UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado de Ingeniería Electrónica Industrial y Automática

Desarrollo de una aplicación para la detección y seguimiento de objetos. Aplicación a vehículos móviles.

Autor del proyecto: David Palazón Hidalgo

Tutora del proyecto: Marina Vallès Miquel

Cotutor del proyecto: Ángel Valera Fernández

Índice de la Memoria

1. Objeto

El objetivo final del presente proyecto es realizar el desarrollo de una aplicación para detectar y seguir objetos a través de un sensor 2D de tecnología LiDAR y una cámara UVC aplicable a cualquier medio de transporte y robótica, en este caso a un automóvil híbrido (robot móvil). La programación de la aplicación de dicho sensor se realiza mediante el uso de la arquitectura software ROS (Robotic Operating System). A su vez en dicho proyecto se añadirá una cámara con puerto USB para la detección y el seguimiento de la silueta del objeto, para una mayor precisión y mejor valoración de los datos adquiridos.

El documento comienza con una introducción al mundo de la robótica, explicando que es y cómo puede ser clasificada, prestando especial atención a la clasificación de la robótica según su arquitectura, dividiéndose en robots fijos y robots móviles.

Una vez introducida la robótica, se va a continuar con la explicación de los sensores, centrándose principalmente en el sector automovilístico, dado que el robot para el cual se ha pensado aplicar es equivalente a un coche eléctrico (propulsado por una pila de hidrógeno). Con esta información se pretende poder clasificar y comprender el funcionamiento del sensor empleado, en nuestro caso el 2D de Hokuyo y la cámara USB.

Para entender las características del sensor Hokuyo URG-04LX-UG01, se ha explicado su funcionamiento y su implementación, integrado en el robot. Para ello, se diseñara e implementarán los controles para dicho sensor y para la cámara USB, los cuales deberán detectar y seguir los distintos objetos con la finalidad de conocer el entorno del móvil y suministrar la información necesaria, con el objetivo de poder crear un patrón de análisis de objetos, tanto en forma como en distancia.

El sensor Hokuyo URG-04LX-UG01 se programa mediante el uso de la arquitectura software ROS, por lo que es necesario conocer el funcionamiento de dicha arquitectura antes de poder programar el sensor. Por el contrario la, dedicada al seguimiento y detección de imágenes a través de cámaras web o de conexión USB 2.0.

Una vez se comprende el funcionamiento de ROS y de ViSP, y la implementación del control de detección y seguimiento de objetos, se puede alcanzar el objetivo del proyecto, que es aplicarlo a un robot móvil.

2. Justificación del Proyecto

2.1. Antecedentes

La motivación de este proyecto viene generada por la constante evolución y creciente demanda de sistemas de monitorización de señales y automatización de mecanismos, y como no, las técnicas de teledetección por sensores láser, conocidas como LiDAR (Light Detection and Ranging). [9][10][11]

Existe una evolución cada día mayor en este sector, parte de este fenómeno es gracias a la aparición de nuevos materiales fotosensibles, circuitos más pequeños y procesadores más potentes, rápidos y eficaces. La mayoría de fabricantes de automóviles han ido añadiendo progresivamente sensores a los vehículos durante las últimas tres décadas, hasta adquirir una gran automatización y control por parte del vehículo, haciendo de este, una maquina más segura y cómoda para el usuario. [1]

Estos sensores y microprocesadores han ayudado a desarrollar sistemas de seguridad tan importantes como el ESP (en alemán “Elektronisches Stabilitätsprogramm”, o en español “programa electrónico de estabilidad”) o ABS (en alemán “Antiblockiersystem”, o en español “sistema antibloqueo de ruedas”), usados y obligatorios en los vehículos actuales. Con ellos se ha aumentado la eficiencia de la seguridad del automóvil, protegiendo mejor a sus usuarios y al resto de conductores. Por otro lado tenemos sensores de proximidad, que ayudan al estacionamiento del vehículo, un ordenador a bordo multifunción (información sobre consumo de combustible, autonomía, temperatura del motor, presión de neumáticos, temperatura exterior etc.). A su vez, se ha conseguido mejorar el consumo de los motores gracias a la aplicación de la electrónica en la mecánica de los mismos (sistemas de inyección y compresión). [2][3]

Actualmente la tecnología LiDAR, ha ganado terreno en la industria actual, principalmente en el campo tanto de la robótica como en el automovilístico. Sus principales ventajas son la velocidad que se reduce a décimas de segundo, un haz de luz láser más estrecho y menos divergente, más fácil de manejar, transportar y mantener, más económico que un radar y funcionar igual que el mismo. Por todas estas ventajas se ha elegido este tipo de tecnología para dicho proyecto. [9][10][11]

2.2. Historia de la Robótica

La robótica es la técnica que se utiliza para el diseño y la construcción de robots y aparatos que realizan operaciones y/o trabajos, generalmente en instalaciones de carácter industrial y como sustitución a la mano de obra humana.

La robótica combina diversas disciplinas, como son:

-Mecánica: soporte y estructura del robot, la cual se articulara y hare de sustento a los circuitos y sensores.

-Electrónica: encargada del suministro de energía como el aporte de los componentes electrónicos.

-Informática: encargada de la programación de los sistemas electrónicos a través de software para controlar la estructura mecánica, sensores y circuitos.

-Inteligencia artificial: parte de la informática y electrónica, la cual permite aprender, tomar decisiones y razonar al robot.

-Ingeniería de control: encargada del uso de controladores y aplicaciones para reducir la intervención humana en un sistema.

Otras áreas o disciplinas seria: algebra, física, los autómatas programables y las máquinas de estados. [4][5]

El principal objetivo de la robótica, es la construcción de dispositivos que funcionen de maneara autónoma, y que a su vez, realicen los trabajos más dificultosos y repetitivos para los humanos.

El inicio de la robótica tal y como la conocemos a día de hoy, se originó en la revolución industrial (Siglo XIX). Aquí se crearon diversas máquinas y mecanismos, principalmente enfocados al sector textil, como el telar, anteriormente ya habían sido desarrollados varios mecanismos, por los cuales los brazos de esculturas se movían e imitaban el movimiento del ser humano. [4][5]

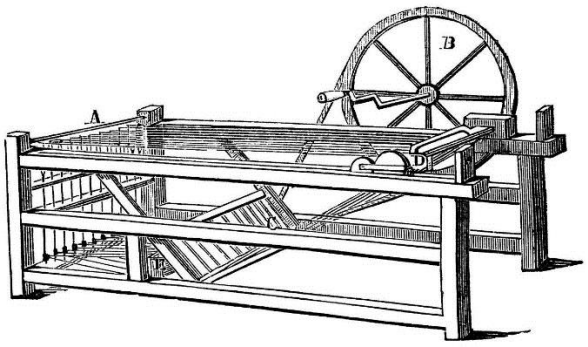


Fig.1.*Telar Revolución Industrial*

La palabra robot se utilizó por primera vez en 1920 en la obra “Los Robots Universales de Rossum”, escrita por Karel Capek. Esta novela trata sobre un hombre que fabrico un

robot y luego este último mata al hombre. La palabra checa 'Robota', significa servidumbre o trabajo forzado, y cuando se tradujo al inglés se convirtió en el término robot. Finalmente el término comúnmente conocido como robótica, fue acuñado por

Isaac Asimov, definiéndola como la ciencia que estudia a los robots. Asimov creo también las Tres Leyes de la Robótica. [4][5]

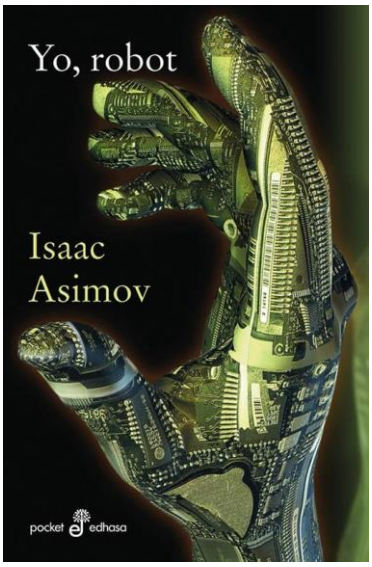


Fig.2. *Novela Yo Robot de Isaac Asimov*

En la ciencia ficción, el hombre ha imaginado a los robots como formas de inteligencia artificial que se hacen con el poder, viajando por el espacio o simplemente teniendo un uso doméstico. Actualmente los robots ocupan un lugar muy importante en el sector industrial: facilitan las labores en las cadenas de montaje y procesos repetitivos, además de su capacidad para llegar a lugares o realizar trabajos a los que el ser humano no puede llegar o realizar.

Numerosos estudios se están encargando de determinar el lugar que ocuparan los robots en el futuro, llegando la mayoría de ellos a la conclusión de que en el futuro la mayoría de los puestos de trabajo que hoy ocupan los seres humanos, serán ocupados por ellos. [4][5]

2.3. Clasificación de los ROBOTS

Podemos distinguir dos tipos diferentes de robots a la hora de clasificarlos. Según su cronología podemos diferenciar su generación, y dependiendo de la estructura, los podemos clasificar según su estructura o arquitectura. [6]

2.3.1. Por Generación

Primera Generación: Robots manipuladores, desarrollados a principios de los 50. Están compuestos principalmente por sistemas mecánicos multifuncionales con un sencillo sistema de control, manual, de secuencia fija o variable.

A su vez, cuentan con un sistema de control en lazo abierto, por lo que no hay realimentación por parte del sensor, realizando tareas preprogramadas ejecutadas de forma secuencial.

Por último, estos robots no se percatan de su entorno, por lo cual la información adquirida es muy escasa o nula. [6]

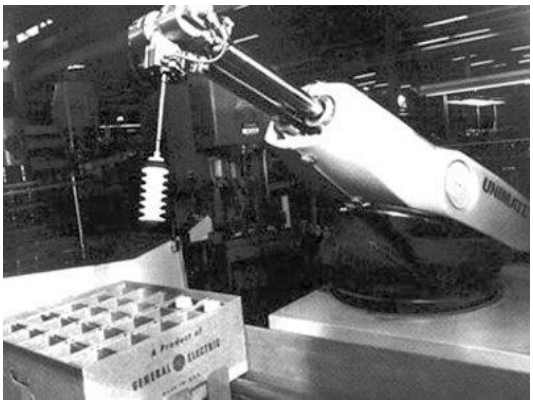


Fig.3. *Robot Primera Generación*

Segunda Generación: Robots de aprendizaje, desarrollados en la década de los 80. Estos repiten una secuencia de movimientos que ha sido ejecutada previamente por un operador humano. Su modo de hacerlo es a través de un dispositivo mecánico, cuyos movimientos son realizados por el operador mientras el robot los sigue y memoriza para su posterior aplicación.

Finalmente, estos robots son un poco más conscientes de su entorno que la previa generación, así mismo disponen de sistemas de control en lazo cerrado, donde a través de sensores adquieren información de su entorno y obtienen cierta capacidad para actuar y adaptarse según los datos adquiridos y computados. [6]

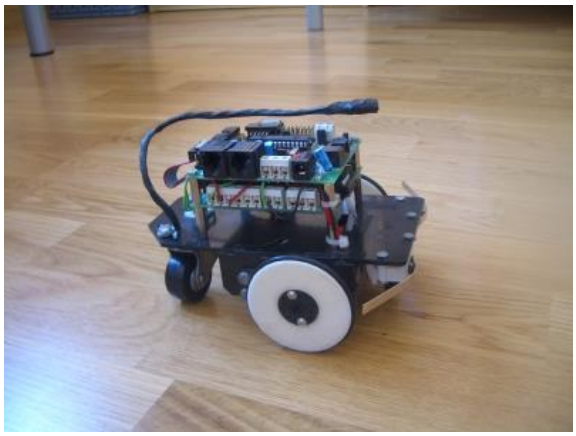


Fig.4. *Robot Segunda Generación*

Tercera Generación: Robots con control sensorizado, el controlador del mismo es una computadora, encargada de ejecutar las ordenes de un programa y enviarlas para que se realicen los movimientos necesario.

Esta generación fue desarrollada durante los años 80s y 90s, los robots incorporan controladores (computadoras) que usando los datos o la información obtenida de los sensores, adquieren la habilidad de ejecutar las órdenes de un programa escrito en alguno de los lenguajes de programación (surgidos a raíz de la necesidad de introducir las instrucciones en dichas maquinas).

Los robots usan un control de tipo lazo cerrado, ahora son conscientes del entorno y se pueden adaptar al mismo. Así mismo los robots se vuelven reprogramables y con el uso de computadoras pueden analizar la información captada de su entorno mediante los sensores (en estas décadas se desarrolla la visión artificial). [6]

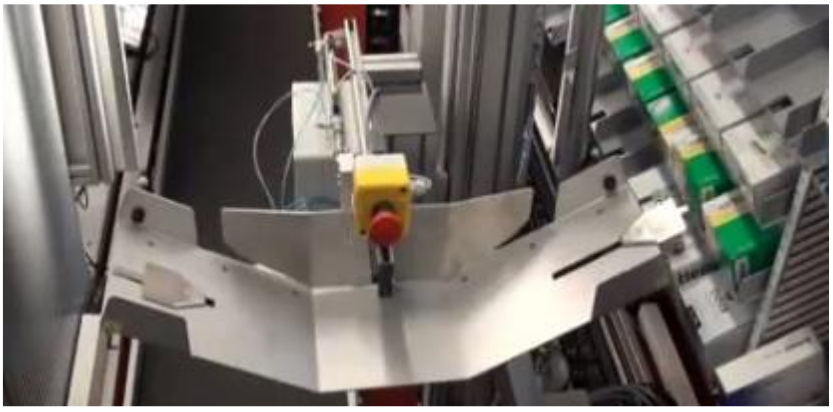


Fig.5. *Robot Tercera Generación*

Cuarta Generación: Robots inteligentes. Similares a la generación anterior, pero poseen sensores que envía información en tiempo real a la computadora de control sobre el estado del proceso. Todo esto permite a robot tomar decisiones inteligentes. Con esta nueva tecnología los robots son considerados "inteligentes", se adaptan y aprenden de su entorno, empleando ese conocimiento y otros métodos de análisis para obtener datos y mejorar su desempeño general en tiempo real, siendo capaz de basar sus acciones en una información más sólida y supervisar el ambiente que les rodea para actuar en situaciones determinadas.

Esta generación dispone de mejores sensores, mejores estrategias de control y análisis, siendo capaces de comprender el entorno y actuar en tiempo real mediante conceptos modélicos. [6]

Fig.6. *Robot Cuarta Generación*

Observando la evolución presentada en esta clasificación, podemos ver cómo ha ido avanzando la robótica con el paso del tiempo, desde los primeros robots que ejecutaban sencillas tareas de forma mecánica, hasta los robots de cuarta generación, denominados de inteligencia artificial, los cuales son capaces de reaccionar e intercambiar datos con su entorno. [6]

2.3.2. Por Arquitectura

La arquitectura está definida por la configuración general del robot, a pesar de que esta puede cambiar su forma según distintas aplicaciones o necesidades. A esto se le conoce por metamorfismo, que puede ir desde un simple cambio de herramienta hasta una compleja alteración de los elementos y sistemas que componen el robot.

Los podemos clasificar en cinco tipos:

Poliarticulados o fijos: En este grupo hayamos robots de diversas formas y configuraciones, cuya principal característica es la de permanecer fijos (aunque pueden ser guiados para realizar desplazamientos limitados) y estar estructurados para mover sus elementos terminales en un determinado espacio de trabajo según uno o más sistemas de coordenadas y con un número limitado de grados de libertad.

Podemos encontrar manipuladores, robots industriales, robots cartesianos, empleados mayoritariamente cuando es necesario abarcar una zona de trabajo amplia o actuar sobre objetos con un plano de simetría vertical. [6]

Fig.7. *Robot Poliarticulado*

Móviles: Robots que poseen gran capacidad de desplazamiento, basados en carros o plataformas móviles y dotados de un sistema locomotor de tipo rodante.

Estos robots móviles siguen su camino por un telemando o guiándose por la información adquirida del entorno a través de sus sensores. Por ello estos robots aseguran el transporte de piezas y componentes en una cadena de fabricación, guiados por pistas materializadas por radiación electromagnética en el suelo o a través de bandas detectadas fotoeléctricamente. [6]

Fig.8. Robot Móvil

Androides: En actual desarrollo. Son robots que intenta reproducir la forma y el comportamiento cinemático del ser humano.

Actualmente están poco evolucionados y sin una utilidad práctica definida, siendo destinados fundamentalmente al estudio y experimentación. [6]

Uno de los aspectos más complejos de estos robots, y sobre el que se centran la mayoría de los trabajos y estudios, es el de la locomoción bípeda, siendo el principal problema a la hora de controlar la dinámica y mantener el equilibrio de los robots.

Fig.9. *Robot Androide*

Zoomórficos: En actual desarrollo. Son robots que podrían considerarse androides, con sistemas de locomoción similares al de algunos animales.

A pesar de la disparidad morfológica de sus sistemas locomotores se puede agrupar en: caminadores y no caminadores.

La principal aplicación de estos robots será la exploración espacial y el estudio de volcanes. [6]

Fig.10. *Robots Zoomórficos*

Híbridos: En actual desarrollo. Este es el último grupo de robots, siendo difícil su clasificación debido a su estructura combinada de los grupos anteriores.

Una vez presentadas las distintas clasificaciones y tipos de robots, veremos la justificación de dicho proyecto y como lo abordaremos. [6]

2.4. Visión Artificial

La visión artificial o también conocida como visión por computador es una disciplina científica tecnológica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes tomadas del mundo real con el fin de producir información numérica o simbólica, capaz de ser trabajado por computadores. Otra definición sobre que es la visión artificial, como "la deducción automática de la estructura y propiedades de un mundo tridimensional posiblemente dinámico, bien a partir de una o de varias imágenes bidimensionales del mundo". [8]

2.4.1. Componentes de un sistema de Visión Artificial

Podemos diferenciar entre los dos tipos en los que se divide la mayoría de sistemas de control y electrónica. El compromiso entre el hardware y el software. [8]

Hardware

- Subsistema de adquisición
- Computador con tarjeta digitalizadora (“frame grabber”).
- Subsistema de visualización

Software

- Software para controlar la adquisición y almacenamiento de la imagen.
- Software para manipular la imagen.
- Software para procesar e interpretar la imagen.

2.4.2. Etapas de Diseño

Adquisición: Proceso de adquirir, transmitir y convertir las imágenes.

Preproceso: Mejora de la imagen, incluye los procesos de intensificación y eliminación de ruido.

Segmentación: Divide la imagen en las partes que la constituyen: los objetos y el fondo. Con una buena segmentación se facilita la solución del problema.

Descripción: Etiqueta los objetos con información útil (características) para identificarlos y distinguirlos.

Reconocimiento: Identifica o clasifica los objetos etiquetados.

Resultados y decisiones: Toma de decisiones una vez clasificados los objetos.

2.4.3. Imagen Digital

Para conocer el enfoque de la visión artificial debemos saber que es una imagen y como es reconocida por las cámaras y sensores. Para ello

Una imagen es una pintura, fotografía o cualquier otra forma de representación visual de un objeto o escena, también la podríamos definir como una proyección del mundo 3D en un mundo 2D.

Una imagen digital, es una matriz bidimensional de números reales o enteros, representados por un número finito de bits. Una imagen digital está formada por pixeles, que son cada pequeña área a la cual se asigna un único número en la imagen digital.

Por lo tanto la resolución espacial de la imagen es la cantidad de pixeles empleados para representar la imagen (densidad de pixeles por área). [8]

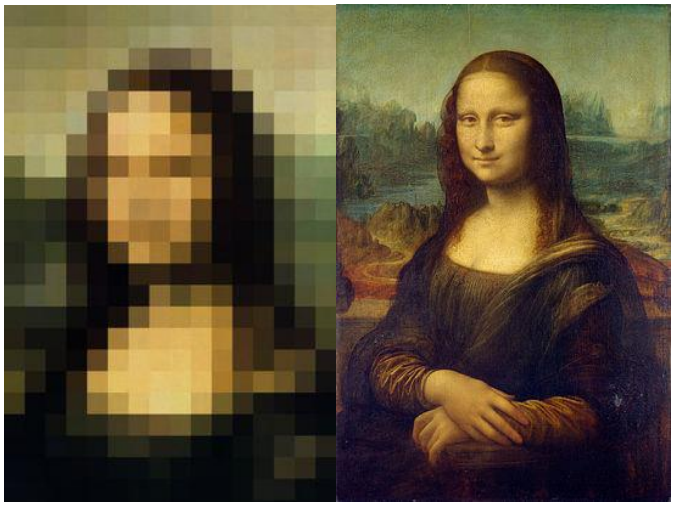


Fig.11. *Resolución y pixelado de una imagen*

Las imágenes a color están compuestas por combinación de tres colores básicos o PRIMARIOS: Rojo (R), Verde (G) y Azul (B), dando el nombre a las imágenes RGB o en color. [8]

2.4.4. Formatos de Almacenamiento

Los formatos de almacenamiento nos permiten guardar las imágenes digitales tomadas o descargadas en una computadora independientemente del sistema operativo que esta emplee. [8]

Formatos de almacenamiento de imágenes:

- Imágenes Aisladas:
  BMP, PCX, GIF, JPEG, TIFT, TGA, NETPBM
- Secuencias de Imágenes:
  MPEG, AVI

Compresión:

- Sin perdida.
- Con perdida.

2.4.5. Tipos de Iluminación

La iluminación tiene como principal objetivo optimizar el contraste, es decir separar características d el a imagen del fondo. [8]

Podemos distinguir varios tipos de iluminación:

- Incandescencia
- Fluorescencia
- Arco voltaico
- Laser
- LED
- Luz ambiental

Fuentes Incandescentes o halógenas

- Bajo coste y fáciles de utilizar.
- Permiten ajustar la intensidad de la luz.
- Desprenden una gran cantidad de calor.
- Su espectro se centra en el rojo, siendo deficiente para azules, verdes o amarillos.

Fibra Óptica

- Proporciona luz fría, no se calienta.
- Se lleva la luz desde la lámpara hasta la escena con la fibra óptica.

Fuentes fluorescentes

- No se calienta.
- Su espectro se centra en los del ojo humano.
- La duración estimada esta en torno a las 10.000 horas de uso.

- Para que sean válidos deben trabajar a una frecuencia de 25 kHz.

LED

- Aplicaciones con poca intensidad de luz.
- Tienen bajo coste y larga vida.
- Actualmente se construyen leds de alta luminosidad.

Láser

- Se usa luz estructurada para resaltar la tercera dimensión del objeto.
- Presenta el problema de la intensidad de la luz con forma gaussiana.

Iluminación por color o infrarojo

- Se emplea la iluminación de distintos colores para crear mayor contraste.

- Un espectro de emisión sobre una superficie de ese mismo color provoca una imagen más clara.
- Un espectro opuesto provoca una imagen más oscura.
- La iluminación IR (infrarojo), se puede emplear para reducir el contraste sobre espectros de colores diferentes.

2.4.6. Técnicas de iluminación

A su vez, existen distintas técnicas de iluminación, según la incidencia de la luz y su ángulo: frontal, lateral, contraluz, coaxial, difusa. Estas se aplicaran según las necesidades que tengamos, a la hora de detectar el objeto. [8]

Frontal o directa

- Para aplicaciones no críticas.
- Interesante para superficies sin brillos.
- Económico.
- Gran variedad de formas.

Frontal con anillo

- Reducción de sombras.
- Centrado de la luz en la imagen.
- Resaltamos características de la superficie.
- Los ring low-angle resalta los defectos en superficies brillantes.

Lateral (Dark Field)

- Iluminación angular para resaltar irregularidades en superficies.
- Enfatiza características en el plano de incidencia.

Contraluz

- Resalte de contornos.
- Localización y medición de objetos.
- Resalte estructura interna de objetos transparentes.
- El objeto de interés está entre la luz y la cámara.
- Se obtiene una imagen en blanco y negro sin matices.

Coaxial (DOAL)

- Iluminación difusa para superficies reflectantes.
- Excelente uniformidad respecto a la luz directa.

Luz difusa uniforme

- Distribución de luz perfecta.
- Aplicaciones con elementos altamente reflectantes de difícil eliminación.

Luz Estructurada. Láser

- Iluminación proyectada de forma controlada con el objetivo de extraer información dimensional.
- Ideal para aplicaciones de reconstrucción 3D, ingeniería inversa, medición y posicionado.

2.4.6. Estándares de Comunicación

Los estándares de comunicación nos marcan el tipo de relación y conexión entre la carama y el PC.

- Comunicación entre la cámara y el PC.
- Tarjetas de adquisición de vídeo.
- Son muchos los aspectos a tener en cuenta a la hora de realizar la elección adecuada: anchos de banda,
    - longitud máxima de cables,
    - entorno ambiental en el que implantarlo,
    - tipo de uso,
    - tiempo de vida,
    - desarrollo de software y otras muchas consideraciones.

Cabe destacar el uso de las interfaces digitales, que son las que se emplearan en este proyecto, concretamente los USB 2.0, con una capacidad de transmitir 480 Mb/s. [8]

2.5. Tecnología LiDAR

La palabra LiDAR proviene del acrónimo Light Detection and Ranging. Esta nueva tecnología otorga un potente sistema para la recolección de datos provista en información 2D y 3D, cuyo ámbito de estudio abarca topografía, teledetección de objetos, ingeniería inversa, diseño 3D etc. [9]

Dicha tecnología fue desarrollada en la década de los 70s, dentro de los programas de investigación de la Agencia Espacial Estadounidense, pero dicho estudio se frenó debido a sus limitadas posibilidades para la época y su elevado coste. Fue a partir de la década de los 80s y gracias a la introducción del GPS cuando su desarrollo comenzó a avanzar.

A pesar de sus múltiples usos y aplicaciones, en el actual proyecto nos centraremos en la detección superficies y distancias de objetos. [10]

Un medidor de distancia que emplee tecnología láser, funciona utilizando el tiempo que tarda un pulso de luz láser en reflejarse sobre la superficie de un blanco y volver al receptor. Este método de medición es conocido como "pulso". El tiempo de vuelo o TOF, es el tiempo transcurrido desde que el rayo se emite hasta que se recibe.

