ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA

"Desarrollo de Aplicaciones y Documentación de las Plataformas Robóticas Pioneer P3-DX y Pioneer3 P3-AT"

Jimena Johanna Morales Balladares
Daniel Fernando Jaramillo Fabara

SANGOLQUI – ECUADOR

2010

CERTIFICACIÓN

Certificamos la realización de este proyecto basado en la documentación de las plataformas robóticas Pioneer P3DX y P3AT por los autores Jimena Morales y Daniel Jaramillo.

Ing. Hugo Ortiz DIRECTOR	Ing. Víctor Proaño CODIRECTOR

AGRADECIMIENTO

Nuestro agradecimiento va para quienes hicieron posible este Proyecto de Grado, a nuestros tutores y amigos del Departamento de Eléctrica y Electrónica de la Escuela Politécnica del Ejército, Ing. Hugo Ortiz e Ing. Víctor Proaño, además un agradecimiento también muy especial a nuestras familias por el apoyo brindado y por su labor cotidiana de orientarnos a ser mejores personas y excelentes profesionales.

DEDICATORIA

Este trabajo va dedicado de manera especial a todas las personas involucradas en nuestra vida cotidiana, profesores y familia en general quienes han hecho posible el desarrollo de este proyecto contribuyendo cada uno con un granito de arena, también va dedicado para todos los aficionados a la robótica, por lo cual esperamos que nuestro trabajo sea de su agrado y esperamos contribuir con una pequeña parte de conocimientos en este fabuloso mundo de la robótica.

PRÓLOGO

En este proyecto de grado se documenta y analiza el funcionamiento de las plataformas robóticas Pioneer3 P3-DX y Pioneer3 P3-AT que posee el Departamento de Eléctrica y Electrónica de la ESPE.

El análisis realizado involucra varios aspectos específicos sobre el uso y funcionamiento de los equipos descritos anteriormente, abarca desde la descripción física de los robots y sus accesorios, hasta los aspectos fundamentales de programación para poder explotar de manera eficiente todos los recursos que brindan estas plataformas robóticas.

Para este proyecto se usan los conceptos de programación de Visual C++, se puede aprender más sobre este lenguaje de programación ya que se involucra aspectos nuevos que permiten desarrollar aplicaciones de manera versátil.

El enfoque de aplicaciones que se detalla en este proyecto comprende aplicaciones sencillas, así como avanzadas, por lo cual esperamos que este trabajo sea de su agrado y de su interés.

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN

1.1. Objetivos del Proyecto

1.1.1. General

- Desarrollar aplicaciones y documentar la funcionalidad de las plataformas robóticas Pioneer3 DX y Pioneer3 AT.

1.1.2. Específicos

- Familiarizarse con el manejo de las plataformas robóticas que utilizan el entorno cliente – servidor.
- Conocer el funcionamiento de los diferentes programas clientes que presenta el fabricante como son ARCOS, ARIA, SonARNL y ACTS así como de sus respectivas interfaces gráficas MobileEyes y SAV.
- Utilizar las herramientas que brinda el fabricante para poder desarrollar las aplicaciones como son MobileSim y Mapper3Basic.
- Conocer y detallar el funcionamiento de los diferentes accesorios que presentan las plataformas así como de sus diversos sensores.
- Documentar todas las funcionalidades de las plataformas robóticas para su posterior uso por parte de los estudiantes.
- Desarrollar aplicaciones de:
  - Seguimiento de trayectorias
  - Control de giros
  - Evasión de obstáculos
  - Control de visión

1.2. Alcance

El alcance del proyecto se enfoca en conocer y documentar las funciones y tareas que pueden realizar las plataformas robóticas Pioneer3 DX y Pioneer3 AT, así como su programación en el entorno cliente – servidor para lo cual se utilizará herramientas como computadores, cámaras de visión y otros accesorios para poder realizar actividades más complejas.

El servidor a utilizar es el mismo con el cual se maneja el microcontrolador del robot llamado ARCOS (Advanced Robotics Control Operating System), y para enlazarse con sus clientes se utilizarán los programas proporcionados por el fabricante como son: ARIA (Advanced Robotics Interface for Applications), SonARNL (Sonar Autonomous Robotic Navigation & Localization), y ACTS (Advanced Color-Tracking Software). Estos servirán para desarrollar las diferentes aplicaciones para las cuales las plataformas fueron diseñadas.

Otro aspecto a enfocarse será la configuración de software de las plataformas, ya que para su correcto funcionamiento requieren de ciertos parámetros dependiendo del entorno en el que se desenvuelvan. Luego de conocer el funcionamiento y desarrollar las diferentes actividades para las que fueron diseñadas las plataformas, se procederá a realizar aplicaciones reales las cuales podrían enfocarse en actividades como vigilancia, reconocimiento de trayectorias, manejo remoto, etc.

1.3. Descripción del proyecto

El proyecto en si tiene como objetivo dar a conocer todas las funcionalidades y capacidades que tienen las plataformas robóticas móviles Pionner3 AT y Pioneer3 DX, para lo cual se mostrará su funcionamiento por partes. En primer lugar se procederá a estudiar todas las características físicas de cada plataforma para saber en qué entornos pueden desenvolverse y que potenciales aplicaciones son las que se podrían desarrollar.

Una vez comprendidas las características de las plataformas se procederá a enfocarse en los accesorios, para poder instalarlos de manera correcta y explotar todo su potencial en las aplicaciones a desarrollarse.

Al tener lista la información necesaria en cuanto al aspecto físico de las plataformas se puede proceder con la parte más importante del proyecto a desarrollarse que es en sí la programación de las plataformas, para lo cual debe entenderse la arquitectura que se maneja y posteriormente comenzar con la programación básica, así como la programación avanzada.

Para entender un poco mejor todo esto en el proyecto se dividirá la parte de programación en dos partes:

- ✓ El programa tipo servidor.
  Para este programa se describirá todo lo necesario en cuanto a configuración y uso que se le puede dar, así como la información necesaria como es la forma de comunicarse, su programación, actualización y varios otros aspectos de mucha importancia.

- ✓ El Programa tipo cliente.
  En esta parte del proyecto se enfocarán los aspectos de programación para realizar las aplicaciones, tanto de programación básica como de programación avanzada, dentro de lo cual se describirán todos los comandos necesarios y las maneras de programar las plataformas robóticas móviles.

Con todos estos elementos se podrá desarrollar las aplicaciones como son seguimiento de trayectorias, control de giros, evasión de obstáculos y control de visión.

CAPÍTULO 2

DESCRIPCIÓN FÍSICA DE LAS PLATAFORMAS

2.1. Preliminares[1]

2.1.1. Qué es un Robot Pioneer?

Un robot Pioneer es una plataforma robótica desarrollada con fines de investigación así como también para realizar aplicaciones ya sean para la vida estudiantil o para la vida cotidiana. Para el caso de estudio se tomarán dos modelos de plataformas desarrolladas por MOBILEROBOTS, las cuales son el Pioneer3 P3-DX y el Pioneer3 P3-AT.

2.1.2. Plataforma de referencia PIONEER

Las plataformas de MOBILEROBOTS establecen ciertas características estándares para proveer de inteligencia a los robots móviles, ya que contienen componentes para el sensamiento y navegación dentro de un entorno real.

Así se han convertido en plataformas de referencia en una gran cantidad de proyectos de investigación, incluyendo varios estudios de la Agencia de Proyectos de Investigación Avanzada del Departamento de Defensa de los Estados Unidos.

Cada plataforma de MOBILEROBOTS tiene un cuerpo robusto de aluminio, un sistema de manejo balanceado (con dos ruedas diferenciales con una ruedilla o con cuatro ruedas con motor independiente), motores DC reversibles controlados electrónicamente, encoders de

[1] Pioneer3 Operation Manuals

alta resolución, energizados por batería. Todos ellos manejados por un microcontrolador interno y un software servidor Mobile-robot.

Estas plataformas vienen con el software de control y un microcontrolador interno, también poseen un Host para realizar aplicaciones con un software tipo cliente que permite el control avanzado del robot, y también para el desarrollo de aplicaciones de acuerdo al entorno.

El desarrollo del software incluye su interfaz propia: ARIA (Advanced Robotics Interface for Aplications), así como ARNetworking, ambas bajo licencia pública GNU y con una documentación completa con librerías totalmente documentadas de C++, Java y Phyton así como de los códigos fuentes.

Muchas de las aplicaciones robóticas desarrolladas vienen generalmente de la comunidad investigadora, incluyendo lenguajes como Saphira desde el SRI internacional, Ayllu de la Universidad de Brandeis, Pyro de Byrn Mawr y College de Swarthmore , Player/Stage de la Universidad de California del Sur y Carmen de la Universidad de Carnegie-Mellon.

2.1.3. Familia de Microcontroladores y Software de operaciones

Actualmente todas las plataformas MOBILEROBOTS utilizan revolucionarios microcontroladores de alto desempeño con un software para control del robot basado en la nueva generación de microprocesadores de 32 bits Renesas SH2-7144 RISC de tal manera que se incluye al microcontrolador P3-SH con ARCOS, que es el software utilizado en este microcontrolador.

2.1.4. Puertos y Alimentación

Las plataformas Pioneer 3 tienen una variedad de puertos de expansión, de alimentación, y de entrada y salida, para el acople e integración con un PC cliente, así como sensores y una variedad de accesorios; todos accesibles desde una interfaz común de aplicación con el software servidor del robot, ARCOS. Entre las características se incluyen:

- ✓ Un microprocesador Renesas SH2 de 44.2368Mhz y 32 bit RISC con una RAM de 32K y 128K Flash.
- ✓ 4 Puertos seriales RS-232 (5 conectores) configurables desde 9.6 hasta 115.2 kilobaudios.
- ✓ 4 Arreglos de sonares ampliables hasta 8 sonares cada uno.
- ✓ 2 Parachoques de 8 bits con entradas digitales.
- ✓ Puerto de entrada y salida del Parachoques con 8 bits, E/S digital, Entrada analógica, y alimentación de 5/12 VDC.
- ✓ Puerto para la corrección del cabeceo mediante el Giroscopio.
- ✓ Puerto para el sensor de inclinación.
- ✓ Puerto para el Joystick de 2 ejes y 2 botones.
- ✓ Panel de Control para el usuario.
- ✓ Conector Serial HOST del Microcontrolador.
- ✓ Indicadores de alimentación principal y un Led bicolor para el nivel de batería.
- ✓ 2 Interruptores auxiliares de encendido (5 y 12VDC) con sus respectivos leds indicadores.
- ✓ Pulsadores para el control de Motores y de RESET.
- ✓ Alarma programable.

Interfaz interna de alimentación del motor (sistema de tracción) con PWM y líneas de control de dirección del motor y 8 bits de entrada digital.

2.2. Pioneer3 P3-DX[2]

2.2.1. Descripción

La plataforma robótica P3-DX cuenta con muchas características interesantes que se describen a continuación, comenzando por decir que esta plataforma funciona bajo una arquitectura denominada cliente – servidor, el cual es un modelo de comunicación en el que el microcontrolador del robot interpreta las órdenes de un programa (cliente) localizado en una PC aparte, el cual puede ser una laptop o una PC de escritorio; también

[2] Pioneer3 Operation Manuals

se puede usar un PC interno, pero este es un accesorio aparte que se describirá posteriormente.

Esta plataforma cuenta con varios puertos entre los cuales hay algunos que son para los accesorios propios del robot, y otros como son los puertos auxiliares para poder implementar otro tipo de sensores, según la aplicación que se vaya a desarrollar.

Así por ejemplo se tiene una gran variedad de sistemas que se pueden implementar entre los que destacan el sistema de visión, las comunicaciones basadas en Ethernet, láser, DGPS, y algunas otras funciones autónomas.

El P3-DX tiene capacidad para 3 baterías las cuales pueden entregar una potencia de 252 vatios si se encuentran cargadas totalmente. Además viene con un anillo de 8 sonares delanteros con opción de incrementar un anillo trasero de 8 sonares. Tiene unos motores potentes y llantas de 19 cm de diámetro con las que puede alcanzar velocidades de 1.6 m/s y llevar cargas de hasta 25 Kg.

A fin de mantener la precisión en los datos a estas velocidades, utiliza codificadores de 500 pasos (encoders) lo que le permite una mayor corrección de deslizamientos y giros. Posee también sistemas de detección que pueden utilizarse para realizar aplicaciones más avanzadas, dentro de estos sistemas se tiene, el de navegación basado en láser, parachoques, pinzas, visión, cámaras estéreo y brújulas.

La plataforma P3-DX con su programa incluido ARIA tiene la habilidad de realizar actividades como por ejemplo:

- Modo de evasión de obstáculos
- Manejo controlado por teclado o Joystick
- Plan de caminos con gradiente de navegación
- Mostrar en un mapa las lecturas de sonares y sistemas láser
- Localización usando el sonar (con opción de actualización mediante láser)

- Sensor de comunicación y Control, información relacionada con el sonar, los codificadores de los motores, entradas y salidas dispuestas por el usuario e información del estado de la batería.
- Ejecución de programas en C y C++.
- Simulación de comportamientos fuera de línea mediante el simulador MobileSim.

El robot tiene múltiples accesorios para desarrollar un sin número de aplicaciones, como se puede observar en la figura 2.1. Con un sistema de mapeo y navegación láser y MobileEyes, el robot puede mapear construcciones y actualizar constantemente su posición dentro de unos pocos centímetros mientras viaja dentro de las áreas asignadas.

Figura. 2.1. **Grupo de robots Pioneer3 P3-DX**

Algunas de las aplicaciones potenciales del robot son:

- cartografía
- tele operación
- localización
- monitoreo
- reconocimiento
- visión
- manipulación

- cooperación
- y otros comportamientos

El P3 – DX funciona mejor en superficies duras, puede atravesar pequeñas gradas y pasar por cables de alimentación así como subir rampas para discapacitados.

2.2.2. Componentes

El Pioneer P3-DX tiene un conjunto de componentes para uso general, en el que constan:

- Cuerpo con puerta de acceso para las baterías
- 1 Batería
- 2 Llantas y una pequeña rueda guía
- 2 Motores con codificadores
- Anillo frontal de sonares
- Microcontrolador
- Tablero del sonar
- ARCOS programa tipo servidor del microcontrolador
- Bus de E/S integrado en el hardware y en el programa ARIA.
- Interfaz robótica ARIA para desarrolladores.
- Manual de operación

Además el robot requiere:

- Comunicación con un PC cliente, mediante uno de los siguientes métodos:
  - radio modem inalámbrico.
  - conexión robot – portátil.
  - enlace robot – PC de escritorio
  - conexión a una computadora a bordo
- Un cargador de baterías
  - carga estándar
  - alta capacidad para reducir el tiempo de carga del 80% o utilizar una computadora como fuente de alimentación (requiere 3 baterías).

2.2.3. Opcional

Entre los accesorios opcionales se destacan:

- Ethernet Inalámbrico
- Mapeo y Navegación Láser
- Brazo robótico Pioneer
- Cámara de color con inclinación
- Cámara de medición estéreo
- Cámara omnidireccional de 360 grados.
- Seguimiento de color
- Brújulas y sensores de posición
- Parachoques
- Pinzas
- Estación de carga

2.2.4. Especificaciones técnicas

La plataforma P3-DX tiene un cuerpo de aluminio de 44cm x 38cm x 22cm con llantas de 19 cm de diámetro. Los dos motores tienen una relación de transmisión de 38.3:1 y contiene codificadores de posición de 500 pasos. Esta plataforma puede rotar moviendo ambas llantas o puede formar un círculo de 32 cm de radio, la pequeña llanta trasera balancea el movimiento.

El P3-DX puede subir pendientes de 25 grados y alturas de 2.5cm. En un piso plano el P3-DX puede moverse a velocidades de 1.6 m/s. A velocidades menores puede llevar cargas de más de 23 Kg. Las cargas incluyen baterías y los accesorios los cuales deben estar balanceados para una correcta operación del robot.

Además de los codificadores de los motores el P3 – DX incluye 8 transductores de ultra sonidos cuyo arreglo provee una cobertura hacia adelante de 180 grados. Su rango de lectura va desde los 15 cm. hasta los 7m. La puerta de acceso hacia las baterías del P3-DX permite un intercambio fácil en caliente y provee de energía entre 18 a 24 horas con 3 baterías totalmente cargadas. Con un cargador de alta capacidad el tiempo de carga se reduce a 2.4 horas.

La parte posterior de la plataforma removible del P3-DX permite el acceso a cualquier computadora opcional interna además del microcontrolador y la tarjeta interna que contiene todos los puertos de entrada y salida entre los cuales constan 8 entradas y 8 salidas digitales más un puerto Análogo/Digital, sus cuatro puertos digitales de entradas pueden ser configurados como puertos A/D y sus 4 salidas digitales pueden utilizarse como salidas PWM. Estas entradas y salidas están integradas en la estructura interna y son accesibles a través de ARIA.

2.3. Pioneer3 P3-AT[3]

2.3.1. Descripción

El P3-AT es una plataforma robótica versátil todo terreno y es por ello que ha sido escogida por muchos becados de DARPA (Agencia de Investigación de Proyectos Avanzados de Defensa) y otras personas que requieren un robot de alto desempeño.

El robot P3-AT ofrece las mismas características que el P3-DX, como se puede observar en la figura 2.2, con algunas diferencias significativas en las cuales cabe destacar sus 4 motores y sus llantas todo terreno, que la hacen muy útil para aplicaciones en exteriores, ya que circula sin inconveniente por superficies de tierra, piedra o pavimento y subir pendientes de hasta 45 grados. Puede alcanzar velocidades de hasta 0.7 metros por segundo y una capacidad de carga de máximo 40Kg.

El P3-AT usa codificadores de 100 pasos con una corrección de tipo inercial recomendada para estimación y compensación de deslizamientos de dirección. Además esta plataforma brinda la posibilidad de poner otro tipo de accesorios como son pinzas, sistemas GPS y cámaras de visión estéreo.

Cabe decir que aunque la plataforma está diseñada para el uso en exteriores, la misma no es a prueba de agua.

[3] Pioneer3 Operation Manuals

Figura. 2.2. **Plataforma Robótica P3-AT**

2.3.2. Componentes

El Pioneer P3-AT consta de:

- 1 Batería
- 4 Llantas para exteriores
- 4 Motores con codificadores
- Microcontrolador
- ARCOS programa tipo servidor del microcontrolador
- Bus de E/S integrado en el hardware y en el programa ARIA.
- Interfaz robótica ARIA para desarrolladores.
- Manual de operación

Además el robot requiere:

- Comunicación con un PC cliente, mediante uno de los siguientes métodos:
    - radio modem inalámbrico.
    - conexión robot – portátil.
    - enlace robot – PC de escritorio
    - o conexión a una computadora a bordo

- Un cargador de baterías
  - carga estándar
  - alta capacidad para reducir el tiempo de carga del 80% o utilizar una computadora como fuente de alimentación (requiere 3 baterías).

2.3.3. Opcional

Los accesorios opcionales del P3-AT son:

- Ethernet inalámbrico
- Medidores láser
- Sonar frontal y trasero
- Cámaras a color con inclinación
- Cámaras nocturnas
- Cámaras de medición estéreo
- GPS
- Brújulas y sensores de posición e inclinación.
- Parachoques
- Pinzas
- Sistema de operación basada en Internet

2.3.4. Especificaciones técnicas

El robot P3-AT tiene un cuerpo de aluminio de las siguientes dimensiones: 50cm x 49cm x 26cm con llantas de 21.5cm de diámetro para manejo en exteriores. Los cuatro motores tienen una relación de transmisión de 38.3:1 y contienen codificadores de 100 pasos. Esta plataforma puede rotar y moverse hacia un lado o puede formar un círculo de 40 cm de radio.

El P3-AT puede subir pendientes de 45 grados y altitudes de hasta 9cm. En un piso plano la plataforma puede moverse a velocidades de 0.7 m/s. A velocidades más lentas en terrenos más planos puede cargar sobre los 40Kg. La capacidad de carga incluye las baterías y todos los accesorios y debe estar balanceada apropiadamente para una operación efectiva del robot.

La plataforma consta de una puerta de fácil acceso para el intercambio de las baterías de una manera fácil y el funcionamiento con 3 baterías totalmente cargadas puede ir desde 3 a 6 horas.

La plataforma P3-AT tiene una nariz removible que da acceso tanto a la computadora a bordo (en caso de tenerla) como también a la tarjeta principal que contiene los puertos para los accesorios así como también al microcontrolador.

En el microcontrolador se tienen varias estradas y salidas que el usuario puede integrar y acceder a través de ARIA.

CAPÍTULO 3

PERIFÉRICOS Y ACCESORIOS[4]

Los robots Pioneer3 tienen varios accesorios opcionales. Para conveniencia, se incluyen una descripción de los accesorios comúnmente integrados para los robots Pioneer Mobile, como se puede observar en la figura. 3.1.

Figura. 3.1. **Robots Pioneer Mobile**

3.1. Joystick

Este accesorio es para el manejo manual del robot y consta de una palanca para control de movimiento, un potenciómetro para control de la velocidad y dos botones de color verde y amarillo. Estos botones son para control del accionamiento de los motores y cada uno cumple con una función específica, la cual se maneja en base a programación y se describirá posteriormente.

[4] Pioneer3 Operation Manuals

El joystick también consta de un conector específico para las plataformas robóticas, el cual va dirigido hacia el puerto respectivo en la tarjeta del microcontrolador.

3.2. Parachoques

Este es un accesorio que sirve para el monitoreo de obstáculos ya que provee el sensamiento del contacto con un obstáculo cuando otros sensores han fallado en detectarlo.

El parachoques es en forma de anillo y se ha segmentado para contacto en cinco diferentes posiciones, como se puede observar en la figura 3.2.

Electrónicamente y mediante programación, los parachoques disparan eventos digitales los cuales se ven reflejados en un registro de valores y permiten que el robot reconozca en el momento que el robot se ha impactado con algo, esto permite realizar varias acciones según la programación, que se verá posteriormente.

El parachoques se conecta al puerto de conexión respectivo de la tarjeta del microcontrolador.

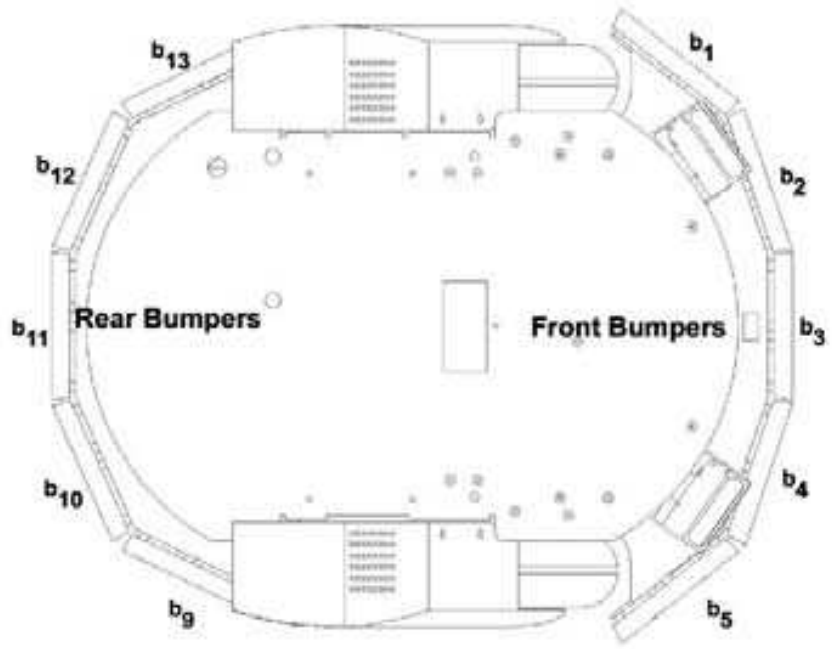

Figura. 3.2. **Distribución de los parachoques en la plataforma robótica P3-DX**

3.3. Recarga Automática

Los plataformas robóticas Pioneer3 DX y AT poseen un accesorio de recarga automática la cual tiene mecanismos de accionamiento tanto manual como automática. Los controles a bordo pueden activar este mecanismo con pulsar el botón de recarga automática o accionándolo mediante los comandos de ARCOS.

Mientras se encuentra conectado, existe un circuito interno que optimiza la carga a sus tres baterías de 7 Amperios/hora y 12 Voltios. (La corriente máxima de carga es de 6A) y provee suficiente energía para la operación de todos los sistemas a bordo. Se puede observar la localización de las baterías en la siguiente figura 3.3.

Figura. 3.3. **Baterías**

3.4. Radio Control y accesorios

Las plataformas Pioneer P3-DX y P3-AT son los servidores en una arquitectura cliente servidor. Un computador es el cliente que se encarga de realizar las actividades inteligentes y aplicaciones de las plataformas móviles. El cliente puede ser ya sea una computadora a bordo, una computadora portátil o una computadora de escritorio que se pueden conectar a través de radio módems o vía módems serial-Ethernet inalámbricos, como se observa en la figura 3.4.

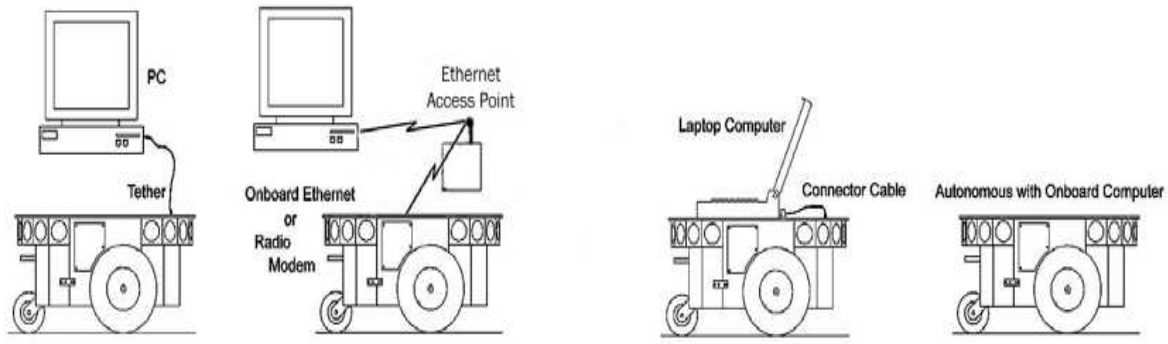

Figura. 3.4. **Formas de conexión de las plataformas robóticas cliente – servidor**

En todos los casos el cliente debe estar conectado al HOST interno o al puerto serial del panel de control de usuario, para que el robot y el programa trabajen juntos. Para las portátiles o computadoras de escritorio, las conexiones se realizan vía cable serial.

Los radio módems pueden reemplazar el cable por un medio inalámbrico. Si se da el uso de radio módems se debe considerar que el uno se coloque dentro del robot y se conecte al puerto serial HOST y el otro modem se conecte al puerto serial de la PC que vaya a utilizarse como cliente. Por lo tanto, en esta configuración existirá una computadora cliente dedicada.

El uso de Radio Ethernet es un poco más complicado, pero es el mejor método ya que permite el uso de varias computadoras en la red para poder ser clientes del robot. Si se tiene una PC a bordo (accesorio del robot), se puede proveer la conexión de Radio Ethernet mediante una tarjeta PCMCIA Ethernet inalámbrica.

Se provee un accesorio Ethernet a Serial el cual se conecta con el microcontrolador del robot. Funciona de tal manera que automáticamente transforma los paquetes basados en comunicación Ethernet en Serial para que éstos puedan ir al microcontrolador y también regresar.

El manejo de un robot a través de manera inalámbrica vía Ethernet con una computadora a bordo es un poco diferente que con un dispositivo Ethernet – Serial. En el primer caso, se corre el programa tipo cliente en la computadora a bordo y se usa el dispositivo

inalámbrico Ethernet para monitorear y controlar la operación de la PC. Para el segundo caso se corre el programa cliente en una computadora LAN remota.

Una mayor desventaja del manejo mediante el dispositivo inalámbrico Ethernet a Serial es que requiere de una conexión consistente con el robot. Una interrupción en la señal de radio (es común que ocurra en la mayoría de instalaciones modernas) permitirá un pobre desempeño del robot y un corto radio de operación.

Es por ello que se recomienda correr el programa cliente en un PC interno, ya que provee un sistema más robusto y autónomo, especialmente cuando éste tiene su propio dispositivo inalámbrico Ethernet. En esta configuración se puede correr el programa tipo cliente e interactuar con el microcontrolador del robot localmente y simplemente contar con la conexión inalámbrica para exportar y operar los controles del cliente. Mayormente, el PC interno es utilizado y necesitado como soporte para mediciones láser, o para procesar tareas de visión en vivo.

3.5. PC Integrado

3.5.1. Descripción

Este accesorio se monta justo atrás de la nariz del robot. La PC integrada para este tipo de plataformas Pioneer3 vienen con 4 puertos seriales, comunicación 10/100 Base-T Ethernet, Puertos para Monitor, Teclado y Mouse, dos puertos USB y un puerto para el disco duro.

Para funciones adicionales como por ejemplo sonido, video, bus PCMCIA y Ethernet inalámbrico, el PC acepta interfaces PC104 y PC104-plus que se encuentra en la tarjeta madre, como se observa en la figura 3.5.

Una conexión de alimentación de 5VDC viene de un convertidor especial de DC localizado muy cerca. Un disco duro especial es montado en la nariz del robot en medio de un ventilador y un parlante de computador.

Figura. 3.5. **PC Integrado**

El PC interno se comunica con el microcontrolador a través del puerto serial HOST y su puerto serial COM1 bajo la plataforma Windows o /dev/ttyS0 bajo plataforma Linux. El microcontrolador automáticamente escoge en la conexión HOST al PC cuando el programa tipo cliente abre el puerto serial. De otra manera el PC no interfiere con los clientes externos conectados a través del puerto serial compartido en el Panel de Control del Usuario.

3.5.2. Panel de control de la computadora

Los puertos de comunicación y de control, así como interruptores e indicadores del PC interno se encuentran en el Panel del control como se indica en la figura 3.6. Se localiza en la parte derecha del robot DX y en el caso el AT se sitúa a lado de los controles de usuario en la parte superior. Los puertos y controles usan conectores comunes: por ejemplo para el monitor conectores DSUB y para el Mouse y el teclado conectores PS/2, El conector Ethernet utiliza el estándar 10/100 Base-T con un enchufe estándar RJ-45.

El interruptor de encendido controla directamente a la PC interna. El led PWR se prende cuando la computadora ha sido energizada; el led HDD se prende cuando el disco duro se encuentra activo, y finalmente el botón de Reset reinicia la PC.

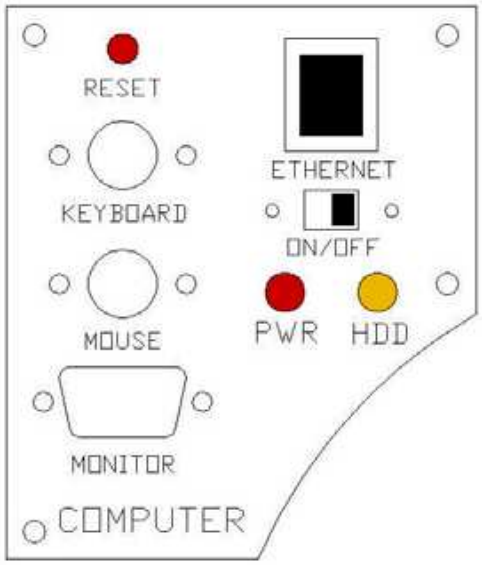

Figura. 3.6. **Panel de control del robot P3-DX**

3.5.3. Operación basada en el PC Integrado

A continuación se detalla una breve explicación del funcionamiento del PC interno. Cabe aclarar que las Plataformas MobileRobots y sus programas funcionan bajo las siguientes plataformas operativas como son Windows, Linux RedHat o Debian. De acuerdo a esto se manejarán estos sistemas operativos en la PC interna.

Cuando se instala y configuran los programas de las plataformas robóticas y sus accesorios se instalan típicamente en /usr/local en Linux y en C:\Program Files\MobileRobots en Windows. De esta manera se instalan los drivers apropiados de los diferentes accesorios, tarjetas de expansión así como controladores de video, audio apropiado. Para ello se revisará más adelante para cada accesorio en específico como por ejemplo para la visión se requiere configurar la interfaz ACTS.

La primera vez que se accede a la PC interna, se recomienda que se sitúe al robot en un sitio en el cual sus llantas se encuentren en el aire para evitar así movimientos inadvertidos y tener luego problemas con las conexiones externas. Se debe enchufar un teclado, monitor y Mouse a sus respectivos tomas en el panel del control. Se debe mover el interruptor de encendido para prender el computador interno.

Después del arranque se inicia la sesión del sistema. Allí se observa que se han creado dos usuarios: uno con sistemas comunes y permisos para lectura y escritura de archivos y el otro con acceso total al programa del PC y el sistemas operativo, conocido como Root en Linux y Administrador en Windows. Si se encontrase alguna clave (usualmente no la hay) debe ser ‘activmedia’.

Cuando se conecta directamente, es recomendable que se inicie con capacidad de accesos total de tal manera que se pueda realizar ajustes en el sistema, cambio de contraseñas, añadir usuarios y configurar parámetros de la red. Hay que tomar en cuenta que bajo la plataforma Linux no es posible iniciar remotamente como root, se debe iniciar como un usuario común y utilizar el comando ‘su-‘ para alcanzar el estado de súper usuario.

Una vez iniciado en un sistema Windows, es cuestión de hacer click con el Mouse para seleccionar los programas y aplicaciones. Con Linux se debe usar el comando ‘startx’ para habilitar el escritorio X-Windows, y el entorno GUI, solo mientras es necesario. Se puede desarrollar algunas actividades de inicio rápido de esta manera, aunque no sería práctico ya que se encuentran conectados el teclado, monitor y Mouse.

Se puede remover estos mientras el sistema se encuentre activo bajo su propio riesgo. Por ello se recomienda realizar estas actividades de inicio rápido en un computador aparte y una vez probadas correrlas y conectarlas a la red del robot mediante una conexión inalámbrica de preferencia.

3.5.4. PC en la red

El conector RJ-45 en el Panel de control de la computadora proporciona un enlace directo de la red con la computadora interna con características de conexión Ethernet 10/100 Base-T. Si se tiene la opción de una tarjeta PCMCIA en el PC se puede insertar una tarjeta Ethernet inalámbrica. La Antena inalámbrica debe colocarse en la parte superior del cuerpo del robot.

Para completar la instalación inalámbrica, se tendrá que proporcionar un punto de acceso a la LAN (se presenta como un accesorio con la mayoría de las unidades). Se conecta el punto de acceso a uno de sus hubs o conmutadores LAN. No se requiere una configuración especial ya que se usa el modo de operación por defecto: cliente-servidor.

Por defecto se tiene el PC instalado con sistemas preestablecidos y probados en una dirección IP fija con una red con configuración clase C. Se debe asignar la misma dirección IP tanto a los puertos de la Ethernet por cable, como a la inalámbrica, por lo general la dirección es 192.168.1.32.

Aunque no es necesario hacerse problema por los drivers o configuraciones de bajo nivel del dispositivo, es posible que antes establecer una conexión de red con el PC interno (no el microcontrolador del robot), aun cuando se coloque un cable cruzado Ethernet a otra PC, se deberá primero reconfigurar los parámetro del la red de la PC del robot.

En pocas palabras para crear una red, en Windows se debe ir a Panel de control, luego Conexión de red, allí se debe crear una red o si se quiere mover los parámetros de otra red existente dando un click con el botón derecho y eligiendo propiedades en donde se podrá cambiar la dirección IP y otros detalles más.

Desde Windows se usa el Panel de Control, Redes, para habilitar o deshabilitar un dispositivo en particular. Con Windows, se necesita una aplicación especial de control remoto para establecer una conexión basada en GUI desde un equipo remoto hacia la PC interna ubicada en la red; por ejemplo VNCserver o XWin32.

Se debe tomar en cuenta que, con la PC interna y conexión inalámbrica Ethernet, a diferencia de los dispositivos inalámbricos Ethernet-serial, no se puede conectar con el microcontrolador del robot directamente a través de la red, es decir, no puede ejecutar una aplicación cliente, como la demostración de ARIA en un PC remoto y optar por conectarse directamente con el servidor del robot seleccionando la dirección IP de la PC interna.

Por el contrario se debe correr la aplicación cliente en el PC interno y exportar la pantalla y los controles a través de la red hacia la PC remota (de preferencia), o se debe usar programas basados en el IPTHRU de ARIA (ver el ejemplo de programa en Aria / ejemplos) para realizar las conversiones IP a serial que necesita la conexión cliente-servidor.

3.5.5. UPS y Generador de Energía

Para proteger los datos del PC interno del robot, se ha habilitado un sistema de detección en ARCOS y un programa que actúa como UPS que accionaría el cierre del sistema operativo en caso de una condición persistente de batería baja.

ARCOS pone el pin 6 DSR del puerto serial del HOST a RS – 232 en alto y pone el Pin 9 en bajo cuando el microcontrolador está funcionando normalmente y la energía de la batería del robot está por encima de los parámetros seteados en la FLASH (ShutdownVolts), que por defecto son de 11 VDC. El Pin RI se activa cuando el nivel de energía por debajo de los ShutdownVolts.

El Generador de energía ejecutado en el sistema a bordo Linux o ups.exe; corriendo bajo Windows, detecta el cambio de estado e inicia el apagado del sistema operativo después de un breve período de tiempo, el cierre del sistema puede ser cancelado por el aumento de la tensión de batería, por ejemplo, si se conecta el cargador.

El Generador de energía monitorea el puerto RI del HOST serial en /dev/ttyS0. Windows ups.exe requiere una puerto serie dedicado, el COM2 en los sistemas actuales, y prefiere monitorear la línea CTS. En consecuencia, el alambre de conexión serial del PC interno será diferente para Linux frente a Windows. La conexión de los pines va en base a un puerto serial RS-232 como se muestra la tabla 3.1.

3.6. Giroscopio

Este accesorio es usado principalmente para la corrección de giros debido a deslizamientos, así como también para el uso de aplicaciones donde el robot necesita

mayor precisión en los giros. El giroscopio proporciona una tasa de rotación máxima de 300 grados por segundo que se almacenan en el puerto analógico AN6 del microcontrolador del robot. En todas las versiones, ARCOS y el cliente ARIA proporcionan apoyo al giroscópico.

PIN	SEÑAL	DESCRIPCION
1	Sin conexión	
2	TXD	Señal de salida
3	RCV	Señal de entrada
4	DTR	La entrada detecta el dispositivo e intercambia en la transmisión y la recepción en el microcontrolador
5	GND	Común
6	DSR	Salida cuando el microcontrolador esta energizado
7	Sin conexión	
8	Sin conexión	
9	HRING	Salida de bajo voltaje que indica al PC que se apague

Tabla. 3.1. **Descripción de pines del conector serial del HOST**

Así mediante programación el microcontrolador puede adquirir, el promedio, y datos relacionados con el giroscopio para utilizarlos en el programa cliente ARIA, el cual utilizará los datos para corregir y calcular la posición actual del robot x, y, z corrigiendo los errores debido al deslizamiento de la rueda. Los parámetros de ajuste del giroscópico se encuentra en el valor del GyroScaler que está ubicado en los parámetros de ARIA respectivos (. "p") por ejemplo el archivo; p3dx-sh.p.

Para ARCOS, su versión 2.0 y posteriores, hay apoyo del lado del servidor para el giroscopio con una resolución más fina y una calibración más fiable. Ya que cuenta con parámetros de rotación configurables por separado tanto en el sentido horario como el anti horario, debido a que estos parámetros tienden a ser diferentes para el cálculo de la corrección de giro.

Los valores basados en FLASH se encuentran permanentemente más asociados a la plataforma y es mucho menos probable que sean cambiados o movidos, como sucede a menudo con la mayor parte de plataformas independientes del programa tipo cliente ARIA.

Si se desea desactivar el giroscopio se lo puede hacer mediante programación y en este caso actuarán solo los codificadores de pasos ubicados en los motores de las ruedas.

3.7. LCD

La pantalla de cristal líquido (LCD) proporciona una ventana donde se muestra el funcionamiento y estado del microcontrolador del robot Pioneer3 como se ve en la figura 3.7, se coloca en uno de los tres puertos seriales auxiliares (normalmente AUX3, pero se puede reubicar cambiando el parámetro FLASH correspondiente al LCD), el módulo soporta cuatro líneas de 20 caracteres para mensajes de estado, y tiene dos interruptores de selección y desplazamiento que permiten revisar valores adicionales de funcionamiento, así como activar y desactivar funciones de servidor.

Figura. 3.7. **Pantalla LCD equipada con el Robot P3-AT**

En modo de funcionamiento normal, la primera línea (# 1) de la pantalla LCD tiene un hilo delgado que actúa como un indicador de actividad del microcontrolador, y el voltaje actual de la batería. El valor parpadea cuando el voltaje de la batería cae por Debajo del ajuste de flash LowBattery. Cuando los motores se enganchan ya sea a través de una conexión de cliente, con el botón de Motor, cuando se conduce con el joystick, o por medio de la pantalla LCD, un carro animado aparece en la esquina superior izquierda de la pantalla LCD, como en la figura 3.8.

Sus ruedas cambian a las letras S o B, si la plataforma para debido al motor o al parachoques, respectivamente. Parpadea cuando se deshabilitar el modo JoyDrive a prueba de errores. Una E parpadeante aparece cuando se presiona el botón STOP en el AT (DX accesorio).

Las dos líneas de la mitad permanecen inactivas la mayor parte, aunque a veces se utilizan para mostrar información de error, tales como durante el apagado del sistema, debido a un nivel muy bajo de la batería (ShutDownVolts que es un parámetro FLASH). Se puede imprimir allí sus propios mensajes con el comando mediante programación de manera temporal, a menos que piense actualizar regularmente el mensaje.

Figura. 3.8. **Muestra de información del estado del programa cliente**

La línea inferior # 4 muestra mensajes de estado del microcontrolador, como cuando se conecta con un cliente o cuando el batería está baja. Se puede oprimir el interruptor de desplazamiento o de selección para activar el modo de pantalla interactiva.

El interruptor de desplazamiento cambia el mensaje de la pantalla y los actúa para cancelar un cambio pendiente. En todos los casos, la pantalla vuelve al modo normal, cancelando toda opción pendiente, si no se presiona el botón de Selección o Desplazamiento después de unos segundos.

3.8. Cámara de visión CANON[5]

3.8.1. Descripción

La VC-C50i de CANON, como se muestra en la figura 3.9, es una cámara versátil que cuenta con un sistema óptico de última generación para un control de la seguridad de alto nivel. Entre sus ventajas está un amplio rango de vigilancia y un rendimiento excelente en condiciones de poca luz. Todas estas características hacen que sea muy práctica para el desarrollo de aplicaciones con las plataformas robóticas entre las cuales se pueden destacar

[5] VC-C50i CANON User Manual

reconocimiento de objetos, colores, trayectorias, operaciones de vigilancia y rescate si se desea realizar aplicaciones un poco más complejas.

Figura. 3.9. **Cámara VC-C50i de CANON**

3.8.2. Características

- Cámara panorámica con inclinación y zoom, con 9 posiciones predeterminadas.
- Zoom óptico de 26x con teleobjetivo y 12x de zoom digital.
- Funcionamiento en condiciones de hasta 1 LUX de luminosidad.
- Iluminador de infrarrojos incorporado
- Fácil configuración y conectividad avanzada
- Función de control en cascada para hasta 9 cámaras

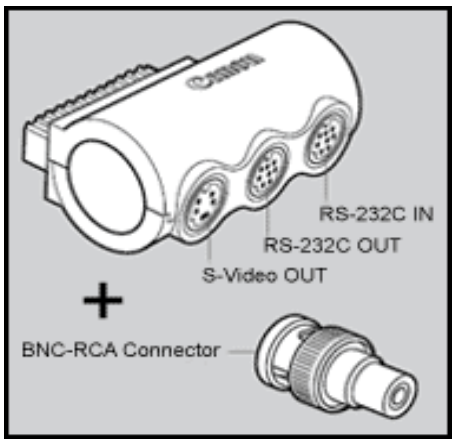

Figura. 3.10. **Adaptador VC-X3 para conexión RS-232C o Video**

- Conectividad con RS-232C o S-Video, con su adaptador como la figura 3.10.
- Inclinación de 120 grados y un rango de 200 grados de vista panorámica
- Incluye control a distancia
- Posee un CCD de 400.000 pixeles que garantizan imágenes de alta calidad.

3.9. Gripper

3.9.1. Descripción

El Gripper Pioneer posee 2 grados de libertad y se coloca en la parte frontal de las plataformas robóticas Pioneer3 DX o Pioneer3 AT. Los dos movimientos que posee radican en la apertura y cierre de las pinzas y en la subida y bajada de todo el mecanismo lo que le permite coger objetos que se encuentren en el suelo y alzarlos hasta una nueva ubicación.

Los movimientos de las pinzas en general se encuentran controlados por dos motores DC reversibles, cuya dirección y potencia están controladas mediante las líneas de salida digital del microcontrolador del robot Pioneer. Internamente posee interruptores de límite que sensan la posición de las pinzas así como de su altura.

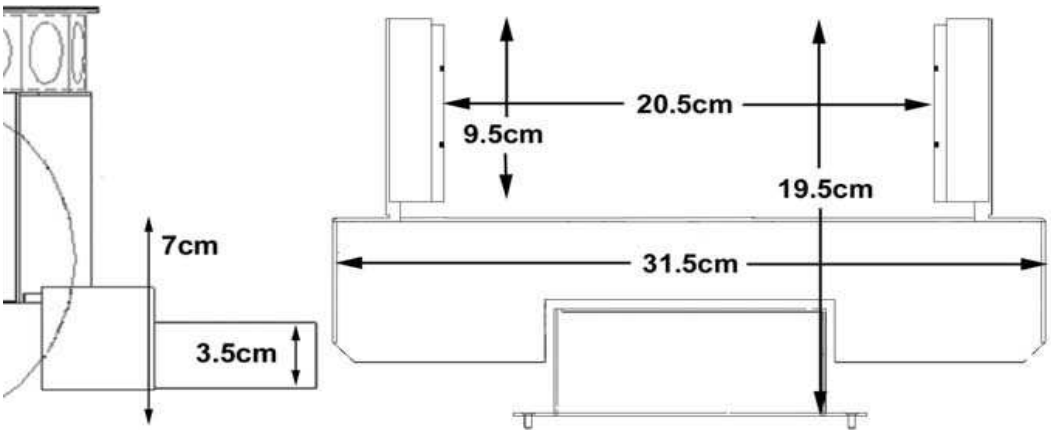

Figura. 3.11. **Dimensiones del Gripper para las plataformas P3-DX y P3-AT**

Cada una de las paletas de las pinzas contienen sensores de tipo infrarrojo que ayudan a detectar objetos cercanos dentro de su alcance; estos sensores también se encuentran manejados por las líneas de salidas digitales del microcontrolador. A pesar de que se puede operar directamente la pinza mediante las líneas de entradas y salidas digitales, también se las puede operar desde los servidores internos que se encuentran incluidos dentro del microcontrolador del robot, de tal manera que para utilizarlo con el cliente es cuestión de activar solo comandos básicos para proceder con el manejo del Gripper.

Figura. 3.12. **Aplicación del Gripper**

3.9.2. Especificaciones

Como se puede observar en la figura 3.11, el gripper posee dimensiones de 7cm de alto por 31.5cm. Ancho en su parte interna donde se encuentra el mecanismo de movimiento. Sus pinzas miden en total 3.5cm de alto por 9.5cm de largo, además la apertura máxima es de 21.5cm, además cada pinza tiene una espuma suave de 2cm de grosor en cuyas hendiduras se encuentran los sensores infrarrojos como se ve en la figura 3.12.

La presión de las pinzas en un objeto puede alcanzar valores desde 200g. hasta un valor máximo de 2Kg. El mecanismo se encuentra muy cerca el suelo en cualquiera de las plataformas móviles y su elevación alcanza los 7cm y puede levantar un peso máximo de 2Kg como se ve en la figura 3.13.

Figura. 3.13. **Gripper**

CAPÍTULO 4

SOFTWARE TIPO SERVIDOR

4.1. Descripción de ARCOS[6]

Todas las plataformas MobileRobots utilizan una arquitectura de control del robot en modo cliente – servidor, como se puede observar en la figura 4.1. En este tipo de arquitectura los servidores internos del robot funcionan para manejar todos los detalles de bajo nivel internos del robot.

Esto incluye aspectos tales como la operación de motores, el disparo del sonar, recoger y reportar la información del sonar y del codificador de las llantas y además de ello maneja todos los comandos que recibe de un programa tipo cliente como lo es ARIA.

Con la arquitectura cliente – servidor, los desarrolladores de aplicaciones robóticas no necesitan conocer muchos detalles del manejo particular del servidor del robot, ya que se aísla al cliente de este nivel bajo de control.

Sin embargo en muchos casos es necesario escribir un control del robot propio y una mayor relación con su programación, para ello se explica a continuación como comunicarse a través de la interfaz cliente – servidor llamado también Programa avanzado del Robot para Control y Operaciones (ARCOS).

Todas las funciones de ARCOS y comandos se pueden utilizar en varios entornos de programación del cliente que vienen con el robot bajo licencias separadas.

[6] Pioneer3 Operation Manuals

4.2. Protocolos de paquetes de información cliente – servidor

Las plataformas Pioneer3 DX y AT se comunican con el cliente utilizando protocolos de paquete de comunicación especiales del entorno cliente – servidor, uno de ellos sirve para paquetes del cliente al servidor y el otro para los SIPs (Paquetes de información del Servidor) desde el servidor hacia el cliente.

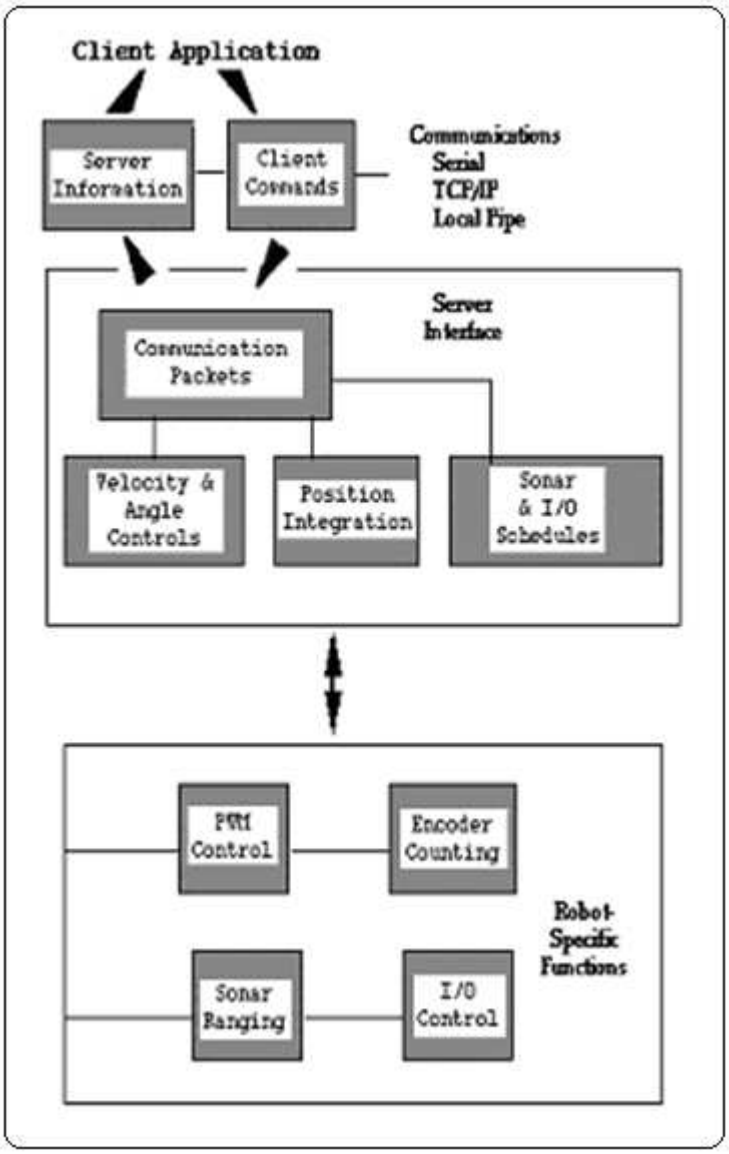

Figura. 4.1. **Arquitectura Cliente – Servidor**

Ambos protocolos son flujos de bytes que constan de cinco partes principales: una cabecera de dos bytes, un byte de la cuenta del número de paquetes de bytes posteriores, el comando del cliente o el tipo de paquete SIP, las tipos de datos de los comandos y los valores de los argumentos de los bytes de datos de los SIP, y finalmente, dos bytes para

comprobación de los paquetes. Los paquetes se encuentran limitados por un máximo de 207 bytes cada uno.

La cabecera de dos bytes cuyas señales son el inicio de un paquete son el inicio de ambos, tanto de los paquetes de los comandos del cliente como también de los SIP s: 0xFA (250) seguido por 0xFB (251). El byte que sigue es el número de bytes posteriores en todo el paquete incluyendo el de comprobación, pero no incluye el byte de cuenta así como los de cabeceras.

Los tipos de datos son simples y dependen del elemento, a continuación se puede ver la descripción: comandos del cliente, tipos del SIP, etc., son solo de 8 bits o un byte. Los argumentos y valores de los comandos pueden ser de 2 bytes, ordenados por el byte menos significativo al inicio. Algunos datos son cadenas de hasta un máximo de 200 bytes, precedidos por un byte de longitud. A diferencia de los demás datos enteros comunes, los dos bytes de comprobación aparecen con su byte más significativo al inicio.

4.3. Comprobación de paquetes

Los paquetes de comprobación del entorno cliente - servidor se calculan añadiendo sucesivamente pares de bytes (el más alto al principio) hacia la comprobación en proceso (inicialmente en cero), discriminando el signo y el sobre flujo. Si hubiese un número impar de bytes, al último byte se lo realiza una operación XOR en el último byte del chequeo.

Se nota que el entero de la comprobación es puesto al final del paquete, con sus bytes en orden inverso de los que son usados para datos, considerando que B0 es el byte más significativo y B1 es el menos significativo, tal como se muestra en la figura 4.2.

4.4. Errores de los paquetes

ARCOS ignora los paquetes de datos de los comandos del cliente cuya cuenta excede los 204 bytes (el total del tamaño de paquetes es de 207 bytes) o que tiene un error de comprobación. El cliente por tanto también debe ignorar los SIPs erróneos.

Debido a la naturaleza en tiempo real de las interacciones móviles robóticas en el cliente – servidor, se tomo la decisión de proveer una interfaz de paquete de comunicación no reconocido. Retransmitiendo la información del servidor o los paquetes de comandos que típicamente no son útiles para el propósito ya que los datos antiguos son útiles para mantener la respuesta de los comportamientos del robot.

```
AREXPORT ArTypes::Byte2 ArRobotPacket::calcCheckSum(void)
{
int i;
unsigned char n;
                                        int c = 0;
i = 3;
n = myBuf[2] - 2;
while (n > 1) {
c += ((unsigned char)myBuf[i]<<8) | (unsigned char)myBuf[i+1];
c = c & 0xffff;
n -= 2;
i += 2;
}
if (n > 0)
c = c ^ (int)((unsigned char) myBuf[i]);
return c;
}
(from MobileRobots ARIA ArRobotPacket.cpp)
```

Figura. 4.2. **Código de Comprobación de Paquetes**

No obstante, la interfaz cliente – servidor proporciona un medio sencillo para hacer frente a los paquetes de comandos ignorados: La mayoría de los comandos de los clientes alteran las variables de estado en el servidor. Mediante un examen de los valores en los respectivos SIPs, el programa cliente puede detectar comandos ignorados y redireccionarse hacia ellos hasta que se produzca su estado correcto.

