cdot p)e - s^2(e \times p) \times e)$$

Reorganizando términos:

$$q(Pp)q^* = P((c^2 + s^2)(e \cdot p)e + (c^2 - s^2)(e \times p) \times e + 2cs(e \times p))$$

Si se aplican igualdades trigonométricas a esta expresión, teniendo en cuenta que $c = \cos(\frac{\theta}{2})$ y $s = \sin(\frac{\theta}{2})$:

$$q(Pp)q^* = P((e \cdot p)e + \cos\theta(e \times p) \times e + \sin\theta(e \times p))$$

De la figura 5.1 se deduce que $p' = (e \cdot p)e + \cos \theta(e \times p) \times e + \sin \theta(e \times p)$, y por lo tanto se puede concluir que:

$$q(Pp)q^* = Pp'$$

Por lo que para un vector cualquiera \vec{v} se cumple que $qvq^* = v'$, donde v' es el cuaternion asociado al vector rotado.

Si se realiza una multiplicación a la izquierda por q_1^* y por la derecha por q_1 a cada término de la ecuación (5.14) (lo que equivale a realizar una rotación dada por el cuaternion q_1^*):

$$q_1^*p'q_1 = q_1^*(q_1pq_1)q_1^* \quad (5.17)$$

$$q_1^*p'q_1 = p \quad (5.18)$$

se obtiene el punto inicial, de lo que se deduce que el conjugado del cuaternion representa una rotación inversa a la del cuaternion original:

$$q = q(\theta, \vec{e}) \Rightarrow q^* = q(\theta, -\vec{e}) = q(-\theta, \vec{e}) \quad (5.19)$$

5.2.2. Composición de rotaciones en coordenadas extrínsecas

El cuaternion de rotación definido por $q(\theta, \vec{e})$ representa una rotación alrededor de un eje \vec{e} fijo al sistema de referencia global.

Realizando una nueva rotación a un vector que ha sufrido una rotación definida por $q_1 = q(\theta_1, \vec{e}_1)$, se obtendrá una rotación compuesta por una primera rotación seguida de otra rotación caracterizada por $q_2 = q(\theta_2, \vec{e}_2)$:

$$\begin{aligned} \text{Primera rotación: } & p' = q_1pq_1^*, \quad q_1 = q(\theta_1, \vec{e}_1) \\ \text{Segunda rotación: } & p'' = q_2p'q_2^*, \quad q_2 = q(\theta_2, \vec{e}_2) \end{aligned}$$

$$p'' = q_2(q_1pq_1^*)q_2^* = (q_2q_1)p(q_1^*q_2^*) = q_{12}pq_{12}^* \quad (5.20)$$

Se puede expresar la composición de dos rotaciones como un nuevo cuaternion que resulta de la multiplicación en orden inverso de los cuaterniones que definen las dos rotaciones:

$$q_{12} = q_2q_1 \quad (5.21)$$

De aquí se deduce que el conjugado del producto de dos cuaterniones es el producto de los conjugados en orden inverso:

$$q_{12} = (q_2q_1)^* = q_1^*q_2^* \quad (5.22)$$

De forma análoga, para n cuaterniones:

$$q_{12\dots(n-1)n} = q_nq_{n-1}\dots q_2q_1 \quad (5.23)$$

$$(q_{12\dots(n-1)n})^* = (q_nq_{n-1}\dots q_2q_1)^* = q_1^*q_2^*\dots q_{n-1}^*q_n^* \quad (5.24)$$

5.2.3. Composición de rotaciones en coordenadas intrínsecas

Para representar una rotación alrededor de un eje expresado en el sistema de referencia local del sólido, se tendrá que realizar la construcción del cuaternion teniendo en cuenta que dicho eje ha sufrido la misma rotación que el sólido con respecto al sistema de referencia global. Supongamos un sólido que ha sufrido una rotación inicial representada por el cuaternion q_1 . Se quiere realizar una rotación de ángulo θ_2 sobre un eje e_2^L en coordenadas locales. Dicho eje expresado en coordenadas globales será:

$$e_2^G = q_1 e_2^L q_1^*$$

Teniendo en cuenta la ecuación (5.6), el cuaternion de rotación asociado al ángulo θ_2 y el vector local e_2^L después de que el sólido haya sufrido una rotación definida por el cuaternion q_1 será:

$$\begin{aligned} q_{2_{q_1}} &= q(\theta_2, \vec{e}_2^G) = \cos\left(\frac{\theta_2}{2}\right) + \sin\left(\frac{\theta_2}{2}\right) e_2^G \\ q_{2_{q_1}} &= c_2 + s_2(q_1 e_2^L q_1^*) \end{aligned} \quad (5.25)$$

5.2.4. Relación entre rotaciones intrínsecas y extrínsecas

El algoritmo de obtención de los ángulos de Euler que se explicará más adelante proporciona una solución en el caso de ejes solidarios al sensor, esto es, en un sistema de coordenadas intrínsecas. Es posible obtener con este algoritmo soluciones en un sistema extrínseco que pueden ser de utilidad en ciertas aplicaciones. A continuación se demostrará que una rotación compuesta por varias rotaciones en el sistema de coordenadas intrínseco del sólido rígido se corresponde a la composición de rotaciones en el sistema extrínseco realizadas en orden inverso.

Se define un cuaternion de rotación asociado al eje local \vec{e}_2^L después de haber sufrido una rotación definida por el cuaternion $q_1 = q(\theta_1, \vec{e}_1)$ como:

$$q_{2_{q_1}} = q(\theta_2, \vec{e}_2^L), \quad \vec{e}_2^L = (q_1 e_2 q_1^*) \vec{v} \quad (5.26)$$

Se va a suponer sin pérdida de generalidad que la primera rotación se ha realizado desde una posición en la que coinciden los sistemas local y global ¹, por lo que:

$$q_1^G = q_1^L = q_1 \quad (5.27)$$

donde q_1^G es la rotación alrededor de un eje en coordenadas globales y q_1^L la rotación en coordenadas locales.

Para demostrar la afirmación de partida se tendrá que demostrar la veracidad de la siguiente igualdad:

$$q_2^L q_1 = q_1 q_2^G \quad (5.28)$$

¹Por definición el origen del cuaternion de orientación ($q = 1$) del sensor es la orientación donde coinciden el sistema global y local. Por ello, el eje de rotación en los dos sistemas coincidirá y por lo tanto el cuaternion de rotación será el mismo.

Se reordenará la igualdad para que los cálculos sean más sencillos:

$$q_2^L q_1 = q_1 q_2^G \iff q_2^L = q_1 q_2^G q_1^* \quad (5.29)$$

Desarrollo del lado izquierdo de la igualdad:

$$q_2^L = c_2 + s_2(q_1 e_2 q_1^*) \quad (5.30)$$

$$\begin{aligned} q_1 e_2 q_1^* &= (c_1 + s_1 e_1) e_2 (c_1 - s_1 e_1) = (c_1 e_2 + s_1 e_1 e_2) (c_1 - s_1 e_1) = \\ &= c_1^2 e_2 + s_1 c_1 e_1 e_2 - c_1 s_1 e_2 e_1 - s_1^2 e_1 e_2 e_1 = c_1^2 e_2 + s_1 c_1 (e_1 e_2 - e_2 e_1) - s_1^2 e_1 e_2 e_1 \end{aligned} \quad (5.31)$$

De (5.9) se tiene lo siguiente:

$$\begin{aligned} e_1 e_2 &= -e_1 \cdot e_2 + e_1 \times e_2 \\ e_2 e_1 &= -e_2 \cdot e_1 + e_2 \times e_1 = -e_1 \cdot e_2 - e_1 \times e_2 \\ e_1 e_2 - e_2 e_1 &= 2(e_1 \times e_2) \end{aligned} \quad (5.32)$$

$$e_1 e_2 e_1 = (-e_1 \cdot e_2 + e_1 \times e_2) e_1 = (-e_1 \cdot e_2) e_1 + (e_1 \times e_2) e_1$$