Los laser de esta tecnología suelen trabajar en rangos de longitudes de onda de 532 a 1550 nm y a una velocidad media de 300.000.000 m/s. En robótica, la mayoría de sensores, realizan un barrido 2D del entorno apuntando a varias direcciones gracias al uso de un espejo que gira y desvía la luz del láser. La precisión de los mismos en altura suele estar delimitada entre 3 y 10 cm, según las especificaciones técnicas de los distintos fabricantes y modelos. [12]

También podemos destacar que este tipo de sensores son muy precisos, rápidos y fiables, capaces de detectar desde unos pocos milímetros hasta varios metros, con un bajo error y una muy alta repetitividad. [11]

El uso de LiDAR requiere conocer las necesidades del proyecto al que se va a aplicar. Primeramente se deberá valorar el tipo de captura de los datos (aérea o terrestre), seguidamente de la naturaleza del objeto o superficie y las especificaciones técnicas del sensor que se emplea.

2.5.1. Terminología básica de LiDAR

Esta tecnología emplea unos términos específicos, pudiendo destacar los siguientes [10][12]:

- Pulso de repetición: se podría traducir al ritmo al que el láser emite los impulsos, la unidad de medida son los Kilohercios (KHz).
- Frecuencia de escaneo: A la vez que el láser emite los pulsos de repetición, el escáner va oscilando de delante hacia atrás.
- Ángulo de escaneo: es el angula desde el que se mueve el escáner de un extremo a otro, la unidad de medida son los grados (º)
- Altitud de vuelo: consideración de la mayor altura bajo unas condiciones dadas.
- Espaciamiento de la línea de vuelo: espacio comprendido entre la altura y el haz laser del sensor.
- Espaciamiento nominal entre puntos: distancia entre los puntos de impacto del LiDAR. Cuanto mayor sea el número de puntos, mayor definición obtendremos de los objetos y superficies.

- Banda: distancia de cubrimiento del LiDAR. Esta distancia depende del ángulo y la altura de vuelo. Cuando el sensor se emplea para la detección terrestre, emplearemos el término área de cubrimiento, en lugar de banda.
- Solape: suma de áreas redundantes.

2.6. ROS

Como el objetivo del proyecto es conseguir realizar el diseño y control de una aplicación para la detección y el seguimiento de objetos a través de una cámara web USB y un sensor de tecnología LiDAR, en este apartado hablaremos del entorno empleado para el sensor láser Hokuyo, ROS. [13]

Fig.12. *Imagen del logo de ROS*

2.6.1. Introducción de ROS

ROS (Robotic Operating System), que es un framework (marco) flexible para el desarrollo del software de robots para cualquier ámbito. A su vez ROS está destinado a servir como una plataforma de software común para las personas que están trabajando en la construcción y el uso de robots.

ROS ha experimentado en los últimos años un gran éxito en los profesionales y estudiantes dedicados a la robótica. Como muestra de ejemplo, ROS dispone de más de 2000 paquetes y librerías de software diseñado y mantenido por más de 600 participantes, en los que se estima que es compatible con aproximadamente el ochenta por ciento de los robots disponibles en el mercado.

ROS se compone de un conjunto de herramientas, librerías, módulos y servicios que simplifican la creación de aplicaciones más complejas para el funcionamiento de robots, tanto para ámbito profesional como de ocio. A su vez proporciona la seguridad de realizar el control de robots en diversas plataformas de forma compleja y robusta con relativa sencillez.

Por lo tanto su creación ha sido enfocada para el mundo real, donde se requerirá implementar en robots controles para la realización de tareas complejas y difíciles, lo cual resulta muy difícil si esto se trata de forma aislada y única para cada caso, ya sea

de forma individual, en laboratorios o en instituciones dedicadas a este ámbito. Como resultado ROS fue construido desde el principio para fomentar la colaboración para el desarrollo de un software para robots. Esta filosofía a unido a diversos grupos de expertos en robótica para producir un entorno fácil y cómodo de usar de forma general aplicado a todo tipo de robots, dentro del cual se incluyen multitud de características específicamente diseñadas para simplificar el diseño de los mismos a gran escala. [13]

2.6.2. Historia de ROS

ROS es un proyecto enorme con multitud de antecesores y contribuyentes. La necesidad de colaborar de forma abierta para formar un framework para robots llevo a muchas personas a formar una comunidad de investigación. Varios proyectos de la Universidad de Standford a mediados del 2000 encarnó la AI (inteligencia artificial), también conocida como STAIR (STandford AI Robot) y los robots personales, creando los prototipos de la casa, con un software flexible y dinámico. A su vez se ampliaron los recursos para ampliar los conceptos mucho más allá de la creación de dichos softwares.

El esfuerzo se vio impulsado por un sinnúmero de investigadores que contribuyo con su tiempo y experiencia para formar el núcleo de ROS y su software fundamental en paquetes y librerías. En todo momento, el software fue desarrollado de forma abierta, usando los permisos de licencia de BSD de código abierto, y poco a poco llegaron a ser ampliamente utilizados en la investigación robótica de esta comunidad.

Desde el principio ROS se ha ido desarrollando en varias instituciones y para múltiples robots. Al principio, esto parecía generar dolor de cabeza, ya que habría sido mucho más sencillo para todos los contribuyentes colocar el código en los mismos servidores. Irónicamente, a lo largo de los años, este supuesto problema ha emergido como una de las grandes fortalezas del sistema ROS: cualquier grupo puede comenzar su propio código de repositorio ROS en sus propios servidores, y que se mantengan el pleno control del mismo. Estos repositorios ya no necesitan a nadie para su permiso, si eligen para hacer su repositorio visible al público, pueden recibir el reconocimiento y el crédito que se merecen por sus logros a la vez que se benefician de la información técnica específica y mejorar, como en todos los proyectos, el software de código abierto.

Actualmente el sistema ROS se compone de decenas de miles de usuarios en todo el mundo, trabajando en diversos ámbitos, desde proyectos como hobby hasta proyectos profesionales. [13]

2.6.3. Filosofía de ROS

ROS ha sido diseñado y desarrollado para funcionar en Unix, siguiendo a su vez su misma filosofía. Esto tiende a hacer sentir ROS "natural" para los desarrolladores que vienen de Unix, pero a su vez algo "críptico" en un primer momento para aquellos que vienen de otros sistemas operativos como Windows o Mac.

ROS cuenta con varios aspectos filosóficos que se describen a continuación:

De igual a igual

El sistema ROS consiste en numerosos programas informáticos pequeños que se conectan uno a otro de manera continua e intercambiando mensajes. Estos mensajes se transmiten directamente de un programa a otro; no hay servicio de enrutamiento central. Aunque esto hace el subyacente más complejo, el resultado es un sistema que escala mejor conforme se incrementa la cantidad de datos.

Plurilingüe

Muchas de las tareas de software son más fáciles de lograr en "alta productividad" de secuencias de comandos en lenguajes como Python o Ruby. Sin embargo, hay ocasiones en que el rendimiento requiere dictar el uso de lenguajes más rápidos, como C++. Hay también varias razones por las que algunos programadores prefieren lenguajes como LISP o MATLAB. Se han librado un sinfín de batallas por correo electrónico para saber cuál sería el lenguaje más adecuado para una determinada tarea. Reconociendo que todas estas opiniones tienen mérito y valor, puntuando que los lenguajes tienen utilidades diferentes según los contextos y según el programador, ROS eligió un enfoque multilingüe.

El software modular de ROS puede ser escrito en cualquier idioma. En este momento existen bibliotecas para C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, Haskell, R, Julia y otros.

Las bibliotecas de cliente de ROS se comunican entre sí siguiendo una convención que describe como los mensajes se "aplanan" o son "serializados", antes de ser trasmitidos a través de la red.

Delgado

Las convenciones de ROS animan a los contribuyentes a crear bibliotecas independientes y luego envolver esas bibliotecas para que puedan enviar y recibir mensajes hacia y desde otros módulos de ROS. Esta capa adicional está destinada a permitir la reutilización del software fuera de ROS para otras aplicaciones, y se simplifica en gran medida la creación de pruebas automatizadas usando herramientas estándar de integración continua.

Libre y código abierto

El núcleo de ROS es liberado bajo la licencia BSD permisiva, lo que permite comercializar y no comercializar su uso. Ros pasa datos entre módulos mediante intercomunicación entre procesos (IPC), lo que significa que los sistemas construidos con ROS pueden otorgar licencias a varios de sus componentes. Los sistemas comerciales, por ejemplo, suelen tener varios módulos de código cerrado que se comunican con un gran número de módulos de código abierto. Los proyectos académicos y de ocio, son a menudo, fuente completamente abierta.

El desarrollo de productos comerciales a menudo se realiza por completo detrás de un firewall, siendo todos estos casos de uso, y más, comunes y perfectamente validos bajo la licencia de ROS. [13]

2.6.4. Instalación de ROS

En este punto se describirá como se debe de realizar la instalación de ROS en nuestra computadora.

Puesto que ROS a día de hoy solo tiene soporte en el sistema operativo Linux, realizaremos la instalación de ROS sobre la versión 14.04 de Ubuntu.

A continuación describiremos paso por paso la instalación de ROS sobre Linux, detallada en la página oficial de ROS http://wiki.ros.org/indigo/Installation/Ubuntu [14]

Configuración de los Repositorios de Ubuntu.

Añadir Repositorios en Ubuntu

1. Main Menu: System > Administration > Software Sources.
2. *Synaptic*: System > Administration >*Synaptic Package Manager*: >> Settings >> Repositories.
3. Main Menu: *Ubuntu Software Center*: >> Edit, Software Sources.

Tabla de Software de Ubuntu

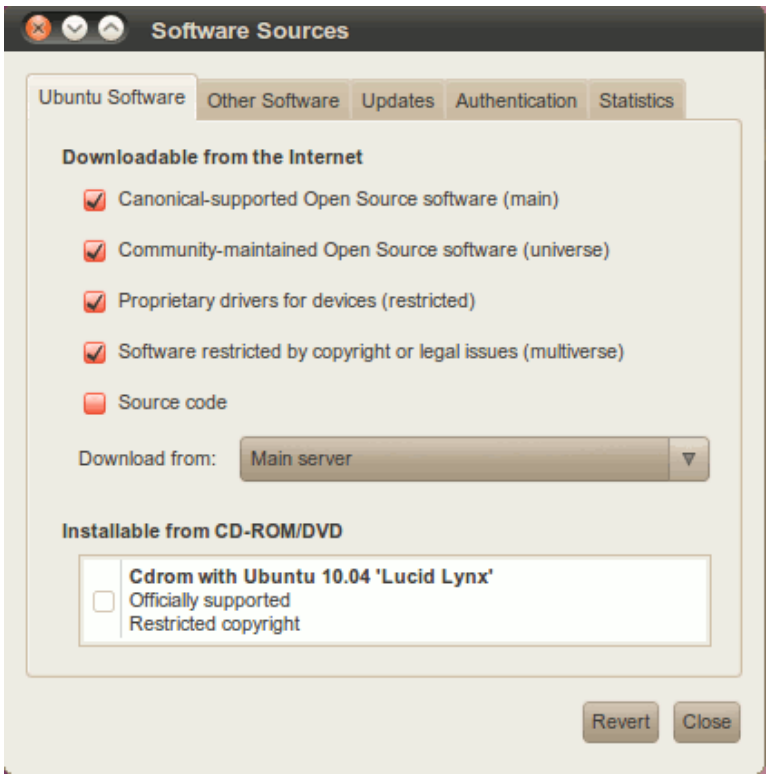


Fig.13. *Recursos del Software*

Añadir Repositorios al Software de Ubuntu

1. Los siguientes repositorios están disponibles por defecto:
    1. “canonical-supported Open Source software (main)”
    2. “proprietary drivers for devices (restricted)”
    3. “Source Code”
2. Seleccionar otros repositories para tener acceso a las propiedades de los drives, copyright, código fuente etc.
    1. “Community-maintained Open Source software – (universe)
    2. “Propietary drivers for devices (restricted)”
    3. “Software restricted by copyright or legal issues (multiverse)”
3. Seleccionar “Close” para guardar los cambios.

Descargar el servidor

1. “Main Server”, “server for…”, or “Other”

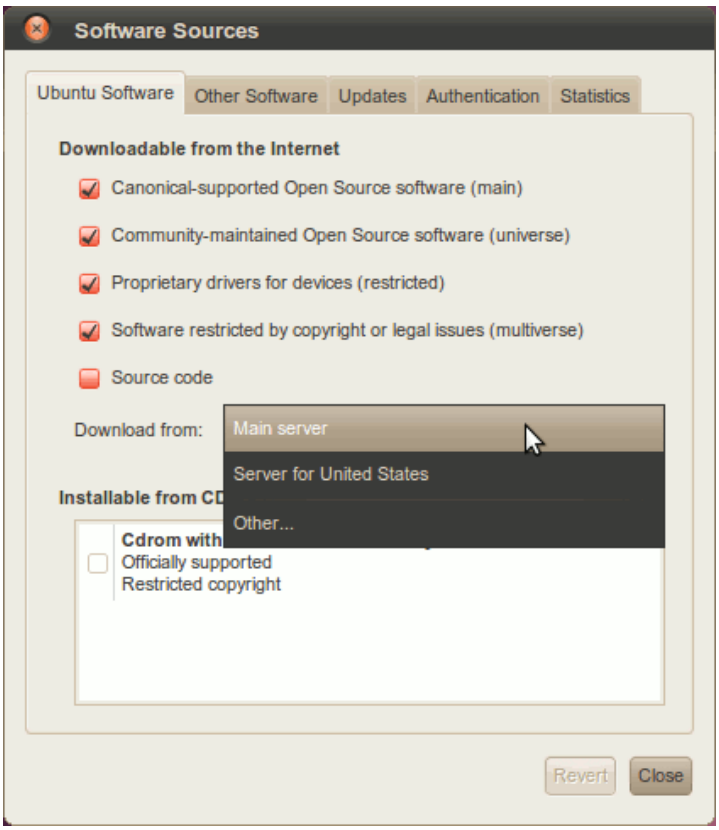


Fig.14. *Recursos del Software: Download Server*

2. Seleccionar “Other”

Primer seleccionamos “Select Best Server”y a continuación una vez nos aparezca, lo elegimos y clicamos en “Choose Server”.

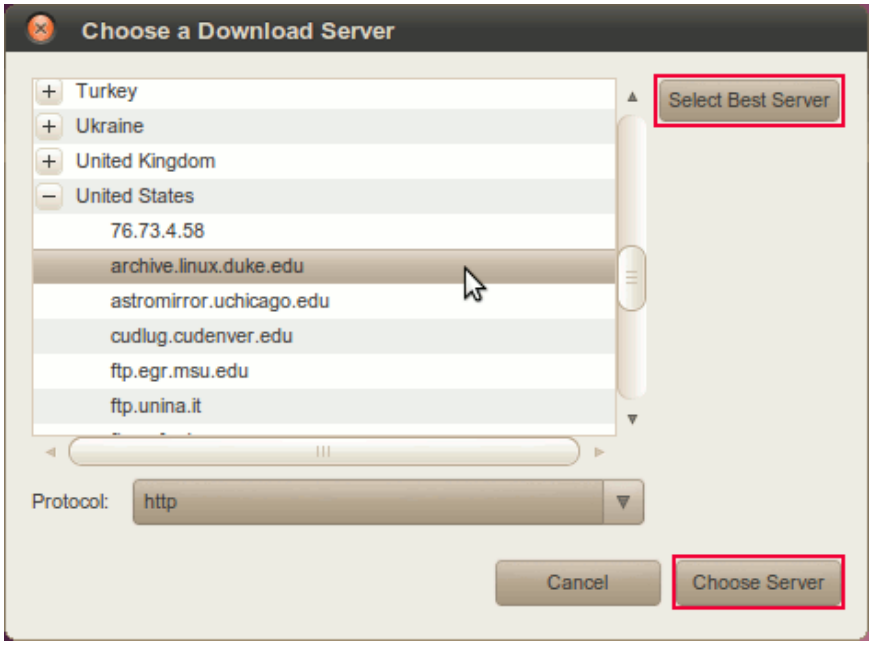


Fig.15. *Recursos del Software: Choose a Download Server*

CD-ROM/DVD

La instalación de CD-ROM debe ser seleccionado o no a través de la ventana mostrada a continuación:

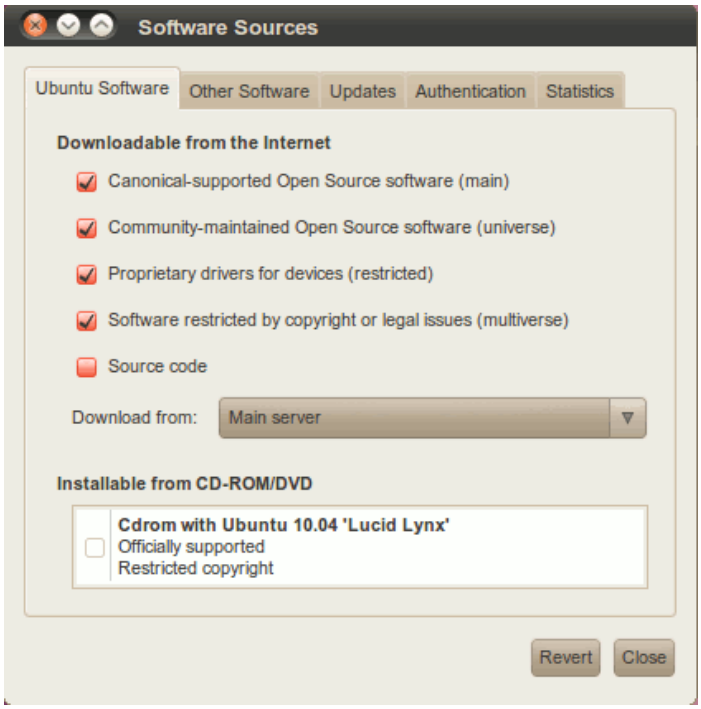


Fig.16. *Recursos del Software: Installable from CD-ROM/DVD*

<u>Tercera partición del Software</u>

Añadir la compartición canónica de repositorios

Marcaremos ambas pestañas mostradas en la siguiente imagen:

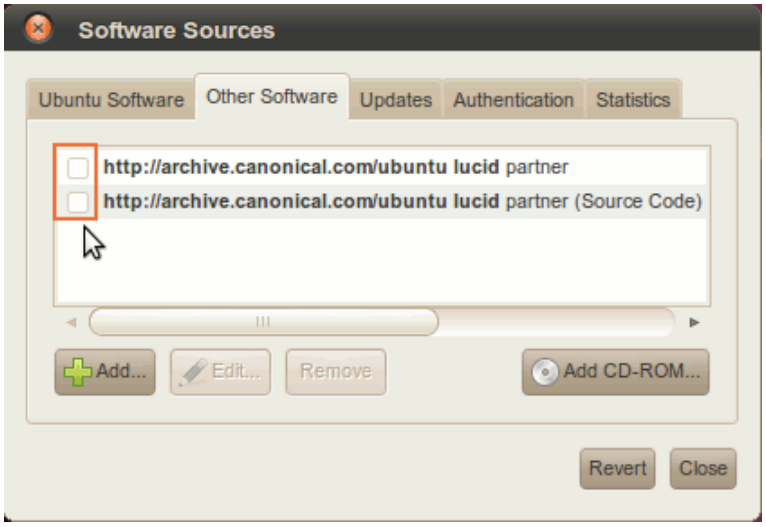


Fig.17. *Adición de la compartición canónica de repositorios*

Añadir PPAs

Para ello deberemos seguir la siguiente ruta: Ubuntu Software Centre > Edit > Software Sources > Other Software y clicar Add.

A continuación nos mostrara la siguiente imagen, donde deberemos escribir: ppa: [username]/[ppaname]

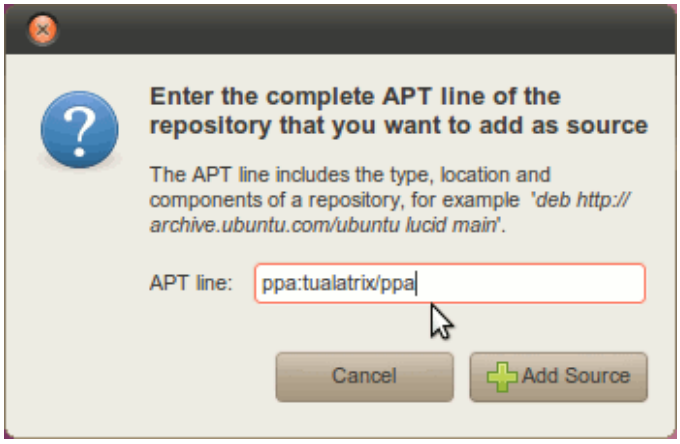


Fig.18. *Adición de PPAs*

Una vez se haya puesto, clicamos en Add Source y Close.

Añadir otros repositorios

Repetiremos como en l proceso anterior, clicamos en Add Source y añadimos:

```
deb http://ppa.launchpad.net/fta/ubuntu raring main
```

Editar Repositorios

Clicamos en el botón Edit y nos aparecerá la siguiente ventana.

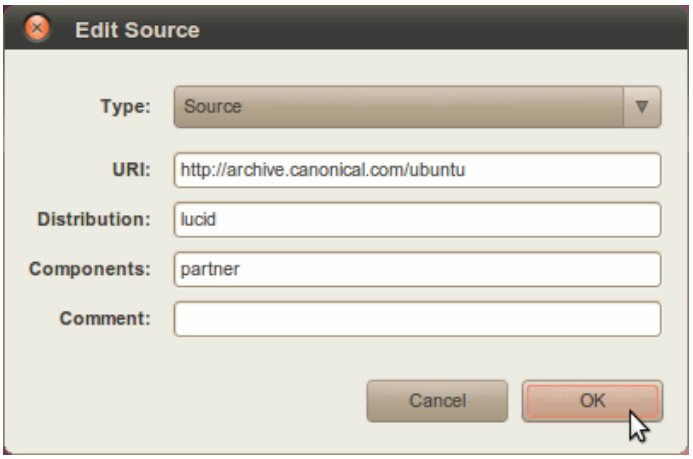


Fig.19. *Edición de repositorios*

1. cdrom
```
cdrom:[description_of_cd]/
```
2. ftp
```
ftp://ftp.domain.ext/path/to/repository
```
3. http
```
http://www.domain.ext/path/to/repository
```
4. smb (works only when the computer is connected to a Samba share)
```
smb://path/to/repository
```
5. nfs (works only if the computer is connected to a NFS share)
```
file://path/to/local/directory
```

Borrar e inhabilitar repositorios

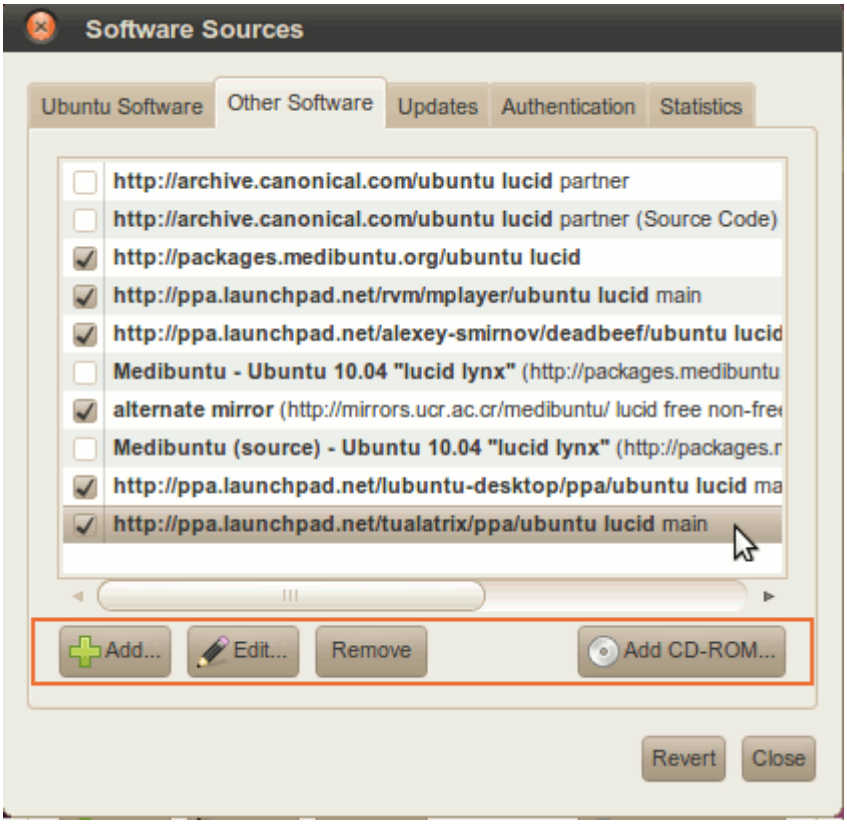


Fig.20. *Ventana para eliminar e inhabilitar repositorios*

Pestaña de Actualizaciones

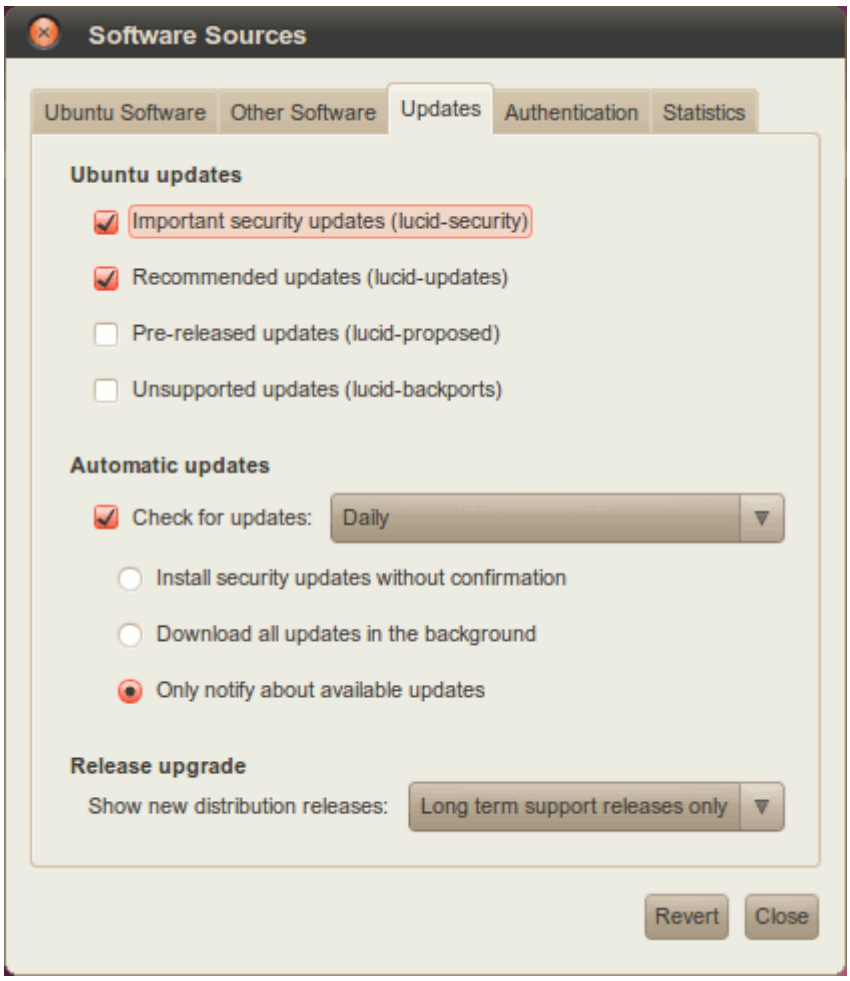


Fig.21. *Ventana de actualizaciones*

Pestaña de Autentificación

Las llaves de autentificación, son obtenidas de los repositorios del software. El mantenedor ofrecerá un lugar a la copia de autentificación en un servidor público, pero se necesita la llave “has” para encontrar dicho lugar. En la imagen a continuación se muestra un ejemplo:

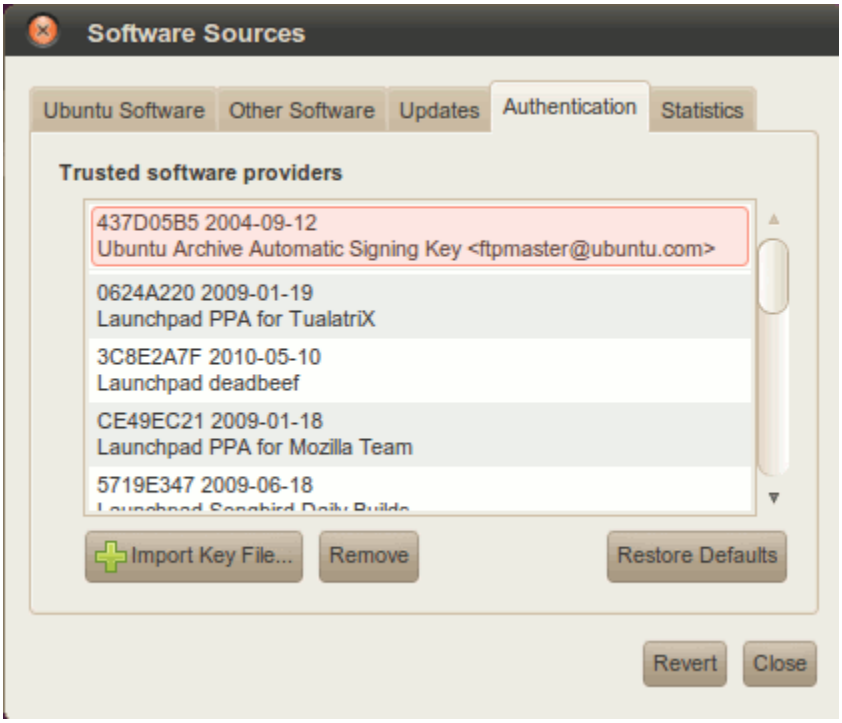


Fig.22. *Ventana de autentificación*

Una vez la llave "hash" es conocida, la llave se puede retribuir usando el siguiente comando:

```
gpg --keyserver [name of keyserver] --recv-keys [keyhash]
gpg --keyserver subkeys.pgp.net --recv-keys CE49EC21
gpg --export --armor CE49EC21 | sudo apt-key add -
```

Una vez se acaba con la configuración de los repositorios continuaremos con la instalación de ROS.

