

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
Departamento de Ingeniería Mecánica



**VALIDACIÓN DE MODELACIÓN CINEMÁTICA Y DINÁMICA DE UN ROBOT
DELTA 3 GDL A TRAVEZ DE ROS Y ADAMS**

Iván Alejandro Fernández Gracia
Rodrigo Eduardo Soto Castro

Profesor guía:
Michael Gabriel Miranda Sandoval

Trabajo de Titulación presentado en conformidad a los requisitos para obtener el Título de Ingeniero Civil en Mecánica.

Santiago – Chile
2020

© Iván Alejandro Fernández Gracia, 2020
© Rodrigo Eduardo Soto Castro, 2020

RESUMEN

Esta tesis describe, crea y valida el modelamiento cinemático y dinámico de un robot paralelo tipo delta de 3 grados de libertad.

En los últimos años ha ido ganando popularidad el uso de la cinemática junto con la dinámica para el control de robots paralelos. El uso de robots paralelos dedicados a las operaciones de 'pick and place' abrió una serie de nuevas perspectivas en el dominio del manejo rápido y preciso de objetos ligeros. Estos son utilizados en diversas áreas en la industria, tales como la agronomía, fabricación, laboratorios farmacéuticos, etc.

Las problemáticas que se identifican acerca del robot delta y que se abordan en esta tesis son principalmente: los sistemas robóticos gestionan una gran complejidad, distinta para cada tarea a realizar y esto trae como consecuencia lentitud y grandes costos en el desarrollo e implementación de la robótica a escala mundial; los esquemas de controles de robots basados solo en la cinemática de posición no son suficientes para una excelente precisión y la dificultad de establecer un modelo dinámico simple que pueda calcularse fácilmente en tiempo real. Por ende, en este trabajo se propone el uso un middleware gratuito orientado a la reutilización de código y control de robots; dos métodos para la modelación cinemática y dinámica; espacio de trabajo donde ejecuta las tareas el robot; visualización en 3D de las partes mecánicas; simulación de trayectorias para comprobar los métodos y validación de los modelos por medio de software de simulación educacional.

Los resultados muestran que el modelamiento cinemático y dinámico de los dos métodos dan valores idénticos y es válido según el software de simulación. Las diferencias despreciables entre los resultados de los modelos y el software de simulación se deben a que este último solo se aproxima al modelo, en otras palabras, no es totalmente idéntico al modelo.

Palabras clave: Robot Delta, Robot Operating System, ROS, ADAMS, Robot Paralelo, Cinemática, Dinámica, Jacobiano, RViz, Espacio de Trabajo, Robótica.

ABSTRACT

This thesis describes, creates and validates the kinematic and dynamic modeling of a delta-type parallel robot with 3 degrees of freedom.

In recent years, the use of kinematics in conjunction with dynamics for the control of parallel robots has been gaining popularity. The use of parallel robots dedicated to 'pick and place' operations opened a series of new perspectives in the domain of fast and precise handling of light objects. These are used in various areas in industry, such as agronomy, manufacturing, pharmaceutical laboratories, etc.

The problems that are identified about the delta robot and that are solved in this thesis are mainly: robotic systems manage a great complexity, different for each task to be performed and this results in slowness and high costs in the development and implementation of robotics worldwide; robot control schemes based only on position kinematics are not enough for excellent precision and the difficulty of establishing a simple dynamic model that can be easily calculated in real time. Therefore, this work proposes the use of a free middleware oriented to the reuse of code and control of robots; two methods for kinematic and dynamic modeling; workspace where the robot executes tasks; 3D visualization of mechanical parts; trajectory simulation to check the methods and validation of the models by means of educational simulation software.

The results show that the kinematic and dynamic modeling of the two methods give identical values and is valid according to the simulation software. The negligible differences between the results of the models and the simulation software are due to the fact that the latter only approximates the model, in other words, it is not totally identical to the model.

Keywords: Delta Robot, Robot Operating System, ROS, ADAMS, Parallel Robot, Kinematics, Dynamics, Jacobian, Rviz, Workspace.

DEDICATORIA

A MIS PADRES MARCELO Y MARCELA
A MI HERMANA MARÍA
A MIS ABUELOS LUIS Y ADELA
A MIS FAMILIARES
A MIS GRANDES AMIGOS

ATENTAMENTE, IVÁN ALEJANDRO FERNÁNDEZ GRACIA

A MIS PADRES OSCAR Y LUPE
A MI HERMANA GABRIELA
A MIS HERMANOS OSCAR Y MARCELO
A MI NOVIA CAMILA
A TODOS LOS QUE ME APOYARON

ATENTAMENTE, RODRIGO EDUARDO SOTO CASTRO

Zeroth Law: “A robot may not harm humanity, or, by inaction, allow humanity to come to harm.”

ISAAC ASIMOV (1920 - 1992)

Agradecimientos

Este trabajo de título realizado en la Universidad de Santiago de Chile es un esfuerzo en el cual, directa o indirectamente, participaron varias personas. Es por eso que dedico unas palabras a todas estas personas que me acompañaron en este largo camino.

En primer lugar agradecer a los profesores Michael Miranda, Héctor Muñoz y Ricardo Manríquez, quienes me motivaron a introducirme en el área de la robótica.

Agradecer al profesor Francisco Sepúlveda Palma, quien fue uno de los primeros académicos que confió en mi como alumno y persona. Me ofreció realizar mi primera ayudantía importante en mi carrera.

A mis amigos de universidad, Raul, Alejandro, Cristian, Claudio y Elias por las risas y alegrías que vivimos en los pasillos de la universidad.

Un especial agradecimiento a Pedro Godoy, mi profesor del colegio Ingles Saint Jonh de Rancagua, quien fue el principal responsable de mi formación matemática y me salvo de ser una persona rebelde.

A mis amigos y amigas, Juan Andres, Jose Miguel, Nicolás, Lucas, Pablo, Sebastian, Chung Chii, Javier, Nataly, Camila Alejandra y Maria Fernanda por distraerme de mis estudios para disfrutar de la vida y por estar junto a mi en los momentos que mas necesitaba apoyo.

A mi hermana Maria, por soportar mi desorden en el departamento donde vivíamos.

A mis padres Marcelo y Marcela, por alimentarme, preocuparse por mi educación y darme amor incondicional.

Atentamente, Iván Alejandro Fernández Gracia

Tabla de Contenido

RESUMEN	I
ABSTRACT	II
DEDICATORIA	III
Agradecimientos	IV
Tabla de Contenido	XI
Índice de Tablas	XIII
Índice de Figuras	XX
1. Introducción	1
1.1. Motivación	1
1.2. Hipótesis de estudio	2
1.3. Objetivos	2
1.3.1. Objetivo general	2
1.3.2. Objetivos específicos	2
1.4. Alcance de la propuesta y limitaciones	3
1.5. Estructura de la tesis	3
2. Estado del arte	5
2.1. Introducción a la robótica	5
2.2. Historia de la robótica	7
2.3. Clasificaciones de robots	10
2.3.1. Clasificación por generación	10
2.3.1.1. Primera generación	10
2.3.1.2. Segunda generación	10
2.3.1.3. Tercera generación	10
2.3.1.4. Cuarta generación	10
2.3.2. Clasificación por arquitectura o estructura	11
2.3.2.1. Poli-articulados	11

2.3.2.2. Zoomórficos	11
2.3.2.3. Móviles	12
2.3.2.4. Androides	12
2.3.3. Clasificación por su movimiento	13
2.3.3.1. Robots articulados	13
2.3.3.2. Robots cartesianos	13
2.3.3.3. Robots SCARA	14
2.3.3.4. Robots delta	14
2.3.3.5. Robots cilíndricos	15
2.3.3.6. Robots polares	15
2.4. Estadísticas y aplicaciones	16
3. Arquitectura de un robot delta	23
3.1. Funcionamiento general	23
3.2. Estructura de un robot delta	25
3.3. Partes mecánicas	27
3.3.1. Investigación	27
3.3.2. Elección del concepto “Delta”	28
3.3.3. Descripción del concepto “Delta” y sus componentes	29
3.4. Software Robor Operating System (ROS)	30
3.4.1. Historia	30
3.4.2. Problema Principal	32
3.4.3. Que es ROS	33
3.4.4. Objetivos de ROS	35
3.4.4.1. Peer to peer (Nodos)	35
3.4.4.2. Orientación hacia las herramientas	35
3.4.4.3. Multilenguaje	36
3.4.4.4. Pequeño	37
3.4.4.5. Libre y Open Source	37
3.4.5. Estadísticas	38
3.4.5.1. Science Robotics	38
3.4.5.2. Metricas de la comunidad ROS	40
3.4.5.3. Repositorio GitHub	41
3.4.6. Versiones	42
3.4.7. Conceptos principales	43
3.4.7.1. Nivel de sistemas de archivos	44
3.4.7.2. Nivel gráfico computacional	47
3.4.7.2.1. Nodes (Nodos)	47
3.4.7.2.2. Ros Master	47
3.4.7.2.3. Topics (Temas)	48
3.4.7.2.4. Services (Servicios)	49
3.4.7.2.5. Actions (Acciones)	50

3.4.7.2.6. Messages (Mensajes)	51
3.4.7.2.7. Parameter Server (Servidor de parámetros)	51
3.4.7.2.8. Bags (Bolsas)	51
3.4.7.2.9. Esquema resumen de nivel gráfico	52
3.4.7.3. Nivel Comunitario	53
3.4.8. Herramientas	54
3.4.8.1. Launch Files	54
3.4.8.2. RQT Graph	54
3.4.8.3. RQT Rosbag	55
3.4.8.4. RQT reconfigure	55
3.4.8.5. Movelt	55
3.4.8.6. Gazebo	56
3.5. Visualización	57
3.5.1. Interfaz de visualización gráfica: ROS visualization (RViz)	57
3.5.2. Paquete tf	58
3.5.3. Unified Robot Description Format (URDF)	61
3.5.3.1. Introducción URDF	61
3.5.3.2. Links	62
3.5.3.3. Joints	63
3.5.4. Herramientas complementarias	65
3.6. ADAMS (Automated Dynamic Analysis of Mechanical Systems)	66
3.6.1. Historia	66
3.6.2. introducción	66
3.6.3. Descripción del ambiente de ADAMS	67
3.6.3.1. Inicio de un nuevo modelo y su configuración	67
3.6.4. Modelo y jerarquía de los datos en ADAMS	70
3.6.4.1. Convención de nombres en ADAMS	71
3.6.4.2. Sistemas de coordenadas	72
3.6.4.2.1. Sistema de coordendas de una Pieza (PSC)	72
3.6.4.2.2. Sistema de coordendas de un Marker	72
3.6.4.3. Parts	73
3.6.4.4. Constrains / Joints	74
3.6.4.5. Measures	74
3.6.5. Adams/Solver	75
4. Modelación física y matemática	78
4.1. Desarrollo de la modelación y métodos	78
4.2. Nomenclatura y sistema de referencia global	79
4.3. Método A	80
4.3.1. Modelación cinemática de posición	80
4.3.1.1. Nomenclatura de parámetros geométricos y sistema de referencia local	80

4.3.1.2. Cinemática directa	82
4.3.1.3. Cinemática inversa	84
4.3.2. Modelación cinemática de la velocidad	87
4.3.2.1. Nomenclatura de parámetros geométricos y sistema de referencia local	87
4.3.2.2. Vectorización y ángulos de las articulaciones	90
4.3.2.3. Jacobiano	92
4.3.2.4. Singularidades	93
4.3.3. Modelación cinemática de aceleración	94
4.3.4. Modelación dinámica	96
4.3.4.1. Teoría de las ecuaciones de Lagrange	96
4.3.4.2. Nomenclatura de parámetros geométricos y sistema de referencia global	98
4.3.4.3. Dinámica inversa	100
4.4. Método B	103
4.4.1. Modulación cinemática de la posición	103
4.4.1.1. Nomenclatura de parámetros geométricos y sistema de referencia local	103
4.4.1.2. Cinemática directa	106
4.4.1.3. Cinemática inversa	109
4.4.2. Modelación cinemática de la velocidad	112
4.4.2.1. Jacobiano	112
4.4.3. Modelación cinemática de aceleración	115
4.4.4. Modelación dinámica	116
4.4.4.1. Introducción a modelos dinámicos	116
4.4.4.2. Principio de trabajo virtual	117
4.4.4.3. Simplificaciones e hipótesis	118
4.4.4.4. Nomenclatura de parámetros geométricos, simplificación de masas y sistema de referencia local	119
4.4.4.5. Dinámica inversa	121
4.5. Espacio de trabajo	122
4.5.1. Definición	122
4.5.2. Métodos	123
4.5.3. Tipos de restricciones	124
4.6. Trayectorias	125
4.6.1. Definición de trayectorias	125
4.6.2. Nomenclatura de trayectorias	126
4.6.3. Objetivo y procedimiento de generación de trayectorias	127
4.6.4. Clasificación de trayectorias	128
4.6.4.1. Según el espacio	128
4.6.4.2. Según la geometría del camino	129
4.6.4.3. Según la ley temporal	130

4.6.4.4. Según la coordinación	130
4.6.4.5. Según el tipo de tarea	131
4.6.5. Trayectorias punto a punto	131
4.6.5.1. Trayectorias en Linea Recta	131
4.6.5.2. Escala temporal de un camino en línea recta	133
4.6.5.2.1. Escala de tiempo polinomial	133
4.6.5.2.2. Perfiles de movimiento trapezoidal	135
5. Especificaciones del robot delta seleccionado	138
6. Desarrollo de la solución	140
6.1. Software ROS	143
6.1.1. Nodo principal	145
6.1.2. Método A	147
6.1.2.1. Cinemática directa	148
6.1.2.2. Cinemática inversa	149
6.1.2.3. Jacobiano	151
6.1.2.4. Modelacion cinematica de Aceleración	153
6.1.2.5. Dinámica Inversa	156
6.1.3. Método B	158
6.1.3.1. Cinemática directa	159
6.1.3.2. Cinemática inversa	160
6.1.3.3. Jacobiano	162
6.1.3.4. Modelacion Cinematica de Aceleracion	164
6.1.3.5. Dinámica Inversa	166
6.1.4. Espacio de Trabajo	168
6.1.5. Trayectoria	172
6.2. Interfaz de visualización RViz	178
6.2.1. Conexión entre ROS y RViz	178
6.2.2. Modelo URDF	180
6.2.2.1. Base Fija	180
6.2.2.2. Brazos	181
6.2.2.3. Antebrazos	183
6.2.2.4. Base Movil	185
6.3. ADAMS	187
6.3.1. Consideraciones preliminares	188
6.3.2. Desarrollo del modelo	188
6.3.2.1. Base fija	190
6.3.2.2. Brazo	195
6.3.2.3. Antebrazo	197
6.3.2.4. Base móvil	198
6.3.2.5. Trayectorias	203

6.3.3. Desarrollo de la simulación	205
6.3.3.1. Masas	205
6.3.3.2. Movimiento	206
6.3.3.3. Mediciones	209
7. Resultados	211
7.1. Visualizador	211
7.1.1. Temas y Nodo	211
7.1.2. Enlaces y Juntas	213
7.1.3. Estructura de arbol URDF	214
7.2. Espacio de Trabajo	215
7.2.1. Temas y Nodo	215
7.2.2. Espacio de trabajo	216
7.2.3. Puntos Alcanzables	216
7.2.4. Proyección plano XY	217
7.2.5. Proyección plano XZ	217
7.2.6. Singularidad J_x	218
7.2.7. Singularidad J_θ	218
7.3. Trayectorias	219
7.3.1. Temas y Nodo	219
7.3.2. Trayectoria 1	220
7.3.3. Trayectoria 2	220
7.3.4. Trayectoria 3	221
7.3.5. Trayectoria 4	221
7.3.6. Trayectoria 5	222
7.3.7. Trayectoria 6	222
7.3.8. Trayectoria 7	223
7.3.9. Trayectoria 8	223
8. Análisis de resultados y proyecciones a futuro	224
8.1. Conclusiones y análisis de resultados	224
8.2. Futuras líneas de investigación	228
BIBLIOGRAFÍA	229
APÉNDICE	
A. Desarrollo de fórmulas teóricas	B.1
A.1. Método A	B.1
A.1.1. Modelación cinemática de posición	B.1
A.1.1.1. Cinemática directa	B.1
A.1.1.2. Cinemática inversa	B.9
A.1.2. Modelación cinemática de velocidad	B.16

A.1.2.1.	Vectorización y ángulos de juntas	B.16
A.1.2.2.	Jacobiano	B.18
A.1.3.	Modelación Cinemática Aceleración (A)	B.21
A.1.4.	Modelación dinámica	B.25
A.1.4.1.	Restricciones	B.25
A.1.4.2.	Lagrangiano, energía cinética y energía potencial	B.26
A.1.4.3.	Multiplicadores de Lagrange	B.27
A.1.4.4.	Torque	B.31
A.2.	Método B	B.34
A.2.1.	Modelación cinemática de aceleración	B.34
A.2.2.	Modelación dinámica	B.37
A.2.2.1.	Parámetros dinámicos	B.37
A.2.2.2.	Dinámica inversa	B.38
B.	Carpetas, comandos y códigos	B.1
B.1.	Carpetas creadas por catkin make	B.1
B.2.	Comandos para compilar nodos en shell	B.2
B.3.	CMakeLists.txt (/simu_visual)	B.5
B.4.	Package.xml (/simu_visual)	B.6
B.5.	Mensajes (/msg)	B.7
B.5.1.	linear_speed_xyz.msg	B.7
B.5.2.	matriz_path_ls.msg	B.7
B.5.3.	parameter_ws.msg	B.7
B.6.	robot_urdfm1_adams.urdf (/urfd)	B.8
B.7.	rviz_tm1_adams.launch (/launch)	B.12
B.8.	Python (/script)	B.13
B.8.1.	pd_tm1_adams.py	B.13
B.8.2.	path_tm1_v2_adams.py	B.19
B.8.3.	linear_speed_f_adams.py	B.25
B.8.4.	trans_rot_ls_adams.py	B.29
B.8.5.	delta_kinematics_t1m_adams.py	B.36
B.8.6.	delta_kinematics_Paderborn_tm1_adams.py	B.40
B.8.7.	jacobian_tm1_adams.py	B.45
B.8.8.	jacobian_Paderborn_tm1_v2_adams.py	B.59
B.8.9.	torque_m1_adams.py	B.69
B.8.10.	torque_m1_Paderborn_v2_adams.py	B.74
B.8.11.	workspace_v2.py	B.79
B.8.12.	posicionador_rviz_realtime_tm1_adams.py	B.85
B.8.13.	codos_tm1_adams.py	B.89

Índice de Tablas

Tabla 2.1. Evolución de la robótica industrial	9
Tabla 4.1. Relación entre la numeración y piezas mecánicas de la figura (4.2).	79
Tabla 4.2. Relación entre la numeración y piezas mecánicas de la figura (4.3).	80
Tabla 4.3. Parámetros geométricos y puntos para cinemática de posición del método A	81
Tabla 4.4. Posición de los centros de las esferas originadas por la traslación de las juntas J_i	83
Tabla 4.5. Coordenadas del centro de las esferas y sus radios utilizados para la solución de la cinemática inversa del método A.	85
Tabla 4.6. Relación entre la numeración y piezas mecánicas de la figura (4.9)	87
Tabla 4.7. Simbología y descripción de las longitudes, puntos, ángulos y sistemas de referencia para el desarrollo de la solución cinemática de velocidad del método A.	89
Tabla 4.8. Descripción de los vectores de la figura (4.11)	90
Tabla 4.9. Relación entre la numeración y piezas mecánicas de la figura (4.13)	98
Tabla 4.10. Simbología y descripción de los sistema de referencia, longitudes, puntos y ángulos principales para el desarrollo de la dinámica inversa del método A.	99
Tabla 4.11. Nombres de partes mecánicas del robot delta del método B.	103
Tabla 4.12. Principales parámetros geométricos y puntos para la solución de la cinemática del método B.	104
Tabla 4.13. Vectorización para la solución de cinemática del método B	105
Tabla 4.14. Parámetros para la solución de la dinámica inversa del método B.	119
Tabla 4.15. Masas, relación de división de masas e inercia de los motores para la solución de la dinámica inversa del método B.	120
Tabla 5.1. Parámetros necesarios para la simulación del robot delta en esta tesis	138
Tabla 6.1. Trayectorias simuladas	146
Tabla 6.2. Restricciones del espacio de trabajo	170
Tabla 6.3. Nombre de juntas del robot delta en URDF	179
Tabla 6.4. Posición XYZ de los puntos que conforman la base fija	190
Tabla 6.5. Posición XYZ de los motores	193

Tabla 7.1. Mensaje matriz_path_ls.msg utilizado por tema m_txyzth12.	212
Tabla 7.2. Mensaje joint_state.msg utilizado por tema joint_state.	212
Tabla 7.3. Mensaje parameter_ws.msg utilizado por el tema input_workspace .	215
Tabla 7.4. Mensaje linear_speed_xyz.msg utilizado por el tema input_ls_final .	219
Tabla A.1. Centro de esferas J'_i que representan la traslación de las juntas J_i . .	B.3
Tabla A.2. Centro de esferas que representa las juntas en los puntos E'_i y F_i . .	B.12

Índice de Figuras

Figura 2.1. Robot humanoide Sophia, IA [1]	5
Figura 2.2. Obra R.U.R (Robots Universal Rossum) creada por Karel Capek [2]	6
Figura 2.3. Reymond Clavel y el robot delta [3]	9
Figura 2.4. Robot Poli-articulado [4]	11
Figura 2.5. Robot zoomórfico: Spot Classic (2015) de Boston Dynamics [5] . .	11
Figura 2.6. Robot móvil: iRobot 510 PackBot.[6]	12
Figura 2.7. Robot androide: Atlas de Boston Dynamics [7]	12
Figura 2.8. Robot Articulado [8]	13
Figura 2.9. Robot Cartesiano [8]	13
Figura 2.10.Robot SCARA [8]	14
Figura 2.11.Robot delta [9][10][11]	14
Figura 2.12.Robot cilíndrico [12][13][14]	15
Figura 2.13.Robot polar [12][13]	15
Figura 2.14.Logo de McKinsey Company [15]	16
Figura 2.15.El potencial técnico para la automatización en los EE. UU. (McKinsey's Company 2017) [16]	17
Figura 2.16.La distribución del empleo ocupacional de BLS (Oficina de estadísticas laborales) 2010 sobre la probabilidad de computarización, junto con la participación en las categorías de probabilidad baja, media y alta. El área total bajo todas las curvas es igual al empleo total de los EE.UU [17].	18
Figura 2.17.Envíos anuales (en miles) de robots industriales en todo el mundo [18].	19
Figura 2.18.Solicitudes de patentes anuales para tecnologías de fabricación avanzadas específicas [19]	20
Figura 2.19.Mapa del mercado de robótica industrial [20]	21
Figura 3.1. Ejemplo de diagrama de flujo de tareas que realiza un robot delta para realizar una trayectoria específica	24
Figura 3.2. Descomposición estructural del robot delta [21]	25
Figura 3.3. Ejemplo de el sistema de control de un robot delta	26
Figura 3.4. Representación de la función general del robot [22]	27
Figura 3.5. Extracto de las soluciones del catálogo ubicado en el apéndice A.2.2 [22]	28

Figura 3.6. Nombre, movimiento relativo. grados de libertad y símbolos de juntas dibujadas en la figura (3.5) [22]	28
Figura 3.7. Descripción visual de un robot delta [22]	29
Figura 3.8. Logo de ROS [23]	30
Figura 3.9. OSRF [24]	30
Figura 3.10. Fundadores y contribuidores de ROS	31
Figura 3.11. Logo Stanford University [25]	31
Figura 3.12. Programación Secuencial	32
Figura 3.13. Diferencias entre un sistema operativo convencional y ROS	33
Figura 3.14. Compatibilidad de ROS y otras plataformas [26]	34
Figura 3.15. Modelo de comunicación ROS: nodos publican y se suscriben a temas [27]	35
Figura 3.16. Ejemplo de multilenguaje de ROS	36
Figura 3.17. Licencia de ROS	37
Figura 3.18. Estadísticas mundiales de ROS [28]	38
Figura 3.19. Desarrollo de ROS en los últimos 10 años [28]	39
Figura 3.20. Logo de ROS Metrics [29]	40
Figura 3.21. Top de países que utilizan ROS en base a la descarga de paquetes registrados hasta el 28 de marzo del año 2021 [29].	40
Figura 3.22. Colección de diferentes métricas para medir la cantidad de usuarios de la comunidad ROS [29]	40
Figura 3.23. Logo GitHub, Inc.	41
Figura 3.24. Visualización de las contribuciones de ROS en GitHub [30]	41
Figura 3.25. Línea temporal e iconos de tortuga para cada versión de ROS [26] . .	42
Figura 3.26. Uso relativo de cada versión de ROS basado en descargas desde packages.ros.org [29].	42
Figura 3.27. Niveles de conceptos de ROS	43
Figura 3.28. Nivel de sistema de archivos ROS [31]	44
Figura 3.29. Espacio de trabajo catkin_make [32]	45
Figura 3.30. Estructura y características de catkin_make [32]	46
Figura 3.31. Estructura de la capa ROS Graph [31]	47
Figura 3.32. Registro de nodos en ROS Master [33].	47
Figura 3.33. Nodos comunicándose a través de topic por medio de un mensaje *.msg [33].	48
Figura 3.34. Comunicación de mensajes por medio de topics [26]	48
Figura 3.35. Comunicación request/response entre nodos que realizan servicios [33].	49
Figura 3.36. Comunicación de mensajes de servicio [26].	49
Figura 3.37. Comunicación bidireccional entre nodos por medio de acciones [33].	50
Figura 3.38. Comunicación de mensajes de acción [26].	50
Figura 3.39. Tipos de datos básicos para mensajes en ROS [26].	51
Figura 3.40. Comunicación de mensajes entre nodos [26]	52
Figura 3.41. Nivel comunitario de ROS [34]	53

Figura 3.42. Visualización de la comunicación entre un nodo publisher (azul) y el nodo subscriber (verde) a travez del tema (rojo) number en RQT Graph.	54
Figura 3.43. Logo Proyecto Movelt	55
Figura 3.44. Logo Gazebo	56
Figura 3.45. Simulación de dron en Gazebo	56
Figura 3.46. Logo de RViz	57
Figura 3.47. ABB IRB 2400 con publicador de estado de juntas en RViz [35]	57
Figura 3.48. Tres visualizaciones diferentes del modelo Minibot URDF. Visualización en RViz, modelo de coliciones y estructura tf [36].	58
Figura 3.49. Ejemplo robot simple compuesto por base móvil y un láser	59
Figura 3.50. Láser recolectando datos respecto a "base_laser", árbol de transformación y datos del láser transformados a marco "base_link".	59
Figura 3.51. Frames de un sistema visualizado mediante tf_tree	60
Figura 3.52. Figura que contiene los marcos tf más utilizados [37]	60
Figura 3.53. Elementos basicos de visualization URDF: links y joints [38].	61
Figura 3.54. Representación de la visual, inercia y colisión de un link con sus respectivos marcos de referencia para URDF [38].	62
Figura 3.55. Descripción de un link en código URDF.	62
Figura 3.56. Representación gráfica de un joint y su relación padre-hijo con los link [38]	63
Figura 3.57. Descripción de un joint en código URDF	64
Figura 3.58. Ejemplo de la representación gráfica URDF de un robot por medio de la herramienta urdf_to_graphviz [38]	65
Figura 3.59. Iamgen de referencia del software Adams	67
Figura 3.60. Ventana de bienvenida al iniciar ADAMS	68
Figura 3.61. Ventana de bienvenida al iniciar ADAMS	69
Figura 3.62. Interfaz de vista en ADAMS	70
Figura 3.63. Jerarquía de datos en ADAMS	71
Figura 3.64. Jerarquía de nombres en ADAMS	71
Figura 3.65. Sistema de coordendas de una Pieza (PSC) [40]	72
Figura 3.66. Sistema de coordendas de un Marker[40]	73
Figura 3.67. Pieza con varias geometrias en ADAMS [40]	73
Figura 3.68. Unión revoluta, de una puerta con su marco [40]	74
Figura 3.69. Mediciones según tipo de objeto en ADAMS [40]	75
 Figura 4.1. Flujo de trabajo para el Metodo A y B	78
Figura 4.2. Sistema de referencia global XYZ y ángulos de los brazos $\theta_{i \in \{1,2,3\}}$	79
Figura 4.3. Sistema de referencia local para la cinemática de posición del método A [41].	80
Figura 4.4. Principales parámetros geométricos y puntos para la cinemática de posición del método A [41].	81
Figura 4.5. Sistema de ecuaciones para la cinemática directa del método A [41].	82
Figura 4.6. Sistema de ecuaciones para la cinemática inversa del método A [41].	84

Figura 4.7. Intersección entre círculos en punto J_1 y proyección del punto E_1 sobre el plano XY [41].	86
Figura 4.8. Rotación del sistema de referencia local en 120° para la solución de la cinemática inversa del método A [41].	86
Figura 4.9. Sistema de referencia local para la cinemática de velocidad del método A.	87
Figura 4.10. Traslación y rotación del sistema de referencia local $O - xyz$ en 120° para la solución de la cinemática de velocidad del metodo A.	88
Figura 4.11. Vectorización y ángulos interiores para la cinemática de velocidad del metodo A.	90
Figura 4.12. Representación de 3 tipos de singularidades de un robot delta [22]	93
Figura 4.13. Sistema de referencia local, parámetros geométricos y masas para la solución de la dinámica inversa del método A [22].	98
Figura 4.14. Sistema de referencia local para la cinemática del método B [44].	103
Figura 4.15. Principales parámetros geométricos y puntos para la solución de la cinemática del método B [44].	104
Figura 4.16. Vectorización para la solución de cinemática del método B [44].	105
Figura 4.17. Traslación de los vectores que representan los antebrazos para crear un sistema de ecuaciones y determinar la cinemática directa del método B [44].	106
Figura 4.18. Vista frontal de la base fija [44].	107
Figura 4.19. Solución de la cinemática directa de posición del método B [44].	109
Figura 4.20. Proyección del punto P_2 sobre el plano YZ [44].	110
Figura 4.21. Rotación del sistema de referencia local para la solución de la cinemática inversa del método B [44].	111
Figura 4.22. Puntos y distancias geométricas para la solución de la cinemática de velocidad del método B [44].	112
Figura 4.23. Representación gráfica de la rotación del sistema de coordenadas local $O-XYZ$ en los ángulos $\varphi_i \in \{i = 1, 2, 3\}$	113
Figura 4.24. Simplificación de masas para la solución de la dinámica inversa del método B.	118
Figura 4.25. Sistema de referencia local y parámetros geométricos para la solución de la dinámica inversa del método B.	119
Figura 4.26. Sistema de referencia local y masas para la solución de la dinámica inversa del método B.	120
Figura 4.27. Aproximacion del espacio de trabajo para un robot delta [60]	122
Figura 4.28. Espacio de trabajo YF003N [63]	123
Figura 4.29. 2 tipos de restricciones del espacio de trabajo	124
Figura 4.30. Descomposición de trayectoria en camino + ley temporal [64]	125
Figura 4.31. Procedimiento típico para generar trayectorias [64]	127
Figura 4.32. Interpolación articular no coordinada [64]	128
Figura 4.33. Interpolación en el espacio cartesiano [64]	128
Figura 4.34. Polinomios [64]	129

Figura 4.35. Trayectorias punto a punto : Interpolador de Velocidad Trapezoidal [64]	130
Figura 4.36.(Izquierda) Un robot 2R con límites de articulación $0^\circ \leq \theta_1 \leq 180^\circ$, $0^\circ \leq \theta_2 \leq 150^\circ$. (Centro superior) Una trayectoria en línea recta en el espacio articular y (arriba a la derecha) el movimiento correspondiente del efecto final en el espacio de la tarea (línea discontinua). Las configuraciones de punto final alcanzables, sujetas a límites de articulación, se indican en gris. (Centro inferior) Esta línea curva en el espacio de la articulación y (parte inferior derecha) la trayectoria de la línea recta correspondiente en la línea discontinua del espacio de la tarea) violaría los límites de la articulación. [65]	132
Figura 4.37. Gráficas de $s(t)$, $\dot{s}(t)$ y $\ddot{s}(t)$ para una escala de tiempo polinomial de tercer orden [65].	134
Figura 4.38. Gráficas de $s(t)$ y $\dot{s}(t)$ para un perfil de movimiento trapezoidal [65] .	135
Figura 4.39. Bang - bang [65]	135
Figura 5.1. Representación gráfica, marco de referencia, dimensiones, masas y orden de ángulos del robot delta a simular.	139
Figura 6.1. Herramientas principales para la solución de la dinámica y cinemática del robot delta.	140
Figura 6.2. Flujo de trabajo para la validación dinámica y cinemática del robot delta.	142
Figura 6.3. Ejemplo representación gráfica de funciones o nodos.	143
Figura 6.4. Conceptos programados en ROS.	144
Figura 6.5. Entradas y salidas del nodo principal	145
Figura 6.6. Función cinemática directa del método A	147
Figura 6.7. Función cinemática inversa del método A	147
Figura 6.8. Función jacobiano del método A	147
Figura 6.9. Función cinemática de aceleración del método A	147
Figura 6.10. Función dinámica inversa del método A	147
Figura 6.11. Función cinemática directa del método B	158
Figura 6.12. Función cinemática inversa del método B	158
Figura 6.13. Función jacobiano del método B	158
Figura 6.14. Función cinemática de aceleración del método B	158
Figura 6.15. Función dinámica inversa del método B	158
Figura 6.16. Diagrama de flujo para la solucion del espacio de trabajo	168
Figura 6.17. Diagrama de flujo de restricciones para el espacio de trabajo	169
Figura 6.18. Entradas y salidas de la función de espacio de trabajo	170
Figura 6.19. Entradas y salidas de la función de trayectoria	172
Figura 6.20. Flujo de trabajo para la creacion de trayectorias	172
Figura 6.21. Ejemplo gráfico del diagrama de flujo para la creación de trayectoria lineal con perfil trapezoidal.	172
Figura 6.22. Interpolador trapezoidal [66].	173

Figura 6.23. Traslacion marco de referencia $X_0Y_0Z_0$ hasta punto P_i	174
Figura 6.24. Rotación marco de referencia $X_{trans}Y_{trans}Z_{trans}$ sobre eje Z_{trans} en un angulo de θ_z	174
Figura 6.25. Rotación marco de referencia $X_{rot_{\theta_z}}Y_{rot_{\theta_z}}Z_{rot_{\theta_z}}$ sobre eje $Y_{rot_{\theta_z}}$ en un angulo de θ_y	175
Figura 6.26. Marco de referencia $X_{rot_{\theta_y}}Y_{rot_{\theta_y}}Z_{rot_{\theta_y}}$ en donde se aplica LSPB. . .	175
Figura 6.27. Diagrama de flujo de pasos para obtener la visualización del robot delta	178
Figura 6.28. Base fija en RViz	180
Figura 6.29. Brazo en RViz	181
Figura 6.30. Antebrazo en RViz	183
Figura 6.31. Base movil en RViz	186
Figura 6.32. Fases de desarrollo para prototipo virtual [40]	187
Figura 6.33. Simplificación de las masas en la base móvil	188
Figura 6.34. Posición del Marker Origen en ADAMS	190
Figura 6.35. Puntos de la base fija unidos mediante polilínea	192
Figura 6.36. Markers que definen la posición de cada actuador ubicados en el perímetro de la circunferencia.	193
Figura 6.37. Markers de cada motor con eje X perpendicular a cada lado de la base fija.	194
Figura 6.38. Masa m_1 de cada brazo en su posición.	195
Figura 6.39. Revolutas entre cada brazo y la base fija.	197
Figura 6.40. Posición de las masas m_2	198
Figura 6.41. Posición del Marker central de la Base Móvil, con su respectiva circunferencia de referencia.	199
Figura 6.42. Posición de las masas que configuran la Base Móvil	200
Figura 6.43. Vista superior del modelo	202
Figura 6.44. Vista frontal del modelo	202
Figura 6.45. Vista isometrica del modelo	203
Figura 6.46. Modelo robot delta, con trayectoria a seguir en la parte inferior . . .	205
Figura 6.47. Ventana para agregar las posiciones angulares en función del tiempo	206
Figura 6.48. <i>Motions</i> en cada unión de revoluta	207
Figura 6.49. Movimiento Robot Delta, inicio de la simulación	208
Figura 6.50. Movimiento Robot Delta, final de la simulación	208
Figura 6.51. Torque obtenido para el motor 1, en la curva numero 1	209
Figura 6.52. Torque obtenido para el motor 2, en la curva numero 1	210
Figura 6.53. Torque obtenido para el motor 2, en la curva numero 1	210
 Figura 7.1. Temas y nodos en ROS para la visualización del robot delta.	211
Figura 7.2. Visualización de links y joints del robot delta en RViz	213
Figura 7.3. Conexión entre los links y joints del robot delta en RViz	213
Figura 7.4. Representación gráfica de la relación padre-hijo del robot delta en RViz.	214

Figura 7.5. Temas y nodos en ROS para el calculo del espacio de trabajo del robot delta	215
Figura 7.6. Espacio de trabajo	216
Figura 7.7. Puntos alcanzables del robot delta sin restricciones de límites (figura (7.6))	216
Figura 7.8. Vista del plano XY de la figura (7.6) (en rojo) y (7.7) (en azul)	217
Figura 7.9. Vista del plano XZ de la figura (7.6) (en rojo) y (7.7) (en azul)	217
Figura 7.10. Puntos alcanzables figura (7.7) (azul) y puntos con restriccion $J_x \approx 0$ (verde)	218
Figura 7.11. Puntos alcanzables figura (7.7) (azul) y puntos con restriccción $J_\theta \approx 0$ (verde)	218
Figura 7.12. Resultado de la dinámica inversa en trayectoria 1	219
Figura 7.13. Resultado de la dinámica inversa en trayectoria 1	220
Figura 7.14. Resultado de la dinámica inversa en trayectoria 2	220
Figura 7.15. Resultado de la dinámica inversa en trayectoria 3	221
Figura 7.16. Resultado de la dinámica inversa en trayectoria 4	221
Figura 7.17. Resultado de la dinámica inversa en trayectoria 5	222
Figura 7.18. Resultado de la dinámica inversa en trayectoria 6	222
Figura 7.19. Resultado de la dinámica inversa en trayectoria 7	223
Figura 7.20. Resultado de torques de la trayectoria 8	223
Figura A.1. Vectorización cinemática directa método A	B.1
Figura A.2. Desplazamiento de vectores para la solución de la cinemática directa del método A	B.2
Figura A.3. Vista superior del efecto final	B.3
Figura A.4. Vectorización de la cinemática inversa del método A	B.9
Figura A.5. Visualización plano YZ para representar la intersección de los 2 círculos en el punto J_1 para la solución de la cinemática inversa en el método A	B.10
Figura A.6. Proyección del punto E_1 sobre el plano YZ	B.11
Figura A.7. Vista superior de la base fija	B.14
Figura A.8. Representación de la rotación del sistema coordenado XYZ en 240° para la solución de la cinemática inversa en el método A	B.15
Figura A.9. Vectorización y ángulos interiores para la solución de la cinemática de velocidad del método A	B.16
Figura A.10. Representación gráfica del centro de masas CoM	B.37
Figura B.1. Espacio de trabajo creado por catkin_make para los algoritmos de los métodos A y B	B.1

Capítulo 1

Introducción

1.1. Motivación

Desde el año 2010 las ventas de robots industriales se han acelerado considerablemente en nuestro país debido a la tendencia continua hacia la automatización y las sucesivas mejoras técnicas en este tipo de sistemas. El estudio de la IDC Worldwide Semiannual Robotics Spending Guide (??) indica que el mercado de la robótica en Latinoamérica presenta un crecimiento en 2019 del 73 % en robots industriales, del 27 % en robots de servicios y de un 0,09 % en robots de consumo. Los robots para producción de alimentos ganan terreno debido a la alta demanda de mano de obra para el trabajo dentro de esta actividad. Sin embargo, este nuevo interés por parte de la industria no se refleja en los estudios y objetivos de las formaciones académicas en las universidades, demostrando nuevamente una de las principales debilidades de nuestro país, la desconexión entre la industria y la academia. Esta desconexión se hace más latente aún con miras a nuestro departamento de mecánica en la USACH, una búsqueda sencilla de la palabra clave “robot” en los repositorios digitales de nuestra universidad, no arroja más de una treintena de resultados posibles, la mayoría de ellos ligados a otros departamentos de la universidad, principalmente eléctrica.

A pesar de esto y con la formación de ingenieros a nuestras espaldas, entendemos que una idea o solución a una problemática, es necesaria adaptarla a la realidad del entorno, en nuestro caso una robótica adaptada a las necesidades y capacidades del país. Es por esto que se elige entre las múltiples ramas de la robótica, los robots de tipo industrial adoptando la definición de la ISO como “Manipulador multifuncional reproducible con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas.” Nos enfocaremos en un robot manipulador de tipo delta, con la esperanza de que en un futuro próximo pueda implementarse y ser de ayuda en la industria agropecuaria de nuestro país, específicamente en sus procesos de packing y pick and place.

La intención de esta tesis es revertir este escenario en nuestro departamento, y renovar las iniciativas en el ámbito de la robótica en nuestros compañeros presentando una guía completa, de todos los ámbitos necesarios para comenzar a desarrollar un proyecto de robot delta, mostrando y resolviendo las problemáticas asociadas a la cinemática y dinámica del robot, la matemática implícita en estas soluciones y el código mediante el cual se llegó a dichas soluciones. Además, se realizarán simulaciones que comprueben los resultados teóricos. Todo esto bajo un ambiente de software libre y presentando muchos ramales para posibles proyectos en el área.

1.2. Hipótesis de estudio

Es posible realizar el modelamiento de la cinemática y dinámica de un robot delta a través de softwares libres y validarla por medio de softwares educativos.

1.3. Objetivos

1.3.1. Objetivo general

Crear el algoritmo que controla el movimiento de un robot delta y validarla a través de un software de simulación.

1.3.2. Objetivos específicos

1. Crear algoritmos que calculen trayectorias lineales en el espacio cartesiano con perfil de velocidad trapezoidal.
2. Crear algoritmos que resuelva la cinemática y dinámica.
3. Crear algoritmos que calculen el espacio de trabajo del robot.
4. Determinar el espacio de trabajo a partir de restricciones impuestas.
5. Simular el movimiento de las piezas mecánicas del robot a través de una herramienta de visualización.
6. Calcular la dinámica por medio de un software de análisis mecánico.
7. Comparar los resultados de la dinámica calculados por los algoritmos y por el software de análisis mecánico.

1.4. Alcance de la propuesta y limitaciones

El alcance de este proyecto por tanto diseñar un algoritmo para el control de su movimiento y verificar los resultados obtenidos a través de software de visualización y simulación dinámica. Es importante señalar que dentro de los alcances de esta tesis no se incluye la fabricación del robot, optimización de dimensionamiento de piezas y optimización de trayectoria.

1.5. Estructura de la tesis

Este documento se organiza en 8 capítulos y 2 anexos que, en rasgos generales, se describen a continuación:

- En el **capítulo (1)** se justifica lo importante que es la realización de este tema de tesis para la Universidad de Santiago, Chile y el mundo. Empieza con la motivación principal de la tesis, señalando argumentos relacionados con el ámbito profesional como ingenieros mecánicos. Posteriormente se presentan antecedentes generales sobre la robótica que validan la realización de esta tesis a través de datos estadísticos. Luego se presenta el problema a investigar con su respectiva descripción, hipótesis, objetivos generales y objetivos específicos. Finalmente se establecen los alcances y limitaciones tanto teóricas como económicas relacionadas con la modelación de un robot.
- En el **capítulo (2)** inicialmente se expone una breve introducción a la robótica explicando su origen y su significado. Posteriormente se presentan los hitos más importantes sobre la robótica en orden cronológico que impactaron en nuestra sociedad. Por otra parte, existen distintas clasificaciones de robot, por lo que se plantean solo las que son más recurrentes en el campo profesional a nivel mundial. Finalmente se destacan datos estadísticos y aplicaciones sobre la robótica en los últimos años.
- El **capítulo (3)** inicia con el funcionamiento general, una estructura básica y la descripción de las partes mecánicas del robot delta. Luego se explica detalladamente cada herramienta que se utiliza en la solución adoptada para la problemática de la modelación cinemática y dinámica del robot delta. Estas herramientas son tres: el software donde se crean los algoritmos para la control de la cinemática y dinámica; la interfaz de visualización de las piezas para la validación cinemática y software de simulación para la validación dinámica.
- El **capítulo (4)** explica matemáticamente los siguientes tópicos: modelación cinemática, modelación dinámica, espacio de trabajo y trayectoria de un robot delta. Se

presenta el sistema de referencia global en que están basados los resultados y los dos métodos utilizados para la solución de la modelación cinemática y dinámica.

- En el **capítulo (5)** se establecen los valores de las dimensiones y masas de las partes mecánicas del robot delta a simular.
- En el **capítulo (6)** se presenta el diagrama de flujo de trabajo con el que se desarrolla la tesis. La idea de este diagrama de flujo es que se vea claramente los pasos que se siguieron en este documento. Posteriormente se explican los algoritmos de cinemática y dinámica, los pasos para calcular el espacio de trabajo con sus respectivas restricciones y los algoritmos de las trayectorias implementados en el software ROS. Se expone el formato URDF de las partes mecánicas del robot delta y la configuración del paquete de visualización RViz. Finalmente se presenta la simulación dinámica de robot delta en el software ADAMS Student. Se da a conocer la configuración del software, la creación de piezas mecánicas, el tipo de junta entre las piezas, las simplificaciones físicas del problema a solucionar, la trayectoria a realizar y la configuración de los sensores de torque en los actuadores.
- En el **capítulo (7)** se dan a conocer los resultados obtenidos por las simulaciones y algoritmos propuestos en el capítulo (6) . Específicamente los resultados son: visualización de las partes mecánicas, espacio de trabajo y comparación de la modelación dinámica del robot delta.
- En el **capítulo (8)** se presenta las conclusiones de los resultados del capítulo (7) y las posibles proyecciones a futuro de esta tesis.
- En el **apéndice (A)** se desarrollan detalladamente la modelación física y matemática vista en el capítulo (4).
- En el **apéndice (B)** se presentan los códigos de los algoritmos escritos en el capítulo (6) para que sean utilizados a futuro fácilmente.

Capítulo 2

Estado del arte

2.1. Introducción a la robótica

La robótica es la ciencia y la tecnología que se ocupa del estudio y funcionamiento de los robots a través del diseño, manufactura y aplicación de estos. El objetivo de la robótica es diseñar un robot eficiente. Los robots están siendo cada vez más eficientes a causa de que los creadores e investigadores se enfocan en que estos puedan pensar y aprender por si solos, para ello implementan la inteligencia artificial. Un ejemplo de la IA es la robot humanoide Sophia, visualizada en la figura (2.1), capaz de interactuar con humanos .



Figura 2.1: Robot humanoide Sophia, IA [1]

La palabra robot fue usada por primera vez en el año 1921 en la obra de teatro R.U.R (Rossum's Universal Robots, figura (2.2)) creada por el escritor checo Karel Čapek (1890 - 1939). El origen etimológico de la palabra es robota, que significa en checo trabajo forzado o esclavo. Posteriormente se emplea la palabra “robótica” en obras de ciencia ficción tales como: Yo Robot (1950) y Robots e imperio (1985) creadas por el escritor y profesor de bioquímica Isaac Asimov (1920-1992).



Figura 2.2: Obra R.U.R (Robots Universal Rossum) creada por Karel Čapek [2]

Hoy en día los expertos en el área de la robótica y automatización no han llegado a un acuerdo para una definición universal de la palabra robot. Es por esta razón que a continuación se presenta la descripción de un robot respecto al punto de vista de tres instituciones importantes:

- **Robot Institute of America (RIA):** “Un robot es un manipulador reprogramable multifuncional diseñado para mover material, partes, herramientas o dispositivos especializados a través de movimientos programados variables para el desarrollo de una variedad de tareas.”
- **Japanese Industrial Robot Association (JIRA):** “Un robot de un dispositivo con grados de libertad que puede ser controlado.”
- **International Federation of Robotics (IFR):** “Un robot es un mecanismo actuado programable en dos o más ejes con un grado de autonomía, que se mueva en su entorno para realizar tareas previstas. *Nota 1: Un robot incluye el sistema de control y la interfase con el sistema de control. Nota 2: La clasificación de robot industrial o robot de servicio se hace de acuerdo con la aplicación prevista.*”

A partir de las tres definiciones anteriores podemos concluir que un robot es un mecanismo programable o re-programable, capaz de interactuar con acciones independientes e inteligentes en un entorno específico para realizar una variedad de tareas previstas.

En este trabajo de título se estudia un robot de tipo delta. Esta compuesto por tres brazos conectados a una base fija y a otra móvil llamada efector. Al estar los tres brazos conectados a la misma base, la cinemática del robot es cerrada. Los brazos se accionan por medio de actuadores que generalmente son motores. En el efector final se encuentra generalmente herramientas para realizar tareas específicas.

2.2. Historia de la robótica

Los robots son una gran noticia hoy en día gracias a las enormes mejoras que han provisto en diversas áreas de la vida de las personas y han abierto un nuevo capítulo en la interacción de humanos y robots para el futuro. Estas enormes mejoras han sido paulatinas, ya que la robótica tiene sus orígenes hace miles de años. A continuación, se presentan algunos de los hitos registrados a través de la historia, que han ayudado a la robótica a convertirse en lo que es hoy.

Año	Hito
400 A.C	El matemático y filósofo italiano Arquitas de Tarento hizo una paloma de madera que volaba.
10-70 D.C	El matemático y científico griego Herón de Alejandría construyó diversos autómatas con forma de ave. Se le atribuye la invención de la primera máquina de vapor, conocida como “aeolipile” y la fuente de Herón.
1452 - 1519	Leonardo Da Vinci diseño 2 autómatas, el primero consiste en un mecanismo que emulaba el movimiento humano vestido de armadura y el segundo un león mecánico
1947	Primer manipulador eléctrico servo-controlado, por Goetz.
1952	Primera máquina de control numérico, que se programa por un lenguaje simbólico Software.
1954	El primer Robot: manipulador tipo brazo articulado que realizaba una secuencia de movimientos programables, desarrollado por George Devol.
1959	George Devol conoció a Joseph Engelberger y juntos fundaron en 1960 la empresa Unimation dedicada a la fabricación de robot
1960	Se produce el primer robot de configuración cilíndrica Versatran, por la compañía American Machine Foundry (AMF)
1961	Unimation instala el primer Unimate en General Motors en los procesos de fundición; mientras que la Ford Motor Company instala un robot Versatran.
1963	La compañía Fuji Yusoki Kogyo de Japón desarrolla el primer robot para aplicaciones de palletizing, llamado Palletizer.
1968	Kawasaki adquiere los derechos de fabricación del Unimate en Japón. Comienza la fabricación e implementación de robots en las industrias de Japón.

Continuación de la tabla (2.1)

	General Motors emplea baterías de robots en el proceso de fabricación de las carrocerías de los coches.
1970	KUKA, empresa alemana, instala la primera línea de soldadura equipada con robots industriales.
1971	Se funda la Japanese Industrial Robot Association (JIRA).
1973	ASEA, empresa sueca, comercializa el primer robot industrial completamente eléctrico, IRB6.
	La empresa KUKA Robotics construye el primer robot articulado electromecánicamente de 6 ejes nombrado FAMULUS.
1974	<p>Se funda el Robot Institute of America (RIA), actualmente llamado Robotic Industries Association.</p> <p>Se introduce el primer robot industrial a España.</p> <p>Comienza en lenguaje de programación AL del que derivan otros robots posteriormente.</p>
1978	Unimation, con el desarrollo de Victor Scheinman, introduce el robot PUMA (Programmable Universal Machine for Assembly) con el lenguaje de programación VAL (Victor's Assembly Language).
1979	A partir del desarrollo del profesor Hiroshi Makino de la Universidad de Yamanashi de Japón se produce el primer robot SCARA (Selective Compliance Assembly Robot Arm) a manos de Sankyo e IBM.
1980	Fundación de la Federación Internacional de Robótica (IFR).
1981	La compañía americana PaR Systems introduce el primer robot Gantry o de plataforma industrial.
1984	<p>Adept introduce el primer robot SCARA de accionamiento directo llamado AdeptOne.</p> <p>ABB, empresa sueca formada a partir de ASEA, produce el IRB 1000, el robot ensamblador más rápido hasta la fecha.</p>
1987	Se constituye la Federación Internacional de Robótica con sede en Estocolmo.
1992	Aparece el robot DELTA, diseñado por el científico suizo Reymond Clavel en su tesis de doctorado.
1994	Motoman presenta el primer sistema de control de robots (MRC) que proporcionó el control sincronizado de dos robots hasta 21 ejes.
1998	ABB, basándose en la estructura del robot Delta, desarrolla el Flex-Picker, considerado el robot de selección más rápido del mundo.
1999	La compañía alemana Reis Robotics, integra una guía de rayo láser integrada dentro de sus brazos robóticos.

Continuación de la tabla (2.1)	
2004	Motoman introduce un sistema de control robótico mejorado (NX100), que provee control sincronizado de cuatro robots hasta 38 ejes.
2006	La compañía de automatización Comau de Italia presenta el primer Teach Pendant Inalámbrico (WiTP).
	KUKA presenta el primer robot de peso ligero (LWR) conformado por una estructura de aluminio.
2007	Motoman lanza robots super rápidos de soldadura por arco, que reduce los tiempos de ciclo en un 15 %
2009	ABB lanza el robot industrial multipropósito más pequeño, IRB120, que pesa solo 25kg y un alcance de 580mm.
2013	KUKA presenta un robot diseñado para una colaboración segura humano-robot del área de trabajo llamado LBR iiwa “Leichtbauroboter intelligent industrial work assistant”, que cuenta con 7 ejes.

Tabla 2.1: Evolución de la robótica industrial

El robot delta fue investigado e inventado en 1985 por el profesor Reymond Clavel (figura (2.3)) y su equipo en el laboratorio de sistemas de robótica de la École Polytechnique Fédérale de Lausanne (EPFL, Suiza). Ellos comenzaron la investigación del robot delta después de una visita a una fábrica de chocolate. El equipo de Clavel estaba buscando aplicaciones de mano de obra repetitivas para robots, y descubrieron que el empaque de bombones de chocolate era un candidato para este tipo de automatización de alta velocidad y baja carga útil. En ese mismo año se completó el prototipo de robot delta, el cual fue patentado. En 1987 la compañía suiza Demareux Robotics and Microtechnology compró una licencia del robot delta y comenzó la producción estos para la industria de empaquetamiento. En 1991 el Dr. Reymond Clavel presentó su tesis doctoral 'Conception d'un robot parallèle rapide à 4 degrés de liberté' y recibió el premio Golden Robot Award patrocinado por ABB Flexible Automation, por su trabajo innovador y desarrollo del robot delta.

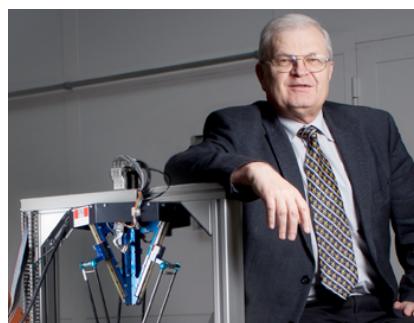


Figura 2.3: Reymond Clavel y el robot delta [3]

2.3. Clasificaciones de robots

2.3.1. Clasificación por generación

2.3.1.1. Primera generación

Robots manipuladores. Se caracterizan por programas de control fijos, relativamente sencillos de lazo abierto (sin retroalimentación). Son capaces de repetir operaciones previamente programadas en ellos, no pueden adaptarse al entorno, por lo que no deben existir perturbaciones externas. Por lo tanto, estos robots son los más adecuados en entornos industriales que realizan operaciones repetitivas.

2.3.1.2. Segunda generación

Robots de aprendizaje. Son adaptativos, pueden operar en entornos variables o parcialmente desconocidos. Estos robots ejecutan una serie de operaciones predefinidas, pero también pueden tener en cuenta los cambios en el entorno y modificar su rutina para realizar sus tareas. Imitan la secuencia de movimientos que ha sido ejecutada previamente por un operador humano. El operador realiza los movimientos requeridos mientras el robot le sigue y los memoriza con ayuda de sensores que recopilan datos.

2.3.1.3. Tercera generación

Robots con control sensorizado. Los robots cuentan con controladores (computadoras) que, usando los datos o la información recopilada de sus sensores, obtienen la habilidad de ejecutar las órdenes de un programa escrito en alguno de los lenguajes de programación que surgen a raíz de la necesidad de introducir las instrucciones deseadas en dichas máquinas.

2.3.1.4. Cuarta generación

Robots inteligentes. Esta etapa se caracteriza por tener sensores mucho más perfeccionados que mandan información al controlador y la analizan mediante estrategias complejas de control. Incorpora inteligencia artificial totalmente. Esto permite una toma inteligente de decisiones y el control del proceso en tiempo real.

2.3.2. Clasificación por arquitectura o estructura

2.3.2.1. Poli-articulados

La característica principal de este tipo de robots es la de ser sedentarios, aunque algunos tienen desplazamientos limitados. Tienen un espacio de trabajo limitado según uno o más sistemas de coordenadas y tienen un número limitado de grados de libertad. En la figura (2.4) se muestra un ejemplo de estos robot, el cual esta encargado de labores de soldadura.

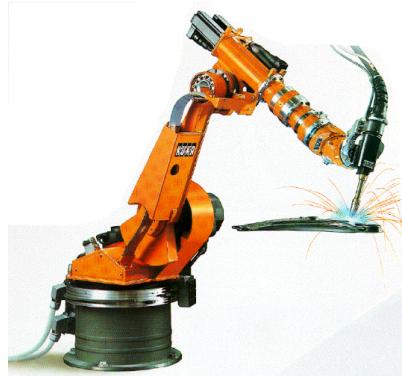


Figura 2.4: Robot Poli-articulado [4]

2.3.2.2. Zoomórficos

La característica principal de este tipo de robot es que imitan los movimientos, desplazamientos y funciones de diversos tipos de seres vivos. Existen dos categorías de robots zoomórficos: no caminadores y caminadores. En la actualidad se encuentran en mayor cantidad los robots caminadores. Un ejemplo de estos ultimos es el robot que se aprecia en la figura (2.5), llamado 'Spot classic de Boston Dynamics'. Las aplicaciones de estos robots son esencialmente en áreas peligrosas tales como la exploración espacial, estudio de volcanes, fines militares e industrias nucleares.



Figura 2.5: Robot zoomórfico: Spot Classic (2015) de Boston Dynamics [5]

2.3.2.3. Móviles

Estos robots tienen una capacidad de desplazamiento amplia, basados en carros o plataformas y equipado de un sistema locomotor comúnmente de ruedas. Se desplazan por una trayectoria o camino por medio de un mando a distancia o guiándose a través de la información capturada por los sensores. Un ejemplo es el iRobot 510 PackBot mostrado en la figura (2.6), que se utiliza para realizar una variedad de misiones, incluida la eliminación de artefactos explosivos, manipulación de materiales peligrosos, etc.



Figura 2.6: Robot móvil: iRobot 510 PackBot.[\[6\]](#)

2.3.2.4. Androides

La particularidad de este tipo de robots es que son antropomorfos, en otras palabras, que tiene forma o apariencia humana y además imitan algunos aspectos de su conducta de manera autónoma. La mayor dificultad que tienen hoy en día los especialistas de este tipo de robots es simular el comportamiento cinemático del ser humano, llamado locomoción bípeda. La locomoción bípeda es la habilidad de los seres vivos de caminar sobre sus dos extremidades inferiores. Boston Dynamics tiene varios robots androides, entre los cuales se encuentra el robot Atlas visualizado en la figura.(2.7).



Figura 2.7: Robot androide: Atlas de Boston Dynamics [\[7\]](#)

2.3.3. Clasificación por su movimiento

2.3.3.1. Robots articulados

Los robots articulados son unos de los tipos más comentados de robots industriales. Se asemejan a un brazo humano en su configuración mecánica. El brazo está conectado a la base con una articulación giratoria. Las articulaciones pueden ser paralelas u ortogonales entre sí. Los robots articulados que tienen seis grados de libertad son los robots industriales más utilizados, ya que el diseño ofrece la máxima flexibilidad.



Figura 2.8: Robot Articulado [8]

2.3.3.2. Robots cartesianos

Los robots cartesianos también se denominan robots rectilíneos y tienen una configuración rectangular. Estos tipos de robots industriales tienen tres articulaciones prismáticas para proporcionar movimiento lineal al deslizarse sobre sus tres ejes perpendiculares (X, Y, Z).



Figura 2.9: Robot Cartesiano [8]

2.3.3.3. Robots SCARA

Los robots SCARA son los robots que pueden hacer 3 traslaciones más una rotación alrededor de un eje vertical. Los ejes rotativos se colocan verticalmente, y el efecto final unido al brazo se mueve vertical. Por la configuración de los brazos de SCARA, son flexibles en los ejes XY y rígidos en el eje Z. Los robots SCARA se especializan en movimientos laterales y se utilizan principalmente para aplicaciones de ensamblaje.

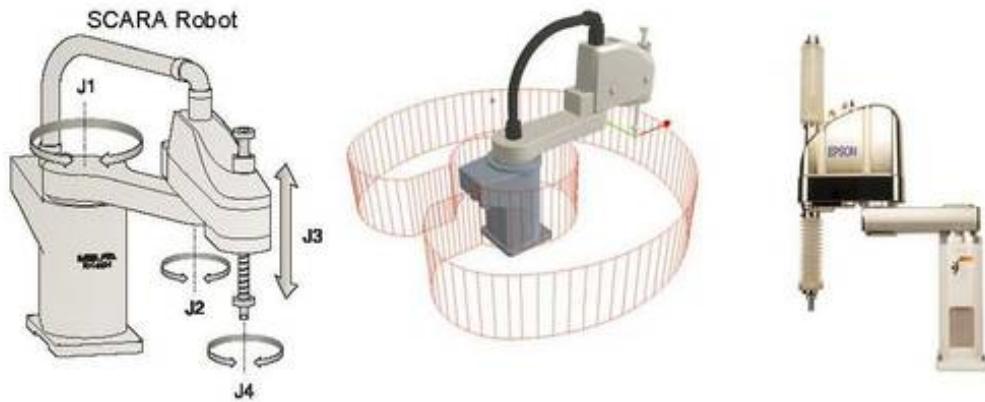


Figura 2.10: Robot SCARA [8]

2.3.3.4. Robots delta

Los robots delta también se les llama robots paralelos, ya que consiste en enlaces de unión paralelos conectados con una base fija común. Debido al control directo de cada junta sobre el efecto final, el posicionamiento de este último se puede controlar fácilmente con sus brazos, lo que resulta un robot de operación de alta velocidad. El espacio de trabajo de los robots delta tienen forma de cúpula. Estos robots se utilizan generalmente para aplicaciones de pick and place.



Figura 2.11: Robot delta [9][10][11]

2.3.3.5. Robots cilíndricos

Los robots cilíndricos tienen al menos una junta giratoria en la base y al menos una junta prismática que conecta los enlaces. Estos robots tienen un espacio de trabajo cilíndrico con un eje pivotante y un brazo extensible que se mueve verticalmente y deslizándose. Por lo tanto, los robots con configuración cilíndrica ofrecen un movimiento lineal vertical y horizontal junto con un movimiento giratorio sobre el eje vertical.

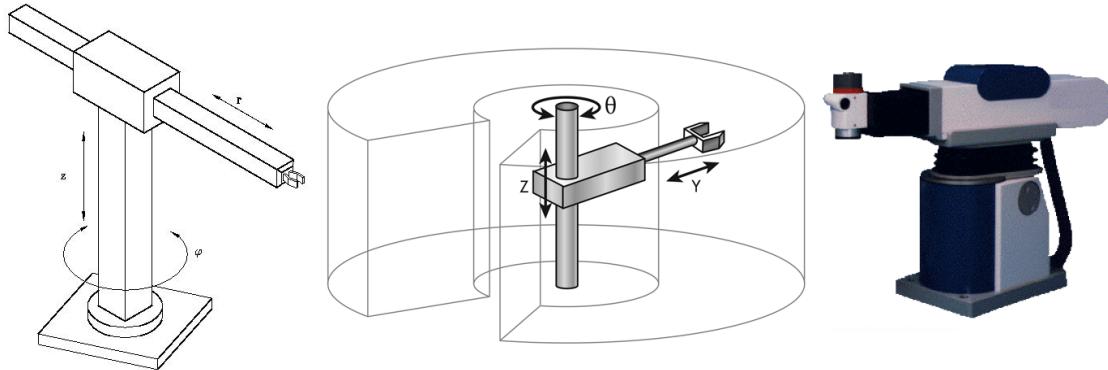


Figura 2.12: Robot cilíndrico [12][13][14]

2.3.3.6. Robots polares

También llamados robots esféricos. En esta configuración el brazo está conectado a la base por una junta giratoria , una combinación de dos juntas rotativas y una junta lineal. Los ejes forman un sistema de coordenadas polares y crean una envoltura de trabajo de forma esférica.

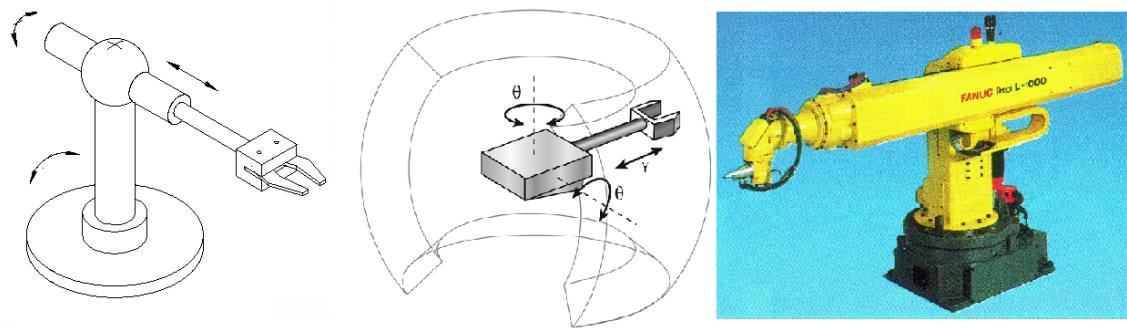


Figura 2.13: Robot polar [12][13]

2.4. Estadísticas y aplicaciones

Muchos tipos de actividades en los sectores industriales tienen el potencial técnico para ser automatizados. Según el reporte “El potencial técnico para la automatización en los EE. UU” de McKinsey Company en enero del 2017, el 50 % de los trabajos realizados por humanos hoy son vulnerables al reemplazo por robots. Eso podría equivaler a una pérdida de \$15 billones dólares en salarios en todo el mundo y \$2.7 billones de dólares en los EE. UU. Esto puede ocurrir por completo entre los años 2035 -2055 aproximadamente.

En la práctica, la implementación de la automatización en nuestro mundo dependerá de más que solo la viabilidad técnica. Existen cinco factores:

1. Viabilidad técnica.
2. Costos para automatizar.
3. La relativa escasez, las habilidades y el costo de los trabajadores de que otro modo podrían realizar la actividad.
4. Beneficios de la automatización más allá de la sustitución del costo laboral.
5. Consideraciones regulatorias y de aceptación social.



Figura 2.14: Logo de McKinsey Company [15]

Según la figura (2.15), las actividades más automatizables son las que gran parte del tiempo implican realizar actividades físicas u operar maquinaria en un entorno predecible. Los trabajadores llevan a cabo acciones específicas en entornos conocidos donde los cambios son relativamente fáciles de anticipar. Dado que las actividades físicas predecibles ocupan un lugar destacado en sectores como la fabricación, el servicio de alimentos y el alojamiento y la venta minorista, estas son las más susceptibles a la automatización basadas solo en consideraciones técnicas.

1. **El sector de servicios:** El alojamiento y servicio de alimentos, donde casi la mitad de todo el tiempo de trabajo implica actividades físicas predecibles y la operación de maquinaria, incluida la preparación, la cocina o el servicio de alimentos; limpieza de áreas de preparación de alimentos; preparar bebidas frías y calientes; y la recolecta platos sucios.
2. **En la manufactura o producción:** Las actividades van desde el envasado de productos hasta la carga de materiales en equipos de producción, desde soldadura hasta mantenimiento de equipos.

3. El comercio minorista: Los minoristas pueden aprovechar, por ejemplo, la gestión de existencias y la logística eficiente, impulsada por la tecnología. Los objetos de embalaje para el envío y almacenamiento de mercancías se encuentran entre las actividades físicas más frecuentes en el comercio minorista y tienen un alto potencial técnico para la automatización.

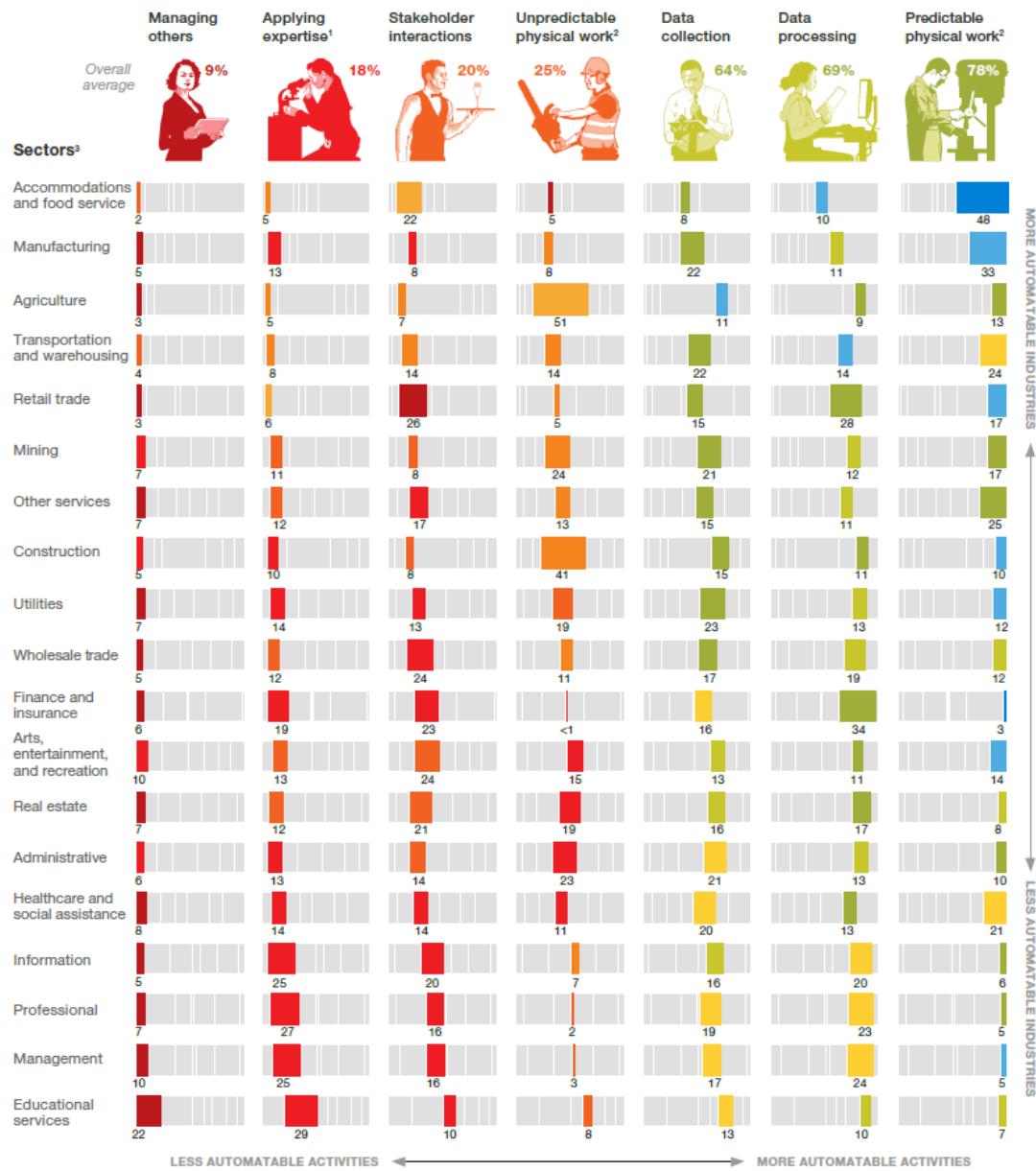


Figura 2.15: El potencial técnico para la automatización en los EE. UU. (McKinsey's Company 2017) [16]

Carl Benedikt Frey y Michael A. Osborne en el año 2013 publicaron su paper “El futuro del empleo: ¿Cuán susceptibles son los trabajos para la computarización?” [17]. El efecto que causó en la población que leyó esta publicación, fue de dudas e incertidumbres, sin embargo, el reporte de la empresa McKinsey Company del 2017 mostrado anteriormente refuerza el trabajo realizado por el economista y por el profesor de machine learning. En la figura (2.16) se visualiza la probabilidad de computarización del trabajo por categorías. Se aprecia que el 47 % de los trabajos corren un alto riesgo de ser computarizados (probabilidad mayor a 70 %).

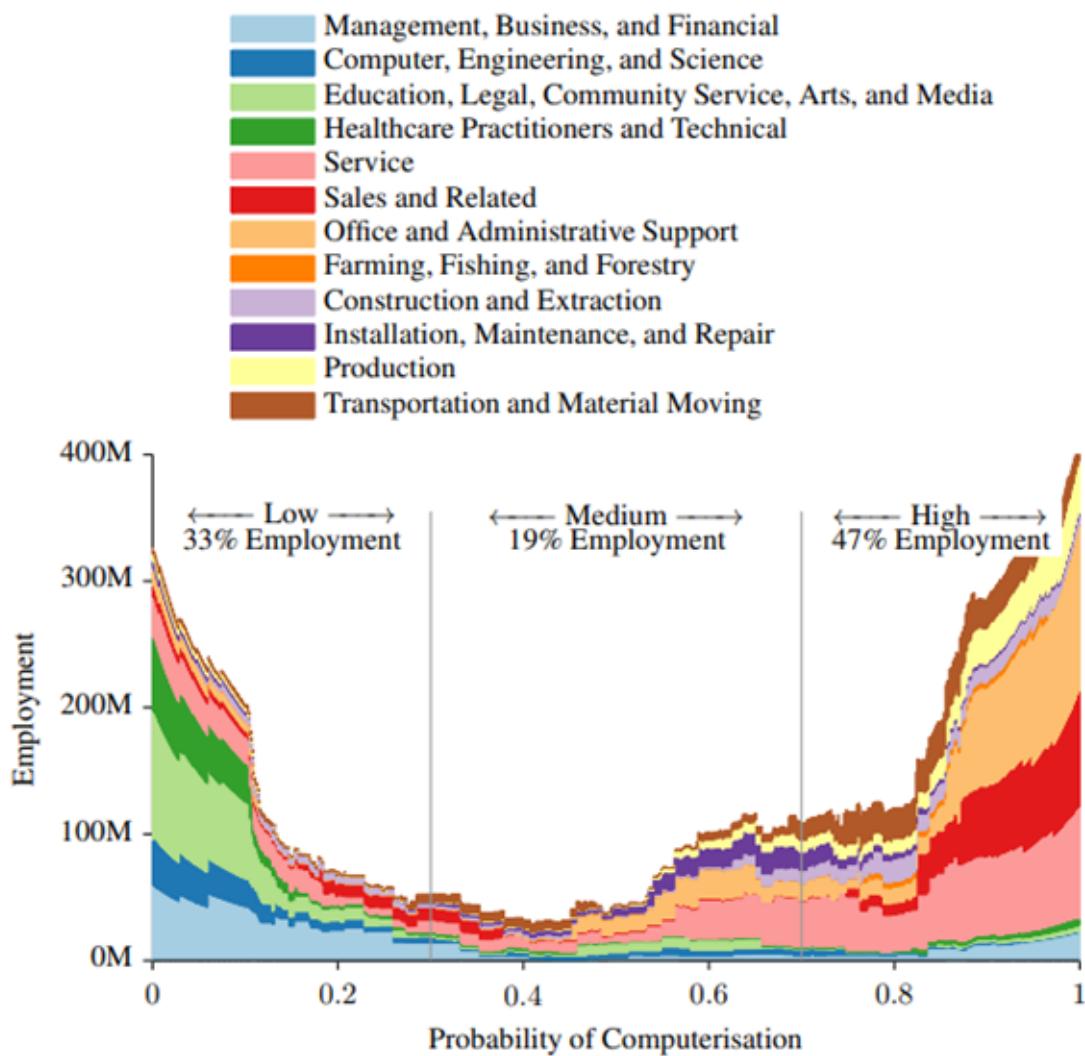


Figura 2.16: La distribución del empleo ocupacional de BLS (Oficina de estadísticas laborales) 2010 sobre la probabilidad de computarización, junto con la participación en las categorías de probabilidad baja, media y alta. El área total bajo todas las curvas es igual al empleo total de los EE.UU [17].

Acerca del tamaño y crecimiento del mercado de robots industriales, se puede decir que pocos han mostrado un crecimiento tan fuerte y sostenido en la última década. La figura (2.17) reafirma este crecimiento. En los últimos años, las instalaciones anuales aumentaron en un 19 % CAGR, lo que resultó en 422,271 nuevas instalaciones globales en 2018, por un valor de USD 16,5 mil millones. Esto fue solo para el hardware robótico, mientras que el software / servicios y el hardware periférico ascendieron aproximadamente al mismo valor cada uno. A finales de 2018, el stock operativo global total de robots fue de 2.439.543. Asia es el motor de crecimiento detrás de estos números (dos de cada tres nuevas instalaciones robóticas ocurren en Asia), con China a la cabeza (aumentando el stock operativo de robots en casi un 30 % interanuales los años pasados). Europa y América se están quedando un poco atrás.

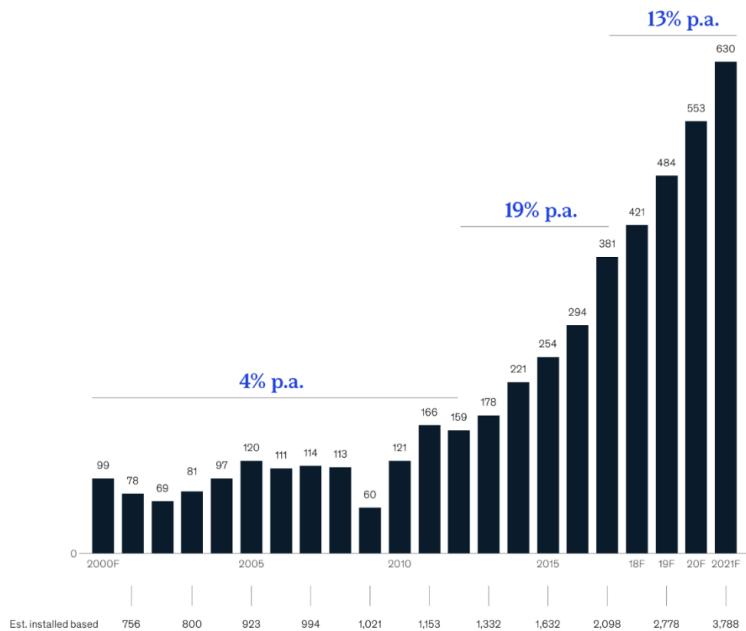


Figura 2.17: Envíos anuales (en miles) de robots industriales en todo el mundo [18].

Además del rápido crecimiento, lo más emocionante de la robótica es el impacto fundamental que tendrá en nuestras vidas futuras, librándonos potencialmente de la mayor parte del trabajo manual. A pesar de una percepción comúnmente diferente, incluso en industrias “altamente automatizadas” como la automotriz, el grado promedio de automatización (número de tareas automatizadas / número de tareas totales) en el ensamblaje final es solo alrededor del 8-10 %. Y la mayoría de las otras industrias están muy por debajo de eso. Por otro lado, generalmente se estima que hasta el 85 % de todas las tareas son ‘automatizables’. Escalar esta curva de automatización ofrece una gran oportunidad para aumentar la productividad y el bienestar global, si nosotros, como sociedad, podemos compartir sus beneficios por igual.

Los robots siguen siendo esencialmente demasiado caros y demasiado 'estúpidos', lo que permite muy pocas áreas de aplicación rentables. Pero nos enfrentamos a una tormenta perfecta de inventos innovadores que pronto desbloquearán un punto de inflexión, donde la robótica será significativamente más barata y más versátil que la mayoría del trabajo manual.

A pesar de no incluir datos de los últimos años, la figura (2.18) muestra las solicitudes anuales de patentes de la UE y los EE. UU. para tecnologías de fabricación avanzadas (incluida la robótica) solo han comenzado a despegar en la última década. A medida que estos esfuerzos exponenciales de I + D alcancen la madurez comercial, se desbloqueará una ola sin precedentes de nuevos (y rentables) casos de uso de robótica.

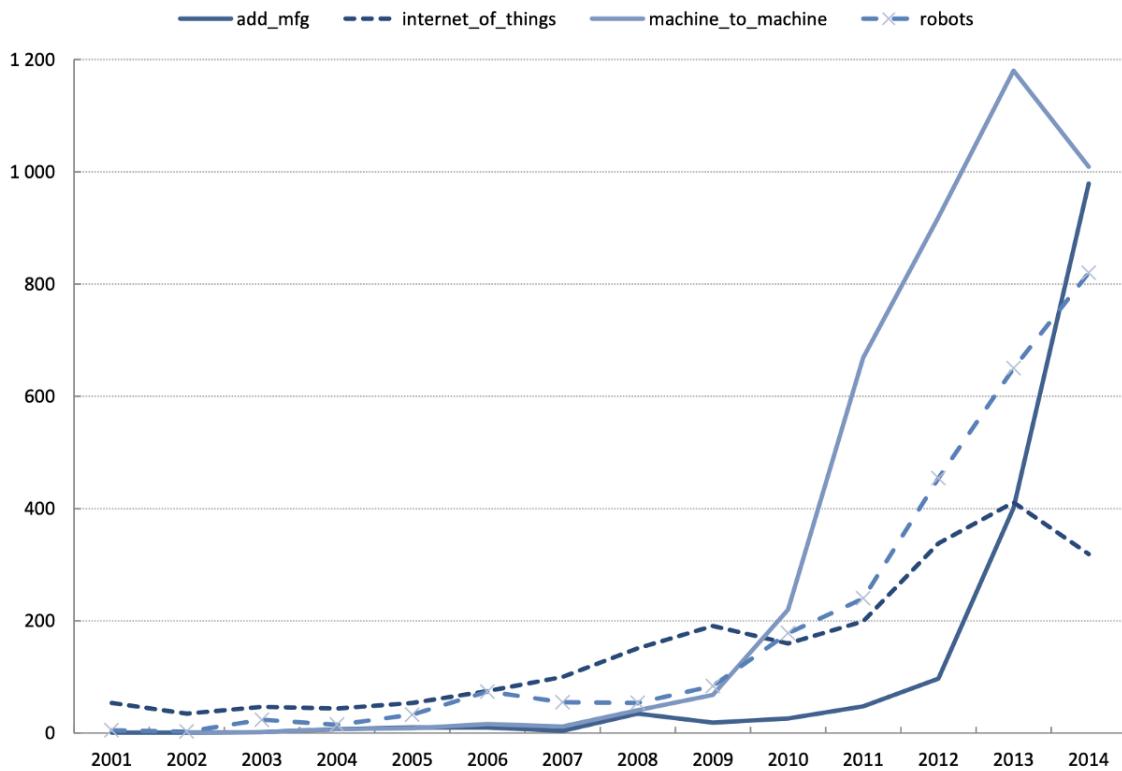


Figura 2.18: Solicitudes anuales de patentes para tecnologías de fabricación avanzadas específicas [19]

Las empresas que están innovando en el área de robótica buscan solucionar problemáticas para que sea más accesible a la mayor cantidad de personas, para este propósito desarrollan nuevas tecnologías para minorar los costos o hacer más flexible la robótica. Se pueden dividir en categorías las principales áreas de innovación en robótica, tales como se presentan en la figura (2.19):

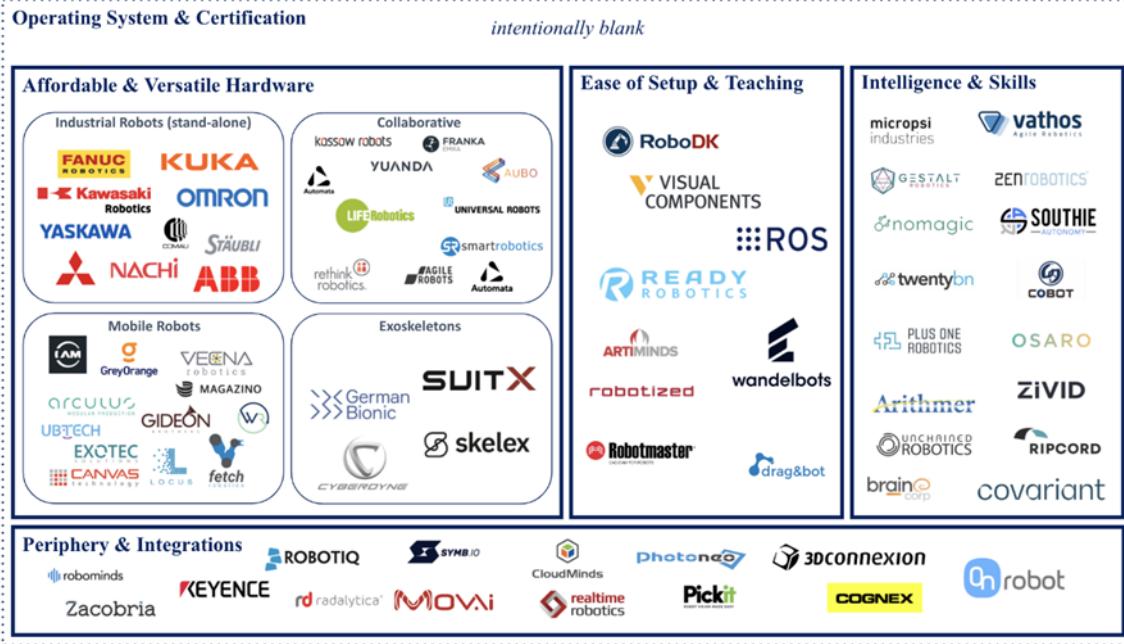


Figura 2.19: Mapa del mercado de robótica industrial [20]

- **Hardware asequible y versátil:** en la última década, los costos promedio de una instalación robótica se han reducido aproximadamente en un 40 %, mientras que su productividad ha aumentado aproximadamente un 5 % por año. Se necesitarán nuevos modelos de negocio para ayudar a fomentar la adopción del mercado (por ejemplo, “robot como servicio” o modelos de alquiler). Al mismo tiempo, las capacidades mecánicas y las propiedades físicas deben mejorar (menos peso, mayor alcance, mayor velocidad, mejor modularidad, mejores sistemas de seguridad, etc.).
- **Facilidad de configuración y enseñanza:** hoy en día, alrededor del 70 % de los costos totales de vida útil de una celda robótica son generados por servicios relacionados con la configuración, la programación y la enseñanza. Por lo general, esto lo realizan una gran cantidad de integradores de sistemas de tamaño pequeño a mediano, y demoran semanas o meses en configurar, programar y certificar una nueva aplicación de robótica. Por lo tanto, aquí no solo enfrentamos un problema de costos y tiempo, sino también un desafío de inflexibilidad.
- **Inteligencia y habilidades:** después de codificar dolorosamente una aplicación robótica, sigue siendo en gran medida rígida e inflexible. Si, por ejemplo, la forma o la posición de las piezas que va a recoger un robot cambian con el tiempo, en la mayoría de los casos esto significaría “fin del juego” para un robot industrial.

- **Periferia e integraciones:** Una celda robótica siempre comprende elementos de hardware adicionales como pinzas y herramientas, sistemas de transporte, sistemas de seguridad, cámaras 2D y 3D, etc. La mayoría de estos sistemas periféricos provienen de diferentes proveedores, no tienen interfaces estándar y no se comunican fácilmente entre ellos. La creación de estándares y la modularización es clave para permitir una experiencia plug & play “consumida” en diferentes proveedores de sistemas.
- **Sistema operativo y certificación:** Hoy en día, no existe un “MS Windows” para la robótica industrial, que permitiría que todas las partes individuales que arman un robot se comuniquen entre sí. A pesar de algunos esfuerzos de código abierto, como ROS2 (Robotics Operating System 2), todavía no se ha encontrado ningún estándar de software real de la industria.

Capítulo 3

Arquitectura de un robot delta

El tema que trata este capítulo es acerca de la descripción del robot delta y las herramientas que se utilizan para la solución de la problemática de la modelación cinemática y dinámica. Inicialmente se explica de forma general el funcionamiento y la estructura básica de un robot delta, con el fin de que el lector de esta tesis entienda lo complejo que es crear un robot por la gran variedad de áreas que lo componen. Luego se muestra gráficamente y se describen las partes mecánicas o piezas físicas con que está estructurado el robot. Posteriormente se presenta el funcionamiento y los componentes del middleware ROS, que es el encargado de controlar y ordenar la lógica de la modelación cinemática y dinámica. Enseguida se enseña la herramienta de ROS llamada Rviz, que es la encargada de la visualización del robot. Finalmente se muestra el software que comprueba la modelación dinámica del sistema, llamado ADAMS.

3.1. Funcionamiento general

La principal tarea de un robot es ir de un punto a otro para realizar una determinada acción. Para realizar dicha acción se debe pasar por una serie de pasos complejos con el fin de asegurar una ejecución exitosa. Con la intención de explicar brevemente los pasos a seguir para lograr dicho objetivo, en la figura (3.1) se muestra un ejemplo básico de un robot delta accionado por motores paso a paso por medio de un diagrama de flujo.

Los pasos del diagrama de flujo son los siguientes:

1. Definir los puntos de inicio y final de la trayectoria del robot.
2. Elegir el tipo de trayectoria: posiciones, velocidades y aceleraciones impuestas.
3. Comprobar la cinemática para obtener la trayectoria angular de los motores .

4. Transformación de la trayectoria cartesiana a la trayectoria en el espacio articular de los actuadores.
5. Comprobar dinámica para asegurar que la trayectoria impuesta no dañe los componentes del robot y los motores.
6. Traducción de trayectoria en el espacio articular a pulsaciones que entiendan los drivers de cada motor.
7. Envío de pulsos al driver de los motores por medio de un controlador.
8. Envío de pulsos del driver hacia los motores paso a paso.

