UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica
I.T. Telecomunicación. Sonido e Imagen

Proyecto Fin de Carrera

"*UTM-30LX and URG-04LX laser range sensors characterization for autonomous robot navigation*"

Marcial Martín Román
Septiembre, 2010

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

I.T. Telecomunicación. Sonido e Imagen

Proyecto Fin de Carrera

"*UTM-30LX and URG-04LX laser range sensors characterization for autonomous robot navigation*"

Autor: Marcial Martín Román

Fdo.:

Director: Pedro M. Núñez Trujillo

Fdo.:

Tribunal Calificador:

Presidente:

Fdo:

Secretario:

Fdo.:

Vocal:

Fdo.:

CALIFICACIÓN:
FECHA:

Agradecimientos a mi familia,

a mi tutor Pedro Núñez por su apoyo e infinita paciencia

y en general a todo aquel al que le gustaría

ver su nombre aquí escrito

Índice de contenidos

Índice de figuras.

Resumen

En la actualidad existe una gran variedad de dispositivos que nos permiten obtener una visión mejorada de nuestro entorno.

El gran avance de la tecnología de los últimos años, ha permitido que dichos dispositivos optimicen su funcionamiento y tamaño de tal forma que están presente en multitud de facetas de nuestra vida diaria como por ejemplo: sensores de presencia para automatización de iluminación, sensores de distancia implementados en vehículos para asistencia en el aparcamiento, etc.

Una de estas aplicaciones del uso de sensores para obtener información del entorno se encuentra enmarcada dentro del área de la robótica autónoma móvil. Es en ella donde vamos a centrar el interés del presente proyecto. De forma similar a los seres vivos, los sensores facilitan la información necesaria para que los robots interpreten el mundo real. Todo robot debe tener al menos un sensor con el que interactuar. La mayoría de los sistemas robóticos incluyen al menos sensores de obstáculos (bumpers) y algún sensor de guiado por infrarrojos o ultrasonidos. Los sensores avanzados, además de detectar algo, son capaces de reportar una medida de lo detectado, como puede ser un sensor de temperatura, o un medidor de distancias ultrasónico por ejemplo.

De la caracterización matemática y estadística de varios de los dispositivos empleados en la configuración de un robot es de lo que versará este proyecto, en concreto de los dispositivos láser empleados en el mapeado y la localización para la navegación.

Capítulo 1: Introducción

Si atendemos a la definición de robot observamos que es una máquina capaz de interactuar con su entorno. Por lo tanto, a menos que se mueva en un entorno controlado y acotado para tal fin, el robot deberá ser capaz de adecuar sus acciones y movimientos de acuerdo con la información que reciba del medio. De esto se deduce la necesidad del uso de sensores de adquisición de datos del entorno.

Al igual que los seres humanos disponemos de un sistema sensorial que nos permite interactuar con el medio que nos rodea, los robots disponen de una serie de sensores de mayor o menor complejidad y sofistificación que les permita conocer el lugar en el que se encuentra, a qué condiciones físicas se enfrenta, la posición de los objetos con los que interactuar, etc.

Por consiguiente, la función del sensor consistirá en trasladar la información del mundo real al mundo abstracto de los microcontroladores.

Finalmente existen una serie de características que debemos tener en cuenta a la hora de elegir un sensor:

- Precisión: la precisión de la medición deber ser tan alta como sea posible, cometiendo el mínimo error entre el valor real y el medido.
- Rango de funcionamiento: se desea un rango de funcionamiento amplio y que el sensor sea preciso en todo el rango.
- Velocidad de respuesta: El transductor debe ser capaz de responder a los cambios de la variable detectada en un tiempo mínimo. Lo ideal sería una respuesta instantánea.
- Calibración: la calibración del sensor debe ser lo más sencilla posible siendo deseable que el tiempo y los procesos necesarios para llevar a cabo la calibración sean mínimos.

- Fiabilidad: el sensor no debe tener fallos frecuentes durante su uso.

1.1 Clasificación de sensores.

Se pueden hacer multitud de clasificaciones de los sensores atendiendo a diferentes parámetros. Hemos decido hacer una clasificación de aquellos sensores que debe montar un robot para captar su entorno en función de las magnitudes físicas que miden. Cabe citar que debido a la gran variedad de sensores existentes en la actualidad he seleccionado aquellos que considero más importantes.

Por lo tanto, la clasificación quedaría de la siguiente forma:

Tipos de sensores
- Sensores de visión
- Sensores de contacto y fuerza
- Sensores de sonido
- Sensores de posición
- Sensores de velocidad
- Sensores de rango

1.1.1 Sensores de visión.

Los sensores de visión nos permiten captar imágenes a través de una o varias cámaras de video. Esta información permite obtener una percepción del área de trabajo del robot desde el punto de vista del mismo.

Actualmente los sistemas de visión artificial, junto con el software de

tratamiento y reconocimiento de imágenes permite el diseño de robots capaces de localizar y reconocer objetos distintos y así, en función de su misión, tomar decisiones y llevar a cabo diferentes tareas.

Hoy en día tanto en robótica como en dispositivos de uso domestico ya no se utilizan cámaras de video con los antiguos sistemas de captación de imagen, las cámaras actuales contienen casi todas un CCD en sus sistemas de captación de imagen.

Un CCD es un circuito integrado cuya característica principal es que posee una matriz de celdas con sensibilidad a la luz alineadas en una disposición que permite "empaquetar" en una superficie pequeña un enorme número de elementos sensibles y manejar esa gran cantidad de información de imagen de una manera relativamente sencilla. Estas características permiten la creación de dispositivos cada vez más pequeños y con una gran calidad de imagen.

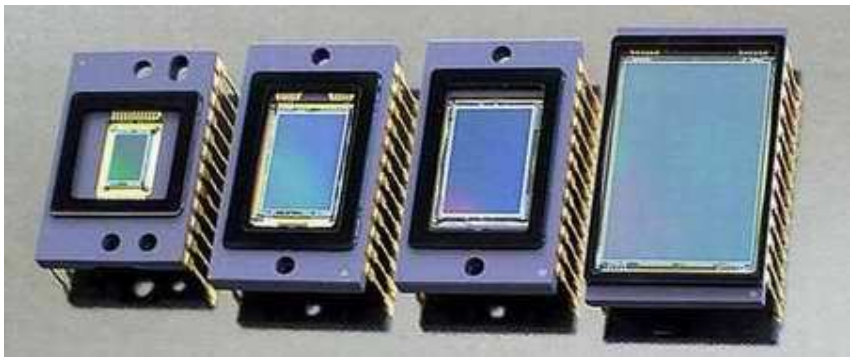

Figura 1: Sensores CCDs de diferentes tamaños.

1.1.2 Sensores de contacto y fuerza.

Los sensores de contacto nos indican simplemente si ha habido contacto o choque con algún objeto, mientras que los sensores de fuerza determinan además de si ha habido contacto con un objeto, la magnitud de la fuerza de dicho contacto. Estos sensores son de gran utilidad para robots que requieren la manipulación de objetos.

Un ejemplo de sensor de fuerza sería el siguiente sensor FlexiForce[1]:

Figura 2: Sensor de fuerza FlexiForce.

Como podemos observas, se trata de un sensor integrado dentro de una membrana de circuito impreso flexible de escaso espesor. El sensor es totalmente plano, lo cual permite colocarlo con facilidad entre dos piezas de la mecánica de nuestro sistema y medir la fuerza que se aplica sin perturbar la dinámica de las pruebas.

1.1.3 Sensores de sonido

Los sensores de sonido pueden tener dos aplicaciones dentro de la robótica: primero, dentro de un sistema de medición de distancia, en el que el micrófono recibe sonidos emitidos desde el mismo robot, luego de que éstos rebotan en los obstáculos que tiene enfrente, es decir, un sistema de sonar; y segundo, un micrófono para captar el sonido ambiente y utilizarlo en algún sentido, como por ejemplo, recibir órdenes de voz.

Para el uso como sonar, la implementación más básica sería colocar un micrófono en el interior de una cavidad direccional (bocina) de modo que haga pantalla a todos los sonidos salvo en una dirección definida. Si el robot detecta un sonido (con otro sensor o con el mismo) hace girar la bocina como un radar y busca la dirección del sonido por una simple

medición del máximo volumen. Es obvio que un sistema así está demasiado sujeto a errores.

Sin embargo se recomienda el empleo de los dispositivos actuales de detección y medición tipo sonar, tales como los medidores de distancia por ultrasonido.

Figura 3: Grupo de sensores de audio.

1.1.4 Sensores de posición.

Estos sensores nos permiten conocer todos los detalles acerca del posicionamiento del robot como su inclinación o la estimación de la posición del robot por el giro de sus rueda, también conocido como odometría.

Ejemplo de estos sensores pueden ser los pendulares o los sensores giróscopos.

Figura 4: Módulos y chip de giróscopo

1.1.5 Sensores de velocidad

La utilidad de estos sensores es la de procurar que la velocidad sea constante ya que una de las necesidades de un sistema propulsado por un motor de corriente continua es la de asegurar que la velocidad no varié, ya que en la mayoría de las aplicaciones de servomecanismos la velocidad de giro debe ser conocida y plausible de ser controlada desde un circuito de comando.

Para solucionar este problema se utiliza un dispositivo de medición cuya salida realimentada en el circuito de manejo de potencia controle las variaciones de velocidad.

A tal efecto se utiliza un tacómetro, que es un artefacto que debe generar una señal proporcional a la velocidad real del motor con la máxima precisión posible.

Figura 5: Motor de CC con tacómetro.

1.1.6 Sensores de rango.

Los sensores de rango son los dispositivos que permiten al robot detectar los posibles obstáculos del entorno en el que debe operar. También pueden ser usados para la medición de distancias. Existen muchos tipos de sensores de rango para la percepción del entorno, como son:

- Sensores basados en ondas acústicas: SONAR
- Sensores basados en señales electromagnéticas: infrarrojos,

RADAR.

- Sensores basados en luz coherente: laser rangefinder.

Este último tipo de sensor de rango basado en laser es sobre el que nos centraremos en este proyecto.

El principio de funcionamiento de estos dispositivos para medir distancias más común es el denominado *principio de tiempo de vuelo pulsional*, que consiste en la medida del tiempo que necesita el un pulso para ir y volver del objetivo. Dado que la electrónica necesaria para medir tiempos del orden de picosegundos (10^{-12}) es muy cara, se han desarrollado otras tecnologías como la detección de desplazamiento de fase o desplazamiento de frecuencia, que emplea la medición de la diferencia de frecuencia entre la onda emitida y la recibida (efecto Doppler) o por triangulación que se basa en relaciones trigonométricas entre ondas emitidas y recibidas.

Figura 6: Sensores de rango.

1.2 Aplicaciones de sensores de rango en robótica.

El uso de sensores de rango en robótica tiene una serie de ventajas. En primer lugar, un láser (ya sea de dos dimensiones, como los que se han usado en el presente trabajo, o tridimensional) tiene como característica una alta resolución, acompañado también de una alta precisión. Ello permite utilizarlos de una forma fiable en la navegación de robots; por ejemplo, en entornos dinámicos, es decir, con presencia de seres humanos u otros robots, o en tareas más complejas como la construcción de mapas (*mapping*) o en la localización

del robot en el entorno.

El proceso para la realización de mapas (mapping) consiste construir un array de distancias del entorno del robot, es decir, para cada ángulo el sensor mide la distancia que existe entre él y el obstáculo (pared, objeto), esa distancia es grabada en un array y con esa información podemos construir una representación del entorno.

Normalmente los sensores láser poseen una ventaja frente a los otros tipo de sensores de rango (infrarrojo) y es que posee un ángulo de detección muy amplio (del orden de 270º) por lo que no es necesaria incluir un dispositivo de rotación (servo) como ocurre en otros casos.

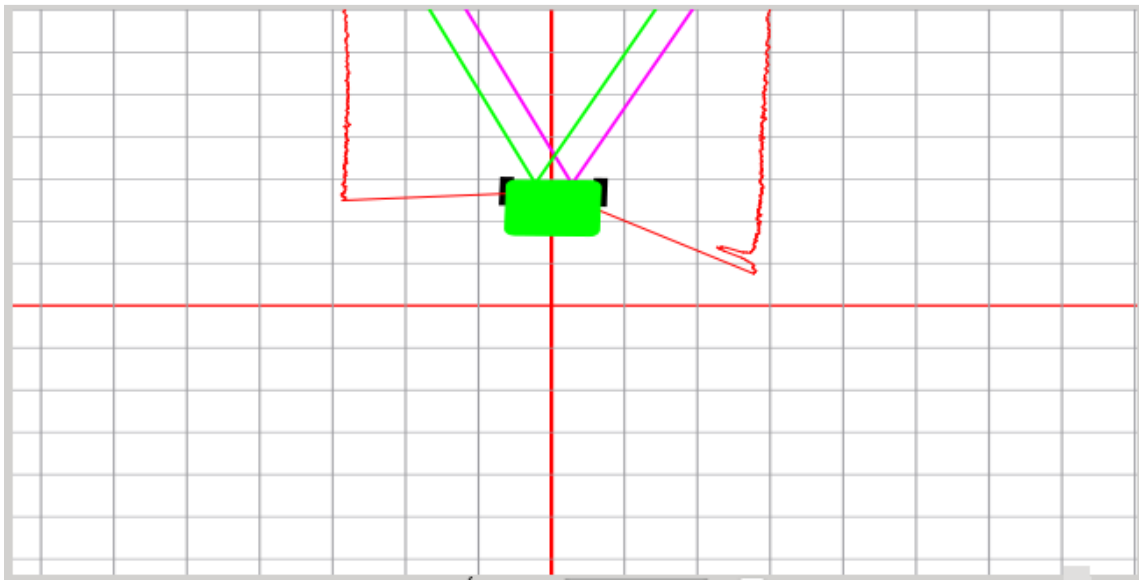

Figura 7: Representación gráfica de la información obtenida del láser.

1.3 Objetivos del proyecto.

Ciertos robots son controlados o se valen de sensores para su funcionamiento. Existe una amplia variedad de dispositivos diseñados para percibir la información externa de una magnitud física y transformarla en un valor electrónico que sea posible introducir al circuito de control, de modo que el robot sea capaz de cuantificarla y reaccionar en consecuencia.

Un robot puede llevar instalados sensores de luz, tales como cámaras de video, CCD, etc., sensores de gravedad (posición) como acelerómetros o giroscopios, y sensores de distancia y posición, como los interferómetros láser.

El objetivo fundamental del presente Proyecto Fin de Carrera será la caracterización matemática y estadística de dos sensores láser que son: URG-04LX y UTM-30LX.

El problema al que me enfrento es que en la hoja de características de dichos láser solo proporcionan una estimación pobre de la precisión del láser. En algunas aplicaciones, como la robótica, donde el agente autónomo se mueve libremente por el entorno, es interesante conocer la precisión de la medida en función a diferentes factores: luminosidad, tipo de superficie, distancia al objetivo, tiempo de funcionamiento del sistema. Las hojas de características ofrecen su información basándose únicamente en parámetros fijos: una determinada distancia o superficie objetivo. Por lo tanto, este proyecto se centra en analizar problemas como el efecto de deriva con el tiempo, variación con la distancia, desviaciones en medidas con diferentes superficies y ángulos de incidencia.

Para ello, se hicieron diferentes medidas, variando las superficies, distancias, ángulos de incidencia, y tiempo de funcionamiento de los sensores láser con el objetivo de obtener los diferentes estadísticos a utilizar por el robot.

Un objetivo secundario del proyecto ha sido la creación de una interfaz gráfica que permita al usuario realizar él mismo dichos experimentos, independientemente del modelo de láser. Dicha interfaz permite elegir el ángulo al que se quiere realizar la caracterización así como el nº de muestras o tiempo de ejecución del experimento. El programa devolverá datos como la

media, desviación estándar y varianza de las medidas, así como sendas gráficas en las que se representan las distancias medidas en tiempo real y la varianza.

Citar que para la consecución de dichos objetivos he tenido que desarrollar los siguientes subjetivos:

1.- Familiarizarse con el lenguaje de programación C++ y el sistema operativo Ubuntu (Linux).

2.- Aprender a utilizar los entornos de desarrollo Kdevelop y Qtcreator.

3.- Comprender el funcionamiento y arquitectura de Robex y Robocomp.

4.- Documentación matemática y estadística para discernir el análisis matemático óptimo.

1.4 Adaptación al entorno del laboratorio.

Como plataforma de experimentación se va a disponer de un robot de la clase RobEx, que se explicará en detalle en el capítulo siguiente.

1.4.1 Plataforma robótica utilizada

Podemos avanzar que RobEx, robot del tipo *differential three wheeled*, es un robot móvil de tipo diferencial, es decir, el movimiento del robot depende únicamente de la velocidad de rotación de sus dos ruedas motrices separadas que se colocan a ambos lados del cuerpo del robot.

Por lo tanto, puede cambiar su dirección mediante la variación de la tasa de rotación de sus ruedas, sin requerir un movimiento de dirección adicional. Además de las ruedas motrices, el robot dispone de una tercera rueda, de giro libre, que sirve de apoyo.

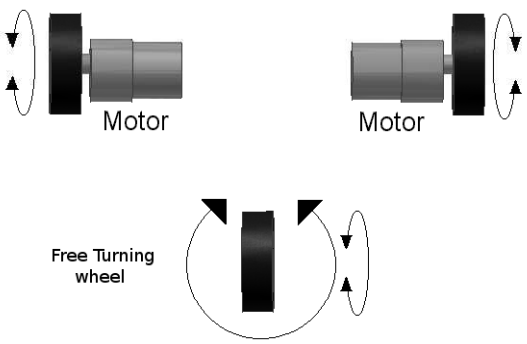

Figura 8: Mecánica de motricidad de RobEx

RobEx está diseñado para llevar uno o varios ordenadores portátiles a bordo para poder realizar los procesos complejos que se requieran. Su principal orientación es la investigación, aunque muy pronto veremos a estos robots realizando tareas en entornos diversos, como puede ser un museo, un instituto o un hospital.

A continuación se muestra una imagen de uno de los robots utilizados en nuestro proyecto y que cuenta con todas las características anteriormente mencionadas, aparte de otras características adicionales que se detallarán en el siguiente capítulo:

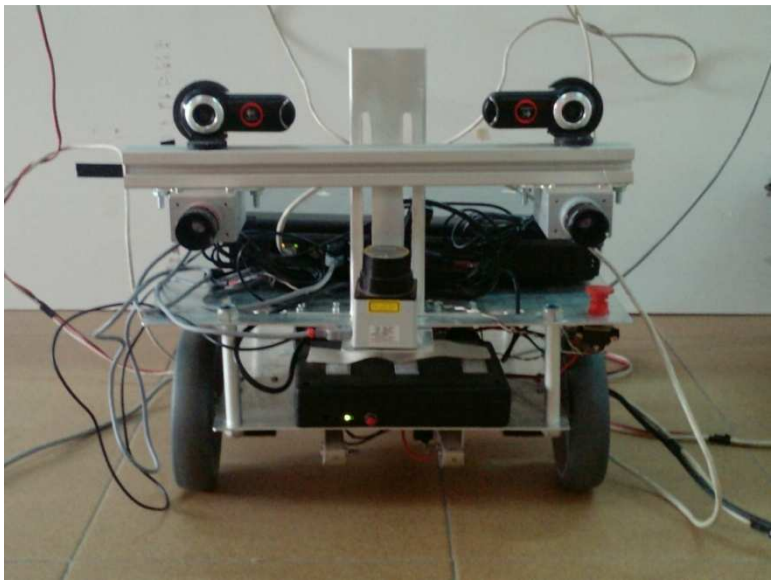

Figura 9: Robot móvil diferencial *RobEx*.