Preparación de sources.list

Prepararemos la computadora para aceptar el software de los paquetes de ROS

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

Preparación de las llaves

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --rec
v-key 0xB01FA116
```

Puesto que nuestra versión de Ubuntu es las 14.04, con el siguiente comando instalaremos algunas dependencias adicionales:

```
sudo apt-get install libgl1-mesa-dev-lts-utopic
```

Instalación

Primeramente, estar seguro de que el paquete Debian esta actualizado:

```
sudo apt-get update
```

Instalación total del escritorio (recomendado)

```
sudo apt-get install ros-indigo-desktop
```

ROS-Base

```
sudo apt-get install ros-indigo-ros-base
```

Paquetes Individuales

```
sudo apt-get install ros-indigo-PACKAGE
```

- Sustituir “PACKAGE” por el paquete que se desea instalar, este caso la “distro” será la de Indigo.

Para encontrar los paquetes disponibles usaremos el siguiente comando:

```
apt-cache search ros-indigo
apt-cache search ros-indigo
```

Inicializar rosdep

Antes de usar ROS, necesitaremos inicializar rosdep. rosdep nos permitirá instalar fácilmente en el sistema los recursos que queremos compilar y son requeridos para el núcleo de los componentes de ROS.

```
sudo rosdep init
rosdep update
```

Preparar el entorno

Es conveniente que las variables del entorno de ROS se añadan de forma automática a nuestra sesión cada vez que lo lanzamos.

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Si lo que queremos es solamente cambiar el entorno usaremos el siguiente comando:

```
source /opt/ros/indigo/setup.bash
```

Conseguir rosinstall

Para instalar esta herramienta en Ubuntu, usaremos el siguiente comando:

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

[14]

2.6.5. Instalación de RViz

Para la instalación de RViz, una vez tengamos ROS instalado, emplearemos el siguiente comando:

```
sudo apt-get install ros-indigo-rviz
```

Este programa complementario a ROS nos abrirá el control de sensor Hokuyo.

Una vez instalado ROS y el complemento de RVIz correctamente en la computadora, pasaremos a la instalación de ViSP. Debemos recordar que la "distro" empleada en este proyecto será la de Indigo. [15]

2.7. Introducción a VISP

ViSP es un software que permite realizar, diseñar e implementar un programa con una arquitectura modular basada en librerías, paquetes y módulos, con infinidad de posibilidades, como: detección de imágenes, control de robots, regulación de sensores, algoritmos de visión artificial etc.

En este caso nos centraremos en Blob, un módulo dedicado al tracking (rastreo) de imágenes, con el que posteriormente aplicaremos una cámara web (live camera) a modo de prueba antes de aplicar el sensor 2D.

A su vez se presentara el ViSP, una plataforma virtual modular, compuesta por paquetes y librerías usadas para la creación de prototipos y el desarrollo de aplicaciones utilizando el seguimiento visual. En este proyecto podemos destacar que ViSP es también, una librería complementaria de ROS. [16]

2.7.1. Instalación de VISP

La instalación de VISP se puede realizar a partir de los paquetes de la página web: https://visp.inria.fr/ , en installation from prebuilt packages on Linux Ubuntu/Debian o por Installation from source on Linux Ubuntu.

El primero de estos dos, por paquetes, es una instalación mucho más rápida y sencilla. El único problema puede ser que no aparezcan las carpetas: visp, visp-build y visp-images en HOME. Lo que se puede solucionar creándolas por uno mismo o añadiendo la instalación por código fuente.

La segunda., la instalación por código fuente, la cual es mucho más larga y lenta, a la vez que puede darnos problemas con el direccionamiento de directorios y librerías. Aun así no debería de dar ningún fallo si se hace correctamente y direccionándola de forma adecuada. [17]

Por sencillez elegiremos la instalación por paquetes binarios:

Instalación de paquetes precompilados en Linux Ubuntu/ Debian

```
sudo apt-get install libvisp-dev libvisp-doc visp-images-data
```

En caso de elegir la instalación de código Fuente para Linux, nos apoyaremos en la página: http://visp-doc.inria.fr/doxygen/visp-daily/tutorial-install-ubuntu.html

A continuación presentaremos los pasos que hemos seguido para su correcta instalación. [17]

Instalación desde el código Fuente en Linux Ubuntu

Instalación de prerrequisitos

- gcc 4.4.x or later. This can be installed with:

```
sudo apt-get install build-essential
```

- CMake 2.6 or higher that could be installed with:

```
sudo apt-get install cmake-curses-gui
```

Instalación de terceras particiones

- OpenCV

```
sudo apt-get install libopencv-dev
```

- libX11 para poder abrir una ventana para mostrar imágenes

```
sudo apt-get install libx11-dev
```

- lapack para beneficiarse de las capacidades matematicas optimizadas

```
sudo apt-get install liblapack-dev
```

- libdc 1394 para tomar imágenes de las cámaras firewire

```
sudo apt-get install libdc1394-22-dev
```

- libv4l para tomar imágenes de las cámaras USB o analógicos

```
sudo apt-get install libv4l-dev
```

- libxml2 para poder configurar los seguidores basados en modelos a partir de archivos XML

```
sudo apt-get install libxml2-dev
```

- Detección del código QR

```
sudo apt-get install libzbar-dev
```

Otras terceras particiones adicionales

- Coin, para ser capaz de soportar vrml modelo CAD usado por los rastreadores basados en modelos

```
sudo apt-get install libcoin80-dev
```

- libjpeg y libpng para apoyar jpeg y png, respectivamente (solo es útil si no está instalado OpenCV)

```
sudo apt-get install libjpeg-dev libpng12-dev
```

- ffmpeg, para ser capaz de leer o codificar secuencias de video comprimido (solo es útil si no está instalado OpenCV)

```
sudo apt-get install libswscale-dev libavutil-dev
libavformat-dev libavcodec-dev libbz2-dev libbz2-1.0
```

- Ogre 3D si se quiere hacer realidad aumentada o simulación

```
sudo apt-get install libogre-1.9-dev libois-dev
```

- Detección de códigos Datamatrix

```
sudo apt-get install libdmtx-dev
```

Obtención de código Fuente de ViSP

- Podemos descargar la última versión como una tarball. Una vez descargada, descomprimiremos el archive usando cualquiera de estos modos

```
tar xvzf visp-x.y.z.tar.gz
```

- También se puede descargar una instantánea diaria. Una vez descargada, descomprimiremos el archivo

```
tar xvzf visp-snapshot-yyyy-mm-dd.tar.gz
```

- O se obtiene la ViSP de vanguardia desde el repositorio GitHub con el siguiente comando

```
$ git clone https://github.com/lagadic/visp.git
```

Configuración de ViSP desde el código fuente

- Crear un primer directorio denotado <binary_dir> donde se quiere construir ViSP. Este directorio contendrá Makefiles generados, archivos objeto y bibliotecas de salida y binarios

```
cd $HOME; mkdir visp-build
```

- Introducir la <binary_dir> y configurar la construcción:

```
cd $HOME/visp-build

cmake ../visp
```

Construir ViSP desde el código Fuente

- Para construir ViSP:

```
make -j4
```

- Para instalar ViSP in /usr/local que es la localización de instalación por defecto:

```
sudo make install
```

- para instalar en ViSP/usr/local que es la ubicación de instalación predeterminada, procederemos con:

```
sudo apt-get install doxygen graphviz texlive-latex-base
```

Luego procederemos con:

```
make -j4 visp_doc
```

Instalación del conjunto de datos de ViSP

```
cd $HOME
unzip ViSP-images-x.y.z.zip
$ ls $HOME/ViSP-images
```

Como crear únicamente librerías de ViSP

```
$ make -j4 visp_modules
```

Como crear un módulo específico de ViSP

```
$ make -j4 visp_<module_name>
```

Una vez se sigan estos comandos por orden, obtendremos las tres carpetas principales de VISP: visp, visp-build y visp images; como se muestra en la siguiente imagen. [17]

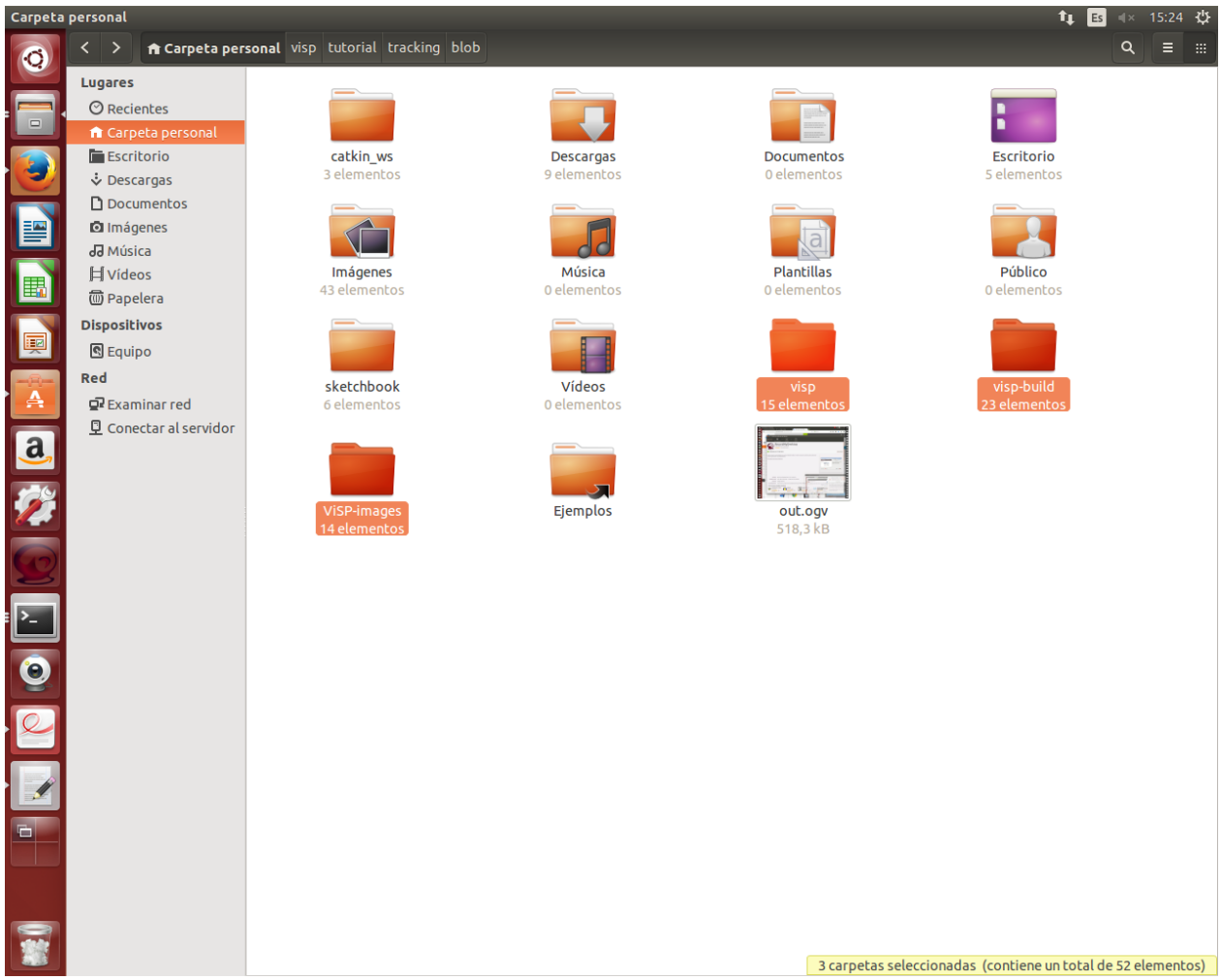


Fig.23. *HOME (carpeta personal) con ViSP instalado*

Una vez las tengamos, la programación se realizará a través de la carpeta visp y visp-buid haciendo uso de los tutoriales (tutorials), concretamente los de blob y keypoint.

De forma complementaria se unirá ViSP con ROS para realizar el auto tracker con un patrón de código QR.

2.8. Justificación del Proyecto

Ya que el objetivo es diseñar una aplicación para la detección y el seguimiento de objetos para una cámara UVC y a la par un sensor láser 2D para conocer la distancia a la que se encuentran, se ha escogido un sensor 2D Hokuyo URG-04LX-UG01 y una cámara UVC USB 2.0 comercial, en este caso una Logitech C920 C. El sensor Hokuyo permite detectar objetos en su línea de trazado laser con gran rapidez y eficacia hasta un radio de ocho metros, a su vez es mucho más rápido que un sensor 3D (que suelen ser empleados en este campo) en equidad de coste, puesto que la tercera dimensión demanda más cálculos, además de su gran modularidad, robustez y abundante material sobre el cual apoyarse para desarrollar el proyecto presente Este sensor

permite una buena relación calidad-precio, por lo cual es idóneo para su aplicación tanto en la industria automovilística donde se demandan grandes cantidades de productos, como en los robots industriales, pudiendo abaratar los costes sin por ello perder la eficiencia.

A su vez la utilización de la cámara C 920 C de Logitech, nos ayudará a detectar y seguir el contorno del objeto, a su vez marcando los puntos que definen el contorno de una figura u objeto para un reconocimiento mucho más rápido y eficiente, aunque no la distancia a la que se encuentra dicha cámara del objeto, ya que de esta parte se encarga el sensor Hokuyo.

3. Factores a considerar

3.1. Normativa

Directiva 2006/42/CE regula un nivel uniforme de seguridad para maquinas con el fin de habilitar el tráfico de mercancías y distribución gratuita en el Espacio Económico Europeo. Se aplica a fabricantes y distribuidores de maquinaria y dispositivos industriales. Esta directiva se centra en el hardware empleado, es decir, en el sensor laser 2D y en la cámara UVC 2.0 USB.

Por otro lado, encontramos el Estándar de Calidad ISO 9001-3, el cual regula el desarrollo de aplicaciones y software a nivel internacional y adoptado por más de 130 países.

4. Soluciones alternativas

El elemento que condiciona el tipo de sensor a utilizar es de tecnología LiDAR y la implementación en el campo de la robótica móvil, dictaminado por la electrónica y un uso modular de su software y control. Por lo tanto, se estudian alternativas en este aspecto del proyecto.

4.1. Sistema Operativo

Sistema Operativo Windows [18]

Ventajas

- Mayormente utilizado: entorno al 70% de las computadoras a nivel mundial emplean Windows como sistema operativo. Lo cual hace más fácil su conocimiento.
- Sencillo: dado a su interfaz con el usuario su uso es más sencillo, a la vez que al estar presente en la mayoría de equipos hace que su uso sea más cómodo y familiar.

Inconvenientes

- Poco eficiente: demanda más memoria que otros sistemas operativos
- Capacidad de cálculo: menor capacidad de cálculo que otros sistemas operativos ya que tiene un tiempo de ciclo mayor.

Sistema Operativo OS [19]

Ventajas

- Buena gestión de recursos: buena administración de la memoria RAM y espacio en disco.
- Buen rendimiento: capaz de administrar los recursos del hardware para desempeñar las tareas sin problemas.

Inconvenientes

- Usado para aplicaciones graficas mayoritariamente.

Sistema Operativo Linux [20]

Ventajas

- Gran potencia de cálculo.
- Gratuito y con una comunidad activa: se evitan los costes de instalarlo y a su vez dispondremos de actualizaciones y programas gratuitos, además de numerosos foros de ayuda y paquetes, librerías
- Total configuración por parte del usuario.
- Modularidad y gran cantidad de librerías disponibles.
- Actualmente es el único SO capaz de soportar ROS.

Inconvenientes

- Poco familiar: al no venir instalado en ninguna computadora de fábrica su uso es más remoto y su conocimiento más limitado.
- Mayor parte de la programación a través del Terminal.

Una vez estudiadas todas las posibles alternativas, la propuesta del uso de Linux sistema operativo y soporte de los programas, siendo el único que cumple con los requisitos de cálculo y modularidad, además de ser el único sistema operativo a día de hoy de soportar ROS. Por lo tanto es el elegido para la implementación del programa con el que se diseñará y controlará el sensor.

Emplearemos la versión 14.04 de Linux Ubuntu, no siendo la más actual, pero sobre la cual la instalación de ROS y ViSP no dan ningún tipo de problemas al estar más avanzado en términos de librerías, paquetes y módulos.

4.2. Programas

Una vez estudiado los diferentes sistemas operativos, se estudiarán los diferentes programas para el diseño y control del programa.

Robotic Operating System (ROS) [13]

Ventajas

- Más comúnmente usado: Empleado (cada vez más) en la mayoría de controles robóticos industriales y domésticos por los fabricantes.
- Comunidad activa y en continua actualización.
- Continuo crecimiento.
- Numerosas librerías y módulos, así como paquetes.
- Emplea multilenguaje.
- En unos años, probablemente sea el framework utilizado por todos (o casi todos) los fabricantes.

Inconvenientes

- Portable solo con Linux.
- Falta de módulos, librerías y paquetes para las versiones más modernas.

OROCOS [21]

Ventajas

- Librerías portables en C++.
- Control en tiempo real por Ethernet.
- Tracking en 3D.

Inconvenientes

- Menos usado
- Menos implementación modular y abierta.

Se ha escogido ROS para la implementación del control del sensor Hokuyo, dado a su mayor modularidad, mayor crecimiento y reconocimiento global y sus programas y librerías destinadas al control de robots.

Cabe destacar que para la cámara web USB 2.0 se empleará la librería ViSP, dicho programa (librería complementaria) no se ha considerado puesto que para la detección y seguimiento de objetos, este programa funciona perfectamente y existen librerías auxiliares y bibliotecas más que suficientes para su implementación, sin necesidad de aumentar la complicidad del proyecto.

4.3. Sensores

Se estudiarán los distintos sensores de tecnología 2D, para la detección y el seguimiento de las figuras y objetos, ya que este será el principal objetivo de dicho proyecto. [22]

Hokuyo UTM-30LX [23]

Ventajas

- Largo alcance de detección: 30 metros
- Buen ancho de ángulo: 270º
- Resistente para entornos abiertos.

Inconvenientes

- Relativamente pesado: 370 gramos.
- Caro: 4500-5000€
- Menor área de medida que el resto.

Hokuyo URG-04LX-UG01 [24]

Ventajas

- Alcance medio: 8 metros
- Ligero: 160 gramos.
- Buena precisión: 60-1000 mm ±30mm
- Buen ancho de ángulo: 240º
- Barato: 1000-1200€

- Bajo consumo de corriente: 500mA o menos (800 mA al encenderlo)

Inconvenientes

- Aparentemente ninguno

Hokuyo URG-04LX [25]

Ventajas

- Ligero: 160 gramos.
- Buena precisión:60-1000mm ±30mm
- Bajo consumo de corriente: 500 mA o menos (800 mA al encenderlo)

Inconvenientes

- Precio medio: 1800-2000€
- Corto rango de detección: 4 metros.

Cabe destacar que los tres sensores candidatos cumplen las mismas especificaciones técnicas en cuanto a resistencia de temperatura (-10ºC a 50ºC y hasta un 85% de humedad) e impactos (196m/s²), así como un consumo contenido en los tres casos y la necesidad de una tensión continua de 5 V, finalmente los tres sensores presentan un ruido de 25 dB o menor.

A pesar de las similitudes el sensor elegido para el proyecto ha sido el Hokuyo URG-04LX-UG01 debido a su menor coste, un alcance medio suficiente para la detección de objetos en el móvil y un buen ángulo de trabajo para la detección. [22]

5. Descripción detallada de la solución

5.1 Sistema y subsistemas

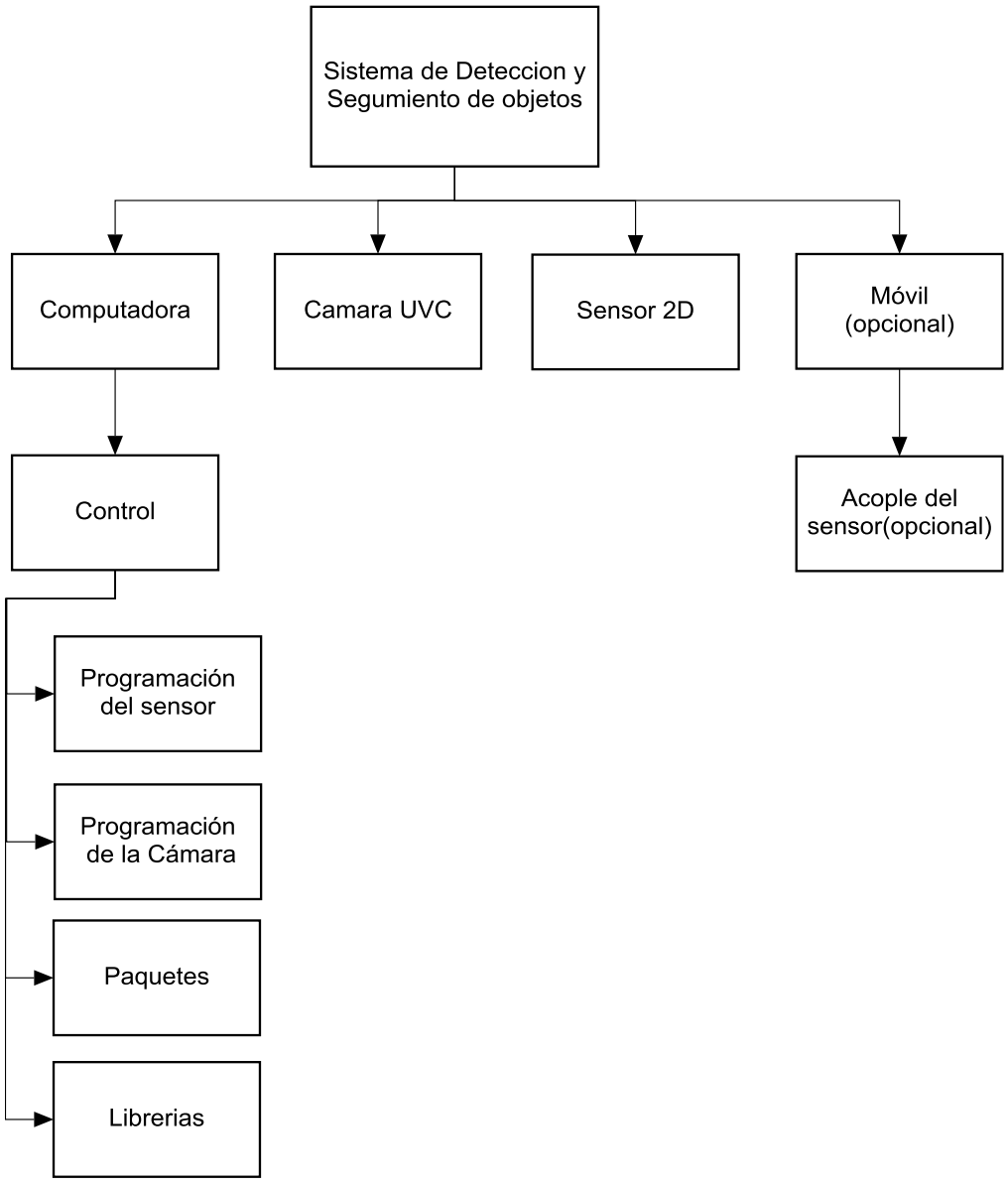


Fig.24. *Sistema detección y seguimiento de objetos*

5.2. Computadora

Se comenzará con la implementación del sistema operativo Linux en el computador a usar (versión 14.04 Ubuntu) según se valoró en las posibles alternativas, después se realizará la instalación de ROS y ViSP con sus respectivas librerías para el funcionamiento correcto del software de la aplicación para el sensor y la cámara.

Como se comentó en el apartado 2.6.4 Instalación de ROS y en el 2.7.1 Instalación de VISP, los instalaremos según los tutoriales de la página web de ROS y VISP.

La instalación de ROS nos generara la carpeta catkin_ws, dentro de la cual se ubicarán las carpetas dentro de las cuales se hallará tanto la aplicación de detección y seguimiento para la cámara UVC como la de detección del sensor láser y los ficheros auxiliares de drivers, librerías y paquetes.

La instalación de VISP, nos generará tres carpetas principales en el HOME o Carpeta Personal: visp, visp-build y visp-images; dentro de las cuales se encuentran los tutoriales, paquetes de librerías y ejecutables y finalmente las imágenes, los paquetes necesarios para hacer las detecciones y seguimientos, respectivamente, que es en lo que se basa dicho proyecto.

El principal uso de la carpeta *visp* y *visp-build*, dentro de las cuales se haya *tutorials* y dentro de la misma *tracking* en la cual usaremos *blob*, *keypoint* (idéntico para ambas carpetas). Aquí haremos uso de los tutoriales para el desarrollo de nuestra aplicación y crear el control de detección y seguimiento de imágenes.

Podemos señalar que para este proyecto se descargó la carpeta *visp-master* y desde la cual se desarrolló el proyecto. El enlace de descarga es:

https://github.com/lagadic/visp.git

Esto se ha realizado puesto que es idéntica a la carpeta *visp* generada en la instalación de ViSP pero actualizada y evitando los fallos de conexión entre módulos y librerías. A su vez se empleará la carpeta *vision-visp* dentro del *catkin_ws/src* donde encontramos el *visp-auto-tracker*.

5.2.1. Control

El control de la aplicación para el sistema de detección y seguimiento de objetos se divide en dos controles distintos. Un control para detectar la distancia a la que se encuentra el objeto mediante el sensor laser Hokuyo y un control para la detección del contorno de un objeto por puntos y su seguimiento mediante una cámara UVC.

En los próximos apartados hablaremos de los controles de ambos dispositivos, así como su funcionamiento y su código.

5.2.1.1. Programación del Sensor

El sensor Hokuyo URG-04LX-UG01 de tecnología LiDAR 2D, es ejecutado a través de ROS.

Primeramente descargaremos el paquete necesario del sensor en la siguiente página:

https://github.com/gbiggs/hokuyoaist

Una vez nos hallemos en dicha página, observamos que se trata del sensor URG-04LX-UG01 y comprobamos en el README para asegurarnos. [26]

