DEPARTAMENTO DE ELÉCTRICA Y ELECTRÓNICA

CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

TESIS PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

TEMA: DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE NAVEGACIÓN POR VOZ PARA ROBOTS MÓVILES CON RUEDAS (RMR) UTILIZANDO LA PLATAFORMA DE DESARROLLO "MICROSOFT ROBOTICS DEVELOPER STUDIO

AUTOR: MANUEL MAURICIO SUNTAXI LLUMIQUINGA

DIRECTOR: ING. GORDILLO RODOLFO

CODIRECTOR: ING. IBARRA ALEXANDER

SANGOLQUI

2015

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

CERTIFICADO

Ing. Rodolfo Gordillo Msc.

Ing. Alexander Ibarra Msc.

CERTIFICAN

Que el trabajo de titulación **"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE NAVEGACIÓN POR VOZ PARA ROBOTS MÓVILES CON RUEDAS (RMR) UTILIZANDO LA PLATAFORMA DE DESARROLLO "MICROSOFT ROBOTICS DEVELOPER STUDIO"**, realizado por el señor Manuel Mauricio Suntaxi Llumiquinga ha sido guiado y revisado periódicamente y cumple normas estatutarias establecidas por la institución, en el Reglamento de Estudiantes de la Universidad de las Fuerzas Armadas – ESPE.

Debido a que se trata de un trabajo de investigación recomiendan su publicación.

El mencionado trabajo consta de un documento empastado y un disco compacto en cual contiene los archivos en formato portátil de Acrobat (pdf), Autorizan al Sr. Manuel Mauricio Suntaxi Llumiquinga que lo entregue al Ingeniero Luis Orozco, en calidad de Coordinador de la Carrera.

Sangolquí, 09 de Abril del 2015

Ing. Rodolfo Gordillo MSc.
DIRECTOR

Ing. Alexander Ibarra MSc.
CO-DIRECTOR

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

Yo, MANUEL MAURICIO SUNTAXI LLUMIQUINGA

DECLARO QUE:

El proyecto de grado denominado **"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE NAVEGACIÓN POR VOZ PARA ROBOTS MÓVILES CON RUEDAS (RMR) UTILIZANDO LA PLATAFORMA DE DESARROLLO "MICROSOFT ROBOTICS DEVELOPER STUDIO"**, ha sido desarrollado en base a una investigación exhaustiva, respetando los derechos intelectuales de terceros, conforme las citas que constan al pie de las paginas correspondientes, cuyas fuentes se incluyen en la bibliografía.

Consecuentemente este trabajo es de mi autoría.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico del proyecto de grado en mención.

Sangolquí, 09 de Abril de 2015

Manuel Mauricio Suntaxi Llumiquinga

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL

AUTORIZACIÓN

Yo, Manuel Mauricio Suntaxi Llumiquinga

Autorizo a la Universidad de las Fuerzas Armadas – ESPE la publicación, en la biblioteca virtual de la Institución del trabajo **"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE NAVEGACIÓN POR VOZ PARA ROBOTS MÓVILES CON RUEDAS (RMR) UTILIZANDO LA PLATAFORMA DE DESARROLLO "MICROSOFT ROBOTICS DEVELOPER STUDIO"** cuyo contenido, ideas y criterios son de mi exclusiva responsabilidad y autoría.

Sangolquí, 09 de Abril de 2015

Manuel Mauricio Suntaxi Llumiquinga

DEDICATORIA

A mis padres por su ejemplo de superación, honradez y constancia, quienes en los momentos más difíciles de este duro caminar supieron siempre darme el ánimo y las fuerzas necesarias para no desfallecer.

AGRADECIMIENTO

A Dios por brindarme la sabiduría necesaria para terminar lo que un día con tanto anhelo inicie. Por ser siempre el amigo fiel e incondicional que nunca me abandono.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURA

ÍNDICE DE CUADRO

RESUMEN

El desarrollo del presente proyecto de titulación tiene como finalidad el diseño e implementación de un sistema de navegación por voz haciendo uso de la plataforma Microsoft Robotics Deeloper Studio. El desarrollo de nuevas generaciones robóticas, tales como robots personales, asistenciales o mascotas, cada día más integradas a nuestros entornos de trabajo y actividades diarias han obligado a formular nuevos paradigmas de interacción entre robots y humanos, encontrando en el lenguaje oral un medio perfecto para establecer una comunicación entre robot y operador. El modelo de su diseño hace uso de la Arquitectura de Control Automática–Deliberativa. En este modelo se pretende dotar al robot de habilidades y comportamientos que imiten al ser humano en la forma de razonar, decidir y actuar. Con el diseño de la Arquitectura de Control establecida el agente móvil estará en la capacidad de identificar comandos de voz tales como avanzar, retroceder, girar a izquierda, girar a la derecha y media vuelta, a más de ejecutar una tarea autónomamente.

ROBÓTICA MÓVIL

SISTEMAS DE NAVEGACIÓN

ARQUITECTURAS DE CONTROL

MICROSOFT ROBOTICS STUDIO

DSS & CCR

ABSTRACT

The development of this project graduation, aims to design and implement a voice navigation system using the mobile platform Microsoft Robotics Developer Studio. The development of new robotic, such as personal, welfare or pet robots, each day more integrated into our workplaces and daily activities have forced developing new paradigms of interaction between robots and human, finding in speech recognition a perfect channel for establishing a communication between robot and operator. The model of its design makes use of the Architecture Auto-Deliberative Control. In this model is to be providing to robot of skills and behaviors that mimic the human in the form of reason, decide and act. With the design of the architecture of control established the mobile agent will be able to identify voice commands such as forward, backward, turn left, turn right and swing, more than run a task autonomously.

CAPITULO 1

DESCRIPCIÓN GENERAL DEL PROYECTO

1.1. ANTECEDENTES

En el mundo actual el desarrollo de robots móviles autónomos ha comenzado aparecer con mayor auge. En Corea del Sur, empresas como Bandai, Hanool, Grandport, MostiTech y Microrobot, por mencionar sólo algunas, están empezando a introducir robots que son básicamente PC's sobre ruedas, con procesamiento equivalente, memoria, almacenamiento, y capacidades de red, a precios asequibles. Además de lo antes mencionado también disponen de sensores de proximidad para evitar obstáculos, sensores de navegación, auto-recarga de las estaciones base, etc.

Si bien estos robots disponen de varias de las capacidades mencionadas anteriormente, una nueva generación está comenzando aparecer, la cual coordina habilidades de navegación con una diestra manipulación de objetos. Sin duda alguna estos robots siguen teniendo un precio fuera del alcance de un producto de consumo, sin embrago esto demuestra que la tecnología se dirige con rapidez y que las cosas están avanzando dentro del campo de la robótica. Robots con manos y brazos representan un potencial casi ilimitado cuando se combina con los avances en el reconocimiento de voz, visión, la navegación y otras tecnologías críticas que los robots personales requieren.

Esta rápida evolución y avance es muy similar a lo que promovió la evolución de las computadoras y la adopción avanzada de la Web como una parte fundamental de nuestras vidas. Sin embargo, esto tiene un potencial aún mayor, dado que las generaciones emergentes de robots pueden desarrollar sus entornos de

comportamiento ya sea dentro del escritorio o de la Web. Todas las principales economías del mundo están invirtiendo dinero en la investigación de robots y nuevas empresas, para estimular el desarrollo de la industria. Todos ven a la robótica como un elemento clave en el crecimiento económico a futuro.

Al igual que con la industria de las computadoras, el éxito no depende sólo de los avances del hardware, un software es el que realmente impuso el crecimiento de la industria del PC. Es por ello que en la actualidad existen diversas plataformas robóticas que nos proporcionan un medio de desarrollo de software para la mayoría de los robots móviles autónomos presentes en el mercado, un ejemplo de ello es la herramienta "MICROSOFT ROBOTICS DEVELOPER STUDIO".

1.2. JUSTIFICACIÓN E IMPORTANCIA

El desarrollo de nuevas generaciones robóticas, tales como: robots personales, robots asistenciales o mascotas; cada día más integradas a nuestros entornos de trabajo y actividades diarias, han permitido que en la robótica actual se diferencien dos grandes áreas, la robótica industrial y la robótica de servicio, entendiéndose esta última en un sentido amplio: servicios personales y a la sociedad; obligando de esta manera a formular nuevos paradigmas de interacción entre robots y humanos, encontrando en el lenguaje oral una interfaz perfecta para establecer una comunicación entre el robot y el operador. Tema que interesa de sobre manera, debido a que establece una mejora en la calidad de vida de cualquier ser humano y en especial de aquellas personas con distintas discapacidades, impedidas de realizar algunas acciones y que por medio de la voz podrían dar órdenes a estos agentes para realizar diversas actividades o tareas.

Por las razones anteriormente expuestas es totalmente justificable tener en cuenta el lenguaje oral como un medio de interactuación entre hombre-robot, y en particular, como asistencia a la navegación. Todo esto como consecuencia del importante avance en el amplio campo del procesado de voz.

Este proyecto permitirá establecer otras formas de interoperación entre robots y humanos, ya que mediante la voz, el robot será capaz de recibir órdenes e información y comunicar su estado o situación dentro del entorno en el cual se encuentra desempeñando una determinada tarea o servicio.

1.3. ALCANCE DEL PROYECTO

Se establece el diseño e implementación de un sistema de navegación por voz para el robot móvil con ruedas (RMR) "iRobot Create" utilizando la herramienta de desarrollo "Microsoft Robotics Developer Studio".

Lo que se pretende conseguir con este proyecto es dotar al robot móvil autónomo, de la capacidad de recibir órdenes a través de la voz y de ejecutar las acciones requeridas. En esta primera aproximación, al robot se le indican órdenes de movimiento para guiarlo en un entorno típico de interiores. Para ello, se va a desarrollar un sistema que permita navegar utilizando comandos de voz, de forma segura y efectiva, en un robot móvil. El sistema propuesto, se engloba dentro de la Arquitectura de Control Hibrida (Deliberativa-Reactiva). En esta arquitectura se pretende dotar al robot de habilidades y comportamientos que imiten al ser humano en la forma de razonar y de actuar, y de esta manera, lograr tener una interacción más intuitiva con éste. La Arquitectura Hibrida consta de dos niveles: el nivel Deliberativo, donde se agrupan las capacidades de razonamiento y toma de decisiones, y el nivel Reactivo, que engloba las capacidades de percibir el entorno y actuar sobre él.

Para que el robot sea capaz de procesar la voz y ejecutar los comandos de navegación mediante la voz, es necesario implementar y añadir diferentes módulos en la arquitectura de control hibrida, tales como sensores virtuales, habilidades deliberativas-reactivas y acciones reflejas.

Con el diseño de la arquitectura de control establecida, el RMR estará en la capacidad de:

- Interpretar las órdenes que el operador le asigne, producto del procesamiento digital de las ondas sonoras captadas por un micrófono.

- Realizar una conexión vía Bluetooth, con una PC desktop o una laptop para la recepción y transmisión de datos.

- Notificar su estado o situación dentro del entorno en el cual se encuentra desempeñando un determinado trabajo.

- Ejecutar una labor o asistencia autónomamente, mediante la secuenciación de los comandos de voz suministrados por él operador.

- Desempeñar las tareas y servicios asignados dentro de un entorno de simulación creado para este propósito, con el fin de evitar inconvenientes al momento de su ejecución.

1.4. OBJETIVOS

1.4.1. Objetivo General

Realizar el diseño e implementación de un sistema de navegación por comandos de voz para el robot móvil con ruedas (RMR) “iRobot Create”.

1.4.2. Objetivos Específicos

- Investigar los principios de las diferentes arquitecturas cognitivas de control (reactiva, deliberativa e hibrida) y técnicas de navegación para la resolución del problema de navegación por voz, mediante robótica cognitiva.

- Analizar la herramienta de desarrollo robótica Microsoft Robotics Developer Studio 2008 (MRDS), con finalidad de desarrollar una aplicación de control cognitiva bajo esta plataforma.

- Establecer un enlace inalámbrico entre el RMR y una PC desktop o una laptop utilizando el protocolo de comunicación IEEE 802.15.1 para la recepción y transmisión de datos.

- Generar entidades 3D simuladas para la creación de un entorno virtual de simulación, que será utilizado para determinar el comportamiento y funcionamiento del RMR antes de su implementación.

- Realizar pruebas de operatividad en el ambiente simulado y real, a fin de analizar y comparar los resultados obtenidos en ambos ambientes de trabajo.

CAPITULO 2

FUNDAMENTOS

2.1. ROBÓTICA COGNITIVA

El campo de la robótica cognitiva está muy relacionado con la conciencia o inteligencia artificial, de hecho se puede decir que la conciencia artificial es un subcampo o una línea de investigación especifica dentro de la robótica cognitiva. Cualquier implementación de la funcionalidad de la consciencia tiene que ser enmarcada en el ámbito de una arquitectura cognitiva (Arrabales, 2007). La conciencia artificial por sí sola no tiene ningún sentido e importancia, a no ser que esté integrada dentro de una plataforma física (hardware), la cual permita establecer procesos de percepción y comportamiento, mediante la lectura de los distintos sensores (infrarrojos, ultrasónicos, bumpers, encoders, etc.) y el acceso a los actuadores.

El objetivo principal que se busca al momento de desarrollar arquitecturas cognitivas de control, es dotar al robot la capacidad de razonar, decidir y ejecutar tareas o servicios que se le han encomendado, consiguiendo así que el sistema se vuelva más robusto, adaptativo y dinámico. Los robots personales, asistenciales o mascotas son un muy buen ejemplo de éste tipo de aplicaciones que la robótica cognitiva podría realizar (ver Figura 1). Dotar al robot de la habilidad de interacción con los seres humanos es una tarea muy compleja que requiere de distintas capacidades cognitivas.

Figura 1. Aldebaran Nao Robot
Fuente: Conscious Robots

Los estudios hechos en inteligencia artificial y neurociencias son una excelente fuente de inspiración en el diseño y construcción de arquitecturas cognitivas de control. Todos estos procesos que se intentan simular en los robots construidos artificialmente, tienen una base biológica en la naturaleza, fundamentalmente dentro de los procesos de razonamiento humano que integra aspectos tales como la memoria y sus mecanismos neuronales subyacentes. Sin embargo hay que tomar muy en cuenta que algunas técnicas clásicas usadas en robótica no tienen en cuenta estos conceptos cognitivos, funcionando correctamente solo en entornos controlados, en lo que se refiere a entornos complejos y aplicaciones robustas, son necesarios procesos cognitivos superiores.

Existen cinco fases para el desarrollo de un proceso cognitivo, cada una de las cuales se detallan a continuación:

- ***Sensación:*** Consiste en la captación de datos (señales electrónicas) por parte del robot a través de una infinidad de sensores que pretenden emular diferentes formas de recibir estímulos.

- ***Percepción:*** Más allá de la sensación que recibe los estímulos, la percepción es la capacidad de interpretar los valores de lectura de los sensores y transformarlos en información útil para el robot.

- ***Atención:*** Permite al robot enfocar uno o más "sentidos" en una tarea determinada o en otros elementos del entorno, para centrarse en ellos obviando la información que no cree es útil o pertinente en ese momento, y recabando la que éste considere sea de valor.

- ***Memoria:*** Consiste en una serie de dispositivos de almacenamiento que guardaran información recopilada del robot, para que ésta pueda ser reutilizada a largo plazo.

- ***Razonamiento Inteligente:*** Se refiere a la elaboración de respuestas lo más adecuadas posibles en función del procesamiento digital de la información, es decir a la capacidad que tiene el robot de comportarse o desempeñar una tarea dentro de un ambiente trabajo, es decir adaptarse a medios inciertos, consiguiendo elaborar respuestas más adecuadas.

Uno de los principales problemas dentro del campo de los robots móviles con ruedas es el diseño de arquitecturas de control robustas que puedan dar solución a problemas utilizando robótica cognitiva. Dentro de éste ámbito una de la Teorías de Arquitectura de Control más difundidas y aceptadas en el campo científico es la propuesta por Brooks, en la cual las tareas y comportamientos se enmarcan dentro de capas totalmente jerarquizadas y en las cuales se estable que los niveles inferiores representan conductas más "primitivas" y que estos tienen prioridad sobre las capas superiores.

Un aspecto a tomar en consideración a la hora de diseñar y construir robots cognitivos eficientes es la correcta implementación de una arquitectura cognitiva cuyo papel fundamental dentro de un robot es la organización de sus capacidades

sensoriales y de actuación con el único fin de generar un conjunto de comportamientos basados en modelos psicológicos, es decir conseguir un serie de respuestas bien organizadas y robustas frente a entornos desconocidos y cambiantes, producto de la recopilación de datos que éste haga a través de sus sensores en un instante determinado a fin de que el robot pueda hacer frente a errores producidos dentro de estos escenarios.

Una buena arquitectura de control permite dotar a los robots o agentes móviles de funciones cognitivas superiores que les permiten razonar, decidir y actuar de manera robusta en entornos desconocidos y cambiantes. Éste tipo de robots debe, por ejemplo, ser capaz de razonar acerca de metas, acciones, recursos (lineales o no lineales, discretos y/o continuos, recargables o prescindibles), cuándo percibir y qué buscar, los estados cognitivos de otros agentes, el tiempo, la ejecución de tareas colaborativas, etc. En resumen, la robótica cognitiva se refiere a la integración del razonamiento, la percepción y la acción en un marco de implementación con uniformidad teórica.

2.2. ROBOTS MÓVILES CON RUEDAS

2.2.1. Introducción

De todas las creaciones realizadas por el hombre, los robots son los que más se asemejan al ser humano ya sea en el aspecto físico o en la forma de razonar, decidir y actuar, lo que explica la gran atención que la humanidad ha puesto sobre ellos. Los campos de su desempeño, ampliamente explotado, cubren áreas tan diversas como el ensamblado, procesamiento especial, investigación, empaquetado, soldadura, domésticos, submarinos, médicos, etc.

El concepto de robot como tal, fue utilizado por primera ocasión a principios del siglo XX por el novelista y autor dramático checo Karel Capek en su obra teatral R.U.R. (Rossum’s Universal Robots) estrenada en Praga en 1921 y

publicada en Inglaterra en 1922. El autor empieza a utilizar la palabra checa "robota" que en el eslovaco significa servidumbre o trabajo forzado y cuando se tradujo al inglés se convirtió en robot. En aquellos tiempos la producción en grandes series se había introducido en numerosas compañías, originando una temática de discusión del poder de las máquinas y el sometimiento de la raza humana por parte de las mismas, argumento de muchas obras teatrales y películas del siglo XX (Ollero Baturone, 2007, pág. 2).

Los robots no solo han tenido su espacio dentro del cine o teatro, también han sido parte de una gran temática dentro de la literatura de los años veinte; un muy buen ejemplo de ello es la obra del escritor y científico ruso Isaac Asimov (1920-1992). En el año de 1950 escribe la obra "I Robot", publicación en la cual introduce las tres famosas leyes de la Robótica que según él deben comandar la inteligencia de los robots, y más en general de los humanoides:

1. Un robot no debe dañar a un ser humano o, por inacción, dejar que un ser humano sufra daño.

2. Un robot debe obedecer las órdenes que le son dadas por un ser humano, excepto cuando estas órdenes estén en oposición con la Primera ley.

3. Un robot debe proteger su propia existencia hasta donde ésta protección no esté en conflicto con la Primera o la Segunda Ley.

Una cuarta ley conocida como la ley cero, publicada en Robots e Imperio (1985), es una extrapolación de anteriores: "Un robot no debe perjudicar a la humanidad o, por omisión de acciones, permitir que la humanidad sufra algún daño". Con etas tres leyes, Asimov plantea una solución al problema que Capek esbozo cuando introdujo el concepto de robots en su obra Rossum's Universal Robots. (Torres, Pomares, Gil, Puente, & Aracil, 2002)

Para encontrar los antecedentes históricos de la robótica nos podemos remontar antiguas civilizaciones como la griega donde se hablaba de seres mecánicos con vida que eran movidos por mecanismos construidos con poleas y bombas hidráulicas. Sin embargo, el concepto de robot como tal, comenzó a forjarse en la civilización árabe, donde se le dio sentido a dichos mecanismos para confort del ser humano.

El auge de la Revolución industrial en el último cuarto del siglo XVIII vino de la mano con el diseño de diversos mecanismos automáticos entre los más importantes cabe mencionar el telar de J. Jaquard (1801), maquina programable mediante una cinta de papel perforada, el trabajo de H. Maillardet, creador de una muñeca mecaniza capaz de hacer dibujos.

A inicios del siglo XX y con el desarrollo de la industrialización a raíz de la Segunda Guerra Mundial aparecen los sustratos electrónicos y lógicos capaces de soportar la programabilidad a gran escala. En 1952 se desarrolla una maquina prototipo de control numérico en el Instituto de Tecnológica de Massachusetts (MIT) y en el año de 1954 el estadounidense George Devol patentó un brazo robótico primitivo que se podía programar para realizar tareas específicas.

Los teleoperadores tienen su origen inmediato en los mecanismos para la manipulación de material radioactivo. Estos sistemas hacían posible que un operador detrás de un muro protector con ventanas apropiadas para observar la operación, transmitiera órdenes hacia un manipulador separado de cierta distancia. Consistía en un par de pinzas “maestra” y “esclava” acopladas por mecanismos que permitían que la pinza “esclava” en contacto con el material radioactivo, reprodujera los movimientos de la pinza “maestra”.

Entre los años 80 y 90 un nuevo enfoque en la Robótica se consolido. Éste nuevo tipo de Robótica se denomina Robótica Autónoma. Los Robots Autónomos

(RA) son sistemas completos que operan eficientemente en entornos complejos sin necesidad de estar constantemente guiados y controlados por operadores humanos.

Una propiedad fundamental de los RA es la de poder reconfigurarse dinámicamente para resolver distintas tareas según las características del entorno impuestas en un momento dado. Se hace énfasis en que son sistemas completos que perciben y actúan en entornos dinámicos y parcialmente impredecibles, coordinando interoperaciones entre capacidades complementarias de sus componentes. La funcionalidad de los RA es muy amplia y variada, abarca desde trabajos en entornos inhabitables, a otros que están diseñados para asistir a personas con discapacidad.

Una característica fundamental y diferenciadora de la robótica avanzada será la estrecha colaboración de los robots con los humanos, tanto en el campo industrial como en el doméstico. Ésta nueva sociedad robotizada implicara un importante cambio en el modo y calidad de vida de los ciudadanos.

La integración de Robots asistentes requiere de desarrollo de sistemas avanzados de interacción y cooperación con los humanos que ameritan de sistemas mecánicos más ligeros que no supongan una eventualidad para las personas. Estos sistemas requieren la integración de nuevos materiales, control, electrónica y diseño. Además se deben mejorar los interfaces de usuario que permitan a las personas sin conocimientos de robótica controlar e interactuar con los Robots de forma natural. Para lograr estos objetivos es preciso dotar a los Robots de capacidades cognitivas que conlleven un mayor esfuerzo para mejorar el aprendizaje, razonamiento y toma de decisiones.

2.2.2. Evolución y Últimos Avances

El desarrollo del campo de la robótica móvil o robots móviles con ruedas se ve influenciada con el afán de expandir aún más el amplio campo de la Robótica, ya

que en un inicio está se veía restringida únicamente al alcance de una estructura metálica sujeta en su base (eje cero).

Teniendo como referencia la autonomía que poseen los robots móviles en la actualidad, estos tienen como predecesores dispositivos o sistemas electromecánicos, tales como los denominados "micromouse" desarrollados a partir de los años treinta y cuya función era la de cubrir caminos dentro de laberintos. Otro ejemplo es la tortuga de Walter, desarrollada en 1948 y que era capaz de evadir obstáculos dentro de un ambiente de trabajo, subir pendientes y recurrir a una estación base para auto recargarse.

Los trabajos mencionados no guardan ninguna relación con los robots móviles desarrollados en los años sesenta cuya aplicación dentro de la industria consistía en seguir líneas trazadas en la planta, siendo estos guiados por cables ubicados bajo el suelo o mediante sensores ópticos (Ollero Baturone, 2007).

Con el nacimiento de nuevas tecnologías de razonamiento y planificación entre 1966 y 1972 se creó en el SRI (Standford Research Institute) el primer robot móvil llamado Shakey (Nilsson, 1984) que era una plataforma móvil autónoma controlada por visión artificial y dotada de un dispositivo táctil. A raíz de estos avances la robótica móvil tuvo un crecimiento de sobremanera en el equipamiento de nuevos dispositivos y modernas técnicas de control.

A inicios de la década de los setenta, el robot Newt fue desarrollado por Hollis (Hollis, 1977). El robot Hilare desarrollado en el LAAS (Laboratory for Analysis and Architecture System) en Francia (Giralt, 1979). En el JPL (Jet Propulsión Laboratory) se desarrolló el Lunar Rover, diseñado particularmente para la exploración planetaria (Thompson, 1977). A finales de esa década, Moravec desarrolló el robot Stanford Cart, capaz de seguir una trayectoria delimitada por una línea establecida en una superficie, en el SAIL (Stanford Artificial Intelligence Lab) (Moravec, 1979). En 1983, el robot Raibert, fue desarrollado en el MIT

(Instituto de Tecnología de Massachusetts), este fue un robot de una sola pata diseñado para estudiar la estabilidad de éstos sistemas (Raibert, 1986).

En 1994, el Instituto de robótica CMU (Carnegie Mellon University) desarrolló el robot Dante II, un robot móvil de seis patas (Bares & Wettergreen, 199). En 1996 también en el CMU, se desarrolló el robot Gyrover, un sistema sin ruedas y patas cuyo funcionamiento se basaba en un giroscopio (Brown, Frerreira, Shu-Jen, & Chris, 1997). Ese mismo año se desarrolló en el MIT el Spring Flamingo, un robot que emulaba el movimiento de un flamenco (Pratt & Pratt, 1998). La NASA (National Aeronautics and Space Administration) en 1997 envío a Marte un robot móvil teleoperado llamado Sojourner rover, cuyo trabajo era enviar fotografías del entorno de dicho planeta (Brian, 1997). En ese mismo año, la empresa japonesa HONDA, dio a conocer el robot P3, el primer humanoide capaz de imitar movimientos del cuerpo humano (Tanie, 2003). En 1998, se desarrolla en la universidad Waseda en Japón, el WABIAN R-III, un robot humanoide. En 1999 en el CMU, Zeglin propuso un nuevo diseño de robot con una pata llamado Bow Leg Hopper, un diseño que permite almacenar la energía potencial de una pata (Zeglin, 1999).

En 2006, Hollis desarrollo el robot Ballbot, un sistema holónomo cuyo movimiento es proporcionado por una esfera ubicada en la parte inferior de la estructura (Lauwers, Kantor, & Hollis, 2006). En 2004 los robots teleoperados Spirit rover y Opportunity rover fueron parte del Programa de Exploración de Marte de la NASA, el único que se mantiene en servicio hasta la actualidad es el Opportunity. En el 2014 el mundo conoció los drones, vehículos aéreos no tripulados con cámara incorporada, que aunque puedan dar la impresión de ser juguetes de aeromodelismo, su utilidad, está muy por encima de ellos. Los robots citados anteriormente, son solo un fragmento de los muchos que se han creado, no obstante, es fácil darse cuenta que las aplicaciones de estos son extensas y que las mismas son casi ilimitadas como consecuencia del desarrollo diario que vive día a día la tecnología.

La clasificación de los robots móviles se hace tomando en cuenta el tipo de locomoción que estos utilizan, son tres los medios de movimiento más utilizados a la hora de su construcción: por ruedas, por patas y orugas. Aunque estos dos últimos han sido ampliamente investigados, Un avance importante dentro del campo de la robótica móvil se ha conseguido con el desarrollo de los Robots Móviles con Ruedas (RMR). Su principal ventaja radica en la eficiencia de ahorro energético al momento de desplazarse por superficies planas y firmes, además de presentar un mecanismo menos complejo en relación a los robots de patas y orugas, lo cual facilita su estudio, desarrollo y construcción.

Cada una de las características y ventajas mencionadas anteriormente influyen en el desarrollo y estudio de este tipo de robots lo cual origina que debamos tener como primera consideración una definición clara y precisa que abarque todo el contexto de los robots móviles con ruedas. Así se puede definir un RMR como un sistema eléctrico-mecánico controlado, que usa como medio de locomoción algún tipo específico de ruedas y que es capaz de desplazarse de manera autónoma o guiada a través de un ambiente controlado o variable hacia una meta establecida en una determinada área de acción.

Hablar de autonomía en un robot, establece que dicho elemento tiene el total control para determinar su rumbo y trayectoria, a través de un proceso de toma de decisiones basado en la lectura e interpretación de los datos entregados por sus sensores en lugar de seguir una secuencia fija de instrucciones como es el caso de los robots guiados.

Un robot móvil está compuesto básicamente de un arreglo cinemático y un sistema de sensores y actuadores, todos estos componentes están íntimamente ligados y por lo general su estudio se lo debería realizar en conjunto, pero se ha visto que existe un mejor resultado de avance y desarrollo al tratarlo y estudiarlos de manera independiente.

2.2.3. Configuraciones Cinemáticas de los Robots Móviles

2.2.3.1. Vehículos con Ruedas

Los robots móviles con ruedas son los dispositivos más recomendados a la hora de desarrollar tareas en ambientes de trabajo libres de obstáculos y superficies compactas, consiguiendo desplazarse así a velocidades relativamente altas.

Como desventaja de su configuración cinemática estos robots están sujetos a posibles deslizamientos a la hora de su impulsión; tomando en cuenta el medio ambiente en el cual esté desempeñando su tarea éste tipo de vehículos también están sujetos a vibraciones y resbales lo cual acarrea problemas al dotarlo de habilidades tales como localización y planificación, razón por la cual no se recomienda su uso en superficies blandas.

Los RMR utilizan distintos prototipos de locomoción en base a ruedas que les otorga diferentes propiedades y características en relación a sus dimensiones, eficiencia energética y maniobrabilidad. Dependiendo de la configuración cinemática que lo conforme, los RMR utilizan cuatro tipos de ruedas para su locomoción, éstas son: convencionales, tipo castor, ruedas de bolas y omnidireccionales, como se pueden observar en la Figura. 2

Figura 2. Tipos de ruedas
Fuente: Robot Magazine

A continuación se prosigue a detallar brevemente las características más relevantes de las configuraciones cinemáticas más comunes en los RMR.

2.2.3.1.1. Ackerman

Se usa casi exclusivamente en la industria del automóvil. Es la configuración que llevan los coches: dos ruedas con tracción traseras, y dos ruedas de dirección delanteras. Ésta configuración está diseñada para que la rueda delantera interior en un giro tenga un ángulo ligeramente más agudo que la exterior $(\theta_1 < \theta_0)$, como se puede apreciar en la Figura. 3 y así evitar así el derrape de las ruedas.

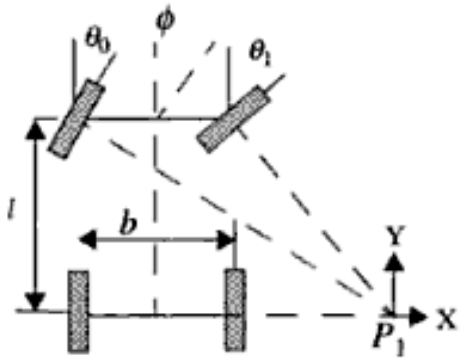


Figura 3. Sistema Ackerman
Fuente: Conscious Robots

Las normales a ambas ruedas se cortan en un punto, que se encuentra sobre la prolongación del eje de las ruedas traseras. Así, se puede comprobar que las trayectorias de ambas ruedas para ángulos de giro constantes son circunferencias concéntricas con centro en el eje de rotación P1.

La relación entre los ángulos de las ruedas de dirección viene dada por la ecuación de Ackerman:

$$\cot(\theta_1) - \cot(\theta_0) = \frac{b}{l}$$

Dónde:

$\theta_1 =$ ángulo relativo de la rueda interior
$\theta_0 =$ ángulo relativo de la rueda exterior
$l =$ separación longitudinal entre ruedas
$b =$ separación lateral entre ruedas

La configuración de Ackerman da una solución bastante eficiente y precisa a la hora de establecer la estimación de la posición durante la navegación (Odometría), a la vez que proporciona un excelente sistema de tracción incluso en superficies inclinadas. Sin embargo la construcción mecánica de un RMR bajo éste modelo cinemático implica una tarea muy compleja a la hora de su ejecución.

2.2.3.1.2. Triciclo clásico

Para éste caso se dispone de tres ruedas en el robot situadas de forma similar a los triciclos de los niños, de ahí su nombre (ver Figura. 4). Las dos ruedas traseras no tienen acoplado ningún motor y sus ruedas se mueven libremente. La rueda delantera sirve tanto para la tracción como para el direccionamiento. El cálculo de la odometría es mucho más sencillo. La posición del robot vendrá dada por el número de pulsos que avanza el encoder de la rueda motora, y la dirección es simplemente la que lleve dicha rueda. La maniobrabilidad es mucho más amplia que en la configuración anterior pero puede presentarse problemas de estabilidad en superficies bruscas y difíciles.

Un problema asociado a ésta configuración es que el centro de gravedad tiende a alejarse de la rueda de tracción en superficies inclinadas mientras el robot lleva la dirección de subida. Esto se ve reflejado en una pérdida de tracción de la rueda

hacia el suelo, lo que implica que la rueda motora siga girando, pero el robot no avance. Esto supone un error acumulado significativo al hacer el análisis odométrico, ya que el robot notifica que su desplazamiento ha sido máximo, cuando en realidad está localizado mucho más atrás.

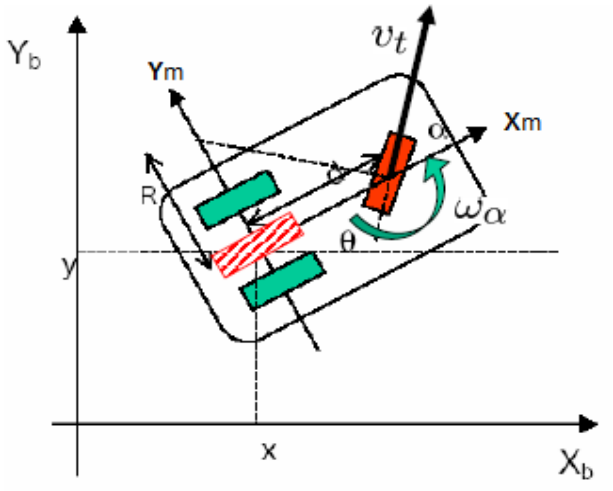


Figura 4. Configuración típica en triciclo

Para una curvatura $\alpha(t)$ medida a partir de la línea de dirección (X_m), el triciclo gira con una velocidad angular $W(t)$ alrededor de una circunferencia cuyo centro de giro es la intersección de las líneas perpendiculares a las ruedas y el radio R es la distancia entre el centro de giro y la línea de dirección.

2.2.3.1.3. Direccionamiento diferencial

La configuración diferencial se presenta como la más sencilla de todas. Consta de dos ruedas situadas diametralmente opuestas en un eje perpendicular a la dirección del robot. Cada una de ellas irá dotada de un motor, de forma que los giros se realizan dándoles diferentes velocidades. Así, si queremos girar a la derecha, daremos mayor velocidad al motor izquierdo. Para girar a la izquierda, será el motor derecho el que posea mayor velocidad. Con dos ruedas es imposible mantener la horizontalidad del robot. Se producen cabeceos al cambiar la

dirección. Para solventar éste problema, se colocan ruedas “locas”. Éstas ruedas no llevan asociadas ningún motor, giran libremente según la velocidad del robot.

Sin embargo, la presencia de más de tres apoyos en el robot (incluidas las dos ruedas de tracción), puede llevar a graves cálculos de odometría en terrenos irregulares, e incluso la pérdida de tracción total. En la Figura. 5, se aprecia cómo la rueda de tracción pierde agarre, haciendo imposible el avance del robot.

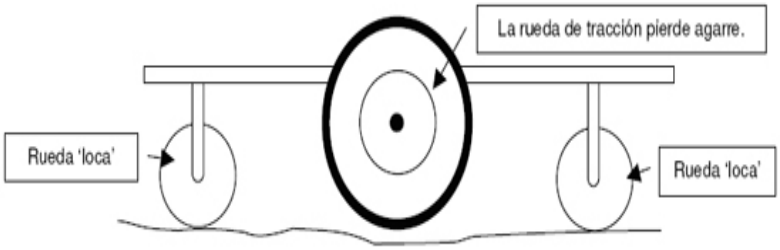


Figura 5. Perdida de tracción en Direccionamiento diferencial

Podemos resumir la configuración diferencial como: dos ruedas con tracción independiente, y una o más ruedas “locas” (ver Figura. 6).

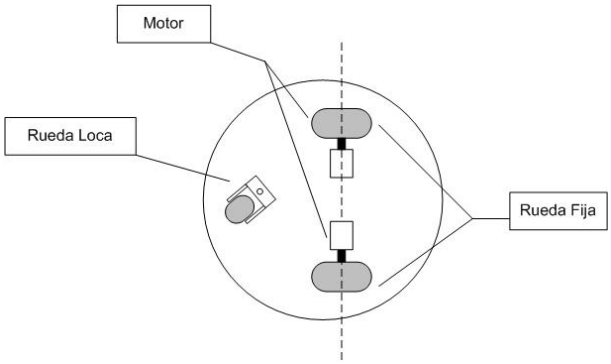


Figura 6. Disposición de las ruedas en configuración diferencial

Para llevar a cabo una navegación por odometría, es necesario acoplar a los motores de las ruedas laterales sendos encoders, de forma que contando los pulsos que avanza cada rueda y teniendo en cuenta el radio de la misma y la reducción del motor, no hay más que aplicar las ecuaciones cinemáticas del robot para hallar la posición exacta en la que se encuentra y el ángulo de desviación respecto a una dirección de referencia.

2.2.3.1.4. Skid Steer

En ésta configuración el robot está equipado de dos ruedas a cada lado que funcionan de manera simultánea. Su movimiento se produce al combinar las velocidades de las ruedas izquierda y derecha. Éste tipo de robots se desempeñan eficientemente en aplicaciones para exteriores como antropología, inspección y obtención de mapas de lugares de difícil y remoto acceso, empleando para ello un sistema de radar ver Figura. 7.

Figura 7. Robot Terregator con locomoción tipo "Skid Steer"
Fuente: Robótica Manipuladores y Robots móviles

2.2.3.1.5. Pistas de deslizamiento

Son plataformas móviles en las que el direccionamiento y la impulsión se logran por medio de pistas de deslizamiento. Tienen una configuración similar a la del Skid Steer, aclarando que las pistas en este caso actúan de forma análoga a ruedas extra diametrales. Éste tipo de configuración presenta un buen rendimiento bajo ambientes difíciles o superficies irregulares teniendo en consideración que la resistencia al desgaste es mayor y la impulsión está menos limitada por el desplazamiento ver Figura. 8.

Figura 8. Robot con locomoción mediante pistas de deslizamiento
Fuente: Robótica Manipuladores y Robots móviles

2.2.3.1.6. Síncronas

Consiste en una configuración innovadora, dispone de tres o más ruedas con tracción y acopladas mecánicamente, de tal manera que todas rotan a la misma velocidad y en la misma dirección. La transmisión se realiza mediante correas concéntricas.

Ésta configuración requiere de una gran sincronización, que redunda en una odometría mejorada reduciendo el deslizamiento de las ruedas respecto al suelo, ya que todas las ruedas generan fuerzas con vectores de igual modulo y paralelos en

todo momento. En la Figura. 9 se puede apreciar la complejidad mecánica que requiere una configuración de éste tipo.

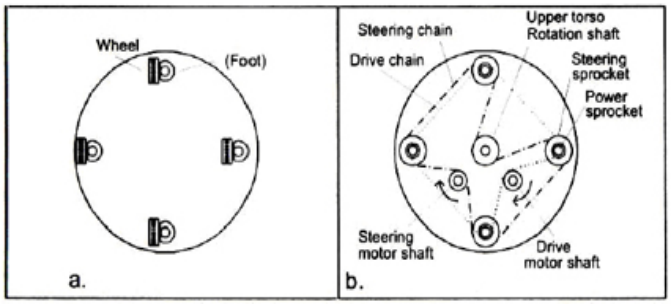


Figura 9. Mecánica necesaria para diseñar una configuración sincronizada
Fuente: Robótica Manipuladores y Robots móviles

2.2.3.1.7. Tracción omnidireccional

Éste tipo de sistema consta de tres ruedas directrices y motrices. Ésta configuración tiene tres grados de libertad, por lo que puede realizar cualquier movimiento, y posicionarse en cualquier orientación y localidad. No presenta restricciones cinemáticas como se puede visualizar en la figura 10

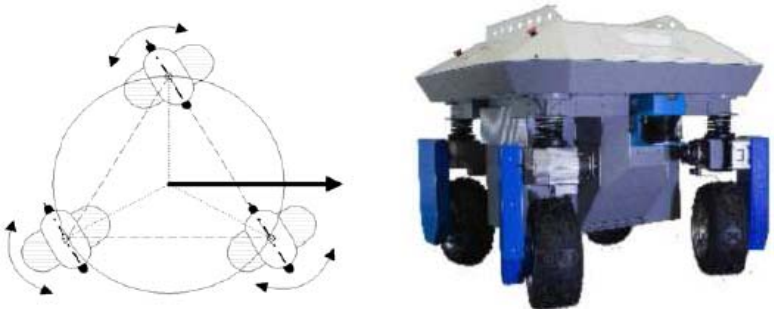


Figura 10. Diagrama de la configuración omnidireccional
Fuente: Robótica Manipuladores y Robots móviles

2.2.3.2. Locomoción mediante patas

Ésta configuración mantiene el robot lejos de la superficie utilizando solo puntos discretos de soporte. Dependiendo de la base de su estructura, este ofrece mayor o menor soporte para atravesar obstáculos, es por ello que tiene mayor estabilidad a la hora de la navegación sobre entornos dinámicos llenos de obstáculos. El uso de patas como medio de locomoción, permite conseguir una omnidireccionalidad y su deslizamiento al desplazarse es menor.

En este tipo de robots los problemas de control y planificación son más complejos que en los RMR además existe un mayor grado de complejidad en sus mecanismos lo cual advierte un mayor consumo de energía en su traslado. A continuación se puede observar unos ejemplos de este tipo de configuración ver Figura. 11 y Figura. 12

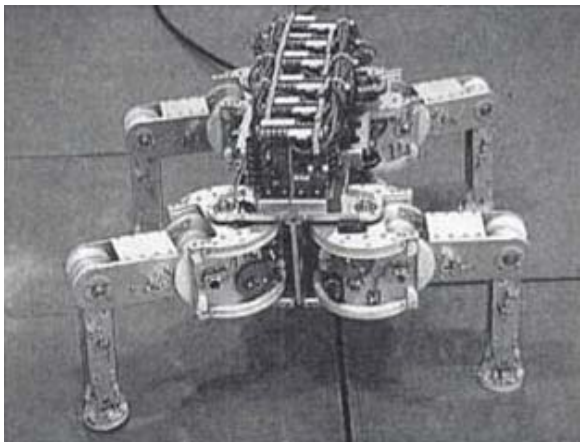


Figura 11. Robot cuadrúpedo
Fuente: Robótica Manipuladores y Robots móviles

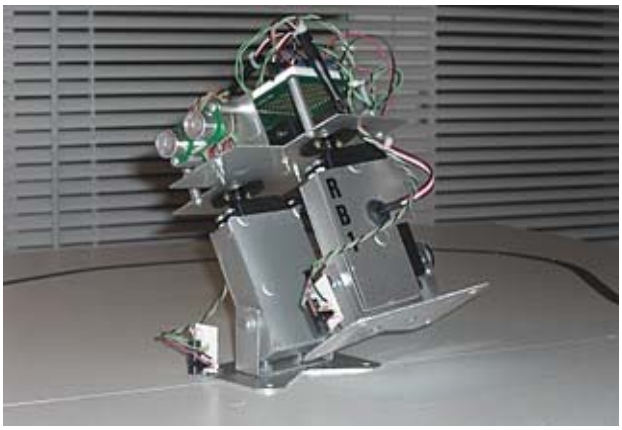


Figura 12. Robot bípedo
Fuente: Robótica Manipuladores y Robots móviles

2.2.3.3. Configuraciones articuladas

Éste tipo de configuración es de gran interés para terrenos difíciles, en la que el cuerpo del robot debe adaptarse a la superficie del terreno. La solución más simple consiste en articular dos o más módulos con locomoción mediante ruedas.

Las configuraciones articuladas con gran cantidad de eslabones (Figura. 13) son apropiadas para caminos estrechos. La seguridad de funcionamiento puede ser mayor debido a la redundancia de su arquitectura, ofreciendo la posibilidad de intercambiar segmentos. El transporte de la estructura también se ve facilitado. Como casos particulares cabe mencionar las estructuras compuestas por segmentos con patas o con ruedas. (Ollero Baturone, 2007)

Figura 13. Robot móvil articulado
Fuente: Robótica Manipuladores y Robots móviles

2.2.3.4. Robots submarinos y aéreos

El interés por la detección, inspección, toma de datos o mantenimiento de instalaciones en lugares inaccesibles para el ser humano debido al acceso total o parcialmente limitado ha motivado el desarrollo de robots aéreos y submarinos para cumplir con estas tareas. Por lo general estos vehículos son el resultado de la evolución de vehículos completamente teleoperados o pilotados por el hombre. En la Figura. 14 se muestra un robot submarino desarrollado por UPC (Universidad Politécnica de Cataluña).

Figura 14. Robot submarino desarrollado por la UPC
Fuente: Robótica Manipuladores y Robots móviles

2.2.4. Actuadores en los RMR

Relativo a los actuadores utilizados para dotar de movimiento a los RMR, es común que se utilicen motores. Existe una gama bastante amplia dependiendo de su empleo, los más utilizados en la robótica móvil son los de corriente directa (DC), por el argumento de que su modelo es lineal, lo que facilita enormemente su control, y específicamente los de imán permanente debido a que el voltaje de control es aplicado al circuito de armadura y el circuito de campo es excitado de manera independiente.

Los motores de DC de imán permanente, son de dos tipos: con escobillas y sin escobillas. Ambos tipos brindan ventajas semejantes, sin embargo, los motores sin escobillas tienen algunas ventajas significativas sobre los motores con escobillas, como por ejemplo:

- Al no contar con escobillas, no se requiere el reemplazo de éstas ni mantenimiento por residuos originados de las mismas.

- No presentan chispas que las escobillas generan, de ésta forma se pueden considerar más seguros en ambientes con vapores o líquidos inflamables.

- La interferencia causada por la conmutación mecánica de las escobillas se minimiza considerablemente mediante una conmutación electrónica.

- Los motores sin escobillas alcanzan velocidades de hasta 50,000 rpm comparadas con las 5,000 rpm aprox. máximas de los motores con escobillas.

A pesar de que estas ventajas parecieran tender la balanza a favor de los motores sin escobillas, existen desventajas cruciales que pueden cambiar la tendencia:

- En los motores sin escobillas no se puede invertir el sentido de giro cambiando la polaridad de sus terminales, esto agrega complejidad y costo a su manejo.

- Los motores sin escobillas son más caros.

- Se requiere un sistema adicional para la conmutación electrónica.

- El controlador de movimiento para un motor sin escobillas es más costoso y complejo que el de su equivalente con escobillas. Al igual que en el arreglo cinemático, cuando se modela un motor de DC se asumen algunas consideraciones, de ésta forma se establece que la única fricción presente es la viscosa, aunque en la práctica se involucran otros tipos de fricción no lineales.

2.3. ARQUITECTURAS COGNITIVAS DE CONTROL

Una arquitectura cognitiva de control se establece como sistemas de software y especificaciones que proporcionan herramientas y lenguajes para el desarrollo de estrategias basadas en comportamientos. Éste diseño abstracto formula un conjunto de componentes estructurales en los cuales se representan la percepción, el razonamiento y la acción; la funcionalidad específica y la interfaz de cada componente; y la topología de interconexión entre componentes (Hayes - Roth, 1992).

Una arquitectura proporciona una manera de organizar un sistema de control por principios. Sin embargo, además de proveer una estructura, ésta impone condiciones acerca de la forma en la cual se debe resolver el problema de control.

Como menciona (Murphy, 2000, pág. 5) en las últimas dos décadas, se ha establecido que en Inteligencia Artificial existen tres primitivas (sensar, planificar,

actuar) en las que comúnmente se acepta que se descompone cualquier acción robótica de alto nivel:

- Un sistema de sensado, faculta al robot para disponer de información sensorial en tiempo real y producir a partir de ésta, información útil para los otros elementos funcionales.

- Un sistema de planificación, instruye al robot para producir las tareas a ejecutar, en base a la información sensorial adquirida y al modelamiento previo del ambiente en el cual se encuentra ejecutando dicha tarea.

- Un sistema de acción, capacita al robot para la ejecución de tareas a través de la interacción con los actuadores.

Actualmente, la organización de la autonomía o inteligencia en robots móviles reconoce tres modelos diferentes:

- Arquitecturas de Control Deliberativas
- Arquitecturas de Control Reactivas
- Arquitecturas de Control Hibridas

Cada una de estas estrategias puede describirse atendiendo a dos factores:

1. La relación existente entre las tres primitivas comunes para cualquier robot: SENSAR, PLANIFICAR, ACTUAR. En el cuadro 1 se describen cada una de estas primitivas en función de los posibles valores de entrada y salida, dependiendo de los valores considerados en cada una de las estrategias se establecen distintas relaciones entre ellas.

