UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

Seleccionar titulación

DESARROLLO DE UNA APLICACIÓN DE POSICIONAMIENTO BASADA EN MARCADORES PARA UN ROBOT MÓVIL.

AUTOR: MUÑOZ ARNAL, BLAI

TUTOR: VALLÉS MIQUEL, MARINA

COTUTOR: VALERA FERNÁNDEZ, ÁNGEL

Curso Académico: 2015-16

RESUMEN

El objetivo del presente trabajo es crear una aplicación que sirva para posicionar robots móviles basándose únicamente en el reconocimiento de marcadores. Paralelamente al desarrollo práctico se ha redactado un documento con el contenido teórico y una descripción del trabajo práctico que permiten la reproducción del mismo.

Para crear la aplicación primero se estudiarán las características principales de los robots y los sistemas actuales de posicionamiento para entender su funcionamiento, sus ventajas y fuentes de error. El siguiente paso será establecer cómo se pueden mejorar y buscar las herramientas necesarias para implementar el reconocimiento de los marcadores. Finalmente se estudiarán las características del Summit XL, robot con el que trabajará, para finalmente implementar la aplicación en el mismo.

Aprendidos los conceptos teóricos se procederá a aprender a utilizar el sistema operativo, Ubuntu, y el middleware, ROS, con los que se trabajará. Se procederá a instalar las librerías necesarias y leer su documentación para entender su funcionamiento. En algunos casos ha sido necesario realizar algunas modificaciones para adaptar la librería a la aplicación objetivo del proyecto.

Finalmente se desarrollan las fórmulas que permiten calcular el posicionamiento del robot y se integran en una aplicación. Se enviará esta aplicación al Summit XL donde será puesta a prueba para verificar su correcto funcionamiento. Al final del documento se encuentran las conclusiones y futuros proyectos que se pueden hacer a partir de la aplicación creada y una serie de anexos.

ÍNDICE

DOCUMENTOS CONTENIDOS TFG

- Memoria.
- Presupuesto

ÍNDICE DE LA MEMORIA

CAPÍTULO 1. INTRODUCCIÓN

Un Ingeniero Industrial es un profesional con amplios conocimientos en muchos campos diferentes, desde el diseño de naves industriales a la dirección y gestión de proyectos. Un campo muy interesante es el de la robótica, donde la investigación permite diseñar nuevos robots más sofisticados capaces de realizar tareas más complejas en un entorno dinámico que evoluciona diariamente.

En el campo de la robótica móvil un problema delicado es el posicionamiento de los robots. Es necesario dotar a los mismos con un hardware y software con la capacidad de posicionar al robot en un mapa. Existen dos grandes corrientes para hacer esto: utilizar un sistema de geolocalización (GPS) o el Dead Reckoning. Los dos sistemas tienen grandes ventajas e inconvenientes.

El GPS comete un error muy grande dentro de edificios, túneles o en zonas geográficas con mala conexión a los satélites. El Dead Reckoning por su naturaleza comete un error que aumenta con el tiempo. Estas fuentes de errores de cada sistema se explicarán con más detalle en el presente documento más adelante.

1.1. Objetivo del trabajo.

El objetivo principal de este proyecto es crear una aplicación de posicionamiento de robots móviles mediante el reconocimiento de marcadores..

El primer paso será entender el funcionamiento básico de Ubuntu y ROS, herramientas que hasta el inicio de este proyecto no se habían gastado antes. Una vez adquiridos estos conocimientos básicos se procederá a seleccionar e instalar una librería de realidad aumentada que tenga la capacidad de reconocer marcadores. Será necesario pues entender cómo funcionan los paquetes, nodos y topics en ROS.

Con el reconocimiento de marcadores se obtendría también la posición relativa del marcador respecto a la cámara. Esta información es vital para el proyecto pues será la información que gastará un nodo a desarrollar que se encargará de calcular la posición relativa del robot respecto al marcador. Finalmente ese mismo nodo leerá, de alguna manera que se determinará llegado el momento, la posición absoluta de los marcadores en el mapa y a partir

de esta información y la posición relativa del robot respecto al marcador calculará la posición absoluta del robot en el mapa.

Un resumen de todos los objetivos del proyecto de manera esquemática es:

- Instalar y entender el funcionamiento de Ubuntu y ROS.
- Seguir los tutoriales de ROS para adquirir los conocimientos para crear aplicaciones en ROS.
- Descargar e instalar las librerías necesarias para el reconocimiento de marcadores.
- Comprobar que el reconocimiento funciona correctamente.
- Desarrollar la parte de cálculo matemático.
- Crear una aplicación que implemente el cálculo matemático.
- Comprobar que el posicionamiento funciona correctamente.

1.2. Estructura del documento.

En el presente documento se plantea un problema y se estudian diversas alternativas para su solución. Al tratarse de un proyecto industrial se ha decidido seguir la estructura típica de los mismos pero con algunas modificaciones perfectamente justificadas. Como es un proyecto académico tan comprimido en el tiempo solo se desarrollarán dos documentos: la memoria y el presupuesto.

Se ha prescindido pues a redactar el pliego de condiciones y no se incluyen planos en el documento. La estructura de cada documento es la siguiente:

Memoria:

- Introducción objetivos: breve justificación de la importancia de la robótica en la industria y descripción de los objetivos de este trabajo y de los conocimientos teóricos necesarios.
- Desarrollo teórico: ante de estudiar los diferentes sistemas de posicionamiento es necesario conocer las características de un robot, los diferentes tipos (se darán las características concretas del Summit XL, robot con el que se trabajará) y finalmente se describirán los conocimientos teóricos necesarios y las herramientas empleadas para programar uno. Entonces se procederá a estudiar los diferentes sistemas de posicionamiento existentes presentando sus ventajas e inconvenientes. Finalmente se introducirá el concepto Visión Artificial.
- Desarrollo práctico: se explicará con más detalle la solución a desarrollar, es decir, el sistema de soporte para el posicionamiento del robot. Se presentará detalladamente todos los pasos adoptadas para llegar a la solución final. En este apartado de la

memoria también se explicará el funcionamiento de las diferentes partes de la solución.

- Conclusión y trabajos futuros: se analizará la solución y si realmente es práctica para el posicionamiento de los robots. Además se presentará una lista con todas las posibles futuras aplicaciones del trabajo a realizar y su posible impacto en la sociedad actual.
- Bibliografía.

Presupuesto

Al tratarse de un proyecto de ingeniería, aunque sea en el ámbito académico, debe adjuntarse un presupuesto en el que quede perfectamente reflejado el coste del mismo.

CAPÍTULO 2: BASE TEÓRICA.

El presente trabajo utiliza conocimientos de muchas ramas de la ingeniería, por lo tanto es necesario tener unos conocimientos básicos sobre cada una de las tecnologías que se van a implementar en el desarrollo del proyecto. Para asegurar este hecho se realizará un estudio sobre los diferentes campos implicados en el proyecto.

El objetivo del estudio teórico no solo es asegurar una total comprensión de las diferentes tecnologías a implementar, también permitirá adquirir el criterio necesario para tomar las decisiones pertinentes a lo largo del desarrollo del proyecto.

Se procede pues a explicar las características básicas y de funcionamiento de los robots, el middleware, ROS, que trabaja sobre Ubuntu, los diferentes sistemas de posicionamiento de robots (con sus respectivas ventajas y desventajas), la visión artificial y la realidad aumentada para finalmente introducir los conceptos básicos de la comunicación entre el ordenador y el robot.

2.1. Robótica: ¿qué es un robot y para qué se utilizan?

Un robot es una máquina automática en la que podemos descargar un programa. Gracias a ésto el robot será capaz de realizar funciones complejas que le permiten sustituir a un operario humano. La principal ventaja de esto es la eliminación de trabajos que por su naturaleza presentan peligros para que los realice una persona.

Estos programas son modificables y se pueden cambiar en función del trabajo que se desee que realice el robot. Es importante destacar que este no tiene porque tener aspecto humano ni realizar funciones parecidas a las de un humano, aunque si hay robots que están específicamente diseñados para parecer y actuar como uno.

Dentro del término robótica, que es más general, se agrupa la especialidad de la ingeniería encargada del diseño, construcción y control de robots. [1]

2.1.1. Breve introducción histórica del robot industrial.

Existe una clasificación muy general de los robots dividiendo a éstos en dos: manipuladores y robots móviles, las características de ambos se explicarán más adelante, antes se hablará de los primeros robots en la industria .

El primer robot manipulador en la industria fue creado por el ingeniero americano George Devol en 1954. Más tarde, en 1956, Unimation Inc. lo desarrolló para darle un uso comercial. En 1959 se introduce un primer prototipo de la compañía en una fábrica de Generals Motors y en 1961 se lanza la primera línea de producción con robots.

Con el tiempo y la mejora en la tecnología se han creado robots más complejos. Esto ha permitido diseñar robots no solo con funciones más complejas y más rápidos sino también implementar cadenas de montaje prácticamente automatizadas, reduciendo notablemente los costes de producción.

El primer robot industrial móvil también llegó en 1954. Barrett Electronics Corp. creó un coche eléctrico para un almacén de alimentos. Con el paso del tiempo se han creado nuevas tecnologías (como la tecnología GPS o las cámaras de visión 3D) que han permitido crear robots móviles mucho más complejos. [2]

En la actualidad las grandes empresas tecnológicas como Apple, Google y Facebook están invirtiendo miles de millones en el desarrollo de coches con conducción completamente autónoma, es decir, sin intervención de un conductor humano.

2.2.2. Tipologías.

Hay muchas clasificaciones de robots diferentes. Pueden ser clasificados por la tecnología que tienen implementada o por su morfología. También diferentes organizaciones tienen sus propias clasificaciones. Por ejemplo: JIRA (Asociación de Robots Japonesa) [3] o AFRI (Asociación Francesa de Robótica Industrial). [4]

En la siguiente sección se explicará las características de los robots manipuladores y móviles. El objetivo de este trabajo es crear un programa para después probarlo en el robot Summit XL, por lo que se dedicará una sección a hablar de este robot.

2.2.2.1. Robot manipulador.

Robot de aspecto antropomórfico (normalmente imitan un brazo) o cartesianos (ejes). Están anclados al suelo o a una estructura por uno de sus extremos, condicionados así a no poder desplazarse. Se implementan principalmente en cadenas de montaje automatizadas. Estos robots están constituidos por una serie de piezas articuladas entre sí. Gracias a esto están dotados de una serie de movimientos o grados de libertad que les permiten realizar una serie de tareas más o menos complejas. Son estos movimientos lo que se programan para que el robot realice la tarea o tareas requeridas en cada caso. [5]

Figura 1: Robot manipulador antropomórfico. [6]

Figura 2: Robot manipulador cartesiano. [7]

2.2.2.2. Robot móvil.

A diferencia de los robots manipuladores los robots móviles se pueden desplazar en el espacio, no están anclados a un punto fijo. Esto los dota de una gran utilidad pero complica bastante tanto su diseño como su programación.

Gracias a las nuevas cámaras 2D y 3D y los nuevos sistemas de geolocalización, los robots móviles pueden ser programados para desplazarse por cualquier territorio en cualquier lugar. Actualmente se comercializan robots capaces de moverse por el agua, por tierra y por el aire. Este documento se centrará en los robots que se desplazan por tierra utilizando ruedas.

Dentro de los robots móviles se diferencian dos en función de la configuración de las ruedas: robots con configuración diferencial y robots con configuración Ackerman.

La configuración diferencial es la más sencilla de implementar siendo además muy barata. Está compuesto por dos ruedas de tracción y una o dos ruedas locas de apoyo. El giro se realiza variando la velocidad de rotación de las dos ruedas motrices, que giran independientemente. El eje de giro se sitúa en medio de estas dos ruedas.

No obstante es un diseño difícil de controlar. Al girar las dos ruedas motrices independientemente es fácil que el robot derrape o que no realice una trayectoria recta. Una variante de esta configuración es utilizar orugas en lugar de ruedas, lo que mejora notablemente la tracción con lo que el problema de los derrapes queda prácticamente solucionado.

En la siguiente imagen se muestra la configuración diferencial. Una ventaja de esta configuración es que permite que el robot gire sobre sí mismo.

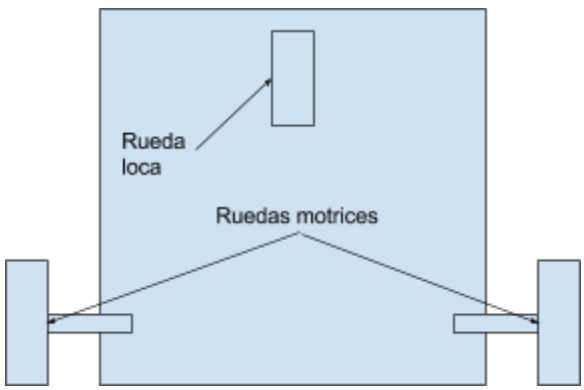


Figura 3: Robot diferencial.

La configuración Ackerman es la que utilizan los automóviles. El robot tiene dos ruedas motrices y dos ruedas orientables, por lo tanto solo se puede modificar la velocidad de las ruedas motrices (ahora conectadas) y la orientación de las dos ruedas. La principal mejora es que se consigue una mejor adherencia con lo que el robot derrapa menos.

Una versión simplificada de esta configuración es el triciclo, aquí en lugar de haber dos ruedas orientables solo hay una. A la hora de calcular las ecuaciones cinemáticas se utilizan las mismas en los dos modelos, pero en el caso de la configuración Akerman se calcula previamente una rueda equivalente a las dos orientables. [8] [9] [10]

En la siguiente imagen se muestra como es el diseño básico de un robot con configuración Akerman.

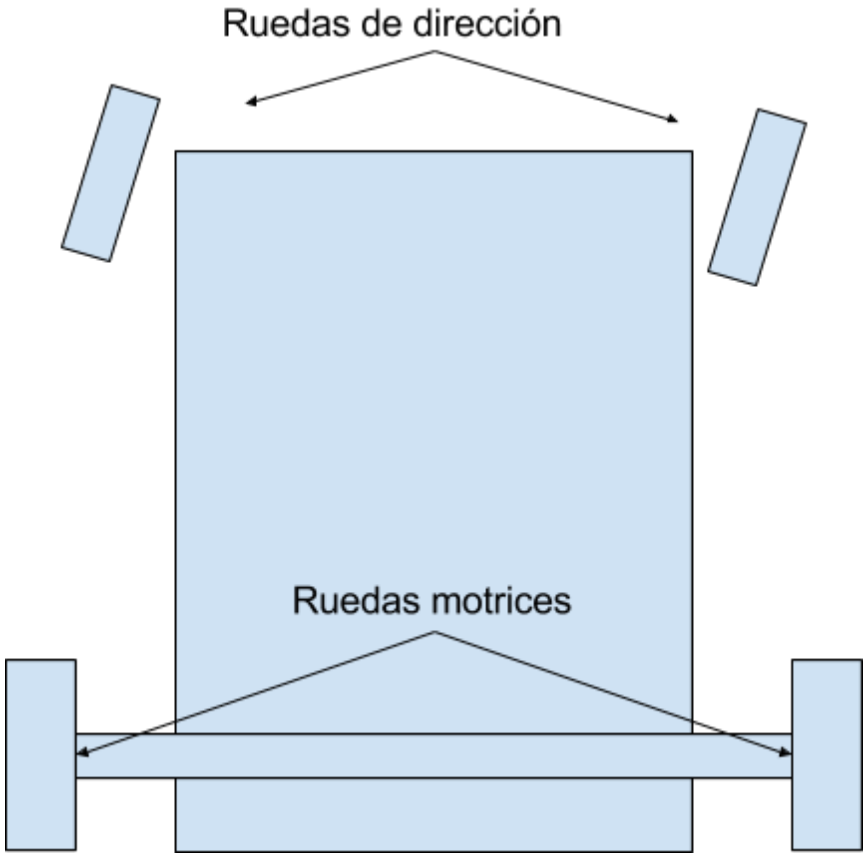


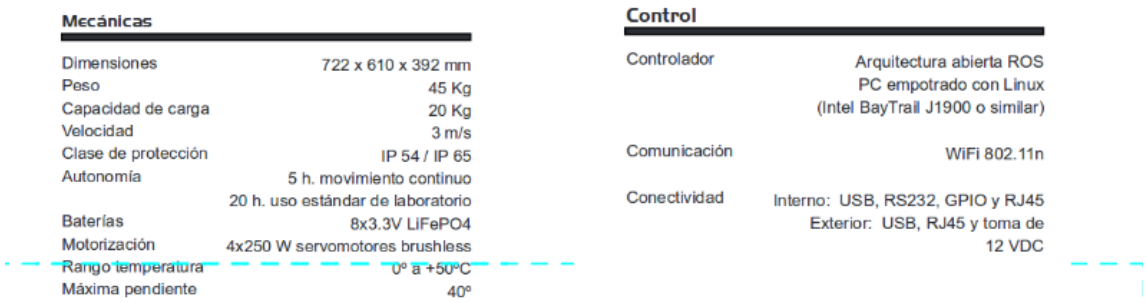
Figura 4: Robot con configuración Akerman

2.2 Summit XL

El robot móvil Summit XL es un robot móvil con 4 ruedas. Tiene un software con una arquitectura modular de código abierto diseñada específicamente para ser programada usando ROS. Está constituido por un cuerpo metálico muy resistente que le permite llevar cargas muy pesadas. Las cuatro ruedas son direccionables y tienen integrado un motor eléctrico. Cada rueda tiene un sistema propio de suspensión modificable alterando, por lo que puede alterar la holgura del robot.