```
1  HokuyoAIST
2  ==========
3
4  Hokuyo range sensor driver.
5
6  This library provides a driver for Hokuyo laser scanner devices using
7  the SCIP protocol version 1 or 2. It has been tested with the Hokuyo
8  URG-04LX, UBG-04LX, UHG-08LX, UTM-30LX and UXM-30LX-E but it should work
9  with any scanner that conforms to these protocol versions, including the
10 URG-04LX-F01 and the URG-04LX-UG01 (Simple-URG).
11
12 Documentation is available online at http://gbiggs.github.com/hokuyoaist/
13
14 This software is developed at the National Institute of Advanced
15 Industrial Science and Technology. Approval number H22PRO-1195. This
16 software is licensed under the Lesser General Public License. See
17 COPYING.LESSER.
18
```

Fig.25. *README HokuyoAIST*

Una vez se haya comprobado satisfactoriamente procedemos con la descarga del paquete. Esta descarga se hará como muestra la imagen:

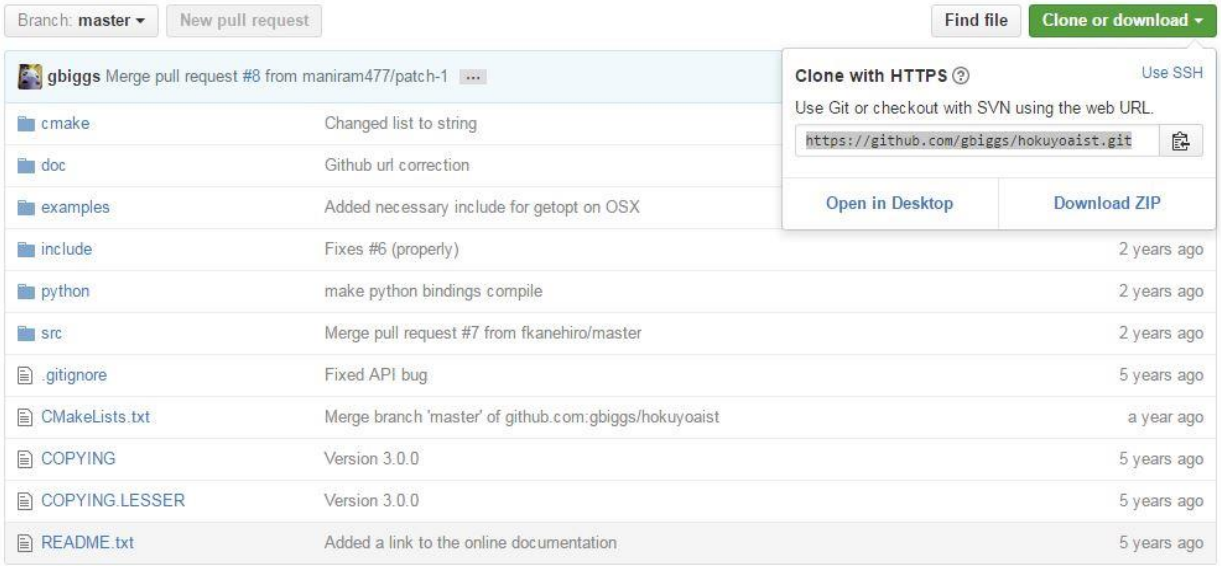


Fig.26. *Instalación por GitHub*

Para ello copiaremos dicho enlace (https://github.com/gbiggs/hokuyoaist.git), iremos al terminal y haremos:

```
git clone "https://github.com/gbiggs/hokuyoaist.git"
```

Con lo que el paquete quedara en descargas, una vez descargado se descomprimirá dentro de la carpeta catkin_ws dentro de HOME.

Una vez terminado este paso nos iremos a ROS para realizar el tutorial de instalación y prueba del sensor.

Para ello lo ejecutaremos a través del terminal de Ubuntu, siguiendo la recomendación de la ROS Wiki, en la siguiente página:

http://wiki.ros.org/hokuyo_node/Tutorials/UsingTheHokuyoNode

Como bien se describe, en este tutorial (How to use Hokuyo Laser Scanners with the hokuyo_node) [27]

Descripción: Este tutorial es una introducción a usar el escáner laser Hokuyo, conectado a nuestro escritorio. Después de leer este tutorial, debes de ser capaz de usar el hokuyo_node para mostrar las capturas del láser por pantalla.

Install (Instalación)

Primero hay que obtener el paquete hokuyo_node, lo podemos instalar usando

```
$ sudo apt-get install ros-%DISTro%-hokuyo-node
```

En este caso nuestra distro será "indigo".

Encendido y Conexión

Estar seguro de que el USB está conectado al computador y las luces están encendidas.

Configuración del Hokuyo

Comenzamos hacienda una lista de los permisos del Hokuyo:

```
$ ls -l /dev/ttyACM0
```

Veremos algo similar a:

- `crw-rw-XX- 1 root dialout 166, 0 2009-10-27 14:18 /dev/ttyACM0`

- Si XX es rw, el láser se ha configurado correctamente.
- Si XX es--, el láser no se ha configurado correctamente y necesitaremos conectarlo:

```
$ sudo chmod a+rw /dev/ttyACM0
```

Empezar un roscore

Para que el hokuyo_node funcione correctamente, el roscore debe de estar en ejecución. En un nuevo terminal:

```
$ roscore
```

Configuración de Parámetros

Esta opción va a acelerar la puesta en marcha del driver, pero dará lugar a una menor precisión a las marcas de tiempo:

```
$ rosparam set hokuyo_node/calibrate_time false
```

Si nuestro Hokuyo no está por defecto en dev/ttyACM0, tenemos que indicar donde se encuentra:

```
$ rosparam set hokuyo_node/port /dev/ttyACM0
```

Poner en marcha el hokuyo_node

En un Nuevo terminal, ejecutaremos el hokuyo_node:

```
$ rosrun hokuyo_node hokuyo_node
```

Viendo algo similar a:

```
[ INFO] 1256687975.743438000: Connected to device with ID: H0807
344
```

Vista de datos

Para ver que todo está funcionando y los datos se publican en ROS, ejecutaremos este código en un Nuevo terminal:

```
$ rosrun rviz rviz -d `rospack find hokuyo_node`/hokuyo_test.vcg
```

En este último paso es posible que de fallos diciendo que las libreras `.rviz` son antiguas y se han actualizado a YAML, puesto que para la toma de datos dicho sensor debe ser abierto con RVIZ de ROS. Para ello se ha implementado la siguiente ruta:

Para acceder a ella desde el terminal de Ubuntu deberemos seguir los siguientes pasos:

1. Introducimos el comando `roscore` en un terminal.

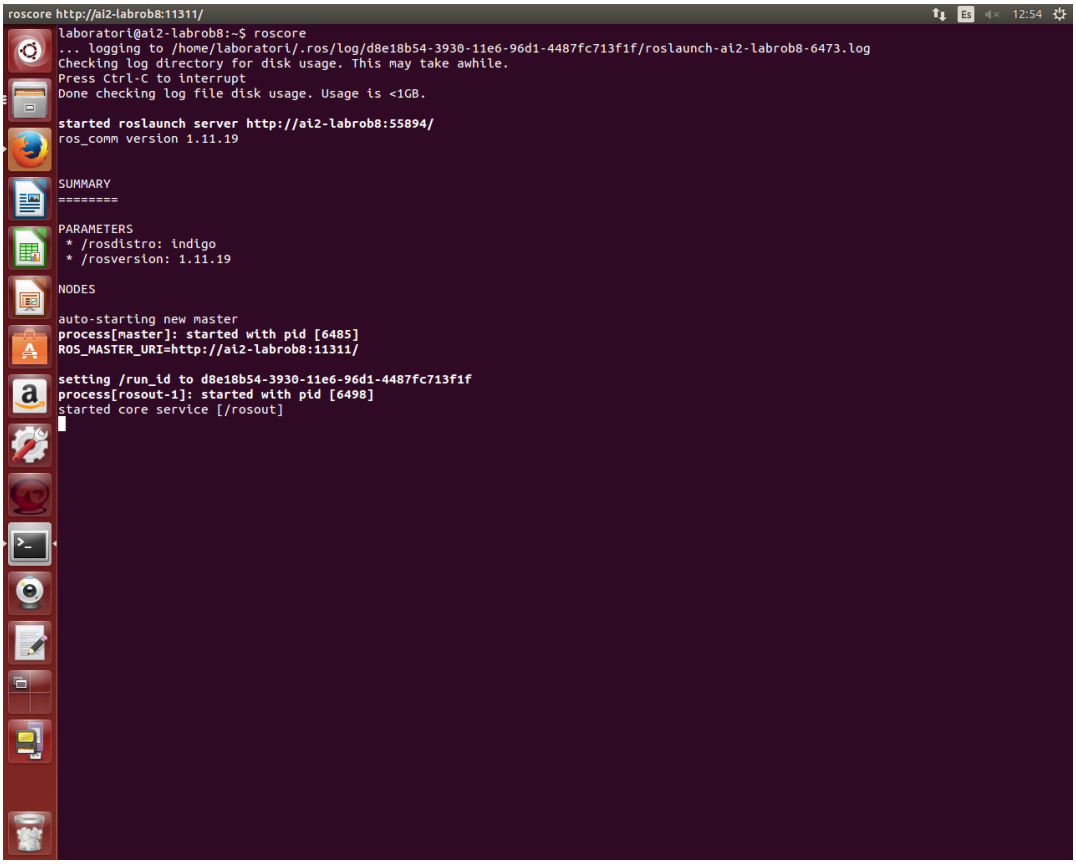


Fig.27. *Roscore para sensor*

2. Sin cerrar el terminal anterior, realizamos:

   `roslaunch display.launch`

   Un posible fallo es un error de conexión del puerto, para asegurarse que está conectado al puerto USB correcto, deberemos ubicarlo en `/catkin_ws`

   Luego una vez ubicado realizaremos `ls/dev/ttyACM*`

   Reconfiguramos dicha dirección en el fichero `display.lauch` y ya estaría solventado el error.

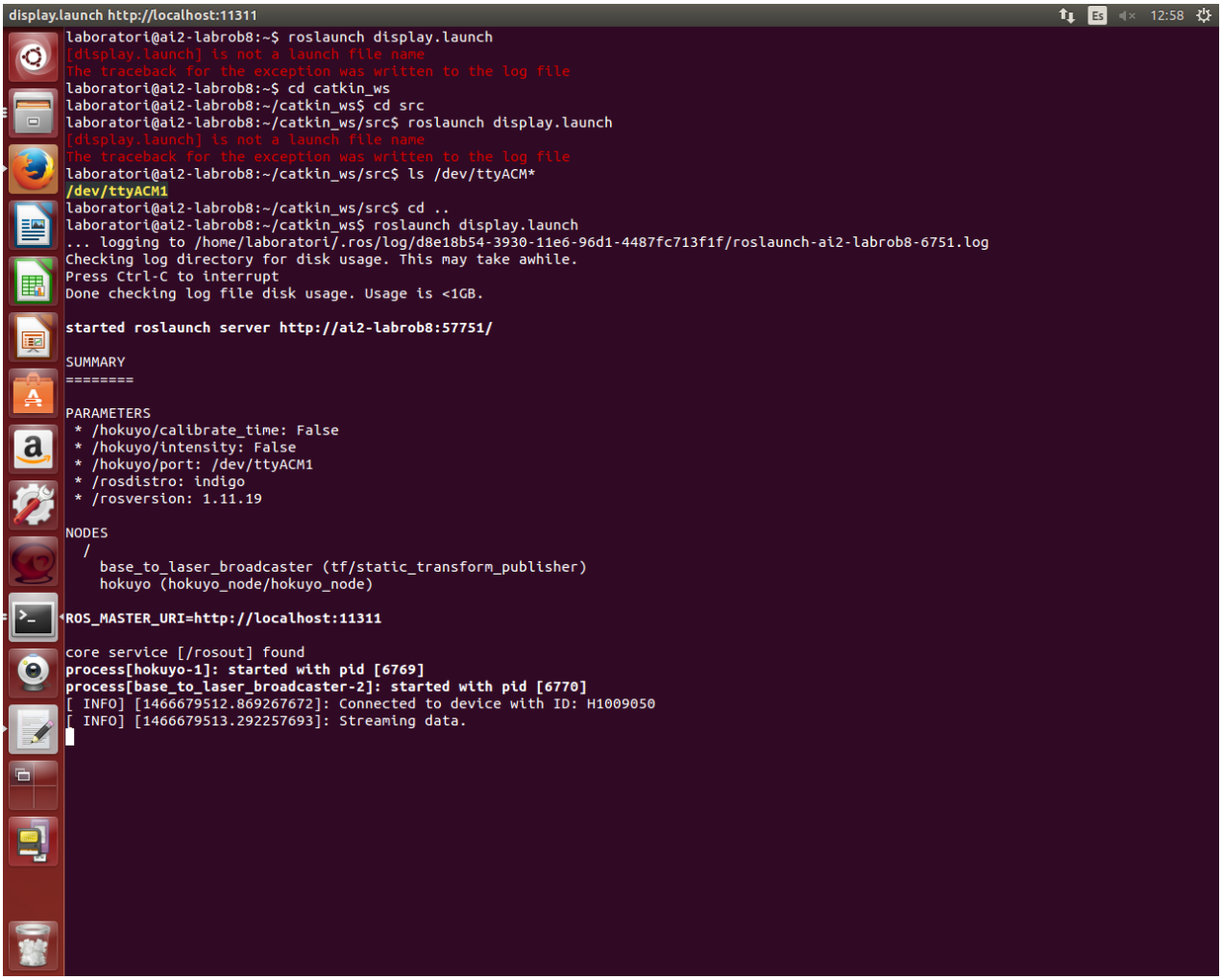


Fig.28. *Roslaunch y prueba de errores*

3. Por último, realizamos el comado `rviz` con lo que nos aparecerá RViz mostrando el menú principal con el sensor conectado.

   Una vez abierto nos aparecerá la siguiente ventana.

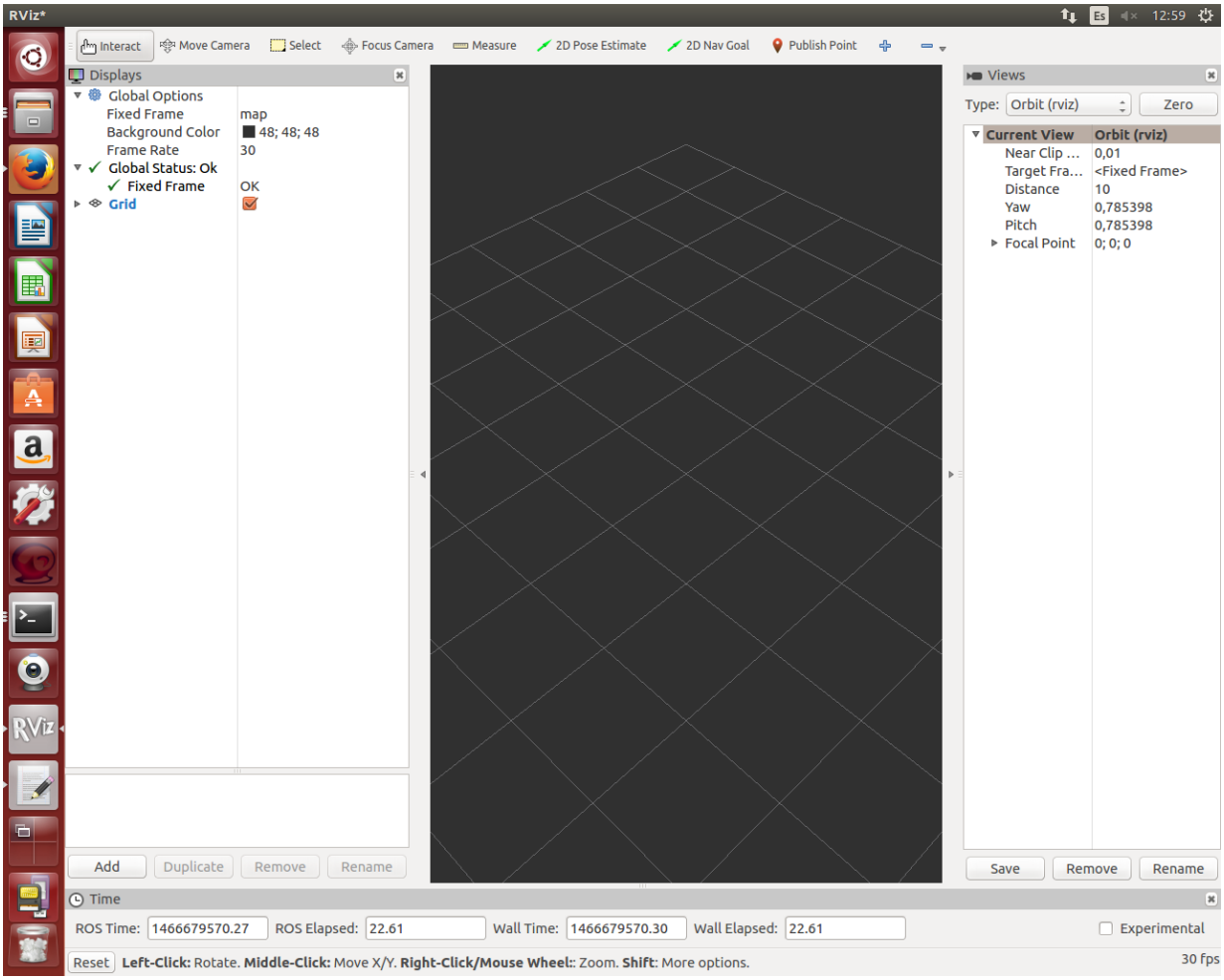


Fig.29. *Entorno RViz sin programar*

4. Desde RViz abierto, iremos a la pestaña superior y realizaremos la siguiente ruta: `File → Open Config → load Escritorio → Laser display.rviz`

   Nos aparecerá el entorno de trabajo del sensor Hokuyo.

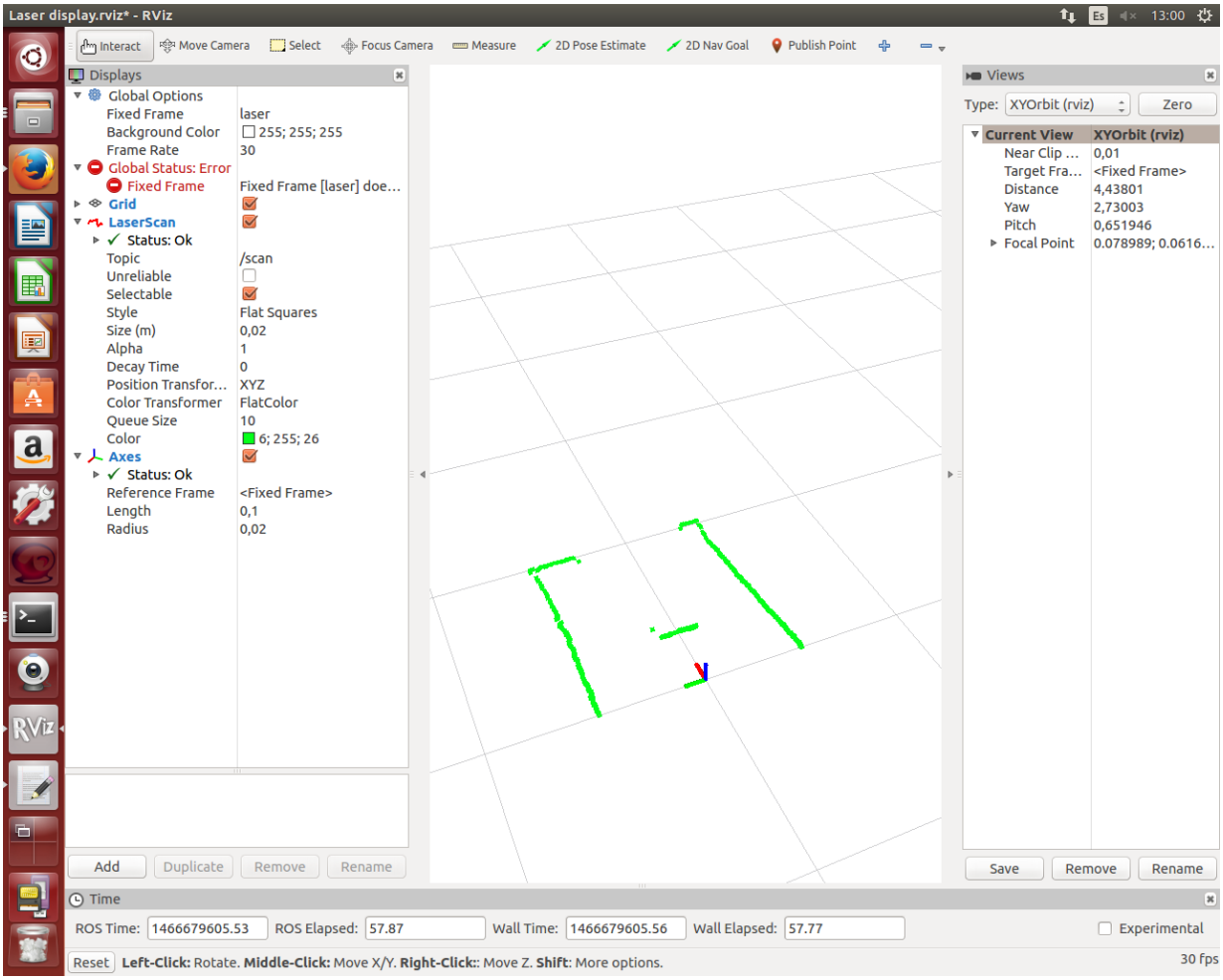


Fig.30. *RViz programado con el control del sensor Hokuyo*

Otra posible opción es calibrarlo a mano. Una vez abierto el RViz, deberemos de calibrar el láser para que nos aparezca la detección de los objetos. Para ello realizaremos los siguientes cambios:

- Fixed name→laser
- Topic → /scan
- Flat Color →(6,255,26) (sería verde)

Dando como resultado final a la siguiente pantalla (idéntica a anterior):

Por último presentamos el fichero `laser display.rviz` y `display.launch`, los cuales al cargarlos nos aparecerá el entorno programado del sensor láser y lanzará la aplicación del sensor.

laser display.rviz

```
Panels:
  - Class: rviz/Displays
      Help Height: 78
      Name: Displays
      Property Tree Widget:
      Expanded:
      - /Global Options1
      - /Status1
      - /LaserScan1
      - /Axes1
      Splitter Ratio: 0.5
      Tree Height: 719
  - Class: rviz/Selection
      Name: Selection
  - Class: rviz/Tool Properties
      Expanded:
      - /2D Pose Estimate1
      - /2D Nav Goal1
      - /Publish Point1
      Name: Tool Properties
      Splitter Ratio: 0.588679
  - Class: rviz/Views
      Expanded:
      - /Current View1
      Name: Views
      Splitter Ratio: 0.5
```

```
    - Class: rviz/Time
      Experimental: false
      Name: Time
      SyncMode: 0
      SyncSource: LaserScan
  Visualization Manager:
    Class: ""
    Displays:
      - Alpha: 0.5
      Cell Size: 1
      Class: rviz/Grid
      Color: 160; 160; 164
      Enabled: true
      Line Style:
      Line Width: 0.03
      Value: Lines
      Name: Grid
      Normal Cell Count: 0
      Offset:
      X: 0
      Y: 0
      Z: 0
      Plane: XY
      Plane Cell Count: 10
      Reference Frame: <Fixed Frame>
      Value: true
      - Alpha: 1
      Autocompute Intensity Bounds: true
      Autocompute Value Bounds:
```