Cuadro 1

Definición de las primitivas en términos de E/S

Primitiva	Entrada	Salida	Descripción
SENSAR	Información sensorial "cruda"	Modelo del Entorno	Traduce la información proveniente de los sensores en un modelo del entorno
PLANIFICAR	Modelo del Entorno + Conocimiento previo	Directivas	Genera un plan de ejecución acorde a la misión a cumplir y al modelo del entorno
ACTUAR	Información sensorial y/o directivas	Comandos para los actuadores	Traduce el plan de ejecución en una secuencia apropiada de comandos para ejecutar acciones motrices

2. La forma como la información sensorial es procesada y distribuida a lo largo del sistema. Ésta hace referencia a cuánto o cómo los sentidos influyen en el agente.

A continuación se realiza una descripción detallada de cada una de las estrategias de control en base a estos factores.

2.3.1. Control Deliberativo

La arquitectura de control deliberativa (Nilsson, 1980) es la más antigua, su filosofía predomino principalmente desde 1967 hasta 1990. Según el paradigma deliberativo un robot móvil primero tiene que percibir su entorno, después planear la acción a realizar y finalmente ejecutar dicha acción (S-P-A: sensar-planificar-actuar). De ésta forma, el ciclo tiene que repetirse hasta completar cada una de las tareas del robot como se puede ver en la Figura. 15.

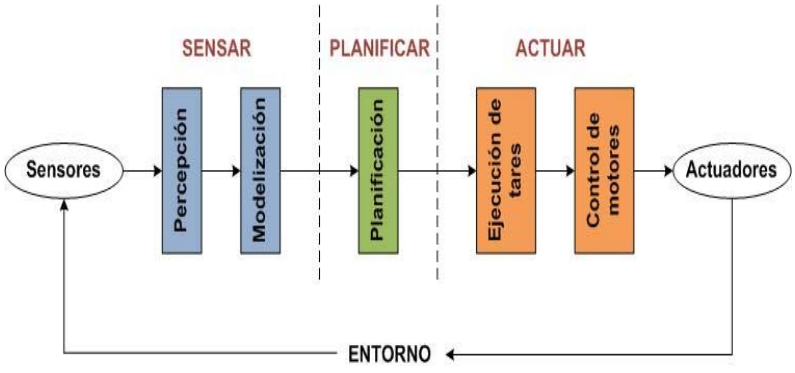


Figura 15. Modelo de una Arquitectura de Control Deliberativa

Éste paradigma hace uso del concepto plan como intención, como medio para seleccionar patrones conductuales que mejor resuelvan las distintas partes de un problema de navegación móvil. Ésta corriente simbólica necesita de representaciones o modelados adecuados del ambiente operativo del robot, sobre los cuales razonar. En algunas ocasiones estas representaciones no son lo suficientemente apropiadas debido al no-determinismo y dinamismo del mundo real.

La principal ventaja de éste tipo de control sobre el elemento reactivo, es que de ser implementado en un agente móvil, se garantiza la determinación de un camino franco a la meta propuesta, siempre y cuando el modelado del entorno sea lo más exacto y completo posible.

2.3.2. Control Reactivo

La arquitectura de control reactiva está infundida en la observación del comportamiento reactivo de los animales, como una reacción frente a los problemas de la arquitectura deliberativa. Presenta una organización Sensar-Actuar (S-A) (ver Figura. 16). Mientras la arquitectura de control deliberativa asume que la entrada que origina una acción es siempre el resultado de un razonamiento o plan, el control reactivo se adjudica que la entrada que origina una acción es siempre la salida directa de un sensor.

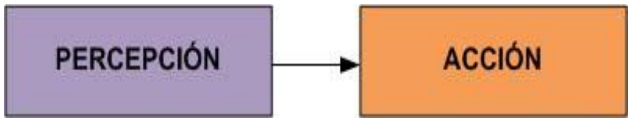


Figura 16. Modelo de una Arquitectura de Control Puramente Reactiva

Según menciona (Brooks, 1986) éste enfoque incorpora información sensorial en tiempo real, sometiendo de ésta manera al agente móvil a ciclos cortos de percepción-acción, lo que permite dotar al robot de la capacidad de producir respuestas inmediatas con acciones motrices ante entornos dinámicos, sin la necesidad de hacer uso de representaciones simbólicas para el modelado total o parcial del medio ambiente operativo del robot, excluyendo así cualquier posibilidad de razonamiento geométrico, temporal y situacional sobre el mundo. Con ésta finalidad, el control se lleva a cabo a través de una colección de reglas condición-acción, que permiten disparar una acción apropiada en la situación definida por la percepción actual del mundo. A pesar de que con ésta nueva estrategia se consiguieron robots que desempeñaban de muy buena manera tareas en entornos dinámicos, los sistemas puramente reactivos no están libres de inconvenientes, su principal desventaja reside en que no dispone de estados internos, lo que reduce su actuación al momento presente, anulando cualquier tipo de anticipación y eliminando la posibilidad de aprender nuevos comportamientos.

Dentro del enfoque reactivo, surgen los sistemas basados en comportamientos liderados por los trabajos de Rodney Brooks (ver Figura. 17).

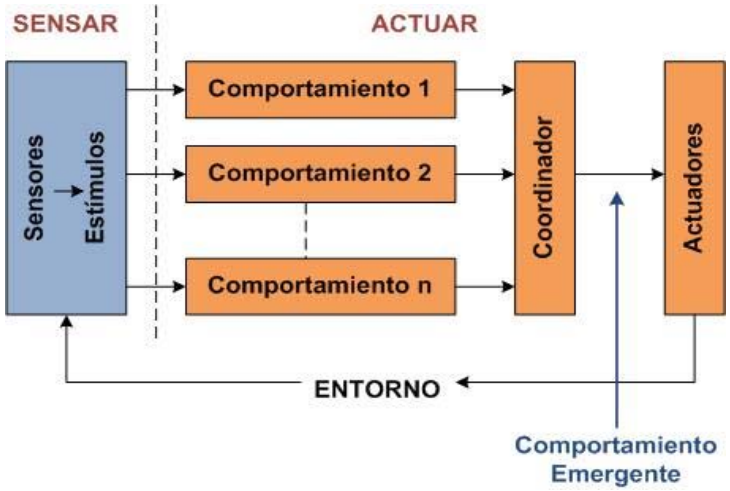


Figura 17. Modelo de una Arquitectura de Control Reactiva basada en comportamientos

Aunque estos sistemas mantienen las ideas principales de los sistemas reactivos, existen diferencias fundamentales entre cada uno de ellos. Los aspectos claves de las arquitecturas basadas en comportamientos, compartidos en su mayoría por el enfoque reactivo, pueden resumirse en los siguientes puntos:

- **El mundo es el mejor modelo de sí mismo:** No existe un modelo interno del mundo, los sistemas basados en comportamientos y en general los reactivos utilizan información sensorial en tiempo real en lugar de un modelado del entorno.

- **Acción situada:** El robot está situado dentro del mundo real, es por ello que debe hacer uso de una percepción del mismo y no debe actuar en función a una representación abstracta de la realidad.

- **Corporeidad:** El robot presenta una presencia física, esto implica que está propenso a sujeciones por parte del mundo real, por lo que sus comportamientos con el ºentorno no pueden ser validados completamente dentro de un ambiente de simulación. El sistema debe desempeñar sus tareas en un ambiente físico.

- **Emergencia:** El comportamiento emergente surge de la interacción del robot y el entorno a través de comportamientos básicos.

2.3.3. Control Hibrido (Reactivo-Deliberativo)

La arquitectura de control hibrida reactiva-deliberativa (Arkin, 1990), combinan ambas metodologías para lograr una operación más robusta en ambientes dinámicos. La idea principal es que el elemento reactivo esté relacionado con la ejecución de conductas, y el deliberativo con la determinación de objetivos y formulación de un plan para poderlos alcanzar (ver Figura. 18).

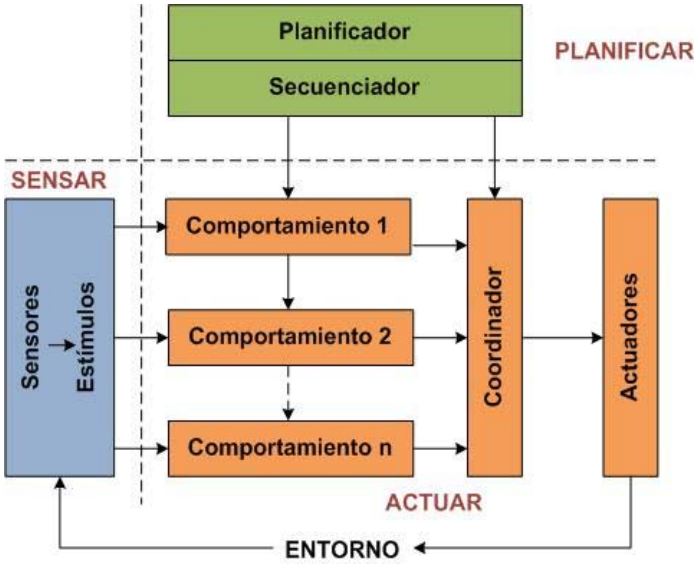


Figura 18. Modelo de una Arquitectura de Control Hibrida

De acuerdo al paradigma hibrido, el robot primero planifica la mejor forma de descomponer la tarea a realizar en varias subtareas, y selecciona los comportamientos adecuados para alcanzar cada subobjetivo. Posterior a ello el robot empieza con la ejecución del paradigma reactivo tal y como se describió anteriormente.

Éste tipo de estrategia puede denotarse a través del ciclo “Planear, Sensar-Actuar” [P, S-A], donde la como indica que en una primera parte se realiza la planificación y posteriormente se ejecutan los patrones de comportamientos (Sentir-Actuar).

La organización y procesamiento de la información sensorial dentro del paradigma hibrido también es una mezcla de los paradigmas deliberativo y reactivo. Ésta información es utilizada a nivel de planificación, para la construcción de un modelo del entorno que se utiliza para la descomposición de tareas y la selección de comportamientos.

Dependiendo del nivel en el que se procese la información, los tiempos requeridos son distintos. En el nivel de ejecución se necesita que la información se procese en tiempo real, debido a que el robot necesita ejecutar constantemente ciclos cortos de percepción-acción, mientras que en el nivel de planificación donde las acciones a realizar requieren de un mayor coste temporal, no se precisa de tiempos de respuesta rápidos, aunque si es necesario tener un conocimiento previo de la información, para constatar la validez de la información en cualquier momento o generar un modelado del entorno.

2.4. NAVEGACIÓN ROBÓTICA

La navegación orientada a objetivos es sin duda alguna uno de los problemas fundamentales a ser resuelto para dotar al robot autonomía total o parcial. La navegación es por lo tanto una habilidad de la que todo robot debe estar dotado. Para que un robot pueda navegar libremente debe mantener información actual de su posición o estado, además de información sobre el objetivo y la relación entre el estado actual y ese objetivo.

Tanto en los sistemas deliberativos como en sus sucesores la navegación se define como la combinación de tres tareas fundamentales:

- *Mapeo:* El robot debe poseer una representación del mundo real en el cual se está desempeñando.

- *Localización:* El agente debe poseer mecanismos que le ayuden a estimar su posición dentro del entorno.

- *Planificación:* El robot debe de estar en la capacidad de decidir qué acción ejecutar para la consecución del objetivo planteado.

La navegación se fundamenta mediante un modelo centralizado del mundo que se le puede proporcionar o que el mismo agente puede aprender. Ese modelo centralizado se denomina mapa. Los mapas, métricos o topológicos, han evolucionado hacia modelos probabilísticos, que constituyen hoy en día la técnica dominante a la hora de representar el entorno en éste tipo de sistemas.

Por otro lado, en el contexto de los sistemas basados en el comportamiento la navegación tiene una clara inspiración biológica, y se basa en comportamientos que incluyen desde la simple tarea de deambular hasta el reconocimiento de los objetivos.

2.4.1. Mapeo

El desarrollo adecuado de un modelo del entorno es indispensable para la realización de las tares de navegación y manipulación. Un modelo apropiado del entorno permite al robot conocer su posición actual y establecer un plan orientado hacia el objetivo propuesto. Tres son los enfoques que permiten solucionar el problema planteado:

1) Enfoque Geométrico consiste en el uso de **Modelos 3D** complejos o en **Mapas de celdas** para la representación del entorno.

Partiendo de que los últimos son los más usados dentro de la generación de mapas. Estos realizan un modelo del entorno como una matriz de celdas, cada una de ellas como representación de una región plana del espacio. Estas celdas llevan asociadas un valor de ocupación para poder determinar si en dicho lugar del espacio existe o no un obstáculo. Ésta información es administrada por el robot para la planificación de trayectorias.

En la siguiente Figura.19 se muestra un ejemplo de mapa de entorno utilizando la técnica de descomposición exacta en celdas.

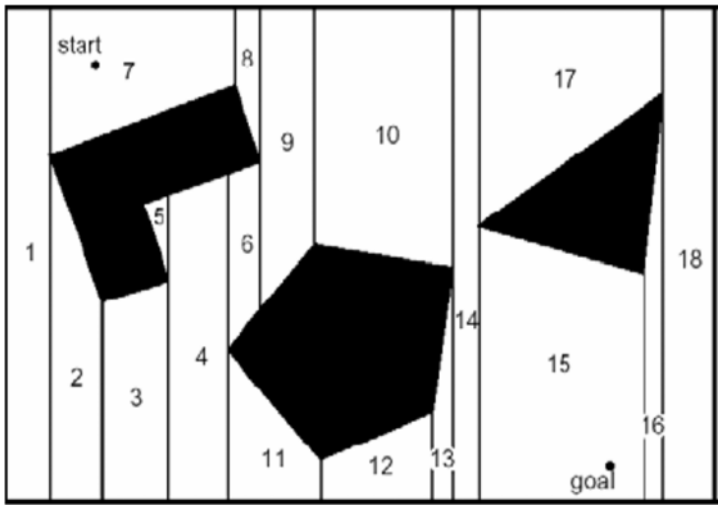


Figura 19. Descomposición exacta en celda de un escenario
Fuente: Robot Moting Planning

En éste tipo de mapas, la posición exacta del robot dentro de cada una de las celdas no es significativa, importando únicamente la destreza del robot para poder transitar entre celdas adyacentes. En escenarios complejos ésta aproximación no siempre es factible realizarla pues requiere de una elevada capacidad para determinar las celdas y por consiguiente, el desarrollo del mapa. Por ello, existen otras alternativas, como la Descomposición fija, en la que el entorno se divide en una serie discreta de áreas del mismo tamaño. En éste modelado, cada una de las áreas aparece ocupada o vacía, dependiendo de si al momento de realizar la

descomposición discreta existe algún obstáculo o no. En la Figura. 20 y 21 podemos observar la descomposición original en celdas y aplicando el método fijo respectivamente.

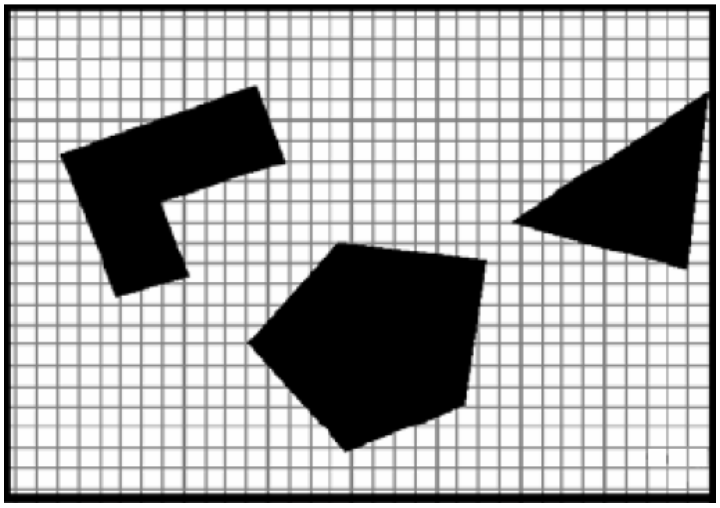


Figura 20. Descomposición original en celdas
Fuente: Robot Moting Planning

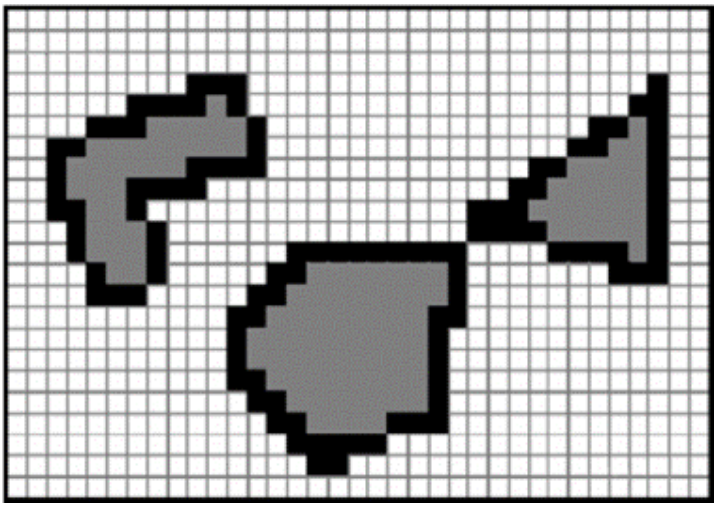


Figura 21. Descomposición fija
Fuente: Robot Moting Planning

Como podemos observar en la Figura. 20, existe una pequeña distancia de separación entre el triángulo y el pentágono, pero debido a la cercanía de las figuras, el método de la descomposición fija la muestra como inexistente, ilustrando de ésta forma la principal desventaja de éste método al no poder representar con exactitud un modelo adecuado del entorno en el cual el agente se encuentra desempeñando una determinada tarea.

Un método para corregir el no modelamiento de distancias cortas entre objetos es la denominada Descomposición Adaptativa, en la que el tamaño de las áreas es variable. Para poder obtener las áreas, inicialmente, se delimita a todo el entorno en una única celda, posteriormente se divide la celda en cuatro celdas iguales. En cada una de las celdas en las cuales se encuentre un obstáculo se vuelve a efectuar una nueva división en cuatro partes iguales, manteniendo intactas las celdas que no estén ocupadas. En las celdas generadas se verifica si en alguna de ellas existen obstáculos y se proceda a realizar una nueva división. Éste proceso se mantiene hasta que ya no se generen nuevas celdas. En la Figura. 22 se puede observar una ilustración de éste método.

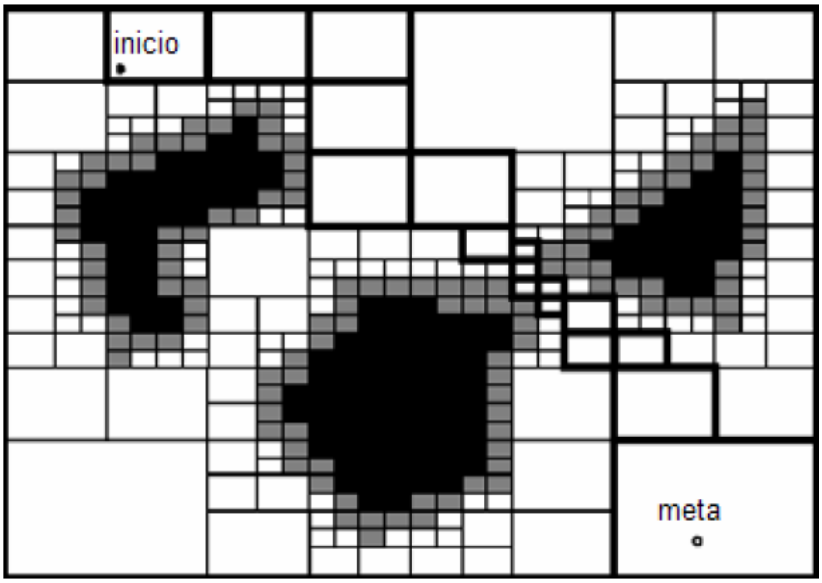


Figura 22. Descomposición Adaptativa

Fuente: Robot Moting Planning

En la actualidad el método más utilizado en la generación de mapas, es el denominado Rejilla de Ocupación. Éste método posee una trama de celdas que inicialmente están desocupadas. Cada una de las celdas dispone de un contador que aumenta conforme los elementos que componen el sistema sensorial detectan un obstáculo. Cuando el contador de una celda supera un límite determinado, ésta se activa como ocupada y se representa con un color más oscuro. El beneficio de éste modelamiento recae en la propiedad de identificar obstáculos temporales, ya que conforme pasa el tiempo el contador de las celdas empieza a disminuir, con lo que es posible que una celda marcada como ocupada volviese a estar como vacía. En la Figura. 23 se puede observar el modelamiento de un entorno ocupando ésta técnica de descomposición.

Figura 23. Rejilla de Ocupación
Fuente: Robot Moting Planning

2) Enfoque Topológico consiste en reconocer diversos elementos del entorno y establecer la relación existente entre ellos. En estos casos, los mapas representan al entorno con un mayor grado de abstracción que los enfoques geométricos. Dentro de éste tipo de enfoque el entorno se encuentra representado como un grafo de conectividad en el cual se genera un mapa topológico a partir de una división en regiones de un mapa de celdas. Cada nodo del grafo representa lugares destacados

en base a los sistemas sensoriales, mientras que los arcos definen el camino entre los nodos.

Éste tipo de mapeo también puede establecer la propia forma de navegar entre nodos. En la Figura. 24 se puede observar un ejemplo de mapa topológico.

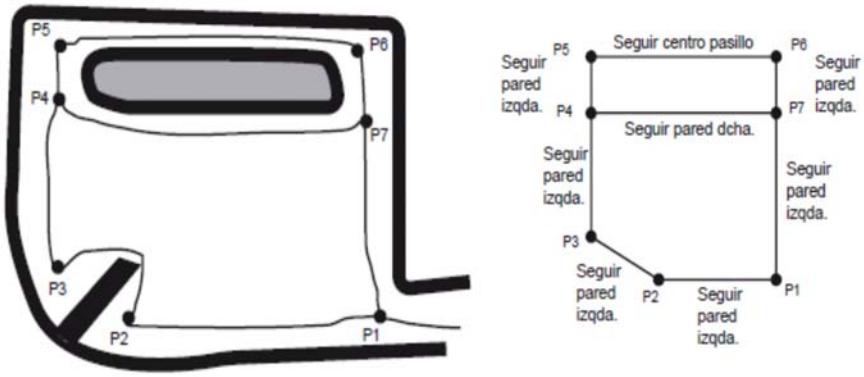


Figura 24. Mapa Topológico
Fuente: Robot Moting Planning

En los mapas topológicos las áreas representadas por los nodos no tiene por qué ser del mismo tamaño. Sin embrago es necesario establecer una marca topológica que pueda ser identificada por los sensores y permita al robot reconocer el nodo en el que se encuentra. Es por ello que para que el robot pueda navegar utilizando un mapa topológico, deben cumplirse dos características importantes: el robot debe estar en la capacidad de poder identificar el nodo en el que se encuentra y debe poseer un mecanismo que permita desplazarse efectivamente entre nodos.

2.4.2. Localización

El conocimiento de la posición dentro de un entorno de trabajo, representa para un robot móvil una de las principales problemáticas al momento de desarrollar diversas tareas de navegación.

La precisión requerida en la localización depende del tipo de aplicación y arquitectura de control empleada. Por ejemplo en los enfoques geométricos de representación del entorno, se requiere el uso de mecanismos precisos de localización, mientras que en los enfoques topológicos se hace uso de posiciones aproximadas.

En el caso de que la posición inicial del robot en el mapa sea conocida, existe la posibilidad de conocer su nueva posición tras realizar un desplazamiento, mediante el uso de la Odometría. Éste proceso consiste en contener una serie de sensores propioceptivos, es decir sensores que toman datos internos al robot, y que informen de los movimientos realizados. Estos datos son pocos fiables debido a los errores frecuentes y acumulables que se producen en los sensores y actuadores, lo que provoca que a medida que el robot se mueve, se vayan acumulando, provocando grandes errores de posicionamiento. A pesar de estas limitaciones, se piensa en éste método como un aparte importante de los sistemas navegación.

La odometría está basada en ecuaciones simples consistentes en hacer lineal el giro de la rueda, midiendo de ésta manera la distancia lineal recorrida por la rueda y la superficie en contacto.

En lo que respecta al conocimiento del robot de su posición dentro del mapa, existen diversas posibilidades. Según (Siegwart & Nourbakhsh, 2004, pág. 194) “la primera de ellas es la denominada Hipótesis única, implica que el robot asume la opinión de que está situado en una posición determinada del mapa, con cierto grado de precisión”.

La principal ventaja de éste método de localización proviene del hecho de que únicamente existe una hipótesis referente al posicionamiento del robot. Facilitando en gran medida la toma de decisiones, pues únicamente se tendrá en cuenta una posición como punto de partida para la siguiente acción a realizar, resulta sencillo

identificar la nueva posición del robot al realizar un desplazamiento, ya que únicamente se va a referir una única nueva posición.

En la Figura. 25 y 26 se muestran ejemplos de localización en distintos mapas utilizando la estrategia de hipótesis única.

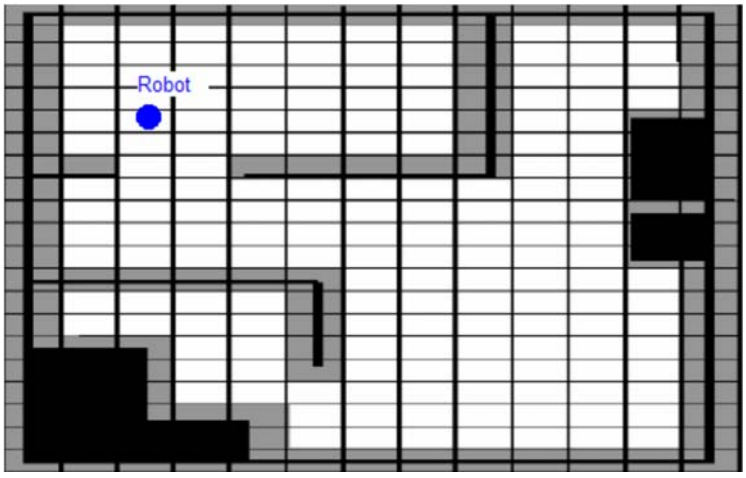


Figura 25. Localización en mapa de celdas
Fuente: Robot Moting Planning

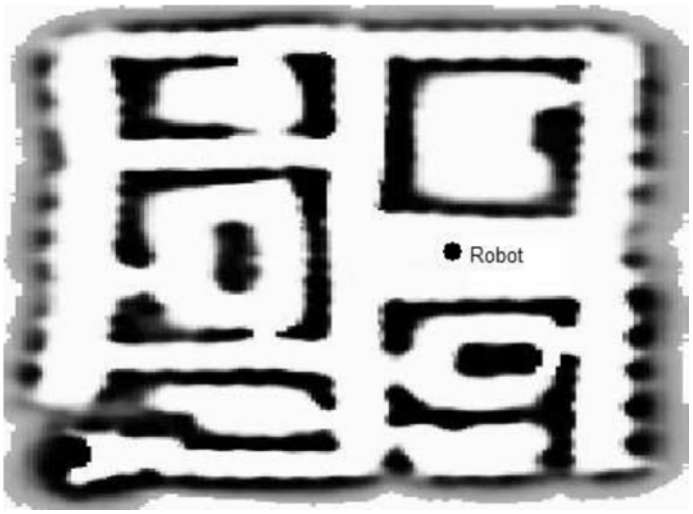


Figura 26. Localización en rejillas de ocupación
Fuente: Robot Moting Planning

Sin embargo existe la denominada Creencia en múltiples hipótesis, a través de la cual el robot tiene diferentes percepciones acerca de su posicionamiento actual. Una representación de éste tipo de creencia en un mapa es la propuesta por el investigador Jean-Claude Latombe y representa el conjunto de posibles posiciones del robot como un polígono convexo en un mapa bidimensional del entorno. Según menciona (Latombe, 1991) ésta representación determina que todas las posiciones contenidas dentro del polígono, son posibles posiciones del robot dentro del entorno de trabajo. De ésta manera, el posible conjunto de posiciones se representa geométricamente dentro del modelado del entorno, sin tener ninguna ponderación en especial, es decir que el robot asigna el mismo grado de probabilidad a todas y cada una de las posibles posiciones.

En lo referente a la posterior fase de planificación, el inconveniente de poseer diferentes hipótesis, consiste en el aumento de la complejidad a la hora de tomar decisiones con respecto a la siguiente acción a realizar, pues existe cierta ambigüedad con respecto a la posición inicial desde la que se inicia el movimiento. Para suplir ésta desventaja, existen diversas alternativas que varían desde delimitar el número de hipótesis posibles que pueden almacenarse, hasta aplicar distintos métodos probabilísticos que me permitan simplificar el proceso de actualización de la hipótesis.

A continuación se describen dos métodos probabilísticos utilizados frecuentemente en procesos de localización, para que estos métodos puedan establecer la posición exacta del robot deben poseer un modelado previo del entorno.

a) Localización de Markov

Éste método basa su funcionamiento en una representación probabilística de las posibles posiciones del robot dentro del entorno. Ya en la práctica éste método necesita de un conjunto discreto de posibles posiciones antes de ser empleado,

dicho conjunto está conformando por miles y millones de posiciones, por lo que se requiere de un mecanismo que me permita actualizar éste conjunto cada vez que el robot realiza un desplazamiento. Ésta actualización consiste en eliminar o agregar nuevas posiciones al conjunto, si así fuera necesario y actualizar la probabilidad marginal $p(X)$, asociada a cada una de las posiciones, en base a la toma de datos por parte de los sensores. Cada una de estas posiciones se representa en un mapa a través de sus coordenadas cartesianas (x, y), y su rotación (r) con respecto a uno de los ejes del robot. Así púes la localización de un punto (l) en el espacio (L) estará dada por la siguiente ecuación:

$$l = (x, y, r)$$

En un determinado instante de tiempo (t), el reconocimiento del robot sobre cada una de sus posibles soluciones está determinado por una función de probabilidad $Bel(L_t)$, mientras que la representación de la probabilidad asociada a cada una de las hipótesis se denota $Bel(l_t)$.

Dependiendo de si la actualización del conjunto de posibles posiciones se realiza por un desplazamiento o por la información tomada por los sensores, se aplicará el modelo de movimiento o el modelo de percepción respectivamente.

- **Modelo de Movimiento** se emplea cada vez que el robot efectúa un desplazamiento (rotación y/o traslación) dentro del entorno, dicho desplazamiento se percibe a través de los sistemas de odometría. Para calcular la distribución de probabilidad $Bel(l')$ para cada una de las hipótesis $Bel(l)$, es decir la probabilidad de que el robot se encuentre en la posición l' después de efectuar un desplazamiento d desde la posible posición l, se denota $p(l'|l, d)$, siendo necesario integrar ambas probabilidades dentro del entorno L.

$$Bel(l') = \int_L p(l'|l, d) * Bel(l) * dl$$

- **Modelo de percepción** se utiliza cada vez que varía la información obtenida por los sensores. Este modelo utiliza la función de probabilidad $p(s|l)$ que indica la posibilidad de tomar lecturas (s) desde una posición determinada (l). Ésta probabilidad se obtiene del análisis del mapa del entorno. Al emplear este método se actualizan las distribuciones de probabilidad asociadas a cada una de las hipótesis de la siguiente manera:

$$Bel(l_t) = p(s|l) * Bel(l_{t-1})$$

b) Filtro de Kalman

Es una técnica que emplea que emplea fusión sensorial. Es decir para que este método pueda aplicarse en problemas de localización requiere que estos sean representados como problemas de fusión sensorial. Este método dispone de varias fases, en las cuales se obtiene información de cada uno de los sensores, elaborando un conjunto de predicciones, y finalizando con un proceso de empate entre los distintos tipos de predicciones realizadas.

En una fase preliminar, este método requiere de una estimación inicial de la posición del robot, en base a su estado de reposo (antes de realizar ningún movimiento) y los valores odométricos obtenidos después de ejecutar un determinado desplazamiento. Es así que para poder determinar con exactitud la posición luego del primer desplazamiento, es necesario conocer con un grado de certeza absoluto, el punto de reposo o de partida del robot. Posterior a realizar la predicción inicial de la posición, se obtiene diversa información del entorno a través de los sensores, lo que permite realizar nuevas predicciones del posicionamiento del robot en el mapa. En último lugar, se dispone de conjuntos que representan las posibles localizaciones, por lo que se realiza un proceso de fusión sensorial para intentar disminuir el grado de incertidumbre de la predicción, producto de los errores acumulados en las distintas mediciones, a través de la selección de características comunes a los distintos conjuntos de predicción. Para

disminuir dicho grado de incertidumbre, se asume que el error producido en las distintas mediciones puede evaluarse mediante una distribución gaussiana, permitiendo obtener excelentes resultados en la mayoría de los casos.

La relación que existe entre la posición y los datos de los distintos sensores tiene un carácter lineal, limitando el uso de este método únicamente a sistemas lineales. Sin embargo, se han desarrollado distintas modificaciones, como el Filtro de Kalman Extendido (EKF), cuya principal aportación respecto al del Filtro de Kalman convencional es su extensión a sistemas no lineales. También permite incorporar los casos en los que la relación entre el estado y las medidas de los sensores externos no puede definirse de forma explícita. La convergencia del EKF depende de diversos factores como pueden ser la estimación inicial, las no linealidades de las ecuaciones, el orden en que se procesan las medidas. De este modo, no existe prueba formal de la convergencia del algoritmo, sino tan solo ciertos tests de consistencia que evalúan el comportamiento del mismo en cada caso.

2.4.3. Mapeo y Localización Simultáneo (SLAM)

Dentro de las muchas tareas que tiene que cumplir el robot, estos se ven enfrentados, a ambientes no determinísticos y dinámicos, en las que las recientes aplicaciones de navegación e interacción entre hombre-robot, requiere tener bajo control el desplazamiento dentro del escenario, evitando a toda costa el contacto directo del robot con obstáculos estáticos o dinámicos. Por esta razón es que a lo largo de los años se ha desarrollado un mecanismo que permite realizar un mapeo y localización simultaneo del entorno, conocido como SLAM (Simultaneous Localization and Maping).

El desplazamiento de los robots en escenarios dinámicos y complejos, presenta tres inconvenientes en lo que a estrategias de mapeo y localización se refiere.

- Reconstrucción de la parte estática de la escena.

- Conocer la ubicación del robot dentro de la escena y en relación a los objetos presentes en ella.

- Detectar los objetos en movimiento, identificarlos, describir su posición y velocidad, para una estimación en tiempo futuro.

La técnica del SLAM, hace uso de distintos tipos de sensores, tales como: sensores de contacto, infrarrojos, ultrasónicos, encoders, etc. Sin embargo, el uso de cámaras de video, en especial cámaras estereoscópicas, han permito al robot obtener información adicional que facilita el cálculo de la profundidad. Los datos obtenidos de las cámaras son tratados con diferentes técnicas de procesamiento digital de imágenes, con el propósito de extraer características del entorno y simplificar el problema de navegación.

El método del SLAM no es un algoritmo de navegación, sino que consiste en un conjunto de subtareas que pueden ser realizadas de distinta manera. Es pues ésta estrategia, una combinación bien balanceada de técnicas de mapeo y localización como las descritas anteriormente. En un primer plano, el robot debe reconocer ciertos objetos dentro del ambiente que le permitan establecer marcas posicionales dentro de este. Dichas marcas pueden ya estar modeladas previamente, lo que faculta al robot a únicamente identificarlas, o ser seleccionadas aleatoriamente. Una vez reconocidas las marcas, el robot se encarga de añadirlas en el interior del mapa que se está elaborando, y actualizar su ubicación dentro de éste.

Posterior a la realización del primer desplazamiento, el robot ejecuta una nueva actualización de su posición dentro del mapa, en base a datos odométricos obtenidos. Estos datos involucran un porcentaje de error en la medición, que puede acumularse a lo largo de distintas iteraciones, provocando que la representación

del mapa y la ubicación del robot dentro de éste, no concuerden con el mundo real, por lo que el robot, después de realizar un proceso de actualización de su ubicación, vuelve a detectar las marcas de referencia identificadas con anterioridad, utilizando diferentes tipos de sensores. Todos los datos entregados por los sensores conllevan, a su vez, cierto grado de error en la medición, por lo que resulta necesario realizar un proceso de fusión sensorial, con el propósito de conseguir una aproximación más real de la posición de las marcas observadas.

Tras adquirir la nueva posición del robot y las marcas de terreno, se actualiza tanto el mapa interno que el robot elabora como su posición en el mismo, no siendo así necesario ningún tipo de información previa al inicio de la navegación.

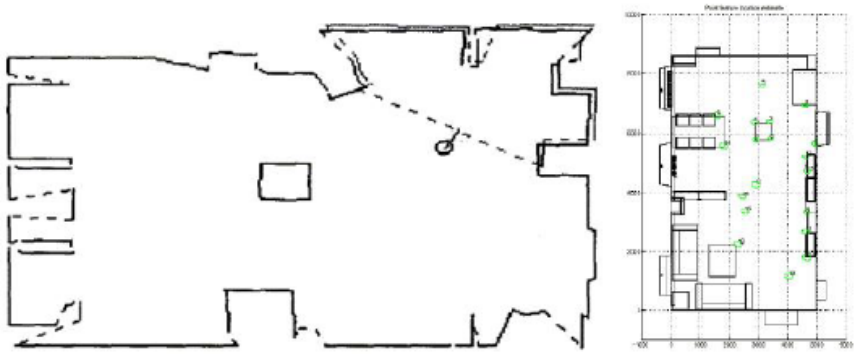


Figura 27. Mapa generado mediante SLAM
Fuente: Robot Moting Planning

2.4.4. Planificación

Posterior al modelamiento del entorno de trabajo, y al desarrollo de mecanismos de localización, que permitan conocer la posición del robot tras un desplazamiento, es necesario establecer una estrategia que me permita dotar al robot de habilidades cognitivas. Dado una posición inicial y un objetivo, planificar consiste en definir una secuencia de movimientos por parte de los actuadores para poder llegar al objetivo deseado en una trayectoria libre de colisiones. Para ello existen distintas alternativas, que me permiten generar un plan de acción, que

garantice en buena medida la consecución de las metas. Para ello, el mecanismo se divide en dos tareas:

- **Planificación global:** Construye o planifica la ruta que permite al robot desplazarse a cada una de las submetas planteadas por el control de la misión, para conseguir una aproximación al camino final seguido, sin tener en cuenta posibles obstáculos.

- **Planificación Local:** Resuelve las obstrucciones presentes sobre la ruta global, en el entorno local al robot, para determinar la ruta real que será seguida. La representación del entorno local se construye mediante la fusión sensorial de los datos entregados por los sensores.

La planificación global puede realizarse antes de que el agente comience a ejecutar una determina tarea, mientras que la planificación local se ejecuta en tiempo real a la navegación. Si un robot navega dentro de un ambiente determinísticos, es totalmente innecesario realizar una planificación local, pero a medida de que se tiene un entorno dinámico aumenta la importancia de la misma.

A continuación se establecen las técnicas de planificación más importantes:

a) Grafo de Visibilidad

Esta técnica de planificación consiste en encontrar el camino más corto entre la posición inicial (x_i, y_i, r_i) del robot y su objetivo o meta $(x_f, y_{f,} r_f)$. Para ello y dado un mapa geométrico del entorno, este método se encarga de unir a estos dos puntos mediante un segmento recto que no cruce ningún obstáculo en el camino. El grafo por lo tanto es la unión de todos aquellos nodos que son visibles. La trayectoria que une el punto de origen con el punto de destino se conseguirá implementando un algoritmo de búsqueda de grafos, por lo que la salida consistirá

en una secuencia de segmentos entre puntos visibles. En la Figura. 28 se ilustra el funcionamiento de este método.

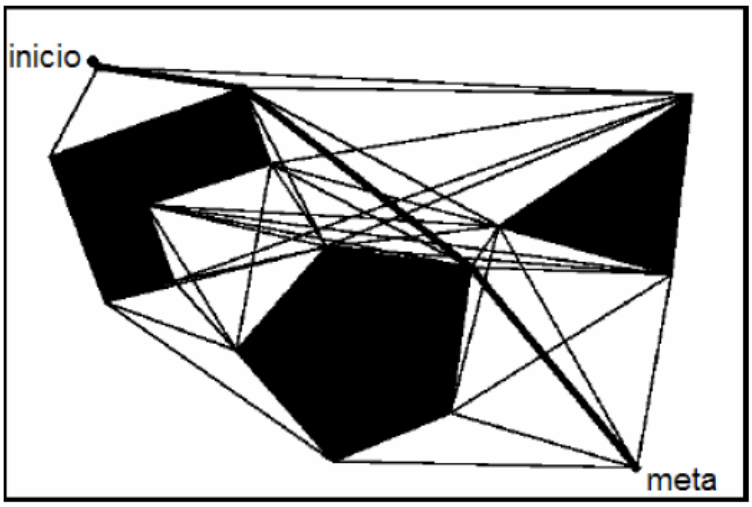


Figura 28. Grafo de visibilidad
Fuente: Robot Moting Planning

En la figura representada, se observan todos los caminos que forman la red, así como el camino más corto que será el seguido por el robot, en líneas más gruesas. Podemos comprobar también que dicho camino, siempre que evada un obstáculo transitara por un espacio muy próximo a éste.

b) Diagrama de Voronoi

Geométricamente, el diagrama de Voronoi es el lugar geométrico de las configuraciones que se encuentran a igual distancia de los dos obstáculos más próximos del entorno, por lo que esta técnica sitúa la trayectoria del robot lo más alejado posible de los dos obstáculos. En la Figura. 29 se ilustra el funcionamiento de esta técnica.

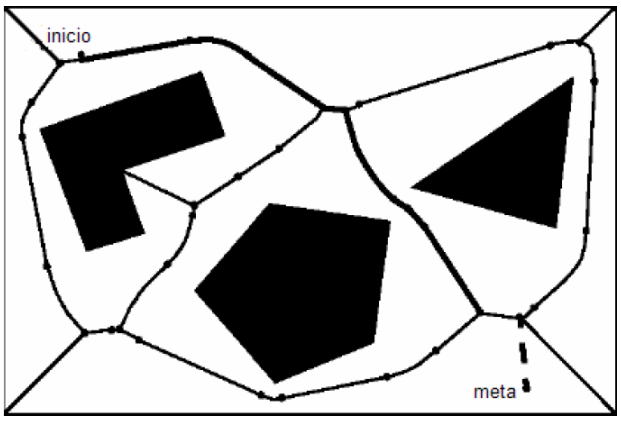


Figura 29. Diagrama de Voronoi
Fuente: Robot Moting Planning

Como podemos observar, el camino recorrido no es el óptimo en términos de distancia, lo que representa la principal desventaja de este algoritmo. Por otro lado, debido a que es posible que la distancia a los obstáculos sea relativamente grande, cabe la posibilidad de que el camino elegido circule tan alejado de los obstáculos, y otros elementos del entorno, que no sea posible percibirlos a través de los sensores, hecho que dificultaría las tareas de localización.

CAPITULO 3

HERRAMIENTAS

3.1. HARDWARE

3.1.1. iRobot Create

iRobot Create es una plataforma robótica móvil pre-ensamblada (ver Figura. 30), que presenta una gran oportunidad para que estudiantes, desarrolladores e investigadores puedan desarrollar programas propios, sonidos, movimientos o incluso añadir algún componente electrónico sin tener que crear un robot móvil desde cero y así hacer una diferencia en el avance de la investigación de la robótica.

Figura 30. IRobot Create
Fuente: Manual del propietario - iRobot

El iRobot Create es uno de los modelos fabricados por iRobot Corporation el cual fue fundado en 1990 por ingenieros del prestigioso Instituto de Tecnología de Massachusetts (MIT), iRobot Corporation está especializada en robots que llevan a cabo tareas de difícil realización y muchas veces de gran peligro para las personas, proporcionando además un gran nivel de eficacia. Movidos por el exclusivo sistema de inteligencia robótica AWARE1, los robots de la compañía pueden navegar en situaciones reales, complejas y dinámicas.

3.1.1.1. Descripción General

iRobot Create es un completo kit de desarrollo robótico que permite programar nuevos comportamientos del robot sin tener que preocuparse de su estructura mecánica y el código que el robot utilice. La OI (Open Interfaz) del iRobot Create provee al usuario de un conjunto de comandos, tales como el manejador de comandos, el demo de comandos, comando de canciones y comando de sensores, que incentiva al programador a desarrollar sus propias aplicaciones.

El iRobot cuenta con 10 demos útiles, como fuente de inspiración para el desarrollo de aplicaciones. Al mismo tiempo que permite el desarrollo de scripts (secuencia de comandos incluyendo los que esperan por eventos). Con un script se puede hacer actuar al robot mediante una rutina, como por ejemplo de "canto y baile".

A esta plataforma móvil se le pueden incorporar y controlar otros hardwares o dispositivos electrónicos, como un brazo robótico, display de luces o un sensor de alcance, etc. El iRobot Create también puede convertirse en un instrumento musical, al utilizar el comando de canciones, se puede escribir y guardar hasta 16 canciones; todas estas aplicaciones mencionadas anteriormente son gracias a la robustez y flexibilidad que la plataforma exhibe. A continuación se presenta una

[1] AWARE es un software comercial de quinta generación que utiliza la industria, dispone de un comprobado lenguaje y tecnologías de código abierto, proporcionando una arquitectura flexible y abierta para el desarrollo a terceros. Técnicamente sofisticado, AWARE está diseñado para resolver las necesidades de desarrollo robóticas del hoy y mañana.

breve descripción de las características más relevantes que posee el robot móvil iRobot Create modelo 4400.

I. Anatomía

Vista desde arriba

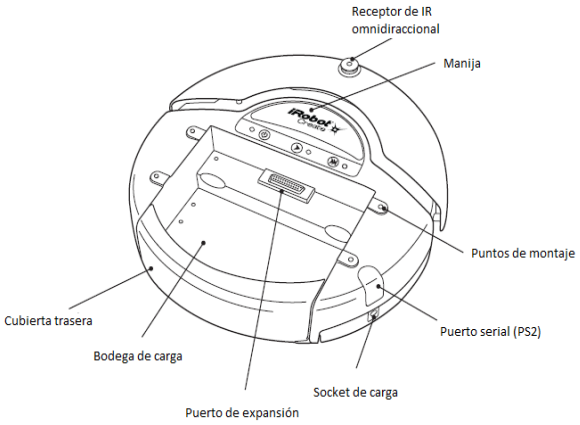


Figura 31. Vista superior del iRobot Create
Fuente: Guía del propietario - iRobot

Vista desde abajo

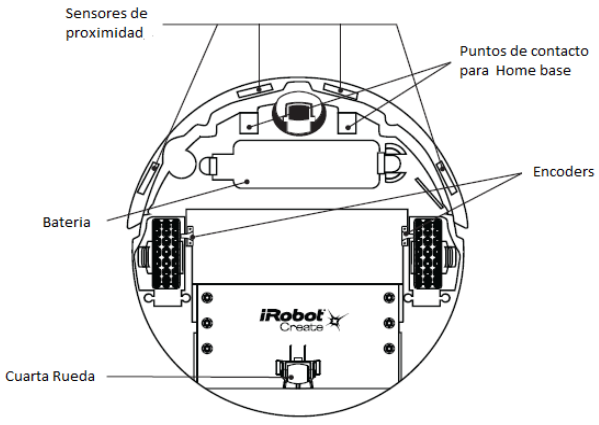


Figura 32. Vista inferior del iRobot Create
Fuente: Guía del propietario - iRobot

Botones y Luces

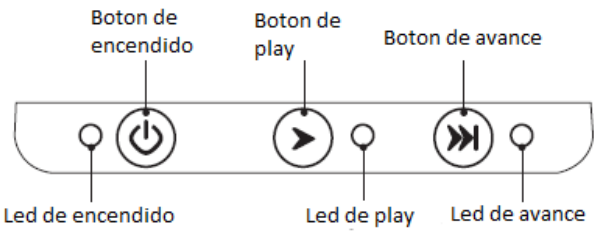


Figura 33. Panel de operación
Fuente: Guía del propietario - iRobot

II. Conexiones Físicas

La OI del Create consta de una interfaz electrónica y una interfaz software, para el control de comportamientos y lectura de sensores del Create. La interfaz electrónica incluye un conector Mini-DIN de 7 pines y un conector DB-25 ubicado en el puerto de expansión para conectar hardware o dispositivos electrónicos como sensores y actuadores. La interfaz software permite manipular comportamientos en el Create y tomar datos de los sensores mediante una serie de comandos que se envían al puerto serie del Create a través de una PC o un microcontrolador que son conectados al puerto PS2 o al Cargo Bay Connector respectivamente. Estos conectores proporcionan dos formas de comunicación serial a niveles de voltaje TTL (0 - 5V).

Mini- DIN Connector

En la Figura. 34 se observa la vista superior de los pines del conector tipo hembra en el Create y en el cuadro 2 se puede encuentra la descripción de los pines del Mini-DIN.

