# Capítulo 3

# Librerías Software

## 3.1. Introducción

En la introducción de este texto ya se ha indicado que la robótica aglutina conocimientos y aplicaciones de ramas muy diversas de la ciencia y la tecnología. En este sentido, la investigación y el desarrollo de nuevos robots o técnicas de navegación se hace una tarea excesivamente compleja.

Es por ello que en en los proyectos de robótica se utilizan de forma genérica o habitual multitud de desarrollos disponibles (ya sean totalmente libres o soluciones privativas, hardware o software) que facilitan la consecución de los objetivos.

Realizamos en esta introducción un breve repaso a algunas de las librerías y middleware más utilizados en la programación de sistemas robóticos, estableciendo una clasificación según la funcionalidad de los mismos según se utilicen en comunicaciones, interfaces gráficas o visión por computador y justificando la elección concreta de cada librería utilizada en el proyecto.

### 3.1.1. Comunicaciones

La percepción cooperativa requiere capas de comunicación adecuados para compartir información. Algunos de los desarrollos más conocidos son:

- **MIRO [5].** Es un framework distribuido orientado a objetos para el control de robots móviles basado en la tecnología CORBA (Common Object Request Broker Architecture). Los componentes básicos de MIRO se han desarrollado en C++ para Linux. Pero debido a la independencia lenguaje de programación de componentes del sistema CORBA se puede escribir en cualquier lenguaje de programación y en cualquier plataforma que proporciona implementaciones CORBA.

- **PLAYER/STAGE [7].** Se ha convertido en un estándar de facto en la comunidad opensource robótica. El servidor Player se utiliza como capa de abstracción de hardware que se puede utilizar con muchos robots reales o simulados, e incluye servicios de comunicación a través de sockets TCP que permiten la intercomunicación.

- **YARP [10][20].** Yet Another Robot Platform es otro middleware de amplia aceptación para el desarrollo de sistemas robóticos. Escrito y mantenido en lenguaje C++ es un sistema multiplataforma, soporta computación distribuida y está orientado principalmente para el control en robots de manera eficiente.

En el proyecto URUS se optó por la utilización de YARP dada su facilidad para su integración con el código escrito en C++, su modularidad, fiabilidad probada en anteriores proyectos del GRVC así como por supuesto ser una herramienta totalmente libre. El siguiente apartado de la memoria se realizará una descripción exhaustiva de esta librería.

### 3.1.2. Interfaces gráficas

Las interfaces gráficas de usuario aparecen como una evolución natural de los intérpretes de comandos utilizados para interactuar con los primigenios sistemas operativos, a pesar de no haber sido capaces de eliminar completamente los intérpretes de comandos en los actuales sistemas, los cuales siguen siendo muy demandados y teniendo un peso específico realmente importante en según qué contextos.

A día de hoy, podemos distinguir varias líneas claramente establecidas, con sus cuotas de mercado correspondientes en el complejo mundo de los entornos gráficos. Actualmente podemos distinguir los entornos gráficos (típicamente llamados de escritorio) de los sistemas Microsoft Windows, el sistema X-Window utilizado en sistemas operativos GNU/Linux, e incluso el Aqua utilizado en Mac OS X de Apple, entre otros menos generalistas.

Tal y como se ha comentado en el capítulo 2, el robot Romeo-4R está soportado por un sistema operativo GNU/Linux con un entorno gráfico por tanto basado en X Window, concretamente, el entorno de escritorio KDE y por lo tanto, para el desarrollo de la interfaz gráfica de usuario las opciones se reducían inicialmente a las librerías para sistemas X Window.

Sin duda las librerías más populares para X Window System son Qt y GTK.

- **Qt [11].** Librería multiplataforma, desarrollada inicialmente por la empresa Trolltech, orientada principalmente al diseño de interfaces de usuario, aunque también admite desarrollo software sin interfaces gráficas.

- **GTK [4].** The GIMP Toolkit es un conjunto de librerías multiplataforma para desarrollar interfaces gráficas de usuario. Dichos desarrollos se integran de forma natural para los entornos gráficos GNOME, XFCE y ROX, ya que han sido implementados con estas misma librería, aunque también se puede usar en el escritorio de Windows, MacOS y otros. Inicialmente fueron creadas para desarrollar el programa de edición de imagen GIMP, sin embargo actualmente se usan bastante por muchos otros programas en los sistemas GNU/Linux. Junto a Qt es una de las librerías más populares para X Window System.

Ambas permiten programar con múltiples lenguajes como C, C++, C#, Ruby, Perl, PHP, Python, ..., ofreciendo ambas además, una serie de clases intuitiva y que incluyen la mayoría de elementos clásicos en las interfaces de usuario.

Se decidió utilizar finalmente la librería Qt ya que, a pesar de que ambas pueden utilizarse en sistemas KDE, el presente en el PC de control de Romeo-4R, la integración de aplicaciones Qt en KDE es mucho más sencilla y natural, al estar el propio entorno KDE desarrollado con esta librería. La situación o relación análoga se da entre los entornos de escritorio GNOME y XFCE con las aplicaciones escritas en GTK.

