Integración de Vídeo Streaming en la Teleoperación del AmigoBot

Proyecto de Fin de Carrera

Resumen

Septiembre 2012

Autor

José Antonio Crespo Toral

Tutores

Jesús Fernando Rodríguez Aragón

Iván Álvarez Navia

Belén Curto Diego

Tabla de contenidos

Lista de figuras

Lista de tablas

1. Introducción

En colaboración con los tutores y tras algunos cambios, se elaboró la propuesta para el proyecto "*Integración de Vídeo Streaming en la Teleoperación del AmigoBot*. El objetivo en un principio, como el propio título nos indica, fue poder recibir vídeo stream de una webcam conectada al pc Debian y realizar un pequeño tratamiento de las imágenes. Tras entregar la propuesta, en colaboración con los tutores, el propósito principal del proyecto se fue desviando haciendo de la integración de vídeo streaming en la teleoperación del robot, una herramienta más para conseguir un objetivo más completo y elaborado. El nuevo objetivo del Proyecto de Fin de Carrera se convertía de esta manera en la implementación de un software que permitiera la navegación de un robot, el Amigobot de *Mobile Robots*, a través de códigos QR mediante el uso de dispositivos iOS, integrando así el objetivo anterior. Se intentó cambiar el título del proyecto a "QrNav: Navegación de un robot a través de códigos QR usando un dispositivo iOS" pero debido a problemas administrativos no se pudo realizar el cambio.

Este software sirve como interfaz de usuario, permitiendo controlar al Amigobot (robot) mediante un *joystick* situado en la pantalla táctil del dispositivo iOS y obtener las imágenes de una webcam conectada al pc (con distribución Linux) para dirigirlo hacia algún código QR y poder leer su contenido. En caso de que dicho QR contenga una dirección, el robot de manera autónoma, se dirigirá hasta la posición indicada gracias a la odometría del mismo. Además, para salvar o remediar en cierta medida los grandes errores de la odometría, se realiza un tratamiento de las imágenes recibidas para situar al robot siempre en una posición centrada que le permita leer el código QR. El software también dispone de un sistema de opciones persistente que permite mejorar el manejo del Amigobot y algún aspecto de la interfaz al gusto del usuario.

2. Objetivos del proyecto

Durante el proceso de desarrollo se pretenden conseguir una serie de objetivos que se dividen en tres aplicaciones:

- La aplicación QrNav (QrNavCliente), será la encargada de decidir qué proceso debe ejecutarse en cada momento en función de los valores que obtenga y enviar los datos al servidor. También actuará como cliente para recibir las imágenes que el servidor (*MJPG-Streamer*) capta de la webcam en tiempo real. Se desarrollará para dispositivos iOS, centrándose en crear una interfaz útil, sencilla e intuitiva y gráficamente trabajada que permite:
  - Controlar la teleoperación del robot
  - Leer un código QR de las imágenes recibidas del servidor mediante biblioteca *Zxing*.
  - Poner en marcha la recepción de vídeo y pausarla.
  - La desconexión con el robot en caso de emergencia.
  - Mostrar en todo momento el estado del robot.

  Contendrá un sistema de opciones persistentes con el que podremos mejorar el manejo del robot con diferentes características y editar algún aspecto gráfico de la interfaz.

  Será capaz de realizar un tratamiento de las imágenes obtenidas buscando un cuadrado rojo alrededor del QR para colocar al robot y, posteriormente, leer un código QR mediante la biblioteca *OpenCV*.

- Un Servidor (ServidorQrNav) que se utilizará para recibir los datos enviados por la aplicación y para controlar al robot en función de esos datos. Entre sus funciones encontramos:
  - Usará la biblioteca *ARIA* para la comunicación con el robot.
  - Ordenará y actuará en función de los parámetros recibidos.
  - Conducirá al robot hasta la posición recibida.
  - Colocará al robot para realizar la lectura en función de los parámetros obtenidos al realizar el tratamiento.

- Un Servidor que transmitirá un flujo continuo de imágenes a través de *HTTP*. Para este servidor se utilizará el programa *MJPG-Streamer*.

Los objetivos aquí presentados son una breve descripción, para conseguir un mayor detalle de estos objetivos se debe consultar el *Anexo 2 – Especificación de requisitos* adjunto a la documentación del proyecto.

3. Conceptos teóricos

Este apartado está dedicado a describir una serie de conceptos teóricos necesarios para la comprensión del proyecto y del resto de la memoria. A continuación se presentan algunos de los más importantes y se omitirán otros conceptos teóricos. Para obtener más información, consultar el apartado tres de la memoria del proyecto.

3.1 Código QR

Un código QR (*quick response code*, «código de respuesta rápida») es un sistema para almacenar información en una matriz de puntos o un código de barras bidimensional

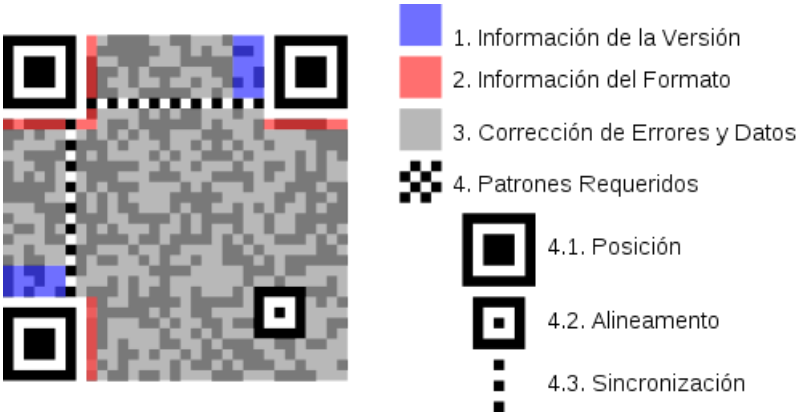

Figura 1 - Estructura de un código Qr

Como observamos en la Figura 1, el código QR está compuesto por tres etiquetas de posición situadas en la parte superior izquierda, superior derecha e inferior izquierda. Éstos se utilizan para localizar el código QR en una imagen y ayudar a servir como marcadores. También son útiles para determinar el grado de inclinación y rotación del código en la imagen.

La versión y formato de la información se almacena en al menos dos lugares alrededor de las etiquetas de posición. Esto permite una mejor recuperación de datos en el caso de que una esquina del código esté dañada u obstruida. Hay también una etiqueta de alineamiento, que puede ser utilizada para los cálculos de alineación adicionales.