De esta ecuación:

$$\begin{aligned} (e_1 \times e_2) e_1 &= -(e_1 \times e_2) \cdot e_1 + (e_1 \times e_2) \times e_1 \\ (e_1 \times e_2) \text{ será un vector perpendicular a } e_1, \text{ por lo que } (e_1 \times e_2) \cdot e_1 &= 0: \end{aligned}$$

$$(e_1 \times e_2) e_1 = (e_1 \times e_2) \times e_1$$

Por lo tanto:

$$e_1 e_2 e_1 = -(e_1 \cdot e_2) e_1 + (e_1 \times e_2) \times e_1 \quad (5.33)$$

Sustituyendo en (5.31):

$$\begin{aligned} q_1 e_2 q_1^* &= c_1^2 e_2 + s_1 c_1 (e_1 e_2 - e_2 e_1) - s_1^2 e_1 e_2 e_1 = \\ &= c_1^2 e_2 + 2s_1 c_1 (e_1 \times e_2) - s_1^2 ((-e_1 \cdot e_2) e_1 + (e_1 \times e_2) \times e_1) \\ q_1 e_2 q_1^* &= c_1^2 e_2 + 2s_1 c_1 (e_1 \times e_2) + s_1^2 (e_1 \cdot e_2) e_1 - s_1^2 (e_1 \times e_2) \times e_1 \end{aligned} \quad (5.34)$$

Finalmente, sustituyendo en (5.30):

$$q_2^L = c_2 + s_2(c_1^2 e_2 + 2s_1 c_1 (e_1 \times e_2) + s_1^2 (e_1 \cdot e_2) e_1 - s_1^2 (e_1 \times e_2) \times e_1)$$

$$q_2^L = c_2 + s_2 c_1^2 e_2 + 2s_1 s_2 c_1 (e_1 \times e_2) + s_1^2 s_2 (e_1 \cdot e_2) e_1 - s_1^2 s_2 (e_1 \times e_2) \times e_1 \quad (5.35)$$

Ahora se procederá a desarrollar el lado derecho de la igualdad (5.29):

$$\begin{aligned}
q_1 q_2^G q_1^* &= (c_1 + s_1 e_1)(c_2 + s_2 e_2)(c_1 - s_1 e_1) = \\
&= (c_1 c_2 + s_2 c_1 e_2 + s_1 c_2 e_1 + s_1 s_2 e_1 e_2)(c_1 - s_1 e_1) = \\
&= c_1^2 c_2 + s_2 c_1^2 e_2 + s_1 c_1 c_2 e_1 + s_1 s_2 c_1 e_1 e_2 - s_1 c_1 c_2 e_1 - s_1 s_2 c_1 e_2 e_1 - s_1^2 c_2 e_1 e_1 - s_1^2 s_2 e_1 e_2 e_1
\end{aligned}$$

Como $\|e_1\| = 1$, se tendrá que:

$$e_1 e_1 = -e_1 \cdot e_1 + e_1 \times e_1 = -1 \quad (5.36)$$

Sustituyendo y reorganizando:

$$q_1 q_2^G q_1^* = c_2 + s_2 c_1^2 e_2 + s_1 s_2 c_1 (e_1 e_2 - e_2 e_1) - s_1^2 s_2 e_1 e_2 e_1$$

De (5.32) y (5.33):

$$q_1 q_2^G q_1^* = c_2 + s_2 c_1^2 e_2 + 2s_1 s_2 c_1 (e_1 \times e_2) - s_1^2 s_2 (-e_1 \cdot e_2) e_1 + (e_1 \times e_2) \times e_1$$

$$q_1 q_2^G q_1^* = c_2 + s_2 c_1^2 e_2 + 2s_1 s_2 c_1 (e_1 \times e_2) + s_1^2 s_2 (e_1 \cdot e_2) e_1 - s_1^2 s_2 (e_1 \times e_2) \times e_1 \quad (5.37)$$

Finalmente se compara (5.35) con (5.37):

$$q_2^L = c_2 + s_2 c_1^2 e_2 + 2s_1 s_2 c_1 (e_1 \times e_2) + s_1^2 s_2 (e_1 \cdot e_2) e_1 - s_1^2 s_2 (e_1 \times e_2) \times e_1$$

$$q_1 q_2^G q_1^* = c_2 + s_2 c_1^2 e_2 + 2s_1 s_2 c_1 (e_1 \times e_2) + s_1^2 s_2 (e_1 \cdot e_2) e_1 - s_1^2 s_2 (e_1 \times e_2) \times e_1$$

Se puede ver que los términos a la derecha de la igualdad son exactamente los mismos, por lo que se tiene que:

$$q_2^L = q_1 q_2^G q_1^* \quad (5.38)$$

Y por lo tanto es verdad la afirmación de partida. Una rotación q_2^L alrededor de un eje local tras una rotación q_1 es la misma que la rotación global q_2^G seguida de la rotación q_1 :

$$q_2^L q_1 = q_1 q_2^G \quad (5.39)$$

Ahora se va a suponer que el cuaternion q_1 está compuesto de otras dos rotaciones, que se van a expresar de forma global y local según la ecuación (5.39):

$$q_1 = q_{1b}^L q_{1a} = q_{1a} q_{1b}^G$$

Sustituyendo en (5.39):

$$q_2^L q_{1b}^L q_{1a} = q_{1a} q_{1b}^G q_2^G$$

Renombrando los términos:

$$q_3^L q_2^L q_1 = q_1 q_2^G q_3^G$$

Aplicando el mismo método de forma sucesiva se puede generalizar el resultado a n rotaciones:

$$q_n^L \cdots q_3^L q_2^L q_1 = q_1 q_2^G q_3^G \cdots q_n^G \quad (5.40)$$

Se concluye que la composición de n rotaciones en coordenadas locales es la misma que la composición de n rotaciones en coordenadas globales realizadas en orden inverso.

5.2.5. Orientación relativa entre dos sólidos

Se tienen dos sólidos con sendos sistemas de coordenadas S_1 y S_2 . Definiremos la rotación relativa del sólido 1 sobre el sólido 2 como la rotación existente entre sus sistemas de coordenadas.

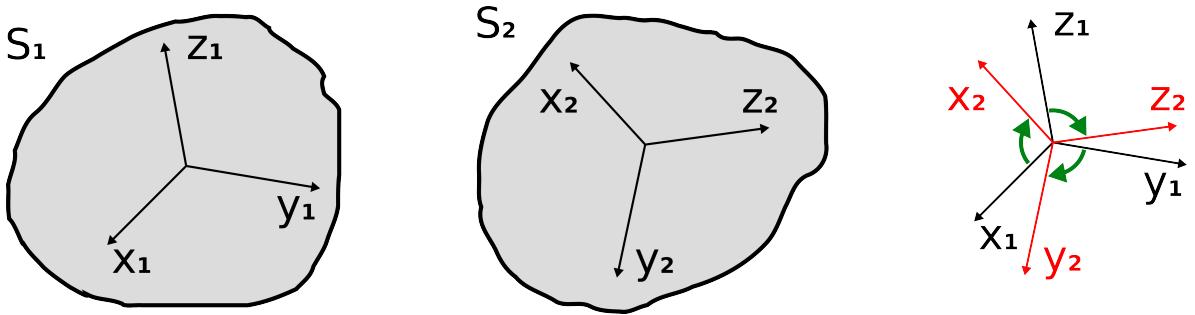


Figura 5.2: Orientación relativa entre dos sólidos rígidos

Otra forma de verlo es que la rotación relativa es la rotación que tendría el sólido 2 si el sistema de coordenadas del sólido 1 fuera el global. De esta forma se podrá calcular el cuaternion que define la orientación relativa entre los dos sólidos aplicando una rotación a ambos de forma que el sistema de coordenadas del sólido 1 coincida con el global. Se llamará q_1 al cuaternion que define la orientación del sólido 1 y q_2 al que define la orientación del sólido q_2 . Si se aplica una rotación q_1^* a ambos, esto es, una rotación inversa a la del sólido 1, se obtiene lo siguiente:

$$\text{Sólido 1: } q'_1 = q_1^* q_1 = 1$$

$$\text{Sólido 2: } q'_2 = q_1^* q_2 = q_{S_2 S_1}$$

donde $q_{S_2 S_1}$ es la orientación relativa del sólido 2 con respecto al sólido 1. Generalizando este resultado se obtiene una expresión general para hallar la rotación relativa de un sólido A con respecto a un sólido B :

$$q_{AS_B} = q_B^* q_A \quad (5.41)$$

Esta expresión tiene interés a lo hora del cálculo de los ángulos de Euler en la articulación de un brazo articulado si se conocen los cuaterniones de rotación globales de los dos eslabones que forman la articulación.