Además, el desarrollo de la aplicación sirvió como testeo y toma de contacto de las posibilidades de la propia librería Qt, ya que previos desarrollos y trabajos realizados en el GRVC habían sido realizados con GTK.

### 3.1.3. Visión por Computador

La visión por computador es una rama de la ciencia y de la técnica que podría definirse como las técnicas o procedimientos que nos permiten extraer información del mundo físico a partir de imágenes, utilizando para ello la ayuda de un computador.

A lo largo de los años se han desarrollado numerosas librerías software que sirven de apoyo o como punto de partida para implementar todas estas técnicas en diferentes tipos de proyectos, entre ellos los relacionados con la robótica. Algunas de estas librerías son:

- **OpenCV [6].** Es una librería libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos.
- **VXL [9].** Visión-X-Library es una colección de librerías de C++ diseñada para la investigación y la aplicación de visión por computador. Fue creado a

partir de TargetJr y el IUE, con el objetivo de hacer un sistema ligero, rápido y consistente. VXL está escrito en ANSI/ISO C++ y está diseñado para ser portado a muchas plataformas.

- **CAMELLIA [1].** Es una librería de código abierto para procesamiento de imágenes y visión por computador. Escrita en C, es multiplataforma (Unix/Linux, Windows) y robusto. Incluye una gran cantidad de funciones para el tratamiento de imagen (filtrado, matemáticas, morfología, etiquetado, deformación, etc.), la mayoría de ellos altamente optimizadas. Al utilizar la estructra CamImage/IplImage para describir las imágenes, es un buen reemplazo a la popular, pero suspendida de Intel IPL y un buen complemento a la librería OpenCV.

En el proyecto URUS se ha utilizado OpenCV por ser sin ningún tipo de dudas la librería de referencia en visión por computador actualmente. Además su integración con el resto de librerías en C++ es un trabajo relativamente sencillo.

Como hemos comentado, en los siguientes apartados haremos una descripción más exhaustiva de las principales librerías utilizadas en el proyecto URUS.

## 3.2. YARP

YARP [10] es un framework de código abierto, multiplataforma, escrito casi en su totalidad en C++, que soporta computación distribuida, orientado principalmente para el control en robots de manera eficiente.

**Figura 3.1: Logo YARP.**

Se trata de un conjunto de librerías, protocolos y herramientas para mantener los módulos y dispositivos limpiamente desacoplados.

Se trata de un middleware que básicamente da soporte a comunicaciones de datos, ya sea genéricos propios o datos típicos usados en robótica. No se trata por tanto de ningún sistema operativo. YARP está escrito por y para investigadores en robótica, en especial la robótica humanoide, en la que se debe trabajar con hardware diverso y complicado de controlar así como con una torre de módulos software igualmente complicados de manejar y comunicar.

YARP fue desarrollado como una herramienta para facilitar todo este trabajo, inicialmente a sus propios desarrolladores y posteriormente, al ser liberado al resto

de desarrolladores y trabajadores en robótica.

Los componentes principales de la librería YARP se pueden desglosar en:

- **libYARP_OS.** Interfaz con el sistema operativo para apoyar fácilmente la transmisión de datos a través de muchos hilos y/o través de muchas máquinas. YARP está escrito para ser independiente del sistema operativo utilizado, y ha sido utilizado en Linux, Windows, Mac OSX y Solaris. YARP utiliza el código abierto de la librería ACE (Adaptive Communication Environment), que es portable a través de una gama muy amplia de entornos, y de la que YARP hereda la portabilidad.

- **libYARP_sig.** Para realizar tareas comunes de procesamiento de señales (visual, auditiva) de manera abierta, facililitando la conexión con otras librerías de uso común, por ejemplo OpenCV.

- **libYARP_dev.** Interfaz con los dispositivos comunes usados en robótica: Framegrabbers, cámaras digitales, paneles de control de motores, etc.

Estos componentes se mantienen por separado. El componente básico es libYARP_OS, que deberá estar disponible antes de que otros componentes pueden ser utilizados.

Para un funcionamiento correcto en tiempo real, la sobrecarga de la red debe reducirse al mínimo, por lo que YARP está diseñado para funcionar en una red aislada o detrás de un firewall, pudiéndose obtener resultados insatisfactorios en caso contrario.

Para interactuar con el hardware, YARP depende de los sistemas operativos específicos a las que las empresas de hardware dan soporte. Actualmente pocas empresas hardware facilitan el código fuente de sus drivers, algo que limita el avance de los sistemas abiertos.

La librería libYARP_dev se estructura para interactuar fácilmente con el código proporcionado por los fabricantes de hardware, a la vez que protege el sistema de dicho código, facilitando posteriores cambios de hardware.