El robot tiene un alto grado de modularidad en el hardware. Eso implica que se pueden añadir sensores fácilmente, además de que las ruedas se pueden cambiar. En el chasis hay un sensor angular de alta precisión además de un encoder en cada rueda con los que se calcula la odometría del robot. Éste puede conducirse por control remoto (con un mando de PS3) o puede tener instalado un programa de conducción autónoma.

Los sensores se pueden personalizar, pero normalmente lleva un Hokuyo laser scanner and a range of RTK-DGPS kits. También tiene conexión interna (USB; RS232 and GPIO) y externa (USB, RJ45 and power supply 12 VDC) para añadir fácilmente componentes personalizados. Las características técnicas del robot son:



Mecánicas	
Dimensiones	722 x 610 x 392 mm
Peso	45 Kg
Capacidad de carga	20 Kg
Velocidad	3 m/s
Clase de protección	IP 54 / IP 65
Autonomía	5 h. movimiento continuo 20 h. uso estándar de laboratorio
Baterías	8x3.3V LiFePO4
Motorización	4x250 W servomotores brushless
Rango temperatura	0º a +50ºC
Máxima pendiente	40º

Control	
Controlador	Arquitectura abierta ROS PC empotrado con Linux (Intel BayTrail J1900 o similar)
Comunicación	WiFi 802.11n
Conectividad	Interno: USB, RS232, GPIO y RJ45 Exterior: USB, RJ45 y toma de 12 VDC

Figura 5: Especificaciones Summit XL [11]

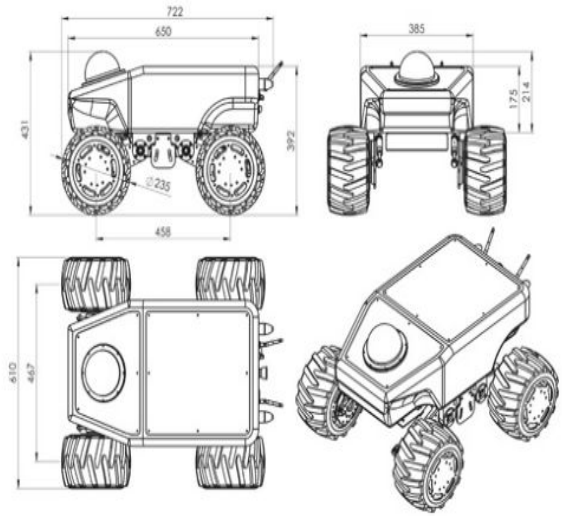


Figura 6: Dimensiones Summit XL [11]

2.3. ¿Qué es ROS?

Al igual que los ordenadores, los robots también tienen un Sistema Operativo. Esto facilita en gran medida la programación del mismo ya que ahorra mucho tiempo en tareas tan básicas como la implementación de los diferentes algoritmos y el acceso a los sensores que pueda tener el robot. La principal ventaja de que los robots tengan un sistema operativo es que permite reutilizar el mismo código para diferentes hardwares.

Al mismo tiempo que los robots tengan sistemas operativos tiene sus desventajas. Las principales son que un sistema operativo dado tiene una serie de aplicaciones implementadas, con lo que existe la posibilidad de que no esté implementada la que se desea utilizar. Otro gran problema es que dos robots con sistemas operativos diferentes pueden tener serios problemas a la hora de comunicarse entre sí.

Existen dos corrientes en las empresas a la hora de diseñar los sistemas operativos de los robots. La primera es crear un sistema operativo propio, la segunda es generar un sistema de código abierto. En el caso del Summit XL el sistema operativo es Ubuntu, pero para facilitar el acceso al hardware del robot, las comunicaciones y el desarrollo de código reutilizable se le ha instalado ROS (Robot Operating System), un middleware. [12]

ROS es un entorno de trabajo libre y abierto el cual agrupa una serie de herramientas, librerías y convenios. Su principal objetivo es facilitar, en la medida de lo posible, la creación de programas que implementan comportamientos complejos en robots. [13]

Pero antes de exponer las principales características de ROS es necesario hacer un breve resumen de su historia. La primera distribución de ROS fue lanzada el 2 de Marzo de 2010. Desde entonces se han ido lanzando nuevas distribuciones siendo la más nueva Kinetic Kame. En el momento de iniciar este trabajo la distribución más nueva era Jade y es en la que se ha realizado casi toda la parte práctica.

El objetivo principal de las distribuciones es el de proporcionar un entorno estable en el que los programadores pueden trabajar con el mismo código base durante años. Las únicas actualizaciones que recibe una distribución tienen como objetivo arreglar los posibles bugs que tenga la misma. Cada robot tiene una serie de necesidades que se intentan cubrir por parte de las distribuciones oficiales. No obstante, al ser software de código abierto se espera que la misma comunidad produzca sus propias distribuciones en el futuro. [14]

Si bien ROS es muy útil también es complejo, todos sus conceptos pueden clasificarse en tres niveles. Éstos son el ROS Filesystem Level, ROS Computation Graph Level y la ROS Community Level. Cada uno de estos niveles es muy amplio por lo que a continuación se explicarán los conceptos más importantes y característicos de ROS.

2.3.1. ROS Filesystem Level.

Este primer nivel hace referencia a todos los archivos que el usuario encontrará en el disco, es decir, todos los directorios y ficheros que se utilizarán. Se explicarán tres elementos:

- Paquetes: principal unidad para organizar el software en ROS. En un paquete se puede encontrar muchos de los elementos de ROS: nodos, librerías, ficheros de configuración o cualquier tipo de fichero que sea útil. Son increíblemente prácticos, pues si un programador busca y encuentra un paquete con las funciones que necesita para su proyecto solo tiene que descargarlo y compilarlo en su workspace. Esta es una de las características que hacen de ROS un sistema increíblemente versátil.
- Paquete Manifests: son ficheros de datos .xml que contienen metadatos sobre los paquetes como sus nombres, versión, descripción, licencia o dependencias.
- Clase de mensaje: definen el tipo de estructura de los mensajes enviados en ROS.

2.3.2. ROS Computation Graph Level.

Este nivel es una red peer-to-peer de procesos de ROS que analizan conjuntamente los datos.

- Nodos: son procesos que realizan la computación, es decir, un nodo es una aplicación que ejecuta ROS. La idea original es que cada nodo contenga las instrucciones para realizar tareas bastante concretas. Un buen uso de esto se hace creando un nodo para el control de las ruedas, otro para el cálculo de la posición y otro para la comunicación con una máquina externa. Como ROS está específicamente diseñado para ser modular, si se programa separando las funciones en nodos (tal y como se ha ejemplificado antes) se obtiene la ventaja de que estos nodos son fácilmente modificables e intercambiables, por lo que se puede modificar una parte del programa sin afectar en modo alguno al resto.
- Máster: proporciona nombres y un buscador al resto del Computation Graph. Sin el Master, los nodos serían incapaces de encontrarse o intercambiar mensajes.
- Mensajes: los nodos se comunican entre sí mediante la transferencia de mensajes siendo éstos una simple estructura de datos. Un mensaje puede contener estructuras anidadas (son muy parecidas a los structs de C) de muchos tipos diferentes.
- Topics: son el sistema de transporte de los mensajes. Con el nombre de un topic se identifica al mensaje que se está enviando. Un nodo puede publicar información al sistema mediante un topic. Al mismo tiempo otro nodo puede suscribe a ese topic accediendo así al mensaje que contiene el topic. Un mismo nodo puede suscribirse y publicar al mismo tiempo tantos topics como sea necesario. Esto supone una característica importante de ROS, pues se diferencia muy claramente la elaboración de información del consumo de la misma.
- Servicios: son una forma alternativa de comunicación entre nodos. A diferencia de los topic que son unidireccionales los servicios permiten una comunicación bidireccional entre dos nodos.

2.3.3. ROS Community Level.

Una gran ventaja de que ROS esté específicamente diseñado para ser modular es que los diferentes paquetes y nodos se pueden intercambiar. Si a eso se le añade que ROS es un sistema de código abierto el resultado ha sido la creación de una comunidad donde se pueden descargar gratuitamente paquetes desde diversas fuentes.

Una de estas fuentes es la ROS Wiki, donde hay paquetes para descargar y tutoriales para aprender a manejar los conceptos del sistema. Otra fuente increíblemente útil es la ROS Answers, foro especializado donde se pueden formular preguntas concretas de ROS para que la comunidad las responda.

Para instalar ROS y aprender a usarlo hemos seguido los tutoriales de esta web: http://wiki.ros.org/ROS/Tutorials. Aquí se explican desde los principios más básicos del entorno de manera clara y concisa a los conceptos más avanzados.

2.3.4. Comunicación usando topics.

Tal y como se ha explicado en ROS un nodo puede tener el rol de publicador, el de suscriptor o tener los dos roles al mismo tiempo. Suponiendo un nodo publicador y otro nodo suscriptor el funcionamiento es muy simple: el nodo publicador envía el mensaje a través del topic con una frecuencia dada. Es precisamente a este topic al que el suscriptor se suscribe para obtener la información del mismo.

Para que todo esto funcione correctamente es necesario que el mensaje tenga una estructura estandarizada. Esto se hace definiendo dicha estructura en un archivo .msg que deberá estar en todos los paquetes que utilicen dicho estructura para enviar mensajes a través de topics. El siguiente esquema muestra cómo se comunicarán 3 nodos siendo uno un publicador, otro un publicador y un suscriptor y un tercer nodo que solo es un suscriptor.

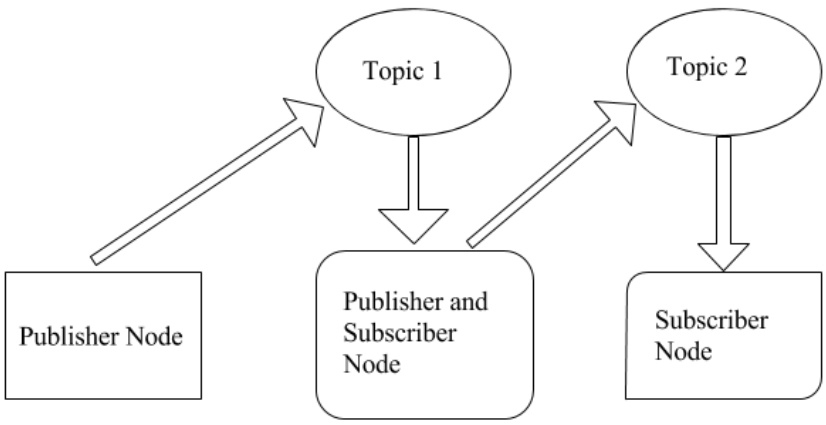


Figura 7: Comunicación entre nodos usando topics.

2.3.5. Comunicación usando servicios.

La principal diferencia respecto a los topics es que permite una reciprocidad entre dos nodos. Esto implica que ya no hay un nodo que publica constantemente información sin importarle a quién le llega, ahora el nodo que requiere la información, el nodo cliente, envía una petición al nodo servicio que responderá con el servicio solicitado.

Des esta manera se mejora la comunicación de los topics al crearse una comunicación bidireccional. La siguiente imagen refleja la idea:

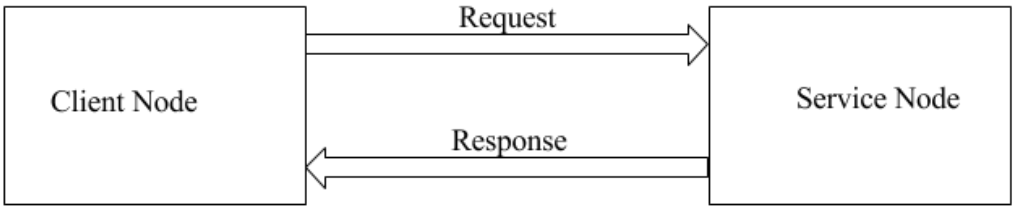


Figura 8: Comunicación entre nodos utilizando los Services.

2.3.6. Trabajar con ROS

ROS no tiene interfaz gráfica propia por lo que es necesario trabajar desde un terminal. Este cambio requiere de un tiempo. Para moverse por los ficheros utilizados, lanzar nodos o publicar topics hay una serie de comandos que es necesario aprender a utilizar. Se procede a exponer los principales comandos de ROS:

- Roscore: inicializa el núcleo de ROS. Es el primer comando que se ejecuta para trabajar con ROS.
- Catkin_make: con este comando se compila el código creado. Es necesario modificar el archivo CMakeLists.txt y ejecutar el comando desde la raíz del workspace.
- Roscd: permite cambiar el directorio actual de trabajo a un paquete, stack o a una localización común sin necesidad de conocer la ruta del mimo, solo con su nombre.
- Rosdep: instala todas las dependencias necesarias de un paquete.
- Rosed: permite editar directamente un archivo dentro paquete, de nuevo, sin necesidad de conocer todo la ruta del mismo.

- Rosrun: permite lanzar un ejecutable, es decir, un nodo, solo con su nombre y el paquete al que pertenece.
- Roslaunch: permite lanzar una serie de nodos y modificar el nombre de los mismos además de algunas de sus variables (topics a los que se suscribe o pública entre otras). El roslaunch utiliza un fichero tipo .xml del que solo es necesario conocer el nombre y el paquete al que pertenece.
- Rosmake: permite construir todas las dependencias requeridas en el orden correcto. Es un asistente que ayuda a construir los paquetes. [15]
- Rosnode: permite obtener información de los nodos que se están ejecutando. Una de las funcionalidades que ofrece más útiles es rosnode list que muestra una lista con todos los nodos ejecutándose. Otra función también muy interesante es rosnode info nombre_nodo que permite ver que topics publica y a cuales se suscribe entre otra información del nodo.
- Rostopic: tiene las mismas funciones que el rosnode pero respecto a los topics. De nuevo las dos más interesantes son los equivalentes rostopic list y el rostopic info nombre_topic que muestra el tipo de mensaje que contiene.
- Rqt_graph: saca por pantalla un diagrama con las interacciones de todos los nodos y topics permitiendo así una fácil visualización y comprensión de las comunicaciones de manera clara y fácil. [16]

2.3.7. Integración de librerías.

Una de las características que hacen de ROS una gran herramienta es el alto grado de integración con las librerías más utilizadas (Gazebo, OpenCV) por lo que la tarea de programar con ROS se hace mucho más fácil.

Además, es muy importante resaltar que la alta modularidad es especialmente útil para trabajos de investigación. ROS se implementará en el ordenador que se utilizará durante el desarrollo del proyecto. Gracias a estas características un mismo programa creado en ROS es fácilmente implementable en diferentes dispositivos.

2.4. Localización de robots.

Una vez quedan perfectamente explicados los principios básicos tanto del hardware de los robots como de su software se procede a profundizar en un aspecto muy importante propio de los robots móviles.

La principal característica, como ya se ha visto anteriormente, en los robots móviles es su capacidad para desplazarse. Esto implica cierto grado de complejidad, pues ahora es necesario disponer de un sistema que localice al robot en un mapa. Para ello existen varios sistemas que utilizan diferentes tecnologías.

Muchas aplicaciones utilizan robots móviles, por ejemplo aspiradoras automatizadas que recorren la casa, robots que transportan mercancía dentro de una fábrica o coches de conducción sin piloto. En todos estos casos es imperativo que el robot tenga un mecanismo que le indique en todo momento su posición con la mayor precisión posible.

En este apartado del trabajo se procederá a estudiar brevemente los principios básicos de funcionamiento de los sistemas de posicionamiento existentes además de sus principales ventajas e inconvenientes. Esto último se hará con la idea de identificar las debilidades de cada sistema para comprobar si hay margen de mejora para realizar un sistema de posicionamiento complementario a los actuales.

2.4.1. GPS.

El posicionamiento de un robot mediante el uso de satélites, es decir, usando la tecnología GPS, suele ser un sistema fiable y muy preciso. El GPS puede llegar a ser muy interesante en muchas aplicaciones y es relativamente fácil de implementar.