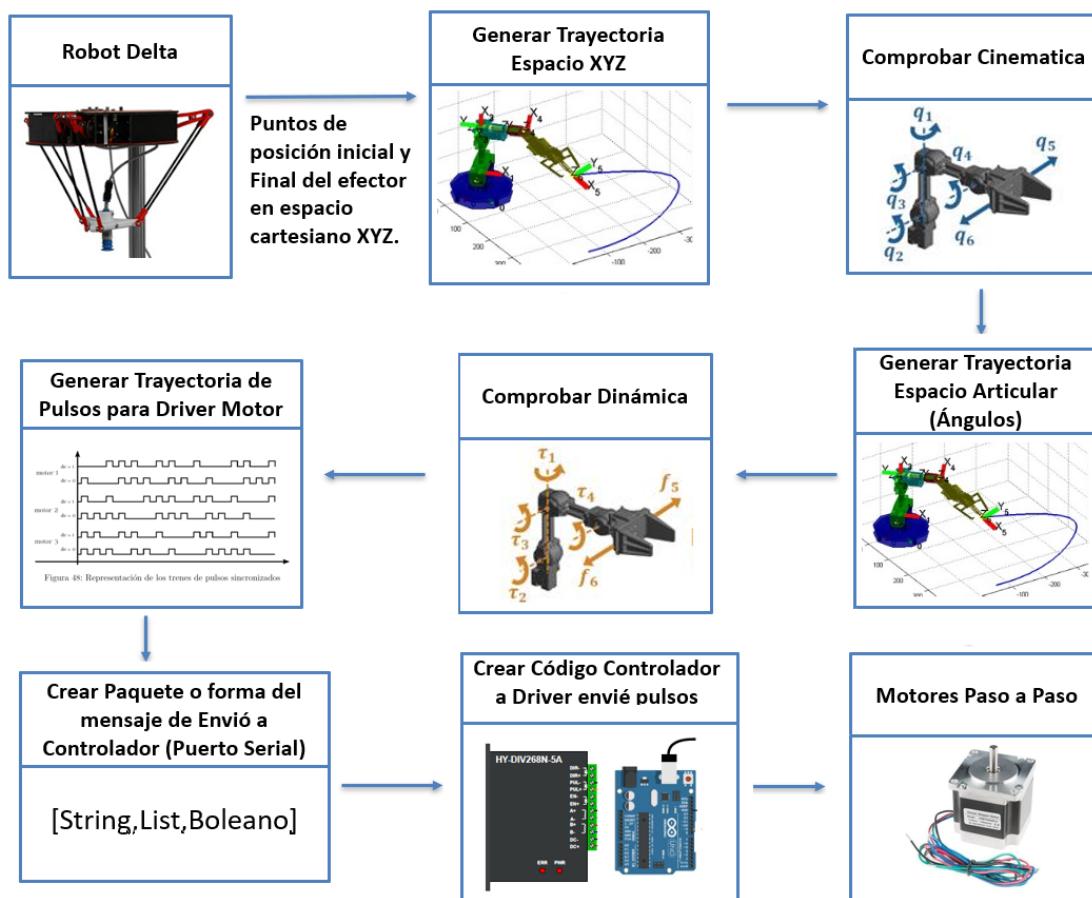


Figura 3.1: Ejemplo de diagrama de flujo de tareas que realiza un robot delta para realizar una trayectoria específica

3.2. Estructura de un robot delta

El robot delta puede ser subdividido por categorías de acuerdo al grupo estructural al que pertenezcan. Estas categorías facilitan la generación de conceptos para cada grupo de manera independiente, como se ilustra en la figura (3.2).

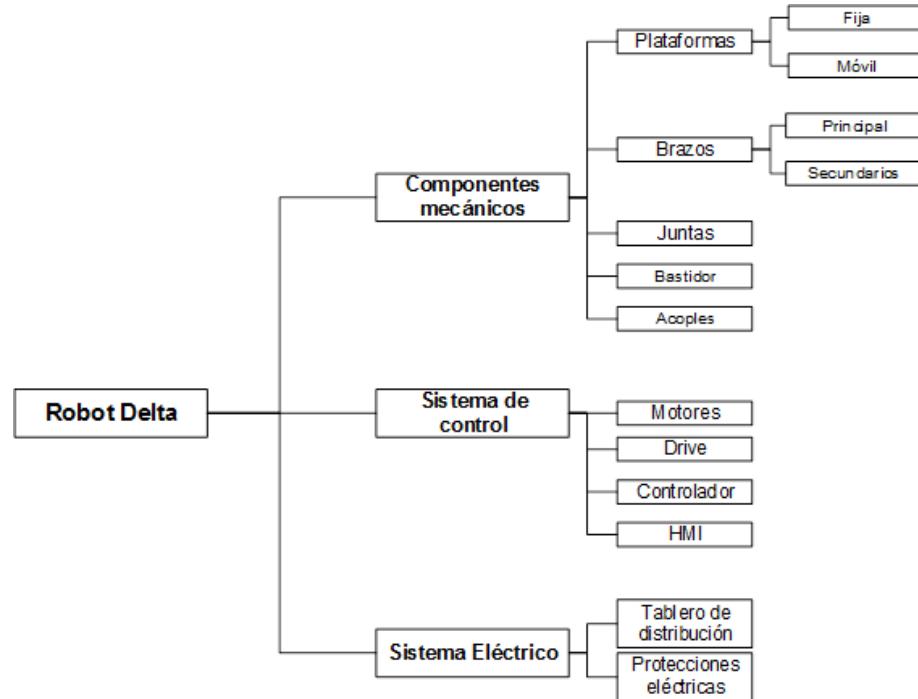


Figura 3.2: Descomposición estructural del robot delta [21]

1. **Los componentes mecánicos:** son todas las piezas físicas que componen el robot delta. Es muy importante la elección del tipo de juntas a elegir ya que restringen el espacio de trabajo del robot delta. Por otro lado, se pueden optimizar las dimensiones de los largos de los brazos y antebrazos del robot con respecto a la energía suministrada a los motores y al espacio de trabajo.
2. **El sistema de control:** es todo lo que está relacionado con el control del movimiento del robot delta. Los motores, que mueven los brazos del robot delta, deben controlarse a través de drivers por la complejidad de su accionamiento.
3. **El controlador:** este puede tener el algoritmo que crea las trayectorias cartesianas para realizar el movimiento del robot de un punto a otro. El HMI es la interfaz que ayuda a visualizar si el movimiento deseado del robot es correcto antes de que se realice.
4. **Los componentes eléctricos:** son todo lo relacionado con la electricidad como fuentes de poder para los motores, cables de conexión, fusibles, switch, etc.

Con el propósito de crear un modelo de un robot delta como guía para el desarrollo de este trabajo, se crea una arquitectura con ejemplos reales, enfocado solo en el sistema de control, tal como se presenta en la figura (3.3) donde:

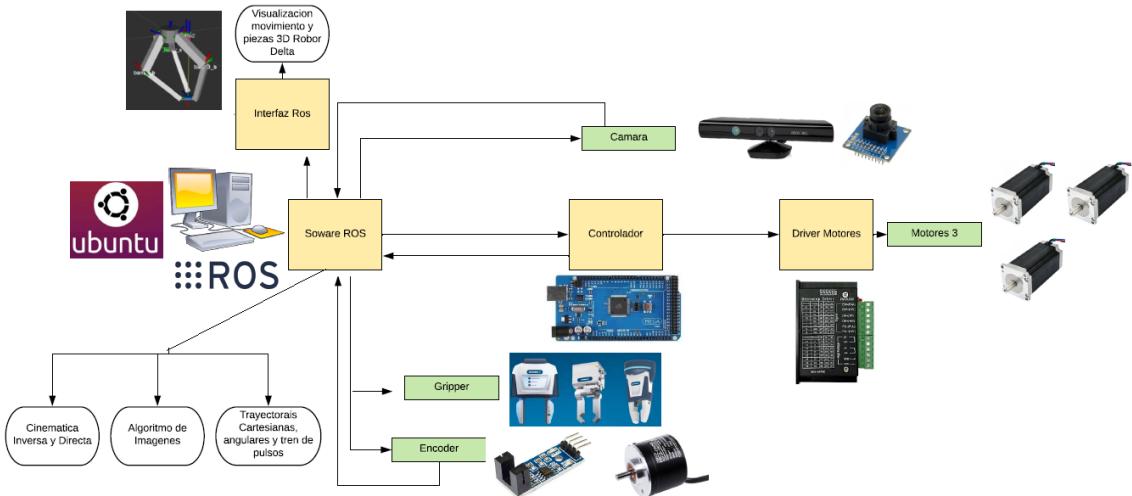


Figura 3.3: Ejemplo de el sistema de control de un robot delta

- **Software ROS:** Es el encargado de procesar y ejecutar los algoritmos de cinemática y dinámica del robot delta. Además, controla el envío, la recepción y el procesamiento de datos de los sensores y del controlador.
- **Controlador:** Realiza la conversión de la trayectoria cartesiana o angular a un formato compatible con el driver de los motores. Se puede utilizar Arduino, Raspberry Pi, etc.
- **Rviz:** Interfaz gráfica que simula las trayectorias y las partes mecánicas del robot.
- **Kinect:** Sensor RGB y de profundidad que controla la posición del robot creando un lazo cerrado.
- **Actuadores:** Motores paso a paso controlados por drivers.
- **Encoder:** Sensor de posición y velocidad angular que controla la precisión los actuadores, creando un lazo cerrado.
- **Gripper:** Pinzas que sujetan los objetos.

Para controlar a un alto nivel cada punto anterior, se necesita de mucho estudio, investigación y práctica, por lo que en esta tesis solo abarca lo relacionado con ROS y la interfaz gráfica Rviz.

3.3. Partes mecánicas

Con el objetivo de describir el robot delta, en esta sección se presenta un resumen del capítulo 2 de la tesis doctoral del ingeniero mecánico Reymond Clavel, el creador de este mecanismo, realizada en École Polytechnique Fédérale de Lausanne [22].

3.3.1. Investigación

El primer objetivo que busca Reymond Clavel es el movimiento de piezas ligeras a gran velocidad en robots, ya que las aplicaciones objetivo de su investigación se encuentran en los campos del envasado en el sector alimentario, despaletización y paletización al inicio o al final de una línea de montaje, el montaje de componentes mecánicos, etc. Para todas estas operaciones se requiere un ritmo alto de producción/operación y los contactos de la industria de Clavel confirmaban esa tendencia en aquellos tiempos.

Para lograr una alta tasa de trabajo durante operaciones que requieren carreras reducidas, el robot debe tener esencialmente una capacidad de aceleración y frenado; esta propiedad se obtiene mediante el uso de potentes actuadores debajo y mediante una estructura móvil muy ligera. Un estudio anterior de robot rápido del Clavel hizo posible probar el uso de gatos hidráulicos a alta velocidad con masas relativamente pesadas, pero por razones de coste y limpieza no querían utilizar energía hidráulica, por lo que lo llevo al enfoque de una “estructura móvil y ligera”.

La búsqueda de una función general de un robot delta se basó en una metodología enseñada en estudios de microtecnología en EPFL. Según la metodología antes mencionada, el primer paso de la cadena de costos consiste en definir la función global del producto considerado. La representación de la función se muestra en la figura 3.4, mediante una caja negra con las diferentes entradas y salidas.

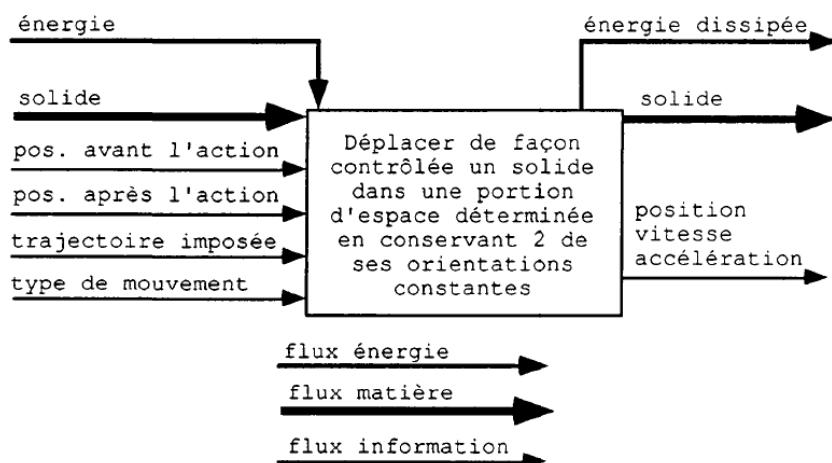


Figura 3.4: Representación de la función general del robot [22]

3.3.2. Elección del concepto “Delta“

El catálogo de soluciones del anexo A2.2 de la tesis doctoral [22] presenta 18 tipos de soluciones principales de la estructura de un robot delta que permiten mantener constantes 2 orientaciones de un sólido. La figura (3.5) muestra las soluciones más interesantes para el objetivo previsto.

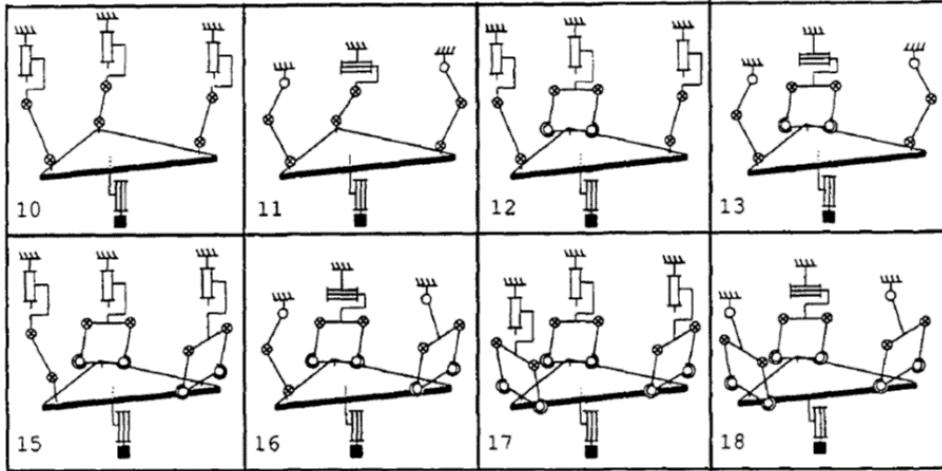


Figura 3.5: Extracto de las soluciones del catálogo ubicado en el apéndice A.2.2 [22]

Nom de la liaison (notation)	Mouvements relatifs	Nombre de degrés de liberté	Symboles	Nom de la liaison (notation)	Mouvements relatifs	Nombre de degrés de liberté	Symboles
liaison pivot (PT)	0 translation 1 rotation	1	c1 c2	liaison cardan (CA)	0 translation 2 rotations	2	c1 c2
liaison glissière (GL)	1 translation 0 rotation	1	c1 c2	liaison rotule (RO)	0 translation 3 rotations	3	c1 c2

Figura 3.6: Nombre, movimiento relativo, grados de libertad y símbolos de juntas dibujadas en la figura (3.5) [22]

Entre estas 18 soluciones, Reymond conserva aquellas que tienen las siguientes particularidades:

- El movimiento en el espacio (con tres grados de libertad) es proporcionado por los tres actuadores fijos a la base fija. Las soluciones 10 a 13 y 15 a 18 de la figura (3.5) cumplen esta condición.
- Los actuadores son del tipo giratorio. Las soluciones 11, 13, 16 y 18 cumplen esta condición.
- La estabilidad del órgano terminal está asegurada por una mayoría de elementos que trabajan en tensión-compresión más que en torsión. Finalmente se adopta la solución número 18.

3.3.3. Descripción del concepto “Delta” y sus componentes

La figura (3.7a) sirve de apoyo para la descripción del robot delta y su funcionamiento. Este es un robot con cuatro grados de libertad. Se compone principalmente de una “base fija” (1) integrada a un marco de soporte de la instalación y una placa móvil (5); el nombre que se le da a esta última pieza es “góndola” (nacelle en francés). La conexión entre la base fija (1) y la góndola (5) se realiza mediante tres cadenas cinemáticas, cada una de ellas está formado por un “brazo” (2) montado en una articulación pivotante sobre la base fija y 2 “barras paralelas” (3) provistas cada una de una articulación (4) en cada extremo. El conjunto anterior formado por 2 barras paralelas y 2 elementos de conexión al brazo y a la góndola, se denomina “paralelogramo”. Cada brazo (2) es impulsado por un “motor de brazo” (7) que, con mayor frecuencia, adopta la forma de un conjunto de motor reductor de sensor. La “Pinza” (10) del motor (6), a través del “eje telescópico” (8) provisto de una articulación tipo cardán (9) tiene oculto uno de sus extremos.

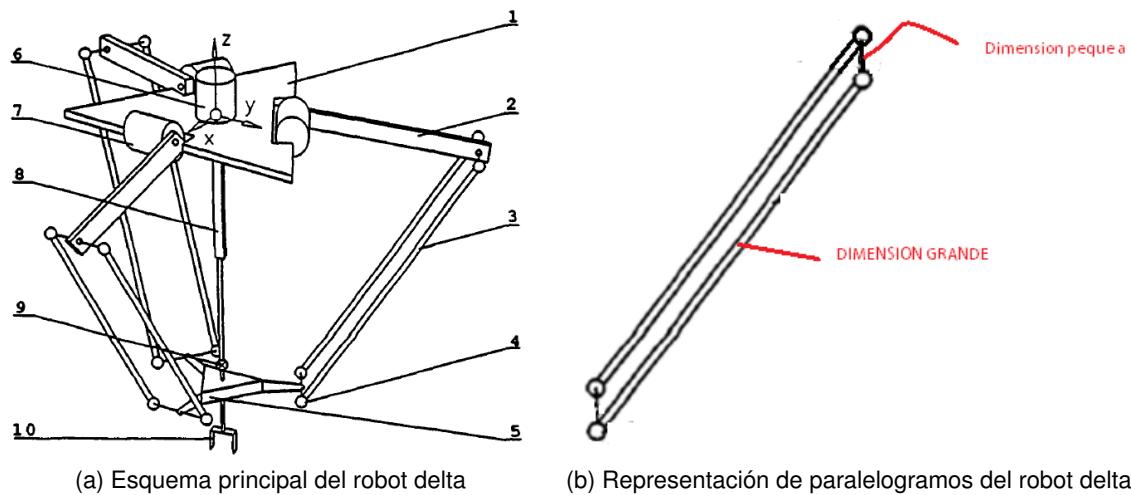


Figura 3.7: Descripción visual de un robot delta [22]

La orientación de la góndola está asegurada constantemente por los 3 paralelogramos (figura (3.7b)), que comprenden cada uno 2 dimensiones pequeñas y 2 dimensiones grandes formadas por las barras paralelas. Cada lado pequeño integrado al extremo de un brazo permanece constantemente paralelo al eje de rotación del brazo en cuestión. Los 3 pares de barras paralelas aseguran que las 3 pequeñas dimensiones integradas a la góndola permanezcan paralelas a las pequeñas dimensiones integradas a los extremos de los brazos, por tanto, paralelas a los ejes de rotación de los brazos que, por construcción, se ubican en el mismo plano. Las juntas en los extremos de las barras paralelas son del tipo de junta esférica, por lo que cada barra puede girar alrededor de su eje longitudinal; esta rotación no altera el comportamiento de esta estructura articulada que forma el paralelogramo del espacio. Una conexión por resortes y estribos entre las 2 barras paralelas simplifica la construcción de las rótulas.

3.4. Software Robor Operating System (ROS)

En esta sección se proporciona una descripción general de Robot Operating System (sus siglas ROS) y sus principales lineamientos. ROS es un marco para desarrollar software de robótica. El software está estructurado bajo un paradigma modular, pequeños paquetes que se comunican entre si mediante rápidos mensajes. Este paradigma se fomenta la reutilización de código y la colaboración global por sobre los entornos particulares.



Figura 3.8: Logo de ROS [23]

3.4.1. Historia

ROS es un gran proyecto que tiene muchos antepasados y contribuyentes. Mucha gente en la comunidad de investigación robótica sintió la necesidad de un marco de colaboración abierto. Varios proyectos en la Universidad de Stanford (figura (3.11)) a mediados de la década de 2000 involucraban inteligencia artificial incorporada e integradora, como el Stanford AI Robot (STAIR) y el programa Personal Robots (PR), que crearon prototipos internos de los tipos de sistemas de software dinámicos y flexibles como lo es ROS. Se basó en Switchyard, que era parte de un proyecto de STAIR y fue escrito por Morgan Quigley en Stanford. En 2007, Willow Garage, Inc., una incubadora de robótica cercana, proporcionó importantes recursos para extender estos conceptos mucho más y crear implementaciones bien probadas. Desde el 2013 hasta el presente, ROS es mantenido permanentemente por Open Source Robotics Foundation (OSRF, figura (3.9)) de Google y desde el 2017 cambio su nombre a Open Robotics.



Figura 3.9: OSRF [24]

La mayoría de las compañías y laboratorios de investigación en robótica están ahora portando su software a ROS. Esta tendencia también es visible en robots industriales, donde compañías están paulatinamente migrando de aplicaciones propietarias a ROS. El movimiento llamado ROS Industrial se ha incrementado en estos últimos años y su objetivo básicamente consiste en extender las capacidades avanzadas de ROS a la automatización y la robótica industrial. Este proyecto comenzó como un intento de colaboración de Yaskawa Motoman Robotics, Southwest Research Institute (SwRI) y Willow Garage (figura (3.10)). En enero de 2012, Shaun Edwards de SwRI fundó un software repositorio, alojado en Github, donde la comunidad robótica puede encontrar interfaces para manipuladores industriales, pinzas, sensores y redes de dispositivos.



Figura 3.10: Fundadores y contribuidores de ROS

Finalizando con la breve historia de ROS, se puede decir que se utiliza en estos momentos en todo el mundo en instituciones académicas, industriales y de investigación. Los desarrolladores han contribuido con miles de paquetes, incluyendo soluciones de algunos de los principales expertos del mundo en áreas específicas. Las nuevas empresas de robots ofrecen interfaces ROS con sus productos y las empresas de robots industriales ya establecidas también están introduciendo ROS. Con la adopción generalizada de ROS como el enfoque estándar de factor para la programación de robots, existe una nueva esperanza de acelerar las capacidades de los robots exponencialmente.



Figura 3.11: Logo Stanford University [25]

3.4.2. Problema Principal

ROS se creó con la intención de permitirle a los investigadores desarrollar rápidamente nuevos sistemas robóticos sin necesidad de reinventar todo nuevamente, mediante el uso de herramientas e interfaces estándar. Para hacer más eficientes los procesos de investigación y de desarrollo de software de robots, se detectaron los siguientes 3 problemas a solucionar:

- Programación secuencial es inadecuada para el mundo asincrónico robótico (figura [\(3.12\)](#))
- Los sistemas robóticos deben gestionar una complejidad significativa
- La abstracción de los detalles de un hardware específico de un robot es difícil.

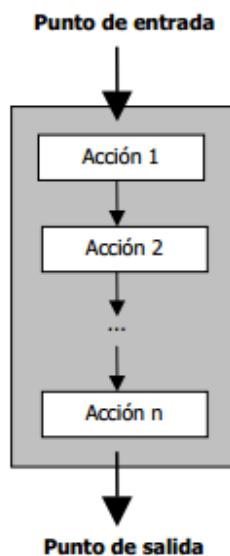


Figura 3.12: Programación Secuencial

La solución para el problema de programación secuencias se basó en “callback”, que son una función que se ejecuta siempre que hay datos disponibles para su procesamiento. Son asincrónico, ya que la devolución de llamada puede ocurrir en cualquier momento. Con respecto a el problema de softwares multifuncionales complejos y atracción de detalles de hardware específicos, se separaron los procesos en “nodos” que se comunican a través de una interfaz de mensajería. Por ejemplo, se separaron los procesos de las cámaras, edometría, escáner láser y creación de mapas y estos interactúan a través de una interfaz que comunica estos procesos por medio de “temas”. Estos últimos conceptos se explican en detalle en secciones posteriores.

3.4.3. Que es ROS

ROS es una abreviatura de Robot Operating System y es una plataforma de desarrollo de aplicaciones en robótica que provee estilos de programación (en particular, que se basa en nodos distribuidos y acoplados libremente); definiciones de interfaz y paradigmas para las comunicaciones entre nodos ; definiciones de interfaz para la incorporación de bibliotecas y paquetes; una colección de herramientas para visualización, depuración, registro de datos y diagnóstico del sistema; un repositorio de código fuente compartido; y puentes a múltiples bibliotecas útiles e independientes de código abierto.

El objetivo principal de ROS es apoyar la reutilización de código en la investigación y el desarrollo de robótica. ROS es un marco distribuido de procesos (también conocido como nodos) que permite que los ejecutables se diseñen individualmente y se acoplen libremente en tiempo de ejecución. Estos procesos se pueden agrupar en paquetes, que se pueden compartir y distribuir fácilmente. ROS también es compatible con un sistema federado de repositorios de código que también permite la distribución de la colaboración. Este diseño, desde el nivel del sistema de archivos hasta el nivel de la comunidad, permite decisiones independientes sobre el desarrollo y la implementación, pero todos pueden combinarse con las herramientas de infraestructura ROS.

ROS no es un sistema operativo convencional como Windows, Linux y Android en el sentido tradicional de gestión y programación de procesos; más bien, es un meta-sistema operativo que se ejecuta en el sistema operativo. Las diferencias entre un OS y ROS se muestran en la figura (3.13). ROS es un middleware, que proporciona los servicios que se espera de un sistema operativo, incluida la abstracción de hardware, control de dispositivos de bajo nivel, implementación de funciones de uso común, transmisión de mensajes entre procesos y administración de paquetes. También proporciona herramientas y bibliotecas para obtener, compilar, escribir y ejecutar código en varios equipos.

Conventional OS	ROS
Explicitly a general purpose OS	Exclusive for Robotic Platform
Ortho-Operational	Meta-Operational
Native Language Programming	Language-independent architecture
Sequential Architecture	Asynchronous Distributed architecture
Programming IDE	Software Frameworks
Proprietary/Open-Source	Open-source under BSD license
Heavily coded frameworks	ROS frameworks are very light
Programs	Nodes
Communication	Messages
Kernel is Included	Kernelless

Figura 3.13: Diferencias entre un sistema operativo convencional y ROS

Las principales características de ROS se pueden resumir de la siguiente manera:

- **Plumbing:** ROS proporciona una infraestructura de mensajería de publicación y suscripción diseñada para respaldar la construcción rápida y sencilla de sistemas informáticos distribuidos.
- **Herramientas** ROS proporciona un amplio conjunto de herramientas para configurar, iniciar, introspectar, depurar, visualizar, registrar, probar y detener sistemas informáticos distribuidos.
- **Capacidades** ROS proporciona una amplia colección de bibliotecas que implementan funcionalidades robóticas útiles, con un enfoque en la movilidad, la manipulación y la percepción.
- **Ecosistema:** ROS es apoyado y mejorado por una gran comunidad, con un fuerte enfoque en la integración y documentación. <https://www.ros.org/> es una ventanilla única que busca y aprende sobre los miles de paquetes ROS que están disponibles a través de desarrolladores de todo el mundo.

La figura (3.14) muestra que la comunicación de datos ROS es compatible no solo con un sistema operativo, sino también con múltiples sistemas operativos, hardware y programas, lo que lo hace muy adecuado para el desarrollo de robots donde se combinan varios hardware.

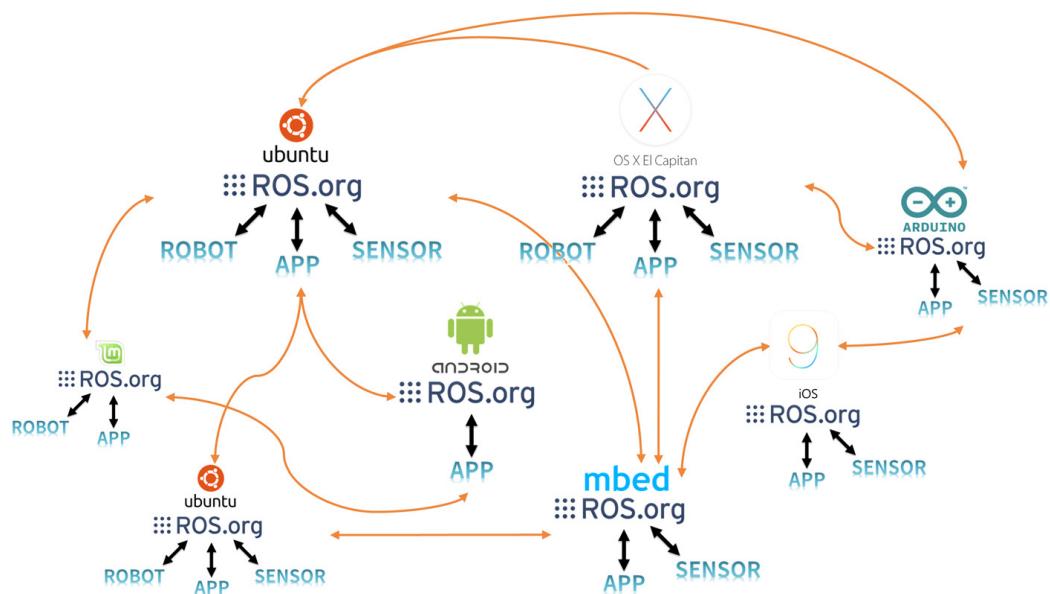


Figura 3.14: Compatibilidad de ROS y otras plataformas [26]

3.4.4. Objetivos de ROS

Todos los marcos de software imponen sus filosofías de desarrollo a sus colaboradores directa o indirectamente, a través de sus modismos y prácticas comunes. En términos generales, ROS sigue la filosofía de desarrollo de software de Unix en varios aspectos clave. Esto tiende a hacer que ROS se sienta “natural” para los desarrolladores que vienen de Unix, pero algo “críptico” al principio para aquellos que han utilizado principalmente entornos de desarrollo gráfico en Windows o Mac OS X. Los siguientes párrafos describen varios aspectos filosóficos de ROS.

3.4.4.1. Peer to peer (Nodos)

Los sistemas ROS consisten en numerosos programas pequeños de computadora que se conectan entre sí e intercambian mensajes continuamente. Estos mensajes viajan directamente de un programa a otro; no hay un servicio de enrutamiento central. Aunque esto hace que la “plumbing” subyacente sea más compleja, el resultado es un sistema que escala mejor a medida que aumenta la cantidad de datos. La mentalidad Peer to peer se representa en la figura (3.15).

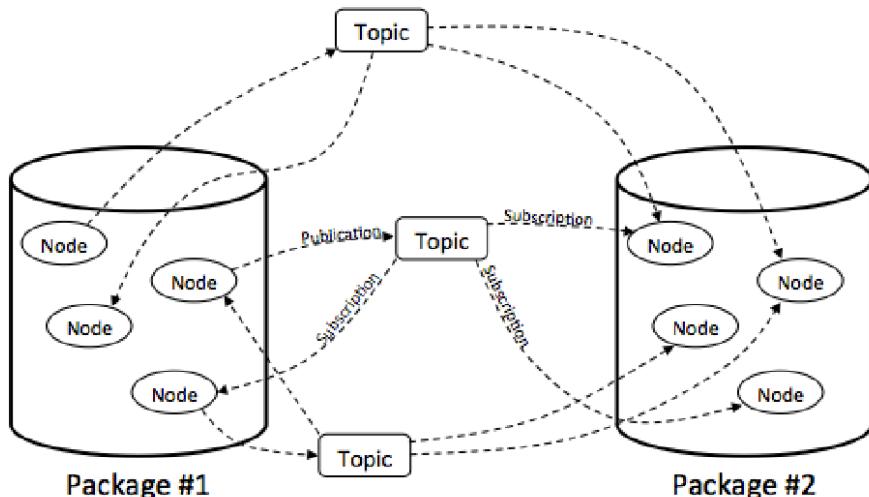


Figura 3.15: Modelo de comunicación ROS: nodos publican y se suscriben a temas [27]

3.4.4.2. Orientación hacia las herramientas

Como lo demuestra la arquitectura duradera de Unix, se pueden crear sistemas de software complejos a partir de muchos programas pequeños y genéricos. A diferencia de muchos otros marcos de software de robótica, ROS no tiene un entorno de ejecución y desarrollo integrado canónico. Tareas como navegar por el árbol del código fuente, visualizar las interconexiones del sistema, trazar gráficamente los flujos de datos, generar

documentación, registrar datos, etc, son todas realizadas por programas separados. Esto fomenta la creación de implementaciones nuevas y mejoradas, ya que (idealmente) se pueden intercambiar por implementaciones más adecuadas para un dominio de tarea en particular. Las versiones recientes de ROS permiten que muchas de estas herramientas se compongan en procesos únicos para mayor eficiencia o para crear interfaces coherentes para operadores o depuración, pero el principio sigue siendo el mismo: las herramientas individuales en sí mismas son relativamente pequeñas y genéricas.

3.4.4.3. Multilenguaje

Muchas tareas de software son más fáciles de realizar en lenguajes de secuencias de comandos de "alta productividad" como Python o Ruby. Sin embargo, hay ocasiones en las que los requisitos de rendimiento exigen el uso de lenguajes más rápidos, como C++. También hay varias razones por las que algunos programadores prefieren lenguajes como Lisp o MATLAB. Se han librado guerras interminables de correo electrónico, se están librando actualmente y, sin duda, se seguirá librando sobre qué idioma es el más adecuado para una tarea en particular. Reconociendo que todas estas opiniones tienen mérito, que los lenguajes tienen diferentes utilidades en diferentes contextos, y que la experiencia única de cada programador es muy importante al elegir un idioma, ROS eligió un enfoque multilingüe. Los módulos de software ROS se pueden escribir en cualquier idioma para el que se haya escrito una biblioteca cliente. En el momento de escribir este artículo, existen bibliotecas cliente para C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, Haskell, R, Julia y otros. Las bibliotecas de cliente ROS se comunican entre sí siguiendo una convención que describe cómo los mensajes se "aplanan" o "serializan" antes de transmitirse a través de la red. La figura (3.17) representa un ejemplo de multilenguaje en ROS. En ella se muestra el nodo "Laser Scanner" escrito en lenguaje Python con el objetivo de obtener datos de un entorno físico por medio de un escáner de láser y el nodo "Map Building" escrito en lenguaje C++ es el encargado de transformar los datos obtenidos por del nodo anterior para poder visualizarlos a la computadora.

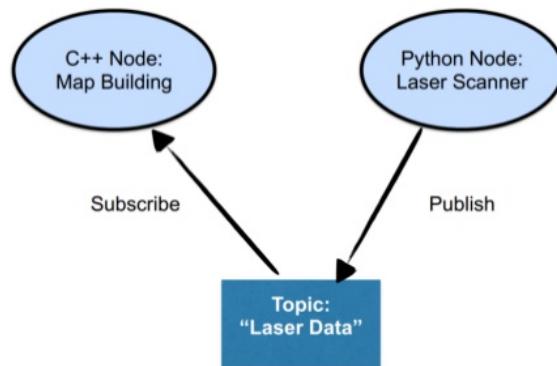


Figura 3.16: Ejemplo de multilenguaje de ROS

3.4.4.4. Pequeño

Las convenciones ROS alientan a los contribuyentes a crear bibliotecas independientes y luego empaquetar esas bibliotecas para que puedan enviar y recibir mensajes hacia y desde otros módulos ROS. Esta capa adicional está destinada a permitir la reutilización de software fuera de ROS para otras aplicaciones, y simplifica en gran medida la creación de pruebas automatizadas utilizando herramientas de integración continua estándar.

3.4.4.5. Libre y Open Source

El núcleo de ROS se publica bajo la licencia BSD permisiva, que permite el uso comercial y no comercial. ROS pasa datos entre módulos mediante comunicación entre procesos (IPC), lo que significa que los sistemas construidos con ROS pueden tener licencias detalladas de sus diversos componentes. Los sistemas comerciales, por ejemplo, a menudo tienen varios módulos de fuente cerrada que se comunican con una gran cantidad de módulos de fuente abierta. Los proyectos académicos y de pasatiempos suelen ser de código abierto. El desarrollo de productos comerciales a menudo se realiza completamente detrás de un firewall. Todos estos casos de uso, y más, son comunes y perfectamente válidos bajo la licencia ROS.



Figura 3.17: Licencia de ROS

3.4.5. Estadísticas

3.4.5.1. Science Robotics

La revista Science Robotics publicó un artículo el año 2017 llamado “Powering the world’s robots 10 years of ROS” [28] que habla sobre la iniciativa ROS-I (Ros Industrial) lanzada el año 2012. Debido a las arquitecturas de software limitadas de los robots industriales actuales, es demasiado caro aplicar capacidades robóticas avanzadas para mejorar la productividad industrial. ROS-I proporciona interfaces para robots industriales comunes y dispositivos sensoriales junto con bibliotecas de software específicas para la automatización de la fabricación. ROS-I ha apoyado un número creciente de hardware industrial, como los robots producidos por ABB, Fanuc y Yaskawa. Además, el Consorcio ROS-I existe para desarrollar la comunidad ROS-I proporcionando apoyo técnico, organizando cursos de capacitación y talleres, estableciendo la hoja de ruta para ROS-I. El Consorcio ROS-I tiene más de 50 miembros en todo el mundo, incluidos institutos de investigación y agencias gubernamentales, integradores de sistemas y usuarios finales, y fabricantes de equipos originales. Además, la próxima versión de ROS 2.0 debería abordar una de las principales limitaciones que ha ralentizado la adopción de ROS en la industria: que es una implementación de middleware interna. ROS 2.0 ahora se basa en el servicio de distribución de datos para la comunicación entre procesos, lo que brinda una confiabilidad mucho mejor con protocolos de calidad de servicio y seguridad con cifrado.

La figura (3.18) muestra algunas estadísticas clave de ROS, las cuales son:

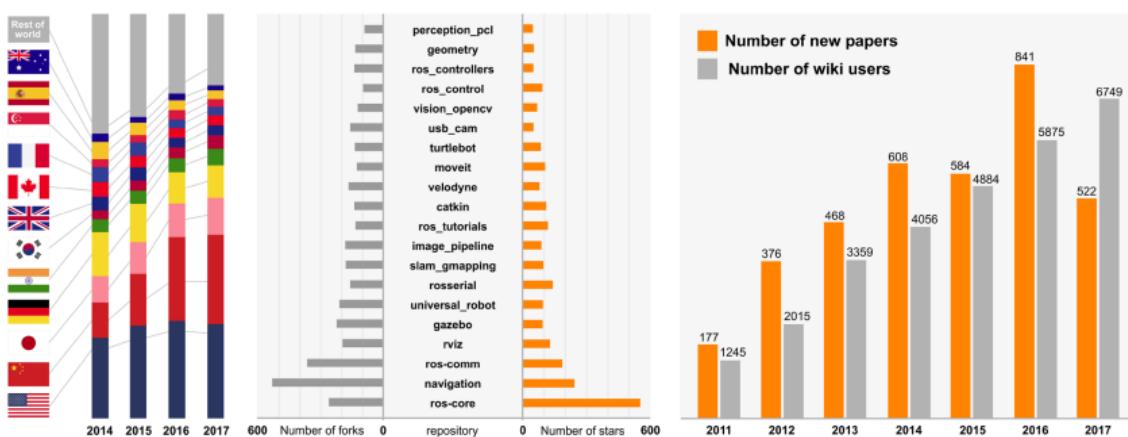


Figura 3.18: Estadísticas mundiales de ROS [28]

- **Los países visitantes de la wiki de ROS:** en los últimos 4 años muestra un cambio creciente de investigadores de países del lejano oriente (información recolectada del informe métrico anual de ROS).

- **Los 20 principales repositorios ROS:** según la cantidad de bifurcaciones y estrellas (de Github, 10 de octubre de 2017).
- **Artículos recientemente publicados:** en los últimos 10 años que han citado el artículo "ROS original" (de la búsqueda de Google Scholar, palabras clave "Sistema operativo de robot ROS") y el número de usuarios de wiki registrados.

La figura (3.19) muestra los diez años de desarrollos de ROS que respaldan la investigación y el desarrollo, incluida la robótica móvil, industrial, quirúrgica y espacial, así como los automóviles autónomos. Se han lanzado 11 distribuciones de ROS en los últimos 10 años, y el gráfico de barras muestra el número de descargas binarias de ROS en cada año. El Consorcio ROS-I se lanzó en 2013 con el objetivo de transformar las capacidades de ROS en tiempo real para robots industriales. ROS 2.0 esta actualmente en desarrollo intenso, y se acaba de lanzar una versión beta.

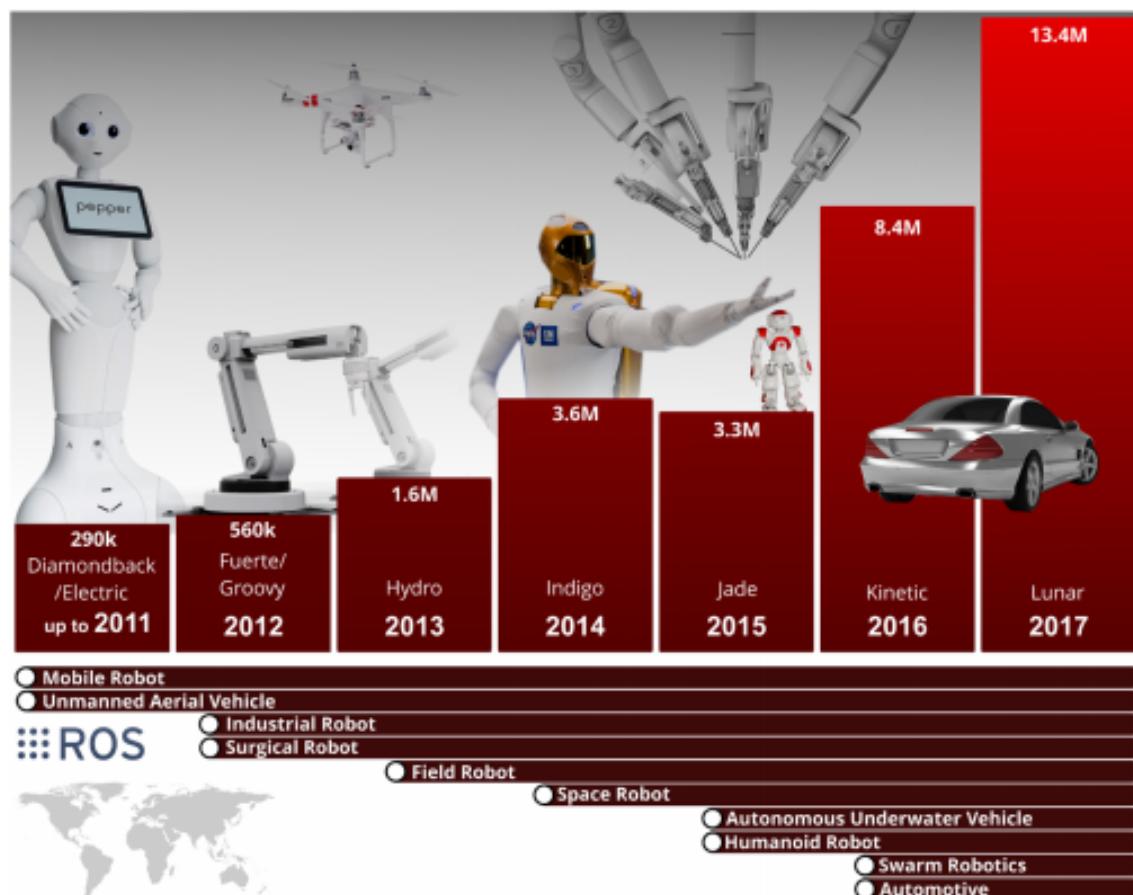


Figura 3.19: Desarrollo de ROS en los últimos 10 años [28]

3.4.5.2. Metricas de la comunidad ROS

Ros Metrics miden aspectos de la comunidad ROS para comprender y rastrear el impacto de su trabajo e identificar áreas de mejora. Se inspiran en las métricas del Proyecto MeeGo .



Figura 3.20: Logo de ROS Metrics [29]

Las figuras (3.21) y (3.22) presentan estadísticas recolectadas de la pagina oficial de ROS Metrics hasta el año 2021. Según la figura (3.21), los países que más han visitado ROS son China, EEUU, Japón y Alemania.



Figura 3.21: Top de países que utilizan ROS en base a la descarga de paquetes registrados hasta el 28 de marzo del año 2021 [29].

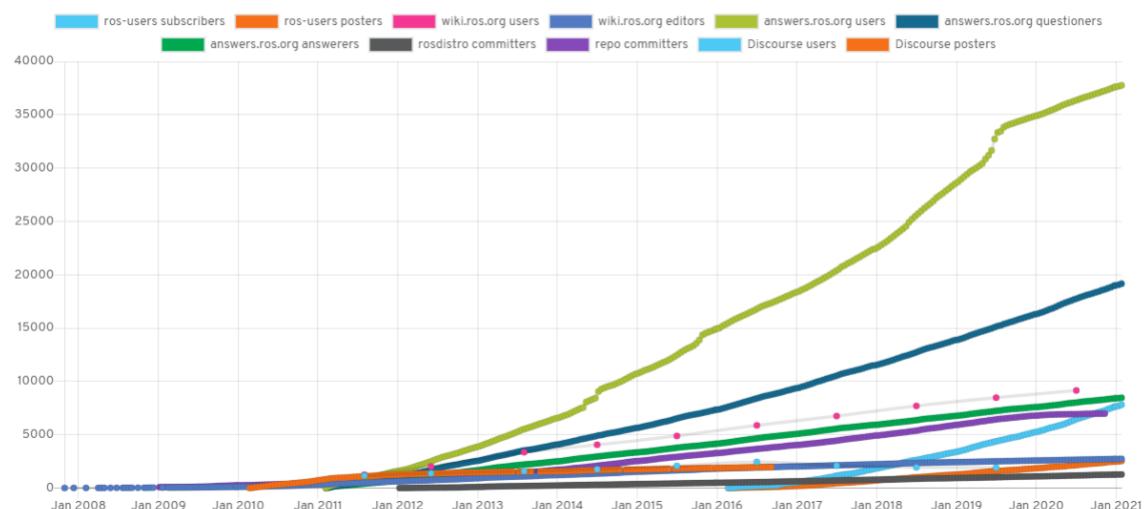


Figura 3.22: Colección de diferentes métricas para medir la cantidad de usuarios de la comunidad ROS [29]

3.4.5.3. Repositorio GitHub

Para comprender GitHub, primero se debe comprender Git. Git es un sistema de control de versiones de código abierto que fue iniciado por Linus Torvalds, la misma persona que creó Linux. Sirve para cuando los desarrolladores crean algo (una aplicación, por ejemplo), realizan cambios constantes en el código, lanzando nuevas versiones hasta y después del primer lanzamiento oficial. Los sistemas de control de versiones mantienen estas revisiones en orden, almacenando las modificaciones en un repositorio central. Esto permite a los desarrolladores colaborar fácilmente, ya que pueden descargar una nueva versión del software, realizar cambios y cargar la revisión más reciente. Todos los desarrolladores pueden ver estos nuevos cambios, descargarlos y contribuir. Del mismo modo, las personas que no tienen nada que ver con el desarrollo de un proyecto aún pueden descargar los archivos y usarlos. Git es una herramienta de línea de comandos, pero el centro alrededor del cual giran todas las cosas relacionadas con Git es el centro Github, donde los desarrolladores almacenan sus proyectos y se conectan con personas de ideas afines.



Figura 3.23: Logo GitHub, Inc.

Dirk Thomas trabajó en ROS durante casi 9 años y envió 15 distribuciones de ROS. Uno de los paquetes que subió al repositorio de GitHub registra el total de autores, commits y repositorios en GitHub con relación a ROS hasta octubre del 2020 (figura 3.24).

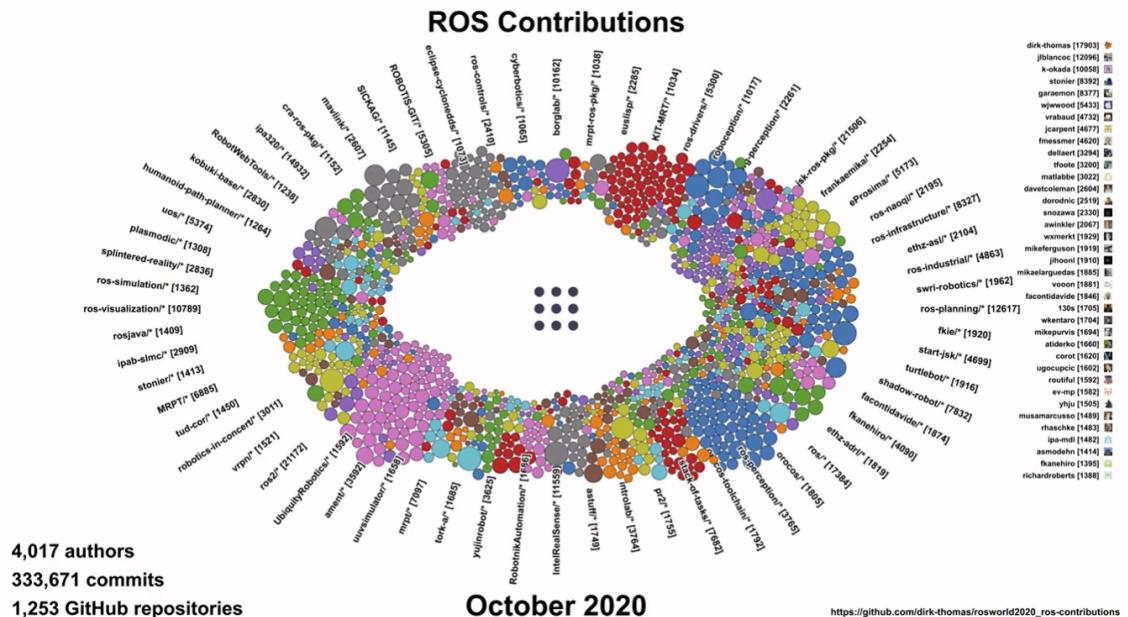


Figura 3.24: Visualización de las contribuciones de ROS en GitHub [30]

3.4.6. Versiones

En 2007, Willow Garage logró la investigación del marco de software de robot que comenzó en el laboratorio de inteligencia artificial de la Universidad de Stanford y continuó el desarrollo bajo el nombre de Robot Operating System. Con la sexta versión de lanzamiento oficial ROS Groovy Galápagos", Willow Garage intentó penetrar en el mercado de robots de servicios comerciales en 2013, pero terminó dividiéndose en varias empresas emergentes y finalmente se entregó a la Open Source Robotics Foundation. Desde entonces, se lanzaron 4 versiones más y, a partir de mayo de 2017, OSRF cambió su nombre a Open Robotics y ha estado desarrollando, operando y administrando ROS. Más recientemente, la undécima versión de ROS, ROS Lunar Loggerhead, fue lanzada el 23 de mayo de 2017. ROS etiqueta la primera letra de cada nombre de lanzamiento en orden alfabético y usa una tortuga como su símbolo (ver figura (3.25)).

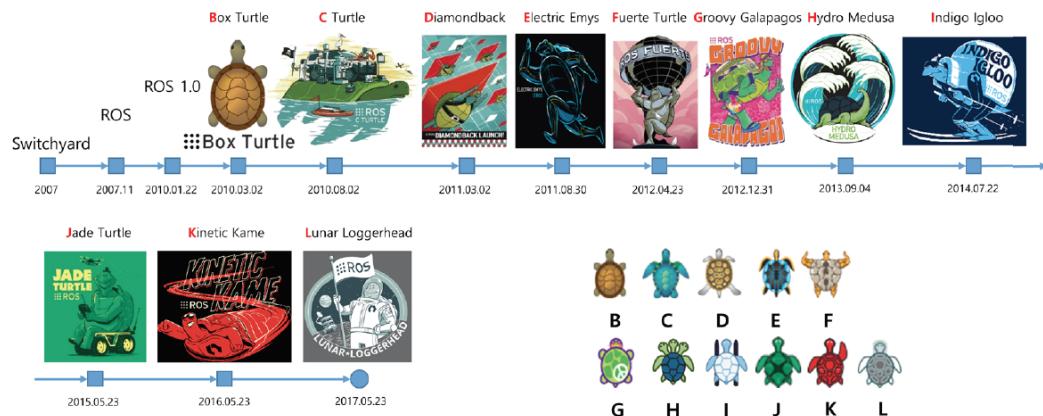


Figura 3.25: Línea temporal e iconos de tortuga para cada versión de ROS [26]

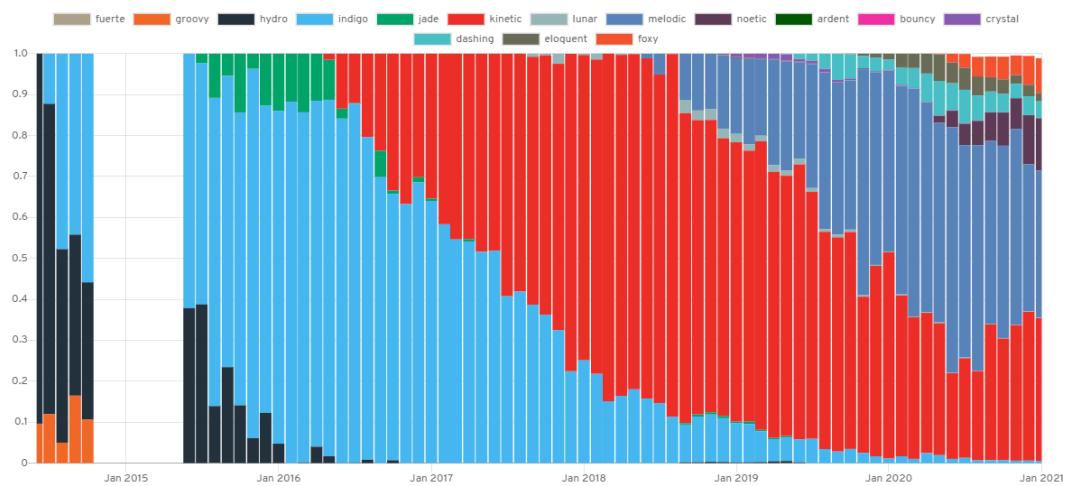


Figura 3.26: Uso relativo de cada versión de ROS basado en descargas desde packages.ros.org [29].

3.4.7. Conceptos principales

Existen tres niveles de conceptos en ROS. El primer nivel llamado "nivel de sistema de archivos" se refiere a la manera en que diferentes archivos están organizados en la computadora, herramientas ROS para administrar código fuente, instrucciones de compilación y definiciones de mensajes. El segundo nivel llamado "nivel de gráfico computacional" son los componentes de la red peer to peer de nodos ROS, es decir, los procesos. Por último, el tercer nivel llamado "nivel comunitario" se relaciona con la forma en que los usuarios comparten software para permitir que todos lo utilicen.

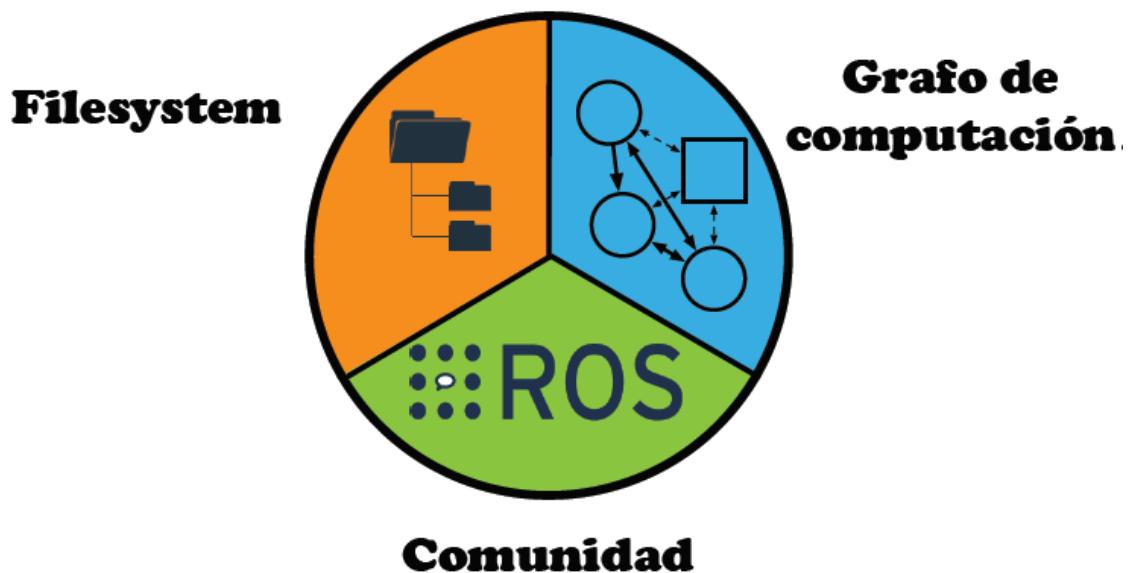


Figura 3.27: Niveles de conceptos de ROS

3.4.7.1. Nivel de sistemas de archivos

Similar a un sistema operativo, los archivos de ROS también se organizan en el disco duro de una manera particular. En este nivel, se puede ver cómo se organizan estos archivos en el disco. La figura (3.28) muestra cómo se organizan los archivos y carpetas:

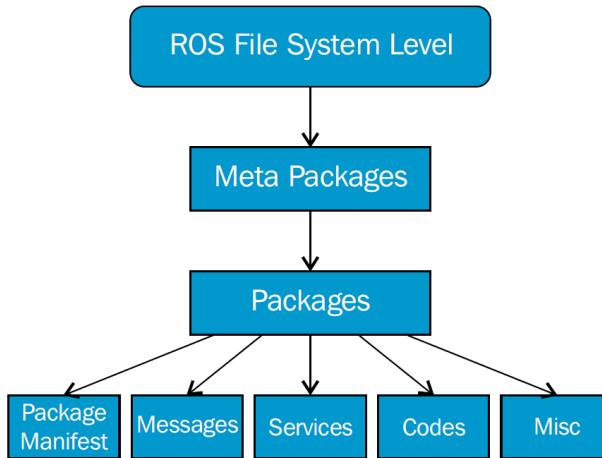


Figura 3.28: Nivel de sistema de archivos ROS [31]

A continuación se presenta la descripción de cada bloque en el sistema de archivos:

- **Meta packages:** Es un único paquete lógico compuesto por muchos paquetes en su interior. Además, los meta paquetes contienen un archivo package.xml que describe la carpeta y sus dependencias.
- **Packages:** Son las unidades más bajas y la principal para organizar el software en ROS. Un paquete puede contener procesos en tiempo de ejecución ROS (nodos), un biblioteca dependiente de ROS, conjuntos de datos, archivos de configuración o archivos de lanzamiento. Cada directorio de paquete debe incluir un archivo CMakeList.txt y package.xml que describa el contenido del paquete y cómo catkin debe interactuar con él.
- **Stacks:** Son grupos de paquetes que se juntan para realizar funciones de alto nivel.
- **Manifests:** Los manifiestos proporcionan metadatos sobre un paquete. Pueden pertenecer a un packages o una Stacks y describen información general sobre un paquete o Stacks específico, como una breve descripción de lo que hace, el nombre del autor, su tipo de licencia y las dependencias con otros paquetes.
- **Message (msg) types:** Las descripciones de los mensajes definen los datos estructurados para mensajes enviados en ROS.
- **Service (srv) types:** Las descripciones de servicio definen la solicitud y estructuras de datos de respuesta para servicios en ROS.

ROS está basado en CMake y puede ser utilizado por muchos sistemas operativos como Linux o Windows. CMake es un generador de herramientas de creación, es una herramienta de nivel superior que simplifica el proceso de compilación y construcción. Puede gestionar grandes proyectos y tiene una buena escalabilidad. Para una plataforma a gran escala como ROS, ha extendido CMake a un sistema de compilación llamado Catkin. Catkin es el sistema de compilación de ROS que genera programas ejecutables, bibliotecas e interfaces y proporciona el concepto de espacio de trabajo en la construcción de cada proyecto.

La estructura del espacio de trabajo catkin, incluye las carpetas /src, /build, /devel. Las tres rutas también pueden incluir otras en algunas opciones de compilación pero estas tres carpetas son las predeterminadas para el sistema de compilación catkin. Sus roles específicos son los siguientes:

- **Espacio fuente (/src):** Contiene el código fuente donde el usuario puede extraer, verificar o clonar el código fuente de los paquetes que quiere construir.
- **Espacio de construcción (/build):** Este es el espacio donde CMake apela para construir los paquetes en el espacio de trabajo de catkin. CMake y Catkin mantienen su información de caché y otros intermedios archivos.
- **Espacio de desarrollo (/devel):** Se encuentran archivos de objetos generados (incluidos archivos de encabezado, bibliotecas de vínculos dinámicos, bibliotecas de vínculos estáticos, archivos ejecutables, etc.) y variables de entorno.

Durante el proceso de compilación, el flujo de trabajo de un catkin workspace es el que se muestra con flechas azules en la figura (3.29). Los dos últimos procesos son generados y administrados automáticamente por el sistema catkin. El uso principal para los usuarios de ROS es la carpeta src, donde los paquetes creados o copiados se almacenan en dicha carpeta. En el momento de la compilación, el sistema de compilación de catkin busca y compila todos los paquetes de código fuente de forma de la ruta /src.

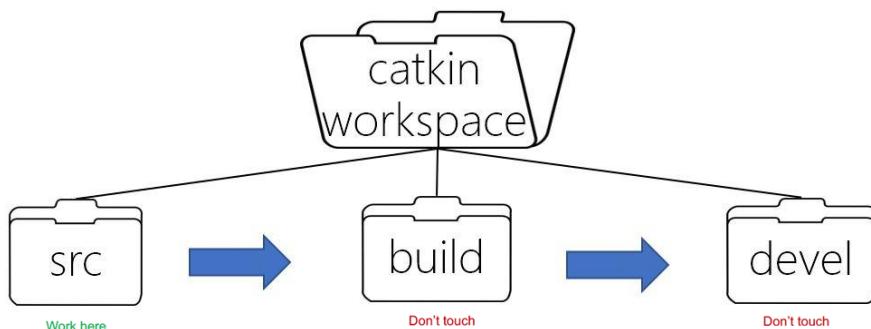


Figura 3.29: Espacio de trabajo catkin_make [32]

Cuando se crea un paquete, normalmente es común encontrar las carpetas que se muestran en la figura (3.30):

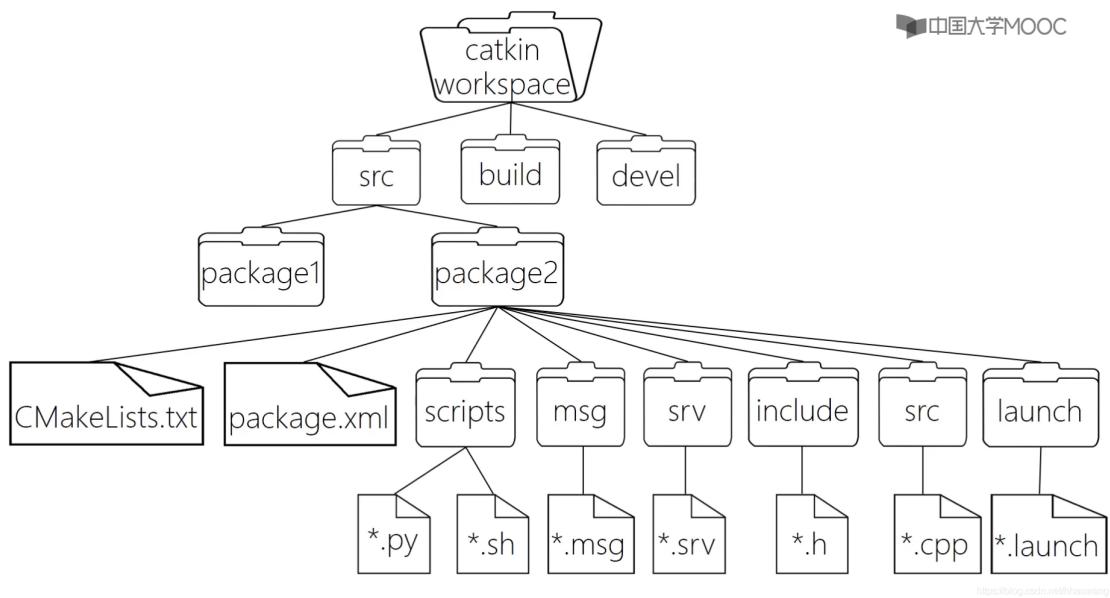


Figura 3.30: Estructura y características de catkin_make [32]

- **/include/**: Esta carpeta contiene los encabezados (*.h) y las bibliotecas (*.lib, *.so).
- **/config/**: Todas las configuraciones se almacenan en esta carpeta incluidos los parámetros definidos, como controlador de robot con una extensión (*.yaml).
- **/launch/**: Contiene los archivos (*.launch) para ejecutar los nodos seleccionados del paquete o de otros paquetes usando el comando <incluir/>.
- **/msg/**: Contiene el tipo de datos del mensaje para nuestra aplicación (*.msg).
- **/srv/**: Esta carpeta contiene el tipo de mensaje para los servicios (*.srv).
- **/actions/**: Contiene el tipo de datos del mensaje utilizado en las acciones (*.action).
- **/src/**: Todo el código que se va a computar en el paquete debe estar dentro de esta carpeta con extensión (*.cpp).
- **package.xml**: Descripción del paquete y lista de dependencias utilizadas por el.
- **CmakeLists.txt**: Lista de requisitos y dependencias que debe compilar el ejecutor del paquete.

3.4.7.2. Nivel gráfico computacional

El nivel gráfico computacional describe la infraestructura que utilizan los componentes ROS para comunicarse. Se basa en una red de procesos peer-to-peer (nodos) y se compone de varias unidades, tales como las que se muestran en la figura (3.31):

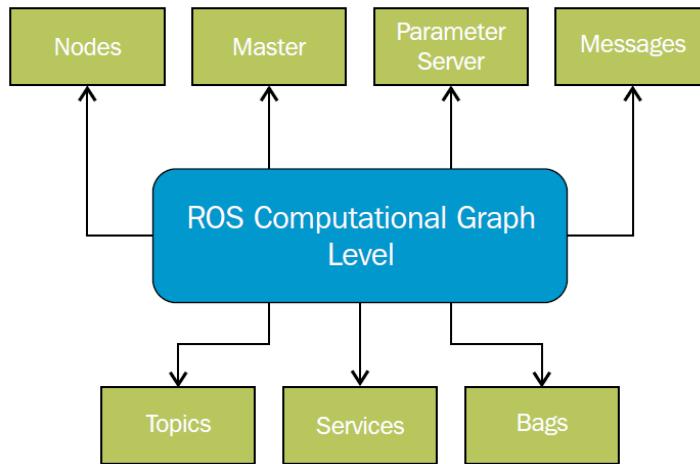


Figura 3.31: Estructura de la capa ROS Graph [31]

3.4.7.2.1. Nodes (Nodos) Son ejecutables en el sistema ROS y realizan la parte computacional. Se pueden conectar a otros nodos mediante 2 tipos de comunicación: Publisher/Subscriber o Services. Cada nodo tiene su propio nombre para ser reconocido y distinguido de los demás nodos. En ROS existen muchos lenguajes para programar un nodo, como C++, Python o Java.

3.4.7.2.2. Ros Master Ros Master habilita el funcionamiento de la red ROS y es el primer proceso que debe ejecutarse cuando estamos usando ROS. Su función es registrar todos los nodos que se ejecutan, los temas y servicios existentes. Esto es esencial para interconectar nodos y monitorear todas las unidades en el sistema.

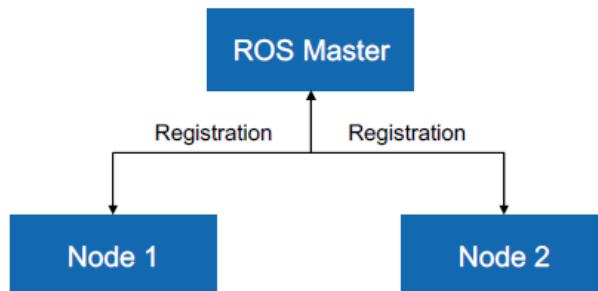


Figura 3.32: Registro de nodos en ROS Master [33].

3.4.7.2.3. Topics (Temas) Los nodos comparten información pasando y recibiendo mensajes. Estos mensajes se publican con un nombre especial llamado tema. Por lo tanto, los nodos pueden publicar o suscribirse a un tema específico para obtener los mensajes deseados. Muchos nodos se pueden suscribir al mismo tema para recibir los mensajes que necesitan. Las comunicaciones entre nodos se realizan de forma unidireccional, por lo que cualquier nodo que envíe la información sobre un tema no puede recibir respuesta en el mismo canal, ni saber si el nodo receptor ha recibido el mensaje.

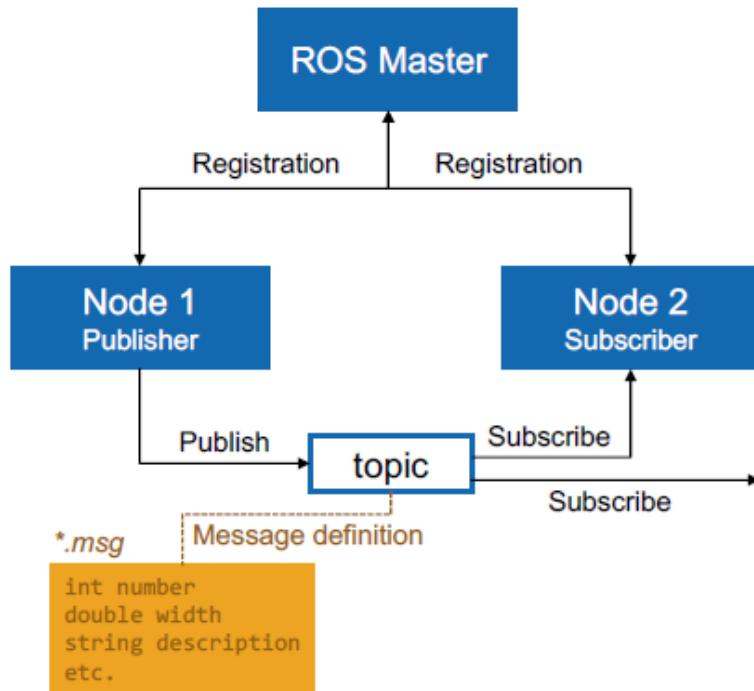


Figura 3.33: Nodos comunicándose a través de topic por medio de un mensaje *.msg [33].

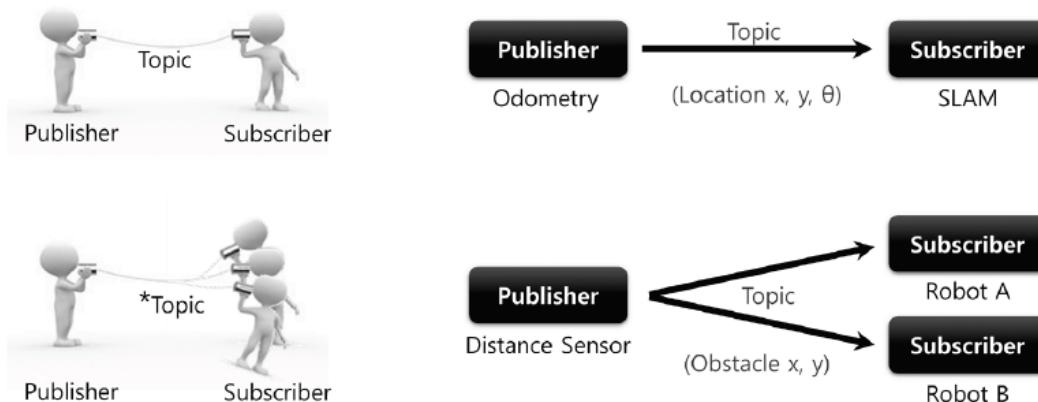


Figura 3.34: Comunicación de mensajes por medio de topics [26]

3.4.7.2.4. Services (Servicios) Aunque el principal método de comunicación entre nodos es a través de temas ya explicado anteriormente, existe otra metodología en ROS llamada llamadas de servicio. Las llamadas de servicio en ROS son de diferentes a los temas debido a las siguientes características: son bidireccionales y uno a uno.

Este proceso síncrono de comunicación se basa en un sistema cliente/servidor o solicitud/respuesta, donde un nodo cliente envía algunos datos llamados solicitud a un nodo servidor y espera hasta que el servidor pueda responder. Los servidores, habiendo recibido esta solicitud, toma alguna acción y enviar algunos datos llamados respuesta al nodo cliente. La descripción del servicio se almacena en un archivo de definición de servicio *.srv, guardado en un subdirectorio /srv.

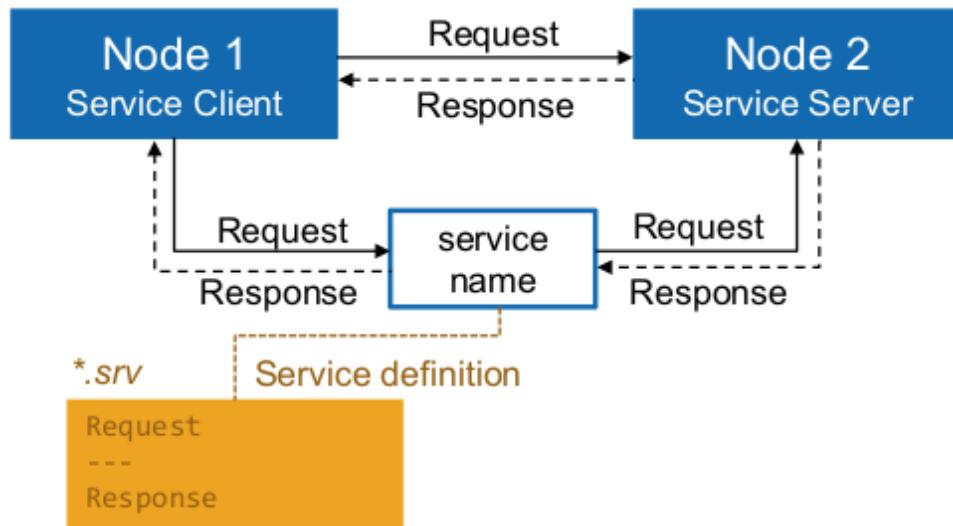


Figura 3.35: Comunicación request/response entre nodos que realizan servicios [33].

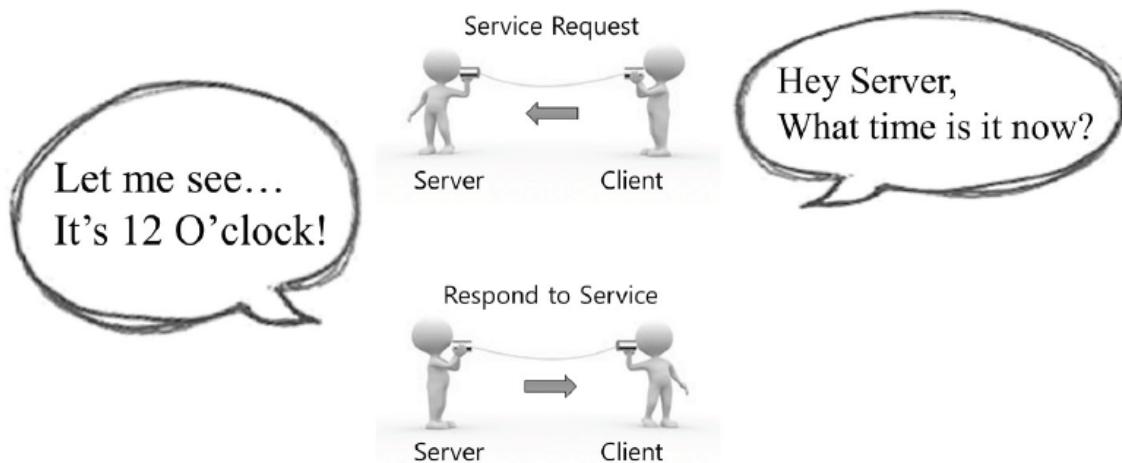


Figura 3.36: Comunicación de mensajes de servicio [26].

3.4.7.2.5. Actions (Acciones) Una acción es una comunicación bidireccional asíncrona entre los nodos activos basados en el cliente de la acción y el sistema del servidor de la acción, como se muestra en la figura (3.37). El sistema de acción, como los servicios explicados anteriormente, envía algo del cliente de acción de datos al servidor de acción llamado objetivo y el servidor de acción responde un resultado. Las acciones se diferencian de los servicios porque el servidor puede proporcionar retroalimentación y estado al cliente en cualquier momento sobre el proceso que se está realizando y el nodo cliente puede cancelar el objetivo anterior requerido para el servidor en cualquier momento. Al igual que en las llamadas de servicio ROS, las acciones deben definir pocos mensajes para comunicar al cliente con el servidor. Esto es posible gracias a las especificaciones de las acciones, que definen estos mensajes en un archivo de acción con extensión *.action. Estos archivos se almacenan en una subcarpeta /action/ en el directorio del paquete. El archivo *.action se divide en 3 secciones. Cada parte está separada por 3 guiones (—). En la primera parte se define el objetivo, en la segunda se establece la retroalimentación y en la última se especifica la definición del resultado.

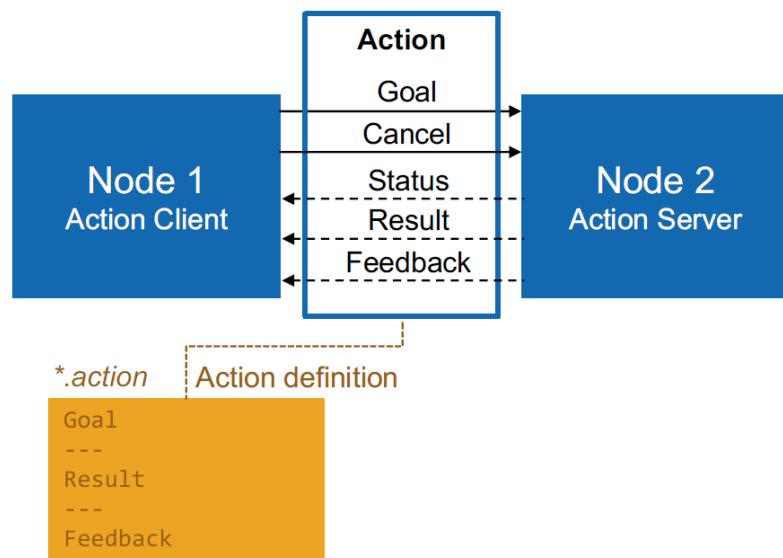


Figura 3.37: Comunicación bidireccional entre nodos por medio de acciones [33].



Figura 3.38: Comunicación de mensajes de acción [26].

3.4.7.2.6. Messages (Mensajes) Un mensaje es simplemente una estructura de datos, que comprende campos escritos. Tipos primitivos estándar (entero, punto flotante, booleano, etc.) son compatibles (figura (3.39)). En muchos paquetes utilizados en ROS se incluyen mensajes especiales llamados 'encabezado', que son básicamente mensajes creados por otros. Hay tres campos definidos en un mensaje de encabezado:

- **uint32 seq:** Corresponde al número del mensaje enviado por un editor determinado
- **time stamp:** La hora exacta a la que se envió el mensaje
- **string frame_id:** Muestra el sistema de referencia utilizado

ROS Data Type	Serialization	C++ Data Type	Python Data Type
bool	unsigned 8-bit int	uint8_t	bool
int8	signed 8-bit int	int8_t	int
uint8	unsigned 8-bit int	uint8_t	int
int16	signed 16-bit int	int16_t	int
uint16	unsigned 16-bit int	uint16_t	int
int32	signed 32-bit int	int32_t	int
uint32	unsigned 32-bit int	uint32_t	int
int64	signed 64-bit int	int64_t	long
uint64	unsigned 64-bit int	uint64_t	long
float32	32-bit IEEE float	float	float
float64	64-bit IEEE float	double	float
string	ascii string	std::string	str
time	secs/nsecs unsigned 32-bit ints	ros::Time	rospy.Time
duration	secs/nsecs signed 32-bit ints	ros::Duration	rospy.Duration

Figura 3.39: Tipos de datos basicos para mensajes en ROS [26].

3.4.7.2.7. Parameter Server (Servidor de parámetros) El servidor de parámetros permite que los datos se almacenen por clave en una ubicación. Actualmente forma parte del Máster. Básicamente es un almacén de constantes.

3.4.7.2.8. Bags (Bolsas) Las bolsas son un formato para guardar y reproducir datos de mensajes ROS. Las bolsas son un mecanismo importante para almacenar datos, como los datos de los sensores, que pueden ser difíciles de recopilar, pero son necesarios para desarrollar y probar algoritmos.

3.4.7.2.9. Esquema resumen de nivel gráfico El sistema ROS permite que diferentes nodos se comuniquen entre sí, intercambiando información y datos. Sin embargo, todo el sistema necesita un ROS Master en ejecución para notar la existencia de otros nodos y comenzar a comunicarse entre sí. El ROS Master permite que los nodos individuales se ubiquen entre sí en el sistema, rastrean a los publicadores y suscriptores de temas y servicios. Un nodo es generalmente un pequeño programa escrito en Python o C ++ que ejecuta alguna tarea o proceso relativamente simple. Los nodos se pueden iniciar y detener de forma independiente entre sí y se comunican pasando mensajes. Un nodo puede publicar mensajes otros nodos sobre determinados temas, servicios o acciones específicos.

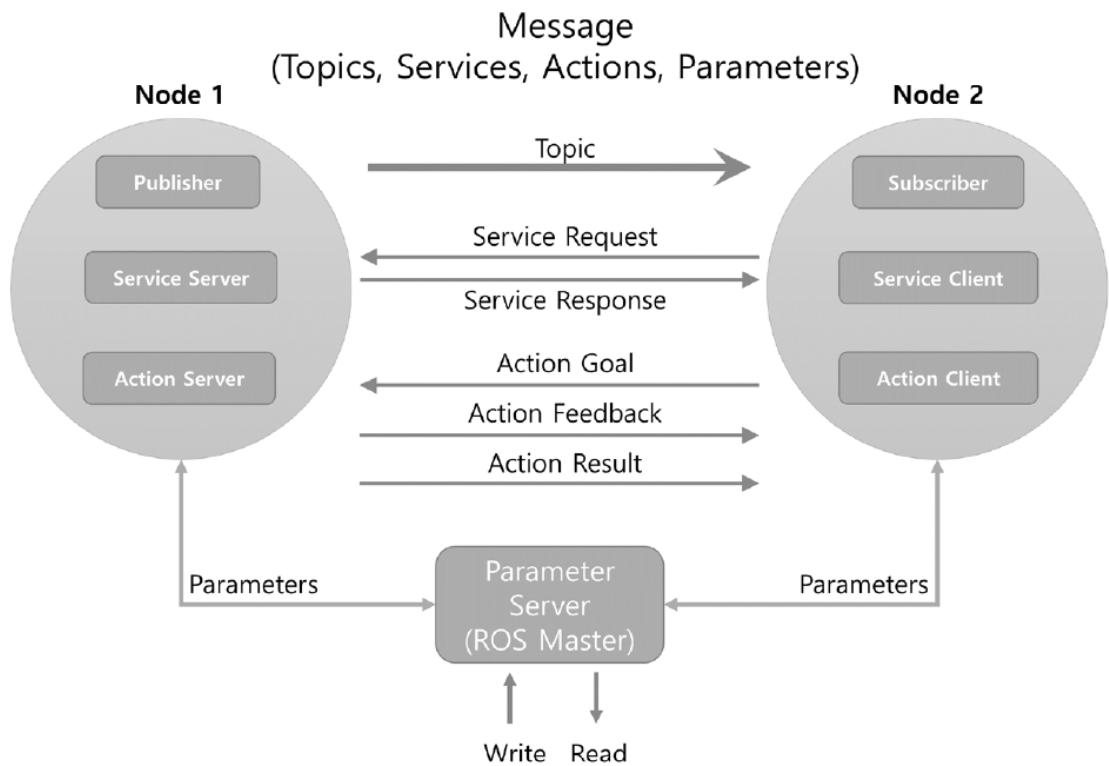


Figura 3.40: Comunicación de mensajes entre nodos [26]

3.4.7.3. Nivel Comunitario

El ecosistema ROS consta de decenas de miles de usuarios en todo el mundo que trabajan en dominios que van desde proyectos de pasatiempos de mesa hasta grandes sistemas de automatización industrial. Una característica importante de ROS es la comunidad que comparte software y código, lo que convierte a ROS en una de las comunidades de robots más grandes. Hay diferentes formas de obtener recursos ROS, las cuales son:

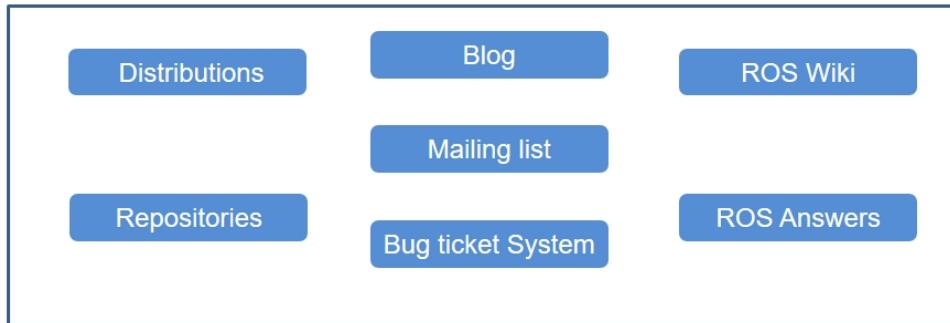


Figura 3.41: Nivel comunitario de ROS [34]

- **Distribuciones:** Las distribuciones ROS son colecciones de stacks versionadas que se pueden instalar. Las distribuciones juegan un papel similar a las distribuciones de Linux: facilitan la instalación de una colección de software y también mantienen versiones consistentes en un conjunto de software (<http://wiki.ros.org/Distributions>).
- **Repositorios:** ROS se basa en una red federada de repositorios de código, donde diferentes instituciones pueden desarrollar y lanzar sus propios componentes de software de robot.
- **El ROS Wiki:** La Wiki de la comunidad ROS es el foro principal para documentar información sobre ROS. Cualquiera puede registrarse para obtener una cuenta y contribuir con su propia documentación, proporcionar correcciones o actualizaciones, escribir tutoriales y más (<http://wiki.ros.org/>).
- **Mailing Lists:** Listas de correo de usuarios ROS
- **Answers:** Página web donde los usuarios comparten preguntas, respuestas y comentarios (<https://answers.ros.org/>).
- **Blog ROS:** Son las noticias de la comunidad ROS (www.ros.org/news/).
- **ROS Discourse:** Es el foro de discusión de la comunidad ROS (<https://discourse.ros.org/>). No es para temas técnicos específicos, sino mas bien para anuncios y noticias más amplias.

3.4.8. Herramientas

Existen varias herramientas que pueden ayudar a la hora de usar ROS. Se debe tener en cuenta estas herramientas GUI como complementarias a las herramientas de línea de comandos. Hay una gran cantidad de herramientas ROS, incluidas las herramientas que los usuarios de ROS también han lanzado personalmente. En otras palabras, las herramientas que discutiremos en este capítulo, no procesan directamente una función en ROS, pero son complementarias y muy útiles para programar en el.

3.4.8.1. Launch Files

En un proyecto ROS, es posible que se desee ejecutar varios nodos ROS al mismo tiempo para realizar sistemas más complejos. ROS tiene una herramienta llamada ros-launch que permite a los usuarios ejecutar numerosos nodos, establecer parámetros de configuración de cada nodo, renombrar los nombres de temas predeterminados e incluso cambiar el nombre del nodo. El propósito de esto es configurar fácilmente el sistema global.

3.4.8.2. RQT Graph

Proporciona la visualización de un sistema ROS, mostrando los nodos y las conexiones entre ellos que permite depurar y comprender el sistema en ejecución y cómo está estructurado.

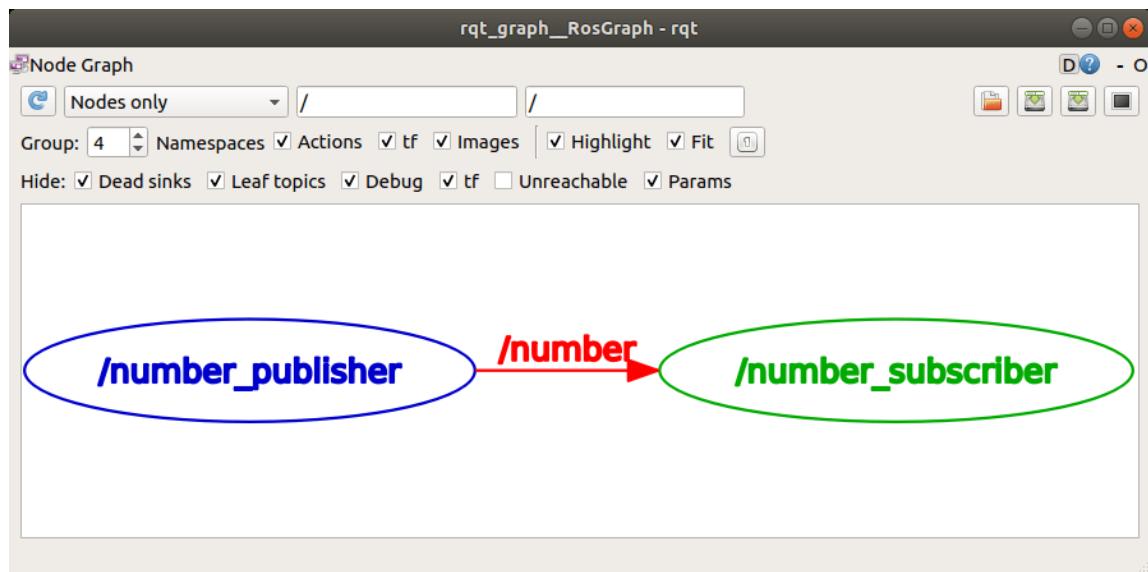


Figura 3.42: Visualización de la comunicación entre un nodo publisher (azul) y el nodo subscriber (verde) a travez del tema (rojo) number en RQT Graph.