En consecuencia, YARP tiene tres niveles de configuración: sistema operativo, hardware, y el nivel de robot. El primer nivel de configuración debe referirse sólo si se va a compilar YARP en un sistema operativo nuevo.

El segundo nivel es el hardware. Una nueva adición en una plataforma existente o una plataforma completamente nueva puede requerir la preparación de algunos drivers YARP de dispositivos. Estos son a todos los efectos clases C++ que soportan los métodos de acceso al hardware que normalmente se implementa a través de

llamadas a función a lo proporcionado por el proveedor de hardware. Esto viene típicamente en la forma de una DLL o una librería estática.

Por último, se pueden preparar los archivos de configuración de una plataforma robótica completamente nueva.

### 3.2.1. Definiciones

La librería YARP soporta la transmisión de un flujo de información del usuario a través de diferentes protocolos: TCP, UDP, MCAST (multi-cast), y memoria compartida, aislando al desarrollado de los detalles intrínsecos a la tecnología y protocolos de la red usada.

Estos protocolos de bajo nivel se denominan *Carriers* o *Portadores* para distinguirlos de los protocolos de más alto nivel. Cada conexión se lleva acabo usando un protocolo y/o red física diferente. El uso de diferentes protocolos permite explotar sus mejores características en cada caso:

- *TCP:* Garantiza la recepción de los mensajes.
- *UDP:* Más rápido que TCP pero sin garantías.
- *Multicast:* Eficiente para distribuir el mismo flujo de información a múltiples destinos.
- *Memoria compartida:* empleada para conexiones locales (seleccionada automáticamente siempre que sea posible, sin necesidad de la intervención del programador).

Si los mensajes siguen las directrices YARP, entonces pueden ser convertidos a conexión en modo texto, para facilitar la monitorización humana y la intervención en el sistema.

Debemos comentar también que el uso de TCP puede presentar problemas de rendimiento en comunicaciones inalámbricas en exteriores, típicamente con los estándares IEEE 802.11 a/b/g.

Para los propósitos de YARP, la comunicación se produce a través de *conexiones* entre entidades con nombre denominadas *Puertos* (Ports). Todo junto forma un grafo dirigido llamado *YARP network* (o *Red YARP*) donde los puertos son los nodos y las conexiones son los arcos.

A cada Puerto se le asigna un nombre único (como */motor/wheels/left*). Todos los Puertos se registran por su nombre en un servidor de nombres denominado *Servidor YARP*. El objetivo es asegurar que si el nombre de un Puerto es conocido, lo

es también todo lo necesario para comunicarte con él desde cualquier máquina.

El propósito de los Puertos es mover *Contenido* (*Content*) (secuencias de Bytes que representan información de usuario) desde un hilo a otro (u otros) a través de las fronteras de los procesos y las máquinas.

El flujo de información puede ser manipulado y monitorizado externamente (por ejemplo desde la línea de comandos) en tiempo de ejecución. En otras palabras, los arcos en la red YARP son totalmente modificables. Un puerto puede enviar Contenido a una cantidad indefinida de Puertos diferentes.

Un puerto puede recibir Contenido de una cantidad indefinida de Puertos diferentes. Si un puerto ha sido configurado para enviar Contenido a otro, se dice que tienen una Conexión.

Las Conexiones pueden ser añadidas o eliminadas libremente y pueden utilizar diferentes Portadores.

El servidor de nombres YARP es un servidor que almacena la información sobre los puertos. Ordena la información por nombre, jugando un papel análogo a los servidores DNS en internet. Para comunicarse con un puerto, las propiedades de dicho puerto necesitan ser conocidas (la máquina en que está ejecutándose, el socket en el que escucha, los portadores que soporta).

El servidor de nombres YARP ofrece el lugar conveniente para almacenar estas propiedades, de manera que sólo es necesario el nombre del puerto para recuperarlas.

### 3.2.2. Propiedades de una red YARP

Una red YARP consiste en las siguientes entidades: un conjunto de puertos, un conjunto de conexiones, un servidor de nombres y un conjunto de registros.

- Todos los puertos tienen un nombre único.
- Toda conexión tiene un Puerto fuente y un puerto objetivo.
- Cada Puerto mantiene una lista de todas las conexiones para las cuales es el puerto objetivo.
- Cada Puerto mantiene una lista de todas las conexiones para las que es el puerto fuente.
- Sólo hay un servidor de nombres en cada red YARP.

