UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE QUITO

CARRERA: INGENIERÍA DE SISTEMAS

Trabajo de titulación previo a la obtención del título de: INGENIERO DE SISTEMAS

TEMA:
DISEÑO, DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA QUE PERMITA CONTROLAR UN BRAZO ROBÓTICO MITSUBISHI RV-2AJ A TRAVÉS DEL PERIFÉRICO KINECT DE MICROSOFT COMO INTERFAZ DE USUARIO.

AUTORES:
CASTRO MANTILLA SLIN GERMÁN
JIMÉNEZ ALMAGRO FERNANDO ESTEBAN

DIRECTOR:
RODRIGO EFRAÍN TUFIÑO CÁRDENAS

Quito, octubre de 2014

DECLARATORIA DE RESPONSABILIDAD Y AUTORIZACIÓN DE USO DELTRABAJO DE TITULACIÓN

Nosotros, autorizamos a la Universidad Politécnica Salesiana la publicación total o parcial de este trabajo de titulación y su reproducción sin fines de lucro.

Además, declaramos que los conceptos y análisis desarrollados y las conclusiones del presente trabajo son de exclusiva responsabilidad de los autores.

Quito, octubre de 2014.

_____

Jiménez Almagro Fernando Esteban

CC: 0502653066

_____

Castro Mantilla Slin Germán

CC: 1719831321

DEDICATORIA

"Cuando fui a la escuela, me preguntaron que quería ser de mayor. Yo respondí: feliz. Me dijeron que yo no entendía la pregunta, y yo les respondí que ellos no entendían la vida".

John Lennon

Dedico este trabajo a mi madre Carmen Amelia Almagro Ruiz, que ha sido mi inspiración y fuerza para poner metas tan altas, a mi padre Gustavo Humberto Jiménez Santacruz por ser el pilar en cada paso que he dado y apoyarme incondicionalmente en cada etapa de mi vida, a mi brow Pablo Andrés Jiménez Almagro por su entrega y compañía.

Y de manera especial a Mercedes Almagro, por enseñarme que siempre vale la pena seguir luchando por nuestros sueños.

Fernando Jiménez Almagro

Gracias a los que me dieron la vida por enseñarme a seguir un camino que nunca estuvo marcado, a cambiar prejuicios y estereotipos por ganas de seguir adelante y que si hay que morir hay que morir de pie.

Slin Castro Mantilla

AGRADECIMIENTO

La realización de este trabajo de titulación ha sido posible gracias a la colaboración de las siguientes personas:

A nuestro director de trabajo de titulación, Ing. Rodrigo Efrain Tufiño Cárdenas, por confiar en nuestro proyecto y brindarnos a más de sus enseñanzas, su amistad.

A nuestra amiga Ing. Jhoana Celi, por su apoyo en el uso de los laboratorios.

A nuestro gran maestro y amigo Ing. Carlos Pillajo, por su conocimiento y apoyo en la implementación de nuestro proyecto.

Fernando Jiménez Almagro

&

Slin Castro Mantilla

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El avance vertiginoso de la tecnología de consumo masivo al igual que la evolución del Internet ha ido planteando el desarrollo de dispositivos cada vez más impresionantes, útiles y que pueden cambiar la manera de controlar y manipular los objetos cotidianos con los que el ser humano interactúa diariamente tal como otros más desarrollados y utilizados en un ámbito industrial.

La problemática del uso de los manipuladores robóticos de los que dispone la Universidad Politécnica Salesiana está en que solo se los enfoca al control de estos de una manera mecánica y didáctica mas no en el desarrollo de nuevas tecnologías siguiendo tendencias de avance tecnológico mundial.

Tras lo expuesto el proyecto será enfocado en unir tres tecnologías usadas para campos bastante variados y que poco tienen que ver entre sí, para obtener como resultado un controlador remoto de un manipulador específicamente el robot Mitsubishi RV-2AJ, que mediante un canal que hace uso de Internet será controlado por el periférico Kinect.

Haciendo uso de conceptos aprendidos en el área de robótica se aborda el estudio de la misma, con mayor énfasis en manipuladores industriales abarcando nociones básicas y técnicas del funcionamiento, también se verá desde un punto de vista profundo el funcionamiento del periférico Kinect para terminar con el análisis de la utilización y desarrollo de servicios web para el desarrollo del controlador remoto.

ABSTRACT

The rapid advance of technology consumer as the evolution of the Internet has been proposing the development of increasingly impressive, useful devices that can change the way to control and manipulate everyday objects that people interact daily as more developed and used in an industrial field.

The issue of the use of robotic available to the Universidad Politécnica Salesiana manipulators is that only focuses on the control of these in a mechanical way and didactic but not in developing new technology trends worldwide following technological advancement.

After what has already exposed the project will be focused on joining three technologies used to quite varied fields and that have little to do with each other in order to obtain results in a remote controller for a manipulator specifically the robot Mitsubishi RV-2AJ, which through a channel makes use of Internet will be controlled by the Kinect peripheral.

Using concepts learned in the area of robotics study addresses the same, with greater emphasis on industrial manipulators covering basic concepts and techniques of operation, will also be from a deeper point of view the operation of the peripheral Kinect to end analysis of the use and development of web Services for the development of remote controller.

INTRODUCCIÓN

En el documento se abordará contenidos que estarán distribuidos en las ramas de robótica e ingeniería de software, además de breves explicaciones sobre el origen de estos, en la primera parte se abordará contenidos relacionados a la al hardware y robótica específicamente a manipuladores, la segunda parte contiene detalles técnicos e históricos sobre la evolución de los servicios web, el transporte de mensajes y los estándares SOAP.

Existen procesos industriales, donde se requiere utilizar brazos robóticos que sean controlados por el usuario por interfaces no amigables o a su vez por sistemas de complejidad avanzada, lo que no permite que sean utilizados por cualquier persona sin conocimientos específicos, el presente análisis se basa en la necesidad de crear una interfaz que permita la interacción con el brazo robótico Mitsubishi RV-2AJ, con un tiempo de respuesta favorable y una buena precisión que permita al usuario final manipularlo sin la necesidad de extender la complejidad del mismo.

Como una manera de innovación a las interfaces de usuario que comúnmente se utilizan en este tipo de manipuladores se propone la creación de un sistema interoperable que permita el control del brazo robótico mediante el movimiento del brazo del usuario basado en el dispositivo Microsoft Kinect, el movimiento se realiza en tiempo real, adicionalmente se permitirá que la conectividad entre el cliente y el servidor final sea realizada desde cualquier lugar mediante la conectividad a Internet. Para lograr este objetivo el proceso inicia con la inicialización del sensor, luego se procede a adquirir los datos de profundidad, para luego empezar con el proceso de calibración del usuario, una vez finalizado este proceso se puede empezar a trabajar con los datos del usuario. El siguiente paso es el cálculo del movimiento del brazo del usuario, aquí se debe tomar los valores necesarios para poder aplicar el algoritmo de seguimiento del brazo que controla el robot. La condición fundamental que existe dentro del sistema es la detección del gesto para finalizar la conexión y el envío de datos, la misma que es considerada antes de proceder a la conexión con el servicio que permite el manejo en tiempo real del brazo.

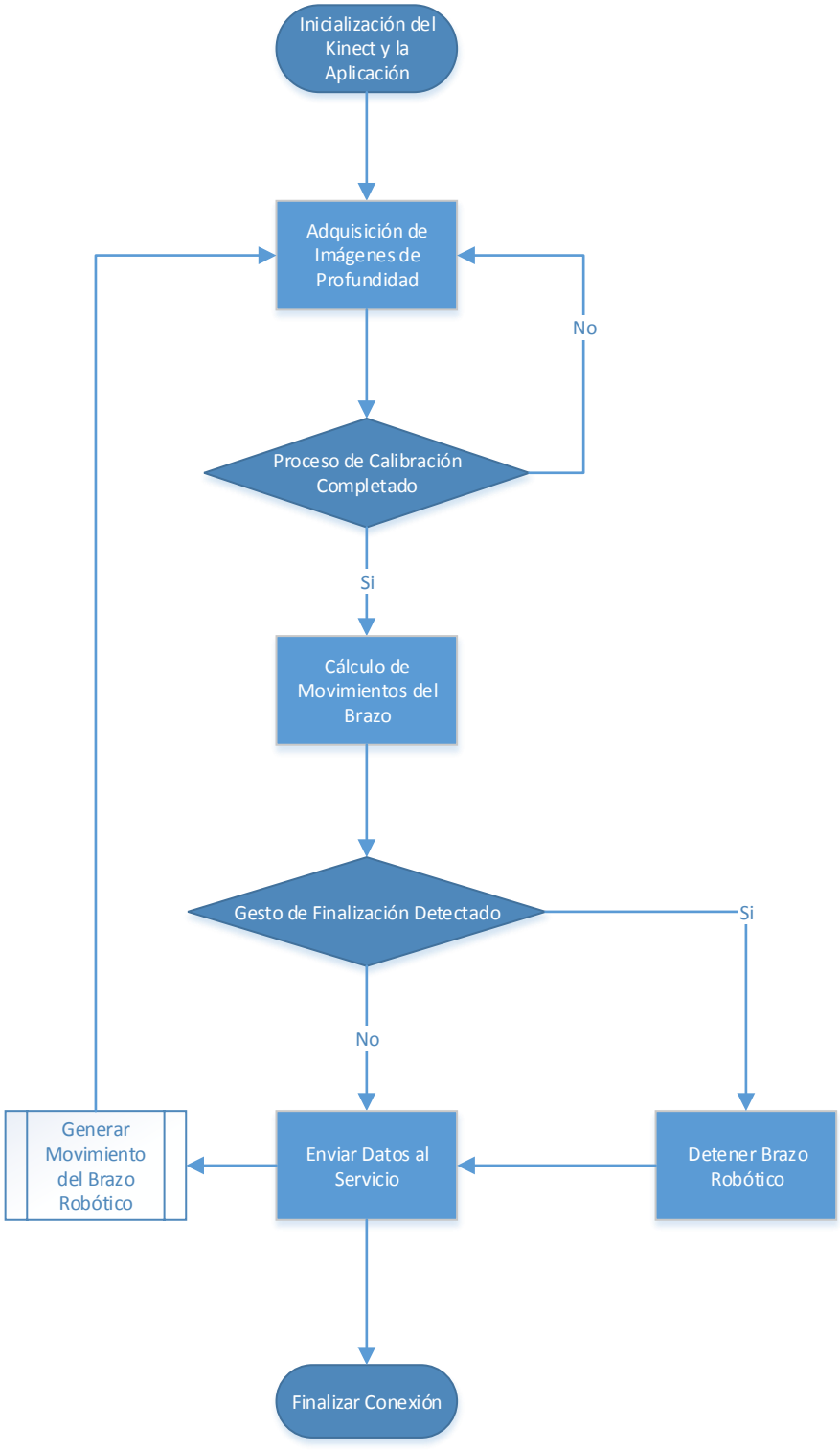


Figura 1. Diagrama de flujo software del sistema

Elaborado por: (Castro, Slin & Jiménez, Fernando)

El sistema en tiempo real implementado, omite algunas etapas propias de un proceso dentro de un sistema de tiempo real, debido a que parte de los datos son adquiridos y procesados del ambiente a través del sensor Kinect y enviados al computador como datos digitales.

El siguiente diagrama presenta las etapas que componen el sistema en tiempo real para el movimiento del brazo robótico Mitsubishi Rv-2AJ y los procesos ejecutados en cada etapa.

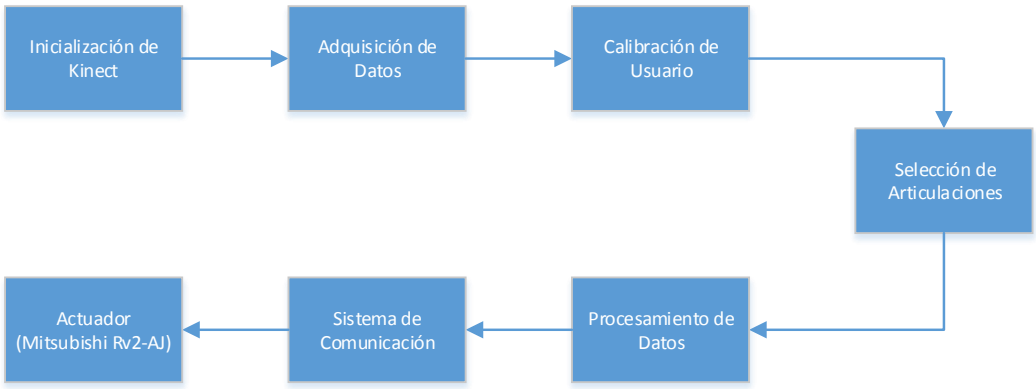


Figura 2. Diagrama de bloques etapas del sistema

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Una definición breve de las etapas del sistema se describe a continuación:

Inicialización del Kinect: El sensor es la variable de entrada que interactúa con el ambiente y entrega los datos pre procesados del mismo, adicionalmente los resultados dependen del tiempo, por este motivo esta es la primera etapa del sistema, tomando en cuenta que a pesar de que el sensor es único entrega un conjunto de variables indistintas debido a los diferentes datos que el Kinect es capaz de entregar al sistema.

Adquisición de imágenes y datos de profundidad: Se capturan todos los datos que envía el sensor hacia la computadora, permitiendo utilizarlos en la implementación del algoritmo de movimiento.

Proceso de calibración: No todos los usuarios presentan el mismo tamaño de extremidades del cuerpo, motivo por el cual se realiza el proceso de calibración que permite la correcta lectura de datos de usuario.

Selección de articulaciones a utilizar: Microsoft Kinect entrega datos referentes a las articulaciones del cuerpo del usuario, sin embargo no toda la información recibida es requerida en el sistema, dentro de esta etapa se adquieren únicamente las articulaciones que servirán para realizar el cálculo del movimiento del brazo, las demás son descartadas.

Cálculo de movimientos del brazo: Se implementan los algoritmos basados en cinemática directa que utilizan datos procesados por etapas anteriores y se calculan los datos finales que serán enviados a la siguiente etapa.

Movimiento del brazo robótico: Un sistema en tiempo real basado en la interconectividad mediante servicios utilizando como canal el Internet , aplica una acción en el ambiente después de procesar las variables de entrada, efectuando el movimiento final del brazo robótico Mitsubishi RV-2AJ a través del Api Serial.

Objetivos

Objetivo general:

Utilizar el periférico Kinect como un dispositivo de interfaz de usuario para la manipulación del brazo robótico Mitsubishi RV-2AJ utilizando la API de comunicación RS232 desarrollada en la UPS de forma remota mediante una Interfaz web.

Objetivos específicos:

- Determinar el entorno de trabajo apropiado para la interacción entre el brazo robótico Mitsubishi RV-2AJ y el periférico Kinect.
- Conocer las ventajas y desventajas de la utilización del periférico Kinect para la captura y procesamiento de imágenes en movimiento.
- Desarrollar el software de procesamiento de imágenes digitales y visión artificial bajo la plataforma .NET basada en el SDK proporcionado por Microsoft para el manejo de Kinect.
- Desarrollar el software de control para el brazo robótico Mitsubishi RV-2AJ bajo la plataforma.NET utilizando el Framework 4.

- Implementar la Interfaz de comunicación web entre el periférico Kinect y el brazo robótico Mitsubishi RV-2AJ mediante el uso de dos computadores ubicados geográficamente en lugares distintos.

- Comprobar la comunicación entre el sistema y el brazo robótico Mitsubishi RV-2AJ mediante el api de comunicación RS232.

- Utilizar WCF para la intercomunicación entre el brazo robótico Mitsubishi RV-2AJ y el periférico Kinect.

- Utilizar WPF como base para la construcción de la interfaz de usuario.

- Investigar el funcionamiento de la clase Streaming dentro del Framework 4 y su correcta convivencia dentro del entorno de trabajo.

- Diseñar la arquitectura para transmisión de datos de acorde a las tecnologías utilizadas.

Justificación

El desarrollo de este trabajo responde a los deseos de construir un sistema basado en conocimientos adquiridos a lo largo de la carrera, especialmente en las cátedras de visión artificial, programación y robótica que sirva como herramienta de interacción, manipulación y control del brazo robótico Mitsubishi RV-2AJ utilizando el periférico KINECT de Microsoft como interfaz de usuario y ayudar a la interacción entre una persona "natural" y el manipulador robótico.

El periférico Kinect es un dispositivo creado por Alex Kipman para el control de la consola de Microsoft XBOX 360 sin la necesidad de utilizar cables conexiones u objetos físicos, dicho periférico funciona mediante una cámara de video , un emisor y un receptor de haces de luz infrarroja, los mismos indican la profundidad del entorno capturado por el dispositivo, la salida al mercado de este dispositivo revoluciono el control de aplicaciones de software y hardware desplazando al uso que se le estaba dando al Wiimote y el uso de acelerómetros en el control y la interacción con entornos en realidad aumentada.

Este dispositivo ha sido utilizado generalmente para la ayuda de personas con discapacidad visual, la robótica se ha integrado de manera óptima a este dispositivo logrando capturar el entorno con procesamiento en tiempo real, considerando en este principio y que la UPS no tiene una aplicación práctica del mismo se ha planteado la utilización del brazo robótico Mitsubishi RV-2AJ a través del dispositivo Kinect.

El prototipo será construido con la finalidad de dar soporte de manera remota en la programación del brazo robótico Mitsubishi RV-2AJ.

CAPÍTULO 1

MARCO TEÓRICO

1.1. Orígenes de la robótica

La palabra robótica nace etimológicamente del idioma checo, a partir de la unión de dos términos: robota que se define como "trabajo forzado" y en rabota que es sinónimo de "servidumbre". De la misma manera, debe notarse que la primera vez que se hace referencia a ella fue en el año 1920 en la obra del escritor KarelCapek titulada "Los robots universales de Rossum" (KarelCapek, 1920).

La robótica es la ciencia que involucra el diseño, la fabricación y la utilización de robots, en tanto que un robot es una máquina con componentes mecánicos y electrónicos que puede programarse para que interactúe con objetos de su entorno realizando tareas específicas y lograr que imite, en cierta forma, el comportamiento humano o animal.

La informática, electrónica, mecánica e ingeniería son disciplinas que se combinan para formar parte de la robótica. Su objetivo es la construcción de dispositivos que funcionen de manera automática o controlada y que realicen trabajos dificultosos o imposibles para los seres humanos debido al ambiente donde se realizan o a la fuerza y precisión que son necesarios para llevarlos a cabo.

El concepto de la creación de máquinas que pueden operar de manera autónoma se remontan a la época clásica, pero la investigación de la funcionalidad y los usos potenciales de los robots no creció sustancialmente sino hasta el siglo 20 una vez que la revolución industrial estuvo en auge, aunque se conocen herramientas o artilugios mecánicos que se remontan mucho antes.

1.2. Historia de la robótica

Desde la antigüedad el hombre ha sentido la fascinación por máquinas que imitan la figura y movimientos de seres animados, de forma que han creado artesanalmente autómatas, desde el imperio griego pasando por las sociedades francesa y suiza del

siglo XVIII, y hasta el siglo XX con interesantes e ingeniosos dispositivos mecánicos para el control automático de movimientos.

La llegada de la robótica se inició en el año 350 A.C. cuando un matemático griego Arquitas de Tarento construyó un ave mecánica, que se llamaba "La Paloma", que fue alimentada con vapor, también el ingeniero Herón de Alejandría (10-70 d. C.) creó numerosos dispositivos automáticos que los usuarios podían modificar, y describió máquinas accionadas por presión de aire, vapor y agua. Por su parte, el estudioso chino Su Song levantó una torre de reloj en 1088 con figuras mecánicas que daban las campanadas de las horas.

Al Jazarií (1136–1206), un inventor musulmán de la dinastía Artuqid, diseñó y construyó una serie de máquinas automatizadas, entre los que había útiles de cocina, autómatas musicales que funcionaban con agua, y en 1206 los primeros robots humanoides programables. Las máquinas tenían el aspecto de cuatro músicos a bordo de un bote en un lago, entreteniendo a los invitados en las fiestas reales. Su mecanismo tenía un tambor programable con clavijas que chocaban con pequeñas palancas que accionaban instrumentos de percusión. Podían cambiarse los ritmos y patrones que tocaba el tamborilero moviendo las clavijas.

Con nuevos avances, Leonardo Da Vinci en el año 1495 diseñó un dispositivo mecánico que parecía un caballero armado. En 1898, Nikola Tesla diseñó el primer robot a control remoto en el Madison Square Garden.

El robot diseñado fue modelado después de un barco. Los primeros robots industriales fueron Unimates desarrolladas por George Devol y JoeEngelberger a finales de los años 50 y principios de los 60. Por lo tanto, JoeEngelberger se ha llamado el "padre de la robótica".

George DevolJr, en 1954 desarrolló el brazo artificial multi-articulado, que es el inicio de los robots modernos. El escritor Isaac Asimov (1920-1992) suele ser considerado como el responsable del concepto de robótica. Este autor, especializado en obras de ciencia ficción y divulgación científica, propuso las Tres leyes de la robótica, una especie de normativa que regula el accionar de los robots de sus libros de ficción pero que, de alcanzarse un grado de desarrollo tecnológico semejante, podrían aplicarse en

la realidad futura. Dichas reglas son impresas como fórmulas matemáticas en los "senderos positrónicos" de la memoria del robot.

La primera ley de la robótica señala que "un robot no debe dañar a una persona o dejar que una persona sufra un daño por su falta de acción". La segunda ley afirma que "un robot debe cumplir con todas las órdenes que le dicta un humano, con la salvedad que se produce si estas órdenes fueran contradictorias respecto a la primera ley". La Tercera ley establece que "un robot debe cuidar su propia integridad, excepto cuando esta protección genera un inconveniente con la primera o la segunda ley". (Asimov, 1950).

Robótica móvil hizo lo propio en 1983, cuando Odetics presentó un vehículo que tenía como sistema de locomoción seis patas, mediante estas era capaz de trepar por encima de objetos. Este robot podía levantar más de 5 y 6 veces su propio peso estacionado y 2.3 veces su peso en movimiento. Los robots completamente autónomos aparecieron en la segunda mitad del siglo XX. El primer robot con control digital y programable fue el Unimate, se instaló en 1961 para levantar y apilar piezas calientes de metal de una máquina de fundición a presión.

Los robots comerciales e industriales son muy comunes hoy en día, utilizados para realizar trabajos de forma más barata, precisa y fiable, que los seres humanos, también se emplean en trabajos que son demasiado repetitivos, peligrosos o nada adecuados para los seres humanos.

Los robots son ampliamente utilizados en manufactura, ensamble, empaque y embalaje, el transporte, la exploración espacial, la cirugía, armamento, laboratorios de investigación, seguridad, y la producción masiva de bienes de consumo e industriales.

1.3. Clasificación de los robots

Hay varios tipos de robots, que se usan en el mundo moderno ejecutando varias tareas como por ejemplo en función de la inteligencia aplicada sin embargo, los robots se pueden clasificar a grandes rasgos tomando en cuenta como su principal característica su movilidad, se los clasifica en dos tipos:

- Robots móviles autónomos.
- Robots industriales.

1.3.1. Robots móviles autónomos.

Son mecanismos o dispositivos que poseen un medio de locomoción, y pueden desplazarse libremente sin tener una fijación directa a un plano, además, pueden estar provistos de sensores utilizados para detección de objetos circundantes y localización respecto a un punto de referencia ,mediante la planificación de trayectorias pueden realizar movimientos adaptándose a su entorno, algunos de los ejemplos de robots móviles autónomos son vehículos guiados por un sistema de control en el cual no intervienen acciones humanas externas, sino por ejemplo pueden ser direccionados utilizando guías eléctricas especiales que utilizan un sensor de medida o pueden tener algoritmos de control que tracen rutas alternativas dependiendo de la posición de objetos externos.

En el ámbito militar, algunos robots están diseñados para detectar objetos peligrosos como bombas y además algunos son capaces de desactivarlas, la mayoría de estos robots son autónomos, ya que han sido creados con la destreza para detectar y adaptarse al entorno en el que realizan su tarea.

1.3.2. Robots industriales.

Existen ciertas dificultades a la hora de establecer una definición formal de lo que es un robot industrial. “La primera de ellas surge de la diferencia conceptual entre el mercado japonés y el euro-americano de lo que es un robot y lo que es un manipulador” (Alonso, 2004). Así, mientras que para los japoneses un robot industrial es cualquier dispositivo mecánico dotado de articulaciones móviles destinado a la manipulación, el mercado occidental es más restrictivo, exigiendo una mayor complejidad, sobre todo en lo relativo al control. “En segundo lugar, y centrándose ya en el concepto occidental, aunque existe una idea común acerca de lo que es un robot industrial, no es fácil ponerse de acuerdo a la hora de determinar una definición formal” (Robots industriales, 2014), además la evolución de la robótica ha ido obligando a diferentes actualizaciones de su definición.

La definición más común y aceptada podría ser la de la Asociación de Industrias de robótica según la cual:

"Un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas" (Robotic Industry Association).

Esta definición, ligeramente modificada, ha sido adoptada por la Organización Internacional de Estándares que define al robot industrial como:

"Manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas" (Organización Internacional de Estándares, 2004).

1.4. Campos de aplicación de la robótica

“El uso de sistemas robóticos puede extenderse a casi todas las áreas en donde se necesite de la ejecución de tareas mecánicas y supervisión, como por ejemplo una exploración sobre el terreno de la superficie marciana o lunar” (Robótica y aplicaciones, 2014) . Se entiende, en este contexto, que tarea mecánica es toda actividad que involucra presencia física y movimiento por parte de su ejecutor, la de supervisión como una actividad en la que se necesita verificar condiciones favorables para que un producto sea aprobado o pase a otra etapa de elaboración, a continuación se exponen algunos campos de aplicación de la robótica.

- **Investigación y exploración**

Es un campo donde los robots presentan la ventaja de resistir mejor los medioambientes hostiles para el ser humano además de operar varios días seguidos sin descanso alguno.

- **Construcción**

Dentro de esta industria se registran proyectos que incluyen el uso de robots como ejecutores de tareas de dimensionamiento, transporte, montaje, entre otras.

- **Automatización industrial**

Es el más relevante y de mayor interés en el ámbito aplicable de la rama, corresponde al uso de robots a fin de mejorar, agilizar y aumentar la producción en los diferentes procesos.

1.4.1. Factores limitantes.

Las aplicaciones de los sistemas robóticos podrían ser innumerables. Pero existen varios factores que impiden el desarrollo y ejecución entre estos se mencionara dos que son fuertes y decisivos y que inhiben el crecimiento y desarrollo de esta tecnología. Estos a considerar son:

- **Limitaciones económicas**

Dado que la robótica es una disciplina de avanzada y en desarrollo, los costos asociados a ella son altísimos, se necesitan numerosos recursos no sólo para su construcción sino también para el desarrollo de tecnología inexistente aun, hay muchas áreas de investigación relacionadas que necesitan también una fuerte inyección de dinero y hacen que en la actualidad un sistema robótico de producción no masiva sea un producto con alto costo económico e investigativo además limitando a que este sea masificado.

- **Limitaciones tecnológicas**

"Un campo de investigación como la robótica está orientado a tratar de llevar a la práctica ideas que pueden haber sido concebidas hace ya mucho tiempo" (Robótica y aplicaciones, 2014). Además, el factor económico, la concreción de dichas ideas dependerá de que se hayan encontrado o desarrollado los medios tecnológicos que la permitan, adaptándonos a nuestro entorno, al estar en un país en vías de desarrollo es muy complicado conseguir material prima y productos desarrollados orientados al desarrollo tecnológico de esta ya sea porque sus costos son altos o no existen en nuestro medio.

1.5. Manipulador robótico

Se entiende por manipulador robótico a un conjunto de segmentos y articulaciones que juntas realizan un trabajo determinado, dependiendo del tipo de manipulador pueden ser convenientemente divididas en tres secciones: el brazo, que consta de uno o más segmentos y articulaciones, la muñeca, y un efector final el cual dependiendo del trabajo al que va a ser destinado dicho manipulador será diseñado e incluido en la fisionomía del robot, debido al origen de este trabajo se hablara de efectores finales con medios de fijación, alternativamente, el manipulador se puede dividir en dos únicas secciones, el brazo y el efector final además de que la muñeca que se encarga de la unión del efector al resto del robot se separa como su propia sección, ya que realiza una función única.

Los robots industriales son manipuladores fijos cuya base se instala permanentemente en un plano pudiendo ser este el suelo o una estación de trabajo, las articulaciones y elementos se enumeran progresivamente desde la base hacia el final, se conoce como grado de libertad a la capacidad del robot para cambiar de posición un segmento dado de este, cada grado de libertad está constituido por la combinación de una articulación y un elemento.

La parte mecánica del robot está compuesta por diversas articulaciones, normalmente se distingue entre el brazo y el efector final que puede ser intercambiable, empleando dispositivos específicos para distintas tareas determinadas.

El aumento del número de grados de libertad aporta mayor maniobrabilidad pero complica aún más el problema del control, esto conlleva a que el manipulador sea menos preciso debido a la acumulación de errores, lo mismo que se ve reflejado en los requerimientos de software necesario para cumplir esta tarea, ya que las ecuaciones que son necesarias para describir el movimiento del manipulador son ecuaciones diferenciales no lineales y resulta difícil obtener soluciones analíticas.

1.5.1. Posicionamiento, orientación y grados de libertad.

En general, el brazo y la muñeca de un manipulador de base realizan dos funciones separadas, de posicionamiento y orientación. En el brazo humano, el hombro y el codo hacen lo propio y el posicionamiento de la muñeca hace la orientación. Cada articulación permite un grado de libertad de movimiento. El número mínimo teórico de grados de libertad para llegar a cualquier lugar en el área de trabajo y orientar la pinza en cualquier orientación es de seis, tres por su ubicación, y tres para la orientación.

Los tres movimientos de torsión que proporcionan orientación son generalmente denominados cabeceo, balanceo y guiñada, para inclinar hacia arriba / abajo, torciendo y doblando hacia la izquierda / derecha, respectivamente. Un buen ejemplo de un manipulador es el brazo humano, que consiste en un hombro, brazo, codo y muñeca. El hombro permite que el brazo superior pueda moverse hacia arriba y hacia abajo que se considera un grado de libertad (GDL). Permite un movimiento hacia adelante y hacia atrás, que es el segundo GDL, a su vez también permite la rotación, que es el tercer GDL. La articulación del codo da el cuarto GDL. La muñeca lanza hacia arriba y abajo, izquierda y derecha, y gira, dando tres GDL en una articulación.

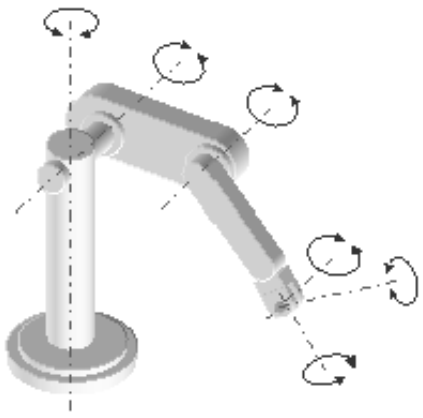


Figura 3. Grados de Libertad de un Robot Manipulador

Fuente: (Platea , s.f.)