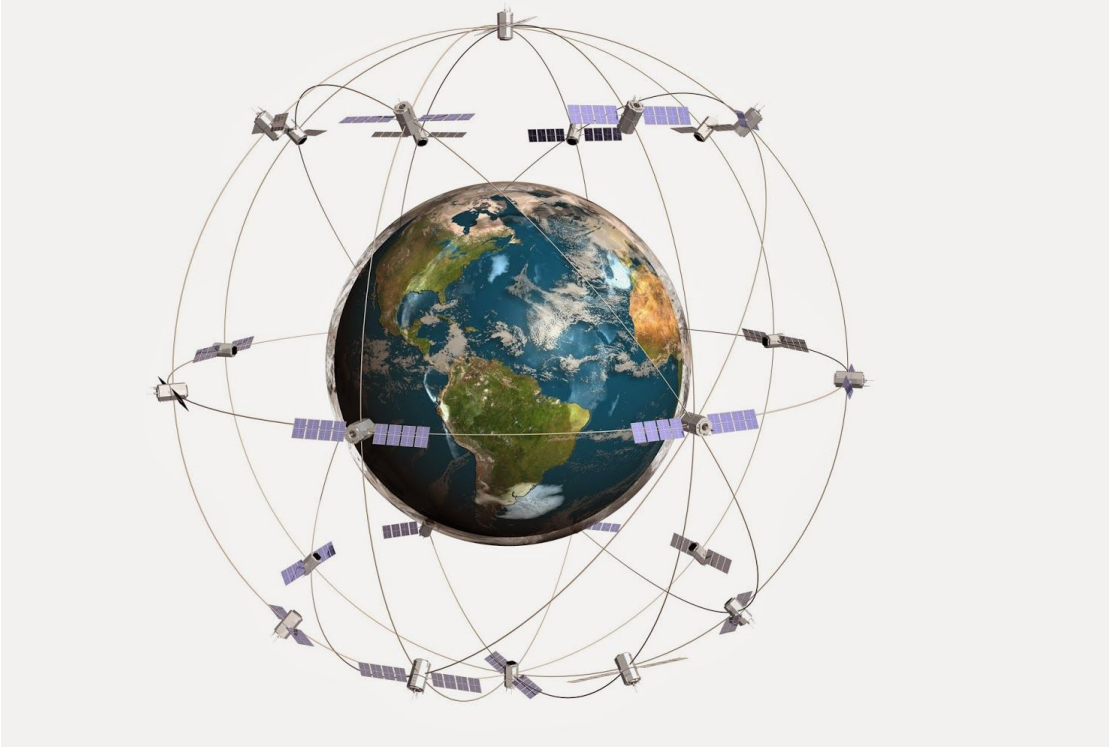


Figura 9: Órbitas satélites.

Figura 8: Representación de una red de satélites. [17]

El GPS trabaja, del mismo modo que lo hacían los antiguos sistemas electrónicos de posicionamiento, con funciones matemáticas de triangulación. El receptor se conecta con un mínimo de 3 satélites para medir la distancia que los separa. Para hacer esto se mide el tiempo que tarda una señal de radiofrecuencia en recorrer esa distancia, la velocidad de la onda de radiofrecuencia en el vacío es conocida y con cálculos matemáticos se calcula la distancia. [18]

En un entorno ideal el cálculo de la posición por este sistema sería perfecto, dicho esto, hay muchos factores que pueden afectar a la precisión del GPS. Para calcular la distancia se utiliza como velocidad de la onda de radiofrecuencia la que tiene en el vacío. El problema es que la onda también viaja por la atmósfera terrestre, donde la composición de la misma provocan que la velocidad de la onda sea ligeramente menor, provocando un error en el cálculo de la distancia. Este error no es constante pues cambios en la composición de la atmósfera tendrán un efecto igualmente variable en el error cometido, pudiendo ser de varios metros. [19] [20]

Un segundo problema especialmente molesto en entornos urbanos es la reflexión de la señal en grandes estructuras, normalmente edificios. El principio de esta fuente de error es muy

simple: al estar el receptor entre edificios es muy fácil que le llegue tanto la señal directa desde el satélite como una serie de señales que han rebotado en los edificios cercanos. Este error puede llegar a tener una magnitud de entre 2 y 4 metros, por lo que no es despreciable. [21]

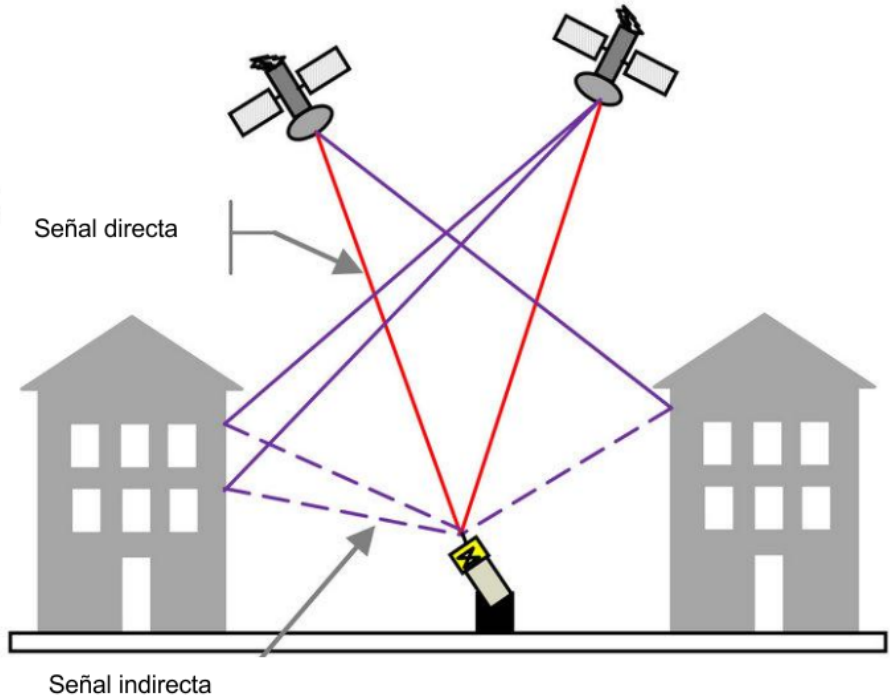


Figura 10: Diferentes recorridos de la onda. [22]

Una manera muy efectiva de eliminar esta fuente de error es utilizar un receptor con tecnología más avanzada. Para eliminar el error provocado por la multi-trayectoria se implementa un sistema denominado Choke ring. Este consiste en rodear al receptor en una serie de cilindros concéntricos de material conductor. Gracias a este sistema se reduce el error a milímetros, pero por sus materiales de construcción y volumen es poco recomendado para robots pequeños. [23]

Otro problema que puede afectar muy seriamente a la precisión del posicionamiento del receptor recibe el nombre de Dilución Geométrica de la Precisión o DGP. Este es inherente al principio básico matemático del sistema. El funcionamiento básico del GPS es que sobre cada satélite se dibuja una esfera virtual, al obtener un mínimo de tres esferas centradas en tres satélites se obtiene un único punto común con el que se posiciona al receptor.

El problema surge cuando estos satélites están demasiado cerca, en este caso el prisma que forman es pequeño. Si este es el caso las esferas virtuales que se dibujan alrededor de cada

satélite tendrán una zona de coincidencia muy grande. Esto provoca cierta confusión en el sistema que da como resultado un error en la medida de la posición. En el siguiente dibujo se ilustra el problema:

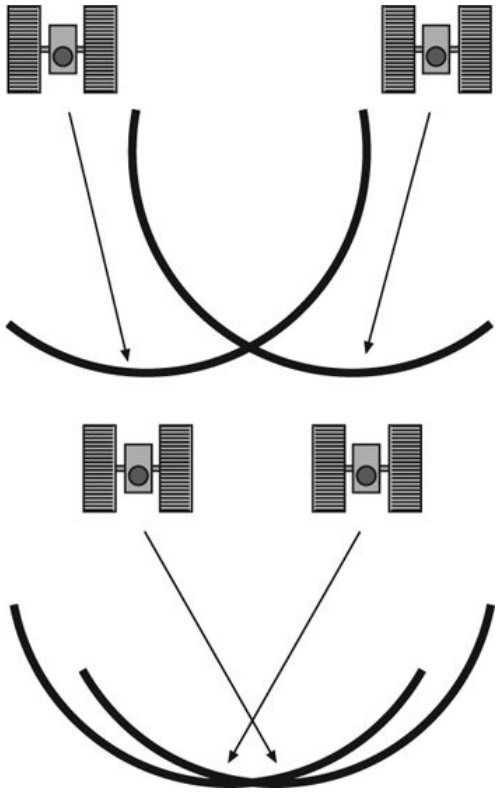


Figura 11: Arriba: DGP correcto, el error es mínimo. Abajo: DGP deficiente, el error es grande. [24]

La solución a este problema es implementar un GPS que escoja los satélites para evitar un DGP deficiente. No obstante esta fuente de error depende de la posición de los satélites, por lo que su eliminación es bastante compleja. [25]

Finalmente otra fuente de error es la diferencia entre el reloj del satélite y el reloj del receptor. El satélite tiene un reloj atómico de cesio cuya precisión es muy superior a la del reloj del receptor. Si a esto se le añade la dilatación temporal debido a la teoría de la relatividad surge un pequeño desfase que provoca un pequeño error en la medición.

A todo esto, que más o menos se puede solucionar empleando tecnología más avanzada, con su inevitable incremento del presupuesto, debe añadirse una consecuencia del funcionamiento básico del sistema se comete siempre un error de 3 metros (mejorable a 0.3 metros si se usa tecnología militar). [26]

Queda claro que el GPS es un excelente sistema de posicionamiento para coches en carretera, barcos en el océano o aviones. Pero si lo que se desea es localizar al robot reduciendo el error hasta las unidades de milímetros es un sistema poco ideal. Además hay que tener en cuenta que bajo ciertas circunstancias, como dentro de edificios túneles o en cuevas donde la señal del mismo es muy mala o directamente se pierde. En estos entornos el error de posicionamiento cometido es enorme, su uso es muy poco recomendable, por lo que se requiere del uso de sistemas alternativos.

2.4.2. Dead Reckoning.

Una alternativa a los sistemas de geolocalización es usar los sistemas Dead Reckoning. El principal problema del GPS es que depende de elementos externos, los satélites, y de que ninguna de las fuentes de error previamente explicadas provoque una desviación sobre la posición real significativa. Por eso era necesario tener un sistema que no dependiera de tales aleatoriedades. Por ello se inventó un sistema que no se precisa de ningún satélite para calcular la posición, con lo que pueden ser implementados en cualquier ambiente o territorio.

Su funcionamiento es muy simple: toda la información que utiliza el sistema para obtener la posición del robot proviene de fuentes de información internas, es decir, de los diferentes sensores que tiene el robot. Es importante remarcar que este sistema no calcula la posición del robot, lo que hace es estimar la misma tomando como información los datos recibidos desde los sensores. Esta es una diferencia muy importante respecto a los sistemas de geolocalización por satélite.

Al utilizar estos sistemas se dispone de dos alternativas a la hora de calcular la posición: se puede partir de una posición aleatoria desconocida y calcular la posición del robot a medida que avanza respecto a ese punto, es decir, una posición relativa. La otra opción es conocer de antemano el punto inicial del robot para así conocer siempre la posición absoluta del robot en lugar de una posición relativa. La elección de un sistema u otro depende de la finalidad del programa. Un aspirador automatizado doméstico utiliza coordenadas locales mientras que un coche con conducción autónoma requiere de coordenadas globales.

Existen dos grandes alternativas a usar como fuente de información en los sistemas Dead Reckoning. La primera es usar una serie de sensores ópticos, estos miden la velocidad de las ruedas y el ángulo de giro. La segunda opción es utilizar un sistema de navegación inercial. Este utiliza sofisticados acelerómetros para medir el movimiento del robot.

No obstante cada uno de estos sistemas tiene sus inconvenientes. El sistema más común es usar unos encoders para obtener la velocidad y ángulo de giro e integrando estos calcular el desplazamiento del robot. El problema reside en que es muy fácil que los elementos mecánicos de los que depende este sistema tengan asociados errores. Es relativamente fácil que el giro o la velocidad real del robot sea diferente al medido por los sensores. También es muy probable que el deslizamiento entre las ruedas y el suelo provoquen falsas medidas. Todo esto provoca un error y, si se tiene en cuenta que es un sistema integrativo, el error se acumula en cada cálculo.

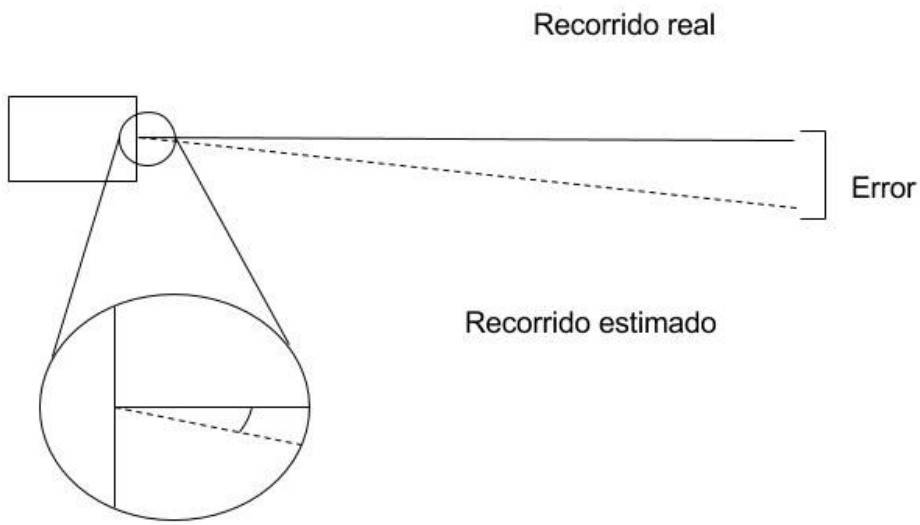


Figura 12: Error Dead Reckoning

Demostración del error (Figura 12) consecuencia de una medición de ángulo errónea. Si el error en una trayectoria recta fuera de 3 grados se acumularía un error de 5.24 centímetros por cada metro avanzado.

Si bien tienen este problema inherente a su naturaleza son muy fáciles de implementar. Los sensores que utiliza son relativamente baratos, consumen poca energía y poca potencia de cómputo.

El segundo segundo sistema que se puede implementar es el Sistema de Navegación Inercial. Este consiste en el uso de acelerómetros diminutos para medir el movimiento del robot. La principal ventaja frente al uso de sensores ópticos es que el error que se comete con este sistema es considerablemente menor, del orden del 0.1%.

El gran dilema con el Sistema de Navegación Inercial es que es extraordinariamente caro. Esto es así ya que los acelerómetros utilizan electrónica de gran precisión y acelerómetros para estabilizar el sistema de toma de datos. Además el sistema funciona correctamente siempre que las aceleraciones sean suaves, por lo que no se puede implementar en robots pequeños que tendrán giros y cambios de velocidad bruscos. Por lo tanto para pequeños robots móviles no es práctico. [27]

En resumen, el sistema Dead Reckoning tiene dos grandes inconvenientes en función del sistema implementado: comete un error que además se acumula con el tiempo o es muy caro de implementar

2.4.3. Resultado del estudio.

El resultado del estudio es claro: tanto los sistemas basados en el GPS como los sistemas basados en el Dead Reckoning funcionan perfectamente bajo condiciones ideales, pero como estas difícilmente se suelen dar lo normal en ambos sistemas es tener un error en el posicionamiento.

Así pues la idea de crear un sistema complementario a estos podría mejorar las características de los mismos dando como resultado un sistema de posicionamiento más preciso.

2.5. Visión artificial.

A medida que los robots son más complejos las funciones que son capaces de realizar son, al mismo tiempo, más completas y sofisticadas. Cada día los ingenieros e investigadores se acercan más a crear una inteligencia artificial que sea capaz de razonar, de analizar el entorno y, en base al mismo, tomar decisiones.

Pero para que esto sea posible es necesario que el robot tenga fuentes de información del mundo que le rodea mucho más complejas que un sensor de proximidad o una placa de presión. Es necesario un sistema que recoja mucha más información del entorno. La visión artificial es un sistema que dota al robot de una capacidad de toma de datos extraordinaria permitiendo dar un enorme paso en la investigación de la inteligencia artificial.

La visión artificial es uno de los campos de la inteligencia artificial y su principal objetivo es construir un sistema que imite la visión humana. Esto no solo implica la toma de imágenes en tiempo real, también es necesario tener un sistema que analice esa información y la procese para finalmente almacenarla, todo esto se hace con un modelo matemático que se aplica a imágenes digitales.

2.5.1. Características.

Las características que hacen de la visión artificial una alternativa tan a tener en cuenta son:

1. A diferencia del ojo humano, que solo puede captar un pequeño rango de frecuencias conocido como rango de luz visible, la visión artificial es capaz de captar todo el espectro electromagnético.