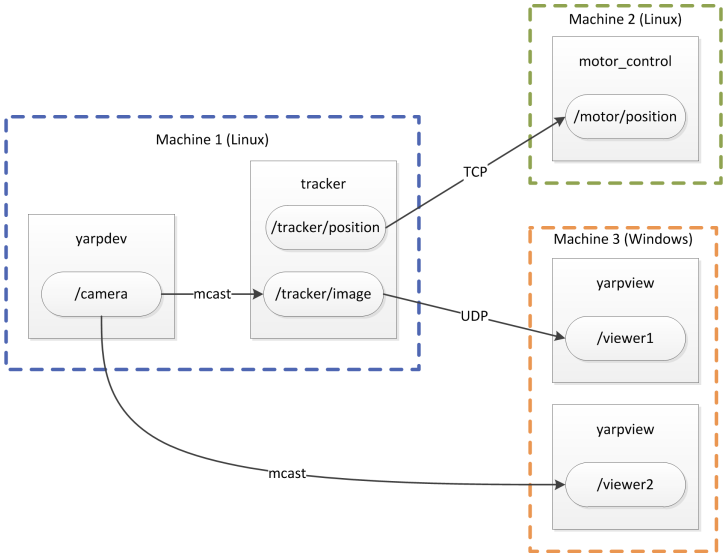


**Figura 3.2: Red YARP en varias máquinas y sistemas operativos.**

- El servidor de nombres mantiene una lista de registros. Cada registro contiene información sobre cada Puerto, identificados por su nombre.

- La comunicación dentro de una red YARP puede ocurrir entre dos Puertos, entre un Puerto y el servidor de nombres, entre un Puerto y una entidad exterior y entre el servidor de nombres y una entidad exterior.

- La comunicación entre dos Puertos ocurren si y sólo si hay una conexión entre ellos. Esta comunicación usa el *protocolo de conexión*.

- Las conexiones concernientes a un Puerto pueden ser creadas, destruidas o accedidas por una comunicación entre una entidad externa y ese Puerto. Esto se hace enviando *comandos de Puerto* usando el protocolo de conexión YARP.

- Los Puertos se comunican con el servidor de nombres usando el *protocolo del servidor de nombres YARP*. Esta comunicación ha de crear, borrar y acceder a registros.

- Las entidades externas también pueden usar el *protocolo de servidor de nombres YARP* para acceder al servidor de nombres.

- La utilidad YARP estándar puede ser utilizada para crear un servidor de nombres y también para actuar como una entidad externa para acceder y modificar la red YARP.

### 3.2.3. Gestión de los puertos

Un Puerto es un objeto que puede leer y escribir valores en objetos del mismo rango difundidos por una red de ordenadores. Pueden ser creados en un programa y añadir y eliminar conexiones tanto desde este programa como desde otro o desde la línea de comandos. Los Puertos se especializan en flujos de información, como las imágenes provenientes de una cámara o los comandos a un motor.

Además es posible cambiar los protocolos de la red sin cambiar ni una sola línea de código. Existen dos tipos de puertos:

- Puerto simple o *Port*: constituye un mini-servidor para comunicación en una red. Mantiene una colección dinámica de conexiones entrantes y salientes. Los datos provenientes de cualquier conexión entrante pueden ser recibidos llamando a la función Port::read. Las llamadas a la función Port::write hace que los datos sean enviados a todas las conexiones salientes.
- Puerto con memoria o *BufferedPort*: constituye un mini-servidor para realizar comunicación en red en background. De esta forma se puede enviar y recibir mensajes sin detener el procesamiento de órdenes.

A pesar de lo conveniente de esta funcionalidad, requiere el entendimiento del ciclo de vida de los objetos escritos y leídos de la red.

### 3.2.4. Empaquetamiento de la información

YARP ofrece un Contenido estándar llamado Bottle que no es más que una simple colección de objetos que pueden ser descritos y transmitidos de una forma portable. Los objetos se almacenan en una lista a la que se puede acceder y añadir elementos.

Por otra parte se pueden crear estructuras propias para almacenar información y usarlos como Contenido de manera sencilla. El objetivo en cualquier caso es encontrar formatos más eficientes de transmisión. Para la implementación de nuestra arquitectura se han implementado estructuras de datos específicos para cada uno de los módulos que envían información.

La información acerca de la herramienta YARP aquí presentada se puede extender acudiendo al sitio web de la librería: `http://eris.liralab.it/yarp`

## 3.3. Qt

Qt [11] es una librería multiplataforma, desarrollada inicialmente por la empresa Trolltech, orientada principalmente al diseño de interfaces de usuario, aunque

también admite desarrollo software sin interfaces gráficas.

**Figura 3.3: Logo Qt.**

Qt es soportado en las plataformas recogidas en la tabla 3.1.

| PLATAFORMA | VERSIÓN |
|---|---|
| Microsoft Windows | 98, NT, 4.0, ME, 2000, XP, ... |
| Unix/X11 | Linux, Sun Solaris, HP-UX, HP Tru64 UNIX, ... |
| Mac OS X | Mac OS X 10.3+ |
| Embedded Linux | Plataformas con soporte framebuffer |

**Tabla 3.1: Plataformas con soporte Qt.**

Asimismo, se ofrecen diferentes ediciones de Qt:

- **Edición Comercial:** Utilizada para desarrollo de software comercial. Se permite distribución tradicional de software comercial, incluyendo actualizaciones gratuitas y soporte técnico.
- **Edición Open Source:** Utilizada para desarrollo de Software Libre y Gratuito. Se provee libre de cargo bajo los términos de las licencias Q Public License y la GNU General Public License.