3.4.8.3. RQT Rosbag

Rosbag es un conjunto de herramientas para grabar y reproducir datos de temas ROS. Los datos se almacenan en archivos de bag y hay disponibles herramientas de línea de comandos para trabajar con bag. Rosbag evita la deserialización y reserialización de mensajes y sus principales herramientas son:

- **rosbag info:** Muestra el contenido de los archivos de la bolsa, como temas grabados, hora de inicio y finalización, número de mensajes, frecuencia y estadísticas de compresión
- **rosbag record:** Escribe el contenido de todos los mensajes publicados sobre esos temas que queremos registrar y la información se almacena en un archivo .bag.
- **rosbag play:** Lee un archivo de bolsa y publica la información sobre temas ROS de forma sincronizada en el tiempo. El sistema ROS puede utilizar esta información como si fuera en tiempo real.

3.4.8.4. RQT reconfigure

Adaptación del paquete “Dynamic reconfigure” que permite modificar en tiempo real todos los parámetros de los nodos, para realizar modificaciones rápidas. La configuración de los parámetros se puede exportar para utilizarla en el futuro.

3.4.8.5. MoveIt

Software basado en la planificación del movimiento, que considera prospección 3D, cinemática, control y navegación. Proporciona una plataforma para el desarrollo de aplicaciones robóticas, evalúa nuevos diseños de robots y productos para la construcción robótica integrado para aplicaciones industriales, I + D, etc.



Figura 3.43: Logo Proyecto MoveIt

3.4.8.6. Gazebo

Es un simulador virtual que brinda la posibilidad de realizar simulaciones de algoritmos, diseños de robots y ejecutar pruebas dentro de escenarios reales con precisión y eficiencia. Proporciona un motor de física robusto, gráficos de alta calidad e interfaces gráficas.



Figura 3.44: Logo Gazebo

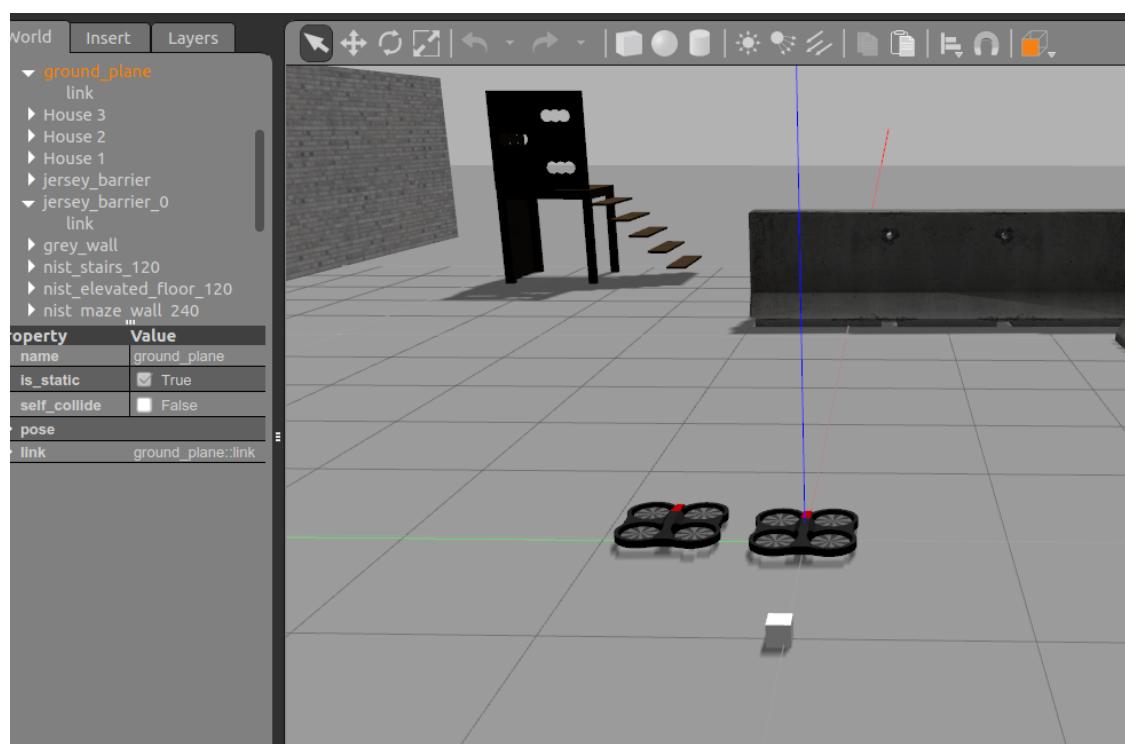


Figura 3.45: Simulación de dron en Gazebo

3.5. Visualización

3.5.1. Interfaz de visualización gráfica: ROS visualization (RViz)

RViz es la abreviación de ROS visualization. Es un entorno de visualización 3D de uso general para robots, sensores y algoritmos. Permite visualizar mapas, robots, objetos, datos láser, imágenes de cámaras, nubes de puntos y marcadores. Como la mayoría de las herramientas ROS, se puede utilizar para cualquier robot y configurar rápidamente para una aplicación en particular.

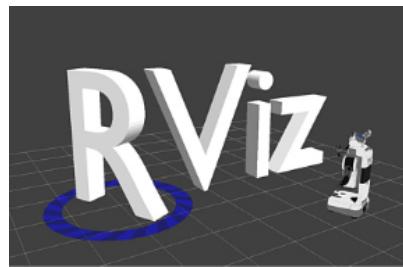


Figura 3.46: Logo de RViz

RViz proporciona una interfaz sencilla para elegir la información que queremos que se muestre. La figura (3.47) muestra un ejemplo de la interfaz RViz. Las display es algo que dibuja algo en el visualización 3D de RViz y probablemente tiene algunas opciones disponibles en la lista de pantallas. Un ejemplo es una nube de puntos, el estado del robot, la modelación de partes mecanicas del robot, coordenadas tf, sensores, cámaras, dimensiones de malla, etc.

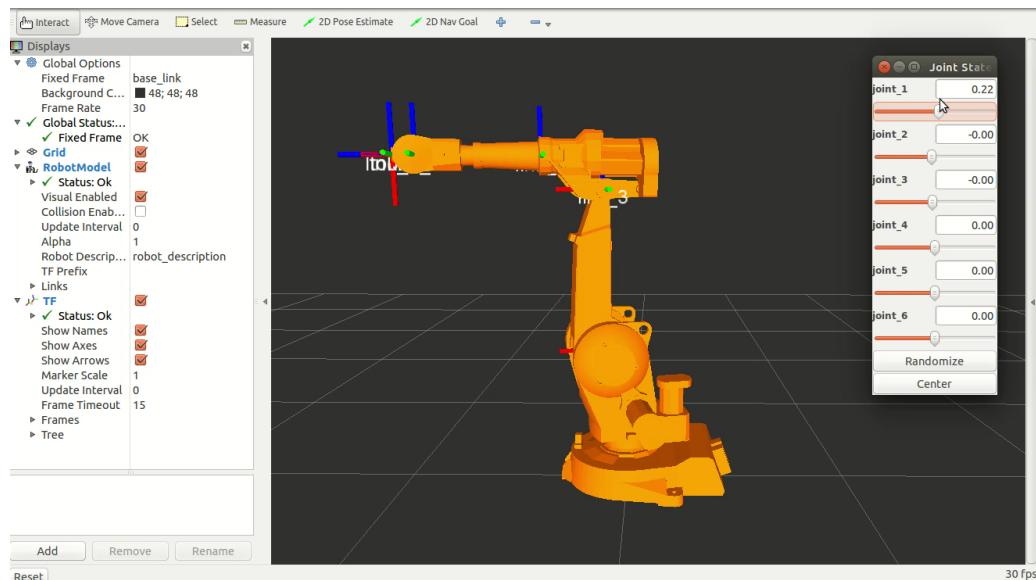


Figura 3.47: ABB IRB 2400 con publicador de estado de juntas en RViz [35]

3.5.2. Paquete tf

Un marco de coordenadas o frame es un concepto importante en ROS. Cualquier robot puede tener varios componentes, como un láser, una cámara, un sonar o brazos, y pueden tener cada uno un marco de coordenadas adjunto. Muchos algoritmos ROS requieren realizar un seguimiento de todos estos marcos de coordenadas.

Tf son las siglas en inglés de marco de transformadas (Transform frames) y es una de las librerías fundamentales de ROS. Tf está diseñada para proporcionar una forma estándar de realizar un seguimiento de los marcos de coordenadas y transformar los datos dentro de todo el sistema a lo largo del tiempo. El paquete tf puede rastrear y mantener la relación entre múltiples marcos de coordenadas. Su función es proporcionar herramientas y funciones para definir todos los marcos de coordenadas de nuestro robot y transformar datos de un marco a otro, como por ejemplo puntos o vectores.

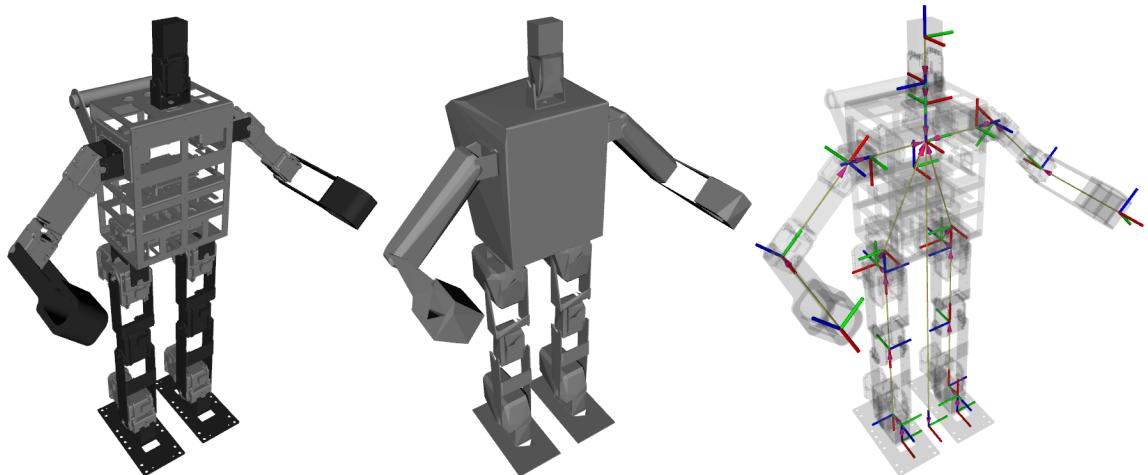


Figura 3.48: Tres visualizaciones diferentes del modelo Minibot URDF. Visualización en RViz, modelo de colisiones y estructura tf [36].

Un ejemplo para explicar las coordenadas tf, es el robot mostrado en la figura (3.50), que tiene una base móvil con un láser montado encima. El objetivo de este robot es evitar obstáculos para no colisionar, como una pared. En referencia al robot, se definen dos marcos de coordenadas: uno correspondiente al punto central de la base móvil del robot llamado "base_link" y otro para el punto central del láser que está montado en la parte superior de la base móvil llamado "base_laser".

El láser recopila datos en forma de distancias desde el punto central del láser a una pared, es decir, el láser tiene datos de distancia en el marco de coordenadas "base_laser". Para evitar obstáculos se debe mover la base móvil, por lo que los datos del láser se deben transformar al sistema de referencia "base_link", en otras palabras, se tiene que definir una relación entre los marcos de coordenadas "base_laser" y "base_link".

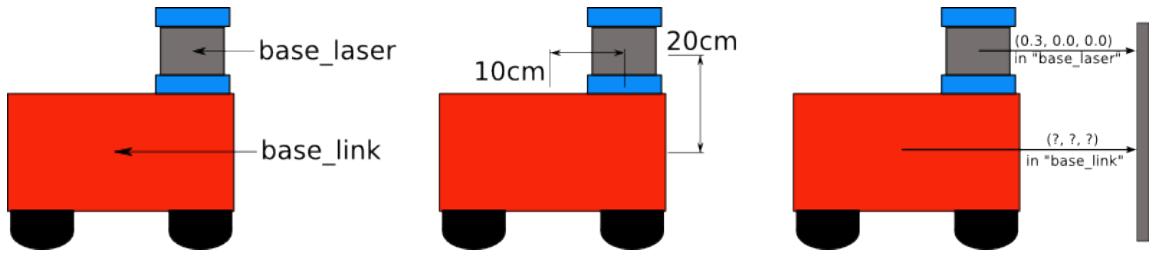


Figura 3.49: Ejemplo robot simple compuesto por base móvil y un láser

Esta relación entre los marcos de referencia son desplazamiento de traslación y rotacional entre los marcos 'base_laser' y "base_link". Gestionar esta relación en todo momento en una simulación del robot, es decir, almacenar y aplicar los desplazamientos adecuados entre los frames en cada momento, se convierte en un verdadero problema a medida que aumenta el número de frames de coordenadas. Afortunadamente, sin embargo, no se tiene que hacer este trabajo ya que en su lugar, se define la relación entre "base_link" y "base_laser" una sola vez usando el paquete tf que gestiona la transformación entre los marcos de coordenadas automáticamente.

Para definir y almacenar la relación entre los marcos "base_link" y "base_laser" usando tf, se necesita crear a un árbol de transformación. Conceptualmente, cada nodo en el árbol de transformación corresponde a un marco de coordenadas y cada borde corresponde a la transformación que debe aplicarse para pasar del nodo actual a su hijo. Tf usa una estructura de árbol para garantizar que haya un solo recorrido que vincule dos marcos de coordenadas cualesquiera y asume que todos los bordes del árbol se dirigen desde los nodos principales a los secundarios.

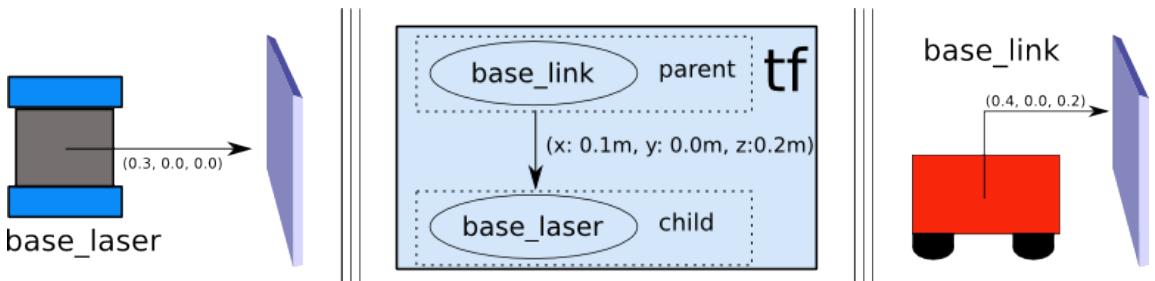


Figura 3.50: Láser recolectando datos respecto a "base_laser", árbol de transformación y datos del láser transformados a marco "base_link".

Tf ofrece una serie de herramientas o aplicaciones que facilitan al usuario la visualización del estado de las transformadas como:

- **tf_monitor:** Imprime la información sobre el árbol de transformación actual.
- **tf_echo:** Imprime información sobre la transformación entre dos frames.
- **tf_tree:** Crea un gráfico visual del árbol de transformación (en formato pdf).

La figura (3.51) muestra un ejemplo de un árbol de transformada de un robot haciendo uso de la herramienta rqt_tf_tree y la figura (3.52) muestra otro ejemplo de los frame de un robot.

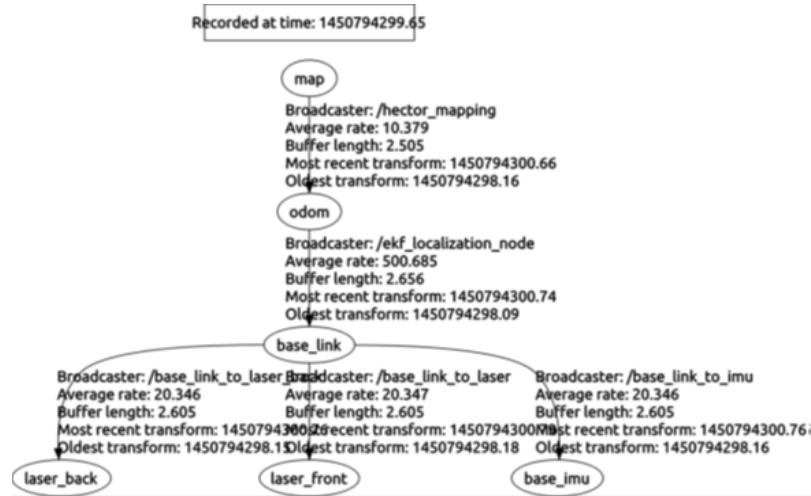


Figura 3.51: Frames de un sistema visualizado mediante tf_tree

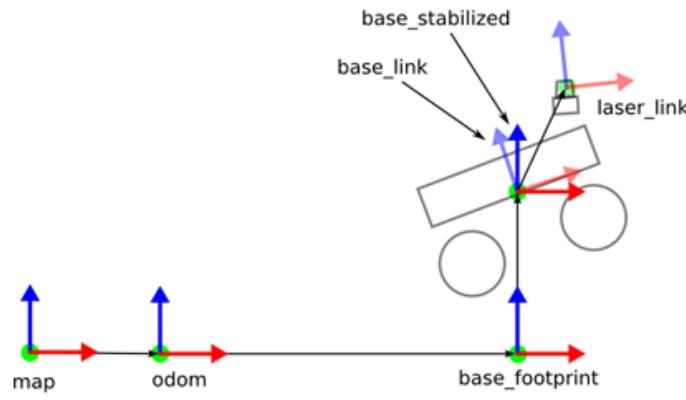


Figura 3.52: Figura que contiene los marcos tf más utilizados [37]

Resumiendo esta sección, TF es una biblioteca con las siguientes características:

- Es una herramienta para realizar un seguimiento de los marcos de coordenadas a lo largo del tiempo.
- Mantiene la relación entre los marcos de coordenadas en una estructura de árbol almacenada en el tiempo
- Permite al usuario transformar puntos y vectores entre marcos de coordenadas en cualquier momento deseado
- Es implementado como modelo de publisher/subscriber en los temas con nombres /tf y /tf_statics

3.5.3. Unified Robot Description Format (URDF)

3.5.3.1. Introducción URDF

En ROS, es posible visualizar un modelo de un robot mediante el uso de archivos de formato de descripción de robot unificado (URDF). Sin embargo, solo aquellos robots que tienen eslabones rígidos conectados mediante articulaciones pueden ser descritos mediante modelos URDF. Se puede usar un modelo URDF para calcular la cinemática, agregar nuevos marcos de coordenadas y moverlos de acuerdo con los valores del codificador del robot. Además, puede incluir otras propiedades físicas como inercia, colisiones, dinámica de articulaciones, etc.

Los archivos URDF están basados en lenguaje XML y como tal, están compuestos por etiquetas XML especiales que pueden ser leídas para extracción de información. La lectura del archivo para extraer la información importante del modelo se denomina parsing y al programa o función que lo realiza, parser.

La descripción del modelo consiste básicamente en unir dos conjuntos: el conjunto de enlaces (link) y el conjunto de uniones (joint). La forma de construir y visualizar un modelo de robot en URDF es escribir y compilar el archivo URDF. Una vez que se crea la representación 3D de un robot mediante el uso de un archivo URDF, es posible utilizar dicha representación para simular el movimiento del robot. Para ello, el usuario debe publicar las condiciones del robot en tf, utilizando un nodo (o nodos) para publicar la información de transformación.

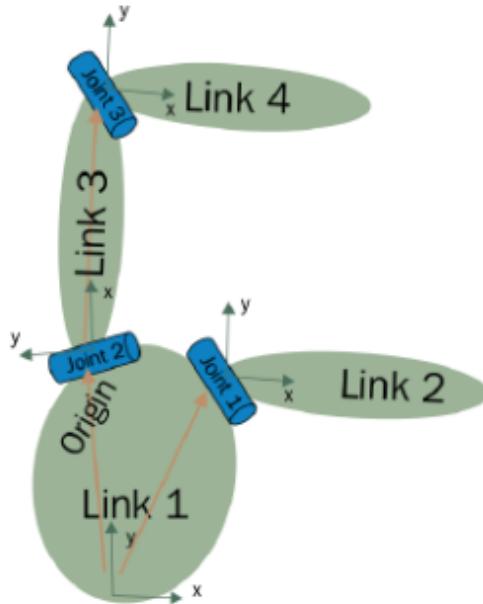


Figura 3.53: Elementos basicos de visualization URDF: links y joints [38].

3.5.3.2. Links

Los eslabones o enlaces (conjunto link) describen la parte física rígida del robot y permite especificar sus propiedades, como tamaño, forma, color o una malla 3D compleja importada. Consta de tres elementos:

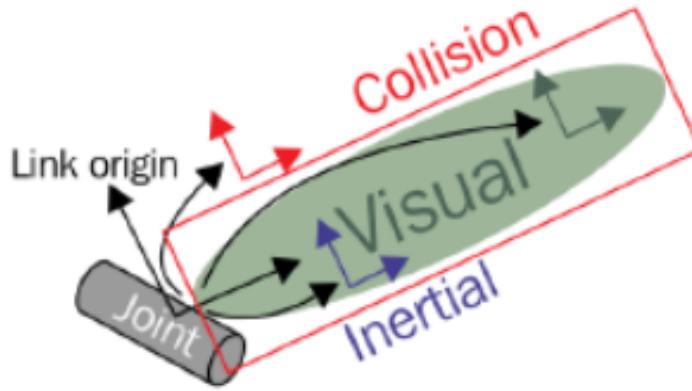


Figura 3.54: Representación de la visual, inercia y colisión de un link con sus respectivos marcos de referencia para URDF [38].

1. **Inertial:** Especifica las propiedades iniciales del link (posición del centro de masa, la masa y la matriz de inercia).
2. **Visual:** Este elemento especifica como debe estar representado el link. Es posible usar sólidos simples como una esfera o un cilindro, o hacer representaciones más trabajadas como mallados o nube de puntos. Se pueden definir los colores e importar texturas.
3. **Collision:** Describe las propiedades de colisión de los links. Pueden ser una forma diferente a la visual, como una figura más simple que sea representativa, a fin de reducir los tiempos de cálculo en las simulaciones.

```
link
  <link name="nombre del eslabón">
    <inertial>.....</inertial>
    <visual> .....</visual>
    <collision>.....</collision>
  </link>
```

Figura 3.55: Descripción de un link en código URDF.

3.5.3.3. Joints

Las articulaciones o uniones (conjunto joint) indican la relación entre los distintos eslabones del robot. Describen la cinemática y dinámica de cada articulación, además de especificar los límites de colisión del robot. Una articulación queda definida cuando se especifican los eslabones 2 eslabones unidos a ella: el primero es llamado parent (parent) y el segundo es llamado child (child).

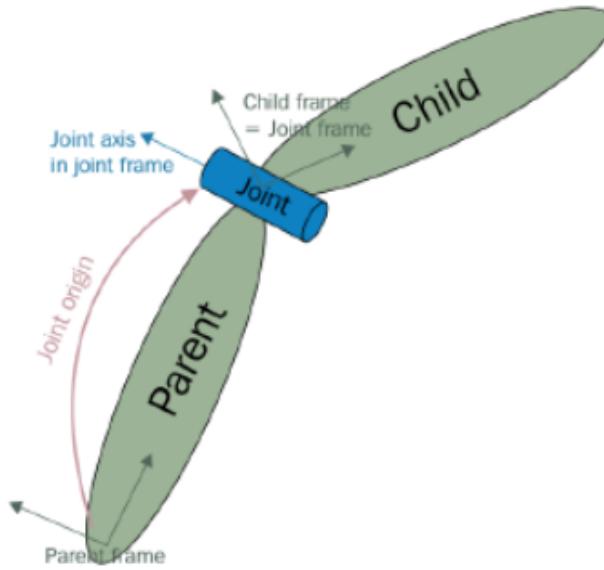


Figura 3.56: Representación gráfica de un joint y su relación padre-hijo con los link [38]

Los tipos de articulaciones son:

1. **Revolute:** Permite que dos sólidos giren alrededor de un eje común y tiene un giro limitado especificado por un límite superior y uno inferior.
2. **Continuous:** Se trata de una articulación de Revolute pero sin límites..
3. **Prismatic:** Una junta que se deslizarse a lo largo de un eje. Tiene un alcance acotado por un límite inferior y uno superior.
4. **Fixed:** No es realmente una articulación porque impide el movimiento. La unión no tiene ningún grado de libertad. Este tipo de articulación no requiere de ninguno de los elementos que describe la articulación..
5. **Floating:** Esta unión mantiene libres todos los grados de libertad..
6. **Planar:** Articulación que permite el movimiento en un plano.

```

joint

<joint name="nombre de la articulacion">
  <parent link="link1"/>
  <child link="link2"/>
  <calibration .... />
  <dynamics damping .... />
  <limit effort .... />
</joint>

```

Figura 3.57: Descripción de un joint en código URDF

Cada conjunto de uniones (joint) está compuesto por hasta nueve elementos:

1. **Origin:** Es la posición donde se encuentra la unión entre el sólido padre (Parent) y el sólido hijo (child), en referencia al parent.
2. **Parent:** El objeto parent en una unión.
3. **Child:** En objeto child en una unión.
4. **Axis:** Eje del articulacion en la referencia global.
5. **Calibration:** Es un elemento opcional. Sirve para determinar la posición absoluta de la articulación.
6. **Dynamics:** Es un elemento opcional. Define la amortiguación y la fricción de la articulación.
7. **Límite:** Sólo obligatoria en las articulaciones de revolución y en las prismáticas. Define el límite inferior y superior de la articulación, el esfuerzo máximo que puede aplicar y la velocidad máxima de esta.
8. **Mimic:** Es un elemento opcional. Se usa para especificar que una articulación imita a otra articulación ya existente.
9. **Safety controller:** Es un elemento opcional. Específica para qué valores debe empezar a limitar la posición o la velocidad de una articulación en función de los límites establecidos en el tag <límit>.

3.5.4. Herramientas complementarias

Algunos paquetes y herramientas utiles para la interaccion con modelos URDF son los siguientes:

- **Joint_state_Publisher**: Este paquete contiene un nodo del mismo nombre que lee la descripción del modelo del robot, encuentra todas las articulaciones (joints) y publica los valore articulares para todas las articulaciones moviles usando sliders.
- **robot_state_Publisher**: Este paquete lee el estado de las articulaciones del robot y publica las posiciones y orientaciones 3D de cada eslabón usando la cinemática obtenida a partir del URDF. La posición 3D del robot se publica como una transformación (tf) que define las relaciones entre los sistemas de coordenadas.
- **xacro**: Significa Xml mACROs y es un formato que permite utilizar variables y otros add-ons para la generación de modelos complejos en formato URDF. De esta forma los modelos pueden ser más fáciles de entender y más mantenibles.

La herramienta urdf_to_graphviz sirve para para obtener un diagrama graphviz de su archivo URDF, tal como se aprecia en la figura (3.58).

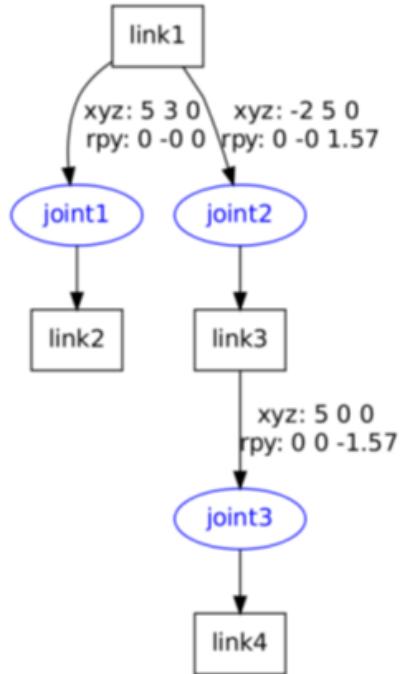


Figura 3.58: Ejemplo de la representación gráfica URDF de un robot por medio de la herramienta urdf_to_graphviz [38]

3.6. ADAMS (Automated Dynamic Analysis of Mechanical Systems)

3.6.1. Historia

ADAMS significa Análisis dinámico automático de sistemas mecánicos y fue desarrollado originalmente por Mechanical Dynamics Inc. (MDI). MDI fue formado por investigadores / desarrolladores del código ADAMS original en la Universidad de Michigan, Ann Arbor, MI, Estados Unidos. Más tarde, fue absorbida por McNeil Schindler Corp (MSC) en 2002.

En el núcleo de ADAMS hay un código de gran desplazamiento llamado ADAMS / Solver, que resuelve ecuaciones numéricas no lineales. En ese entonces los modelos se creaban en formato de texto y luego se enviaban a ADAMS / Solver.

A principios de los 90, se lanzó ADAMS / View, que permitió a los usuarios crear, simular y examinar resultados en un único entorno gráfico de usuario (GUI). Hoy, MSC produce muchos paquetes de análisis de ingeniería general como MSC.NASTRAN, MSC.PATRAN, MSC.DYTRAN, etc. y también paquetes que atienden a usuarios específicos de la industria como MSC.ADAMS / Car, MSC.ADAMS / Rail, MSC.ADAMS / Engine, etc.

3.6.2. Introducción

Adams es un programa para simulación dinámica y análisis de movimiento cinemático en múltiples cuerpos. Ayuda en el estudio de la dinámica de las partes móviles, como cargas y fuerzas que se distribuyen a lo largo de los sistemas mecánicos, para mejorar y optimizar el rendimiento de los productos. Permite crear y probar prototipos virtuales de los sistemas mecánicos en una fracción del tiempo y costo requerido para la estructura física y la prueba, incorporando la física real de forma simultánea a la resolución de ecuaciones de cinemática, estática, y la dinámica.

Un **sistema multicuerpo** es la modelización de un sistema mecánico como un conjunto de sólidos rígidos o flexibles conectados entre sí por un conjunto de uniones. Este conjunto forma un sistema físico cuya cinemática y dinámica se pueden describir con una serie de ecuaciones diferenciales y algebraicas.[\[39\]](#)

Excepto en el caso de sistemas mecánicos muy sencillos, la resolución de las ecuaciones que describen el movimiento de un sistema multicuerpo requiere con frecuencia la ayuda de programas informáticos de cálculo numérico. Por este motivo, la simulación y el análisis de sistemas multicuerpo están estrechamente ligados a disciplinas como el álgebra lineal y la programación.

Hoy en día existen diversos programas que se encargan de darle un aspecto más visual a todas las ecuaciones subyacentes por debajo en la resolución de este tipo de

sistemas, en nuestra tesis utilizaremos ADAMS, dado que se puede adquirir bajo una licencia estudiantil por un semestre, previa creación y validación de una cuenta universitaria.

Pese a que nuestra hipótesis nos guía a regirnos solo por software libre, esto preocupa sobretodo en la formulación, en la validación hemos intentado ocupar los mejores estándares para verificar los códigos propuestos.

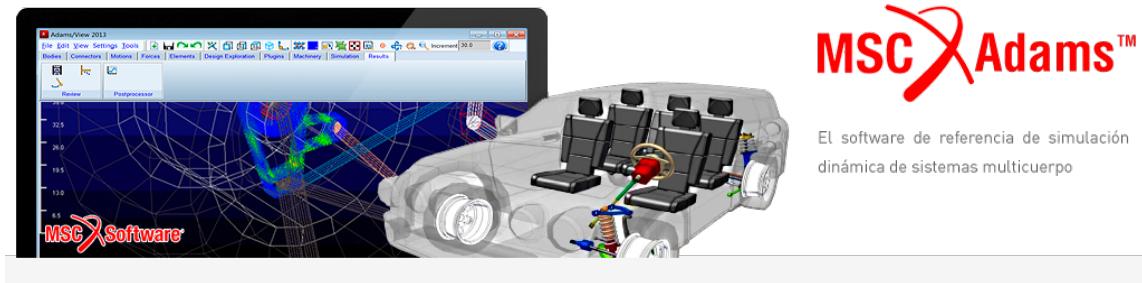


Figura 3.59: Imagen de referencia del software Adams

3.6.3. Descripción del ambiente de ADAMS

La versión actualizada de ADAMS trabajada durante la tesis es la 2020 FP1, y la descripción del ambiente se hará bajo el sistema operativo Windows dada su extensión, y con el fin de llegar a la mayor cantidad de personas.

3.6.3.1. Inicio de un nuevo modelo y su configuración

Al iniciar ADAMS lo primero que encontraremos será el diálogo de bienvenida mostrado en la figura (3.60), la ventana nos da tres opciones, New Model para iniciar un nuevo modelo; Existing Model para abrir la base de datos de un modelo existente (los archivos ADAMS tienen extensión .bin) y finalmente la opción exit, para cerrar ADAMS.

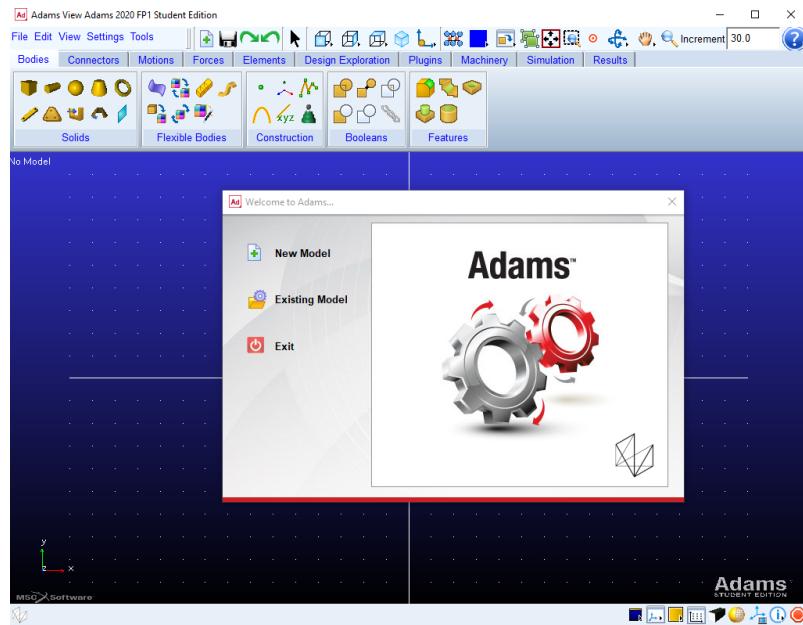


Figura 3.60: Ventana de bienvenida al iniciar ADAMS

En el fondo de la ventana de bienvenida, ya podemos observar con antelación el GUI de ADAMS, pero no podremos interactuar con él, hasta terminar la configuración inicial del modelo. Para iniciar un nuevo modelo, damos clic en **New Model**, y se deplegará la ventana de configuración inicial que se muestra en la figura (3.61). Las configuraciones posibles son:

- **Model name:** El nombre del modelo
- **Gravity:** Aquí se configura la fuerza de gravedad, tenemos tres posibilidades:
 - Earth Normal (-Global Y): Gravedad normal de $1G\left(\frac{m}{s^2}\right)$ en dirección -Y. Es posible cambiar este valor en cualquier momento dentro del modelo.
 - No gravity: El modelo no contara con fuerza de gravedad.
 - Other: Permite que uno ajuste la gravedad como se desee. El cuadro de diálogo Configuración de gravedad aparece después de terminar la configuración inicial.
- **Units:** El sistema de medición empleado en el modelo, ADAMS permite varias posibilidades:
 - MMKS: Largo en milímetros, masa en kilogramos y fuerza en Newton.
 - MKS: Largo en metros, masa en kilogramos y fuerza en Newton.
 - CGS: Largo en centímetros, masa en gramos y fuerza en Dyne.
 - IPS: Largo en pulgadas, masa en slug y fuerza en libras

- **Working Directory:** El directorio donde se guardaran todos los datos y archivos del modelo.

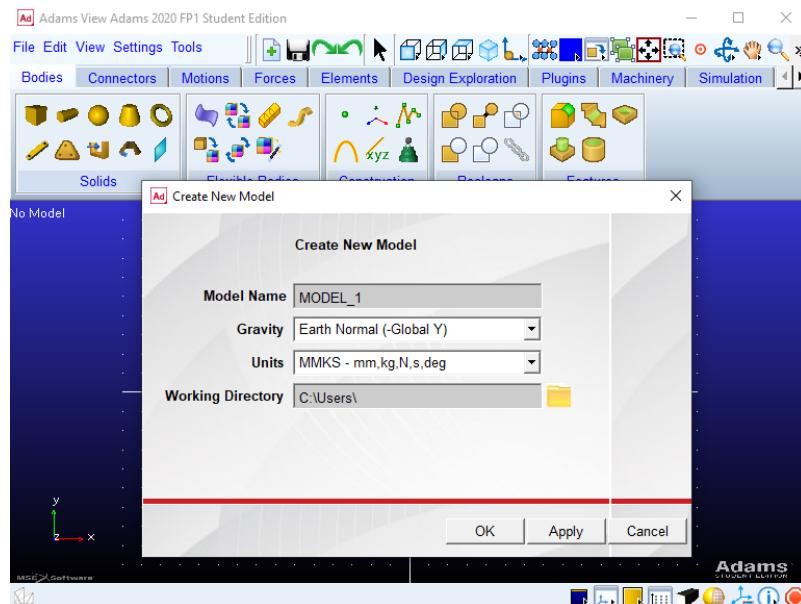


Figura 3.61: Ventana de bienvenida al iniciar ADAMS

Luego de darle una configuración inicial a nuestro modelo, inicia la ventana principal de ADAMS, la cual se observa en la figura (3.62). A continuación se detallan sus principales puntos de interés: (NO ME GUSTA COMO QUEDÓ, PERO CORREGIRÉ DESPUÉS, DEBERÍAN IR LOS NOMBRES EN VEZ DE LOS NÚMEROS)

1. Menu principal
2. Barra de herramientas principal
3. Árbol del modelo
4. Barra de herramientas de estado

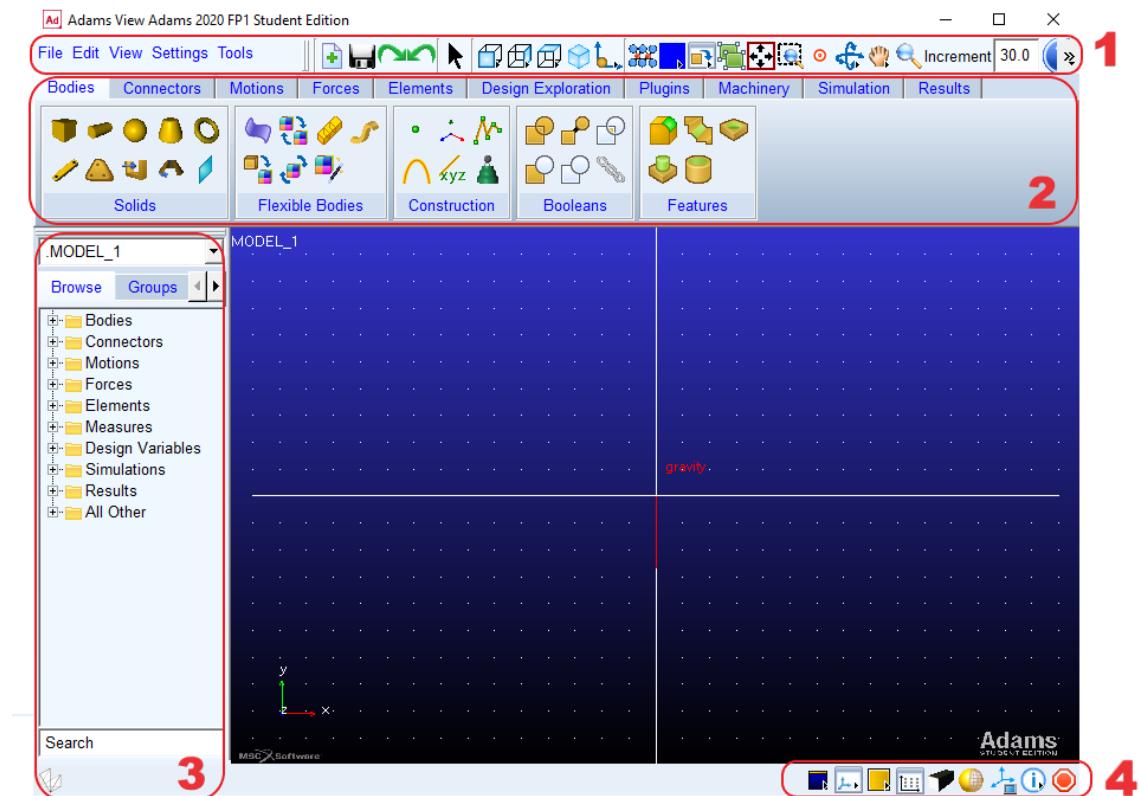


Figura 3.62: Interfaz de vista en ADAMS

3.6.4. Modelo y jerarquía de los datos en ADAMS

Una base de datos de modelado almacena toda la información sobre cualquier cosa que uno hace en el entorno de ADAMS. La base de datos de modelado ADAMS es una base de datos jerárquica. Cada objeto en la base de datos tiene un objeto que lo posee, llamado su padre, y muchos objetos poseen otros objetos, llamados sus hijos. El nivel superior de los objetos de la base de datos son modelos, vistas, gráficos y bibliotecas que contienen elementos tales como cuadros de diálogo.

En la figura 3.63 se muestra la jerarquía de una base de datos llamada ejemplo_1 que contiene un modelo y una gráfica del modelo:

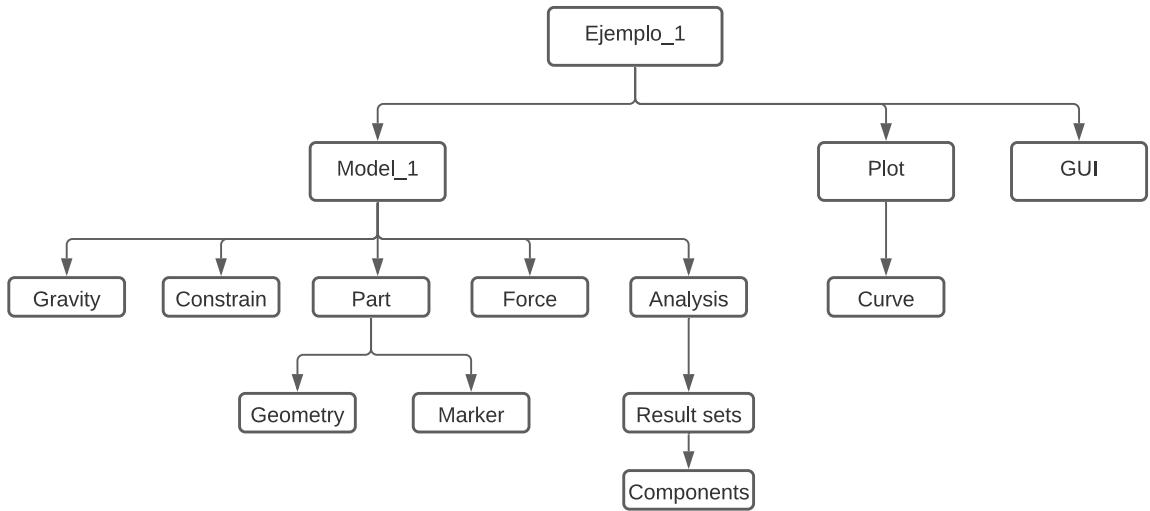


Figura 3.63: Jerarquía de datos en ADAMS

3.6.4.1. Convención de nombres en ADAMS

Los nombres para cada uno de los objetos en la base de datos, siguen la misma estructura jerárquica de los datos, en la figura 3.64 se observan los nombres otorgados por adams a un modelo como el del ejemplo anterior 3.63.

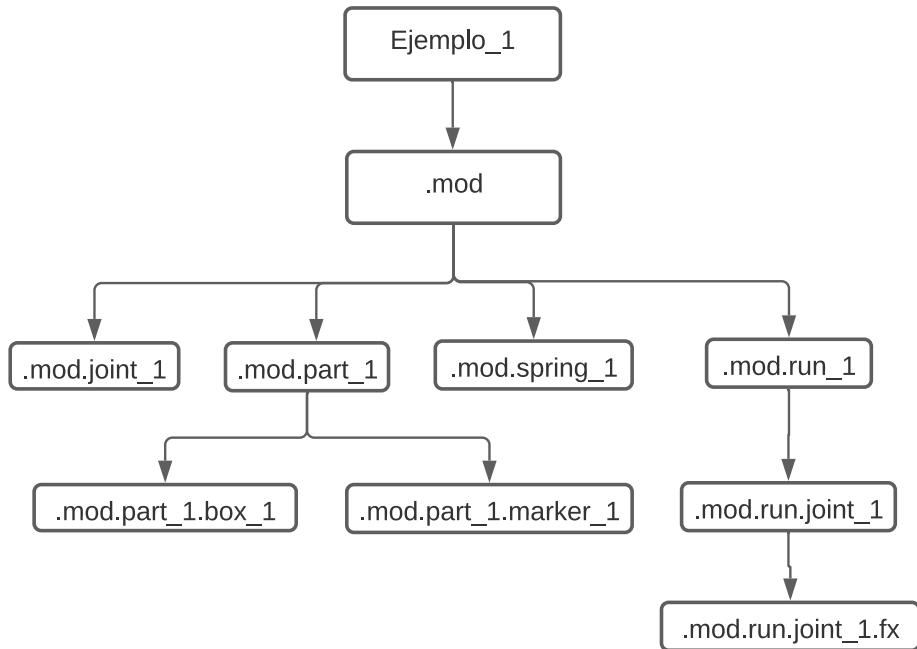


Figura 3.64: Jerarquía de nombres en ADAMS

3.6.4.2. Sistemas de coordenadas

Un sistema de coordenadas es esencial para medir y definir la cinemática y dinámica de cualquier objeto, ADAMS inicia por defecto con un sistema cartesiano, pero es posible elegir otros sistemas como cilíndrico y esférico.

En ADAMS encontramos dos tipos de sistemas de coordenadas:

1. Global coordinate system (GCS)

- Unido de forma rígida a la tierra (ground part).
- Define de forma absoluta el punto (0,0,0) del modelo y provee los ejes de referencia para crear futuros sistema locales de coordenadas.

2. Local coordinate system (LCS) existen dos tipos:

- Sistema de coordenadas de una pieza (PCS).
- Los Markers que generemos.

3.6.4.2.1. Sistema de coordendas de una Pieza (PSC) Estos sistemas de coordenadas se generan automáticamente al crear una pieza, y solo existe uno por parte. La orientación y posición se definen respecto del marco de referencia global (GCS).

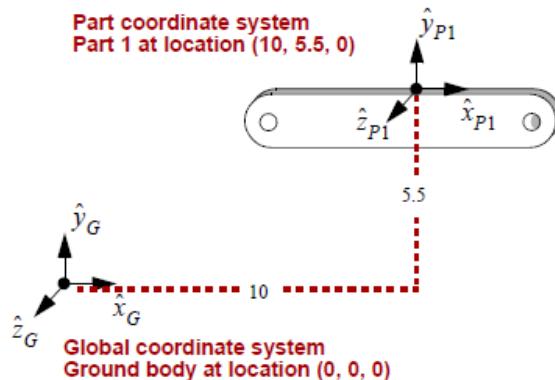


Figura 3.65: Sistema de coordendas de una Pieza (PSC) [40]

3.6.4.2.2. Sistema de coordendas de un Marker Estos sistemas de coordenadas se adhieren a una pieza y se desplazan con ella. Pueden existir muchos para una misma pieza y su posición y orientación se definen por defecto respecto al GSC, pero es posible cambiarla a un PSC.

Se usan generalmente para definir posiciones o direcciones en las cuales queremos general algo, ya sea una restricción o una medición entre otros.

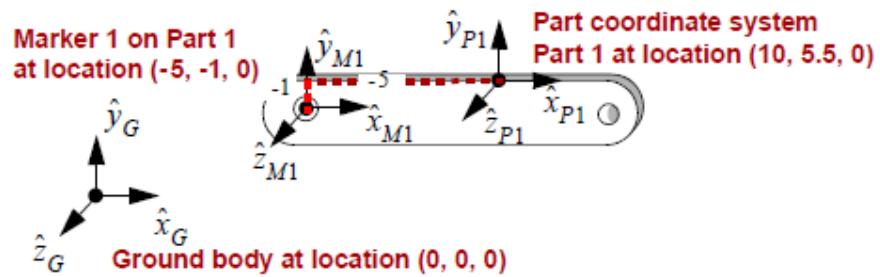


Figura 3.66: Sistema de coordenadas de un Marker[40]

3.6.4.3. Parts

Definen un cuerpo que puede moverse relativo a otro y ademas tiene las siguientes propiedades:

- Masa
- Inercia
- Posición y orientación inicial
- Velocidad inicial

Se utiliza para agregar gráficos y mejorar la visualización de una pieza usando propiedades como:

- Largo
- Radio
- Ancho

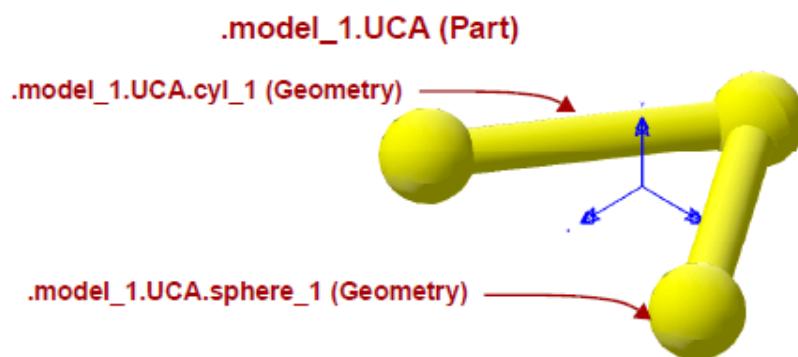


Figura 3.67: Pieza con varias geometrias en ADAMS [40]

Existen dos tipos de piezas en adams:

1. Cuerpos rígidos

- Son partes móviles
- Poseen masa e inercia
- No se pueden deformar

2. Cuerpos flexibles

- Son partes móviles
- Poseen masa e inercia
- Puede doblarse cuando se aplican fuerzas a ellas

La pieza **ground** es la única parte del modelo que debe permanecer estacionaria en todo momento. Adams / View crea la parte del suelo automáticamente cuando se crea un modelo. El usuario también puede definir un nuevo o existente parte como parte de tierra. La parte del suelo no tiene propiedades de masa ni velocidades iniciales y no suma grados de libertad en el modelo.

3.6.4.4. Constraints / Joints

Son restricciones relativas al movimiento entre dos piezas y se representan en ADAMS como conexiones ideales. Estas restricciones disminuyen los grados de libertad en rotación y translación de una pieza, la figura (3.68) presenta el ejemplo de una unión de revoluta.

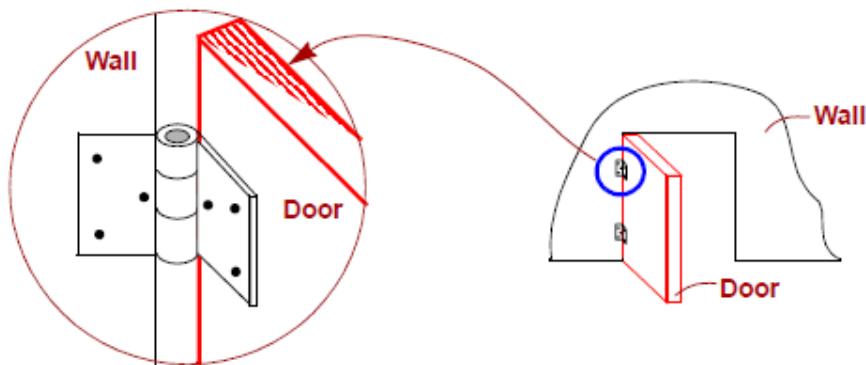


Figura 3.68: Unión revoluta, de una puerta con su marco [40]

3.6.4.5. Measures

Las mediciones en ADAMS representan datos que deseamos medir mientras se desarrolla una simulación, estos datos pueden ser:

- Desplazamiento, velocidad y aceleración de un punto en una pieza
- Fuerzas en una unión
- Ángulo entre dos piezas
- Otros datos derivados de funciones definidas por el usuario

Los datos son visualizados gráficamente durante la simulación y son guardados para su posterior análisis/exportación. La figura (3.69) presenta algunos tipos de mediciones posibles de hacer:

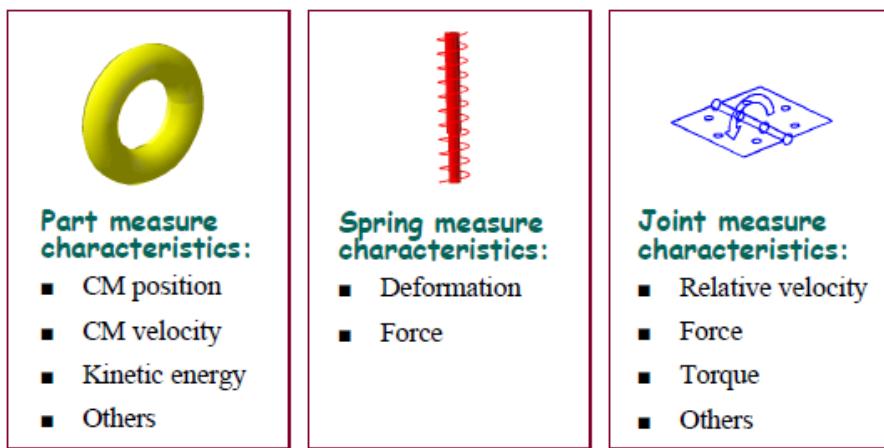


Figura 3.69: Mediciones según tipo de objeto en ADAMS [40]

3.6.5. Adams/Solver

ADAMS / Solver se utiliza para simular la evolución temporal de un sistema mecánico. En cada momento, ADAMS /Solver es capaz de indicar la posición, orientación y velocidades asociadas con cada parte del modelo. La posición y orientación de una pieza se controla mediante lo que se denomina coordenadas generalizadas. La elección de coordenadas generalizadas no es única, por ejemplo, se pueden elegir coordenadas cartesianas o coordenadas esféricas. Asimismo, se podría elegir entre coordenadas globales o relativas, en las que la configuración de una pieza (posición, orientación, velocidades) se define con respecto al origen u otra pieza, respectivamente. La observación clave es que una vez que se selecciona un conjunto de coordenadas generalizadas, se espera que defina de manera única la configuración de cada parte del sistema en una instancia de tiempo determinada.

En este contexto, en ADAMS / Solver, la posición de un cuerpo rígido está definida por tres coordenadas cartesianas x, y, z:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (3.1)$$

La orientación de un cuerpo rígido está definida por un conjunto de tres ángulos de Euler que corresponden a la secuencia de rotación 3-1-3: ψ, ϕ, θ , respectivamente. Estos tres ángulos se almacenan en una matriz:

$$\epsilon = \begin{bmatrix} \psi \\ \phi \\ \theta \end{bmatrix} \quad (3.2)$$

El conjunto de coordenadas generalizadas asociadas con el cuerpo rígido. I en ADAMS se denota en lo que sigue por:

$$q_i = \begin{bmatrix} P_i \\ \epsilon_i \end{bmatrix} \quad (3.3)$$

Con base en esta elección de coordenadas generalizadas, la velocidad longitudinal y angular del cuerpo se obtienen como:

$$u = \dot{P} \quad (3.4)$$

$$\omega = B\dot{\epsilon} \quad (3.5)$$

donde

$$B = \begin{bmatrix} \sin(\phi)\sin(\theta) & 0 & \cos(\phi) \\ \cos(\phi)\sin(\theta) & 0 & -\sin(\phi) \\ \cos(\phi) & 1 & 0 \end{bmatrix} \quad (3.6)$$

ADAMS plantea las ecuaciones de movimiento mediante la formulación de euler-lagrange:

$$\frac{d}{dt} \left(\frac{\delta K}{\delta \dot{q}} \right)^T - \left(\frac{\delta K}{\delta q} \right)^T + \Phi_q^T \lambda = Q \quad (3.7)$$

Donde K es la energía cinética definida por:

$$K = \frac{1}{2} u^T * M u + \frac{1}{2} \omega^T J \omega \quad (3.8)$$

El sistema de ecuaciones que rige el comportamiento de cualquier mecanismo en

ADAMS / Solver consiste en:

- Seis ecuaciones dinámicas de primer orden para cada parte (que relacionan las fuerzas y aceleraciones)
- Seis ecuaciones cinemáticas de primer orden para cada parte (que relacionan posiciones con velocidades)
- Una única restricción algebraica para cada restricción de movimiento
- Una única ecuación algebraica para cada componente de fuerza escalar
- Cualquier número de ecuaciones diferenciales algebraicas o de primer orden definidas por el usuario

El sistema de ecuaciones anterior esta formado por Ecuaciones diferenciales y algebraicas mixtas no lineales (DAE).

Existen varias formas de resolver este sistema de ecuaciones DAE en ADAMS. La forma mas común y el que usamos en nuestra tesis con el integrados GSTIF.

En general el sistema de ecuaciones es de la forma

$$G(q, q', t) = 0 \quad (3.9)$$

Que es un sistema de N ecuaciones con 2N incógnitas, para reducir el numero de incógnitas a N, la derivada en el tiempo de cada componente se aproxima en base al método de diferenciación hacia atrás (BDF o el metodo de euler alrevez) hasta que se cumplan los criterios de convergencia del corrector, o hasta que el corrector alcanza el número máximo de iteraciones, que está bajo su control.

Con el método de euler alreves se discretiza el sistema, todas las derivadas en el tiempo de primer orden del as ecuaciones de movimiento se discretizan para producir un conjunto de ecuaciones algebraicas no lineales, a estas se les suma las ecuaciones de las uniones y las fuerzas, este sistema se termina resolviendo mediante Newton-Rapson.

Capítulo 4

Modelación física y matemática

4.1. Desarrollo de la modelación y métodos

En primero lugar, con el objetivo de modelar la cinemática y la dinámica de un robot delta de manera correcta, se presentan 2 métodos que son programados y comparados en este texto. Estos dos métodos deben dar los mismos resultados si es que son programados correctamente. Se les llama método A y método B. Cada etapas de los dos métodos tienen el mismo objetivo, sin embargo, el planteamiento del problema cinemático y dinámico es tratado de forma diferente. Para comparar los dos métodos se crea un marco de referencia global en el robot delta. Por ende, en los códigos de los algoritmos es necesario que las salidas o resultados estén en correlación a este sistema de referencia global. La figura (4.1) muestra el diagrama de flujo de los dos métodos con el objetivo de cada etapa.



Figura 4.1: Flujo de trabajo para el Método A y B

En segundo lugar, se explica que es el espacio de trabajo de un robot delta y los parámetros que influyen en su volumen, forma, densidad espacial, etc. Para ello, se utiliza la modelación cinemática y se privilegia el método A sobre la B por razones en relación a el cálculo del jacobiano que se explican en dicha sección.

Por último, se presentan las trayectorias más comunes implementadas en el desarrollo de los robots. De todas las trayectorias presentadas se profundizan 2, las cuales se utilizan para simular el mecanismo del robot delta y comparar los resultados tanto de la modelación cinemática como de la modelación dinámica.

4.2. Nomenclatura y sistema de referencia global

En la figura (4.2) se muestra el sistema de referencia global XYZ y el orden de numeración de los ángulos $\theta_{i \in \{1,2,3\}}$, los cuales son utilizados para representar todos los resultados obtenidos en esta tesis.

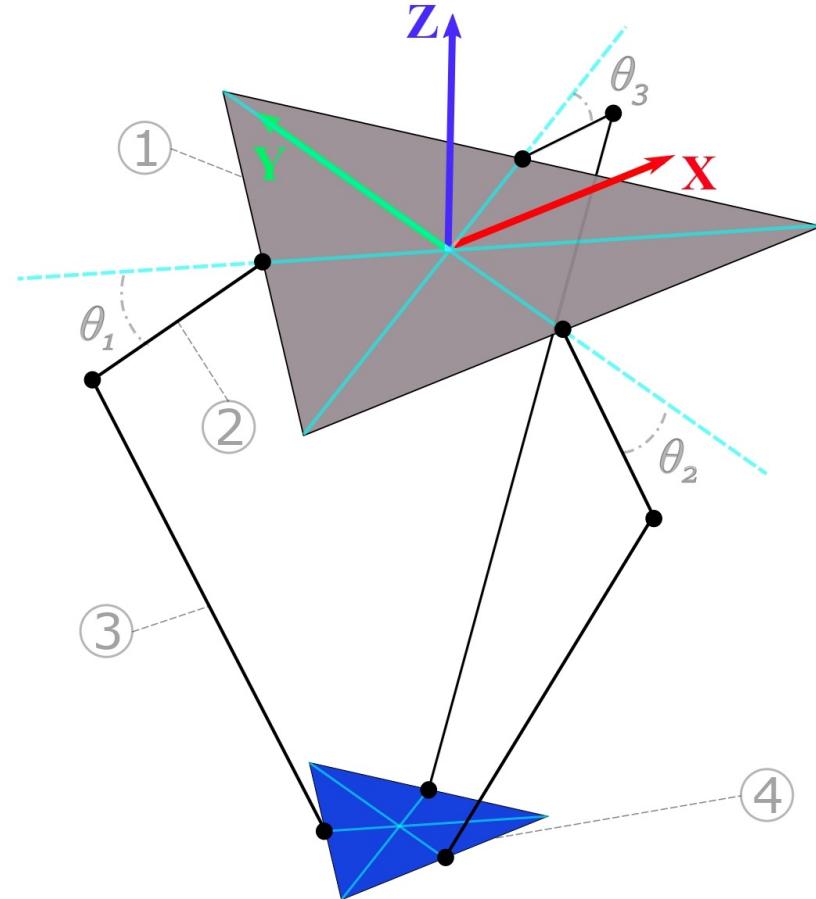


Figura 4.2: Sistema de referencia global XYZ y ángulos de los brazos $\theta_{i \in \{1,2,3\}}$.

La tabla (4.1) relaciona el nombre de las piezas mecánicas básicas del robot delta con su numeración en la figura (4.2).

Número	Pieza Mecánica
1	Base fija
2	Brazo
3	Antebrazo
4	Base Móvil

Tabla 4.1: Relación entre la numeración y piezas mecánicas de la figura (4.2).

4.3. Método A

4.3.1. Modelación cinemática de posición

En esta sección, se da a conocer una solución para calcular la cinemática inversa y cinemática directa de un robot paralelo tipo delta. Con el fin de solucionar el problema cinemático, se emplean las ideas y modelación del robot delta propuestas por Lois Rilo Antelo [41].

4.3.1.1. Nomenclatura de parámetros geométricos y sistema de referencia local

La figura (4.3) presenta el robot delta simplificado con el respectivo sistema de referencia local XYZ y el orden de numeración de los ángulos $\theta_{i \in \{1,2,3\}}$ para la solución de la cinemática de posición del método A.

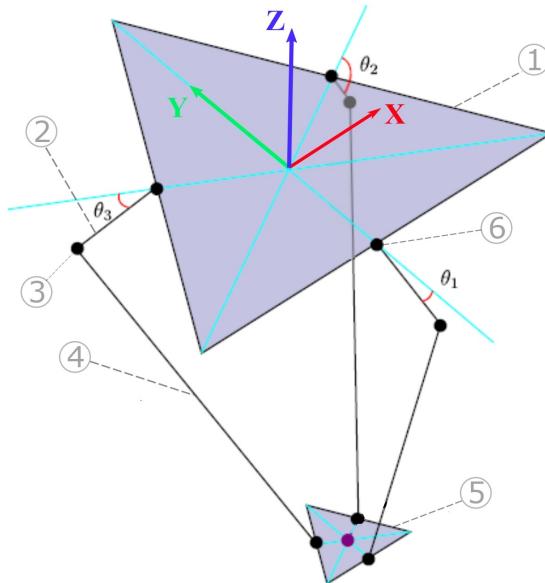


Figura 4.3: Sistema de referencia local para la cinemática de posición del método A [41].

La tabla (4.2) muestra la relación entre la numeración en la figura (4.3) y su nombre:

Número	Nombre
1	Base fija
2	Brazo
3	Junta esférica
4	Antebrazo
5	Efecto final
6	Actuador

Tabla 4.2: Relación entre la numeración y piezas mecánicas de la figura (4.3).

En la figura (4.3) se aprecia las piezas mecánicas básicas de un robot delta, tales como la base fija, el efecto móvil y las 3 cadenas cinemáticas similares a un brazo robótico. Cada cadena esta compuestas por un actuador, que funciona como una junta revoluta, un brazo, una junta esférica que une el brazo con el antebrazo, un antebrazo y otra junta esférica que une al antebrazo con el efecto final. Los 3 actuadores están situados en los puntos medios de los lados del triangulo de la base fija.

La figura (4.4) y la tabla (4.3) presentan la nomenclatura de los principales parámetros geométricos y puntos utilizados para el desarrollo de la solución cinemática de posición:

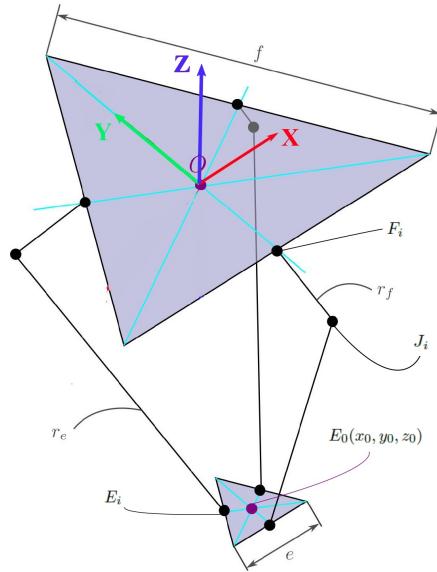


Figura 4.4: Principales parámetros geométricos y puntos para la cinemática de posición del método A [41].

Simbología	Descripción
r_f	Longitud del brazo
r_e	Longitud del antebrazo
f	Lado de base fija
e	Lado del efecto
F_i	Coordenadas de la posición del actuador $i \in \{1, 2, 3\}$
J_i	Coordenadas de las juntas esféricas que une el brazo $i \in \{1, 2, 3\}$ con su antebrazo respectivo
E_i	Coordenadas de las juntas esféricas que unen el antebrazo con el efecto (punto medio del lado del triangulo que forma el efecto) $i \in \{1, 2, 3\}$.
$E_0(x_0, y_0, z_0)$	Coordenadas del centroide del efecto

Tabla 4.3: Parámetros geométricos y puntos para cinemática de posición del método A

4.3.1.2. Cinemática directa

El fin de la cinemática directa es hallar la posición del efecto final del robot delta dada una configuración articular, en este caso la posición angular de los actuadores.

$$(\theta_1, \theta_2, \theta_3) \rightarrow E_0(x_0, y_0, z_0) \quad (4.1)$$

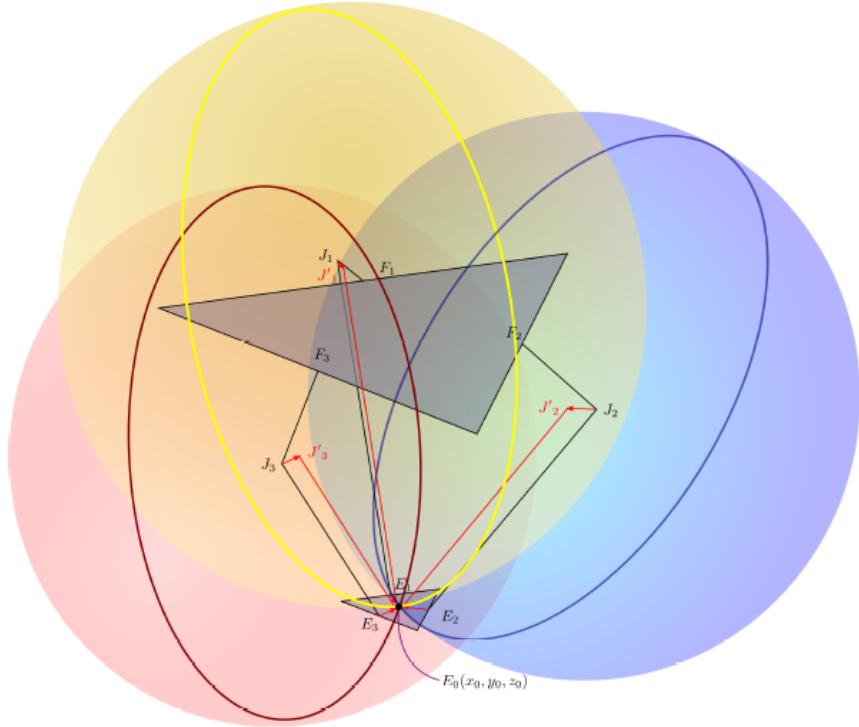


Figura 4.5: Sistema de ecuaciones para la cinemática directa del método A [41].

La posición en el espacio de cada brazo del robot delta ya esta definida gracias a que se conocen los ángulos de los actuadores ($\theta_1, \theta_2, \theta_3$) y la longitud de los brazos r_f .

Por otro lado, sobre los antebrazos solo se conoce la posición de las juntas esféricas J_i que los une con los brazos. Como resultado del incompleto conocimiento espacial de estas piezas, la posición y orientación de cada antebrazo esta restringida por esferas con centro en la junta esférica J_i y radio equivalente al largo de cada antebrazo r_e .

Para finalizar, se realiza una translación de las esferas mencionadas anteriormente para obtener las coordenadas del centroide del efecto final E_0 . Esta translación produce 3 nuevas esferas, con nuevos centros y de radio equivalente al largo del antebrazo r_e . Las nuevas esferas se intersecan en el centroide del efecto final E_0 . Como último paso, para calcular las coordenadas del punto $E_0(x_0, y_0, z_0)$ se realiza un sistema de ecuaciones no lineal con 3 restricciones (esferas que contienen los antebrazos) y 3 incógnitas (coordenadas (x_0, y_0, z_0) del efecto final) .

Los 3 centros de las nuevas esferas de radio r_e que se intersecan en el centroide del efecto $E_0(x_0, y_0, z_0)$ son:

Centros	Centros de esferas
(x_1, y_1, z_1)	$J'_1 \left(0, \left[-\frac{f-e}{2\sqrt{3}} - r_f \cos(\theta_1) \right], -r_f \sin(\theta_1) \right)$
(x_2, y_2, z_2)	$J'_2 \left(\left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_2) \right] \cos(30^\circ), \left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_2) \right] \sin(30^\circ), -r_f \sin(\theta_2) \right)$
(x_3, y_3, z_3)	$J'_3 \left(-\left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_3) \right] \cos(30^\circ), \left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_3) \right] \sin(30^\circ), -r_f \sin(\theta_3) \right)$

Tabla 4.4: Posición de los centros de las esferas originadas por la translación de las juntas J_i

Para simplificar las ecuaciones algebraicas de las esferas, se escriben los centros con la nomenclatura (x_i, y_i, z_i) donde $i \in \{1, 2, 3\}$. El sistema de ecuaciones es:

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = r_e^2 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_e^2 \\ (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = r_e^2 \end{cases} \quad (4.2)$$

Después de un extenso desarrollo algébrico, la solución del sistema de ecuaciones (4.2) es:

$$E_0(x_0, y_0, z_0) = \left(a_1 z + b_1, a_2 z + b_2, \frac{-B - \sqrt{(B^2) - (4AC)}}{2 * A} \right) \quad (4.3)$$

Donde:

$$z = \frac{-B - \sqrt{(B^2) - (4AC)}}{2A} \quad (4.4)$$

$$A = a_1^2 + a_2^2 + 1 \quad (4.5)$$

$$B = 2(a_1(b_1 - x_1) + a_2(b_2 - y_1) - z_1) \quad (4.6)$$

$$C = ((b_1 - x_1)^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2) \quad (4.7)$$

$$a_1 = \frac{(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)}{d} \quad (4.8)$$

$$b_1 = \left(\frac{1}{2 * (-d)} \right) * ((w_2 - w_1)(y_3 - y_1) - (w_3 - w_1)(y_2 - y_1)) \quad (4.9)$$

$$a_2 = \frac{-1}{d} * [(z_2 - z_1)x_3 - (z_3 - z_1)x_2 + (z_3 - z_2)x_1] \quad (4.10)$$

$$b_2 = \frac{1}{2d} * [(w_2 - w_1)x_3 - (w_3 - w_1)x_2 + (w_3 - w_2)x_1] \quad (4.11)$$

$$d = (y_2 - y_1)x_3 - (y_3 - y_1)x_2 - (y_2 - y_3)x_1 \quad (4.12)$$

$$w_i = x_i^2 + y_i^2 + z_i^2 \quad (4.13)$$

4.3.1.3. Cinemática inversa

El objetivo de la cinemática inversa es hallar la posición de los actuadores en el espacio articular dada la posición cartesiana del centroide del efecto final.

$$E_0(x_0, y_0, z_0) \rightarrow (\theta_1, \theta_2, \theta_3) \quad (4.14)$$

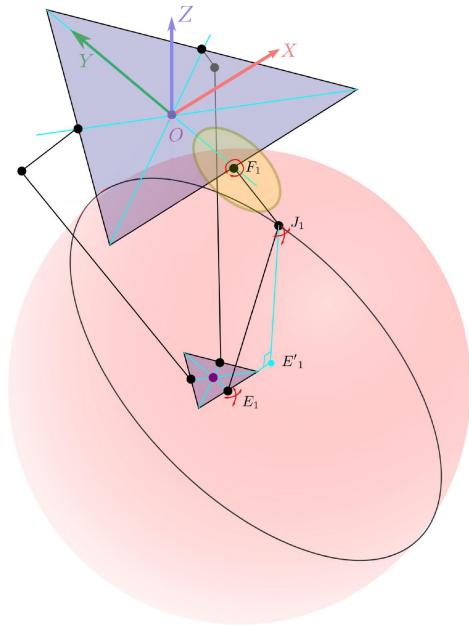


Figura 4.6: Sistema de ecuaciones para la cinemática inversa del método A [41].

Se tiene la información de las coordenadas de las juntas esféricas E_i a causa de que se sabe la posición centroide del efecto final $E_0(x_0, y_0, z_0)$ y que cada lado del triángulo formado por la base fija son paralelos a los lados del triángulo formado por el efecto, es decir, tanto la orientación como la inclinación entre la base fija y el efecto son iguales.

En relación con los antebrazos, solo se conoce el punto E_i , que coincide con las posiciones de cada junta esférica unida al efecto. Esto trae como consecuencia que cada antebrazo esté restringido por esferas con centros en las juntas esféricas E_i con radio equivalente al largo del antebrazo r_e .

Acerca de los brazos solo se tiene conocimiento de los puntos F_i , que son la posición de cada actuador. Los actuadores son juntas revolutas y restringen el movimiento de cada brazo en un plano. La configuración espacial de estos planos para cada cadena cinemática es determinada por los puntos F_i, O (origen) y el plano de la que contiene la base fija. El plano que contiene la base fija es perpendicular a los planos que restringen el movimiento de los brazos. En consecuencia, la posición de los puntos extremos de cada brazo, es decir, los puntos J_i (junta esférica), están restringidos por circunferencia como se muestra en la figura (4.6).

Para finalizar, se calcula la intersección entre las circunferencias para cada cadena cinemática, en otras palabras, los puntos J_i . Una vez obtenida la posición de las 3 juntas J_i , con álgebra simple se calculan los ángulos θ_1, θ_2 y θ_3 .

En el caso del actuador $i = 1$, los centros de las circunferencias que se intersecan en la junta esférica J_1 son:

Centros	Centros esferas	Radio
(y_1, z_1)	$F_1 (y_{F_1} z_{F_1}) = \left(-\frac{f}{2\sqrt{3}}, 0\right)$	r_f
(y_2, z_2)	$E'_1 (y_{E'_1} z_{E'_1}) = (y_0 - \frac{e}{2\sqrt{3}}, z_0)$	$r_2 = \sqrt{r_e^2 - x_o^2}$

Tabla 4.5: Coordenadas del centro de las esferas y sus radios utilizados para la solución de la cinemática inversa del método A.

Las ecuaciones cartesianas de las dos circunferencias quedan representadas en términos de $y_1, z_1, r_f, r_f, y_2, z_2, \sqrt{r_e^2 - x_o^2}$ para simplificar la escritura de los cálculos:

$$\begin{cases} (y - y_1)^2 + (z - z_1)^2 = r_f^2 \\ (y - y_2)^2 + (z - z_2)^2 = r_e^2 - x_o^2 \end{cases} \quad (4.15)$$

La solución del sistema de ecuaciones (4.15) es:

$$J_1 = (x_{J_1}, y_{J_1}, z_{J_1}) = (0, y, a + by) \quad (4.16)$$

Donde:

$$y = \frac{(y_1 - ab) - \sqrt{[-(a + by_1)^2 + (b^2 + 1)r_f^2]}}{(b^2 + 1)} \quad (4.17)$$

$$b = \frac{(y_1 - y_2)}{z_2} \quad (4.18)$$

$$a = \frac{x_o^2 + y_2^2 + z_0^2 + r_f^2 - r_e^2 - y_1^2}{2z_0} \quad (4.19)$$

En último lugar, se determina el ángulo θ_1 por medio del triángulo rectángulo formado en el brazo y la proyección del mismo en el plano XY:

$$\theta_1 = \arctan \left(\frac{z_{J_1}}{y_{F_1} - y_{J_1}} \right) \quad (4.20)$$

Donde: $y_{F_1} = -\frac{f}{2\sqrt{3}}$

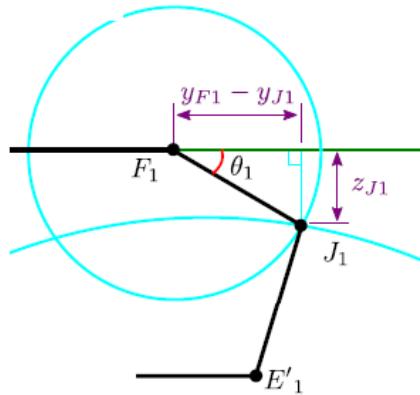


Figura 4.7: Intersección entre círculos en punto J_1 y proyección del punto E_1 sobre el plano XY [41].

Se emplea el mismo procedimiento anterior para encontrar los ángulos θ_2 y θ_3 por medio de matrices de rotación, con un ángulo de rotación de 120° para la cadena cinemática con el actuador 2 y de 240° para la cadena cinemática con el actuador 3. Estas matrices giran el sistema de referencia local en 120° y 240° .

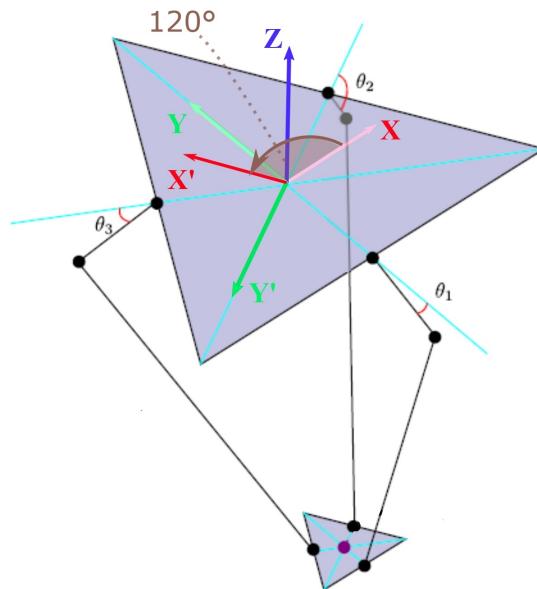


Figura 4.8: Rotación del sistema de referencia local en 120° para la solución de la cinemática inversa del método A [41].

4.3.2. Modelación cinemática de la velocidad

En esta sección, se da a conocer un método para calcular la matriz jacobiana y su estrecha relación con las singularidades de un robot paralelo tipo delta. Se implementan las ideas extraídas del paper [42].

4.3.2.1. Nomenclatura de parámetros geométricos y sistema de referencia local

En la figura (4.9) se visualizan 3 cadenas cinemáticas las cuales se componen de 4 partes mecánicas: la base fija, los brazos accionados, las barras seguidoras y la plataforma móvil.

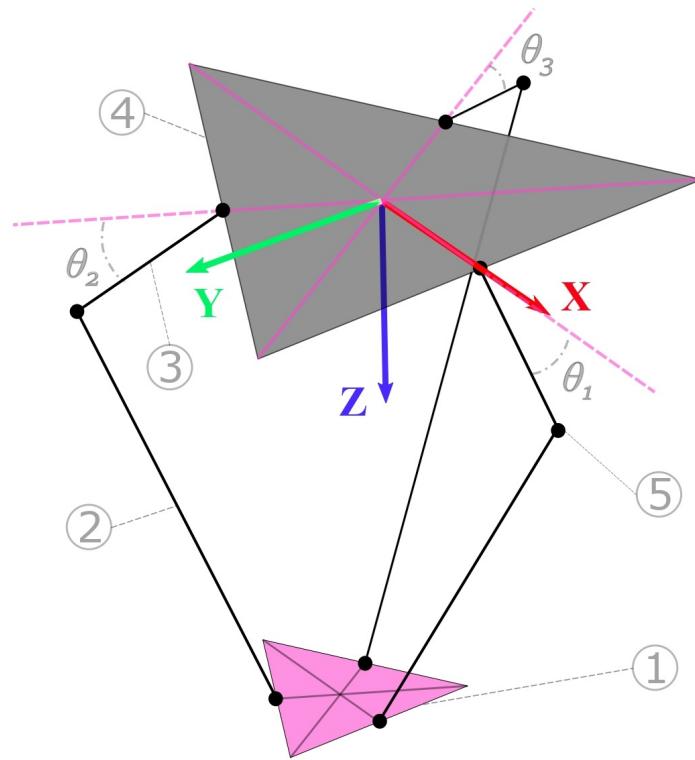


Figura 4.9: Sistema de referencia local para la cinemática de velocidad del método A.

La tabla (4.6) muestra la relación entre la numeración en la figura (4.9) y su nombre.

Número	Nombre
1	plataforma móvil
2	barras seguidoras
3	brazos accionados
4	base fija
5	Juntas universales

Tabla 4.6: Relación entre la numeración y piezas mecánicas de la figura (4.9)

Cada cadena contiene una junta giratoria activada por actuadores en la base fija en la posición donde se conectan los brazos accionados. Cada brazo accionado está conectado a 2 barras seguidoras a través de juntas de 3 grados de libertad. Las juntas son una combinación de junta universal y cojinete. Las barras seguidoras también están conectadas a la plataforma móvil a través del mismo tipo de juntas de 3 grados de libertad, es decir, una combinación de junta universal y cojinete.

Como se muestra en la figura (4.10), un sistema de referencia $O - xyz$ se adjunta al centro O de la plataforma fija, donde los ejes x e y se encuentran en el plano fijo y el eje z apunta hacia abajo verticalmente. Otro sistema de referencia $A_i - x_iy_iz_i$ se adjunta a la base fija en el punto A_i , de modo que el eje x_i está colineal con la línea extendida OA_i , el eje y_i se dirige a lo largo del eje de la articulación giratoria en A_i y el eje z_i es paralelo al eje Z . Los angulos $\phi_{i \in \{1,2,3\}}$ se mide desde el eje x al eje x_i , y son parámetros constantes del diseño del manipulador.

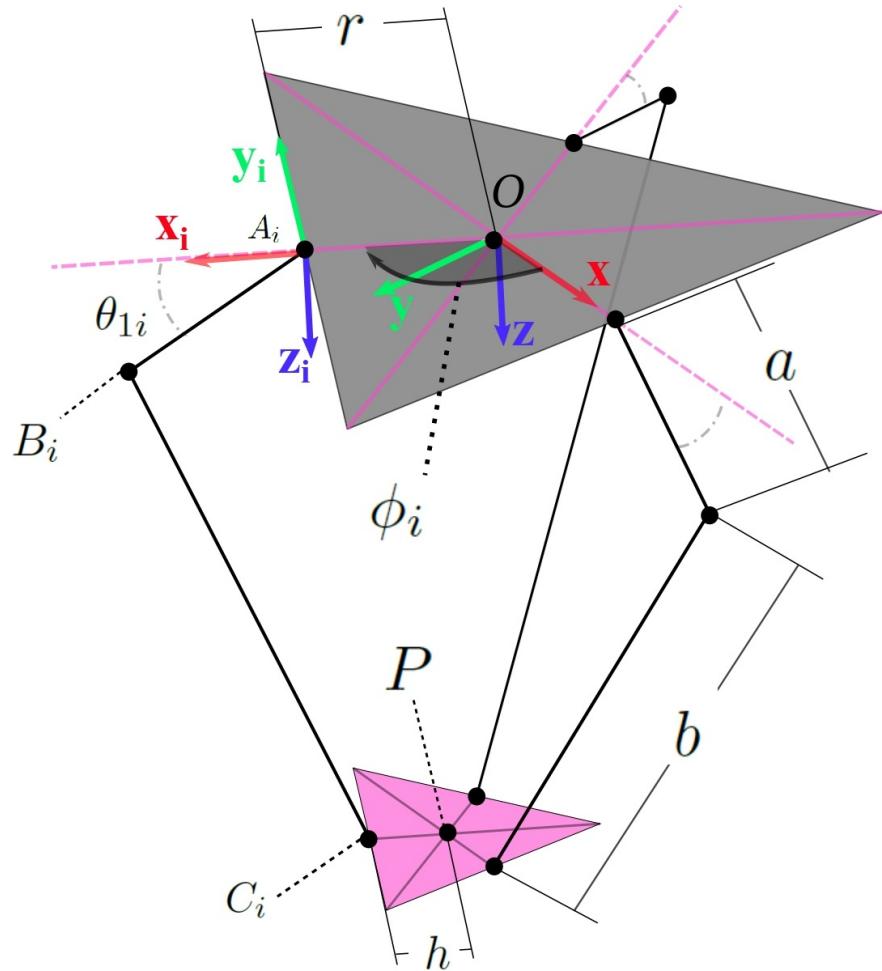


Figura 4.10: Traslación y rotación del sistema de referencia local $O - xyz$ en 120° para la solución de la cinemática de velocidad del metodo A.

La tabla (4.7) presenta la simbología de las principales longitudes, puntos, ángulos y sistemas de referencia para el desarrollo de la solución cinemática de velocidad:

Simbología	Descripción
a	Longitud de los brazos accionados
b	Longitud de las barras seguidoras
r	Longitud desde el centro de la base fija a un actuador $i \in \{1, 2, 3\}$
h	Longitud del centro de la plataforma móvil a una junta universal unida a él.
A_i	Punto que representa la posición de actuador $i \in \{1, 2, 3\}$
B_i	Punto que representa la posición de las juntas universales que unen los brazos accionados y las barras seguidoras
C_i	Punto que representa la posición de las juntas universales que unen las barras seguidoras con la plataforma móvil
P	Punto que representa el centro de la plataforma móvil.
	Ángulo que representa la posición angular de los 3 actuadores sobre el plano de la base fija $\phi_i \in \{0^\circ, 120^\circ, 240^\circ\}$
θ_{1i}	Ángulo que representa la posición angular de las barras accionadas respecto al eje que contiene los puntos del origen del sistema de referencia local y punto A_i
$O - xyz$	Sistema de referencia local
$A_i - x_i y_i z_i$	Sistema de referencia local trasladado y rotado a los puntos A_i dependiendo del ángulo ϕ_i .

Tabla 4.7: Simbología y descripción de las longitudes, puntos, ángulos y sistemas de referencia para el desarrollo de la solución cinemática de velocidad del método A.

4.3.2.2. Vectorización y ángulos de las articulaciones

La figura (4.11) define los ángulos de articulación internos asociados con la extremidad i . El ángulo θ_{2i} se define desde la línea extendida de $\overrightarrow{A_iB_i}$ hasta la línea definida por la intersección del plano del paralelogramo y el plano $A_i - x_i z_i$. El ángulo θ_{3i} se mide desde la dirección del eje y_i hasta el vector $\overrightarrow{B_iC_i}$.

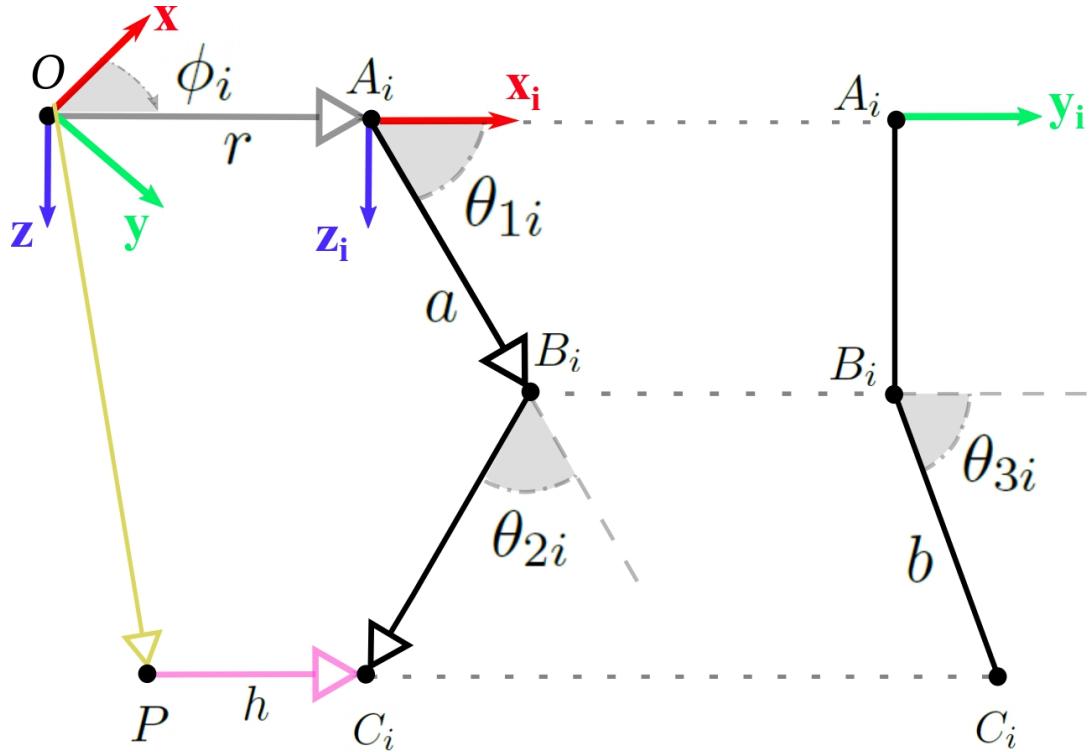


Figura 4.11: Vectorización y ángulos interiores para la cinemática de velocidad del método A.

En la tabla (4.7) se presentan las descripciones de los vectores mostrados en la figura (4.11).

