Reconocimiento de Posición e Inclinación de la Mano para Control de Movimiento de una Plataforma Delta con Kinect

Por
Edgar Andrés Gutiérrez Cáceres

Una tesis presentada al Departamento de Ingeniería Electrónica de la
Pontificia Universidad Javeriana,
en el cumplimiento parcial de los requisitos del grado de:

Magister en Ingeniería Electrónica
Énfasis en Sistemas Digitales

Maestría En Ingeniería Electrónica
Departamento de Ingeniería Electrónica
Pontificia Universidad Javeriana
Noviembre, 2015

Pontificia Universidad Javeriana

Como asesor del estudiante candidato para graduarse, he leído y revisado la tesis de Edgar Andrés Gutiérrez Cáceres en su forma final y he encontrado que en primer lugar, su contenido, los esfuerzos de investigación, los objetivos, el formato, las citas y las referencias son coherentes y aceptables dentro de los requisitos en el área de Robótica y Sistemas digitales, y en segundo lugar, el manuscrito final es satisfactorio para ser revisado por el comité ejecutivo y está listo para su presentación con el fin de obtener el grado de Maestría en Ingeniería Electrónica de la Pontificia Universidad Javeriana, Bogotá, Colombia.

_____

Ing. Julián David Colorado Montaño
Docente Maestría Ingeniería Electrónica
Pontificia Universidad Javeriana

ABSTRACT

This thesis aims to design and implement a prototype of a robot delta is manipulated from the recognition of position and inclination of the hand to the generation of movement thereof. In addition, it presents the goal of cinematic study of robot for geometric analysis and initial design parameters delivering CAD platform for further implementation.

Finally, deliver a graphical user interface which can control the movements of the robot from a computer through Kinect.

In this scope it is proposed as handling the delta platform, by detecting the pose coordinates and hand of an operator. As imaging system, the Microsoft Kinect® equipment is used. The algorithmic development for handling the data provided by the platform, is done by the Free Software Processing program, initially created for graphic design and systems that have boomed for development projects Kinect®. As drive means using Arduino is formulated as an element of Open Hardware.

On the other hand a proposal for designing a Solidworks delta robot is delivered. It should be noted that work specifically focuses on the design of the platform, image processing for detecting hand and no parameter like path planning platform strategy or some control of it.

AGRADECIMIENTOS

A dios, por permitirme culminar esta gran etapa de mi vida, por cuidar de mí y de mi familia.

A la *Santísima Virgen de Monguí*, la cual siempre me brindo de su gran protección en los continuos viajes y trayectos manteniéndome sano y salvo hasta el seno de mi hogar.

A mi esposa *Ana María*, por su amor y apoyo incondicional, los cuales han sido mi mayor motivación para el esfuerzo constante de esta familia que se está formando.

A mi hijo *Alberto* que con su alegría llena este hogar cada día y me muestra cada día como ser un mejor padre y ejemplo.

Sofía hija aunque eres muy pequeña como para entender esto, quiero que sepas que tus dos añitos y medio, ha significado más para mí que toda mi vida entera, gracias a ti he comprendido lo importante que es apreciar las cosas que tu estas aprendiendo, *siempre estaré ahí para ti* ☺.

A mi padre *Edgar David* y Mi Madre *Luz Herminda*, gracias por la formación que siempre me han dado como una persona honrada y que siempre piensa en la humildad y sencillez de nuestras acciones y la forma de tratar a la gente.

A mi hermana *Erly Johanna*, que la Bacteriología siempre te de alegrías y te lleve al éxito que te mereces.

A mis Amigos y colegas, *Camilo, Luis, Fabián*, gracias por el apoyo en mi carrera.

A la *Universidad Santo Tomas*, por permitirme y apoyarme laboral y académicamente, mediante la ayuda de perfeccionamiento Docente.

A la *Pontificia Universidad Javeriana*, En especial a los Ingenieros *Cesar Niño, Enrique González, Francisco Viveros, Carlos Parra, Pedro Vizcaya y Diego Patiño*, Todos ustedes aportaron cosas muy valiosas en mi formación, estoy muy agradecido por el conocimiento que me aportaron y las nuevas proyecciones que ahora veo a partir de estas.

Finalmente y especialmente, al ingeniero *Julián Colorado Montaño*. Ingeniero he conocido un maestro en el área de Robótica, se ha convertido en un referente muy importante para mi vida académica, gracias por su apoyo en mi formación de Maestría y su asesoría.

A todos ustedes y los que seguramente me faltarían por nombrar gracias simplemente gracias.

Contenido

1 CAPITULO I: INTRODUCCIÓN 14

1.1 MOTIVACIÓN 14

1.2 OBJETIVOS 15

1.2.1 Objetivo General 15

1.2.2 Objetivos Específicos 15

1.3 METAS Y ALCANCES 15

1.4 ESQUEMA DEL DOCUMENTO 16

2. CAPITULO II: INFORMACION GENERAL ROBOTS PARALELOS 19

2.1 VISION GENERAL 19

2.2 TIPOS DE ROBOTS PARALELOS 20

2.2.1 MANIPULADORES PLANOS CON TRES GRADOS DE LIBERTAD ACCIONADOS POR MEDIO DE ACTUADORES LINEALES O GIRATORIOS 23

2.2.2 MANIPULADORES ESPACIALES CON TRES GRADOS DE LIBERTAD ACCIONADOS POR MEDIO DE ACTUADORES GIRATORIOS 23

2.2.3 MANIPULADORES ESPACIALES CON TRES GRADOS DE LIBERTAD ACCIONADOS POR MEDIO DE ACTUADORES LINEALES 24

2.3 ROBOT DELTA 25

2.4 CONCLUSION 26

3. CAPITULO III: DISEÑO DE LA PLATAFORMA DELTA 28

3.1 VISION GENERAL 28

3.2 Diseño de Piezas en Solidworks 2015 28

3.2.1 BASE FIJA, SOPORTE MOTORES Y BASE MOVIL 28

3.2.2 SERVOMOTOR HS-311 29

3.2.3 EFECTOR O PINZA FINAL 30

3.2.4 BARRA MUSLO, PIERNA Y UNION BASE MOVIL 30

3.2.5 ROD END, TORNILLO Y TUERCA 30

3.2.6 PLATAFORMA DELTA DISEÑADA 31

3.3 CONCLUSION 33

4. CAPITULO IV: MODELADO DE LA PLATAFORMA DELTA 35

4.1 VISION GENERAL 35

4.2 GEOMETRIA DEL ROBOT 35

4.3 CINEMÁTICA INVERSA 40

4.3.1 VARIABLE DE ARTICULACIÓN $\boldsymbol{\theta 11}$ 41

4.3.2 VARIABLE DE ARTICULACIÓN $\boldsymbol{\theta 12}$ 41
4.3.3 VARIABLE DE ARTICULACIÓN $\boldsymbol{\theta 13}$ 41
4.4 CINEMÁTICA DIRECTA 42
4.5 MATRIZ JACOBIANA 45
4.6 SINGULARIDADES DE LA CINEMÁTICA INVERSA 47
4.7 SINGULARIDADES DE LA CINEMÁTICA DIRECTA 48
4.8 COMBINACIÓN DE SINGULARIDAD 48
4.9 DESTREZA DEL ROBOT DELTA 48
4.10 ESPACIO DE TRABAJO DEL ROBOT DELTA 49
4.11 CONCLUSION 52
5. CAPITULO V: IDENTIFICACION DE LA MANO TRATAMIENTO DE IMAGEN DEL KINECT 54
5.1 VISION GENERAL 54
5.2 KINECT 54
5.2.1 SISTEMA DE RASTREO 56
5.2.2 RECONOCIMIENTO DE VOZ 57
5.2.3 MOTOR 57
5.2.4 PRINCIPIO DE FUNCIONAMIENTO KINECT 57
5.2.5 DATOS DE PROFUNDIDAD 58
5.3 PROCESSING 60
5.4 ANALISIS CINEMATICO INVERSO A NIVEL DE SOFTWARE 61
5.5 Diagrama de flujo general del algoritmo 63
5.6 CLASS DELTAROBOT 64
5.7 CLASS DELTALEG 70
5.8 ROBOT DELTA CONTROLADO POR KINECT 72
5.9 CONTROL DEL GRIPPER 77
5.10 CONCLUSION 80
6. CAPITULO V: DISEÑO SISTEMA ELECTRONICO 82
6.1 VISION GENERAL 82
6.2 ARDUINO 82
6.3 LENGUAJE DE PROGRAMACION 82
6.3.1 EQUIPO ARDUINO UNO VERSION R3 83
6.4 DISEÑO Y CONSTRUCCION CIRCUITO ELECTRONICO ROBOT DELTA 84

6.5 ALGORITMO IMPLEMENTADO ARDUINO 87
6.6 CONCLUSION 88
7. CAPITULO VII: RESULTADOS 90
7.1 VISION GENERAL 90
7.2 MEDICION DE DESPLAZAMIENTO, VELOCIDAD Y ACELERACION DEL SISTEMA PRUEBAS EN SOFTWARE 90
7.3 PROTOCOLO DE PRUEBAS 96
7.3.1 MOVIMIENTO VERTICAL 96
7.3.2 MOVIMIENTO HORIZONTAL 96
7.3.3 MOVIMIENTO CIRCULAR 97
7.4 MEDICION DE CORRIENTE Y POTENCIA 98
7.5 CONCLUSION 99
8. CAPITULO VIII: CONCLUSIONES GENERALES Y TRABAJO FUTURO 101
8.1 RESEÑA 101
8.1.1 CONSIDERACIONES 101
8.1.2 CONCLUSION GENERAL 101
8.2 TRABAJO FUTURO 102
8.3 PUBLICACIONES EN RELACION CON ESTE TRABAJO DE TESIS 102
9. CAPITULO IX: REFERENCIAS 104
APENDICE A: DISEÑO DELTA SOLIDWORKS 109
1. BASE FIJA 109
2. SOPORTE MOTORES 109
3. BASE MOVIL 109
4. SERVOMOTOR Y EJE 110
5. PINZA 110
6. MUSLO, PIERNA Y UNION BASE MOVIL 110
7. ROD END, TORNILLO Y TUERCA 111
APENDICE B: ALGORITMO RECONOCIMIENTO 113
1. CODIGO CLASS DELTAROBOT 113
2. PUBLIC VOID MOVETO 114
3. PUBLIC VOID DRAW 114
4. VOID DRAWEFFECTOR 115
5. PUBLIC VOID UPDATEGRIP 116

6. CLASS DELTALEG....116
7. VOID MOVETO....116
8. VOID GETWORLDCOORDINATES....117
9. PUBLIC VOID DRAW....118
10. SIMULACION ROBOT DELTA MANIPULACION CON EL MOVIMIENTO DEL MOUSE....119
11. PROGRAMA COMPLETO ROBOT DELTA PROCESSING Y KINECT....120
12. PROGRAMA ADICIONAL ENVIO DATOS TARJETA ARDUINO UNO DESDE PROCESSING..124
APENDICE C: ALGORITMO ARDUINO....127
1. CODIGO ARDUINO....127

LISTA DE FIGURAS

Fig. 1. Tipos de Robots paralelos. 14
Fig. 2Esquema General del Sistema 16
Fig. 3 Plataforma de movimiento espacial patentada por J.E.Gwinnett 20
Fig. 4 Robot paralelo patentado por W.L.V. Pollard 20
Fig. 5 Primer prototipo robot paralelo ideado por V.E. Gough 21
Fig. 6 MAST, Multi – Axis Simulation Table 21
Fig. 7Plataforma de Stewart 21
Fig. 8 Simulador de vuelo patentado por K.L Cappel 22
Fig. 9 Simulador de vuelo patentado por K.L Cappel 22
Fig. 10 Diferentes tipos de manipuladores planos con actuadores prismáticos y giratorios 23
Fig. 11 Robots de tres grados de libertad con actuadores giratorios, (a) Delta, (b) Ojo de Águila, (c) Capaman 23
Fig. 12 Robots de tres grados de libertad con actuadores lineales, (a) Orthoglide, (b) Tricept, (c) 3-UPU. 24
Fig. 13 Robots de seis grados de libertad, (a) Active Wrist, (b) Hexaglide, (c) Tri-Scott. 24
Fig. 14 Robot Delta 26
Fig. 15 Base Fija Robot Delta 28
Fig. 16 Base Motor Robot Delta 29
Fig. 17 Base Móvil Robot Delta 29
Fig. 18 Servomotor y conector eje 29
Fig. 19 Pinza o efector del robot delta 30
Fig. 20 Base Muslo, Pierna y Unión 30
Fig. 21 Pinza o efector del robot delta 31
Fig. 22 Robot delta Final 31
Fig. 23 Plataforma Delta Real 32
Fig. 24 Estructura del Robot paralelo RUU 36
Fig. 25 Modelo geométrico del robot Delta 37
Fig. 26 Descripción de los ángulos: izq) Vista Frontal der) Vista lateral. 37
Fig. 27 Forma geométrica de: a) la base fija y b) la plataforma móvil. 39
Fig. 28 Representación vectorial de la primera cadena cinemática. 40
Fig. 29 Cadena cinemática i en las coordenadas $XiYiZi$ 44
Fig. 30 Espacio de Trabajo del Robot Delta, método de Monte Carlo. 50
Fig. 31 Restricciones del ángulo $\theta i3$. 50
Fig. 32 Restricciones del ángulo $\theta i2$. 51
Fig. 33 Diagrama de flujo del programa del espacio de trabajo del robot. 52
Fig. 34 Secciones principales del Sensor Kinect® 54
Fig. 35 Composición interna del Sensor Kinect® 55
Fig. 36 Distancias de detección Kinect modo por defecto y modo Near 55
Fig. 37 Composición interna del Sensor Kinect® 56
Fig. 38 Principios de Kinect® 58
Fig. 39 Barrido espacial Sensor Kinect® 58
Fig. 40 Triangulación de cada punto, entre una imagen patrón y la imagen capturada. 59

Fig. 41 Influencia de la distancia y orientación del sensor en la detección de la imagen. 59
Fig. 42 Regiones de captura de imagen. 59
Fig. 43 Logo de apertura software Processing. 60
Fig. 44 Análisis geométrico Robot delta..... 61
Fig. 45 Diagrama de Flujo 63
Fig. 46 Sección 1 Diagrama de flujo class DeltaRobot 64
Fig. 47 Class definidos para el robot Delta 65
Fig. 48 Dos robots delta, con 5 y 150 patas. 65
Fig. 49 Sección 2 Diagrama de flujo class DeltaRobot 66
Fig. 50 Sección 3 Diagrama de flujo class DeltaRobot 68
Fig. 51 Simulación del efector y del gripper..... 68
Fig. 52 Sección 4 Diagrama de flujo class DeltaRobot 69
Fig. 53 Sección 5 Diagrama de flujo class DeltaRobot 69
Fig. 54 Sección 1 Diagrama de flujo class DeltaLeg..... 70
Fig. 55 Sección 2 Diagrama de flujo class DeltaLeg..... 70
Fig. 56 Sección 3 Diagrama de flujo class DeltaLeg..... 71
Fig. 57 Sección 4 Diagrama de flujo class DeltaLeg..... 71
Fig. 58 Simulación del Robot Delta por medio del ratón 72
Fig. 59 El robot Delta es manipulado por gestos de las manos..... 73
Fig. 60 Sección 1 Programa Principal Robot delta con Kinect 73
Fig. 61 Sección 2 Programa Principal Robot delta con Kinect 74
Fig. 62 Sección 3 Programa Principal Robot delta con Kinect 74
Fig. 63 Origen de la mano y la posición actual de la mano 75
Fig. 64 Sección 4 Programa Principal Robot delta con Kinect 75
Fig. 65 Sección 5 Programa Principal Robot delta con Kinect 76
Fig. 66 Sección 6 Programa Principal Robot delta con Kinect 76
Fig. 67 Sección 7 Programa Principal Robot delta con Kinect 77
Fig. 68 Punto de inicio de la mano 78
Fig. 69 Seguimiento ancho de la Mano 78
Fig. 70 Seguimiento de la inclinación de la mano..... 78
Fig. 71 Sección 8 Programa Principal Robot delta con Kinect 79
Fig. 72 Sección 9 Programa Principal Robot delta con Kinect 79
Fig. 73 Sección 10 Programa Principal Robot delta con Kinect 80
Fig. 74 Sección 11 Programa Principal Robot delta con Kinect 80
Fig. 75 Arduino Open – Source Community 82
Fig. 76 Ventana de Bienvenida Software Arduino..... 82
Fig. 77 Secciones de la placa Arduino Uno R3..... 83
Fig. 78 Circuito implementado..... 84
Fig. 79 Circuito Base Proteus ISIS..... 85
Fig. 80 Circuito Base Proteus ARES..... 85
Fig. 81 Vista 3D superior e inferior de la Baquelita. 86
Fig. 82 Baquelita finalmente construida. 86
Fig. 83 Sección 1 Programa Arduino 87
Fig. 84 Sección 2 Programa Arduino 87

Fig. 85 Sección 3 Programa Arduino 88
Fig. 86 Sección 4 Programa Arduino 88
Fig. 87 Sección 5 Programa Arduino 88
Fig. 88 Delta en Simmechanics 90
Fig. 89Diseño simulink 90
Fig. 90Planta simulink Delta Simmechanics 91
Fig. 91Diseño Pierna robot delta Simmechanics 91
Fig. 92 Control PID Simulink 91
Fig. 93 Modelo delta creado en Simulink 92
Fig. 94 Análisis torque de los servomotores de la base 92
Fig. 94 posición, velocidad y aceleración del efector 93
Fig. 96Trayectoria Generada 94
Fig. 97 Ángulos Servomotores 94
Fig. 98Velocidad de los motores 95
Fig. 99 Aceleración de los Servomotores 95
Fig. 100 ACS712 96
Fig. 101 ACS712 96
Fig. 102 ACS712 97
Fig. 103 ACS712 98
Fig. 104 Conversión dato ADC a parámetros de I y P 98
Fig. 105 Bloque creado para la conversión del ADC a I y P 98
Fig. 106 Medición de Corriente 99

LISTA DE TABLAS

Tabla 1 Características Material y Masa de la Plataforma Delta 31
Tabla 2 Características Material y Costo de las piezas 32
Tabla 3 Características Material y Masa de la Plataforma Delta 35
Tabla 4 Lista de materiales para la construcción de la baquela 86
Tabla 5 Consumos y %Error de la Prueba 96
Tabla 6 Consumos y %Error de la Prueba 97
Tabla 7 Consumos y %Error de la Prueba 97

CAPITULO I:
INTRODUCCION

1 CAPITULO I: INTRODUCCIÓN

La robótica durante los últimos años ha crecido en cuanto a propuestas de investigación, esto se debe a la necesidad de realizar trabajos en habitas peligrosas para el ser humano. Es importante la tecnología como medio de ayuda o herramienta para las personas. No se puede pretender realizar un robot que reemplace completamente lo que puede llegar a hacer una persona, debemos ser conscientes de esto y de las implicaciones sociales, laborales y personales como ética propia de un profesional que trabaja en esta área. Tal vez uno de los atractivos de la implementación de los sistemas robóticos en la industria, es la necesidad de producción en un largo periodo de tiempo, lo cual sería inhumano proponer como empleo.

Según la organización Internacional para la estandarización (ISO), los robots se definen como un manipulador multifuncional reprogramable, capaz de mover materiales, piezas, herramientas o dispositivos especiales a través de movimientos variables programados, para el desempeño de tareas diversas. Usualmente, una clasificación inicial se da en términos de industriales, no industriales o de uso especial, pero sin embargo algunos se dividen en sistemas seriales, paralelos, entre otros. Pero cabe resaltar que la presente tesis se enfoca en los sistemas paralelos, aunque de forma específica, la presente tesis se centra en el robot paralelo tipo Delta.

En Fig.1. Se encuentran algunos tipos de robots paralelos

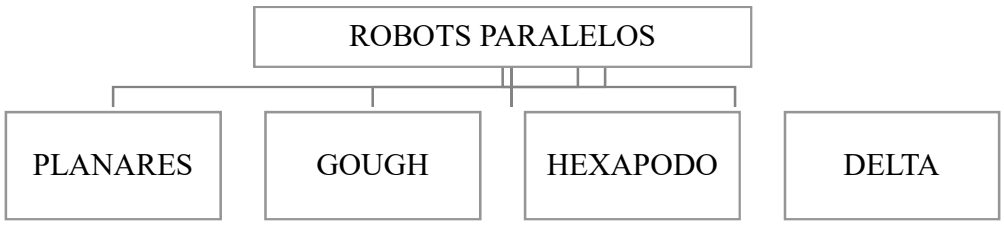


Fig. 1. Tipos de Robots paralelos.

1.1 MOTIVACIÓN

Este proyecto se inspiró en el desarrollo de un sistema creado en el programa de adaptación de Arquitectura y Computación MSc en la Bartlett, UCL. El cual ha creado la primera versión del robot delta impulsado por Kinect junto con Miriam Dall'Igna artista de instalaciones. Los cuales realizaron su prototipo más grande en la instalación llamada "Motivo Coloquios" en el prestigioso Centro Pompidou en París, junto con Ruairi Glynn y la London School of Speech and Drama. Una versión inicial del proyecto se puede observar en http://vimeo.com/20594424 y el rendimiento en el Centro Pompidou en http://vimeo.com/25792567.

Académicamente, se espera la tesis “Reconocimiento de Posición e Inclinación de la mano para Control de movimiento de una Plataforma Delta con Kinect” sea material de consulta de los estudiantes de ingeniería en Electrónica para ampliación de sus conocimientos acerca de manipuladores paralelos con el uso del sistema Kinect.

Este proyecto busca un beneficio a quienes necesiten una propuesta de implementación de Robots paralelos de alta respuesta y que podrá ser objeto a futuras investigaciones, desarrollos y/o avances en áreas como:

- La industria: en proyectos que impliquen la tele-operación del robot Delta o su implementación en celdas de manufactura.

- Realidad virtual: adaptar el robot como un dispositivo háptico y/o sensor de fuerza/torque para la interacción con ambientes virtuales en este caso el Kinect.

- Educación: Caso de estudio como diseño de una plataforma paralela delta.

1.2 OBJETIVOS

A continuación se expone los objetivos de la presente tesis.

1.2.1 Objetivo General

Construcción de un Prototipo de Hardware Modular para la acción de movimiento de una plataforma robótica delta a partir del reconocimiento de posición e inclinación de la mano mediante la integración de Software Kinect, Processing y Arduino.

1.2.2 Objetivos Específicos

- Desarrollar un algoritmo de reconocimiento de pose y distancia de la mano.
- Realizar la integración entre Kinect, Processing y Arduino.
- Construir un prototipo de plataforma Delta.
- Realizar el análisis de comportamiento de la plataforma, a partir de las coordenadas de la mano hasta el movimiento del efector.
- Realizar el análisis de la cinemática inversa del Robot tipo Delta.

1.3 METAS Y ALCANCES

Esta tesis de grado tiene como meta el diseñar e implementar un prototipo de un robot Delta que sea manipulable desde el reconocimiento de posición e inclinación de la mano para la generación del movimiento del mismo. Como se observa en la Fig.1. Adicionalmente, presenta como meta el estudio cinemático del robot por análisis geométrico, entregando así parámetros iniciales de diseño en CAD de la plataforma, para su posterior implementación.

Finalmente entregar una interfaz gráfica de usuario en la cual se puedan controlar los movimientos del robot desde un computador por medio de Kinect.

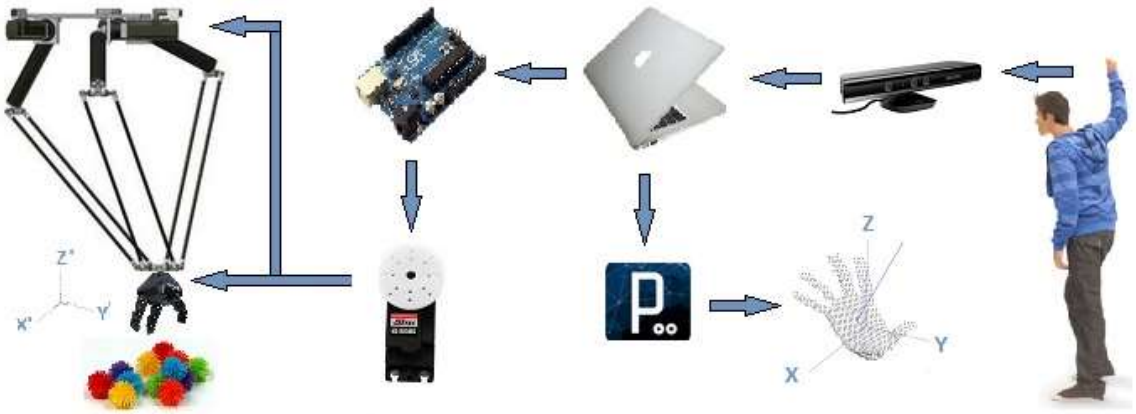


Fig. 2Esquema General del Sistema

En la presente se propone como alcance manipulación la plataforma delta, mediante la detección de la pose y coordenadas de la mano de un operario. Como sistema de adquisición de imágenes se usa el equipo Kinect® de Microsoft®. El desarrollo algorítmico para la manipulación de los datos entregados por la plataforma, se realiza mediante Software Libre con el programa Processing, inicialmente creado para sistemas de diseño gráfico y que han tenido un gran auge para desarrollo de proyectos con Kinect®. Como medio de accionamiento se formula la utilización de Arduino, como elemento de Open Hardware.

Por otro lado se entrega una propuesta de diseño de un robot delta en Solidworks. Cabe resaltar que el trabajo específicamente se centra en el diseño de la plataforma, procesamiento de la imagen para detección de la mano y no de parámetros como planificación de trayectorias de la plataforma o alguna estrategia de control de la misma.

1.4 ESQUEMA DEL DOCUMENTO

La presente tesis se encuentra distribuida de la siguiente forma:

Inicialmente, ***el capítulo II***, se enfoca a la información general de los tipos de robots paralelos como estado del arte inicial, el cual determina las diversas propuestas presentadas por la comunidad científica.

El capítulo III, se centra específicamente en el diseño de la plataforma delta a implementar mediante el diseño de cada una de las piezas como complimiento del objetivo propuesto: diseño del robot. El software utilizado como se encuentra en este capítulo está enfocado a la herramienta Solidworks.

El capítulo IV, se centra en brindar al lector de la presente tesis una información general del Modelado de la plataforma (cinemática sistema delta).

El capítulo V, da a conocer al lector las rutinas principales implementadas en Processing para la identificación de la mano del usuario y a partir de crear un sistema de referencia para manipular la plataforma.

Como es necesario poner a prueba el sistema diseñado y el algoritmo de detección de la mano en el campo de trabajo de Kinect se describe el diseño del sistema electrónico de trabajo ***capítulo VI.***

Durante la construcción del documento usted observará en algunos capítulos resultados del trabajo realizado, sin embargo en el ***capítulo VII***, encontrará algunas mediciones básicas del sistema como consumo de corriente, potencia, entre otras. Lo anterior mediante sensorica implementada.

En el ***Capítulo VIII***, se da a conocer las conclusiones generales del trabajo de tesis aquí planteado y propuestas de trabajo futuro relacionada con la presente.

Se Finaliza el documento con las respectivas ***Referencias*** y los ***apéndices*** necesarios de la investigación aquí desarrollada.

CAPITULO II:
INFORMACION GENERAL ROBOT PARALELOS

2. CAPITULO II: INFORMACION GENERAL ROBOTS PARALELOS

2.1 VISION GENERAL

Un robot es una máquina con actuadores, sensores y sistemas control, que puede llegar a realizar las tareas de los seres humanos, pero que sin embargo no busca reemplazarlos, sino como medio de apoyo a las personas. Inicialmente, se desarrollaron sistemas robóticos simples que realizaban u operaban labores fáciles y repetitivas en lugares de trabajo con dificultad de acceso o de gran peligro para las personas, disminuyendo los costos y tiempos de implementación en el proceso de manufactura de los productos, herramientas, materiales e incrementando su calidad y productividad.

Actualmente, los robots son vitales en los procesos industriales dedicados a manufacturas, siendo comúnmente utilizados en operaciones como: pick and place, paletización, despaletización, inserción y clasificación. Existen otros casos en donde el efector final o elemento final del robot deja de ser una pinza o garra para ser una herramienta de trabajo. El tipo de utensilio depende de la operación o tarea a realizar por el robot, siendo las tareas de mayor utilidad: soldadura, pintura, operaciones de mecanizado, cortes, control de calidad de productos, entre otras[1].

En la historia, los robots de configuración serial eran los designados en realizar las tareas nombradas anteriormente. Un robot de configuración serial se forma a partir de la solución al problema de la cinemática directa del robot. Otra forma de solucionar la cinemática del robot es por medio de un análisis de su geometría o empleando un software como ADAMS[2].

Un propuesta adicional para la industria es el manipulador paralelo como mecanismo compuesto de una plataforma móvil y una plataforma fija, conectadas por al menos dos cadenas cinemáticas, siendo una cadena cinemática la unión de dos o más eslabones. Generalmente, el número de grados de libertad del robot es igual al número de cadenas cinemáticas que lo conforman porque cada una de ellas es accionada por un actuador[3].

Pero los sistemas robóticos en la industria demandan de una gran velocidad de trabajo dentro de un espacio de trabajo reducido[4]. Para solucionar el problema del espacio de trabajo, se han desarrollado nuevas formas de robots paralelos, en donde se combinan articulaciones de revolución con prismáticas o las articulaciones prismáticas remplazan a las de revoluta. Estas nuevas configuraciones logran un mayor espacio de trabajo pero se reduce la velocidad de movimiento del robot, debido a que los actuadores lineales no pueden alcanzar las velocidades que alcanzan los actuadores de revolución[5].

En el presente capítulo se hace una revisión de los diferentes tipos de robots paralelos y de igual forma una revisión especifica del robot delta propuesto por clavel, el cual es utilizado

como plataforma de estudio y validación para el algoritmo implementado. Por otra parte se mencionan las contribuciones de la presente tesis.

2.2 TIPOS DE ROBOTS PARALELOS

Uno de los primeros robots paralelos de los cuales se tiene registro es una plataforma sobre la cual se encontraban las sillas de un teatro con el fin de crear un movimiento que le diera una sensación de realismo en cuanto al contexto de una función, buscando una apariencia más real al espectáculo.

Esta plataforma fue diseñada por J.E Gwinnett [6] y llamada "*Amusement Device*" y finalmente fue patentada en 1931. Al realizar una investigación de la plataforma propuesta por J.E Gwinnett se encontró que esta plataforma no llego a construirse a pesar de su gran diseño.

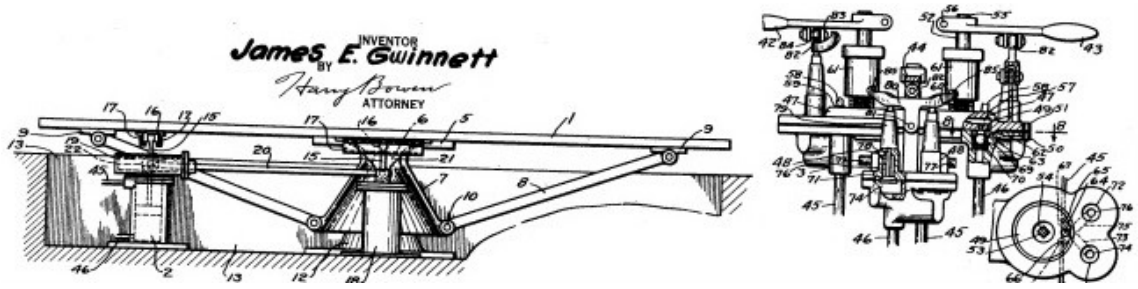


Fig. 3 Plataforma de movimiento espacial patentada por J.E.Gwinnett

Otra gran propuesta fue realizada por el inventor W.LV. Pollard [7], el cual en 1942 patento un robot paralelo (Position-Controlling Apparatus) como propuesta para ser aplicada en la concentración de pinturas en automóviles. Fig. 4. Al parecer, este robot tampoco llego a construirse.

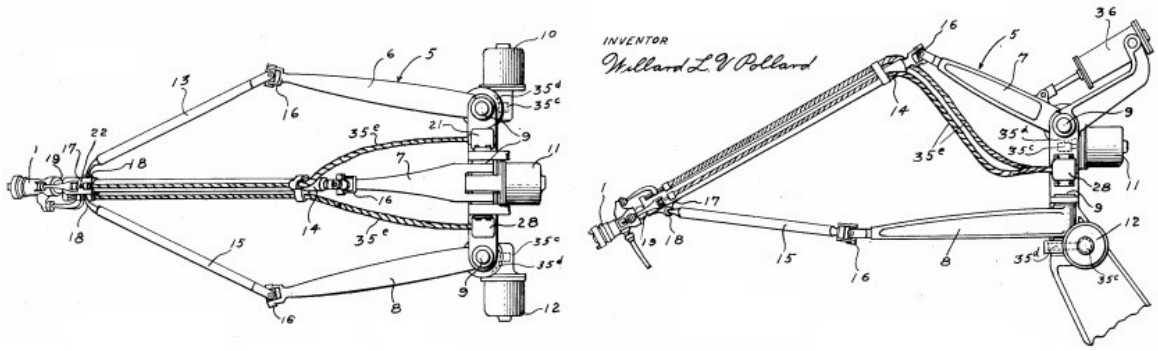


Fig. 4 Robot paralelo patentado por W.L.V. Pollard

Cinco años después en 1947 el inventor V.E. Gough ideó un robot paralelo con seis actuadores lineales o prismáticos formando una estructura en forma de octaedro, Fig.5. Este Robot con seis grados de libertad fue utilizado en la empresa Dunlop para el ensayo de neumáticos de aviación. Pero esta estructura de robot paralelo fue presentada y accionada en el congreso de FISITA en 1962 [8], la cual la convirtió en una de las estructuras paralelas más conocidas y la que ha tenido mayor éxito en la industria de esa época. En la actualidad,

injustamente, muchos autores le denominan plataforma de Stewart cuando se le debería conocer como plataforma de Gough.

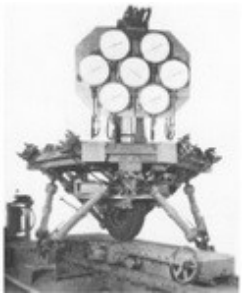


Fig. 5 Primer prototipo robot paralelo ideado por V.E. Gough

V.E. Gough en la presentación de su invento, citaba la existencia de unas mesas (MAST, Multi-Axis Simulation Table) anteriores a su invento. Estas mesas están accionadas por seis actuadores lineales, tres verticales y tres horizontales, las cuales buscaban determinan la deformación de una placa ante la aplicación de oscilaciones en diversos sentidos y la estabilidad de la misma ante cambios drásticos de orientación tal como se da a conocer en la Fig.6.

