

UNIVERSIDAD DE OVIEDO

CENTRO INTERNACIONAL DE POSTGRADO

MASTER EN INGENIERÍA MECATRÓNICA

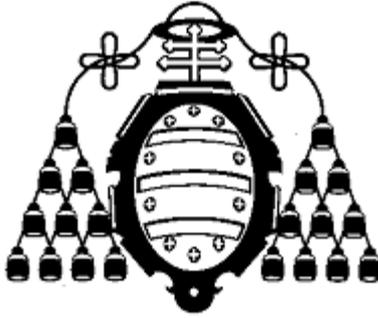
TRABAJO FIN DE MÁSTER

TALLER DE ROBÓTICA Y VISIÓN ARTIFICIAL PARA ESTUDIOS DE GRADO

JULIO 2013

ALUMNO: FERNANDO JOSÉ SOTO SÁNCHEZ

TUTOR: JUAN CARLOS ÁLVAREZ ÁLVAREZ



UNIVERSIDAD DE OVIEDO

CENTRO INTERNACIONAL DE POSTGRADO

MASTER EN INGENIERÍA MECATRÓNICA

TRABAJO FIN DE MÁSTER

TALLER DE ROBÓTICA Y VISIÓN ARTIFICIAL PARA ESTUDIOS DE GRADO

JULIO 2013

Fernando José Soto Sánchez

Juan Carlos Álvarez Álvarez

[Firma]

[Firma]

RESUMEN

El objetivo de este Trabajo Fin de Master es documentar, mediante procedimientos y ejercicios, una introducción a algunas materias propias de un Taller de Robótica para Estudios de Grado.

El planteamiento del trabajo consiste en utilizar una plataforma de desarrollo *software* de robots llamada ROS (*Robot Operating System*) como elemento integrador para experimentar con sensores, actuadores y procesamiento de información.

El trabajo se ha estructurado en 4 secciones:

1. **Sensores:** Aquí se introducen algunos dispositivos de uso frecuente en robótica: un sensor de profundidad, una cámaras USB, un joystick y un medidor de distancias láser. Se resumen sus características principales y los procedimientos necesarios para utilizarlos a través de ROS.
2. **Actuadores:** Se describen servomotores avanzados Dynamixel, la plataforma *pan-tilt* PTU-D46, y el manipulador móvil Kuka Youbot. También se indican los procedimientos de uso mediante ROS.
3. **Integración:** Se describen ejercicios que combinan un sensor y un actuador de los descritos en las secciones previas, demostrando la utilidad de la plataforma ROS como elemento de integración.

	Kuka Youbot	PTU	Dynamixel	Coche Arduino
Asus Xtion	X	X		
Joystick	X	X		X
Cámara USB			X	
Teclado	X			

4. **Apéndices:** Se incluyen aquí algunos tutoriales básicos sobre materias periféricas a la robótica aunque muy relacionadas, como el sistema operativo Linux, el lenguaje de programación C++, el entorno de desarrollo Eclipse CDT, la librería de visión artificial OpenCV, comunicación serie, procedimientos de uso frecuente en ROS y herramientas de control de versiones software (*subversion* y *git*). Esta sección también contiene algunos listados de código fuente referenciado en diferentes capítulos, con la intención de facilitar la legibilidad de las secciones anteriores.

PALABRAS CLAVE

Robótica - ROS – Visión Artificial - Sensores - Actuadores

ÍNDICE GENERAL

1.	INTRODUCCIÓN.....	1
1.1.	ROS	1
1.2.	FUNCIONALIDAD DE ROS.....	2
1.3.	ARQUITECTURA DE ROS.....	2
1.3.1.	<i>Sistema de Ficheros.....</i>	2
1.3.2.	<i>Estructura funcional.....</i>	3
1.3.3.	<i>Escenario de establecimiento de conexión a un topic.....</i>	3
1.3.4.	<i>Escenario de establecimiento de conexión a un service.....</i>	4
2.	SENSORES.....	5
2.1.	SENSOR ASUS XTION. INTRODUCCIÓN OPENNI.....	5
2.1.1.	<i>Instalación típica de OpenNI en Ubuntu</i>	6
2.1.2.	<i>Creación de un proyecto de ejemplo con Eclipse.....</i>	6
2.1.3.	<i>Programación con OpenNI.....</i>	8
2.1.4.	<i>Integración del sensor Asus Xtion en ROS</i>	8
2.1.5.	<i>Reconocimiento esquelético con openni_tracker.....</i>	8
2.2.	JOYSTICK USB GENÉRICO COMPATIBLE LINUX	10
2.2.1.	<i>Configuración de un Joystick en Linux.....</i>	10
2.2.2.	<i>Integración de un Joystick en ROS</i>	11
2.3.	CÁMARA USB GENÉRICA COMPATIBLE LINUX	12
2.3.1.	<i>Instalación de cámara en Linux.....</i>	12
2.3.2.	<i>Integración ROS de cámara USB</i>	13
2.3.3.	<i>Calibración</i>	14
2.3.4.	<i>Procedimiento de calibración con tablero de ajedrez</i>	16
2.3.5.	<i>Calibración con OpenCV a través de ROS.....</i>	16
2.5.	MEDIDOR LÁSER SICK LMS200	19
2.5.1.	<i>Configuración para uso con ROS</i>	20
3.	ACTUADORES.....	23
3.1.	ACTUADORES ROBOTIS DYNAMIXEL.....	23
3.1.1.	<i>Actuadores AX-12A y AX-12W</i>	23
3.1.2.	<i>Acceso a la configuración e información de estado</i>	24
3.1.3.	<i>Dynamixel SDK.....</i>	24
3.1.4.	<i>USB2Dynamixel.....</i>	24
3.1.5.	<i>Controlador de motores CM-900.....</i>	25
3.1.6.	<i>Integración ROS</i>	25
3.3.	UNIDAD PAN-TILT PTU-D46	29
3.3.1.	<i>Conexiones.....</i>	29
3.3.2.	<i>Operativa básica.....</i>	29
3.3.3.	<i>Integración ROS</i>	31
3.4.	KUKA YUBOT.....	32
3.4.1.	<i>Descripción General.....</i>	32
3.4.2.	<i>Plataforma Móvil.....</i>	32
3.4.3.	<i>Brazo.....</i>	33
3.4.4.	<i>Manejo del Robot.....</i>	33
3.4.5.	<i>Programación mediante Kuka Youbot API.....</i>	34
3.4.6.	<i>Descripción del YouBot para simulación</i>	34
4.	INTEGRACIÓN.....	35
4.1.	INTERACCIÓN ASUS XTION + PTU-D46 PARA SEGUIMIENTO DE PUNTO MÁS CERCANO	35
4.1.1.	<i>Introducción.....</i>	35
4.1.2.	<i>Determinación de los parámetros de orientación pan y tilt.....</i>	36
4.1.3.	<i>Efecto del desajuste entre sensor de imagen y eje tilt.....</i>	37

4.2.	TELEOPERACIÓN DE UNIDAD PTU-46 MEDIANTE JOYSTICK	39
4.2.1.	Introducción	39
4.2.2.	Programación del nodo ROS	39
4.3.	TELEOPERACIÓN DE PLATAFORMA MÓVIL YOUTBOT MEDIANTE TECLADO	41
4.3.1.	Nodo vel_publisher	41
4.3.2.	Nodo youbot_base_move.....	43
4.4.	IMITACIÓN DE MOVIMIENTO ESQUELETAL CON BRAZO YOUTBOT.....	45
4.4.1.	Configuración ROS	46
4.4.2.	Programa de lectura de posición esquelética y obtención de ángulos	48
4.4.3.	Programa de movimiento de brazo	51
4.5.	SEGUIMIENTO DE OBJETOS CON WEBCAM, OPENCV Y ACTUADORES AX-12A	53
4.5.1.	Montaje de actuadores y sensor.....	54
4.5.2.	Creación del paquete ROS	54
4.5.3.	Configuración de los actuadores Dynamixel	54
4.5.4.	Configuración de la cámara USB	56
4.5.5.	Detección de objetos por su color mediante OpenCV.....	56
4.5.6.	Orientación de la cámara para centrar el objetivo.....	57
4.6.	TELEOPERACIÓN DE PLATAFORMA MÓVIL ARDUINO MEDIANTE JOYSTICK	58
4.6.1.	Descripción de la plataforma móvil.....	58
4.6.2.	Control de velocidad en lazo abierto usando tracción diferencial	60
4.6.3.	La plataforma Arduino.....	61
4.6.4.	El Lenguaje Arduino	61
4.6.5.	El Gestor de Arranque	62
4.6.6.	La plataforma Arduino UNO	62
4.6.7.	Motor Shield de Adafruit.....	62
4.6.8.	Motor Shield oficial de la web de Arduino.....	64
4.6.9.	Comunicación entre nodo ROS y Arduino	65
4.6.10.	Programa ROS para control de plataforma móvil mediante joystick	67
4.6.11.	Programa de control de motores en Arduino	69
4.6.12.	Pruebas.....	72
5.	APÉNDICE I. HERRAMIENTAS.....	73
5.1.	INTRO LINUX	73
5.1.1.	Linux Shell.....	73
5.1.2.	Inicio de sesión.....	73
5.1.3.	Comandos básicos.....	74
5.1.4.	Combinaciones de teclas en Bash	75
5.1.5.	Sistema de Ficheros Linux	75
5.1.6.	Estructura de directorios	77
5.1.7.	Montaje del sistema de ficheros	78
5.1.8.	Permisos de acceso	78
5.1.9.	Gestor de Paquetes	79
5.2.	IDE ECLIPSE.....	81
5.2.1.	Introducción	81
5.2.2.	Instalación del IDE en Linux.....	81
5.2.3.	Creación de un proyecto	82
5.2.4.	Depuración.....	84
5.2.5.	Configuración específica ROS	84
5.3.	PROGRAMACIÓN ORIENTADA A OBJETOS EN C++	86
5.3.1.	Introducción	86
5.3.2.	Clases y Objetos	86
5.3.3.	El Puntero Implícito this	88
5.3.4.	Herencia.....	88
5.3.5.	Control de Acceso	89
5.3.6.	Polimorfismo	90
5.3.7.	Composición y Agregación	91
5.4.	CONTROL DE REVISIONES SOFTWARE.....	93
5.4.1.	Introducción	93

5.4.2.	<i>Conceptos básicos y terminología:</i>	93
5.4.3.	<i>Formas de cooperación</i>	94
5.4.4.	<i>Arquitecturas de almacenamiento (repositorios)</i>	94
5.4.5.	<i>Subversion</i>	94
5.4.6.	<i>Git</i>	96
5.5.	PROCEDIMIENTOS BÁSICOS ROS	98
5.5.1.	<i>Instalación de ROS</i>	98
5.5.2.	<i>Creación y uso de espacios de trabajo ROS</i>	99
5.5.3.	<i>Navegación por el sistema de ficheros ROS</i>	102
5.5.4.	<i>Creación de paquetes ROS</i>	103
5.6.	EL FORMATO URDF DE DESCRIPCIÓN DE ROBOTS	108
5.6.1.	<i>Elemento <robot></i>	108
5.6.2.	<i>Elemento <link></i>	108
5.6.3.	<i>Elemento <joint></i>	110
5.6.4.	<i>Elemento <sensor></i>	111
5.6.5.	<i>Modelo URDF simplificado del Kuka YouBot</i>	112
5.7.	VISIÓN ARTIFICIAL MEDIANTE OPENCV	113
5.7.1.	<i>Introducción</i>	113
5.7.2.	<i>Instalación</i>	113
5.7.3.	<i>Creación de un programa con OpenCV</i>	114
5.7.4.	<i>Preparación del entorno de compilación con cmake</i>	114
5.7.5.	<i>Programación OpenCV desde IDE Eclipse CDT</i>	115
5.7.6.	<i>Creación de un paquete ROS para programación OpenCV</i>	117
5.8.	COMUNICACIÓN SERIE EN LINUX	120
5.8.1.	<i>Introducción</i>	120
5.8.2.	<i>El estándar RS-232</i>	120
5.8.3.	<i>UART</i>	121
5.8.4.	<i>Termios</i>	122
5.8.5.	<i>Comunicación serie C++</i>	123
5.9.	CONFIGURACIÓN GPIO Y UART EN BEAGLEBONE	124
5.9.1.	<i>Habilitación de UART 3 y 5</i>	125
5.9.2.	<i>Prueba de comunicaciones</i>	129
6.	APÉNDICE II. CÓDIGO FUENTE	131
6.1.	LIBRERÍA COMUNICACIÓN SERIE	131
6.2.	DETERMINACIÓN DE COORDENADAS CON ASUS XTION Y OPENNI.....	137
6.3.	CONTROL UNIDAD PAN TILT PTU-D46 CON ROS	140
6.4.	SEGUIMIENTO DE PUNTO MÁS CERCANO CON PTU-46 Y OPENNI.....	144
6.5.	MODELO URDF SIMPLIFICADO YOUTBOT.....	150
6.6.	PROGRAMA SEGUIMIENTO CON CÁMARA USB, OPENCV Y ACTUADORES DYNAMIXEL.....	153
6.6.1.	<i>Interfaz de Usuario</i>	153
6.6.2.	<i>Identificación de objetos</i>	153
6.6.3.	<i>Determinación de los ángulos de corrección</i>	153
6.6.4.	<i>Comunicación motores</i>	153
	BIBLIOGRAFÍA	161

1. INTRODUCCIÓN

La robótica es un área del conocimiento que enlaza numerosas disciplinas, tales como cinemática, dinámica, mecanismos, actuadores, sensorización, control, planificación de movimiento o programación de tareas. Todas estas disciplinas tienen en común en su aplicación a la robótica el uso de software en mayor o menor medida, ya sea para cálculo y diseño, simulación, depuración de errores, configuración o simplemente para su funcionamiento.

Actualmente existen diversas plataformas de software relacionado con la robótica (ROS, Orocos, Microsoft Robotics Studio, Orca, etc.) surgidas con el objetivo de integrar el uso de diferentes tecnologías y facilitar la colaboración.

En este trabajo se pretende introducir una de ellas, denominada ROS, mediante la realización de ejercicios habituales en un taller de robótica, como pueden ser la captura de datos del entorno mediante de sensores, el procesamiento de los datos adquiridos, la toma de decisiones a partir de los datos percibidos y procesados o la ejecución de acciones de control sobre actuadores.

1.1. ROS

ROS (*Robot Operating System*) proporciona un entorno de programación distribuido de código abierto (*open source*) para el control de robots, tanto físicamente como en simulación, que aporta abstracción hardware, control de bajo nivel, comunicación entre procesos y gestión de paquetes software.

El matiz *open source* es fundamental. Existen actualmente infinidad de repositorios de libre acceso con soluciones ROS de todo tipo (SLAM, reconocimiento 3D, planificación de movimiento o *machine learning*). Esta disponibilidad de soluciones a problemas cotidianos en el campo de la robótica ahorra tiempo y esfuerzo y permite al diseñador centrarse en su área de interés sin ocuparse de solucionar problemas ya resueltos por otros.

El objetivo principal de ROS es compartir y colaborar. Además persigue la facilidad de integración con otras plataformas, independencia respecto al lenguaje de programación, facilidad de realización de pruebas y escalabilidad.

En la actualidad ROS funciona sobre plataformas tipo Unix, está mayoritariamente probado en sistemas Linux y Mac OS X y oficialmente soportado para determinadas versiones de Ubuntu. Regularmente se publican distribuciones ROS, consistentes en el núcleo del sistema ROS y un conjunto de utilidades compatibles.

1.2. Funcionalidad de ROS.

Entre las muchas funcionalidades que aporta ROS al desarrollo de software en robótica se pueden destacar las siguientes:

- **Modularidad:** la estructura modular de ROS permite dividir el trabajo. Es posible trabajar de modo independiente en un nodo de procesamiento de imagen leyendo y entregando información en interfaces bien definidos sin entrar en detalles de captura o representación.
- **Visualización y simulación:** ROS proporciona herramientas como *rviz* y *Gazebo* capaces de representar e interactuar con modelos 2D o 3D. Para ello define un lenguaje de descripción de robots (URDF).
- **Monitorización y Depuración:** mediante utilidades como *rxgraph* es posible visualizar gráficamente las relaciones (publicación o suscripción) entre nodos. Con *rostopic* se puede consultar qué información hay disponible y su frecuencia de publicación. Con *rosviz* es posible grabar sesiones (por ejemplo información de sensores) y reproducirlas más tarde.
- **Programación.** El proceso de creación e instalación de paquetes ROS simplifica en gran medida tareas de configuración (compilación, localización de librerías, etc.) y resolución de dependencias.
- **Ejecución:** la herramienta *roslaunch* permite ejecutar nodos estableciendo previamente los parámetros necesarios de configuración (por ejemplo, el nombre de un dispositivo de entrada/salida) o arrancando otros nodos relacionados.

1.3. Arquitectura de ROS.

ROS es fundamentalmente un sistema cliente/servidor. Consta de una serie de nodos (programas) que se comunican entre sí a través de *topics* (difusión) o *services* (comunicación interactiva).

Existe un nodo principal (*master*) al que los nodos acuden para localizarse mutuamente y obtener información de configuración.

1.3.1. SISTEMA DE FICHEROS.

En el nivel de sistema de ficheros ROS define los siguientes elementos:

- **Packages:** Son la unidad principal de organización de software. Pueden contener procesos ROS, librerías, datos, archivos de configuración o cualquier agrupación de información coherente.

- **Manifests:** Archivos XML que contienen información sobre *packages* (información de licencia, dependencias, etc).
- **Stacks:** Colecciones de *packages* relacionados que aportan funcionalidad agregada.
- **Stack Manifest:** Archivos XML que contienen información sobre *stacks*.
- **Message types:** Estructuras de mensajes ROS
- **Service types:** Descripciones de servicios (definición de estructuras de peticiones y respuestas).

1.3.2. ESTRUCTURA FUNCIONAL.

Este nivel se corresponde con la red *peer-to-peer* de procesos ROS que se ejecutan simultáneamente

- **Nodes:** Procesos. Normalmente diseñados de manera modular, de modo que un sistema está formado por diversos *nodes*, cada uno con una funcionalidad concreta. La programación de *nodes* ROS se realiza usando librerías cliente ROS, como *roscpp* (C++), *rospy* (Python) o *roslisp* (LISP).
- **Master:** Proceso con funcionalidades de registro y consulta a los *nodes*. Permite que estos se encuentren y se comuniquen.
- **Parameter Server:** Almacén centralizado de datos. Es parte del *master*.
- **Messages:** Datos con una estructura determinada intercambiados entre *nodes*.
- **Topics:** Mecanismos de difusión de mensajes. Uno o varios *nodes* publican *messages* en determinado *topic* al que otros *nodes* pueden suscribirse. Permite desacoplar producción y consumo de información. Los *nodes* consumidores de un *topic* no necesitan saber de los que publican.
- **Services:** Mecanismos de comunicación de mensajes de tipo petición / respuesta.
- **Bags:** Formatos para el almacenamiento y reproducción de mensajes.

El *Master* actúa como un servicio de directorio. Almacena información de registro de *topics* y *services*. Los *nodes* se comunican con el *Master* para registrar información propia o solicitar información de registro de otros *nodes*.

Los *nodes* utilizan el *Master* para encontrarse mutuamente, pero una vez localizados se comunican entre sí directamente.

1.3.3. ESCENARIO DE ESTABLECIMIENTO DE CONEXIÓN A UN TOPIC.

1. Al iniciarse el *subscriber* lee de los argumentos de la línea de comandos el nombre del *topic* al que se debe suscribir.
2. Al iniciarse el *publisher* lee de los argumentos de la línea de comandos el nombre del *topic* en el que debe publicar.
3. El *subscriber* se registra en el *master*.
4. El *publisher* se registra en el *master*.
5. El *master* informa al *subscriber* sobre el nuevo *publisher*.

6. El *subscriber* contacta al *publisher* para solicitar una conexión al *topic* y negociar el protocolo de transporte.
7. El *publisher* envía al *subscriber* los datos necesarios para comunicarse mediante el protocolo elegido.
8. El *subscriber* se conecta al *publisher* según la configuración proporcionada.

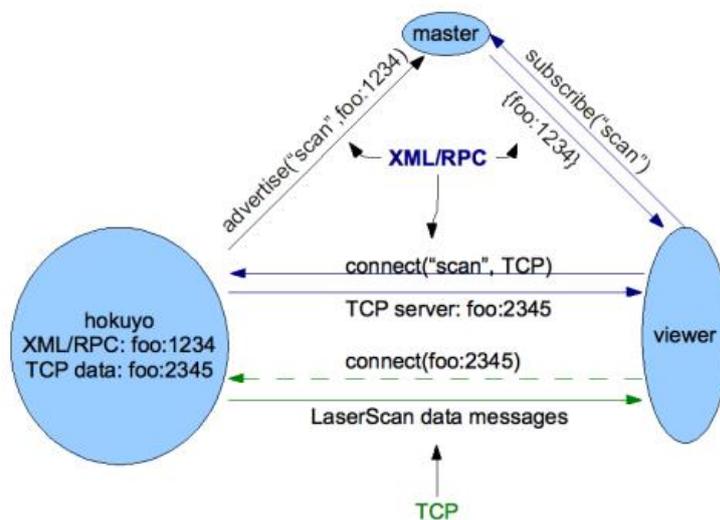


Figura 1.1- Esquema de conexión a un topic ROS

1.3.4. ESCENARIO DE ESTABLECIMIENTO DE CONEXIÓN A UN SERVICE.

1. Registro del *service* en el *master*.
2. El cliente consulta el *service* en el *master*.
3. El cliente establece una conexión TCP con el *service*.
4. El cliente y el *service* intercambian una cabecera de conexión.
5. El cliente envía al *service* un mensaje de petición serializado.
6. El *service* envía al cliente un mensaje de respuesta serializado.

En los capítulos siguientes se detalla con ejemplos el uso de ROS en aplicaciones robóticas. Finalmente, en los apéndices, se muestran algunas indicaciones sobre procedimientos y funcionalidades de uso frecuente en ROS.

2. SENSORES

2.1. Sensor Asus Xtion. Introducción OpenNI

En este capítulo utilizaremos la plataforma OpenNI para obtener información de un sensor de profundidad **Asus Xtion Pro Live**. El procedimiento es igualmente válido para otros modelos (MS Kinect, Primesense Carmine, etc).



Figura 2.1 - Sensor imagen 3D Asus Xtion

A continuación se listan las características de este sensor:

1. Distancia de uso: entre 0.8 y 3.5 metros.
2. Campo de visión: 58° (horizontal), 45° (vertical), 70° (diagonal).
3. Sensor: RGB + Profundidad + Micrófono.
4. Resolución Profundidad: VGA (640x480) a 30fps o QVGA (320x240) a 60fps.
5. Resolución RGB: SXGA (1280x1024).
6. Interfaz USB 2.0.
7. Programación OpenNI SDK (C++/C#/Java, entorno Windows y Linux).
8. Dimensiones: 18 x 3.5 x 5 cm.

Al igual que otros dispositivos similares que incorporan la tecnología **Primesense**, la percepción de profundidad se consigue mediante la emisión de una serie de haces infrarrojos y la detección de su reflejo mediante un sensor de imagen CMOS. Los patrones detectados se procesan en un sistema embebido (*SoC o System on a Chip*) generando una imagen de profundidad.

Para utilizar la información de este sensor en aplicaciones propias, se utilizan las tecnologías OpenNI y NITE.

La plataforma OpenNI es un SDK (*software development kit*) de código abierto para el desarrollo de aplicaciones y librerías *middleware* (*middleware* es un *software* que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, *software*, redes, *hardware* y/o sistemas operativos). Básicamente se encarga de proporcionar acceso a los datos obtenidos por el sensor.

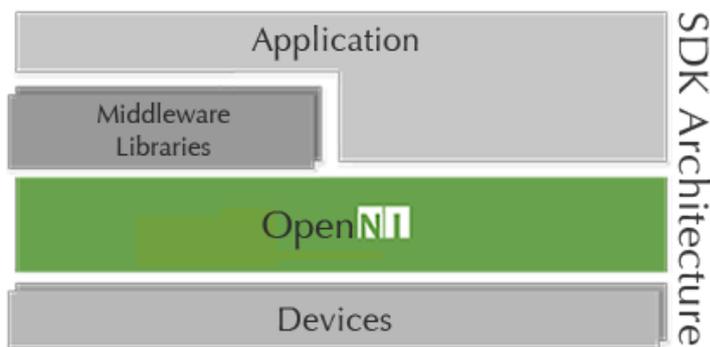


Figura 2.2 - Arquitectura OpenNI

NITE (Natural interface Technology for End-User) constituye la capa de algoritmos de percepción, principalmente reconocimiento de manos o del cuerpo humano y su utilización en identificación y seguimiento, análisis de escena, o seguimiento de uniones del esqueleto, etc.

2.1.1. INSTALACIÓN TÍPICA DE OPENNI EN UBUNTU

Para instalar OpenNI en Ubuntu descargamos los archivos binarios para Linux correspondientes a nuestra arquitectura (x86, x64 o ARM) del sitio web **openni.org** y descomprimos paquete zip en una carpeta. A continuación, desde un terminal ejecutamos los siguientes comandos:

```
tar -xvf OpenNI-Linux-x86-2.1.0.tar.bz2
cd OpenNI-2.1.0
sudo ./install.sh
```

El instalador creará un archivo llamado *OpenNIDevEnvironment* en la misma carpeta para establecer las variables de entorno *OPENNI2_INCLUDE* y *OPENNI2_REDIST* (para invocar cuando sea necesario mediante el comando *source*).

```
source OpenNIDevEnvironment
export | grep OPENNI
```

Puede resultar conveniente añadir el contenido del archivo *OpenNIDevEnvironment* a nuestro *~/.bashrc* para asegurarnos que las variables de entorno están disponibles en cualquier terminal.

Si se desea instalar a partir del código fuente, éste está disponible en la siguiente dirección: **<https://github.com/OpenNI/OpenNI2>**.

2.1.2. CREACIÓN DE UN PROYECTO DE EJEMPLO CON ECLIPSE

Para trabajar cómodamente con OpenNI utilizando el IDE Eclipse CDT, hay que realizar algunas configuraciones previas:

- Añadir la ruta $\${OPENNI2_INCLUDE}$ a la lista de *Includes* del proyecto
- Copiar los archivos disponibles en $\${OPENNI2_REDIST}$ en la carpeta de ejecución

- Añadir la carpeta de ejecución a la lista de rutas de búsqueda de librerías (opción `-L` del compilador)
- Añadir `libOpenNI2` a la lista de librerías (opción `-l` del compilador)
- Añadir la opción `"-Wl,-rpath ./"` en la llamada al `linker`.

A continuación crearemos un proyecto Eclipse siguiendo las indicaciones anteriores:

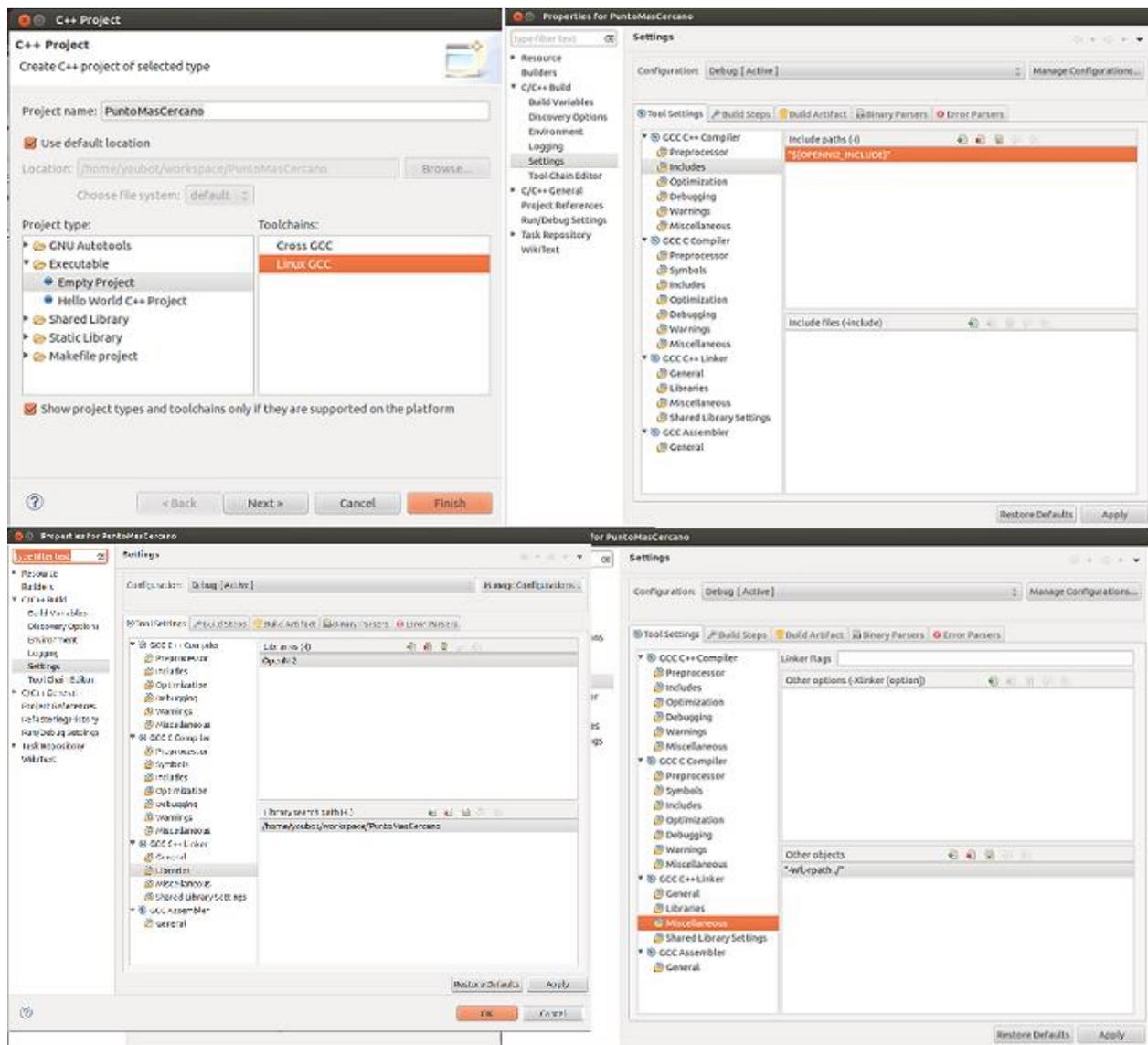


Figura 2.3 - Proyecto OpenNI en Eclipse

Suponiendo que Eclipse ha creado el proyecto en la ruta `~/workspace/PuntoMasCercano` copiaremos los archivos de redistribución de OpenNI a esta carpeta:

```
cd ~/workspace/PuntoMasCercano
cp -R ${OPENNI2_REDIST}/* .
```

Si no tenemos definida la variable de entorno a los archivos de redistribución (ver `OpenNI2DevEnvironment` en instrucciones de instalación) podemos reemplazar `${OPENNI2_REDIST}` por la ruta correspondiente.

2.1.3. PROGRAMACIÓN CON OPENNI

Algunas consideraciones a tener en cuenta a la hora de programar una aplicación con OpenNI:

1. Inicializar los drivers mediante *openni::OpenNI::initialize()*.
2. Si algo va mal, una llamada a *openni::OpenNI::getExtendedError()* puede ayudar a determinar los motivos.
3. Para identificar dispositivos compatibles conectados se puede usar *openni::OpenNI::enumerateDevices()*. SI no importa cuál (o si solo hay uno), se puede acceder usando *openni::ANY_DEVICE* como *URI*.
4. Cuando ya no necesitemos un stream de video que hayamos creado hay que liberarlo mediante *close()*.
5. Cuando ya no necesitemos un dispositivo hay que liberarlo mediante *destroy()*.
6. Al finalizar la aplicación llamar a *openni::OpenNI::shutdown()* para liberar los recursos OpenNI (*drivers*).

La instalación de OpenNI contiene numerosos ejemplos en la carpeta samples. A partir de ellos de ellos se ha programado el siguiente ejemplo imprime por pantalla las coordenadas (x,y,z) del punto más cercano detectado por el un sensor de profundidad (Asus Xtion, MS Kinect o cualquier otro compatible con OpenNI).

En el apartado 6.2 se muestra un ejemplo para determinar el punto más cercano detectado con una cámara de profundidad (Asus Xtion o MS Kinect) y mostrarlo por consola.

2.1.4. INTEGRACIÓN DEL SENSOR ASUS XTION EN ROS

Si pretendemos trabajar con ROS, entonces usaremos los paquetes de instalación correspondientes a la versión de ROS que utilicemos. Por ejemplo para ROS Fuerte el siguiente comando instalará el paquete ROS *openni_launch* así como las dependencias necesarias.

```
sudo apt-get install ros-fuerte-openni-launch
```

2.1.5. RECONOCIMIENTO ESQUELETAL CON OPENNI TRACKER

El siguiente programa muestra por pantalla el ángulo formado por el hombro, el codo y la muñeca derecha de una persona situada frente al sensor. Para ello se suscribe a los topics generados por el nodo *openni_tracker* (que es necesario arrancar previamente).

Una vez obtenidos los orígenes de coordenadas correspondientes a los tres elementos, calcula el ángulo formado por los vectores codo-hombro y codo-muñeca a partir de su producto escalar.

Listado de código fuente:

```

#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <skeleton2youbot/YouBotManipulatorJointAngles.h>

#define PI 3.1415927

tfScalar getScalarProduct(const tf::Vector3* v1, const tf::Vector3* v2) {
    return v1->x() * v2->x() + v1->y() * v2->y() + v1->z() * v2->z();
}

tfScalar getMagnitude(const tf::Vector3* v1) {
    return sqrt(getScalarProduct(v1,v1));
}

int main(int argc, char** argv){
    ros::init(argc, argv, "skeleton2youbot");

    ros::NodeHandle node;

    tf::TransformListener listener;

    ros::Rate rate(5.0);
    while (node.ok()){
        tf::StampedTransform transform_elbow_shoulder;
        tf::StampedTransform transform_elbow_hand;
        try{
            //ros::Time now = ros::Time::now();
            ros::Time now = ros::Time(0);
            listener.waitForTransform("/right_elbow_1", "/right_shoulder_1",
now,ros::Duration(3.0));
            listener.lookupTransform("/right_elbow_1", "/right_shoulder_1", now,
transform_elbow_shoulder);
            listener.lookupTransform("/right_elbow_1", "/right_hand_1", now, transform_elbow_hand);

            tf::Vector3 ES = transform_elbow_shoulder.getOrigin();
            tf::Vector3 EH = transform_elbow_hand.getOrigin();

            tfScalar sp = getScalarProduct(&ES,&EH);
            tfScalar mES = getMagnitude(&ES);
            tfScalar mEH = getMagnitude(&EH);

            if ((mES > 0)&&(mEH > 0)) {

                tfScalar cosTheta = sp / (mES * mEH);
                tfScalar theta = acos(cosTheta);

                printf("Angle S-E-H: %g (|ES| = %g, |EH| = %g)\n",theta * 180 / PI,mES,mEH);
            }
        } catch (tf::TransformException ex) {
            ROS_ERROR("%s",ex.what());
        }

        rate.sleep();
    }
    return 0;
};

```

2.2. Joystick USB genérico compatible Linux

En este apartado se describe el proceso de configuración de un joystick genérico compatible Linux para su utilización en ROS.



Figura 2.4 - Joystick USB compatible Linux

2.2.1. CONFIGURACIÓN DE UN JOYSTICK EN LINUX

El procedimiento de configuración de un joystick en Linux y su integración en ROS se detalla en [1] y se resume a continuación:

Instalar dependencias necesarias: paquetes de soporte de joystick para Linux:

```
sudo apt-get install joystick jstest-gtk
rosdep install joy
rosmake joy
```

Conectar el joystick y comprobar que Linux lo reconoce mediante *lsusb*

```
...
Bus 002 Device 005: ID 0079:0011 DragonRise Inc. Gamepad
```

y le asigna un archivo de dispositivo en la carpeta */dev* mediante *ls /dev/input/js**

```
crw-r--r--  1 root root 13, 0 Jun 25 12:46 /dev/input/js0
```

Ejecutar la utilidad de prueba de joystick *sudo jstest /dev/input/js0*

```
Driver version is 2.1.0.
Joystick (USB Gamepad ) has 2 axes (X, Y)
and 10 buttons (Trigger, ThumbBtn, ThumbBtn2, TopBtn, TopBtn2, PinkieBtn, BaseBtn, BaseBtn2,
BaseBtn3, BaseBtn4).
Testing ... (interrupt to exit)
Axes:  0:  0  1:  0 Buttons:  0:off  1:off  2:off  3:off  4:off  5:off  6:off  7:off
8:off  9:off
```

2.2.2. INTEGRACIÓN DE UN JOYSTICK EN ROS

Una vez configurado el joystick para funcionar en Linux, realizaremos los siguientes pasos para que funcione con ROS:

- Asignar permisos de lectura y escritura a todos los usuarios mediante `sudo chmod a+rw /dev/input/js0`
- Con el ROS Master en ejecución (roscore), indicar el dispositivo de entrada asignado al joystick mediante `rosparam set joy_node/dev "/dev/input/js0"`
- Iniciar el programa `joy_node` del paquete `joy` mediante `roslaunch joy joy_node`.

La salida de `joy_node` tras un movimiento del joystick será similar a esta:

```
---
header:
  seq: 792
  stamp:
    secs: 1372162213
    nsecs: 161655355
  frame_id: ''
axes: [-0.0, -1.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
---
```

Como se puede observar, los movimientos de la palanca se codifican en dos coordenadas, correspondientes a dos ejes perpendiculares.

Además el mensaje incluye el estado de los botones (1 pulsado, 0 no pulsado), el instante de tiempo de generación del mensaje y el número de secuencia (número que va incrementándose en uno en cada mensaje movimiento).

2.3. Cámara USB genérica compatible Linux

2.3.1. INSTALACIÓN DE CÁMARA EN LINUX



Figura 2.5 - Cámara USB compatible Linux

Tratándose de una cámara moderna lo más probable es que no requiera instalación de *drivers* y el sistema operativo la reconozca sin problemas. Para comprobar rápidamente si este es el caso, podemos utilizar la utilidad *cheese*. Para instalar *cheese*:

```
sudo apt-get install cheese
```

Sin más que ejecutarla veremos inmediatamente la imagen de la webcam:



Figura 2.6 – Visualización de imagen cámara USB mediante utilidad cheese

2.3.2. INTEGRACIÓN ROS DE CÁMARA USB

Existen varios paquetes ROS para captura de video de una cámara USB. En [1] se realiza una comparativa de varios *drivers*. Para este trabajo se ha seleccionado el paquete *uvc_cam* de Erick Perko por disponibilidad para ROS Fuerte y estar documentado con ejemplos en [2].

Este paquete no está disponible a través del gestor de paquetes de Ubuntu apt, por lo que instalaremos desde el código fuente. Suponiendo que nuestro espacio de trabajo ROS está en *~/fuerte_workspace*.

```
cd ~/fuerte_workspace
git clone https://github.com/ericperko/uvc_cam.git
rosmake uvc_cam
```

Una vez instalado ejecutamos el nodo ROS encargado de publicar la información recogida por la cámara en diferentes topics:

```
roslaunch uvc_cam test_uvc.launch
```

Los nuevos topics disponibles (listados mediante *rostopic list*) serán:

```
/camera/camera_info
/camera/image_raw
/camera/image_raw/compressed
/camera/image_raw/compressed/parameter_descriptions
/camera/image_raw/compressed/parameter_updates
/camera/image_raw/compressedDepth
/camera/image_raw/compressedDepth/parameter_descriptions
/camera/image_raw/compressedDepth/parameter_updates
/camera/image_raw/theora
/camera/image_raw/theora/parameter_descriptions
/camera/image_raw/theora/parameter_updates
```

Podemos visualizar la imagen publicada en el topic */camera/image_raw* mediante

```
roslaunch image_view image_view image:=/camera/image_raw
```

El resultado ha de ser una imagen como la obtenida mediante *cheese*.

2.3.3. CALIBRACIÓN

La calibración consiste en un conjunto de operaciones para compensar las desviaciones impuestas por los defectos en la fabricación de las lentes y en el montaje de la óptica. Es fundamental para poder relacionar las medidas de la imagen con las del mundo real.

En [3] se explica en detalle el proceso de calibración de una cámara y el significado de todos los parámetros involucrados. A continuación se resumen algunos conceptos básicos.

Según el modelo de cámara estenopeica (o de *pinhole*) las dimensiones de imagen y objeto se relacionan mediante la expresión:

$$-x = \frac{f \cdot X}{Z} \tag{2.1}$$

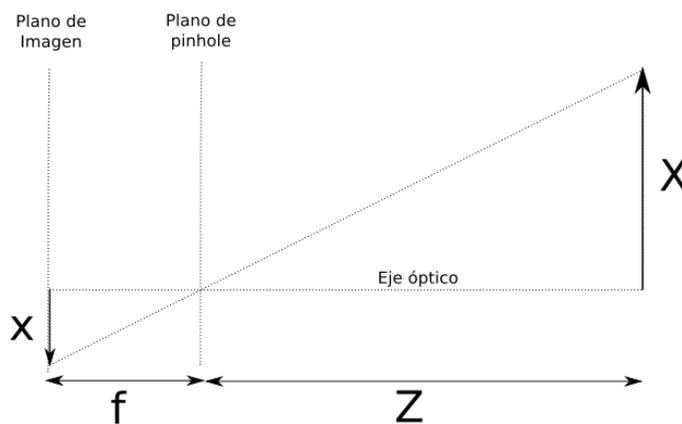


Figura 2.7 - Modelo de cámara estenopeica

Podemos obviar el signo utilizando un modelo equivalente en donde todos los rayos parten del centro de proyección y la imagen no está invertida.