Trolltech también provee de un amplio rango de componentes para la industria y plataformas específicas que hacen un complemento perfecto de Qt en el ámbito industrial. Algunas de estas soluciones se encuentran disponibles para ciertos clientes mientras que otras están disponibles para todos los usuarios de Qt.

Algunos ejemplos de programas de amplia difusión basados en Qt son:

- Adobe Photoshop Album, aplicación para organizar imagenes.
- Doxygen, API generadora de documentación.
- Google Earth, simulador de mapas en 3D.
- KDE, popular entorno de escritorio para sistemas operativos tipo-Unix.
- Texmaker y LyX, GUIs para LaTeX.
- Mathematica, la versión de Linux usa Qt para el GUI.

- Qt Creator, el entorno de desarrollo integrado, software libre y multiplataforma de Nokia.
- Quantum GIS, sistema de Información Geográfica.
- Skype, aplicación de VOIP.

Algunas de las principales características de la librería son:

- Basado en una librería de clases de C++ intuitiva.
- Portabilidad entre sistemas operativos empotrados y de escritorio.
- Herramientas integradas de desarollo multiplataforma (IDE).
- Alto rendimiento en ejecución y poco espacio en disco y memoria en sistemas empotrados.

Qt 4 se compone de diferentes módulos, donde cada uno de los cuales reside en una librería independiente. Según las funcionalidades que se quieran implementar en la aplicación se irán incluyendo dichos módulos o minilibrerías, los cuales se recogen en la tabla 3.2.

| **MÓDULO** | **DESCRIPCIÓN** |
|---|---|
| QtCore | Clases de no GUI usadas por otros módulos |
| QtGui | Componentes de las interfaces gráficas de usuario |
| QtNetwork | Clases para programación de aplicaciones en red |
| QtOpenGL | Clases con soporte para OpenGL |
| QtSql | Clases para integración con SQL |
| QtScript | Clases para evaluación de Sripts Qt |
| QtSvg | Clases par amostrar el contenido de ficheros SVG |
| QtXml | Clases para el manejo de XML |
| QtDesigner | Clases para QtDesigner |
| QtUiTools | Clases para el manejo de formularios de QtDesigners en las aplicaciones |
| QtAssistant | Soporte para ayuda en Línea |
| Qt3Support | Clases con soporte de compatibilidad para Qt 3 |
| QtTest | Clases con herramientas para pruebas |

**Tabla 3.2: Módulos Qt.**

Los módulos de extensión disponibles en la versión comercial de Qt para sistemas Windows se recogen en la tabla 3.3.

Mientras que para sistemas Linux disponibles en todas las ediciones de Qt se incluye el módulo QtDBus, que incluye clases para el sistema de comunicaciones

| MÓDULO | DESCRIPCIÓN |
|---|---|
| QaxContainer | Extensiones para el acceso controles ActiveX |
| QaxServer | Extensiones para el desarrollo de servidores ActiveX |

**Tabla 3.3: Módulos Extra Qt en Sistemas Windows.**

Inter-Procesos D-Bus.

Los módulos principales en cualquier aplicación gráfica Qt son QtCore y QtGui.

El módulo QtCore forma parte de todas las ediciones de las librerías Qt, dependiendo el resto de módulos de la librería de este módulo. Para incluir la definición de las clases del método debemos incluir en nuestro código la sentencia:

```
1 #include <QtCore>
```

En primer lugar incluye la definición del namespace <Qt>, donde se incluyen numerosos identificadores usados en toda la librería. Incluye asimismo la definición de numerosas clases y objetos que son básicas para usar en otros elementos o widgets del resto de la librería.

Algunos ejemplos importantes se recogen en la tabla 3.4.

| MÓDULO | DESCRIPCIÓN |
|---|---|
| QCoreApplication | Bucle de eventos para aplicaciones Qt en consola |
| QPointF | Define un punto en el plano con precisión float precision |
| QString | Cadena de caracteres Unicode |
| QTime | Funciones de Tiempo y Reloj |
| QTimer | Temporizadores repetitivos y disparadores |
| QTimerEvent | Parámetros que describen un evento de tiempo |
| QIODevice | Clase interfaz base para todos los dispositivos de I/O en Qt |
| QFile | Interfaz para leer y escribir en ficheros |
| QEvent | Clase base de todas las clases de eventos |

**Tabla 3.4: Clases Módulo QtCore.**

El módulo QtGui extiende las funcionalidades del módulo QtCore incorporando todas las definiciones de clases utilizadas para el desarrollo de interfaces gráficas de usuario. Incorpora las definiciones de clases recogidas en la tabla 3.5.