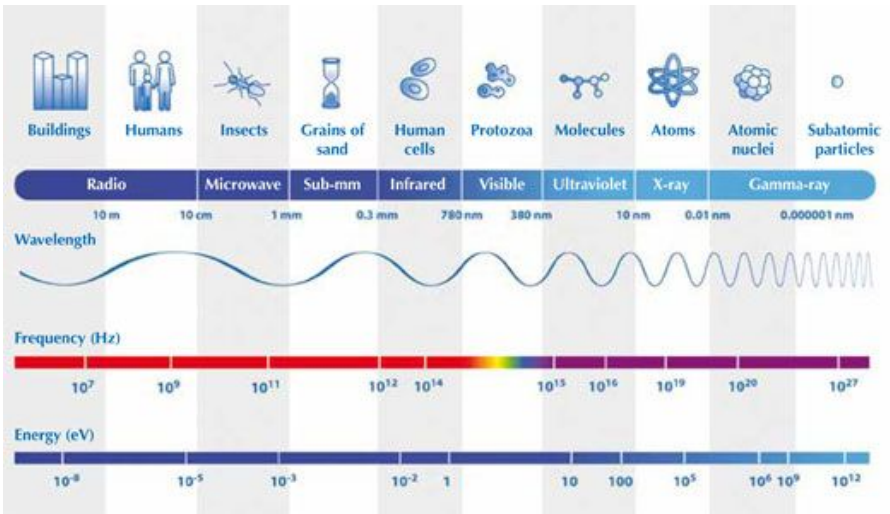


Figura 13: Espectro Electromagnético. [28]

2. La velocidad de respuesta de la visión artificial ya es mucho más rápida que la del ojo humano. Además esta velocidad aumenta diariamente a medida que avanza la investigación.
3. La calidad de un sistema de visión artificial no varía con el tiempo, permanece constante durante toda su vida útil. En humanos la visión se ve afectada por el cansancio y las emociones.
4. La visión artificial es muy precisa a la hora de realizar mediciones.

5. Es posible aplicar la visión artificial en entornos con condiciones muy adversas para una persona ya sea por temperaturas extremas, por una composición de la atmósfera venenosa o por la presencia de radiación. [29]

2.5.2. Aplicaciones de la visión artificial.

En la actualidad la visión artificial tiene muchos usos y cada día está más implementada en los diferentes sectores de la industria. Algunos de los usos de esta tecnología son:

- Metrología óptica: en las cadenas de montaje es necesario realizar un control de calidad sobre las piezas. Con la visión artificial es posible verificar si las piezas tienen las dimensiones requeridas. Este sistema tiene dos grandes ventajas: permite la verificación del 100% de las piezas y es muy preciso, llegando al orden de magnitud de centésimas de milímetro.
- Seguridad biométrica: el uso de la visión artificial como medida de seguridad es muy fiable por la alta precisión del sistema en sí mismo. El reconocimiento facial, ocular o dactilar ofrece una seguridad muy alta.
- Aplicación en la medicina: algunas de las máquinas más modernas que se utilizan en el campo de la salud se basan en la visión artificial. Gracias a las resonancias magnéticas y a las tomografías se es capaz de detectar tumores en estadios iniciales. [30]
- Decodificación: identificación de códigos 1D, símbolos 2D y otros modelos de marcadores. [31]

Se observa pues el gran peso que tiene la visión artificial en la sociedad actual y todo el potencial que tiene, pues permite que un sistema informatizado obtenga información del mundo que le rodea utilizando una imagen digital actualizada en tiempo real.

2.5.3. Elección de una librería.

Vistas las características de la visión artificial es necesario escoger una librería. Afortunadamente en la actualidad desarrollar una aplicación basada en la visión artificial es mucho más fácil que antaño ya no siendo necesario tener grandes conocimientos en programación. Hoy en día la mayoría de las librerías tienen una gran cantidad de

herramientas de software con muchísimas herramientas disponibles para el programador. De esta manera crear un aplicación es mucho más sencillo.

La librería necesaria para este proyecto debe cumplir los siguientes requisitos:

- Libre: sería muy útil encontrar una librería libre y gratuita para poder integrarla fácilmente en el proyecto y si fuera necesario, modificarla.
- Con un gran soporte: también se valorará mucho que esté muy extendida para poder encontrar manuales o tutoriales con facilidad.
- Debe estar disponible en Ubuntu.
- Debe poder ser programada en C y/o C++.
- Debe ser eficiente en el análisis y procesamiento de imágenes en tiempo real.

Debido a lo comprimido en el tiempo que está este proyecto no es inteligente dedicar un tiempo que no se tiene a realizar un análisis de las diferentes librerías existentes. En lugar de eso se le ha preguntado a la tutora del proyecto si conocía una librería con las características deseadas. La solución aportada es la librería OpenCV. Cumple con creces todas las características deseadas para el proyecto. Algunas de las características de OpenCV son:

- Reconocimiento de formas: de objetos y facial.
- Compatibilidad con la realidad aumentada.
- Compatibilidad con la robótica, proporciona una visión artificial.
- Calibración de cámaras.
- Interpretación de gestos: permite cierta interacción con el usuario. [32]

2.6. Realidad aumentada.

Si bien la visión artificial es increíblemente útil es posible llevar la tecnología un paso más allá. A diferencia de la visión artificial, la realidad aumentada no solo se apoya en la imagen captada de la realidad física creando una realidad 100% virtual, lo que hace es combinar la realidad física con una realidad virtual creando así una realidad mixta.

Una manera de visualizar este concepto es utilizar la escala de continuidad de la realidad “Milgram-Virtuality Continuum”:

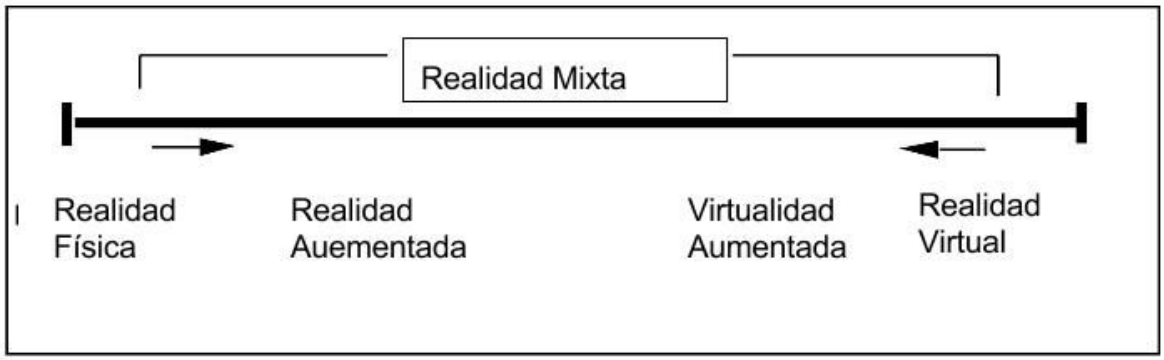


Figura 14: Realidad mixta.

2.6.1. Componentes básicos.

Pero antes de profundizar más en la tecnología de la realidad aumentada es necesario exponer los componentes básicos de cualquier sistema que la utilice:

- Un monitor: es imprescindible tener una pantalla donde poder visualizar la información, es decir, la realidad mixta creada. En la actualidad se puede utilizar un teléfono móvil o incluso unas gafas.
- Cámara: elemento básico en el que empieza todo. Permite captar la realidad física y convertirla en una imagen digital.
- Marcador: activa la realidad aumentada, puede ser una imagen, un punto geográfico o un objeto
- Información virtual: una vez se ha obtenido la imagen digital de la realidad física es necesario obtener el resto de la información, la realidad digital.
- Software: una vez se tiene la realidad física y la realidad aumentada es necesario que un programa fusione las dos partes y las muestre en la pantalla. [33]

2.6.2. Aplicaciones.

Como se ha visto la realidad aumentada conlleva un gasto en material y un trabajo relativamente complicada de desarrollo de software. Es pues necesario exponer un par de ejemplos para justificar su implementación en el proyecto:

1. Catálogo de ikea: desde el 2014 es posible descargarse una aplicación específica del catálogo de Ikea que se basa en la realidad aumentada. Esta aplicación permite

visionar cómo quedaría la casa en función de los muebles que se seleccionen, facilitando mucho la elección de los muebles a comprar.

2. Accesorios Mercedes-Benz: comprar un coche supone un gasto muy superior al de un sofá y de este hecho son muy conscientes en la empresa alemana. Por este motivo crearon una aplicación que permitía ver en tiempo real como quedaría una tapicería o piezas en uno de sus coches. [34]

Las aplicaciones de la realidad aumentada, como se ha visto, pueden llegar a tener un gran impacto en la industria. Si se explota todo lo que puede ofrecer el cambio que sufrirá ya nos solo la industria pero la sociedad en general será muy significativo.

2.6.3. Niveles de la Realidad Aumentada.

Dicho eso existen diferentes niveles de realidad aumentada. Estos niveles se definen como la complejidad que presentan las aplicaciones que basan su funcionamiento en la realidad aumentada en función de las tecnologías que implementan. Así pues cuando mayor sea el nivel de realidad aumentada de una aplicación sus funciones serán más complejas. El siguiente sistema de clasificación está propuesto por Lens-Fitzgerald: [35]

- Nivel 0: hiperenlaces en el mundo físico. La aplicación detecta y escanea un código de barras o un marcador 2D. El código o marcador sirve de hipervínculo para acceder a una página web.

Figura 15: Ejemplo de código QR

- Nivel 1: la realidad aumentada se basa en marcadores. En este nivel el activador de la realidad aumentada es un marcador (imagen sin color, de forma cuadrangular y que contiene un dibujo esquemático) sobre el cual se dibuja un objeto 3D virtual.

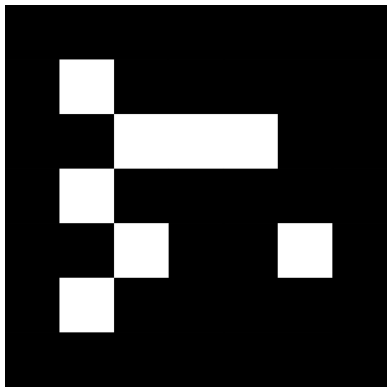


Figura 16: Ejemplo de marcador.

- Nivel 2: realidad aumentada sin marcadores. Ya no se precisa el uso de marcadores, el activador de la realidad aumentada es un objeto, imágenes o coordenadas GPS.

Figura 17: Ejemplo de realidad aumentada de nivel 2. [36]

- Nivel 3: visión aumentada. Es la tecnología más avanzada. Se alcanza el nivel 3 cuando se integra la realidad aumentada en las gafas (como las Google Glass) o en lentes de contacto. Así se consigue una experiencia mucho más inmersiva y personalizada. [37]

Figura 18: Gafas de realidad aumentada (nivel 3). [38]

2.6.5. Selección librería de Realidad Aumentada.

Seleccionada ya una librería de realidad virtual es necesario seleccionar una librería de realidad aumentada, ya que el objetivo último de este proyecto es crear una sistema de posicionamiento basado en marcadores. Para hacer esto primero es necesario establecer las características de la librería a implementar:

- Compatibilidad con OpenCV: puesto que es la librería de realidad virtual a implementar, es imprescindible que la librería de realidad aumentada sea compatible con la primera.
- Reconocimiento de marcadores 2D: es necesario que los reconozca para poder cumplir una de las necesidades más básicas de este proyecto.
- Disponible en Ubuntu y ROS.
- Librería de código abierto, por los mismos motivos esgrimidos en la elección de una librería de realidad virtual.
- Que sea fácil de implementar.

Se ha realizado una búsqueda en la web de ayuda de ROS y se ha encontrado una librería perfecta para el proyecto, aruco_ros. Se trata de una librería basa en OpenCV que se especializa en el reconocimiento de marcadores. Además tiene una licencia BSD, está especialmente pensada para su implementación en proyectos académicos y tiene una serie de ejemplos para que la adaptación a su funcionamiento sea lo más rápida posible. [39]

2.7. Comunicación ordenador-Summit XL.

Una vez construidos todos los paquetes y launchers y tras probarlos debidamente en el ordenador es necesario transferir todo el trabajo del ordenador al Summit XL. Esto parece una tarea complicada pero realmente no lo es tanto. El SUMMIT XL tiene integrado un ordenador con Ubuntu empotrado como sistema operativo, por lo que hay que acceder a este. La solución más fácil sería acceder al mismo directamente, pero como esto no se puede hacer es necesario buscar una alternativa.

Transferir los archivos al ordenador del Summit XL es solo el primer paso. Una vez conseguido esto es necesario poder acceder al ordenador del mismo para poder realizar las modificaciones de código necesarias, compilar el código y ejecutar los nodos construidos,

Para ello utilizaremos un programa llamado SSH. Éste implemente un protocolo (serie de normas que permiten que dos sistemas independientes se comuniquen entre sí para intercambiar información, en este caso el nombre del protocolo coincide con el nombre del programa que lo implementa) cuya idea fundamental es la de acceder a máquinas remotas (en este caso el Summit XL) a través de una conexión de red.

Con esto podemos acceder al terminal del robot y ejecutar cualquier comando. Desafortunadamente no permite el intercambio de información entre la máquina remota y el ordenador.

Para transferir archivos al Summit XL la primera idea, por simplicidad, es usar el protocolo FTP.

Entendido esto, se ha procedido a utilizar una herramienta, el SCP, que permite intercambiar datos por SSH. El SCP es un protocolo que permite al usuario transferir datos a una máquina (en este caso el Summit XL) de una manera segura (pues cifra la información). Si bien es cierto que puede ser un poco engorroso comparado con el protocolo FTP, tiene la inconmensurable ventaja de poder transferir datos a cualquier directorio de la máquina, no solo a los específicamente diseñados para tal fin (como es en el caso del protocolo TFP).

El comando utilizado para llevar a cabo tal transferencia de información es el scp. Es bastante fácil de implementar. También es importante destacar la versatilidad del comando, pues permite la transferencia en los dos sentidos (cargar datos en la máquina remota o descargarlos desde ésta al ordenador). [40]

Capítulo 3: Desarrollo práctico.

3.1. Introducción y planteamiento.

Establecidas pues las bases teóricas imprescindibles para entender el trabajo realizado se procede a exponer la parte práctica del mismo.

Puesto que el objetivo del trabajo es, como ya se ha explicado, crear un sistema de posicionamiento basado en el reconocimiento de marcadores que trabaje como apoyo a los sistemas de posicionamiento Dead Reckoning y GPS será necesario seleccionar, instalar y configurar un paquete de realidad aumentada.

En esta tecnología, la realidad aumentada, hay diferentes niveles de complejidad. Al aumentar ésta, más avanzadas serán las funciones que se podrá implementar. En este caso interesa utilizar el nivel 1 que permite el reconocimiento de patrones 2D, en este proyecto, los marcadores. La principal característica de esto es que permite no solo identificar al marcador, también calcula la posición relativa de éste con respecto a la cámara, lo que va a ser increíblemente útil para calcular la posición del robot.

Todo lo descrito hasta ahora se implementará utilizando la herramienta ROS. En el apartado dedicado a ROS en el Capítulo 2: Desarrollo teórico queda muy claro que su modularidad y compatibilidad con una gran cantidad de robots, además de su comunidad de desarrolladores dando soporte, hacen del mismo una herramienta increíblemente versátil.

Así pues se procede, antes de empezar con el desarrollo práctico, a establecer una serie de normas orientativas para facilitar la ejecución del mismo y que el proyecto tenga una estructura más organizada y, por tanto, más reproducible en el futuro por compañeros:

- Descargar e instalar Ubuntu.
- Leer los manuales de ROS.
- Instalar ROS en el ordenador para poder seguir los tutoriales de la web oficial para entender bien todos los conceptos, características y funciones de ROS.

- Asimilados los conceptos básicos de ROS, investigar cual es la mejor herramienta para implementar el reconocimiento de marcadores.
- Descargar los packages necesarios.
- Pensar la estructura de la aplicación a desarrollar para que la misma cumpla con los objetivos del proyecto. Se desarrollará dos aplicaciones para ser implementadas en el Summit XL:
    - Desarrollar una primera aplicación que a partir de la posición relativa del marcador respecto al robot calcule la posición relativa del robot respecto al marcador.
    - Implementar una segunda aplicación que añada el cálculo de la posición absoluta del robot dentro de un mapa a las funciones de la aplicación anterior.

Se crearán dos aplicaciones para tener una primera versión más simple que servirá para comprobar que el sistema matemático utilizado funciona correctamente sin tener que establecer la posición de los marcadores, tarea que requerirá de una gran cantidad de tiempo. A partir de este apartado se explicará con detalle todo el trabajo elaborado, explicando en cada paso dado, los procedimientos desarrollados con las convenientes justificaciones sobre las decisiones en cada caso particular.

3.2. Material y equipo.

Todo el equipo requerido durante el proyecto será descrito a continuación, pues durante el desarrollo del proyecto se ha requerido el uso de diferente material y equipos.

Durante el desarrollo del proyecto se han utilizado dos ordenadores. El primero de ellos es el portátil personal, con conexión a internet sobre el que se ha hecho gran parte del proyecto: realizar los tutoriales necesarios para entender ROS, instalar y ejecutar las librerías necesarias y programar los nodos necesarios. Las características técnicas del ordenador son las de la siguiente tabla:

Tabla 1: Ordenador portátil en el que se ha realizado la mayor parte del proyecto.