5.3. Cálculo de la posición del brazo

Una vez sean conocidas las orientaciones de los sensores, sabiendo que dichos sensores se encuentran solidarios al brazo y cuyo eje x se ha orientado en sentido longitudinal de acuerdo a la figura 5.3, resulta muy sencillo obtener las posiciones de cada punto del brazo sabiendo su posición inicial.

Un brazo articulado simple se definirá por un conjunto de 3 longitudes $\{L_1, L_2, L_3\}$ que representan las distancias hombro-codo, codo-muñeca y muñeca-punta de dedo respectivamente, y por 3 cuaterniones $\{q_1, q_2, q_3\}$ que indican el estado de orientación de brazo, antebrazo y mano respectivamente. Dado que para los sensores xsens la posición inicial es aquella en la que su eje x apunta al norte magnético, y el eje z es vertical hacia arriba, se establecerá como posición inicial del brazo aquella en la que el eje longitudinal de todos los eslabones apunta hacia el norte magnético. En esta posición se cumple que:

$$q_1 = q_2 = q_3 = 1$$

Además como el eje x se corresponde con el elemento base i , los puntos iniciales de hombro, codo, muñeca y punta de dedo son:

$$p_0^i = 0 \quad p_1^i = L_1 i \quad p_2^i = p_1^i + L_2 i = (L_1 + L_2) i \quad p_3^i = p_2^i + L_3 i = (L_1 + L_2 + L_3) i$$

Se tomará como origen de coordenadas global la posición del hombro p_0 . Cada eslabón tendrá por origen de coordenadas local la articulación inmediatamente anterior.

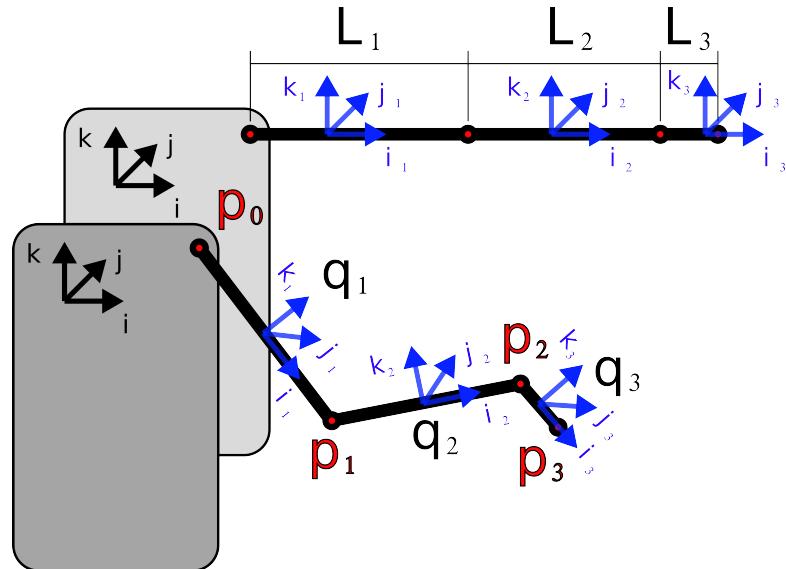


Figura 5.3: Posiciones del brazo inicial y genérica. L_1, L_2 y L_3 son las longitudes de brazo, antebrazo y mano respectivamente, y q_1, q_2 y q_3 son sus respectivas orientaciones. p_0, p_1, p_2 y p_3 son las posiciones de hombro, codo, muñeca y punta de dedo respectivamente.

Dada una posición genérica del brazo definida por el conjunto de cuaterniones $\{q_i\} = \{q_1, q_2, q_3\}$, las posiciones de cada articulación se pueden calcular de esta forma:

$$\text{Hombro: } p_0 = 0$$

$$\text{Codo: } p_1 = q_1(L_1 i)q_1^*$$

$$\text{Muñeca: } p_2 = p_1 + q_2(L_2 i)q_2^*$$

$$\text{Punta de dedo: } p_3 = p_2 + q_3(L_3 i)q_3^*$$

Además se pueden obtener las posiciones local y global de un punto cualquiera r (no necesariamente articulación) perteneciente a un eslabón K , del que se conoce su posición local inicial r_L^i :

$$r_L = q_K r_L^i q_K^*$$

$$r_G = p_{K-1} + r_L = p_{K-1} + q_K r_L^i q_K^*$$

donde p_{K-1} es la posición de la articulación $K - 1$. Se puede generalizar este resultado a la posición de un punto r cualquiera situado en la articulación K de un brazo articulado, no necesariamente humano, con N eslabones, definido por las longitudes de los eslabones $\{L_i\} = \{L_1, L_2, \dots, L_N\}$ y orientaciones $\{q_i\} = \{q_1, q_2, \dots, q_N\}$:

$$r_G = p_{K-1} + q_K r_L^i q_K^* = p_{K-2} + q_{K-1}(L_{K-1} i)q_{K-1}^* + q_K r_L^i q_K^* = \dots$$

$$r_G = q_K r_L^i q_K^* + \sum_{m=1}^{K-1} q_m(L_m i)q_m^* \quad (5.42)$$

donde r_L^i es la posición local inicial de r dentro del eslabón.

5.4. Obtención de los ángulos de Euler a partir del cuaternión de orientación

En ciertas aplicaciones, como por ejemplo cuando se pretende mover las articulaciones de un brazo robótico, resultan más útiles los ángulos de Euler que el cuaternión o la matriz de rotación. Normalmente los brazos robóticos presentan articulaciones de un grado de libertad en las que la característica que define el estado de la articulación es el ángulo. Se puede definir un estado de rotación mediante tres ángulos –ángulos de Euler– sobre tres los tres ejes principales x, y, z . A estas rotaciones se les denomina rotaciones principales. En este apartado se verá como realizar el cálculo de los ángulos de Euler a partir del cuaternión de rotación.

Para la obtención de los ángulos de Euler se usará la convención para los ángulos $\langle \psi : yaw(Z), \theta : pitch(Y), \phi : roll(X) \rangle$ en un sistema de coordenadas intrínsecas, lo que equivale a $\langle \phi, \theta, \psi \rangle$ en coordenadas extrínsecas. Se supondrá el sistema de coordenadas local situado sobre la superficie de una esfera de radio unitario, con un sistema de coordenadas global, de tal modo que el eje x del sistema de coordenadas local sea siempre normal a la superficie de la esfera y el eje z se tomará por el momento apuntando al norte de la esfera y paralelo a su superficie. Se tomará como origen de coordenadas (O) la posición en que coinciden los sistemas de coordenadas local y global.

Se mueve un punto desde el origen de coordenadas O hasta un punto P , tal como se muestra en la figura 5.4. El punto P tiene por coordenadas:

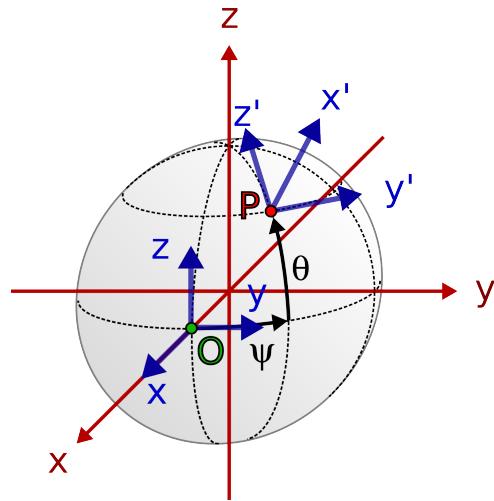


Figura 5.4: Sistema de coordenadas local sobre la esfera

$$P = \cos \psi \cos \theta i + \sin \psi \cos \theta j + \sin \theta k$$

Como la esfera es de radio unitario, resulta obvio que el punto P tiene las mismas coordenadas que el vector i' perteneciente al sistema de coordenadas local, si éste tuviera por origen el centro de la esfera.

$$i' = \cos \psi \cos \theta i + \sin \psi \cos \theta j + \sin \theta k \quad (5.43)$$

Utilizando las componentes del vector i' resulta sencillo obtener los ángulos ψ y θ :

$$\frac{i'_y}{i'_x} = \frac{\sin \psi \cos \theta}{\cos \psi \cos \theta} = \tan \psi \iff \psi = \arctan \left(\frac{i'_y}{i'_x} \right) \quad (5.44)$$

$$i'_z = \sin \theta \iff \theta = \arcsin i'_z \quad (5.45)$$

A continuación se realiza la tercera rotación sobre el eje x' . Los ejes y'' y z'' así obtenidos permanecerán tangentes a la superficie de la esfera, mientras el eje x'' coincidirá con x' . Esta situación final se muestra en la figura 5.5.