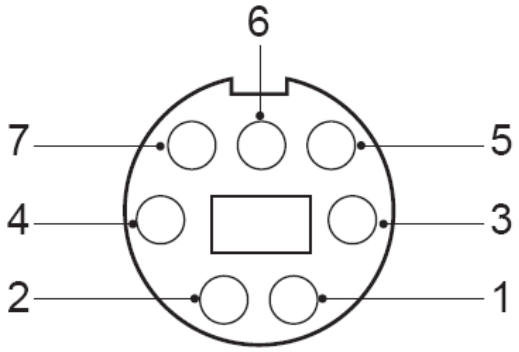


Figura 34. Vista superior Del Conector Mini – DIN

Fuente: Interfaz Abierta - iRobot

Cuadro 2

Descripción de pines del Mini – DIN

Pin	Nombre	Descripción
1	Vpwr	Create battery + (unregulated)
2	Vpwr	Create battery + (unregulated)
3	RXD	0 – 5V Serial input to Create
4	TXD	0 – 5V Serial output from Create
5	BRC	Baud Rate Change
6	GND	Create battery ground
7	GND	Create battery ground

Fuente: Interfaz Abierta - iRobot

Como se observa en el Cuadro. 2 los pines RXD y TXD del Mini-DIN operan con niveles de voltaje lógicos de 0 – 5V, mientras que el puerto serie del computador trabaja con niveles de voltaje de acuerdo a la norma RS-232, por lo que es necesario cambiar los niveles de tensión, para ello se debe utilizar cable serie del Create en lugar de un cable serie normal, ya que el cable serie del Create contiene todo el hardware necesario para cambiar los niveles de tensión, mientras que el cable serie normal no.

Cargo Bay Connector

El cargo bay connector está ubicado en la parte media de la bodega de carga, contiene 25 pines etiquetados que pueden usarse para fijar dispositivos electrónicos y otros dispositivos periféricos tales como sensores adicionales. El mismo proporciona cuatro entradas digitales, una entrada analógica, tres salidas digitales, tres "low-side driver outputs" (útiles para el manejo de los motores), un indicador de carga, un conmutador de alimentación de entrada, transmisor y receptor serie, una referencia de 5V, tierra de la batería y el voltaje de la batería (Ver Figura. 35 y cuadro 3)

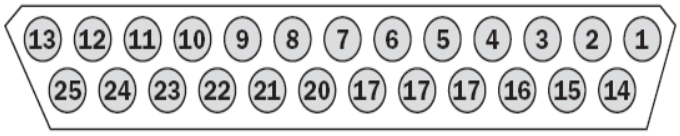


Figura 35. Conector DB – 25

Fuente: Interfaz Abierta - iRobot

Cuadro 3

Descripción de pines del conector DB-25

Pin	Nombre	Descripción
1	RXD	0 – 5V Serial input to Create
2	TXD	0 – 5V Serial output from Create
3	Power control toggle	Turns Create on or off on a low-to-high transition
4	Analog input	0 - 5V analog input to Create
5	Digital input 1	0 - 5V digital input to Create
6	Digital input 3	0 - 5V digital input to Create
7	Digital output 1	0 - 5V, 20 mA digital output from Create
8	Switched 5V	Provides a regulated 5V 100 mA supply and analog reference voltage when Create is switched on

CONTINUA ⟶

9	**Vpwr**	**Create battery voltage (unregulated), 0,5A**
10	Switched Vpwr	Provides battery power @ 1,5 A when Create is powered on.
11	Switched Vpwr	Provides battery power @ 1,5 A when Create is powered on.
12	Switched Vpwr	Provides battery power @ 1,5 A when Create is powered on.
13	Robot charging	When Create is charging, this pin is high (5V)
14	**GND**	**Create battery ground**
15	Device Detect/Baud Rate Change Pin	0-5V digital input to Create which can also be used to change the baud rate to 19200 (see below)
16	GND	Create battery ground
17	Digital input 0	0 - 5V digital input to Create
18	Digital input 2	0 - 5V digital input to Create
19	Digital output 0	0 - 5V, 20 mA digital output from Create
20	Digital output 2	0 - 5V, 20 mA digital output from Create
21	GND	Create battery ground
22	Low side driver 0	0,5A low side driver from Create
23	Low side driver 1	0,5A low side driver from Create
24	Low side driver 2	1,5A low side driver from Create
25	GND	Create battery ground

Fuente: Interfaz Abierta - iRobot

III. Accesorios

El iRobot Create cuenta con una amplia gama de accesorios compatibles con su arquitectura, entre los más importantes tenemos:

Virtual Wall

Este dispositivo crea una barrera invisible (ver Figura. 36) que el robot no atravesará emitiendo señales de infrarrojos que el robot detectara con su receptor infrarrojo de barrera. La barrera invisible puede oscilar entre tres a ocho pies de largo. Se puede definir el rango de la barrera invisible ajustando un control deslizante ubicado sobre el Virtual Wall. Para crear una barrera invisible superior a ocho pies de ancho, se necesita Virtual Walls adicionales.

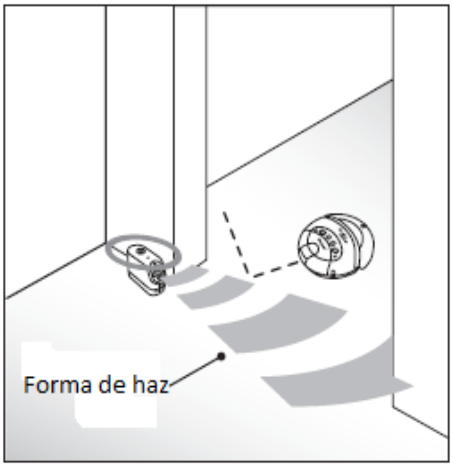


Figura 36. Virtual Wall
Fuente: Guía del propietario - iRobot

Home Base

Este dispositivo (ver Figura. 37) permite al robot cargar su batería en caso de que lo necesite. Si durante la demostración de cobertura, la batería se está agotando y el receptor de infrarrojos omnidireccional del iRobot Create detecta el transmisor de señales infrarrojas del Home Base, iRobot Create se acoplara automáticamente sobre el Home Base para recargarse. Además de lo ya mencionado este dispositivo cuenta con tres transmisores infrarrojos que se pueden utilizar en aplicaciones personales.

Figura 37. Home Base
Fuente: Guía del propietario – iRobot

Control remoto

El control remoto (ver Figura 38) por infrarrojos permite controlar el iRobot desde el otro lado de la habitación. Con el control remoto, se puede seleccionar e iniciar una demostración incorporada, así como conducir directamente el robot con el botón de control direccional. Con la interfaz abierta, se pueden leer todas las señales transmitidas por el mando a distancia a través del puerto serie del iRobot, así como agregar la funcionalidad del control remoto para distintas aplicaciones generadas por el programador.

Figura 38. Control remote
Fuente: Guía del propietario - iRobot

Command Module

El Command Module trabaja con el iRobot Create, facilitando un modo de escribir programas en C o C ++ para el control del robot, también permite añadir hardware adicional para ampliar las capacidades del robot, sin necesidad de tener que estar sujeto a un ordenador personal para que este pueda ejecutar comportamientos más complejos.

El Command Module (ver Figura 39) va conectado directamente al Cargo Bay Connector del iRobot, este tiene cuatro puertos de expansión que proporcionan entradas y salidas adicionales para el hadware adicional que se desee conectar, tres de los puertos se encuentran de manera distribuida sobre la parte superior para proporcionar puntos de acceso fáciles para una serie de sensores, y un conector está ubicado en la parte lateral frente a la bodega de carga para el fácil montaje de dispositivos electrónicos.

Consta de un microcontrolador Atmel AVR ATMega168, con el cual se puede reprogramar el iRobot realizando una descarga del código de la Pc hacia el robot, a través de un cable USB.

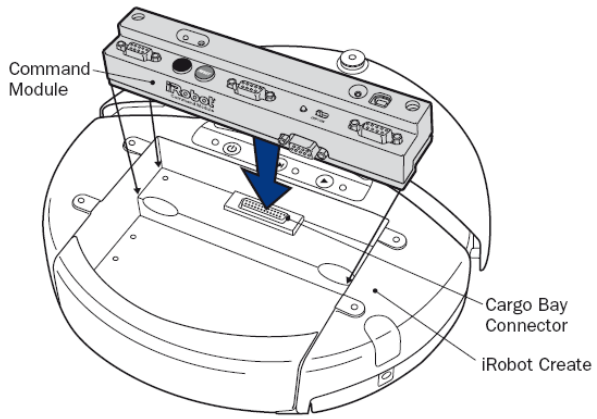


Figura 39. Command Module

Fuente: Manual del propietario - iRobo

BAM (Bluetooth Adapter Module)

Sirve para controlar al iRobot mediante redes inalámbricas hasta 300 pies (100m) de distancia. Al crear una conexión con el BAM se tendrá plena autoridad para el control inalámbrico del robot, iRobot Create puede ser controlado desde cualquier Windows o Linux PC o equipo Macintosh.

El BAM (ver Figura.40) se conecta al iRobot Create a través del puerto de expansión sin ningún tipo de cables extra, proporcionando una conexión serie virtual entre el ordenador y el robot Create, así el equipo puede comunicarse con iRobot Create de la misma manera que si este estuviese conectado a través de un cable serial.

Figura 40. Bluetooth Adapter Module
Fuente: Guía del propietario - iRobot

3.1.1.2. Descripción de los Sensores

Los sensores permiten la adquisición de datos necesarios para el control del robot. Dentro del estudio de sensores hay que tomar en cuenta la medida de las magnitudes y su correcta representación en forma compatible para su procesamiento.

En la toma de medidas siempre existe un cierto grado de incertidumbre. En principio, el incremento de la información hace posible la reducción de esta, para ellos se trata de tomar más medidas o de emplear sensores redundantes.

Existen diferentes transmisores de información basados en distintos principios físicos y químicos. Así, entre los principios y parámetros involucrados cabe mencionar:

- Mecánica: posición, velocidad, tamaño, fuerza, etc.
- Termotecnia: temperatura, calor, entropía, etc.
- Magnetismo: intensidad de campo, densidad de flujo, permeabilidad, etc.
- Química: concentración de un material, estructura cristalina, etc.
- Radiación (ondas electromagnéticas) de todas las frecuencias, desde ondas de radio a rayos γ: intensidad, frecuencia, polarización, etc.

Con respecto al procesamiento y transmisión de la información pueden emplearse también distintas tecnologías con limitaciones físicas diferentes:

- Hidráulica mediante el empleo de componentes fluídicos. En este caso existe el límite de la velocidad del sonido en un fluido, que es aproximadamente 103 m/s.
- Eléctrica y electrónica en la actualidad se emplean circuitos electrónicos. El límite de la velocidad viene dado por la movilidad de las cargas en un material semiconductor, que es aproximadamente 105 m/s.

- Radiante empleando componentes ópticos. El límite es la velocidad de la luz en la guía, aproximadamente 108 m/s.

En la actualidad, se maneja casi con exclusividad el procesamiento electrónico de la señal; para su empleo es necesario traducir las magnitudes físicas a señales eléctricas. Los sensores realizan transformaciones de energía frecuentemente, así por ejemplo en un codificador óptico, la señal de entrada es la rotación mecánica del eje y la salida es una señal eléctrica, en esta transformación se emplea un haz de luz como fuente auxiliar de energía, esta energía modulada por la rotación mecánica produce una señal de salida digital que puede ser interpretada por una PC o un microcontrolador.

Los sensores presentes en el iRobot Create modelo 4400 son tres:

a. Sensor mecánico de choque (Bumper)
b. Sensor de rotación (Encoder)
c. Sensor de proximidad IR (Cliff Sensor)

Sin embargo se pueden añadir muchos más sensores con la ayuda del Command Module, el cual cuenta con cuatro puertos de expansión más, a los cuales podemos conectar.

a. Sensor mecánico de choque

Características Técnicas

Los sensores mecánicos de choque también denominados bumpers (Figura. 41), fundamentan su detección en el contacto mecánico del elemento a detectar con una parte del sensor (palanca de accionamiento). Este contacto mecánico

produce el cierre de un conmutador (NA) de dos posiciones con muelle de retorno a la posición de reposo.

Figura 41. Sensores mecánicos de choque
Fuente: Tecnotec - Robokit

El iRobot Create cuenta con dos sensores mecánicos de choque (bumpers), uno al lado izquierdo y otro al lado derecho, su disposición e infraestructura dentro del Create se pueden observar en la Figura. 42

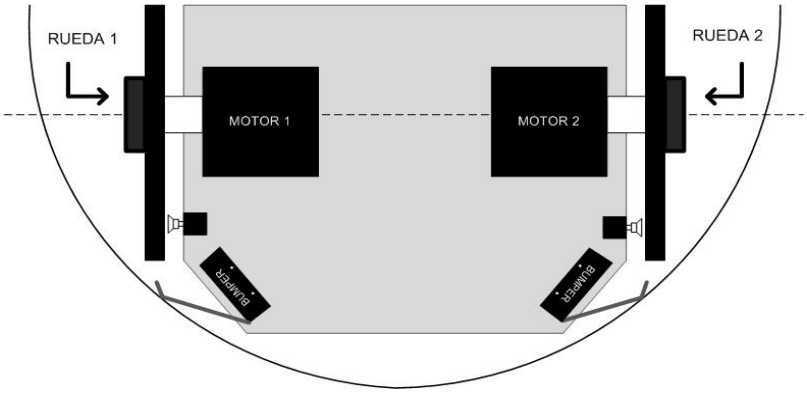


Figura 42. Sensores mecánicos de choque en el iRobot Create

Funcionamiento

Cuando el bumper está en reposo, sin accionar, el terminal o patilla común (C) y el terminal R están en contacto. Cuando se aplica presión sobre la palanca, el terminal C entra en contacto con el terminal A: el bumper pasa a la posición activo. En ese momento se oye un clic que nos indica el contacto entre terminales, lo que ocurre cuando la palanca llega prácticamente al final de su recorrido.

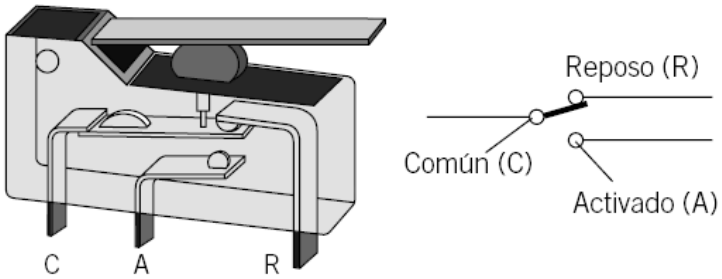


Figura 43. Funcionamiento de los sensores mecánicos de choque
Fuente: Hacking Roomba Magazine

b. Sensor de rotación

Características Técnicas

Los sensores de rotación son también llamados encoders; a partir de un desplazamiento lineal o angular estos sensores van a proporcionar una señal digital, lo que permitirá obtener una medida absoluta o incremental del desplazamiento. Estos encoders hacen uso de una señal óptica para determinar la magnitud de un desplazamiento o posición angular.

Figura 44. Sensor de Rotation en el IRobot Create
Fuente: Robot Magazine

Funcionamiento

El encoder posee una fuente de luz, un fotodiodo receptor y un disco óptico con una serie de marcas (ver Figura. 45). El eje cuya posición se quiere medir va acoplado al disco, a medida que el eje gira se van generando pulsos en el receptor cada vez que la luz atraviese las marcas, llevando una cuenta de estos pulsos es posible conocer la posición del eje.

La resolución depende del número de marcas que se pueden poner físicamente en el disco.

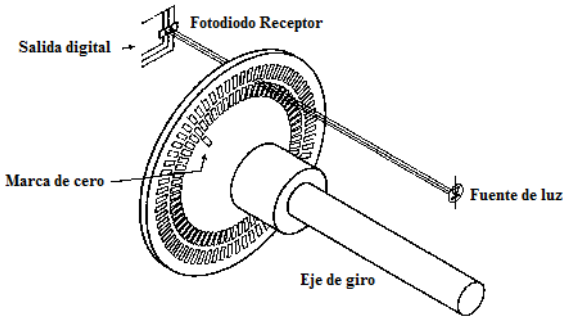


Figura 45. Funcionamiento de los sensores de rotación
Fuente: Hacking Roomba Magazine

c. Sensor de proximidad IR

Características Técnicas

Los sensores de proximidad IR mejor conocidos en el Create como Cliff sensors (ver Figura. 46), hace uso de cuatro entradas análogas para la toma de datos: señal de vacío izquierda, señal de vacío frontal izquierda, señal de vacío derecha y señal de vacío frontal derecha

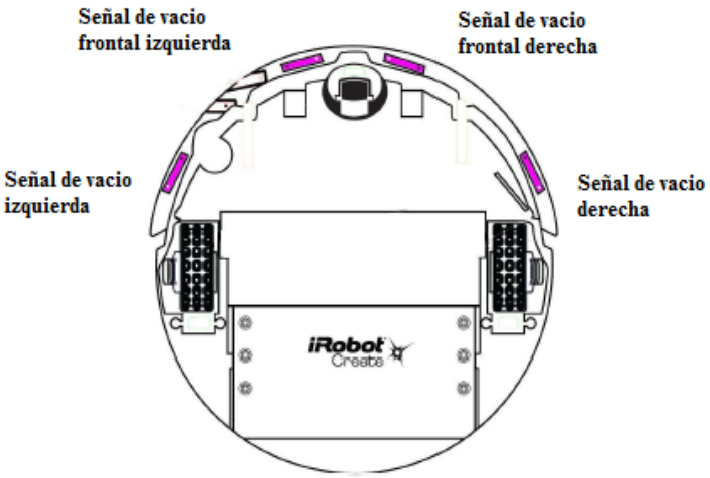


Figura 46. Cliff Sensor presentes en el iRobot Create
Fuente: Guía del propietario - iRobot

Se tiene que tener presente que esta configuración es sensible a la luz del ambiente perjudicando las medidas, pueden dar lugar a errores, así pues es importante trabajar en ambientes de luz controlada.

Funcionamiento

Este tipo de sensor presenta una cara frontal en la que encontramos tanto al LED como al fototransistor. Debido a esta configuración el sistema tiene que medir la radiación proveniente del reflejo de la luz emitida por el LED. Hay que

tener muy en cuenta el coeficiente de reflectividad del objeto, el funcionamiento del sensor será diferente según el tipo de superficie. En la Figura. 47 se puede observar el rango de valores de lectura dependiendo del tipo de superficie.

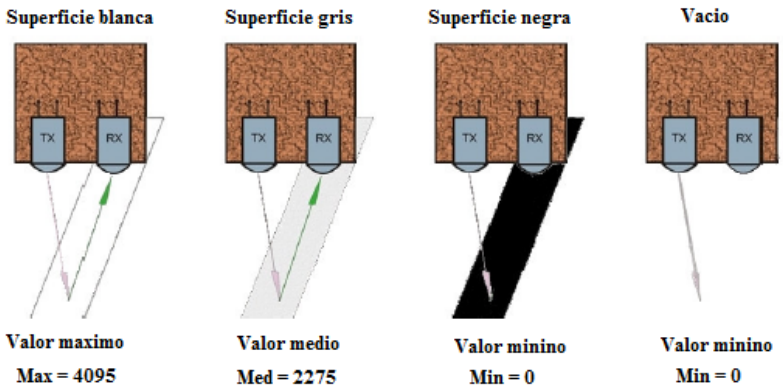


Figura 47. Funcionamiento de los Cliff Sensors
Fuente: Hacking Roomba Magazine

3.1.1.3. Descripción de los Efectores y Actuadores

Un efector corresponde a cualquier dispositivo que afecte o modifique al medio ambiente. Ejemplos de efectores robóticos son piernas, ruedas, brazos, dedos y pinzas. Un efector robótico esta siempre bajo el control del robot. Cualquier mecanismo que permita al efector ejecutar una acción es denominado como un actuador. Ejemplos de actuadores robóticos son motores eléctricos (servomotores, de paso, de corriente continua, etc.), cilindros neumáticos e hidráulicos.

3.1.1.3.1. Efectores presentes en el Create

a. Ruedas principales

Como se puede observar en la Figura. 48 el iRobot Create dispone de dos ruedas principales izquierda y derecha de idénticas características, por lo que la descripción de estas se realizara de forma general.

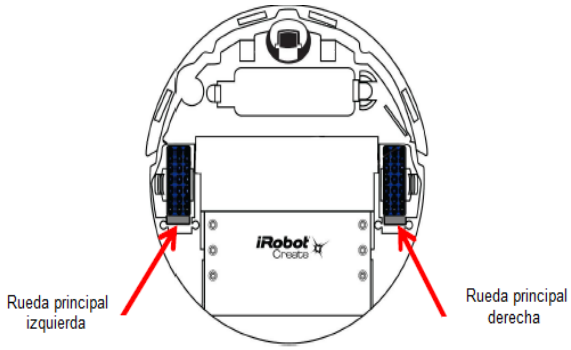


Figura 48. Descripción de pines del conector DB-25
Fuente: Guía del propietario - iRobot

Características Técnicas

Las ruedas del iRobot son de mando directo, alcanza velocidades entre -500 a 500 mm/s, se debe aclarar que los signos indican el sentido de giro de las ruedas.

El diámetro de las ruedas es de 6.6 cm. Las ruedas se caracterizan por ser del tipo convencional, poseer un labrado estilo grapa (mayor adherencia con la superficie en contacto) y por contener una robusta suspensión. Se encuentran en posición retráctil sujetas por clips (ver Figura. 49).

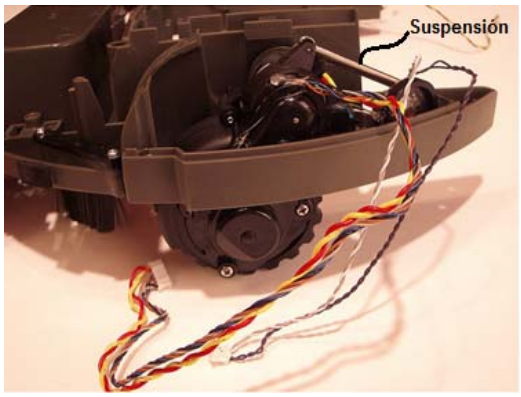


Figura 49. Sistema de suspensión
Fuente: Robot Magazine

Estas ruedas se adaptan a una correa, la cual está conectada al eje del motor; esto hace que la fuerza del motor ponga en movimiento a la rueda (ver Figura. 50).

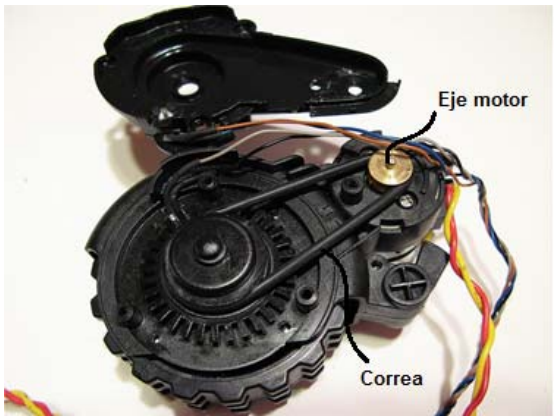


Figura 50. Sistema de transmisión de las ruedas principales
Fuente: Robot Magazine

Como se mencionó anteriormente el Create viene equipado con ruedas retractiles sujetas por clips, sin embargo estos clips se pueden extraer, ha este nuevo posicionamiento se lo conoce como publicación.

Las ventajas tanto en el posicionamiento retráctil como en el de publicación son:

- La rueda en posicionamiento retráctil (los clips en su lugar), otorga al iRobot Create una mayor estabilidad. Esto es sumamente ventajoso si está trabajando con una carga útil dispuesta en la bahía de carga.

- La rueda en posicionamiento de publicación (clips eliminados) proporciona al iRobot Create una mayor movilidad. Esto es útil si se desea que el iRobot Create pueda desplazarse sobre superficies irregulares.

b. Rueda delantera fija

Características Técnicas

Es una rueda fija del tipo castor (ver Figura 51) cuyo diámetro aproximado es de 2.4 cm., brinda soporte a la parte delantera del Create, ofreciendo una mayor estabilidad al robot. Contiene un eje giratorio que permite al iRobot Create girar sobre su propio eje.

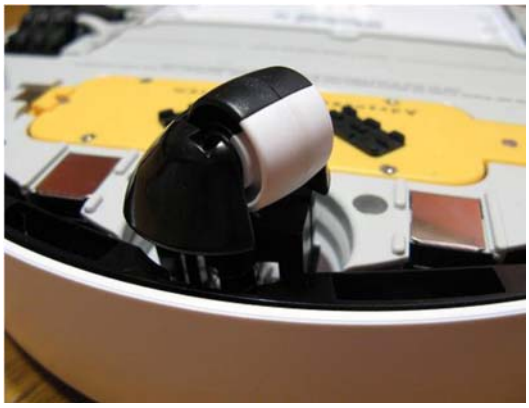


Figura 51. Rueda delantera fija
Fuente: Hacking Room

c. Rueda trasera fija

Características Técnicas

Es una rueda fija cuyo diámetro aproximado es de 2.4 cm., brinda soporte a la parte trasera del Create, ofreciendo una mayor estabilidad al robot, evitando así que esta se arrastre cuando se añade carga o algún dispositivo a la bahía de carga útil, como se puede visualizar en la Figura. 52.

Figura 52. Rueda trasera fija
Fuente: Hacking Room

3.1.1.3.2. Actuadores presentes en el Create

a. Motores

Características Técnicas

El iRobot Create dispone de dos motores principales de corriente continua, tanto para la rueda izquierda como para la derecha, su voltaje de alimentación es de aproximadamente 6V, aunque puede irse a los extremos en aplicaciones especiales de motores, estos pueden funcionar por encima o por debajo de la tensión nominal (para cumplir con otros requisitos de diseño). El motor está conectado a la rueda a través de una transmisión por correa (ver Figura. 53).

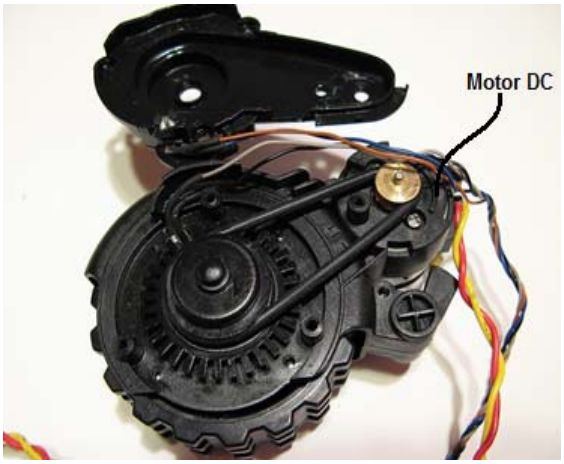


Figura 53. Motor Corriente Continua

Fuente: Robot Magazine

Dentro de las descripciones más relevantes se detallan las siguientes:

- No supera la velocidad de 8.200 rpm.
- Dimensiones: 22mm x 24.2mm.
- Peso: 46g.
- Encoder Integrado: 512 pulse linear encoder.

3.2. SOFTWARE

3.2.1. Microsoft Robotics Developer Studio 2008 (MSRS)

Es una plataforma de desarrollo robótica para aficionados, académicos y comerciantes, que posibilita el proceso de creación de aplicaciones de control robóticas para una serie de plataformas hardware, bajo un modelo de programación grafico basado en flujo de datos. Entre otras, las funcionalidades clave de Microsoft Robotics Studio son proporcionar una plataforma de desarrollo extremo a extremo, soporte de concurrencia en tiempo de ejecución orientado a

servicios y una plataforma escalable y extensible. Los principales componentes de esta plataforma de desarrollo robótica son:

- Un lenguaje visual de programación (VPL), que permite la creación intuitiva y estructurada de aplicaciones para robots.

- Un entorno de simulación en tres dimensiones, basado en el motor de simulación física AGEIA Physx.

- Soporte en tiempo de ejecución (Runtime) que gestiona la entrada/salida asíncrona, la concurrencia y la distribución de servicios.

En términos generales una aplicación de Microsoft Robotics Studio es una coordinación (orquestación) de distintos servicios distribuidos y asíncronos. Por ejemplo, un sensor se representa por un servicio que ofrece una entrada de información, un actuador tendrá otro servicio asociado que representara la respuesta física del robot, finalmente, un servicio controlador se encarga de interpretar la información obtenida por los sensores y enviar a través de comandos apropiados a los actuadores.

La coordinación se produce en la comunicación asíncrona entre este conjunto de servicios. Es decir que si dentro del Create, el servicio de parachoques (bumper) detecta un impacto, este envía un mensaje al servicio controlador, que a su vez decidirá qué mensaje enviar al servicio de control de las ruedas para realizar la acción correspondiente ante una determinada circunstancia. El grado de complejidad de este escenario aumenta cuando el número de sensores y actuadores aumenta relativamente. El funcionamiento de los servicios asociados a cada uno de los sensores y actuadores sigue siendo de igual forma al ejemplo anteriormente citado, sin embargo, el servicio coordinador (o servicios coordinadores) deben manejar demasiada cantidad de datos en tiempo de ejecución y aplicar complejas capacidades de control.

3.2.1.1. MSRS Runtime

El soporte en tiempo de ejecución o Runtime de Robotics Studio, como se muestra en la Figura. 54, consta de dos elementos substanciales que hacen posible la construcción, supervisión, despliegue y funcionamiento de un gran nivel de aplicaciones. Estos dos componentes son el CCR (Concurrency and Coordination Runtime) y el DSS (Decentralized Software Services).

CLR es compatible con tiempos de ejecución de la base y proporciona acceso a. NET Framework.

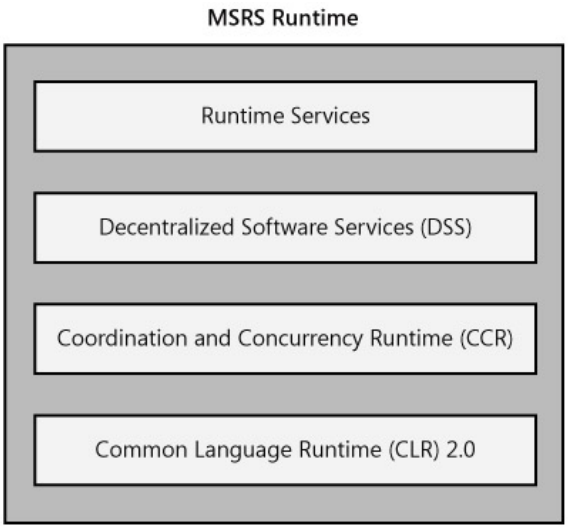


Figura 54. MSRS Runtime
Fuente: Professional Microsoft Robotics Developer Studio

3.2.1.1.1. Decentralized Software Services (DSS)

El DSS combina la arquitectura tradicional Web con partes de la arquitectura de Servicios de la Web. El modelo de la arquitectura resultante se basa en servicios para poder coordinar aplicaciones distribuidas, requiere baja carga computacional y está basado en la tecnología .NET.

Estos servicios siguen un modelo REST (Transferencia de estado representacional), su comportamiento está basado en cambios de estado internos, y en los mensajes y notificaciones que intercambian entre sí. Los servicios se pueden considerar como una unidad básica computacional que se utilizan en MSRS para construir aplicaciones, estos servicios se ejecutan independientemente y se comunican a través de mensajes que atraviesan sus puertos. Todos los servicios precisan de un nodo DSS para poder ser ejecutados. Un nodo DSS es un ambiente de ejecución que proporciona soporte a los servicios para ser creados y manejados hasta que se eliminen o el nodo DSS se pare. En la Figura. 55 se puede observar el modelo de servicios DSS

.

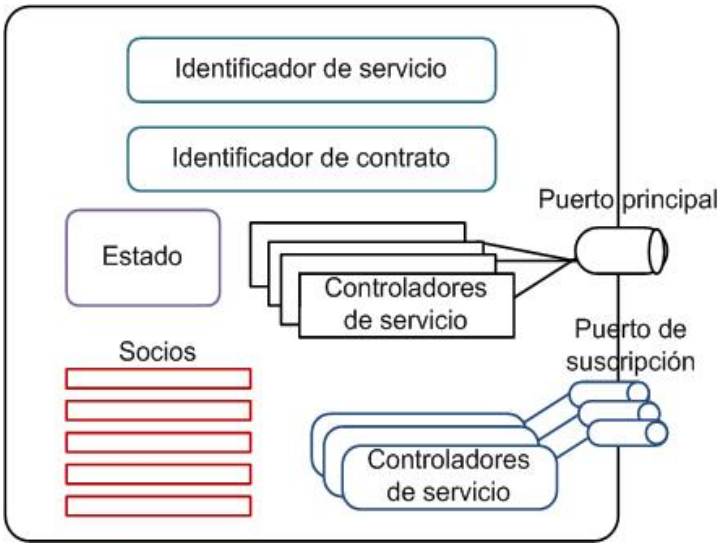


Figura 55. Modelo de servicios DSS

El modelo de servicios DSS ha sido planteado para facilitar la reutilización de servicios, consiguiendo un manejo sencillo. Los principales componentes de un servicio se describen a continuación:

- **Identificador de servicio:** Este identificador se crea cada vez que se ejecuta una instancia del servicio, proporcionando identidad al servicio, dando opción a

que otros servicios puedan identificarlo y comunicarse con él o incluso permitir a un navegador Web acceder a él.

- **Identificador de contrato:** Se trata de una descripción o identificación única de la funcionalidad que implementa el servicio, aquí se definen las operaciones que va a soportar el servicio, así como el tipo de datos que requieren estas operaciones, de modo que otros servicios puedan reutilizarlo

- **Estado del servicio:** Contiene el estado del servicio en cualquier momento de la ejecución. Este estado contiene cualquier tipo de información que deba ser recuperada, modificada o supervisada de un servicio del DSS.

- **Servicios asociados:** Son servicios especiales que interactúan con otros y se encargan de que funcionen adecuadamente. Se declaran usando el atributo Partner.

- **Manejadores de servicio:** Genera un registro de cada una de las operaciones DSS definidas sobre el puerto principal. Se definen utilizando el atributo ServiceHandlers.

- **Puerto principal:** Es un puerto CCR. A través de este puerto el servicio recibe mensajes de solicitud de ejecución de operaciones del exterior por parte de los servicios asociados y emite los mensajes de respuesta oportunos. Este puerto es un miembro privado de la clase Service y se identifica con el atributo ServicePort.

- **Notificaciones:** Son un conjunto de mensajes de aviso generados por parte del servicio, referente a la actualización de su estado y que pone a disposición de los servicios asociados para que estos puedan hacer uso de ellos.

- **Puerto de suscripción:** Es un puerto a través del cual los servicios asociados pueden acceder a las notificaciones de estado suscitadas dentro del servicio.

3.2.1.1.2. Concurrency and Coordination Runtime (CCR)

En el desarrollo de aplicaciones de control para robots uno de los principales aspectos a tomar en cuenta es la ejecución y coordinación de múltiples tareas interactivas y simultaneas.

El CCR es la pieza que une el soporte en tiempo de ejecución de MSRS a un servicio. Introducido al mismo tiempo que MSRS, el CCR permite a las aplicaciones realizar operaciones asincrónicas y concurrentes. El CCR es una librería de .NET que se puede utilizar en cualquier aplicación que requiera de procesamiento asincrónico. De esta manera se protege a los desarrolladores de todas las dificultades que involucra la escritura de código referente a la asignación de hilos de ejecución para las distintas tareas.

El procesamiento asincrónico es importante para las aplicaciones robóticas porque, sin él, el robot únicamente movería una sola rueda, brazo o pierna a la vez, teniendo que esperar a que cada servicio sea completado antes de poder inicializar un nuevo servicio.

Los robots normalmente son equipados con múltiples sensores que leen todo tipo de datos dentro de sus entornos de trabajo. Al mismo tiempo, tienen actuadores que se utilizan para realizar un movimiento o lograr un objetivo. Todos estos sensores y actuadores deben estar coordinados para lograr un objetivo en común, que es el correcto funcionamiento del robot. El CCR maneja los hilos individualmente que son necesarios para que esto suceda.

Microsoft Robotics hace uso de funciones definidas en el CCR para aminorar la complejidad de este tipo de entornos. El CCR ofrece al programador un

conjunto de clases que le permite desarrollar fácilmente diversos modelos de coordinación. Existen tres componentes fundamentales del CCR:

a) Los puertos (clases Port y PortSet)
b) Las primitivas de coordinación (clase Arbiter)
c) Las tareas y colas de ejecución (clases Dispatcher, DispatcherQueue y Task)

a) Clase Port y PortSet

La Clase Port de CCR es la primitiva que más frecuentemente se utiliza al programar con MSRS. La clase Port es una cola FIFO (First in – First out) de objetos que implementa las interfaces IPortRecieve e Iport. Estas interfaces separan los métodos que añaden elementos al puerto de los que los recuperan. Hay que tener muy en cuenta que añadir un objeto a un puerto es una tarea asíncrona. La tarea de recuperación de un elemento de un puerto se puedan dar de en dos escenarios diferentes:

Si el puerto se usa como una cola pasiva, no se realiza ninguna tarea cuando se ha enviado algún elemento al puerto. Los elementos se pueden recuperar llamando al método Test, que devuelve el valor booleano false y un objeto null si no hay ningún elemento. Este escenario se usa cuando se tiene algún otro mecanismo para ejecutar el código del evento y lo único que se persigue es el uso eficiente de una cola FIFO. Además, proporciona la flexibilidad de añadir un receptor al puerto posteriormente.

Si el puerto se usa como una cola activa, las tareas están programadas de tal forma que cuando se envía un objeto al puerto estas se ejecutan. Como se verá a continuación, uno o más **Arbiters** tienen registrado el puerto. Esta opción es la más común de uso del puerto al programar con CCR.

La clase **PortSet,** tiene la misma funcionalidad de **Port,** con la única diferencia que esta clase acepta varias clases de elementos, es un puerto que representa diferentes tipos de datos y puede recibir cualquiera de ellos. Por lo general se crea un PortSet que soporta N tipos diferentes y es capaz de coordinar los diferentes mensajes independientemente.

b) Clase Arbiter

Las primitivas de coordinación o **Arbiters** son las clases que coordinan la ejecución del código. El CCR permite asignar hilos de ejecución a las distintas tareas. Esto se consigue a través de los mensajes recibidos en los puertos. Hay dos escenarios en los que el CCR permite la coordinación del flujo de ejecución del programa:

- **Coordinación de peticiones de entrada.** Por ejemplo, la espera a una respuesta HTTP en un puerto de red.

- **Coordinación de respuesta de una o más peticiones.** Por ejemplo, la espera de la finalización de ejecución de una función con éxito o con error.

Para trabajar con las primitivas de coordinación se usa la clase Arbiter que proporciona diversos métodos estáticos como los siguientes:

- Arbiter.FromTask - Crea instancias de Task.

- Arbiter.Choice - Crea instancias de Choice.

- Arbiter.Receive - Crea instancias de Receive.

- Arbiter.Interleave - Crea instancias de Interleave.

Un receptor de un elemento simple asocia un delegado (delegate – o método anónimo) de usuario cuya entrada es un parámetro simple de tipo T, donde la instancia sería Port<T>. Si la opción de persistencia es true, el receptor ejecutará el delegado por cada elemento enviado. Si por el contrario, la opción de persistencia es false, el receptor sólo ejecutará el delegado para un elemento y luego se retirará del puerto.

A continuación se explican brevemente algunas de las primitivas de coordinación más usadas en el CCR. Arbiter.Choice es importante, ya que permite crear una jerarquía de Arbiters y así invocarlos en el orden correcto, comprobando anteriormente, si el código de usuario está preparado para ser ejecutado. Esto permite al programador, expresar una coordinación compleja en apenas unas líneas de código. Además, es un patrón que especifica una elección entre diferentes tareas a ejecutar. Otro método de coordinación importante que implementa Arbiter es Interleave. En este caso el CCR está pendiente de las respuestas y activaciones hasta que el proceso ha terminado. El programador sólo tiene que especificar qué protección necesita. Esta es la solución que se propone en MSRS al acceso de recursos privados que habitualmente ocurre por no estar debidamente protegidos.

Cuando tenemos múltiples elementos receptores, podemos distinguir dos posibles escenarios:

- Los conocidos como Joins son receptores que están preparados para recibir de uno o más puertos, si uno de los intentos falla, vuelven a ponerlo en la cola y esperan un tiempo hasta que las condiciones sean propicias para conseguirlo con éxito. Se puede usar para garantizar acceso a recursos múltiples. El número de puertos y de elementos se puede establecer en tiempo de compilación o en tiempo de ejecución.

- Receptores que eliminan elementos por cada puerto que participa en la recepción. Cuando se llega a un número satisfactorio de elementos, se ejecuta el

delegado. Se suele utilizar en escenarios en los que existe múltiples respuestas pendientes.

c) Tareas y colas de ejecución

Las aplicaciones robóticas de manera general se ejecutan en un entorno multitarea. Debido a esto es necesario disponer de un mecanismo que permita asignar hilos de ejecución a las distintas tareas. En el CCR este proceso se lleva a cabo gracias a los dispensadores. Es posible continuar con la ejecución de ciertas partes del código mientras otros quedan a la espera de que acontezcan ciertos eventos, como cuando se ejecuta un receptor a la espera de que llegue un mensaje al puerto. Para una correcta orquestación o coordinación de los recursos de ejecución se cuenta con las siguientes clases:

- **La interfaz ITask y las implementaciones Task e IterativeTask.** Solamente las clases que implementan ITask se pueden programar. Los Arbiters son un ejemplo de ello, es decir, se pueden programar y activar correctamente.

- **La clase de DispatcherQueue.** Implementa una cola FIFO de Task (tareas), es decir, delegados que identifican los métodos que se pueden ejecutar.

- **La clase Dispatcher**. Maneja las tareas de los hilos del SO y el desencolado de unas o más instancias DispatcherQueue.

A continuación se puede observar en la Figura. 56 un esquema del sistema de coordinación multitarea del CCR.

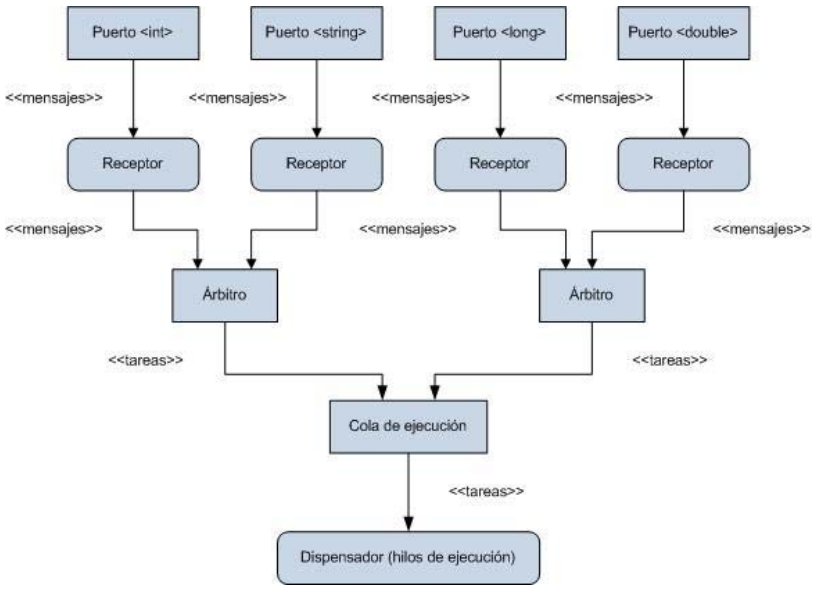


Figura 56. Esquema de coordinación multitarea en CCR

Esta representación esboza el modelo de coordinación multitarea suministrada por el CCR. Los mensajes de solicitud de ejecución de operaciones por parte de los servicios asociados llegan a través de los distintos puertos de comunicación especializados en un determinado tipo de mensaje. Tras la llegada de un determinado mensaje a uno de los puertos, el respectivo receptor para ese mensaje lo captura y el árbitro, en base al tipo de mensaje recibido, decide qué tarea (encapsulada como un objeto ITask) debe ejecutarse y la almacena en una cola de ejecución a la espera de ser ejecutada. Por último, el dispensador recorre toda la cola de ejecución, asignando a las distintas tareas almacenadas, un hilo para que puedan ser ejecutadas, consiguiendo así un entorno multitarea.

3.2.1.2. Visual Simulation Environment

Microsoft Robotics Studio surgió como una herramienta para acelerar el desarrollo y adopción de la robótica. Gran parte de este esfuerzo está centrado en la simulación. La herramienta Visual Simulation Environment (VSE) incluida con el MSRS ofrece a los programadores una alternativa de acceso al mundo de la

robótica sin la necesidad de algún hardware. Repotenciado por AGEIA PhysX, VSE ofrece gráficos de alta calidad, comparable a los generados por los mejores juegos de computadora (ver Figura. 57).

La simulación permite a los programadores experimentar con robots nuevos incluso antes de construirlos físicamente. También se puede desarrollar aplicaciones para robots existentes y luego probar los resultados en los robots físicos. Además, proporciona al desarrollador un mecanismo para corregir y eliminar fallos básicos que se podrán probar con anticipación en el entorno virtual.

Una característica adicional a las ya mencionadas, es que permite simular casi con exactitud la misma aplicación que se ejecutara en el robot real dentro del escenario de trabajo. Hay que tomar en cuenta que la simulación tiene sus limitantes, ya que el mundo real es no determinístico y dinámico.

Figura 57. Simulador de Robotics Studio
Fuente: Professional Microsoft Robotics Developer Studio

El entorno de simulación visual está compuesto de varios módulos:

- **Simulation Engine Service:** Responsable del progreso del tiempo en el motor físico de la simulación.

- **Managed Physics Engine Wrapper:** Abstrae al programador del nivel bajo del motor físico.

- **Native Physics Engine Library:** Permite la aceleración del hardware a través de AGEIA PhysX Technology.

- **Entities:** Representa el hardware y los objetos físicos en el mundo simulado. Un gran número de entidades están definidas en MSRS permitiendo crear rápidamente un entorno de simulación.

3.2.1.3. Visual Programming Language

Microsoft Visual Programming Language (VPL) es un entorno de desarrollo de aplicaciones diseñada bajo un modelo de programación gráfica, basada en el flujo de datos en vez del típico flujo de control que se encuentra en la programación convencional. En lugar de una serie de comandos imperativos ejecutados secuencialmente, un programa de flujo de datos se parece más a una serie de trabajadores en una línea de ensamblaje quienes realizan su tarea conforme el material llega a cada una de sus líneas de producción. Como resultado de esto VPL es una herramienta adecuada para la programación de una variedad de escenarios que hagan uso del procesamiento concurrente o distribuido.

El objetivo de este lenguaje de programación es facilitar a los programadores novatos, el entendimiento de conceptos como variables o lógica. Sin embrago, no está limitado únicamente a este tipo de programadores. Este lenguaje de programación llama la atención de programadores avanzados por su facilidad al momento de desarrollar prototipos o generar código. Esta herramienta no solo está sujeta al desarrollo de aplicaciones robóticas a través de su caja de herramientas, ya que al estar basada en la tecnología .NET, su arquitectura subyacente permite el desarrollo de otras aplicaciones tecnológicas. Producto de todos estos beneficios,

VPL abarca una gran audiencia de usuarios, que van desde novatos pasando por estudiantes hasta expertos programadores o aficionados.

El flujo de datos que maneja Visual Programming Language fundamenta su desempeño en conexiones secuenciales de actividades representadas por bloques con entradas y salidas que pueden ser conectados con otros bloques. El pin derecho de todo bloque representa la salida y el izquierdo la entrada. Las actividades pueden representar tareas preconstruidas, control de flujo de datos, funciones u otros módulos de código. Esto permitirá manejar a un robot como un conjunto de servicios que se ejecutan de manera concurrente y cuya orquestación entre sí, resultara clave para definir el comportamiento del RMR.

Las actividades pueden incluirse asimismo como composiciones de otras actividades. Éstas hacen posible la composición de actividades en base a la reutilización de bloques funcionales. En este sentido, una aplicación construida en VPL es en sí misma una actividad.

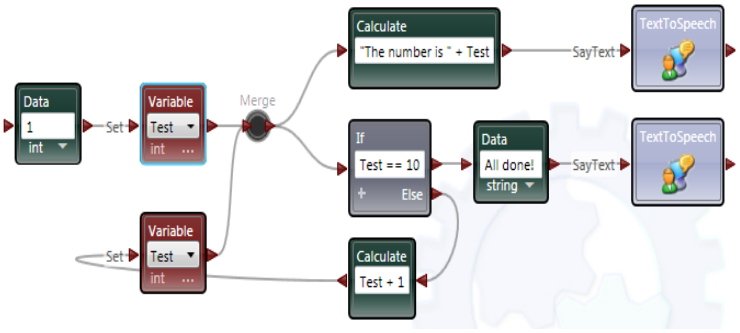


Figura 58. Diagrama VPL

Fuente: Professional Microsoft Robotics Developer Studio

Los bloques contienen los nombres de la propia actividad y las fronteras de ella. También pueden contener diferentes atributos o gráficos que definen la finalidad de la actividad. Además, una actividad puede tener múltiples conexiones

de entradas y sus correspondientes conexiones de salida. Una conexión de salida puede representar varias cosas: un resultado o una notificación.

Un servicio o actividad necesita saber su estado inicial y también las relaciones con los llamados Servicios asociados, citados anteriormente. Es por ello que debemos configurarlos, con opciones como la velocidad o el tipo de puerto que usa.

Una vez conectados los bloques, el proyecto se puede ejecutar y subsanar los errores que aparezcan. Además, se podrá generar código del proyecto realizado. Este código se genera en C# y puede servir como punto de partida para crear aplicaciones más complejas.