4.5. Conexión Cliente – Servidor

Antes de ejercer algún tipo de control, una aplicación cliente primero debe establecer una conexión con el robot servidor, a través del puerto serial HOST interno del microcontrolador. Después de establecer una comunicación, el cliente debe enviar comandos hacia el servidor así como también recibirá la información de operación desde el mismo.

Cuando se inicia por primera vez o se resetea, ARCOS permanece en un estado especial de espera, esperando paquetes de comunicación para establecer una conexión cliente–servidor. Para establecer conexión, la aplicación cliente debe enviar una serie de tres paquetes de sincronización que contienen los comandos SYNC0, SYNC1 y SYNC2 en ese orden, y recuperar las respuestas del servidor.

Específicamente y como un ejemplo se describe a continuación el protocolo del comando del cliente, cuya secuencia de bytes de sincronización son:
SYNC0: 250, 251, 3, 0, 0, 0
SYNC1: 250, 251, 3, 1, 0, 1
SYNC2: 250, 251, 3, 2, 0, 2

En el modo espera, ARCOS realiza un eco enviando los paquetes de vuelta al cliente. El cliente debe escuchar para retornar los paquetes y debe esperar la siguiente sincronización después de que ha recibido el eco apropiado originado por el servidor.

4.6. Autoconfiguración (SYNC2)

ARCOS envía automáticamente información de identificación al cliente después del último paquete de sincronización (SYNC2). Los valores de configuración son tres cadenas de tipo NULL que comprenden el nombre, tipo y subtipo del robot guardados en FLASH. Se puede nombrar al robot de una manera particular con la herramienta de configuración que se proporciona. El tipo y subtipo son constantes puestas en la fábrica que normalmente no se los cambia.

El tipo generalmente es una cadena con el nombre Pioneer, mientras que el subtipo depende del modelo del robot: por ejemplo P3DX-SH o P3AT-SH. Los clientes usan estas cadenas de identificación para autoconfigurar sus propios parámetros. ARIA, en cambio, carga y usa los parámetros relacionados al robot que son archivos que se encuentran en el directorio especial Aria/params.

4.7. Apertura de los servidores –OPEN-

Una vez que se ha establecido la comunicación con ARCOS, el cliente debe enviar el comando de apertura OPEN comand0 #1 (250, 251, 3, 1, 0, 1) al servidor, el cual ocasiona que el microcontrolador realice alguna operaciones de “limpieza”, empieza sus varios servidores, como por ejemplo los sonares y motores, y empieza a transmitir la información del servidor al cliente.

En la primera conexión, los motores del robot se encuentran deshabilitados, independientemente del estado de la última conexión. Para habilitar los motores después de establecer una conexión, se lo debe hacer manualmente (presionando el botón blanco de los motores ubicado en el panel de control), desde el LCD se puede habilitar de forma interactiva, o mediante el cliente enviando un ENABLE (comando #4) con un argumento entero de 1. Posteriormente se verá los comandos de los clientes.

Una vez conectado, enviar el al comando #4 (ENABLE) o pulsar el botón blanco ubicado en el panel de control para habilitar los controles.

4.8. Paquetes de información de los servidores.

Una vez realizados los pasos previos, ARCOS automáticamente y repetidamente envía paquetes de información a través del puerto serial del HOST hacia el cliente conectado.

Los SIPs estándar de ARCOS informan al cliente acerca de un número de estados de operación y lecturas, usando el orden y tipos de datos que se describen en la siguiente tabla. ARCOS también soporta varios tipos adicionales de SIPs. Para más detalles se muestra la tabla 4.1.

4.9. Mantener el pulso –PULSE-

ARCOS posee por razones de seguridad un Watchdog del que se espera que una vez conectado, el robot reciba al menos un paquete de comandos del cliente por cada ciclo del Watchdog, como se define en la FLASH del robot (por defecto es de dos segundos). De otra manera, se asume que la conexión cliente – servidor se ha perdido y el robot se para.

Algunos clientes basados en ARIA, a manera de una buena práctica envían el comando #0 PULSE (250, 251, 3, 0, 0, 0) después del comando OPEN, esto para que el robot sepa que aún se mantiene una operación.

ETIQUETA	DATA	DESCRIPCION
HEADER	2 BYTES	Exactamente en orden 0xFA (250), 0xFB (251)
BYTE COUNT	BYTE	Número de bytes +2 (de comprobación), no se incluye la cabecera o los bytes de cuenta.
TYPE	0X3S	Estado de los motores; s=2 cuando se encuentran parados o 3 cuando se el robot está en movimiento.
XPOS	INT	Codificador de las llantas y giroscopio opcional integrado dispuesto en milímetros (DistConvFactor ≠ = 1.0).
YPOS	INT	
THPOS	INT	Orientación en unidades angulares (AngleConvFactor ≠ = 0.001534 radianes por unidad angular = 2π/4096).
L VEL	INT	Velocidades de las llantas en milímetros por segundo (VelConvFactor ≠ = 1.0).
R VEL	INT	
BATTERY	BYTE	Carga de la batería en décimas de voltios (Por ejemplo 101 = 10.1 voltios)
STALL AND BUMPERS	UINT	Indicadores de paro de motores y de parachoques. El bit 0 es el indicador de paro de la llanta izquierda, se pone en 1 si se encuentra parada. Los bits 1 – 7 corresponden al primer parachoques, a los estados de las E/S digitales. El Bit 8 corresponde al indicador de para de la llanta derecha y los bits del 9 – 15 corresponden a los estados de las E/S del parachoques posterior, dependiendo si se encuentra conectado.
CONTROL	INT	Es el punto deseado de la posición angular del servo manejado por el servidor y es en grados.
FLAGS	UINT	El Bit 0 muestra el estado de los motores, los bits 1 – 4 corresponden al estado del arreglo de los sonares, los bits 5 y 6 son de STOP, los bits 7 y 8 son para los sensores infrarrojos o los láseres, el bit 9 es del botón de accionamiento del Joystick, el bit 10 muestra el estado del accesorio de carga automática.
COMPASS	BYTE	Corresponde a un accesorio de brújula electrónica que mide 2 grados por unidad.
SONAR COUNT	BYTE	Número de lecturas nuevas del sonar incluidas en las SIPs
NUMBER	BYTE	Si la cuenta del sonar > 0, su número discos será de 0 – 31; correspondientes a las lecturas siguientes.
RANGE	UINT	Valor del rango corregido del sonar en milímetros (RangeConvFactor ≠ = 1.0).
…Resto de las lecturas del sonar…		
GRIP STATE	BYTE	Byte del estado del Gripper
ANPORT	BYTE	Puerto análogo seleccionado 1 – 5.
ANALOG	BYTE	Lectura de la entrada análoga seleccionada (0 – 255 = 0 – 5 VDC).
DIGIN	BYTE	Byte codificado de la entrada digital.
DIGOUT	BYTE	Byte codificado de la salida digital.
BATTERYX10	INT	Voltaje actual de la batería en 0.1 voltios (útil especialmente para voltajes de batería mayores a 25.5)
CHARGESTATE	BYTE	Para la versión 1.5 y posterior de ARCOS. El estado de la recarga automática se muestra con -1 = desconocido, 0 = no se está cargando, 1 = fuerte, 2 = sobrecarga, 3 = flotante (terminado de cargar).
ROTVEL	INT	Velocidad actual de rotación en grados x 10 por segundo.
FAULTFLAGS	INT	No se usa.
CHECKSUM	INT	Integridad de paquetes de verificación.

Tabla. 4.1. **Contenidos SIP estándar**

Otro caso sería cuando una aplicación se deje pausada por algún tiempo o si el robots no realiza acción alguna, se recomienda enviar periódicamente el comando PULSE para hacerle saber al servidor que el cliente aún lo necesita y que se encuentra bien. De ahí PULSE no tiene algún otro propósito.

Si el robot deja de funcionar debido a un problema en la comunicación con el cliente, el puede “revivir” una vez que reciba un comando del cliente y automáticamente acelera a la última velocidad especificada y el punto en el cual se encontraba la parte delantera (cabecera).

4.10. Cierre de conexión –CLOSE-

Para cerrar la conexión cliente – servidor, que automáticamente inhabilita los motores y otras funciones como el sonar, simplemente se envía el comando #2 CLOSE (250, 251, 3, 2, 0, 2). La mayoría de parámetros de operación de ARCOS retornan al valor por defecto ubicado en la FLASH del robot una vez que se ha desconectado el cliente.

4.11. Comandos del Cliente

ARCOS tiene un formato estructurado de comandos para recibir y responder a las direcciones desde un cliente para controlar y operar el robot. Los comandos del cliente se componen de un número de bytes de mando, seguidos si se requiere por el comando, por un byte de descripción del tipo de argumento y luego su valor, tal como se observa en la tabla 4.2.

El número de comandos del cliente que se pueden enviar por segundo dependen de la tasa de trasmisión del HOST, del promedio, el número de bytes por comando, la sincronización de la comunicación, etc.

El procesador de comandos de ARCOS corre en un ciclo de un milisegundo por interrupción, pero la velocidad de respuesta del servidor depende del comando. Normalmente, se limita el comando del cliente a un máximo de uno por cada 3 a 5 milisegundos, o se prepara en ese tiempo para recuperarse de comandos perdidos.

COMANDO	#	ARGS	DESCRIPCION	VERSION
			Antes de la Conexión con el Cliente	
SYNC0	0	ninguno	Empieza la conexión. Envía en secuencia. ARCOS hace eco de los comandos enviados hacia el cliente, y el robot especifica una auto-sincronización después de SYNC2.	1.0
SYNC1	1	ninguno		
SYNC2	2	ninguno		
			Después de Establecer Conexión	
PULSE	0	ninguno	Resetea el Watchdog del Servidor.	1.0
OPEN	1	ninguno	Inicia los Servidores.	1.0
CLOSE	2	ninguno	Cierra los Servidores y la Conexión con el Cliente.	1.0
POLLING	3	Str	Cambia la secuencia de muestreo del sonar.	1.0
ENABLE	4	Int	1 = Habilita; 0 =Deshabilita los motores	1.0
SETA	5	Int	Establece la aceleración de traslación, si es positiva, o la desaceleración, si es negativa; en mm/seg2.	1.0
SETV	6	Int	Establece la Velocidad de Traslación Máxima o de movimiento; mm/seg.	1.0
SETO	7	ninguno	Resetea la posición actual al origen 0, 0, 0.	1.0
MOVE	8	Int	Se mueve una distancia hacia delante (+) o hacia atrás (-) en mm, a la velocidad de SETV.	1.0
ROTATE	9	Int	Rota en sentido horario (+) o anti horario (-) en grados por segundo a la velocidad establecida en SETRV.	1.0
SETRV	10	Int	Establece la velocidad de rotación máxima o de giro; en grados/seg.	1.0
VEL	11	Int	Se traslada en mm/seg hacia delante (+) o hacia atrás (-) (limitada por la velocidad SETV).	1.0
HEAD	12	Int	Gira la cabecera del robot a la velocidad absoluta de SETRV; ± grados (+ anti horario).	1.0
DHEAD	13	Int	Gira la cabecera del robot a una velocidad relativa a la posición en sentido anti horario (+) horario (-) en grados.	1.0
SAY	15	Str	Toca al menos 20 segundos, de tonos pares mediante el pizo eléctrico ubicado en el panel de Usuario.	1.0
JOYREQUEST	17	Int	Envía una o varias órdenes continuas (>1) o las para (0) a los SIPs del Joystick.	1.3
CONFIG	18	ninguno	Requiere un SIP de configuración.	1.0
ENCODER	19	Int	Envía una o varias órdenes continuas (>1) o las para (0) a los SIPs del Codificador.	1.0
RVEL	21	Int	Rota el robot en sentido anti horario (+) horario (-) grados/seg. (llamados a SETRV).	1.0
DCHEAD	22	Int	Ajusta la cabecera del robot relacionándola con el último punto establecido; ± grados (+ = anti horario).	1.0
SETRA	23	Int	Cambia la desaceleración (-) o aceleración (+) de rotación, en grados/seg2.	1.0
SONAR	28	Int	1 =habilita, 0 = deshabilita todos los sonares; de otra manera usa los bits 1-3 para especificar un número de arreglo individual 1-4.	1.0
STOP	29	ninguno	Para el robot. Los motores se mantienen habilitados.	1.0
DIGOUT	30	2 bytes	Establece (1) o resetea (0) los puertos de salida del usuario. Los Bits 8-15 son un byte de máscara que selecciona si se pone (1), cambia el estado de los puertos de salida; Bits 0-7 Establecen (1) o resetea (0) los puertos	1.0

			seleccionados.	
VEL2	32	2 bytes	Establece velocidades independientes en las llantas; los Bits 0-7 para la llanta derecha, los Bits 8-15 para la llanta izquierda en incrementos de 20 mm/seg.	1.1
GRIPPER	33	Int	Comandos de los servidores del Gripper. Mirar la programación en ARIA del Gripper para más detalles.	1.0
ADSEL	35	Int	Selecciona el número de puerto A/D para reportar un valor del ANPORT en un SIP estándar.	1.0
GRIPPERVAL	36	Int	Valores del Servidor del Gripper	1.0
GRIPREQUEST	37	Int	Envía una o varias órdenes continuas (>1) o las para (0) a los SIPs del Gripper.	1.0
GYROCALCW	38	Uint	Establece el valor de calibración en sentido horario del giroscopio.	2.0
GYROCALCCW	39	Uint	Establece el valor de calibración en sentido anti horario del giroscopio.	2.0
IOREQUEST	40	Int	Envía una o varias órdenes continuas (>1) o las para (0) a los SIPs de las E/S.	1.0
TTY2	42	Str	Envía una cadena de argumentos al dispositivo serial conectado al puerto serial AUX1.	1.0
GETAUX	43	Uint	Solicita recuperar de 1 – 200 bytes del puerto serial AUX1; 0 vuelca el buffer.	1.0
BUMPSTALL	44	Int	Para el robot según el accionamiento de los parachoques si: no (0), solo el frente (1) mientras se mueve hacia delante, solo atrás (2) si se mueve de reversa, o ambos (3) cuando se activan los parachoques cuando el robot se mueve en una dirección específica.	1.0
TCM2	45	Int	Comandos del módulo TCM2; 0 = módulo apagado, sin lecturas; 1 = solo la brújula, lecturas en SIPs estándar; 2 = envía un paquete TCM2; 3 = envía paquetes continuos TCM2 (cada ciclo); 4 = calibración de usuario; 5 = auto calibración; 6 = para la auto calibración, envía un paquete, y regresa al modo 1; 7 = resetea el software.	1.0
JOYDRIVE	47	Int	1 = permite el manejo con el Joystick desde el puerto mientras está conectado con el cliente; 0 lo deshabilita.	1.0
SONARCYCLE	48	Uint	Cambia el tiempo del ciclo del sonar, en milisegundos.	1.0
HOSTBAUD	50	Int	Cambia la tasa de baud del puerto HOST serial a 0 = 9600, 1 = 19200, 2 = 38400, 3 = 57600 o 4 = 115200.	1.0
AUX1BAUD	51	Int	Cambia la tasa de baud del puerto AUX1 serial (de manera igual al HOSTBAUD).	1.0
AUX2BAUD	52	Int	Cambia la tasa de baud del puerto AUX2 serial (de manera igual al HOSTBAUD).	1.0
AUX3BAUD	53	Int	Cambia la tasa de baud del puerto AUX3 serial (de manera igual al HOSTBAUD).	1.0
E_STOP	55	ninguno	Parada de emergencia; produce una desaceleración abrupta.	1.0
M_STALL	56	Int	Argumento 1 = El botón de los motores causo una parada.	1.0
GYROREQUEST	58	Int	Si el Cliente (HasGyro 1), requiere una o varias órdenes continuas (>1) o si Para (0) los SIPs del Giroscopio. Si el Cliente (HasGyro	1.0

			2), 0 lo deshabilita y 1 habilita el giroscopio.	
LCDWRITE	59	Str	Muestra un mensaje en el display del accesorio LCD: byte 0 = la columna de inicio (1-19); byte 1 = fila de inicio (1-4); byte 2 = 1 el contenido de la primera línea se encuentra limpio, de otro modo 0; byte 3 = 1 si hay más de 20 caracteres, NULL si está terminado.	1.3
TTY4	60	Str	Envía una cadena de argumentos hacia un dispositivo conectado al puerto serial AUX3.	1.0
GETAUX3	61	Int	Solicita recuperar de 1 – 200 bytes del dispositivo conectado puerto serial AUX3; 0 vuelca el buffer.1.0	1.0
TTY3	66	Str	Envía una cadena de argumentos hacia un dispositivo conectado al puerto serial AUX2.	1.0
GETAUX2	67	Int	Solicita recuperar de 1 – 200 bytes del dispositivo conectado puerto serial AUX2; 0 vuelca el buffer.	1.0
CHARGE	68	Int	0 = despliega, 1 = libera el dispositivo de carga automática.	1.0
ARM	70-80	Int	Comandos relacionados con el accesorio del brazo. Para más detalles consultar el manual del accesorio.	1.1
ROTKP	82	Int	Cambia el valor Proporcional del parámetro PID de rotación.	1.0
ROTKV	83	Int	Cambia el valor Derivativo del parámetro PID de rotación.	1.0
ROTKI	84	Int	Cambia el valor Integral del parámetro PID de rotación.	1.0
TRANSKP	85	Int	Cambia el valor Proporcional del parámetro PID de traslación.	1.0
TRANSKV	86	Int	Cambia el valor Derivativo del parámetro PID de traslación.	1.0
TRANSKI	87	Int	Cambia el valor Integral del parámetro PID de traslación.	1.0
REVCOUNT	88	Int	Cambia la cuenta diferencial del codificador	1.1
DRIFTFACTOR	89	Int	Cambia el factor de deslizamiento	1.0
SOUNDTOG	92	Int	0 = silencio, 1 = habilita el Pizo eléctrico del panel de control	1.0
TICKSMM	93	Int	Cambia el número de orificios a considerarse en el codificador en cada vuelta de la llanta. En mm.	1.1
BATTEST	250	Int	Se puede establecer artificialmente el valor del voltaje de la batería en décimas de voltios (100 = 10V); 0 para regresar al voltaje real.	1.0
RESET	253	ninguno	Hace que ocurra el reseteo del microcontrolador.	1.0
MAINTENANCE	255	ninguno	Comienza el modo mantenimiento en el microcontrolador (ARSHstub).	1.0

Tabla. 4.2. **Comandos de ARCOS para el manejo desde el cliente**

4.12. Robot en movimiento

Los servidores de control de motores de ARCOS, aceptan diferentes comandos de movimiento que son de dos tipos: el de movimiento independiente de llantas (VEL2) o los

de movimientos de traslación y rotación (VEL, ROT, etc.). En realidad, los comandos VEL2 son recompuestos en el servidor dentro de las componentes de traslación y rotación y sus límites correspondientes así como sus (des)aceleraciones aplicadas, se pude observar estos comandos según la tabla 4.3.

Los servidores de ARCOS automáticamente abandonan cualquier punto establecido de traslación o rotación y cambian a valores independientes de rotación/traslación cuando el cliente requiere de uno de estos tipos de comandos.

ROTACION	
HEAD (#12)	Gira la cabecera del robot a la velocidad absoluta de SETRV
DHEAD (#13) DCHEAD (#22)	Gira la cabecera relativamente hacia un punto de control a la velocidad SETRV máxima.
ROTATE (#9)	Gira a la velocidad SETRV.
RVEL (#21)	Rota en sentido anti horario (+) horario (-) grados/seg. limados a SETRV
TRASLACION	
VEL (#11)	Traslada adelanta/reversa a la velocidad SETV descrita
MOVE (#8)	Mueve una distancia hacia adelante (+5000mm máximo) o hacia atrás (-4999mm máximo) a la velocidad SETV
LLANTAS INDEPENDIENTES	
VEL2 (#32)	Establece la velocidad para cada lado del robot.

Tabla. 4.3. **Comandos del cliente relacionados al movimiento**

Se debe notar, que una vez conectado, los motores del robot se encuentran deshabilitados, independientemente del estado que tenían al conectarse la última vez. De acuerdo a esto, se debe habilitar los motores manualmente (el botón blanco de motores ubicados en el panel de control) o mediante el envío por parte del cliente de ENABLE, es decir el comando #4 con los valores respectivos. Se puede monitorear el estado de los motores con el Bit 0 del Entero Flags en los SIPs estándar.

4.13. Comandos del movimiento del cliente

Cuando ARCOS recibe un comando de movimiento, acelera o desacelera el robot con parámetros de traslación SETA (comando #5) y rotación SETRA (comando #23) hasta que alcance su velocidad SETV (comando #6) de traslación máxima y SETRV (comando #10) de rotación máxima, o los equivalente del VEL2, o esté cerca de su meta. De acuerdo a esto, los movimientos de los puntos establecidos se resumen en una funciona trapezoidal, que ARCOS recalcula cada vez que recibe un nuevo comando de movimiento.

ARCOS automáticamente limita las velocidades especificadas VEL2, VEL y RVEL a sus valores previamente dispuestos, que son modificables a través del cliente SETVEL y SETRV y los toma como constantes dentro de la FLASH de los cuales la plataforma robótica dependerá. De manera similar, los parámetros distintos de aceleraciones y desaceleraciones son limitados por estas constantes almacenadas en la FLASH.

ARCOS inicia estos valores de los parámetros FLASH relacionados con el inicio o reseteo del microcontrolador, y con la primera conexión con un cliente. Los límites de velocidad, ya sean de la FLASH o cuando son cambiados por SETV o SETRV, toman efecto luego, y no en la traslación o rotación actual. Los valores máximos regresan a sus valores FLASH por defecto cuando se desconecta del cliente.

El comando de traslación MOVE, comando #8, envía la plataforma hacia adelante a una distancia especificada en milímetros si el argumento es positivo, o hacia atrás si es un argumento negativo. Cada movimiento está limitado por +5000 milímetros y -4999 milímetros respectivamente.

Se puede enviar un comando de orientación HEAD (#12), DHEAD (#13) o DCHEAD (#22) con un argumento o valor en grados para dirigir el robot con respecto a su ángulo de partida interno hacia un puto específico (±0-180 grados), o inmediatamente hacia un punto relativo, o también relativo a su punto actual (conseguidos con los últimos comandos enviados), respectivamente. En general, los argumentos positivos de los comandos para dirección de posiciones relativas giran el robot en una dirección anti horaria.

Sin embargo, el robot para llegar a su punto gira en la dirección en la cual pueda alcanzar el punto de manera más eficiente. De acuerdo a esto, si se ubica parámetros mayores a 180 grados, el robot automáticamente los reducirá a 180 o menos grados con un cambio importante en la dirección de rotación para alcanzar de manera más eficientemente el punto pedido.

El comando #55 E_STOP o el botón de parada de emergencia (Pioneer3 AT) anula la desaceleración y ocasiona una parada abrupta del robot en la distancia y tiempo más cortos

posibles. De esta manera, el robot frena y baja a velocidades de cero tanto en traslación como en rotación con una desaceleración muy alta y se mantiene parado hasta que reciba un comando posterior en el que se establezca un nuevo movimiento en el cliente ya sea de traslación o rotación, o simplemente hasta que el botón de STOP sea liberado.

4.14. Controles PID

Los servidores de manejo de ARCOS utilizan un sistema común PID (Proporcional-Integral-Derivativo) con una realimentación del codificador para ajustar con una señal PWM (Modulación de Ancho de Pulso) el manejo y potencia de los motores. El ciclo del motor es de 50 microsegundos (20KHz); el ancho de pulso es proporcional a este ciclo del motor de 0 – 500 para 0 a 100%. Los servidores de ARCOS recalculan y ajustan la trayectoria y velocidades cada 5 milisegundos.

Los valores PID para traslación y rotación y el PWM máximo son configurables dentro de los parámetros FLASH del microcontrolador. También se puede actualizar temporalmente los valores PID con los comandos #84 al #87 del cliente de ARCOS Los cambios hechos en la marcha permanecen hasta que el cliente se desconecte. Los valores PID de traslación también se aplican para el modo de velocidades independientes en cada llanta.

El término P o valor Kp amplifica la ganancia del sistema amplificando el error de posición. Ganancias altas pueden tener una tendencia a disparar las velocidades establecidas; pequeñas ganancias pueden limitar el disparo pero causan que el sistema sea lento. Varias pruebas han demostrado que un robot totalmente cargado funciona mejor con un Kp establecido entre 15 a 30, mientras que un robot sin carga puede funcionar mejor con un Kp en el rango de 20 a 50.

El término D provee un factor de ganancia PID que es proporcional a la velocidad de salida. Tiene un mejor efecto en la amortiguación y en minimizar las oscilaciones del sistema de manejo. Este término usualmente es el primero que debe ajustarse si se encuentra una respuesta no satisfactoria del sistema. Normalmente, se ha encontrado que el Kv trabaja mejor en el rango de 10 a 30 para robots ligeramente o totalmente cargados respectivamente. Si el robot comenzara a vibrar o a interrumpirse, se debe reducir el Kv.

El término I correspondiente al Ki modera cualquier error de estado estacionario lo que limita las fluctuaciones de las velocidades durante el curso de movimiento. El robot rebajará a cero cualquier error de posición ocasionado por los comandos. Un factor de Ki demasiado alto puede causar una oscilación del motor cuando la carga cambia, como cuando sube una pendiente o acelera a una nueva velocidad. En consecuencia, se usa un valor del Ki normalmente ubicado en el rango de 0 a 10 para plataformas ligeras o pesadas respectivamente.

4.15. Integración de Posición

Las plataformas MobileRobots siguen su posición y orientación basadas en la estimación del movimiento de las llantas y de las lecturas de sus codificadores, además de las lecturas del giroscopio cuando este se encuentra colocado y activado. El robot basado en ARCOS mantiene sus coordenadas internas de posición en unidades representado en la figura 4.3, el cual maneja la plataforma dependientemente, pero al reportar estos valores los saca en milímetros que son valores de la plataforma independientes y en unidades angulares ($2\pi/4096$) en los SIP estándar (Xpos, Ypos, y Thpos).

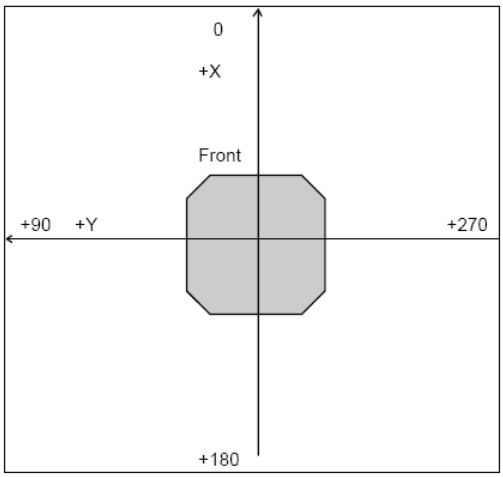

Figura. 4.3. **Sistema de coordenadas interno**

Se debe tener en cuenta que el registro entre coordenadas internas y externas se deterioran rápidamente con el movimiento debido al juego de la transmisión, desviación y deslizamiento de las llantas y muchos otros factores del mundo real. Se puede confiar en la habilidad de estimación del robot para un rango corto, en el orden de unos pocos metros y

una o dos revoluciones, dependiendo de la superficie. Las alfombras tienen a ser peores que los suelos duros.

Además, el movimiento demasiado rápido o demasiado lento tiende a exacerbar, los errores de posición absoluta. Por lo cual se considera la capacidad de estimación del robot como medio de vinculación para las lecturas de los sensores que se toman cada corto tiempo, pero no como un método para mantener al robot en curso respecto al mapa global.

En el inicio, el robot se encuentra en el origen (0, 0, 0) apuntando a lo largo del eje X positivo en 0 grados. Los valores absolutos varían entre 0 y ± 4096 en unidades angulares desde +180 hasta -179 grados. Se puede resetear las coordenadas internas a 0, 0, 0 con el comando #7 SETO.

4.16. DriftFactor, RevCount y TicksMM

Son tres comandos del cliente que permiten cambiar, aunque sea momentáneamente para la conexión actual del cliente – servidor, ciertos valores que afectan la traslación, rotación y desviación del robot. El TickMM es el número de pasos del codificador por milímetro para la rotación de la llanta, velocidad de traslación y cálculos de distancia. El valor por defecto de la FLASH puede ser cambiado sobre la marcha mientras dura la sesión de conexión con el cliente con el comando #93 del cliente TICKSMM de valor entero sin signo (uint).

El DriftFactor es un valor con signo que se incrementa a razón de 1/8192 que se suma o se resta del codificador de la llanta izquierda para corregir las diferencias de radio de giro y consecuentemente el deslizamiento producido por la rotación y traslación. El valor del DriftFactor se sitúa por defecto al inicio y al resetear el robot, y puede ser cambiado sobre la marcha con el comando #89 DRIFTFACTOR que es de tipo entero sin signo.

El parámetro RevCount es el número diferencial de los pasos del codificador para un giro de 180 grados del robot y se usa para calcular y ejecutar giros. Como el TicksMM y el DrifyFactor, el RevCount tiene sus parámetros por defecto en la FLASH al inicio o al

resetear el robot, y puede ser cambiado sobre la marcha con el comando #88 REVCOUNT del cliente que es de tipo entero sin signo.

4.17. Sonar

Cuando se conecta con el cliente y se abre mediante él, ARCOS automáticamente empieza a disparar los sonares, un disco a la vez simultáneamente para cada uno de los cuatro arreglos máximos, como se haya secuenciado y habilitado en los parámetros FLASH del robot. Los servidores del sonar también empiezan a enviar los resultados de las lecturas del rango al cliente mediante los SIPs estándar. Se puede observar los discos del sonar en la figura 4.4.

4.17.1. Habilitar/ Deshabilitar el Sonar

Para habilitar o deshabilitar uno o todos los arreglos de sonares se utiliza el comando # 28 del cliente SONAR. Se establece el bit 0 del comando SONAR en 1 para habilitar o en 0 para deshabilitar los arreglos de sonares. Se establece los argumentos de los bits 1-3 para un número de arreglo individual así se establece los valores del 1 al 4 para escoger el arreglo, si se escoge el número cero afecta a todos los arreglos a la vez.

Por ejemplo, un valor de uno habilita todos los arreglos, mientras que un valor de 6 apaga el arreglo número 3. Se puede monitorear el estado de los arreglos en la FLASH observando los flags los bits 1-4 de los SIPs estándar. El respectivo bit se establece si el arreglo esta activado.

4.17.2. Secuencia de consulta (POLLING)

Cada arreglo se dispara a una tasa de muestreo y una secuencia definida en los parámetros de la FLASH interna del microcontrolador del robot. Para cambiar la secuencia de disparo se usa el comando #3 POLLING del cliente y el comando #48 SONAR_CYCLE para cambiar la tasa de muestreo. Los cambios se mantienen hasta que se reinicie la conexión cliente – servidor.

La cadena de valores del comando POLLING consisten en una secuencia de números de sonares del 1 al 32. Los números del sonar del 1 al 8 se añaden en la secuencia correspondiente al primer arreglo; los números del 9 al 16 son para la secuencia del arreglo número 2; del 17 al 24 son para la secuencia del arreglo número 3 y finalmente del 25 al 32 para el cuarto arreglo.

Se puede incluir sobre los 16 números de sonares en la secuencia para cualquier arreglo simple. Solo los números de los sonares de los arreglos cuyos números aparezcan en el argumento van a ser secuenciados de nuevo.

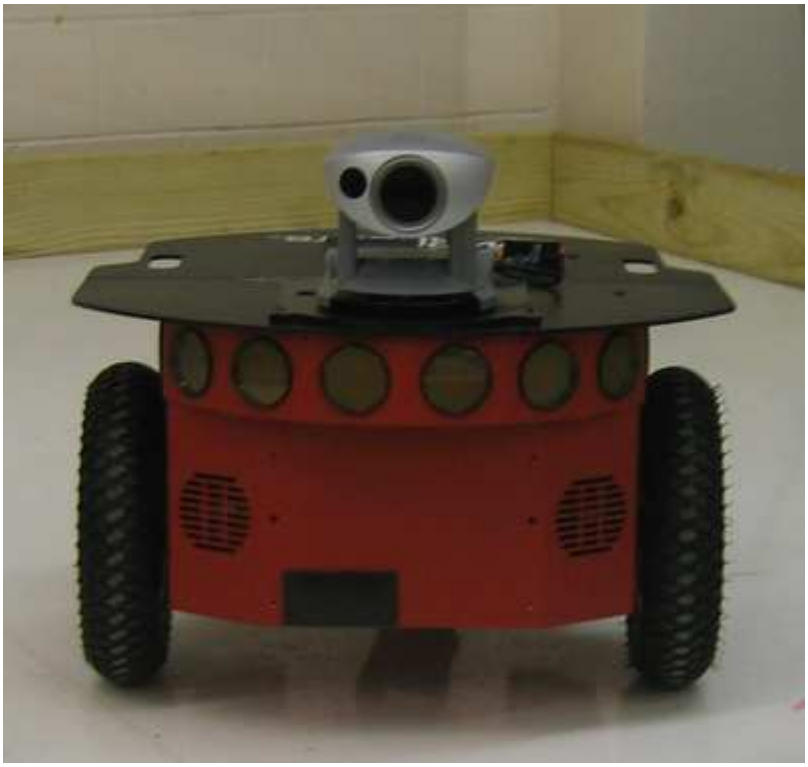

Figura. 4.4. **Discos sensores tipo Sonar**

Se puede repetir un número de sonar 2 o más veces en una secuencia. Si el número del sonar no aparece en la secuencia alterada, el disco no se disparará. Si se repite un número de sonar en la secuencia, se debe saber que el ARIA y sus demás clientes ignoran la primera lectura si los datos del rango del sonar aparecen en la misma SIP.

Por razones de compatibilidad con otros sistemas operativos existentes para el robot, si la cadena de argumentos se encuentra vacía, todos los sonares en los arreglos se deshabilitan, pero su secuencia de muestreo se mantiene inalterable, es como si se enviara al comando SONAR un valor de argumento de cero.

4.17.3. Tasa de consulta (POLLING)

Para las versiones recientes del microcontrolador, la tasa de muestreo del sonar ha sido ajustada para cada arreglo: cada disco del sonar es muestreado cada 40 milisegundos. De manera que, para un arreglo de ocho discos con una secuencia de muestreo (12345678 por defecto), cada uno de los discos será muestreado cada 320 milisegundos.

Este ciclo común de tiempo se acomoda para el rango máximo del sonar de 5 metros, usado para aplicaciones generales, incluyendo actividades de reconocimiento y localización. Para otras aplicaciones, como por ejemplo la evasión de obstáculos cercanos, una tasa de muestro más corta pero más rápida es mejor.

Para cambiar el ciclo del sonar sobre la marcha se usa el comando #48 SONARCYCLE, cuyo argumento es de tipo entero y se especifica en milisegundos. Los valores mínimo y máximo son de 2 y 120 milisegundos respectivamente. El valor por defecto especificado en la FLASH, normalmente está en el rango de 40 milisegundos.

4.17.4. Lecturas del rango del sonar

Solo las lecturas del sonar que son tomadas desde el último SIP estándar, no se toman en cuenta después de eso. Las lecturas aparecen en el SIP estándar en tres bytes: el primer byte del sonar es su referencia, y va numerado desde el sonar #0 al #31 (uno menos que los referenciados en el comando del cliente); los siguientes dos bytes son el entero más pequeño que representa las lecturas del rango en milímetros.

ARCOS reportará una lectura máxima de 5000 cuando no existe un eco de retorno o si el obstáculo se encuentra más cerca de los 120 milímetros.

4.18. Paradas y emergencias

Una plataforma equipada con parachoques frontales y/o traseros, puede ser detenida si ARCOS detecta que una parada ha sido activada debido al accionamiento de uno de los parachoques y lo notifica al cliente que detiene al robot inmediatamente.

Para deshabilitar el comportamiento de Parada debido al accionamiento de uno de los parachoques se puede utilizar el comando #44 BUMPSTALL cuyo argumento puesto a 0 desactivará este comportamiento, y si se quiere volver a activar se puede utilizar cualquiera de las siguientes opciones: se pone 1 si se quiere que funcione solamente al accionarse el parachoques delantero, se pone 2 si solo funciona el parachoques trasero, y finalmente el argumento 3 si se quiere que se activen las paradas al activarse cualquiera de los dos parachoques delantero o trasero.

En un caso de emergencia, el cliente puede parar el robot rápidamente, ya que no está sujeta a una desaceleración normal. Para ese caso se puede utilizar el comando #55 E_STOP.

Otra acción que puede simular una para al igual como lo hace el comando BUMPSTALL es el comando E-STOP, pero esta vez lo hace a través de un botón de parada STOP. El interruptor de STOP se comunica a través de un puerto digital de E/S del microcontrolador, el cual notifica a ARCOS de la condición existente. ARCOS para los motores del robot, frenándolos y enviando varias paradas continuas.

Diferente de otro tipo de paradas, la parada de emergencia también deshabilita los motores, cosa que no hacen las demás. Luego de ello se debe habilitar los motores manualmente (Botón blando de MOTORES) o mediante programación (comando #4 ENABLE).

Los servidores de la Parada de emergencia notifican al programa cliente de su condición a través de los bytes de stall (parada) y en el Bit 5 del Byte de las FLAGS estándar, para que el cliente pueda responder de manera diferente ante una Parada de Emergencia que ante una parada normal.

Normalmente la parada de emergencia se encuentra habilitada y se puede cambiar su estado utilizando el comando #56 E_STALL de ARCOS. Enviando el argumento 0 se deshabilita la parada de emergencia E_STALL, mientras que un argumento de 1 lo habilita, estas condiciones se pueden observar en la siguiente tabla 4.4.

BIT	CONDICION SI SE ACTIVA
0	Motores habilitados
1	Arreglo del sonar #1 habilitado
2	Arreglo del sonar #2 habilitado
3	Arreglo del sonar #3 habilitado
4	Arreglo del sonar #4 habilitado
5	Botón de STOP presionado
6	E_STALL activada
7	Pared lejana detectada (IR)
8	Pared cercana detectada (IR)
9	Botón 1 presionado del Joystick
10	Potencia de carga “óptima”
11-15	Reservados

Tabla. 4.4. **Bits de Banderas Estándar (FLAGS)**

4.19. Paquetes y comandos de los accesorios

Varios paquetes de información (SIPs) del robot de ARCOS están diseñados para trabajar directamente con los accesorios.

Cuando un cliente pide un comando relacionado con ARCOS, se envía una o varias transmisiones continuas de los paquetes de información de ARCOS hacia el cliente a través de línea de comunicación serial.

Los paquetes prolongados son enviados inmediatamente antes (como el GYROpac o JOYSTICKpac) o después (como el IOpac) de los SIPs estándar que ARCOS envía al cliente cada cierto ciclo SypCycle, en milisegundos.

Los SIPs estándar tienen prioridad pero sin embargo se debe procurar ajustar la tasa de transmisión en baud del HOST, para que se puedan acomodar todos los paquetes en el ciclo correspondiente, o de otra manera algunos paquetes no se enviarán nunca.

4.19.1. Procesamiento de paquetes

Al igual que con los SIP estándar, todos los paquetes de información de ARCOS, tiene una cabecera de encapsulamiento (0xFA, 0xFB; 250, 251), byte de cuenta, byte de tipo de paquete y los de comprobación. Corresponde al cliente analizar los paquetes, ordenados por el tipo de contenido. Para ello se verá posteriormente la programación de los clientes.

4.19.2. Comando CONFIG y el CONFIGpac

Al enviar el comando #18 sin argumentos el ARCOS envía de vuelta un paquete de 32 bits (0x20), el CONFIGpac SIP, que contiene los parámetros de operación del robot. Se puede utilizar el CONFIGpac para examinar varios valores por defecto utilizados en la FLASH y para el funcionamiento, que son apropiados, y se pueden cambiar por otros comandos de clientes como el SETV y el ROTKV según la tabla 4.5.

4.20. SERIAL

Las tasas de baudios para los puertos seriales AUX y HOST inicialmente son establecidas por los valores almacenados en la FLASH por defecto y vuelve a estos valores siempre que se resetee el microcontrolador o se una vez desconectado del cliente.

Para un manejo avanzado de los puertos seriales desde el lado del cliente, ARCOS provee cuatro comandos que permitirán establecer las tasas de transmisión de los puertos seriales HOST (HOSTBAUD #50), Aux1 (AUX1BAUD #51), Aux2 (AUXBAUD #52) y Aux3 (AUXBAUD #53). Se puede utilizar los argumentos enteros 0 = 9600, 1 = 19.2k, 2 = 38.8k, 3 = 57.6k o 4 = 115.2k Baudios para las tasas de transmisión respectivas.

Con la opción de auto-baudios, el puerto serial del HOST automáticamente regresa a su valor por defecto establecido en la FLASH, si después de ser reseteado por el comando HOSTBAUD no recibe un paquete válido o comando dentro de los siguientes 500 milisegundos.

ETIQUETA	DATO	DESCRIPCION
Header	Int	Paquete común de cabecera = 0XFAFB
Byte Count	byte	Número de bytes de datos siguientes
Packet Type	byte	ENCODERpac = 0x20
Robot Type	Str	Normalmente “Pioneer”
Subtype	Str	Identifica el modelo de MobileRobots; ejemplo “p3dx-sh”
Sernum	Str	Número serial del robot
4mots	byte	Anticuado (=1 si es AT con sistema servidor P2OS)
RotVeltop	Int	Velocidad máxima de rotación; grados/seg.
Transveltop	Int	Velocidad máxima de traslación; mm/seg.
Rotacctop	Int	(Des) aceleración máxima de rotación; grados/seg2.
Transacctop	Int	(Des) aceleración máxima de traslación; mm/seg2.
PWMmax	Int	Máximo PWM del motor (limitado a 500).
Name	Str	Nombre único dado al robot
SIPcycle	byte	Tiempo del ciclo de los paquetes de información del servidor; ms.
Hostbaud	byte	Tasa de transmisión en baud para el cliente – servidor del HOST serial: 0 = 9.6k, 1 = 19.2k. 2 = 38.4k, 3 = 56.8k, 4 = 115.2k.
Auxbaud	byte	Tasa de transmisión en baud para el puerto serial AUX1, igual que el Hostbaud.
Gripper	Int	0 si no hay gripper; caso contrario 1.
Front Sonar	Int	1 si el sonar delantero está habilitado, de otra manera 0.
Rear Sonar	byte	1 si el sonar trasero está habilitado, de otra manera 0.
Lowbattery	Int	En décimas de voltio, la alarma se activa cuando la carga de la batería baja de este valor.
Revcount	Int	Es el número de espacios diferenciales del codificador que trabajan en una vuelta de 180 grados del robot.
Watchdog	Int	El tiempo en ms., antes de que el robot se pare si no ha recibido algún comando del cliente. Se reinicia al restablecerse la conexión.
P2mpacs	byte	1 activa los SIPS alternos, no se usa por ARCOS.
Stallval	Int	El PWM máximo antes de una parada. Si es mayor al PWM no se activa.
Stallcount	Int	El tiempo en ms. en el que se recupera después de una parada. El motor se inhabilita durante este tiempo.
Joyvel	Int	Velocidad de traslación establecida para manejo con joystick, mm/seg.
Joyrvel	Int	Velocidad de rotación establecida para menejo con joystick, mm/seg.
Rotvelmax	Int	Máxima velocidad de rotación actual; grados/seg.
Transvelmax	Int	Máxima velocidad de traslación actual; mm/seg.
Rotacc	Int	Aceleración actual de rotación; grados/seg2.
Rotdecel	Int	Desaceleración actual de rotación; grados/seg2.
Rotkp	Int	Valor Proporcional actual del PID para rotación.
Rotkv	Int	Valor Derivativo actual del PID para rotación.
Rotki	Int	Valor Integral actual del PID para rotación.
Transacc	Int	Aceleración actual de traslación; grados/seg2.
Transdecel	Int	Desaceleración actual de traslación; grados/seg2.
Trasnkp	Int	Valor Proporcional actual del PID para traslación.
Transkv	Int	Valor Derivativo actual del PID para traslación.
Transki	Int	Valor Integral actual del PID para traslación.
Frontbumps	byte	Número de segmentos del parachoques delantero.
Rearbumps	byte	Número de segmentos del parachoques trasero.
Charger	byte	1 para los modelos P3/Peoplebot o 2 si tiene el mecanismo de recarga automática y el circuito instalado en el robot, de otra manera 0.
Sonarcycle	byte	El ciclo de tiempo para la respuesta del sonar en milisegundos.
Autobaud	byte	1 si el cliente puede cambiar la tasa de baudios; 2 si esta implementada la opción de auto-baudios.
HasGyro	byte	1 o 2 si el robot está equipado con el dispositivo de corrección de giros; sino no lo está 0.
DriftFactor	Int	Valor del factor de deslizamiento utilizado.
Aux2baud	byte	Tasa de baudios para el puerto serial AUX2, mirar el HostBaud

Aux3baud	byte	Tasa de baudios para el puerto serial AUX3, mirar el HostBaud
Ticksmm	Int	Número de espacios del codificador por milímetro movido de la llanta.
Shutdownvolts	Int	Voltios DC x10 a los que el PC interno lo apaga.
Para la versión 1.5		
Versionmajor Versionminor	Str Str	Cadenas terminadas con NULL, para los números de versiones de ARCOS.
Chargethreshold	Int	Voltios DC x10 para el cargador del robot PowerBot
GyroCW	Int	Factor de calibración del giroscopio en sentido horario.
GyroCCW	Int	Factor de calibración del giroscopio en sentido anti horario.

Tabla. 4.5. **Contenidos del CONFIGpac**

4.20.1. Transferencias del HOST al AUX serial.

Para el control de uno de los puertos Auxiliares por parte del cliente se recurre a los comandos TT2 comando #42, TTY3 comando #66, TTY4 comando # 60, que corresponden a los auxiliares AUX1, AUX2 y AUX3 respectivamente y que comunican mediante una cadena con el HOST.

ARCOS también posee tres buffers circulares para recibir los datos seriales provenientes de los puertos Auxiliares respectivos. ARCOS envía porciones sucesivas del buffer al cliente mediante el HOST serial, con su respectivo SIP: SERAUXpac (type =176; 0XB0), SERAUX2pac (type =184; 0XB8) y el SERAUX3pac (type =200; 0XC8), para obtener la información de los auxiliares se puede usar el GETAUX comando #43 para el Aux1, GETAUX2 comando #67 para el Aux2 y el GETAUX3 comando #61 para el Aux3.

Los valores enteros que se pueden utilizar son el 0 para anular el contenido del buffer respectivo, y si no se puede ocupar un valor hasta de 253 bytes para que ARCOS espere coger toda esa información proveniente del puerto auxiliar serial y enviarlos en el respectivo SIP: SERAUXpac, SERAUX2pac o SERAUX3pac.

4.21. Codificadores

El comando #19 ENCODER puede con un argumento de 1 manejar un codificador, si se coloca con un valor de 2 o más maneja una transmisión continua de paquetes en el SIP ENCODERpac (type 144; 0x90). Para descontinuar los paquetes se utilizan un argumento de 0 en el comando según los datos dado por la tabla 4.6.

ETIQUETA	DATOS	DESCRIPCION
Header	Integer	Exactamente 0xFA, 0xFB
Byte count	byte	Número de bytes de datos + 2 (comprobación)
Type	Byte	0x90
Left encoder	Integer Integer	Porción actual menos significante, o más significante de las cuentas acumuladas del codificador de la rueda izquierda.
Right encoder	Integer Integer	Porción actual menos significante, o más significante de las cuentas acumuladas del codificador de la rueda derecha.
Checksum	Integer	Comprobación de la integridad del paquete.

Tabla. 4.6**. Contenidos del SIP del ENCODERpac**

4.22. Sonidos del Buzzer

Los robots Pioneer3 poseen un buzzer de tipo pizo eléctrico en el panel de control que notifica de las condiciones del sistema, como por ejemplo estados de batería baja o paradas tal como se puede observar en la tabla 4.7.

Para una operación silenciosa, se puede recurrir al comando #92 SOUNTOG con un argumento de 0 para silenciar el zumbido desde el microcontrolador, o si se quiere habilitarlo nuevamente se utiliza un argumento de 1.

El comando #15 SAY permite ejecutar un sonido propio a través del buzzer. El argumento consiste en una longitud de cadena específica de bytes pares tanto de tonos como de duración. La duración es medida en incrementos de 20 milisegundos.

Un valor de tono de cero establece un silencio (descanso musical). Las siguientes 127 frecuencias (1-127) son las notas correspondientes MIDI. Los tonos y frecuencias se calculan de la siguiente manera:

Tono – 127 * 32

Las frecuencias equivalentes van desde 1 a 4096, en incrementos de 32Hz. Excepto por las notas MIDI, se debe experimentar con los tonos. A continuación se muestra la secuencia generada por ARCOS para la señal de alerta cuando el robot para o está con las baterías bajas:

50, 100, 20, 0, 50, 60, 0

Condición	Sonido del zumbador
Inicio	Un solo sonido
Inicio cliente	Un solo sonido de conexión
Cierre cliente	Un solo sonido de conexión
Batería baja	Pitos repetitivos
Parada de motores	Pitos repetitivos
Calibración del joystick	Pitos repetitivos
Modo Auto prueba	Pitos repetitivos
Joystick calibrado	Un solo sonido
Tiempo cumplido del Watchdog	Pitos repetitivos
Para de emergencia	Pitos repetitivos

Tabla. 4.7. **Sonidos del buzzer relacionados al funcionamiento del sistema**

4.23. TCM2

El TCM2 es un accesorio que integra un inclinómetro, magnetómetro, termómetro y brújula y se conecta a uno de los puertos seriales del microcontrolador ARCOS. Cuando se encuentra habilitado, en especial los servidores de la brújula TCM2 se reportan las lecturas de la cabecera del robot por cada incremento de ±2 grados como un byte de la brújula en los SIPs estándar, según se observa los argumentos de comando #45 TCM2 en la tabla 4.8 y paquetes de información del TCM2 en la tabla 4.9.

Para calibrar el dispositivo se puede usar el comando #45 TCM2 client, así como para pedir información adicional en forma del TCM2pac. ARIA soporta el TCM2 de una buena manera. Para mayor información se detallará en el capítulo siguiente del Programa tipo cliente.

4.23.1. Calibración

Se debe calibrar el módulo TCM2 al menos una vez después de instalarlo en el robot y también es recomendable que se lo calibre si se cambia de ambiente de operación.

Para calibrar de manera simple se puede ejecutar la demostración de ARIA, entrar en el modo brújula, luego utiliza el modo de calibración, después de ello se debe entrar en el modo de tele-operación y utilizar las flechas izquierda o derecha para girar el robot en revoluciones de 720 grados.

Luego se vuelve al modo de calibración y se para la calibración. De manera alternativa se puede usar comandos directos para enviar desde el cliente un argumento al comando TCM2 de ’45 4’ para iniciar o de ’45 6’ para parar la calibración.

Comando ARCOS	**Argumento (Comando TCM2)**	**Acción**
45	0	Módulo apagado (solo software; no potencia baja en espera)
	1	Solo brújula (por defecto); lectura en el SIP estándar
	2	Envía SIP TCM2 simple.
	3	Envía SIPs TCM2 continuos.
	4	Habilita la calibración de usuario.
	5	Habilita la auto calibración.
	6	Para la calibración y envía un SIP TCM2, luego regresa por defecto al modo 1.
	7	Restablecimiento.

Tabla. 4.8. **Argumentos del comando #45 del TCM2**

ETIQUETA	**DATOS**	**VALOR/DESCRIPCION**
Header	Integer	0xFAFB
Byte count	Byte	23/Número de bytes + comprobación
Packet type	Byte	0xC0
Pitch	Integer	Grados en incrementos de 10; la máxima inclinación depende del tipo de módulo TCM2.
Roll	Integer	Grados en incrementos de 10; la máxima rotación depende del tipo de módulo TCM2.
X	integer	Campo magnético en el componente X, medido en incrementos de 100 uT.
Y	Integer	Campo magnético en el componente Y, medido en incrementos de 100 uT.
Z	Integer	Campo magnético en el componente Z, medido en incrementos de 100 uT.
Temperature	Integer	Grados de temperatura en incrementos de 10.
Error	Integer	Mapa de Bits de Códigos de error.
Calibration Scores	Byte	Puntuación H (0-9)
	Byte	Puntuación V (0-9)
	Integer	Puntuación M en incrementos de 100.
Checksum	Integer	Comprobación SIP.

Tabla. 4.9. **Paquetes de información del TCM2**

De manera similar para calibrar la inclinación y la rotación, mientras se mueve el robot es necesario mantener la brújula en la mano para girarla de lado a lado mientras se la rota. El comando #45 TCM2 de ARCOS en el modo 4 seguido del 6 causa que el módulo TCM2 se calibre y posteriormente almacene esos datos. Luego regresa al modo 1 después de enviar un SIP simple del TCM2 que contiene la calibración actualizada hacia el cliente conectado.

Este paquete y sus posteriores contienen los valores de calibración en los cuales se puede evaluar los parámetros calibrados.

4.23.2. Modo 7 -Reseteo

El comando de reseteo del módulo TCM2 obliga a un reseteo lento del módulo. Si se quiere un reseteo rápido se lo puede forzar a través del puerto auxiliar. Una vez iniciado o reseteado, el módulo TCM2 vuelve a su estado por defecto. Los valores establecidos para el módulo desde la fábrica se encuentran establecidos en 9600 baudios de tasa de transmisión para los datos estándar de la brújula, inclinómetro, termómetro (en grados Farenheit) y el magnetómetro, todos ellos se encuentran también en modo de espera y responden ante solo el carácter h. Se puede cambiar los parámetros de ARCOS manualmente mediante los comandos TTY2 o a través del software de Windows del módulo TCM2 proporcionado por el fabricante.

4.24. Computador Interno

La comunicación entre el PC a bordo y el microcontrolador se realiza en modo serial RS – 232 a través del respectivo COM1 (Windows) o el /dev/ttyS0 (Linux), y sus puertos HOST internos. Se debe establecer la tasa de comunicación del microcontrolador (HostBaud) para que sea igual al del puerto serial del PC a bordo.

El Pin9 RI en el puerto del HOST inicia en bajo y se pone en alto cuando las baterías se descargan por debajo del valor establecido en el parámetro ShutDownVolts de la FLASH. Se usa el programa genpowered para detectar la señal bajo y apagar automáticamente el PC, esto en el entorno Linux. Para el entorno Windows es un poco más complicado.

El programa genpowered de Windows como por ejemplo el ups.exe requiere un puerto serial específico que prefiere utilizar una línea CTS para indicar un voltaje bajo. DE acuerdo a esto, se puentea la señal RI del HOST COM1 a la señal del pin CTS del puerto adyacente COM2 en la computadora a bordo. Para mayor conveniencia el PC Versalogic VSBC8 ubicado en los robots Pioneer más nuevos comparten el conector del Pin 20 de la tarjeta madre con el COM1 y el COM2. De esta manera, para habitar en Windows el

ups.exe se puentea del Pin8 (COM1 RI) al Pin 16 (COM2 CTS) en el conector serial del VSBC8. Para otros tipos de implementación se lo puede hacer de manera similar, ya que la configuración del UPS permite escoger del puerto COM1 al COM4.

Una vez conectado el puerto, se inicia Windows como Administrador, luego se debe ir al Menú Inicio: Configuración: Panel de Control: Opciones de Energía y se selecciona el paquete USB. Selecciona seleccionar y en el cuadro de diálogo UPS selección se escoge el puerto COM2 (u otro), fabricante genérico, modelo simple. Y se da un click en siguiente.

En la interfaz de configuración UPS: se debe observar en la ventana y seleccionar la opción Falla de energía/ Encender baterías. Luego se presiona finalizar para almacenar la configuración y cerrar el cuadro de diálogo. Finalmente se presiona OK o Aplicar para habilitar los programas del UPS.