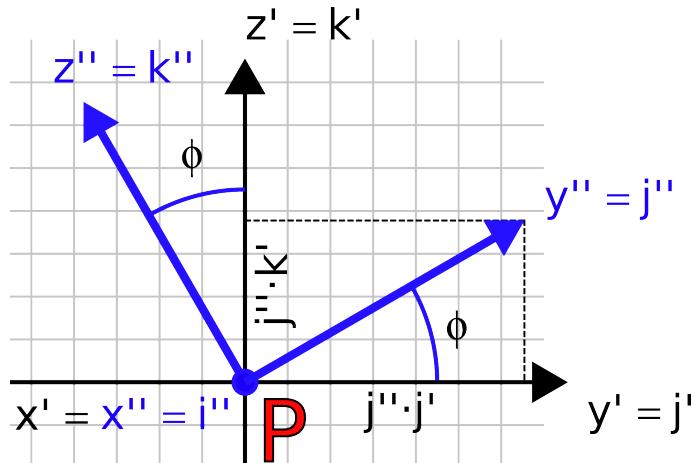


Figura 5.5: Tercera rotación en el sistema de coordenadas local sobre el punto P de la esfera

El estado del sistema de coordenadas tras la tercera rotación es el que va a ser dado por el cuaternion de orientación que nos proporciona el sensor, al que se denominará q_s . Sin embargo, para el cálculo del último de los ángulos (ϕ) se necesitará conocer el hipotético estado anterior a dicha rotación. Sabiendo que el vector i'' es igual a i' , es posible obtener el sistema de coordenadas en el estado intermedio.

El sistema de coordenadas final es el siguiente:

$$\begin{cases} i'' = q_s i q_s^* = i''_x i + i''_y j + i''_z k \\ j'' = q_s j q_s^* = j''_x i + j''_y j + j''_z k \\ k'' = q_s k q_s^* = k''_x i + k''_y j + k''_z k \end{cases}$$

Para el cálculo del conjunto $\{i', j', k'\}$ se parte de que $i' = i''$. Además se sabe que el vector j' será perpendicular a k y a la proyección de i'' sobre el plano xy de la esfera. Esta proyección será la resultante de eliminar la componente en k de i'' :

$$i''_{xy} = i''_x i + i''_y j$$

De la ecuación (5.43) se puede deducir que el módulo de la proyección de este vector será:

$$\|i''_{xy}\| = \sqrt{(\cos \psi \cos \theta)^2 + (\sin \psi \cos \theta)^2} = \cos \theta$$

Para el cálculo de j' será necesario utilizar el vector i''_{xy} normalizado ya que si no se obtendría un vector cuyo módulo no sería unitario:

$$(i''_{xy})_u = \frac{i''_{xy}}{\cos \theta} = \frac{1}{\cos \theta} (i''_x i + i''_y j)$$

Ahora se puede calcular j' :

$$\begin{aligned} j' &= k \times (i''_{xy})_u = \frac{1}{\cos \theta} k \times (i''_x i + i''_y j) \\ j' &= \frac{1}{\cos \theta} (-i''_y i + i''_x j) \end{aligned} \tag{5.46}$$

Finalmente, una vez conocidos i' y j' , es posible obtener k' :

$$k' = i' \times j'$$

En resumen:

$$\begin{cases} i' = i''_x i + i''_y j + i''_z k \\ j' = \frac{1}{\cos \theta} (-i''_y i + i''_x j) \\ k' = i' \times j' \end{cases} \tag{5.47}$$

Finalmente, conociendo el vector j' se puede obtener el último de los ángulos buscados:

$$\tan \phi = \frac{\sin \phi}{\cos \phi} = \frac{j'' \cdot k'}{j'' \cdot j'} \iff \phi = \arctan \frac{j'' \cdot k'}{j'' \cdot j'} \quad (5.48)$$

El algoritmo de cálculo de los ángulos de Euler puede resumirse en los siguientes pasos, conocido el cuaternion de orientación del sensor q_s ²:

Algoritmo de cálculo de los ángulos de Euler a partir del cuaternion de orientación			
1º	Cálculo de i'' y j''	$i' = i'' = q_s i q_s^*$	$j'' = q_s j q_s^*$
2º	Obtención de los ángulos ψ y θ	$\psi = \arctan \left(\frac{i'_y}{i'_x} \right)$	$\theta = \arcsin i'_z$
3º	Cálculo del vector j' y k'	$j' = \frac{1}{\cos \theta} (-i''_y i + i''_x j)$	$k' = i' \times j'$
4º	Obtención del tercer ángulo, ϕ		$\phi = \arctan \frac{j'' \cdot k'}{j'' \cdot j'}$

Cuadro 5.2: Algoritmo de cálculo de los ángulos de Euler a partir del cuaternion de orientación.

5.4.1. Ampliación del intervalo de los ángulos obtenidos

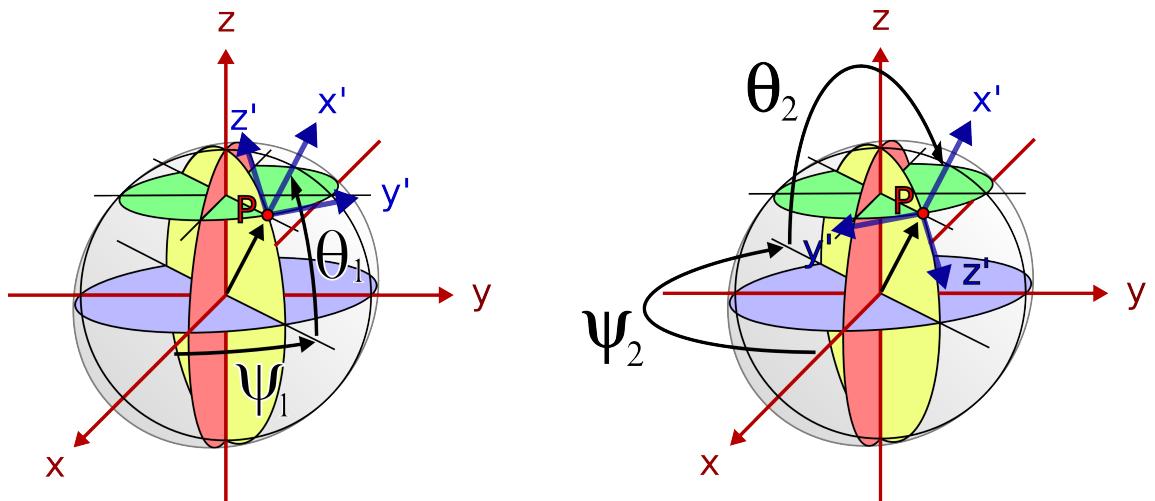
El algoritmo presentado en el cuadro 5.2 da una solución para el ángulo θ dentro del intervalo $(-\frac{\pi}{2}, \frac{\pi}{2})$. Puede pasar que la articulación del brazo robótico cuyo eje se refiera al ángulo θ se mueva en un intervalo más amplio. O que se quieran mover únicamente dos articulaciones mediante un sensor prescindiendo de la tercera rotación, y se necesite que $\phi \approx 0$ ya que de otro modo los otros dos ángulos no se corresponden muy bien con la realidad. por ello resultará de interés poder obtener soluciones de θ que no se restrinjan al intervalo $(-\frac{\pi}{2}, \frac{\pi}{2})$. En los apartados siguientes se dará un par de posibles soluciones a este problema.

Generación de una segunda solución a partir de la primera

Si se amplía el intervalo de θ a $(-\pi, \pi)$, se obtiene que cada posición en la esfera puede representarse de dos formas distintas mediante los ángulos ψ y θ .