Para incluir las definiciones de ambos módulos bastaría con la sentencia:

| MÓDULO | DESCRIPCIÓN |
|---|---|
| QAbstractButton | Clase base abstracta para elementos tipo botón |
| QAction | Acción abstracta de la interfaz de usuario insertada entre dos widgets |
| QActionEvent | Evento generado cuando una QAction es añadida, eliminada o cambiada |
| QApplication | Gestiona el control del flujo de la aplicación gráfica y sus parámetros principales |
| QBitmap | Mapas de píxeles monocromos (1 bit de profundidad) |
| QButtonGroup | Contenedor para organizar grupos de widget buttons |
| QCheckBox | Checkbox con etiqueta de texto |
| QComboBox | Botón combinado y lista desplegable |
| QDialog | Clase base para las ventanas de diálogo |
| QDialogButtonBox | Widget que presenta botones en un layout apropiado |
| QDrag | Soporte para acciones de arrastrar y soltar MIME-based |
| QGraphicsItem | Clase base para todos los elementos gráficos en una QGraphicsScene |
| QGraphicsScene | Superficie para la gestión de un gran número de elementos gráficos bidimensionales |
| QGraphicsView | Widget que muestra el contenido de una QGraphicsScene |
| QIcon | Iconos escalables y de diferentes modos y estados |
| QImage | Tratamiento de imágenes |
| QPushButton | Botón de Acción |
| QLCDNumber | Dmuestra un número con formato de display LCD |

**Tabla 3.5: Clases Módulo QtGui.**

```
1 #include <QtGui>
```

El módulo QtGui es parte de la Edición Ligera de Escritorio de Qt, la Edición de Escritorio Qt y la Edición Open Source de Qt.

Estos dos módulos recogen la mayoría de elementos que se utilizarán en cualquier aplicación basada en Qt. Si queremos añadir funcionalidades de red, de gráficos 3D, etc., debemos recurrir a otros módulos de Qt o a librerías externas, pero *¿cómo podemos comunicar entre sí todos estos elementos?*

Mediante los llamados Signals y Slots. El mecanismo de comunicación de objetos mediante señales y slots es un elemento central de Qt y probablemente la parte que más se diferencia de las características proporcionadas por otros frameworks de desarrollo.

En la programación de interfaces de usuario gráficas, cuando cambiamos un widget, a menudo queremos notificárselo a otro widget. De manera más general, queremos que los objetos de cualquier tipo sean capaces de comunicarse entre sí.

Por ejemplo, si un usuario hace clic en un botón de cierre, es probable que

quieren llamar a la función para cerrar la ventana close().

En sistemas más antiguos, este tipo de comunicación se lograba con las devoluciones de llamada (callbacks).

Un callback es un puntero a una función, así que si quieres una función de procesamiento para informarle sobre algún evento, debías pasar un puntero a otra función (la devolución de llamada) a la función de procesamiento.

La función de procesamiento a continuación, llama a la devolución de llamada cuando corresponda. Los sistemas basados en callbacks tienen dos defectos fundamentales: en primer lugar, no son de tipo seguro.

Nunca podemos estar seguros de que la función de procesamiento llamará a la devolución de llamada con los argumentos correctos. En segundo lugar, la devolución de llamada está fuertemente ligada a la función de procesamiento ya que la función de procesamiento debe saber a que callback llamar.

En Qt, tenemos una alternativa a la técnica de devolución de llamada: utilizamos señales y slots. Una señal se emite cuando se produce un evento en un widgets en particular. Los widgets de Qt tienen muchas señales predefinidas, pero siempre podemos hacer una subclase del widget para agregar nuestras señales propias para ellos.

Un slot es una función que se llama en respuesta a una señal particular. Igualmente los widgets de Qt tienen muchos slots predefinidos, pero es una práctica común crear subclases de widgets y añadir sus propias slots para que pueda manejar las señales que nos interesen.

El mecanismo de señales y slots sí es de tipo seguro: la firma de una señal debe coincidir con la firma del slot de recepción.

De hecho, un slot puede tener una firma más corta que la señal que recibe, ya que puede pasar por alto parámetros extra.

Dado que las firmas son compatibles, el compilador puede ayudarnos a detectar los desajustes de tipo.

Señales y slots están débilmente acoplados: una clase que emite una señal ni sabe ni le importa qué slot va a recibir la señal.