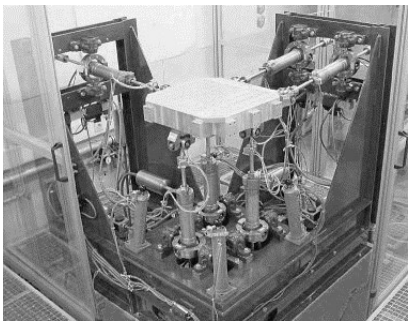


Fig. 6 MAST, Multi – Axis Simulation Table

En 1965 D. Stewart [9] presentó una plataforma con seis grados de libertad para ser utilizada como simulador de vuelo, la cual pretende someter al piloto ante cualquier situación de movimiento en cuanto a condiciones de vuelo que cambian de orientación y localización de forma drástica, esta plataforma se ha utilizado como plataforma de enseñanza y entrenamiento para los pilotos hasta la época actual, como se observa en la Fig.7 y que de igual forma ha sido renovada por la actualización tecnología que se vive recientemente.

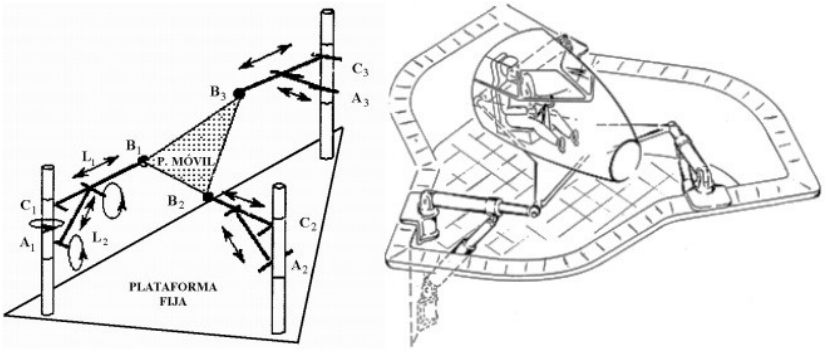


Fig. 7Plataforma de Stewart

En 1967 surge otra propuesta esta vez por K.L Cappel [10] quien patentó un simulador de vuelo utilizando la misma estructura que la plataforma de Gough. Fig. 8. Esta fue utilizada para comprobar el efecto de los elementos de un avión ante las variaciones de velocidad y la generación de alta presión provocada por los altos rangos de movimiento.

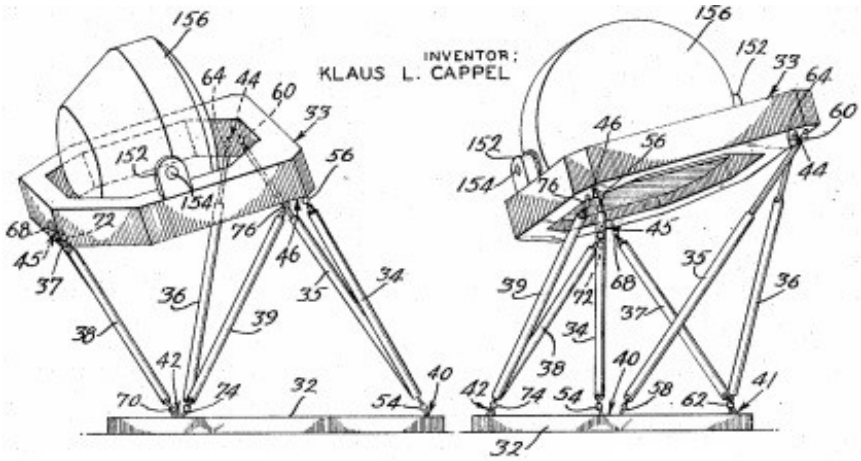


Fig. 8 Simulador de vuelo patentado por K.L Cappel

Se debe tener presente que los inventores que fueron anteriormente mencionados se consideran pioneros en el área de la robótica paralela, ya que desarrollaron sus prototipos e inventos sin conocimiento previo de los trabajos realizados anteriormente a sus propuestas de diseño. Otra propuesta de diseño de robot paralelo es el manipulador paralelo 6 – RUS con seis grados de libertad activado por actuadores giratorios presentado por K.H. Hunt [11] en 1983, el cual se observa en la Fig. 9.

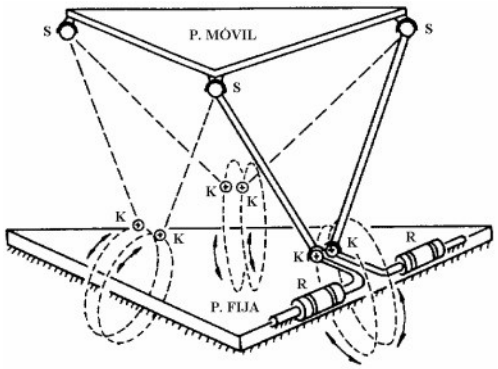


Fig. 9 Simulador de vuelo patentado por K.L Cappel

A partir del año de 1970 surgió la necesidad de conseguir un entrenamiento más económico para los pilotos de aviación que la realización de vuelos reales, hizo que se desarrollasen gran cantidad de simuladores de vuelo la mayoría nacientes de la carrera espacial en 1967. La mayoría de estos simuladores estaban basados en la estructura de Gough. Teniendo en cuenta la recopilación de publicaciones más relevantes sobre robots paralelos realizada por J.P. Merlet [12] hasta 1969 tiene recogidas 11 publicaciones. En la década de 1970 figuran 13 publicaciones. En la década de 1980 el número aumenta hasta 125. En la década de 1990

recoge 879 publicaciones. Y entre el año 2000 y la actualidad 1023 publicaciones. Teniendo en cuenta este interés sobre los robots paralelos, se creó un aumento considerable en el año de forma exponencial. Sin querer ser exhaustivo, se citan los siguientes tipos robots presentados o analizados en las publicaciones recogidas por J.P. Merlet con la clasificación propuesta por este:

2.2.1 MANIPULADORES PLANOS CON TRES GRADOS DE LIBERTAD ACCIONADOS POR MEDIO DE ACTUADORES LINEALES O GIRATORIOS

Los robots paralelos no son solo de trabajo en sistemas de coordenadas OXYZ, sino también se trabajan en configuración OXY o conocidos como sistemas planares, que dependiendo de sus actuadores obedecen a una configuración determinada como los sistemas RRR, el cual está compuesto netamente por actuadores rotacionales (R). De igual forma, estos manipuladores podrán tener inmersos movimientos traslacionales como la configuración RPP (Rotacional Prismático Prismático). Otras configuraciones se observan en la Fig.10.

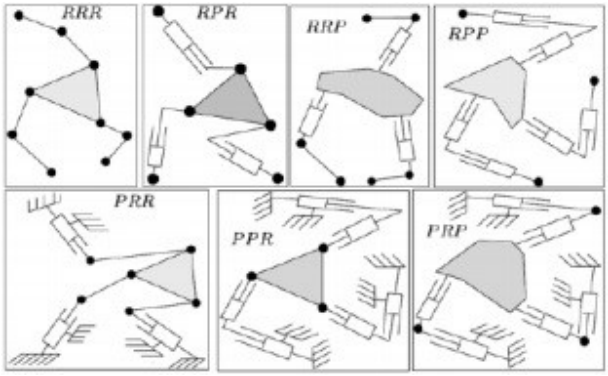


Fig. 10 Diferentes tipos de manipuladores planos con actuadores prismáticos y giratorios

2.2.2 MANIPULADORES ESPACIALES CON TRES GRADOS DE LIBERTAD ACCIONADOS POR MEDIO DE ACTUADORES GIRATORIOS

Dentro de los manipuladores con tres grados de libertad con sistema giratorio, se encuentran tres configuraciones de creciente estudio el primero de estos conocido como el robot Delta propuesto por R. Clavel [13], Fig.11. El Ojo de Águila propuesto por C. Gosselin [14], Fig.11. Y el Capaman propuesto por M. Ceccarelli [15] Fig. 11

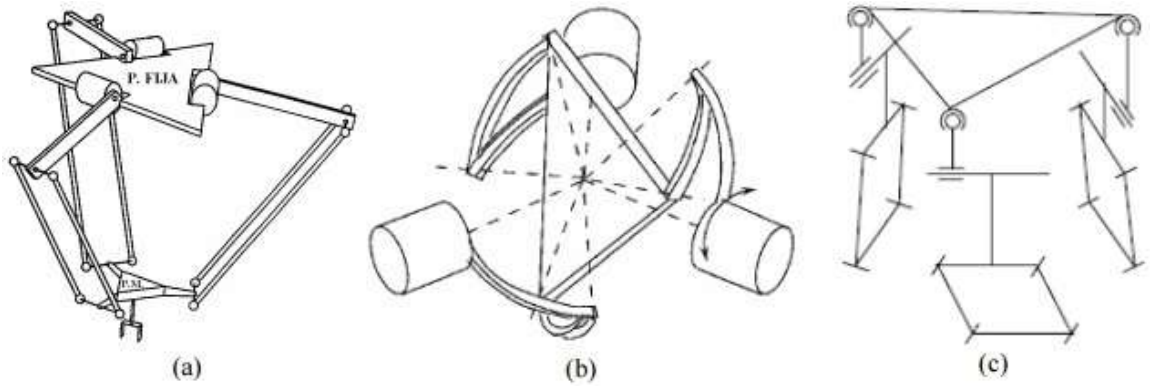


Fig. 11 Robots de tres grados de libertad con actuadores giratorios, (a) Delta, (b) Ojo de Águila, (c) Capaman.

2.2.3 MANIPULADORES ESPACIALES CON TRES GRADOS DE LIBERTAD ACCIONADOS POR MEDIO DE ACTUADORES LINEALES

En cuanto a los manipuladores con tres grados de libertad con actuadores lineales, el robot Linapod propuesto por P.B. Zobel [12], similar al Delta pero con actuadores lineales. El Orthoglide propuesto por P. Wenger y D. Chablat [13], Fig. 12(a). El Tricept patentado por K.E. Neumann [14], Fig. 12(b). Y el 3-UPU propuesto por L.W. Tsai [15], Fig. 12(c).

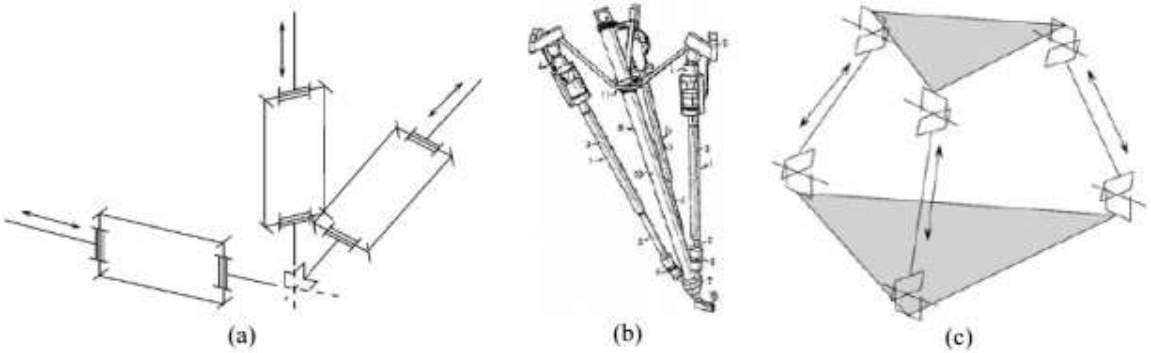


Fig. 12 Robots de tres grados de libertad con actuadores lineales, (a) Orthoglide, (b) Tricept, (c) 3-UPU.

Para las configuraciones de los robots paralelos de 4 y 5 grados de libertad las cadenas de unión de las plataformas no pueden tener la misma estructura, lo que añade más complejidad al control del robot. Aquí se pueden citar: El Delta que es el robot Delta de R. Clavel al que se le añade un giro de la pinza final. Y el Tricep de K.E. Neumann al que se le añaden dos grados de libertad a partir de la plataforma móvil.

Si nos referimos a los robots paralelos de 6 grados de libertad, aparte de los mencionados anteriormente de V.E. Gough, D. Stewart y K.H. Hunt se pueden mencionar los siguientes: El Active Wrist presentado por J.P. Merlet y C. Gosselin [16], Fig. 13(a). El Hexaglide presentado por M. Honegger y otros [17], Fig. 13(b). Y el Tri-Scott presentado por I. Zabalza y otros [18], Fig. 13(c).

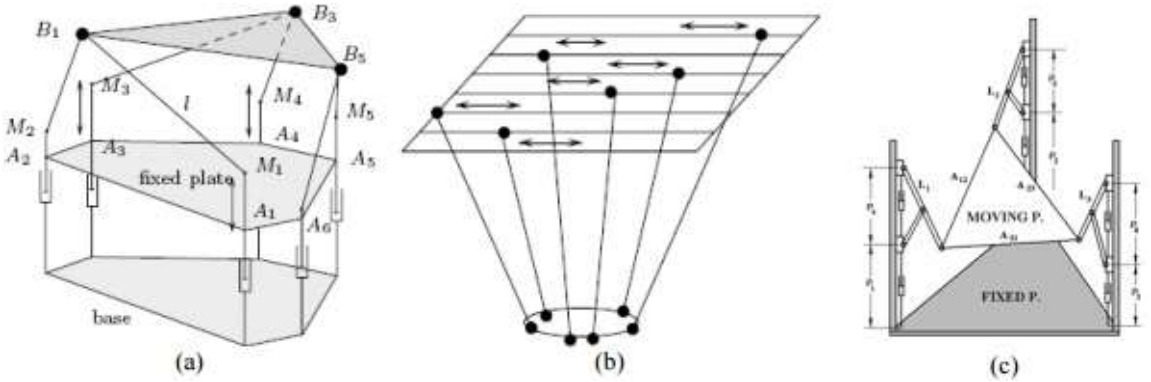


Fig. 13 Robots de seis grados de libertad, (a) Active Wrist, (b) Hexaglide, (c) Tri-Scott.

Es importante mencionar que la información recogida por los autores a través de los asistentes al Workshop: Fundamental issues and future research directions for parallel mechanisms and manipulators, celebrado en Québec en 2002 y a los Congresos de ARK (Advances in robot Kinematics) celebrados en Caldas de Malavella en 2002 y en Sestri

Levante en 2004, de empresas fabricantes de Robots y del libro publicado por J.P. Merlet [19], se supone que la mayoría de estructuras de robots paralelos presentados por publicaciones en revistas y congresos no han pasado de ser un estudio teórico. De otra gran cantidad de estructuras solamente se ha construido algún prototipo. Y los que han tenido un cierto éxito siendo comercializados, son: El robot Delta presentado por R. Clavel. El robot Tricept patentado por K.N. Neumann. Y la Plataforma de Gough.

De los robots Delta y Delta-4, según el Dr. M. Bouri[16] de la Escuela Politécnica Federal de Lausana, hay alrededor de 4000 unidades funcionando en todo el mundo. Del robot Tricept, según el Dr. J.L. Olazagoitia de la empresa PKM Tricept, hay del orden de 300 unidades funcionando. Y de plataformas de Goug, la previsión es más complicada ya que existen muchos fabricantes, pero se estima que puede haber funcionado en el mundo del orden de 20000 unidades.

Los robots Delta y Delta-4[17], debido a la poca masa de sus brazos que permiten la realización de hasta 5 maniobras por segundo se utilizan en la manipulación de objetos y debido a su sencillez que permite su limpieza por medio de un chorro de agua se utilizan principalmente para la manipulación de productos alimenticios. El robot Tricept, debido a su estructura en forma de tetraedro que le confiere una gran rigidez, se utiliza como máquina herramienta para trabajos de precisión media, del orden de 0.05 milímetros. De la plataforma de Gough existen infinidad de tamaños, desde ejemplares de algunos milímetros hasta otros de varios metros. Algunas aplicaciones de este tipo de robot son: Orientación de antenas, telescopios y paneles solares, Aislamiento y producción de vibraciones.

En cirugía para posicionamiento de microscópios e incluso de pacientes. Simuladores de vuelo para aviones y helicópteros así como simuladores de conducción de vehículos, carretillas elevadoras, etc. En la industria para ensamblaje de componentes, para posicionamiento de piezas y utillajes y como máquina herramienta para precisiones medias, del orden de 0.05 milímetros.

2.3 ROBOT DELTA

En la presente tesis la plataform[13]a a utilizar es el robot delta, el cual es un tipo de robot paralelo inventado por Reymond Clavel en la década de 1980 y patentado en 1990 en estados unidos. La idea principal de este es el uso de paralelogramos en el diseño de las piernas del robot con el fin de mantener la orientación de una plataforma en el extremo, o efector, el cual tiene tres grados de libertad de traslación, pero no rotación.

El robot Fig.14 consta de sólo tres motores que controlan la rotación de las tres piernas, que transforman estas rotaciones en una pose específica determinado por la conectividad y la naturaleza de las juntas de la estructura[18]. También se han implementado algoritmos para maximizar el volumen de trabajo de los robots paralelos usando: algoritmos genéticos, el método SQP, el método CRS, entre otros.

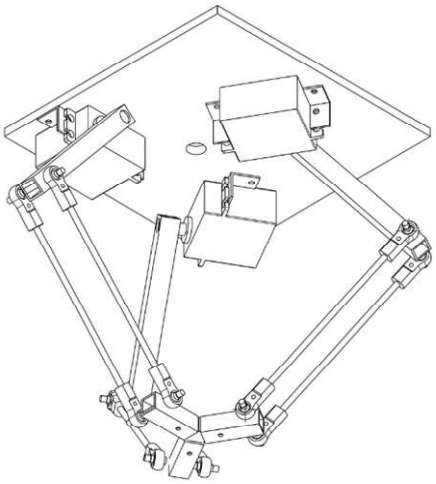


Fig. 14 Robot Delta

La plataforma Delta de gran importancia por su alta velocidad de operación, conveniente para la industria de manufactura, clasificación y selección de piezas. Se presenta desde varias décadas como un campo atractivo de investigación, por ejemplo, las diversas discusiones que se originan a partir del estudio cinemático del mismo[19]. Actualmente, se han planteado diversas aplicaciones para este tipo de plataformas. Su enfoque puede ir desde realizar tareas repetitivas, mediante algoritmos implementados en sistemas embebidos, donde su acción de desplazamiento se encuentra predeterminada, hasta otros, que dependerán de variables externas de elementos que interactúan de alguna forma con la plataforma. Como se mencionó, el robot delta tiene tres miembros (a pesar de que hay algunas plataformas de cuatro piernas) unidos a una base fija en un extremo y al efector del robot en el otro extremo. La base fija acomoda tres servos y normalmente el sistema embebido, esta es la única parte estática del robot. Las piernas constituyen una cadena cinemática que traduce los estados rotacionales de los servos en posiciones espaciales específicas del efector, que se compone de una base giratoria y una pinza, controlado por dos pequeños servos[20].

2.4 CONCLUSION

El aporte de este capítulo se enfoca a la construcción del estado del arte correspondiente a los robots paralelos, los documentos referenciados brindan una mayor visión en cuanto a los desarrollos aquí expresados.

CAPITULO III:
DISEÑO DE LA PLATAFORMA DELTA

3. CAPITULO III: DISEÑO DE LA PLATAFORMA DELTA

3.1 VISION GENERAL

En el presente capítulo se presenta el diseño de la Plataforma Delta, iniciando con la utilización del software Solidworks 2015, para el modelado de cada una de las piezas necesarias para su construcción donde se determinan características del material a utilizar hasta su implementación en físico[20].

3.2 DISEÑO DE PIEZAS EN SOLIDWORKS 2015

Para la implementación del robot delta se determinó que era necesario el modelado de las piezas del robot, para tal fin se da a conocer a continuación el modelado de cada una de estas[21]. Las medidas correspondientes a las piezas diseñadas y construidas se encuentran en el Apéndice A titulada Diseño Delta Solidworks.

3.2.1 BASE FIJA, SOPORTE MOTORES Y BASE MOVIL

Para la estructura del robot delta se hace necesario el diseño de la base fija; la cual es la encargada de alojar los tres servomotores principales que otorga movimiento a las piernas del robot delta, tal cual como se observó en el capítulo anterior en Fig.14. En la Fig. 15 se encuentra la forma de la base utilizada, el material que se decidió para su construcción obedece a un acrílico de espesor de 5mm, todas las medidas de las piezas están dadas en mm.

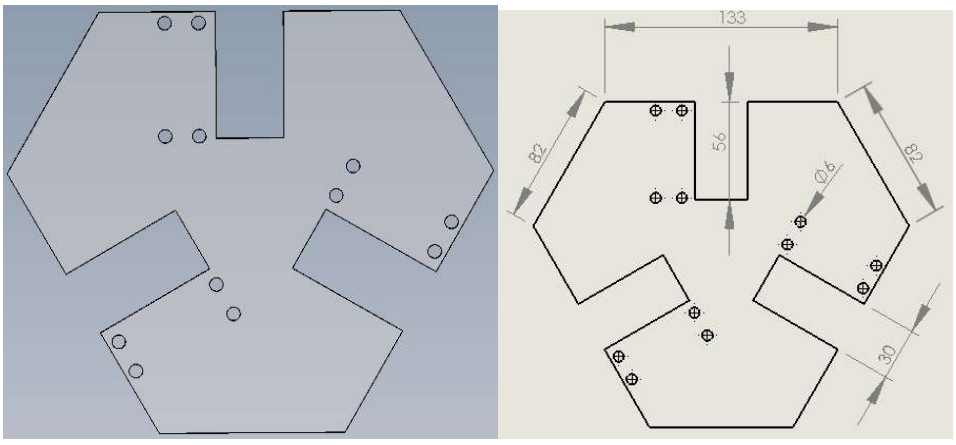


Fig. 15 Base Fija Robot Delta

Teniendo en cuenta Fig.15, se encuentra que se tiene definida la posición donde se encuentran los servomotores, para lograr generar un soporte para los mismos se diseñó una pequeña placa en acrílico que permitieran el reposo y soporte de los mismos a la Base Fija Fig.16.

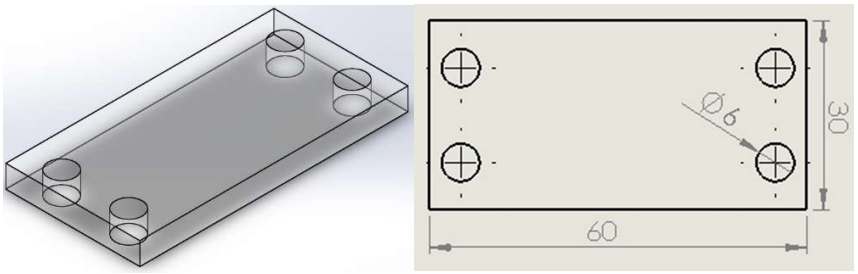


Fig. 16 Base Motor Robot Delta

Entre las piezas más importantes de la plataforma delta se encuentra la base móvil del efector donde se encuentra alojada la pinza y la unión con las piernas del robot delta. Las medidas y espacios de los elementos están especialmente distribuidos según la morfología del robot delta Fig.17.

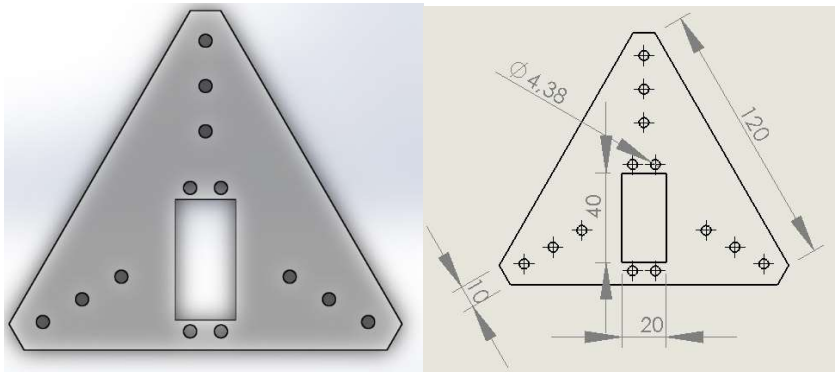


Fig. 17 Base Móvil Robot Delta

3.2.2 SERVOMOTOR HS-311

La plataforma delta diseñada requiere de 5 servomotores, distribuidos 3 para la manipulación de las piernas y 2 para el movimiento del efector. En Fig.18 se encuentra el diseño del servomotor HS- 311 con su respectivo conector del eje.

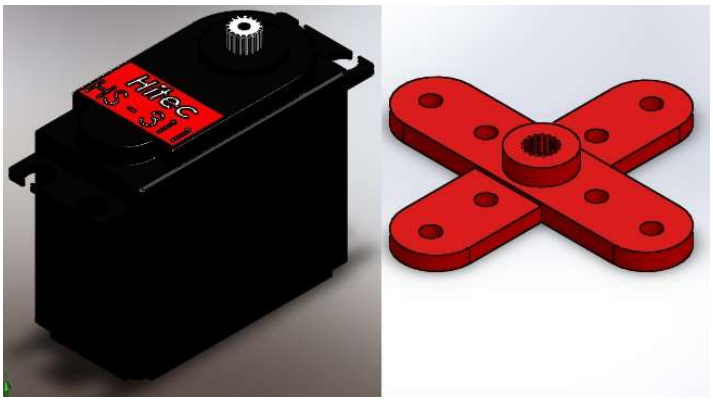


Fig. 18 Servomotor y conector eje

3.2.3 EFECTOR O PINZA FINAL

El sistema de pinza de plástico es uno de los pocos elementos al igual que el motor que no se construyeron artesanalmente, debido a su fácil adquisición y bajo costo fue un aporte importante para el desarrollo del proyecto pero igual que los elementos anteriores se procedió a realizar el diseño en Solidworks para análisis de aportes de datos en cuanto a peso del sistema Fig.19. Observe que se sugiere la conexión de los motores en la parte superior de la pinza en relación a la rotación de la misma y para la apertura y cierre por medio de la rotación del círculo central.

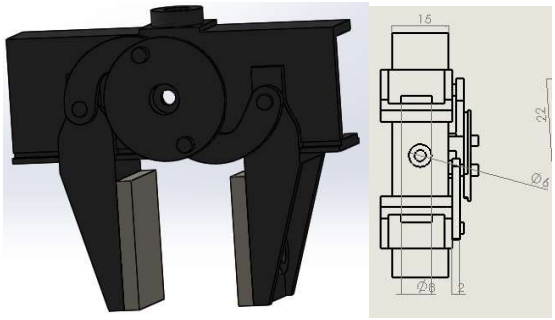


Fig. 19 Pinza o efector del robot delta

3.2.4 BARRA MUSLO, PIERNA Y UNION BASE MOVIL

El material utilizado para estos tres materiales fue aluminio, aunque cabe resaltar que tal vez de la plataforma delta el "muslo" y la "pierna" del delta son los de mayor aporte en el cálculo cinemático, ya que el mismo dependerá de la distancia de estos. En Fig.20. Se observa las piezas, al igual que la unión que se necesita para unir la pierna con la base móvil.

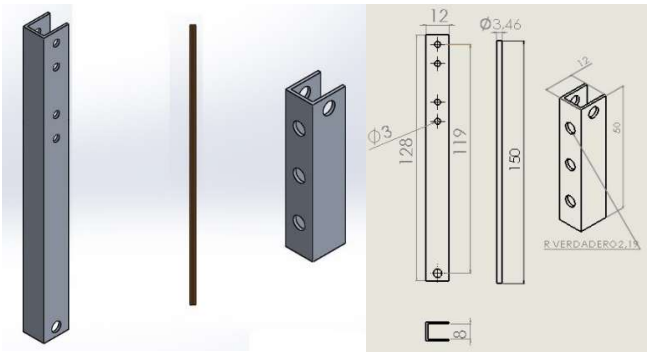


Fig. 20 Base Muslo, Pierna y Unión

3.2.5 ROD END, TORNILLO Y TUERCA

Considerados como elementos de unión o juntas tal vez el más importante es el "Rod End", ya que su movilidad característica es la que ayudara a definir el área de trabajo del Robot delta Fig.21.

Fig. 21 Pinza o efector del robot delta

3.2.6 PLATAFORMA DELTA DISEÑADA

Fig. 22 Robot delta Final

Teniendo en cuenta las piezas anteriormente descritas, se obtuvo la plataforma delta final Fig.20 con las siguientes características:

Tabla 1 Características Material y Masa de la Plataforma Delta

Cant.	Nombre	Material	Masa U. (gr.)	Total Masa (gr.)
1	Base Fija	Acrílico	144.49	144.49
3	Base Motor	Acrílico	10.12	30.36
1	Base Efector	Acrílico	44.17	44.17
5	Servomotor HS-311	PE Alta densidad	27.05	135.25
3	Ejes Servomotor	PP Homopolimero	2.58	7.74
3	Barras Muslo	Aluminio Aleación 1060	10.83	32.49
3	Barra soporte unión	Aluminio Aleación 1060	4.12	12.36
12	Rod End	ABS y acero cromado	1.23	14.76
31	Tornillos	Hierro dúctil	1.68	52.08
31	Tuercas	Hierro dúctil	0.16	4.96
6	Barra Pierna	Bronce Aluminio	10.44	62.64
		Total	**256.87**	**541.3**

La tabla anterior expresa la cantidad de elementos que fueron diseñados y utilizados en el ensamblaje final del sistema de igual forma, se determinó la masa en gramos de cada una de las piezas encontrando que el robot delta diseñado presenta una masa de 541.3 gramos como se observó en la Tabla.1. A continuación se encuentra relacionado el costo de materiales relacionado únicamente en cuanto a la plataforma delta Tabla.2.

Tabla 2 Características Material y Costo de las piezas

Cant.	Nombre	Material	Valor Unitario	Valor Total
1	Base Fija	Acrílico	60.000	60.000
3	Base Motor	Acrílico	5.000	15.000
1	Base Efector	Acrílico	20.000	20.000
5	Servomotor HS-311	PE Alta densidad	30.000	150.000
3	Ejes Servomotor	PP Homopolimero	500	1500
3	Barras Muslo	Aluminio Aleación 1060	2.500	7.000
3	Barra soporte unión	Aluminio Aleación 1060	2.500	7.000
12	Rod End	ABS y acero cromado	2000	24.000
31	Tornillos	Hierro dúctil	100	3.100
31	Tuercas	Hierro dúctil	100	3.100
6	Barra Pierna	Bronce Aluminio	2.500	15.000
		Total	**125.200**	**305.700**

En Fig.23. Se observa la versión real de la plataforma delta diseñada en Solidworks y la cual es la plataforma que obedecerá el reconocimiento de los movimientos de la plataforma.

Fig. 23 Plataforma Delta Real

3.3 CONCLUSION

Se realizó el diseño de todas las piezas que intervienen en la plataforma delta, donde a partir de las medidas obtenidas se procedió a realizar la construcción de la plataforma delta real, se hizo necesaria la construcción de la plataforma debido al alto costo que tiene este tipo de dispositivos en la industria, por tanto se creó una propuesta barata de estudio y fácil construcción para la actividad de investigación académica básica. Las medidas de la plataforma, en cuanto al muslo, base, pierna y base del efector aportara al análisis cinemático de la misma.

CAPITULO IV:
MODELADO DE LA PLATAFORMA DELTA

4. CAPITULO IV: MODELADO DE LA PLATAFORMA DELTA

4.1 VISION GENERAL

Para poder obtener un modelo matemático que describa la cinemática del robot, primero se debe analizar su geometría. Como se observa en Fig.24, el robot Delta está formado por una base fija, en donde se ubicaron tres servomotores $(\boldsymbol{S_1}, \boldsymbol{S_2}\ y\ \boldsymbol{S_3})$ a una misma distancia del centro de la base y separados $\boldsymbol{120^o}$entre sí; una base móvil conectada a la base fija por medio de tres cadenas cinemáticas, cada una compuesta por dos eslabones, los cuales se conectan entre sí por medio de junturas universales $(\boldsymbol{U})$. Adicionalmente, una segunda plataforma móvil se une a la primera y cuenta con un cuarto servomotor $\boldsymbol{S_4}$ para permitir la orientación del robot[22]–[27].

Teniendo en cuenta el diseño planteado en el capítulo III, se obtienen las características de dimensiones y trabajo de la plataforma delta. Observe la tabla.3.

Tabla 3 Características Material y Masa de la Plataforma Delta

Parámetros	**Datos**
Control Servos HS311 ancho de pulso	1500 uSeg 0°
Voltaje de operación	5 voltios
Corriente máxima de operación	1,7 A
Distancia del Muslo	100 mm
Distancia de la pierna	160 mm
Distancia L de la base Fija	70 mm
Distancia l de la base movil	50 mm
Velocidad de Operación	0,19 seg/60° sin carga
Torque	3,0 kg/cm
Xmax, Xmin	-8cm, 8 cm
Ymax, Ymin	8 cm, -8 cm
Zmax, Zmin	30 cm, 8 cm
Junturas Rod Ends de traxxas	5 mm de diámetro interno

4.2 GEOMETRIA DEL ROBOT

El sistema de referencia ***XYZ*** se ubica en el centro de la base fija, con el eje ***X*** perpendicular al eje del servomotor $\boldsymbol{S_1}$ y el eje **Z** perpendicular a la base.