Simbología	Descripción
$\overrightarrow{A_iB_i}$	Vector que representa el brazo accionado $i \in \{1, 2, 3\}$
$\overrightarrow{B_iC_i}$	Vector que representa la barra seguidora $i \in \{1, 2, 3\}$.
\overrightarrow{OP}	Vector que representa el centro de la plataforma móvil.
$\overrightarrow{PC_i}$	Vector que representa la distancia entre el centro de la plataforma móvil y una junta universal unidad a la cadena cinemática $i \in \{1, 2, 3\}$.
$\overrightarrow{OA_i}$	Vector que representa la posición de los actuadores respecto al origen O .

Tabla 4.8: Descripción de los vectores de la figura (4.11)

Se puede escribir una ecuación de cierre de bucle para cada extremidad $i \in \{1, 2, 3\}$ vectorialmente:

$$\overrightarrow{A_i B_i} + \overrightarrow{B_i C_i} = \overrightarrow{O P} + \overrightarrow{P C_i} - \overrightarrow{O A_i} \quad (4.21)$$

Reescribiendo la ecuación (4.21) en el marco de coordenadas $A_i - x_i y_i z_i$, conduce a la siguiente representación matricial:

$$a \begin{bmatrix} \cos \theta_{1i} \\ 0 \\ \sin \theta_{1i} \end{bmatrix} + b \begin{bmatrix} \sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i}) \end{bmatrix} = \begin{bmatrix} \cos \phi_i & \sin \phi_i & 0 \\ -\sin \phi_i & \cos \phi_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} h \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_{xi} \\ c_{yi} \\ c_{zi} \end{bmatrix} \quad (4.22)$$

La posición del punto $C_i(c_{xi}, c_{yi}, c_{zi})$ esta en relación con el marco de coordenadas $A_i - x_i y_i z_i$.

Realizando álgebra en la ecuación (4.22) se pueden determinar los ángulos θ_{2i} y θ_{3i} con las siguientes ecuaciones:

$$\theta_{3i} = \cos^{-1} \frac{c_{yi}}{b} \quad (4.23)$$

$$\theta_{2i} = \cos^{-1} \left(\frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2ab \sin \theta_{3i}} \right) \quad (4.24)$$

Una vez determinados los ángulos θ_{3i} y θ_{2i} , solo queda un par de ecuaciones con el ángulo θ_{1i} como la única incógnita que se deriva de la ecuación (4.22). Por lo tanto, es posible determinar θ_{1i} fácilmente.

4.3.2.3. Jacobiano

La matriz jacobiana tradicional proporciona una transformación de la velocidad del efecto final en el espacio cartesiano a las velocidades articulares accionadas.

Con referencia a la figura (4.11), una ecuación de cierre de bucle para una extremidad i en marco de coordenadas $A_i - x_i y_i z_i$ se puede escribir como:

$$\overrightarrow{OP} + \overrightarrow{PC_i} = \overrightarrow{OA_i} + \overrightarrow{A_i B_i} + \overrightarrow{B_i C_i} \quad (4.25)$$

Al diferenciar vectorialmente la ecuación (4.25) con respecto a el tiempo y separando las componentes de velocidad de la plataforma móvil de las velocidades angulares de los actuadores se llega a la siguiente expresión:

$$J_x v_p = J_\theta \dot{\theta} \quad (4.26)$$

La expresión matricial de la ecuación (4.26) es:

$$\begin{bmatrix} J_{1x} & J_{1y} & J_{1z} \\ J_{2x} & J_{2y} & J_{2z} \\ J_{3x} & J_{3y} & J_{3z} \end{bmatrix} \begin{bmatrix} v_{px} \\ v_{py} \\ v_{pz} \end{bmatrix} = \begin{bmatrix} J_{1\theta} & 0 & 0 \\ 0 & J_{2\theta} & 0 \\ 0 & 0 & J_{3\theta} \end{bmatrix} \begin{bmatrix} \dot{\theta}_{11} \\ \dot{\theta}_{12} \\ \dot{\theta}_{13} \end{bmatrix} \quad (4.27)$$

Donde:

$$J_{ix} = \cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \cos \phi_i - \cos \theta_{3i} \sin \phi_i \quad (4.28)$$

$$J_{iy} = \cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \sin \phi_i + \cos \theta_{3i} \cos \phi_i \quad (4.29)$$

$$J_{iz} = \sin(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \quad (4.30)$$

$$J_{i\theta} = a \sin \theta_{2i} \sin \theta_{3i} \quad (4.31)$$

Manipulando algebraicamente la ecuación (4.26):

$$v_p = J \dot{\theta} \quad (4.32)$$

Donde:

- $J = J_x^{-1} J_\theta$ es el jacobiano del robot delta
- $v_p = [v_{px}, v_{py}, v_{pz}]^T$ es la velocidad del punto P en la plataforma móvil
- $\dot{\theta} = [\dot{\theta}_{11}, \dot{\theta}_{12}, \dot{\theta}_{13}]^T$ es la velocidad angular de los actuadores

Finalmente, el jacobiano es J y representa el cambio de las posiciones en el espacio cartesiano de la plataforma móvil respecto al cambio de los ángulos en el espacio articular de los actuadores.

4.3.2.4. Singularidades

Se debe tener especial cuidado en el diseño de estos manipuladores paralelos para evitar singularidades. Cuando un manipulador está en una posición singular, la matriz jacobiana también es singular. En el caso de los manipuladores paralelos conviene trabajar con un jacobiano de dos partes: matriz jacobiana inversa J_θ y la directa J_x . La ventaja de este jacobiano bipartito es que permite de forma natural, la identificación y clasificación de varios tipos de singularidades. Una de las formas de analizar las singularidades es igualando el determinante de la matriz jacobiana a cero y extraer varias posturas indeseables robot delta. Desde la ecuación (4.26) se puede observar y clasificar las singularidades en 3 situaciones:

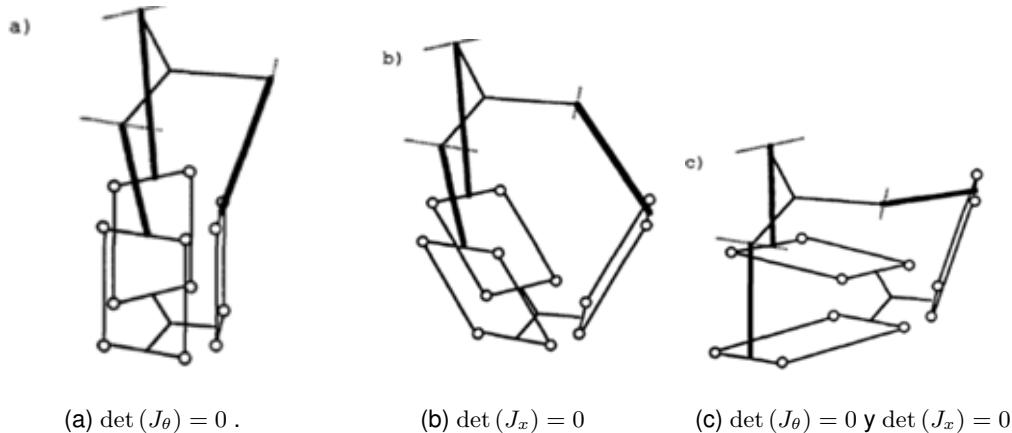


Figura 4.12: Representación de 3 tipos de singularidades de un robot delta [22]

- Cuando $\det(J_\theta) = 0$. Esto significa que $\theta_{2i} = 0 \vee \pi$, o $\theta_{3i} = 0 \vee \pi$, para $i \in \{1, 2, 3\}$. En esta situación, los tres pares de barras seguidoras son paralelos. Por tanto, la plataforma móvil tiene 3 grados de libertad, se mueve a lo largo de una superficie esférica y gira sobre el eje perpendicular a la plataforma móvil (figura (4.12a)). Al mismo tiempo, esta configuración representa el límite del espacio de trabajo, es decir, los puntos más lejanos que puede alcanzar el robot.
- Cuando $\det(J_x) = 0$. Esto significa que $\theta_{1i} + \theta_{2i} = 0 \vee \pi$, o $\theta_{3i} = 0 \vee \pi$, para $i \in \{1, 2, 3\}$. En esta situación, dos pares de barras seguidoras son paralelas. Los la plataforma móvil tiene 1 grado de libertad, es decir, la plataforma móvil se mueve en una sola dirección (figura (4.12b)).
- Cuando $\det(J_\theta) = 0$ y $\det(J_x) = 0$. Esta situación ocurre cuando $\theta_{3i} = 0 \vee \pi$, para $i \in \{1, 2, 3\}$. En esta situación, dos pares de barras seguidoras están en el mismo plano o dos planos paralelos. La plataforma móvil tiene 1 grado de libertad, es decir, la plataforma móvil gira solamente sobre el eje horizontal (figura (4.12c)).

4.3.3. Modelación cinemática de aceleración

En esta sección se presentan las ecuaciones para determinar la aceleración angular de los actuadores del robot delta. La aceleración angular se determina derivando matricialmente la ecuación (4.26) de la sección (4.3.2) llamada modelación cinemática de velocidad para el método A.

$$\ddot{\theta} = J_{\theta}^{-1} * \left[\dot{J}_x v_p + J_x a_p - \dot{J}_{\theta} \dot{\theta} \right] \quad (4.33)$$

Las matrices en la ecuación (4.33) son:

$$\begin{aligned} \begin{bmatrix} \ddot{\theta}_{11} \\ \ddot{\theta}_{12} \\ \ddot{\theta}_{13} \end{bmatrix} &= \begin{bmatrix} J_{1\theta} & 0 & 0 \\ 0 & J_{2\theta} & 0 \\ 0 & 0 & J_{3\theta} \end{bmatrix}^{-1} \\ &* \left[\begin{bmatrix} \dot{J}_{1x} & \dot{J}_{1y} & \dot{J}_{1z} \\ \dot{J}_{2x} & \dot{J}_{2y} & \dot{J}_{2z} \\ \dot{J}_{3x} & \dot{J}_{3y} & \dot{J}_{3z} \end{bmatrix} \begin{bmatrix} v_{px} \\ v_{py} \\ v_{pz} \end{bmatrix} + \begin{bmatrix} J_{1x} & J_{1y} & J_{1z} \\ J_{2x} & J_{2y} & J_{2z} \\ J_{3x} & J_{3y} & J_{3z} \end{bmatrix} \begin{bmatrix} a_{px} \\ a_{py} \\ a_{pz} \end{bmatrix} - \begin{bmatrix} \dot{J}_{1\theta} & 0 & 0 \\ 0 & \dot{J}_{2\theta} & 0 \\ 0 & 0 & \dot{J}_{3\theta} \end{bmatrix} \begin{bmatrix} \dot{\theta}_{11} \\ \dot{\theta}_{12} \\ \dot{\theta}_{13} \end{bmatrix} \right] \end{aligned} \quad (4.34)$$

Donde:

$$\dot{J}_{ix} = A'_{ix} - B'_{ix} \quad (4.35)$$

$$A'_{ix} = \cos\phi_i * \left[[-\sin(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin\theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) * \cos\theta_{3i} * \dot{\theta}_{3i}] \right] \quad (4.36)$$

$$B'_{ix} = \sin\phi_i * [-\sin\theta_{3i} * \dot{\theta}_{3i}] \quad (4.37)$$

$$\dot{J}_{iy} = A'_{iy} + B'_{iy} \quad (4.38)$$

$$A'_{iy} = \sin\phi_i * \left[[-\sin(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin\theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) * \cos\theta_{3i} * \dot{\theta}_{3i}] \right] \quad (4.39)$$

$$B'_{iy} = \cos\phi_i * [-\sin\theta_{3i} * \dot{\theta}_{3i}] \quad (4.40)$$

$$\dot{J}_{iz} = [\cos(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin\theta_{3i}] + [\sin(\theta_{1i} + \theta_{2i}) * \cos\theta_{3i} * \dot{\theta}_{3i}] \quad (4.41)$$

$$\dot{J}_{i\theta} = a \left[[\cos\theta_{2i} * \dot{\theta}_{2i} * \sin\theta_{3i}] + [\sin\theta_{2i} * \cos\theta_{3i} * \dot{\theta}_{3i}] \right] \quad (4.42)$$

$$\dot{\theta}_{3i} = \left[\frac{-1}{\sqrt{1 - \left(\frac{c_{yi}}{b} \right)^2}} \right] * \left[\frac{c'_{yi}}{b} \right] \quad (4.43)$$

$$c'_{yi} = [-\sin\phi_i * v_{px} + \cos\phi_i * v_{py}] \quad (4.44)$$

$$\dot{\theta}_{2i} = \left[\frac{-1}{\sqrt{1 - (K)^2}} \right] * [K'] \quad (4.45)$$

$$K = \frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2ab \sin \theta_{3i}} \quad (4.46)$$

$$K' = \left[\frac{1}{2ab} \right] * \left[\frac{A_{11} - A_{12}}{B_1} \right] \quad (4.47)$$

$$A_{11} = [2c_{xi}c'_{xi} + 2c_{yi}c'_{yi} + 2c_{zi}c'_{zi}] * [\sin\theta_{3i}] \quad (4.48)$$

$$A_{12} = [c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2] * [\cos \theta_{3i} * \dot{\theta}_{3i}] \quad (4.49)$$

$$B_1 = [\sin \theta_{3i}]^2 \quad (4.50)$$

$$c'_{xi} = \cos\phi_i * v_{px} + \sin\phi_i * v_{py} \quad (4.51)$$

$$c'_{zi} = v_{pz} \quad (4.52)$$

4.3.4. Modelación dinámica

La aplicación directa de las leyes de Newton al movimiento de sistemas de partículas es sustituida por una propuesta más general, un método para encontrar las ecuaciones de movimiento para todos los sistemas dinámicos, desarrollado por el matemático Joseph Louis Lagrange. En el anexo (A) se detalla esta propuesta. Primero se habla de los conceptos básicos hasta llegar a la formulación de Lagrange. Luego se obtienen las ecuaciones de Lagrange partiendo del principio de D'Alembert. Finalmente se aplican las ecuaciones de Lagrange al robot delta para determinar el torque de cada actuador. Toda la información teórica en esta sección es recopilada del libro [43].

4.3.4.1. Teoría de las ecuaciones de Lagrange

Se dice que las ligaduras holónomas son empleadas en forma explícita cuando no se utilizan para eliminar las coordenadas dependientes, efectuándose la descripción del sistema de partículas dado con la totalidad (dependientes + independientes) de sus coordenadas. Las ecuaciones de Lagrange para sistemas con ligaduras holónomas usadas de forma explícita se representan en la siguiente formula:

$$\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{q}_j} \right) - \frac{\delta L}{\delta q_j} = Q_j^{(lig)}(q_j) + Q_j^{(NU)}(q_j) \quad (4.53)$$

Con $j = 1, 2, 3, \dots, \eta = 3N$

Donde:

$$Q_j^{(lig)}(q_j) = \sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta q_j} \quad (4.54)$$

L representa el lagrangiana o lagrangiano:

$$L(q_j, \dot{q}_j, t) = T(q_j, \dot{q}_j, t) - V(q_j, \dot{q}_j, t) \quad (4.55)$$

Las ecuaciones (4.53) son las ecuaciones de Lagrange para ligaduras holónomas $f_i^{(h)}(q_i, t) = 0$ cuando son usadas en forma explícita. Representan un conjunto de $\eta = 3N$ ecuaciones para el conjunto completo de coordenadas (dependientes + independientes) $q_1, q_2, q_3, \dots, q_{\eta=3N}$. Estas ecuaciones, en conjunto con las $K^{(h)}$ ecuaciones de ligadura dadas por $f_i^{(h)}(q_i, t) = 0$ forman un sistema de $3N + K^{(h)}$ ecuaciones para $3N + K^{(h)}$ incógnitas, $3N$ coordenadas q_j y $K^{(h)}$ multiplicadores de Lagrange λ_l , quedando así determinado dicho sistema de ecuaciones. Aquí las ligaduras entran en forma explícita en los $Q_j^{(lig)}$ dados por la ecuación (4.54), que son fuerzas adicionales que actúan sobre el sistema. Debido a que estas fuerzas están relacionadas con las ligaduras se les da el nombre de fuerzas generalizadas de ligadura, las cuales no realizan trabajo virtual como lo requiere la validez del Principio de D'Alembert.

$Q_j^{(NU)}$ son llamadas comúnmente fuerzas generalizadas, sin embargo, no solo son fuerzas. Estas dependen de la dimensión de las coordenadas generalizadas q_j que se utiliza en la ecuación de Lagrange :

1. Si q_j es una longitud, entonces $Q_j^{(NU)}$ es una fuerza.
2. Si q_j es un ángulo, entonces $Q_j^{(NU)}$ es un torque.
3. Si q_j es una superficie, entonces $Q_j^{(NU)}$ es una tensión.
4. Si q_j es un volumen, entonces es $Q_j^{(NU)}$ una presión.

4.3.4.2. Nomenclatura de parámetros geométricos y sistema de referencia global

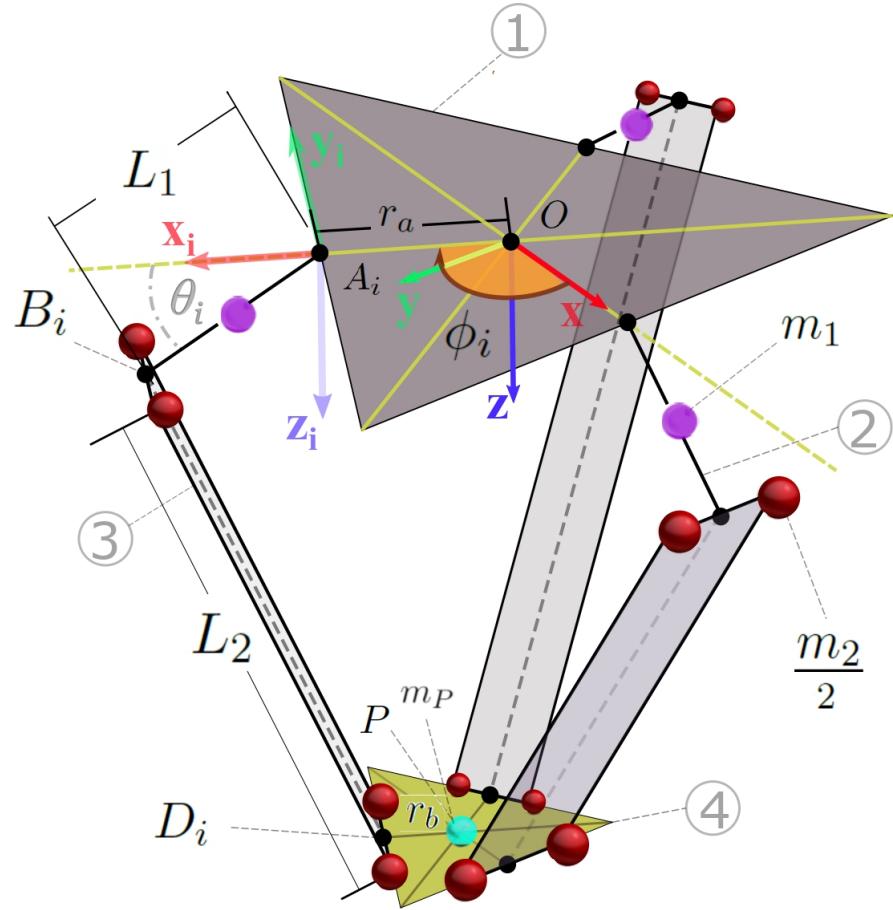


Figura 4.13: Sistema de referencia local, parámetros geométricos y masas para la solución de la dinámica inversa del método A [22].

En la figura (4.13) se pueden visualizar 3 cadenas cinemáticas $i \in \{1, 2, 3\}$ las cuales se componen de 4 partes mecánicas: la base fija, brazos, antebrazos y base móvil. La unión entre la base fija y los brazos representa una junta tipo revoluta accionadas por motores. También se muestra la simplificación de masas para resolver la dinámica del robot delta.

En la tabla (4.9) muestra la relación entre la numeración en la figura (4.13) y su nombre.

Número	Descripción
1	Base fija
2	Brazo
3	Antebrazo
4	Base Movil

Tabla 4.9: Relación entre la numeración y piezas mecánicas de la figura (4.13)

En la tabla (4.10) se presenta la simbología de los sistema de referencia, magnitudes, puntos y ángulos principales para el desarrollo de la dinámica inversa del robot delta.

Símbolo	Descripción
L_1	Longitud brazo
L_2	Longitud antebrazo
r_a	Radio de circulo inscrito en triangulo de base fija
r_b	Radio de circulo inscrito en triangulo de base móvil
m_1	Masa de un brazo posicionada en su centroide
m_2	Masa de una de las dos barras de los antebrazos, distribuida equitativamente en las extremidades.
m_P	Masa de la base móvil.
A_i	Punto de unión entre base fija y brazo para la cadena cinemática $i \in \{1, 2, 3\}$
B_i	Punto de unión entre brazo y antebrazo para la cadena cinemática $i \in \{1, 2, 3\}$
D_i	Punto de unión entre antebrazo y base móvil para la cadena cinemática $i \in \{1, 2, 3\}$
P	Punto que representa el centroide base móvil
ϕ_i	Ángulo que representa la posición angular de los 3 actuadores sobre el plano de la base fija en relación al sistema de referencia local para la cadena cinemática $i \in \{1, 2, 3\}$.
θ_i	Ángulo que representa la posición angular de los brazos respecto al eje construido por el punto del origen del sistema de referencia local y para la cadena cinemática $i \in \{1, 2, 3\}$.
$O - xyz$	Sistema de referencia local utilizado para resolver la dinámica del robot delta
$A_i - x_i y_i z_i$	Sistema de referencia con origen coincidente en la junta revoluta que une la base fija con los brazos para la cadena cinemática $i \in \{1, 2, 3\}$
F_x, F_y, F_z	Fuerzas externas sobre la base móvil

Tabla 4.10: Simbología y descripción de los sistema de referencia, longitudes, puntos y ángulos principales para el desarrollo de la dinámica inversa del método A.

4.3.4.3. Dinámica inversa

Un paso importante en el proceso de diseño de un robot es comprender el comportamiento del dispositivo mientras se mueve por su espacio de trabajo o al realizar una tarea específica. Este comportamiento se determina mediante el estudio de la dinámica del mecanismo, donde se pueden determinar las fuerzas que actúan sobre los elementos y los pares requeridos por los actuadores. En consecuencia, cada componente debe optimizarse en dimensiones y material para ser utilizado en los procesos de fabricación. Por lo tanto, es esencial buscar cuáles son los enfoques comunes utilizados para calcular la dinámica del robot. Hay tres enfoques: en primer lugar, la formulación de Newton-Euler, que es un método tradicional basado en las leyes de Newton, pero necesita una gran cantidad de ecuaciones, ya que requiere establecer un número n de ecuaciones para cada elemento con el fin de resolver el sistema. Como resultado, es el método más lento en cuanto a eficiencia computacional. En segundo lugar, el principio de trabajo virtual, que es una derivación del método de Euler y Lagrange utilizando el cálculo de variaciones, se ha utilizado para estudiar la mecánica de los cuerpos deformables. Este enfoque establece que el trabajo total realizado por una fuerza a lo largo de la trayectoria sobre una partícula se puede calcular si esas fuerzas actúan cuando la partícula se mueve del punto A al punto B. Finalmente el tercer enfoque es la formulación lagrangiana, que se basa en variaciones de cálculo, establece que un sistema dinámico puede expresarse en términos de su energía cinética y potencial conduciendo de manera sencilla a la solución del problema. Además, se considera una buena opción para el control en tiempo real de manipuladores paralelos.

Newton estableció los fundamentos de la dinámica donde se deben identificar las fuerzas que producen el cambio para resolver la dinámica de un cuerpo. En su enfoque, se utiliza un diagrama de cuerpo libre para representar todas las fuerzas de cada cuerpo para desarrollar el análisis. Por el contrario, Leibniz, un contemporáneo de Newton, pensó que la acción de una fuerza podría medirse analizando los cambios en la energía potencial y cinética. Más tarde, los métodos variacionales aparecieron formalmente gracias a Lagrange y Hamilton. En este enfoque, la energía, que es una cantidad escalar, facilita el análisis del trabajo realizado en comparación con vectores como la velocidad y la aceleración, que pueden volverse engorrosos en algunos sistemas de coordenadas. La ventaja de utilizar un enfoque lagrangiano es que mientras que en la mecánica vectorial es necesario definir un sistema de coordenadas específico para todos los objetos analizados, en los métodos variacionales no importa si un objeto está en coordenadas cilíndricas y el otro en coordenadas esféricas. Los detalles no son importantes siempre que puedan expresarse en términos de coordenadas que tienen tres coordenadas que se refieren al centro de masa del cuerpo y tres coordenadas a la orientación específica del cuerpo en el espacio.

Las ecuaciones de Lagrange para el robot delta en este trabajo se definen a partir de 6 coordenadas generalizadas y 3 restricciones holónomas de forma explícita. Por lo tanto, la modelación dinámica se resume en las siguientes ecuaciones:

$$\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{q}_j} \right) - \frac{\delta L}{\delta q_j} = Q_j^{(lig)}(q_j) + Q_j^{(NU)}(q_j) \quad (4.56)$$

$$Q_j^{(lig)}(q_j) = \sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta q_j} \quad (4.57)$$

$$L(q_j, \dot{q}_j, t) = T(q_j, \dot{q}_j, t) - V(q_j, \dot{q}_j, t) \quad (4.58)$$

$$T = \left[\frac{1}{2} m_P (\dot{X}_p^2 + \dot{Y}_p^2 + \dot{Z}_p^2) \right] + \left[\left(\frac{1}{6} m_1 L_1^2 \right) * \sum_{i=1}^3 (\dot{\theta}_i)^2 \right] + \left(\frac{m_2}{2} \right) * \left[\sum_{i=1}^3 (\dot{X}_p^2 + \dot{Y}_p^2 + \dot{Z}_p^2) + (L_1^2 \dot{\theta}_i^2) \right] \quad (4.59)$$

$$V = -[m_p g Z_p] - \left[\frac{1}{2} m_1 g L_1 \sum_{i=1}^3 \sin \theta_i \right] - \left[m_2 g L_1 \sum_{i=1}^3 \sin \theta_i \right] - [3 m_2 g Z_p] \quad (4.60)$$

$$q_j \in \{X_p, Y_p, Z_p, \theta_1, \theta_2, \theta_3\} \quad (4.61)$$

$$K^{(h)} = 3 \quad (4.62)$$

$$f_i^{(h)}(\theta_i, \phi_i) = (X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i)^2 + (Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i)^2 + (Z_P - L_1 \sin \theta_i)^2 - L_2^2 \quad (4.63)$$

Donde L es el lagrangiano, T es la energía cinética total, V es la energía potencial, q_j es la j-ésima coordenada generalizada, (X_p, Y_p, Z_p) representan las coordenadas del centroide de la plataforma móvil, $(\theta_1, \theta_2, \theta_3)$ es la posición angular de los motores, $Q_j^{(NU)}$ son fuerzas externas generalizadas (no potencial), λ_l es el multiplicador de Lagrange, $K^{(h)}$ cantidad de restricciones holónomas y $f_l^{(h)}$ son la ecuación de restricción holónomas de forma explícita. Las fuerzas de fricción no son restricciones a pesar de que juegan un papel importante en el análisis de la dinámica, por lo que pueden tratarse por separado. Una vez que se introduce la ecuación de Lagrange para el movimiento y se establecen los parámetros necesarios para su análisis, es posible calcular el torque en los actuadores de un robot delta bajo una trayectoria específica.

Se sabe que, si en las ecuaciones de Lagrange la coordenada generalizada q_j es un ángulo, entonces $Q_j^{(NU)}(q_j)$ representa un torque. Por lo tanto, para determinar el torque necesario en los motores para realizar una determinada trayectoria de la base móvil, se resuelven las ecuaciones de Lagrange con las coordenadas generalizadas $q_j = [\theta_1, \theta_2, \theta_3]$. La ecuación que determina los torques de los motores $i \in \{1, 2, 3\}$ es:

$$\begin{aligned}\tau_i = & \left(\frac{1}{3}m_1 + m_2\right)L_1^2\ddot{\theta}_i - \left(\frac{1}{2}m_1 + m_2\right)gL_1 \cos \theta_i \\ & - 2\lambda_i L_1 [(X_P \cos \phi_i + Y_P \sin \phi_i - r) \sin \theta_i - Z_P \cos \theta_i]\end{aligned}\quad (4.64)$$

Donde los coeficientes de Lagrange λ_i se determinan resolviendo el sistema de ecuaciones a partir de:

$$(m_p + 3m_2) \ddot{X}_p - 2 \sum_{l=i=1}^3 \lambda_l (X_P - r \cos \phi_i - L_1 \cos \theta_i \cos \phi_i) = F_{px} \quad (4.65)$$

$$(m_p + 3m_2) \ddot{Y}_p - 2 \sum_{l=i=1}^3 \lambda_l (Y_P - r \sin \phi_i - L_1 \cos \theta_i \sin \phi_i) = F_{py} \quad (4.66)$$

$$(m_p + 3m_2) \ddot{Z}_p - 2 \sum_{l=i=1}^3 \lambda_l (Z_P - L_1 \sin \theta_i) - (m_p + 3m_2) g = F_{pz} \quad (4.67)$$

El término (F_{px}, F_{py}, F_{pz}) son las fuerzas externas que se aplican a la base móvil y $r = r_a - r_b$.

4.4. Método B

4.4.1. Modulación cinemática de la posición

Con el objetivo de modelar la cinemática del robot delta, en esta sección se implementan las ideas expuestas en la tesis doctoral [44], que aborda en uno de sus capítulos la problemática de cinemática directa e inversa.

4.4.1.1. Nomenclatura de parámetros geométricos y sistema de referencia local

En la figura (4.14) se presenta el sistema de referencia local, las partes mecánicas y la nomenclatura de los parámetros del robot delta simplificado para resolver el problema de cinemática.

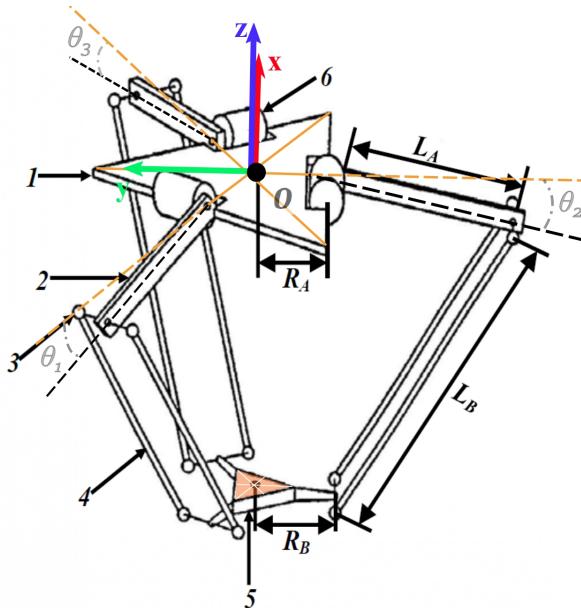


Figura 4.14: Sistema de referencia local para la cinemática del método B [44].

La tabla (4.11) muestra la relación entre la numeración en la figura (4.14) y su nombre:

Número	Nombre
1	Base Fija
2	Brazo
3	Junta esférica
4	Antebrazo
5	Efecto final
6	Actuador

Tabla 4.11: Nombres de partes mecánicas del robot delta del método B.

Antes de empezar los cálculos de cinemática, es mejor explicar algunos términos que se utilizan ampliamente en la formulación cinemática.

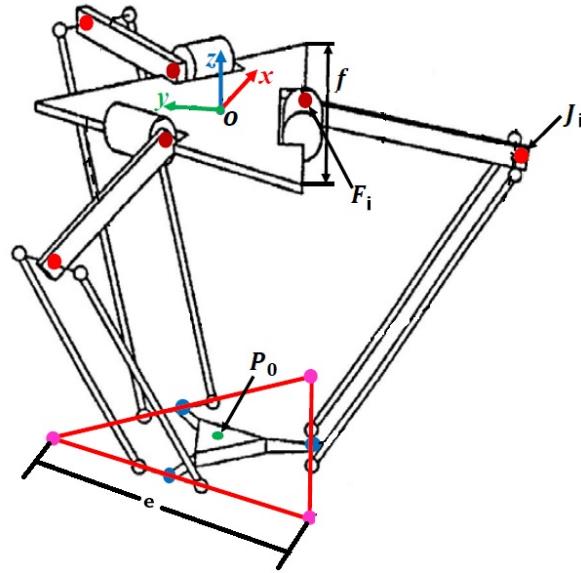


Figura 4.15: Principales parámetros geométricos y puntos para la solución de la cinemática del método B [44].

Los parámetros necesarios para resolver la cinemática del robot delta se presentan en las figuras (4.14), (4.15) y la tabla (4.12):

Parametro	Descripción
L_A	Largo del brazo
L_B	Largo del antebrazo
R_A	Distancia entre el centro de la base fija y la junta revoluta o actuador
R_B	Distancia entre el centro del efecto final y la junta que lo une con el antebrazo
f	Longitud de un lado del triángulo equilátero inscribe el círculo formado por los puntos F'_1, F'_2 y F'_3 (base fija)
e	Longitud de un lado del equilátero triángulo inscribe el círculo formado por los puntos P'_1, P'_2 y P'_3 (efecto final)
θ_i	Angulo de los actuadores
J_i	Punto de la junta esférica que conecta los brazos con el antebrazo
F_i	Punto de la posición de los actuadores
P_0	Posición del final efecto en el espacio cartesiano con respecto al sistema de coordenadas con origen O en el centroide de la base fija

Tabla 4.12: Principales parámetros geométricos y puntos para la solución de la cinemática del método B.

La definición de los vectores utilizados para la formulación de la solución de cinemática del robot paralelo delta se muestra en la figura (4.16) y en la tabla (4.13).

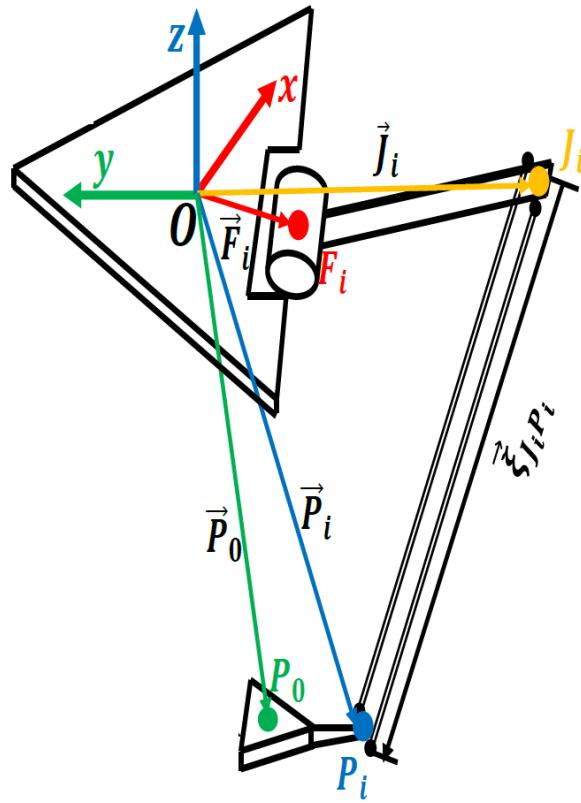


Figura 4.16: Vectorización para la solución de cinemática del método B [44].

Parámetro	Descripción
\vec{P}_0	Vector con punto inicial en el centro de la base fija y con extremo en el centro del efecto final
\vec{P}_i	Vector que inicia en el centro de la base fija y su extremo se encuentra en la junta esférica que conecta el efecto con el antebrazo $i \in \{1, 2, 3\}$.
\vec{J}_i	Vector que inicia en el centro de la base fija y su extremo se encuentra en la junta esférica que conecta el brazo con el antebrazo $i \in \{1, 2, 3\}$.
\vec{F}_i	Vector que inicia en el centro de la base fija y su extremo se encuentra en la posición de los motores o actuadores $i \in \{1, 2, 3\}$
$\overrightarrow{\xi_{F_i J_i}}$	Vector que inicia en la junta esférica que conecta el brazo con el antebrazo y su extremo se encuentra en la junta esférica que conecta el efecto con el antebrazo $i \in \{1, 2, 3\}$. Este vector es una representación de los antebrazos.

Tabla 4.13: Vectorización para la solución de cinemática del método B

4.4.1.2. Cinemática directa

En la cinemática directa se calcula la posición del efecto final del manipulador robótico a partir de la información dada de los ángulos en los actuadores.

$$\theta_1, \theta_2, \theta_3 \rightarrow P_0 (P_{0x}, P_{0y}, P_{0z}) \quad (4.68)$$

El método de solución de la cinemática directa en esta sección es el mismo que el empleado para el método A. Lo único que cambia es la nomenclatura de los parámetros y el orden de numeración de los ángulos de los actuadores (figura (4.18)).

Como los ángulos $\theta_1, \theta_2, \theta_3$ se conocen en el problema de cinemática directa, las coordenadas de los puntos J_1, J_2, J_3 se pueden encontrar fácilmente. Los antebrazos $\overline{J_1P_1}, \overline{J_2P_2}, \overline{J_3P_3}$ pueden girar libremente alrededor de los puntos J_1, J_2, J_3 respectivamente. A fin de calcular la cinemática directa, los puntos J_1, J_2, J_3 se trasladan a J'_1, J'_2, J'_3 , utilizando los vectores de translación $\xi_{P_1P_0}, \xi_{P_2P_0}, \xi_{P_3P_0}$ respectivamente, como se muestra en la figura (4.17) (vectores color amarillo).

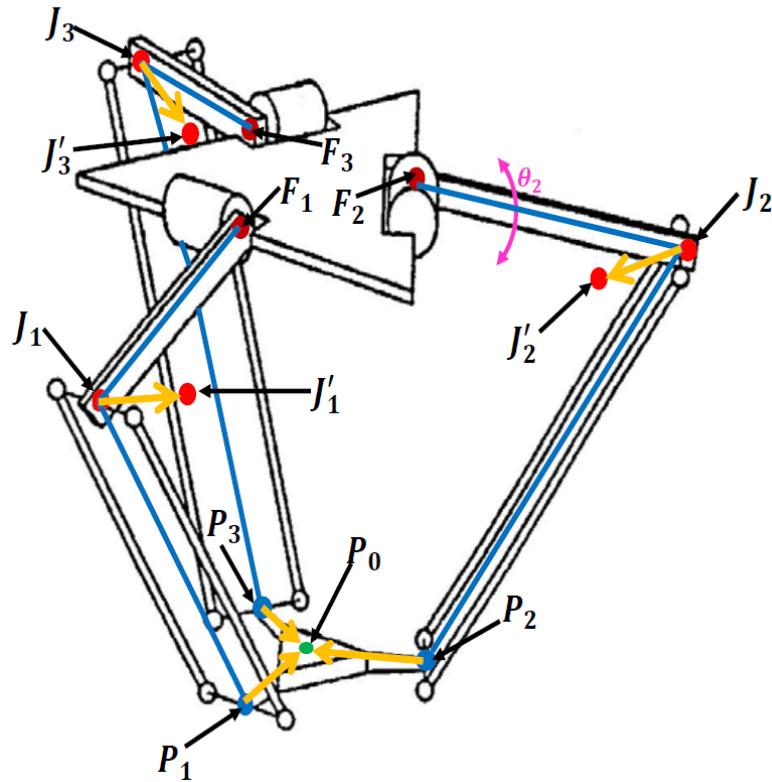


Figura 4.17: Traslación de los vectores que representan los antebrazos para crear un sistema de ecuaciones y determinar la cinemática directa del método B [44].

Como resultado de esta traslación se producen tres esferas con radio L_B y centro en los puntos J'_1, J'_2, J'_3 que se cruzan en el punto P_0 .

Resumiendo, al igual que metodo A, el resultado de las traslaciones de las 3 esferas con centros en las juntas esféricas J_1, J_2, J_3 producen tres nuevas esferas con radio L_B y centro en los puntos J'_1, J'_2, J'_3 que se cruzan en el punto P_0 . Las coordenadas del vector $\vec{P}_0 = [P_{0x}, P_{0y}, P_{0z}]^T$ se pueden obtener resolviendo las tres ecuaciones que representan las nuevas esferas en el espacio cartesiano simultáneamente. Las ecuaciones son:

$$(P_{0x} - J'_{1x})^2 + (P_{0y} - J'_{1y})^2 + (P_{0z} - J'_{1z})^2 = L_B^2 \quad (4.69)$$

$$(P_{0x} - J'_{2x})^2 + (P_{0y} - J'_{2y})^2 + (P_{0z} - J'_{2z})^2 = L_B^2 \quad (4.70)$$

$$(P_{0x} - J'_{3x})^2 + (P_{0y} - J'_{3y})^2 + (P_{0z} - J'_{3z})^2 = L_B^2 \quad (4.71)$$

Por lo tanto, se dispone de un sistema de ecuaciones con 3 ecuaciones y con 3 incógnitas P_{0x}, P_{0y}, P_{0z} .

Para la calcular los centros J'_1, J'_2, J'_3 primero se miden las magnitudes de otros vectores que son fundamentales para identificar de mejor forma estos puntos en el espacio y facilitar su cálculo. En las figuras (4.18) y (4.19b) se muestra la vista superior del robot paralelo delta. Desde esta vista, se puede calcular la magnitud de la traslación de J_i a J'_i gracias a que la geometría del efecto final es un triángulo equilátero con lados definidos e . Esta geometría básica también permite encontrar la distancia desde el centro de la base fija y los actuadores..

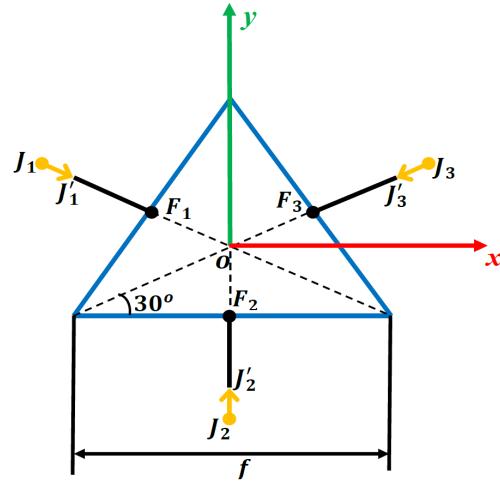


Figura 4.18: Vista frontal de la base fija [44].

$$\|\overrightarrow{\xi_{OF_i}}\| = \frac{f}{2} \tan(30) = \frac{f}{2\sqrt{3}}, i \in \{1, 2, 3\} \quad (4.72)$$

$$\|\overrightarrow{\xi_{J_i J'_i}}\| = \frac{e}{2} \tan(30) = \frac{e}{2\sqrt{3}}, i \in \{1, 2, 3\} \quad (4.73)$$

El movimiento de los brazos está restringido por una junta tipo revoluta, por ende, es fácil calcular la distancia entre el punto F_i y la proyección del punto J_i en el plano que contiene a la base fija:

$$\|\overrightarrow{\xi_{F_i J_i}}\| = L_a \cos(\theta_i), i \in \{1, 2, 3\} \quad (4.74)$$

Según las ecuaciones (4.99), (4.100) y (4.101), se forman tres esferas con centros en los puntos J'_1, J'_2, J'_3 . Las coordenadas de los centros de las esferas se pueden calcular utilizando los vectores de traslación antes mencionados. Las coordenadas de los puntos J'_1, J'_2, J'_3 se dan en las ecuaciones (4.75), (4.76) y (4.77) respectivamente:

$$\vec{J}'_1 = \left[\left(\frac{(f - e)}{2\sqrt{3}} + L_A \cos(\theta_1) \right) \cos(30^\circ), \left(\frac{(f - e)}{2\sqrt{3}} + L_A \cos(\theta_1) \right) \sin(30^\circ), -L_A \sin(\theta_1) \right] \quad (4.75)$$

$$\vec{J}'_2 = \left[0, -\frac{(f - e)}{2\sqrt{3}} - L_A \cos(\theta_2), -L_A \sin(\theta_2) \right] \quad (4.76)$$

$$\vec{J}'_3 = \left[\left(\frac{(f - e)}{2\sqrt{3}} + L_A \cos(\theta_3) \right) \cos(30^\circ), \left(\frac{(f - e)}{2\sqrt{3}} + L_A \cos(\theta_3) \right) \sin(30^\circ), -L_A \sin(\theta_3) \right] \quad (4.77)$$

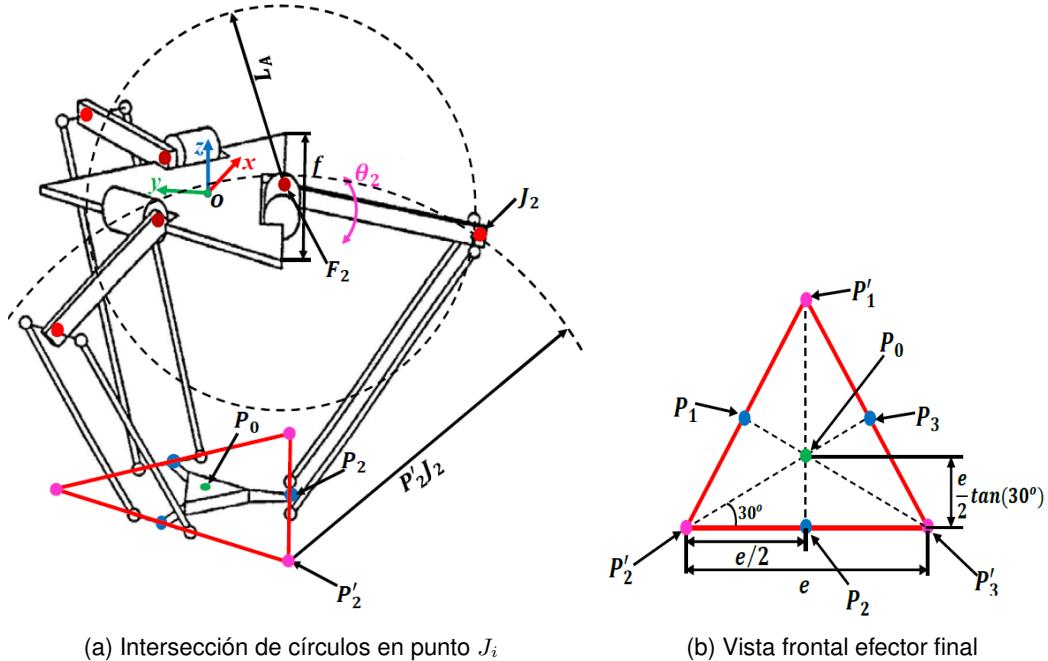
Después de calcular todos los vectores de traslación y las coordenadas de los centros de las esferas, el sistema de ecuaciones (4.99), (4.100) y (4.101) se puede resolver para obtener la posición del efecto final P_{0x}, P_{0y}, P_{0z} . El método algebraico para resolver el sistema de ecuaciones, en otras palabras, la intersección de las 3 esferas trasladadas, es el mismo que en el método A, específicamente el sistema de ecuaciones de la ecuación (4.2).

4.4.1.3. Cinemática inversa

En la cinemática inversa, se calculan los ángulos de los actuadores gracias a la posición dada de efecto final en el espacio cartesiano.

$$P_0 (P_{0x}, P_{0y}, P_{0z}) \rightarrow \theta_1, \theta_2, \theta_3 \quad (4.78)$$

El método de solución de la cinemática inversa en esta sección es el mismo que el empleado para el método A. Lo único que cambia es la nomenclatura de los parámetros y el orden de numeración de los ángulos de los actuadores (figura (4.19)).



(a) Intersección de círculos en punto J_2 (b) Vista frontal efecto final

Figura 4.19: Solución de la cinemática directa de posición del método B [44].

Primero se empieza por calcular en ángulo θ_2 . Se trabaja sobre el marco de referencia YZ , ya que el brazo $\overline{F_2J_2}$ solo puede moverse en ese plano como se muestra en la figura (4.19a). Desde la figura (4.19b) se desprenden las coordenadas de punto $F_2 = \left[0, \frac{-f}{(2\sqrt{3})}, 0\right]$. Sobre el plano YZ se proyecta el antebrazo para obtener una segunda restricción geométrica relacionada con la junta J_2 . En consecuencia, la posición de la junta esférica J_2 está restringida por 2 circunferencias en el plano YZ . Se dibuja el primer círculo con centro F_2 y radio L_A . El segundo círculo se dibuja con el centro en el punto P'_2 y radio $\sqrt{L_B^2 - P_{0x}^2}$, donde el punto $P'_2 = \left[0, P_{0y} - \frac{e}{2\sqrt{3}}, P_{0z}\right]^T$ es la proyección del punto $P_2 = \left[P_{0x}, P_{0y} - \frac{e}{2\sqrt{3}}, P_{0z}\right]^T$ sobre plano YZ y el radio de la circunferencia es calculado de la expresión $\|\vec{\xi}_{P'_2J_2}\| = \sqrt{\|\vec{\xi}_{P_2J_2}\|^2 - \|\vec{\xi}_{P'_2P_2}\|^2} = \sqrt{L_B^2 - P_{0x}^2}$ como se aprecia en la figura (4.20).

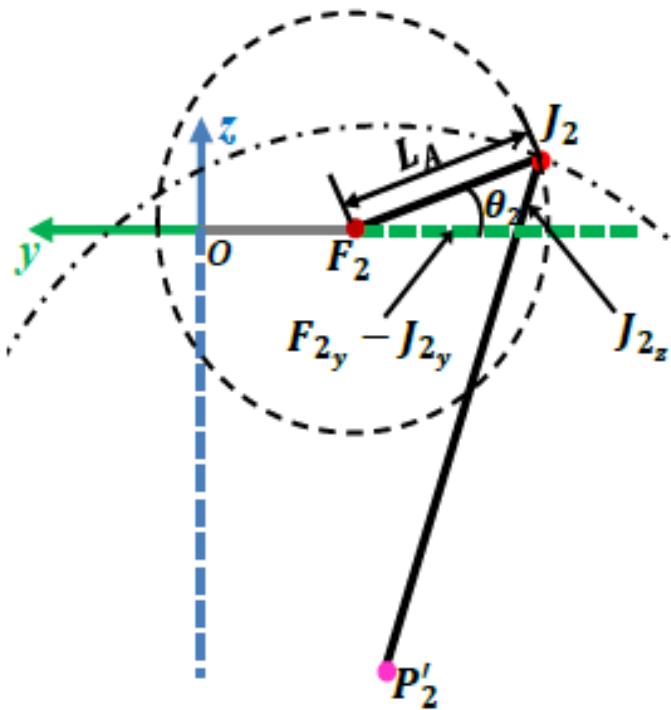


Figura 4.20: Proyección del punto P_2 sobre el plano YZ [44].

La ecuación general de la circunferencia de las ecuaciones (4.79) y (4.80) representan los círculos con radio L_A y $\sqrt{L_B^2 - P_{0x}^2}$ respectivamente, como se muestra en la figura (4.20).

$$\begin{aligned} (J_{2y} - F_{2y})^2 + (J_{2z} - F_{2z})^2 &= L_A^2 \\ \Rightarrow \left(J_{2y} + \frac{f}{2\sqrt{3}} \right)^2 + J_{2z}^2 &= L_A^2 \end{aligned} \quad (4.79)$$

$$\begin{aligned} (J_{2y} - P'_2)_y^2 + (J_{2z} - P'_2)_z^2 &= L_B^2 - P_{0z}^2 \\ \Rightarrow \left(J_{2y} - P_{0y} + \frac{e}{2\sqrt{3}} \right)^2 + (J_{2z} - P_{0z})^2 &= L_B^2 - P_{0z}^2 \end{aligned} \quad (4.80)$$

El valor de J_{2y} y J_{2z} se puede calcular resolviendo simultáneamente la ecuación (4.79) y (4.80) al igual que en el método A. Todas las demás parámetros en este sistema de ecuaciones se asumen como conocidos o se obtienen a partir de la estructura geométrica del robot paralelo delta. Una vez que se conocen estos valores, se puede usar trigonometría simple para calcular el θ_2 como se indica en la siguiente ecuación:

$$\theta_2 = \tan^{-1} \left(\frac{J_{2z}}{F_{2y} - J_{2y}} \right) \quad (4.81)$$

La estructura simétrica del robot paralelo delta da la ventaja de calcular los ángulos θ_1 y θ_3 faltantes aplicando el mismo procedimiento para el ángulo θ_2 . Para ello, se rotar el marco de referencia local en un ángulo de 120° en sentido antihorario y horario para θ_3 y θ_1 respectivamente.

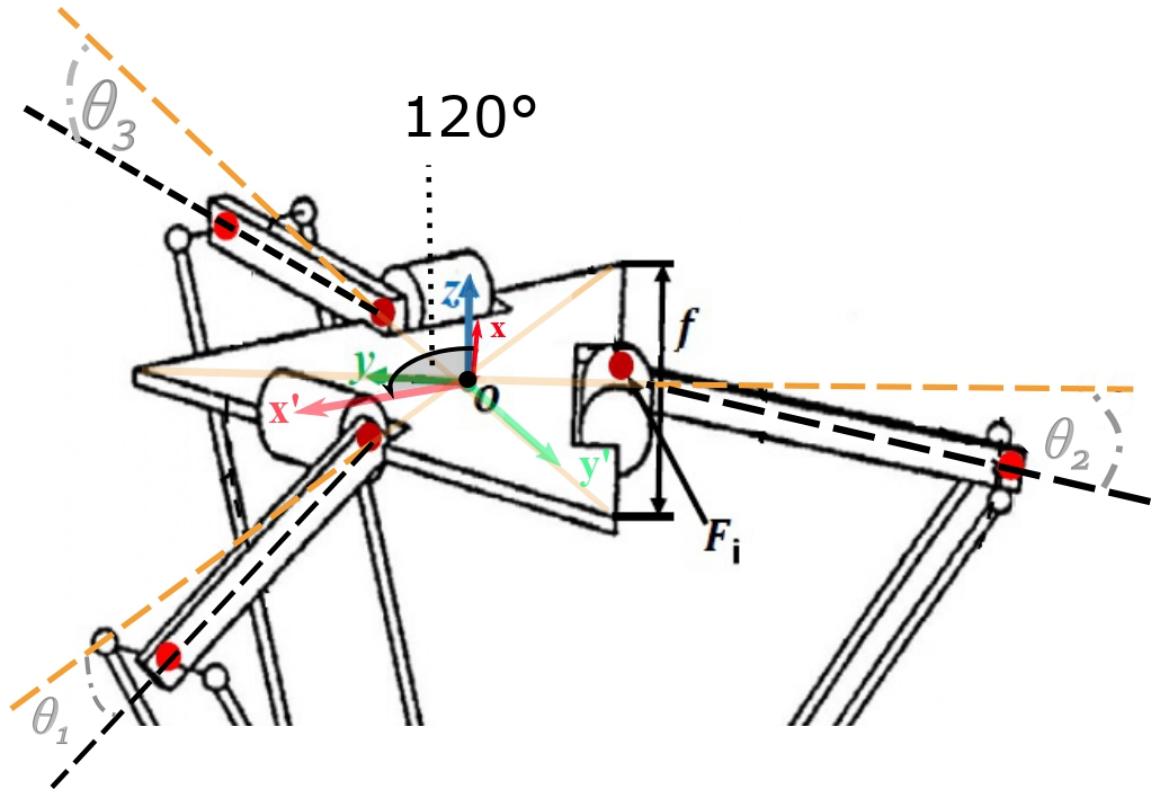


Figura 4.21: Rotación del sistema de referencia local para la solución de la cinemática inversa del método B [44].

4.4.2. Modelación cinemática de la velocidad

Con el objetivo de modelar la cinemática de velocidad del robot delta, en esta sección se implementa las ideas expuestas en el paper [44], con los mismos parámetros, jerga y nomenclatura de la sección (4.4.1). La base de la modelación cinemática de velocidad es determinar la matriz jacobiana J . Esta juega un papel importante en el modelo dinámico del manipulador robótico como se aprecia en secciones posteriores.

La matriz jacobiana para el robot paralelo delta fue calculada por primera vez por Codourey [45]. En este método, las derivadas parciales se calcularon numéricamente. La matriz jacobiana para robots paralelos también se puede calcular vinculando la variable de espacio cartesiano con las variables de espacio de articulación mediante un conjunto de ecuaciones restringidas. Guglielmetti [46] fue la primera persona que aplicó este enfoque para calcular la matriz jacobiana para el robot paralelo Delta. Codourey [47] aborda una versión simplificada de la formulación de Guglielmetti. En esta tesis, se adopta el método de Codourey para calcular la matriz jacobiana del método B.

4.4.2.1. Jacobiano

La matriz jacobiana J describe la relación entre las velocidades cartesianas y las velocidades articulares como se indica en la ecuación (4.82).

$$\vec{\dot{P}_0} = J \vec{\dot{\theta}} \quad (4.82)$$

Para encontrar una expresión de la matriz jacobiana, se utiliza la longitud de los antebrazos de un robot paralelo delta como ecuaciones de restricción.

$$\|\vec{\xi}_{J_i P_i}\|^2 - L_B^2 = 0 \ ; i \in \{1, 2, 3\} \quad (4.83)$$

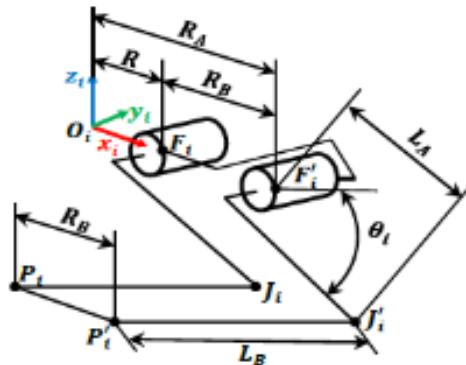


Figura 4.22: Puntos y distancias geométricas para la solución de la cinemática de velocidad del método B [44].

Definiendo en términos de \vec{s}_i el vector $\vec{\xi}_{J_i P_i}$, la ecuación (4.83) se puede escribir como:

$$\vec{s}_i^T \cdot \vec{s}_i - L_B^2 = 0 \quad (4.84)$$

Donde:

$$\vec{s}_i = \vec{P}_0 - (\vec{F}_i + \vec{\xi}_{F_i J_i}) \quad (4.85)$$

$$= \begin{bmatrix} P_{0x} \\ P_{0y} \\ P_{0z} \end{bmatrix} - R_i^R \left(\begin{bmatrix} R \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} L_A \cos(\theta_i) \\ 0 \\ -L_A \sin(\theta_i) \end{bmatrix} \right) \quad (4.86)$$

En la ecuación (4.86), $[P_{0x}, P_{0y}, P_{0z}]$ es la posición del efecto final \vec{P}_0 , y el superíndice R es el marco de referencia local $O - xyz$ como se muestra en la figura (4.22). R es la diferencia entre R_A y R_B . Debido a la simetría en la estructura del robot paralelo delta, cada brazo se puede tratar por separado. Cada brazo está separado por un ángulo de 120° grados y la posición del sistema de referencia correspondiente para cada brazo $O_i - x_i y_i z_i$ es el mismo que el superíndice R pero girado en un ángulo para cada brazo $i \in \{1, 2, 3\}$, respectivamente. La matriz de transformación o la matriz de rotación se indica en la ecuación (4.87).

$$R_i^R = \begin{bmatrix} \cos(\varphi_i) & -\sin(\varphi_i) & 0 \\ \sin(\varphi_i) & \cos(\varphi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.87)$$

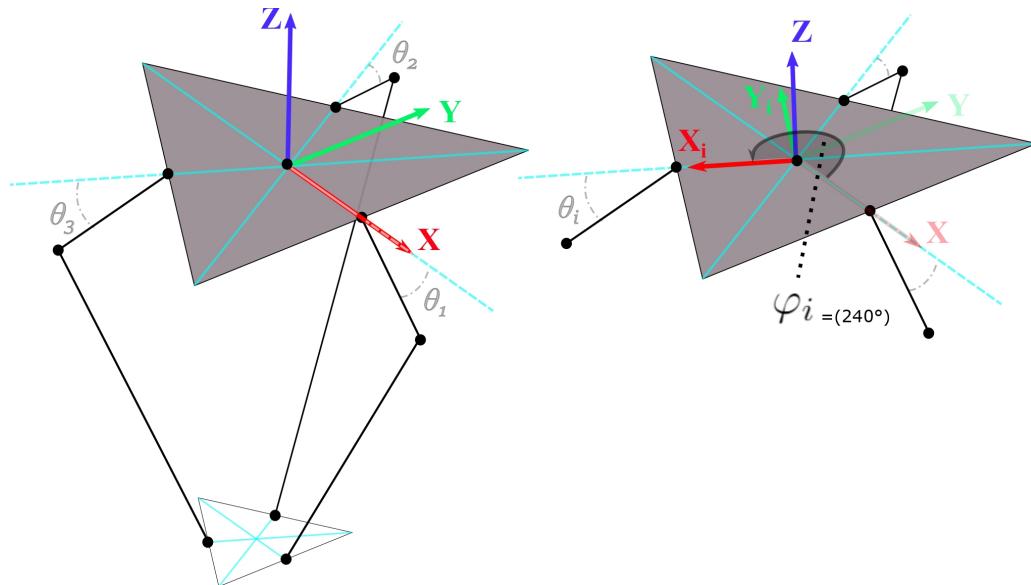


Figura 4.23: Representación gráfica de la rotación del sistema de coordenadas local $O-XYZ$ en los ángulos $\varphi_i \in \{i = 1, 2, 3\}$

Derivando la ecuación (4.84) para obtener una expresión del jacobiano J :

$$\vec{s}_i^T \cdot \vec{\dot{s}}_i = 0 \quad (4.88)$$

Donde la derivada temporal del término \vec{s}_i viene dada por:

$$\vec{\dot{s}}_i = \begin{bmatrix} \dot{P}_{0x} \\ \dot{P}_{0y} \\ \dot{P}_{0z} \end{bmatrix} + R_i^R \begin{bmatrix} L_A \sin(\theta_i) \\ 0 \\ L_A \cos(\theta_i) \end{bmatrix} \dot{\theta}_i = \vec{P}_0 + \vec{b}_i \dot{\theta}_i ; i \in \{1, 2, 3\} \quad (4.89)$$

Remplazando la ecuación (4.89) en la ecuación (4.88):

$$\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix} \vec{P}_0 + \begin{bmatrix} \vec{s}_1^T \cdot \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \cdot \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \cdot \vec{b}_3 \end{bmatrix} \vec{\dot{\theta}}_i = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.90)$$

Reordenando con la finalidad de separar los términos de velocidad lineal del efecto con los de velocidad angular de los actuadores y usando la definición de la ecuación (4.82), el jacobiano J es:

$$J = -J_1 J_2 = - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \begin{bmatrix} \vec{s}_1^T \cdot \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \cdot \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \cdot \vec{b}_3 \end{bmatrix} \quad (4.91)$$

Donde:

$$\vec{b}_i = R_i^R \begin{bmatrix} L_A \sin(\theta_i) \\ 0 \\ L_A \cos(\theta_i) \end{bmatrix} ; i \in \{1, 2, 3\} \quad (4.92)$$

En el caso de los robots en serie, la matriz jacobiana es solo una función de los angulos $\vec{\theta}$, pero para los robots paralelos, la matriz jacobiana depende de la información del espacio articular $\vec{\theta}$ así como de la posición cartesiana del efecto final \vec{P}_0 .

4.4.3. Modelación cinemática de aceleración

En esta sección se presentan las ecuaciones para determinar la aceleración angular de los actuadores del robot delta. La aceleración se determina derivando 2 veces matricialmente la ecuación (4.84) de la sección (4.4.2) de modelación cinemática de velocidad para el método B.

La aceleración angular de los motores del robot delta se calcula con la siguiente formula:

$$\vec{\ddot{\theta}} = J^{-1} \left[\vec{\ddot{P}_0} + \begin{bmatrix} \vec{s_1}^T \\ \vec{s_2}^T \\ \vec{s_3}^T \end{bmatrix}^{-1} * \left(\begin{bmatrix} \vec{\dot{s}_1}^T \\ \vec{\dot{s}_2}^T \\ \vec{\dot{s}_3}^T \end{bmatrix} J + K \right) * \vec{\dot{\theta}} \right] \quad (4.93)$$

Donde:

$$K = \begin{bmatrix} \vec{s_1}^T \vec{b}_1 + \vec{s_1}^T \vec{\dot{b}}_1 & 0 & 0 \\ 0 & \vec{s_2}^T \vec{b}_2 + \vec{s_2}^T \vec{\dot{b}}_2 & 0 \\ 0 & 0 & \vec{s_3}^T \vec{b}_3 + \vec{s_3}^T \vec{\dot{b}}_3 \end{bmatrix} \quad (4.94)$$

$$\vec{\dot{b}}_i = \begin{bmatrix} L_A \cos(\theta_i) \\ 0 \\ -L_A \sin(\theta_i) \end{bmatrix} \vec{\dot{\theta}}_i \quad (4.95)$$

$$\vec{\dot{s}}_i = \begin{bmatrix} \dot{P}_{0x} \\ \dot{P}_{0y} \\ \dot{P}_{0z} \end{bmatrix} + R_i^R * \begin{bmatrix} L_A \sin(\theta_i) \\ 0 \\ L_A \cos(\theta_i) \end{bmatrix} \dot{\theta}_i = \vec{\dot{P}_0} + \vec{b}_i \dot{\theta}_i \quad (4.96)$$

4.4.4. Modelación dinámica

En esta sección se presenta un método para resolver la dinámica inversa de un robot delta inspirado en los trabajos de Zeeshan Shareef [44], Collins F. Adetu, Carl A. Moore, Jr. y Rodney G. Roberts [48]. La modelacion dinámica utiliza como base el principio de trabajo virtual desarrollado por Codourey [49].

4.4.4.1. Introducción a modelos dinámicos

El modelo dinámico de manipuladores robóticos juega un papel importante en la optimización de la trayectoria. La principal dificultad para calcular el modelo dinámico es que debe ser realizable y que se pueda calcular en tiempo real. Una forma sencilla de resolver el modelo dinámico de manipuladores paralelos es la cadena cerrada en uniones pasivas. Esta simplificación proporciona relajación en las condiciones de cierre. El modelo dinámico final será la suma de todos los robots individuales así creados. Kleinfinger [50] utiliza esta técnica y aplicó los multiplicadores de Lagrange para calcular el modelo dinámico.

Otro método muy útil para derivar el modelo dinámico de robótica se basa en el principio de trabajo virtual. Según este principio, la contribución de todas las fuerzas inerciales debe ser igual a la contribución de todas las fuerzas no inerciales. Este método simplifica el problema y es igualmente eficiente tanto para robots en serie como en paralelo [45], [51], [52].

Otra forma de derivar el modelo dinámico es utilizar el principio de trabajo virtual de Lagrange-d'Alembert. Kokkinis y Stoughton [53], Nakamura [54] y Wang y Chen [55] han obtenido con éxito un modelo dinámico que utiliza el principio de trabajo virtual de Lagrange-d'Alembert.

Al incorporar todas las masas, la inercia y no linealidades, el modelo dinámico se vuelve muy complicado y computacionalmente costoso, como consecuencia no puede usarse para aplicaciones en tiempo real. Se puede derivar un modelo dinámico practicable para aplicaciones en tiempo real despreciando las masas y la inercia de los antebrazos, propuesto por Ji [56].

Esta tesis se deriva el modelo dinámico utilizando el principio de trabajo virtual presentado por Codourey [49].

4.4.4.2. Principio de trabajo virtual

Para obtener el torque de los actuadores se emplea el principio del trabajo virtual. El principio establece que, en el equilibrio, el trabajo virtual δW , realizado por todas las fuerzas externas F que actúan sobre un cuerpo durante cualquier desplazamiento virtual δr , consistente con las restricciones estructurales impuestas al cuerpo, es igual a cero. Este principio se ilustra matemáticamente en la ecuación (4.97).

$$\delta W = \sum_{i=1}^N F_i * \delta r_i = 0 \quad (4.97)$$

En la ecuación (4.97) solo se consideran las fuerzas externas, todas las fuerzas internas, es decir, las fuerzas de restricción y reacción se ignoran porque estas fuerzas no realizan ningún trabajo virtual. El principio de trabajo virtual se utiliza tradicionalmente para resolver problemas estáticos. Sin embargo, para un sistema que no está en reposo, la fuerza (fuerza de inercia) como resultado de la masa del cuerpo m , que acelera a una tasa a , se incluye en la ecuación (4.97). Esta extensión del principio de trabajo virtual para casos dinámicos se conoce como principio de D'Alembert [48]. La ecuación (4.98) es una extensión de la ecuación (4.97) con la fuerza de inercia incluida:

$$\delta W = \sum_{i=1}^N (F_i - m_i a_i) * \delta r_i = 0 \quad (4.98)$$

Un cuerpo rígido que es capaz de realizar movimientos tanto de traslación como de rotación, por lo tanto la ecuación (4.98) generalmente se escribe como:

$$\delta W = \sum_{i=1}^N \left[(F_i - m_i a_i) * \delta r_i + (\tau - I \ddot{\theta}) * \delta \theta \right] = 0 \quad (4.99)$$

donde, τ es el par externo que actúa sobre el cuerpo, I es el momento de inercia, $\ddot{\theta}$ la aceleración angular y $\delta \theta$ es el desplazamiento angular virtual.

4.4.4.3. Simplificaciones e hipótesis

Antes de proceder a derivar el modelo dinámico, existen unas pocas hipótesis para hacer que el modelo sea factible y computacionalmente eficiente. Las hipótesis simplificadoras son:

- Debido a que los antebrazos se construyen con materiales ligeros, es posible simplificar el problema dinámico ignorando la inercia rotacional de las varillas paralelas, es decir, se desprecia la inercia rotacional del antebrazo.
- Se ignoran los efectos de fricción entre las piezas y la elasticidad de los materiales.
- Para fines analíticos, las masas del antebrazo se dividen equitativamente y se colocan en las extremidades. Por lo tanto, la mitad de la masa de la barra está centrada en la extremidad superior (es decir, la junta que une el brazo con el antebrazo), mientras que la otra mitad está centrada en la extremidad inferior (es decir, la junta que une los antebrazos con el efecto final). Esta simplificación se muestra en la figura (4.24).

Debido a estas hipótesis, el robot paralelo delta se reduce a los tres brazos y al efecto final.

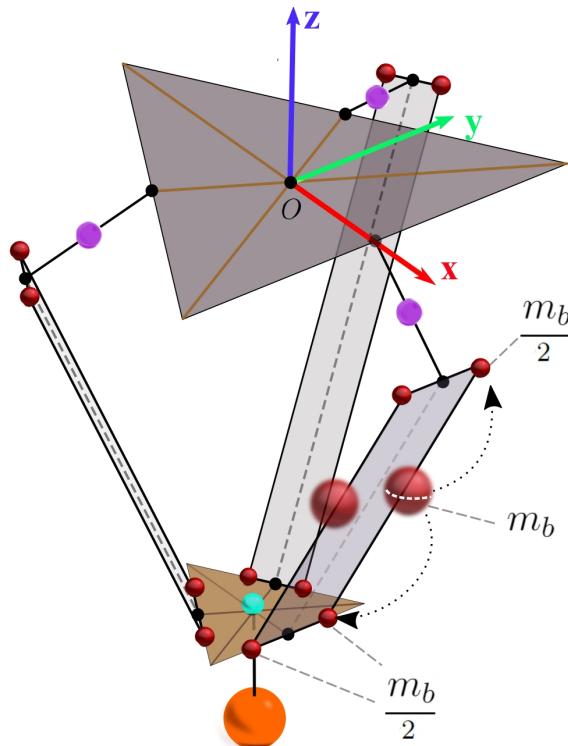


Figura 4.24: Simplificación de masas para la solución de la dinámica inversa del método B.

4.4.4.4. Nomenclatura de parámetros geométricos, simplificación de masas y sistema de referencia local

En la tabla (4.14), la figura (B.1) y la figura (4.26) representa la nomenclatura de los parámetros principales para resolver el problema dinámico del robot delta y la simplificación de masas:

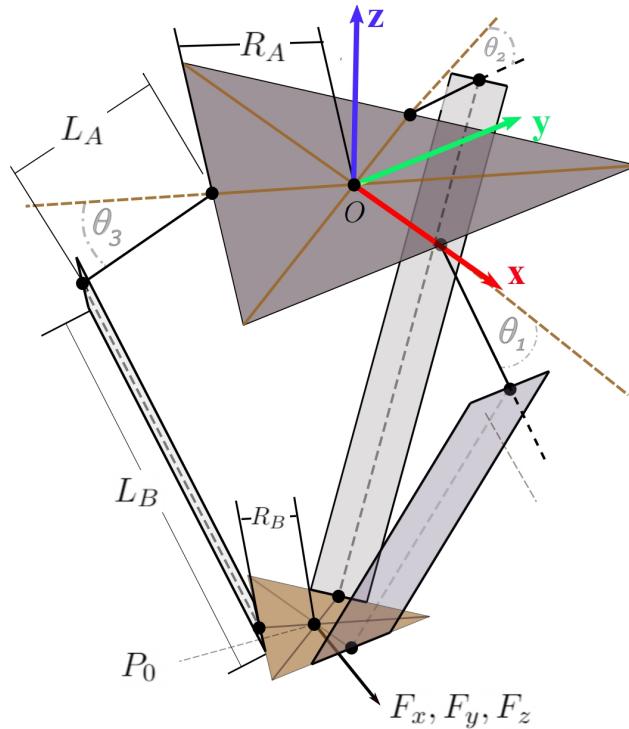


Figura 4.25: Sistema de referencia local y parámetros geométricos para la solución de la dinámica inversa del método B.

Parámetro	Descripción
L_A	Largo del brazo
L_B	Largo del antebrazo
R_A	Distancia entre el centro de la base fija y la junta revoluta o actuador
R_B	Distancia entre el centro del efecto final a la junta que lo une con el antebrazo
$O - xyz$	Sistema de referencia local para resolver el problema dinámico
F_x, F_y, F_z	Fuerzas externas sobre efecto final
θ_i	Ángulo del actuador $i \in \{1, 2, 3\}$
P_0	Centroide del efecto final

Tabla 4.14: Parámetros para la solución de la dinámica inversa del método B.

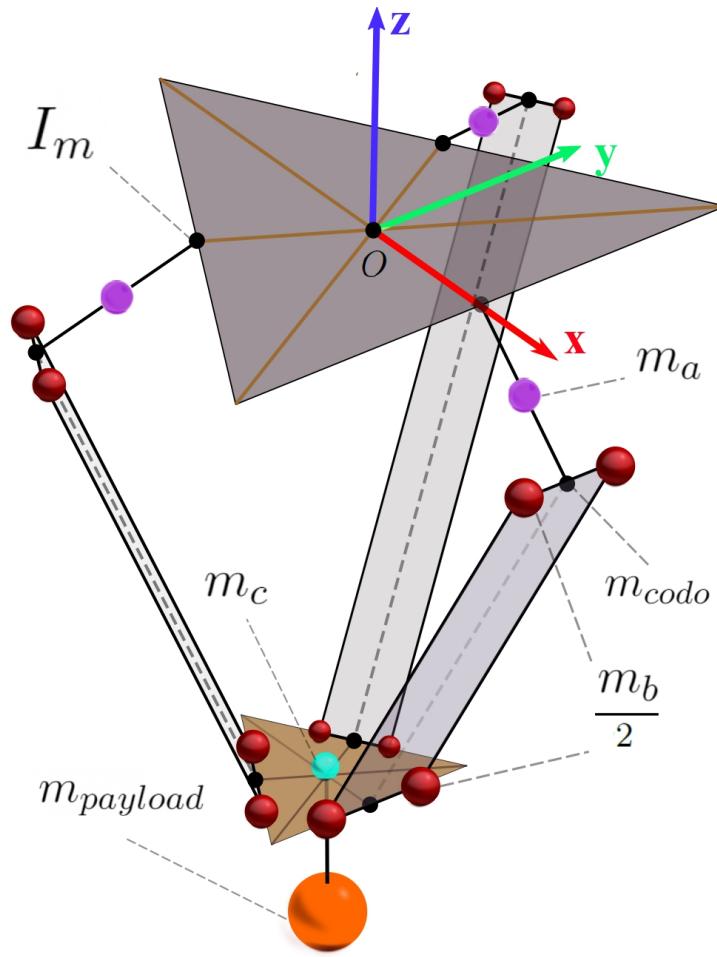


Figura 4.26: Sistema de referencia local y masas para la solución de la dinámica inversa del método B.

Parametro	Descripción
m_a	Masa del brazo
m_b	Masa del antebrazo solo una varilla
m_c	Masa de la plataforma móvil
$m_{payload}$	Masa de objeto adherida a la plataforma móvil que se desea desplazar en una tarea particular
m_{codo}	Masa de la junta esférica que unen los brazos con los antebrazos
r	Relación de división de masas de los antebrazos para simplificación dinámica
I_m	Inercia de los motores

Tabla 4.15: Masas, relación de división de masas e inercia de los motores para la solución de la dinámica inversa del método B.

4.4.4.5. Dinámica inversa

Aplicando el principio de trabajo virtual a la simplificación del robot delta, se determinan los torques de los actuadores $\vec{\tau} = [\tau_1, \tau_2, \tau_3]^T$ con la siguiente expresión:

$$\vec{\tau} = I_b \vec{\ddot{\theta}} + J^T m_{nt} \vec{P}_0 - J^T \vec{F}_g - \vec{\tau}_{Gb} \quad (4.100)$$

Sustituyendo $\vec{P}_0 = J \vec{\dot{\theta}} + J \vec{\dot{\theta}}$

$$\vec{\tau} = (I_b + J^T m_{nt} J) \vec{\ddot{\theta}} + (J^T m_{nt} J) \vec{\dot{\theta}} + (-J^T \vec{F}_g - \vec{\tau}_{Gb}) \quad (4.101)$$

A partir de la ecuación (4.101), se identificar fácilmente la matriz de masa $M(\theta)$, la matriz $C(\theta, \dot{\theta})$ de coeficiente de Coriolis y centrífuga, y el vector de términos de gravedad $\vec{G}(\theta)$ comparándolo con la dinámica estándar:

$$\vec{\tau} = M(\theta) \vec{\ddot{\theta}} + C(\theta, \dot{\theta}) \vec{\dot{\theta}} + \vec{G}(\theta) \quad (4.102)$$

Dónde:

$$M(\theta) = I_b + J^T m_{nt} J \quad (4.103)$$

$$C(\theta, \dot{\theta}) = J^T m_{nt} J \quad (4.104)$$

$$\vec{G}(\theta) = -J^T \vec{F}_g - \vec{\tau}_{Gb} \quad (4.105)$$

$$I_b = \begin{bmatrix} I_{b1} & 0 & 0 \\ 0 & I_{b2} & 0 \\ 0 & 0 & I_{b3} \end{bmatrix} \quad (4.106)$$

$$I_{bi} = I_m + L_A^2 \left(\frac{m_a}{3} + m_{codo} + 2 * r * m_b \right) \quad (4.107)$$

$$\vec{\ddot{\theta}} = [\ddot{\theta}_1 \ddot{\theta}_2 \ddot{\theta}_3]^T \quad (4.108)$$

$$m_{nt} = m_c + m_{payload} + 3 * 2 * (1 - r) m_b \quad (4.109)$$

$$\vec{P}_0 = [\ddot{P}_{0x} \ddot{P}_{0y} \ddot{P}_{0z}]^T \quad (4.110)$$

$$\vec{F}_g = m_{nt} [00 - g]^T \quad (4.111)$$

$$\vec{\tau}_{Gb} = g * CoM * (m_a + m_{codo} + 2 * r * m_b) [\cos(\theta_1) \cos(\theta_2) \cos(\theta_3)]^T \quad (4.112)$$

$$CoM = L_A \frac{\frac{1}{2}m_a + m_{codo} + 2 * r * m_b}{m_a + m_{codo} + 2 * r * m_b} \quad (4.113)$$

El detalle de esta sección se encuentra en el anexo (A)

4.5. Espacio de trabajo

En esta sección se da a conocer la definición de espacio de trabajo por algunos científicos que se han especializado en esta problemática para robots, se presentan de manera general los 2 métodos más utilizados para determinar el espacio y se muestran las restricciones de un robot delta relacionadas con este tema.

4.5.1. Definición

En los últimos años, los robots que incluyen una estructura paralela atrajeron la atención de los investigadores del mundo académico. Entre los robots con estructura paralela más famosos se encuentran los provistos de la estructura paralela delta de 3 grados de libertad. Al comparar los robots que incluyen manipuladores de estructuras en serie con los que incluyen manipuladores de estructuras paralelas, se puede notar que: la estructura paralela tiene una lista de ventajas, como alta rigidez, disponibilidad para transportar objetos más pesados, posicionamiento más preciso, etc. Estas ventajas también vienen dadas por el hecho de que las fuerzas iniciales y de gravedad del objeto manipulado son absorbidas por cada enlace cinemático [57][58]. Las desventajas son: espacio de trabajo más estrecho y un control más difícil [58].

El espacio de trabajo de un robot se define como la región en el espacio cartesiano tridimensional que puede ser alcanzada por un punto de su efecto final [59], es decir, en el caso de un robot delta, es la región en el espacio tridimensional que puede alcanzar el punto central de su plataforma móvil.

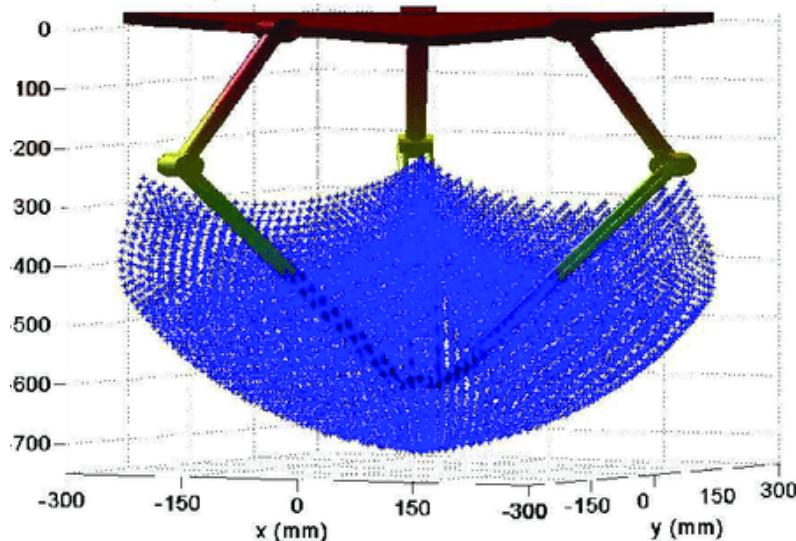


Figura 4.27: Aproximación del espacio de trabajo para un robot delta [60]

4.5.2. Métodos

El espacio de trabajo de los robots ha sido estudiado intensamente a lo largo de los años por varios investigadores. Los espacios de trabajo más comunes, según Merlet, son: el espacio de trabajo de traducción, el espacio de trabajo de orientación, el espacio de trabajo accesible, el espacio de trabajo de orientación inclusiva, el espacio de trabajo de orientación total, el espacio de trabajo de destreza, el espacio de trabajo total con orientación reducida [57] y [61].

Se han utilizado básicamente las siguientes categorías de métodos para determinar el espacio de trabajo: métodos geométricos y métodos de digitalización, siendo este último el más usado. El método geométrico [62] se basa en obtener un objeto geométrico que describa todas las posibles posiciones del efecto final y que satisfagan las restricciones del robot. Se obtiene un objeto geométrico para cada cadena cinemática del robot y el espacio de trabajo es la intersección de los objetos geométricos obtenidos. El método de discretización [62], 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 (Ottaviano and Ceccarelli, 2000). 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.

En esta tesis se basa en el método de discretización, sin embargo, será modificado para que el algoritmo calcule la cantidad exacta de los puntos en el espacio que es capaz de alcanzar la plataforma móvil del robot [62]. 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 cartesiano del centro de la plataforma móvil. El punto pertenece al espacio de trabajo del robot sí y solo sí cumple con todas las restricciones impuestas.

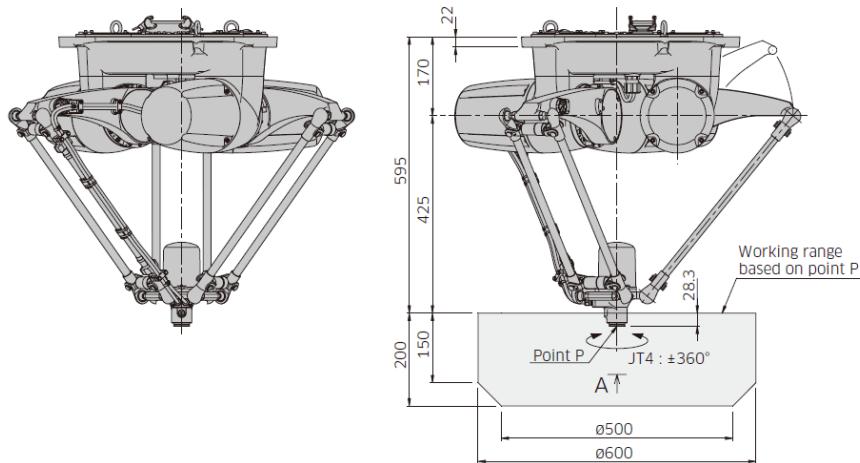


Figura 4.28: Espacio de trabajo YF003N [63]

4.5.3. Tipos de restricciones

El movimiento de los robots manipuladores dentro del espacio de trabajo puede estar restringido por varios factores, tales como los límites constructivos de los acoplamientos cinemáticos pasivos, los límites dados por los dispositivos de accionamiento de los acoplamientos cinemáticos activos, las cohesiones dadas por los elementos constructivos del robot, así como por puntos o áreas de singularidad que pueden dividir el espacio de trabajo en varios componentes [57].

Las restricciones que se toman en cuenta en este documento para determinar el espacio de trabajo del robot delta son:

- Límites impuesto en los ángulos de los actuadores $[\theta_{1i,min} - \theta_{1i,max}]$ para cada actuador $i \in \{1, 2, 3\}$.
- Resolución basada en tamaño del paso de los actuadores, es decir, la discretización del rango impuesto por los límites del punto anterior $\Delta\theta_{1i}$ para cada actuador $i \in \{1, 2, 3\}$.
- Restricciones de ángulos internos θ_{2i} y θ_{3i} en base a restricción de las juntas o rotulas.
- Singularidades que se determinan mediante el determinante del jacobiano $J = J_x^{-1}J_\theta$ cuando este es cercano a 0. Estas singularidades son las mismas explicadas en la sección (4.3.2.4).
- Límites comúnmente impuestos por los fabricantes. Generalmente son volúmenes geométricos como cilindros o paralelepípedos.

En la figura (4.29) se visualizan de mejor manera las 5 restricciones impuestas.

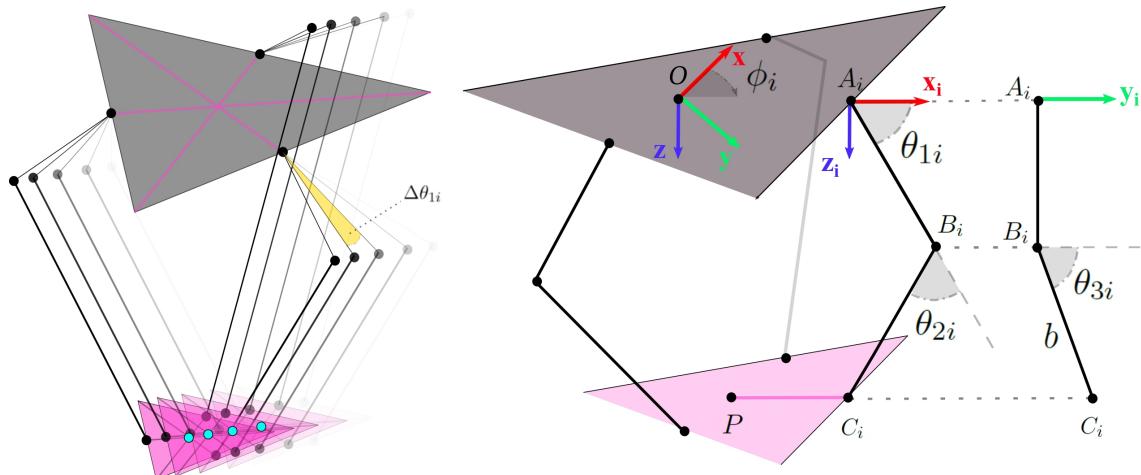


Figura 4.29: 2 tipos de restricciones del espacio de trabajo

4.6. Trayectorias

El objetivo principal de esta sección es explicar de manera general la implementación de las trayectorias en robots y dar a conocer detalladamente la trayectoria que se utiliza en este trabajo de grado.

En primer lugar, se define que es una trayectoria, se exponen sus objetivos y se propone un diagrama de flujo con relación a los pasos que se realizan para obtenerlas.

En segundo lugar, se presentan 5 clasificaciones de trayectorias más comunes por los científicos y expertos en robótica para todo tipo de robots.

Finalmente, se describe la trayectoria como la combinación de 2 partes: una descripción puramente geométrica de la secuencia de configuraciones logradas por el robot y una escala de tiempo que especifica los tiempos en que se alcanzan esas configuraciones. A partir del punto de vista anterior, se consideran el caso de las trayectorias punto a punto tanto en el espacio de articulaciones como en el espacio de cartesianas.