El patrón de cuadros continuado, tanto vertical como horizontalmente entre dos etiquetas de posiciones vecinas, permite una rutina de calibración para realizar cálculos y determinar el tamaño de las regiones individuales en blanco y negro. Una de las principales fortalezas del Código QR son sus múltiples niveles de corrección de errores. Se utiliza una técnica de corrección de errores de Reed-Solomon y, dependiendo del nivel de corrupción de datos, puede soportar hasta un 30% de la restauración de éstos.

3.2 Odometría

La odometría son las técnicas de posicionamiento que emplean la información procedente de la rotación de las ruedas para obtener una aproximación de la posición real en la que se encuentra un sistema móvil, en un determinado instante y respecto a un sistema de referencia inicial.

En general, son suficientes tres parámetros (*X, Y, θ*) para conocer la posición de un sistema móvil:

- La posición respecto al eje *X* que para nuestro robot es el eje vertical.
- La posición respecto al eje *Y* que en el caso del robot es el eje horizontal.
- La orientación del robot *Th* o *θ* que indica el ángulo hacia el que se encuentra orientado.

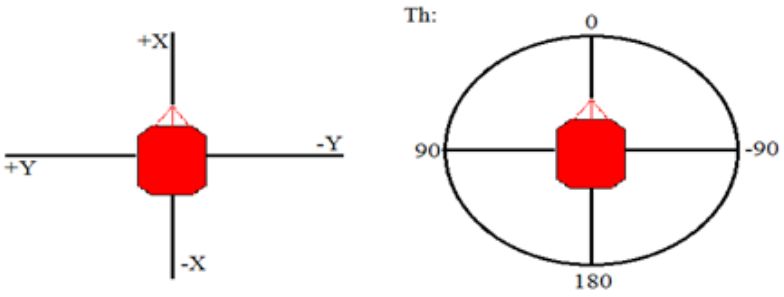

Figura 2 - Coordenadas y ejes del Amigobot

Cuando el robot se enciende y conecta con un dispositivo, toma la posición actual en la que se encuentra como posición inicial de referencia, es decir, en ese momento estará en la posición (0, 0, 0).

Sin embargo la idea fundamental de la odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo cual conlleva una inevitable acumulación de errores. En concreto, la acumulación de errores de orientación causa grandes errores en la estimación de la posición, que van aumentando proporcionalmente con la distancia recorrida por el robot.

La odometría se basa en ecuaciones simples que se pueden implementar fácilmente y que utilizan datos de *encoders* situados en las ruedas del robot. Sin embargo, la odometría también está basada en la suposición de que las revoluciones de las ruedas pueden ser traducidas en un desplazamiento lineal relativo al suelo. Esta suposición no tiene una validez absoluta. Un ejemplo extremo se produce cuando una de las ruedas patina sobre una mancha de aceite y la otra no, entonces el *encoder* asociado registrará revoluciones en la rueda, aunque éstas no correspondan a un desplazamiento lineal de la rueda. Todos estos errores se pueden agrupar en dos categorías: errores sistemáticos que son muy graves porque son constantes (como el mal alineamiento de las ruedas) y errores no sistemáticos que pueden aparecer en cualquier momento (como los suelos desnivelados o el patinaje de las ruedas).

4. Técnicas y herramientas

En este apartado se realizará un pequeño resumen de los lenguajes de programación y herramientas utilizados en el desarrollo del proyecto.

En el proyecto se ha seguido el Proceso Unificado para el desarrollo del software, empleando Gantt Proyect para la gestión del proyecto, durante la fase de elicitación de requisitos la metodología propuesta por Durán Toro y Bernárdez Jiménez junto con el programa REM, el lenguaje UML para la notación en la fase de análisis y diseño junto con la herramienta Visual Paradigm, así como el entorno DIA para la realización de los diagramas de flujo de los algoritmos realizados.

Como lenguaje de progrmación se ha utilizado Objective-C para el desarrollo de la aplicación iOS usando como IDE (*integrated development environment*, «Entorno de desarrollo integrado») XCode 4 y C++ para el desarrollo del ServidorQrNav.

También se han utilizado las siguientes herramientas:

- La biblioteca **OpenCV**[1] (*Open Source Computer Vision*), para realizar el tratamiento de las imágenes recibidas en el iPhone. La biblioteca está escrita en C++, aunque en el proyecto está adaptada a Objective-C para poder usarla en la programación de dispositivos iOS.

- La biblioteca ***ZXing***[2], nos servirá para realizar la decodificación de los códigos QR en el iPhone. Está escrita para varios lenguajes, entre ellos Objective-C.

- La biblioteca **ARIA**[3] *(Advanced Robot Interface for Applications*) de Mobile Robots, se utilizará para el desarrollo del servidor que transmitirá las órdenes al Amigobot. Esta biblioteca está escrita en C++.

- El programa ***MJPG-streamer***[4], que utilizaremos de servidor para que obtenga las imágenes de la webcam y las transmita a través de HTTP (*Hypertext Transfer Protocol*, «protocolo de transferencia de hipertexto») al cliente.

- Para la conexión entre el iPhone y el Pc Debian se ha usado **Zeroconf**[5] (*Zero Configuration Networking)* que es un conjunto de técnicas que permiten crear de forma automática una red *IP* sin configuración o servidores especiales. Pero *Zeroconf* es una iniciativa que nos propone una manera de hacer las cosas, no una implementación específica. Se han usado

[1] http://opencv.org/
[2] http://code.google.com/p/zxing/
[3] http://robots.mobilerobots.com/wiki/ARIA
[4] http://sourceforge.net/projects/mjpg-streamer/
[5] http://www.zeroconf.org/

las implementaciones de **Bonjour**[6] para Mac OS X y **Avahi**[7] para distribuciones Linux.

- **Microsoft Word** que es un software dedicado al procesamiento de textos y ha sido empleado para la realización de la documentación.
- **Adobe Photoshop**[8] que es una aplicación informática en forma de taller de pintura y fotografía que está destinado para la edición, retoque fotográfico y pintura a base de imágenes de mapa de bits. Este programa ha sido usado para la realización de la parte gráfica de la aplicación.
- **MobileSim** es el simulador ofrecido por *ARIA* para poder realizar pruebas con un robot virtual que imita el comportamiento real del Amigobot. Este simulador ha sido utilizado para realizar pruebas antes de ejecutar el proyecto con el robot real.

También fueron investigadas las bibliotecas *Video for Linux 2*[9] (V4L2) que es una interfaz de programación de aplicaciones (*API*) orientada a la captura de imágenes desde múltiples dispositivos y *Zbar*[10] que es un software de código abierto para la lectura de códigos de barras a partir de diversas fuentes.

En cuanto al hardware usado:

- **iPhone 4**, como dispositivo iOS de Apple para la ejecución de la aplicación QrNav.
- **Amigobot** de Mobile Robots, es un robot de pequeño coste para proyectos de educación y colaboración.
- **Webcam Logitech**[11] **Quickcam Pro 9000**, cámara capaz de producir un vídeo fluido y natural e instantáneas de hasta 8 megapíxeles.