1.4.2 Entorno de trabajo

El entorno por el que va a moverse el robot está ubicado concretamente en la sala del Laboratorio de Robótica de la Universidad de Extremadura (Robolab).

Es un espacio constituido principalmente por paredes poligonales y en el que se han introducido diferentes objetos que actúan como obstáculos y para dar una sensación de ser un entorno de la vida real.

Es en esta sala donde se han realizado los experimentos y a continuación se muestra una fotografía de la misma:

Figura 10: Entorno de trabajo

1.5 Organización de la memoria.

La estructura de este documento se divide en una serie de capítulos bien diferenciados, para facilitar el entendimiento y seguimiento del trabajo realizado a lo largo del proyecto. A continuación se muestra una pequeña descripción de cada uno de ellos.

La memoria del presente Proyecto Fin de Carrera va a ser desarrollada concretamente en siete capítulos y un anexo.

Capitulo 1.- Introducción: en este capítulo se pretende introducir al lector en el mundo de la robótica en general y de los sensores empleados para obtener información del entorno en particular. Se ofrece una clasificación de los sensores más importantes empleados actualmente en robótica así como sus aplicaciones. Posteriormente se comentan los objetivos del proyecto y se describe el entorno en el que se llevaran a cabo los experimentos. Es el capitulo en el que el lector se encuentra en este momento.

Capítulo 2.- Sensores de rango empleados en robótica: en este apartado se realiza una descripción del principio de funcionamiento de estos sensores para posteriormente ofrecer los detalles técnicos de los sensores láser usados así como otros de distinto fabricante.

Capítulo 3.- Robex y Robocomp: en este apartado se va a proporcionar una toma de contacto con el robot sobre el que se realizarán las pruebas y con la arquitectura de componentes, estructura a seguir en la implementación de la solución.

Capítulo 4.- Caracterización de sensores láser: en este capítulo se detallan las expresiones matemáticas empleadas para llevar a cabo la caracterización de los sensores láser, así como los efectos que se pretenden estudiar y las condiciones bajo las cuales se realizan los experimentos.

Capítulo 5.- Diseño e implementación del sistema: en este capítulo se detalla la estructura de componentes que se propone como solución al sistema, se comentan los componentes preexistentes a utilizar y se establecen las funcionalidades del sistema basándonos en el nuevo componente creado y en su implementación mediante programación en C++. También se muestra el diseño del interfaz gráfico mediante la herramienta Qt4 designer.

Capítulo 6.- Experimentos y Resultados: en este capítulo se comentan los principales experimentos llevados a cabo junto con los resultados obtenidos y se muestran visualizaciones del desarrollo de los mismos.

Capitulo 7.- Conclusiones: dentro de este último apartado se hace una valoración global del trabajo realizado y se comparan los resultados obtenidos de los diferentes sensores.

Para finalizar la memoria se incluyen los apartados de anexo y bibliografía en los cuales se puede consultar el grueso del código desarrollado para el proyecto, características técnicas de los sensores, así como la relación de referencias bibliográficas consultadas en la realización de este proyecto.

Capítulo 2: Sensores de rango empleados en robótica

Tal y como hemos comentado en la introducción, uno de los sensores empleados en robótica para la percepción del entorno son los denominados sensores de rango. Existen sensores de rango basados en diferentes tecnología, como por ejemplo: basados en ondas electromagnéticas (RADAR) o en pulsos acústicos (SONAR). Sin embargo para este proyecto nos centraremos en los sensores basados en láser también conocidos como *Laser Range Finder* [2].

Su principio de funcionamiento es similar al sonar pero con ondas electromagnéticas y luz coherente, que es posible gracias a la producción de pulsos muy breves que pueden permanecer colimados a lo largo de distancias considerables. La precisión dependerá de la distancia, de la divergencia del haz y de la eficiencia de potencia.

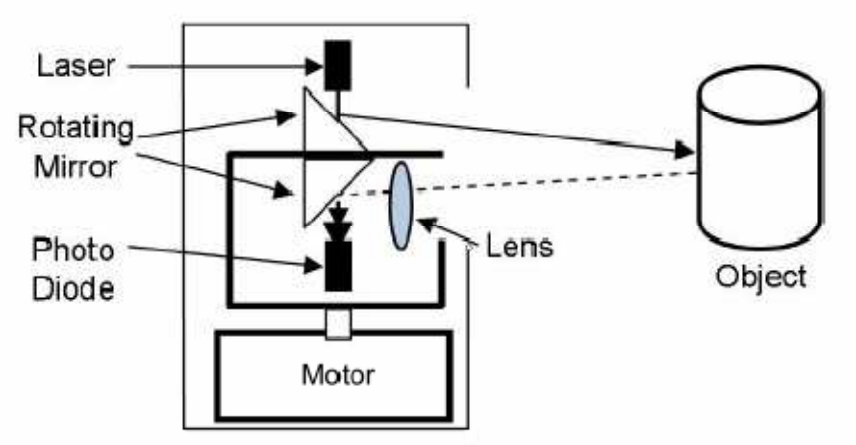

Figura 11: Esquema funcionamiento Laser Range Finder

Estos sensores poseen diferentes procedimientos para la estimación de la distancia según modelo. Dos de ellos son los siguientes:

- Procedimiento Phase-shifted: basado en la detección de desplazamiento de fase. Bajo ciertas condiciones (f= 5MHz, λ= 60m) produce cierta ambigüedad cada 60m, es decir, un obstáculo a 5m producirá la misma estimación de distancia que a 65 m.

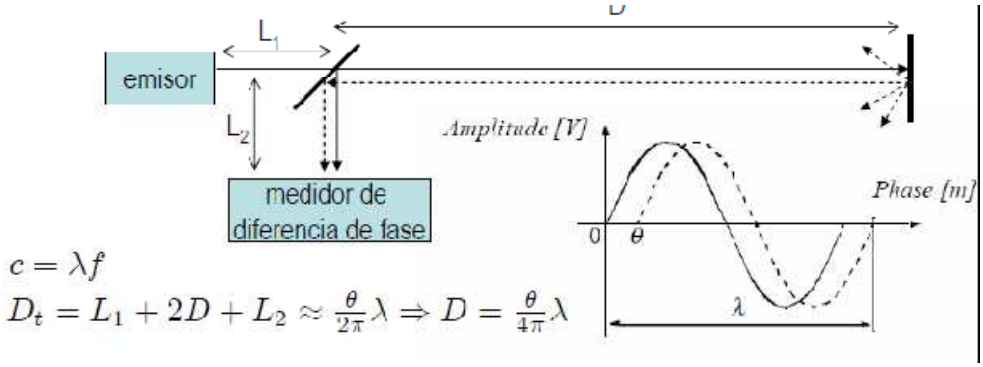

Figura 12: Esquema funcionamiento phase-shift.

- Procedimiento por triangulación: se basa en la utilización de de relaciones geométricas entre la onda emitida y recibida. La distancia medible dependerá del tamaño y resolución de la cámara lineal.

En el siguiente esquema podemos apreciar sus principios de funcionamiento:

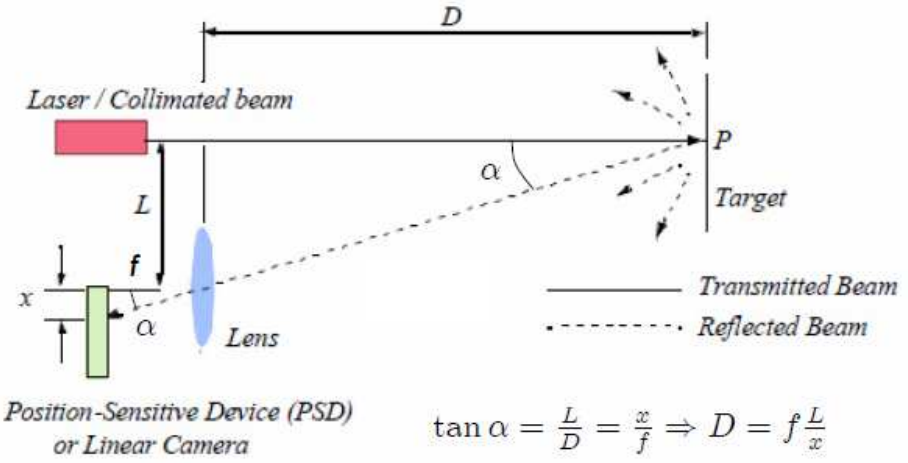

Figura 13: Esquema procedimiento por triangulación.

Citar que el componente desarrollado para este proyecto puede trabajar con cualquier sensor láser, sin embargo en el laboratorio solo se están disponibles los siguientes sensores, de los cuales se proporciona una información detalla de sus características.

La hoja de característica de cada uno de estos sensores así como sus datos más relevantes pueden encontrarse en el anexo I de esta memoria.

- **Láser Hokuyo URG-04LX:**

  Este láser se cataloga dentro de la categoría de *Scanning Láser Range Finder* y se trata de un sensor optimizado para el reconocimiento de entornos. Dadas sus características, hacen a este sensor ser el adecuado para su uso en robots autónomos de nueva generación o en sistemas de seguridad [4].

  - **Características:**
    - Alta precisión, elevada resolución y buena amplitud angular que proporcionan la mejor solución para robots autónomos móviles en entornos desconocidos.
    - Su compacto tamaño permite una mayor libertad de diseño. Un peso ligero y un consumo reducido de energía le permiten realizar operaciones de larga duración.
    - No le influye cuan brillante sea el entorno. Posee un excelente funcionamiento en la oscuridad.
    - Reconocimiento del tamaño y la posición de un cuerpo humano sin incurrir en su privacidad.

Los posibles usos de este sensor son numerosos, podemos destacar: reconocimiento de entorno para robots autónomos, detección de intrusos en edificios, detección de obstáculos para AGV, etc.

El láser es el siguiente:

Figura 14: Sensor láser Hokuyo URG-04LX.

- **<u>Láser Hokuyo UTM-30LX.</u>**

  Este láser también es un *Scanning Láser Range Finder* [5] y dadas sus características es adecuado para robots inteligentes con alta velocidad de movimiento por su amplio rango y su rapidez de respuesta. El sensor se muestra en la siguiente imagen:

Figura 15: Sensor láser Hokuyo UTM-30LX

- **Características:**
  - Adecuado para realizar medidas en exteriores.
  - Gran rango de detección, hasta 30 m.
  - Amplio ángulo de visión (270º), con una resolución de 0,25º.
  - Compacto y ligero.
  - 12 v de tensión de trabajo, con interfaz USB 2.0 de alta velocidad.

Otros sensores láser de rango empleados en tareas en el área de la robótica, como el mapping o la localización son los sensores de la casa SICK. Las características de los modelos más comúnmente usados son los siguientes:

- **SICK LMS200.**

  Perteneciente a la familia de los laser Range Finder [6], este sensor ofrece las características adecuadas tanto de funcionamiento como de dimensiones par su empleo en el campo de la robótica. El sensor se muestra en la siguiente figura:

Figura 16: Sensor láser SICK LMS200.

- **Características:**
    - No requiere objetivos con propiedades reflectivas específicas.
    - Los datos de medida están disponibles en tiempo real.
    - Iluminación del objetivo innecesaria.
    - Instalación y uso simples.
    - Admite cualquier posición de montaje.

Este sensor tiene múltiples usos como pueden ser: navegación, medición de distancias, monitorización de un área,…

- **<u>SICK LMS211.</u>**

  Sensor de la misma familia que el anterior podemos catalogarlo como de rango medio. Su amplia variedad de versiones disponibles ofrece una solución a medida tanto para tareas en exterior como interior. El sensor es el siguiente:

Figura 17: Sensor SICK LMS211.

- **Característica:**
    - Adecuado para la realización de medidas en el exterior.
    - Rango de operación de 0-80 m.

- Gran resistencia a factores ambientales.
- No requiere de reflectores.
- Datos de medición en tiempo real.

Dadas sus características es adecuado para la medición de objetos y distancias, monitorización de áreas en el ámbito de seguridad y para la realización de mapeados.

Capítulo 3: Robex y Robocomp

El entorno en el que se desarrolla el proyecto queda caracterizado por el robot sobre el que será integrado (familia RobEx) y por el sistema de componentes (RoboComp). El robot en el que se llevarán a cabo las pruebas es un robot de la serie RobEx. Estos son robots móviles con conectividad wireless y/o Ethernet, disponen de un ordenador de a bordo en el que se ejecutan algunos componentes y desde el que pueden interactuar con otros componentes ejecutados de forma remota. En general las tareas del ordenador de a bordo se centran en el control del hardware del robot: movimiento de la base y captura de las señales de los dispositivos incorporados, láser, cámaras, micrófonos, etc.

El sistema de componentes llamado RoboComp puede definirse como un grafo de componentes o procesos que pueden ser distribuidos en varios procesadores proporcionando un sistema de comunicación que permite el intercambio de información entre sí mediante una síntesis muy simple.

El proyecto consistirá en la creación de un nuevo componente a partir de uno base con las características anteriormente descritas. Para ello se aprovechan todas las funcionalidades ya cubiertas por los componentes existentes así como los conceptos y metodología de desarrollo. Por la razón fundamental de la reutilización en la programación orientada a componentes, estos están pensados con la esperanza de que en un futuro no muy lejano, otros los utilizarán para incorporar nuevas características al robot.

Tanto RobEx como RoboComp son proyectos desarrollados en el Laboratorio de Robótica de la Universidad de Extremadura. Son libres y se puede acceder a ellos mediante sus correspondientes páginas web [7] [8]. Dado que los resultados del proyecto de fin de carrera se han incluido dentro de RoboComp, el software desarrollado en él se distribuye bajo la misma licencia.

A lo largo de este capítulo se describirá el entorno de desarrollo y herramientas usadas, se hará una breve descripción del robot RobEx, se introducirá el paradigma de la programación orientada a componentes y, finalmente, se hará una breve descripción del repositorio de componentes RoboComp.

3.1 Robex

El robot RobEx [9] es una base robótica libre desarrollada en el Laboratorio de Robótica y Visión Artificial de la UEx, Robolab [10]. Es de tipo diferencial, esto es, el movimiento del robot depende únicamente de la velocidad de rotación de sus dos ruedas motrices. Además de las ruedas motrices, el robot dispone de una tercera rueda, de giro libre, que sirve de apoyo.

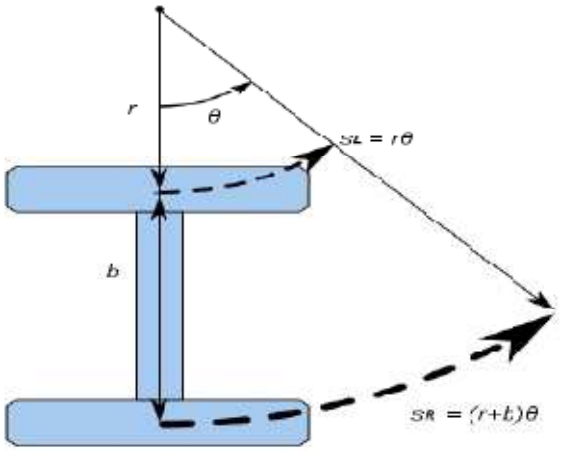

Figura 18: Dirección diferencial

La simplicidad de diseño y de los cálculos asociados al modelo diferencial son las principales razones que han definido el diseño, no sólo de RobEx, sino de otros muchos robots y gamas de ellos, como pueden ser: Segway, Kephera, o Roomba.

Sus planos se distribuyen bajo la licencia “Creative Commons Attribution-Share Alike 3.0”[11] . Por tanto, su diseño está abierto a cualquiera que lo quiera consultar o contribuir a él. Al ser de tipo diferencial su construcción es relativamente sencilla. Si se tienen conocimientos de electrónica, cualquiera puede construirla.

RobEx está diseñado para llevar uno o varios ordenadores portátiles a bordo para realizar los procesos complejos que sean necesarios, hasta hoy relacionados con la visión, la audición o la inteligencia artificial.

Tanto los motores de la base, de corriente continua, como el resto de la electrónica, se alimentan de una batería de polímero de litio como las que suelen usarse para extender la autonomía de los ordenadores portátiles. La base está controlada por un microcontrolador dedicado al que se accede a través de una interfaz RS232 sobre USB.

Su principal orientación es la investigación y la docencia. Aunque cada vez resultan más versátiles y pronto darán el salto a la realización de trabajos concretos en empresas. Llevan utilizándose más de tres años a diario en las asignaturas de Robótica y Teoría de Sistemas que se imparten en la carrera de Ingeniería en Informática de la UEX. El origen de su diseño y desarrollo hasta su estado actual nació de los diferentes proyectos de investigación realizados en Robolab a partir de su creación en el año 1999. Desde su primera versión, cada mejora y nueva funcionalidad que incorpora se prueba intensivamente tanto en el laboratorio como en las aulas, por lo que se consigue una robustez considerable.

Los objetivos de diseño de los robots RobEx son:

- Ser apropiado para entornos estructurados, pero que a la vez pueda ser modificado para otros tipos de terreno.

- Conseguir un robot de bajo coste y fácil de construir con capacidad de procesamiento intensivo a bordo.

- Que el precio no esté reñido con la calidad: fiabilidad y robustez.

- Poder ser ampliado con diversos accesorios de sensorización y manipulación, así como llevar varios portátiles a bordo.

- Servir de plataforma para formar a futuros investigadores.

Dar cabida a la experimentación utilizando hardware real en entornos controlados, una de las asignaturas pendientes en un plan de estudios tan teórico. Como se indicó antes, el robot RobEx es hardware libre. Todo el software desarrollado se distribuye bajo GPL. Con esto se pretende:

- Que cualquier persona tenga acceso al diseño y software del robot, y pueda fabricarlo por sí misma.

- Que la comunidad participe en la mejora y evolución del robot.

- Que cualquier empresa pueda usar o vender RobEx, pero que cualquier modificación hecha al robot o a su software se haga pública y mantenga la misma licencia.

3.1.1 Características técnicas

En la figura se muestra una representación del diseño de la parte mecánica de Robex.

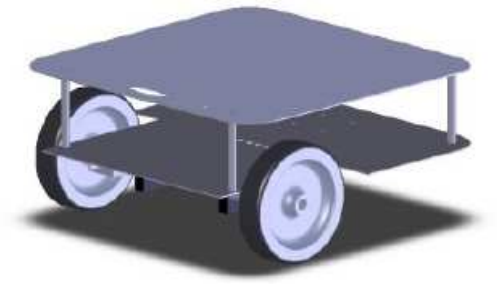

Figura 19: Representación del chasis de Robex.