Equipo portátil Lenovo	
Procesador	AMD A4-5000 APU with Radeon(TM) HD Graphics × 4
Sistema Operativo	Ubuntu 14.04 LTS (64 bits)

Memoria RAM	15.1GiB
Tarjeta gráfica	Gallium 0.4 on AMD KABINI
Disco duro	180.7GB (partición en un disco duro de 1TB)

Una vez se ha desarrollado la aplicación de posicionamiento y se ha testeado correctamente en el portátil personal es necesario probar que funciona en el Summit XL. Para conectarse al Summit XL se ha utilizado un portátil diferente. Las características de este nuevo portátil son las siguientes:

Tabla 2: Especificaciones segundo portátil

Equipo portátil Asus	
Procesador	Intel® Core™ i5-3230M CPU @ 2.60GHz × 4
Sistema Operativo	Ubuntu 14.04 LTS (64 bits)
Memoria RAM	3.7GiB
Tarjeta gráfica	Intel® Ivybridge Mobile
Disco duro	487.7 GB

El portátil en el que se ha realizado gran parte de este proyecto tiene instalada la versión del sistema operativo de Microsoft Windows 8.1. Se ha decidido no utilizarla por dos motivos:

1. Se trata de un sistema operativo que no es libre. Esto implica que la comunidad "Open Source", que suele realizar sus proyectos sobre software libre, no será una fuente de información.
2. ROS está principalmente desarrollado para Linux.

Queda claro que Windows no es una buena elección para este proyecto, así pues se ha decidido instalar Ubuntu en el portátil. La principal ventaja es que se trata de un sistema operativo libre, por lo que el software es libre, lo que implica que es accesible y modificable

por el usuario que lo descargue. Se ha procedido pues a particionar el disco duro y a instalar la versión de Ubuntu 14.04 LTS de 64 bits.

Para la implementación de la realidad aumentada es necesario disponer de una cámara. La elección de la cámara tenía que cumplir una serie de parámetros:

- Compatibilidad con ROS: era absolutamente imperativo que la cámara tuviera los drivers para poder ser integrada en ROS.
- Disponibilidad: preferiblemente se ha escogido una cámara que no fuese necesario acudir al mercado para comprarla.
- Económica: en caso de que las cámaras disponibles no cumplieran el punto 1 se adquiriría la cámara más económica.

Durante el desarrollo del proyecto se han utilizado dos cámaras. La primera de ella se ha utilizado durante el desarrollo de la aplicación en el ordenador personal, es decir, mientras no se trabajaba en el laboratorio del "ai2". En los dos casos se trata de cámaras webcam estándar por el simple motivo de que la mayoría de librerías son compatibles con este tipo de cámaras.

La primera cámara empleada es la que iba integrada en el portátil. La calibración, integración y configuración han sido muy fáciles y las especificaciones eran aceptables, pues en este momento del desarrollo se busca que la aplicación funcione, no una alta precisión. Más adelante en el laboratorio se integrará al proyecto una cámara con mejores prestaciones.

Una vez creada la aplicación de posicionamiento se ha ido al "ai2" a intentar implementarla al Summit XL. En el laboratorio se ha seleccionado una cámara con mejores especificaciones para que la aplicación sea mucho más precisa. La cámara utilizada es el modelo Logitech C920C cuyas características son las de la siguiente tabla:

Tabla 3: Especificaciones cámara.

Especificaciones	
Soporte UVC	Sí
Micrófono	Sí, estéreo
Enfoque	Automático
Campo visual	78º
Tipo conexión	USB 2.0

Longitud focal	3.67 mm
Resolución óptica	hasta 15 MP
Captura de imagen	2 MP, 3 MP, 6 MP, 15MP
Captura de vídeo	360p, 480p, 720p, 1080p
Frames/segundo (máximo)	1080p, 30fps
Dimensiones	94 mm x 24 mm x 29 mm, 162 gr

Finalmente se ha trabajado sobre el robot Summit XL de la empresa española Robotnik, pero la descripción de las características y especificaciones se ha realizado en el Capítulo 2, apartado 2.2.

3.3. Instalar y aprender a usar Ubuntu y ROS.

Durante el desarrollo del proyecto se han afrontado múltiples complicaciones e imprevistos. De los problemas más grande a los que se ha tenido que buscar una solución es el hecho de que nunca se había trabajado con el sistema operativo Ubuntu. Afortunadamente la web oficial del sistema operativo tiene una guía donde se explica perfectamente cómo instalar el sistema operativo en un ordenador (http://www.ubuntu.com/download/desktop).

Pero instalar Ubuntu era solo el primer paso, como nunca se había trabajado con este era necesario pues adquirir unos conocimientos mínimos del funcionamiento del sistema operativo. Para adquirir estos conocimientos se ha seguido una guía donde se explican las herramientas y funcionalidades básicas de Linux (http://www.ee.surrey.ac.uk/Teaching/Unix/).

La adaptación a Ubuntu puede parece un asunto sin importancia, pero ha requerido de una cantidad de tiempo superior a la inicialmente prevista. El hecho de tener que pasar de una interfaz gráfica a utilizar una terminal como modo de descarga de programas, como medio para escribir el código de los diferentes programas o como medio para navegar entre los directorios y crear nuevos ha supuesto una inversión de tiempo y esfuerzo relativamente grande.

Pero el verdadero problema ha llegado con ROS. Si bien es relativamente fácil de instalar, especialmente por el excelente tutorial de su web oficial (http://wiki.ros.org/ROS/Tutorials), es una herramienta que jamás se había utilizado en ninguna asignatura previa a la realización del trabajo. Asimilar el modelo modular puede ser difícil.

En la web uno puede encontrar una gran cantidad de tutoriales. Estos van desde un primer tutorial en el que se explica como instalar ROS y como configurarlo a los conceptos más avanzados. También se incluye tutoriales de algunas de las librerías disponibles en ROS. En todos los casos su lectura es muy provechosa.

No obstante, como consecuencia de los nulos conocimientos de ROS y lo comprimido del proyecto en el tiempo, era necesario escoger los tutoriales mínimos que requeriría el proyecto para ser desarrollado convenientemente. Así pues se han seguido minuciosamente los primeros 11 tutoriales del primer nivel. Gracias a esto se ha entendido los puntos básicos de ROS:

- Cómo se crean, construyen y funcionan los paquetes de ROS.
- El funcionamiento básico de los nodos.
- Entender la estructura de los topics y cómo se publican por terminal.
- La modularidad del sistema.
- El concepto básico detrás de los ficheros .launch.
- Cómo crear un nodo publisher y un nodo Subscriber.

Comprender el funcionamiento de ROS ha sido una ardua tarea que ha consumido una enorme cantidad de tiempo. Es importante resaltar este hecho pues cualquiera que quiera iniciar un proyecto que se base en ROS y no lo haya estudiado antes, como en el presente caso, debe saber que va a tener que dedicar una importante cantidad de tiempo a entender el funcionamiento del mismo

Finalmente un problema no tan grave que ha sido necesario solucionar es el hecho de que ROS está programado principalmente en C++ y en menor medida en Python, ambos lenguajes de programación desconocidos en el momento en el que empezó el proyecto. Ha sido necesario pues seguir unos tutoriales básicos para, al menos, tener una mínima idea de los conceptos básicos de C++ diferentes del lenguaje C, que sí se conoce. Buscando en Google se encuentran fácilmente cientos de tutoriales gratuitos. Se ha elegido este (no obstante, cualquier tutorial habría sido válido):

- http://www.cplusplus.com/doc/tutorial/

Adquiridos pues, con un gran esfuerzo, los conocimientos imprescindibles para poder crear una aplicación con lenguaje de programación C++ en ROS en un ordenador con Ubuntu, se procede a desarrollar la aplicación.

3.4. Selección de una librería de realidad aumentada.

El objetivo último de este proyecto es crear una aplicación de posicionamiento basada en el reconocimiento de marcadores. Así pues el primer paso a dar en el desarrollo del proyecto es seleccionar una librería que tenga entre sus funciones reconocer marcadores.

También es importante señalar que no se utilizará todo el potencial de la realidad aumentada. Como ya se ha explicado en el Capítulo 2 existen diferentes niveles de realidad aumentada y la principal característica de esta tecnología consiste en la adición de información de una realidad virtual a la realidad física, creando así una realidad mixta. Esta es una característica que no se va a implementar en el proyecto puesto que lo único que interesa es el reconocimiento de marcadores. Se necesitará pues una librería de realidad aumentada de nivel 1.

La librería seleccionada para el proyecto es aruco_ros, se trata de una librería de realidad aumentada basa en la librería de visión artificial OpenCV. Desarrollada por la Universidad de Córdoba en su investigación de la realidad aumentada se ajusta perfectamente a los requisitos de este proyecto. Además de tener una licencia BSD, ofrece la posibilidad de utilizar hasta 1024 marcadores de tamaño variable. Su integración con ROS es absoluta.

3.4.1. Marcadores de la librería.

Los 1024 marcadores de los que dispone la librería son cuadrados y sin ningún color. Están conformados por una serie de cuadrados blancos y negros que forman un patrón, el negro representa un 0 y el blanco un 1. Cada marcador tiene asociado un identificador único que va desde el 0 hasta el 1023. Este identificador se obtiene decodificando el código que forman los cuadrados blancos y negros. La codificación es una versión modificada del código Hamming.

No se aplica el código de Hamming sin variar porque de ese modo el marcador con identificador 0 sería un cuadrado negro, lo que provocaría muchos errores de identificación de falsos marcadores. Para evitar esto se ha cambiado el identificador del marcador 0.

Cada marcador tiene una matriz de cuadrados blancos y negros de 5 filas por 5 columnas, es decir, 5 palabras de 5 bits. Si se hacen las matemáticas salen un total de 33554432 posibles combinaciones pero al tener implementada la codificación Hamming solo 2 bits de cada palabra contienen información, el resto están para depurar errores

.

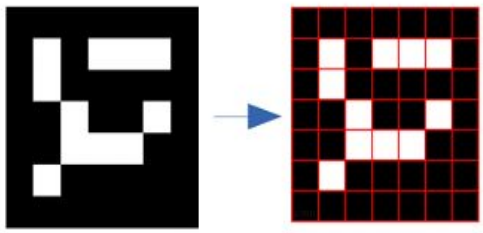


Figura 19: Marcador, obtención de las 5 palabras.

3.4.2. Mecanismo de detección de marcadores.

Aruco_ros tiene una rutina para identificar y decodificar los marcadore, elementos fácilmente detectables. La detección se realiza así:

1. Se obtienen los bordes de todos los objetos captados aplicando un umbral adaptativo (Figura 20).

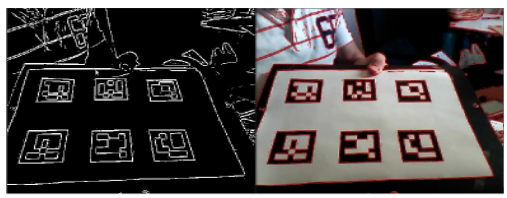


Figura 20.

2. Con este primer paso se han obtenido todos los bordes de la imagen, incluidos algunos que no son marcadores. El resto del proceso tiene como objetivo eliminar estos objetos no deseados. Para hacer esto la imagen es filtrada (Figura 21). Finalmente se ordenan los marcadores en sentido horario (Figura 22).

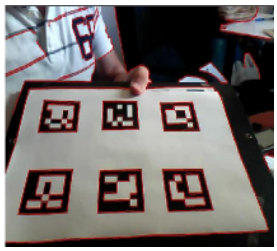


Figura 21..

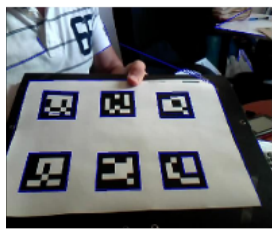


Figura 22.

3. Es posible que en el primer paso se hayan detectado rectángulos no solo en el contorno del marcador sino en su interior, es necesario eliminar el error para que únicamente permanezcan los rectángulos de la parte exterior del borde del marcador (Figura 23).

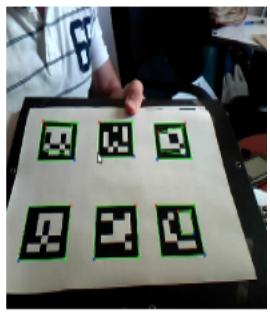


Figura 23.

4. Identificación del marcador:
    - Se elimina la perspectiva para poder trabajar con una imagen frontal del marcador (Figura 25).
    - Limitar el área utilizando el algoritmo de Otsu.
    - Cada marcador tiene una cuadrilla de cuadrados de 7x7 de los cuales el anillo exterior no es información del identificador del marcador. Lo primero que se hace es comprobar que existe este anillo negro para seguidamente leer el código del marcador y comprobar si es válido.

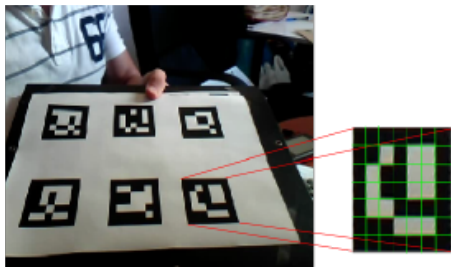


Figura 25

5. Una vez determinados los marcadores válidos se procede a refinar las esquinas realizando una interpolación subpixel.
6. Como último paso, si la cámara ha sido calibrada correctamente, se obtiene la posición relativa del marcador respecto a la cámara.

3.4.3. Nodos de Aruco_ros.

Es importante resaltar que aruco_ros es una adaptación de Aruco para ROS. La principal diferencia reside en el hecho de que Aruco lanza aplicaciones individuales a las que sería muy difícil comunicar con los nodos de nuestra aplicación. En cambio en aruco_ros todas las aplicaciones se lanzan como nodos que publican topics, por lo que la comunicación es muy sencilla.

Siguiendo la misma idea que todos los paquetes de ROS, aruco_ros es de código abierto, por lo que cualquier usuario puede descargarse el paquete, acceder al código del mismo y si fuera necesario modificarlo. Todo esto convierten a este paquete en ideal para este proyecto, pues se podrá modificar el código necesario y su integración con ROS es absoluta.

También es importante resaltar las diferencias entre aruco_ros, herramienta que se empleará en este proyecto, y Aruco: no se han adaptado todas las funciones de Aruco, aquellas que no tienen sentido existir en ROS (como las relacionadas con Android), no están. Todas las aplicaciones de Aruco se han reducido a 3 nodos independientes, estos son los archivos que contienen su código:

1. Simple_single.cpp: con este fichero se lanza el nodo aruco_single. Este nodo permite reconocer un marcador en tiempo real a partir de la información de una cámara. En su archivo se pueden modificar bastantes parámetros, pero los dos más importantes son el tamaño de marcador y el identificador. Si bien es cierto que el nodo detectará

todos los marcadores presentes en la imagen solo publicará información de aquel marcador cuyo identificador coincida con el indicado en el código del nodo.

De todos los topics con información que publica el más interesante para este proyecto es el que indica la posición y rotación relativa del marcador respecto a la cámara. Finalmente hay que aclarar que no hay que confundir el nombre del nodo, aruco_single, con el nombre del archivo con el código del nodo, simple_single.cpp.

2. Simple_souble.cpp: con este fichero se lanza el nodo aruco_double, de nuevo no confundir nombre del nodo con nombre del fichero con su código. El funcionamiento es idéntico al caso anterior, pero ahora hay que pasarle los parámetros de tamaño e identificación de dos marcadores. El nodo publicará topics con la información de los dos marcadores.

3. Marker_publish.cpp: este fichero lanza el nodo aruco_marker_publisher. La principal diferencia entre este nodo y los dos anteriores es que solo necesita como parámetro el tamaño de los marcadores. Esto implica que el nodo no solo reconoce todos los marcadores que haya en la imagen de la cámara, también publica, entre otra información, la posición relativa de cada marcador respecto a la cámara.

De los tres nodos disponibles el que más interesa para el desarrollo del presente proyecto es el tercero. Si bien es cierto que es el más complejo es el más flexible y por lo tanto el que mejor se adapta a la aplicacóin que se está creando. Recordando el objetivo último del proyecto, que es crear una aplicación de posicionamiento de robots basada en marcadores, es de suponer que en la trayectoria del robot este se encontrará con una gran cantidad de marcadores. Lo necesario en este caso es que el robot sea capaz de identificar correctamente cualquier marcador, pues no se puede anticipar que marcador se encontrará y lanzar un nodo por cada marcador es una idea absurda. Así pues se decide utilizar el nodo aruco_marker_publisher.

3.5. Configuración de de aruco_ros.

Seleccionada la librería que se va implementar y el nodo de esta que se utilizará el siguiente paso en el proceso es instalar la librería. No obstante no se puede instalar aruco_ros directamente o, al menos, no sería suficiente. Una de las características más importantes de

ROS es la modularidad de sus aplicaciones. Gracias a esto sus aplicaciones son fácilmente trasladables de un proyecto a otro.