[6] http://www.apple.com/es/support/bonjour/
[7] http://avahi.org/
[8] http://www.adobe.com/es/products/photoshop.html
[9] http://linuxtv.org/downloads/v4l-dvb-apis/
[10] http://zbar.sourceforge.net/
[11] http://www.logitech.com/es-es

5. Aspectos relevantes del desarrollo

En esta sección se tratará de introducir aquellos aspectos más relevantes por su importancia dentro del proyecto y su dificultad en la concepción o en el desarrollo. Muchos de estos aspectos relevantes han sido omitidos en este resumen, para obtener información sobre ellos y profundizar en los que aquí se presentan podemos consultar este mismo apartado en la memoria.

5.1 Arquitectura Cliente-Servidor

La arquitectura de la aplicación encaja perfectamente en el paradigma de Cliente-Servidor que separa el proyecto en tres partes (ver Figura 3):

- La aplicación QrNav, que puede ejecutarse en cualquier dispositivo iOS. Realiza la función de cliente interactuando con el usuario a través de la pantalla, procesando los datos y comunicándoselos al ServidorQrNav y recibiendo las imágenes de la webcam conectada al Pc Debian.
- El Servidor que controla al Amigobot (ServidorQrNav), que recibirá los datos y, en función de los mismos, manejará al robot usando el API de ARIA.
- Programa *MJPG-Streamer*, que nos realiza la función de servidor para transmitir el flujo imágenes obtenidas de la webcam a través de HTTP.

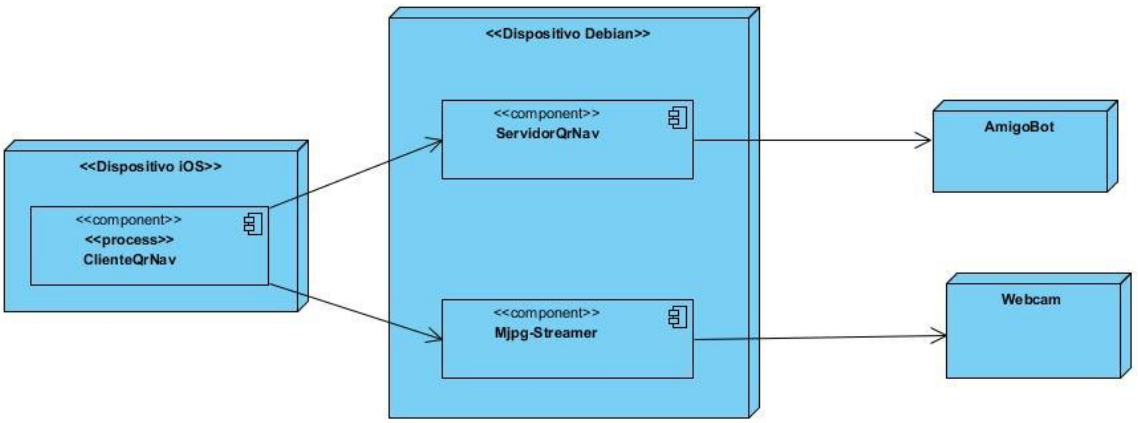

Figura 3 - Diagrama de Despliegue

5.2 Protocolo de comunicación

Nuestro ClienteQrNav está continuamente comunicándose con nuestro ServidorQrNav. Esta comunicación sigue un protocolo.

El clienteQrNav envía 10 parámetros *float* a dicho servidor:

Parámetro	Descripción
Vel	Indica la velocidad del Amigobot
Wel	Indica la velocidad angular del Amigobot
QrFlag	Bandera que indica si se ha leído un QR con dirección de navegación
X	Coordenada X de la posición final de navegación
Y	Coordenada Y de la posición final de navegación
Th	Coordenada Th de la posición final de navegación
Emergency	Bandera que indica si se ha pulsado el botón de emergencia
ColocaFlag	Bandera que indica que si ha realizado el tratamiento de la imagen para la colocación
Pto	Punto medio del cuadrado Rojo
Área	Área del cuadrado Rojo

Tabla 1 - Parámetros protocolo de comunicación

El servidorQrNav sólo le envía un parámetro entero, `acabado`, al clienteQrNav y dependiendo de su valor:

- 0, el Amigobot está realizando el algoritmo de NavegaciónQr o ColocaciónQr.
- 1, indica al cliente que debe realizar el tratamiento de la imagen para la colocación.
- 2, es el valor normal que significa que el robot está en el modo Teleoperación.
- 3, indica al cliente que está delante de un código QR y debe intentar leerlo.
- 4, indica al cliente que el algoritmo de colocación ha fallado.

Estos son los parámetros que, tanto clienteQrNav como servidorQrNav, se transmiten continuamente a través de los *socket* conectados.

Antes de ver el algoritmo debemos saber la función de ciertas variables:

Variable	Descripción
Navegando	Indica si el hilo de navegación está ejecutando el algoritmo de NavegaciónQr. Su valor inicial es 0
Colocando	Indica si el hilo de colocación está ejecutando el algoritmo de Colocación. Su valor inicial es 0
Destino	Variable que indica si el Amigobot ha terminado de realizar el algoritmo de NavegaciónQr llegando a la posición indicada. 0 – No ha llegado, 1- Ha llegado. Valor inicial es 0
Fin	Variable que indica si el Amigobot ha terminado de realizar el algoritmo de Colocación llegando a su posición. 0 – No ha llegado, 1- Ha llegado. Valor inicial es 0
Colocaciones	Variable que indica el resultado de la ColocaciónQr del robot

Tabla 2 - Variables protocolo de comunicación

A continuación se presenta el diagrama de flujo del algoritmo de comunicación realizado por el servidor:

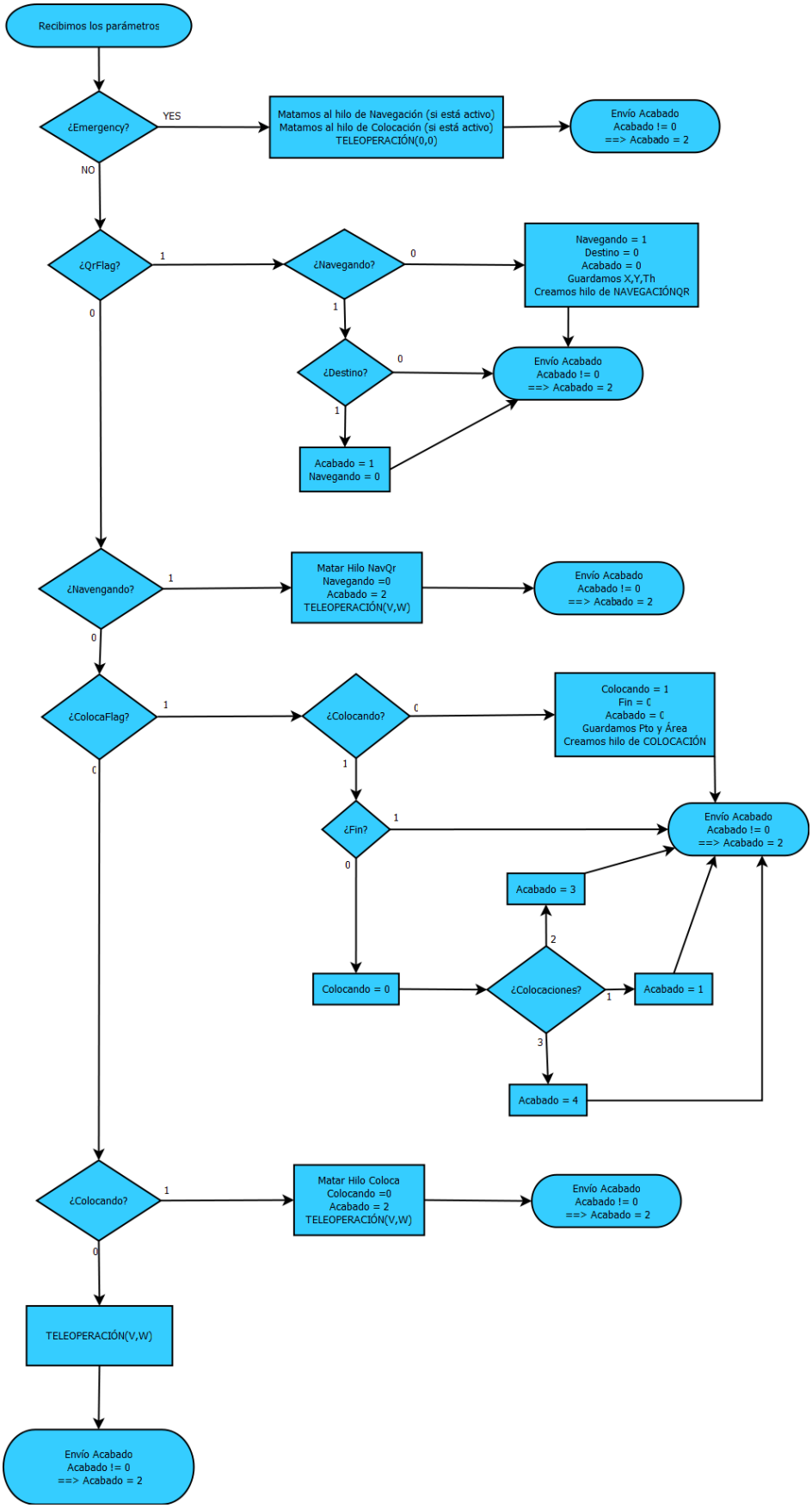

Figura 4 - Diagrama de flujo del protocolo de comunicación ServidorQrNav

5.3 Transmisión de Vídeo Streaming en tiempo real

Esta transmisión se realiza mediante el programa *MJPG-Streamer*, ya que permite obtener, con el *plugin* de entrada *input_uvc.so*, las imágenes de la webcam conectada al pc (mediante *VideoForLinux2*) y con el *plugin* de salida *output_http.so*, que hace de servidor web, transmitir el flujo de imágenes a través de *HTTP* a cualquier cliente que lo solicite. Además este programa permite configurar el servidor con diversos parámetros. En el proyecto este programa establece los *frames* del vídeo streaming a un tamaño de 640 x 480 píxeles y se transmiten a una velocidad de 15fps.

Como cliente se ha utilizado un protocolo de código abierto desarrollado por Hao Hu[12] para XCode que actúa como cliente *HTTP* y recibe un flujo de imágenes[13].

5.4 Cliente: QrNav

La aplicación cliente se encarga de proporcionar al usuario una interfaz simple e intuitiva y algunas de sus funciones son interactuar con el usuario, recibir las imágenes, realizar un tratamiento de las imágenes, decodificar códigos QR, conectar y comunicarse con el ServidorQrNav, etc.

El ClienteQrNav es el núcleo de nuestro proyecto, en otras palabras, es el que decide qué hacer y el que realiza todos los cálculos necesarios para, posteriormente, transmitírselo al servidor.

En el desarrollo de esta aplicación se han utilizado diversos patrones de diseño como el patrón *Singleton*, *MVC* y *Delegation.*

5.4.1 Gestión en MandoViewController

Esta clase es la encargada de administrar todo lo que ocurre en esta vista de Mandos. Entre sus funciones se encuentran mostrar las imágenes recibidas en pantalla, decodificar el código QR de la imagen cuando sea necesario y realizar el algoritmo de colocación para la obtención de los parámetros `area` y `Pto`.

Esta clase debe realizar las operaciones establecidas según el parámetro que recibe del servidorQrNav. Esta gestión se realiza en el método que se ejecuta cada vez que se recibe una imagen correctamente.

A continuación se explica el proceso que realiza esta función:

[12] http://www.haohu.de
[13] https://github.com/horsson/mjpeg-iphone

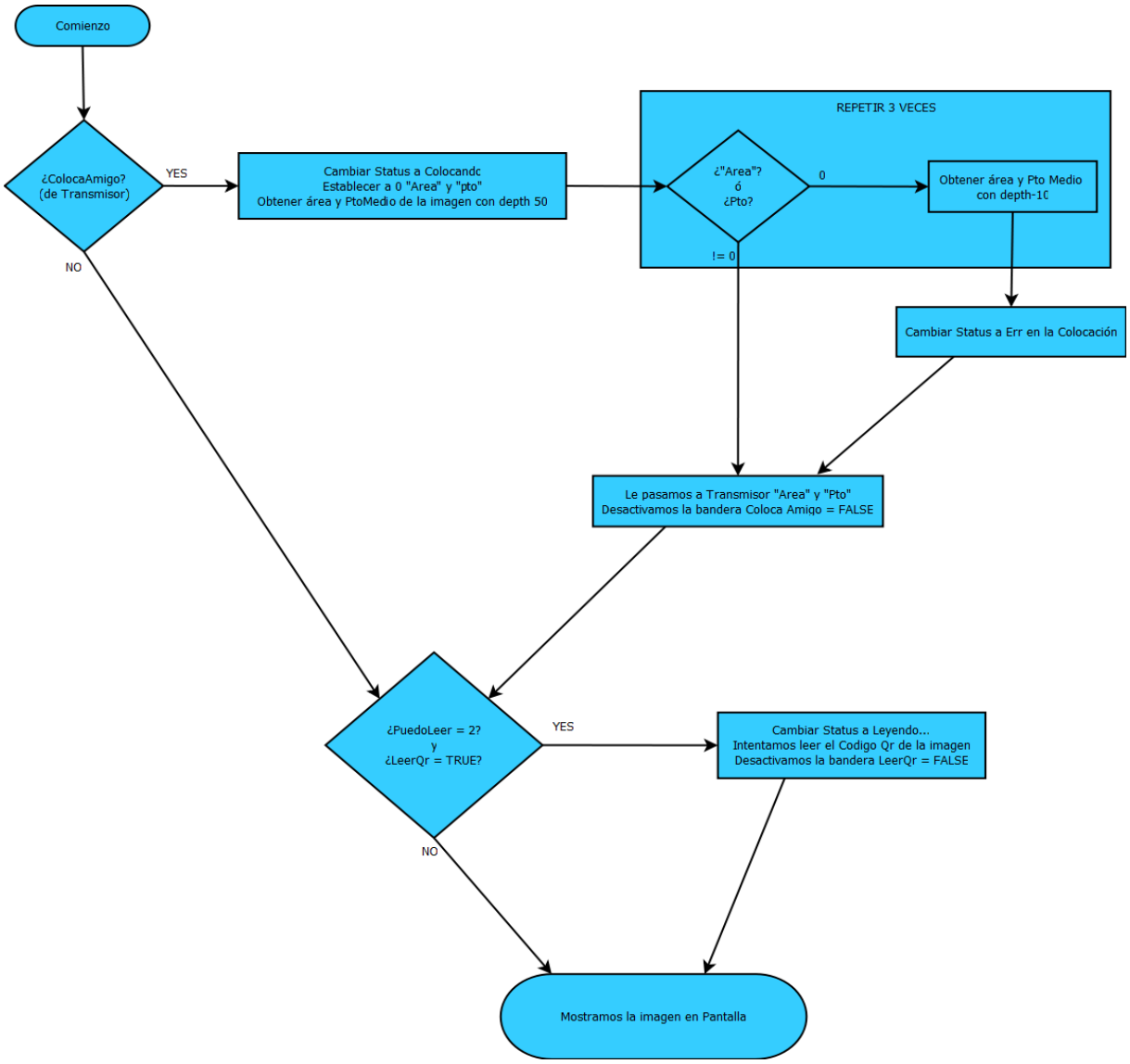

Figura 5 - Diagrama de flujo de control de operación en QrNav

5.4.1.1 Leer Código QR

En este apartado se detallará cómo se lleva a cabo la lectura de códigos QR.

Leer un código QR se realiza gracias a la biblioteca *Zxing*, que nos proporciona la clase `QrCodeReader` que decodifica códigos QR gracias a un protocolo `DecodeDelegate`, que llamará a una función si consigue decodificar algún código QR en la imagen o, por el contrario, llamará a otra función distinta si no consigue leer el código QR o no lo hubiera.

Cuando un código Qr se decodifica con éxito se intenta parsear la cadena leída del QR para saber si contiene una dirección, es decir, la cadena es del tipo "*Goto X:- Y:- Th:-*". En el caso de que se trate de una dirección obtendremos los valores de las coordenadas X, Y y Th y se las transmitiremos al ServidorQrNav para que comience la navegación.

5.4.1.2 ColocaciónQr

Para disminuir los fallos provocados por la odometría se introdujo un nuevo punto de referencia, que los códigos QR estuvieran enmarcados dentro de un cuadrado rojo intenso (R 255, G 0, B 0). Para que una vez el robot haya llegado a la posición de navegación, el ServidorQrNav pueda colocarlo con respecto al punto medio y el área de ese cuadrado rojo.

Para realizar el tratamiento de imágenes se utiliza la biblioteca *OpenCV*. El proceso consiste en transformar la *UIImage* donde se encuentra la imagen a *IplImagen* (variable que almacena las imágenes de *OpenCV*) y modificando las funciones (del ejemplo "*squares.c*" que vienen adjuntas a la biblioteca *OpenCV)* buscamos el cuadrado rojo dentro de la imagen y rellenamos su área. Una vez tenemos la imagen con nuestro cuadrado solo tenemos que buscar píxel a píxel cuál es el punto más a la izquierda del cuadrado (xmin) y cuál es el que está más a la derecha (xmax) y calcula su área contando el número de píxeles que tiene el cuadrado. Así es como conseguimos obtener el área y el punto medio `((xmin+ xmax)/2)` de la imagen que recibimos de la webcam.

En este proceso pueden aparecer errores por causa de la luz que dan lugar a problemas de saturación, como por ejemplo, que la luz se refleje en el código QR y en el cuadrado rojo y el dispositivo no lo detecte.

5.4.2 Movimiento y Gestión de los Eventos Touch

En la aplicación QrNav nos encontramos un *joystick* para poder teledirigir al Amigobot. Las interacciones del usuario con dicho *joystick* generan tres eventos:

- TouchesBegan, se genera cuando el usuario toca la pantalla.
- TouchesMoved, se genera cuando el usuario mueve el dedo por la pantalla.
- TouchesEnded, se genera al levantar el dedo de la pantalla.

Estos eventos envían a la clase *MandoViewController* las coordenadas en las que se encuentra el *joystick* en cada momento. Dichas pulsaciones sólo pueden ir, en el eje y, desde la coordenada 350 a 442 y, en el eje x, de 162 a 254. Ambas tienen un intervalo de 46 positivos y 46 negativos, por lo que podemos ver en la Tabla 3 la traducción de las coordenadas a velocidad:

EJE X			**EJE Y**		
350	396	442	162	208	254
-46	0	46	46	0	-46
Velocidad de Giro			**Velocidad**		
Máxima a la izquierda	Nula	Máxima a la derecha	Máxima hacia adelante	Nula	Máxima hacia atrás

Tabla 3 - Coordenadas a intervalos

5.5 ServidorQrNav

El servidorQrNav es el encargado de realizar parte del protocolo de comunicación por el que recibirá la información que le transmite el ClienteQrNav y la usará para el control del Amigobot.

5.5.1 Estructura

El servidor se compone de dos clases y de la función *main*:

- La función *main*, es la función con la que comienza el ServidorQrNav. Su tarea es recibir los parámetros con los que se ha ejecutado el programa, crear una instancia de la clase *Receptor* y llamar al método `empezarRecepcion` pasándole los parámetros recibidos en el *main*.
- La clase *Receptor* es la encargada de manejar los aspectos de la conexión y de comunicarse continuamente con el cliente.
- La clase *ControladorAmigobot* es la que se comunica con el robot.