En la figura 5.6) se muestran los ángulos de las dos posibles soluciones. Sea una solución $s_1 = \{\psi_1, \theta_1, \phi_1\}$ obtenida mediante el algoritmo descrito en el cuadro 5.2; Se deduce que la segunda solución es:

²A la hora de implementar el algoritmo, se utilizará la función atan2 en lugar de arctan

Figura 5.6: Dos soluciones para θ (pitch) y ψ (yaw)

$$s_2 = \begin{cases} \psi_2 = \begin{cases} \psi_1 + \pi : & \psi_1 < 0 \\ \psi_1 - \pi : & \psi_1 \geq 0 \end{cases} \\ \theta_2 = \begin{cases} -\pi - \theta_1 : & \theta_1 < 0 \\ \pi - \theta_1 : & \theta_1 \geq 0 \end{cases} \\ \phi_2 = \begin{cases} \phi_1 + \pi : & \phi_1 < 0 \\ \phi_1 - \pi : & \phi_1 \geq 0 \end{cases} \end{cases}$$

Para decidir que solución será la buena se deberá establecer un criterio de selección. Por ejemplo, en el caso de querer mover dos articulaciones, se sabe que ϕ tiene que ser aproximadamente 0. En el momento en que la primera solución nos de un valor de ϕ muy diferente de 0, se escogerá la segunda solución.

En la práctica, este método funciona bastante bien siempre y cuando $\theta \not\approx \frac{\pi}{2}$. En esta zona de la esfera se tiene que $\cos \theta \approx 0$, y a la hora de calcular j' (5.2), el término $\frac{1}{\cos \theta}$ amplifica enormemente el ruido propio del sensor, lo que genera saltos de magnitud apreciable en los valores de los ángulos obtenidos, aún cuando el sensor permanece estático.

Permutación de los ejes locales

Sea una articulación cuyo eje de rotación coincide con el eje y de los dos sensores dispuestos para medir su estado. Es esta situación se deduce de antemano que el vector base i local siempre se localizará en torno a un círculo máximo de la esfera, por lo que habrá zonas en las que nunca se situará.

FALTA

Capítulo 6

IMPLEMENTACIÓN DEL SOFTWARE

En este capítulo se describirá detalladamente el funcionamiento de los programas realizados para el presente proyecto. Todo el software se ha incluido dentro de un *stack* de ROS –llamado *youbot-xsens-controller*–, de forma que su instalación¹ en otros ordenadores con ROS sea sencilla y rápida.

El stack *youbot-xsens-controller* está compuesto por los siguientes paquetes, que pueden instalados de forma independiente si así se deseara:

- **xsens_driver**: driver para la comunicación sensor/máster - PC
- **dfv**: librería para el manejo de cuaterniones, vectores y matrices.
- **youbot_controller**: programa que realiza el control del robot *Youbot* utilizando los datos de los sensores *xsens*.

Para la creación de los distintos programas se han utilizado las herramientas que proporciona ROS para la compilación de programas y gestión de paquetes. En la figura 6.1 se presenta un esquema simplificado de las comunicaciones entre los distintos programas que forman el sistema.

6.1. Driver para la comunicación xbus Master/xsens - PC

Uno de los problemas más importantes a resolver en este proyecto es conseguir una comunicación fluida de los datos de los sensores hacia el PC. La comunicación entre un dispositivo y el ordenador se realiza mediante un programa llamado *driver*, que actúa de interfaz entre el dispositivo físico y el sistema operativo del PC, utilizando algún tipo de canal de datos (USB, bluetooth, etc.). En el caso del driver para los sensores *xsens*, la comunicación se realizará a través del puerto USB.

¹Se describirá el proceso de instalación en el anexo *INSTALACIÓN Y PUESTA EN MARCHA DEL SOFTWARE*

En la documentación del *Kit de Desarrollo de Software* de *xsens* vienen proporcionadas algunas clases y estructuras en lenguaje C++ para la comunicación de bajo nivel con el sensor/máster. Para la realización del driver se han aprovechado estas clases y se han encapsulado dentro de una clase *wrapper*, llamada *xsens::Driver*, que proporciona una interfaz sencilla hacia las clases de bajo nivel. El programa principal del *driver* realiza la configuración de los sensores, y lee los datos a través del *wrapper*, publicándolos en *topics* de ROS.

Los archivos de la documentación que se han incluído en el driver son los siguientes (incluyen los archivos de extensión .cpp y .h):

- cmt1: implementa las comunicaciones de bajo nivel con el puerto serie.
- cmt2: interfaz para mensajes y ficheros de registro.
- cmt3: capa de más alto nivel para las comunicaciones con el sensor. Proporciona una interfaz independiente del sistema operativo.
- cmtdef: definiciones de constantes y tipos de datos.
- cmtmessage: gestión de mensajes.
- cmtpacket: gestión de los paquetes de información en los que se basa la comunicación entre el sensor y el PC.
- cmtscan: funciones para el escaneo de los puertos del PC.
- pstdint: una versión portable de la librería stdint, que incluye definiciones precisas de tipos de datos enteros.
- xsens_exception: definición de la clase Exception usada por el driver.
- xsens_fifoqueue: contiene la implementación de una cola FIFO.
- xsens_file: definición de algunos tipos de datos.
- xsens_janitors: implementación de distintos tipos de *janitors*, que realizan funciones de limpieza de objetos al perder su *scope*.
- xsens_list: implementación de una clase lista utilizada en el driver.
- xsens_std: definición de los valores de retorno de las funciones del driver.
- xsens_time: implementación de funciones y constantes para la gestión del tiempo.

6.1.1. Funcionamiento de la clase *xsens::Driver*

La clase *xsens::Driver* actúa de interfaz hacia las clases de más bajo nivel encargadas de la comunicación del puerto serie. El objetivo perseguido con su creación es encapsular y automatizar la configuración de los sensores y la lectura de datos, de forma que no sea necesario realizar todo este proceso manualmente. Resumiendo brevemente, las funciones que realiza esta clase son:

1. Realizar un escaneo de los puertos USB del PC con el objetivo de detectar tipo y número de sensores conectados.

2. Configurar los sensores en el modo deseado (datos que devolverá, forma de representar las orientaciones, etc.).
3. Establecer los sensores en modo de medición.
4. Actualizar los datos miembro cada vez que llega una nueva medición de los sensores.

Esta clase contiene una serie de métodos miembro que permiten interactuar con ella. En la siguiente lista se muestran los de mayor utilidad:

- Métodos para establecer la configuración del sensor:

- **void SetOutputMode(CmtOutputMode output_mode)**

Permite establecer el modo de salida del sensor, el cual define el tipo de datos que devolverá. Acepta un parámetro de tipo *CmtOutputMode*, que puede ser combinación de varios de los siguientes valores:

- CMT_OUTPUTMODE_RAW: datos crudos de los acelerómetros, giroscopos y magnetómetros sin calibrar.
- CMT_OUTPUTMODE_TEMP: temperatura registrada por el sensor.
- CMT_OUTPUTMODE_CALIB: datos calibrados de los acelerómetros, giroscopos y magnetómetros.
- CMT_OUTPUTMODE_ORIENT: orientación del sensor.

Por ejemplo, para configurar el sensor de modo que devuelva los datos calibrados, la orientación y la temperatura se llamará a la función de la siguiente forma:

```
1 driver.SetOutputMode(CMT_OUTPUTMODE_CALIB | CMT_OUTPUTMODE_ORIENT |  
CMT_OUTPUTMODE_TEMP);
```

- **void SetOutputSettings(CmtOutputSettings output_settings)**

Permite establecer la forma de representación de la orientación del sensor. Acepta un parámetro de tipo *CmtOutputSettings*, que puede ser uno de los siguientes valores:

- CMT_OUTPUTSETTINGS_ORIENTMODE_EULER: Orientación del sensor en forma de ángulos de Euler
- CMT_OUTPUTSETTINGS_ORIENTMODE_MATRIX: Matriz de rotación.
- CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION: Cuaternión de orientación.

Para configurar el sensor de modo que devuelva su orientación forma de cuaternión, se pasará a la función el siguiente parámetro:

```
1 driver.SetOutputSettings(CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION);
```

- **void SetAlignmentMatrix(unsigned int sensor_index, CmtMatrix alignment_matrix)**

Establece una matriz de rotación inicial al sensor cuyo identificador se le pase al parámetro *sensor_index* *sensor_index*.