La estructura del robot está formada por dos planchas de aluminio que se separan y se afianzan entre sí mediante cuatro tubos de acero. Para colocar los motores en el chasis se usan soportes de acero, unidos a estos motores se encuentran las dos ruedas motrices (las que aparecen en la figura), mientras que la rueda de giro libre va colocada en la parte posterior de la plataforma.

Para disponer de autonomía energética, el robot incorpora una batería recargable a bordo. Para ello se utiliza un batería de polímero de litio de 21,6V y 3500mAh. Estas baterías son de consumo doméstico común y se suelen utilizar para ampliar la autonomía de ordenadores portátiles, por lo que son asequibles y están sobradamente probadas. Estas baterías proveen al robot de una autonomía de algo menos de dos horas.

El sistema de control también se encarga de realizar las operaciones y cálculos necesarios para el control PID de los motores. El bucle de control PID de cada motor funciona a una frecuencia de 1 Khz. Por tanto, cada milisegundo se calcula la tensión de salida más apropiada para alcanzar el objetivo actual. En otros términos, el motor sólo está “sin control” o en lazo abierto por periodos de 1milisegundo.

La base dispone de un circuito integrado, LSI7266R1, que hace de contador de los pulsos procedentes de los dos codificadores ópticos. Éste se comunica con el microcontrolador mediante dos buses, uno de datos de 8 líneas de entrada y

salida y otro de control de 5 líneas de entrada. Esto permite obtener los datos necesarios para llevar las cuentas de la odometría.

3.1.2 Características adicionales

A esta forma “básica” de RobEx se han incorporado una serie de componentes adicionales que aumentan sus capacidades, como por ejemplo una torreta estéreo o un sensor láser . Además de estos, RobEx dispone de una serie de componentes adicionales que no se comentarán puesto que no han sido usados durante el desarrollo del proyecto. Como ejemplo podemos destacar un sistema de captura y digitalización de audio, un elevador de cargas o un sistema inercial de 5 gdl’s, 3 acelerómetros lineales y 2 giróscopos.

3.1.2.1 Torreta estéreo

La torreta estéreo está formada por una estructura en forma de U invertida con dos pies sobre los que se sitúan las cámaras, esta estructura está diseñada en aluminio lo que le confiere una gran solidez. Cada cámara está acoplada a su base a través de una sujeción ensamblada a un motor que proporciona un giro independiente sobre el eje vertical. En el interior de la U, en uno de sus laterales, se sitúa un tercer motor que hace girar toda la estructura, proporcionando un movimiento rotatorio simultáneo de ambas cámaras sobre el eje horizontal.

Este diseño admite, por lo tanto, 3 grados de libertad sobre la cabeza robótica: uno de elevación (tilt) común a las dos cámaras, y dos para el giro (pan) de cada cámara. No obstante, el rango de giro de cada cámara está limitado a causa del diseño mecánico para evitar el choque de éstas con la estructura, por lo que no todos los movimientos son posibles.

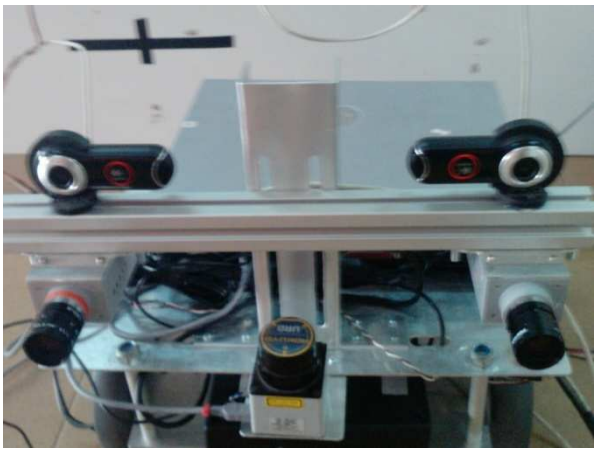

Figura 20: Torreta estéreo Robex

Los motores utilizados son servos Dynamixel RX-10 con microcontrolador incorporado y bus digital de comunicación convertible a USB mediante la que se conectan al ordenador de a bordo. Toda esta estructura es controlada por un componente “cammotionComp” diseñado específicamente para esta tarea. Este componente será comentado en posteriores capítulos.

3.2 Programación orientada a componentes

Dos de los principales problemas que se presentan cuando se crea software son la escalabilidad y la reusabilidad. Estos problemas son especialmente agudos cuando se trata de software que se va a emplear en robótica, pues la reutilización es algo excepcional en este campo. A pesar de la importancia de la reusabilidad, generalmente se suele perder de vista este aspecto y se acaba creando software monolítico y poco utilizable.

En el ámbito de la robótica es muy común que los investigadores implementen todos los algoritmos con un diseño rígido y orientado a una tarea y/o a un robot específico, en muchos casos debido a unos requerimientos de tiempo y funcionalidad muy específicos. De ser así, cuando finaliza la etapa de implementación el software desarrollado acaba siendo imposible de utilizar. Suele estar tan ligado a una plataforma o tarea específica que resulta más práctico

empezar de cero (debido a las dependencias y efectos colaterales derivados de su rigidez).

La programación orientada a componentes surge como solución a este tipo de problemas. Es un enfoque que no tiene necesariamente que ver con concurrencia o computación distribuida, sino con cómo se organiza el software. La programación orientada a objetos representó un gran avance respecto a la programación estructurada, sin embargo, cuando el número de clases y sus interdependencias crece, resulta demasiado difícil entender el sistema globalmente. Es por tanto beneficioso disponer de un grado mayor de encapsulamiento, que aúne diferentes clases relacionadas bajo una interfaz única, y permita comprender el sistema con menor grado de detalle. Muchos ven la programación orientada a componentes, que se propuso para solucionar este tipo de problemas, como el siguiente paso tras la programación orientada a objetos.

Un componente es un programa que provee una interfaz que otros programas o componentes pueden utilizar. Es una pieza de código elaborado o a elaborar que codifica una determinada funcionalidad. Son los elementos básicos en la construcción de las aplicaciones en esta arquitectura, que se juntan y combinan para llevar a cabo una tarea concreta. A su vez, estos programas hacen uso de programación orientada a objetos (así como en programación orientada a objetos se hace uso de programación estructurada). Esta división en piezas de software de mayor tamaño que las clases, implementadas como subprogramas independientes ayuda a mitigar los problemas de los que se ha hablado antes y a aislar errores.

Además, desde su carácter intrínsecamente distribuido ayudan a repartir la carga de cómputo entre núcleos, incluso, dependiendo de la tecnología usada, entre diferentes ordenadores en red.

Desde el punto de vista del diseño se pueden ver como una gran clase que ofrece métodos públicos. La única diferencia desde este punto de vista es que la complejidad introducida por las clases de las que depende el componente (o

clase) que no son del dominio del problema, desaparece porque la interfaz del componente las esconde. Un componente puede ser arbitrariamente complejo, pero un paso atrás, lo único que se ve es la interfaz que ofrece. Esto es lo que lo define como componente.

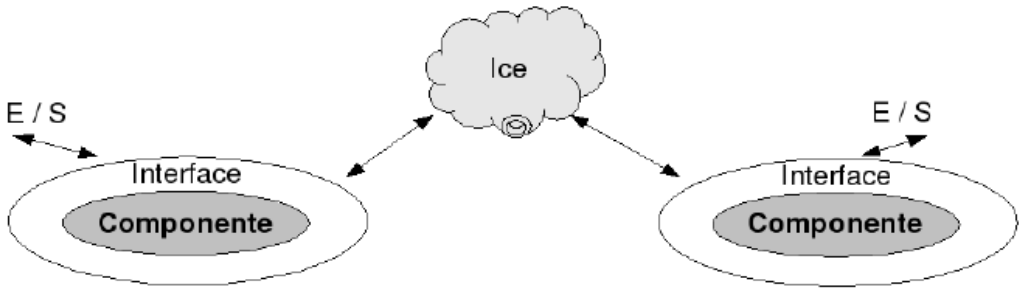

Figura 21: Grafo comunicación entre componentes.

Por tanto, si cada componente realiza una serie de tareas o responde a una serie de órdenes, necesitamos quien se encargue de esa comunicación. Es donde entra en juego Ice, pieza clave para la comprensión de la arquitectura y para la funcionalidad de la misma. Ice es un framework de comunicación muy interesante para su uso en robótica, sus características principales serán comentadas posteriormente. A pesar de estar desarrollado por una empresa privada es de fuentes abiertas y está licenciado bajo GPL. Además de su carácter libre hay otras dos razones para elegir Ice.

- La primera razón es su facilidad de uso: al contrario que otras tecnologías como Corba, la filosofía de Ice es soportar sólo aquellas características que los usuarios vayan realmente a necesitar y evitar introducir otras características que raramente se usan y dificultan el aprendizaje.

- La segunda razón es la eficiencia: si bien Ice codifica la comunicación eficientemente tanto en términos de tiempo como de espacio, otras alternativas suelen codificar la comunicación en XML. El uso de XML tiene ventajas en otras aplicaciones donde es conveniente que los humanos puedan entender el tráfico y no sean factores críticos ni la latencia ni la eficiencia, pero dentro de un robot esto no ocurre.

Ice soporta dos tipos de comunicación, por llamada remota (tipo RPC) o por suscripción (mediante un servicio que hace de servidor de mensajes). Sin embargo, este último introduce un paso intermedio cuya latencia hace desaconsejable su uso en robótica donde la ejecución en tiempo real es fundamental.

Para realizar una conexión con un componente lo único que se ha de conocer es su “endpoint”, es decir, la información necesaria para realizar la conexión: la dirección o nombre del ordenador al que se va conectar, el protocolo, el puerto y el nombre de la interfaz. Por ejemplo:

$$< \text{nombre} >:< \text{tcp/udp} > -\text{p puerto} - \text{h host}$$

donde ‘nombre’ es el nombre de la interfaz del componente al que se quiere hacer la conexión, ‘puerto’ es el puerto tcp o udp en el que el componente esté escuchando y ‘host’ el nombre del ordenador donde el componente se está ejecutando.

Desde el punto de vista del programador, una vez la conexión está hecha, el uso de componentes Ice es extremadamente simple. Cuando se realiza una conexión se crea una instancia de un proxy al componente en forma de objeto.

Al ejecutar un método del componente proxy el framework se encarga automáticamente de redirigir la llamada al componente remoto, por lo que la instancia del proxy se utiliza como si se tratase de una instancia de un objeto con la funcionalidad del componente.

Esta idea de usar un recurso remoto dentro de un programa no es nueva. Antes de la llegada de la programación orientada a objetos, ya había un protocolo en Unix llamado RPC para hacer llamadas remotas a procedimientos. Más tarde surgieron tecnologías como RMI, CORBA, DCOM, o Ice.

Para usar programación orientada a componentes se ha de dividir el diseño del software en piezas que ofrezcan una interfaz. A cambio obtendremos mayor reusabilidad, utilizando los mismos componentes en diferentes contextos, de esta forma se reduce considerablemente el tiempo, el coste y el esfuerzo de desarrollo de nuevas aplicaciones, aumentando a la vez la flexibilidad, reutilización y fiabilidad de las mismas. Será más fácil aislar y encontrar fallos, consiguiendo eliminar la necesidad de contemplar cientos de clases para comprender el software desarrollado.

3.3 RoboComp

RoboComp, es un repositorio de componentes basados en Ice con aplicaciones en robótica y visión artificial. RoboComp se comenzó a desarrollar en Robolab en 2005. Actualmente el proyecto ha sido migrado a SourceForge, donde, además de tener la página del proyecto (http://sf.net/projects/robocomp), dispone de un wiki (http://robocomp.wiki.sf.net/), donde hay documentación y un repositorio al que se puede acceder incluso directamente con un navegador web (https://robocomp.svn.sf.net/svnroot/robocomp).

Dispone de componentes para captura y visualización de vídeo, control del robot RobEx, detección y mantenimiento de regiones de interés (ROI), lectura y visualización de láser, lectura de joystick y navegación, entre otros muchos. Además dispone de un generador automático de componentes nuevos, de un programa gráfico de manipulación y control de componentes, managerComp y hasta de un componente encargado de grabar los datos producidos por los sensores para su posterior

reproducción, replayComp.

La figura muestra un grafo en el que se pueden ver los componentes de los que dispone RoboComp y las dependencias entre ellos.

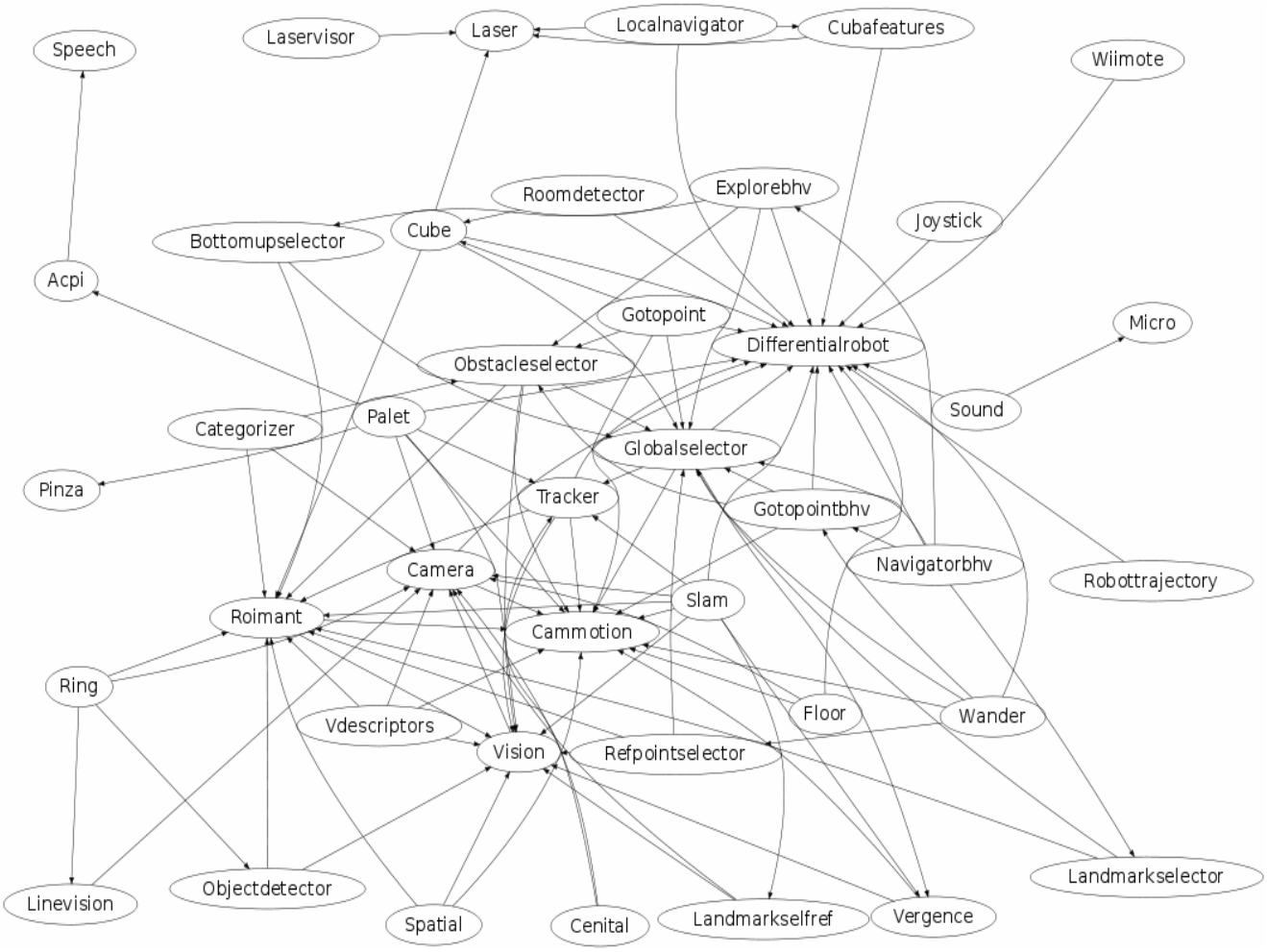

Figura 22: Grafo de componentes de Robocomp

Todos los componentes, así como clases auxiliares que se desarrollen a lo largo del proyecto se encontrarán inmersos dentro de este repositorio por lo que resulta esencial conocer su estructura.

3.4Entorno de desarrollo

A la luz de lo leído es fácil suponer que para desarrollar una aplicación basada en esta arquitectura, es necesario utilizar un conjunto de herramientas más o menos complejas y librerías que soporten las tareas exigidas a las capacidades del robot. La naturaleza de los problemas, eminentemente en tiempo real, a su vez exige que el tiempo de proceso sea mínimo y siempre se persigue, por filosofía del laboratorio y en la medida de lo posible, un mínimo consumo de recursos. En las siguientes páginas se describe el software utilizado.

3.4.1 IPP

IPP es una biblioteca empleada para procesar señales de una y dos dimensiones. La elección de esta biblioteca frente a otras de similar objeto se tomó por su gran eficiencia. Esta biblioteca es competencia directa, ya que IPP es desarrollada por Intel, La diferencia en el rendimiento respecto a sus competidores radica en que hace uso de las instrucciones SIMD que aportan las extensiones de x86, 3Dnow!, MMX y SSE y derivadas que Intel incluye en sus procesadores.

IPP es software privativo. A pesar de que es gratuito para el uso personal bajo GNU/Linux, hay que pagar la licencia si se desea usar comercialmente, y su código fuente no es público.

La eficiencia del software desarrollado, no sólo en este proyecto sino en el resto del software se debe en gran parte a la de estas bibliotecas. De ellas se han usado multitud de funciones en otros componentes relacionados con el tratamiento de imágenes. En este trabajo se ha utilizado para el análisis de la señal en el dominio de las frecuencias.

3.4.2 Qt4

Qt4 es un framework de desarrollo cuyo objeto principal es la creación de interfaces gráficas. A pesar de que sea este su principal uso, engloba una gran cantidad de funcionalidades distintas: interfaz multiplataforma con el sistema

operativo (sistema de ficheros, procesos, hilos entre otras cosas), comunicación por red mediante sockets, interfaz con OpenGL, conexiones SQL, renderizado de HTML, y módulos de reproducción y streaming multimedia. Atendiendo al lenguaje de programación, Qt está escrita en C++, pero existen multitud de bindings que hacen posible su uso desde otros lenguajes como Java, C#, Python, Perl y otros muchos.

3.4.3 Qt4 designer

Qt 4 designer es una herramienta de la empresa Trolltecha para diseñar y construir interfaces gráficos de usuario desde los componentes Qt. Permite diseñar y construir widgets y dialogs de una forma sencilla y eficaz.

Una característica esencial de este diseñador es que permite aprovechar las señales y slots de Qt lo que facilita la tarea de conexión del interfaz con el código interno de la aplicación.

3.4.4 Ice

Ice es el middleware del que se habló en la sección anterior que permite crear componentes software que pueden funcionar de forma distribuida sobre plataformas heterogéneas, de forma que puedan comunicarse entre sí y llevar a cabo un trabajo conjunto.