Se puede cambiar los valores del registro para que el PC se apague en vez de un tiempo de 1 minuto, a 2 minutos después de recibir una notificación de batería baja por parte del microcontrolador: Se usa el editor de registros hasta llegar a [HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\UPS\Config. Se cambia el valor de ShutdownOnBatteryWait de 1 a 2.

Para probar el ups.exe se puede utilizar el comando del cliente #250 que lo pone en modo mantenimiento. Para ello se debe enviar un voltaje falso por debajo del valor del ShutDownVolts para simular una condición de batería baja. ARCOS debe emitir alertas, luego de que el cliente se desconecta por un minuto y envía la señal de apagar al PC en el Pin de RI.

Para regresar al valor del voltaje real sede los hace enviando el argumento 0 al comando #250. El reseteo del microcontrolador cancela el cierre del sistema, a menos de que las baterías tengan un voltaje demasiado bajo.

Si el computador interno accionara falsamente en genpowerd o el ups.exe se debe poner el microcontrolador en modo de mantenimiento y arreglar los parámetros del computador a bordo.

4.25. Entradas y Salidas (E/S)

El microcontrolador SH2 viene con un número de puerto E/S que permite usar sensores y otros accesorios, cuya localización se muestra en el Anexo A.

4.25.1. E/S del usuario

Los conectores de las E/S del microcontrolador del Pioneer3 contienen 8 entradas digitales (ID0-7) y 8 puertos de salidas digitales, así como un puerto análogo – digital (AN0). El mapa de bits y los estados de los 16 puertos digitales y del puerto análogo aparecen de manera automática y continua en el SIP estándar en sus respectivos bytes DIGIN, DIGOUT y ANALOG. Cuando no existe una conexión física, las entradas digitales y los valores del puerto AN pueden variar sin previo aviso.

Para escoger uno o más puertos de las salidas digitales DIGOUT se debe usar el comando #30 en el microcontrolador ARCOS. Eléctricamente, los valores de los puertos digitales en alto (1) son de 5VDC (Vcc) y en bajo (0) son de 0VDC (GND). Las salidas DIGOUT toman como argumento 2 bytes (entero sin signo). El primer byte es una máscara cuyo bit patrón selecciona (1) o ignora (0) el estado del bit correspondiente, en el segundo byte se selecciona el puerto digital al seleccionar (1) o deseleccionar (0).

Por ejemplo, en el comando de ARCOS configurar las salidas digitales cero y tres (OD0 y OD3), resetear el puerto cuatro (OD4) y dejar los demás como están:
250, 251, 6, 30, 27, 25, 9, 55, 36

4.25.2. Paquetes E/S

No todos los puertos análogos y digitales aparecen en el SIP estándar. Por lo tanto, el programa cliente debe solicitar los SIPs IOpac (type = 240; 0xF0), los cuales contienen

todos los puertos E/S asociados con el microcontrolador y aparecen en varios conectores, incluyendo E/S del usuario, E/S generales, parachoques y también IRs.

El comando #40 IOREQUEST puede tener tres argumentos de cero, uno o dos. El argumento uno solicita un paquete simple a ser enviado en el siguiente ciclo de comunicaciones entre el cliente y el servidor. El argumento dos dice a ARCOS que envíe paquetes IOpac continuamente, aproximadamente uno por ciclo dependiendo de la velocidad del puerto serial y otros SIPs pendientes. El argumente cero del IOREQUEST para el envío continuo de los paquetes IOpac como se detalla a continuación en la tabla 4.10.

ETIQUETA	DATO	VALOR	DESCRIPCION
Header	2	0xFA, 0xFB	Cabecera común
Byte count	1	22	Numero de bytes +2
Type	1	0xF0	Tipo de paquete
N DIGIN	1	4	Número de bytes de entradas digitales
DIGIN	1	Varía 0-255	Mapa de bits ID0-8
Frontbumps*	1	Varía 0-255	Mapa de bits del parachoques frontal
Rearbumps*	1	Varía 0-255	Mapa de bits del parachoques trasero
IRs	1	Varía 0-255	Entradas IRs
N DIGOUT	1	1	Número de bytes de salidas digitales
DIGOUT	1	Varía 0-255	Bytes de las salidas digitales
N AN	1	9	Número de valores del puerto A/D
AN	10	5 enteros variando 0-1023	Valores de entradas de los puertos análogos 0-7 con una resolución de 10 bits: 0-1023 = 0-5 VDC
Battery	2	0-1023	Entrada análoga de la batería (AN3 Pioneer3)
Checksum	2	Varía	Cálculos de comprobación

Tabla. 4.10. **Contenido del paquete IOpac**

4.25.3. E/S de Parachoques e IRs

Dos conectores microfit de 10 posiciones en el microcontrolador ARCOS proveen 16 entradas digitales que normalmente son usadas para el accesorio de parachoques, pero puede ser utilizados para otros propósitos.

Del mismo modo, el conector del motor en el microcontrolador posee ocho entradas digitales que normalmente son usadas para el desempeño del los sensores IRs, cuyos estados se muestran en el mapa de bits.

Normalmente estas entradas tanto de sensores IRs como de los parachoques se encuentran en alto (1) y se vuelven bajo (0) cuando uno de los puertos se ha activado. Las entradas de los parachoques incluso se activan cuando se realizan paradas en el SIP estándar, pero a diferencia del IOpac, estos son modificados por la máscara InvertBumps. Todos los parachoques y datos de los IRs aparecen en el paquete IOpac.

4.26. Joystick

El comando #17 JOYREQUEST puede tener tres argumentos: 0,1 y 2, para solicitar información acerca del Josytick, si éste se encuentra habilitado. El argumento 1 solicita un paquete simple (tyoe = 248; 0xF8) a ser enviado en el siguiente ciclo de comunicaciones del cliente.

El argumento de 2 dice a ARCOS que envíe paquetes JOYSTICKpac de manera continua aproximadamente uno por ciclo dependiendo de la velocidad del puerto serial y otros SIPs pendientes. El argumento de 0 en el JOYREQUEST para los paquetes continuos JOYSTICKpac; según la siguiente tabla 4.11 su contenido es:

ETIQUETA	DATOS	VALOR	DESCRIPCION
Header	2	0XFA, 0XFB	Cabecera común
Byte count	1	11	Variable
Type	1	0XF8	Tipo de paquete
Button 0	1	0 o 1	1 = Botón presionado
Button 1	1	0 o 1	1 = Botón presionado
X-axis	2	Varía 0-1023	Rotación análoga
Y-axis	2	Varía 0-1023	Traslación análoga
Throttle	2	Varía 0-1023	Parámetros del acelerador
checksum	2	Variable	Cálculos de comprobación

Tabla. 4.11. **Contenido del paquete JOYSTICKpac**

4.27. Gripper

ARCOS es compatible con el GRIPPERpac (type = 224; 0xE0) y con su comando relacionado #37 GRIPREQUEST para obtener información del estado y configuración de los servidores. Ver tablas 4.12 y 4.13.

Normalmente se encuentra deshabilitado, el programa cliente es quien se encarga de pedir uno o varios paquetes continuos del Gripper (Con un argumento mayor a 1 en el comando). Para parar el envío continuo de los paquetes se pone el argumento 0 en el GRIPREQUEST.

ETIQUETA	DATO	DESCRIPCION
header	Int	Exactamente 0xFA, 0xFB
Byte count	Byte	Número de bytes + 2 (comprobación)
Type	Byte	Tipo de paquete = 0xE0
hasgripper	Byte	Tipo de Gripper: 0 = ninguno, 1 = Pioneer, 2 = Peoplebot
Grip_state	Byte	Mire la tabla que sigue
Grasp_time	Byte	Control del tiempo de presión de agarre, en ms.
checksum	Integer	Cálculos de comprobación

Tabla. 4.12. **Contenido del paquete GRIPPERpac**

	BIT	ESTADO ALTO (1)
G	0	Paletas abiertas
R	1	Paletas cerradas
I	2	Paletas moviéndose
P	3	Error del Gripper
L	4	Transporte arriba
I	5	Transporte abajo
F	6	Trasporte en movimiento
T	7	Error de transporte

Tabla. 4.13. **Byte GRIP_STATE del GRIPPERpac**

Se debe notar que los bits de estado e información del Gripper pueden ser obtenidos desde los valores respectivos de DIGIN o DIGOUT de os SIPs estándar relacionados con los estados de las E/S del usuario.

4.27.1. Corrección de deslizamiento en giros

Con el accesorio giroscopio, el programa cliente (ARIA 1.3 y posterior) en ARCOS (2.0 y posterior) puede detectar y compensar los cambios por deslizamiento que no son detectados por los codificadores, por ejemplo en el deslizamiento de las ruedas. El microcontrolador soporta el giroscopio mediante los puertos de entrada análogos - digitales AN6 y AN7.

Para habilitar el giroscopio, se debe establecer el parámetro FLASH HasGyro con el valor de 1 en el lado del cliente o de 2 cuando el manejo se hace a través del lado del servidor

usando la herramienta ARCOScf que se analizará posteriormente. Si se establece a cero el giroscopio no se habilita.

ARCOS recoge y promedia una tasa de 10 bits (0-1023) de datos del giro y 8 bits (0-255) de datos de temperatura en un periodo de tiempo de 25 milisegundos, que es el ancho de banda del giroscopio. Cuando no se encuentra en movimiento el valor se centra alrededor de 512, dependiendo de la temperatura del giroscopio y de otros factores de calibración que se usan en el deslizamiento y se corrigen sobre la marcha. Los valores bajo ese punto indicado indican un sentido anti horario, mientras que sobre ese valor son del sentido horario. La tasa máxima del giroscopio es de 300 grados por segundo.

4.27.2. Giroscopio desde el lado del cliente

Con el HasGyro puesto a uno, ARCOS recoge las lecturas del giroscopio y por petición del cliente con el comando #58 GYROREQUEST y argumento 1 (cero para cancelar), envía los datos recolectados justo antes del SIP estándar al cliente conectado mediante el paquete de información del servidor GYROpac (type = 0x98) para su procesamiento. EL contenido del SIP se puede observar en la tabla 4.14.

El análisis de los datos del giroscopio así como de sus modificaciones posteriores para las correcciones del deslizamiento son realizadas por el cliente, y son soportadas en las últimas versiones de ARIA (1.3 y posteriores).

Para más detalles se puede observar el archivo Aria/src/ArAnalogGyro.cpp. Los factores de corrección y calibración del giroscopio por parte del cliente se pueden encontrar en el valor GyroScaler del archivo de parámetros del robot Aria/params/*.p, y debe ser calibrado específicamente para un modelo de robot en particular y su radio de giro. Para más detalles de analizará luego la sección de Calibración y mantenimiento.

El GYROpac consiste en un byte de cuenta de la tasa de giro y temperatura recolectadas desde el último ciclo (típicamente un ciclo de 100 milisegundos), seguido de un número par de tasa de giro/temperatura entero/byte.

ETIQUETA	DATO	VALOR	DESCRIPCION
Header	2	0xFA, 0xFB	Cabecera común
Byte count	1	Xx	Variable
Tyoe	1	0x98	Tipo de paquete
N Pairs	1	X	Número de datos pares del giroscopio
Para Pares N (N pairs)			
Rate	2	Varía 0-1023	Tasa de Giro
Temperature	1	Varía 0-255	Temperatura del giroscopio
Cheksum	2	Variable	Cálculos de comprobación

Tabla. 4.14. **Contenidos SIP del GYROpac**

4.27.3. Giroscopio desde el lado del servidor

Con el parámetro FLASH HasGyro puesto en 2, ARCOS maneja el giroscopio internamente. Aspectos tales como lectura de promedios cada 25 milisegundos, pero no envía el paquete GYROpac. En vez de ello ARCOS fusiona las lecturas del giroscopio y el codificador para calcular las posiciones Xpos, Ypos y Thpos en los SIPs estándar.

Al igual que en ARIA, el giroscopio tratado desde el lado del servidor automáticamente detecta su punto céntrico el cual utiliza para compensar los deslizamientos. A diferencia del lado del cliente, en el servidor el giroscopio presenta dos factores de calibración en la FLASH, que son el GyroCW y el GyroCWW, los cuales determinan las tasas de giro actuales en el sentido horario y anti horario respectivamente y necesitan ser establecidas para cada para giroscopio del robot, algo que comúnmente se hace en la fábrica y que necesita ser hecho periódicamente.

El comando #38 (GYROCALCW) y el comando #39 (GYROCALCCW) del cliente permiten cambiar los valores horarios y anti horarios de manera temporal, para ayudar durante la calibración. Más detalles se muestran en la sección de calibración y mantenimiento.

Si el giroscopio en el lado del servidor funciona mal, de tal manera que no sea capaz de determinar el punto céntrico, el robot realiza una parada, y emite un pitido excesivo desde el pizo eléctrico del robot. El comando GYROREQUEST puesto en cero deshabilita el giroscopio desde el lado del cliente, mientras que un argumento de 1 lo vuelve a habilitar.

4.28. Sistema de Recarga Automática

El accesorio de recarga automática u su circuito asociado de manejo de cargas en el robot pueden ser controlados desde el microcontrolador del robot.

4.28.1. Control de puertos digitales

Cuando se establece el puerto “inhibido” OD4 en alto (1) en el Pin 10 del conector de E/S del Usuario, el mecanismo de contacto del robot procede a desengancharse, se retrae de la plataforma de carga u queda lista para un futuro despliegue. Cuando el puerto OD5 de “despliegue” en el Pin 12, se pone el alto, el puerto OD4 se pone en bajo, y se despliega el mecanismo de contacto el cual queda listo para la plataforma de carga.

Cuando se encuentra en la posición de despliegue, el mecanismo se estabiliza y requiere un menor esfuerzo para hacer contacto. Si entra en contacto el robot con la plataforma de carga, el circuito interno del robot se activa y hace que el mecanismo de despliegue realice un menor esfuerzo para mantener al robot mientras éste se carga.

Para minimizar el calentamiento y un daño eventual del actuador, la línea de despliegue debe ser activada solo por cortos periodos, máximo por 10 segundos cada ciclo.

El programa cliente puede iniciar el mecanismo de carga automática activando o desactivando los puertos de salidas digitales, con el comando #30 de ARCOS DIGOUT. Sin embargo, para mejores resultados, se recomienda utilizar los servidores de ARCOS de recarga automática.

4.28.2. Servidores de recarga automática

Para utilizar los servidores de recarga automática de ARCOS, se debe primero habilitarlos en los parámetros FLASH del robot. Para ello se usa la herramienta de configuración ARCOScf y se establece el calor del parámetro Charger a uno (cero para deshabilitar) y se guarda el valor.

Posteriormente, para una operación autónoma del robot con el mecanismo de recarga automática, estableciendo una conexión cliente servidor entre ARIA o un cliente similar y el microcontrolador del robot. El comando #68 CHARGE de ARCOS con un argumento entero de 1 para detener automáticamente el movimiento del robot y desplegar el mecanismo de contactos para la carga del robot.

Si el robot no se engancha con la plataforma de carga después de 5 segundos el mecanismo se retrae automáticamente, durante este tiempo el sistema del robot no responde. De acuerdo a esto, el cliente debe esperar al menos este tiempo para volver a reanudar la actividad.

Mientras los motores se encuentran enganchados, el mecanismo de carga no puede ser desplegado, excepto por el comando CHARGE. Para mejor control y seguridad, se debe considerar usar el comando #68 de ARCOS con un argumento de cero para cancelar de manera satisfactoria la carga y retraer el mecanismo de recarga automática.

Además de los comandos mencionados, también se puede cancelar la recarga y retraer su mecanismo manualmente con el botón de despliegue de carga, como se explico en el capítulo correspondiente a los accesorios del robot. Se debe tener en cuenta que los comportamientos manejados por el cliente del mecanismo de recarga pueden ser contrarios a las acciones que se quiere.

Por ejemplo, el cliente puede, enganchar los motores y hacer automáticamente que el robot se acople con el mecanismo de carga y restablecer su carga si nota que se ha perdido la energía de carga al presionar el botón de despliegue del mecanismo de recarga.

El programa cliente activar o desactivar la conexión entre cliente – servidor sin interrumpir la recarga, siempre y cuando el contacto con el dispositivo de recarga se mantenga y no se realice otra acción que pueda interferir con la carga. Una vez que se ha desconectado el cliente, se aplican las reglas para el acoplamiento manual explicados con anterioridad.

4.28.3. Monitoreo del ciclo de recarga

Tres señales digitales indican los estados de recarga de la batería con el mecanismo de recarga automática. Todos ellos aparecen en los SIPs estándar.

La señal que indica que la "potencia está bien" aparece en ambos bits de las E/S del usuario DIGIN correspondientes al bit 6 y el bit 10 en las FLAGS enteras de los SIPs estándar, aunque sus estados están inversamente relacionados. El bit 6 del DIGIN normalmente se encuentra en alto (1) cuando no se está cargando o cuando el mecanismo de recarga automática no está instalado, se pone en bajo (0) cuando el sistema se engancha con la plataforma de carga.

Por el contrario, el bit 10 en las FLAGS se encuentra normalmente en bajo y se cambia a lato cuando el robot se encuentra cargándose. Por razones de compatibilidad con diferentes accesorios de recarga, se recomienda que el cliente monitoree estos bits de las FLAGS y no la línea de los DIGIN para determinar su el robot se encuentra recibiendo energía de la plataforma de carga, se observa estos estados según la tabla 4.15.

ESTADO DE CARGA	SOBRECARGA (ID7)	~VOLTIOS	CORRIENTE DE CARGA	ESTADO DEL BYTE DE CARGA DEL SIP
Desconocido	?	?	?	-1
No carga	0 o 1	Cualquiera	0	0
Fallo	1	Descarga- ~14V	6 A	1
Sobrecarga	0	~14-14.7	Decrece a ~1 A	2
Flotante	1	~13.5	< 1 A	3

Tabla. 4.15**. Estados del ciclo de recarga**

Los bytes de DIGIN y DIGOUT de los SIPs estándar también indican los estados de carga sede sus entradas y salidas digitales asociadas. Los bit 4 y 5 del DIGOUT se encuentran inhibidos y despliegan los puertos de salidas como se describiera anteriormente.

El bit 7 del DIGIN, correspondiente al conector de E/S del usuario de la entrada digital ID7 en el pin 15, indica el ciclo de recarga de la batería, con el valor SIP de la batería, esto ayuda al cliente del robot a determinar inmediatamente la vida de la batería y el tiempo de operación.

El bit ID7 de “sobrecarga” se establece en 1 cuando la batería se encuentra muy por debajo del voltaje de carga y a su máxima corriente. Durante este “Fallo” del periodo de carga, el voltaje de la batería se eleva alrededor de 13.8-14V. El bit ID7 de sobrecarga se establece en bajo (0) cuando las baterías se encuentran en un nivel de 80% a 90% de la carga total: es decir desde 13.8 a 14.7V.

Finalmente el cargador regresa a “flotante”, manteniendo la carga total a mucho menos corriente y un voltaje de carga (~13.5V). En modo flotante, el bit de sobrecarga ID7 se pone a cero.

Las versiones de ARCOS 1.5 y posteriores incluyen un byte de estado de carga al final de los SIP estándar que decodifica estos estados de carga para el usuario. De acuerdo a esto, el cliente puede tomar decisiones estratégicas de recarga al monitorear el estado del byte de carga o los bits individuales de “potencia-bien” y “sobrecarga”, así como el voltaje de la batería.

Hay un aspecto también que se debe tomar en cuenta que las baterías de plomo tienen un periodo más largo de carga y por tanto en modo flotante normalmente se carga una vez al día.

4.29. Actualización y Reconfiguración de ARCOS

4.29.1. Calibración y mantenimiento

Las plataformas Pioneer3 requieren de un mantenimiento mínimo detallado a continuación.

4.29.1.1. Inflamiento de las llantas

Para una navegación eficaz de las plataformas se debe tener las llantas correctamente infladas. Cada neumático debe tener una presión de 23 psi, Si se cambia este valor, se debe

también ajustar los valores FLASH del driftFactor, el TicksMM y el revCount, explicados anteriormente.

4.29.1.2. Calibración del Robot

El robot viene con parámetros FLASH ajustados para un buen funcionamiento en superficies planas con su carga original. Si se va a operar el robot en diferentes superficies y con cargas más livianas o pesadas, probablemente se necesita recalibrar varios de los parámetros de operación, como son el driftFactor, revCount, ticksMM y los PIDs.

El programa de demostración de ARIA, tiene dos modos que pueden ayudar a realizar estas configuraciones. En el modo ‘p’ de posición, la demostración muestra la actual posición. Para resetear a cero solo se presiona la tecla ‘r’. Si se presiona la flecha izquierda o derecha el robot gira 90 grados en la dirección horaria y anti-horaria respectivamente. Las flechas arriba y abajo hacen que el robot avance un metro hacia delante o hacia atrás respectivamente.

El modo directo ‘d’ del ARIA permite cambiar una variedad de parámetros sobre la marcha, enviando un número de comando al servidor ARCOS y su valor a usarse en la sesión actual. Para reemplazar los valores por defecto en la FLASH, se debe usar el ARCOScf.

De acuerdo a esto, para una calibración apropiada del robot, en primer lugar se debe usar ARCOScf, grabar y guardar en un disco los valores actuales para los parámetros respectivos, como por ejemplo para el drftFactory el revCount. Luego debe conectarse con la demo del ARIA y enganchar el modo de posición para mover el robot. Tan preciso como sea posible se debe medir su posición actual y de movimiento y luego usar el modo directo para ajustar los valores reportados para la operación actual del robot y el ambiente de operación.

Nota: Deshabilitar desde el servidor el giroscopio (HasGyro 2) antes de calibrar el driftFactor y el revCount.

4.29.1.3. Calibraciones Estándar

Se debe comenzar con el driftFactor ya que este valor afecta también al ticksMM y al revCount. Para ello se debe dibujar una línea en el suelo paralela a la trayectoria de traslación del robot, luego de ello se debe manejar el robot hacia delante y hacia atrás al menos tres metros, y finalmente se ajusta el driftFactor (comando #89) para minimizar el deslizamiento del robot respecto a la línea dibujada.

Luego se debe manejar el robot hacia delante o hacia atrás uno o más metros y compara con la distancia actual trasladada, que para mayor exactitud se la puede tomar de la demostración de ARIA que da la distancia recorrida en milímetros. Luego se ajusta el ticksMM (comando #93), para que su valor coincida con el de la distancia recorrida.

Del mismo modo, se debe rotar el robot u como parar el ángulo medido con el reportado en la cabecera (th). Se ajusta el revCount de acuerdo a la medida del los espacios diferenciales del codificador hasta alcanzar una rotación de 180 grados de acuerdo al comando #88.

Finalmente, se debe manejar el valor y ajustar sus parámetros PID, los valores de aceleración y velocidad hasta alcanzar la operación deseada de acuerdo al entorno de operación.

4.29.1.4. Calibración del Giroscopio

Con el cliente (HasGyro puesto a 1), ARIA recupera la tasa de datos y los maneja desde el lado del cliente. Consecuentemente, no se debe deshabilitarla cuando se realiza las calibraciones con el programa de demostración del ARIA. En efecto, el modo de posición ‘p’ muestra los valores del giroscopio y del codificador separadamente parta luego calibrarlos. Se debe luego ajustar los valores del GyroScaler en el archivo de parámentros de ARIA: Aria/params/*.p (por ejemplo p3dx-sh.p)

Cuando se usa desde el lado del servidor, se lo debe deshabilitar cuando se realiza las calibraciones de los codificadores estándar, con el comando #58 del cliente y un argumento de 0 o simplemente poner el valor de HasGyro a 0. Luego después de habilitarlo nuevamente, se ajusta el GyroCW y el GyroCWW como se los hizo con el revCount.

Antes de enviar estos valores a la FLASJ es mejor probarlos con los comandos de los clientes GYROCALCW (#38) y el GYROCALCCW (#39).

Cuando el comportamiento del robot sea el correcto, es decir que el robot se mueve y rota las distancias y giros correctos, y se maneja con un buen desempeño, se debe almacenar estos valores en los parámetros de la FLASH del robot con el ARCOScf, no sin antes guardar una copia en un archivo .rop para una futura referencia.

4.29.1.5.Lubricación de manejo

Los motores y cajas de los robots Pioneer3 son sellados y tienen un mecanismo de auto lubricación, por lo cual no es necesario poner grasa ni aceite adentro. Ocasionalmente se puede poner una o dos gotas de aceite en los ejes entre las llantas y la carcasa. Otro aspecto importante también es el de mantener limpios los ejes de pedazos de alfombra o lanas que puedan enredarse en los ejes.

4.30. Baterías

Las baterías de plomo con las que funciona el robot Pioneer3 mantienen un periodo largo la última carga, sin embargo, una descarga severa puede afectar de manera considerable a la batería, por lo cual es recomendable que no se opere el robot si el voltaje de las baterías cae por debajo de los 11V.

4.30.1. Cambio de Baterías

Exceptuando por el mecanismo de recarga automática, el robot Pioneer tiene puertas especiales y con seguro para un fácil acceso a las baterías. Simplemente se debe quitar el seguro de la puerta, y localizar la o las baterías que se vaya a cambiar.

Para remover una batería, se la debe jalar, para ello cuenta con una ventosa de accesorio que ayuda a succionar la batería. Los contactos en la placa de alimentación y en las baterías eliminan la necesidad de conexión con cables.

De manera similar, solo se debe deslizar las baterías para ponerlas en su caja correspondiente. Se debe cargar las baterías de tal manera que se equilibre el peso del robot, así por ejemplo si se usa una batería ésta debe ir en el centro y si se usan 2 debe ir una a cada lado.

4.30.2. Cambio de batería en caliente (Hot swap)

Se puede cambiar las baterías del Pioneer3 sin interrumpir la operación del los sistemas internos (exceptuando los motores por supuesto), para ello se debe conectar el cargador, el cual alimentará el robot mientras se cambia la(s) batería(s). O si se tiene dos o tres baterías, se puede cambiar cada una por otra totalmente cargada, mientras se encuentre al menos una batería para proveer energía.

4.30.3. Carga de las baterías

Si se tiene un accesorio de carga estándar, se lo debe situar en un tomacorriente con voltaje de 110V AC (o 220V estándar Europeo). Se localiza el plug al final del cable y se lo inserta en el espacio de carga del robot, que se encuentra cerca del interruptor de encendido. El Led del cargador indica el estado de carga, tal como lo marca en su carcasa.

Le toma al robot al menos 12 horas o menos para cargar totalmente las baterías, todo dependiendo del nivel de descarga. Si se usa el accesorio de carga rápida puede tomar solo unas 3 horas. Sin embargo se puede operar el robot mientras se está cargando, aunque esto restringe su movimiento.

4.30.4. Sistema de carga automática

El sistema de carga automático optimiza las condiciones para poder cargar las 3 baterías de plomo de 21Ahr y 12V que tiene el robot para operar todos los sistemas a bordo.

4.30.5. Cargadores de batería alternativos

El terminal interno del centro del cargador corresponde al positivo (+). Mientras que el lado que se encuentra en el exterior es el negativo (-). Tiene un diodo de protección interna que asegura la protección ante una equivocación de la polaridad.

De todas maneras, si se decide utilizar otro cargador alterno, se debe asegurar la correcta conexión de los terminales del cargador con el terminal del robot.

Un cargador alterno al menos debe proveer 0.75 A a un voltaje de 13.75 a 14VDC por batería, y no más de 2.25 A por batería. Por ejemplo el accesorio de carga rápida es un cargador de 4 A y debe ser usado para cargar al menos 2 baterías.

Si se utiliza otro cargador se debe asegurar que sea limitado en su corriente y voltaje para no sobrecargar las baterías y que éstas se dañen.

CAPÍTULO 5

SOFTWARE TIPO CLIENTE

5.1. ARIA[7]

El software de Interfaz Robótica Avanzada para Aplicaciones (Advanced Robotics Interface for Applications, ARIA) es un entorno basado en código abierto desarrollado en C++ que provee una interfaz robusta al cliente para una variedad de sistemas robóticos inteligentes, incluyendo el microcontrolador del robot y accesorios opcionales. ARIA también viene con el cliente ARNetworking el cual provee una capa crítica para comunicaciones basadas en TCP/IP con las plataformas robóticas existentes en la red. Su estructura se observa en la figura 5.1.

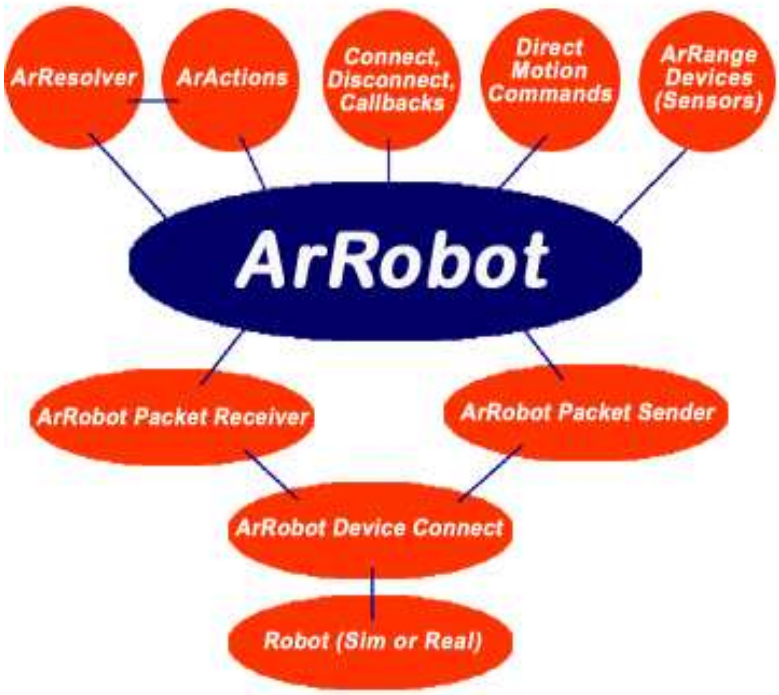

Figura. 5.1. **Estructura de ARIA**

[7] Pioneer3 Operation Manuals

ARIA/ARNetworking son las plataformas ideales de integración para el control adecuado del software del robot (software servidor), ya que ellos son capaces de manejar los detalles de bajo nivel en las interacciones del entorno cliente – servidor, incluyendo la creación de redes y comunicaciones seriales, comandos e información del servidor, procesamiento de paquetes, ciclos de tiempos y multitareas de soporte y control de accesorios como por ejemplo el escaneo mediante láser, movimiento de los giroscopios, sonares y muchos otros.

ARIA y ARNetworking vienen con su propio código fuente, el cual se pueda revisar y modificar para usar las diferentes aplicaciones incluyendo aplicaciones de sensores. Además de ser una Interfaz de Programación para Aplicaciones (API) con la cual se pueden desarrollar varias tareas, ARIA también viene con diferentes librerías que sirven de base para incrementar capacidades adicionales, como rutinas de navegación avanzadas, las cuales son desarrolladas con librerías de ARNL o SONARNL que trabajan en conjunto con las librerías de ARIA.

ARNetworking viene incluida en el ARIA y se encarga básicamente de la comunicación en la red. Dentro del entorno ARIA se incluyen también otras librerías que se utilizan para fines especializados como pueden ser, síntesis de voz, grabación y reproducción de audio por la red, captura de imágenes, seguimiento de colores, etc. Todas estas aplicaciones pueden ser desarrolladas y probadas antes de correrlas en el robot, mediante el simulador que incorpora la plataforma llamado MobileSim.

Para empezar a trabajar con el ARIA es necesario estar familiarizado con el entorno de programación C++, sus conceptos, el uso de clases y objetos con herencia simple, punteros, uso de memoria, contenedores STL (plantilla estándar de la librería), y el proceso de compilación y ejecución.

5.1.1. Entorno Java y Phyton

Para mayor versatilidad se han incorporado dos nuevos entornos de programación que son Java y Phyton, los cuales están abiertos para cualquiera de los clientes de MobileRobots como son ARIA, ARNetworking, SONARNL y ARNL. Para ello cada uno cuenta con una biblioteca, la cual es tomada por SWIG (Interfaz que conecta un programa desarrollado en

C o C++ con una plataforma de programación de alto nivel) y conectada con su respectivo lenguaje de programación.

5.1.2. Licencia

ARIA trabaja bajo licencia pública GNU, lo cual significa que todo el código incluido puede ser copiado. Sin embargo si se distribuye cualquier trabajo, se debe incorporar todo el código fuente, incluyendo ARIA con todas sus modificaciones hechas, bajos los mismo términos de licencia. Se desarrolló bajo esta forma de código abierto para que el usuario pueda modificar y compartir las mejoras desarrolladas con la comunidad robótica de MobileRobots.

5.1.3. Documentación y convención de código

ARIA sigue la siguiente convención de códigos:

1. Los nombres de Clases comienzan con letra mayúscula
2. Enumeraciones comienzan con letra mayúscula o todo en mayúsculas.
3. Evitar siempre que sean posibles definiciones pre-procesador (en lugar de enumeraciones o métodos en línea).
4. Variables miembro en clases están precedidas por “my”
5. Variables estáticas en clases están precedidas por “our”
6. Las funciones miembro siempre comienzan con minúscula.
7. Cada palabra debe comenzar en mayúsculas, excepto en una variable o nombre de método como por ejemplo: likeThisForExample.
8. Todas las clases pueden ser usadas en programas multi-tarea. Proporcionan una API para protegerlo de un posible mal funcionamiento.

5.1.4. Relación Cliente – Servidor (ARIA - Robot)

El núcleo de procesos del robot móvil “servidor” se ejecuta a través del microcontrolador mediante ARCOS, tal como ya se lo describió con anterioridad. Estos procesos manejan las tareas de bajo nivel más críticas y sensibles del proceso de control y operación del robot, incluyendo aspectos como requerimientos de movimiento, estado de la cabecera del

robot, posición estimada, información de sensores por ejemplo sonar y brújula y también el manejo de accesorios como la cámara PTZ, el inclinómetro/brújula TCM2, y el brazo robótico Pionner 5-DOF.

La “plataforma robótica” se denomina al conjunto compuesto por el robot, su microcontrolador, el firmware y dispositivos integrados como el sonar. Sin embargo el firmware del robot no realiza tareas de alto nivel.

El cliente que se encuentra conectado mediante una PC realiza estas tareas de alto nivel más complejas para el control del robot, sus estrategias y tareas, tales como detección, evasión de obstáculos, fusión de sensores, localización, tareas de reconocimiento, mapeo, navegación inteligente, control de la cámara PTZ, movimiento del brazo robótico y mucho más.

El objetivo de ARIA es soportar todas estas aplicaciones del cliente y su comunicación con el firmware del robot, así como a los dispositivos que se conectan a la computadora a través de la plataforma robótica mediante un programa remoto.

El corazón de ARIA es la clase **ArRobot.** Esta clase se encarga de manejar todo el ciclo de comunicación con el firmware del robot, así como la recepción y provisión del acceso a los datos de operación de la plataforma robótica y provoca que las tareas en el ciclo y los comandos determinados sean enviados de vuelta del robot.

También sirve como referencia para otros objetos de ARIA (como dispositivos de rango) y una serie de funciones generales relacionadas con el robot móvil.

A través de esta infraestructura, ARIA provee un mecanismo potente para combinar comportamientos independientes, para lograr la coordinación del control de movimiento y una orientación inteligente; con estas acciones es posible implementar de manera fácil aplicaciones como tele-operación guiada, seguimiento visual, navegación autónoma, etc.

Otras clases de ARIA proveen interfaces para acceder al control de otros dispositivos como sensores donde se puede manipular el estado de reflección del sonar, telémetro del láser, unidades de inclinación y giro, brazos, dispositivos de navegación inercial y muchos otros.

5.1.5. Comunicación del Robot

Una de las funciones más importantes de ARIA, y una de las primeras cosas que la aplicación desarrollada debe hacer es establecer la conexión entre una instancia del objeto ArRobot y el sistema operativo del robot (firmware).

Además del robot móvil en sí, algunos accesorios como el sonar, el Gripper, la cámara PTZ, el brazo robótico, la brújula y otros más se encuentran conectados internamente a las líneas digitales de E/S del puerto AUX del microcontrolador, y también utilizan la conexión como lo hace el robot. (Por lo tanto la interfaz de las clases de estos objetos requiere una referencia a un objeto ArRobot que debe estar conectada para que los dispositivos funcionen).

Otros dispositivos como el medidor láser, tarjetas de video se conectan directamente a la computadora a bordo. Hay varias maneras de conectar un equipo que ejecuta ARIA al microcontrolador del robot o a un simulador. Se puede observar los diferentes seis modos de conexión entre ARIA y el robot en las figuras 5.2, 5.3, 5.4.

5.1.6. Conexión con el Robot o el Simulador

La clase ArSimpleConnector puede ser utilizada para configurar y realizar la conexión con el robot y opcionalmente con el medidor láser, basado en registros de parámetros del robot y configuración en tiempo de ejecución mediante argumentos de las líneas de comandos. ArSimpleConnector es usado en todos los programas de demostración del ARIA, incluyendo los ejemplos simpleConnect::cpp, wander::cpp y demo::cpp.

Entre otros beneficios ArSimpleConnector primero intenta conectarse con un simulador en un puerto local TCP, y si el simulador no se encuentra corriendo, se conectará al puerto local serial del robot.

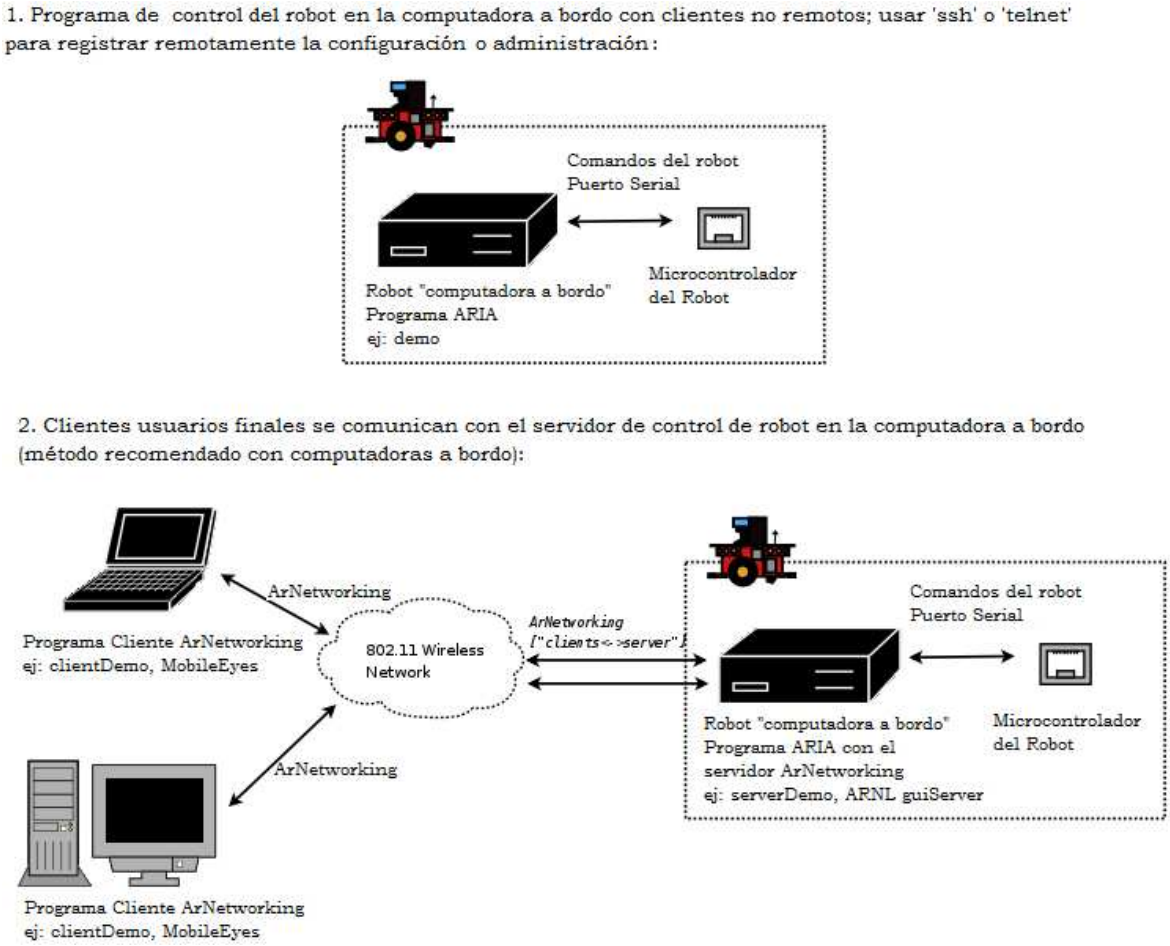

Figura. 5.2. **Modo de Conexión 1-2**

Esto hace fácil el desarrollar y depurar un programa utilizando el simulador, para luego simplemente copiarlo al robot y ejecutarlo desde ahí sin modificaciones.

ArSimpleConnector también analiza la estructura de algunos argumentos de línea de comandos si es que se suministran, los que pueden especificar explícitamente un nombre de anfitrión remoto, un puerto a conectarse via TCP o para especificar un puerto local serial alterno, para la conexión del robot. Si un programa usa ArSimpleConnector ejecutando la línea de comando “-help” imprimirá una lista de opciones.

A continuación se muestra un ejemplo en el cual se usa ArSimpleConnector para conectar las clases ArRobot y ArSick a su respectivo hardware (ArRobot al puerto serial por defecto o al simulador vía TCP y el objeto ArSick al respectivo medidor láser vía su respectivo puerto serial por defecto).

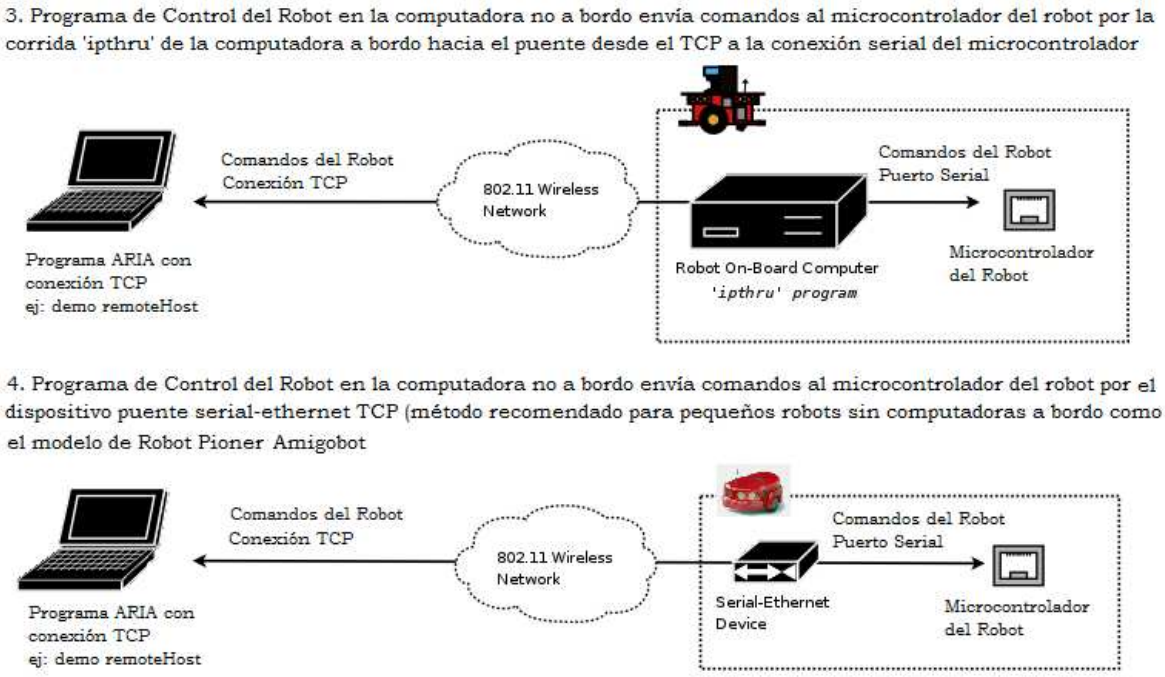

Figura. 5.3. **Modo de Conexión 3-4**

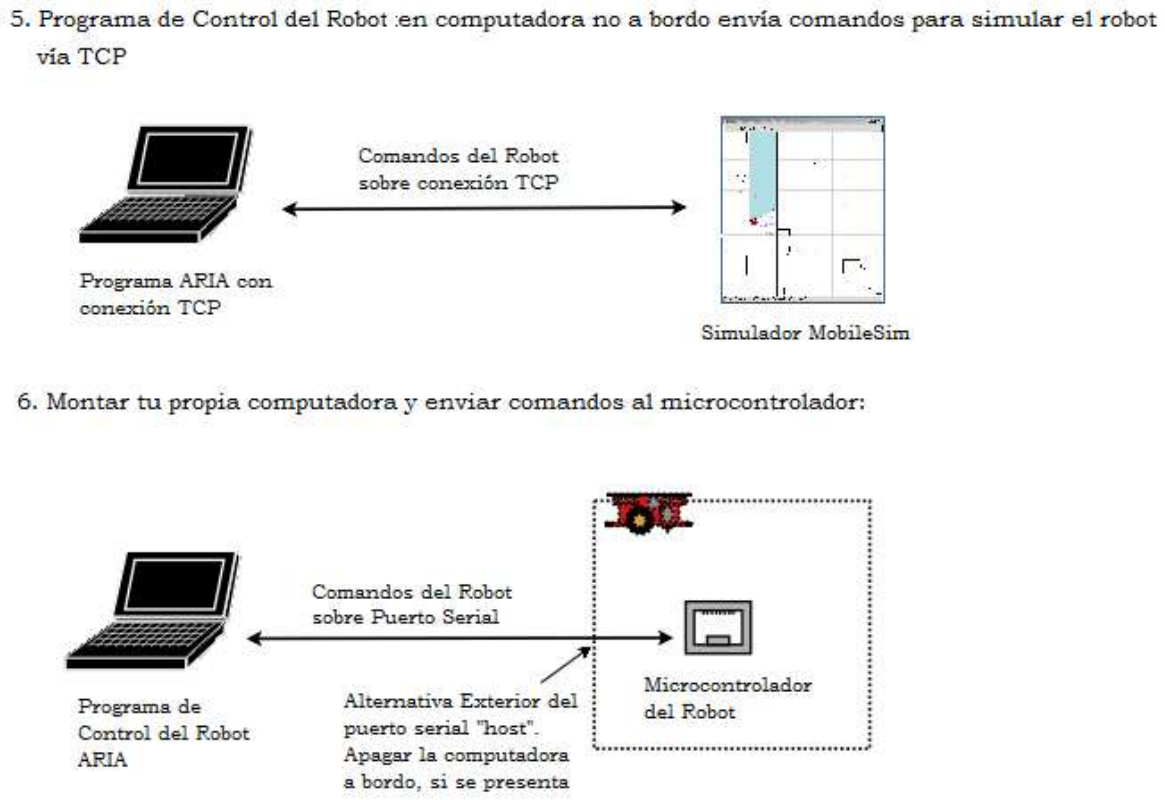

Figura. 5.4. **Modo de Conexión 5-6**

Si se quiere tener mayor control en la forma de conectar al robot, se verá posteriormente la manera difícil de conexión en la cual se abarcan más detalles, según la siguiente figura 5.5.

```
#include "Aria.h"
int main(int argc, char** argv)
{
  Aria::init();
  ArSimpleConnector connector(&argc, argv);
  ArRobot robot;
  ArSick sickLaser;

  // Chequea las líneas de commando usadas por SimpleConnector
  if(!connector.parseArgs())
  {
    connector.logOptions();
    exit(1);
  }

  // Conecta el robot, corre los dispositivos en el programa, y
conecta al láser.
  if(!connector.connectRobot(&robot))
  {
                              // Error!
    ...
  }
  robot.runAsync(true);
  sickLaser.runAsync();
  if(!connector.connectLaser(&sickLaser))
  {
    // Error!
    ...
  }
```

Figura. 5.5. **Ejemplo ArSimpleConnector**

5.1.7. ArRobot

Como ya se dijo con anterioridad ArRobot es el corazón del robot, el cual actúa como puerta de enlace de comunicaciones del robot y es la parte central del manejo del estado del robot, también es la herramienta para sincronizar las tareas y llamadas añadidas en el programa, objetos ArRobot, etc.

Los comandos del cliente y los paquetes de información del servidor (SIP) entre el ARIA y la plataforma móvil o el simulador utiliza protocolos basados en paquetes de información (En este contexto el cliente está utilizando el programa ARIA para operar el robot y el servidor es el firmware de la plataforma robótica).

ArRobot (usando las clases ArDeviceConnection, ArRobotPacketReceiver, ArRobotPacketSender, ArRobotPacket y ArSerialConnection) se ocupa de los detalles de la construcción y manejo de paquetes de los comandos al robot, así como también recibe y decodifica los paquetes recibidos del servidor (robot).

5.1.8. Manejo de paquetes

Los paquetes de Información del Servidor (SIPs) son paquetes enviados por el robot servidor que contiene información actualizada sobre el robot y sus accesorios. El SIP estándar es enviado por el robot a un cliente conectado, automáticamente cada 100 milisegundos (esta frecuencia puede ser configurada en los parámetros del firmware).

Este contiene la información de la posición actual del robot, así como una estimación de velocidad de traslación y rotación del robot, las lecturas actualizadas del sonar, el voltaje de la batería, estados de las E/S analógicas y digitales, y más.

Estos datos son almacenados y usados por el Estado de Reflexión de ArRobot (Posteriormente se tratará este estado) y son accesibles mediante métodos de la clase ArRobot. (En el código fuente de ArRobot el SIP estándar es también conocido como paquete “motor”).

Otros SIPs utilizan el mismo formato de paquetes del SIP estándar pero con un “tipo” de paquete diferente. Algunos ejemplos de otros SIPs son los datos de los puertos de E/S, datos del Gripper, o datos especiales del robot como la tasa de los codificadores. Para recibir estos SIPs extendidos, el programa debe llamarlos.

En ARIA esto es normalmente hecho por las clases de interfaces con dispositivos cuando ellos son inicializados o cuando la conexión con el robot se ha establecido. También se puede adjuntar un manipulador de paquetes desarrollados de ArRobot utilizando ArRobot::addPacketHandler(). Se puede hacer esto para realizar un procesamiento adicional de los SIPs extendidos, o si se crea una clase alterna de interfaces con los dispositivos.

5.1.9. Paquetes de comando

Para controlar la plataforma robótica, el programa cliente envía paquetes de comando a través de la conexión del robot. Esto puede hacerse usando las funciones de comandos de movimientos (Motion Command Functions), usando Acciones (Actions), o a un nivel más básico usando Comandos Directos (Direct Commands).

El resultado de cada uno de estos métodos son paquetes de comandos enviados al robot. Esto quiere decir que si se usan ambos métodos el de Acciones o Comandos de movimiento, o si módulos independientes del programa están siendo enviados, puede haber conflictos.

5.1.10. Ciclo de sincronización del Robot

El SIP estándar es enviado en un ciclo constante, y la recepción de este SIP ocasiona una nueva iteración del ciclo de procesamiento de tareas sincronizadas de ArRobot. Este ciclo consiste en una serie de tareas sincronizadas, que incluye el manejo de paquetes SIP, la invocación de las tareas de interpretación del sensor, el manejo de resolución de las acciones, el estado de reflexión, y la invocación de las tareas de usuario, todo en ese orden. Se puede visualizar mejor el ciclo de trabajo de ArRobot en la figura 5.6.

Dado que el ciclo de tareas es (normalmente) activada por la recepción de cada uno de los SIP (a menos que la plataforma robótica empiece a fallar al enviar los SIPs, o si la tarea de ciclo está completamente desasociado de la conexión del robot), cada tarea se invoca en un orden previsible, y tiene los datos más recientes para actuar, ninguna tarea se perderá la oportunidad de utilizar un SIP, y siempre que cada tarea no tome mucho tiempo para ejecutarse, cada SIP es manejado tan pronto como sea posible después de que el robot lo envíe.

5.1.11. Ciclo de tareas de ArRobot

Para empezar el ciclo de procesamiento, se llama a ArRobot::run () para entrar en el ciclo de manera sincrónica, o ArRobot::Async() para ejecutar el ciclo de una nueva base. Si se usa ArRobot::stopRunning() se detiene el ciclo de procesamiento.

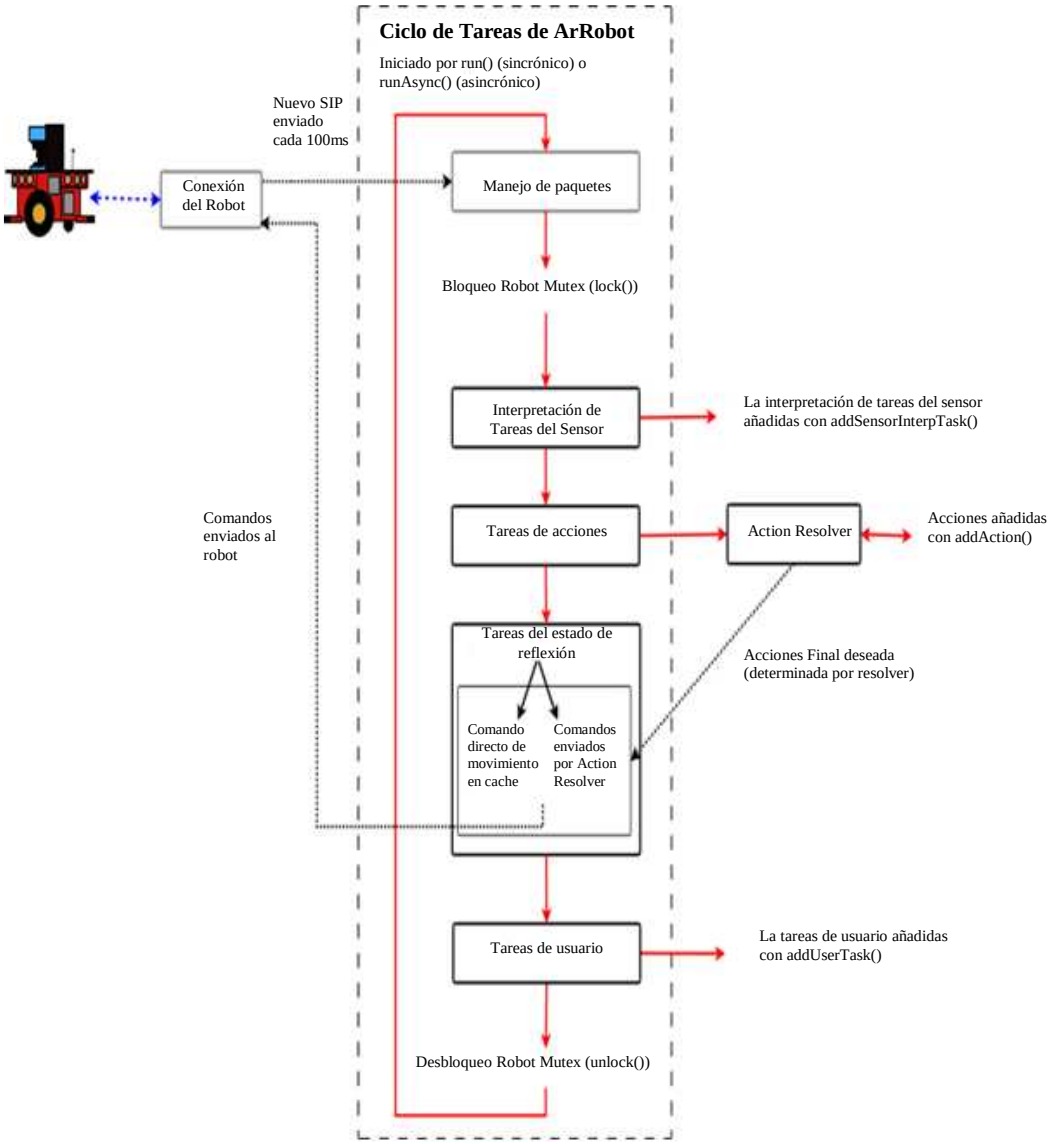

Figura. 5.6. **Ciclo de trabajo de ArRobot**

ArRobot proporciona métodos para añadir su propia interpretación de sensores y tareas genéricas de servicios repetidos del usuario.