3.2.2. Solid Works

El software de automatización de diseño mecánico de SolidWorks es una herramienta de diseño de **modelado sólido paramétrica** y **basada en operaciones** que aprovecha la facilidad de aprendizaje de la interfaz gráfica de usuario de Windows. Puede crear modelos sólidos en 3D **totalmente asociativos** con o sin **restricciones** mientras utiliza al mismo tiempo las relaciones automáticas o definidas por el usuario para capturar la **intención del diseño.**

A continuación se detallan los términos que aparecen en **negrita:**

- **Modelado sólido**

Un modelado solido es el tipo más completo de modelado geométrico presente dentro de los sistemas CAD (Computer-Aided Design). Contiene toda la geometría de superficie y alámbrica necesaria para detallar las aristas y las caras del modelo. A parte de la información geométrica, este contiene toda la información topológica

que es la que interrelaciona la geometría. Un ejemplo de información topológica seria que superficies se encuentran en qué arista. La inteligencia hace que funciones como el redondeo resulten tan fáciles como seleccionar una arista y especificar un radio.

- **Paramétrico**

Las cotas y las relaciones utilizadas para crear operaciones se capturan y se almacenan en el modelo. Gracias a ello, no únicamente es posible capturar la intención del diseño, sino que a más de esto se pueden realizar de una manera sencilla y rápida cambios en el modelo.

- **Basado en operaciones**

Del mismo modo que un ensamblaje está compuesto por una serie de piezas individuales, un modelo de SolidWorks también está compuesto por elementos individuales. Dichos elementos se denominan operaciones.

- **Totalmente asociativo**

Un modelo de SolidWorks es completamente asociativo a los dibujos y ensamblajes a los que hace referencia. Los cambios que se efectúan en el modelo se reflejan automáticamente en los ensamblajes y dibujos relacionados. De un modo similar, puede realizar modificaciones en el contexto del dibujo o del ensamblaje y tener la certeza de que esas notificaciones se traducen en el modelo.

- **Restricciones**

Las restricciones geométricas paralelas, perpendiculares, horizontales, verticales, concéntricas y coincidentes son únicamente algunas de las restricciones

que SolidWorks acepta. Además se puede hacer uso de ecuaciones para establecer relaciones matemáticas entre los parámetros.

- **Intención de diseño**

La intención de diseño es el plan que ha establecido con relación al comportamiento que debe presentar el modelo al modificarlo. Por ejemplo si se modela una saliente que contenga un taladro hasta una profundidad específica, el taladro debería desplazarse al desplazarse el saliente. Las técnicas utilizadas para crear el modelo determinan cómo y qué tipo de intención del diseño se captura.

3.2.3. Integración de Servicios

Microsoft Robotics Development Studio posee un soporte en tiempo real orientado a servicios y está desarrollado sobre el entorno .NET Framework, soporta lenguajes de programación como VB.NET, Python, Visual Basic, VPL o C#.

El modelo de arquitectura se basa en la estructura Cliente-Servidor. Para entenderlo de una manera mejor, cuando el programador ha creado un proyecto de MRDS, crea un servicio, y este servicio es ejecutado por un cliente. En este caso, el cliente es una Máquina Virtual llamada DSS. Los servicios son bloques básicos que se utilizan en MRDS para construir aplicaciones, cada servicio puede combinarse con otros servicios para crear dicha aplicaciones y serán denominados como **partners.** A este proceso de integración se lo conoce como **orchestration.**

El DSS da soporte a servicios distribuidos, combinando la arquitectura tradicional Web con parte de la arquitectura de servicios Web. Típicamente una aplicación está compuesta por un conjunto de servicios, los cuales requieren de un nodo DSS que es quien proporciona un entorno de soporte a los servicios (su

creación y su manejo), hasta que son eliminados o hasta que el DSS se detiene. La aplicación del modelo de software de servicios descentralizados, hace que sea sencillo el acceso, y la respuesta del estado del robot, usando un navegador Web o una aplicación basada en Windows. También facilita la reutilización modular de los servicios usando un modelo compuesto. Es posible construir funciones de alto nivel usando componentes simples que proponen la reutilización del código de los módulos con una mayor seguridad.

Dentro del desarrollo de este proyecto se hará uso de un conjunto de servicios para el control del iRobot Create, estos servicios se ejecutaran en paralelo y dependiendo de la correcta orquestación entre sí definirán el comportamiento del mismo.

CAPITULO 4

DISEÑO DE LA SOLUCIÓN

4.1.REQUISITOS DE LA SOLUCIÓN

HARDWARE

El proyecto está basado en la plataforma móvil iRobot Create, pero cualquier robot con una configuración cinemática de tracción diferencial puede sustituirlo, también puede ser ejecutado usando el simulador Visual Simulation Environment de Microsoft Robotics Studio.

Además de esto se necesitara de un micrófono conectado a una PC para transmitir las señales de voz al computador, este los procesara y los transmitirá al iRobot Create a través del Bluetooth Module Adapter.

SOFTWARE

Este proyecto está desarrollado bajo el lenguaje de programación Visual Programing Language de Microsfot Robotics Developer Studio.

Además de esto se necesita del software de Reconocimiento de Voz que viene comúnmente instalada con la versión estándar de Windows 7, Vista o XP.

Antes de iniciar con el proceso de desarrollo de la aplicación se deberá de configurar el Reconocimiento de Voz a través del Panel de Control de Windows,

de otra manera si existiesen inconvenientes al momento de la compilación del programa no habrá como establecer si la causa del error está en el diseño de la aplicación o si el Reconocimiento de Voz no está trabajando correctamente.

4.2.ARQUITECTURA DE LA APLICACIÓN

Estudios psicológicos han demostrado la existencia de dos tipos de conducta el intencional o controlado y el automático o reflejo. Por procesos controlados se entiende aquellos que requieren de la capacidad de razonamiento. Los procesos automáticos apenas requieren la intervención de la atención. Estas habilidades son la base de la arquitectura Hibrida o Automática-Deliberativa.

La comunicación entre estos dos niveles es bidireccional y se comunican a través de la memoria a corto plazo, en la que se encuentra información acerca del estado del robot y de dos flujos de información, el de eventos y el de órdenes de ejecución.

NIVEL DELIBERATIVO

En este nivel se encuentran los servicios que requieren razonamiento (habilidades deliberativas), las cuales no producen respuestas inmediatas ya que requieren de tiempo para la toma de decisiones en base a la información que disponen. Estas habilidades son ejecutadas por un **Secuenciador Principal,** ver en la igura 59, que es quien se encarga de coordinar las actividades para conseguir que el robot ejecute correctamente una tarea. Estas actividades se llevan a cabo de forma secuencial. La secuencia principal de ejecución necesaria para la consecución de una tarea viene establecida previamente y va a marcar el comportamiento del robot y cómo éste va a proceder en las diversas situaciones. Ejemplos de implementación de habilidades deliberativas son los planificadores, navegadores y los módulos que realizan el modelado del entorno (Barber, 2002).

Entre estas habilidades se encuentra la habilidad Navegador con Voz responsable de activar las habilidades del nivel Automático, que se describirá más adelante.

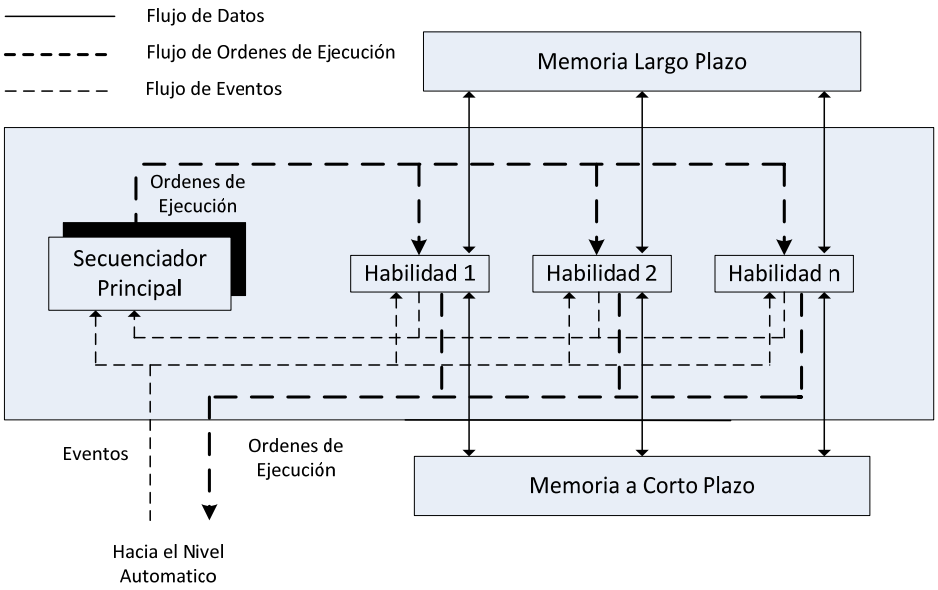


Figura 59. Nivel Deliberativo

NIVEL AUTOMÁTICO

El Nivel Automático se encarga del control a bajo nivel de los elementos electrónicos con los cuales está equipada la plataforma. Este nivel dota al robot de la capacidad de reacción ante entornos dinámicamente cambiantes en tiempo real. La figura 60 indica los diferentes elementos que constituyen este nivel:

- Sensores y actuadores, consiste en la toma e interpretación de los valores de lectura de los sensores y transformarlos en información útil para que pueda ser transmitido a los actuadores.

- Acciones reflejas, son las respuestas automáticas y prioritarias a los estímulos.

- Habilidades automáticas, son las capacidades sensoriales y motoras del sistema y son la base del concepto de comportamiento.

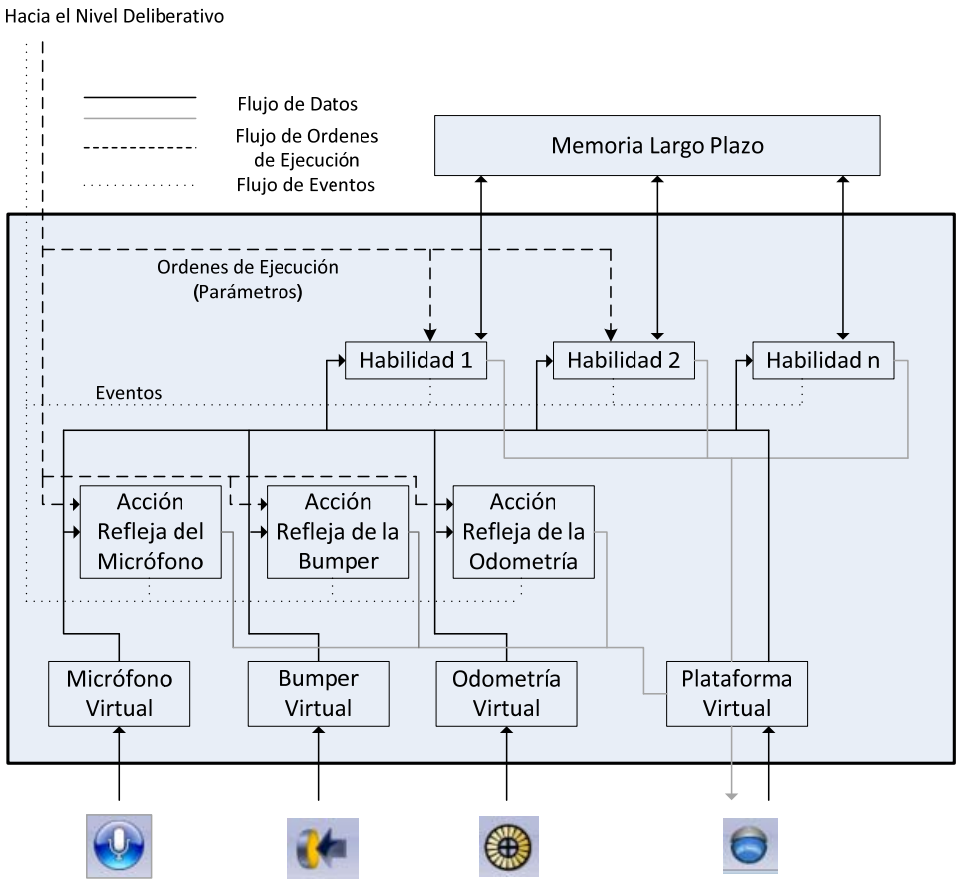


Figura 60. Nivel Automático

4.3.DESARROLLO DE LA APLICACIÓN

En este apartado se escribirá el código que permira a un ser humano controlar los movimientos del robot móvil con ruedas a través del reconocimiento de voz. Además se permitirá al humano enseñar tareas simples al robot, mediante la secuenciación de varios comandos de voz. El desarrollo de este trabajo abarca algunas de las características más avanzadas del Visual Programing Language, en particular hace uso conocimientos tales como actividades, servicios, concurrencia, coordinación, orquestación y listas.

Para que el robot pueda recibir señales de voz, se requiere hacer uso del servicio **Speech Recognizer** en el diagrama. Este servicio usa una gramática que define las palabras y frases que debería ser reconocidas por parte del robot, sin una gramática el Speech Recognizer no podrá interpretar ningún comando y el programa no podrá trabajar o ejecutarse correctamente.

La forma más fácil de crear una gramática es correr el servicio **Speech Recognizer Gui,** el cual proporciona una página web como una interfaz de configuración inicial.

Este servicio no requiere de ninguna conexión inicial para poder realizar la configuración del Reconocimiento de voz, únicamente se necesita colocarlo sobre el diagrama, de manera que este servicio conjuntamente con el Speech Recognition se pondrá en marcha cuando se ejecute el programa, ver Anexo 2 para mayor detalle de la configuración de este servicio.

A continuación se describe la solución que hemos desarrollado. Para ello, se relatan los distintos pasos que hemos realizado para implementar esta solución y las razones por las que tomamos todas y cada una de las decisiones de implementación.

PASO UNO: CONSTRUCCIÓN DEL DIAGRAMA PRINCIPAL

Siempre que se proporcione un comando de control al robot, el operador necesita conocer que es lo que el servicio Speech Recognition pudo interpretar. Para mostrar esta información sin desplegar un sin número de cuadros de dialogo, se puede utilizar un Flexible Dialog, este es un mecanismo útil de realimentación, a través del cual se conoce si el reconocimiento de voz está o no trabajando. En la figura 61 se puede observar la configuración del Flexible Dialog para el diagrama principal.

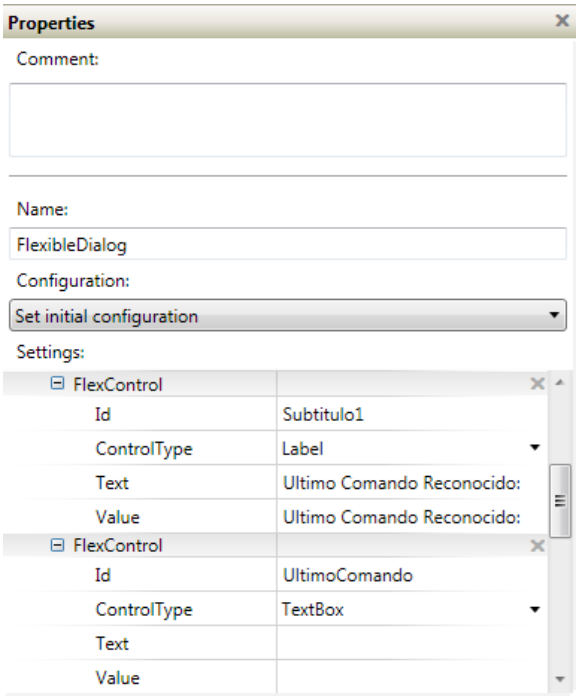


Figura 61. Configuración Flexible Dialog Diagrama Principal

En la figura anterior deberían aparecer veinte y cuatro controles sobre el Flexible Dialog, pero solo dos están visibles en la figura. El detalle del resto de controles está presente en el siguiente cuadro 4.

Cuadro 4

Configuración Flexible Dialog

Id	ControlType	Text	Value
Separador1	Separador		
Separador2	Label		
Titulo	Label	Departamento de Electrónica	Departamento de Electrónica
Separador3	Separador		
Separador4	Label		
Separador5	Separador		
Subtitulo	Label	Reconocimiento por voz	Reconocimiento por voz
Subtitulo1	Label	Ultimo comando reconocido	Ultimo comando reconocido
Ultimo comando	TextBox		
Subtitulo2	Label	Ejecución Autónoma de Comandos	Ejecución Autónoma de Comandos
Ejecución Autónoma de Comandos	TextBox		
Separador6	Label		
Separador7	Label		
Subtitulo3	Label	Bateria	
Subtitulo4	Label	Voltaje[mV]:	Voltaje[mV]:
VoltajeBateria	TextBox		
Subtitulo5	Label	Corriente[mA]	Corriente[mA]
CorrienteBateria	TextBox		
Subtitulo6	Label	Temperatura[ºC]	Temperatura[ºC]

CONTINUA →

TemperaturaBateria	**TextBox**		
Subtitulo7	Label	Carga[mAh]	Carga[mAh]
CargaBateria	TextBox		
Subtitulo8	Label	Capacidad[mAh]	Capacidad[mAh]
CapacidadBateria	TextBox		

Cuando se ejecute el diagrama, el servicio Flexible Dialog aparecerá como se indica a continuación. Los comandos verbales que son reconocidos son presentados como se muestra en la figura 62.

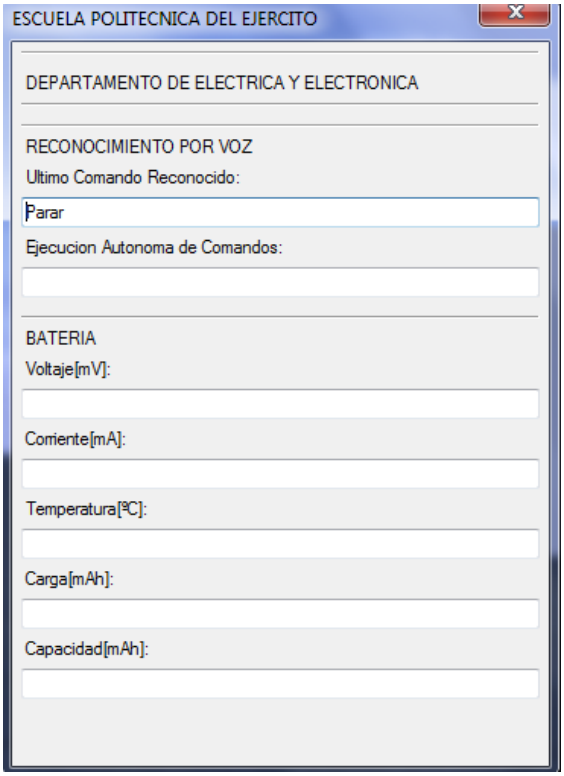


Figura 62. Presentación Flexible Dialog

Posterior a la configuración del Flexible Dialog se necesita identificar el comando de control suministrado por el operador para lo cual se tiene que agregar

un bloque **Calculate,** y conectarlo al puerto de notificación del **Speech Recognizer.** Dentro del bloque **Calculate** debemos seleccionar la tarea **Text.**

Realizamos otra conexión desde el pin de notificación hacia el Flexible Dialog seleccionando **UpdateControl** como el tipo de solicitud de mensajes a manejar y establecemos los parámetros en el cuadro de dialogo Data Connections, como se muestra en la figura 63.

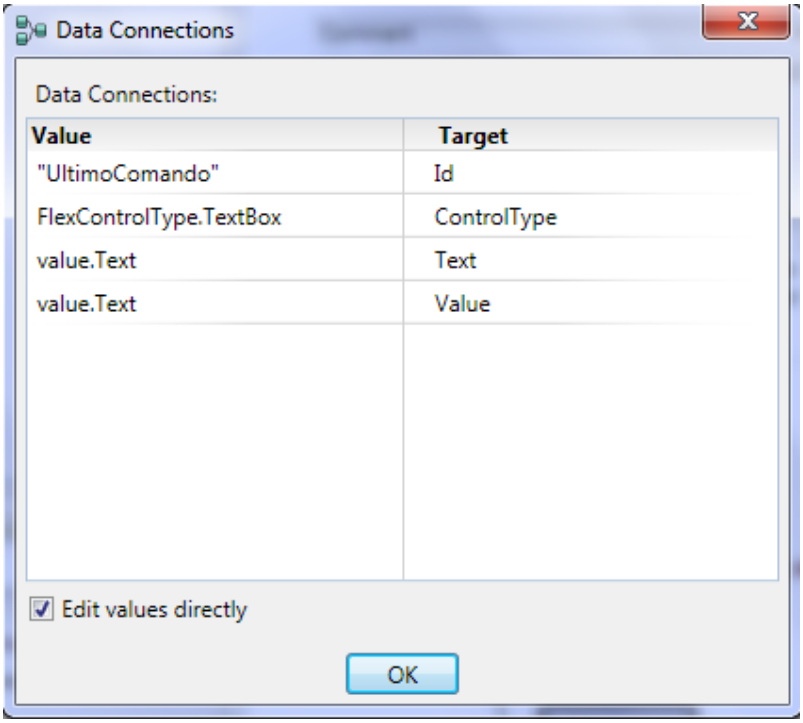


Figura 63. Flexible Dialog-Data Connections

Ahora se necesita procesar las tareas de control por lo que los comandos verbales que fueron reconocidos por el servicio Speech Recognition tienen que pasar a una nueva actividad personalizada llamada **ActividadCentral.** Renombramos su plantilla **Action** por **ACSeñaldeVoz** y definimos una entrada tipo string llamad**a ACSdVComandosInput** como se muestra en la figura 64.

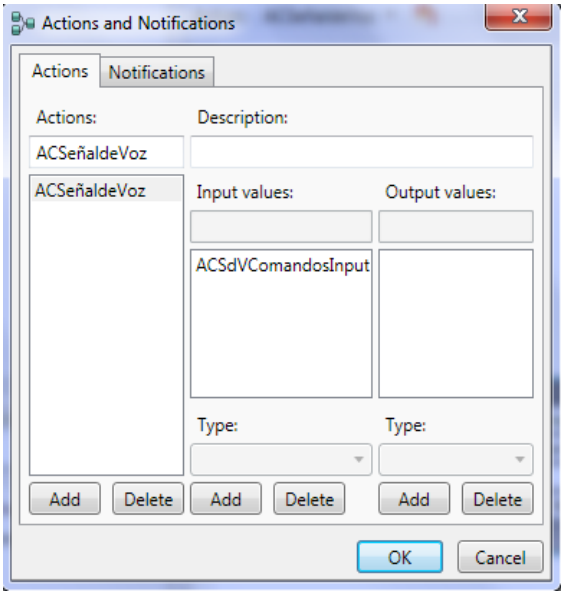


Figura 64. Configuración Actividad Central

El flujo de datos resultante entre las actividades y servicios para el Diagrama principal se muestran en la figura 65 este primer paso me va a permitir procesar las señales de control que el operador suministre al robot a través de un micrófono.

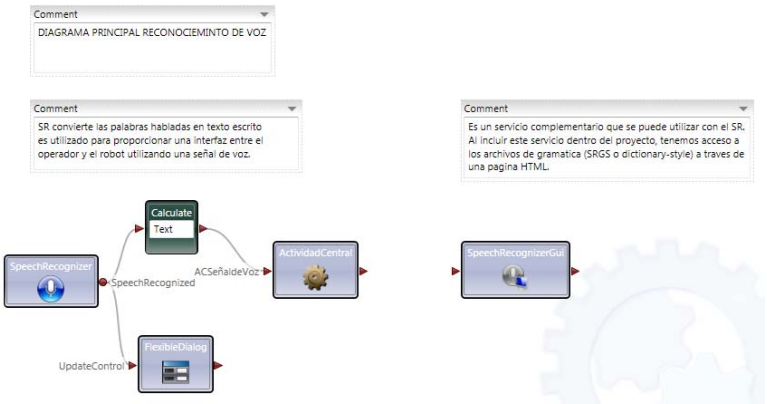


Figura 65. Comandos de procesamiento de Voz

PASO DOS: MANDO DE CONTROL VERBAL

La primera parte de la labor, es el mando de control verbal para el robot. Para lo cual se precisa añadir una nueva actividad personalizada en la plantilla **ACSeñaldeVoz** de la actividad **ActividadCentral** llamada **EjecucionAsincroma.** En esta actividad se necesita agregar una entrada llamada **EAEdCComandosInput** de tipo **string** que va ser quien contenga los comandos de control producto del reconocimiento de voz por parte del servicio **Speech Recognition.**

Una vez que se ha configurado la entrada de la actividad es momento de generar el código para la actividad. Todo lo que se necesita para esto es una declaración **If** que enlace la entrada **EAEdCComandosInput** y decida que comandos pasar al servicio **GenericDiffentialDrive** para que sean ejecutados, una herramienta importante para la ejecución de la tarea de rotación es el servicio WaitForDriveCompletion que permite establecer un tiempo de espera para que los comandos que ejecutan la rotación en el robot puedan completarse. De lo contrario, la siguiente acción se ejecutaría inmediatamente durante la reproducción de una de ellas.

Las salidas desde los bloques **Generic Differential Drive** tienen que ir fusionados y conectados a la salida de la actividad para que los comandos de control puedan ser ejecutados por el robot, de lo contrario solo se realizaría el procesamiento de la señal sin tener acceso directo a los servomotores del robot impidiendo así su desplazamiento. En la figura 66 se puede observar la actividad Ejecución Asíncrona completa.

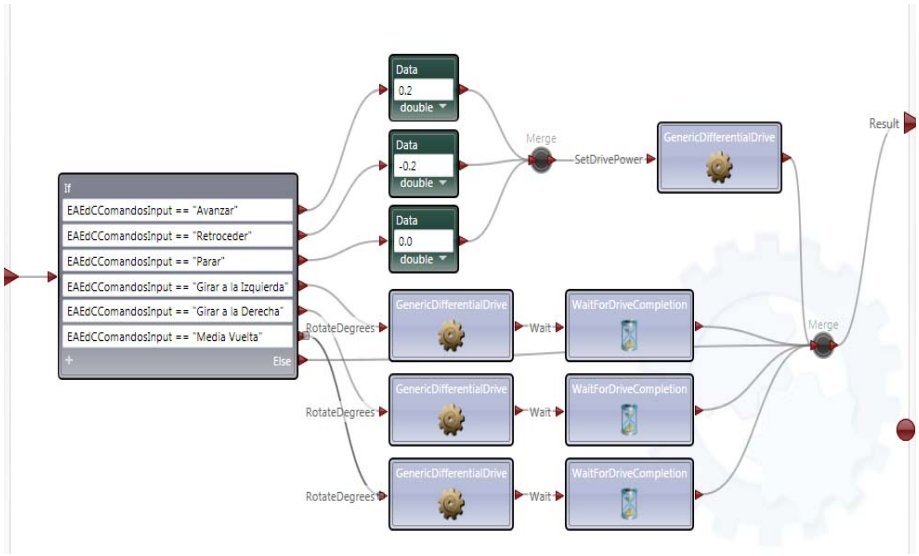


Figura 66. Ejecución Asíncrona – Diagrama completo

Los valores de rotación se pueden establecer dentro de un rango de 0º a 360º siendo por lo general los ángulos de 30º y 45º los más usados para hacer pequeñas rotaciones hacia la izquierda o la derecha. Sin embargo también se pueden establecer ángulos de 90º los cuales equivalen a ¼ de giro. El comando media vuelta tiene que contener una configuración de rotación de 180º. De la misma que se configuran las opciones de giro, hay que seleccionar una configuración adecuada para la potencia de alimentación de los dos motores de las ruedas del robot, permitiendo de esta manera que el robot se pueda mover ya sea para adelante o para atrás. A continuación se presenta el cuadro 5 donde se puede observar las características de configuración del servicio **GenericDifferentialDrive** para cada uno de los comandos de voz procesados.

Cuadro 5

Valores configuración servicio GenericDifferntialDrive

Comando	Operación	Potencia de Alimentación	Grados de Rotación
Avanzar	SetDrivePower	0,2	ND
Retroceder	SetDrivePower	-0,2	ND
Parar	SetDrivePower	0,0	ND
Girar a la Izquierda	RotateDegrees	0,2	45
Girar a la derecha	RotateDegrees	0,2	-45
Media Vuelta	RotateDegrees	0,2	180

Una vez finalizada la configuración es necesario establecer un **Manifest** (una lista de todos los servicios necesarios y sus socios) para dar soporte al bloque **GenericDifferentialDrive** antes de que la aplicación sea arrancada. **iRobot.Drive.Manifest.xml** o **iRobot.Create.Simulation.Manifest.xml** son dos tipos de **Manifest** que se puede utilizar en el diseño de la aplicación, el uno refiere a la utilización de la plataforma física iRobot Create y el otro a una simulación de ambiente robótica establecida previamente a traves de la herramienta Visual Simulation Environment de Microsoft Robotics Studio. Como punto de estudio y de pruebas inicialmente se trabajó bajo el **Manifest iRobot.Create.Simulation.Manifest.xml** en el cual se pueden implementar todos los algoritmos de control que se van diseñando sin temor a dañar la plataforma robótica.

Dentro de la plantilla **ACSeñaldeVoz** de la actividad **ActividadCentral** conectamos la actividad **EjecucionAsincrona** creada con anticipación a su pin de entrada y salida. Esto asegura que los valores que vienen desde la entrada de la actividad **ActividadCentral** pasen a la entrada **EAEdCComandosInput** dentro de la actividad **EjecucionAsincrona.**

Finalizando con la última tarea se puede arrancar el programa y probar el control verbal sobre el modelo simulado de la plataforma móvil iRobotCreate.

Al ejecutar el programa por primera vez se requiere ingresar los comandos que se pretende sean interpretados por el robot dentro de la gramática del servicio **SpeechrecognitionGUI** para que sean reconocidos cuando el **Flexible Dialog** se ejecute.

Para establecer la configuración inicial del servicio de reconocimiento de voz se tiene que abrir un navegador web e ingresar a la dirección URL http://localhost:50000/speechrecognizergui, este es un medio de acceso que me permite examinar servicios que han sido inicializados por el **DSSHost** que es el programa responsable de implementar el entorno de ejecución para MRDS.

Se debe agregar las siguientes frases al diccionario, ver cuadro 7, dejando el campo **Semantic Value** en blanco.

Cuadro 6

Lista de comandos ingresados en servicio SpeechRecognizerGUI

Avanzar
Retroceder
Parar
Girar a la Izquierda
Girar a la Derecha
Media Vuelta
Iniciar Entrenamiento
Detener Entrenamiento
Ejecutar Tarea

Al finalizar el ingreso de las frases, el diccionario en la URL debe observarse como en la figura 67.

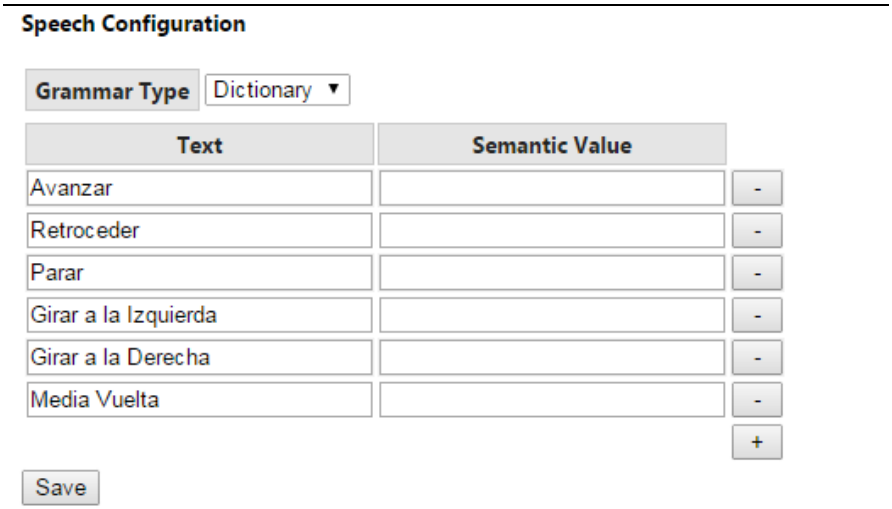


Figura 67. Configuración del reconocimiento de Voz

Al realizar unas pruebas de reconocimiento estas se verificaran si han sido identificadas con éxito dentro del **Speech Events** en el examinador de servicios URL, ver Figura 68.

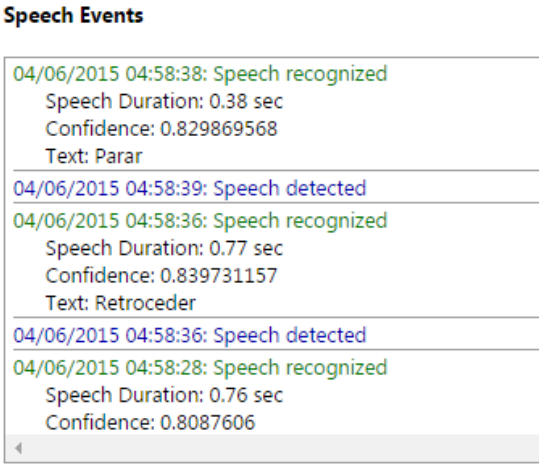


Figura 68. Identificador de Eventos de Reconocimiento de Voz.

PASO TRES: SEGUIMIENTO DEL ESTADO DEL ROBOT

Una vez que se ha dotado al robot de la capacidad de interpretar comandos de voz para ser controlado, el siguiente paso es permitir que el robot aprenda una tarea y pueda repetirla. Así en este orden lo que primero se necesita hacer es obtener el estado del robot.

Cuando el robot es instruido para ejecutar la tarea **Iniciar Entrenamiento,** lo que se necesita es actualizar su estado registrando que está ahora dentro del estado **Entrenamiento.** Si por lo contrario el operador suministra el comando **Detener Entrenamiento** cuando el robot está en el estado **Entrenamiento,** esto quiere decir que el robot ha finalizado el aprendizaje de una tarea y su estado en ese momento cambia a **FinEntrenamiento.**

Podría presentarse el caso en el cual entre los comandos **Iniciar Entrenamiento** y **Detener Entrenamiento** el operador no suministra ninguna otra instrucción por lo que si se manda a ejecutar la sentencia **Ejecutar Tarea** el programa está en la capacidad de hacerse cargo de esta particularidad.

Para ejecutar este paso se inicializa sobre la plantilla **Start** de la actividad **ActividadCentral** dos variables **Entrenamiento** y **FinEntrenamiento** con un valor de False, ver figura 69.

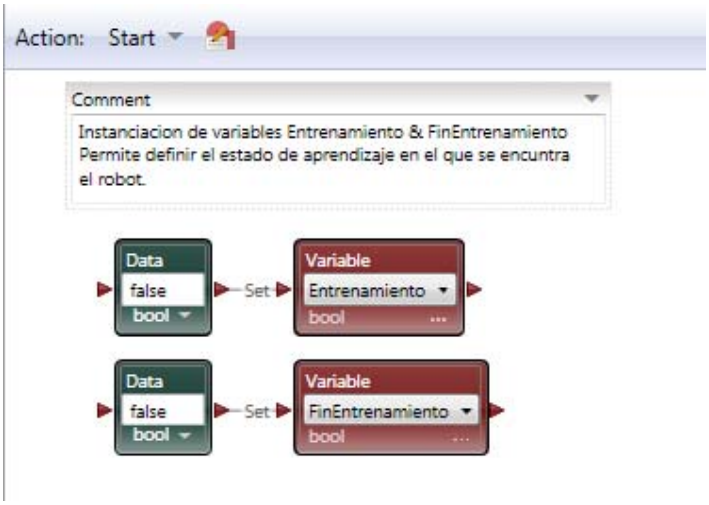


Figura 69. Iniciación Variables Entrenamiento y Fin Entrenamiento

Ahora se necesita actualizar estas dos variables en la plantilla **ACSeñaldeVoz,** según sus valores de entrada (**ACSdVComandosInput**) a la actividad **ActividadCentral.** Para lo cual se requiere agregar un bloque **If** y conectarla a la entrada de la acción, desconectando la actividad **EjecucionAsincrona** y reconectándola al pin de salida de la opción **Else** de la declaración **If.**

Se definen dos condiciones de evaluación dentro de la declaración **If:**

1era.- Si value.ACSdVComandosInput == "Iniciar Entrenamiento" entonces Entrenamiento == true

2da.- Si value.ACSdVComandosInput == "Detener Entrenamiento" && state.Entrenamiento entonces FinEntrenamiento == True y Entrenamiento == False.

Se requiere conectar el flujo de control de las dos condiciones con el flujo de control de la actividad **EjecucionAsincrona** que está conectada al pin result de la plantilla **ACSeñaldeVoz,** Ver Figura 70.

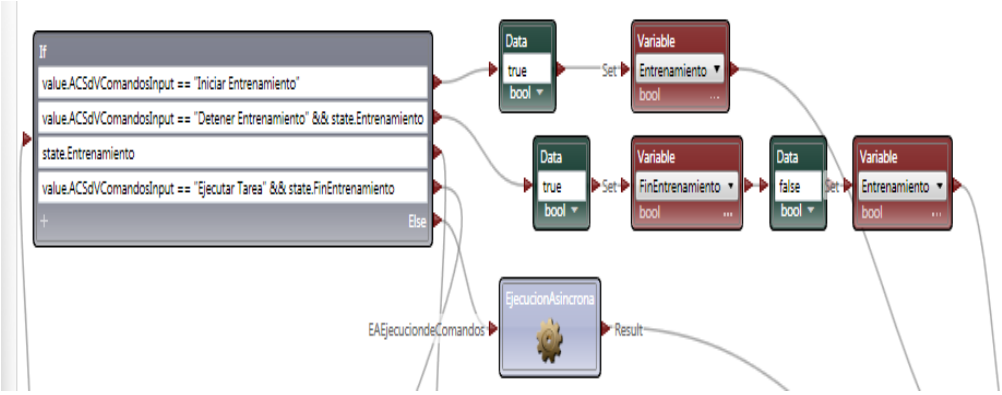


Figura 70. AC Señal de Voz –Diagrama Temporal

PASO 4: REGISTRO Y TEMPORIZACIÓN DE LOS COMANDOS DE VOZ

En este paso se trabajara sobre la parte del código donde el robot tiene que almacenar la secuencia de acciones que se necesita para aprender una tarea. Cada vez que el robot recibe una orden del operador este la ejecutara además de almacenarla y registrar el tiempo en el que el robot inicio la ejecución del comando.

Hay un sin número de maneras que se podrían elegir para estructurar esta parte del diseño, una forma es encapsular la parte del código que da instrucciones al robot para llevar acabo la acción y determinar el tiempo en el que el robot inicio la ejecución del comando, para ello se creara una actividad llamada **RegistrodeTiempo.** Esta actividad es muy similar a la actividad descrita anteriormente como **EjecucionAsincrona,** excepto que enviara a través de su pin de respuesta el comando de voz suministrado y el registro de tiempo obtenido.

Para esto se debe agregar una nueva actividad personalizada a la plantilla **ACSeñaldeVoz** de actividad **ActividadCentral** y nombrarla como **RegistrodeTiempo,** para empezar se necesita crear una plantilla Action llamada

RdTEjecucionActividad y añadir una entrada de tipo **string** nombrándola como **RdTEAComandosInput** y cuatro valores de salida **Hora, Minuto, Segundo** y **Milisegundo,** como se indica en la Figura 71**.**

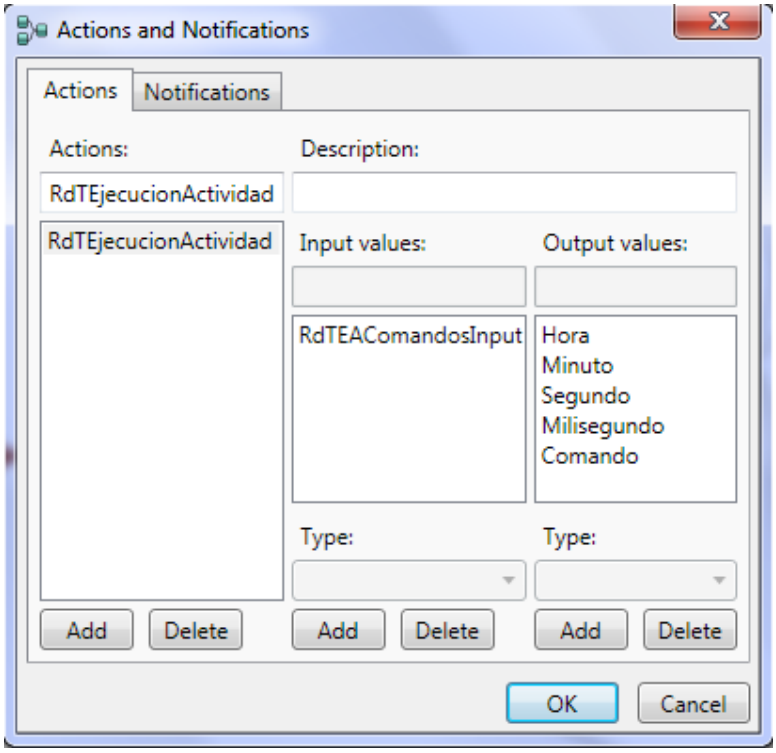


Figura 71. Configuración actividad Registro de Tiempo

A continuación se requiere añadir la actividad **EjecucionAsincrona** a la actividad **RegistrodeTiempo** y conectar sus pines de entrada.

La siguiente tarea consiste en obtener el tiempo de ejecución de cada uno de los comandos de voz para lo cual se debe conectar la salida de **EjecucionAsincrona** a un servicio **Timer** y en el cuadro de dialogo **Conections** se tiene que elegir la operación **GetCurrentTime.** El tiempo actual se encuentra disponible mediante la conexión a la salida del **Timer.** En VPL no existe una actividad **List** que contenga un tipo de dato **Time** para poder registrar en una lista los elementos hora, minuto, segundo y milisegundo del tiempo de ejecución de los

comandos de voz. Pero al ser estos componentes todos del tipo da dato **int** pueden ser almacenados dentro de las Listas.

Para realizar esta tarea se necesita conectar cuatro bloques **Calculate** al **Timer** para extraer cada uno de los componentes del tiempo (hora, minuto, segundo y milisegundo). Al finalizar se debe pasar toda la información de registro y temporización de los comandos de voz suministrados por el operador como parte del resultado de la acción, combinando todos los valores en una actividad **Join** y conectándola al pin R**esult** de la actividad **RegistrodeTiempo,** ver Figura 72**.**

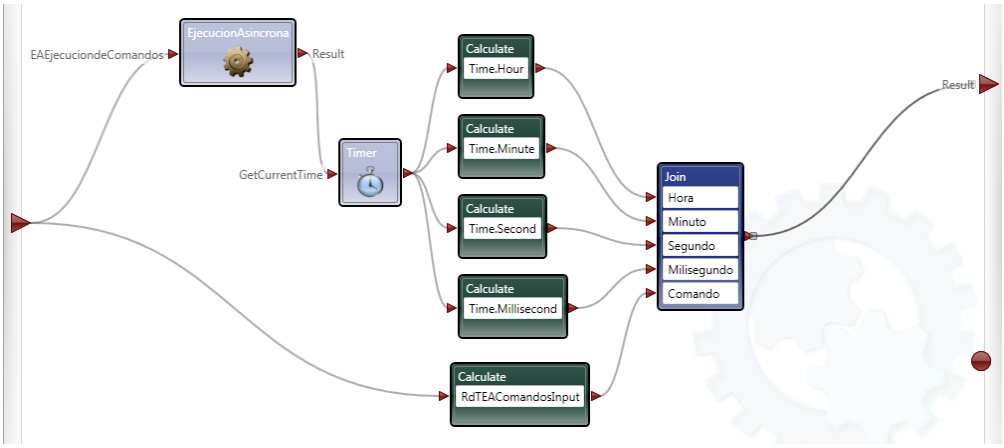


Figura 72. Actividad Registro de Tiempo – Diagrama Completo

Cuando se conecta al pin acción **Result** desde el **Join** hay que asegurar que se han conectado correctamente cada una de las salidas. Una vez que se ha verificado esto la actividad está completa, ver Figura 73.

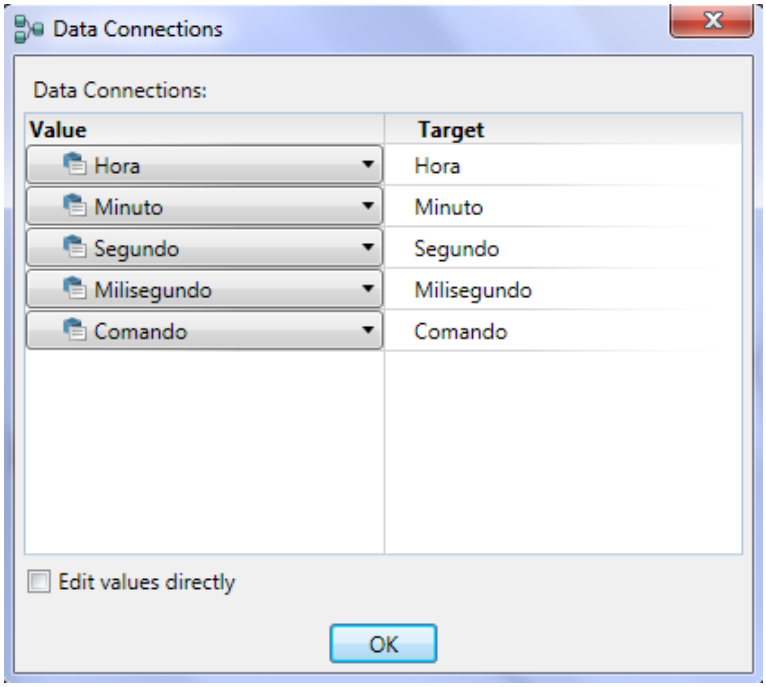


Figura 73. Conexión de datos Join – Pin Result Actividad Registro de Tiempo

PASO CINCO: ALMACENAMIENTO DE ACCIONES Y TIEMPOS

Cuando el robot recibe los comandos en el estado **Entrenamiento,** este necesita almacenarlos, para que posteriormente puedan ser ejecutados secuencialmente. El robot también requiere almacenar el tiempo que cada comando estuvo en operación para que pueda realizar correctamente la tarea, para llevar este paso a cabo se necesita hacer uso de la actividad **List** para almacenar esta información.

Para almacenar los comandos se requiere definir dentro de la plantilla **Start** de la actividad **ActividadCentral** una variable tipo **List,** por lo que se agrega un bloque **List** que genera una lista vacía de elementos de datos, para crear una lista se debe seleccionar el tipo de dato para los elementos en este caso debe ser de tipo **string** y se lo debe conectar a un bloque **Variable** definiendo su tipo de dato como **List of String,** ver Figura 74.

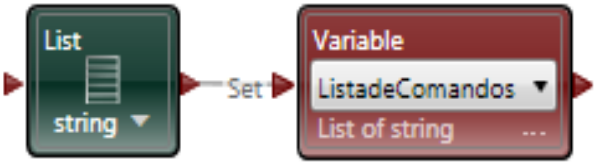


Figura 74. Inicialización Lista Almacenamiento Comandos

También es necesario crear listas de las horas, minutos, segundos y milisegundos, estas variables deben ser del tipo **List of Int** de esta manera se registrara la declaración de cada movimiento, ver Figura 75.

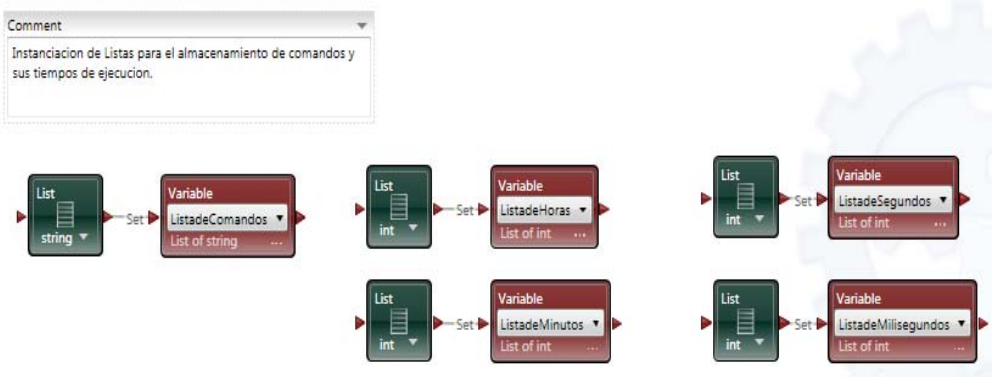


Figura 75. Inicialización de Variables y listas – Actividad Central

El siguiente paso es escribir el código para agregar a las listas los comandos que el robot ejecute y el tiempo que cada uno de estos esté en operación dentro del evento **Entrenamiento.** Para ello se necesita agregar una nueva actividad personalizada dentro de la plantilla **ACSeñeldeVoz** de la actividad **ActividadCentral** llamada **Memoria.** Los valores de entrada a esta actividad deben de ser cada una de las listas inicializadas en la plantilla **Start** de la actividad **ActividadCentral** así como un elemento para anexar a cada una de las listas. El valor de salida es una nueva lista con cada uno de los elementos almacenados, ver Figura 76.