1.5.2. Estructura mecánica del robot manipulador.

El sistema mecánico consiste en una estructura de mecanismos de posicionamiento y orientación del efector final, incluyendo una base. Los robots manipuladores son, brazos articulados, es decir, una cadena formada por un conjunto de eslabones o elementos interconectados mediante articulaciones, las cuales permiten el movimiento relativo entre eslabones consecutivos. Cada uno de los movimientos independientes que puede realizar cada articulación (de rotación o traslación) se llama grado de libertad (GDL), éstos son los que permiten posicionar y orientar en el espacio al efector final. Los robots industriales varían entre 4 y 6 GDL, Sin embargo, existen robots con más GDL para aplicaciones especiales.

1.5.2.1. Tipos de articulaciones.

Existen diferentes tipos de articulaciones, las más utilizadas en robótica son las siguientes:

La articulación de rotación tiene 1 GDL se desplaza variando el ángulo del segmento haciendo eje en la articulación.

La articulación de traslación lineal tiene 1 GDL a lo largo del eje de la articulación.

En la articulación cilíndrica existen dos GDL.: Uno de rotación y uno de traslación.

La articulación esférica combina giros en tres direcciones rotacionales en el espacio (2 GDL).

Estructuras o configuraciones clásicas

“La estructura típica de un manipulador consiste en un brazo compuesto por elementos con articulaciones entre ellos. En el último enlace se coloca un órgano terminal o efector final tal como una pinza o un dispositivo especial para realizar operaciones” (Robótica y aplicaciones, 2014).

“Se consideran, en primer lugar, las estructuras más utilizadas como brazo de un robot manipulador” (Baturone, 2001). Estas estructuras tienen diferentes propiedades en cuanto a espacio de trabajo y accesibilidad a posiciones determinadas.

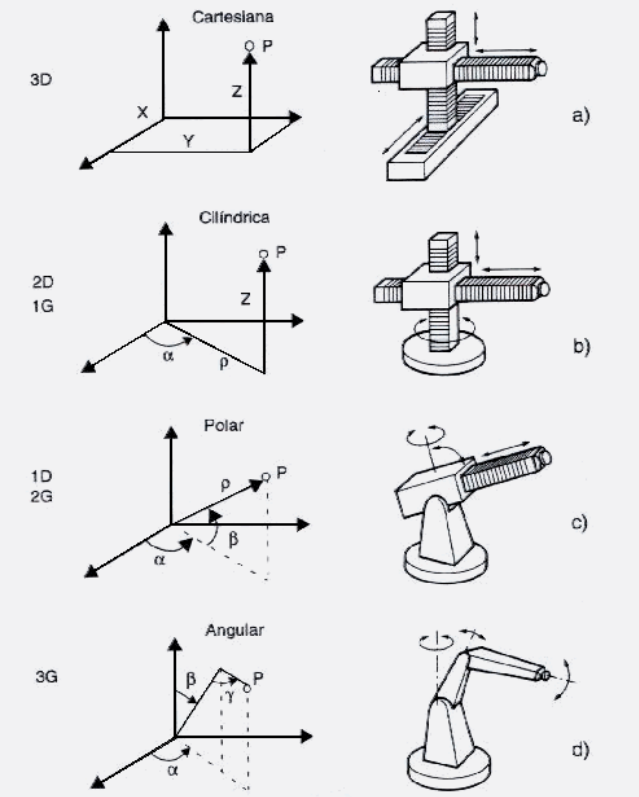


Figura 4. Estructuras básicas de manipuladores

Fuente: (Estación Automatizada, s.f.)

La ubicación del TCP se especifica en algún sistema coordenado, fundamentalmente dado por la configuración mecánica del robot. Existen las siguientes configuraciones clásicas en los manipuladores:

Cartesiana

Tiene 3 articulaciones traslacionales y la posición del TCP se da en coordenadas cartesianas o rectangulares.

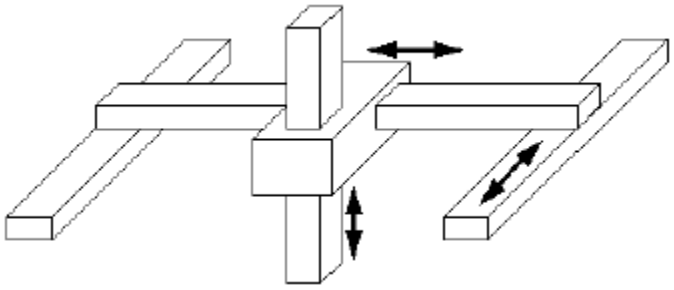


Figura 5. Configuración cartesiana

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Cilíndrico

Tiene 2 articulaciones traslacionales y 1 rotacional, la posición del TCP se da en coordenadas cilíndricas.

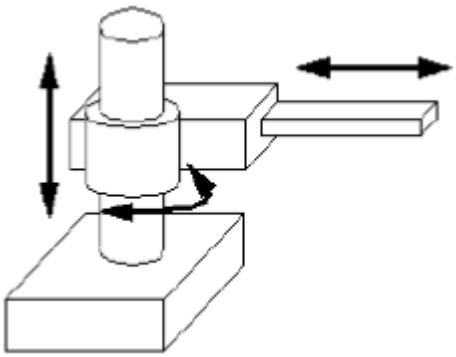


Figura 6. Configuración cilíndrica

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esférico

"Tiene 2 articulaciones rotacionales y 1 traslacional, la posición del TCP se da en coordenadas esféricas o polares" (Liñan, 1995).

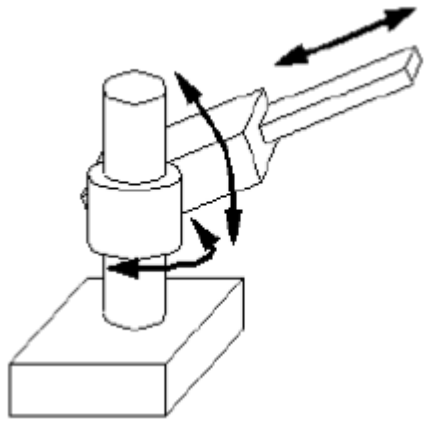


Figura 7. Configuración esférica

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Angular

Tiene sólo articulaciones rotacionales y la posición del TCP se da en coordenadas angulares relativas a cada eslabón consecutivo. La mayoría de este tipo se parece al brazo humano con torso, hombro, codo y muñeca.

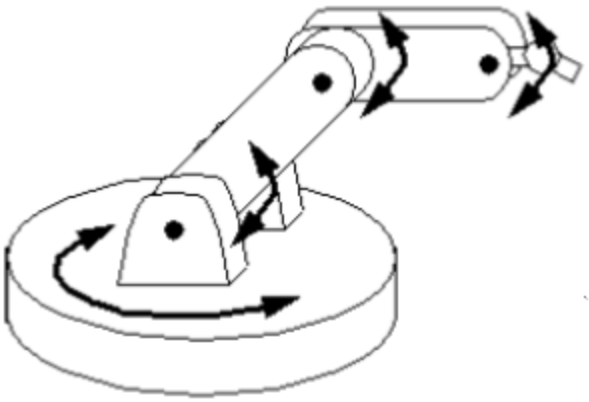


Figura 8. Configuración angular

Elaborado por: (Castro, Slin & Jiménez, Fernando)

SCARA (Selective Compliance Assembly Robot Arms)

Especial para montaje o ensamble en un plano. Tiene 3 articulaciones de rotación respecto a 2 ejes paralelos, y una de traslación en sentido perpendicular al plano.

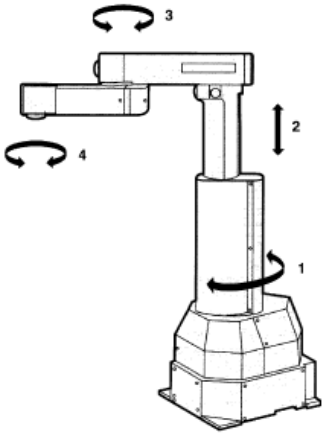


Figura 9. Configuración SCARA

Elaborado por: (Castro, Slin & Jiménez, Fernando)

El espacio de trabajo es el conjunto de puntos en el espacio que el TCP del efector final puede alcanzar, lo cual depende de la estructura mecánica. Para un robot cartesiano (como una grúa) el espacio de trabajo podría ser un cubo, para los robots más sofisticados los espacios podrían ser de una forma esférica.

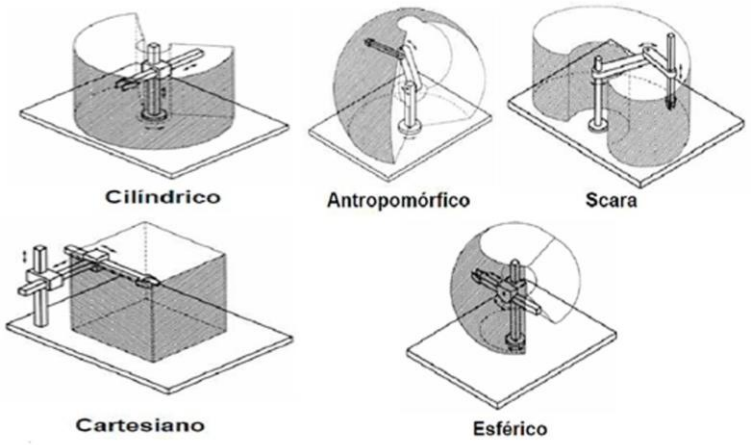


Figura 10. Espacio de trabajo estructuras básicas manipuladores

Fuente: (Ollero Baturone, 2001)

La resolución espacial (precisión) es el incremento más pequeño de movimiento en que el robot puede dividir su espacio de trabajo. Ésta depende de dos factores: los sistemas de control del movimiento, y las inexactitudes o limitaciones de los

actuadores y de la estructura mecánica. Es más fácil de conceptuar estos factores cuando se refiere a un grado de libertad.

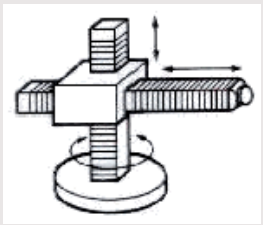
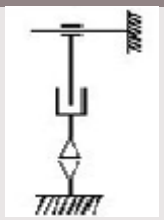
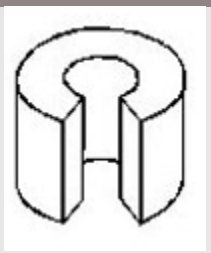
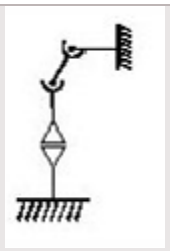
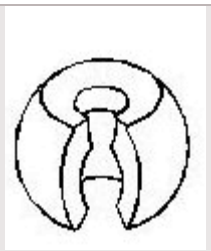
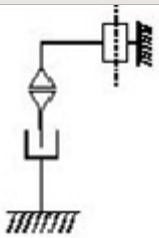
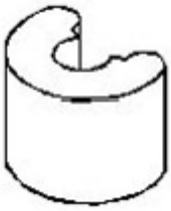
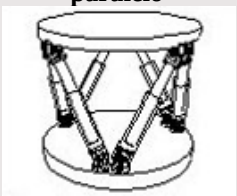
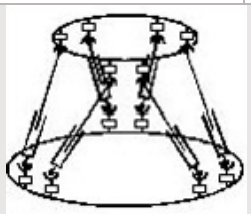
Configuración geométrica	Estructura cinemática	espacio de trabajo	Ejemplo
cilíndrico 			
polar 			
esférico 			
SCARA 			
paralelo 			

Figura 11. Configuraciones geométricas, estructura cinemática, espacio de accesibilidad y ejemplos de robots industriales

Elaborado por: (Castro, Slin & Jiménez, Fernando)

1.5.2.2. Actuadores en robots

Los actuadores son dispositivos que generan la fuerza necesaria para mover o animar a la estructura mecánica. Estos pueden ser según la energía que consuman clasificados en de tipo neumático, hidráulico o eléctrico.

La energía neumática dota a sus actuadores de una gran velocidad de respuesta, junto a un bajo coste, pero de precisión limitada, teniendo por ejemplo a cilindros de simple o doble efecto y motores neumáticos (de aletas rotativas o pistones axiales) (Articulaciones Robóticas, s.f.) .Los actuadores de tipo hidráulico se destinan a tareas que requieren una gran potencia y grandes capacidades de carga.

Existen como en el caso de los neumáticos, actuadores de tipo cilindro y del tipo de motores de aletas y pistones. "El grado de compresibilidad de los aceites usados es considerablemente menor al del aire, por lo que la precisión obtenida en este caso es mayor. Por motivos similares, es más fácil en ellos realizar un control continuo" (IEEE, 2013), pudiendo posicionar su eje en todo un rango de valores (haciendo uso de servocontrol) con notable precisión. Presenta desventajas por fugas de aceite debido a las excesivas presiones, además requiere equipos de filtrado de partículas, eliminación de aire, sistemas de refrigeración y unidades de control de distribución.

"Electromecánicos, como motores eléctricos que cubren la gama de media y baja potencia, acaparan el campo de la robótica, por su gran precisión en el control de su movimiento y las ventajas inherentes al manejo de sus variables eléctricas" (Jones, 1993). Entre los motores eléctricos utilizados en robótica se puede mencionar los motores de corriente directa servo controlado, motor paso a paso y otros actuadores electromecánicos sin escobillas.

1.6. Visión artificial

1.6.1. *La luz.*

La primera teoría sobre el comportamiento de la luz fue proporcionada por Euclides en el 330 antes de Cristo, el suponía que la luz era una especie de rayo que salía del ojo y alcanzaba el objeto, aunque poseía varios vacíos e incongruencias fue aceptada. Paso bastante tiempo para que en el año 1000 D.C. Alhacen afirmó que la luz procedía de una fuente y se dirigía a nuestros ojos después de ser reflejada por los objetos

circundantes. En el siglo XVII fue cuando se realizaron los mayores progresos de la mano de Newton, se hicieron notables avances en la "teoría del color" y la "dispersión". Newton era defensor de la "teoría corpuscular", según esta la luz estaba formada por un flujo de partículas proyectadas por un cuerpo luminoso. Al mismo tiempo, otros científicos, como Hooke y Huygens, defendían una teoría ondulatoria que explicaba mejor ciertos hechos como por ejemplo que dos haces luminosos se crucen sin perturbarse. En este modelo se define el concepto de rayo de luz como una línea imaginaria paralela a la dirección de propagación de la onda. El principal problema de la teoría ondulatoria consistía en que no existía ninguna evidencia empírica del medio en el que necesitaba la onda para propagarse dicho medio debía estar en todo el espacio, a este se lo denominó éter, se razonó que debería tener una alta densidad para permitir la velocidad de propagación que caracteriza a la luz. Por ello, debido a que Newton era mucho más conocido y respetado, la teoría corpuscular se impuso alrededor de doscientos años a la ondulatoria.

En el siglo XIX, los trabajos de Young, Fresnel y Foucault corroboraron la mayoría de las objeciones propuestas por Newton a la teoría ondulatoria teniendo un sinfín de éxitos en el campo de la óptica. Maxwell en 1873 con su teoría electromagnética dio el impulso definitivo a favor de la naturaleza ondulatoria de la luz, explicaba a la luz como una radiación con naturaleza ondulatoria que se puede propagar en el vacío, de esta manera no se necesitaba el concepto de éter.

En 1900 se percibió un nuevo fenómeno que fue llamado "efecto fotoeléctrico", que proporcionó evidencias experimentales de que la luz tenía también carácter corpuscular en su interacción con la materia, esto llevó de nuevo al replanteamiento de la naturaleza de la luz. Albert Einstein, Louis de Broglie y otros desarrollaron una nueva teoría llamada "teoría onda-corpúsculo" considerando que la luz estaba formada por unas partículas, los fotones, cada una de las cuales tiene asociada una ecuación de ondas, con esto cuando la luz interactúa con la materia , como en el efecto fotoeléctrico, se utiliza un modelo corpuscular para poder explicar dicha interacción, mientras que para explicar fenómenos relativos a su propagación, como en la difracción se aplica el modelo ondulatorio. Este modelo dual, onda-corpúsculo, permite explicar la totalidad de los fenómenos producidos por la luz.

1.6.2. Visión humana.

La visión es la capacidad del ser humano para percibir e interpretar el entorno gracias a radiaciones luminosas producidas por una fuente y reflejadas por los objetos presentes en el entorno, para que estas sean percibidas por el ojo deben poseer una longitud de onda entre 380nm y 760nm aproximadamente, el ojo es el órgano principal del sistema de visión, dentro de este la luz percibida se transforma en impulsos nerviosos que son transportados mediante el nervio óptico al cerebro , el sistema de visión se encuentra formado por el globo ocular, esta es una esfera compuesta de líquido transparente conocido como humor acuoso y se encuentra en la cámara anterior del ojo y contribuye a la refracción de los rayos de luz para que converjan en la retina, en dicho proceso la luz reflejada en los objetos es enfocada mediante la córnea y el cristalino, penetra por la pupila atravesando el humor acuoso y se proyecta en la retina donde se encuentran células foto receptoras conocidas como conos y bastones las mismas que lo transforman en impulsos nerviosos y mediante el nervio óptico son llevadas hacia el córtex.

En el cerebro los impulsos son procesados e interpretados en imágenes, existen varios tipos de visión como son la monocular y binocular las que nos permiten percibir mediante factores psicológicos profundidad y relieve.

1.6.3. Historia.

En la década de los sesenta se empieza a incursionar en este campo, se deseaba conectar una cámara a una computadora mediante una interfaz para obtener datos de ella, precisamente en 1961 un científico norteamericano llamado Larry Roberts dio la pauta para el inicio de la visión artificial al crear un programa llamado “el mundo de micro-bloques” el cual con la ayuda de la cámara pudo “mirar”.

Dicho programa “El mundo de micro-bloques” consistía en una cámara y un computador mediante los cuales se podía percibir un grupo de bloques agrupados, y mediante el procesamiento de imágenes los mostraba en otra perspectiva.

La lógica del programa se centraba en las variaciones de la intensidad de grises de la imagen que era adquirida para delimitar cada lado del bloque y la unión de estos bordes para obtener las esquinas, demostrando así que se estaba procesando lo que se captaba

por la cámara, además de con un conjunto de cálculos mostrarlo de una forma distinta. Este programa era limitado ya que no podía identificar bloques que no estuvieran almacenados previamente dentro de la memoria del programa.

La velocidad de procesamiento de las computadoras fue el problema principal para que este campo no se desarrolle lo suficiente, en parte debido que para procesar una imagen se necesitaba una cantidad muy grande de tiempo y además consumía todos los recursos del computador.

En la década de los 90 aparecieron computadores con mejores características que eran una gran ventaja y con los cuales lograron mejorar la apreciación de este tipo de aplicaciones. Las primeras tareas de visión artificial se basaron en el reconocimiento de formas, segmentación de objetos, y filtrado de bordes, estos programas empezaron a utilizarse industrialmente lo que ayudo a su desarrollo porque entonces se empezaron a invertir más cantidades de capital en el estudio de este campo.

Junto a la mejora de la velocidad de procesamiento de los ordenadores también está el avance en varias aéreas, la base de los algoritmos de visión artificial tales como la geometría proyectiva, pares estéreo y auto calibración han abierto la puerta para utilizar información tridimensional dentro de este proceso.

1.6.4. Conceptos de visión artificial.

La visión artificial es recolectar información del mundo que lo rodea a través de una cámara en forma de imágenes, pasando primero por varios pasos como la mejora de iluminación, luego de un proceso de digitalización donde se transforma a los colores en vectores de color y posición pasar a procesar obteniendo histogramas, gráficos estadísticos, umbrales y patrones utilizando la información que esta provee para representar el ambiente.

Representación de la realidad

La representación de la realidad mediante un computador se consigue a través de la recolección de información desde las imágenes que se capturan, obteniendo de estas información como brillo, intensidad de colores, formas etc. aunque la mayoría de ocasiones se utilizan imágenes estáticas también se puede procesar video ya que estos

son un conjunto de cuadros de imágenes todo depende de la finalidad de la aplicación del programa y de la capacidad de procesamiento del computador utilizado.

Una imagen bidimensional digitalizada consiste en un conjunto de bytes en estos se almacenan magnitudes como son la intensidad de color o canales de color, dependiendo del estándar utilizado para el procesamiento de la imagen como pueden ser RGB o CMYK además de la posición de dichos colores representados por coordenadas polares del plano cartesiano X & Y.

En este trabajo de titulación además de utilizar imágenes bidimensionales se utilizará un componente más siendo este la profundidad de los objetos capturados, teniendo una representación en tres dimensiones del entorno en el que se trabaja.

1.6.5. Etapas de un sistema de visión artificial.

En el sistema de visión humano los elementos que intervienen e interactúan como lo son luz, el objeto, el cerebro y el ojo humano, dentro de un sistema de visión artificial intervienen los mismos elementos cambiando el ojo humano por la cámara y el cerebro por la computadora, existen varias razones para creer que el primer paso dentro de este procesamiento consiste en encontrar los elementos más simples que componen el entorno y descomponerlos (como lo son los segmentos y arcos).

Después que el cerebro interpreta la escena produce una orden para actuar en consecuencia para percibirlo. La visión artificial hace un intento por imitar dicho comportamiento, dentro de esta se identifican cuatro fases principales:

La primera fase es puramente sensorial, consiste en la captura o adquisición de las imágenes digitales mediante algún tipo de sensor que tradicionalmente es una cámara.

La segunda etapa consiste en el tratamiento (procesamiento) digital de las imágenes, con la finalidad de facilitar la identificación de patrones como lo son los bordes y las esquinas en etapas posteriores. Dentro de esta etapa de procesamiento es donde, mediante filtros y transformaciones geométricas, se eliminan partes indeseables de la imagen y ruidos.

La siguiente fase se conoce como segmentación, y consiste en aislar los elementos que componen una escena para su procesamiento individual.

Por último se llega a la etapa consiste en el reconocimiento o clasificación para los objetivos finales de la aplicación. En ella se pretende distinguir los objetos segmentados, gracias al análisis de ciertas características que se establecen previamente para diferenciarlos.

Las fases descritas anteriormente no siempre se cumplen en este orden sino que varían dependiendo del objetivo final.

En el desarrollo del presente documento se explicaran más a profundidad las diferentes etapas utilizadas y las características principales que los rodean.

1.7. El Kinect

1.7.1. Historia.

El Kinect fue concebido originalmente con el nombre de " Proyecto Natal" es un controlador de juego libre utilizado para la consola desarrollada por Microsoft la XBOX 360 y actualmente la XBOX ONE , posteriormente fue introducida en el sistema operativo del computador en las versiones del sistema Operativo Windows en sus versiones 7 y 8 como Framework oficial aunque además existen varias distribuciones de controladores para Linux , el Kinect permite a los usuarios interactuar con el sistema sin la necesidad de tener contacto físico con este , mediante dicho periférico se puede reconocer gestos, objetos, imágenes y comandos de voz, este dispositivo fue lanzado en América del Norte el 4 de noviembre del 2010.

De acuerdo con Robert J.Bach Microsoft Research invirtió veinte años en el desarrollo de la tecnología Kinect el anuncio de este dispositivo fue en la ElectronicEntertainment Expo 2009 con su nombre Clave "Proyect Natal".

También se tiene conocimiento de que el hardware está basado en un diseño de referencia y la tecnología 3D-calor fabricados por la compañía israelí de desarrollo PrimeSense Ltd., Microsoft estuvo desarrollando un dispositivo de interfaz natural desde mucho antes, pero el modelo previsto por esta empresa se adaptó a sus necesidades la misma que antes de ofrecer esta tecnología a Microsoft lo ofreció a Apple.

Durante este tiempo de desarrollo Microsoft Research dedico innumerables horas de trabajo en el acoplamiento de diferentes tecnologías para lograr concebir este dispositivo, dentro del funcionamiento de este se puede distinguir el procesamiento de imágenes, percepción de profundidad y procesamiento de voz.

Figura 12. Dispositivo Kinect

Fuente: (Microsoft, s.f.)

1.7.2. Características.

El Kinect es una barra horizontal de aproximadamente 23 cm (9 pulgadas) conectada a una base circular con un eje de articulación de rótula con un motor. Este dispositivo, Está formado por una cámara, un sensor de profundidad, un arreglo de 4 micrófonos y un procesador integrado. Su funcionamiento fue adaptado y distribuido junto con la consola XBOX 360, en la actualidad es compatible con Windows 7 con el software licenciado, aunque existen varias versiones de controladores no oficiales que funcionan sobre plataformas LINUX y Android que figuran entre los más conocidos.

1.7.3. Funcionamiento.

Este dispositivo basa su funcionamiento en hardware y software, el dispositivo se conecta al terminal con un puerto que respeta los estándares USB pero posee una variación en su estructura ya que necesita aparte de los 5 v que provee el puerto 12 voltios adiciónales para su funcionamiento .

Figura 13. Puerto Kinect

Fuente: (Blogcdn, s.f.)

Hardware

Este dispositivo consta de una parte mecánica y motora la cual está concentrada en la base del mismo mediante un sistema de servo control, dentro de esta se encuentra el centro de movimiento conformado por un motor DC con una caja de moto reducción los cuales proveen movimiento vertical en alrededor de 30 grados, este sirve para enfocar el entorno cuando este ha salido de foco además cuenta con un acelerómetro para verificar la inclinación.

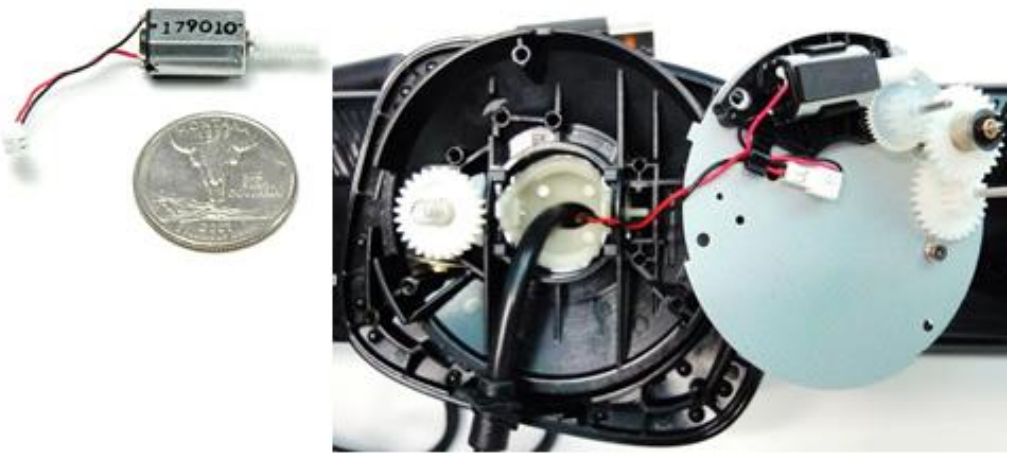


Figura 14. Parte Mecánica del dispositivo Kinect

Fuente: (Parte Mecánica del Dispositivo Kinect, s.f.)

Control

Está conformada por una placa base la cual se encarga de recibir y transmitir todos los datos recogidos por los sensores mediante una interfaz USB, posee una unidad de memoria DDR2SDRAM de 64 MB, una memoria Flash NOR de 1 Mb.

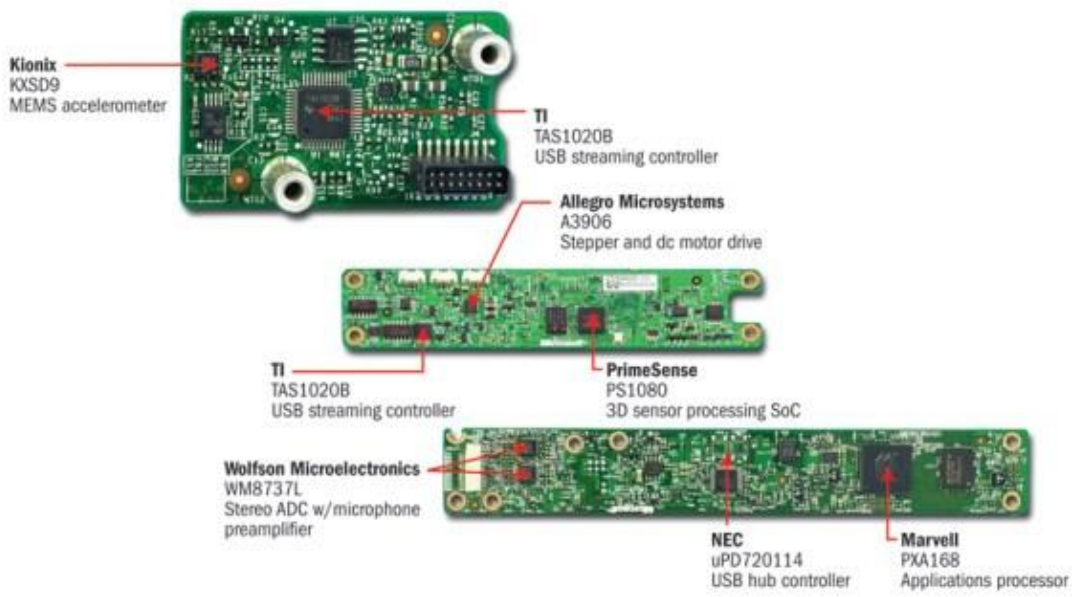


Figura 15. Hardware del dispositivo Kinect

Fuente: (Hardware del Dispositivo Kinect, s.f.)

Los sensores infrarrojos son sensores de imágenes CMOS de Aptina Imaging y el modelo elegido utiliza el sensor de infrarrojos MT9M001, con píxeles de 5,2 micras,

"Por el lado de la entrada de información RGB se utiliza una cámara color, con un sensor MT9M112 y los dos sensores mencionados poseen una resolución de 1,3 mega-píxeles" (Microsoft, s.f.). La interfaz utilizada para dar forma al "procesamiento sensorial" es un chip PS1080 de PrimeSense, que se encarga de todos los controles del sistema, del proyector de infrarrojos, de los procesos de la información que cada cámara recoge y de las entradas de audio, el PS1080 establece su enlace a través de una conectividad USB 2.0 hacia uno de los procesadores principales de aplicaciones dentro de Kinect, se tiene también convertidores analógicos-digitales estéreo, WM8737L de Wolfson Microelectronics, encargados de realizar la pre amplificación

de sonido para la matriz de micrófonos para el control de inclinación cuenta con un acelerómetro MEMS, el KXSD9 de Kionix.

> Un controlador de un motor paso a paso, el A3906 de Allegro Microsystems, también se tiene otros componentes como son el controlador uPD720114 que cumple la función de concentrador USB de NEC y un par de circuitos integrados de Texas Instruments: el TAS1020B que es un controlador de audio USB Streaming y un ADS7830 que es un convertidor analógico-digital de ocho canales y de 8 bits (Microsoft, s.f.).

Sensado

Está compuesto por un sistema de visión artificial y una matriz de micrófonos para procesamiento de comandos por voz. En el reconocimiento de comandos de voz este dispositivo usa un arreglo de cuatro micrófonos, ya que así logrará tener datos suficientes y mediante el software cancelar el ruido de fondo, detectar la fuente del sonido y luego realizar el reconocimiento de comandos, el desarrollo de esta tecnología también tomo mucho tiempo se tiene registro de que en 1996 Microsoft Research invirtió más recursos en el procesamiento de lenguaje natural contratando a especialistas renombrados de todo el mundo.

Sistema de visión

La percepción de movimiento es la característica más destacada y explotada del Kinect y en esta sección se lo abordara profundamente, se la dividió en varias secciones para su mejor comprensión.