Para establecer una matriz de rotación inicial identidad al sensor de id 0:

```

1   driver.SetAlignmentMatrix(0,
    xsens::DfvToCmtMatrix(dfv::Matrix::Identity(3)));

```

- **bool Initialize()**

Se llamará a esta función después de haber establecido los parámetros del sensor mediante las funciones descritas anteriormente. Esta función pone al sensor en modo configuración, lo prepara para que devuelva los datos deseados y lo pone en modo de medida.

- Métodos para leer datos del sensor:

- **unsigned int GetMtCount()**

Devuelve el número de dispositivos xsens conectados al PC, excluyendo el máster xbus.

- **CmtOutputMode GetOutputMode() const**

Devuelve el modo de salida de los sensores. Los valores de retorno son los mismos que los que puede aceptar el parámetro de la función *SetOutputMode()* descrita anteriormente.

- **CmtOutputSettings GetOutputSettings() const**

Devuelve la forma de representación de la orientación de los sensores. Los valores de retorno son los mismos que los que puede aceptar el parámetro de la función *SetOutputSettings()* descrita anteriormente.

- **bool SpinOnce()**

Esta función realiza un ciclo en la toma de datos del sensor. Se queda esperando a que llegue un mensaje procedente del sensor, y a continuación actualiza todas las estructuras de datos con los nuevos valores obtenidos. Se ha de llamar a esta función antes de ejecutar alguna de las funciones descritas a continuación. Devuelve *true* si todos los datos fueron leídos de forma correcta, y *false* en caso contrario.

- **CmtQuat& GetOriQuat(int mt_index = 0)**

Devuelve el cuaternion de orientación del sensor con número de identificación especificado. Si se deja el parámetro en blanco, devuelve el cuaternion de orientación del sensor de id 0.

- **CmtMatrix& GetOriMatrix(int mt_index = 0)**

Devuelve la matriz de orientación del sensor con número de identificación especificado. Si se deja el parámetro en blanco, devuelve la matriz de orientación del sensor de id 0.

- **CmtEuler& GetOriEuler(int mt_index = 0)**

Devuelve los ángulos de Euler del sensor con número de identificación especificado. Si se deja el parámetro en blanco, devuelve los ángulos de Euler del sensor de id 0.

- **CmtRawData& GetRawData(int mt_index = 0)**

Devuelve los datos crudos de los acelerómetros, giróscopos y magnetómetros del sensor con número de identificación especificado. Si se deja el parámetro en blanco, devuelve los datos crudos del sensor de id 0.

- **CmtCalData& GetCalData(int mt_index = 0)**

Devuelve los datos calibrados de los acelerómetros, giróscopos y magnetómetros del sensor con número de identificación especificado. Si se deja el parámetro en blanco, devuelve los datos calibrados del sensor de id 0.

A continuación se presenta un ejemplo mínimo de cómo utilizar la clase xsens::Driver para tomar datos de un sensor:

```

1 #include <xSENS_driver/xsens_driver.h>
2 #include <dfv/dfv.h>
3 #include <xSENS_driver/utils.h>
4
5 int main(int argc, char** argv)
6 {
7     xsens::Driver driver;
8
9     // Configuración del sensor
10    driver.SetOutputMode(CMT_OUTPUTMODE_CALIB | CMT_OUTPUTMODE_ORIENT |
11                           CMT_OUTPUTMODE_TEMP);
12    driver.SetOutputSettings(CMT_OUTPUTSETTINGS_ORIENTMODE_QUATERNION);
13    driver.SetAlignmentMatrix(0,
14                             xsens::DfvToCmtMatrix(dfv::Matrix::Identity(3)));
15    driver.Initialize();
16
17    // Toma de datos
18    while(driver.SpinOnce())
19    {
20        // Código para lectura y procesamiento de datos
21        // ...
22    }
23
24    return 0;
}

```

6.1.2. Programa principal: xsens_node

El nodo *xsens_node* es el encargado de publicar en topics de ROS los datos leídos de los sensores. Para la lectura de los datos del sensor hace uso de la clase xsens::Driver descrita en el apartado anterior. Podría considerarse que este programa actúa de interfaz entre dicha clase y ROS. Este programa realiza las siguientes tareas:

1. Detección del número de sensores conectados al PC.
2. Configuración de los sensores:
 - Modo de salida: datos calibrados y orientación.
 - Forma de representación de la orientación: cuaternión.
 - Matriz de rotación inicial identidad.
3. Inicialización y creación de los topics y parámetros de ROS. En concreto se crean los topics y parámetros descritos en los cuadros 6.1 y 6.2 respectivamente.

4. Lectura de los datos del sensor y publicación en los topics de ROS correspondientes.

Topics		
Nombre	Tipo	Descripción
/xsens_node/sensorX/acc	geometry_msgs::Vector3Stamped	Acelerómetros calibrados
/xsens_node/sensorX/gyr	geometry_msgs::Vector3Stamped	Giróscopos calibrados
/xsens_node/sensorX/mag	geometry_msgs::Vector3Stamped	Magnetómetros calibrados
/xsens_node/sensorX/raw_acc	geometry_msgs::Vector3Stamped	Acelerómetros crudos
/xsens_node/sensorX/raw_gyr	geometry_msgs::Vector3Stamped	Giróscopos crudos
/xsens_node/sensorX/raw_mag	geometry_msgs::Vector3Stamped	Magnetómetros crudos
/xsens_node/sensorX/ori_quat	geometry_msgs::QuaternionStamped	Cuaternion de orientación
/xsens_node/sensorX/ori_matrix	std_msgs::Float64MultiArray	Matriz de orientación
/xsens_node/sensorX/ori_euler	std_msgs::Float64MultiArray	Ángulos de Euler

Cuadro 6.1: Topics publicados por el programa *xsens_node*. La X en *sensorX* representa el número de identificación del sensor. Se crea un topic de cada tipo por cada sensor conectado al máster. Los topics creados también dependen de la configuración de los sensores. Por ejemplo, si los sensores son configurados para que den sólo los datos calibrados, no se crearán los topics de orientación.

Parámetros		
Nombre	Tipo	Descripción
/xsens_node/sensor_count	entero	Número de sensores
/xsens_node/output_mode	entero	Modo de salida
/xsens_node/output_settings	entero	Forma de representación de la orientación

Cuadro 6.2: Topics publicados por el programa *xsens_node*.

6.1.3. Cómo tomar datos del sensor desde otro programa

Junto con el paquete *xsens_driver* se incluye además una librería con las clases *xsens::SensorSubscriber* y *xsens::SensorSubscriberList*. Estas clases proporcionan una forma muy sencilla de acceder a los topics publicados por el programa *xsens_node*. Al declarar un objeto del tipo *xsens::SensorSubscriberList*, automáticamente dicho objeto se autoconfigura leyendo los parámetros de ROS con la información de configuración de los sensores (cuadro 6.2) e inmediatamente se pone a leer los datos de los topics correspondientes (cuadro 6.1 a excepción de los datos en crudo), sin tener que hacer nada más que pasarle como parámetro el *handle* del nodo en que sea declarado.

Un objeto de clase *xsens::SensorSubscriberList* proporciona los siguientes métodos para lectura de datos del sensor:

- **unsigned int GetMtCount() const**
Devuelve el número de sensores xsens detectados.
- **const dfv::Vector3 GetAcc(unsigned int mt_index) const**
Devuelve un vector con los datos de los acelerómetros del sensor de número de identificación que se le pase como parámetro.

- **const dfv::Vector3 GetGyr(unsigned int mt_index) const**

Devuelve un vector con los datos de los giróscopos del sensor de número de identificación que se le pase como parámetro.

- **const dfv::Vector3 GetMag(unsigned int mt_index) const**

Devuelve un vector con los datos de los magnetómetros del sensor de número de identificación que se le pase como parámetro.