```
Max Value: 10
Min Value: -10
Value: true
Axis: Z
Channel Name: intensity
Class: rviz/LaserScan
Color: 6; 255; 26
Color Transformer: FlatColor
Decay Time: 0
Enabled: true
Invert Rainbow: false
Max Color: 255; 255; 255
Max Intensity: 4096
Min Color: 0; 0; 0
Min Intensity: 0
Name: LaserScan
Position Transformer: XYZ
Queue Size: 10
Selectable: true
Size (Pixels): 3
Size (m): 0.02
Style: Flat Squares
Topic: /scan
Unreliable: false
Use Fixed Frame: true
Use rainbow: true
Value: true
- Class: rviz/Axes
Enabled: true
```

```
    Length: 0.1
    Name: Axes
    Radius: 0.02
    Reference Frame: <Fixed Frame>
    Value: true
Enabled: true
Global Options:
    Background Color: 255; 255; 255
    Fixed Frame: laser
    Frame Rate: 30
Name: root
Tools:
    - Class: rviz/Interact
    Hide Inactive Objects: true
    - Class: rviz/MoveCamera
    - Class: rviz/Select
    - Class: rviz/FocusCamera
    - Class: rviz/Measure
    - Class: rviz/SetInitialPose
    Topic: /initialpose
    - Class: rviz/SetGoal
    Topic: /move_base_simple/goal
    - Class: rviz/PublishPoint
    Single click: true
    Topic: /clicked_point
Value: true
Views:
    Current:
    Class: rviz/XYOrbit
```

```
      Distance: 1.47098
      Enable Stereo Rendering:
      Stereo Eye Separation: 0.06
      Stereo Focal Distance: 1
      Swap Stereo Eyes: false
      Value: false
      Focal Point:
      X: 0.0789895
      Y: 0.0616411
      Z: 9.53674e-07
      Name: Current View
      Near Clip Distance: 0.01
      Pitch: 0.886946
      Target Frame: <Fixed Frame>
      Value: XYOrbit (rviz)
      Yaw: 1.44503
      Saved: ~
Window Geometry:
  Displays:
      collapsed: false
  Height: 1000
  Hide Left Dock: false
  Hide Right Dock: false
  QMainWindow State:
000000ff00000000fd0000000400000000000016a0000035efc0200000008fb
0000001200530065006c00650063007400690006f006e00000001e10000009b00
0000064000fffffffb0000001e0054006f006f006c00200050007200 6f00700065
00720074006900650073020000001ed000001df00000185000000a3fb00000012
005600690065007700730020005400 6f006f02000001df000002110000018500
000122fb000000200054006f006f006c00200050007200 6f0070006500720074
00690065007300320300000288000011d000002210000017afb000000100044
006900730070006c00610079007301000000280000035e000000dd00ffffffb
```

```
0000002000730065006c00650063007400690006f006e00200062007500660066
00650072020000013800000aa0000023a00000294fb0000001400570069006 4
00650053007400650072006500 6f02000000e6000000d2000003ee0000030bfb
0000000c004b0069006e00650063007402000001860000010600000 30c000002
6100000001000001 0f0000035efc0200000003fb0000001e0054006f006f006c
00200050007200 6f00700065007200740069006500730100000041000000 7800
00000000000000fb0000000a00560069006500770073010000002800000 35e00
0000b000ffffffb0000001200530065006c00650063007400 69006f006e0100
00025a000000b2000000000000000000000002000004900000 00a9fc01000000
01fb0000000a00560069006500770073030000004e00000080000002e1000001
97000000030000 04bf0000003efc0100000002fb0000000800540069006d0065
0100000000000004bf000002f600ffffffb0000000800540069006d00650100
0000000000045000000000000000000000023a0000035e00000040000000400
0000080000000 8fc00000001000000020000000100000 00a0054006f006f006c
00730100000000ffffffff0000000000000000
```

```
  Selection:
     collapsed: false
  Time:
     collapsed: false
  Tool Properties:
     collapsed: false
  Views:
     collapsed: false
  Width: 1215
  X: 55
  Y: -14
```

display.launch

```
<!--

Este launch muestra como arrancar el nodo del hokuyo.
```

```
Debes cambiar el puerto del sensor laser para que pueda ser
detectado:

/dev/ttyACM1.(aquí cambiaremos la configuración del Puerto USB,
según necesitemos empleando ls/dev/ttyACM*)


Si tenemos rviz, podemos descomentarlo para visualizer el nodo
por RViz.

laser data.


También podemos usar el rostopic para ver los datos:

rostopic echo /scan


-->


<launch>

  <node name="hokuyo" pkg="hokuyo_node" type="hokuyo_node"
respawn="false" output="screen">


     <!-- Starts up faster, but timestamps will be inaccurate. -
->
     <param name="calibrate_time" type="bool" value="false"/>


     <!-- Set the port to connect to here -->

     <param name="port" type="string" value="/dev/ttyACM1"/>


     <param name="intensity" type="bool" value="false"/>

  </node>


<node pkg="tf" type="static_transform_publisher"
name="base_to_laser_broadcaster" args="0 0 0 0 0 0 map scan 100"
/>
```

```
  <!--
  <node name="rviz" pkg="rviz" type="rviz" respawn="false"
output="screen" args="-d $(find hokuyo_node)/hokuyo_test.vcg"/>
  -->


</launch>
```

5.2.1.2. Programación de la Cámara

Una vez instalado Linux y ViSP en la computadora, teniendo las carpetas adecuadas y actualizadas con la versión 3.0.0 y versión 3.0.1 (visp-master) [28], procederemos con la realización de los tutoriales, en este caso comenzaremos con el *Getting Started: How to create and build a CMake project that uses ViSP on Unix or Windows* con el cual abriremos imágenes e implementaremos el cmake, en este caso un CMakeLists.txt para usar ViSP en Linux a su vez con el comando *./tutorial-(llamada) (imagen.ppm)* hacemos llamadas a imágenes que se mostraran por pantalla. En otro orden cabe destacar la compilación de los *.cpp* para la realización de los ejecutables con *cmake .* en primer lugar y posteriormente con *make*. Una vez se creen los ejecutables (del mismo modo para cualquier tutorial) se hace la llamada *./tutorial-(llamada) (imagen.ppm/video.)* y mostrará la imagen o video, según demandemos. [29]

Dicho tutorial es necesario para la realización de ejecutables con los que ejecutaremos los programas y a su vez direccionar las llamadas a imágenes y videos. Su ruta es:

~visp-master/tutorial/image

Crear el archivo CMake

Ahora que tenemos creado el archivo CMakeLists.txt, será algo parecido a esto:

```
project(tutorial-image)
cmake_minimum_required(VERSION 2.8)
find_package(VISP REQUIRED)
include_directories(${VISP_INCLUDE_DIRS})
add_executable(tutorial-viewer tutorial-viewer.cpp)
target_link_libraries(tutorial-viewer
          ${VISP_LIBRARIES})
```

El find_package() CMake commando busca el archivo VISPConfig.cmake que se corresponde a las variables:

VISP_INCLUDE_DIRS: ViSP y la localización de las terceras particiones (third-party).

VISP_LIBRARIES: ViSP y el nombre y la localización de las terceras librerías (third-libraries).

El CMakeLists.txt previo puede aparecer con la siguiente forma:

```
project(tutorial-image)
cmake_minimum_required(VERSION 2.8)
find_package(VISP REQUIRED)
if(VISP_FOUND)
 include(${VISP_USE_FILE})
endif(VISP_FOUND)
add_executable(tutorial-viewer tutorial-
          viewer.cpp)
```

Configurar tu proyecto

Como se comentó anteriormente usaremos el commando *cmake .*

Por defecto el cmake busca el archive VISPConfig.cmake en /usr/share o /usr/local/share. Si ViSP no fue instalado en /usr o /usr/local, es posible que se muestre el siguiente error.

```
CMake Error at CMakeLists.txt:5 (find_package):
     Could not find module FindVISP.cmake or a
     configuration file for package
VISP.
Adjust CMAKE_MODULE_PATH to find
          FindVISP.cmake or set VISP_DIR to the
directory containing a CMake configuration
          file for VISP. The file will
have one of the following names:

VISPConfig.cmake
visp-config.cmake
```

Para ayudar al cmake a encontrar el archivo VISPConfig.cmake lo direccionaremos con VISP_DIR y llamaremos al cmake de nuevo:

```
export VISP_DIR=/home/ViSP-install-
     folder/lib/<multi-arch-folder>/cmake/visp
cmake .
```

O añadir una definición adicional del VISP_DIR:

```
cmake -DVISP_DIR=/home/ViSP-install-
      folder/lib/<multi-arch-folder>/cmake/visp .
```

Generar el ejecutable

Usamos solamente el comando *make*

Lanzar el ejecutable

Usamos como se comentó anteriormente el comando con la siguiente forma:
./tutorial-(llamada) (imagen.ppm/video.)

Una vez acabado este paso, realizaremos los tutoriales de *Tracking: Blob Tracking, Keypoint Tracking* y *Markeless model-based Tracking*. [30]

El control a realizar se basará en el Tutorial de Blob Tracking y Keypoint, concretamente en el archivo tutorial-blob-tracker-live-v4l2.cpp y tutorial-ktl-tracker.cpp, con el cual se harán las llamadas de video para una cámara web Logitech C920 C conectada por USB.

Para ello comprobamos que el código es idéntico al que se presenta:

Tutorial-blob-tracker-live-v4l2.cpp

```
#include <visp3/core/vpConfig.h>
#ifdef VISP_HAVE_MODULE_SENSOR
#include <visp3/sensor/vpV4l2Grabber.h>
#endif
#include <visp3/gui/vpDisplayGDI.h>
#include <visp3/gui/vpDisplayGTK.h>
#include <visp3/gui/vpDisplayOpenCV.h>
#include <visp3/gui/vpDisplayX.h>
#include <visp3/blob/vpDot2.h>
int main()
{
```

```
#if ((defined(VISP_HAVE_V4L2) || (VISP_HAVE_OPENCV_VERSION >=
      0x020100)) && (defined(VISP_HAVE_X11) || defined(VISP_HAVE_GDI)
      || defined(VISP_HAVE_OPENCV) || defined(VISP_HAVE_GTK)))
vpImage<unsigned char> I; // Create a gray level image container
#if defined(VISP_HAVE_V4L2)
vpV4l2Grabber g;
g.open(I);
#elif defined(VISP_HAVE_OPENCV)
cv::VideoCapture g(0); // open the default camera
if(!g.isOpened()) { // check if we succeeded
std::cout << "Failed to open the camera" << std::endl;
return -1;
}
cv::Mat frame;
g >> frame; // get a new frame from camera
vpImageConvert::convert(frame, I);
#endif
#if defined(VISP_HAVE_X11)
vpDisplayX d(I, 0, 0, "Camera view");
#elif defined(VISP_HAVE_GDI)
vpDisplayGDI d(I, 0, 0, "Camera view");

#elif defined(VISP_HAVE_OPENCV)
vpDisplayOpenCV d(I, 0, 0, "Camera view");
#elif defined(VISP_HAVE_GTK)
vpDisplayGTK d(I, 0, 0, "Camera view");
#endif
vpDot2 blob;
blob.setGraphics(true);
blob.setGraphicsThickness(2);
```

```
vpImagePoint germ;
bool init_done = false;
std::cout << "Click!!!" << std::endl;
while(1) {
try {
#if defined(VISP_HAVE_V4L2)
g.acquire(I);
#elif defined(VISP_HAVE_OPENCV)
g >> frame;
vpImageConvert::convert(frame, I);
#endif
vpDisplay::display(I);
if (! init_done) {
vpDisplay::displayText(I, vpImagePoint(10,10), "Click in the blob to
      initialize the tracker", vpColor::red);
if (vpDisplay::getClick(I, germ, false)) {
blob.initTracking(I, germ);
init_done = true;
}
}
else {
blob.track(I);
}
vpDisplay::flush(I);
}
catch(...) {
init_done = false;
}
}
```

```
#endif
}
```

Tutorial-ktl-tracker.cpp

```
#include <visp3/core/vpImageConvert.h>
#include <visp3/klt/vpKltOpencv.h>
#include <visp3/gui/vpDisplayOpenCV.h>
#include <visp3/io/vpVideoReader.h>
int main(int argc, const char *argv[])
{
#ifdef VISP_HAVE_OPENCV
try {
bool opt_init_by_click = false;
for (int i=0; i<argc; i++) {
if (std::string(argv[i]) == "--init-by-click")
opt_init_by_click = true;
else if (std::string(argv[i]) == "--help") {
std::cout << "Usage: " << argv[0] << " [--init-by-click] [--help]" <<
      std::endl;
return 0;
}
}
vpVideoReader reader;
reader.setFileName("video-postcard.mpeg");
vpImage<unsigned char> I;
reader.acquire(I);
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
IplImage * cvI = NULL;
#else
```

```
cv::Mat cvI;
#endif
vpImageConvert::convert(I, cvI);
vpDisplayOpenCV d(I, 0, 0, "Klt tracking");
vpDisplay::display(I);
vpDisplay::flush(I);
vpKltOpencv tracker;
tracker.setMaxFeatures(200);
tracker.setWindowSize(10);
tracker.setQuality(0.01);
tracker.setMinDistance(15);
tracker.setHarrisFreeParameter(0.04);
tracker.setBlockSize(9);
tracker.setUseHarris(1);
tracker.setPyramidLevels(3);
// Initialise the tracking
if (opt_init_by_click) {
vpMouseButton::vpMouseButtonType button = vpMouseButton::button1;
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
std::vector<CvPoint2D32f> feature;
#else
std::vector<cv::Point2f> feature;
#endif
vpImagePoint ip;
do {
vpDisplay::displayText(I, 10, 10,
"Left click to select a point, right to start tracking",
vpColor::red);
if (vpDisplay::getClick(I, ip, button, false)) {
```

```
if (button == vpMouseButton::button1) {
feature.push_back(cv::Point2f((float)ip.get_u(), (float)ip.get_v()));
vpDisplay::displayCross(I, ip, 12, vpColor::green);
}
}
vpDisplay::flush(I);
vpTime::wait(20);
} while(button != vpMouseButton::button3);
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
tracker.initTracking(cvI, &feature[0], feature.size());
#else
tracker.initTracking(cvI, feature);
#endif
}
else {
tracker.initTracking(cvI);
}
std::cout << "Tracker initialized with " << tracker.getNbFeatures() <<
      " features" << std::endl;
while ( ! reader.end() )
{
reader.acquire(I);
vpDisplay::display(I);
vpImageConvert::convert(I, cvI);
if (opt_init_by_click && reader.getFrameIndex() ==
      reader.getFirstFrameIndex() + 20) {
vpMouseButton::vpMouseButtonType button = vpMouseButton::button1;
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
std::vector<CvPoint2D32f> feature;
#else
```

```
std::vector<cv::Point2f> feature;
#endif
vpImagePoint ip;
do {
vpDisplay::displayText(I, 10, 10,
"Left click to select a point, right to start tracking",
vpColor::red);
if (vpDisplay::getClick(I, ip, button, false)) {
if (button == vpMouseButton::button1) {
feature.push_back(cv::Point2f((float)ip.get_u(), (float)ip.get_v()));
vpDisplay::displayCross(I, ip, 12, vpColor::green);
}
}
vpDisplay::flush(I);
vpTime::wait(20);
} while(button != vpMouseButton::button3);
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
tracker.initTracking(cvI, &feature[0], feature.size());
#else
tracker.initTracking(cvI, feature);
#endif
}
tracker.track(cvI);
tracker.display(I, vpColor::red);
vpDisplay::flush(I);
}
vpDisplay::getClick(I);
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
cvReleaseImage(&cvI);
```

```
#endif
return 0;
}
catch(vpException e) {
std::cout << "Catch an exception: " << e << std::endl;
}
#else
(void)argc;
(void)argv;
#endif
}
```

Como podemos comprobar desde el terminal de Linux, y con la dirección correcta:

```
cd /visp-master/tutorial/tracking/blob
```

```
cd /visp-master/tutorial/tracking/keypoint
```

Y añadiendo posteriormente el título del tutorial a realizar para comprobar su correcto funcionamiento (tutorial-blob-tracker-live-v4l2.cpp y tutorial-ktl-tracker.cpp) con el siguiente comando:

```
cd$visp-master$tutorial$tracking$blob$  ./tutorial-blob-tracker-live-v4l2
```

```
cd$visp-master$tutorial$tracking$blob$./tutorial-ktl-tracker video.mpeg
```

Es posible que el V4l2 de fallos sobre el V4l2Grabber, esto se debe a una incorrecta instalación o un fallo por parte de la instalación a la hora de ubicar las librerías. Para solucionar dicho fallo nos aseguraremos de que todas las librerías están instaladas y que se los ejecutables se encuentran en la carpeta `visp-master` y `visp-build`, sin necesidad de realizar el `cmake .` y `make`. [30]

Para acabar el control de la cámara UVC, implementaremos el `tutorial-klt-tracker-live-v4l2.cpp`. Siendo un código mixto entre el tutorial-blob-tracker-live-v4l2.cpp y tutorial-ktl-tracker.cpp [31]

Tutorial-klt-tracker-live-v4l2.cpp

```
#include <visp/vpImageConvert.h>
#include <visp/vpKltOpencv.h>
#include <visp/vpDisplayOpenCV.h>
#include <visp/vpVideoReader.h>
#include <visp/vpV4l2Grabber.h>
int main(int argc, const char *argv[])
{
#if defined(VISP_HAVE_OPENCV) && (defined(VISP_HAVE_V4L2) ||
      (VISP_HAVE_OPENCV_VERSION >= 0x020100))
try {
bool opt_init_by_click = false;
int opt_device = 0;
for (int i=0; i<argc; i++) {
if (std::string(argv[i]) == "--init-by-click")
opt_init_by_click = true;
else if (std::string(argv[i]) == "--device")
opt_device = atoi(argv[i+1]);
else if (std::string(argv[i]) == "--help") {
std::cout << "Usage: " << argv[0] << " [--init-by-click] [--device
      <camera device>] [--help]" << std::endl;
return 0;
}
}
vpImage<unsigned char> I;
#if defined(VISP_HAVE_V4L2)
vpV4l2Grabber g;
std::ostringstream device;
device << "/dev/video" << opt_device;
g.setDevice(device.str());
```

```
g.open(I);
g.acquire(I);
#elif defined(VISP_HAVE_OPENCV)
cv::VideoCapture g(opt_device);
if(!g.isOpened()) { // check if we succeeded
std::cout << "Failed to open the camera" << std::endl;
return -1;
}
cv::Mat frame;
g >> frame; // get a new frame from camera
vpImageConvert::convert(frame, I);
#endif
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
IplImage * cvI = NULL;
#else
cv::Mat cvI;
#endif
vpImageConvert::convert(I, cvI);
// Display initialisation
vpDisplayOpenCV d(I, 0, 0, "Klt tracking");
vpDisplay::display(I);
vpDisplay::flush(I);
vpKltOpencv tracker;
// Set tracker parameters
tracker.setMaxFeatures(200);
tracker.setWindowSize(10);
tracker.setQuality(0.01);
tracker.setMinDistance(15);
tracker.setHarrisFreeParameter(0.04);
```

```
tracker.setBlockSize(9);
tracker.setUseHarris(1);
tracker.setPyramidLevels(3);
// Initialise the tracking
if (opt_init_by_click) {
#if (VISP_HAVE_OPENCV_VERSION >= 0x020408)
vpMouseButton::vpMouseButtonType button;
std::vector<cv::Point2f> guess;
vpImagePoint ip;
do {
vpDisplay::displayText(I, 10, 10, "Left click to select a point, right
      to start tracking", vpColor::red);
if (vpDisplay::getClick(I, ip, button, false)) {
if (button == vpMouseButton::button1) {
guess.push_back(cv::Point2f((float)ip.get_u(), (float)ip.get_v()));
vpDisplay::displayText(I, 10, 10, "Left click to select a point, right
      to start tracking", vpColor::red);
vpDisplay::displayCross(I, ip, 12, vpColor::green);
}
}
vpDisplay::flush(I);
vpTime::wait(20);
} while(button != vpMouseButton::button3);
tracker.initTracking(cvI, guess);
#endif
}
else {
tracker.initTracking(cvI);
}
while ( 1 ) {
```

```
#if defined(VISP_HAVE_V4L2)
g.acquire(I);
#elif defined(VISP_HAVE_OPENCV)
g >> frame;
vpImageConvert::convert(frame, I);
#endif
vpDisplay::display(I);
vpImageConvert::convert(I, cvI);
tracker.track(cvI);
tracker.display(I, vpColor::red);
vpDisplay::displayText(I, 10, 10, "Click to quit", vpColor::red);
vpDisplay::flush(I);
if (vpDisplay::getClick(I, false))
break;
}
#if (VISP_HAVE_OPENCV_VERSION < 0x020408)
cvReleaseImage(&cvI);
#endif
return 0;
}
catch(vpException e) {
std::cout << "Catch an exception: " << e << std::endl;
}
#else
(void)argc;
(void)argv;
#endif
}
```

Para ello a través del terminal direccionaremos a la carpeta correcta:

```
~visp-master/build/tutorial/tracking/keypoint
```

Una vez nos encontremos en dicha carpeta, empleando el comando:

```
./tutorial-klt-tracker-live-v4l2
```

Nos aparecerá una ventana mostrando el tracking de la cámara UVC:

Fig.31. *Vista de la cámara UVC con el control*

Como podemos comprobar, la imagen nos aparecerá en escala de grises. Basándonos en la imagen real capturada por la cámara UVC sin el control de keypoint, perderá resolución.

La imagen perderá resolución y aparecerá en escala de grises.

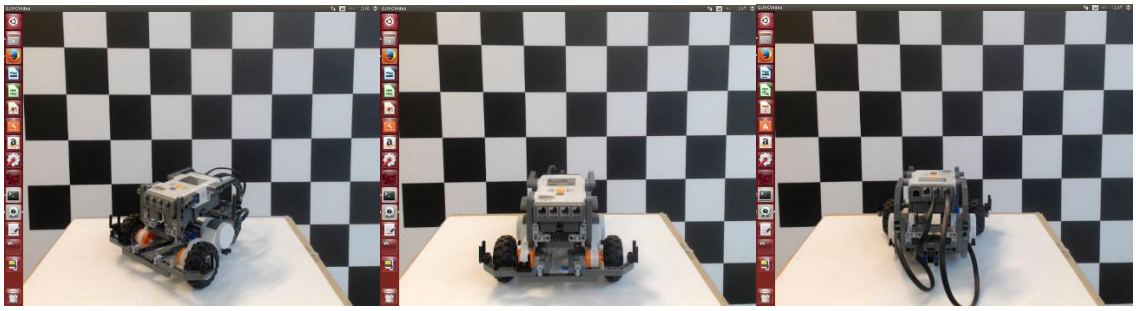


Fig.32. *Vista de la cámara UVC sin el control*

Una vez se tenga esta aplicación exclusiva en ViSP, avanzaremos un paso más y uniremos la librería ViSP a ROS aplicada a la cámara UVC.

Lo primero que deberemos realizar será la instalación de los paquetes necesarios para la implementación de ViSP en ROS. Desde la página de ROS:

http://wiki.ros.org/vision_visp [36]

Instalación por paquetes precompliados

```
sudo apt-get install ros-indigo-vision-visp
```

Instalación por código fuente

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ~/catkin_ws
catkin_make
```

Llevar la fuente

```
cd ~/catkin_ws/src
```

Copiamos el enlace de GitHub:

```
git clone https://github.com/lagadic/vision_visp.git
```

Comprobamos la rama elegida, en nuestro caso será Indigo:

```
cd vision_visp
git checkout indigo
```

Instalación de las dependencias

```
source ~/catkin_ws/devel/setup.bash
rosdep update
rosdep install visp_bridge
```

Construir la fuente

```
cd ~/catkin_ws
catkin_make -j4 -DCMAKE_BUILD_TYPE=Release
```

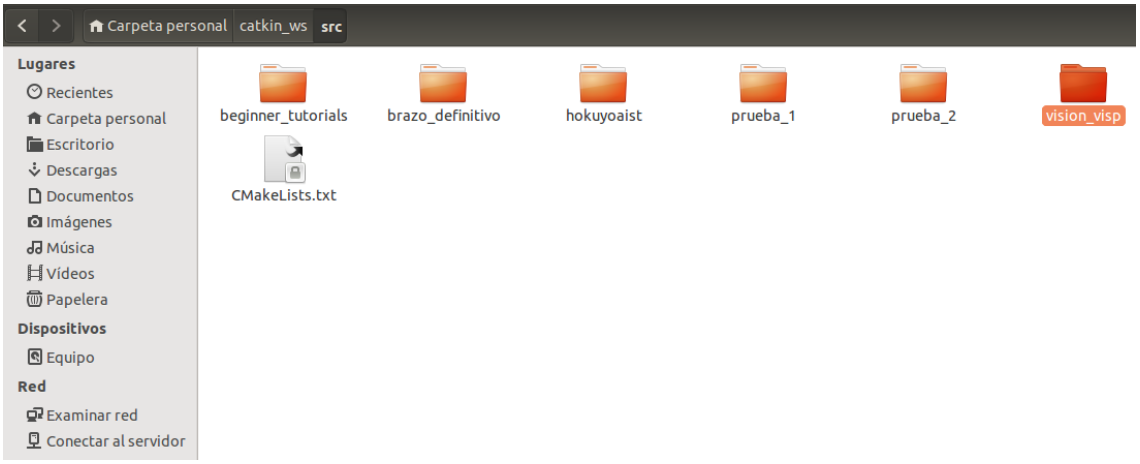


Fig.33. *Carpeta visión_visp dentro de src*

Con esto ya estaría el paquete vision_visp listo para utilizar.

El próximo paso que realizaremos para el control de la cámara será la calibración de la misma con el nuevo paquete de ViSP en ROS y la conexión del puerto USB.

En primer lugar deberemos ir a la carpeta de visp_camera_calibration, con la siguiente dirección:

`catkin_ws/src/visión_visp`

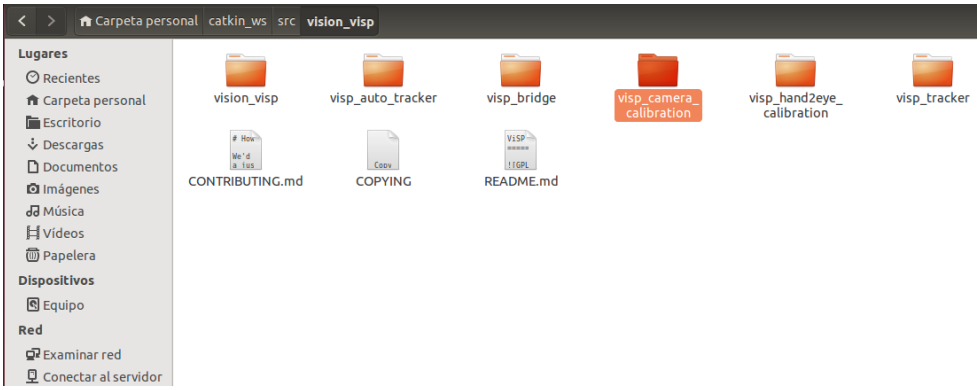


Fig.34. *Carpeta visp_camera_calibration dentro de visión_visp*

<u>Calibración de la cámara:</u>

1. Desde el terminal y una vez nos direccionemos, realizamos un `cmake .` y después un `make`.

2. Luego para asegurarnos de que está instalado el paquete de visp dentro del visión_visp, realizamos `rosdep install visp_camera_calibration`.