Es usado por RoboComp y, por tanto, por los componentes desarrollados en el proyecto de fin de carrera. Es el principal producto de la empresa estadounidense ZeroC, y se distribuye bajo una doble licencia GPL+ privativa para habilitar que las empresas desarrollen software privativo con Ice, a cambio del pago de una licencia.

Las principales características de Ice frente a otras tecnologías similares es su rapidez, baja latencia y escalabilidad.

Uno de los problemas a resolver a la hora de crear componentes software es la creación de un lenguaje de definición de interfaces. Ice usa Slice, un lenguaje que se creó específicamente para este propósito.

3.4.5 Cmake

Cmake es una aplicación que permite generar automáticamente ficheros Makefile y ficheros de proyecto de varios IDE como Kdevelop o Eclipse entre otros. Cmake permite delegar la creación de ficheros Makefile, consiguiendo un resultado multiplataforma y robusto, sin llegar a perder el control del proceso de compilación.

Cmake es una iniciativa libre que nació como respuesta a la ausencia de una alternativa suficientemente buena durante el desarrollo de una librería llamada ITK. Si bien dispone de soporte para Qt y algunas otras extensiones, es muy simple hacer nuevas extensiones.

3.4.6 Kdevelop

Es un entorno de desarrollo integrado para sistemas GNU/Linux y otros sistemas Unix, publicado bajo licencia GPL, orientado al uso bajo el entorno gráfico KDE, aunque también funciona con otros entornos, como Gnome.

El mismo nombre alude a su perfil: Kdevelop – KDE Development Environment (Entorno de Desarrollo para KDE). A diferencia de muchas otras interfaces de desarrollo, Kdevelop no cuenta con un compilador propio, por lo que depende de gcc para producir código binario.

3.4.7 managerComp

La aplicación managerComp permite visualizar, tanto gráficamente como en una lista, el estado de los componentes configurados en tiempo real. A pesar de estar integrado en RoboComp, se detalla independientemente por no ser un componente propiamente dicho. Además de la visualización del estado de los componentes, también permite “arrancarlos” y “pararlos”.

3.4.8 replayComp

replayComp es un componente de Robocomp que permite grabar y reproducir el uso de cualquier componente estable implementado para el robot. Es una herramienta muy útil para el desarrollo de programas que no necesiten emplear el robot en tiempo real, que ha sido mi caso en la realización de este proyecto.

Por lo tanto, replayComp permite grabar y reproducir los componentes Camera, Base, CamMotion y Micro interfaces.

La siguiente captura muestra la reproducción de un fichero grabado:

Figura 23: Captura replayComp.

3.4.9 OpenSSH

OpenSSH es una implementación libre del protocolo SSH (Secure Shell) que ofrece tanto la parte del servidor como la del cliente. Esta herramienta es importante en la puesta en marcha de proyecto porque, junto a managerComp, nos permite ejecutar rápida y remotamente los componentes que se deseen. Además, gracias a que permite autenticación basada en llaves, se puede trabajar de forma segura sin necesidad de escribir la contraseña cada vez que se quiere cambiar el estado de algún componente.

OpenSSH es una iniciativa de los desarrolladores de OpenBSD. Se distribuye con distintas licencias (dependiendo de la pieza), pero a pesar de esto, todas ellas son libres, GPL o BSD.

3.4.10 GNU/Linux

El sistema operativo usado durante el desarrollo y los experimentos, GNU/Linux. Gracias a él disponemos de un entorno de desarrollo gratuito y libre, herramientas de compilación, depuración, y una gran cantidad de bibliotecas complementarias. Se ha utilizado la distribución Kubuntu y Ubuntu pero por razones subjetivas.

Capitulo 4: Caracterización de sensores láser

En este capítulo nos centraremos en explicar los descriptores estadísticos empleadas para llevar a cabo el estudio así como los diferentes efectos tales como la deriva por tiempo así como la influencia de la luz ambiente o del color de la superficie empleada, que nos ayudaran a caracterizar los láseres.

4.1 Adquisición de los datos del sensor láser.

La información suministrada por el sensor láser en un escaneo es normalmente muy densa y tiene una precisión angular bastante buena. Las imágenes de rango suministradas por los sensores de rango tienen normalmente la forma $\{(\rho, \theta)_{l|l=1\ldots Nr}\}$, en la cual $(\rho, \theta)_l$ son las coordenadas polares de l-th de la lectura de rango (ρ_l es la medida de distancia de un obstáculo al eje de rotación del sensor en la dirección θ_l), y N_R es el número de lecturas de rango las cuales están relacionadas con el rango angular de la medida R y la resolución angular del láser $\Delta\theta$ por $N_R = R/\Delta\theta$. Las medidas escaneadas son adquiridas por el láser con una resolución angular $\Delta\theta = \theta_l - \theta_{l-1}$.

4.2 Descriptores estadísticos utilizados.

Para la caracterización de los sensores láser hemos utilizado expresiones matemáticas de carácter estadístico relativas a la distancia medida por el sensor con el fin de conocer la precisión y sus comportamientos ante diferentes condiciones experimentales para poder aportar una información más detallada y concisa a la aportada por el fabricante.

Las ecuaciones empleadas son:

4.2.1 **Media aritmética:** también conocida como promedio o simplemente media, consiste en la suma de un número finito de valores

divididos entre el número de sumandos. Una de las limitaciones de la es que se ve afectada por valores extremos; valores muy altos tienden a aumentarla mientras que valores muy bajos tienden a reducirla, lo que implica que puede dejar de ser representativa de la población. Debido a ello, hemos de apoyarnos en otras ecuaciones para llevar a cabo la caracterización. Su expresión es:

$$\bar{x} = \frac{\sum_{i=1}^{n} a_i}{n} = \frac{a_1 + \cdots + a_n}{n}$$

4.2.2 **Desviación estándar:** desviación típica (σ) es una medida de centralización o dispersión para variables de razón (ratio o cociente) y de intervalo, de gran utilidad en la estadística descriptiva. Se define como la raíz cuadrada de la varianza. Junto con este valor, la desviación típica es una medida (cuadrática) que informa de la media de distancias que tienen los datos respecto de su media aritmética, expresada en las mismas unidades que la variable. Para conocer con detalle un conjunto de datos, no basta con conocer las medidas de tendencia central, sino que necesitamos conocer también la desviación que representan los datos en su distribución respecto de la media aritmética de dicha distribución, con objeto de tener una visión de los mismos más acorde con la realidad a la hora de describirlos e interpretarlos para la toma de decisiones. Su ecuación es:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

4.2.3 **Varianza:** el término varianza hace referencia a la media de las desviaciones cuadráticas de una variable aleatoria, con relación al valor medio de esta**.** La varianza de una variable aleatoria, por lo tanto, es una medida de su dispersión. Se trata de la esperanza del cuadrado de la desviación de la variable frente su media y se mide en una unidad diferente. Por ejemplo: en los casos en que la variable mide una distancia en kilómetros, su varianza se expresa en kilómetros al cuadrado. Cabe destacar que las medidas de dispersión (también conocidas como medidas de variabilidad) se encargan de expresar la variabilidad de una distribución por medio de un número, en los casos en que las diferentes puntuaciones de la variable están muy alejadas de la media. A mayor valor de la medida de dispersión, mayor variabilidad. En cambio, a menor valor, mayor homogeneidad. Lo que hace la varianza es medir la variabilidad de la variable aleatoria. Es importante tener en cuenta que, en ciertos casos, es preferible utilizar otras medidas de dispersión ante las características de las distribuciones.

$$\sigma^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n}\left(x_i - \bar{x}\right)\right]$$

4.2.4 **Matriz de covarianza:** Cuando en un estudio se mide la relación bivariada entre más de dos variables, frecuentemente la información se expresa en forma matricial. En nuestro caso particular, debido a que trabajamos con datos obtenidos a través del scan realizado por un láser, hemos de suponer que se producirán errores debido a una gran resolución angular. Los puntos siendo procesados en coordenadas cartesianas son el resultado de una transformación no-lineal de puntos en coordenadas polares:

$$x_i = r_i + cos\theta_i \qquad\qquad y_i = r_i + sin\theta_i$$

Esto significa que los errores en las dos coordenadas cartesianas son correlados (Diosi and Kleeman). De hecho, si los errores en la distancia y orientación asumimos que son independientes con media cero y desviaciones estándar σ_r y σ_θ , respectivamente, entonces la matriz de covarianza asociada a la distancia medida i en coordenadas cartesianas puede ser aproximada por la expresión de primer orden de Taylor de la siguiente forma:

$$C_{xyi} = J_r \begin{pmatrix} \sigma_\theta^2 & 0 \\ 0 & \sigma_r^2 \end{pmatrix} J_r^T = \begin{pmatrix} \sigma_r^2 s^2 + r_i^2 \sigma_\theta^2 c^2 & \sigma_r^2 sc + r_i^2 \sigma_\theta^2 cs \\ \sigma_r^2 sc + r_i^2 \sigma_\theta^2 cs & \sigma_r^2 s^2 + r_i^2 \sigma_\theta^2 s^2 \end{pmatrix}$$

Donde J_r es el Jacobiano(consultar anexo III) de la transformación de coordenadas polares a Cartesianas, *c* y *s* son $\cos\theta_i$ y $\sin\theta_i$ respectivamente.

4.3 Efectos a estudiar.

En este apartado se presentan los diferentes efectos y experimentos que se llevarán a cabo con el fin de realizar una caracterización exhaustiva de los dispositivos láser. En este proyecto se analizarán el efecto deriva, la influencia del ángulo de incidencia y la distancia, el efecto de la luz ambiente y las dependencias del color, brillo y material de la superficie utilizada.

A continuación se explican en profundidad estos efectos:

4.3.1 **Efecto de deriva**: consiste en la variación de valores en una medida continua. Es una efecto muy importante a la hora de caracterizar los láser ya que nos da una idea de cómo se comportará nuestro láser en función del tiempo de trabajo. Suele ser causado por la disipación de calor del láser. La medida se llevará a cabo con el laser

situado a 1.5 m del objetivo y se tomará la medida de 0º. La medida tendrá una duración de 50 minutos.

4.3.2 **Influencia del ángulo de incidencia**: mediante este experimente se pretende comprobar la variación de los valores medidos a medida que vamos rotando el sensor. La medida se realizara situando el sensor a 1.5 m de objetivo y posteriormente se irán tomando desde 0º hasta el ángulo de incidencia máximo para el cual se produce captura. El incremento será de 20º y se adquirirán 1000 muestras. Para comparar la influencia de la superficie en la medida, este experimento se llevó a cabo en primer lugar sobre una pared de superficie lucida y en segundo lugar sobre una pared recubierta de baldosines de color blanco.

4.3.3 **Influencia de la distancia**: el experimento consistirá en variar la distancia que separa al sensor del objetivo. Los parámetros del experimento son: medida realizadas de 1 a 3 metros con incrementos de 1 m para el laser URG-04LX, con adquisición de 1000 muestras y sobre la pared lucida y de baldosines.

Para el sensor láser UTM-30LX debido a las condiciones del área de trabajo el rango de medición será de 1-8 metros.

4.3.4 **Influencia de la luz ambiente**: se pretende comprobar experimentalmente como influyen diferentes condiciones de luz ambiente en el funcionamiento del sensor con el fin de caracterizar la luz ambiente óptima de trabajo.

4.3.5 **Influencia del color, brillo y material de superficie utilizada**: para comprobar este efecto se llevaran a cabo una serie de experimentos en los que se variaran los parámetros de la superficie sobre la que actúa el láser.

Capítulo 5: Diseño e implementación del sistema

Este capítulo está enfocado a explicar de forma minuciosa el proceso de diseño y desarrollo de nuestro componente para la caracterización de sensores láseres. Además de explicar el nuevo componente también comentaré las principales características y funcionalidades de los componentes ya existente y que sirven de base a nuestro proyecto.

El sistema ha sido implementado en el lenguaje de programación de alto nivel C++ y en cuanto a la estructura del programa hemos utilizado programación orientada a componentes.

El resultado ha sido un programa con una estructura fácil de entender y modificar, consiguiendo así un programa nada monolítico y reutilizable.

Para obtener una visión global de como interactúa nuestro componente con los ya existentes podemos observar el siguiente gráfico:

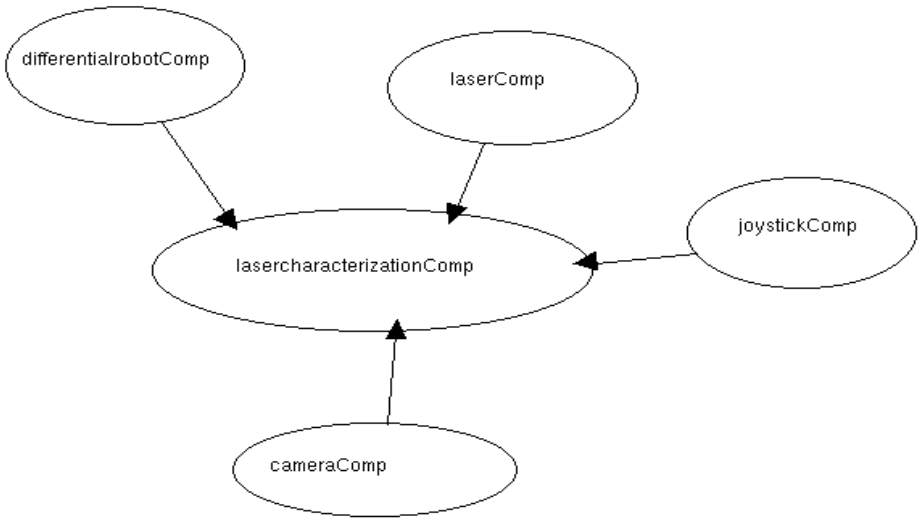

Figura 24: Grafo de interconexión de componentes.

5.1 Primeros pasos

Como se ha comentado previamente, el proyecto parte de una serie de componentes ya desarrollados, que interactuarán con nuestro componente para conformar el sistema que hemos desarrollado. En esta sección se van a comentar estos componentes y el componente genérico que nos ha servido de base para crear nuestro lasercharacterizationComp, este es el basicmonitorComp.

Todos ellos se encuentran en continua revisión y ampliación, puede dirigirse a http://robocomp.wiki.sourceforge.net" para obtener la última versión disponible o ampliar la información.

5.1.1 basicmonitorComp

El componente basicmonitorComp constituye la base sobre la que hemos construido nuestro lasercharacterizationComp. Es un componente muy básico que está preparado para obtener la captura instantánea del láser y de las cámaras. Nuestro trabajo ha consistido en añadir ciertas funciones adicionales e ir mejorando su interfaz gráfica para adecuarla a los propósitos de nuestro proyecto.

En la interfaz de este componente, se reserva un espacio en el que se representan unos ejes y que nos ha servido para crear nuestro "mundo", es decir el entorno que rodea al robot.

Todos los componentes del sistema van a trabajar con visión estereoscópica, que aunque para nuestro proyecto es meramente informativa, en la interfaz gráfica se nos muestra en tiempo real lo que está viendo el robot.

5.1.2 cameraComp

El componente cameraComp es otro de los primeros componentes de RoboComp. Su principal función es capturar vídeo y atender a las peticiones de imágenes que otros componentes realicen. Además, ha de adjuntar a las imágenes la configuración de la posición que tenía la torreta y el estado de la odometría cuando la imagen fue tomada. Tiene capacidad tanto para mandar imágenes de una cámara, como de dos a la vez. Para evitar problemas de sincronización, cuando trabaja con dos cámaras, puede devolver ambas imágenes simultáneamente en una única llamada. Esta característica es básica para trabajar con visión estereoscópica. Otra de las características interesantes de cameraComp es que permite trabajar con distintos tipos de cámaras: v4l2 (Video For Linux v2), IEEE 1394 (FireWire), o mediante una canalización Unix hacia Mplayer (lo que además de extender aun más el abanico de cámaras soportadas, permite la captura de imágenes desde ficheros de vídeo y recientemente desde el simulador de robots Gazebo).

5.1.3 joystickComp

El componente joystickComp es también un componente ya previamente creado que, aunque de relativa sencillez, es de vital importancia para el desarrollo de nuestro proyecto, ya que nos permite controlar los desplazamientos del robot a nuestro antojo. Es el que nos ha permitido recorrer las trayectorias que se han llevado a cabo en la realización de los experimentos.

5.1.4 laserComp

El componente laserComp nos permite establecer la conexión con el láser y que éste nos devuelva los datos referentes a las mediciones que realiza en cada barrido: la distancia a cada punto del scan y el ángulo correspondiente. Para la

autolocalización del robot, nuestro sistema de coordenadas nos muestra tres datos: las coordenadas del punto en el plano (X , Z) y el desvío angular, *Yaw*, con respeto a la posición inicial.

5.1.5 differentialrobotComp

El componente differentialrobotComp es el que controla la base robótica que, con una determinada velocidad de rotación de sus dos ruedas motrices, mueve al robot, dirigiéndolo con una tercera rueda de giro libre. Mediante las dependencias atribuidas en nuestro *.xml* hacemos que, al arrancar este componente, se arranquen también el laserComp y el cameraComp.

5.2 Creación de un componente genérico

Antes de proceder con la especificación de los componentes construidos se mostrará cómo crear un componente genérico. Cuando se incluye un nuevo componente en RoboComp se usa generalmente una herramienta que lo genera automáticamente. El generador de código también incluye el código necesario para crear una conexión a los componentes que se le especifiquen.

A los componentes que se generen automáticamente se le pueden añadir después todas las clases que sean necesarias. Además, muchas veces se desea añadir en el fichero de configuración algún parámetro específico y seguramente, modificar el fichero de interfaz por defecto (que no ofrece ningún método). De no usar el generador automático de código, la creación de componentes, incluso pequeños, sería bastante tediosa y susceptible a errores, ya que es necesario integrar el código del middleware Ice y el código propio del componente.

Por tanto, la creación del nuevo componente se llevará a cabo utilizando esta herramienta, siguiendo la estructura básica definida mediante esta creación.

Esta estructura es la que se observa en la figura siguiente:

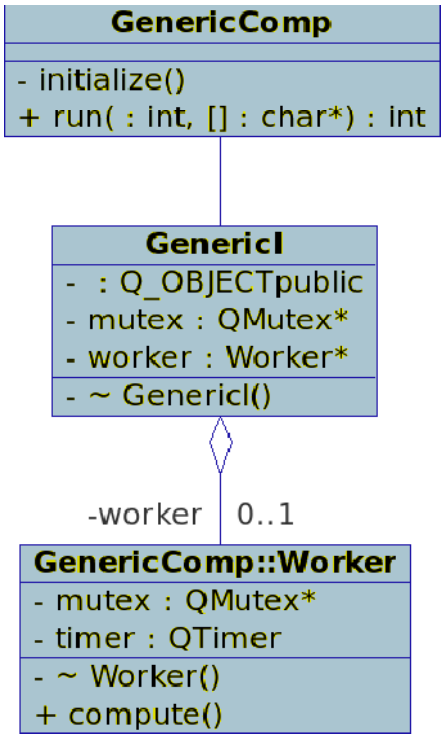

Figura 25: Diagrama de clases de un componente genérico.