Para añadir una tarea de servicio repetido, se debe crear un objeto función ArFunctor, y luego se añade usando ArRobot::addSensorInterpTask() o ArRobot::addUserTask(). Estas tareas pueden ser removidas usando ArRobot::remSensorInterpTask() o ArRobot::remUserTask().

Si se desea también es posible ejecutar el ciclo de procesamiento sin la conexión a un robot. Este ciclo alterno tiene su propio tiempo determinado de 100 milisegundos por ciclo, el cual puede ser examinado o reseteado con ArRobot::getCycleTime() y ArRobot::setCycleTime(). Durante el ciclo normal sincronizado, ArRobot espera hasta dos veces el ciclo de tiempo para un SIP estándar desde el robot, antes de comenzar el ciclo de forma automática.

También se puede desvincular el ciclo de procesamiento del ArRobot del procesamiento del SIP proveniente, si se llama al ArRobot::setCycleChained() (“Chained” significa que si es el final de un ciclo anterior, esto pone en marcha el siguiente después de una espera adecuada para coincidir con la frecuencia de ciclo deseada). Sin embargo, al hacerlo, se puede degradar el rendimiento, debido a que el ciclo del robot se ejecutará cada ArRobot::getCycleTime() milisegundos, y cada vez que la lectura más reciente del SIP es usada (incluso si el robot ha enviado más de uno desde el último ciclo).

La tarea de Sincronización del ArRobot es implementada como un árbol, con cinco ramas grandes. Aunque es poco frecuente que lo haga, un programa cliente puede modificar este árbol o desactivar funciones en las ramas. Si una tarea en particular es deshabilitada, ninguno de sus hijos podrá ser llamado. La raíz de la tarea del árbol puede ser obtenida al llamar a ArRobot::getSyncTaskRoot(), el cual devuelve un objeto ArSyncTask.

5.1.12. Estado de reflexión

El estado de reflexión en la clase ArRobot es la manera en que ARIA mantiene una imagen de las condiciones de operación y valores del robot, tales como posición estimada, velocidad actual, voltaje de la batería, etc. que son extraídos del último SIP estándar.

Los métodos de ArRobot para examinar estos valores incluyen:
ArRobot::getPose(), ArRobot::GetX(), ArRobot::GetY(), ArRobot::GetTh(), ArRobot::GetVel(), ArRobot::GetRotVel(), ArRobot::GetBatteryVoltage(), ArRobot::isLeftMotorStalled(), ArRobot::isRightMotorStalled(), ArRobot::getCompass(), ArRobot::getAnalogPortSelected(), ArRobot::getAnalog(), ArRobot::getDigIn(), ArRobot::getDigOut().

El SIP estándar también contiene las actualizaciones de la lecturas de los sonares, que se reflejan en ArRobot y son examinadas con los métodos: ArRobot::getNumSonar (), ArRobot::getSonarRange (), ArRobot::isSonarNew (), ArRobot::getSonarReading (), ArRobot::getClosestSonarRange (), ArRobot::getClosestSonarNumber (). La clase de la interfaz del sonar, ArSonarDevice, también recibe esta información.

ArRobot también utiliza la tarea del estado de reflexión para enviar comandos de movimiento requeridos previamente al robot, por lo que los comandos de movimiento enviados al robot reflejan estos valores deseados establecidos en el estado de reflexión de ArRobot mediante acciones o métodos de comandos de movimiento, y también para que el tiempo de espera en el watchdog no expire y deshabilite el robot (si no hay comandos de movimientos establecidos, ArCommands::PULSE es enviado cada ciclo).

Se puede ajustar aún más el estado de reflexión de los comandos de movimiento enviando la tasa de repetición si es necesario mediante ArRobot::setStateReflectionRefreshTime().

Si se desea se puede apagar el estado reflector del control de movimiento en el constructor ArRobot::ArRobot() (se debe establecer el parámetro de doStateReflection a falso).

Esto hará que las funciones de comando de movimiento solo se envíen una vez directamente al robot cuando se les llama, en lugar de almacenar el comando para enviar en cada ciclo.

5.1.13. Servicios repetidos del robot

Hay varios servicios repetitivos invocados por ArRobot en eventos de conexión. Se puede añadirlos o removerlos con las funciones:
ArRobot::addConnectCB(), ArRobot::remConnectCB(), ArRobot::addFailedConnectCB(), ArRobot::remFailedConnectCB(), ArRobot::addDisconnectNormallyCB(), ArRobot::remDisconnectNormallyCB(), ArRobot::addDisconnectOnErrorCB(), ArRobot::remDisconnectOnErrorCB(), ArRobot::addRunExitCB(), ArRobot::remRunExitCB().

5.1.14. Control del robot con Acciones y Comandos

El cliente ARIA puede manejar el robot y ejecutar sus diversos accesorios a través de los comandos Directos de ArRobot, los comandos de Movimiento o a través de Acciones.

5.1.15. Comandos Directos

En el nivel más bajo de acceso al robot se puede enviar paquetes de comandos directamente al robot o a la plataforma de simulación a través de ArRobot.

Los comandos Directos consisten en un número de comando de 1 byte seguido por ninguno o más argumentos, como se lo definió en el sistema operativo del robot ARCOS.

Por ejemplo el comando #4, ENABLE, habilita los motores del robot si va acompañado del argumento 1, y los deshabilita con el argumento 0.

Para los comandos que no tienen argumento se utiliza el ArRobot::com(), como el PULSE; se usa ArRobot::comInt() para un argumento entero de 2 bytes, con signo o sin signo, como el comando ENABLE; el ArRobot::com2Bytes() se usa para comandos que acepten dos bytes individuales como argumentos, como el comando VEL2; y finalmente el ArRobot::comStr() o ArRobot::comStrN() para un argumento tipo cadena de la forma NULL o longitud determinada, respectivamente, como el comando de secuencia POLLING del sonar.

La clase ArCommands contienen un enum con todos los comandos directos; por ejemplo ArCommands::ENABLE. No todos los comandos directos son soportados en todas las plataformas de MobileRobots, aunque los comandos no reconocidos (mal escritos) son ignorados.

Para mayor información de los comandos se puede revisar la sección previa correspondiente al manejo de ARCOS. Para la mayoría de comandos, existe un método en ArRobot que envía el comando, ya sea inmediatamente, o lo almacena para la tarea del estado de reflexión a enviar.

Sin embargo, el método de comandos directos permite enviar cualquier comando inusual o especial directamente a la plataforma robótica o al simulador, sin ningún proceso de intervención.

5.1.16. Funciones de comandos de movimiento

Este tipo de comandos se ubican un nivel por encima de los comandos directos. Estos son simples comandos de movimiento explícitos enviados por la tarea del estado de reflexión del ArRobot.

Por ejemplo ArRobot::setVel() para establecer la velocidad de traslación, ArRobot::serRotVel() para establecer la velocidad de rotación, ArRobot::setVel2() para establecer la velocidad de cada rueda por separado, ArRobot::setHeading() para establecer un ángulo de giro de la cabecera, ArRobot::move() para manejar una distancia medida o ArRobot::stop() para detener todo movimiento.

ArRobot también provee métodos para establecer límites de velocidad, más allá de los ya establecidos en el firmware. Estas funciones de movimiento funcionan independientemente o con el estado de reflexión, y ArRobot puede reenviar comandos cada ciclo para tratar de alcanzar el estado deseado.

Se debe tener en cuenta que un comando Directo o de Movimiento puede entrar en conflicto con los controles de las acciones y otros procesos de nivel superior y dar lugar a consecuencias inesperadas. Para cancelar el efecto de un comando de movimiento y su efecto posterior se usa el comando ArRobot::clearDirectMotion() para que la Acción sea capaz de retomar el control del robot.

El limitar el tiempo del comando de Movimiento previene otras Acciones de movimiento y se puede hacer mediante ArRobot::setDirectMotionPrecedenceTime(). En caso contrario, el comando de movimiento puede prevenir las Acciones para siempre. Para ver el tiempo en el cual el comando de movimiento tiene prioridad una vez que se ha establecido se puede utilizar ArRobot::getDirectMotionPrecedenceTime().

5.1.17. Acciones

Si bien las secuencias simples de comandos de movimientos pueden ser fáciles de usar, al tratar de alcanzar movimientos más sofisticados solo con comandos de movimiento puede tornarse un poco difícil.

Para hacer esto posible y definir comportamientos complejos fuera de componentes independientemente utilizables, ARIA provee un sistema de Acciones de Alto nivel.

Las Acciones son objetos individuales que independientemente proveen requerimientos de movimiento, los cuales son evaluados y luego combinados cada ciclo para producir comandos de movimiento establecidos al final de este proceso.

Esto permite construir comportamientos complejos a partir de la construcción de bloques simples para un control de movimiento dinámico y continuo.

Las Acciones se definen mediante la creación de sub-Clases a partir de la clase base ArAction la cual sobrecarga le método ArAction::fire(). ARIA también incluye algunas clases de acciones previamente hechas, las cuales se analizarán a continuación:

5.1.17.1. ArActionAvoidFront

Esta acción controla la evasión de obstáculos, controlando la rotación y la traslación. Esta acción utiliza cualquier dispositivo de rango disponible y que haya sido añadido al robot para la evasión de obstáculos. Para ver los parámetros utilizados se verá a continuación el constructor ArActionVoidFront de las tablas 5.1, 5.2 y la figura 5.7.

Esta acción también realiza tareas que otras no lo hacen como por ejemplo el chequeo de un dispositivo específico de hardware.

Para ello utiliza el tableSensingIR, el cual si se encuentra establecido en los parámetros del robot, usa los puertos DigIn0 y DigIn1 en donde se encuentra conectado el tableSensingIR. Si el TableIRIfAvail se encuentra deshabilitado el constructor ignorará la acción.

	ArActionAvoidFront (const char *name="avoid front obstacles", double obstacleDistance=450, double avoidVelocity=200, double turnAmount=15, bool useTableIRIfAvail=true) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionAvoidFront () Destructor.

Tabla. 5.1. **Funciones Miembro Públicas ArAtionAvoidFront**

Double	myAvoidVel
ArFunctorC< ArActionAvoidFront >	myConnectCB
ArActionDesired	myDesired
Double	myObsDist
ArSectors	myQuadrants
Double	myTurnAmount
Double	myTurnAmountParam
Int	myTurning
Bool	myUseTableIRIfAvail

Tabla. 5.2. **Atributos Protegidos ArAtionAvoidFront**

```
ArActionAvoidFront::ArActionAvoidFront (const char * name = "avoid front obstacles",
                                        double obstacleDistance = 450,
                                        double avoidVelocity = 200,
                                        double turnAmount = 15,
                                        bool useTableIRIfAvail = true
                                        )
```

Figura. 5.7. **Documentación de Constructores y Destructores ArAtionAvoidFront**

Parámetros ArActionAvoidFront

Name	El nombre de la acción.
obstacleDistance	Distancia a la cual girar para empezar la evasión (mm)
avoidVelocity	Velocidad seteada mientras evade un obstáculo (mm/seg.)
turnAmount	Grados que debe girar mientras detecte un obstáculo.
useTableIRIfAvail	Uso de la tabla de sensores IR si está disponible.

5.1.17.2. ArActionAvoidSide

Acción para evitar impactos en paredes ubicadas en ángulo con el robot. Esta acción verifica los sensores para saber si se encuentra cerca de una pared en un ángulo que otras

acciones de evasión no pueden detectar. Ver los parámetros de las tablas 5.3, 5.4 y la figura 5.8.

	ArActionAvoidSide (const char *name="Avoid side", double obstacleDistance=300, double turnAmount=5) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionAvoidSide () Destructor.

Tabla. 5.3. **Funciones Miembro Públicas ArActionAvoidSide**

ArActionDesired	myDesired
Double	myObsDist
Double	myTurnAmount
Bool	myTurning

Tabla. 5.4. **Atributos Protegidos ArActionAvoidSide**

```
ArActionAvoidSide::ArActionAvoidSide ( const char * name = "Avoid side",
                                       double obstacleDistance = 300,
                                              double turnAmount = 5
                                       )
```

Figura. 5.8. **Documentación de Constructores y Destructores ArActionAvoidSide**

Parámetros ArActionAvoidSide

Name	El nombre de la acción.
obstacleDistance	Distancia a la cual girar para empezar la evasión (mm)
turnAmount	Grados que debe girar mientras detecte un obstáculo.

5.1.17.3. ArActionBumpers

Acción que toma medidas en caso de que se disparen los parachoques. Esta clase básicamente responde a los parachoques que posee el robot, la actividad de las cosas que debe hacer que se deciden en el archivo de parámetros. Si el robot va hacia delante y el robot choca con el parachoques delantero, se regresa y gira. Si el robot va hacia atrás y chocan sus parachoques el robot camina hacia delante y gira. Ver los parámetros de las tablas 5.5, 5.6 y la figura 5.9.

	ArActionBumpers (const char *name="bumpers", double backOffSpeed=100, int backOffTime=3000, int turnTime=3000, bool setMaximums=false) Constructor.
Double	findDegreesToTurn (int bumpValue, int whichBumper)
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionBumpers () Destructor.

Tabla. 5.5. **Funciones Miembro Públicas ArActionBumpers**

Double	myBackOffSpeed
Int	myBackOffTime
Int	myBumpMask
ArActionDesired	myDesired
Bool	myFiring
Double	myHeading
Bool	mySetMaximums
Double	mySpeed
ArTime	myStartBack
Int	myTurnTime

Tabla. 5.6. **Atributos Protegidos ArActionBumpers**

```
ArActionBumpers::ArActionBumpers (const char * name = "bumpers",
                                  double backOffSpeed = 100,
                                  int backOffTime = 3000,
                                  int turnTime = 3000,
                                  bool setMaximums = false
                                  )
```

Figura. 5.9. **Documentación de Constructores y Destructores ArActionBumpers**

Parámetros ArActionBumpers

Name	El nombre de la acción.
backOffSpeed	Velocidad a la cual regresa (mm/seg.)
backOffTime	Número de milisegundos para regresar
turnTime	Número de milisegundos para permitir el giro.
setMaximums	Si esta activada utiliza la velocidad límite en el regreso.

5.1.17.4. ArActionColorFollow

Es una acción que mueve el robot a través de una larga trayectoria que aparece en el campo de visión actual. Ver los parámetros de las tablas 5.7, 5.8 y 5.9.

Enum	LocationState { **LEFT**, **RIGHT**, **CENTER** }
Enum	MoveState { **FOLLOWING**, **ACQUIRING**, **STOPPED** }
Enum	TargetState { **NO_TARGET**, **TARGET** }

Tabla. 5.7. **Tipos Públicos ArActionColorFollow**

	ArActionColorFollow (const char *name, ArACTS_1_2 *acts, ArPTZ *camera, double speed=200, int width=160, int height=120)
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
Bool	getAcquire ()
Bool	getBlob ()
Int	getChannel ()
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Bool	getMovement ()
Void	setAcquire (bool acquire)
Void	setCamera (ArPTZ*camera)
Bool	setChannel (int channel)
Void	startMovement (void)
Void	stopMovement (void)
Virtual	~ArActionColorFollow (void)

Tabla. 5.8. **Funciones Miembro Públicas ArActionColorFollow**

Bool	killMovement
Bool	myAcquire
ArACTS_1_2*	myActs
ArPTZ*	myCamera
Int	myChannel
ArActionDesired	myDesired
Int	myHeight
ArTime	myLastSeen
LocationState	myLocation
Int	myMaxTime
MoveState	myMove
Double	mySpeed
TargetState	myState
Int	myWidth

Tabla. 5.9. **Atributos Protegidos ArActionColorFollow**

5.1.17.5. ArActionConstantVelocity

Es una acción para ir directamente a una velocidad constante. Ver los parámetros de las tablas 5.10, 5.11 y la figura 5.10.

Parámetros ArActionConstantVelocity

Name El nombre de la acción.

Velocity Velocidad a la que se desplaza (mm/seg.)

	ArActionConstantVelocity (const char *name="Constant Velocity", double velocity=400) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionConstantVelocity () Destructor.

Tabla. 5.10. **Funciones Miembro Públicas ArActionConstantVelocity**

ArActionDesired	myDesired
Double	myVelocity

Tabla. 5.11. **Atributos Protegidos ArActionConstantVelocity**

Documentación de Constructores y Destructores

ArActionConstantVelocity::ArActionConstantVelocity (const char * *name* = `"Constant Velocity"`,
double *velocity* = `400`
)

Figura. 5.10. **Documentación de Constructores y Destructores ArActionConstantVelocity**

5.1.17.6. ArActionDeceleratingLimiter

Es una acción para limitar el movimiento del robot hacia delante basado en lecturas del sensor. Esta acción usa los sensores de rango para encontrar una velocidad a la cual desplazarse y pueda incrementar su desaceleración de tal manera que el robot no choque con nada, si encuentra un obstáculo realiza una para a fin de evitar una colisión. Se puede utilizar esta acción en la parte baja de la lista de prioridades, de esta manera funcionara mejor. Ver los parámetros de las tablas 5.12, 5.13 y la figura 5.11 y 5.12.

Parámetros ArActionDeceleratingLimiter

Name El nombre de la acción.

Fordwars Si hay una acción hacia delante (true); hacia atrás (false)

Establecer los parámetros (No se usa si está funcionando el addToConfig)

Parámetros ArActionDeceleratingLimiter

Clearance Distancia a la cual realiza la estop (mm).

sideClearanceAtSlowSpeed Distancia a la cual se detiene con respecto al lado si se encuentra viajando a velocidad baja (mm)

paddingAtSlowSpeed	Distancia adicional a la separación a la cual debe parar a velocidad baja (mm).
slowSpeed	Velocidad considerada “baja” (mm/seg.).
sideClearanceAtFastSpeed	Distancia a la cual se detiene con respecto al lado, si se encuentra viajando a velocidad alta (mm)
paddingAtFastSpeed	Distancia adicional a la separación a la cual debe parar a velocidad alta (mm).
fastSpeed	Velocidad considerada “alta” (mm/seg.)
preferredDecel	La máxima desaceleración baja para obstáculos (a menos que se mantenga una separación suficiente, para desacelerar rápido).
useEStop	Si se detecta algo sin la separación se produce una parada de emergencia.
maxEmergencyDecel	Último límite de desaceleración que se aplica cuando se detecta un obstáculo sin la separación suficiente (mm/seg2.). Si es 0 se usa la máxima desaceleración del robot.

Void	addToConfig (ArConfig *config, const char *section, const char *prefix=NULL) Añadir con prefijo en la sección de ArConfig
	ArActionDeceleratingLimiter (const char *name="limitAndDecel", bool forwards=true) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Bool	getForwards (void) Obtiene si se controla cuando va hacia adelante
Bool	getUseLocationDependentDevices (void) Se establece si se usa dispositivos de rango dependientes o no de la localización.
Void	setForwards (bool fordwars) Se establece si se controla al ir hacia adelante
Void	setParameters (double clearance=100, double sideClearanceAtSlowSpeed=50, double paddingAtSlowSpeed=50, double slowSpeed=200, double sideClearanceAtFastSpeed=400, double paddingAtFastSpeed=300, double fastSpeed=1000, double preferredDecel=600, bool useEStop=false, double maxEmergencyDecel=0) Establece los parámetros (No se debe usar si se usa addToConfig).
Void	setUseLocationDependentDevices (void) Se establece si se usa dispositivos de rango dependientes o no de la localización.
Virtual	~ArActionDeceleratingLimiter () Destructor.

Tabla. 5.12. **Funciones Miembro Públicas ArActionDeceleratingLimiter**

Double	myClearance
ArActionDesired	myDesired
Double	myFastSpeed
Bool	myFordwars
Bool	myLastStopped
Double	myMaxEmergencyDecel
Double	myPaddingAtFastSpeed
Double	myPaddingAtSlowSpeed
Double	myPreferredDecel
Double	mySideClearanceAtFastSpeed
Double	mySideClearanceAtSlowSpeed
Double	mySlowSpeed
Bool	myUseEStop
Bool	myUseLocationDependentDevices

Tabla. 5.13. **Atributos Protegidos ArActionDeceleratingLimiter**

```
ArActionDeceleratingLimiter::ArActionDeceleratingLimiter ( const char *  name =
"limitAndDecel",
                                                           bool  forwards = true)
```

Figura. 5.11. **Documentación de Constructores y Destructores ArActionDeceleratingLimiter**

```
void ArActionDeceleratingLimiter::setParameters ( double  clearance = 100,
                                                  double  sideClearanceAtSlowSpeed = 50,
                                                  double  paddingAtSlowSpeed = 50,
                                                  double  slowSpeed = 200,
                                                  double  sideClearanceAtFastSpeed = 400,
                                                  double  paddingAtFastSpeed = 300,
                                                  double  fastSpeed = 1000,
                                                  double  preferredDecel = 600,
                                                  bool  useEStop = false,
                                                  double  maxEmergencyDecel = 0
                                                  )
```

Figura. 5.12. **Documentación de la Función Miembro ArActionDeceleratingLimiter**

5.1.17.7. ArActionDriveDistance

La acción va al ArPose, maneja una distancia arreglada y para el robot cuando ha recorrido la distancia apropiada. Se puede establecer la distancia con setDistance(), para cancelar este movimiento se usa cancelDistance() y para saber si se ha llegado al punto deseado con haveAchievedDistance().Ver los parámetros de las tablas 5.14, 5.15 y 5.16.

Se puede decir al robot que camine hacia atrás si se establece una distancia negativa. Esta acción no evade obstáculos ni nada, pero se puede añadir una acción ArAction para ello estableciéndola con una prioridad mayor.

	ArActionDriveDistance (const char *name="driveDistance", double speed=400, double deceleration=200)
Void	cancelDistance (void) Cancela la meta que el robot tiene.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
Double	getDeceleration (void) Obtiene la desaceleración que la acción utilizará (mm/seg2.)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Double	getSpeed (void) Obtiene la velocidad que la acción utilizará (mm/seg.)
Bool	haveAchievedDistance (void) Mira si la meta ha sido alcanzada
Void	setDeceleration (double deceleration = 200) Establece la desaceleración que la acción utilizará (mm/seg2.)
Void	setDistance (double distance, bool useEncoders = true) Establece una nueva meta y establece a la acción para que vaya allí.
Void	setPrinting (bool printing) Se establece si se está imprimiendo o no.
Void	setSpeed (double Speed = 400) Establece la velocidad que la acción utilizará (mm/seg.)
Bool	usingEncoders (void) Obtiene al usar los codificadores su posición o la posición normal
Virtual	~ArActionDriveDistance ()

Tabla. 5.14. **Funciones Miembro Públicas ArActionDriveDistance**

Enum	State { **STATE_NO_DISTANCE**, **STATE_ACHIEVED_DISTANCE**, **STATE_GOING_DISTANCE** }

Tabla. 5.15. **Tipos Protegidos ArActionDriveDistance**

Double	myDeceleration
ArActionDesired	myDesired
Double	myDistance
Double	myDistTravelled
ArPose	myLastPose
Double	myLastVel
Bool	myPrinting
Double	mySpeed
State	myState
Bool	myUseEncoders

Tabla. 5.16. **Atributos Protegidos ArActionDriveDistance**

5.1.17.8. ArActionGoto

La acción va al ArPose, maneja una posición establecida y para el robot cuando ha recorrido la distancia apropiada. Se puede establecer una nueva meta con setGoal(), para cancelar este movimiento se usa cancelGoal() y para saber si se ha llegado a la meta deseada con haveAchievedGoal().Ver los parámetros de las tablas 5.17, 5.18 y 5.19 y la figura 5.13.

Una vez que la meta ha sido alcanzada la acción para cualquier requerimiento hecho. Esta acción no evade obstáculos ni nada, pero se puede añadir una acción ArAction para ello estableciéndola con una prioridad mayor.

	ArActionGoto (const char *name="goto", ArPose goal=ArPose(0.0, 0.0, 0.0), double closeDist=100, double speed=400, double speedToTurnAt=150, double turnAmount=7)
Void	cancelGoal (void) Cancela la meta, anulando cualquier movimiento.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
Double	getCloseDist (void) Obtiene la distancia la cual esta suficientemente cerca de la meta (mm)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
ArPose	getGoal (void) Obtiene la meta la acción tiene
Double	getSpeed (void) Obtiene la velocidad que la acción utilizará (mm/seg.)
Bool	haveAchievedGoal (void)
Void	setCloseDist (double closeDist) Establece la distancia la cual esta suficientemente cerca de la meta (mm)
Void	setGoal (ArPose goal) Establece una nueva meta y establece a la acción ir allí.
Void	setSpeed (double Speed) Establece la velocidad que la acción utilizará (mm/seg.)
Virtual	~ArActionGoto ()

Tabla. 5.17. **Funciones Miembro Públicas ArActionGoto**

Enum	State { **STATE_NO_GOAL**, **STATE_ACHIEVED_GOAL**, **STATE_GOING_TO_GOAL** }

Tabla. 5.18. **Tipos Protegidos ArActionGoto**

Double	myCloseDist
Double	myCurTurnDir
ArActionDesired	myDesired
Double	myDirectionToTurn
ArPose	myGoal
ArPose	myOldGoal
Bool	myPrinting
Double	mySpeed
Double	mySpeedToTurnAt
State	myState
Double	myTurnAmount
Bool	myTurnedBack

Tabla. 5.19. **Atributos Protegidos ArActionGoto**

ArActionDesired * ArActionGoto::fire (ArActionDesired *currentDesired*) [virtual]

Figura. 5.13. **Documentación de Constructores y Destructores de la Función Miembro ArActionGoto**

Llamada por la acción resolver, requiere moverse hacia una meta si existe.

Parámetros ArActionGoto

currentDesired Acción actual deseada desde el resolver.

virtual ArActionDesired* ArActionGoto::getDesired (void) [inline,virtual]

Usada por la acción resolver, retorna la acción actual deseada.

bool ArActionGoto::haveAchievedGoal (void)

Mira si la meta ha sido alcanzada. Esto se logra cuando el robot ha reportado la posición más cercana a la posición de la meta.

5.1.17.9. ArActionGotoStraight

La acción va al ArPose, maneja directo hacia una posición establecida y para el robot cuando ha recorrido la distancia apropiada.

Se puede establecer una nueva posición de meta con setGoal(), para cancelar este movimiento se usa cancelGoal() y para saber si se ha llegado a la meta deseada con haveAchievedGoal().

Para los argumentos de las metas y metas de los codificadores, se las puede hacer que vuelvan poniendo el parámetro backwards en trae, si se pone el parámetro justDistance en trae solo tendrá en cuenta el recorrer la distancia pedida, si se encuentra en falso tratará de llegar lo más cerca de la meta pedida.

Esta acción no evade obstáculos ni nada, pero se puede añadir una acción ArAction para ello estableciéndola con una prioridad mayor.

Ver los parámetros de las tablas 5.20, 5.21 y 5.22.

	ArActionGotoStraight (const char *name="goto", double speed=400)
Void	cancelGoal (void) Cancela la meta que tiene el robot.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
Bool	getBacking (void) Establece si se respalda o no el lugar (Establecido en las metas)
Doublé	getCloseDist (void) Obtiene cuan cerca debe llegar si es que no se encuentra habilitado el modo just distance.
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
ArPose	getEncoderGoal (void) Obtiene la meta que la acción tiene.
ArPose	getGoal (void) Obtiene la meta la acción tiene.
Double	getSpeed (void) Obtiene la velocidad que la acción utilizará (mm/seg.)
Bool	haveAchievedGoal (void) Mira si la meta ha sido alcanzada
Void	setCloseDist (double closeDist=100) Establece la distancia la cual esta suficientemente cerca de la meta si no esta habilitado el modo just distance.
Void	setEncoderGoal (ArPose EncoderGoal, bool backwards = false, bool justDistance = true) Establece una nueva meta y pide a la acción ir allí.
Void	setGoal (ArPose goal, bool backwards = false, bool justDistance = true) Establece una nueva meta y establece a la acción ir allí.
Void	setGoalRel (double dist, double deltaHeading, bool backwards = false, bool justDistance = true) Establece la meta de manera relative.
Void	setSpeed (double Speed) Establece la velocidad que la acción utilizará (mm/seg.)
Bool	usingEncoderGoal (void) Obtienen si se está usando la meta normal o la meta con codificadores.
Virtual	~ArActionGotoStraight ()

Tabla. 5.20. **Funciones Miembro Públicas ArActionGotoStraight**

Enum	State { **STATE_NO_GOAL**, **STATE_ACHIEVED_GOAL**, **STATE_GOING_TO_GOAL** }

Tabla. 5.21**. Tipos Protegidos ArActionGotoStraight**

Bool	myBacking
Double	myCloseDist
ArActionDesired	myDesired
Double	myDist
Double	myDistTravelled
ArPose	myEncoderGoal
ArPose	myGoal
Bool	myJustDist
ArPose	myLastPose
Bool	myPrinting
Double	mySpeed
State	myState
Bool	myUseEncoderGoal

Tabla. 5.22. **Atributos Protegidos ArActionGotoStraight**

5.1.17.10. ArActionInput

Acción que toma una entrada desde una salida para control del robot. Esta acción establece como se quiere conducir. Ver los parámetros de las tablas 5.23, 5.24 y 5.25 y la figura 5.14.

	ArActionInput (const char *name="Input") Constructor.
Void	clear (void) Borra lo que no utiliza de la velocidad o giro.
void	deltaHeadingFromCurrent (double delta) Incrementa o decrementa el giro actual.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Void	setHeading (double heading) Establece la partida.
Void	setRotVel (double rotVel) Establece una velocidad rotacional.
Void	setVel (double Vel) Establece una velocidad (cancela deltaVel)
Virtual	~ArActionInput () Destructor.

Tabla. 5.23. **Funciones Miembro Públicas ArActionInput**

Enum	RotRegime { **NONE**, **ROTVEL**, **DELTAHEADING**, **SETHEADING** }

Tabla. 5.24. **Tipos Protegidos ArActionInput**

ArActionDesired	myDesired
RotRegime	myRotRegime
Double	myRotVal
Bool	myUsingVel
Double	myVelSet

Tabla. 5.25. **Atributos Protegidos ArActionInput**

```
ArActionInput::ArActionInput ( const char *  name = "Input" )
```

Figura. 5.14. **Documentación de Constructores y Destructores de la Función Miembro ArActionInput**

Parámetros ArActionInput

Name Nombre de la acción.

5.1.17.11. ArActionIRs

Acción para hacer respaldos de las activaciones de los sensores de corto rango IRs. Si el robot tiene sensores infrarrojos binarios montados en el frente, esta acción responderá a la

activación de uno de los sensores, debido a giros u otros objetos tal como lo hacen los parachoques. Esta acción asume que si un sensor infrarrojo es activado, debido al movimiento hacia delante y su encuentro con un objeto, deberá retroceder para una buena reacción. Ver los parámetros de las tablas 5.26, y 5.27 y la figura 5.15.

	ArActionIRs (const char *name="IRs", double backOffSpeed=100, int backOffTime=5000, int turnTime=3000, bool setMaximums=false) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual void	setRobot (ArRobot *robot)
virtual	~ArActionIRs () Destructor.

Tabla. 5.26. **Funciones Miembro Públicas ArActionIRs**

std::vector < int >	cycleCounters
Double	myBackOffSpeed
Int	myBackOffTime
ArActionDesired	myDesired
Bool	myFiring
Double	Myheading
ArRobotParams	myParams
Bool	mySetMaximums
Double	mySpeed
ArTime	myStartback
Int	myStopTime
Int	myTurnTime
ArTime	stoppedSince

Tabla. 5.27. **Atributos Protegidos ArActionIRs**

```
ArActionIRs::ArActionIRs ( const char *  name = "IRs",
                           double  backOffSpeed = 100,
                           int  backOffTime = 5000,
                           int  turnTime = 3000,
                           bool  setMaximums = false
                           )
```

Figura. 5.15. **Documentación de Constructores y Destructores de la Función Miembro ArActionIRs**

Parámetros ArActionIRs

Name	El nombre de la acción.
backOffSpeed	Velocidad a la cual regresa (mm/seg.)
backOffTime	Número de milisegundos para regresar.
turnTime	Número de milisegundos permitidos para el giro.

setMaximums	Si esta habilitado, establece la máxima velocidad deseada cuando backOffSpeed está realizando la acción, de otra manera utiliza los límites existentes.

5.1.17.12. ArActionJoydrive

Esta acción usa el joystick como entrada para manejo del robot. Esta clase crea su propio ArJoyHandler para obtener una entrada desde el joystick. Posteriormente toma una escala entre el 0 y el máximo para velocidades de giro y traslación, arriba y abajo en el joystick sirven para ir hacia delante y hacia atrás respectivamente, mientras que para ir a los lados sirven la dirección derecha e izquierda. Se debe pulsar uno de los botones del joystick para que la clase le preste atención. Ver los parámetros de las tablas 5.28, y 5.29 y la figura 5.16.

Nota:

El joystick no guarda la información de la calibración, por eso se debe calibrar el joystick cada vez que se vaya a usar. Para hacer esto, se debe presionar el botón al menos por medio segundo cuando la palanca está en el centro, luego se debe soltar el botón, y poner la palanca en la parte superior izquierda por al menos medio segundo y luego llevarla a la esquina inferior derecha por al menos medio segundo.

Esta acción es para la conducción de todo el robot con el joystick, para ello se debe mantener presionado el botón y luego manejar con la palanca hacia donde quiera moverse el robot. Antes de ello se debe calibrar el joystick para un correcto funcionamiento.

Parámetros ArActionJoydrive

Name	El nombre de la acción.
TransVelMax	La máxima velocidad a la cual la acción joydrive irá, y que alcanza cuando el joystick se encuentre todo hacia delante.
turnAmountMax	La máxima cantidad de giro que la acción realizará.
stopIfNoButtonPressed	Si es true, y no se ha presionado el botón, el robot no se moverá, de otro modo no hará nada.

useOSCalForJoystick Si esta como true, se usará la calibración del sistema operativo, de otro modo se usará la calibración del usuario.

	ArActionJoydrive (const char *name="joydrive", double transVelMax=400, double turnAmountMax=15, bool stopIfNoButtonPressed=true, bool useOSCalForJoystick=true) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
ArJoyHandler	getJoyHandler (void) Obtiene el JoyHandler.
Bool	getStopIfNoButtonPressed (void) Obtiene si no se ha presionado el botón de stop, de otra manera no hace nada.
Bool	getUseOSCal (void) Obtiene si se está usando la calibración del Sistema Operativo o no.
Bool	jostickInited (void) Si el joystick está inicializado o no.
Void	setSpeeds (double transVelMax, double turnAmountMax) Establece velocidades
Void	setStopIfNoButtonPressed (bool stopIfNoButtonPressed) Establece si no se ha presionado el botón de stop, de otra manera no hace nada.
Void	setThrottleParams (double lowSpeed, double highSpeed) Establece los parámetros sobre el acelerador (El acelerador no se usa a menos que se lo llame).
Void	setUseOSCal (bool UseOSCal) Establece si se usa la calibración del Sistema Operativo o no.
Virtual	~ArActionJoydrive () Destructor.

Tabla. 5.28. **Funciones Miembro Públicas ArActionJoydrive**

ArActionDesired	myDesired
Double	myHighThrottle
ArJoyHandler*	myJoyHandler
Double	myLowThrottle
Bool	myPreviousUseOSCal
Bool	myStopIfNoButtonPressed
Double	myTransVelMax
Double	myTurnAmountMax
Bool	myUseOSCal
Bool	myUseThrottle

Tabla. 5.29**. Atributos Protegidos ArActionJoydrive**

```
ArActionJoydrive::ArActionJoydrive ( const char *  name = "joydrive",
                                     double  transVelMax = 400,
                                     double  turnAmountMax = 15,
                                                  bool  stopIfNoButtonPressed = true,
                                     bool  useOSCalForJoystick = true
                                     )
```

Figura. 5.16. **Documentación de Constructores y Destructores de la Función Miembro ArActionJoydrive**

bool ArActionJoydrive::getUseOSCal (void)

Obtiene cuando la calibración del sistema operativo esta siendo utilizada por el joystick.

void ArActionJoydrive::setUseOSCal (bool *useOSCal*)

Establece cuando la calibración del sistema operativo esta siendo utilizada por el joystick.

5.1.17.13. ArActionKeydrive

La acción utilizará las flechas del teclado como entrada para el manejo del robot. Ver los parámetros de las tablas 5.30, y 5.31.

Virtual void	activate (void)
	ArActionKeydrive (const char *name="keydrive", double transVelMax=400, double turnAmountMax=24, double velIncrement=25, double turnIncrement=8) Constructor.
Virtual void	desactivate (void)
Void	down (void) Internamente, llama a la flecha de abajo.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Void	giveUpKeys (void) Da a la acción las teclas que utilizará para manejar.
Void	left (void) Internamente, llama a la flecha izquierda.
Void	right (void) Internamente, llama a la flecha derecha.
Void	setIncrements (double velIncrement, double turnIncrement) Establece la cantidad de incrementos.
Virtual void	setRobot (ArRobot *robot)
Void	setSpeeds (double transVelMax, double turnAmountMax) Establece las velocidades máximas
Void	space (void) Internamente, llama a la tecla espacio.
Void	takeKeys (void) Toma las teclas que la acción quiere usar para el manejo.
Void	up (void) Internamente, llama a la flecha arriba.
Virtual	~ArActionKeydrive () Destructor.

Tabla. 5.30. **Funciones Miembro Públicas ArActionKeydrive**

Double	myDeltaVel
ArActionDesired	myDesired
Double	myDesiredSpeed
ArFunctorC< ArActionKeyDrive>	myDownCB
ArFunctorC< ArActionKeyDrive>	myLeftCB
ArFunctorC< ArActionKeyDrive>	myRightCB
ArFunctorC< ArActionKeyDrive>	mySpaceCB
Bool	mySpeedReset
Double	myTransVelMax
Double	myTurnAmount
Double	myTurnAmountMax
Double	myTurnIncrement
ArFunctorC< ArActionKeyDrive>	myUpCB
Double	myVelIncrement

Tabla. 5.31. **Atributos Protegidos ArActionKeydrive**

5.1.17.14. ArActionLimiterBackwards

Acción para limitar el movimiento hacia atrás, basado en la lectura de los sensores. Esta clase limita los movimientos hacia atrás, de acuerdo a las lecturas de los sensores y los parámetros dados. Ver los parámetros de las tablas 5.32, y 5.33 y la figura 5.17.

Cuando el sensor detecta un obstáculo atrás, más cercano a la distancia dada en los parámetros, la acción pide al robot que desacelere o pare cualquier movimiento hacia atrás.

	ArActionLimiterBackwards (const char *name="speed limiter", double stopDistance=-250, double slowDistance=-600, double maxBackwardsSpeed=-250, double widthRatio=1.5) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionLimiterBackwards () Destructor.

Tabla. 5.32. **Funciones Miembro Públicas ArActionBackwards**

ArActionDesired	myDesired
Double	myMaxBackwardsSpeed
Double	mySlowDist
Double	myStopDist
Double	myWidthRatio

Tabla. 5.33. **Atributos Protegidos ArActionBackwards**

Parámetros ArActionBackwards

Name El nombre de la acción.

stopDistance Distancia a la cual parar. (mm).

slowDistance	Distancia a la cual frenar (mm).
maxBackwardsSpeed	Velocidad de reversa máxima, la escala permitida desde esta velocidad hasta 0 para frenar a la distancia (mm/seg).
widthRatio	El radio alrededor del robot en el cual chequea por las lecturas del sensor.

```
ArActionLimiterBackwards::ArActionLimiterBackwards ( const char * name = "speed limiter",
                                                     double stopDistance = -250,
                                                     double slowDistance = -600,
                                                     double maxBackwardsSpeed = -250,
                                                     double widthRatio = 1.5
                                                    )
```

Figura. 5.17. **Documentación de Constructores y Destructores de la Función Miembro ArActionBackwards**

5.1.17.15. ArActionLimiterFordwars

Acción que limita el movimiento del robot hacia delante según las lecturas de los sensores. Esta acción usa los sensores para determinar una velocidad máxima a la cual el robot se desplace hacia delante. Cuando el sensor detecte obstáculos más cerca de lo que los parámetros establecen, la acción pide que el robot desacelere o pare. Ver los parámetros de las tablas 5.34, y 5.35 y la figura 5.18 y 5.19.

	ArActionLimiterForwards (const char *name="speed limiter", double stopDistance=250, double slowDistance=1000, double slowSpeed=200, double widthRatio=1) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Void	setParameters (double stopDistance=250, double slowDistance=1000, double slowSpeed=200, double widthRatio=1)
Virtual	~ArActionLimiterFordwards () Destructor.

Tabla. 5.34. **Funciones Miembro Públicas ArActionLimiterFordwars**

ArActionDesired	myDesired
Bool	myLastStopped
Double	mySlowDist
Double	mySlowSpeed
Double	myStopDist
Double	myWidthRatio

Tabla. 5.35. **Atributos Protegidos ArActionLimiterFordwars**

```
ArActionLimiterForwards::ArActionLimiterForwards ( const char * name = "speed limiter",
                                                    double stopDistance = 250,
                                                    double slowDistance = 1000,
                                                    double slowSpeed = 200,
                                                    double widthRatio = 1
                                                   )
```

Figura. 5.18. **Documentación de Constructores y Destructores de la Función Miembro ArActionLimiterFordwars**

Parámetros ArActionLimiterFordwars

Name El nombre de la acción.

stopDistance Distancia a la cual parar. (mm).

slowDistance Distancia a la cual frenar (mm).

slowSpeed Velocidad permitida en el rango slowDistance, su escala seta permitida desde 0 hasta slowDistance (mm/seg).

widthRatio Relación entre el ancho de la caja a buscar con el radio del robot.

```
ArActionLimiterForwards::ArActionLimiterForwards ( double stopDistance = 250,
                                                    double slowDistance = 1000,
                                                    double slowSpeed = 200,
                                                    double widthRatio = 1
                                                   )
```

Figura. 5.19. **Documentación de la Función Miembro ArActionLimiterFordwars**

5.1.17.16. ArActionLimiterTableSensor

Acción para limitar la velocidad (y parada) en función de lo que la tabla de sensores muestre. Esta acción limita la velocidad del robot a cero si es que la tabla ve algo en frente. La acción solo funciona si el robot tiene tabla de sensores, esto quiere decir que debe estar activado en los parámetros del robot con true. Ver los parámetros de las tablas 5.36, y 5.37.

	ArActionLimiterTableSensor (const char *name="TableSensorLimiter") Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionLimiterTableSensor () Destructor.

Tabla. 5.36. **Funciones Miembro Públicas ArActionLimiterTableSensor**

ArActionDesired	myDesired

Tabla. 5.37. **Atributos Protegidos ArActionLimiterTableSensor**

5.1.17.17. ArActionMovementParameters

Esta clase es para establecer máximos de velocidades, aceleraciones y desaceleraciones. Ver los parámetros de las tablas 5.38, y 5.39 y la figura 5.20.

Void	addToConfig (ArConfig *config, const char *section, const char *prefix=NULL) Se añade al ArConfig dado, en la sección, con prefijo.
	ArActionMovementParameters (const char *name="MovementParameters", bool overrideFaster=true) Constructor.
Void	disable (void) Deshabilita la acción (separado de la desactivación).
Void	enable (void) Habilita la acción (separado de la activación).
Void	enableOnceFromSector (ArMapObject *mapObject) Habilita esta acción de tal manera que trabajará desde el sector de llamadas.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Void	setParameters (double maxVel=0, double maxNegVel=0, double transAccel=0, double transDecel=0, double rotVelMax=0, double rotAccel=0, double rotDecel=0) Establece los parámetros (No se usa si se esta usando el addToConfig).
Virtual	~ArActionMovementParameters () Destructor.

Tabla. 5.38. **Funciones Miembro Públicas ArActionMovementParameters**

ArActionDesired	myDesired
Bool	myEnabled
Bool	myEnabledOnce
Double	myMaxNegVel
Double	myMaxRotVel
Double	myMaxVel
Bool	myOverrideFaster
Double	myRotAccel
Double	myRotDecel
Double	myTransAccel
Double	myTransDecel

Tabla. 5.39. **Atributos Protegidos ArActionMovementParameters**

Parámetros

Name El nombre de la acción.

overrideFaster Si esta en true, aumenta la velocidad máxima establecida a la velocidad máxima total con mayor fuerza.

```
ArActionMovementParameters::ArActionMovementParameters ( const char *  name =
"MovementParameters",
                                                          bool  overrideFaster = true
                                                        )
```

Figura. 5.20. **Documentación de Constructores y Destructores de la Función Miembro ArActionMovementParameters**

5.1.17.18. ArActionRatioInput

Es una acción que pide un movimiento basado en relaciones abstractas dadas por diferentes fuentes de entrada. Esta acción interpreta los comandos de manejo de entrada en forma de tres relaciones abstractas de traslación, rotación y aceleración.

Los rangos de entrada de la velocidad de traslación va desde -100 a 100, en donde -100 significa la velocidad máxima de retroceso y 100 la velocidad máxima hacia delante, mientras que si se sitúa un valor de 0 quiere decir que no hay velocidad de traslación, y los valores intermedios varían en forma lineal con el porcentaje.

De manera similar los rangos de la velocidad de rotación varían entre -100 y 100, donde -100 es la velocidad máxima de rotación en sentido horario y 100 indica el máximo valor de velocidad de rotación máximo en sentido anti horario, mientras que 0 indica que no hay rotación. Las entradas de la aceleración va en el rango de 0 (no se mueve) hasta 100 (máximo movimiento).

Para proveer estas entradas se usan objetos separados (ArRatioInputKeydrive, ArRatioInputJoydrive, ArRAtionInputRobotJoydrive). Cuando esta acción es activada se resetean todas sus relaciones de entrada a 0 incluyendo la aceleración. Los parámetros de configuración son usados para asignar las relaciones máximas a las velocidades actuales del robot.

Estos son establecidos por defecto a las velocidades de configuración máxima del robot en el inicio y se puede anular con los parámetros del ArConfig (y llamar al addToConfig()) o del setParameters().Ver los parámetros de las tablas 5.40, y 5.41 y la figura 5.21, 5.22, 5.23 y 5.24.

Virtual void	activate (void)
	ArActionMovementParameters (const char *name="MovementParameters", bool overrideFaster=true) Constructor.
Void	addActivateCallback (ArFunctor *functor, ArListPos::Pos position=ArListPos::LAST) Añade una devolución de llamada que se llama cuando esta acción es activada.
Void	addDeactivateCallback (ArFunctor *functor, ArListPos::Pos position=ArListPos::LAST) Añade una devolución de llamada que se llama cuando esta acción es desactivada.
Void	addFireCallback (int priority, ArFunctor *functor) Añade una devolución de llamada que se llama desde un disparo de llamada.
Void	addToConfig (ArConfig *config, const char *section) Añade una sección en la configuración.
	ArActionRatioInput (const char *name="RatioInput") Constructor.
Virtual void	desactivate (void)
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Double	getRotRatio (void) Obtiene el radio de rotación (desde -100 (todo a la derecha) hasta 100 (todo a la izquierda).
Double	getTransRatio (void) Obtiene la relación de traslación (desde -100 (retroceso al máximo) hasta 100 (hacia delante al máximo).
Void	remActivateCallback (ArFunctor *functor) Quita una devolución de llamada que fue llamada al activar la acción.
Void	remDeactivateCallback (ArFunctor *functor) Quita una devolución de llamada que fue llamada al desactivar la acción.
Void	remFireCallback (ArFunctor *functor) Quita una devolución de llamada que fue llamada desde una acción disparada por una devolución de llamada.
Void	setParameters (double fullThrottleForwards, double fullThrottleBackwards, double rotAtFullForwards, double rotAtFullBackwards, double rotAtStopped) Establece los parámetros.
Void	setRatios (double transRatio, double rotRatio, double throttleRatio) Establece las relaciones.
Void	setRotRatio (double rotRatio) Establece el radio de rotación (desde -100 (todo a la derecha) hasta 100 (todo a la izquierda).
Void	setThrottleRatio (double throttleRatio) Establece la relación de aceleración (desde 0 (parado) hasta 100 (aceleración total).
Void	setTransRatio (double transRatio) Establece la relación de traslación (desde -100 (retroceso al máximo) hasta 100 (hacia delante al máximo).
Virtual	~ArActionRatioInput () Destructor.

Tabla. 5.40. **Funciones Miembro Públicas ArActionRadioInput**

Parámetros ArActionRadioInput

Name El nombre de la acción.

Std::list< ArFunctor* >	myActivateCallbacks
Std::list< ArFunctor* >	myDesactivateCallbacks
ArActionDesired	myDesired
Std::multimap< int, ArFunctor* >	myFireCallbacks
Double	myFullThrottleBackwards
Double	myFullThrottleForwards
Bool	myPrinting
Double	myRotAtFullBackwards
Double	myRotAtFullForwards
Double	myRotAtStopped
Double	myRotDeadZone
Double	myRotRatio
Double	myThrottleRatio
Double	myTransDeadZone
Double	myTransRatio

Tabla. 5.41. **Atributos Protegidos ArActionRadioInput**

```
ArActionRatioInput::ArActionRatioInput ( const char *  name = "RatioInput" )
```

Figura. 5.21. **Documentación de Constructores y Destructores de la Función Miembro ArActionRadioInput**

```
void ArActionRatioInput::addFireCallback ( int  priority,
                                           ArFunctor *  functor
                                         )
```

Figura. 5.22. **Documentación de la Función Miembro ArActionRadioInput**

Añade una devolución de llamada que es llamada desde la activación de una acción. Estas devoluciones de llamadas son llamadas en el orden del menos número al mayor número, esto significa que los números mayores siguen siendo más importantes en la aceleración ya que los números menores la anularán.

```
void ArActionRatioInput::setParameters ( double  fullThrottleForwards,
                                         double  fullThrottleBackwards,
                                         double  rotAtFullForwards,
                                         double  rotAtFullBackwards,
                                         double  rotAtStopped
                                       )
```

Figura. 5.23. **Seteo de Parámetros de la Función Miembro ArActionRadioInput**

Parámetros ArActionRadioInput

fullThrottleForwards Velocidad a la que se va hacia delante en la máxima aceleración. (mm/seg.).

fullThrottleBackwards	Velocidad a la que se va hacia atrás en la máxima aceleración. (mm/seg.).
rotAtFullForwards	Velocidad a la que gira a la máxima aceleración hacia adelante.
rotAtFullBackwards	Velocidad a la que gira a la máxima aceleración hacia atrás.
rotAtStopped	Velocidad a la que gira si está parado.

```
void ArActionRatioInput::setRatios ( double  transRatio,
                                     double  rotRatio,
                                     double  throttleRatio
                                   )
```

Figura. 5.24. **Seteo de Rangos de la Función Miembro ArActionRadioInput**

Establece relaciones: Comprueba las entradas y las fija en el rango válido.

```
void ArActionRatioInput::setRotRatio ( double  rotRatio  )
```

Establece la relación de rotación (desde -100 (todo a la derecha) hasta 100 (todo a la izquierda). Chequea las entradas para valores mayores a 100 o menores a -100 y los ubica en el rango.

```
void ArActionRatioInput::setThrottleRatio ( double  throttleRatio  )
```

Establece la relación de aceleración (desde 0 (parado) hasta 100 (aceleración total)). Chequea las entradas para valores mayores a 100 o menores a 0 y los ubica en el rango.

```
void ArActionRatioInput::setTransRatio ( double  transRatio  )
```

Establece la relación de traslación (desde -100 (todo a la derecha) hasta 100 (todo a la izquierda).

Chequea las entradas para valores mayores a 100 o menores a -100 y los ubica en el rango.

5.1.17.19. ArActionRobotJoydrive

Esta acción utiliza la entrada del joystick para el manejo del robot. Esta clase crea su propio ArJoyHandler para obtener la entrada del joystick.

Posteriormente toma una escala entre el 0 y el máximo para velocidades de giro y traslación, arriba y abajo en el joystick sirven para ir hacia delante y hacia atrás respectivamente, mientras que para ir a los lados sirven la dirección derecha e izquierda.

Se debe pulsar uno de los botones del joystick para que la clase le preste atención. Ver los parámetros de las tablas 5.42, 5.43 y 5.44 y la figura 5.25.

Nota:

El joystick no guarda la información de la calibración, por eso se debe calibrar el joystick cada vez que se vaya a usar.

Para hacer esto, se debe presionar el botón al menos por medio segundo cuando la palanca está en el centro, luego se debe soltar el botón, y poner la palanca en la parte superior izquierda por al menos medio segundo y luego llevarla a la esquina inferior derecha por al menos medio segundo.

	ArActionRobotJoydrive (const char *name="robotJoyDrive", bool requireDeadmanPushed=true) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual void	setRobot (ArRobot *robot)
Virtual	~ArActionRobotJoydrive () Destructor.

Tabla. 5.42. **Funciones Miembro Públicas ArActionRobotJoydrive**

Void	connectCallback (void)
Bool	handleJoystickPacket (ArRobotPacket *packet)

Tabla. 5.43. **Funciones Miembro Protegidas ArActionRadioInput**

Int	myButton1
Int	myButton2
ArFunctorC<ArActionRobotJoyDrive>	myConnectCB
Bool	myDeadZoneLast
ArActionDesired	myDesired
ArRetFunctor1C< bool, ArActionRobotJoydrive, ArRobotPacket* >	myHandleJoystickPacketCB
Int	myJoyX
Int	myJoyY
ArTime	myPacketReceivedTime
Bool	myRequireDeadmanPushed
Int	myThrottle

Tabla. 5.44. **Atributos Protegidos ArActionRadioInput**

```
ArActionRobotJoydrive::ArActionRobotJoydrive ( const char *  name = "robotJoyDrive",
                                               bool  requireDeadmanPushed = true
                                             )
```

Figura. 5.25. **Documentación de Constructores y Destructores de la Función Miembro ArActionRobotJoydrive**

Parámetros ArActionRobotJoydrive

Name Nombre de la acción.

requireDeadmanPushed Si el se mantiene presionado para manejar, si se encuentra falso se manejará sin importar el estado del botón.

5.1.17.20. ArActionStallRecover

Acción para recuperarse de una parada. La acción trata de recuperar si una de las ruedas se ha parado, realiza una serie de acciones para poder recuperarse de la parada. Ver los parámetros de las tablas 5.45, 5.46, 5.47 y 5.48 y la figura 5.26.

Void	addToConfig (ArConfig *config, const char *sectionName, int priority=ArPriority::NORMAL)
	ArActionStallRecover (const char *name="stall recover", double obstacleDistance=225, int cyclesToMove=50, double speed=150, double degreesToTurn=45) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionStallRecover () Destructor.

Tabla. 5.45. **Funciones Miembro Públicas ArActionStallRecover**

Enum	State { **STATE_NOTHING** = 0, **STATE_GOING** }
Enum	What { **BACK** = 0x1, **FORWARD** = 0x2, **TURN** = 0x4, **TURN_LEFT** = 0x8, **TURN_RIGHT** = 0x10, **MOVEMASK** = BACK | FORWARD, **TURNMASK** = TURN | TURN_LEFT | TURN_RIGHT }

Tabla. 5.46. **Tipos Protegidos ArActionStallRecover**

Void	addSequence (int sequence)
Void	doit (void)

Tabla. 5.47. **Funciones Miembro Protegidas ArActionStallRecover**

ArActionDesired	myDesired
Int	myCount
Int	myCyclesToMove
Int	myCyclesToTurn
Double	myDegreesToTurn
Double	myDesiredHeading
Int	myDoing
time_t	myLastFired
Double	myObstacleDistance
ArResolver*	myResolver
Std::map< int, int >	mySequence
Int	mySequenceNum
Int	mySequencePos
Int	mySideStalled
Double	mySpeed
Int	myState

Tabla. 5.48. **Atributos Protegidos ArActionStallRecover**

```
ArActionStallRecover::ArActionStallRecover ( const char *  name = "stall recover",
                                             double  obstacleDistance = 225,
                                             int  cyclesToMove = 50,
                                             double  speed = 150,
                                             double  degreesToTurn = 45
                                            )
```

Figura. 5.26. **Documentación de Constructores y Destructores de la Función Miembro ArActionStallRecover**

Parámetros ArActionStallRecover

Name	Nombre de la acción.
obstacleDistance	Distancia a la cual no se mueve debido a un obstáculo. (mm).
cyclesToMove	Número de ciclos para moverse (# de ciclos)
Speedy	Velocidad a la cual ir para adelante o regresar (mm/seg.)
degreesToTurn	Número de grados para girar.