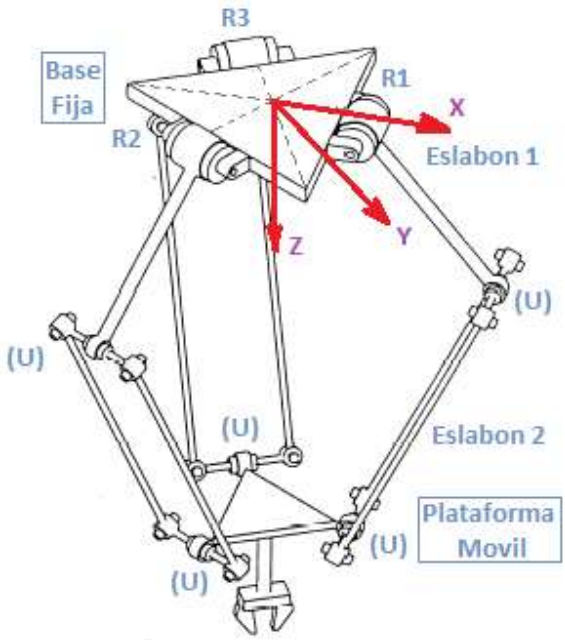


Fig. 24 Estructura del Robot paralelo RUU

Los grados de libertad del robot se determinan empleando Eq. (1.1), donde Hunt[11] plantea que los grados de libertad DOF[1]. El número de grados de libertad del robot es la suma de los grados de libertad de las articulaciones que lo componen.

$$F = \lambda(n - j - 1) + \sum_{i=1}^{j} f_i \qquad (1.1)$$

Donde $\boldsymbol{F}$ son los DOF del mecanismo; $\boldsymbol{\lambda}$ los DOF del espacio en el cual el mecanismo funciona ($\boldsymbol{\lambda = 6}$, para mecanismos espaciales y $\boldsymbol{\lambda = 3}$ para mecanismos planares); **n** el número de cuerpos rígidos del mecanismo; **j** el número de articulaciones y $\boldsymbol{f}_i$ los DOF de la articulación i. Como el robot delta a analizar es un mecanismo espacial, entonces $\boldsymbol{\lambda = 6}$, adicionalmente; esta forado por seis eslabones, una plataforma fija y dos móviles (**n=9**); cuenta con cuatro articulaciones de revolución y seis articulaciones universales $\boldsymbol{j = 10}$; cada articulación de revolución cuenta con un grado de libertad y cada articulación universal cuenta con dos grados de libertad $\sum_{i=1}^{j} 16$; entonces los grados de libertad del robot Delta son:

$$\boldsymbol{F = 6(9 - 10 - 1) + 16 = 4} \quad (1.2)$$

De esta forma, el robot Delta cuenta con tres grados de libertad para posicionar y un grado de libertad para orientar su efector final en el espacio de trabajo. Como se mencionó anteriormente, el robot consta de tres cadenas cinemáticas ($\boldsymbol{i = 1, 2\ y\ 3}$), cada una de ellas formadas por dos eslabones, cuyas dimensiones son $\boldsymbol{L_1}$ y $\boldsymbol{L_2}$ respectivamente. En la Fig.25. se puede apreciar que el eslabón L_1 se une a la base fija por medio de la articulación A_i y al eslabón L_2 por la articulación B_i. El eslabón L_2 se une a la plataforma móvil por medio de la

[1] Por sus siglas en ingles Degrees of Fredom

articulación C_i. Finalmente, β_i es el ángulo que hay entre el eslabón L_1 y L_2. El centro de la plataforma móvil representa la posición del robot Delta $\boldsymbol{P} = [\boldsymbol{P_x}\ \boldsymbol{P_y}\ \boldsymbol{P_z}]^{\boldsymbol{T}}$.

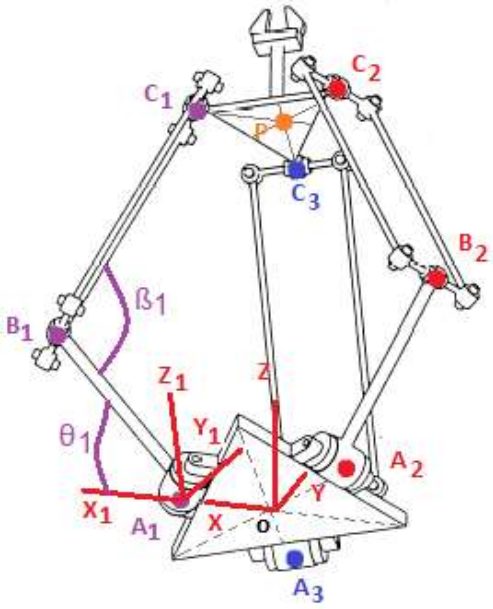


Fig. 25 Modelo geométrico del robot Delta

La Fig.26. define los ángulos de las articulaciones asociados con la i-esima cadena cinemática, donde $\overrightarrow{OP}$ es el vector posición del centro de la plataforma móvil, θ_{1i} es el ángulo que se forma entre X_{i1} y el vector $\overrightarrow{A_iB_i}$ y la intersección del plano del paralelogramo y el plano $X_{i1} - Z_{i1}$ y θ_{3i} es el ángulo que se forma de la dirección Y_{i1} hasta $\overrightarrow{B_iC_i}$. En total hay nueve ángulos de articulaciones ($\theta_{i1}, \theta_{i2}\ y\ \theta_{i3}$ para i=1, 2 y 3) asociados al manipulador, siendo $\theta_{11}, \theta_{12}\ y\ \theta_{13}$ las variables de las articulaciones de los actuadores.

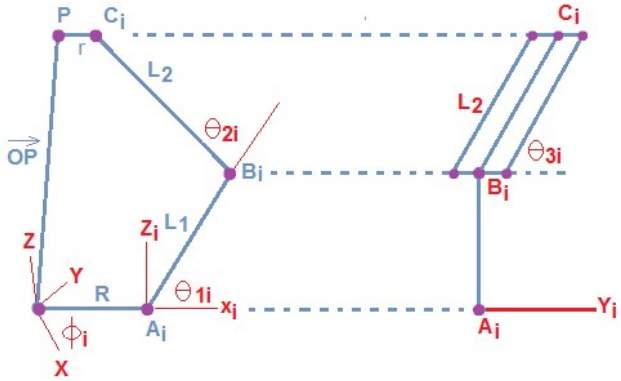


Fig. 26 Descripción de los ángulos: izq) Vista Frontal der) Vista lateral.

Una matriz de transformación $\boldsymbol{T}$ representa la orientación y la posición de un sistema $O'X'Y'Z'$ rotado y trasladado con respecto a un sistema de referencia $OXYZ$. Esta matriz es

útil para conocer las coordenadas del vector $\vec{r} = [r_x, r_y, r_z]^T$ en el sistema de referencia, a partir del vector $\vec{r}' = [r_x', r_y', r_z']^T$.

$$\begin{bmatrix} r_X \\ r_Y \\ r_Z \\ 1 \end{bmatrix} = T \begin{bmatrix} r_X' \\ r_Y' \\ r_Z' \\ 1 \end{bmatrix} \quad (1.3)$$

Donde la matriz **T** es:

$$T = \begin{bmatrix} R_{3x3} & P_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} Rotacion & Traslacion \\ 0_{1x3} & 1 \end{bmatrix} \quad (1.4)$$

El sistema A_i, X_i, Y_i, Z_i esta trasladado R a lo largo de OX y rotado ϕ_i alrededor de OZ, con respecto al sistema de referencia OXYZ. Para conocer los vectores $\overrightarrow{OA_i}$, $\overrightarrow{OB_i}$ y $\overrightarrow{OC_i}$, conociendo los vectores $\overrightarrow{A_iB_i}$ y $\overrightarrow{A_iC_i}$ se utiliza una traslación seguida de una rotación. Si se traslada un vector con coordenadas (x, y, z), seguida de una rotación sobre el eje *Z*, la matriz *T* es:

$$T\big(P, (Z, \alpha)\big) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & x\cos(\alpha) - y\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) & 0 & x\sin(\alpha) + y\cos(\alpha) \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.5)$$

Aplicando la Eq.1.1 con la transformación homogénea Eq.1.3 se obtienen los vectores $\overrightarrow{OA_i}$, $\overrightarrow{OB_i}$ y $\overrightarrow{OC_i}$:

$$\overrightarrow{OA_i} = \begin{bmatrix} \cos(\phi_i) & -\sin(\phi_i) & 0 & R\cos(\phi_i) \\ \sin(\phi_i) & \cos(\phi_i) & 0 & R\sin(\phi_i) \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} R\cos(\phi_i) \\ R\sin(\phi_i) \\ 0 \\ 1 \end{bmatrix} \quad (1.6)$$

$$\overrightarrow{OB_i} = \begin{bmatrix} \cos(\phi_i) & -\sin(\phi_i) & 0 & R\cos(\phi_i) \\ \sin(\phi_i) & \cos(\phi_i) & 0 & R\sin(\phi_i) \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L_1\cos(\theta_{1i}) \\ 0 \\ L_1\sin(\theta_{1i}) \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos(\phi_i)\,(R + L_1\cos(\theta_{1i})) \\ \sin(\phi_i)\,(R + L_1\cos(\theta_{1i})) \\ L_1\sin(\theta_{1i}) \\ 1 \end{bmatrix} (1.5)$$

$$\overrightarrow{OC_i} = \begin{bmatrix} \cos(\phi_i) & -\sin(\phi_i) & 0 & R\cos(\phi_i) \\ \sin(\phi_i) & \cos(\phi_i) & 0 & R\sin(\phi_i) \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_X\cos(\phi_i) + P_Y\sin(\phi_i) + r - R \\ P_Y\cos(\phi_i) - P_X\sin(\phi_i) \\ P_Z \\ 1 \end{bmatrix} = \begin{bmatrix} P_X + r\cos(\phi_i) \\ P_Y + r\sin(\phi_i) \\ P_Z \\ 1 \end{bmatrix} \quad (1.6)$$

Adicionalmente, se puede encontrar los vectores $\overrightarrow{B_iA_i}$, $\overrightarrow{B_iC_i}$ y $\overrightarrow{A_iC_i}$:

$$\overrightarrow{B_iA_i} = \overrightarrow{OA_i} - \overrightarrow{OB_i} = \begin{bmatrix} -L_1\cos(\phi_i)\cos(\theta_{1i}) \\ -L_1\sin(\phi_i)\cos(\theta_{1i}) \\ -L_1\sin(\theta_{1i}) \end{bmatrix} \quad (1.7)$$

$$\overrightarrow{B_iC_i} = \overrightarrow{OC_i} - \overrightarrow{OB_i} = \begin{bmatrix} P_X - \cos(\phi_i)\,(R + L_1\cos(\theta_{1i}) - r) \\ P_Y - \sin(\phi_i)\,(R + L_1\cos(\theta_{1i}) - r) \\ P_Z - L_1\sin(\theta_{1i}) \end{bmatrix} \quad (1.8)$$

$$\overrightarrow{A_iC_i} = \overrightarrow{OC_i} - \overrightarrow{OA_i} = \begin{bmatrix} P_X + \cos(\phi_i)\,(r - R) \\ P_Y + r\sin(\phi_i)\,(r - R) \\ P_Z \end{bmatrix} \quad (1.9)$$

Por facilidad de expresión, en las Eq1.1 a la Eq1.9 se denota el coseno de un ángulo $\cos(\theta_{1i})$ como $C_{\theta_{1i}}$ y $\sin(\theta_{1i})$ como $S_{\theta_{1i}}$.

Geométricamente, la base fija y la plataforma móvil son dos triángulos equiláteros de lados L y $\boldsymbol{l}$ respectivamente. Como se aprecia en la Fig.27 a la distancia del centro de la base a A_i es R. En la Fig.27 se aprecia que la distancia del punto P a C_i es r.

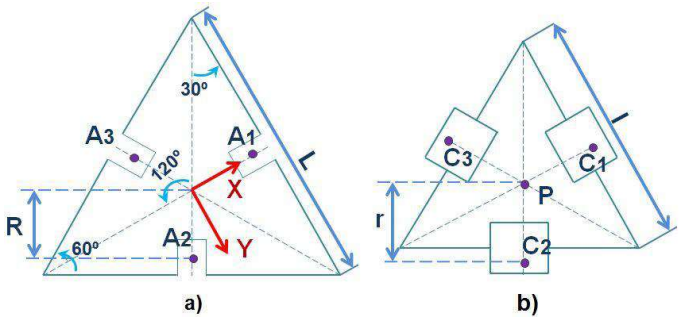


Fig. 27 Forma geométrica de: a) la base fija y b) la plataforma móvil.

4.3 CINEMÁTICA INVERSA

La cinemática inversa permite conocer las variables de los actuadores si se conoce la posición del efector final, en este caso, la posición de la plataforma móvil. Solucionar la cinemática inversa le permite al sistema de control enviarle órdenes a los actuadores para llevar el efector final de un robot hasta un punto deseado en el espacio[28]–[33]. Gracias a la simetría del robot Delta, cada cadena cinemática puede ser analizada independientemente.

Es posible formar un triángulo con vértices $A_iB_iC_i$ como se aprecia en la Fig.28. y se puede encontrar la magnitud A_iC_i por medio del teorema del coseno:

$$A_iC_i^2 = L_1^2 + L_2^2 - 2L_1L_2\cos(\beta_i) \quad (1.10)$$

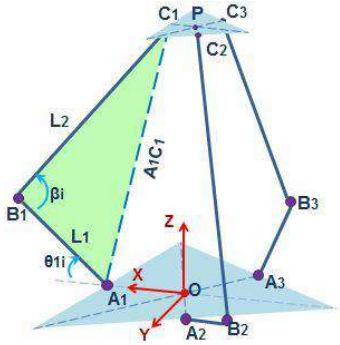


Fig. 28 Representación vectorial de la primera cadena cinemática.

En la Eq.10 la única incógnita es $\cos(\beta_i)$, la cual se puede encontrar usando la definición del producto escalar:

$$\overrightarrow{B_iA_i} * \overrightarrow{B_iC_i} = (B_iA_i)(B_iC_i)\cos(\beta_i) \quad (1.11)$$

Como $B_iA_i = L_1$ y $B_iC_i = L_2$ entonces:

$$\cos(\beta_i) = \frac{\overrightarrow{B_iA_i} * \overrightarrow{B_iC_i}}{L_1L_2} \quad (1.12)$$

Remplazando la Eq.12 en la Eq.10 se obtiene:

$$A_iC_i^2 = L_1^2 + L_2^2 - \overrightarrow{B_iA_i} * \overrightarrow{B_iC_i} \quad (1.13)$$

Donde:

$$2\overrightarrow{B_iA_i} * \overrightarrow{B_iC_i} = 2\left(L_1^2 - L_1C_{\theta 1i}\left(P_XC_{\phi i} + P_YS_{\phi i} + r - R\right) - L_1S_{\theta 1i}P_z\right) \quad (1.14)$$

Remplazando la Eq.14 en la Eq.13 se obtiene:

$$A_iC_i^2 = L_1^2 - L_2^2 + 2L_1C_{\theta 1i}\left(P_XC_{\phi i} + P_YS_{\phi i} + r - R\right) + 2L_1S_{\theta 1i}P_z \quad (1.15)$$

La magnitud del vector de la Eq.9 $\overrightarrow{AC_i}$ es:

$$A_iC_i^2 = \left(P_X + C_{\phi i}(r - R)\right)^2 + \left(P_Y + S_{\phi i}(r - R)\right)^2 + P_Z^2 \quad (1.16)$$

Igualando Eq.16 y la Eq.15 se obtiene una ecuación de la forma:

$$a_i S_{\theta 1i} + b_i C_{\theta 1i} = c_i \quad (1.17)$$

Donde:

$$ca_i = 2P_Z L_1$$

$$cb_i = 2L_1\left(P_X C_{\phi i} + P_Y S_{\phi i} + r - R\right)$$

$$cc_i = \left(P_X + C_{\phi i}(r-R)\right)^2 + \left(P_Y + S_{\phi i}(r-R)\right)^2 + P_Z^2 + L_1^2 - L_2^2$$

Es posible rescribir la Eq.17 teniendo en cuenta la identidad trigonométrica de suma de senos y cosenos:

$$\theta_{1i} = \sin^{-1}\left(\frac{cc_i}{\sqrt{ca_i^2 + cb_i^2}}\right) - \tan^{-1}\left(\frac{cb_i}{ca_i}\right) \quad (1.18)$$

4.3.1 VARIABLE DE ARTICULACIÓN $\boldsymbol{\theta_{11}}$

Para encontrar el ángulo θ_{11}, i toma el valor de uno y el ángulo ϕ_1 es de 0°, entonces:

$$ca_1 = 2P_Z L_1$$

$$cb_1 = 2L_1(P_X + r - R)$$

$$cc_1 = (P_X + r - R)^2 + {P_Y}^2 + P_Z^2 + L_1^2 - L_2^2$$

$$\theta_{11} = \sin^{-1}\left(\frac{cc_1}{\sqrt{ca_1^2 + cb_1^2}}\right) - \tan^{-1}\left(\frac{cb_1}{ca_1}\right) \quad (1.19)$$

4.3.2 VARIABLE DE ARTICULACIÓN $\boldsymbol{\theta_{12}}$

Para encontrar el ángulo θ_{12}, i toma el valor de dos y el ángulo ϕ_2 es de 120°, entonces:

$$ca_2 = 2P_Z L_1$$

$$cb_2 = L_1\left(-P_X + \sqrt{3}P_Y + 2r - 2R\right)$$

$$cc_2 = \left(P_X - \frac{1}{2}(r-R)\right)^2 + \left(P_Y - \frac{\sqrt{3}}{2}(r-R)\right)^2 + P_Z^2 + L_1^2 - L_2^2$$

$$\theta_{12} = \sin^{-1}\left(\frac{cc_2}{\sqrt{ca_2^2 + cb_2^2}}\right) - \tan^{-1}\left(\frac{cb_2}{ca_2}\right) \quad (1.20)$$

4.3.3 VARIABLE DE ARTICULACIÓN $\boldsymbol{\theta_{13}}$

Para encontrar el ángulo θ_{13}, i toma el valor de tres y el ángulo ϕ_3 es de 240°, entonces:

$$ca_3 = 2P_Z L_1$$

$$cb_3 = L_1\left(-2P_X + \sqrt{3}P_Y + 2r - 2R\right)$$

$$cc_3 = \left(P_X - \frac{1}{2}(r-R)\right)^2 + \left(P_Y - \frac{\sqrt{3}}{2}(r-R)\right)^2 + P_Z^2 + L_1^2 - L_2^2$$

$$\theta_{13} = \sin^{-1}\left(\frac{cc_3}{\sqrt{ca_3^2 + cb_3^2}}\right) - \tan^{-1}\left(\frac{cb_3}{ca_3}\right) \qquad (1.20)$$

4.4 CINEMÁTICA DIRECTA

En la sección anterior se definió el vector $\overrightarrow{B_i C_i}$, Eq.1.8, cuya magnitud es L_2:

$$B_i C_i = \left(P_X - C_{\phi i}(R + L_1 C_{\theta 1i} - r)\right)^2 + \left(P_Y - S_{\phi i}(R + L_1 C_{\theta 1i} - r)\right)^2 + (P_Z - L_1 S_{\theta 1i})^2 = L_2^2 \ (1.22)$$

De la Eq.22 se obtiene un sistema de tres ecuaciones no lineales que relaciona las variables de las articulaciones ($\theta_{11}, \theta_{12}\ y\ \theta_{13}$) con la posición del efector móvil (P_X, P_Y, P_Z):

$$f_1 = (P_X - R - L_1 C_{\theta 11} + r)^2 + {P_Y}^2 + (P_Z - L_1 S_{\theta 1}\ )^2 - L_2^2 \qquad (1.23)$$

$$f_2 = \left(P_X - \tfrac{1}{2}(R - r + L_1 C_{\theta 12})\right)^2 + \left(P_Y - \tfrac{\sqrt{3}}{2}(R - r + L_1 C_{\theta 1}\ )\right)^2 + (P_Z - L_1 S_{\theta 12})^2 - L_2^2$$
(1.24)

$$f_3 = \left(P_X - \tfrac{1}{2}(R - r + L_1 C_{\theta 13})\right)^2 + \left(P_Y - \tfrac{\sqrt{3}}{2}(R - r + L_1 C_{\theta 1}\ )\right)^2 + (P_Z - L_1 S_{\theta 13})^2 - L_2^2$$
(1.25)

De la Eq.23 a la Eq.25 se puede escribir de la forma:

$$f_1 = {P_X}^2 + 2a_1 P_X + {P_Y}^2 + {P_Z}^2 - 2a_2 P_Z - a_3 \qquad (1.26)$$

Donde:

$$a_1 = r - R - L_1 c_{\theta 11}$$

$$a_2 = L_1 s_{\theta 11}$$

$$a_3 = L_2^2 - a_1^2 - a_2^2$$

$$f_2 = {P_X}^2 + b_1 P_X + {P_Y}^2 - \sqrt{3} b_1 P_Y + {P_Z}^2 - 2b_2 P_Z - b_3 \quad (1.27)$$

Donde:

$$b_1 = R - r + L_1 c_{\theta 12}$$

$$b_2 = L_1 s_{\theta 12}$$

$$b_3 = L_2^2 - b_1^2 - b_2^2$$

$$f_3 = {P_X}^2 + c_1 P_X + {P_Y}^2 - \sqrt{3} c_1 P_Y + {P_Z}^2 - 2c_2 P_Z - c_3 \quad (1.28)$$

Donde:

$$c_1 = R - r + L_1 c_{\theta 13}$$

$$c_2 = L_1 s_{\theta 13}$$

$$c_3 = L_2^2 - c_1^2 - c_2^2$$

Se resuelven las Eq.26 a Eq.28 para encontrar P_X y P_Y. Primero se restan las Eq.27 y Eq.28 de la Eq.26 respectivamente. De las dos ecuaciones resultantes, se despeja P_X y se igualan las ecuaciones para poder encontrar P_Y y luego P_X, obteniendo:

$$P_Y = \frac{A + B P_Z}{C} \quad (1.29)$$

Donde:

$$A = \frac{a_3 - b_3}{2a_1 - b_1} - \frac{a_3 - c_3}{2a_1 - c_1}$$

$$B = \frac{2(c_2 - a_2)}{2a_1 - c_1} - \frac{2(b_2 - a_2)}{2a_1 - b_1}$$

$$C = \frac{\sqrt{3} b_1}{2a_1 - b_1} - \frac{\sqrt{3} c_1}{2a_1 - c_1}$$

$$P_X = D - E P_Z \quad (1.30)$$

Donde:

$$D = \frac{a_3 - b_3}{2a_1 - b_1} - \frac{\sqrt{3} b_1 A}{C(2a_1 - b_1)}$$

$$E = \frac{2(b_2 - a_2)}{2a_1 - b_1} + \frac{\sqrt{3} b_1 B}{C(2a_1 - b_1)}$$

Sustituyendo la Eq.29 y Eq.30 en la Eq.26 se obtiene:

$$a P_Z^2 + b P_Z + c = 0 \quad (1.31)$$

Donde:

$$a = E^2 + \left(\frac{B}{C}\right)^2 + 1$$

$$b = 2\left(\frac{AB}{C^2} - DE - a_1E - a_2\right)$$

$$c = D^2 + \left(\frac{A}{C}\right)^2 + 2a_1D - a_3$$

La Eq.31 tiene dos posibles soluciones, pero como el sistema de referencia Delta es el centro de la plataforma fija, P_Z no puede tomar valores negativos, entonces:

$$P_Z = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad (1.32)$$

La cinemática directa e inversa solo son válidas para puntos a los que pueda acceder la plataforma móvil sin que sus componentes colisionen. Para detectar una colisión entre componentes del robot es necesario encontrar los ángulos β_i, θ_{2i} y θ_{3i}.

En la Fig.29 se aprecia que el vector $\overrightarrow{A_iC_i}$ es igual a la suma de los vectores $\overrightarrow{A_iB_i}$ y $\overrightarrow{B_iC_i}$ se puede expresar como:

$$\overrightarrow{A_iC_i} = \begin{bmatrix} P_X c_{\phi i} + P_Y s_{\phi i} + r - R \\ P_Y c_{\phi i} - P_X s_{\phi i} \\ P_Z \end{bmatrix} \qquad (1.33)$$

De la Fig.29 se observa que:

$$\overrightarrow{A_iB_i} + \overrightarrow{B_iC_i} = \begin{bmatrix} L_1C_{\theta 1i} + L_2C_{\theta 3i}C(180^\circ - \theta_{1i} - \theta_{2i}) \\ L_2S_{\theta 3i} \\ L_1S_{\theta 1i} + L_2C_{\theta 3i}S(180^\circ - \theta_{1i} - \theta_{2i}) \end{bmatrix} \qquad (1.34)$$

Igualando la Eq.33 y la Eq.34 se obtiene el valor de los ángulos θ_{3i} y θ_{2i}:

$$\theta_{3i} = \sin^{-1}\left(\frac{P_Y c_{\phi i} - P_Y s_{\phi i}}{L_2}\right) \qquad (1.35)$$

$$\theta_{2i} = 180^\circ - \sin^{-1}\left(\frac{P_Z - L_1 s_{\theta 1i}}{L_2 c_{\theta 3i}}\right) - \theta_{1i} \qquad (1.36)$$

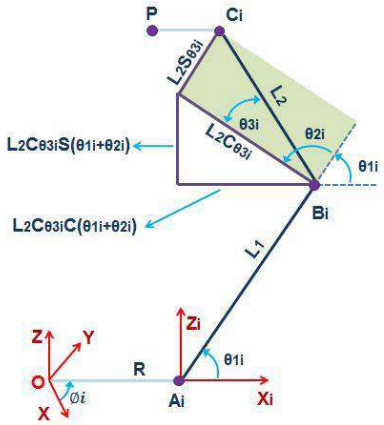


Fig. 29 Cadena cinemática i en las coordenadas $X_iY_iZ_i$

Para encontrar el ángulo β_i es necesario conocer la posición de la plataforma móvil con respecto a la plataforma fija, para encontrar la magnitud del vector A_iC_i y poder β_i despejar la Eq.10:

$$\beta_i = \cos^{-1}\left(\frac{L_1^2+L_2^2-A_iC_i^2}{2L_1L_2}\right) \qquad (1.37)$$

Se considera que no existen colisiones entre los componentes del robot si satisfacen las siguientes restricciones:

- $10^\circ \leq \beta_i \leq 180^\circ$
- $10^\circ \leq \theta_{2i} \leq 180^\circ$
- $-45^\circ \leq \theta_{3i} \leq 45^\circ$

La restricción del ángulo θ_{3i} depende de las junturas universales. Por lo tanto, el intervalo puede aumentar o disminuir una vez se mecanicen las junturas.

4.5 MATRIZ JACOBIANA

Las cinemáticas directa e inversa no tienen en cuenta las fuerzas o pares que actúen sobre el robot. La cinemática diferencial relaciona las velocidades angulares y lineales del efector con las velocidades en las articulaciones[12], [4], [34]–[36].

Gosselin[14] y Angeles plantean que una cadena cinemática cerrada se caracteriza por un vector de entrada θ n-dimensional, que corresponde a las articulaciones activas, y un vector de salida x m-dimensional, que corresponde a las coordenadas cartesianas de la plataforma móvil. La relación entre esos dos vectores:

$$F(\theta, x) = 0 \qquad (1.38)$$

Donde F es una función implícita n-dimensional de θ, x y 0 es un vector cero de n-dimensiones.

La rigidez de un manipulador paralelo en un punto de su espacio de trabajo se puede caracterizar por su matriz de rigidez. Esta matriz, relaciona las fuerzas y torques en el efector final con sus desplazamientos angulares y lineales. Diferenciando la Eq.38 con respecto al tiempo se obtiene una relación entre las velocidades de la entrada y la salida en un manipulador paralelo:

$$\dot{\theta} = Jv \qquad (1.39) \text{ (revisar jacobiano inverso)}$$

Donde $\dot{\theta}$ es el vector de las velocidades en las articulaciones activas, J es la matriz Jacobiana y v es un vector de seis dimensiones que contiene las velocidades lineales y angulares del efector final:

$$v = \begin{bmatrix} v_P \\ \omega_P \end{bmatrix} \qquad (1.40)$$

Gosselin y Angeles separan la matriz Jacobiana en dos matrices, una matriz asociada con la cinemática directa (J_X) y otra matriz asociada con la cinemática inversa (J_θ), donde:

$$J_X = \frac{\partial F}{\partial x} \quad \text{Y} \quad J_\theta = \frac{\partial F}{\partial \theta} \qquad (1.41)$$

Entonces, la matriz Jacobiana de un robot paralelo queda definida como $J = (J_\theta)^{-1} J_x$, la cual corresponde a la matriz Jacobiana inversa de un manipulador en serie. De esta forma, la Eq.39 se puede escribir:

$$\dot{\theta} = (J_\theta)^{-1} J_x \dot{x} \qquad (1.42)$$

Donde $\dot{\theta}$ es el vector de las velocidades de los actuadores y $\dot{x}$ es el vector de la velocidad del punto *P* en la plataforma móvil en el sistema de referencia *XYZ*. Entonces, la Eq.42 puede escribirse:

$$\begin{bmatrix} \dot{\theta_{11}} \\ \dot{\theta_{12}} \\ \dot{\theta_{13}} \end{bmatrix} = (J_\theta)^{-1} J_x \begin{bmatrix} V_{PX} \\ V_{PY} \\ V_{PZ} \end{bmatrix} \qquad (1.43)$$

Referente a la Fig.26, una ecuación del lazo cerrado de la i-esima cadena cinemática puede ser:

$$\overrightarrow{OP} + \overrightarrow{PC_1} = \overrightarrow{OA_i} + \overrightarrow{A_iB_i} + \overrightarrow{B_iC_i} \qquad (1.44)$$

Derivando la Eq.44 con respecto al tiempo y expresando el resultado en el sistema $X_iY_iZ_i$:

$$\overrightarrow{V_P} = \overrightarrow{\omega_{1i}}\, x\, \overrightarrow{A_iB_i} + \overrightarrow{\omega_{2i}}\, x\, \overrightarrow{B_iC_i} \qquad (1.45)$$

Donde $\overrightarrow{V_P}$ es la velocidad lineal de la plataforma móvil y ω_{ji} es la velocidad angular del j-esimo eslabón de la i-esima cadena cinemática.

De la Eq.45 las velocidades $\overrightarrow{\omega_{2i}}$ no son de interés debido a que son las velocidades de las articulaciones pasivas[2], por lo tanto, se eliminan multiplicando ambos lados de la ecuación por $\overrightarrow{B_iC_i}$:

$$\overrightarrow{B_iC_i} * \overrightarrow{V_P} = \overrightarrow{B_iC_i}\left(\overrightarrow{\omega_{1i}}\, x\, \overrightarrow{A_iB_i}\right) + \overrightarrow{B_iC_i}\left(\overrightarrow{\omega_{2i}}\, x\, \overrightarrow{B_iC_i}\right)$$

Donde $\overrightarrow{B_iC_i}\left(\overrightarrow{\omega_{2i}}\, x\, \overrightarrow{B_iC_i}\right) = 0$ por la propiedad de *cancelación por ortogonalidad* del producto cruz. Entonces:

$$\overrightarrow{B_iC_i} * \overrightarrow{V_P} = \overrightarrow{B_iC_i}\left(\overrightarrow{\omega_{1i}}\, x\, \overrightarrow{A_iB_i}\right) \qquad (1.46)$$

Donde:

[2] Junturas Universales

$$\overrightarrow{B_\iota C_\iota} = \begin{bmatrix} L_2 C_{\theta 3i}(\theta_{1i} + \theta_{2i}) \\ L_2 S_{\theta 3i} \\ L_2 C_{\theta 3i} S(\theta_{1i} + \theta_{2i}) \end{bmatrix}, \overrightarrow{V_P} = \begin{bmatrix} V_{pX} C_{\phi i} + V_{pY} S_{\phi i} \\ V_{pY} C_{\phi i} - V_{pX} S_{\phi i} \\ V_{PZ} \end{bmatrix}, \overrightarrow{\omega_{1\iota}} = \begin{bmatrix} 0 \\ -\dot{\theta_{1\iota}} \\ 0 \end{bmatrix} \text{ y } \overrightarrow{A_\iota B_\iota} = \begin{bmatrix} L_1 C_{\theta 1i} \\ 0 \\ L_1 S_{\theta 1i} \end{bmatrix}$$

Resolviendo la Eq.46 y simplificando se obtiene:

$$j_{xi1} V_{pX} + j_{xi}\ V_{pY} + j_{xi3} V_{pZ} = j_{\theta i} \dot{\theta_{1J}} \qquad (1.47)$$

Donde:

$$j_{xi1} = C_{\theta 3i} C(\theta_{1i} + \theta_{2i}) C_{\phi i} - S_{\theta 3i} S_{\phi i}$$

$$j_{xi2} = C_{\theta 3i} C(\theta_{1i} + \theta_{2i}) S_{\phi i} + S_{\theta 3i} C_{\phi i}$$

$$j_{xi}\ = C_{\theta 3i} S(\theta_{1i} + \theta_{2i})$$

$$j_{\theta i} = L_1 S_{\theta 2i} C_{\theta 3i}$$

La Eq.47 representa un sistema de tres ecuaciones que se puede escribir de la forma $J_x \dot{x} = J_\theta \dot{\theta}$:

$$\begin{bmatrix} j_{x11} & j_{x12} & j_{x13} \\ j_{x21} & j_{x22} & j_{x23} \\ j_{x31} & j_{x32} & j_{33} \end{bmatrix} \begin{bmatrix} V_{pX} \\ V_{pY} \\ V_{pZ} \end{bmatrix} = \begin{bmatrix} j_{\theta 1} & 0 & 0 \\ 0 & j_{\theta 2} & 0 \\ 0 & 0 & j_{\theta 3} \end{bmatrix} \begin{bmatrix} \dot{\theta_{11}} \\ \dot{\theta_{12}} \\ \dot{\theta_{13}} \end{bmatrix} \qquad (1.48)$$

Entonces, la matriz jacobiana del robot Delta es:

$$J = (J_\theta)^{-1} J_x = \left(\begin{bmatrix} j_{\theta 1} & 0 & 0 \\ 0 & j_{\theta 2} & 0 \\ 0 & 0 & j_{\theta 3} \end{bmatrix} \right)^{-1} \begin{bmatrix} j_{x11} & j_{x12} & j_{x1} \\ j_{x21} & j_{x22} & j_{x23} \\ j_{x31} & j_{x32} & j_{33} \end{bmatrix} \qquad (1.49)$$

Dependiendo de cuál matriz es singular, un mecanismo de lazo cerrado puede tener una configuración singular de cinemática directa, una configuración singular de cinemática inversa, o ambas.

La singularidad es uno de los mayores problemas en un robot paralelo debido a que en esas configuraciones el robot no puede ser controlado y existen infinitas fuerzas/torques en sus junturas, lo que puede generar su ruptura.

4.6 SINGULARIDADES DE LA CINEMÁTICA INVERSA

Se dice que existe una singularidad de la cinemática inversa cuando el determinante de J_θ tiende a cero:

$$det(J_\theta) = (S_{\theta 21} C_{\theta 31})(S_{\theta 22} C_{\theta 32})(S_{\theta 23} C_{\theta 33}) = 0 \qquad (1.50)$$

Esto implica que movimientos infinitesimales de la plataforma móvil a lo largo de ciertas direcciones no pueden realizarse, es decir, el manipulador es capaz de resistir fuerzas y torques en esas direcciones[8], [3], [37]–[41]. Por lo tanto, es manipulador pierde uno o más

DOF. Generalmente, estas singularidades ocurren en los límites del espacio de trabajo del manipulador.