Como se puede observar en la figura, además del código de programa principal, el componente plantilla consta de tres clases: GenericComp, GenericI y Worker[12].

Cuando se usa el generador de código, éste modifica los nombres de las clases GenericComp y GenericI en función del nombre del componente, cambiando *Generic* por lo que corresponda.

Cada componente consta de, al menos, dos hilos de ejecución, siendo estos el hilo principal y un hilo separado para responder a las llamadas Ice. El último de estos dos hilos se crea cuando se instancia una interfaz Ice (a la clase creada se le tiene que pasar como parámetro la clase que se hace cargo de las llamadas). El hilo principal de ejecución se encarga de la parte activa del componente, de hacer las llamadas Ice necesarias si las hay, y los cálculos asociados al comportamiento del componente.

Para facilitar la comprensión del código la clase GenericComp delega en Worker todo el trabajo y GenericI queda como una clase que, de forma transparente y asíncrona, responde a las llamadas remotas. Las clases Worker y GenericI, que como ya hemos visto se encargan de la parte interna y externa respectivamente, se ejecutan en exclusión mutua gracias a un mutex que comparten. El bloqueo del mutex se realiza cada vez que GenericI recibe una llamada o que Worker va a cambiar alguna parte de su estado que pueda modificar las respuestas de GenericI a sus invocaciones remotas. Como se puede ver en el diagrama, GenericI tiene acceso a la clase Worker. Dicha condición, que sólo se da en ese sentido, es necesaria porque generalmente toda la información relativa al estado del componente se guarda dentro de Worker.

5.2.1 Archivo *.xml*

Para poder arrancar los componentes y trabajar con ellos desde el managerComp de una forma cómoda e intuitiva, es necesario crear un archivo *.xml* que nos permita proceder desde la herramienta mangerComp y no como se hacía años atrás desde la terminal del P.C.

XML es un método para introducir datos estructurados en un fichero de texto. Cuando pensamos en “datos estructurados” pensamos en hojas de cálculo, parámetros de configuración, dibujos técnicos, etc. XML consiste en una serie de reglas o pautas para planificar formatos de texto para tales datos, de manera que produzcan archivos que sean fácilmente generados y leídos por un ordenador; evitando problemas comunes como la falta de soporte o la dependencia de una determinada plataforma.

Nuestro archivo .xml (figura 26) establecerá los nodos que vamos a utilizar, los puertos a los que han de conectarse éstos, las dependencias entre los distintos nodos y cómo arrancar y detener la ejecución de los mismos.

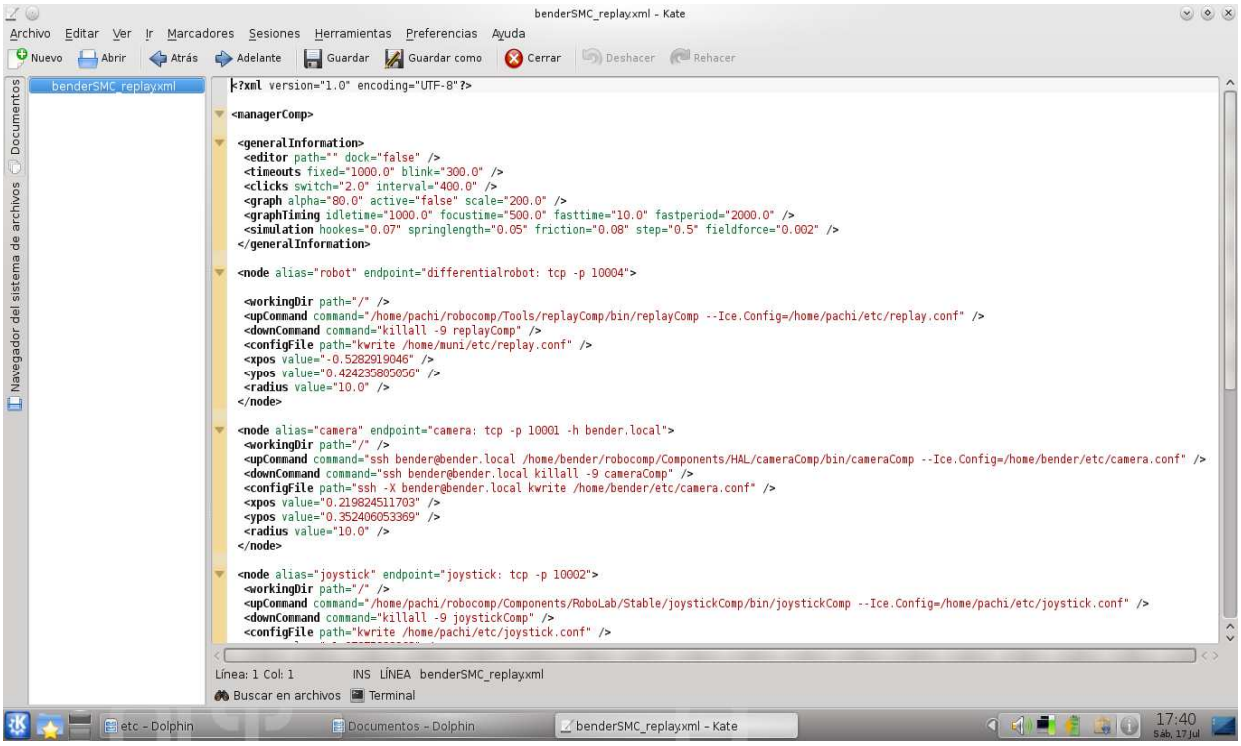

Figura 26: Captura del código del archivo .xml

5.3 Diseño del nuevo componente

Como ya hemos comentado anteriormente, para el desarrollo de nuestro componente, denominado lasercharacterizationComp, hemos partido de la base de un componente genérico incluido en el repositorio de robocomp, para posteriormente incluir en su código las modificaciones necesarias para nuestro proyecto.

A lo largo de este apartado comentaremos dichas modificaciones tanto en el código del programa como en su interfaz gráfica.

Reseñar antes de comenzar, que la práctica totalidad del código incluido en este capítulo se encuentra en dos archivos de nuestro componente que son: *worker.cpp* y *worker.h*

5.3.1 Interfaz gráfica

La interfaz gráfica del componente consiste en una ventana o cuadro que aparece cuando es ejecutado el componente y en la que se muestra la información y utilidades que incorpore el mismo.

Para el desarrollo de nuestra interfaz hemos empleado una herramienta denominada *Qt4 designer*. Qt es un toolkit de plataforma cruzada (cross-platform) desarrollado por Trolltech[13] (empresa Noruega) para construir GUIs en C++ y puede funcionar en varias plataformas: Microsoft Windows, Unix/X11, Mac OS X. Con él se han desarrollado aplicaciones tales como: Google Earth, Adobe Photoshop Elements, Skype y KDE.

Con *Qt Designer* se pueden crear fácilmente ventanas y colocar objetos (botones, barras, ect.) dentro de ellas de una forma simple y eficaz. Una de las ventajas del uso de Qt es que puedes ver cómo va quedando tu interfaz a medida que lo estas desarrollando, además permite enlazar los botones y cuadros del interfaz con el código del componente, tarea que simplifica enormemente el trabajo al programador.

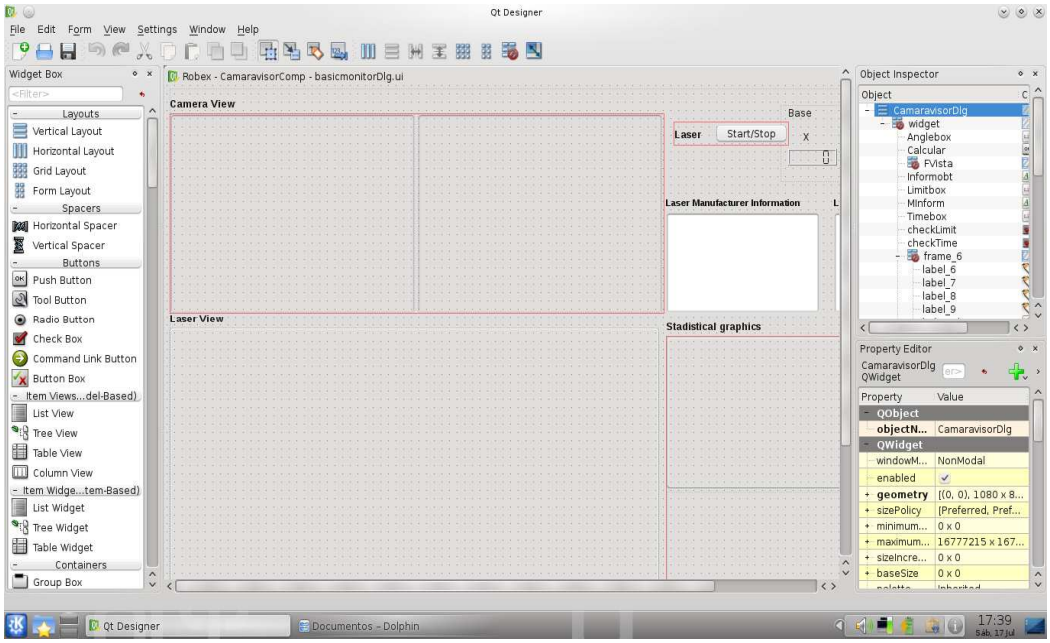

Figura 27: Captura Qt4 Desigener.

Los elementos gráficos de *Qt designer* se denominan *widgets* y la clase base de cada uno de estos elementos es la clase *Qwidget*. Esta clase tiene una serie de características y funciones, las cuales describen y modifican la apariencia, el comportamiento y la manera de diseñar el *widget.*

Cada vez que realicemos modificaciones sobre nuestro interfaz deberemos hacer por consola un *cmake.* Par que dichos cambios sean guardados en el código del componente.

Tras esta introducción pasemos a explicar con detalle la interfaz de nuestro componente. A la hora de diseñarla buscamos obtener una herramienta simple y eficaz que permitiera al usuario elegir las condiciones de medida, implementado de tal forma que no se pudieran introducir valores incompatibles.

A continuación se ofrece una captura de la apariencia final de nuestra interfaz trabajando el componente en modo replay.

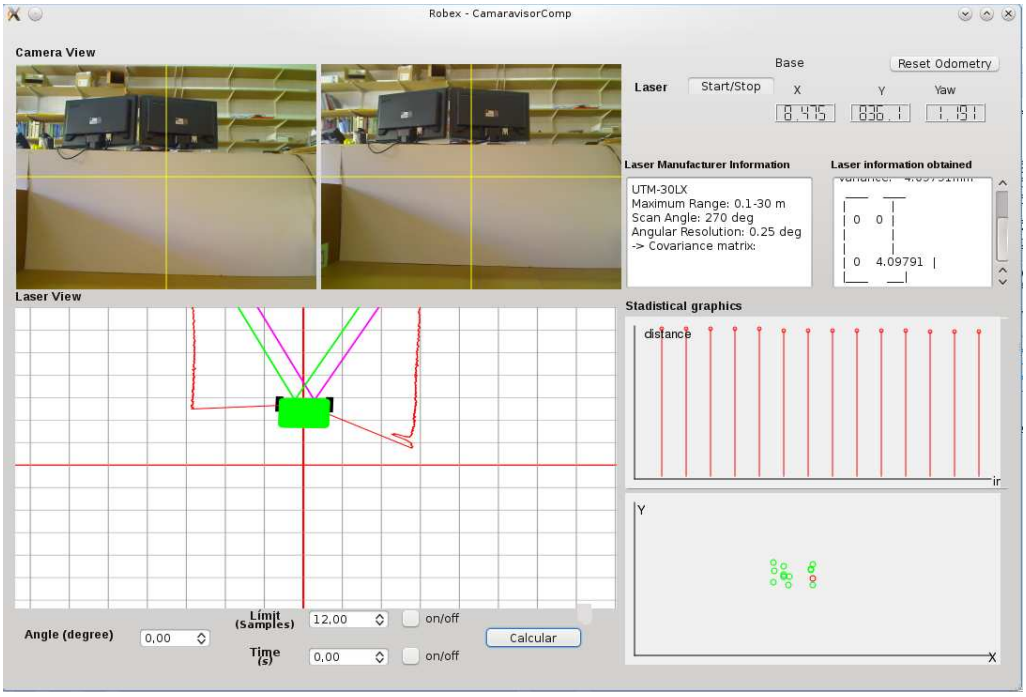

Figura 28: Captura interfaz funcionando en modo replay.

Tal y como podemos observar en la imagen, nuestro interfaz consta de los siguientes módulos:

- *Módulo de cámaras.*

Como ya hemos dicho anteriormente, queremos que nuestro sistema trabaje con visión estereoscópica, que aunque es meramente ilustrativa, nos parece interesante poder observar desde la interfaz gráfica lo que está viendo el robot en tiempo real. Para ello había que tomar los datos de las cámaras – izquierda y derecha por separado y reproducirlos en las ventanas destinadas a ello en la interfaz.

Esta parte se implementa en la función *getCamara*, como puede apreciarse en las siguientes líneas de código:

```
void Worker::getCamara(){

 try

        {

                _camara_prx->getYImage(0, imgV,  hState, bState);  //Cambiar en
                camaraComp

                memcpy(imgI->bits(), &imgV[0], camParams.size);

                _camara_prx->getYImage(1, imgV,  hState, bState);  //Cambiar en
                camaraComp

                memcpy(imgD->bits(), &imgV[0], camParams.size);

                //innerModel->updatePropioception( bState, hState );

                qvisorI->setImage(imgI);

                qvisorD->setImage(imgD);

                // qvisorI->drawCrossHair(Qt::yellow);

                // qvisorD->drawCrossHair(Qt::yellow);

        }

catch(const Ice::Exception& ex)

        {

                qDebug(" Error talking to CamaraComp requesting images... \n");

                textOut->append( QString("Error talking to CamaraComp requesting
                images..."));

                textOut->append( QString("Error talking to CamaraComp requesting
                images..."));

        }

}
```

Figura 29: Función GetCamara

El resultado es el siguiente:

Figura 30: Captura modulo de cámaras.

- *Módulo de mundo virtual*

Consiste en un espacio gráfico o frame denominado “mundo” en el que aparecen los movimientos del robot así como una representación grafica de lo que está captando el láser del entorno.

La implementación de este módulo se lleva a cabo en diferentes funciones de la siguiente forma:

En primer lugar debemos crear en la función principal un nuevo Qworld al que le asignaremos el frame que creamos con el Qt4 designer para tal fin.

```
world = new qWorld( FVista, rectBeta);

world->show();
```

A continuación deberemos implementar en la función *compute* unas líneas de código con el fin de que se dibuje de forma continuada el robot así como unos ejes que nos sirvan de referencia.

```
world->drawRobot( innerModel );
world->drawAxis(Qt::gray, 8);
world->update();
```

Como podemos comprobar en el código para dibujar el robot se emplea la clase *innerModel*, que proporciona un modelo del robot, que al estar en la función *compute* permanecerá continuamente actualizado.

Finalmente queda desarrollar el código que dibuje una representación de lo que está captando el láser. Para ello, trabajaremos en la función *getLaser* de la

siguiente forma:

```
static int pantx = bState.x;
static int pantz = bState.z;
        {
                laserData = _laser_prx->getLaserData ();
                for (unsigned int i=0; i<laserData.size(); i++)
                        {
                           QMat p = innerModel->laserToWorld( laserData[i].dist ,
                                                   laserData[i].angle );
                           world->drawLine(QLine(pantx, pantz, p(0), p(2)), Qt::red);
                           pantx = p(0);
                           pantz = p(2);
                        }
                world->drawLine(QLine(pantx, pantz, bState.x, bState.z), Qt::red);
                pantx = bState.x;
                pantz = bState.z;
        }
```

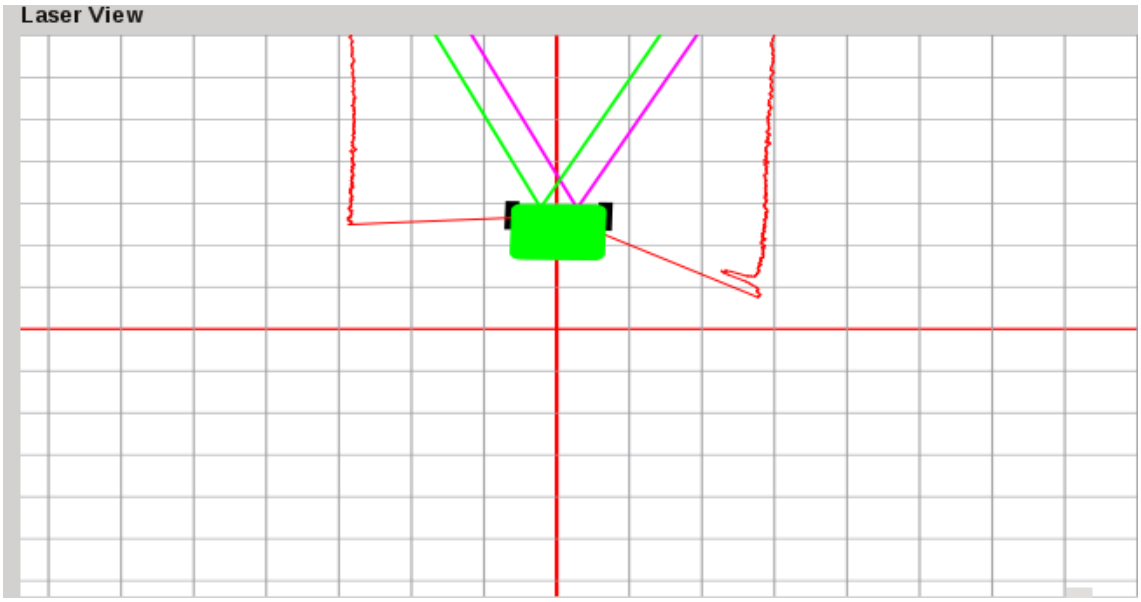

Figura 31: Captura módulo de mundo virtual.

- *Módulo de información.*

Este módulo fue el más sencillo de implementar, ya que consiste en dos ventanas que muestran en primer lugar, la información de utilidad proporcionada por el fabricante, y en segundo lugar los datos calculados por nuestro componente, es decir, valor medio, varianza, …

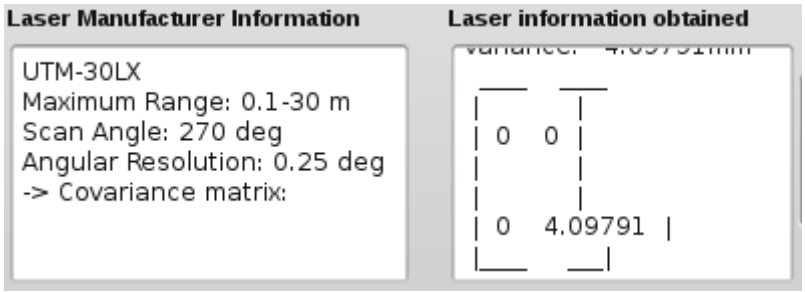

Figura 32: Captura módulo de información.