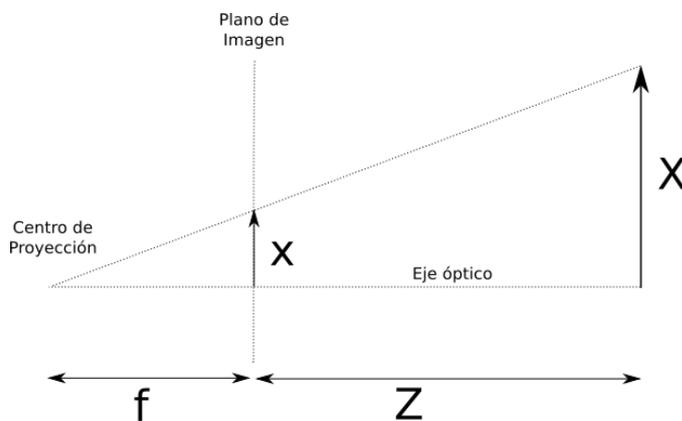


Figura 2.8 - Modelo equivalente sin inversión de imagen

Esta relación entre medidas reales y medidas de la imagen es cierta si el orificio (*pinhole*) está perfectamente alineado con el eje óptico. De no ser así se produciría una desviación c respecto al origen de coordenadas en el plano de imagen. De modo que la expresión sería (prescindiendo ya del signo).

$$\mathbf{x} = \frac{\mathbf{f} \cdot \mathbf{X}}{Z} + \mathbf{c} \quad (2.2)$$

Normalmente los píxeles de las cámaras de bajo coste son rectangulares y no cuadrados, lo que podemos considerar en forma de dos distancias focales distintas. Así tendremos:

$$x = \frac{f_x \cdot X}{Z} + c_x \quad (2.3)$$

$$y = \frac{f_y \cdot Y}{Z} + c_y \quad (2.4)$$

La relación de correspondencia entre las coordenadas del mundo real y el plano de imagen se denomina **transformación proyectiva** y se representa habitualmente en coordenadas homogéneas, facilitando la agrupación de los **parámetros intrínsecos** de la cámara (f_x , f_y , c_x y c_y) en una misma matriz \mathbf{M} . Llamando \mathbf{Q} (X, Y, Z) a las coordenadas tridimensionales del mundo real y \mathbf{q} (x, y, w) a las coordenadas bidimensionales (x, y) de la imagen con una componente adicional ($w = Z$) incluida por conveniencia para la representación en coordenadas homogéneas, se cumple que:

$$q = MQ, \quad q = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \quad M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (2.5)$$

La determinación de la matriz de parámetros intrínsecos \mathbf{M} es uno de los objetivos de la calibración.

Otros dos parámetros a determinar son la **distorsión radial** y la **distorsión tangencial**. El primero se debe a la forma de la lente (resulta más económico fabricar lentes esféricas que parabólicas) y es responsable de efectos indeseados en los bordes del plano de imagen, como el efecto “ojo de pez”. El segundo se debe a falta de paralelismo entre la lente y el plano de imagen, dando lugar al estrechamiento de la imagen en uno de sus lados.

Hay tres parámetros relevantes de distorsión radial y dos de tangencial que se suelen agrupar en un **vector de distorsión**.

2.3.4. PROCEDIMIENTO DE CALIBRACIÓN CON TABLERO DE AJEDREZ

La calibración se suele realizar mediante el análisis de un objeto plano desde diferentes puntos de vista, siendo uno de los patrones más utilizados el de tablero de ajedrez.

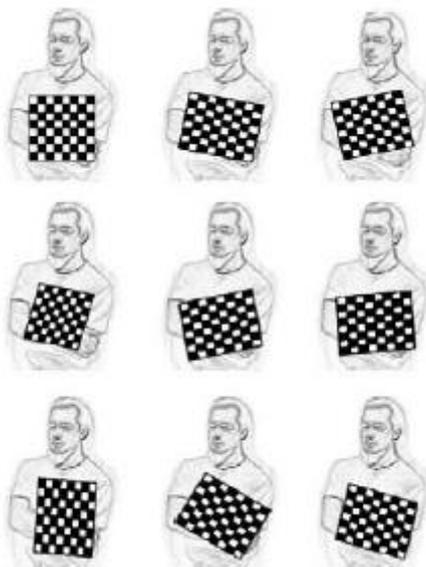


Figura 2.9 - Procedimiento de calibración con patrón de ajedrez.

La correspondencia entre estos planos se realiza mediante rotaciones y traslaciones que, en coordenadas homogéneas, se reducen a productos de matrices. La correspondencia proyectiva de un plano sobre otro se denomina **homeografía planar**.

Como se explica detalladamente en [3], para determinar los 4 factores intrínsecos (matriz M) y los 5 extrínsecos (3 radiales y dos tangenciales) bastan 2 vistas de un patrón con 3 x 3 bordes internos (cuadrícula de 4 x 4). En la práctica se utilizan al menos 10 vistas de patrones con 7 x 8 bordes internos.

2.3.5. CALIBRACIÓN CON OPENCV A TRAVÉS DE ROS

El paquete ROS *camera_calibration* realiza las operaciones descritas anteriormente mediante las librerías OpenCV. Para instalarlo ejecutamos los siguientes comandos.

```
rosdep install camera_calibration
rosmake camera_calibration
```

Necesitamos imprimir un patrón de ajedrez (en la web del paquete se puede descargar uno). En este caso se ha utilizado un patrón de 10 x 7 bordes internos como el de la figura siguiente, con cuadrados de lado 25mm.

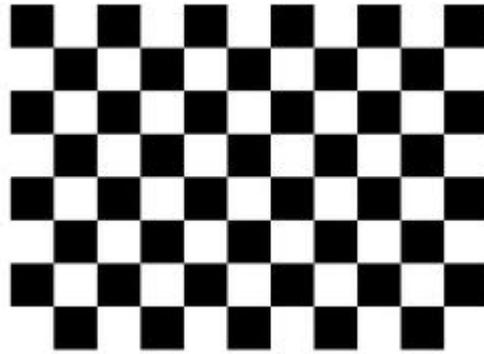


Figura 2.10 - Patrón de ajedrez

Antes de ejecutar el paquete *camera_calibration* los topics de imagen deben estar publicados. En este caso se ha utilizado el paquete *uvc_cam* para obtener la información de imagen

```
roslaunch uvc_cam test_uvc.launch
```

Y ejecutamos el nodo de calibración

```
rosruncamera calibration cameracalibrator.py --size 10x7 --square 0.025
image:=/camera/image_raw camera:=camera
```

Ahora, visualizando la imagen captada por la cámara para comprobar que el tablero completo está contenido dentro del cuadro de imagen lo presentamos desde diversas distancias y orientaciones (cuantas más y más variadas mejor).

Tras un tiempo de captura de muestras, el botón *calibrate* se activará. Lo pulsamos y esperamos unos minutos hasta que el proceso nos permita pulsar los botones *save* y *commit*.

```
Wrote calibration data to /tmp/calibrationdata.tar.gz
D = [-0.09743308996747745, 0.4791795930788984, 0.020704850787040267, 0.005314147621121779,
0.0]
K = [1107.4654595535123, 0.0, 321.74654316096706, 0.0, 1109.9643878744796, 291.3033994907627,
0.0, 0.0, 1.0]
R = [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]
P = [1102.7243603515626, 0.0, 322.94461220802174, 0.0, 0.0, 1098.22875, 295.1489078700752,
0.0, 0.0, 0.0, 1.0, 0.0]
# oST version 5.0 parameters

[image]

width
640

height
480

[narrow_stereo/left]

camera matrix
1107.465460 0.000000 321.746543
0.000000 1109.964388 291.303399
0.000000 0.000000 1.000000

distortion
```

```
-0.097433 0.479180 0.020705 0.005314 0.000000  
  
rectification  
1.000000 0.000000 0.000000  
0.000000 1.000000 0.000000  
0.000000 0.000000 1.000000  
  
projection  
1102.724360 0.000000 322.944612 0.000000  
0.000000 1098.228750 295.148908 0.000000  
0.000000 0.000000 1.000000 0.000000
```

Y fijándonos en la ventana de ejecución de *uvc_cam*, podemos comprobar cómo al finalizar la calibración éste detecta la disponibilidad de los datos y los almacena

```
[ INFO] [1373304579.470795903]: writing calibration data to  
/home/nando/.ros/camera_info/camera.yaml  
[ INFO] [1373304579.531564097]: [camera] calibration matches video mode now
```

En cuanto a los resultados de calibración buscados:

9. **M** (matriz de parámetros intrínsecos) viene indicada como **K** y *camera matrix*.
10. El **vector de distorsión** viene indicado como **D** y *distortion*.

2.5. Medidor láser Sick LMS200

El sistema de medición Sick LMS200 realiza barridos del entorno con un campo de visión radial utilizando haces de luz láser infrarroja.

A este tipo de sistemas se les conoce también como *lidar* (o radar láser) o sensores de tipo *time-of-flight* y se utilizan principalmente para monitorización de áreas, detección y medición de objetos o determinación de posiciones.



Figura 2.11- Medidor laser Sick LMS200

El principio de funcionamiento consiste en un haz de luz pulsante que se refleja en los objetos del entorno y es detectado por un receptor. El tiempo transcurrido entre la emisión del haz y la detección de su reflejo es proporcional a la distancia entre el sensor y la superficie reflectante.

Las principales ventajas de los sensores *lidar* son:

- Medición sin necesidad de contacto.
- Rapidez de barrido. Permiten medir objetos en movimiento.
- No requieren propiedades especiales en las superficies a medir, reflectores ni marcadores (aunque su alcance depende de la reflectividad).
- Proporcionan datos en tiempo real, por lo que son aptos para actividades de control.
- Son sistema activos, no requiere iluminación externa.

En las unidades Sick LMS200 la dirección del haz pulsante se modifica por un espejo rotatorio interno. En función de la configuración seleccionada, se emiten pulsos con una resolución angular de 0.25°, 0.5° ó 1°. La resolución es de 10 mm, y el error de medida típico es de ± 15 mm. El alcance típico (considerando superficies con un 10% de reflectividad) es de 10m.

El sensor Sick LMS200 puede realizar operaciones de preprocesamiento de las mediciones, como promediar valores de entre 2 y 250 barridos o transmitir únicamente información de un sector determinado (100° ó 180°).

Para extraer los datos del sensor se utiliza interfaz serie (RS-232 ó RS-422). La unidad se alimenta con 24V DC.

2.5.1. CONFIGURACIÓN PARA USO CON ROS

El soporte ROS lo proporciona el paquete *sicktoolbox_wrapper*. Para instalarlo ejecutaremos:

```
rosdep install sicktoolbox_wrapper
```

Conviene asegurarse de que el puerto serie tiene permisos de lectura y escritura para todos los usuarios. De no ser así los asignamos. Si el dispositivo serie es */dev/ttyUSB0*:

```
chmod a+rw /dev/ttyUSB0
```

Es necesario establecer el parámetro *sicklms/port* con el nombre del dispositivo serie y la velocidad de comunicación. Ejemplo:

```
rosparam seticklms/port /dev/ttyUSB0
rosparam seticklms/ baud 38400
```

Una vez configurados los parámetros de comunicación podemos ejecutar *sicklms*:

```
roslaunch sicktoolbox_wrappericklms
```

La salida será similar a esta:

```
[ WARN] [1373553974.127271103]: The use_rep_117 parameter has not been set or is set to false.
Please see: http://ros.org/wiki/rep_117/migration
*** Attempting to initialize the Sick LMS...
  Attempting to open device @ /dev/ttyUSB0
    Device opened!
  Attempting to start buffer monitor...
    Buffer monitor started!
  Attempting to set requested baud rate...
    Operating @ 38400bps
  Attempting to sync driver...
    Driver synchronized!
*** Init. complete: Sick LMS is online and ready!
Sick Type: Sick LMS 200-30106
Scan Angle: 180 (deg)
Scan Resolution: 0.5 (deg)
Measuring Mode: 8m/80m; 3 reflector bits
Measuring Units: Millimeters (mm)

[ INFO] [1373553974.879613188]: Variant setup not requested or identical to actual (180,
0.500000)
[ INFO] [1373553974.879738378]: Measuring units setup not requested or identical to actual
('Millimeters (mm)')
  Requesting partial scan data stream...
    Data stream started!
```

A partir de este momento los datos medidos se publicarán en el *topic scan*. Los mensajes publicados son de tipo *sensor_msgs/LaserScan*:

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

donde los ángulos están expresados en radianes, el tiempo entre medidas y el tiempo entre barridos en segundos, y los valores de rango en metros.

Podríamos querer además almacenar una sesión de lectura para procesarla o analizarla en otro momento. Para ello usaremos *roscap*.

```
nando@mac:~$ roscap record -o sick_scan -l 1000
[ INFO] [1373554290.270588613]: Subscribing to scan
[ INFO] [1373554290.273062717]: Recording to sick_2013-07-11-16-51-30.bag.
```


3. ACTUADORES

3.1. Actuadores Robotis Dynamixel

Los actuadores Robotis Dynamixel son sistemas que integran motor, reducción, controlador y red de interconexión. Entre otras aplicaciones, estos actuadores se utilizan en sistemas educativos de robótica **Bioloid**.



Figura 3.1 - Actuadores Dynamixel AX12A y AX12W

Existen numerosos modelos con características muy distintas. A continuación se describen los modelos utilizados en este proyecto.

3.1.1. ACTUADORES AX-12A Y AX-12W

- Constituyen la gama más económica de Dynamixel.
- Pesan menos de 55 g y sus dimensiones son 32x50x40 mm.
- La reducción del modelo AX-12A es de 254:1, mientras que la del AX-12W es de 32:1.
- Disponen de sensor de posición con una resolución de $(300^\circ / 1024 = 0.3^\circ)$.
- Pueden trabajar con tensiones de entre 9 y 12V DC.
- Disponen de accesorios de fijación para combinar fácilmente varios actuadores
- Se controlan mediante un bus TTL (TTL level multidrop) en el que cada actuador tienen un identificador único. La interconexión se realiza en configuración *daisy chain*, lo que facilita el cableado.
- Admiten configuración de par con una granularidad de 1024 niveles.

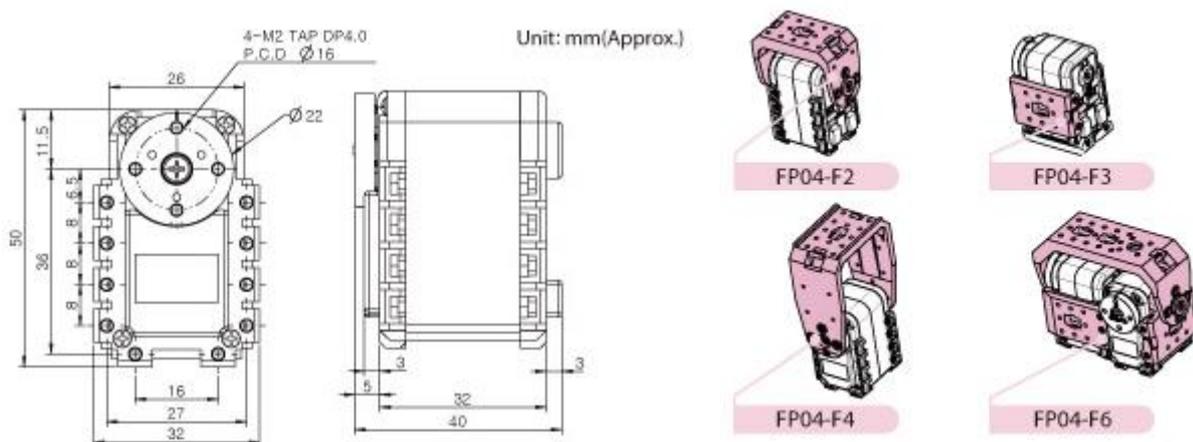


Figura 3.2 - Dimensiones y configuraciones de montaje de actuadores Dynamixel AX

3.1.2. ACCESO A LA CONFIGURACIÓN E INFORMACIÓN DE ESTADO

Estos actuadores disponen de una zona de memoria EEPROM donde se almacena la configuración de la unidad (modelo, versión de firmware, velocidad de comunicación, límites de posición, etc.) y una zona RAM donde se pueden consultar parámetros de funcionamiento (voltaje, velocidad, temperatura, carga, etc.)

3.1.3. DYNAMIXEL SDK

Existe un SDK en lenguaje C disponible para acceder a los actuadores programáticamente disponible la web de Robotis.

3.1.4. USB2DYNAMIXEL

Además de mediante un controlador Dynamixel, es posible acceder a los actuadores directamente mediante un cable USB basado en el integrado FTDI. El cable dispone de conector DB9 u conector molex de 4 pines para RS-232 y RS-485 en los modelos de alta gama, y molex de 3 pines para los modelos TTL (modelos AX). A continuación se muestra el detalle de conexión para TTL:

3 Pin Cable		
Pin No.	Signal	Pin Figure
1	GND	
2	NOT Connected	
3	DATA (TTL)	

Figura 3.3 - Conexión actuadores AX

3.1.5. CONTROLADOR DE MOTORES CM-900

El CM-900 es un controlador de motores Dynamixel *open source* de bajo coste basado en microcontrolador Cortex-M3 que permite entre otras cosas controlar los actuadores Dynamixel programando en lenguaje *Sketch* desde el IDE Robotis (inspirado en Arduino).

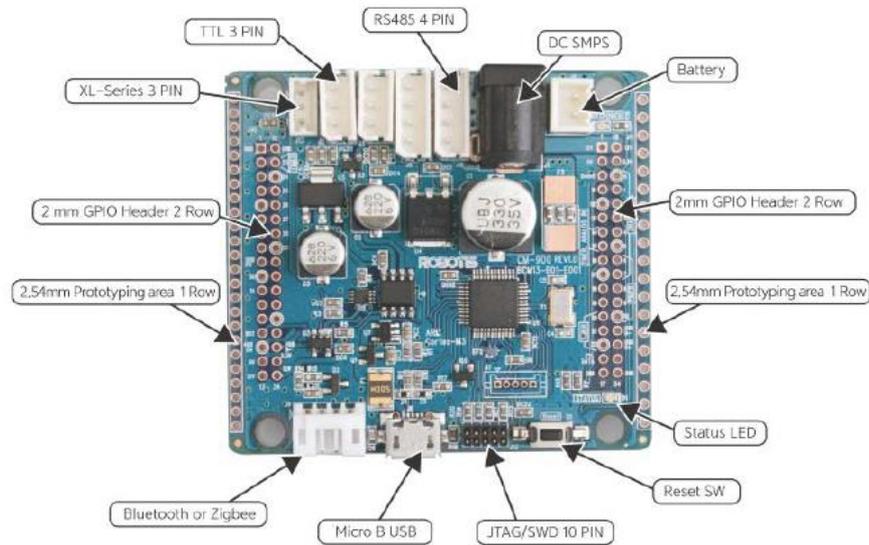


Figura 3.4. - Controlador de actuadores CM-900

Entre las muchas aplicaciones que se pueden desarrollar de este controlador, una de especial interés es la de comunicar un PC al bus de actuadores. Es decir, puede emular la funcionalidad del cable USB2Dynamixel a menos de la mitad de precio. El código fuente para esta funcionalidad está disponible en [3] (ver artículo *Robotis CM-900 as a tosser for Dynamixel commands*). Para programarla es necesario instalar previamente el IDE Robotis.

3.1.6. INTEGRACIÓN ROS

Existe soporte ROS para estos actuadores mediante el *stack dynamixel_motor*. Para instalarlo en ROS Fuerte:

```
sudo apt-get install ros-fuerte-dynamixel-motor
```

Siguiendo los tutoriales disponibles en el sitio web de ROS y utilizando un actuador AX-12A y controlador CM-900 convenientemente programado para dar acceso al bus de actuadores, se puede comprobar la integración con ROS mediante los siguientes procedimientos.

Acceso al bus y visualización de datos

- Crear un paquete ROS añadiendo la dependencia *dynamixel_controllers*

```
roscatkin create_pkg test_dynamixel dynamixel_controllers
```

- Crear una carpeta llamada *launch* dentro del paquete, y un archivo *controller_manager.launch* en la carpeta con los detalles de nuestra configuración. Por ejemplo, suponiendo que en el bus solamente hay un actuador AX-12A con identificador único 1 (es el valor de fábrica), y que estamos usando el CM-900 (reconocido por Linux como ROBOTIS Virtual COM Port, y normalmente accesible mediante */dev/ttyACM0*) la configuración sería la siguiente:

```
<launch>
  <node name="dynamixel manager" pkg="dynamixel_controllers" type="controller_manager.py"
    required="true" output="screen">
    <rosparam>
      namespace: dxl manager
      serial ports:
        pan tilt port:
          port_name: "/dev/ttyACM0"
          baud_rate: 1000000
          min motor id: 1
          max motor id: 2
          update rate: 20
    </rosparam>
  </node>
</launch>
```

- Ejecutar el archivo launch recién creado

```
roslaunch test_dynamixel controller_manager.launch
```

- Visualizar mediante *rostopic* los valores del actuador AX-12A publicados

```
rostopic echo /motor_states/pan_tilt_port
```

```
motor_states:
-
  timestamp: 1373075843.41
  id: 1
  goal: 835
  position: 835
  error: 0
  speed: 0
  load: 0.0
  voltage: 12.3
  temperature: 37
  moving: False
---
```

El mensaje mostrado indica que el actuador con ID 1 ha recibido un comando para situarse en la posición 835 (siendo los valores posibles de 0 a 1023) y se encuentra precisamente en esta posición. Como consecuencia de esto el error es 0, la velocidad es 0 y la variable *moving* tiene

valor *False*. Además, en este momento la carga soportada es 0 (se puede ver variar ese parámetro tratando de forzar el servo ligeramente con la mano).

Control de los actuadores conectados al bus

1. Creación de un archivo de configuración *tilt.yaml*. con el siguiente contenido en la carpeta raíz del paquete.

```
tilt_controller:
  controller:
    package: dynamixel_controllers
    module: joint_position_controller
    type: JointPositionController
  joint_name: tilt_joint
  joint_speed: 1.17
  motor:
    id: 1
    init: 512
    min: 0
    max: 1023
```

2. Creación de un archivo *start_tilt_controller.launch* en la carpeta launch del paquete

```
<launch>
  <!-- Start tilt joint controller -->
  <rosparam file="$(find test_dynamixel)/tilt.yaml" command="load"/>
  <node name="tilt_controller_spawner" pkg="dynamixel_controllers"
type="controller_spawner.py"
    args="--manager=dxl_manager
        --port pan_tilt_port
        tilt_controller"
    output="screen"/>
</launch>
```

3. Iniciar el *controller_manager* (ver epígrafe anterior)

```
roslaunch test_dynamixel_controller_manager.launch
```

4. Iniciar el *start_tilt_controller*

```
roslaunch test_dynamixel_start_tilt_controller.launch
```

5. Comprobar la lista de topics

```
rostopic list
```

```
/diagnostics
/motor_states/pan_tilt_port
/rosout
/rosout_agg
/tilt_controller/command
/tilt_controller/state
```

6. Mover el motor mediante publicación de un mensaje en el topic */tilt_controller/command*

```
rostopic pub -1 /tilt_controller/command std_msgs/Float64 -- 1.5
```

Conviene destacar un inconveniente de estos actuadores, documentado en [4] (*12.3 A Note Regarding Dynamixel Hardware*): los conectores de los actuadores son muy sensibles, de modo que un ligero movimiento entre el macho y la hembra puede causar pérdida de señal y potencialmente un fallo en los *drivers*, siendo en ocasiones necesario reiniciar ambos (los servos y los *drivers*). Por ello se recomienda el uso de bridas u otros medios para asegurar la fijación de los cables a los conectores.

Probablemente debido a este factor, se ha observado que las órdenes enviadas mediante `rostopic pub` no siempre resultan en movimiento de los actuadores.

3.3. Unidad pan-tilt PTU-D46

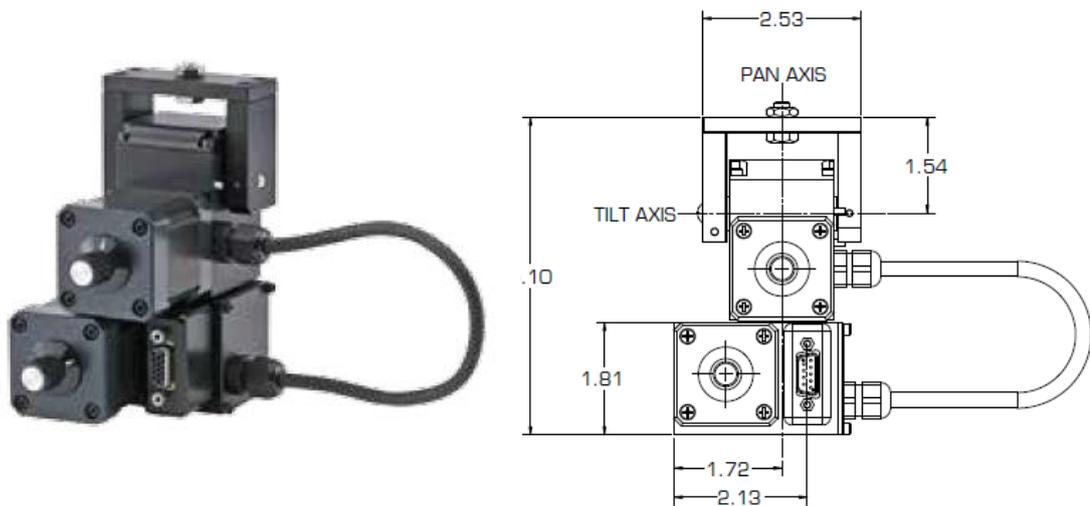


Figura 3.5 - Unidad pan-tilt PTU-D46

PTU-D46 es una familia de unidades *pan-tilt* en miniatura, programables en velocidad, aceleración, potencia y otros parámetros. Entre sus principales características se encuentran las siguientes:

- Interfaz RS-232 o RS-485.
- Aceptan comandos ASCII o binarios (API propietaria, no libre).
- Admiten un máximo de 60 comandos por segundo.
- Pueden moverse a velocidades de hasta 300° por segundo.
- En función del modelo, soportan hasta 4 Kg de carga.
- Resolución de 3 milésimas de grado.
- Rango PAN: -47° a 31°
- Rango TILT: +/- 159°

3.3.1. CONEXIONES

- Cable rojo etiquetado como "V AJUSTABLE": alimentación motores (entre 9 y 30V).
- Cable rojo etiquetado como "5V": alimentación controlador (5V)
- Cable negro etiquetado como "0V": GND.
- Conexión cable RS-232 (DB9) a puerto serie

3.3.2. OPERATIVA BÁSICA

La unidad pan/tilt admite un gran número de comandos y configuraciones que se escapan al alcance de este proyecto y no se detallarán en este capítulo. A continuación se introducen brevemente los comandos relevantes para los ejercicios realizados.

La unidad PTU se controla mediante comunicación serie RS-232 (con 9600 baudios, sin paridad, 8 bits de datos y 1 bit de parada). Los comandos de posición se especifican en pasos. La resolución o relación entre pasos y segundos de grado es por defecto de 185.1428 segundos/paso. Así, 10° equivalen a $10 \times 3600 / 185.1428 = 195$ pasos.

Una orden de posicionamiento relativo de 10° en eje PAN y de -20° en eje TILT con ejecución inmediata sería

```
I
PO195
TO-390
```

Detrás de cada comando la unidad PTU espera siempre un caracter separador, que puede ser un espacio en blanco o un retorno de carro.

Para llevar los ejes PAN y TILT a sus orientaciones iniciales con posicionamiento absoluto y ejecución inmediata y esperar a que los comandos se completen.

```
I
PP0
TP0
A
```

Para llevar los ejes PAN y TILT a la posición 500 de manera simultánea (es decir, no empezar el movimiento PAN antes que el TILT) con posicionamiento absoluto y esperar a que los comandos se completen.

```
S
PP0
TP0
A
```

Para interrogar a la unidad PTU-46 sobre la posición actual del eje PAN

```
PP
```

y el eje TILT

```
TP
```

Para forzar la detención inmediata de la unidad

H

La unidad responderá con un asterisco (*) si acepta el comando, seguido del dato solicitado si se trata de una pregunta. En caso de error devolverá un signo de exclamación (!) seguido de los detalles del error.

Para probar la comunicación desde Linux se puede usar la aplicación de terminal *minicom*.

3.3.3. INTEGRACIÓN ROS

No disponible públicamente (o, al menos, no encontrada). Para integrar este dispositivo en ROS se ha desarrollado una librería C++ para comunicación serie mediante *termios* disponible en los apéndices.

La integración con ROS de la unidad pan/tilt consistirá en la comunicación bidireccional con el dispositivo mediante *topics* utilizando mensajes de tipo *JointState*. que contienen los siguientes campos:

```
Header header
string[] name
float64[] position
float64[] velocity
float64[] effort
```

El nodo de control se encargará de la traducción de mensajes ROS a comandos serie y viceversa.

El código fuente del nodo de integración desarrollado se incluye en el apartado **6.3** de los apéndices.

3.4. Kuka Youbot

3.4.1. DESCRIPCIÓN GENERAL



Figura 3.6 - Manipulador móvil Kuka YouBot

Kuka Youbot es un manipulador móvil específicamente diseñado para educación e investigación que permite un control total sobre sus sensores y actuadores a través de software libre. Consta de los siguientes elementos:

1. Plataforma móvil omnidireccional con ordenador integrado.
2. Brazo con 5 grados de libertad y una pinza. Puede funcionar controlada por la plataforma móvil o por otro ordenador.

3.4.2. PLATAFORMA MÓVIL

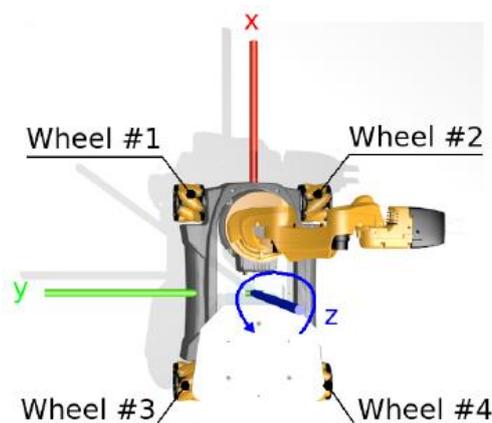


Figura 3.7 - Plataforma Móvil Kuka YouBot

La plataforma móvil omnidireccional utiliza 4 ruedas *mecanum* (ruedas suizas). El acceso al control de los motores se realiza mediante interfaz *EtherCAT*.

3.4.3. BRAZO

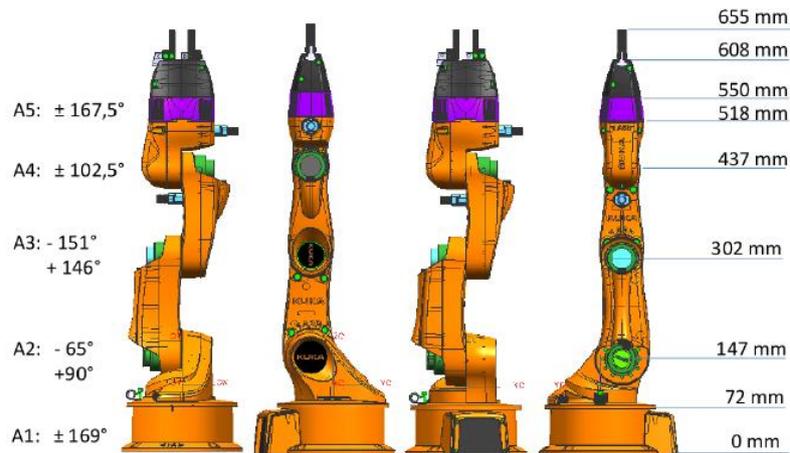


Figura 3.8 - Brazo manipulador Kuka YouBot

El brazo del Youbot forma una cadena cinemática con 5 ejes de revolución terminado en un efector con pinza. Como en la plataforma móvil, el acceso al control de los motores se realiza mediante *EtherCAT*.

3.4.4. MANEJO DEL ROBOT

Todos los actuadores cuentan con encoders incrementales, por lo que no se dispone de una posición absoluta de las uniones tras el arranque de la plataforma. Por eso es necesario que antes de iniciar ninguna operación con el brazo del Youbot este se posicione manualmente en la posición inicial (*home*).

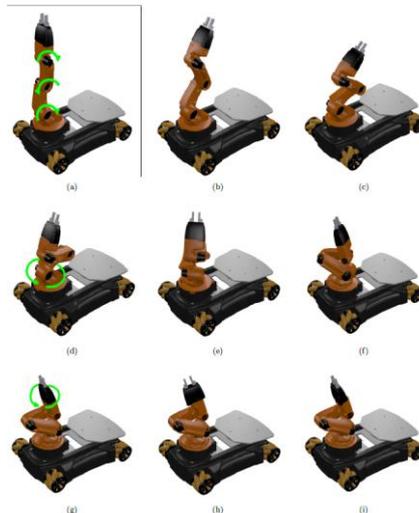


Figura 3.9 - Secuencia de posicionamiento manual del brazo

Con el robot se proporciona una distribución USB-Live de Linux Ubuntu 12.04 ya preparada con la configuración y aplicaciones necesarias para empezar a trabajar con el robot, que puede ser instalada en el ordenador integrado en la base móvil o en cualquier otro ordenador.

Youbot se puede controlar mediante su API C++ o y/o mediante ROS Fuerte. Las herramientas necesarias para ambos modos se proporcionan preinstaladas a través del lápiz USB.

3.4.5. PROGRAMACIÓN MEDIANTE KUKA YUBOT API

El API Kuka Youbot OODL es un conjunto de librerías C++ *open source* para controlar el YouBot a nivel de sus uniones. Proporciona control total de la funcionalidad *firmware*, encapsula la comunicación EtherCAT y soporta sistema operativo Linux. No depende de ninguna plataforma y en la actualidad existen integraciones ROS y Orocos.

Permite comandar y monitorizar posición, velocidad y corriente en las uniones, así como establecer parámetros (como valores PID)

Permite mover la base en un espacio cartesiano especificando velocidad lineal y angular.

Para el API el brazo está representado por una cadena cinemática de 5 grados de libertad y la base móvil por un conjunto de uniones de revolución. Utiliza las siguientes clases (C++) para acceder a los diferentes elementos:

- **YouBotManipulator**: representa al YouBot como agregación de uniones y una pinza
- **YouBotBase**: representa la plataforma omnidireccional
- **YouBotJoint**: representa una unión de la base o del brazo

Documentación de Referencia del API: https://github.com/youbot/youbot_driver/wiki

3.4.6. DESCRIPCIÓN DEL YUBOT PARA SIMULACIÓN

La integración de ROS en el YouBot incluye una descripción URDF para su uso en simulación con *rviz* o *Gazebo*. Antes de usarla por primera vez es necesario compilarla.

```
rosmake youbot_description
```

Para visualizarla basta ejecutar los siguientes comandos:

```
roscore&  
roslaunch youbot_description joint_robot_publisher  
roslaunch youbot_description youbot_description.launch  
roslaunch rviz rviz
```

Es posible representar en simulación los movimientos del robot real mediante publicación del estado de sus uniones

```
roslaunch youbot_oodl youbot_joint_state_publisher.launch
```

4. INTEGRACIÓN

4.1. Interacción Asus Xtion + PTU-D46 para seguimiento de punto más cercano

4.1.1. INTRODUCCIÓN

En este capítulo se describe un ejemplo de interacción entre un sensor de profundidad Asus Xtion y la unidad pan tilt PTU-46 para realizar una aplicación de seguimiento.



Figura 4.1 - Cámara Asus Xtion sobre unidad pan-tilt

En el apartado **6.2** de los apéndices se incluye un programa ejemplo para obtener el punto más cercano detectado por un sensor de profundidad 3D utilizando OpenNI. En este apartado partiremos de este código y de la librería de comunicación serie también disponible en los apéndices utilizaremos las coordenadas del punto más cercano detectado para orientar la base pan tilt.

La estrategia consiste en reorientar la base pan-tilt buscando posicionar el punto más cercano detectado en las coordenadas $(x,y) = (0,0)$.

En este ejercicio no ha sido necesario el uso de ROS puesto que se ha accedido directamente al API OpenNI de la cámara y al puerto serie.

4.1.2. DETERMINACIÓN DE LOS PARÁMETROS DE ORIENTACIÓN PAN Y TILT

Algunas consideraciones iniciales:

1. La cámara Asus Xtion toma como eje Z al eje perpendicular al plano de imagen, de modo que si colocamos la cámara enfocándonos a nosotros, el eje Y es positivo hacia arriba y el eje X hacia nuestra derecha.
1. Consideraremos *tilt* al ángulo de elevación necesario rotando sobre el eje X para alinear P con el eje Z una vez corregido el ángulo *pan* (o, lo que es lo mismo, el ángulo formado por el vector OP sobre el plano XZ).
1. La cámara se posicionará sobre la base pan-tilt, de modo que un tilt de 0° significa que la línea proyectiva es paralela al plano XZ.

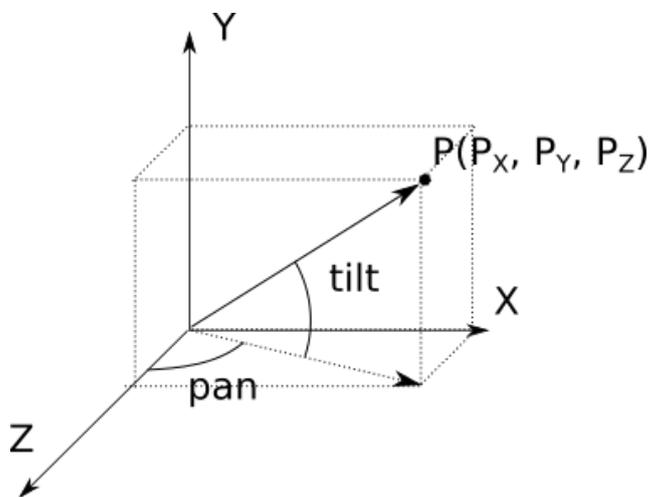


Figura 4.2 – Ángulos de orientación pan-tilt para centrar el objetivo

El objetivo es situar el punto P sobre el eje Z mediante ajuste de los ángulos *pan* y *tilt*.

Si la ubicación del sensor coincide con el punto de intersección de los ejes *pan* y *tilt*, partiendo de las coordenadas cartesianas $P(P_x, P_y, P_z)$, y llamando OP al vector que une el origen de coordenadas del sensor con el punto P podemos determinar el ángulo *tilt* proyectando OP sobre el eje Y. El ángulo *pan* será el formado por la proyección de OP sobre el plano XZ con el eje Z. Así:

$$|OP| \operatorname{sen}(tilt) = P_y \quad (4.1)$$

$$\operatorname{tg}(pan) = P_x / P_z \quad (4.2)$$

4.1.3. EFECTO DEL DESAJUSTE ENTRE SENSOR DE IMAGEN Y EJE TILT

En este caso, las dimensiones del sensor de profundidad y el modo de fijación de ésta al actuador *pan-tilt* hacen que exista una distancia de 70 mm entre el sensor de imagen y el eje *tilt*, que puede ser apreciable en seguimiento de objetos cercanos.

En la figura siguiente se representa el desajuste mediante el vector OC_1 . Se representan dos dimensiones, correspondientes al plano definido por C_1 , el eje Y y el punto P, por lo que la coordenada $P_{Z'}$ en este plano es

$$P_{Z'} = \frac{P_z}{\cos(\text{pan})} \quad (4.3)$$

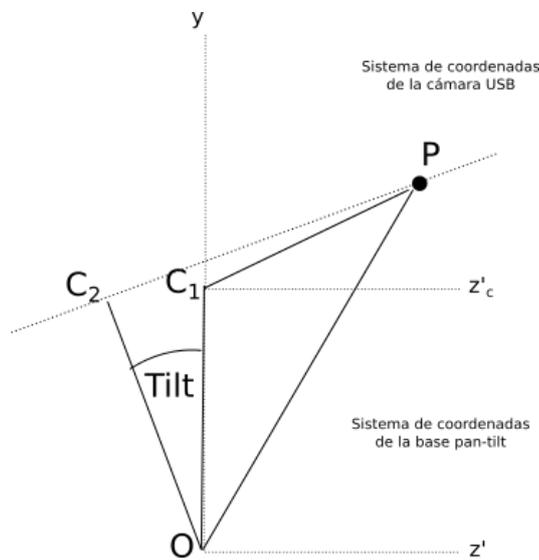


Figura 4.3 - Desajuste entre sensor de imagen y eje tilt

El ángulo de corrección *tilt* coincide la pendiente de la recta que pasa por C_2 y P (que será el nuevo eje Z_C de la orientación objetivo).

Utilizando coordenadas referidas a la base *pan-tilt* sabemos que:

$$C_{2z} = -L \cdot \text{sen}(\text{tilt}) \quad (4.4)$$

$$C_{2y} = L \cdot \text{cos}(\text{tilt}) \quad (4.5)$$

Por lo que, llamando $|OC_1| = |OC_2| = L$ (distancia entre sensor y eje *tilt*) y $|C_2P| = d$ (distancia entre el sensor y el punto P tras realizar la corrección *tilt*).

$$d \cdot \text{cos}(\text{tilt}) = \frac{P_z}{\cos(\text{pan})} + L \cdot \text{sen}(\text{tilt}) \quad (4.6)$$

$$d \cdot \text{sen}(\text{tilt}) = P_y - L \cdot \text{cos}(\text{tilt}) \quad (4.7)$$

Además, se cumple que:

$$|OP|^2 = d^2 + L^2 \quad (4.8)$$

Operando podemos determinar el ángulo *tilt* a partir de las coordenadas de P y la distancia L entre el eje *tilt* y el sensor según la expresión siguiente:

$$\text{sen}(\textit{tilt}) = \frac{(|OP|^2 - L^2)^{1/2} \cdot P_Y - L \cdot \frac{P_Z}{\cos(\textit{pan})}}{|OP|^2} \quad (4.9)$$

Si sustituimos L=0 podemos comprobar que la expresión es consistente con la expresión (4.1) indicada previamente correspondiente al caso sin desajuste entre sensor y eje *tilt*.

Se ha comprobado que el desajuste entre la posición del sensor de imagen y el eje *tilt* tiene un efecto apreciable. Sin compensación del desajuste la corrección tilt es mayor de lo debido, por lo que se produce una oscilación constante en el seguimiento vertical. Por el contrario, con compensación la oscilación desaparece.

El código fuente de este ejercicio está disponible en el apartado **6.4** de los apéndices.

4.2. Teleoperación de unidad pTU-46 mediante Joystick

4.2.1. INTRODUCCIÓN

Este ejercicio consiste en controlar el movimiento de una unidad PTU-46 mediante un Joystick USB compatible Linux.

Los movimientos del joystick se publicarán en el topic ROS `/joy` utilizando el paquete ROS Joy. Un nodo ROS se suscribirá al topic Joy y se comunicará con la unidad PTU-46 via serie. Los mensajes procedentes del joystick se traducirán a comandos de posición relativa de la unidad, asociando izquierda y derecha con el eje *PAN*, y arriba y abajo con el eje *TILT*. Una indicación direccional del joystick equivaldrá a un giro relativo de 10° del eje correspondiente (*PAN* o *TILT*) de la unidad. Así, para dar media vuelta al eje *PAN* cuando éste está situado en el extremo derecho de su recorrido es necesario pulsar 18 veces el joystick hacia la izquierda.

La unidad PTU-46 es capaz de orientar un objeto respecto el eje Z (movimiento PAN) y respecto al plano horizontal (movimiento TILT). Este tipo de actuadores es de gran utilidad en la orientación de cámaras de video-vigilancia (comúnmente conocidas como cámaras PTZ o Pan-Tilt-Zoom). Admite comandos de posición y velocidad mediante una conexión serie RS-232

El Joystick utilizado es compatible Linux y dispone de conexión USB. Se trata de un dispositivo de bajo coste que únicamente indica movimientos de "todo o nada" en los ejes: (-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0) y (1, 1). Además cuenta con 10 botones (B1 a B10) reconocibles por el driver genérico para joystick de Linux.