4.6.1. Definición de trayectorias

Un camino geométrico p es un conjunto de puntos en el espacio articular o cartesiano que un manipulador de un robot debe seguir, en otras palabras, es una descripción puramente geométrica. La ecuación (4.114) representa un camino geométrico en el espacio cartesiano:

$$p(s) = [x(s), y(s), z(s)]^T \quad (4.114)$$

Una trayectoria es un camino geométrico $p(s)$ más consideraciones temporales $s(t)$. Estas consideraciones temporales pueden estar restringidas por velocidades o aceleraciones impuestas a lo largo del camino. Usualmente se escoge un camino y luego se escoge la ley temporal para la trayectoria.

$$p(s(t)) = [x(s(t)), y(s(t)), z(s(t))]^T \quad (4.115)$$

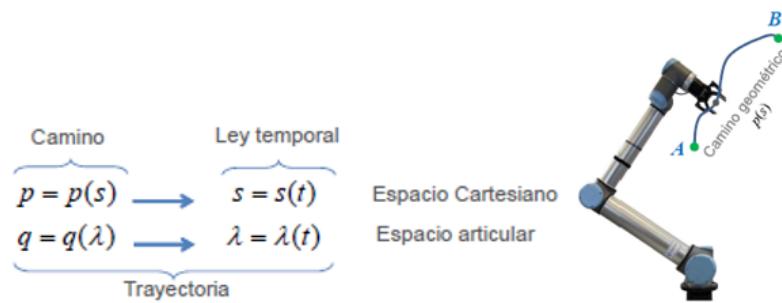


Figura 4.30: Descomposición de trayectoria en camino + ley temporal [64]

4.6.2. Nomenclatura de trayectorias

Un camino geométrico $\theta(s)$ esta en función de un parámetro de camino escalar s , que asume que es 0 al comienzo del camino y 1 al final, en un punto en el espacio de configuración del robot Θ .

$$\theta : s \rightarrow \Theta \quad (4.116)$$

$$\theta : [0; 1] \rightarrow \Theta \quad (4.117)$$

A medida que s aumenta de 0 a 1, el robot se mueve a lo largo de la trayectoria. A veces, s se toma como tiempo y se permite que varíe desde el tiempo $s = 0$ hasta el tiempo total de movimiento $s = T$, pero a menudo es útil separar el papel del parámetro de trayectoria geométrica s del parámetro de tiempo t . Una escala de tiempo $s(t)$ se le asigna un valor s a cada tiempo $t \in [0; T]$:

$$s : t \rightarrow [0; 1] \quad (4.118)$$

$$s : [0; T] \rightarrow [0; 1] \quad (4.119)$$

Juntos, un camino geométrico $\theta(s)$ y una escala de tiempo $s(t)$ definen una trayectoria $\theta(s(t))$ o $\theta(t)$ para abreviar. Usando la regla de la cadena, la velocidad y la aceleración a lo largo de la trayectoria se pueden escribir como:

$$\dot{\theta} = \frac{d\theta}{ds} \dot{s} \quad (4.120)$$

$$\ddot{\theta} = \frac{d\theta}{ds} \ddot{s} + \frac{d^2\theta}{ds^2} \dot{s}^2 \quad (4.121)$$

Para asegurar que la aceleración del robot (y por lo tanto la dinámica) esté bien definida, $\theta(s)$ y $s(t)$ deben ser dos veces diferenciales.

4.6.3. Objetivo y procedimiento de generación de trayectorias

El control cinemático en robótica es una herramienta que permite establecer cuáles son las trayectorias que debe seguir cada articulación del robot a lo largo del tiempo para conseguir los objetivos fijados por el usuario, tales como:

- Punto de destino
- Tipo de trayectoria del extremo
- Tiempo total invertido

Para ello es necesario tomar en cuenta las restricciones físicas de los accionamientos y criterios de calidad tales como sensibilidad, precisión, repetitividad, etc.

Un procedimiento típico para la generación de trayectorias en robots es el que se aprecia en la figura (4.31):

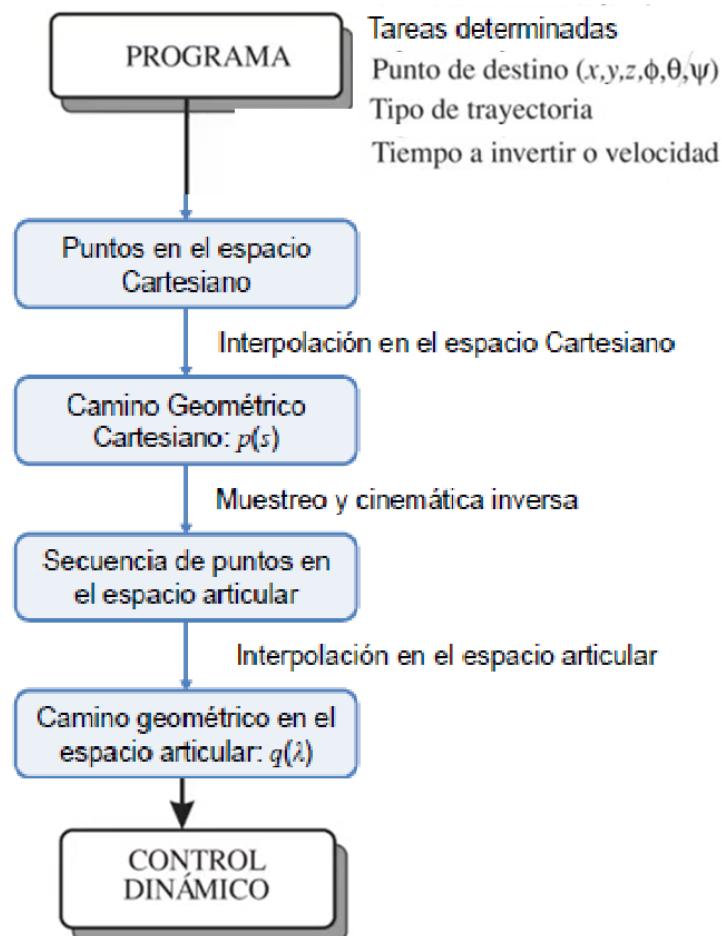


Figura 4.31: Procedimiento típico para generar trayectorias [64]

4.6.4. Clasificación de trayectorias

En esta sección se presentan 5 clasificaciones mas utilizadas para agrupar las trayectorias de todo tipo de robots.

4.6.4.1. Segundo el espacio

Según la importancia que se le asigna al camino de la trayectoria de un brazo robótico o un efecto final de un punto inicial a un punto final en una tarea específica, se pueden dividir en 2 grupos: trayectorias cartesianas y trayectorias articulares. Las trayectorias cartesianas se utilizan cuando es necesario que el robot siga una determinada trayectoria geométrica. Acerca de estas trayectorias, es importante recalcar que sirven para evitar obstáculos, la visualización del camino generado es más fácil y requiere de cinemática inversa. Por el contrario, las trayectorias articulares se ocupan cuando se requiere ir de un punto a otro sin importar la trayectoria.

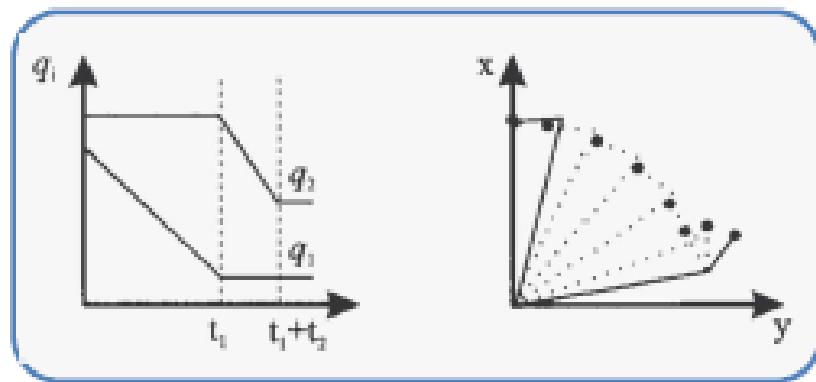


Figura 4.32: Interpolación articular no coordinada [64]

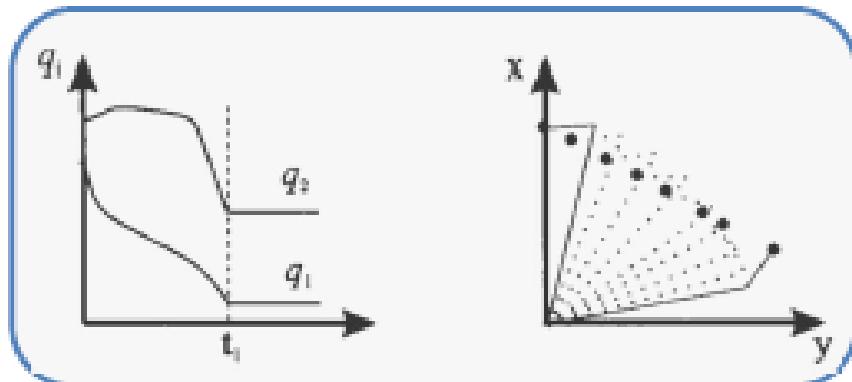


Figura 4.33: Interpolación en el espacio cartesiano [64]

4.6.4.2. Según la geometría del camino

El punto de vista principal de esta clasificación es en la forma de la función que representa la trayectoria, ya sea cartesiana o articular. Existen muchos tipos, algunos de estos son:

- Trayectorias rectilíneas
- Trayectorias polinomiales
- Trayectorias exponenciales
- Trayectorias cicloides

- Lineal: $q = a_1t + a_0 \leftarrow$ 2 parámetros (a_1, a_0)
Se requiere 2 ecuaciones para hallar a_1, a_0
- Cuadrático: $q = a_2t^2 + a_1t + a_0 \leftarrow$ 3 parámetros (a_2, a_1, a_0)
Se requiere 3 ecuaciones
- Cúbico: $q = a_3t^3 + a_2t^2 + a_1t + a_0 \leftarrow$ 4 parámetros (a_3, a_2, a_1, a_0)
Se requiere 4 ecuaciones
- Grado 4: $q = a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \leftarrow$ 5 parámetros (a_4, a_3, a_2, a_1, a_0)
Se requiere 5 ecuaciones
- Grado 5: $q = a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \leftarrow$ 6 parámetros ($a_5, a_4, a_3, a_2, a_1, a_0$), se requiere 6 ecuaciones
- Grado n : $q = a_nt^n + a_{n-1}t^{n-1} + \cdots + a_1t + a_0 \leftarrow$ n parámetros ($a_n, a_{n-1}, \dots, a_1, a_0$)
Se requiere $n+1$ ecuaciones

Figura 4.34: Polinomios [64]

4.6.4.3. Según la ley temporal

Esta clasificación se basa en las especificaciones de puntos en el camino (velocidades, posición donde detenerse), restricciones impuestas por los actuadores o tareas específicas (máximo torque, máxima velocidad) o en considerar criterios de optimización (mínimo tiempo, mínima energía o una combinación de ambos). Ejemplos de aquellas son:

- Trayectorias tipo bang-bang (on/off) en aceleración
- Trayectorias trapezoidales en velocidad
- Trayectorias polinomiales

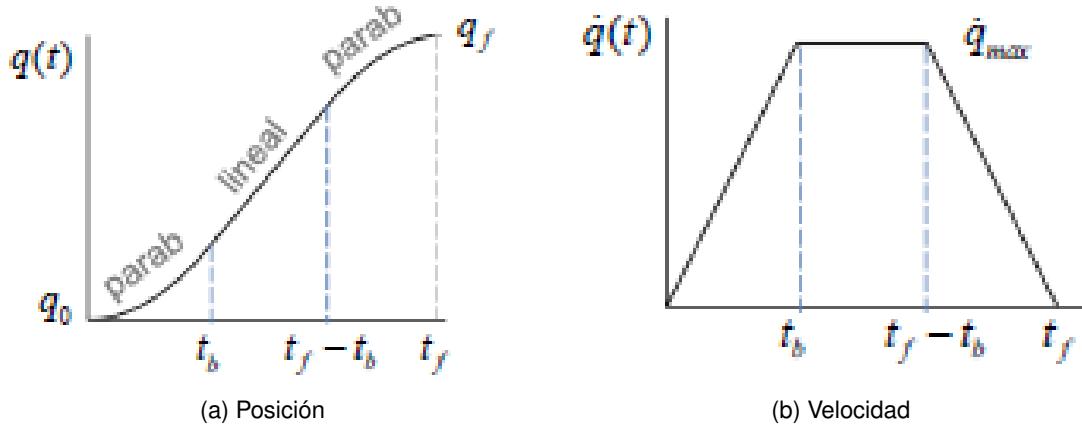


Figura 4.35: Trayectorias punto a punto : Interpolador de Velocidad Trapezoidal [64]

4.6.4.4. Según la coordinación

Esta categoría toma toda la atención a la sincronía que tienen las articulaciones en un desplazamiento de un robot de un punto a otro. Se puede dividir en dos grupos: trayectorias coordinadas e independientes. Las trayectorias coordinadas se refieren a que todas las articulaciones inician y terminan el movimiento al mismo tiempo y en simultáneo, mientras que las trayectorias independientes el movimiento de cada articulación es independiente.

4.6.4.5. Segundo el tipo de tarea

Dependiendo de la tarea que se quiera realizar, se pueden subdividir en 3 tipos las trayectorias según los puntos por los que debe recorrer el efecto final o el ultimo brazo del robot en el espacio cartesiano o articular:

- Trayectorias punto a punto: ir de un punto a otro sin importar la trayectoria.
- Trayectorias de puntos vías: ir de un punto a otro, pero pasando por puntos intermedios.
- Trayectorias continuas (continuidad de velocidad, aceleración): ir de un punto a otro por una trayectoria específica teóricamente de puntos infinitos.

4.6.5. Trayectorias punto a punto

El tipo de movimiento más simple es desde el reposo en una configuración hasta el reposo en otra. A esto lo llamamos movimiento de punto a punto. El tipo de trayectoria más simple para el movimiento de punto a punto es una línea recta. Las rutas en línea recta y sus escalas de tiempo se analizan a continuación. Las ideas en esta sección son extraídas del libro creado por los académicos de la Universidad de Northwestern [65].

4.6.5.1. Trayectorias en Línea Recta

Una línea recta comienza con una configuración inicial θ_{inicio} hasta una configuración final θ_{fin} que pueden ser definidas en espacio de articulaciones o en espacio cartesiano. La línea recta se puede escribir:

$$\theta(s) = \theta_{\text{inicio}} + s(\theta_{\text{fin}} - \theta_{\text{inicio}}); s \in [0, 1] \quad (4.122)$$

Sus derivadas son:

$$\frac{d\theta}{ds} = \theta_{\text{fin}} - \theta_{\text{inicio}} \quad (4.123)$$

$$\frac{d^2\theta}{ds^2} = 0 \quad (4.124)$$

Las líneas rectas en el espacio articular generalmente no producen un movimiento en línea recta del efecto final en el espacio cartesiano. Si se desean movimientos en línea recta en el espacio cartesiano, X_{inicio} y X_{fin} pueden especificar las configuraciones de inicio y finalización. Si X_{inicio} y X_{fin} están representados por un conjunto mínimo de coordenadas, entonces una línea recta se define como:

$$X(s) = X_{\text{inicio}} + s(X_{\text{fin}} - X_{\text{inicio}}); s \in [0, 1] \quad (4.125)$$

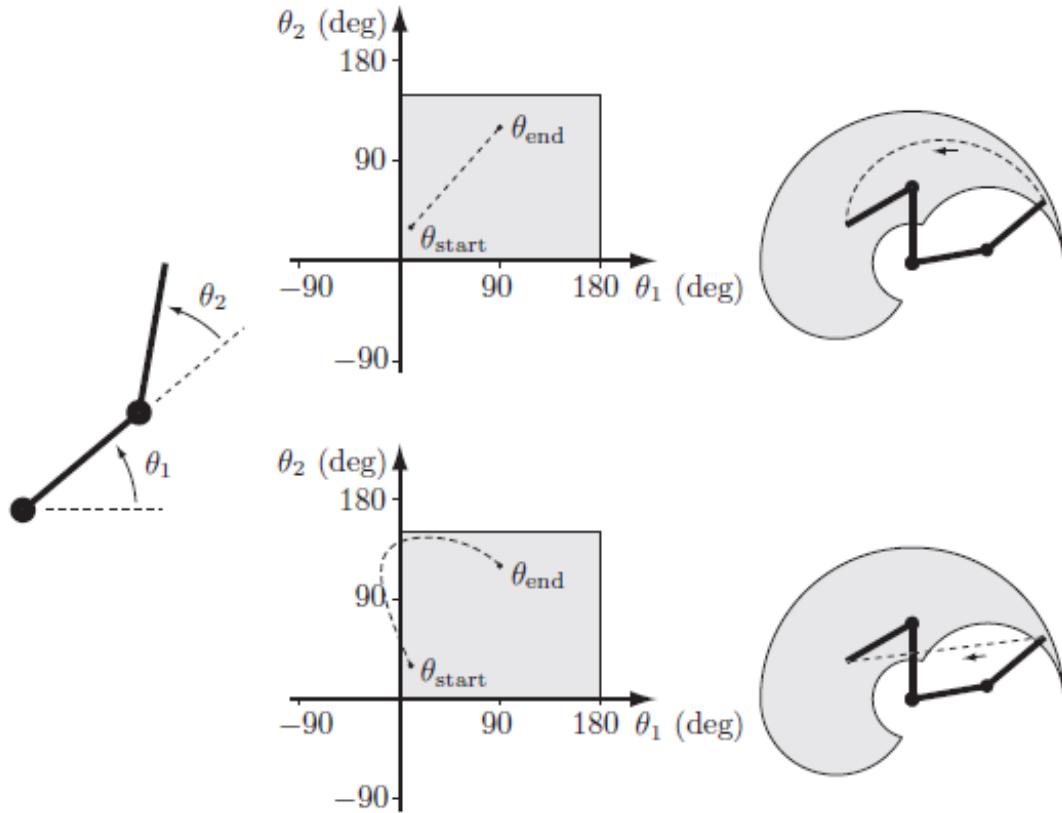


Figura 4.36: (Izquierda) Un robot 2R con límites de articulación $0^\circ \leq \theta_1 \leq 180^\circ$, $0^\circ \leq \theta_2 \leq 150^\circ$. (Centro superior) Una trayectoria en línea recta en el espacio articular y (arriba a la derecha) el movimiento correspondiente del efecto final en el espacio de la tarea (línea discontinua). Las configuraciones de punto final alcanzables, sujetas a límites de articulación, se indican en gris. (Centro inferior) Esta línea curva en el espacio de la articulación y (parte inferior derecha) la trayectoria de la línea recta correspondiente en la línea discontinua del espacio de la tarea) violaría los límites de la articulación. [65]

En comparación con el caso en el que se utilizan coordenadas articuladas, se deben abordar las siguientes cuestiones:

- Si el camino pasa cerca de una singularidad cinemática, las velocidades de las articulaciones pueden volverse excesivamente grandes para la mayoría de las escalas de tiempo del camino.
- Dado que el espacio de trabajo en el que se desenvuelve el efecto final de un robot puede no ser convexo en coordenadas X , algunos puntos en una línea recta entre dos puntos finales alcanzables pueden no ser accesibles.

4.6.5.2. Escala temporal de un camino en línea recta

Una escala de tiempo $s(t)$ de una trayectoria debe garantizar que el movimiento sea lo suficientemente suave y que se satisfaga cualquier restricción sobre la velocidad y aceleración del robot. Para una trayectoria en línea recta en el espacio articular de la forma de la ecuación (4.122), las velocidades y aceleraciones articulares escaladas en el tiempo son respectivamente:

$$\dot{\theta} = \dot{s} (\theta_{fin} - \theta_{inicio}) \quad (4.126)$$

$$\ddot{\theta} = \ddot{s} (\theta_{fin} - \theta_{inicio}) \quad (4.127)$$

Para un camino en línea recta en el espacio cartesiano parametrizado por un conjunto mínimo de coordenadas $X \in \Re^m$, simplemente se reemplaza $\theta, \dot{\theta}, \ddot{\theta}$ por X, \dot{X}, \ddot{X} .

4.6.5.2.1. Escala de tiempo polinomial Una forma conveniente para la escala de tiempo $s(t)$ es un polinomio cúbico en función del tiempo:

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \quad (4.128)$$

Un movimiento de punto a punto desde el tiempo 0 al T imponer las restricciones iniciales:

$$s(0) = \dot{s}(0) = 0 \quad (4.129)$$

y las restricciones terminales:

$$s(T) = 1 \quad (4.130)$$

$$\dot{s}(T) = 0 \quad (4.131)$$

Derivando la ecuación (4.128):

$$\dot{s}(t) = a_1 + 2a_2 t + 3a_3 t^2 \quad (4.132)$$

Evaluando la ecuación (4.132) en $t = 0$, $t = T$ y resolviendo las cuatro restricciones representadas por las ecuaciones (4.129),(4.130) y (4.131) :

$$a_0 = 0 ; a_1 = 0 ; a_2 = \frac{3}{T^2} ; a_3 = -\frac{2}{T^3} \quad (4.133)$$

Reemplazando los coeficientes a_i en la ecuación (4.128):

$$s(t) = a_2 t^2 + a_3 t^3 = \left(\frac{3}{T^2}\right) t^2 + \left(-\frac{2}{T^3}\right) t^3 \quad (4.134)$$

Sustituyendo la ecuación (4.134) y sus derivadas en las ecuaciones (4.122),(4.126) y (4.127) :

$$\theta(t) = \theta_{inicio} + \left(\frac{3t^2}{T^2} - \frac{2t^3}{T^3}\right) (\theta_{fin} - \theta_{inicio}) \quad (4.135)$$

$$\dot{\theta}(t) = \left(\frac{6t}{T^2} - \frac{6t^2}{T^3}\right) (\theta_{fin} - \theta_{inicio}) \quad (4.136)$$

$$\ddot{\theta}(t) = \left(\frac{6}{T^2} - \frac{12t}{T^3}\right) (\theta_{fin} - \theta_{inicio}) \quad (4.137)$$

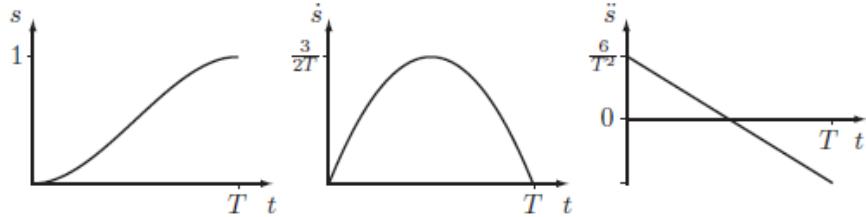


Figura 4.37: Gráficas de $s(t)$, $\dot{s}(t)$ y $\ddot{s}(t)$ para una escala de tiempo polinomial de tercer orden [65].

Las velocidades máximas de la articulación se alcanzan en el punto medio del movimiento, $t = T/2$:

$$\dot{\theta}_{max} = \frac{3}{2T} (\theta_{fin} - \theta_{inicio}) \quad (4.138)$$

Las aceleraciones y desaceleraciones máximas de la articulación se logran en $t = 0$ y $t = T$:

$$\ddot{\theta}_{max} = \left| \frac{6}{T^2} (\theta_{fin} - \theta_{inicio}) \right| \quad (4.139)$$

$$\ddot{\theta}_{min} = - \left| \frac{6}{T^2} (\theta_{fin} - \theta_{inicio}) \right| \quad (4.140)$$

Si existen límites conocidos en las velocidades máximas de la articulación $|\dot{\theta}| \leq \dot{\theta}_{limite}$ y las aceleraciones máximas de la articulación $|\ddot{\theta}| \leq \ddot{\theta}_{limite}$, estos límites se pueden verificar para ver si el tiempo de movimiento solicitado T es factible. Alternativamente, se podría resolver T para encontrar el tiempo de movimiento mínimo posible que satisfaga la restricción de velocidad o aceleración más restrictiva.

4.6.5.2.2. Perfiles de movimiento trapezoidal Las escalas de tiempo trapezoidales son bastante comunes en el control motor, particularmente para el movimiento de una sola articulación. El movimiento punto a punto consta de una fase de aceleración constante $\ddot{s} = a$ de tiempo t_a , seguida de una fase de velocidad constante $\dot{s} = v$ de tiempo $t = T - 2t_a$, seguida de una fase de desaceleración constante $\ddot{s} = -a$ de tiempo t_a . El resultado del perfil \dot{s} es un trapezoide y el perfil s es la concatenación de una parábola, un segmento lineal y una parábola en función del tiempo (figura (4.38)).

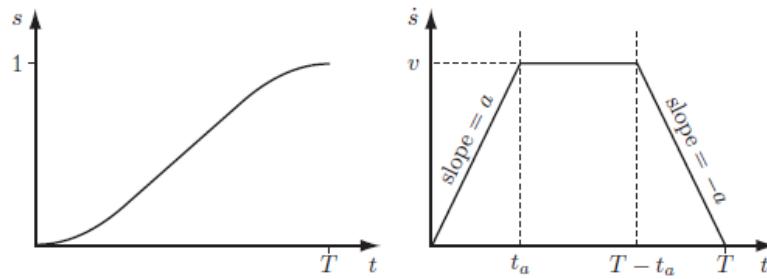


Figura 4.38: Gráficas de $s(t)$ y $\dot{s}(t)$ para un perfil de movimiento trapezoidal [65]

La escala de tiempo trapezoidal no es tan suave como la escala de tiempo cúbico, pero tiene la ventaja, si se saben los límites constantes de las velocidades en las articulaciones $\dot{\theta}_{limite} \in \mathbb{R}^n$ y los límites de aceleraciones de las articulaciones $\ddot{\theta}_{limite} \in \mathbb{R}^n$, entonces el movimiento trapezoidal usado por la mayor v y a debe satisfacer:

$$|(\theta_{final} - \theta_{inicio})v| \leq \dot{\theta}_{limite} \quad (4.141)$$

$$|(\theta_{final} - \theta_{inicio})a| \leq \ddot{\theta}_{limite} \quad (4.142)$$

Si $v^2/a > 1$, el robot nunca alcanza la velocidad v durante el movimiento. El movimiento trifásico de aceleración-velocidad constante-desaceleración se convierte en un movimiento bifásico de aceleración-desaceleración llamado “bang-bang” y el perfil trapezoidal $\dot{s}(t)$ de la figura (4.38) se convierte en un triángulo.

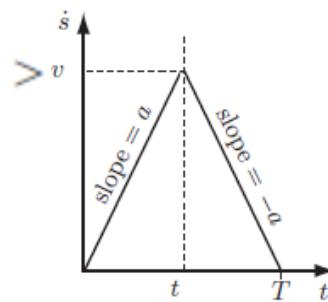


Figura 4.39: Bang - bang [65]

Suponiendo que $v^2/a \leq 1$, el movimiento trapezoidal está completamente especificado por v, a, t_a y T , pero solo dos de estos pueden especificarse independientemente ya que deben satisfacer $s(T) = 1$ y $v = at_a$. Es poco probable que se especifique t_a de forma independiente, por lo que se elimina de las ecuaciones de movimiento mediante la sustitución $t_a = v/a$. El perfil de movimiento durante las tres etapas (aceleración, velocidad constante, desaceleración) se puede escribir en términos de v, a y T de la siguiente manera:

Para $0 \leq t \leq \frac{v}{a}$:

$$\ddot{s}(t) = a \quad (4.143)$$

$$\dot{s}(t) = at \quad (4.144)$$

$$s(t) = \frac{1}{2}at^2 \quad (4.145)$$

Para $\frac{v}{a} \leq t \leq (T - \frac{v}{a})$:

$$\ddot{s}(t) = 0 \quad (4.146)$$

$$\dot{s}(t) = v \quad (4.147)$$

$$s(t) = vt - \frac{v^2}{2a} \quad (4.148)$$

Para $(T - \frac{v}{a}) \leq t \leq T$:

$$\ddot{s}(t) = -a \quad (4.149)$$

$$\dot{s}(t) = a(T - t) \quad (4.150)$$

$$s(t) = \frac{2avT - 2v^2 - a^2(t - T)^2}{2a} \quad (4.151)$$

Dado que solo dos de v, a y T pueden elegirse independientemente, se tiene 3 opciones:

- Se elige v y a tal que $v^2/a \leq 1$, asegurando un perfil trapezoidal de tres etapas y se resuelve $s(T) = 1$ para T :

$$T = \frac{a + v^2}{va} \quad (4.152)$$

Si v y a corresponden a las mayores velocidades y aceleraciones articulares posibles, este es el tiempo mínimo posible para el movimiento.

- Se elige v y T de modo que $2 \geq vT \geq 1$, garantizando un perfil trapezoidal de tres etapas y que la velocidad máxima v sea suficiente para alcanzar $s = 1$ en el tiempo T . Resolviendo $s(T) = 1$ para a :

$$a = \frac{v^2}{vT - 1} \quad (4.153)$$

- Eligiendo a y T tal que $aT^2 \geq 4$, asegurando que el movimiento sea completo en el tiempo T , resolviendo $s(T) = 1$ para v :

$$v = \frac{1}{2}(aT - \sqrt{a}\sqrt{aT^2 - 4}) \quad (4.154)$$

Esta misma representación del perfil trapezoidal en esta sección tambien se puede encontrar en el libro [66].

Capítulo 5

Especificaciones del robot delta seleccionado

En este capítulo se presenta los valores de las dimensiones, masas e inercias que caracterizan el robot delta a simular. Se extraer los valores del prototipo del robot delta en tesis [67], ya que en este documento no abarca la optimización dimensional de las piezas por energía y tampoco se tiene los límites de un espacio de trabajo determinado para una aplicación específica. Los datos se ingresan al archivo *pd_tm1_adams.py*.

Parametro	Descripción	Valor
L_1	Longitud de un brazo	620 [mm]
L_2	Longitud de un antebrazo	880 [mm]
R_A	Radio de la base fija	210 [mm]
R_B	Radio de la plataforma móvil	50 [mm]
m_1	Masa de un brazo	2.213 [kg]
m_2	Masa de una varilla del antebrazo	657.5 [kg]
m_p	Masa de la plataforma móvil	0.510 [kg]
$m_{payload}$	Masa de carga que es trasladada por el efecto	0 [kg]
m_{elbow}	Masa de juntas que conectas los brazos con los antebrazos	0 [kg]
I_m	Inercia de los actuadores	0 [kg^*m^2]
r_{mass}	Relación masas antebrazo	0.5
g	Gravedad	9.81 [m/s^2]

Tabla 5.1: Parámetros necesarios para la simulación del robot delta en esta tesis

La simplificación del robot delta se presenta en la figura (5.1).

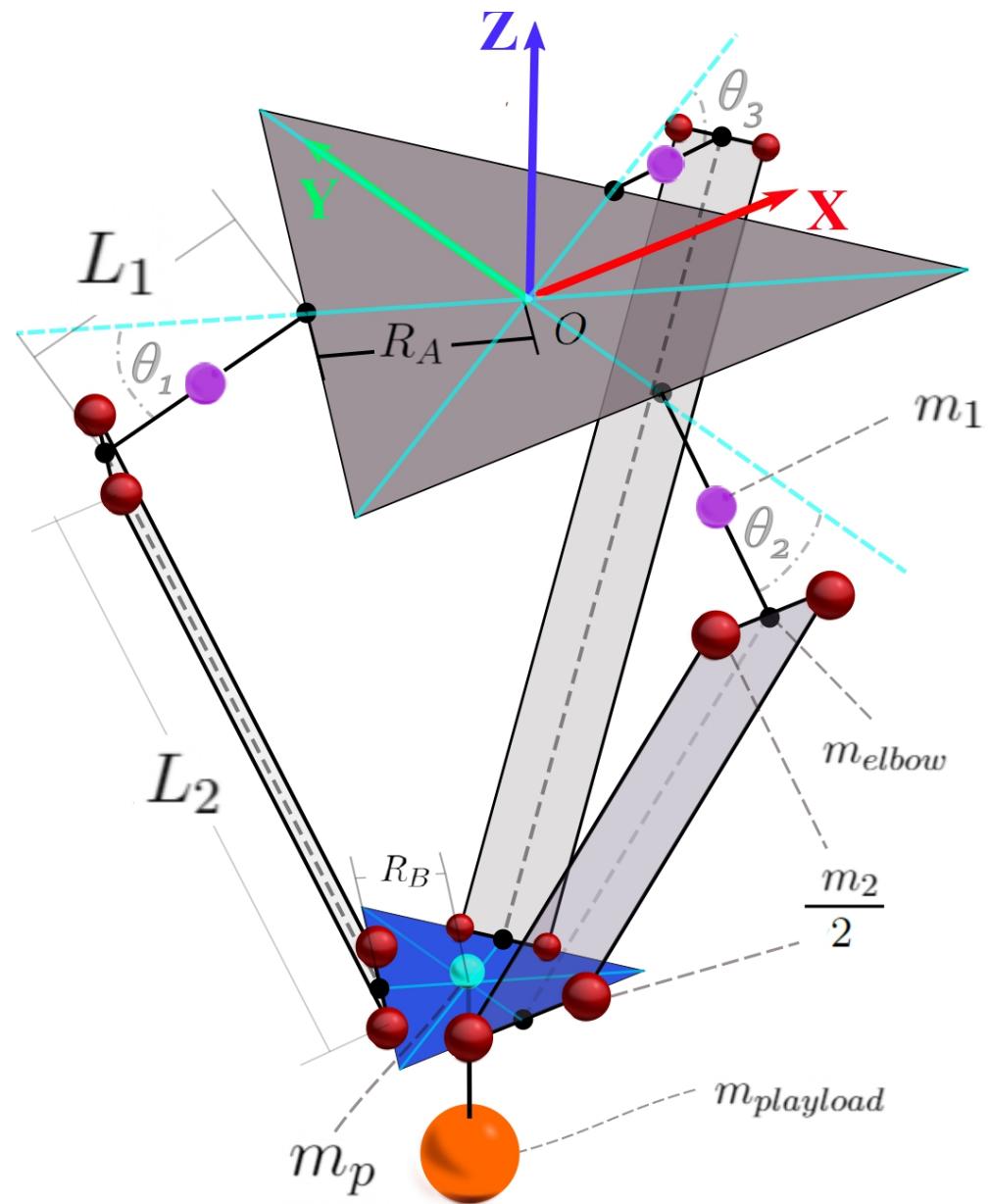


Figura 5.1: Representación gráfica, marco de referencia, dimensiones, masas y orden de ángulos del robot delta a simular.

Capítulo 6

Desarrollo de la solución

En este capítulo se explica la solución adoptada para validar la cinemática y dinámica de un robot delta. Para ello se implementan 3 herramientas a este trabajo: ROS, RViz y ADAMS. Se utiliza la versión de ROS es Melodic y Python 2.7.

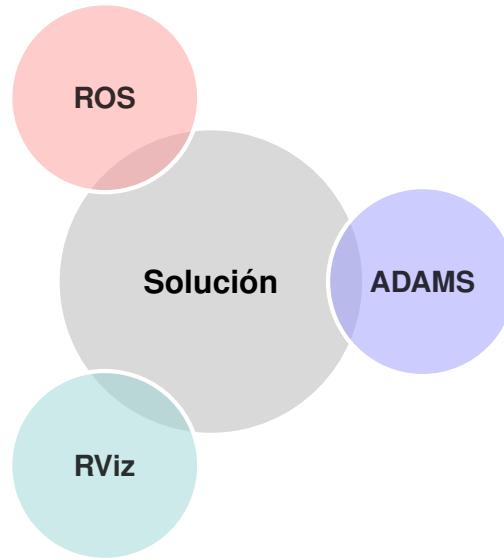


Figura 6.1: Herramientas principales para la solución de la dinámica y cinemática del robot delta.

Se siguen una serie de pasos para validar la cinemática y dinámica del robot delta:

1. **ROS y RViz**

- **Trayectoria:** Se crear una trayectoria lineal (perfil de velocidad trapezoidal trapezoidal) en el espacio cartesiano del centroide del efecto a partir de un punto inicial $P_i(x, y, z)$, un punto final $P_f(x, y, z)$, velocidad maxima v_{max} y aceleración maxima a_{max} deseada. Se obtienen los datos de la trayectoria XYZ , escala de tiempo $s(t)$, velocidad $\dot{X}\dot{Y}\dot{Z}$ y aceleración $\ddot{X}\ddot{Y}\ddot{Z}$.

- **Cinemática inversa:** Se calcular la trayectoria en el espacio articular $\theta_{i \in \{1,2,3\}}$ de los motores del robot delta.
- **Visualización:** En RViz se inicia la simulación de la trayectoria creada en los puntos anteriores. Esta visualización es una primera aproximación (a simple vista) de la validación cinemática del robot delta. Específicamente se valida la cinemática inversa.
- **Guardar trayectoria:** Se guarda en un archivo .txt los caminos geométricos XYZ , $\theta_{i \in \{1,2,3\}}$ y la escala de tiempo $s(t)$.
- **Velocidad y aceleración de los actuadores:** Por medio de la cinemática de velocidad (Jacobiano) y la cinemática de aceleración se calculan las velocidades $\dot{\theta}_{i \in \{1,2,3\}}$ y las aceleraciones. $\ddot{\theta}_{i \in \{1,2,3\}}$ de la trayectoria en el espacio articular.
- **Torque de motores:** A partir de los datos de posición, velocidad y aceleración de los motores se calculan los torques de ellos $\tau_{i \in \{1,2,3\}}$.
- **Guardar torques:** Se guarda en un archivo .txt los torques calculados de los motores (actuadores) con los algoritmos compilados en ROS.

2. ADAMS

- **Importación de la trayectoria:** Se importan los datos de la trayectoria en el espacio articular $\theta_{i \in \{1,2,3\}}$, espacio cartesiano XYZ y la escala de tiempo $s(t)$ desde ROS.
- **Dibujar trayectoria:** Se dibuja la trayectoria cartesiana XYZ importada desde ROS en el espacio de simulación de ADAMS para comprobar a simple vista que la trayectoria articular (impuesta a los motores de ADAMS) sea la correcta al momento de la simulación. El centro del efecto final debe seguir dicha trayectoria cartesiana lineal si es que el robot delta está bien configurado en ADAMS.
- **Trayectoria articular y escala de tiempo:** Se simula la trayectoria creada en ROS en el robot delta configurado en ADAMS, imponiendo a los motores (o sensores tipo junta) la trayectoria articular $\theta_{i \in \{1,2,3\}}$ y su respectiva escala de tiempo $s(t)$ exportada desde ROS.
- **Guardar torque:** Se guarda en un archivo .txt el torque producido por los motores en simulación de ADAMS.

3. Validación cinemática y dinámica

- Se comparan los resultados de los torques por ROS y ADAMS. Si estos tienen errores relativamente bajos se asume que los algoritmos de cinemática y dinámica compilados en ROS y la configuración del robot delta en ADAMS son correctas.

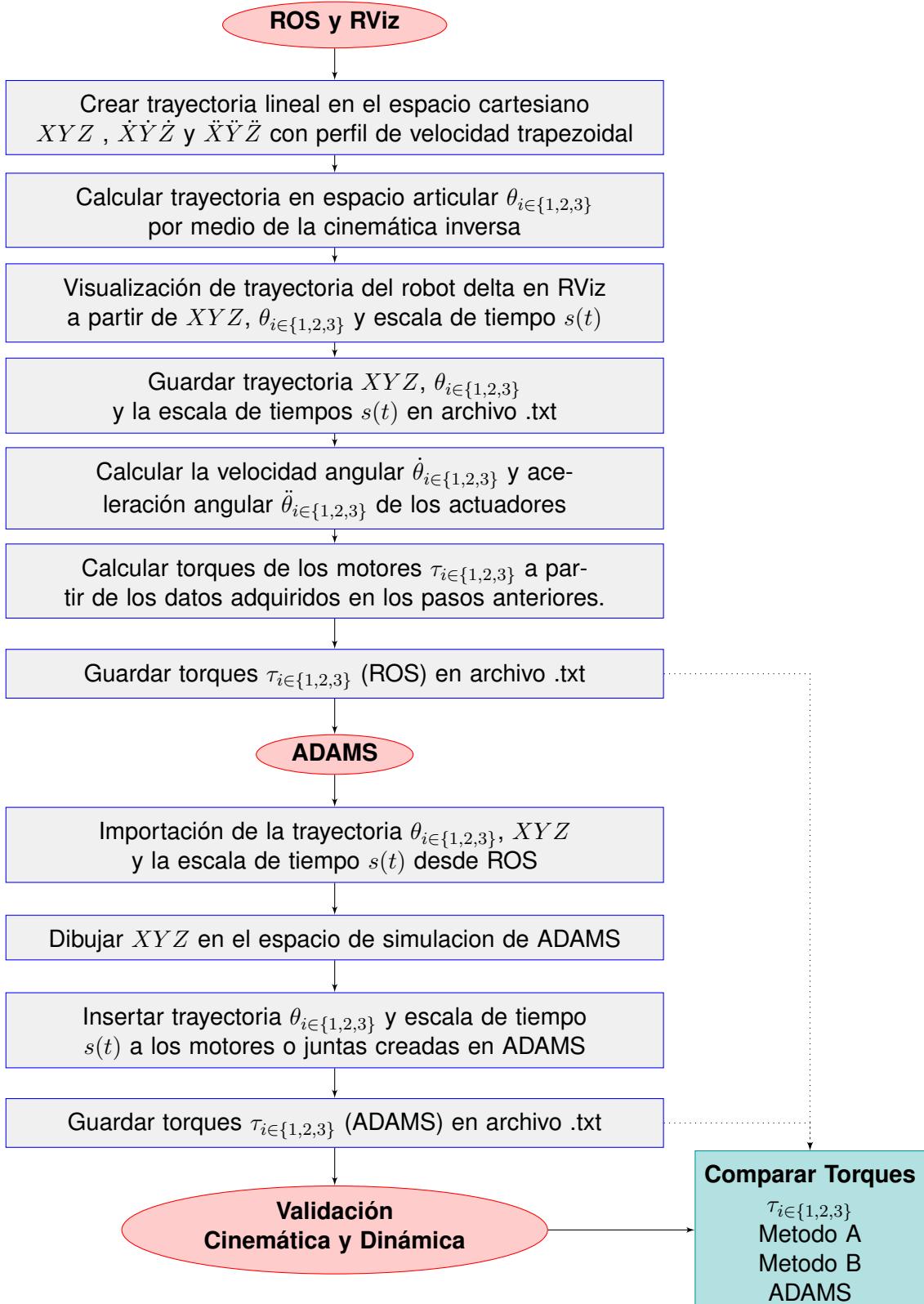


Figura 6.2: Flujo de trabajo para la validación dinámica y cinemática del robot delta.

6.1. Software ROS

Robot Operating Processing es una herramienta que sirve para ordenar y dividir en tareas específicas un robot complejo simplificando a los ingenieros la creación de estos. Estas tareas se representan en nodos, funciones y subfunciones que son escritos en lenguaje de programación python. En esta sección, estos algoritmos son explicados brevemente como una función, con entradas y salidas definidas como se muestra en la figura (6.3) y explicados detalladamente en pseudocódigos como se puede visualizar en el algoritmo (1). Los algoritmos se dividen en una función principal y en subfunciones.



Figura 6.3: Ejemplo representación gráfica de funciones o nodos.

Algorithm 1: Nombre del Algoritmo

Objetivo: Explicación breve del objetivo principal del algoritmo

Nombre archivo: Nombre del archivo donde se encuentra el código del algoritmo

Input: *Entradas*

Output: *Salidas*

```
/* FUNCION PRINCIPAL */  
1 nombre_funcion_principal(Input):  
    /* Algoritmo funcion principal */  
2     return Output  
/* SUBFUNCIONES */  
3 nombre_subfunciones(Input):  
    /* Algoritmo de subfunciones */  
4     return Output
```

Las tareas realizadas en ROS son principalmente los conceptos teóricos del capítulo (4):

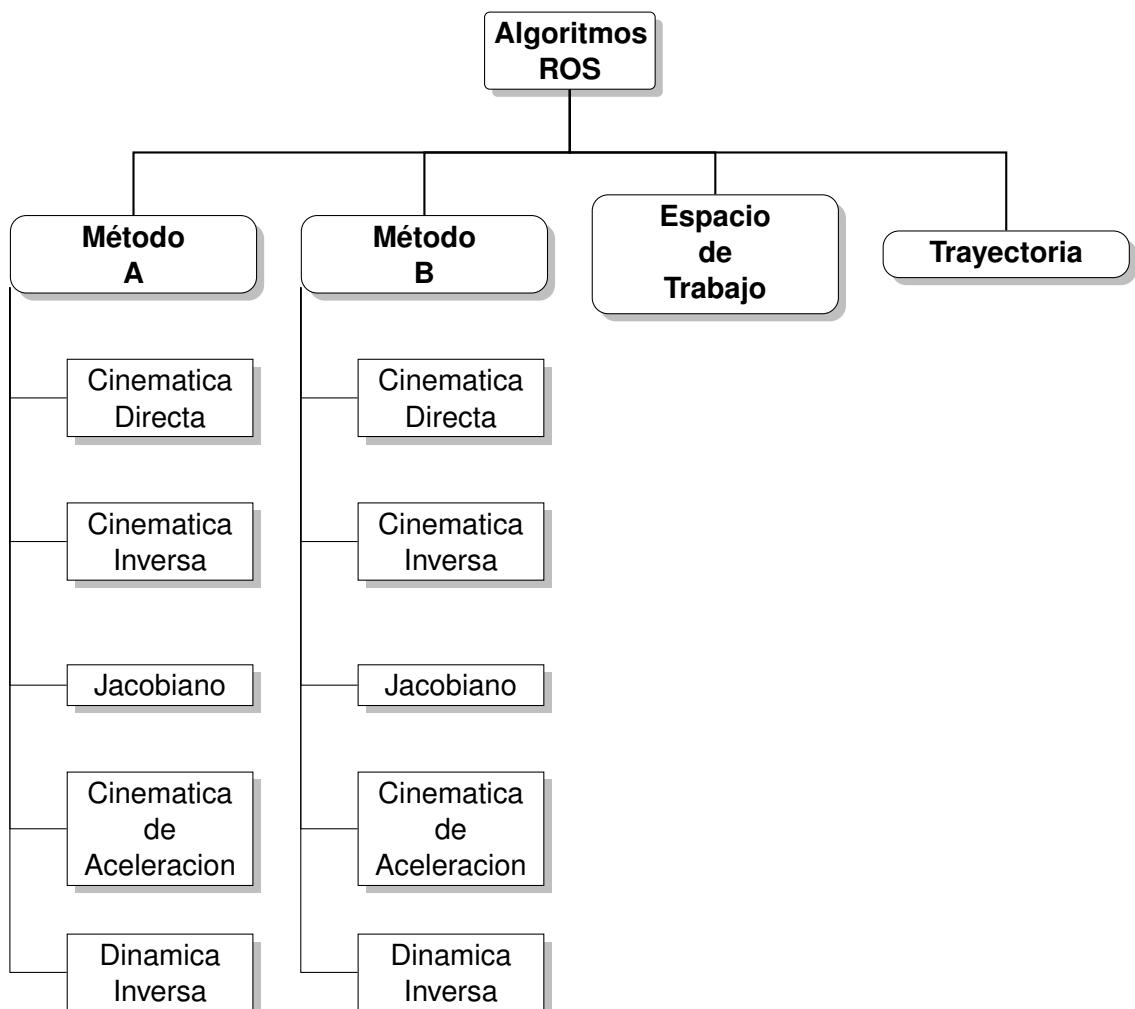


Figura 6.4: Conceptos programados en ROS.

6.1.1. Nodo principal

Los algoritmos de los métodos A y B son utilizados finalmente para obtener el torque de los motores de un robot delta a partir de una trayectoria lineal específica del efecto final. Estos algoritmos son compilados en un nodo llamado nodo principal.

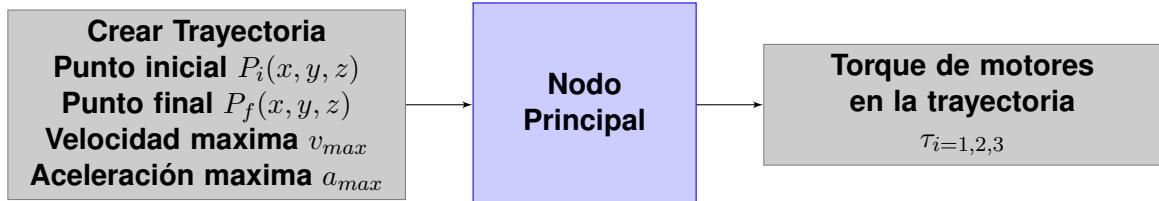


Figura 6.5: Entradas y salidas del nodo principal

El pseudocódigo de el nodo es el siguiente:

Algorithm 2: Nodo Principal

Objetivo: Compilar los algoritmos de los Metodos A y B para obtener el torque de los motores del robot delta

Nombre archivo: *path_tm1_v2_adams.py*

Input: Punto inicial $P_i(x, y, z)$, punto final $P_f(x, y, z)$, velocidad maxima v_{max} y aceleración maxima a_{max} (datos para crear trayectoria)

Output: Torque $\tau_{i=1,2,3}$ de los métodos A y B

```

1 nodo( $P_i(x, y, z)$ ,  $P_f(x, y, z)$ ,  $v_{max}$ ,  $a_{max}$ ):
    /* Nota: El nombre de las funciones en este pseudocódigo no son
       los nombres de las funciones en el archivo .py , sino son
       nombres de referencia en base al objetivo de cada función para
       explicar el flujo de trabajo que tiene como finalidad obtener
       el torque de los motores. */
    /* Crear Trayectoria */
2   [ $XYZ$ ;  $\dot{X}\dot{Y}\dot{Z}$ ;  $\ddot{X}\ddot{Y}\ddot{Z}$ ,  $s(t)$ ] = trayectoria( $P_i(x, y, z)$ ,  $P_f(x, y, z)$ ,  $v_{max}$ ,  $a_{max}$ )
    /* Metodo A */
3   [ $\theta_1, \theta_2, \theta_3$ ] = cinematica.inversa( $XYZ$ )
4   visualizacion.trayectoria.( $XYZ, \theta_1\theta_2\theta_3, s(t)$ ) // RViz
5   guardar.trayectoria( $XYZ, \theta_1\theta_2\theta_3, s(t)$ )
6   ( $\dot{\theta}_{i=1,2,3}, \ddot{\theta}_{i=1,2,3}$ ) = velocidad.y.aceleracion.angular( $XYZ$ ;  $\dot{X}\dot{Y}\dot{Z}$ ;  $\ddot{X}\ddot{Y}\ddot{Z}$ )
7    $\tau_{A,i=1,2,3} = torque.A(XYZ; \dot{X}\dot{Y}\dot{Z}; \ddot{X}\ddot{Y}\ddot{Z}, \theta_{i=1,2,3}, \dot{\theta}_{i=1,2,3}, \ddot{\theta}_{i=1,2,3})$ 
    /* Metodo B */
8   El metodo B tiene un flujo casi identico al del metodo A
9    $\tau_{B,i=1,2,3} = torque.B(XYZ; \dot{X}\dot{Y}\dot{Z}; \ddot{X}\ddot{Y}\ddot{Z}, \theta_{i=1,2,3}, \dot{\theta}_{i=1,2,3}, \ddot{\theta}_{i=1,2,3})$ 
10  return  $\tau_{A,i=1,2,3}$  y  $\tau_{B,i=1,2,3}$ 
  
```

Las trayectorias que se crean para comprobar los algoritmos de cinemática y dinámica son:

Trayectoria	$P_i(x, y, z)[mm]$	$P_f(x, y, z)[mm]$	$v_{max}[\frac{mm}{s}]$	$a_{max}[\frac{mm}{s^2}]$
1	(-300,0,-450)	(300,150,-750)	2000	40000
2	(-300,-150,-450)	(300,150,-750)	2000	40000
3	(300,-150,-450)	(-300,150,-750)	2000	40000
4	(400,150,-450)	(-400,150,-450)	2000	40000
5	(-300,0,-450)	(300,150,-750)	200	10000
6	(-300,-150,-450)	(300,150,-750)	200	10000
7	(300,-150,-450)	(-300,150,-750)	200	10000
8	(400,150,-450)	(-400,150,-450)	200	10000

Tabla 6.1: Trayectorias simuladas

Los puntos iniciales y finales son en base al sistema de referencia global presentado en la figura (4.2). Los valores de las velocidades y aceleraciones máximas son en dirección tangencial a las trayectorias lineales creadas.

6.1.2. Método A

Las figuras de la (6.6) a la (6.10) representan de forma visual los algoritmos, tareas o funciones planteadas en esta sección con respecto al método A del capítulo (4), de igual forma que en la figura (6.3). Las entradas y las salidas de cada función son escritas con las misma nomenclatura de cada sección con que fue inspirado cada algoritmo respectivamente. Es notorio recalcar que cada algoritmo es obtenido de diferentes referencias bibliográficas, por lo que los sistemas de referencia y el orden numérico de los ángulos de los actuadores son diferentes. Por ende, las entradas y las salidas en cada algoritmo son transformadas del sistema global al sistema de referencia local y viceversa.



Figura 6.6: Función cinemática directa del método A

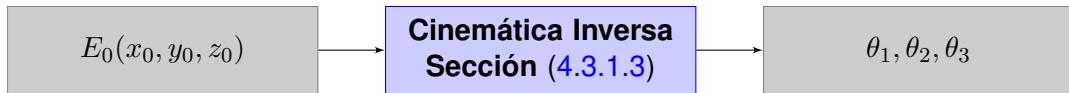


Figura 6.7: Función cinemática inversa del método A

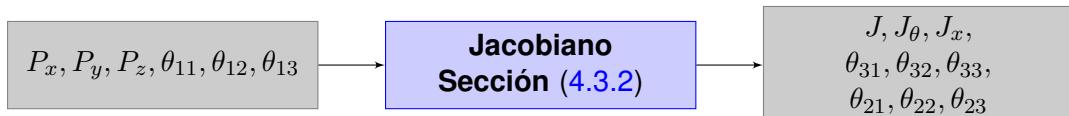


Figura 6.8: Función jacobiano del método A

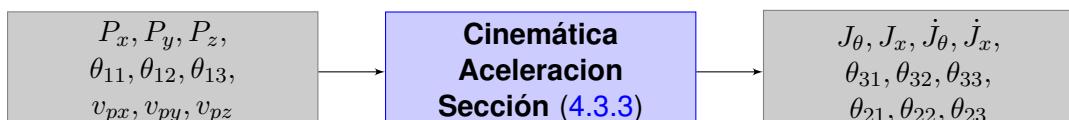


Figura 6.9: Función cinemática de aceleración del método A

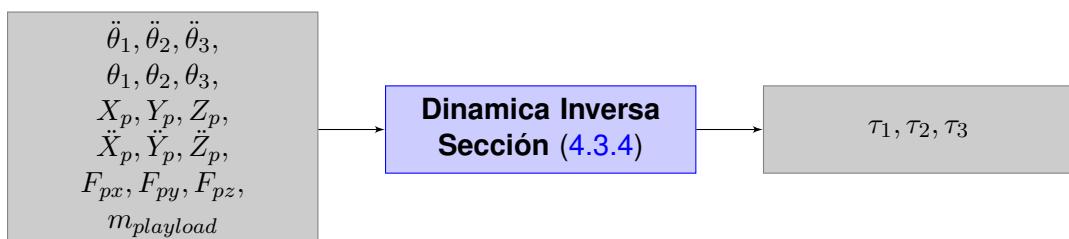


Figura 6.10: Función dinámica inversa del método A

6.1.2.1. Cinemática directa

Algorithm 3: Cinemática Directa método A

Objetivo: Hallar la posición del efecto final del robot delta dada una configuración articular

Nombre archivo: *delta_kinematics_tlm_adams.py*

Input: $\theta_1, \theta_2, \theta_3$

Output: $E_0(x_0, y_0, z_0)$

/* FUNCION PRINCIPAL */

```

1 forward( $\theta_1, \theta_2, \theta_3$ ):
    /* Cambiar angulos al orden del sistema de referencia local      */
    /* Calcular Centros de esferas (Ecuaciones tabla (4.4))          */
2   ( $x_1, y_1, z_1$ )= $J'_1\left(0, \left[-\frac{f-e}{2\sqrt{3}} - r_f \cos(\theta_1)\right], -r_f \sin(\theta_1)\right)$ 
3   ( $x_2, y_2, z_2$ )= $J'_2\left(\left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_2)\right] \cos(30^\circ), \left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_2)\right] \sin(30^\circ), -r_f \sin(\theta_2)\right)$ 

4   ( $x_3, y_3, z_3$ )= $J'_3\left(-\left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_3)\right] \cos(30^\circ), \left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_3)\right] \sin(30^\circ), -r_f \sin(\theta_3)\right)$ 

    /* Calcular intersección de las 3 esferas (Ecuaciones de la (4.3) a
       la (4.13))                                                 */
5    $d = (y_2 - y_1)x_3 - (y_3 - y_1)x_2 - (y_2 - y_3)x_1$ 
6    $w_1 = x_1^2 + y_1^2 + z_1^2$ 
7    $w_2 = x_2^2 + y_2^2 + z_2^2$ 
8    $w_3 = x_3^2 + y_3^2 + z_3^2$ 
9    $a_1 = \frac{(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)}{d}$ 
10   $b_1 = \left(\frac{1}{2*(-d)}\right) * ((w_2 - w_1)(y_3 - y_1) - (w_3 - w_1)(y_2 - y_1))$ 
11   $a_2 = \frac{-1}{d} * [(z_2 - z_1)x_3 - (z_3 - z_1)x_2 + (z_3 - z_2)x_1]$ 
12   $b_2 = \frac{1}{2d} * [(w_2 - w_1)x_3 - (w_3 - w_1)x_2 + (w_3 - w_2)x_1]$ 
13   $A = a_1^2 + a_2^2 + 1$ 
14   $B = 2(a_1(b_1 - x_1) + a_2(b_2 - y_1) - z_1)$ 
15   $C = ((b_1 - x_1)^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2)$ 
16   $E_0(x_0, y_0, z_0) = \left(a_1 z + b_1, a_2 z + b_2, \frac{-B - \sqrt{(B^2) - (4AC)}}{2A}\right)$ 
17  return ( $x_0, y_0, z_0$ )

```

6.1.2.2. Cinemática inversa

Algorithm 4: Cinemática Inversa método A

Objetivo: Hallar la posición de los actuadores en el espacio articular dada la posición del centroide del efecto final

Nombre archivo: *delta_kinematics_tlm_adams.py*

Input: $E_0(x_0, y_0, z_0)$

Output: $\theta_1, \theta_2, \theta_3$

```
/* FUNCION PRINCIPAL */  
1 inverse(x0, y0, z0):  
    /* Rotación del sistema de referencia Local en 0° */  
2     x0° = x0  
3     y0° = y0  
4     z0° = z0  
    /* Rotación del sistema de referencia Local en 120° */  
5     x120° = x0 * cos120 + y0 * sin120  
6     y120° = y0 * cos120 - x0 * sin120  
7     z120° = z0  
    /* Rotación del sistema de referencia Local en 240° */  
8     x240° = x0 * cos120 - y0 * sin120  
9     y240° = y0 * cos120 + x0 * sin120  
10    z240° = z0  
    /* Calcular angulos de los actuadores */  
    /* Cambiar orden de ángulos al sistema Referencia Global */  
11    θ2 = angle_yz(x0°, y0°, z0°, 0°)  
12    θ3 = angle_yz(x120°, y120°, z120°, 120°)  
13    θ1 = angle_yz(x240°, y240°, z240°, 240°)  
14    return [θ1, θ2, θ3]
```

Algorithm 4: Cinemática Inversa método A (Continuacion...)

```
/* SUBFUNCIONES */  
1 angle_yz( $x_{\phi_i}, y_{\phi_i}, z_{\phi_i}, \phi_i$ ):  
    /* Actuador  $F_i$  en el plano  $Y'Z'$  (sistema de referencia Local  
    rotado en  $\phi_i$ ) */  
    2  $y_1 = y_{F_i} = -\frac{f}{2\sqrt{3}};$   
    3  $z_1 = z_{F_i} = 0;$   
    /* Proyección de la Junta Esférica  $E_i$  en el plano  $Y'Z'$  (sistema de  
    referencia Local rotado en  $\phi_i$ ) */  
    4  $y_2 = y_{E'_i} = y_{\phi_i} - \frac{e}{2\sqrt{3}};$   
    5  $z_2 = z_{E'_i} = z_{\phi_i};$   
    /* Junta Esférica  $J_i$  de la cadena cinemática i en posición  $\phi_i$  */  
    6  $a = \frac{x_{\phi_i}^2 + y_2^2 + z_{\phi_i}^2 + r_f^2 - r_e^2 - y_1^2}{2z_{\phi_i}};$   
    7  $b = \frac{(y_1 - y_2)}{z_2};$   
    8  $x_{J_i} = 0;$   
    9  $y_{J_i} = \frac{(y_1 - ab) - \sqrt{-(a+by_1)^2 + (b^2+1)r_f^2}}{b^2+1};$   
10  $z_{J_i} = a + b * y_{J_i};$   
    /* Calculo del Angulo  $\theta_i$  de la cadena cinemática i en posición  $\phi_i$   
    ecuacion (4.20) */  
11  $\theta_i = \arctan\left(\frac{z_{J_i}}{y_{F_i} - y_{J_i}}\right);$   
12 return  $\theta_i;$   
13
```

6.1.2.3. Jacobiano

Algorithm 5: Jacobiano método A

Objetivo: Hallar la matriz jacobiana

Nombre archivo: *jacobian_tm1_adams.py*

Input: $P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13}$

Output: $J_\theta, J_x, \theta_{31}, \theta_{32}, \theta_{33}, \theta_{21}, \theta_{22}, \theta_{23}, J$

```

/* FUNCION PRINCIPAL
1 jacobian_total( $P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13}$ ):
   /* Cambiar ángulos y punto P al sistema Referencia Local */ 
   2 ( $\phi_0^\circ, \phi_{120^\circ}, \phi_{240^\circ}$ ) = (0, 120, 240)
   /* Calcular angulo  $\theta_{3i}$  y  $\theta_{2i}$  */ 
   3  $\theta_{31} = calculo\_theta3i(P_x, P_y, \phi_0^\circ)$ 
   4  $\theta_{32} = calculo\_theta3i(P_x, P_y, \phi_{120^\circ})$ 
   5  $\theta_{33} = calculo\_theta3i(P_x, P_y, \phi_{240^\circ})$ 
   6  $\theta_{21} = calculo\_theta2i(P_x, P_y, P_z, \theta_{31}, \phi_0^\circ)$ 
   7  $\theta_{22} = calculo\_theta2i(P_x, P_y, P_z, \theta_{32}, \phi_{120^\circ})$ 
   8  $\theta_{23} = calculo\_theta2i(P_x, P_y, P_z, \theta_{33}, \phi_{240^\circ})$ 
   /* Calcular Matriz  $J_x$  */ 
   9  $J_{1x} = calculo\_jix(\theta_{11}, \theta_{21}, \theta_{31}, \phi_0^\circ)$ 
  10  $J_{2x} = calculo\_jix(\theta_{12}, \theta_{22}, \theta_{32}, \phi_{120^\circ})$ 
  11  $J_{3x} = calculo\_jix(\theta_{13}, \theta_{23}, \theta_{33}, \phi_{240^\circ})$ 
  12  $J_{1y} = calculo\_jiy(\theta_{11}, \theta_{21}, \theta_{31}, \phi_0^\circ)$ 
  13  $J_{2y} = calculo\_jiy(\theta_{12}, \theta_{22}, \theta_{32}, \phi_{120^\circ})$ 
  14  $J_{3y} = calculo\_jiy(\theta_{13}, \theta_{23}, \theta_{33}, \phi_{240^\circ})$ 
  15  $J_{1z} = calculo\_jiz(\theta_{11}, \theta_{21}, \theta_{31}, \phi_0^\circ)$ 
  16  $J_{2z} = calculo\_jiz(\theta_{12}, \theta_{22}, \theta_{32}, \phi_{120^\circ})$ 
  17  $J_{3z} = calculo\_jiz(\theta_{13}, \theta_{23}, \theta_{33}, \phi_{240^\circ})$ 
  18  $J_x = [J_{1x}, J_{1y}, J_{1z}; J_{2x}, J_{2y}, J_{2z}; J_{3x}, J_{3y}, J_{3z}]$ 
   /* Calcular Matriz  $J_\theta$  */ 
  19  $J_{1\theta} = calculo\_jthetai(\theta_{21}, \theta_{31}, \phi_0^\circ)$ 
  20  $J_{2\theta} = calculo\_jthetai(\theta_{22}, \theta_{32}, \phi_{120^\circ})$ 
  21  $J_{3\theta} = calculo\_jthetai(\theta_{23}, \theta_{33}, \phi_{240^\circ})$ 
  22  $J_\theta = [J_{1\theta}, 0, 0; 0, J_{2\theta}, 0; 0, 0, J_{3\theta}]$ 
   /* Calcular Matriz  $J$  ecuación (??) */ 
  23  $J = J_x^{-1} J_\theta$ 
   /* Cambiar al sistema Referencia Global al operar */ 
  24 return [ $J_\theta, J_x, \theta_{31}, \theta_{32}, \theta_{33}, \theta_{21}, \theta_{22}, \theta_{23}, J$ ]
  25

```

Algorithm 5: Jacobiano método A (Continuacion...)

```
/* SUBFUNCIONES */  
/* Calcular angulo  $\theta_{3i}$  ecuación (4.23) */  
1 calculo_theta3i( $P_x, P_y, \phi_i$ ):  
2    $c_{yi} = (-P_x) * (\sin(\phi_i)) + (P_y) * (\cos(\phi_i));$   
3    $\theta_{3i} = \cos^{-1} \frac{c_{yi}}{b};$   
4   return  $\theta_{3i};$   
/* Calcular angulo  $\theta_{2i}$  ecuación (4.24) */  
5 calculo_theta2i( $P_x, P_y, P_z, \theta_{3i}, \phi_i$ ):  
6    $c_{xi} = (P_x) * (\cos(\phi_i)) + (P_y) * (\sin(\phi_i)) + h - r;$   
7    $c_{yi} = (-P_x) * (\sin(\phi_i)) + (P_y) * (\cos(\phi_i));$   
8    $c_{zi} = P_z;$   
9    $\theta_{2i} = \cos^{-1} \left( \frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2ab \sin \theta_{3i}} \right);$   
10  return  $\theta_{2i};$   
/* Calcular componentes Matriz  $J_x$  */  
/* Calcular componentes Matriz  $J_{ix}$  ecuación (4.28) */  
11 calculo_jix( $\theta_{1i}, \theta_{2i}, \theta_{3i}, \phi_i$ ):  
12    $J_{ix} = \cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \cos \phi_i - \cos \theta_{3i} \sin \phi_i;$   
13   return  $J_{ix};$   
/* Calcular componentes  $J_{iy}$  ecuación (4.29) */  
14 calculo_jiy( $\theta_{1i}, \theta_{2i}, \theta_{3i}, \phi_i$ ):  
15    $J_{iy} = \cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \sin \phi_i + \cos \theta_{3i} \cos \phi_i;$   
16   return  $J_{iy};$   
/* Calcular componentes  $J_{iz}$  ecuación (4.30) */  
17 calculo_jiz( $\theta_{1i}, \theta_{2i}, \theta_{3i}, \phi_i$ ):  
18    $J_{iz} = \sin(\theta_{1i} + \theta_{2i}) \sin \theta_{3i};$   
19   return  $J_{iz};$   
/* Calcular componentes  $J_\theta$  ecuación (4.31) */  
20 calculo_jthetai( $\theta_{2i}, \theta_{3i}, \phi_i$ ):  
21    $J_{i\theta} = a \sin \theta_{2i} \sin \theta_{3i};$   
22   return  $J_{i\theta};$ 
```

6.1.2.4. Modelacion cinematica de Aceleración

Algorithm 6: Aceleración método A

Objetivo: Hallar las matrices para calcular la aceleración angular de los actuadores

Nombre archivo: jacobian_tm1_adams.py

Input: $P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13}, v_{px}, v_{py}, v_{pz}$

Output: $J_\theta, J_x, \dot{J}_\theta, \dot{J}_x, \theta_{31}, \theta_{32}, \theta_{33}, \theta_{21}, \theta_{22}, \theta_{23}$

```

/* FUNCION PRINCIPAL
1 jacobian_total_der( $P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13}, v_{px}, v_{py}, v_{pz}$ ):
    /* Cambiar ángulos y punto P al sistema Referencia Local      */
    /* Utilizar funcion de Jacobiano Metodo A                      */
2   [ $J_\theta, J_x, \theta_{31}, \theta_{32}, \theta_{33}, \theta_{21}, \theta_{22}, \theta_{23}, \dot{J}$ ] = jacobian_total_der( $P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13}$ )
    /* Matriz  $\dot{\theta}_{1i}$                                          */
3   
$$\begin{bmatrix} \dot{\theta}_{11} \\ \dot{\theta}_{12} \\ \dot{\theta}_{13} \end{bmatrix} = (J_\theta^{-1} * J_x) * \begin{bmatrix} v_{px} \\ v_{py} \\ v_{pz} \end{bmatrix}$$

    /* Velocidad angular  $\dot{\theta}_{3i}$                            */
4    $\dot{\theta}_{31} = calculo_theta3i_der(P_x, P_y, \phi_0^\circ, v_{px}, v_{py})$ 
5    $\dot{\theta}_{32} = calculo_theta3i_der(P_x, P_y, \phi_{120^\circ}, v_{px}, v_{py})$ 
6    $\dot{\theta}_{33} = calculo_theta3i_der(P_x, P_y, \phi_{240^\circ}, v_{px}, v_{py})$ 
    /* Velocidad angular  $\dot{\theta}_{2i}$                            */
7    $\dot{\theta}_{21} = calculo_theta2i_der(P_x, P_y, P_z, \phi_0^\circ, v_{px}, v_{py}, v_{pz}, \theta_{31}, \dot{\theta}_{31})$ 
8    $\dot{\theta}_{22} = calculo_theta2i_der(P_x, P_y, P_z, \phi_{120^\circ}, v_{px}, v_{py}, v_{pz}, \theta_{32}, \dot{\theta}_{32})$ 
9    $\dot{\theta}_{23} = calculo_theta2i_der(P_x, P_y, P_z, \phi_{240^\circ}, v_{px}, v_{py}, v_{pz}, \theta_{33}, \dot{\theta}_{33})$ 
    /*  $\dot{J}_{ix}$  para  $i \in \{1, 2, 3\}$                                 */
10   $\dot{J}_{ix} = calculo_jix_der(\phi_i, \theta_{1i}, \theta_{2i}, \theta_{3i}, \dot{\theta}_{1i}, \dot{\theta}_{2i}, \dot{\theta}_{3i})$ 
    /*  $\dot{J}_{iy}$  para  $i \in \{1, 2, 3\}$                                 */
11   $\dot{J}_{iy} = calculo_jiy_der(\phi_i, \theta_{1i}, \theta_{2i}, \theta_{3i}, \dot{\theta}_{1i}, \dot{\theta}_{2i}, \dot{\theta}_{3i})$ 
    /*  $\dot{J}_{iz}$  para  $i \in \{1, 2, 3\}$                                 */
12   $\dot{J}_{iz} = calculo_jiz_der(\phi_i, \theta_{1i}, \theta_{2i}, \theta_{3i}, \dot{\theta}_{1i}, \dot{\theta}_{2i}, \dot{\theta}_{3i})$ 
    /*  $\dot{J}_{i\theta}$  para  $i \in \{1, 2, 3\}$                                 */
13   $\dot{J}_{i\theta} = calculo_jthetai_der(\phi_i, \theta_{2i}, \theta_{3i}, \dot{\theta}_{2i}, \dot{\theta}_{3i})$ 
    /* Cambiar al sistema Referencia Global al operar           */
14  return [ $J_\theta, J_x, \dot{J}_\theta, \dot{J}_x, \theta_{31}, \theta_{32}, \theta_{33}, \theta_{21}, \theta_{22}, \theta_{23}$ ]

```

Algorithm 6: Aceleración método A (Continuacion...)

```

/* SUBFUNCIONES */  

/* Calcular  $\dot{\theta}_{3i}$  ecuacion (4.43) */  

1 calculo_theta3i_der( $P_x, P_y, \phi_i^\circ, v_{px}, v_{py}$ ):  

2    $c_{yi} = (-P_x) * (\sin(\phi_i)) + (P_y) * (\cos(\phi_i));$   

3    $c'_{yi} = [-\sin\phi_i * v_{px} + \cos\phi_i * v_{py}];$   

4    $A = \left[ \frac{c'_{yi}}{b} \right];$   

5    $B = \left[ -1 * \sqrt{1 - \left( \frac{c_{yi}}{b} \right)^2} \right];$   

6    $\dot{\theta}_{3i} = \frac{A}{B};$   

7   return  $\dot{\theta}_{3i};$   

/* Calcular  $\dot{\theta}_{2i}$  ecuacion (4.45) */  

8 calculo_theta2i_der( $P_x, P_y, P_z, \phi_i^\circ, v_{px}, v_{py}, v_{pz}, \theta_{3i}, \dot{\theta}_{3i}$ ):  

9    $c_{xi} = (P_x) * (\cos(\phi_i)) + (P_y) * (\sin(\phi_i)) + h - r;$   

10   $c_{yi} = (-P_x) * (\sin(\phi_i)) + (P_y) * (\cos(\phi_i));$   

11   $c_{zi} = P_z;$   

12   $c'_{xi} = \cos\phi_i * v_{px} + \sin\phi_i * v_{py};$   

13   $c'_{yi} = [-\sin\phi_i * v_{px} + \cos\phi_i * v_{py}];$   

14   $c'_{zi} = v_{pz};$   

15   $A_{11} = [2c_{xi}c'_{xi} + 2c_{yi}c'_{yi} + 2c_{zi}c'_{zi}] * [\sin\theta_{3i}];$   

16   $A_{12} = [c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2] * [\cos\theta_{3i} * \dot{\theta}_{3i}];$   

17   $B_1 = [\sin\theta_{3i}]^2;$   

18   $K' = \left[ \frac{1}{2ab} \right] * \left[ \frac{A_{11}-A_{12}}{B_1} \right];$   

19   $K = [c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2] / [2ab * \sin\theta_{3i}];$   

20   $\dot{\theta}_{2i} = \left[ \frac{-K'}{\sqrt{1-(K)^2}} \right];$   

21  return  $\dot{\theta}_{2i};$   

/* Calcular  $J_{ix}$  ecuacion (4.35) */  

22 calculo_jix_der( $\phi_i, \theta_{1i}, \theta_{2i}, \theta_{3i}, \dot{\theta}_{1i}, \dot{\theta}_{2i}, \dot{\theta}_{3i}$ ):  

23   $A_1 = -\sin(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin\theta_{3i};$   

24   $A_2 = \cos(\theta_{1i} + \theta_{2i}) * \cos(\theta_{3i}) * \dot{\theta}_{3i};$   

25   $A'_{ix} = (\cos(\phi_i)) * [A_1 + A_2];$   

26   $B'_{ix} = (\sin\phi_i) * [-\sin\theta_{3i} * \dot{\theta}_{3i}];$   

27   $J_{ix} = A'_{ix} - B'_{ix};$   

28  return  $J_{ix};$ 

```

Algorithm 6: Aceleración método A (Continuacion...)

```
/* SUBFUNCIONES */  
/* Calcular  $\dot{J}_{iy}$  ecuacion (4.38) */  
1 calculo_jiy_der( $\phi_i, \theta_{1i}, \theta_{2i}, \theta_{3i}, \dot{\theta}_{1i}, \dot{\theta}_{2i}, \dot{\theta}_{3i}$ ):  
2    $A_1 = [-\sin(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin \theta_{3i}];$   
3    $A_2 = + [\cos(\theta_{1i} + \theta_{2i}) * \cos \theta_{3i} * \dot{\theta}_{3i}];$   
4    $A'_{iy} = \sin \phi_i * (A_1 + A_2);$   
5    $B'_{iy} = \cos \phi_i * [-\sin \theta_{3i} * \dot{\theta}_{3i}];$   
6    $\dot{J}_{iy} = A'_{iy} + B'_{iy};$   
7   return  $\dot{J}_{iy};$   
/* Calcular  $\dot{J}_{iz}$  ecuacion (4.41) */  
8 calculo_jiz_der( $\theta_{1i}, \theta_{2i}, \theta_{3i}, \dot{\theta}_{1i}, \dot{\theta}_{2i}, \dot{\theta}_{3i}$ ):  
9    $A = [\cos(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin \theta_{3i}];$   
10   $B = [\sin(\theta_{1i} + \theta_{2i}) * \cos \theta_{3i} * \dot{\theta}_{3i}];$   
11   $\dot{J}_{iz} = A + B;$   
12  return  $\dot{J}_{iz};$   
/* Calcular  $\dot{J}_{i\theta}$  ecuacion (4.42) */  
13 calculo_jthetai_der( $\theta_{2i}, \theta_{3i}, \dot{\theta}_{2i}, \dot{\theta}_{3i}$ ):  
14   $A = \cos \theta_{2i} * \dot{\theta}_{2i} * \sin \theta_{3i};$   
15   $B = \sin \theta_{2i} * \cos \theta_{3i} * \dot{\theta}_{3i};$   
16   $\dot{J}_{i\theta} = a * (A + B);$   
17  return  $\dot{J}_{i\theta};$ 
```

6.1.2.5. Dinámica Inversa

Algorithm 7: Dinámica Inversa método A

Objetivo: Calcular el torque de los actuadores de un robot delta

Nombre archivo: *torque_m1_adams.py*

Input: $\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3, \theta_1, \theta_2, \theta_3, X_p, Y_p, Z_p, \ddot{X}_p, \ddot{Y}_p, \ddot{Z}_p, F_{px}, F_{py}, F_{pz}, m_{payload}$

Output: τ_2, τ_1, τ_3

```

/* FUNCION PRINCIPAL */
```

- 1 **ti**($\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3, \theta_1, \theta_2, \theta_3, X_p, Y_p, Z_p, \ddot{X}_p, \ddot{Y}_p, \ddot{Z}_p, F_{px}, F_{py}, F_{pz}, m_{payload}$):
- 2 /* Cambiar velocidades, aceleraciones, fuerzas y ordenar de
 ángulos según sistema de referencia local */
- 3 /* Cambiar unidades a SI */
- 4 /* Calcular multiplicadores de Lagrange λ_i $i \in \{1, 2, 3\}$ */
- 5 $[\lambda_1, \lambda_2, \lambda_3]^T = system_tq(\phi_{0^\circ}, \phi_{120^\circ}, \phi_{240^\circ}, \theta_1, \theta_2, \theta_3, X_p, Y_p, Z_p, F_{px}, F_{py}, F_{pz},$
- 6 $\ddot{X}_p, \ddot{Y}_p, \ddot{Z}_p, m_{payload})$
- 7 /* Calcular torques τ_i para $i \in \{1, 2, 3\}$ */
- 8 $\tau_1 = ti_puntual(\phi_{0^\circ}, \ddot{\theta}_1, \theta_1, X_p, Y_p, Z_p, \lambda_1)$
- 9 $\tau_2 = ti_puntual(\phi_{120^\circ}, \ddot{\theta}_2, \theta_2, X_p, Y_p, Z_p, \lambda_2)$
- 10 $\tau_3 = ti_puntual(\phi_{240^\circ}, \ddot{\theta}_3, \theta_3, X_p, Y_p, Z_p, \lambda_3)$
- 11 /* Cambiar orden de torques a sistema global */
- 12 **return** $[\tau_2, \tau_1, \tau_3]$

```

/* SUBFUNCIONES */
```

- 13 /* Calcular torques τ_i para $i \in \{1, 2, 3\}$ */
- 14 **ti_puntual**($\phi_i^\circ, \ddot{\theta}_i, \theta_i, X_p, Y_p, Z_p, \lambda_i$):
- 15 /* Resolver ecuacion (4.64) */
- 16 $A = \left(\frac{1}{3}m_1 + m_2\right) L_1^2 \ddot{\theta}_i$
- 17 $B = \left(\frac{1}{2}m_1 + m_2\right) g L_1 \cos \theta_i$
- 18 $C = 2\lambda_i L_1 [(X_P \cos \phi_i + Y_P \sin \phi_i - r) \sin \theta_i - Z_P \cos \theta_i]$
- 19 $\tau_i = A - B - C$
- 20 **return** τ_i

Algorithm 7: Dinámica Inversa método A (Continuación...)

```
/* SUBFUNCIONES */  
/* Calcular multiplicadores de Lagrange  $\lambda_i$   $i \in \{1, 2, 3\}$  */  
1 system_tq( $\phi_0^\circ, \phi_{120^\circ}, \phi_{240^\circ}, \theta_1, \theta_2, \theta_3, X_p, Y_p, Z_p, F_{px}, F_{py}, F_{pz}, \ddot{X}_p, \ddot{Y}_p, \ddot{Z}_p, m_{payload}$ ):  
    /* Resolver sistema ecuaciones  $A\lambda_i = B$ , modelado desde las  
    ecuaciones (4.65),(4.66),(4.67) */  
    /* Matriz A */  
2      $A_{0,0} = 2(X_p - r\cos\phi_1 - L_1\cos\theta_1\cos\phi_0^\circ);$   
3      $A_{0,1} = 2(X_p - r\cos\phi_2 - L_1\cos\theta_2\cos\phi_{120^\circ});$   
4      $A_{0,2} = 2(X_p - r\cos\phi_3 - L_1\cos\theta_3\cos\phi_{240^\circ});$   
5      $A_{1,0} = 2(Y_p - r\sin\phi_1 - L_1\cos\theta_1\sin\phi_0^\circ);$   
6      $A_{1,1} = 2(Y_p - r\sin\phi_2 - L_1\cos\theta_2\sin\phi_{120^\circ});$   
7      $A_{1,2} = 2(Y_p - r\sin\phi_3 - L_1\cos\theta_3\sin\phi_{240^\circ});$   
8      $A_{2,0} = 2(Z_p - L_1\sin\theta_1);$   
9      $A_{2,1} = 2(Z_p - L_1\sin\theta_2);$   
10     $A_{2,2} = 2(Z_p - L_1\sin\theta_3);$   
    /* Matriz B */  
    /* Al termino  $m_p$  se le suma el termino  $m_{payload}$  que es la masa del  
    objeto que mueve el robot delta, si es que moviera uno. */  
11     $B_{0,0} = [(m_p + m_{payload}) + 3m_2]\ddot{X}_p - F_{px};$   
12     $B_{1,0} = [(m_p + m_{payload}) + 3m_2]\ddot{Y}_p - F_{py};$   
13     $B_{2,0} = [(m_p + m_{payload}) + 3m_2][\ddot{Z}_p - g] - F_{pz};$   
    /* Calculo de  $\lambda_i$  */  
14     $[\lambda_1, \lambda_2, \lambda_3]^T = A^{-1}B;$   
15    return  $[\lambda_1, \lambda_2, \lambda_3]^T;$   
16
```

6.1.3. Método B

Al igual que la sección anterior, se presentan en las figuras (6.11) a la (6.15) las tareas con respecto al método B del capítulo (4).