Elementos y captura de imágenes

Se distinguen aparte de la placa de control dos elementos principales que intervienen en el proceso de captura de imágenes, los mismos que son:

1. Una cámara RGB es un sensor CMOS el MT9M112, este se encarga de tomar la imagen del entorno con una resolución de 1.3 megapíxeles y provee al terminal una imagen plana de dos dimensiones donde se puede diferenciar los ejes X & Y, de estas imágenes que luego serán procesadas se obtendrán colores e intensidad de brillo.

2. Un emisor de rayos de luz infrarroja y un sensor infrarrojo CMOS de Aptina Imaging y el modelo elegido utiliza el sensor de infrarrojos MT9M001 los mismos que al funcionar en conjunto forman un sensor de profundidad que detecta la distancia de un objetivo con el principio de un sensor ultrasónico o un sonar en el cual primero se envía una señal en esta caso son varios haces de luz infrarroja estos rebotan en el objeto que se tenga enfrente regresan a receptor en donde el chip integrado del Kinect cuenta el tiempo de que se tardó en regresar y con esa magnitud se calcula la distancia la cual tiene un error promedio de 1 cm. y 3 mm. Con la imagen proporcionada por estos sensores se obtiene una imagen de profundidad y obtener un eje z para la captura total la cual al integrar la imagen provista por la cámara RGB brinda un entorno en 3D.

Detección de movimientos

Para el reconocimiento de imágenes Microsoft recopilo una base de datos de 500000 posiciones del cuerpo humano y luego lo redujeron a 100000 filtrando las imágenes y tomando aquellos frames donde la distancia entre una pose y otra sea no menor a 5 cm. En base a estas muestras, generaron un modelo para inferir la posición del cuerpo.

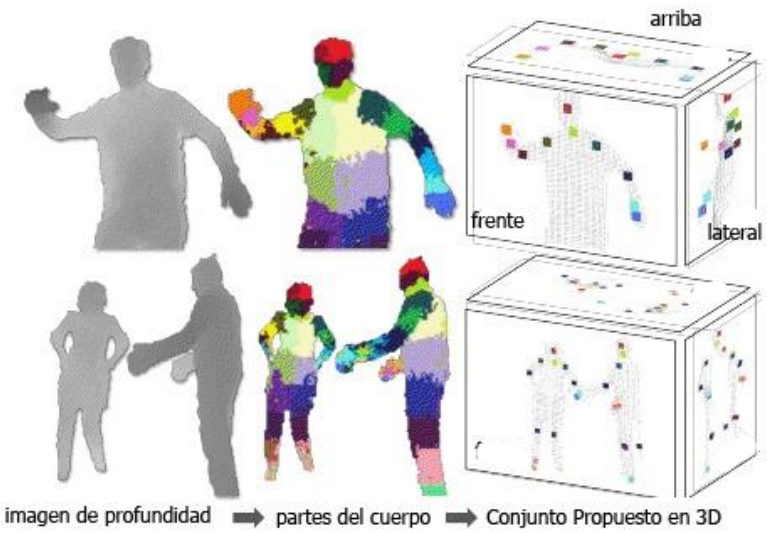


Figura 16. Detección de imágenes

Fuente: (Detección de Imágenes Dispositivo Kinect, s.f.)

Durante el proceso de captura este dispositivo procesa un frame cada 5 milisegundos, para reconocer en tiempo real la posición del cuerpo del jugador o los jugadores, el algoritmo identifica las principales partes del cuerpo con pixeles de distintos colores en tiempo real, utilizando 3 imágenes simples, interpola estos puntos y elabora la posible posición del cuerpo en base al modelo generado anteriormente.

Lo remarcable es que la velocidad de procesamiento del CPU de la consola Xbox 360 puede ejecutar este algoritmo a una velocidad de 200 frames por segundo para determinar la posición del cuerpo.

Manejo de imágenes con el SDK

Hay que tener en cuenta que pixel es la menor unidad homogénea en color que forma parte de una imagen digital. Gracias al SDK de Kinect que provee Microsoft para Windows se puede obtener los datos de las cámaras, Las imágenes que se obtienen del sensor se codifican en un vector de bytes mediante la tarjeta de control incrustada en este. Para entender esta codificación hay que tener presente como se estructura una imagen.

Cada pixel de la imagen se compone de 4 componentes que representan los canales RGB (colores rojo, verde y azul) más una componente que corresponde con el valor de transparencia (alfa), en el caso de imágenes RGB, o un valor vacío, si es de tipo RGB.

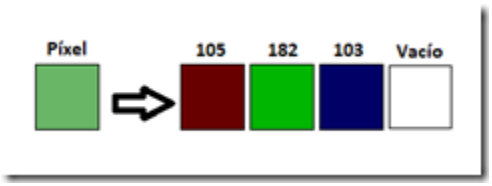


Figura 17. Pixel de imagen

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Cada componente del píxel tiene un valor decimal que varía entre 0 y 255 lo que corresponde a un byte. De esta forma el vector de bytes que se obtiene de la recolección del sensor (Cámara RGB), es una representación de esos píxeles organizados de arriba abajo y de izquierda a derecha como está definida la configuración del monitor de una

computadora donde los 4 primeros elementos del vector serán los valores rojo, verde, azul y alfa del píxel de arriba a la izquierda.

Figura 18. Componente pixel

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Cuando se usan las cámaras de profundidad el procedimiento cambia. Al igual que con la cámara RGB también se conseguirá un vector de bytes pero en esta ocasión esos bytes no corresponden a los canales RGB de la intensidad de color sino con la distancia del píxel al sensor.

Al tener 2 cámaras de infrarrojos cada píxel se corresponde con 2 bytes en el vector siendo éstos el valor de la distancia de ese píxel a cada cámara. La organización de los píxeles es la misma que con la cámara RGB, los 2 primeros bytes es la distancia del píxel de la posición de arriba a la izquierda al sensor y los 2 últimos son del píxel de abajo a la derecha.

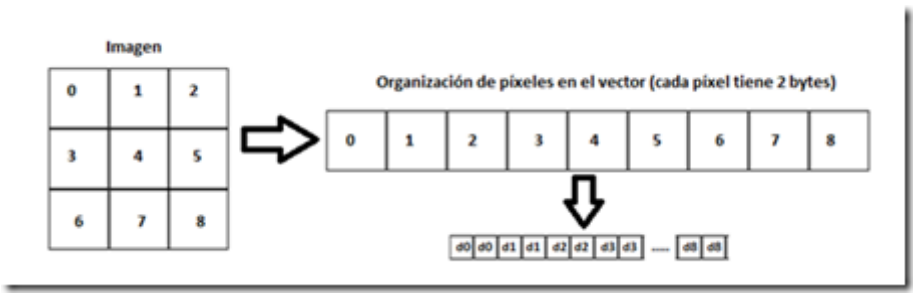


Figura 19.Organización de pixeles en vector

Elaborado por: (Castro, Slin & Jiménez, Fernando)

1.8. Robot Mitsubishi RV-2AJ

El brazo robótico Mitsubishi incorpora otros accesorios (sensores, entradas y salidas digitales) que para el desarrollo del proyecto no se van a emplear, es por esto que no se mencionan, no obstante la estación de robot adquirida por la UPS se puede ver en la figura a continuación.

Figura 20. Estación de robot Mitsubishi RV-2AJ

Fuente: (Festo, s.f.)

El brazo robótico RV-2AJ de Mitsubishi se trata de un robot angular de brazo articulado vertical con 5 GD que se utiliza para el transporte de piezas. El diseño del

RV-2AJ lo hace ideal para aplicaciones donde no sobra el espacio y con movimiento de cargas de hasta 2 Kg de peso (Chamorro, 1998).

Este robot tiene un alcance de 410mm, y combina una velocidad máxima de 2,100mm/s con una precisión de ±0.02mm. Los servomotores de corriente alterna, unidos a codificadores de posición absolutos, garantizan fiabilidad y bajo mantenimiento.

"Estos codificadores permiten apagar el robot en cualquier momento y que al conectar de nuevo la alimentación, el robot podrá continuar trabajando desde la posición actual" (Chamorro, 1998). El brazo tiene integradas en su interior una serie de conductos que permiten la conexión de pinzas y ventosas neumáticas. También tiene integrado un conector para cuatro sensores, y la posibilidad de utilizar pinzas de accionamiento eléctrico. En la tabla se puede ver un resumen de las características del brazo robótico.

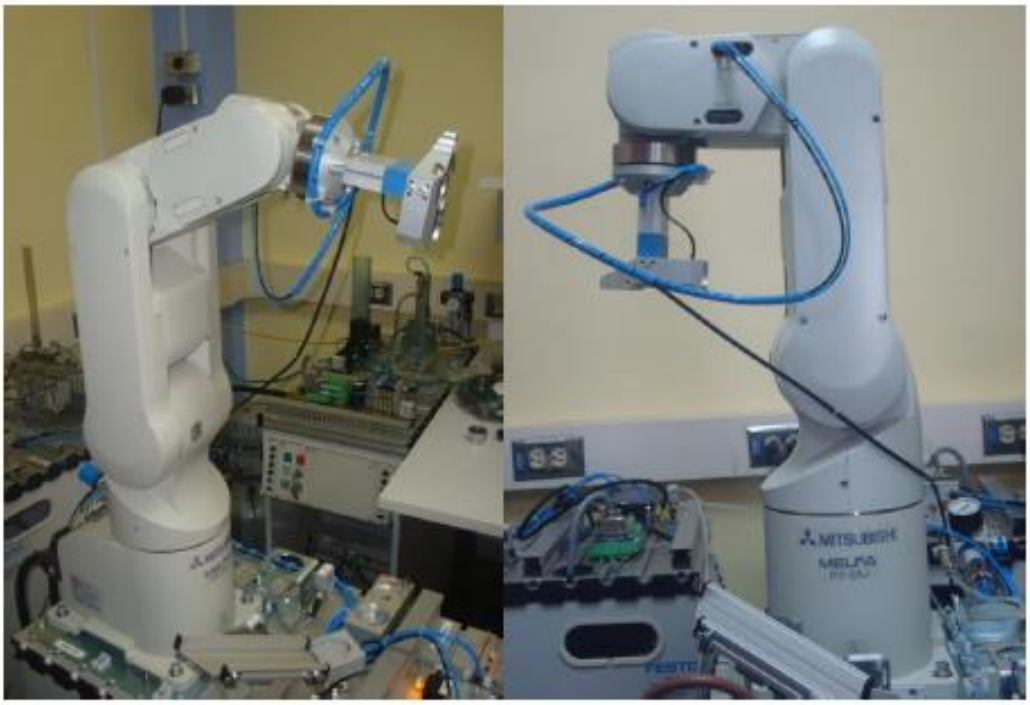


Figura 21. Robot Mitsubishi RV-2AJ.

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Tabla 1. Características del brazo robótico RV-2AJ de Mitsubishi

Brazo Robótico RV-2AJ		
Grados de libertad		5
Motores		Servomotores AC (Ejes J1, J3 y J5 con freno)
Detección de Posición		Codficadores absolutos
Maxima carga (Kg.)		2
Longitud del Brazo (mm.)		250
Alcance radial máximo (mm.)		410
Limite (Grados)	J1	±150
	J2	180(-60 a +120)
	J3	230 (-110 a 120)
	J4	-
	J5	±90
	J6	±200
Velocidad (grados por segundo)	J1	180
	J2	90
	J3	135
	J4	-
	J5	180
	J6	210
Velocidad máxima (mm/s)		2100
Repetitividad (mm.)		±0.0.2
Instalación		Suelo, techo o base FESTO
Controlador del robot		CR1-571

Fuente: (Mitsubishi)

"Este brazo cuenta con un controlador CR1-571 el cual es el cerebro del robot y es donde radica el sistema de control del mismo" (Chamorro, 1998) . El CR1-571 se basa en la misma arquitectura que utilizan los robots de mayores dimensiones de la marca, trabajando con las mismas posibilidades y lo más importante, el mismo lenguaje de programación.

> El corazón del controlador del robot es un CPU - RISC de 64 bits que permite la ejecución en paralelo de hasta 32 programas en modo multitarea. Un aspecto muy importante es el puerto RS-232 que se utiliza para descargar los programas hechos en la computadora y también se utiliza para comunicar el robot con dispositivos externos (Chamorro, 1998).

Otro punto importante del controlador es que permite almacenar los programas en memoria, de esta forma se pueden guardar 88 programas de 5000 líneas cada uno y también se puede guardar 2500 posiciones. En la tabla se puede ver un resumen de las características del controlador.

Figura 22. Controlador CR1-571.

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Adicionalmente el brazo cuenta con una botonera con un display LCD con 4 líneas x16 dígitos (con iluminación de fondo), se conecta al controlador vía RS-422. La botonera o TeachBox se utiliza para determinar y grabar posiciones del brazo del robot para ayudar en la programación, desde borrar un programa o hacerlo nuevo y modificar uno existente. También permite mover manualmente cada articulación del brazo, abrir y cerrar las pinzas entre otras cosas por lo que realmente representa una gran herramienta para la persona que esté programando u operando el robot.

Figura 23. T/B botonera o TeachBox (R2&TB).

Elaborado por: (Castro, Slin & Jiménez, Fernando)

1.9. Objetivos en la mesa y posiciones peligrosas

Una vez que se conocía la posición y orientación de la pinza se procedió a hacer restricciones de movimiento de acuerdo a su ubicación. Dichas restricciones consistieron en que la pinza no fuera a golpear ningún objeto de la mesa ni a sí mismo lo que corresponde a posiciones peligrosas (ver análisis de resultados).

Cabe destacar que el controlador del brazo robótico permite elegir entre tres métodos de posicionado para alcanzar el lugar deseado y son los siguientes:

1. XYZ Jog: En este modo el brazo robótico se moverá respecto al eje de coordenadas situado en la base.

2. JointJog: En este modo se puede mover individualmente cada eje del motor en formato polar (grados).

3. XYZ Tool: En este modo el brazo robótico se moverá respeto al eje de coordenadas situado en el centro la herramienta (En este caso la pinza).

Para el proyecto el brazo opera en el modo JointJog, es decir mueve cada eje independientemente a través de la interfaz háptica alternando en un momento entre J1, J2, J3 y J5, J6.

1.10. Comunicación serial RS-232

La comunicación serial es un protocolo muy común para comunicación entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora. RS-232 (Recommended Standard 232) es una interfaz que designa una norma para el intercambio de una serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE (Equipo de Comunicación de datos), aunque existen otras en las que también se utiliza la interfaz RS-232. Cuando se transmite información a través de una línea serie es necesario utilizar un sistema de codificación que permita resolver los siguientes problemas:

- **Sincronización de bits:** El receptor necesita saber dónde comienza y donde termina cada bit de la señal recibida para efectuar el muestreo de la misma en el centro del intervalo de cada símbolo.

- **Sincronización del carácter:** La información serie se transmite por definición bit a bit, pero la misma tiene sentido en bytes.

- **Sincronización del mensaje:** Es necesario conocer el inicio y fin de una cadena de caracteres por parte del receptor para, para poder detectar algún error en la comunicación de un mensaje por ejemplo,

En una comunicación en general, se pueden establecer canales para la comunicación de acuerdo a tres técnicas:

- **Simplex:** La comunicación serie usa una dirección y una línea de comunicación. Siempre existirá un transmisor y un receptor, no ambos.

- **Half dúplex:** La comunicación serie se establece a través de una sola línea, pero en ambos sentidos. En un momento el transmisor enviará información y en otro recibirá, por lo que no se puede transferir información en ambos sentidos de forma simultánea.

- **Full dúplex:** Se utilizan dos líneas (una transmisora y otra receptora) y se transfiere información en ambos sentidos. La ventaja de este método es que se puede transmitir y recibir información de manera simultánea.

En cuanto a modos de trasmisión existen dos modos básicos para realizar la transmisión de datos y son:

- Transmisión asíncrona
- Transmisión síncrona

Comunicación asíncrona

Cuando se opera en modo asíncrono no existe una línea de reloj común que establezca la duración de un bit y el carácter puede ser enviado en cualquier momento. Esto implica que cada dispositivo tenga su propio reloj y que previamente se haya acordado que ambos dispositivos transmitirán datos a la misma velocidad (Tipantuña, 2003).

En la transmisión asíncrona un carácter a transmitir es incluido con un indicador de inicio y fin de carácter, lo que se conoce como bit de inicio y un bit de parada. Durante el lapso en que no se transmite ningún byte, el canal debe estar en alto (1 lógico), de la misma forma al bit de parada se le asigna también un "1". Por otro lado, al bit de inicio del carácter a transmitir se le asigna un "0" lógico, por lo que un cambio de nivel de "1" a "0" lógico indicará al receptor que un nuevo carácter será transmitido.

La transmisión asíncrona definida por la norma RS-232, se basa en las siguientes reglas:

- Cuando no se envían datos por la línea, ésta se mantiene en estado alto.
- Cuando se desea transmitir un carácter, se envía primero un bit de inicio que pone la línea en bajo durante el tiempo de un bit.
- Luego se envían todos los bits del mensaje a transmitir con los intervalos que marca el reloj de transmisión. Por convenio se transmiten entre 5 y 8 bits.
- Se envía primero el bit menos significativo y de último el bit más significativo.

- Con el último bit del mensaje se envía el bit (o los bits) del final hace que la línea se ponga a 1 por lo menos durante el tiempo mínimo de un bit. Estos bits pueden ser un bit de paridad para detectar errores y el bit o bits de stop, que indican el fin de la transmisión de un carácter.

Los datos codificados por esta regla, pueden ser recibidos de acuerdo a los siguientes pasos:

- Esperar la transición 1 a 0 en la señal recibida.
- Activar el reloj con una frecuencia igual a la del transmisor.
- Muestrear la señal recibida al ritmo de ese reloj para formar el mensaje.
- Leer un bit más de la línea y comprobar si es 1 para confirmar que no ha habido error en la sincronización.

Comunicación síncrona

Es un método más eficiente de comunicación en cuanto a velocidad de transmisión, esto porque no existe ningún tipo de información adicional entre los caracteres a ser transmitidos. Cuando se transmite de manera síncrona lo primero que se envía es un octeto de sincronismo, el cual realiza la misma función que el bit de inicio en la transmisión asíncrona, indicar al receptor que se va enviar un mensaje. Este carácter utiliza la señal local de reloj para determinar cuándo y con qué frecuencia será muestreada la señal, es decir, permite sincronizar los relojes de los dispositivos transmisor y receptor.

CAPÍTULO 2

ANÁLISIS Y DISEÑO

2.1. Análisis de viabilidad

Para realizar este análisis se ha tomado en cuenta dos aspectos importantes dentro del desarrollo de un proyecto, como son: viabilidad técnica y económica.

2.1.1. Viabilidad técnica.

El proyecto va a ser desarrollado utilizando tres tipos de tecnología de Microsoft, como son WCF, Windows Forms y WPF, ademas de un motor de base de datos como lo es SQL Server 2008, el sistema operativo en el cual funcionará será Windows en sus versiones de XP , 7 u 8 , adicionalmente se necesitará de un servidor con Windows Server 2008 .

El hardware que se tiene a disposicion para el desarrollo del mismo es :

- Brazo Mitsubishi RV2AJ

  Funciones: Ejecución de movimientos

- Microsoft Kineckt

  Funciones: Captura y preprocesamiento de imágenes

- Computador portátil DELL XPS 15z

  Procesador: Intel CORE i7
  Memoria: 6 GB en RAM
  Disco: 750 GB
  sistema: Windows 7
  Arquitectura: 64 bits
  Funciones: desarrollo, pruebas, recepción y envío de movimientos

- Computador portátil DELL e6220

  Procesador: Intel CORE i7
  Memoria: 8 GB en RAM

Disco: 250 GB SSD

sistema: Windows 8

Arquitectura: 64 bits

Funciones: desarrollo, pruebas y base de datos

2.1.2. Viabilidad económica.

Se ha estimado el costo del proyecto partiendo del salario de un desarrollador por 8 horas diarias fijando el salario de este en 1000 mensuales, se necesitan 2 desarrolladores por 3 meses teniendo una inversión total detallada a continuación.

Tabla 2. Tabla de viabilidad económica

Detalle	Cantidad	Valor
Salario mensual desarrollador	3	$3.000,00
Dispositivo Kinect	1	$ 350,00
Gastos operativos	1	$ 300,00
Total por tres meses		$ 3.650,00

Elaborado por: (Castro, Slin & Jiménez, Fernando)

La totalidad de los gastos será cubierta por los tesistas, tomando en cuenta el análisis correspondiente se puede determinar que la elaboración y puesta en práctica del proyecto es totalmente viable y factible de implementación.

Se optó por el desarrollo del proyecto debido a que al finalizar la implementación se lograra un sistema que permita un aprendizaje eficaz en el uso del brazo robótico Mitsubishi RV-2AJ, además que en caso de utilizarlo en la industria reduciría los gastos de producción y operación.

2.2. Diseño

2.2.1. Casos de uso.

Los diagramas expuestos a continuacion identifican los módulos que conforman el sistema conjuntamente con los actores que intervienen. Cada diagrama detallará un componentente y permitirá dividir al sistema para su desarrollo agil.

2.2.1.1. Caso de uso #1: Servicio web.

Este caso de uso representa el funcionamiento del servicio web teniendo como actores a las dos aplicaciones que funcionan como clientes y realizan operaciones para la inserción de datos.

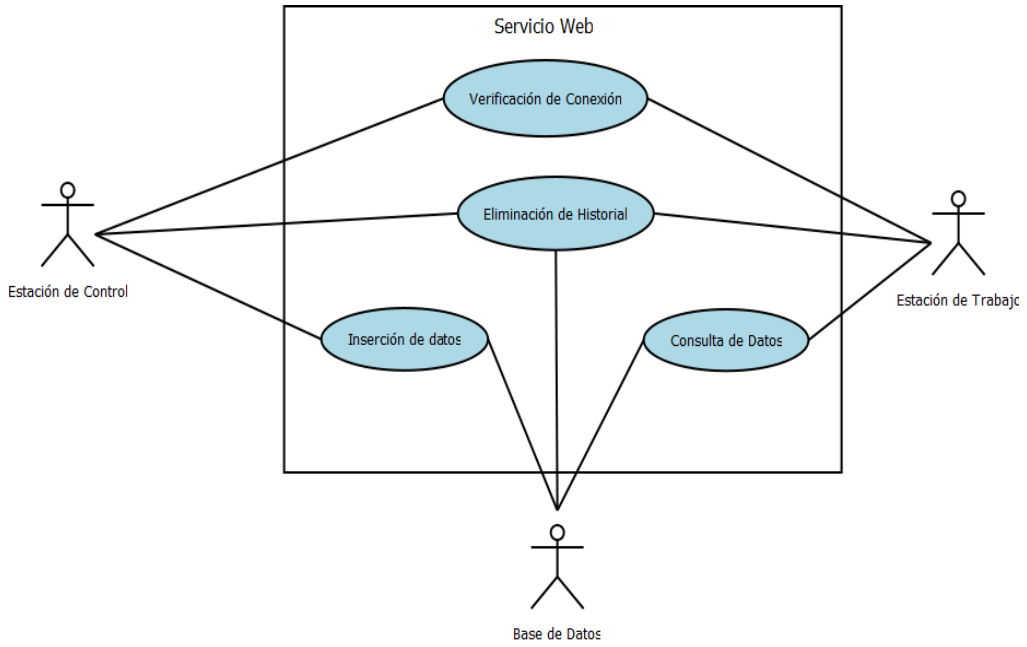


Figura 24. Diagrama caso de uso servicio web

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.1.2. Caso de uso #2: Estación de control.

Este caso de uso representa el funcionamiento de la estación de control teniendo como actores al periférico Kinect y el usuario, indica las operaciones de configuración y envío de datos.

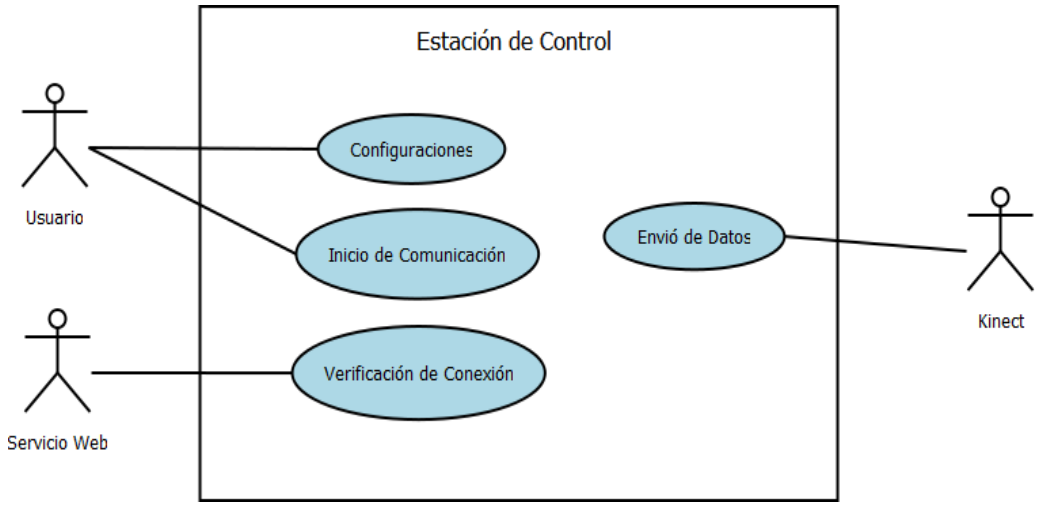


Figura 25. Diagrama caso de uso estación de control

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.1.3. Caso de uso #3: Estación de trabajo.

Este caso de uso representa el funcionamiento de la estación de trabajo teniendo como actores al brazo robótico y el usuario además indica las operaciones de configuración y envío de datos.

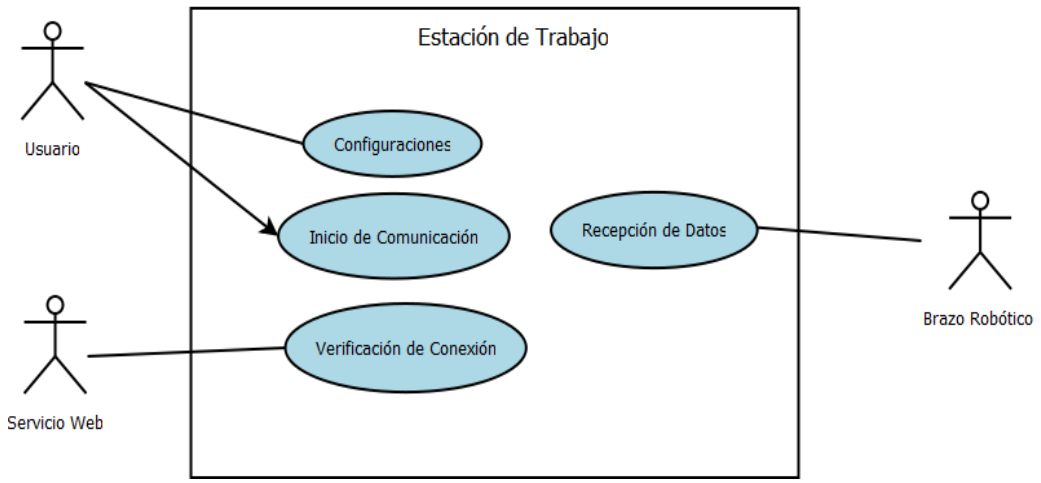


Figura 26. Diagrama caso de uso estación de trabajo

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.2. Diagrama de bloques general.

De acuerdo a los casos de uso planteados anteriormente se definió un diagrama de bloques con tres módulos los mismos que se encuentran subdivididos dependiendo de la funcionalidad de cada uno.

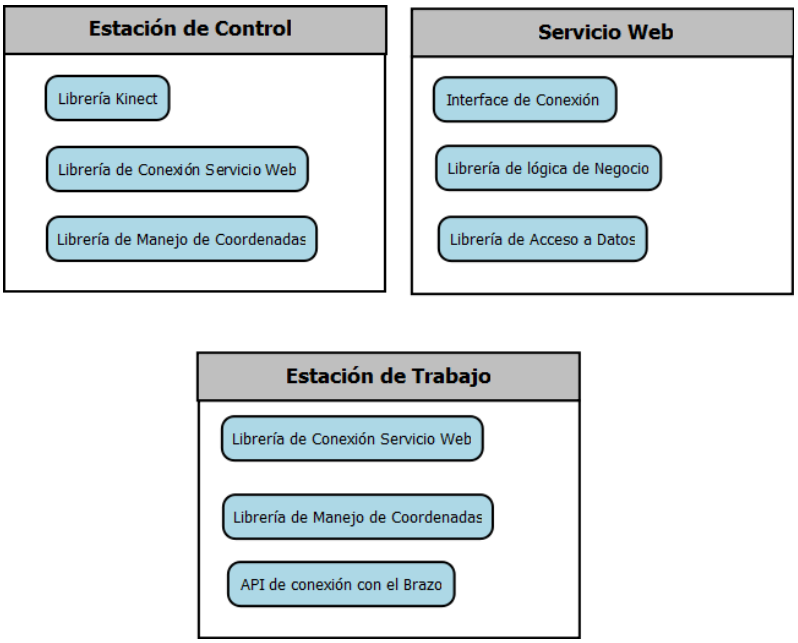


Figura 27. Diagrama de bloques del sistema

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.3. Base de datos.

La base de datos ha sido diseñada a partir del análisis de los requerimientos y funcionalidades del sistema obteniendo así el siguiente diagrama de base de datos.

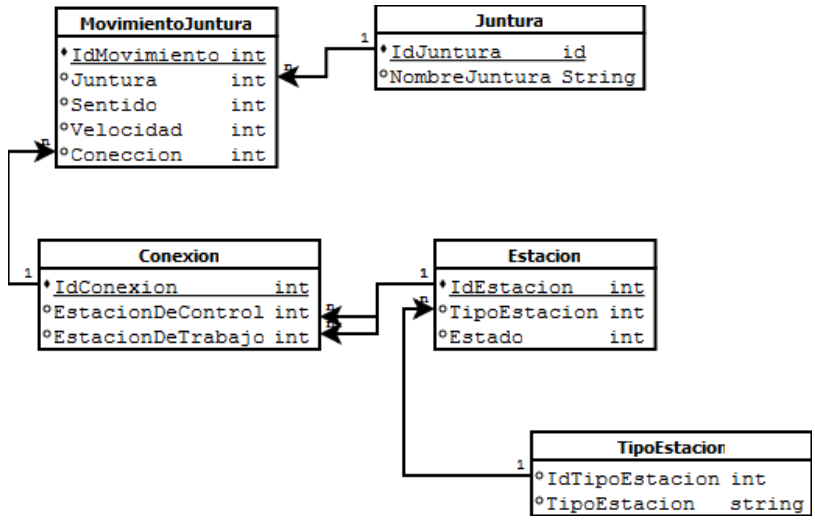


Figura 28. Modelo de la BDD

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Tabla 3. Descripción de BDD

Tabla	Descripción
MovimientoJuntura	En esta tabla se registran los movimientos con respecto a una estación
Juntura	En esta tabla se enlistan las Junturas del brazo
Conexión	En esta tabla se registra la estación de control y la estación de trabajo a la que está asociada
Estación	En esta tabla se enlistan las estaciones de trabajo y control
TipoEstacion	En esta tabla se encuentran los tipos de estaciones

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Se ha decidido usar los nombres en español para la mejor comprensión del negocio por parte de nuevos desarrolladores o interesados en usar este proyecto, además se decidió usar el estilo CamelCase para la definición de tablas y atributos.