5.5.2 Publicación del servicio. Avahi

Para conectar nuestro dispositivo iOS con el ordenador que controlará el Amigobot, se ha optado (como ya he explicado anteriormente) por usar *Zeroconf*, en concreto para el servidor, la implementación Avahi.

El funcionamiento de *Zeroconf* en cuanto a los servicios es el siguiente: un servidor publica un servicio, un cliente busca un tipo determinado de servicios y elige un servicio y obtiene su *IP* y su Puerto.

En Avahi hay varios métodos para publicar un servicio. Uno de ellos consiste en crear en la carpeta: */etc/avahi/services* un archivo del tipo *.service*, este archivo será un archivo tipo *NXML* que contendrá los datos del servicio. Una vez creado el archivo *.service*, desde nuestro dispositivo iOS accedemos a la lista de servicios, nos aparecerá y nos permitirá conectar con nuestro ServidorQrNav.

Avahi realizará el resto del trabajo por nosotros, ya que su *daemon* presente en el sistema se encargará de publicar el servicio y de responder a las peticiones que recaigan sobre él.

5.5.3 Navegación Qr

Se trata de una de las partes centrales del proyecto. Consiste en que cuando se recibe la bandera de navegación activada con las coordenadas de la dirección (`x`, y, `th`), el Amigobot se dirija hacia dicha dirección sin que el usuario tenga que realizar ninguna interacción con el dispositivo iOS.

Esta navegación se realiza en forma de L, en primer lugar recorre la distancia en el eje Y, posteriormente en el eje X y por último se coloca según la orientación utilizando la odometría de las ruedas para el control de las coordenadas.

5.5.3.1 Controladores de Avance y Giro

En este capítulo se explican los algoritmos de los controladores de Avance y Giro. Son controladores lineales que permiten al robot recorrer la distancia total que necesita desplazarse (tanto en los ejes X e Y como en el eje Th) en 3 fases:

- **Fase de aceleración**, que consiste en acelerar hasta la velocidad máxima desde una velocidad inicial de cero.
- **Fase de velocidad constante**, en esta fase nos desplazamos a una velocidad máxima constantemente.
- **Fase de deceleración**, vamos decelerando desde la velocidad máxima hasta llegar a cero.

Estas fases ocupan un tercio de la distancia total.

En la Figura 6 y Figura 7 se pueden observar los diagramas de flujo de ambos controladores.