4.2.2. PROGRAMACIÓN DEL NODO ROS

El nodo ROS debe ser capaz de realizar las siguientes acciones

1. Escuchar el topic `/joy` para capturar comandos de movimiento.
2. Traducir los comandos de movimiento en comandos PTU-46.
3. Interrogar periódicamente la unidad PTU-46 para determinar su orientación (PAN/TILT).
4. Traducir las respuestas PTU-46 a mensajes *JointState*.
5. Publicar periódicamente mensajes *JointState* en el topic `/ptu_46_pos_out` con la orientación del PTU-46.

A continuación se muestra un extracto del código fuente de un nodo ROS, capaz de suscribirse a un topic y publicar en otro. A esta estructura hay que añadirle la funcionalidad de comunicación serie con la unidad PTU-46, la traducción de mensajes joystick a comandos PTU-46, y de respuestas PTU-46 a mensajes *JointState*.

El código fuente completo de una posible implementación de este ejercicio está disponible en el epígrafe 6.3 de los apéndices. Se pueden encontrar ejemplos de programación serie en el epígrafe 6.1 y en [4], así como documentación más exhaustiva en [5] y [6].

```

#include <iostream>
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "sensor_msgs/Joy.h"
// Grados a girar el eje del PTU en cada indicación del joystick
#define JOYSTEP PAN DEG 10.0
#define JOYSTEP TILT DEG 10.0

// Declaración de funciones callback
void joyCallback(const sensor_msgs::Joy& inJoyCommand);

int main(int argc, char** argv) {

    // Creación de mensaje JointState con dos componentes (PAN y TILT)
    thePtuJointState.name.resize(2); // array de tamaño 2: PAN y TILT
    thePtuJointState.position.resize(2); // array de tamaño 2: PAN y TILT
    thePtuJointState.name[0] = "PAN";
    thePtuJointState.name[1] = "TILT";

    // Rutinas de inicialización ROS
    ros::init(argc, argv, "pan_tilt_ptu_46");
    ros::NodeHandle theNodeHandle;
    myNodeHandle = &theNodeHandle;

    // Creación de suscriptor a topic /joy
    ros::Subscriber theJoySubscriber = theNodeHandle.subscribe("/joy", 10, joyCallback);

    // Creación de publicador a topic /ptu 46 pos out
    ros::Publisher thePublisher =
theNodeHandle.advertise<sensor_msgs::JointState>("/ptu_46_pos_out",10);

    ROS_INFO("Nodo de control PTU-46 iniciado");

    // Init PTU-46
    // Inicialización de comunicación serie
    // ...

    ros::Rate r(1); // 1 hz
    while (myNodeHandle->ok()) {

        // Determinar la posición PAN y TILT mediante comunicación serie con la unidad
        // y actualizar con ella el mensaje thePtuJointState
        // ...
        // Publicar la posición obtenida
        thePublisher.publish(thePtuJointState);

        ros::spinOnce();
    }

    cout << "Programa terminado" << endl;
    return 0;
}

void joyCallback(const sensor_msgs::Joy& inJoyCommand) {

    char thePanCommand[16];
    char theTiltCommand[16];

    // Extraer la información de ejes del mensaje ROS Joy
    float theAxesH = (float)inJoyCommand.axes[0];
    float theAxesV = (float)inJoyCommand.axes[1];

    printf("Recibido comando joystick (%g,%g)\n",theAxesV,theAxesH);

    // Interpretar los comandos de Joystick como instrucciones
    // para mover JOYSTEP PAN DEG (JOYSTEP PAN TILT) grados el eje correspondiente (PAN o TILT)
    en el sentido indicado

    if (theAxesH != 0.0) {
        // Enviar comando de posición PAN a unidad PTU ...
    }

    if (theAxesV != 0.0) {
        // Enviar comando de posición TILT a unidad PTU ...
    }
}

```

4.3. Teleoperación de plataforma móvil YouBot mediante teclado

Este ejercicio consiste en la programación de dos nodos ROS.

El primero genera mensajes ROS de velocidad a partir de entradas de teclado y los publica en un *rostopic* llamado *cmd_ref_vel*.

- Las teclas de flecha controlan la traslación de la base (avance, retroceso, desplazamientos laterales).
- Las teclas 'x' e 'y' controlan el giro de la base (horario o antihorario).
- Las teclas 's' detiene el movimiento del robot.

El segundo se suscribe al rostopic 'cmd_ref_vel' y transmite a la base del youbot los mensajes de velocidad publicados a la base del robot.

4.3.1. NODO VEL PUBLISHER

Este nodo se encarga de publicar en el topic '*cmd_ref_vel*' comandos de velocidad ante eventos de teclado. La pulsación de las flechas del teclado genera comandos de velocidad (*Twist*) con componente lineal. La pulsación de 'x' e 'y' generan comandos de velocidad con componente angular.

Tras la generación de un comando de velocidad, la base del YouBot permanece obedeciéndola indefinidamente, por lo que es necesario revocar la orden mediante un comando *Twist()* de velocidad lineal y angular 0. Este comando se envía cuando el usuario pulsa la letra 's'.

```

/*
 * vel_publisher.cpp
 * Autor: fsotosan
 * Programa para publicar mensajes Twist en el topic ROS 'cmd_ref_vel' usando las teclas de flecha, la 'x' y
 la 'y'
 * La lectura de teclado usando terminos se ha extraido de turtlebot teleop
 http://github.com/turtlebot/turtlebot_apps.git
 * La parte de suscripción a un topic está extraída de https://raw.github.com/ros/ros_tutorials/groovy-
 devel/roscpp_tutorials/talker/talker.cpp
 */

#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include <iostream>
#include <termios.h>

const double LINEAR_VEL = 0.1; // metros por segundo
const double PI = 3.1415926;
const double ANGULAR_VEL = PI/8; // radianes por segundo

#define KEYCODE_R 0x43
#define KEYCODE_L 0x44
#define KEYCODE_U 0x41
#define KEYCODE_D 0x42
#define KEYCODE_Q 0x71

using namespace std;
int kfd = 0;
struct termios cooked, raw;

int getVelInput(geometry_msgs::Twist *outVel);
void initConsole();

int main(int argc, char **argv) {

```

```

ros::init(argc, argv, "vel_publisher");
ros::NodeHandle theNodeHandle;
ros::Publisher thePublisher = theNodeHandle.advertise<geometry_msgs::Twist>("cmd_ref_vel",10); //
Publicaremos en el canal 'cmd_vel_ref'. Buffer de 100 mensajes
ros::Rate theLoopRate(5); // 5 Hz
geometry_msgs::Twist theVel;

initConsole();

cout << "Pulsa las flechas para avanzar. Pulsa 'x' o 'y' para girar" << endl;

while(ros::ok()) {
    if(getVelInput(&theVel)) {
        thePublisher.publish(theVel);
        // [Fernando 20130408] No publicar mensaje de parada inmediatamente
        // ya que se ha comprobado que el youbot no llega a moverse
        //thePublisher.publish(geometry_msgs::Twist());
    }
    theLoopRate.sleep();
}

cout << "Programa terminado" << endl;
tcsetattr(kfd, TCSANOW, &cooked);
return 0;
}

int getVelInput(geometry_msgs::Twist *outVel) {

    int c = 0;

    outVel->linear.x = 0.0;
    outVel->linear.y = 0.0;
    outVel->linear.z = 0.0;
    outVel->angular.x = 0.0;
    outVel->angular.y = 0.0;
    outVel->angular.z = 0.0;

    if(read(kfd, &c, 1) < 0) {
        perror("read():");
        exit(-1);
    }

    switch (c) {
        case KEYCODE_D:
            outVel->linear.x = -LINEAR_VEL;
            break;
        case KEYCODE_U:
            outVel->linear.x = LINEAR_VEL;
            break;
        case KEYCODE_L:
            outVel->linear.y = -LINEAR_VEL;
            break;
        case KEYCODE_R:
            outVel->linear.y = LINEAR_VEL;
            break;
        case 'y':
            outVel->angular.z = ANGULAR_VEL;
            break;
        case 'x':
            outVel->angular.z = -ANGULAR_VEL;
            break;
        case 's':
            // [Fernando 20130408] Al pulsar una 's' se envia un mensaje de parada
            break;
        default:
            return 0;
            break;
    }

    return 1;
}

void initConsole() {

    // Código extraído de turtlebot_teleop

    // get the console in raw mode
    tcsetattr(kfd, &cooked);
    memcpy(&raw, &cooked, sizeof(struct termios));
    raw.c_lflag &= ~(ICANON | ECHO);
    // Setting a new line, then end of file
    raw.c_cc[VEOL] = 1;
    raw.c_cc[VEOF] = 2;
    tcsetattr(kfd, TCSANOW, &raw);
}

```

4.3.2. NODO YUBOT_BASE_MOVE

Este nodo escucha los comandos de velocidad publicados en el topic *cmd_ref_vel* y los transforma en instrucciones de velocidad de la API YouBot.

Como protección ante un eventual fallo de comunicaciones tras el envío de un comando de velocidad, el nodo *youbot_base_move* ordenará la parada de los actuadores transcurrido un segundo desde la última orden recibida a través del topic.

```

/*
 * youbot base move.cpp
 * Programa para mover la base del youbot a partir de mensajes de velocidad publicados en el topic
 'cmd_ref_vel'
 * Autor: fsotosan
 * La parte de suscripción a un topic está extraída de https://raw.githubusercontent.com/ros/ros_tutorials/groovy-
 devel/roscpp_tutorials/listener/listener.cpp
 * La parte de movimiento del youbot está extraída de https://github.com/youbot/youbot_applications.git
 (hello_world_demo)
 * */

#include "youbot/YouBotBase.hpp"
#include "youbot/YouBotManipulator.hpp"
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"

using namespace youbot;
using namespace std;

void velCallback(const geometry_msgs::Twist::ConstPtr& inTwist);
void timerCallback(const ros::TimerEvent&);
void moveBase(float inLongV, float inTransV, float inAngV);
void stopBase();

int i = 0;
YouBotBase* myYouBotBase = 0;
bool myYouBotHasBase = false;
ros::Timer myTimer;
ros::NodeHandle* myNodeHandle = 0;

int main(int argc, char **argv) {

    ros::init(argc, argv, "youbot_base_move");
    ros::NodeHandle theNodeHandle;
    myNodeHandle = &theNodeHandle;
    ros::Subscriber theSubscriber = theNodeHandle.subscribe("cmd_ref_vel", 10, velCallback);

    ROS_INFO("Programa youbot_base_move iniciado");
    ROS_INFO("YUBOT CONFIGURATIONS DIR: %s", YUBOT_CONFIGURATIONS_DIR);

    try {
        myYouBotBase = new YouBotBase("youbot-base", YUBOT_CONFIGURATIONS_DIR);
        myYouBotBase->doJointCommutation();
        myYouBotHasBase = true;
    } catch (std::exception& e) {
        LOG(warning) << e.what();
        myYouBotHasBase = false;
    }

    ros::spin();

    cout << "Programa terminado" << endl;
    return 0;
}

void velCallback(const geometry_msgs::Twist::ConstPtr& inTwist) {

    // Paramos el temporizador para evitar la detención automática de la base

    myTimer.stop();

    // Indicamos que se ha recibido un mensaje de velocidad

    ROS_INFO("%d - Recibido mensaje Twist: vLineal = (%g,%g,%g), vAngular = (%g,%g,%g)\n", i++, inTwist-
>linear.x, inTwist->linear.y, inTwist->linear.z, inTwist->angular.x, inTwist->angular.y, inTwist->angular.z);

    // Ejecutamos la orden

```

```
    moveBase(inTwist->linear.x, inTwist->linear.y, inTwist->angular.z);

    // Iniciamos un temporizador para detener el robot si no se reciben nuevas órdenes en un segundo
    myTimer = myNodeHandle->createTimer(ros::Duration(1.0),timerCallback, true);
}

void timerCallback(const ros::TimerEvent&) {

    ROS_INFO("Deteniendo la base por timeout");
    stopBase();
}

void stopBase() {

    moveBase(0.0,0.0,0.0);
}

void moveBase(float inLongV, float inTransV, float inAngV) {

    /*
    * la api OODL utiliza unidades boost
    */
    quantity<si::velocity> longitudinalVelocity = inLongV * meter_per_second;
    quantity<si::velocity> transversalVelocity = inTransV * meter_per_second;
    quantity<si::angular velocity> angularVelocity = inAngV * radian per second;

    if (myYouBotHasBase) {
        myYouBotBase->setBaseVelocity(longitudinalVelocity, transversalVelocity, angularVelocity);
    } else {
        ROS_INFO("No hay base");
    }
}
}
```

4.4. Imitación de movimiento esquelético con brazo Youbot

En este ejercicio se utiliza el sensor de profundidad Asus Xtion para determinar la posición de muñecas, hombros, cadera y rodillas.

El reconocimiento esquelético a través de Asus Xtion se proporciona mediante las plataformas OpenNI y NITE, descritas ambas en el capítulo dedicado a este sensor. Además, es posible acceder desde ROS a esta funcionalidad, a través del paquete *openni_tracker*.

El paquete *openni_tracker* reconoce al individuo situado en el campo de visión de la cámara y publica la posición de sus articulaciones mediante *frames* en diferentes *topics* (*/head*, */neck*, */torso*, */left_shoulder*, */left_hand*, */left_hip*, */left_elbow*, */left_knee*, */left_foot*, etc.).

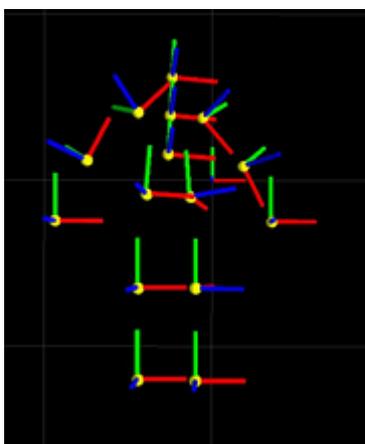


Figura 4.4 -Representación en rviz de los frames generados por *openni_tracker*

La orientación de los frames proporcionada por *openni_tracker* no resulta siempre fiable (realmente solo el *frame* correspondiente a la cabeza parece inclinarse imitando al individuo con cierta repetibilidad en las pruebas realizadas), por lo que en este ejercicio solamente utilizaremos la información de posición.

Así, conociendo las coordenadas cartesianas de muñeca, codo y hombro podemos determinar los vectores codo-muñeca y codo-hombro y obtener el ángulo formado por muñeca, codo y hombro a partir de su producto escalar, con independencia de la orientación del individuo.

Conocidos los ángulos formados por diferentes ternas de articulaciones, podemos asignarlos uno a cada unión del brazo del manipulador Kuka Youbot.

4.4.1. CONFIGURACIÓN ROS

Además de nuestro propio *software*, necesitaremos el paquete *openni_tracker* para la publicación de la posición de las articulaciones.

```
sudo apt-get install ros-fuerte-openni-tracker
```

El ejercicio consta de dos ejecutables: uno para leer la información publicada por *openni_tracker* y publicar mensajes de movimiento para el brazo del Youbot, y otro que leerá estos mensajes de movimiento y los convertirá en comandos específicos de la API OODL de youbot. Esta división permitirá además instalar opcionalmente la cámara Xtion en un PC distinto al del propio Youbot.

Crearemos pues un paquete ROS con dos archivos fuente (uno por cada ejecutable) y declaramos como dependencias los paquetes *tf* y *youbot_driver*.

```
roscreeate-pkg skeleton2youbot tf youbot driver
```

Aprovecharemos este ejercicio para crear nuestro propio mensaje ROS, con el que enviar la pose de 5 articulaciones simultáneamente (realmente es innecesario puesto que la integración ROS de Youbot ya proporciona uno, pero se incluye aquí con fines ilustrativos). Para ello crearemos una carpeta msg y, dentro de ella, un archivo *YouBotManipulatorJointAngles.msg*:

```
cd skeleton2youbot
mkdir src
vi YouBoyManipulatorJointAngles.cpp
```

con el siguiente contenido:

```
Header header
float32 A1
float32 A2
float32 A3
float32 A4
float32 A5
```

En este caso, la configuración de compilación en *CMakeLists.txt* difiere con respecto a la creada por defecto por *roscreeate-pkg* y usada en otros ejercicios. Se han utilizado partes de los *CMakeLists.txt* disponibles en los ejemplos proporcionados con YouBot. Además, al definir un mensaje ROS en este paquete, se ha utilizado el comando *rosbuild_genmsgs()*.

El contenido del fichero *CMakeLists.txt* es el siguiente:

```

PROJECT(youbot_manipulator_move)
cmake_minimum_required(VERSION 2.8)

## check required prerequisites
SET(CMAKE_MODULE_PATH "${PROJECT_SOURCE_DIR}/external/cmake_modules")
SET(Boost_USE_STATIC_LIBS ON)
SET(Boost_ADDITIONAL_VERSIONS "1.39" "1.39.0" "1.40" "1.40.0" "1.41" "1.41.0" "1.42" "1.42.0" "1.43" "1.43.0"
"1.44" "1.44.0" "1.45" "1.45.0" "1.46" "1.46.0")
FIND_PACKAGE(Boost REQUIRED COMPONENTS thread date_time filesystem system)

OPTION(USE_ROS "Enable ROS as compile tool" ON)

SET(ROS_ROOT_PATH $ENV{ROS_ROOT})
IF(DEFINED ROS_ROOT_PATH AND USE_ROS MATCHES ON)
    include($ENV{ROS_ROOT}/core/rosbuild/rosbuild.cmake)
    rosbuild_init()
    rosbuild_find_ros_package(youbot_driver)
    SET(OODL_YOUBOT_LIBRARIES YouBotDriver)
    SET(OODL_YOUBOT_CONFIG_DIR ${youbot_driver_PACKAGE_PATH}/config/)
ELSE(DEFINED ROS_ROOT_PATH AND USE_ROS MATCHES ON)
    FIND_PACKAGE(OODL_YouBot REQUIRED)
ENDIF(DEFINED ROS_ROOT_PATH AND USE_ROS MATCHES ON)

## preprocessor definitions
ADD_DEFINITIONS(-DYOUBOT_CONFIGURATIONS_DIR="${OODL_YOUBOT_CONFIG_DIR}/")

## build parameters and paths
SET(CMAKE_BUILD_TYPE DEBUG) #enable debug mode (e.g. for embedded gdb in eclipse )
SET(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin/ CACHE PATH "Configure the executable output path.")
SET(LIBRARY_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/lib/ CACHE PATH "Configure the library output path.")

## Set include directories
INCLUDE_DIRECTORIES(
    ${OODL_YOUBOT_INCLUDE_DIR}
    ${Boost_INCLUDE_DIR}
)

#uncomment if you have defined messages
rosbuild_genmsg()
#uncomment if you have defined services
#rosbuild_gensrv()

rosbuild_add_executable(skeletonframes_to_youbotangles src/skeletonframes_to_youbotangles.cpp)
rosbuild_add_executable(youbot_arm_move src/youbot_arm_move.cpp)

```

4.4.2. PROGRAMA DE LECTURA DE POSICIÓN ESQUELETAL Y OBTENCIÓN DE ÁNGULOS

El programa *skeletonframes_to_youbotangles* lee los *frames* de las articulaciones de los topics correspondientes. Todos estos frames están referidos a una misma referencia común. Pero a nosotros nos interesa más la posición relativa entre componentes de una misma terna. Para ello utilizamos el comando *lookupTransform*. que proporciona pose de un *frame* en coordenadas del otro (transformaciones):

```
theListener.lookupTransform("/right_elbow_1", "/right_shoulder_1", theTime, theTransformElbowShoulder);
theListener.lookupTransform("/right_elbow_1", "/right_hand_1", theTime, theTransformElbowHand);
```

A partir de las transformaciones, obtendremos los vectores deseados mediante la función *getOrigin()*.

```
tf::Vector3 theVectorElbowHand = theTransformElbowHand.getOrigin();
tf::Vector3 theVectorElbowShoulder = theTransformElbowShoulder.getOrigin();
```

Conviene indicar que para obtener el ángulo buscado con el signo correcto operaremos con vectores que parten del mismo punto, por lo que en ocasiones necesitaremos invertir el sentido.

```
tf::Vector3 theVectorShoulderElbow = -1 * theVectorElbowShoulder;
```

Una vez obtenidos todos los vectores de interés, calcularemos los ángulos mediante la función *getAngle()* que hemos programado para calcular el ángulo formado por dos vectores conocidos aplicando la definición de producto escalar.

```
tfScalar theAngleHandElbowShoulder = getAngle(&theVectorElbowHand, &theVectorElbowShoulder);
```

Para más detalles se lista el código fuente completo a continuación:

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <skeleton2youbot/YouBotManipulatorJointAngles.h>
#define PI 3.1415927

tfScalar getScalarProduct(const tf::Vector3* v1, const tf::Vector3* v2) {
    return v1->x() * v2->x() + v1->y() * v2->y() + v1->z() * v2->z();
}

tfScalar getMagnitude(const tf::Vector3* v1) {
    return sqrt(getScalarProduct(v1, v1));
}

tfScalar getAngle(const tf::Vector3* inVector1, const tf::Vector3* inVector2) {
    tfScalar theScalarProd = getScalarProduct(inVector1, inVector2);
    tfScalar theM1 = getMagnitude(inVector1);
```

```

tfScalar theM2 = getMagnitude(inVector2);

if ((theM1 > 0) && (theM2 > 0)) {
    tfScalar theCosTheta = theScalarProd / (theM1 * theM2);
    tfScalar theTheta = acos(theCosTheta);
    return theTheta;
}
return 0;
}
}

int main(int argc, char** argv){
    ros::init(argc, argv, "skeleton2youbot");
    ros::NodeHandle theNodeHandle;
    tf::TransformListener theListener;
    ros::Publisher thePublisher =
theNodeHandle.advertise<skeleton2youbot::YouBotManipulatorJointAngles>("cmd_ref_pos",10); // Publicaremos en
el canal 'cmd_pos_ref'. Buffer de 10 mensajes

    while (theNodeHandle.ok()){
        tf::StampedTransform theTransformElbowShoulder,
theTransformElbowHand,
theTransformShoulderHip,
theTransformHipKnee,
theTransformFootKnee;

        ros::Rate theRate(5.0);
        try{
            //ros::Time theTime = ros::Time::now();
            ros::Time theTime = ros::Time(0);
            theListener.waitForTransform("/right_elbow_1", "/right_shoulder_1",
theTime,ros::Duration(3.0));
            theListener.lookupTransform("/right_elbow_1", "/right_shoulder_1", theTime,
theTransformElbowShoulder);
            theListener.lookupTransform("/right_elbow_1", "/right_hand_1", theTime,
theTransformElbowHand);
            //theListener.lookupTransform("/right_shoulder_1", "/right_hip_1", theTime,
theTransformShoulderHip);
            theListener.lookupTransform("/torso_1", "/right_hip_1", theTime,
theTransformShoulderHip);
            theListener.lookupTransform("/right_hip_1", "/right_knee_1", theTime,
theTransformHipKnee);
            theListener.lookupTransform("/right_foot_1", "/right_knee_1", theTime,
theTransformFootKnee);

            tf::Vector3 theVectorElbowHand = theTransformElbowHand.getOrigin();
            tf::Vector3 theVectorElbowShoulder = theTransformElbowShoulder.getOrigin();
            tf::Vector3 theVectorShoulderElbow = -1 * theVectorElbowShoulder;
            tf::Vector3 theVectorShoulderHip = theTransformShoulderHip.getOrigin();
            tf::Vector3 theVectorHipShoulder = -1 * theVectorShoulderHip;
            tf::Vector3 theVectorHipKnee = theTransformHipKnee.getOrigin();
            tf::Vector3 theVectorKneeHip = -1 * theVectorHipKnee;
            tf::Vector3 theVectorFootKnee = theTransformFootKnee.getOrigin();
            tf::Vector3 theVectorKneeFoot = -1 * theVectorFootKnee;

            tfScalar theAngleHandElbowShoulder =
getAngle(&theVectorElbowHand,&theVectorElbowShoulder);
            tfScalar theAngleElbowShoulderHip =
getAngle(&theVectorShoulderElbow,&theVectorShoulderHip);
            tfScalar theAngleShoulderHipKnee =
getAngle(&theVectorHipShoulder,&theVectorHipKnee);
            tfScalar theAngleHipKneeFoot = getAngle(&theVectorKneeHip,&theVectorKneeFoot);

```

```
        printf("HES: %g, ESH: %g, SHK: %g, HKF: %g\n",theAngleHandElbowShoulder * 180 / PI
,
theAngleElbowShoulderHip * 180 / PI
,
theAngleShoulderHipKnee * 180 / PI
,
theAngleHipKneeFoot * 180 / PI);

        skeleton2youbot::YouBotManipulatorJointAngles theMsg;
        theMsg.A1 = theAngleHandElbowShoulder;
        theMsg.A2 = theAngleElbowShoulderHip;
        theMsg.A3 = theAngleShoulderHipKnee;
        theMsg.A4 = theAngleHipKneeFoot;
        thePublisher.publish(theMsg);
    } catch (tf::TransformException ex) {
        ROS_ERROR("%s",ex.what());
    }
    theRate.sleep();
}
return 0;
};
```

4.4.3. PROGRAMA DE MOVIMIENTO DE BRAZO

El programa *youbot_arm_move* se suscribe al *topic cmd_ref_pos* creado por el programa anterior, con las posiciones de las 5 articulaciones del brazo. Tras ejecutar algunas comprobaciones de límites por seguridad de las uniones, comunica a los actuadores correspondientes las posiciones solicitadas.

A continuación se muestra el código fuente completo del programa

```
#include <iostream>
#include "youbot/YouBotBase.hpp"
#include "youbot/YouBotManipulator.hpp"
#include "ros/ros.h"
#include <skeleton2youbot/YouBotManipulatorJointAngles.h>

using namespace youbot;
using namespace std;

#define PI 3.1415927

#define A1_FOLD_RADIANS      0.100692
#define A1_UNFOLD_RADIANS   5.84014
#define A2_FOLD_RADIANS     0.0100692
#define A2_UNFOLD_RADIANS   2.617
#define A3_FOLD_RADIANS     -5.02655
#define A3_UNFOLD_RADIANS   -0.015708
#define A4_FOLD_RADIANS     0.221239
#define A4_UNFOLD_RADIANS   3.4292
#define A5_FOLD_RADIANS     0.11062
#define A5_UNFOLD_RADIANS   5.64159

void posCallback(const skeleton2youbot::YouBotManipulatorJointAngles::ConstPtr& inAngles);

int i = 0;
YouBotManipulator* myYouBotArm = 0;
bool myYouBotHasArm = false;
ros::NodeHandle* myNodeHandle = 0;

int main(int argc, char** argv) {

    ros::init(argc, argv, "youbot_arm_move");
    ros::NodeHandle theNodeHandle;
    myNodeHandle = &theNodeHandle;
    ros::Subscriber theSubscriber = theNodeHandle.subscribe("cmd_ref_pos", 10, posCallback);

    ROS_INFO("Programa youbot_arm_move iniciado");
    ROS_INFO("YOUBOT_CONFIGURATIONS_DIR: %s", YOUBOT_CONFIGURATIONS_DIR);

    try {
        myYouBotArm = new YouBotManipulator("youbot-manipulator", YOUBOT_CONFIGURATIONS_DIR);
        myYouBotArm->doJointCommutation();
        myYouBotArm->calibrateManipulator();
        myYouBotHasArm = true;
    } catch (std::exception& e) {
        LOG(warning) << e.what();
    }
}
```

```
        myYouBotHasArm = false;
    }
    ros::spin();
    cout << "Programa terminado" << endl;
    return 0;
}

double translateRange(float inInput, float inMinInput, float inMaxInput, float inMinOutput, float
inMaxOutput) {

    if (inInput < inMinInput) {
        inInput = inMinInput;
    } else if (inInput > inMaxInput) {
        inInput = inMaxInput;
    }
    return inMinOutput + (inInput - inMinInput)*(inMaxOutput - inMinOutput) / (inMaxInput - inMinInput);
}

void posCallback(const skeleton2youbot::YouBotManipulatorJointAngles::ConstPtr& inAngles) {

    JointAngleSetpoint theJointAngle;
    theJointAngle = translateRange(inAngles->A1, 0, PI, A1_FOLD_RADIANS, A1_UNFOLD_RADIANS) * radian;
    myYouBotArm->getArmJoint(1).setData(theJointAngle);
    theJointAngle = translateRange(inAngles->A2, 0, PI, A2_FOLD_RADIANS, A2_UNFOLD_RADIANS) * radian;
    myYouBotArm->getArmJoint(2).setData(theJointAngle);
    theJointAngle = translateRange(inAngles->A3, 0, PI, A3_FOLD_RADIANS, A3_UNFOLD_RADIANS) * radian;
    myYouBotArm->getArmJoint(3).setData(theJointAngle);
    theJointAngle = translateRange(inAngles->A4, 0, PI, A4_FOLD_RADIANS, A4_UNFOLD_RADIANS) * radian;
    myYouBotArm->getArmJoint(4).setData(theJointAngle);
    theJointAngle = translateRange(inAngles->A5, 0, PI, A5_FOLD_RADIANS, A5_UNFOLD_RADIANS) * radian;
    myYouBotArm->getArmJoint(5).setData(theJointAngle);

    // Indicamos que se ha recibido un mensaje de posición

    ROS_INFO("%d - Recibido mensaje YouBotManipulatorJointAngles: A = (%g,%g,%g,%g,%g), \n", i++,
inAngles->A1, inAngles->A2, inAngles->A3, inAngles->A4, inAngles->A5);
}
}
```

4.5. Seguimiento de objetos con webcam, OpenCV y actuadores AX-12A

Este ejercicio consiste en combinar dos actuadores Dynamixel AX-12A formando una unidad *pan-tilt* a la que se acoplará una cámara USB. La comunicación con los actuadores se realizará a través de ROS.

El objetivo es orientar la cámara automáticamente hacia un objeto identificado por su color. La posición de la pelota se determinará mediante procesamiento de imagen utilizando OpenCV.

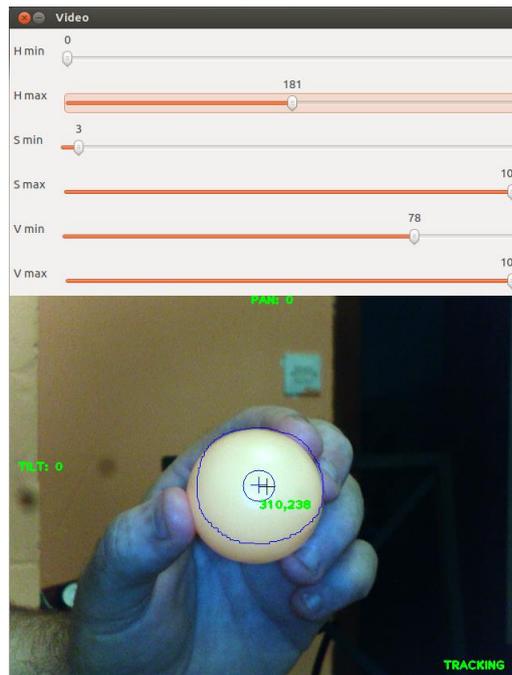


Figura 4.5 - Captura de pantalla con aplicación de seguimiento

El color del objeto se podrá seleccionar mediante una interfaz de usuario consistente en 3 pares de barras deslizantes que establecerán los valores máximo y mínimo de los parámetros HSV (*Hue, Saturation and Value*).

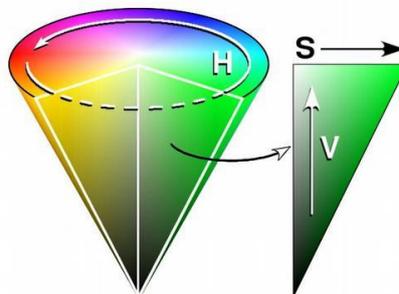


Figura 4.6 - Representación de color mediante HSV

La acción de seguimiento se podrá habilitar o deshabilitar pulsando la letra 't' (por particularidades de la instalación de opencv a través de paquetes ROS no es sencillo utilizar controles que requieran enlace a librerías Qt).

4.5.1. MONTAJE DE ACTUADORES Y SENSOR

Para simplificar las operaciones de seguimiento los ejes de rotación *pan* y *tilt* se han hecho coincidir con el sensor de imagen. Para ello ha sido necesario desmontar la cámara USB y configurar el movimiento con ayuda de algunas piezas de meccano.

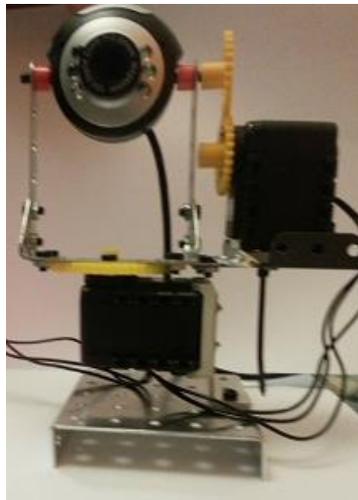


Figura 4.7 - Montaje pan-tilt para cámara USB

4.5.2. CREACIÓN DEL PAQUETE ROS

En la creación del paquete ROS se incluirán como dependencias el paquete *dynamixel_controllers* para los actuadores, el paquete *uvc_cam* para la interacción con la cámara USB y los paquetes *cv_bridge*, *opencv2*, *sensor_msgs*, *std_msgs*, e *image_transport* para el procesamiento de imágenes y *roscpp*, ya que programaremos en C++.

```
roscmake pkg pan_tilt_cvtrack dynamixel_controllers cv_bridge opencv2 sensor_msgs std_msgs
image_transport uvc_cam roscpp
export PATH=~/.fuerte/workspace/pan_tilt_cvtrack:$PATH
rosmake pan_tilt_cvtrack
```

El código fuente completo de este ejercicio está disponible en el epígrafe **6.6** de los apéndices.

4.5.3. CONFIGURACIÓN DE LOS ACTUADORES DYNAMIXEL

En primer lugar debemos asegurarnos de que los actuadores tienen identificadores distintos, ya que por defecto tienen asignado el ID 1. Se puede modificar el ID utilizando la utilidad *DynamixelWizard*, incluido en el software *RoboPlus* (windows) o bien desde ROS mediante el paquete *Arbotix*. En este caso se ha optado por el primer método.

Como ya se indicó en el capítulo de actuadores *dynamixel* necesitamos varios archivos: administrador de controladores (*pan_tilt_cm.launch*), archivo de configuración (*pan_tilt.yaml*) y archivo de inicio de controladores (*pan_tilt_sc.launch*). En este caso en el *pan_tilt_cm.launch* incluiremos dos actuadores con IDs 1 y 2.

```
<launch>
  <node name="dynamixel manager" pkg="dynamixel_controllers" type="controller_manager.py"
required="true" output="screen">
    <rosparam>
      namespace: dxl_manager
      serial_ports:
        pan tilt port:
          port name: "/dev/ttyACM0"
          baud_rate: 1000000
          min_motor_id: 1
          max_motor_id: 2
          update rate: 20
    </rosparam>
  </node>
</launch>
```

Ahora el archivo de configuración hará referencia a dos uniones: *pan_joint* y *tilt_joint*.
Contenido de archivo de configuración *pan_tilt.yaml*

```
joints: ['pan joint', 'tilt joint']
pan joint:
  controller:
    package: dynamixel_controllers
    module: joint position controller
    type: JointPositionController
  joint name: pan joint
  joint speed: 1.17
  motor:
    id: 1
    init: 512
    min: 0
    max: 1023
tilt_joint:
  controller:
    package: dynamixel_controllers
    module: joint position controller
    type: JointPositionController
  joint_name: tilt_joint
  joint_speed: 1.17
  motor:
    id: 2
    init: 512
    min: 0
    max: 1023
```

Finalmente el lanzador de controladores hace referencia a las dos uniones definidas en el archivo de configuración. Contenido de *pan_tilt_sc.launch*:

```
<launch>
  <!-- Start joint controller -->
  <rosparam file="$(find test_dynamixel)/pan_tilt.yaml" command="load"/>
  <node name="pan_tilt_controller_spawner" pkg="dynamixel_controllers"
type="controller_spawner.py"
    args="--manager=dxl manager
        --port pan tilt port
        --type=simple
        pan_joint
        tilt_joint"
    output="screen"/>
</launch>
```

Los actuadores son ahora accesibles desde ROS. Para llevar a la posición cero los actuadores:

```
rostopic pub -1 /pan joint/command std_msgs/Float64 -- 0.0
rostopic pub -1 /tilt_joint/command std_msgs/Float64 -- 0.0
```

4.5.4. CONFIGURACIÓN DE LA CÁMARA USB

El procedimiento de instalación de la cámara en linux y ROS se describe en el epígrafe **2.3**.

Para publicar los *topics* de imagen simplemente ejecutamos el siguiente comando