2.2.4. Interfaces de usuario.

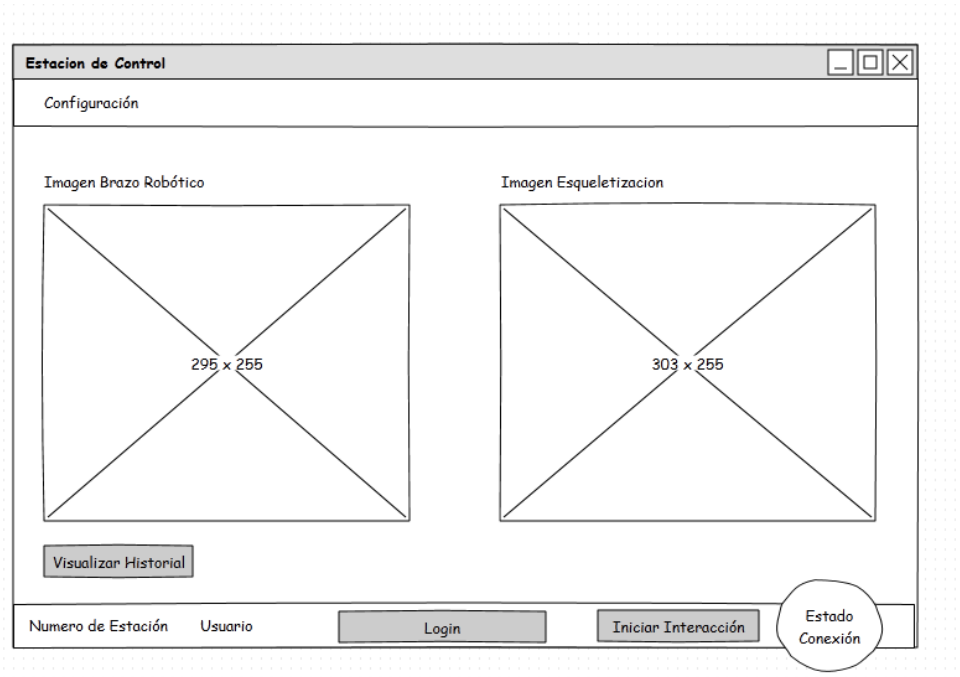


Figura 29. Estacion de control

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Tal como se puede apreciar la Interface estará compuesta por un botón de configuración el que podrá definir la Workstation a la que estará ligada, la imagen del brazo robótico será una representación de las posiciones del brazo robótico, la imagen de la esqueletización provendrá del Kinect y representara las posiciones que toma la pantalla, el botón de visualizar historial permite consultar todas las posiciones tomadas por el brazo , el botón de Login servirá para autenticarse en el servicio web, el botón de iniciar interacción le indica al servicio y al brazo que la comunicación ha empezado.

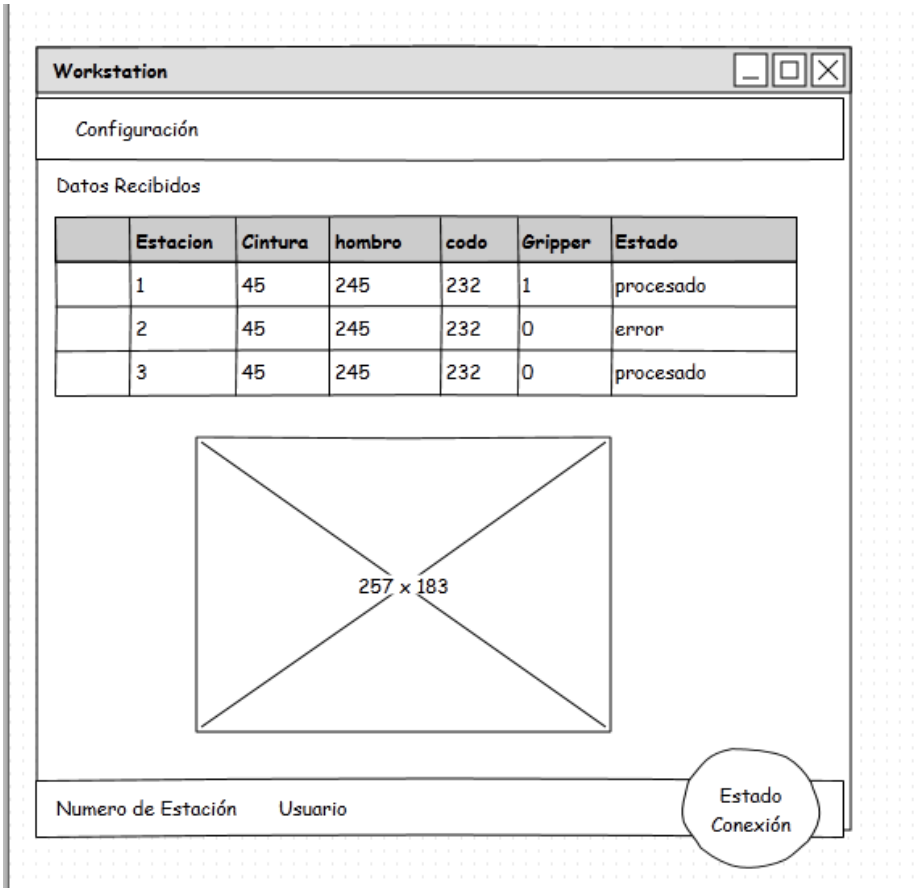


Figura 30. Workstation

Elaborado por: (Castro, Slin & Jiménez, Fernando)

A pesar de que la Workstation solo es un API de comunicación también tendrá una interfaz gráfica simple en la cual se podrá visualizar los movimientos recibidos y si estos fueron procesados o no además de una representación de las coordenadas del brazo, en la parte inferior se podrá visualizar el número de estación y el usuario conectado.

2.2.5. Diagramas de secuencia.

A continuación se detallará el funcionamiento del sistema a partir de los diagramas de secuencia los mismos que se encuentran divididos en tres partes principales, la verificación de la conexión, la inserción de nuevas estaciones y la interacción del Kinect con los movimientos.

2.2.5.1. Obtención de estaciones.

El cliente estación de control consume un método del servicio web y este a su vez consultara a la base de datos y se devolverá las estaciones de control disponibles para la conexión.

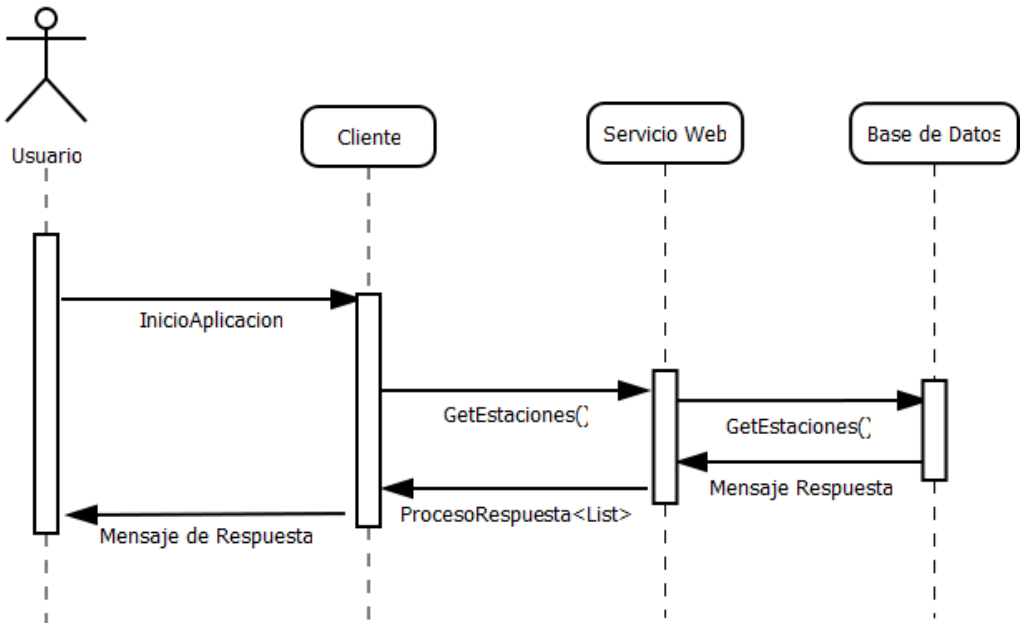


Figura 31. Diagrama de secuencia obtención de estaciones

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.5.2. Verificación de conexión

El cliente seleccionara en la estación de control una estación de trabajo para el envío de datos y verificara si esta se encuentra disponible mediante el consumo de un método en el servicio web.

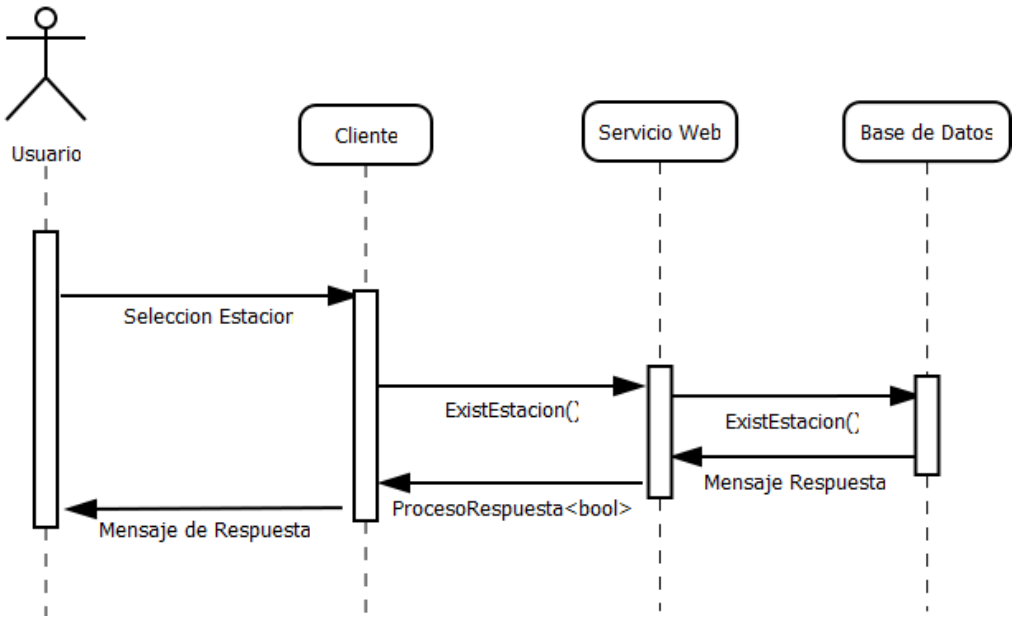


Figura 32. Diagrama de secuencia verificación de conexión

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.5.3. Inserción de estación

El cliente enviará en la estación de control o de trabajo petición para la inserción de una nueva mediante el consumo de un método en el servicio web.

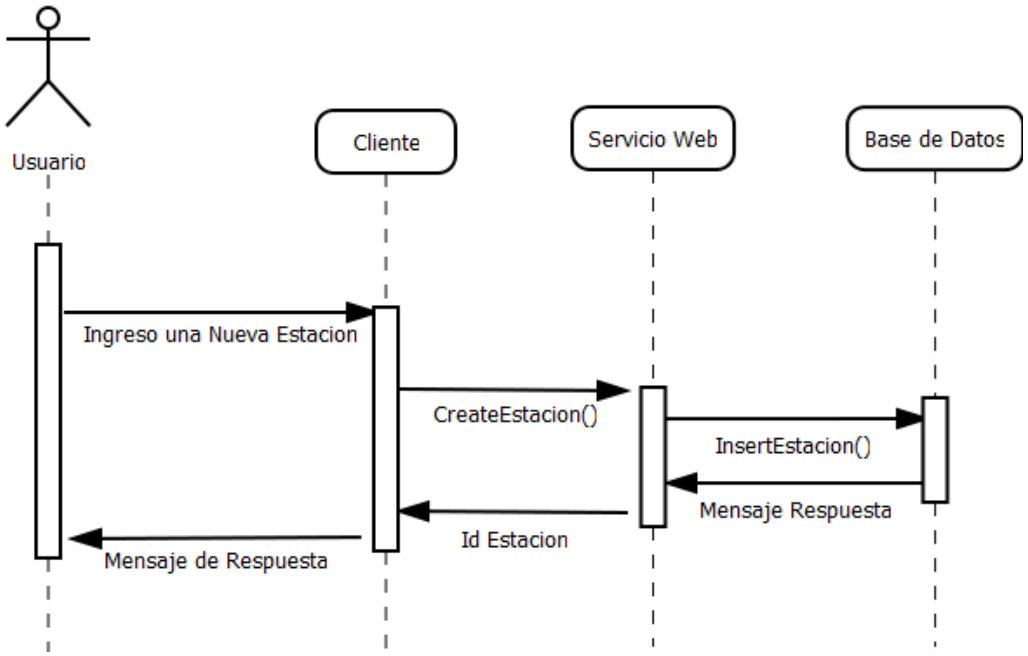


Figura 33. Diagrama de secuencia inserción de estación

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.5.4. Cambio de estado de una estación.

El usuario enviará desde la estación de control o de trabajo petición para el cambio de estado en el servicio web, siendo así al iniciar la aplicación el estado pasara a verdadero y al cerrarla pasara a falso.

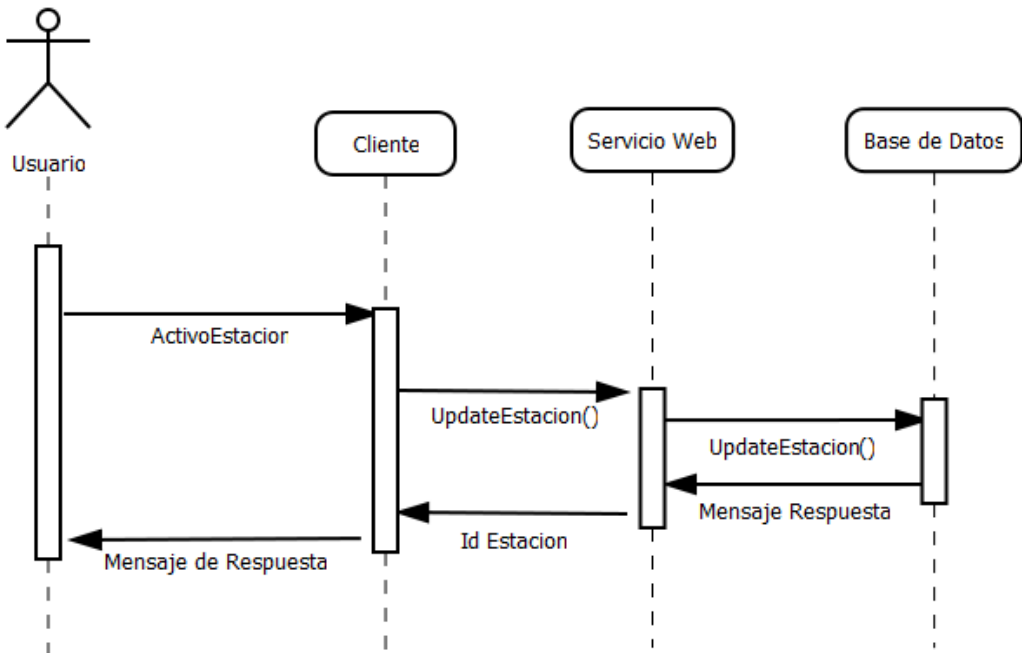


Figura 34. Diagrama de secuencia cambio de estado de una estación

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.5.5. Ingreso de un movimiento.

El usuario realizará un movimiento y será capturado por el periférico Kinect este pasará a la estación de control y a su vez esta consumirá un método en el servicio web en el que almacenará la juntura, el movimiento, la velocidad y la estación que envía el dato, este funcionalidad podrá ser consumida de manera síncrona o asíncrona dependiendo de los requerimientos del cliente.

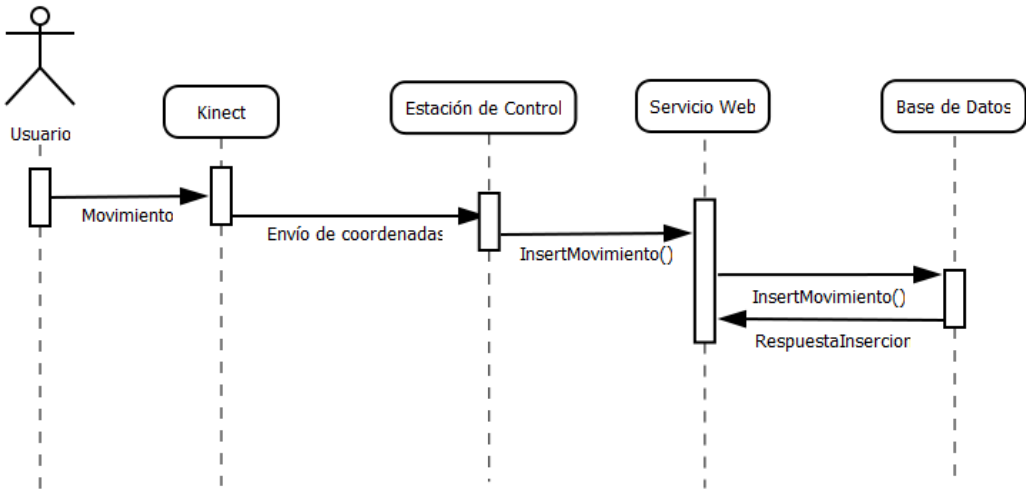


Figura 35. Diagrama de secuencia ingreso de movimiento

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.5.6. Lectura de un movimiento.

La estación de trabajo consumirá un método de consulta del servicio web obtendrá los movimientos por realizar y se los transmitirá al brazo.

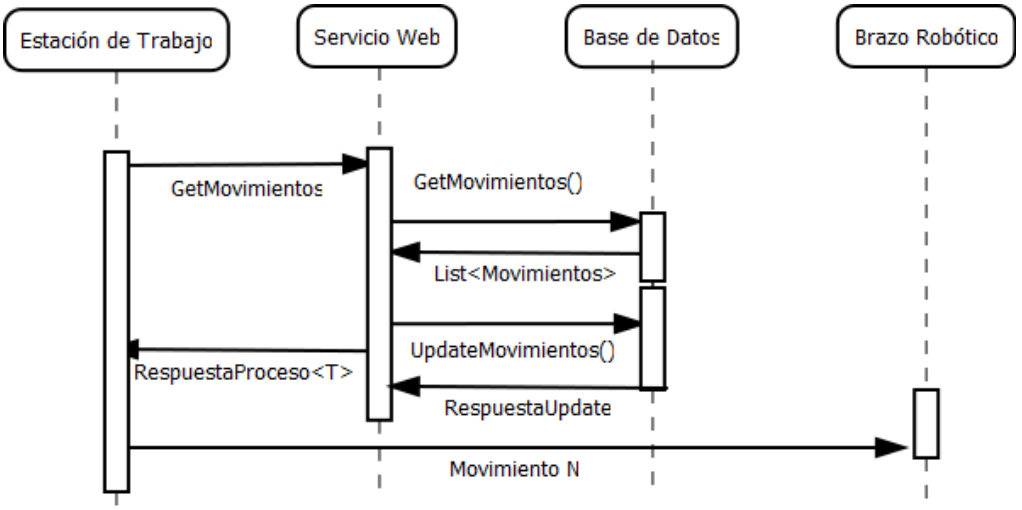


Figura 36. Diagrama de secuencia lectura de movimiento

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.6. Arquitectura del sistema.

El diseño de este aplicativo está basado en tres componentes: la estación de control, el servicio web y la workstation las mismas que se intercomunicarán a través de TCP/IP, para el desarrollo del prototipo las pruebas se las realizará dentro de una intranet conformada por 3 máquinas.

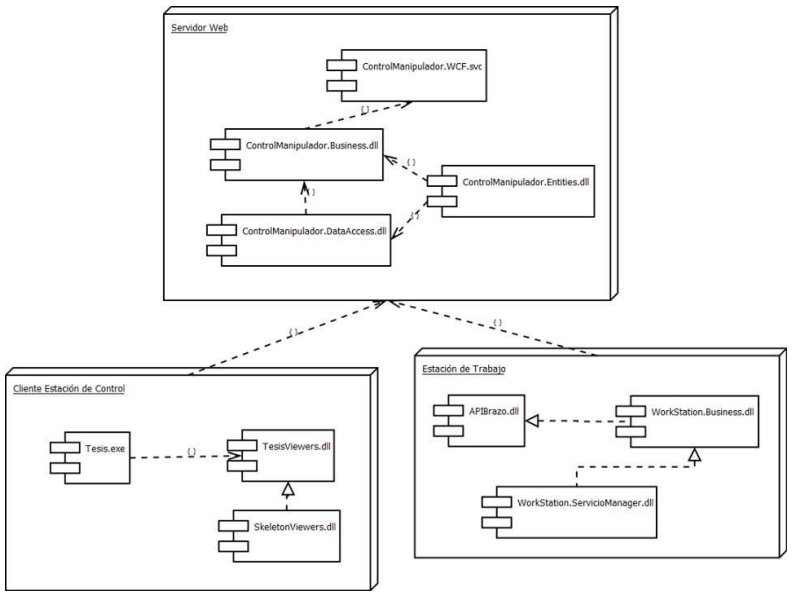


Figura 37. Arquitectura del sistema

Elaborado por: (Castro, Slin & Jiménez, Fernando)

La estación de control será una aplicación basada en el modelo de abstracción MVC (modelo – vista – controlador), esta será desarrollada en un lenguaje de alto nivel con tecnología y estándares SOA la cual interactuará con el periférico Kinect mediante el Framework de este dispositivo y enviará la información al servicio web.

MVC es un patrón en el desarrollo de software en donde se separa los datos de una aplicación en GUI (interfaz gráfica de usuario) y la BLL (capa de lógica de Negocio) en 3 componentes separados.

El modelo consiste en la representación de la información con la que el sistema interactúa, la vista consiste esencialmente en la GUI, y el controlador es el que maneja los eventos que son accionados por el usuario en la GUI.

La ventaja de usar este modelo de abstracción consiste en separar específicamente la GUI de los controladores de esta acopando los eventos accionados en la interfaz a la lógica de negocio trasladando las validaciones de interfaz a una capa inferior y permitiendo el cambio de interfaz gráfica de una manera más limpia fácil y rápida.

El servicio web utilizará la tecnología WCF provista por Microsoft para la intercomunicación entre la estación de control y la workstation, su arquitectura se encuentra definida en el punto anterior, la cual estará conectada a una base de datos (definida en el apartado 2.2.3) en la cual se almacenaran las workstation y estaciones de control además, de los registros de los comandos enviados al brazo robótico.

2.2.6.1. Servicio web.

El servicio web está desarrollado en Visual Studio 2012 utilizando Framework 4.0. Utilizando tecnología WCF.

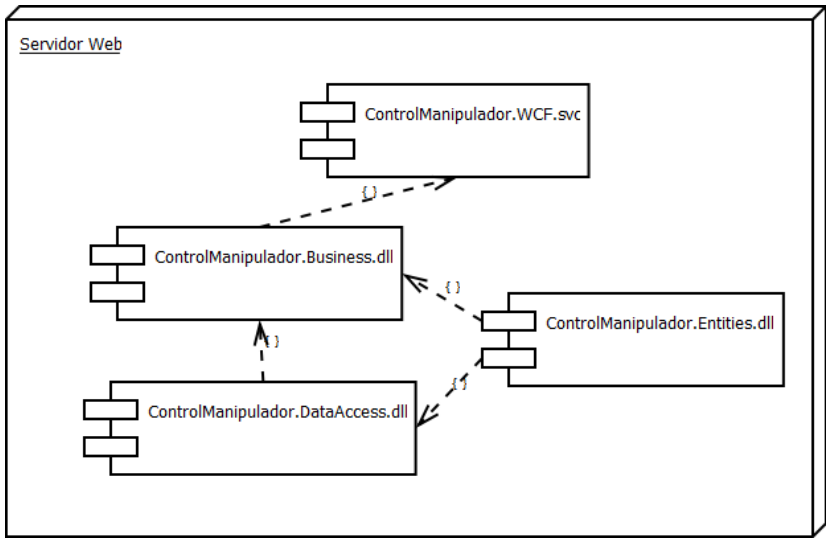


Figura 38. Diagrama de despliegue servicio web

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.6.1.1. ControlManipulador.Business.dll.

Este paquete contiene la lógica de negocio y utilitarios, es el núcleo del funcionamiento del servicio, se encarga de transportar las peticiones de consulta, modificación,

eliminación, además de contener la clase de excepciones que maneja los errores en nivel de lógica de negocio.

2.2.6.1.2. ControlManipulador.dataAccess.dll.

Este paquete contiene la capa de persistencia y los dataContext, se encarga de procesar las peticiones enviadas por la lógica de negocio además de manejar los CRUD de la base de datos.

2.2.6.1.3. ControlManipulador.Entities.dll.

Este paquete contiene las entidades de negocio, los mismos que son una representación lógica de la BDD.

2.2.6.2. Cliente Estación de Control.

El cliente "Tesis" consume las vistas generadas en la librería TesisViewers.dll, para presentar al usuario final el sistema compuesto por el conjunto de vistas configuraciones y parámetros del dispositivo Kinect, a su vez implementa la vista principal encargada de la creación, verificación y conexión del sistema con el dispositivo Kinect.

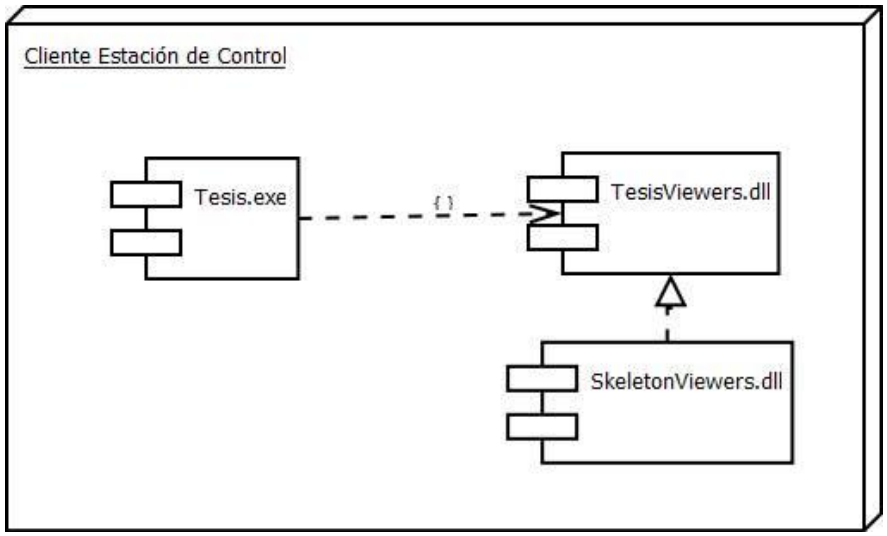


Figura 39. Diagrama estación de control

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.6.3. Cliente estación de trabajo

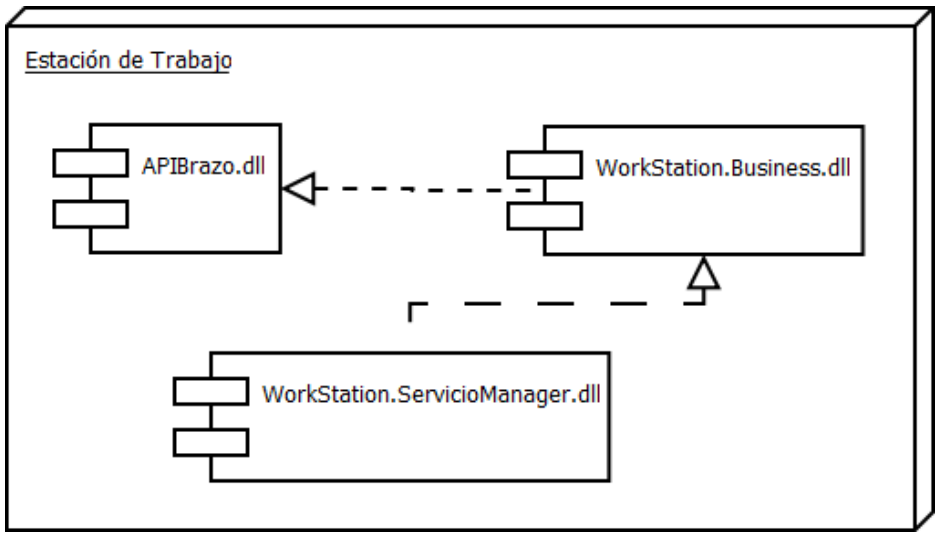


Figura 40. Diagrama cliente estación de trabajo

Elaborado por: (Castro, Slin & Jiménez, Fernando)

2.2.6.3.1. APIbrazo.dll.

Este paquete se encarga de la comunicación serial hacia el brazo robot, fue desarrollado basándose en el trabajo de titulación Diseño y construcción de un API (Interfaz de programación de Aplicaciones) para la plataforma .NET que permita controlar el brazo robótico Mitsubishi RV-2aj a través del Puerto Serial. (Martínez Coral & Freire Muñóz, 2012).

2.2.6.3.2. WorkStation.Business.dll.

Este se encarga de gestionar toda la lógica de negocio del cliente interpretando lo que recibe del servicio y transformándolo para transmitirlo al api y luego al brazo.

2.2.6.3.3. WorkStation.servicioManager.dll.

Se encarga de la gestión de información consumiendo los métodos en del servicio web para la lectura de datos.

CAPÍTULO 3

DESARROLLO

El sistema ha sido desarrollado utilizando el IDE de desarrollo VS2012 aprovechando las ventajas del Framework .net 4.0, utilizando el patrón MVVM para la aplicación cliente, y la tecnología WCF en la implementación de los servicios de consumo de la aplicación, para el almacenamiento de datos se ha optado por utilizar SQL 2012.

3.1. Tecnología

El desarrollo del proyecto ha sido realizado utilizando la arquitectura de capas, implementando la utilización de servicios web para la intercomunicación entre el cliente y la estación de trabajo, para ello se ha realizado el desarrollo con la tecnología .net con el lenguaje C#, el servicio ha sido desarrollado utilizando WCF, y el patrón MMV para el cliente.

3.1.1. Servicio web.

Un servicio web es un tipo de tecnología que utiliza estándares y protocolos para el intercambio de información entre aplicaciones, estas pueden haber sido desarrolladas en diferentes lenguajes de programación, se comunican mediante un canal que puede ser una red LAN, WAN o Internet. Los servicios web fueron creados por la necesidad de implementar un estándar de comunicación que reduzca la dificultad que representa la comunicación entre aplicaciones.

Antes de la creación de los servicios web (WS) se empezó a desarrollar estándares similares aunque estos no tuvieron mucho éxito y difusión principalmente por no comunicar aplicaciones desarrolladas con licencia Microsoft con otras. Se tienen algunos ejemplos de estos estándares, entre los más comunes están DCOM desarrollado por Microsoft y CORBA.

DCOM presenta un gran problema ya que al usar RemoteProcedureCall (RPC) para intercomunicar los diferentes nodos de las aplicaciones representa un serio problema de seguridad y la implementación de este sobre un medio como Internet se hace casi imposible debido a que se encuentra con varios Firewalls que bloquean los mensajes de comunicación.

CORBA tiene algunas ventajas sobre DCOM pero también tiene problemas en la implementación sobre Internet ya que IIOP no es reconocido por algunos firewalls, además de que no existe soporte directo de plataformas Microsoft hacia Corba.