No obstante esa característica implica que cada aplicación realice una serie de funciones muy concretas y que, como en este caso, sea necesario instalar otros paquetes complementarios que subsanen la carencia de ciertas funciones. En este caso para que aruco_ros funciona correctamente es necesario tener disponible un topic con la información de la cámara. Para esto es necesario instalar los paquetes OpenCV y usb_cam.

3.5.1. Instalación del paquete usb_cam.

La instalación de OpenCV es más compleja y será explicada detalladamente en los anexos. En cuanto a la instalación del paquete usb_cam es muy sencilla. Como este se encuentra en los repositorios oficiales de ROS será tan sencillo como ejecutar el siguiente comando en una terminal de ubuntu:

- `sudo apt-get install ros-hydro-usb-cam`

Para comprobar que se ha instalado correctamente podemos lanzar el nodo del paquete. Para hacer esto primero es necesario lanzar el máster de ROS con el comando:

- `roscore`

Siempre que se empiece a trabajar con ROS y no se haya ejecutado el anterior comando en la sesión de trabajo será necesario ejecutarlo. Ahora, abriendo una nueva terminal, lanzamos el nodo usb_cam_node del paquete usb_cam ejecutando el siguiente comando:

- `rosrun usb_cam usb_cam_node`

Para verificar que el nodo está funcionando correctamente se puede ejecutar el siguiente comando:

- `rosnode list`

con el que se debería ver esto:

```
/rosout
/usb_cam
```

Para ver una lista de los topics publicados debe ejecutarse el siguiente comando:

- `rostopic list`

con lo que debería verse la siguiente lista:

```
/rosout
/rosout_agg
/usb_cam/camera_info
/usb_cam/image_raw
/usb_cam/image_raw/compressed
/usb_cam/image_raw/compressed/parameter_descriptions
/usb_cam/image_raw/compressed/parameter_updates
/usb_cam/image_raw/compressedDepth
/usb_cam/image_raw/compressedDepth/parameter_descriptions
/usb_cam/image_raw/compressedDepth/parameter_updates
/usb_cam/image_raw/theora
/usb_cam/image_raw/theora/parameter_descriptions
/usb_cam/image_raw/theora/parameter_updates
```

Al ejecutar el nodo usb_cam es muy probable que salga un warning indicando la carencia de un fichero de calibración. Este fichero es imprescindible si se quiere conseguir que la posición relativa del marcador respecto a la cámara sea precisa. Pero antes de nada es posible comprobar que el nodo capta correctamente la imagen. Manteniendo activo el nodo usb_cam ejecutamos en una nueva terminal el siguiente comando:

- `rqt_image_view`

Con esto se abrirá una ventana donde se podrá visualizar las imágenes captadas por la cámara. Así pues se procede a calibrar la cámara, para lo que es necesario instalar las dependencias requeridas ejecutando con los dos siguientes comandos en un nuevo terminal:

- `rosdep install camera_calibration`
- `rosmake camera_calibration`

Ya es posible realizar la calibración. Para llevarla a cabo es necesario disponer de un tablero tipo ajedrez, es decir, un patrón de cuadrados blancos y negros. Se puede utilizar un tablero de ajedrez o se puede descargar un modelo. En este caso se ha utilizado un tablero de ajedrez. La aplicación de calibración requiere de tres parámetros: la imagen de la cámara, el número de cuadrados interiores y el tamaño de los mismos.

Para realizar la calibración se tiene que lanzar previamente el nodo usb_cam y en una nueva terminal ejecutar el siguiente comando (es recomendable hacer esto en una habitación amplia y con buena iluminación):

- `rosrun camera_calibration cameracalibrator.py --size 7x7 --square 0.04 image:=/usb_cam/image_raw camera:=/usb_cam`

El número de cuadrados y su tamaño dependerá del tablero utilizado. Al ejecutar el comando aparecerá una ventana con la imagen de la cámara y una serie de barras a la derecha. Para calibrar la cámara hay que mover el tablero enfrente de la misma hasta que todas las barras se llenen y se pongan de color verde. Cuando se de esta situación hay que clicar a save.

Ahora hay que localizar el archivo con los datos de la calibración. Para ello se mira el terminal donde se ha ejecutado la anterior orden y se sigue la ruta indicada. Se mueve el archivo .yaml al workspace y se cambia su nombre a camera.yaml. Más adelante se le indicará al nodo usb_cam donde debe buscar el archivo de calibración.

3.5.2. Instalación de aruco_ros.

Con los dos paquetes necesarios para el correcto funcionamiento de aruco_ros correctamente instalados se produce pues a isntalar aruco_ros. En este caso no se encontrará al paquete entre los repositorios oficiales, por lo que se podrá implementar la misma metodología de instalación.

Así pues será necesario obtener el paquete desde internet directamente, descargarlo en el espacio de trabajo (directorio src dentro del catkin_ws), esto se hace con los comandos:

- `cd ~/catkin_ws/src`
- `git clone https://github.com/pal-robotics/aruco_ros.git`

El primer comando sirve para cambiar el directorio al espacio de trabajo y el segundo sirve para descargar el paquete. Ahora es necesario compilar el paquete par que pueda ser utilizado. Para esto ejecutamos los siguientes comandos:

- `cd ..`
- `catkin_make`

Ahora ya se disponen de todas las herramientas para construir la aplicación que calcule la posición del robot a partir del reconocimiento de marcadores. Pero antes se generará rápidamente un fichero .launch que lance el nodo usb_cam y el nodo aruco_marker_publisher. El fichero .launch, que se guardará en la carpeta launch dentro de aruco_ros, contiene el siguiente código:

```
<launch>
<node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
output="screen" >
<param name="video_device" value="/dev/video0" />
<param name="image_width" value="1280" />
<param name="image_height" value="720" />
<param name="pixel_format" value="mjpeg" />
<param name="framerate" value="30" />
<param name="camera_frame_id" value="usb_cam" />
<param name="io_method" value="mmap"/>
<param name="camera_info_url" value="
file:///home/blai/catkin_ws/camera.yaml"/>
<param name="camera_name" value="narrow_stereo"/>
</node>
<arg name="markerSize1" default="0.184"/> <!-- en metros -->
<arg name="eye1" default="left"/>
<arg name="marker_frame1" default="aruco_marker_frame"/>
<arg name="ref_frame1" default=""/>
<node pkg="aruco_ros" type="marker_publisher" name="topo">
<remap from="/camera_info" to="/usb_cam/camera_info" />
```

```
<remap from="/image" to="/usb_cam/image_raw" />
<param name="image_is_rectified" value="True"/>
<param name="marker_size" value="$(arg markerSize1)"/>
<param name="reference_frame" value="$(arg ref_frame1)"/>
<!--frame in which the marker pose will be refered -->
<param name="camera_frame" value="stereo_gazebo_$(arg
eye1)_camera_optical_frame"/>
<param name="marker_frame" value="$(arg marker_frame1)" />
<param name="frame_id" value="100" />
</node>
</launch>
```

Se han puesto en negrita los valores que se deben personalizar para cada caso concreto, serán descritos a continuación por orden de aparición en el código:

1. dev/video0: sirve para escoger la cámara de donde se obtendrá la imagen. Normalmente es dev/video0, pero si se tiene una webcam por usb en un portátil con webcam integrada debe ponerse dev/video1.
2. 1280 y 720: la resolución de la imagen.
3. home/blai/catkin_ws/: indica el directorio donde se tiene guardado el fichero de calibración.
4. 0.184: tamaño de los marcadores a identificar.
5. topo: se ha decidido cambiar el nombre del nodo original a uno más manejable..

Se lanza el fichero, de nombre nodo.launch, para comprobar que todo funciona correctamente. Se ejecuta en un terminal el comando:

- `roslaunch aruco_ros nodo.launch`

Entonces en tes terminales nuevos ejecutamos estos tres comandos:

- `rqt_image_view`
- `rqt_graph`
- `rostopic echo topo/markers`

La función del primer comando ya ha sido descrito. En cuanto a los otros dos el primero permite ver un diagrama con todos los nodos ejecutándose y los topics que los conectan. En este caso se debería ver esto:

Figura 26.

El primer y tercer comando muestran por pantalla la imagen de la cámara y el contenido del topic con la información de la posición de los marcadores:

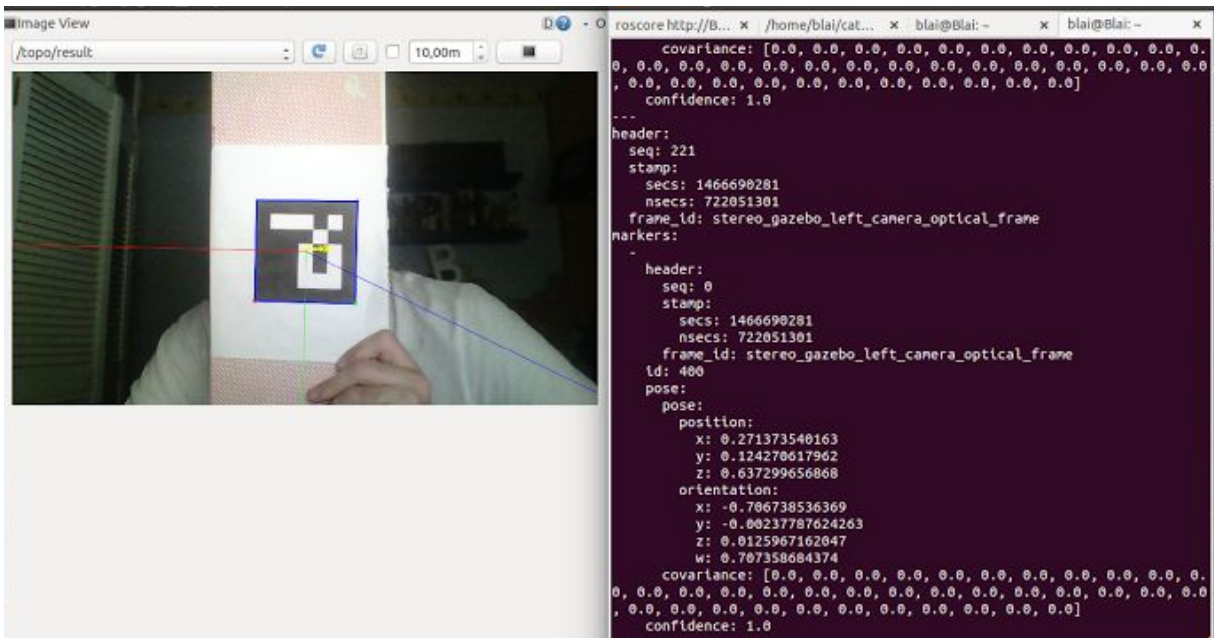


Figura 27

Se observa pues que el reconocimiento de marcadores funciona correctamente. En la parte derecha de la imagen se tiene el topic publicado por el nodo topo, este contiene la distancia del marcador a la cámara, la orientación del mismo y la covarianza.

La instalación y configuración de aruco_ros se ha realizado correctamente. Ahora ya se dispone de la posición relativa del marcador respecto a la cámara, el siguiente paso es aplicar funciones matemáticas para obtener la posición de la cámara.

3.6. Desarrollo matemático e implementación.

3.6.1. Planteamiento.

El concepto clave que se aplicará para calcular la posición absoluta del robot a partir de la distancia relativa del marcador respecto a la cámara es realmente muy sencillo pero extremadamente difícil de visualizar. Para ello se utilizará una imagen en dos dimensiones, en lugar de las 3 con las que trabajará la aplicación, para que la comprensión del concepto sea más sencilla.

Se desarrollará un nodo con una serie de caĺculos matemáticos, principalmente trigonométricos, para realizar dos transformaciones consecutivas. La primera de ellas consiste en transformar el vector relativo que tiene la posición relativa del marcador respecto a la cámara en un vector que tenga la posición relativa de la cámara respecto al marcador. Esto implica una traslación del sistema de coordenadas de la cámara y un giro del mismo:

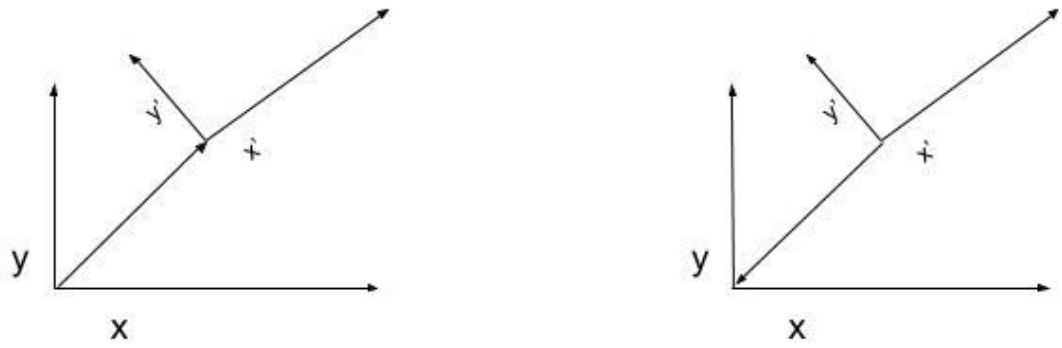


Figura 28

En la figura se observa como se le “da la vuelta” al vector. Se pasaría de tener las coordenadas del vector respecto al eje de coordenadas de la cámara (x,y) a tenerlo respecto al eje de coordenadas del marcador (x’,y’). De este modo se tendría la posición relativa de la cámara respecto al marcador.

Una vez se tiene esta posición relativa y conociendo la posición y orientación del marcador en el mapa del edificio, ciudad o donde se quiera localizar el robot, solo se tienen que repetir exactamente los mismos cálculos, solo que ahora se gira y traslada el vector para que esté en las coordenadas del sistema global. De esta manera se obtendría la posición absoluta del robot basándose la aplicación únicamente en el reconocimiento de marcadores. Como se está trabajando en el porttil y no en el robot esta transformación se implementará más adelante, es decir, primero se creará una aplicación que realice la primera transformación y después una segunda aplicación que calcule la posición absoluta del robot.

3.6.2. Cálculo matemático.

El siguiente paso es conseguir las fórmulas matemáticas que implementes la traslación y giro necesarios descritos en el apartado anterior. Estas fórmulas ya existen y por tanto no es necesario desarrollarlas para este proyecto.

Se ha realizado una úbsqueda en internet y se ha encontrado el siguiente documento: http://es.slideshare.net/camiloasilva3/rotacin-matricial donde se expone toda la informaciń necsaria para este proyecto sobre matrices de rotación. Antes de explicar la propiedad más importante de las matrices de rotación se presentarán las matrices de giros simple, es decir, cuando se quiere corregir un giro respecto al eje x, y o z (figura).

$$ROT(x,y,\varphi) = \begin{pmatrix} Cos(\varphi) & Sen(\varphi) & 0 \\ -Sen(\varphi) & Cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$ROT(x,z,\varphi) = \begin{pmatrix} Cos(\varphi) & 0 & Sen(\varphi) \\ 0 & 1 & 0 \\ -Sen(\varphi) & 0 & Cos(\varphi) \end{pmatrix}$$

$$ROT(y,z,\varphi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & Cos(\varphi) & Sen(\varphi) \\ 0 & -Sen(\varphi) & Cos(\varphi) \end{pmatrix}$$

Figura 29

La principal propiedad de las matrices de rotación simples es que se pueden multiplicar para generar así una matriz de giro compuesto. En este proyecto no es posible predecir el giro que habrá que aplicar al vector posición, por lo que se requiere de una matriz que calcule las

nuevas coordenadas del vector para cualquier combinación de giro posible. Así pues se han multiplicado las 3 matrices dando como resultado (se sustituye cos por C y sin por S):

$$\mathrm{ROT} = \begin{bmatrix} C\phi C\theta & C\phi S\theta S\psi - S\phi C\psi & C\phi S\theta C\psi + S\phi S\psi \\ S\phi C\theta & S\phi S\theta S\psi - C\phi C\psi & S\phi S\theta C\psi - S\phi S\psi \\ -S\theta & C\theta S\psi & C\theta C\psi \end{bmatrix}$$