```
roslaunch uvc_cam test_uvc.launch
```

4.5.5. DETECCIÓN DE OBJETOS POR SU COLOR MEDIANTE OPENCV

A continuación se detalla el procedimiento para detectar el centro de un objeto de un color determinado utilizando OpenCV.

6. **Obtención de imagen umbral:** Para discriminar los elementos del color deseado del resto la imagen se filtra obteniendo una imagen umbral, donde para cada pixel se indica si pertenece o no pertenece al rango de colores válido. Esto se consigue con la función OpenCV *inRange*. Para delimitar el rango de colores deseado se utiliza representación HSV (Hue, Saturation, Value) en vez de la representación BGR (Blue, Green, Red) por defecto en OpenCV. El motivo es que el formato HSV distingue entre intensidad (*luma*) e información de color (*chroma*), lo que añade robustez frente a diferencias de iluminación. La traducción a HSV se realiza mediante la función *cvtColor*.
7. **Eliminación de ruido:** La imagen umbral obtenida mediante filtro de color contendrá seguramente discontinuidades. Llamando unos a los píxeles del color correcto, y ceros al resto, las zonas correspondientes al objeto detectado tendrán mayor densidad de unos, mientras que el resto de zonas tendrán mayor densidad de ceros, pero en ambos casos habrá cierto número de elementos minoritarios. Para eliminarlos se utiliza una combinación de operaciones morfológicas: **erosión** y **dilatación**. Ambas implican una convolución de la imagen con un *kernel* o patrón de píxeles. La primera reemplaza cada pixel de la imagen con el máximo local obtenido de la convolución. La segunda lo hace con el mínimo. El efecto de la erosión es eliminar los unos de las zonas de ceros. El efecto de la dilatación es eliminar los ceros de las zonas de unos. La combinación de ambas elimina ambigüedad. Las funciones OpenCV que llevan a cabo estas operaciones son *erode* y *dilate*.
8. **Obtención de contornos y momentos:** A partir del mapa de unos y ceros, la función OpenCV *findContours* es capaz de identificar los contornos de las zonas de detección. Para cada contorno pueden determinarse ciertas propiedades como el área o las coordenadas del centroide mediante sus momentos. La función para obtener los momentos de un contorno es *moments*.

4.5.6. ORIENTACIÓN DE LA CÁMARA PARA CENTRAR EL OBJETIVO

En este punto se asume que la cámara web ha sido previamente calibrada. El procedimiento de calibración se describe en el capítulo **Cámara USB genérica compatible linux**. Tal como se indica en este capítulo, la calibración proporciona una matriz de parámetros intrínsecos y un vector de parámetros de distorsión.

Se utilizarán los parámetros intrínsecos (f_x , f_y , c_x , c_y) para obtener los ángulos de corrección *pan* y *tilt*. Las unidades en las que se proporcionan estos parámetros son *pixels*.

Los parámetros de distorsión se utilizarán opcionalmente para corregir la imagen antes de realizar los cálculos de orientación.

Teniendo en cuenta que el origen de coordenadas de la imagen proporcionada por la cámara USB está situado en la esquina superior izquierda, y que la resolución de la cámara es de 640x480 *pixels*, las coordenadas de la esquina inferior derecha de la imagen serán $x=640$ e $y=480$ respectivamente. De modo que la estrategia de seguimiento será mantener el objetivo en el centro de la imagen ($x=320$, $y=240$). No obstante, si los parámetros intrínsecos c_x , c_y no coinciden con este centro teórico, se podrán utilizar estos. El centro seleccionado se representará mediante una cruz de color negro.

El ángulo de corrección *pan* se obtendrá como la arcotangente del cociente entre la distancia del blanco al centro de la imagen y la distancia focal.

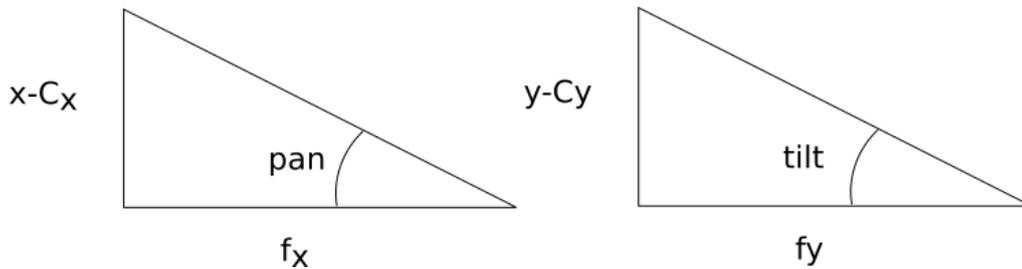


Figura 4.8 - Determinación de ángulos de corrección pan-tilt

$$tg(pan) = \frac{x - c_x}{f_x} \tag{4.9}$$

$$tg(tilt) = \frac{y - c_{yx}}{f_y} \tag{4.10}$$

Conviene indicar que el ángulo de corrección determinado de este modo ha de entenderse relativo a la posición de los motores en cada momento, no se trata de una posición absoluta.

4.6. Teleoperación de plataforma móvil Arduino mediante Joystick

El objetivo de este ejercicio es introducir brevemente la plataforma Arduino para controlar una plataforma móvil de bajo coste.

El ejercicio consiste en las siguientes tareas:

- Generar comandos de velocidad ROS (Twist) a partir de movimientos de un joystick
- Traducir comandos de velocidad a velocidades lineales de cada rueda.
- Comunicar la red ROS via serie con una plataforma **Arduino Uno**.
- Aplicar los comandos de velocidad a cada rueda mediante entrada/salida (PWM, GPIO) y un shield Arduino de control de motores dotado de drivers L293D o L298N.

4.6.1. DESCRIPCIÓN DE LA PLATAFORMA MÓVIL

La plataforma está formada por dos conjuntos de motor DC + reducción + rueda, una rueda loca y una superficie sobre la que ensamblar estos componentes.

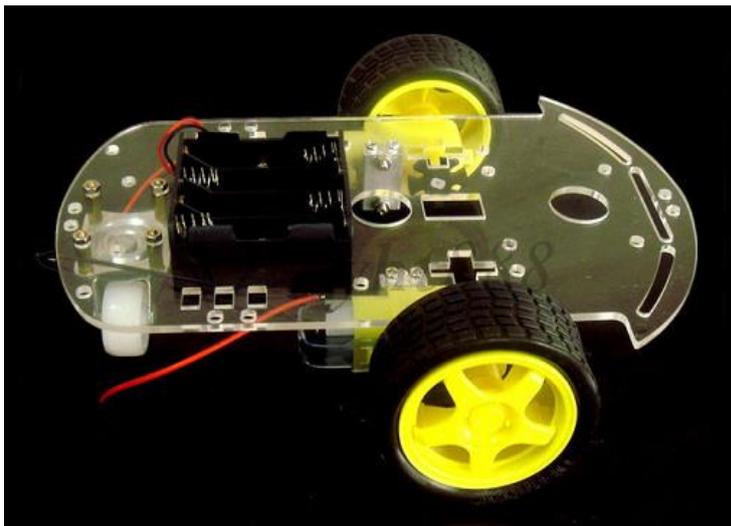


Figura 4.9 - Plataforma móvil con motores DC y reducción

Los motores funcionan entre 3 y 6V. En principio no consumen más de 150 mA en las condiciones de trabajo y el conjunto puede alcanzar una velocidad lineal de 47 metros/minuto sin carga.

Working Voltage	Parameters	DC 3V	DC 5V	DC 6V
Motor Parameters (No Gear Box)	RPM	125R / minute		
	Current	80-100mA		
	Reduction	48:1		
Gear Box Parameters	No-load Speed	125R/minute	200R/minute	230R/minute
	Load Speed	95R/minute	1600R/minute	175R/minute
	Output Torque	0.8kg.cm	1.0kg.cm	1.1kg.cm
	No-load car Speed	25.9meter/ minute	41.4meter/mi nute	47.7meter/mi nute
	Current	110-130mA	120-140mA	130-150mA
	Max wheel diameter	6.5cm		
	Dimensions	70mmx22mmx18mm		
	Weight	50g		
	Noise	<65dB		

Figura 4.10 - Especificaciones motores DC y reducción

Las dimensiones del conjunto motor dc + reducción se indican a continuación:

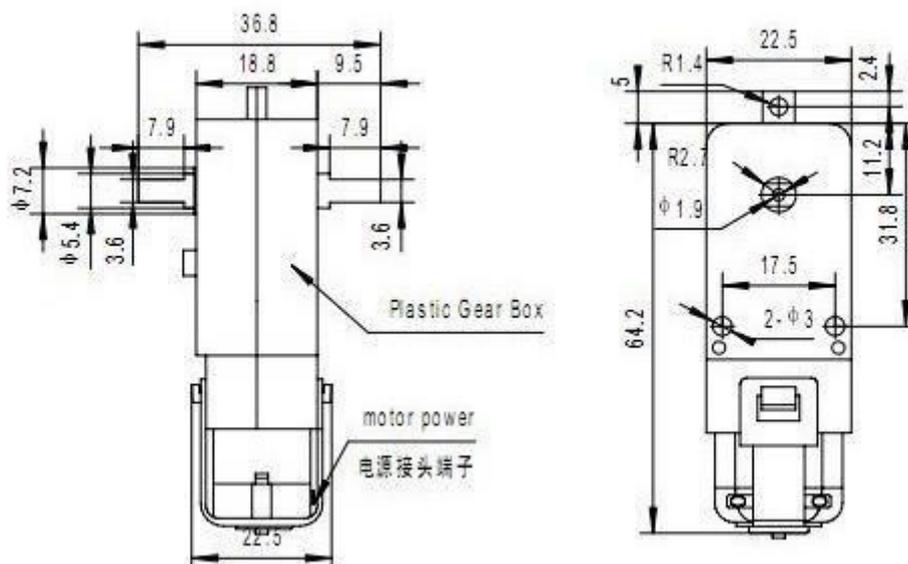


Figura 4.11 - Dimensiones motores DC y reducción

4.6.2. CONTROL DE VELOCIDAD EN LAZO ABIERTO USANDO TRACCIÓN DIFERENCIAL

La plataforma móvil consta de dos ruedas paralelas de tracción independientes alineadas sobre un eje en la parte delantera, y una rueda loca en la parte posterior. Con esta disposición, el modo de cambiar de dirección es aplicar velocidades distintas a las ruedas de tracción.

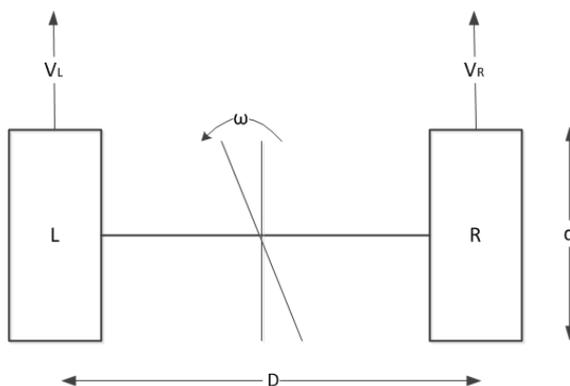


Figura 4.12 - Giro mediante diferencia de velocidades

Para traducir una orden de velocidad lineal y otra de velocidad angular a velocidades lineales de cada rueda:

- Consideramos la velocidad lineal como la media de las velocidades lineales de las dos ruedas

$$\mathbf{V} = (\mathbf{V}_R + \mathbf{V}_L) / 2 \quad (4.11)$$

- La componentes lineal y angular se relacionan mediante el radio de giro. En este caso $r = D/2$ siendo D la distancia entre las ruedas, considerando que el eje de giro pasa por el punto medio del eje de tracción.

$$\mathbf{V}_R = \omega \cdot r = \omega \cdot D / 2 \quad (4.12)$$

- De modo que para cumplir simultáneamente una orden de velocidad lineal V y una orden de velocidad angular ω tenemos dos ecuaciones con dos incógnitas cuya solución es:

$$\mathbf{V}_R = \mathbf{V} + \omega \cdot D / 2 \quad (4.13)$$

$$\mathbf{V}_L = \mathbf{V} - \omega \cdot D / 2 \quad (4.14)$$

- Conocidas las velocidades lineales resultantes para cada rueda, obtenemos las velocidades angulares en rpm en función del diámetro d de las ruedas:

$$\omega_R = V_R \cdot 60 / (\pi \cdot d) \quad (4.15)$$

$$\omega_L = V_L \cdot 60 / (\pi \cdot d) \quad (4.16)$$

4.6.3. LA PLATAFORMA ARDUINO

Un sistema Arduino consiste en:

- Una placa de *hardware* libre que incorpora un microcontrolador reprogramable y una serie de pines hembra para dar acceso a los puertos de entrada/salida del micro, de modo que resulte sencillo conectar sensores y actuadores.
- Un software gratuito, libre y multiplataforma (Linux, Mac, Windows) para programar el microcontrolador.
- Un lenguaje de programación libre.

Entre las ventajas de Arduino, cabe destacar la numerosa comunidad de usuarios que comparte ideas y enriquece la documentación. Además las placas Arduino son baratas, reutilizables para diferentes proyectos y versátiles.

4.6.4. EL LENGUAJE ARDUINO

El lenguaje Arduino tiene la misma sintaxis que C en la declaración de variables, funciones, inclusión de librerías, estructuras de control (if, while, etc.). Un programa Arduino sencillo normalmente consta de dos funciones: *setup()* y *loop()*. La primera debe contener las acciones de inicialización a realizar al iniciarse el programa (al conectar la alimentación del micro o tras un *reset*), como puede ser la configuración del sentido de los pines de entrada/salida digital, la velocidad de comunicación mediante UART, o la inicialización de variables. Mientras que la segunda constituye el bucle principal que normalmente programaríamos dentro de la función *main()* si se tratase de un programa C estándar para microcontrolador.

Ejemplo:

```
#include <AFMotor.h>

#define BYTE INICIO DE TRAMA 'A'
#define BYTE INICIO DE TRAMA 'Z'
...
void setup() {
  Serial.begin(9600);
  ...
}

void loop() {

  char byteLeido;

  if (Serial.available() ) {
    byteLeido=Serial.read();
    ...
  }
}
```

4.6.5. EL GESTOR DE ARRANQUE

La facilidad de programación de los sistemas Arduino se deben en parte a que el microcontrolador viene con un *firmware* preprogramado en la memoria flash conocido como *bootloader*. Por eso si se reemplaza el microcontrolador por otro sin *bootloader* es necesario introducirlo mediante un programador ISP para que el sistema Arduino sea totalmente funcional y pueda volver a programarse mediante cable USB.

4.6.6. LA PLATAFORMA ARDUINO UNO

La placa Arduino UNO incorpora un microcontrolador Atmel de la familia ATmega328, con 14 entradas/salidas digitales (6 utilizables para PWM), 6 entradas analógicas, un oscilador cerámico de 16MHz y conexiones USB e ICSP para programación y depuración.

Conviene indicar aunque la tensión de alimentación nominal es de 5V, si se alimenta con baterías se recomienda un voltaje de salida ofrecido de 9 a 12 V (el regulador presente en la placa soportaría hasta 20V) y la intensidad de corriente debe ser como mínimo de 250mA.

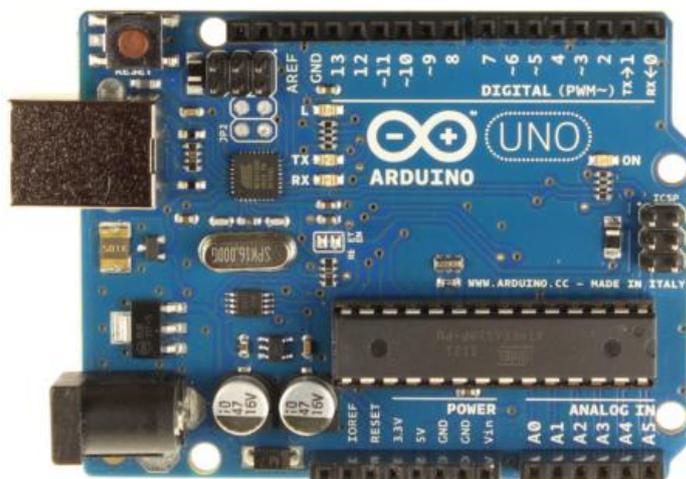


Figura 4.13 - Plataforma Arduino UNO

4.6.7. MOTOR SHIELD DE ADAFRUIT

El *Motor Shield* de Adafruit para Arduino permite controlar 2 servos y 4 motores DC bidireccionales (o 2 motores paso a paso, tanto unipolares como bipolares). El L293D proporciona 0.6mA por puente. Permite obtener la alimentación de la placa Arduino o utilizar una alimentación independiente para los motores (selección mediante *jumper*).



Figura 4.14 - Motor Shield de Adafruit

El circuito utilizado para controlar la velocidad del motor es un medio-puente en H. El caso A se produce con el transistor "Drive" en conducción y el "Flywheel" en corte. Y el caso B en la situación opuesta. El transistor "Flywheel" es necesario para evitar un corte brusco de corriente entre las bornas del motor, ya que este tiene bobinas que se oponen al cambio de corriente pudiendo generar arcos voltaicos y dañar el transistor "Drive" ($V_L = di/dt$ elevado).

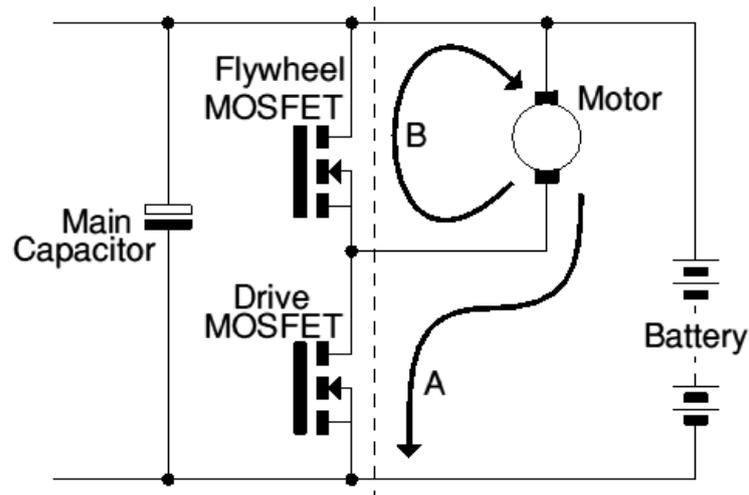


Figura 4.15 - Control de motor mediante medio puente en H

Concretamente, en el montaje mediante L293D utilizado en el shield, el montaje es el siguiente (lado izquierdo). La señal PWM que modula la velocidad se aplica en el pin 1, mientras que en los pines 2 y 7 se aplican tensiones fijas (y opuestas) para determinar el sentido de giro. A mayor duty de PWM, mayor velocidad.

En el integrado L293D los diodos de protección vienen incluidos en el propio chip.

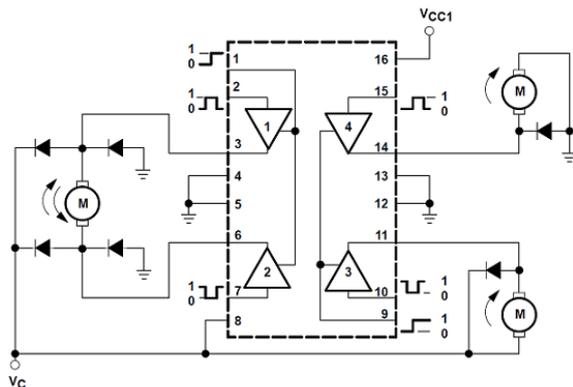


Figura 4.16 - Configuración típica control motor con L293

4.6.8. MOTOR SHIELD OFICIAL DE LA WEB DE ARDUINO

El Motor Shield oficial de Arduino incorpora el integrado L298, de funcionalidad similar a la del L293 descrito anteriormente pero con capacidad de entregar hasta 2A de corriente por canal.

Permite controlar 2 motores DC o un motor paso a paso.

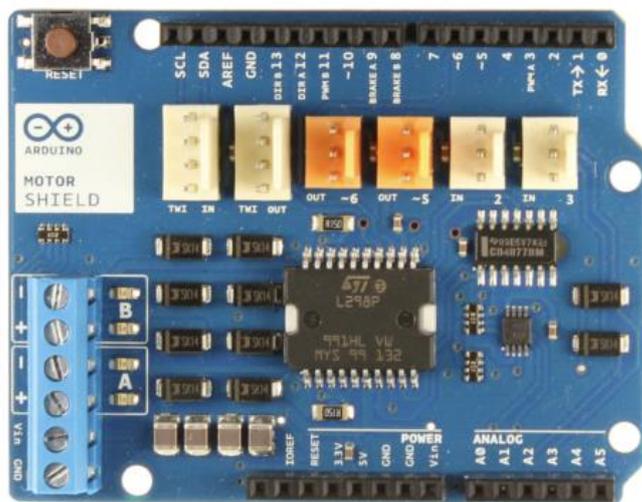


Figura 4.17 - Motor Shield oficial Arduino

4.6.9. COMUNICACIÓN ENTRE NODO ROS Y ARDUINO

Para acceder a la UART de Arduino desde un PC se utilizará un adaptador USB a TTL.



Figura 4.18 - Cable conversión USB a TTL

bien sea directamente (fase de desarrollo y depuración) o intercalando un dispositivo inalámbrico (módulos XBee) para darle autonomía.



Figura 4.19 - Módulo XBee

Los parámetros de comunicación a utilizar serán 9600bps 8N1.

Cabe destacar que ROS dispone de utilidades específicas para trabajar con comunicaciones serie que facilitan tareas como la serialización de *topics*. El paquete **rosserial** cuenta con herramientas específicas para Arduino.

Sin embargo en este caso no se ha considerado necesario el uso de esta herramienta por la sencillez del protocolo seleccionado.

La comunicación será unidireccional. Desde el PC se enviarán órdenes de velocidad lineal y angular, según el siguiente formato:

- **Velocidad Lineal:** expresada en milímetros por segundo en un número entero de 2 bytes con signo.
- **Velocidad Angular:** expresada en grados por segundo en un número entero de 1 byte con signo.

La selección de unidades (mm/s y grados/s) puede resultar poco convencional. El motivo para utilizar estas medidas es que facilitan el uso de números enteros y campos cortos (dos bytes), preferidos frente a representaciones en coma flotante por agilizar cálculos y simplificar la comunicación.

Para transmitir la información de velocidad se ha definido la siguiente trama serie de 7 bytes:

FRAME_START	VLIN_HI	VLIN_LO	VANG_HI	VANG_LO	FRAME_END_1	FRAME_END_2
-------------	---------	---------	---------	---------	-------------	-------------

- **FRAME_START:** byte indicador de inicio de trama. Por conveniencia para facilitar la depuración se ha seleccionado un símbolo representable ASCII: la letra 'A' (0x41).
- **VLIN_HI:** byte más significativo del entero de 16 bits con la orden de velocidad lineal.
- **VLIN_LO:** byte menos significativo del entero de 16 bits con la orden de velocidad lineal.
- **VANG_HI:** byte más significativo del entero de 16 bits con la orden de velocidad angular.
- **VANG_LO:** byte menos significativo del entero de 16 bits con la orden de velocidad angular.
- **FRAME_END_1 y FRAME_END_2:** bytes indicadores de fin de trama. Por conveniencia para facilitar la depuración se ha seleccionado un símbolo representable ASCII: la letra 'Z' (0x5A).

Tomando como velocidad máxima de referencia los 47.7m/min indicados en las especificaciones de la plataforma (con alimentación de 6V), el rango de velocidades lineales posibles es de -783 mm/s a +783 mm/s. Y las velocidades angulares posibles entre -560°/s y +560°/s.

Así, para movernos a 0.5 m/s (500 mm/s o 0x01F4 en hexadecimal) y en un momento dado efectuar un cambio de trayectoria de 30° en medio segundo (60°/s o 0x3C en hexadecimal), enviaríamos las siguientes tramas:

```
t=000ms: A01F400ZZ (velocidad lineal 0.5 m/s, velocidad angular 0°/s)
t=100ms: A01F43CZZ (velocidad lineal 0.5 m/s, velocidad angular 60°/s)
t=600ms: A01F400ZZ (velocidad lineal 0.5 m/s, velocidad angular 0°/s)
```

4.6.10. PROGRAMA ROS PARA CONTROL DE PLATAFORMA MÓVIL MEDIANTE JOYSTICK

El nodo debe suscribirse al topic joy y traducir los mensajes del joystick a tramas con información de velocidad lineal y angular para enviar a la plataforma móvil.

La comunicación con la plataforma móvil se realizará mediante comunicación serie, utilizando un cable USB a TTL para la desarrollo y mediante un par de módulos XBee (comunicación inalámbrica mediante *bluetooth*) para funcionamiento autónomo.

El programa sigue el esquema general de un nodo ROS suscriptor, en el que una función callback recibe los mensajes recibidos del topic joy y los procesa. El procesamiento consiste en la generación de una trama con información de velocidad y su transmisión serie. Para la transmisión serie se ha utilizado la librería Serial_Q disponible en los apéndices.

Las respuestas de la plataforma móvil se muestran por pantalla para facilitar la depuración.

```
#include <iostream>
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "sensor_msgs/Joy.h"
#include "Serial_Q.h"

#define SERIALDEVICE "/dev/ttyUSB0"

using namespace std;

void joyCallback(const sensor_msgs::Joy& inJoyCommand);
bool processPtuComm();
void getVelCommand(int inVelLineal, int velAngular, char* inComando);

ros::NodeHandle* myNodeHandle = 0;

Serial::Serial_Q *Ptu;

char tmpChar;
char status;
char mySerialDevice[] = "/dev/ttyUSB0";

sensor_msgs::JointState thePtuJointState;

int main(int argc, char** argv) {

    ros::init(argc, argv, "plataforma movil con arduino");
    ros::NodeHandle theNodeHandle;
    myNodeHandle = &theNodeHandle;
    ros::Subscriber theJoySubscriber = theNodeHandle.subscribe("/joy", 10, joyCallback);

    ROS_INFO("Nodo plataforma_movil_con_arduino iniciado");

    Ptu = new Serial::Serial_Q(mySerialDevice, B9600);

    if (Ptu->LastError != NULL) {
        ROS_ERROR("Error conectando con plataforma: %s",Ptu->LastError);
        return -1;
    }

    ROS_INFO("Puerto serie abierto %d",B9600);

    ros::Rate r(1); // 1 hz
    while (myNodeHandle->ok()) {

        //processPtuComm();
        ros::spinOnce();
    }
}
```

```

    }
    cout << "Programa terminado" << endl;
    return 0;
}

void getVelCommand(int inVelLineal, int inVelAngular, char* inComando) {

    int i;
    char zero = '0';

    //ROS_INFO("Lin: %d. Ang: %d", inVelLineal, inVelAngular);
    inComando[0] = 'A';
    inComando[1] = (unsigned char)((inVelLineal&0xFF00) >> 8);
    inComando[2] = (unsigned char)(inVelLineal&0x00FF);
    inComando[3] = (unsigned char)((inVelAngular&0xFF00) >> 8);
    inComando[4] = (unsigned char)(inVelAngular&0x00FF);
    inComando[5] = 'Z';
    inComando[6] = 'Z';
    inComando[7] = 0;

    // Los ceros no llegan a través del cable USB a TTL (investigar motivo, de momento
    reemplazar en origen y destino el valor 0 por su representación ASCII)
    for(i=1;i<=4;i++) {
        if (inComando[i] == 0) {
            inComando[i] = zero;
        }
    }
}

void joyCallback(const sensor_msgs::Joy& inJoyCommand) {

    char theCommand[8];

    float theAxesH = (float)inJoyCommand.axes[0];
    float theAxesV = (float)inJoyCommand.axes[1];

    printf("Recibido comando joystick (%g,%g)\n",theAxesV,theAxesH);

    int theLinVel = -theAxesV * 626;
    int theAngVel = theAxesH * 448;

    if (theLinVel != 0) {
        theAngVel = -theAngVel / 4;
    }

    getVelCommand(theLinVel,theAngVel,theCommand);

    //do {
        Ptu->send(theCommand,7);
    //
    //} while (!processPtuComm());

    ros::Duration(0.1).sleep();

    processPtuComm();
}

bool processPtuComm() {

    string theResp;
    std::size_t theFound;

    if (Ptu->checkDataAndEnqueue()) {
        theResp.assign((char *)Ptu->getFullQueueContent(true));
        ROS_INFO("Plataforma móvil dice: %s",theResp.c_str());
    }

    theFound=theResp.find("*");
    return (theFound!=std::string::npos);
}

```

4.6.11. PROGRAMA DE CONTROL DE MOTORES EN ARDUINO

Para la realización de este ejercicio ha sido necesario programar una plataforma Arduino UNO para realizar las siguientes acciones:

1. Recibir vía serie mediante la única UART disponible las tramas de 6 bytes definidas anteriormente e interpretarlas para extraer los datos contenidos. Para ello se definen 6 estados para distinguir en qué punto de la comunicación de una trama nos encontramos. Si la comprobación de bytes de inicio y fin de trama es satisfactoria, se aceptan y procesan las órdenes de velocidad.
2. Transformar las órdenes de velocidad recibidas en *duties* PWM para cada uno de los motores. Para ello es necesario considerar algunas dimensiones específicas de la plataforma móvil, tales como la distancia entre ruedas de tracción o el diámetro de las ruedas, según se ha indicado anteriormente en el apartado de control de velocidad en lazo abierto para una plataforma de tracción diferencial

A continuación se incluye el código fuente encargado de recibir las órdenes de velocidad vía serie. El programa es válido para el Motor Shield oficial de Arduino. Para usarlo con el shield de Adafruit habría que realizar ligeras modificaciones.

```
#define ESPERANDO_INICIO_TRAMA 0
#define ESPERANDO_VLINEAL_HI 1
#define ESPERANDO_VLINEAL_LO 2
#define ESPERANDO_VANGULAR_HI 3
#define ESPERANDO_VANGULAR_LO 4
#define ESPERANDO_FIN_TRAMA_1 5
#define ESPERANDO_FIN_TRAMA_2 6
#define TRAMA_OK 7

#define BYTE_INICIO_DE_TRAMA 'A'
#define BYTE_FIN_DE_TRAMA_1 'Z'
#define BYTE_FIN_DE_TRAMA_2 'Z'

#define MOTOR_MAX_RPM 230

/* Distancia entre ruedas y diametro rueda en mm */

#define LONGITUD_SEMIEJE_TRACCION_MM 80.0
#define DIAMETRO_RUEDA_MM 65.0

#define PIN_DIR_IZQUIERDA 12
#define PIN_DIR_DERECHA 13
#define PIN_FRENO_IZQUIERDA 9
#define PIN_FRENO_DERECHA 8
#define PIN_PWM_IZQUIERDA 3
#define PIN_PWM_DERECHA 11

unsigned char comando[8] = "";

unsigned char vLinealHi;
unsigned char vLinealLo;
unsigned char vAngularHi;
unsigned char vAngularLo;
signed int vLineal mm segundo;
signed int vAngular grados segundo;
unsigned char pwm derecha;
```

```
unsigned char pwm_izquierda;
int estado;
float mm segundo a rpm = 60 / (PI * DIAMETRO RUEDA MM);
float grados_segundo_a_rad_segundo = PI / 180.0;

int cuenta = 0;

void setup() {

  Serial.begin(9600);

  pinMode(PIN DIR DERECHA,HIGH);
  pinMode(PIN DIR IZQUIERDA,HIGH);
  pinMode(PIN FRENO DERECHA,LOW);
  pinMode(PIN FRENO IZQUIERDA,LOW);

  // Establecemos como salidas los pines correspondientes
  // a las señales de direccion y freno de los dos motores

  estado = ESPERANDO_INICIO_TRAMA;

  Serial.println("READY");
}

void printComando() {

  int i;
  for(i=0;i<7;i++) {
    Serial.print("comando[");
    Serial.print(i,DEC);
    Serial.print("] = ");
    if ((i==0)|| (i==5)|| (i==6)) {
      Serial.println(comando[i]);
    } else {
      Serial.print("0x");
      Serial.println(comando[i],HEX);
    }
  }
}

void loop() {

  unsigned char byteLeido;
  int i;

  if (Serial.available() > 0) {

    byteLeido=Serial.read();

    //Serial.println(byteLeido,DEC);

    if (byteLeido != 0) {

      switch(estado) {

        case ESPERANDO_INICIO_TRAMA:

          for(i=0;i<8;i++) {
            comando[i] = 0;
          }
          if (byteLeido==BYTE_INICIO_DE_TRAMA) {
            comando[estado++] = byteLeido;
          } else {
            //Serial.println("!A");
          }
          break;

        case ESPERANDO_VLINEAL_HI:

          comando[estado++] = byteLeido;
          break;

        case ESPERANDO_VLINEAL_LO:
```

```

    comando[estado++] = byteLeido;
    break;

case ESPERANDO_VANGULAR_HI:

    comando[estado++] = byteLeido;
    break;

case ESPERANDO_VANGULAR_LO:

    comando[estado++] = byteLeido;
    break;

case ESPERANDO_FIN_TRAMA_1:

    comando[estado++] = byteLeido;
    if (byteLeido != BYTE_FIN_DE_TRAMA_1) {
        estado=ESPERANDO_INICIO_TRAMA;
        Serial.println("!Z1");
        printComando();
    }

    break;

case ESPERANDO_FIN_TRAMA_2:

    comando[estado++] = byteLeido;
    if (byteLeido != BYTE_FIN_DE_TRAMA_2) {
        estado=ESPERANDO_INICIO_TRAMA;
        Serial.println("!Z2");
        printComando();
    }

    break;

default:

    break;

}

}

if (estado==TRAMA_OK) {

    float vLinealRuedaIzquierda_mm_segundo, vLinealRuedaDerecha_mm_segundo;
    float rpm_derecha, rpm_izquierda;
    float vComponenteLinealDeVelocidadAngular_mm_segundo;

    // Interpretamos comando

    for (i=1;i<=4;i++) {
        if(comando[i]!='0') {
            comando[i] = 0;
        }
    }

    vLinealHi = comando[1];
    vLinealLo = comando[2];
    vAngularHi = comando[3];
    vAngularLo = comando[4];

    // Reacomponemos el dato de velocidad a partir de sus bytes alto y bajo

    vLineal_mm_segundo = ((unsigned int)vLinealHi)*256 + (unsigned int)vLinealLo;
    vAngular_grados_segundo = ((unsigned int)vAngularHi)*256 + (unsigned int)vAngularLo;

    Serial.print("*");
    Serial.print("Lin");
    Serial.print(vLineal_mm_segundo,DEC);
    Serial.print("Ang");
    Serial.print(vAngular_grados_segundo,DEC);
    Serial.print("*");

    // Consideramos vAngular positiva en sentido antihorario

```

```
vComponenteLinealDeVelocidadAngular_mm_segundo = vAngular_grados_segundo *
LONGITUD SEMIEJE TRACCION MM * grados segundo a rad segundo;

vLinealRuedaDerecha_mm_segundo = vLineal_mm_segundo +
vComponenteLinealDeVelocidadAngular_mm_segundo;
vLinealRuedaIzquierda_mm_segundo = vLineal_mm_segundo -
vComponenteLinealDeVelocidadAngular mm segundo;

// Obtenemos las velocidades de giro de las ruedas en rpm

rpm_derecha = vLinealRuedaDerecha_mm_segundo * mm_segundo_a_rpm;

if (rpm_derecha > MOTOR_MAX_RPM)
    rpm_derecha = MOTOR_MAX_RPM;
else if (rpm_derecha < -MOTOR_MAX_RPM)
    rpm_derecha = -MOTOR_MAX_RPM;

rpm_izquierda = vLinealRuedaIzquierda_mm_segundo * mm_segundo_a_rpm;

if (rpm_izquierda > MOTOR_MAX_RPM)
    rpm_izquierda = MOTOR_MAX_RPM;
else if (rpm_izquierda < -MOTOR_MAX_RPM)
    rpm_izquierda = -MOTOR_MAX_RPM;

// y las traducimos a duties PWM

pwm_derecha = abs(rpm_derecha * 255 / MOTOR_MAX_RPM);
pwm_izquierda = abs(rpm_izquierda * 255 / MOTOR_MAX_RPM);

// Establecemos el sentido de giro

digitalWrite(PIN_DIR_DERECHA, (rpm_derecha > 0)?LOW:HIGH);
digitalWrite(PIN_DIR_IZQUIERDA, (rpm_izquierda > 0)?LOW:HIGH);

// Enviamos duties PWM a motores

analogWrite(PIN_PWM_IZQUIERDA, pwm_izquierda);
analogWrite(PIN_PWM_DERECHA, pwm_derecha);

// Preparamos la siguiente recepcion

estado=ESPERANDO_INICIO_TRAMA;

}

}
```

4.6.12. PRUEBAS

La realización de este ejercicio ha demostrado que la plataforma funciona correctamente a velocidades medias o altas, pero no consigue iniciar la marcha a baja velocidad, lo que se traduce en una apreciable falta de linealidad. En tales condiciones los motores no son capaces de vencer la resistencia inicial para comenzar a moverse (tras ayudarles con la mano continúan moviéndose).

5. APÉNDICE I. HERRAMIENTAS

5.1. Intro Linux

En este capítulo se incluye una introducción muy básica de manejo del sistema operativo Linux a nivel de usuario, suficiente para seguir los ejemplos y procedimientos incluidos en este TFM. Para un estudio más exhaustivo se recomienda acudir a la bibliografía indicada al fina.

GNU/Linux es uno de los términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux con el sistema GNU. Su desarrollo es uno de los ejemplos más prominentes de software libre; todo su código fuente puede ser utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL y otra serie de licencias libres.

El nombre GNU, GNU's Not Unix (GNU no es Unix) viene de las herramientas básicas de sistema operativo creadas por el proyecto GNU, iniciado por Richard Stallman en 1983 y mantenido por la FSF. El nombre Linux viene del núcleo Linux, inicialmente escrito por Linus Torvalds en 1991.

5.1.1. LINUX SHELL

Es un intérprete de comandos. Permite al usuario interactuar con el sistema. Existen varios (sh, ksh, csh, bash ...) siendo Bash el intérprete por defecto en la mayoría de las distribuciones Linux actuales.

5.1.2. INICIO DE SESIÓN

Al iniciar sesión en un shell se desencadenan una serie de acciones que configuran el entorno. En el caso de Bash estas acciones se especifican en los siguientes ficheros:

```
/etc/profile
~/.bash_profile
~/.bashrc
```

y al cerrar la sesión:

```
~/.bash_logout
```

El símbolo '~' equivale a carpeta **home**. Por ejemplo, si el usuario se llama laura, `~/.bashrc` probablemente equivalga a `/home/laura/.bashrc`. Ejecutando el comando `cd` sin mas argumentos, nos posicionaremos en nuestro **home**.

5.1.3. COMANDOS BÁSICOS

A continuación se listan una serie de comandos de uso frecuente:

ls	muestra la lista de ficheros y directorios contenidos en el directorio actual (como <i>dir</i> en DOS).
cd <i>nombredirectorio</i>	cambia al directorio llamado <i>nombredirectorio</i> .
find <i>ruta expr_búsqueda acción</i>	busca archivos (ver tutorial)
passwd	modifica la contraseña del usuario actual.
cat <i>nombrefichero</i>	muestra el contenido del fichero llamado <i>nombrefichero</i> .
pwd	muestra el directorio actual.
exit o logout	cierra la sesión actual.
man <i>nombrecomando</i>	muestra la ayuda (man) del comando <i>nombrecomando</i> .
touch <i>nombrefichero</i>	crea un fichero vacío llamado <i>nombrefichero</i> .
cp <i>ficheroorigen ficherodestino</i>	copia el fichero <i>ficheroorigen</i> como <i>ficherodestino</i> .
mv <i>ficheroorigen ficherodestino</i>	mueve el fichero <i>ficheroorigen</i> a <i>ficherodestino</i> .
mkdir <i>nombredirectorio</i>	crea un directorio llamado <i>nombredirectorio</i> .
rmdir <i>nombredirectorio</i>	elimina un directorio llamado <i>nombredirectorio</i> .
rm <i>nombrefichero</i>	elimina un fichero llamado <i>nombrefichero</i> .
chmod <i>permisos nombrefichero</i>	modifica los permisos de acceso de un fichero.
chown <i>usuario nombrefichero</i>	establece <i>usuario</i> como propietario del fichero <i>nombrefichero</i> .
chgrp <i>grupo nombrefichero</i>	establece <i>grupo</i> como grupo asociado al fichero <i>nombrefichero</i> .

La mayoría de estos comandos admiten modificadores (ej: **ls -la** muestra una información más detallada) que extienden la funcionalidad indicada anteriormente. Se recomienda utilizar **man nombrecomando** para ver las opciones disponibles.

En ocasiones es necesario realizar operaciones con privilegios de superusuario. Para ello se precede el comando **sudo** al comando a ejecutar y el sistema pedirá introducir la contraseña (ej: **sudo chmod 777 nombrefichero**).

Existen muchos más comandos. Un listado mas exhaustivo puede encontrarse en <http://ss64.com/bash/>

5.1.4. COMBINACIONES DE TECLAS EN BASH

Ctrl+A	mueve el cursor al inicio de la línea de comandos.
Ctrl+C	fuerza la terminación de un programa.
Ctrl+D	equivale a exit o logout .
Ctrl+A	mueve el cursor al final de la línea de comandos.
Ctrl+H	genera el caracter de borrado.
Ctrl+L	limpia el terminal.
Ctrl+R	búsqueda en en el histórico de comandos.
Ctrl+Z	suspende la ejecución de un programa.
Flechas arriba/abajo	navega por el historial de comandos.
Flechas izquierda/derecha	mueve el cursor a lo largo de la línea de comandos.
Shift+PageUp/PageDown	desplaza a lo largo del buffer del terminal.
Tab	autocompletar nombre de fichero o de comando.
Tab Tab	muestra las posibilidades de autocompletar.

5.1.5. SISTEMA DE FICHEROS LINUX

Se puede definir el sistema de ficheros de un sistema operativo como aquellas estructuras lógicas y sus correspondientes métodos que utiliza el propio sistema para organizar los ficheros en disco [10].

Los sistema de ficheros GNU/Linux poseen una estructuración jerárquica o “en árbol”. Es decir, el sistema contiene unos directorios (que a su vez podrían contener más subdirectorios), que asocian características de ficheros con los ficheros guardados en la **partición**. Normalmente el soporte físico puede contener más de una partición, y cada una de esas partiones requieren de

formato lógico mediante un sistema de ficheros específico. Los sistemas de ficheros permiten estructurar la información para poder mostrarla, tanto de forma gráfica (si el sistema cuenta con esta característica) como de forma textual mediante los denominados gestores de archivos.

La relación descrita entre los ficheros y la forma de localizarlos se realiza, en los sistemas GNU/Linux, mediante una tabla de asignación de **inodo**. Un inodo contiene los parámetros característicos del objeto referenciado (permisos, fechas, ubicación...). Este objeto puede ser tanto un fichero, un directorio, un enlace simbólico y, por generalizar, cualquier objeto que puede ser entendido por el sistema de ficheros.

Cada inodo está identificado por un número entero único (en el sistema de ficheros), y los directorios son los responsables de guardar ternas de número de inodo y nombre identificativo de fichero. Cada fichero posee un único inodo, pero puede tener varios nombres que lo identifiquen.

Para poder referenciar un fichero se puede utilizar la cadena de texto denominada **ruta**. La ruta es el resultado de la concatenación de los nombres identificativos de directorios y subdirectorios que dibujan la estructura arbórea hasta llegar al fichero, más el nombre del propio fichero.

Los requerimientos esperables de un sistema de ficheros pueden ser:

1. poder acceder a la información (ficheros) de forma óptima
2. soportar permisos de usuario, del grupo del usuario y del “resto de mundo”
3. soportar listas de control de acceso (denominadas ACL's)
4. garantizar la coherencia de la información, así como evitar la fragmentación
5. permitir enlaces (simbólicos y duros)
6. poder recuperar la información después de una caída de tensión brusca (*journaling*)

Existen diversos tipos de sistema de ficheros:

- de disco (*ext2, ext3, reiserfs, xfs, jfs, iso9660, ...*)
- en red (*NFS, CIFS, ...*)
- virtuales (*VFS, SysFS, ..*)
- especiales (*SWAP, ...*)

En los sistemas GNU/Linux, los sistemas de ficheros pueden ser creados desde un terminal mediante la orden “**mkfs**”. La opción que crea uno u otro sistema es la “-t”, a la que se debe añadir el dispositivo (la partición, por ejemplo) físico donde crear el sistema de ficheros.

Por ejemplo, si tuviésemos que crear un sistema de ficheros ext3 en la segunda partición de nuestro primer disco, la orden sería:

```
#mkfs -t ext3 /dev/hda2
```

Para evitar realizar repetidamente la operación anterior en sistema de ficheros habituales, se especifican los parámetros de montaje para cada sistema en el archivo */etc/fstab* de modo que se monten automáticamente en el arranque.

5.1.6. ESTRUCTURA DE DIRECTORIOS

La estructura de directorios habitual de un sistema Linux depende en gran medida de la distribución. En el caso de las distribuciones Debian (y sus derivadas) la estructura es la siguiente:

<i>/boot</i>	contiene el núcleo (kernel) e información indispensable para el arranque del sistema.
<i>/bin</i>	guarda unos pocos programas que estarán disponibles incluso en los modos de ejecución más restringidos (como bash, cat, ls, login, ps).
<i>/sbin</i>	aquí encontramos los programas disponibles sólo para el administrador incluso en los modos de ejecución más restringidos (por ejemplo fsck, getty, halt).
<i>/usr</i>	Programas accesibles a usuarios finales y datos de estos programas que no requieren ser modificados (datos de sólo lectura).
<i>/floppy, /cdrom, /mnt, /media</i>	son directorios para montar disquettes, CD-ROMs y otros sistemas de archivos o dispositivos.
<i>/lib</i>	Librerías indispensables y módulos (especialmente requeridas durante el arranque del sistema).
<i>/etc</i>	Archivos de configuración de diversos programas y servicios.
<i>/dev</i>	Abstracciones a los dispositivos conectados (o que podrían conectarse) al ordenador.
<i>/home</i>	Mantiene información de los usuarios del sistema.
<i>/root</i>	Mantiene información del administrador del sistema.
<i>/tmp</i>	Archivos temporales creados por algunos programas, que serán borrados por el sistema operativo durante el arranque.
<i>/var</i>	En este directorio los programas que lo requieran pueden mantener archivos que deban modificarse frecuentemente.
<i>/proc</i>	Este directorio es virtual, no está presente en el disco, porque es creado por el sistema para intercambiar información con más facilidad.

5.1.7. MONTAJE DEL SISTEMA DE FICHEROS

Para utilizar sistemas de ficheros adicionales al de nuestro sistema Linux (por ejemplo para leer un lapiz de memoria o un CD), es necesario "montarlos" previamente. Esto significa establecer una correspondencia entre el nuevo sistema de ficheros y una ubicación (una carpeta) en el original. Esto se realiza mediante el comando **mount** y normalmente requiere privilegios de superusuario y posiblemente la especificación del tipo de sistema de ficheros a montar.

Por ejemplo, para acceder a la segunda partición de nuestro disco, sabiendo que es de tipo *ext3* podríamos ejecutar el siguiente comando:

```
#mount -t ext3 /dev/hda2 /mnt
```

Hecho esto, la el directorio raíz del nuevo sistema estará accesible en la carpeta */mnt* del original

5.1.8. PERMISOS DE ACCESO

Se pueden establecer permisos sobre un archivo o directorio para:

1. El propietario de un fichero o directorio.
2. Usuarios pertenecientes a grupos a los que también pertenezca el propietario.
3. Resto de usuarios.

Ejemplo:

con el comando `ls -l` veremos un listado de ficheros y directorios. La primera columna contiene entradas de 10 caracteres. `-rw-rw-r---`

El significado de esta notación es el siguiente (caracteres numerados de izquierda a derecha):

1. se trata de un directorio (d) o un fichero (-)
2. el propietario tiene permisos de lectura (r) o no (-)
3. el propietario tiene permisos de escritura (w) o no (-)
4. el propietario tiene permisos de ejecución (x) o no (-)
5. usuarios del mismo grupo tienen permisos de lectura (r) o no (-)
6. usuarios del mismo grupo tienen permisos de escritura (w) o no (-)
7. usuarios del mismo grupo tienen permisos de ejecución (x) o no (-)
8. resto de usuarios tienen permisos de lectura (r) o no (-)
9. resto de usuarios tienen permisos de escritura (w) o no (-)
10. resto de usuarios tienen permisos de ejecución (x) o no (-)

Estos permisos se pueden cambiar con el comando **chmod**

Ejemplos:

- **chmod +x nombrefichero** daría permisos de ejecución al fichero nombrefichero (a propietario, grupo y resto)
- **chmod 664 nombrefichero** daría permisos de lectura y escritura a propietario y grupo, y solo lectura a resto. Cada dígito se interpreta como su representación binaria sobre los caracteres rwx indicados anteriormente (000 = nada permitido, 110 = lectura y escritura permitidas, 111 = todo permitido, ...)

5.1.9. GESTOR DE PAQUETES

Un sistema de gestión de paquetes es una colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software. Suelen tener la capacidad de desinstalar los paquetes de manera recursiva, de forma que se eliminan todos los paquetes que dependen del paquete a desinstalar y todos los paquetes de los que el paquete a desinstalar depende, respectivamente.

En sistemas derivados de Debian, la gestión de paquetes consiste fundamentalmente en las herramientas complementarias **apt** y **dpkg**.

Comandos APT

apt-get install <package>	Descarga de un paquete y de todas sus dependencias, instalación o actualización.
apt-get remove [--purge] <package>	Desinstalación de un paquete y todos los paquetes que dependen de él.
apt-get update	Actualización de la información del gestor desde los servidores de paquetes (los servidores o <i>mirrors</i> se configuran en <i>/etc/apt/sources.list</i>).
apt-get upgrade	Actualización de los paquetes instalados a sus versiones más recientes siempre y cuando las dependencias no requieran la instalación de nuevos paquetes.
apt-get dist-upgrade	Actualización de los paquetes instalados a sus versiones más recientes, instalando o desinstalando las dependencias necesarias si han cambiado.
apt-cache search <pattern>	Búsqueda de paquetes que contengan determinado texto en el nombre o la descripción.
apt-cache show <package>	Descripción de un paquete.
apt-cache showpkg <package>	Descripción detallada de un paquete y sus relaciones con otros paquetes.

Comandos dpkg

dpkg -i <package.deb>	Instalación de un archivo de instalación Debian.
dpkg -c <package.deb>	Listado del contenido de un archivo de instalación Debian.
dpkg -I <package.deb>	Listado de información de paquete a partir de un archivo de instalación Debian.
dpkg -r <package>	Desinstalación de un paquete.
dpkg -P <package>	Purga de un paquete instalado (equivalente a desinstalación mas eliminación de archivos de configuración).
dpkg -L <package>	Listado de archivos instalados por un paquete.
dpkg -s <package>	Descripción de un paquete instalado.
dpkg-reconfigure <package>	Reconfiguración de un paquete instalado.

5.2. IDE Eclipse

5.2.1. INTRODUCCIÓN

En este tutorial se describe brevemente el manejo del IDE Eclipse específico para C++, denominado C/C++ Development Toolikt (CDT).

5.2.2. INSTALACIÓN DEL IDE EN LINUX

La instalación del CDT en Linux puede hacerse facilmente mediante el gestor de paquetes de la distribución (por ejemplo apt en sistemas Debian o Ubuntu)

```
sudo apt-get install eclipse-cdt
```

Aunque por este método es posible que la versión instalada no sea la más reciente. Si queremos la mas reciente podemos realizar la instalación de modo manual. Para ello descargamos la versión mas reciente de www.eclipse.org/downloads.

El programa se llama **Eclipse IDE for C/C++ developers** y está disponible para plataformas de 32 o 64 bits.

El programa descargado no necesita compilación, aunque es posible que sea necesario instalar algunos prerequisites, como el entorno de ejecución Java. Si es necesario lo instalaremos utilizando apt:

```
sudo apt-get install openjdk-7-jre
```

Una vez descargado el programa e instalados los prerequisites, descomprimos el fichero de instalación y (opcionalmente) movemos la carpeta resultante a */opt*, de modo que los archivos del programa estén en */opt/eclipse/*.

Para lanzar el programa desde el escritorio podemos crear un fichero llamado **eclipse.desktop** en la carpeta escritorio (*~/Desktop*) con el siguiente contenido:

```
[Desktop Entry]
Type=Application
Name=Eclipse
Comment=Eclipse Integrated Development Environment
Icon=/opt/eclipse/icon.xpm
Exec=bash -i -c "/opt/eclipse/eclipse"
Terminal=false
Categories=Development;IDE;
```

Existen otras métodos (por ejemplo para la integración en Unity) que pueden seguirse, pero es importante que la entrada **Exec** contenga la llamada al ejecutable mediante el comando **bash -i -c**

`"/opt/eclipse/eclipse"` (y no directamente mediante `/opt/eclipse/eclipse`) para que antes de abrir eclipse se carguen las variables de entorno de nuestro usuario (`~/bashrc`).

5.2.3. CREACIÓN DE UN PROYECTO

Ejecutar **File->New->C++ Project**. Seleccionar **Empty Project**. Toolchain **Linux GCC**. Darle un nombre al proyecto. Pulsar **Finish**.

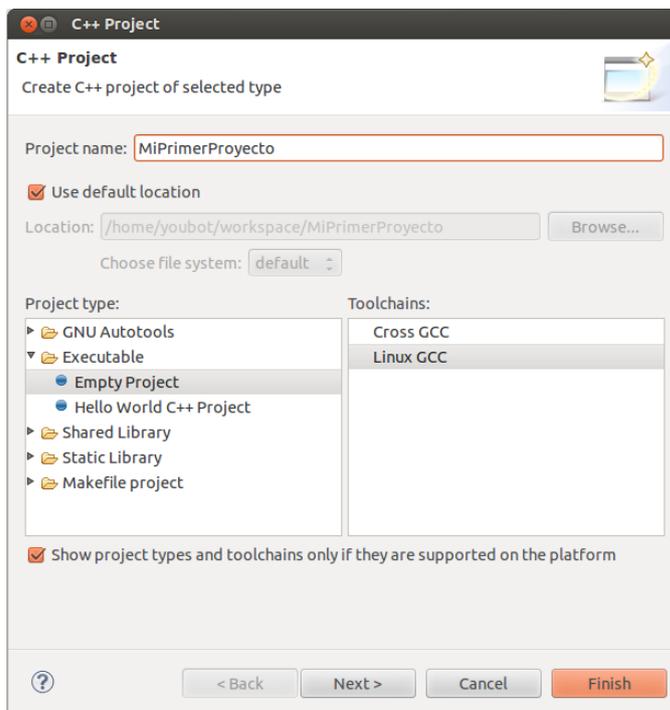


Figura 5.1 - Creación de un prooyecto con Eclipse CDT (I)

Añadir un fichero fuente mediante **File->New->Source File**. Darle un nombre al fichero (terminado en `.cpp`). Pulsar **Finish**

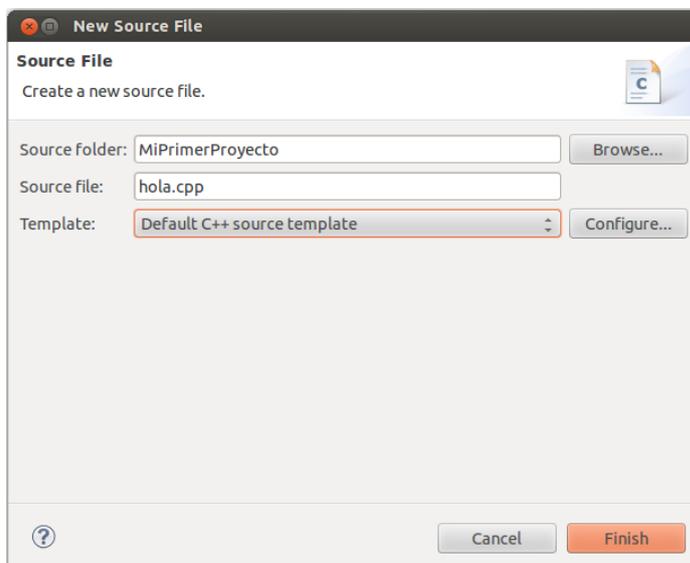
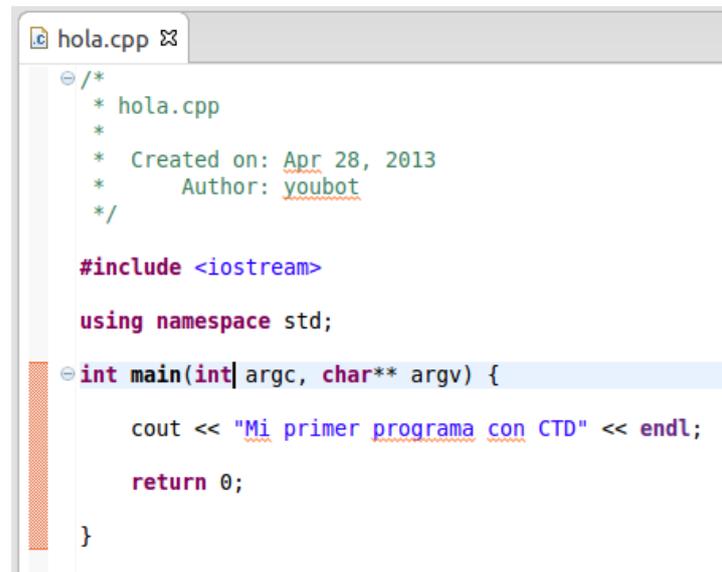


Figura 5.2 - Creación de un prooyecto con Eclipse CDT (II)

Escribir un pequeño programa de prueba



```

hola.cpp
/*
 * hola.cpp
 *
 * Created on: Apr 28, 2013
 * Author: youbot
 */

#include <iostream>

using namespace std;

int main(int argc, char** argv) {

    cout << "Mi primer programa con CTD" << endl;

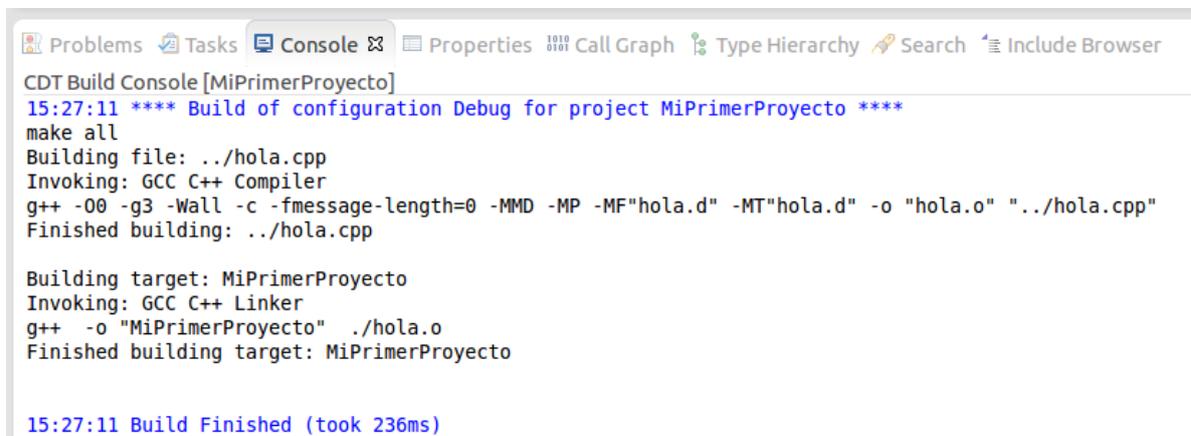
    return 0;

}

```

Figura 5.3 - Programa de prueba C++

Compilar y generar ejecutable mediante **Project->Build All**. La ventana de consola debería mostrar una serie de mensajes indicando las acciones realizadas y el resultado



```

Problems Tasks Console Properties Call Graph Type Hierarchy Search Include Browser
CDT Build Console [MiPrimerProyecto]
15:27:11 **** Build of configuration Debug for project MiPrimerProyecto ****
make all
Building file: ../hola.cpp
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"hola.d" -MT"hola.d" -o "hola.o" "../hola.cpp"
Finished building: ../hola.cpp

Building target: MiPrimerProyecto
Invoking: GCC C++ Linker
g++ -o "MiPrimerProyecto" ./hola.o
Finished building target: MiPrimerProyecto

15:27:11 Build Finished (took 236ms)

```

Figura 5.4 - Resultado de compilación

Si no se produjeron errores de compilación, es posible ejecutar el programa de prueba mediante **Run->Run**. Ahora la consola debería mostrar la salida de nuestro programa.

Puesto que no hemos hecho cambios en la configuración por defecto, la compilación se habrá hecho en modo de depuración (*debug*). Suponiendo que nuestro espacio de trabajo para Eclipse se llame *workspace* y esté situado en nuestro directorio *home*, se habrá generado un ejecutable con el nombre del proyecto en la carpeta **~/workspace/MiPrimerProyecto/Debug**.

El proyecto se habrá creado con dos configuraciones: *Debug* y *Release*. La primera está optimizada para depurar el código (ejecución paso a paso, uso de *breakpoints*, inspección de

variables durante la ejecución) y es la adecuada en la etapa de desarrollo. La segunda está optimizada para ejecución en producción (más rápida, no apta para actividades de depuración). La configuración activa por defecto es *Debug*. Se puede cambiar a *Release* mediante **Project->Build Configurations->Set Active**

También es posible modificar el nombre y la ubicación del ejecutable desde **Run->Run Configurations**

5.2.4. DEPURACIÓN

Para ejecutar el programa en modo depuración hay que pinchar **Run->Debug**. En este modo se pueden establecer puntos de ruptura o *breakpoints*. Es una indicación en una línea específica del programa en la que queremos que se detenga el flujo del mismo. Se puede añadir/eliminar un *breakpoint* mediante **Run->Toggle Breakpoint**

Cuando ejecutamos el programa en modo depuración, puede ser de utilidad visualizar el contenido de algunas variables o expresiones. Para ello se pueden seleccionar en el editor y añadir a la lista de variables visualizadas con la opción **Add watch expression** del menú de contexto (botón derecho del ratón) o bien desde la misma ventana de expresiones de depuración.

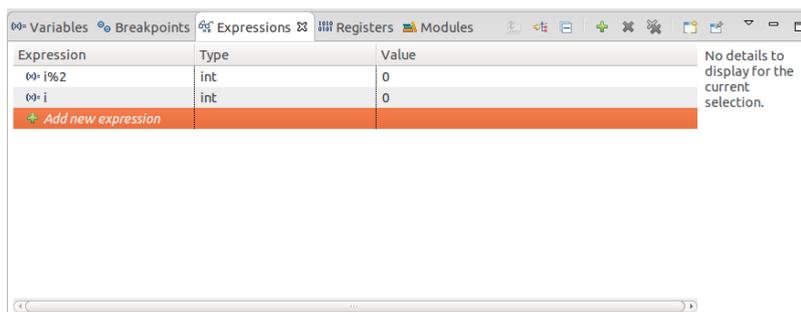


Figura 5.5 - Ventana de monitorización (watch)

5.2.5. CONFIGURACIÓN ESPECÍFICA ROS

Para trabajar con paquetes ROS hay que realizar algunas modificaciones en los archivos de configuración

1. Para cada paquete que se desea abrir con Eclipse, ejecutar los siguientes comandos (desde la carpeta del paquete)
 - Para **ROS Fuerte** (espacio de trabajo generado con ros_ws)

```
make eclipse-project
cmake -G"Eclipse CDT4 - Unix Makefiles"
```

1. Para **ROS Groovy** (espacio de trabajo generado con catkin)

```
cd ~/catkin_ws
```

```
catkin make --force-cmake -G"Eclipse CDT4 - Unix Makefiles"
```

1. Desde Eclipse, importar el proyecto mediante File->Import->General->Existing Projects into Workspace y seleccionar la carpeta raíz del paquete.
2. Una vez importado, especificar las variables de entorno necesarias para la ejecución en Run->Run Configurations->C/C++ application. Dentro de la pestaña Environment añadir las variables de entorno ROS_ROOT y ROS_MASTER_URI. Ejemplo:

```
ROS_ROOT=/opt/ros/ fuerte/share/ros  
ROS_MASTER_URI=http://localhost:11311
```

5.3. Programación Orientada a Objetos en C++

5.3.1. INTRODUCCIÓN

Según [3] un sistema se califica como Orientado a Objetos cuando reúne las características de: abstracción, encapsulación, herencia y polimorfismo.

- **Abstracción** consiste en la generalización conceptual de un determinado conjunto de objetos y de sus atributos y propiedades.
- **Encapsulación** se refiere a la capacidad de agrupar en un entorno distintos elementos.
- **Herencia** está fuertemente ligada a la reutilización de código. En C++ se implementa mediante un mecanismo denominado derivación de clases, de modo que se posibilita el uso del código creado para la **clase base** por parte de sus **clases derivadas**.
- **Polimorfismo** se refiere a la posibilidad de acceder a un variado rango de funciones distintas a través de un mismo interfaz.

5.3.2. CLASES Y OBJETOS

La abstracción y encapsulación en C++ están representada por el concepto de **clase**. Abstracción en el sentido de que una clase define una serie de atributos genéricos de determinados objetos con características comunes. Y encapsulación en cuanto a que la clase comprende tanto los datos de que constan los objetos como los procedimientos que permiten manipularlos.

Una clase en C++ puede verse como un *struct* en C que, además de variables, admite funciones y permite diferentes niveles de acceso en función de unas etiquetas.

La finalidad de las clases en C++ es proporcionar una herramienta para crear nuevos tipos. Una clase es un tipo definido por el programador. La idea fundamental en definir un nuevo tipo consiste en mantener separados los detalles de implementación de las propiedades para su uso.

Un **objeto** es una instancia de una clase. Se crea mediante una función denominada **constructor**. Los constructores tienen el mismo nombre que la clase y su objetivo es realizar las funciones necesarias de inicialización. Una clase puede tener varios constructores, cada uno con diferentes argumentos.

Si un objeto hace uso de recursos (ficheros, memoria, mecanismos de bloqueo, etc.) es conveniente incluir el código necesario para su liberación en un **destructor**, que será llamado de manera automática cuando el objeto salga fuera de ámbito. El nombre del destructor consiste en el nombre de la clase precedido del símbolo '~'.

Ejemplo:

Mónica y Alberto son objetos de clase *Persona*. La clase *Persona* tiene dos **atributos** o miembros (*Nombre* y *Edad*), un **método** o función miembro (*Habla*), un constructor (*Persona*) y un destructor (*~Persona*).

```
#include <iostream>
#include <cstring>

using namespace std;

class Persona {
public:
    char* Nombre;
    int Edad;
    Persona(const char* inNombre, int inEdad);
    ~Persona();
    void Habla();
};

Persona::Persona(const char* inNombre, int inEdad) {
    Nombre = new char[strlen(inNombre)+1];
    strcpy(Nombre, inNombre);
    Edad = inEdad;
}

Persona::~Persona() {
    cout << Nombre << " ha muerto a los " << Edad << " años." << endl;
}

void Persona::Habla() {
    cout << "hola! mi nombre es " << Nombre << endl;
}

int main(int argc, char **argv) {

    Persona alberto("Alberto",25);
    Persona monica("Mónica",28);
    alberto.Habla();
    monica.Habla();

    if (alberto.Edad > monica.Edad) {
        cout << alberto.Nombre << " es mayor que " << monica.Nombre << endl;
    } else {
        cout << alberto.Nombre << " NO es mayor que " << monica.Nombre << endl;
    }
    return 0;
}
```

Compilación y Ejecución del programa:

```
youbot@ubuntu:~$ mkdir borrame
youbot@ubuntu:~/borrame$ vi test1.cpp
youbot@ubuntu:~/borrame$ g++ test1.cpp -o test1
youbot@ubuntu:~/borrame$ ./test1
hola! mi nombre es Alberto
hola! mi nombre es Mónica
Alberto NO es mayor que Mónica
Mónica ha muerto a los 28 años.
Alberto ha muerto a los 25 años.
```

5.3.3. EL PUNTERO IMPLÍCITO THIS

Para referirse a miembros de un objeto sin ambigüedades a veces es útil utilizar el puntero implícito **this**. Por ejemplo, la función Habla() de la clase Persona podría haberse escrito como:

```
void Persona::Habla() { cout << "hola! mi nombre es " << this->Nombre << endl; }
```

donde this se entiende implícitamente declarado como:

```
Persona const* this;
```

5.3.4. HERENCIA

Julián es un objeto de clase Asesino, que **hereda** de *Persona* los métodos y atributos y añade el método público *Mata* y el atributo privado *NumCrímenes*. La clase *Asesino* se denomina clase **derivada** de *Persona*.

En C++ es posible derivar de más de una clase.

```
...
class Asesino : public Persona {
private:
    int NumCrímenes;
public:
    Asesino(const char* inNombre, int inEdad);
    ~Asesino();

    void Mata(Persona* inVictima);
};

Asesino::Asesino(const char* inNombre, int inEdad) : Persona(inNombre,inEdad) {
    NumCrímenes = 0;
}

Asesino::~Asesino() {
    cout << "Tras dejar " << NumCrímenes << " crímenes a sus espaldas ";
}

void Asesino::Mata(Persona* inVictima) {
    NumCrímenes++;
    delete inVictima;
}

int main(int argc, char **argv) {

    Persona alberto("Alberto",25);
    Persona monica("Mónica",28);

    Persona* victima1 = new Persona("Juan",22);
    Persona* victima2 = new Persona("María",19);
    Asesino julian("Julián",35);

    alberto.Habla();
    monica.Habla();
    julian.Habla();

    julian.Mata(victima1);
```

```
    julian.Mata(victima2);

    if (alberto.Edad > monica.Edad) {
        cout << alberto.Nombre << " es mayor que " << monica.Nombre << endl;
    } else {
        cout << alberto.Nombre << " NO es mayor que " << monica.Nombre << endl;
    }

    return 0;
}
```

Ejecución del programa modificado:

```
hola! mi nombre es Alberto
hola! mi nombre es Mónica
hola! mi nombre es Julián
Juan ha muerto a los 22 años.
María ha muerto a los 19 años.
Alberto NO es mayor que Mónica
Tras dejar 2 crímenes a sus espaldas Julián ha muerto a los 35 años.
Mónica ha muerto a los 28 años.
Alberto ha muerto a los 25 años.
```

5.3.5. CONTROL DE ACCESO

Mediante las etiquetas *public*, *private*, *protected* y *friend* en la definición de una clase, se puede especificar la accesibilidad de sus miembros.

- Los miembros *public* no tienen restricciones de acceso. Forman parte de la interfaz de la clase con el exterior
- Los miembros *private* solamente son accesibles por otros miembros de la misma clase.
- Los miembros *protected* son accesibles por los miembros de la misma clase y de sus clases derivadas.

Los miembros *private* y *protected* se usan para la funcionalidad interna de la clase, que no interesa conocer desde el punto de vista del uso de los objetos de la clase.

El método *Mata* de la clase *Asesino* puede invocarse desde la función *main* porque es público, sin embargo si tratásemos de acceder al atributo privado *NumCrímenes* obtendríamos un error de compilación. Al ser privado solamente puede ser accedido por los métodos de la propia clase (como por ejemplo su destructor). Si fuera protegido (*protected*) sería también accesible por los métodos de clases derivadas.

1. Una función declarada como *friend* en una clase tiene acceso a los miembros privados de la clase sin ser estrictamente miembro de la misma. Esto es de utilidad por ejemplo si se necesita acceder a miembros privados de más de una clase. Si tenemos las clases *Vector* y *Matriz*, podemos definir el operador producto de *Matriz* por *Vector* como función amiga de ambas (ejemplo extraído de [12]).

5.3.6. POLIMORFISMO

En C++ se establece mediante la **sobrecarga** de identificadores y operadores y mediante las **funciones virtuales**.

El término sobrecarga se refiere al uso del mismo identificador u operador en distintos contextos con distintos significados. Por ejemplo la suma de números reales y la suma de números imaginarios mediante el operador "+" da lugar a operaciones diferentes.

Mediante la sobrecarga de funciones un mismo nombre puede representar distintas funciones con distinto tipo y número de argumentos, lo que favorece la legibilidad del código.

Las funciones virtuales son funciones miembro de una clase destinadas a ser definidas o redefinidas en clases derivadas. Se denominan **virtuales puras** cuando solamente están definidas en clases derivadas (es decir, en la clase base no se implementan). Si una clase contiene métodos virtuales puros se denomina clase **abstracta** y no admite ser instanciada.

Ejemplo:

```
#include <iostream>
#include <cstring>

using namespace std;

class Abstracta {
public:
    // con el modificador 'virtual' indicamos que es una función virtual
    // Al igualar a 0 indicamos que la función es además virtual pura
    virtual void funcionVirtualPura() = 0;
};

class HijaDeAbstracta : public Abstracta {
public:
    HijaDeAbstracta(){};
    virtual ~HijaDeAbstracta(){};
    void funcionVirtualPura();
};

void HijaDeAbstracta::funcionVirtualPura(){
    cout << "Soy un objeto de clase HijaDeAbstracta y esta es mi versión de
funcionVirtualPura()" << endl;
}

class NoAbstracta {
public:
    NoAbstracta(){};
    virtual ~NoAbstracta(){};
    virtual void funcionVirtual();
};

void NoAbstracta::funcionVirtual(){
    cout << "Soy un objeto de clase NoAbstracta y esta es mi versión de funcionVirtual()" <<
endl;
}

class HijaDeNoAbstracta : public NoAbstracta {
public:
    HijaDeNoAbstracta(){};
    virtual ~HijaDeNoAbstracta(){};
};
```

```

    virtual void funcionVirtual();
};

void HijaDeNoAbstracta::funcionVirtual(){
    cout << "Soy un objeto de clase HijaDeNoAbstracta y esta es mi versión de funcionVirtual()"
<< endl;
}
int main(int argc, char **argv) {
    NoAbstracta NoA;
    HijaDeNoAbstracta HijaDeNoA;
    HijaDeAbstracta HijaDeA;

    NoA.funcionVirtual();
    HijaDeNoA.funcionVirtual();
    HijaDeA.funcionVirtualPura();
    // Abstracta A; Esto generaría un error de compilación!
    return 0;
}

```

Ejecución:

```

Soy un objeto de clase NoAbstracta y esta es mi versión de funcionVirtual()
Soy un objeto de clase HijaDeNoAbstracta y esta es mi versión de funcionVirtual()
Soy un objeto de clase HijaDeAbstracta y esta es mi versión de funcionVirtualPura()

```

5.3.7. COMPOSICIÓN Y AGREGACIÓN

La composición en C++ es el uso de clases como miembros de otras clases en la que existe una relación de pertenencia. Por ejemplo la clase persona podría estar compuesta por dos miembros de clase Brazo y dos de clase Pierna. La creación/destrucción de un objeto Persona implica la creación/destrucción de sus extremidades.

La agregación en C++ es el uso de clases como miembros de otras clases sin relación de pertenencia. La clase Duo está compuesta por dos miembros de clase Persona. Estas dos personas existen antes de la creación y permanecen después de la destrucción del dúo.

Ejemplo:

```

#include <iostream>
#include <cstring>

using namespace std;
enum lado {
    izquierdo = 0,
    derecho = 1
};
class ExtremidadPar {
public:
    lado Lado;
    ExtremidadPar(lado inLado) {
        this->Lado = inLado;
    }
};

class Brazo : ExtremidadPar {
public:
    Brazo(lado inLado) : ExtremidadPar(inLado) {}
};

class Pierna : ExtremidadPar {

```

```
public:
    Pierna(lado inLado) : ExtremidadPar(inLado) {}
};

class Persona {
public:
    char nombre[100];
    Brazo BrazoIzquierdo;
    Brazo BrazoDerecho;
    Pierna PiernaIzquierda;
    Pierna PiernaDerecha;

    Persona(const char* inNombre): BrazoIzquierdo(izquierdo), BrazoDerecho(derecho),
    PiernaIzquierda(izquierdo), PiernaDerecha(derecho) {
        strcpy(nombre,inNombre);
    }
};

class Duo {
public:
    Persona* p[2];
    Duo(Persona* p0, Persona* p1){
        p[0] = p0;
        p[1] = p1;
    }
};

int main(int argc, char **argv) {
    Persona p1("Susana");
    Persona p2("Beatriz");
    Duo d(&p1, &p2);
}
```

La diferencia entre composición y agregación en C++ es principalmente semántica, pudiendo ser su implementación muy similar.

5.4. Control de Revisiones Software

5.4.1. INTRODUCCIÓN

En [18] se define control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación.

Un sistema de control de versiones debe proporcionar:

1. Un mecanismo de almacenamiento de los elementos que deba gestionar.
2. La posibilidad de realizar cambios sobre los elementos almacenados.
3. El registro histórico de las acciones realizadas.

En este tutorial se describen brevemente los sistemas Subversion y Git.

5.4.2. CONCEPTOS BÁSICOS Y TERMINOLOGÍA:

- Los sistemas de control de versiones almacenan los datos actuales e históricos en un **repositorio**.
- La historia de la información se referencian mediante **revisiones**. La revisión más reciente se denomina **HEAD**. Determinadas revisiones relevantes (versiones públicas, versiones especialmente estables, versiones instaladas en producción, etc.) suelen etiquetarse para ser fácilmente referenciadas (**TAGS**).
- En ocasiones puede ser conveniente realizar ramificaciones o bifurcaciones de la línea de desarrollo. Por ejemplo, una modificación compleja o drástica que vaya a desestabilizar la línea de desarrollo por un tiempo prolongado es mejor separarla creando un **branch** y, una vez terminada y estable, reintegrarla (**merge**).
- A la rama principal se la conoce como **trunk**.
- La actualización de una rama con los cambios procedentes de la principal (**trunk**) se llama **reverse merge**
- A la acción de enviar al repositorio modificaciones locales para integrarlas en la línea de desarrollo se la denomina **commit** y genera una nueva revisión.
- A la acción de crear una copia local que se corresponda con una determinada versión del repositorio se la denomina **check-out**. Si la copia local no está sujeta a operaciones de control de versiones se denomina **export**. La copia local sujeta a operaciones de control de versiones se llama **working copy** o copia de trabajo.
- A la acción de actualizar una copia local o **check-out** para sincronizarla con los últimos cambios del repositorio se la denomina **update**.
- A la acción de añadir un archivo o directorio a la línea de desarrollo se la denomina **import**
- Cuando dos programadores trabajan simultáneamente sobre una misma versión de un archivo puede suceder que las modificaciones de ambos sean compatibles o que entren en

conflicto (por ejemplo, si ambos modifican una misma línea del archivo con diferente contenido). El primero en realizar un *commit* conseguirá fijar su modificación en la línea de desarrollo generando una nueva revisión. El segundo se verá forzado a actualizar su copia local modificada (mediante un *merge*) y resolver los conflictos creados por modificaciones incompatibles con las del primer programador.

5.4.3. FORMAS DE COOPERACIÓN

Existen dos esquemas básicos de funcionamiento para que los usuarios puedan ir aportando sus modificaciones:

- **Exclusiva:** En este esquema para poder realizar un cambio es necesario comunicar al repositorio el elemento que se desea modificar y el sistema se encargará de impedir que otro usuario pueda modificar dicho elemento. Una vez hecha la modificación, ésta se comparte con el resto de colaboradores. Si se ha terminado de modificar un elemento entonces se libera ese elemento para que otros lo puedan modificar. Este modo de funcionamiento es el que usa por ejemplo *SourceSafe*. Otros sistemas de control de versiones (Ejemplos *subversion*), aunque no obligan a usar este sistema, disponen de mecanismos que permiten implementarlo.
- **Colaborativa:** En este esquema cada usuario modifica la copia local y cuando el usuario decide compartir los cambios el sistema automáticamente intenta combinar las diversas modificaciones. El principal problema es la posible aparición de conflictos que deban ser solucionados manualmente o las posibles inconsistencias que surjan al modificar el mismo fichero por varias personas no coordinadas. Además, esta semántica no es apropiada para ficheros binarios. Los sistemas de control de versiones *subversion* o *Git* permiten implementar este modo de funcionamiento.

5.4.4. ARQUITECTURAS DE ALMACENAMIENTO (REPOSITORIOS)

- **Centralizados:** existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son *CVS* y *Subversion*.

Distribuidos: Cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: *Git* y *Mercurial*.

5.4.5. SUBVERSION

Subversion es un sistema de control de versiones diseñado específicamente para reemplazar al popular *CVS*. En *Subversion* todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en un instante determinado del repositorio que se está trabajando.

La estructura habitual de un repositorio de *Subversion* es:

- **Trunk**: desarrollo principal.
- **Tags**: ubicación de las versiones congeladas.
- **Branches**: ubicación con versiones de desarrollo paralelas al trunk.

Documentación de referencia: <http://svnbook.red-bean.com/>

Guía Rápida Subversion

svn checkout -username nombre -password contraseña http://svn.collab.net/repos/svn/trunk micopiadetrabajo	Generar una copia local de la rama principal <i>trunk</i> de un repositorio remoto. En repositorios accesibles públicamente se omitirían las credenciales.
svn update	Actualizar nuestra copia local con los últimos cambios del repositorio. El comando actualiza por defecto a la revisión <i>HEAD</i> (la más reciente). Si queremos actualizar a una revisión concreta añadiríamos el modificador -r numrevision
svn stat	Comprobar el estado de la copia local (identificar archivos desactualizados, archivos no pertenecientes al repositorio, etc.)
svn revert	Revertir modificaciones.
svn diff nombearchivo	Ver las diferencias entre la versión local modificada de un archivo y la versión del repositorio.
svn commit -m "comentario explicativo de la modificación" nombearchivo	Incorporar los cambios de un archivo al repositorio. Si se omite el nombre del archivo se entiende que el commit incluye todos los archivos modificados.
svn add nombearchivo	Añadir un archivo local al repositorio. La transferencia se realizará en el siguiente <i>commit</i> .
svn import -m "Añadida carpeta" nombrecarpeta http://urldemirepositorio/ruta/hasta/nombrecarpeta	Añadir una carpeta con todo su contenido.

svn move nombrearchivo nuevonombrearchivo	Renombrar un archivo.
svn copy nombrearchivo nuevonombrearchivo	Copiar un archivo.
svn delete nombrearchivo nuevonombrearchivo	Eliminar un archivo.
svn blame nombrearchivo	Mostrar los números de revisión y nombre de los responsables de las modificaciones de un fichero.

Es importante destacar que las operaciones de renombrar, copiar, mover o eliminar no serán incorporadas al repositorio hasta que se realice un *commit*.

5.4.6. Git

Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente

Documentación de referencia: <http://git-scm.com/documentation>

Guía Rápida Git

git clone git://projects.archlinux.org/pacman.git pacman	El check-out en Git se llama clone . Crear una copia local de git://projects.archlinux.org/pacman en la carpeta pacman.
git config user.name "Your Name" git config user.email "me@example.com"	Establecen las credenciales para identificarnos al realizar <i>commits</i> .
git branch trabajo git checkout trabajo	En Git es habitual no trabajar en la rama principal (<i>master</i>). Se suele crear un <i>branch</i> . Crear un <i>branch</i> llamado <i>trabajo</i> y apuntar a éste en la copia local.
git rebase master	Es posible que se hayan producido cambios en la rama <i>master</i> desde que iniciamos nuestro desarrollo en la rama <i>trabajo</i> . Actualizar <i>trabajo</i> con los últimos cambios de <i>master</i> .
git checkout master	Volver a la rama <i>master</i> . No se puede abandonar una rama con modificaciones locales. Para cambiar de una rama a otra es necesario realizar un <i>commit</i> de los cambios realizados (o bien revertirlos si no interesan).
git merge trabajo	Hemos terminado nuestro desarrollo en la rama <i>trabajo</i> .

	Incorporar los cambios a la rama <i>master</i> .
git branch	Ver las ramas disponibles.
git pull	Actualizar nuestra copia local con los últimos cambios realizados en la rama actual.
git add ficherosmodificado1 ficherosmodificado2	Añade las modificaciones en los archivos indicados a la lista de cambios a incorporar en el siguiente <i>commit</i>
git commit -s	Incorporar las modificaciones en la rama actual.
git push	Transferir cambios al repositorio remoto. A diferencia de en <i>Subversion</i> no viene implícito en el <i>commit</i> .
git log	Ver las modificaciones de la rama actual.

5.5. Procedimientos básicos ROS

A continuación se incluyen algunos procedimientos relativos al uso, configuración e instalación de ROS:

1. Instalación de ROS
2. Creación y uso de espacios de trabajo
3. Navegación por el sistema de ficheros
4. Creación de paquetes
5. Formato URDF de descripción de robots

Conviene indicar que algunos procedimientos difieren según qué versión de ROS se utilice. A fecha de redacción de estos tutoriales (Mayo 2013) la versión más reciente de ROS es **Groovy**. Sin embargo la aplicación principal de este trabajo (Youbot) está oficialmente soportada para la versión anterior (**Fuerte**). De modo que se detallarán algunos procedimientos para ambas versiones de ROS. Salvo que se indique lo contrario, el sistema operativo se supondrá **Ubuntu 12.04 (Precise)**.

5.5.1. INSTALACIÓN DE ROS

En este tutorial se describe el proceso de instalación de ROS (Fuerte y Groovy) en **Ubuntu 12.04**. Información más reciente sobre instalaciones de ROS puede encontrarse en [1] y [2].

En primer lugar es necesario indicar al gestor de paquetes dónde obtener los paquetes ROS. Para ello modificamos el fichero `/etc/apt/sources.list.d` y actualizaremos apt:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" >
/etc/apt/sources.list.d/ros-latest.list'
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
sudo apt-get update
```

A continuación instalaremos el sistema completo. En ROS Fuerte el nombre del paquete es **ros-fuerte-desktop-full**. EN ROS Groovy sería **ros-groovy-desktop-full**.

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

Para el correcto funcionamiento del sistema ROS es necesario establecer una serie de variables de entorno. Para ello existe un script en `/opt/ros/fuerte/setup.bash`. De modo que incluiremos una llamada a este script en nuestro `~/.bashrc`. de modo que las variables estén disponibles cada vez que abramos un terminal.

Los siguientes comandos realizan la acción indicada y muestran las variables de entorno ROS (como anteriormente, se asume ROS Fuerte. Para Groovy simplemente reemplazar por `/opt/ros/groovy/setup.bash` en la primera instrucción).

```
echo "source /opt/ros/fuerte/setup.bash" >> ~/.bashrc
. ~/.bashrc
export | grep ROS
```

Si todo ha ido bien ROS estará instalado en nuestro sistema. Para comprobarlo podemos iniciar el *Master* con el comando.

```
roscore
```

Si vemos un mensaje como el siguiente, la instalación se habrá realizado correctamente. Podemos finalizar el proceso mediante Ctrl+C.

```
...
process[rosout-1]: started with pid [2904]
started core service [/rosout]
```

Finalmente instalamos las utilidades **python-rosinstall** y **python-rosdep** que no vienen por defecto en el paquete de instalación de ROS pero son utilizadas muy habitualmente

```
sudo apt-get install python-rosinstall python-rosdep
```

Si el comando anterior diese algún problema, instalaremos rosdep mediante **pip**.

```
sudo apt-get install python-rosinstall python-rosdep
```

5.5.2. CREACIÓN Y USO DE ESPACIOS DE TRABAJO ROS

Es conveniente crear un espacio personal donde experimentar con paquetes ROS o crear paquetes propios.

Aunque las librerías principales de ROS suelen instalarse en carpetas bien conocidas del sistema operativo mediante gestores de paquetes (como APT), puede que para experimentar con un paquete ROS no deseemos incluirlo en la instalación base, pero si queremos que tanto los recursos de la instalación base como el paquete que estamos probando tengan visibilidad mutua de modo que se puedan resolver sus dependencias. Por eso algunos paquetes ROS pueden estar dispersos en carpetas seleccionadas de modo arbitrario y sin embargo disponibles mediante la inclusión de su ruta en la variable de entorno *ROS_PACKAGE_PATH*.

Mediante la manipulación de esta variable y la variable *ROS_WORKSPACE* es posible personalizar la instalación dando visibilidad a paquetes propios creados o instalados por el usuario o incluso reemplazar paquetes de la instalación base con versiones experimentales. El entorno así obtenido, que difiere de nuestra instalación base ROS, se conoce con el nombre de **espacio de trabajo *Overlay***.

De este modo en un mismo sistema se puede trabajar con diferentes *overlays* de manera simultánea. Sin embargo la gestión manual de las diferentes configuraciones puede ser compleja.

El espacio de trabajo y el procedimiento de creación difiere según qué versión de ROS se utilice.

Creación de un espacio de trabajo en ROS Fuerte mediante *rosws*

La herramienta *rosws* proporciona un interfaz unificado para la obtención de paquetes ROS mediante herramientas de control de revisiones software como SVN, Git o Mercurial, así como para la gestión de paquetes ROS instalados en el sistema. Para utilizarla es necesario haber instalado *rosinstall* (indicado anteriormente en las instrucciones de instalación).

Para crear un *overlay* en nuestro *home* y suponiendo que la instalación de ROS se ha realizado en la carpeta por defecto `/opt/ros/ fuerte` usaremos el siguiente comando:

```
rosws init ~/fuerte_workspace /opt/ros/fuerte
```

El comando anterior buscará un archivo *rosinstall* en `/opt/ros/fuerte` y creará la carpeta `~/fuerte_workspace` con los siguientes archivos:

```
~/fuerte_workspace/.rosinstall  
~/fuerte_workspace/setup.bash  
~/fuerte_workspace/setup.sh  
~/fuerte_workspace/setup.zsh
```

De modo que cuando queramos establecer el entorno ROS correspondiente a esta configuración concreta, ejecutaremos el comando:

```
source ~/fuerte_workspace/setup.bash
```

Para añadir a este *overlay* un paquete ROS llamado *taller_youbot* disponible en un repositorio Git en la url `https://github.com/fsotosan/taller_youbot`:

```
rosws set taller_youbot --git https://github.com/fsotosan/taller_youbot.git  
source ~/fuerte_workspace/setup.bash  
rosws update
```

Documentación de Referencia **rosws**:

<https://github.com/vcstools/rosinstall/blob/master/doc/rosws.rst>

Creación de un espacio de trabajo en ROS Groovy mediante *catkin*

Un espacio de trabajo *catkin* es una carpeta para modificar, compilar o instalar paquetes *catkin*. Puede contener hasta cuatro espacios diferentes (fuentes, compilación, desarrollo e instalación), cada uno con una utilidad distinta en el proceso de desarrollo. Suele tener la siguiente estructura:

```

workspace folder/          -- WORKSPACE
  src/                    -- SOURCE SPACE
    CMakeLists.txt        -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CMakeLists.txt
      package.xml
      ...
  build/                  -- BUILD SPACE
    CATKIN_IGNORE         -- Keeps catkin from walking this directory
  devel/                  -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/                -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...

```

Espacio de Fuentes (src)

Contiene el código fuente de los paquetes catkin. Es donde se realiza el check-out de los paquetes. En su carpeta raíz contiene un enlace simbólico al archivo CMakeLists.txt creado con la ejecución del comando *catkin_init_workspace*.

Espacio de Compilación (build)

Desde aquí se invoca al comando CMake para realizar la compilación de los archivos del espacio de fuentes.

Espacio de Desarrollo (dev)

En este espacio se ubican los resultados de la compilación previamente a ser instalados, permitiendo realizar pruebas y modificaciones sin necesidad de instalar. Indicado en *CATKIN_DEVEL_PREFIX* apunta por defecto a *<build space>/develspace*.

Espacio de Instalación (install)

Destino de los archivos de instalación. Indicado en *CMAKE_INSTALL_PREFIX* apunta por defecto a */usr/local*, por lo que debe ser modificado para evitar problemas de desinstalación y conflictos entre distintas versiones de ROS.