Para incluir la librería *xsens_driver* en otro proyecto se tendrá que incluir la siguiente línea al final del archivo *CMakeLists.txt* del proyecto de ROS:

```
1 target_link_libraries(nombre_del_ejecutable dfv xsens_driver)
```

En el siguiente cuadro se presenta un ejemplo de como usar la clase *xSENS::SensorSubscriberList* para tomar datos de una red de sensores *xSens*:

```
1 #include <xSENS_driver/xSens_sensor_subscriber.h>
2
3 int main(int argc, char** argv)
4 {
5     // Inicialización de ROS
6     ros::init(argc, argv, "xSens_sensr_subscriber_node");
7     ros::NodeHandle node_handle;
8
9     // Declaracion del objeto sensor_subscriber_list
10    // Se autoconfigura de acuerdo a los parámetros detectados
11    // e inmediatamente comienza la lectura de los datos
12    xSens::SensorSubscriberList sensor_subscriber_list(node_handle);
13
14    while(ros::ok())
15    {
16        // Código para lectura y procesamiento de datos de
17        // sensor_subscriber_list
18        // ...
19
20        // Bucle a 10 Hz
21        ros::spinOnce();
22        ros::Duration(0.1).sleep();
23    }
}
```

6.2. Librería para manejo de cuaterniones, vectores y matrices

6.3. Controlador del brazo robótico del robot Youbot real

En el paquete *youbot_controller* se implementa un controlador de las articulaciones del brazo de un robot Youbot. Este programa se ha de ejecutar a la vez que los drivers

xsens_node y *youbot_oodl*, para poder tomar los datos de los sensores xsens y transmitir las órdenes al robot. Este controlador realiza las tareas resumidas a continuación:

1. Detección de número y configuración de los sensores xsens a través de los topics y parámetros publicados por el driver *xsens_node*. Si no hay 3 sensores conectados se termina el programa imprimiendo en pantalla una advertencia de que son necesarios 3 sensores para que el programa funcione correctamente.
2. Si detecta que todo está correcto, inmediatamente el programa comienza a leer los cuaterniones de orientación de los tres sensores. Con los tres cuaterniones, el programa halla los cuaterniones relativos entre cada sensor, y con éstos obtiene los ángulos de Euler de la articulación correspondiente². En la figura 6.2 se muestran las articulaciones del brazo y su identificación.
3. A continuación se asignan determinados ángulos obtenidos a las articulaciones correspondientes del robot.
4. Finalmente se publica el mensaje en el correspondiente topic de ROS con las posiciones de las articulaciones del robot, para que sea leído por el driver *youbot_oodl*.

6.3.1. El driver *youbot_oodl*

El driver *youbot_oodl* es un programa que crea una capa sobre la API de YouBot con el objetivo de poder realizar el control del robot mediante el envío de mensajes a ciertos topics ROS. En el cuadro 6.3 se muestran el nombre del topic y tipo de mensaje que se pueden utilizar para controlar el robot.

Topics a los que está suscrito el driver <i>youbot_oodl</i>		
Nombre (espacio /arm_1)	Tipo	Descripción
/arm_controller/position_command	brics_actuator::JointPositions	Comando con la posición objetivo de las articulaciones
/arm_controller/velocity_command	brics_actuator::JointVelocities	Comando con la velocidad objetivo de las articulaciones
/gripper_controller/position_command	brics_actuator::JointPositions	Comando con la posición objetivo del gripper

Cuadro 6.3: Topics a los que está suscrito el driver *youbot_oodl* de los que lee la posición y velocidad deseadas para cada articulación.

6.3.2. La clase Youbot

Con un objetivo similar al de la creación de la clase *xsens::SensorSubscriberList*, la clase Youbot tiene como fin encapsular las comunicaciones de ROS entre el programa *youbot_controller* y el driver *youbot_oodl* para que no interfieran en el desarrollo del programa. Esta clase es la encargada de gestionar la publicación de mensajes en los topics

²En el capítulo CÁLCULO DE LAS POSICIONES Y ORIENTACIONES DE LOS SENSORES se muestra una explicación detallada de las matemáticas utilizadas para realizar esta tarea.

de ROS que utiliza el driver *youbot_oodl* (cuadro 6.3).

Esta clase tiene un funcionamiento muy sencillo. Sólamente es necesario pasarle el *handle* del nodo en el que se declare la clase. Para modificar los valores objetivo de las articulaciones y el *gripper* hay que modificar las variables miembro *joint_positions[]* y *gripper_positions[]*. Con el método *PublishMessage()*, se publica en los topics correspondientes los mensajes con las posiciones de las articulaciones. En el siguiente ejemplo se muestra un esquema básico de cómo trabajar con la clase:

```

1 #include <youbot_controller/youbot.h>
2
3 int main(int argc, char** argv)
4 {
5     // inicialización de ROS
6     ros::init(argc, argv, "youbot_example");
7     ros::NodeHandle node_handle;
8
9     // Declaración de un objeto Youbot
10    Youbot youbot(node_handle);
11
12    while(ros::ok())
13    {
14        // Asignación de los valores objetivo de cada articulación
15        youbot.joint_positions[0] = 3.141592;
16        youbot.joint_positions[1] = 2.718182;
17        youbot.joint_positions[2] = -1.618033;
18        youbot.joint_positions[3] = 1.414213;
19        youbot.joint_positions[4] = 0.577215;
20
21        // Publicación del mensaje
22        youbot.PublishMessage();
23
24        ros::spinOnce();
25        ros::Duration(0.1).sleep();
26    }
27
28    return 0;
29 }
```

6.4. Otras aplicaciones de visualización y control

6.4.1. Visualizador de la posición del brazo

6.4.2. Controlador de un simulador del brazo robótico del robot Youbot

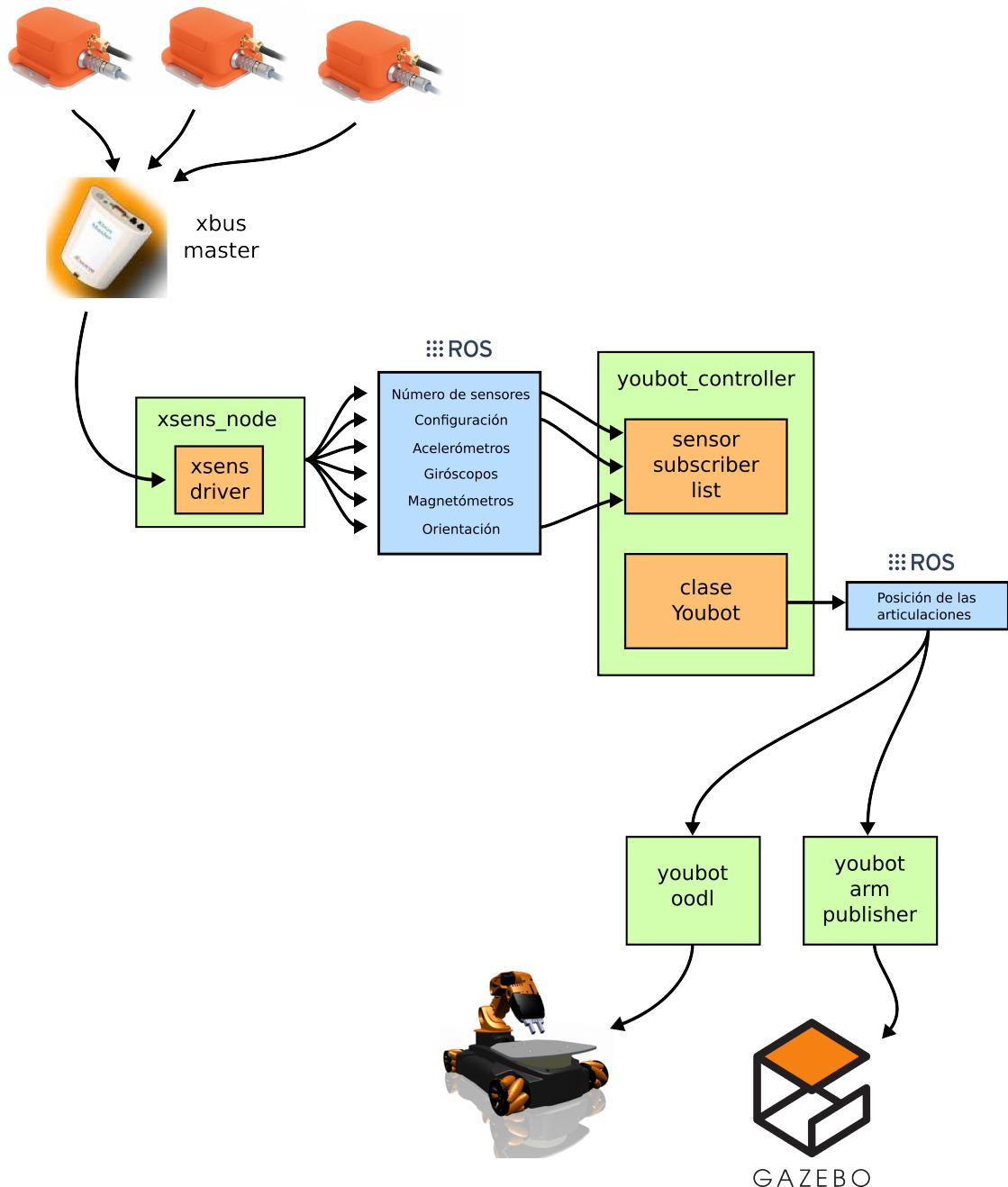


Figura 6.1: Esquema simplificado de las comunicaciones entre programas. En azul se representan los topics y parámetros de ROS, en verde los nodos y en naranja las clases principales.