Figura 6.11: Función cinemática directa del método B



Figura 6.12: Función cinemática inversa del método B

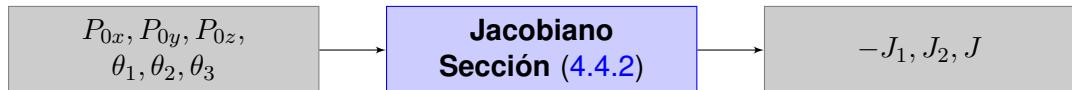


Figura 6.13: Función jacobiano del método B

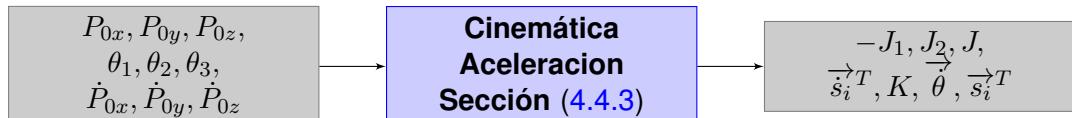


Figura 6.14: Función cinemática de aceleración del método B



Figura 6.15: Función dinámica inversa del método B

6.1.3.1. Cinemática directa

Algorithm 8: Cinemática Directa método B

Objetivo: Hallar la posición del efecto final del robot delta dada una configuración articular

Nombre archivo: *delta_kinematics_Paderborn_tm1_adams.py*

Input: $\theta_1, \theta_2, \theta_3$

Output: $P_0(P_{0x}, P_{0y}, P_{0z})$

/* FUNCION PRINCIPAL

*/

```

1 forward_Paderborn( $\theta_1, \theta_2, \theta_3$ ):
    /* Calcular Centros de esferas (Ecuaciones (4.75),(4.76),(4.77)) */
    /* Ordenar angulos segun sistema de referencia Local */
    2  $(x_1, y_1, z_1) = \vec{J}_2^T \left[ 0, -\frac{(f-e)}{2\sqrt{3}} - L_A \cos(\theta_2), -L_A \sin(\theta_2) \right]$ 
    3  $(x_2, y_2, z_2) = \vec{J}_3^T \left[ \left( \frac{(f-e)}{2\sqrt{3}} + L_A \cos(\theta_3) \right) \cos(30^\circ), \left( \frac{(f-e)}{2\sqrt{3}} + L_A \cos(\theta_3) \right) \sin(30^\circ), -L_A \sin(\theta_3) \right]$ 
    4  $(x_3, y_3, z_3) = \vec{J}_1^T \left[ \left( \frac{(f-e)}{2\sqrt{3}} + L_A \cos(\theta_1) \right) \cos(30^\circ), \left( \frac{(f-e)}{2\sqrt{3}} + L_A \cos(\theta_1) \right) \sin(30^\circ), -L_A \sin(\theta_1) \right]$ 
    /* Calcular intersección de las 3 esferas (Ecuaciones de la (4.3) a
       la (4.13))
    5  $d = (y_2 - y_1)x_3 - (y_3 - y_1)x_2 - (y_2 - y_3)x_1$ 
    6  $w_1 = x_1^2 + y_1^2 + z_1^2$ 
    7  $w_2 = x_2^2 + y_2^2 + z_2^2$ 
    8  $w_3 = x_3^2 + y_3^2 + z_3^2$ 
    9  $a_1 = \frac{(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)}{d}$ 
    10  $b_1 = \left( \frac{1}{2*(-d)} \right) * ((w_2 - w_1)(y_3 - y_1) - (w_3 - w_1)(y_2 - y_1))$ 
    11  $a_2 = \frac{-1}{d} * [(z_2 - z_1)x_3 - (z_3 - z_1)x_2 + (z_3 - z_2)x_1]$ 
    12  $b_2 = \frac{1}{2d} * [(w_2 - w_1)x_3 - (w_3 - w_1)x_2 + (w_3 - w_2)x_1]$ 
    13  $A = a_1^2 + a_2^2 + 1$ 
    14  $B = 2(a_1(b_1 - x_1) + a_2(b_2 - y_1) - z_1)$ 
    15  $C = ((b_1 - x_1)^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2)$ 
    16  $P_0(P_{0x}, P_{0y}, P_{0z}) = \left( a_1z + b_1, a_2z + b_2, \frac{-B - \sqrt{(B^2) - (4AC)}}{2A} \right)$ 
    17 return  $(P_{0x}, P_{0y}, P_{0z})$ 

```

6.1.3.2. Cinemática inversa

Algorithm 9: Cinemática Inversa método B

Objetivo: Hallar la posición de los actuadores en el espacio articular dada la posición del centroide del efecto final

Nombre archivo: *delta_kinematics_Paderborn_tm1_adams.py*

Input: $P_0(P_{0x}, P_{0y}, P_{0z})$

Output: $\theta_1, \theta_2, \theta_3$

```
/* FUNCION PRINCIPAL
1 inverse_Paderborn( $P_{0x}, P_{0y}, P_{0z}$ ):
    /* Rotación del sistema de referencia Local en 0°          */
2      $x_0^\circ = P_{0x}$ 
3      $y_0^\circ = P_{0y}$ 
4      $z_0^\circ = P_{0z}$ 
    /* Rotación del sistema de referencia Local en 120°        */
5      $x_{120^\circ} = P_{0x} * \cos 120 + P_{0y} * \sin 120$ 
6      $y_{120^\circ} = P_{0y} * \cos 120 - P_{0x} * \sin 120$ 
7      $z_{120^\circ} = P_{0z}$ 
    /* Rotación del sistema de referencia Local en 240°        */
8      $x_{240^\circ} = P_{0x} * \cos 120 - P_{0y} * \sin 120$ 
9      $y_{240^\circ} = P_{0y} * \cos 120 + P_{0x} * \sin 120$ 
10     $z_{240^\circ} = P_{0z}$ 
    /* Calcular angulos de los actuadores                      */
    /* Cambiar orden de ángulos al sistema Referencia Global   */
11     $\theta_2 = angle_yz_Paderborn(x_0^\circ, y_0^\circ, z_0^\circ, 0^\circ)$ 
12     $\theta_3 = angle_yz_Paderborn(x_{120^\circ}, y_{120^\circ}, z_{120^\circ}, 120^\circ)$ 
13     $\theta_1 = angle_yz_Paderborn(x_{240^\circ}, y_{240^\circ}, z_{240^\circ}, 240^\circ)$ 
14    return [ $\theta_1, \theta_2, \theta_3$ ]
```

Algorithm 9: Cinemática Inversa método B (Continuacion...)

```
/* SUBFUNCIONES */  
1 angle_yz_Paderborn( $x_{\varphi_i}, y_{\varphi_i}, z_{\varphi_i}, \varphi_i$ ):  
    /* Actuador  $F_i$  en el plano  $Y'Z'$  (sistema de referencia Local  
    rotado en  $\varphi_i$ ) */  
    2  $y_1 = y_{F_i} = -\frac{f}{2\sqrt{3}};$   
    3  $z_1 = z_{F_i} = 0;$   
    /* Proyección de la Junta Esférica  $P_0$  en el plano  $Y'Z'$  (sistema de  
    referencia Local rotado en  $\varphi_i$ ) */  
    4  $y_2 = y_{P'_0} = y_{\varphi_i} - \frac{e}{2\sqrt{3}};$   
    5  $z_2 = z_{P'_0} = z_{\varphi_i};$   
    /* Junta Esférica  $J_i$  de la cadena cinemática i en posición  $\varphi_i$  */  
    6  $a = \frac{x_{\varphi_i}^2 + y_2^2 + z_{\varphi_i}^2 + R_A^2 - R_B^2 - y_1^2}{2z_{\varphi_i}},$   
    7  $b = \frac{(y_1 - y_2)}{z_2},$   
    8  $x_{J_i} = 0;$   
    9  $y_{J_i} = \frac{(y_1 - ab) - \sqrt{-(a+by_1)^2 + (b^2+1)R_A^2}}{b^2+1};$   
    10  $z_{J_i} = a + b * y_{J_i};$   
    /* Calculo del Angulo  $\theta_i$  de la cadena cinemática i en posición  $\varphi_i$  */  
    11  $\theta_i = \arctan\left(\frac{z_{J_i}}{y_{F_i} - y_{J_i}}\right);$   
    12 return  $\theta_i;$   
13
```

6.1.3.3. Jacobiano

Algorithm 10: Jacobiano método B

Objetivo: Hallar la matriz jacobiana

Nombre archivo: *jacobian_Paderborn_tm1_v2_adams.py*

Input: $P_{0x}, P_{0y}, P_{0z}, \theta_1, \theta_2, \theta_3$

Output: $-J_1, J_2, J$

```

/* FUNCION PRINCIPAL */

1 jacobian_calculo_Paderborn( $P_{0x}, P_{0y}, P_{0z}, \theta_{11}, \theta_{12}, \theta_{13}$ ):
    /* Cambiar ángulos y punto P al sistema Referencia Local */ 
    /* Posición actuadores  $\varphi$  respecto a ejes YX S.ref Local */
2    $(\phi_{0^\circ}, \phi_{120^\circ}, \phi_{240^\circ}) = (0, 120, 240)$ 
    /* Calcular  $\vec{s}_i^T$  Transpose */ 
3    $\vec{s}_1^T = si(P_{0x}, P_{0y}, P_{0z}, \theta_1, \varphi_{0^\circ})^T$ 
4    $\vec{s}_2^T = si(P_{0x}, P_{0y}, P_{0z}, \theta_2, \varphi_{120^\circ})^T$ 
5    $\vec{s}_3^T = si(P_{0x}, P_{0y}, P_{0z}, \theta_3, \varphi_{240^\circ})^T$ 
    /* Calcular  $\vec{b}_i$  */ 
6    $\vec{b}_1 = bi(\theta_1, \varphi_{0^\circ})$ 
7    $\vec{b}_2 = bi(\theta_2, \varphi_{120^\circ})$ 
8    $\vec{b}_3 = bi(\theta_3, \varphi_{240^\circ})$ 
    /* Calcular componentes  $J_1$  */ 
9   
$$J_1 = \begin{bmatrix} \vec{s}_1^T(0) & \vec{s}_1^T(1) & \vec{s}_1^T(2) \\ \vec{s}_2^T(0) & \vec{s}_2^T(1) & \vec{s}_2^T(2) \\ \vec{s}_3^T(0) & \vec{s}_3^T(1) & \vec{s}_3^T(2) \end{bmatrix}^{-1}$$

    /* Calcular  $J_2$  */ 
10  
$$J_2 = \begin{bmatrix} \vec{s}_1^T \cdot \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \cdot \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \cdot \vec{b}_3 \end{bmatrix}$$

    /* Calcular  $J$  ecuación (4.91) */ 
11   $J = -J_1 J_2$ 
    /* Cambiar al sistema Referencia Global al operar  $J$  */ 
12  return  $[-J_1, J_2, J]$ 

```

Algorithm 10: Jacobiano método B (Continuacion...)

```
/* SUBFUNCIONES */  
/* Calcular  $\vec{s}_i$  */  
1 si( $P_{0x}, P_{0y}, P_{0z}, \theta_i, \varphi_i$ ):  
    /* Vector efector final */  
    2  $\vec{P}_0 = \begin{bmatrix} P_{0x} \\ P_{0y} \\ P_{0z} \end{bmatrix};$   
    /* Matriz de rotacion respecto a ángulo  $\varphi_i$  */  
    3  $R_i^R = matri\_rot(\varphi_i);$   
    /* Vector posicion actuador i respecto a sistema referencia local */  
    4  $R = R_A - R_B;$   
    5  $\vec{F}_i = R_i^R \begin{bmatrix} R \\ 0 \\ 0 \end{bmatrix};$   
    /* Vector brazo i respecto a sistema referencia local */  
    6  $\vec{\xi}_{F_i J_i} = R_i^R \begin{bmatrix} L_A \cos(\theta_i) \\ 0 \\ -L_A \sin(\theta_i) \end{bmatrix};$   
    /* calculo  $\vec{s}_i$  ecuacion (4.86) */  
    7  $\vec{s}_i = \vec{P}_0 - (\vec{F}_i + \vec{\xi}_{F_i J_i});$   
    return  $\vec{s}_i;$   
/* Calcular  $\vec{b}_i$  */  
9 bi( $\theta_i, \varphi_i$ ):  
    /* Matriz de rotacion respecto a ángulo  $\varphi_i$  */  
    10  $R_i^R = matri\_rot(\varphi_i);$   
    /* calculo  $\vec{b}_i$  ecuacion (4.92) */  
    11  $\vec{b}_i = R_i^R \begin{bmatrix} L_A \sin(\theta_i) \\ 0 \\ L_A \cos(\theta_i) \end{bmatrix};$   
    return  $\vec{b}_i;$   
/* Calcular Matriz de rotacion  $R_i^R$  ecuacion (4.87) */  
13 matri_rot( $\varphi_i$ ):  
    14  $R_i^R = \begin{bmatrix} \cos(\varphi_i) & -\sin(\varphi_i) & 0 \\ \sin(\varphi_i) & \cos(\varphi_i) & 0 \\ 0 & 0 & 1 \end{bmatrix};$   
    return  $R_i^R;$ 
```

6.1.3.4. Modelacion Cinematica de Aceleracion

Algorithm 11: Modelacion Cinematica de Aceleracion método B

Objetivo: Hallar la matrices para calcular la aceleración angular de los actuadores

Nombre archivo: jacobian_Paderborn_tm1_v2_adams.py

Input: $P_{0x}, P_{0y}, P_{0z}, \theta_1, \theta_2, \theta_3, \dot{P}_{0x}, \dot{P}_{0y}, \dot{P}_{0z}$

Output: $-J_1, J_2, J, \vec{s}_i^T, K, \vec{\theta}, \vec{s}_i^T$

/* FUNCION PRINCIPAL */

```

1 jacobian_total_tm1( $P_{0x}, P_{0y}, P_{0z}, \theta_1, \theta_2, \theta_3, \dot{P}_{0x}, \dot{P}_{0y}, \dot{P}_{0z}$ ):
    /* Cambiar ángulos y punto P al sistema Referencia Local */
    /* Posición actuadores  $\varphi$  respecto a ejes YX S.ref Local */
    /* Usar función de jacobiano metodo B */

2    $[-J_1, J_2, J] = jacobian\_calculo\_Paderborn(P_{0x}, P_{0y}, P_{0z}, \theta_1, \theta_2, \theta_3)$ 
    /* Calcular matriz  $\vec{\theta}$  */

3    $\vec{\theta} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} = J^{-1} * \begin{bmatrix} \dot{P}_{0x} \\ \dot{P}_{0y} \\ \dot{P}_{0z} \end{bmatrix} = J^{-1} * \vec{P}_0$ 
    /* Calcular Transpose  $\vec{s}_i^T$  para  $i = 1, 2, 3$  (1x3) */

4    $\vec{s}_1^T = [si\_p(\dot{P}_{0x}, \dot{P}_{0y}, \dot{P}_{0z}, \varphi_{0^\circ}, \theta_1, \dot{\theta}_1)]^T$ 
5    $\vec{s}_2^T = [si\_p(\dot{P}_{0x}, \dot{P}_{0y}, \dot{P}_{0z}, \varphi_{120^\circ}, \theta_2, \dot{\theta}_2)]^T$ 
6    $\vec{s}_3^T = [si\_p(\dot{P}_{0x}, \dot{P}_{0y}, \dot{P}_{0z}, \varphi_{240^\circ}, \theta_3, \dot{\theta}_3)]^T$ 
    /* Calcular Matriz  $\vec{s}_i^T$  (3x3) */

7    $\vec{s}_i^T = \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}$ 
    /* Calcular  $\vec{b}_i$  */

8    $\vec{b}_1 = bi\_p(\varphi_{0^\circ}, \theta_1, \dot{\theta}_1)$ 
9    $\vec{b}_2 = bi\_p(\varphi_{120^\circ}, \theta_2, \dot{\theta}_2)$ 
10   $\vec{b}_3 = bi\_p(\varphi_{240^\circ}, \theta_3, \dot{\theta}_3)$ 
    /* Calcular Matriz K ecuacion (4.94) */

11   $K = \begin{bmatrix} \vec{s}_1^T \vec{b}_1 + \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 + \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 + \vec{s}_3^T \vec{b}_3 \end{bmatrix}$ 
12  return  $[-J_1, J_2, J, \vec{s}_i^T, K, \vec{\theta}, \vec{s}_i^T]$ 

```

Algorithm 11: Modelacion Cinematica de Aceleracion método B (Continuacion...)

```

/* SUBFUNCIONES */  

/* Calcular Transpose  $\vec{s}_i$  para  $i = 1, 2, 3$  (1x3) */  

1 si_p( $\dot{P}_{0x}, \dot{P}_{0y}, \dot{P}_{0z}, \varphi^\circ, \theta_i, \dot{\theta}_i$ ):  

    /* calculo ecuación (4.96) */  

    2 
$$\vec{s}_i = \begin{bmatrix} \dot{P}_{0x} \\ \dot{P}_{0y} \\ \dot{P}_{0z} \end{bmatrix} + R_i^R * \begin{bmatrix} L_A \sin(\theta_i) \\ 0 \\ L_A \cos(\theta_i) \end{bmatrix} \dot{\theta}_i = \vec{P}_0 + \vec{b}_i \dot{\theta}_i;$$
  

    3 return  $\vec{s}_i$   

/* Calcular  $\vec{b}_i$  */  

4 bi_p( $\varphi^\circ, \theta_i, \dot{\theta}_i$ ):  

    /* calculo ecuacion (4.95) */  

    5 
$$\vec{b}_i = \begin{bmatrix} L_A \cos(\theta_i) \\ 0 \\ -L_A \sin(\theta_i) \end{bmatrix} \vec{\dot{\theta}}_i;$$
  

    6 return  $\vec{b}_i$ 

```

6.1.3.5. Dinámica Inversa

Algorithm 12: Dinámica Inversa método B

Objetivo: Calcular el torque de los actuadores de un robot delta

Nombre archivo: *torque_m1_Paderborn_v2_adams.py*

Input: $\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3, \theta_1, \theta_2, \theta_3, F_{px}, F_{py}, F_{pz}, J, m_{payload}, \dot{J}, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3$

Output: τ_1, τ_2, τ_3

```

/* FUNCION PRINCIPAL                                         */
1 ti(\ddot{\theta}_1,\ddot{\theta}_2,\ddot{\theta}_3,\theta_1,\theta_2,\theta_3,F_{px},F_{py},F_{pz},J,m_{payload},\dot{J},\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3):
    /* Cambiar velocidades, aceleraciones, fuerzas y ordenar de      */
    /* ángulos según sistema de referencia local                      */
    /* Cambiar unidades a SI                                         */
    /* Calcular torque de los actuadores \tau_1,\tau_2,\tau_3          */
2   \tau = ti_puntual(\theta_1,\theta_2,\theta_3,\ddot{\theta}_1,\ddot{\theta}_2,\ddot{\theta}_3,J,F_{px},F_{py},F_{pz},m_{payload},\dot{J},\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3)
3   [\tau_1,\tau_2,\tau_3] = \tau
    /* Cambiar orden de torques a sistema global                     */
4   return [\tau_1,\tau_2,\tau_3]                                     */

/* SUBFUNCIONES                                         */
/* Calcular torque \tau = [\tau_1,\tau_2,\tau_3]                  */
5   ti_puntual(\theta_1,\theta_2,\theta_3,\ddot{\theta}_1,\ddot{\theta}_2,\ddot{\theta}_3,J,F_{px},F_{py},F_{pz},m_{payload},\dot{J},\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3):
6     \vec{\theta} = [\theta_1,\theta_2,\theta_3]^T
7     \vec{\ddot{\theta}} = [\ddot{\theta}_1,\ddot{\theta}_2,\ddot{\theta}_3]^T
8     \vec{\dot{\theta}} = [\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3]^T
9     \vec{F_g} = [F_{px},F_{py},F_{pz}]^T
10    m_{nt} = mnt(m_{payload})
11    I_b = inercia_b()
        /* Resolver ecuación (4.102)                                */
12    M_1(\theta) = I_b \vec{\theta}
13    M_2(\theta) = J^T m_{mt} J \vec{\dot{\theta}}
14    C(\theta, \dot{\theta}) = J^T m_{mt} \dot{J} \vec{\dot{\theta}}
15    \vec{G_1}(\theta) = -J^T \vec{F_g}
16    \vec{G_2}(\theta) = \vec{\tau_{Gb}} = -g * CoM * (m_a + m_codo + 2rm_b)
17    \vec{\tau} = M_1(\theta) \vec{\theta} + M_2(\theta) \vec{\dot{\theta}} + C(\theta, \dot{\theta}) \vec{\dot{\theta}} + \vec{G_1}(\theta) + \vec{G_2}(\theta)
18    return [\tau_1,\tau_2,\tau_3]

```

Algorithm 12: Dinámica Inversa método B (Continuación...)

```
/* SUBFUNCIONES */  
/* Calcular masa total del efecto final */  
1 mnt( $m_{payload}$ ):  
2    $m_{nt} = m_c + m_{payload} + 3 * 2 * (1 - r) * m_b;$   
3   return  $m_{nt};$   
/* Calcular matriz de Inercia */  
4 inercia_b():  
5    $I_{bi} = I_m + L_A^2 \left( \frac{m_a}{3} + m_{codo} + 2 * r * m_b \right);$   
6   
$$I_b = \begin{bmatrix} I_{bi} & 0 & 0 \\ 0 & I_{bi} & 0 \\ 0 & 0 & I_{bi} \end{bmatrix};$$
  
7   return  $m_{nt};$ 
```

6.1.4. Espacio de Trabajo

Encontrar el espacio de trabajo de un robot delta significa calcular todas las posiciones posibles y viables del efecto final. El siguiente diagrama de flujo representa una de las muchas soluciones que existen, donde las dimensiones del robot y los parámetros de las restricciones son valores preestablecidos. Es importante recalcar que la nomenclatura en esta sección es la misma que se implementa para determinar el jacobiano del robot delta de la sección (4.3.2.3).

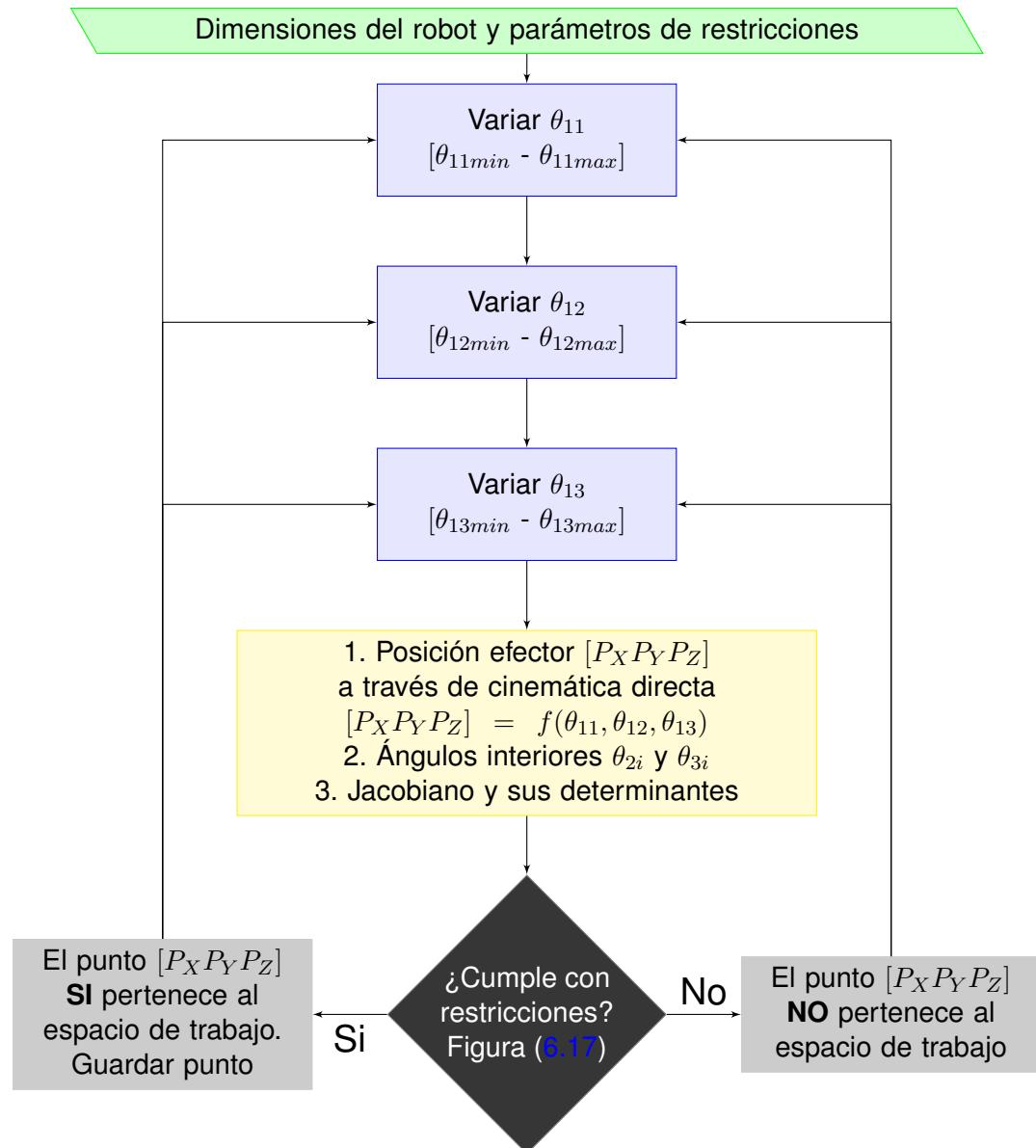


Figura 6.16: Diagrama de flujo para la solución del espacio de trabajo

La idea principal de la solución propuesta es calcular un grupo significativo de configuraciones (P_x, P_y, P_z) posibles del efecto final en base a sus dimensiones y evaluar si cumplen con restricciones impuestas. Se obtienen las configuraciones cartesianas del efecto (P_x, P_y, P_z) utilizando la cinemática directa y variando los angulos $\theta_{11}, \theta_{12}, \theta_{13}$ en los rangos $[\theta_{11min} - \theta_{11max}], [\theta_{12min} - \theta_{12max}], [\theta_{13min} - \theta_{13max}]$ respectivamente. Todos estos rangos se discretizan con un paso impuesto $\Delta\theta_{1i}$. Si el punto (P_x, P_y, P_z) cumple con todas las restricciones, el punto pertenece al espacio de trabajo. Al contrario, si el punto no cumple con una o mas restricciones, este no pertenece al espacio de trabajo.

Las restricciones son las mismas que las expuestas en la sección (4.5.3). El orden en que se verifican las restricciones se muestran en el siguiente diagrama de flujo:

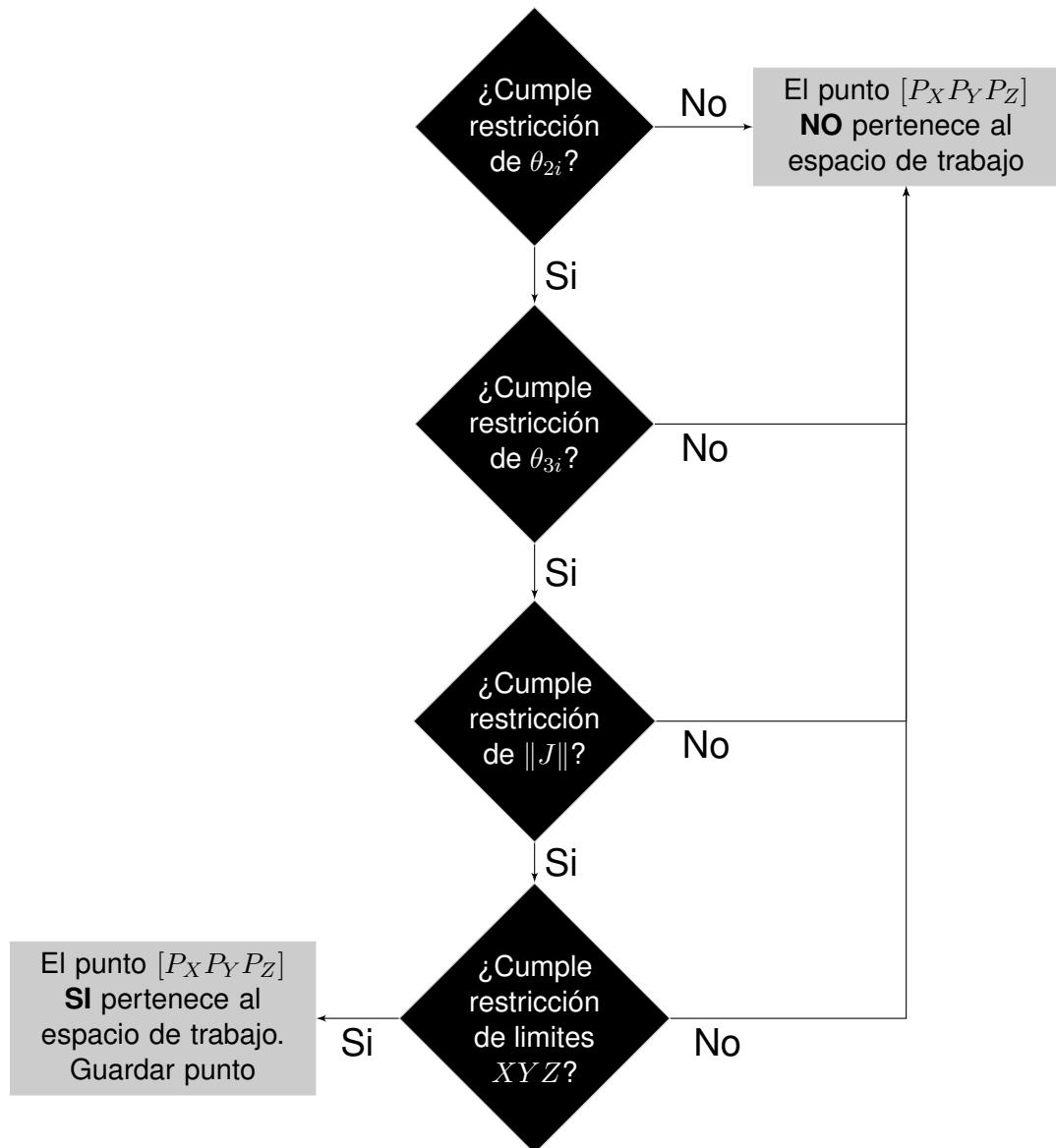


Figura 6.17: Diagrama de flujo de restricciones para el espacio de trabajo

Para calcular el espacio de trabajo se crea un nodo en ROS. La tarea específica de este nodo es calcular, guardar y graficar los puntos (P_x, P_y, P_z) del efecto final que cumplen con las restricciones impuestas. La representación gráfica del nodo se presenta en la figura (6.18):

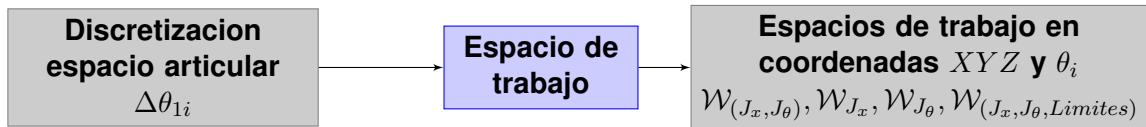


Figura 6.18: Entradas y salidas de la función de espacio de trabajo

En este nodo, las dimensiones del robot y las restricciones son establecidas antes de calcular el espacio de trabajo. La única entrada es el paso de la discretización de los rangos $\Delta\theta_{1i}$. La cinemática directa y el jacobiano se calculan a partir del método A, específicamente los desarrollados en las secciones (4.3.1.2) y (4.3.2.3) respectivamente. Se emplea este método por la razón de que el jacobiano J se puede bipartir en dos matrices (J_x y J_θ), donde cada una de ellas representa una singularidad específica. Además, el método A tiene la ventaja de tener las ecuaciones de los ángulos interiores θ_{2i} y θ_{3i} .

El valor de las restricciones en este trabajo de grado se encuentran en la tabla (6.2).

Restricción	Explicación	Min	Max
θ_{1i}	Colisión entre los brazos y la base fija	-90°	90°
$\Delta\theta_{1i}$	Pasos de discretización de rangos θ_{1i} muy bajos implican costo computacional alto	5°	
θ_{2i}	Ángulo debe ser menor a 180° ya que en esa situación el brazo y el antebrazo podrían ser colineales y producir singularidades.	5°	175°
θ_{3i}	Ángulo respecto a la inclinación máxima de las rotulas por catálogo, generalmente son de 13°	45°	135°
J_x	Depende de la precisión o factor de seguridad subjetivo	$6 * 10^{-1}$	
J_θ	Depende de la precisión o factor de seguridad subjetivo	$4 * 10^{-3}$	
X	Límite X de espacio de trabajo impuesto por el fabricante	-400[mm]	+400[mm]
Y	Límite Y de espacio de trabajo impuesto por el fabricante	-400[mm]	+400[mm]
Z	Límite Z de espacio de trabajo impuesto por el fabricante	-300[mm]	-750[mm]

Tabla 6.2: Restricciones del espacio de trabajo

El siguiente algoritmo presenta el pseudocódigo de la función dentro del nodo que determinar el espacio de trabajo:

Algorithm 13: Espacio de Trabajo

Objetivo: Encontrar el espacio de trabajo del robot delta a partir de sus dimensiones y restricciones impuestas.

Nombre archivo: *workspace_v2.py*

Input: $\Delta\theta_{1i}$

Output: $[\mathcal{W}_{(J_x, J_\theta)}, \mathcal{W}_{J_x}, \mathcal{W}_{J_\theta}, \mathcal{W}_{(J_x, J_\theta, Limites)}]$

/* FUNCION PRINCIPAL */

```

1 espaciotrabajo( $\Delta\theta_{1i}$ ):
    /* Barrido de angulos  $\theta_{11}$  ,  $\theta_{12}$  y  $\theta_{13}$  con incrementos de  $\Delta\theta_{1i}$  */
    2 for  $\theta_{11} \in [\theta_{11min} - \theta_{11max}]; \Delta\theta_{1i}$  do
        3     for  $\theta_{12} \in [\theta_{12min} - \theta_{12max}]; \Delta\theta_{1i}$  do
            4         for  $\theta_{13} \in [\theta_{13min} - \theta_{13max}]; \Delta\theta_{1i}$  do
                /* Cinematica Directa */
                5                 ( $P_x, P_y, P_z$ ) = forward( $\theta_{12}, \theta_{13}, \theta_{11}$ )
                /* Jacobiano */
                6                 [ $J_\theta, J_x, \theta_{31}, \theta_{32}, \theta_{33}, \theta_{21}, \theta_{22}, \theta_{23}, J$ ] =
                    jacobian_total( $P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13}$ )
                /* Determinantes Jacobiano */
                7                  $|J_\theta| = \det(J_\theta)$ 
                8                  $|J_x| = \det(J_x)$ 
                /* Restricciones */
                9                 if Restricciones  $\theta_{2i}$  then
                    10                     if Restricciones  $\theta_{3i}$  then
                        11                         if Restricciones  $|J_x| \circ |J_\theta|$  then
                            12                              $\mathcal{W}_{(J_x, J_\theta)} = (P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13})$ 
                            13                             if Restricciones LimitesXYZ then
                                14                                  $\mathcal{W}_{(J_x, J_\theta, limites)} = (P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13})$ 
                            15                         else
                                16                             ( $P_x, P_y, P_z$ ) No pertenece al espacio de trabajo
                            17                         if Restricciones  $|J_x|$  then
                                18                              $\mathcal{W}_{J_x} = (P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13})$ 
                            19                         if Restricciones  $|J_\theta|$  then
                                20                              $\mathcal{W}_{J_\theta} = (P_x, P_y, P_z, \theta_{11}, \theta_{12}, \theta_{13})$ 
                21             return  $[\mathcal{W}_{(J_x, J_\theta)}, \mathcal{W}_{J_x}, \mathcal{W}_{J_\theta}, \mathcal{W}_{(J_x, J_\theta, Limites)}]$ 

```

6.1.5. Trayectoria

En esta sección se explica el desarrollo para determinar la trayectoria en el espacio articular del robot delta, es decir, la trayectoria angular de los actuadores a partir de una trayectoria lineal interpolada en el espacio cartesiano XYZ que recorre el efecto final de un punto inicial P_i a un punto final P_f con restricciones impuestas por el perfil de velocidad trapezoidal en dirección del camino geométrico.



Figura 6.19: Entradas y salidas de la función de trayectoria

Esta trayectoria se compone de un camino geométrico lineal en el espacio cartesiano XYZ llamado 'punto a punto en linea recta' y la escala temporal es 'perfil de movimiento trapezoidal respecto a la Velocidad' o 'linear segment with parabolic blends (LSPB)', visto en la sección (4.6).

El diagrama de flujo para el desarrollo de la trayectoria se presenta en la figura (6.20).

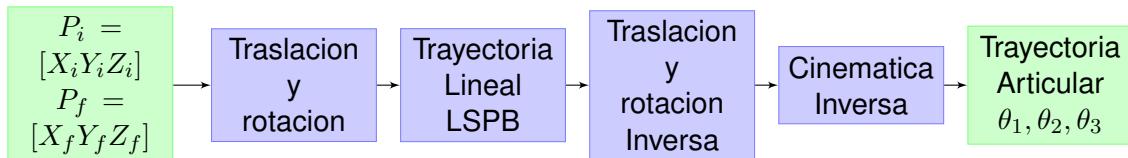


Figura 6.20: Flujo de trabajo para la creacion de trayectorias

A partir del punto inicial P_i y el punto final P_f , se realiza una traslación y dos rotaciones del marco de referencia global para aplicar la escala temporal de perfil trapezoidal y la interpolación del camino geométrico lineal. Luego se aplica traslación y rotaciones inversas para obtener la trayectoria en el espacio cartesiano respecto al sistema de referencia global. Finalmente se emplea la cinemática inversa para conseguir las trayectorias en el espacio articular de los actuadores.

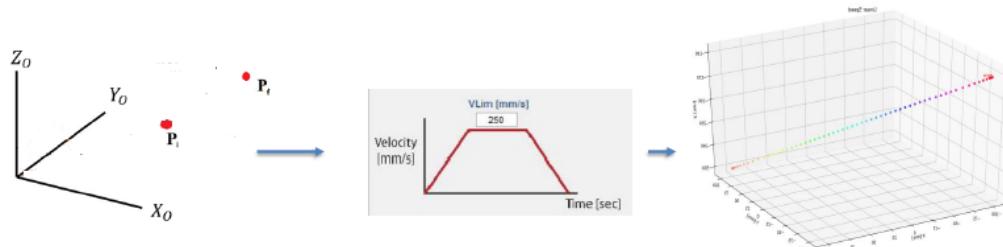


Figura 6.21: Ejemplo gráfico del diagrama de flujo para la creación de trayectoria lineal con perfil trapezoidal.

El perfil de movimiento trapezoidal visto en la sección (4.6.5.2.2) se puede modificar para que las ecuaciones consideren el punto inicial y final especificando la velocidad máxima y la aceleración máxima de la siguiente manera [66]:

$$q(t) = \begin{cases} q_0 + s\frac{a}{2}t^2 & 0 < t \leq \tau \\ q_0 - s\frac{V^2}{2a} + sVt & \tau < t \leq T - \tau \\ q_1 + s\left(-\frac{aT^2}{2} + aTt - \frac{a}{2}t^2\right) & T - \tau < t \leq T \end{cases} \quad (6.1)$$

Donde q_0 es el punto inicial, q_1 es el punto final, V es la velocidad máxima permitida y a es la aceleración máxima permitida.

$$\tau = \frac{V}{a} \quad (6.2)$$

$$T = s\frac{q_1 - q_0}{V} + \frac{V}{a} \quad (6.3)$$

$$s = signo(q_1 - q_0) \quad (6.4)$$

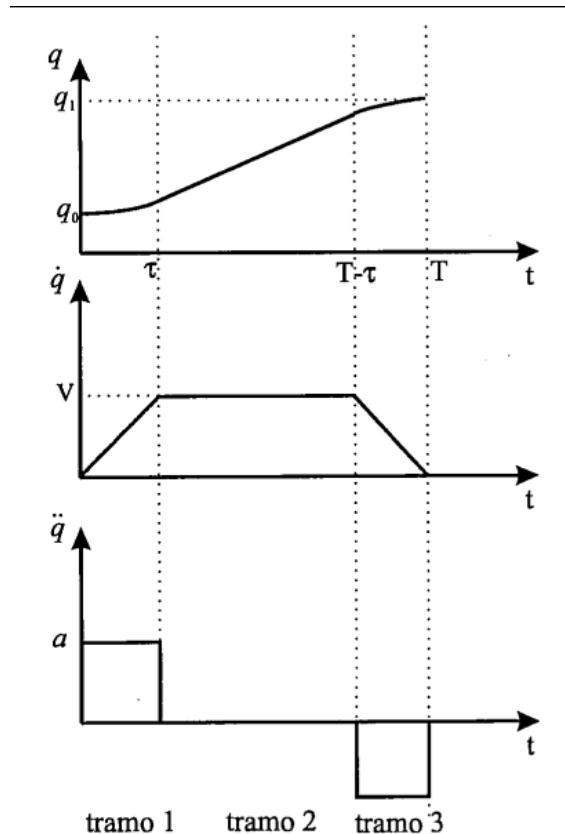


Figura 6.22: Interpolator trapezoidal [66].

Para utilizar el perfil de velocidad trapezoidal en el camino geométrico lineal interpolado desde el punto inicial P_i al punto final P_f se aplican matrices de rotaciones.

Primero se traslada el sistema de referencia global $O-X_0Y_0Z_0$ al punto inicial P_i generando un nuevo sistema de referencia $X_{trans}Y_{trans}Z_{trans}$.

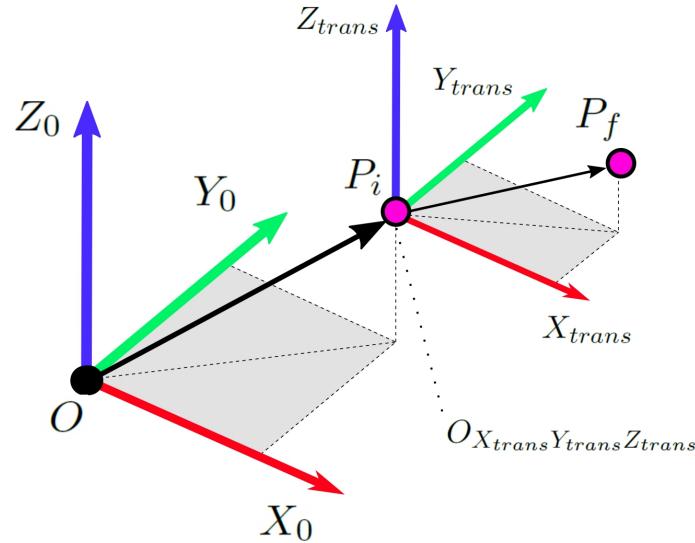


Figura 6.23: Traslación marco de referencia $X_0Y_0Z_0$ hasta punto P_i

Luego se rota $X_{trans}Y_{trans}Z_{trans}$ sobre el eje Z_{trans} en un ángulo θ_z generando un nuevo sistema de referencia $X_{rot\theta_z}Y_{rot\theta_z}Z_{rot\theta_z}$. θ_z es el angulo interno entre el eje X_{trans} y el vector $\overrightarrow{O_{X_{trans}Y_{trans}Z_{trans}}P'_f}$ donde P'_f es la proyección de P_f sobre el plano $Y_{trans}Z_{trans}$.

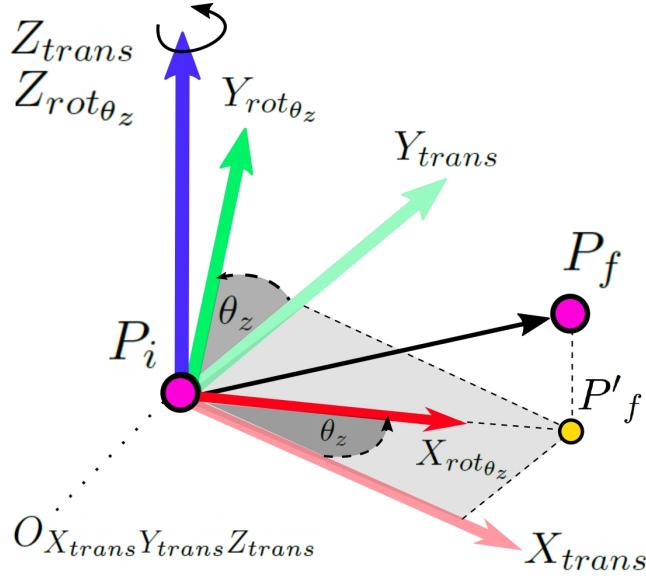


Figura 6.24: Rotación marco de referencia $X_{trans}Y_{trans}Z_{trans}$ sobre eje Z_{trans} en un angulo de θ_z .

Finalmente el marco de referencia $X_{rot\theta_z} Y_{rot\theta_z} Z_{rot\theta_z}$ se rota sobre el eje $Y_{rot\theta_z}$ en un angulo de θ_y generando el ultimo sistema de referencia $X_{rot\theta_y} Y_{rot\theta_y} Z_{rot\theta_y}$. θ_y es el angulo interno entre el eje $X_{rot\theta_z}$ y el vector $\overrightarrow{O_{X_{rot\theta_z} Y_{rot\theta_z} Z_{rot\theta_z}} P''_f}$ donde P''_f es el punto P_f con respecto al sistema de referencia $X_{rot\theta_z} Y_{rot\theta_z} Z_{rot\theta_z}$. En el marco de referencia $X_{rot\theta_y} Y_{rot\theta_y} Z_{rot\theta_y}$ se aplica el perfil de velocidad trapezoidal que define la escala temporal de la trayectoria y la interpolación en el espacio cartesiano del camino geométrico lineal.

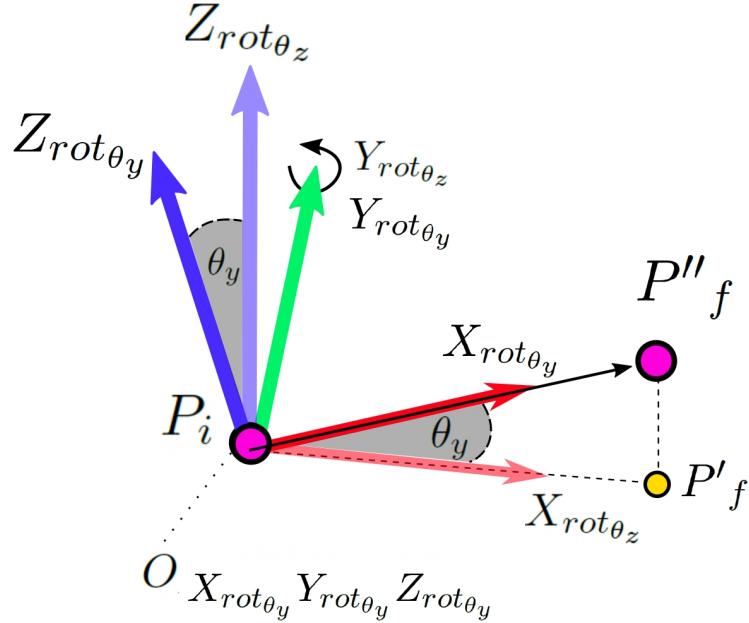


Figura 6.25: Rotación marco de referencia $X_{rot\theta_z} Y_{rot\theta_z} Z_{rot\theta_z}$ sobre eje $Y_{rot\theta_z}$ en un angulo de θ_y .

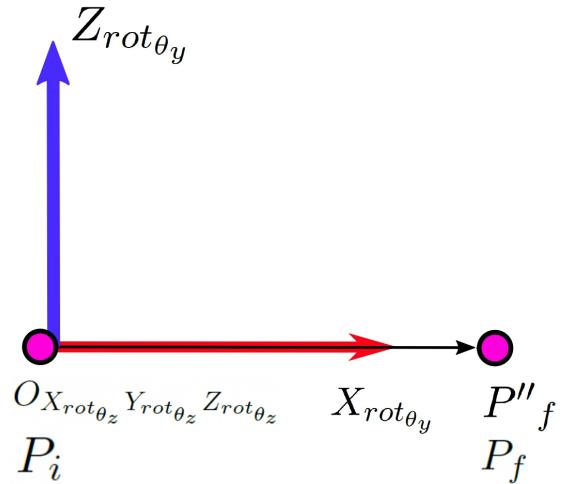


Figura 6.26: Marco de referencia $X_{rot\theta_y} Y_{rot\theta_y} Z_{rot\theta_y}$ en donde se aplica LSPB.

Algorithm 14: Trayectoria

Objetivo: Escala de tiempo perfil de velocidad trapezoidal (LSPB) e interpolación de camino geométrico lineal

Nombre archivo: *linear_speed_f_adams.py*

Input: $q_0, q_1, v_{max}, a_{max}, n_{\Delta Tramo1}, n_{\Delta Tramo2}$

Output: $tiempo, X_{rot_{\theta_y}}, \dot{X}_{rot_{\theta_y}}, \ddot{X}_{rot_{\theta_y}}$

/* FUNCION PRINCIPAL */

1 **ls_v_a_total**($q_0, q_1, v_{max}, a_{max}, n_{\Delta Tramo1}, n_{\Delta Tramo2}$):

 /* Parametros de Curva LSPB */

 2 $n_{\Delta Tramo3} = n_{\Delta Tramo1}$ // División Tramo 3 = Tramo 1

 3 $\tau = v_{max}/a_{max}$ // Tiempo total del tramo 1 y tramo 3

 4 $s = signo(q_1 - q_0)$

 5 $T = s \frac{(q_1 - q_0)}{v_{max}} + \tau$ // Tiempo total de la trayectoria

 6 $\Delta_{tramo1} = (\tau)/n_{\Delta Tramo1}$

 7 $\Delta_{tramo2} = (T - 2\tau)/n_{\Delta Tramo2}$

 8 $\Delta_{tramo3} = \Delta_{tramo1}$

 9 $n_{total} = \sum_{i=1}^3 n_{\Delta Tramo i} + 1$ // Total puntos de discretizacion

 /* Verificar si alcanza la v_{max} para v_{max} */

 10 $T_f = 2\sqrt{\frac{q_1 - q_0}{a_{max}}}$ // Tiempo total de la trayectoria para a_{max}

 11 $v_{max.por.acel} = a_{max}(T_f/2)$

 12 **if** $v_{max.por.acel} \leq v_{max}$ **then**

 13 $v_{max} = v_{max.por.acel}$

 14 $\tau = T_f/2$

 15 $T = T_f$

 16 $\Delta_{tramo1} = (\tau)/n_{\Delta Tramo1}$

 17 $\Delta_{tramo2} = 0$

 18 $n_{\Delta Tramo2} = 0$

 19 $n_{total} = \sum_{i=1}^3 n_{\Delta Tramo i} + 1$

 /* Calulcar Curva LSPB */

 20 **for** i in range[0, n_{total}] **do**

 21 $tiempo_i = tiempo_i + \Delta_{tramo i}$ // $\Delta_{tramo i}$ depende de i

 22 $X = ls_v_a_puntual(q_0, q_1, v_{max}, a_{max}, tiempo_i)$ // LSPB

 /* Resultados Posición, Velocidad, Aceleración de LSPB */

 23 $(X_{rot_{\theta_y}}[i], \dot{X}_{rot_{\theta_y}}[i], \ddot{X}_{rot_{\theta_y}}[i]) = (X[0], X[1], X[2])$

 24 $tiempo[i] = tiempo_i$

25 **return** [$tiempo, X_{rot_{\theta_y}}, \dot{X}_{rot_{\theta_y}}, \ddot{X}_{rot_{\theta_y}}$]

Algorithm 14: Trayectoria (Continuacion...)

```
/* SUBFUNCIONES */  
1 ls_v_a_puntual( $q_0, q_1, v_{max}, a_{max}, tiempo_{actual}$ ):  
2    $\tau = v_{max}/a_{max};$   
3    $s = signo(q_1 - q_0);$   
4    $T = s \frac{(q_1 - q_0)}{v_{max}} + \tau;$   
   /* Ecuaciones LSPB (6.1) */  
   /* Tramo 1 */  
5   if ( $0 \leq tiempo_{actual} \leq \tau$ ) then  
6      $q_{actual} = q_0 + s \frac{a_{max}}{2} (tiempo_{actual})^2;$   
7      $v_{actual} = s a_{max} (tiempo_{actual});$   
8      $a_{actual} = s a_{max};$   
9     return [ $q_{actual}, v_{actual}, a_{actual}$ ];  
   /* Tramo 2 */  
10  else if ( $\tau < tiempo_{actual} \leq T - \tau$ ) then  
11     $q_{actual} = q_0 - (s \frac{v_{max}^2}{2a_{max}}) + (s * v_{max} * tiempo_{actual});$   
12     $v_{actual} = s * v_{max};$   
13     $a_{actual} = 0;$   
14    return [ $q_{actual}, v_{actual}, a_{actual}$ ];  
   /* Tramo 3 */  
15  else if ( $T - \tau < tiempo_{actual} \leq T$ ) then  
16     $q_{actual} = q_1 + s \left( -\frac{a_{max}T^2}{2} + a_{max}T(tiempo_{actual}) - \frac{a_{max}}{2} (tiempo_{actual})^2 \right);$   
17     $v_{actual} = s(a_{max}T - a_{max}tiempo_{actual});$   
18     $a_{actual} = s(-a_{max});$   
19    return [ $q_{actual}, v_{actual}, a_{actual}$ ];
```

6.2. Interfaz de visualización RViz

RViz da la posibilidad de visualizar los componentes mecánicos de un robot de una forma mas simple. En esta tesis se desea simular una trayectoria lineal de la plataforma móvil de un robot delta. Para cumplir con dicho objetivo se necesitan en cada momento de la trayectoria el valor de los ángulos de las juntas que unen los componentes y la posición xyz del centroide de la plataforma móvil. La posición en el espacio de los brazos y antebrazos se determinan a partir de los ángulos de cada junta relativos a otras partes mecánicas y la posición de la plataforma móvil se determina mediante las coordenadas de su centroide xyz relativo al sistema de referencia global. Por lo tanto, se configura la visualización de las partes mecánicas en 2 grupos por separado (las 3 cadenas cinemáticas y la plataforma móvil). Si la cinemática inversa es correcta, entonces la plataforma móvil debe calzar perfectamente entre las rotulas (final de los antebrazos) que unen los antebrazos con la plataforma móvil.

6.2.1. Conexión entre ROS y RViz

Básicamente, se puede obtener la visualización de la trayectoria del robot delta en 3 pasos:



Figura 6.27: Diagrama de flujo de pasos para obtener la visualización del robot delta

- **Paso 1:** Se exportan de la trayectoria creada en ROS los siguientes datos: las trayectoria en el espacio angular de los motores $\theta_{i \in \{1,2,3\}}$, la trayectoria en el espacio cartesiano del centroide de la plataforma móvil XYZ y la escala de tiempo utilizada $s(t)$. Luego se determinan los ángulos interiores θ_{2i} y θ_{3i} (vistos en la sección (4.3.2.2)) de las 3 cadenas cinemáticas. Esto es posible creando un nodo llamado `posicionador_rviz_realtime_tm1_adams.py` que espera como datos de entrada los datos de la trayectoria $\theta_{i \in \{1,2,3\}}$, XYZ y $s(t)$ y como salida un objeto de tipo `JointState()`. Este ultimo objeto contiene los datos de los ángulos interiores en radianes, ángulo de los motores en radianes y la posición xyz de la plataforma móvil en metros. Además, a cada ángulo y coordenada se le asigna un nombre único de

tipo string, los cuales son usados para relacionar las juntas configuradas para en RViz y el modelo URDF con respecto al valor numérico de cada una de ellas.

- **Paso 2:** Transformar los valores de cada junta y coordenada adquirida del paso 1 a tf. RViz tiene un nodo integrado llamado *estados_robot_pub.py* que tiene como requisito de entrada los valores de cada junta con cada nombre único respectivamente y como salida los valores tf con cada nombre único respectivamente. La configuración de los tf del robot delta es gracias al modelo URDF.
- **Paso 3:** Anterior a crear una trayectoria en ROS se debe configurar el robot delta en formato URDF. Los nombres de cada juntas y coordenadas del efecto en el archivo URDF en formato .xml son exactamente los mismo que en los pasos 1 y 2. Se inserta el modelo del robot URDF a RViz para que los valores tf calculados en el paso 2 sean usados para dar movimiento a las piezas creadas por el archivo URDF.

Los nombres de cada junta se muestran en la siguiente tabla:

Simbología	Nombre	Descripción
θ_{11}	base_brazo1	Ángulo del motor en cadena cinemática 1
θ_{12}	base_brazo2	Ángulo del motor en cadena cinemática 2
θ_{13}	base_brazo3	Ángulo del motor en cadena cinemática 3
θ_{21}	codo1_a	Ángulo interior 2 de la rotula que une el brazo con el antebrazo en cadena cinemática 1
θ_{31}	codo1_b	Ángulo interior 3 de la rotula que une el brazo con el antebrazo en cadena cinemática 1
θ_{22}	codo2_a	Ángulo interior 2 de la rotula que une el brazo con el antebrazo en cadena cinemática 2
θ_{32}	codo2_b	Ángulo interior 3 de la rotula que une el brazo con el antebrazo en cadena cinemática 2
θ_{23}	codo3_a	Ángulo interior 2 de la rotula que une el brazo con el antebrazo en cadena cinemática 3
θ_{33}	codo3_b	Ángulo interior 2 de la rotula que une el brazo con el antebrazo en cadena cinemática 3
X	act_x	Posición X del centroide de la plataforma móvil
Y	act_y	Posición Y del centroide de la plataforma móvi
Z	act_z	Posición Z del centroide de la plataforma móvi

Tabla 6.3: Nombre de juntas del robot delta en URDF

6.2.2. Modelo URDF

Para visualizar un modelo de un robot se usa un archivos de formato de descripción de robot unificado (URDF) basados en xml. En esta sección se presenta la configuración del modelo del robot delta mostrando el código para los enlaces, juntas y sus respectivas uniones padre-hijo para cada uno de ellos. La nomenclatura utilizada es la del capítulo (5). El nombre del archivo donde están contenidos todos estos códigos es robot.urdfm1.adams.urdf.

6.2.2.1. Base Fija

La base fija es un enlace que se le llama "**base_link**" y se representa como un cilindro de radio R_a .

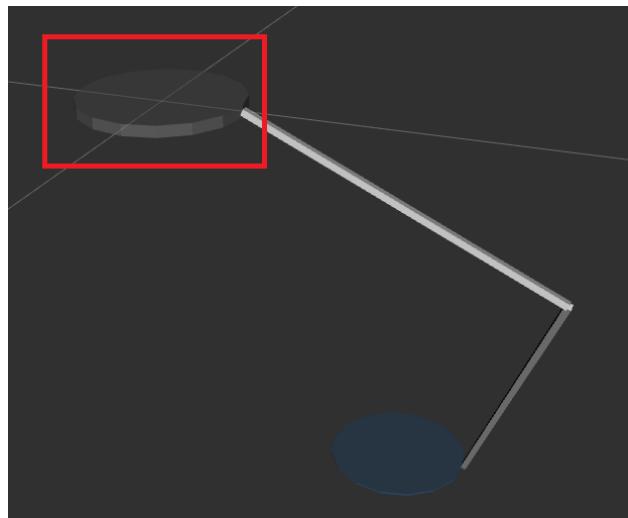


Figura 6.28: Base fija en RViz

```
<!--BASE SUPERIOR: ----- -->
<link name="base_link">
    <visual>
        <geometry>
            <cylinder length="0.005" radius="0.210"/>
        </geometry>
        <material name="gris">
            <color rgba="0.55 0.55 0.55 0.5"/>
        </material>
    </visual>
</link>
```

6.2.2.2. Brazos

Los brazos son enlaces que se les nombra "**brazo1** , **brazo2** y **brazo3**" y se representan como un paralelepípedo tipo box de largo L_a . Estos brazos se unen a la base fija a través de una junta tipo revoluta que representan los motores o actuadores del robot delta. Las juntas se les asignan los nombres "**base_brazo1** , **base_brazo2** y **base_brazo3**" para cada brazo respectivamente. En el código de las juntas se configura las conexiones padre-hijo de los enlaces.

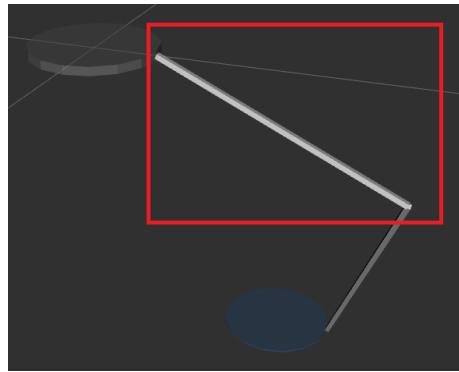


Figura 6.29: Brazo en RViz

```
<!--BRAZOS SUPERIORES: _____-->
<!--BRAZO_1-->
<link name="brazo1">
    <visual>
        <origin xyz="0.310 0 0" rpy="0 0 0"/> <!--La/2-->
        <geometry>
            <box size="0.620 0.003 0.003"/> <!--La -->
        </geometry>
        <material name="blanco">
            <color rgba="0.9 0.9 0.9 1"/>
        </material>
    </visual>
</link>
<joint name="base_brazo1" type="revolute">
    <axis xyz="0 1 0"/>
    <limit effort="100.0" lower="0.0" upper="1.8" velocity="0.5"/>
    <parent link="base_link"/>
    <child link="brazo1"/>
    <origin xyz="-0.181865335 0.105 0" rpy="0 0 2.61799"/>
</joint>
```

```

<!--BRAZO_2-->
<link name="brazo2">
    <visual>
        <origin xyz="0.310 0 0" rpy="0 0 0"/> <!--La/2-->
        <geometry>
            <box size="0.620 0.003 0.003"/> <!--La -->
        </geometry>
        <material name="blanco"/>
    </visual>
</link>

<joint name="base_brazo2" type="revolute">
    <axis xyz="0 1 0"/>
    <limit effort="100.0" lower="0.0" upper="1.8" velocity="0.5"/>
    <parent link="base_link"/>
    <child link="brazo2"/>
    <origin xyz="0 -0.210 0" rpy="0 0 -1.57075"/>
</joint>

<!--BRAZO_3-->
<link name="brazo3">
    <visual>
        <origin xyz="0.310 0 0" rpy="0 0 0"/><!--La/2-->
        <geometry>
            <box size="0.620 0.003 0.003"/> <!--La -->
        </geometry>
        <material name="blanco"/>
    </visual>
</link>
<joint name="base_brazo3" type="revolute">
    <axis xyz="0 1 0"/>
    <limit effort="100.0" lower="0.0" upper="1.8" velocity="0.5"/>
    <parent link="base_link"/>
    <child link="brazo3"/>
    <origin xyz="0.181865335 0.105 0" rpy="0 0 0.523599"/>
</joint>
```

6.2.2.3. Antebrazos

Los antebrazos son enlaces que se les nombra "**barra1_b** , **barra2_b** y **barra3_b**" y se representan como un paralelepípedo tipo box de largo L_b . Estos antebrazos se unen cada uno a su respectivo brazo través de dos juntas tipo revoluta que representan rotulas. Las juntas se les asignan los nombres '**codo1_a** , **codo1_b** , **codo2_a** , **codo2_b** , **codo3_a** , **codo3_b**' para cada brazo respectivamente. En el código de las juntas se configura las conexiones padre-hijo que determina al fin y al cabo la posición de los antebrazos respecto al brazo.

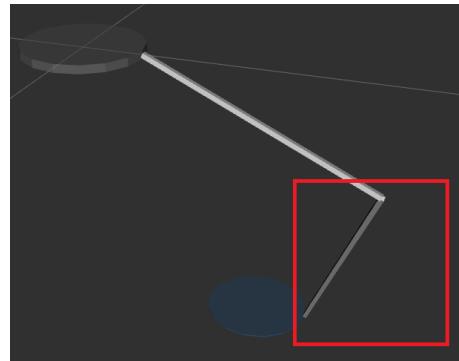


Figura 6.30: Antebrazo en RViz

```
<!--BRAZOS INFERIORES: ----->
<!--Ante_BRAZO_1-->
<link name="barra1_a"/>
<joint name="codo1_a" type="revolute">
    <axis xyz="0 -1 0"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
    <parent link="brazo1"/>
    <child link="barra1_a"/>
    <origin xyz="0.620 0 0" rpy="0 0 0"/>
</joint>
<link name="barra1_b">
    <visual>
        <origin xyz="-0.440 0 0" rpy="0 0 0"/> <!--Lb/2-->
        <geometry>
            <box size="0.880 0.003 0.003"/> <!--Lb-->
        </geometry>
        <material name="blanco"/>
    </visual>
</link>
```

```

<joint name="codo1_b" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
    <parent link="barra1_a"/>
    <child link="barra1_b"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
</joint>

<!--Ante_BRAZO_2-->
<link name="barra2_a"/>
<joint name="codo2_a" type="revolute">
    <axis xyz="0 -1 0"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
    <parent link="brazo2"/>
    <child link="barra2_a"/>
    <origin xyz="0.620 0 0" rpy="0 0 0"/> <!--la = 150-->
</joint>
<link name="barra2_b">
    <visual>
        <origin xyz="-0.440 0 0" rpy="0 0 0"/> <!--Lb-/2-->
        <geometry>
            <box size="0.880 0.003 0.003"/> <!--Lb-->
        </geometry>
        <material name="blanco"/>
    </visual>
</link>
<joint name="codo2_b" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
    <parent link="barra2_a"/>
    <child link="barra2_b"/>
    <origin xyz="0 0 0" rpy="0 0 0"/> <!--02 = 62.0652 -107.5000 0-->
</joint>

<!--Ante_BRAZO_3-->
<link name="barra3_a"/>
<joint name="codo3_a" type="revolute">
    <axis xyz="0 -1 0"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>

```

```

<parent link="brazo3"/>
<child link="barra3_a"/>
<origin xyz="0.620 0 0" rpy="0 0 0"/> <!--02 = 62.0652 -107.5000
0-->
</joint>
<link name="barra3_b">
    <visual>
        <origin xyz="-0.440 0 0" rpy="0 0 0"/><!--Lb-/2-->
        <geometry>
            <box size="0.880 0.003 0.003"/> <!--Lb-->
        </geometry>
        <material name="blanco"/>
    </visual>
</link>
<joint name="codo3_b" type="revolute">
    <axis xyz="0 0 1"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
    <parent link="barra3_a"/>
    <child link="barra3_b"/>
    <origin xyz="0 0 0" rpy="0 0 0"/> <!--02 = 62.0652 -107.5000 0-->
</joint>
```

6.2.2.4. Base Movil

La base móvil es un enlace que se le nombra '**actuador**' y se representa como un cilindro de radio R_b . La posición y orientación de este enlace no es respecto a ningún otro enlace, solo se determina en relación al sistema de referencia global. Para posicionar el centroide del cilindro, se utilizan 3 juntas tipo prisma, que representan las coordenadas x,y,z. Los nombres de estas juntas son '**act_x**', '**act_y**' y '**act_z**'-

```

<link name="actuador">
    <visual>
        <origin xyz="0 0 0" rpy="0 -1.57075 0"/>
        <geometry>
            <cylinder length="0.001" radius="0.050"/>
        </geometry>
        <material name="colorte">
            <color rgba="0.15 0.55 0.95 0.3"/>
        </material>
```

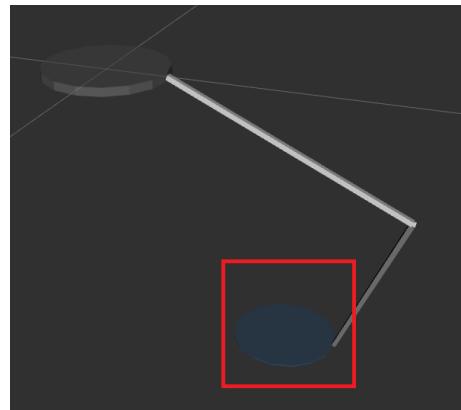


Figura 6.31: Base móvil en RViz

```
</visual>
</link>
<link name="aux1"/>
<link name="aux2"/>
<joint name="act_x" type="prismatic">
    <parent link="base_link"/>
    <child link="aux1"/>
    <limit effort="100.0" lower="-1" upper="1" velocity="0.5"/>
    <origin rpy="0 0 0" xyz="0 0 0"/>
</joint>
<joint name="act_y" type="prismatic">
    <parent link="aux1"/>
    <child link="aux2"/>
    <limit effort="100.0" lower="-1" upper="1" velocity="0.5"/>
    <origin rpy="0 0 1.57075" xyz="0 0 0"/>
</joint>
<joint name="act_z" type="prismatic">
    <parent link="aux2"/>
    <child link="actuador"/>
    <limit effort="100.0" lower="0" upper="1" velocity="0.5"/>
    <origin rpy="0 -1.57075 0" xyz="0 0 0"/>
</joint>
```

6.3. ADAMS

En esta sección se detalla el desarrollo del prototipo en ADAMS y su simulación, presentando todas las consideraciones tomadas en su elaboración, para el posterior análisis de los datos obtenidos.

El diseño de un prototipo en ADAMS puede ser tan complejo como se quiera, en la figura (6.32) se observa el proceso de desarrollo para un prototipo en ADAMS, en nuestro caso, el modelo busca validar la cinemática y dinámica del robot delta en estudio por lo que trabajaremos solo en las fases de **construcción** ("build") y **prueba**. ("test").

Virtual Prototyping Process

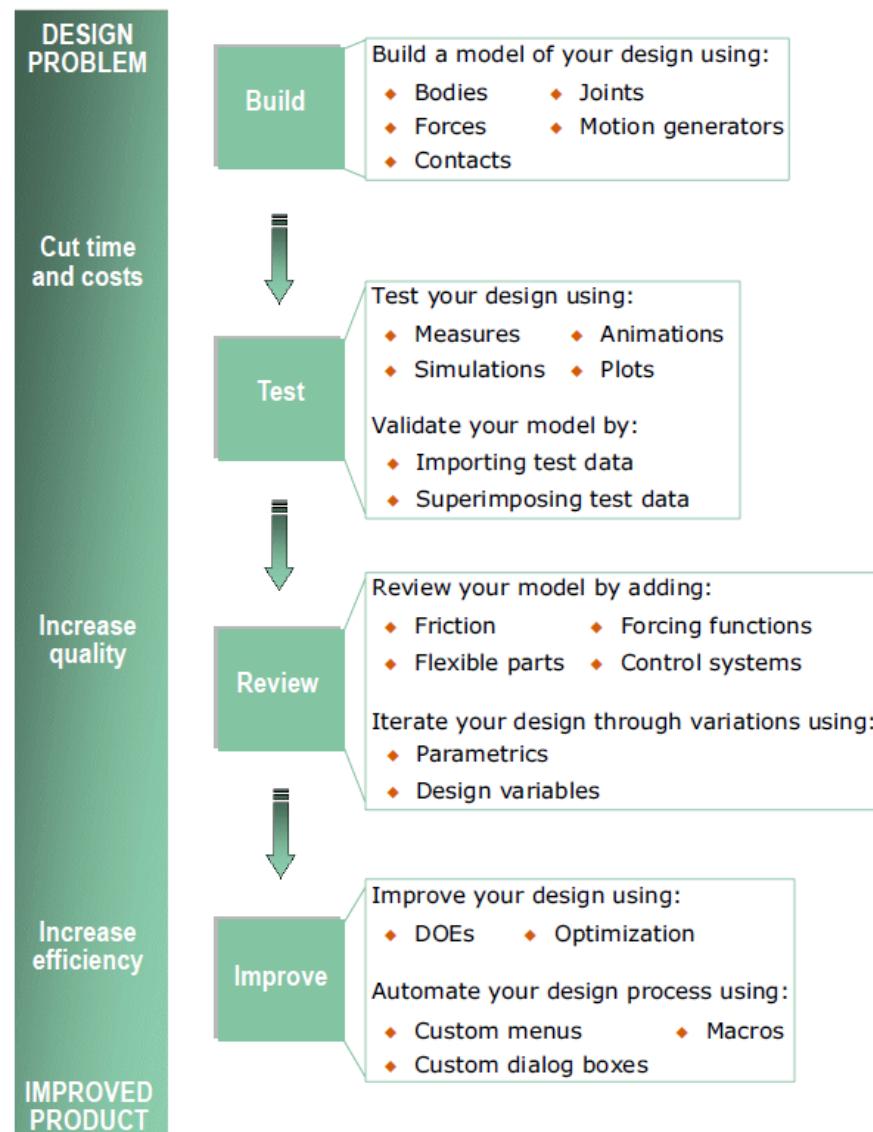


Figura 6.32: Fases de desarrollo para prototipo virtual [40]

6.3.1. Consideraciones preliminares

Para desarrollar de manera eficiente el modelo, y que sus resultados puedan ser comparados con los obtenidos mediante ROS, fue necesario establecer las siguientes simplificaciones preliminares:

- Las medidas utilizadas en cada pieza del modelo, son las establecidas en la tabla (6.1) y la nomenclatura proviene de la figura (5.1).
- La simplificación dinámica establecida en (4.4.4.3) en cuanto a la posición y distribución de las masas, es también aplicada al modelo en ADAMS, a fin de que ambos resultados puedan ser comparados.
- Se incluye a la simplificación anterior una propia del modelo en ADAMS, en la que la masa central del efecto m_p se divide en tres, agregando un tercio a cada masa $\frac{m_2}{2}$ ubicada al centro de los lados del efecto como se muestra en la figura (6.33).

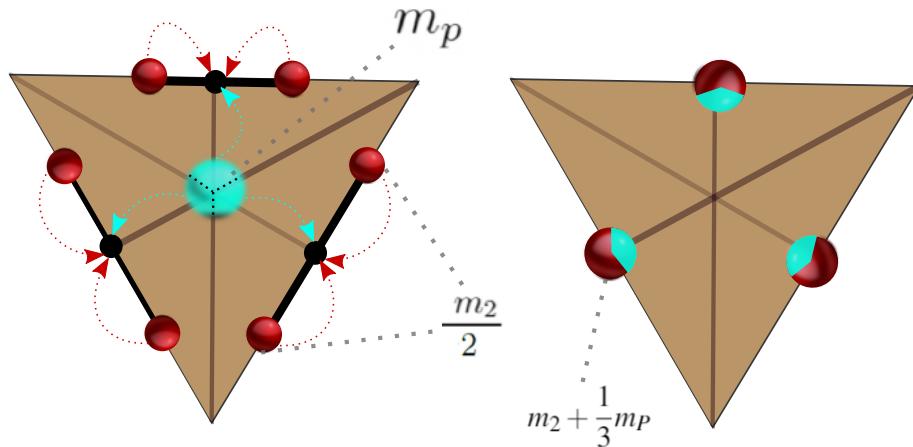


Figura 6.33: Simplificación de las masas en la base móvil

- Consideradas las simplificaciones dinámicas, el modelo del robot será el indicado en la figura (4.26)
- Existe otra simplificación propia del modelo en ADAMS, detallada en la sección de la base fija.

6.3.2. Desarrollo del modelo

El modelo se desarrollará de arriba hacia abajo, con base en la introducción a ADAMS presentada en el capítulo (3) abrimos ADAMS y generamos un nuevo modelo con la siguiente configuración:

◆ New Model

- ✓ **Model Name:** RobotDelta
- ✓ **Gravity:** Other
- ✓ **Units:** MMKS
- ✓ **Workin Directory:** A elección del usuario

◆ Gravity Settings

- ✓ **X:** 0.0
- ✓ **Y:** 0.0
- ✓ **Z:** -9806.65

Luego de terminar la configuración inicial se deberá desplegar el GUI de ADAMS, similar a la figura (3.62), con la diferencia de que el nombre del modelo sera el configurado anteriormente.

Para iniciar nuestro modelo, colocaremos un Marker unido a la tierra, que nos servirá de referencia global para todo nuestro modelo. En la barra de herramientas principal, sección de *Construction* seleccionamos *Marker* y configuramos las siguientes opciones:

◆ Geometry Marker

- ✓ Add to ground
- ✓ Global XY Plane

◆ Clic derecho en GRID

- ✓ XYZ (0.0,0.0,0.0)

Luego de generado el marker, lo buscamos en el árbol del modelo y hacemos clic derecho para renombrarlo como *Marker_Origen..*. La figura (6.34) presenta el resultado esperado, el eje coordenado mostrado en el centro, esta unido a tierra y no se moverá ni rotara con el robot. Este Marker sera la referencia global y según el se definirán los demás ejes locales.

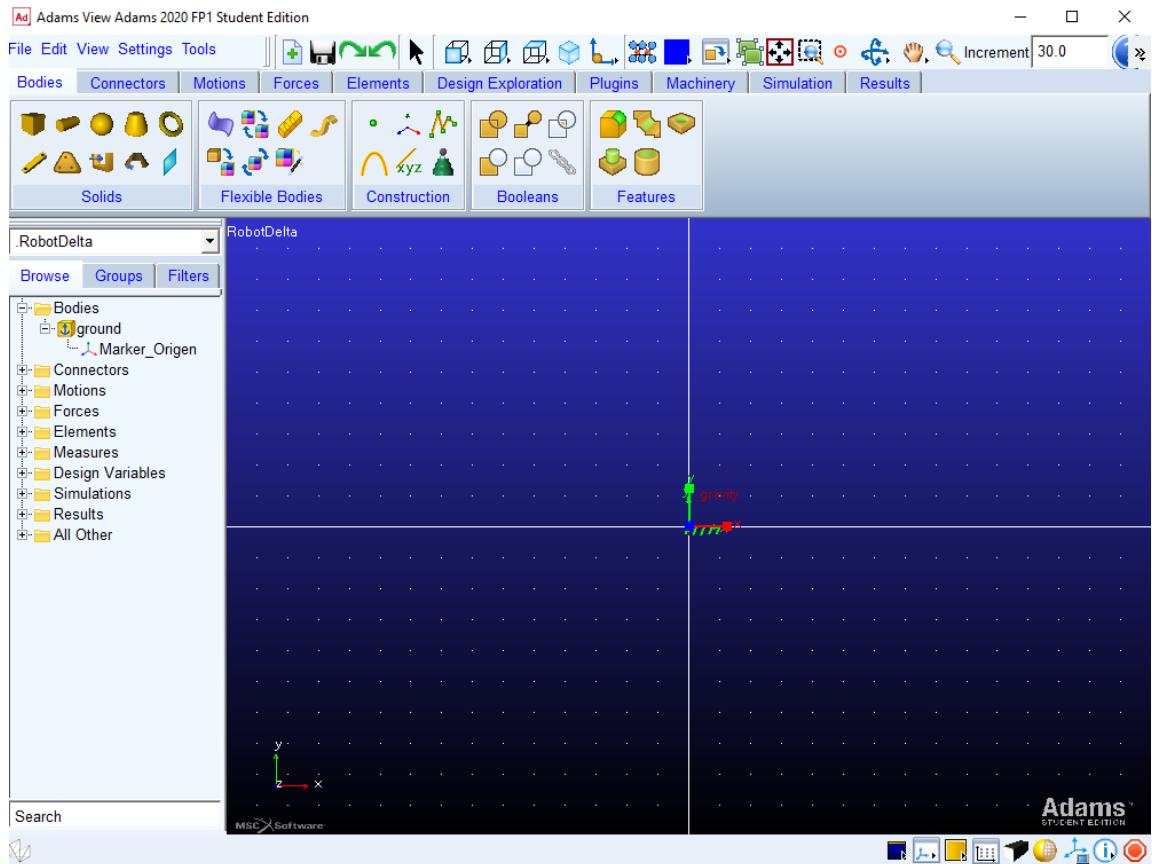


Figura 6.34: Posición del Marker Origen en ADAMS

6.3.2.1. Base fija

La base fija como su nombre lo indica, esta unida rígidamente al bastidor del robot, por lo que su masa no interviene de ninguna forma en la cinemática y dinámica del robot. Sí nos importan las posiciones de los actuadores que están unidos a la base, por lo que haremos una representación de la base fija mediante líneas sin masa, a modo de poder ubicar luego los actuadores en ella.

La posición de los puntos que generan el triángulo de la base fija se presentan en la tabla (6.4).

Actuador	X [mm]	Y [mm]	Z [mm]
Punto 1	-363.73066959	-210.0	0.0
Punto 2	363.73066959	-210.0	0.0
Punto 3	0	420	0.0

Tabla 6.4: Posición XYZ de los puntos que conforman la base fija

En el mismo apartado *Construction* de la barra de herramientas principal seleccionamos *Point* y agregamos la siguiente configuración:

- ◆ Geometry Point
 - ✓ On ground
 - ✓ Don't attach
- ◆ Click derecho en el GRID
 - ✓ XYZ(-363.73066959,-210.0,0.0)

Repetimos los pasos para los puntos 2 y 3 y finalmente los renombramos con la siguiente nomenclatura *BaseFija_Punto_N*, donde N es el numero del punto.

Una vez dibujados los tres puntos en el *GRID*, los uniremos mediante una *polilínea*, esta se ubica en el apartado *Construction*.

- ◆ Geometry PolyLine
 - ✓ On ground
 - ✓ PolyLine
- ◆ Clic izquierdo en el GRID
 - ✓ Damos clic en cada uno de los puntos creados anteriormente para unirlos con la polilínea.
 - ✓ Finalizamos la polilínea con clic derecho en el GRID.

Renombramos la polilínea como *BaseFija_Triangulo*. El resultado de los pasos anteriores, se presenta en la figura (6.35).

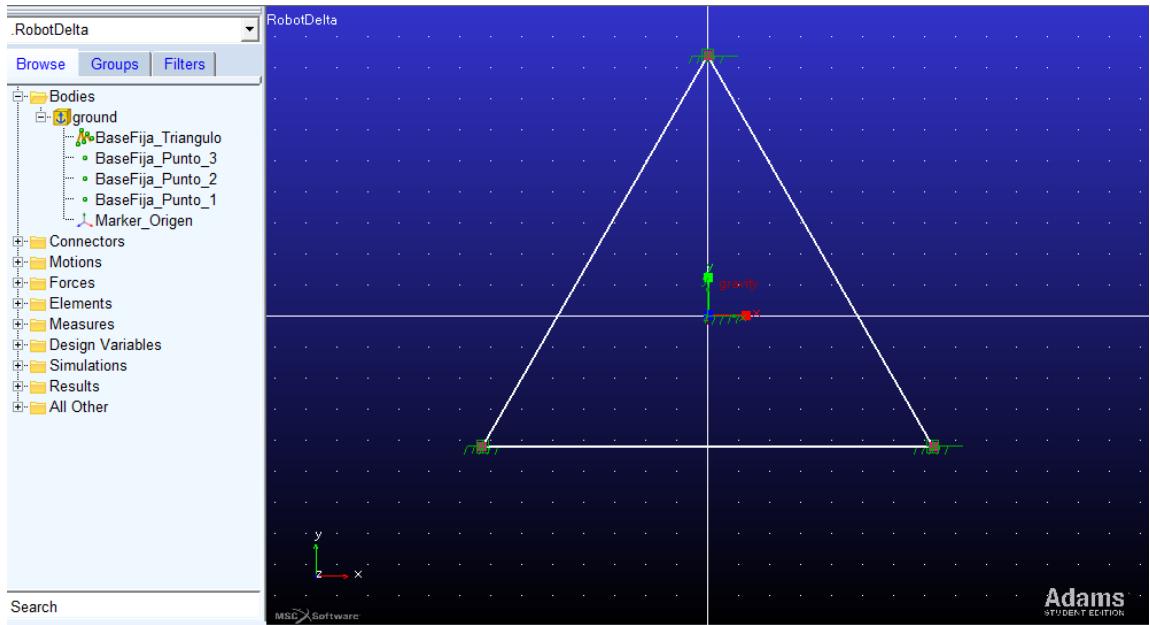


Figura 6.35: Puntos de la base fija unidos mediante polilínea

El calculo de las posiciones de los actuadores se obtiene con el radio de la base fija, obtenido de la tabla tabla (6.1), el cual simula un circulo centrado en el origen, donde los actuadores se ubican en algún punto del perímetro del círculo. Agregaremos esta circunferencia como referencia. Nuevamente en el apartado de *Construction*, seleccionamos *Arco*:

◆ Geometry Arc

- ✓ On ground
- ✓ Marcamos la opción *Radius* a 210 mm
- ✓ Marcamos la opción *Circle*

◆ Clic izquierdo en el GRID

- ✓ Damos clic en el marker central *Origen*

De la figura (5.1) se observa que el actuador numero dos (definido por el ángulo θ) se encuentra en el eje Y a una distancia igual al radio R_A . La ubicación de los otros dos actuadores sera una rotación de 120° y 240° , ya que en un robot delta los motores están idealmente igualmente espaciados.

Del álgebra lineal sabemos que una rotación en dos dimensiones queda definida por las ecuaciones (6.6) y (6.7) donde θ sera 120° , x e y la posición del actuador anterior, x' e y' la posición buscada del siguiente actuador.

$$x' = x\cos(\theta) - y\sin(\theta) \quad (6.5)$$

$$(6.6)$$

$$y' = x\sin(\theta) + y\cos(\theta) \quad (6.7)$$

La tabla (6.5) presenta las posiciones de cada actuador.

Actuador	X [mm]	Y [mm]	Z [mm]
Motor 1	-181.865335	105.0	0.0
Motor 2	181.865335	105.0	0.0
Motor 3	0	-210	0.0

Tabla 6.5: Posición XYZ de los motores

En la posición de cada motor crearemos otro marker de igual forma que lo hicimos para el *Origen* pero con las posiciones de la tabla (6.5). Renombramos cada uno de estos markers con la siguiente nomenclatura *MARKER_Motor_N*, donde N es la numeración del motor. El resultado deberá ser similar a la figura (6.36)

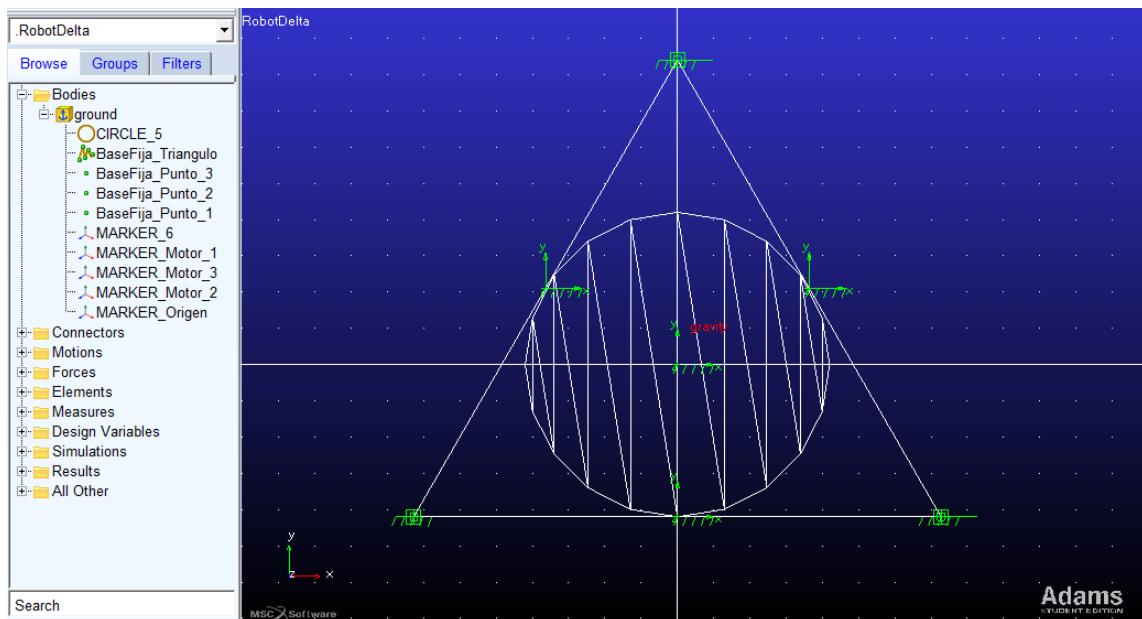


Figura 6.36: Markers que definen la posición de cada actuador ubicados en el perímetro de la circunferencia.

Luego de verificar que los actuadores se ubican justo en el perímetro de la circunfe-

rencia, podemos eliminarla, ya que no sera necesaria.

Se observa en al figura (6.36) que los markers de cada motor, quedan definidos en la misma orientación que el Marker Origen, debemos modificar dicha orientación de tal manera que el eje X de cada Marker de motor, quede perpendicular al lado del triangulo de la base fija al que pertenece, esto para que nuestro modelo quede en sincronía con la nomenclatura y direcciones asumidas en el capitulo (5). Con algo de geometría se obtiene que el giro de cada marker viene dado por los siguientes vectores:

- **Marker Motor 1:** (30, 0, 0)
- **Marker Motor 1:** (150, 0, 0)
- **Marker Motor 1:** (270, 0, 0)

Para girar los marker, damos doble clic en el nombre del marker en el árbol del modelo y configuramos las rotaciones:

- ◆ Marker Modify
- ✓ **Orientation:** (150.0, 0.0, 0.0)

Repetimos los pasos para los otros dos motores. Al finalizar los pasos nuestro modelo deberá ser similar al de la figura (6.37).

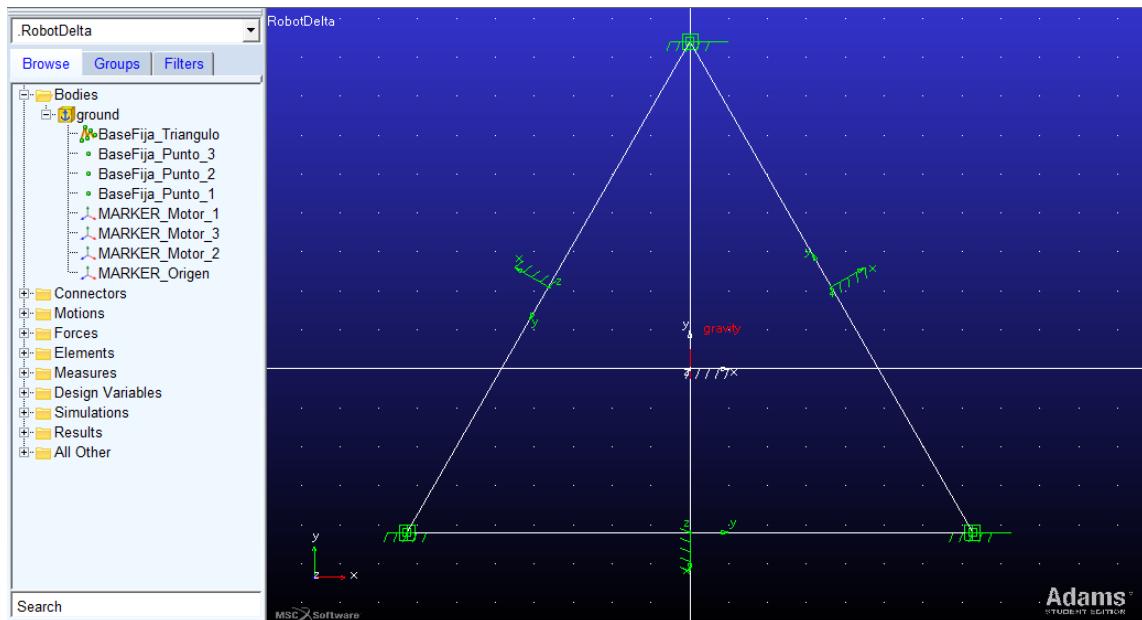


Figura 6.37: Markers de cada motor con eje X perpendicular a cada lado de la base fija.

6.3.2.2. Brazo

En base a la figura (6.32), a continuación agregaremos la masa m_1 , que simula el brazo del robot. Esta masa esta ubicada al centro del brazo, es decir a una distancia de 310[mm] desde el efecto.

Agregamos la masa del brazo numero tres (el brazo unido al motor tres), para esto en la barra de herramientas principal, en la sección de *Solids* seleccionamos *Sphere* y configuramos:

- ◆ Geometry Sphere
 - ✓ New Part
 - ✓ Seleccionamos **Radius** con un valor de 5[cm]
- ◆ Clic derecho en GRID
 - ✓ XYZ (310.0, 0.0, 0.0)
 - ✓ Rel. To Object
 - ✓ Escribimos “MARKER_Motor_3”

Se genera un nuevo cuerpo, el cual veremos en el árbol del modelo probablemente llamado *Part_2* o similar, lo renombramos a **M2_BRAZO3** y repetimos las instrucciones para los otros dos brazos, el resultado deberá ser similar a la figura (6.38)

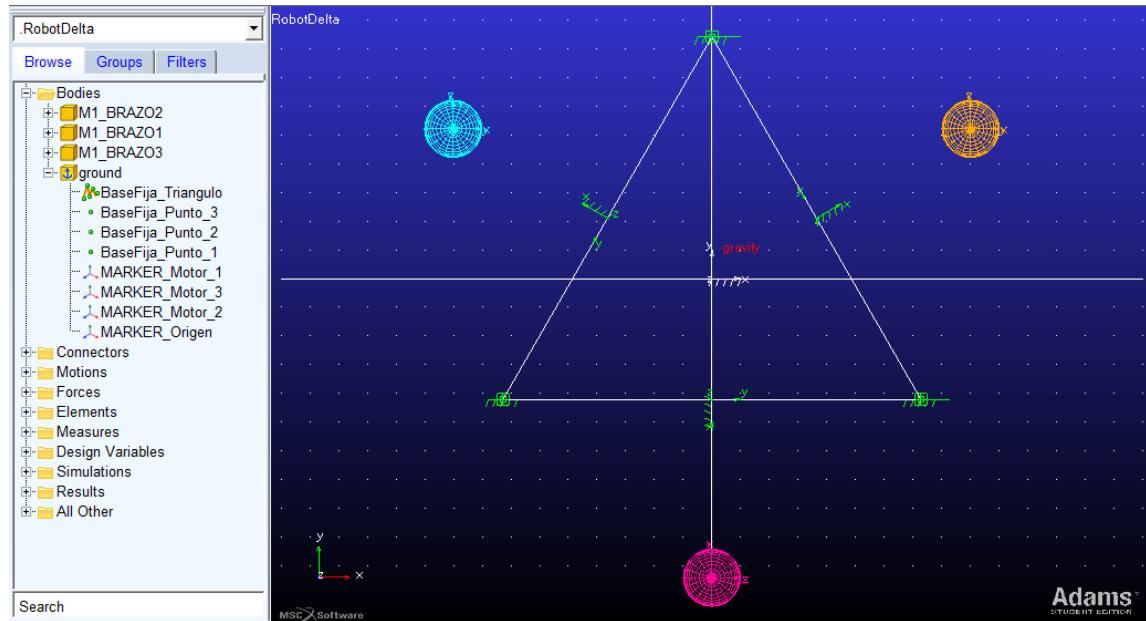


Figura 6.38: Masa m_1 de cada brazo en su posición.

Una ves dispuestas las masas que representan a los brazos, es necesario unirlas a la base fija o en nuestro caso la tierra, que seria el bastidor del robot, de forma que pueda

haber un movimiento relativo entre ellas, en específico una rotación. Este giro se logra agregando una unión de tipo *revoluta* entre el brazo y la base fija, a modo de hombro que solo permitirá el giro en el eje Y del Marker de cada motor.

Agregamos la *revoluta* al brazo numero tres dando clic en la barra de herramientas principal, en la pestaña de *Connectors*, apartado de *Joints* y buscamos la unión de *Revolute* y agregamos la siguiente configuración:

◆ Revolute Joint

- ✓ 2 Bodies - 1 Location
- ✓ Pick Geomtry Feature
- ✓ Pick Body
- ✓ Pick Body

◆ GRID

- ✓ Seleccionamos la primera pieza, en este caso la tierra (clic en cualquier parte del *GRID*, que no tenga un cuerpo)
- ✓ Seleccionamos la segunda pieza, en este caso la masa *M1_BRAZO3*
- ✓ Seleccionamos una dirección, en este caso el eje Y del **MARKER_Motor 3**

En el árbol de modelo, en la carpeta de *Connectors* ahora aparecerá nuestra unión de revoluta, renombramos a *Revoluta_Motor_3*.

Agregamos una polilínea para simular con una linea la sección del brazo que une el motor y la masa m_1 .

◆ Geometry PolyLine

- ✓ Add to Part
- ✓ PolyLine

◆ Clic izquierdo en el GRID

- ✓ Damos clic a la parte donde queremos agregar la linea, en este caso la masa m_1 del brazo tres.
- ✓ Iniciamos la polilínea dando clic sobre la masa m_1 nuevamente y otro clic en el Marker del motor 3, luego finalizamos la polilínea con clic derecho en el GRID.

Repetimos los pasos para los otros dos brazos, el resultado deberá ser similar a la figura (6.39)

Hasta ahora el comportamiento de los brazos debiera ser similar al de una masa rotando con radio fijo alrededor de la posición de cada motor.

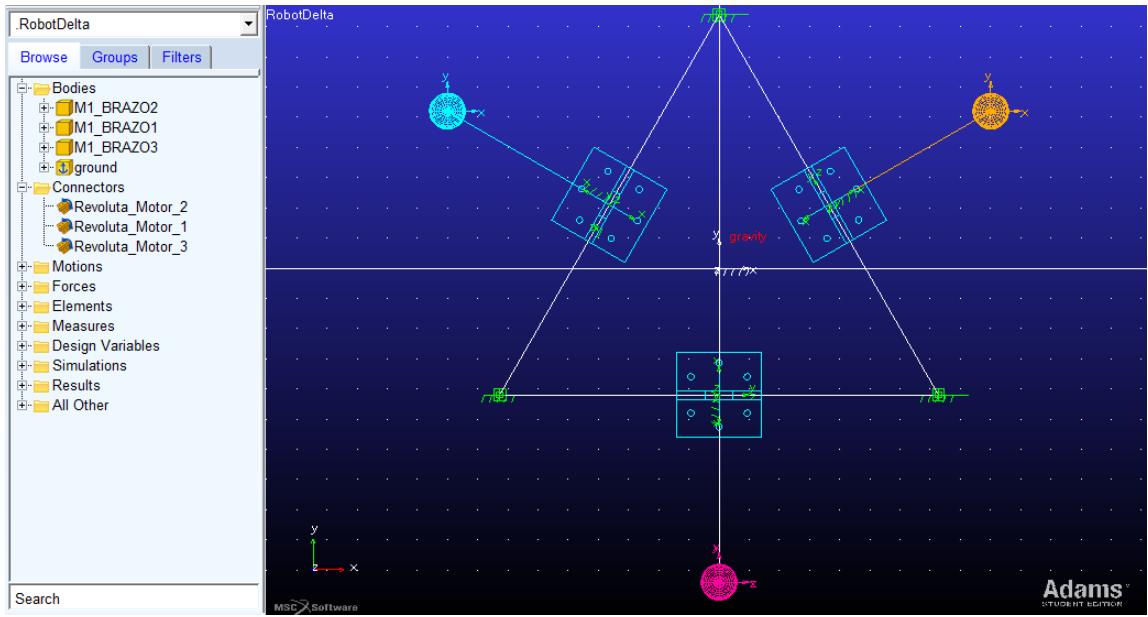


Figura 6.39: Revolutas entre cada brazo y la base fija.