3. Realizamos `catkin_make -DCMAKE_BUILD_TYPE=Release --pkg visp_camera_calibration`

4. Para evitar los posibles fallos de conexión entre el hardware (cámara UVC) y el código de calibración, instalaremos el paquete: usb_cam-test.launch, descargado de la página GitHub:

https://github.com/bosch-ros-pkg/usb_cam/blob/develop/launch/usb_cam-test.launch

usb_cam-test.launch

```
<launch>

      <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
     output="screen" >

          <param name="video_device" value="/dev/video0" />
          (la cámara está encendida)

          <param name="image_width" value="640" />  (ancho de
          la imagen)

          <param name="image_height" value="480" />  (alto de
          la imagen)

          <param name="pixel_format" value="yuyv" /> (posibles
          valores de la imagen)

          <param name="camera_frame_id" value="usb_cam" />
          (ventana/marco de la cámara)

          <param name="io_method" value="mmap"/> (mapa de
          entrada y salida)

     </node>

     <node name="image_view" pkg="image_view" type="image_view"
     respawn="false" output="screen">

          <remap from="image" to="/usb_cam/image_raw"/>
          (almacenamiento de información auxiliar)

          <param name="autosize" value="true" /> (autotamaño)

     </node>

</launch>
```

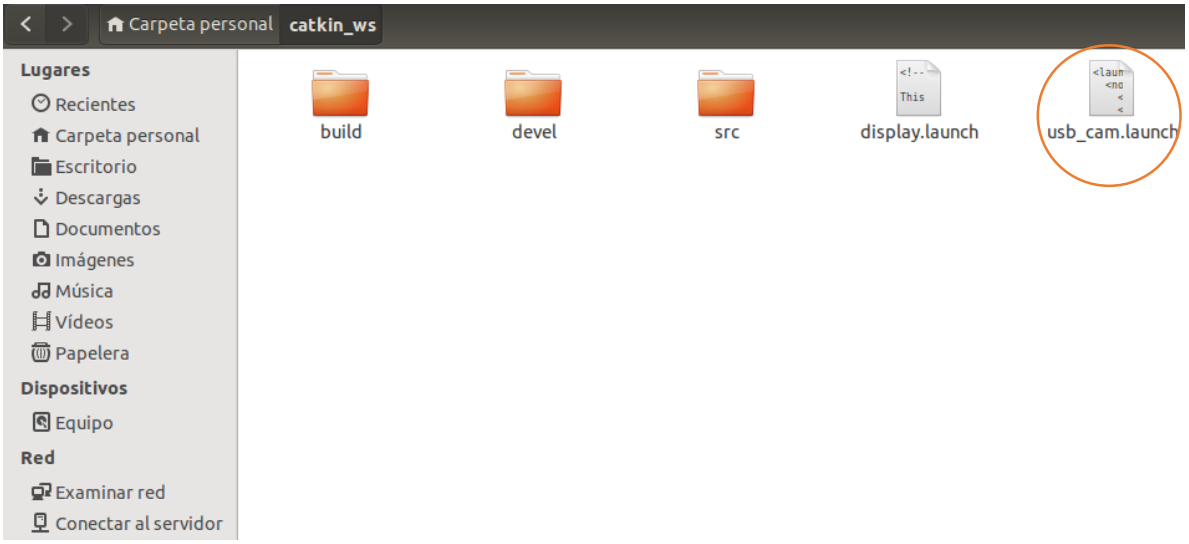


Fig.35. *Ubicación de usb_cam.launch*

5. Creamos e implementamos el fichero calib-live-firewire.launch para la plantilla del patrón de calibración de círculos.

calib-live-firewire.launch

```
<launch>
  <!-- % rxconsole -->
  <node pkg="rqt_console" name="rqt_console"
type="rqt_console"/>


  <node name="usb_cam" pkg="usb_cam" type="usb_cam_node"
output="screen" >
      <param name="video_device" value="/dev/video0" />
      <param name="image_width" value="640" />
      <param name="image_height" value="480" />
      <param name="pixel_format" value="yuyv" />
      <param name="camera_frame_id" value="usb_cam" />
      <param name="io_method" value="mmap"/>
  </node>
```

```
  <!-- % rosrun image_view image_view image:=/usb_cam/image_raw
-->

  <node pkg="image_view" type="image_view"
name="my_image_raw_viewer" args="image:=/usb_cam/image_raw"/>


     <arg name="calibration_path" default="calibration.ini" />

     <group ns="visp_camera_calibration">

                <node pkg="visp_camera_calibration"
name="visp_camera_calibration_calibrator"
type="visp_camera_calibration_calibrator"/>

                <node pkg="visp_camera_calibration"
name="visp_camera_calibration_image_processing"
type="visp_camera_calibration_image_processing"
args="camera_prefix:=/usb_cam">

                     <param name="gray_level_precision"
value="0.7" />

                     <param name="size_precision" value="0.5"
/>

                     <param name="pause_at_each_frame"
value="False" />

                     <param name="calibration_path"
type="string" value="$(arg calibration_path)" />


                     <!-- 3D coordinates of all points on the
calibration pattern. In this example, points are planar -->

                     <rosparam param="model_points_x">[0.0,
0.03, 0.06, 0.09, 0.12, 0.15, 0.0, 0.03, 0.06, 0.09, 0.12, 0.15,
0.0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.0, 0.03, 0.06, 0.09, 0.12,
0.15, 0.0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.0, 0.03, 0.06, 0.09,
0.12, 0.15]</rosparam>

                     <rosparam param="model_points_y">[0.0,
0.00, 0.00, 0.00, 0.00, 0.00, .03, 0.03, 0.03, 0.03, 0.03, 0.03,
.06, 0.06, 0.06, 0.06, 0.06, 0.06, .09, 0.09, 0.09, 0.09, 0.09,
0.09, 0.12,0.12, 0.12, 0.12, 0.12, 0.12, 0.15,0.15, 0.15, 0.15,
0.15, 0.15]</rosparam>
```

```
                    <rosparam param="model_points_z">[0.0,
0.00, 0.00, 0.00, 0.00, 0.00, 0.0, 0.00, 0.00, 0.00, 0.00, 0.00,
0.0, 0.00, 0.00, 0.00, 0.00, 0.00, 0.0, 0.00, 0.00, 0.00, 0.00,
0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.0, 0.00, 0.00, 0.00,
0.00,0.00]</rosparam>


                    <!-- 3D coordinates of 4 points the user
has to select to initialise the calibration process -->

                    <rosparam param="selected_points_x">[0.03,
0.03, 0.09, 0.12]</rosparam>

                    <rosparam param="selected_points_y">[0.03,
0.12, 0.12, 0.03]</rosparam>

                    <rosparam param="selected_points_z">[0.00,
0.00, 0.00, 0.00]</rosparam>


                </node>

    </group>

</launch>
```

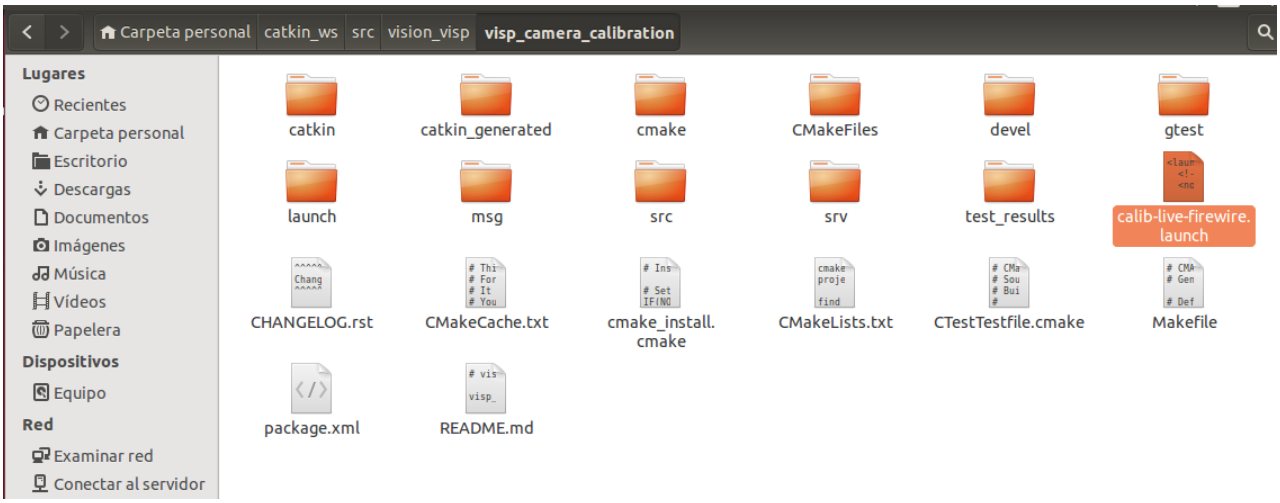


Fig.36. *calib-live-firewire.launch dentro de visp_camera_calibration*

6. Ejecutamos un `roscore` en un nuevo terminal.

7. Ejecutamos el nuevo launch con:

```
roslaunch catkin_ws/ src/ visión_visp/
visp_camera_calibration/ calib-live-firewire.launch
```

8. Cogeremos el patrón de la calibración de círculos y lo enfocaremos con la cámara UVC. Lo moveremos en distintas posiciones y ángulos para tener una calibración más precisa.

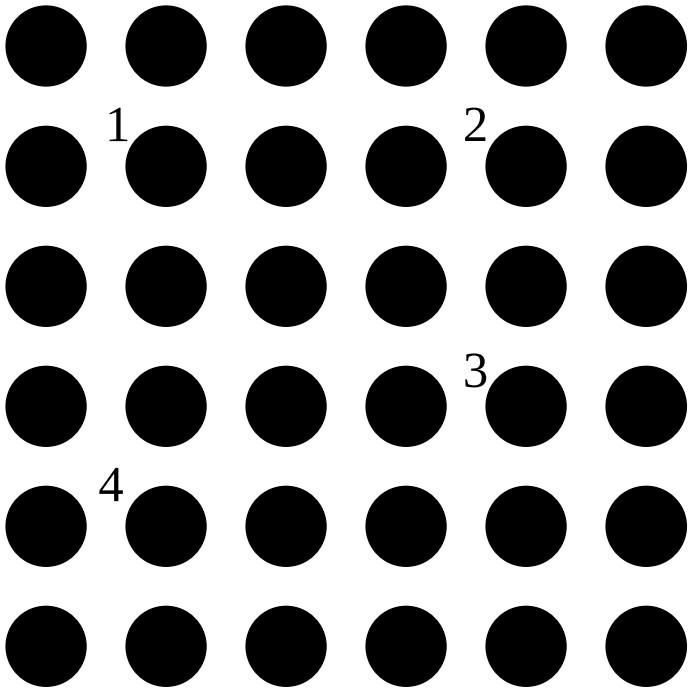


Fig.37. *Patrón de calibración de círculos.*

9. Realizamos una llamada de servicio para observar la configuracion de la cámara con el comando: `rosservice call visp_camera_calibration/calibrate 2 640 480`

- El 2 se refiere a la calibración de la cámara con el posible ruido. En caso de poner 1 la calibración se estimaría sin ruido.
- Los valores de 640 y 480, son el ancho y alto respectivamente de la imagen.

10. Para mostrar la información de la calibración por pantalla: `./ros $nano calibration.ini`

11. Una vez tenemos la cámara calibrada y comprobamos que los datos por pantalla son los correctos estará lista para detectar y seguir el código QR.

Nota: la resolución de la cámara puede ser cambiada de 640x480, por cualquier otra, siempre que no se sobrepase el límite de 1920x1080, dado que la resolución máxima de la cámara C920 C de Logitech es de 1080p.

Una vez tengamos la cámara UVC calibrada con el patrón circular y la hayamos ajustado procederemos a ejecutar la detección del patrón del código QR.

Puesto que el paquete `vision_visp` de la librería ViSP implementada en ROS, incorpora la carpeta `visp_auto_tracker`, usaremos el `tracklive_usb.launch` para realizar el tracking (seguimiento) de código QR. [35][36]

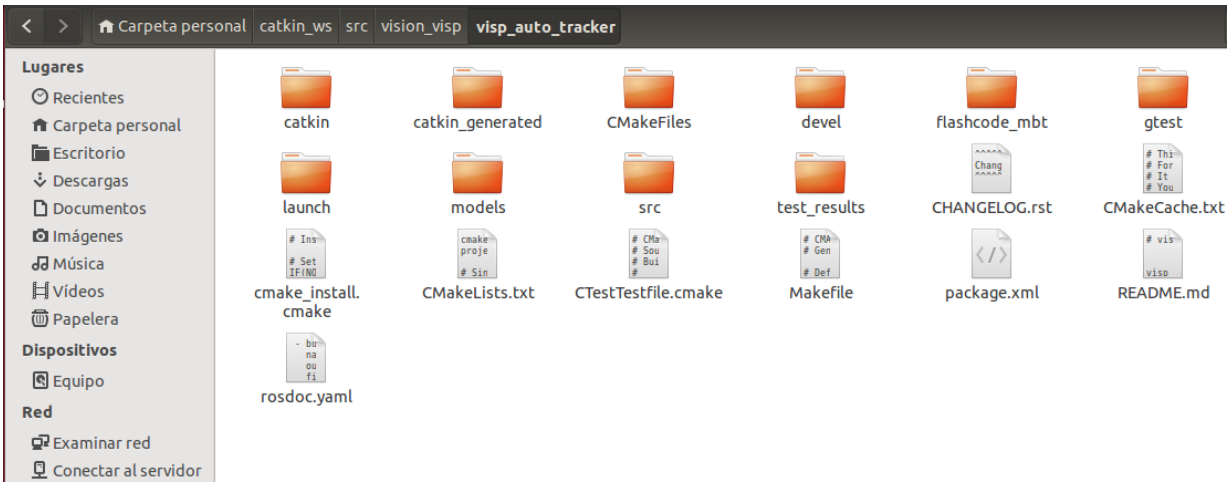


Fig.38. *Carpeta visp_auto_tracker*

Este seguimiento se basa como se comentó anteriormente en el posicionamiento de puntos. Estos puntos se pueden configurar en el patrón del código QR y Flash, presente en la carpeta de `visp_auto_tracker/models.`

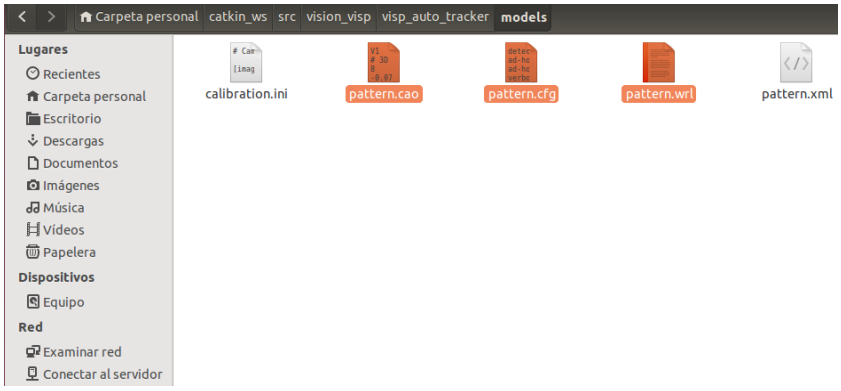


Fig.39. *Patrones para el código QR y Flash*

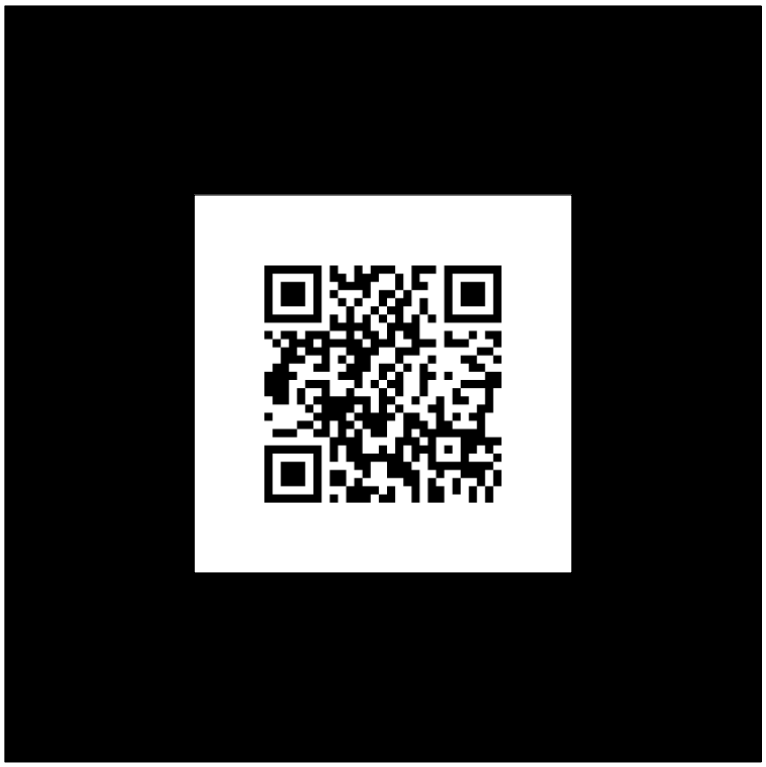


Fig.40. *Código QR de detección y seguimiento*

Pattern.cfg

```
#set the detector type: "zbar" to detect QR code, "dmtx" to detect fla
shcode (podemos cambiarlo para detector QR o Flash)

detector-type= zbar (configurado para QR)

#enable recovery mode when the tracker fails

ad-hoc-recovery= 1


#point 1

flashcode-coordinates= -0.0275 (eje x)

flashcode-coordinates= -0.0275 (eje y)

flashcode-coordinates= 0.000 (eje z)

#point 2

flashcode-coordinates= 0.0275

flashcode-coordinates= -0.0275

flashcode-coordinates= 0.000

#point 3
```

```
flashcode-coordinates= 0.0275
flashcode-coordinates= 0.0275
flashcode-coordinates= 0.000
#point 4
flashcode-coordinates= -0.0275
flashcode-coordinates= 0.0275
flashcode-coordinates= 0.000

#point 1
inner-coordinates= -0.03825
inner-coordinates= -0.03825
inner-coordinates= 0.000
#point 2
inner-coordinates= 0.03825
inner-coordinates= -0.03825
inner-coordinates= 0.000
#point 3
inner-coordinates= 0.03825
inner-coordinates= 0.03825
inner-coordinates= 0.000
#point 4
inner-coordinates= -0.03825
inner-coordinates= 0.03825
inner-coordinates= 0.000

#point 1
outer-coordinates= -0.0765
outer-coordinates= -0.0765
outer-coordinates= 0.000
#point 2
outer-coordinates= 0.0765
outer-coordinates= -0.0765
outer-coordinates= 0.000
#point 3
outer-coordinates= 0.0765
```

```
outer-coordinates= 0.0765
outer-coordinates= 0.000
```

```
#point 4
outer-coordinates= -0.0765
outer-coordinates= 0.0765
outer-coordinates= 0.000
```

Modificando este código se podría implementar el seguimiento tanto para el código QR o el código Flash según se requiera, así como el posicionamiento de los puntos de detección para el posterior seguimiento de los mismos en un plano XYZ. La modificación de estos puntos está relacionada directamente con el código QR o Flash a detectar, por ello para emplear el mismo código QR en nuestras pruebas no han sido modificados.

Una vez tengamos todo a punto, lanzaremos el `tracklive_usb.launch` desde el terminal. Para ello lo direccionaremos a la carpeta:

```
cd catkin_ws/src/vision_visp/visp_auto_tracker/launch
```

Ya estando en la carpeta realizamos el comando: `roslaunch tracklive_usb.launch`, mostrándonos por pantalla la detección del código QR.

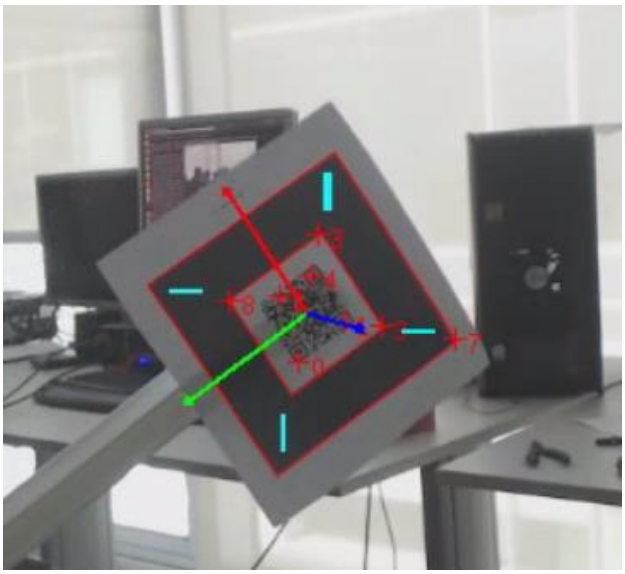


Fig.41. *Código QR detectado y seguido con el auto tracker*

Fig.42. *Conexión de nodos Cámara- Visp-auto-tracker*

El eje X se corresponde a la flecha roja, el eje Y con la flecha verde y el eje Z con la flecha azul. Como se puede observar el valor en el eje Z es siempre de 0 puesto que se trata de una cámara que captura una imagen en un plano 2D sobre el plano XY.

Debemos señalar que se han realizado varias pruebas con este código QR una vez se calibro la cámara correctamente y se ejecutó la aplicación, impreso en papel en diferentes tamaños: 2.5x2.5 cm, 6x6 cm, 8x8 cm, 10x10 cm y el original de 15x15 cm. Tras probar todos estos códigos en diferentes tamaños hemos podido observar que debido a una mala resolución de la imagen impresa en papel el tamaño mínimo el cual es detectado es el de 10x10 cm, siendo el de 15x15 el detectado con mayor facilidad. También podemos modificar la resolución de la cámara aumentándola a su máximo de 1920x1080, donde detectará y seguirá con mayor facilidad el código QR a mayor distancia, pero seguirá estando limitado por la calidad de la impresión del código QR y los reflejos ocasionados por la incidencia de la luz.

Una vez ha sido detectado por la cámara se ha movido a distintas distancias y velocidades, girándolo hasta perder la detección y luego volviéndola a recuperar. Realizando estas pruebas hemos conseguido saber que una vez detecta el patrón lo puede seguir hasta una distancia de 3.6 metros, incluso moviéndose; a una distancia de 1.15 metros es capaz de seguir el QR hasta un ángulo aproximado de unos 75º. Con estas condiciones es fácil seguir cualquier objeto móvil que lleve el código QR pegado. Las pruebas que realizaremos en el laboratorio serán sobre el Summit (robot móvil) y un brazo robot (realizando movimientos sobre un plano 2D y sobre un plano 3D).

Nota: Todas las pruebas se han realizado con la resolución estándar de 640x480 con la que funcionan la mayoría de cámaras UVC.

tracklive_usb.launch

```
<!-- -*- xml -*-

-->
<launch>
     <!-- Launch the tracking node -->
          <node pkg="visp_auto_tracker"
     type="visp_auto_tracker" name="visp_auto_tracker"
     output="screen">
```

```
      <param name="model_path" value="$(find
      visp_auto_tracker)/models" />
      <param name="model_name" value="pattern" />
      <param name="debug_display" value="True" />
      <remap from="/visp_auto_tracker/camera_info"
      to="/usb_cam/camera_info"/>
      <remap from="/visp_auto_tracker/image_raw"
      to="/usb_cam/image_raw"/>
</node>

<!-- Launch the usb camera acquisition node -->
      <node pkg="usb_cam" type="usb_cam_node" name="usb_cam"
      output="screen">
      <param name="image_width" value="640" />
      <param name="image_height" value="480" />
      <param name="video_device" value="/dev/video0" />
      <param name="pixel_format" value="yuyv" />
      <!-- rename the camera name topic into /camera/image_raw to
match the one in visp_auto_tracker/models/calibration.ini file -
->
      <param name="camera_name" value="/camera/image_raw" />
      <param name="camera_info_url"
value="package://visp_auto_tracker/models/calibration.ini"
type="string" />
      </node>
</launch>
```

Nota: con este código se detecta en video el código QR sobre una superficie plana. Los parámetros de la cámara han sido fijados en el rosparam, calibrados con anterioridad.

Una vez implementado este sistema de auto tracking de ViSP en ROS, lo aplicaremos a un móvil para realizar las pruebas de video y el ploteado de gráficas a través de Matlab.

Esto se realizará ejecutando una serie de comandos por terminal para conectar los nodos pertinentes usando el topic generado por el `visp_auto_tracker` y el ejecutable por cámara UVC `tracklive_usb.launch`.