A partir de estos problemas surgen los WS que proveen comunicación estándar entre las aplicaciones Microsoft con otras, en 1999 se planteó el nuevo estándar utilizando XML, SOAP, WSDL y UDDI.

3.1.1.1. Componentes de un servicio web.

Los WS están formados por un conjunto de protocolos y estándares los mismos que se encargan de cumplir diferentes funciones a lo largo de los procesos de trasmisión de datos a pesar de que existen varios protocolos que pueden intervenir se describirán los más importantes y principalmente los que van a estar presentes en el desarrollo de este proyecto. Se puede descomponer los componentes por grupos dentro del WS, se tienen a los encargados de la mensajería y dirección, suministro y descripción de interfaces dentro de este está presente el protocolo de simple acceso a objetos (SOAP), WSDL, WS-Addressing.

3.1.1.2. Protocolo de simple acceso a objetos (SOAP).

Es un protocolo estándar que define la comunicación entre procesos con el intercambio de datos en XML, este protocolo fue creado por Microsoft e IBM entre los más reconocidos y deriva de otro creado por David Winer en 1998 con nombre XML-RPC se encuentra bajo el auspicio de la W3C.

La comunicación entre procesos se lleva a cabo a través de un mensaje llamado mensaje SOAP y está formado de 4 elementos que son:

El elemento Envelope identifica el documento XML como un mensaje SOAP.

El elemento de encabezado es opcional y contiene información específica de la aplicación (como la autenticación, pago, etc.) sobre el mensaje SOAP.

El elemento del cuerpo que contiene información de la llamada y la respuesta es decir los datos que se están intercambiando entre las aplicaciones.

El elemento Fault contiene errores e información de estado funcionando como un log de todos los procesos erróneos dentro del intercambio de mensajes.

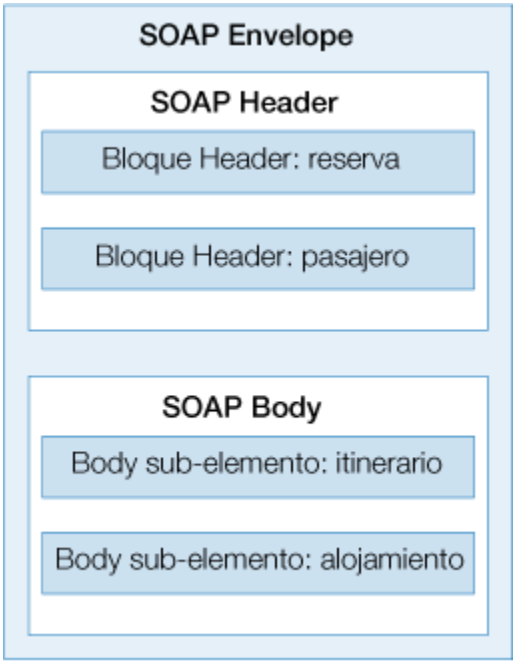


Figura 41. Estructura mensaje soap

Fuente: (The World Wide Web Consortium, s.f.)

3.1.1.3. Leguaje de descripción de servicios (WSDL).

Es un formato XML que se utiliza para describir la interfaz publica de los WS en donde se especifica los requisitos del protocolo los formatos de mensajes necesarios para la interacción de aplicaciones, la operaciones y mensajes que forman parte del WS, siendo así un cliente del WS se conecta al servidor donde se encuentre el servicio y puede mostrar la operaciones disponibles para su uso además contiene la descripción de tipos de datos especiales mediante un archivo .xsd o XML Schema.

3.1.1.4. XML Schema.

El propósito del estándar XML Schema es definir la estructura de los documentos XML que estén asignados a tal esquema y los tipos de datos válidos para cada elemento y atributo. En este sentido las posibilidades de control sobre la estructura y los tipos de datos son muy amplias.

> Al restringir el contenido de los ficheros XML es posible intercambiar información entre aplicaciones con gran seguridad. Disminuye el trabajo de comprobar la estructura de los ficheros y el tipo de los datos, XML Schema tiene un enfoque modular que recuerda a la programación orientada a objetos y que facilita la reutilización de código (UPM, 2004).

Los tipos de datos tienen en XML Schema la función de las clases en la POO. El usuario puede construir tipos de datos a partir de tipos predefinidos, agrupando elementos y atributos de una determinada forma y con mecanismos de extensión parecidos a la herencia. Los tipos de datos se clasifican en función de los elementos y atributos que contienen.

Los tipos de datos en XML Schema pueden ser simples o complejos. XML Schema incluye el uso de namespaces. Los "espacios de nombres" permiten definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se anteponga un prefijo al nombre del elemento. El uso de namespaces también evita confusiones en la reutilización de código.

"Es posible agrupar atributos, haciendo más comprensible el uso de un grupo de aspectos de varios elementos distintos, pero con denominador común, que deben ir juntos en cada uno de estos elementos" (UPM, 2004). Los ficheros XML Schema se escriben en el propio lenguaje XML.

Tipos simples

Tipos simples son aquellos que no tienen ni elementos hijos ni atributos.

Son tipos simples:

- Tipos predefinidos de XML: string, double, boolean, etc.
- List (lista de datos separados por espacios).
- Union (tipo de dato derivado de la unión de tipos predefinidos).

Tipos complejos

Son tipos complejos aquellos que tienen elementos hijos y/o atributos. Pueden tener nombre o ser anónimos. Si tienen nombre pueden ser reutilizados dentro del mismo XML Schema o por otros XML Schemas. Es posible "mezclar" o combinar elementos y texto (UPM, 2004)**.**

3.1.1.5. Web services addresing (WS-Addresing).

Mediante esta especificación se puede identificar los mensajes y WS independientemente del protocolo de transporte que se esté usando, este estándar define un espacio de nombres para la identificación de WS. Gracias a este se puede hacer peticiones pasando por nodos como Firewalls Gateway etc.

También se tiene protocolos de seguridad presentes para el intercambio de información y autenticación de servicios.

"Los protocolos de seguridad para servicios web comienzan por la especificación WS-Security que define una arquitectura basada en señales para comunicaciones seguras" (IBM, s.f.). Existen seis principales especificaciones de componentes construidas sobre esa base como:

- WS-Policy y sus especificaciones relacionadas, que definen las reglas de políticas sobre la interacción de servicios.
- WS-Trust, que define el modelo confiable para el intercambio seguro.
- WS-Privacy, que define cómo se mantiene la privacidad de las informaciones.
- WS-SecureConversation, que define cómo establecer una sesión protegida entre servicios para intercambiar datos usando las reglas definidas en WS-Policy, WS-Trust, y WS-Privacy.
- WS-Federation, que define las reglas de identidad distribuida y de la gestión de esa identidad.
- WS-Authorization, que maneja el procesamiento de autorización para acceder a los datos e intercambiarlos.

Además, del modelo de seguridad existen las especificaciones específicas de las aplicaciones, incluyendo el lenguaje de ejecución de procesos empresariales para servicios web (BPEL4WS), que define las operaciones de flujos de trabajo, y WS-Transaction y WS-Coordination, que funcionan en conjunto para manejar el procesamiento distribuido de transacciones.

3.1.2. C Sharp (C#).

Es un lenguaje de programación orientado a objetos desarrollado por Microsoft específicamente cuando Anders Hejlsberg formó un equipo para el desarrollo de un lenguaje que iba a ser conocido como COOl C orientedobject lenguaje y este fue incluido dentro del IDE .NET con un sistema de librerías utilizando un sistema de código gestionado llamado Simple Managed C (SMC), su lenguaje se deriva de C y C++ utilizando un modelo de orientación a objetos aunque está diseñado para ser compilado dentro del IDE de .NET en el sistema operativo Windows también se ha desarrollado compiladores para distintos S.O. como lo son Unix, IOS, Linux entre los más reconocidos.

3.1.3. Windows comunication foundation (WCF).

Este es un modelo de programación orientado al desarrollo de aplicaciones con arquitectura orientada a servicios basadas en el intercambio de Mensajes SOA, lo conforman un grupo de bibliotecas alojadas dentro del Net Framework 3 ,4 y 4.5 hasta la fecha sobre diferentes canales de transporte, proporciona un modelo estándar de comunicación entre sistemas haciéndolos interoperables con otros que pueden estar desarrollados en diversos lenguajes como lo son J2EE, SAP.

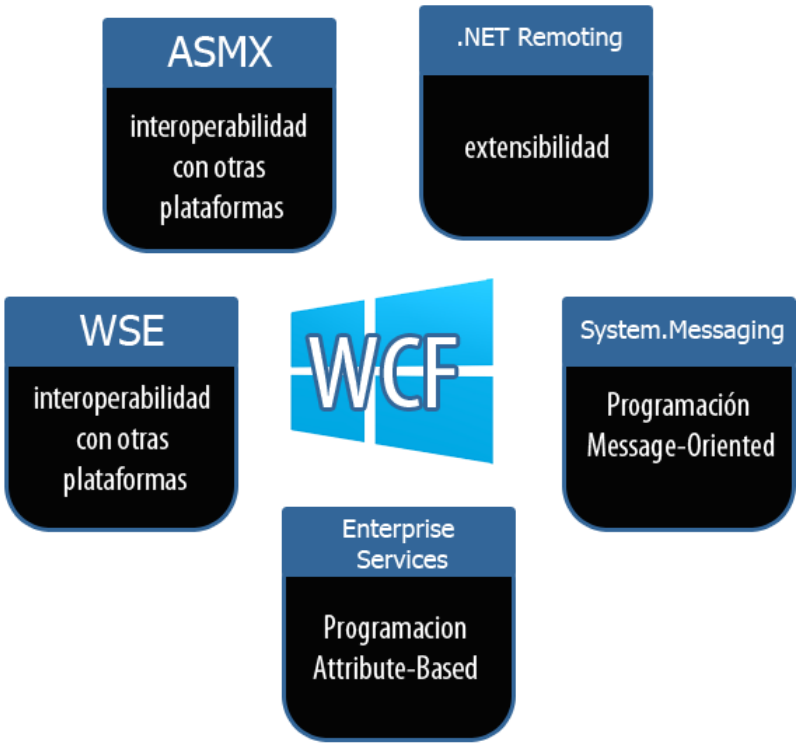


Figura 42. Modelo WCF

Elaborado por: (Castro, Slin & Jiménez, Fernando)

WCF no solo es un modelo en el que se pueden desarrollar servicios sino también clientes. Los clientes son aplicaciones que inician la comunicación con un llamado al servicio, servicio es una aplicación que mientras este levantado, estará escuchando las peticiones de los clientes para iniciar el intercambio de mensajes. Los mensajes son enviados entre endpoints. Un endpoint es un lugar donde un mensaje es enviado, o recibido, o ambos.

Un servicio expone en su alojamiento uno o más aplicación endpoints los cuales pueden ser consumidos desde el cliente (entiéndase por cliente una aplicación que pueda solo consumir los datos para los fines que esta requiera u otro servicio SOA), un cliente genera un endpoint que sea compatible con cualquiera los endpoints de un servicio dado.

La interacción entre un servicio y un cliente compatibles forman un communicationstack.

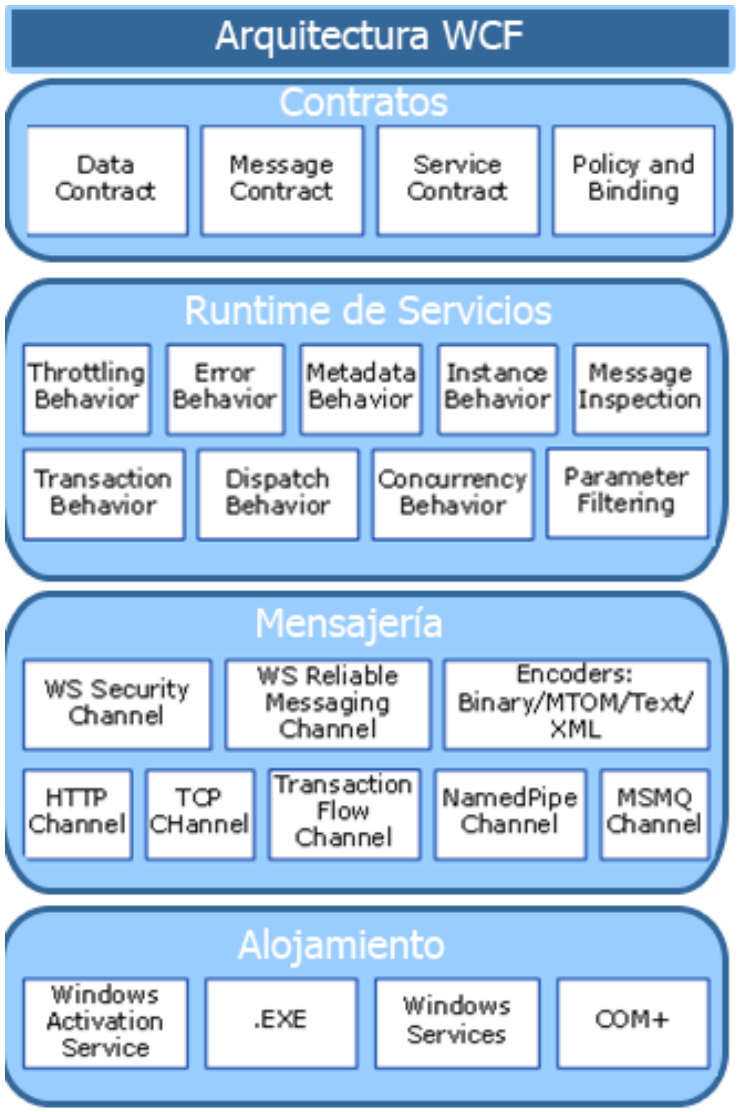


Figura 43. Arquitectura de una Aplicación WCF

Elaborado por: (Castro, Slin & Jiménez, Fernando)

3.1.3.1. Contratos y Descripciones.

Los contratos definen los parámetros que constituyen los servicios en lenguaje XML de tal forma que puedan ser procesados por cualquier sistema que entienda dicho lenguaje, esta descripción está parametrizada por los protocolos SOAP, de esta manera se puede obtener un control especifico del mensaje cuando la interoperabilidad lo

exige, dentro del contrato se enlistan las firmas de los métodos implementados en el servicio.

Las directivas y los enlaces definen las condiciones que exige la comunicación por estos servicios, dentro de este como mínimo se debe citar el tipo de transporte utilizado como por ejemplo si es TCP o HTTP y una codificación, las directivas de este están conformadas por los requisitos de seguridad puntos de acceso entre otros.

3.1.3.2. Tiempo de ejecución de servicio.

Dentro de la capa de tiempo de ejecución del servicio se albergan las reacciones que se producirán durante la operación ya que por ejemplo se lo puede configurar unilateral mente es decir que simplemente haga una operación solicitada sin esperar una reacción del cliente o configurarlo como bidireccional haciendo que el servicio espere una respuesta para proseguir con su siguiente operación. El limitar las peticiones hacia el servicio controla el número de mensajes intercambiados con el cliente durante el tiempo de ejecución limita el número de errores por funcionamiento del servicio además de limitar la cantidad de información que pueda ser obtenida por un usuario mal intencionado, el control de los metadatos dentro del tiempo de ejecución es vital ya que estos supervisan la información que será mostrada al "mundo entero".

3.1.3.3. Mensajería.

La capa de la mensajería es creada mediante canales, estos son componentes que procesan mensajes de acuerdo al modelo de negocio de los métodos contenidos en el servicio, los canales funcionan en los mensajes y sus encabezados, se tiene 2 tipos de canales aquellos encargados del transporte y los canales de protocolo.

Los canales de transporte cumplen la función de leer y escribir mensajes de los puntos de comunicación hacia el "Exterior".

Los canales de protocolo implementan los protocolos dentro del procesamiento de mensajes esta capa de mensajes a menudo añade nuevos metadatos en la cabecera para que el mensaje pueda ser comprendido.

3.1.3.4. Alojamiento y activación.

Consiste en la forma final del servicio pudiendo ser este un ejecutable o un conjunto de librerías que se aloja en un host y activado por un agente externo pudiendo ser este el IIS (Internet InformationService) o WAS (servicio de activación de Windows) también se poseen componentes nativos del sistema operativo como el COM plus.

3.1.4. Net Framework.

Está compuesto por un conjunto de lenguajes de programación para que estos puedan ser interpretados por el sistema operativo, La biblioteca de clases base o BCl y el entorno común de ejecución para lenguajes, o CLR por sus siglas en inglés. Su función principal es proveer al compilador los recursos de lenguaje necesarios para que este cree los ejecutables o librerías desarrolladas sobre el IDE VS .NET.

3.1.5. Biblioteca de clases base de .NET.

La biblioteca de clases base (BCL por sus siglas en inglés) maneja la mayoría de las operaciones básicas que se encuentran involucradas en el desarrollo de aplicaciones, incluyendo entre otras:

- Interacción con los dispositivos periféricos
- Manejo de datos (ADO.NET)
- Administración de memoria
- Cifrado de datos
- Transmisión y recepción de datos por distintos medios (XML, TCP/IP)
- Administración de componentes web que corren tanto en el servidor como en el cliente (ASP.NET)
- Manejo y administración de excepciones
- Manejo del sistema de ventanas
- Herramientas de despliegue de gráficos (GDI+)

- Herramientas de seguridad e integración con la seguridad del sistema operativo
- Manejo de tipos de datos unificado
- Interacción con otras aplicaciones
- Manejo de cadenas de caracteres y expresiones regulares
- Operaciones aritméticas
- Manipulación de fechas, zonas horarias y periodos de tiempo
- Manejo de arreglos de datos y colecciones
- Manipulación de archivos de imágenes
- Aleatoriedad
- Generación de código
- Manejo de idiomas
- Auto descripción de código
- Interacción con el API Win32 o Windows API
- Compilación de código
- Esta funcionalidad se encuentra organizada por medio de espacios de nombres jerárquicos

La biblioteca de clases base se clasifica, en cuatro grupos clave:

- ASP.NET y servicios web XML
- Windows Forms
- ADO.NET
- .NET

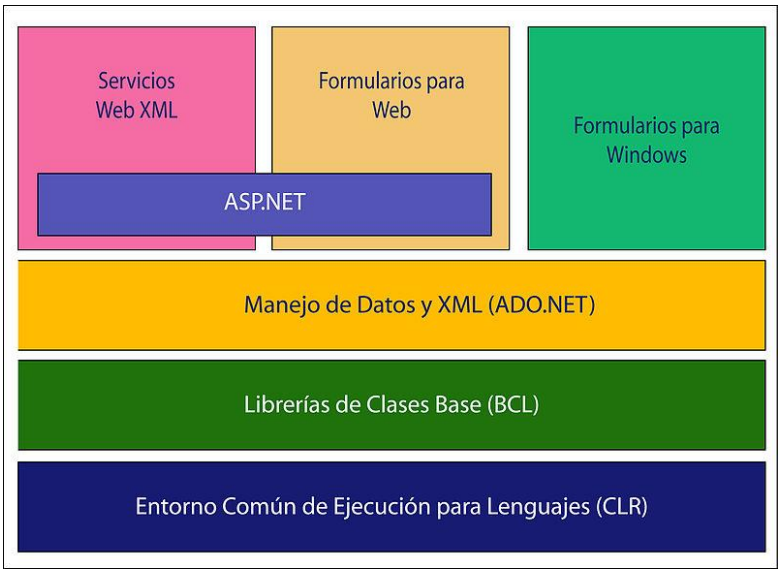


Figura 44. Diagrama básico de la biblioteca de clases base.

Fuente: (Microsoft, 2013)

3.1.6. Patrón modelview view model (MVVM).

El principal objetivo del patrón es proporcionar mecanismos que permitan separar el desarrollo de la interfaz de usuario (View) del desarrollo de la lógica y estado de presentación (encapsulada en el ViewModel) y del modelo de datos (encapsulada en el Model) (Microsoft, 2013).

El resultado es que MVVM facilita y agiliza el desarrollo al mismo tiempo que permite construir aplicaciones desacopladas y testeables.

Los desarrolladores que hayan tenido la oportunidad de realizar aplicaciones WPF o Silverlight (o para Windows Phone o Windows 8) pero que no han usado MVVM, habrán experimentado que en el archivo de código asociado a la vista (denominado code-behind) se suele desarrollar una importante parte del código; esto puede incluir lógica de negocio, lógica de acceso a datos manejadores de eventos etc. Se trata, pues, de un código que se encuentra fuertemente acoplado a la vista y que hace referencia directamente a elementos visuales o controles definidos en el código XAML. Existen numerosas desventajas en el uso de este modelo de desarrollo, por citar algunas, este código no podría ser testeado mediante tests unitarios, ni permitiría desarrollar por

separado la interfaz de usuario. Además, la modificación de cualquier elemento de la UI implicaría necesariamente la modificación del code-behind (alto acoplamiento).

Figura 45. Modelo MMV

Elaborado por: (Castro, Slin & Jiménez, Fernando)

MVVM va a permitir trasladar la mayoría de nuestro código desde el code-behind a una nueva capa de abstracción que se denominaraViewModel. En el ViewModel no habrá referencia alguna a elementos de la vista; sus propiedades, comandos y métodos harán uso del modelo de datos (Model) para quedar listas y expuestas para que sus consumidores (ya sea una vista o un test unitario) se enlacen a ellos mediante el denominado data binding. Vistas y/o tests no son más que consumidores del ViewModel.

3.1.6.1. Data binding.

En efecto, el patrón MVVM gira en torno al concepto del "enlace de datos", o data binding. El concepto de data binding aparece con XAML. El data binding es una conexión entre la interfaz de usuario y un objeto de datos que permite que los datos fluyan entre ellos. Cuando se establece un enlace y los datos cambian, los elementos de la interfaz de usuario enlazados a los datos pueden mostrar los cambios automáticamente. De manera similar, los cambios que realiza el usuario en un elemento de la interfaz de usuario se pueden guardar en el objeto de datos.

Se puede hablar largo y tendido del concepto de data binding, pero de momento solamente se resumirá brevemente la idea que permita poder seguir el hilo de explicación del patrón MVVM. Todos los enlaces incluyen lo siguiente:

- Un origen de enlace, que es un objeto con los datos que se desean representar.

- Un destino de enlace, que es una propiedad del elemento de la UI que se usa para representar los datos.

- Un objeto binding, que mueve los datos entre el origen y el destino y que puede cambiar su formato a través de un convertidor de valores opcional.

Por ejemplo, el data binding haría referencia al enlace de la propiedad "Text" de un elemento visual de tipo TextBox a la propiedad "Apellido" de un hipotético objeto de tipo "Persona".

En el código XAML que describe la vista se tendrá:

```
<TextBox: Name="SurnameTextBlock" Text="{binding Apellido}"/>
```

Mientras que la propiedad origen de datos podría ser tal y como está (no preocuparse ahora si no se entiende todo el código, se explicará en el apartado siguiente):

```
privatestring_apellido = string.Empty;
publicstringApellido
{
    get
    {
        return_apellido;
    }
    set
    {
        if(_apellido == value)
        {
            return;
        }
    _apellido = value;
    RaisePropertyChanged(Apellido);
    }
}
```

Figura 46. Declaración de propiedad

Elaborado por: (Castro, Slin & Jiménez, Fernando)

En este escenario, la vista sabe que debe representar en un TextBox el valor contenido en una propiedad llamada Apellido. En la clase Persona existe una propiedad llamada Apellido, pero ¿Cómo se puede enlazar ambas? El elemento que permite se denomina dataContext (contexto de datos). Este contexto de datos puede establecerse a cualquier nivel de la vista (la página completa, un elemento en particular, un grupo de elementos.). Por tanto, definiendo en la vista que su dataContext es la clase Persona se habrá cerrado el círculo. El dataContext hay que definirlo siempre en la vista:

```
publicMainPage(){
this.InitializeComponent();
this.dataContext = newPersona();
}
```

En MVVM, por todo lo que se ha comentado, será establecido que el dataContext de una vista será su correspondiente ViewModel. Aunque hay quien pueda discrepar de ello, es aconsejable tener un sólo ViewModel para cada vista.

Los enlaces pueden tener varias direcciones:

- Los enlaces OneTime actualizan el destino con los datos del origen cuando se crea el enlace.
- Los enlaces OneWay actualizan el destino con los datos de origen cuando se crea el enlace y cada vez que cambian los datos. Este es el modo predeterminado.
- Los enlaces TwoWay actualizan tanto el destino como el origen cuando alguno de ellos cambia.

Dejando que el sistema de data binding se encargue de la sincronización entre estas propiedades, se podrá desarrollar nuestra lógica sin tenernos que preocupar de la actualización de la interfaz de usuario.

3.1.6.2. INotifyPropertyChanged.

Para que funcione la sincronización proporcionada por el mecanismo de data binding se tiene que dotar a las propiedades de capacidad para notificar sus cambios. Esto se podrá conseguir si nuestro objeto origen implementa la interfaz INotifyPropertyChanged (definida en System.ComponentModel).Se concluye entonces, que nuestros objetos son "observables".

En MVVM, como ya se ha dicho, la vista se enlaza a su correspondiente ViewModel. Por tanto, la clase ViewModel deberá implementar la INotifyPropertyChanged. Como lo habitual será tener varias vistas y varios ViewModel (y como no se desea implementar esta interfaz en cada uno de los ViewModel) lo más habitual será que se desarrolle una clase base en la que se implemente la interfaz (ViewModelBase) y que todos los ViewModel hereden de esta clase base.

Se retomará ahora el código de la propiedad Apellido que ya fue presentado en el apartado anterior donde está descrita en una clase que implementa la interfaz INotifyPropertyChanged y que permite usar un método llamado RaisePropertyChanged para propagar las modificaciones en el valor de la propiedad indicando. De este modo, todos aquellos elementos de la UI que tengan un binding a esa propiedad actualizarán también su valor.

3.1.6.3. Commandbinding.

Hasta ahora se abordó que gracias al data binding y a la existencia de objetos observables se puede asociar propiedades de elementos de la UI con propiedades de objetos de datos. Pero las aplicaciones basadas en XAML no sólo ofrecen un mecanismo de data binding sino que también se dispone del denominado **Commandbinding**. Gracias al Commandbinding se tiene la capacidad de enlazar eventos de la interfaz de usuario con objetos en los que se describirán las acciones a realizar.

Para hacer uso del Commandbinding se necesita crear objetos que implementen la interfaz ICommand (situada en System.Windows.Input). A estos objetos se les llamara simplemente **Commands**. Esta interfaz define un método **Execute** donde se ubica el código que debe ejecutarse cuando el comando sea invocado (que vendría a ser el código que originalmente está situado en los manejadores de eventos del code-behid). También define el método **CanExecute** que retorna un booleano en el que se determina si el comando puede ser ejecutado o no. Ambos métodos pueden tomar un sólo parámetro.

Existen varios ejemplos en la red para implementar ICommand, pero por lo general en los proyectos no se realiza este trabajo, sino que se adicionara alguna de las implementaciones ya existentes, como ActionCommand (ExpressionBlend SDK), DelegateCommand (Prism) o, el que se utilizará en esta serie RelayCommand (de MVVM Light).

A continuación se muestra un ejemplo con RelayCommand. En la clase que sirve de dataContext de la vista se definiría el objeto RelayCommand y los métodos Execute y CanExecute:

```
privateRelayCommand _goHomeCommand;

publicRelayCommandGoHomeCommand
{
    get
    {
        return_goHomeCommand ?? (_goHomeCommand = newRelayCommand(
        ExecuteGoHomeCommand,
        CanExecuteGoHomeCommand));
    }
}

privatevoidExecuteGoHomeCommand()
{
    Navigate(typeof(HomeView));
}

privateboolCanExecuteGoHomeCommand()
{
    returntrue;
}
```

Figura 47. Declaración de comando

Elaborado por: (Castro, Slin & Jiménez, Fernando)

A la hora de invocar un ICommand desde la vista se podrá apreciar varios casos. Existen elementos de la UI (como los botones) que ya de por sí tienen una propiedad llamada Command, en este caso sería tan simple como:

```
<ButtonCommand="{bindingGoHomeCommand}"/>
```

Observar el paralelismo que existe entre data binding / Propiedades, y data Commanding / Commands.

Pero ¿qué pasa con aquellos elementos de la UI que no disponen de una propiedad Command? Aunque no es nativo de XAML, existen clases que nos permiten dirigir eventos a comandos declarativamente (en XAML). A esto suele llamarse **AttachedBehaviours**.Se puede encontrar un ejemplo en la biblioteca System.Windows.Interactivity.dll.

```
    xmlns:i="clr-
    namespace:System.Windows.Interactivity;assembly=System.Windows.Interactivity"
```

Haciendo uso de ella se pueden definir disparadores de eventos:

```
    <TextBlockText="{binding name}"FontSize="18">
        <i:Interaction.Triggers>
        <i:EventTriggerEventName="MouseLeftButtonDown">
<!-- Código a ejecutar al producirse el evento -->
        </i:EventTrigger>
         </i:Interaction.Triggers>
     </TextBlock>
```

En el código anterior únicamente nos faltaría definir la acción a ejecutar. Esta acción puede definirse a través de un objeto de tipo TriggerAction. El paquete al MVVM Light nos proporciona una clase denominada **EventToCommand** que nos permite ejecutar la acción **Execute** de un**ICommand** cuando acontezca el evento definido. Siendo:

```
 xmlns:cmd="clr
 namespace:GalaSoft.MvvmLight.Command;assembly=GalaSoft.MvvmLight.Extras.WP8"
```

El código anterior como quedaría como sigue:

```
<TextBlockText="{binding name}"FontSize="18">
<i:Interaction.Triggers>
<i:EventTriggerEventName="MouseLeftButtonDown">
<cmd:EventToCommandCommand="{binding MyICommand}"/>
</i:EventTrigger>
</i:Interaction.Triggers>
</TextBlock>
```

Figura 48. Declaración de trigger de eventos

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Pues, MVVM también se aprovecha del mecanismo de Commandbinding para permitir la lógica de interacción con el usuario se desarrolle en el ViewModel de forma totalmente desacoplada de la vista.

3.1.6.4. Model.

Hasta ahora se ha abordado el papel de la vista y del ViewModel y de los mecanismos de comunicación entre ellos que proporcionan el data binding y el data commanding. Se pasara a analizar el tercer elemento del patrón MVVM: El Model.

El Model encapsula toda la lógica de acceso a datos. Esto no sólo incluye la lógica para gestionar una conexión a datos (vía ADO.NET, EF, WCF, Odata, etc), sino que también incluye toda aquella lógica necesaria para validar y preparar los datos que van a ser consumidos por los ViewModel. A veces la frontera entre lo que debe hacerse en

el Model o en el ViewModel acerca de la gestión de los datos es algo confusa y motivo de debate.

En todo caso las clases que implementan el Model no deben referenciar ni tener dependencias con la vista ni a los ViewModel. Su único fin es proporcionar métodos que puedan ser consumidos por los ViewModel para obtener datos validados y bien formateados, reportando cualquier error que haya podido acontecer.