5.1.17.21. ArActionStop

Acción para detener el robot. Esta acción simplemente establece el robot a la velocidad de 0 y un ángulo de inicio de 0 grados. Ver los parámetros de las tablas 5.49, y 5.50 y la figura 5.27.

	ArActionStop (const char *name="stop") Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionStop () Destructor.

Tabla. 5.49. **Funciones Miembro Públicas ArActionStop**

ArActionDesired	myDesired

Tabla. 5.50. **Atributos Protegidos ArActionStop**

```
ArActionStop::ArActionStop ( const char * name = "stop" )
```

Figura. 5.27. **Documentación de Constructores y Destructores de la Función Miembro ArActionStop**

Parámetros ArActionStop

Name Nombre de la acción.

5.1.17.22. ArActionTriangleDriveTo

Acción para manejar hacia objetivo tipo triángulo con el ArLineFinder. Esta acción usa el ArLineFinder para encontrar líneas continuas en el rango del láser que se encuentra en un ángulo, formando el punto de la forma triangular, a través del cual el robot se manejará y se alineará.

Si el objeto ArLineFinder no se ha dado en el constructor, posteriormente buscará un ArRangeDevice en el ArRobot llamado láser, y luego creará su propio objeto ArLineFinder usando este dispositivo de rango.

Los parámetros que describen la forma triangular, deben ser establecidos en el setTriangleParams(). El valor por defecto consiste en una forma triangular de dos líneas de

254 mm que se cruzan en un ángulo de 135 grados. Esta es la forma triangular de las estaciones de carga. Se pueden construir otros objetivos con cualquier otro material que pueda ser sensado por el láser, como madera, cartón o metal.

Si el SetTwoStageApproach() se encuentra en true (por defecto) primero se manejará el robot por medio metro de la distancia del punto deseado (el distFromVertex desde el punto del vértice a lo largo de la bisección del ángulo entre segmentos) y a continuación después de que llegue a ese punto girará en dirección del vértice y se manejará hasta el punto final.

Si el SetTwoStageApproach se establece con false, se manejará hasta el punto final. Se detendrá a la distancia closeDist a menos que se requiera un gran giro donde se requiere un segundo stop que se dispara a 2 veces la distancia closeDist.

Se maneja a la velocidad establecida en el constructor. Si el robot está más cerca del vértice de lo que debería para el acercamiento, se saltará el acercamiento, si el robot está más cerca del vértice que del punto final, regresará.

Si se desea que la acción maneje directamente hacia el vértice en lugar de hacer hacia un punto en frente de debe poner con argumento de true para llamar al setGotoVertex, y dejará de hacer esto cuando la distancia del vértice sea cero, se puede hacer esto también hasta que un parachoques se accione debido a la acción.

Si se desea que el robot se posicione usando su frente se debe usar setPositionFront. Ver los parámetros de las tablas 5.51, 5.52, 5.53, 5.54 y 5.55 y la figura 5.28 y 5.29.

enum	State { STATE_INACTIVE, STATE_ACQUIRE, STATE_SEARCHING, STATE_GOTO_APPROACH, STATE_ALIGN_APPROACH, STATE_GOTO_VERTEX, STATE_GOTO_FINAL, STATE_ALIGN_FINAL, STATE_SUCCEEDED, STATE_FAILED }

Tabla. 5.51. **Tipos Públicos ArActionTriangleDriveTo**

Virtual void	activate (void)
	ArActionTriangleDriveTo (const char *name="triangleDriveTo", double finalDistFromVertex=400, double approachDistFromVertex=1000, double speed=200, double closeDist=100, double acquireTurnSpeed=30) Constructor.
Virtual void	desactivate(void)
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
Bool	getAcquire (void) Obtiene si el giro a realizarse para ver el triángulo descrito
Bool	getAdjustVertex (void) Obtiene cuanto se ajusta el vértice o no.
Data*	getData (void)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Double	getFinalDistFromVertex (void) Obtiene la distancia final desde el vértice.
Bool	getGotoVertex (void) Obtiene si siempre se va hacia el vértice y no para el punto.
Double	getIgnoreTriangleDist (void) Obtiene la distancia del triángulo a la cual empieza a ignorarlo.
ArLineFinder*	getLineFinder () Establece el seguidor de línea utilizado.
Bool	getSaveData (void)
State	getState (void)
Bool	getUseIgnoreInGotoVertexMode (void) Obtiene si se está ignorando el triángulo en el modo de vértice.
Int	getVertexUnseenStopMSecs (void) Obtiene cuanto tiempo se debe mover sin ver el vértice (0 es sin tiempo)
Void	setAcquire (bool acquire=false) Establece el giro para poder alcanzar el vértice
Void	setAdjustVertex (bool AdjustVertex) Establece cuanto se debe ajustar el vértice o no.
Void	setFinalDistFromVertex (double dist) Establece la distancia final desde el vértice
void	setGotoToVertex (bool gotoVertex) Establece si siempre debe ir al vértice y no al punto de en frente.
Void	setIgnoreTriangleDist (double dist=250, bool useIgnoreInGotoVertexMode=false) Establece la distancia del triángulo a la cual empieza a ignorarlo.
Void	setLineFinder (ArLineFinder *lineFinder) Establece el seguidor de línea a usar.
Void	getLogging (void) Obtiene si se ha iniciado el manejo o no.
Bool	setLogging (bool logging) Establece si se ha iniciado el manejo o no.
Void	setMaxDistBetweenLinePoints (int maxDistBetweenLinePoints=0) Establece la máxima distancia entre dos puntos en una línea.
Void	setMaxLateralDist (int maxLateralDist=0) Esrablece la máxima distancia lateral desde el robot a la línea del triángulo.
Void	setParameters (double finalDistFromVertex=400, double approachDistFromVertex=1000, double speed=200, double closeDist=100, double acquireTurnSpeed=30)
Virtual void	setRobot (ArRobot *robot)
Void	setSaveData (bool saveData)
Void	setTriangleParams (double line1Length=254, double angleBetween=135, double line2Length=254) Establece los parámetros del triángulo que se está buscando.

Void	setVertexOffset (int localXOffset, int localYOffset, double thOffset) Establece el offset del vértice.
Void	setVertexUnseenStopMSecs (int vertexUnseenStopMSecs=4000) Establece cuanto tiempo se debe mover sin ver el vértice (0 es sin tiempo).
Virtual	~ArActionTriangleDriveTo () Destructor.

Tabla. 5.52**. Funciones Miembro Públicas ArActionTriangleDriveTo**

ArPose	findPoseFromVertex (double distFromVertex)
Void	findTriangle (bool initial, bool goStraight=false)

Tabla. 5.53. **Funciones Miembro Protegidas ArActionTriangleDriveTo**

Bool	myAcquire
Double	myAcquireTurnSpeed
Bool	myAdjustVertex
Double	myAngleBetween
Double	myApproachDistFromVertex
Double	myCloseDist
Data*	myData
ArMutex	myDataMutex
ArActionDesired	myDesired
Double	myFinalDistFromVertex
Unsigned int	myGotLinesCounter
Bool	myGotoVertex
Double	myIgnoreTriangleDist
ArRangeDevice*	myLaser
Double	myLine1Length
Double	myLine2Length
ArLineFinder*	myLineFinder
Std::map< int, ArLineFinderSegment* >*	myLines
Int	myLocalXOffset
Int	myLocalYOffset
Int	myMaxDistBetweenLinePoints
Int	myMaxLateralDist
Bool	myOwnLineFinder
Bool	myPrinting
Bool	mySaveData
Double	mySpeed
State	myState
Double	myThOffset
Bool	myTwoStageApproach
Bool	myUseIgnoreInGoto
ArPose	myVertex
Bool	myVertexSeen
ArTime	myVertexSeenLast
Int	myVertexUnseenStopMSecs

Tabla. 5.54. **Atributos Protegidos ArActionTriangleDriveTo**

Class	Data

Tabla. 5.55. **Clases ArActionTriangleDriveTo**

enum ArActionTriangleDriveTo::State

Figura. 5.28. **Documentación de la Enumeración Miembro ArActionTriangleDriveTo**

Parámetros ArActionTriangleDriveTo

STATE_INACTIVE	No se esta tratando
STATE_ACQUIRE	Encontrando el objetivo
STATE_SEARCHING	Giro para tartar de encontrar el objetivo.
STATE_GOTO_APPROACH	Manejo hacia el punto cercano
STATE_ALIGN_APPROACH	Alineación con el objeto por primera vez
STATE_GOTO_VERTEX	Manejo hacia el vértice
STATE_GOTO_FINAL	Manejo hacia el punto final
STATE_ALIGN_FINAL	Alineación con el objeto por última vez
STATE_SUCCEEDED	Señalando al objetivo
STATE_FAILED	Si no se ha adqurido información y encontrado el vértice falla la operación

void ArActionTriangleDriveTo::findTriangle (bool *initial*, bool *goStraight* = `false`)
`[protected]`

Figura. 5.29. **Documentación de la Función Miembro ArActionTriangleDriveTo**

Busca los puntos del vértice y su ángulo

Parámetros ArActionTriangleDriveTo

Inicial Al buscar el vértice inicial, se busca en frente la esquina más cercana:

Diferencia el largo de la línea 1.

Diferencia el largo de la línea 2.

ArActionTriangleDriveTo::Data * ArActionTriangleDriveTo::getData (void)

Solo para uso interno, obtiene los datos guardados.

bool ArActionTriangleDriveTo::getSaveData (void) `[inline]`

Solo para uso interno, obtiene si se han guardado los datos o no.

void ArActionTriangleDriveTo::setSaveData (bool *saveData*) [inline]

Solo para uso interno, establece si se han guardado los datos o no.

5.1.17.23. ArActionTurn

Acción para girar cuando las acciones con mayor prioridad han limitado la velocidad. Esta acción esta básicamente hecha de modo que se puede taner varios limitadores de diferentes tipos para mantener la velocidad bajo control, y luego mezclarlos para entrar en el modo Wander. Ver los parámetros de las tablas 5.56 y 5.57.

	ArActionTurn (const char *name="turn", double speedStartTurn=200, double speedFullTurn=100, double turnAmount=15) Constructor.
virtual ArActionDesired *	fire (ArActionDesired currentDesired)
virtual const ArActionDesired *	getDesired (void) const
virtual ArActionDesired *	getDesired (void)
Virtual	~ArActionTurn () Destructor.

Tabla. 5.56. **Funciones Miembro Públicas ArActionTurn**

ArActionDesired	myDesired
Double	mySpeedFull
Double	mySpeedStart
Double	myTurnAmount
Double	myTurning

Tabla. 5.57. **Atributos Protegidos ArActionTurn**

Todas estas acciones (1-23) son enlazadas con el objeto ArRobot mediante ArRobot::addAction(), y se ponen en orden en la lista de prioridades. Cuando una acción es añadida al robot se llama al objeto ArAction::setRobot(). Se puede anular esto en la subclase de la acción.

Las acciones son evaluadas por la acción de resolución de ArRobot en orden descendiente de prioridad (la prioridad más alta al inicio, la más baja al final) en cada ciclo de tarea justo antes del estado de reflexión.

La acción de resolución invoca cada método de la acción fire(), combinando los comandos de movimiento deseados con un objeto simple de ArActionDesired, el cual es usado luego en el estado de reflexión para enviar los comandos del movimiento al robot.

Nota: Si se envía comandos simples mientras se encuentran acciones activas pueden existir conflictos entre estos dos.

5.1.17.24. Acción Deseada

Los objetos ArActionDesired son usados para pasar los valores del canal de la acción deseada y fortalezas de cada uno de los métodos ArAction::fire() de la resolución. Un objeto ArActionDesired siempre debe ser reseteado (ArActionDesired::reset()) antes de volverlo a usar.

Existen seis canales de acción: la velocidad (ArActionDesired::setVel), la partida (ArActionDesired::setDeltaHeading o ArActionDesired::setHeading para una partida total) la velocidad máxima de traslación hacia delante (ArActionDesired::setMaxVel), la velocidad máxima de traslación hacia atrás (ArActionDesired::setMaxNegVel) y la velocidad máxima de rotación (ArActionDesired::setMaxRotVel).

La acción da a cada canal una fuerza entre 0.0 la más baja y 1.0 la más alta. Las fuerzas son usadas por la resolución (resolver) para calcular el efecto relativo de los canales asociados cuando se combinan múltiples acciones de movimientos deseados.

Los canales de la velocidad máxima, la velocidad máxima negativa y la velocidad de rotación, simplemente imponen límites de velocidad y por tanto un control indirecto del robot.

Para un uso más avanzado los objetos ArActionDesired pueden fusionarse (ArActionDesired::merge) y promediarse (ArActionDesired::startAverage, ArActionDesired::addAverage, ArActiondesired::endAverage).

5.1.17.25. ArResolver (resolvedor de acciones)

El ArResolver es la clase base de la acción de resolución. La resolución por defecto que utiliza ArRobot es el ArPriorityResolver. La resolución utilizada por ArRobot se puede cambiar llamando a ArRobot::setResolver, si se quiere crear una implementación alternativa del ArResolver.

Pero se debe tomar en cuenta que debe haber una sola resolución por cada objeto ArRobot. (A pesar de que una resolución si podrá contener varias resoluciones dentro de sí). Se debe tener en cuenta que aunque un robot tiene una resolución en particular vinculada a él, la petición de la resolución no está ligada a ningún robot.

El ArPriorityResolver trabaja en base a la lista de acciones en prioridad descendente, estableciendo en cada canal de movimiento (velocidad de traslación, partida, velocidad máxima, etc.) los valores deseados basándose en las acciones a realizarse (que regresan de los métodos fire()) en proporción con sus respectivas fortalezas así como de las prioridades de las acciones, actualizando cada movimiento del canal hasta que su fuerza sea de 1.0 o hasta que la lista de acciones se haya agotado.

Una vez que el canal haya alcanzado 1.0 de fuerza, no se pueden realizar más cambios en ese canal (así es como se previene que las acciones de menor prioridad hagan cambios sobre las de mayor prioridad dentro del canal). Las mismas acciones de prioridad son promediadas si ambas proveen una salida en el mismo canal.

A manera de ejemplo se muestra una tabla que indica en cada paso los valores de las acciones deseadas y la fortaleza de la velocidad de cada canal, y el cambio resultante de la velocidad final de canal de la resolución y la decisión de fuerza, para cuatro acciones ficticias A, B, C y D.

En el ejemplo de la tabla 5.58, se nota que las acciones B y C tienen la misma prioridad y son promediadas antes de ser combinadas con los valores previos calculados en el paso 1. El resultado de esta combinación ((velocidad deseada B:-100)*(Fuerza velocidad B:1.0)+(velocidad deseada C:200)*(Fuerza velocidad C:0.5))/2 => (-100 +100)/2 => 0. Por

lo tanto las acciones B y C se terminan anulando mutuamente. Combinando este resultado con los “valores actuales deseados” calculados en el paso 1 resulta (velocidad deseada paso 1:-400)*(fuerza deseada paso 1:0.25) + (velocidad deseada paso 4:0)*(fuerza deseada paso 4:0.75) => -100.

Paso #	**Acción**	**Prioridad**	**Valor de la velocidad de canal de la acción deseada**	**Fuerza de la velocidad de canal de la acción deseada**	**Valor actual de velocidad final**	**Fuerza actual de velocidad final**
1	A	4	-400	0.25	-400	0.25
2	B	3	-100	1.0	Combinadas para usar en el paso 4	
3	C	3	200	0.50		
4	B & C	3	0	0.75	-100	1.0
5	D	1	500	0.50	Sin cambio	Sin cambio
Resultado Final					-100	1.0

Tabla. 5.58**. Ejemplos de Acciones Ficticias**

En este ejemplo, resulta que en el paso 5, la acción D no tiene ningún efecto ya que la fuerza del canal ha llegado a 1.0 en el paso 4, antes que la acción fuera considerada por la resolución. El mismo método se utiliza para los otros canales.

5.1.18. Acciones Predefinidas

ARIA tiene varias clases de acciones predefinidas. Las clases de acciones de movimiento tienen un prefijo “%ArAction” y establece uno o ambos canales, tanto de la velocidad de traslación (setVel) y el de partida (setDeltaHeading y setHeading) que tiene efecto en el movimiento.

Las clases de acciones limitantes están seguidas por el prefijo “%ArActionLimiter” y establece uno o más canales de la velocidad máxima de traslación y rotación, para disminuir o prevenir el movimiento, y se basa usualmente en las lecturas de los sensores cercanos.

5.1.19. Acciones Mixtas

Las acciones son más útiles cuando se las mezcla. En algunos programas es necesario mezclar acciones de límite y de movimiento. Se puede utilizar diversas funciones límites

como por ejemplo Limiter o LimiterFar y acciones de movimiento como pueden ser el JoyDriveAct y el KeyDriveAct.

Se debe considerar que las acciones de limitación tienen mayor prioridad que las de movimiento, debido a que se debe realizar un manejo preventivo del robot para tomar en cuenta potenciales obstáculos detectados por el robot.

Para el desarrollo de los programas se puede recurrir a otras funciones más avanzadas de ARIA que contribuyen al comportamiento del robot. Estas acciones son individuales y contribuyen de manera discreta al manejo del robot por lo cual son fácilmente reutilizables.

Por ejemplo, una acción limitante que previene que el robot choque con una pared cuando se mueve hacia delante, se puede utilizar como tal en otro programa y tiene el mismo efecto, excepto cuando se maneje el robot con un joystick, pero puede tener una acción de menor prioridad como la de seguir una pelota de color utilizando la identificación de objetos.

La acción de seguimiento de color no necesita conocer nada acerca de las acciones de conducción segura ya que las acciones de mayor prioridad se encargan de ello. Existen programas como cuando se desarrolla la evasión de obstáculos que demuestran como las diferentes acciones pueden usarse y cómo interactúan.

La acción de recuperación de paradas (ArActionStallRecover) influencia en el robot solo cuando los motores se encuentran parados, deshabilitando las acciones de más baja prioridad mediante el uso de las fuerzas trasnacional y rotacional hasta que el robot salga del estado de parada. Dentro de la evasión de obstáculos también actúan otras acciones como por ejemplo ArActionAvoidFront y ArActionConstantVelocity.

5.1.20. Grupos de Acciones

Los grupos de acciones permiten fácilmente habilitar (activar) o deshabilitar (desactivar) una serie de acciones a la vez. Para ello se debe crear en primer lugar un ArActionGroup

ligado al ArRobot. Luego cuando se añada una ArAction al ArActionGroup, se añade automáticamente al ArRobot así como al Grupo. Algunos Grupos de acciones ya se encuentran predeterminados en ARIA.

5.1.21. Dispositivos de medición de rango

Los dispositivos de rango (ArRangeDevice) son abstracciones de los sensores los cuales detectan la presencia de obstáculos en el espacio alrededor del robot, proveyendo una serie de lecturas espaciales en el tiempo real. Las clases de los dispositivos de rango de ARIA transforman todas las lecturas en puntos específicos en el mismo sistema de coordenadas globales en dos dimensiones. (Este es el mismo sistema de coordenadas del sistema de posición del ArRobot).

Actualmente, existen cuatro implementaciones primarias del ArRangeDevice incluidas en ARIA: sonar (ArSonarDevice), Láser (ArSick), los parachoques del robot (ArBumpers), y los sensores infrarrojos (ArIRs). Además, el ArForbiddenRangeDevice es un dispositivo de rango virtual que crea lecturas de rango de las fronteras “Área Prohibida” y “Línea Prohibida” en las regiones del ArMap.

El ArRangeDevice espera dos tipos de objeto ArRangeBuffer para almacenar lecturas: actuales y acumuladas, aunque no todas las implementaciones de ArRangeDevice proveen datos al buffer acumulado. Cada buffer tiene dos formatos diferentes de lecturas: box y polar (ArRangeDevice::currentReadingPolar(), ArRangeDevice::currentReadingBox(), ArRangeDevice::cumulativeReadingPolar(), ArRangeDevice::cumulativeReadingBox()).

El buffer actual contiene la serie más recientes de lecturas; el buffer acumulativo contiene las lecturas que se almacenaron durante un periodo de tiempo, y está limitado por el tamaño del buffer (ArRangeBuffer::setSize()).

Algunos dispositivos de rango también proveen lecturas “crudas”, las cuales son valores originales dados por el propio dispositivo. Los dispositivos de rango se encuentran conectados a una instancia específica de ArRobot, para obtener la posición y otra

información del robot cuando las lecturas son recibidas y almacenadas, y también para proveer una manera de encontrar todos los dispositivos de rango ligados al robot. Algunos dispositivos de rango utilizan la conexión del robot para comunicarse con su dispositivo (ArSonarDevice, ArBumpers, ArIRs).

Se puede ligar un ArRangeDevice a un objeto ArRobot con ArRobot::addRangeDevice() y se lo puede quitar con el ArRobot::remRangeDevice(). La lista de todos los dispositivos conectados puede ser consultada utilizando ArRobot::findRangeDevice() y ArRobot::hasRangeDevice(). También se la puede obtener si se llama a ArRobot::getRangeDeviceList().

(Se debe tener en cuenta que si el sonar se encuentra integrado con el microcontrolador, y el microcontrolador siempre envía datos del sonar al robot (si el robot tiene sonar) aún así se debe adjuntar el objeto ArRangeDevice al robot para poder usarlo).

El ArRobot incluye algunos métodos útiles para realizar operaciones comunes en todos los dispositivos adjuntados, incluyendo:
ArRobot::checkRangeDevicesCurrentPolar(),
ArRobot::checkRangeDevicesCurrentBox(),
ArRobot::checkRangesDevicesCumulativePolar(),
ArRobot::checkRangeDevicesCumulativeBox (),
para encontrar el rango de lecturas más cercano al robot dentro de algunas regiones.

Cada dispositivo de rango tiene un mutex (Usa ArRangeDevice::lockDevice() y ArRangeDevice::unlockDevice() para bloquearlo o desbloquearlo) de esta manera puede accederse seguramente desde múltiples hilos.

Por ejemplo el ArSick utiliza un hilo para leer los datos del láser, pero también funciona el checkRangeDevice() en el ArRobot lockDevice() de tal manera que se puedan leer los datos sin tener conflictos con las lecturas del hilo del ArSick, y luego se usa el unlockDevice() cuando ya se haya hecho la lectura. Posteriormente se verá la acción de enlace mediante “hilos”.

5.1.22. Functors

Los functors se utilizan en todo el ARIA, y son la abreviación de la función puntero. Un Functor permite llamar a una función sin la necesidad de conocer la declaración de la función. En lugar de ello el compilador y el enlazador (linker) averiguan la manera más apropiada de llamar a la función.

Las funciones de punteros son plenamente soportadas por el lenguaje C. C++ trata a las funciones punteros como C, pero para los métodos de llamadas de clases, se requiere un objeto, así como el tipo de información acerca de la clase. Por lo tanto, ARIA contiene un conjunto de plantillas de clases que contienen esta información.

ARIA hace un uso intensivo del ArFunctors para la “llamada” de funciones. Para crear una instancia de un Functor, primero se debe determinar cuántos parámetros la función necesita y si retorna algún valor. Muchas veces se usa un puntero a la clase base abstracta ArFunctor, la cual puede ser invocada sin parámetros y sin valores de retorno.

Las subclases son usadas para funciones con diferentes números de parámetros y valores de retorno. Por ejemplo ArFunctor1, ArFunctor2, ArRetFunctor, ArRetFunctor1, y ArRetFunctor2. Cuando se invocan los parámetros pueden ser suministrados y pasados a la función objetivo o al método, y un valor de retorno también puede darse. Los tipos de argumentos y/o valores de retornos son dados como argumentos plantilla.

Cuando se crea un objeto functor se debe proveer el tipo e instancia de un objeto para invocar al método de; o específicamente poner el estado de la función como clase global.

Para ello se debe utilizar una clase base en concreto del ArFunctor en lugar de las clases abstractas: ArFunctorC, ArFunctor1C, ArFunctor2C, ArRetFunctorC, ArRetFunctor1C, ArRetFunctor2C, ArGlobalFunctor, ArGlobalFunctor1, etc.

Como ejemplo se puede observar en la figura 5.30 el ExampleClass, el cual es una clase que contiene una función llamada aFunction(). Una vez que el Functor se ha creado de esta manera, puede pasar ahora a ser una función de ARIA que quiere la llamada de un Functor.

Y el método ExampleClass::aFunction() puede ser llamado en el objeto obj cuando el functor es invocado.

```
    class ExampleClass {
    public:
        void aFunction(int n);
    };

    ...

    ExampleClass obj;
    ArFunctor1C<ExampleClass, int> functor(obj,
&ExampleClass::aFunction);

    ...

    functor.invoke(42);
```

Figura. 5.30. **Ejemplo de Clases**

Para invocar a un functor, simplemente se llama a la función invocada en el functor. Si ésta toma parámetros, la llamada la invoca con estos parámetros. Si el Functor tiene un valor de retorno, llama al invocador. El valor de retorno de la función será pasado de vuelta a través de la función invocadora.

5.1.23. Entrada del Joystick y del Teclado

ARIA ofrece varias clases que permiten utilizar la entrada del joystick y del teclado, así como clases de Acción (Acciones) que usan estas entradas para el manejo del robot.
ArJoyHandler es una interfaz que cruza datos de la plataforma al Joystick.

Sus funciones principales son ArJoyHandler::getButtons, ArJoyHandler::getAdjusted, ArJoyHandler::setSpeeds y ArJoyHandler::getDoubles. ArKeyHandler es una interfaz que cruza los eventos de teclado hacia la plataforma. Su función de tecla es ArKeyHandler::addKeyHandler la cual vincula una tecla específica a una función dada. Contiene un enum para ArKeyHandler::KEY que contiene valores para teclas especiales.

También se puede adjuntar una tecla de manejo al robot mediante ArRobot::attachKeyHandler(), el cual añade algunas teclas por defecto, incluyendo un

manejador de la tecla escape que desconecta y cierra el programa (especialmente útil en Windows, en donde Ctrl-C o la terminal de cierre no la limpiarán apropiadamente). Debido a que un PC solo puede manejar un teclado, ARIA mantiene un puntero global ArKeyHandler, el cual puede ser consultado mediante Aria::getKeyHandler().

Nota: ARIA provee dos acciones simples, ArActionKeydrive y ArActionJoydrive, para manejar el robot desde una entrada de teclado o desde el joystick. El ArActionRatioInput combina varias fuentes de entrada (como teclado, joystick de computadora, joystick de la plataforma robótica o comandos de teleoperación recibidos mediante ArNetworking) en una manera más consistente y configurable.

5.1.24. Threading

ARIA es altamente multi-enlazable. En esta sección se presenta alguno de los conceptos fundamentales detrás de la escritura de código enlazado de ARIA. ARIA provee un número de clases de soporte para hacer más fácil la escritura de código enlazado orientado a objetos. Estas clases son: ArASyncTask, ArCondition, ArMutex, y ArThread.

El código enlazado seguro en la mayoría de casos proporciona la coordinación entre enlaces cuando manejan los mismos datos. Si se quiere evitar el problema de lecturas o escrituras de datos de uno o más enlaces al mismo tiempo que otros enlaces lean o escriban los datos, se necesita proteger los datos con objetos de sincronización.

5.1.25. Objetos de sincronización de enlaces

En ARIA, los objetos de sincronización son ArMutex y ArCondition. ArMutex es la más útil. ArMutex (mutex es la abreviación de exclusión mutua) provee una envoltura alrededor de las llamadas del sistema (funciones pthreads en Linux y funciones CriticalSection en Windows) que excluyen a otros enlaces de continuar mientras el objeto mutex está “bloqueado”.

Cuando los enlaces bloquean el mutex mientras acceden a datos compartidos, se asegura que solo un enlace acceda a los datos compartidos a la vez. Por lo tanto, la manera

apropiada de usar el mutex es bloquearlo justo antes de acceder a los datos compartidos, y siempre desbloquearlo cuando ya se ha hecho. Si el mutex no ha sido bloqueado, entonces es bloqueado y el enlace continúa.

Si el mutex ya ha sido bloqueado por otro enlace, entonces se bloquea en la llamada de bloqueo hasta que el otro enlace lo desbloque. Si un mutex no se bloquea nunca (por ejemplo, la función devuelve una condición de error desbloquearla), entonces cualquier intento de bloqueo será bloqueado para siempre, creando así un “punto muerto”.

La documentación de un método puede indicar cuando un bloqueo es necesario antes de usarlo; en general, cuando se usa un objeto que puede ser compartido por otros enlaces, todos los enlaces que usan dicho objeto deben bloquear el mismo mutex (usualmente contenida sin la clase objeto con métodos dados para el bloqueo y el desbloqueo) mientras se usa el objeto.

ArCondition es una utilidad usada ocasionalmente que pone el enlace actual a dormir hasta que otra señal de enlace lo despierte y continúe ejecutándose. Esto se puede usar para esperar en un enlace una cantidad indefinida de tiempo hasta que se produzca algún evento en otro enlace que señale el ArCondition.

5.1.26. Clase de tareas asincrónicas

ARIA proporciona el ArASyncTask el cual puede ser sub claseado para implementar un enlace largo y su estado como un objeto. A diferencia de las tareas sincronizadas del robot, las tareas asincrónicas corren en enlaces separados. Al igual que ArMutex, esta clase envuelve las llamadas de enlaces del sistema operativo en forma de una plataforma cruzada.

Normalmente el ArASyncTask representa un enlace que se ejecuta en un bucle durante todo el programa. Para usar ArASyncTask, se deriva una clase de ArASyncTask y anula la función ArASyncTask::runThread(). Esta función es llamada automáticamente en el nuevo enlace cuando el nuevo enlace ha sido creado. Para crear e iniciar un enlace, se llama a

ArASyncTask::create(). Cuando la función ArASyncTask::runThread() termina, el enlace terminará y será destruido. Enlaces separados pueden pedir que la tarea termine mediante la llamada a ArAsyncTask::stopRunning(), y con el enlace se puede comprobar esta solicitud con ArASyncTask::getRunningWithLock().

Esta clase es principalmente una envoltura conveniente alrededor de ArThread de modo que se puede crear un objeto propio que encierre el concepto de un enlace.

5.1.27. Datos Globales

La clase estática de ARIA contiene diversos datos globales de Aria. ARIA contiene una lista de todas las instancias de ArRobot. Para encontrar un robot por su nombre se usa Aria::findRobot(), o también se puede usar Aria::getRobotList() para obtenerlo de una lista de robots.

Para encontrar rutas de alto nivel se usa Aria::getDirectory() (Por lo general se encuentra en C:\Program Files\MobileRobots\Aria en Windows, y en /usr/local/Aria en Linux). Esto es útil, por ejemplo, para localizar los archivos de parámetros del robot para detalles de operaciones individuales. Se puede usar también Aria::setDirectory() para cambiar esta ruta del programa si se necesita que se anule lo que ARIA ha decidido.

Para hacer la inicialización global se llama a Aria::init() al inicio del programa, y para terminarlo se usa Aria::exit() para terminar todos los enlaces de ARIA antes de finalizar el programa. La clase Aria también contiene objetos globales para compartir los parámetros de configuración y otra información.

5.1.28. Clases de Dispositivos e Interfaces de accesorios

Además de los dispositivos de rango, ARIA incluye clases para comunicarse con varios otros tipos de dispositivos. (La mayoría de estos dispositivos son accesorios opcionales, y no todos los robots los tienen instalados).

- ✓ ArPTZ define una interfaz común y acceso a subclases de dispositivos específicos, incluyendo:
  - o ArVCC4 que provee paneo, inclinación, zoom y otros controles de la cámara CANON a través del puerto serial AUX del microcontrolador del robot.
  - o ArSonyPTZ que provee paneo, inclinación, zoom y otros controles de la cámara Sony a través del puerto serial AUX del microcontrolador del robot.
  - o ArDPPTU provee control de la dirección de percepción de la unidad de Paneo/Inclinación.
- ✓ ArAnalogGyro habilita el giroscopio interno del robot, si se encuentra presente y habilita la configuración en el firmware, después que se conecta el ArRobot y corrige automáticamente el componente de posición del valor de posición de ArRobot como nuevo dato recibido. El giroscopio también mide su propia temperatura como parte de la operación y el ArAnalogGyro hace que este valor se encuentre disponible, para poder detectar condiciones dañinas de alta temperatura en la electrónica del robot.
- ✓ ArGripper provee acceso al Gripper del robot Pioneer.
- ✓ ArP2Arm provee acceso al brazo robótico.
- ✓ ArTCM2 provee acceso a la brújula TCM2, si se encuentra presente.
- ✓ ArACTS_1_2 se comunica con el programa ACTS para obtener información de seguimiento de trayectorias.
- ✓ ArVersalogicIO provee acceso a los puertos de E/S digitales y analógicos en la tarjeta madre Versalogic (Solo en Linux). (El soporte depende si el robot tiene una tarjeta madre Versalogic internamente).
- ✓ ArSystemStatus provee datos acerca del sistema operativo (Solo en Linux) como el uso de la CPU y la intensidad de señal inalámbrica.
- ✓ ArGPS provee acceso a los datos recibidos desde el dispositivo de Posicionamiento Global (GPS). Las subclases implementan acciones especiales requeridas para dispositivos específicos, como por ejemplo el ArNovatelGPS para el dispositivo Novatel G2 y otros similares.

Algunas interfaces de dispositivos son soportadas por librerías adicionales.

5.1.29. Clases Utilitarias

Algunas de las clases utilitarias de propósito general se incluyen con: ArMath, ArUtil, ArTime, ArPose, ArLog, ArSectors, ArRingQueue.

5.1.30. ArConfig

ArConfig es un mecanismo para almacenar parámetros de configuración para diferentes módulos independientes en un archivo de texto. La clase Global Aria mantiene un puntero global ArConfig al cual cualquier módulo del programa puede acceder.

Para registrar un nuevo parámetro con ArConfig se usa ArConfig::addParam, y se usa ArConfig::addProcessFileCB para registrar un regreso de llamada de un functor llamado cuando la configuración cambia (al cargar el archivo con ArConfig::parseFile, o por otros medios tales como una actualización desde un cliente remoto a través de ArNetworking).

5.1.31. Grupos de Información Compartida

En un programa compuesto por varios módulos independientes, a menudo es necesario cambiar o combinar datos entre ellos de una manera general e inmediata. Para hacer esto ARIA provee la clase ArStringInfoGroup, de la cual la clase global ARIA contiene una instancia (además para una instancia de ArConfig la cual es usada específicamente para información de configuración la cual cambia frecuentemente, se carga un archivo o se puede manejar por una entrada de usuario).

Un ejemplo de ArStringInfoGroup es la clase ArServerInfoStrings contenida en la librería auxiliar de ArNetworking. En la inicialización del programa, una llamada del functor puede añadirse al objeto global ArStringInfoGroup el cual al invocarlo inmediatamente pasa un par de cadenas tecla/valor desde el objeto ArStringInfoGroup a un objeto ArServerInfoStrings, el cual provee acceso a estos datos a través de la red (por ejemplo hacia MobileEyes).

Los componentes independientes del programa pueden cambiar sus valores en el objeto InfoGroup sin necesidad de un conocimiento especial de los destinatarios de los datos (para este ejemplo es la clase ArServerInfoStrings).

Debido a que MobileEyes despliega los datos en una pequeña tabla de textos junto con otra información del robot como la posición en velocidad, esta es una manera útil para proveer información estadística acerca de la operación actual del software del robot. (En Linux, por ejemplo, se puede usar la clase ArSystemStatus para publicar la información desde el sistema operativo, cargándola desde el CPU).

5.1.32. Mapas

En las aplicaciones de robot móviles, a menudo es necesario almacenar mapas del entorno del robot, para usarlo en la navegación, localización, etc. ARIA provee la clase ArMap para leer los datos de los mapas desde un archivo, obteniendo y modificando sus contenidos en la aplicación, y escribiéndolos de vuelta en un archivo.

El ArMap contiene datos acerca del ambiente sensado/sensible (paredes, obstáculo, etc.) y los objetos dados por el usuario como puntos de metas. La documentación para la clase ArMap describe el formato del archivo del mapa en detalle. ARNL, SonARNL y MobileSim usan el formato de archivos de mapas de ArMap.

5.1.33. Sockets

La clase ArSocket es una envoltura alrededor del socket de la capa de comunicación de la red del sistema operativo. ARIA utiliza principalmente ArSocket para abrir un puerto de un servidor y conectarlo a otro puerto servidor.

Para conectarse a un puerto, simplemente se construye un socket que contenga el nombre del host o la dirección IP del Host, el número del puerto, y el tipo de socket de ARIA (TCP o UDP), por ejemplo:

```
ArSocket sock("host.name.com", 4040, ArSocket::TCP);
```

O se llama a la función ArSocket::Connect(), de la siguiente manera:

```
ArSocket sock;
sock.connect("host.name.com", 4040, ArSocket::TCP);
```

Para abrir un Puerto servidor, se construye un socket:

```
ArSocket sock(4040, true, ArSocket::TCP);
```

O se llama

```
ArSocket::open(4040, ArSocket::TCP);
```

5.1.34. ArNetworking

Para una infraestructura más avanzada de red, ArNetworking proporciona un sistema ampliable de las actualizaciones y pedidos de datos entre las aplicaciones del cliente o servidor vía TCP o UDP, usando el mismo concepto de "paquete" base como comunicación del robot.

Por ejemplo, se usa el ArNetworking para conectar múltiples robots que trabajan juntos, fuera de las interfaces de usuario para los servidores de control a bordo, o los programas de control del robot de los datos fuera de los recursos.

5.1.35. Sonido y Voz

ARIA provee soporte para sonido, y librerías separadas que se usan para la voz y el sonido de la red.

1. El ArSoundsQueue de ARIA provee un método para manejar la salida de sonido generada por varios componentes de una aplicación de ARIA grande en secuencia y

bucles. Es un administrador de reproducción de sonido y síntesis de voz, el cual usa el último enlace seguro de requerimientos de sonido y síntesis de voz, y ejecuta un procesamiento de fondo para ello.

Se puede usar para la mayoría de reproducciones de audio no triviales y síntesis de voz necesitadas. Estas librerías de síntesis de voz, librería de reconocimiento de voz, y la librería NetAudio son diseñadas para ser usadas en conjunto con la clase ArSoundsQueue para coordinar el uso de dispositivos de audio.

2. El ArSoundPlayer de ARIA una plataforma cruzada básica de capacidad de reproducción de archivos de audio. Los archivos de audio se encuentran en formato WAV (Windows RIFF). Esta clase provee métodos estáticos que pueden ser usados por ArSoundsQueue para reproducción de archivos de audio.

3. Las librerías separadas proporcionan envolturas alrededor de algunas síntesis de voz (Texto a Voz) y reconocimiento de productos:

- ✓ La librería ArSpeechSynth_Festival utiliza el sistema libre Festival de la Universidad de Edimburgo para realizar el procesamiento de voz. Proporciona una clase ArFestival como una envoltura alrededor de Festival.
- ✓ La librería ArSpeechSynth_Cepstral utiliza la librería Swift de Cepstral, incluida para realizar el procesamiento de voz. Para ello provee la clase ArCepstral. Esta clase ofrece algunas características extra mejores que las de ArFestival, las cuales proveen voces de alta calidad para el uso con Swift.
- ✓ La librería ArSpeechRec_Sphinx usa Sphinx de la Universidad de Carnegie Mellon para realizar el reconocimiento de voz. Para ello ArSphinx provee una interfaz.

Las funciones comunes de ambas librerías de procesamiento de voz se incluyen en la clase base ArSpeechSynth.

4. La librería separada ArNetAudio dispone de la grabación de audio, transmisión y reproducción en la red.

✓ ArNetAudioServer automáticamente decodifica y reproduce o graba y codifica de nuevo el audio para un servidor de red (por lo general se ejecuta en la computadora a bordo del robot) y envía y recibe la codificación de audio desde/hacia un cliente.

✓ ArNetAudioClient automáticamente decodifica y reproduce o graba y codifica de nuevo el audio para un cliente de red, enviando y recibiendo el audio codificado hacia/desde un servidor (es decir, a la computadora a bordo del robot).

✓ ArNetAudioIO es la clase común que realiza la codificación/decodificación de voz de las E/S de la plataforma cruzada de audio (a través del codec gratuito Speex). Es utilizado por ArNetAudioServer y ArNetAudioClient, pero también está disponible para uso independiente.

5.1.36. Emacs

A continuación se muestra la especificación reconfiguración que los desarrolladores de MobileRobots.Inc usan en sus archivos .emacs, y también se muestra para el caso en el que se quiera modificar el código usando Emacs y no lidiar con las diferencias.

```
(setq c-default-style '((other . "user")))
(c-set-offset 'substatement-open 0)
(c-set-offset 'defun-block-intro 2)
(c-set-offset 'statement-block-intro 2)
(c-set-offset 'substatement 2)
(c-set-offset 'topmost-intro -2)
(c-set-offset 'arglist-intro '++)
(c-set-offset 'statement-case-intro '*)
(c-set-offset 'member-init-intro 2)
(c-set-offset 'inline-open 0)
(c-set-offset 'brace-list-intro 2)
(c-set-offset 'statement-cont 0)
(defvar c-mode-hook 'c++-mode)
```

ARIA utiliza algunas características de C++ que algunos programadores pueden no tomar en cuenta, y que incluyen algunas soluciones para las diferentes plataformas.

ARIA hace un uso intensivo de la Plantilla Estándar de la Librería (STL). Debido a ello se debe entender la STL con el fin de obtener el mejor uso de algunas de las características de las partes más avanzadas de ARIA.

5.1.37. Argumentos por defecto

En la declaración de la función un valor por defecto de un argumento debe ser especificado. Los argumentos con valores por defecto pueden ser omitidos en la llamada a la función. Por ejemplo, después de declarar la función con un valor por defecto para su argumento entero:

```
void foo(int number = 3);
```

Este valor puede usarse de dos formas:

```
// Uso del valor por defecto para el argumento:
     foo();

     // O, sin el uso del valor por defecto:
     foo(99);
```

Esta conducta es muy útil para tener valores por defecto para opciones no muy claras las cuales usualmente no necesitan cambiarse, pero aún se permite el cambio de estos valores si es necesario sin hacer ARIA más compleja.

Se debe tener en cuenta que la definición de esta función no tiene la asignación en ella, solo la declaración. Por lo tanto la definición para la función del ejemplo sería así:

```
  void foo(int number)
  {
    //...
  }
```

5.1.38. Encadenamiento del constructor

El encadenamiento del constructor es bastante sencillo, aunque en ocasiones no es usado por los programadores de C++. Cada constructor puede dar argumentos a los constructores de las variables miembro que contiene y a los constructores de las clases las cuales hereda. Por ejemplo si se tiene:

```
class BaseClass
{
public:
  BaseClass(int someNumber);
};
```

y

```
class SubClass : public BaseClass
{
public:
  SubClass(void);
  int anotherNumber;
};
```

Cuando se escribe un constructor para una subclase, se puede inicializar ambas la baseClass y el anotherNumber:

```
  SubClass::SubClass(void) : BaseClass(3), anotherNumber(37)
  {
      // ...
  }
```

Se nota como los constructores al ser inicializados deben ser seguidos por dos puntos (:) después del constructor, y deben ser separados por comas. Las variables miembro deben ser inicializadas en el orden en que se encuentren en la clase.

Se debe tener en cuenta que solo la inicialización de los enteros no es lo único o útil, pero si se usa para inicializar los functors será mucho más útil.

El encadenamiento del constructor es usado en varios sectores por ARIA, lo cual se debe tener claro para la comprensión de ARIA, todo lo anterior es lo que se necesita saber para dar un buen uso del cliente.

Si en Windows se pasa de un std::string a un DLL se puede usar el const char*, pero para todo el almacenamiento interno y todos los reportes se pasan de vuelta los std::Springs hacia ARIA.

5.1.39. AREXPORT

Debido a la configuración se Windows para usar DLL's, esta macro es usada para tener cuidado de los atributos requeridos en la declaración para las DLL's. En gran parte los usuarios no necesitan preocuparse por AREXPORT, solo funciones que tienen AREXPORT y funciones en línea se pueden utilizar con archivos DLL en Windows.

5.1.40. Uso Avanzado de ARIA

La capa más básica de ARIA es ArDeviceConnection y sus subclases, las cuales manejan un bajo nivel de comunicación con el servidor del robot. En la parte superior de la capa de conexión, se tiene una capa de paquetes. ArBasePacket y ArRobotPacket, es decir los algoritmos básicos para la construcción de paquetes de comandos y los paquetes decodificados de información del servidor.

Por encima de la capa de paquetes se encuentra el paquete que maneja las clases, ArRobotPacketReceiver y ArRobotPacketSender, que envían y reciben paquetes hacia y desde el robot. Por último, en la parte superior de todas estas capas inferiores se encuentra ArRobot, que es el punto de encuentro de todas las cosas, y que puede ser usado en un formato muy básico sin ser muy notorio.

ArRobot se ha construido en base a tareas, acciones, estado de reflexión y así sucesivamente, todos los cuales pueden ser deshabilitados desde el constructor (ArRobot::ArRobot) y son ignoradas o reimplementadas.

Se debe notar que si todo lo que se hace es deshabilitar el estado de reflexión, se afecta el envío mediante ArRobot de los comandos de movimiento, no se recibe SIPs desde el robot, no se realizan ninguna de las actividades en las que ArRobot participa dentro del bucle de programación en ningún momento, por lo tanto es probable que no valga la pena construir un conjunto de tareas, pero si el usuario puede hacerlo se deja a su consideración.

Otra cosa que se puede destacar es que se puede llamar a ArRobot::loopOnce el cual ejecutará el lazo una vez y regresará. Esto se hace para que se pueda usar ARIA desde una estructura de control propia.

Si se usa loopOnce se puede encontrar que también es beneficioso para llamar a ArRobot::incCounter, para que el contador del bucle sea actualizado. También se puede llamar solo al ArRobot::packetHandler, ArRobot::actionHandler, o al ArRobot::stateReflector por su propia cuenta, ya que estas son las funciones más importantes, por ello se debe pensar que si se hace un lazo propio se debe probablemente llamar a ArRobot::incCounter de cualquier manera, de esta manera es como el sonar es conocido si es nuevo o no.

Se recomienda que cualquier cosa que se haga se deba usar el mismo tipo de enlace/bloqueo para que ARIA lo observe.

5.1.41. Archivos de parámetros del robot

Se encuentran en el directorio Aria/params, son genéricos, así como archivos de parámetros individuales que contienen información específica y por defecto del robot que ARIA usa para designar al robot e interpretar correctamente la información del servidor que el robot envía de vuelta al cliente.

Cada robot tiene un tipo y un subtipo (por ejemplo, “Pionner” y “p3dx-sh”), así como un nombre modificable por el usuario, insertados en los parámetros FLASH del robot (Para más detalles ver la sección anterior). Estos parámetros son enviados al cliente ARIA justo después de establecer la conexión cliente – servidor y que ARIA imprime su salida. ARIA

primero carga los parámetros desde el subtipo de archivo de parámetros, luego desde el archivo de parámetros en el cual se establecen y se resetean las variables globales en los contenidos de cada archivo.

De acuerdo a esto, el subtipo puede añadir o cambiar los parámetros dados por defecto y el otro archivo nombrado de parámetros que tiene la última decisión sobre las cosas. ARIA tiene el subtipo de archivo de parámetros en su directorio params. Un ejemplo de archivo de parámetros es p3dx.p (viejo Pioneer3 DX H8), p3dx-sh.p (nuevo Pioneer3 DX SH), p3at-sh.p (nuevo Pionner3 AT SH), p2de.p (Pioneer2 DE), peoplebot-sh.p (nuevo Peoplebot SH).

Para reemplazar un subtipo de archivo de parámetro, se debe copiarlo en un nuevo archivo llamado con el nombre del robot como se especifica en el parámetro FLASH, añadiendo el sufijo “.p”, y cambiando cualquier parámetro necesario para un robot específico. Por ejemplo, ARIA utiliza RobotRadius para determinar los límites de giro del robot en la mayoría de rutinas para la evasión de obstáculos.

El valor por defecto para el robot P2AT no tiene en cuenta los parachoques. De acuerdo a esto, se debe crear un nuevo archivo de parámetros que redefina el RobotRadius para este tipo de robot. ARIA utiliza los valores de la sección de factores de conversión del archivo de parámetros para transformar la información de los datos dependientes del servidor del robot en tasas y dimensiones normales.

Por ejemplo, el DistConvFactor convierte los datos de la posición del robot, medida en espacios del codificador, en milímetros. ARIA consulta la sección de accesorios del archivo de parámetros del robot, para determinar que accesorios del robot puede tener que no puede ser interpretado por otros medios.

Por ejemplo, los valores del parachoques del robot P2 aparecen en el SIP estándar en los valores de para, pero si un anillo de parachoques no se encuentra conectado, estos valores se mantienen flotantes y oscilan entre encendido y apagado. Una definición de accesorios

en el archivo de parámetros indica a ARIA si tiene o no que utilizar los valores del parachoques.

Finalmente, la sección del sonar del archivo de parámetros contiene información acerca del número de sonares y su geometría de modo que ARIA pueda relacionar las lecturas del sonar con la posición relativa al centro del robot.

5.1.42. Funcionamiento del archivo de parámetros

El archivo de parámetros es muy parecido en su formato a un archivo de Windows INI. Contiene secciones y palabras claves/datos pares. Los comentarios comienzan con un punto y coma. Un identificador de sección es una palabra clave entre corchetes, como por ejemplo: [ConvFactors]

Las palabras claves y los datos se encuentran separados por uno o más espacios en una sola línea, y puede incluir varias definiciones de valores de datos. Cada palabra clave tiene su propio comportamiento en la manera que analiza sus datos. Por ejemplo:
Keyword data1 data2 data3

La apariencia no importa para los identificadores de sección y los nombres de las palabras claves. Algunos parámetros pueden tener múltiples instancias en el archivo. Un buen ejemplo es la unidad de sonar. Por ejemplo:
SonarUnit 0 73 105 90
SonarUnit 1 130 78 41

5.1.43. Conexión rápida del robot o el simulador

El ArDeviceConnection es un objeto de comunicaciones de ARIA; el ArSerialConnection y el ArTcpConnection son sus herramientas internas más comúnmente utilizadas para manejar la comunicación de un robot MobileRobots o ActivMedia, o el simulador del robot. Estas clases no son específicas del dispositivo, sin embargo, se usa ArSerialConnection, para la instancia, o para la configuración de un puerto serial y

establecer la conexión con un accesorio del robot, como por ejemplo el accesorio de medición láser.

5.1.44. Apertura de la conexión

Después de crear y abrir la conexión de un dispositivo, se debe asociar con los controladores de manejo de ARIA, más comúnmente con el ArRobot::setDeviceConnection para el robot o el simulador.

Por ejemplo, en el inicio de un programa de ARIA, se especifica el dispositivo de conexión y se lo asocia con el robot:

```
  ArTcpConnection con;
  ArRobot robot;
```

Luego en el programa, después de inicializar el sistema ARIA (Aria::init(); es obligatorio), se establece el puerto de conexión a sus valores predeterminados (para TCP, el host es “localhost” y el número de puerto es 8101), y luego se abre el puerto:

```
con.setPort();
 if (!con.openSimple())
  {
    printf("Open failed.");
    Aria::shutdown();
    return 1;
  }
```

Las conexiones TCP y seriales tienen su propia implementación de apertura la cual no es heredada, pero tiene argumentos por defecto que hacen abierto el trabajo genérico para todos los casos predeterminados.

Y la apertura retorna un entero de estado el cual puede pasar a ser re-implementado y heredado. El ArDeviceConnection::getOpenMessage se usa con el fin de recuperar la

cadena de estado relacionada, la cual es útil en los reportes de errores al usuario sin necesidad de conocer sobre el dispositivo subyacente.

5.1.45. Conexión del robot cliente – servidor

Después de asociar el dispositivo con el robot, se lo conecta con el servidor del robot, por ejemplo con el ArRobot::blockingConnect o ArRobot::AsyncConnect, para establecer la conexión cliente servidor entre ARIA ArRobot y el microcontrolador del robot o el simulador. El método blockingConnect no regresa desde la llamada hasta que la conexión sea satisfactoria o falle:

```
robot.setDeviceConnection(&con);
if (!robot.blockingConnect())
{
  printf("Could not connect to robot... Exiting.");
  Aria::shutdown();
  return 1;
}
```

Los ejemplos anteriores se conectan con el simulador a través del socket TCP del PC. Se usa el tcpConn.setPort (host,port) para establecer el hostname del TCP o la dirección IP y su número de socket relacionado con otra máquina de la red. Por ejemplo, se usa tcpConn.setPort (“bill”, 8101); para conectar el simulador el cual está ejecutándose en la computadora de la red “bill” a través del puerto 8101.

Para conectarse con el robot a través del puerto serial se reemplaza el ArTcpConnection con; con el ArSerialConnection con (/dev/ttyS0 o COM1), u otro que se especifique con ArSerialConnection::setPort(), como el con.setPort(“COM3”). En algún momento, se puede abrir el puerto con el con.open() más detallado.

5.1.46. Lectura, escritura, cierre y marcas de tiempo de la conexión

Las dos funciones principales de la conexión de un dispositivo son ArDeviceConnection::read y ArDeviceConnection::write. Bastante simples. El ArDeviceConnection::close también es heredado y muy importante. Probablemente no se

requiera usar la lectura o escritura directa hacia el dispositivo del robot, aunque no implica que no se pueda. Más bien, ArRobot provee un host con métodos convenientes que los paquetes de comandos del robot, y que reúnen y distribuyen los diferentes paquetes de información del robot, de modo que no se tiene que atender estos detalles.

Todas las subclases ArDeviceConnection cuentan con el soporte para las marcas de tiempo (ArDeviceConnection::getTimeRead). Con la conexión del robot, las marcas de tiempo se limitan a especificar en qué tiempo las SIPs del robot llegan, lo cual puede ser de mucha utilidad para interpolar la ubicación del robot de manera más precisa.

5.2. SonARNL[8]

Se puede crear, editar y usar mapas y planos de pisos para aplicaciones robóticas avanzadas, incluyendo localización y navegación basadas en el sonar y el láser, tal como se puede visualizar en la figura 5.31. Todos los programas, incluyendo ARIA con ARNetworking, MobileSim™, Mapper3-Basic, SonARNL y el cliente GUI MobileEyes™, Mapper3 con ARNL son las últimas herramientas usadas para localización y navegación basada en láser.

5.2.1. Modos de Operación

Se puede operar el robot Pioneer3 en uno de los 4 modos:

- ✓ Servidor
- ✓ Manejo manual
- ✓ Mantenimiento
- ✓ Independiente

5.2.2. Modo Servidor

El microcontrolador Renesas SH2 viene con una memoria flash de 128K reprogramable y una memoria RAM dinámica de 32K, el cual tiene instalado su software ARCOS.

[8] Pioneer3 Operation Manuals

ARCOS brinda la ventaja de una tecnología moderna de cliente servidor, para control del robot, lo cual permite desarrollar avanzadas tareas robóticas móviles.