Los pasos seguidos para realizar correctamente el ploteado han sido los siguientes:

1. En un terminal lanzamos un `roscore`.
2. En otro terminal distinto y sin cerrar el anterior, lanzamos el `tracklive_usb.launch`

3. En un nuevo terminal ejecutamos `rqt_plot`

   Una vez nos aparezca la ventana del rqt_plot, insertamos: `/visp_auto_tracker/object_position/pose/position`

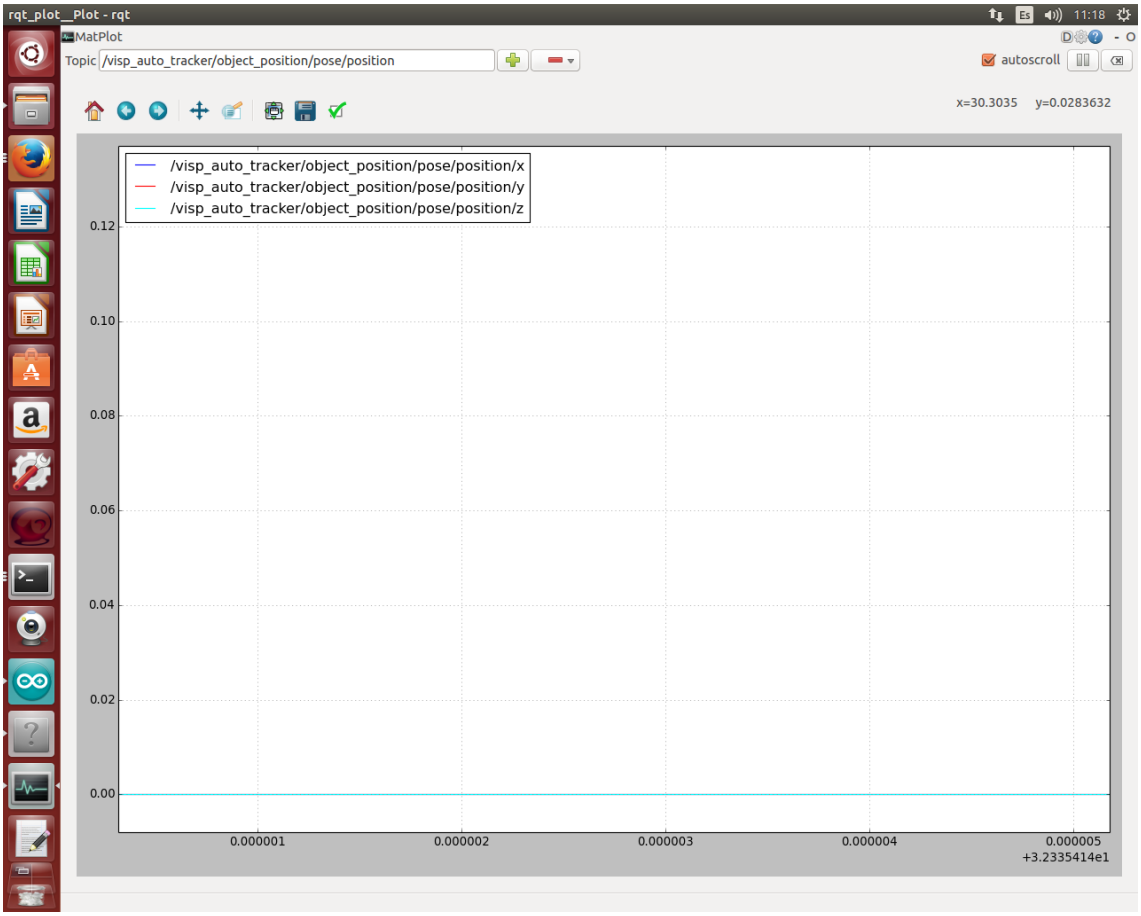


Fig.43. *Grafico del rqt_plot configurado para la posición del objeto*

4. Ahora, con la cámara enfocando el código QR podemos observar cómo se dibuja la posición según los ejes x y z según lo movamos.

5. Sin cerrar nada en un nuevo terminal, abrimos `rosbag record -a -O (nombre del archivo .bag que deseemos guardar)`

Nota: -a para todos los topic.

6. Una vez hayamos grabado cierto tiempo de video moviendo el código QR mientras es detectado y seguido, cerramos el `rosbag record`, el `rqt_plot` y el `tacklive_usb.launch` en dicho orden. Una vez cerrados abrimos nuevamente un `roscore`, esto se hace asi para evitar fallos en el

direccionamiento de los notos y que la dirección del futuro `rosbag` sea errónea.

7. En un terminal lanzamos: `rosbag play (nombre del archivo .bag guardado con anterioridad)—-pause`

   Usamos --pause para evitar que se ejecute hasta que pulsemos la barra espaciadora.

8. En otro terminal, lanzamos:

   `rostopic echo /visp_auto_tracker/object_position`

   Una vez este pulsamos la barra espaciadora en el `rosbag play` y nos mostrara por pantalla los datos grabados de `rosbag record` con el `rqt_plot`.

9. Empleando el siguiente código, podremos emplear el archivo generado .bag para graficarlo por Matlab.

   Nota: necesitaremos una versión de Matlab 2015 o superior, puesto que usaremos el paquete Robotic System Toolbox, empelado para unir ROS con Matlab y poder plotear.

CargarBag.m

```
%%Fichero para trabajar con los BAG, de ROS.

%Inicializar primero el entorno con ros.

rosinit

%%

%Cargar el BAG, en el Workspace de Matlab.

bag =
rosbag('/Users/Borja/Desktop/MatlabDadesRBCAR/navigation.bag')

%%

%Ver los posibles topics disponibles en el entorno en este caso
en el BAG.

topics_al_bag=bag.AvailableTopics


%%
```

```
%Si quieres ver los mensajes del bag puedes hacer el siguiente
comando.

LLista_missatges = bag.MessageList

%Si quieres tambien es posible añadir algunos comandos para
filtrar un numero concreto

%de mensajes. MessageList(500:505,:)

%%

% También se puede filtrar un tipo de mensaje del bag en
concreto, para entender hay que saber que no solo los topics
publican mensajes

%También lo hacen algunas de las transformadas por eso primer
hay que indicar que tipo de mensaje es.

bagselect1 = select(bag,'Topic','/rbcar_controller/command')


%%

%Se puede también conseguir el tiempo concreto en el que se
inicia el bag.

starttime = bag.StartTime;

stoptime = bag.EndTime;

%%

% Leer una serie determinada de mensajes.

msgs = readMessages(bagselect1);

size(msgs)

%%

%Como plotear con todos los tiempos dados.

ts = timeseries(bagselect1, 'Drive.SteeringAngle','Drive.Speed')
%podemos modificar en este caso el Drive por

%Ver los tiempos.

dades_ts = ts.Data

%meaning

mean_ts = mean(ts)
```

```
%Dibujar el TimeSeries.

figure

plot(ts,'LineWidth',3)
```

10. Una vez hayamos cargado este programa de Matlab, desde la ventana de comandos, realizamos los plots pertinentes para el valor de X, Y y Z.

```
Command Window
>> load('x_y_z_pruebabrazo.mat')
>> hold on
>> grid
>> plot (ts_x,'r') %valor de X en rojo
>> plot (ts_y,'g') %valor de Y en verde
>> plot (ts_z,'b') %valor de Z en azul
>>
```

Fig.44. *Comandos plot para las gráficas de X, Y y Z*

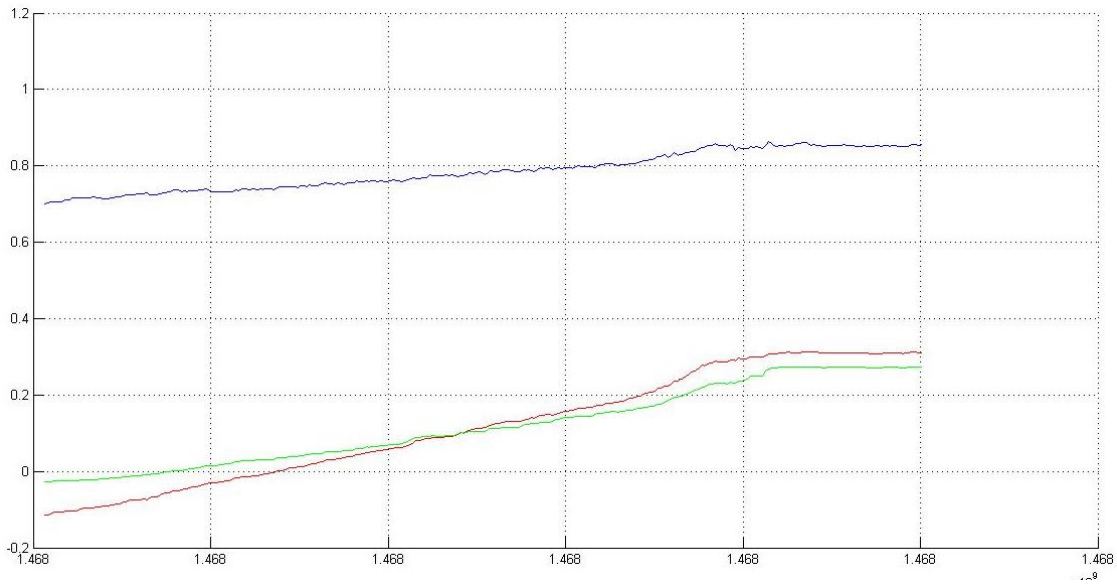


Fig.45. *Gráficas de X, Y y Z*

Ahora que ya tenemos el archivo .bag que sigue la trayectoria de un móvil, en este caso el prototipo de un brazo robot en tiempo real. Comprobamos que el seguimiento es el esperado y se corresponde con el del brazo robot.

Para ello sabiendo que el eje de referencia XYZ es distinto para el código QR y para el brazo robot, debemos de pasarlo a los mismos ejes. Se empleará el archivo de Matlab creado para ello ComparativaBrazo.mat.

Este modelo se implementará con los siguientes comandos y ecuaciones, las cuales transforman los valores de los ejes para poder compararlos:

```
ts_z_brazo=ts_x+0.85+0.1138 %(cambiamos la z del brazo robot a la x del tracking, los valores numéricos se deben a la posición del código QR sobre el brazo)


ts_z_def=ts_z-2*(ts_z-0.85) %(z definitiva)

ts_z_def=ts_z_brazo-2*(ts_z_brazo-0.85)

ts_x_brazo=-1*ts_y  %(la x del brazo robot se corresponde a la –Y del tracking)

ts_x_brazo=ts_y
```

En resumen:

- Eje X del brazo es igual al Eje –Y del código QR.
- Eje Y del brazo es igual al Eje Z del código QR.
- Eje Z del brazo es igual al Eje X del código QR.

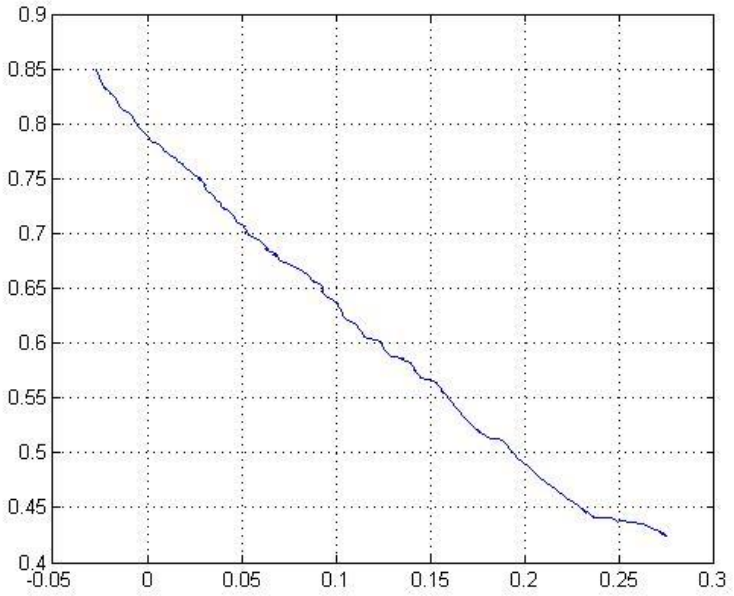


Fig.46. *Gráficas YZ del brazo robot*

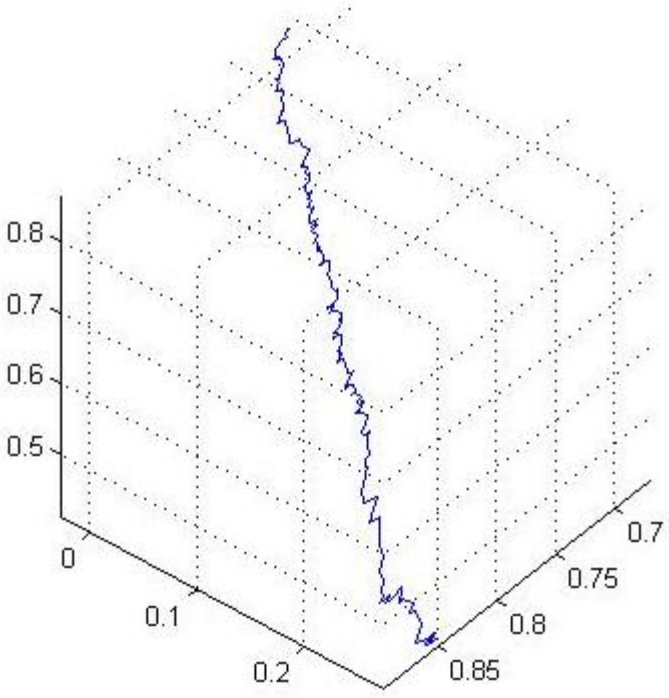


Fig.47. *Gráficas XYZ del brazo robot en 3D*

Una vez tengamos los modelos tanto del seguimiento del código QR en el brazo y del propio brazo,

- plot3(ts_x_brazo.data,ts_y_brazo.data,ts_z_def.data) %(gráfica del brazo en 3D)
- plot3(ts_y.data,ts_z.data,ts_x.data,'r') %(gráfica del código QR en 3D, transofrmada para la comparativa)

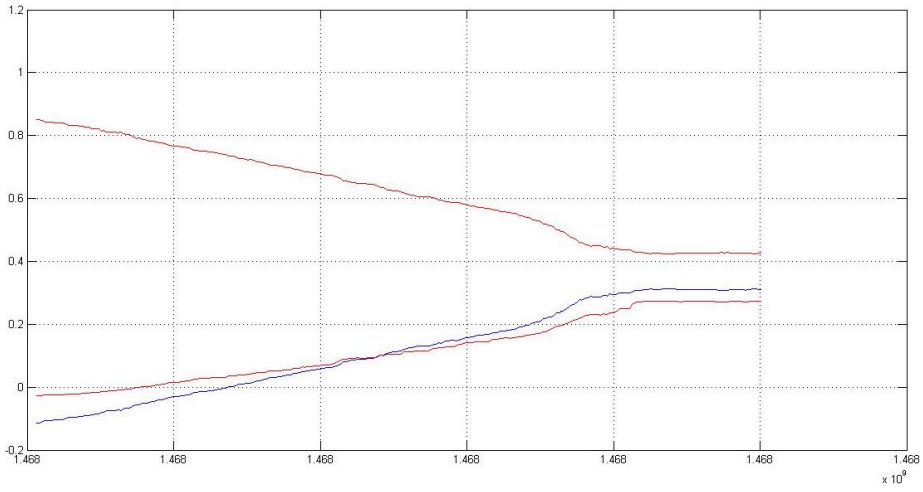


Fig.48. *Gráficas 2D comparativa*

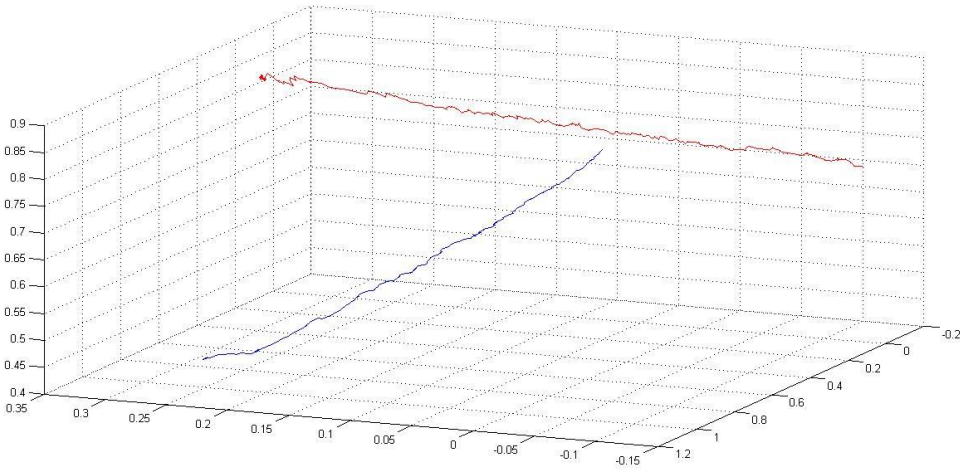


Fig.49. *Comparativa 3D de las gráficas del brazo y código QR sin transformar*

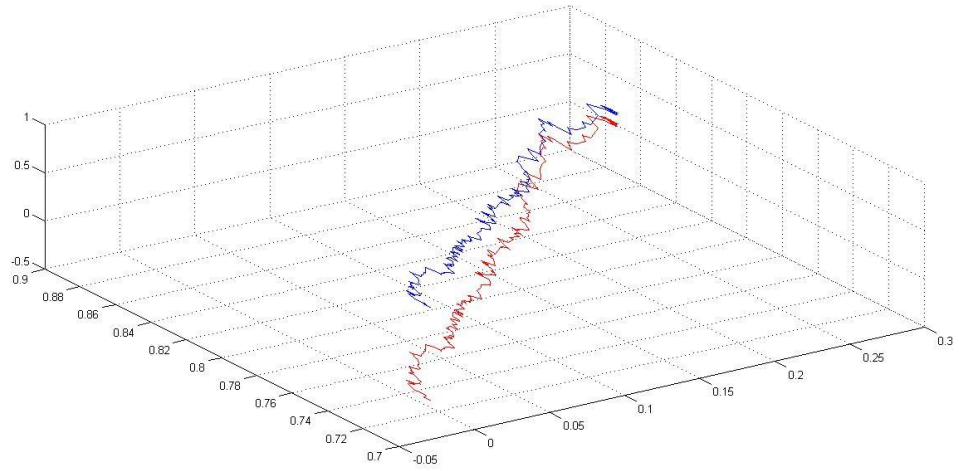


Fig.50. *Comparativa 3D de las gráficas del brazo y código QR transformada*

Como se puede observar, el seguimiento del código QR (gráfica roja) sobre el movimiento del brazo robot (gráfica azul) es prácticamente idéntico, por la salvedad de la separación existente la posición de código QR sobre la punta del brazo robot y el pandeo del papel sobre el cual está impreso. Además podemos añadir el leve error de los servos del brazo y la desviación generada en ambas graficas debido a tirones en el movimiento.

Con esto cerramos el apartado de control de la cámara UVC. Como hemos podido observar con ella podemos hacer la detección automática y seguimiento de puntos tanto con ViSP solo como implementando con ROS, siendo este último mucho más eficiente y más completo. A su vez obtenemos información en tiempo real y podemos graficar los movimientos del móvil que detecta y sigue gracias al código QR.

5.2.1.3. Paquetes

En este proyecto se han hecho uso de dos tipos de paquetes, los empleados por ROS y los de ViSP. A pesar de que ViSP es una librería complementaria a ROS, no se instalan con los mismos paquetes, ni emplean siempre los mismos códigos.

Los paquetes de ROS empleados en este proyecto son el necesario para realizar la instalación RViz y el del sensor Hokuyo.

Los paquetes que se usarán en ViSP ya vienen con la instalación, y serán los que ejecutaremos a través del terminal. Para la ejecución de paquetes previamente deberemos convertirlos en ejecutables como se explicó. Los principales serán Blob y Keypoint.

Cabe destacar que los paquetes de drivers tanto para la cámara UVC y sensor 2D han de instalarse para la detección de los mismos por parte de la computadora y su correcto funcionamiento. Dentro de estos drivers destacan el hokuyo_node para el sensor y el usb_cam para la cámara.

5.2.1.4. Librerías

Las librerías que se emplearán en este proyecto serán las necesarias para el correcto funcionamiento de ROS y ViSP, siendo algunas de ellas comunes para ambos.

Estas librerías ya se instalaran con la instalación propia de ROS y ViSP. En caso de no hallarse en la instalación de los mismos siempre los podemos bajar a través de la página: https://github.com/

Las principales librerías que debemos asegurar tener instaladas son: libX11, libv4l, libopencv, liblapack, libdc1394, libxml2, libzbar y por supuesto ViSP, que aunque es empleado como software no deja de ser una librería complementaria de ROS.

Para Matlab, necesitaremos una versión a partir del Matlab 2015 a o posterior, ya que incluye el paquete Robotic System Toolbox, necesario para obtener la información de los ficheros .bag, creados con rosbag para graficar la posición de los objetos.

Con la correcta instalación de las librerías evitaremos fallos de conexión, llamadas y la creación de los ejecutables. Esto se consigue prestando especial atención a los comandos de recomendación a las terceras particiones (Recommended 3rd parties).

5.3. Cámara UVC

La cámara UVC 2.0 USB empleada es de la marca Logitech modelo C920 C.

Esta cámara es de alta resolución (HD 1080p (1920x1080) y a 30 frames por segundo), con lo que a pesar de perder resolución con el control del keypoint, conseguiremos mantener una resolución aceptable, no ocurriendo esto con el visp_auto_tracker, ya que mantiene la resolución pudiendo incluso variarla según nuestras necesidades.

A su vez el ángulo de visión es de 78º, suficiente para detectar y realizar el seguimiento de los objetos en el rango en el que trabaja el sensor Hokuyo, rondando los 6 a 14 metros. En este proyecto las pruebas se han realizado a distancias menores, comprendidas entre 1 y 4 metros, obteniendo resultados satisfactorios. Para el caso de 6 a 14 metros deberíamos cambiar la resolución a la máxima, como se explicaba en el apartado anterior.

Como es obvio, esta cámara se conecta a la computadora mediante un cable USB 2.0. Dado que nuestro robot móvil posee varios de estos puertos no será problema conectarla. También podemos destacar que esta cámara posee un soporte para sujetarse a una superficie ya sea por agarre propio de la pinza o atornillada a un soporte.

Finalmente, este dispositivo puede funcionar sin problemas una vez se instalen los drivers pertinentes (proporcionados por Logitech), en Linux independientemente de la arquitectura de 32 y 64 bits. [32]

Fig.51. *Cámara UVC C920 C de Logitech*

5.4. Sensor 2D

El sensor empleado en este proyecto, como ya se valoró en el punto 4.3 Sensores, será el Hokuyo URG-04LX-UG01, de tecnología LiDAR 2D.

Este sensor utiliza tecnología láser 2D para la detección de objetos, es decir, transmite un pulso de luz y mide el tiempo que tarda en reflejarse sobre un blanco y volver al remitente, en este caso el receptor del sensor.

Como especificaciones generales de este dispositivo podemos destacar varias:

- Área de escaneo de 240º, la cual limitaremos a 180º para recoger la información que nos interesa en ese ángulo.
- Longitud de onda de 785 nm.
- Haz laser menor de 20mm a 2000mm, con una divergencia menor entre 40 mm y 4000mm. Esto nos indica una buena precisión de emisión y recepción del haz, disminuyendo el error, y una buena distancia máxima de detección.
- Bajo consumo, que ronda los 500 mA y los 800 mA durante el encendido, alimentado por 5 V DC.
- Conexión de interface a través de puerto USB 2.0 a 12 Mbps, lo cual nos facilita la tarea para acoplarlo al RBCAR, puesto que este cuenta con varios puertos USB.
- Gran resistencia a impactos, 196 m/s^2. Indicándonos que se trata de un dispositivo resistente, perfecto para robótica móvil dado que puede estar expuesto a golpes y colisiones accidentales.
- Resolución angular de 0.36º. Siendo esta la capacidad de poder distinguir entre dos objetos muy próximos. Con el valor de este sensor podremos distinguir la mayoría de los objetos que podemos encontrar en un rango de 1 a 8 metros, que es el alcance máximo de este.
- Tiempo de muestreo de 100 milisegundos/escaneo. Al ser bajo puede realizar un gran número de barridos para actualizar la información en tiempo real.
- Buena resistencia ambiental, entre los -10ºC y los 50ºC con una humedad del 85%.

Estas características y gran versatilidad hacen que el sensor Hokuyo URG-04LX-UG01 sea idóneo para aplicarlo a robótica móvil. [24]

Fig.52. *Sensor Hokuyo URG-04LX-UG01*

5.5. Móvil (opcional)

El robot móvil al que se podría aplicar el sensor y la cámara UVC será la plataforma móvil RBCAR de Robotnik, entre otros muchos tantos, que emplea una cinemática Ackerman de tracción. Esta cinemática es controlada por un motor AC a través de un encoder incremental y un encoder absoluto.

A su vez debido a la disposición de un cajón trasero y su estructura mecánica, puede llevar cargas (en este caso se ha colocado una pila de hidrógeno procedente de otro grupo de investigación). Con el conjunto de sensores instalados en el robot se puede navegar de forma autónoma o tele-operado por un mando con joystick, como si de un vehículo eléctrico se tratase.

A parte del conjunto mecánico muy similar a vehículos convencionales, como frenos de tambor, también puede montar accesorios propios de la empresa Robotnik, como sensores laser Hokuyo, Sick o DGPS entre otros.

Destacar la conectividad de dicho vehículo a través de puertos USB, con los que podremos acoplar de forma rápida y sencilla todo tipo de componentes. Finalmente señalar que RBCAR usa una arquitectura modular basada en ROS (como ya hemos visto). [33]

Fig.53. *Robot Móvil RBCAR*

5.5.1. Acople del sensor (opcional)

El sensor Hokuyo URG-04LX-UG01, deberá ir montando en el robot móvil, en este caso el RBCAR descrito en el apartado anterior. Para ello se insertó un acople metálico en el frontal del móvil.

Fig.54. *Acople del sensor en el frontal del RBCAR*

Gracias a esta pieza el sensor se puede acoplar a la perfección en el frontal del vehículo para recoger la información pertinente y detectar los objetos a una buena altura y con un rango de 6 a 8 metros de distancia.

Dicho acople sitúa al sensor a una altura final de 68 cm sobre el suelo. La forma del acople está dispuesta en posición angular a 28.5 º, y atornillada con cuatro arandelas y tuercas a un soporte metálico que cubre el sensor a la par que lo protege de impactos.

La base sobre la cual se atornilla el sensor tiene unas dimensiones de 10x10 cm y presenta cuatro agujeros realizados a 2.5cm de cada extremo y con un radio de 0.4 cm.

Fig.55. *Lateral del acople*

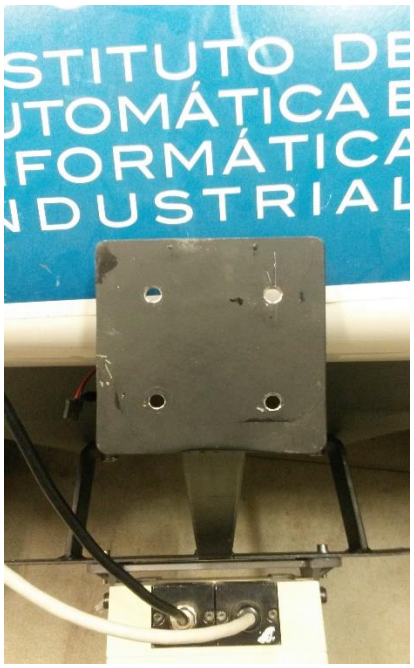


Fig.56. *Base del acople*

Para acoplar la cámara UVC no existe de momento ningún acople, pero se podría poner de forma opcional sobre el mismo sensor Hokuyo sin interferir en su campo de visión. En un futuro se podría insertar un acople específico para la cámara situándolo en una posición algo más elevada que la del sensor láser para no interferir en su campo de visión y a su vez pode detectar de forma más eficaz los códigos QR.