Este esquema resume esquemáticamente todo lo que se ha abordado comentado acerca del patrón:

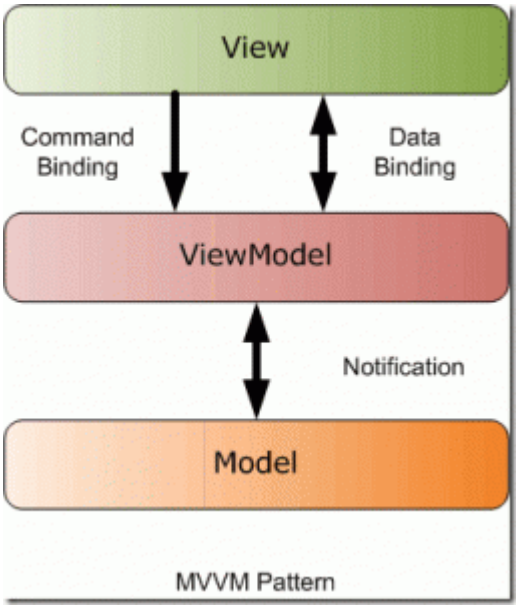


Figura 49. Patrón MVVM

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Con esta (breve) introducción teórica al patrón MVVM ya se puede obtener una conclusión importante para poder compartir código. Al ser posible separar la lógica de negocio y el modelo de datos de las particularidades de la vista, si se sabe desarrollar un ViewModel lo suficientemente desacoplado se podrá utilizar el mismo código en ambos proyectos W8 y WP8.

3.1.7. Lenguaje del robot.

En las máquinas controladas por sistemas informáticos, el lenguaje es el medio que utiliza el hombre para gobernar su funcionamiento, por lo que su correcta adaptación con la tarea a realizar y la sencillez de manejo, son factores determinantes del rendimiento obtenido en los robots industriales (Festo, s.f.).

Existen diversas maneras de comunicarse con un robot, entre las soluciones más populares para lograr este propósito están reconocimiento de palabras separadas, enseñanza y repetición y lenguajes de programación de alto nivel.

La enseñanza y repetición, también conocido como guiado, es la solución más común utilizada en el presente para los robots industriales. Este método implica enseñar al robot dirigiéndole los movimientos que el usuario desea que realice. La enseñanza y repetición se lleva a cabo normalmente con los siguientes pasos:

- Dirigiendo al robot con un movimiento lento utilizando el control manual para realizar la tarea completa y grabando los ángulos del movimiento del robot en los lugares adecuados para que vuelva a repetir el movimiento;
- Reproduciendo y repitiendo el movimiento enseñado;
- Si el movimiento enseñado es correcto, entonces se hace funcionar al robot a la velocidad correcta en el modo repetitivo.

Los lenguajes de programación de alto nivel suministran una solución más general para resolver el problema de comunicación hombre-robot. En la década anterior, los robots fueron utilizados con éxito en áreas tales como soldadura por arco voltaico o pintura con spray utilizando el guiado (Lenguajes de programación de los Robots, s.f.).

Estas tareas no requieren interacción entre el robot y su entorno y pueden ser programadas fácilmente por guiado.

Sin embargo, la utilización de robots para llevar a cabo las tareas requieren técnicas de programación en lenguajes de alto nivel ya que el robot de la línea de producción suele confiar en la realimentación de los sensores y este tipo de

interacción sólo puede solo puede ser mantenida por métodos de programación que contengan condiciones (Lenguajes de programación de los Robots, s.f.).

"Los lenguajes clásicos empleados en informática, como el FORTRAN, BASIC, PASCAL, etc., no disponen de las instrucciones y comandos específicos que necesitan los robots, para aproximarse a su configuración y a los trabajos que han de realizar" (Lenguajes de programación de los Robots, s.f.). Esta circunstancia, ha obligado a los constructores de robots e investigadores a diseñar lenguajes propios de la robótica. Sin embargo, los lenguajes desarrollados hasta el momento, se han dirigido a un determinado modelo de manipulador y a una tarea concreta, lo que ha impedido la aparición de lenguajes transportables entre máquinas y por lo tanto de carácter universal (Lenguajes de programación de los Robots, s.f.).

La estructura del sistema informático del robot varía notablemente, según el nivel y complejidad del lenguaje y de la base de datos que requiera.

3.1.7.1. Melfa Basic IV.

Melfa Basic IV fue el lenguaje que se utilizó para la elaboración del programa que corre en el controlador para este proyecto, no obstante el CR1-571 también soporta el lenguaje MovemasterCommand. El potente lenguaje de programación MELFA-BASIC IV sirve para controlar los robots y está basado en el lenguaje BASIC estándar. "Este lenguaje de programación permite la creación de programas altamente complejos y además no sólo controla secuencias sencillas de movimientos, sino que ejecuta por sí mismo cálculos de extrema complejidad sin tener que estar conectado a una PC" (Alonso, 2004). Esto es posible gracias a una extensa biblioteca de funciones, entre las que se cuentan también las funciones trigonométricas.

También, como cualquier otro lenguaje de programación, Melfa Basic IVpermite hacer ciclos, condiciones, interrupciones, declarar variable de varios tipos, por ejemplo: de posición, de articulación, variables para hacer operaciones aritméticas o bien para manejar una cadena de caracteres.

3.2. Herramientas

3.2.1. Visual Studio .NET.

Este es un IDE de desarrollo bastante amplio desarrollado por Microsoft el cual provee un conjunto de librerías bastante completa además de la comprensión de un grupo amplio de lenguajes de programación teniendo a la vez integrada la función para la ejecución de servicios sin la necesidad de configurar un servidor de aplicaciones externo también se puede encontrar un pequeño motor de base de datos y herramientas de control, depuración y versionamiento de código entre los más notorios.

3.2.2. SDK Kinect.

El SDK (kit de desarrollo) de Kinect además de contener los controladores del periférico también incluye Apis interfaces de dispositivos y documentación, dentro de este se pueden utilizar las herramientas provistas para los desarrolladores de videojuegos tales como la esqueletización, la captura de profundidad de imágenes y el procesamiento de voz, este SDK se integra muy fácilmente con el IDE VS .NET pudiendo ser utilizada desde los lenguajes de VB y C#.

3.2.3. Internet Information Services (IIS)

Es un servidor web y un conjunto de aplicaciones que permite a la PC con sistema operativo Windows desde Windows NT convertirse en un servidor de aplicaciones web tanto localmente como en el Internet, provee servicios para ejecutar paginas en ASP, servicios web nativamente y también PHP y JSP mediante la instalación de pluggins adicionales, puede configurarse con protocolos seguros además de proveer servicios de FTP, SMTP, NNTP y HTTP/HTTPS.

3.2.4. SQL Server.

Es un sistema de gestión de base de datos creado por Microsoft para la administración de datos sobre el sistema operativo Windows, este soporta transacciones, procedimientos almacenados, también posee un entorno gráfico en el cual es posible utilizar comandos DDL y DML gráficamente, soporta el trabajo en modo cliente servidor dejando a los clientes acceder a la información que se encuentra en el servidor

En el manejo de SQL mediante líneas de comando se utiliza el SQLCMD

Las desventajas de MSSQL es que usa AddressWindowingExtension (AWE) para hacer el direccionamiento de 64-bit. Esto le imposibilita utilizar la administración dinámica de memoria, y sólo permite guardar un máximo de 64 GB de memoria compartida, no trabaja con compresión de datos (excepto la versión 2008 Enterprise Edition), por lo que las bases de datos en aplicaciones empresariales llegan a ocupar mucho espacio en disco.

3.2.5. Visual StudioUnitTesting Framework.

Es una suite de Microsoft especializada en pruebas unitarias para mejorar la calidad y el correcto desarrollo de la capa de negocio de la aplicación, dentro de esta se puede realizar pruebas de métodos y clases implementadas dentro de la solución permitiendo simular todas los posibles valores que tomaran las variables dentro del sistema, trabaja creando métodos de prueba para el código desarrollado e indicando los errores que se pueden producir a la asignación de valores a las variables.

3.2.6. Ciros Studio.

"CIROS es un sistema general de simulación en 3D hecho en Alemania. Su flexibilidad lo hace idóneo para una gran variedad de áreas de aplicación además de poseer una gran variedad de opciones de equipamiento" (Festo, s.f.). Existen varias versiones de CIROS: CIROS® Robotics, CIROS® Mechatronics, CIROS® Production y CIROS® Studio entre otros. En el presento proyecto, se utilizó CIROS® Studio el cual representa una herramienta profesional para mediante una interfaz amigable (ver figura 3.7) llevar a cabo el modelado, la programación y la simulación de un entorno industrial. Algunas de sus características son:

- Posee todos los entornos de aprendizaje de mecatrónica, robótica y automatización industrial en una misma plataforma.
- Nueva interface de usuario orientada a la manipulación para tareas y proyectos con un nuevo asistente didáctico en línea.
- Bibliotecas con sistemas de robots industriales y numerosos componentes de automatización.

- Soporta los lenguajes MovemasterCommand y Melfa Basic IV para los robots Mitsubishi.

- Interfaz de gran alcance para el control del robot de Mitsubishi a través de Ethernet TCP/IP o el puerto serie.

- Permite descargar los programas y archivos, visualizar en línea los datos del sistema del robot.

- Ejecución y depuración del programa línea por línea.

- Simulación en 3D en tiempo real: incluye simulación de transporte, de las conexiones de la tubería y las cadenas de la energía.

- Detección de colisiones a través del cambio de color o mensajes de advertencia con y sin acuse de recibo.

- Simulación del sensor: desde el sensor inductivo para la cámara, casi todos los sensores y sus propiedades físicas pueden ser simuladas. De esta manera es posible analizar la interacción realista con el equipo periférico.

- Cliente OPC con el menú de configuración para la comunicación con cualquier servidor OPC.

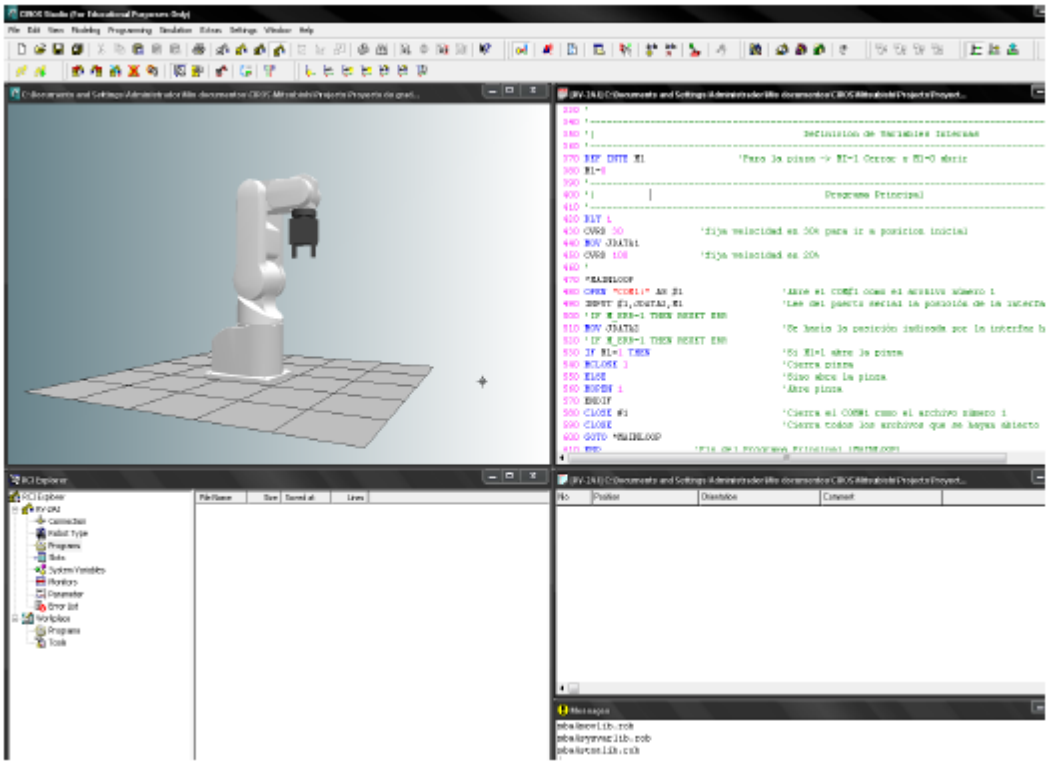


Figura 50. Ambiente de trabajo del software CIROS Studio

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Los requerimientos que debe de tener la PC en la que se va a instalar CIROS son los siguientes:

- Procesador de 1.0 GHz, aunque el recomendado es de 2.2 GHz.
- sistema operativo: Windows XP Service Pack 2 ó Windows Vista.18
- Memoria RAM de 512 MB, aunque se recomienda 2 GB
- 5 GB de espacio en el disco duro, aunque se recomienda 20 GB
- Windows XP o Vista, Microsoft Internet Explorer Versión 5.0 o superior
- Tarjeta de video NVIDIA de 128MB, aunque se recomienda 7800GTX, 512 MB o superior
- Monitor de 17” con una resolución de 1024 x 768 pixeles, aunque se recomienda uno de 19" con resolución de 1280 x 1024 píxeles

- Un puerto USB para la llave (dongle) del software CIROS que hace de licencia
- Unidad de DVD-ROM
- 1 puerto serie
- Adobe Acrobat Reader versión 6.0 o superior

3.2.7. Api de Comunicación RS232.

El protocolo para la conexión de la computadora con el controlador CR1-571 fue el serial bajo el estándar RS-232. En un principio se quiso usar USB, no obstante solo los controladores que le siguen al CR1-571 poseen ese tipo de conexión (Mitsubishi). Por otro lado, para el intercambio de datos inicialmente se pensó en protocolos industriales tales como DDE u OPC, pero debido a la facilidad que representa el uso del puerto serial en la computadora se prefirió usar el estándar.

RS-232. El control del puerto serie se realizó en C++ utilizando las funciones y estructuras que el WIN API32 tiene implementadas para realizar esta tarea. Para usar el puerto serial correctamente, se necesitó cuatro funciones: abrir (configuración), escribir, leer y finalmente cerrar el puerto.

3.3. Servicio web.

El servicio web está desarrollado en Visual Studio 2012 utilizando Framework 4.0. utilizando tecnología WCF.

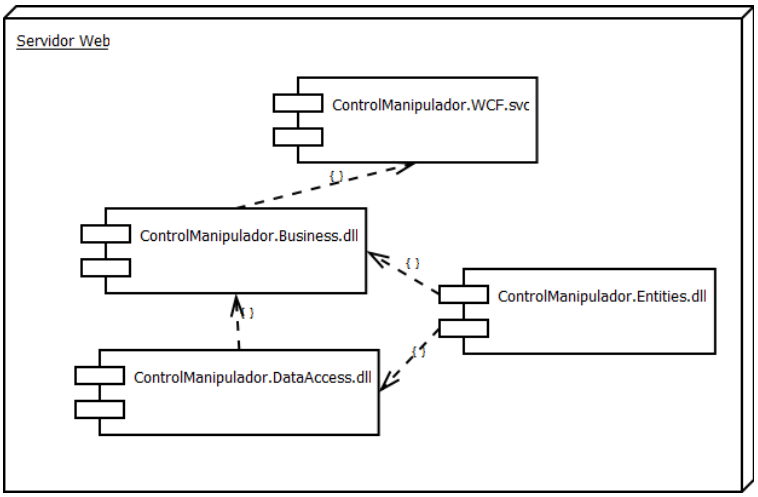


Figura 51. Diagrama de Despliegue servicio web este no diagrama de clases

Elaborado por: (Castro, Slin & Jiménez, Fernando)

El proyecto está separado por su naturaleza en 2 partes pasa su explicación, primero se explicara la lógica del negocio donde se encuentra todo el código funcional del servicio esta parte se subdivide en 3 proyectos, se tiene la lógica de negocio (ControlManipulador.Business.dll), el acceso a datos (ControlManipulador.dataAcces.dll), y las entidades (ControlManipulador.Entities.dll). La segunda parte se compone del servicio contratos e implementación contenidos en ControlManipulador.WCF.svc.

3.3.1. Entidades de negocio.

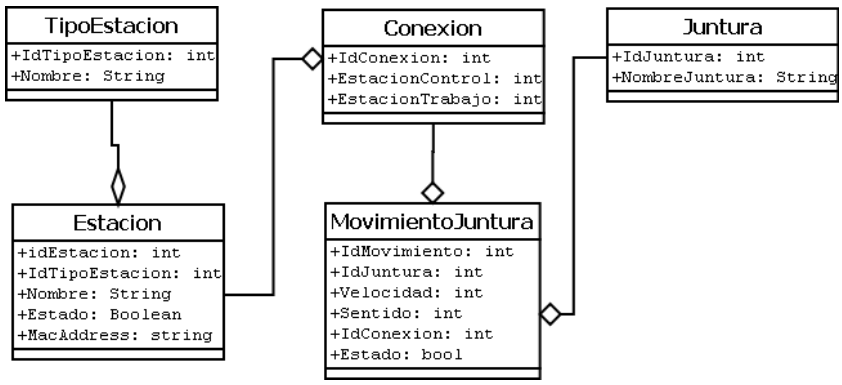


Figura 52. Diagrama de clases de entidades de negocios

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Para empezar a entender el funcionamiento del servicio se describirá cada una de las entidades del negocio, hay que tomar en cuenta que usando el patrón Code First primero se generan las entidades y luego con estas se crea la base de datos que en este caso será construida utilizando EntityFramework, cada clase contendrá la información necesaria en sus propiedades para generar la columna, el tipo de dato y restricciones a estos.

3.3.1.1. Clase Juntura.

```
public class Juntura
    {
        [Key]
        [databaseGenerated(databaseGeneratedOption.Identity)]
        public int IdJuntura { get; set; }

        [Column(TypeName = "Varchar")]
        public string NombreJuntura { get; set; }
    }
```

Figura 53. Declaración de clase Juntura

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta entidad o clase está formada por la información necesaria para identificar a la juntura del manipulador.

3.3.1.2. Clase Estacion.

```
public class Estacion
 {
     [Key]
     [databaseGenerated(databaseGeneratedOption.Identity)]
     public int IdEstacion { get; set; }

     [ForeignKey("TipoEstacion")]
     public int IdTipoEstacion { get; set; }

     [Column(TypeName = "Varchar")]
     public string Nombre { get; set; }

     public bool Estado { get; set; }

     public string MacAdress { get; set; }

     public TipoEstacion TipoEstacion { get; set; }
 }
```

Figura 54. Declaración de clase Estacion

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta contiene la información de la estación, además de tener una clave foránea de la tabla TipoEstacion. Entre sus campos principales se encuentra el identificador IdEstacion que es auto numérico y clave primaria que junto con el campo MacAdress son únicos por cada registro.

3.3.1.3. Clase TipoEstacion.

```
public class TipoEstacion
{
    [Key]
    [databaseGenerated(databaseGeneratedOption.Identity)]
    public int IdTipoEstacion { get; set; }

    [Column(TypeName = "Varchar")]
    public string Nombre { get; set; }
}
```

Figura 55. Declaración de clase TipoEstacion

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta contiene la información de los tipos de estaciones. Entre sus campos principales se encuentra el identificador IdTipoEstacion que es auto numérico.

3.3.1.4. Clase MovimientoJuntura.

```
public class MovimientoJuntura
    {
        [Key]
        [databaseGenerated(databaseGeneratedOption.Identity)]
        public int IdMovimiento { get; set; }

        [ForeignKey("Juntura")]
        public int IdJuntura { get; set; }

        public Juntura Juntura { get; set; }

        public int Velocidad { get; set; }
        public int Sentido { get; set; }

        [ForeignKey("Conexion")]
        public int IdConexion { get; set; }

        public Conexion Conexion { get; set; }

        public bool Estado { get; set; }
    }
```

Figura 56. Declaración de clase TipoJuntura

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta contiene la información de los movimientos por estación y juntura, además de tener dos claves foráneas de las tablas **"Coneccion"** y **"Juntura"**. Entre sus campos principales se encuentra el identificador IdMovimiento que es auto numérico y clave.

3.3.1.5. Clase "Conexion".

```
public class Conexión
    {
        [Key]
        [databaseGenerated(databaseGeneratedOption.Identity)]
        public int IdConexion { get; set; }

        [ForeignKey("Estacion")]
        public int EstacionControl { get; set; }

        public int EstacionTrabajo { get; set; }

        public Estacion Estacion { get; set; }
    }
```

Figura 57. Declaración de clase "Conexión"

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta contiene la información de las conexiones, además de tener una clave foránea de la tabla **"Estacion"**. Esta tabla relaciona el movimiento con las estaciones que intervienen en este, entre sus campos principales se encuentra el identificador IdConexion que es auto numérico y clave primaria.

3.3.1.6. Clase "RespuestaProceso".

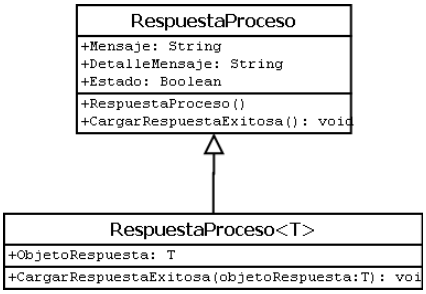


Figura 58. Diagrama de clases entidad respuesta proceso

Elaborado por: (Castro, Slin & Jiménez, Fernando)

```
[dataContract]
public class RespuestaProceso
{

    public RespuestaProceso()
    {
        Estado = false;
        Mensaje = String.Empty;
        DetalleMensaje = string.Empty;
    }

    [dataMember]
    public string Mensaje { get; set; }
    [dataMember]
    public string DetalleMensaje { get; set; }

    [dataMember]
    public Boolean Estado { get; set; }


    public void CargarRespuestaExitosa()
    {
        Estado = true;
        Mensaje = "Operación Exitosa";
        DetalleMensaje = "Operación Exitosa";
    }
}

[dataContract]
public class RespuestaProceso<T> : RespuestaProceso
{
    [dataMember]
    public T ObjetoRespuesta { get; set; }

    public void CargarRespuestaExitosa(T objetoRespuesta)
    {
        CargarRespuestaExitosa();
        ObjetoRespuesta = objetoRespuesta;
    }
}
```

Figura 59. Declaración de contratos de servicios

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Estas clases no estarán replicadas en la base de datos ya que se encargan de transportar las respuestas desde los niveles de lógica hacia la interfaz , la primera contiene las propiedades de estado y mensajes de error o éxito, además contiene métodos para cargar directamente las respuestas si estas son correctas, la segunda hereda de la primera para usar sus propiedades, pero esta es genérica ya que de esta manera se puede transportar cualquier tipo de variable sin la necesidad de declarar una de cada tipo.

3.3.2. Lógica de negocio (BL).

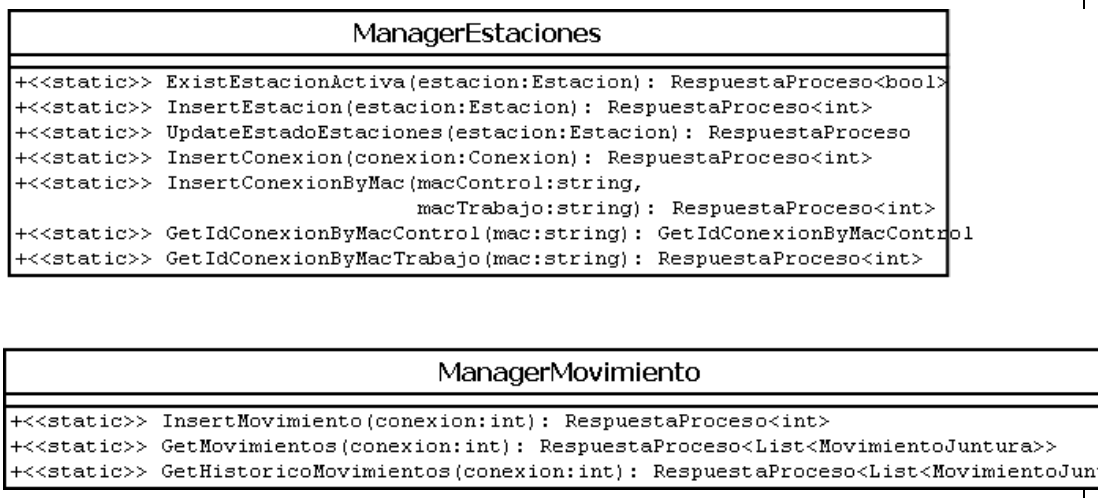


Figura 60. Diagrama de clases BL

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta capa es invocada por la implementación de contratos del servicio WCF, sus clases serán estáticas para que se conserven en memoria en tiempo de ejecución de esta manera será más rápido y fácil acceder a ellas.

Para su fácil identificación su nombre estará precedido por la palabra manager. Esta capa en la solución representa una librería en C# y está formada por 2 clases.

3.3.2.1. Clase "ManagerEstaciones".

Esta clase es la encargada de orquestar las operaciones referentes a la estación, es pública para que pueda ser accedida desde otras partes del proyecto sin la necesidad de usar una interfaz, la clase general no es estática pero contiene elementos que si lo son.

```
 public static RespuestaProceso<bool> ExistEstacionActiva(Estacion estacion)
        {
            var respuesta = new RespuestaProceso<bool>();

            Try
            {

respuesta.CargarRespuestaExitosa(ModelEstaciones.ExistEstacionActiva(estacion));
            }
            catch (Exception ex)
            {
                respuesta.Estado = false;
                respuesta.Mensaje = "No se pudo comprobar la Conexion";
                respuesta.DetalleMensaje = ex.Message;
```

Figura 61. Declaración de método ExistEstacionActiva

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Es un método de ejemplo que puede describir el funcionamiento general de todos los integrantes de esta capa, es público para que pueda ser invocado desde los contratos del servicio y estático para su rápido acceso, devuelve un objeto de la clase RespuestaProceso, contiene un captor de excepciones que los errores producidos en el programa, asignando el estado de la transacción a falso y el mensaje de error.

Entre los métodos principales de esta clase se tiene:

```
public static RespuestaProceso<int> InsertConexionByMac(string macControl, string
macTrabajo)
        {
            var respuesta = new RespuestaProceso<int>();

            try
            {

                var conexion = new Conexion { EstacionControl =
ModelEstaciones.GetIdEstacionByMac(macControl), EstacionTrabajo =
ModelEstaciones.GetIdEstacionByMac(macTrabajo) };
                var idMovimiento = ModelEstaciones.InsertConexion(conexion);
                respuesta.CargarRespuestaExitosa(idMovimiento);
            }
            catch (Exception ex)
            {
                respuesta.Estado = false;
                respuesta.Mensaje = "No se pudo comprobar la Conexion";
                respuesta.DetalleMensaje = ex.Message;

            }

            return respuesta;
        }
```

Figura 62. Declaración de método InsertConexionByMac

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Este método se encarga de insertar las conexiones dependiendo del MacAdress de la estación de control, recibe las 2 MacAdress previamente creadas, luego crea un objeto

de tipo conexión obteniendo los id de las estaciones a partir de los datos recibidos, después hace una llamada a la capa de persistencia para ingresarlos, todo este proceso está entre el captor de excepciones, y retorna un objeto de tipo RespuestaProceso que contiene el Id de la conexión creada.

```
public static RespuestaProceso UpdateEstadoEstaciones(Estacion estacion)
        {
            var respuesta = new RespuestaProceso();

            try
            {
                if (!ModelEstaciones.ExistEstacion(estacion))
                    ModelEstaciones.InsertEstacion(estacion);

                ModelEstaciones.UpdateEstadoEstacion(estacion);
                respuesta.CargarRespuestaExitosa();
            }
            catch (Exception ex)
            {
                respuesta.Estado = false;
                respuesta.Mensaje = "No se pudo Obtener las Estaciones";
                respuesta.DetalleMensaje = ex.Message;

            }

            return respuesta;
        }
```

Figura 63. Declaración de método UpdateEstadoEstaciones

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Este método se encarga de crear o actualizar una estación, recibe una variable de tipo estación y devuelve una variable de tipo RespuestaProceso.

3.3.2.2. Clase ManagerMovimiento.

Esta clase es la encargada de orquestar las operaciones referentes a los movimientos, es pública para que pueda ser accedida desde otras partes del proyecto sin la necesidad de usar una interfaz, la clase general no es estática pero contiene elementos que sí lo son.

Entre los métodos principales de esta clase se tiene:

```
public  static  RespuestaProceso<List<MovimientoJuntura>>  GetHistoricoMovimientos(int
conexion)
        {
            var respuesta = new RespuestaProceso<List<MovimientoJuntura>>();

            try
            {
                var movimientos = ModelMovimientos.GetHistoricoMovimientos(conexion);
                respuesta.CargarRespuestaExitosa(movimientos);
            }
            catch (Exception ex)
            {
                respuesta.Estado = false;
                respuesta.Mensaje = "No se pudo obtener movimientos";
                respuesta.DetalleMensaje = ex.Message;

            }

            return respuesta;
        }
```

Figura 64. Declaración de método GetHistoricoMovimientos

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Este método se encarga de insertar las consultar los movimientos por el id de conexión, todo este proceso está entre el captor de excepciones, y retorna un objeto de tipo RespuestaProceso que contiene la lista de movimientos históricos.

3.3.3. Acceso a datos (DA)

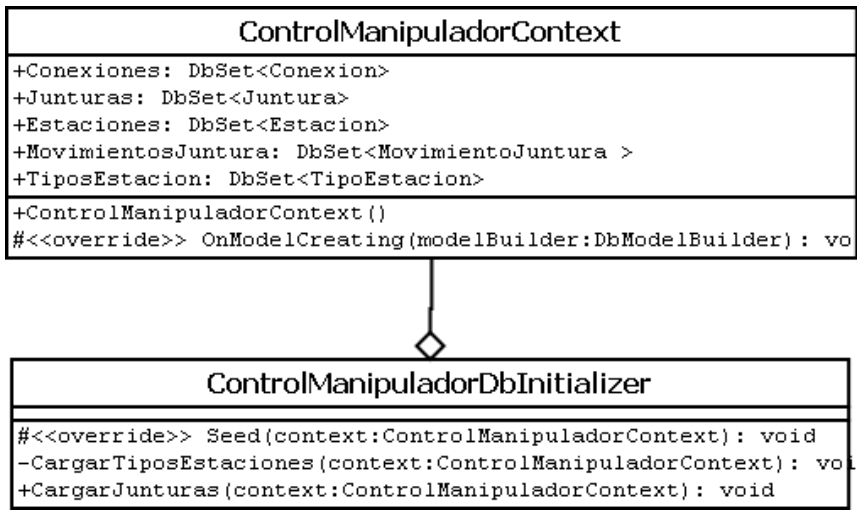


Figura 65. Diagrama de clases DA

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Está representado en la solución como una librería de clases de C# es la capa encargada de crear e interactuar con la base de datos.

3.3.3.1. Clase ControlManipuladorContext.

```
public class ControlManipuladorContext : DbContext
    {
     public ControlManipuladorContext()
         : base("SuscripcionesConeccion")
    {

        database.SetInitializer(new ControlManipuladorDbInitializer());

    }

    public DbSet<Conexion> Conexiones { get; set; }
    public DbSet<Juntura> Junturas { get; set; }
    public DbSet<Estacion> Estaciones { get; set; }
    public DbSet<MovimientoJuntura > MovimientosJuntura { get; set; }
    public DbSet<TipoEstacion> TiposEstacion { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
        }
    }
```

Figura 66. Declaración de clase ControlManipuladorContext

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta clase se encarga de gestionar los contextos con la base de datos, hereda de DbContext (clase nativa de EntityFramework) se utiliza un constructor donde se puede definir el string de conexión alojado en el archivo web.config, y dentro de este se llama a la clase que inicializa la base de datos. Luego están declaradas como propiedades a las clases que aparecerán en la base de datos, finalmente se sobrescribe el método OnModelCreating para fijar que la base de datos se cree cada vez que el modelo cambie.