Las señales y slots de Qt aseguran que si se conecta una señal a un slot, el slot se llama con los parámetros de la señal en el momento adecuado. Señales y slots pueden tener cualquier número de argumentos y de cualquier tipo. Es por tanto un

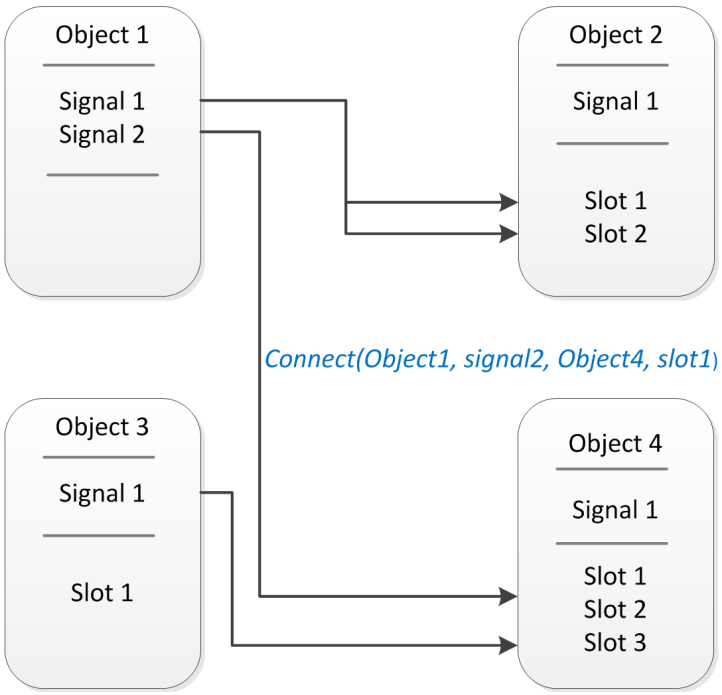


**Figura 3.4: Diagrama de conexión de señales y slots diversos objetos Qt.**

mecanismo totalmente seguro.

Todas las clases que heredan de QObject o una de sus subclases (por ejemplo, QWidget) pueden contener señales y slots. Las señales son emitidas por objetos cuando cambian su estado de forma que puede ser interesante para los otros objetos. Todo esto es lo que el objeto hace para comunicarse. No sabe ni le importa si algo está recibiendo las señales que emite. Esta es la encapsulación de información veraz, y asegura que el objeto puede ser utilizado como un componente de software.

Los slots se pueden utilizar para la recepción de señales, pero también son miembros de las funciones normales. Así como un objeto no sabe si algo recibe sus señales, un slot no sabe si tiene señales conectadas a ella. Esto asegura que con Qt podemos crear componentes verdaderamente independientes.

Se puede conectar tantas señales como se requiera a un único slot, así como también es posible conectar una misma señal a los slots que sea necesario. Incluso es posible conectar una señal directamente a otra señal. Esta emitirá la segunda señal de inmediato cada vez que la primero se emite.

Juntos, señales y slots constituyen un potente mecanismo de programación de componentes.

Qt no es sólamente las librerías con sus clases, elementos y mecanismos de comunicación. Actualmente para facilitar la escritura de aplicaciones Qt disponemos de completos entornos de desarrollo, que separados en distintos módulos facilitan enormemente el proceso de desarrollo de una aplicación.

En concreto, para el desarrollo de la interfaz de Romeo se ha aprovechado la integración con Qt que implementa el conocido entorno de desarrollo KDevelop así como QtDesigner para la creación de los formularios y ventanas de la aplicación.

QtDesigner en un potente generador de layouts y formularios para el diseño de interfaces de gráficas de usuario. Al igual que el resto de Qt es totalmente multiplataforma. QtDesigner permite diseñar y contruir rápidamente widgets y diálogos con formularios que formarán parte de la aplicación final. Los formularios y ventanas creadas con QtDesigner son totalmente funcionales y pueden generarse una preview de tal modo que podemos comprobar y asegurar el correcto funcionamiento y apariencia que teníamos en mente.

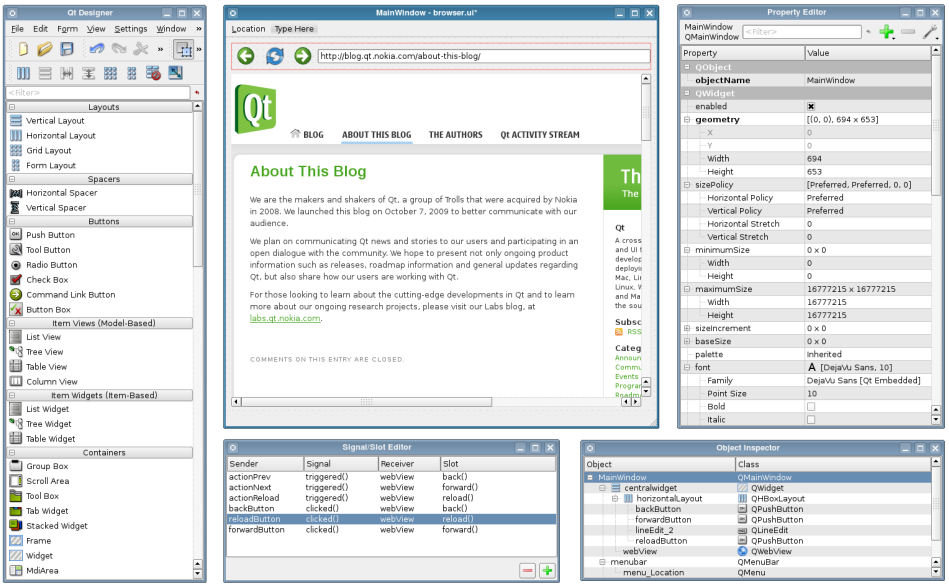


**Figura 3.5: Qt Designer.**

Características y beneficios:

- Diseño de interfaces rápido con la funcionalidad Arrastrar y Soltar.
- Genera widgets a medida o utiliza los estándares de la librería.