El modo servidor provee acceso fácil a las funcionalidades del robot mientras trabaja el software de alto nivel en una computadora host.

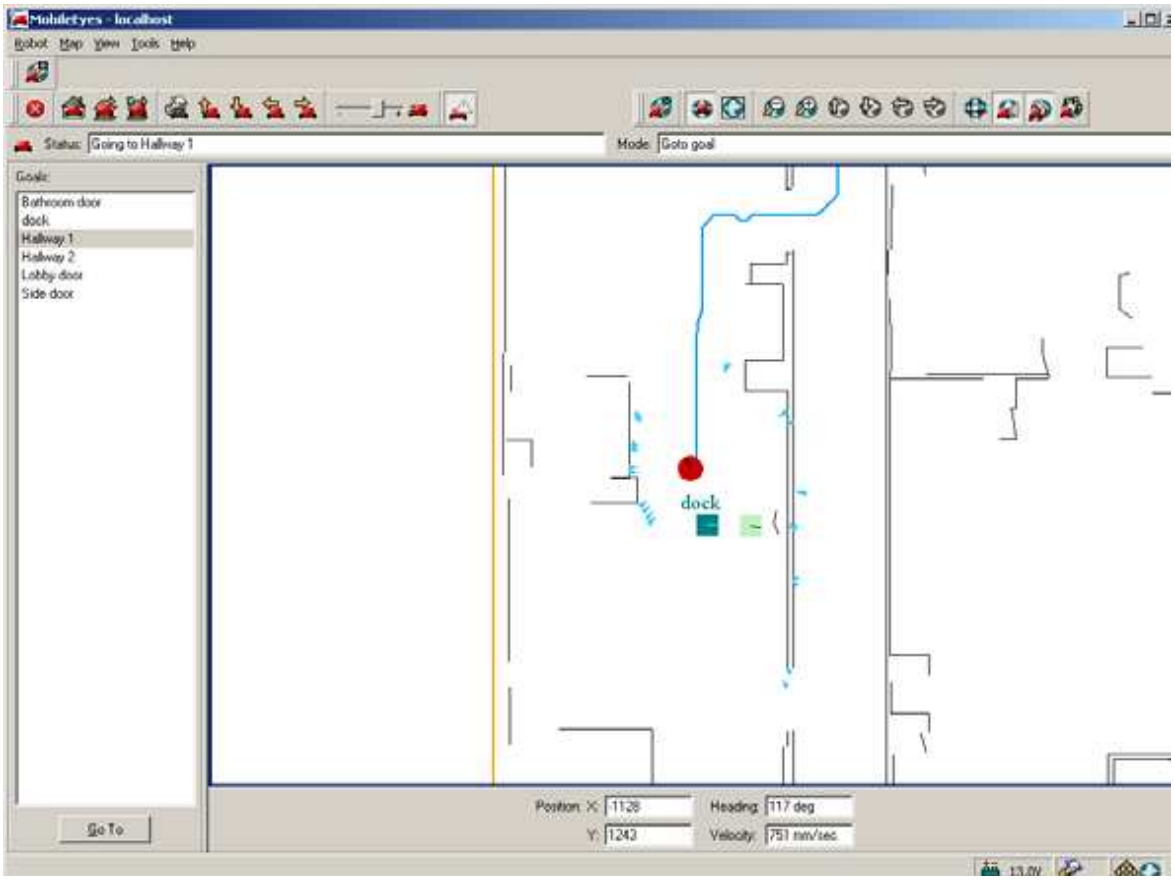

Figura. 5.31. **Mapa Detallado**

5.2.3. Modo Mantenimiento en Independiente

Para experimentos de operación a nivel del microcontrolador para las funciones robóticas, se puede reprogramar la FLASH interna para operaciones directas e independientes del robot.

Para ello se proveen los medios para cargar y depurar las características internas, pero no se puede cambiar el código interno del microcontrolador para trabajar en modo independiente.

Los elementos que se proveen para la reprogramación del la FLASH del microcontrolador SH2 pueden ser utilizadas para actualizar el software ARCOS del robot. En el modo especial de mantenimiento, incluso se puede ajustar los parámetros de operación del robot usados por defecto en el inicio o al resetear el robot.

5.2.4. Modo Manejo Manual

Finalmente se provee el software interno y el hardware del microcontrolador que permite manejar el robot desde un Joystick conectado al robot mientras no se encuentre conectado con otro cliente.

5.2.5. Sistema de Localización y Direccionamiento

ARNL es un software de desarrollo para la localización y navegación inteligente de los robots dentro de un mapeo para ambientes cerrados. Es el complemento de ARIA en conjunto con el sistema de red ArNetworking para facilidad en la visualización y control de clientes remotos.

También se puede programar y operar además del uso del ARIA, o utilizar un simulador para ARNL. Existen dos librerías para ARNL.

La primera es llamada ARNL y utiliza datos obtenidos de un rango para el escaneo de láser o de los sensores tipo sonar para localizar y detectar obstáculos, todo eso incluido en la navegación láser del sistema.

La segunda librería es llamada SONARNL y es la versión de ARNL que incluye solo la localización y navegación sonar. Utiliza la información del sensor de rango junto con el dato odométrico del robot para realizar el respectivo mapeo de un ambiente.

Los mapas incluyen áreas prohibidas, metas y posiciones de “home”. Se puede construir mapas usando el software Mapper3, cuyo único requisito es tomar en cuenta todas las

medidas reales del ambiente o lugar a construir y ubicar correctamente los diferentes obstáculos.

5.2.6. Arquitectura y Operación

SonARNL desarrolla dos tipos de acciones, localización y navegación siguiendo una cierta dirección de un mapa. Cada una de estas tareas está desarrollada dentro de su propio ambiente mientras se ejecute el programa.

La red que trabaja con el sistema basado en ARIA está separada para trabajar con SonARNL sobre una red basada en TCP/IP. Las respectivas librerías de las clases contiene todo los métodos que la aplicación necesita para operar, obtener información y determinar rangos.

Para programar respectivamente ARNL y SonARNL utiliza las librerías Arnl/Aria.h, Arnl/Arnl.h y si se necesita Arnl/ArNetworking.h que se encuentra dentro de Arnl/lib. En las aplicaciones de SonARNL se necesita incluir al menos tres objetos para que esté disponible una localización segura: ArRobot para los detalles del robot, ArSick para los rangos y ArMap para los detalles del mapa.

Para direccionamiento y evasión de obstáculos se necesita cuatro objetos: ArRobot para el robot, ArSick para el láser, ArRangeDevice para el sonar y ARMap para los detalles del mapa.

Una vez que la localización y direccionamiento están inicializados y disponibles para su uso, SonARNL utiliza parámetros iniciales para arrancar el robot.

Para cambiar las características del robot y su comportamiento se necesita resetear estos parámetros en los respectivos módulos dentro de params/arnl.p.

5.2.7. Localización[9]

El Sistema de Localización y Navegación Robótica Avanzada ARNL utiliza asincrónicamente la clase ArLocalizationTask para localizar un robot dentro de un mapa en un espacio de puertas adentro.

Previamente se crea y edita el mapa a ser utilizado con el software Mapper3 o un programa similar. El mapa deberá contener las medidas exactas y los obstáculos que posee el ambiente; el programa Mapper3 tiene la facilidad de ubicar puntos de referencia como diferentes metas, puntos de llegada y de salida, lugares prohibidos, etc.

ArLocalization utiliza los sensores tipo sonar para ubicarse y realizar las diferentes tareas como evasión de obstáculos. Mediante un algoritmo llamado Monte Carlo ubica el robot dentro un mapa dado en su ambiente de operación, transforma información odométrica del robot y el rango de operación del láser para su funcionamiento.

ArLocalizationTask necesita haber sido inicializado los objetos ArRobot, ArSick y ArMap respectivamente. Para utilizar el modo de localización se puede seguir los siguientes pasos:

Crear la clase ArLocalizationTask incluyendo las clases ArRobot y ArSick y crear un mapa del ambiente del robot, explícitamente como archivo ArMap, por ejemplo:

```
  // Crea la tarea de localización
  ArLocalizationTask *myLocTask = new ArLocalizationTask(myRobot, mySick,
mapname);
```

Inicializar la localización del robot dentro del mapa:

```
  // Establece la posición inicial en las coordenadas (0,0,0
  myLocaTask->localizeRobotAtHomeBlocking();
```

[9] Ejemplo de Localización usando SonARNL, ver Anexo C

Se puede resetear a la posición “home” (punto inicial ubicado previamente en el mapa) en cualquier momento antes o después de la inicialización utilizando localizeRobotAtHomeBlocking() o sus funciones referentes

```
// Actualiza la posición de localización actual
myLocaTask->forceUpdatePose(pose, ns, xstd, ystd, tstd);
```

Si después se vuelve a utilizar la tarea ArLocalizatinTask, automáticamente relocaliza el robot en el mapa. Para conservar los datos de referencia, se hace una actualización de datos para la localización sol cuando el robot se ha movido dentro de sus límites de translación y rotación. Sus parámetros de origen están dentro de +-200 mm de traslación y +- 5 grados de rotación, y pueden ser cambiados dentro de los parámetros de configuración.

5.2.8. Direccionamiento[10]

ArPathPlanningTask es el módulo de navegación de SonARNL y trabaja como una tarea asíncrona con la infraestructura del control de ARIA. Su trabajo es planear o moverse a la más corta y rápida dirección para manejar el robot desde su actual posición hasta cualquier otro punto alcanzable del mapa previamente construido en el software Mapper3 o similar.

Navega hacia una meta específica utilizando ArMap el cual indica obstáculos sensibles, ya sean paredes, mueblería, áreas o líneas prohibidas, puntos de meta, de salida, de llegada, de inicio (home).

Dentro del mapa busca las celdas lo suficientemente lejanas de obstáculo a obstáculo para tener una idea global de su actual dirección y movilizar el robot hacia una meta específica.

Activa el ArAction para manejar el robot durante todo el trayecto que le lleve a ese punto y mientras navega utiliza un método dinámico de redireccionamiento para evitar obstáculos no mapeados. Se puede ajustar los parámetros de direccionamiento local y global.

[10] Ejemplo de Direccionamiento usando SonARNL, ver Anexo D

ArPathPlanningTask necesita funcionalidad del ARIA para simular instantáneamente el direccionamiento y la localización en conjunto con ArLocalizationTask o ArSonarLocalizationTask.

```
  myPathPlanningTask = new ArPathPlanningTask(myRobot, mySick, mySonar,
myMap);
    // En el ejemplo establece la posición inicial.
    myPathPlanningTask->pathPlanToPose(ArPose(0,0,0), true);
```

Se puede enviar al robot a una meta o una posición relativa por nombre dentro del objeto ArMap como ejemplo Dock:

```
    // Envia al robot a un punto dock
    myPathPlanningTask->pathPlanToGoal("Dock");
```

Una vez que la meta esta seteada, ArPathPlanningTask automáticamente maneja al robot eficientemente hacia esa meta.

```
    ArFunctor1<ArPose> goal_done(goalDone); // Define goalDone
    ArFunctor1<ArPose> goal_failed(goalFailed); // goalFailed
    ArFunctor1<ArPose> goal_interrupt(goalInterrupt); // and
goalInterrupted handlers.
    myPathPlanningTask->addGoalDoneCB(&goal_done); // Implement each as
task callbacks
    myPathPlanningTask->addGoalFailedCB(&goal_failed);
    myPathPlanningTask->addGoalInterruptedCB(&goal_interrupt);
    ArPathPlanningTask::PathPlanningState s = myPathPlanningTask-
>getState();
    char[512] text;
    myPathPlanningTask->getStatusString(text, sizeof(text));
    printf("Planning status: %s.\n", text);
```

5.2.9. Localización Sonar

La clase ArSonarLocalizationTask trabaja asincrónicamente para localizar un robot en un mapa conocido en un ambiente cerrado. Además, crea y edita una línea de mapa con el software Mapper3 o similar.

ArSonarLocalizationTask implementa un algoritmo de localización llamado Monte Carlo el cual localiza al robot dentro una línea en el mapa de su ambiente de operación como derivado de la información odométrica del robot y del rango de operación del sonar.

La localización significa inicializar y recorrer el robot utilizando los objetos ArRobot, ArSonarDevice y ArMap. Se siguen los siguientes pasos para utilizar la localización sonar:

Se crea la clase ArSonarLocalizationTask inicializado con ARRobot yArSonarDevice, por ejemplo:

```
  // Create the localization task (it will start its own thread here)
  ArSonarLocalizationTask *myLocaTask  = new
ArSonarLocalizationTask(myRobot, mySonar, mapname);
```

Inicializar la localización del robot dentro del mapa:

```
  // Set the initial pose to the robot's "Home" position from the map, or
  // (0,0,0) if none, then let the localization thread take over.
  myLocaTask->localizeRobotAtHomeBlocking();
```

Se puede forzar en cualquier momento a que regrese a su punto inicial HOME:

```
  // Force the current localization pose
  myLocaTask->forceUpdatePose(pose, ns, xstd, ystd, tstd);
```

Para una mejor comprensión de cómo se realiza la localización y el direccionamiento, estos serán empleados en el ejemplo de aplicaciones correspondiente al seguimiento de trayectorias.

5.3. ACTS[11]

El Sistema de Rastreo de Color ActivMedia (ActivMedia Color Tracking System, ACTS) es un software que en combinación con una cámara a color y un capturador de tramas permite rastrear hasta 320 objetos a color. ACTS es muy usado como un sensor de visión

[11] Activ Media Color Tracking System User Manual

para aplicaciones en robótica y para la identificación de objetos, vigilancia y muchas otras aplicaciones de máquinas de visión.

5.3.1. Componentes Básicos

- Manual de Usuario de ACTS
- EZ-Train, Consola Principal
- Servidores de rastreo de blobs de ACTS
- Soporte de gráficos FLTK, Fast Light Tool Kit, (Kit de Herramientas de Luz Rápida) y bibliotecas de software.

5.3.2. Componentes Suministrados por el Usuario

- Configuración de cámara a color con opciones de paneado, inclinación hacia arriba y abajo, y zoom:
    - Cámara con sistema de codificación NTSC (National Television System Commitee, Comisión Nacional de Sistemas de Televisión) o PAL (Phase Alternating Line, Línea de Fase Alternada)
    - Tarjeta de video VCE de Imperx Inc. y controladores respectivos.
- Herramientas de desarrollo C++
- Interfaz avanzado de Robótica ARIA para Aplicaciones

5.3.3. Framegrabbers para Windows

Para los usuarios del sistema operativo Microsoft Windows de 32-bit, ACTS soporta cualquier framegrabber (capturador de tramas) o sistema de imagen (tarjeta de video) que use la interfaz de Video para Windows (VFW). Adicionalmente, la placa de captura de Imagenation PXC200 y los respectivos controladores de la misma compañía pueden ser usados. Para laptops, una tarjeta de video Imperx VCE PCMCIA es la más usual.

Nota: Para todas las versiones ACTS, hace uso de las Herramientas de Luz Rápida (FLTK, Fast Light Tool Kit) para la interfaz de usuario gráfica. Las bibliotecas requeridas vienen incluidas con ACTS y son parte del proceso de instalación.

5.3.4. Características Generales

- Soporte para la interfaz de Video para Windows (Video for Windows, VFW).
- Aplicación del servidor SAV (Servidor de Audio y Video) para visualización remota en tiempo real de imágenes comprimidas en JPEG.
- Ventana de parámetros de vídeo independiente para ajustar diferentes tipos de parámetros.
- Parámetros de video individual se pueden pasar en la línea de comando.

5.3.5. Instalación

Actualmente ACTS se puede instalar en todas las versiones de RedHat para Linux o en Microsoft Windows para PCs. Se puede entrenar ACTS y desarrollar simples pruebas sin usar la cámara o el capturador de tramas. Se utiliza como componentes mínimos una tarjeta de video, cámara a color, y el controlador respectivo. ACTS trabaja mejor en modo de video de 24-bits, sin embargo los modos de 8-bits y 16-bits pueden ser usados sin problemas.

Ninguna de las ventanas en ACTS es más grande que 640x480 pixeles así que, las configuraciones de ventanas más básicas (mímino 8-bits), trabajan correctamente. Al correr el archivo ejecutable de instalación de ACTS, se guarda la información en la dirección:

C:\Program Files\ActivMedia Robotics\ACTS

Para la tarjeta de video Imperx VCE, se guarda el controlador VCE_Tools.dll dentro de la siguiente dirección:

C:\Program Files\MobileRobots\ACTS\bin

Alternativamente se copia el mismo archivo mencionado anteriormente dentro del directorio del sistema de Windows, Windows\System (WINNT\System32), donde

eventualmente será encontrado por el sistema operativo y compartido por todas las aplicaciones incluidas las de ACTS.

5.3.6. Ejemplo de Demostración

En la versión demostrativa de ACTS no puede conectarse con otro cliente distinto del propio EZ-Train de ACTS. Una vez que corra el programa de ACTS, se abren dos ventanas: la consola EZ-Train y una ventana de gráficos asociados que contienen información de compra. Una tercera ventana contiene texto DOS que también aparece con las versiones de Windows.

Se despliega el menú File de EZ-Train y se escoge *Load Image* para cargar la imagen. Desde el cuadro de diálogo resultante, se selecciona su propia imagen o la imagen bin/test.ppm preparada, como se observa en la figura 5.32. Alternativamente, al dar clic en el botón Live Video de la consola EZ-Train se activa la captura de video.

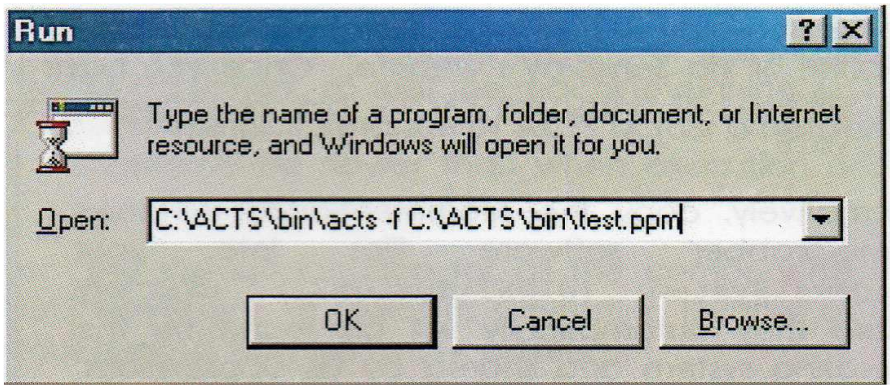

Figura. 5.32. **Inicio de ACTS con una Imagen Estática**

5.3.7. Modo de Entrenamiento en modo de Imágenes Estáticas

Si no está instalada una tarjeta de video o una cámara, se inicia ACTS con la imagen que se eligió y aparecerá en la ventana de la imagen asociada con la consola EZ-Train. Luego se maneja EZ-Train como lo haría con una imagen de video en vivo.

5.3.8. Opciones de Carga de Inicio y Preferencias

ACTS tiene opciones de carga de inicio para localizar los datos de los varios entornos operativos y diferentes condiciones, como se observa en la tabla 5.59. También hay múltiples ejecutables en el directorio bin dependiendo de la plataforma y versión.

Opción	Argumento	Significado
Ninguna	(none)	Predeterminado; iniciar el modo de cliente EZ-Train sin los canales o configuración definida. La ventana de imagen de instrucción contiene video de capturador de tramas en vivo disponible, o una imagen estática de tutorial.
-c	**Config filename**	Carga el nombre de la configuración <nombre del archivo>
-d	**Devname**	Selecciona el dispositivo apropiado para la placa de captura.
-f	**Image filename**	Carga un archivo de imagen .ppm en lugar de utilizar la placa de captura
-G	**Driver type**	Tipo de driver especificado (use –h para ver los tipos de soporte
-h	None	Imprime el listado de ayuda (así como la lista de tipos de drivers soportados)
-H	**Height**	La altura de las imágenes en las filas
-i	**Invert**	Imagen invertida verticalmente
-n	**Number**	Especifica el número de canal
-p	**Portnum**	Puerto TCP para comunicarse
-P	**Parameter**	Muestra los parámetros de la cámara admisibles usados en la opción –s
-R	**Frames/sec**	Especifica la velocidad máxima de captura de video
-s	**Set**	Fija un <param> de la placa de captura con <value> Ejemplo: -s brillo 0.3 contraste 0.1 Use la opción –P para ver qué opciones están disponibles para el tipo de cámara seleccionada
-S	**Disable SAV**	Desactiva el servidor del software A/V
-t	**Config filename**	Carga el archivo de configuración <filename> y corre ACTS sin los gráficos (sólo Linux)
-V	**PAL | NTSC**	Especifica los formatos PAL o NTSC. El predeterminado es NTSC
-w	**Píxel width**	El ancho mínimo que un archivo grande debe ser antes de que éste sea rastreado
-W	**Image width**	El ancho de la imagen en columnas
-x	None	Usa la placa de captura adaptable a PXC200 con el driver bt848. ADVERTENCIA! Úselo solamente si un PXC está instalado; de otro modo, ACTS podría hacer que su ordenador deje de trabajar.

Tabla. 5.59. **Opciones de Carga de Inicio de ACTS**

Estas y otras opciones de operación son también almacenadas en el directorio principal del usuario o en el archivo de preferencias del directorio de ACTS como ACTS.pref, el cual puede ser modificado con un editor de texto común como se observa en la tabla 5.60.

Los servidores de ACTS procesan los protocolos de información de video, para identificar y rastrear los objetos coloridos y comparten la información extraída con los clientes. El

software cliente EZ-Train permite capacitar ACTS para encontrar los objetos coloridos y que los servidores realicen el seguimiento.

Nombre del archivo de imagen	
Nombre del archivo de configuración	
Recorrido mínimo de largo y ancho	5
Ancho de la imagen	160
Altura de la imagen	120
Dirección del dispositivo transportador de datos	5001
Tipo de placa de captura	vfw (Windows)
Dispositivo de placa de captura	-1 (Windows)
Cuadros por segundo	30
Graficos de muestra	True
Cámara PAL	False
Placa PXC	False
Imagen invertida	False
Canal de placa de captura	1

Tabla. 5.60. **El archivo ACTS.pref con valores predeterminados**

5.3.9. Funcionamiento de ACTS

ACTS necesita una cámara a color y una tarjeta de video para tomar fotos instantáneas digitales y procesarlas. Una vez que el dato de video es capturado, los servidores de ACTS ejecutan y se encargan de la operación conocida como *segmentación de color* para clasificar los elementos digitales de color, es decir los pixeles, dentro de dos grupos: interesante y poco interesante.

Los pixeles interesantse: rojo, verde, y azul (RGB: Red, Green, Blue), componentes elementales de color coinciden con los almacenados en cualquiera de las tablas de búsqueda de más de 32 diferentes colores. Los pixeles poco interesantes, por supuesto, no corresponden a ninguno de los colores RGB de ninguna de las tablas.

Una vez identificados, los pixeles interesantes son agrupados dentro de un archivo multimedia, exactamente regiones coloridas de pixeles adyacentes cómo mínimo diez pixeles de ancho. Los archivos multimedia más pequeños que 10 pixeles, generalmente son ruido y deben ser ignorados.

Los servidores de ACTS caracterizan cada objeto colorido de acuerdo a su canal equivalente, posición en el cuadro de video, tamaño y dimensiones, y comunica esa información a los clientes, como EZ-Train.

Un usuario de ARIA puede controlar un robot móvil y utilizar la información del archivo multimedia (blob) para guiar a la máquina hacia un objetivo de color en movimiento.

5.3.10. Consideraciones de Desempeño

Actualmente ACTS puede rastrear simultáneamente hasta 320 blobs independientes con una velocidad máxima de la tarjeta de video de 30 cuadros por segundo (diez blobs en cada uno de los 32 canales de color independientes).

Depende de una variedad de factores para que ACTS encuentre y rastree los respectivos blob de un archivo multimedia, lo que incluye recursos computarizados (RAM, velocidad del procesador, calidad del capturador de tramas, etc.); variables ambientales, así como variaciones en la luminosidad (iluminación y sombras); sutiles variaciones de color entre objetos similares; colores de fondo; y muchas otras complicaciones del mundo real.

El factor más importante en el funcionamiento de ACTS, sin embargo, es qué tan bien se entrene el sistema para los objetos de color que se desee rastrear.

5.3.11. Secuencia Típica de Operación

1. Encender el sistema y la cámara para ver los objetos de rastreo en el medio.
2. Iniciar EZ-Train en el modo de video en vivo.
3. Seleccionar uno de los 32 canales para entrenar en la consola EZ-Train.
4. Cargar o crear una nueva Tabla de Búsqueda (LUT) para ese canal.
5. Utilizar herramientas de EZ-Train para seleccionar objetos de color a rastrear.
6. Si es posible, reposicionar los objetos dentro del entorno y agregar/remover/ los colores desde el canal de entrenamiento.
7. Refinar la selección de color con el modo de entrenamiento manual.
8. Guardar el canal LUT.

9. Repetir los pasos del 3 al 8 para cada tipo diferente de objeto para rastreo.
10. Preparar un archivo de configuración de tiempo de arranque que contenga los nombres de los archivos LUT guardados.
11. Cerrar u ocultar la ventana de EZ-Train
12. Conectar la información del seguimiento de la imagen con los clientes de aplicación, tal como un robot móvil controlado con ARIA.

5.3.12. Usando ACTS paso a paso

5.3.12.1. Paso 1. Arrancar ACTS

Iniciar ACTS el cual está conectado al cliente EZ-Train, con Video en vivo (si está disponible), y con la configuración predeterminada (ninguna) y los archivos LUT.

Dar doble clic sobre el ícono de acts(.exe) en el directorio:

C:\Program Files\MobileRobots\ACTS\bin, o use el menú Start en el cuadro de diálogo Run…

Una vez que ACTS inicie, este se abrirá y desplegará dos o tres ventanas:

- La consola EZ-Train, ver figura 5.33.
- Otra ventana de EZ-Train que contiene una imagen.
- Una tercera ventana, una abierta por Windows, que contiene los mensajes del sistema informativos.

5.3.12.2. Paso 2. Cargar la Imagen del Tutorial

Cargar la imagen estática del tutorial con el menú File de la consola EZ-Train al ítem Load Image, y seleccionar test.ppm del cuadro de diálogo, ver figura 5.34. La versión Demo de ACTS se abre con un mensaje promocional en la ventana de entrenamiento.

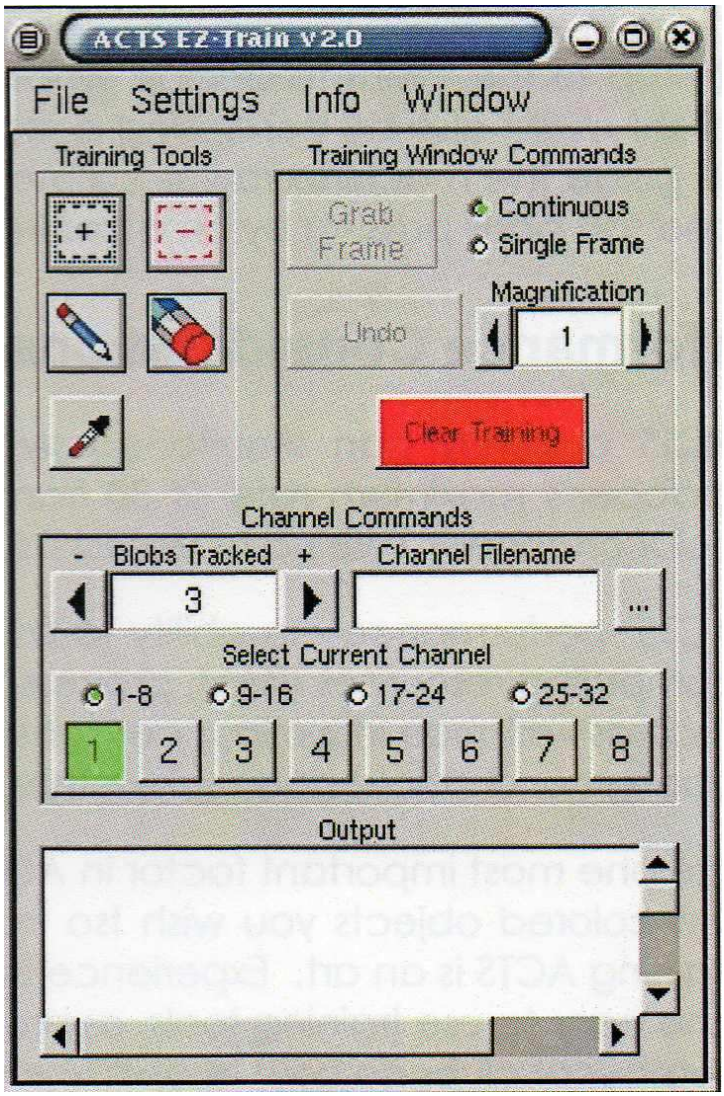

Figura. 5.33. **La consola EZ-Train**

La ventana de entrenamiento arranca a una resolución de 160x120. Usar la sección de Magnificación de la consola EZ-Train para incrementar el tamaño de la imagen hasta 6 veces.

5.3.12.3. Paso 3. Seleccionar un Canal y Número de Blobs Rastreados

Se mostrará una imagen, hay un libro rojo y un vaso rojo. Primero, seleccionar el canal 1 dando clic en su botón correspondiente en la consola EZ-Train. Desde donde hay solo 2 objetos que se va a rastrear, dar un clic en las flechas sobre Number of Blobs Tracked en la consola EZ-Train hasta que el número sea 2.

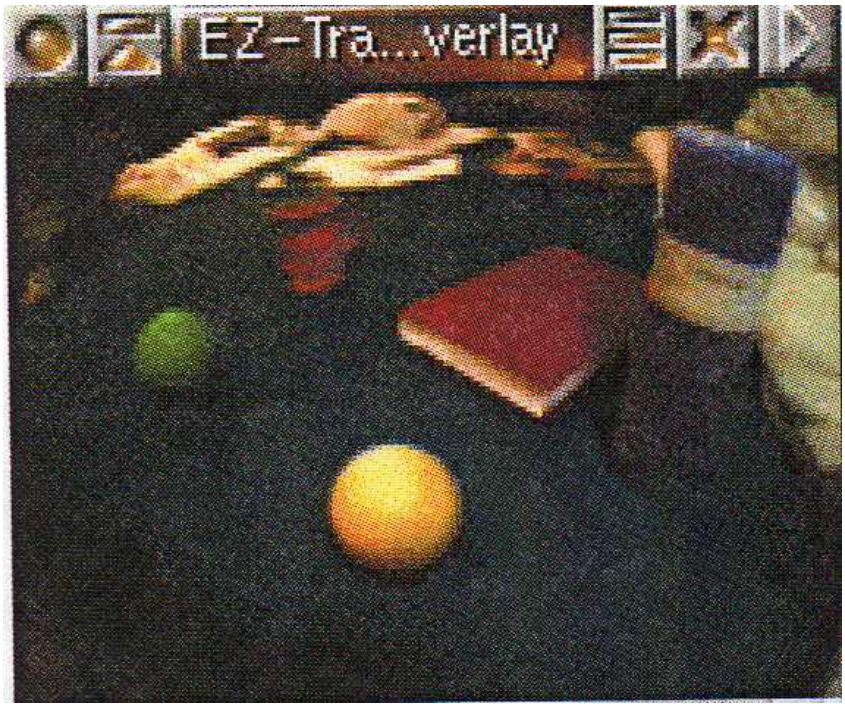

Figura. 5.34**. Imagen de Muestra de la EZ-Train**

El canal seleccionado debe ser el mismo con el cual se va a trabajar, ya sea para el uso del tutorial o para cualquier tipo de proyecto. Ver figura 5.35.

5.3.12.4. Paso 4. Objetos seleccionados

Hacer clic en el botón Add  en la consola EZ-Train para iniciar los colores agregados a su canal de rastreo. En la ventana de entrenamiento, hacer clic y arrastrar el ratón para crear un rectángulo dentro del libro rojo.

Cambiar la forma de ese rectángulo para seleccionar tantos pixeles como sea posible dentro del libro rojo. Ser cuidadoso de no seleccionar algún pixel fuera del libro.

Soltar el ratón y los pixeles dentro de la caja roja automáticamente serán añadidos a los colores del canal actual. La ventana de entrenamiento está en modo Overlay (Superposición), así los blobs encontrados por la EZ-Train son destacadas en la imagen.

Cambiar el tamaño de la imagen para hacer fácil la selección de los pixeles individuales.

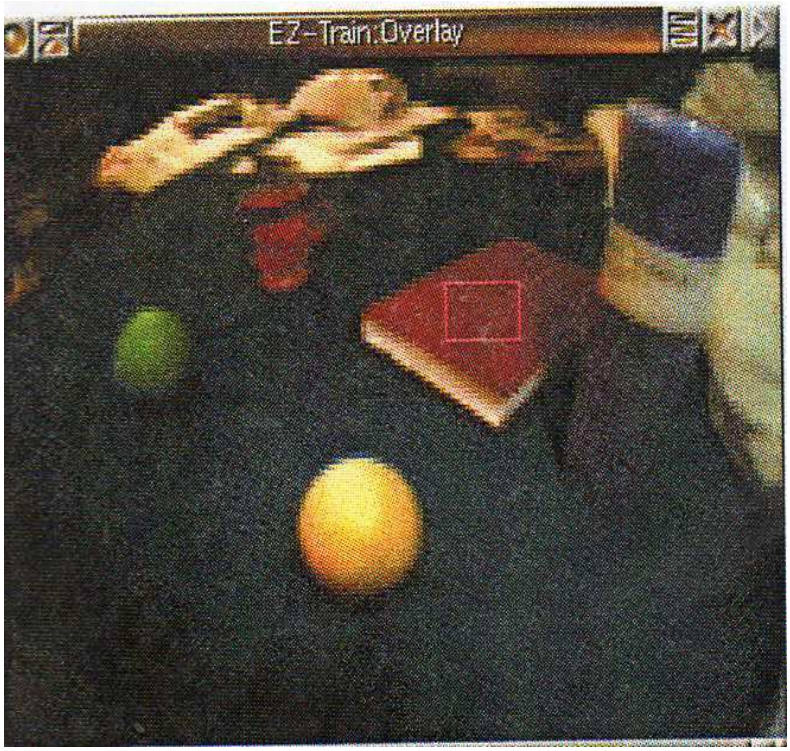

Figura. 5.35. **Herramienta de rectángulo seleccionando colores**

5.3.12.5. Paso 5. Remover Colores y Deshacer

Similar a la herramienta Add, hacer clic y operar la herramienta tool  para seleccionar áreas de color que no se quiera incluir en el canal. Al mismo tiempo que se quite los colores, previamente los blobs que destaquen también pueden desaparecer. Cambiar el tamaño de la imagen para trabajar más cerca con el color.

Se puede hacer clic también en el botón Deshacer (undo) en la consola EZ-Train para revertir la última operación que modificó los datos del canal. Hacer un clic en el botón Borrar instrucción (clear training) para borrar completamente datos del canal y empezar las instrucciones nuevamente, ver figura 5.36.

5.3.12.6. Paso 6. Cambiar las vistas de instrucción

El modo Superposición (overlay) en la ventana de instrucción puede ser confuso. Hacer clic derecho dentro de Training Window o presione la tecla "2" mientras el apuntador del ratón está dentro de la ventana de instrucción para darle una vista aérea de los blobs, llamado modo Thresh.

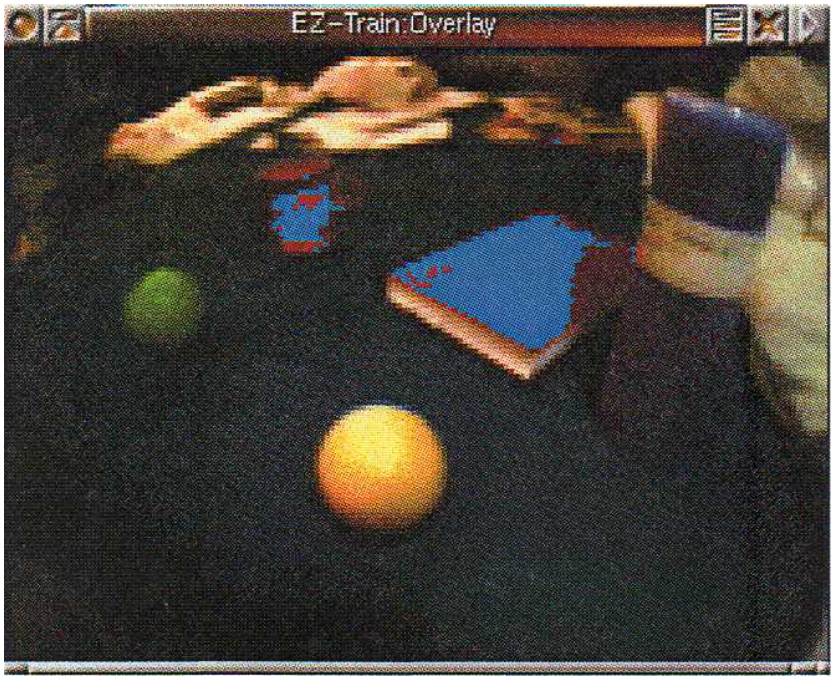

Figura. 5.36. **Blobs extraídos Cubriendo la ventana de Instrucción**

El programa despliega los pixeles interesantes, colores que aparecen en el canal actual, en blanco y todos los demás en negro, como se puede observar en las figuras 5.37 y 5.38. Los blobs, las regiones continuas de los pixeles interesantes, son destacados con un cuadrado púrpura. El centro de giro de cada blob es un pixel púrpura individual.

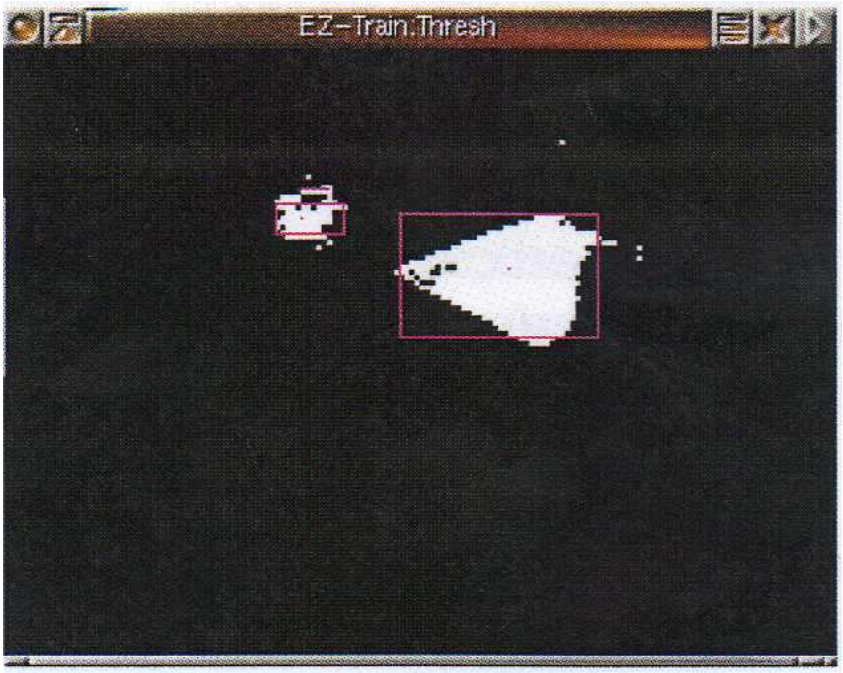

Figura. 5.37. **Vista Aérea**

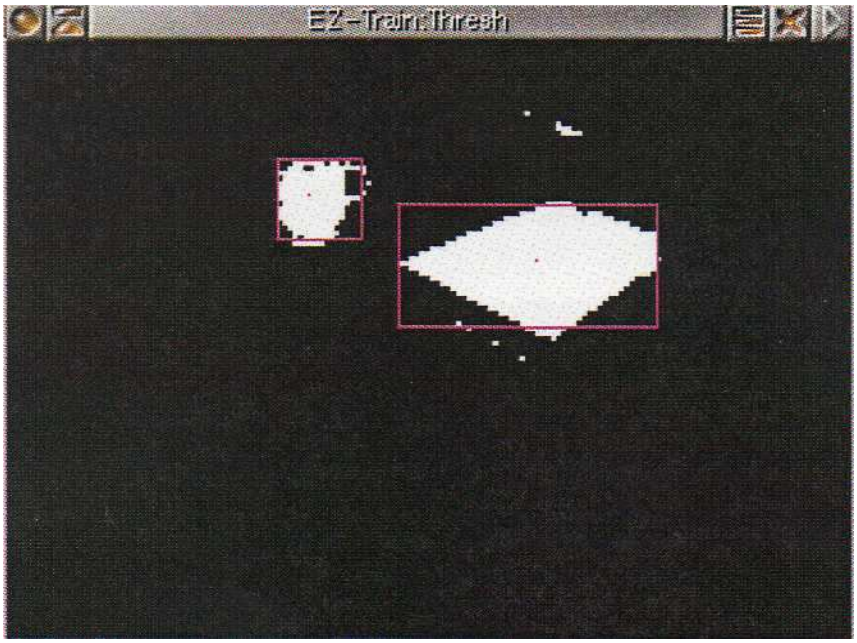

Figura. 5.38. **Pulir la instrucción para rastrear el objeto entero**

Para volver al modo Overlay, hacer clic derecho dentro de la ventana de instrucción 3 veces o presionar la tecla "1" mientras el ratón está dentro de la ventana Training Window.

Se debe comparar la imagen Thresh con la imagen original, si no se ha capturado todos los pixeles que componen el libro o el vaso. Agregar más pixeles (Add) para definir mejor los objetos como blobs.

5.3.12.7. Paso 7. Ver datos de Blob

Se puede observar que ACTS no toma en cuenta los objetos pequeños a pesar de que estos contienen pixeles interesantes, esto se debe a dos factores:

Primero, ACTS solo rastrea los números seleccionados de blobs (dos para este ejemplo) desde los más grandes hasta los más pequeños objetos en la imagen (tamaño elegido por el área en pixeles).

Segundo, blobs muy pequeños, menores a los pixeles son ignorados por ACTS, y son indicadores de ruido.

Al seleccionar la opción Estadísticas de los blobs (Blob stats) del menú Info de la consola EZ-Train se observa en la figura 5.39. El campo de salida inferior de la consola despliega las estadísticas de los blob rastreados actualmente, incluyendo el área 'a', centro de las masas coordenadas 'x' y 'y', y las filas y columnas de la caja que rodea al blob.

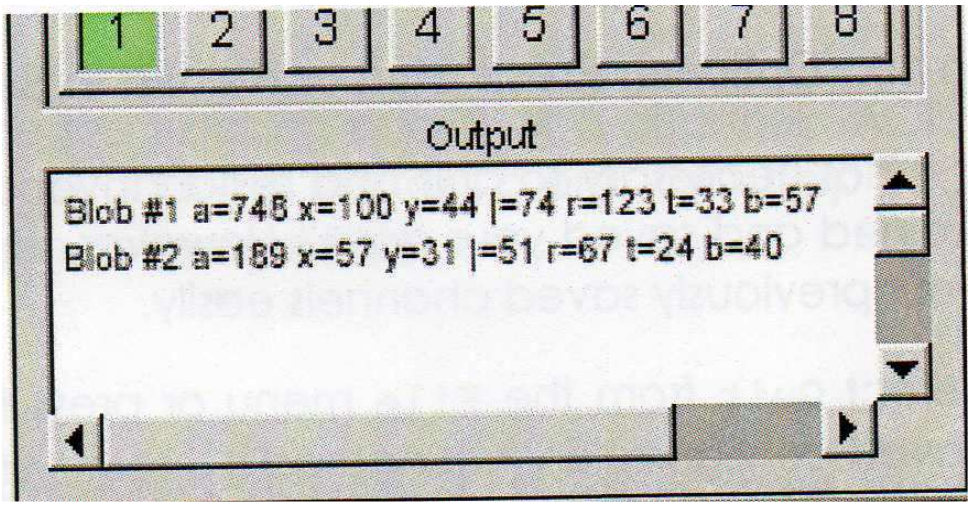

Figura. 5.39. **Estadísticas de los blob**

5.3.12.8. Paso 8. Guardar los datos del canal

Una vez finalizado el entrenamiento en un canal particular, guardar la tabla de búsqueda de color (LUT) en un archivo del disco. Fijarse en el nombre del archivo predeterminado, channel11.lut, en la consola. Ese nombre no guarda los datos.

Seleccionar Save Channel del menú File en la consola EZ para guardar el archivo LUT. Use Save channel as… para seleccionar un directorio diferente o cambiar el nombre y hacer clic en el botón ok para guardar el canal como en la figura 5.40.

5.3.12.9. Paso 9. Crear una Configuración de Activación

Las configuraciones de activación en ACTS son colecciones guardadas de canales y parámetros de operación. Comúnmente usados para definir los entornos de operación para ACTS en el servicio para un cliente de aplicación, así como en el modo de servidor, una configuración de activación de ACTS también puede ser cargada para edición con el cliente de EZ-Train.

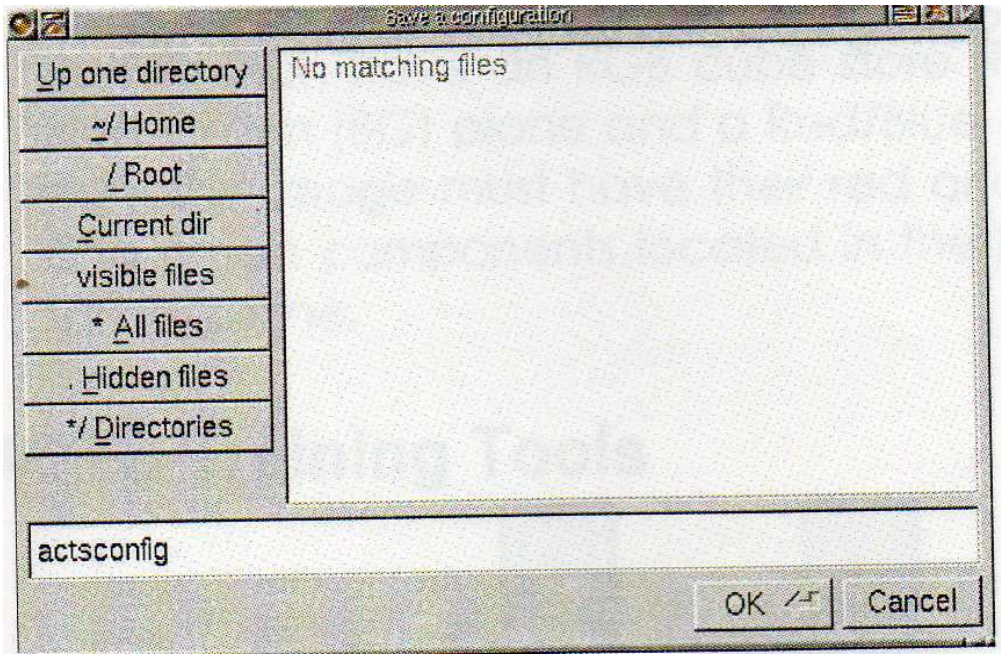

Figura. 5.40. **Guardando una configuración de activación**

Si se ha definido un canal (channel1.lut) y colocado su número de blob-tracking. Se guarda esa configuración como un archivo de configuración de ejecución. Se selecciona Save Runtime Config desde el menú File en la consola EZ-Train.

En el cuadro de diálogo, escoger el directorio de destino y si se desea se cambie el nombre del archivo, luego se hace clic en ok para guardar la configuración.

Nota: Guardar un archivo de Config de ejecución no guardará los archivos de canal individuales. Estos necesitan ser guardados separadamente en adición al archivo de ACTS Config.

5.3.12.10. Paso 10. Inicio Fresco o Reinicio en Modo Servidor

Una vez que haya inicialmente entrenado y guardado los datos no es necesario abandonar y reiniciar si continua trabajando con ACTS. Sin embargo, se puede restaurar los canales almecenados previamente.

Seleccione Quit desde File o presione la tecla ESC para salir de ACTS. Ahora reinicie ACTS con esa configuración guardada, actsconfig y la imagen respectiva test.ppm para entrenamiento. La figura 5.41 muestra el arranque con un archivo de configuración.

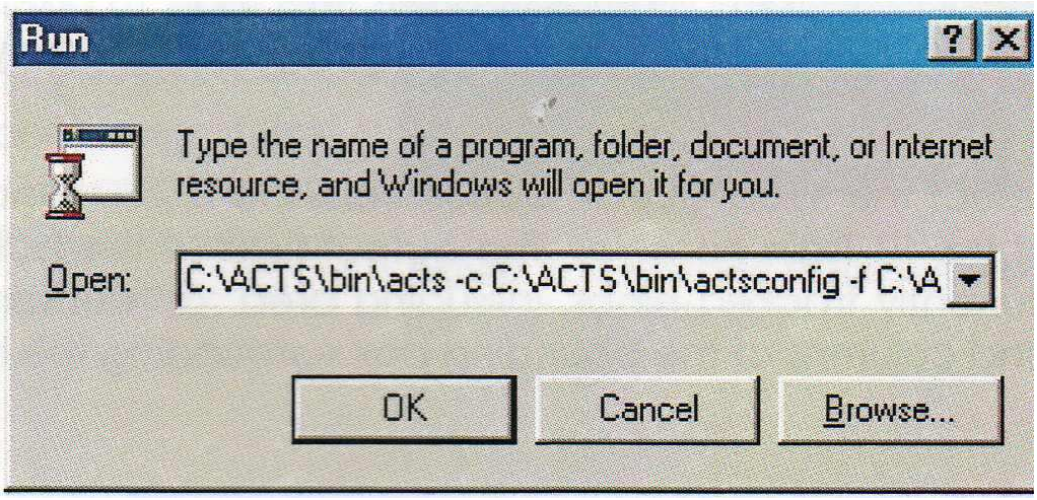

Figura. 5.41**. Puesta en marcha de ACTS con el archivo de configuración**

La imagen del libro y el vaso debería ser visible con la incrustación Thresh azul. Los blobs rastreados deben tener las mismas estadísticas también.

5.3.13. Herramientas de entrenamiento y Sugerencias

Para entrenar un canal de color y rastrear objetos en una variedad de entornos se necesita un número de factores que pueden influenciar grandemente en la calidad de rastreo de un canal. ACTS contiene muchas herramientas para definir y clarificar los colores que ACTS usará para distinguir los objetos con color y utilizarlos en aplicaciones de seguimiento de objetos por color, etc.

5.3.14. Canales

Los pixeles que componen un video típicamente son valores de 24-bit, tres veces ocho bits por cada componente rojo, verde y azul que componen cada pixel. ACTS clasifica cada pixel mediante la comparación de sus componentes RGB contra una tabla de búsqueda almacenada dentro de cada uno de los 32 canales de color.

Si los componentes del pixel son encontrados ahí, este es clasificado como interesante por el canal y es rastreado. Los pixeles que no son indexados en el canal de color, son ignorados.

Almacenar todas las combinaciones de color posibles para imágenes de 160x120 de 24-bits requerirá cerca de 17 millones de localizaciones distintas de índice. Este es “por canal”, multiplique por 32 para considerar todos los canales que ACTS soporta.

A más de la excesiva memoria, inicializar e iterar a través de esta cantidad de datos podría poner a prueba incluso a los procesadores más rápidos. ACTS utiliza un esquema que involucra proyectar los datos de los pixeles dentro de un conjunto de planos de tres colores. En lugar de tener un cubo RGB de almacenamiento de datos, son usados tres planos de color, un Rojo/Verde (RG), uno Azul/Verde (BG) y uno Rojo/Azul (RB).

Para ser clasificados como interesantes, los pixeles de la imagen deben tener sus componentes rojo y verde ubicados en el plano RG, sus componentes azul y verde ubicados en el plano BG y sus componentes rojo y azul ubicados en el plano RB.

5.3.15. Herramientas de Entrenamiento de ACTS

ACTS posee una gama de herramientas que son útiles en el momento que se desea iniciar el programa, varias de ellas se pueden observar en la figura 5.42. De hecho, la manera más común y rápida de agregar colores a un canal de ACTS es con el ratón sobre la herramienta Add. Esta es la herramienta predeterminada cuando se inicia con EZ-Train. Se puede hacer clic y arrastrar esta herramienta en la ventana de entrenamiento para dibujar una incrustación rectangular en la imagen, el interior de la cual delimita el conjunto de pixeles que se añadirá al canal.

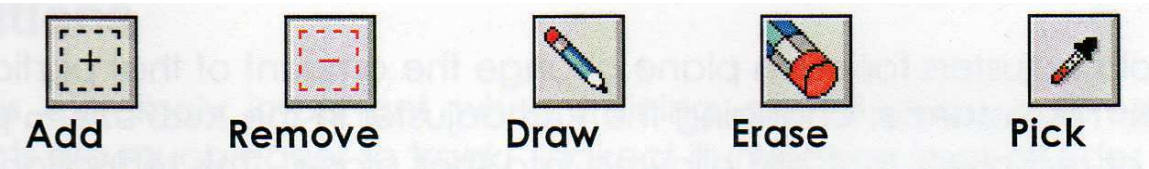

Figura. 5.42**. Herramientas de Instrucción de ACTS**

Sin embargo, simplemente agregando valores de pixel al canal generalmente no es lo suficientemente robusto para rastrear un objeto de color. Además del único valor del pixel,

un conjunto de valores de pixeles adicionales que veamos a ese valor son también agregados al canal de color.

En consecuencia, cada uno de los tres planos de color recibe un cuadrado de color que consiste en múltiples pixeles para ser agregados a la tabla. Esta redundancia ayuda a hacer mucho más robusto el entrenamiento del color asegurando seguro que las regiones agregadas a los canales de color sean más o menos contiguas. Esto ayuda a compensar las condiciones variables de iluminación y otros ruidos ambientales.

La herramienta Remove borra valores de pixeles desde el canal. Se usa como la herramienta Add, excepto cuando los contenidos de color seleccionado son removidos del canal. Los pixeles removidos del canal no son solo los pixeles seleccionados, sino también valores de pixeles inmediatos.

Así, tratar de remover pocos pixeles periféricos del canal (como aquellos pixeles no deseados que fueron seleccionados en el ejemplo) puede remover pixeles de los colores aparentemente no relacionados.

La herramienta Draw le permite apuntar y dar clic para seleccionar las regiones de la Ventana de Entrenamiento y añadir al canal. Por default, Draw añade un pixel individual en la Ventana de Entrenamiento al canal cada vez que da clic con el ratón. Se cambia el tamaño de esta herramienta con los ajustes de Brush Size en los Settings de la EZ-Train desde un pixel individual hasta un cuadrado de 81-pixeles.

La herramienta Erase, como la herramienta Draw, selecciona sectores discretos en la imagen de la Ventana de Entrenamiento, pero remueve los colores del canal. Su tamaño también es establecido por el ítem Brush Size en el menú Settings de la EZ-Train.

Finalmente, la herramienta Pick sirve para dar clic en un pixel individual en la Ventana de Entrenamiento y tener sus valores RGB impresos en la ventana de texto Output en la parte inferior de la consola EZ-Train.

Pick es una herramienta de diagnóstico importante para ver como un valor particular de un pixel puede cambiar con los entornos variables, como por efecto de una sombra en movimiento.

5.3.16. El Modo de Entrenamiento Manual

El panel de control del modo de entrenamiento manual permite ajustar finamente varias configuraciones para el canal seleccionado. Cada plano de color que comprende el canal de color de ACTS tiene 4 ajustadores asociados y 3 botones titulados Reset Limits, Fill Plane y Clear Plane.

Se puede modificar varios límites para ajustar una selección de valores de valores de los planos incluidos. Estos parámetros, dos niveles de color y dos niveles de brillo, definen qué pixeles de color son permitidos en el canal como se ve en la figura 5.43.

Inicialmente, no se configuran límites, esto significa que todos los parámetros son establecidos a su valor máximo o mínimo y todos los colores de pixeles están permitidos.