Para que se cumpla la Eq.50 $S_{\theta 2i} = 0$ o $C_{\theta 3i} = 0$, por lo tanto, esta singularidad ocurre cuando se satisface alguna de las siguientes condiciones:

- $\theta_{2i} = 0^{\circ}\ o\ 180^{\circ}$
- $\theta_{3i} = 90^{\circ}\ o\ 270^{\circ}$

4.7 SINGULARIDADES DE LA CINEMÁTICA DIRECTA

Se dice que existe una singularidad de la cinemática directa cuando el determinante de J_x tiende a cero:

$$det(J_x) = 0$$

La plataforma móvil posee movimientos infinitesimales en algunas direcciones mientras los actuadores están bloqueados completamente, es decir, la velocidad de la plataforma móvil es diferente de cero a pesar que las velocidades son cero.

4.8 COMBINACIÓN DE SINGULARIDAD

Se dice que las singularidades son combinadas cuando los determinantes de J_θ y de J_x tienden a cero. Este tipo de singularidad es común en robots espaciales. La plataforma móvil puede experimentar algunos movimientos infinitesimales mientras los actuadores están fijos o por el contrario permanecer inmóvil mientras los actuadores presentan movimientos infinitesimales.

4.9 DESTREZA DEL ROBOT DELTA

La matriz Jacobiana permite identificar las configuraciones no deseadas en un robot Delta (singularidades)[1], [42]–[45]. Adicionalmente, permite conocer el nivel de destreza de cada configuración del robot por medio del número de condición de la matriz Jacobiana:

$$\lambda = \|J\| * \|J^{-1}\| \qquad (1.52)$$

Debido a que la el número de condición de una matriz se define como la norma-2 de la matriz multiplicada por la norma-2 de la matriz inversa.

La destreza local de un robot paralelo se define como la inversa del número de condición de la matriz Jacobiana:

$$\varsigma = \frac{1}{\lambda}, \quad 0 \leq \varsigma \leq 1 \qquad (1.53)$$

Si el valor de ς es cero es porque el robot está en una configuración singular y si el valor de ς es uno, la configuración del robot se conoce como isotrópica.

En vista que el índice ς permite analizar únicamente cada configuración del robot paralelo, Gosselin y Angeles en 1991, plantearon un índice que permite medir la destreza de un robot en todo su espacio de trabajo (índice global):

$$\eta = \frac{\int_W \varsigma dW}{\int_W dW} \qquad (1.54)$$

Donde el numerador $\int_W \varsigma dW$ representa la sumatoria de las destrezas locales del robot Delta en todo su espacio de trabajo y el denominador $\int_W dW$ es el valor del espacio de trabajo del robot.

4.10 ESPACIO DE TRABAJO DEL ROBOT DELTA

El espacio de trabajo de un robot se considera como la región en el plano cartesiano tridimensional que puede ser alcanzada por un punto de su efector final, en el caso de un robot Delta, es la región en el espacio tridimensional que puede alcanzar un punto de su plataforma móvil.

Al inicio, los investigadores solo se preocuparon por encontrar el espacio de trabajo de los robots paralelos, es por ello, que en 1994 Merlet presentó un algoritmo para determinar si la trayectoria de un manipulador paralelo entre dos puntos se podía realizar, con el fin de verificar que las trayectorias del manipulador estuvieran dentro de su espacio de trabajo[46], [5].

Diseñar un robot Delta para un máximo espacio de trabajo no garantiza que el robot vaya a ser óptimo para aplicaciones prácticas y es factible que resulte un manipulador con características cinemáticas no deseadas como lo son una baja destreza y maniobrabilidad. Por este motivo, esta investigación se centra en diseñar un robot Delta que logre alcanzar la mayor cantidad de puntos en el espacio.

Existen básicamente dos métodos para encontrar el espacio de trabajo de un manipulador paralelo: el método geométrico y el método de discretización, siendo este último el más usado. El método de discretización, que se basa en métodos numéricos, consiste en discretizar el espacio en tres dimensiones, resolviendo la cinemática inversa para cada punto y verificando las restricciones que limitan dicho espacio de trabajo. El método numérico más conocido para encontrar el espacio de trabajo de un manipulador paralelo es el método de Monte Carlo y consiste en:

1. Definir un volumen de trabajo ($\boldsymbol{V}$) que encierre el volumen de trabajo del robot Delta.

2. Generar un gran número ($\boldsymbol{n_{total}}$) de puntos, seleccionados aleatoriamente, que se encuentren dentro del Volumen $\boldsymbol{V}$.

3. Utilizar la cinemática inversa para evaluar cada punto y determinar si se encuentra dentro del espacio de trabajo del robot. Una vez se evalúan todos los puntos, se obtiene el número total de puntos dentro del espacio de trabajo del robot ($\boldsymbol{n_{in}}$).

4. El volumen del espacio de trabajo de robot (Fig.30) se calcula multiplicando el volumen de trabajo $\boldsymbol{V}$ con el número total de puntos que alcanzo el robot y dividiendo por el número total de puntos seleccionados $\boldsymbol{V' = \frac{n_{in}}{n_{total}} V.}$

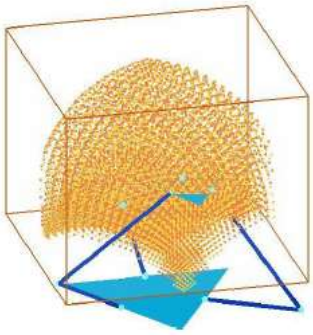


Fig. 30 Espacio de Trabajo del Robot Delta, método de Monte Carlo.

Como se aprecia en las Fig.31 y Fig.32, el espacio de trabajo del robot Delta depende de:

- Las dimensiones del Robot $(L_1, L_2, R\ y\ r)$.
- Las restricciones de los actuadores $(\theta_{11}, \theta_{12}\ y\ \theta_{13})$.
- Las restricciones de las junturas universales $(\theta_{21}, \theta_{22}, \theta_{23}, \theta_{31}, \theta_{32}\ y\ \theta_{33})$.

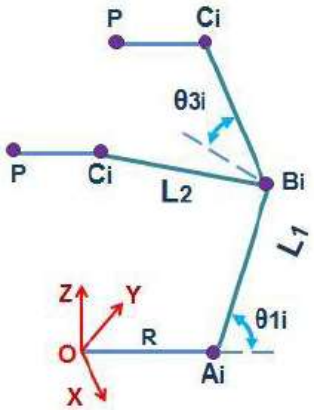


Fig. 31 Restricciones del ángulo θ_{i3}.

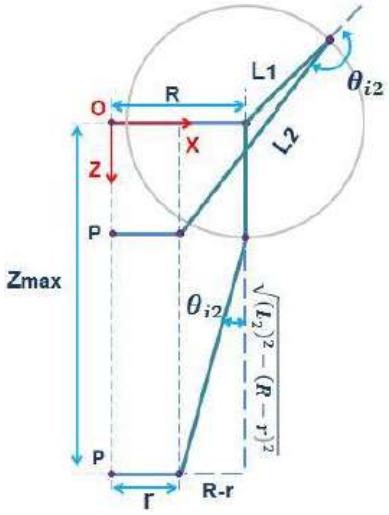


Fig. 32 Restricciones del ángulo θ_{i2}.

Adicionalmente, en la Fig.32 se observa que la plataforma móvil alcanza su altura máxima cuando las variables de los tres actuadores son 90°:

$$P_{zmax} = L_1 + \sqrt{(L_2)^2 - (R-r)^2} \quad (1.55)$$

La exactitud del volumen de trabajo del robot Delta, depende de la probabilidad de que los puntos seleccionados aleatoriamente puedan ser alcanzados por el robot. Por esta razón, en este trabajo de grado se plantea un algoritmo para calcular la cantidad exacta de los puntos en el espacio que es capaz de alcanzar la plataforma móvil del robot. El algoritmo se basa en la solución de la cinemática directa para todas las posibles combinaciones de los actuadores, obteniendo en cada caso un punto en el espacio. El punto pertenece al espacio de trabajo del robot sí y solo sí:

- $\theta_{2i} \in [10°, 180°)$ y $\theta_{3i} \in [-45°, 45°]$ para $\boldsymbol{i = 1, 2\ y\ 3.}$

La Fig.33 muestra el diagrama de flujo del algoritmo.

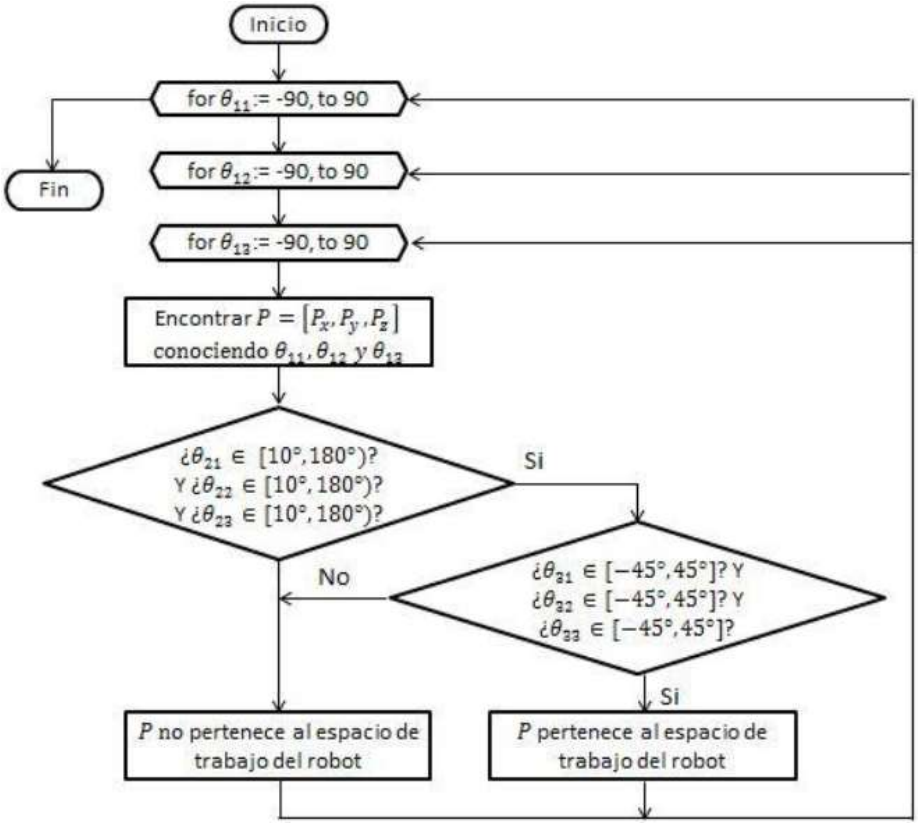


Fig. 33 Diagrama de flujo del programa del espacio de trabajo del robot.

El paso incremental de las variables de los actuadores $(\theta_{11}, \theta_{12}\ y\ \theta_{13})$ depende de la sensibilidad de los actuadores. Entre menor sea este incremento, más exacto es el espacio de trabajo que se obtiene. Para encontrar la destreza global del robot, primero se calcula el índice local de destreza de cada punto que pertenece a su espacio de trabajo, luego se suman todos los índices locales y se dividen por la cantidad de puntos que el efector final del robot es capaz de alcanzar en el espacio.

4.11 CONCLUSION

Por medio de este capítulo se otorgó al lector una visión general del análisis cinemático de una plataforma delta, de igual forma se hace necesario este análisis, ya que la cinemática inversa del sistema se requiere para la implementación del mismo en el algoritmo del sistema.

CAPITULO V:

IDENTIFICACION DE LA MANO TRATAMIENTO DE IMAGEN DEL KINECT

5. CAPITULO V: IDENTIFICACION DE LA MANO TRATAMIENTO DE IMAGEN DEL KINECT

5.1 VISION GENERAL

Hasta este punto se obtuvo un robot delta con soporte, y se ha construido un análisis cinemático de la plataforma. Ahora bien, el paso a seguir es la creación del software que corresponda a los datos captados y entregados por Kinect y la Plataforma física. Esta función es cumplida por el boceto de Processing, que le indica a los tres servos ir a ciertas posiciones con el fin de mover el efector a un punto preciso en el espacio.

El usuario mediante un movimiento de la mano derecha dentro del área de visión de la plataforma Kinect buscara desplazar el robot Delta. Los datos captados por el Kinect serán entregados al PC para realizar el tratamiento del mismo por el Software Processing, según los datos analizados junto a las acciones por parte del usuario, se envían órdenes o comandos específicos mediante bits a la tarjeta de desarrollo Arduino UNO Versión R3.

5.2 KINECT

El sensor de Kinect® es una barra horizontal de aproximadamente 23 cm, conectada a una pequeña base circular con un eje de articulación de rótula. El dispositivo cuenta con una cámara RGB, un sensor de profundidad, un micrófono de múltiples matrices y un procesador que proporciona captura de movimiento de los objetos en 3D, reconocimiento facial y capacidades de reconocimiento de voz. El sensor de profundidad es un proyector de infrarrojos combinado con un sensor CMOS monocromo que permite a Kinect® ver recintos en 3D en cualquier condición de luz ambiental. El rango de detección de la profundidad del sensor es ajustable por software pero con limitaciones de zona que observaremos más adelante.

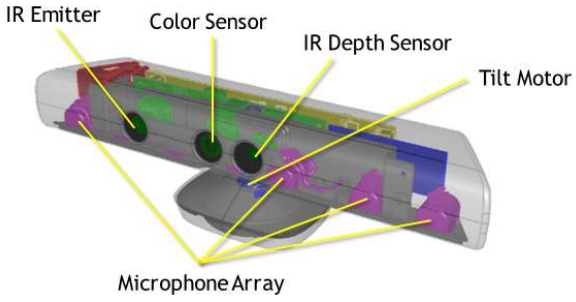


Fig. 34 Secciones principales del Sensor Kinect®

El sensor de Kinect® adquiere imágenes de video con un sensor CMOS de colores a una frecuencia de 30 Hz, en colores RGB de 32 bits y resolución VGA de 640x480 pixeles. El canal de video monocromo CMOS es de 16 bits, resolución QVGA de 320x240 pixeles con hasta 65.535 niveles de sensibilidad[47].

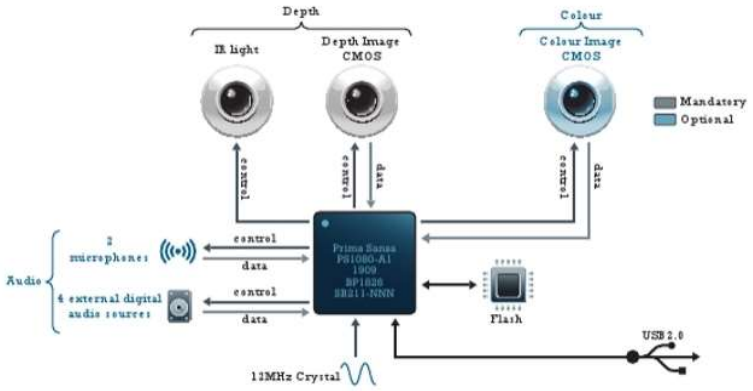


Fig. 35 Composición interna del Sensor Kinect®

Para calcular distancias entre un cuerpo y el sensor, el sensor un haz laser infrarrojo que proyecta un patrón de puntos sobre los cuerpos cuya distancia se determina. Una cámara infrarroja capta este patrón y por hardware calcula la profundidad de cada punto. El rango de profundidad del sensor de Kinect® esta entre 0.4 y 4m, como se muestra en la Fig.36 existen dos modos; default y Near para determinar distancias. La cámara de Kinect® funciona con Hardware y Software propios para el reconocimiento de imagen, esta posee dos funciones principales:

- Generar un mapa en 3D de la imagen que tiene en su campo visual.

- Reconocer humanos en movimiento entre los objetos de la imagen a partir de diferentes segmentos de las articulaciones del cuerpo y un esquema en escala de grises.

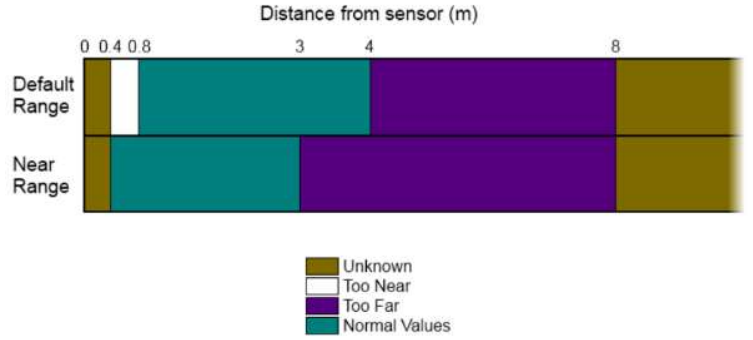


Fig. 36 Distancias de detección Kinect modo por defecto y modo Near

El procesador es capaz de interpretar los movimientos que se registran de los objetos capturados por la cámara de Kinect® en "eventos con significado" que aparecen en pantalla[48]. Los movimientos buscados por el algoritmo son contextualizados. Por ejemplo, si se está aplicando el sensor Kinect® a un juego como Kinect® Adventures, donde una balsa desciende por la corriente del río, dado que este juego requiere movimientos tales como

agacharse o tumbarse, el algoritmo buscará la identificación de estos movimientos en tiempo real para producir eventos en pantalla.

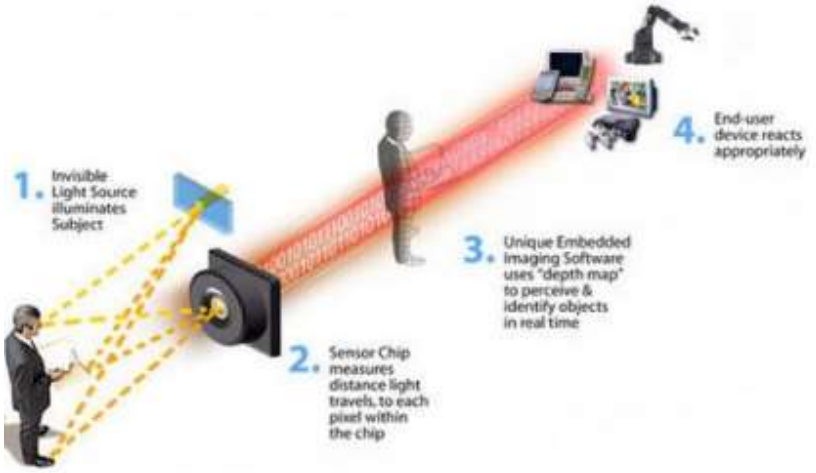


Fig. 37 Composición interna del Sensor Kinect®

El dispositivo como tal, está compuesto por tres partes funcionales:

- Sistema de rastreo (tracking).
- Reconocimiento de voz.
- Motor de rotula.

5.2.1 SISTEMA DE RASTREO

El sistema óptico de Kinect®, permite seguir movimientos en tiempo real. Está formado por la integración de tecnología que se ha venido desarrollando desde hace 15 años; pero con efectos y funcionalidades que se han descubierto recientemente. Está conformada por dos partes: un proyector y una cámara VGA infrarroja, los cuales funcionan proyectando un láser (el cual Microsoft declara que es seguro) a través de toda el área donde se encuentra el dispositivo generando lo que se llama 'depth field' (campo de profundidad). Esencialmente, todos los pixeles que Kinect® recibe como ruido de IR son convertidos en una escala de colores, haciendo que los cuerpos dependiendo de la distancia se capturen como rojos, verdes, azules hasta llegar a tonos grises, que representan a objetos muy lejanos[48].

Debido a las características del dispositivo y el software SDK asociado al mismo, las imágenes pueden ser capturadas y haciéndolas pasar por una serie de filtros para que Kinect® determine que es una persona y que no lo es. Para esto se siguen una serie de parámetros relacionados con las longitudes y características propias del cuerpo humano; por lo cual, con estos modelos en procesamiento, se facilita el reconocimiento y captura de cada parte del cuerpo. Una vez que la información es separada del resto, se convierte cada identificación del cuerpo en un esqueleto con articulaciones móviles. Kinect® está precargado con una base de datos de 200 poses, así que puede llenar los espacios si haces un movimiento que obstruya

la visión de la cámara. Por otro lado, Kinect® efectúa cada una de estas tareas en un tiempo total de 30 cuadros por segundo[48].

5.2.2 RECONOCIMIENTO DE VOZ

Inicialmente, el subsistema de micrófonos presentaba a limitante de capturar e interpretar sonidos a solo 3 metros de la consola, ignorando ruidos ambientales y otros sonidos diferentes de la voz humana en el proceso. Para solucionar este problema, Microsoft incorporó un arreglo de micrófonos localizados en las áreas laterales del dispositivo, específicamente uno a la izquierda y tres a la derecha; esta es la razón por la cual el dispositivo es tan ancho. Este arreglo de micrófonos es óptimo para capturar voces a la distancia, pero requiere ayuda de un procesador de audio, encargado de cancelar los ruidos provenientes del entorno externo al dispositivo, mientras que un software llamado 'Beam Forming' trabaja en conjunto con la cámara para averiguar la posición del usuario y crear una esfera de sonido que lo envuelva, permitiendo enfocarse solo en el sonido del mismo, ignorando cualquier otro. Kinect® tiene además un modelo acústico para cada país y dialectos para regiones específicas, construido gracias a cientos de actores alrededor del mundo, cuyas voces fueron procesadas al hablar de varias formas.

Igual que con el sistema óptico, la captura de información se realiza todo el tiempo, pues el sistema de sonido trabaja en un sistema de micrófono abierto, haciendo que los micrófonos estén a la escucha durante todo el tiempo[48].

5.2.3 MOTOR

Este elemento es capaz de mover la unidad verticalmente hacia abajo y hacia arriba con 30 grados de libertad. Para garantizar un movimiento limpio y libre de ruido, se efectuaron pruebas de esfuerzo al motor bajo calor extremo, uso prolongado (cientos de tilts al día durante varios meses), y se verifico que la calibración sea exacta (grado por grado de inclinación). Todo esto, realizado en un cuarto libre de ruido, creado para asegurarse de que el motor y el proceso de tilting no fueran oídos por los usuarios, pues Microsoft insiste que el movimiento del motor solo deberá generar 24 decibeles de ruido, en un cuarto promedio con 40 decibeles. El motor permite operar al mismo tiempo el sistema de zoom de la cámara, permitiendo expandir el área de juego. Aparte de todas las características anteriormente mencionadas, se encuentra presente un ventilador que entrará en acción cuando las condiciones ambientales lo ameriten, sin interferir en la acción de los micrófonos[47].

5.2.4 PRINCIPIO DE FUNCIONAMIENTO KINECT

Lo que convierte al dispositivo en un elemento simple y sencillo que incorpora todos los elementos necesarios para examinar los patrones de luz sin problemas de interferencia de luz externa, es la forma en que aprovecha la interacción entre el proyector de infrarrojos y la cámara IR CMOS. A continuación se detalla una breve explicación de cómo funciona este dispositivo en la Fig.38.

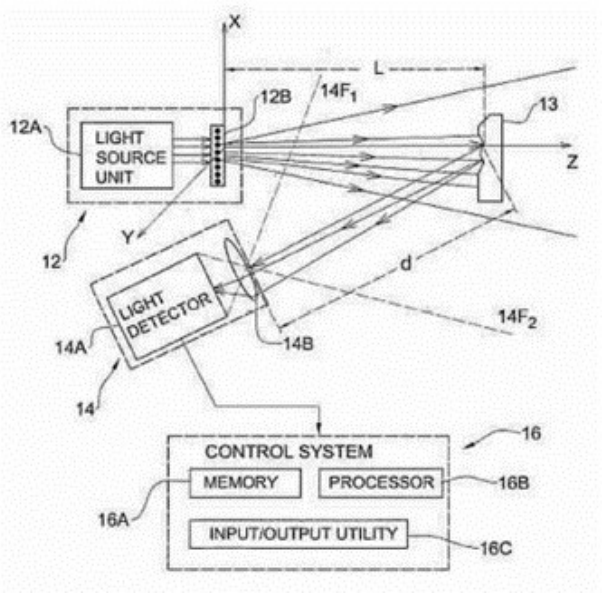


Fig. 38 Principios de Kinect®

En primer lugar se proyecta un patrón conocido (en forma de motas o puntos) sobre un entorno u objeto(s) cercano disponiendo una escena, luego la cámara de IR CMOS se encarga de observar la escena proyectada; la proyección es generada por un difusor y un elemento difractor de luz infrarroja. Finalmente, para poder efectuar esta operación es necesario que el proyector y la cámara estén calibrados adecuadamente[47].

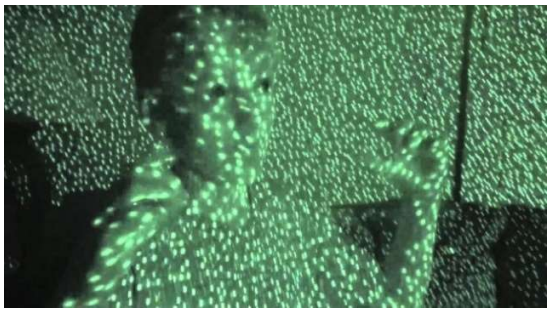


Fig. 39 Barrido espacial Sensor Kinect®

5.2.5 DATOS DE PROFUNDIDAD

Para el cálculo de los datos de profundidad, el sensor efectúa una triangulación de cada punto, entre una imagen virtual (patrón) y el patrón observado. Cada punto o mancha tiene asociado un correspondiente punto o mancha dentro del sensor, esto gracias a que Kinect® incorpora diferentes patrones de imágenes y de sonido de diferentes personas almacenadas en un procesador dentro del dispositivo, facilitando la captación por parte del sensor.

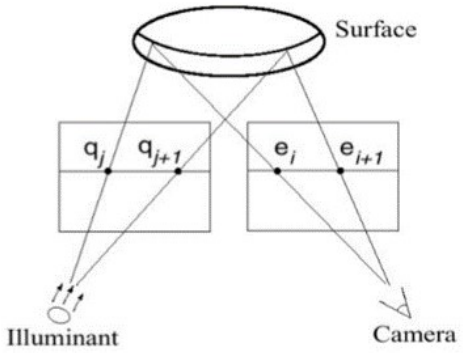


Fig. 40 Triangulación de cada punto, entre una imagen patrón y la imagen capturada.

Una vez se tenga calibrado el punto patrón, se calcula el mapa 3D del cuadro (o frame) inicial y la dirección x asociada a los cambios del punto observado, esto con el fin de renovar el mapa 3D. La calibración del dispositivo se lleva a cabo en el momento de la fabricación del mismo, al igual que el conjunto de imágenes de referencia que fueron tomadas en diferentes lugares y posteriormente almacenadas en memoria.

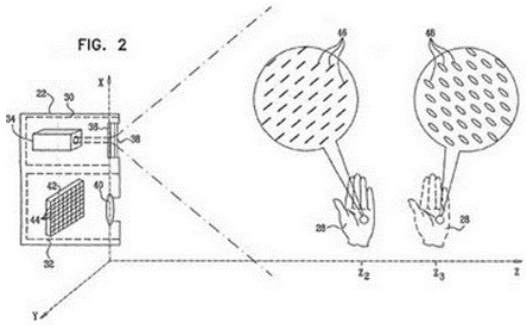


Fig. 41 Influencia de la distancia y orientación del sensor en la detección de la imagen.

El tamaño y forma de los puntos o manchas, dependerá mucho de la distancia y orientación propia del sensor, además de que el dispositivo utiliza 3 diferentes tamaños de puntos o manchas sobre 3 regiones a diferentes distancias; por tal motivo entre más cerca este el objeto habrá una mayor exactitud y entre más lejos se encuentre habrá una menor exactitud, como se puede observar en la Fig.42, en donde tanto el difusor como el elemento difractor, se encargan de generar los patrones de puntos sobre tres regiones concretas.

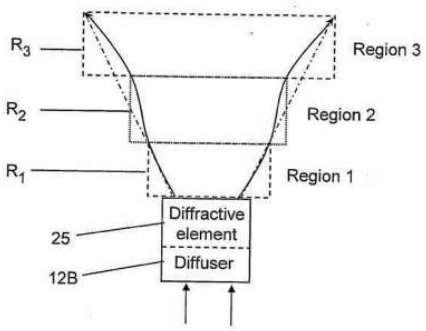


Fig. 42 Regiones de captura de imagen.

Acorde a la región en donde se encuentre el objeto de análisis, el patrón de puntos será diferente, por lo cual:

- La Región 1, permite obtener una superficie de profundidad de alta precisión para objetos ubicados a una distancia aproximada de 0,8 a 1,2 m del sensor.

- La Región 2, permite obtener una superficie de profundidad de precisión media para objetos ubicados a una distancia aproximada de 1,2 a 2 m del sensor.

- La Región 3, permite obtener una superficie de profundidad de baja precisión para objetos ubicados a una distancia aproximada de 2 a 3,5 m del sensor.

Por requerimientos del proyecto, se optó por trabajar en un rango comprendido entre la región 1 y la región 2, con el fin de obtener patrones adecuados para el rango de operación.

5.3 PROCESSING

Fig. 43 Logo de apertura software Processing.

Processing es un lenguaje de programación de código abierto y se utiliza como entorno de desarrollo para las personas que desean crear imágenes, animaciones e interacciones. Processing es desarrollado inicialmente para servir como un cuaderno de bocetos de software y para enseñar los fundamentos de la programación informática en un contexto visual, processing también se ha convertido en una herramienta para la generación de trabajos profesionales. Hoy en día, hay decenas de miles de estudiantes, artistas, diseñadores, investigadores y aficionados que utilizan processing para el aprendizaje, creación de prototipos y producción. Processing se basa en Java, es uno de los lenguajes de programación más extendidos en la actualidad. Java es un lenguaje de programación orientado a objetos, multi-plataforma. El código que se escribe en Java se compila en bytecode que es ejecutado luego por la Java Virtual Machine que vive en su ordenador.

Esto permite al programador escribir software sin tener que preocuparse por el sistema operativo (OS) que se ejecuta en él. Como se puede deducir, esto es una gran ventaja cuando se está tratando de escribir software para ser utilizado en diferentes máquinas[3]. Processing

[3] www.processing.org/

fue iniciado por Ben Fry y Casey Reas en la primavera de 2001, como estudiantes de posgrado en el MIT Media Lab dentro de la dirección de John Maeda y el grupo de investigación en Computación. El desarrollo continuó, mientras que Casey continuó con su carrera artística y de docencia y Ben siguió un doctorado fundando Fathom Information Design.

Processing y sus proyectos hermanos han inspirado a más de veinte libros educativos. Con el lanzamiento de procesamiento 2.0, se dio un nuevo paso adelante y establecieron la Fundación de Processing. Hasta este punto, producción se ha desarrollado casi exclusivamente por voluntarios. La financiación es esencial para apoyar la gran base de usuarios de Processing y mantener la alta calidad del software, dicha financiación viene de aportes de usuarios y empresas desarrolladoras de software.

El enfoque principal de la Fundación Processing es desarrollar y distribuir el software de procesamiento, tanto la interfaz de programación de aplicaciones de núcleo (API), el entorno de programación y el entorno de desarrollo de Procesamiento (PDE). Para cumplir con este cargo, se invita a donaciones de individuos y organizaciones para ayudar en el desarrollo del software de procesamiento.

5.4 ANALISIS CINEMATICO INVERSO A NIVEL DE SOFTWARE

En primer lugar, se desarrolló las clases[4] del robot delta y test de ellos con ejemplos sencillos. Se sugiere que cuando esté seguro que la simulación está funcionando adecuadamente, se debe agregar todos los métodos de Kinect y de comunicación serie, y luego llevarlo a la práctica mediante el programa de Arduino que traducirá los datos en serie en impulsos que los servos pueden entender. Pero inicialmente, se debe aclarar un concepto importante.

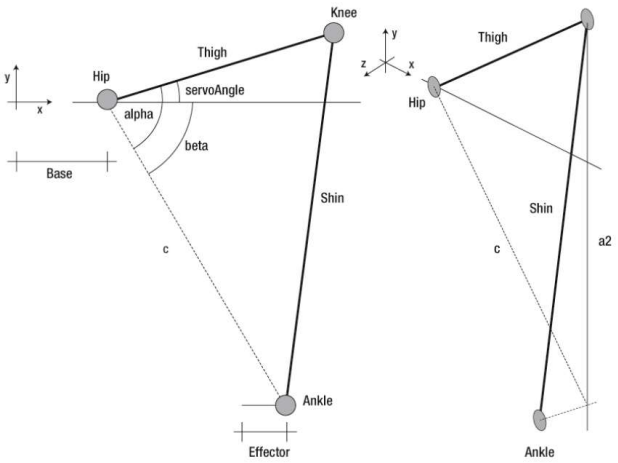


Fig. 44 Análisis geométrico Robot delta

Cuando se trabaja con robots, a menudo es necesario hacer uso de la cinemática inversa. Cuando se tiene un objeto flexible, articulado, o de cadena cinemática (como su robot delta,

[4] Secciones, estructuras o funciones de código

o cualquier otro robot articulado), hay dos problemas que normalmente necesita para resolver la hora de estudiar sus movimientos.

- Si me traslado todos los motores o actuadores en la estructura a ciertos estados, ¿cuál es la postura que resulta del robot?
- Si quiero conseguir una pose específica, ¿cuáles son los estados requeridos de mis actuadores en para lograrlo?

La primera pregunta puede ser respondida por cinemática directa, el segundo por la cinemática inversa. En este caso, se enfocó en llevar el efector del robot a puntos específicos en el espacio para que pueda seguir la mano; esto está dentro de la esfera de la cinemática inversa. Se tiene una estructura bastante sencilla, impulsado por sólo tres servos (los otros dos son sin relación con la posición del robot en el espacio), pero hay que saber cómo las tres coordenadas espaciales independientes de x, y, z que define la posición deseada del efector se pueden traducir de nuevo a través de la cadena cinemática de las tres rotaciones de los servos. Esto se logra mediante el análisis de la conexión y la geometría del robot y las limitaciones que esta geometría impone sobre la cinemática del robot tal como se observó en el capítulo anterior.