6.3.2.3. Antebrazo

El antebrazo se compone de cuatro masas, dos por cada barrilla como se observa en la figura (6.32), la simplificación detallada el inicio del capítulo, simplifica las masas superiores de cada barrilla como la suma de ellas ubicada en la posición de m_{elbow} , lo mismo sucede en la parte inferior de las barillas.

En base a esa simplificación, agregamos la masa m_2 del brazo numero tres de forma similar a la m_1 , pero al doble de distancia, tomando como referencia el Marker del motor correspondiente:

◆ Geometry Sphere

- ✓ New Part
- ✓ Seleccionamos **Radius** con un valor de 5[cm]

◆ Clic derecho en GRID

- ✓ XYZ (620.0, 0.0, 0.0)
- ✓ Rel. To Object
- ✓ Escribimos “MARKER_Motor_3”

La relación de posición entre m_2 y m_1 deberá ser una unión rígida, ya que representa el brazo del robot. Para agregar una unión rígida, en la pestaña de *Connectors*, apartado de *Joints* buscamos la unión *Rigid* y agregamos la siguiente configuración:

◆ Fixed Joint

- ✓ 2 Bodies - 1 Location
- ✓ Normal to Grid
- ✓ Pick Body
- ✓ Pick Body

◆ GRID

- ✓ Seleccionamos la primera pieza, en este caso la masa m_1
- ✓ Seleccionamos la segunda pieza, en este caso la masa m_2
- ✓ Seleccionamos una posición, en este caso la masa m_1

Renombramos la masa como *M2_BRAZO3* y la unión como *Rigid_Brazo3*. Además agregamos una polilínea de manera similar a las anteriores, que unan la masa m_1 con m_2 .

Repetimos los pasos para los otros dos brazos, el resultado deberá ser similar a la figura (6.40)

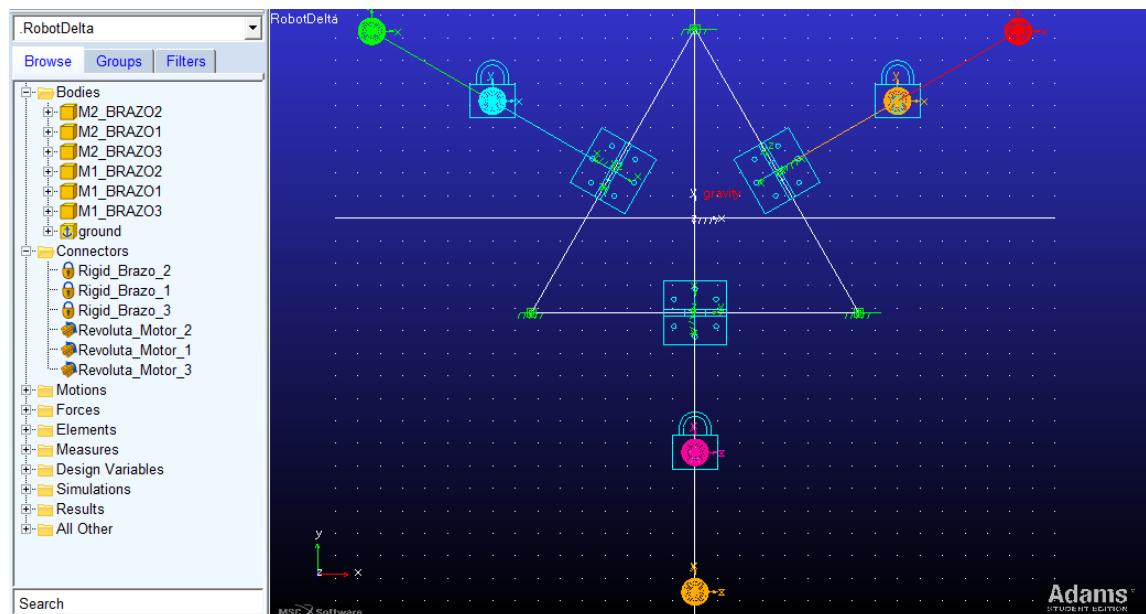


Figura 6.40: Posición de las masas m_2 .

6.3.2.4. Base móvil

La base móvil se compondrá de tres masas posicionadas a cada lado del triángulo como se observa en la figura (4.2). Para ubicar estas masas es necesario aplicar las

ecuaciones de cinemática directa descritas en el capítulo (4). Nuestro modelo se encuentra en la posición inicial del robot, es decir los ángulos θ de los efectores están en su posición inicial (0,0,0), ayudado de la ecuación (4.3) obtenemos que la posición del centro del efecto es (0.0, 0.0, -407.4309757). En dicha posición creamos un marker que nos servirá de referencia para localizar las distintas masas del efecto y lo renombramos como *MARKER_BaseFija_Origen*:

- ◆ Geometry Marker
 - ✓ Add to ground
 - ✓ Global XY Plane
- ◆ Clic derecho en GRID
 - ✓ XYZ (0.0,0.0,-407.4309757)

De forma similar a como agregamos los Marker de cada motor, agregamos una circunferencia de referencia con radio R_b según la tabla (6.1). El resultado deberá ser similar al a figura (6.41)

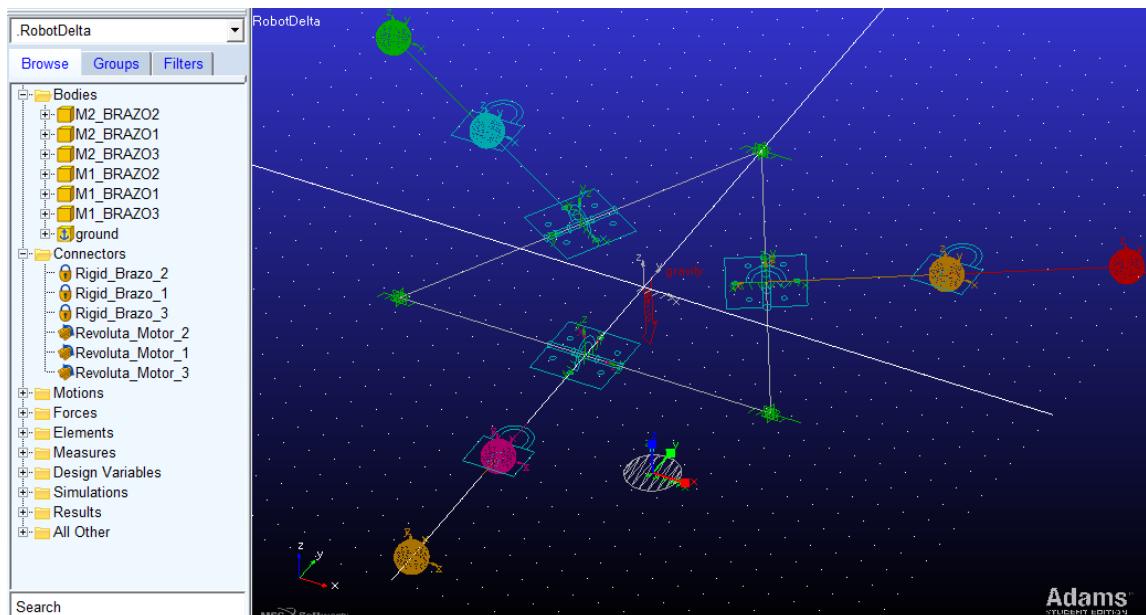


Figura 6.41: Posición del Marker central de la Base Móvil, con su respectiva circunferencia de referencia.

Dado el radio de la circunferencia sabemos que las masas del efecto, deberán estar ubicadas en algún lugar de su perímetro, similar a como lo hicimos que con los Marker de los motores, partiremos ubicando la masa unida al brazo dos, que se localiza encima del eje Y, para luego ubicar las otras dos masas, con rotaciones de 120° mediante la ecuación (??):

◆ Geometry Sphere

- ✓ New Part
- ✓ Seleccionamos **Radius** con un valor de 5[cm]
- ◆ Clic derecho en GRID
- ✓ XYZ (0.0, -50, 0.0)
- ✓ Rel. To Object
- ✓ Escribimos “MARKER_BaseFija_Origen”

Renombramos la masa como *MP_BRAZO2* y la unimos a la masa *M2_BRAZO2* mediante una polilínea.

Repetimos los pasos anteriores, haciendo giros de 120° desde la masa *MP_BRAZO2* para ubicar las masas de los otros dos brazos. Finalmente eliminamos la circunferencia de referencia y agregamos una polilínea que une las tres masas del efecto para simular el triángulo de la base fija, el resultado deberá ser similar a la figura (6.42)

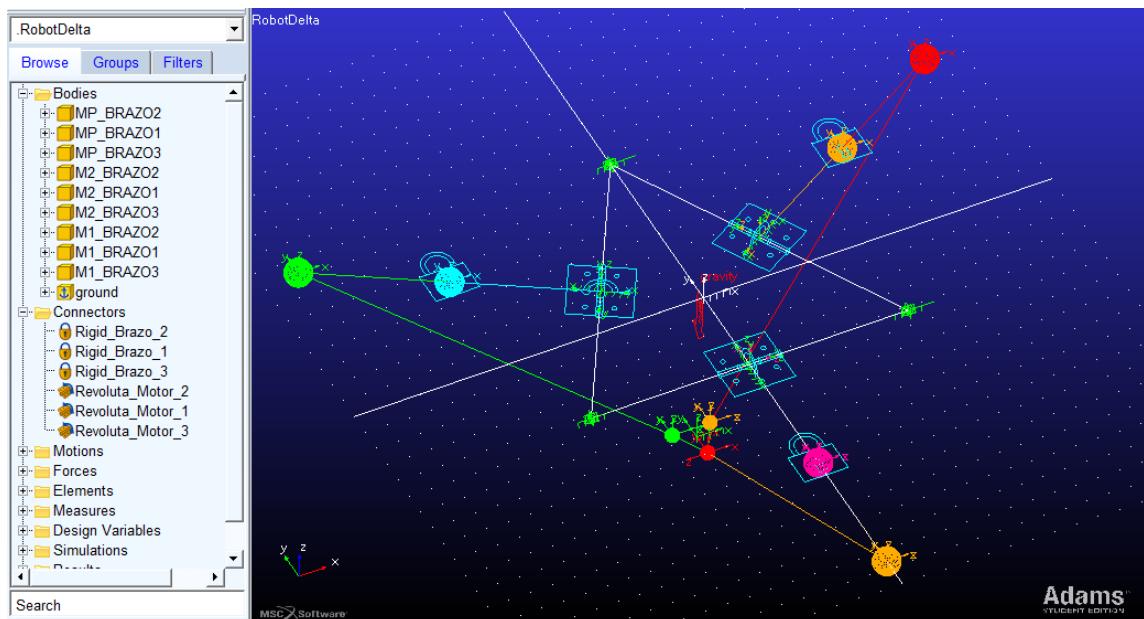


Figura 6.42: Posición de las masas que configuran la Base Móvil

Debemos unir ahora las masas *MP_BRAZO3* y *M2_BRAZO3* con una rotula que simule el movimiento del robot en dicha articulación, para ello en la barra de herramientas principal, en la pestaña de *Connectors*, apartado de *Joints* y buscamos la unión de *Spherical* y agregamos la siguiente configuración:

◆ Spherical Joint

- ✓ 2 Bodies - 1 Location
- ✓ Normal to Grid
- ✓ Pick Body
- ✓ Pick Body

◆ GRID

- ✓ Seleccionamos la primera pieza, en este caso la masa m_2
- ✓ Seleccionamos la segunda pieza, en este caso la masa m_p
- ✓ Seleccionamos una posición, en este caso la masa m_2

Renombramos la unión como *Spherical_Brazo_3* y repetimos los pasos para los otros dos brazos.

Para finalizar el modelo, debemos agregar alguna restricción de movimiento a las masas m_p que simulan la base móvil, para ello lo lógico sería agregar una unión rígida similar a la ya ocupada, dado que el triángulo de la base móvil es un sólido rígido. A pesar de ello surge un problema al aplicar dicha lógica en ADAMS dado que cuando establezcamos dicha unión rígida, las tres cadenas cinemáticas separadas de cada brazo del robot, se unirán para formar una cadena cinemática cerrada, la cual tiene una solución única para la posición de cada pieza, dicha posición a pesar de la cantidad de decimales ocupados en el desarrollo del modelo, no es posible lograrla, por lo que se produce un error, de que el modelo no tiene solución. Para solucionar dicho problema, se utilizaran como unión, rotulas esféricas que permiten variaciones de posición entre las tres masas, esto provocará alguna diferencia en los resultados otorgados por ADAMS y ROS, pero deberán ser de muy baja embreadura, ya que hemos trabajado con varios decimales para lograr la mayor precisión.

Agregamos la rotula de manera similar a la anterior, en el siguiente orden:

1. MP_BRAZO1 con MP_BRAZO2
2. MP_BRAZO2 con MP_BRAZO3
3. MP_BRAZO3 con MP_BRAZO1

Renombramos las rotulas a *Spherical_MP_N* donde N es el numero del brazo donde esta unido la rotula. Las figuras (6.43), (6.44), (6.45) presentan un resumen del modelo.

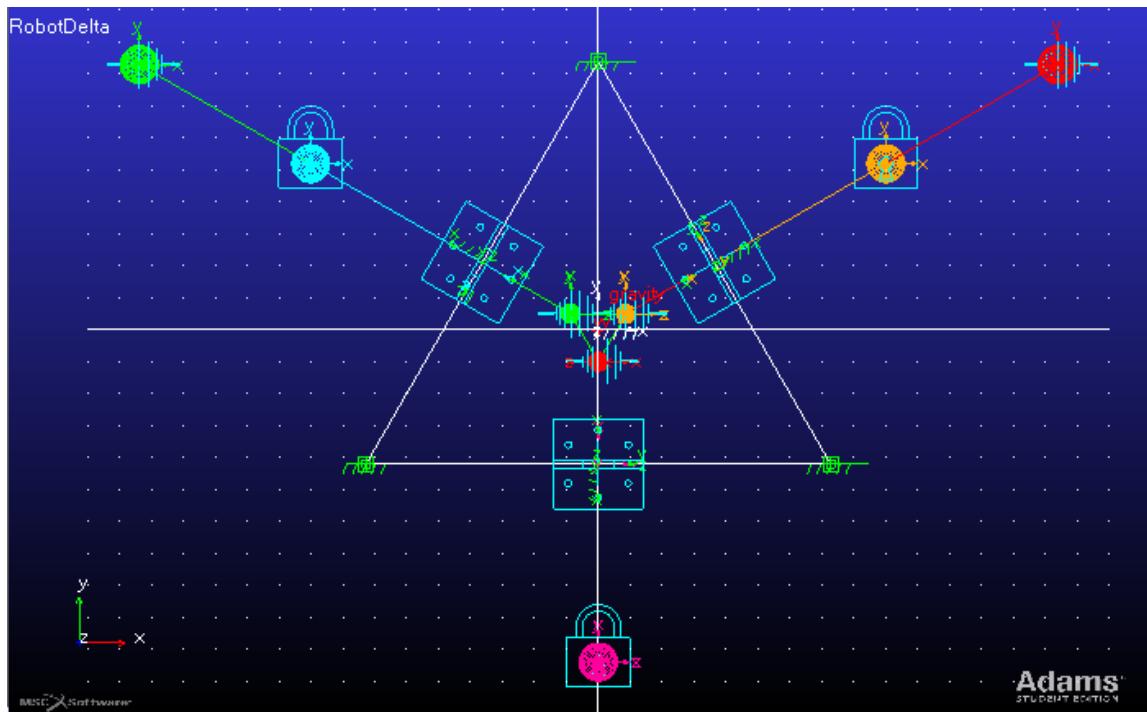


Figura 6.43: Vista superior del modelo

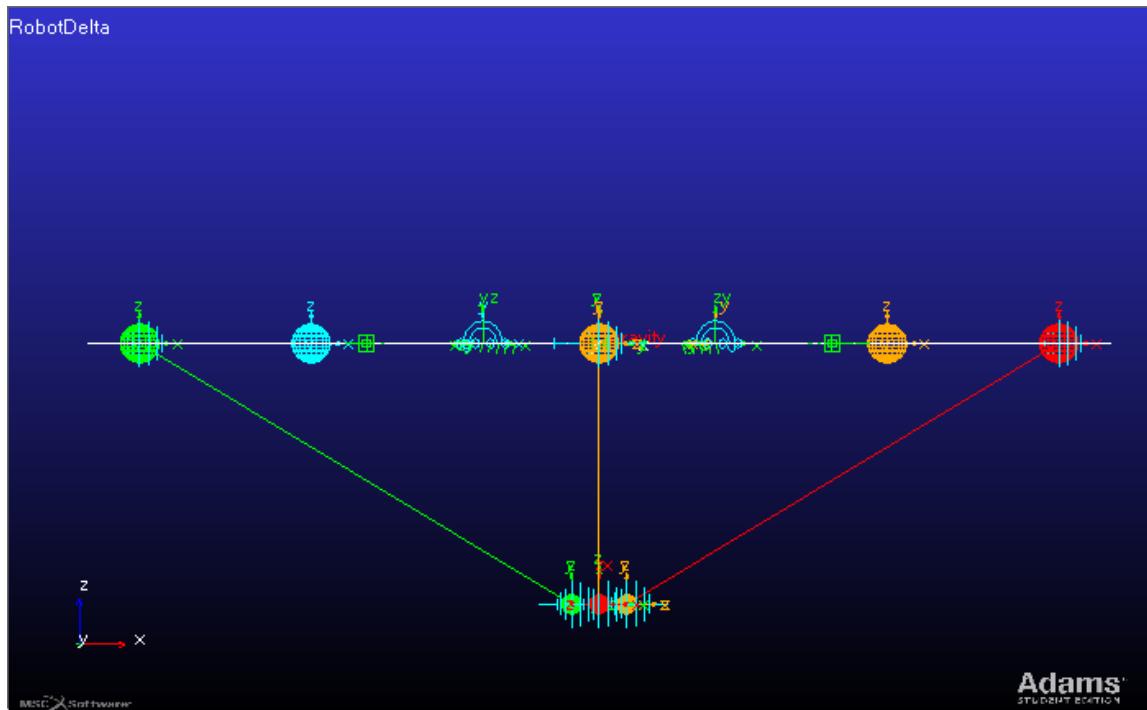


Figura 6.44: Vista frontal del modelo

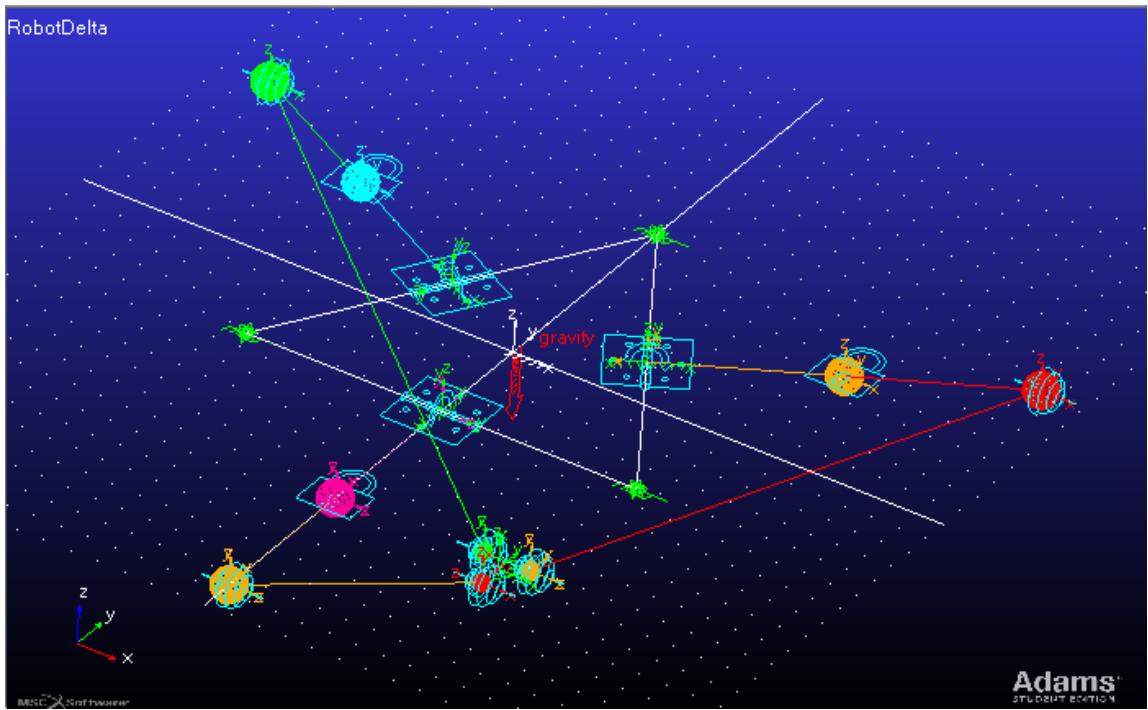


Figura 6.45: Vista isométrica del modelo

6.3.2.5. Trayectorias

Agregaremos ahora los datos XYZ de la trayectoria de la curva 1 (6.1) para dibujarla en nuestro modelo, con el fin de poder observar luego en la simulación como el centro de la base móvil (efector) pasa por los puntos de la curva.

Para crear una trayectoria en ADAMS primero se debe agregar la matriz con los datos, en la barra de herramientas principal debemos dirigirnos a la pestaña de *Elements*, sección de *Data Elements* y seleccionar *Matriz*:

- ◆ Create Matrix
 - ✓ **Row Count:** 251
 - ✓ **Column Count:** 3
 - ✓ **Values:** Insertar los datos XYZ (posición del efector) obtenidos de ROS, todo en una columna partiendo por X luego Y luego Z.

Renombramos la matriz a *MATRIX_CURVA_1*.

Luego de creada la matriz, esta se debe relacionar con una curva, en al misma sección de *Data Elements* seleccionamos *Curve*:

- ◆ Create Curve

- ✓ **Matrix Name:** MATRIX_CURVA_1.

Renombramos la curva a CURVA_1. Hasta ahora solo hemos agregado elementos de datos, los cuales ayudan a integrar datos externos a ADAMS, pero no crean ninguna geometría aun en el espacio de trabajo, para dibujar la curva en este, en la barra de herramientas principal pestaña de *Bodies*, sección de *Construction* y seleccionamos *Spline*:

◆ Geometry: Spline

- ✓ On Ground
- ✓ Deseleccionamos *Closed*
- ✓ Points

◆ GRID

- ✓ Clic izquierdo en el GRID hasta formar cuatro puntos cualesquiera, para finalizar clic derecho.

Obtendremos una curva Spline en algún lugar de nuestro espacio de trabajo, para relacionarla con la curva y matriz anteriores, le daremos doble clic a la Spline en el árbol del modelo (ground) y configuramos sus datos:

◆ Modify a Geometry Spline

- ✓ **Name:** Spline_Curva_1
- ✓ **Reference Marker:** MARKER_Origen
- ✓ **Reference Curve :** CURVA_1
- ✓ **Reference Matrix :** MATRIX_CURVA_1

El resultado deberá ser similar a la figura (6.46) la linea inferior sera la curva que deberá seguir el efecto de principio a fin durante la simulación.

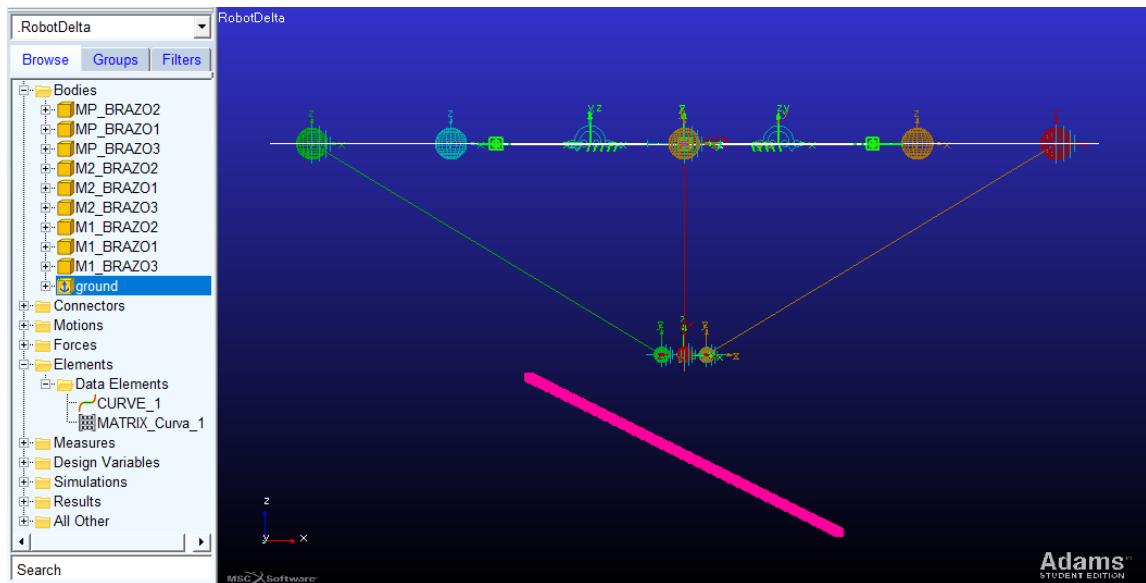


Figura 6.46: Modelo robot delta, con trayectoria a seguir en la parte inferior

6.3.3. Desarrollo de la simulación

Hasta ahora nuestro modelo comprende de la estructura base del robot, con las simplificaciones del caso y una trayectoria a seguir, para lograr desarrollar una simulación cinemática y dinámica y poder comprar los resultados, hará falta agregarle masas a las piezas y movimientos a las uniones.

6.3.3.1. Masas

Basado en las simplificaciones del modelo, el valor de cada masas es:

- $m_1 = 2.213 \text{ [kg]}$
- $m_2 = 0.6575 \text{ [kg]}$
- $m_p = 0.8275 \text{ [kg]}$

Para agregar dichas masas al modelo, basta hacer doble clic en cada una de las piezas, por ejemplo damos doble clic en *M1_BRAZO3* y agregamos la siguiente configuración:

- ◆ Modify Body
- ✓ **Mass:** (2.213kg)
- ✓ **Ixx:** 0.0

✓ **Iyy**: 0.0

✓ **Izz**: 0.0

Repetimos los pasos con cada una de las piezas, dándoles la masa que les corresponde.

6.3.3.2. Movimiento

Dado que el movimiento que se quiere generar es el del efecto pasando por la trayectoria anteriormente dibujada, el input necesario para nuestro modelo, son los ángulos de los actuadores, los cuales mediante cinemática directa, deberían transformarse en los distintos puntos XYZ de los que se compone la trayectoria.

Generaremos una *Spline* con los datos de ángulos y tiempos para la curva 1, para ello en la pestaña de *Elements* en la sección de *Data Elements*, seleccionamos *Spline*, se abrirá una ventana similar a la figura (6.47) , en donde Y serán los ángulos θ de la trayectoria e X los tiempos t . Agregamos los datos de las posiciones angulares del motor y sus tiempos.

	X	Y
1	0.0	0.492958044
2	1.0E-03	0.492920281
3	2.0E-03	0.492806988
4	3.0E-03	0.492618149
5	4.0E-03	0.492353742
6	5.0E-03	0.492013735
7	6.0E-03	0.491598083
8	7.0E-03	0.491106738
9	8.0E-03	0.490539636
10	9.0E-03	0.48989671
11	1.0E-02	0.489177881
12	1.1E-02	0.488383061
13	1.2E-02	0.487512154
14	1.3E-02	0.486565056
15	1.4E-02	0.485541654
16	1.5E-02	0.484441828
17	1.6E-02	0.483265447

Figura 6.47: Ventana para agregar las posiciones angulares en función del tiempo

Repetimos los pasos hasta generar las tres *Spline* correspondientes ala trayectoria de cada motor. Renombramos las *Spline* a THETA_N, donde N es la numeración de cada

motor. Debemos ser cuidadosos en agregar los ángulos *theta* correspondientes a cada motor en su *Spline*.

Agregamos ahora el movimiento de cada motor, iniciamos con el brazo numero tres, en la pestaña de *Motions* sección de *Joint Motions*, seleccionamos *Rotational*:

◆ Rotational Joint Motion

- ✓ Seleccionamos la unión de revoluta del brazo tres (Revoluta_Motor_3)

En el árbol del modelo, en al carpeta de *Motions* aparecerá un nuevo movimiento llamado *Motion_1*, cambiamos el numero para que coincida con la numeración del motor. Repetimos los pasos para las otras dos uniones. El resultado deberá ser similar a la figura (6.48).

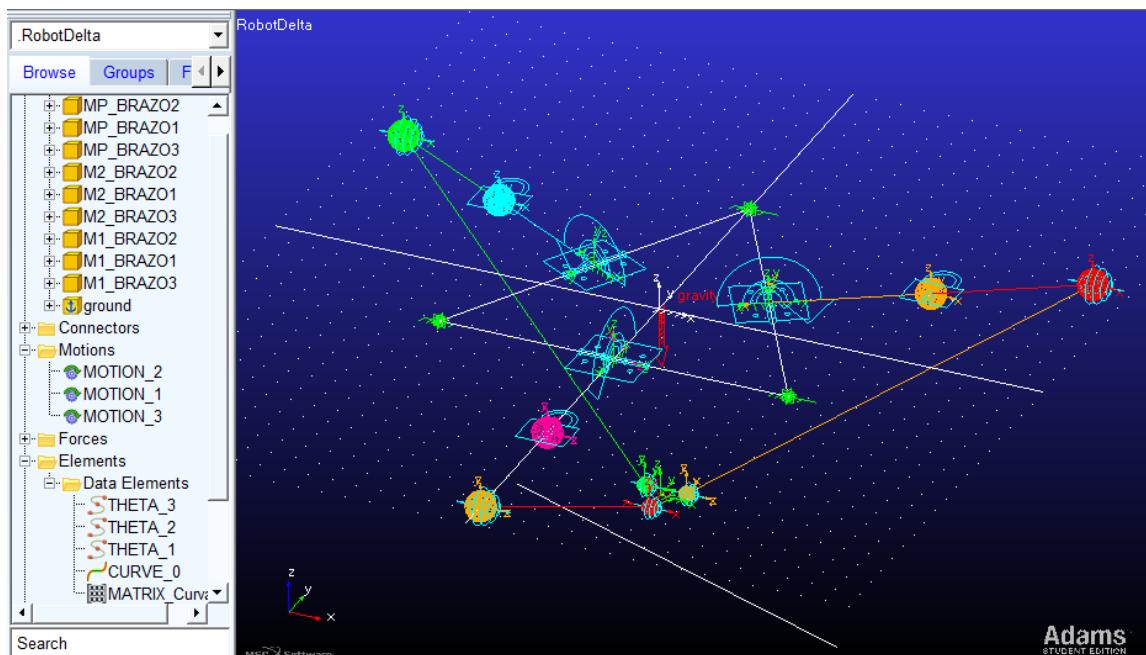


Figura 6.48: *Motions* en cada unión de revoluta

Finalizado este paso, nuestro modelo ya tendrá la trayectoria angular en cada una de sus uniones de revoluta, que simular los motores del motor. Con esta trayectoria angular, ya podemos reproducir una simulación sencilla sin ninguna medición y comprobar si la base móvil efectivamente desplaza por la curva que generamos.

Para agregar una simulación en ADAMS nos dirigimos a la barra de herramientas principal, pestaña de *Simulation* sección de *Simulate* y damos clic a *Interactive Simulation* y damos la siguiente configuración:

◆ Simulation Control

- ✓ **End Time:** 0.39 (El tiempo se obtiene de los datos angulares vs tiempo, el ultimo de ellos sera el tiempo final)
- ✓ **Step Size:** 1.0E-03 (Un valor estimado para lograr apreciar el movimiento del modelo).
- ✓ **Sim Type:** Dynamic

Al hacer clic en play, correrá la simulación y debemos observar como el efecto circula por la trayectoria anteriormente dibujada, en las figuras (6.49) y (6.50) se muestran el inicio y final de la simulación.

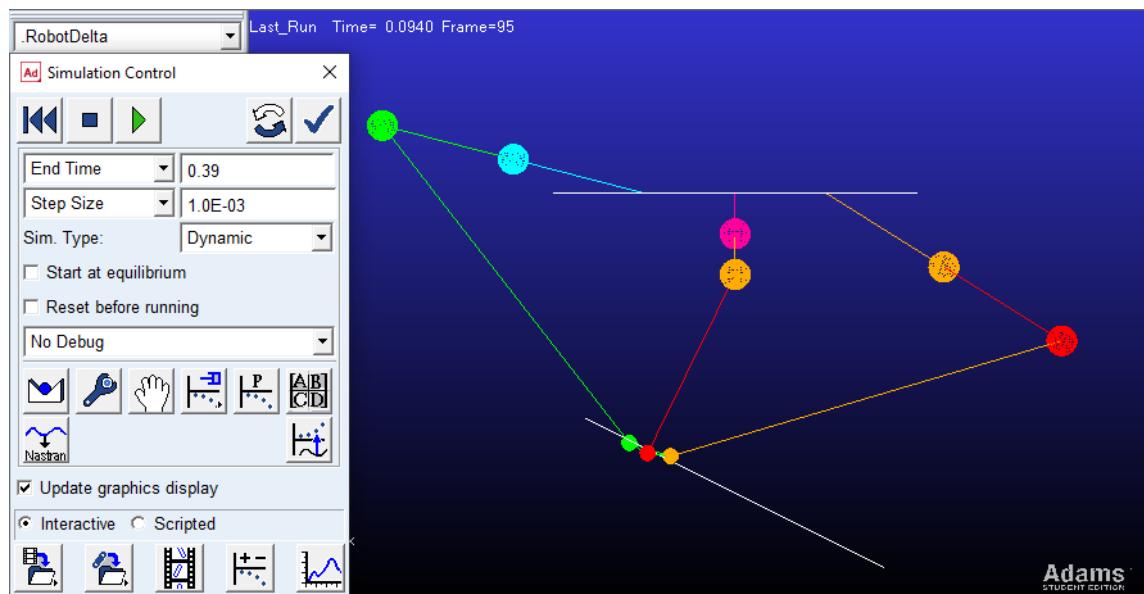


Figura 6.49: Movimiento Robot Delta, inicio de la simulación

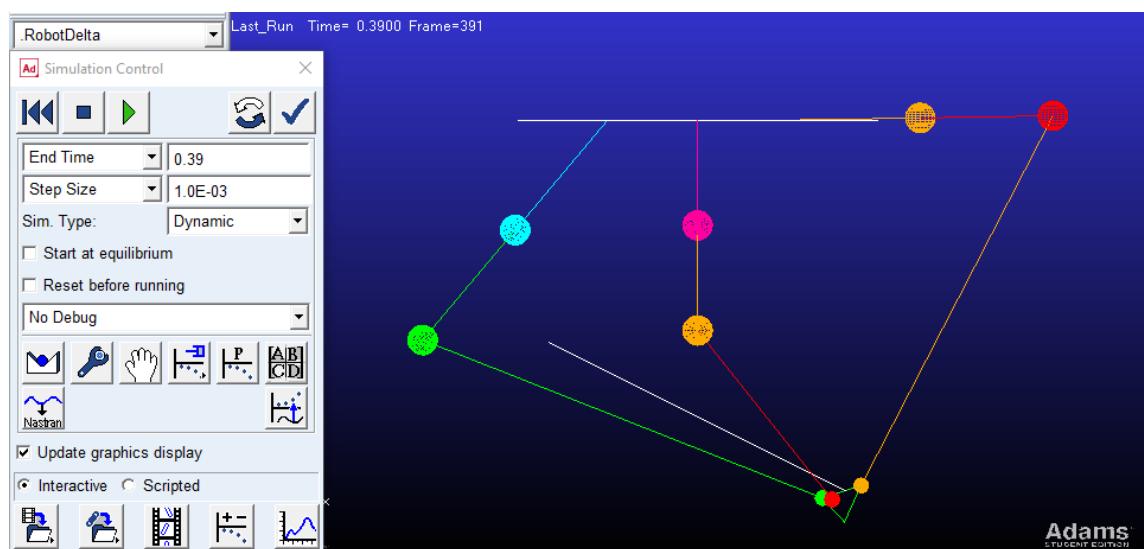


Figura 6.50: Movimiento Robot Delta, final de la simulación

6.3.3.3. Mediciones

Con la simulación anterior, ya tenemos una primera aproximación al resultado, pero debemos generar datos y mediciones que se puedan comparar con las obtenidas por ROS para validar estas ultimas. Nos interesa en específico los torques generados en las uniones de revoluta que simulan los torques de cada motor del robot.

Para generar mediciones, en la barra de herramientas principal en la pestaña de *Design Evaluation*, sección de *Measures* seleccionamos *Function Measure*:

- ◆ Create or Modify a Function Measure

- ✓ En el recuadro escribimos la siguiente función:
`MOTION(.RobotDelta.MOTION_N, 0, 8,MARKER_xx).`
Donde N es la numeración del motor que se medirá y xx es el Marker generado al crear el *Motion* de cada motor.
- ✓ Measure Name: `.RobotDelta.Torque_Motor_1`.
- ✓ General Attributes
 - **Units:** Torque

Creamos una medición de forma similar para cada *Motion* y corremos nuevamente la simulación, el resultado de las gráficas de torque obtenidas se presenta en las figuras (6.51), (6.52), (6.53).

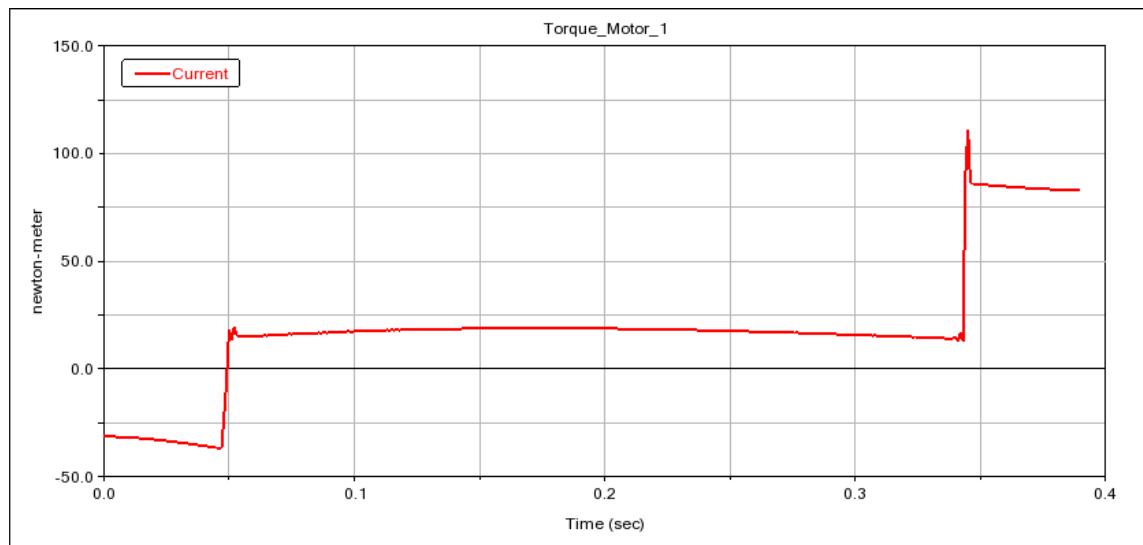


Figura 6.51: Torque obtenido para el motor 1, en la curva numero 1

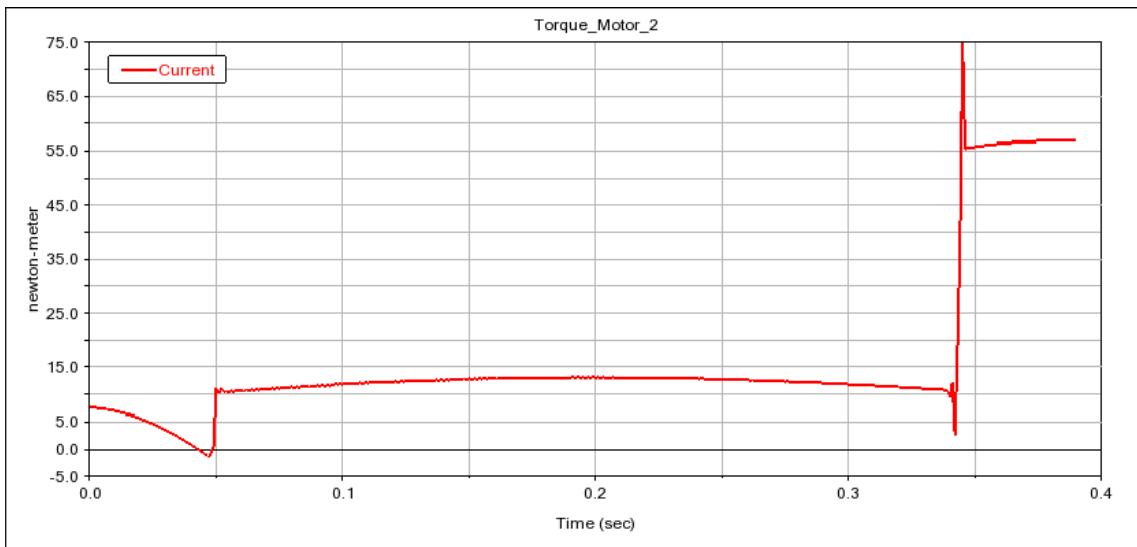


Figura 6.52: Torque obtenido para el motor 2, en la curva numero 1

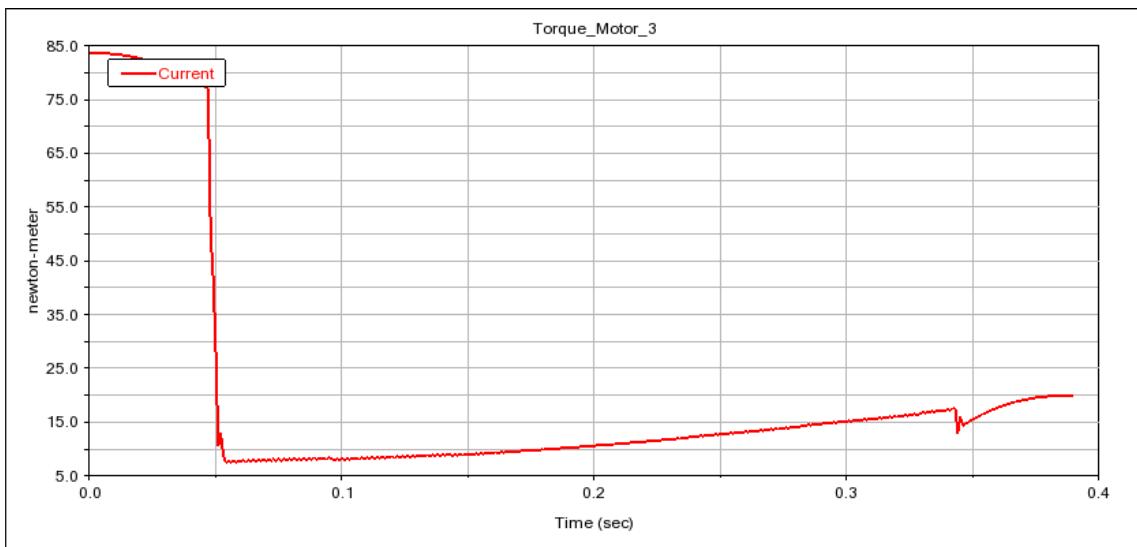


Figura 6.53: Torque obtenido para el motor 2, en la curva numero 1

Los datos representados por estas mediciones es posible exportarlos para compararlos con los obtenidos por ROS, dicha comparación se detalla en el capítulo [7](#).

Capítulo 7

Resultados

7.1. Visualizador

En esta sección se presentan los resultados de la visualización del robot delta en RViz. Primero se muestra la explicación gráfica de la conexión de los nodos y temas creados en ROS. Luego se muestran los enlaces y los marcos referencia de cada enlace en tf.

7.1.1. Temas y Nodo

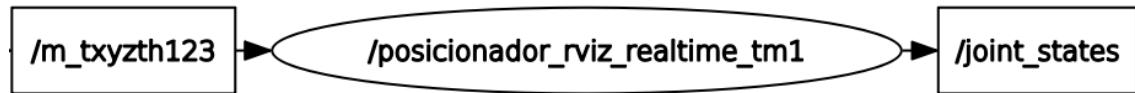


Figura 7.1: Temas y nodos en ROS para la visualización del robot delta.

- **Subscriber:** El tema *m_txxyzth123* está configurado por un mensaje escrito en el archivo llamado *matriz_path_ls.msg*, compuesto por los datos de la tabla (7.1). Las matrices *x,y,z* son los puntos de la trayectoria lineal en el espacio cartesiano de la base móvil. Las matrices *th1,th2,th3* son la trayectoria en el espacio articular de los actuadores. La matriz de *tiempo* es la escala de tiempo de la trayectoria a simular.
- **Nodo:** El nodo con el nombre *posicionador_rviz_realtime_tm1* es el encargado de calcular los ángulos de las articulación o juntas del robot delta a partir de las coordenadas en el espacio cartesiano xyz de la trayectoria de la base móvil. Los puntos *xyz* son utilizados para simular el movimiento la base móvil y los ángulos de cada junta para el movimiento de las 3 cadenas cinemáticas.

- **Publisher:** El tema *joint_states* esta configurado por un mensaje escrito en el archivo llamado *JointState.msg*, compuesto por los datos mostrados en la tabla (7.2). Este es un mensaje que contiene datos para describir el estado de un conjunto de juntas controladas por torque. Cada articulación (revoluta o prismática) se identifica de forma única por su nombre. *Name* es el nombre de cada articulación, *position* es la posición de la articulación (rad o m), *velocity* es la velocidad de la articulación (rad/s o m/s) y *effort* es el esfuerzo que se aplica en la articulación (Nm o N). En esta tesis solo se utilizan las matrices de los nombres y posición de cada junta.

Tipo de dato	Nombre
bool	permiso
int64	id_call
float32[]	x
float32[]	y
float32[]	z
float32[]	th1
float32[]	th2
float32[]	th3
float32[]	tiempo

Tabla 7.1: Mensaje matriz_path_ls.msg utilizado por tema m_txyzth12.

Tipo de dato	Nombre
string[]	name
float64[]	position
float64[]	velocity
float64[]	effort

Tabla 7.2: Mensaje joint_state.msg utilizado por tema joint_state.

7.1.2. Enlaces y Juntas

La figura (7.2) representa la visualización de los enlaces del robot delta. La base fija es un cilindro de color gris, la base móvil es un cilindro de color azul y tanto los brazos como los antebrazos son cajas alargadas de color blanco. Además, se muestra los nombres de cada enlace con sus marcos de referencia tf respectivos.

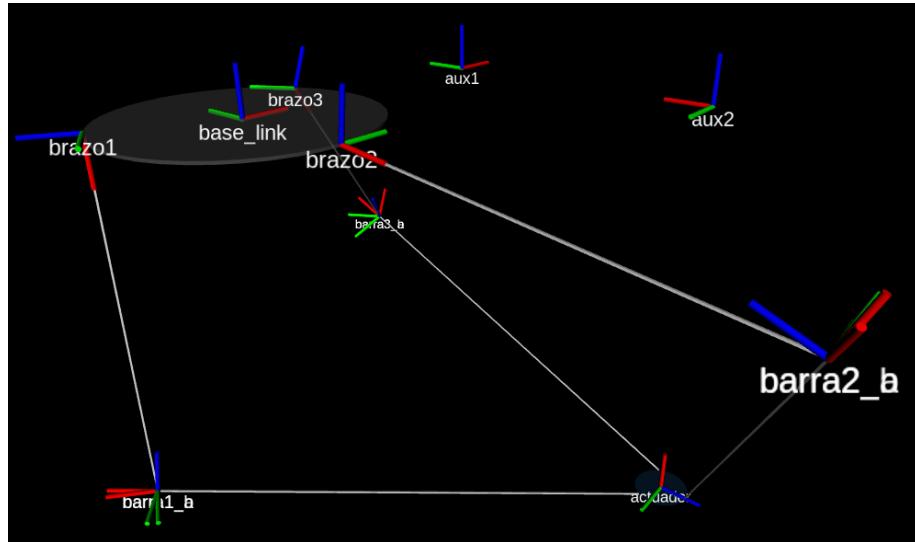


Figura 7.2: Visualización de links y joints del robot delta en RViz

La figura (7.3) se muestra la relación entre cada enlace, es decir, la relación padre-hijo. Se aprecian 4 cadenas cinemáticas las cuales son: 3 compuestas por la base fija-brazo-antebrazo y 1 compuesta por la base móvil.

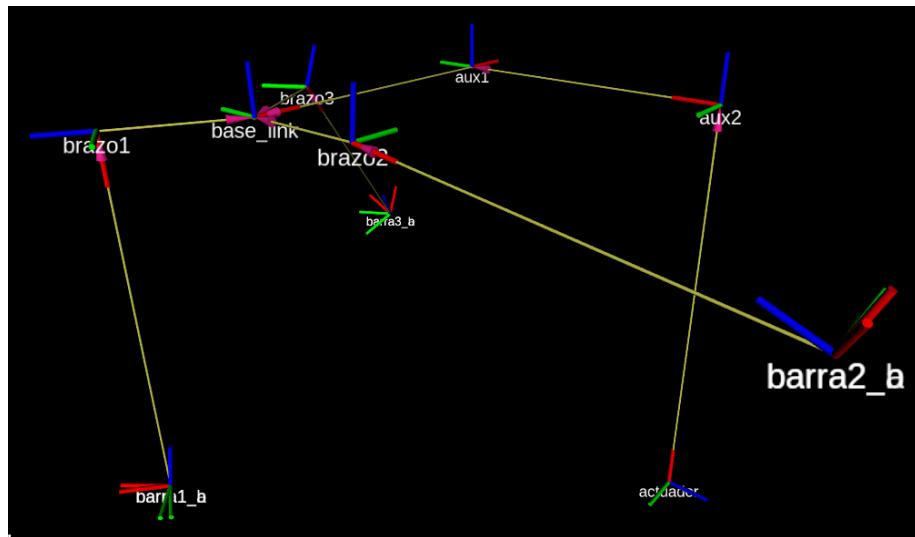


Figura 7.3: Conexión entre los links y joints del robot delta en RViz

7.1.3. Estructura de árbol URDF

La estructura de árbol URDF es una representación gráfica de la relación padre-hijo entre los enlaces y sus marcos de referencia. En la figura (7.4), los nombres de los enlaces son los en rectángulos negros y las juntas en círculos azules. Las juntas son las encargadas de configurar la relación padre-hijo. En la figura *xyz* se refiere a la traslación del marco de referencia de los hijos respecto al marco de referencia del padre y, de modo similar, *rpy* es la rotación del marco de referencia de los hijos respecto a los de los padres. Al igual que en la sección (7.1.2), se aprecian 4 cadenas cinemáticas.

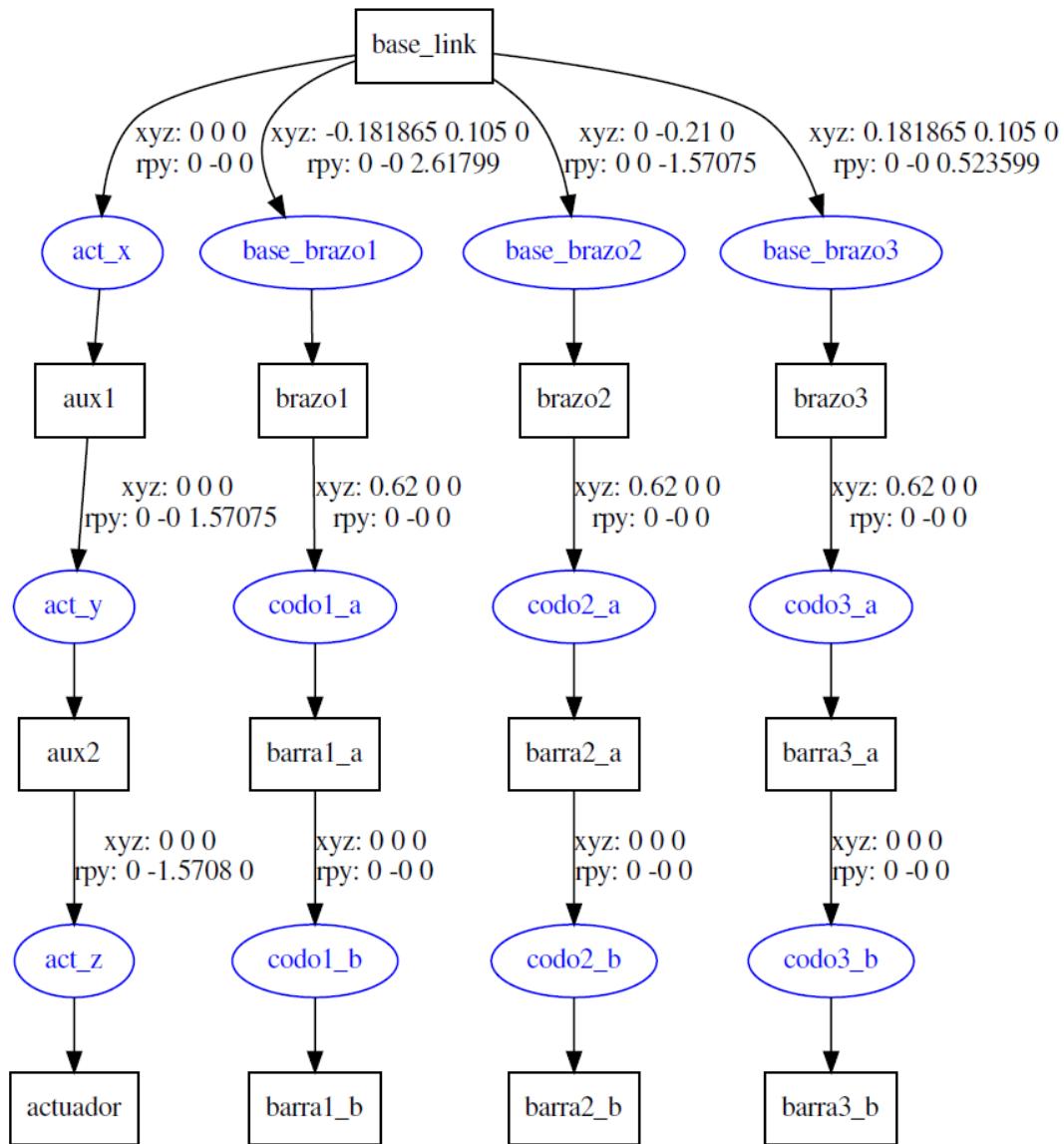


Figura 7.4: Representación gráfica de la relación padre-hijo del robot delta en RViz.

7.2. Espacio de Trabajo

En esta sección se presentan los resultados del espacio de trabajo del robot delta del capítulo (6) con las restricciones impuestas en la tabla (6.2).

7.2.1. Temas y Nodo



Figura 7.5: Temas y nodos en ROS para el calculo del espacio de trabajo del robot delta

- **Subscriber:** el tema *input_workspace* esta configurado por un mensaje escrito en el archivo llamado *parameter_ws.msg*, compuesto por los datos de la tabla (7.3). *Step* es la discretizacion de los ángulos de los motores en grados para graficar el espacio de trabajo. *Graficar_realtime* es una entrada que confirma si se quieren graficar los resultados del espacio de trabajo.
- **Nodo:** el nodo con el nombre *workspace_delta* es el nodo encargado de generar el espacio de trabajo del robot delta a partir de restricciones impuestas.

Tipo de dato	Nombre
bool	graficar_realtime
int64	step

Tabla 7.3: Mensaje parameter_ws.msg utilizado por el tema input_workspace

7.2.2. Espacio de trabajo

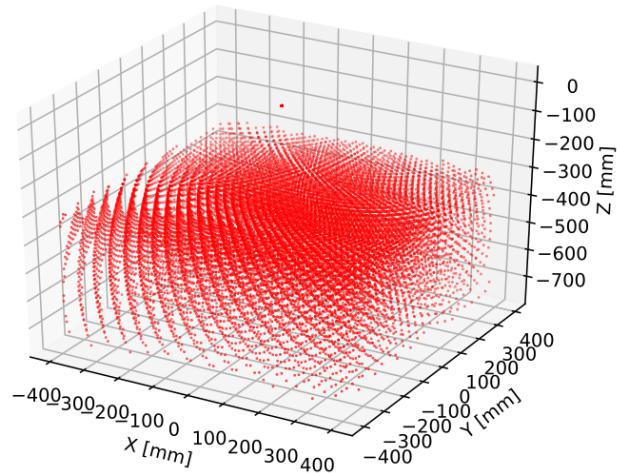


Figura 7.6: Espacio de trabajo

7.2.3. Puntos Alcanzables

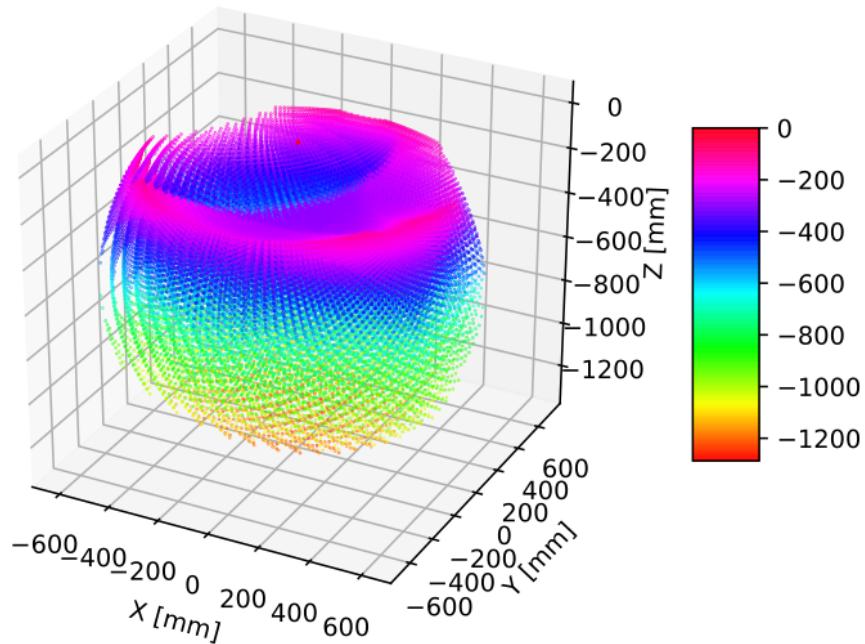


Figura 7.7: Puntos alcanzables del robot delta sin restricciones de límites (figura (7.6))

7.2.4. Proyección plano XY

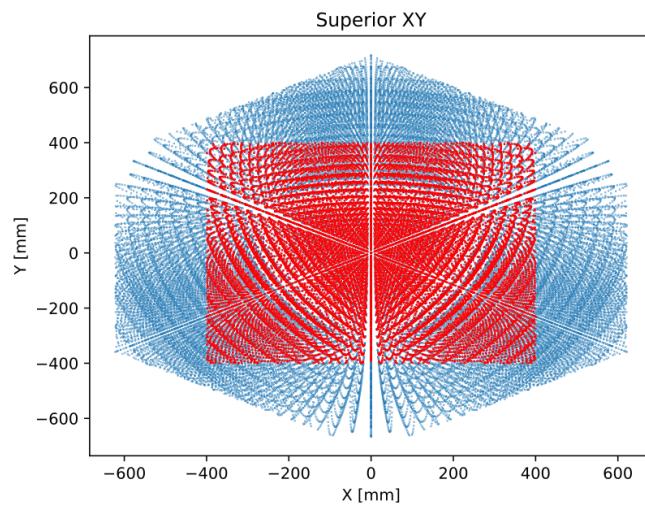


Figura 7.8: Vista del plano XY de la figura (7.6) (en rojo) y (7.7) (en azul)

7.2.5. Proyección plano XZ

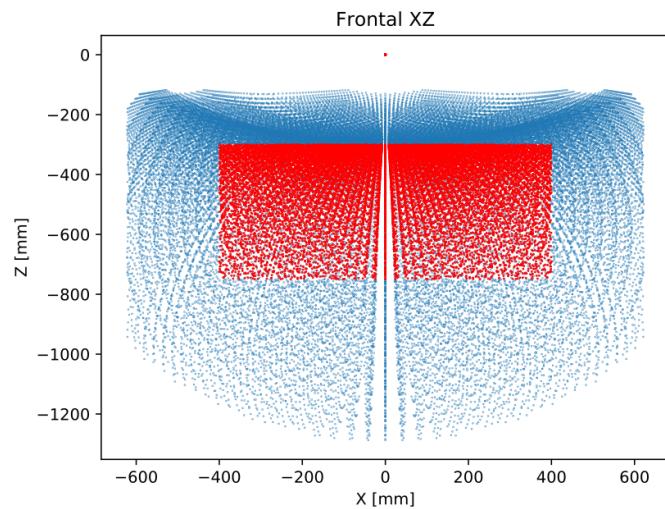


Figura 7.9: Vista del plano XZ de la figura (7.6) (en rojo) y (7.7) (en azul)

7.2.6. Singularidad J_x

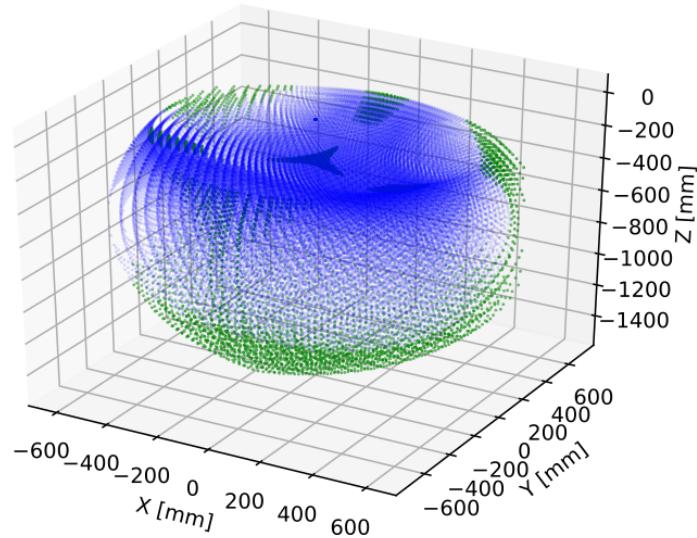


Figura 7.10: Puntos alcanzables figura (7.7) (azul) y puntos con restriccion $J_x \approx 0$ (verde)

7.2.7. Singularidad J_θ

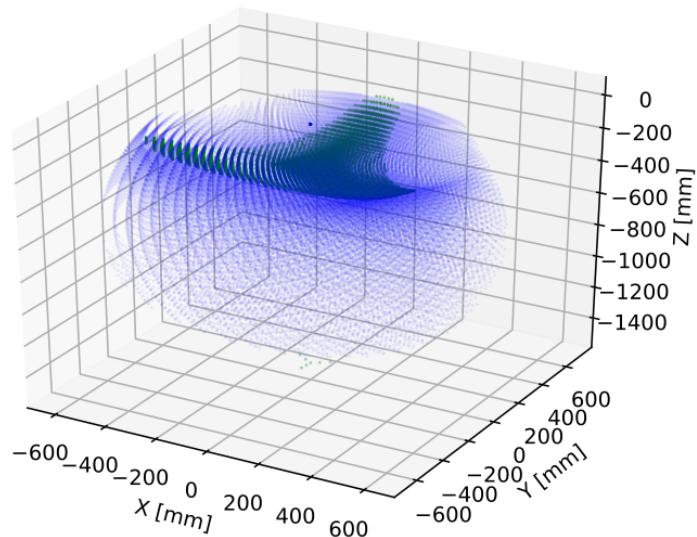


Figura 7.11: Puntos alcanzables figura (7.7) (azul) y puntos con restriccion $J_\theta \approx 0$ (verde)

7.3. Trayectorias

En esta sección se presentan los resultados de los torques aplicados a los actuadores del robot delta del capítulo (5) para realizar las trayectorias impuestas de la sección (6.1.1).

7.3.1. Temas y Nodo



Figura 7.12: Resultado de la dinámica inversa en trayectoria 1

- **Subscriber:** el tema *input_ls_final* esta configurado por un mensaje escrito en el archivo llamado *linear_speed_xyz.msg*, compuesto por los datos de la tabla (7.4). El punto inicial de la trayectoria es (xo,yo,zo) . El punto final de la trayectoria es (xf,yf,zf) . La velocidad y aceleración máxima permitida en la trayectoria lineal es *vmax* y *amax* respectivamente. El tamaño de la división del perfil de velocidad trapezoidal de la trayectoria en el área de aceleración y desaceleración es *paso1*, mientras que en el área de velocidad constante es *paso2*. El numero de la trayectoria a simular es *num_tray*
- **Nodo:** el nodo con el nombre *torque_metodo_1* es el nodo encargado de calcular el torque que debe accionar los motores del robot delta para producir una trayectoria lineal con velocidad tipo trapezoidal de la base móvil.

Tipo de dato	Nombre
float32	xo
float32	yo
float32	zo
float32	xf
float32	yf
float32	zf
float32	vmax
float32	amax
int64	paso1
int64	paso2
int64	num_tray

Tabla 7.4: Mensaje *linear_speed_xyz.msg* utilizado por el tema *input_ls_final*

7.3.2. Trayectoria 1

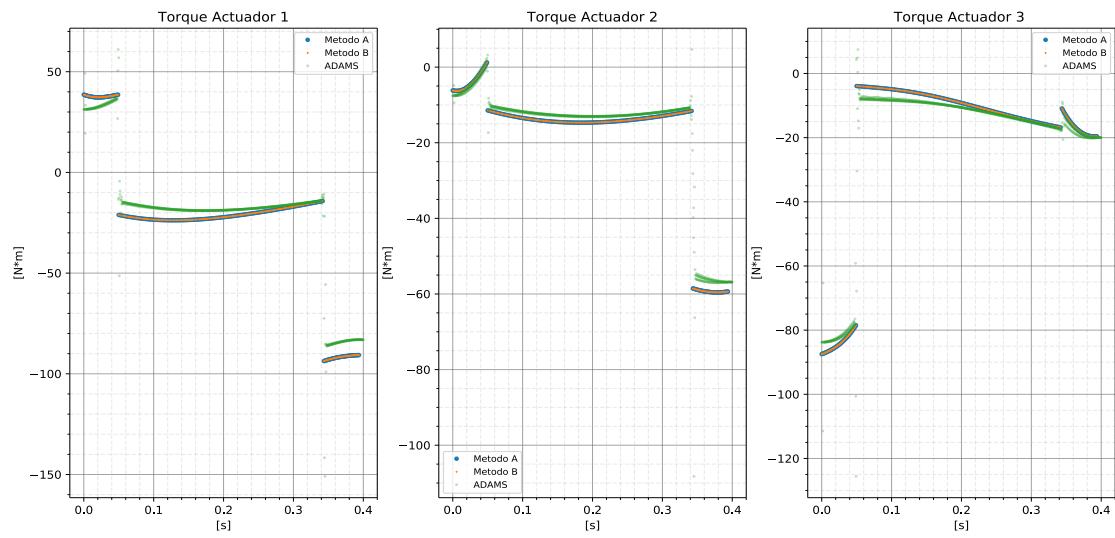


Figura 7.13: Resultado de la dinámica inversa en trayectoria 1

7.3.3. Trayectoria 2

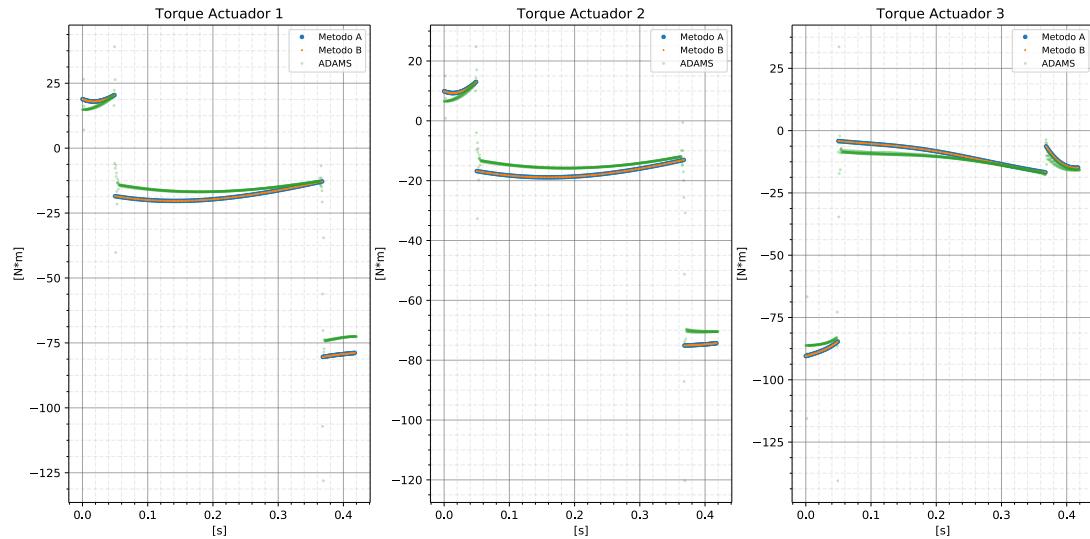


Figura 7.14: Resultado de la dinámica inversa en trayectoria 2

7.3.4. Trayectoria 3

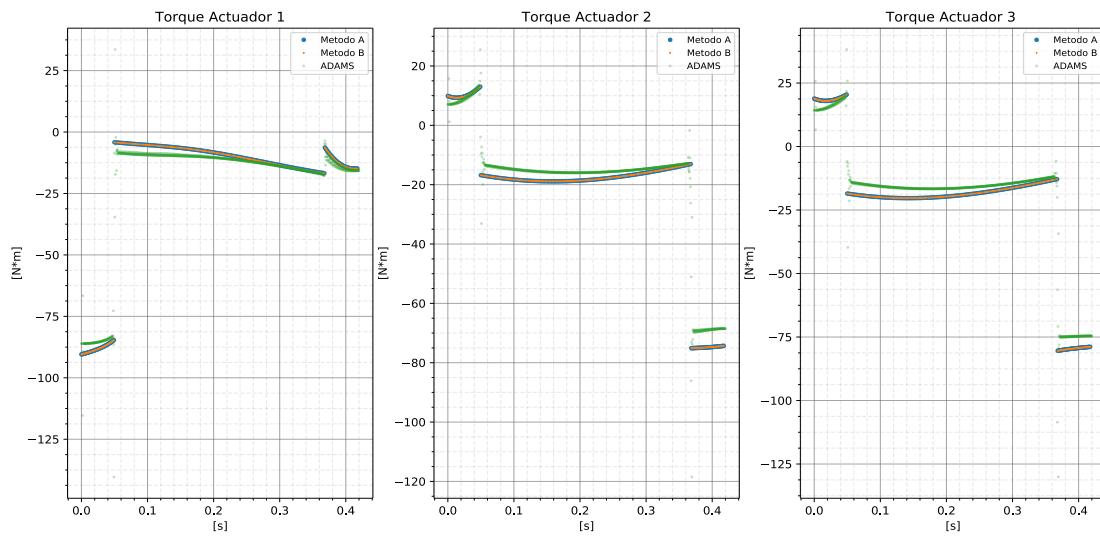


Figura 7.15: Resultado de la dinámica inversa en trayectoria 3

7.3.5. Trayectoria 4

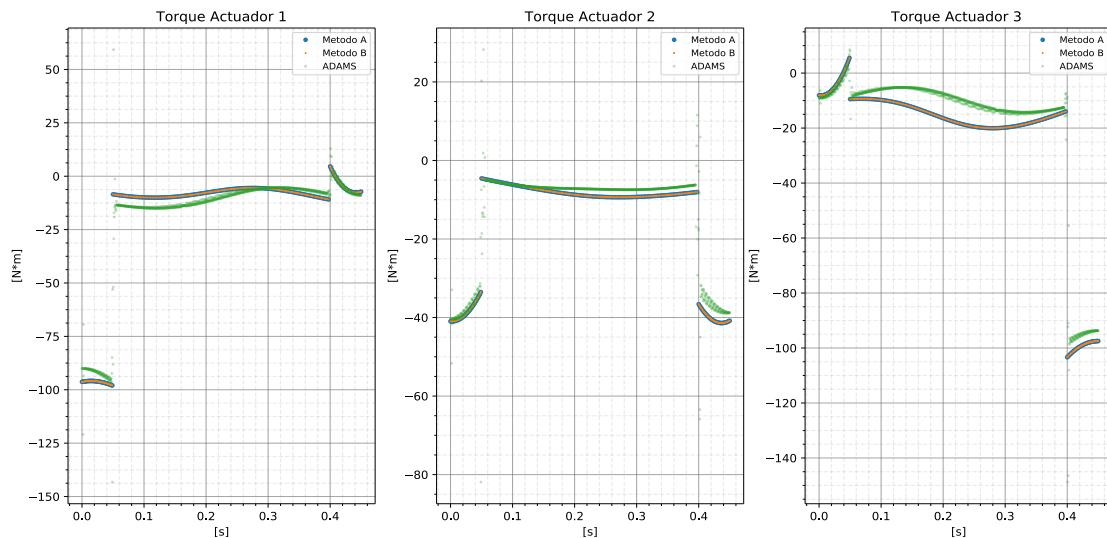


Figura 7.16: Resultado de la dinámica inversa en trayectoria 4

7.3.6. Trayectoria 5

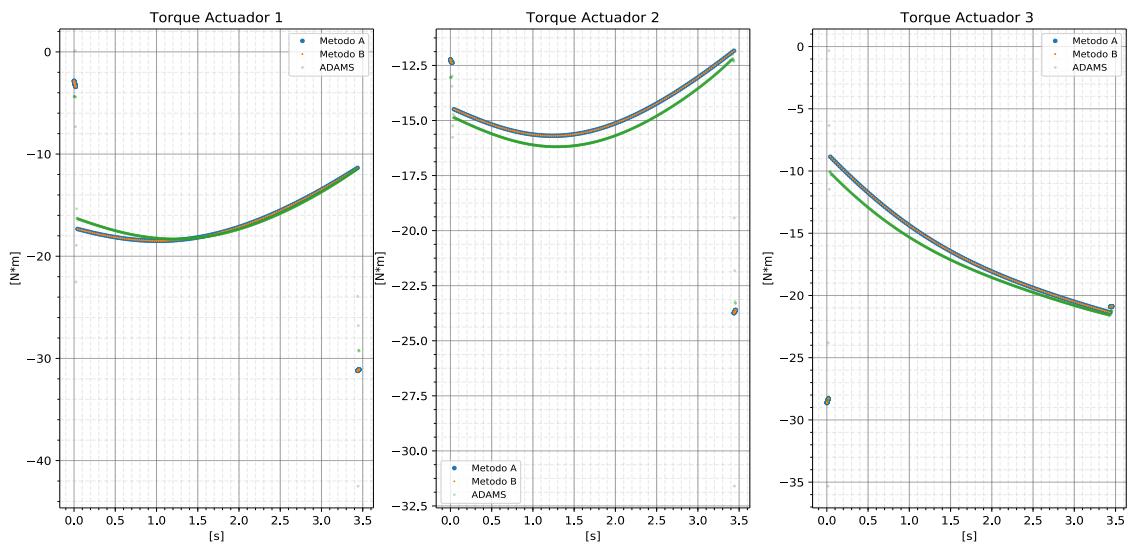


Figura 7.17: Resultado de la dinámica inversa en trayectoria 5

7.3.7. Trayectoria 6

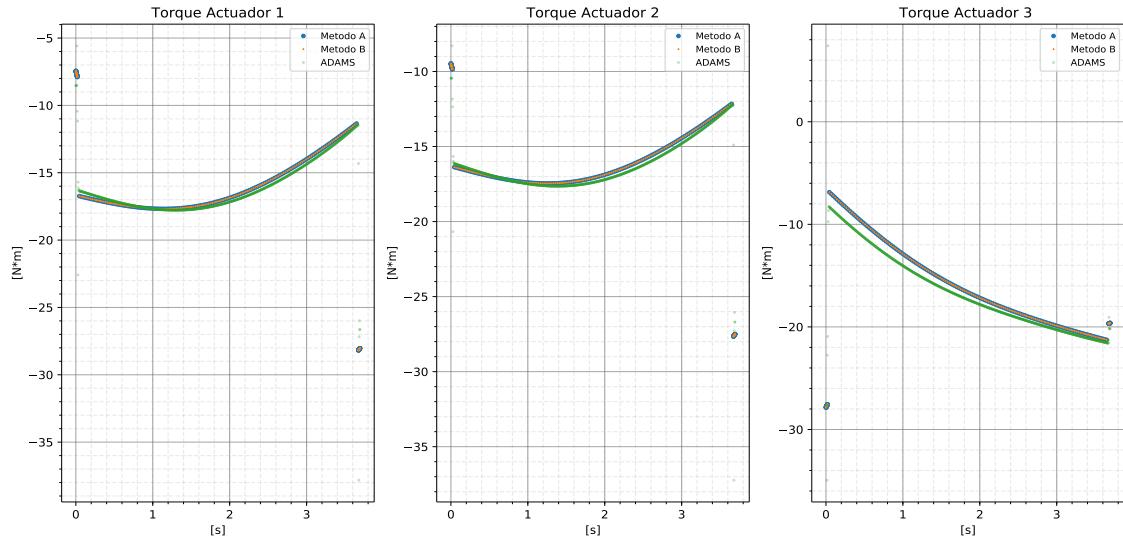


Figura 7.18: Resultado de la dinámica inversa en trayectoria 6

7.3.8. Trayectoria 7

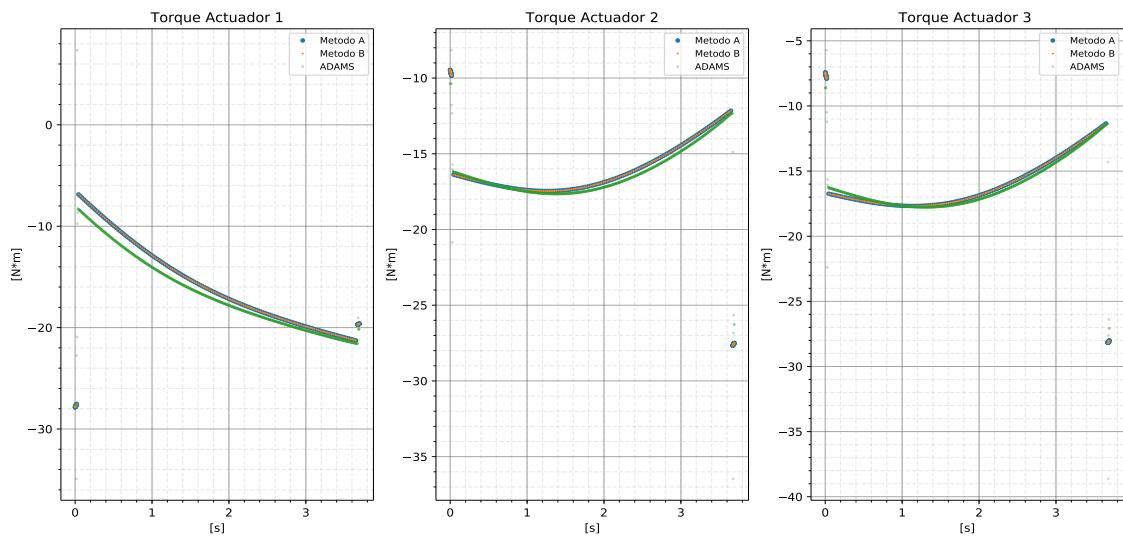


Figura 7.19: Resultado de la dinámica inversa en trayectoria 7

7.3.9. Trayectoria 8

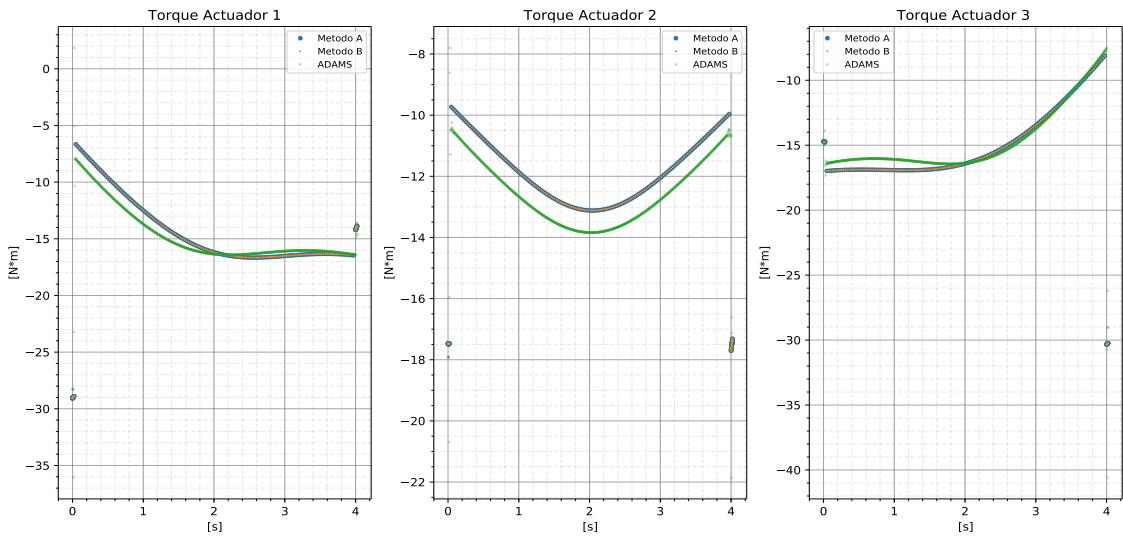


Figura 7.20: Resultado de torques de la trayectoria 8

Capítulo 8

Análisis de resultados y proyecciones a futuro

8.1. Conclusiones y análisis de resultados

La presente tesis tiene como objetivo principal modelar y validar la cinemática y dinámica de un robot delta. Para cumplir con el objetivo anterior, se utilizan diversas herramientas, tales como ROS, RViz y ADAMS. Los siguientes puntos muestran las conclusiones sobre la utilización de las herramientas y resultados:

■ Robot Operating System

El middleware ROS facilita el desarrollo en código abierto del control de robot complejos separando las tareas a realizar por nodos.

Las tareas que se logran segmentar en este documento son 3: visualización 3D de las partes mecánicas, calculo de la dinámica en los actuadores a partir de trayectorias lineales de la base móvil y espacio de trabajo del robot delta.

Recordar que este es el inicio de otras investigaciones, por lo que es fácil acoplar otras tareas a estos nodos. Además, se crea un archivo que contiene las especificaciones del robot delta como valores de entrada, por lo que la modelación cinemática y dinámica para otro robot con distintos valores geométricos, masas o restricciones del espacio de trabajo es posible con estos nodos si se cambian los valores en dicho archivo.

En proyectos futuros es recomendable segregar aun mas estas tareas representadas en nodos, como por ejemplo un nodo que se encargue solo de la trayectoria, otro solo de la cinemática, otro solo de la dinámica, etc.

■ ROS Visualization

Se consigue crear el modelo tridimensional sobre el cual se ha implementado la geometría de un robot delta. Al realizar distintas simulaciones de trayectorias lineales de la base móvil se comprueba a simple vista que el algoritmo de cinemática inversa y el calculo de los ángulos interiores del robot delta es valido ya que la base móvil calza perfectamente con los extremos de los antebrazos. El fundamento de esta conclusión se basa en que la cadena cinemática para el control de la base móvil es distinta a la de los brazos y antebrazos. El movimiento de la base móvil se da por medio de la trayectoria lineal en el espacio cartesiano (datos de entrada XYZ), el movimiento de los brazos es gracias a los ángulos de los brazos calculados por la cinemática inversa ($\theta_{1i \in \{1,2,3\}} = f(XYZ)$) y el movimiento de los antebrazos debido a los ángulos interiores de la junta esférica brazo-antebrazo ($\theta_{2i \in \{1,2,3\}} = f(XYZ, \theta_{1i \in \{1,2,3\}})$ y $\theta_{3i \in \{1,2,3\}} = f(XYZ, \theta_{1i \in \{1,2,3\}})$).

■ Espacio de trabajo

Se cumple el objetivo de determinar el espacio de trabajo del robot delta, identificando singularidades y puntos críticos para prevenir daños en las partes mecánicas y/o en los actuadores. Al variar los valores de las restricciones del espacio de trabajo se producen cambios significativos, tanto en la forma, densidad y puntos alcanzables del espacio.

1. **Límites de ángulos de los brazos θ_{1i} :** El límite mínimo y máximo de los ángulos de los actuadores o de los brazos mecánicos θ_{1i} , se utilizan principalmente para prevenir colisiones entre los brazos y la base fija o actuadores. Además, al imponer límites desde -90° hasta 90° , se obtiene fácilmente una sola solución para la modelación de la cinemática directa.
2. **Discretización angular de los brazos $\Delta\theta_{1i}$:** La discretización del rango de los ángulos de los brazos $\Delta\theta_{1i}$, influye en la densidad de puntos del espacio de trabajo. Entre más pequeña es la discretización, más puntos alcanza la plataforma móvil del robot delta. El valor de la discretización realmente depende del tipo de actuador. En la actualidad, los actuadores como los motores paso a paso tienen discretización de $\Delta\theta_{1i} = 0,00228^\circ$. En esta tesis se utiliza $\Delta\theta_{1i} = 5^\circ$, ya que para valores menores, la librería encargada de los gráficos en python no procesa la visualización del total de puntos alcanzables.
3. **Límites de ángulos interiores θ_{2i} y θ_{3i} :** Sobre las restricciones de los ángulos interiores, es decir, las limitaciones de las juntas esféricas brazo-antebrazo, se concluye que influyen en la forma del espacio de trabajo. En la vista superior del espacio de trabajo se puede apreciar que las restricciones dan como resultado una forma hexagonal y en la vista isométrica se aprecia una forma de cúpula invertida. La restricción de θ_{2i} se utiliza principalmente para evitar

la colisión entre los brazos y antebrazos de una misma cadena cinemática y para singularidades como lo son las colinealidades entre un brazo y su antebrazo. La restricción de θ_{3i} es impuesta principalmente por los catálogos de los fabricantes de las juntas esféricas. Es importante recalcar que estos dos ángulos tienen gran implicación en singularidades que se aprecian cuando el determinante de la matriz jacobiana es cercano a 0.

4. **Determinante de jacobiano J_x y J_θ :** Cuando el determinante de una de estas dos matrices es cercano a 0, se producen singularidades como las explicadas en la sección (4.3.2.4). Las posiciones de los puntos con estas restricciones se encuentran en la periferia del los puntos alcanzables con las restricciones angulares dichas en 1,2 y 3.
5. **Límite impuesto por fabricantes:** Se utiliza una forma de ortoedro para el espacio de trabajo por la simplificación al momento de crear los algoritmos. Los valores de este octaedro son impuestos con la intención de que los puntos alcanzados por la base móvil cumplan con las restricciones angulares y del determinante de jacobianos dichas en 1, 2, 3 y 4.

■ **Validación cinemática y dinámica**

Finalmente, se indican las conclusiones del objetivo principal de esta tesis, la modelación cinemática y dinámica del robot delta. La modelación dinámica busca determinar los torques de los actuadores a partir de los datos de posición, velocidad y aceleración de la base móvil.

Para modelar la cinemática y dinámica se proponen 2 métodos analíticos, llamados métodos A y B. Estos métodos son algoritmos compilados en ROS y se validan por medio del software de simulación ADAMS. Es notorio recalcar que al validar la modelación dinámica también se valida la modelación cinemática, ya que se necesita esta última para determinar los torques de los actuadores del robot delta.

Para ver que los modelos estén correctos, se realizan 8 trayectorias lineales con perfil de velocidad trapezoidal para la base móvil. Son 4 trayectorias rápidas y 4 lentas. Estas 8 trayectorias se simulan en ROS y ADAMS.

Al comparar los resultados de los torques producidos por las 8 trayectorias calculados con los métodos A y B, se puede concluir que no importa los métodos de cinemática y dinámica que se utilicen, siempre deben dar resultados iguales si la modelación y simplificaciones son las mismas. En otras palabras, los resultados de los métodos A y B son casi perfectamente iguales por decimales insignificantes. Se recomienda utilizar el método que tenga menor costo computacional.

Al comparar los resultados de los torques producidos por las 8 trayectorias calculados por los 2 métodos y ADAMS, se puede decir que las curvas son similares,

tanto en su valor como en su forma. Los errores son mínimos, por lo que se puede concluir que la modelación cinemática y dinámica de los 2 métodos es valida. Estos errores pequeños se deben principalmente por 2 configuraciones en ADAMS. La primera se debe a que en ADAMS no se configura robot delta con cadenas cinemáticas cerradas, ya que el tipo de juntas entre la masa de la base fija y las masas de los extremos inferiores de los antebrazos no es la correcta respecto a los métodos A y B. La relación correcta entre estas masas sería configurándolas con juntas de tipo fijas para que la base móvil sea paralela a la base fija, pero al hacer esto, ADAMS no logra resolver la dinámica. Esto trae como consecuencia que la base móvil se incline al momento de simular las trayectorias lineales. La segunda configuración es similar de la primera, error en la solución de la dinámica en ADAMS al imponer cadenas cinemáticas cerradas. Para poder simular la trayectoria, se simplifica la masa de la base móvil dividiéndola en 3 y desplazándolas a los extremos inferiores de los antebrazos, es decir, a la posición de las juntas antebrazo - base móvil. Como la distancia entre el centroide de la base fija y las juntas antebrazo - base móvil es pequeña, la influencia de esta simplificación en el torque de los actuadores es baja. Un factor importante a considerar es verificar a futuro si esta diferencia entre los métodos y ADAMS se debe al método conque ADAMS resuelve la dinámica. ADAMS utiliza para calcular las derivadas el método de Euler invertido y para resolver los sistemas de ecuaciones dinámicos y cinemáticos Newton Raphson en forma matricial. A pesar de todos estas complicaciones, se puede asumir valida la modelación cinemática y dinámica del robot delta ya que en la practica, errores pequeños en la escritura del código y en la modelación de robots conllevan a errores grandes en los resultados dinámicos, especialmente en trayectorias de alta velocidad.