- *Módulo de odometría.*

Aunque para el desarrollo de nuestro proyecto no sea de utilidad conocer la localización exacta del robot, hemos implementado una función que nos muestre

la posición en cada instante del robot en el espacio bidimensional (X y Z), así como una desviación angular como dato de refuerzo (Yaw).

Estos datos son obtenidos gracias a que la base dispone de un circuito integrado, LSI7266R1, que hace de contador de los pulsos procedentes de los dos codificadores ópticos.

Por otro lado también hemos implementado la capacidad de resetear la odometría a elección del usuario y así poder tomar como punto inicial, coordenadas (0,0,0), el punto elegido. Para ello se ha creado un botón que al pulsarlo llama a la función *resetOdometer*. Todo esto corresponde a las siguientes líneas de código.

```
void Worker : : compute ( )
{
        try {
                _base_prx->getBaseState(bState);
                innerModel->updatePropioception( bState, hState );
                 lcdNumber->display(bState.x);
                  lcdNumber_2->display(bState.z);
                 lcdNumber_3->display(bState.alpha*180/M_PI);
}
void Worker::on_BReset_clicked()
{
        try
        {
                _base_prx->resetOdometer();
        }
}
```

Figura 33: Método inicializar y resetear odometría.

El resultado sería el siguiente:

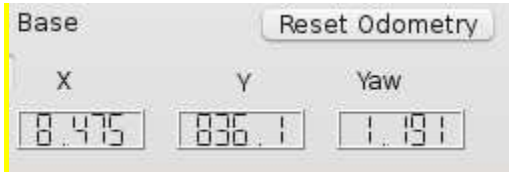

Figura 34: Captura módulo odometría.

- *Captura del láser.*

  Parte fundamental de nuestro proyecto ya que todo el trabajo se desarrolla sobre los datos obtenidos por el láser. Para la obtención de la información del láser, deberemos seguir una serie de pasos que comentaremos a continuación.

  - Buttom de conexión del láser:

  Mediante este botón podemos ordenar la conexión y desconexión del láser. Por lo tanto, deberá pulsarse cada vez que queramos obtener los datos estadísticos.

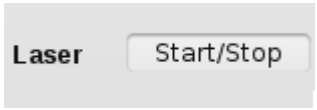

Figura 35: Buttom conexión/desconexión láser.

Su implementación es la siguiente:

```
void Worker::compute( )
{
if (pbLaser->isChecked()) getLaser();
}
```

Figura 36: Implementación botón de conexión del láser

Con esto conseguiremos que al pulsar el buttom se llamara a la función getlaser que provee la información del sensor.

- *Submódulo de selección de parámetros.*

La creación de este submódulo se llevó a cabo con el fin de evitar cierta rigidez en el componente puesto que permite al usuario elegir los parámetros de sus mediciones, tales como: ofrecer la posibilidad de determinar la duración en segundos del experimento o introducir el número de muestras a estudiar, así como el ángulo sobre el cual se llevará a cabo el experimento.

La implementación se realizó a través de bucles if anidados de tal forma que aparezca en pantalla una ventana de error si se hace una selección de parámetros incorrecta.

Citar la dificultad de obtener una función que devolviera un valor real del tiempo de ejecución del programa. Finalmente conseguimos obtener resultados aceptables empleando la clase Qtimer.

En el siguiente fragmento de código se expone lo arriba explicado:

```
timer = new QTimer();
connect(timer,SIGNAL(timeout()),this,SLOT(compute()));
timer->start(1000/camParams.FPS);
```

Figura 37: Inicialización tiempo de ejecución de programa

```
void Worker::compute( )
{
        if (Calcular->isChecked()) {
         // Comment: Tomamos el ángulo introducido.
         angle = deg2rad(Angle box->value());
            if ((checkLimit->isChecked()) && (checkTime->isChecked())){
                mostrarMensajeInformacion();
                seguir = false;
               Calcular->setChecked(false);
               checkTime->setChecked(false);
               checkLimit->setChecked(false);
               }
        else if (((checkLimit->isChecked())||(checkTime->isChecked()))&&(Calcular-
        >isChecked())&&!(pbLaser->isChecked())){
                mostrarMensajeInformacion();
                seguir = false;
                Calcular->setChecked(false);
                checkTime->setChecked(false);
               checkLimit->setChecked(false);
                }
```

Figura 38: Método selección de parámetros

```
else if (checkLimit->isChecked()){
            Limite = Limitbox->value();
             seguir = true;
             }
 else if (checkTime->isChecked()){
               time = Timebox->value()*1000;
               if (!seguir) init = QTime::currentTime();
               seguir = true;
                 }
             }
if (seguir)   {
             CalcularDatos();
  }
}
```

Figura 39: Continuación código anterior

```
void Worker::mostrarMensajeInformacion()
{
          QMessageBox::StandardButton respuesta;
          respuesta = QMessageBox::information(this, trUtf8(""), trUtf8("Error, invalid action."));

}
```

Figura 40: Ventana de información para acción inválida

El resultado final es el siguiente:

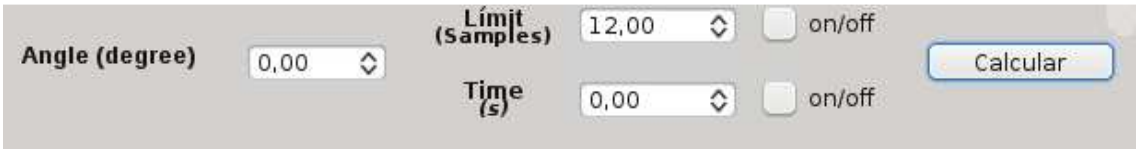

Figura 41: Módulo selección de parámetros.

- *Obtención de los datos.*

Pese a que sitúo este apartado de obtención de los datos en último lugar, fue la primera fase que desarrollé debido a que es la parte central de este proyecto. En este apartado se explicará la implementación de código cuyo objetivo es caracterizar estadísticamente el láser a estudio.

La ejecución del programa consta de tres fases que se ven reflejadas en el desarrollo del código. Estas fases son en primer lugar, el almacenamiento de los datos obtenidos del láser, en segundo lugar el proceso matemático para la obtención de los parámetro estadísticos y por último, sendas representaciones gráficas de dichos parámetros.

Vayamos fase por fase comentando el proceso de implementación:

- *Fase 1.*

Como ya hemos comentado anteriormente, en esta fase se desarrolló una función que obtenga ciertos datos del láser, en concreto, el ángulo de medida y la distancia. Para obtener esta información utilizamos la función *laserprx* que permite la comunicación con el láser.

Una vez obtenidos los datos, el siguiente paso fue guardar en vectores tanto las distancias medidas por separado como la suma de todas ellas con el fin de utilizarlos en cálculos posteriores.

Con el objetivo de ahorrar memoria, empleamos vectores dinámicos, cuyas dimensiones se van actualizando a medida que se le van introduciendo datos.

Finalmente quedaría implementar mediante una serie de bucles if el almacenamiento de los datos en función de si el límite es de muestras o de tiempo.

Dada la extensión del código no ha sido incluido como en etapas anteriores. Puede consultarse en el apartado anexo función StoreData página 100.

- *Fase 2*

El objetivo de esta fase es la de implementar las funciones matemáticas de valor medio, desviación estándar y varianza que nos permitirán conocer las características del láser a estudio de una forma más exhaustiva.
El código correspondiente puede verse en el anexo funciones CalcularDesviación y cacularDatos en páginas 101 y 102.

- *Fase 3*

En esta última fase el trabajo se centró en desarrollar dos representaciones gráficas con el fin de obtener un resultado gráfico de aquello que se está calculando. La primera de ellas consiste en una representación continua de las distancias medidas con el fin de apreciar su variación con el tiempo, y en la segunda gráfica podemos observar la deviación en la medida a través de la representación de cada punto medido en coordenadas cartesianas.

La ejecución de esta fase conllevó ciertas dificultades debido a que no existía ninguna herramienta adecuada de dibujo dentro de las clases qt.
Lo solucionamos valiéndonos de las funciones drawLine y drawEllipse y empleando como lienzo una ventana qWorld.

El resultado es el siguiente:

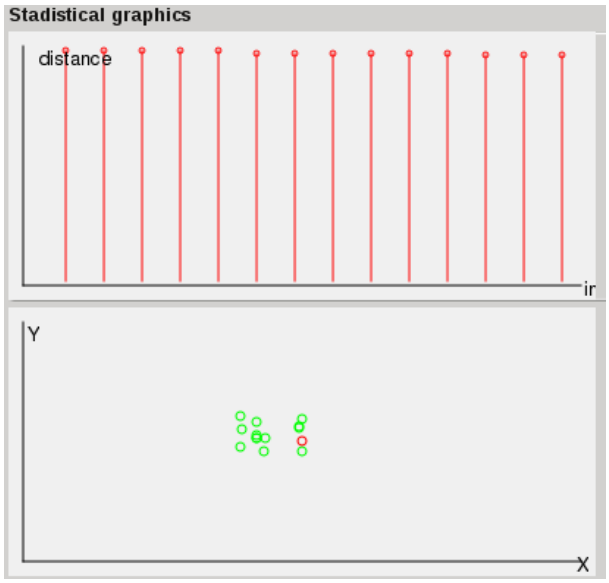

Figura 42: Módulo de gráficas del interfaz.

Puede consultarse su implementación en el anexo II

Capítulo 6: Experimentos y resultados

En este capítulo describiremos los experimentos llevados a cabo para obtener una caracterización más detallada de los sensores lásers disponibles en el laboratorio. Presentaremos los resultados estadísticos para cada sensor así como sendas gráficas mostrando los resultados obtenidos en los experimentos.

Con la realización de estos experimentos también se ha llevado a cabo una comprobación exhaustiva de la robustez y estabilidad del sistema desarrollado para este proyecto.

Antes de comenzar a explicar con detalle los experimentos y resultados me gustaría describir la herramienta que no permite conectar todos los componentes para la gestión del robot, este es managerComp.

6.1 managerComp

La aplicación managerComp permite visualizar, tanto gráficamente como en una lista, el estado de los componentes configurados en tiempo real. A pesar de estar integrado en RoboComp, se detalla independientemente por no ser un componente propiamente dicho. Además de la visualización del estado de los componentes, también permite encender y apagarlos quedando el nodo en verde si está funcionando y en rojo si no. Es una aplicación muy útil cuando es necesario usar gran cantidad de componentes ya que te permite manejarlos en una sola ventana, mientras que de otra forma necesitaríamos un ventana de línea Al igual que el resto de herramientas de RoboComp, se distribuye bajo licencia GPL.

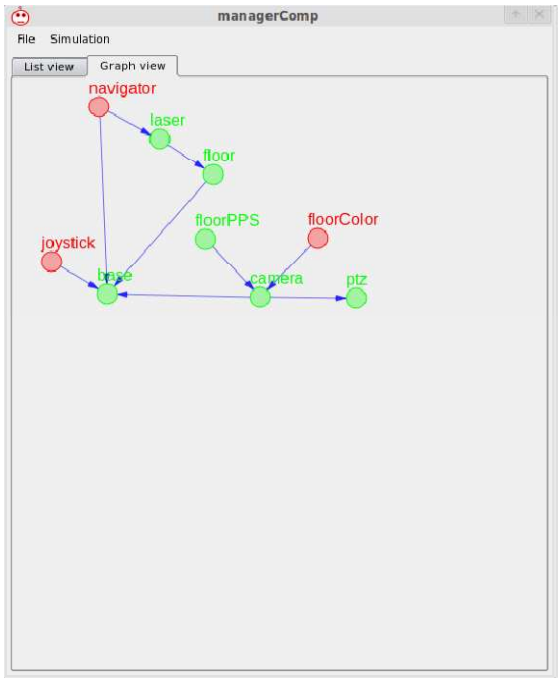

Figura 43: Captura de la vista gráfica de la aplicación managerComp

6.2 Experimentos Hokuyo URG-04LX

A continuación se describirán los experimentos llevados a cabo con este sensor así como los resultados obtenidos.

6.2.1 Variación con la distancia al objetivo.

Para la realización de este experimento tomamos 3 medidas de 1000 muestras cada una situando al sensor a una distancia comprendida de 1-3 m de una pared de lucido naranja en primer lugar y en segundo de una pared de baldosines blancos brillante. El ángulo de incidencia tomado fue de 0º. Comentar que pese a que este sensor tiene un alcance de hasta 4 metros, dadas las condiciones del área de trabajo tuvimos que hacer las medidas hasta un máximo de 3 m.

Figura 44: Captura experimento variación con la distancia al objetivo.

Los resultados obtenidos son:

Resultados pared de lucido naranja:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
1 m	995.15	2.845	1.688
2 m	2042.92	2.637	1.624
3 m	3062.72	3.451	1.858

Figura 45: Tabla con resultados sensor URG-04LX de variación con la distancia pared naranja.

Resultados pared de baldosines blancos brillante:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
1 m	1023.31	2.082	1.443
2 m	2013.29	2.458	1.568
3 m	2978,44	3,265	1,806

Figura 46: Tabla con resultados sensor URG-04LX de variación con la distancia pared azulejos.

A continuación representamos gráficamente los resultados para observarlos de forma clara:

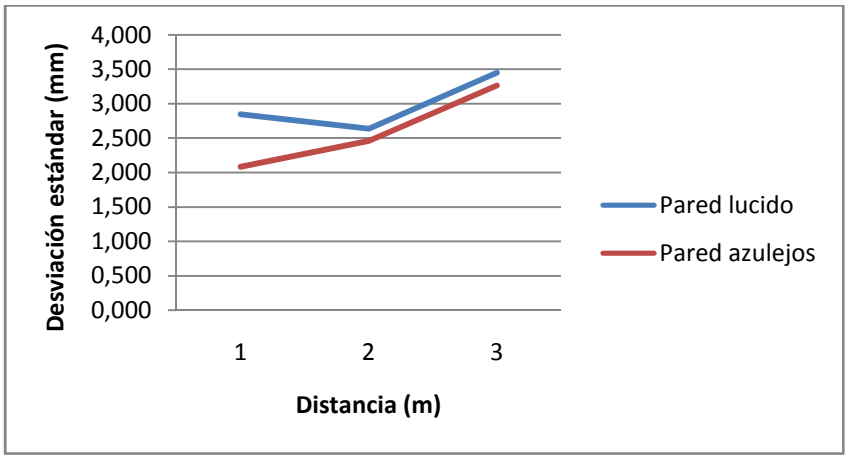

Figura 47: Comparativa valores desviación estándar variando la distancia para láser URG-04LX.

Tal y como podemos observar tanto en la gráfica como en los resultados, se obtienen unos valores de desviación que se va incrementando ligeramente a medida de la distancia aumenta. Esto es debido a que a medida que aumenta la distancia entre el láser y el objetivo, el diámetro del rayo láser también aumenta lo que conlleva una disminución de la intensidad del rayo reflejado capturado. Este efecto puede provocar que exista una diferencia creciente entre la distancia medida y la distancia nominal, además de un aumento de la dispersión, hecho que podemos comprobar con los valores de varianza obtenidos.

Resaltar que hemos obtenido valores tanto de desviación estándar como de varianza ligeramente inferiores en la pared de azulejos con respecto a la de lucido. Esto puede ser debido a que debido al color blanco de la pared de azulejo y a su superficie brillante, la mayor parte del rayo láser es reflejado inmediatamente después de alcanzarla siendo muy inferior la absorción del rayo por lo que se obtiene una lectura mejor.

6.2.2 Variación con el ángulo de incidencia

Para la ejecución de este experimento se situó el robot a 1.5 metros del objetivo y se tomaron 1000 muestras variando el ángulo de medida de 0º hasta el máximo ángulo de captura de sensor. Explicar que lo que variamos es el ángulo del sensor siendo el punto de medida siempre el mismo.

Al igual que para el experimento anterior, se realizaron las medidas tanto para la pared de lucido naranja como para la de azulejos.

Los resultados obtenidos para la pared de lucido son los siguientes:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
0º	1486,54	2,360	1,536
20º	1502,29	2,235	1,495
40º	1530,39	2,470	1,572
60º	1568,62	2,472	1,572
70º	1590,35	2,606	1,614

Figura 48: Tabla de resultados de URG-04LX obtenidos para diferentes ángulos sobre pared lucida naranja.

El límite de captura (máximo ángulo que permite tomar medidas) es de 78º.

Resultados para pared de azulejos blancos brillantes:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
0º	1508,66	2,273	1,508
20º	1517,79	2,309	1,519
40º	1542,99	2,288	1,513
60º	1580,91	2,235	1,495
70º	1605,25	2,329	1,526

Figura 49: Tabla de resultados de URG-04LX obtenidos para diferentes ángulos sobre pared azulejos blancos.

El límite de captura para la pared de azulejos es de 76º.

En la siguiente gráfica podemos observar una comparativa de las variaciones de la desviación estándar respectos a los dos tipos de pared medidas:

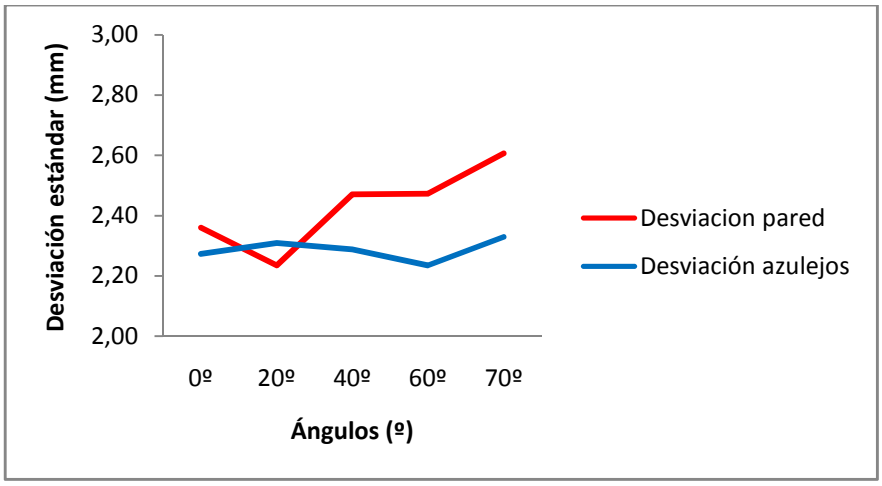

Figura 50: Comparativa valores desviación estándar variando el ángulo de incidencia para láser URG-04LX.

A la vista de los resultados obtenidos podemos ver como para la pared de lucido, salvo por el valor a 20º, va incrementado su desviación estándar así como el valor medio de la distancia. Esto es debido a que a medida que el ángulo de incidencia aumenta, la parte principal del rayo láser incidente no es reflejado en la dirección del sensor, produciendo así una medida ligeramente deficiente.

Afirmar también que si bien el ángulo de incidencia influye en el error cometido sobre la medida, este es inferior a los valores obtenidos para diferentes distancias.

Para el caso de la pared de azulejos, hemos obtenido unos valores de desviación y dispersión bastante estables, con poca variación. Una posible explicación sería las condiciones superiores reflectivas de esta superficie.