El objetivo de la cinemática inversa de una pierna es encontrar la rotación del servo necesario para alcanzar una posición efectora específica en el espacio. Supongamos por un segundo que se está tratando con un problema bidimensional simple. Si usted necesita saber el ángulo del servo para el diagrama de Fig.44, la respuesta se obtiene a través de la geometría plana. El código que proporciona la respuesta es el siguiente:

```
float c = dist (posTep.x+effectorSize, posTemp.y, baseSize, 0);
float alpha = acos((-a2 + thigh*thigh + c * c) / (2*thigh*c));
float beta = -atan2(posTemp.y, posTemp.x);
servoAngle = alpha – beta;
```

Pero el análisis como se mencionó anteriormente no puede obedecer a un mundo de dos dimensiones, por lo que necesita para resolver el problema de una estructura tridimensional. Si observamos bien en la Fig.44, se puede ver que el problema se puede reducir a un problema de dos dimensiones en el que la dimensión se reduce a su proyección sobre el plano del servo de rotación. Esta proyección, llamada ***a2*** en el código, se obtiene mediante la teoría del triángulo simple.

```
float a2 = shin * shin – posTemp.z * posTemp.z
```

Una vez que se conoce a2, sólo tiene que sustituirlo en las ecuaciones anteriores y se obtiene la respuesta al problema tridimensional[5]. Posteriormente se tiene que estructurar las clases del robot delta.

```
float c = dist(posTemp.x + effectorSize, posTemp.y, baseSize, 0);
float alpha = acos((-a2 + thigh * thigh + c * c) / (2 * thigh * c));
float beta = -atan2(posTemp.y, posTemp.x);
servoAngle = alpha - beta;
```

5.5 DIAGRAMA DE FLUJO GENERAL DEL ALGORITMO

A continuación se da a conocer el diagrama de flujo general del algoritmo, la cual será explicada en especificidad en las siguientes secciones:

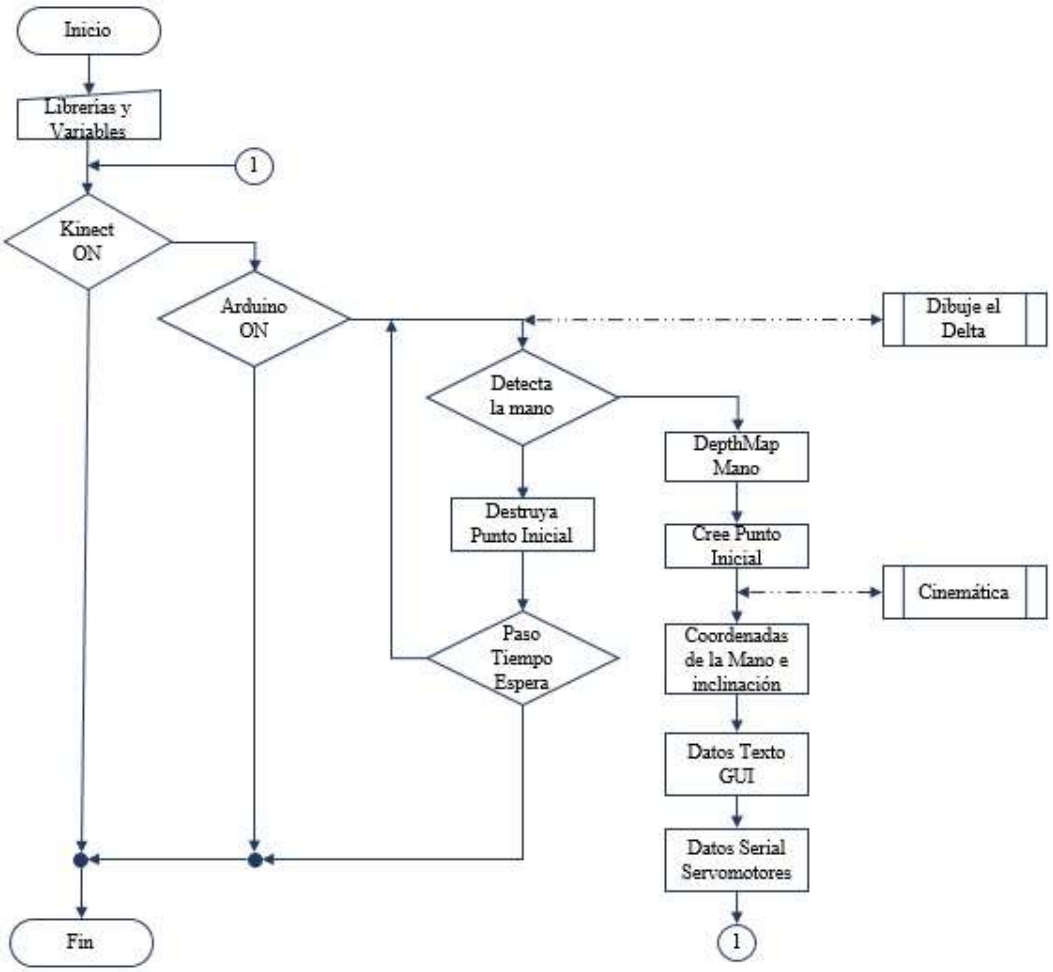


Fig. 45 Diagrama de Flujo

[5] Otra ventaja de los robots delta es el grado en que se han estudiado y la disponibilidad de buena información en línea acerca de ellos, si requiere mayor información del estudio de cinemática retorne al capítulo IV.

5.6 CLASS DELTAROBOT

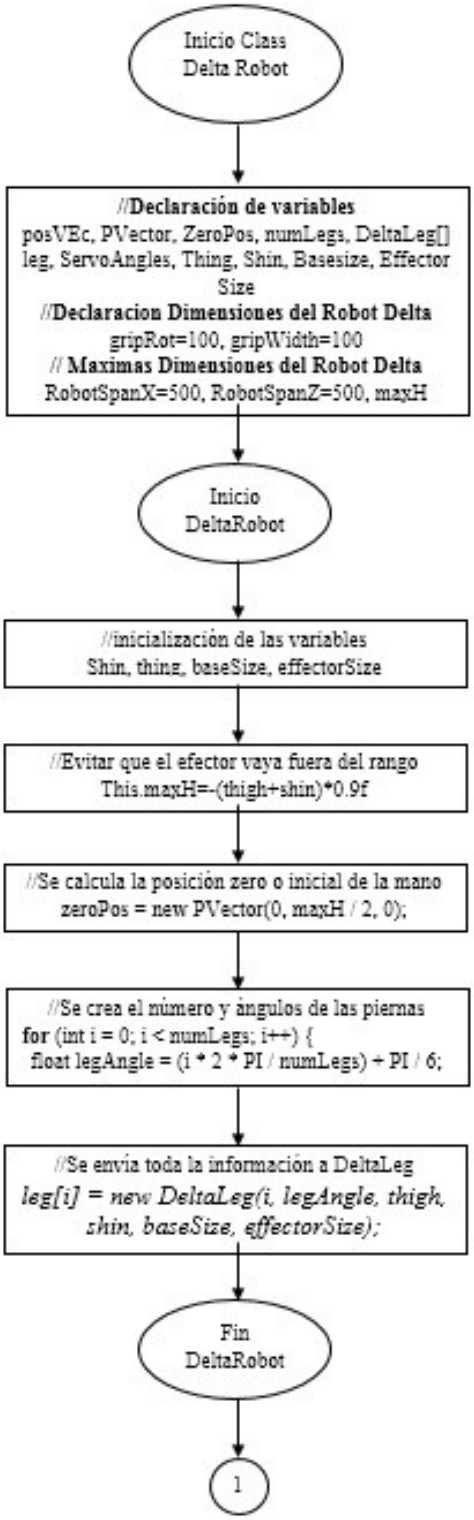


Fig. 46 Sección 1 Diagrama de flujo class DeltaRobot

La simulación robot delta consiste en la definición de la función ***class DeltaRobot*** Fig.46, que contiene los principales parámetros y rutinas, y una función ***class DeltaLeg***, en las que se ocupan de la cinemática inversa y las funciones de dibujo para cada pierna. Se va a iniciar a explicar el ***class DeltaRobot (Apoyados en la Fig.43 y el Apéndice A)***. Se necesita una serie de instrucciones que definirán la posición del efector en cada momento definido en el tiempo, y un ***zeroPos PVector*** que establece la posición de la cual se definen los movimientos de su efector.

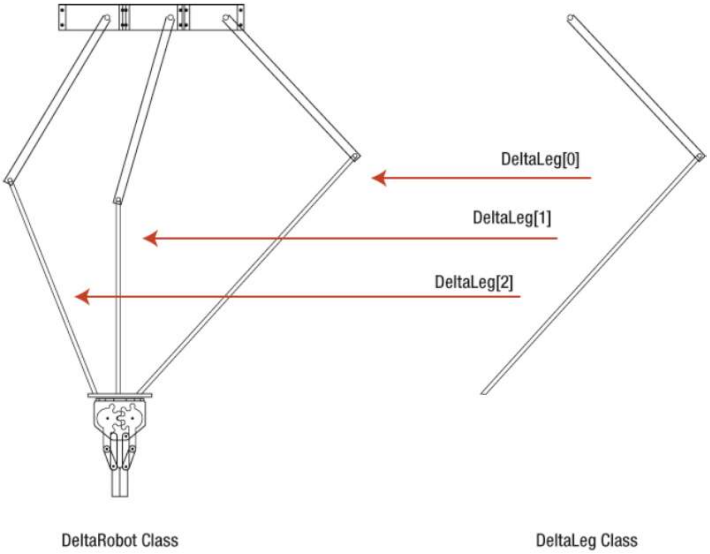


Fig. 47 Class definidos para el robot Delta

Se define una variable llamada ***numLegs*** que define el número de patas del robot. Sí, en realidad se puede aumentar el número de piernas para 4, 5, o cualquier número superior. El hecho de que no hay ningún robot paralelo de 23 patas es porque cualquier gran número de patas no sería más que una pérdida de material (tres es el número mínimo necesario de piernas para la estabilidad). Una vez dicho esto, uno de los robots delta comercial más rápido, el Adept Quattro, se basa en una arquitectura de cuatro patas. Los resultados son ciertamente curiosos y visibles en la Fig.47.

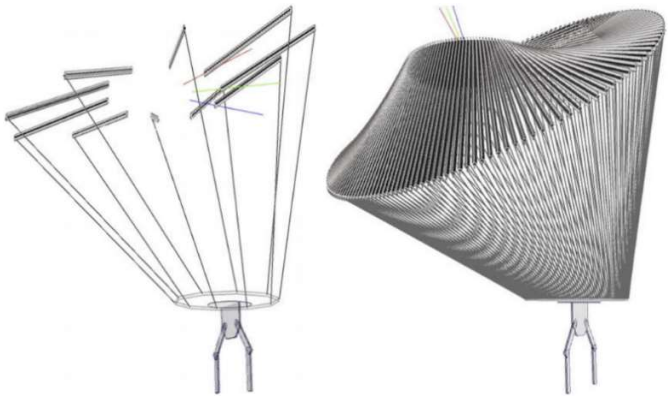


Fig. 48 Dos robots delta, con 5 y 150 patas.

Posteriormente, se estableció una matriz llamada ***DeltaLeg*** para contener las tres piernas del robot (o cualquier otro número), y un ***array*** de tipo ***float*** para los ángulos de las articulaciones accionadas por los servos (las articulaciones superiores). También declara las variables que establecen las dimensiones del robot delta específico, y tres variables que definen el lapso máximo del robot en los tres ejes: ***robotSpanX***, ***robotSpanZ*** y ***MaxH***). Es necesario tener en cuenta estas distancias para evitar enviar al robot a lugares inalcanzables para él, ya que esto llevaría a la desaparición de las piernas en el caso de la simulación, o peor, un robot dañado si usted está manejando el robot físico. Ahora bien, se toma como parámetros las longitudes de los elementos de la cadera y tobillo, y el tamaño de la base y el efector. Tenga en cuenta que cuando se hace referencia a los tamaños de las bases y efectoras, se refiere a la distancia entre el centro de la base / efector y el eje de la articulación unida al elemento.

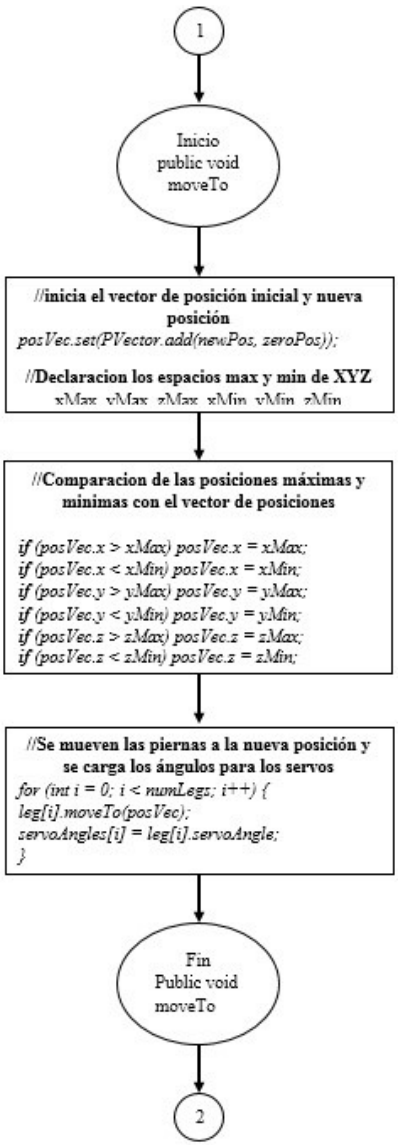


Fig. 49 Sección 2 Diagrama de flujo class DeltaRobot

Los ***DeltaLegs*** necesitan ser inicializados a las dimensiones correctas, las que se extrae a partir de los parámetros del robot. Cada pierna se inicializa a un cierto ángulo, el cual se encuentra dividiendo el círculo completo (2 * pi radianes) por el número de piernas. En el esquema, se ha añadido PI / 6 porque se quiere centrar las piernas desde el punto de vista de Kinect.

Una vez que se inicializan todas las variables, se agrega algunos métodos para el class del main del robot delta. Los métodos públicos que se llaman desde el boceto principal son ***moveTo ()*** y ***draw ()***. El método ***moveTo () (Fig.48 y Apéndice B)*** construye un PVector como parámetro. Este método se utiliza para decirle al robot que debe ir a una posición específica de su dibujo principal. Se tiene en cuenta que este PVector debe ser la posición relativa del efector del robot hasta el punto especificado como la posición cero.

La posición resultante del efector del robot es el resultado de sumar el PVector entrante a la posición cero. Dentro de este método, y antes de pasar a la nueva posición para cada pierna, se necesita asegurarse que la posición es una posición "legal", lo que significa una posición físicamente accesible por el robot. La posición es legal si está contenido dentro de un paralelepípedo de las dimensiones definidas por las variables ***robotSpanX***, ***robotSpanZ*** y ***MaxH***. Si una de sus coordenadas está fuera de este volumen virtual, se debe recortar el valor al valor máximo del movimiento en ese eje.

Por último, mover cada pierna a su nueva posición y preguntar qué ángulo hay que mover los servos a fin de lograrlo. El ángulo de servo se almacena en una matriz para ser utilizada más tarde. Mediante este paso, se tienen todos los datos que necesita para conducir el robot (los tres ángulos de servo), por lo que ahora se puede implementar un protocolo de comunicación en serie y empezar a conducir el robot de inmediato.

Pero se debe desarrollar la class un poco más allá para incluir una interfaz de visualización completamente nueva para el robot. En esta última instancia, se tiene un modelo virtual del robot delta que se comporta exactamente igual que el modelo físico y que se puede visualizar en la pantalla. Esto es importante para el proyecto porque es una buena idea para asegurarse de que todo funciona antes de enviar los valores a los servos, pero todo el proyecto puede funcionar perfectamente sin la parte visual. También se debe considerar que esta simulación es de gran ayuda para entender el comportamiento de la máquina, y más tarde va a resultar muy útil cuando se trae la distancia de los datos de entrada del Kinect.

El método **public void draw() (Fig.49 y Apendice C)** es llamado de su rutina principal cuando se quiere imprimir el robot delta en la pantalla. Este método cambia la matriz principal a una posición diferente por lo que el robot delta se muestra a una cierta altura en lugar del origen de coordenadas, ya que el final de este origen es el dispositivo Kinect. Luego se llama a la función propia de cada pierna **draw ()** y la función **drawEffector()** que describe a la pinza.

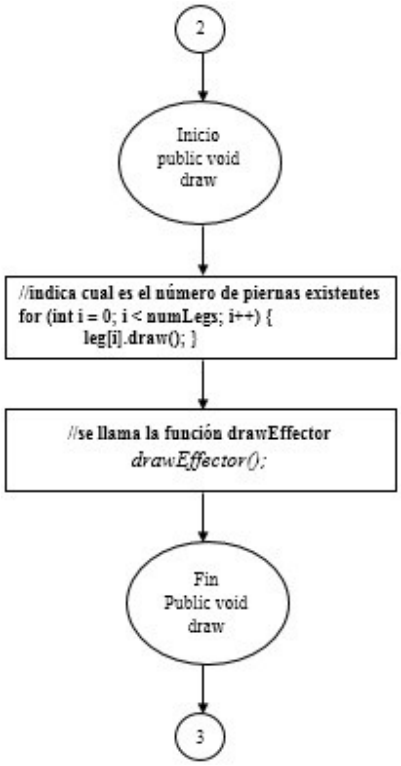


Fig. 50 Sección 3 Diagrama de flujo class DeltaRobot

La función **drawEffector() (Fig.50 y Apéndice D)** muestra el efector y la pinza que se ha unido a él. Principalmente es un ejercicio de cambio de matrices de transformación y elementos de dibujo por lo que terminan teniendo un modelo simplificado de la pinza que se puede girar, abrir y cerrar a voluntad, Fig.49.

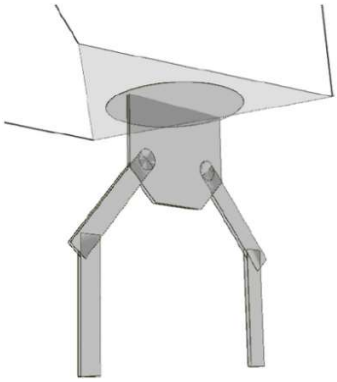


Fig. 51 Simulación del efector y del gripper

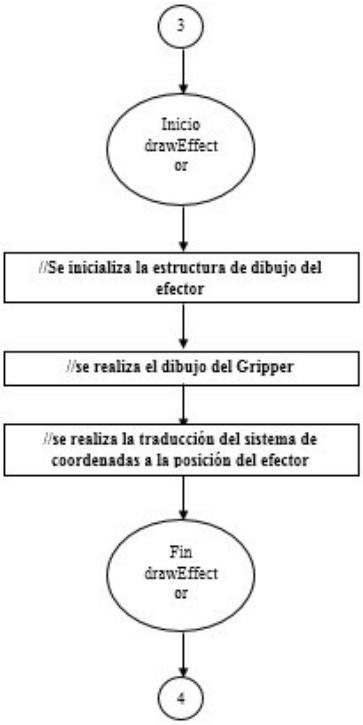


Fig. 52 Sección 4 Diagrama de flujo class DeltaRobot

Hay una función más que añadir, pero sólo se utiliza en el siguiente ejemplo. El método **updateGrip() (Fig.50 y Apendice E)** se puede llamar desde el boceto principal cada vez que desee cambiar la rotación y el ancho de la pinza.

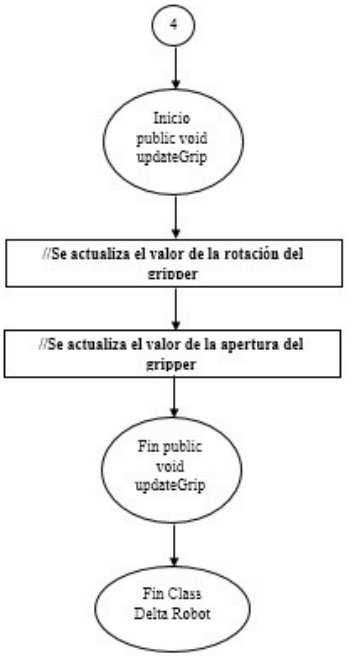


Fig. 53 Sección 5 Diagrama de flujo class DeltaRobot

5.7 CLASS DELTALEG

Fig. 54 Sección 1 Diagrama de flujo class DeltaLeg

El class principal **DeltaRobot** está completa ahora, y se ha construido las interfaces necesarias para conducir el robot de su programa principal, pero la parte más importante del código aún no se ha implementado: la elaboración de la cinemática inversa del robot y conseguir en el servo ángulos necesarios para lograr la postura deseada. La class **deltaRobot**, al mover el robot a una nueva posición, utiliza un "recuadro negro" que le da los ángulos de los servos. Este cuadro negro es la **class deltaLeg (Fig.53 y Apendice F)**. Esta clase tiene una serie de variables que definen el **ID** de la pierna, la posición del efector, el ángulo del servo, el ángulo de la pierna, y la coordenadas del mundo real de las articulaciones de la pierna, que se utiliza para la visualización.

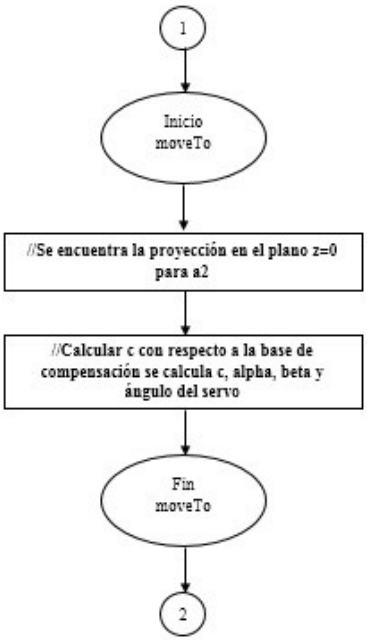


Fig. 55 Sección 2 Diagrama de flujo class DeltaLeg

La siguiente función, **moveTo (Fig.54y Apendice G)**, es en esta función donde se realiza el mayor trabajo. Esta función convierte la posición corta vector de entrada en una rotación de los servos. Este es el equivalente de decir que esta es la función a realizar la cinemática inversa del robot. (La cinemática inversa del robot delta fue previamente analizada, por lo que sólo se tiene que repetir el código en este punto.)

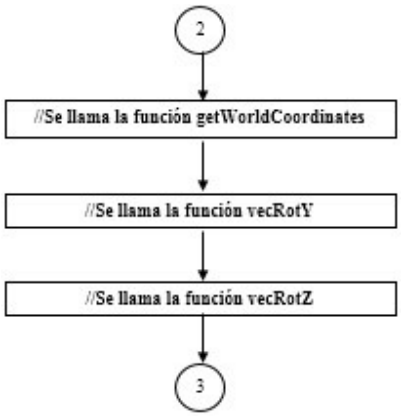


Fig. 56 Sección 3 Diagrama de flujo class DeltaLeg

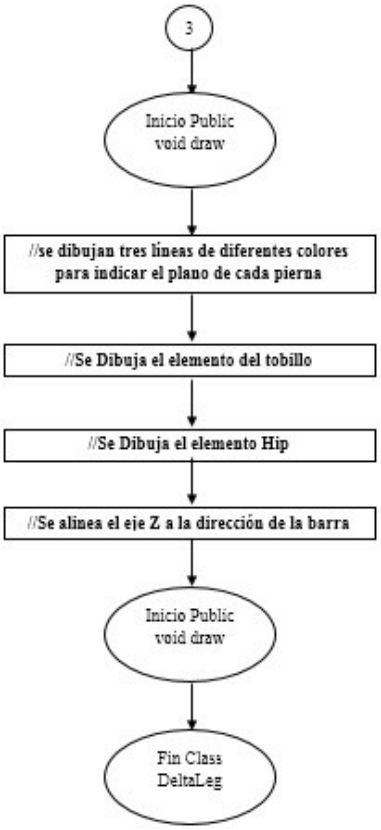


Fig. 57 Sección 4 Diagrama de flujo class DeltaLeg

La función **getWorldCoordinates() (Fig.55 y Apendice H)** actualiza los PVectors que definen las articulaciones de la pierna en coordenadas del mundo, por lo que puede dibujarlos en la pantalla. Se utiliza **vecRotY()** y **vecRotZ()** como funciones de ayuda para realizar las rotaciones de vector. La función **public draw() (Fig.56 y Apendice I)** es llamada desde la función **deltaRobot draw()** y se encarga de mostrar la geometría de la pierna en la pantalla. Una vez más, no se va a entrar en mucho detalle aquí, pero se puede seguir los comentarios encontrados en el código.

Se ha incluido todas las funciones necesarias para el manejo y la visualización de un robot delta en dos clases que se pueden añadir a cualquier sketch de procesamiento, así se puede ejecutar un primer algoritmo ***(Apendice J)***. Donde se pone en práctica la más simple de las aplicaciones e impulsar un modelo virtual de un robot delta con respecto a la posición del ratón. Se debe importar **OpenGL** y su biblioteca **Orbit Kinect**; posteriormente, se inicializa un objeto **deltaRobot** para iniciar. Luego se tiene que hacer es, dentro del bucle de la órbita (para que pueda girar alrededor), se crea un **PVector** de movimiento con las coordenadas del ratón, se busca mover el robot delta a ese punto en el espacio, y luego dibujar en la pantalla para ver el resultado. Cuando se ejecute el sketch y se deberá obtener una imagen de un robot delta en la pantalla Fig.57. Si mueve el puntero del ratón sobre el dibujo, el robot debe moverse en consecuencia.

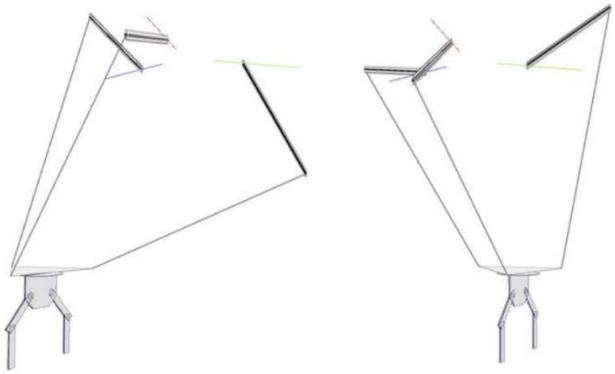


Fig. 58 Simulación del Robot Delta por medio del ratón

Cuando se ve por primera vez un robot delta en acción, es impresionante por la riqueza de los movimientos que surgieron a partir de una estructura tan simple.

5.8 ROBOT DELTA CONTROLADO POR KINECT

Se va a volver a utilizar las dos clases que ya se han creado para desarrollar una aplicación más compleja **(Apéndice K)**. El objetivo es dar a conocer las capacidades de seguimiento de la mano de Kinect NITE para mover la simulación del robot delta, y luego implementar su propia rutina para mover la pinza por la inclinación de la mano, para la apertura y el cierre de la pinza Fig.58. Luego se le agrega una rutina de comunicación serie para enviar el estado del servo al Arduino para que se pueda conectar así el software directamente en el robot físico y manipularlo con la mano.

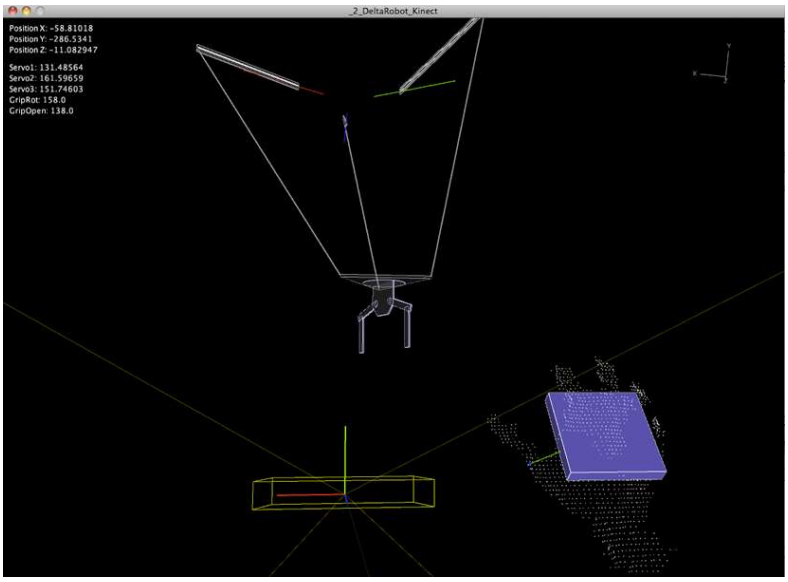


Fig. 59 El robot Delta es manipulado por gestos de las manos

Se Importa el conjunto completo de las bibliotecas que se han estado trabajando (esto significa simple-OpenNI, comunicación Serial, OpenGL, y KinectOrbit) y luego se declaran las variables correspondientes. Se está utilizando seguimiento de la mano, por lo que necesita el Gestor de sesiones del NITE y los Puntos de Control. La variable Boolean serial se puede establecer en falso si se desea ejecutar el sketch sin una tarjeta Arduino conectado al ordenador.

Fig. 60 Sección 1 Programa Principal Robot delta con Kinect

Como se quiere la capacidad de controlar la posición del robot y el estado de la pinza, se deberá declarar el **PVector motionVec** y los dos flotantes **gripRot** y **gripWidth**.

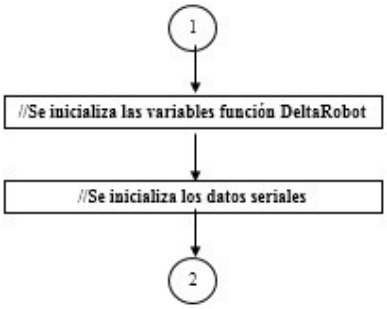


Fig. 61 Sección 2 Programa Principal Robot delta con Kinect

La función **setup()** inicializa todos los objetos que se han declarado con anterioridad y permite utilizar todas las capacidades del NITE que se van a utilizar. Se tiene en cuenta que sólo se está agregando el reconocimiento de gestos para el Administrador de la sesión, ya que quieren tener un mejor control de la creación de la mano en tiempo de ejecución.

Fig. 62 Sección 3 Programa Principal Robot delta con Kinect

Si se incluye **RaiseHand,** se observa que tan pronto como su mano entra en el campo de vista del Kinect dispara un evento de creación para la mano; añadiendo sólo agitando la mano, se

observa que se puede controlar cuándo y dónde se inicia el seguimiento de la mano, lo que es útil en una etapa posterior. Por supuesto, también se inicializa el objeto robot delta para las dimensiones (en mm) del robot que se construyó y se establece la comunicación serial por puerto en el ordenador al que está conectada la tarjeta Arduino.

Ahora se adiciona las funciones **XvN** de los puntos de control. Se establece una bandera de seguimiento de la mano o se desactivan y actualizan su **PVector** de la mano y el historial de la posición de la mano en un **ArrayList**. Se añade una línea más a la función **onPointCreate**. Debido a que se desea realizar un seguimiento de sus movimientos de la mano después de haber movido la mano al frente del Kinect, es necesario configurar el **handOrigin PVector** hasta el punto en el espacio, donde la agitación se llevó a cabo Fig.62. Esto permite establecer su vector de movimiento como el vector de desplazamiento relativo de la mano y del punto de creación de la mano.

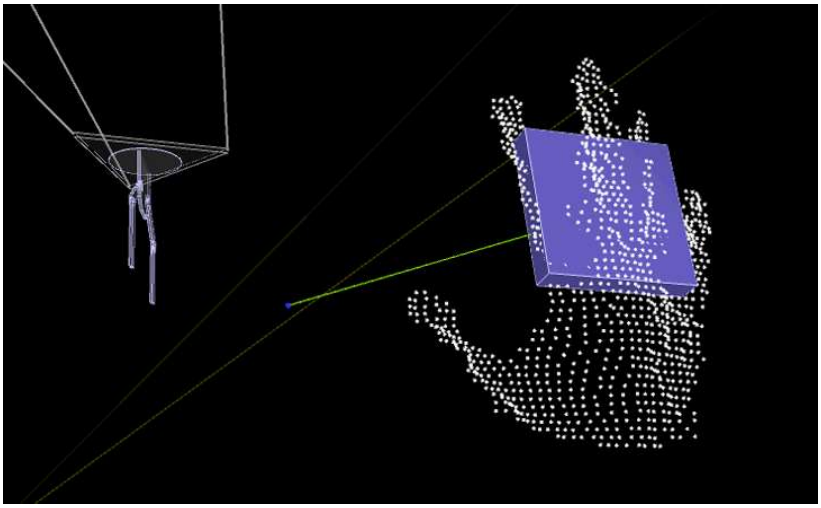


Fig. 63 Origen de la mano y la posición actual de la mano

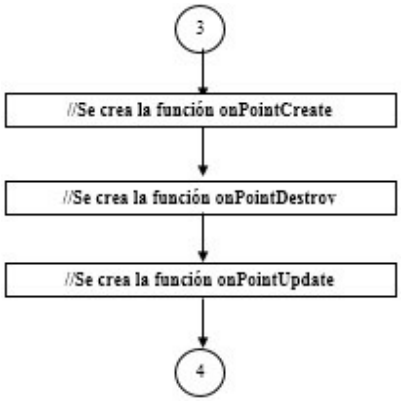


Fig. 64 Sección 4 Programa Principal Robot delta con Kinect

El main **draw()** actualiza el objeto del Kinect y establece un loop del **Orbit Kinect**. Dentro de este loop, se actualiza la posición de la mano y la dibuja, mediante dos funciones específicas que se implementan.

Fig. 65 Sección 5 Programa Principal Robot delta con Kinect

Sólo para tener una idea del movimiento relativo entre la posición actual de la mano y el punto de origen (donde se creó la mano), se dibuja una línea verde entre los dos.

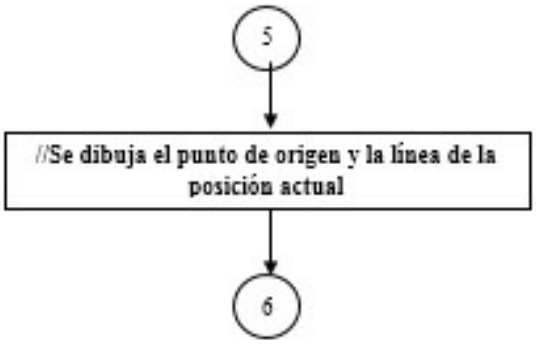


Fig. 66 Sección 6 Programa Principal Robot delta con Kinect

Y ahora se almacena ese vector de movimiento relativo en el **motionVec PVector** y se utiliza como un parámetro para mover el robot delta a su nueva posición. A continuación, se procede a elaborar el robot delta y enviar los datos en serie. Se añade una función **displayText** donde se imprimen en la pantalla los datos que se están enviando al puerto serie para los datos de control de los servo.