Los dos ajustadores de umbral de color para cada plano cambian la cantidad de ese color en particular que está permitido en el plano. Por ejemplo, cambiar el ajustador Red en el plano Red/Green aumentará o disminuirá la cantidad de pixeles rojos que está permitido dentro de ese plano.

Esta reducción se muestra por los pixeles blancos del plano que están siendo reemplazados por los pixeles negros.

Cualquier color que pase a estar en este plano cuando estos parámetros estén cambiados son transformados al negro y ya no son rastreados. Para restaurarlos, se cambia el ajustador apropiado para revelar el color de nuevo o se presiona el botón Reset Limits.

Los ajustadores de límite de color removerán los pixeles empezando en la dirección de las agujas del reloj y dirección contra horaria.

El ajustador High Bright cambia según la cantidad de pixeles valorados como “más claros” (cerca al blanco) sean clasificados como interesantes. Esto limita y remueve pixeles desde la parte superior derecha.

El ajustador Low Brighter cambia según la cantidad de pixeles valorados como "obscuros" pixeles valorados (cerca al negro) sean clasificados como interesantes. Esto remueve y limita pixeles desde la parte inferior izquierda del plano.

El botón Fill Plane activa cada pixel en el plano de color correspondiente. Si cada plano se llena en esta forma, ACTS ha clasificado cada pixel como de interés y la ventana de EZ-Train debería ser completamente iluminada.

ACTS se puede entrenar rellenando cada plano de color y luego ajustando los límites en los planos para quitar lo que no se necesita. Este es un enfoque diferente del mecanismo aditivo usado en la ventana de EZ-Train pero no debería ser más útil en algunas circunstancias.

El botón Clear Plane borra cada pixel en el plano de color correspondiente, revirtiéndolo al blanco.

5.3.17. Condiciones de Iluminación

La calidad de la iluminación es extremadamente importante cuando se entrene y rastree objetos de color. Los objetos correctamente iluminados son mucho más fáciles de rastrear, la iluminación indirecta a todos los lados es probablemente la mejor, debido a que las fuentes de iluminación puntuales (bombillas) producirán sombras y se reflejan en las superficies brillantes de los objetos que están siendo rastreados.

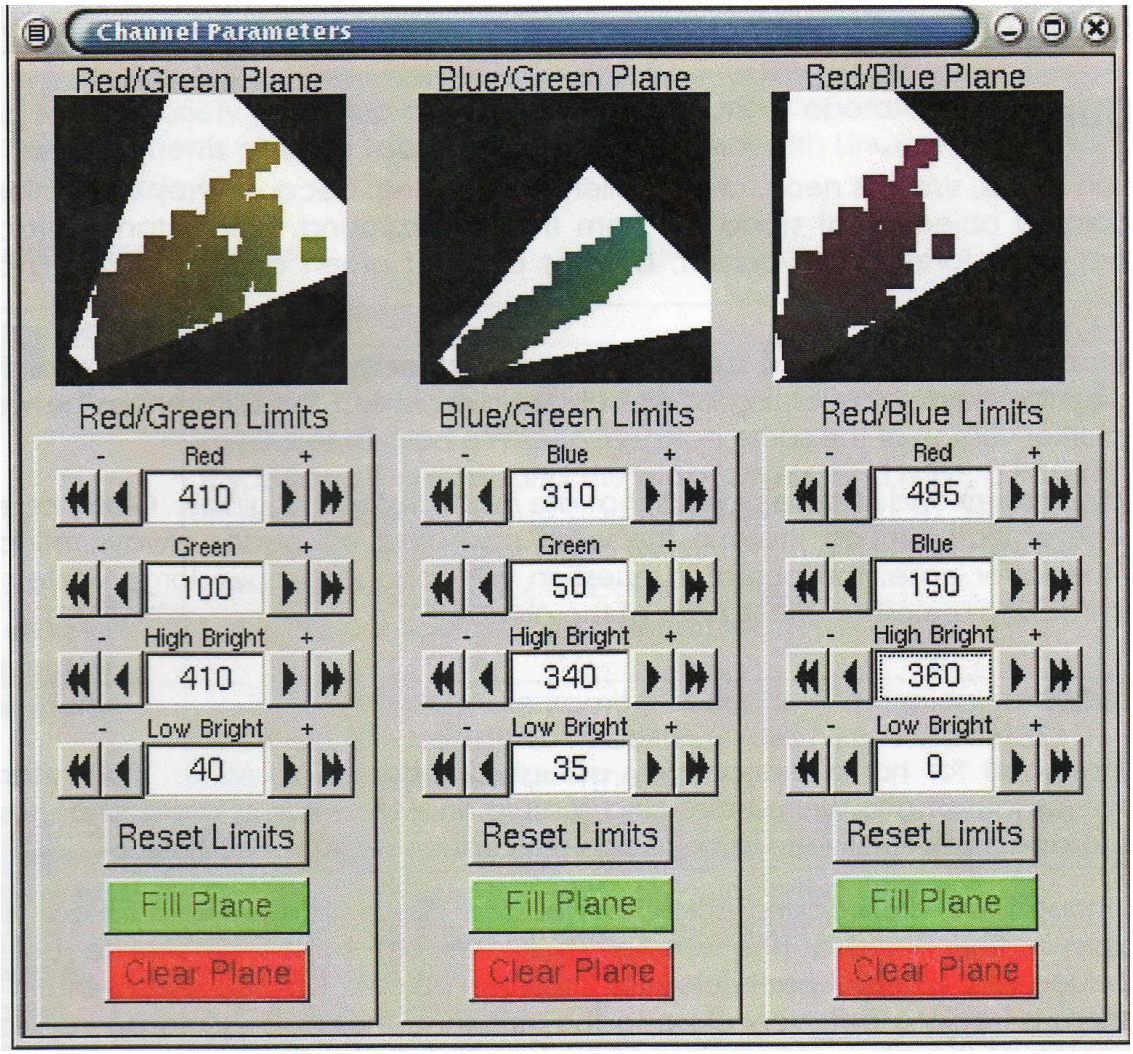

Figura. 5.43. **Canal de Parámetros de Colores de ACTS**

Se debe evitar tener el objetivo " retroiluminado", puesto que esto puede causar sombras y probablemente "arruinar" los colores. Las luces fluorescentes son mejores que las incandescentes porque tienen un espectro de color blanco. Cuando entrene un canal para rastrear un objeto es muy importante ver el objeto bajo varias condiciones. Después de entrenarlo bajo una condición de luz, ver al objeto bajo otra condición de iluminación, se debe entrenar lo quea necesario.

Por ejemplo, después del entrenamiento en un canal, para rastrear un objeto de un color particular, produce una sombra por encima del objeto. Esto revelará, particularmente cuando la Ventana de Entrenamiento está en el modo *Thresh* que varios colores de pixel en el objeto no está siendo rastreados en el canal. Esos colores pueden ser luego añadidos al canal para hacer totalmente más robusto el entrenamiento de color.

5.3.18. Selección de Colores

Los colores de alta saturación, como el neón, son excelentes para el rastreo, porque son distintivos. También se debe escoger objetos de color que se diferencian de su fondo, por ejemplo, el rastreo de un objeto de color naranja brillante sobre una alfombra verde-bosque, pero no un objeto verde sobre un fondo verde oscuro.

Si las condiciones de iluminación cambian o si el objeto se mueve más lejos de la cámara, la cantidad de luz vista por la cámara reflejado en los objetos disminuye. Las diferencias entre un color claro y una versión oscura del mismo color serán difíciles de ver.

Si es posible, es mejor escoger objetos que sean mate en lugar de brillantes. Los objetos brillantes reflejan los colores de los objetos alrededor de ellos, y casi siempre reflejan las fuentes de luz cercanas. En este último caso, el objeto en cuestión aparecerá teniendo manchas blancas, haciendo inútil el seguimiento de color en esas partes del objeto.

5.3.19. Efectos de cámara

Lo que puede ser bueno para los videos caseros puede interferir con ACTS. Por ejemplo, algunas cámaras tienen una característica de balance de blancos automático, en el cual la cámara cambia los valores de color de los pixeles de la imagen relativos a sus vecinos para producir una imagen clara.

ACTS trabaja con colores de rastreo y no con bordes, debido a que las condiciones de luz tienen diferentes colores en sus espectros (el sol vs. las luces fluorescentes), el balance de blancos producirá colores de pixel muy diferentes para el mismo objeto.

Esto también significa que el objeto que la cámara capta está muy cerca a los lentes (y en consecuencia aparece muy grande en la imagen), los colores de otros objetos en la imagen empezarán a variar conforme la cámara intenta "balancear" los valores de color.

5.3.20. Opciones de Arranque (Ejecución)

ACTS viene con una variedad de opciones de arranque para hacer frente a diversos entornos operativos. Los argumentos de líneas de comandos dados en la tabla 5.61 son las opciones disponibles.

Opción	Argumento	Significado
(none)	(none)	Predeterminado; modo de cliente EZ-Train para carga de inicio sin canales o configuración definida. La ventana de imagen de capacitación contendrá video de placa de captura en vivo. Si está disponible, o imagen de tutorial
-c	**Config filename**	Cargue el archivo de configuración <nombre del archivo>
-d	**Devname**	Selecciona el dispositivo apropiado para placa de captura. Ajustes característicos, Windows: 0-9, o -1 para seleccionar la primera placa de captura disponible
-f	**Imagename**	Carga el archivo de imagen .ppm en vez de uasr la placa de captura
-G	**Driver type**	Especifica el tipo de driver (use –h para ver los tipos soportados)
-h	none	Imprime la lista de ayuda (así como también las listas de los tipos de drivers soportados)
-H	**Height**	Altura de la imagen en filas
-i	**Invert**	Imagen invertida verticalmente
-n	**Number**	Especifica el número de canal
-p	**Portnum**	Puerto TCP para comunicar
-P	**Parameter**	Muestra los parámetros de la cámara disponibles usados in la opción –s
-R	**Frames/sec**	Especifica la velocidad de cptura de video
-s	**Set**	Establece un <parámetro> con <valor> para la placa de captura. Ejemplo: -s brillo 0.3 –s contraste 0.1 usa la opción –P para ver qué opciones están disponibles para el tipo de cámara actualmente seleccionada
-S	**Disable SAV**	Desactiva el servidor del Software A/V
-t	**Config file**	Carga el archivo de configuración <archivo> y corre sin gráficos (solo Linux)
-V	**PAL | NTSC**	Especifica el formato PAL o NTSC. NTSC está predeterminado
-w	**Píxel width**	El ancho mínimo que un blob debería tener antes de que sea rastreado.
-W	**Image width**	Ancho de la imagen en columnas
-x	None	Usa la placa de captura compatible de PXC200 con el driver bt848. AVERTENCIA! Solo usa si un PXC está instalado; de otro modo, ACTS podría hacer que el computador falle inesperadamente

Tabla. 5.61. **Opciones de Carga de Inicio y Tiempo de Activación**

5.3.21. La Ventana de Entrenamiento

La ventana de entrenamiento contiene el video en vivo del capturador de tramas o un archivo de imagen estática, así como también superposiciones generadas por ACTS y visualizaciones de los canales seleccionados de color y blobs extraídos.

Se usa para ver el trabajo de ACTS y para entrenar sus tablas de búsqueda de color desde la consola EZ-Train y el modo de entrenamiento manual.

Cuando el ratón está en la ventana de entrenamiento el botón del medio actúa como el botón Grab Frame en la consola EZ-Train: cuando sujete el video en vivo, éste cambia la vista desde un fotograma instantáneo para tomar y volver de nuevo.

Dar un clic en el botón derecho del ratón en la ventana de entrenamiento o presionar la tecla inteligente para cambiar en forma cíclica entre los modos: *Overlay, Thresh, Raw,* y *Visible.* Si el modo *Active* no aparece en la imagen, aparecerá también en la barra de titulo de Windows.

5.3.21.1. Modos de las Ventanas de Entrenamiento

- Modo Overlay: la ventana de entrenamiento muestra los datos del pixel y aparece en el video o en el archivo de la imagen cargada. Todos los pixeles de ACTS encontrados de interés son de color azul, verde o rojo dependiendo de la selección de su *Thresh Color* de las opciones del menú de herramientas de la EZ-Train. Tecla inteligente de atajo de overlay: 1

- Modo Thresh: despliega todos los pixeles de interés, blancos y todos los que no son de interés como negros. Los blob agrupados aparecen como rectángulos púrpuras alrededor de las regiones de interés. El centro de la masa para cada blob es mostrado por un único pixel púrpura dentro de cada rectángulo. Números de blobs rastreados en la consola EZ-Train limitan el número de rectángulos desplegados. Tecla inteligente de atajo de Thresh: 2

- Modo Raw: si el modo overlay no funciona para el muestreo de pixeles de colores especiales. Tecla inteligente de atajo de Raw: 3

- Modo Visible: es como el modo Raw, pero despliega todos los pixeles de interés como sus colores naturales de blanco. No hay rectángulos desplegados para los blobs. Tecla inteligente de atajo de visible: 4

5.3.22. La Consola EZ-Train

La consola EZ-Train está donde se disponga el proceso de entrenamiento del color. La sección de las herramientas de entrenamiento de la consola le permiten escoger una función de entrenamiento para operar el ratón, incluyendo, adicionando y removiendo las regiones individuales o huecas de los pixeles desde el canal actualmente activo y los valores de ejemplo RGB.

Para activar una herramienta en particular se da clic en el ícono de la consola o se presiona la tecla de atajo mientras el cursor está en la ventana de entrenamiento. El cursor del ratón cambia al ícono de las herramientas cuando está sobre la ventana de entrenamiento.

5.3.23. Configuración del Canal

Muchas herramientas y cuadros de diálogo de la consola permiten setear número de parámetros de cada una de los más de treinta y dos canales de color rastreados.

5.3.23.1. Canales Actuales Seleccionados

Esta sección de botones circulares de la consola es quizá lo más importante de otras características de la consola en lo que se refiere a la alteración de los datos del canal. Los botones 1-8, 9-16, 17-24 o 25-32 seleccionan un banco diferente de ocho canales.

Cuando cambia de canal, ACTS automáticamente va a los parámetros del canal seleccionado incluyendo el número de blobs rastreados y añadiendo información de color en el canal.

Cuando seleccione un canal diferente, la imagen de raw en la venta de entrenamiento no cambiará porque los parámetros de color rastreados probablemente han cambiado, los blobs indicados en la ventana de overlay y modos thresh podrían cambiar también.

5.3.23.2. Números de Blobs Rastreados

La herramienta de blobs rastreados sirve para configurar el número de objetos que desee rastrear en el canal actual. Se puede incrementar o disminuir el valor a un máximo de diez.

ACTS reportará a la información del cliente acerca del máximo de ese número de los blobs más largos que detecte del cuadro de la imagen actual. El número de blobs predeterminado es tres.

5.3.23.3. Nombre de Archivo del Canal

El nombre dentro del cuadro de texto Channel Filename es el nombre que ACTS usa para la configuración del mismo, se puede cambiar el nombre y guardarlo para un uso futuro.

Alternativamente ingresando un nombre se carga la información de canal guardada dentro del canal actual, si el archivo existe. Dar un clic en el botón ellipsis ("…") para activar un cuadro de diálogo del archivo de selección, los contenidos son cargados dentro del canal actual.

5.3.24. Comandos de la Ventana de Entrenamiento

La configuración de controles en los comandos de la ventana de entrenamiento de la consola administra varios parámetros de la ventana de entrenamiento.

5.3.24.1. Modo Continuous Frame Grab

En este modo ACTS trabaja para grabar un cuadro, procesarlo y desplegar la imagen desde la ventana de entrenamiento. El ACTS más rápido puede grabar treinta cuadros por segundo. Cuando selecciona el botón circular Continuous, el botón Grab Frame se vuelve gris.

5.3.24.2. Modo Single Frame Grab

El botón circular de Single Frame sirve para parar continuamente la grabación de cuadros y desplegar la última imagen grabada o para tomar una nueva foto fija.

5.3.24.3. Modo Image View

Si está cargada una imagen estática de un archivo dentro de la ventana de entrenamiento, el modo frame-grabbing del botón Live Video, comandará para desplegar imágenes desde la placa de captura de nuevo.

5.3.24.4. Modo Training Window Closed

Si se cierra la ventana de entrenamiento con el botón Close Window en la esquina superior derecha de la ventana, se puede probar el tiempo óptimo de rastreo del color mientras está en entrenamiento. En la ausencia de una imagen el botón de Training Window Commands se convierte en Show EZ-Train.

5.3.24.5. Undo

Sirve para cancelar la última operación de entrenamiento sobre un canal. Cada canal tiene su propio espacio de memoria Undo, entonces puede regresar y revertir un cambio en un canal incluso si se ha modificado muchas configuraciones en otro canal.

5.3.24.6. Magnification

Se puede reducir o agrandar el tamaño de la ventana de entrenamiento para reducir el tamaño o en la fila derecha para aumentar el tamaño en la herramienta Magnification del comando de ventana de entrenamiento. La mínima magnificación es 1x y la máxima 6x.

5.3.24.7. Borrar Entrenamiento

Remueve todos los valores de color de un canal haciendo uso del botón Clear Training. Entonces un cuadro de diálogo aparecerá haciendo borrar completamente el canal.

5.3.24.8. Load and Save Channel y Save As.

Cuando seleccione los ítems *Load and Save Channel* y *Save As* del menú File active un cuadro de dialogo de la selección de archivo común o guarde respectivamente los datos del canal en el disco.

5.3.24.9. Configuración de Tiempo de Carga

Use el cuadro de diálogo File Selection activado por este ítem para cargar una configuración de tiempo de carga de canales dentro de ACTS.

5.3.24.10. Load Image

Load image permite seleccionar un archivo .ppm para cargar dentro de la ventana de entrenamiento. Esto es muy útil para computadoras que no tengan tarjeta de video o si se desea demostrar ACTS sin mayores problemas.

ACTS puede cargar archivos .ppm de cualquier tamaño tal como son guardados en color 24-bit y serán cambiados el tamaño para adecuar el tamaño de la ventana.

5.3.24.11. Quit

Esta es la mejor manera de salir de ACTS. Se recomienda guardar los canales y todo lo que se necesite antes de salir de ACTS.

5.3.24.12. Menú Settings

El menú settings en el menú principal de la consola consiste de una serie de menús jerárquicos e ítems que controlan varios parámetros de cómo ACTS opera en la ventana de entrenamiento.

5.3.24.13. Modo View

Los ítems del modo View seleccionan la apariencia de los datos del canal en la ventana de entrenamiento incluyendo Overlay, Thresh, Raw y Visible.

5.3.24.14. Brush Size

Brush Size le permite seleccionar desde un pixel a un cuadrado de pixeles de 9x9 para las herramientas Draw y Erase.

5.3.24.15. Draw Color

Por predeterminación todas las herramientas de entrenamiento usan un rectángulo rojo en la ventana de entrenamiento para delimitar sus áreas de influencia. Esto puede ser molesto cuando se trata de entrenar sobre objetos rojos. Con este menú de herramientas se puede cambiar el Draw Color a azul o verde o volver al rojo para mejor visibilidad.

5.3.24.16. Thresh Color

En modo Overlay los pixeles de interés son disimulados por una cubierta azul. Use el thresh color para tener mejor visibilidad dependiendo del fondo de la imagen.

5.3.24.17. Menú Info

Los ítems encontrados en el menú Info despliegan sus resultados fuera de la parte inferior de la ventana de la consola.

5.3.24.18. Timing

Cuando selecciona Timing, éste despliega cuantos milisegundos toma ACTS para procesar un cuadro de video. Esto también muestra el número de cuadros por segundo que han sido obtenidos.

CAPÍTULO 6

DESARROLLO DE APLICACIONES

Una vez conocido el funcionamiento y la programación es necesario realizar algunos ejemplos de aplicaciones que se pueden aplicar en el laboratorio y también se los puede usar en la vida cotidiana.

Dentro de las aplicaciones consideradas base, se desarrollaron cuatro tipos:

- Seguimiento de Trayectorias
- Control de Giros
- Evasión de Obstáculos
- Control de Visión

Todas estas aplicaciones involucran todos los sensores y accesorios disponibles en el robot, con lo cual ya se puede tener una idea básica del procedimiento de programación a seguir para el desarrollo de aplicaciones.

A continuación se detallarán los ejemplos desarrollados en el laboratorio:

6.1. Seguimiento de Trayectorias

Una de las aplicaciones más importantes dentro de lo que se refiere al manejo de plataformas robóticas es la de seguir trayectorias asignadas, mediante el uso de mapas, o mediante líneas u objetos.

Para este caso se lo realizará con mapas, por lo que se considera para el ejemplo que es la mejor manera de demostrar cómo se puede seguir una trayectoria asignada, usando todos

los sensores que dispone y el comportamiento que presenta la plataforma al seguir dicha trayectoria de manera autónoma.

La siguiente aplicación desarrollada utilizará las librerías del cliente ARIA y de SonARNL, las cuales proporcionan la versatilidad de adaptarse con los diferentes accesorios que poseen las plataformas robóticas para poder cumplir con el objetivo en el momento de seguir la trayectoria asignada.

Para el seguimiento de trayectorias la aplicación se basará en el seguimiento de mapas utilizando una de las interfaces gráficas y el cliente SonARNL que proporciona el fabricante, aplicación parecida al gráfico 6.1.

Figura. 6.1. **Seguimiento de Trayectorias**

Para comenzar se elabora un mapa con la ayuda del programa Mapper3Basic, de la parte inferior de los laboratorios del departamento de electrónica, en el cual se detallarán todas las medidas reales de las instalaciones, así como la ubicación de las zonas prohibidas, las cuales pueden estar conformadas por lugares como por ejemplo acceso a gradas, espacios

huecos, o cualquier otro lugar que ponga en peligro la estructura del robot, si éste transitase por estos lugares.

Otro aspecto importante a considerar son los puntos de partida y de llegada, según los cuales se determinará la trayectoria a seguir, también se puede incluir puntos de recarga del robot, pero para ello hace falta la estación de recarga automática.

Luego de haber realizado el mapa se procederá a cargarlo en el robot, para ello se ingresara en la ventana de comandos de Windows, en la cual se debe ingresar la dirección del mapa a cargar, por ejemplo, para el mapa del laboratorio cuyo nombre es laboratorios.map, se debe escribir en la ventana de comandos:

Cd C:\Program Files\MobileRobots\Arnl\bin guiServerSonar –map laboratorios.map

Con esta operación se accederá al servidor del robot mediante el cliente SonARNL y se cargará el mapa en el microcontrolador.

Para poder seguir las trayectorias así como iniciar el funcionamiento del robot dentro del mapa dibujado se utiliza la interfaz gráfica MobileEyes, en la cual se puede visualizar el comportamiento del robot, y también se lo puede dirigir de modo que se mueva de manera manual o autónoma.

Para esta aplicación se han tomado como punto de partida la puerta del laboratorio del CIM y el punto de llegada el laboratorio de control industrial en la parte inferior, como se muestra en la figura 6.2. A continuación se muestra el código fuente utilizado por el cliente, el cual permite la navegación del robot por el mapa creado.

Mobileyes envía a este programa muchos requerimientos de manera continua para que el robot pueda llegar a la meta establecida, así como también obtener el mapa, los parámetros de configuración, así como los estados de los sensores y del robot.

Dentro de este programa se involucran casi en su totalidad los aspectos de funcionamiento del robot, así como el enlace son sus sensores, interfaces de hardware y software y sus parámetros de configuración.

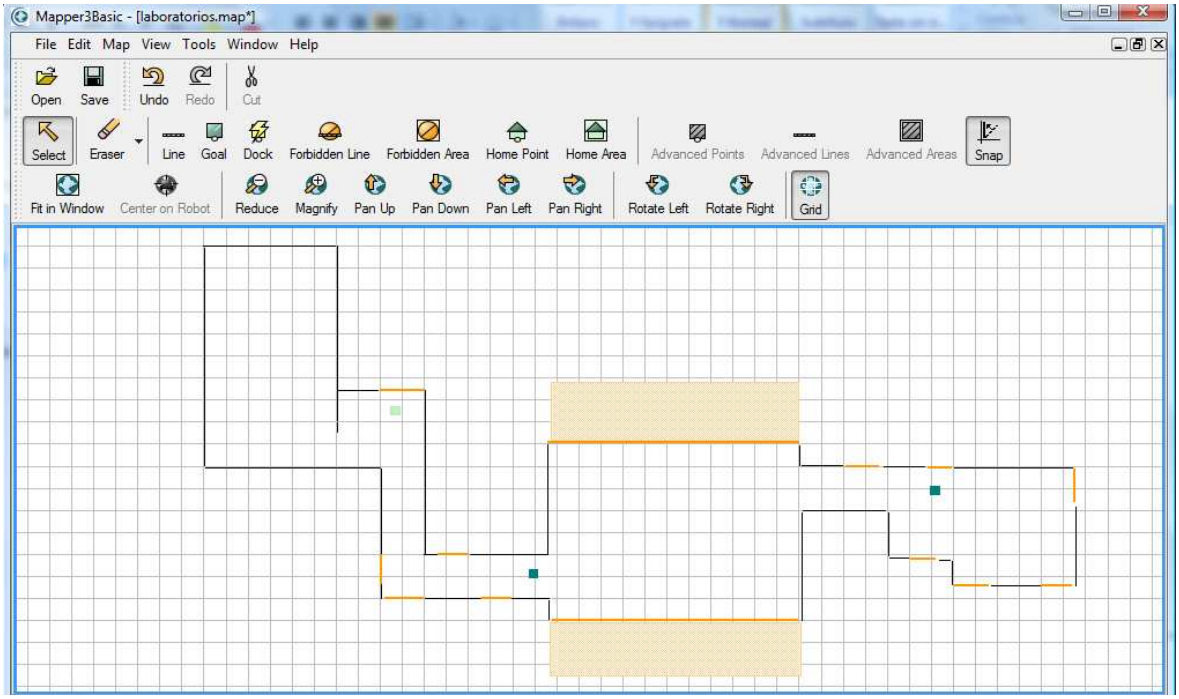

Figura. 6.2. **Mapa de la planta baja de los laboratorios de Electrónica ESPE**

```
guiServer.cpp
#include "Aria.h"
#include "ArNetworking.h"
#include "Arnl.h"

#include "ArSonarLocalizationTask.h"

void logOptions(const char *progname)
{
  ArLog::log(ArLog::Normal, "Usage: %s [options]\n", progname);
  ArLog::log(ArLog::Normal, "[options] are any program options listed
below, or any ARNL configuration");
  ArLog::log(ArLog::Normal, "parameters as -name <value>, see
params/arnl.p for list.");
  ArLog::log(ArLog::Normal, "For example, -map <map file>.");
  Aria::logOptions();
}

bool gyroErrored = false;
```

```
const char* getGyroStatusString(ArRobot* robot)
{
  if(!robot || !robot->getOrigRobotConfig() || robot-
>getOrigRobotConfig()->getGyroType() < 2) return "N/A";
  if(robot->getFaultFlags() & ArUtil::BIT4)
  {
    gyroErrored = true;
    return "ERROR/OFF";
  }
  if(gyroErrored)
  {
    return "OK but error before";
  }
  return "OK";
}

int main(int argc, char **argv)
{
  // Se inicializa la información general de Aria y Arnl
  Aria::init();
  Arnl::init();


  // Objeto robot
  ArRobot robot;

  // Análisis del los argumentos en la línea de comandos.
  ArArgumentParser parser(&argc, argv);

  // Establece simpleConnector, para conexión con el robot y el láser
  //ArSimpleConnector simpleConnector(&parser);
  ArRobotConnector robotConnector(&parser, &robot);

  // Conexión con el robot
  if (!robotConnector.connectRobot())
  {
    ArLog::log(ArLog::Normal, "Error: Could not connect to robot...
exiting");
    Aria::exit(3);
  }

  // Establece donde se buscarán los archivos. Arnl::init() establece a
```

```
  // Aria por defecto
  // carpeta por defecto para el Arnl; addDirectories() anexa esta
  // carpeta "examples" .
  char fileDir[1024];
  ArUtil::addDirectories(fileDir, sizeof(fileDir), Aria::getDirectory(),
                  "examples");

  // Para dirigir mensajes de acceso a un archivo, o para cambiar el
nivel de acceso se usa estas llamadas:
  //ArLog::init(ArLog::File, ArLog::Normal, "log.txt", true, true);
  //ArLog::init(ArLog::File, ArLog::Verbose);

  // Añade una sección a la configuración para cambiar los parámetros del
ArLog
  ArLog::addToConfig(Aria::getConfig());

  // Establece el giroscopio e incorpora automáticamente las correccions
del giroscopio
  ArAnalogGyro gyro(&robot);

  // Servidor de red
  ArServerBase server;

  // Establece el simpleOpener, usado para configurar el servidor de red
  ArServerSimpleOpener simpleOpener(&parser);

  // Carga los argumentos por defecto para la computadora
  parser.loadDefaultArguments();

  // Análisis de argumentos
  if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
  {
    logOptions(argv[0]);
    Aria::exit(1);
  }

  // Provoca que se llame a Aria::exit(9) si el robot se desconecta
  // inesperadamemte
    ArGlobalFunctor1<int> shutdownFunctor(&Aria::exit, 9);
  robot.addDisconnectOnErrorCB(&shutdownFunctor);

  // Se crea una objeto ArSonarDevice (subclase ArRangeDevice) y
```

```
// se lo conecta al robot.
ArSonarDevice sonarDev;
robot.addRangeDevice(&sonarDev);

// Este objeto permitirá que se cambien los parámetros que permiten el
// movimiento del robot que son cambiados a través de la sección de
// configuración del robot del global ArConfig.
ArRobotConfig robotConfig(&robot);

// Incluye las opciones de configuracipin del giroscopio en la sección
// de configuración del robot.
robotConfig.addAnalogGyro(&gyro);

// Empieza el funcionamiento del robot.
robot.runAsync(true);

  /* Creación y configuración de un objeto mapa*/

// Se establece el objeto mapa, esto hará que se busquen los archivos
// en la carpeta de ejemplos(a menos que el archivo empieze con /, \,o.
// Se puede tomar la car´peta de salida 'fileDir' para buscar en la
//carpeta actual del programa.
// Cuando la configuración del archivo se carga en el ArConfig, si este
//se especifica como un archivo de mapa, luego este archivo sera
//cargado como el mapa en el robot.
ArMap map(fileDir);
// Establece un parámetro para ignorar archivos vacios
map.setIgnoreEmptyFileName(true);
// Se ignora el caro en el que si alguien usa MobileEyes
// MobilePlanner desde Windows y cambia algún nombre en el mapa,
// este deberá seguir funcionando.
map.setIgnoreCase(true);

  /* Creación de patrones de localización y planificación de rutas */

ArPathPlanningTask pathTask(&robot, &sonarDev, &map);

ArLog::log(ArLog::Normal, "Creating sonar localization task");
ArSonarLocalizationTask locTask(&robot, &sonarDev, &map);

  /* Inicio del servidor */
```

```
// Abrir el servidor
if (!simpleOpener.open(&server, fileDir, 240))
{
  ArLog::log(ArLog::Normal, "Error: Could not open server.");
  exit(2);
}

  /* Creación de varios servicios que proveen acceso de la red al
   * cliente (como MobileEyes), mientras se añaden varias
   * característicasas adicionales al ARNL */

/* Se puede añadir dispositivos de rango adicionales para el robot y la
 * planificación de rutas (de tal manera que evita obstáculos ¨
 * detectados por estos dispositivos) */

// Añade infrarrojos al robot y la tarea de planificación de rutas
robot.lock();
ArIRs irs;
robot.addRangeDevice(&irs);
pathTask.addRangeDevice(&irs, ArPathPlanningTask::CURRENT);

// Añade los parachoques
ArBumpers bumpers;
robot.addRangeDevice(&bumpers);
pathTask.addRangeDevice(&bumpers, ArPathPlanningTask::CURRENT);

// Añade dispositivos de rango los cuales se usan para detectar las
// zonas no autorizadas en el mapa y manda las lecturas al ARNL
ArForbiddenRangeDevice forbidden(&map);
robot.addRangeDevice(&forbidden);
pathTask.addRangeDevice(&forbidden, ArPathPlanningTask::CURRENT);

robot.unlock();

// Acción para parar el robot cuando la localización se corta pero no
// si el robot se encuentra “perdido”.
ArActionSlowDownWhenNotCertain actionSlowDown(&locTask);
pathTask.getPathPlanActionGroup()->addAction(&actionSlowDown, 140);

// Acción que para el robot cuando este se encuentra “perdido”
ArActionLost actionLostPath(&locTask, &pathTask);
pathTask.getPathPlanActionGroup()->addAction(&actionLostPath, 150);
```

```
  // Servicios para observar los datos en del robot en el mapa:
  ArServerInfoDrawings drawings(&server);
  drawings.addRobotsRangeDevices(&robot);

  /* Para dibujar una caja alrededor del area local de planificación de
ruta se puede usar lo siguiente */
  ArDrawingData drawingDataP("polyLine", ArColor(200,200,200), 1, 75);
  ArFunctor2C<ArPathPlanningTask, ArServerClient *, ArNetPacket *>
    drawingFunctorP(&pathTask, &ArPathPlanningTask::drawSearchRectangle);
  drawings.addDrawing(&drawingDataP, "Local Plan Area",
&drawingFunctorP);

  // comandos "Habituales" . Se puede añadir los comandos habituales que
se desee, los cuales estarán disponibles en MobileEyes' en la pestaña de
custom commands en la barra de herramientas
  ArServerHandlerCommands commands(&server);

  // Se provee varios tipos de información al cliente:
  ArServerInfoRobot serverInfoRobot(&server, &robot);
  ArServerInfoSensor serverInfoSensor(&server, &robot);
  ArServerInfoPath serverInfoPath(&server, &robot, &pathTask);
  serverInfoPath.addSearchRectangleDrawing(&drawings);
  serverInfoPath.addControlCommands(&commands);

  // Provee información de localización y permite al cliente (MobileEyes)
  // dar la relocalización de una posición dada:
  ArServerInfoLocalization serverInfoLocalization(&server, &robot,
&locTask);
  ArServerHandlerLocalization serverLocHandler(&server, &robot,
&locTask);

  // Si se está usando MobileSim, el ArServerHandlerLocalization le envía
  // un comando para mover el robot a una posición real tal como se lo
  // hace manualmente a través de MobileEyes. Para desabilitar esa opción
  // se usa el constructor: ArServerHandlerLocalization
  // serverLocHandler(&server, &robot, true, false);
  // El quinto argumento determina si es que se envia el comando al
  // MobileSim.

  // Provee el mapa al cleinte (Y controles relacionados):
  ArServerHandlerMap serverMap(&server, &map);
```

```
// Estos objetos añaden algunos comandos simples (por defecto) hacia
// los 'commands' para probarlos y desbugearlos:
ArServerSimpleComUC uCCommands(&commands, &robot);
// Envia cualquier comando hacia el microcontrolador
ArServerSimpleComMovementLogging loggingCommands(&commands, &robot);
// Configuración de inicialización
ArServerSimpleComLogRobotConfig configCommands(&commands, &robot);
// Empieza la inicialización de los parámetros de configuración del
// robot ArServerSimpleServerCommands serverCommands(&commands,
// &server); monitoreo del comportamiento de la red

// servicios que permite al cliente monitorear el estatus de la
// comunicación entre el robot y el cliente
ArServerHandlerCommMonitor handlerCommMonitor(&server);

// Servicio que permite al cliente cambiar los parámetros de
// configuración en el ArConfig
ArServerHandlerConfig handlerConfig(&server, Aria::getConfig(),
                           Arnl::getTypicalDefaultParamFileName(),
                           Aria::getDirectory());

/* Establece los posibles modes para control remoto desde el cliente
 * como MobileEyes: */
// Modo para ir a una meta o un punto específico:
ArServerModeGoto modeGoto(&server, &robot, &pathTask, &map,
                     locTask.getRobotHome(),
                     locTask.getRobotHomeCallback());


// Modo para Parar y mantener la parada:
ArServerModeStop modeStop(&server, &robot);


// Modos de Teleoperación para menajar por teclado, joystick, etc:
ArServerModeRatioDrive modeRatioDrive(&server, &robot);
//  ArServerModeDrive modeDrive(&server, &robot);
// Modo antiguo para compatibilidad
// Previene el manejo por teleoperación si la localización se pierde
// usando una acción de prioridad más altas, la cual habilita cuando
```

```
  // algún modo en particular se encuentra activo.
  // (Se debe entrar en modo no seguro para manejar cuando se pierde la
  // localización.)
  ArActionLost actionLostRatioDrive(&locTask, &pathTask,
&modeRatioDrive);
  modeRatioDrive.getActionGroup()->addAction(&actionLostRatioDrive, 110);

  // Añade la sección de modo de manejo a la configuración y también
  // algunos comandos simples:
  modeRatioDrive.addToConfig(Aria::getConfig(), "Teleop settings");
  modeRatioDrive.addControlCommands(&commands);

  // Modo Wander
  ArServerModeWander modeWander(&server, &robot);
  ArActionLost actionLostWander(&locTask, &pathTask, &modeWander);
  modeWander.getActionGroup()->addAction(&actionLostWander, 110);

  // Se provee una pequeña tabla de información importante para el
  // cliente que despliega al operador. Se puede añadir llamadas propias
  // para mortrar cualquier dato que se quiera.
  ArServerInfoStrings stringInfo(&server);
  Aria::getInfoGroup()-
>addAddStringCallback(stringInfo.getAddStringFunctor());

  // Provee un conjunto de datos informativos (se debe encender en
  // MobileEyes con View->Custom Details)

  Aria::getInfoGroup()->addStringInt(
        "Motor Packet Count", 10,
        new ArConstRetFunctorC<int, ArRobot>(&robot,
                                    &ArRobot::getMotorPacCount));

  Aria::getInfoGroup()->addStringDouble(
        "Sonar Localization Score", 8,
        new ArRetFunctorC<double, ArSonarLocalizationTask>(
              &locTask,
      &ArSonarLocalizationTask::getLocalizationScore),
        "%.03f");
  Aria::getInfoGroup()->addStringInt(
        "Sonar Loc Num Samples", 8,
        new ArRetFunctorC<int, ArSonarLocalizationTask>(
```

```
            &locTask, &ArSonarLocalizationTask::getCurrentNumSamples),
        "%4d");

  // Muestra el estado del giroscopio se esta habilitado y esta siendo
  // manejado por el firmware (en los tipo de gyro 2, 3, o 4).
  // (Si el firmware detecta un error de comunicación con en giroscopio o
  // IMU, retorna una bandera y para de usarlo).
  // (Este tipo de parámetro del giroscopio, y la bandera de falla, están
  // solo en ARCOS)
  if(robot.getOrigRobotConfig() && robot.getOrigRobotConfig()-
>getGyroType() > 1)
  {
    Aria::getInfoGroup()->addStringString(
          "Gyro/IMU Status", 10,
          new ArGlobalRetFunctor1<const char*,
ArRobot*>(&getGyroStatusString, &robot)
      );
  }

  // Realiza un modo de Parada
  modeStop.addAsDefaultMode();

  /*
  // Si se usa el simulador, se mueve el robot de regreso a la posición
  // de inicio, y resetea todo el entorno.
  // Esto permitirá al localizeRobotAtHomeBlocking() una probabilida baja
  // de trabajar (este trata de reestablacer los puntos actuales del
  // entorno (los cuales deben estar en 0,0,0) en todo el mapa en los
  // puntos de inicio. (ignorados por el robot real).
  //robot.com(ArCommands::SIM_RESET);
  */

  // crea una clase de almacenamiento de posición esto permitirá al
  // programa mantener la ruta entre la cual el robto se despliega...
  // después desde este archivo se puede guardar la posición del robot en
  // un archivo

  ArPoseStorage poseStorage(&robot);
  // Si se podría reesttablecer la posición desde el archivo, luego se
  // establecerá en el simulador (esto no hará nada en el robto real)...
  // si no se puede reestablecer la posición entonce solo se resetea la
  // posición del robot (una vez más no se realizará nada en un robot
```

```
  // real)
  if (poseStorage.restorePose("robotPose"))
    serverLocHandler.setSimPose(robot.getPose());
  else
    robot.com(ArCommands::SIM_RESET);

  /* Servicios de transferencia de archivos: */

#ifdef WIN32
  // Aun no se encuentra implementado para windows.
  ArLog::log(ArLog::Normal, "Note, file upload/download services are not
implemented for Windows; not enabling them.");
#else
  // Este bloque permitirá establecer dond se obtienen y guardan los
  // archivos hacia/desde, solo se debe comentar esta sección si no se
  // quiere que suceda
/*
  ArServerFileLister fileLister(&server, fileDir);
  ArServerFileToClient fileToClient(&server, fileDir);
  ArServerFileFromClient fileFromClient(&server, fileDir, "/tmp");
  ArServerDeleteFileOnServer deleteFileOnServer(&server, fileDir);
*/
#endif

    /* Emisión de Video, y controles de cámara (Requiere SAVserver o
ACTS) */

  // Envía cuanquier video si ACTS o SAV server se encuentran
  // ejecutándose.
    ArHybridForwarderVideo videoForwarder(&server, "localhost", 7070);

  // Pone una cámara para usar en caso de tener video. La colección de
  // cámaras tiene muchas cámaras ptz
  ArPTZ *camera = NULL;
  ArServerHandlerCamera *handlerCamera = NULL;
  ArCameraCollection *cameraCollection = NULL;

  // Si se tiene video se establece una cámara
  if (videoForwarder.isForwardingVideo())
  {

    cameraCollection = new ArCameraCollection();
```

```
    cameraCollection->addCamera("Cam1", "VCC4", "Camera", "VCC4");

    videoForwarder.setCameraName("Cam1");
    videoForwarder.addToCameraCollection(*cameraCollection);

    bool invertedCamera = false;
    camera = new ArVCC4(&robot,    invertedCamera,
                                   ArVCC4::COMM_UNKNOWN, true, true);
    camera->init();

    handlerCamera = new ArServerHandlerCamera("Cam1",
                                                        &server,

&robot,

camera,

cameraCollection);

    pathTask.addGoalFinishedCB(
          new ArFunctorC<ArServerHandlerCamera>(
                handlerCamera,
                &ArServerHandlerCamera::cameraModeLookAtGoalClearGoal));
  }

  // Después de las cámaras / videos debe ser creado y añadido a la
  // colección luego empezar el servidor de colección.
    if (cameraCollection != NULL) {
    new ArServerHandlerCameraCollection(&server, cameraCollection);
  }

    /* Carga los valores de configuración, mapa y se sa inicio */

   Aria::getConfig()->useArgumentParser(&parser);

  // Lectura en los archivos de parámetros.
  ArLog::log(ArLog::Normal, "Loading config file %s into ArConfig...",
Arnl::getTypicalParamFileName());
  if (!Aria::getConfig()->parseFile(Arnl::getTypicalParamFileName()))
  {
```

```
    ArLog::log(ArLog::Normal, "Trouble loading configuration file,
exiting");
    Aria::exit(5);
  }

  // Se previene si se encuentran parámetros desconocidos
  if (!simpleOpener.checkAndLog() || !parser.checkHelpAndWarnUnparsed())
  {
    logOptions(argv[0]);
    Aria::exit(6);
  }

  // Se previene si no existe un mapa cargado
  if (map.getFileName() == NULL || strlen(map.getFileName()) <= 0)
  {
    ArLog::log(ArLog::Normal, "");
    ArLog::log(ArLog::Normal, "### No map file is set up, you can make a
map with the following procedure");
    ArLog::log(ArLog::Normal, "   0) You can find this information in
README.txt or docs/SonarMapping.txt");
    ArLog::log(ArLog::Normal, "   1) Start up Mapper3Basic");
    ArLog::log(ArLog::Normal, "   2) Go to File->New");
    ArLog::log(ArLog::Normal, "   3) Draw a line map of your area (make
sure it is to scale)");
    ArLog::log(ArLog::Normal, "   4) Go to File->Save on Robot");
    ArLog::log(ArLog::Normal, "   5) In MobileEyes, go to Tools->Robot
Config");
    ArLog::log(ArLog::Normal, "   6) Choose the Files section");
    ArLog::log(ArLog::Normal, "   7) Enter the path and name of your new
.map file for the value of the Map parameter.");
    ArLog::log(ArLog::Normal, "   8) Press OK and your new map should
become the map used");
    ArLog::log(ArLog::Normal, "");
  }

  // Se imprime un mensaje al usuario del directorio donde se encuentran
  // los mapas
  ArLog::log(ArLog::Normal, "");
  ArLog::log(ArLog::Normal,
           "Directory for maps and file serving: %s", fileDir);
```

```
  ArLog::log(ArLog::Normal, "See the ARNL README.txt for more
information");
  ArLog::log(ArLog::Normal, "");

  // Se da una localización inicial del robot. Este prueba todos los
  // puntos de inicio en el mapa, y luego establece la posición inicial,
  // tanto  como sea pocible para dar inicio a la operación. Si lo
  // realiza de manera exitosa guarda la posición encontrada como la
  // mejor, y la establece como punto de inicio, la cual puede ser
  // obtenida desde la tarea de localización (esto es usado cuando el
  // cliente realiza la acción “Go to Home”).
  locTask.localizeRobotAtHomeBlocking();

  // Comienza la ejecución del servidor.
  server.runAsync();

  // Se añade una tecla de manejo de tal manera que se pueda salir al
  // presionar esc. Esta tecla tambíen previene el funcionamiento en
  // sitios no adecuados.

  ArKeyHandler *keyHandler;
  if ((keyHandler = Aria::getKeyHandler()) == NULL)
  {
    keyHandler = new ArKeyHandler;
    Aria::setKeyHandler(keyHandler);
    robot.lock();
    robot.attachKeyHandler(keyHandler);
    robot.unlock();
    puts("Server running. To exit, press escape.");
  }

  // Habilita los motores y espera hasta que el robot termine
  // (desconexión, etc.) o hasta que el programa sea finalizado.
  robot.enableMotors();
  robot.waitForRunExit();
  Aria::exit(0);
}
```

Como se puede observar en esta aplicación se detalla paso a paso cada sección y su función dentro del programa, lo cual permite que el robot, pueda seguir la trayectoria deseada,

además brinda la oportunidad de añadir varias opciones extras como poner una cámara de video, usar sensores láser lo cual permitirán realizar otro tipo de aplicaciones más avanzadas como podría ser el caso de tareas de rescate mientras el robot rutea el mapa que está siguiendo.

6.2. Control de Giros

Una de las aplicaciones más importantes para la utilización de las plataformas robóticas es saber controlar los giros de manera adecuada, de tal manera que el robot pueda corregir errores por causa de deslizamientos en las superficies, así como también obedezca instrucciones precisas en el caso de que se desarrolle aplicaciones de localización y navegación.

Para poder cumplir con este objetivo se recurrirá a emplear los diferentes sensores y dispositivos que posee la plataforma robótica, para lo cual se tendrá que recurrir a la herramienta tipo cliente ARIA, la cual permitirá realizar rutinas de programación que demuestren como se utilizan las diferentes órdenes de giro que permitan controlar el robot para posteriores aplicaciones.

Para dar inicio a esta aplicación se muestra el código fuente empleado, el cual tendrá sus comentarios respectivos paso a paso y al final se dará un breve resumen de la versatilidad de dicha aplicación desarrollada.

gyroExample.cpp

```
#include "Aria.h"
#include "ArAnalogGyro.h"
class GyroTask
{
public:
// El constructor, debe utilizar el encadenamiento del
// constructor para inicializar su base
      // class ArSimpleUserTask
      GyroTask(ArRobot *robot);
      // destructor vacio
      ~GyroTask(void) {}
```

```
      // la tarea que se quiere hacer
  void doTask(void);
protected:
      //double myHeading;
      ArAnalogGyro *myGyro;
      ArRobot *myRobot;
      ArFunctorC<GyroTask> myTaskCB;
};

// el constructor, notese como el usa el encadenamiento para
// inicializar myTaskCB
GyroTask::GyroTask(ArRobot *robot) :
  myTaskCB(this, &GyroTask::doTask)

{
  ArKeyHandler *keyHandler;
  myRobot = robot;
  // se añade al robot
  myRobot->addUserTask("GyroTask", 50, &myTaskCB);
  myGyro = new ArAnalogGyro(myRobot);
  if ((keyHandler = Aria::getKeyHandler()) == NULL)
  {
    keyHandler = new ArKeyHandler;
    Aria::setKeyHandler(keyHandler);
    if (myRobot != NULL)
      myRobot->attachKeyHandler(keyHandler);
    else
      ArLog::log(ArLog::Terse, "GyroTask: No hay un robot para adjuntar
un keyHandler, keyHandling no funciona... o bien realice su propio
keyHandler y manejelo usted mismo, hacer un keyhandler y adjuntelo al
robot, o dar éste al robot para que lo adjunte.");
  }
  keyHandler->addKeyHandler('1', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 10));
keyHandler->addKeyHandler('2', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 20));
keyHandler->addKeyHandler('3', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 30));
keyHandler->addKeyHandler('4', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 40));
```

```
keyHandler->addKeyHandler('5', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 50));
keyHandler->addKeyHandler('6', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 60));
keyHandler->addKeyHandler('7', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 70));
keyHandler->addKeyHandler('8', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 80));
keyHandler->addKeyHandler('9', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 90));
keyHandler->addKeyHandler('0', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, 100));

keyHandler->addKeyHandler(ArKeyHandler::SPACE, new
ArFunctorC<ArRobot>(myRobot,&ArRobot::stop));

keyHandler->addKeyHandler('q', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -10));
keyHandler->addKeyHandler('w', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -20));
keyHandler->addKeyHandler('e', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -30));
keyHandler->addKeyHandler('r', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -40));
keyHandler->addKeyHandler('t', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -50));
keyHandler->addKeyHandler('y', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -60));
keyHandler->addKeyHandler('u', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -70));
keyHandler->addKeyHandler('i', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -80));
keyHandler->addKeyHandler('o', new ArFunctor1C<ArRobot,
double>(myRobot,&ArRobot::setRotVel, -90));
keyHandler->addKeyHandler('p', new ArFunctor1C<ArRobot, double>(myRobot,
&ArRobot::setRotVel, -100));
keyHandler->addKeyHandler('a', new ArFunctor1C<ArRobot, double>(myRobot,
&ArRobot::setHeading, 0));
keyHandler->addKeyHandler('s', new ArFunctor1C<ArRobot, double>(myRobot,
&ArRobot::setHeading, 90));
```

```
keyHandler->addKeyHandler('d', new ArFunctor1C<ArRobot, double>(myRobot,
&ArRobot::setHeading, 180));
keyHandler->addKeyHandler('f', new ArFunctor1C<ArRobot, double>(myRobot,
&ArRobot::setHeading, 270));
}

void GyroTask::doTask(void)
{
  /*
  double degrees = -((myRobot->getAnalog() * 5.0 / 255) - 2.509) * 150 /
2.5 * 1.265;
  if (fabs(degrees) < 2)
    degrees = 0;
  myHeading += degrees * .025;
  printf("%10f %10f %10f %10f\n", myRobot->getAnalog() * 5.0 / 255,
degrees,
             myRobot->getRotVel(), myHeading);
  fflush(stdout);
  */
  printf("gyro %10f robot %10f mixed %10f temp %d ave %g\n", myGyro-
>getHeading(),  myRobot->getRawEncoderPose().getTh(),  myRobot->getTh(),
myGyro->getTemperature(), myGyro->getAverage());
}

int main(int argc, char **argv)
{
  // robot
  ArRobot robot;

  // la acción joydrive
  ArActionJoydrive joydriveAct;
  // la acción keydrive
  ArActionKeydrive keydriveAct;

  GyroTask gyro(&robot);

// dispositivo del sonar, así el imitador podrá funcionar, y
// debe ser añadido al robot
  ArSonarDevice sonar;

  ArSimpleConnector connector(&argc, argv);
```

```
  if (!connector.parseArgs() || argc > 1)
  {
    connector.logOptions();
    exit(1);
  }

  // Inicio obligatorio
  Aria::init();

printf("Este programa permite el uso del joystick o teclado para controlar el robot.\nSe puede usar las teclas para manejar y la barra espaciadora para parar.\nPara control con el joystick presionar el botón de disparo y luego maneje.\nPresione la tecla ESC para salir.\n");

  // Si no hay un joystick, se debe dar a conocerlo al robot
  if (!joydriveAct.joystickInited())
    printf("No tiene joystick, solo las flecjas en el teclado funcionarán.\n");

  // establece que el joystick no haga nada si no esta
  // presionado el botón
  joydriveAct.setStopIfNoButtonPressed(false);

  // añadir el sonar al robot
  robot.addRangeDevice(&sonar);

  // trata de conectar, si falla finaliza
  if (!connector.connectRobot(&robot))
  {
    printf("Could not connect to robot... exiting\n");
    Aria::shutdown();
    return 1;
  }

  robot.comInt(ArCommands::ENABLE, 1);

  robot.addAction(&joydriveAct, 50);
  robot.addAction(&keydriveAct, 45);

// establece la acción joydrive de tal manera que permita a la
// acción keydrive empezar si no existe algún botón presionado
```

```
  joydriveAct.setStopIfNoButtonPressed(false);

  // ejecuta el robot, true para que el pueda finalizar si la
  // conexión se ha perdido
  robot.run(true);

  // Finalización
  Aria::shutdown();
  return 0;
}
```

6.3. Evasión de Obstáculos

La evasión de obstáculos es unas de las aplicaciones más comunes y a la vez más útiles dentro de la robótica, ya que permite un manejo tanto autónomo como controlado de las plataformas robóticas de tal manera que el robot pueda evadir posibles obstáculos en su trayectoria, así como decidir cuando se ha encontrado con uno y poder así tomar las mejores decisiones que permitan reposicionar el robot, ejemplo figura 6.3.

Figura. 6.3. **Aplicación de evasión de obstáculos siguiendo una trayectoria**

Esta aplicación permite involucrar la mayor parte de los sensores del robot, como por ejemplo el parachoques, los sonares tanto frontales como traseros (opcionales), también se pueden añadir otros tipos de sonares como por ejemplo láser o la brújula que sirven para localización y posicionamiento del robot.

A continuación se muestra el código fuente de la aplicación en la cual se indica como se desarrolló para una mejor comprensión de la aplicación.

```
gotoActionExample.cpp
#include "Aria.h"

int main(int argc, char **argv)
{
  Aria::init();
  ArArgumentParser argParser(&argc, argv);
  argParser.loadDefaultArguments();
  ArRobot robot;
  ArRobotConnector robotConnector(&argParser, &robot);
  ArLaserConnector laserConnector(&argParser, &robot, &robotConnector);

  // Siempre se trata de conectar con el primer laser:
  argParser.addDefaultArgument("-connectLaser");

  if(!robotConnector.connectRobot())
  {
    ArLog::log(ArLog::Terse, "No se pudo conectar al robot.");
    if(argParser.checkHelpAndWarnUnparsed())
    {
        // -help no dado, solo salir.
        Aria::logOptions();
        Aria::exit(1);
    }
  }

  // Análisis de argumentos de disparo
  if (!Aria::parseArgs() || !argParser.checkHelpAndWarnUnparsed())
  {
    Aria::logOptions();
    Aria::exit(1);
```

```
  }

  ArKeyHandler keyHandler;
  Aria::setKeyHandler(&keyHandler);
  robot.attachKeyHandler(&keyHandler);

  puts("Este programa hará funcionar al robot en modo wander. Se usa
algunas acciones de evasión\n"
  "si se detecta obstáculos, de otra manera solo se mantendrá\n"
  "una velocidad constante hacia adelante.\n\nPresione CTRL-C o ESC para
salir.");

  ArSonarDevice sonar;
  robot.addRangeDevice(&sonar);

  robot.runAsync(true);

  // Trata de conectarse con el láser. Si falla, lo da a
  // conocer, y continua usando solo el sonar
  if(!laserConnector.connectLasers())
  {
    ArLog::log(ArLog::Normal, "Advertencia: no se puede conectar con los
laser requeridos, el modo wander solo usará el sonar.");
  }

  // enciende los motores, apaga los sonidos del robot amigobot
  robot.enableMotors();
  robot.comInt(ArCommands::SOUNDTOG, 0);

  // añade un conjunto de acciones para combinarlas en el modo
  // de evasión de obstáculos y para tener el comportamiento
  // deseado
  ArActionStallRecover recover;
  ArActionBumpers bumpers;
  ArActionAvoidFront avoidFrontNear("Avoid Front Near", 225, 0);
  ArActionAvoidFront avoidFrontFar;
  ArActionConstantVelocity constantVelocity("Constant Velocity", 400);
  robot.addAction(&recover, 100);
  robot.addAction(&bumpers, 75);
  robot.addAction(&avoidFrontNear, 50);
  robot.addAction(&avoidFrontFar, 49);
```

```
  robot.addAction(&constantVelocity, 25);

  // espera un lazo de tareas para terminar antes de finalizar
  // el programa
  robot.waitForRunExit();

  Aria::exit(0);
}
```

Esta es una aplicación sencilla en la que se involucra el tipo de programación mediante acciones que fue especificado en el capítulo anterior, cuya función es detectar un obstáculo mediante cualquiera de los sensores y luego evadirlo, para esta aplicación se utilizan acciones para los sonares y los parachoques, así como también se especifica que se debe hacer en caso de encontrarse con un obstáculo en la parte posterior o anterior del robot y a qué velocidad debe moverse posterior al encuentro para poder evadir dicho obstáculo.