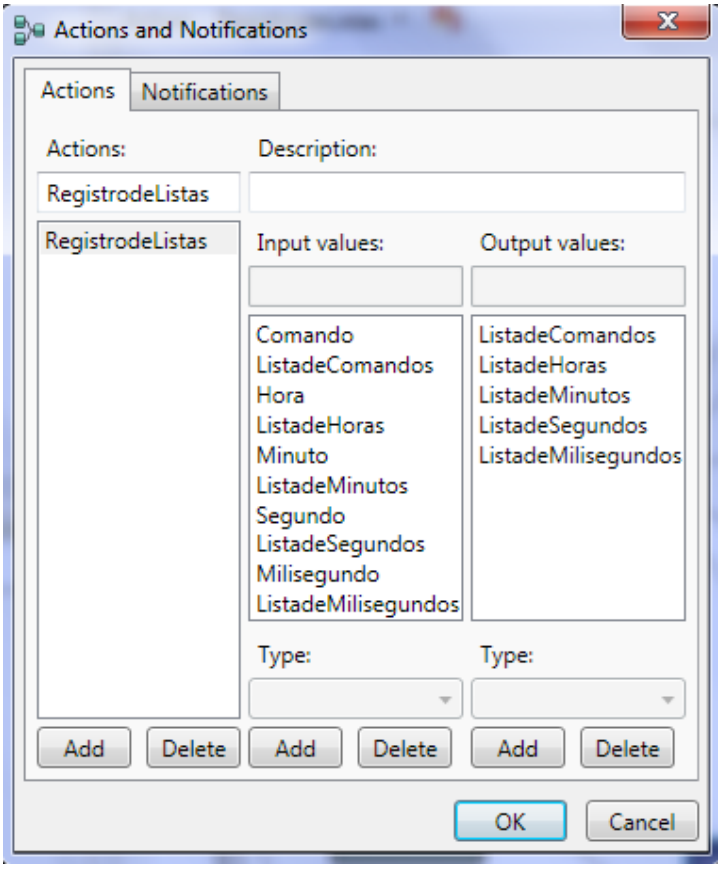


Figura 76. Valores de Entrada y Salida – Actividad Memoria

Para realizar este proceso se debe hacer uso de la actividad **ListFunctions** que recepta una variable **List** y un elemento a añadir y los anexa a través de la operación **Append.**

Cabe mencionar que una actividad **ListFunctions** actúa como si fuera un bloque **Join,** el flujo de control debe alcanzar el bloque **ListFunctions** en ambas entradas antes de que continúe. En la Figura 77 se puede observar el diagrama completo de la actividad **Memoria.**

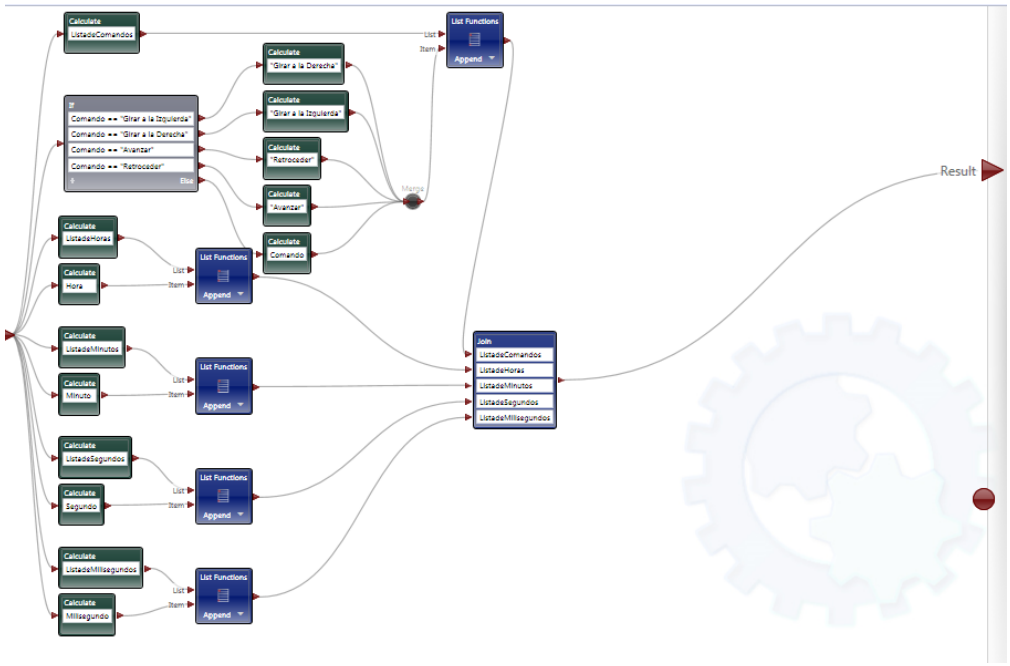


Figura 77. Diagrama completo Actividad Memoria

PASO SEIS: CONTROLADOR DE COMANDOS EN EL ESTADO ENTRENAMIENTO.

Ahora se puede usar las dos actividades **RegistrodeTiempo** y **Memoria** para el caso en el que el robot recibe un comando cuando está en el estado **Entrenamiento.**

Para empezar se debe añadir otra condición (**state.Entrenamiento**) a la declaración **If** de la actividad **ActividadCentral.** Si la condición es verdadera entonces se llama a la plantilla **RdTEjecucionActividad** en la actividad **RegistrodeTiempo** para pasar los datos de voz registrados en el valor de entrada **ACSdVComandosInput** de la actividad **ActividadCentral.** La actividad **RegistrodeTiempo** retornara los registros de tiempo**, Hora, Minuto, Segundo** y **Milisegundo** en los valores de salida de dicha actividad así como el comando suministrado por el operador. Estos valores se tienen que transferir a la actividad

Memoria, por lo que se tienen que conectar estas dos actividades. El cuadro de dialogo **Data Connections** para este flujo de datos se muestra en la Figura 78.

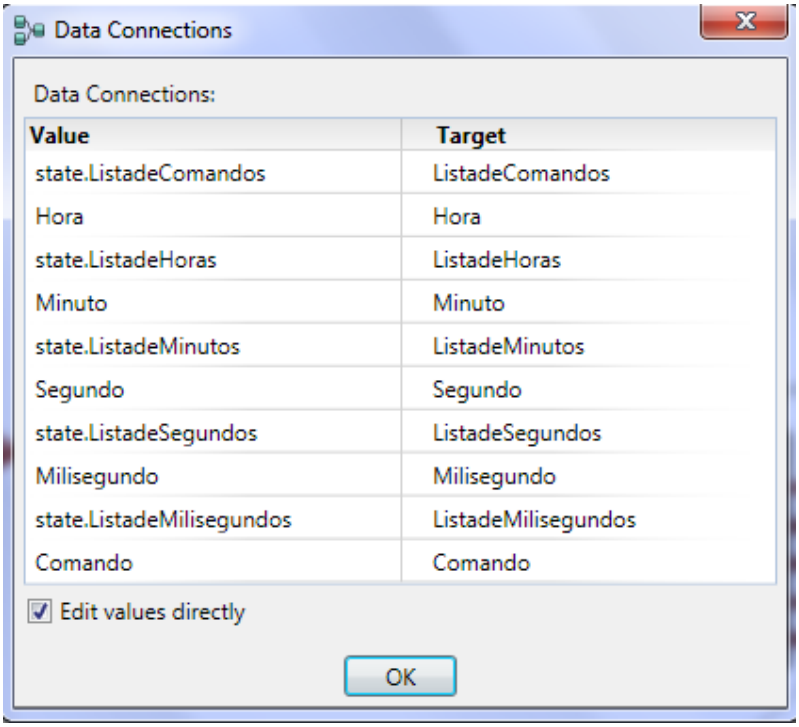


Figura 78. Conexión de datos entre actividades Registro de Tiempo y Memoria

La actividad **Memoria** retorna nuevas listas, por lo que se necesita asignarlas a las variables tipo **List** y luego establecer el flujo de datos de cada una en un **Join** para conectar su salida a un **Merge** y posteriormente al pin **Result** de la actividad **ActividadCentral,** de esta manera se habrá terminado de realizar el ***Controlador*** para el caso en el que el robot reciba un comando mientras se encuentra dentro del estado **Entrenamiento,** ver Figura 79.

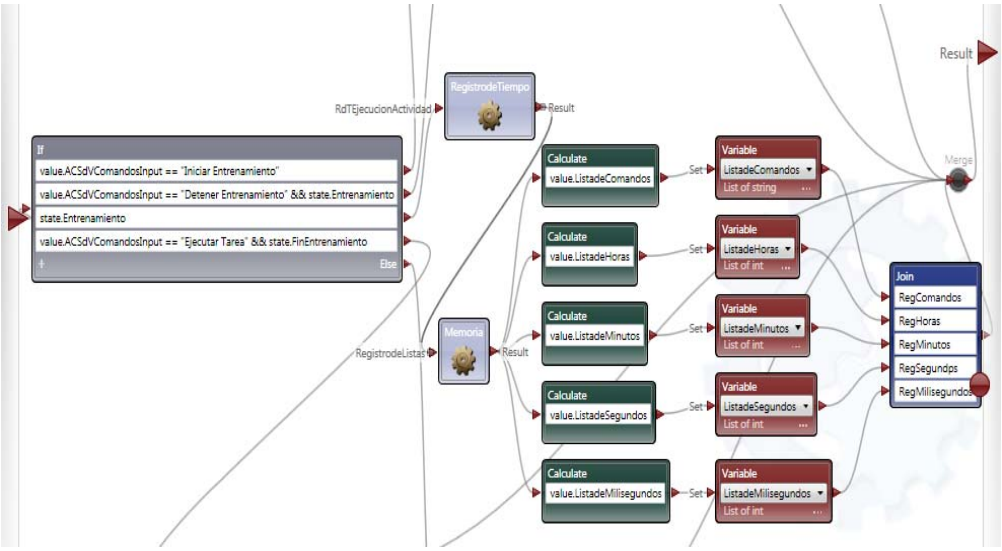


Figura 79. Controlador de Comandos en el estado Entrenamiento

PASO SIETE: CREACIÓN ACTIVIDAD "EJECUCIONAUTONOMA"

El comando final al cual el robot debe responder es el comando **Ejecutar Tarea.** En otras palabras se necesita escribir el código para llevar a cabo la secuencia de acciones que el robot ha aprendido.

Para esto en la plantilla **ACSeñaldeVoz** de la actividad **ActividadCentral** se debe añadir a la declaración **If** la condición **value.ACSdVComandosInput == "Ejecutar Tarea" && state.FinEntrenamiento.** A continuación se necesita crear una nueva actividad y nombrarla como **EjecucionAutonoma.** Esta actividad no tiene salidas, pero requiere cada una de las listas como entradas, una vez generada la nueva actividad se requiere conectar su entrada a la nueva condición **If** y su salida al **Merge.** El diagrama completo de la actividad **ActividadCentral** se muestra en la Figura 80.

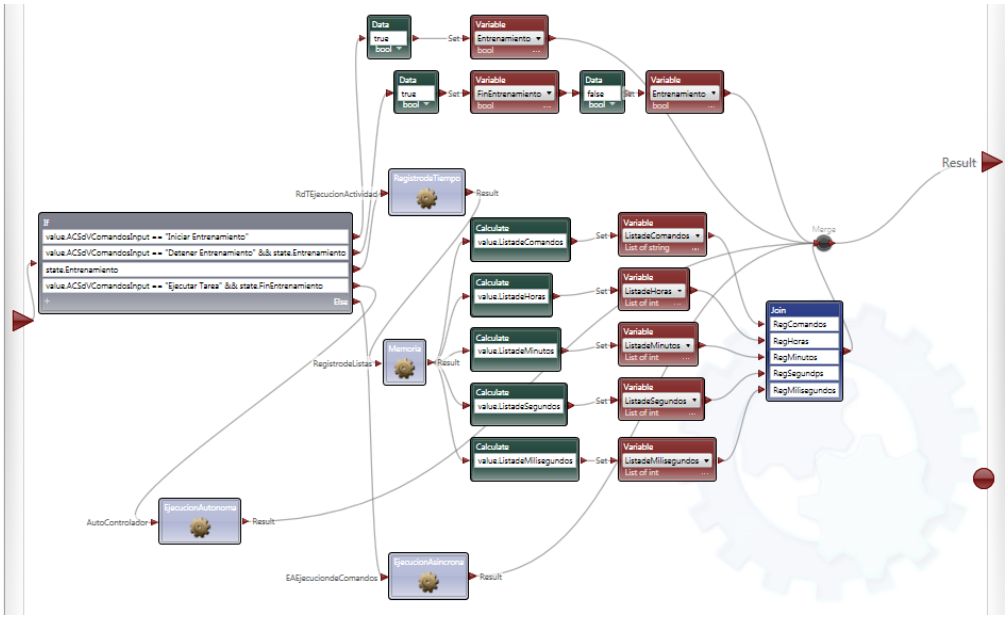


Figura 80. Diagrama Completo Actividad Central

PASO 8: CREACIÓN ACTIVIDAD "DESCOMPOSICIONDELISTAS"

Se necesita algo de código auxiliar para escribir la actividad **EjecucionAutonoma.** El argumento en la actividad **EjecucionAutonoma** puede ser escrita como un método recursivo. La idea tras el concepto de recursividad es pasar los argumentos de las listas de comando e información de tiempo. Este proceso comienza ejecutando el ítem ubicado en la primera posición de la lista, después de esperar una cantidad apropiada de tiempo volverá a repetirse el proceso, en cada iteración se genera una nueva lista sin el primer elemento extraído al momento de su ejecución así el proceso se repetirá hasta que las listas lleguen a estar vacías.

Para desarrollar este paso se tiene un nueva actividad personalizada llamada **DescomposiondeListas** dentro de la actividad **EjecucionAutonoma.** La entrada y salida de esta actividad son las listas que se necesitan extraer.

Para extraer un ítem de una lista se puede usar una actividad **ListFunctions** y su operación **RemoveItem.** Esta función requiere de dos entradas una **List** y un índice que especifica la posición del ítem (este tiene que ser de tipo **int).** Puesto que lo que se requiere es extraer el primer elemento de la cada lista, el índice que debe ingresar en la **ListFunctions** es cero, ver Figura 81.

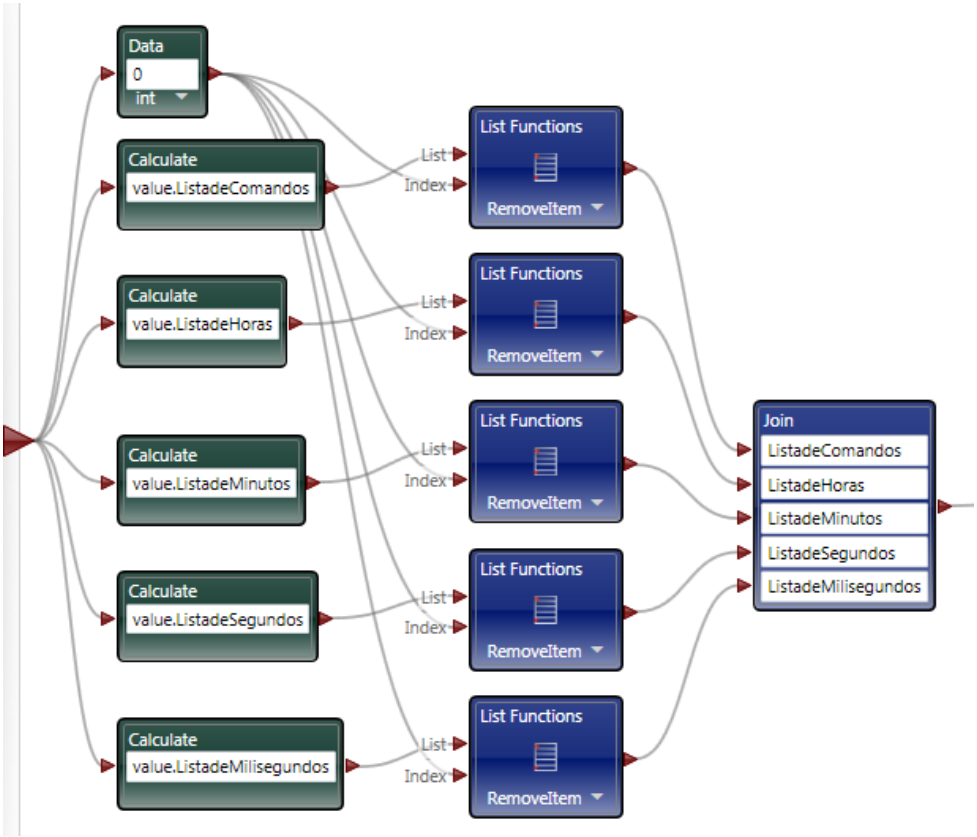


Figura 81. Actividad Descomposición de Listas

PASO NUEVE: CALCULAR LA DIFERENCIA DE TIEMPO

Si dentro de la actividad **EjecucionAutonoma** el comando actual en la posición cero de la lista no es **Avanzar** o **Retroceder,** simplemente se hace un llamado a las actividades **EjecucionAsincrona, DescomposiciondeListas** y **EjecucionAutonoma** (Recursividad). Sin embrago si el comando es **Avanzar** o

Retroceder se necesita calcular la cantidad de tiempo entre el momento en el cual cualquiera de estos dos comandos y el siguiente en la lista fueron ejecutados. Esta es la cantidad de tiempo de espera antes de que se ejecute la recursión.

Para ejecutar esta tarea se necesita crear una nueva actividad llamada **CalculoDifTiempo,** los valores de entrada de esta actividad son el tiempo de inicio y fin en el que el comando estuvo en ejecución (en términos de los componentes hora, minuto, segundo y milisegundo) y el valor de salida es la diferencia de tiempo en milisegundos, ver Figura 82 y Figura 83.

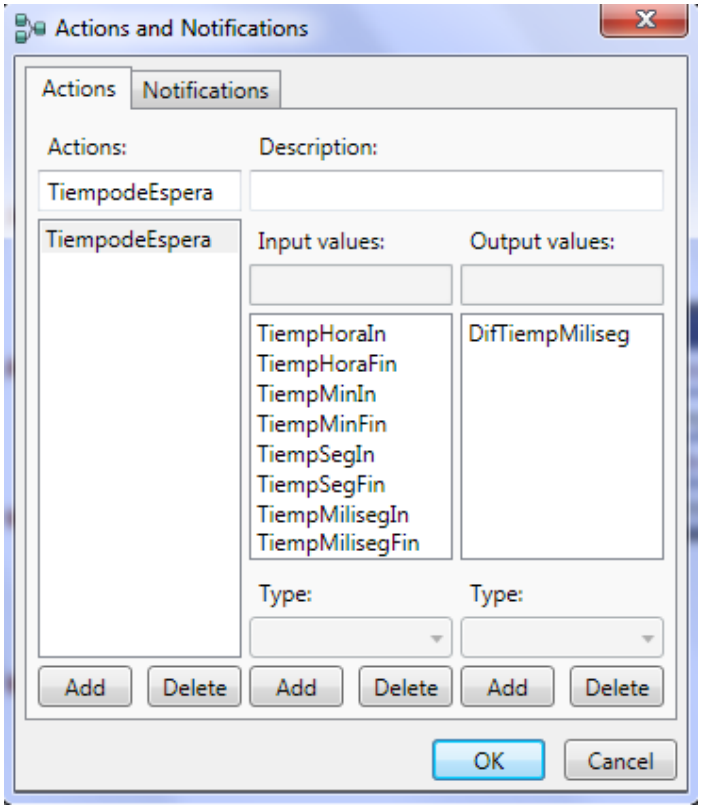


Figura 82. Valores de entrada y salida Actividad CalculoDifTiempo

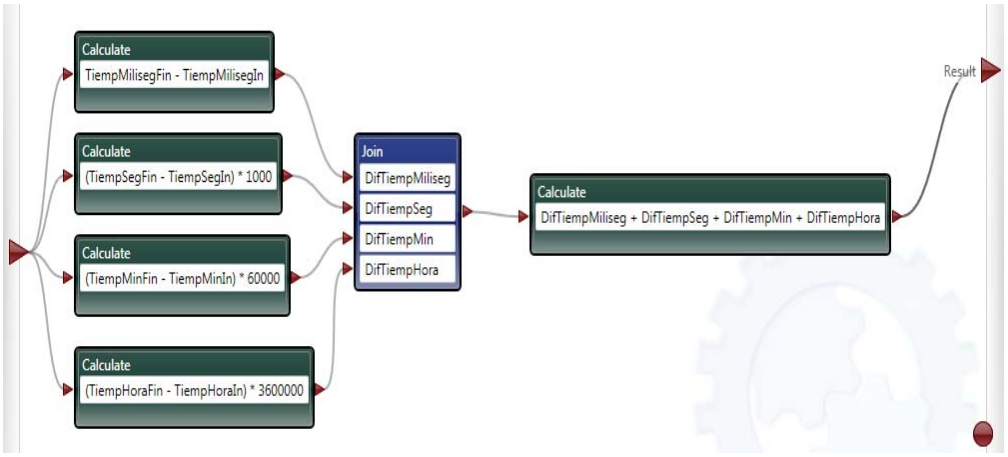


Figura 83. Actividad CalculoDifTiempo completa

PASO DIEZ: ORQUESTACIÓN DE ACTIVIDADES USANDO RECURSIÓN

Las siguientes instrucciones permitirán la construcción de la última parte del diagrama. Sin embargo hay algunos puntos clave que se debe tomar en cuenta. A la entrada de la platilla **AutoControlador** de la actividad **EjecucionAutonoma** del diagrama se establece la condición y la prueba que determina la finalización de la recursividad a través de una condición **If** cuando no hay más elementos en las listas. Cuando esto ocurre, un mensaje es enviado al **FlexibleDialog** especificando que la ejecución de las tares ha terminado "***Ejecucion de Tareas Completada***". Sin embargo si aún hay elementos dentro de las listas para su ejecución, el comando actual se muestra en el **FlexibleDialog** y es procesado.

En la parte superior del diagrama se define una nueva condición **If** si el comando es **Avanzar** o **Retroceder** este código debe ejecutar el comando, iniciar un temporizador y a continuación ejecutar el siguiente comando cuando el temporizador expire. Finalmente dentro de este flujo de datos se llama una vez más a la actividad **EjecucionAutonoma** dentro de sí misma (técnica de programación conocida como ***Recursión***), la cual permite ejecutar cada uno de los

comandos presentes dentro de las listas hasta que estas quedan vacías. Para asegurar que el comando que acaba de ser ejecutado es eliminado, se llama a la función **DescomposiciondeListas.** La actividad **Join** asegura que ningún flujo de datos pasa hasta que el servicio **Timer** haya finalizado y los elementos de las listas hayan sido extraídos.

La sección media del diagrama controla la opción **Else** de la sentencia **If,** donde el comando suministrado por el operador no es la sentencia **Avanzar** o **Retroceder.** Tal como en el caso anterior las listas tienen que ser extraídas y el comando debe ser ejecutado, en este caso no existe ningún temporizador (**Timer**). En la Figura 84 Se puede observar el diagrama completo de la Actividad **EjecucionAutonoma.**

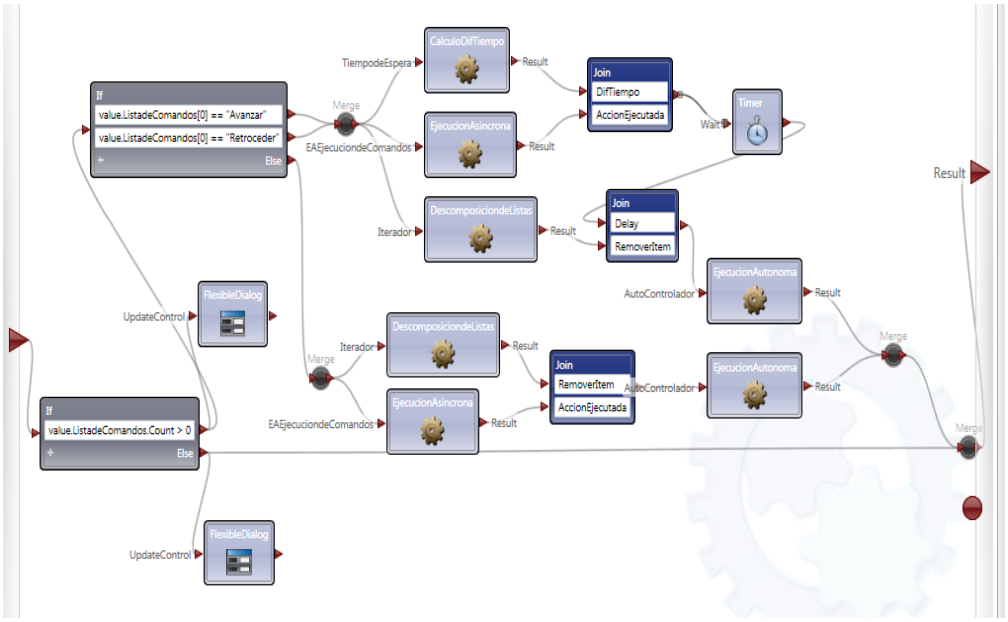


Figura 84. Actividad EjecuciónAutónoma – Diagrama Completo

PASO ONCE: IMPLEMENTACIÓN COMPORTAMIENTO REACTIVO Y MONITOREO DEL CONSUMO DE ENERGÍA DE LA PLATAFORMA.

Como parte del desarrollo de este proyecto dentro de la actividad **ActividadCentral** se ha implementado un comportamiento reactivo que trabaja conjuntamente con el sistema de navegación por voz, y se ha establecido un flujo de datos para monitorear el consumo de energía de la plataforma móvil.

Para diseñar la primera tarea se ha hecho uso del servicio **GenericContactSensor,** que dispone de una serie de operaciones que permiten conocer el estado de los sensores de contacto (bumpers) del iRobot Create, se conecta el pin de notificación con una actividad **Calculate** mediante la operación **ContactSensorUpdate, Calculate** entregara una valor booleano de true o false, el cual se ha de almacenar en la variable **BumperCreate.** A continuación se valida la señal a través de una condición **If,** si BumperCreate == true, entonces el sensor de contacto ha sido accionado por un obstáculo presente en el camino y la señal pasa a una actividad **Date** que la asocia con un valor tipo **String** denominado **BumperOn,** este estado se notifica a través del servicio **TextToSpeech (**servicio diseñado para proveer a la aplicación de una interface verbal) y **FlexibleDialog.** El flujo de datos de la activación del servicio **GenericContactSensor** se transmite hacia la actividad **ActividadCentral** y posteriormente hacia el valor de entrada de la plantilla **EAEjecuciondeComandos** de la actividad **EjecucionAsincrona** para ser evaluado por la condición **If** y establecer un valor de cero en cada uno de los motores del robot deteniendo por completo el movimiento, ver Figura 85.

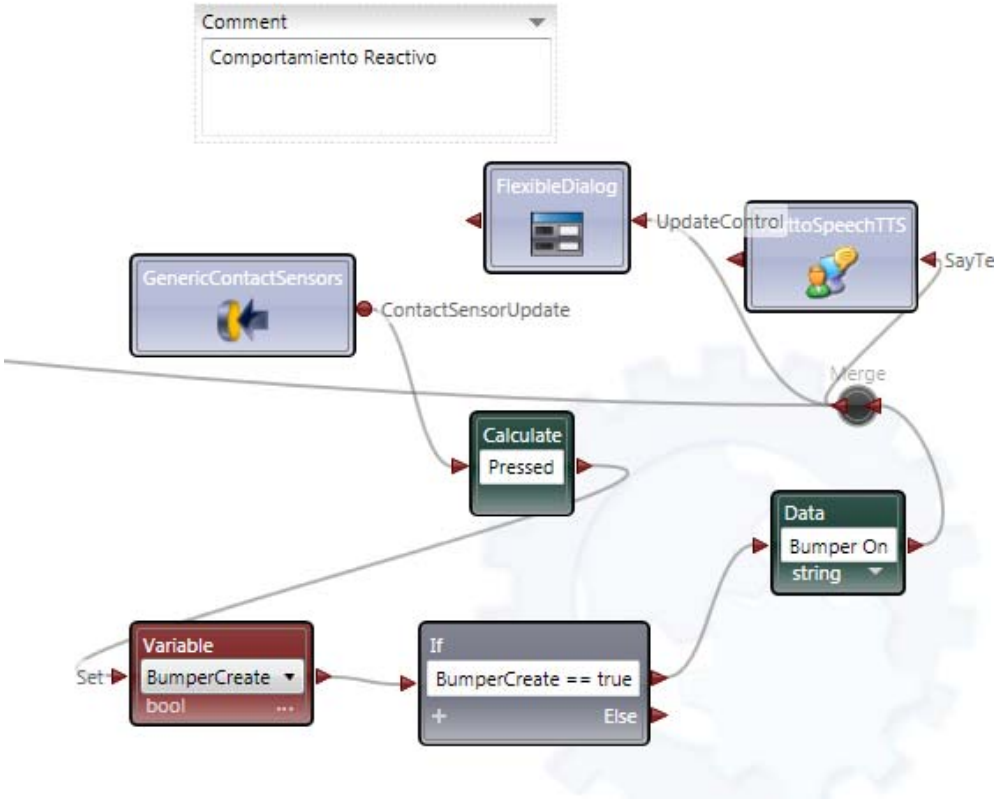


Figura 85. Implementación Acción Reactiva

La segunda parte del proceso se define insertando un servicio **iRobotCreateRoomba** y enlazándolo con un bloque **Calculate** a través de la operación **UpdateControl,** la cual tiene acceso directo a las notificaciones de los estados: Voltage, Current, Temperature, Charge y Capacity. Cada uno de estos estados se presenta al usuario a través del servicio **FlexibleDialog,** ver Figura 86**.**

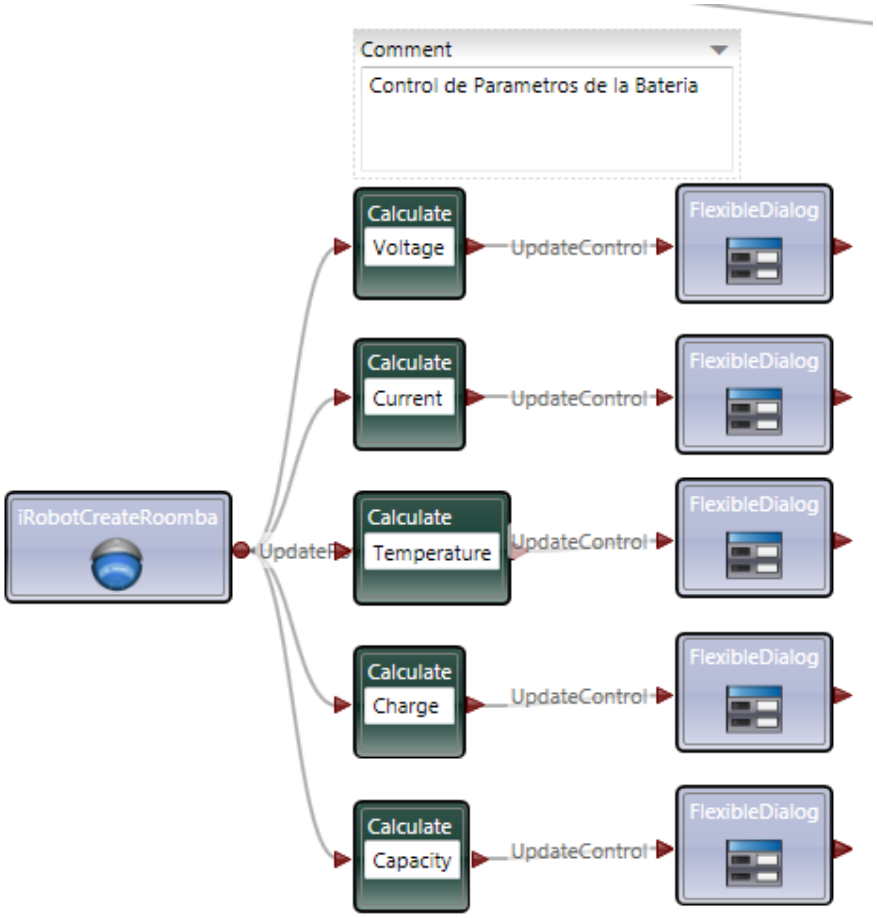


Figura 86. Monitoreo Consumo de Energía Plataforma

4.4.FUNCIONAMIENTO DE LA APLICACIÓN

El proyecto consiste en manipular el IRobot Create mediante la voz para que ejecute instrucciones, que se definirán en el programa Microsoft Robotics Developer Studio, mediante el Lenguaje VPL (Visual Programming Lenguage), esto siempre siguiendo un modelo de programación basada en una Arquitectura de Control Hibrida. El reconocimiento de voz se hará a través del Software Speech Recognition. El robot podrá realizar movimientos hacia delante, atrás, izquierda, derecha, media vuelta y detenerse, además está en la capacidad de aceptar comandos tales como iniciar, detener y ejecutar un entrenamiento básico de acuerdo a unos datos establecidos que fueron guardados en un colector de palabras, que servirá como variable entrante al programa y este se encargará de

validarla, procesar y enviar las instrucciones adecuadas al robot a través del protocolo de comunicación IEEE 802.15.1 o mejor conocido como Bluetooth.

El siguiente cuadro esboza el problema de la tarea de aprendizaje, el robot debe estar en la capacidad de ejecutar los siguientes comandos de control presentes a continuación.

Cuadro 7

Cuadro de Acciones - Plataforma móvil

Avanzar	Establece la operación DrivePower del servicio Generic Differential Drive de ambas llantas del robot en 0.2
Retroceder	Establece la operación DrivePower del servicio Generic Differential Drive de ambas llantas del robot en -0.2
Parar	Establece la operación DrivePower del servicio Generic Differential Drive de ambas llantas del robot en 0.0
Girar a la Izquierda	Hace que el robot gire 45º a la izquierda
Girar a la Derecha	Hace que el robot gire 45º a la derecha
Media Vuelta	Hace que el robot gire 180º
Iniciar Entrenamiento	Cuando el robot entra en este estado, registra los comandos suministrados por el operador para ser ejecutados posteriormente
Detener Entrenamiento	Este comando detiene el registro de instrucciones por parte del robot
Ejecutar Tarea	Si el robot ha aprendido una tarea y el comando Detener Entrenamiento ha sido ejecutado, el robot deberá responder a este comando para ejecutar las acciones aprendidas.

El objetivo fundamental de este proyecto es que el ser humano sea capaz de controlar verbalmente el robot, como tal este siempre deberá ejecutar cualquier comando de control que sea declarado por el operador.

CAPITULO 5

PRUEBAS DE OPERATIVIDAD Y RESULTADOS

5.1 DESCRIPCIÓN DE ENTORNO DE PRUEBAS

Las pruebas de operatividad y funcionamiento se las desarrollo en dos ambientes de trabajo uno real y el otro simulado haciendo uso del simulador Visual Simulation Environment de Microsoft Robotics Developer Studio (MRDS), ver Figuras 87 y 88.

Figura 87.Pruebas de Funcionamiento Ambiente Real

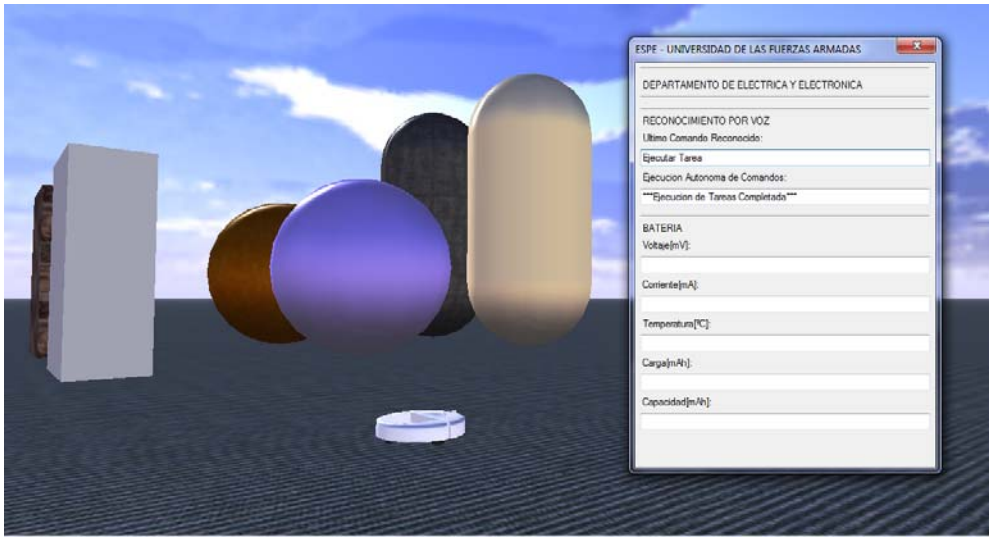


Figura 88. Pruebas de Funcionamiento Ambiente Real

5.2 PRUEBAS REALIZADAS

Para enviar las órdenes al robot se utilizó un micrófono estéreo con auriculares incorporados conectados a la tarjeta de sonido de la PC. Este dispositivo es el que mejor se adapta al proceso de reconocimiento por voz realizado por MRDS.

Debido a que durante todo el desarrollo del proyecto se utilizó una programación orientada al flujo de datos (VPL) lo que me permite manejar un desarrollo de código modular y escalable es aquí donde vamos a definir dos etapas para realizar las pruebas de funcionamiento de la plataforma.

La **primera etapa** de evaluación la vamos a definir en la identificación de los comandos AVANZAR, RETROCEDER, GIRAR A LA IZQUIERDA, GIRAR A LA DERECHA, PARAR y MEDIA VUELTA, estableciendo para el análisis la tasa de aciertos que cada palabra puede tener dada por la siguiente formula.

$$Tasa\ de\ Aciertos = \frac{Nº\ de\ Aciertos}{Nº\ de\ Casos} * 100$$

La **segunda etapa** del análisis está enfocada en determinar el porcentaje de efectividad al momento de ejecutar un tarea de aprendizaje, el robot tiene que ser capaz de imitar los movimientos que el operador le instruyo.

Para cada uno de estos análisis se tomó en cuenta varios factores que podrían influir en el reconocimiento de voz como lo son el género del operador, el nivel de ruido presente en el ambiente, y la hora a la cual se realizaron las pruebas, ya que no es lo mismo poner aprueba el sistema en el día, cuando existe un mayor nivel de ruido en el ambiente, a que en la noche cuando los niveles de ruido son bajos. En el siguiente cuadro se muestran las consideraciones en la toma de datos.

Cuadro 8

Variables de muestreo

GENERO	HORA	MUESTRAS/COMANDO	RUIDO AMBIENTE [DB]
Hombre	Mañana	5	35 – 45 [dB]
Hombre	Noche	5	35 – 45 [dB]
Mujer	Mañana	5	80 – 95 [dB]
Mujer	Noche	5	80 – 95 [dB]

Todas las pruebas fueron realizadas en un entorno de interiores y bajo un ambiente estable y controlado. El rango de variación del ruido ambiental fue tomado usando la aplicación Sound Meter de Google Play. A continuación en las figuras 89 y 90 se registran algunos datos de muestreo de la fecha en que se realizaron las pruebas de operatividad.

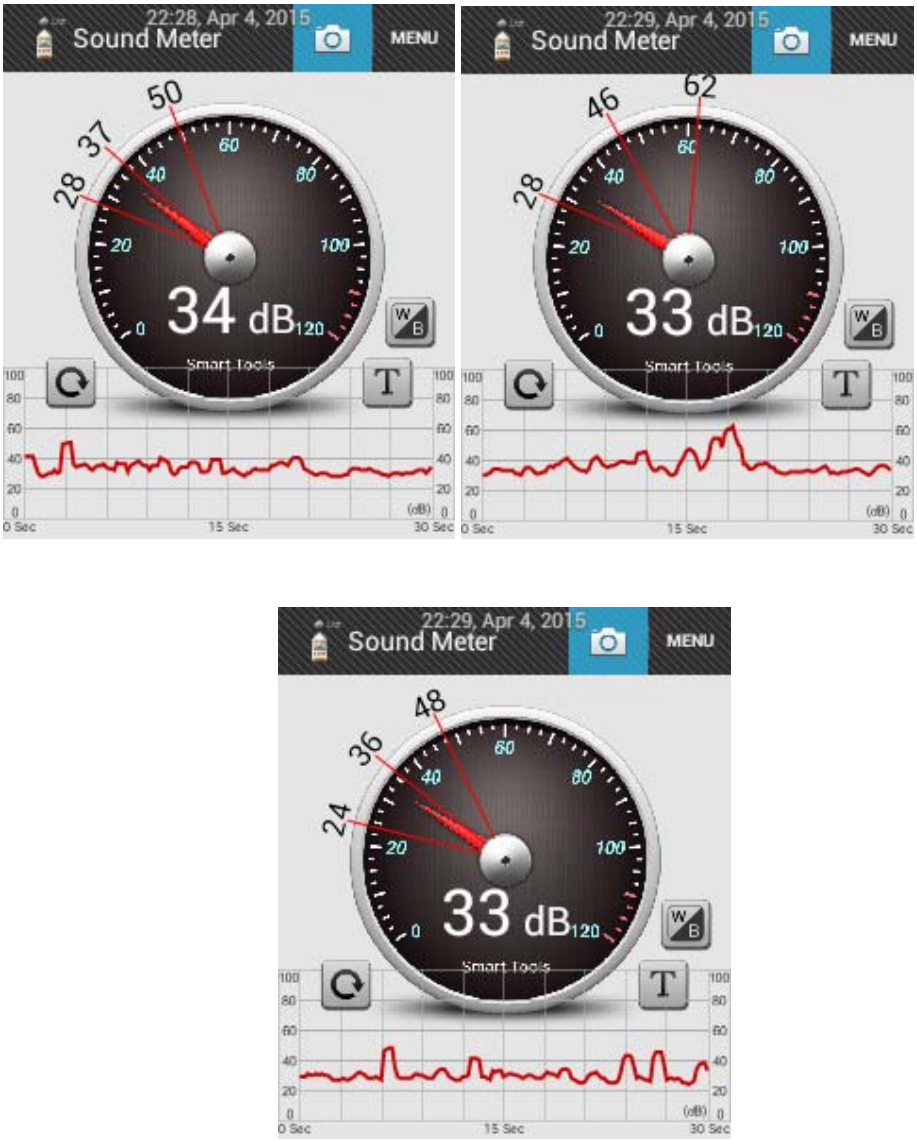


Figura 89. Toma de datos - Ruido Ambiental Mañana

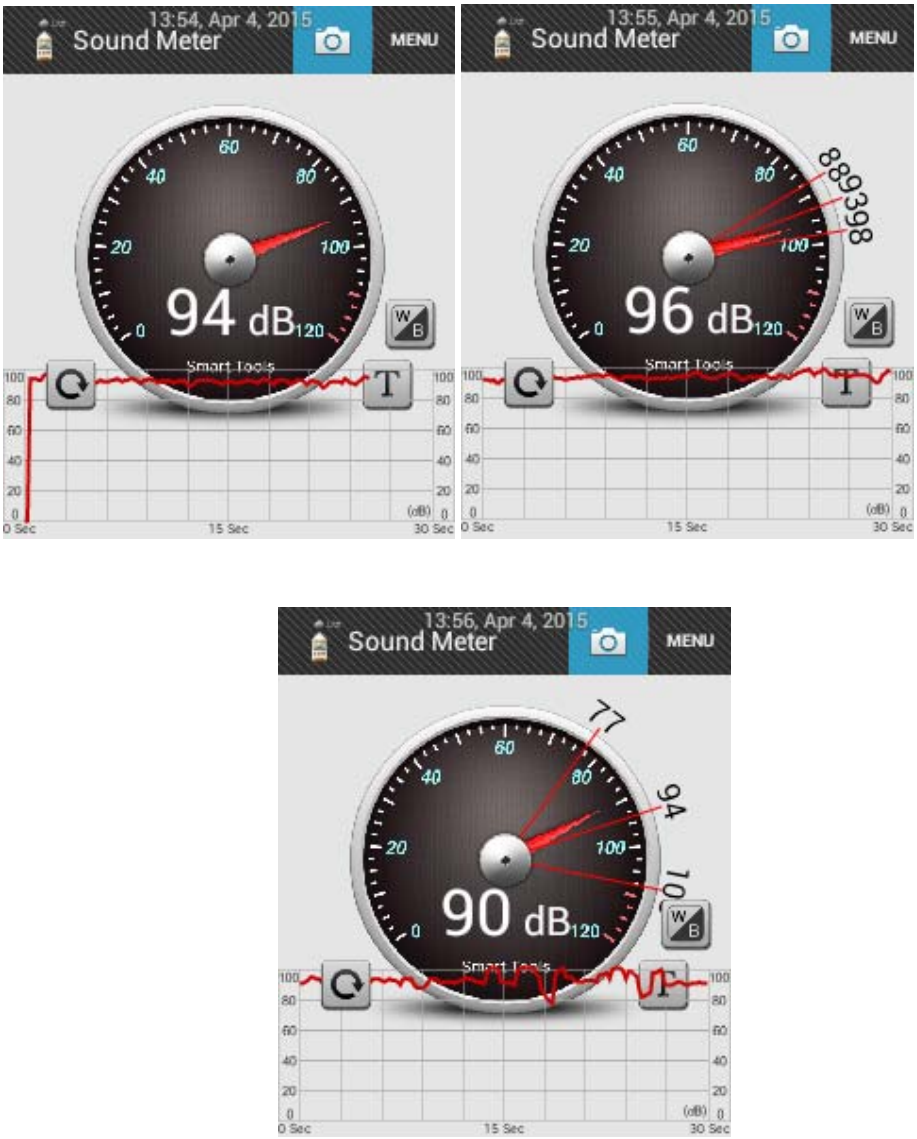


Figura 90. Toma de datos - Ruido Ambiental Noche

ANÁLISIS PRIMERA ETAPA DE EVALUACIÓN

De acuerdo a las consideraciones ya mencionadas anteriormente para este primer proceso de evaluación, y con el afán de realizar una comparación no solo del reconocimiento de voz en un ambiente simulado y real, se consideró que la prueba en ambiente real se realice en la mañana y la prueba en ambiente simulado se la realice en la noche, todo esto con el objetivo de ver de qué manera

influenciaba el ruido ambiental en el reconocimiento de voz, es por ello que en el cuadro 9, 10, 11, 12 y en las figuras 91, 92,93 y 94 que se muestran a continuación se presentan los valores de aciertos y desaciertos al momento de que el operador suministra un comando de voz al robot.