- Previews instantáneas.
- Generación de código C++ o Java de los prototipos de la interfaz.
- Integración de QtDesigner con múltiples IDEs (Visual Studio, Eclipse, Kdevelop, ...).
- Construcción de interfaces de usuario totalmente funcionales mediante el uso de las señales y slots propias de Qt.

Comentar en último lugar que la versión más actual del desarrollo ha sido lanzada en su versión Qt 5.0 en el momento de escritura de este texto, mientras que la aplicación Romeo HMI fue desarrollada con la versión 4.3 de la librería.

## 3.4. OPENCV

OpenCV (Open Source Computer Vision) [6] es una librería de código abierto para la programación de sistemas de visión por ordenador en tiempo real.


**Figura 3.6: OpenCV Logo.**

OpenCV está escrito en C y C++ y es multiplataforma, ejecutándose bajo Linux, Windows y Mac OS X.

Actualmente existen desarrollos en activo trabajando en interfaces para Python, Ruby, Matlab y otros lenguajes de programación.

OpenCV fue diseñado para obtener eficiencia computacional en ejecución y con un fuerte enfoque para las aplicaciones en tiempo real. Está escrito en lenguaje optimizado en C y además puede beneficiarse del uso de procesadores multinúcleo.

Uno de los objetivos del proyecto OpenCV es proveer de una infraestructura de visión por computador fácil de usar, que facilite a los desarrolladores construir sofisticadas aplicaciones de visión de manera rápida y eficiente. La librería OpenCV contiene más de 500 funciones que incluyen muchas áreas de la visión por computador, tales como inspección en la fabricación de productos, imágenes médicas, seguridad, interfaces de usuario, calibración de cámaras, visión estéreo, así como aplicaciones específicas para robótica.

OpenCV está estructurado en cinco componentes principales, cuatro de las cuales se muestran en la figura 3.7.

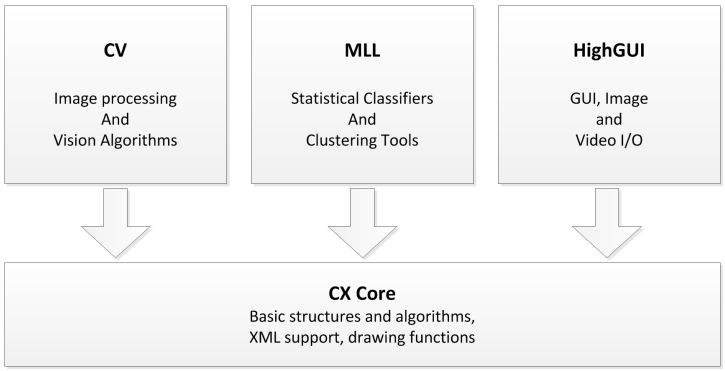


**Figura 3.7: Estructura librería OpenCV.**

El componente CV contiene el procesamiento base de imágenes y algoritmos de nivel superior de visión por computador. MLL (Machine Learning Library) es la librería de aprendizaje automático, que incluye a muchos clasificadores estadísticos y herramientas de clustering. HighGUI contiene rutinas de salida y funciones para el almacenamiento y carga de vídeo e imágenes, y CXCore contiene las estructuras de datos básicos y el contenido.

La figura 3.7 no incluye el último módulo CvAux, que contiene áreas abandonadas del proyecto (integrado HMM y reconocimiento facial) y algoritmos experimentales (segmentación de fondo/primer plano).

CvAux no está especialmente bien documentado aunque cubre las siguientes áreas:

- Eigen objects, una técnica de reconocimiento computacionalmente eficiente, es decir, en esencia, un patrón de reconocimiento de formas.
- 1D y 2D modelos ocultos de Markov, una técnica de reconocimiento de estadística resuelto por programación dinámica.
- Los HMM incrustado (las observaciones de una HMM padres mismos son HMM).
- Reconocimiento gestual apoyado en visión estéreo.
- Extensiones a la triangulación de Delaunay, secuencias, etc.
- Visión estéreo.
- Descriptores de textura.

- Ojos y seguimiento de la boca.
- Seguimiento 3D.
- Búsqueda de esqueletos (vías centrales) de los objetos en una escena.
- Distorsión intermedios puntos de vista entre dos puntos de vista de la cámara.
- Antecedentes en primer plano de segmentación.
- Vigilancia por video.
- Clases para calibración de cámaras de C++ (las funciones de C y el motor están en CV).

**Figura 3.8: Proyecto URUS. Seguimiento visual de personas para experimento de guiado.**