6.2.3 Influencia del color, brillo y material de superficie utilizada.

Para la realización de este experimento se fabricaron sendas superficies sobre las que realizar las medidas. Para la fabricación se emplearon cartulinas blancas de tamaño DIN-A4 y se recubrieron de papel pinocho de colores amarillo, rojo, azul, negro y blanco, y una última cartulina recubierta de papel de plata. El resultado final puede verse en la siguiente figura:

Figura 51: Captura conjunto de cartulinas de prueba.

Para la consecución del experimento se situó al robot a 1.5 m de la superficie objetivo y se tomaron medidas de 1000 muestras a 0º para cada superficie. Los resultados pueden verse en la siguiente tabla:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
Rojo	1498,079	2,192	1,481
Azul	1501,85	2,759	1,661
Amarillo	1498,72	2,364	1,538
Negro	1507,72	2,676	1,636
Blanco	1503,58	2,209	1,486
Plata	1504,60	3,336	1,826

Figura 52: Tabla resultados URG-04LX para diferentes materiales y colores.

En el siguiente gráfico podemos apreciar los resultados:

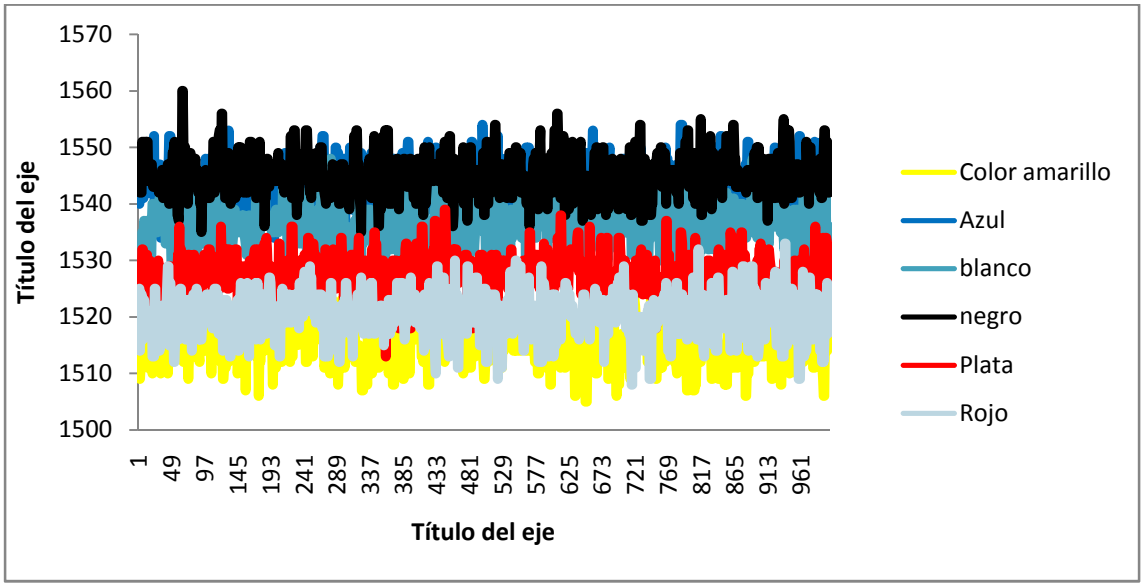

Figura 53: Distancias medidas para diferentes colores y superficies con láser URG-04LX

De los resultados obtenidos podemos deducir que los colores rojo y amarillo son los que han medido una distancia más corta como podemos ver en sus valores medio, debido a que el tiempo de vuelo del laser es menor al ser reflejado casi en su totalidad inmediatamente al incidir. La cartulina recubierta de plata debería estar entre los colores con menor distancia, si atendemos a los resultados obtenidos por Cang [14] y Lee[15] en experimentos similares ya que posee unas propiedades reflectivas óptimas, sin embargo si consultamos la tabla comprobamos que posee la mayor desviación estándar y varianza de todos. Una posible explicación es que el papel tenía un diseño de rejilla de panal de abeja lo que podría haber producido una dispersión del rayo láser incidente.

Por último comentar que uno de los colores con mayor distancia media y desviación es el negro, ya que absorbe parte del rayo láser por un tiempo para luego volver al láser, aumentando así el tiempo de vuelo y por lo tanto la estimación de la distancia.

6.2.4 Influencia de la luz ambiente.

Desde que los robots autónomos relacionados con el presente proyecto tienen que operar en regiones oscuras, el efecto de la ausencia de luz debe ser comprobado. Sin embargo, debido a ciertas condiciones del área de experimentación decidimos no llevar a cabo este experimento y adoptar los resultados obtenidos por Laurent Kneip y Fabien Tache en su trabajo [16], ya que tal y como ellos observan solo se produce una desviación en la media de ±5 mm, siendo inapreciable las diferencias en la desviación estándar.

6.2.5 Efecto de deriva.

Para obtener una impresión de la importancia del efecto de deriva, hemos llevado a cabo un experimento con las siguientes condiciones: se situó el robot a 1.5 m de la pared de lucido naranja del laboratorio y con un ángulo de incidencia de 0º se realizo una medida de 1 hora (aproximadamente 19651 muestras).

Los resultados estadísticos obtenidos son : media = 1399,83 mm; desviación estándar = 2,63 mm; varianza = 1,621 mm^2.

Para tener una visión más clara de la deriva, en la siguiente gráfica se muestran los valores de la distancia obtenidos respecto al nº de muestras.

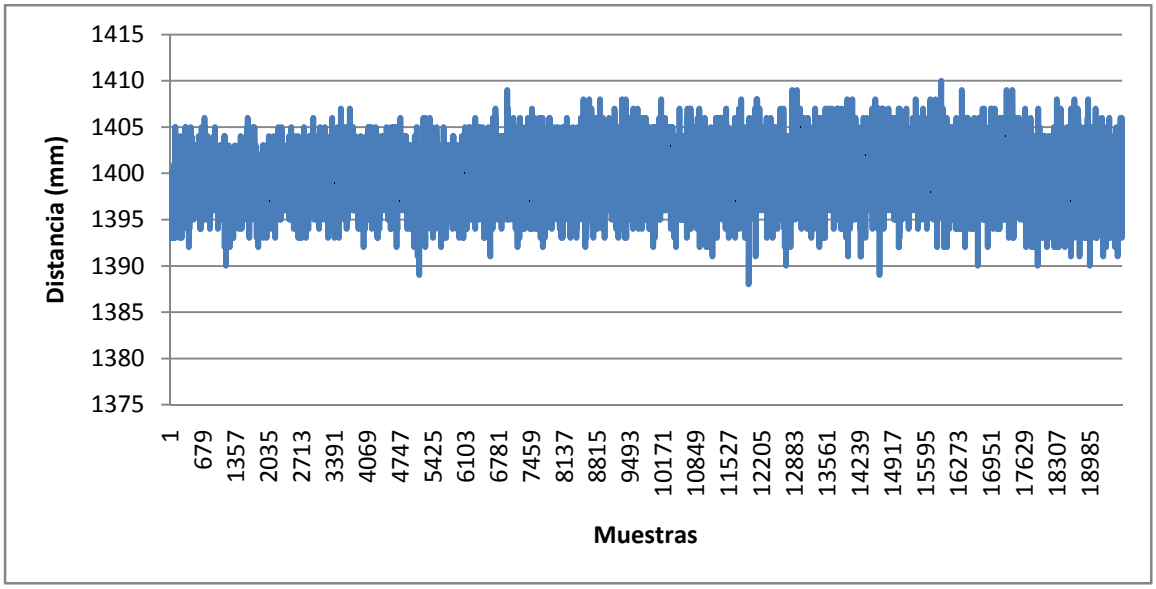

Figura 54: Gráfica efecto deriva sensor láser URG-04LX.

Si observamos la gráfica comprobamos como se va incrementado la distancia tomada a medida que transcurre el tiempo. También se observa una fluctuación en la medida que comienza por 10 mm y que acercándose al final de la medida llega a ser de 20 mm. Esto es debido probablemente a un incremento de la temperatura de funcionamiento del sensor unido a la disipación de calor por parte del motor que incorpora el sensor.

6.3 Experimentos Hokuyo UTM-30LX

Al igual que en el subapartado anterior a continuación se muestran tanto los experimentos como los resultados obtenidos para este sensor.

6.3.1 Variación con la distancia al objetivo.

El experimento realizado es similar al del otro sensor salvo que para este laser, dado que su rango de detección es de 30 m y teniendo en cuenta las condiciones del área de trabajo, hemos realizado las medidas de 1 a 8 metros con un incremento de 1 m con el fin de observar de una forma más clara las variaciones con la distancia.

Las medida se tomaron tanto para la pared de lucido como para la de azulejos.

Los resultados obtenidos son representados en la siguiente tabla:

- Pared de lucido naranja:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
1 m	1029,86	4,400	2,098
2 m	2018,315	3,800	1,949
3 m	3042,71	3,917	1,979
4 m	4006,83	3,634	1,906
5 m	5006,60	4,655	2,157
6 m	5999,29	5,435	2,331
7 m	7092,92	5,679	2,383
8 m	8065,72	5,531	2,352

Figura 55: Tabla con resultados sensor UTM-30LX de variación con la distancia pared naranja.

- Pared de azulejos de color blanco brillante:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
1 m	1024,796	4,272	2,067
2 m	2039,52	3,858	1,964
3 m	2979,93	4,061	2,015
4 m	4017,49	3,834	1,958
5 m	5010,41	4,860	2,204
6 m	6021,06	5,454	2,335
7 m	6986,12	5,660	2,379
8 m	7989,48	5,374	2,318

Figura 56: Tabla con resultados sensor UTM-30LX de variación con la distancia pared azulejos

Para poder observar de forma más clara la tendencia que sigue la desviación estándar se representa la siguiente gráfica:

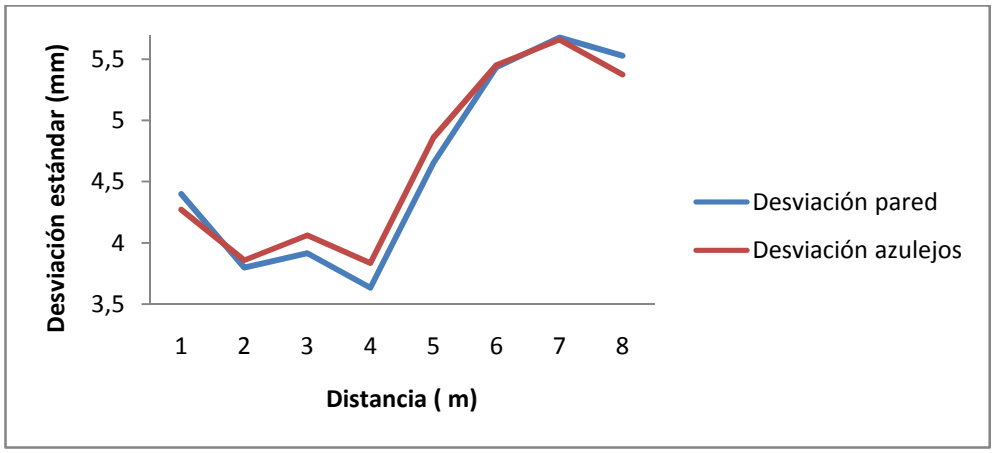

Figura 57: Tabla tendencia desviaciones estándar para sensor UTM-30LX variando la distancia.

Para este sensor volvemos a observar como aumenta la desviación a medida que aumenta la distancia entre el sensor y el objetivo, obteniéndose valores ligeramente inferiores para el caso de la pared de azulejos. La posible explicación a este efecto ya ha sido incluida en el mismo apartado correspondiente al sensor URG-04LX.

6.3.2 Variación con el ángulo de incidencia.

Para la realización de este experimento se situó el robot a 1.5 m del objetivo y se fue variando el ángulo de medida de 0º-80º con un incremento de 20º y una adquisición de 1000 muestras por medida. Al igual que para el otro sensor, las medidas se realizaron para ambos tipos de pared.

Comentar que para el sensor URG-04LX se tomaron medidas hasta 70º debido a que su ángulo máximo de captura está por debajo de 80º.

Los resultados para la pared de lucido son los siguientes:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
0º	1522,10	3,785	1,946
20º	1534,83	3,921	1,980
40º	1557,35	3,774	1,943
60º	1597,40	3,668	1,915
80º	1536,79	3,910	1,977

Figura 58: Tabla de resultados de UTM-30LX obtenidos para diferentes ángulos sobre pared de lucido naranja.

Ángulo máximo de captura de 127º.

Resultados obtenidos para pared de azulejos:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
0º	1528,46	3,652	1,911
20º	1540,08	3,938	1,984
40º	1562,54	4,039	2,009
60º	1535,28	3,895	1,973
80º	1549,13	3,819	1,954

Figura 59: Tabla de resultados de UTM-30LX obtenidos para diferentes ángulos sobre pared de azulejos.

Ángulo máximo de captura de 120º.

En la siguiente gráfica podemos observar la tendencia de variación de las desviaciones estándar según el ángulo de incidencia para ambas paredes.

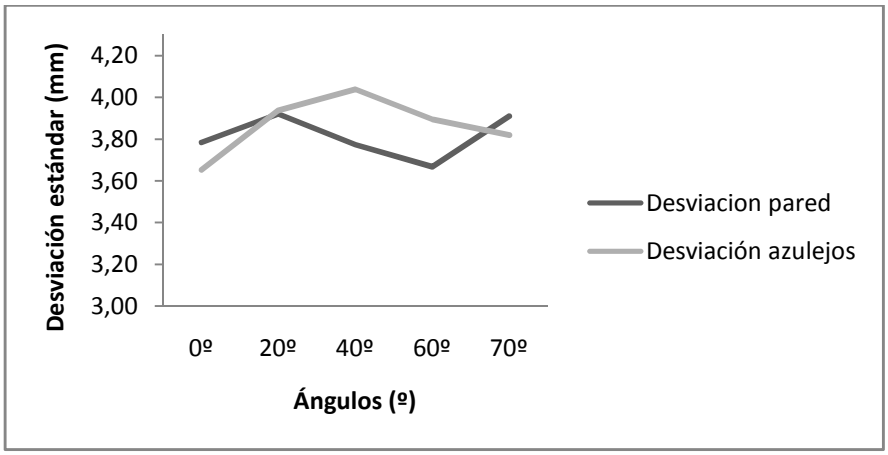

Figura 60: Comparativa valores desviación estándar variando el ángulo de incidencia para láser URG-04LX.

Los resultados obtenidos para diferentes ángulos de incidencia son bastante similares para ambas superficies y no existe una gran variación al aumentar el ángulo. Si bien los valores obtenidos tanto de desviación como de valor medio son superiores a los del otro láser, el hecho de que de unas mediciones en cierto modo constantes lo hace preferible.

6.3.3 Influencia del color, brillo y material de superficie utilizada.

Para la realización de este experimento se siguieron las mismas condiciones que para el otro sensor láser.

. Los resultados pueden verse en la siguiente tabla:

	Media (mm)	Desviación estándar (mm)	Varianza (mm^2)
Rojo	1520,29	4,023	2,006
Azul	1543,33	3,933	1,983
Amarillo	1515,39	3,807	1,951

Negro	1545,03	3,744	1,935
Blanco	1536,35	3,731	1,931
Plata	1527,34	3,653	1,911

Figura 61: Tabla resultados UTM-30LX para diferentes materiales y colores.

En este caso volvemos a tener las distancias menores para los colores rojo y amarillo y de la mayor para el color negro como cabría de esperar. Resaltar que volvemos a obtener valores más altos de desviación estándar para este sensor.

6.3.4 Influencia de la luz ambiente.

Desde que los robots autónomos relacionados con el presente proyecto tienen que operar en regiones oscuras, el efecto de la ausencia de luz debe ser comprobado. Sin embargo, debido a ciertas condiciones del área de experimentación decidimos no llevar a cabo este experimento y adoptar los resultados obtenidos por Laurent Kneip y Fabien Tache en su trabajo, ya que tal y como ellos observan solo se produce una desviación en la media de ±5 mm, siendo inapreciable las diferencias en la desviación estándar.

6.3.5 Efecto de deriva.

El procedimiento seguido para la realización de este experimento es similar al ejecutado para el otro sensor.

Los resultados obtenidos son: media = 1362,39 mm ; desviación estándar = 4,50 mm ; varianza = 2,12 mm^2.

Observando la siguiente gráfica podemos observar los efectos de la deriva:

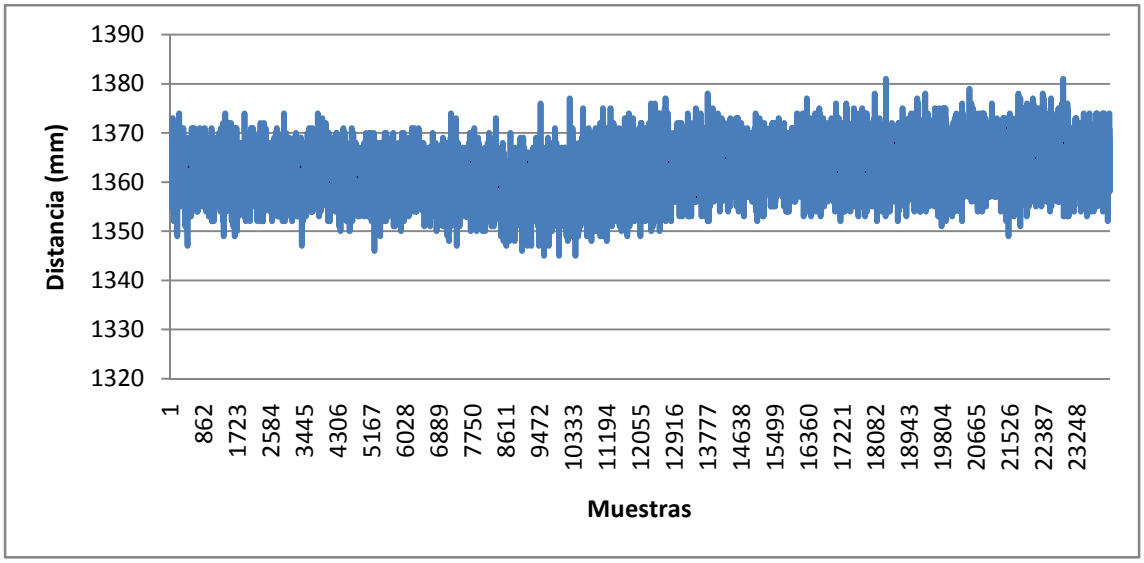

Figura 62: Gráfica efecto deriva sensor láser UTM-30LX.

Comprobamos que la fluctuación inicial es de unos 20 mm, llegando a ser de hasta 30 mm a mitad de medida. Vuelve a ser un resultado elevado comparado con los del otro sensor. De nuevo se aprecia como a medida que aumenta el tiempo se adquieren valores de la distancia más altos.

6.4 Comparación de sensores.

A la vista de los resultados podemos asegurar que si bien en cuanto a las características técnicas es superior el sensor UTM-30LX en distancia de medida, rango de detección, etc. Posee unos valores estadísticos de error también superiores con respecto al sensor URG-04LX. Destacar que en cuanto al experimento de la variación del ángulo de incidencia pese a que ha obtenido valores de error también superiores, estos valores permanecen casi constantes por lo que se obtiene una medición semejante para los diferentes ángulos de incidencia., mientras que para el sensor URG-04LX existe un fuerte dependencia con el ángulo de incidencia.