Figura 30

3.6.3. Implementación en un nodo.

El siguiente paso es escribir un el código de un nodo que se encargue de calcular los giros y traslaciones. En este momento hay dos problemas a solucionar: al escribir el código no se pueden implementar matrices y la orientación que se obtiene del nodo topo está en una rotación con cuaterniones. El siguiente fragmento de código pertenece al nodo y calcula la posición relativa del robot respecto al marcador:

```
ros::Publisher pub;
geometry_msgs::PoseWithCovariance absoluta;
float matriz[5][8];

void chatterCallback(const aruco_msgs::MarkerArray::Ptr msg)
{
float x2,y2,z2,x1,y1,z1,x, y, z,alfa, beta, gamma, absoluto;
int i;
i=msg->markers[0].id;
i=(i/100);
x=-msg->markers[0].pose.pose.position.x;
y=-msg->markers[0].pose.pose.position.y;
z=-msg->markers[0].pose.pose.position.z;
tf::Quaternion q(msg->markers[0].pose.pose.orientation.x,
msg->markers[0].pose.pose.orientation.y,
msg->markers[0].pose.pose.orientation.z,
msg->markers[0].pose.pose.orientation.w);
```

```
      tf::Matrix3x3 m(q);
      double roll, pitch, yaw;
      m.getRPY(roll, pitch, yaw);

alfa=pitch;
beta=roll;
gamma=yaw;

x1=(cos(beta))*(cos(gamma))*x-(cos(beta))*(sin(gamma))*y+(sin(beta))
*z;
y1=((cos(alfa))*(sin(gamma))+(sin(alfa))*(sin(beta))*(cos(gamma)))*x
+((cos(alfa))*(cos(gamma))-(sin(alfa))*(sin(beta))*(sin(gamma)))*y-(
(sin(alfa))*(cos(beta)))*z;
z1=((sin(alfa))*(sin(gamma))-(cos(alfa))*(sin(beta))*(cos(gamma)))*x
+((sin(alfa))*(cos(gamma))+(cos(alfa))*(sin(beta))*(sin(gamma)))*y+(
(cos(alfa))*(cos(beta)))*z;

absoluta.pose.position.x=x1;
absoluta.pose.position.y=y1;
absoluta.pose.position.z=z1;
absoluta.covariance[0]=0;
}

int main(int argc, char **argv)
{
 ros::init(argc, argv, "buscador");
 ros::NodeHandle n;
 ros::Rate loop_rate(100);

 ros::Subscriber sub = n.subscribe("/topo/markers", 1000,
chatterCallback);
 ros::Publisher
pub=n.advertise<geometry_msgs::PoseWithCovariance>("abs", 100);
 while (ros::ok())
      { absoluta.covariance[0]=10000;
      pub.publish(absoluta);
      ros::spinOnce();
```

```
      loop_rate.sleep();
      }
 return 0;
}
```

Después de las fórmulas matemáticas que calculan el posicionamiento del robot el código más importante del programa es este:

```
tf::Quaternion q(msg->markers[0].pose.pose.orientation.x,
msg->markers[0].pose.pose.orientation.y,
msg->markers[0].pose.pose.orientation.z,
msg->markers[0].pose.pose.orientation.w);
      tf::Matrix3x3 m(q);
      double roll, pitch, yaw;
      m.getRPY(roll, pitch, yaw);
```

En este código se transforma la rotación con cuaterniones a ángulos de Euler, más manejables. Escrito el código se modifica el fichero CMakeLists.txt para que al compilar se incluya el nodo. Para ello se añade esta linea al fichero de texto:

```
add_executable(distancia_marcador src/distancia_marcador.cpp)
target_link_libraries(distancia_marcador ${catkin_LIBRARIES})
```

Ahora compilamos el código con el comando:

- `catkin_make`

3.6.4. Comprobación nodo.

Para comprobar que todo va correctamente lanzamos el nodo topo y ejecutamos en una nueva terminal el siguiente comando:

- `rosrun aruco_ros distancia_marcador`

Si se ejecuta el comando en una nueva terminal:

- `rostopic echo abs`

Se observa por pantalla las coordenadas de la posición relativa del robot respecto al marcador. Una manera rápida de comprobar que no hay un error es comparar el valor absoluto de la posición del marcador respecto de la cámara con la posición de la cámara respecto del marcador. Esto se hace añadiendo al código anterior la siguientes líneas:

```
float err;
err=sqrt(x1*x1+y1*y1+z1*z1)-sqrt(x*x+y*y+z*z);
absoluta.pose.orientation.x=err;
```

Al haber modificado el código del nodo es necesario compilarlo de nuevo. Una vez hecho esto se lanzan los nodos exactamente igual que antes y se observa el valor del error. Este tiene un orden de magnitud de 10^(-8) metros, por lo ques es seguro afirmar que el error es despreciable.

3.7. Implementación de la aplicación en el robot Summit XL.

3.7.1. Planteamiento.

Con la posición relativa del robot respecto al marcador ya calculada es el momento de instalar la aplicación en el robot y de modificarla para que calcule la posición absoluta del robot. Para realizar esto se ha trabajado en el "ai2" en un portátil diferente. Este portátil establecerá una conexión ssh con el Summit XL para operar directamente en su ordenador. Es de gran ayuda que el Summit XL ya tenga instalados los paquetes OpenCV, usb_cam y aruco_ros.

El principal añadido en la aplicación que se ejecutará en el Summit XL es que necesitará, de alguna manera, conocer la posición y orientación de los marcadores que se encuentre en su desplazamiento.

Una vez modificado la aplicación para que calcule la posición absoluta del robot se ejecutarán una serie de comandos en una terminal para poder acceder a los topics del Summit XL de tal manera que se pueda visualizar la imagen en directo de la cámara del mismo.

3.7.2. Modificación de la aplicación.

Serán necesarios unas pequeñas modificaciones en el código del nodo topo. No obstante antes de modificarlo se creará una copia del mismo, de modo se tendrán dos aplicaciones. La primera tendrá las mismas funciones que la del apartado anterior y la segunda será la que se modifique, con nombre posicionamiento.cpp.

Se procede pues a trabajar en la modificación de la aplicación. La primera de ellas es añadir una segunda tanda de transformación, pues se pretende obtener el posicionamiento respecto a un punto de origen, obteniendo así la posición del robot.

Así pues se añaden las fórmulas matemáticas con un cambio. Es necesario que el robot no solo reconozca el identificador de los marcadores que vea mientras se mueve, también es es necesario que conozca la posición y orientación de estos. Para esto se ha decidido crear un fichero de texto con la posición y orientación de cada marcador. Este fichero podría ser así (se presenta un ejemplo con 5 marcadores posicionados):

Tabla 4.

Identificador	Posición (metros)			Orientación (radianes)		
	x	y	z	alfa	beta	gamma
1	6.4	2.6	0.6	2.6	2.0	0.6
2	12.6	9.6	0.8	3.6	2.6	0.9
3	16.5	4.8	0.7	2.6	3.4	1.8
4	22.6	16.6	1.0	0.0	3.14	2.9
5	2.6	23.6	0.3	0.6	3.6	2.6

Para que la aplicación pueda leer el fichero de datos se añade la siguiente función:

```
void leer(void);
float matriz[5][7];
```

```
void leer(void){ //esta función lee la posición de los marcadores
FILE *f
=fopen("/home/summit/catkin_ws/src/aruco_ros/aruco_ros/src/posicion.
txt", "r");
int i, j, x;
for(i=0;i<5;i++){
for(j=0;j<7;j++){
      x=fscanf(f, "%f", &matriz[i][j]);
printf("%f \n", matriz[i][j]); }}}
```

Esta función será llamada únicamente una vez desde la función main. De esta manera se permite la reubicación de los marcadores sin la tener que volver a compilar la aplicación. Será tan sencillo como apagar la aplicación, modificar el fichero de datos y volver a lanzar la aplicación. La aplicación muestra en pantalla los datos leídos del fichero para que se pueda verificar que se ha leído correctamente la posición de los marcadores.

En negrita quedan los factores que se pueden modificar. El primero es la ruta del archivo con el posicionamiento de los marcadores. El segundo factor es el tamaño de la matriz a leer del documento.

La segunda transformación queda pues así:

```
alfa=matriz[i][4];
beta=matriz[i][5];
gamma=matriz[i][6];

x2=(cos(beta))*(cos(gamma))*x1-(cos(beta))*(sin(gamma))*y1+(sin(beta
))*z1;

y2=((cos(alfa))*(sin(gamma))+(sin(alfa))*(sin(beta))*(cos(gamma)))*x
1+((cos(alfa))*(cos(gamma))-(sin(alfa))*(sin(beta))*(sin(gamma)))*y1
-((sin(alfa))*(cos(beta)))*z1;

z2=((sin(alfa))*(sin(gamma))-(cos(alfa))*(sin(beta))*(cos(gamma)))*x
1+((sin(alfa))*(cos(gamma))+(cos(alfa))*(sin(beta))*(sin(gamma)))*y1
+((cos(alfa))*(cos(beta)))*z1;
```

Finalmente se modifica la información que se va a publicar, quedando así:

```
absoluta.pose.position.x=x2+matriz[i][1];
absoluta.pose.position.y=y2+matriz[i][2];
absoluta.pose.position.z=z2+matriz[i][3];
```

Así pues la aplicación se ha programado correctamente. El siguiente paso es compilar, pero para hacer esto primero es necesario enviar el nuevo código al Summit XL.

3.7.3. Conexión con el Summit XL.

Establecer una conexión con el Summit XL es relativamente sencillo y se establecerá esta conexión para realizar dos cosas:

1. Enviar el nuevo código al Summit XL.
2. Compilar en el workspace del Summit XL la aplicación.

Los dos objetivos a cumplir en este apartado necesitan de unos comandos diferentes. Para poder enviar un archivo desde el portátil al Summit XL primero hay que encender este último. Esto se hace girando la rueda verde y después pulsando el botón azul hasta que se observe una luz. Entonces el Summit XL emitirá una red Wi-Fi a la que hay que conectarse desde el ordenador, entonces se enviará el fichero usando los cuatro siguientes comandos:

- `scp ~/catkin_ws/src/aruco_ros/aruco_ros/src/distancia_marcador.cpp summit@192.168.0.200:~/catkin_ws/src/aruco_ros/aruco_ros/src`

- `scp ~/catkin_ws/src/aruco_ros/aruco_ros/src/posicionamiento.cpp summit@192.168.0.200:~/catkin_ws/src/aruco_ros/aruco_ros/src`

- `scp ~/catkin_ws/src/aruco_ros/aruco_ros/launch/nodo.launch summit@192.168.0.200:~/catkin_ws/src/aruco_ros/aruco_ros/src`

- `scp ~/catkin_ws/src/aruco_ros/posicion.txt summit@192.168.0.200:~/catkin_ws/src/aruco_ros/aruco_ros/src`

Se establece ahora una conexión ssh con el Summit XL, esto se hace abriendo un terminal y ejecutando el siguiente comando:

- `ssh summit@192.168.0.200`

Hay que escribir la contraseña, que es: `R0b0tn1K.` Entonces se verá por pantalla esto:

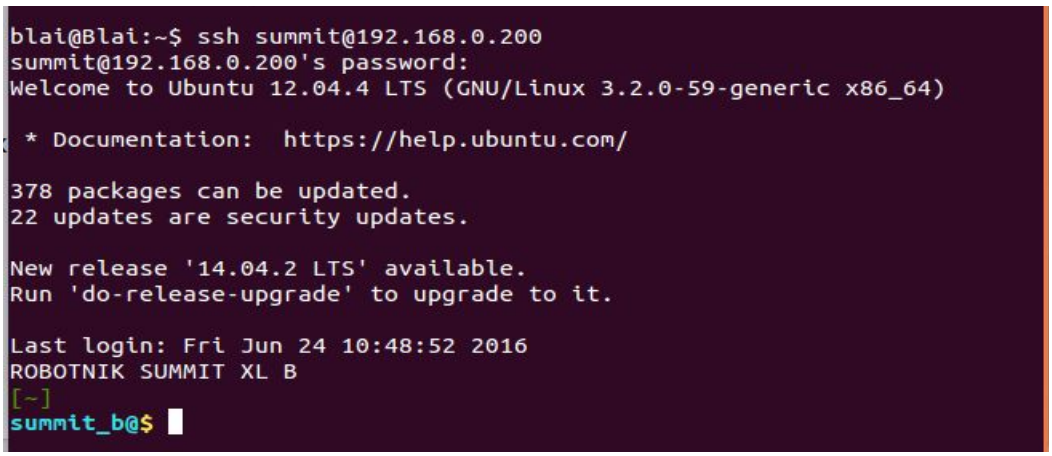


Figura 31

Así pues ya se ha establecido una conexión ssh con el Summit XL. Es importante resaltar que por cada terminal que queramos tener conectado al robot se tendrá que repetir este proceso. Antes de poder compilar el código en el workspace del Summit XL es necesario modificar el archivo CMakeLists.txt añadiendo estas líneas:

```
add_executable(distancia_marcador src/distancia_marcador.cpp)
target_link_libraries(distancia_marcador ${catkin_LIBRARIES})

add_executable(posicionamiento src/posicionamiento.cpp)
target_link_libraries(posicionamiento ${catkin_LIBRARIES})
```

Ahora ya casi se puede lanzar la aplicación en el robot. Se debe compilar exactamente como se ha hecho hasta ahora. Una vez se ha compilado con éxito se procede a calibrar la cámara, pues el Summit XL no incorpora ninguna, así que se utilizará una cámara Logitech C920C. Esto ya se ha explicado en apartados anteriores y simplemente hay que seguir el mismo proceso. Finalmente hay que modificar el fichero nodo.launch, pues ahora el fichero con los datos de la calibración está en otro directorio.

3.7.4. Comprobación de la aplicación en el Summit XL.

El último paso en este proyecto es comprobar que la aplicación funciona en el Summit XL. No obstante, por la falta de tiempo, solo se comprobará que funciona la aplicación que calcula la posición relativa del robot respecto a los marcadores. No da tiempo a crear correctamente el fichero con el posicionamiento de los marcadores. Esto no supone ningún problema, pues desde un punto de vista matemático es repetir los mismos cálculos, por lo que si el Summit XL es capaz de realizarlos una vez será perfectamente capaz de realizarlos dos veces.

Para lanzar la aplicación dentro del Summit XL se accede al ordenador del mismo como ya se ha explicado y, en un nuevo terminal que no esté conectado al Summit XL se ejecutan los siguientes comandos:

- `export ROS_MASTER_URI=http://192.168.0.200:11311`
- `export ROS_IP=192.168.0.50`

Gracias a esto se puede acceder al topic del nodo topo por pantalla, es decir, se puede visualizar en la pantalla del portátil la imagen, con cierto retraso, de la cámara del Summit XL. Ahora se pegará por el laboratorio un marcador de 20 cm de lado y utilizando un mando de PS3 se moverá el robot por los pasillos para que capte los marcadores, el resultado obtenido es el siguiente (se trata de un montaje con la imagen captada por la cámara y el vector posición calculado):

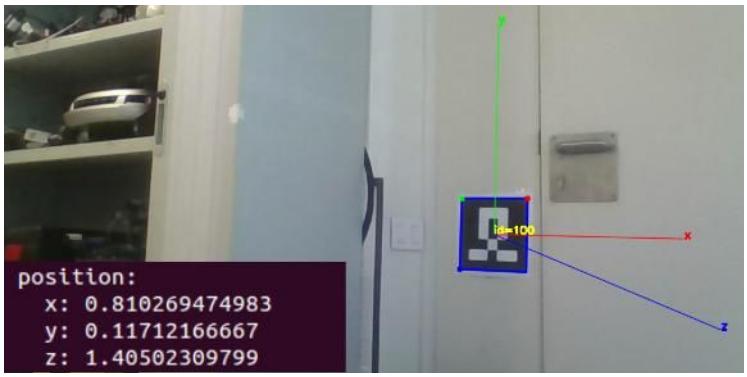


Figura 32.

CAPÍTULO 4: CONCLUSIONES Y FUTUROS PROYECTOS.

Acabada el trabajo práctico del proyecto es posible establecer una serie de conclusiones del trabajo realizado durante el desarrollo del mismo y una evaluación del cumplimiento de los objetivos establecido al inicio de este.

Antes que nada es razonable establecer que se ha cumplido el objetivo principal de este proyecto. La aplicación de posicionamiento de robots basada en el reconocimiento de marcadores funciona perfectamente y se ha hecho todo trabajando siempre con ROS.

Se han estudiado las infinitas posibilidades de la visión artificial y de la realidad aumentada. LA librería usada, aruco_ros, es tan versátil como potente siendo, al mismo tiempo, relativamente fácil de instalar y muy fácil de configurar. El resultado de implementar esta tecnología ha sido muy satisfactorio, pues los resultados son excelentes.

Mención especial se merece. La opinión que se ha ido construido a lo largo del desarrollo de este proyecto sobre esta herramienta es que es muy potente, versátil y relativamente fácil de implementar. Es una alternativa que se debería considerar muy seriamente siempre que se quiera iniciar un proyecto con robots. Es necesario que los ingenieros tengan conocimiento de herramientas como esta.

Repasando los objetivos concretos establecidos se puede concluir:

- Se ha conseguido instalar y entender tanto Ubuntu como ROS.
- Se ha logrado instalar aruco_ros junto con las librerías complementarias necesarios y configurarlas correctamente. Se ha entendido el funcionamiento básico de estas.
- Se ha desarrollado una aplicación que tomando los datos de aruco_ros, es decir, la posición relativa del marcador respecto al robot, calcule la posición absoluta del robot en un mapa o en el globo terráqueo.
- Se ha instalado correctamente la aplicación en el robot Summit XL. Se ha aprendido a enviar paquetes desde el ordenador al Summit XL, compilarlos y ejecutarlos en este.