Voz: Hombre
Escenario: Real
Hora: Mañana

Cuadro 9

Datos Ambiente Real – Voz Hombre

REAL - VOZ HOMBRE		
	ACIERTOS	**DESACIERTOS**
AVANZAR	4	1
PARAR	3	2
RETROCEDER	3	2
GIRAR A LA IZQUIERDA	4	1
GIRAR A LA DERECHA	4	1
MEDIA VUELTA	4	1
TOTAL	**22**	**8**

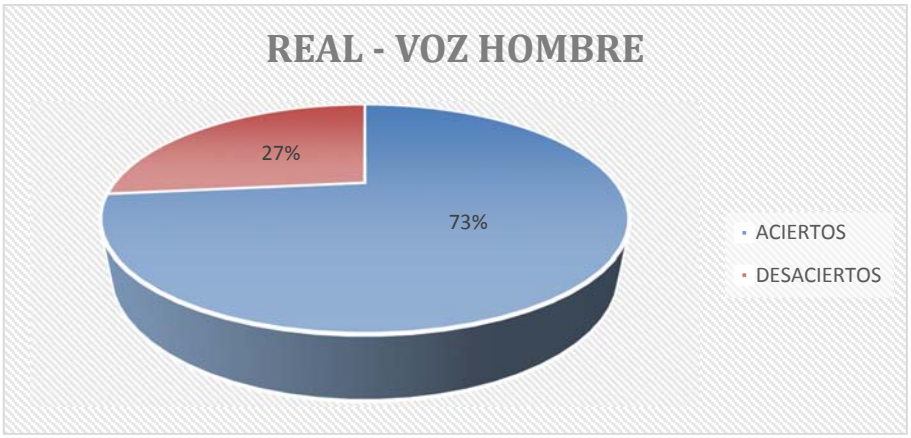


Figura 91. Tasa de Acierto - Ambiente Real – Voz Hombre

Escenario: Real

Voz: Mujer

Hora: Mañana

Cuadro 10

Datos Ambiente Real – Voz Mujer

REAL - VOZ MUJER		
	ACIERTOS	**DESACIERTOS**
AVANZAR	3	2
PARAR	3	2
RETROCEDER	3	2
GIRAR A LA IZQUIERDA	3	2
GIRAR A LA DERECHA	3	2
MEDIA VUELTA	4	1
TOTAL	**19**	**11**

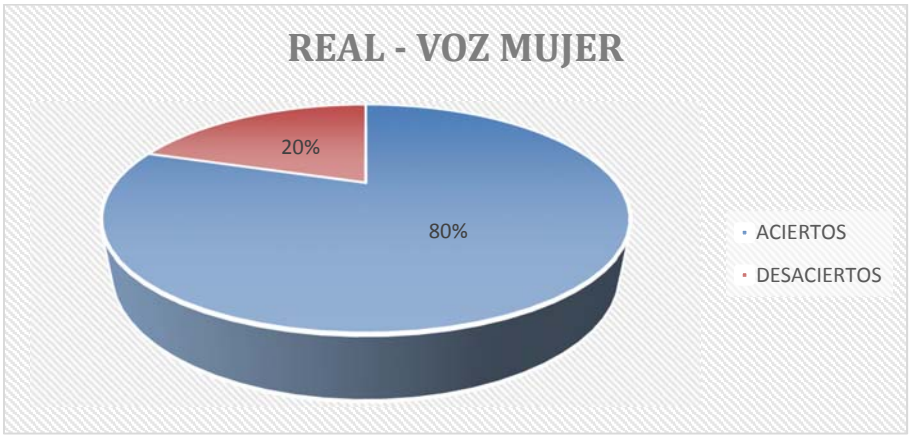


Figura 92. Tasa de Acierto - Ambiente Real – Voz Mujer

Escenario: Simulación

Voz: Hombre

Hora: Noche

Cuadro 11

Datos Ambiente Simulado – Voz Hombre

SIMULACIÓN - VOZ HOMBRE		
	ACIERTOS	**DESACIERTOS**
AVANZAR	4	1
PARAR	5	0
RETROCEDER	5	0
GIRAR A LA IZQUIERDA	5	0
GIRAR A LA DERECHA	4	1
MEDIA VUELTA	3	2
TOTAL	**26**	**4**

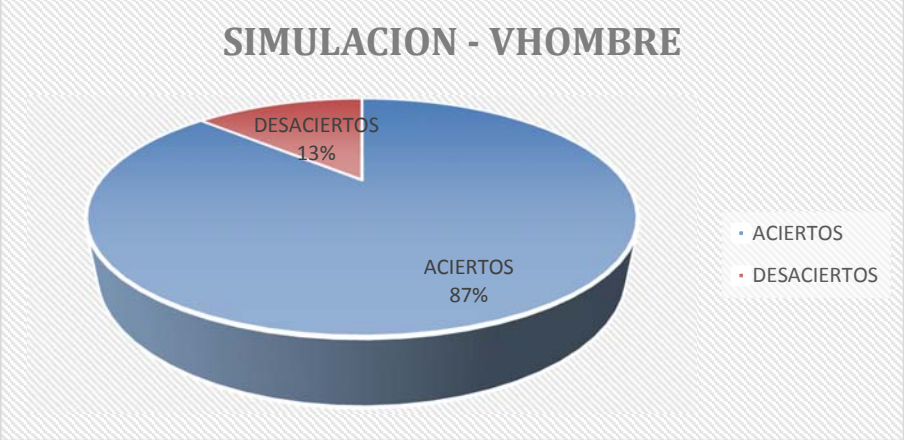


Figura 93. Tasa de Acierto - Ambiente Simulado – Voz Hombre

Escenario: Simulación

Voz: Mujer

Hora: Noche

Cuadro 12

Datos Ambiente Simulado – Voz Mujer

SIMULACIÓN - VOZ MUJER		
	ACIERTOS	**DESACIERTOS**
AVANZAR	4	1
PARAR	4	1
RETROCEDER	5	0
GIRAR A LA IZQUIERDA	5	0
GIRAR A LA DERECHA	5	0
MEDIA VUELTA	4	1
TOTAL	**27**	**3**

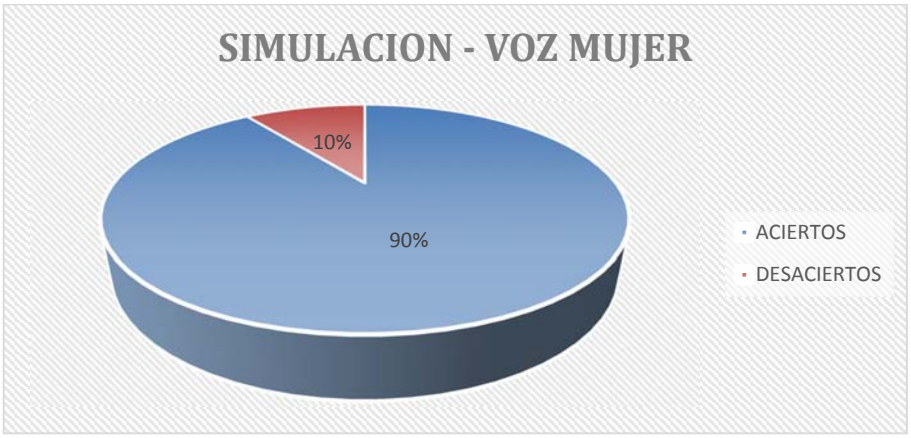


Figura 94. Tasa de Acierto - Ambiente Simulado – Voz Mujer

ANÁLISIS SEGUNDA ETAPA DE EVALUACIÓN

Este punto del análisis trata de evaluar el comportamiento de la arquitectura de control AD implementada a través de la plataforma MRDS, para ello vamos a considerar el entrenamiento de la tarea mostrada a continuación (ver figura 95) y verificar si la plataforma puede reproducir cada uno de los comandos suministrados en modo **Iniciar Entrenamiento.**

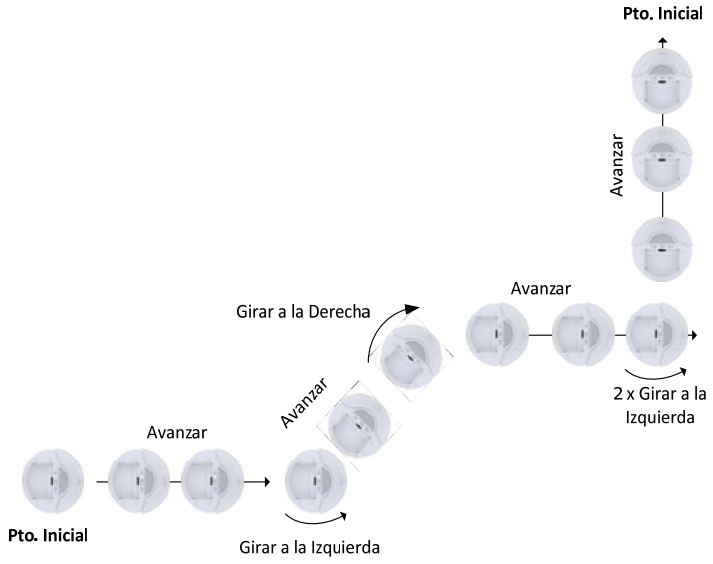


Figura 95. Tarea a reproducir por el iRobot en modo Iniciar Entrenamiento

La evaluación de este caso se lo va a realizar en un ambiente simulado y real, utilizando un hombre como sujeto de pruebas, el número de muestras a evaluarse es 5. A continuación se presentan los resultados obtenidos de este análisis en el cuadro 13, 14 y en la figura 96 y 97

Cuadro 13

Datos Ambiente Real – Voz Hombre – Ejecución de Tareas

REAL - VOZ HOMBRE		
INTENTO REP. TAREAS	**ACIERTOS**	**DESACIERTOS**
UNO	0	1
DOS	0	1
TRES	1	0
CUATRO	0	1
CINCO	0	1
TOTAL	**1**	**4**

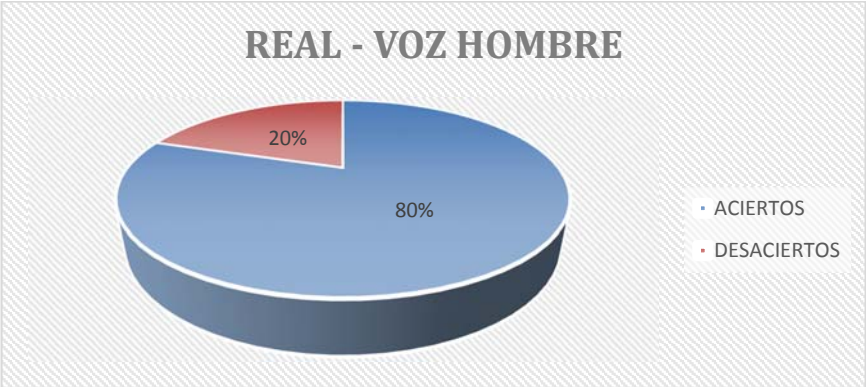


Figura 96.Tasa de Acierto - Ambiente Real – Voz Hombre – Ejecución Tareas

Cuadro 14

Datos Ambiente Simulado – Voz Hombre – Ejecución de Tareas

SIMULACIÓN - VOZ HOMBRE		
INTENTO REP. TAREAS	**ACIERTOS**	**DESACIERTOS**
UNO	1	0
DOS	1	0
TRES	0	1
CUATRO	1	0
CINCO	1	0
TOTAL	**4**	**1**

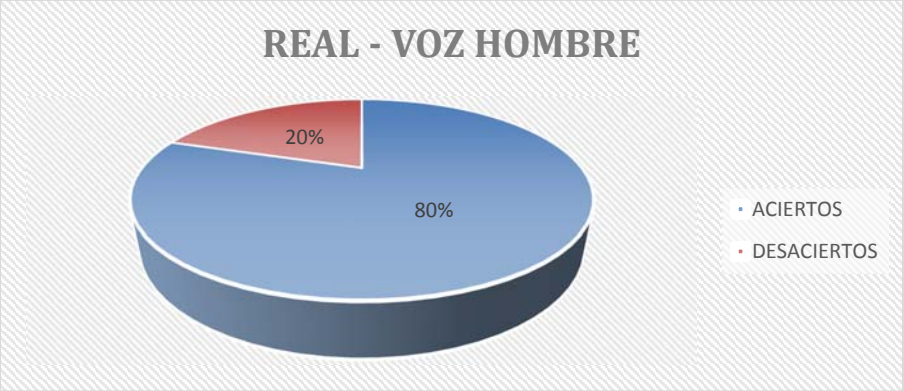


Figura 97.Tasa de Acierto – Ambiente Simulado – Voz Hombre – Ejecución Tareas

5.3 RESULTADOS OBTENIDOS

PRIMER CASO

El análisis de las gráficas claramente establece que el proceso de reconocimiento de voz tiene una tasa de acierto buena, como puntos a tomar en consideración se puede establecer que un ambiente bajo en decibelios proporciona una eficiencia aún mejor del programa, el tipo de sujeto (hombre/mujer) empleado en esta toma de datos casi no representa un factor a tomar en consideración en cuanto a porcentaje de eficiencia se refiere. Por lo que se puede establecer una relación inversa entre un ambiente ruidoso y el porcentaje de acierto en el reconocimiento de un comando, cuanto más razonable sea el ruido en un ambiente 20 – 30 [dB] mejor será el reconocimiento de voz.

SEGUNDO CASO

Dentro del análisis del segundo caso en donde el principal objetivo es evaluar el desempeño de la Tarea de Ejecución Autónoma, se puede observar claramente que en los primeros datos obtenidos al tratar de Ejecutar la Tarea por el iRobot la mayoría fueron fallidas, debido al problema de reconocimiento de voz, luego el

Speech Recognition fue reconociendo los patrones de voz de la persona conforme iba repitiendo las instrucciones; esto generó una mayor cantidad de eventos exitosos en las siguientes pruebas; sin embargo en todo momento se percibía un inconveniente con la tasa de muestreo (**Polling Interval)** de la comunicación entre la PC y el modulo BAM de la plataforma, ya que al mandar a ejecutar cualquiera de las sentencias asociadas a la operación **RotateDegrees** del Servicio **GenericDifferentialDrive** mientras el robot estaba en modo **EjecutarEntrenamiento,** el código perdía acceso al estado de los encoders de la plataforma.

CAPITULO 6

CONCLUSIONES Y RECOMENDACIONES

El hecho de abordar un tema de investigación nuevo, desde el punto de vista ingenieril siempre va a traer consigo la adquisición de nuevas herramientas y conocimientos que ayuden a fortalecer el pensamiento técnico-científico. Al término de este proyecto sin duda alguna uno de los puntos más relevantes dentro del análisis y estudio de las plataformas robóticas móviles es el hecho de dotarlas de habilidades y destrezas que las asemejen cada día más al comportamiento del ser humano.

El avance de la robótica ha establecido nuevos conceptos de control para estas plataformas, lo cual nos ha llevado a buscar herramientas de programación mucho más robustas y amigables, la plataforma Microsoft Robotics Delevelopment Studio (MRDS) es el principal eje de trabajo de este proyecto, al manejar una arquitectura orientada a servicios permite tratar a una plataforma robótica como un conjunto de servicios que se ejecutan en paralelo y cuya orquestación entre si resulta clave para definir el modelo de arquitectura Automático- Reactiva que se está tratando de implementar.

Uno de los puntos más fuertes en el desarrollo de aplicaciones robóticas a través de MRDS es que permite gestionar la concurrencia y coordinación de los anteriores servicios mencionados. También proporciona sencillas directivas para el control de la programación asíncrona.

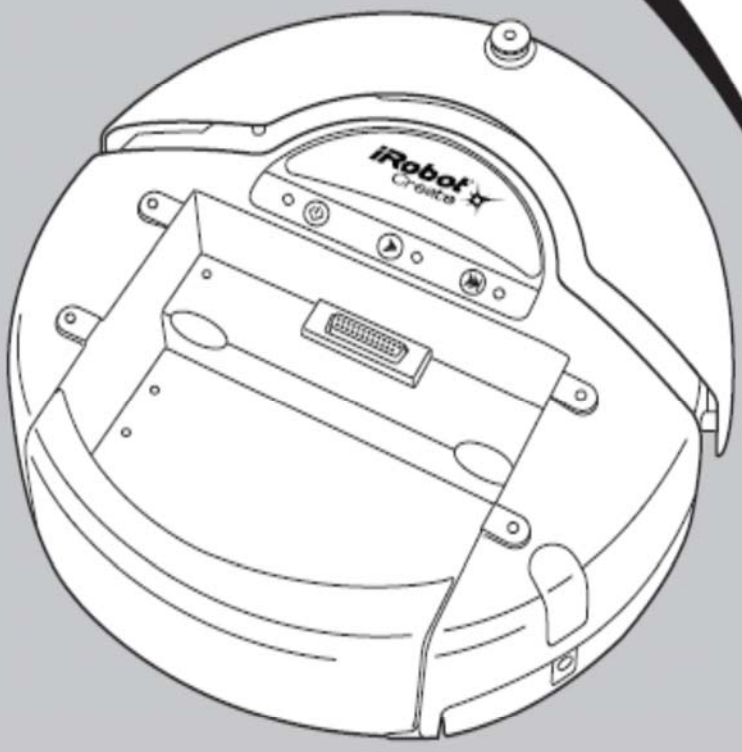
Como conclusión final se puede establecer que MRDS cuenta con una gran plataforma de simulación de entornos virtuales en tres dimensiones. Gracias a esto se puede simular con gran precisión la dinámica e interacciones físicas entre los elementos del entorno.

REFERENCIAS

1. Barber, R. (2002). Self-generation by a mobile robot of topological maps of corridors. In Proceedings of the 2002 IEEE International Conference on Robotics and Automation, 2662-2667.
2. Bares, J., & Wettergreen, D. (199). Dante II: Technical Description, Results, and Lessons Learned. The International Journal of Robotics Research, 621-649.
3. Brian, M. (1997). Mars pathfinder flight system integration and test. Aerospace Conference Proceedings IEEE, 1-8.
4. Brown, B., Frerreira, E., Shu-Jen, T., & Chris, P. (1997). Control of the Gyrover: a single-wheel gyroscopically stabilized robot. Proceedings IEEE Int. Conference on Robotics and Automation, 39-44
5. Giralt, G. (1979). A multi-level planning and navigation system for a mobile robot. Proceedings of the WCAI, 335-338.
6. Lauwers, T., Kantor, G., & Hollis, R. (2006). A dinamically stable single-wheeled mobile robot with inverse mouse-ball drive. Proc. IEEE International Conference on Robotics and Automation, 15-19.
7. Torres, F., Pomares, J., Gil, P., Puente, S., & Aracil, R. (2002). Introducción a la Robótica. Madrid: Pearson Educación.
8. Hayes - Roth, B. (1992). Artificial Intelligent . Estados Unidos: Elsevier Science
9. Ollero Baturone, A. (2007). Robótica manipuladores y robots móviles. España: Marcombo, Alfaomega.
10. Arrabales, R. (2007). Conscious Robots. Obtenido de http://www.conscious-robots.com/en/conscious-machines/the-field-of-machine-consciousness/cognitive-rob-2.html
11. Hollis, R. (1977). Newt: A mobile, cognitive robot. Byte Magazine, 30-45.
    Latombe, J. C. (1991). Robot Motion Planing. EEUU: Kluwer Academic Publishers
    Moravec, H. (1979). Visual mapping by a robot rover. Proc. 6th IJCAI.
12. Murphy, R. (2000). Introduction to AI Robotics . Estados Unidos: MIT Press .
13. Nilsson, N. (1984). Shakey the robot. Technical NOte 323.
14. Pratt, J., & Pratt, G. (1998). Intuitive Control of a Planar Bipedal Walking Robot.
15. Proceedings IEEE International Conference on Robotics an Automation, 16-21.
16. Raibert, M. (1986). Legged robots that balance. MIT.Press.
17. Siegwart, R., & Nourbakhsh, I. (2004). Introduction to autonomous mobile robots. Estados Unidos: MIT Press.
18. Tanie, K. (2003). Humanoid robot and its application possibility, multisensor fusion and integration for intelligent systems. Proceedings of IEEE International Conference.
19. Thompson, A. (1977). The navigation system of the JPL robot. Proceedings 5th IJCAI.
20. Zeglin, G. (1999). The Bow Leg Hopping Robot. The Robotics Institute Carnegie Mellon University.

ANEXO 1

iRobot® Create
OPEN INTERFACE



i♥Robots

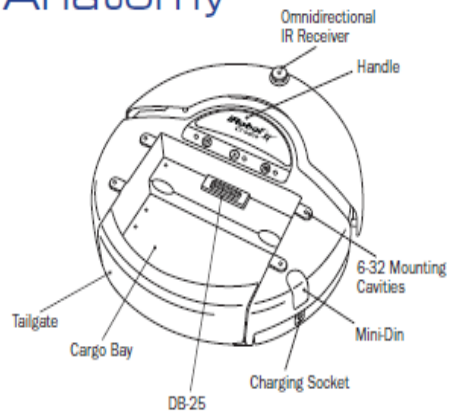
iRobot
www.irobot.com

Table of Contents

iRobot Create® Open Interface Overview

The Create Open Interface (OI) consists of an electronic interface and a software interface for controlling Create's behavior and reading its sensors. The electronic interface includes a 7 pin Mini-DIN connector and a DB-25 connector in the Cargo Bay for connecting hardware and electronics for sensors and actuators such as a robotic arm or light sensor to Create. The software interface lets you manipulate Create's behavior and read its sensors through a series of commands including mode commands, actuator commands, song commands, demo commands, and sensor commands that you send to Create's serial port by way of a PC or microcontroller that is connected to the Mini-DIN connector or Cargo Bay Connector.

Anatomy

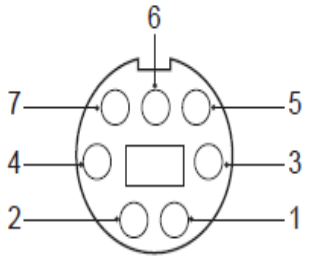


Physical Connections

To use the OI, a processor capable of generating serial commands such as a PC or a microcontroller must be connected to the external Mini-DIN connector or the Cargo Bay Connector on Create. These connectors provide two-way, serial communication at TTL (0 – 5V) levels. The connectors also provide an unregulated direct connection to iRobot Create's battery, which you can use to power the OI applications. The Cargo Bay Connector also provides a regulated 5V power supply and several input and output pins (see details below). The Mini-DIN connector is located in the rear right side of Create, beneath a snap-fit plastic guard, while the Cargo Bay Connector is located in the front middle of the cargo bay.

Mini-DIN Connector

This diagram shows the pinout of the top view of the female connector in Create. Note that pins 5,6 and 7 are towards the outside circumference of Create.

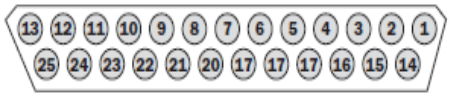


Pin	Name	Description
1	Vpwr	Create battery + (unregulated)
2	Vpwr	Create battery + (unregulated)
3	RXD	0 – 5V Serial input to Create
4	TXD	0 – 5V Serial output from Create
5	BRC	Baud Rate Change
6	GND	Create battery ground
7	GND	Create battery ground

Since the RXD and TXD pins use 0 – 5V logic voltage and the PC serial ports use different voltages (rs232 levels), it is necessary to shift voltage levels. To do this, use an iRobot Create serial cable rather than a normal serial cable, as the iRobot Create serial cable contains all of the necessary hardware to shift the voltage levels, whereas the normal serial cable does not.

Cargo Bay Connector

The Cargo Bay Connector, located in the front middle of the cargo bay, contains 25 pins that you can use to attach electronics for peripheral devices such as additional sensors. The Cargo Bay Connector provides four digital inputs, an analog input, three digital outputs, three high-current low side driver outputs (useful for driving motors), a charging indicator, a power toggle input, serial Tx and Rx, a 5V reference, battery ground and battery voltage.



Pin	Name	Description
1	RXD	0 – 5V Serial input to Create
2	TXD	0 – 5V Serial output from Create
3	Power control toggle	Turns Create on or off on a low-to-high transition
4	Analog input	0 - 5V analog input to Create
5	Digital input 1	0 - 5V digital input to Create
6	Digital input 3	0 - 5V digital input to Create
7	Digital output 1	0 - 5V, 20 mA digital output from Create
8	Switched 5V	Provides a regulated 5V 100 mA supply and analog reference voltage when Create is switched on
9	Vpwr	Create battery voltage (unregulated), 0.5A
10	Switched Vpwr	Provides battery power @ 1.5 A when Create is powered on.
11	Switched Vpwr	Provides battery power @ 1.5 A when Create is powered on.
12	Switched Vpwr	Provides battery power @ 1.5 A when Create is powered on.
13	Robot charging	When Create is charging, this pin is high (5V)
14	GND	Create battery ground
15	Device Detect/Baud Rate Change Pin	0-5V digital input to Create which can also be used to change the baud rate to 19200 (see below)
16	GND	Create battery ground
17	Digital input 0	0 - 5V digital input to Create
18	Digital input 2	0 - 5V digital input to Create
19	Digital output 0	0 - 5V, 20 mA digital output from Create
20	Digital output 2	0 - 5V, 20 mA digital output from Create
21	GND	Create battery ground
22	Low side driver 0	0.5A low side driver from Create
23	Low side driver 1	0.5A low side driver from Create
24	Low side driver 2	1.5A low side driver from Create
25	GND	Create battery ground

Serial Port Settings

Baud: 57600 or 19200 (see below)

Data bits: 8

Parity: None

Stop bits: 1

Flow control: None

By default, iRobot Create communicates at 57600 baud. If you are using a microcontroller that does not support 57600 baud, there are two ways to force Create to switch to 19200:

Method 1:

When powering on Create, hold down the Play button. After about 4 seconds, Create plays a tune of descending pitches. Create will communicate at 19200 baud until the power is turned off, the battery is removed and reinserted, the battery voltage falls below the minimum required for processor operation, or the baud rate is explicitly changed by way of the OI.

Method 2:

Use the Baud Rate Change pin (pin 15 on the Cargo Bay Connector/pin 5 on the Mini-DIN connector) to change Create's baud rate. After turning on Create, wait 2 seconds and then pulse the Baud Rate Change low three times. Each pulse should last between 50 and 500 milliseconds. Create will communicate at 19200 baud until the processor loses battery power or the baud rate is explicitly changed by way of the OI.

iRobot Create Open Interface Modes

The Create OI has four operating modes: Off, Passive, Safe, and Full. After a battery change or when is first supplied, the OI is in "off" mode. When it is off, the OI listens at the default baud rate (57600 or 19200 - see Serial Port Settings above) for an OI Start command. Once it receives the Start command, you can enter into any one of the four operating modes by sending a mode command to the OI. You can also switch between operating modes at any time by sending a command to the OI for the operating mode that you want to use.

Passive Mode

Upon sending the Start command or any one of the demo commands (which also starts the specific demo, e.g., Spot Cover, Cover, Cover and Dock, or Demo), the OI enters into Passive mode. When the OI is in Passive mode, you can request and receive sensor data using any of the sensors commands, but you cannot change the current command parameters for the actuators (motors, speaker, lights, low side drivers, digital outputs) to something else. To change how one of the actuators operates, you must switch from Passive mode to Full mode or Safe mode.

While in Passive mode, you can read Create's sensors, watch Create perform any one of its ten built-in demos, and charge the battery.

Safe Mode

When you send a Safe command to the OI, Create enters into Safe mode. Safe mode gives you full control of Create, with the exception of the following safety-related conditions:

- Detection of a cliff while moving forward (or moving backward with a small turning radius, less than one robot radius).
- Detection of a wheel drop (on any wheel).
- Charger plugged in and powered.

Should one of the above safety-related conditions occur while the OI is in Safe mode, Create stops all motors and reverts to the Passive mode.

If no commands are sent to the OI when in Safe mode, Create waits with all motors and LEDs off and does not respond to Play or Advance button presses or other sensor input.

Note that charging terminates when you enter Safe Mode.

Full Mode

When you send a Full command to the OI, Create enters into Full mode. Full mode gives you complete control over Create, all of its actuators, and all of the safety-related conditions that are restricted when the OI is in Safe mode, as Full mode shuts off the cliff, wheel-drop and internal charger safety features. To put the OI back into Safe mode, you must send the Safe command.

If no commands are sent to the OI when in Full mode, Create waits with all motors and LEDs off and does not respond to Play or Advance button presses or other sensor input.

Note that charging terminates when you enter Full Mode.

Open Interface Command Reference

The following is a list of all of iRobot Create's Open Interface commands. Each command starts with a one-byte opcode. Some of the commands must be followed by data bytes. All of Create's OI commands including their required data bytes are described below.

NOTE: Always send the required number of data bytes for the command, otherwise, the processor will enter and remain in a "waiting" state until all of the required data bytes are received.

Getting Started Commands

The following commands start the Open Interface and get it ready for use.

Start **Opcode:128** **Data Bytes: 0**

This command starts the OI. You must always send the Start command before sending any other commands to the OI.

- Serial sequence: [128].
- Available in modes: Passive, Safe, or Full
- Changes mode to: Passive. Create beeps once to acknowledge it is starting from "off" mode.

Baud **Opcode: 129** **Data Bytes: 1**

This command sets the baud rate in bits per second (bps) at which OI commands and data are sent according to the baud code sent in the data byte. The default baud rate at power up is 57600 bps, but the starting baud rate can be changed to 19200 by holding down the Play button while powering on Create until you hear a sequence of descending tones. Once the baud rate is changed, it persists until Create is power cycled by pressing the power button or removing the battery, or when the battery voltage falls below the minimum required for processor operation. You must wait 100ms after sending this command before sending additional commands at the new baud rate.

Note: at a baud rate of 115200, there must be at least 200µs between the onset of each character, or some characters may not be received.

- Serial sequence: [129][Baud Code]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Baud data byte 1: Baud Code (0 - 11)

Baud Code	Baud Rate in BPS
0	300
1	600
2	1200
3	2400
4	4800
5	9600
6	14400
7	19200
8	28800
9	38400
10	57600
11	115200

Mode Commands

Create has four operating modes: Off, Passive, Safe, and Full. Create powers on in the Passive mode. The following commands change Create's OI mode.

Safe **Opcode: 131** **Data Bytes: 0**

This command puts the OI into Safe mode, enabling user control of Create. It turns off all LEDs. The OI can be in Passive, Safe, or Full mode to accept this command.

- Serial sequence: [131]
- Available in modes: Passive, Safe, or Full
- Changes mode to: Safe

Note: The effect and usage of the Control command (130) is identical to the Safe command. The Control command is deprecated but is present for backward compatibility with the Roomba Open Interface. Use Safe command instead.

Full **Opcode: 132** **Data Bytes: 0**

This command gives you complete control over Create by putting the OI into Full mode, and turning off the cliff, wheel-drop and internal charger safety features. That is, in Full mode, Create executes any command that you send it, even if the internal charger is plugged in, or the robot senses a cliff or wheel drop.

- Serial sequence: [132]
- Available in modes: Passive, Safe, or Full
- Changes mode to: Full

Note: Use the Start command (128) to change the mode to Passive.

Demo Commands

The following are commands to start iRobot Create's built-in demos.

Demo **Opcode: 136** **Data Bytes: 1**

This command starts the requested built-in demo.

- Serial sequence: [136][Which-demo]
- Available in modes: Passive, Safe, or Full
- Changes mode to: Passive
- Demo data byte 1: Demo number (-1 - 9)

Demo Names, Descriptions and Numbers

Number	Demo	Description
-1 (255)	Abort current demo	Stops the demo that Create is currently performing.
0	Cover	Create attempts to cover an entire room using a combination of behaviors, such as random bounce, wall following, and spiraling.
1	Cover and Dock	Identical to the Cover demo, with one exception. If Create sees an infrared signal from an iRobot Home Base, it uses that signal to dock with the Home Base and recharge itself.
2	Spot Cover	Create covers an area around its starting position by spiraling outward, then inward.
3	Mouse	Create drives in search of a wall. Once a wall is found, Create drives along the wall, traveling around circumference of the room.
4	Drive Figure Eight	Create continuously drives in a figure 8 pattern.
5	Wimp	Create drives forward when pushed from behind. If Create hits an obstacle while driving, it drives away from the obstacle.
6	Home	Create drives toward an iRobot Virtual Wall as long as the back and sides of the virtual wall receiver are blinded by black electrical tape. A Virtual Wall emits infrared signals that Create sees with its Omnidirectional Infrared Receiver, located on top of the bumper. If you want Create to home in on a Virtual Wall, cover all but a small opening in the front of the infrared receiver with black electrical tape. Create spins to locate a virtual wall, then drives toward it. Once Create hits the wall or another obstacle, it stops.
7	Tag	Identical to the Home demo, except Create drives into multiple virtual walls by bumping into one, turning around, driving to the next virtual wall, bumping into it and turning around to bump into the next virtual wall.
8	Pachelbel	Create plays the notes of Pachelbel's Canon in sequence when cliff sensors are activated.
9	Banjo	Create plays a note of a chord for each of its four cliff sensors. Select the chord using the bumper, as follows: • No bumper: G major. • Right/left bumper: D major 7 • Both bumpers (center): C major

You can also call the Cover, Cover and Seek Dock and Spot Demos using the Opcodes specified below. This is present for ensuring backward compatibility with the Roomba OI.

Cover **Opcode: 135** **Data Bytes: 0**

This command starts the Cover demo.

- Serial sequence: [135]
- Available in modes: Passive, Safe, or Full
- Changes mode to: Passive

Cover and Dock **Opcode: 143** **Data Bytes: 0**

This command starts the Cover and Dock demo.

- Serial sequence: [143]
- Available in modes: Passive, Safe, or Full
- Changes mode to: Passive

Spot **Opcode: 134** **Data Bytes: 0**

This command starts the Spot Cover demo.

- Serial sequence: [134]
- Available in modes: Passive, Safe, or Full
- Changes mode to: Passive

Actuator Commands

The following commands control iRobot Create's actuators: wheels, speaker, LEDS, digital outputs and low side driver outputs.

Drive **Opcode: 137** **Data Bytes: 4**

This command controls Create's drive wheels. It takes four data bytes, interpreted as two 16-bit signed values using two's complement. The first two bytes specify the average velocity of the drive wheels in millimeters per second (mm/s), with the high byte being sent first. The next two bytes specify the radius in millimeters at which Create will turn. The longer radii make Create drive straighter, while the shorter radii make Create turn more. The radius is measured from the center of the turning circle to the center of Create. A Drive command with a positive velocity and a positive radius makes Create drive forward while turning toward the left. A negative radius makes Create turn toward the right. Special cases for the radius make Create turn in place or drive straight, as specified below. A negative velocity makes Create drive backward.

NOTE: Internal and environmental restrictions may prevent Create from accurately carrying out some drive commands. For example, it may not be possible for Create to drive at full speed in an arc with a large radius of curvature.

- Serial sequence: [137] [Velocity high byte] [Velocity low byte] [Radius high byte] [Radius low byte]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Drive data byte 1: Velocity (-500 – 500 mm/s)
- Drive data byte 2: Radius (-2000 – 2000 mm)

Special cases:

Straight = 32768 or 32767 = hex 8000 or 7FFF

Turn in place clockwise = hex FFFF

Turn in place counter-clockwise = hex 0001

Example:

To drive in reverse at a velocity of -200 mm/s while turning at a radius of 500mm, send the following serial byte sequence:

[137] [255] [56] [1] [244]

Velocity = -200 = hex FF38 = [hex FF] [hex 38] = [255] [56]

Radius = 500 = hex 01F4 = [hex 01] [hex F4] = [1] [244]

Drive Direct **Opcode: 145** **Data Bytes: 4**

This command lets you control the forward and backward motion of Create's drive wheels independently. It takes four data bytes, which are interpreted as two 16-bit signed values using two's complement. The first two bytes specify the velocity of the right wheel in millimeters per second (mm/s), with the high byte sent first. The next two bytes specify the velocity of the left wheel, in the same format. A positive velocity makes that wheel drive forward, while a negative velocity makes it drive backward.

- Serial sequence: [145] [Right velocity high byte] [Right velocity low byte] [Left velocity high byte] [Left velocity low byte]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Drive Direct data byte 1: Right wheel velocity (-500 – 500 mm/s)
- Drive Direct data byte 1: Left wheel velocity (-500 – 500 mm/s)

LEDs **Opcode: 139** **Data Bytes: 3**

This command controls the LEDs on Create. The state of the Play and Advance LEDs is specified by two bits in the first data byte. The power LED is specified by two data bytes: one for the color and the other for the intensity.

- Serial sequence: [139] [LED Bits] [Power Color] [Power Intensity]
- Available in modes: Safe or Full
- Changes mode to: No Change
- LEDs data byte 1: LED Bits (0 – 10)

Advance and Play use green LEDs. 0 = off, 1 = on

Bit	7	6	5	4	3	2	1	0
LED	n/a	n/a	n/a	n/a	Advance	n/a	Play	n/a

Power uses a bicolor (red/green) LED. The intensity and color of this LED can be controlled with 8-bit resolution.

- LEDs data byte 2: Power LED Color (0 – 255) 0 = green, 255 = red. Intermediate values are intermediate colors (orange, yellow, etc).
- LEDs data byte 3: Power LED Intensity (0 – 255) 0 = off, 255 = full intensity. Intermediate values are intermediate intensities.

Example:

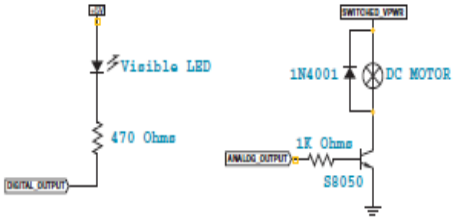
To turn on the Advance LED and light the Power LED green at half intensity, send the serial byte sequence [139] [8] [0] [128].

Digital Outputs **Opcode: 147** **Data Bytes: 1**

This command controls the state of the 3 digital output pins on the 25 pin Cargo Bay Connector. The digital outputs can provide up to 20 mA of current.

- Serial sequence: [147] [Output Bits]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Digital Outputs data byte 1: Output Bits (0 –7); 0 = low (0V); 1 = high (5V).

Example schematics



Warning: When the Robot is switched ON, the Digital Outputs are High for the first 3 seconds during the initialization of the bootloader

Bit	7	6	5	4	3	2	1	0
Output						digital-out-2 (pin 20)	digital-out-1 (pin 7)	digital-out-0 (pin 19)

PWM Low Side Drivers **Opcode: 144** **Data Bytes: 3**

This command lets you control the three low side drivers with variable power. With each data byte, you specify the PWM duty cycle for the low side driver (max 128). For example, if you want to control a driver with 25% of battery voltage, choose a duty cycle of 128 * 25% = 32.

- Serial sequence: [144] [Low Side Driver 2 Duty Cycle] [Low Side Driver 1 Duty Cycle] [Low Side Driver 0 Duty Cycle]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Low Side Drivers data byte 1: Duty cycle for low side driver 2 (0 - 128)
- Low Side Drivers data byte 2: Duty cycle for low side driver 1 (0 - 128)
- Low Side Drivers data byte 3: Duty cycle for low side driver 0 (0 - 128)

Example:

To turn on low side driver 2 at 25% and low side driver 0 at 100%, send the serial byte sequence [144][32][0][128]

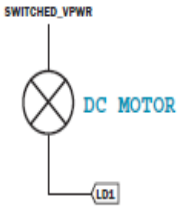
Low Side Drivers **Opcode: 138** **Data Bytes: 1**

This command lets you control the three low side drivers. The state of each driver is specified by one bit in the data byte.

Low side drivers 0 and 1 can provide up to 0.5A of current. Low side driver 2 can provide up to 1.5 A of current. If too much current is requested, the current is limited and the overcurrent flag is set (sensor packet 14).

- Serial sequence: [138] [Driver Bits]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Low Side Drivers data byte 1: Driver bits (0 – 7)

Example schematic



0 = off, 1 = on at 100% PWM duty cycle

Bit	7	6	5	4	3	2	1	0
Output						Side Driver 2 (pin 24)	Low Side Driver 1 (pin 22)	Low Side Driver 0 (pin 23)

Example:

To turn on only low side driver 1, send the serial byte sequence [138] [2].

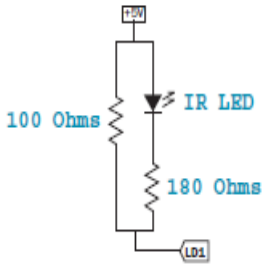
Note: Speed control of motors uses the PWM Low Side Drivers Command. This command exists for Backward compatibility with the Roomba OI.

Send IR **Opcode: 151** **Data Bytes: 1**

This command sends the requested byte out of low side driver 1 (pin 23 on the Cargo Bay Connector), using the format expected by iRobot Create's IR receiver. You must use a preload resistor (suggested value: 100 ohms) in parallel with the IR LED and its resistor in order turn it on.

- Serial sequence: [151][Byte Value]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Send IR data byte 1: Byte value to send (0 - 255)

Example Schematic



Song **Opcode: 140** **Data Bytes: 2N+2,**

where N is the number of notes in the song

This command lets you specify up to sixteen songs to the OI that you can play at a later time. Each song is associated with a song number. The Play command uses the song number to identify your song selection. Each song can contain up to sixteen notes. Each note is associated with a note number that uses MIDI note definitions and a duration that is specified in fractions of a second. The number of data bytes varies, depending on the length of the song specified. A one note song is specified by four data bytes. For each additional note within a song, add two data bytes.

- Serial sequence: [140] [Song Number] [Song Length] [Note Number 1] [Note Duration 1] [Note Number 2] [Note Duration 2], etc.
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Song data byte 1: Song Number (0 – 15)

  The song number associated with the specific song. If you send a second Song command, using the same song number, the old song is overwritten.
- Song data byte 2: Song Length (1 – 16)

  The length of the song, according to the number of musical notes within the song.
- Song data bytes 3, 5, 7, etc.: Note Number (31 – 127)

  The pitch of the musical note Create will play, according to the MIDI note numbering scheme. The lowest musical note that Create will play is Note #31. Create considers all musical notes outside the range of 31 – 127 as rest notes, and will make no sound during the duration of those notes.
- Song data bytes 4, 6, 8, etc.: Note Duration (0 – 255)

  The duration of a musical note, in increments of 1/64th of a second. Example: a half-second long musical note has a duration value of 32

Note Durations

Number	Note	Frequency	Number	Note	Frequency	Number	Note	Frequency
			60	C	261.6	96	C	2093.0
			61	C#	277.2	97	C#	2217.5
			62	D	293.7	98	D	2349.3
			63	D#	311.1	99	D#	2489.0
			64	E	329.6	100	E	2637.0
			65	F	349.2	101	F	2793.8
			66	F#	370.0	102	F#	2960.0
31	G	49.0	67	G	392.0	103	G	3136.0
32	G#	51.0	68	G#	415.3	104	G#	3322.4
33	A	55.0	69	A	440.0	105	A	3520.0
34	A#	58.3	70	A#	466.2	106	A#	3729.3
35	B	61.7	71	B	493.9	107	B	3951.1
36	C	65.4	72	C	523.3	108	C	4186.0
37	C#	69.3	73	C#	554.4	109	C#	4434.9
38	D	73.4	74	D	587.3	110	D	4698.6
39	D#	77.8	75	D#	622.3	111	D#	4978.0
40	E	82.4	76	E	659.3	112	E	5274.0
41	F	87.3	77	F	698.5	113	F	5587.7
42	F#	92.5	78	F#	740.0	114	F#	5919.9
43	G	98.0	79	G	784.0	115	G	6271.9
44	G#	103.8	80	G#	830.6	116	G#	6644.9
45	A	110.0	81	A	880.0	117	A	7040.0
46	A#	116.5	82	A#	932.3	118	A#	7458.6
47	B	123.5	83	B	987.8	119	B	7902.1
48	C	130.8	84	C	1046.5	120	C	8372.0
49	C#	138.6	85	C#	1108.7	121	C#	8869.8
50	D	146.8	86	D	1174.7	122	D	9397.3
51	D#	155.6	87	D#	1244.5	123	D#	9956.1
52	E	164.8	88	E	1318.5	124	E	10548.1
53	F	174.6	89	F	1396.9	125	F	11175.3
54	F#	185.0	90	F#	1480.0	126	F#	11839.8
55	G	196.0	91	G	1568.0	127	G	12543.9
56	G#	207.7	92	G#	1661.2			
57	A	220.0	93	A	1760.0			
58	A#	233.1	94	A#	1864.7			
59	B	246.9	95	B	1975.5			

Play Song **Opcode: 141** **Data Bytes: 1**

This command lets you select a song to play from the songs added to iRobot Create using the Song command. You must add one or more songs to Create using the Song command in order for the Play command to work. Also, this command does not work if a song is already playing. Wait until a currently playing song is done before sending this command. Note that the "song playing" sensor packet can be used to check whether Create is ready to accept this command.

- Serial sequence: [141] [Song Number]
- Available in modes: Safe or Full
- Changes mode to: No Change
- Play Song data byte 1: Song Number (0 – 15)

  The number of the song Create is to play.

Input Commands

The following commands let you read the state of Create's built-in sensors, digital and analog inputs, and some internal state variables. Create updates these values internally every 15 ms. Do not send these commands more frequently than that.

Sensors **Opcode: 142** **Data Bytes: 1**

This command requests the OI to send a packet of sensor data bytes. There are 43 different sensor data packets. Each provides a value of a specific sensor or group of sensors.

For more information on sensor packets, refer to the next section, "Create Open Interface Sensor Packets."

- Serial sequence: [142] [Packet ID]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Sensors data byte 1: Packet ID (0 - 42)

  Identifies which of the 43 sensor data packets should be sent back by the OI. A value of 6 indicates a packet with all of the sensor data. Values of 0 through 5 indicate specific subgroups of the sensor data (see Sensors Quick Reference below).

Query List **Opcode: 149** **Data Bytes: N + 1,**

where N is the number of packets requested.

This command lets you ask for a list of sensor packets. The result is returned once, as in the Sensors command. The robot returns the packets in the order you specify.

- Serial sequence: [149][Number of Packets] [Packet ID 1][Packet ID 2]...[Packet ID N]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Query List data byte 1: Number of packets requested (0 - 255)
- Query List data bytes 2 - N: IDs of packets requested (0 - 42)

Example:

To get the state of the left cliff sensor (packet 9) and the Virtual Wall detector (packet 13), send the following sequence:

[149] [2] [9] [13]

Stream **Opcode: 148** **Data Bytes: N + 1,**

where N is the number of packets requested.

This command starts a continuous stream of data packets. The list of packets requested is sent every 15 ms, which is the rate iRobot Create uses to update data.

This is the best method of requesting sensor data if you are controlling Create over a wireless network (which has poor real-time characteristics) with software running on a desktop computer.

- Serial sequence: [148] [Number of packets] [Packet ID 1] [Packet ID 2] [Packet ID 3] etc.
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Stream data byte 1: Number of packets requested (0 - 43)
- Stream data bytes 2 - N: IDs of packets requested (0 - 42)

The format of the data returned is:

[19][N-bytes][Packet ID 1][Packet 1 data...]
[Packet ID 2][Packet 2 data...][Checksum]

N-bytes is the number of bytes between the n-bytes byte and the checksum.

The checksum is a 1-byte value. It is the 8-bit complement of all of the bytes between the header and the checksum. That is, if you add all of the bytes after the checksum, and the checksum, the low byte of the result will be 0.

Example:

To get data from Create's left cliff signal (packet 29) and Virtual Wall detector (packet 13), send the following command string to Create:

[148] [2] [29] [13]

NOTE: The left cliff signal is a 2-byte packet and the IR Sensor is a 1-byte packet.

Create starts streaming data that looks like this:

19	5	29	2 25	13	0	182
header	n-bytes	packet ID 1	Packet data 1 (2 bytes)	packet ID 2	packet data 2 (1 byte)	Checksum

NOTE: Checksum computation: (5 + 29 + 2 + 25 + 13 + 0 + 182) = 256 and (256 & 0xFF) = 0.

In the above stream segment, Create's left cliff signal value was 549 (0x0225) and there was no virtual wall signal.

It is up to you not to request more data than can be sent at the current baud rate in the 15 ms time slot. For example, at 57600 baud, a maximum of 86 bytes can be sent in 15 ms:

15 ms / 10 bits (8 data + start + stop) * 57600 = 86.4

If more data is requested, the data stream will eventually become corrupted. This can be confirmed by checking the checksum.

The header byte and checksum can be used to align your receiving program with the data. All data chunks start with 19 and end with the 1-byte checksum

Pause/Resume Stream **Opcode: 150** **Data Bytes: 1**

This command lets you stop and restart the steam without clearing the list of requested packets.

- Serial sequence: [150][Stream State]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Pause/Resume Stream data byte 1: Requested stream state (0 - 1)

An argument of 0 stops the stream without clearing the list of requested packets. An argument of 1 starts the stream using the list of packets last requested.

Script Commands

The following commands let you specify a script for iRobot Create to play at a later time.

Script **Opcode: 152** **Data Bytes: N + 1**

where N is the number of bytes in the script.

This command specifies a script to be played later. A script consists of OI commands and can be up to 100 bytes long. There is no flow control, but "wait" commands (see below) cause Create to hold its current state until the specified event is detected.

- Serial sequence: [152] [Script Length] [Opcode 1] [Opcode 2] [Opcode 3] etc.
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Script data byte 1: Script Length (0 – 100) Specifies the length of the script in terms of the number of commands. Specify a length of 0 to clear the current script.
- Script data bytes 2 and above: Open Interface commands and data bytes

Tip: To make a script loop forever, use Play Script (153) as the last command.

Example Scripts:

Drive 40 cm and stop:

152 13 137 1 44 128 0 156 1 144 137 0 0 0 0

Toggle led on bump:

152 17 158 5 158 251 139 2 0 0 158 5 158 251 139 0 0 0 153

Drive in a square:

152 17 137 1 44 128 0 156 1 144 137 1 44 0 1 157 0 90 153

Play Script **Opcode: 153** **Data Bytes: 0**

This command loads a previously defined OI script into the serial input queue for playback.

- Serial sequence: [153]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change

Show Script **Opcode: 154** **Data Bytes: 0**

This command returns the values of a previously stored script, starting with the number of bytes in the script and followed by the script's commands and data bytes. It first halts the sensor stream, if one has been started with a Stream or Pause/Resume Stream command. To restart the stream, send Pause/Resume Stream (opcode 150).

- Serial sequence: [154]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change

Wait Commands

The following commands cause Create to wait for a specific time, distance, angle of rotation, or event to occur. While it is waiting, Create does not change its state, nor does it react to any inputs, serial or otherwise. These commands are intended for use in scripting only.

Wait Time **Opcode: 155** **Data Bytes: 1**

This command causes Create to wait for the specified time. During this time, Create's state does not change, nor does it react to any inputs, serial or otherwise.

- Serial sequence: [155] [time]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Wait Time data byte 1: Time (0 - 255)

Specifies time to wait in tenths of a second with a resolution of 15 ms.

Wait Distance **Opcode: 156** **Data Bytes: 2**

This command causes iRobot Create to wait until it has traveled the specified distance in mm. When Create travels forward, the distance is incremented. When Create travels backward, the distance is decremented. If the wheels are passively rotated in either direction, the distance is incremented. Until Create travels the specified distance, its state does not change, nor does it react to any inputs, serial or otherwise.

NOTE: This command resets the distance variable that is returned in Sensors packets 19, 2 and 6.

- Serial sequence: [156] [Distance high byte] [Distance low byte]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Wait Distance data bytes 1-2: 16-bit signed distance in mm, high byte first (-32767 -32768)

Wait Angle **Opcode: 157** **Data Bytes: 2**

This command causes Create to wait until it has rotated through specified angle in degrees. When Create turns counterclockwise, the angle is incremented. When Create turns clockwise, the angle is decremented. Until Create turns through the specified angle, its state does not change, nor does it react to any inputs, serial or otherwise.

NOTE: This command resets the angle variable that is returned in Sensors packets 20, 2 and 6.

- Serial sequence: [157] [Angle high byte] [Angle low byte]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Wait Angle data bytes 1-2: 16-bit signed angle in degrees, high byte first (-32767 -32768)

Wait Event **Opcode: 158** **Data Bytes: 1 signed**

This command causes Create to wait until it detects the specified event. Until the specified event is detected, Create's state does not change, nor does it react to any inputs, serial or otherwise.

- Serial sequence: [158] [Event number]
- Available in modes: Passive, Safe, or Full
- Changes mode to: No Change
- Wait Event data byte 1: Signed event number (1 to 20 and -1 to -20)

To wait for the inverse of an event, send the negative of its number using two's complement notation. For example, to wait for no bumps, send the serial byte sequence [158] [-5], which is equivalent to [158] [251].