6. Justificación de la solución adoptada

6.1 Bases teóricas

El sistema de detección y seguimiento de objetos está formado por una parte de software y otra de hardware. La unión de ambas partes formará el sistema completo descrito en este proyecto.

Como resulta obvio, lo correspondiente a la parte de hardware se basa en: la computadora, el sensor Hokuyo URG-04LX-UG01 y la UVC 2.0 USB Logitech C920 C (de forma opcional en este proyecto, el robot móvil RBCAR para el cual ha sido diseñada, no como hardware pero si como parte física).

El conjunto de hardware podrá funcionar con dos tensiones.

- 5V de tensión continua
- 5V proporcionados por los puertos USB del computador, el cual es alimentado a su vez por 230 V de tensión continua.

El sensor Hokuyo URG-04LX-UG01, ha sido elegido dada su versatilidad, robustez, precisión, y modularidad de implementación con ROS. La camara UVC 2.0 USB C920 C de Logitech se ha elegido dada su gran resolución para poder detectar y seguir a una distancia considerable los patrones de los códigos QR.

En el lado del software podemos puntualizar la implementación del sistema de detección y seguimiento de imágenes, realizado para la cámara y para el sensor.

El control de la cámara se basará en un trazado de puntos según el código expuesto. Dichos puntos se ubicarán en los vértices de los objetos y estructuras. El código del tracking por keypoint a través de cámara hace llamadas a varias librerías, instaladas con anterioridad junto con ViSP, para hacer posible la detección y ubicación de los puntos, así como su posterior seguimiento. En segundo lugar se unirá ViSP con ROS a través del paquete vision_visp. Gracias a este paquete podremos realizar el seguimiento del patrón de un código QR con la cámara UVC, el cual se pondrá en la superficie del objeto a seguir.

Por el contrario el control para el sensor Hokuyo, se basará en la limitación del ángulo de escaneo del dispositivo para evitar saturar de información innecesaria al sistema y la configuración de una interface sencilla para el usuario. Los datos de medición obtenidos se mostraran por pantalla, con un error mínimo, oscilando entre 0.5 y 1 cm, del modelo medido de forma física con un metro.

Empleando ambos elementos, cámara y sensor, con sus implementaciones de control respectivas, obtenemos en conjunto la aplicación para detectar y seguir objetos. El conjunto es capaz de conocer la distancia a la que se encuentra nuestro vehículo o robot móvil del objeto y es capaz de poder detectar tanto su contorno, como seguir el objeto detectado gracias a la implementación el código QR.

6.2 Programación

Todo el código ha sido programados en C++, a su vez se ha hecho uso del terminal de Linux para la instalación de ROS y ViSP, así como los diversos paquetes y librerías, la supervisión y puesta a punto de la calibración de la cámara, y la detección y el seguimiento de objetos de la misma. Este lenguaje de programación, no ha sido elegido ya que desde su instalación es uno de los requeridos por ROS y VISP. La otra opción que se dejaba era la de implementar los códigos con Python, pero se eligió C++ por su familiaridad y claridad, además de ser un lenguaje mucho más amplio y empleado. [34]

El código está dividido en tres partes diferenciadas. La primera se corresponde a la instalación tanto de ROS como de ViSP. La segunda se corresponde a los paquetes y librerías de los tutoriales, proporcionados al realizar los comandos requeridos, encontrados en la "Wiki" de cada uno de los respectivos programas. Finalmente la tercera parte se corresponde a ViSP, donde se centra el proyecto en el Blob y tracking de imágenes de la cámara UVC Logitech C920 C y su posterior unión de ViSP con ROS, finalmente ROS, con el que programaremos el sensor Hokuyo URG-04LX-UG01 para medir la distancia hacia el objeto y el ancho del mismo.

En la primera parte se realiza la instalación de ROS, conectando correctamente y según indica en la página web los puertos Ethernet y servidores universales. Posteriormente se realiza la instalación con los comandos "sudo apt" añadiendo las librerías y paquetes pertinentes, descritos y dados en la página oficial de ROS.

Una vez acabemos la instalación de ROS, procederemos con la de ViSP del modo en el que se describe en el apartado 2.7.1 Instalación de ViSP, cuya información proviene de la página oficial de ViSP.

En la segunda parte instalaremos los paquetes y librerías requeridas por los tutoriales, a su vez se ejecutaran los mismos y se comprobara el correcto funcionamiento de estos. Sin el correcto funcionamiento y comprobación de los tutoriales, librerías y paquetes no será posible realizar el control de la cámara y del sensor, por lo tanto en esta parte deberemos prestar especial atención a los mensajes (warnings) y posibles fallos para evitar complicaciones en el control.

Por último, en la tercera parte, será donde se realizará el control tanto de la cámara UVC como del sensor Hokuyo.

Como ya se ha descrito anteriormente, el control del sensor Hokuyo se implementará con ROS, mientras que el control de la cámara UVC será a través de ViSP y posteriormente se añadirá otra aplicación con ViSP dentro de ROS. El desarrollo de los controles los podemos observar en los apartados 5.2.1.1 y 5.2.1.2 respectivamente.

7. Conclusión

Una vez finalizado el Proyecto presente, podemos decir que se ha cumplido con el objetivo final del mismo, el de realizar el control para detectar y seguir objetos. Podemos señalar que los objetivos propuestos se han conseguido satisfactoriamente y con una buena resolución.

Para la obtención de dichos resultados finales, se ha comenzado por lo más básico, una breve introducción de los sensores en la automoción, la introducción del mundo de la robótica, con la que se ha avanzado rápidamente en el sector industrial en las últimas cinco décadas, posteriormente un breve descripción de la visión artificial y sus

principales conceptos y como han ayudado a mejorar los sistemas robóticos y la recolección de información a través de imágenes. Una vez hemos puesto en situación la trayectoria de la robótica y los sistemas de visión, nos centramos en el desarrollo de la programación de la aplicación para la detección y seguimiento de objetos.

Durante todo el proyecto se han ido explicando las características de los dos dispositivos: cámara UVC y sensor laser Hokuyo. También se ha explicado cómo realizar correctamente la instalación de ROS y ViSP para evitar problemas en la ejecución del programa de control, así mismo se ha fraccionado todo el control en pequeños tutoriales y programas guiados a través de imágenes y comandos paso por paso. Como refuerzo para evitar confusiones ante los posibles fallos o errores que nos pudieran aparecer se ha realizado una ayuda para solventarlos de la forma más cómoda y rápida.

Como se ha descrito en el documento, se han realizado dos tipos de control, a pesar de que ambos trabajan para la detección y el seguimiento de objetos.

El primero implementado con ROS para el sensor Hokuyo URG-04LX-UG01. Con este control como hemos observado, podemos conocer el ancho y la distancia a la que se encuentra el sensor y por lo tanto el móvil de un objeto dentro de un rango de 8 metros. Este rango como hemos visto se encuentra en una zona media y la precisión es más que aceptable, puesto que el sensor laser permite un error máximo de 3 cm, habiendo conseguido reducirlo a 0.5 cm, por lo que la calibración se ha realizado de forma perfecta a través de ROS y testeándolo por RViz. Por último se realizaron pruebas con diferentes objetos y a distintas distancias, comprobando que los resultados eran los esperados: gran velocidad de cálculo y muestreo, buena resolución de medida con un bajo error y un alcance excelente, limitado en este proyecto por comodidad de las mediciones.

El segundo control, esta vez implementado con el programa complementario de ROS, ViSP, se aplica a la cámara web UVC 2.0 USB C920 C de Logitech. El principal objetivo de este control es detectar el contorno de los objetos empleando para ello patrones de puntos que se ubican en los vértices de dichos objetos, siguiéndolos en un plano 2D del espacio. Para ello ha sido necesario calibrar la cámara, configurar la resolución y probar el programa mixto de tracking y keypoint para una cámara en tiempo real. Para comprobar su correcto funcionamiento se han grabado varios objetos moviéndolos en varias direcciones y en distintas distancias, obteniendo resultados satisfactorios y con una buena resolución de cálculo de puntos. Como modificación sobre esta implementación se unió ViSP con ROS para terminar el auto tracker por el seguimiento del patrón de código QR, realizando pruebas sobre algunos móviles y mostrando su correcto funcionamiento. La comprobación se ha realizado tanto en tiempo real mostrando la información pertinente por pantalla como una gráfica implementada

para Matlab y que su lectura sea más fácil y cómoda, a la hora de contrastar información sobre la trayectoria del móvil que ha seguido.

No podemos olvidar las dificultades halladas durante la realización de este proyecto, pudiendo destacar los direccionamientos de librerías y paquetes, ocasionados por una mala instalación de ViSP y en alguna ocasión una mala realización de los ejecutables. Todo esto solucionado revisando la instalación del mismo y los direccionamientos.

También podemos destacar algunos problemas de conectividad con ROS y RViz en la computadora, esta vez ocasionados por versiones anteriores, lo cual se solucionó fácilmente instalando los paquetes y librerías actualizadas.

Finalmente se puede decir que se ha cumplido el objetivo del proyecto, pues como hemos podido observar, los resultados obtenidos en las pruebas prácticas después de la implementación teórica de la programación han funcionado obteniendo los resultados esperados.

Como proyectos futuros podemos usar el potencial de esta aplicación para aplicarlo a vehículos móviles, como convoyes de camiones de reparto en carretera, donde el primer camión que sería la cabeza del grupo contaría con el sensor laser y la cámara UVC para la detección de los objetos y el control de distancia a los mismos para evitar colisiones, mientras que el resto de camiones seguirían al que va en primer lugar, gracias a un código QR puesto en la parte posterior del mismo. Otro ejemplo seria en minería, donde se necesitan procesos repetitivos de carga para camiones, donde se volvería a aplicar la idea anterior.

A su vez, esta aplicación no solo debe limitarse a vehículos y robots móviles, puesto que con la implantación de código QR podríamos revisar y optimizar el movimiento de brazos robots al poder seguir su posición punto por punto con gran precisión.

8. Anexos

8.1 Documentación Técnica

ÍNDICE

Date:2009.8.20

Scanning Laser Range Finder URG-04LX-UG01 (Simple-URG)

Specifications

Symbol	Amended Reason			Pages	Date	Corrector	Amendment No
Approved by	Checked by	Drawn by	Designed by	Title	Scanning Laser Range Finder URG-04LX-UG01 Specifications		
MORI	MAEDA	YAMAMOTO	MORI	Drawing No.	C-42-3635		1/4

HOKUYO AUTOMATIC

1. General

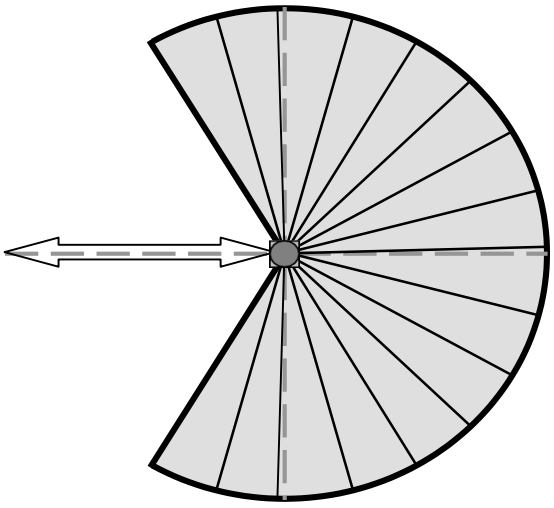
URG-04LX-UG01 is a laser sensor for area scanning. The light source of the sensor is infrared laser of wavelength 785nm with laser class 1 safety. Scan area is 240º semicircle with maximum radius 4000mm. Pitch angle is 0.36º and sensor outputs the distance measured at every point (683 steps). Laser beam diameter is less than 20mm at 2000mm with maximum divergence 40mm at 4000mm.

Principle of distance measurement is based on calculation of the phase difference, due to which it is possible to obtain stable measurement with minimum influence from object's color and reflectance.

Non-radiated area: 120º

Detection Area: 240º

Max. Distance: 4000mm



Power: 5v DC

(USB buspower)

Figure 1

Note

Figure 1 shows the detectable area for white Kent sheet (70mm×70mm). Detection distance may vary with size and object.

2. Important Notice

This sensor is designed for indoor use only.
This sensor is not a safety device/tool
This sensor is not for use in military applications
Read specifications carefully before use.

Title	URG-04LX-UG01 Specification	Drawing No	C-42-3635	2/4

3. Specifications

Product Name	Scanning Laser Range Finder
Model	URG-04LX-UG01
Light source	Semiconductor laser diode (λ=785nm), Laser safety Class 1 (IEC60825-1)
Power source	5V DC ±5% (USB buspower)
Current consumption	500mA or less (Rush current 800mA)
Detection distance	20mm ~ 4000mm
Accuracy	Distance 20mm ~ 1000mm : ±30mm* Distance 20mm ~ 4000mm : ±3% of measurement*
Resolution	1 mm
Scan Angle	240°
Angular Resolution	0.36°
Scan Time	100msec/scan
Interface	USB Version 2.0 FS mode (12Mbps)
Ambient (Temperature/Humidity)	-10 ~ 50ºC / 85% or less (without dew and frost)
Preservation temperature	-25 ~ 75ºC
Ambient Light Resistance	10000Lx or less
Vibration Resistance	Double amplitude 1.5mm 10 ~ 55Hz, 2 hours each in X, Y and Z direction, and 98m/s² 55Hz ~ 150Hz in 2 minutes sweep, 1 hours each in X, Y and Z direction
Impact Resistance	196 m/s², 10 times each in X, Y and Z direction
Protective Structure	Optics : IP64 Case : IP40
Insulation Resistance	10MΩ for DC 500Vmegger
Weight	Approx. 160 g
Case	Polycarbonate
External dimension (W×D×H)	50×50×70mm (Reference design sheet No.3502)

*Under standard test conditions with white Kent sheet 70mm×70mm

4. Quality reference value

Operating Vibration resistance	19.6m/s², 10Hz ~ 150Hz with 2 minutes sweep, 0.5 hours each in X, Y and Z direction
Operating Impact resistance	49 m/s², 10 times each in X, Y and Z direction
Angular Speed	360 deg/s
Angular Acceleration	π/2 rad/s²
Life	5 years (Varies depending upon the operating conditions)
Sound level	25db or less (at 300mm)
FDA	This product complies with 21 CFR parts 1040.10 and 1040.11. (Accession Number 0521258-002)

5. Interface

CN USB-mini (5 Pin)
Cable is not included. Use commercially available compatible unit.

Note:
Refer specifications number C-42-3320B for communication protocol (SCIP2.0).

6. Notice:
Supply voltage is DC 5Volts. Sensor will damage if high voltage is supplied.

USB buspower may not be sufficient during the start up of URG-04LX-UG01 in some PCs. Use the auxiliary cable supplied with the sensor to avoid the problem.

The maximum data step is 683 points. Sensor's angular resolution is 0.3515625º (360º /1024 steps) and angular range is 239.765625º ((683-1)×360/1024)

Angular resolution can be specified form the host. Read communication protocol specification (No C-42-3320B) for details.

USB driver is communication device class (CDC) supported by standard operating system. The device is connected as a COM port with the same utility.

Plug and play function is not supported.

Title	URG-04LX-UG01 Specification	Drawing No	C-42-3635	4/4

Face-to-face communication for ultimate

Cisco abber™ collaboration

The Logitech C920-C Webcam

The Logitech C920-C Webcam, designed exclusively for Cisco®, is the professional webcam that seamlessly integrates with Cisco solutions to deliver true-to-life HD 1080p[1] quality video. With a 78-degree field of view, UVC H.264 encoding and omnidirectional mics, the C920-C brings high-quality desktop face-to-face collaboration to UC.

Features:

3. HD 1080p[1] video at up to 30 frames per second
4. UVC H.264 encoding technology
5. 78-degree field of view with true widescreen
6. Logitech RightLight™ 2 technology and autofocus
7. Omnidirectional dual stereo microphones

- Convenient privacy shutter
- Multiple mounting options
- USB plug-and-play set up
- Cisco® Compatible[2]

cisco Compatible

Logitech UC for Real

DATA SHEET

Logitech C920-C Webcam

Product Specifications

Price $119.99
Part # 960-000945

Without clip:
Width 5 inches (126mm) Height 1.2 inches (29mm) Depth 1.3 inches (32mm)

With clip:
Width 5 inches (126mm) Height 2.9 inches (73mm) Depth 1.8 inches (45mm)

Weight 6.0 ounces (170g)

System Requirements

- Windows® 7
- Mac OS® 10.6 or higher
- SUSE Linux® 11 SP2 (for Cisco® VXC6215)
- USB Port

Warranty

3-year limited hardware

Inside the Box

C920-C Webcam
Quick Start Guide

FEATURE SPOTLIGHT

HD 1080p[1] video at up to 30 frames per second

Make a vivid impression with true-to-life, high definition video calls.

UVC H.264 encoding technology Frees up system bandwidth by putting video processing within the camera.

78-degree field of view with true widescreen

See more without having to reposition the camera or crop and zoom.

Logitech RightLight™ 2 technology and autofocus

Webcam intelligently adjusts to improve visual quality in low light and backlit situations at multiple distances.

Omnidirectional dual stereo microphones Designed to capture sound for clear communication in larger spaces like offices and small conference rooms.

Convenient privacy shutter

Easy lens closure for added privacy and security when the webcam is not in use.

Multiple mounting options

Freedom to mount the camera wherever it works best – LCD screen, notebook, or tabletop – by using the attached clip or the tripod embedded thread.

USB plug-and-play set up

Easy to set up and easy to use.

Cisco® compatible[2]

Designed exclusively for Cisco® to work seamlessly.

Logitech
Tools Designed for the Anywhere Workplace

9. Bibliografía

[1] Historia de los sensores en la automoción: http://mecatronico45.blogspot.es/tags/sensor/

[2] ABS Sistema antibloqueo de frenado, Wikipedia: https://es.wikipedia.org/wiki/Sistema_antibloqueo_de_ruedas

[3] ESP control de estabilidad, Wikipedia: https://es.wikipedia.org/wiki/Control_de_estabilidad

[4] Historia de la Robótica: http://robotiica.blogspot.com.es/2007/10/historia-de-la-robtica.html

[5] Historia de la Robótica, Wikipedia: https://es.wikipedia.org/wiki/Rob%C3%B3tica

[6] Clasificación de Robots: http://robotec11.tripod.com/id4.html

[7] Tipos de Robots: https://informaticoscarvajal.wordpress.com/robotica/clasificacion-de-los-robots/

[8] Apuntes de Visión Artificial, 4º Grado de Ingeniería Electrónica, José Antonio Bernabé Chumillas y Carlos Ricolfe Viala.

[9] Tecnología LiDAR, Wikipedia: https://es.wikipedia.org/wiki/LIDAR

[10] Lo básico sobre LiDAR: http://www.tecnicayterritorio.com/2012/04/lo-mas-basico-que-deberias-saber-sobre.html

[11] Aplicaciones de la teledetección laser (LiDAR): http://ambiental.cedex.es/docs/Ingenieria-Civil-142-2006-Aplicaciones-LiDAR.pdf

[12] Sensor Láser de telemetría: http://wiki.robotica.webs.upv.es/wiki-de-robotica/sensores/sensores-proximidad/sensor-laser/

[13] Programming Robots with ROS, de Morgan Quigley, Brian Gerkey y William D.Smart, 2013

[14] Ubuntu Install for ROS: http://wiki.ros.org/indigo/Installation/Ubuntu

[15] RViz Install: http://wiki.ros.org/rviz/UserGuide

[16] ViSP: https://visp.inria.fr/

[17] Install ViSP: https://visp.inria.fr/install/

[18] Sistema Operativo Windows y sus características: http://www.informaticamoderna.com/Mic_Win.htm

[19] Sistema Operativo OS y sus características: https://es.wikipedia.org/wiki/Mac_OS

[20] Sistema Operativo Linux y sus características: https://es.wikipedia.org/wiki/GNU/Linux

[21] OROCOS: http://www.orocos.org/

[22] Sensores Hokuyo: http://www.hokuyo-aut.jp/02sensor/index.html#scanner

[23] Sensor Hokuyo UTM-30LX http://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html

[24] Sensor Hokuyo URG-04LX-UG01: http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx_ug01.html

[25] Sensor Hokuyo URG-04LX: http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx.html

[26] Paquete instalador sensor Hokuyo URg-04LX-UG01: http://wiki.ros.org/rviz/UserGuide

[27] How to use Hokuyo Laser Scanners: http://wiki.ros.org/hokuyo_node/Tutorials/UsingTheHokuyoNode

[28] Tutorial de instalación: http://visp-doc.inria.fr/doxygen/visp-3.0.0/index.html#tuto_install_sdk

[29] Tutorial inicio y CMake de ViSP http://visp-doc.inria.fr/doxygen/visp-3.0.0/tutorial-getting-started.html

[30] Tutorial Blob Tracking de ViSP: http://visp-doc.inria.fr/doxygen/visp-3.0.0/tutorial-tracking-blob.html

[31] Tutorial Keypoint tracking de ViSP: http://visp-doc.inria.fr/doxygen/visp-3.0.0/tutorial-tracking-keypoint.html

[32] Cámara C920C de Logitech: http://support.logitech.com/en_gb/product/c920-c-webcam

[33] RBCAR de Robotnik: http://www.robotnik.es/robots-moviles/rbcar/

[34] Learning ROS for Robotics Programming, de Aaron Martinez y Enrique Fernández.

[35] ViSP auto tracker: http://wiki.ros.org/visp_auto_tracker?distro=indigo

[36] Vision ViSP: http://wiki.ros.org/vision_visp

[37] Guía de Presupuestos CTT UPV

10. Presupuesto

10.1 Necesidad del presupuesto

Antes de comenzar con el propio presupuesto, nos encontramos en la necesidad de justificar la existencia de este. Como es necesario cuantificar, tanto la mano de obra que se ha invertido en dicho proyecto, como la creación de un inventario de los materiales empleados para el mismo. Dado que todo proyecto de ingeniería debe concluir con su presupuesto, lo realizaremos a pesar de que este documento tiene un carácter educativo dentro del ámbito de investigación, ya que en un futuro podría ser desarrollado e implementado en robot móviles y vehículos, y de ahí la necesidad de conocer el coste económico que supondría desarrollarlo.

El presupuesto ha sido diseñado y elaborado a partir del documento "Recomendaciones en la elaboración de presupuestos en actividades I+D+I" del Centro de Apoyo a la Innovación, la Investigación y la Transferencia de Tecnología (CTT), en su versión para el año 2015. Hemos cambiado la documentación de presupuestos estándar en ingeniería por esta, ya que se trata de un proyecto de investigación a pequeña escala y ha sido considerado más acorde para el mismo. [37]

10.2 Estudio económico

10.2.1. Coste de Personal

Para el cálculo de los costes del personal se ha empleado la siguiente formula:

Coste de personal (€)= Coste por hora (€/h) x Número de horas empleadas (h)

Dentro de este coste por hora ya tenemos considerado los gastos de la Seguridad Social, las Indemnizaciones y los costes indirectos.

Para el cálculo del coste por hora, y considerando el documento anteriormente citado, presentamos la siguiente tabla que nos ofrece la información respecto al personal. Tomaremos como referencia el salario correspondiente a un titulado medio (nosotros mismos), a pesar de que aparece un salario máximo (36.2 €/h) y mínimo (26.8€/h), nosotros tomaremos un valor medio entre ambos para calcular el presupuesto, en este caso 31.5 €/h.

Con esta información, obtenemos la siguiente tabla:

Acción	Tiempo (h)	Coste (€/h)	Total (€)
Instalación Ubuntu 14.04	**2**	**31.5**	**63**
Instalación ROS	**20**	**31.5**	**630**
Instalación ViSP	**20**	**31.5**	**630**
Implementación control sensor	**25**	**31.5**	**787.5**
Implementación control cámara	**50**	**31.5**	**1575**
Descarga de paquetes y librerías	**10**	**31.5**	**315**
Calibración cámara y sensor	**25**	**31.5**	**787.5**
Pruebas del proyecto	**49**	**31.5**	**1543.5**
Redacción de documentos	**99**	**31.5**	**3118.5**
Total	**300**		**9450**

Si este coste lo desglosamos según los porcentajes correspondientes a la Seguridad Social (32.1%), Indemnizaciones (3,04%) y costes indirectos (16.5 €/h), obtenemos la siguiente tabla:

Concepto	Precio (€)
Honorarios	**1179.27**
Seguridad Social	**3033.45**
Indemnizaciones	**287.28**
Costes Indirectos	**4950**
Total	**9450**

10.2.2. Material Inventariable

La fórmula empleada para el cálculo del material de inventario también está presente en el documento de CTT:

$$\text{Coste material inventariable} = \frac{Número\ de\ meses\ de\ uso}{12\frac{meses}{año}\ x\ Periodo\ de\ amortización}\ x\ Coste\ equipo\ x\ Porcentaje\ uso\ equipo$$

La referencia para el periodo de amortización se ha vuelto a tomar del CTT, el cual define que para equipos didácticos y de investigación un periodo de 10 años, mientras que para el equipo informático y de software empleado nos da una referencia de 6 años.

El porcentaje de uso que se ha tomado ha sido en horas de tarea realizada, en las que el equipo o programa informático ha sido utilizado un número de horas totales en el proyecto.

Considerándolo, obtenemos la siguiente tabla:

Equipo	Tiempo de uso (meses)	Años amortización	Coste (€)	Porcentaje de uso	TOTAL (€)
Ordenador Portátil Asus	4	6	750	100	41.66
Ubuntu 14.04	4	6	0	65	0
ROS Indigo	4	6	0	35	0
Simulador RViz	4	6	0	10	0
VISP	4	6	0	55	0
Sensor Hokuyo URG-04Lx-UG01	1	10	1150	12	1.15
Camara UVC 2.0 USB C920 C Logitech	1	10	110	13	0.119
TOTAL					42.93

10.2.3. Material Fungible

En este apartado se considera todo el material de corta vida útil empleado en el proyecto.

Concepto	Precio (€)
Impresión	3.5
Encuadernación	5
Folios (500 uds)	4.65
TOTAL	13.15

10.2 Estudio económico

Concepto	TOTAL (€)
Mano de Obra (300h)	9450
Material Inventariable	42.93
Material Fungible	13.15
Subtotal	9506.08
IVA (21%)	1996.27
TOTAL	11502.36