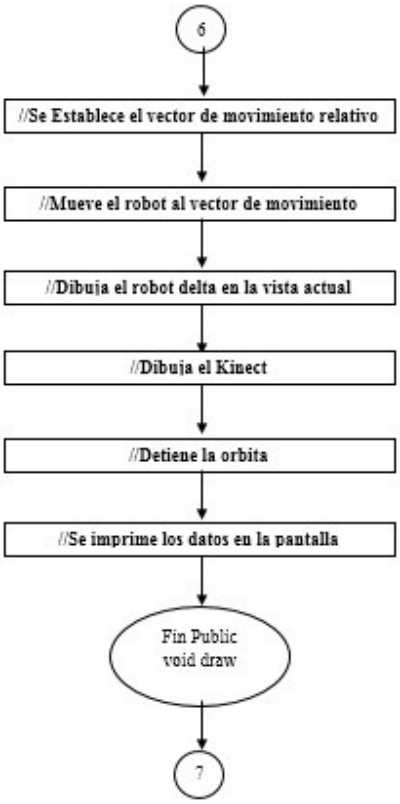


Fig. 67 Sección 7 Programa Principal Robot delta con Kinect

5.9 CONTROL DEL GRIPPER

Ahora bien se desea implementar una rutina para controlar la pinza del robot delta con movimientos de la mano. Se debe tener en cuenta usar NITE, ya que se ha elaborado la posición de la mano, que es el centro de la palma, y se está utilizando para fijar la posición del robot en el espacio. Ahora se busca abrir y cerrar la pinza al abrir y cerrar la mano.

Hay una manera sencilla de implementar esta característica. Se sabe que la mano se define por el conjunto de puntos alrededor de la posición **handVec**, para que se pueda analizar su nube de puntos y se seleccione todos los puntos dentro de una cierta distancia del vector de la mano. Después de algunas pruebas, se establece 100 mm a una distancia razonable. Si muestra los puntos que caen dentro de esta regla, se obtiene un "guante blanco" Fig.67, que se utiliza para extraer el estado de su mano.

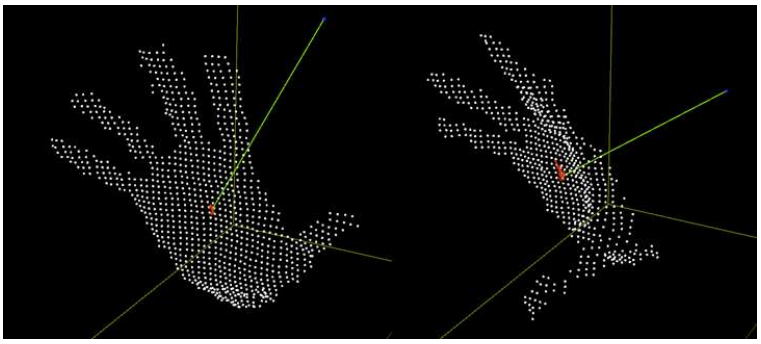


Fig. 68 Punto de inicio de la mano

Una vez que se identifiquen los puntos, se puede pensar en lo que cambia al abrir y cerrar la mano. Hay dos formas de hacerlo: cuando se abre la mano, el ancho total de las nubes de puntos se incrementa, lo que quiere decir que la distancia horizontal entre los puntos aumenta más a la derecha y aumenta más a la izquierda (la punta de los dedos pulgar y meñique) Fig.68.

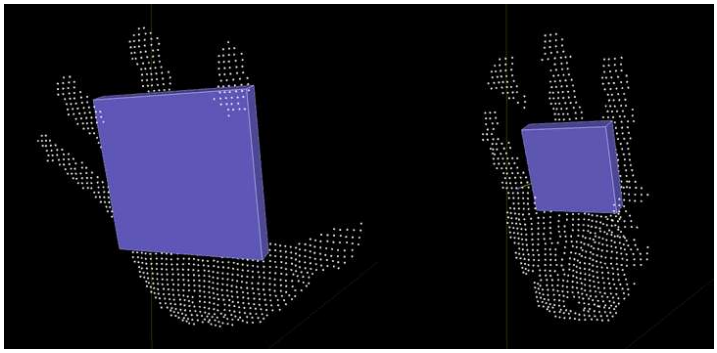


Fig. 69 Seguimiento ancho de la Mano

De la misma manera, se puede observar que al inclinar la mano se contrae y al revés, la altura total de la nube de puntos, que se pueden utilizar posteriormente para definir la rotación de la pinza Fig.69.

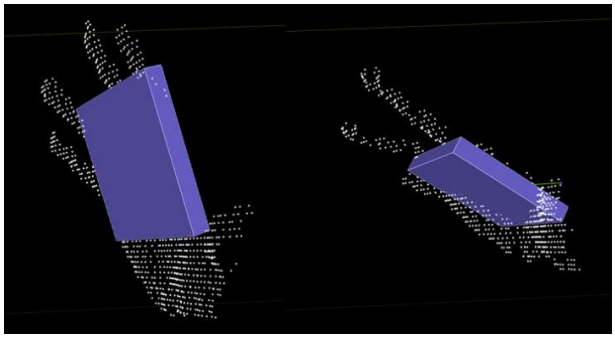


Fig. 70 Seguimiento de la inclinación de la mano

Se necesita encontrar estos puntos y calcular sus distancias. Para lo cual se implementó el presente código **(Apéndice K)**:

Fig. 71 Sección 8 Programa Principal Robot delta con Kinect

Después de ejecutar este loop, tiene cuatro **PVectors** para almacenar los cuatro puntos que se necesita para el propósito que se necesitan. Usted va a dibujar un aparatito de control utilizando estos puntos. Este aparatito es un cubo, cuyo tamaño va a cambiar de acuerdo al ancho de su mano. Del mismo modo, la inclinación a tratar de coincidir con la inclinación de la mano, en función de su altura.

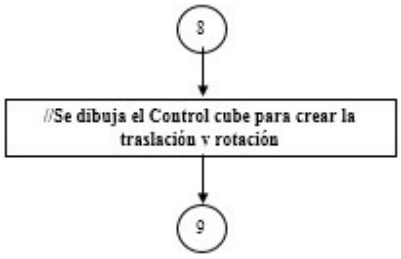


Fig. 72 Sección 9 Programa Principal Robot delta con Kinect

Después de ejecutar el código, se probaron algunos rangos para los valores de anchura y altura de la mano. Un rango de 65-200 funciona bastante bien para ambos parámetros. Si se asigna este rango de 0 a 255, tendrá un valor muy leve para ser transmitido a la tarjeta Arduino. Se puede almacenar los valores asignados como **gripWidth** y **gripRot**.

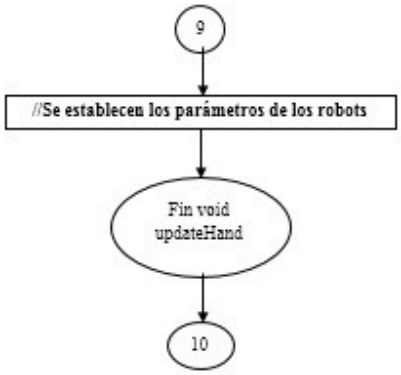


Fig. 73 Sección 10 Programa Principal Robot delta con Kinect

Se incluye la función **drawHand()** a partir de los sketch previos para llevar un registro de posiciones de las manos anteriores.

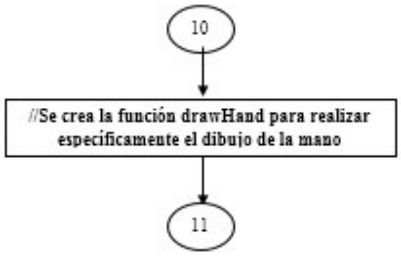


Fig. 74 Sección 11 Programa Principal Robot delta con Kinect

5.10 CONCLUSION

El presente capítulo se da a conocer el algoritmo implementado en Processing para la determinación de la posición y la pose de la mano con el fin de controlar las características de los sistemas. Cabe resaltar que la detección de la mano y giro de la pinza, dependerán del área e inclinación de la mano del operario. Dentro del Apéndice de la presente tesis se encuentra el código completo utilizado, es importante recalcar que la cinemática inversa del sistema se encuentra inmersa en el algoritmo.

CAPITULO VI:
DISEÑO SISTEMA ELECTRONICO

6. CAPITULO V: DISEÑO SISTEMA ELECTRONICO

6.1 VISION GENERAL

En este capítulo se da a conocer la electrónica implementada para la activación de los servomotores, el sistema microcontrolado propuesto se basa en Arduino UNO, para lo cual fue necesario diseñar una pequeña baquela para la conexión de los servomotores. Posteriormente, se realizó la programación para la recepción de los datos enviados por Processing.

6.2 ARDUINO

Fig. 75 Arduino Open – Source Community

Arduino[6] es una plataforma de Hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares. El Hardware consiste en una placa con un microcontrolador Atmel AVR y puertos de entrada/salida. Los microcontroladores más usados son el ATmega168, ATmega328, ATmega1208, ATmega8 por su sencillez y bajo coste que permiten el desarrollo de múltiples diseños. Por otro lado el software consiste en un entorno de desarrollo que implementa el lenguaje de programación Processing/Wiring y el cargador de arranque (boot loader) que corre en la placa. Al ser open-hardware, tanto su diseño como su distribución son libre. Es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia. Arduino puede tomar información del entorno a través de sus pines de entrada de toda una gama de sensores y puede afectar aquello que le rodea controlando leds, motores y otros actuadores[7].

6.3 LENGUAJE DE PROGRAMACION

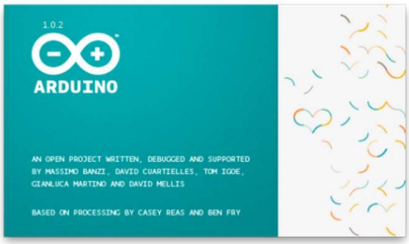


Fig. 76 Ventana de Bienvenida Software Arduino

[6] http://arduino.cc/en/
[7] http://burutek.org/es/arduino/

El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino (basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador, si bien tienen la posibilidad de hacerlo y comunicar con diferentes tipos de software (p.ej. Flash, Processing, MaxMSP)[8]. La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el popular lenguaje de programación de alto nivel Processing. Sin embargo, es posible utilizar otros lenguajes de programación y aplicaciones populares en Arduino. Algunos ejemplos son:

- Java, Flash (mediante ActionScript), Processing, Pure Data

Esto es posible debido a que Arduino se comunica mediante la transmisión de datos en formato serie que es algo que la mayoría de los lenguajes anteriormente citados soportan. Para los que no soportan el formato serie de forma nativa, es posible utilizar software intermediario que traduzca los mensajes enviados por ambas partes para permitir una comunicación fluida. Es bastante interesante tener la posibilidad de interactuar Arduino mediante esta gran variedad de sistemas y lenguajes puesto que dependiendo de cuales sean las necesidades del problema que vamos a resolver podremos aprovecharnos de la gran compatibilidad de comunicación que ofrece[9].

6.3.1 EQUIPO ARDUINO UNO VERSION R3

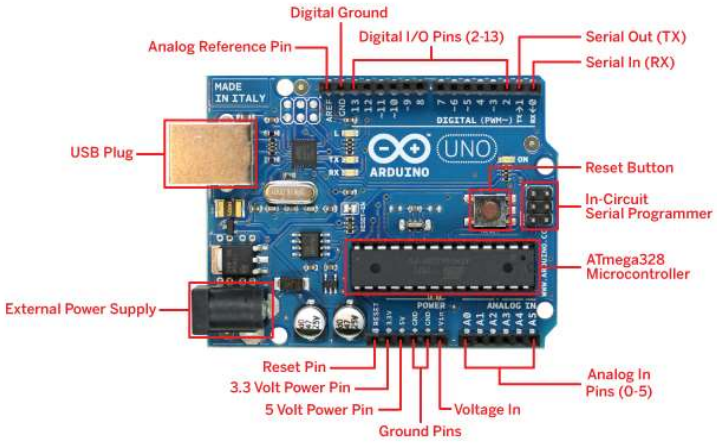


Fig. 77 Secciones de la placa Arduino Uno R3

El Arduino Uno R3. Además de todas las características de las anteriores versiones, el Uno R3 ahora utiliza un ATmega16U2 lugar del ATmega8U2 encontrado en el Uno R1 (o el FTDI encontrado en las generaciones anteriores). Esto permite una mayor velocidad de transferencia y más memoria. No necesita drivers para Linux o Mac (se necesita drivers para Windows y se incluyen en el Arduino IDE), y se tienen mejoras como comunicación con

[8] http://burutek.org/es/arduino/
[9] http://burutek.org/es/arduino/

teclado, ratón, joystick, etc[10]. El Arduino Uno R3 también añade pines SDA y SCL al lado de la AREF. Además, hay dos nuevos pines colocados cerca del pin RESET. Una de ellas es la instrucción IOREF que permiten a los escudos adaptarse a la tensión suministrada desde la placa. El otro es un no conectado y se reserva para usos futuros. El Arduino Uno R3 trabaja con todos los escudos existentes, pero puede adaptarse a los nuevos escudos que utilizan estos pines adicionales[11].

6.4 DISEÑO Y CONSTRUCCION CIRCUITO ELECTRONICO ROBOT DELTA

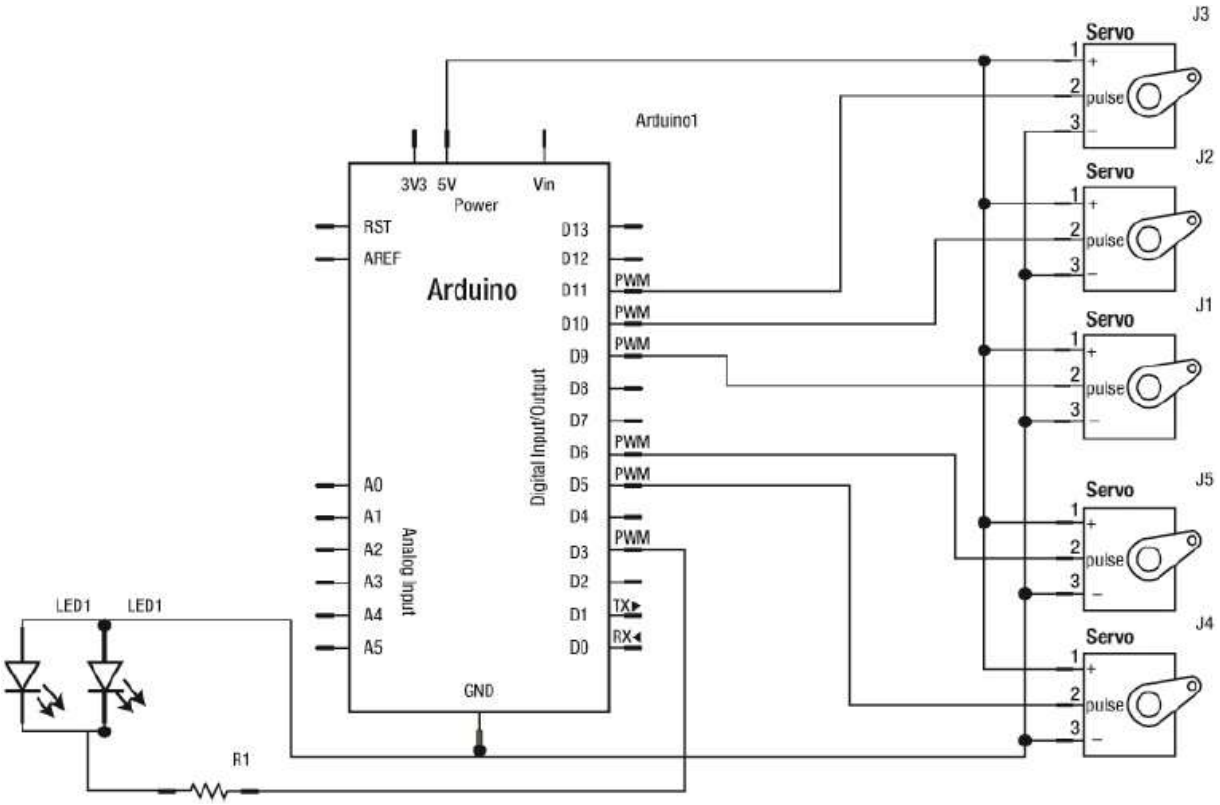


Fig. 78 Circuito implementado

Ahora en el cableado del sistema eléctrico. Se tienen dos LED y dos servos en la pinza más tres servos grandes para las piernas, y hay que conectarlos al circuito principal. Se conectaron algunos cables de extensión de la pinza para conectar los LED y los servos, desde una de las piernas con amarres.

Luego de conectar los cables de extensión para los tres servos en las piernas. Se tuvo en cuenta que se necesita una fuente de alimentación externa de 5V para este proyecto; la fuente de alimentación de 5V que proviene de la tarjeta Arduino no es lo suficientemente estable para los requisitos de potencia de trabajo de todos los servos al mismo tiempo. El circuito consta de cinco servos (tres para las piernas y dos de la pinza), dos LEDs con sus resistencias,

[10] https://www.sparkfun.com/products/11021
[11] https://www.sparkfun.com/products/11021

el Arduino, y una fuente de alimentación externa. Se propone usar las salidas PWM del Arduino y distribuirlos de la siguiente forma: 9, 10 y 11 para el control de los tres servos utilizados para las piernas. Los Pines 3, 5 y 6 de control de servos y el control de la pinza y los LEDs, como en la Fig.77.

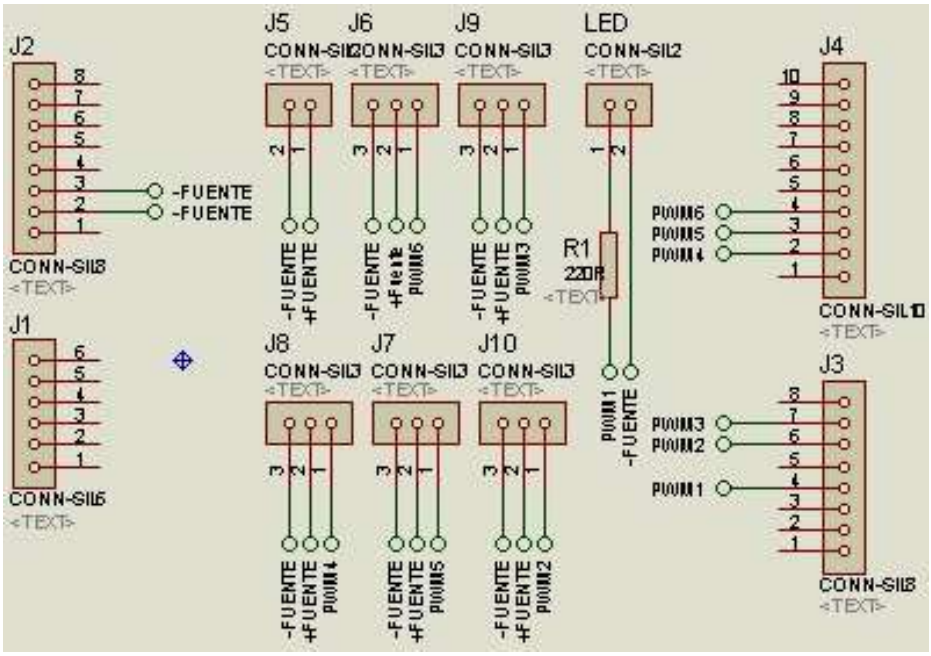


Fig. 79 Circuito Base Proteus ISIS

En la Fig.78, se muestra el circuito equivalente de la Fig.77 en el software proteus ISIS para construir la baquelita a usar en la plataforma delta.

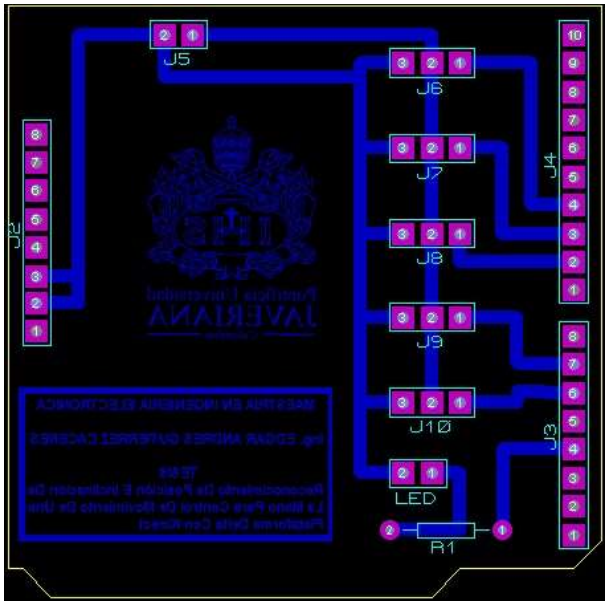


Fig. 80 Circuito Base Proteus ARES

En la Fig.79 se muestra el diseño de la tarjeta en ARES para su posterior construcción, De igual forma la Fig.80 muestra una opción de vista 3D de la baquela.

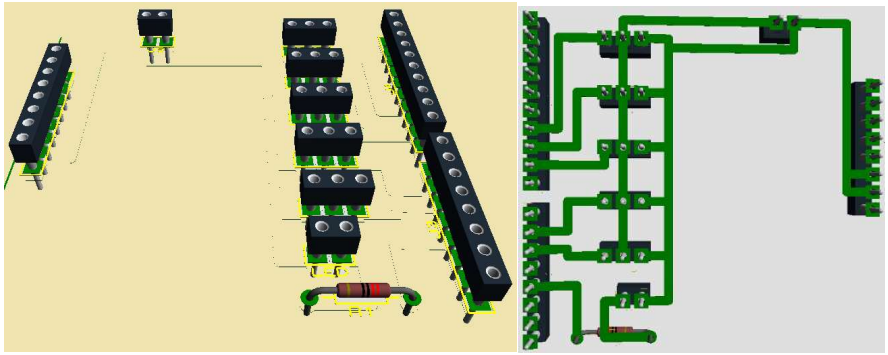


Fig. 81 Vista 3D superior e inferior de la Baquelita.

En la Tabla.3. Se encuentra una lista de los materiales necesarios para construir la baquela.

Tabla 4 Lista de materiales para la construcción de la baquela

CANTIDAD	DESCRIPCIÓN	VALOR* UNITARIO	VALOR* TOTAL
1	Resistencia de 220 Ohm	100	100
1	Diodo LED	100	100
1	Conector macho de 10 pines	100	100
2	Conector macho de 8 pines	100	200
2	Conector macho de 2 pines	100	200
5	Conector macho de 3 pines	100	500
1	Baquela de 10x10 cm	3200	3200
1	Acido para baquela	500	500
1	50 cm de Estaño	200	200
	TOTAL	4500	5100
	**Precio en pesos colombianos moneda legal y vigente*		

Se observa en la Fig.81 se puede ver la tarjeta finalmente construida.

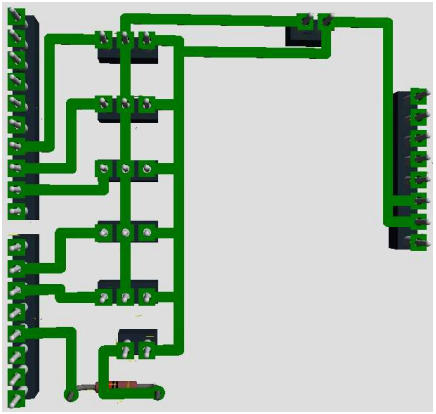


Fig. 82 Baquelita finalmente construida.

6.5 ALGORITMO IMPLEMENTADO ARDUINO

El código que se ejecuta dentro de la tarjeta Arduino tiene una función simple **(Apéndice M):** recibe datos en serie (9600 baudios) que asigna el rango de los servos y envía el mensaje apropiado a cada uno de ellos. Se necesita variables para almacenar los valores temporales de los datos entrantes y luego los enteros para el números del pin y los diferentes impulsos para los cinco servos.

Los datos tipo **long previousMillis** y los intervalos con el espacio entre los mensajes son enviados a los servos.

Fig. 83 Sección 1 Programa Arduino

Se establece todos los pines como salidas e inicializa el puerto serie.

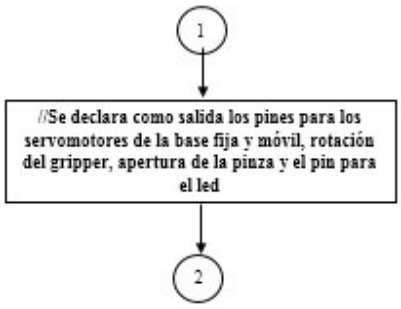


Fig. 84 Sección 2 Programa Arduino

En el **loop** principal, se escribe un pulso alto a los LED, ya que está indicando en todo momento que está sirviendo, y luego se comprueba el estado de la memoria intermedia serial. Si se ha recibido más de ocho valores, que es lo que se está esperando, y si el primero de ellos es el carácter 'X', se debe empezar a leer los valores como bytes.

Después de cada dos valores, se recompone el entero usando del lenguaje C la función **word()**, que devuelve datos de tipo de word (número sin signo de 16 bits, es lo mismo que un entero sin signo "*unsigned int"*) de dos bytes.

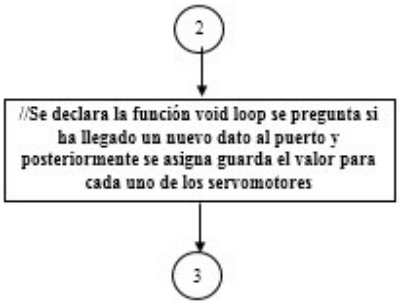


Fig. 85 Sección 3 Programa Arduino

Y ahora se reasignan los impulsos de sus rangos esperados para el rango del servo de 500 a 2500.

Fig. 86 Sección 4 Programa Arduino

Y, por último, si han transcurrido 20 milisegundos, se envía los impulsos de los servo para actualizar sus ángulos. Se debe recordar que los diferentes servos tienen ligeramente diferentes rangos, por lo que debe probar los servos y encontrar el rango correcto que los lleva de 0 a 180 grados antes de la reasignación de los valores.

Fig. 87 Sección 5 Programa Arduino

6.6 CONCLUSION

El sistema microcontrolado es de fácil manipulación ya que tan solo requiere recibir los datos enviados por puerto serial y analizados por el algoritmo hecho en Processing, realmente podríamos utilizar cualquier plataforma para el control de los motores. Sin embargo, se dio a conocer una propuesta de la implementación.

CAPITULO VII:
RESULTADOS

7. CAPITULO VII: RESULTADOS

7.1 VISION GENERAL

En esta sección se presenta los resultados experimentales y simulaciones de presentadas, se indica los sensores básicos para la medición y análisis de los datos entregados. Cabe resaltar que el presente trabajo de tesis se enfoca en el desarrollo del entorno de procesamiento de datos del Kinect y el diseño de la plataforma delta tal como se muestra en los resultados de las secciones anteriores, más no la asignación de alguna trayectoria definida. Sin embargo se entrega datos de análisis para posteriores aportes en esta sección como resultados de aporte de las variables básicas de la plataforma delta y respuesta a una planificación de trayectoria básica.

7.2 MEDICION DE DESPLAZAMIENTO, VELOCIDAD Y ACELERACION DEL SISTEMA PRUEBAS EN SOFTWARE

Para determinar el desplazamiento de la plataforma se realizó un algoritmo en simmechanics de Matlab. Fig.87.

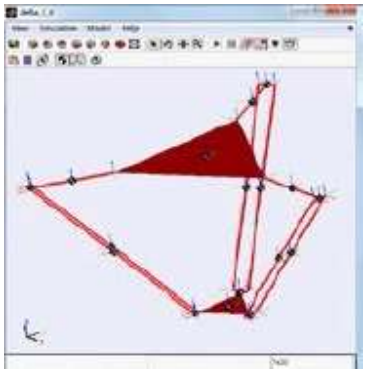


Fig. 88 Delta en Simmechanics

En Fig.88. Se puede observar que el sistema propuesto requiere de tres bloques principales, el primero enfocado a la planificación de la trayectoria, unida a la cinemática inversa. El segundo bloque dedicado al controlador PID y el tercero se enfoca al diseño del sistema e Simmechanics todo esto realizado en Matlab.

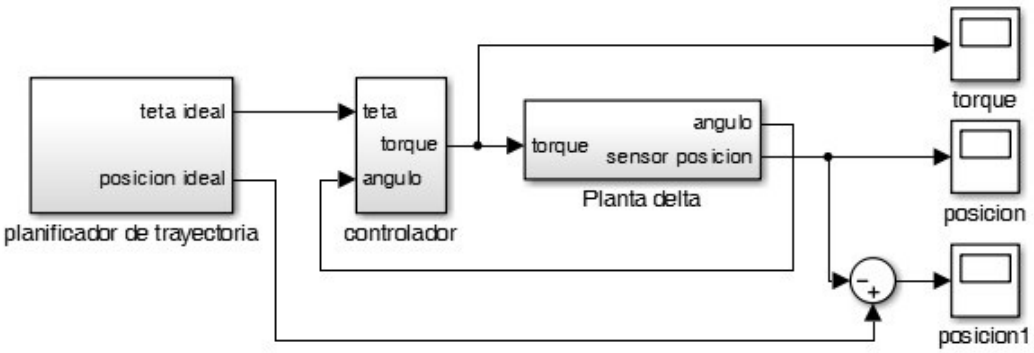


Fig. 89Diseño simulink

Dentro del sistema planta utilizado en simmechanics se planteó la estructura de la Fig.89 donde se encuentra el diseño de la Base, el muslo, la pierna y el efector del robot delta se creó un subsistema de cada una de las piernas, lo anterior debido a que se asignaron las coordenadas del vector de trabajo.

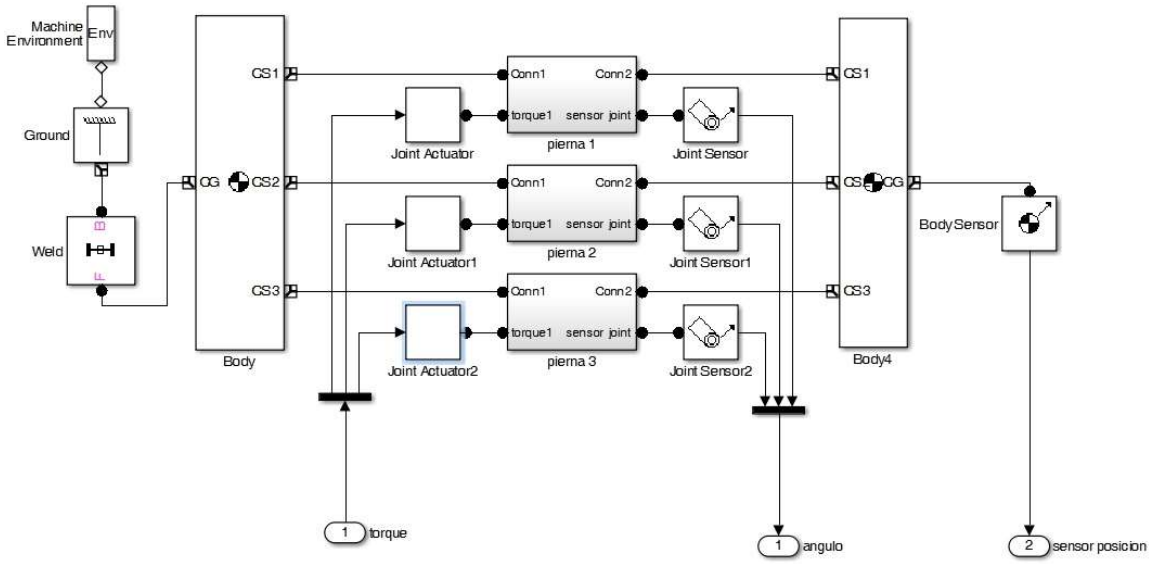


Fig. 90Planta simulink Delta Simmechanics

En la Fig.90 se aprecia un mejor detalle del sistema de cada una de las piernas del robot delta.

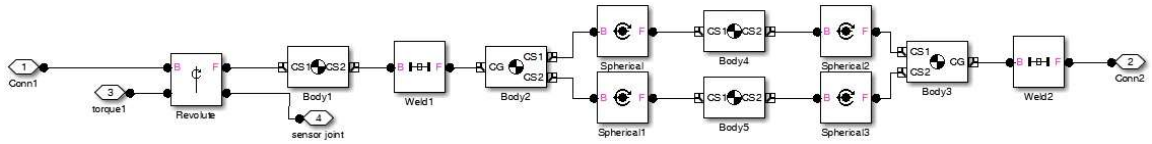


Fig. 91Diseño Pierna robot delta Simmechanics

El sistema de control es fácil de determinar ya que se requiere de un sistema PID tradicional

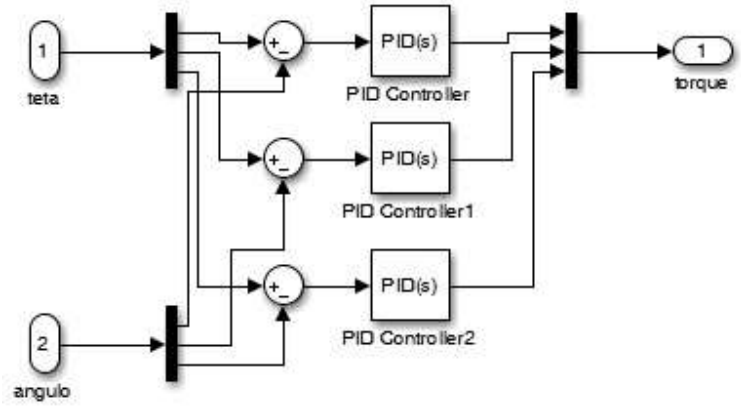


Fig. 92 Control PID Simulink

Teniendo en cuenta esto se crea finalmente el sistema en Matlab Fig. 92.