También se ha aprendido a emitir en directo las imágenes que capta la cámara del Summit Xl en la pantalla del portátil.

- Se ha observado la capacidad que tiene ROS de dotar a sus aplicaciones de una reusabilidad absoluta, pudiendo instalarse la aplicación desarrollada en el presente trabajo en cualquier robot con ROS.

Por lo tanto se puede afirmar perfectamente que se han cumplido todos los objetivos inicialmente establecidos para este proyecto, lo que demuestra que todo el trabajo realizado está justificado. Además se han desarrollado los principios básicos para que se desarrollen nuevas aplicaciones.

Para empezar podría plantearse sustituir la cámara web por tecnología más compleja como cámaras de alta velocidad, cámaras con un ángulo notoriamente superior (de 360º) o cámaras nocturnas.

Una aplicación muy interesante sería añadir trabajo realizado la librería robot_localization. Esta herramienta permite fusionar los datos provenientes de los sensores, por lo que podría realizarse un seguimiento de todos los puntos de posición del robot.

La evolución de esta aplicación más interesante requeriría de muchísimo tiempo y esfuerzo pero supondría un paso cualitativo de incalculable valor en la creación de sistemas de conducción autónoma. . La idea sería poner cámaras y marcadores a cada coche, de maner que cada coche pueda no solo calcular la distancia a los coches mś cercanos, sino intercambiar la posición de los coches cercanos. Con esta aplicación cada coche obtendría la posición y trayectoria de los coches que le rodean.

No obstante, por la falta de tiempo, ninguna de estas ideas se ha podido implementar en el presente proyecto. Así pues queda abierta la puerta para que cualquiera continúe con cualquiera de estas líneas de investigación, pues el potencial de ROS es infinito.

CAPÍTULO 5: BIBLIOGRAFÍA

[1] Británica Global Edition: búsqueda definición de robot.
[2] Británica Global Edition: historia del robot.
[3] Asociación de Robots Japonesa, http://robotec11.tripod.com/id4.html
[4] Asociación Francesa de Robótica Industrial, http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/industrial.htm
[5] Robots industriales, http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.html
[6] Imágen de un robot manipulador, https://www.logismarket.com.ar/hurtado-rivas/robot-manipulador/2132765000-1179608940-p.html
[7] Imágen de un robot manipulador, http://www.robolan.net/productos/ver_producto.php?Nfamilia=1080559510&Nproduct=1083075955&elemento_inicial=12
[8] Información de los robots móviles: https://en.wikipedia.org/wiki/Mobile_robot
[9] Información de los robots móviles: http://eii.unex.es/profesores/jisuarez/descargas/otras/Robotmov.pdf
[10] Información de los robots móviles: Apuntes de Laboratorio de Automatización y Control (2015). Ángel Valera. Introducción a la robótica.
[11] Características Summit XL: http://www.robotnik.eu/mobile-robots/summit-xl/
[12] Información de lo que es el middleware: https://es.wikipedia.org/wiki/Middleware
[13] Definición de ROS: http://www.ros.org/about-ros/
[14] Información de las distribuciones de ROS: http://wiki.ros.org/Distributions
[15] Información del comando rosmake: http://wiki.ros.org/rosmake
[16] Comandos más importantes en ROS: http://wiki.ros.org/ROS/CommandLineTools
[17] Red de Satélites: http://www.gpsdemontana.com/2014/03/que-significa-que-un-gps-tenga-hotfix-y.html
[18] Funcionamiento GPS: http://www.asifunciona.com/electronica/af_gps/af_gps_10.htm
[19] Error provocado por la atmósfera en el GPS: http://www.mio.com/technology-gps-accuracy.htm
[20] Error provocado por la atmósfera en el GPS: http://www.alsitel.com/tecnico/gps/errores.htm
[21] Error GPS edificios: http://file.scirp.org/Html/7-8501064_31523.htm
[22] GPS Signal Short-Term Propagation Characteristics Modeling in Urban Areas for Precise Navigation Applications: http://file.scirp.org/Html/7-8501064_31523.htm
[23] Choke ring antenna: https://en.wikipedia.org/wiki/Choke_ring_antenna

[24] Table 1. Common errors associated with GPS signals:
http://pubs.ext.vt.edu/442/442-503/442-503.html
[25] çerror por la posición satélites: http://www.alsitel.com/tecnico/gps/errores.html
[26] Errores GPS y eliminación: https://nagarvil.webs.upv.es/errores-atmosfericos-gnss-gps/
[27] Sistema de Navegación Inercial:
http://docplayer.es/7014016-Proyecto-de-grado-control-y-comportamiento-de-robots-omnidireccionales-posicionamiento-y-sensor-data-fusion.html
[28] Espectro Electromagnético:
http://www.las2orillas.co/la-gran-riqueza-intangible-del-espectro-electromagnetico/
[29] Ventajas de la visión artificial:
http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/3D/VisionArtificial/index.html
[30] Aplicación visión artificial: http://es.slideshare.net/eliponzoni/visin-artificial-original
[31] Aplicación visión artificial: http://sivartsl.com/descargas/artificial.pdf
[32] Características de OpenCV: http://opencv.org/documentation.html
[33] Componentes básicos de la realidad aumentada:
http://qode.pro/blog/que-es-la-realidad-aumentada/
[34] Aplicaciones de la realidad aumentada:
http://blogginzenith.zenithmedia.es/que-es-y-como-funciona-la-realidad-aumentada-diczionario/
[35] Concpeto niveles de la realidad aumentada:
https://es.wikipedia.org/wiki/Realidad_aumentada#Tecnolog.C3.ADa
[36] Realidad aumetada nievl 2: https://www.youtube.com/watch?v=U2jSzmvm_WA
[37] Definición niveles de la realidad aumentada:
http://www.nubemia.com/realidad-aumentada-en-la-educacion/
[38] Realidad aumentada nivel 3:
http://www.nubemia.com/realidad-aumentada-en-la-educacion/
[39] Documentación aruco_ros: http://www.uco.es/investiga/grupos/ava/node/26
[40] Transferir archivos con SCP por SSH:
http://www.desarrolloweb.com/articulos/transferir-archivos-scp-ssh.html

ANEXOS.

ANEXO I: INSTALACIÓN DE OPENCV

A pesar de que existen guías de instalación de OpenCV se ha considerado muy conveniente añadir al proyecto unas instrucciones para instalar OpenCV 2.4.9.

El primer paso es abrir un nuevo terminal y actualizar al completo el sistema:

- `sudo apt-get update`
- `sudo apt-get upgrade`

El siguiente paso es instalar las dependencias y librerías que requiere OpenV:

- `sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk libtbb-dev libeigen3-dev yasm libfaac-dev libopencore-amrnb-dev libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-extra libv4l-dev libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev default-jdk ant libvtk5-qt4-dev`

Se procede pues a descargar y descomprimir el código de OpenCV

- `cd ~`
- `wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.4.9/opencv-2.4.9.zip`
- `unzip opencv-2.4.9.zip`
- `cd opencv-2.4.9`

Ya se tiene el código descargado y descomprimido, se procede pues a instalarlo con Cmake. Para ello se creará un directorio "build". La herramienta CMake permite seleccionar las secciones de OpenCV que se desean compilar, generando un fichero Makefile. En este proyecto se han compilado las siguientes partes:

- `mkdir build`
- `cd build`
- ```
  cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON
  -D WITH_V4L=ON -D INSTALL_C_EXAMPLES=ON -D
  INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON
  -DWITH_QT=ON -D WITH_OPENGL=ON -D WITH_VTK=ON ..
  ```

Ahora se procede a compilar e instalar OpenCV 2.4.9

- `make`
- `sudo make install`

El paso final esconfigurar OpenCV, esto se hace modificando los ficheros pertinentes. El primero de ellos es opencv.conf, se puede abrir con el editor de texto que más guste al usuario. Se abre el fichero con gedit y se añade la siguiente línea (es normal que el fichero esté en blanco):

```
/usr/local/lib
```

Se ejecuta, en una terminal:

- `sudo ldconfig`

Para modificar el fichero bash.bashrc se añade al mismo:

```
sudo gedit /etc/bash.bashrc

PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

Con estos pasos ya se tiene instalado y configurado OpenCV. Ya se dispone de OpenCV 2.4.9 en el ordenador con compatibilidad 3D, Java, Python, OpenGL, Qt, etc.
```

ANEXO II: CÓDIGO DE LA APLICACIÓN.

Aquí se recoge la totalidad del código de la aplicación donde se calcula el posicionamiento del robot:

```
#include "ros/ros.h"
#include "geometry_msgs/PoseStamped.h"
#include "geometry_msgs/Pose.h"
#include <iostream>
#include <fstream>
#include "std_msgs/String.h"
#include <aruco_msgs/MarkerArray.h>
#include <std_msgs/UInt32MultiArray.h>
#include <std_msgs/Float64.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <tf/transform_datatypes.h>
ros::Publisher pub;
geometry_msgs::PoseWithCovariance absoluta;
float err;
float matriz[5][7];
void leer(void);
void leer(void){ //esta función lee la posición de los marcadores
FILE *f
=fopen("/home/blai/catkin_ws/src/aruco_ros/aruco_ros/src/posicion.tx
t", "r");
int i, j, x;
for(i=0;i<5;i++){
for(j=0;j<7;j++){
      x=fscanf(f, "%f", &matriz[i][j]);
printf("%f \n", matriz[i][j]);
}}}
void chatterCallback(const aruco_msgs::MarkerArray::Ptr msg)
{
float x2,y2,z2,x1,y1,z1,x, y, z,alfa, beta, gamma, absoluto;
int i;
i=msg->markers[0].id;
x=-msg->markers[0].pose.pose.position.x;
y=-msg->markers[0].pose.pose.position.y;
z=-msg->markers[0].pose.pose.position.z;
```

```
tf::Quaternion q(msg->markers[0].pose.pose.orientation.x,
msg->markers[0].pose.pose.orientation.y,
msg->markers[0].pose.pose.orientation.z,
msg->markers[0].pose.pose.orientation.w);
     tf::Matrix3x3 m(q);
     double roll, pitch, yaw;
     m.getRPY(roll, pitch, yaw);
alfa=pitch;
beta=roll;
gamma=yaw;
x1=(cos(beta))*(cos(gamma))*x-(cos(beta))*(sin(gamma))*y+(sin(beta))
*z;
y1=((cos(alfa))*(sin(gamma))+(sin(alfa))*(sin(beta))*(cos(gamma)))*x
+((cos(alfa))*(cos(gamma))-(sin(alfa))*(sin(beta))*(sin(gamma)))*y-(
(sin(alfa))*(cos(beta)))*z;
z1=((sin(alfa))*(sin(gamma))-(cos(alfa))*(sin(beta))*(cos(gamma)))*x
+((sin(alfa))*(cos(gamma))+(cos(alfa))*(sin(beta))*(sin(gamma)))*y+(
(cos(alfa))*(cos(beta)))*z;
alfa=matriz[i][4];
beta=matriz[i][5];
gamma=matriz[i][6];
x2=(cos(beta))*(cos(gamma))*x1-(cos(beta))*(sin(gamma))*y1+(sin(beta
))*z1;
y2=((cos(alfa))*(sin(gamma))+(sin(alfa))*(sin(beta))*(cos(gamma)))*x
1+((cos(alfa))*(cos(gamma))-(sin(alfa))*(sin(beta))*(sin(gamma)))*y1
-((sin(alfa))*(cos(beta)))*z1;
z2=((sin(alfa))*(sin(gamma))-(cos(alfa))*(sin(beta))*(cos(gamma)))*x
1+((sin(alfa))*(cos(gamma))+(cos(alfa))*(sin(beta))*(sin(gamma)))*y1
+((cos(alfa))*(cos(beta)))*z1;
err=sqrt(x1*x1+y1*y1+z1*z1)-sqrt(x*x+y*y+z*z);
absoluta.pose.position.x=x2+matriz[i][1];
absoluta.pose.position.y=y2+matriz[i][2];
absoluta.pose.position.z=z2+matriz[i][3];
absoluta.pose.orientation.x=err;}
int main(int argc, char **argv){
  ros::init(argc, argv, "buscador");
  ros::NodeHandle n;
  ros::Rate loop_rate(100);
  leer();
   ros::Subscriber sub = n.subscribe("/topo/markers", 1000,
chatterCallback);
```

```
  ros::Publisher
pub=n.advertise<geometry_msgs::PoseWithCovariance>("abs", 100);
  while (ros::ok())
      { absoluta.covariance[0]=10000;
      pub.publish(absoluta);
      ros::spinOnce();
      loop_rate.sleep();
      }

  return 0;}
```

PRESUPUESTO

1. Justificación del presupuesto.

A pesar de tratarse de un proyecto en el ámbito académico su naturaleza es el de un proyecto de ingeniería. Por lo tanto es necesario que la memoria del mismo contenga un presupuesto con todos los gastos que supone realizar el trabajo. Por eso se añadirá una cuantificación económica de todo el equipo y materiales empleados en el desarrollo del proyecto además del esfuerzo que ha supuesto.

El presente presupuesto seguirá las "RECOMENDACIONES EN LA ELABORACIÓN DE PRESUPUESTOS EN ACTIVIDADES DE I+D+I" del CTT (Centro de Apoyo a la Innovación, la Investigación y la Transferencia de Tecnología) de la Universidad Politécnica de Valencia. El documento citado servirá como modelo y se prescinde así del formato típico de un presupuesto con unidades de obra.

2. Estudio económico.

2.1. Costes de personal.

Para calcular el coste de la mano de obra se utilizará la siguiente expresión:

$$Coste\ (€) = Coste\ por\ hora\ (€/h)\ x\ Horas\ de\ trabajo\ (h)$$

En este cálculo del precio se incluyen los honorarios, los costes indirectos y la seguridad social e indemnizaciones. Como coste de hora se han seguido los datos de tarifas recomendadas del ejercicio de 2014 del anexo 2 del documento arriba citado para graduados en GITI.

Se observa que los honorarios de un titulado medio como personal eventual (se supone un trabajo de 1760 horas anuales, lo que supone 40 horas semanales) oscila entre 26.8 y 36.2 €/hora. Esto incluye gastos de Seguridad Social (32.1%), indemnizaciones (3,04%) y costes indirectos (16.5€/h).

Se ha escogido un valor medio de ese intervalo: 31.5 €/hora

Se procede a calcular el costo total:

Tabla 5.

Tarea	Horas	Coste(€)/Hora	Costo parcial (€)
Búsqueda de información, documentación y realización de tutoriales de ROS.	35	31.5	1102.5
Instalación del software (Ubuntu, ROS, aruco_ros)	25		787.5
Configuración y estudio de las librerías.	50		1575
Desarrollo aplicación de posicionamiento.	80		2520
Instalación de la aplicación en el Summit XL.	60		1890
Redacción de los documentos.	50		1575
Total	300		9450

Este precio total se puede desglosar para ver la cuantía destinada a los gastos de la Seguridad Social, Indemnizaciones y Costos Indirectos.

Tabla 6.

Concepto	Coste (€)
Honorarios Ingeniero.	3329.88
Seguridad Social.	1068.89
Indemnizaciones.	101.23
Costes Indirectos.	4950
Total.	9450

2.2 Material inventariable.

El CTT aporta la siguiente expresión para el cálculo del coste de amortización de todo el material empleado (hardware y software):

$$Coste\ (€) = Precio\ x\ \frac{Meses\ de\ Uso}{12\ meses/año}\ x\ \frac{1}{Periodo\ de\ amortización\ (año)}$$

Se establece un periodo de amortización de 6 años para el software y el equipo informático. El periodo de amortización para el material de investigación es de 10 años. El coste del material empleado es el siguiente:

Tabla 7.

Concepto	Precio (€)	Meses de uso	Periodo de amortización (años)	Unidades	Coste (€)
Ordenador portátil Lenovo	500	3	6	1	20.83
Ordenador portátil Asus	650	1	6	1	9.03
Logitech HD Pro Webcam C920	105	1	10	1	0.88
Robot SUMMIT XL	9500	1	10	1	79.17
Joystick Playstation 3	59.99	1	10	1	0.50
Ubuntu 14.04 LTS	0	4	6	2	0
Total					110.41

2.3. Material Fungible

Finalmente se calcula el costo del material fungible.

Tanla 8.

Concepto	Coste (€)
Marcadores (10 de diferente tamaño)	0.4
Folios (90)	2.7
Impresión y encuadernación	14
Total	17.1

3. Resumen del presupuesto

Tabla 9.

Concepto	Coste (€)
Mano de obra (300h graduado ing. tec. ind.)	9450
Material inventariable	110.41
Material fungible	17.1
Subtotal	9577.51
IVA (21%)	2011.28
Total	11588.79

El coste final del proyecto asciende a once mil quinientos cincuenta y ocho Euro con setenta y nueve céntimos.