Wait Event: Unsigned Equivalent of Inverse

Event	Number	Unsigned Equivalent of Inverse
Wheel Drop	1	255
Front Wheel Drop	2	254
Left Wheel Drop	3	253
Right Wheel Drop	4	252
Bump	5	251
Left Bump	6	250
Right Bump	7	249
Virtual Wall	8	248
Wall	9	247
Cliff	10	246
Left Cliff	11	245
Front Left Cliff	12	244
Front Right Cliff	13	243
Right Cliff	14	242
Home Base	15	241
Advance Button	16	240
Play Button	17	239
Digital Input 0	18	238
Digital Input 1	19	237
Digital Input 2	20	236
Digital Input 3	21	235
OI Mode = Passive	22	234

iRobot Create Open Interface Sensor Packets

Create sends back one of 43 different sensor data packets, depending on the value of the packet data byte, when responding to a Sensors command, Query List command, or Stream command's request for a packet of sensor data bytes. Some packets contain groups of other packets. Some of the sensor data values are 16 bit values.

Most of the packets (numbers 7 – 42) contain the value of a single sensor or variable, which can be either 1 byte or 2 bytes. Two-byte packets correspond to 16-bit values, sent high byte first.

Some of the packets (0-6) contain groups of the single-value packets.

Wait Event: Unsigned Equivalent of Inverse

Group Packet ID	Packet Size	Contains packets
0	26 bytes	7 - 26
1	10 bytes	7 - 16
2	6 bytes	17 - 20
3	10 bytes	21 - 26
4	14 bytes	27 - 34
5	12 bytes	35 - 42
6	52 bytes	7 - 42

Bumps and Wheel Drops **Packet ID: 7** **Data Bytes: 1**
unsigned

The state of the bumper (0 = no bump, 1 = bump) and wheel drop sensors (0 = wheel raised, 1 = wheel dropped) are sent as individual bits.

Range: 0 - 31

Bit	7	6	5	4	3	2	1	0
Sensor	n/a	n/a	n/a	Wheeldrop Caster	Wheeldrop Left	Wheeldrop Right	Bump Left	Bump Right

Wall **Packet ID: 8** **Data Bytes: 1**
unsigned

The state of the wall sensor is sent as a 1 bit value (0 = no wall, 1 = wall seen).

Range: 0 – 1

Cliff Left **Packet ID: 9** **Data Bytes: 1**
unsigned

The state of the cliff sensor on the left side of Create is sent as a 1 bit value (0 = no cliff, 1 = cliff).

Range: 0 – 1

Cliff Front Left **Packet ID: 10** **Data Bytes: 1**
unsigned

The state of the cliff sensor on the front left of Create is sent as a 1 bit value (0 = no cliff, 1 = cliff).

Range: 0 – 1

Cliff Front Right **Packet ID: 11** **Data Bytes: 1**
unsigned

The state of the cliff sensor on the front right of Create is sent as a 1 bit value (0 = no cliff, 1 = cliff)

Range: 0 – 1

Cliff Right **Packet ID: 12** **Data Bytes: 1**
unsigned

The state of the cliff sensor on the right side of Create is sent as a 1 bit value (0 = no cliff, 1 = cliff)

Range: 0 – 1

Virtual Wall **Packet ID: 13** **Data Bytes: 1**
unsigned

The state of the virtual wall detector is sent as a 1 bit value (0 = no virtual wall detected, 1 = virtual wall detected). Note that the force field on top of the Home Base also trips this sensor.

Range: 0 – 1

Low Side Driver and Wheel Overcurrents **Packet ID: 14** **Data Bytes: 1**
unsigned

The state of the three Low Side driver and two wheel overcurrent sensors are sent as individual bits (0 = no overcurrent, 1 = overcurrent).

Driver	Current Limit
LD0	0.5A
LD1	0.5A
LD2	1.6A
Wheels	1.0A

Range: 0 - 31

Bit	7	6	5	4	3	2	1	0
Motor	n/a	n/a	n/a	Left Wheel	Right Wheel	LD2	LD0	LD1

Unused Bytes **Packet IDs: 15 - 16** **Data Bytes: 1**

Unused bytes: Two unused bytes are sent after the overcurrent byte when the requested packet is 0, 1, or 6. The value of the two unused bytes is always 0.

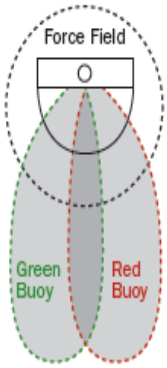
Range: 0

Infrared Byte **Packet ID: 17** **Data Bytes: 1**
unsigned

This value identifies the IR byte currently being received by iRobot Create. A value of 255 indicates that no IR byte is being received. These bytes include those sent by the Roomba Remote, the Home Base, Create robots using the Send IR command, and user-created devices.

Range: 0 – 255

Dock beam configuration



Sent by iRobot Device	Character Value	Character Name
Remote Control	129	Left
	130	Forward
	131	Right
	132	Spot
	133	Max
	134	Small
	135	Medium
	136	Large / Clean
	137	PAUSE
	138	Power
	139	arc-forward-left
	140	arc-forward-right
	141	drive-stop

Sent by iRobot Device	Character Value	Character Name
Scheduling Remote	142	Send All
	143	Seek Dock
Home Base	240	Reserved
	248	Red Buoy
	244	Green Buoy
	242	Force Field
	252	Red Buoy and Green Buoy
	250	Red Buoy and Force Field
	246	Green Buoy and Force Field
	254	Red Buoy, Green Buoy and Force Field

Buttons **Packet ID: 18** **Data Bytes: 1**
unsigned

The state of Create's Play and Advance buttons are sent as individual bits (0 = button not pressed, 1 = button pressed).

Range: 0 - 5

Bit	7	6	5	4	3	2	1	0
Button	n/a	n/a	n/a	n/a	n/a	Advance	n/a	Play

Distance **Packet ID: 19** **Data Bytes: 2**
signed

The distance that Create has traveled in millimeters since the distance it was last requested is sent as a signed 16-bit value, high byte first. This is the same as the sum of the distance traveled by both wheels divided by two. Positive values indicate travel in the forward direction; negative values indicate travel in the reverse direction. If the value is not polled frequently enough, it is capped at its minimum or maximum.

Range: -32768 – 32767

Angle **Packet ID: 20** **Data Bytes: 2**
signed

The angle in degrees that iRobot Create has turned since the angle was last requested is sent as a signed 16-bit value, high byte first. Counter-clockwise angles are positive and clockwise angles are negative. If the value is not polled frequently enough, it is capped at its minimum or maximum.

Range: -32768 – 32767

NOTE: Create uses wheel encoders to measure distance and angle. If the wheels slip, the actual distance or angle traveled may differ from Create's measurements.

Charging State **Packet ID: 21** **Data Bytes: 1**
unsigned

This code indicates Create's current charging state.

Range: 0 - 5

Code	Charging State
0	Not charging
1	Reconditioning Charging
2	Full Charging
3	Trickle Charging
4	Waiting
5	Charging Fault Condition

Voltage **Packet ID: 22** **Data Bytes: 2**
unsigned

This code indicates the voltage of Create's battery in millivolts (mV).

Range: 0 – 65535

Current **Packet ID: 23** **Data Bytes: 2**
signed

The current in milliamps (mA) flowing into or out of Create's battery. Negative currents indicate that the current is flowing out of the battery, as during normal running. Positive currents indicate that the current is flowing into the battery, as during charging.

Range: -32768 – 32767

Battery Temperature **Packet ID: 24** **Data Bytes: 1**
signed

The temperature of Create's battery in degrees Celsius.

Range: -128 – 127

Battery Charge **Packet ID: 25** **Data Bytes: 2**
unsigned

The current charge of Create's battery in milliamp-hours (mAh). The charge value decreases as the battery is depleted during running and increases when the battery is charged. Note that this value will not be accurate if you are using the alkaline battery pack.

Range: 0 – 65535

Battery Capacity **Packet ID: 26** **Data Bytes: 2**
unsigned

The estimated charge capacity of Create's battery in milliamp-hours (mAh). Note that this value is inaccurate if you are using the alkaline battery pack.

Range: 0 – 65535

Wall Signal **Packet ID: 27** **Data Bytes: 2**
unsigned

The strength of the wall sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Cliff Left Signal **Packet ID: 28** **Data Bytes: 2**
unsigned

The strength of the left cliff sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Cliff Front Left Signal **Packet ID: 29** **Data Bytes: 2**
unsigned

The strength of the front left cliff sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Cliff Front Right Signal **Packet ID: 30** **Data Bytes: 2**
unsigned

The strength of the front right cliff sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Cliff Right Signal **Packet ID: 31** **Data Bytes: 2**
unsigned

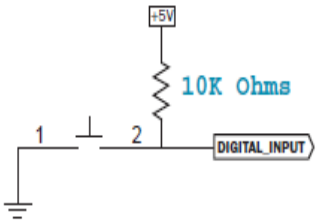
The strength of the right cliff sensor's signal is returned as an unsigned 16-bit value, high byte first.

Range: 0-4095

Cargo Bay Digital Inputs **Packet ID: 32** **Data Bytes: 1**
unsigned

The state of the digital inputs on the 25-pin Cargo Bay Connector are sent as individual bits (0 = low, 1 = high (5V)). Note that the Baud Rate Change pin is active low; it is high by default.

Example Schematic



Range: 0 - 31

Bit	7	6	5	4	3	2	1	0
Button	n/a	n/a	n/a	Device Detect /Baud Rate Change (pin 15)	Digital Input 3 (pin 6)	Digital Input 2 (pin 18)	Digital Input 1 (pin 5)	Digital Input 0 (pin 17)

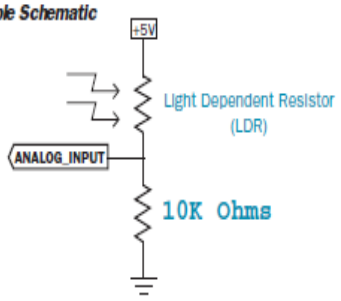
Device Detect pin can be used to change Baud Rate. When device detect/baud rate change Bit is low, the Baud Rate is 19200. Otherwise it it 57600

Cargo Bay Analog Signal **Packet ID: 33** **Data Bytes: 2**
unsigned

The 10-bit value of the analog input on the 25-pin Cargo Bay Connector is returned, high byte first. 0 = 0 volts; 1023 = 5 volts. The analog input is on pin 4.

Range: 0 - 1023

Example Schematic



Charging Sources Available **Packet ID: 34** **Data Bytes: 1**
unsigned

iRobot Create's connection to the Home Base and Internal Charger are returned as individual bits, as below.

Range: 0 - 3
1 = charging source present and powered; 0 = charging source not present or not powered.

Bit	7	6	5	4	3	2	1	0
Charging Source	n/a	n/a	n/a	n/a	n/a	n/a	Home Base	Internal Charger

OI Mode **Packet ID: 35** **Data Bytes: 1**
unsigned

Create's connection to the Home Base and Internal Charger are returned as individual bits, as below.

Range: 0 - 3

Number	Mode
0	Off
1	Passive
2	Safe
3	Full

Song Number	Packet ID: 36	Data Bytes: 1
		unsigned

The currently selected OI song is returned.

Range: 0-15

Song Playing	Packet ID: 37	Data Bytes: 1
		unsigned

The state of the OI song player is returned. 1 = OI song currently playing; 0 = OI song not playing.

Range: 0-1

Number of Stream Packets	Packet ID: 38	Data Bytes: 1
		unsigned

The number of data stream packets is returned.

Range: 0-43

Requested Velocity	Packet ID: 39	Data Bytes: 2
		signed

The velocity most recently requested with a Drive command is returned as a signed 16-bit number, high byte first.

Range: -500 - 500 mm/s

Requested Radius	Packet ID: 40	Data Bytes: 2
		signed

The radius most recently requested with a Drive command is returned as a signed 16-bit number, high byte first.

Range: -32768 - 32767 mm

Requested Right Velocity	Packet ID: 41	Data Bytes: 2
		signed

The right wheel velocity most recently requested with a Drive Direct command is returned as a signed 16-bit number, high byte first.

Range: -500 - 500 mm/s

Requested Left Velocity	Packet ID: 42	Data Bytes: 2
		signed

The left wheel velocity most recently requested with a Drive Direct command is returned as a signed 16-bit number, high byte first.

Range: -500 - 500 mm/s

iRobot Create Open Interface Commands Quick Reference

Create OI Commands Quick Reference Table

Command	*Opcode*	*Data Bytes: 1*	*Data Bytes: 2*	*Data Bytes: 3*	*Data Bytes: 4*	*Etc.*
Start	128					
Baud	129	Baud Code: (0 – 11)				
Control	130					
Safe	131					
Full	132					
Spot	134					
Cover	135					
Demo	136	Demos (-1 - 9)				
Drive	137	Velocity (-500 – 500 mm/s)		Radius (-2000 – 2000 mm)		
Low Side Drivers	138	Output Bits (0 – 7)				
LEDs	139	LED Bits (0 – 10)	Power LED Color (0 – 255)	Power LED Intensity (0 – 255)		
Song	140	Song Number (0 - 15)	Song Length (1 - 16)	Note Number 1 (31 – 27)	Note Duration 1 (0 - 255)	Note Number 2, etc.
Play	141	Song Number: (0 – 15)				
Sensors	142	Packet ID: (0 – 42)				
Cover and Dock	143					
PWM Low Side Drivers	144	Low Side Driver 2 Duty Cycle (0 - 128)	Low Side Driver 1 Duty Cycle (0 - 128)	Low Side Driver 0 Duty Cycle (0 - 128)		
Drive Direct	145	Right wheel velocity (-500 – 500 mm/s)		Left wheel velocity (-500 – 500 mm/s)		
Digital Outputs	147	Output Bits (0 –7)				
Stream	148	Number of Packets	Packet ID 1 (0 – 42)	Packet ID 2, etc.		
Query List	149	Packet ID 1 (0 – 42)	Packet ID 2, etc.			
Pause/Resume Stream	150	Range: 0-1				
Send IR	151	Byte (0 - 255)				
Script	152	Script Length: (1 – 100)	Command Opcode 1	Command Data Byte 1, etc.	Command Opcode 2	Etc.
Play Script	153					
Show Script	154					
Wait Time	155	Time (0 – 255 seconds/10)				
Wait Distance	156	Distance (-32767 - 32768 mm)				
Wait Angle	157	Angle (-32767 - 32768 degrees)				
Wait Event	158	Event ID (1 to 20 and -1 to -20)				

Baud Code (0 – 11)

Baud Code	Baud Rate in BPS
0	300
1	600
2	1200
3	2400
4	4800
5	9600
6	14400
7	19200
8	28800
9	38400
10	57600
11	115200

LEDs Data Byte 1: LED Bits (0 – 10)

Advance and Play use green LEDs: 0 = off, 1 = on

Bit	7	6	5	4	3	2	1	0
LED	n/a	n/a	n/a	n/a	Advance	n/a	Play	n/a

LEDs Data Bytes 2 and 3: Power LED Color and Intensity (0 - 255)

Power uses a bicolor (red/green) LED. The intensity and color of this LED can be controlled with 8-bit resolution.

LEDs data byte 2: Power LED Color (0 – 255)

0 = green, 255 = red. Intermediate values are intermediate colors (orange, yellow, etc).

LEDs data byte 3: Power LED Intensity (0 – 255)

0 = off, 255 = full intensity. Intermediate values are intermediate intensities.

Digital Outputs Data Byte 1: Output Bits (0 –7)

0 = low (0V); 1 = high (5V).

The digital outputs can provide up to 20 mA of current.

Bit	7	6	5	4	3	2	1	0
Output						digital output 2 (pin 20)	digital output 1 (pin 7)	digital output 0 (pin 19)

Low Side Drivers Data Byte 1: Output bits (0 – 7)

0 = off, 1 = on at 100% pwm duty cycle

Bit	7	6	5	4	3	2	1	0
Output						Low Side Driver 2	Low Side Driver 1	Low Side Driver 0

iRobot

iRobot Create Open Interface Sensor Packets Quick Reference

Create sends back one of 43 different sensor data packets in response to a Sensors command, depending on the value of the packet ID data byte. Some packets contain groups of other packets. Group packetsr send their component values in sequential order. Some of the sensor data values are 16 bit values. These values are sent as two bytes, high byte first.

Group Packet Sizes and Contents

Group Packet ID	Packet Size	Contains Packets
0	26 bytes	7 - 26
1	10 bytes	7 - 16
2	6 bytes	17 - 20
3	10 bytes	21 - 26
4	14 bytes	27 - 34
5	12 bytes	35 - 42
6	52 bytes	7 - 42

Group Packet Sizes and Contents

Packet Membership							Name	Bytes	Value Range	Units
0	1					6	7 Bumps and Wheel Drops	1	0 - 31	
							8 Wall	1	0 - 1	
							9 Cliff Left	1	0 - 1	
							10 Cliff Front Left	1	0 - 1	
							11 Cliff Front Right	1	0 - 1	
							12 Cliff Right	1	0 - 1	
							13 Virtual Wall	1	0 - 1	
							14 Overcurrents	1	0 - 31	
							15 Unused	1	0	
							16 Unused	1	0	
		2					17 IR Byte	1	0 - 255	
							18 Buttons	1	0 - 15	
							19 Distance	2	-32768 - 32767	mm
							20 Angle	2	-32768 - 32767	mm
			3				21 Charging State	1	0 - 5	
							22 Voltage	2	0 - 65535	mV
							23 Current	2	-32768 - 32767	mA
							24 Battery Temperature	1	-128 - 127	degrees Celsius
							25 Battery Charge	2	0 - 65535	mAh
							26 Battery Capacity	2	0 - 65535	mAh

Packet Membership							Name	Bytes	Value Range	Units
				4			27 Wall Signal	2	0 - 4095	
							28 Cliff Left Signal	2	0 - 4095	
							29 Cliff Front Left Signal	2	0 - 4095	
							30 Cliff Front Right Signal	2	0 - 4095	
							31 Cliff Right Signal	2	0 - 4095	
							32 User Digital Inputs	1	0 - 31	
							33 User Analog Input	2	0 - 1023	
							34 Charging Sources Available	1	0 - 3	
				5			35 OI Mode	1	0 - 3	
							36 Song Number	1	0 - 15	
							37 Song Playing	1	0 - 1	
							38 Number of Stream Packets	1	0 - 42	
							39 Velocity	2	-500 - 500	mm/s
							40 Radius	2	-32768 - 32767	mm
							41 Right Velocity	2	-500 - 500	mm/s
							42 Left Velocity	2	-500 - 500	mm/s

Charging State Codes

Code	Charging State
0	Not charging
1	Reconditioning Charging
2	Full Charging
3	Trickle Charging
4	Waiting
5	Charging Fault Condition

Bumps and Wheel Drops

Bit	7	6	5	4	3	2	1	0
Sensor	n/a	n/a	n/a	Wheel-drop Caster	Wheel-drop Left	Wheel-drop Right	Bump Left	Bump Right

Low Side Driver and Wheel Overcurrents

Bit	7	6	5	4	3	2	1	0
Motor	n/a	n/a	n/a	Left Wheel	Right Wheel	Low Side Driver 2	Low Side Driver 0	Low Side Driver 1

Buttons

Bit	7	6	5	4	3	2	1	0
Button	n/a	n/a	n/a	n/a	n/a	Advance	n/a	Play

Cargo Bay Digital Inputs

Bit	7	6	5	4	3	2	1	0
Button	n/a	n/a	n/a	Baud Rate Change (pin 15)	Digital Input 3 (pin 6)	Digital Input 2 (pin 18)	Digital Input 1 (pin 5)	Digital Input 0 (pin 17)

Charging Sources Available

Bit	7	6	5	4	3	2	1	0
Charging Source	n/a	n/a	n/a	n/a	n/a	n/a	Home Base	Internal Charger

OI Mode

Number	Mode
0	Off
1	Passive
2	Safe
3	Full

ANEXO 2

VISUAL PROGRAMING LANGUAGE

1. INTRODUCCIÓN

Microsoft Visual Programming Language (VPL) es un entorno de desarrollo de aplicaciones diseñada bajo un modelo de programación gráfica, basada en el flujo de datos en vez del típico flujo de control que se encuentra en la programación convencional, ver Figura 1. En lugar de una serie de comandos imperativos ejecutados secuencialmente, un programa de flujo de datos se parece más a una serie de trabajadores en una línea de ensamblaje quienes realizan su tarea conforme el material llega a cada una de sus líneas de producción. Como resultado de esto VPL es una herramienta adecuada para la programación de una variedad de escenarios que hagan uso del procesamiento concurrente o distribuido.

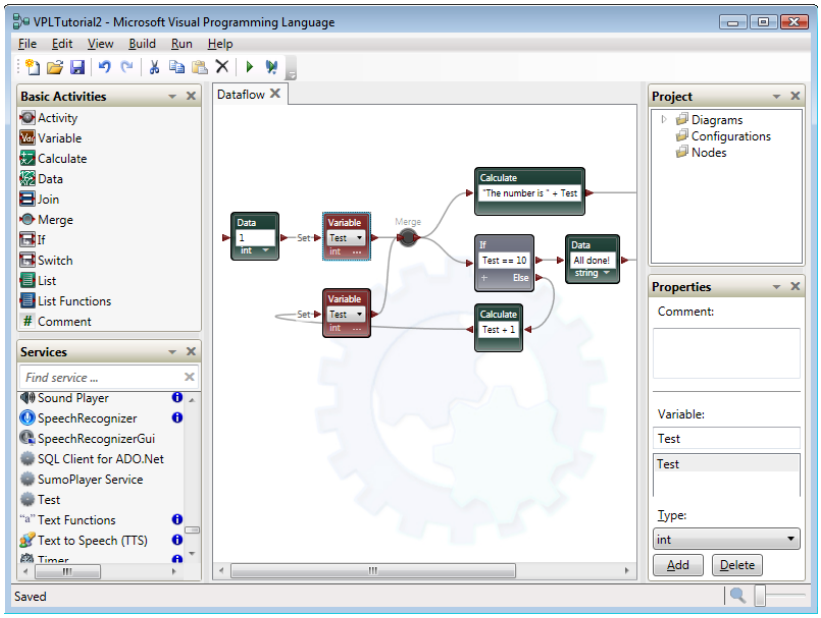


Figura 1. Ventana Principal MRDS 2008 R3

Un flujo de datos en VPL consiste de una conexión secuencial de actividades que están representadas como bloques con entradas y salidas que pueden ser conectadas a otros bloques de actividad, ver Figura 2.

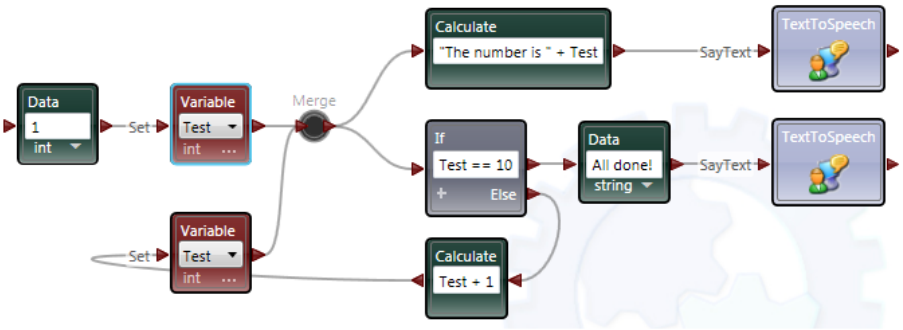


Figura 2. Ejemplo Diagrama VPL

Las actividades pueden representar funciones de control o procesamiento del flujo de datos, a través de los servicios DSS pre-construidos o mediante las actividades que uno mismo puede crear en VPL. Las conexiones entre las actividades intercambian información a manera de mensajes. La aplicación resultante es por lo tanto referida como una orquestación de flujo de datos.

Las actividades pueden también incluir construcciones de otras actividades. Esto permite componer nuestras propias actividades mediante la combinación de otras ya existentes y a continuación volver a utilizar la composición como un bloque de construcción. En este sentido una aplicación creada en VPL es en sí misma una actividad.

Un bloque de actividad incluye el nombre de la actividad y sus puntos de conexión. También puede incluir gráficos que ilustren su propósito, elementos de interfaz de usuario como cuadros de texto que pueden permitir al usuario introducir valores, tareas o transformaciones de datos utilizados en una actividad, ver Figura 3

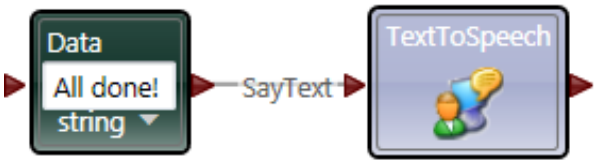


Figura 3. Bloques de Actividades y Servicios

Las actividades son conectadas a través de pines de conexión. El pin de conexión del lado izquierdo representa el punto de conexión para los mensajes entrantes y el pin sobre el lado derecho representa el punto de conexión para los mensajes salientes.Una actividad recibe mensajes que contienen datos a través de sus pines de conexión entrantes.

Un bloque de actividad, habilita y procesa los mensajes de datos entrantes tan pronto se recibe un mensaje entrante valido. Para que los datos sean transmitidos a través de la salida, la actividad de recepción debe replicar los datos y ponerlos sobre su conexión de salida.

Una actividad puede tener múltiples pines de conexión entrantes, cada uno con su propio conjunto de pines de conexión salientes. Los pines de entrada son representados como flechas que apuntan al bloque. Los pines de conexión de salida pueden ser de dos tipos:

Result output: También conocida como salida de respuesta.
Notification output: Conocida como salida de evento o publicación.

La salida de respuesta se muestra como una flecha que apunta hacia afuera del bloque, mientras que la salida de evento o publicación se muestran como pines de conexión redondos. Ver Figura 4.

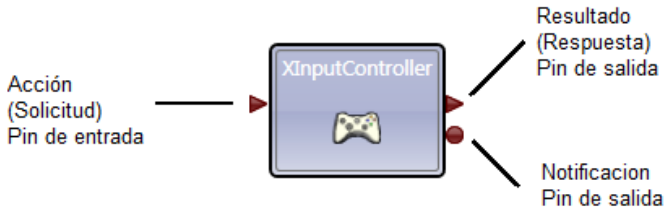


Figura4. Pines de conexión de una actividad

La salida de respuesta es usada en situaciones en donde el mensaje de datos saliente es enviado como el resultado de una acción específica de entrada (o solicitud) de mensajes. La salida de evento o publicación puede enviar información resultante de un mensaje entrante, pero más frecuentemente envía un mensaje como el resultado de un cambio interno de su estado.

Las conexiones de respuesta y notificación también difieren en cómo estos entregan los mensajes de datos. Dada una conexión de respuesta esta requiere de un mensaje entrante, una vez que la solicitud es recibida y procesada, el mensaje resultante es enviado y el proceso se da por finalizado. Sin embargo, las salidas de notificación pueden generar mensajes varias veces. Como conclusión las salidas de respuesta son usadas típicamente para consultar el estado actual de una actividad. Por otro lado las salidas de evento o publicación son usadas para proporcionar actualizaciones continuas sobre el estado de una actividad, eliminando la necesidad de solicitar varias veces el estado de una actividad.

2. PRIMEROS PASOS

VPL es un lenguaje de programación basado en el flujo datos, donde se puede crear un programa mediante un gráfico o diagrama de actividades (bloques que representan servicios DSS u otras actividades) vinculados entre sí mediante conexiones que definen como transitaran los datos entre las actividades. Uno o más diagramas constituyen lo que es llamado un **programa** en otros lenguajes de programación, aunque en VPL esto es referido como un **proyecto**.

Los mensajes transitan a lo largo de las conexiones entre las actividades y pueden llevar datos o simplemente actuar para controlar el flujo lógico del diagrama. En VPL tanto el flujo de datos como el de control pueden ejecutarse en paralelo porque el soporte en tiempo de ejecución DSS provee una base para la ejecución asíncrona.

2.1 VPL USER INTERFACE

La ventana de inicio de VPL muestra un **menubar** y un **toolbar** para acceder a los comandos, un **área central o de trabajo** para construir la aplicación robótica, contiene además varias ventanas secundarias que incluyen **toolboxes** que muestran una lista de actividades (para el control de flujo de datos) y servicios (código pre-escrito que realiza funciones genéricas o especificas), contenido actual del proyecto (esto es diagramas y archivos de configuración) y una sección para editar las propiedades de los elementos seleccionados, en la Figura 5 se muestran cada una de las ventanas descritas anteriormente.

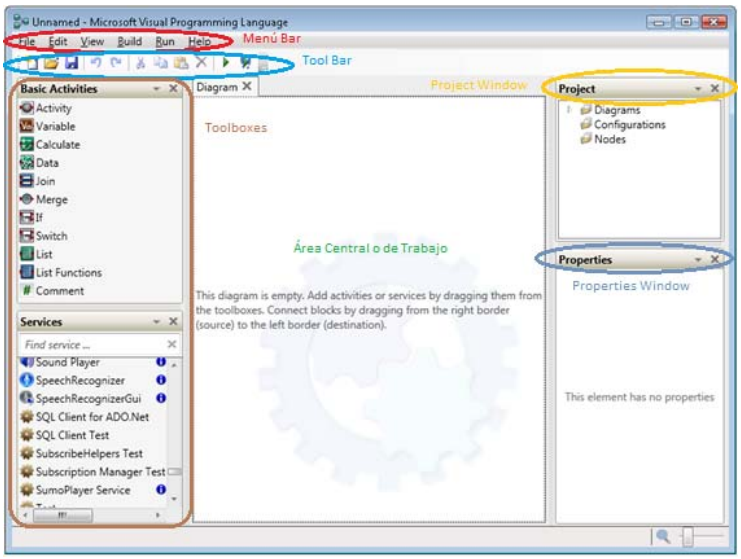


Figura 5. Interfaz de Usuario Visual Programing Language

Las ventanas de toolbox, project window, y properties windows son desmontables y movibles, estos pueden ser reordenados dentro de la ventana principal de VPL. Se puede tener varios diagramas dentro de un único programa, que pueden ser accedidas a través de pestañas ubicadas en la parte superior del diagrama.

Se puede ocultar o volver a mostrar una ventana a través del comando Toolboxes sobre el Menú **View.** Por lo que si se cierra inadvertidamente un toolbox haciendo click sobre la 'x' en la parte superior derecha de cada ventana, se puede volver nuevamente abrirla.

Para construir un diagrama de flujo de datos se debe arrastrar y soltar una actividad o servicio desde el cuadro de herramientas **Basic Activities** o **Services** hacia el área de trabajo del diagrama, también es posible añadir una actividad haciendo doble click sobre el bloque a utilizarse. Al pasar el mouse sobre un bloque de actividad o servicio este me brinda una descripción acerca de la manera en que puede ser utilizado. Si el elemento también incluye un icono **"i"**, este también proporciona información adicional. Al hacer click sobre el icono este abre un navegador web para mostrar la página con la correspondiente información del elemento.

Cuando se tiene varios bloques sobre el diagrama, estos se conectan haciendo click en el pin de conexión de salida del bloque (en el lado derecho del bloque) y arrastrando al pin de conexión de entrada del otro bloque (lado izquierdo del bloque).

- **BASIC ACTIVITIES**

La ventana del cuadro de herramientas **Basic Activities,** incluye un bloque de actividad para definir un dato (**Data**), controlar el flujo de datos (**If** y **Switch**), combinar mensajes (**Join** y **Merge**), realizar cálculos (**Calculate**) y almacenar

resultados en variables (**Variable**). Además incluye una actividad **Comment** que permite colocar un bloque de texto en donde se pueden realizar comentarios sobre el diagrama, ver Figura 6.

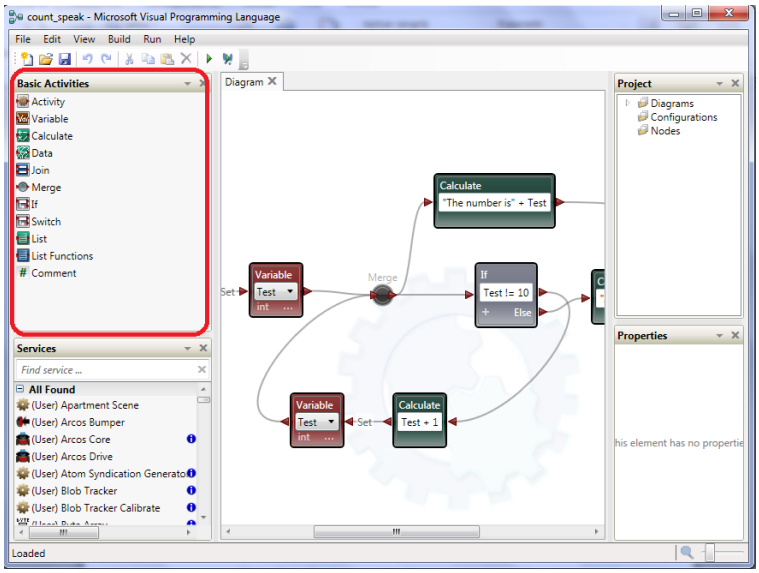


Figura 6. Ventana Actividades Básicas

VPL también incluye un bloque **Activity** que permite crear actividades personalizadas. Estas aceptan información, la procesan y arrojan un resultado. Es posible pensar que una actividad personalizada es como una subrutina si se está familiarizado con otros lenguajes de programación, la cual me permite realizar la misma serie de pasos en varias partes del diagrama, proporcionando un camino fácil para escribir el código una sola vez y reutilizarlo varias veces.

- **SERVICES**

El toolbox **Services** muestra los servicios disponibles, incluidos al momento de la instalación del programa. La lista puede ser expandida mediante la adición de servicios DSS a la carpeta “bin”, ver Figura 7.

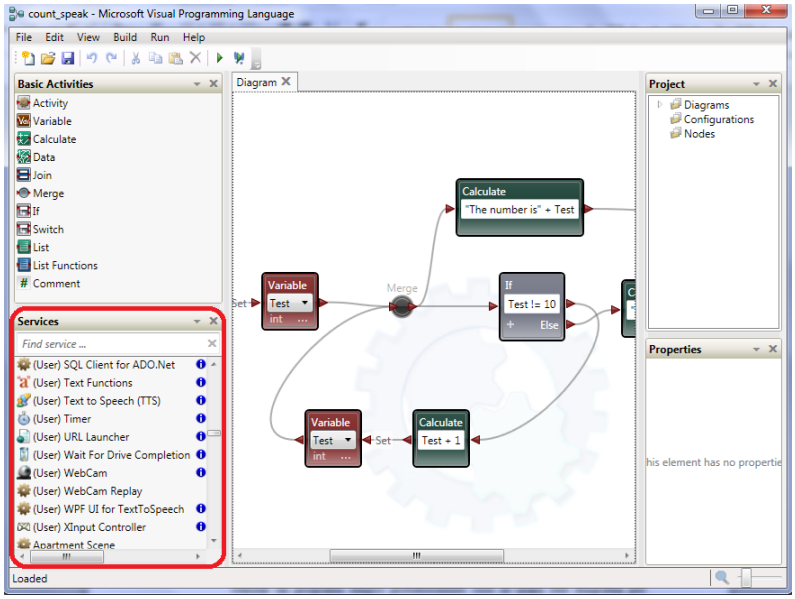


Figura7. Ventana de Servicios

Algunos de los servicios disponibles en el toolbox, no son destinados para su uso en VPL. En particular cualquier servicio que lleve por nombre “Tutorial” es una aplicación independiente que no puede ser utilizada en el diagrama de VPL. Del mismo modo si la edición de MRDS incluye el simulador, los servicios que empiecen por “Simulation” no son adecuados para ser usados en VPL.

- **DIAGRAM WINDOW**

El primer diagrama en un proyecto es siempre llamado **Diagram.** Para realizar un programa simple probablemente solo se usara este diagrama por defecto, ver Figura 8.

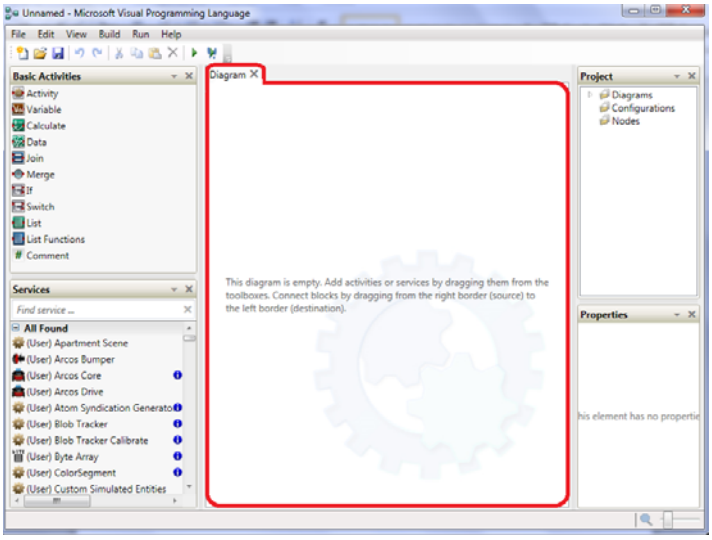


Figura 8. Diagram Window

Se puede crear una nuevo diagrama desde el menú File seleccionando Add Diagram. Agregar diagramas es una forma de estructurar el proyecto en piezas modulares, haciendo que el proyecto sea más manejable. Todos y cada uno de los diagramas que forman parte del proyecto corren en paralelo. Cuando se crea un nuevo diagrama una nueva pestaña aparecerá en la parte superior de la ventana de VPL. En ella se pueden adherir bloques de actividad o servicio.

- **PROJECT WINDOW**

La ventana **Project** muestra una lista de los diagramas y archivos de configuración que se incluyen en el proyecto. Para poder acceder a un diagrama ir a la ventana "Project", ubicarse en la pestaña "Diagrams" y abrirla, a continuación dar doble click sobre el nombre del diagrama o hacer click derecho y seleccionar abrir, ver Figura 9.

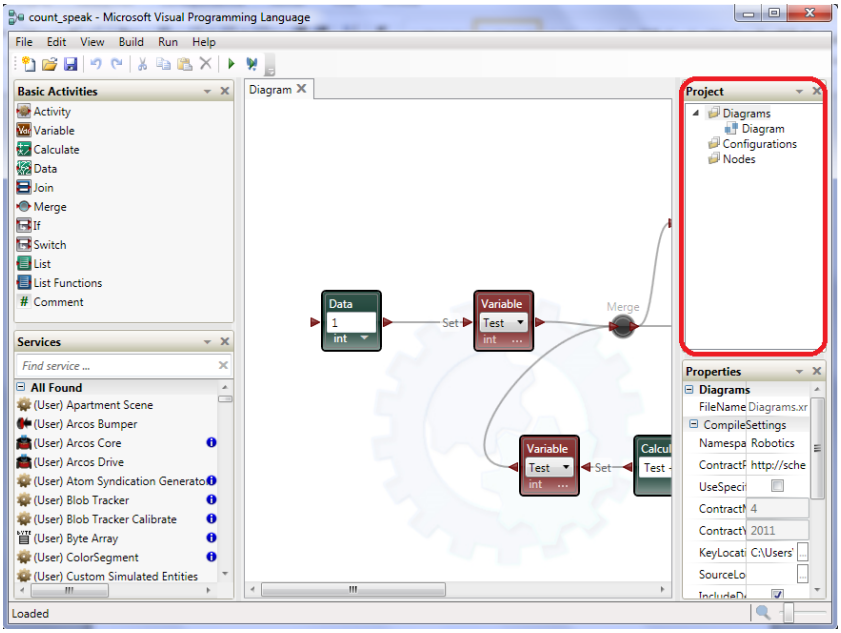


Figura 9. Project Windows

Para añadir más diagramas al proyecto dirigirse a la ventana "Project" y en la pestaña "Diagrams" dar click derecho y seleccionar "Add Diagram". Otra forma es desde el menú **File**. Cada vez que se adhiera un nuevo diagrama, su nombre será "Diagram" o si ya está siendo utilizado, VPL añadirá un número al final del nombre para hacer de este un archivo único.Para cambiar el nombre de un diagrama, hacer click sobre su entrada dentro de la ventana **Project** y editar el nombre en la ventana **Propiedades.** Los nombres pueden contener únicamente letras, números y el carácter de subrayado. Estos siempre deberán empezar con una letra. Hay que tratar de dar nombres significativos a los diagramas para poder recordar la función que desempeñan.

- **PROPERTIES WINDOW**

La ventana **Properties** muestra las propiedades del elemento actualmente seleccionado. Esto permite configurar una conexión de actividades (incluyendo los servicios) o diagrama. Para cambiar las propiedades de una actividad, seleccionar

la actividad mediante un click y las propiedades del bloque aparecerán por defecto dentro de la ventana **Properties,** ver Figura 10.

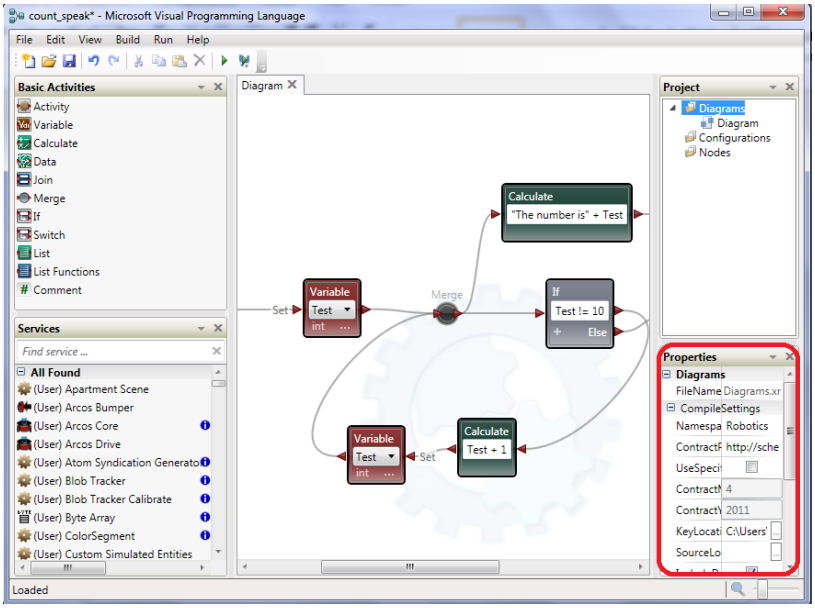


Figura 10. Properties Windows

3. CREAR UN PROYECTO

Cuando se inicia VPL por primera vez para iniciar con un nuevo proyecto, a través del menú **File** se selecciona **New** y VPL mostrara automáticamente un nuevo diagrama. Para desarrollar una aplicación robótica se puede arrastrar y soltar los elementos desde las cajas de herramienta **Basic Activities** o **Services** o hacer doble click sobre ellos, para de esta manera colocarlos dentro del diagrama. Ver Figura 11.

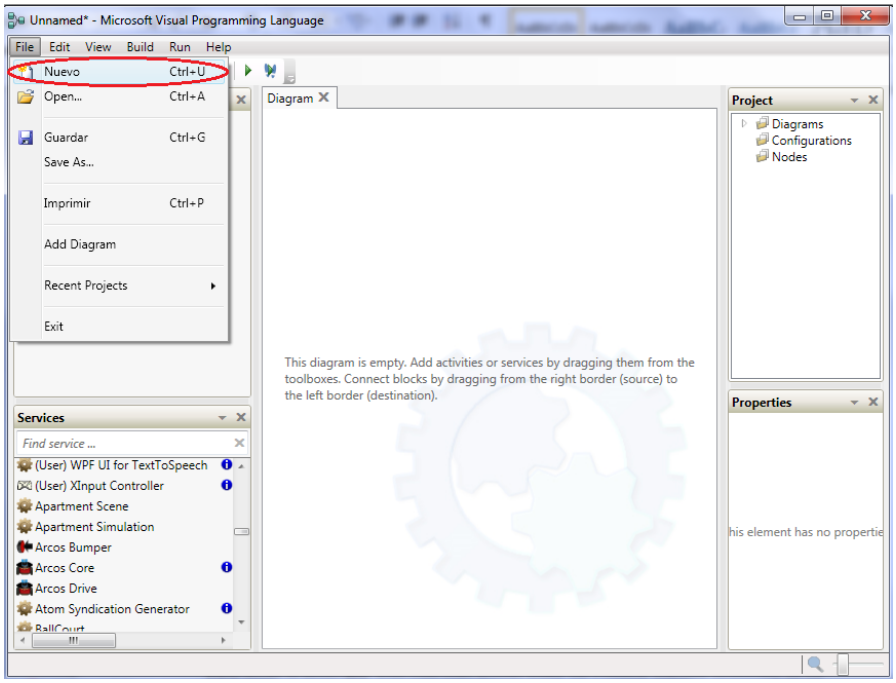


Figura 11. Creación de diagrama

Si una actividad o servicio requiere de valores que el programador debe establecer, se debe seleccionar el bloque haciendo un click sobre él, y configurarlo desde la ventana **Properties**. Algunas actividades pre-construidas como por ejemplo **Data**, **Variable**, **If**, **Switch**, **Calculate** y **Comment**, permite establecer valores directamente en el bloque, ver Figura 12.

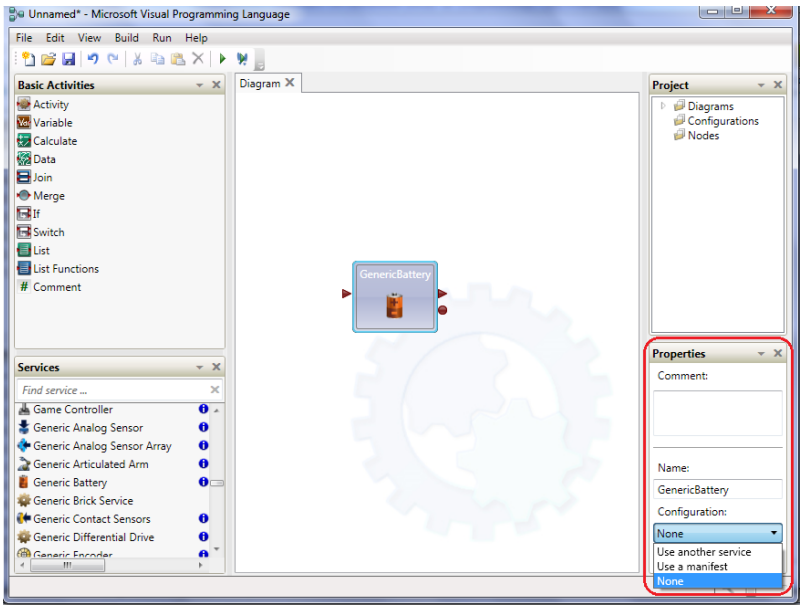


Figura 12. Configuración inicial de una actividad o servicio

Si se arrastra un servicio que ya existe desde el cuadro de herramientas **Services** hacia el diagrama, aparecerá un mensaje preguntando si desea crear una instancia del servicio o crear una referencia del servicio existente, ver Figura 13.

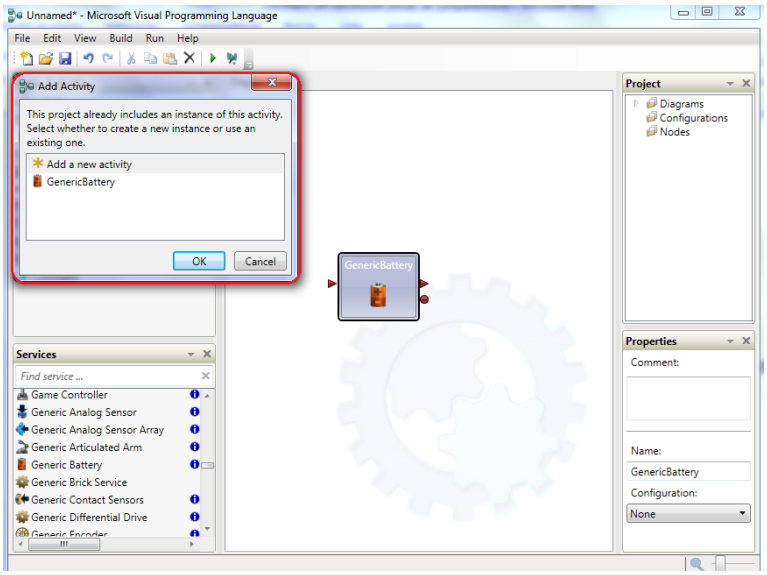


Figura 13. Instanciación o referencia de un nuevo servicio

Para crear una conexión de flujo de datos desde una actividad o servicio hacia otro, simplemente arrastramos un conector desde el pin de respuesta o notificación de un bloque al pin de conexión de entrada de otro bloque.

3.1. GUARDAR UN DIAGRAMA

Se puede guardar los cambios hechos en el diagrama únicamente seleccionando el comando **Save** desde el menú **File**. Este comando guarda todos los cambios actuales realizados en los diagramas y configura los archivos para el proyecto (archivados dentro del Project Window). Además se puede usar el comando **Save As** para guardar el proyecto bajo un nombre diferente, Ver Figura 14.