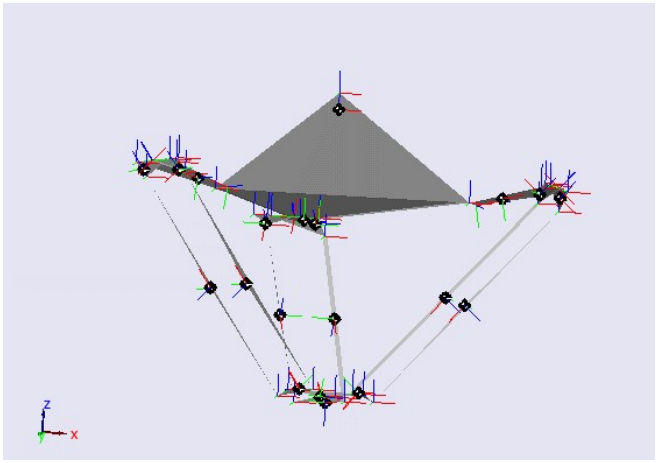


Fig. 93 Modelo delta creado en Simulink

Teniendo la estructura importada en Matlab se realizaron dos pruebas la primera en función de subida y bajada del efector en relación al eje z. Obteniéndose los siguientes resultados por ejemplo en cuento al torque provocado en los servomotores de la base del robot delta como se ve en al Fig.94:

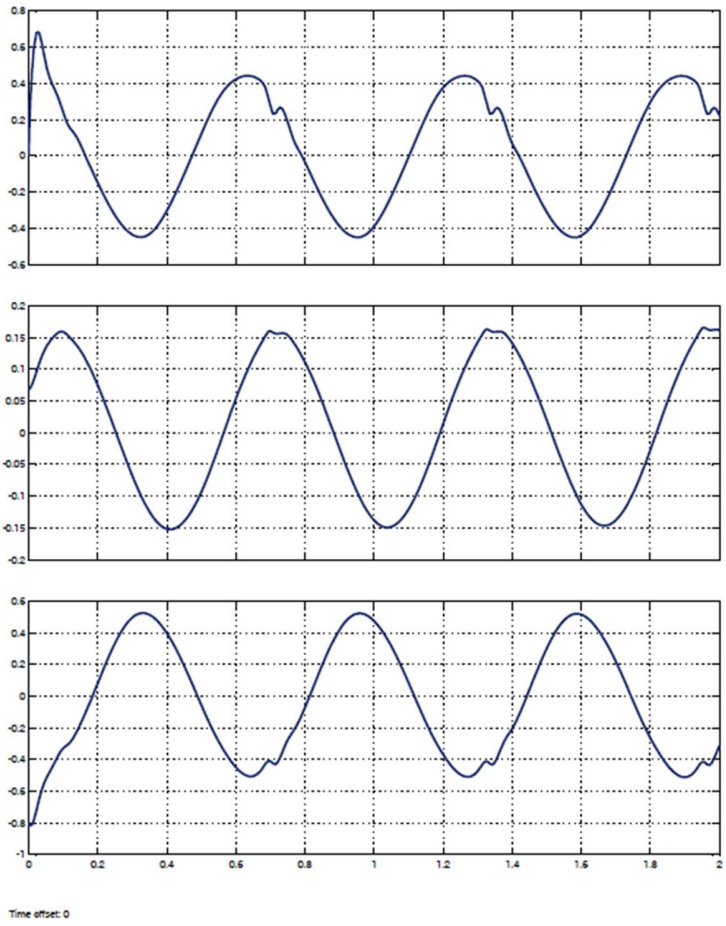


Fig. 94 Análisis torque de los servomotores de la base

Ahora bien, se estudió el desplazamiento en cuanto el eje XYZ, velocidad y aceleración en los mismos respectivos ejes.

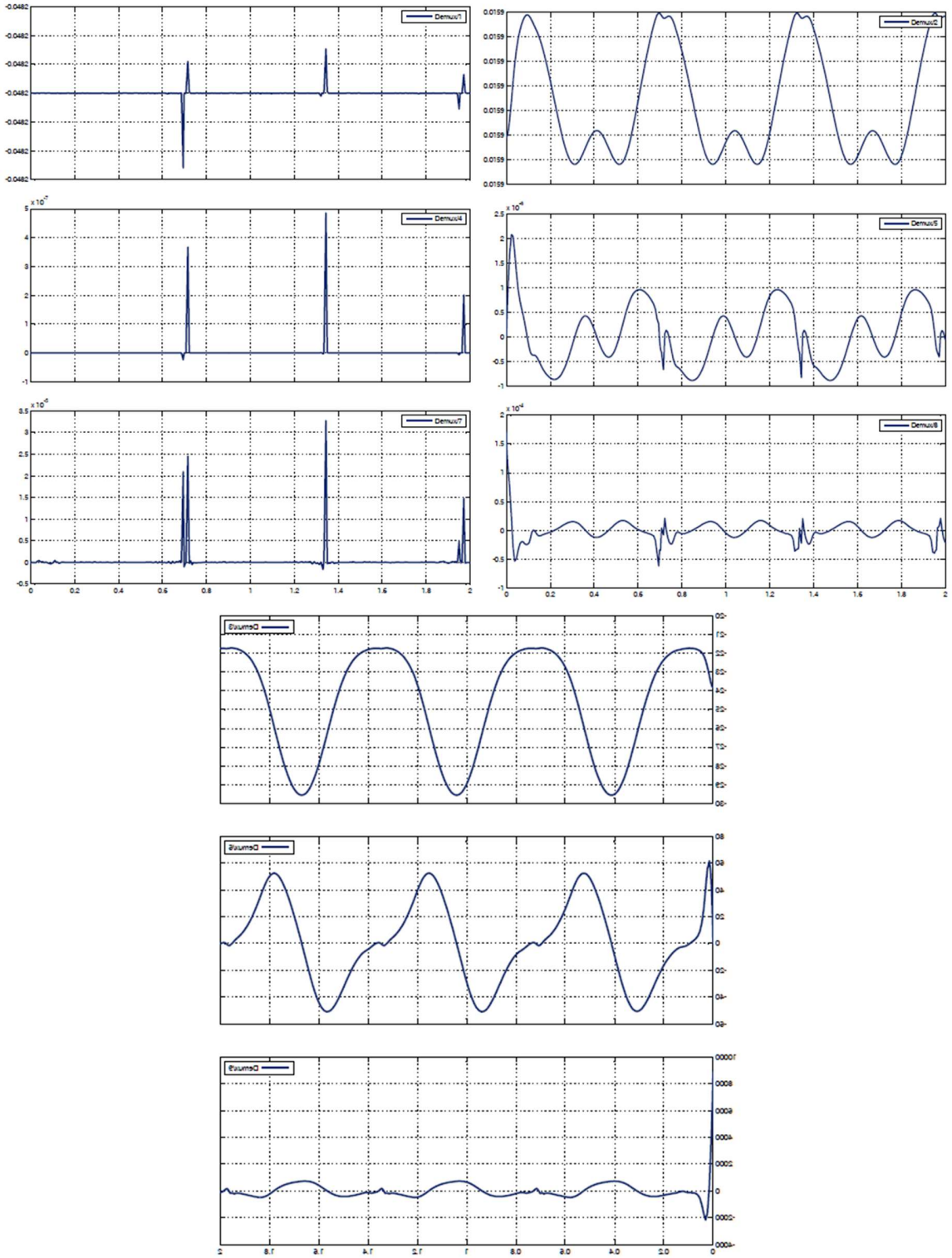


Fig. 95 posición, velocidad y aceleración del efector

En la figura anterior se observa que el error en el eje X con referencia en movimiento de la plataforma es de -0.0482 mm, y el error en el eje Y es igual a 0.0189 mm. Recordando que el movimiento es provocado en referencia al eje Z. La trayectoria propuesta para la toma de datos es la mostrada en la Fig.92

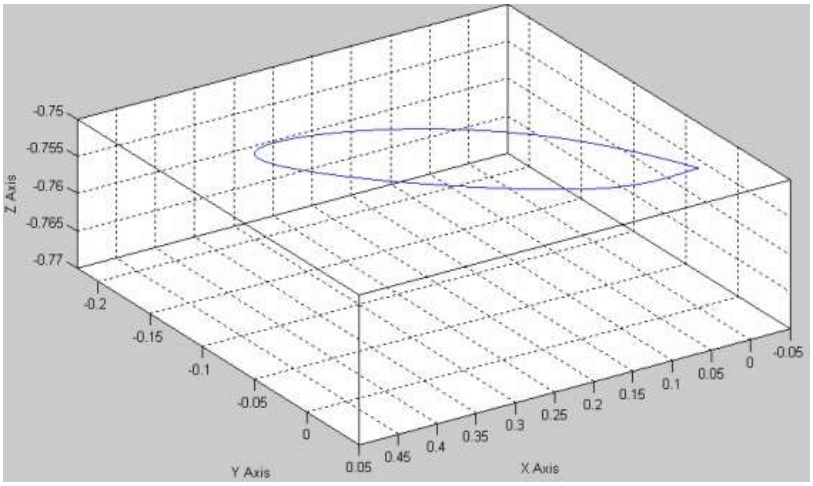


Fig. 96Trayectoria Generada

Teniendo en cuenta la trayectoria anterior, se obtuvieron datos del desplazamiento en grados, la velocidad y aceleración en cada uno de los servomotores, tal como se muestra en Fig.93. Fig.94 y Fig.95.

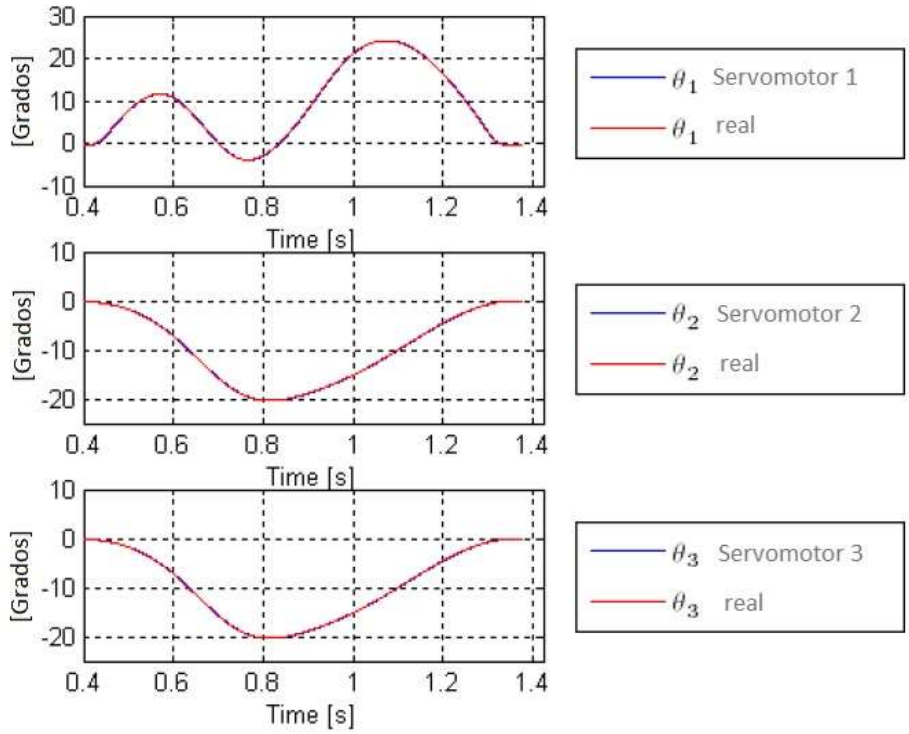


Fig. 97 Ángulos Servomotores

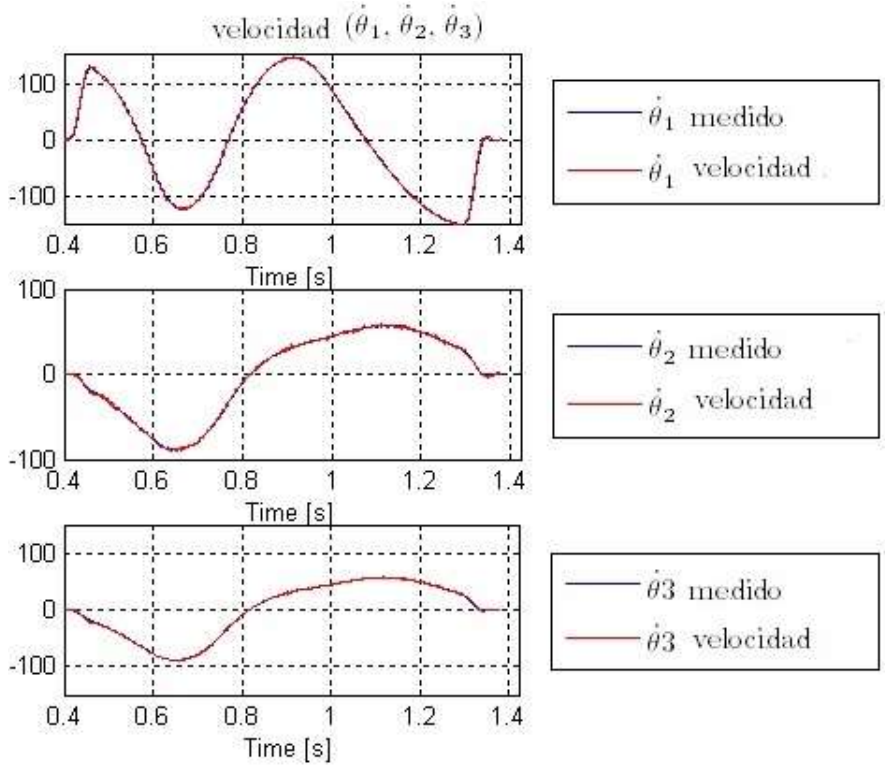


Fig. 98Velocidad de los motores

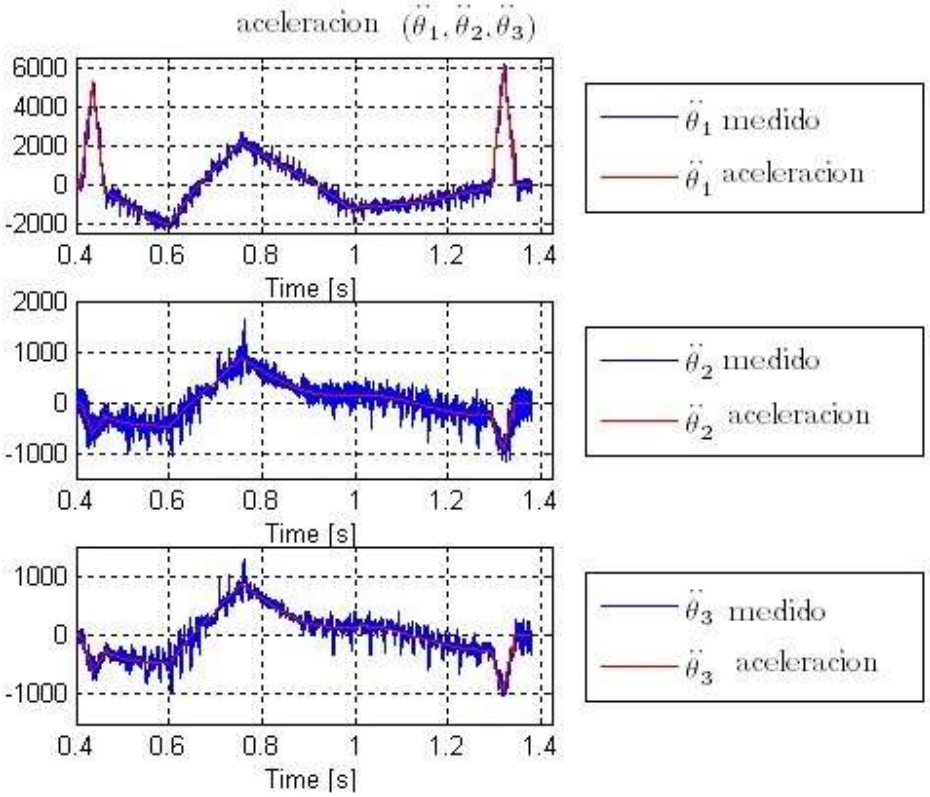


Fig. 99 Aceleración de los Servomotores

7.3 PROTOCOLO DE PRUEBAS

Se realizaron tres pruebas básicas; el primero correspondiente a subir y bajar la mano (movimiento vertical). El segundo consiste en mover la mano de izquierda a derecha (movimiento horizontal). Y por último, un movimiento circular sobre el eje horizontal.

7.3.1 MOVIMIENTO VERTICAL

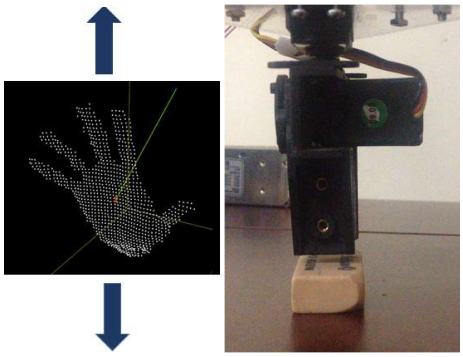


Fig. 100 ACS712

Una vez identificada la mano se realizaron pruebas de aciertos sobre un solo punto encontrándose la siguiente información desde una trayectoria generada y el operario.

Tabla 5 Consumos y %Error de la Prueba

Parámetros	**Datos**	
Frecuencia op.	100Hz	60Hz
%Error	80/100	65/100
Máximo I1	1,57 A	1,71A
Máximo I2	1,34 A	1,67A
Máximo I3	1,52 A	1,69A
Mínimo I1	0,57 A	0,63A
Mínimo I2	0,46 A	0,52A
Mínimo I3	0,49 A	0,61A

7.3.2 MOVIMIENTO HORIZONTAL

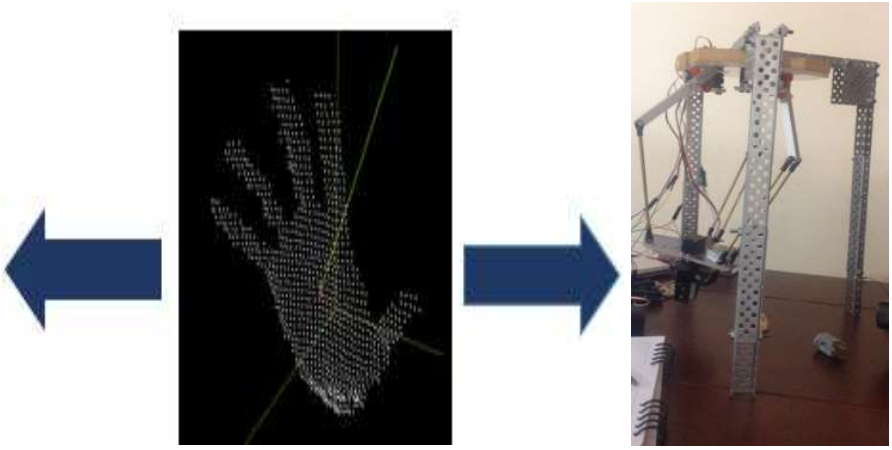


Fig. 101 ACS712

De igual forma se realizaron pruebas de aciertos sobre un solo punto encontrándose la siguiente información desde una trayectoria generada y el operario.

Tabla 6 Consumos y %Error de la Prueba

Parámetros	Datos	
Frecuencia op.	100Hz	60Hz
%Error	80/100	65/100
Máximo I1	1,42A	1,51A
Máximo I2	1,58 A	1,59A
Máximo I3	1,46 A	1,49A
Mínimo I1	0,63 A	0,73A
Mínimo I2	0,58 A	0,61A
Mínimo I3	0,60 A	0,81A

7.3.3 MOVIMIENTO CIRCULAR

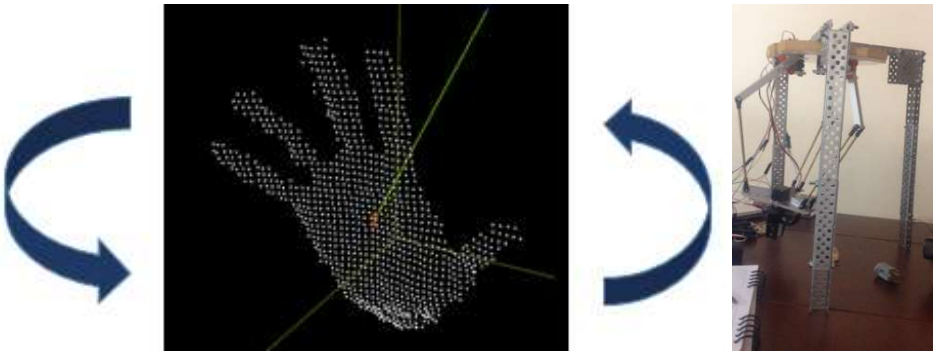


Fig. 102 ACS712

Se recolectaron datos de la trayectoria circular en cuanto a consumo y cantidad de veces que pudo repetir la acción circular.

Tabla 7 Consumos y %Error de la Prueba

Parámetros	Datos	
Frecuencia op.	100Hz	60Hz
%Error	75/100	60/100
Máximo I1	1,12A	1,15A
Máximo I2	1,18 A	1,29A
Máximo I3	1,61 A	1,09A
Mínimo I1	0,63 A	0,73A
Mínimo I2	0,58 A	0,61A
Mínimo I3	0,60 A	0,81A

7.4 MEDICION DE CORRIENTE Y POTENCIA

Para determinar el consumo de energía se utilizó el sensor de corriente de referencia ACS712, el cual es un sensor de efecto Hall de un máximo de 5A posee una etapa de amplificación de medidas pequeñas. El ACS712 se encuentra presente en cada uno de los servomotores de la plataforma (3 de la base, 1 de la muñeca y 1 de apertura y cierre de la pinza)

Fig. 103 ACS712

Para el análisis de corriente de consumo de cada servomotor y teniendo encuentra la medición se realiza el análisis de consumo de potencia del sistema. En la Fig.88. Se observa el modelo en matlab propuesto para esta medición teniendo en cuenta que el dato entregado por el sensor es análogo se necesita de una conversión de los datos digitales a variables de Corriente y potencia del sistema.

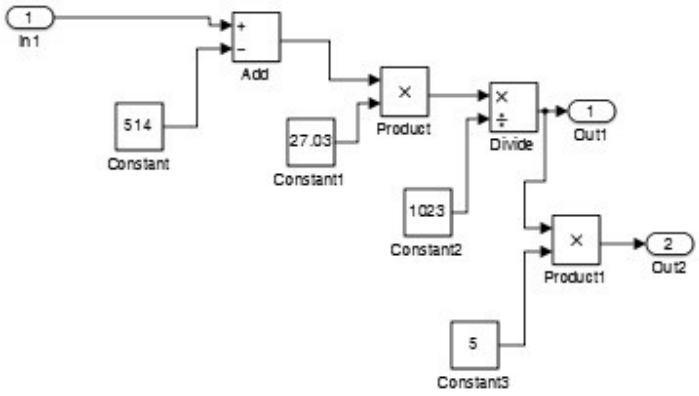


Fig. 104 Conversión dato ADC a parámetros de I y P

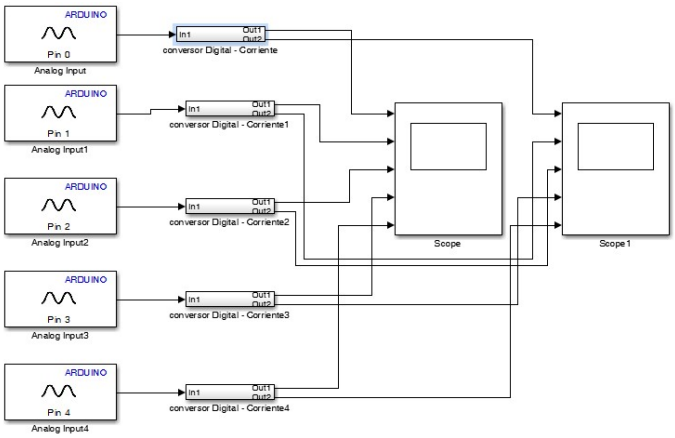


Fig. 105 Bloque creado para la conversión del ADC a I y P

Teniendo en cuenta esto se analizó el consumo de corriente de la plataforma obteniéndose la siguiente señal de corriente, iniciando por muñeca, pinza y los tres servomotores de la base fija.

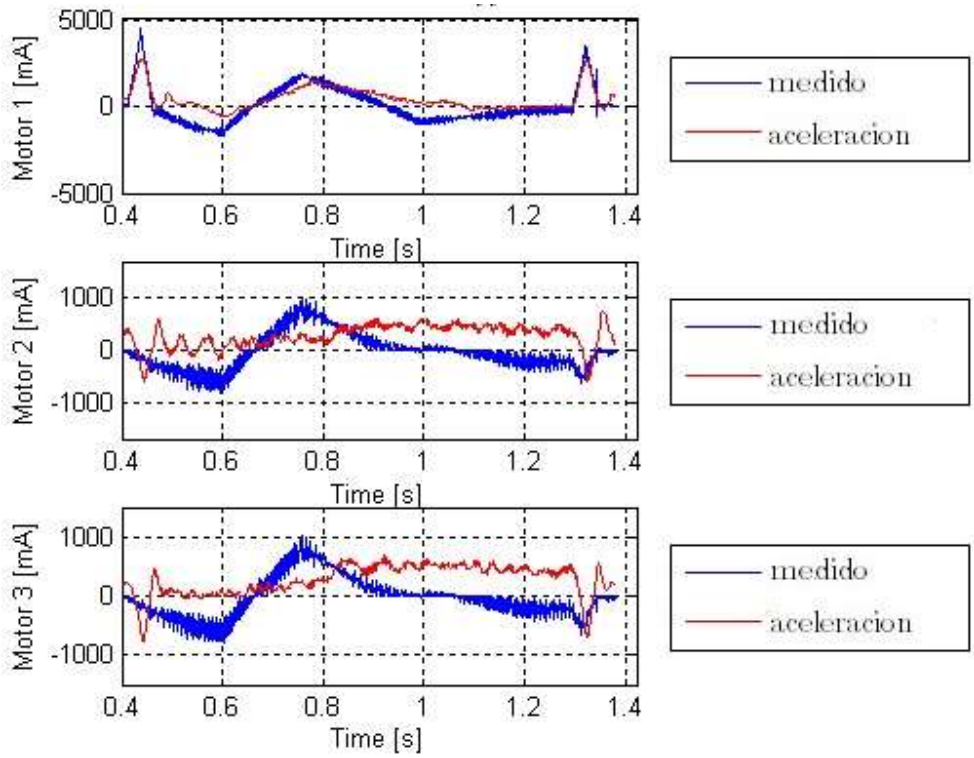


Fig. 106 Medición de Corriente

7.5 CONCLUSION

Se hace entrega de parámetros básicos del sistema, nuevamente se recuerda que la finalidad del proyecto es la interfaz gráfica donde se observa el procesamiento de los tratamiento de los datos entregados por Kinect y construidos en el algoritmo propuesto.

De igual forma, se puede concluir que el sistema presenta una buena respuesta ante la presencia de una planificación de trayectoria, como lo es en este caso un circulo, cabe resaltar que el consumo máximo del sistema se encuentra en el rango de 1 Amper, aunque es importante denotar que presenta un pico alto de corriente cuando el sistema pretende un cambio de aceleración demasiado rápida o de respuesta de movimiento inmediata.

CAPITULO VIII:
CONCLUSIONES GENERALES Y TRABAJO FUTURO

8. CAPITULO VIII: CONCLUSIONES GENERALES Y TRABAJO FUTURO

8.1 RESEÑA

Para el desarrollo de esta tesis se tuvieron las siguientes consideraciones para el correcto desarrollo de la misma, se enlistan con la intensión sea perfectamente replicable por otros investigadores que deseen proseguir con este trabajo en cuanto a trabajos futuros:

8.1.1 CONSIDERACIONES

- Para trabajar en Solidworks el diseño de las piezas se hizo necesario tener en cuenta la instalación de tener en Windows instalado Service pack 2.

- El reconocimiento del Kinect como instalación del software requiere sea instalado los siguientes paquetes y en el mismo orden ya que se encontró que a pesar de ser software libre se encontraron algunas inconsistencias en la ejecución del sistema o reconocimiento del Kinect cuando en la práctica se instalaban en otro orden:

    - Openni-win64-1.5.4.0-dev
    - Sensor-win64-5.1.2.1-redist
    - Nite-win64-1.5.4.0
    - Sensor-win64-5.1.2.1-redist

- Se sugiere trabajar con la versión 1.5.1 de Processing aunque una versión muy antigua se encontró el no correcto funcionamiento del sistema. De igual forma se hace necesario la instalación del jdk-6u21-windows-x64.

- El sistema de Processing solo iniciara con la simulación y ejecución del algoritmo si se tiene conectado la tarjeta Arduino en algún puerto del PC

8.1.2 CONCLUSION GENERAL

El sistema implementado en cuanto a la plataforma delta obedece las especificaciones de diseño como primer objetivo a cumplir, se tuvo en cuenta la creación de un algoritmo que pudiera determinar la forma de operación en cuanto movimiento de la plataforma delta a partir de una plataforma tan común como lo es Kinect.

Cada uno de los capítulos brindo aportes al desarrollo del proyecto en cuanto a aspectos como funcionalidad, análisis y desarrollo, se observó fenómenos de trabajo y se expresó algunas consideraciones durante el desarrollo del presente texto.

8.2 TRABAJO FUTURO

Referente a trabajos propuestos a futuro, el autor se permite sugerir los siguientes desarrollos u optimizaciones del proceso, que superan el alcance planteado por el autor en el presente trabajo de tesis.

- Integración del sistema de simulación y desarrollo de piezas con matlab para crear un entorno de simulación de la plataforma delta para el usuario.

- Otorgamiento de varias planificaciones de trayectorias aplicadas a la plataforma delta pero ignorando el sistema de detección de la mano, ya que no sería coherente crear una trayectoria cuando esta es definida por el movimiento indeterminado del usuario.

- Plantear alguna estrategia de control servo visual de la plataforma con cámara abordo, ya que el sistema aquí planteado presenta cámara afuera.

- Buscar una alternativa de adquisición de imagen diferente a Kinect como el Lip Motion, la cual fue tecnología emergente con la cual no fue planteada el proyecto.

8.3 PUBLICACIONES EN RELACION CON ESTE TRABAJO DE TESIS

El presente proyecto genero divulgación en diferentes eventos como el IV Congreso internacional de Instrumentación control y telecomunicaciones IV CIICT 215 de la Universidad Santo Tomas Tunja

Pero tal vez los logros más importantes del proyecto son:

- Estancia Beca como docente investigador a la Universidad Católica San Pablo Arequipa, Peru. Por medio del programa Alianza del Pacifico, Pronabec peru.

- Publicación artículo ***Robot Dibujante controlado mediante el sensor Kinect*** en la Revista Colombiana de Tecnologías de Avanzada ISSN: 1692-7257 Volumen 2 No 24 -2014 Categoría C. http://www.unipamplona.edu.co/unipamplona/portalIG/home_40/recursos/04_v19_24/revista_24/27072015/06.pdf

CAPITULO IX:
REFERENCIAS

9. CAPITULO IX: REFERENCIAS

[1] S.-D. Stan, M. Manic, C. Szep, and R. Balan, "Performance analysis of 3 DOF Delta parallel robot," *2011 4th Int. Conf. Hum. Syst. Interact. HSI 2011*, pp. 215–220, May 2011.

[2] J. Kovar, O. Andrs, L. Brezina, and V. Singule, "Laboratory delta robot for mechatronic education purposes," pp. 1209–1212, 2012.

[3] H. Of and T. H. E. Delta, "Planning Optimal Motions for a DELTA Parallel Robot," *2006 14th Mediterr. Conf. Control Autom.*, pp. 1–6, Dec. 2006.

[4] J. Mei, Y. Ni, Y. Li, and L. Zhang, "The error modeling and accuracy synthesis of a 3-DOF parallel robot Delta-S," *2009 Int. Conf. Inf. Autom.*, pp. 289–294, 2009.

[5] J. Zhang, "The mathematical model and direct kinematics solution analysis of Delta parallel robot," *2009 2nd IEEE Int. Conf. Comput. Sci. Inf. Technol.*, pp. 450–454, 2009.

[6] J. Gwinnett, "Amusement device," *US Pat. 1,789,680*, 1931.

[7] P. C. Apparatus, "2,286,571 4," 1958.

[8] Y. D. Patel and P. M. George, "Parallel Manipulators Applications — A Survey," *Mordern Mech. Eng.*, vol. 2, no. 3, pp. 57–64, 2012.

[9] D. Stewart, "A platform with six degrees of freedom," *Arch. Proc. Inst. Mech. Eng. 1847-1982 (vols 1-196)*, vol. 180, no. 1965, pp. 371–386, 2006.

[10] K. Cappel, "Motion simulator," *US Pat. 3,295,224*, 1967.

[11] S. Barbara, D. Schmitz, P. Khosla, T. Kanade, R. Cited, K. H. Hunt, C. Victoria, K. Sugimoto, M. Algebra, C. F. Earl, J. Rooney, H. Inoue, T. Second, and I. Symposium, "4,762,016," pp. 1–6, 1988.

[12] J.-P. Merlet, *Parallel Robots*, vol. 208, no. 49. 2006.

[13] R. Clavel, "Dynamic Analysis of Clavel ' S," vol. 0, pp. 4116–4121, 2003.

[14] C. Gosselin, "Kinematic Analysis Optimization and Programming of Parallel Robotic Manipulators.pdf." p. 252, 1985.

[15] M. Ceccarelli, "A new 3 D.O.F. spatial parallel mechanism," *Mech. Mach. Theory*, vol. 32, no. 8, pp. 895–902, 1997.

[16] M. Bouri and R. Clavel, "The linear delta: developments and applications," *Proc. Int. Symp. Robot. Ger. Conf. Robot.*, pp. 1198–1205, 2010.

[17] D. C. Carp-Ciocardia and S. Staicu, "Dynamics of Delta parallel robot with prismatic actuators," *2005 IEEE Int. Conf. Mechatronics, ICM '05*, vol. 2005, pp. 870–875, 2005.

[18] a. Codourey, "Dynamic modelling and mass matrix evaluation of the DELTA parallel robot for axes decoupling control," *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS '96*, vol. 3, pp. 1211–1218, 1996.

[19] M. L. T. Cossio, L. F. Giesen, G. Araya, M. L. S. Pérez-Cotapos, R. L. VERGARA, M. Manca, R. A. Tohme, S. D. Holmberg, T. Bressmann, D. R. Lirio, J. S. Román, R. G. Solís, S. Thakur, S. N. Rao, E. L. Modelado, A. D. E. La, C. Durante, U. N. A. Tradición, M. En, E. L. Espejo, D. E. L. A. S. Fuentes, U. A. De Yucatán, C. M. Lenin, L. F. Cian, M. J. Douglas, L. Plata, and F. Héritier, "No Title No Title," *Uma ética para quantos?*, vol. XXXIII, no. 2, pp. 81–87, 2012.