3.3.3.2. Clase ControlManipuladorDbInitializer.

```
class ControlManipuladorDbInitializer :
DropCreatedatabaseIfModelChanges<ControlManipuladorContext>
    {
        protected override void Seed(ControlManipuladorContext context)
        {
            CargarJunturas(context);
            CargarTiposEstaciones(context);
           // CargarEstaciones(context);
        }

        private void CargarJunturas(ControlManipuladorContext context)
        {
            IList<Juntura> defaultJunturas = new List<Juntura>();


            defaultJunturas.Add(new Juntura {NombreJuntura="Cintura"});
            defaultJunturas.Add(new Juntura { NombreJuntura = "Hombro" });
            defaultJunturas.Add(new Juntura { NombreJuntura = "Codo" });
            defaultJunturas.Add(new Juntura { NombreJuntura = "Munieca" });
            defaultJunturas.Add(new Juntura { NombreJuntura = "Gripper" });

            foreach (var prm in defaultJunturas)
                context.Junturas.Add(prm);

            base.Seed(context);
        }
        private void CargarTiposEstaciones(ControlManipuladorContext context)
        {
            IList<TipoEstacion> defaultEstaciones = new List<TipoEstacion>();


            defaultEstaciones.Add(new TipoEstacion { Nombre = "Trabajo" });
            defaultEstaciones.Add(new TipoEstacion { Nombre = "Control" });

            foreach (var prm in defaultEstaciones)
                context.TiposEstacion.Add(prm);

            base.Seed(context);
        }
        private void CargarEstaciones(ControlManipuladorContext context)
        {
            IList<Estacion> defaultEstaciones = new List<Estacion>();


            defaultEstaciones.Add(new                    Estacion                    {
Nombre="ETrabajo1",IdTipoEstacion=1,Estado=true });
            defaultEstaciones.Add(new Estacion { Nombre = "EControl",IdTipoEstacion =
2, Estado = true });

            foreach (var prm in defaultEstaciones)
                context.Estaciones.Add(prm);

            base.Seed(context);
        }
    }
```

Figura 67. Declaración de clase ControlManipuladorDbInitializer

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Esta clase se encarga de cargar los datos iniciales a la base generada por el contexto, en la cual se cargan estaciones, tipos de estaciones y las junturas.

3.3.3.3. Interacción con BDD.

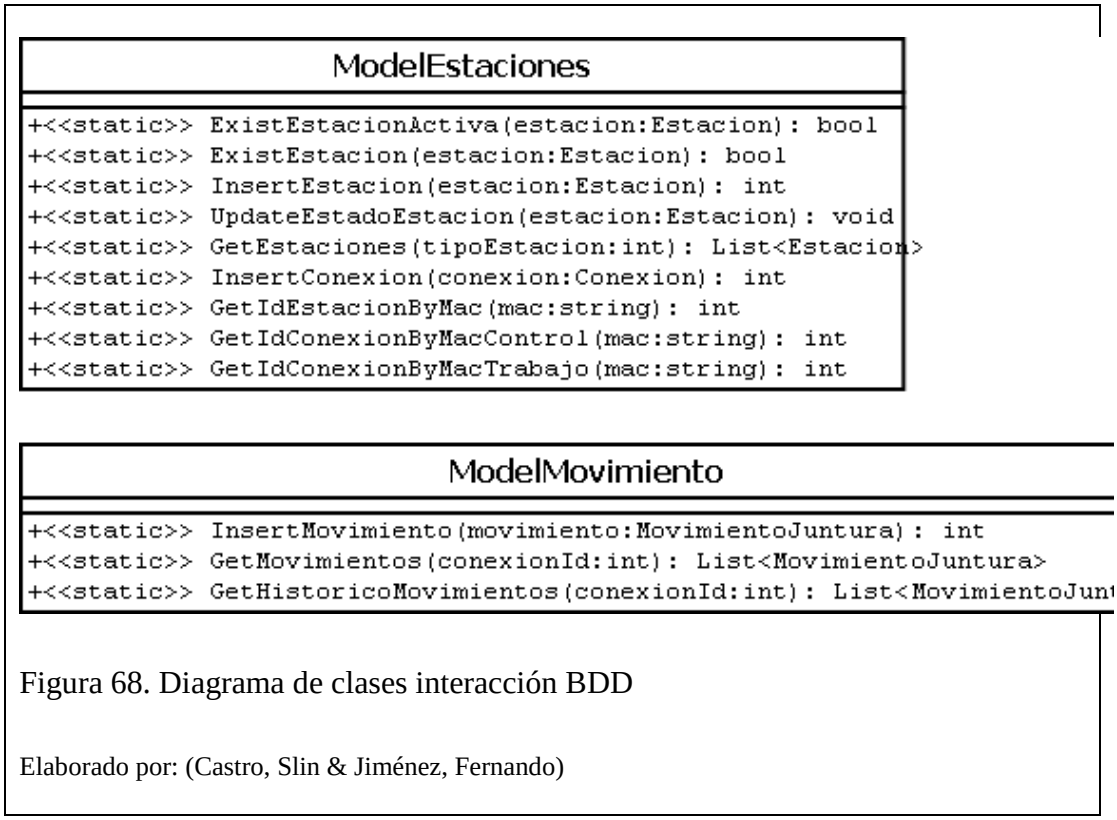


Figura 68. Diagrama de clases interacción BDD

Elaborado por: (Castro, Slin & Jiménez, Fernando)

3.3.3.3.1. ModelEstaciones.

Esta clase se encarga de la gestión con la base de datos de la información de las estaciones, entre sus métodos principales están:

```
public static void UpdateEstadoEstacion(Estacion estacion){
            using (var ctx=new ControlManipuladorContext())            {
                var       request       =       ctx.Estaciones.FirstOrDefault(x       =>
x.Nombre.Equals(estacion.Nombre) || x.IdEstacion.Equals(estacion.IdEstacion));
                if (request == null)                {
                    throw new ArgumentException("No existe la Estacion a activar");
                }
                request.Estado = estacion.Estado;
                ctx.SaveChanges();     }      }
```

Aquí se recibe un objeto de tipo estación, se abre la conexión hacia el contexto, se ejecuta el update de la estación requerida y se guarda los cambios.

3.3.3.3.2. ModelMovimientos.

Esta clase se encarga de la gestión con la base de datos de la información de las conexiones y movimientos, entre sus métodos principales se tiene:

```
public static List<MovimientoJuntura> GetMovimientos(int conexionId)
        {
            using (var ctx = new ControlManipuladorContext())
            {
                var movimientos = ctx.MovimientosJuntura.Where(x=>x.Estado &&
x.IdConexion.Equals(conexionId));
                var respuestaMovimientos = movimientos.ToList();
                if (movimientos != null)
                {
                    movimientos.ToList().ForEach(x => x.Estado = false);
                    ctx.SaveChanges();
                    return respuestaMovimientos;
                }


                throw new ArgumentException("No se encontraron Estaciones");
            }
        }
```

Figura 69. Declaración de método GetMovimientos

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Aquí se recibe un id de conexión, se procede a abrir la conexión hacia el contexto, y se ejecuta la consulta con una expresión Lambda para terminar con la devolución de la información.

3.3.4. Test.

La carpeta test contiene 2 proyectos, un proyecto que consta de un WinForm utilizado como un mock de un cliente para probar el funcionamiento manipulable del servicio web, el otro proyecto es de tipo test, la estructura general de los métodos de prueba es:

```
[Test, Category("Prueba")]
publicvoid Prueba()
{
Assert.IsTrue(true);
 }
```

Figura 70. Declaración de método de prueba

Elaborado por: (Castro, Slin & Jiménez, Fernando)

En el decorador se indica 2 argumentos, el primero indica que es un método de prueba y el segundo asigna la categoría para clasificar las pruebas, el método es público para que el test explorer pueda acceder a este, y es void porque al ser un método de prueba solo interesa el resultado que dependiendo de la configuración del entorno se podrá mostrar gráficamente o como una respuesta en consola, assert es la clase que contiene los validadores para comprobar si se está ejecutando bien la acción.

```
 [Test, Category("Mantenimiento Programa")]
 [Explicit]
publicvoidVerificarInsercion()
 {
ControlRespuesta control = newMantenimientoPrograma().InsertarProgramas(_programa);
Assert.IsTrue(control.Estado);
 }
```

Figura 71. Declaración de test unitario de verficacion de inserción

Elaborado por: (Castro, Slin & Jiménez, Fernando)

En este método se puede verificar que además de los decoradores de prueba se contiene otro donde indica que este método solo será ejecutado manualmente cuando sea requerido.

3.3.5. WCF.

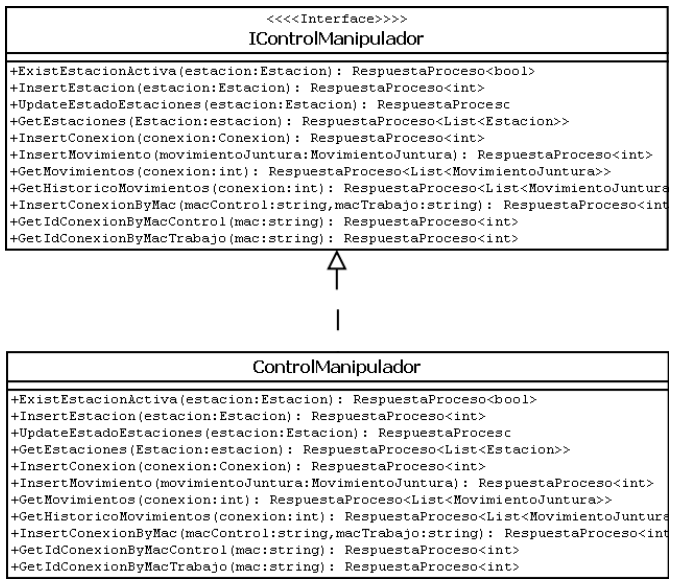


Figura 72. Diagrama de clases WCF

Elaborado por: (Castro, Slin & Jiménez, Fernando)

WCF se encuentra la interface y los contratos del servicio, en esta están definidos los tipos de dato a intercambiar mediante la comunicación y la definición de los métodos a usarse en el servicio.

```
[ServiceContract]
   public interface IControlManipulador
   {

       [OperationContract]
       RespuestaProceso<bool> ExistEstacionActiva(Estacion estacion);

       [OperationContract]
       RespuestaProceso<int> InsertEstacion(Estacion estacion);

       [OperationContract]
       RespuestaProceso UpdateEstadoEstaciones(Estacion estacion);

       [OperationContract]
       RespuestaProceso<List<Estacion>> GetEstaciones(Estacion estacion);

       [OperationContract]
       RespuestaProceso<int> InsertConexion(Conexion conexion);

       [OperationContract]
       RespuestaProceso<int> InsertMovimiento(MovimientoJuntura movimiento);

       [OperationContract]
       RespuestaProceso<List<MovimientoJuntura>> GetMovimientos(int conexion);

       [OperationContract]
       RespuestaProceso<List<MovimientoJuntura>>GetHistoricoMovimientos(int
conexion);

       [OperationContract]
       RespuestaProceso<int>    InsertConexionByMac(string    macControl,    string
macTrabajo);

       [OperationContract]
       RespuestaProceso<int> GetIdConexionByMacControl(string mac);

       [OperationContract]
       RespuestaProceso<int> GetIdConexionByMacTrabajo(string mac);
   }
```

Figura 73. Declaración de interface de contrato

Elaborado por: (Castro, Slin & Jiménez, Fernando)

El decorador datacontract se utiliza para indicar al compilador que esta será la clase utilizada para el intercambio de datos. Es público ya que es un objeto general para el intercambio de respuestas en la aplicación.

En esta interface se encuentran todas las operaciones que podrán ser usadas al invocar los servicios web y descritas a continuación.

ExistConexionActiva: recibe un objeto tipo estación y verifica que esta se encuentre con un estado true en la BDD y responde un objeto tipo respuesta operación conteniendo un booleano.

InsertEstacion: recibe un objeto tipo estación, verifica que esta exista en la BDD, si existe la cambia de estado y si no la crea responde un objeto tipo respuesta operación conteniendo el id de estación.

UpdateEstadoEstaciones: recibe un objeto tipo estación y responde un objeto tipo Respuesta operación conteniendo un booleano.

GetEstaciones: recibe un objeto tipo estación y devuelve todas las estaciones disponibles.

InsertConexion: recibe un objeto tipo conexión, lo inserta y devuelve el id de Conexión.

InsertMovimiento: recibe un objeto tipo movimientoJuntura, lo inserta y devuelve el id de Conexión.

GetMovimientos: recibe un id de conexión y devuelve todas los movimientos activos para esa conexión.

GetHistoricoMovimientos: recibe un id de conexión y devuelve todas los movimientos para esa conexión.

InsertConexion: recibe las mac de las estaciones, realiza las operaciones necesarias para insertarla y devuelve el id de Conexión.

GetIdConexionbyMacControl: recibe una mac y devuelve el id de la estación.

GetIdConexionbyMacTrabajo: recibe una mac y devuelve el id de la estación.

3.3.5.1. Contratos.

```
publicclassControlManipulador : IControlManipulador
```

La clase control manipulador hereda de la interface del servicio y será donde se implementen los métodos, en esta clase se realizaran las llamadas a la lógica de negocio.

Dentro de la interface del servicio se mostraran los métodos que podrán ser accedidos desde cualquier cliente del servicio web, el decorador servicecontract indica al compilador que esta interface es la contenedora de dichos métodos.

```
[ServiceContract]
publicinterfaceIControlManipulador
```

El decorador OperationContract indica que el método declarado a continuación es de libre acceso por el cliente.

```
[OperationContract]
ControlRespuestaInsertarPrograma(ControlRespuesta programa);
```

3.3.6. WEBS.

Contiene las configuraciones del servicio y la localización de la interfaz del servicio y sirve de host para la aplicación, la carpeta Bin contiene los archivos .dll de todo el proyecto facilitando el despliegue de la aplicación ya que si realiza un cambio en una librería solo se remplazara esta y no se tendrá que recompilar y sustituir todo el proyecto.

```
<%@ServiceHostLanguage="C#"Debug="true"Service="ControlManipulador.WCF.IControlMani
pulador.ControlManipulador"%>
```

Esta línea es de vital importancia ya que es la ejecutada al iniciar el servicio, indica el lenguaje, el modo debug que en fase de producción será cambiada a false para impedir el debug remoto, y la localización de la interface desde donde se ejecutaran todos los métodos.

Web.Config

Lleva la información general de cómo funcionará el servicio en el ambiente web, tomando en cuenta aspectos como protocolos, tipos de mensajería, bindings, entre otros las líneas más relevantes del config son:

```
<identity>
<dnsvalue="localhost"/>
</identity>
```

Como en este caso la aplicación corre localmente el valor de la dns será el localhost.

```
<endpoint
address=""binding="basicHttpbinding"contract="ControlManipulador.WCF.IControlManipu
lador.IControlManipulador">
```

Los endpoints indican el tipo de binding que tendrá la aplicación en este caso será basicHttpbinding ya que no posee seguridad para la conexión.

```
<serviceMetadatahttpGetEnabled="True"httpsGetEnabled="false"/>
```

Indica el protocolo de comunicación entre el servicio y su cliente.

```
<connectionStrings>
<addname="ControlManipulador.DA.Properties.Settings.ControlbrazoConnectionString"
connectionString="data          Source=.;Initial          Catalog=Controlbrazo;User
ID=sa;Password=sa.1"
providerName="System.data.SqlClient" />
</connectionStrings>
```

Los connectionStrings contienen la información de conexión a la base de datos, así cuando el servicio sea desplegado solo se tendrá que cambiar en esta línea la localización y credenciales de la base de datos sin depender de compilar nuevamente el proyecto

3.4. Clientes estación de control y trabajo.

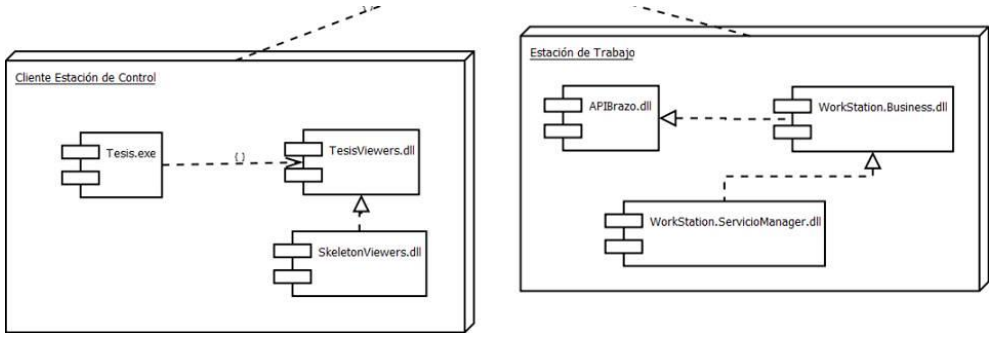


Figura 74. Clientes estación de control y trabajo

Elaborado por: (Castro, Slin & Jiménez, Fernando)

El cliente Tesis consume las vistas generadas mediante el patrón MVVM almacenadas en la librería TesisViewers.dll, para permitir al usuario final utilizar el sistema compuesto por el conjunto de configuraciones y parámetros del dispositivo Kinect, obtenidos a partir de la conexión y utilización del SDK Microsoft para Kinect, a su vez se implementa una interfaz principal utilizando la vista encargada de la creación, verificación y conexión del sistema con el dispositivo Kinect.

En la pantalla Principal denominada KinectWindow se presenta el conjunto de imágenes procesadas mediante los algoritmos de generación de esqueletos creado en la librería TesisViewers.dll, así como la imagen en tiempo real de la cámara IP, permitiendo la visualización del brazo robótico Mitsubishi RV-2AJ en tiempo real, de esta manera el usuario final tiene conocimiento del resultado de sus acciones en el sistema y como interactuar de manera adecuada con el mismo.

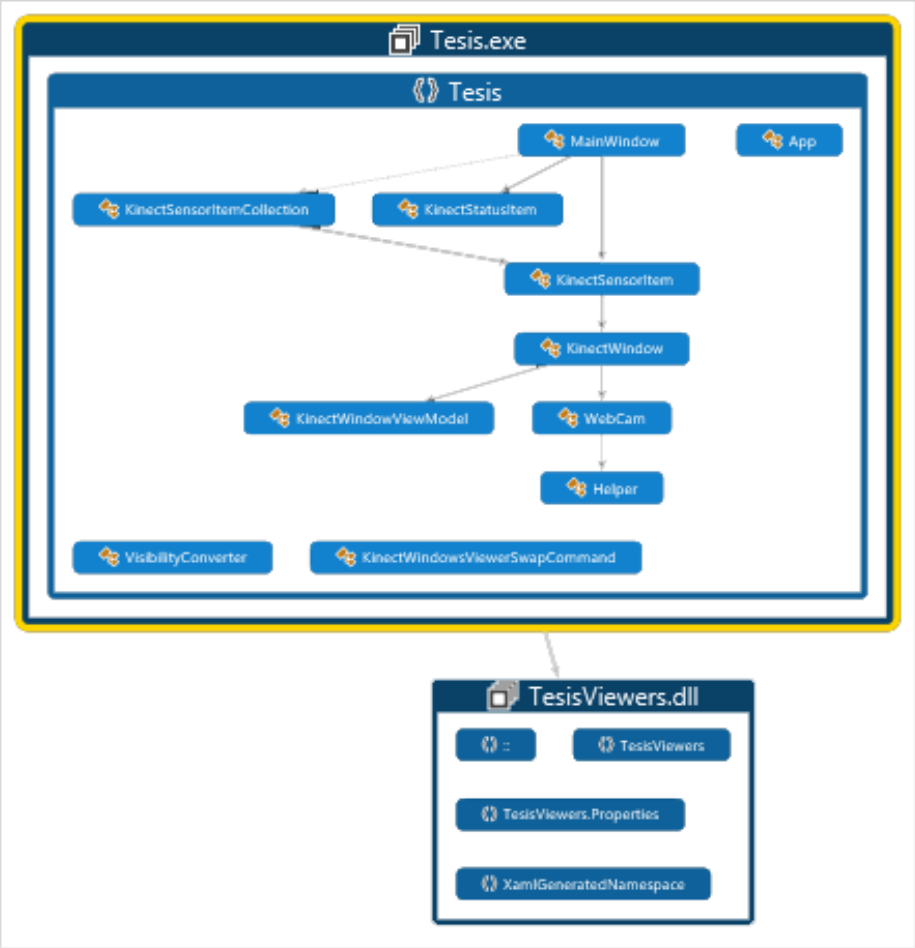


Figura 75. Modelo del Cliente Kinect

Elaborado por: (Castro, Slin & Jiménez, Fernando)

3.4.1. TesisViewers.

Se ha creado una librería que contiene las vistas necesarias para la presentación e interacción del sistema con el dispositivo Kinect utilizando las referencias de Microsoft.Kinect siendo las más importantes las clases KinectSettings y KinectSkeletonViewer.

3.4.2. KinectSettings.

Esta clase implementa las vistas KinectSettingsViewModel y KinectSkeletonChosser, las cuales permiten el ajuste del dispositivo Kinect tanto en su ángulo de inclinación como la manera en la que se trata la esqueletización.

Implementa la propiedad elevationAngle ubicada en la clase KinectSensorManager, la misma que permite de manera dirigida variar el ángulo de inclinación del dispositivo permitiendo así alinear las cámaras para poder capturar la imagen para la esqueletización de manera óptima.

3.4.3. KinectSekelonViewer.

En esta clase se implementa el visor de esqueletos el cual utiliza la clase KinectSkeleton para generar el esqueleto del usuario programáticamente utilizando la captura y procesamiento de imagen del dispositivo Kinect, a la vez permite la creación de datos de movimiento y ubicación de usuario en pantalla, enviando las tramas de captura al servicio web.

El método principal y el encargado de todo el procesamiento es DrawBone cuya firma es **(DrawingContext, JointType jointType1, JointType jointType2).**

De manera integral se necesita el contexto gráfico en donde será presentado el esqueleto denominado drawingContext del Tipo DrawingContext, adicional a ello se integran al proceso los tipos de junturas denominados jointType1 y jointType2 del tipo JointType.

Una vez determinado el tipo de Joint así como el contexto gráfico se procede a su procesamiento utilizando el método DrawBonesAndJoints con la firma DrawBonesAndJoints (DrawingContextdrawingContext), en este método se determina el tipo y ubicación de las junturas y puntos de imagen capturando los datos en pantalla para ser procesado y enviados al servicio web.

3.4.4. Api brazo.

En esta clase se implementa el manejo conexión y desconexión con el brazo robótico Mitsubishi RV-2AJ a través de la comunicación serial utilizando el bus de comunicaciones de Microsoft.

Tabla 4. Constructores Serial Port

Nombre	Descripción
SerialPort()	Inicializa una nueva instancia de la clase SerialPort.
SerialPort(IContainer)	Inicializa una nueva instancia de la clase SerialPort utilizando el objeto IContainer especificado.
SerialPort(String)	Inicializa una instancia nueva de la clase SerialPort utilizando el nombre de puerto especificado.
SerialPort(String, Int32)	Inicializa una instancia nueva de la clase SerialPort utilizando el nombre de puerto y la velocidad en baudios especificados.
SerialPort(String, Int32, Parity)	Inicializa una instancia nueva de la clase SerialPort utilizando el nombre del puerto, la velocidad en baudios y el bit de paridad especificados.
SerialPort(String, Int32, Parity, Int32)	Inicializa una instancia nueva de la clase SerialPort utilizando el nombre del puerto, la velocidad en baudios, el bit de paridad y los bits de datos especificados.
SerialPort(String, Int32, Parity, Int32, StopBits)	Inicializa una instancia nueva de la clase SerialPort utilizando el nombre del puerto, la velocidad en baudios, el bit de paridad, los bits de datos y el bit de parada especificados.

Fuente: (Microsoft, s.f.)

CAPÍTULO 4

PRUEBAS

La batería de prueba se ha dividido de manera que se pueda abarcar los tres bloques generales del desarrollo y a su vez se ha incluido un proceso de pruebas de integración entre las partes actuantes del sistema, se las realizará de esta manera debido a la facilidad que presenta encontrar errores al realizar una prueba por cada componente y posteriormente realizar una prueba de integración general. Las pruebas de los dos clientes tanto la estación de trabajo como la estación de control han sido realizadas manualmente ya que estas requieren de una comunicación con periféricos que difícilmente se pueden simular de una manera óptima, en tanto que las pruebas realizadas al servicio web se las realizará con un componente adicional de VisualStudio llamado NUnit, de esta manera se evita crear interfaces para el envío y recepción de datos creando un ambiente automatizado de pruebas.

4.1. NUnit

Es un Framework de pruebas unitarias de xUnit que además de ser opensource también se lo puede encontrar disponible para Java, este permite comprobar el resultado de una operación con una clase nativa de este como lo es el Assert, esta clase tiene varios miembros como lo son el IsTrue, AreEqual entre otros los mismos que devolverán una respuesta Booleana y permitirán saber si la operación realizada concluyo exitosamente.

NUnit es ampliamente usado en el desarrollo de software con metodologías agiles debido a que permite implementar pruebas unitarias aplicadas con TDD (Test DrivenDevelopment) reduciendo así el riesgo de errores en las etapas de despliegue y publicación de aplicaciones, si bien este componente es usado para TDD en el caso del servicio no se podrá aplicar este concepto ya que se necesitan escenarios variables en la aplicación de estas pruebas y en TDD utilizar escenarios variables y bases de datos no es recomendable por lo que se optó por usarlo para simular procesos a nivel de lógica de negocio y así poderlas ejecutar automáticamente mejorando el desarrollo de pruebas.

4.2. Pruebas del servicio web

Para este proceso de pruebas se copiaran los resultados obtenidos con la ejecución de las pruebas automáticas mediante la utilización de NUnit.

Tabla 5. Iteración1 servicio web

Operación de Prueba	Resultado	Error	Observación
Existencia de Estaciones	Fallida	No Existe Cadena de conexión	Verificar la existencia de la cadena de Conexión
Consulta de Estaciones	Fallida		no puede proceder por el punto anterior
Actualización de Estaciones	Fallida		no puede proceder por el punto anterior
Consulta de Movimientos	Fallida		no puede proceder por el punto anterior
Inserción de Conexión	Fallida		no puede proceder por el punto anterior
Inserción de Movimientos	Fallida		no puede proceder por el punto anterior

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Los resultados obtenidos en esta iteración fallaron debido a que la cadena de conexión no estaba debidamente configurada para la base de datos.

Tabla 6. Iteración 2 servicio web

Operación de Prueba	Resultado	Error	Observación
Existencia de Estaciones	Exitosa		
Consulta de Estaciones	Exitosa		
Actualización de Estaciones	Fallida	Error en el Cast al tipo List	Transformar al tipoIEnumerable
Consulta de Movimientos	Fallida	Error en el Cast al tipo List	Transformar al tipoIEnumerable
Inserción de Conexión	Fallida	No Existe la clave foránea	Insertar Los Catálogos primero
Inserción de Movimientos	Fallida	No Existe la clave foránea	Insertar Los Catálogos primero

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Se configuró la cadena de conexión, pero se obtuvo un error en la conversión del tipo de datos de las entidades hacia el front del servicio.

Tabla 7. Iteración 3 servicio web

Operación de Prueba	Resultado	Error	Observación
Existencia de Estaciones	Exitosa		
Consulta de Estaciones	Exitosa		
Actualización de Estaciones	Fallida	No Existe el registro	comprobar que el registro existe
Consulta de Movimientos	Exitosa		
Inserción de Conexión	Exitosa		
Inserción de Movimientos	Exitosa		

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Este error se produjo por no tener datos consistentes en la base de datos.

Tabla 8. Iteración 4 servicio web

Operación de Prueba	Resultado
Existencia de Estaciones	Exitosa
Consulta de Estaciones	Exitosa
Actualización de Estaciones	Exitosa
Consulta de Movimientos	Exitosa
Inserción de Conexión	Exitosa
Inserción de Movimientos	Exitosa

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Todas las pruebas resultaron exitosas.

4.3. Pruebas del cliente estación de control

Para la ejecución de estas pruebas no se tiene un proceso automatizado ya que se tiene interacción con periféricos externos, se las realizaron tomando encuentra criterios técnicos y observables.

Tabla 9. Iteración 1 cliente estación de control

Operación de Prueba	Resultado	Error	Observación
Conexión Kinect	Fallida	El dato Kinect es nulo	Iniciar Kinect
Lectura de datos Kinect	Fallida		no puede proceder por el punto anterior
Carga de DatosEsqueletizacion	Fallida		no puede proceder por el punto anterior
Transformación de coordenadas	Fallida		no puede proceder por el punto anterior
Graficacion de Puntos	Fallida		no puede proceder por el punto anterior
Actualización de la Vista	Fallida		no puede proceder por el punto anterior
Calculo del Centro de Gravedad	Fallida		no puede proceder por el punto anterior

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Todas las pruebas fallaron ya que la operación principal en este caso la lectura del Kinect no se pudo realizar.

Tabla 10. Iteración 2 cliente estación de control

Operación de Prueba	Resultado	Error	Observación
Conexión Kinect	Exitosa		
Lectura de datos Kinect	Exitosa		
Carga de DatosEsqueletizacion	Exitosa		
Transformación de coordenadas	Fallida	Falla en el Cast de Variables	Cambiar el Tipo de Datos
Graficacion de Puntos	Fallida	Falla en el Cast de Variables	Iniciar Kinect
Actualización de la Vista	Fallida	sin datos	
Calculo del Centro de Gravedad	Fallida	sin datos	

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Se obtuvo una falla ya que no se pudo convertir directamente las variables devueltas por el Kinect en coordenadas cartesianas.

Tabla 11. Iteración 3 cliente estación de control

Operación de Prueba	Resultado	Error	Observación
Conexión Kinect	Exitosa		
Lectura de datos Kinect	Exitosa		
Carga de DatosEsqueletizacion	Exitosa		
Transformación de coordenadas	Exitosa		
Graficacion de Puntos	Exitosa		
Actualización de la Vista	Fallida	sin datos en el modelo	carga de vector de coordenadas
Calculo del Centro de Gravedad	Fallida	Sin datos para el calculo	carga de vector de coordenadas

Elaborado por: (Castro, Slin & Jiménez, Fernando)

No se inicializo el vector de coordenadas antes de pasar el modelo a la vista produciendo un error tipo NullReference.

Tabla 12. Iteración 4 cliente estación de control

Operación de Prueba	Resultado
Conexión Kinect	Exitosa
Lectura de datos Kinect	Exitosa
Carga de DatosEsqueletizacion	Exitosa
Transformación de coordenadas	Exitosa
Graficacion de Puntos	Exitosa
Actualización de la Vista	Exitosa
Calculo del Centro de Gravedad	Exitosa

Elaborado por: (Castro, Slin & Jiménez, Fernando)

4.4. Pruebas del cliente estación de trabajo

Para las pruebas de funcionamiento de la estación de trabajo se verificó los datos enviados con los movimientos del brazo.

Tabla 13. Iteración 1 cliente estación de trabajo

Operación de Prueba	Resultado	Error	Observación
Apertura de Puerto serial	Exitosa		
Envió de datos por puerto	Exitosa		
Creación de Comandos	Fallida	No se Reconoce el Comando	verificación de comandos
Encolamiento de Datos	Fallida		no puede proceder por el punto anterior
Inicialización del brazo	Fallida		no puede proceder por el punto anterior

Elaborado por: (Castro, Slin & Jiménez, Fernando)

No se enviaron de forma correcta los comandos por puerto serial recibiendo un error de reconocimiento de comandos.