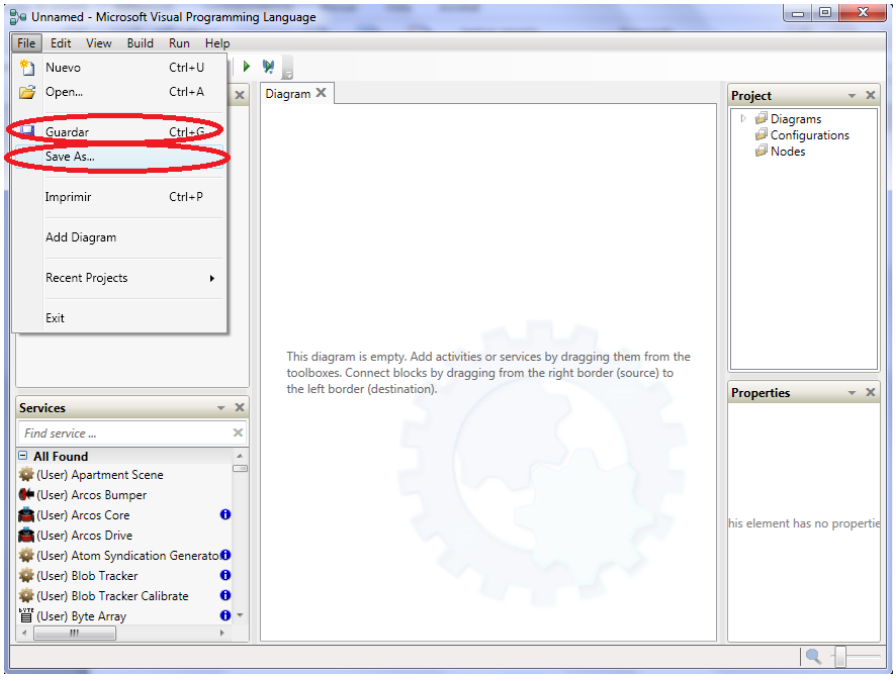


Figura 14. Guardar Diagrama

3.2. CARGAR UN DIAGRAMA

Para cargar un diagrama o más específicamente el proyecto que contiene el diagrama hay que usar en comando **Open** desde el menú **File**. Esto reemplazara los diagramas existentes que estén abiertos. Si no se ha guardado las modificaciones en el esquema existente, primero se preguntara si se desea guardar los cambios realizados. VPL realiza un seguimiento de los proyectos recientemente guardados y permite seleccionarlos desde el comando **Recent Project** desde el menú **File,** ver Figura 15.

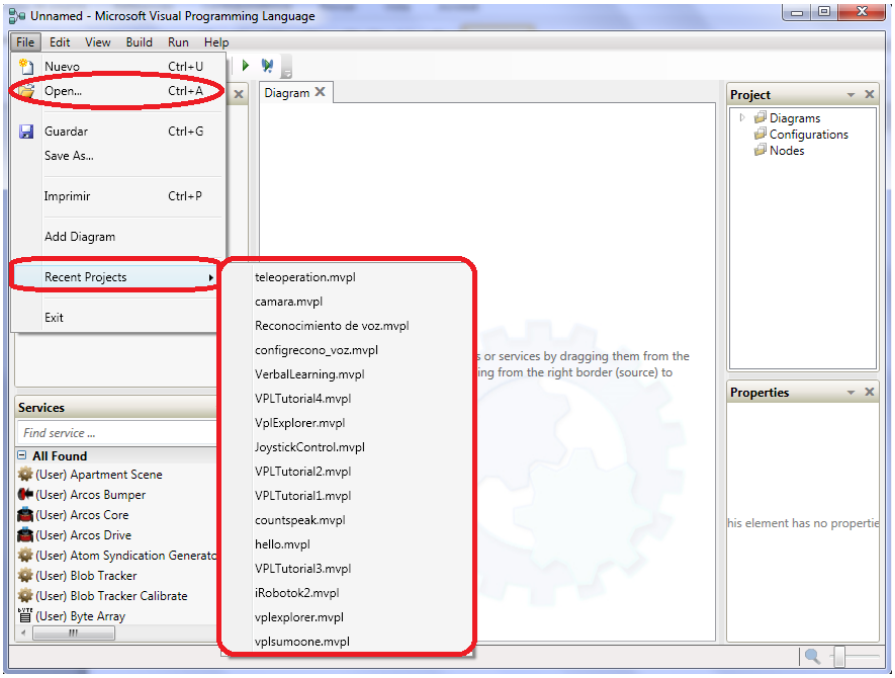


Figura 15. Cargar Diagrama

3.3. IMPRIMIR UN DIAGRAMA

Para poder imprimir un diagrama se debe seleccionar el comando **Print** desde el menú **File**. Un dialogo de impresión permitirá seleccionar la impresora y las opciones de impresión, ver Figura 16

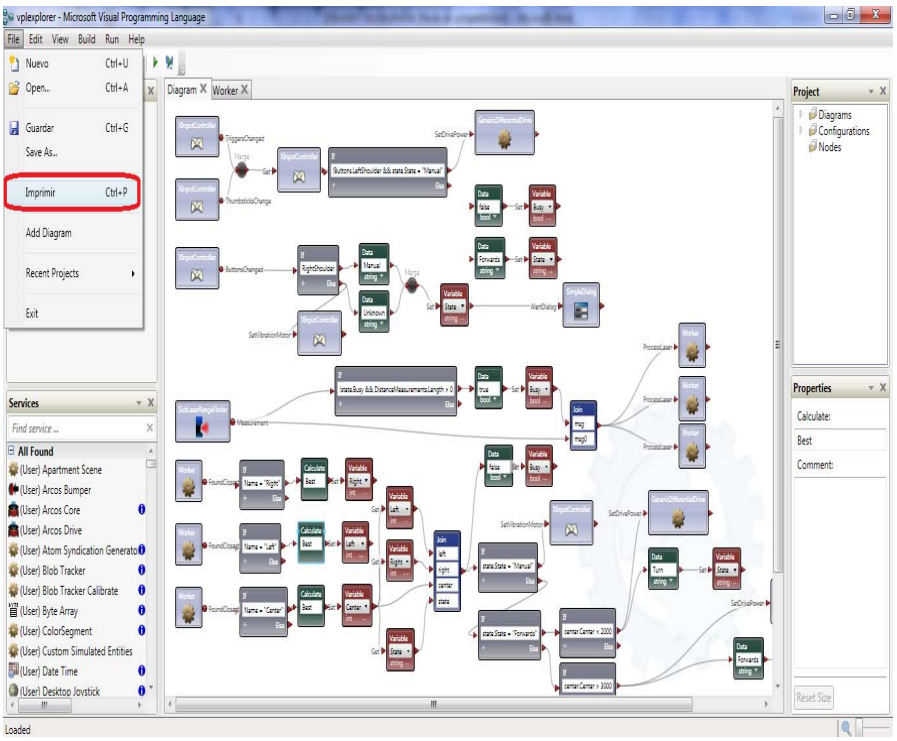


Figura 16. Imprimir Diagrama

3.4. SALIR DE VPL

En cualquier momento se puede abandonar el proyecto de VPL, seleccionando el comando **Exit** desde el menú **File**. Si no se ha procedido a guardar las cambios hechos sobre el diagrama, un cuadro de dialogo aparecerá preguntando por dicha opción, ver Figura 17.

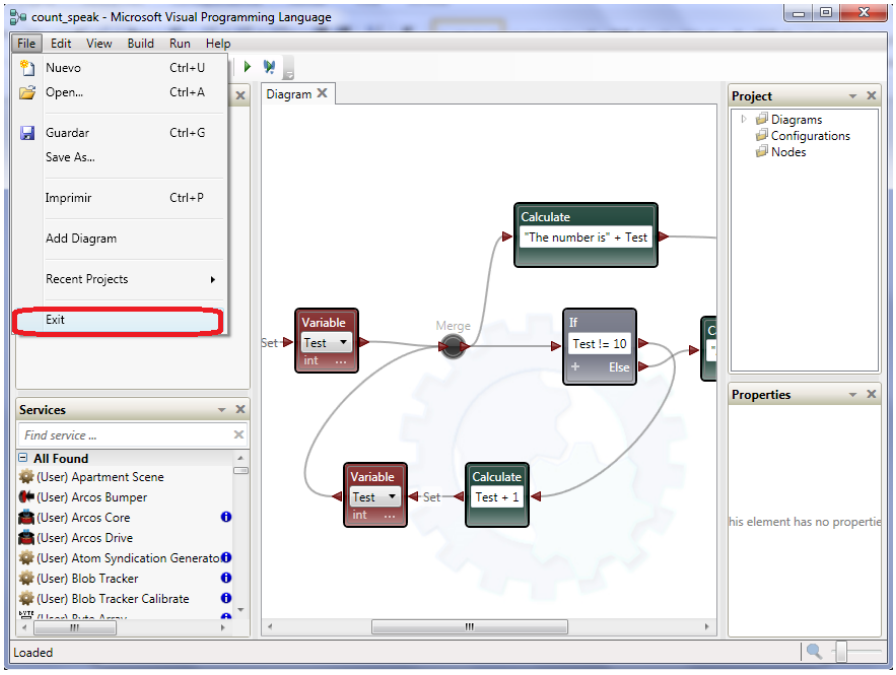


Figura 17. Salir de Visual Programing Language

4. CONEXIÓN ENTRE ACTIVIDADES

Las conexiones hechas entre los diferentes bloques de actividad, definen los mensajes de datos que se comparten. Para conectar dos bloques de actividad, se debe arrastrar desde el pin de conexión de salida (respuesta o notificación) de una actividad al pin de conexión de entrada de otra actividad. Si existe varias formas de conectar salidas a entradas, un cuadro de dialogo llamado **Connections** aparecerá. Si no existe opciones de conexión, el cuadro de dialogo no aparecerá. A menudo, las opciones disponibles para la salida de una actividad dependen del tipo de conexión de entrada que se establezca.

El cuadro de diálogo **Connections** permitirá definir cuál de las entradas y salidas se desea conectar. Ver Figura 18.

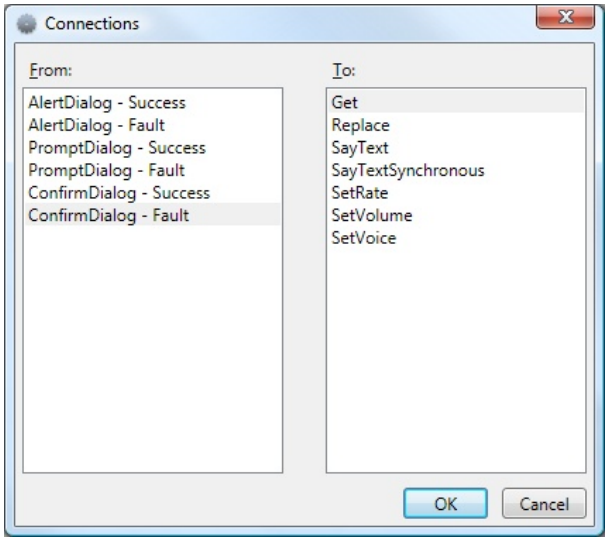


Figura 18. Cuadro de Dialogo Connections

Para seleccionar el tipo de enlace, seleccione una conexión de salida de la lista de la izquierda (**From**) y una acción de entrada de la lista de la derecha (**To**) y haga clic en Aceptar.

Alguna veces las conexiones entre las actividades también requieren seleccionar los datos que van a fluir a través de la conexión. Cuando esto sucede un cuadro de dialogo llamado **Data Connections** aparecerá y mostrara el tipo de datos que maneja el remitente del mensaje sobre el lado izquierdo, y los datos que maneja el destinatario del lado derecho. Dentro de las opciones para los datos de entrada, estos pueden incluir: valores predeterminados (cero para datos de tipo numérico, false para tipos booleanos y null para otro tipo de datos), valores de datos y valores de sub-datos, ver Figura 19.

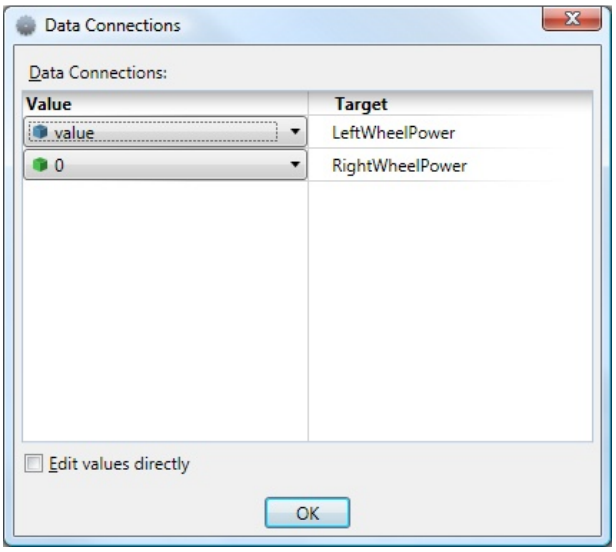


Figura 19. Cuadro de Dialogo Data Connections

En la parte inferior del cuadro de diálogo hay una casilla llamada **"Edit values directly"**. Esta característica se puede utilizar cuando se requiere manejar expresiones más complejas, o si no se puede encontrar un valor en particular desde la lista desplegable.

Para cambiar la conexión de una actividad a otra, primero hay que seleccionar el enlace de la conexión y arrastrar uno de los extremos que se desea volver a conectar hacia el bloque seleccionado. También se puede cambiar la configuración de las conexiones o datos mediante el menú contextual emergente para el enlace de la conexión.

Las actividades pueden soportar varias solicitudes, pero cada bloque puede únicamente aceptar una conexión de entrada al mismo tiempo. Si ya se ha conectado un enlace a una actividad y se desea conectar una nueva acción al pin de entrada, a continuación crear una **copia de referencia** de dicha actividad arrastrando desde el toolbox **Services** un nuevo bloque, y entonces enlazar el flujo de datos al nuevo elemento.

Como ya se mencionó anteriormente solo se puede crear una conexión de enlace al mismo pin de entrada, sin embargo se puede usar una actividad **Merge** o **Join** para enrutar los mensajes desde múltiples actividades al mismo pin de entrada de un bloque.

Tanto **Merge** como **Join** proporcionan una función similar para consolidar o conectar secuencias separadas o flujo de datos, pero presentan algunas diferencias destacables. La actividad **Join**, no puede ser usada para conectar dos o más conexiones de notificación. Esto también ocurre incluso cuando las conexiones podrían no ser provenientes directamente desde una salida de evento o notificación, pero donde estas residen en el flujo de datos que se está intentando conectar. En tal caso se usa una actividad **Merge** en su lugar. Del mismo modo hay que tener en cuenta que esta actividad reenvía los mensajes tan pronto como son recibidos, a lo contrario del elemento **Join** que únicamente reenvía los mensajes cuando estos son recibidos en todas sus entradas.

Se puede crear múltiples enlaces de conexión desde el pin de salida de un bloque. Sin embargo si una actividad ofrece más de una opción en su pin de resultado o notificación, solo se puede conectar uno de estos a la vez.

5. CREAR Y EDITAR ACTIVIDADES

VPL permite al programador crear sus propias actividades personalizadas, que están compuestas por un flujo de datos de otras actividades. Esto permite volver a utilizar secuencias comunes de información dentro del programa.

Para poder comenzar, primero hay que arrastrar un icono **Activity** desde el cuadro de herramientas **Basic Activities** y abrirlo haciendo doble click sobre el bloque. Esto abre una nueva ventana dentro del área de trabajo de VPL, ver Figura 20.

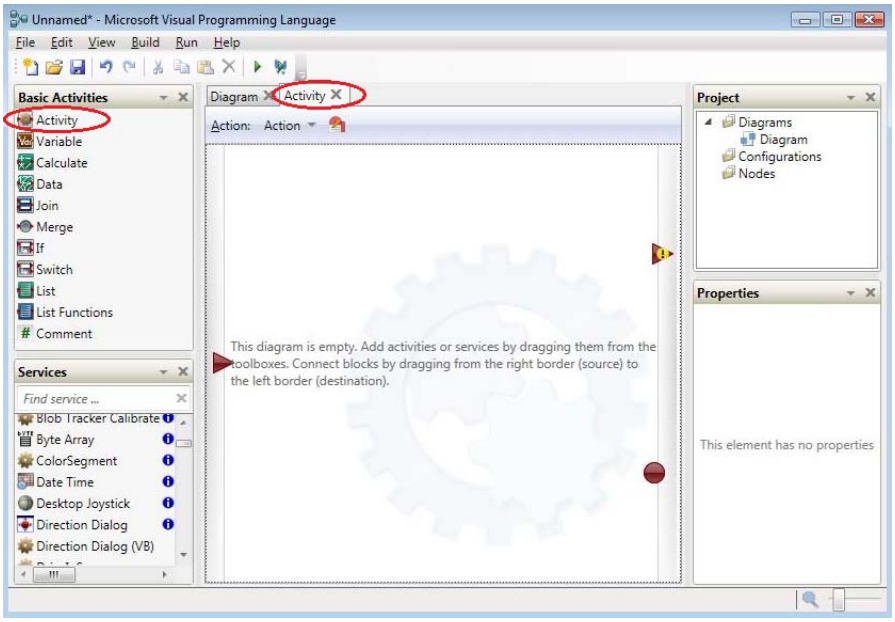


Figura 20. Ventana Activity dentro de VPL

Al igual que con las actividades pre-construidas, la actividad personaliza puede soportar una o más acciones de entrada (controladores). VPL crea automáticamente una platilla **Start** and **Action** por defecto, para la actividad personalizada.

La platilla **Start** permite definir un flujo de datos que se inicializan cuando la actividad personalizada arranca, es decir cuando el proyecto se ejecuta, sin tener en cuenta las conexiones de entrada de la actividad. **Start** puede realizar tareas tales como la inicialización de variables. La plantilla **Action** posibilita definir un flujo de datos interno que permite realizar conexiones tanto de entrada como de salida.

Si la plantilla **Start** o **Action** no están actualmente visibles cuando se genera una actividad, utilizar la opción desplegable justo por debajo de la pestaña con el nombre de la actividad, ver Figura 21.

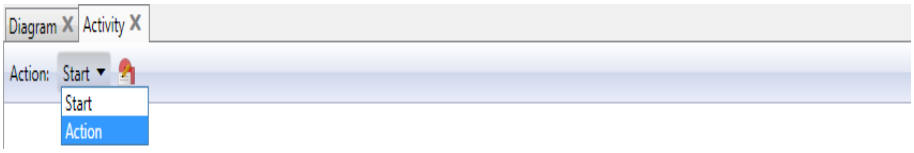


Figura 21. Selección de la Plantilla Start o Action en una actividad personalizada

A continuación se pueden insertar otras actividades desde el cuadro de herramientas **Basic Activity** o **Services** y conectar estos bloques de igual manera como se lo hace dentro del diagrama principal de VPL. Para permitir que la actividad personalizada sea conectada a otra rutina, se debe enlazar el flujo de datos interno a los pines ubicados a sus extremos, en la Figura. 22 se puede observar los pines de conexión internas de las cuales dispone una actividad personalizada.

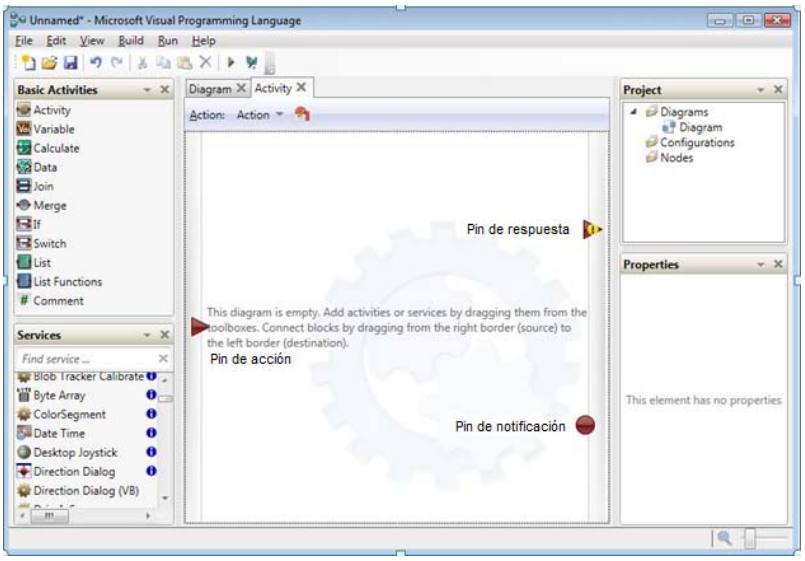


Figura 22. Pines de conexión internos-actividad personalizada

Si se desea que la actividad soporte un enlace de entrada, se arrastrara una conexión desde el pin de acción localizado sobre el borde izquierdo de la actividad personalizada hacia la entrada del bloque correspondiente al flujo de datos interno que se ha definido.

Por lo general una plantilla **Action** también proporciona una salida. Para habilitar el intercambio de mensajes entre el flujo de datos interno de la actividad personalizada y el flujo de datos del diagrama principal o cualquier otro, arrastrar una conexión desde el pin de salida correspondiente del flujo de datos interno, al pin de conexión localizado en el borde derecho de la actividad personalizada.

Si la salida es el resultado de una entrada, se usara el pin de conexión **Respuesta**. Si la actividad proporciona notificaciones, arrastrar una conexión al pin redondo **Notificación**.

Se puede usar el cuadro de dialogo **Actions and Notification** para definir el nombre y el tipo de dato para la entrada externa hacia la actividad y los pines de conexión de salida.

Para una acción se puede agregar un valor de entrada o de salida haciendo click en la etiqueta **Action** en el cuadro de diálogo.

Y al hacer click en el botón **Add** se agregaran los valores de entrada y salida. Dentro de esta pestaña un puede configurar el nombre, descripción y el tipo de dato para la plantilla **Action.**

Se puede añadir o eliminar varios valores de esta forma. De manera similar se puede agregar, editar o eliminar los valores de notificaciones haciendo click sobre la pestaña **Notifications.**

Cuando se ha terminado de editar las conexiones de entrada y salida hacer click en OK.

El diálogo **Actions and Notificactions** también permite crear pines de conexión de notificación adicionales. Para añadir una notificación, hacer click en

la pestaña **Notifications** en el diálogo, entonces hacer click en el **Add** Button y proporcionar un nombre y un tipo de dato. Después, hacer click en ok se puede arrastrar una conexión desde una salida de la actividad personalizada al pin de notificación que se acaba de crear.

También se puede utilizar el diálogo **Actions and Notifications** para crear controladores de acción adicionales para l actividad personalizada. Para añadir otro controlador de acción, hacer click en la pestaña **Actions** en el cuadro de diálogo y hacer click en **Add** Button (se necesita añadir los valores de entrada y salida para la nueva acción).

Para pasar a la acción recien creada, seleccionar esta desde la lista desplegable "Action" en la parte superior de la página "Activity".

6. USANDO SERVICIOS

Los servicios son programas compilados por separados, usualmente escrito en C#, que pueden ser usados de manera similar a las actividades dentro de un diagrama de VPL. Se pueden añadir servicios al diagrama desde la caja de herramientas Services.

Cuando se arrastra un servicio desde el toolbox **Services** hacia el diagrama la primera vez el servicio es configurado de forma automática para ser creado al ejecutar el diagrama. Puede que se tenga que configurar el servicio para que funcione correctamente.

Observar el siguiente bloque de actividad para un servicio que implementa el servicio de **Timer**. Ver Figura 23

Figura 23. Servicio Timer

- **OPERATIONS**

El pin de entrada (también llamado de acción o petición) sobre el lado izquierdo del bloque (flecha apuntando al bloque) se puede utilizar para enviar las solicitudes al servicio Timer. Tales como GetCurrentTime, SetTimer and Wait. Este tipo de mensajes son también a menudo referidos como operaciones en terminología DSS.

En el protocolo DSS hay varias operaciones estándar que los servicios pueden optar por aplicar.

SOLICITUD	DESCRIPCIÓN
DROP	Cierra el servicio. Este no es visible a través de la interfaz con VPL
GET	Devuelve una copia del actual estado interno del servicio. A menudo esta es la información que tú quieres obtener desde un servicio.
QUERY	Devuelve solo una parte del estado interno del servicio. Generalmente este tipo de operaciones tiene nombres que son específicos para el servicio.
REPLACE	Reemplaza todo el estado del servicio con el valor especificado.
UPDATE	Similar a la solicitud reemplace, pero únicamente reemplaza una parte del estado de servicio. Las operaciones de actualizaciones usualmente tienen nombres que son específicos para cada servicio.

- **RESPONSES**

En respuesta una solicitud, el servicio Timer emite un mensaje sobre el pin Result en el lado derecho del bloque. Para **GetCurrentTime**, la respuesta es la fecha y hora actual cuando se recibió la solicitud. Para **SetTimer** y **Wait** la respuesta es únicamente un mensaje **Success**. Sin embargo hay una diferencia. Una solicitud **Wait** no recibe una respuesta hasta que el intervalo de tiempo especificado haya transcurrido. Haciendo que el flujo de datos pare en este punto.

La solicitud **SetTimer** es un poco más compleja. Además de enviar de vuelta una respuesta inmediatamente, decidiendo que la solicitud fue aceptada, el servicio **Timer** envía una notificación un tiempo después.

- **NOTIFICACIONES**

El pin redondo ubicado en el lado derecho del bloque es conocido como pin de notificación. Las notificaciones ocurren cuando algún evento se produce, lo que hace que el servicio actualice su estado interno.

Los tipos de mensaje para las notificaciones suelen ser los mismos como para las operaciones del tipo de actualización.

Para el servicio Timer, las notificaciones son enviadas cuando un temporizador se dispara, debido a una anterior solicitud de **SetTimer**. Hay que temar muy en cuenta que en este caso dos mensajes son recibidos desde el servicio Timer: Una respuesta inmediata **Success** cuando se realiza la solicitud y una notificación **TimerComplete** algún tiempo después.

En VPL cuando se enganchas un bloque **activity** hasta el pin de notificación de un servicio, este recibe todas las notificaciones desde el servicio durante todo el

tiempo en el que el programa se está ejecutando. Esto se reconoce como la suscripción al servicio, si el servicio tiene varios tipos diferentes de notificaciones, se puede agregar más conexiones con tan solo arrastrar una nueva conexión desde el pin de notificación. Ver figura 24

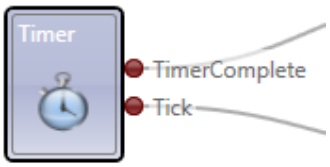


Figura 24. Pin de notificación servicio Timer

Observar que cuando se conecta un servicio o actividad al pin de notificación de otro servicio, el pin de entrada desparece. Por lo tanto hay que añadir otro bloque de servicio al diagrama si también se desea enviar solicitudes.

Si se arrastra un servicio a un diagrama donde ya se ha agregado previamente, un cuadro de diálogo aparecerá preguntando si desea crear una nueva actividad o usar el ya existente. Ver Figura 25

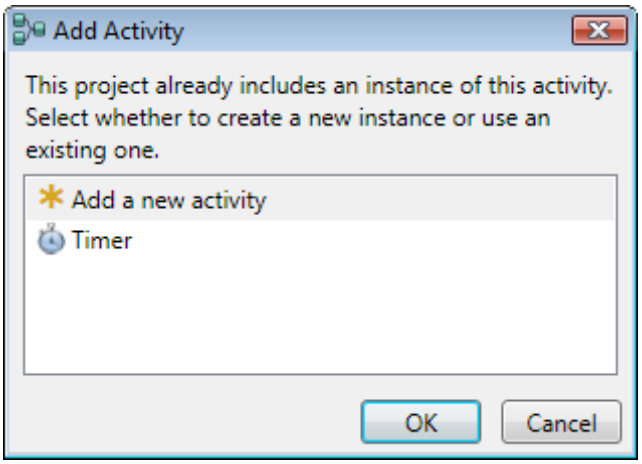


Figura 25. Creacción Nuevo servicio Timer

En VPL se puede usar servicios existentes una y otra vez con tan solo crear nuevos bloques de actividad para ellos. Esto es útil si se desea utilizar un servicio desde varios lugares en el diagrama porque hace que el diseño sea más limpio y más fácil de leer. Hay que tener cuidado de no crear nuevas actividades si solamente requiere de una, de lo contrario VPL iniciará varias instancias y esto se pondrá confuso. Es fácil observar si se ha creado varias instancias porque los nombres de las actividades deberían ser únicos.

- **FAULTS**

Si el diagram de datos dentro del programa está funcionando como se espera, entonces siempre se obtendrán las respuestas que se busca de un servicio.

Si un servicio no puede completar la solicitud o la rechaza como inaceptables, entonces devolverá un error **(Fault).** Los **Faults** son algunas veces también referidos como **Exceptions**. Técnicamente son diferentes pero ambos indicant que se ha producido un error inesperado.

Hay diferentes tipos de **Faults** y estos son específicos para determinados servicios por lo que no es posible proporcionar una lista de todos los posibles **Faults**.

Un mensaje **Fault** contiene la siguiente información: Código, Razón, Nodo, Rol y Detalle.

Todos o algunos de los campos antes mencionados pueden estar vacíos. En general, los tipos de **Faults** que pueden ocurrir son: OperationFailed, InsufficientResources, ActionNotSupported, MessageNotSupported, UnknownContract, UnknownEntry y DuplicateEntry.

Es una buena práctica siempre comprobar un **Faults** al enviar una solicitud a un servicio. VPL permite conectar el pin de salida de una servicio varias veces. La respuesta de espera se puede remitir a diversas actividades. Del mismo modo la respuesta **Fault** puede ser enviada a otros lugares para ser evaluado. El siguiente digrama muestra como establecer un tiempi de retardo de un segundo (1000 ms). Si la inicialización del Timer es satisfactoriaentonces otras dos actividades serán ejecutadas. Sin embargo, si existe un error entonces un bloque simple **Dialog** es mostrado sobre la lapantalla con un mensaje de error. Ver figura 26.

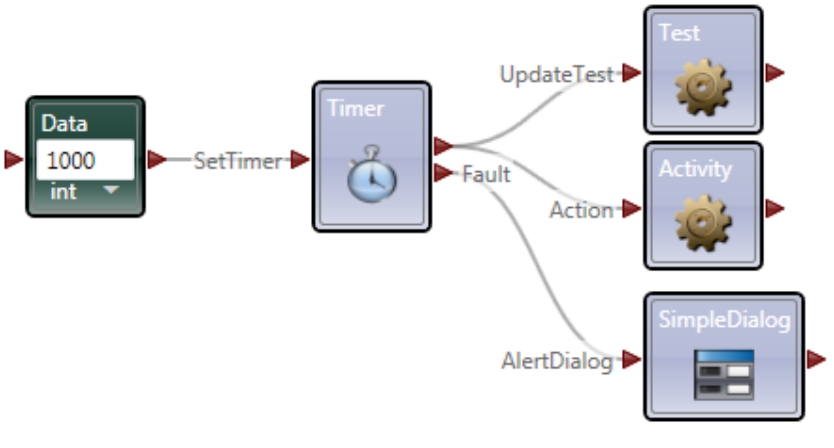


Figura 26. Evaluación condición Fault

7. SERVICIOS DE INICIO O ARRANQUE

VPL viene con un conjunto de servicios que pueden ayudar a empezar a escribir aplicaciones.

No todos los servicios de inicio están presentes en todas las ediciones. Algunos servicios no están destinados a ser utilizados en los diagramas de VPL, ya que son aplicaciones que requieren de un entorno de desarrollo C#.

En particular cualquier servicio que tiene un nombre que contiene, **Tutorial** es más probable que pretenda ser una aplicación independiente que no pueda ser usada desde VPL.

Del mismo modo si un servicio tiene un nombre que incluye **Simulation** no son adecuados para usarse en VPL a menos que explícitamente se mencione en su documentación.

A continuación se detallan algunos de los servicios presentes dentro del Visual Programming Language:

Utilities: Estos servicios proveen funcionalidades básicas que incluyen servicios como: data-time, texto, registro y funciones matemáticas. Las Utilities a menudo pueden simplificar los diagramas VPL, y se puede observar que estos son usados frecuentemente a través de ejemplos y tutoriales.

Servicios de interfaz de usuario: estos servicios proveen la capacidad para interactuar con el usuario en una variedad de formas tales como servicios de reconocimieto de voz, control del Kinect y conttol de juego Xbox,

Servicios robóticos: estos servicios proporciona abstracciones para una variedad de hardware orientado a la robótica, estos servicios incluyen: sensores de contacto, controladores diferenciales, motores, cámaras web, brazos articulados y más.

Simulation services: estos servicios pueden ser usados con entidades en el Visual Simulation Enviroment, tales como robots y sensores.

8. CONFIGURACIÓN DE SERVICIOS

Las actividades que representan los servicios DSS pueden requerir información acerca de cómo estos deberán arrancar. Estas configuraciones se conocen como estado inicial. Además, los servicios socios necesariamente necesitan ser arrancados o instanciados. Los detalles del servicio se muestran en la **ventana propiedades,** cuando el bloque ha sido seleccionado.

Para establecer la configuración inicial se la puede realizar a través de la ventana **Set Configuration,** haciendo clic sobre esta opción desde el menú **Edit**.

Dependiendo del servicio se puede tener una de las cuatro opciones para definir la configuración inicial.

- **Establecer la configuración inicial**

Un servicio puede tener un estado inicial que incluye su configuración. Esta puede incluir propiedades del estado como su puerto de comunicación serie y la velocidad o tasa de transmisión, también puede incluir propiedades físicas. Estas propiedades de configuración inicial pueden ser establecidas a través de la ventana **Properties** o sobre la etiqueta **Initial State** de la ventana **Set Configuration.**

Algunos servicios pueden definir socios, usados para su funcionamiento. Los socios son otros servicios que proporcionan soporte para un servicio. Para los socios se puede definir el **servicio específicos** que el socio usara así como las propiedades de configuración para este.

Se puede definir que servicio usar para los socios, arrastrándolos desde la caja de herramientas de servicios hacia sus entradas sobre la pestaña **Partners** de la

ventana **Set Configuration**. Se puede entonces establecer la configuración para los socios.

Cuando se establece la configuración inicial de un servicio VPL agrega un archivo de este al proyecto. Si un servicio es borrado o eliminado y se ha establecido su configuración inicial, VPL mostraría una advertencia pidiendo que se confirme la eliminación ya que esto también eliminara el archivo de configuración inicial.

Este archivo también se puede eliminar mediante su selección ubicada en la ventana **Project** opción **Configurations.**

Si se elimina el archivo de configuración, se pueden restaurar los parámetros de configuración por defecto haciendo clic en **Create Initial State** que aparece en la ventana propiedades o en la página de configuración para el servicio. También se puede importar un archivo de configuración ya existente.

- **USANDO OTRO SERVICIO**

Esta opción permite usar la información de configuración de otro servicio para inicializarlo. Se puede seleccionar un servicio específico a ser usado, únicamente arrastrando un servicio a la página de configuración. O se puede seleccionar la opción **Use Existing Service** o **Create a New One** la cual solicitara al DSS runtime encontrar un servicio de arranque compatible o intentar arrancar un servicio adecuado y usar esa configuración. Esto permite utilizar servicios genéricos en el diagrama y simplemente cambiar el servicio actual que se utiliza en el **runtim**e. Sin cambiar su diagrama.

- **USANDO UN MANIFEST**

Esta opción permite seleccionar un archivo manifest existente para iniciar el servicio. Un **manifest** es un archivo especial que describe un conjunto de servicio a ser instanciados y sus configuraciones. Usar el comando **Import Manifest** para mostrar una lista de archivos existentes y seleccionar un servicio que sea inicializado por este. También se puede optar por dejar que el DSS runtime encuentre o cree un servicio adecuado.

- **NONE**

Esta opción indica que se desea inicializar el servicio sin ninguna configuración específica. Esta es la opción predeterminada para la mayoría de los servicios simples. Un buen ejemplo incluyen los servicios: **MathFunctions**, **Simple Dialog** o **Text to Speech.**

9. EJECUTAR Y DEPURAR

Para ejecutar un proyecto de VPL abierto se debe hacer clic en **Start** desde el menú **Run.** Esto mostrará el cuadro de dialogo de ejecución y arrancará el DSS runtime y ejecutará el proyecto así como cualquier otro servicio que se haya definido como socio.

El cuadro de diálogo Run aparece con un hipervínculo a una página web de depuración, una lista que muestra la secuencia de mensajes, que proporciona información sobre el estado de la puesta en marcha del proyecto. Un menú filtro en la parte superior de la lista permite definir que mensajes pueden ser mostrados. Tener en cuenta que hay un pequeño signo más (+) a la izquierda de cada mensajes. Al hacer clic sobre este se expande el mensaje. Ver Figura 27.

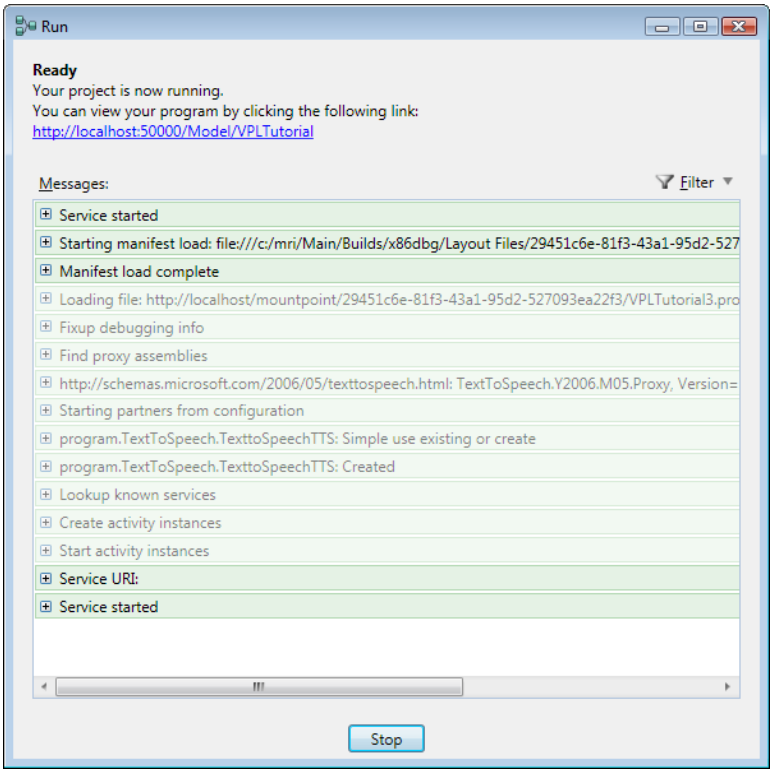


Figura 27. Ventana Run de Ejecución de Depuración

Los errores que ocurren durante la ejecución del programa deben provocar mensajes que tienen que ser mostrados en el cuadro de diálogo **Run.** Para poder observar dichos mensajes es posible que haya que desplazarse por la lista de mensajes. En este caso el menú **Filter** es de gran utilidad porque se puede configurar para que únicamente muestre mensajes de error.

Para detener el programa hay que hacer clic en el botón **Stop** del cuadro de diálogo o en la pestaña (x) de dicha ventana.

- **DEPURACIÓN DE LA APLICACIÓN VPL**

VPL incluye varias características para ayudar a depurar el proyecto. El primero es un conjunto de pequeños iconos que pueden aparecer al diseñar el flujo

de datos. Estos aparecen cuando se tienen valores perdidos o incompatibles para las conexiones u otras configuraciones.

Algunos iconos de depuración son solo de asesoramiento mientras que otros identifican los errores en el tiempo de diseño que puede ocasionar que el programa no se ejecute correctamente.

Cuando se ejecuta el proyecto, existen otras opciones de depuración, mediante el uso de la vista de depuración (Debug View). Una manera de acceder al **Debug View** es haciendo clic en el hipervínculo mostrado en el cuadro de dialogo **Run**. Al hacer clic en este enlace un navegador web se pone en marcha y carga la página web de la vista de depuración que muestra información sobre la ejecución del proyecto. Sin embargo debido a que el proyecto puede ejecutarse antes de que la página **Debug View** aparesca, está esta información puede tener un valor limitado. Una mejor alternativa es usar el comando **Debug Start** desde el menú **Run**. Este comando también muestra la página **Debug View** pero detiene su ejecución en el primer bloque de actividad del diagrama, proporcionando botones que permitan el paso a través del programa. Ver Figura 28.

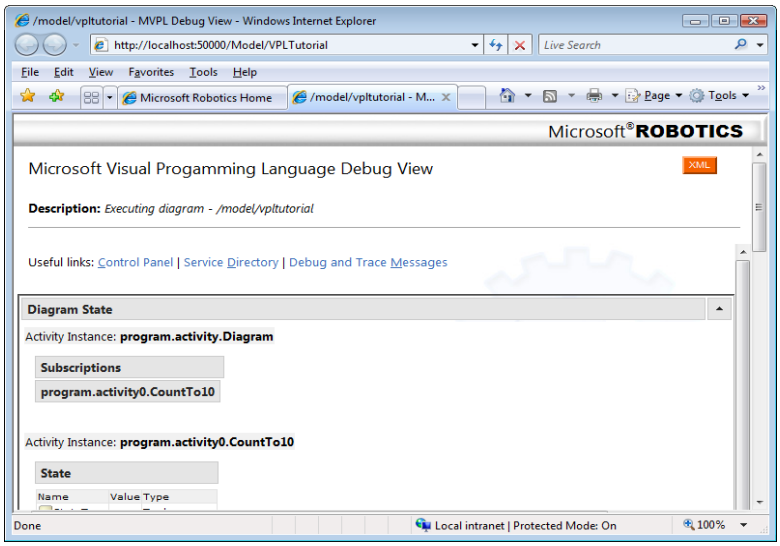


Figura 28. Diagrama de estado

La página **Debug View** muestra las siguientes secciones las cuales proporcionan información acerca del estado actual del diagrama. Cada sección se puede ocultar o volver a mostrar utilizando los botones en el extremo derecho del encabezado de la sección. Si la página no muestra ninguna información, intentar utilizar los comandos de **Refresh** del navegador para recargar la página.

- **ENLACES UTILES**

En la parte superior de la página hay enlaces a otras páginas que proporcionan acceso a la salida del sistema de servicio DSS, a continuación se detallamuna lista de ellas:

Control panel: muestra la salida del servicio DSS Control Panel que permite iniciar y detener manualmente los servicios DSS y también puede ser usado para inspeccionar los contratos de los servicios disponibles.

Service Directory: muestra la salida del servicio **Service Directory** el cual mantiene un directorio de ejecución de servicios DSS.

Depuración y seguimiento de los mensajes: muestra los mensajes de depuración y registro de la **Console Output Services** que son generados por un nodo DSS en ejecución.

- **DIAGRAMA DE ESTADO**

La sección de la página de diagrama de estado muestra la información sobre el flujo de datos incluyendo los valores de estado actual y las actividades personalizadas.

Normalmente esto será una colección de variables que se han definido, sus valores actuales y sus tipos. También muestra el diagrama de las suscripciones, aquellos servicios que proporcionan conexiones de notificación en el diagrama o en la actividad personalizada

Es posible hacer clic en el hipervínculo para observar una vista del estado actual del servicio.

- **ACTIVIDAD ACTUAL "Current activity"**

Esta sección muestra una vista del diagrama y tres botones **Run**, **Run Slowly** y **Stop to the Next Activity** (que alterna con **Pause**). Estos botones se pueden utilizar para controlar la ejecución del programa. Ver Figura 29.

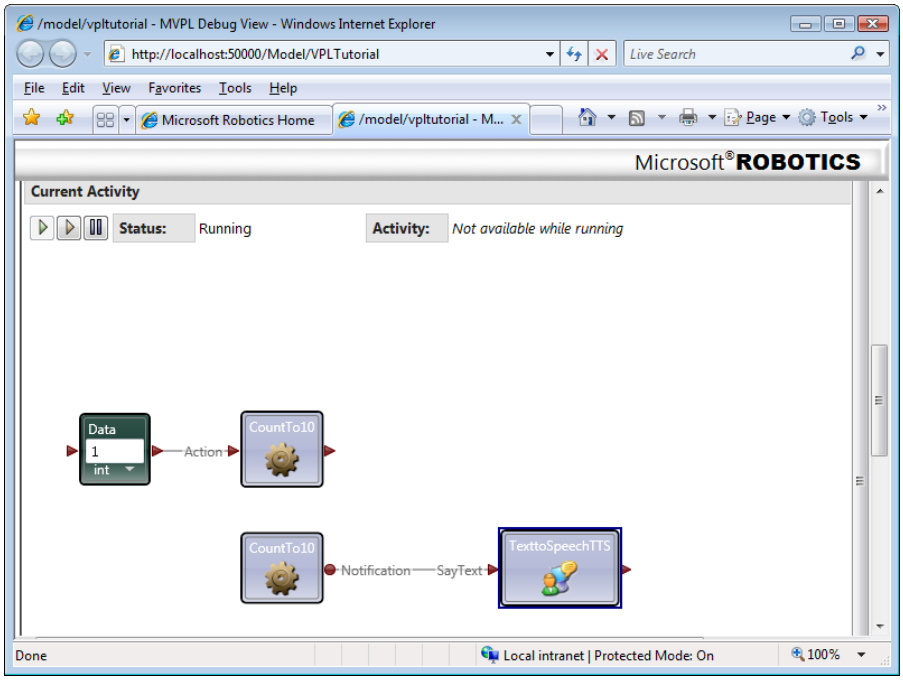


Figura 29. Ventana de Actividad Actual

Al presionar el botón **Stop to the Next Activity** se mueve la ejecución al siguiente bloque y se resalta. En cada paso el valor actual y el tipo de mensaje que se ha enviado entre las actividades se muestra en la parte inferior de la sección **Current Activity**. Hay que tener en cuenta que cada paso no necesariamente es secuencial, ya que puede tener partes de su flujo de datos que se ejecutan en paralelo.

Cuando un programa está ejecutándose, el bloque de actividad actual esta resaltado. Normalmente el programa se ejecutara muy rápidamente sin poder observar el bloque resaltado que está en operación en ese momento ya que el flujo de datos va de un bloque a otro, es por ello que hay un botón llamado **Run Slowly.**

- **PUNTO DE INTERRUPCIÓN**

Esta sección muestra todos los Breakpoints que se establece (pueden ser creados en la siguiente sección). Un Breakpoints es una manera de pausar la ejecución de una actividad específica, permitiendo la vista del estado de la actividad en este punto. Una vez que se establece un Breakpoints se puede hacer uso de los botones **Run** o **Run Slowly** en la sección **Current Activity** para avanzar a este **Breakpoint** (si es el caso del primer Breakpoints o si se tiene un conjunto de varios Breakpoint)

Cada Breakpoint contiene un botón de borrado **Clear** que permite removerlo cuando ya no se lo necesita. También dispone de un botón de deshabilitado **Disable** que permite ignorarlo temporalmente mientras se está ejecutando. El botón **Disable** al ser presionado cambia a un botón **Enable** lo que permitirá restaurar el **Breakpoint**. El uso de estos botones vuelve a mostrar la página **Debug View** automáticamente. Si la página no se vuelve a mostrar correctamente, intentar actualizar el navegador.

- **ACTIVIDADES PENDIENTES**

Esta sección muestra la siguiente actividad que se ejecutará mientras dura la depuración (puede haber más de una actividad pendiente si hay flujos de datos en paralelo). Esto también muestra información de estado sobre las actividades e incluye un SetBP (Set Break Point) que permite pausar la ejecución en esa actividad.

Si se utiliza el botón SetBP para establecer un punto de interrupción, la página **Debug View** se actualizara automáticamente para mostrar el nuevo punto en la sección **Breakpoints.**

- **CONFIGURACIÓN DEL PUERTO**

Dado que el proyecto en si es un servicio DSS, cuando VPL corre el proyecto este pone en marcha el DSS runtime y carga el servicio. Para hacer esto un puerto HTTP y TCP son necesarios. VPL automáticamente establece los valores por defecto para el proyecto. Sin embargo si está ejecutando varias instancias del VPL al mismo tiempo u otras aplicaciones que podrían estar utilizando el mismo Puerto de comunicación, se puedes cambiar la configuración del puerto haciendo clic en el comando **Portssettings** sobre el menu **Run** y colocar nuevos valores en el diálogo de resultado.

Se puede también usar la configuración del puerto para accesar a la información de depuración de la aplicación actual usando un navegador Web y configurando sus direcciones a la ubicación del **Port** de la PC; por ejemplo usando la configuración del puerto por defecto, la URL http://localhost:5000/console/output, que mostrará una página Web que contiene el Cosole Output Service y enlaces a la información relacionada. Ver figura 30.

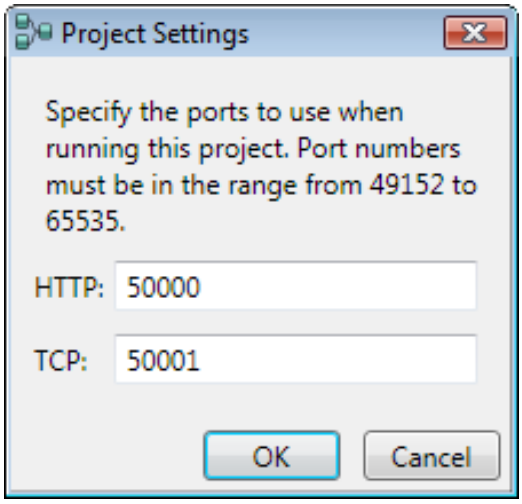


Figura 30. Configuración puerto de comunicación VPL

ACTA DE ENTREGA

El proyecto fue entregado al Departamento de Eléctrica y Electrónica y reposa en la Universidad de las Fuerzas Armadas – ESPE, desde:

Sangolquí, 9 DE ABRIL de 2015

ELABORADO POR:

MANUEL MAURICIO SUNTAXI LLUMIQUINGA

171885964-6

AUTORIDAD:

Ing. Luis Orozco MSc.

DIRECTOR DE LA CARRERA DE INGENIERÍA EN ELECTRÓNICA, AUTOMATIZACIÓN Y CONTROL