Finalmente citar que dado que ambos sensores poseen diferentes rangos de medida, es complicado ofrecer una comparación equitativa.

Conclusiones

Mediante el presente Proyecto Fin de Carrera se ha presentado la implementación de un sistema que permite la caracterización estadística de cualquier sensor láser de rango, así como la ejecución de una serie de experimento que nos permitieran hacernos una idea de las capacidades de los dos sensores disponibles en el laboratorio de robótica de la Universidad de Extremadura, Robolab, y que son: Hokuyo URG-04LX y Hokuyo UTM-30LX.

El resultado de la implementación del sistema fue una herramienta versátil con un alto grado de reusabilidad y mantenibilidad, gracias a la utilización para su desarrollo, del lenguaje de programación C++ y del paradigma de programación orientada a componentes.

Lo más importante de todo es que se ha obtenido una herramienta que en primer lugar, gracias a su diseño orientado a componentes, será fácil acoplarlo a otros componentes existentes para que funcionen en conjunto, extendiendo así las capacidades del robot y en segundo lugar, debido a su diseño permite una selección de gran cantidad de parámetros de medida lo que permite consultar datos específicos en tiempo real acerca de las medidas que esté tomando el sensor.

Anexo I

Como ya hemos comentado anteriormente, en este apartado se incluirán las características técnicas de los sensores láser empleados, además de aquellas partes del código que dada su extensión no se incluyeron en capítulos anteriores,

Características técnicas sensor láser Hokuyo URG-04LX.

Nombre del producto	Scanning Laser Range Finder
Modelo	URG-04LX
Fuente de luz	Semiconductor laser diode (λ =785nm), Laser power : less than 0.8mW Laser safety Class 1 (IEC60825-1)
Fuente de voltaje	5V DC ±5%
Corriente suministrada	500mA or less (Start-up current 800mA)
Detección	60 mm ~ 4,095 mm (distancia de precisión garantizada) 20mm ~ 5,600mm (Distance)
Precision	Distance 20 ~ 1000mm : ±10mm Distance 1000 ~ 4000mm : ±1% of measurement
Resolución	1 mm
Ángulo de escaneo	240º
Resolución angular	0.36° (360° /1024)
Velocidad de escaneo	100msec/scan
Interfaz	RS-232C (19.2, 57.6, 115.2 ,500 ,750 kbps) USB Version 2.0 FS mode (12Mbps)
Condiciones ambientales	-10 ~ 50oC / 85%RH or less (without dew and frost)
Temperatura almacenamiento	-25 ~ 75oC
Resistencia luz ambiente	10000Lx or less
Resistencia a vibraciones	1.5mm double amplitude, 10 ~ 55Hz, X, Y and Z direction (2 hours), 98m/s2 55Hz ~ 150Hz in 2 minutes sweep, 1 hour each in X, Y and Z direction

Resistencia a golpes	196 m/s2, 10 times each in X, Y and Z direction
Estructura de protección	Optics : IP64 Case : IP40
Aislamiento	10MΩ for DC 500Vmegger
Peso	Approx. 160 g
Carcasa	Policarbonato
Dimensiones (WxDxH)	50×50×70mm

Figura 63: Tabla de características laser Hokuyo URG-04LX

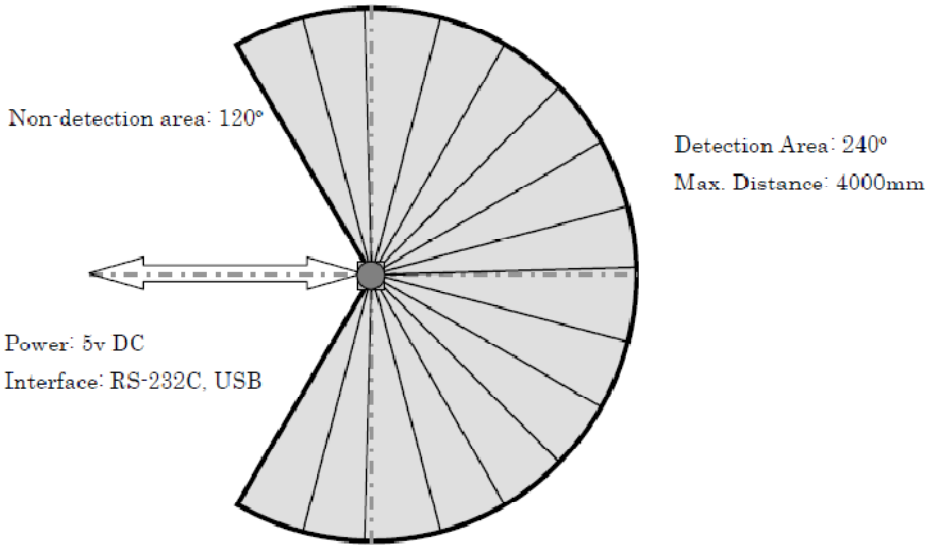

Figura 64: Esquema área de detección sensor láser Hokuyo URG-04LX .

Características técnicas sensor láser Hokuyo UTM-30LX

Nombre del producto	Scanning Laser Range Finder
Modelo	UTM-30LX
Fuente de luz	Laser Semiconductor λ = 905nm, Laser Class 1
Fuente de voltaje	12V DC ± 10%
Corriente suministrada	Max: 1ª, Normal : 0.7ª
Detección	Guaranteed Range: 0.1 ~ 30m (White Kent Sheet) Maximum Range : 0.1 ~ 60m Minimum detectable width at 10m : 130mm (Vary with distance)

Precision	Under 3000lx : White Kent Sheet: ±30mm (0.1m to 10m) Under 100000lx : White Kent Sheet: ±50mm (0.1m to 10m)
Resolución	1mm 8.6 – 10m : σ <10mm, 10 – 30m : σ <30mm (White Kent Sheet) Under 3000lx : σ = 10mm*1 (White Kent Sheet up to 10m) Under 100000lx : σ = 30mm*1 (White Kent Sheet up to 10m)
Ángulo de escaneo	270º
Resolución angular	0.25º (360°/1024)
Velocidad de escaneo	25ms（Motor speed : 2400rpm）
Interfaz	USB Ver2.0 Full Speed (12Mbps)
Condiciones ambientales	-10ºC ~ ＋50ºC Less than 85%RH (Without Dew, Frost)
Temperatura almacenamiento	-25 ~ 75oC
Resistencia a vibraciones	10 ~ 55Hz Double amplitude 1.5mm in each X, Y, Z axis for 2hrs. 55 ~ 200Hz 98m/s2 sweep of 2min in each X, Y, Z axis for 1hrs.
Resistencia a golpes	196m/s2 In each X, Y, Z axis 10 times.
Estructura de protección	Optics : IP64
Aislamiento	10MΩ for DC 500Vmegger
Peso	Approx. 210 g
Carcasa	Policarbonato
Dimensiones (WxDxH)	60mm×60mm×85mm

Figura 65: Tabla de características laser Hokuyo UTM-30LX

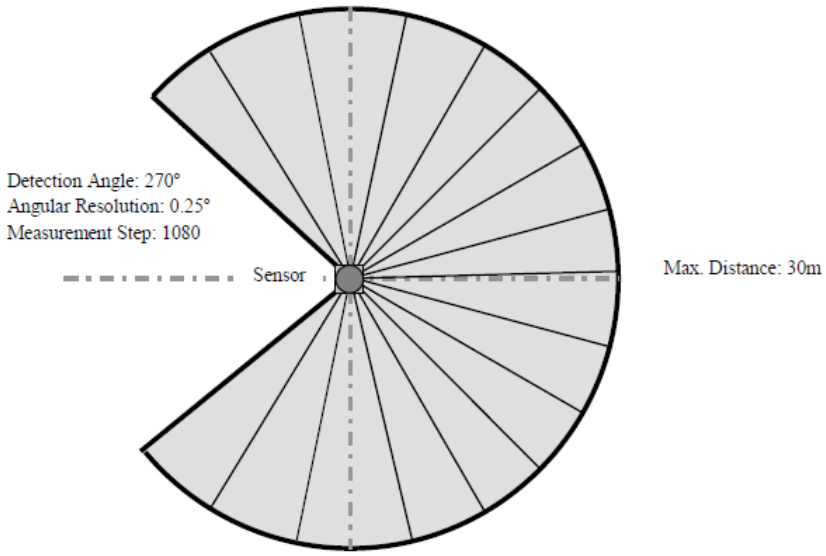

Figura 66: Esquema área de detección sensor láser Hokuyo UTM-30LX

Anexo II

Función StoreData:

```
int Worker::storeData(RoboCompLaser::TLaserData laserData, float angle)
{
  float inc_angle;
  confData = _laser_prx->getLaserConfData();
  inc_angle = confData.angleRes;
  interval = init.elapsed(); /* ms */
  Informobt->append( QString("intervalo: %1").arg(interval/1000));
        if (checkLimit->isChecked())
         {
              if (cont <= Limite)
              {
                 for (unsigned int i=0; i<laserData.size(); i++)
                {
 if (((laserData[i].angle - inc_angle) < angle) && (angle < (laserData[i].angle + inc_angle)))
                  {
                    sumadist+=laserData[i].dist;
                    distancia.resize(dist_i+1);
                    dist_i = distancia.size();
                     cont++;
                  }
 }
               return 0;
              }
             else return 1;
    }
     else if (checkTime->isChecked())
     {
       if (interval <= time)
              {
                for (unsigned int i=0; i<laserData.size(); i++)
                {
if (((laserData[i].angle - inc_angle) < angle) && (angle < (laserData[i].angle + inc_angle)))
                  {
                    sumadist+=laserData[i].dist;
                     distancia.resize(dist_i + 1);
                     dist_i = distancia.size();
                     distancia[cont] = laserData[i].dist;
                     cont++;
                  }
                }
                return 0;

              }
             else return 1;
     }
}
```

Figura 67: Función storeData

Función CalcularDatos:

```
void Worker::CalcularDatos()
{

    comprobar = storeData(laserData, angle);
    drawdistances();
    drawvariance();

    // Comment: Una vez obtenidos todos los datos de distancia procedemos a
calcular:

     if (comprobar == 1) {

     // Comment: Average distance
     promedio = sumadist/cont;

     // Comment: standar deviation
     sigma = calcularDesviacion(sigma);

     // Comment: Variance
     varianza = sqrt(sigma);

     Informobt->clear();
     Informobt->append( QString("Mean:   %1mm").arg(promedio));
     Informobt->append( QString("Standard Deviation:   %1 mm²").arg(sigma));
     Informobt->append( QString("Variance:   %1mm").arg(varianza));
     seguir = false;
     dist_i = 0;
     sumadist = 0;
     cont = 0;
     Calcular->setChecked(false);
     checkTime->setChecked(false);
     checkLimit->setChecked(false);
     }
    else {
      if (checkTime->isChecked())
          Informobt->append( QString("calculando resultados 1").arg(interval/1000));
     else
          Informobt->append( QString("calculando resultados %1").arg(cont));
    }
}
```

Figura 68: Función CalcularDatos

Función calcularDesviación

```
float Worker::calcularDesviacion(float sigma{
//storeData(laserData, angle, dist_i);

for (signed int i=0; i < cont; i++)
{float Worker::calcularDesviacion(float sigma)
{
//storeData(laserData, angle, dist_i);

std::cout << " SIZE " << distancia.size() << std::endl; // Aqui size es 0
std::cout << "Limite: "<< Limite << std::endl;

for (signed int i=0; i < cont; i++)
{
std::cout<<"dist[i]" << distancia[i] << std::endl;
sigma += pow((distancia[i] - promedio), 2);

}

sigma = sigma/cont;

sigma = sqrt(sigma);

return sigma;
}
std::cout<<"dist[i]" << distancia[i] << std::endl;
sigma += pow((distancia[i] - promedio), 2);

}

sigma = sigma/cont;

sigma = sqrt(sigma);

return sigma;
}
```

Figura 69: Función calcularDesviación.

Función drawdistances:

```
void Worker::drawdistances( )
{

    graphVar->drawLine(QLine(10, graphVar->getHeight() - 10, 10, 10), Qt::black);
    graphVar->drawLine(QLine(10, graphVar->getHeight() - 10, graphVar-
>getWidth() - 10, graphVar->getHeight() - 10), Qt::black);
    graphVar->drawText(QPoint(13, 13), "distance", 10, Qt::black);
    graphVar->drawText(QPoint(graphVar->getWidth() - 13, graphVar->getHeight()
- 13), "index", 10, Qt::black);

    graphVar->show();

    if (distancia.size()!= 0)
    {
     double max = distancia[0];

     for (int i = 1; i < (int) distancia.size(); i++)
     {
          if (distancia[i] > max) max = distancia[i];
     }

     double gap = graphVar->getWidth()/distancia.size();

     int back = distancia[0];

            for (int i = 1; i < (int) distancia.size(); i++)
            {
              graphVar->drawLine(QLine(i*gap + 13, graphVar->getHeight() - 13,
i*gap + 13, (graphVar->getHeight() - (graphVar->getHeight()-
13)*distancia[i]/max)), Qt::red);
              graphVar->drawEllipse(QPoint( i*gap + 13, (graphVar->getHeight() -
(graphVar->getHeight()-13)*distancia[i]/max)), 2, 2, Qt::red);
              back = distancia[i];
            }
    }

        graphVar->update();
}
```

Figura 70: Función drawdistances.

Función drawvariance.

```
void Worker::drawvariance( )
{

    graphXY->drawLine(QLine(10, graphXY->getHeight() - 10, 10, 10), Qt::black);
    graphXY->drawLine(QLine(10, graphXY->getHeight() - 10, graphXY-
>getWidth() - 10, graphXY->getHeight() - 10), Qt::black);
    graphXY->drawText(QPoint(13, 13), "Y", 10, Qt::black);
    graphXY->drawText(QPoint(graphXY->getWidth() - 13, graphXY->getHeight()
- 13), "X", 10, Qt::black);
    graphXY->drawEllipse(QPoint(graphXY->getWidth()/2 , graphXY-
>getHeight()/2 ), 3, 3, Qt::red);

    graphXY->show();


    if (distancia.size()!= 0)
    {

          X0 = distancia[1]*cos(angles[1]);
          Y0 = distancia[1]*sin(angles[1]);


          for (int i = 2; i < (int) distancia.size(); i++)
          {

            Xi = distancia[i]*cos(angles[i]);
            Yi = distancia[i]*sin(angles[i]);

            Xpos = (Xi - X0) + graphXY->getWidth()/2; //*graphXY-
>getWidth()/2)/X0;
            Ypos = (Yi - Y0) + graphXY->getHeight()/2; //*graphXY-
>getHeight()/2)/Y0;

            if (Y0==0) Ypos = graphXY->getHeight()/2;
            if (X0==0) Xpos = graphXY->getWidth()/2;

            graphXY->drawEllipse(QPoint(Xpos , Ypos ), 3, 3, Qt::green);

          }

   }
   graphXY->update();

}
```

Figura 71: Función drawvariance.

Anexo III

Definición de Jacobiano.

En cálculo vectorial, se llama jacobiano o determinante jacobiano al determinante de la matriz jacobiana.

La matriz jacobiana es una matriz formada por las derivadas parciales de primer orden de una función. Una de las aplicaciones más interesantes de esta matriz es la posibilidad de aproximar linealmente a la función en un punto. En este sentido, el jacobiano representa la derivada de una función multivariable.

Propiamente deberíamos hablar más que de matriz jacobiana de diferencial jacobiana o aplicación lineal jacobiana ya que la forma de la matriz dependerá de la base o coordenadas elegidas. Es decir, dadas dos bases diferentes la aplicación lineal jacobiana tendrá componentes diferentes aún tratándose del mismo objeto matemático. La propiedad básica de la "matriz" jacobiana es la siguiente, dada una aplicación cualquiera $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^m$continua es decir $\mathbf{F} \in \mathcal{C}^{(k)}(\mathbb{R}^n, \mathbb{R}^m)$ se dirá que es diferenciable si existe una aplicación lineal $\boldsymbol{\lambda} \in \mathcal{L}(\mathbb{R}^n, \mathbb{R}^m)$tal que:

$$\lim_{\|\mathbf{x}-\mathbf{y}\|\to 0} \frac{\|(\mathbf{F}(\mathbf{x}) - \mathbf{F}(\mathbf{y})) - \boldsymbol{\lambda}(\mathbf{x}-\mathbf{y})\|}{\|\mathbf{x}-\mathbf{y}\|} = 0$$

Supongamos $\mathbf{F}:\mathbb{R}^n \to \mathbb{R}^m$es una función que va del espacio euclídeo n-dimensional a otro espacio euclídeo m-dimensional. Esta función está determinada por m funciones escalares reales:

$$y_i = F_i(x_1, \ldots, x_n), \qquad \mathbf{y} = \mathbf{F}(\mathbf{x}) = (F_1(\mathbf{x}), \ldots, F_m(\mathbf{x}))$$

Cuando la función anterior es diferenciable, entonces las derivadas parciales de éstas m funciones pueden ser organizadas en una matriz m por n, la matriz jacobiana de F:

$$\begin{bmatrix} \dfrac{\partial y_1}{\partial x_1} & \cdots & \dfrac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial y_m}{\partial x_1} & \cdots & \dfrac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Esta matriz es notada de diversas maneras:

$$J_{\mathbf{F}}(x_1, \ldots, x_n) \qquad \text{o} \qquad \frac{\partial(y_1, \ldots, y_m)}{\partial(x_1, \ldots, x_n)} \qquad \text{o} \qquad D\mathbf{F}(x_1, \ldots, x_n)$$

Bibliografía

[1] http://www.tekscan.com/flexiforce.html

[2] http://en.wikipedia.org/wiki/Laser_rangefinder

[3] http://www.slideshare.net/diego5wh/sensores-de-distancia

[4] http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx.html

[5] http://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html

[6] http://www.sick.es/es/productos/autoident/medicion_laser/interior/es.html

[7] RoboComp http://sourceforge.net/apps/mediawiki/robocomp

/index.php?title=Main_Page#Components.

[8] Robolab. RobEx arena http://robexarena.com.

[9] Ramón Cintas Peña "Técnicas visuales de reconocimiento de objetos en robots móviles ". Proyecto Fin de Carrera 2009.

[10] Página del laboratorio de robótica de la UEX http://robolab.unex.es/

[11] Creative Commons España http://es.creativecommons.org.

[12] Luis J. Manso Fernández-Argüelles – " *Navegación visual en robots móviles"*.

[13] http://es.wikipedia.org/wiki/Qt_Development_Frameworks

[14] Cang Ye and Johann Borenstein - *Characterization of a 2-D Laser Scanner for Mobile Robot Obstacle Negotiation*. The University of Michigan.

[15] Kyeong-Hwan Lee, Reza Ehsani - *Comparison of two 2D laser scanners for sensing objectdistances, shapes, and surface patterns*. University of Florida.

[16] Yoichi Okubo*, Cang Ye**, and Johann Borenstein - Characterization of the Hokuyo URG-04LX Laser Rangefinderfor Mobile Robot Obstacle Negotiation