Figura 6.2: Articulaciones del brazo del robot Youbot

Capítulo 7

IMPLEMENTACIÓN FÍSICA

7.1. Selección de componentes

7.2. Montaje

7.3. Ajuste

Capítulo 8

PROTOCOLO DE PRUEBAS. REDISEÑO

Capítulo 9

RESULTADOS OBTENIDOS

Capítulo 10

CONCLUSIONES

Capítulo 11

BIBLIOGRAFÍA

Parte II

Anexos

Apéndice A

INSTALACIÓN Y PUESTA EN MARCHA DEL SOFTWARE

En el presente capítulo se describirá como instalar las siguientes herramientas:

- ROS Fuerte
- Simulador Gazebo
- Simulador del brazo robótico del Youbot
- Stack de ROS con los programas creados para este proyecto

Se parte de la suposición de que se tiene disponible un ordenador con el sistema operativo Ubuntu 12.04 LTS instalado de forma nativa.

A.1. Instalación de ROS Fuerte

La versión de ROS que se utilizará es ROS Fuerte. Esta elección se debe a que dicha versión es compatible con el simulador Gazebo, con el que posteriormente se realizará la visualización en 3D del modelo. Además es la versión instalada en el PC embebido del robot Youbot.

A.1.1. Configuración de los repositorios de Ubuntu

Para comenzar con la instalación se procederá a abrir el Centro de Software de Ubuntu y en la barra de menú de dicho programa, se seleccionará en el menú Edit la opción Software Sources. En la pestaña Ubuntu Software se comprobará que están seleccionados los repositorios restricted, universe y multiverse, tal como aparece en la figura A.1.

A.1.2. Configuración del archivo sources.list

El archivo sources.list le dice al gestor de paquetes de Ubuntu de dónde puede obtener cada paquete de ROS. Se abrirá un terminal y se ejecutará el siguiente comando:

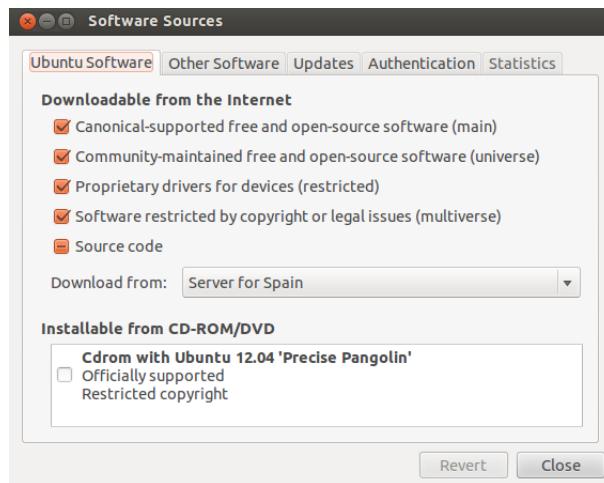


Figura A.1: Ventana Software Sources

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main"
> /etc/apt/sources.list.d/ros-latest.list'
```

Este comando crea un archivo de texto en la ruta especificada como parámetro, que contiene la dirección de donde descargar los paquetes para la versión específica de ROS que se tenga instalada, en este caso ROS Fuerte.

A.1.3. Configuración de la keys

En el terminal se ejecutará el siguiente comando:

```
$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

A.1.4. Descarga e instalación

Se actualizará el índice de paquetes de Ubuntu para tener la seguridad de que el servidor de ROS.org está indexado:

```
$ sudo apt-get update
```

A continuación se procederá a descargar e instalar la versión completa de ROS. En el terminal se ejecutará el siguiente comando:

```
$ sudo apt-get install ros-fuerte-desktop-full
```

Esta instalación traerá consigo las herramientas mostradas en la siguiente lista, entre muchas otras:

- ROS
- rx (herramientas para interfaz gráfica: rxbag, rxgraph, rxplot, ...)
- rviz (herramienta de visualización 3D)
- librerías genéricas para robots
- Simuladores 2D/3D (entre ellos Gazebo)
- Navegación y percepción 2D y 3D

A.1.5. Configuración del entorno

Cada vez que se inicie un nuevo terminal es necesario añadir las variables de entorno. Si se quisiera, se puede automatizar dicha tarea ejecutando el comando:

```
$ echo \source /opt/ros/fuerte/setup.bash" >> ~/.bashrc
```

Este comando añade la línea source /opt/ros/fuerte/setup.bash al archivo /.bashrc. Este archivo contiene la configuración inicial del terminal, y se ejecuta cada vez que abrimos un nuevo terminal. Posteriormente se ejecutará el archivo anterior para actualizar el terminal. De esta forma reconocerá los nuevos comandos de ROS:

```
$ . ~/.bashrc
```

A.2. Instalación del simulador Gazebo

Si se ha realizado el proceso de instalación de ROS detallado anteriormente, el simulador Gazebo ya vendrá de serie con dicha versión de ROS, y por lo tanto no haría falta seguir los pasos de este apartado. Si se hubiera instalado ROS de otra manera y éste no hubiera venido con el programa Gazebo, se tendría que instalar el programa de forma manual. Para instalar la versión de Gazebo preparada para comunicarse con ROS se ejecutará el siguiente comando:

```
$ sudo apt-get install ros-fuerte-simulator-gazebo
```

A.3. Instalación del simulador de brazo robótico del YouBot

A.4. Instalación del stack youbot-xsens-controller

A.4.1. Instalación desde el CD adjunto al proyecto

A.4.2. Instalación desde git

Apéndice B

SOLUCIÓN DE PROBLEMAS

B.1. Error iniciando Gazebo

```
Msg Waiting for master
Msg Connected to gazebo master @ http://localhost:11345
Exception [Master.cc:69] Unable to start server[Address already in use]
```

```
terminate called after throwing an instance of 'gazebo::common::Exception'
Aborted (core dumped)
[gazebo-1] process has died [pid 2795, exit code 134, cmd /opt/ros/fuerte/stacks/
/opt/ros/fuerte/stacks/simulator_gazebo/gazebo_worlds/worlds/empty.world __name:=_
__log:=/home/daniel/.ros/log/772c2f96-ab75-11e2-a2fc-001de05009b5/gazebo-1.log].
log file: /home/daniel/.ros/log/772c2f96-ab75-11e2-a2fc-001de05009b5/gazebo-1*.log
LightListWidget::OnLightMsg
```

Solución: Ejecutar comando:

```
$ ps ax | grep [g]z
```

Ver si hay un proceso gzserver

```
3118 ? S1 12:47
/opt/ros/fuerte/stacks/simulator_gazebo/gazebo/bin/gzserver
/opt/ros/fuerte/stacks/simulator_gazebo/gazebo_worlds/worlds/empty.world __name:=_
__log:=/home/daniel/.ros/log/8188cc76-ab73-11e2-a4ec-001de05009b5/gazebo-1.log
-s /opt/ros/fuerte/stacks/simulator_gazebo/gazebo/lib/libgazebo_ros_paths_plugin.so
-s /opt/ros/fuerte/stacks/simulator_gazebo/gazebo/lib/libgazebo_ros_api_plugin.so
```

Si lo hay, ejecutar *System Monitor* y matar el proceso *gzserver*.