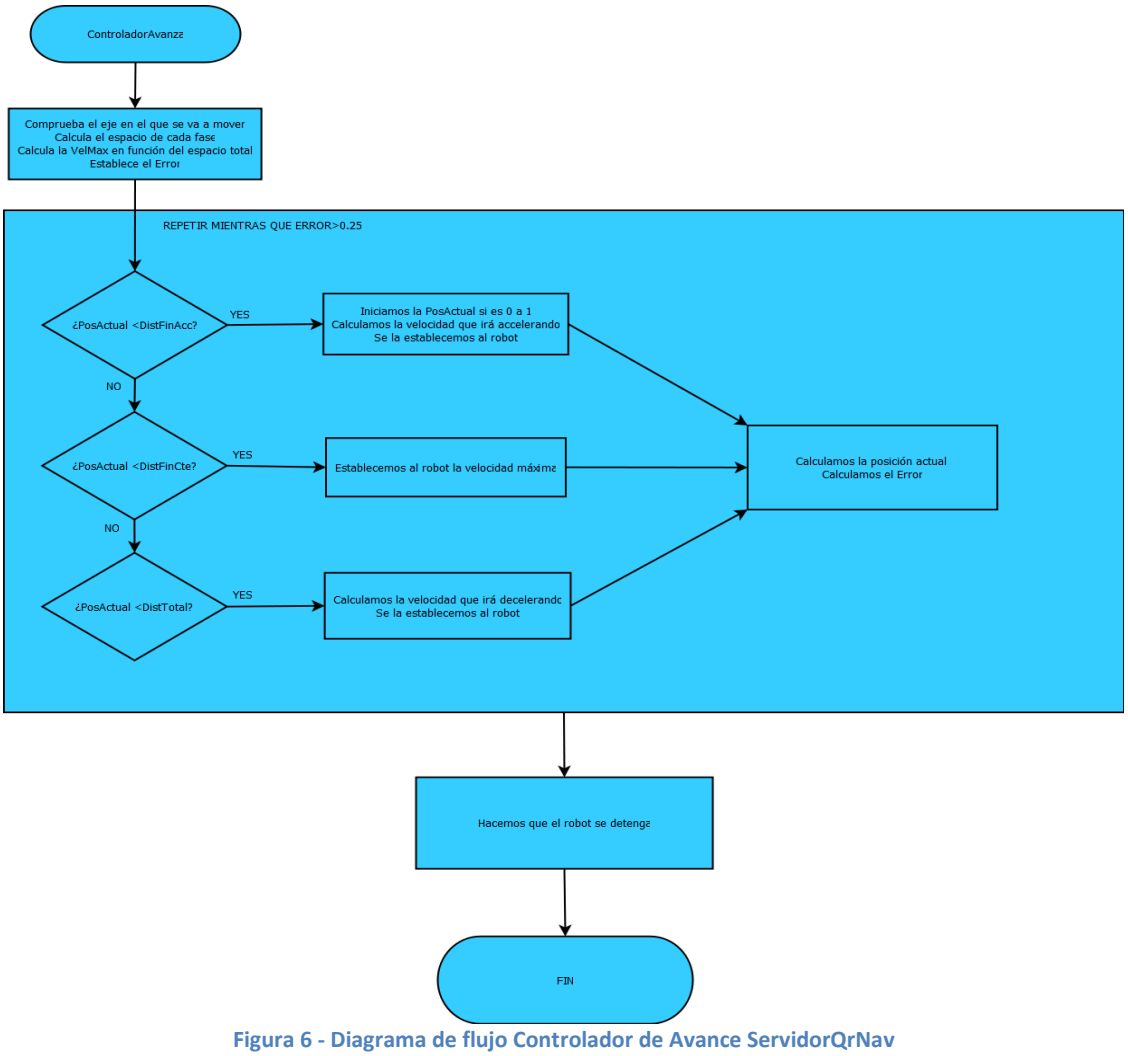

Figura 6 - Diagrama de flujo Controlador de Avance ServidorQrNav

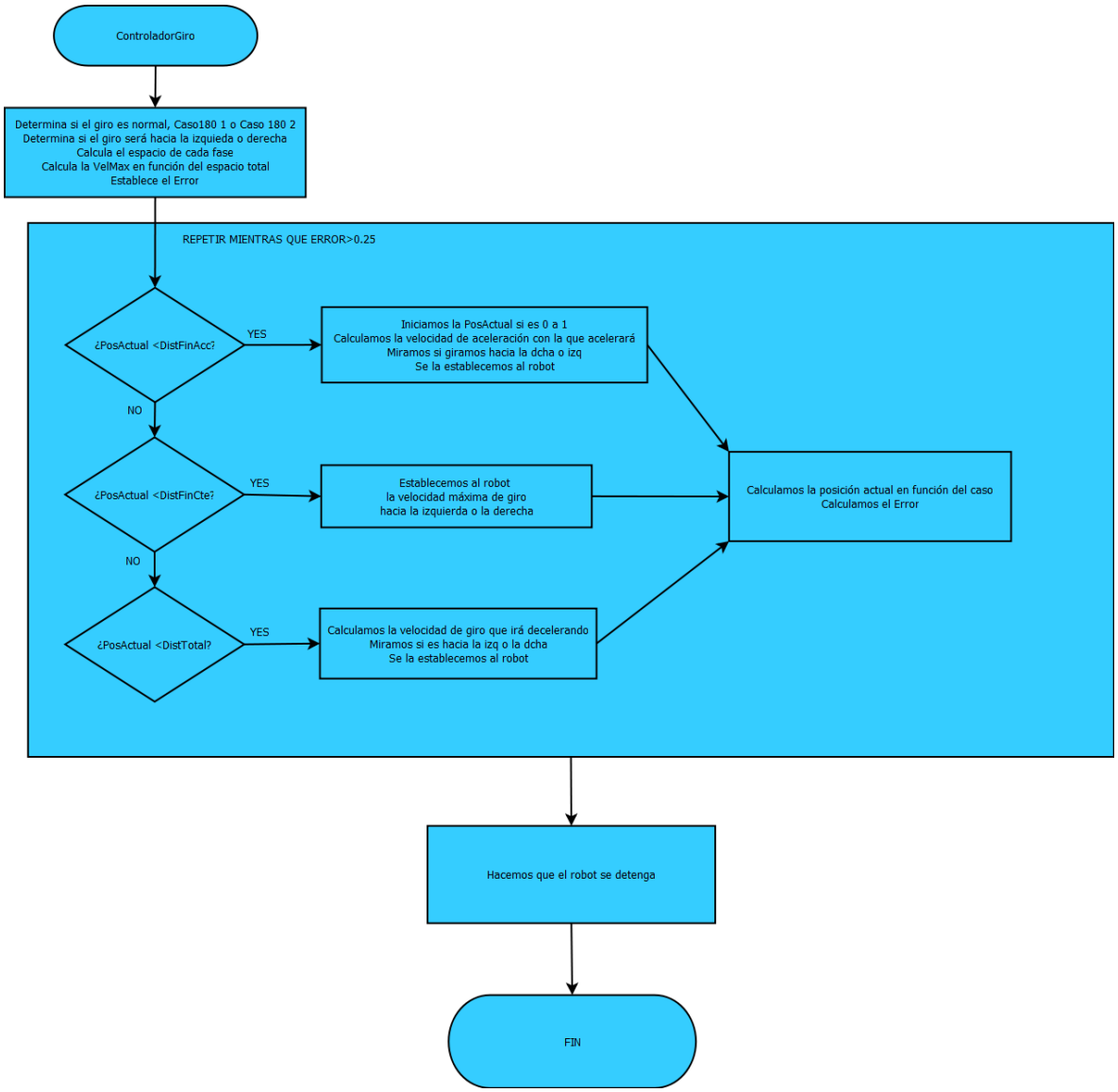

Figura 7 - Diagrama de flujo Controlador Giro ServidorQrNav

Errores

A continuación se presentan algunos de los errores que pueden producirse durante este proceso:

- Los suelos con superficie irregular. Esto provoca fallos en la odometría ya que, cuando se encuentra un saliente, el robot contará más espacio del que existe en una superficie uniforme y, si es un entrante, contará menos espacio. En cambio en suelos con superficies lisas las ruedas pueden llegar a patinar, haciendo que el espacio patinado no cuente en la odometría.

- Las limitaciones del Amigobot que no es capaz de establecer la misma potencia en cada rueda al mismo tiempo, provocando que se desvíe hacia la izquierda cuando se mueve en línea recta.

5.5.4 Colocación del Amigobot

En este apartado se explica el algoritmo de Colocación Qr que permite al ServidorQrNav colocar al robot en una posición centrada respecto al código QR para poder leer (una vez ha realizado la navegación) mediante los parámetros recibidos del QrNavCliente.

Este algoritmo consiste en centrar al robot respecto a un cuadrado rojo usando para ellos su área y su punto medio:

En primer lugar centramos al robot haciendo que su punto medio se encuentre dentro del intervalo [400-200], si el punto medio se encuentra a la derecha de la imagen (>400) significa que el QR está a la derecha del robot por lo que giramos 90º a la derecha, avanzamos recto y volvemos a girar 90º a la izquierda. De esta manera conseguiríamos centrar un poco el código en nuestro campo de visión. Análogamente ocurría si el punto medio estuviera a la izquierda (<200).

En segundo lugar debemos hacer que el área se encuentre dentro del intervalo [40000- 25000], ordenando al robot que avance para acercarse en caso de que el área sea menor puesto que estaremos lejos del código QR y ordenando que retroceda si es mayor ya que estaremos demasiado cerca del código QR.

6. Trabajos relacionados

El presente proyecto es una continuación del proyecto “TAI: Teleoperación de un Amigobot usando dispositivos iOS” de Francisco Javier Esteban Vicente de la Universidad de Salamanca, el cual consiste en una aplicación cliente que se comunica con un servidor conectado al Amigobot para teleoperarlo mediante inclinación o dos botones.

Este proyecto ha servido de base pudiendo reutilizar la conexión mediante un servicio entre el cliente y el servidor y parte de la estructura. Los demás aspectos no han podido ser reutilizados para el presente proyecto, por lo que han sido creados de nuevo y se han introducido otras partes totalmente novedosas.

7. Descripción funcional de la aplicación

En este apartado se realizará una descripción global del proyecto desde un punto de vista tanto de software como de hardware.

En primer lugar, podemos observar en la Figura 8 cómo está estructurado el proyecto desde una vista de hardware.