Tabla 14. Iteración 2 cliente estación de trabajo

Operación de Prueba	Resultado	Error	Observación
Apertura de Puerto serial	Exitosa		
Envió de datos por puerto	Exitosa		
Creación de Comandos	Exitosa		
Encolamiento de Datos	Fallida	Sobrecarga del Buffer de Mem	limpiar el Buffer de datos
Inicialización del brazo	Fallida		no puede proceder por el punto anterior

Elaborado por: (Castro, Slin & Jiménez, Fernando)

En esta iteración debido al envío masivo de comandos se sobrecargo el buffer del puerto serial asignado por el IDE.

Tabla 15. Iteración 3 cliente estación de trabajo

Operación de Prueba	Resultado
Apertura de Puerto serial	Exitosa
Envió de datos por puerto	Exitosa
Creación de Comandos	Exitosa
Encolamiento de Datos	Exitosa
Inicialización del brazo	Exitosa

Elaborado por: (Castro, Slin & Jiménez, Fernando)

4.5. Pruebas generales de integración

Estas pruebas se realizaran mediante el consumo del servicio web desde un cliente de prueba y luego desde los clientes reales.

Tabla 16. Iteración 1 pruebas generales de integración

Operación de Prueba	Resultado	Error	Observación
Activación de Estación	Fallida	No se lee el servicio	Verificar el WSDL
Consulta de Estación	Fallida		no puede proceder por el punto anterior
Consulta de estaciones	Fallida		no puede proceder por el punto anterior
Envío de Movimientos	Fallida		no puede proceder por el punto anterior
Lectura de Movimientos	Fallida		no puede proceder por el punto anterior

Elaborado por: (Castro, Slin & Jiménez, Fernando)

No se pudo conectar al servicio debido a que no se configuró el tipo de protocolo adecuado entre los clientes y el servicio.

Tabla 17. Iteración 2 pruebas generales de integración

Operación de Prueba	Resultado	Error	Observación
Activación de Estación	Exitosa		
Consulta de Estación	Exitosa		
Consulta de estaciones	Exitosa		
Envío de Movimientos	Fallida	Clave Duplicada	
Lectura de Movimientos	Fallida		no puede proceder por el punto anterior

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Se produjo un error por inconsistencia de datos en la base.

Tabla 18. Iteración 3 pruebas generales de integración

Operación de Prueba	Resultado
Activación de Estación	Exitosa
Consulta de Estación	Exitosa
Consulta de estaciones	Exitosa
envío de Movimientos	Exitosa
Lectura de Movimientos	Exitosa

Elaborado por: (Castro, Slin & Jiménez, Fernando)

4.6. Pruebas de carga de datos del servicio web

Como culminación de la etapa de pruebas se han planteado escenarios de carga para comprobar la efectividad del servicio .

Escenario de carga con insercion de 1000 datos en el servicio web.

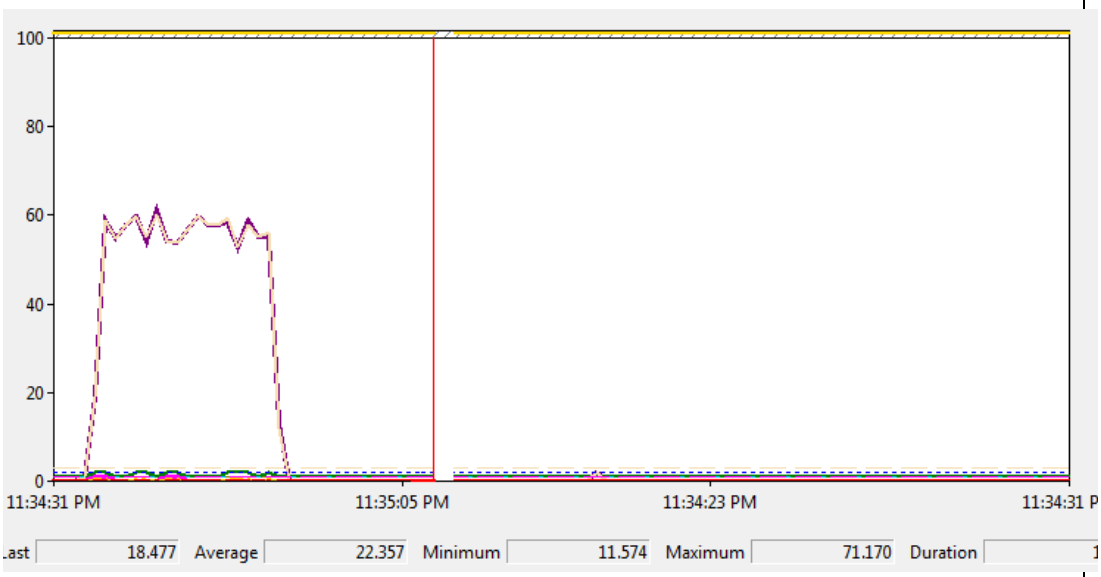


Figura 76. Prueba de carga

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Se ha logrado insertar exitosamente 1000 registros en un tiempo de 25s con un uso del 60% de capacidad de procesamiento del servidor con una media de 1 registro cada 25 ms.

Insercion de datos con una insercion de 10000 datos .

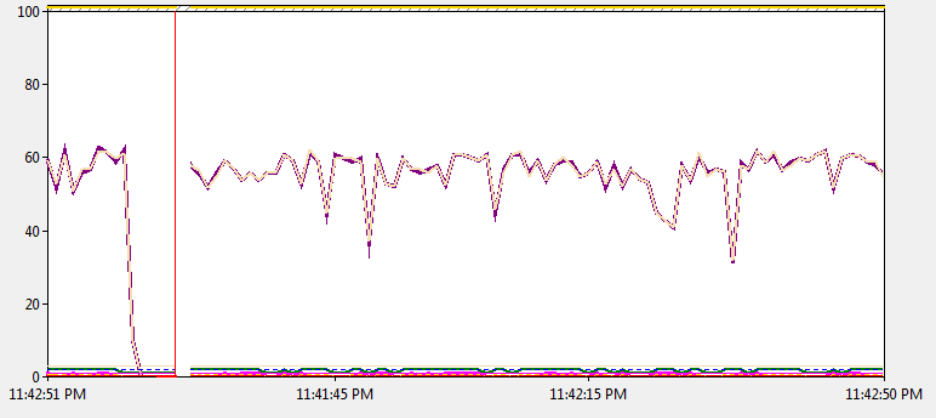


Figura 77. Prueba de carga

Elaborado por: (Castro, Slin & Jiménez, Fernando)

La segunda prueba se realizó con la inserción de 10000 registros automáticamente desde el cliente de pruebas obteniendo un resultado total de inserción de datos en 250 ms.

Además, de las pruebas de carga se tiene un 100% de respuesta exitosa en 12 269 llamadas.

\\DELL-PC		
Processor Information	_Total	
% Processor Time	13.773	
ServiceModelService 4.0.0.0	ControlManipulador@|ControlManipulador|ControlManipulador.svc	
Calls		12,269.000
Calls Duration		0.000
Calls Failed		0.000
Calls Failed Per Second		0.000
Calls Faulted		0.000
Calls Faulted Per Second		0.000
Calls Outstanding		0.000
Calls Per Second		0.000
Instances		1.000
Instances Created Per Second		0.000
Percent Of Max Concurrent Calls		0.000
Percent Of Max Concurrent Instances		0.000

Figura 78. Resultado monitoreo

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Se realizó una prueba de carga de consulta de los 10000 datos insertados anteriormenteen donde se realizo en 5s .

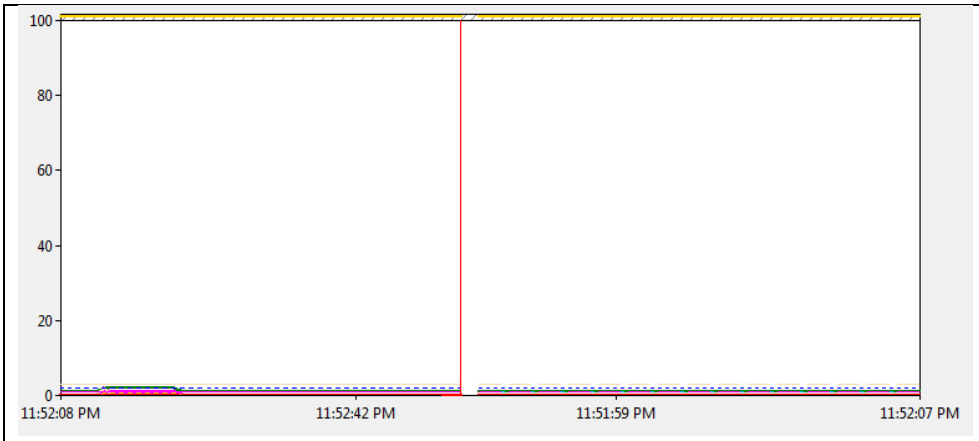


Figura 79. Prueba de carga

Elaborado por: (Castro, Slin & Jiménez, Fernando)

4.7. Resultados de interfaz

Una vez que se ha culminado el proceso de pruebas de procesos, se ha determinado la prueba de la interfaz de control para comprobar la efectividad del usuario al momento de controlar el sistema de manera intuitiva, el usuario puede maniobrar de manera virtual la pantalla y seleccionar la acción presionando de manera virtual el botón enfocado, una vez seleccionado se presenta la pantalla informativa en donde se marca la selección de la acción.

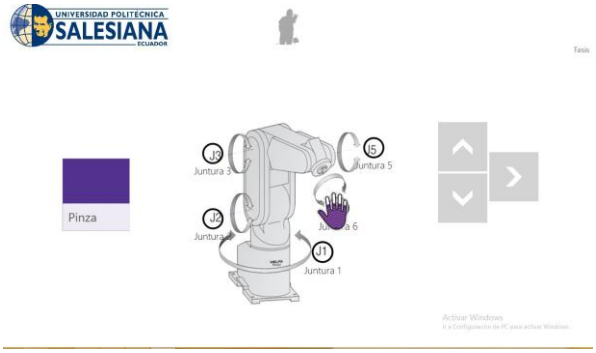


Figura 80. Selección de juntura

Elaborado por: (Castro, Slin & Jiménez, Fernando)

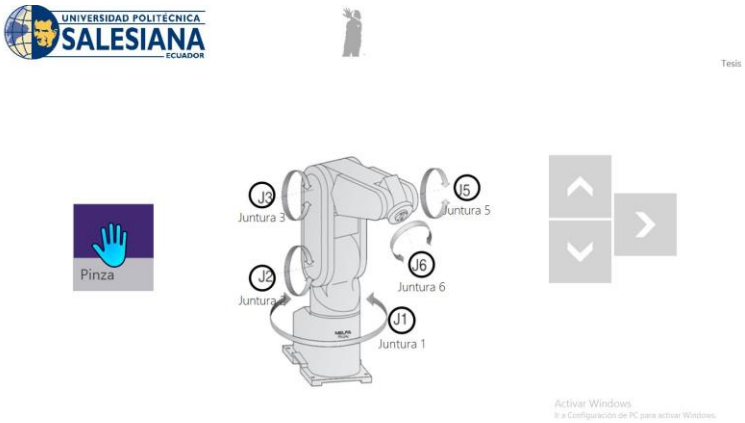


Figura 81. Selección de acción de pinza

Elaborado por: (Castro, Slin & Jiménez, Fernando)

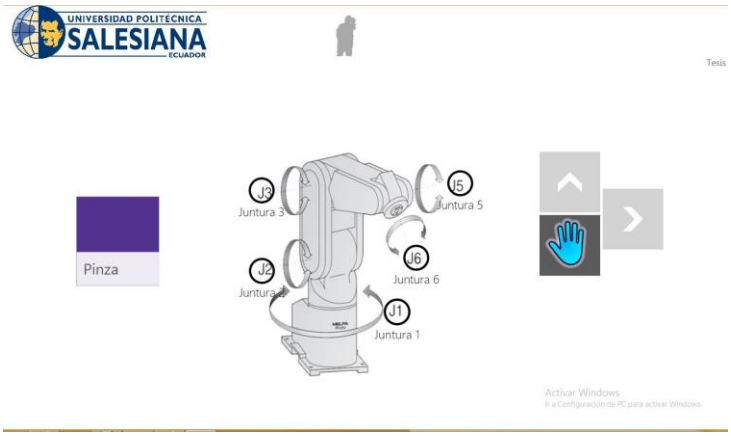


Figura 82. Selección ángulo de movimiento

Elaborado por: (Castro, Slin & Jiménez, Fernando)

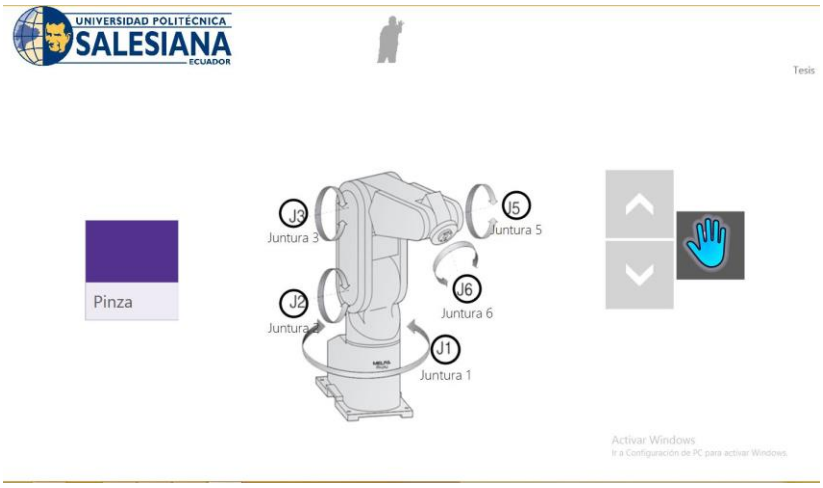


Figura 83. Selección envío de dato

Elaborado por: (Castro, Slin & Jiménez, Fernando)

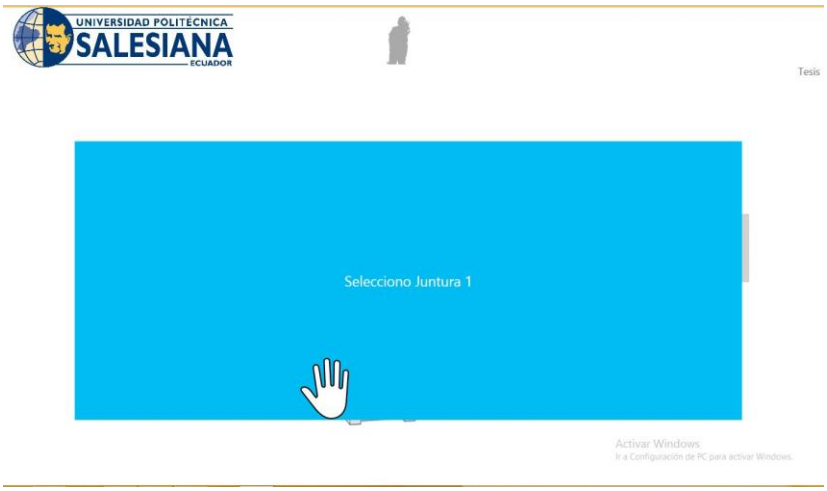


Figura 84. Presentación de acción seleccionada

Elaborado por: (Castro, Slin & Jiménez, Fernando)

CONCLUSIONES

El uso de servicios web para la interoperabilidad a través de Internet es la mejor alternativa cuando no se posee una IP fija.

La comunicación más recomendable es REST por el enfoque de velocidad en la comunicación.

REST es más liviano al no contener una envoltura XML utilizando transporte por credenciales o certificados aumentando el tiempo de transporte y garantizando la integridad y confidencialidad de la información.

La abstracción de código en la lógica de negocio logra brindar al aplicativo mejora de tiempo de depuración y mantenimiento más rápido para en la corrección de bugs.

Al manejar un estándar de programación a nivel de clases y componentes en la aplicación se reduce el tiempo de desarrollo del sistema.

La aplicación de pruebas unitarias en el ciclo de desarrollo del sistema permite al desarrollador tener la certeza de que el código utilizado en la solución es confiable y reduce el tiempo de pruebas de funcionamiento.

Aunque el uso de XAML está descontinuado por parte de Microsoft es la mejor tecnología aplicada con patrones de diseño adecuados funciona de manera efectiva y confiable

XML permite un desarrollo más amigable al usuario que los winforms o Applets.

La continua actualización de frameworks de desarrollo con .NET permite hacer un seguimiento de la evolución en el lenguaje y las ventajas que presentan con respecto a sus predecesores

Para aumentar la velocidad de acceso en la capa de persistencia y permitir una facilidad en el manejo de datos se utiliza EntityFramework.

Usar clases asíncronas en la estación de trabajo para la comunicación a bajo nivel con puerto serial mejora el tiempo de ejecución.

El periférico Kinect es altamente confiable como dispositivo de interfaz humana al tener un alto grado de resolución y un excelente tiempo de respuesta.

El uso del periférico Kinect puede cambiar la forma de percibir aplicaciones al no tener la necesidad de manipulación física por parte del usuario.

RECOMENDACIONES

Usar logs de errores a nivel de lógica de negocio para identificar los errores de manera eficaz y oportuna en el ciclo de funcionamiento de la aplicación.

Usar TDD para el desarrollo de módulos con complejidad de funcionamiento para tener conciencia de que estos están exitosamente desarrollados y evitar redundancia en las pruebas.

Usar LINQ y expresiones LAMBDA para la manipulación de datos dentro de la aplicación para mejorar el rendimiento y la velocidad de procesamiento de datos.

Utilizar la distribución del software por módulos para hacer un mantenimiento y despliegue ágil de la aplicación.

Utilizar el httpBasicbinding para la etapa de pruebas debido a la simplicidad en la configuración de los clientes del servicio.

Usar entidades de negocio para la fácil manipulación de datos, ya que permite el cambio de una manera más limpia que con el uso de variables primitivas.

Usar el Framework de Kinect más actualizado por las facilidades de desarrollo implementadas, acortando el tiempo de desarrollo y la simplicidad de uso.

Usar el periférico Kinect en ambientes amplios e iluminados para salvaguardar la integridad física del usuario.

Delimitar un espacio de trabajo seguro para el manipulador ya que en modo de ejecución puede romper elementos de la mesa de trabajo.

Verificar el estado del manipulador antes de empezar la comunicación con el servidor web.

Configurar la velocidad de movimiento del brazo y los parámetros de comunicación antes de empezar la manipulación remota de este.

LISTA DE REFERENCIAS

Alonso, J. A. (2004). *Tecnologías de la Información y de la Comunicación*. Ra-Ma . Obtenido de http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm.

Arkin, R. (1998). *Behavior-Based Robotics*. The MIT Press.

Articulaciones Robóticas. (s.f.). Obtenido de Página con la información de las articulaciones de los robots: http://www.robotmatrix.org/ArticulateRobotic.htm

Asimov, I. (1950). *Yo, robot*.

Barrientos, A. (2007). *Fundamentos de la Robótica*. España: Editorial McGraw Hill.

Baturone, A. O. (2001). *Robótica: manipuladores y robots móviles*. Marcombo.

Blogcdn. (s.f.). *http://www.blogcdn.com/es.engadget.com/media/2010/06/06-14-10kinectport.jpg*. Obtenido de http://www.blogcdn.com/es.engadget.com/media/2010/06/06-14-10kinectport.jpg.

Braunl, T. (2006). *Embedded Robotics,*. Australia: Editorial Springer.

Campion, G. B.-N. (1996). *Structural properties and classifications of kinematics and dynamics models of wheeled mobile robots*. IEEE Transactions on Robotics and Automation.

Castro, Slin & Jiménez, Fernando. (s.f.). Los Autores.

Chamorro, M. (1998). *CONTROL DE UN BRAZO ROBÓTICO PUMA APLICADO A SOLDADURA DE GEOMETRÍAS COMPLEJAS TRIDIMENSIONALES*.

Cox, I. (1991). *Blanche – An experiment in guidance and navegation of an autonomous robot vehicle*. IEEE, Trans RA.

damowski, J. S. (1990). *Desenvolvimento de un robo móvel*. San Pablo: Escola Politecnica de la Universidade de Sao Paulo.

Detección de Imágenes Dispositivo Kinect. (s.f.). Obtenido de División de Microsoft donde se investiga los diversos temas de ciencia de equipo y problemas: http://research.microsoft.com/en-us/

Engelberger . (1980). *Tratado de Robótica*.

Estación Automatizada. (s.f.). Obtenido de Estación automatizada para representar de manera didáctica el proceso de ensamble que se realiza en la industria: http://www.monografias.com/trabajos16/estacion-robotica/estacion-robotica.shtml

Festo. (s.f.). *Estación de Trabajo Festo*. Obtenido de Festo Didactic: http://www.festo-didactic.com/ov3/media/customers/1100/rv_2sdb.jpg

González Jiménez, J. y. (1996). *Estimación de la posición de un robot móvil*. España.

Hardware del Dispositivo Kinect. (s.f.). Obtenido de Partes y Equipos de Tecnología: http://www.ifix.com/

IBM. (s.f.). *WebServices*. Obtenido de www.ibm.com/developerworks/ssa/webservices/newto/service.html.

IEEE. (2013). Grados de Libertad de un Robot Manipulador. *Robóticas, Articulaciones*.

J. C. Alexander, J. H. (1989). *On the Kinematics of Wheeled Mobile Robots*. The International Journal of Robotics Research.

Jones, J. a. (1993). *Mobile Robots. Inspiration to Implementation*. A.K. Peters, Ltd. USA.

KarelCapek. (1920). *Los robots universales de Rossum*.

L. Armesto, L. G. (2003). *Multi-rate line tracking for mobile robots based on artificial vision and odometry*.

Lenguajes de programación de los Robots. (s.f.). Obtenido de http://www.monografias.com/trabajos3/progrob/progrob.shtml.

Liñan, J. (1995). *Apuntes Control Robots*.

Lozano-Pérez, T. (s.f.). *Foreword: Mobile Robot and Robotics*. 1990: Springer-Verlag.

Martínez Coral, M. P., & Freire Muñóz, A. S. (2012). *Diseño y construcción de un API (Interfaz de Programación de Aplicaciones) para la Plataforma .NET que permita controlar el brazo robótico Mitsubishi RV-2aj a través del Puerto Serial*.

Microsoft. (2013). *MSDN*. Obtenido de MSDN.

Microsoft. (s.f.). *Developer Network*. Obtenido de http://msdn.microsoft.com/es-es/library/system.io.ports.serialport(v=vs.110).aspx

Microsoft. (s.f.). *Dispositivo Kinect*. Obtenido de http://upload.wikimedia.org/wikipedia/commons/6/67/Xbox-360-Kinect-Standalone.png

Mitsubishi. (s.f.). *RV-1A/2AJ Series INSTRUCTION MANUAL*.

Ollero Baturone, A. (2001). *Robótica Manipuladores y Robots Móviles*. Barcelona, España.

Organización Internacional de Estándares. (2004). Robótica.

Parte Mecánica del Dispositivo Kinect. (s.f.). Obtenido de Imagen donde se describe la parte mecánica del dispositivo Kinect: http://www.geek.com/wp-content/uploads/2010/11/Kinect_motor.jpg

Platea . (s.f.). Obtenido de http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm.

RIA. (s.f.). *RoboticIndustryAssociation*. Obtenido de RoboticIndustryAssociation.

Robotic Industry Association. (s.f.). *RIA*.

Robótica y aplicaciones. (2014). Obtenido de http://www.monografias.com/trabajos10/robap/robap.shtml.

Robots industriales. (2014). Obtenido de http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm.

The World Wide Web Consortium. (s.f.). Obtenido de El World Wide Web Consortium (W3C) es una comunidad internacional que desarrolla estándares que aseguran el crecimiento de la Web a largo plazo: http://www.w3.org

Tipantuña, J. (2003). *Diseño e impplementacion de un sistema de seguridad en tiempo real monitoreado por internet*.

UPM. (2004). *Qué es XML Schema*. Obtenido de http://mat21.etsii.upm.es/mbs/mechxml/que_es_xml_schema.htm.

ANEXOS

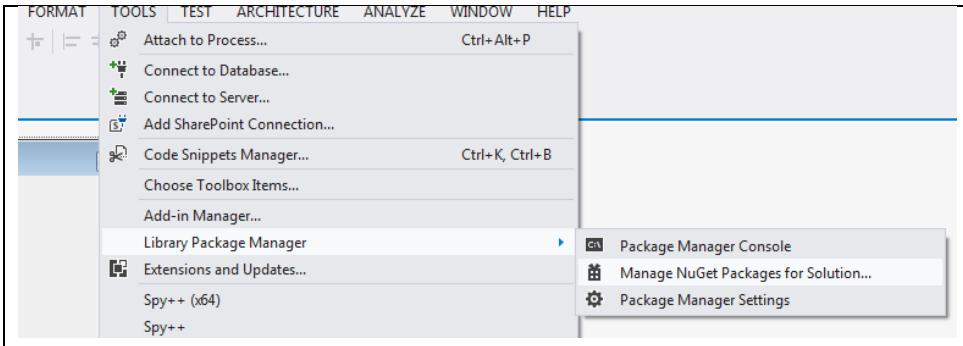
Anexo 1. Configuración de nUnit

Al ser un componente externo al IDE utilizado se requiere que previamente se haya ejecutado el proceso de instalación, permitiendo posteriormente mostrar la integración de este Framework con VisualStudio2012, tomando en cuenta que para la instalación y configuración sea exitosa se debe realizar una prueba de conectividad a internet.

Se lo puede configurar de 2 maneras, mediante la consola de componentes de VS2012 o a través del asistente.

A través del asistente:

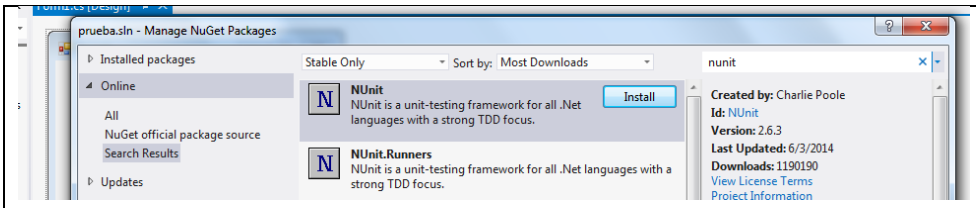
Se Selecciona menu>Tools > Library Package Manager> ManageNuggetPackagesforSolution.



Manejador paquetes NuGet

Elaborado por: (Castro, Slin & Jiménez, Fernando)

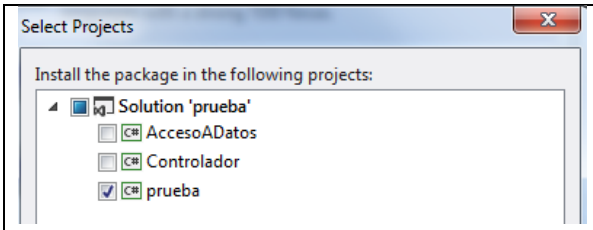
Se despliega una ventana como la siguiente, se busca el paquete NUnit y se presiona Install.



Ubicación NUnit en NuGet

Elaborado por: (Castro, Slin & Jiménez, Fernando)

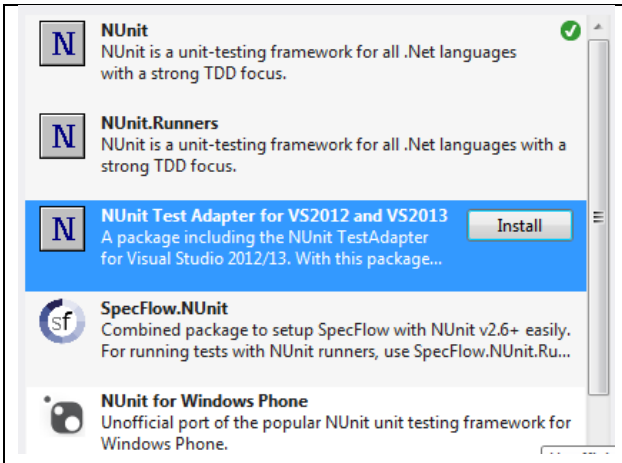
Se selecciona el proyecto en el que se instalará este paquete, en este caso se seleccionará el proyecto prueba , se selecciona solo el proyecto de pruebas ya que el instalarlo no produce un problema en el funcionamiento pero si un alto impacto en el rendimiento debido a que en los proyectos en los que se instale este paquete se requerirá que se levanten las librerias al cargar el proyecto.



Selección proyecto NUnit

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Una vez realizado el proceso de instalación el paquete NUnit en el proyecto de pruebas pero aún hace falta instalar un adaptador para que estas pruebas puedan ser ejecutadas desde el mismo IDE.



Selección adaptador NUnit

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Una vez instalado estos dos complementos se puede empezar a realizar las pruebas unitarias, para esto se debe crear una nueva clase y referenciar a la librería NUnit.Framework además de colocar el decorador de TestFixture a la clase de pruebas.

Para realizar un método de pruebas se le coloca el decorador de Test y opcionalmente la categoría.

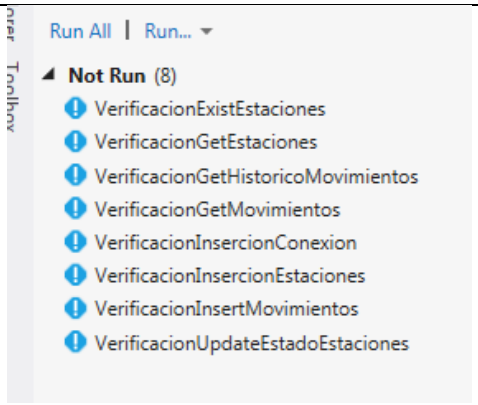
```
using ControlManipulador.Entities;
using NUnit.Framework;

namespace ControlManipulador.Test
{
    [TestFixture]
    public class TestEstaciones
    {
        [Test, Category("CRUD Estaciones")]
        public void VerificacionExistEstaciones()
```

Creación clase prueba NUnit

Elaborado por: (Castro, Slin & Jiménez, Fernando)

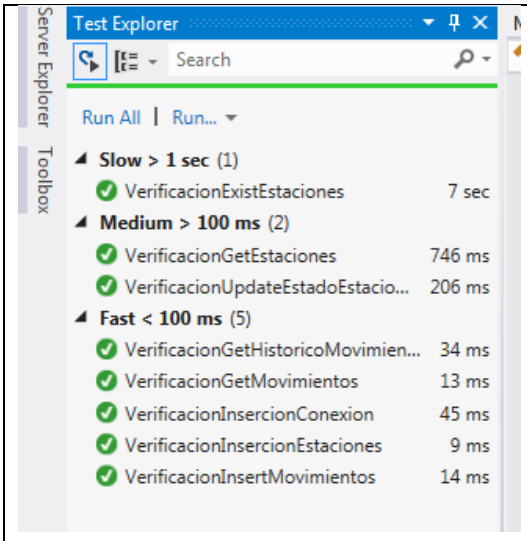
Al compilar el proyecto en la ventana de test Explorer aparecerá la prueba desarrollada.



Listado de pruebas

Elaborado por: (Castro, Slin & Jiménez, Fernando)

Y al ejecutarlas se mostraran si fueron ejecutadas exitosamente o no, además de un detalle de la duración de cada operación permitiendo hacer una prueba de velocidad de ejecución.



Resultados pruebas

Elaborado por: (Castro, Slin & Jiménez, Fernando)