[20] E. Courteille, D. Deblaise, and P. Maurine, "Design optimization of a delta-like parallel robot through global stiffness performance evaluation," *2009 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS 2009*, pp. 5159–5166, 2009.

[21] G. Ecorchard and P. Maurine, "Self-calibration of Delta parallel robots with elastic deformation compensation," *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, pp. 462–467, 2005.

[22] M. Fikret, A. Calderon, and A. Carlos, *Practical Robot Design Game Playing Robots*. 2014.

[23] J. Humberto and C. Rojas, "Proyecto mecatrónico de brazo robot cartesiano integrado a una celda de almacenamiento y recuperación automatizada AS / RS de un Sistema Flexible de Manufactura FMS," vol. 6, 2009.

[24] H. Kino, "Principle of orthogonalization for completely restrained parallel wire driven robot," *Proc. 2003 IEEE/ASME Int. Conf. Adv. Intell. Mechatronics (AIM 2003)*, no. Aim, pp. 509–514, 2003.

[25] J. Kovar, O. Andrs, L. Brezina, and V. Singule, "Laboratory delta robot for mechatronic education purposes," *Int. Symp. Power Electron. Power Electron. Electr. Drives, Autom. Motion*, pp. 1209–1212, Jun. 2012.

[26] N. Lauzier and C. Gosselin, "3-DOF Cartesian Force Limiting Device Based on the Delta architecture for safe physical human-robot interaction," *Robot. Autom. (ICRA), 2010 IEEE Int. Conf.*, pp. 3420–3425, 2010.

[27] Y. Li and Q. Xu, "Dynamic analysis of a modified DELTA parallel robot for cardiopulmonary resuscitation," *Intell. Robot. Syst. 2005. (IROS 2005). 2005 IEEE/RSJ Int. Conf.*, pp. 233–238, 2005.

[28] H. Lin, C. Wen, S. Lin, Y. Tai, and C. Liu, "Robust Control for a Delta Robot," pp. 880–885, 2012.

[29] O. Linda, S. Member, M. Manic, and S. Member, "Uncertainty-Robust Design of Interval Type-2 Fuzzy Logic Controller for Delta Parallel Robot," vol. 7, no. 4, pp. 661–670, 2011.

[30] O. Linda and M. Manic, "Evaluating uncertainty resiliency of Type-2 Fuzzy Logic Controllers for parallel delta robot," *4th Int. Conf. Hum. Syst. Interact. HSI 2011*, pp. 91–97, 2011.

[31] Y. Lou, G. Liu, and Z. Li, "Randomized Optimal Design of Parallel Manipulators," *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 2, pp. 223–233, Apr. 2008.

[32] Y. Lou, S. Member, Y. Zhang, R. Huang, and X. Chen, "Optimization Algorithms

for Kinematically Optimal Design of Parallel Manipulators," vol. 11, no. 2, pp. 574–584, 2014.

[33] R. P. Manipulators, "Short Papers _____ _____," vol. 20, no. 1, pp. 117–121, 2004.

[34] K. Miller, "Experimental verification of modeling of DELTA robot dynamics by direct application of Hamilton's principle," *Proc. 1995 IEEE Int. Conf. Robot. Autom.*, vol. 1, pp. 532–537, 1965.

[35] M. Müller, T. U. Dortmund, P. Systems, and I. P. S. Professorship, "Cycle Time Estimation for a Delta-Type Robot Factors of Influence," pp. 225–231, 2014.

[36] M. Mustafa, R. Misuari, and H. Daniyal, "Forward Kinematics of 3 Degree of Freedom Delta Robot," *2007 5th Student Conf. Res. Dev.*, no. December, pp. 3–6, 2007.

[37] M. Rachedi, "Application of an H ' control strategy to the Parallel Delta."

[38] J. D. Rueda and L. ??ngel, "Structural analysis of a delta-type parallel industrial robot using Flexible dynamic of ANSYS 11.0," *IECON Proc. (Industrial Electron. Conf.*, pp. 2247–2252, 2009.

[39] D. Rule, "Tuning having Minimum Structure."

[40] D. Schütz, S. Soetebier, P. Last, and A. Raatz, "Adapted Task Configuration of a Deltapod," pp. 545–549.

[41] Shiming Ji, Guan Wang, Zhongfei Wang, Yuehua Wan, and Qiaoling Yuan, "Optimal design of a linear delta robot for the prescribed regular-shaped dexterous workspace," *2008 7th World Congr. Intell. Control Autom.*, pp. 2333–2338, 2008.

[42] U. Thomas and F. M. Wahl, "A system for automatic planning, evaluation and execution of assembly sequences for industrial robots," *Proc. 2001 IEEE/RSJ Int. Conf. Intell. Robot. Syst. Expand. Soc. Role Robot. Next Millenn. (Cat. No.01CH37180)*, vol. 3, pp. 1458–1464, 2001.

[43] Y. Tsumaki, H. Eguchi, and R. Tadakuma, "A novel Delta-type parallel mechanism with wire-pulleys," *2012 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 1567–1572, 2012.

[44] T. Uzunovic, E. Golubovic, E. a. Baran, and A. Sabanovic, "Configuration space control of a parallel Delta robot with a neural network based inverse kinematics," *2013 8th Int. Conf. Electr. Electron. Eng.*, pp. 497–501, Nov. 2013.

[45] E. Yime, "DISEÑO ÓPTIMO DE UN ROBOT PARALELO CON CONFIGURACIÓN DELTA PARA APLICACIONES," pp. 110–119, 2010.

[46] J. Zhang, L. Shi, R. Gao, and C. Lian, "A Method for obtaining direct and inverse pose solutions to Delta parallel robot based on ADAMS," *Mechatronics Autom. 2009. ICMA 2009. Int. Conf.*, pp. 1332–1336, 2009.

[47] C. Thermi, "A DATASET OF KINECT-BASED 3D SCANS Alexandros

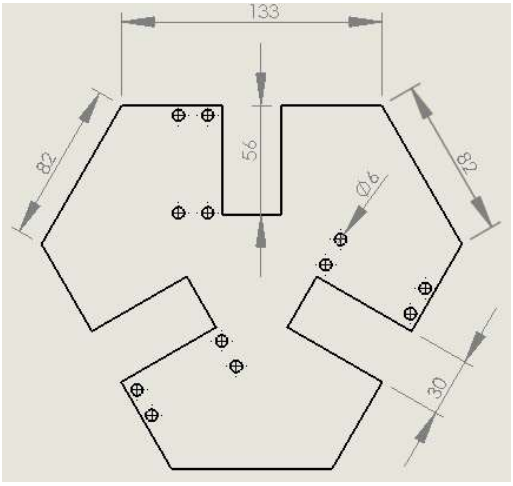
Doumanoglou , Stylianos Asteriadis , Dimitrios S . Alexiadis , Information Technologies Institute , Centre for Research and Technology Hellas ,."

[48] P. Rakprayoon, M. Ruchanurucks, and A. Coundoul, "Kinect-based obstacle detection for manipulator," *2011 IEEE/SICE Int. Symp. Syst. Integr.*, pp. 68–73, Dec. 2011.

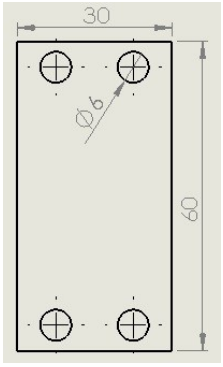
APENDICE A:
DISEÑO DELTA SOLIDWORKS

APENDICE A: DISEÑO DELTA SOLIDWORKS

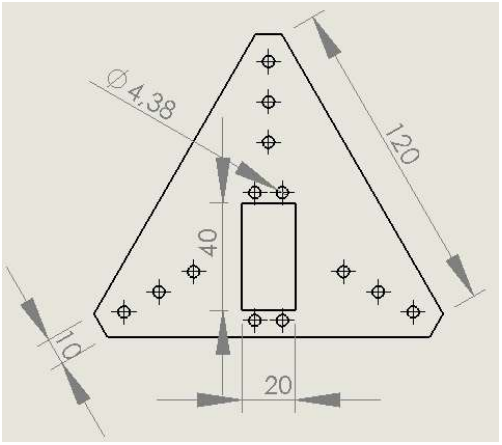
1. BASE FIJA



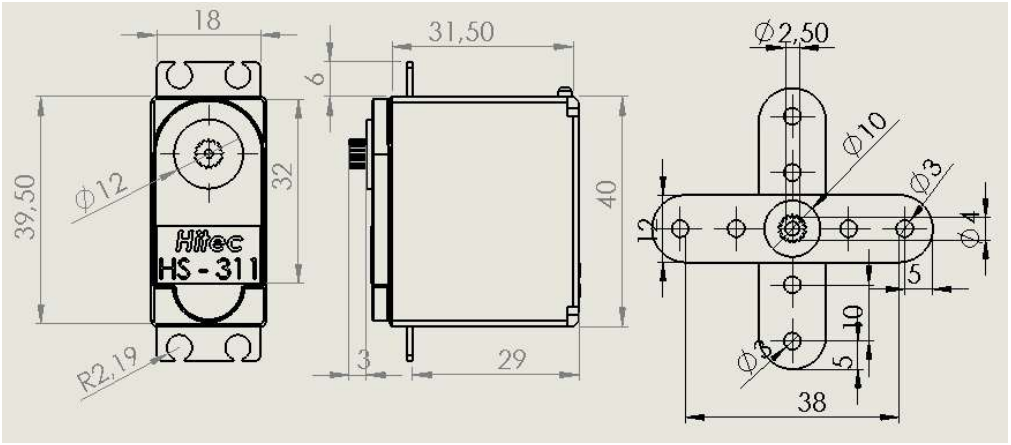
2. SOPORTE MOTORES



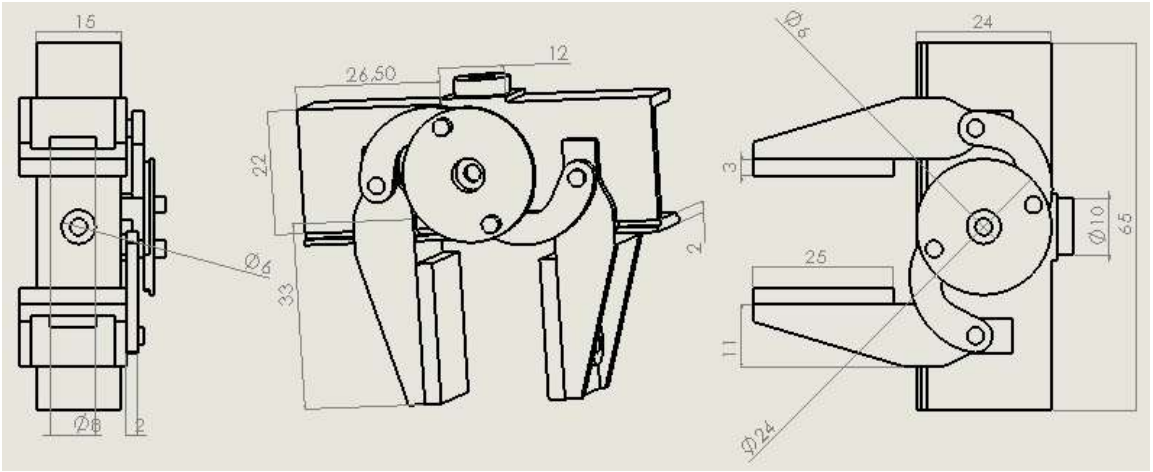
3. BASE MOVIL



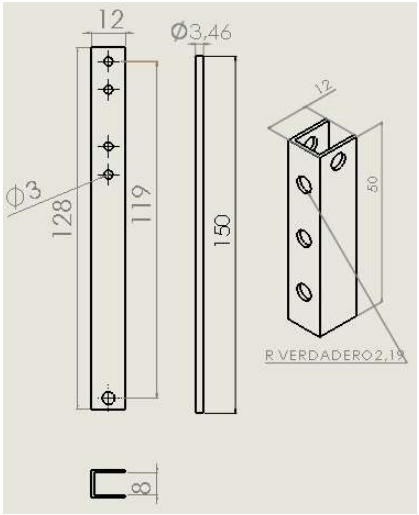
4. SERVOMOTOR Y EJE



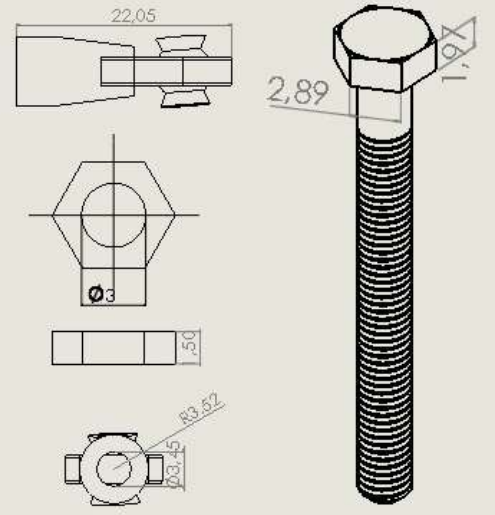
5. PINZA



6. MUSLO, PIERNA Y UNION BASE MOVIL



7. ROD END, TORNILLO Y TUERCA



APENDICE B:
ALGORITMO RECONOCIMIENTO

APENDICE B: ALGORITMO RECONOCIMIENTO

1. CODIGO CLASS DELTAROBOT

```
class DeltaRobot {
        PVector posVec = new PVector(); // Posición del efector
        PVector zeroPos;
        int numLegs = 3; // Numero de piernas

        DeltaLeg[] leg = new DeltaLeg[numLegs]; // Creación de un array para el deltaLegs
        float[] servoAngles = new float[numLegs]; // Guarda los ángulos de cada pierna
        float thigh, shin, baseSize, effectorSize; // Dimensiones del Robot Delta
        float gripRot = 100;
        float gripWidth = 100;
        // Maximas dimensiones del Robot Delta
        float robotSpanX = 500;
        float robotSpanZ = 500;
        float maxH;


        DeltaRobot(float thigh, float shin, float baseSize, float effectorSize) {
                // inicializacion de variables
                this.thigh = thigh;
                this.shin = shin;
                this.baseSize = baseSize;
                this.effectorSize = effectorSize;
                this.maxH = -(thigh + shin) * 0.9f; // Evita que el efector vaya fuera de rango
                zeroPos = new PVector(0, maxH / 2, 0);

                for (int i = 0; i < numLegs; i++) {
                float legAngle = (i * 2 * PI / numLegs) + PI / 6;
                leg[i] = new DeltaLeg(i, legAngle, thigh, shin, baseSize, effectorSize);
                }
        }
```

2. PUBLIC VOID MOVETO

```
public void moveTo(PVector newPos) {
        posVec.set(PVector.add(newPos, zeroPos));

        float xMax = robotSpanX * 0.5f;
        float xMin = -robotSpanX * 0.5f;
        float zMax = robotSpanZ * 0.5f;
        float zMin = -robotSpanZ * 0.5f;
        float yMax = -200;
        float yMin = 2*maxH+200;
        if (posVec.x > xMax) posVec.x = xMax;
        if (posVec.x < xMin) posVec.x = xMin;
        if (posVec.y > yMax) posVec.y = yMax;
        if (posVec.y < yMin) posVec.y = yMin;
        if (posVec.z > zMax) posVec.z = zMax;
        if (posVec.z < zMin) posVec.z = zMin;

        for (int i = 0; i < numLegs; i++) {
                leg[i].moveTo(posVec); // mueve las piernas a la nueva posición
                servoAngles[i] = leg[i].servoAngle; // carga los ángulos para los
        servos
        }
}
```

3. PUBLIC VOID DRAW

```
public void draw() {
        stroke(50);
        pushMatrix();
        translate(0, -maxH, 0);
        for (int i = 0; i < numLegs; i++) {
                leg[i].draw();
                }
        drawEffector();
        popMatrix();
}
```

4. VOID DRAWEFFECTOR

```
void drawEffector() {
// Estructura de Dibujo del Efector
        stroke(150);
        fill(150, 50);
        beginShape();
        for (int i = 0; i < numLegs; i++) {
        vertex(leg[i].ankleVec.x, leg[i].ankleVec.y,
        leg[i].ankleVec.z);
        }
        endShape(CLOSE);

// Dibujo del Gripper
        stroke(200, 200, 255);
        fill(200, 50);

// Traducción del Sistema de coordenadas a la posición del efector
        pushMatrix();
        translate(posVec.x, posVec.y - 5, posVec.z);
        rotateX(-PI/2);
        ellipse(0, 0, effectorSize / 1.2f, effectorSize / 1.2f);
        rotate(map(gripRot, 35, 180, -PI / 2, PI / 2));

for (int j = -1; j < 2; j += 2) {
        translate(0, 2 * j, 0);
        beginShape();
        vertex(-30, 0, 0);
        vertex(30, 0, 0);
        vertex(30, 0, -35);
        vertex(15, 0, -50);
        vertex(-15, 0, -50);
        vertex(-30, 0, -35);
        endShape(CLOSE);

for (int i = -1; i < 2; i += 2) {
        pushMatrix();
        translate(i * 20, 0, -30);
        rotateX(PI / 2);
        ellipse(0, 0, 10, 10);
        rotate(i * map(gripWidth, 50, 150, 0, PI / 2.2f));
        rect(-5, -60, 10, 60);
        translate(0, -50, 0);
        rotate(-i * map(gripWidth, 50, 150, 0, PI / 2.2f));
```

```
        rect(-5, -60, 10, 60);
        popMatrix();
        }
        }

popMatrix();
}
```

5. PUBLIC VOID UPDATEGRIP

```
public void updateGrip(float gripRot, float gripWidth) {
        this.gripRot = gripRot;
        this.gripWidth = gripWidth;
        }
}
```

6. CLASS DELTALEG

```
class DeltaLeg {
        int id; // id de la pierna
        PVector posVec = new PVector(); // Posición del Efector
        float servoAngle; // Angulo del servo y el plano XZ
        float legAngle; // Y el Angulo de rotación de la pierna
// Posición de las junturas universales.
        PVector hipVec, kneeVec, ankleVec;
        float thigh, shin, baseSize, effectorSize; // Tamaño de los elementos del robot
        DeltaLeg(int id, float legAngle, float thigh, float shin, float base, float
effector) {
                this.id = id;
                this.legAngle = legAngle;
                this.baseSize = base;
                this.effectorSize = effector;
                this.thigh = thigh;
                this.shin = shin;
                }
```

7. VOID MOVETO

```
void moveTo(PVector thisPos) {
        posVec.set(thisPos);
        PVector posTemp = vecRotY(thisPos, -legAngle);
// Encontrar la proyección en el plano z = 0, cuadrado
        float a2 = shin * shin - posTemp.z * posTemp.z;
// calcular c con respecto a la base de compensación
        float c = dist(posTemp.x + effectorSize, posTemp.y, baseSize, 0);
```

```
        float alpha = (float) Math.acos((-a2 + thigh * thigh + c * c) / (2 * thigh * c));
        float beta = -(float) Math.atan2(posTemp.y, posTemp.x);
        servoAngle = alpha - beta;
        getWorldCoordinates();
}
```

8. VOID GETWORLDCOORDINATES

```
void getWorldCoordinates () {
        // Vectores sin rotacion de las articulaciones
        hipVec = vecRotY(new PVector(baseSize, 0, 0), legAngle);
        kneeVec = vecRotZ(new PVector(thigh, 0, 0), servoAngle);
        kneeVec = vecRotY(kneeVec, legAngle);
        ankleVec = new PVector(posVec.x + (effectorSize * (float)
Math.cos(legAngle)), posVec.y,
        posVec.z - 5 + (effectorSize * (float) Math.sin(legAngle)));
}

PVector vecRotY(PVector vecIn, float phi) {
        // Giro del vector alrededor del eje "y"
        PVector rotatedVec = new PVector();
        rotatedVec.x = vecIn.x * cos(phi) - vecIn.z * sin(phi);
        rotatedVec.z = vecIn.x * sin(phi) + vecIn.z * cos(phi);
        rotatedVec.y = vecIn.y;
        return rotatedVec;
}

PVector vecRotZ(PVector vecIn, float phi) {
        // Giro del vector alrededor del eje "z"
        PVector rotatedVec = new PVector();
        rotatedVec.x = vecIn.x * cos(phi) - vecIn.y * sin(phi);
        rotatedVec.y = vecIn.x * sin(phi) + vecIn.y * cos(phi);
        rotatedVec.z = vecIn.z;
        return rotatedVec;
}
```

9. PUBLIC VOID DRAW

```
public void draw() {
// Dibuja  tres líneas para indicar el plano de cada pierna
        pushMatrix();
        translate(0, 0, 0);
        rotateY(-legAngle);
        translate(baseSize, 0, 0);
        if (id == 0) stroke(255, 0, 0);
        if (id == 1) stroke(0, 255, 0);
        if (id == 2) stroke(0, 0, 255);
        line(-baseSize / 2, 0, 0, 3 / 2 * baseSize, 0, 0);
        popMatrix();

// Dibuja el elemento del tobillo
        stroke(150);
        strokeWeight(2);
        line(kneeVec.x, kneeVec.y, kneeVec.z, ankleVec.x, ankleVec.y,
        ankleVec.z);
        stroke(150, 140, 140);
        fill(50);
        beginShape();
        vertex(hipVec.x, hipVec.y + 5, hipVec.z);
        vertex(hipVec.x, hipVec.y - 5, hipVec.z);
        vertex(kneeVec.x, kneeVec.y - 5, kneeVec.z);
        vertex(kneeVec.x, kneeVec.y + 5, kneeVec.z);
        endShape(PConstants.CLOSE);
        strokeWeight(1);

// Dibuja el elemento Hip
        stroke(0);
        fill(255);

// Alinea el eje Z a la dirección de la barra
        PVector dirVec = PVector.sub(kneeVec, hipVec);
        PVector centVec = PVector.add(hipVec, PVector.mult(dirVec, 0.5f));
        PVector new_dir = dirVec.get();
        PVector new_up = new PVector(0.0f, 0.0f, 1.0f);
        new_up.normalize();
        PVector crss = dirVec.cross(new_up);
        float theAngle = PVector.angleBetween(new_dir, new_up);
        crss.normalize();
        pushMatrix();
        translate(centVec.x, centVec.y, centVec.z);
```

```
            rotate(-theAngle, crss.x, crss.y, crss.z);

        // rotate(servoAngle);
            box(dirVec.mag() / 50, dirVec.mag() / 50, dirVec.mag());
            popMatrix();
            }
}
```

10. SIMULACION ROBOT DELTA MANIPULACION CON EL MOVIMIENTO DEL MOUSE

```
import processing.opengl.*;
import kinectOrbit.KinectOrbit;
import SimpleOpenNI.*;
// Initialize Orbit and simple-openni Objects
KinectOrbit myOrbit;
SimpleOpenNI kinect;
// Robot Delta
DeltaRobot dRobot;
PVector motionVec;

public void setup() {
        size(1200, 900, OPENGL);
        smooth();
// Orbit
        myOrbit = new KinectOrbit(this, 0, "kinect");
        myOrbit.setCSScale(100);
// Se inicializa el DeltaRobot con las dimensiones reales
        dRobot = new DeltaRobot(250, 430, 90, 80);
}


public void draw() {
        background(0);
        myOrbit.pushOrbit(this); // Inicio de Orbita
        motionVec = new PVector(width/2-mouseX, 0, height/2-mouseY);
        // Establece el vector de movimiento
        dRobot.moveTo(motionVec); // Mueve el robot al vector de movimiento
relativo
        dRobot.draw(); // Dibuja el robot delta en la vista actual.
        myOrbit.popOrbit(this); // detiene la orbita
}
```

11. PROGRAMA COMPLETO ROBOT DELTA PROCESSING Y KINECT

```
import processing.opengl.*;
import processing.serial.*;
import SimpleOpenNI.*;
import kinectOrbit.KinectOrbit;

// Inicializar Orbit y objetos del Simple-openni
KinectOrbit myOrbit;
SimpleOpenNI kinect;
Serial myPort;
boolean serial = true;

// NITE
XnVSessionManager sessionManager;
XnVPointControl pointControl;

// Fuente del texto de la pantalla
PFont font;

// Variables para la detección de la mano
boolean handsTrackFlag;
PVector handOrigin = new PVector();
PVector handVec = new PVector();
ArrayList<PVector> handVecList = new ArrayList<PVector>();
int handVecListSize = 30;
PVector[] realWorldPoint;

// Robot Delta
DeltaRobot dRobot;
PVector motionVec;
float gripRot;
float gripWidth;

private float[] serialMsg = new float[5]; // Envio de los valores para enviar a la tarjeta Arduino


public void setup() {
        size(800, 600, OPENGL);
        smooth();
// Orbita
        myOrbit = new KinectOrbit(this, 0, "kinect");
        myOrbit.drawCS(true);
```

```
        myOrbit.drawGizmo(true);
        myOrbit.setCSScale(100);

// Objeto Simple-openni
        kinect = new SimpleOpenNI(this);
        kinect.setMirror(false);
// Habilita la generación del depthMap, mano + gestos
        kinect.enableDepth();
        kinect.enableGesture();
        kinect.enableHands();

// Configuracion del NITE
        sessionManager = kinect.createSessionManager("Wave", "Wave");
        // Configuración NITE para los puntos de control de la mano
        pointControl = new XnVPointControl();
        pointControl.RegisterPointCreate(this);
        pointControl.RegisterPointDestroy(this);
        pointControl.RegisterPointUpdate(this);
        sessionManager.AddListener(pointControl);

// Matriz para almacenar los puntos escaneados
        realWorldPoint = new PVector[kinect.depthHeight() * kinect.depthWidth()];
        for (int i = 0; i < realWorldPoint.length; i++) {
                realWorldPoint[i] = new PVector();
        }
// Inicializa la Fuente
        font = loadFont("SansSerif-12.vlw");
        // Inicializa el Robot Delta con las dimensiones reales
        dRobot = new DeltaRobot(250, 430, 90, 80);
        if (serial) {
// Inicializa la comunicacion serial
        String portName = Serial.list()[0]; // Esto asigna el puerto de la
computadora.
        myPort = new Serial(this, portName, 9600);
        }
}


public void onPointCreate(XnVHandPointContext pContext) {
        println("onPointCreate:");
        handsTrackFlag = true;
        handVec.set(pContext.getPtPosition().getX(), pContext.getPtPosition()
        .getY(), pContext.getPtPosition().getZ());
        handVecList.clear();
```

```
        handVecList.add(handVec.get());
        handOrigin = handVec.get();
        }
public void onPointDestroy(int nID) {
        println("PointDestroy: " + nID);
        handsTrackFlag = false;
        }
public void onPointUpdate(XnVHandPointContext pContext) {
        handVec.set(pContext.getPtPosition().getX(), pContext.getPtPosition()
        .getY(), pContext.getPtPosition().getZ());
        handVecList.add(0, handVec.get());
        if (handVecList.size() >= handVecListSize) { // remove the last point
        handVecList.remove(handVecList.size() - 1);
        }
}

public void draw() {
        background(0);
// Actualizacion del dato del Kinect
        kinect.update();
// configuracion del  NITE
        kinect.update(sessionManager);
        myOrbit.pushOrbit(this); // inicializa la orbita
        if (handsTrackFlag) {
        updateHand();
        drawHand();
}

// Dibuja el punto de origen, y la línea de la posición actual
        pushStyle();
        stroke(0, 0, 255);
        strokeWeight(5);
        point(handOrigin.x, handOrigin.y, handOrigin.z);
        popStyle();
        stroke(0, 255, 0);
        line(handOrigin.x, handOrigin.y, handOrigin.z, handVec.x, handVec.y,
        handVec.z);
        motionVec = PVector.sub(handVec, handOrigin);// Establece el vector de
        movimiento relativo
        dRobot.moveTo(motionVec); // Mueva el robot al vector de movimiento
relativo
        dRobot.draw(); // Dibuja el robot delta en la vista actual.
```

```
        kinect.drawCamFrustum(); // Dibuja el Kinect
        myOrbit.popOrbit(this); // Detiene la orbita

        if (serial) {
                sendSerialData();
                }
        displayText(); // Imprime los datos en la pantalla
        }


void updateHand() {
// se dibuja el punto del mapa de profundidad 3D
        int steps = 3; // Se puede acelerar el cálculo mediante el uso de menos puntos
        int index;
        stroke(255);

// Inicializa todos los PVectors al centro de la mano
        PVector handLeft = handVec.get();
        PVector handRight = handVec.get();
        PVector handTop = handVec.get();
        PVector handBottom = handVec.get();
        for (int y = 0; y < kinect.depthHeight(); y += steps) {
        for (int x = 0; x < kinect.depthWidth(); x += steps) {
        index = x + y * kinect.depthWidth();
        realWorldPoint[index] = kinect.depthMapRealWorld()[index].get();
        if (realWorldPoint[index].dist(handVec) < 100) {

// Dibuja una nube de puntos de la definición de la mano
        point(realWorldPoint[index].x,                realWorldPoint[index].y,
realWorldPoint[index].z);
        if (realWorldPoint[index].x > handRight.x) handRight =
realWorldPoint[index].get();
        if (realWorldPoint[index].x < handLeft.x) handLeft =
realWorldPoint[index].get();
        if (realWorldPoint[index].y > handTop.y) handTop =
realWorldPoint[index].get();
        if (realWorldPoint[index].y < handBottom.y) handBottom =
realWorldPoint[index].get();
                }
        }
}
```

```
// Dibuja el Control Cube
        fill(100, 100, 200);
        pushMatrix();
        translate(handVec.x, handVec.y, handVec.z);
        rotateX(radians(handTop.y - handBottom.y));
        box((handRight.x - handLeft.x) / 2, (handRight.x - handLeft.x) / 2, 10);
        popMatrix();


// Se establecen los parámetros de los robots
        gripWidth = lerp(gripWidth, map(handRight.x - handLeft.x, 65, 200, 0, 255),
0.2f);
        gripRot = lerp(gripRot, map(handTop.y - handBottom.y, 65, 200, 0, 255),
0.2f);
        dRobot.updateGrip(gripRot, gripWidth);
}
void drawHand() {
        stroke(255, 0, 0);
        pushStyle();
        strokeWeight(6);
        point(handVec.x, handVec.y, handVec.z);
        popStyle();
        noFill();
        Iterator itr = handVecList.iterator();
        beginShape();
        while (itr.hasNext ()) {
                PVector p = (PVector) itr.next();
                vertex(p.x, p.y, p.z);
        }
        endShape();
}
```

12. PROGRAMA ADICIONAL ENVIO DATOS TARJETA ARDUINO UNO DESDE PROCESSING

```
void sendSerialData() {
        myPort.write('X');
        for (int i=0;i<dRobot.numLegs;i++) {
        int serialAngle = (int)map(dRobot.servoAngles[i], radians(-90), radians(90),
0, 2000);
        serialMsg[i] = serialAngle;
        byte MSB = (byte)((serialAngle >> 8) & 0xFF);
        byte LSB = (byte)(serialAngle & 0xFF);
        myPort.write(MSB);
```

```
        myPort.write(LSB);
        }
        myPort.write((int)(gripRot));
        serialMsg[3] = (int)(gripRot);
        myPort.write((int)(gripWidth));
        serialMsg[4] = (int)(gripWidth);
        }
void displayText() {
        fill(255);
        textFont(font, 12);
        text("Position X: " + dRobot.posVec.x + "\nPosition Y: " + dRobot.posVec.y
        + "\nPosition Z: " + dRobot.posVec.z, 10, 20);
        text("Servo1: " + serialMsg[0] + "\nServo2: " + serialMsg[1]
        + "\nServo3: " + serialMsg[2] + "\nGripRot: " + serialMsg[3]
        + "\nGripWidth: " + serialMsg[4], 10, 80);
}
```

APENDICE C:
ALGORITMO ARDUINO

APENDICE C: ALGORITMO ARDUINO

1. CODIGO ARDUINO

```
unsigned int tempHandRot, tempGrip;
unsigned int servo1Pos, servo2Pos, servo3Pos;
int ledPin = 3;
int servo1Pin = 9;
int pulse1 = 1500;
int servo2Pin = 10;
int pulse2 = 1500;
int servo3Pin = 11;
int pulse3 = 1500;
int handRotPin = 5;
int handRotPulse = 1500;
int gripPin = 6;
int gripPulse = 1500;
long previousMillis = 0;
long interval = 20;
int speedServo1 = 0;
int speedServo2 = 0;
int speedServo3 = 0;
int handRotSpeed = 20;
int gripSpeed = 20;


void setup() {
        pinMode (ledPin, OUTPUT);
        pinMode (servo1Pin, OUTPUT);
        pinMode (servo2Pin, OUTPUT);
        pinMode (servo3Pin, OUTPUT);
        pinMode (handRotPin, OUTPUT);
        pinMode (gripPin, OUTPUT);

        Serial.begin(9600); // Start serial communication at 9600 bps
        }

void loop() {
        digitalWrite(ledPin, HIGH);
        if (Serial.available()>8) { // If data is available to read,
                char led=Serial.read();
                if (led=='X'){
                        byte MSB1 = Serial.read();
                        byte LSB1 = Serial.read();
```

```
                    servo1Pos = word(MSB1, LSB1);
                    byte MSB2 = Serial.read();
                    byte LSB2 = Serial.read();
                    servo2Pos = word(MSB2, LSB2);

                    byte MSB3 = Serial.read();
                    byte LSB3 = Serial.read();
                    servo3Pos = word(MSB3, LSB3);

                    tempHandRot = Serial.read();
                    tempGrip = Serial.read();
                    }
                }

        pulse1 = (int)map(servo1Pos,0,2000,500,2500);
        pulse2 = (int)map(servo2Pos,0,2000,500,2500);
        pulse3 = (int)map(servo3Pos,0,2000,500,2500);
        handRotPulse = (int)map(tempHandRot,0,200,2500,500);
        gripPulse = (int)map(tempGrip,0,220,500,2500);


        unsigned long currentMillis = millis();
        if(currentMillis - previousMillis > interval) {
                previousMillis = currentMillis;
                updateServo(servo1Pin, pulse1);
                updateServo(servo2Pin, pulse2);
                updateServo(servo3Pin, pulse3);
                updateServo(handRotPin, handRotPulse);
                updateServo(gripPin, gripPulse);
                }
    }

    void updateServo (int pin, int pulse){
            digitalWrite(pin, HIGH);
            delayMicroseconds(pulse);
            digitalWrite(pin, LOW);
}
```