Creación de un espacio de trabajo catkin

En primer lugar creamos la carpeta raíz y una subcarpeta de fuentes *src* e inicializamos mediante la utilidad *catkin_init_workspace*:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin init workspace
```

Aún sin haber añadido ningún archivo fuente ya podemos llamar a la herramienta de compilación. Nos creará los espacios de compilación y desarrollo en sus ubicaciones por defecto, así como los archivos de configuración *setup.*sh*.

```
cd ~/catkin_ws/
catkin make
```

Para cambiar a este nuevo espacio u *overlay* ejecutaremos

```
source devel/setup.bash
```

5.5.3. NAVEGACIÓN POR EL SISTEMA DE FICHEROS ROS

Las utilidades ROS están distribuidas en diversos paquetes. Esto no resulta cómodo a la hora de acceder a ellas usando comandos *bash* (como *ls*, *cd*, etc).

ROS dispone de algunas facilidades para no perderse dentro del sistema de archivos del sistema operativo: **roscpp** y **rospack**

Comando	Ejemplo	Descripción
<code>rospack find <package_name></code>	<code>rospack find roscpp</code>	devuelve la ruta del paquete llamado <code>package_name</code> o informa de que no está instalado
<code>roscd <locationname/><subdir>></code>	<code>roscd roscpp/cmake</code>	equivale al comando <code>cd</code> de <i>bash</i> con la ventaja de que utiliza las variables de entorno de ROS para referenciar los paquetes por su nombre. De este modo no es necesario conocer la ruta exacta de un paquete ROS para acceder a su contenido
<code>rosls <locationname/><subdir>></code>	<code>rosls roscpp_tutorials</code>	de manera análoga a <code>roscd</code> , <code>rosls</code> equivale a <code>ls</code> de <i>bash</i> con la capacidad de referenciar por nombre
<code>rosed <package_name> <file_name></code>	<code>rosed roscpp_tutorials add_two_ints_server.cpp</code>	permite editar un fichero de un paquete referenciado por su nombre

roscd, *rospd*, *rosls*, *rosmake*, *rostrun*, *roscd*, *roscp*, *roslaunch*, *roscupdate*, *roscbag*, *roscparam*, *roscnode*, *rostopic*, *roscservice* o *roscmsg*.

5.5.4. CREACIÓN DE PAQUETES ROS

El proceso de creación de un paquete ROS difiere en Fuerte y Groovy. El primero usa **roscbuild** y el segundo **catkin**.

En adelante se asumirá que nos encontramos dentro de un espacio de trabajo ROS.

Creación de paquetes con roscbuild (ROS Fuerte)

La estructura básica mínima de éste tipo de paquete ROS consiste en una carpeta con el mismo nombre que el paquete y los siguientes archivos:

Makefile	<p>Los makefiles son los ficheros de texto que utiliza make para llevar la gestión de la compilación de programas.</p> <p>Están ordenados en forma de reglas, especificando qué es lo que hay que hacer para obtener un módulo en concreto.</p> <p>Puesto que ROS utiliza CMake, el Makefile de un nuevo paquete normalmente consistirá en una línea con una instrucción: include \$(shell roscpack find mk)/cmake.mk.</p> <p>- Documentación de referencia de Makefiles ROS: Makefile</p>
CMakeLists.txt	<p>Este fichero contiene las instrucciones de compilación para la herramienta CMake, tales como la generación de ejecutables, así como la ubicación de librerías.</p> <p>Es en este fichero donde se especifican además las opciones específicas ROS.</p> <p>- La documentación específica de instrucciones ROS para CMakeLists.txt se puede consultar en roscbuild/CMakeLists</p> <p>- Documentación de referencia de CMake: CMake Tutorial</p>
manifest.xml	<p>Contiene una especificación mínima del paquete, y es utilizada por varias herramientas ROS de compilación, documentación y distribución.</p> <p>En él se indican, por ejemplo, las dependencias de un paquete. Debe respetar el formato XML</p> <p>- Documentación de referencia: Manifest</p>

Los paquetes ROS suelen contener además otros recursos:

bin/	ejecutables producto de la compilación
include/	cabeceras de inclusión C++ (importante indicar <i>export</i> en el Manifest)

msg/	ROS <i>messages</i>
src/	Archivos de código fuente.
srv/	ROS <i>services</i>
scripts/	<i>scripts</i> ejecutables
mainpage.dox	documentación (ver referencia ROS de Doxygen)

El comando **roscreeate-pkg** crea la estructura de ficheros básica anterior. Basta con indicar el nombre del nuevo paquete y, opcionalmente, el nombre de aquellos paquetes ROS de los que depende:

```
roscreeate-pkg <package name> <depend1> <depend2> <depend3>
```

A partir de este momento, nuestro paquete será accesible por las utilidades de ROS (*roscd*, etc.)

El siguiente paso será editar los archivos *manifest.xml* y *CMakeLists.txt* de acuerdo a las necesidades específicas de nuestro proyecto.

Compilación de paquetes con ROS Fuerte

La compilación en Fuerte se realiza con la herramienta *rosmake*, similar a *make* pero específica para ROS, de modo que se encarga de la compilación de el paquete y sus dependencias en el orden adecuado.

```
rosmake <package>
```

Creación de paquetes con catkin (ROS Groovy)

Un paquete catkin debe cumplir los siguientes requisitos:

- Un archivo *package.xml*.
- Un archivo *CMakeLists.txt*.
- La carpeta contenedora solo admite un paquete.

En el caso más sencillo:

```
my_package/  
  CMakeLists.txt  
  package.xml
```

Se recomienda que los paquetes se ubiquen dentro del espacio de fuentes del espacio de trabajo:

```
workspace folder/      -- WORKSPACE
src/                   -- SOURCE SPACE
  CMakeLists.txt       -- 'Toplevel' CMake file, provided by catkin
  package_1/
    CMakeLists.txt     -- CMakeLists.txt file for package_1
    package.xml        -- Package manifest for package_1
    ...
  package_n/
    CMakeLists.txt     -- CMakeLists.txt file for package_n
    package.xml        -- Package manifest for package_n
```

Para crear un paquete catkin nos situamos en el espacio de fuentes (carpeta *src*) de nuestro espacio de trabajo catkin y utilizamos la herramienta *catkin_create_pkg*.

Los argumentos de *catkin_create_pkg* son el nombre del paquete y la lista de paquetes de los que depende. Ejemplo:

```
cd ~/catkin_ws/src
catkin_create_pkg <package_name> <depend1> <depend2> ...
```

El siguiente paso será editar los ficheros *package.xml* y *CMakeLists.txt* de acuerdo a las necesidades específicas de nuestro proyecto.

El archivo *package.xml*

El archivo *package.xml* debe respetar el formato XML y como mínimo debe contener las siguientes entradas:

1. `<name>` - Nombre del paquete
2. `<version>` - Versión del paquete (3 números enteros separados por puntos)
3. `<description>` - Descripción del contenido del paquete
4. `<maintainer>` - Nombre de la persona encargada de mantener el paquete
5. `<license>` - La licencia bajo la que se distribuye el paquete.

Además, la utilidad *catkin_create_pkg* habrá rellenado las dependencias del paquete de acuerdo a lo indicado en los argumentos.

1. Dependencias Build Tool: Herramientas que el paquete necesita para realizar el proceso de compilación. Normalmente será únicamente *catkin*, salvo que se trabaje con compilación cruzada
2. Dependencias Build: Paquetes requeridos para poder compilar este paquete
3. Dependencias Run: Paquetes requeridos para poder ejecutar este paquete
4. Dependencias Test: Solamente aquellas dependencias adicionales necesarias para unit testing

Ejemplo:

```
<package>
  <name>mipaquete</name>
  <version>1.2.4</version>
  <description>
    Esto es un paquete de prueba.
  </description>
  <maintainer email="a@b.c">Nombre de quien mantiene el paquete</maintainer>
  <license>BSD</license>

  <url>http://ros.org/wiki/foo_core</url>
  <author>Nombre de quien ha programado el paquete</author>

  <buildtool depend>catkin</buildtool depend>

  <build_depend>message_generation</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>std_msgs</build_depend>

  <run_depend>message_runtime</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
</package>
```

El archivo CMakeLists.txt

Indica a la utilidad CMake cómo compilar y cómo instalar un paquete. Debe contener el siguiente formato:

- Versión mínima de catkin (*cmake_minimum_required*)
- Nombre del paquete (*project()*)
- Búsqueda de paquetes CMake/Catkin necesarios para compilar (*find_package()*)
- Generadores de Message/Service/Action ROS (*add_message_files()*, *add_service_files()*, *add_action_files()*)
- Invocaciones a Message/Service/Action ROS (*generate_messages()*)
- Indicaciones *export* para compilación (*catkin_package()*)
- Producto de la compilación (librerías o ejecutables) (*add_library()/add_executable()/target_link_libraries()*)
- Tests (*catkin_add_gtest()*)
- Reglas de instalación (*install()*)

Ejemplo:

```
cmake_minimum_required(VERSION 2.8.3)

project(nombredelpaquete)

find_package(catkin REQUIRED)

catkin_package(
  INCLUDE_DIRS include
  LIBRARIES ${PROJECT_NAME}
  CATKIN_DEPENDS roscpp nodelet
  DEPENDS eigen opencv)

set_target_properties(rviz_image_view
  PROPERTIES OUTPUT_NAME image view)
```

```
        PREFIX "")
include_directories(include ${Boost INCLUDE_DIRS} ${catkin INCLUDE_DIRS})
link_directories(~/my_libs)
add_executable(myProgram src/main.cpp src/some_file.cpp src/another_file.cpp)
add_library(${PROJECT_NAME} ${${PROJECT_NAME}_SRCS})
```

Compilación de paquetes catkin en ROS Groovy

Una vez añadidos los archivos fuente del paquete en la carpeta correspondiente (por ejemplo `~/catkin_ws/src/nombredelpaquete/src`) se edita el archivo `CMakeLists.txt` del espacio de trabajo y se llama a `catkin_make`

```
cd ~/catkin_ws
catkin_make install
```

Esto generará un espacio de instalación (`~/catkin_ws/install`) con los archivos de configuración correspondientes (`setup.*sh`) que permiten cambiar a este espacio.

5.6. El Formato URDF de descripción de robots

El formato URDF (Unified Robot Description Format) es un formato XML para describir robots. Consta de una serie de especificaciones para definir propiedades cinemáticas y dinámicas de eslabones y uniones, así como características de algunos sensores.

El formato supone que el robot está compuesto por eslabones rígidos unidos mediante articulaciones, y que se puede definir como una estructura de árbol. Esto lo hace incompatible con robots flexibles y robots paralelos.

5.6.1. ELEMENTO <ROBOT>

Es el nodo raíz de una descripción URDF. Admite el atributo **name**.

```
<robot name="mimodeloderobot">
  ...
</robot>
```

5.6.2. ELEMENTO <LINK>

Eslabón. Admite el atributo **name** y representaciones visual, inercial y de colisión. Esta separación permite dar una imagen realista y detallada del modelo utilizando por ejemplo una definición STL como representación visual, y a la vez utilizar representaciones inerciales y de colisión sencillas para aligerar la carga computacional de la simulación.

Elemento <inertial>

Propiedades inerciales del eslabón, descritas mediante los elementos <origin>, <mass> e <inertia>

- <origin>: *Pose* del sistema de referencia inercial con respecto al sistema de referencia del eslabón. Debe corresponder con el centro de gravedad. Los ejes del sistema de referencia inercial no tienen por qué coincidir con los ejes de inercia principales. Admite los atributos **xyz** (traslación relativa al sistema de referencia del eslabón) y **rpy** (rotación relativa al sistema de referencia del eslabón especificada con los ángulos *roll*, *pitch* y *yaw* en radianes).
- <mass>: Masa del eslabón indicada en el atributo **value**.
- <inertia>: Matriz de inercia rotacional. Solamente 6 componentes: **Ixx**, **Ixy**, **Ixz**, **Iyy**, **Iyz** e **Izz** por ser la matriz simétrica.

Elemento <visual>

Propiedades visuales del eslabón.

- **<origin>**: *Pose* del sistema de referencia inercial con respecto al sistema de referencia del eslabón. (ver descripción de <origin> en elemento <visual>).
- **<geometry>**: Forma del eslabón. Pueden estar definida como una de las formas geométrica simples disponibles como **<box>** (definida por 3 magnitudes x, y, z), **<cylinder>** (atributos **radius** y **length**) y **<sphere>** (atributo **radius**) o mediante un fichero local indicado como elemento **<mesh>** (atributo **filename**)
- **<material>**: Especificación de color y textura. Definido por los elementos opcionales **<color>** (atributo **rgba**: 4 números en el rango [0,1] con los valores de RGB y Alpha) y **<texture>** (archivo indicado en atributo **filename**). Si se ha definido previamente puede reutilizarse referenciando el material mediante su atributo **name**.

Elemento <collision>

Geometría de colisión del eslabón.

- **<origin>**: *Pose* del sistema de referencia de colisión con respecto al sistema de referencia del eslabón. (ver descripción de <origin> en elemento <inertial>).
- **<geometry>**: Forma de la geometría de colisión (ver descripción de geometría en elemento <visual>).

Ejemplo

```
<link name="my link">
  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100" />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="1 1 1" />
    </geometry>
    <material name="Cyan">
      <color rgba="0 255 255 1.0"/>
    </material>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>
</link>
```

5.6.3. ELEMENTO <JOINT>

Describe las características cinemáticas y dinámicas de una unión, así como sus límites de seguridad.

Admite los atributos obligatorios *name* y *type*. Tipos admitidos:

- **revolute**: permite rotación alrededor de un eje con límites inferior y superior (expresados en radianes).
- **continuous**: permite rotación sin límites alrededor de un eje.
- **prismatic**: permite desplazamiento a lo largo de un eje con límites inferior y superior (expresados en metros).
- **fixed**: unión fija. No hay grados de libertad.
- **floating**: flotante (6 grados de libertad).
- **planar**: permite desplazamiento sobre un plano perpendicular a un eje.

Elementos

- **<origin>**: Transformación desde el sistema de referencia padre al hijo. Atributos de traslación **xyz** y rotación **rpy**.
- **<parent>**: Nombre del eslabón padre.
- **<child>**: Nombre del eslabón hijo.
- **<axis>**: Eje de referencia. Atributo **xyz** (vector normalizado, por defecto (1,0,0)). No aplica para uniones fija y flotante.
- **<calibration>**: Posiciones de referencia de la unión para calibrar su posición absoluta. Atributos **rising** y **falling** para disparar flancos ascendente y descendente respectivamente ante un desplazamiento positivo de la unión.
- **<dynamics>**: Propiedades físicas de la unión: amortiguamiento (expresado en N*s/m en juntas prismáticas y en N*s/rad en juntas de revolución) y fricción estática (expresada en N en juntas prismáticas y en N*m en juntas de revolución).
- **<limit>**: Límites (para prismáticas y de revolución). Atributos **lower** y **upper** (opcionales y expresados en radianes o metros según tipo), **effort** (obligatorio, máximo esfuerzo soportado) y **velocity** (obligatorio, máxima velocidad permitida)
- **<mimic>**: *Nueva en Groovy*. Permite especificar que la unión imita a otra unión especificada por nombre en el atributo obligatorio **joint**. Permite modificar el comportamiento de la unión con respecto a la unión de referencia con los factores **multiplier** (por defecto 1) y **offset** (por defecto 0) según la expresión $value = multiplier * other_joint_value + offset$.
- **<safety_controller>**: Opcional. Puede contener los siguientes atributos: **soft_lower_limit** (opcional, 0 por defecto, ha de ser mayor que el límite inferior definido en el elemento limit de la unión), **soft_upper_limit** (opcional, 0 por defecto, ha de ser menor que el límite superior definido en el elemento limit de la unión), **k_position** (opcional, 0 por defecto, especifica la relación entre los límites de posición y velocidad), **k_velocity** (obligatorio, especifica la relación entre los límites de esfuerzo y

velocidad. Documentación de referencia sobre límites de seguridad: Joint Safety Limits Explained

```
<joint name="my joint" type="floating">
  <origin xyz="0 0 1" rpy="0 0 3.1416"/>
  <parent link="link1"/>
  <child link="link2"/>

  <calibration rising="0.0"/>
  <dynamics damping="0.0" friction="0.0"/>
  <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
  <safety_controller k_velocity="10" k_position="15" soft_lower_limit="-2.0"
soft upper limit="0.5" />
</joint>
```

5.6.4. ELEMENTO <SENSOR>

Define las propiedades de algunos sensores visuales. Admite el atributo obligatorio **name** y el opcional **update_rate** (frecuencia de recepción de datos expresada en Hz)

Elementos

- **<parent>**: Obligatorio. Nombre del eslabón al que está asociado el sensor
- **<origin>**: Opcional. *Pose* del sistema de referencia óptico del sensor respecto del sistema de referencia del eslabón padre. Por convenio z adelante, x derecha, y abajo. Atributos **xyz** y **rpy**.
- **<camera>**: Opcional. Sin atributos. Contiene elemento obligatorio **<image>** con atributos obligatorios **width** y **height** (pixels), **format** (definido en image_encodings.h), **hfov** (campo de visión horizontal en radianes), **near** (distancia mínima de visión en metros) y **far** (distancia máxima de visión en metros)
- **<ray>**: Opcional. Sin atributos. Contiene elementos opcionales **<horizontal>** y **<vertical>**, ambos con atributos opcionales **samples** (número de rayos simulados por ciclo de barrido), **resolution** (1 por defecto. Interpolación si menor a 1. Promedio si mayor que 1), **min_angle** (radianes) y **max_angle** (radianes)

Ejemplo Cámara

```
<sensor name="my_camera_sensor" update_rate="20">
  <parent link="optical frame link name"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <camera>
    <image width="640" height="480" hfov="1.5708" format="RGB8" near="0.01" far="50.0"/>
  </camera>
</sensor>
```

Ejemplo Láser scan

```
<sensor name="my ray sensor" update rate="20">
  <parent link="optical frame link name"/>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <ray>
    <horizontal samples="100" resolution="1" min angle="-1.5708" max angle="1.5708"/>
    <vertical samples="1" resolution="1" min angle="0" max angle="0"/>
  </ray>
</sensor>
```

5.6.5. MODELO URDF SIMPLIFICADO DEL KUKA YOUNBOT

Como ejercicio para seguir los tutoriales de simulación se ha realizado un modelo URDF simplificado del Youbot a partir del procedimiento descrito en el tutorial ROS ***“Building a Movable Robot Model with URDF”***

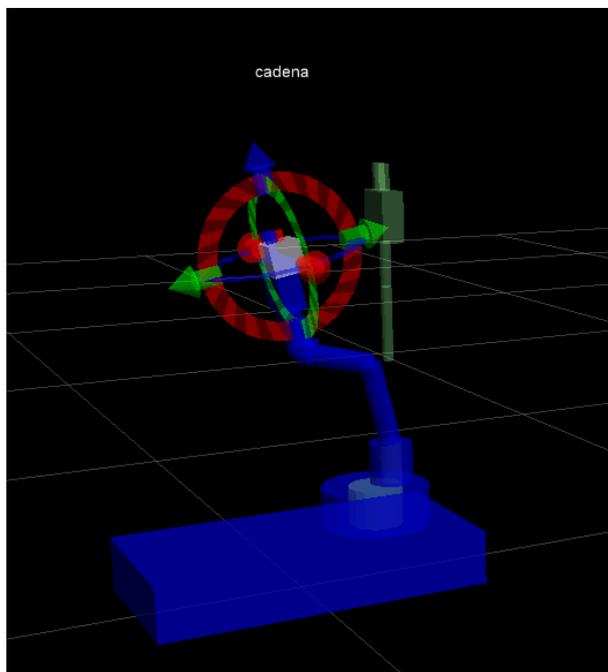


Figura 5.5 - Representación boceto URDF de Youbot

Con este modelo se ha seguido el ***“Planning Description Configuration Wizard”*** obteniendo como resultado un paquete ROS de navegación específico para el modelo introducido. La descripción de los ficheros generados se explica en ***“Understanding and adjusting the auto-generated arm navigation application”***

Una vez creado el paquete de navegación se puede lanzar ***“Planning Components Visualizer”*** y por ejemplo configurar gráficamente las posiciones inicial y final de una cadena cinemática definida en el asistente anterior.

Los tutoriales mencionados en este epógrafe se pueden encontrar todos en [1]. El modelo URDF utilizado se lista en el apartado **6.5**.

5.7. Visión artificial mediante OpenCV

5.7.1. INTRODUCCIÓN

OpenCV es un conjunto de librerías *open source* con algoritmos de visión artificial. A la fecha de redacción de este tutorial la versión estable recomendada es la 2.4.5. por lo que se asume esta versión como referencia para el resto del contenido.

Para su manejo OpenCV proporciona un API C++ estructurada de manera modular :

1. **core**: Estructuras de datos y funciones básicas necesarias para el resto de módulos
2. **imgproc**: Funciones de procesamiento de imagen (filtrado, transformaciones, etc.)
3. **video**: Análisis de video (estimación de movimiento, eliminación de fondos, seguimiento de objetos, etc.)
4. **calib3d**: Operaciones relacionadas con imagen 3D (estimación de *pose*, calibración de cámaras, reconstrucción 3D, etc.)
5. **features2d**: Detección de bordes, esquinas, etc.
6. **objdetect**: Detección de objetos predefinidos (caras, ojos, etc.)
7. **highgui**: Interfaz de captura de video, codecs y funcionalidad básica para interfaz de usuario
8. **gpu**: Algoritmos de aceleración mediante GPU
9. otros

Documentación de Referencia OpenCV: <http://docs.opencv.org/index.html>

5.7.2. INSTALACIÓN

Es posible que ya tengamos instalado OpenCV en nuestro sistema. Si trabajamos con ROS, omitiremos este paso (deberíamos ver las carpetas *opencv* y *opencv2* en la carpeta de includes de la instalación ROS */opt/ros/ fuerte/include*). De no ser así, al final de este capítulo se indica el procedimiento de instalación y uso de OpenCV en ROS.

Para instalar OpenCV sin ROS ejecutaremos los siguientes comandos:

```
sudo apt-get install build-essential
wget https://github.com/Itseez/opencv/archive/2.4.5.tar.gz
tar -xvf 2.4.5.tar.gz
cd opencv-2.4.5
mkdir release
cd release
cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install
```

5.7.3. CREACIÓN DE UN PROGRAMA CON OPENCV

El siguiente programa utiliza la API de OpenCV para generar una imagen y mostrarla.

```
#include <cv.h>
#include <highgui.h>

using namespace cv;

int main ( int argc, char **argv )
{
    cvNamedWindow( "My Window", 1 );
    IplImage *img = cvCreateImage( cvSize( 640, 480 ), IPL_DEPTH_8U, 1 );
    CvFont font;
    double hScale = 1.0;
    double vScale = 1.0;
    int lineWidth = 1;
    cvInitFont( &font, CV_FONT_HERSHEY_SIMPLEX | CV_FONT_ITALIC,
                hScale, vScale, 0, lineWidth );
    cvPutText( img, "Hello World!", cvPoint( 200, 400 ), &font,
               cvScalar( 255, 255, 0 ) );
    cvShowImage( "My Window", img );
    cvWaitKey();
    return 0;
}
```

5.7.4. PREPARACIÓN DEL ENTORNO DE COMPILACIÓN CON CMAKE

Crearemos una carpeta para el ejemplo (*test_opencv*) y una subcarpeta *src* y salvaremos el programa como *test_opencv/src/helloworld.cpp*.

A continuación creamos el archivo *test_opencv/CMakeLists.txt* con el siguiente contenido:

```
cmake_minimum_required(VERSION 2.8)
PROJECT( helloworld proj )
FIND_PACKAGE( OpenCV REQUIRED )
ADD_EXECUTABLE( helloworld src/helloworld.cpp )
TARGET_LINK_LIBRARIES( helloworld ${OpenCV_LIBS} )
```

Y compilamos desde la carpeta raíz del ejemplo:

```
cd test_opencv
cmake .
make
```

Obteniendo el ejecutable *helloworld*. que muestra una ventana con fondo negro con la frase “Hello World!” escrita en la parte inferior,

5.7.5. PROGRAMACIÓN OPENCV DESDE IDE ECLIPSE CDT

En el siguiente enlace se explica detalladamente el proceso de creación de un proyecto Eclipse para trabajar con OpenCV: [Using OpenCV with Eclipse mediante dos métodos.](#)

A continuación se indica cómo adaptar el ejemplo anterior a Eclipse:

- Desde la carpeta raíz del ejemplo (test_opencv) ejecutamos ***cmake-gui***.
- Seleccionamos la carpeta test_opencv en los campos "Where is the source code" y "Where to build the binaries".

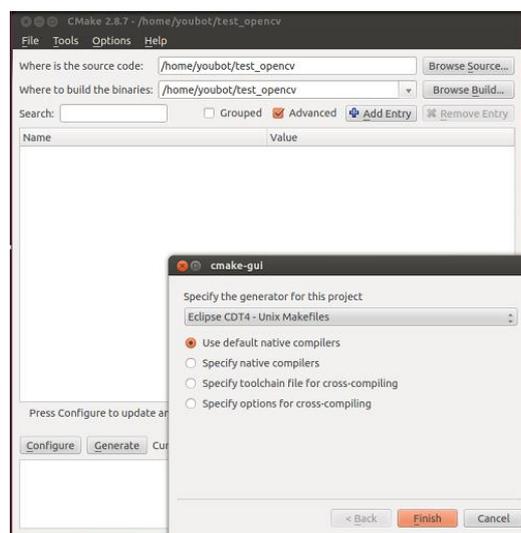


Figura 5.6 - Configuración cmake-gui (I)

- Pulsamos *Configure* (opción "Eclipse CDT4 - Unix Makefiles" cuando nos pregunte). Veremos una lista de parámetros en rojo y comprobaremos que hay rutas específicas para OpenCV.

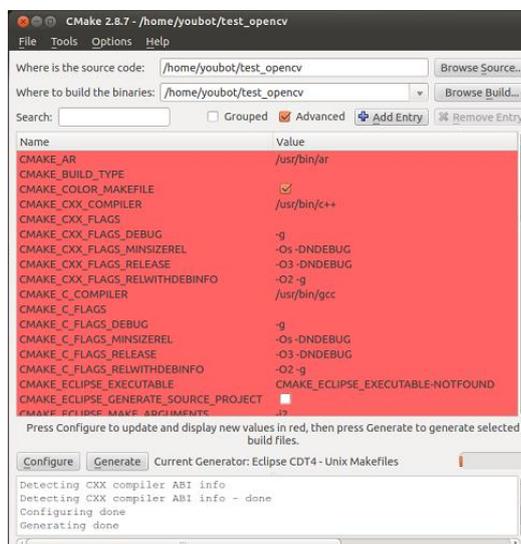


Figura 5.7 - Configuración cmake-gui (II)

- Pulsamos *Generate* y salimos de *cmake-gui*.
- Ejecutamos **make** desde *test_opencv* comprobando que la compilación se realiza correctamente y se produce un ejecutable.

```
youbot@ubuntu:~/test_opencv$ cmake-gui
youbot@ubuntu:~/test_opencv$ make
Scanning dependencies of target helloworld
[100%] Building CXX object CMakeFiles/helloworld.dir/src/helloworld.cpp.o
Linking CXX executable helloworld
[100%] Built target helloworld
youbot@ubuntu:~/test_opencv$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  CMakeLists.txt  helloworld  Makefile  src
youbot@ubuntu:~/test_opencv$
```

Figura 5.8 – Compilación proyecto *cmake*

- Abrimos **eclipse** e importamos el proyecto mediante *Import->C/C++->Existing Code as a Makefile Project* y seleccionamos "Linux GCC" en *Toolchain for Indexer Settings*. Pulsamos *Finish*.

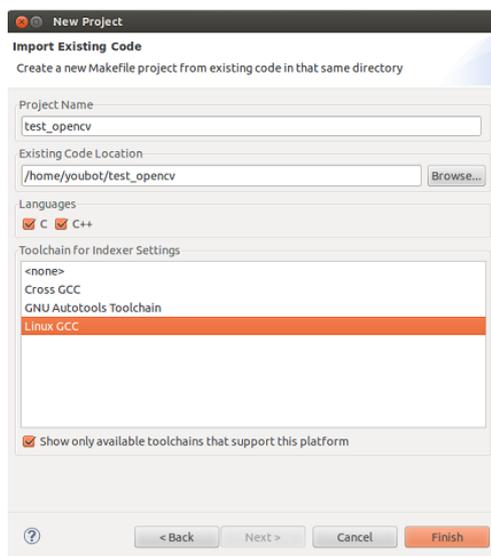


Figura 5.9 – Importación de proyecto desde Eclipse CDT

- Ahora ya podemos compilar mediante *Project->Build All*. Sin embargo cuando abrimos el código fuente veremos errores de resolución. Para solucionarlo añadiremos a la lista de *Includes* de compilación C/C++ (en propiedades de proyecto) la ruta correspondiente según nuestra instalación de OpenCV (podría ser */usr/local/include/opencv* si hemos instalado siguiendo este tutorial o por ejemplo */opt/ros/ fuerte/include/opencv* si lo hemos hecho indirectamente a través de ROS; en cualquier caso debemos ser consistentes con las rutas mostradas en el asistente *cmake-gui*).

5.7.6. CREACIÓN DE UN PAQUETE ROS PARA PROGRAMACIÓN OPENCV

Con ROS podemos empezar a utilizar OpenCV de manera más sencilla. Simplemente indicando como dependencias los paquetes *cv_bridge* (para pasar de imágenes ROS a OpenCV y viceversa) y, por supuesto, *opencv2*. Es conveniente incluir además algunos otros que seguramente vayamos a utilizar: *sensor_msgs*, *std_msgs* e *image_transport*.

La instalación de OpenCV desde ROS es inmediata

```
sudo apt-get install ros-fuerte-opencv2
```

A continuación se resume el proceso de creación de un paquete ROS para programar con OpenCV con una sencilla aplicación: identificar una bola de color rojo dentro de una imagen de prueba.

Para ello se realizarán los siguientes pasos:

1. **Filtrar** la imagen de manera que sólo permanezcan los elementos con componente roja
2. **Erosionar** la imagen filtrada para eliminar objetos pequeños con componente roja
3. **Dilatar** la imagen erosionada para resaltar los objetos que han sobrevivido a los pasos anteriores.

En cuanto al filtro, se utilizará representación HSV (*Hue, Saturation and Value*) en vez de BGR que es la representación por defecto en OpenCV.

```
roscmake-pkg [nombre_paquete] cv_bridge opencv2 sensor_msgs std_msgs image_transport
```

Una vez creado añadiremos la ruta del paquete a la variable de entorno `ROS_PACKAGE_PATH`, una carpeta `src` y un fichero de código fuente (*src/nombre_fuente.cpp*) con el siguiente contenido:

```
#include <stdio.h>
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <ros/ros.h>

using namespace cv;
using namespace std;

char thePicPath[] = "./pic.jpg";

Mat redBallFilter(const Mat& src) {
    Scalar HSV_ROJO_MIN(0,80,100);
    Scalar HSV_ROJO_MAX(8,255,200);

    Mat hsv, redBallOnly;

    // Comprobamos que el formato de imagen es el esperado
    assert(src.type() == CV_8UC3);
```

```
// Obtenemos la representación HSV de la imagen BGR (ojo, BGR y no RGB!!)
cvtColor(src, hsv, CV_BGR2HSV);

// Filtrado en HSV (en OpenCV 0<=H<=180, 0<=S<=255, 0<=V<=255)
inRange(hsv, HSV_ROJO_MIN, HSV_ROJO_MAX, redBallOnly);

// Definimos kernel para operación de dilatación
Mat element = getStructuringElement(MORPH_ELLIPSE, Size(25, 25));

// Erosión
erode(redBallOnly, redBallOnly, Mat());

// Dilatación
dilate(redBallOnly, redBallOnly, element);

return redBallOnly;
}

int main( int argc, char** argv ) {

    // Leemos una imagen
    Mat input = imread(thePicPath);

    // La mostramos por pantalla
    imshow("original", input);

    // Esperamos pulsación de tecla
    waitKey();

    // Aplicamos filtro de búsqueda de bolas rojas
    Mat redOnly = redBallFilter(input);

    // Mostramos resultado filtrado
    imshow("pelota roja", redOnly);

    // Esperamos pulsación de tecla
    waitKey();

    return 0;
}
```

A continuación editaríamos el archivo *CMakeLists.txt* para indicar la ruta al archivo fuente y el nombre del ejecutable

```
rosbuild_add_executable(nombre_ejecutable src/nombre_fuente.cpp)
```

Y finalmente compilamos