6.4. Control de Visión

La aplicación de control de visión permite desarrollar muchas aplicaciones para la vida real, en la actualidad es la más utilizada dentro del campo de reconocimiento de trayectorias, objetos y en casos más avanzados puede identificar expresiones humanas, así como también diferenciación de colores, como ejemplo se puede visualizar en la figura 6.4.

Para demostrar el uso de esta herramienta, se ha implementado una aplicación la cual consiste en detectar una línea un objeto identificado previamente en la aplicación ACTS y luego desarrollar el programa que permita al robot identificar la información necesaria y de esa manera pueda cumplir con el objetivo propuesto.

Para comenzar con esta aplicación se hace uso del cliente ACTS, el cual es una aplicación que permite configurar los parámetros de la cámara así como también la imagen con la cual vamos a trabajar para que el robot reconozca la trayectoria o el objeto deseado. Para ello se sigue los pasos indicados con anterioridad en la explicación de ACTS.

Posterior a ello se debe escribir el código que permitirá seguir a la trayectoria u objeto deseado, y se muestra a continuación.

Figura. 6.4. **Identificación de colores**

```
actsColorFollowingExample.cpp
#include "Aria.h"

// Sigue una acción que mueve el robot a través de la trama más larga que
// aparezca en el campo de visión del robot.
class Chase : public ArAction
{

public:

  // Muestra el estado de la acción a seguir
  enum State {
    NO_TARGET,      // No existe objetivo en la visión
    TARGET,         // Existe objetivo en la visión
  };

  // Constructor
  Chase(ArACTS_1_2 *acts, ArVCC4 *camera);

  // Destructor
```

```
  ~Chase(void);

  // La acción
  ArActionDesired *fire(ArActionDesired currentDesired);

  // Establece el canal de ACTS del cual se quiere obtener la trama a
  // seguir
  bool setChannel(int channel);

  // Retorna el estado actual de la acción
  State getState(void) { return myState; }

  // Alto y ancho de pixeles desde el frame-grabber
  enum {
    WIDTH = 160,
    HEIGHT = 120
  };

protected:
  ArActionDesired myDesired;
  ArACTS_1_2 *myActs;
  ArVCC4 *myCamera;
  ArTime myLastSeen;
  State myState;
  int myChannel;
  int myMaxTime;
};

// Constructor: Inicializa la acción a seguir
Chase::Chase(ArACTS_1_2 *acts, ArVCC4 *camera) :
    ArAction("Chase", "Chases the largest blob.")
{
  myActs = acts;
  myCamera = camera;
  myChannel = 0;
  myState = NO_TARGET;
  setChannel(1);
  myLastSeen.setToNow();
  myMaxTime = 1000;
}
```

```
// Destructor
Chase::~Chase(void) {}

// La acción a seguir
ArActionDesired *Chase::fire(ArActionDesired currentDesired)
{
  ArACTSBlob blob;
  ArACTSBlob largestBlob;

  bool flag = false;

  int numberOfBlobs;
  int blobArea = 10;

  double xRel, yRel;

  // Resetea la acción deseada
  myDesired.reset();

  numberOfBlobs = myActs->getNumBlobs(myChannel);

  // Si hay tramas a seguir se las establece aqui
  if(numberOfBlobs != 0)
  {
    for(int i = 0; i < numberOfBlobs; i++)
    {
      myActs->getBlob(myChannel, i + 1, &blob);
      if(blob.getArea() > blobArea)
      {
        flag = true;
        blobArea = blob.getArea();
        largestBlob = blob;
      }
    }

    myLastSeen.setToNow();
  }

  // Si no se tiene tramas durante un tiempo
  if (myLastSeen.mSecSince() > myMaxTime)
  {
```

```
    if(myState != NO_TARGET) ArLog::log(ArLog::Normal, "Target Lost");
    myState = NO_TARGET;
  }
  else
  {
    // Si se observa una trama no vista antes
    if(myState != TARGET) {
      ArLog::log(ArLog::Normal, "Target Aquired");
      ArLog::log(ArLog::Normal, "(Using channel %d with %d blobs)",
myChannel, numberOfBlobs);
    }
    myState = TARGET;
  }

  if(myState == TARGET && flag == true)
  {
    // Determina el centro de gravedad d la trama y es relativo al centro
    // de la cámara
    xRel = (double)(largestBlob.getXCG() - WIDTH/2.0)  / (double)WIDTH;
    yRel = (double)(largestBlob.getYCG() - HEIGHT/2.0) / (double)HEIGHT;

    // Dirige la cámara hacia la trama
    if(!(ArMath::fabs(yRel) < .20))
    {
      if (-yRel > 0)
        myCamera->tiltRel(1);
      else
        myCamera->tiltRel(-1);
    }

    // Establece la dirección y velocidad del robot
    if (ArMath::fabs(xRel) < .10)
    {
      myDesired.setDeltaHeading(0);
    }
    else
    {
      if (ArMath::fabs(-xRel * 10) <= 10)
        myDesired.setDeltaHeading(-xRel * 10);
      else if (-xRel > 0)
        myDesired.setDeltaHeading(10);
```

```
      else
        myDesired.setDeltaHeading(-10);

    }

    myDesired.setVel(200);
    return &myDesired;
  }

  // Si no se tiene un objetivo, no se establece ninguna acción y se la
  // pasa a una prioridad menor para control del robot.
  return &myDesired;
}

// Establece el canal desde el cual se obtendrá la información de la
// trama.
bool Chase::setChannel(int channel)
{
  if (channel >= 1 && channel <= ArACTS_1_2::NUM_CHANNELS)
  {
    myChannel = channel;
    return true;
  }
  else
    return false;
}

// Llama a habilitar o no la acción de manejo a través de teclado
toggleAction(ArAction* action)
{
  if(action->isActive()) {
    action->deactivate();
    ArLog::log(ArLog::Normal, "%s action is now deactivated.", action-
>getName());
  }
  else {
    action->activate();
    ArLog::log(ArLog::Normal, "%s action is now activated.", action-
>getName());
  }
}
```

```
// Función principal
int main(int argc, char** argv)
{
  Aria::init();

  // El robot
  ArRobot robot;

  // Una tecla de manejo para obtener entradas desde el teclado
  ArKeyHandler keyHandler;
  Aria::setKeyHandler(&keyHandler);

  // Se activa el sonar para una evasión de obtáculos básica
  ArSonarDevice sonar;

  // Cámara (Cannon VC-C4)
  ArVCC4 vcc4 (&robot);

  // ACTS, Para seguimiento de las tramas de color
  ArACTS_1_2 acts;

  // Argumentos de las líneas de comandos
  ArArgumentParser argParser(&argc, argv);
  argParser.loadDefaultArguments();

  // Manera simple de conexión (takes arguments from argParser)
  ArSimpleConnector simpleConnector(&argParser);

  // Argumentos de análisis
  if (!Aria::parseArgs())
  {
    Aria::logOptions();
    keyHandler.restore();
    Aria::shutdown();
    return 1;
  }

  // Acciónes de limitación de movimiento (si se detecta obstáculos)
  ArActionLimiterForwards limiter("speed limiter near", 350, 800, 200);
```

```
  ArActionLimiterForwards limiterFar("speed limiter far", 400, 1250,
300);
  ArActionLimiterBackwards backwardsLimiter;
  ArActionConstantVelocity stop("stop", 0);
  //ArActionConstantVelocity backup("backup", -200);

  // Acción que determina el color a seguir definido a través de:
  Chase chase(&acts, &vcc4);

  // Acción de teleoperación mediante teclado
  ArActionKeydrive keydriveAction;

  // Usa “a” para activar o desactivar el manejo a través de teclado
  keyHandler.addKeyHandler('a', new
ArGlobalFunctor1<ArAction*>(&toggleAction, &keydriveAction));

  // Permite a Aria conocer el manejo a través de teclado
  Aria::setKeyHandler(&keyHandler);

  // Añade una tecla de manejo al robot
  robot.attachKeyHandler(&keyHandler);

  // Añade el sonar al robot
  robot.addRangeDevice(&sonar);

  // Conexión al robot
  if (!simpleConnector.connectRobot(&robot))
  {
    ArLog::log(ArLog::Terse, "Error: Could not connect to robot...
exiting\n");
    keyHandler.restore();
    Aria::exit(1);
  }

  // Abre una conexión a ACTS
  if(!acts.openPort(&robot))
  {
    ArLog::log(ArLog::Terse, "Error: Could not connect to ACTS...
exiting.");
    keyHandler.restore();
    Aria::exit(2);   }
```

```
  // Inicia la cámara
  vcc4.init();

  // Espera un Segundo...
  ArUtil::sleep(1000);

  // Mantiene el robot para que vaya a maxima velocidad
  robot.setAbsoluteMaxTransVel(400);

  // Habilita los motores
  robot.comInt(ArCommands::ENABLE, 1);

  // Espera...
  ArUtil::sleep(200);

  // Añade las acciones al robot en orden descendente de importancia.
  robot.addAction(&limiter, 7);
  robot.addAction(&limiterFar, 6);
  robot.addAction(&backwardsLimiter, 5);
  robot.addAction(&keydriveAction, 4);
  robot.addAction(&chase, 3);
  robot.addAction(&stop, 1);

  // Empieza con una acción de manejo mediante teclado deshabilitada. Usa
  // “a” para encenderlo
  keydriveAction.deactivate();

  // Corre el ciclo de proceso del robot hasta que se pierda la conexión
  ArLog::log(ArLog::Normal, "Running. Train ACTS to detect a color to
drive towards an object, or use 'a' key to switch to keyboard driving
mode.");
  robot.run(true);

  Aria::shutdown();
  return 0;
}
```

Como se puede observar en el programa, la función de detección de tramas o colores la establece el cliente ACTS, es en este programa donde se debe establecer previamente que color o que trayectoria se debe seguir, tal como se lo indico anteriormente en la sección correspondiente a ACTS.

Luego de establecer la acción a seguir se lo involucra directamente con el programa estableciendo de esta manera el canal del cual se debe tomar información, así como que debe hacer el robot en caso de que no tenga objetivo o si lo tenga. Luego de ello realiza el seguimiento de la trayectoria mientras no exista un obstáculo que detenga al robot.

ACTS es el encargado de enviar toda la información de la trama al robot, mientras que el robot se encarga de otro tipo de acciones como establecer velocidades, acciones de acuerdo a la prioridad y también la habilitación de sensores, manejo mediante teclado. Es aquí en este ejemplo donde se puede observar el verdadero potencial del entorno cliente – servidor dentro del cual funcionan las plataformas robóticas Pioneer3 DX y Pioneer3 AT.

Como se ve no es tan complejo el desarrollo de este tipo de aplicaciones mas si requiere un conocimiento detallado de las acciones que pueden desarrollar las plataformas y los clientes con los cuales se involucra dentro de todas las aplicaciones desarrolladas a través de este proyecto.

CAPÍTULO 7

PRUEBAS Y RESULTADOS

Luego de realizar las respectivas pruebas de las aplicaciones descritas anteriormente se han obtenido los siguientes resultados:

7.1. Seguimiento de Trayectorias

En el seguimiento de trayectorias el comportamiento de las plataformas robóticas es aceptable, no presenta mayores inconvenientes en su manejo, pero se destacarán algunas observaciones que son de mucha consideración:

- La trayectoria probada en los robots, fue dibujada en base a los planos reales de los laboratorios del departamento de Eléctrica y Electrónica, y consistió en ir desde un punto ubicado en la puerta de entrada al laboratorio de CIM hasta la entrada del laboratorio de electrofluidos.

- El robot Pionner3 DX que fue la que paso la prueba recorrió sin mayores inconvenientes esta prueba, con excepción de las partes en donde el espacio por donde paso el robot era inferior a 50cm de ancho, y en las áreas abiertas en donde no existían paredes.

- Dentro de las áreas pequeñas el robot no podía identificar un espacio para poder continuar con su trayectoria, siendo la principal causa los parámetros de configuración del sonar, los cuales fueron probados con parámetros de fábrica y posteriormente modificados para un mejor comportamiento.

- En las áreas abiertas el robot si bien continuaba con la trayectoria tendía a perderse ya que no tiene parámetros de referencia en donde guiarse, esto sucedía mientras al robot se le asignaba varias metas intermedias en la trayectoria antes de llegar al punto final.

- Otro aspecto a tomar muy en cuenta es el poner varios puntos intermedios de metas antes de llegar a la meta final, por lo cual el robot se perdía con mucha frecuencia, ya que tendía a reposicionarse mucho.

- Finalmente la prueba más exitosa fue al configurar los parámetros de los sonares de acuerdo al entorno del mapa dibujado, y con tan solo un punto de inicio y un punto de llegada, mientras en la trayectoria se reposicionaba automáticamente para poder cumplir con el objetivo.

7.2. Control de Giros

Esta aplicación es una de las más sencillas realizadas, pero es de vital importancia para poder tener el control preciso del robot. En la prueba realizada con este programa se obtuvieron algunos resultados que deben considerarse al momento de utilizar el giroscopio:

- Se probó el giroscopio de dos maneras diferentes, en sentido horario y anti horario, tanto con movimiento continuo como también con giros desde 0 grados hasta 360 grados con intervalos de 90 grados.

- En el movimiento continuo en sentido horario se pudo apreciar como el giroscopio realiza de una manera bastante precisa los giros deseados, en las cuales se probaron velocidades desde 10 hasta 100, en todas ellas se podía comprobar que el robot regresa con mucha precisión al punto de partida original 0,0,0.

- En el movimiento continuo en sentido anti horario se probaron velocidades desde -10 hasta -100, en todas ellas se podía comprobar que el robot regresa con mucha precisión al punto de partida original 0,0,0.

- Finalmente se probaron los giros hacia un determinado punto deseado desde el inicio hasta los 90, 180 y 270 grados, en los cuales el giroscopio calcula la ruta más corta es decir si se encuentra en el punto de origen y requiere ir hacia 90 grados girará en sentido horario 90 grados, mientras que si se le pide ir a la posición 270 grados girará desde el inicio 90 grados en sentido anti horario.

- El robot escoge la ruta más corta hacia el ángulo deseado dependiendo del último punto en el cual se encuentre.

7.3. Evasión de Obstáculos

En esta aplicación se obtuvieron algunos resultados bastante positivos, ya que el robot cumplió con su objetivo de manera muy eficiente, pero cabe recalcar algunas observaciones hechas, las cuales deben tenerse muy en cuenta:

- Las pruebas se hicieron tanto en lugares pequeños como en espacios abiertos, en las dos el robot evade los obstáculos considerados grandes, con los pequeños que estén a su altura tiene un poco de problema sin embargo los parachoques puede corregir estas deficiencias de los sonares.

- Si algún objeto pequeño como por ejemplo el cable de la computadora se cruza en frente de uno de los sonares provoca un mal funcionamiento ya que no permite identificar bien los posibles objetos que se encuentren en frente del robot.

- Cuando existen objetos en movimiento en frente del robot, como por ejemplo las piernas de la gente, no permiten identificar con mucha precisión los objetos y a veces tiende a chocar con los parachoques, tomando un tiempo para recuperarse de esa parada.

- Para un mejor rendimiento se configuró los sonares de acuerdo al espacio en donde se desenvuelva, tanto para espacio abierto en los cuales es mejor tener una respuesta

más rápida y un poco lejana, como para ambiente cerrado en los cuales se requiere que pueda circular en espacios más pequeños sin chocarse.

7.4. Control de Visión

Esta aplicación es una de las más importantes de las cuales se debe tomar en cuenta, ya que puede servir para muchos más usos en la vida real, las pruebas realizas y sus resultados obtenidos se detallan a continuación:

- Se probó la visión para reconocer tres tipos de objetos tanto pequeños, medianos y grandes, todos ellos con colores diferentes, pero con un mismo ángulo de visión.

- Para los objetos pequeños se escogió una pieza rectangular de madera de color amarillo de dimensiones aproximadas de 8 x 4 x 1 cm, la cual se la ubicó en el suelo y se la procedió a mover en diferentes ángulos, lo que se obtuvo fue que el robot reconoce el objeto y no confunde con otros objetos del ambiente, pero el reconocimiento del objeto es un poco lento.

- Posteriormente se escogió una caja de color café de dimensiones aproximadas de 30 x 20 x 25 cm, con la cual se hizo el mismo procedimiento que con la pieza de color amarillo, si bien la respuesta es un poco más rápida luego tendía a confundirse con otras gamas de color café como el color de la puerta del laboratorio, sin embargo cumplió su objetivo de identificar la caja.

- Finalmente la última prueba fue con una persona, a la cual se hizo identificar un color azul del pantalón; esta fue la prueba más satisfactoria ya que el robot tenía una respuesta más rápida y no confundirá los colores, además que se pudo apreciar el comportamiento de otros sensores como los sonares que evitaban que el robot choquen con la persona y permitan seguirla.

  De todas formas la respuesta no es tan rápida como debería ser para una aplicación real, la cual se podría mejorar en base a programación.

CAPÍTULO 8

CONCLUSIONES Y RECOMENDACIONES

8.1. Conclusiones

- El entorno cliente – servidor que manejan las plataformas robóticas Pionner P3-DX y la P3-AT es sin duda la mejor manera de realizar aplicaciones sin sobrecargar mucha información dentro del microcontrolador, lo cual permite un mejor desempeño al momento de realizar las operaciones necesarias para cumplir el objetivo deseado.

- El uso de programas separados para el cliente y el servidor hace más eficiente el desarrollo de aplicaciones, es decir, existe un programa específico para cada cosa como ya se detallo anteriormente, así por ejemplo ARIA es un programa de tipo más general, mientras otros programas como por ejemplo el ACTS se dedica específicamente a la sección de video, lo cual hace más fácil desarrollar cada aplicación dependiendo la misma y utilizar así un solo programa o varios en conjunto.

- Las aplicaciones MobileSim y Mapper3Basic son muy eficientes aunque presentan algunas limitaciones, pero de todas formas cumplen muy bien su función tanto de simular las aplicaciones desarrolladas, así como de trazar mapas respectivamente, aunque cabe recalcar que en algunos casos no se comporta la simulación tal como en el entorno real, especialmente cuando se trata de entornos abiertos, es decir que no presentan obstáculos o paredes.

- Los accesorios y sensores probados en las aplicaciones se comportan de una manera predecible, aunque en muchos casos se debe estudiar previamente el entorno donde se va a realizar la aplicación, ya que en muchos casos se requiere de una configuración adecuada para un funcionamiento óptimo del robot dentro del entorno, dentro de los sensores y accesorios que funcionaron de mejor manera se encuentran la cámara de visión, el giroscopio y los sonares mostrando gran precisión y eficiencia en cada aplicación desarrollada.

- Dentro de las aplicaciones desarrolladas que utilizaron solo sensores, las de mejor desempeño fueron las desarrolladas en interiores, ya que en exteriores el robot tendía a perderse debido a que no tenía objetos de referencia, sin embargo al activar el giroscopio mejoraba el comportamiento de manera significativa.

- En la aplicación de visión no hubo mayores inconvenientes, excepto con los objetos pequeños y también aquellos que tenían colores opacos que la cámara confundía con otros colores del entorno.

8.2. Recomendaciones

- Antes de iniciar la operación del robot es recomendable verificar que las baterías se encuentren correctamente cargadas y al ubicarlas dentro del robot, verificar que se sitúen en el lado correcto de modo que no haya problemas con la polaridad y evitar así un corto circuito.

- Si el robot no se encendiera por alguna razón o simplemente queda en standby sin realizar ninguna operación se debe verificar el fusible que cada plataforma posee, si se encontrase bien éste se debe reinstalar el software ARCOS en la plataforma; de esta manera quedará reiniciada correctamente. Se debe tomar en cuenta que se debe instalar de preferencia la última versión del software ubicada en el CD que tiene este trabajo de complemento.

- Se recomienda verificar los parámetro de configuración en la conexión cliente – servidor, ya que de otra manera no será posible realizar la comunicación entre la plataforma y el PC; de manera general los parámetros ya vienen por defecto configurados, sin embargo se debe tener muy en cuenta que la comunicación siempre se realiza por defecto en el puerto COM1 del PC.

- En el momento de realizar los programas se recomienda seguir la arquitectura mostrada como ejemplo en estas aplicaciones, ya que se debe seguir un estricto orden para que los programas se compilen de manera adecuada.

- Se debe tomar muy en cuenta que las aplicaciones se desarrollaron en visual C++ y debido a las múltiples formas de programación mostradas en los capítulos anteriores, éstas solo se compilan en el programa Visual Studio 2005 o 2008, siendo esta última versión la utilizada para las aplicaciones.

  Si se utiliza versiones anteriores se deben hacer muchas correcciones, y es muy posible que en la mayoría de los casos no se compile la aplicación. Esta es la recomendación **más importante** a tomarse en cuenta.

- Es recomendable que antes de empezar a utilizar las plataformas, se conozca un poco de su funcionamiento y de preferencia que se tenga conceptos muy claros en cuestiones de programación, así como de robótica ya que existen parámetro muy delicados como por ejemplo los valores PID, que pueden afectar el desempeño de la plataforma con valores no adecuados.

- Si bien las plataformas son hechas con un cuerpo de aluminio y son robustas, no se debe abusar de ello y usarlas dentro de los rangos especificados para cada plataforma como se indicó anteriormente, además se debe realizar un mantenimiento periódico de las mismas en aspectos de limpieza, aire de las llantas y en el caso de la plataforma P3-AT ajustar las cadenas de la transmisión.

ANEXOS

9.1. Anexo A

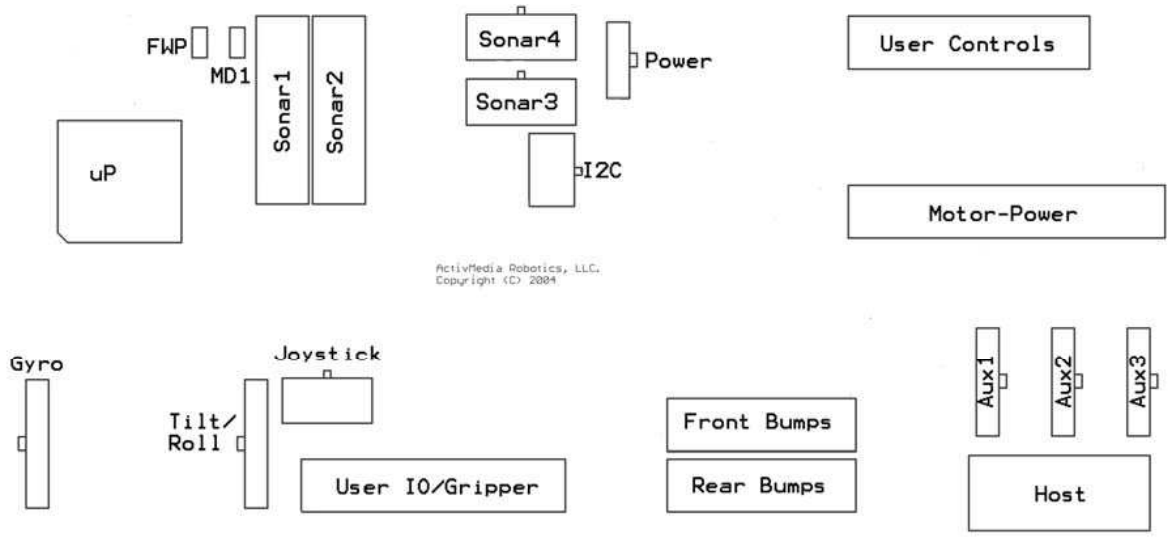

Figura. 9.1**. Sensores en la Placa del Microcontrolador SH2**

9.2. Anexo B

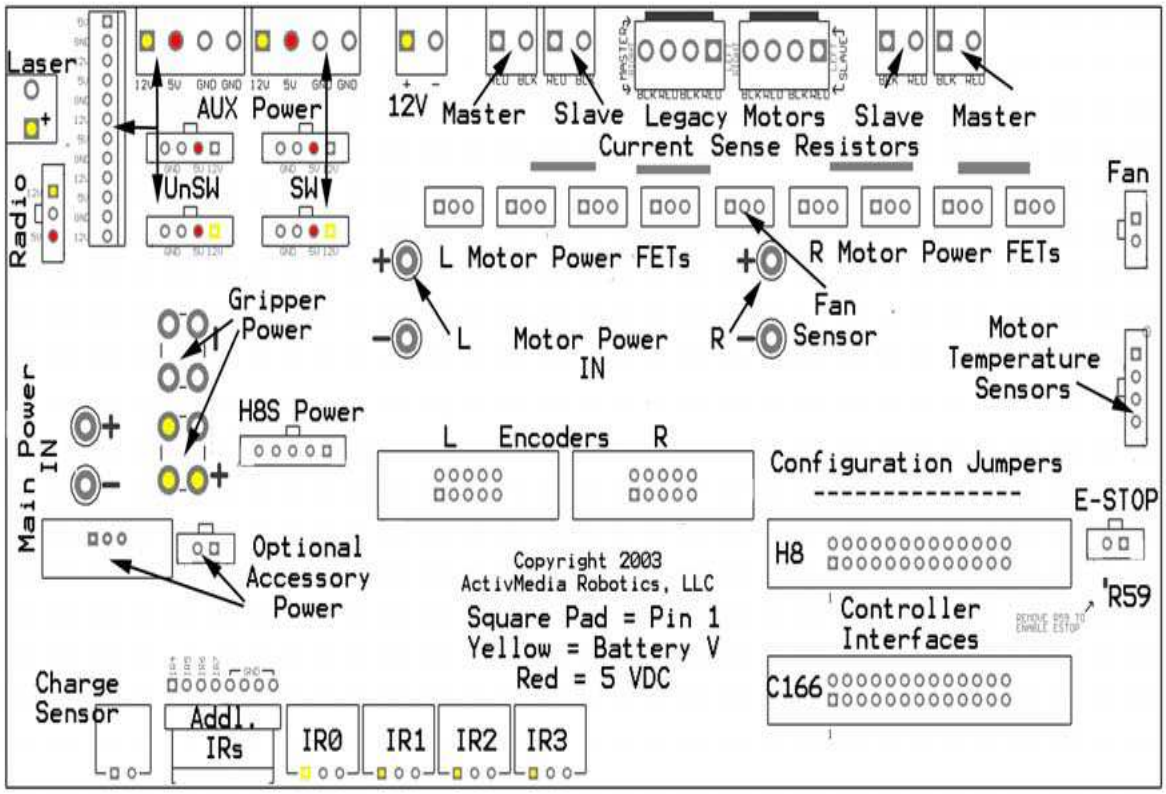

Figura. 9.2. **Placa de Control de Motores**

9.3. Anexo C

Ejemplo de Localización usando SONARNL

Este ejemplo sirve para localizar y también provee al servidor una operación remota con MobileEyes:

```
 */
 #include "Aria.h"
 #include "ArNetworking.h"
 #include "Arnl.h"

 int main(int argc, char *argv[])
 {
   Aria::init();
   Arnl::init();

   ArRobot robot;
   ArServerBase server;
   ArArgumentParser parser(&argc, argv);
   parser.loadDefaultArguments();
   ArSimpleConnector simpleConnector(&parser);
   ArServerSimpleOpener simpleOpener (&parser);

   // Establece el giroscopio
   ArAnalogGyro gyro (&robot);

   // Analiza argumentos para el simple connector.
   if (!Aria::parseArgs ())
   {
     ArLog::log (ArLog::Normal, "\nUsage: %s -map mapfilename\n",
argv[0]);
     Aria::logOptions ();
     Aria::exit (1);
   }

   // Si no se usa Sonarnl, se usa el SICK laser
 #ifndef SONARNL
   ArSick sick;
   robot.addRangeDevice (&sick);
 #endif // ifndef SONARNL
```

```
   // Sonar, usado por Sonarnl, así como por los modos de
   // teleoperación y wander.
   ArSonarDevice sonarDev;
   robot.addRangeDevice (&sonarDev);

   // Usa el directorio "examples" como el lugar para mantener los
   // archivos de mapas
   char fileDir[1024];
   ArUtil::addDirectories (fileDir, sizeof (fileDir), Aria::getDirectory
(),"examples");



   // Establece el mapa, esto hará que se busque archives en el
   // directorio de ejemplos
      ArMap arMap (fileDir);
   // Ignora los archivos con nombre vacío
   arMap.setIgnoreEmptyFileName (true);

 #ifndef SONARNL
   // Inicializa el Arnl para localización por láser
   ArLocalizationTask locTask (&robot, &sick, &arMap);
 #else
   // Inicializa Sonarnl para localización por sonar
   ArSonarLocalizationTask locTask (&robot, &sonarDev, &arMap);
 #endif // ifndef SONARNL

   // Añade controles de inicio para configuración
   ArLog::addToConfig (Aria::getConfig ());

   // Abre el Puerto del servidor
   if (!simpleOpener.open (&server, fileDir, 240))
   {
     if (simpleOpener.wasUserFileBad ())
       ArLog::log (ArLog::Normal, "Bad user file");
     else
       ArLog::log (ArLog::Normal, "Could not open server port");
     exit (2);
   }

   // Conexión al robot
```

```
  if (!simpleConnector.connectRobot (&robot))
  {
    ArLog::log (ArLog::Normal, "Could not connect to robot... exiting");
    Aria::exit (3);
  }

  robot.enableMotors ();
  robot.clearDirectMotion ();

  // Resetea el simulador a us posición de inicio
  robot.comInt (ArCommands::RESETSIMTOORIGIN, 1);

#ifndef SONARNL
  // Establece el laser antes que el robot lo use
  simpleConnector.setupLaser (&sick);
#endif
// ifndef SONARNL

   // Empieza el lazo de tareas del robot
  robot.runAsync (true);

#ifndef SONARNL
  // Empieza el lazo de tareas del láser
  sick.runAsync ();

  // Conexión al láser
  if (!sick.blockingConnect ())
  {
    ArLog::log (ArLog::Normal, "Couldn't connect to sick, exiting");
    Aria::exit (4);
  }
#endif // ifndef SONARNL

  ArUtil::sleep (300);

  // Parachoques.
  ArBumpers bumpers;
  robot.addRangeDevice (&bumpers);

  // Regiones prohibidas del mapa
  ArForbiddenRangeDevice forbidden (&arMap);
```

```
   robot.addRangeDevice (&forbidden);

   // Objetos que proveen servicios de red.
   ArServerInfoRobot serverInfoRobot (&server, &robot);
   ArServerInfoSensor serverInfoSensor (&server, &robot);
   ArServerInfoLocalization serverInfoLocalization (&server, &robot,
&locTask);
   ArServerHandleLocalization serverLocHandler (&server, &robot,
&locTask);
#ifdef SONARNL
   ArServerHandlerMap serverMap (&server, &arMap,
ArServerHandlerMap::LINES);
#else
   ArServerHandlerMap serverMap (&server, &arMap,
ArServerHandlerMap::BOTH);
 #endif

   // Dibujo de servicios en el despliegue del mapa:
   ArServerInfoDrawings drawings (&server);
   drawings.addRobotsRangeDevices (&robot);

   // Comandos
   ArServerHandlerCommands commands (&server);
   ArServerSimpleComUC uCCommands (&commands, &robot);
   ArServerSimpleComMovementLogging loggingCommands (&commands, &robot);
   ArServerSimpleComGyro gyroCommands (&commands, &robot, &gyro);
   ArServerSimpleComLogRobotConfig configCommands (&commands, &robot);

   // Establece los posibles modos para control remoto desde un cliente
   // como MobileEyes:

   // Para parar y mantener parado:
   ArServerModeStop modeStop (&server, &robot);

   // Si se tiene laser de crea esta clase que deshabilita
   // automáticamente el sonar cuando se para y lo enciende cuando se
   // mueve de nuevo
 #ifndef SONARNL
   ArSonarAutoDisabler sonarAutoDisabler (&robot);
 #endif
```

```
   // Teleoperación por teclado, joystick, etc:
   ArServerModeRatioDrive modeRatioDrive (&server, &robot);

   // Modos de configuración de teleoperación y comandos especiales:
   modeRatioDrive.addControlCommands (&commands);
   modeRatioDrive.addToConfig (Aria::getConfig (), "Teleop settings");

   // Previene el manejo normal si la localización se pierde:
   ArActionLost actionLostRatioDrive (&locTask, NULL, &modeRatioDrive);
   modeRatioDrive.getActionGroup ()->addAction (&actionLostRatioDrive,
110);

   // Modo Wander:
   ArServerModeWander
   modeWander (&server, &robot);

   // Previene la evasión de obstáculos si se pierde el robot:
   ArActionLost
   actionLostWander (&locTask, NULL, &modeWander);
   modeWander.getActionGroup ()->addAction (&actionLostWander, 110);

   // Provee una pequeña tabla de información interesante para el cliente
   // para desplegar al operador:
      ArServerInfoStrings
      stringInfo (&server);
   Aria::getInfoGroup ()->addAddStringCallback (stringInfo.
                          getAddStringFunctor ());

   // Despliega los puntos de localización y las estadísticas de
   // comunicación del láser si no hay el SonArnl:
 #ifdef SONARNL
   Aria::getInfoGroup ()->addStringDouble ("Localization Score", 8,
     new ArRetFunctorC < double, ArSonarLocalizationTask > (&locTask,
                                &ArSonarLocalizationTask::
                                getLocalizationScore),
                               "%.03f");
 #else

   Aria::getInfoGroup ()->addStringDouble ("Localization Score", 8,
                     new ArRetFunctorC < double,
                     ArLocalizationTask > (&locTask,
```

```
                                    &ArLocalizationTask::
                                    getLocalizationScore),
                        "%.03f");
   Aria::getInfoGroup ()->addStringInt ("Laser Packet Count", 10,
                        new ArRetFunctorC < int, ArSick > (&sick,
                                    &ArSick::
                                    getSickPacCount));
 #endif

 Aria::getInfoGroup ()->addStringInt ("Motor Packet Count", 10,
                        new ArConstRetFunctorC < int,
                        ArRobot > (&robot,
                           &ArRobot::getMotorPacCount));

 #ifndef SONARNL
   // Establece el punto de recarga si hay un accesorio de carga
   // automática en el robot.
   ArServerModeDock *modeDock = NULL;
   if (modeDock != NULL)
   {
     modeDock->checkDock ();
     modeDock->addAsDefaultMode ();
     modeDock->addToConfig (Aria::getConfig ());
     modeDock->addControlCommands (&commands);
   }
 #endif
// ifndef SONARNL

   modeStop.addAsDefaultMode ();

    // Crea el servicio que permite al cliente cambiar los parámetros de
   // configuración en el ArConfig
   ArServerHandlerConfig handlerConfig (&server, Aria::getConfig (),
                        Arnl::getTypicalDefaultParamFileName (),
                        Aria::getDirectory ());

    // Lectura de los parámetros en los archivos.
   Aria::getConfig ()->useArgumentParser (&parser);
   if (!Aria::getConfig ()->parseFile (Arnl::getTypicalParamFileName ()))
   {
```

```
     ArLog::log (ArLog::Normal, "Trouble loading configuration file,
exiting");
     Aria::exit (5);
   }
   // Previene acerca de parámetros desconocidos.
   if (!simpleOpener.checkAndLog () || !parser.checkHelpAndWarnUnparsed
())
   {
     ArLog::log (ArLog::Normal, "\nUsage: %s -map mapfilename\n",
argv[0]);
    simpleConnector.logOptions ();
    simpleOpener.logOptions ();
    Aria::exit (6);
  }
  // Error si no hay mapa
  if (arMap.getFileName () == NULL || strlen (arMap.getFileName ()) <= 0)
   {
     ArLog::log (ArLog::Terse,
         "Warning, no map given. Use the -map command-line argument or
modify the config using MobileEyes or by editing the parameter file.");
     ArLog::log (ArLog::Terse,
         "See the ARNL documentation, including MAPPING.txt or
SONAR_MAPPING.txt.");
   }
   ArLog::log (ArLog::Normal, "Directory for maps and file serving: %s",
fileDir);
   ArLog::log (ArLog::Normal, "See the ARNL README.txt for more
information");

    robot.unlock();

   // Localiza el robot en el punto de inicio.
   locTask.localizeRobotAtHomeBlocking();
   server.runAsync();

   ArLog::log(ArLog::Normal, "Server now running on port %d. Press ol-C
to exit.", server.getTcpPort());

   robot.waitForRunExit();
   Aria::exit(0);
  }
```

9.4. Anexo D

Ejemplo de Direccionamiento usando SONARNL

Este ejemplo sirve para entender el direccionamiento de trayectoria y también provee al servidor una operación remota con MobileEyes:

```
*/
#include "Aria.h"
#include "ArNetworking.h"
#include "ArSystemStatus.h"
#include "Arnl.h"

int
main(int argc, char *argv[])
{
  // Inicializa localización de Aria, Arnl y sus argumentos.
  Aria::init();
  Arnl::init();

  // Carga a un archivo
  //ArLog::init(ArLog::File, ArLog::Normal, "log.txt", true, true);
  //ArLog::init(ArLog::File, ArLog::Verbose);

  // Para adquirir el uso del CPU e información wireless desde Linux
  ArSystemStatus::runRefreshThread();

  // El objeto robot
  ArRobot robot;

  // Servidor
  ArServerBase server;

  // Análisis de los argumentos de la línea de comandos.
  ArArgumentParser parser(&argc, argv);

   // Establece el simpleConnector
   ArSimpleConnector simpleConnector(&parser);

   // Establece el simpleOpener
   ArServerSimpleOpener simpleOpener(&parser);
```

```
  // Establece el cliente para el servidor central
  ArClientSwitchManager clientSwitch(&server, &parser);

  // Carga los argumentos por defecto para la computadora
  parser.loadDefaultArguments();

  // Establece el giroscopio
  ArAnalogGyro gyro(&robot);

  // Analiza argumentos para el simple connector.
  if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
  {
    ArLog::log(ArLog::Normal, "\nUsage: %s -map mapfilename\n", 0]);
    Aria::logOptions();
    Aria::exit(1);
  }

 #ifndef SONARNL
  // El objeto laser se usará si se tiene uno
  ArSick sick;

  // Añade el laser al robot
  robot.addRangeDevice(&sick);
#endif

  // Sonar, debe ser añadido al robot, usado por teleoperación y
  // el modo wander para detector obstáculos, y para localización si
  // esta el SONARNL
 ArSonarDevice sonarDev;

 // Añade el sonar al robot
 robot.addRangeDevice(&sonarDev);

 // Establece donde se debe buscar por archivos
 char fileDir[1024];
 ArUtil::addDirectories(fileDir, sizeof(fileDir), Aria::getDirectory(),
            "examples");

  // Establece el mapa, buscará archives en el directorio de ejemplos (a
  // menos que el nombre del archive empiece con /,\,o.)
  // Se puede sacar el argumento 'fileDir' para buscar en el actual
```

```
  // directorio
     ArMap arMap(fileDir);
  // se establece para ignorer archivos con nombres vacios (de otra
  // manera el parseFile en el config fallará )
  arMap.setIgnoreEmptyFileName(true);


#ifdef SONARNL
  // Realiza la tarea de ruta (para el láser)
  ArPathPlanningTask pathTask(&robot, &sonarDev, &arMap);
#else
  // Realiza la tarea de ruta (para el láser)
  ArPathPlanningTask pathTask(&robot, &sick, &sonarDev, &arMap);
#endif

 // Establece cosas para que los datos puedan ser logueados (solo lo
 // hace con el láser debido a que corre con una conexión serial a 9600)

 ArDataLogger dataLogger(&robot);
 dataLogger.addToConfig(Aria::getConfig());

 // Añade el logueamiento al config
 ArLog::addToConfig(Aria::getConfig());

 // Primero se abre el servidor
 if (!simpleOpener.open(&server, fileDir, 240))
 {
   if (simpleOpener.wasUserFileBad())
     ArLog::log(ArLog::Normal, "Bad user file");
   else
     ArLog::log(ArLog::Normal, "Could not open server port");
   exit(2);
 }

 // Conexión al robot
 if (!simpleConnector.connectRobot(&robot))
 {
   ArLog::log(ArLog::Normal, "Could not connect to robot... g");
   Aria::exit(3);
 }

 // Establece una clase que pone el movimiento y los parámetros del
```

```
 // giroscopio del ArConfig
  ArRobotConfig robotConfig(&robot);
  robotConfig.addAnalogGyro(&gyro);

  robot.enableMotors();
  robot.clearDirectMotion();

  // Si se conecta al simulador, lo resetea para empezar
  robot.comInt(ArCommands::RESETSIMTOORIGIN, 1);
  robot.moveTo(ArPose(0,0,0));


#ifndef SONARNL
  // Establece el laser usando el conector
  simpleConnector.setupLaser(&sick);
#endif

  // Empieza el lazo de tareas del robot
  robot.runAsync(true);

#ifndef SONARNL
  // Empieza el lazo de tareas del láser
  sick.runAsync();

  // Intenta conectarse al láser
  if (!sick.blockingConnect())
    ArLog::log(ArLog::Normal, "Couldn't connect to SICK laser, it be
used");
  else
    ArLog::log(ArLog::Normal, "Connected to laser.");
#endif


  // Añade dispositivos de rango adicionales al robot y a la tarea de
  // planificación de rutas
 // IRs si el robot los tiene
  robot.lock();
  ArIRs irs;
  robot.addRangeDevice(&irs);
  pathTask.addRangeDevice(&irs, ArPathPlanningTask::CURRENT);

  // Parachoques.
```

```
  ArBumpers bumpers;
  robot.addRangeDevice(&bumpers);
  pathTask.addRangeDevice(&bumpers, ArPathPlanningTask::CURRENT);

  // Regiones prohibidas desde el mapa
  ArForbiddenRangeDevice forbidden(&arMap);
  robot.addRangeDevice(&forbidden);
  pathTask.addRangeDevice(&forbidden, ArPathPlanningTask::CURRENT);

  // Esta es la parte para añadir un dispositivo de rango el cual
  // mantiene los datos del sensor y los borra apropiadamente para
  // replantear los bloques de rutas establecidos

   ArGlobalReplanningRangeDevice replanDev(&pathTask);

   // Crea objetos que anadem nuevos servicios de red

   // Dibujo en el display del mapa:
   ArServerInfoDrawings drawings(&server);
   drawings.addRobotsRangeDevices(&robot);
   drawings.addRangeDevice(&replanDev);

   /* Si se desea dibujar el destino habilitar este código:
   ArServerDrawingDestination destination(
       &drawings, &pathTask, "destination",
       500, 500,
       new ArDrawingData("polyDots",
                 ArColor(0xff, 0xff, 0x0),
                 800, // size
                 49), // just below the robot
   */

   /* Si se desea ver el plan local de área de ruta se usa esto
   ArDrawingData drawingDataP("polyLine", ArColor(200,200,200), 1, 75);
   ArFunctor2C<ArPathPlanningTask, ArServerClient *, ArNetPacket *>
   drawingFunctorP(pathTask, &ArPathPlanningTask::drawSearchRectangle);
   drawings.addDrawing(&drawingDataP, "Local Plan Area",
&drawingFunctorP);
   */

   /* Si se desea ver los puntos que componen el plan local además de
```

```
   // los del plan principal se usa esto:
   ArDrawingData drawingDataP2("polyDots", ArColor(0,128,0), 100, 70);
   ArFunctor2C<ArPathPlanningTask, ArServerClient *, ArNetPacket *>
   drawingFunctorP2(pathTask, &ArPathPlanningTask::drawPathPoints);
   drawings.addDrawing(&drawingDataP2, "Path Points", &drawingFunctorP2);
   */

   // Comandos simples:
   ArServerHandlerCommands commands(&server);

    // Se prove varios tipos de información al cliente:
   ArServerInfoRobot serverInfoRobot(&server, &robot);
   ArServerInfoSensor serverInfoSensor(&server, &robot);
   ArServerInfoPath serverInfoPath(&server, &robot, &pathTask);
   serverInfoPath.addSearchRectangleDrawing(&drawings);
   serverInfoPath.addControlCommands(&commands);

   // Provee el mapa al cliente (y sus controles relacionados):
   ArServerHandlerMap serverMap(&server, &arMap);

    // Añade algunos commandos simples para pruebas y debugging:
   ArServerSimpleComUC uCCommands(&commands, &robot);
// Envía cualquier commando al microcontrolador
   ArServerSimpleComMovementLogging loggingCommands(&commands, &robot);
// configure el logueo
   ArServerSimpleComGyro gyroCommands(&commands, &robot, &gyro);
// Monitorea el giroscopio
   ArServerSimpleComLogRobotConfig configCommands(&commands, &robot);
// Activa el registro de parámetro de configuración del robot
   ArServerSimpleServerCommands serverCommands(&commands, &server);
// Monitorea el comportamineto de la red

   /* Establece los modos posibles para el control remoto desde un
      cliente como MobileEyes:
    */

   // Modo para ir a una meta u otro punto específico:
   ArServerModeGoto modeGoto(&server, &robot, &pathTask, &arMap,
ArPose(0,0,0));

   // Añade un comando simple que permite dar una lista de metas par air
```

```
  // en recorrido, en vez de dar todas.
   modeGoto.addTourGoalsInListSimpleCommand(&commands);

   // Modo para parar y mantener parado:
   ArServerModeStop modeStop(&server, &robot);

 #ifndef SONARNL
   // Causa que el sonar se apague automáticamente cuando el robot se
   // para, y se enciende cuando se envían comandos para moverlo. (Nota,
   // esto no se debe hacer si se necesita los datos del sonar para
   // localizar)
   ArSonarAutoDisabler sonarAutoDisabler(&robot);
 #endif

   // Modos de Teleoperación para manejar por teclado, joystick,etc:
   ArServerModeRatioDrive modeRatioDrive(&server, &robot);
   // Nuevo modo mejorado
   ArServerModeDrive modeDrive(&server, &robot);
   // Viejo modo para compatibilidad
   // Modos de configuración de manejo:
   modeRatioDrive.addToConfig(Aria::getConfig(), "Teleop settings");
   modeDrive.addControlCommands(&commands);
   modeRatioDrive.addControlCommands(&commands);

   // Modo Wander
   ArServerModeWander modeWander(&server, &robot);

    // Se provee una pequeña table de información para el cliente
   // para despleguar al operador:
   ArServerInfoStrings stringInfo(&server);
   Aria::getInfoGroup()-
>addAddStringCallback(stringInfo.getAddStringFunctor());

   Aria::getInfoGroup()->addStringDouble(
       "CPU", 8,
       new ArGlobalRetFunctor<double>(&ArSystemStatus::getCPUPercent),
       "%.03f");

 #ifndef SONARNL
   Aria::getInfoGroup()->addStringInt(
       "Laser Packet Count", 10,
```

```
      new ArRetFunctorC<int, ArSick>(&sick,
                    &ArSick::getSickPacCount));
#endif

  Aria::getInfoGroup()->addStringInt(
      "Motor Packet Count", 10,
      new ArConstRetFunctorC<int, ArRobot>(&robot,
                          &ArRobot::getMotorPacCount));

   // Hace el modo stop por defecto
  modeStop.addAsDefaultMode();

  /* Servicios de transferencia de archivos: */

#ifdef WIN32
  // Esta parte de archives de servidor no funciona bajo windows aún
ArLog::log(ArLog::Normal, "Note, file upload/download services are not
implemented for Windows; not enabling them.");
#else
  // Este bloque permite establecer de donde se toma y se ponen los
  // archivos hacia/donde
  // /*
  ArServerFileLister fileLister(&server, fileDir);
  ArServerFileToClient fileToClient(&server, fileDir);
  ArServerFileFromClient fileFromClient(&server, fileDir, "/tmp");
  ArServerDeleteFileOnServer deleteFileOnServer(&server, fileDir);
  // */
#endif

  // Crea el servicio que permite al cliente monitorear la comunicación
  // entre el robot y el cliente.
  //
  ArServerHandlerCommMonitor handlerCommMonitor(&server);

  // Crea el servicio que permite cambiar los parámetros de
  // configuración en el ArConfig
  ArServerHandlerConfig handlerConfig(&server, Aria::getConfig(),
                      Arnl::getTypicalDefaultParamFileName(),
                      Aria::getDirectory());

  // Lee los archives de parámetros.
```

```
   Aria::getConfig()->useArgumentParser(&parser);
   if (!Aria::getConfig()->parseFile(Arnl::getTypicalParamFileName()))
   {
     ArLog::log(ArLog::Normal, "Trouble loading configuration file,
exiting");
     Aria::exit(5);
   }

   // Previene acerca de parámetros desconocidos.
   if (!simpleOpener.checkAndLog() || !parser.checkHelpAndWarnUnparsed())
   {
     ArLog::log(ArLog::Normal, "\nUsage: %s -map mapfilename\n",
argv[0]);
     simpleConnector.logOptions();
     simpleOpener.logOptions();
     Aria::exit(6);
   }

   // Previene si no existe un mapa
   if (arMap.getFileName() == NULL || strlen(arMap.getFileName()) <= 0)
   {
     ArLog::log(ArLog::Normal, "");
     ArLog::log(ArLog::Normal, "### Warning, No map file is set up, you
can make a map with sickLogger or guiServer, and/or Mapper3; More info in
docs/Mapping.txt and README.txt. Set the map with the -map command line
option, or by changing the config with MobileEyes or by editing the
config file.");
    ArLog::log(ArLog::Normal, "");
  }

  // Averigua dond se quiere poner los archives
  ArLog::log(ArLog::Normal, "");
  ArLog::log(ArLog::Normal,
         "Directory for maps and file serving: %s", fileDir);

  ArLog::log(ArLog::Normal, "See the ARNL README.txt for more
information");
   ArLog::log(ArLog::Normal, "");

  // Si se quiere que MobileSim trate de cargar el mismo mapa que se usa
  // en el guiServer se debe habilitar el comentario en la siguiente
```

```
  // línea y este objeto enviará el comando a MobileSim para realizarlo,
  // pero se debe asegurar que se empiece el MobileSim desde el
  // directorio Arnl/examples o usar la opción --cwd, para que los
  // nombres de los mapas usados por MobileSim sean iguales a los usados
  // por el guiServer
   //ArSimMapSwitcher mapSwitcher(&robot, &arMap);

    /* Finalmente, se emepiza a corer el robot: */
   robot.unlock();
   server.runAsync();

   // Añade un key handler (Usa esc en windows para salir del programa)

   ArKeyHandler *keyHandler;
   if ((keyHandler = Aria::getKeyHandler()) == NULL)
   {
     keyHandler = new ArKeyHandler;
     Aria::setKeyHandler(keyHandler);
     robot.lock();
     robot.attachKeyHandler(keyHandler);
     robot.unlock();
     printf("To exit, press escape.\n");
   }
   robot.waitForRunExit();
   Aria::exit(0);
}
```

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO DE TÉRMINOS

ACTS

ActivMedia Color Tracking System, Sistema de Rastreo de Color ActivMedia

API

Interfaz de Programación para Aplicaciones

ARCOS

Advanced Robotics Control Operating System, Sistema Robótico Avanzado de Control y Operación

ARIA

Advanced Robotics Interface for Applications, Interfaz Robótica Avanzada para Aplicaciones

ARNetworking

Advanced Robotics Networking, Red Avanzada de Robótica

Blob

Regiones continuas de pixeles interesantes (colores que aparecen en el canal actual)

Bumpers

Parachoques

DARPA

Agencia de Investigación de Proyectos Avanzados de Defensa

DGPS

Differential Global Position System, Sistema Diferencial de Posicionamiento Global, proporciona a los receptores de GPS correcciones de los datos recibidos de los satélites GPS, con el fin de proporcionar una mayor precisión en la posición calculada.

DLL

Dynamic Link Library, Librerías de enlace dinámico

FLTK

Fast Light Tool Kit, Kit de Herramientas para Luz Rápida

Framegrabber

Placa de Captura, en sistema de tarjeta de video

LUT

Tabla de búsqueda de color

PCMCIA

Personal Computer Memory Card International Association, Asociación Internacional que desarrolla tarjetas de memoria utilizadas en los ordenadores personales.

PID

Proporcional Integral Derivativo

PWM

Modulación de Ancho de Pulso

PTZ

Pan Tilt Zoom, Encuadre Giro Zoom, características de algunas cámaras que permite setear hasta 16 posiciones para ajustar rápidamente la visión del lente de la cámara.

RGB

Red, Green, Blue, Gama de colores rojo, verde, azul

SIP

Paquete de información del Servidor

SonARNL

Sonar Autonomous Robotic Navigation & Localization, Sonar para Navegación y Localización Robótica Autónoma

STL

Plantilla Estándar de la Librería

SWIG

Interfaz que conecta un programa desarrollado en C o C++ con una plataforma de programación de alto nivel.

TCM2

Accesorio que integra inclinómetro, magnetómetro, termómetro y brújula y se conecta a uno de los puertos seriales del microcontrolador

TCP/IP

Transmission Control Protocol, Protocolo de Control de Transmisión, Protocolo de Internet para enviar datos a través de la red.

UDP

User Datagram Protocol, Protocolo de Datagrama de Usuario, red para la transmisión de datos que no requieren la confirmación del destinatario de los datos enviados

VFW

Video para Windows

REFERENCIA BIBLIOGRÁFICA

- ***Pioneer3 Operations Manual,*** 2007 MobileRobots Inc. all rights reserved, 75 págs.

- ***ActivMedia Color Tracking System User Manual,*** Diciembre 2003 ActivMedia Robotics LLC, version 6, all rights reserved, 47 págs.

- ***Pioneer 3-DX, 3-AT, PeopleBot & PowerBot***, Software and Documentation

- ***VC-C50i/VC-50iR Communication Camera Instruction Manual,*** Junio 2005 Manual de Instrucciones CANON CAMERAS, 246 págs.

- http://www.mobilerobots.com, Software, Hardware, MobileRobots Inc.

- http://www.activmedia.com, Software, Aplicaciones

- http://www.robots.mobilerobots.com, Software and Documentation

- http://www.imperx.com/frame-grabbers/vce-pro, Drivers and Documentation

- http://www.canon.es/For_Home/Product_Finder/Web_Cameras/Web_Cameras/vb_c50i/index.asp, Drivers, Manual, Documentation

- http://www.usa.canon.com/consumer/controller?act=ModelInfoAct&tabact=DownloadDetailTabAct&fcategoryid=362&modelid=11206, Software and Documentation

Entregado, Junio del 2010

Ing. Víctor Proaño
DIRECTOR DE CARRERA

Jimena Morales
AUTOR

Daniel Jaramillo
AUTOR