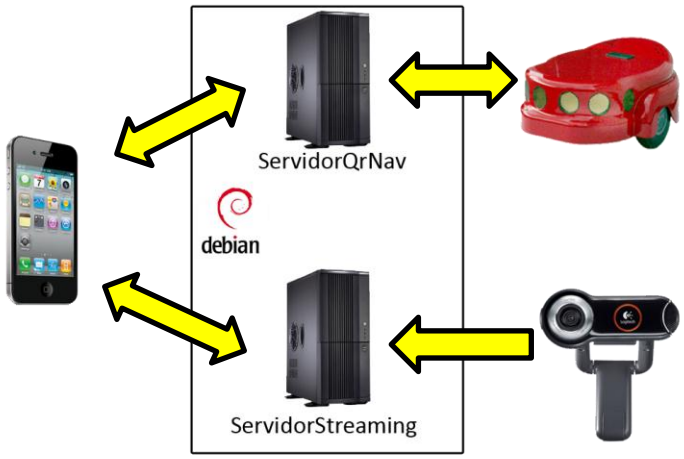

Figura 8 - Estructura Hardware

El proyecto se divide en tres programas:

- **QrNav**, una aplicación para dispositivos iOS que sirve de interfaz para el usuario y que deberá realizar diversas operaciones. Es el encargado de decidir qué proceso debe ejecutarse en cada momento. Se comunica con el ServidorQrNav enviándole las tareas que debe realizar con el robot y con el servidor streaming recibiendo las imágenes de la webcam.

- **ServidorQrNav**, servidor encargado de controlar al robot. Se comunica con la aplicación QrNav recibiendo las tareas que tiene que realizar con el robot y enviándole el resultado de dichas tareas.

- **Servidor Streaming** (*MJPG-Streamer*), servidor que obtiene las imágenes de la webcam y se las transmite a la aplicación QrNav a través de *HTTP*.

En la Figura 9 se observa este paso de mensajes antes descrito:

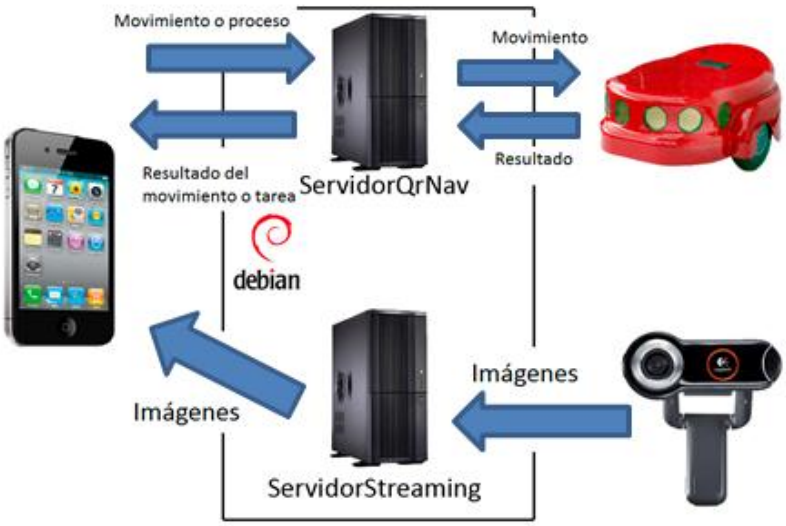

Figura 9 - Estructura paso de mensajes

La aplicación QrNav consta de tres secciones dividas en:

❖ **Bonjour Connect** (Figura 10), nos permite conectar con el ServidorQrNav seleccionando uno de los servicios que se muestran en la lista del dispositivo iOS.

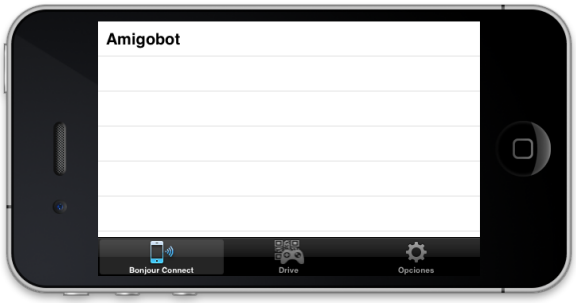

Figura 10- Pestaña Bonjour Connect con servidor disponible

Figura 11 - Pestaña de Mandos en funcionamiento

❖ **Mandos** (Figura 11), esta sección nos permite visualizar lo que el robot ve a través de la webcam, comenzar la recepción de vídeo o pausarla, manejar al Amigobot a través del *joystick*, mandar a la aplicación leer un código QR, cancelar la comunicación en caso de emergencia, etc.

❖ **Opciones** (Figura 12), nos permite modificar unos valores de ajuste para personalizar nuestra aplicación para facilitarnos el manejo del Amigobot a nuestro gusto.

Figura 12 - Pestaña Opciones

8. Conclusiones

En primer lugar, se consideran superados todos los objetivos que se marcaron al comienzo del proyecto. La construcción del proyecto, así como la realización de toda la documentación, me ha servido para afianzar y practicar los conceptos teóricos adquiridos a lo largo de la carrera.

Cabe destacar, que se han comprendido las fases que deben seguirse a la hora de desarrollar un producto software, así como la importancia que tienen las fases previas de análisis y diseño, ya que van a marcar de manera muy determinante la calidad del producto final, así como ayudan a reducir el tiempo total de desarrollo.

A pesar de considerar que el presente proyecto posee una dificultad elevada, al haber utilizado áreas y ámbitos con los que no había tratado previamente (como puede ser Objetive-C, tratamiento de imágenes, robótica, etc) considero que trabajar sobre él me ha resultado beneficioso, puesto que me ha aportado bastantes conocimientos sobre las áreas anteriormente mencionadas. Además, considero que a nivel personal ha sido un motivo de superación puesto que me enfrentaba a algo desconocido pero que he conseguido resolver satisfactoriamente. Sin duda, acerté en la elección del proyecto.

La tarea más costosa y en la que más tiempo se empleó fue, sin lugar a dudas, la transmisión de vídeo streaming en tiempo real, puesto que fue la parte a la que más tiempo dediqué y de la que no existe apenas documentación.

8.1 Líneas Futuras

En este apartado se van a describir algunas de las posibles maneras de ampliar el proyecto:

- Mejorar los errores que tiene la odometría con un robot más potente y ruedas de mayor calidad e incluyendo nuevas partes en el proyecto para tener más puntos de referencia en relación a la posición en la que nos encontramos. Corregir los errores de tratamiento de imágenes provocados por la luz ambiental.

- Cambiar los controladores lineales a controladores PID (propocional integral derivativo) comúnmente usados en el área de robótica.

- Añadir una nueva funcionalidad en la que el usuario pueda elegir entre dos direcciones que lea de un código QR, es decir, al decodificar un código QR el usuario podrá seleccionar la dirección que quiere seguir. Por ejemplo, en un museo, para ir hasta una escultura u otra.

- Crear una pequeña interfaz gráfica para entornos Linux que lanzara al ServidorQrNav y al programa *MJPG-Streamer*, en lugar de tener que hacerlo desde consola y de forma separada.