```
rosmake nombre_paquete
```

obteniendo en la carpeta bin un ejecutable llamado *nombre_ejecutable*. A continuación se muestra el resultado:



Figura 5.10 – Identificación de objeto de color rojo con OpenCV

5.8. Comunicación Serie en Linux

5.8.1. INTRODUCCIÓN

Entenderemos por comunicación serie el proceso de envío de datos de manera secuencial, bit por bit, sobre un canal de comunicación.

Entre las tecnologías más utilizadas de comunicación serie en el ámbito de la informática doméstica e industrial se encuentran: RS-232, RS-423, RS-485, USB, Firewire, Ethernet, SCSI o SATA. En el ámbito de la electrónica son habituales las tecnologías de I2C, SPI. Y en telecomunicaciones *Fibre Channel*, *SDH* o *SONET*. Estas tecnologías difieren en especificaciones y uso, pero todas tienen una característica en común: los bits se transmiten de uno en uno.

A continuación se introducen algunos conceptos relevantes para el tipo de comunicación utilizado en este trabajo.

5.8.2. EL ESTÁNDAR RS-232

Las comunicaciones seriales en los PCs han utilizado tradicionalmente el estandar RS-232. Según esta especificación, los dispositivos se clasifican en DCE (Data Communications Equipment) y DTE (Data Terminal Equipment). Normalmente el DTE era un PC y el DCE podía ser una impresora, un modem, u otro dispositivo.

Las características eléctricas se definen en la especificación RS232C:

1. Marca (cero lógico): entre +3V y +25V.
2. Espacio (uno lógico): entre -3V y -25V.
3. Región entre -3V y +3V no utilizada.
4. La tensión en circuito abierto no debe superar 25V respecto a GND.
5. La corriente en cortocircuito no debe superar 500mA. El *driver* debería soportar esta condición sin sufrir daños.

Aunque en la actualidad los PCs han dejado de incorporar el "puerto serie" (normalmente un conector DB9 situado en la parte posterior del PC), son muchos los dispositivos que usan este tipo de conexión para cubrir distancias cortas (hasta 15 metros) y bajas velocidades.

Permite comunicación *simplex* (unidireccional), *full duplex* (bidireccional simultánea) o *half duplex* (bidireccional no simultánea).

Las señales más habituales definidas por RS-232 son las siguientes:

Señal	Función	Pin (DB9)
G	Masa común.	5
TD	Transmisión	3
RD	Recepción.	2
DTR	<i>Data Terminal Ready</i> . UART lista para conectar.	4
DSR	<i>Data Set Ready</i> . Modem listo para conectar.	6
RTS	<i>Request to Send</i> . UART lista para intercambio de datos.	7
CTS	<i>Clear to Send</i> . Modem listo para intercambio de datos.	8
DCD	<i>Data Carrier Detect</i> . Modem detecta portadora (desde la red telefónica)	1
RI	<i>Ring Indicator</i> . Modem está recibiendo tono de llamada (desde la red telefónica).	9

Aunque en la práctica no siempre se utilizan todas. Es muy habitual utilizar solamente 3 señales: G, TD y TR, prescindiendo de la capacidad de control de flujo mediante *hardware*, especialmente en aplicaciones con microcontrolador.

Normalmente en estos casos la comunicación se realiza conectando TD y TR a las señales RX y TX de una UART, a través de un *driver* de adaptación de voltajes MAX232). Con estas 3 señales es aún posible establecer un control de flujo *software* mediante el envío del byte Xoff (ASCII 19) para indicar a la otra parte que deje de enviar y Xon (ASCII 17) para indicar a la otra parte que puede reanudar los envíos. Esto es útil cuando alguna de las partes tiene limitaciones en la velocidad de procesamiento de los datos que recibe.

5.8.3. UART

Una UART (Universal Asynchronous Receiver / Transmitter) es un dispositivo encargado de gestionar comunicaciones seriales. Básicamente se encarga de serializar bytes locales (recibidos en paralelo mediante 8 señales, una por bit) para enviarlos por la salida TD serie empezando por el menos significativo, deserializar bits recibidos por la entrada RD serie entregándolos localmente en paralelo mediante 8 (una por bit), y gestionar la señalización y control de flujo.

Si las señales de la UART tienen niveles TTL es necesaria una adaptación al medio físico utilizado (por ejemplo MAX232). Algunas UART con buses de transmisión y recepción separados, soportan directamente varios tipos de niveles lógicos.

Los parámetros de configuración de la UART relevantes son los siguientes:

- Baudrate (velocidad). Velocidad de línea. Valores habituales (en bps): 1200, 2400, 4800, 9600, 28800, 57600, 115200 ...
- Bits de datos. Valores posibles: 5, 7 y 8. Indica el número de bits de información útil que viaja en cada trama.
- Bits de parada. Valores posibles: 1, 1.5 y 2. Indica el número de bits de indicación de fin de trama (unos lógicos). No se especifica bits de inicio ya que siempre se utiliza un solo bit (cero lógico)
- Paridad: Par, Impar o Ninguna. Indica si se incluye comprobación de paridad
- Control de flujo: HW, SW o Ninguno. Indica se se realiza control de flujo.

5.8.4. TERMIOS

termios es un API de los sistemas tipo Unix para operaciones de entrada/salida mediante terminal. Uno de sus posibles usos es la gestión de una comunicación a través de un dispositivo serial mediante llamadas genéricas al sistema operativo como *open*, *read* *ywrite*.

Con *termios* es posible seleccionar diferentes configuraciones de comunicación. Las más comunes en cuanto a las operaciones de lectura son:

1. **Con bloqueo:** las lecturas bloquean el flujo del proceso hasta que se reciban datos
2. **Sin bloqueo:** las lecturas devuelven el control inmediatamente con independencia de que haya datos disponibles
3. **Asíncrona:** el sistema operativo notifica la llegada de datos al puerto serie mediante una señal (*SIGIO*) que el proceso puede capturar, interrumpiendo momentáneamente su flujo normal para atender la lectura.

En cuanto a la interpretación de los datos transmitidos, existen diferentes modos:

1. **Modo canónico:** La información se transmite línea por línea y se interpretan caracteres especiales (*termios* fue originalmente concebido para gestión de terminales de texto)
2. **Modo no canónico:** Los caracteres se procesan individualmente sin interpretación. Este es el modo que nos interesa para comunicación entre hardware, ya que permite enviar bytes de cualquier valor (entre 0 y 255) sin que sean confundidos con comandos con significado especial para un terminal.

La API está programada en C y sus métodos son accesibles mediante inclusión de **termios.h**.

5.8.5. COMUNICACIÓN SERIE C++

La comunicación serie se utilizará en diversos ejemplos, principalmente relacionada con dispositivos a integrar en ROS. Puesto que la configuración de *termios* no es trivial, y los programas ROS de este trabajo están desarrollados en C++, se ha considerado conveniente la programación de una clase C++ que se encargue de encapsular los detalles de configuración de *termios* y permita un uso orientado a objetos de la conexión.

En el epígrafe **6.1** se muestra el código fuente de una librería C++ desarrollada en este trabajo para encapsular los detalles de *termios* y simplificar el uso del puerto serie.

La librería consta de 3 clases, definidas en el espacio de nombres **Serial**:

- ***Serial::Serial***. Se encarga de abrir y configurar el dispositivo serie mediante *termios*. Admite los parámetros típicos de configuración serie (velocidad, bits de datos, bits de parada y paridad) y permite seleccionar uno de tres modos de comportamiento en cuanto a lecturas: síncrona con bloqueo, síncrona sin bloqueo, y asíncrona mediante uso de señales (en este caso admite además como parámetro un puntero a una función a desarrollar por el usuario que gestione la recepción de la señal *SIGIO*)
- ***Serial::Queue***. Cola circular
- ***Serial::Serial_Q***. Clase derivada de *Serial::Serial* de configuración simplificada, que fija algunos parámetros (8 bits de datos, 1 bit de parada, sin paridad y comunicación síncrona sin bloqueo), admitiendo para crear la conexión el nombre del dispositivo serie y la velocidad de línea. Incorpora además una cola de entrada (utilizando la clase *Serial::Queue*) que permite almacenar los caracteres recibidos via serie, así como con

El código está comentado exhaustivamente para mostrar el significado de todos los parámetros de configuración utilizados.

5.9. Configuración GPIO y UART en BeagleBone

Para comunicar un sistema embebido como el Beaglebone con un dispositivo contiguo, como puede ser un controlador de motores o una tarjeta de adquisición de datos, existen diversos mecanismos como el bus I2C, SPI, GPIO o la UART.

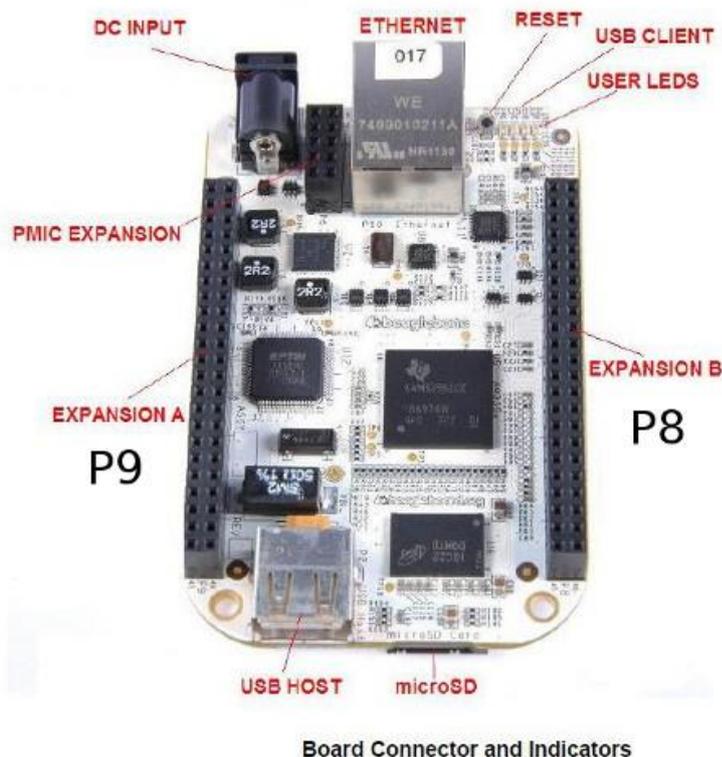


Figura 5.11 - Descripción componentes BeagleBone

En este ejercicio se utilizarán dos UART para simular una comunicación entre el Beaglebone y un dispositivo externo (el propio BeagleBone hará el dispositivo externo mediante la segunda UART).

La distribución de Linux utilizada es Angstrom, pero los procedimientos indicados son igualmente aplicables a otras distribuciones como Ubuntu o ArchLinux.

El beaglebone dispone de 5 UART. La UART1, accesible por defecto desde el sistema operativo con el nombre de `"/dev/ttyO0"`, es la que usamos para conectarnos por consola desde un PC mediante un cable USB (el beaglebone dispone de un chip FTDI de conversión USB a serie).

5.9.1. HABILITACIÓN DE UART 3 Y 5

Consultando la documentación de referencia del BeagleBone localizamos dos UART en el *header* de expansión **P9**: UART5 (pines 11 y 13) y UART3 (pines 21 y 22).

SIGNAL NAME	PIN	CONN	PIN	SIGNAL NAME	
	GND	1	2	GND	
	VDD_3V3EXP	3	4	VDD_3V3EXP	
	VDD_5V	5	6	VDD_5V	
	SYS_5V	7	8	SYS_5V	
PWR_BUTTON*		9	10	A10	SYS_RESETn
UART4_RXD	T17	11	12	U18	GPIO1_28
UART4_TXD	U17	13	14	U14	EHRPWM1A
GPIO1_16	R13	15	16	T14	EHRPWM1B
I2C1_SCL	A16	17	18	B16	I2C1_SDA
I2C2_SCL	D17	19	20	D18	I2C2_SDA
UART2_TXD	B17	21	22	A17	UART2_RXD
GPIO1_17	V14	23	24	D15	UART1_TXD
GPIO3_21	A14	25	26	D16	UART1_RXD
GPIO3_19	C13	27	28	C12	SPI1_CS0
SPI1_D0	B13	29	30	D12	SPI1_D1
SPI1_SCLK	A13	31	32	VDD_ADC(1.8V)	
AIN4	C8	33	34	GNDA_ADC	
AIN6	A5	35	36	A5	AIN5
AIN2	B7	37	38	A7	AIN3
AIN0	B6	39	40	C7	AIN1
CLKOUT2	D14	41	42	C18	GPIO0_7
	GND	43	44	GND	
	GND	45	46	GND	

Figura 5.12 – Distribución de pines en header P9

Aquí encontramos una primera dificultad: podríamos esperar que del mismo modo que accedemos a la UART1 por su nombre de dispositivo `"/dev/ttyO0"`, deberíamos encontrar las UART 3 y 5 en `"/dev/ttyO2"` y `"/dev/ttyO4"` respectivamente. Sin embargo, estos dispositivos no se encuentran disponibles en `"/dev"`.

Esto es debido a que el procesador del BeagleBone dispone de más recursos que pines de entrada/salida, y resuelve el acceso a los recursos multiplexando los pines en 7 modos. Así, un mismo pin puede configurarse para 7 funciones diferentes, según los valores introducidos en un registro del procesador

El modo de cada pin se establece en el arranque de Linux mediante la lectura del árbol de dispositivos [6]. Si el modo por defecto no coincide con la función deseada, como es nuestro caso, tendremos que modificar la configuración.

Los modos disponibles por pin del *header* P9 se puede consultar en el manual de referencia del BeagleBone [1] y los detalles de configuración en el manual de referencia del procesador [2]. La siguiente tabla realizada por Derek Molloy para el BeagleBone Black resume los detalles que necesitamos para configurar el *header* P9

Header pin	SPINS	ADDR/OFFSET	Name	GPIO NO.	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	PIN	Notes
P9_01			GND											Ground
P9_02			GND											Ground
P9_03			DC_3.3V											250mA Max Current
P9_04			DC_3.3V											250mA Max Current
P9_05			VDD_5V											1A Max Current (only if DC jack powered)
P9_06			VDD_5V											1A Max Current (only if DC jack powered)
P9_07			SYS_5V											250mA Max Current
P9_08			SYS_5V											250mA Max Current
P9_09			PWR_BUTTON											Has a 5V Level (pulled up by TP565217C)
P9_10			SYS_RESETn										A10	RESET_OUT
P9_11	28	0x870/070	UART5_RXD	30	gpio030	uart4_rxd_mux2	mnc1_sddc	rmi2_crs_dv	gpmc_csn4	mi2_crs	RESET_OUT		A10	Has a 5V Level (pulled up by TP565217C)
P9_12	30	0x878/078	GPIO1_28	60	gpio128	mcsap0_aclv_mux3	gpmc_dir	mnc2_dat3	gpmc_csn6	mi2_col	gpmc_be1n		U18	NB: GPIOs limit current to 4-6mA output and approx. 8mA on input.
P9_13	29	0x874/074	UART4_TXD	31	gpio031	uart4_txd_mux2	mnc2_sddc	rmi2_rnerr	gpmc_csn5	mi2_rnerr	gpmc_wpn		U17	
P9_14	18	0x848/048	EHRPWM1A	50	gpio150	ehrpwm1a_mux3	gpmc_a18	mnc2_dat1	rgmi2_tst3	mi2_dat3	gpmc_a2		U14	
P9_15	16	0x840/040	GPIO1_16	48	gpio116	ehrpwm1_tripzone_input	gpmc_a16	mnc2_twn	rmi2_tst1	mi2_tst1	gpmc_a0		R13	
P9_16	19	0x84c/04c	EHRPWM1B	51	gpio151	ehrpwm1b_mux1	gpmc_a19	mnc2_dat2	rgmi2_tst2	mi2_tst2	gpmc_a3		T14	
P9_17	87	0x95c/15c	I2C1_SCL	5	gpio05		ehrpwm0_sync	I2C1_SCL	mnc2_sddc	spio_cs0			A16	
P9_18	86	0x958/158	I2C1_SDA	4	gpio04		ehrpwm0_tripzone	I2C1_SDA	mnc1_sddc	spio_d1			B16	
P9_19	95	0x97c/17c	I2C2_SCL	13	gpio113		spii_cs1	I2C2_SCL	dcana0_rx	timer5	uart1_rstn		D17	Allocated (Group: pinmux_i2c2_pins)
P9_20	94	0x978/178	I2C2_SDA	12	gpio112		spii_cs0	I2C2_SDA	dcana0_tx	timer5	uart1_ctsn		D18	Allocated (Group: pinmux_i2c2_pins)
P9_21	85	0x954/154	UART2_TXD	3	gpio03	EMU3_mux1	ehrpwm0B	I2C2_SCL	uhc1_sda	uart2_txd	spio_d0		B17	
P9_22	84	0x950/150	UART2_RXD	2	gpio02	EMU2_mux1	ehrpwm0A	I2C2_SDA	uhc1_sda	uart2_rxd	spio_sck		A17	
P9_23	17	0x844/044	GPIO1_17	49	gpio117	ehrpwm0_sync	gpmc_a17	mnc2_dat0	rgmi2_rdv	gmi2_rdv	gpmc_a1		V14	
P9_24	97	0x984/184	UART1_TXD	15	gpio115		I2C1_SCL	dcana1_rx	mnc2_sddc	uart1_txd			D15	
P9_25	107	0x9bc/1bc	GPIO3_21	117	gpio321	EMU4_mux2	mcsap1_aur1	mcsap0_aur3	eQEP0_strobe	mcsap0_ahclk			A14	Allocated (Group: mcsap0_pins)
P9_26	96	0x980/180	UART1_RXD	14	gpio114		I2C1_SDA	dcana1_tx	mnc1_sddc	uart1_rxd			D16	
P9_27	105	0x9b4/1b4	GPIO3_19	115	gpio319	EMU2_mux2	mcsap1_fix	mcsap0_aur3	eQEP0B_in	mcsap0_tsr			C13	
P9_28	103	0x9b0/1b0	SPI1_CS0	113	gpio313		spii_cs0	mcsap0_aur2	ehrpwm0_sync	mcsap0_ahclk			C12	Allocated (Group: mcsap0_pins)
P9_29	101	0x994/194	SPI1_D0	111	gpio311		mnc1_sddc_mux1	spii_d0	ehrpwm0B	mcsap0_fix			B13	Allocated (Group: mcsap0_pins)
P9_30	102	0x998/198	SPI1_D1	112	gpio312		mnc2_sddc_mux1	spii_d1	ehrpwm0_tripzone	mcsap0_aur0			D12	
P9_31	100	0x990/190	SPI1_SCLK	110	gpio310		mnc2_sddc_mux1	spii_sck	ehrpwm0A	mcsap0_ahclk			A13	Allocated (Group: mcsap0_pins) Voltage Reference for ADC (NB: 1.8V)
P9_32			VADC											
P9_33			AIN4										C8	NB: 1.8V tolerant
P9_34			AIN0											Ground for ADC
P9_35			AIN5										A8	NB: 1.8V tolerant
P9_36			AIN5										B8	NB: 1.8V tolerant
P9_37			AIN2										B7	NB: 1.8V tolerant
P9_38			AIN3										A7	NB: 1.8V tolerant
P9_39			AIN0										B6	NB: 1.8V tolerant
P9_40			AIN1										C7	NB: 1.8V tolerant
P9_41A	109	0x9b1/1b1	CLKOUT2	20	gpio320	EMU3_mux0	timer7_mux1	ckout2		tcclk	xdma_event_intr1		D14	Both signals are connected to P21 of P11
P9_41B		0x9b1/1b1	GPIO3_20	116	gpio320		emu3	Mcsap1_aur0		eQEP0_index	mcsap0_aur1		D13	Both signals are connected to P21 of P11
P9_42A	89	0x964/164	GPIO0_7	7	gpio07	xdma_event_intr2	mnc1_sddc	pr1_eca0p_eca0_capin_apwm_0	spii_cs1	uart3_txd	eCAP0_in_PWM0_out		C18	Both signals are connected to P22 of P11
P9_42B		0x964/164	GPIO3_18	114	gpio318		spii_sck	Mcsap1_ahclk	Mcsap0_aur2	eQEP0A_in	Mcsap0_aclkr		B12	Both signals are connected to P22 of P11 - See Pg.50 of the SPM
P9_43			GND											Ground
P9_44			GND											Ground
P9_45			GND											Ground
P9_46			GND											Ground

Figura 5.13 - Tabla resumen funciones GPIO BeagleBone

Para la modificación de la información contenida en el árbol de dispositivos nos fijaremos en las entradas correspondientes a P9_11, P9_13, P9_21 y P9_22. Concretamente necesitamos conocer el *offset* de los pines a modificar (para identificarlos en el mapa memoria) y el modo de funcionamiento deseado.

UART3:

1. P9_21: Offset 0x154. La función deseada (uart2_txd) viene determinada por el Modo 1
2. P9_22: Offset 0x150. La función deseada (uart2_rxd) viene determinada por el Modo 1

UART5:

- P9_13: Offset 0x74. La función deseada (uart4_txd_mux2) viene determinada por el Modo 6
- P9_11: Offset 0x70. La función deseada (uart4_rxd_mux2) viene determinada por el Modo 6

Con la información anterior construiremos una palabra de configuración de 32 bits, donde el modo se representa en los 3 bits menos significativos, los bits 4 y 5 corresponden a la configuración de resistencias internas de pullup/pulldown. El bit 6 determina si es entrada o salida. El resto lo pondremos a 0.

Table 9-61. conf_<module>_<pin> Register Field Descriptions

Bit	Field	Type	Reset	Description
31-20	Reserved	R	0h	
19-7	Reserved	R	0h	
6	conf_<module>_<pin>_slewctrl	R/W	X	Select between faster or slower slew rate 0: Fast 1: Slow Reset value is pad-dependent.
5	conf_<module>_<pin>_rx_active	R/W	1h	Input enable value for the PAD 0: Receiver disabled 1: Receiver enabled
4	conf_<module>_<pin>_pullup_type_sel	R/W	X	Pad pullup/pulldown type selection 0: Pulldown selected 1: Pullup selected Reset value is pad-dependent.
3	conf_<module>_<pin>_pullup_enable	R/W	X	Pad pullup/pulldown enable 0: Pullup/pulldown enabled 1: Pullup/pulldown disabled Reset value is pad-dependent.
2-0	conf_<module>_<pin>_mux_mode	R/W	X	Pad functional signal mux select. Reset value is pad-dependent.

Figura 3 Registro de configuración GPIO en procesador

Así tendremos:

1. P9_21: UART2_TXD Salida (1), Pulldown (00), Modo 1 (001) -> 0b100001 -> **0x21**
2. P9_21: UART2_RXD Entrada (0), Pulldown (00), Modo 1 (001) -> 0b000001 -> **0x01**
3. P9_13: UART4_TXD Salida (1), Pulldown (00), Modo 6 (110) -> 0b100110 -> **0x26**
4. P9_11: UART4_RXD Entrada (0), Pulldown (00), Modo 6 (110) -> 0b000110 -> **0x06**

Para introducir la configuración anterior crearemos un *overlay* que modifica el comportamiento de los pines sin necesidad de modificar todo el arbol de dispositivos

```

/*
 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * Modificado por Fernando Soto para habilitar dos UART en P9
 */

/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

    /* identification */
    part-number = "uart5";

    fragment@0 {
        target = <&am33xx pinmux>;
        __overlay__ {
            pinctrl_uarts_3_and_5: pinctrl_uarts_3_and5_pins {
                pinctrl-single,pins = <
                    0x070 0x26 /* P9_11 = UART4 TXD, MODE6 */
                    0x074 0x06 /* P9_13 = UART4 RXD, MODE6 */
                    0x150 0x21 /* P9_22 = UART2_RXD, MODE1 */
                    0x154 0x01 /* P9_21 = UART2_TXD, MODE1 */
                >;
            };
        };
    };

    fragment@1{

```

```
target = <&uart5>;
overlay {
    status = "okay";
};
};
fragment@2{
target = <&uart3>;
__overlay__ {
    status = "okay";
};
};
fragment@3 {
target = <&ocp>;
overlay {
    test helper: helper {
        compatible = "bone-pinmux-helper";
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_uarts_3_and_5>;
        status = "okay";
    };
};
};
};
```

1. Introducimos el siguiente contenido en un archivo que llamaremos *enable_uarts*
2. Compilamos usando el compilador de árbol de dispositivos indicando la versión de firmware de nuestro BeagleBone (añadiendo -00A0 al nombre del resultado)

```
dtc -O dtb -o enable_uarts-00A0.dtbo -b 0 -@ enable_uarts.dts
```

3. Copiamos el resultado a la carpeta */lib/firmware*

```
cp enable_uarts-00A0.dtbo /lib/firmware
```

4. Por comodidad creamos las siguientes variables de entorno

```
export PINS=/sys/kernel/debug/pinctrl/44e10800.pinmux/pins
export SLOTS=/sys/devices/bone_capemgr.7/slots
```

comprobando que las rutas son correctas (podría ser bone_capemgr.9 si trabajamos con un BeagleBone Black)

5. Comprobamos el contenido de \$SLOTS y \$PINS antes de activar el *overlay*

```
cat $SLOTS
```

el resultado será algo así:

```
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
```

6. Para ver solamente el contenido a modificar de \$PINS filtramos por las direcciones de memoria indicadas en la tabla para los pines correspondientes (campo ADDR)

```
cat $PINS | egrep "950|954|874|870"
```

observando que están todos en Modo 7 con pulldown deshabilitado

```
pin 28 (44e10870) 00000037 pinctrl-single
pin 29 (44e10874) 00000037 pinctrl-single
pin 84 (44e10950) 00000037 pinctrl-single
pin 85 (44e10954) 00000037 pinctrl-single
```

7. Activamos el *overlay*

```
echo enable_uarts > $SLOTS
```

Ahora volvemos a listar \$SLOTS y \$PINS comprobando las diferencias

```
0: 54:PF---
1: 55:PF---
2: 56:PF---
3: 57:PF---
4: ff:P-O-L Override Board Name,00A0,Override Manuf,enable-uarts
```

```
pin 28 (44e10870) 00000026 pinctrl-single
pin 29 (44e10874) 00000006 pinctrl-single
pin 84 (44e10950) 00000021 pinctrl-single
pin 85 (44e10954) 00000001 pinctrl-single
```

5.9.2. PRUEBA DE COMUNICACIONES

Para probar las UART recientemente habilitadas podemos conectarlas entre si (UART2_TXD con UART4_RXD y UART2_RXD con UART4_TXD). Si además conectamos un diodo LED a cada sentido de comunicación (conectado a GND del BeagleBone mediante resistencias de pulldown) podremos ver un ligero parpadeo cuando haya comunicación.

Y finalmente comprobar la conexión abriendo dos terminales al BB y ejecutando los siguientes comandos (uno en cada terminal)

```
cat /dev/ttyO2
```

```
echo hola > /dev/ttyO4
```

El resultado esperado es visualizar en la salida de /dev/ttyO2 la palabra "hola" introducida por /dev/ttyO4

6. APÉNDICE II. CÓDIGO FUENTE

6.1. Librería Comunicación Serie

Archivo de cabecera Serial_Q.h

```

/*
 * Serial_Q.h
 *
 * Created on: Jun 28, 2013
 * Author: youbot
 */

#ifndef SERIAL_Q_H_
#define SERIAL_Q_H_

#include <iostream>          /* Definiciones de la librería estandar de entrada/salida de C++ */
#include <cstdio>            /* Definiciones de la librería estandar de entrada/salida */
#include <cstdlib>          /* Definiciones de la librería estandar de propósito general */
#include <cstring>          /* Definiciones relacionadas con cadenas de caracteres */
#include <cerrno>           /* Definiciones de códigos de error */
#include <csignal>          /* Definiciones de señales */

using namespace std;

extern "C" {

    #include <termios.h>    /* Definiciones relacionadas con control de terminal POSIX */
    #include <fcntl.h>      /* Definiciones relacionadas con control de ficheros */
    #include <stdbool.h>    /* Definiciones para uso de variables booleanas */
}

#define Parity_None        IGNPAR
#define Parity_Even        PARENB
#define Parity_Odd         PARENB | PARODD

#define StopBits_1        0
#define StopBits_2        CSTOPB

#define DataBits_5         CS5
#define DataBits_6         CS6
#define DataBits_7         CS7
#define DataBits_8         CS8

#define ReadMode_SyncBlocking          0
#define ReadMode_SyncNonBlocking      1
#define ReadMode_AsyncWithSignal      2

typedef void (*signalHandler)(int);

namespace Serial {

    ////////////////////////////////////////////////////
    // Clase para comunicación serie con terminos//
    ////////////////////////////////////////////////////

    class Serial {

    private:

        // Miembros privados

        struct termios myOldTIO, myNewTIO;
        struct sigaction mySigAction;

        char* myDevice;
        int myBaudRate;
        int myDataBits;
        int myStopBits;
        int myParity;
        int myReadMode;

    protected:

        int myHandler;

        signalHandler mySignalHandler;

    public:

        // Miembros públicos

```

```

        char* LastError;

        Serial(const char* inDevice, int inBaudRate, int inDataBits, int inParity, int inStopBits,
int inReadMode, signalHandler inSignalHandler = NULL);

        ~Serial();

        bool connect();
        int send(const char* inBytes);
        int send(const char* inBytes, int inNumBytes);
        int receive(char* inBytes, int inMaxNumBytes);

};

//////////
// Cola //
//////////

class Queue {

private:

        // Miembros privados

        int mySize;
        unsigned char* myBuff;
        int myHead;
        int myNumBytes;

public:

        // Miembros públicos

        Queue(int inSize);

        ~Queue();

        int getNumBytes();
        int getSize();
        unsigned char* getFullContent(bool inDequeueBytes);
        unsigned char dequeue();
        void enqueue(const unsigned char c);

};

//////////
// Clase de comunicación serie simplificada //
// - Se configura solo con la velocidad y el nombre del dispositivo //
// usando los demás parámetros por defecto (los habituales 8N1) //
// - Utiliza lectura síncrona sin bloqueo insertando los bytes recibidos //
// en una cola (en las llamadas a checkDataAndEnqueue()) //
//////////

class Serial Q : public Serial {

private:

        Queue* myQ;

public:

        Serial Q(const char* inDevice, int inBaudRate);

        bool checkDataAndEnqueue();
        unsigned char dequeue();
        unsigned char* getFullQueueContent(bool inDequeueBytes);
        int getNumBytesInQ();

};

}

#endif /* SERIAL_Q_H_ */

```

Archivo de código Serial_Q.cpp

```

/*
 * Serial.cpp
 *
 * Created on: Jun 27, 2013
 * Author: Fernando Soto
 *
 * Clases:
 *
 * Serial: implementación en C++ de comunicación serie con términos
 * Queue: cola
 * Serial Q: clase para comunicación serie simplificada
 */

#include "Serial_Q.h"

namespace Serial {

    Serial::Serial(const char* inDevice, int inBaudRate, int inDataBits, int inParity, int inStopBits,
int inReadMode, signalHandler inSignalHandler) {

        int theFlags;

        // Inicializamos los miembros privados con los datos de configuración

        this->myDevice = strdup(inDevice);
        this->myBaudRate = inBaudRate;
        this->myDataBits = inDataBits;
        this->myParity = inParity;
        this->myStopBits = inStopBits;
        this->myReadMode = inReadMode;

        cout << "Iniciando conexión serie en " << this->myDevice << " a " << this->myBaudRate << "
bps " << this->myDataBits << this->myParity << this->myStopBits << " en modo " << this->myReadMode << endl;

        this->mySignalHandler = inSignalHandler;
        this->LastError = NULL;

        // Establecemos los Flags de configuración para abrir el dispositivo serie
        // O_RDWR para comunicación bidireccional (lectura/escritura)
        // O_NOCTTY para que el terminal no controle nuestro proceso.
        // (esto evita entre otras cosas que se reciban señales inesperadas procedentes del
terminal).

        theFlags = O_RDWR | O_NOCTTY;

        // Configuración según tipo de lectura

        if (this->myReadMode != ReadMode SyncBlocking) {

            // O_NONBLOCK para indicar que las lecturas deben retornar el control
inmediatamente,
            // sin esperar por un número mínimo de bytes recibidos.

            theFlags |= O_NONBLOCK;

        }

        // Abrimos dispositivo serie.
        // Si no hay error, open devolverá un entero positivo

        this->myHandler = open(this->myDevice,theFlags);

        // Si se produce un error, el código de error se puede obtener en la variable errno
        // y sus significado mediante strerror(errno).
        // Es necesario incluir cabecera errno.h para esto.

        if (this->myHandler < 0) {
            this->LastError = strerror(errno);
        }

        // Salvamos la configuración original del dispositivo serie en myOldTIO

        tcgetattr(this->myHandler,&this->myOldTIO);

        // Inicializamos la nueva configuración (myNewTIO) con ceros

        bzero(&this->myNewTIO, sizeof(this->myNewTIO));

        // Comprobamos si se trata de modo asíncrono

        if (this->myReadMode == ReadMode_AsyncWithSignal) {

            // Las lecturas asincrónicas se gestionan mediante el uso de señales.

```

```

// El sistema operativo envia una señal SIGIO cada vez que llegue un dato al dispositivo serie
// Es responsabilidad del usuario de esta clase el programar la función que reaccionará ante la
señal
// y pasar un puntero a la misma en la llamada constructor de Serial.
// Importante: el envío de señales a un proceso dormido (mediante sleep) provoca su reanudación.

// Asignamos la función delegada para hacerse cargo de la señal

this->mySigAction.sa handler = this->mySignalHandler;
//mySigAction.sa mask = 0;
this->mySigAction.sa flags = 0;
this->mySigAction.sa_restorer = NULL;

sigaction(SIGIO,&this->mySigAction,NULL);

// Permitimos al proceso recibir señales

fcntl(this->myHandler, F_SETOWN, getpid());

// Hacemos asincrono el descriptor

fcntl(this->myHandler, F_SETFL, FASYNC); // FASYNC | FNDELAY indicado en
http://ulisse.elettra.trieste.it/services/doc/serial/
}

////////////////////////////////////
// Modo de control //
////////////////////////////////////

// Sin control de flujo
// No modificar el propietario del dispositivo. Habilitar recepción.

this->myNewTIO.c_cflag = CLOCAL | CREAD;

this->myNewTIO.c_cflag &= ~CRTSCTS;

// velocidad, bits de datos, bits de parada ...

this->myNewTIO.c_cflag |= this->myBaudRate | this->myDataBits | this->myStopBits | this-
>myParity;

////////////////////////////////////
// Modo de entrada //
////////////////////////////////////

this->myNewTIO.c_iflag = 0;

////////////////////////////////////
// Modo de salida //
////////////////////////////////////

// Nada en especial.

this->myNewTIO.c_oflag = 0;

////////////////////////////////////
// Modo local //
////////////////////////////////////

// No Canónico. Sin eco, ...

this->myNewTIO.c_lflag = 0;

////////////////////////////////////
// Configuración específica de modo NO CANÓNICO //
////////////////////////////////////

// VTIME: Tiempo mínimo (en décimas de segundo) a esperar por posibles
// VMIN: recepciones de bytes antes de que read() devuelva el control

if (this->myReadMode == ReadMode_SyncBlocking) {

    this->myNewTIO.c_cc[VTIME] = 5;
    this->myNewTIO.c_cc[VMIN] = 9;

} else {

    this->myNewTIO.c_cc[VTIME] = 0;
    this->myNewTIO.c_cc[VMIN] = 0;

}

////////////////////////////////////
// Fin de configuración //
////////////////////////////////////

// Limpiamos las colas de entrada y salida
// descartando los bytes recibidos aún no leídos
// y los bytes enviados al dispositivo aún no transmitidos

```

```

        tcflush(this->myHandler, TCIOFLUSH);

        // Aplicamos la nueva configuración
        tcsetattr(this->myHandler, TCSANOW, &this->myNewTIO);
    }

    // Destructor
    Serial::~Serial() {
        // Liberamos los recursos utilizados por el objeto Serial
        // 1. Restablecer los parámetros originales del dispositivo
        // 2. Cerrar el dispositivo

        if (this->myHandler > 0) {
            tcsetattr(this->myHandler, TCSANOW, &this->myOldTIO);
            close(this->myHandler);
        }
    }

    // Función de envío de cadena de caracteres.
    // Se asume que la cadena está correctamente terminada (por '\0')

    int Serial::send(const char* inBytes) {
        return this->send(inBytes, strlen(inBytes));
    }

    int Serial::send(const char* inBytes, int inNumBytes) {
        int theBytesSent;
        theBytesSent = write(this->myHandler, inBytes, inNumBytes);
        cout << "Serial::send. Enviados " << theBytesSent << " bytes" << endl;
        return theBytesSent;
    }

    // Función de recepción de cadena de caracteres

    int Serial::receive(char* inBytes, int inMaxNumBytes) {
        int theBytesReceived;
        theBytesReceived = read(this->myHandler, inBytes, inMaxNumBytes);
        //cout << "Serial::receive. Recibidos " << theBytesReceived << " bytes" << endl;
        return theBytesReceived;
    }

    // Constructor
    Queue::Queue(int inSize) {
        this->mySize = inSize;
        this->myBuff = (unsigned char *)malloc(this->mySize);
        this->myNumBytes = 0;
        this->myHead = 0;
    }

    // Insertar un byte en la cola

    void Queue::enqueue(unsigned char inC) {
        if (this->myNumBytes < this->mySize) {
            this->myHead++;
            if (this->myHead >= this->mySize) {
                this->myHead = 0;
            }
            this->myBuff[this->myHead] = inC;
            this->myNumBytes++;
        } else {
            cout << "Overflow!" << endl;
        }
    }

    // Extraer un byte de la cola

    unsigned char Queue::dequeue() {
        char outByte = 0;
        int theTail;

        if (this->myNumBytes > 0) {
            theTail = this->myHead - this->myNumBytes + 1;
            if (theTail < 0) theTail += this->mySize;
            outByte = this->myBuff[theTail];
            this->myNumBytes--;
        }
    }

```

```

        return outByte;
    }

    // Mostrar el contenido de la cola (opcionalmente consumiendo los bytes leídos)
    unsigned char* Queue::getFullContent(bool inDequeueBytes) {

        int i;
        int theLength = this->myNumBytes;
        unsigned char* outStr = (unsigned char*)malloc(theLength+1);

        if(outStr == NULL) return NULL;
        for (i=0;i<theLength;i++) {
            outStr[i] = this->dequeue();
            if (!inDequeueBytes) {
                this->enqueue(outStr[i]);
            }
        }
        outStr[theLength] = '\0';

        return outStr;
    }

    // Mostrar el número de bytes disponibles en la cola
    int Queue::getNumBytes() {
        return this->myNumBytes;
    }

    // Muestrar el tamaño máximo de la cola
    int Queue::getSize() {
        return this->mySize;
    }

    // Constructor
    Serial Q::Serial Q(const char* inDevice, int inBaudRate) : Serial(inDevice, inBaudRate, DataBits 8,
    Parity None, StopBits 1, ReadMode SyncNonBlocking) {
        this->myQ = new Queue(4096);
    }

    // Cada llamada a checkDataAndEnqueue() inserta en la cola todos los datos disponibles en el
    dispositivo

    bool Serial Q::checkDataAndEnqueue() {

        int theBytesRead, i;
        char* theBuff;

        if (this->myQ->getNumBytes() < this->myQ->getSize()) {
            theBuff = (char*)malloc(32);
            theBytesRead = Serial::receive(theBuff,32);
            while ((theBytesRead > 0)&&(this->myQ->getNumBytes() < this->myQ->getSize())) {
                for (i = 0;i< theBytesRead; i++) {
                    this->myQ->enqueue((unsigned char)theBuff[i]);
                }
                theBytesRead = Serial::receive(theBuff,256);
            }
        }

        return (myQ->getNumBytes() > 0);
    }

    // Extrae de la cola el siguiente byte
    unsigned char Serial_Q::dequeue() {
        return this->myQ->dequeue();
    }

    // Devuelve todo el contenido de la cola. Mantiene o no en la cola los caracteres devueltos según
    sen indique en inDequeueBytes
    unsigned char* Serial_Q::getFullQueueContent(bool inDequeueBytes) {
        return this->myQ->getFullContent(inDequeueBytes);
    }

    int Serial Q::getNumBytesInQ() {
        return this->myQ->getNumBytes();
    }
}

```

6.2. Determinación de coordenadas con Asus Xtion y OpenNI

El siguiente programa ejemplo imprime por pantalla las coordenadas (x,y,z) del punto más cercano detectado por el un sensor de profundidad (Asus Xtion, MS Kinect o cualquier otro compatible con OpenNI).

```
#include <iostream>
#include <OpenNI.h>

using namespace std;

openni::Status theStatus;
openni::Device theDevice;
openni::VideoStream theDepth;

class Pixel3D {
public:
    int X;
    int Y;
    int Z;

    void imprime() {
        printf("Pixel3D (%d,%d,%d)\n",X,Y,Z);
    }
};

// Obtiene las coordenadas del punto con menor profundidad
// Las coordenadas X e Y identifican un número de pixel dentro del cuadro
// La coordenada Z es la profundidad
openni::Status calculaPuntoMasCercano(Pixel3D* closestPoint, openni::VideoFrameRef* rawFrame) {

    openni::DepthPixel* pDepth = (openni::DepthPixel*)rawFrame->getData();
    bool found = false;
    closestPoint->Z = 0xffff;
    int width = rawFrame->getWidth();
    int height = rawFrame->getHeight();

    for (int y = 0; y < height; ++y)
        for (int x = 0; x < width; ++x, ++pDepth)
        {
            if (*pDepth < closestPoint->Z && *pDepth != 0)
            {
                closestPoint->X = x;
                closestPoint->Y = y;
                closestPoint->Z = *pDepth;
                found = true;
            }
        }

    if (!found)
    {
        return openni::STATUS_ERROR;
    }

    return openni::STATUS_OK;
}

int main(int argc, char ** argv) {

    cout << "Inicializando ...";
    theStatus = openni::OpenNI::initialize();

    if (theStatus != openni::STATUS_OK) {
        printf("Device open failed:\n%s\n", openni::OpenNI::getExtendedError());
        openni::OpenNI::shutdown();
        return 1;
    }

    cout << "Iniciando sensor de profundidad ...";
```

```

openni::Array<openni::DeviceInfo> theDevices;
openni::OpenNI::enumerateDevices(&theDevices);

// Busca dispositivos compatibles
// Inicializa la variable theDevice con cada uno
// E imprime por pantalla su numero de serie

for (int i = 0; i != theDevices.getSize(); ++i) {
    const openni::DeviceInfo& theDeviceInfo = theDevices[i];
    string uri = theDeviceInfo.getUri();
    theDevice.open(uri.c_str());
    char theSerialNumber[1024];
    theDevice.getProperty(ONI_DEVICE_PROPERTY_SERIAL_NUMBER, &theSerialNumber);
    cout << "Device " << i << ". Serial Number: " << theSerialNumber << endl;
}

// Crea un stream de cuadros a partir del dispositivo inicializado anteriormente
// Se indica que el dispositivo es de tipo profundidad

theStatus = theDepth.create(theDevice, openni::SENSOR_DEPTH);

if (theStatus == openni::STATUS_OK) {
    theStatus = theDepth.start();
    if (theStatus != openni::STATUS_OK) {
        printf("Couldn't start depth stream:\n%s\n", openni::OpenNI::getExtendedError());
        theDepth.destroy();
    }
} else {
    printf("Couldn't find depth stream:\n%s\n", openni::OpenNI::getExtendedError());
}

if (!theDepth.isValid()) {
    printf("No valid streams. Exiting\n");
    openni::OpenNI::shutdown();
    return 2;
}

openni::VideoFrameRef theRawFrame;
Pixel3D theClosestPoint;

// Por cada cuadro recibido se obtiene el punto más cercano y se muestra por pantalla
while(theDepth.isValid()){

    openni::Status rc = theDepth.readFrame(&theRawFrame);
    if (rc != openni::STATUS_OK) {
        printf("readFrame failed\n%s\n", openni::OpenNI::getExtendedError());
    } else {
        calculaPuntoMasCercano(&theClosestPoint, &theRawFrame);
        theClosestPoint.imprime();
    }
}

theDepth.stop();
theDepth.destroy();

cout << "Terminando" << endl;
openni::OpenNI::shutdown();

return 0;
}

```

Compilando y ejecutando el programa anterior en el proyecto de eclipse recién creado podemos observar como cambia la salida al mover el sensor o acercar la mano.

Sin embargo la información proporcionada no sirve aún para determinar la posición del punto más cercano, puesto que está expresada en píxeles.

Estas unidades se pueden transformar en coordenadas reales mediante la clase *openni::CoordinateConverter*. Así que modificaremos el código añadiendo una clase *Point*.

```
class Point {
public:
    float X;
    float Y;
    float Z;

    void imprime() {
        printf("Point (%g,%g,%g)\n",X,Y,Z);
    }
};
```

Y, llamando a la función de conversión para obtener las coordenadas reales obtenemos las coordenadas en milímetros:

```
...
    openni::VideoFrameRef theRawFrame;
    Pixel3D theClosestPoint;
    Point theRealPoint;

    // Por cada cuadro recibido se obtiene el punto más cercano y se muestra por pantalla
    while(theDepth.isValid()){

        openni::Status rc = theDepth.readFrame(&theRawFrame);
        if (rc != openni::STATUS_OK) {
            printf("readFrame failed\n%s\n", openni::OpenNI::getExtendedError());
        } else {
            calculaPuntoMasCercano(&theClosestPoint,&theRawFrame);
            //theClosestPoint.imprime();

            openni::CoordinateConverter::convertDepthToWorld(theDepth,theClosestPoint.X,theClosestPoint.Y,theClosestPoint.Z,&theRealPoint.X, &theRealPoint.Y, &theRealPoint.Z);
            theRealPoint.imprime();
        }
    }
}
```

6.3. Control Unidad PAN Tilt PTU-D46 con ROS

A continuación se muestra el listado de código fuente del nodo ROS utilizado para controlar la unidad pan-tilt PTU-D46 a través de ROS.

La información llega al nodo mediante los topics *pan_tilt_pos_in* (mensajes de tipo *JointState*) y *joy* (mensajes tipo *Joy*). El nodo se encarga de traducir los mensajes de posición recibidos en comandos para la unidad PTU-D46 y transmitirlos mediante comunicación serie.

```
#include <iostream>
#include "ros/ros.h"
#include "sensor_msgs/JointState.h"
#include "sensor_msgs/Joy.h"
#include "Serial_Q.h"

#define PI 3.1415927

// Resolución por defecto del PTU en segundos de grado

#define PAN_RESOLUTION 185.1428
#define TILT_RESOLUTION 185.1428

#define JOYSTEP PAN DEG 10.0
#define JOYSTEP TILT DEG 10.0

#define SERIALDEVICE "/dev/ttyUSB0"

#define PTU_OK 0
#define PTU_ERROR 1
#define PTU_UNEXPECTED 2

#define PAN 0
#define TILT 1

#define ABSOLUTE 0
#define RELATIVE 1

using namespace std;

void posCallback(const sensor_msgs::JointState& inJointState);
void joyCallback(const sensor_msgs::Joy& inJoyCommand);
bool getPosCommand(float inDeg, int inTargetJoint, int inMode, char* outCommand);
void mydelay_ms(int milliseconds);
void processPtuComm();

ros::NodeHandle* myNodeHandle = 0;

Serial::Serial_Q *Ptu;

char tmpChar;
char status;
char mySerialDevice[] = "/dev/ttyUSB0";

sensor_msgs::JointState thePtuJointState;

enum estado {
    IDLE = 0,
    WAIT_COMMAND_CONF = 1,
    WAIT_POS_PAN = 2,
    WAIT_POS_TILT = 3
};

estado STATUS = IDLE;

void envia(int inFd, const char* inStr) {

    Ptu->send("I ");
    processPtuComm();
    Ptu->send("inStr ");
    processPtuComm();
    Ptu->send("A ");
    processPtuComm();
}

int main(int argc, char** argv) {

    thePtuJointState.name.resize(2); // array de tamaño 2: PAN y TILT
    thePtuJointState.position.resize(2); // array de tamaño 2: PAN y TILT
```

```

thePtuJointState.name[0] = "PAN";
thePtuJointState.name[1] = "TILT";

ros::init(argc, argv, "pan_tilt_ptu_46");
ros::NodeHandle theNodeHandle;
myNodeHandle = &theNodeHandle;
//ros::Subscriber thePosSubscriber = theNodeHandle.subscribe("/ptu_46_pos_in", 10, posCallback);
ros::Subscriber theJoySubscriber = theNodeHandle.subscribe("/joy", 10, joyCallback);
ros::Publisher thePublisher =
theNodeHandle.advertise<sensor_msgs::JointState>("/ptu_46_pos_out",10);

ROS_INFO("Nodo de control PTU-46 iniciado");

// Init PTU-46

Ptu = new Serial::Serial Q(mySerialDevice, B9600);

if (Ptu->LastError != NULL) {
    ROS_ERROR("Error conectando con unidad PTU-46: %s",Ptu->LastError);
    return -1;
}

ROS_INFO("Puerto serie abierto %d",B9600);

// Enviamos comando de modo de ejecución inmediato

Ptu->send("I "); // Modo inmediato
Ptu->send("FT "); // Respuestas escuetas

ros::Rate r(1); // 1 hz
while (myNodeHandle->ok()) {
/*
    // Enviamos al PTU petición de posición PAN
    Ptu->send("PP\n");
    STATUS = WAIT_POS_PAN;
    processPtuComm();

    // Enviamos al PTU petición de posición TILT
    Ptu->send("TP\n");
    STATUS = WAIT_POS TILT;
    processPtuComm();
*/
    // Publicamos la posición obtenida
    thePublisher.publish(thePtuJointState);
    ros::spinOnce();
}

cout << "Programa terminado" << endl;
return 0;
}

bool getPosCommand(float inDeg, int inTargetJoint, int inMode, char* outCommand) {

    int thePosVal;
    char theParam = 'P';
    char theMode = ABSOLUTE;
    bool ok = true;

    switch(inTargetJoint) {
        case PAN:
            thePosVal = (int) (inDeg * 3600 / PAN_RESOLUTION);
            theParam = 'P';
            break;
        case TILT:
            thePosVal = (int) (inDeg * 3600 / TILT_RESOLUTION);
            theParam = 'T';
            break;
        default: // inesperado
            ok = false;
    }

    switch (inMode) {
        case ABSOLUTE: // PP o TP
            theMode = 'P';
            break;
        case RELATIVE: // PO o TO
            theMode = 'O';
            break;
        default: // inesperado
            ok = false;
    }

    if (ok) {
        sprintf(outCommand,"%c%c%d ",theParam,theMode,thePosVal);
    }
}

```

```

        return ok;
    }

void posCallback(const sensor_msgs::JointState& inJointState) {

    float thePanDeg = (float)inJointState.position[0] * 180 / PI;
    float theTiltDeg = (float)inJointState.position[1] * 180 / PI;
    char thePanCommand[16];
    char theTiltCommand[16];
    bool theBuildCommandOk = false;

    theBuildCommandOk = getPosCommand(thePanDeg, PAN, ABSOLUTE, thePanCommand);
    theBuildCommandOk = theBuildCommandOk && getPosCommand(theTiltDeg, TILT, ABSOLUTE, theTiltCommand);

    if (theBuildCommandOk) {
        if (PtU->send(thePanCommand) > 0) {
            STATUS = WAIT_COMMAND_CONF;
            processPtUComm();
        }
        if (PtU->send(theTiltCommand) > 0) {
            STATUS = WAIT_COMMAND_CONF;
            processPtUComm();
        }
    } else {
        // Error construyendo mensaje. no enviar nada
        // ...
    }
}

void joyCallback(const sensor_msgs::Joy& inJoyCommand) {

    char thePanCommand[16];
    char theTiltCommand[16];
    float theAxesH = (float)inJoyCommand.axes[0];
    float theAxesV = (float)inJoyCommand.axes[1];

    printf("Recibido comando joystick (%g,%g)\n", theAxesV, theAxesH);

    // Interpretar los comandos de Joystick como instrucciones
    // para mover JOYSTEP_PAN_DEG (JOYSTEP_PAN_TILT) grados el eje correspondiente (PAN o TILT) en el
    sentido indicado

    if (theAxesH != 0.0) {
        getPosCommand(theAxesH*JOYSTEP_PAN_DEG, PAN, RELATIVE, thePanCommand);
        if (PtU->send(thePanCommand) > 0) {
            STATUS = WAIT_COMMAND_CONF;
            processPtUComm();
        }
    }

    if (theAxesV != 0.0) {
        getPosCommand(theAxesV*JOYSTEP_TILT_DEG, TILT, RELATIVE, theTiltCommand);
        if (PtU->send(theTiltCommand) > 0) {
            STATUS = WAIT_COMMAND_CONF;
            processPtUComm();
        }
    }
}

/*
 * Función de retardo con espera activa
 * Por si se usa recepción serie asíncrona mediante señales
 * las señales recibidas al llegar datos serie no dejan terminar el período establecido en sleep
 * (sleep pausa el proceso hasta que se cumpla el intervalo establecido o hasta que se reciba una señal)
 */
void mydelay_ms(int milliseconds) {

    time_t t = time(NULL) + milliseconds;
    while(time(NULL) < t);
}

bool extractInt(const string inStr, int* outNum) {

    stringstream ss(inStr);
    string tmp;
    ss >> tmp >> *outNum;
    return true;
}

void processPtUComm() {

    string theResp;
    std::size_t theFound;
}

```

```

if (PtU->checkDataAndEnqueue()) {

    theResp.assign((char *)PtU->getFullQueueContent(true));

    // Un signo de exclamación indica que se ha producido un error

    theFound = theResp.find("!");
    if (theFound != string::npos) {
        STATUS = IDLE;
        ROS_ERROR("PTU-46 ha devuelto un error: %s",theResp.c_str());
        return;
    }

    switch (STATUS) {

        case IDLE:

            // Recepción inesperada
            // Mostramos datos recibidos

            ROS_ERROR("PTU-46 ha enviado un dato inesperado: %s",theResp.c_str());

            break;

        case WAIT_COMMAND_CONF:

            theFound = theResp.find("*");
            if (theFound != string::npos) {
                ROS_INFO("PTU-46 ha confirmado la última orden:
%s",theResp.c_str());

                STATUS = IDLE;
            }

            break;

        case WAIT_POS_PAN:

            break;

        case WAIT_POS_TILT:

            // Comprobamos que el comando se ha admitido
            // Y eliminamos el asterisco de confirmación

            int thePos;

            theFound = theResp.find("*");
            if (theFound != string::npos) {
                theResp.erase(0,theFound+1);
                STATUS = IDLE;
            }

            // Leemos el dato devuelto

            if (extractInt(theResp,&thePos)) {

                float theDeg;

                if (STATUS == WAIT_POS_PAN) {
                    theDeg = (float)thePos * PAN_RESOLUTION / 3600;
                    thePtUJointState.position[0] = theDeg;
                } else if (STATUS == WAIT_POS_TILT) {
                    theDeg = (float)thePos * TILT_RESOLUTION / 3600;
                    thePtUJointState.position[1] = theDeg;
                }

            }

            STATUS = IDLE;
            break;

        default:

            STATUS = IDLE;
            break;

    }

}

return;
}

```

6.4. Seguimiento de punto más cercano con PTU-46 y OpenNI

```

#include <iostream>
#include <OpenNI.h>
#include <cmath>
#include "ros/ros.h"
#include "Serial_Q.h"

#define PI 3.14159265359
#define PAN_RESOLUTION 185.1428
#define TILT_RESOLUTION 185.1428
#define SERIALDEVICE "/dev/ttyUSB0"

#define PAN 0
#define TILT 1

#define ABSOLUTE 0
#define RELATIVE 1

#define OFFSET_CAMARA_EJE_TILT_MM 70

enum estado {
    IDLE = 0,
    WAIT_COMMAND_CONF = 1,
    WAIT_POS_PAN = 2,
    WAIT_POS_TILT = 3
};

estado STATUS = IDLE;

using namespace std;

openni::Status theStatus;
openni::Device theDevice;
openni::VideoStream theDepth;

//ros::NodeHandle* myNodeHandle = 0;

Serial::Serial_Q *Ptu;

class Pixel3D {
public:
    int X;
    int Y;
    unsigned short int Z;

    void print() {
        printf("Pixel3D (%d,%d,%d)\n",X,Y,Z);
    }
};

class Point {
public:
    float X;
    float Y;
    float Z;

    Point() {
        this->X = 0.0;
        this->Y = 0.0;
        this->Z = 0.0;
    }

    Point(float inX, float inY, float inZ) {
        this->X = inX;
        this->Y = inY;
        this->Z = inZ;
    }

    void print() {
        printf("Point (%g,%g,%g)\n",X,Y,Z);
    }
};

class Vector {
public:

```

```

float X;
float Y;
float Z;

Vector(Point p1, Point p2) {

    this->X = p2.X - p1.X;
    this->Y = p2.Y - p1.Y;
    this->Z = p2.Z - p1.Z;

}

float getModulo() {

    return sqrt(this->X*this->X + this->Y*this->Y+this->Z*this->Z);

}

float getTilt() {
    return getTiltConsideringOffsetY(0);
}

float getTiltConsideringOffsetY(float inOffsetY) {

    float theModulo = this->getModulo();
    float theCosPan = cos(getPan());
    if ((theModulo > 0)&&(theCosPan != 0)) {
        return asin((sqrt(theModulo*theModulo - inOffsetY*inOffsetY)*this->Y -
inOffsetY*this->Z / theCosPan)/(theModulo*theModulo));
    } else {
        return 0.0;
    }

}

float getPan() {

    if (this->Z > 0) {
        return atan2(this->X, this->Z);
    } else {
        return 0.0;
    }

}

void print() {
    printf("Point (%g,%g,%g)\n",this->X,this->Y,this->Z);
}

void printPanTilt() {
    printf("MODULO: %g, PAN: %g, TILT: %g \n",this->getModulo(),this->getPan(),this->getTilt());
}

void printPanTiltDeg() {
    printf("MODULO: %d, PAN: %d, TILT: %d \n", (int)this->getModulo(), (int)(this-
>getPan()*180/PI), (int)(this->getTilt()*180/PI));
}

};

// Obtiene las coordenadas del punto con menor profundidad
// Las coordenadas X e Y identifican un número de pixel dentro del cuadro
// La coordenada Z es la profundidad
openni::Status calculaPuntoMasCercano(Pixel3D* closestPoint, openni::VideoFrameRef* rawFrame) {

    openni::DepthPixel* pDepth = (openni::DepthPixel*)rawFrame->getData();
    bool found = false;
    closestPoint->Z = 0xffff;
    int width = rawFrame->getWidth();
    int height = rawFrame->getHeight();

    for (int y = 0; y < height; ++y)
        for (int x = 0; x < width; ++x, ++pDepth)
            {
                if (*pDepth < closestPoint->Z && *pDepth != 0)
                {
                    closestPoint->X = x;
                    closestPoint->Y = y;
                    closestPoint->Z = *pDepth;
                    found = true;
                }
            }

    if (!found)
    {
        return openni::STATUS_ERROR;
    }
}

```

```

        return openni::STATUS OK;
    }

bool getPosCommand(float inDeg, int inTargetJoint, int inMode, char* outCommand) {

    int thePosVal;
    char theParam = 'P';
    char theMode = ABSOLUTE;
    bool ok = true;

    switch(inTargetJoint) {
        case PAN:
            thePosVal = (int) (inDeg * 3600 / PAN_RESOLUTION);
            theParam = 'P';
            break;
        case TILT:
            thePosVal = (int) (inDeg * 3600 / TILT_RESOLUTION);
            theParam = 'T';
            break;
        default: // inesperado
            ok = false;
    }

    switch (inMode) {
        case ABSOLUTE: // PP o TP
            theMode = 'P';
            break;
        case RELATIVE: // PO o TO
            theMode = 'O';
            break;
        default: // inesperado
            ok = false;
    }

    if (ok) {
        sprintf(outCommand, "%c%c%d ", theParam, theMode, thePosVal);
    }

    printf("%s\n", outCommand);

    return ok;
}

bool extractInt(const string inStr, int* outNum) {

    stringstream ss(inStr);
    string tmp;
    ss >> tmp >> *outNum;
    return true;
}

void processPtuComm() {

    string theResp;
    std::size_t theFound;

    if (Ptu->checkDataAndEnqueue()) {

        theResp.assign((char *)Ptu->getFullQueueContent(true));

        // Un signo de exclamación indica que se ha producido un error

        theFound = theResp.find("!");
        if (theFound != string::npos) {
            STATUS = IDLE;
            printf("PTU-46 ha devuelto un error: %s", theResp.c_str());
            return;
        }

        switch (STATUS) {

            case IDLE:

                // Recepción inesperada
                // Mostramos datos recibidos

                ROS_ERROR("PTU-46 ha enviado un dato inesperado: %s", theResp.c_str());

                break;

            case WAIT_COMMAND_CONF:

                theFound = theResp.find("**");
                if (theFound != string::npos) {
                    printf("PTU-46 ha confirmado la última orden:
%s", theResp.c_str());

                    STATUS = IDLE;
                }
            }
        }
    }
}

```

```

    }

    break;

case WAIT_POS_PAN:

    break;

case WAIT_POS_TILT:

    // Comprobamos que el comando se ha admitido
    // Y eliminamos el asterisco de confirmación

    int thePos;

    theFound = theResp.find("*");
    if (theFound != string::npos) {
        theResp.erase(0,theFound+1);
        STATUS = IDLE;
    }

    // Leemos el dato devuelto

    if (extractInt(theResp,&thePos)) {

        float theDeg;

        if (STATUS == WAIT_POS_PAN) {
            theDeg = (float)thePos * PAN_RESOLUTION / 3600;
            //thePtuJointState.position[0] = theDeg;
        } else if (STATUS == WAIT_POS_TILT) {
            theDeg = (float)thePos * TILT_RESOLUTION / 3600;
            //thePtuJointState.position[1] = theDeg;
        }

    }

    STATUS = IDLE;
    break;

default:

    STATUS = IDLE;
    break;

}

}

return;
}

void movePtu(float inPanDeg, float inTiltDeg) {

    char thePanCommand[16];
    char theTiltCommand[16];
    bool theBuildCommandOk = false;

    theBuildCommandOk = getPosCommand(inPanDeg,PAN,RELATIVE,thePanCommand);

    theBuildCommandOk = theBuildCommandOk && getPosCommand(inTiltDeg,TILT,RELATIVE,theTiltCommand);

    if (theBuildCommandOk) {
        if (Ptu->send(thePanCommand) > 0) {
            STATUS = WAIT_COMMAND_CONF;
            processPtuComm();
            //Ptu->send("A ");
        }
        if (Ptu->send(theTiltCommand) > 0) {
            STATUS = WAIT_COMMAND_CONF;
            processPtuComm();
            //Ptu->send("A ");
        }
    } else {
        // Error construyendo mensaje. no enviar nada
        // ...
    }

}

void ceroPtu() {

    Ptu->send("I "); // Modo inmediato
    usleep(200000);
    processPtuComm();
    Ptu->send("FT "); // Respuestas escuetas
    usleep(200000);
}

```

```

    processPtuParam();
    PtuParam->send("PP0 "); // Posición PAN 0
    usleep(200000);
    processPtuParam();
    PtuParam->send("A "); // Espera alcanzar las posiciones indicadas
    usleep(200000);
    processPtuParam();
    PtuParam->send("TP-300 "); // Posición TILT max
    usleep(200000);
    processPtuParam();
    PtuParam->send("A "); // Espera alcanzar las posiciones indicadas
    usleep(200000);
    processPtuParam();
    PtuParam->send("TP600 "); // Posición TILT max
    usleep(200000);
    processPtuParam();
    PtuParam->send("A "); // Espera alcanzar las posiciones indicadas
    usleep(200000);
    processPtuParam();
}

int main(int argc, char ** argv) {

    cout << "Inicializando PTU ..." << endl;

    // Init PTU-46

    PtuParam = new Serial::Serial Q(SERIALDEVICE, B9600);

    ceroPtuParam();

    usleep(500000);

    movePtuParam(0,-20);

    cout << "Inicializando OpenNI ..." << endl;
    theStatus = openni::OpenNI::initialize();

    if (theStatus != openni::STATUS_OK) {
        printf("Device open failed:\n%s\n", openni::OpenNI::getExtendedError());
        openni::OpenNI::shutdown();
        return 1;
    }

    cout << "Iniciando sensor de profundidad ...";

    openni::Array<openni::DeviceInfo> theDevices;
    openni::OpenNI::enumerateDevices(&theDevices);

    // Busca dispositivos compatibles
    // Inicializa la variable theDevice con cada uno
    // E imprime por pantalla su numero de serie

    for (int i = 0; i != theDevices.getSize(); ++i) {
        const openni::DeviceInfo& theDeviceInfo = theDevices[i];
        string uri = theDeviceInfo.getUri();
        theDevice.open(uri.c_str());
        char theSerialNumber[1024];
        theDevice.getProperty(ONI_DEVICE_PROPERTY_SERIAL_NUMBER, &theSerialNumber);
        cout << "Device " << i << ". Serial Number: " << theSerialNumber << endl;
    }

    // Crea un stream de cuadros a partir del dispositivo inicializado anteriormente
    // Se indica que el dispositivo es de tipo profundidad

    theStatus = theDepth.create(theDevice, openni::SENSOR_DEPTH);

    if (theStatus == openni::STATUS_OK) {
        theStatus = theDepth.start();
        if (theStatus != openni::STATUS_OK) {
            printf("Couldn't start depth stream:\n%s\n", openni::OpenNI::getExtendedError());
            theDepth.destroy();
        }
    } else {
        printf("Couldn't find depth stream:\n%s\n", openni::OpenNI::getExtendedError());
    }

    if (!theDepth.isValid()) {
        printf("No valid streams. Exiting\n");
        openni::OpenNI::shutdown();
        return 2;
    }

    openni::VideoFrameRef theRawFrame;
    Pixel3D theClosestPoint;
    Point theRealPoint;
    Point theOrigin;

    float thePanDeg, theTiltDeg, theDist;

```

```

// Por cada cuadro recibido se obtiene el punto más cercano y se muestra por pantalla
while(theDepth.isValid()){

    openni::Status rc = theDepth.readFrame(&theRawFrame);
    if (rc != openni::STATUS_OK) {
        printf("readFrame failed\n%s\n", openni::OpenNI::getExtendedError());
    } else {
        calculaPuntoMasCercano (&theClosestPoint, &theRawFrame);
        //theClosestPoint.print();

        openni::CoordinateConverter::convertDepthToWorld(theDepth, theClosestPoint.X, theClosestPoint.Y, theClosestPoint.Z, &theRealPoint.X, &theRealPoint.Y, &theRealPoint.Z);

        // Adaptamos las coordenadas para considerar el desajuste
        // entre las coordenadas de la cámara y las de la base pan-tilt

        theRealPoint.Y += OFFSET_CAMARA_EJE_TILT_MM / 1000;

        theRealPoint.print();
        Vector theV(theOrigin, theRealPoint);

        thePanDeg = theV.getPan()*180/PI;
        //theTiltDeg = theV.getTiltConsideringOffsetY(OFFSET_CAMARA_EJE_TILT_MM /
1000)*180/PI - 90;
        theTiltDeg = theV.getTiltConsideringOffsetY(OFFSET_CAMARA_EJE_TILT_MM /
1000)*180/PI;

        theDist = theV.getModulo();

        //if (theDist > 600) {
        if ((abs(thePanDeg) > 2) || (abs(theTiltDeg) > 2)) {
            movePtu(thePanDeg, theTiltDeg);
        }
        //}

        theV.printPanTiltDeg();

        usleep(1500000);
    }

}

theDepth.stop();
theDepth.destroy();

cout << "Terminando" << endl;
openni::OpenNI::shutdown();

return 0;
}

```

6.5. Modelo URDF simplificado Youbot

A continuación se muestra una posible descripción URDF simplificada del Kuka Youbot con comentarios para identificar el significado de los nodos y atributos utilizados.

Conviene indicar que para que este código sea funcional en simulación es necesario eliminar los comentarios XML (`<!-- .. -->`), ya que estos causan problemas en rviz.

```
<?xml version="1.0"?>
<robot name="youbotin">

  <!--
  Base rectangular 30x60x10 cm.
  Visualización elevada 5cm (la mitad de la altura) resp ejes coordenados
  puesto que las figuras geométricas en rviz tienen su origen en el centro
  -->
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.3 0.6 0.1"/>
      </geometry>
      <material name="azul">
        <color rgba="0 0 0.6 0.75"/>
      </material>
      <origin xyz="0 0 0.05" rpy="0 0 0"/>
    </visual>
  </link>

  <!--
  Plataforma giratoria cilíndrica de radio 10cm y altura 7.2cm
  Visualización elevada la mitad de la altura resp ejes
  -->
  <link name="link0">
    <visual>
      <geometry>
        <cylinder radius="0.1" length="0.072"/>
      </geometry>
      <material name="azul"/>
      <origin xyz="0 0 0.036" rpy="0 0 0"/>
    </visual>
  </link>

  <!--
  Unión de la base con el link cilíndrico
  centrada respecto eje X de la base, desplazada 15cm sobre el eje Y
  y con elevación igual al grosor de la base
  límites de giro entre -169° y 169° (pi/2) (eje de giro Z)
  -->
  <joint name="A1" type="revolute">
    <parent link="base link"/>
    <child link="link0"/>
    <origin xyz="0 0.15 0.1"/>
    <limit effort="1000.0" lower="-2.95" upper="2.95" velocity="0.5"/>
    <axis xyz="0 0 1"/>
  </joint>

  <!--
  primer link del brazo. Longitud 7.5cm
  visualización elevada la mitad de la altura
  -->
  <link name="link1">
    <visual>
      <geometry>
        <cylinder radius="0.04" length="0.075"/>
      </geometry>
      <material name="azul"/>
      <origin xyz="0 0 0.0375" rpy="0 0 0"/>
    </visual>
  </link>

  <!--
  Unión FIJA de la plataforma con el primer link
  desplazada 33cm sobre el eje Y respecto del eje de giro de la plataforma
  y con elevación igual a la longitud del link
  -->
  <joint name="fija" type="fixed">
    <parent link="link0"/>
    <child link="link1"/>
    <origin xyz="0 0.033 0.075"/>
  </joint>

  <!--
```

```

Segundo link del brazo. Longitud 15.5cm
visualización elevada la mitad de la altura
-->
<link name="link2">
  <visual>
    <geometry>
      <cylinder radius="0.02" length="0.155"/>
    </geometry>
    <material name="azul"/>
    <origin xyz="0 0 0.0775" rpy="0 0 0"/>
  </visual>
</link>

<!--
Unión de los links primero y segundo del brazo
Elevación igual a la longitud del primer link (7.5cm)
Límites de giro -65° a 90° (eje de giro X)
-->
<joint name="A2" type="revolute">
  <parent link="link1"/>
  <child link="link2"/>
  <origin xyz="0 0 0.075" rpy="0 0 0"/>
  <limit effort="1000.0" lower="-1.134" upper="1.57" velocity="0.5"/>
  <axis xyz="1 0 0"/>
</joint>

<!--
Tercer link del brazo. Longitud 13.5cm
visualización elevada la mitad de la altura
-->
<link name="link3">
  <visual>
    <geometry>
      <cylinder radius="0.02" length="0.135"/>
    </geometry>
    <material name="azul"/>
    <origin xyz="0 0 0.0675" rpy="0 0 0"/>
  </visual>
</link>

<!--
Unión entre segundo y tercer links del brazo
Elevación igual a la longitud del segundo link
Límites de giro -151° a 146° (eje de giro X)
-->
<joint name="A3" type="revolute">
  <parent link="link2"/>
  <child link="link3"/>
  <origin xyz="0 0 0.155" rpy="0 0 0"/>
  <limit effort="1000.0" lower="-2.635" upper="2.548" velocity="0.5"/>
  <axis xyz="1 0 0"/>
</joint>

<!--
Cuarto link del brazo. Longitud 8.1cm
visualización elevada la mitad de la altura
-->
<link name="link4">
  <visual>
    <geometry>
      <cylinder radius="0.02" length="0.081"/>
    </geometry>
    <material name="azul"/>
    <origin xyz="0 0 0.0405" rpy="0 0 0"/>
  </visual>
</link>

<!--
Unión entre tercer y cuarto links del brazo
Elevación igual a la longitud del tercer link
Límites de giro -102.5° a 102.5° (eje de giro X)
-->
<joint name="A4" type="revolute">
  <parent link="link3"/>
  <child link="link4"/>
  <origin xyz="0 0 0.135" rpy="0 0 0"/>
  <limit effort="1000.0" lower="-1.789" upper="1.789" velocity="0.5"/>
  <axis xyz="1 0 0"/>
</joint>

<!--
Quinto link
Longitud 9cm.. Visualización elevada la mitad de la longitud
-->
<link name="link5">
  <visual>
    <geometry>
      <box size="0.08 0.04 0.09"/>
    </geometry>

```

```
<material name="azul"/>
<origin xyz="0 0 0.045" rpy="0 0 0"/>
</visual>
</link>

<!--
Unión entre cuarto y quinto links del brazo
Elevación igual a la longitud del cuarto link
Límites de giro -167.5° a 167.5° (eje de giro Z)
-->
<joint name="A5" type="revolute">
  <parent link="link4"/>
  <child link="link5"/>
  <origin xyz="0 0 0.081" rpy="0 0 0"/>
  <limit effort="1000.0" lower="-2.923" upper="2.923" velocity="0.5"/>
  <axis xyz="0 0 1"/>
</joint>

<!--
Primer link de la pinza (simplificado)
Longitud 4.7cm
-->
<link name="gripper1">
  <visual>
    <geometry>
      <box size="0.01 0.02 0.047"/>
    </geometry>
    <material name="azul"/>
    <origin xyz="0 0 0.0235" rpy="0 0 0"/>
  </visual>
</link>

<!--
Segundo link de la pinza (simplificado)
Longitud 4.7cm
-->
<link name="gripper2">
  <visual>
    <geometry>
      <box size="0.01 0.02 0.047"/>
    </geometry>
    <material name="azul"/>
    <origin xyz="0 0 0.0235" rpy="0 0 0"/>
  </visual>
</link>

<!--
Unión fija entre el quinto link y pinza
-->
<joint name="G1" type="fixed">
  <parent link="link5"/>
  <child link="gripper1"/>
  <origin xyz="-0.02 0 0.09" rpy="0 0 0"/>
  <limit effort="1000.0" lower="-2.923" upper="2.923" velocity="0.5"/>
  <axis xyz="0 0 1"/>
</joint>

<!--
Unión prismática entre links pinza
-->
<joint name="G2" type="prismatic">
  <parent link="gripper1"/>
  <child link="gripper2"/>
  <origin xyz="0.02 0 0" rpy="0 0 0"/>
  <limit effort="1000.0" lower="0" upper="0.02" velocity="0.5"/>
  <axis xyz="1 0 0"/>
</joint>

</robot>
```

6.6. Programa Seguimiento con cámara USB, OpenCV y actuadores Dynamixel

6.6.1. INTERFAZ DE USUARIO

Consiste en la presentación de la imagen y varias barras deslizantes que permiten al usuario seleccionar los límites HSV para sintonizar el color de identificación. La pulsación de la letra ‘t’ activa o desactiva el modo de seguimiento del mayor objeto identificado.

Mediante superposición de gráficos y texto, se aporta información de estado (seguimiento activado, ángulos de corrección y centro de referencia).

6.6.2. IDENTIFICACIÓN DE OBJETOS

Captura y procesa la imagen procedente de la cámara USB mediante la librería OpenCV, con una frecuencia de captura limitada a 30 Hz.

1. La función *colorFilter()* realiza el filtrado HSV de acuerdo a los parámetros seleccionados por el usuario, así como las operaciones morfológicas de erosión y dilatación, devolviendo como resultado una imagen binaria (blanco indicando cumple, negro no cumple).
1. La función *findObjects()* obtiene un vector de contornos a partir de la imagen binaria obtenida en la función anterior e identifica el centro y el área del mayor contorno detectado (o indica mediante área 0 que ninguna de las detecciones alcanza los valores mínimos establecidos).

6.6.3. DETERMINACIÓN DE LOS ÁNGULOS DE CORRECCIÓN

Los ángulos de corrección se obtienen a partir del centroide de la imagen identificada, los parámetros intrínsecos de la cámara y la posición instantánea de los actuadores. La ejecución de acciones de control está limitada para no saturar el bus de actuadores.

Se aplica una histéresis equivalente al 5% del rango en cada eje para evitar vibraciones una vez el objeto está enfocado.

6.6.4. COMUNICACIÓN MOTORES

Mediante suscripción al *topic /motor_states/pan_tilt_port* se leen los estados de los actuadores Dynamixel (la frecuencia de publicación nos viene impuesta y es de 20Hz).

Para ejecutar las acciones de control se emiten mensajes de posición en los *topics /pan_joint/command* y */tilt_joint/command* para los actuadores *pan* y *tilt* respectivamente.

```

/*
 * Programa para seguir un objeto de color.
 * Autor: Fernando Soto.
 * A partir de las siguientes referencias y ejemplos de código:
 * - http://opencv-srf.blogspot.com.es/2010/09/object-detection-using-color-seperation.html
 * - Real-Time Object Tracking Using OpenCV: https://www.youtube.com/watch?v=bSeFrPrqZ2A
 */

#include <stdio.h>
#include <iostream>
#include <string>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <ros/ros.h>
#include <std_msgs/Float64.h>
#include <dynamixel_msgs/MotorStateList.h>
#include <dynamixel_msgs/MotorState.h>

#define PI 3.1415927

#define FRAME_HEIGHT 480
#define FRAME_WIDTH 640

#define CALIBRATION_FILE_PATH "../camera.yaml"

using namespace cv;
using namespace std;

double currentPanPosRad = 0.0;
double currentTiltPosRad = 0.0;

Mat colorFilter(const Mat* src, Scalar inColorMinHSV, Scalar inColorMaxHSV);
vector< vector<Point> > findObjects(Mat* inFrame, Scalar inColorMinHSV, Scalar inColorMaxHSV, Point2f*
outLargestMatchCenter, float* outLargestMatchArea);
string intToString(int number);
string floatToString(float number);
double digiPos2Deg(int inPos);

bool IsTracking = false;

Scalar HsvMin;
Scalar HsvMax;

std_msgs::Float64 thePanMsg, theTiltMsg;

// Funciones callback para ajuste de límites por pantalla

void onHsvHMin(int theSliderValue, void*) { HsvMin[0] = theSliderValue*180/360; }
void onHsvHMax(int theSliderValue, void*) { HsvMax[0] = theSliderValue*180/360; }
void onHsvSMin(int theSliderValue, void*) { HsvMin[1] = theSliderValue*255/100; }
void onHsvSMax(int theSliderValue, void*) { HsvMax[1] = theSliderValue*255/100; }
void onHsvVMin(int theSliderValue, void*) { HsvMin[2] = theSliderValue*255/100; }
void onHsvVMax(int theSliderValue, void*) { HsvMax[2] = theSliderValue*255/100; }

void motorStatesCallback(const dynamixel_msgs::MotorStateList& inMsg);

int theFrameCounter = 0;

int main( int argc, char** argv ) {

```

```

VideoCapture theCapture(1);
Mat theFrame, theFrame2;
vector< vector<Point> > theBallContours;
Point2f theCentro;
float theArea = 0.0;

// Leemos los parámetros intrínsecos de la cámara
// y los coeficientes de distorsión del archivo de calibración

/*
// Problema con el formato generado por ROS.
// No coincide con el esperado por OpenCV a través de FileStorage

Mat theCameraMatrix, theDistCoeffs;
FileStorage theCalibrationFile(CALIBRATION_FILE_PATH, FileStorage::READ);
theCalibrationFile["cameraMatrix"] >> theCameraMatrix;
theCalibrationFile["distCoeffs"] >> theDistCoeffs;
theCalibrationFile.release();
*/

float theCM[] = {1107.46545955351, 0, 321.746543160967, 0, 1109.96438787448, 291.303399490763, 0, 0,
1};
float theDC[] = {-0.0974330899674774, 0.479179593078898, 0.0207048507870403, 0.00531414762112178,
0};
float thePC[] = {1102.72436035156, 0, 322.944612208022, 0, 0, 1098.22875, 295.148907870075, 0, 0, 0,
1, 0};

Mat theCameraMatrix = Mat(3, 3, CV_32F, theCM).clone();
Mat theDistCoeffs = Mat(1, 5, CV_32F, theDC).clone();
Mat theProjectionMatrix = Mat(3, 4, CV_32F, thePC).clone();

float fx = theCameraMatrix.at<float>(0,0);
float fy = theCameraMatrix.at<float>(1,1);
float cx = theCameraMatrix.at<float>(0,2);
float cy = theCameraMatrix.at<float>(1,2);

cout << "Parámetros intrínsecos: c=(" << cx << ", " << cy << "), fx=" << fx << ", fy=" << fy << endl;

Scalar theTargetIndicatorColor(255,0,00); // BGR Azul

// Rangos de representación HSV en OpenCV: 0<=H<=180, 0<=S<=255, 0<=V<=255

HsvMin[0] = 4; // H min
HsvMin[1] = 34; // S min
HsvMin[2] = 169; // V min

HsvMax[0] = 18; // H min
HsvMax[1] = 218; // S min
HsvMax[2] = 255; // V min

if(!theCapture.isOpened()) {
    cout << "Error abriendo captura de imagen" << endl;
    return -1;
}

ros::init(argc, argv, "pan_tilt_ptu_46");
ros::NodeHandle theNodeHandle;
ros::Publisher theTiltPublisher =

```

```

theNodeHandle.advertise<std_msgs::Float64>("/tilt_joint/command",10);
    ros::Publisher thePanPublisher =
theNodeHandle.advertise<std_msgs::Float64>("/pan_joint/command",10);
    ros::Subscriber theMotorStatesSubscriber = theNodeHandle.subscribe("/motor_states/pan_tilt_port",
10, motorStatesCallback);

    thePanMsg.data = 0.0;
    thePanPublisher.publish(thePanMsg);

    theTiltMsg.data = 0.0;
    theTiltPublisher.publish(theTiltMsg);

    usleep(500000);

    namedWindow("Video",CV_WINDOW_AUTOSIZE);

    createTrackbar("H min", "Video", NULL, 360, &onHsvHMin);
    createTrackbar("H max", "Video", NULL, 360, &onHsvHMax);
    createTrackbar("S min", "Video", NULL, 100, &onHsvSMin);
    createTrackbar("S max", "Video", NULL, 100, &onHsvSMax);
    createTrackbar("V min", "Video", NULL, 100, &onHsvVMin);
    createTrackbar("V max", "Video", NULL, 100, &onHsvVMax);

    //createButton("Track",&onChkTrackChanged,NULL,CV_CHECKBOX,0); // Qt NO soportado por la instalación
de OpenCV de ROS

    setTrackbarPos("H min", "Video", 213);
    setTrackbarPos("H max", "Video", 322);
    setTrackbarPos("S min", "Video", 12);
    setTrackbarPos("S max", "Video", 98);
    setTrackbarPos("V min", "Video", 17);
    setTrackbarPos("V max", "Video", 100);

    float theLoopRate = 30;

    //int theRefCenterX = (int)cx;
    int theRefCenterX = FRAME_WIDTH/2;
    //int theRefCenterY = (int)cy;
    int theRefCenterY = FRAME_HEIGHT/2;

    ros::Rate theRosRate(theLoopRate); // 30 hz

    while(theNodeHandle.ok()) {

        // Obtenemos una imagen

        theCapture >> theFrame;

        // (Opcional)Corregimos la imagen a partir de los parámetros de calibración conocidos
        //undistort(theFrame2, theFrame, theCameraMatrix, theDistCoeffs);
        //theFrame = theFrame2.clone();

        // Ejecutamos función de detección.
        // La función devolverá un vector de contornos con todos los candidatos
        // Las variables theCentro y theArea contienen las coordenadas (en pixeles) del candidato
con mayor área

        theBallContours = findObjects(&theFrame, HsvMin, HsvMax,&theCentro,&theArea);

```

```

        if (theArea > 0) {

            // Si la detección ha sido satisfactoria
            // marcamos el objetivo con un círculo y una cruz en el centroide

            //circle(theFrame,theCentro,sqrt(theArea/PI),theTargetIndicatorColor);
            circle(theFrame,theCentro,20,theTargetIndicatorColor);
            line(theFrame,Point(theCentro.x-10, theCentro.y), Point(theCentro.x+10,
theCentro.y), theTargetIndicatorColor);
            line(theFrame,Point(theCentro.x, theCentro.y-10), Point(theCentro.x,
theCentro.y+10), theTargetIndicatorColor);

            // Señalar el centro cx,cy con una cruz
            line(theFrame,Point(theRefCenterX-10, theRefCenterY), Point(theRefCenterX+10,
theRefCenterY), Scalar(0,0,0));
            line(theFrame,Point(theRefCenterX, theRefCenterY-10), Point(theRefCenterX,
theRefCenterY+10), Scalar(0,0,0));
            //line(theFrame,Point(FRAME_WIDTH/2-10, FRAME_HEIGHT/2), Point(FRAME_WIDTH/2+10,
FRAME_HEIGHT/2), Scalar(0,0,0));
            //line(theFrame,Point(FRAME_WIDTH/2, FRAME_HEIGHT/2-10), Point(FRAME_WIDTH/2,
FRAME_HEIGHT/2+10), Scalar(0,0,0));

            // Añadimos texto con las coordenadas

            putText(theFrame,intToString(theCentro.x)+"",intToString(theCentro.y),Point(theCentro.x,theCentro.y
+30),1,1,Scalar(0,255,0),2);

            // Representamos el contorno de todos los candidatos

            drawContours(theFrame,theBallContours,-1,theTargetIndicatorColor);

            //cout << "X: " << theCentro.x << ", Y: " << theCentro.y << ", area: " << theArea
<< endl;

            // Tratar de entender cómo obtener la corrección en radianes
            // http://stackoverflow.com/questions/13957150/opencv-computing-camera-position-
rotation

            // Ajuste de ángulos pan y tilt:
            // El centro de la imagen (en pixels) viene dado por los parámetros intrínsecos cx
y cy
            // (en caso de no disponer de información de calibración tomaríamos FRAME_WIDTH/2 y
FRAME_HEIGHT/2)
            // La distancia focal (en pixels) viene dado por los parámetros de calibración fx y
fy

            // Ajuste del ángulo PAN
            // a partir de la coordenada X y los parámetros cx y fx: atan2(x-cx,fx)

            double thePanCorrection = -atan2(theCentro.x-theRefCenterX,fx);

            thePanMsg.data = currentPanPosRad + thePanCorrection;
            //thePanMsg.data = -atan2(theCentro.x-FRAME_WIDTH/2,fx);
            putText(theFrame,"PAN:
"+intToString(round(thePanCorrection*180/PI)),Point(FRAME_WIDTH/2 - 20, 10),1,1,Scalar(0,255,0),2);

            // Ajuste del ángulo TILT

```

```

        // a partir de la coordenada Y y los parámetros cy y fx: atan2(y-cy,fx)

        double theTiltCorrection = atan2(theCentro.y-theRefCenterY, fy);

        theTiltMsg.data = currentTiltPosRad + theTiltCorrection;

        //theTiltMsg.data = atan2(theCentro.y-FRAME_HEIGHT/2, fy);
        putText(theFrame, "TILT:
"+intToString(round(theTiltCorrection*180/PI)), Point(10, FRAME_HEIGHT/2 - 20), 1, 1, Scalar(0, 255, 0), 2);

        // Indicar si estamos en modo tracking

        if (IsTracking) putText(theFrame, "TRACKING", Point(FRAME_WIDTH - 100, FRAME_HEIGHT -
10), 1, 1, Scalar(0, 255, 0), 2);

        // Limitar las acciones de control para no saturar el bus de motores

        if (theFrameCounter >= 5) {

            theFrameCounter = 0;

            if ((IsTracking)&&(abs(theRefCenterX - (int)theCentro.x) > FRAME_WIDTH/20))
{
                ROS_INFO_STREAM("Comando PAN: " << thePanMsg.data << " rad / " <<
round(thePanMsg.data*180/PI) << " grados aprox.");
                thePanPublisher.publish(thePanMsg);
                usleep(200000);
            }

            if ((IsTracking)&&(abs(theRefCenterY - (int)theCentro.y) >
FRAME_HEIGHT/20)) {
                ROS_INFO_STREAM("Comando TILT: " << theTiltMsg.data << " rad / "
<< round(theTiltMsg.data*180/PI) << " grados aprox.");
                theTiltPublisher.publish(theTiltMsg);
            }
        }
    }

    // Mostramos por pantalla la imagen modificada con la información de detección
    imshow("Video", theFrame);

    // Pausa hasta que el usuario pulse tecla o transcurran X milisegundos
    // Solo importa la letra 't' para hacer un toggle del modo tracking
    if (waitKey(1000/theLoopRate) == 116) {
        IsTracking = (IsTracking)?false:true;
    }
    theRosRate.sleep();
    ros::spinOnce();
    theFrameCounter++;
}
return 0;
}

Mat colorFilter(const Mat* inFrame, Scalar inColorMinHSV, Scalar inColorMaxHSV) {

    Mat theHsvFrame, outThresholdedFrame;

    // Comprobamos que el formato de imagen es el esperado
    assert(inFrame->type() == CV_8UC3);
    // Obtenemos la representación HSV de la imagen BGR (ojo, BGR y no RGB!!)

```

```

// ya que HSV presenta mejores características para operaciones de filtrado de color
cvtColor(*inFrame,theHsvFrame,CV_BGR2HSV);

// Filtrado en HSV
inRange(theHsvFrame, inColorMinHSV, inColorMaxHSV, outThresholdedFrame);

// Definimos kernel para operaciones de erosión y dilatación
// Los elementos rectangulares implican menor carga computacional
Mat theErodeElement = getStructuringElement(MORPH_RECT, Size(3, 3));
Mat theDilateElement = getStructuringElement(MORPH_RECT, Size(8, 8));

// Invertir
//bitwise_not(theThresholdedFrame, theThresholdedFrame);
// Erosión.
// Eliminamos el ruido (conicidencias dispersas y aisladas de pequeño tamaño)
for(int i=0;i<2;i++)
    erode(outThresholdedFrame, outThresholdedFrame, theErodeElement);

// Dilatación
// Reforzamos las detecciones que han sobrevivido a la erosión
for(int i=0;i<2;i++)
    dilate(outThresholdedFrame, outThresholdedFrame, theDilateElement);

return outThresholdedFrame;
}

vector< vector<Point> > findObjects(Mat* inFrame, Scalar inColorMinHSV, Scalar inColorMaxHSV, Point2f*
outCentro, float* outArea) {

vector<vector<Point> > outContours;
vector<Vec4i> theHierarchy;
vector<Point> theLargestTargetContour;
CvMoments theMoments;
int theTargetContourNum = 0;
double theMaxArea = 0.0;

// Aplicamos filtro de búsqueda de bolas del co
Mat theThresholdedImage = colorFilter(inFrame, inColorMinHSV, inColorMaxHSV);

// Buscamos contornos
findContours(theThresholdedImage, outContours, theHierarchy,CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);

// Comprobamos el número de contornos detectados.
// En caso de haber más de uno establecemos el blanco en el de mayor área.
// Si el número de detecciones es mayor que 5 entendemos que es debido a una detección errónea
int theNumObjects = theHierarchy.size();

//if ((theNumObjects > 0)&&(theNumObjects < 15)) {
if ((theNumObjects > 0)) {
    // Iteramos sobre los objetos detectados

    for (int theIndex = 0; theIndex >= 0; theIndex = theHierarchy[theIndex][0]) {
        float theTmpArea = contourArea(outContours[theIndex]);
        if (theTmpArea > theMaxArea) {
            theTargetContourNum = theIndex;
            theMaxArea = theTmpArea;
        }
    }
}
}
}

```

```

// Si se ha seleccionado un objetivo con un area aceptable (area de la imagen entre 128)
// obtenemos su centro de masa a partir del vector de momentos
// Si no, devolvemos area 0 y punto -1,-1 para indicar que no consideramos detección
if (theMaxArea > FRAME_HEIGHT*FRAME_WIDTH/128) {
    theLargestTargetContour = outContours[theTargetContourNum];
    theMoments = moments(theLargestTargetContour, false);
    *outCentro = Point2f(theMoments.m10/theMoments.m00,theMoments.m01/theMoments.m00);
    *outArea = theMaxArea;
} else {
    *outCentro = Point2f(-1,-1);
    *outArea = 0.0;
}
return outContours;
}
string intToString(int number) {
    std::stringstream ss;
    ss << number;
    return ss.str();
}
string floatToString(float number) {
    std::stringstream ss;
    ss << number;
    return ss.str();
}
double digiPos2Deg(int inPos) {
    return -150.0 + (float)inPos*300.0/1023.0;
}

void motorStatesCallback(const dynamixel_msgs::MotorStateList& inMsg) {
    static int i = 0;

    // Descartamos 9 de cada 10 mensajes para no ralentizar el proceso
    if (i >= 10) {

        i = 0;

        for(int i=0;i<2;i++) {
            dynamixel_msgs::MotorState theMotorState = inMsg.motor_states[i];
            float thePosDeg = digiPos2Deg(theMotorState.position);
            float thePosRad = thePosDeg*PI/180.0;
            float theGoalDeg = digiPos2Deg(theMotorState.goal);
            float theGoalRad = theGoalDeg*PI/180.0;
            int isMoving = theMotorState.moving;
            string theMotorName = (i==0)?"PAN":"TILT";

            if (i==0) {
                currentPanPosRad = thePosRad;
            } else {
                currentTiltPosRad = thePosRad;
            }

            ROS_INFO_STREAM("Motor " << theMotorName << ". Pos actual: " << thePosRad << " rad
(" << thePosDeg << " grados). Goal: " << theGoalRad << " rad ( " << theGoalDeg << " grados). " << ((isMoving
== 0)?"Parado":"En movimiento"));
        }
        i++;
    }
}

```

Bibliografía

- [1] Sitio Web de Ros. <http://www.ros.org>
- [2] ROS by Example FUERTE Vol 1. *R. Patrick Goebel*.
- [3] Learning OpenCV. *Gary Bradski, Adrian Kaehler. O'Reilly*.
- [4] Serial Programming HowTo <http://www.tldp.org/HOWTO/pdf/Serial-Programming-HOWTO.pdf>
- [5] Basics of Serial Communications. <http://ulisse.elettra.trieste.it/services/doc/serial/>
- [6] Serial Programming/termios. http://en.wikibooks.org/wiki/Serial_Programming/termios
- [7] Apuntes Montaje y Verificación de Prototipo. I. Álvarez. <http://isa.uniovi.es/~ialvarez/Curso/Mecatronica/>
- [8] Arduino. Curso Práctico de Formación. *Óscar Torrente Artero. RC Libros*.
- [9] Sitio web de Arduino. <http://arduino.cc/>
- [10] Sitio web del Observatorio Tecnológico MECD. <http://recursostic.educacion.es>
- [11] Programación Orientada a Objetos en C++. *Ricardo Devís Botella. Paraninfo*
- [12] The C++ Programming Language. *Bjarne Stroustrup*. 3rd Ed. AddisonWesley
- [13] The Linux Documentation Project. <http://www.tldp.org>
- [14] Linux in a Nutshell. O'Reilly. *Ellen Siever; Aaron Weber; Stephen Figgins; Robert Love; Arnold Robbins*
- [15] Eclipse C++ Tutorial. <http://cs.txstate.edu/~redwingb/tutorial.pdf>
- [16] Wikipedia Control de Versiones. http://es.wikipedia.org/wiki/Control_de_versiones
- [17] Control de versiones con Subversion. <http://svnbook.red-bean.com/>
- [18] Documentación Oficial Git. <http://git-scm.com/documentation>
- [19] Documentación OpenCV. <http://docs.opencv.org/>
- [20] Sitio Oficial OpenNI. <http://www.openni.org>
- [21] Sitio web de Derek Molloy. <http://derekmolloy.ie/>