8.2. Futuras líneas de investigación

A fin de aumentar el conocimiento en el área de la robótica en nuestra universidad y la industrial, sería de interés estudiar los siguientes puntos:

- **Optimización dimensional:** Optimización de las dimensiones del robot delta, tales como la base fija, brazos, antebrazos y base móvil basado en el menor consumo de energía.
- **Optimización de trayectoria:** Algoritmo de optimización de trayectoria considerando torques mínimo y máximo permitido por los actuadores. La función objetivo se basa en una proporción entre gastar la mínima energía y/o el tiempo mínimo de trayectoria. Ejemplos de estos algoritmos son: Phase Plane Method y Dynamic Programming Discrete Optimal Control.
- **Trayectoria pick-and-place:** Método de planificación de trayectoria de tiempo óptimo basado en curvas quínticas de Pitágoras-Hodógrafa (PH) (polinomial 3-4-5) para realizar la operación suave y estable a alta velocidad.
- **Controlador PID:** Este artículo presenta el control de seguimiento para un manipulador robótico tipo delta que emplea controladores PID.
- **Lazo Cerrado:** Implementar datos de retroalimentación para controlar la trayectoria. Se pueden utilizar sensor de imágenes para la posición de la base móvil, sensores de posición angular y torque para los actuadores.
- **Calibración con redes neuronales:** Desarrollo de un algoritmo de calibración para el posicionamiento visual de Delta Robot basado en red neuronal
- **Cinemática de posición con machine learning:** Análisis de seguridad mediante cinemática directa del robot paralelo delta mediante aprendizaje automático
- **Modelo 3D:** Crear un modelo URDF mas complejo. Agregar mas marcos de referencia y que la visualización de piezas sea en base a nube de puntos *.stl.
- **Pinzas:** Diseño e Implementación de un prototipo de una pinza para manipular objetos frágiles
- **Visión por computadora:** Algoritmos de detección de objetos en cinta transportadora
- **Actuadores:** Aplicación de motores paso a paso controlando driver de motores con trayectoria de pulsos.
- **Curva de torque:** Obtener torques máximos y mínimos vs la velocidad angular de los motores paso a paso con sensor de torque rotacional

Bibliografía

1. **DIARIO LIBRE.** *República Dominicana recibirá a Sophia, la robot humanoide [online]*. 2019 [visitado 2021-02-27]. Disponible desde: <https://www.diariolibre.com/actualidad/tecnologia/republica-dominicana-recibira-a-sophia-la-robot-humanoide-EK14690589>.
2. **MATHURES, Paul.** *Growing up with humans [online]*. The Telegraph, 2021 [visitado 2021-02-27]. Disponible desde: <https://www.telegraphindia.com/science-tech/growing-up-with-humans/cid/1807352#>.
3. **EPFL.** *Le père du robot delta revient sur sa carrière [online]*. 2012 [visitado 2021-02-27]. Disponible desde: [https://sti.epfl.ch/reymond-clavel-creator-of-the-delta-robot-reflects-on-his-career](https://sti.epfl.ch/reymond-clavel-creator-of-the-delta-robot-reflects-on-his-career/#reymond-clavel-creator-of-the-delta-robot-reflects-on-his-career).
4. **FLORES CABALLERO, Geovanni.** *Visión por computadora para la recolección de objetos mediante el control del robot manipulador Scrbot ER4pc*. Mexico, Tesis de Maestría. Instituto Politécnico Nacional (IPN). Disponible desde: <http://tesis.ipn.mx/handle/123456789/19975>.
5. **BOSTON DYNAMICS.** *Spot classic (2015) [online]*. 2019 [visitado 2021-02-27]. Disponible desde: <https://www.bostondynamics.com/sites/default/files/2019-09/spotclassic2.jpg>.
6. **IROBOT CORP.** *iRobot recibe un pedido de \$ 7.6 millones de la Marina de los EE. UU. [online]*. PRNewswire, 2014 [visitado 2021-02-28]. Disponible desde: <https://www.prnewswire.com/news-releases/irobot-receives-76-million-order-from-the-us-navy-577446978.html>.
7. **D'ONFRO, Jillian.** *Here Are All The Crazy Military Robots Google Just Bought [online]*. Business Insider, 2013 [visitado 2021-02-28]. Disponible desde: <https://www.businessinsider.com/google-boston-dynamics-robots-2013-12>.
8. **GONG.** *Hola a todos, permítanme presentarles, estos son robots [online]*. Kknews, 2017 [visitado 2021-02-28]. Disponible desde: <https://kknews.cc/code/qomegpy.html>.
9. **MESA MONTOYA, Carlos Andrés; HERRERA LÓPEZ, Marlon Jhair y HOLGUÍN LONDONO, Germán Andrés.** *Prototipado virtual y cosimulación aplicado a un manipulador paralelo tipo Delta de tres grados de libertad para estudios de comportamiento cinemático y cinético*. Scientia et Technica [online]. 2018, vol. 23, n.º 2 [visitado 2021-02-28]. ISSN 0122-1701. Disponible desde: <https://dialnet.unirioja.es/servlet/articulo?codigo=6643333>.

10. **VÄLIMÄKI, Pasi.** *Delta-robotin mekaniikkasuunnittelu*. 2015. Tesis de Grado. Universidad de Ciencias Aplicadas de Seinäjoki. Disponible desde: <http://urn.fi/URN:NBN:fi:amk-201505045792>.
11. **LASELLE, Rush.** *Solar Cell Manufacturing and Robot Automation: Right Fit, Right Robot* [online]. Tech Briefs Media Group, 2012 [visitado 2021-02-28]. Disponible desde: <https://www.techbriefs.com/component/content/article/tb/pub/features/articles/12870>.
12. **ATHANASIOS, Loukas.** *The industrial Robot Types and Their Different Uses* [online]. How To Robot, 2020 [visitado 2021-02-28]. Disponible desde: <https://www.howtorobot.com/expert-insight/industrial-robot-types-and-their-different-uses>.
13. **GONZALEZ, Carlos.** *What's the Difference Between Industrial Robots?* [online]. Machine Design, 2016 [visitado 2021-02-28] Disponible desde url: <https://www.machinedesign.com/markets/robotics/article/21835000/whats-the-difference-between-industrial-robots>.
14. **SOFFAR, Heba.** *Automation, Industrial Robots types importance (Cartesian, SCARA robots, Cylindrical, Parallel ,Delta, Polar and Articulated Robots)* [online]. Science online, 2019 [visitado 2021-02-28]. Disponible desde: <https://www.online-sciences.com/robotics/automation-industrial-robots-types-importance-cartesian-scara-robots-cylindrical-parallel-delta-polar-and-articulated-robots/>.
15. **CONSULTANCY.EU.** *Meet McKinsey Company in Europe during its Expedition event* [online]. 2019 [visitado 2021-02-28]. Disponible desde: <https://www.consultancy.eu/news/2882/meet-mckinsey-company-in-europe-during-its-expedition-event>.
16. **MANYIKA, James; CHUI, Michael y MIREMADI, Mehdi.** *These Are the Jobs Least Likely to Go to Robots* [online]. Fortune Media IP Limited, 2016 [visitado 2021-02-28]. Disponible desde: <https://fortune.com/2016/07/11/skills-gap-automation/>.
17. **FREY, Carl Benedikt y OSBORNE, Michael A.** *The future of employment: How susceptible are jobs to computerisation? Technological Forecasting and Social Change*. 2017, vol. 114, págs. 254-280. ISSN 0040-1625. Disponible desde DOI: <https://doi.org/10.1016/j.techfore.2016.08.019>.
18. **TEULIERES, Marc; TILLEY, Jonathan; BOLZ, Lea; LUDWIG-DEHM, Peter Manuel y WÄGNER, Susanne.** *Industrial robotics: Insights into the sector's future growth dynamics* [online]. McKinsey Company, 2019 [visitado 2021-02-28] Disponible desde <https://www.mckinsey.com/~/media/McKinsey/Industries/Advanced%20Electronics/Our%20Insights/Growth%20dynamics%20in%20industrial%20robotics/Industrial-robotics-Insights-into-the-sectors-future-growth-dynamics.ashx>.
19. **DE BACKER, Koen; DESTEFANO, Timothy; MENON, Carlo y SUH, Jung Ran.** *Industrial robotics and the global organisation of production*. 2018. Disponible desde DOI: <https://doi.org/https://doi.org/10.1787/dd98ff58-en>.

20. STOCKINGER, Georg. *Robotics will eat the world* [online]. Paua Ventures, 2020 [visitado 2021-02-28]. Disponible desde: <https://medium.com/paua-insights/robotics-will-eat-the-world-48f1415d25fd>.
21. GONZALEZ, Felipe; JIMENEZ, Alexander y CARDENAS, Pedro-F. *Robot paralelo tipo Diseño de un sistema de movimiento de ejes complejos: Robot paralelo tipo Delta.* 2013. Disponible desde DOI: [10.13140/RG.2.2.25952.74246](https://doi.org/10.13140/RG.2.2.25952.74246). Tesis de grado. Universidad Nacional de Colombia.
22. CLAVEL, Reymond. *Conception d'un robot parallèle rapide à 4 degrés de liberté.* 1991, págs. 146. Disponible desde DOI: [10.5075/epfl-thesis-925](https://doi.org/10.5075/epfl-thesis-925).
23. KUMAR, Akshay. *Learn all about Robot Operating System 2, including how it's used and how it differs from the original ROS.* [online]. Marker Pro, 2020. urlalso: <https://maker.pro/ros/tutorial/robot-operating-system-2-ros-2-introduction-and-getting-started>.
24. ACKERMAN, Evan. *Open Source Robotics Foundation Accelerates ROS Transition, Hires Key ROS Developers From Willow* [online]. IEEE Spectrum, 2013 Disponible desde url: <https://spectrum.ieee.org/automaton/robotics/robotics-software/osrf-accelerates-ros-transition>.
25. GUESS, A.R. *Semantic Web Jobs: Stanford University* [online]. Dataversity Digital LLC, 2014 Disponible desde url: <https://www.dataversity.net/semantic-web-jobs-stanford-university-4/#>.
26. PYO, YoonSeok; CHO, HanCheol; JUNG, RyuWoon y LIM, TaeHoon. *A Handbook is written by TurtleBot3 Developers.* ROS Robot Programming. 1st. ROBOTIS Co.,Ltd., 2017. ISBN 979-11-962307-1-5. Disponible desde url: <https://community.robotsource.org/t/download-the-ros-robot-programming-book-for-free/51/>.
27. LUM, Joshua S. *Utilizing Robot Operating System (ROS) in robot vision and control.* 2015. Tesis de grado. Naval Postgraduate School. Disponible desde url: <https://calhoun.nps.edu/handle/10945/47300>.
28. ZHANG, Lin; MERRIFIELD, Robert; DEGUET, Anton y YANG, Guang-Zhong. Powering the world's robots—10 years of ROS. *Science Robotics.* 2017, vol. 2, n.º 11. Disponible desde DOI: [10.1126/scirobotics.aar1868](https://doi.org/10.1126/scirobotics.aar1868).
29. ROS METRICS Disponible desde url: <https://www.metrics.ros.org>.
30. THOMAS, Dirk. *Visualize ROS contributions over time* [online]. GitHub Disponible desde url: https://github.com/dirk-thomas/rosworld2020_ros-contributions.
31. JOSEPH, Lentin. *Mastering ROS for Robotics Programming.* Ed. por PUBLISHING, Packt. Packt Publishing, 2015. ISBN 9781783551798. Disponible desde url: <https://www.packtpub.com/hardware-and-creative/mastering-ros-robotics-programming>.
32. WANG, Hao. *ROS study notes (6) catkin_make compilation* [online]. CSDN, 2019 Disponible desde url: <https://blog.csdn.net/li528405176/article/details/82351915>.
33. FANKHAUSER, Péter; JUD, Dominic; WERMELINGER, Martin y HUTTER, Marco. *Programming for Robotics - Introduction to ROS.* 2017. Disponible desde DOI: [10.13140/RG.2.2.14140.44161](https://doi.org/10.13140/RG.2.2.14140.44161).

34. **JOSEPH, Lentin.** *ROS Robotics Projects*. Ed. por PUBLISHING, Packt. Packt Publishing, 2017. ISBN 9781783554713. Disponible desde url: <https://www.packtpub.com/product/ros-robotics-projects/9781783554713>.
35. **MAHTANI, Anil; SANCHEZ, Luis; FERNANDEZ, Enrique; MARTINEZ, Aaron y JOSEPH, Lentin.** *ROS Programming: Building Powerful Robots*. Ed. por PUBLISHING, Packt. Packt Publishing, 2018. ISBN 9781788627436. Disponible desde url: <https://www.packtpub.com/hardware-and-creative/ros-programming-building-powerful-robots#additional>.
36. **BESTMANN, Marc; FAKULTÄT, Min; ZHANG, Jianwei y HENDRICH, N.** 2017. Tesis doctoral.
37. **KOUBAA, Anis.** *Robot Operating System (ROS): The Complete Reference (Volume 1)*. 1st. Springer Publishing Company, Incorporated, 2016. ISBN 3319260529.
38. **ROS, Wiki.** *URDF Tutorials* [online]. Wiki ROS, 2016 [visitado 2021-03-01]. Disponible desde: <http://wiki.ros.org/urdf/Tutorials>.
39. **AHMED A. SHABANA.** *Dynamics of Multibody Systems*. University of Illinois, Chicago: Cambridge University Press, 2013. Disponible desde DOI: <https://doi.org/10.1017/CBO9781107337213>.
40. **MSC ADAMS.** *Basic ADAMS Full Simulation Training Guide*. 11.0.^a ed. Mechanical Dynamics, Incorporated, 2001.
41. **RILO ANTELO, Lois.** *Diseño e implementación de un sistema de control para la representación gráfica a partir de imágenes*. 2016. Projecte Final de Màster Oficial. Escola Tècnica Superior d'Enginyeria Industrial de Barcelona. Disponible desde url: <https://upcommons.upc.edu/handle/2117/98930>.
42. **HSU, Kuei-shu; KARKOUB, M.; TSAI, M. y HER, M.** *Modelling and index analysis of a Delta-type mechanism. Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*. Vol. 218, n.^o 3. ISSN 1464-4193. Disponible desde DOI: [10.1243/1464419042035944](https://doi.org/10.1243/1464419042035944).
43. **SOLDOVIERI C., T.** *INTRODUCCION A LA MECANICA DE LAGRANGE Y HAMILTON*. Universidad de Zulia, Venezuela: Preprint, 2013. Disponible desde url: <http://tsoldovieritsweb.ihostfull.com/documentos/IntMecLagHamAct522019.pdf>.
44. **SHAREEF, Zeeshan.** *Path Planning and Trajectory Optimization of Delta Parallel Robot*. Verlagsschriftenreihe des Heinz Nixdorf Instituts, Paderborn, 2015. Dissertation. Fakultät für Maschinenbau, Universität Paderborn. Disponible desde url: <https://www.hni.uni-paderborn.de/pub/9065>.
45. **CODOUREY, Alain.** *Contribution à la commande des robots rapides et précis application au robot delta à entraînement direct*. 1991. Disponible desde DOI: [10.5075/epfl-thesis-922](https://doi.org/10.5075/epfl-thesis-922).
46. **GUGLIELMETTI, Philippe.** *Model-based control of fast parallel robots a global approach in operational space*. 1994, págs. 172. Disponible desde DOI: [10.5075/epfl-thesis-1228](https://doi.org/10.5075/epfl-thesis-1228).
47. **CODOUREY, Alain.** *Dynamic Modeling of Parallel Robots for Computed-Torque Control Implementation. The International Journal of Robotics Research*. 1998, vol. 17, n.^o 12, págs. 1325-1336. Disponible desde DOI: [10.1177/027836499801701205](https://doi.org/10.1177/027836499801701205).

48. **ADETU, C. F.; MOORE, C. A. y ROBERTS, R. G.** *Dynamic modeling and control of the OMEGA-3 parallel manipulator*. En: *2009 IEEE International Conference on Systems, Man and Cybernetics*. 2009, págs. 3599-3604. Disponible desde DOI: [10.1109/ICSMC.2009.5346850](https://doi.org/10.1109/ICSMC.2009.5346850).
49. **CODOUREY, A.** *Dynamic modelling and mass matrix evaluation of the DELTA parallel robot for axes decoupling control*. En: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS '96*. 1996, vol. 3, 1211-1218 vol.3. Disponible desde DOI: [10.1109/IROS.1996.568973](https://doi.org/10.1109/IROS.1996.568973).
50. **KLEINFINGER, J.F. y KHALIL, W.** *MODELISATION DYNAMIQUE DE ROBOTS A CHAINE: CINEMATIQUE SIMPLE, ARBORESCENTE, OU FERMEE, EN VUE DE LEUR COMMANDE*. 1986. Lille tesis. Disponible desde url: <https://books.google.cl/books?id=dUSCtgAACAAJ>.
51. **CODOUREY, A. y BURDET, E.** *A body-oriented method for finding a linear form of the dynamic equation of fully parallel robots*. En: *Proceedings of International Conference on Robotics and Automation*. 1997, vol. 2, 1612-1618 vol.2. Disponible desde DOI: [10.1109/ROBOT.1997.614371](https://doi.org/10.1109/ROBOT.1997.614371).
52. **ZHANG, Chang-De y SONG, Shin-Min.** *An efficient method for inverse dynamics of manipulators based on the virtual work principle*. *Journal of Robotic Systems*. 1993, vol. 10, n.º 5, págs. 605-627.
53. **KOKKINIS, T y STOUGHTON, R.** *Dynamics and control of closed-chain robot arms with application to a new direct-drive robot arm*. *Int. Journal of Robotics and Automation*. 1991, vol. 6, n.º 1, págs. 131-145.
54. **NAKAMURA, Yoshihiko.** *Advanced robotics : redundancy and optimization*. Reading, Massachusetts: Addison-Wesley, 1991. ISBN 0-201-15198-7.
55. **LT, Wang.** *On the dynamic analysis of general parallel robotic manipulators*. *Int. J. Robotics Automat*. 1994, vol. 9, págs. 81-87.
56. **JI, Zhiming.** *Study of the effect of leg inertia in Stewart platforms*. En: ANON (ed.). *Proceedings - IEEE International Conference on Robotics and Automation*. Publ by IEEE, 1993, págs. 121-126. *Proceedings - IEEE International Conference on Robotics and Automation*. ISBN 0818634529. *Proceedings of the IEEE International Conference on Robotics and Automation ; Conference date: 02-05-1993 Through 06-05-1993*.
57. **LARIBI, M.A.; ROMDHANE, L. y ZEGHLOUL, S.** *Advanced Synthesis of the DELTA Parallel Robot for a Specified Workspace*. En: WU, Huapeng (ed.). *Parallel Manipulators Towards New Applications*. Rijeka: IntechOpen, 2008, cap. 10. Disponible desde DOI: [10.5772/5432](https://doi.org/10.5772/5432).
58. **DASH, Anjan Kumar; CHEN, I-Ming; YEO, Song Huat y YANG, Guilin.** *Workspace generation and planning singularity-free path for parallel manipulators*. *Mechanism and Machine Theory*. 2005, vol. 40, n.º 7, págs. 776-805. ISSN 0094-114X. Disponible desde DOI: <https://doi.org/10.1016/j.mechmachtheory.2005.01.001>.
59. **LARIBI, M.A.; ROMDHANE, L. y ZEGHLOUL, S.** *Analysis and dimensional synthesis of the DELTA robot for a prescribed workspace*. *Mechanism and Machine Theory*. 2007, vol. 42, n.º 7, págs. 859-870. ISSN 0094-114X. Disponible desde DOI: <https://doi.org/10.1016/j.mechmachtheory.2006.06.012>.

60. **BENTALEB, Toufik; HAMMACHE, Hakim y BELOUCHRANI, Mohamed.** Works-space, Accuracy Analysis and Kinematic Calibration of a "Delta" Parallel Robots. En: 2011.
61. **AFFI, Z.; ROMDHANE, L. y MAALEJ, A.** *Dimensional synthesis of a 3-translational-DOF in-parallel manipulator for a desired workspace*. European Journal of Mechanics - A/Solids. 2004, vol. 23, n.º 2, págs. 311-324. ISSN 0997-7538. Disponible desde DOI: <https://doi.org/10.1016/j.euromechsol.2004.01.003>.
62. **URREA, L. M. y MEDINA, S. A.** *Diseño e implementación de una plataforma robótica tipo delta*. Repositorio Institucional UMNG, 2013. Universidad Militar Nueva Granada. Disponible desde url: <http://hdl.handle.net/10654/9953>.
63. **KAWASAKI, Robot.** Y series High-speed picking robot [online]. 2019 [visitado 2021-03-02]. Disponible desde: http://kawasakirobotics.nl/index.php?id_product=37&controller=product&id_lang=2.
64. **RAMOS, Oscar E.** Generación de Trayectorias Cinemáticas. *Curso: Fundamentos de Robótica*. 2020. Disponible desde url: <https://profesores.utec.edu.pe/oramos/teaching/201/fund-robotica/lectures.html>.
65. **LYNCH, Kevin M. y PARK, Frank C.** *Modern Robotics: Mechanics, Planning, and Control*. 1st. USA: Cambridge University Press, 2017. ISBN 1107156300.
66. **BARRIENTOS, A.; BALAGUER, C. y PEÑIN, L.** *Fundamentos de robótica*. 2st. McGraw-Hill Interamericana de España S.L., 2007. ISBN 9788448156367.
67. **SILVA, Luis Ángel.** *Control visual de robots paralelos : análisis, desarrollo y aplicación a la plataforma RoboTenis*. 2005. Tesis Doctoral. E.T.S.I. Industriales (UPM). Disponible desde url: <http://oa.upm.es/378/>.

APÉNDICE

Apéndice A

Desarrollo de formulas teóricas

En este anexo se presentan el desarrollo detallado de las fórmulas presentadas en el capítulo 4.

A.1. Método A

A.1.1. Modelación cinemática de posición

A.1.1.1. Cinemática directa

La cinemática directa busca encontrar la posición del centroide del efecto en el espacio cartesiano por medio de la configuración del espacio articular de los actuadores del robot delta.

$$(\theta_1, \theta_2, \theta_3) \rightarrow E_0 (x_0, y_0, z_0)$$

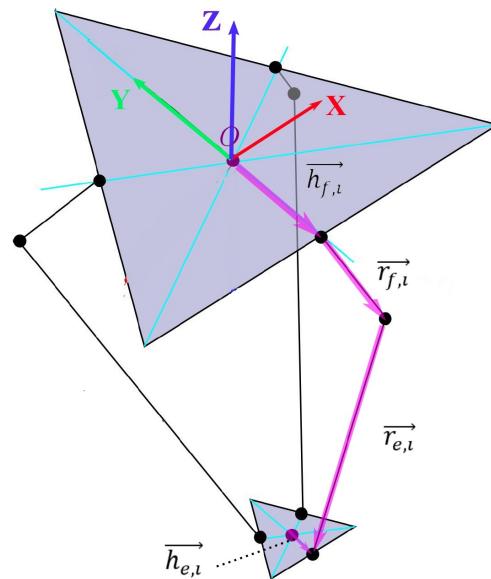
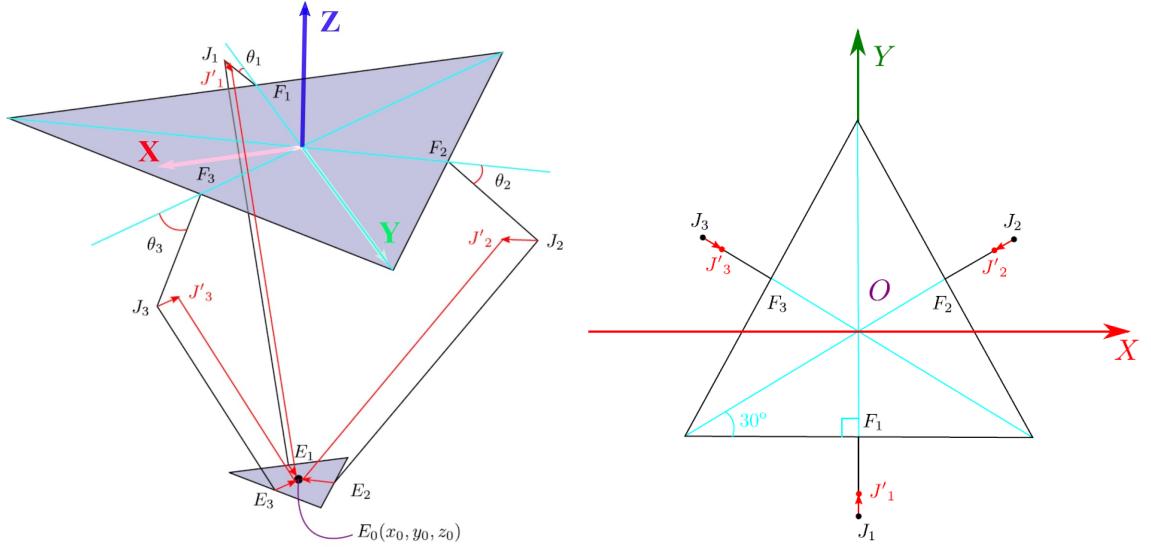


Figura A.1: Vectorización cinemática directa método A



(a) Representacion grafica desplazamiento del vector que representa el antebrazo hacia el punto E_0

(b) Vista superior base fija

Figura A.2: Desplazamiento de vectores para la solución de la cinemática directa del método A

Las posiciones en el espacio de los vectores $\vec{r}_{f,i}$, que representan los brazos del robot delta, ya están definidos en magnitud y sentido debido a que se conoce los ángulos de los actuadores ($\theta_1, \theta_2, \theta_3$) y la longitud de los brazos.

Sobre el vector $\vec{r}_{e,i}$, que representa los antebrazos, solo se conoce el punto inicial J_i que coincide con el extremo del vector $\vec{r}_{f,i}$. En consecuencia, la orientación para cada vector $\vec{r}_{e,i}$, está restringido por una esfera con centro en la junta esférica J_i , que conecta el brazo con el antebrazo, con radio equivalente al largo del antebrazo r_e .

Finalmente, para obtener las coordenadas del centroide del efecto final $E_0(x_0, y_0, z_0)$, se realiza una traslación de las esferas mencionadas anteriormente. Esta traslación depende de la dirección y longitud del vector $\vec{h}_{e,i}$ para cada cadena x respectivamente. Se generan 3 nuevas esferas, con centros en J'_i y de radio equivalente al largo del antebrazo r_e . Estas esferas trasladadas se intersectan en un punto en común, el centroide del efecto $E_0(x_0, y_0, z_0)$. Por lo tanto, se calculan las coordenadas del punto $E_0(x_0, y_0, z_0)$ realizando un sistema de ecuaciones no lineal con 3 restricciones impuestas por las 3 esferas.

El primer paso para calcular el centroide del efecto final E_0 es encontrar una manera de representar las posiciones de las juntas J_i (brazo-antebrazo) con los parámetros conocidos. Por medio de geometría básica se pueden determinar otros parámetros para representar el punto J_i , como se puede visualizar en la figura A.2.

El efecto tiene forma geométrica de un triángulo equilátero y se tienen las dimensiones de los lados (e), por ende, se puede determinar la magnitud del desplazamiento de las esferas $J_i J'_i$ de la siguiente manera:

$$J_1 J'_1 = J_2 J'_2 = J_3 J'_3 = \frac{e}{2} \tan 30^\circ = \frac{e}{2\sqrt{3}}$$

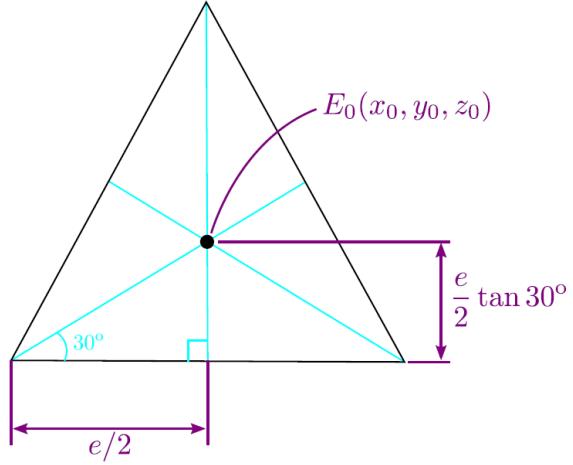


Figura A.3: Vista superior del efecto final

Similar a como se calcula el desplazamiento de las esferas $J_i J'_i$, se determina la magnitud de la distancia entre el centroide de la base fija y la posición de los actuadores OF_i :

$$OF_1 = OF_2 = OF_3 = \frac{f}{2} \tan 30^\circ = \frac{f}{2\sqrt{3}}$$

La distancia perpendicular entre el plano que contiene la base fija y el punto donde se encuentran las juntas esféricas J_i que unen los brazos con sus antebrazos, está en función de cada ángulo de los actuadores $i = 1, 2, 3$, dada por la siguiente ecuación:

$$F_i J_i = r_f \cos(\theta_i), \forall i = 1, 2, 3$$

Por lo tanto, los centros de las esferas de radio r_e que se intersectan en el centroide $E_0(x_0, y_0, z_0)$ del efecto son:

Centros esferas

$$(x_1, y_1, z_1) = J'_1 \left(0, \left[-\frac{f-e}{2\sqrt{3}} - r_f \cos(\theta_1) \right], -r_f \sin(\theta_1) \right)$$

$$(x_2, y_2, z_2) = J'_2 \left(\left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_2) \right] \cos(30^\circ), \left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_2) \right] \sin(30^\circ), -r_f \sin(\theta_2) \right)$$

$$(x_3, y_3, z_3) = J'_3 \left(-\left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_3) \right] \cos(30^\circ), \left[\frac{f-e}{2\sqrt{3}} + r_f \cos(\theta_3) \right] \sin(30^\circ), -r_f \sin(\theta_3) \right)$$

Tabla A.1: Centro de esferas J'_i que representan la translación de las juntas J_i

A consecuencia de las extensas representaciones de los centros parametrizados, se resumen las ecuaciones cartesianas de las esferas de la siguiente forma:

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = r_e^2 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_e^2 \\ (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = r_e^2 \end{cases}$$

Los centros (x_1, y_1, z_1) , (x_2, y_2, z_2) , (x_3, y_3, z_3) se muestran en la tabla A.1.

Ampliando las ecuaciones anteriores:

$$\begin{cases} x^2 + y^2 + z^2 - 2x_1x - 2y_1y - 2z_1z = r_e^2 - x_1^2 - y_1^2 - z_1^2 \\ x^2 + y^2 + z^2 - 2x_2x - 2y_2y - 2z_2z = r_e^2 - x_2^2 - y_2^2 - z_2^2 \\ x^2 + y^2 + z^2 - 2x_3x - 2y_3y - 2z_3z = r_e^2 - x_3^2 - y_3^2 - z_3^2 \end{cases}$$

Se crean nuevas constantes $w_{i \in \{1,2,3\}}$ con el objetivo de simplificar las ecuaciones anteriores:

$$w_i = x_i^2 + y_i^2 + z_i^2$$

Reemplazando w_i en las ecuaciones:

$$\begin{cases} x^2 + y^2 + z^2 - 2x_1x - 2y_1y - 2z_1z = r_e^2 - w_1 & (\text{A.1a}) \\ x^2 + y^2 + z^2 - 2x_2x - 2y_2y - 2z_2z = r_e^2 - w_2 & (\text{A.1b}) \\ x^2 + y^2 + z^2 - 2x_3x - 2y_3y - 2z_3z = r_e^2 - w_3 & (\text{A.1c}) \end{cases}$$

Restando las ecuaciones (A.1a)-(A.1b), (A.1a)-(A.1c), (A.1b)-(A.1c):

$$\begin{cases} (x_1 - x_2)x + (y_1 - y_2)y + (z_1 - z_2)z = \frac{(w_1 - w_2)}{2} & (\text{A.2a}) \\ (x_1 - x_3)x + (y_1 - y_3)y + (z_1 - z_3)z = \frac{(w_1 - w_3)}{2} & (\text{A.2b}) \\ (x_2 - x_3)x + (y_2 - y_3)y + (z_2 - z_3)z = \frac{(w_2 - w_3)}{2} & (\text{A.2c}) \end{cases}$$

Los sistemas de ecuaciones no lineales son complejos, por lo que se proponen los siguientes pasos para encontrar la solución (x, y, z) .

1. Se despeja x de una de las ecuaciones (A.2a), (A.2b) o (A.2c) en función de z , es decir, $x(z)$.
2. Se despeja y de una de las ecuaciones (A.2a), (A.2b) o (A.2c) (no elegida anteriormente) en función de z , es decir, $y(z)$.
3. Con las 2 ecuaciones encontradas en los pasos 1 y 2 en función de z , se reemplazan en (A.1a) para encontrar la solución de z , en consecuencia se puede obtener finalmente la solución (x, y, z) .

Paso 1:

Multiplicando (A.2b) * $^{(y_2-y_1)}_{(y_1-y_3)}$:

$$\frac{(y_2 - y_1)}{(y_1 - y_3)} (x_1 - x_3) x + (y_2 - y_1) y + \frac{(y_2 - y_1)}{(y_1 - y_3)} (z_1 - z_3) z = \frac{(y_2 - y_1)}{(y_1 - y_3)} \frac{(w_1 - w_3)}{2} \quad (\text{A.3})$$

Sumando (A.2a)+ (A.3) para eliminar y :

$$\left[\frac{(y_2 - y_1)}{(y_1 - y_3)} (x_1 - x_3) + (x_1 - x_2) \right] x + \left[\frac{(y_2 - y_1)}{(y_1 - y_3)} (z_1 - z_3) + (z_1 - z_2) \right] z = \frac{(y_2 - y_1)}{(y_1 - y_3)} \frac{(w_1 - w_3)}{2} + \frac{(w_1 - w_2)}{2} \quad (\text{A.4})$$

Despejando x en función de z en la ecuación (A.4) para obtener $x = a_1 z + b_1$:

$$\begin{aligned} & \left[\frac{(y_2 - y_1)}{(y_1 - y_3)} (x_1 - x_3) + (x_1 - x_2) \right] x = \\ & - \left[\frac{(y_2 - y_1)}{(y_1 - y_3)} (z_1 - z_3) + (z_1 - z_2) \right] z + \frac{(y_2 - y_1)}{(y_1 - y_3)} \frac{(w_1 - w_3)}{2} + \frac{(w_1 - w_2)}{2} \end{aligned} \quad (\text{A.5})$$

Multiplicando (A.5) * $(y_1 - y_3)$:

$$\begin{aligned} & [(y_2 - y_1) (x_1 - x_3) + (y_1 - y_3) (x_1 - x_2)] x = \\ & - [(y_2 - y_1) (z_1 - z_3) + (y_1 - y_3) (z_1 - z_2)] z + \\ & (y_2 - y_1) \frac{(w_1 - w_3)}{2} + (y_1 - y_3) \frac{(w_1 - w_2)}{2} \end{aligned} \quad (\text{A.6})$$

Multiplicando (A.6) * $\frac{1}{[(y_2 - y_1)(x_1 - x_3) + (y_1 - y_3)(x_1 - x_2)]}$

$$\begin{aligned} x = & \left[\frac{-1}{[(y_2 - y_1)(x_1 - x_3) + (y_1 - y_3)(x_1 - x_2)]} * [(y_2 - y_1)(z_1 - z_3) + (y_1 - y_3)(z_1 - z_2)] \right] z \\ & + \left[\left(\frac{1}{[(y_2 - y_1)(x_1 - x_3) + (y_1 - y_3)(x_1 - x_2)]} \right) \right. \\ & \left. * \left((y_2 - y_1) \frac{(w_1 - w_3)}{2} + (y_1 - y_3) \frac{(w_1 - w_2)}{2} \right) \right] \end{aligned} \quad (\text{A.7})$$

Creando d para simplificar las ecuaciones:

$$\begin{aligned} d &= - [(y_2 - y_1) (x_1 - x_3) + (y_1 - y_3) (x_1 - x_2)] \\ \implies d &= - [(y_2 x_1 - y_1 x_1 - y_2 x_3 + y_1 x_3) + (y_1 x_1 - y_3 x_1 - y_1 x_2 + y_3 x_2)] \\ \implies d &= - [y_2 x_1 - y_3 x_1 - y_2 x_3 + y_1 x_3 + -y_1 x_2 + y_3 x_2] \\ \implies d &= - [(y_2 - y_3) x_1 + (y_3 - y_1) x_2 + (y_1 - y_2) x_3] \\ \implies d &= - [(y_2 - y_3) x_1 + (y_3 - y_1) x_2 + (y_1 - y_2) x_3] \end{aligned}$$

$$d = (y_2 - y_1) x_3 - (y_3 - y_1) x_2 - (y_2 - y_3) x_1 \quad (\text{A.8})$$

Agrupando términos de la siguiente manera:

$$x = a_1 z + b_1$$

Donde a_1 :

$$\begin{aligned} a_1 &= \frac{(y_2 - y_1)(z_1 - z_3) + (y_1 - y_3)(z_1 - z_2)}{d} \\ \implies a_1 &= \frac{(z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1)}{d} \end{aligned} \quad (\text{A.9})$$

Donde b_1 :

$$\begin{aligned} b_1 &= \left(\frac{1}{[(y_2 - y_1)(x_1 - x_3) + (y_1 - y_3)(x_1 - x_2)]} \right) \\ &\quad * \left((y_2 - y_1) \frac{(w_1 - w_3)}{2} + (y_1 - y_3) \frac{(w_1 - w_2)}{2} \right) \\ \implies b_1 &= \left(\frac{1}{2 * (-d)} \right) * [(w_2 - w_1)(y_3 - y_1) - (w_3 - w_1)(y_2 - y_1)] \end{aligned} \quad (\text{A.10})$$

Paso 2:

Multiplicado (A.2b) * $\frac{(x_2 - x_1)}{(x_1 - x_3)}$:

$$(x_2 - x_1)x + \frac{(x_2 - x_1)}{(x_1 - x_3)}(y_1 - y_3)y + \frac{(x_2 - x_1)}{(x_1 - x_3)}(z_1 - z_3)z = \frac{(x_2 - x_1)}{(x_1 - x_3)} \frac{(w_1 - w_3)}{2} \quad (\text{A.11})$$

Sumando (A.2a) + (A.11) para eliminar y :

$$\begin{aligned} \left[\frac{(x_2 - x_1)}{(x_1 - x_3)}(y_1 - y_3) + (y_1 - y_2) \right] y + \left[\frac{(x_2 - x_1)}{(x_1 - x_3)}(z_1 - z_3) + (z_1 - z_2) \right] z &= \\ \left[\frac{(x_2 - x_1)}{(x_1 - x_3)} \frac{(w_1 - w_3)}{2} + \frac{(w_1 - w_2)}{2} \right] \end{aligned} \quad (\text{A.12})$$

Despejando y en función de z en la ecuación (A.12) obteniendo $y = a_2 z + b_2$:

$$\begin{aligned} & \left[\frac{(x_2 - x_1)}{(x_1 - x_3)} (y_1 - y_3) + (y_1 - y_2) \right] y = \\ & - \left[\frac{(x_2 - x_1)}{(x_1 - x_3)} (z_1 - z_3) + (z_1 - z_2) \right] z + \left[\frac{(x_2 - x_1)}{(x_1 - x_3)} \frac{(w_1 - w_3)}{2} + \frac{(w_1 - w_2)}{2} \right] \end{aligned} \quad (\text{A.13})$$

Multiplicando (A.13) * $(x_1 - x_3)$

$$\begin{aligned} & [(x_2 - x_1)(y_1 - y_3) + (x_1 - x_3)(y_1 - y_2)] y = \\ & - [(x_2 - x_1)(z_1 - z_3) + (x_1 - x_3)(z_1 - z_2)] z + \\ & \left[(x_2 - x_1) \frac{(w_1 - w_3)}{2} + (x_1 - x_3) \frac{(w_1 - w_2)}{2} \right] \end{aligned} \quad (\text{A.14})$$

Multiplicando (A.14) * $\frac{1}{[(x_2 - x_1)(y_1 - y_3) + (x_1 - x_3)(y_1 - y_2)]}$ = (A.14) * $\frac{1}{d}$

$$y = \frac{-[(x_2 - x_1)(z_1 - z_3) + (x_1 - x_3)(z_1 - z_2)]}{d} z + \left[(x_2 - x_1) \frac{(w_1 - w_3)}{2d} + (x_1 - x_3) \frac{(w_1 - w_2)}{2d} \right] \quad (\text{A.15})$$

Agrupando términos de la siguiente manera:

$$y = a_2 z + b_2$$

Donde a_2 :

$$\begin{aligned} a_2 &= \frac{-[(x_2 - x_1)(z_1 - z_3) + (x_1 - x_3)(z_1 - z_2)]}{d} \\ \implies a_2 &= \frac{-1}{d} * [(z_2 - z_1)x_3 - (z_3 - z_1)x_2 + (z_3 - z_2)x_1] \end{aligned} \quad (\text{A.16})$$

Donde b_2 :

$$\begin{aligned} b_2 &= \left[(x_2 - x_1) \frac{(w_1 - w_3)}{2d} + (x_1 - x_3) \frac{(w_1 - w_2)}{2d} \right] \\ \implies b_2 &= \frac{1}{2d} * [(x_1 - x_3)(w_1 - w_2) + (x_2 - x_1)(w_1 - w_3)] \\ \implies b_2 &= \frac{1}{2d} * [(x_1w_1 - x_3w_1 - x_1w_2 + x_3w_2) + (x_2w_1 - x_1w_1 - x_2w_3 + x_1w_3)] \\ \implies b_2 &= \frac{1}{2d} * [(-x_3w_1 + x_3w_2) + (x_2w_1 - x_2w_3) + (x_1w_3 - x_1w_2)] \end{aligned}$$

$$b_2 = \frac{1}{2d} * [(w_2 - w_1)x_3 - (w_3 - w_1)x_2 + (w_3 - w_2)x_1] \quad (\text{A.17})$$

Paso 3:

Sustituyendo $x = a_1z + b_1$ y $y = a_2z + b_2$ en (A.1a):

$$(a_1z + b_1)^2 + (a_2z + b_2)^2 + z^2 - 2x_1(a_1z + b_1) - 2y_1(a_2z + b_2) - 2z_1z = r_e^2 - x_1^2 - y_1^2 - z_1^2$$

Reordenando z como una función cuadrática:

$$\begin{aligned} &\implies (a_1^2z^2 + 2a_1b_1z + b_1^2) + (a_2^2z^2 + 2a_2b_2z + b_2^2) + z^2 - \\ &2x_1(a_1z + b_1) - 2y_1(a_2z + b_2) - 2z_1z = r_e^2 - x_1^2 - y_1^2 - z_1^2 \\ &\implies (a_1^2 + a_2^2 + 1)z^2 + (2a_1b_1 + 2a_2b_2 - 2x_1a_1 - 2y_1a_2 - 2z_1)z + \\ &(b_1^2 + b_2^2 - 2x_1b_1 - 2y_1b_2) = r_e^2 - x_1^2 - y_1^2 - z_1^2 \\ &\implies (a_1^2 + a_2^2 + 1)z^2 + (2a_1b_1 + 2a_2b_2 - 2x_1a_1 - 2y_1a_2 - 2z_1)z + \\ &(b_1^2 + b_2^2 - 2x_1b_1 - 2y_1b_2 - r_e^2 + x_1^2 + y_1^2 + z_1^2) = 0 \\ &\implies (a_1^2 + a_2^2 + 1)z^2 + 2(a_1(b_1 - x_1) + a_2(b_2 - y_1) - z_1)z + \\ &(b_1^2 + b_2^2 - 2y_1b_2 + y_1^2 - 2x_1b_1 + x_1^2 + z_1^2 - r_e^2) = 0 \\ &\implies (a_1^2 + a_2^2 + 1)z^2 + 2(a_1(b_1 - x_1) + a_2(b_2 - y_1) - z_1)z + \\ &((b_1 - x_1)^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2) = 0 \end{aligned}$$

Agregando nuevas constantes para simplificar la expresión anterior:

$$A * z^2 + B * z + C = 0$$

Donde:

$$\begin{aligned} A &= a_1^2 + a_2^2 + 1 \\ B &= 2[a_1(b_1 - x_1) + a_2(b_2 - y_1) - z_1] \\ C &= (b_1 - x_1)^2 + (b_2 - y_1)^2 + z_1^2 - r_e^2 \end{aligned}$$

Resolviendo la ecuación de segundo grado se obtiene:

$$z = \frac{-B \pm \sqrt{B^2 - 4AC}}{2 * A}$$

Se puede apreciar que z tiene dos soluciones, pero solo una es físicamente posible (la raíz de la ecuación que tenga coordenada z mayor valor negativa), $z = \frac{-B - \sqrt{B^2 - 4AC}}{2 * A}$. Por lo tanto, la solución del sistema de ecuaciones es:

$$(x, y, z) = (a_1z + b_1, a_2z + b_2, z) = \left(a_1z + b_1, a_2z + b_2, \frac{-B - \sqrt{B^2 - 4AC}}{2 * A} \right)$$

A.1.1.2. Cinemática inversa

El objetivo de la cinemática inversa es encontrar los ángulos de los actuadores del robot delta conociendo la posición del centroide de su efecto final:

$$E_0 (x_0, y_0, z_0) \rightarrow (\theta_1, \theta_2, \theta_3)$$

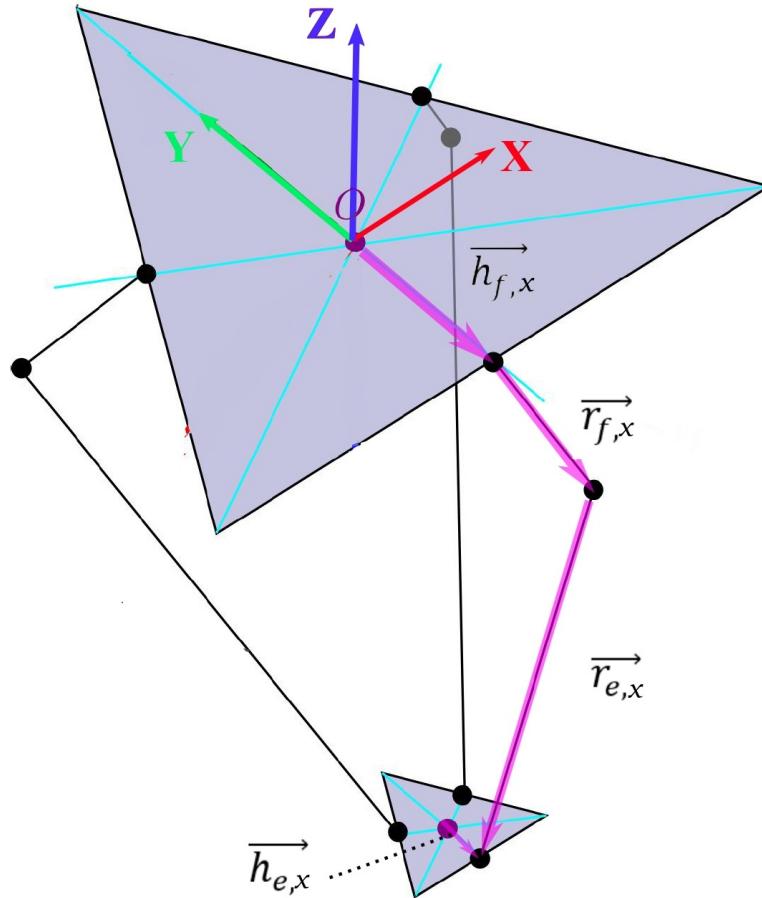


Figura A.4: Vectorización de la cinemática inversa del método A

Se tiene conocimiento de las posiciones de todos los puntos sobre el efecto, a causa de que se sabe posición del centroide del efecto final $E_0 (x_0, y_0, z_0)$, la base fija es paralela al efecto y además estas dos últimas partes mecánicas mencionadas tienen la misma orientación. Por lo tanto, se conocen las coordenadas de las juntas esféricas E_i que unen los antebrazos con el efecto y a su vez los vectores $\overrightarrow{h_{e,x}}$ que representan la posición de las juntas esféricas E_i con respecto al centroide del efecto.

Con respecto a los vectores $\overrightarrow{r_{e,x}}$, que representan los antebrazos, solo se conoce el punto final E_i , que coincide con las posiciones de cada junta esférica unida al efecto. Entonces, la orientación de cada vector $\overrightarrow{r_{e,x}}$, en cada uno de las 3 cadenas, está restringida por esferas con centros en las juntas esféricas E_i con radio equivalente al largo del antebrazo r_e .

En cuanto a los vectores $\vec{r}_{f,x}$ solo se conocen sus puntos iniciales F_i , que coinciden con la posición de cada actuador $i \in \{1, 2, 3\}$, sobre la base fija. Por otro lado, los actuadores son simplificados como juntas revolutas, por ende, restringen a cada uno de los vectores $\vec{r}_{f,x}$ a moverse en un plano que contiene un punto en la posición del actuador F_i y un vector $\vec{h}_{f,x}$. Las restricciones anteriores traen como consecuencia que las coordenadas de los puntos finales de vector $\vec{r}_{f,x}$, es decir su extremo, estén sobre circunferencias. Como se aprecia en la figura A.4.

Finalmente, para obtener los ángulos de los actuadores $(\theta_1, \theta_2, \theta_3)$, se calcula las intersecciones entre las circunferencias formadas por el punto final de $\vec{r}_{f,x}$ (con centro en F_i y de radio r_f) y la esfera formada por el vector $\vec{r}_{e,x}$ (con centro en E_i y de radio r_e) para cada cadena cinemática $i \in \{1, 2, 3\}$. Esta intersección no es más que la junta esférica J_i , recordando que la función de estas es unir los brazos con los antebrazos. Una vez obtenida la posición de las 3 juntas J_i , con álgebra simple se pueden obtener los valores de los ángulos $(\theta_1, \theta_2, \theta_3)$.

Como antes se ha mencionado, el primer paso para encontrar la configuración del espacio articular $(\theta_1, \theta_2, \theta_3)$ es calcular las posiciones de las juntas J_i . Se empieza por la junta J_1 que esta sobre el plano YZ de nuestro sistema de referencia local. Se puede captar en la figura A.5 que la intersección de la esfera y el plano YZ es una circunferencia con centro en el punto E'_1 y de radio $\overline{E'_1 J_1}$. El punto E'_1 es la proyección de E_1 en el plano YZ .

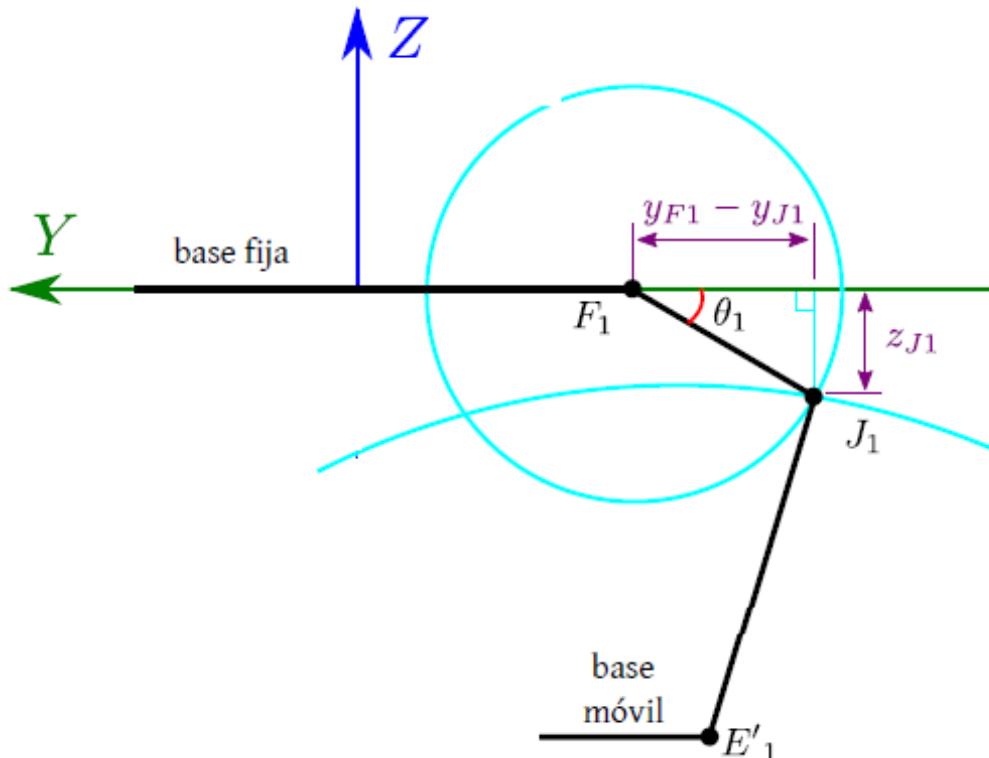


Figura A.5: Visualización plano YZ para representar la intersección de los 2 círculos en el punto J_1 para la solución de la cinemática inversa en el método A

El punto E_1 que se usa para determinar el centro de la circunferencia E'_1 gracias a su proyección en el plano YZ , se calcula a partir de geometría simple de un triángulo equilátero de lado e y de posición del centroide $E_0(x_0, y_0, z_0)$ equivalente a la del efector final:

$$E_0E_1 = \frac{e}{2}\tan 30 = \frac{e}{2\sqrt{3}}$$

$$E_1 \left(x_0, y_0 - \frac{e}{2\sqrt{3}}, z_0 \right) \Rightarrow E'_1 \left(0, y_0 - \frac{e}{2\sqrt{3}}, z_0 \right)$$

El radio $\overline{E'_1J_1}$ de la circunferencia formada por la proyección del antebrazo en el plano YZ , se puede calcular por medio del teorema de pitagoras donde:

$$E_1E'_1 = x_0$$

$$\Rightarrow E'_1J_1 = \sqrt{E_1J_1^2 - E_1E'_1^2} = \sqrt{r_e^2 - x_0^2}$$

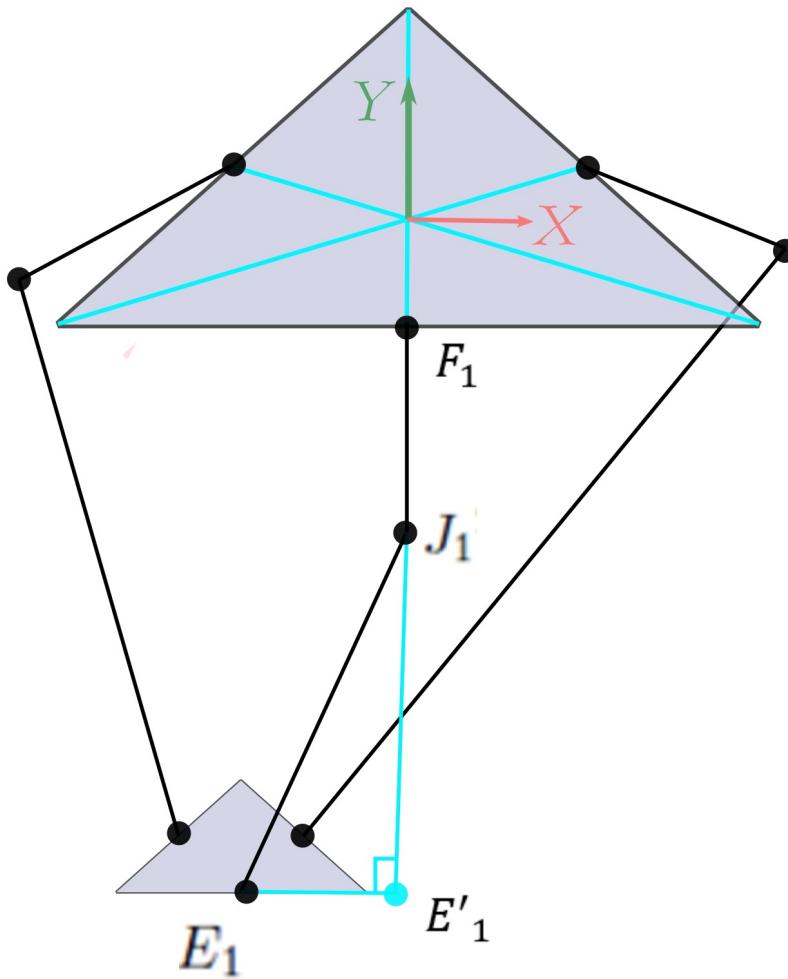


Figura A.6: Proyección del punto E_1 sobre el plano YZ

Resumiendo todo lo anterior, los centros de las circunferencias que intersectan la junta esférica J_1 son:

Centros esferas	Radio
$(y_1, z_1) = F_1 (y_{F_1} z_{F_1}) = \left(-\frac{f}{2\sqrt{3}}, 0\right)$	r_f
$(y_2, z_2) = E'_1 (y_{E'_1} z_{E'_1}) = (y_0 - \frac{e}{2\sqrt{3}}, z_0)$	$r_2 = \sqrt{r_e^2 - x_o^2}$

Tabla A.2: Centro de esferas que representa las juntas en los puntos E'_i y F_i

Ecuaciones cartesianas de las circunferencias son:

$$\begin{cases} (y - y_1)^2 + (z - z_1)^2 = r_f^2 \\ (y - y_2)^2 + (z - z_2)^2 = r_e^2 - x_o^2 \end{cases}$$

Desarrollando las ecuaciones anteriores y reordenando:

$$\begin{cases} y^2 + z^2 - 2yy_1 - 2zz_1 = r_f^2 - y_1^2 - z_1^2 & (A.18a) \\ y^2 + z^2 - 2yy_2 - 2zz_2 = r_e^2 - x_o^2 - y_2^2 - z_2^2 & (A.18b) \end{cases}$$

Restando (A.18a)-(A.18b):

$$-2y(y_1 - y_2) - 2z(z_1 - z_2) = r_f^2 - y_1^2 - z_1^2 - r_e^2 + x_o^2 + y_2^2 + z_2^2$$

Despejando z en función de y :

$$z = \frac{-(y_1 - y_2)}{(z_1 - z_2)} * y + \frac{r_f^2 - y_1^2 - z_1^2 - r_e^2 + x_o^2 + y_2^2 + z_2^2}{-2(z_1 - z_2)}$$

Donde:

$$z = a + by \quad (A.19)$$

$$b = \frac{-(y_1 - y_2)}{(z_1 - z_2)}$$

$$a = \frac{r_f^2 - y_1^2 - z_1^2 - r_e^2 + x_o^2 + y_2^2 + z_2^2}{-2(z_1 - z_2)}$$

Reemplazando $z_1 = 0$, $y_2 = y_0 - \frac{e}{2\sqrt{3}}$, $y_1 = -\frac{f}{2\sqrt{3}}$:

$$\begin{aligned} b &= \frac{(y_1 - y_2)}{z_2} \\ a &= \frac{x_o^2 + y_2^2 + z_0^2 + r_f^2 - r_e^2 - y_1^2}{2z_0} \\ \implies a &= \frac{x_o^2 + \left(y_0 - \frac{e}{2\sqrt{3}}\right)^2 + z_0^2 + r_f^2 - r_e^2 - \left(-\frac{f}{2\sqrt{3}}\right)^2}{2z_0} \end{aligned}$$

Reemplazando la ecuación (A.19) en (A.18a):

$$(y - y_1)^2 + [(a + by) - z_1]^2 = r_f^2$$

Desarrollando la ecuación anterior:

$$\begin{aligned} (y^2 - 2yy_1 + y_1^2) + (b^2y^2 + 2aby + a^2) - 2(a + by)z_1 + z_1^2 &= r_f^2 \\ \implies (b^2 + 1)y^2 + (-2y_1 + 2ab)y + (y_1^2 + a^2 - r_f^2) + z_1(-2(a + by) + z_1) &= 0 \quad (\text{A.20}) \end{aligned}$$

La solución y para la ecuación cuadrática (A.20) está dada por:

$$y = \frac{-(2ab - 2y_1) \pm \sqrt{d}}{2(b^2 + 1)}$$

Se aprecian dos soluciones, donde se elige la $y = \frac{-(2ab - 2y_1) - \sqrt{d}}{2(b^2 + 1)}$ para que el ángulo entre el brazo y el antebrazo no exceda los 180° .

El discriminante d es:

$$d = (2ab - 2y_1)^2 - 4 * (b^2 + 1) * \left[(y_1^2 + a^2 - r_f^2) + z_1 * (-2(a + by) + z_1) \right]$$

Reemplazando $z_1 = 0$ y reordenando:

$$\begin{aligned} d &= (2ab - 2y_1)^2 - 4 * (b^2 + 1) * (y_1^2 + a^2 - r_f^2) \\ \implies d &= 4 \left[(ab - y_1)^2 - (b^2 + 1) * (y_1^2 + a^2 - r_f^2) \right] \\ \implies d &= 4 \left[(a^2b^2 - 2aby_1 + y_1^2) + (-b^2y_1^2 - b^2a^2 + b^2r_f^2) - (y_1^2 + a^2 - r_f^2) \right] \\ \implies d &= 4 \left[-a^2 - 2aby_1 - b^2y_1^2 + b^2r_f^2 + r_f^2 \right] \\ \implies d &= 4 \left[-(a + by_1)^2 + (b^2 + 1)r_f^2 \right] \end{aligned}$$

Por lo tanto, la solución y para el sistema de ecuaciones de las circunferencias es:

$$y = \frac{-(2ab - 2y_1) - \sqrt{4[-(a + by_1)^2 + (b^2 + 1)r_f^2]}}{2(b^2 + 1)}$$

$$\implies y = \frac{(y_1 - ab) - \sqrt{[-(a + by_1)^2 + (b^2 + 1)r_f^2]}}{(b^2 + 1)}$$

Por lo tanto, la solución del sistema de ecuaciones es:

$$J_1 = (x_{J_1}, y_{J_1}, z_{J_1}) = (0, y, a + by)$$

$$J_1 = \left(0, \frac{(y_1 - ab) - \sqrt{[(a + by_1)^2 + (b^2 + 1)r_f^2]}}{(b^2 + 1)}, a + b \frac{(y_1 - ab) - \sqrt{[(a + by_1)^2 + (b^2 + 1)r_f^2]}}{(b^2 + 1)} \right)$$

Finalmente, se determina el ángulo θ_1 por medio del triángulo rectángulo formado en el brazo y la proyección del mismo en el plano XY , como se ilustra en la figura A.5:

$$\theta_1 = \arctan\left(\frac{z_{J_1}}{y_{F_1} - y_{J_1}}\right)$$

Donde y_{F_1} se obtiene de la geometría básica que tiene la base fija (figura A.7):

$$F_1(0, -\frac{f}{2\sqrt{3}}, 0)$$

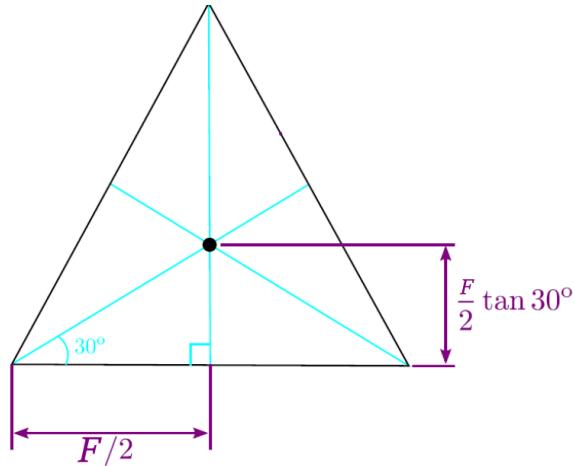


Figura A.7: Vista superior de la base fija

Aprovechando la simetría del robot delta, se utiliza el mismo método empleado en la solución de θ_1 para los ángulos θ_2 y θ_3 . Para calcularlos, se utilizan matrices de rotación con un ángulo de rotación de 120° para la cadena cinemática con el actuador 2 y de 240° para la cadena cinemática con el actuador 3. Estas matrices giran el sistema de referencia local en 120° y 240° grados como se observa en la figura A.8:

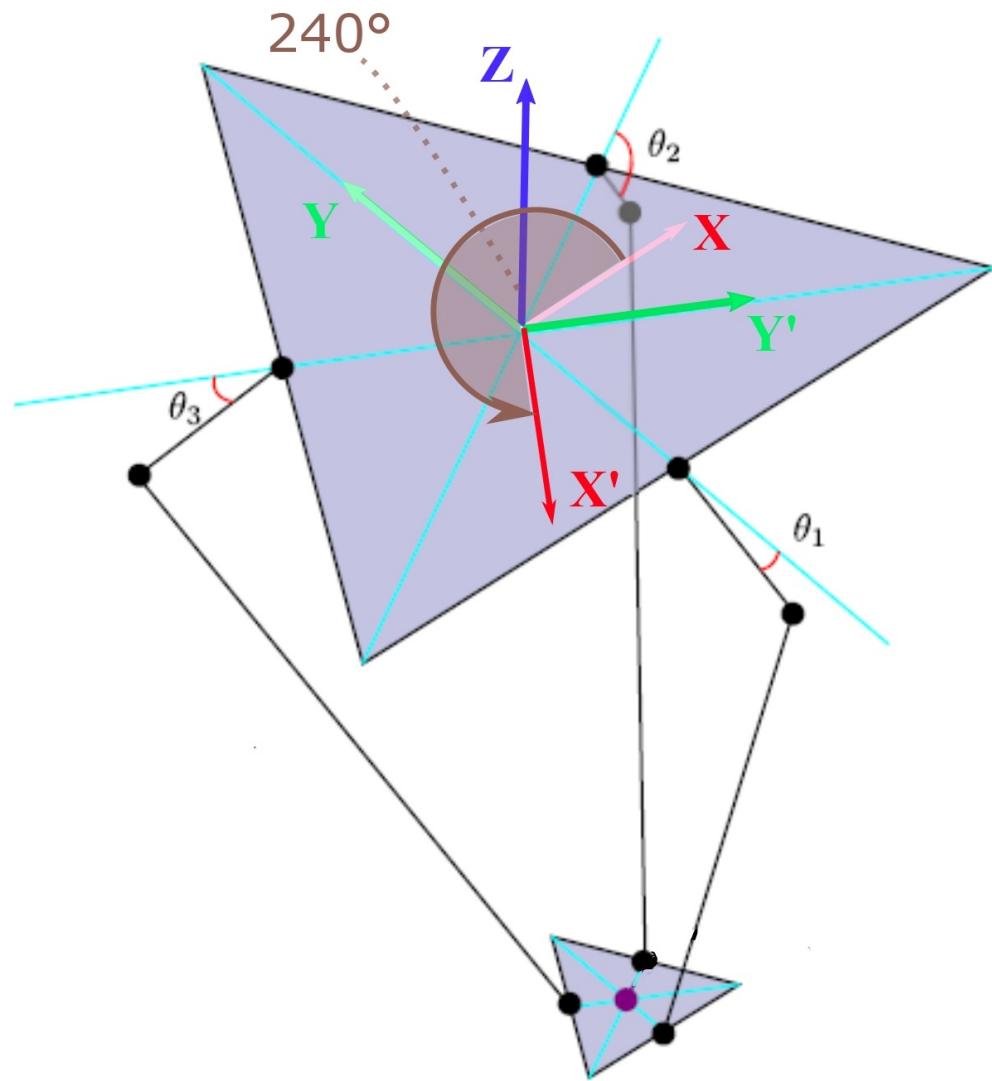


Figura A.8: Representación de la rotación del sistema coordenado XYZ en 240° para la solución de la cinemática inversa en el método A

A.1.2. Modelación cinemática de velocidad

A.1.2.1. Vectorización y ángulos de juntas

La figura 4.11 define los ángulos de articulación asociados con la extremidad i , donde \vec{p} es el vector de posición del centroide de la plataforma móvil, θ_{1i} se mide desde el eje x_i hasta $\overrightarrow{A_iB_i}$, θ_{2i} se define desde la línea extendida de $\overrightarrow{A_iB_i}$ hasta la línea definida por la intersección del plano del paralelogramo y el plano $x_i - z_i$ y θ_{3i} se mide desde la dirección y_i hasta $\overrightarrow{B_iC_i}$. En general, hay 9 ángulos articulares, θ_{1i} , θ_{2i} y θ_{3i} para $i \in \{1, 2, 3\}$, asociados con el manipulador. Los ángulos de los actuadores son θ_{11}, θ_{21} y θ_{31} .

Se puede escribir una ecuación de cierre de bucle para cada extremidad vectorialmente:

$$\overrightarrow{A_iB_i} + \overrightarrow{B_iC_i} = \overrightarrow{OP} + \overrightarrow{PC_i} - \overrightarrow{OA_i} \quad (\text{A.21})$$

Reescribiendo A.21 en el marco de coordenadas $A_i - x_iy_iz_i$ conduce a la siguiente representación matricial:

$$a \begin{bmatrix} \cos \theta_{1i} \\ 0 \\ \sin \theta_{1i} \end{bmatrix} + b \begin{bmatrix} \sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i}) \end{bmatrix} = \begin{bmatrix} c_{xi} \\ c_{yi} \\ c_{zi} \end{bmatrix} \quad (\text{A.22})$$

$$\begin{bmatrix} c_{xi} \\ c_{yi} \\ c_{zi} \end{bmatrix} = \begin{bmatrix} \cos \phi_i & \sin \phi_i & 0 \\ -\sin \phi_i & \cos \phi_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} h \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix} \quad (\text{A.23})$$

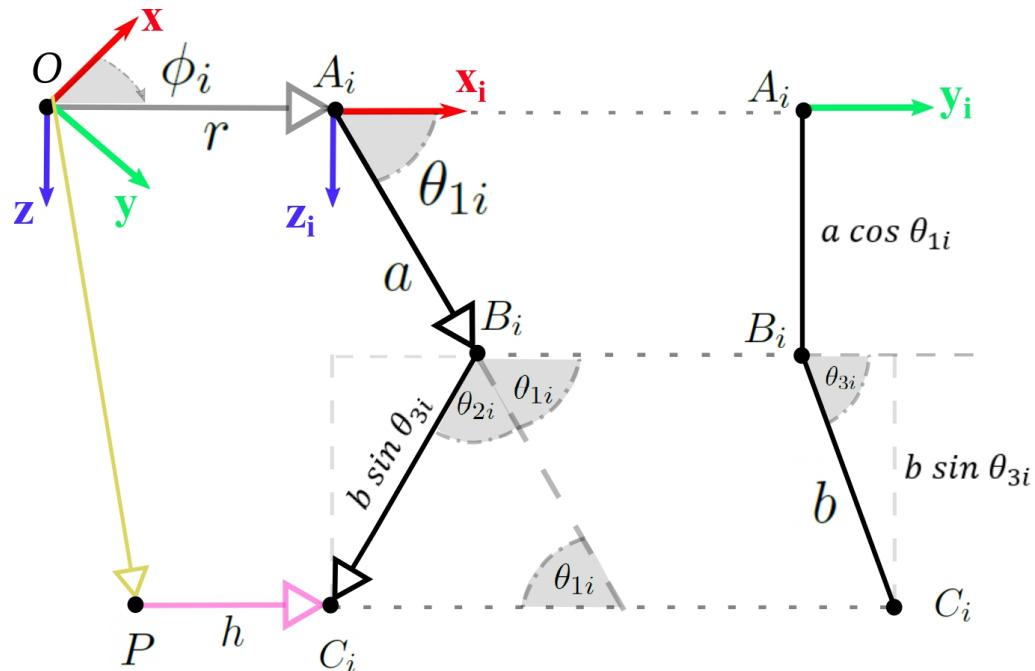


Figura A.9: Vectorización y ángulos interiores para la solución de la cinemática de velocidad del método A

La figura A.9 denota la posición del punto C_i en relación con el marco de coordenadas $A_i - x_i y_i z_i$, a y b son las longitudes de los enlaces $\overrightarrow{A_i B_i}$, y $\overrightarrow{B_i C_i}$ respectivamente y $\vec{p} = [p_x, p_y, p_z]^T$ es el vector de posición del punto P relativo al sistema de coordenadas $O - xyz$.

El ángulo θ_{3i} se calcula resolviendo la segunda fila de la ecuación A.22:

$$\theta_{3i} = \cos^{-1} \frac{c_{yi}}{b} \quad (\text{A.24})$$

Con θ_{3i} determinado, se genera una ecuación con θ_{2i} como la única incógnita, sumando los cuadrados de c_{xi} , c_{yi} y c_{zi} en la ecuación A.22 aplicando la norma a los vectores:

$$\begin{aligned} & \left\| a \begin{bmatrix} \cos \theta_{1i} \\ 0 \\ \sin \theta_{1i} \end{bmatrix} + b \begin{bmatrix} \sin \theta_{3i} \cos(\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin(\theta_{1i} + \theta_{2i}) \end{bmatrix} \right\| = \left\| \begin{bmatrix} c_{xi} \\ c_{yi} \\ c_{zi} \end{bmatrix} \right\| \\ \Rightarrow & (a \cos \theta_{1i} + b \sin \theta_{3i} \cos(\theta_{1i} + \theta_{2i}))^2 + \\ & (b \cos \theta_{3i})^2 + (a \sin \theta_{1i} + b \sin \theta_{3i} \sin(\theta_{1i} + \theta_{2i}))^2 = c_{xi}^2 + c_{yi}^2 + c_{zi}^2 \\ \Rightarrow & [a^2 \cos^2 \theta_{1i} + ab \cos \theta_{1i} \sin \theta_{3i} \cos(\theta_{1i} + \theta_{2i}) + b^2 \sin^2 \theta_{3i} \cos^2(\theta_{1i} + \theta_{2i})] + \\ & [b^2 \cos^2 \theta_{3i}] + [a^2 \sin^2 \theta_{1i} + 2ab \sin \theta_{1i} \sin \theta_{3i} \sin(\theta_{1i} + \theta_{2i}) + b^2 \sin^2 \theta_{3i} \sin^2(\theta_{1i} + \theta_{2i})] = \\ & c_{xi}^2 + c_{yi}^2 + c_{zi}^2 \\ \Rightarrow & a^2 (\cos^2 \theta_{1i} + \sin^2 \theta_{1i}) + b^2 \sin^2 \theta_{3i} (\cos^2(\theta_{1i} + \theta_{2i}) + \sin^2(\theta_{1i} + \theta_{2i})) + \\ & 2ab \sin \theta_{3i} (\cos \theta_{1i} \cos(\theta_{1i} + \theta_{2i}) + \sin \theta_{1i} \sin(\theta_{1i} + \theta_{2i})) + b^2 \cos^2 \theta_{3i} = c_{xi}^2 + c_{yi}^2 + c_{zi}^2 \end{aligned}$$

Utilizando las siguientes propiedades trigonométricas:

$$\begin{aligned} \cos^2 \theta_{1i} + \sin^2 \theta_{1i} &= 1 \\ \cos^2(\theta_{1i} + \theta_{2i}) + \sin^2(\theta_{1i} + \theta_{2i}) &= 1 \\ \cos \theta_{1i} \cos(\theta_{1i} + \theta_{2i}) + \sin \theta_{1i} \sin(\theta_{1i} + \theta_{2i}) &= \cos(\theta_{1i} - (\theta_{1i} + \theta_{2i})) = \cos(\theta_{2i}) \end{aligned}$$

Reemplazando:

$$\Rightarrow a^2 + b^2 \sin^2 \theta_{3i} + 2ab \sin \theta_{3i} \cos \theta_{2i} + b^2 \cos^2 \theta_{3i} = c_{xi}^2 + c_{yi}^2 + c_{zi}^2$$

Donde se sabe que:

$$(\sin^2 \theta_{3i} + \cos^2 \theta_{3i}) = 1$$

Entonces:

$$a^2 + b^2 + 2ab \sin \theta_{3i} \cos \theta_{2i} + b^2 = c_{xi}^2 + c_{yi}^2 + c_{zi}^2$$

Despejando θ_{2i} :

$$\theta_{2i} = \cos^{-1} \left(\frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2ab \sin \theta_{3i}} \right) \quad (\text{A.25})$$

Con θ_{3i} y θ_{2i} determinados, solo queda un par de ecuaciones con θ_{1i} como la única incógnita que desde la ecuación A.22. Por lo tanto, es posible resolver el valor de θ_{1i} .

A.1.2.2. Jacobiano

Con referencia a la figura A.9, una ecuación de cierre de bucle para una extremidad i sistema de referencia $A_i - x_i y_i z_i$ se puede escribir como:

$$\overrightarrow{OP} + \overrightarrow{PC}_i = \overrightarrow{OA}_i + \overrightarrow{A_i B_i} + \overrightarrow{B_i C_i} \quad (\text{A.26})$$

Al diferenciar vectorialmente la ecuación A.26 con respecto al tiempo:

$$\dot{\overrightarrow{OP}} + \dot{\overrightarrow{PC}_i} = \dot{\overrightarrow{OA}_i} + \dot{\overrightarrow{A_i B_i}} + \dot{\overrightarrow{B_i C_i}}$$

Donde $\dot{\overrightarrow{PC}_i} = \dot{\overrightarrow{OA}_i} = 0$, entonces:

$$\vec{v}_p = \vec{\omega}_{1i} \times \vec{a}_i + \vec{\omega}_{2i} \times \vec{b}_i \quad (\text{A.27})$$

Donde $\vec{v}_p = [v_{px}, v_{py}, v_{pz}]^T$ es la velocidad lineal de la plataforma móvil, $\vec{a}_i = \overrightarrow{A_i B_i}$, $\vec{b}_i = \overrightarrow{B_i C_i}$ y $\vec{\omega}_{ji}$ es la velocidad angular del eslabón j de la cadena cinemática i . Para eliminar $\vec{\omega}_{2i}$ es necesario hacer un producto punto en ambos lados de la ecuación A.27 por un vector unitario \hat{b}_i . Por lo tanto:

$$\begin{aligned} \hat{b}_i \cdot \vec{v}_p &= \hat{b}_i \cdot (\vec{\omega}_{1i} \times \vec{a}_i + \vec{\omega}_{2i} \times \vec{b}_i) \\ &\Rightarrow \hat{b}_i \cdot \vec{v}_p = \hat{b}_i \cdot (\vec{\omega}_{1i} \times \vec{a}_i) + \hat{b}_i \cdot (\vec{\omega}_{2i} \times \vec{b}_i) \\ &\Rightarrow \hat{b}_i \cdot \vec{v}_p = \vec{\omega}_{1i} \cdot (\vec{a}_i \times \hat{b}_i) + \vec{\omega}_{2i} \cdot (\vec{b}_i \times \hat{b}_i) \\ &\Rightarrow \hat{b}_i \cdot \vec{v}_p = \vec{\omega}_{1i} \cdot (\vec{a}_i \times \hat{b}_i) \\ &\Rightarrow \hat{b}_i \cdot \vec{v}_p = \hat{b}_i \cdot (\vec{\omega}_{1i} \times \vec{a}_i) \end{aligned}$$

Por lo tanto, la igualdad que se utiliza para calcular el jacobiano es:

$$\hat{b}_i \cdot \vec{v}_p = \hat{b}_i \cdot (\vec{\omega}_{1i} \times \vec{a}_i) \quad (\text{A.28})$$

El vector unitario del vector \vec{b}_i es calculado con la siguiente expresión:

$$\hat{b}_i = \frac{\vec{b}_i}{\|\vec{b}_i\|} = \frac{\vec{b}_i}{b} = \begin{bmatrix} \sin \theta_{3i} \cos(\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin(\theta_{1i} + \theta_{2i}) \end{bmatrix}$$

Al reescribir los vectores de la ecuación A.28 en el marco de coordenadas $A_i - x_i y_i z_i$ conduce a:

$$\vec{a}_i = a \begin{bmatrix} \cos \theta_{1i} \\ 0 \\ \sin \theta_{1i} \end{bmatrix}; \vec{b}_i = b \begin{bmatrix} \sin \theta_{3i} \cos(\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin(\theta_{1i} + \theta_{2i}) \end{bmatrix}; \quad \vec{\omega}_{1i} = \begin{bmatrix} 0 \\ -\dot{\theta}_{1i} \\ 0 \end{bmatrix}; \vec{v}_p = \begin{bmatrix} v_{px} \cos \phi_i + v_{py} \sin \phi_i \\ -v_{px} \sin \phi_i + v_{py} \cos \phi_i \\ v_{pz} \end{bmatrix}$$

El signo negativo en la segunda fila de $\vec{\omega}_{1i}$ es solo una cuestión de convención.

Calculando $\hat{b}_i \cdot \vec{v_p}$, sustituyendo los valores \hat{b}_i y $\vec{v_p}$:

$$\begin{aligned}
\hat{b}_i \cdot \vec{v_p} &= \begin{bmatrix} \sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i}) \end{bmatrix} \cdot \begin{bmatrix} v_{px} \cos \phi_i + v_{py} \sin \phi_i \\ -v_{px} \sin \phi_i + v_{py} \cos \phi_i \\ v_{pz} \end{bmatrix} \\
&= [(\sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i})) * (v_{px} \cos \phi_i + v_{py} \sin \phi_i)] + \\
&\quad [(\cos \theta_{3i}) * (-v_{px} \sin \phi_i + v_{py} \cos \phi_i)] + \\
&\quad [(\sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i})) * (v_{pz})] \\
&= [\sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \cos \phi_i - \cos \theta_{3i} \sin \phi_i] v_{px} + \\
&\quad [\sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \sin \phi_i + \cos \theta_{3i} \cos \phi_i] v_{py} + \\
&\quad [(\sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i}))] v_{pz} \\
&= J_{ix} v_{px} + J_{iy} v_{py} + J_{iz} v_{pz}
\end{aligned}$$

Por lo tanto:

$$\hat{b}_i \cdot \vec{v_p} = J_{ix} v_{px} + J_{iy} v_{py} + J_{iz} v_{pz} \quad (\text{A.29})$$

Donde:

$$J_{ix} = \cos (\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \cos \phi_i - \cos \theta_{3i} \sin \phi_i \quad (\text{A.30})$$

$$J_{iy} = \cos (\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \sin \phi_i + \cos \theta_{3i} \cos \phi_i \quad (\text{A.31})$$

$$J_{iz} = \sin (\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \quad (\text{A.32})$$

Calculando $\hat{b}_i \cdot (\vec{\omega_{1i}} \times \vec{a_i})$, sustituyendo los valores $\vec{a_i}$, \hat{b}_i , $\vec{\omega_{1i}}$:

$$\begin{aligned}
\hat{b}_i \cdot (\vec{\omega_{1i}} \times \vec{a_i}) &= \begin{bmatrix} \sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i}) \end{bmatrix} \cdot \left(\begin{bmatrix} 0 \\ -\dot{\theta_{1i}} \\ 0 \end{bmatrix} \times a \begin{bmatrix} \cos \theta_{1i} \\ 0 \\ \sin \theta_{1i} \end{bmatrix} \right) \\
&= \begin{bmatrix} \sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \\ \cos \theta_{3i} \\ \sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i}) \end{bmatrix} \cdot a \begin{bmatrix} -\sin \theta_{1i} \dot{\theta_{1i}} \\ 0 \\ \cos \theta_{1i} \dot{\theta_{1i}} \end{bmatrix} \\
&= a \left(-\sin \theta_{3i} \cos (\theta_{1i} + \theta_{2i}) \sin \theta_{1i} \dot{\theta_{1i}} \right) + \left(\sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i}) \cos \theta_{1i} \dot{\theta_{1i}} \right) \\
&= a \dot{\theta_{1i}} \sin \theta_{3i} (-\cos (\theta_{1i} + \theta_{2i}) \sin \theta_{1i} + \sin (\theta_{1i} + \theta_{2i}) \cos \theta_{1i}) \\
&= a \dot{\theta_{1i}} \sin \theta_{3i} (-\cos (\theta_{1i} + \theta_{2i}) \sin \theta_{1i} + \sin (\theta_{1i} + \theta_{2i}) \cos \theta_{1i}) \\
&= a \dot{\theta_{1i}} \sin \theta_{3i} \sin (\theta_{1i} + \theta_{2i} - \theta_{1i}) \\
&= a \dot{\theta_{1i}} \sin \theta_{3i} \sin \theta_{2i} \\
&= a \sin \theta_{2i} \sin \theta_{3i} \dot{\theta_{1i}}
\end{aligned}$$

Por lo tanto:

$$\hat{b}_i \cdot (\vec{\omega_{1i}} \times \vec{a_i}) = a \sin \theta_{2i} \sin \theta_{3i} \dot{\theta_{1i}} \quad (\text{A.33})$$

Sustituyendo las ecuaciones A.29 y A.33 en la ecuación A.28:

$$J_{ix}v_{px} + J_{iy}v_{py} + J_{iz}v_{pz} = a \sin \theta_{2i} \sin \theta_{3i} \dot{\theta}_{1i} \quad (\text{A.34})$$

Al expandir la ecuación A.34 para $i = 1, 2, 3$ se obtienen tres ecuaciones escalares:

$$\begin{aligned} J_{1x}v_{px} + J_{1y}v_{py} + J_{1z}v_{pz} &= a \sin \theta_{21} \sin \theta_{31} \dot{\theta}_{11} \\ J_{2x}v_{px} + J_{2y}v_{py} + J_{2z}v_{pz} &= a \sin \theta_{22} \sin \theta_{32} \dot{\theta}_{12} \\ J_{3x}v_{px} + J_{3y}v_{py} + J_{3z}v_{pz} &= a \sin \theta_{23} \sin \theta_{33} \dot{\theta}_{13} \end{aligned}$$

Se ensambla en una forma matricial como:

$$\begin{bmatrix} J_{1x} & J_{1y} & J_{1z} \\ J_{2x} & J_{2y} & J_{2z} \\ J_{3x} & J_{3y} & J_{3z} \end{bmatrix} \begin{bmatrix} v_{px} \\ v_{py} \\ v_{pz} \end{bmatrix} = \begin{bmatrix} a \sin \theta_{21} \sin \theta_{31} & 0 & 0 \\ 0 & a \sin \theta_{22} \sin \theta_{32} & 0 \\ 0 & 0 & a \sin \theta_{23} \sin \theta_{33} \end{bmatrix} \begin{bmatrix} \dot{\theta}_{11} \\ \dot{\theta}_{12} \\ \dot{\theta}_{13} \end{bmatrix}$$

Se les asignan símbolos las matrices como:

$$J_x v_p = J_\theta \dot{\theta} \quad (\text{A.35})$$

Donde:

$$J_\theta = \begin{bmatrix} J_{1\theta} & 0 & 0 \\ 0 & J_{2\theta} & 0 \\ 0 & 0 & J_{3\theta} \end{bmatrix}; J_x = \begin{bmatrix} J_{1x} & J_{1y} & J_{1z} \\ J_{2x} & J_{2y} & J_{2z} \\ J_{3x} & J_{3y} & J_{3z} \end{bmatrix}$$

Asignando subíndices estándar para las 3 cadenas cinemáticas:

$$J_{i\theta} = a \sin \theta_{2i} \sin \theta_{3i}, i = \{1, 2, 3\} \quad (\text{A.36})$$

Manipulando la ecuación anterior algebraicamente:

$$v_p = J \dot{\theta} \quad (\text{A.37})$$

Finalmente, el jacobiano es J y representa el cambio de las posiciones en el espacio cartesiano de la plataforma móvil respecto al cambio de los ángulos en el espacio articular de los actuadores:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} \end{bmatrix} \quad (\text{A.38})$$

Donde:

- $J = J_x^{-1} J_\theta$ es el jacobiano del robot delta
- $v_p = [v_{px}, v_{py}, v_{pz}]^T$ es la velocidad del punto p en la plataforma móvil
- $\dot{\theta} = [\dot{\theta}_{11}, \dot{\theta}_{12}, \dot{\theta}_{13}]^T$ es la velocidad angular de los actuadores

A.1.3. Modelación Cinemática Aceleración (A)

Desde la sección A.1.2, modelación cinemática de velocidad, se tiene que:

$$J_x v_p = J_\theta \dot{\theta} \quad (\text{A.39})$$

$$\begin{bmatrix} J_{1x} & J_{1y} & J_{1z} \\ J_{2x} & J_{2y} & J_{2z} \\ J_{3x} & J_{3y} & J_{3z} \end{bmatrix} \begin{bmatrix} v_{px} \\ v_{py} \\ v_{pz} \end{bmatrix} = \begin{bmatrix} J_{1\theta} & 0 & 0 \\ 0 & J_{2\theta} & 0 \\ 0 & 0 & J_{3\theta} \end{bmatrix} \begin{bmatrix} \dot{\theta}_{11} \\ \dot{\theta}_{12} \\ \dot{\theta}_{13} \end{bmatrix}$$

Derivando matricialmente la ecuación A.39:

$$\begin{aligned} \dot{J}_x v_p + J_x \dot{v}_p &= \dot{J}_\theta \dot{\theta} + J_\theta \ddot{\theta} \\ \implies \dot{J}_x v_p + J_x a_p &= \dot{J}_\theta \dot{\theta} + J_\theta \ddot{\theta} \end{aligned}$$

Despejando la aceleración angular de los actuadores:

$$\begin{aligned} J_\theta \ddot{\theta} &= \dot{J}_x v_p + J_x a_p - \dot{J}_\theta \dot{\theta} \\ \implies \ddot{\theta} &= J_\theta^{-1} * [\dot{J}_x v_p + J_x a_p - \dot{J}_\theta \dot{\theta}] \end{aligned} \quad (\text{A.40})$$

Donde :

$$\begin{aligned} \dot{J}_x &= \begin{bmatrix} \dot{J}_{1x} & \dot{J}_{1y} & \dot{J}_{1z} \\ \dot{J}_{2x} & \dot{J}_{2y} & \dot{J}_{2z} \\ \dot{J}_{3x} & \dot{J}_{3y} & \dot{J}_{3z} \end{bmatrix} \\ \dot{J}_\theta &= \begin{bmatrix} \dot{J}_{1\theta} & 0 & 0 \\ 0 & \dot{J}_{2\theta} & 0 \\ 0 & 0 & \dot{J}_{3\theta} \end{bmatrix} \end{aligned}$$

Calculando las derivadas de los elementos de la matriz J_x donde se sabe desde la cinemática de velocidad que:

$$\begin{aligned} J_{ix} &= \cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \cos \phi_i - \cos \theta_{3i} \sin \phi_i \\ J_{iy} &= \cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \sin \phi_i + \cos \theta_{3i} \cos \phi_i \\ J_{iz} &= \sin(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \end{aligned}$$

Derivando J_{ix} respecto al tiempo:

$$\dot{J}_{ix} = [\cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \cos \phi_i]' - [\cos \theta_{3i} \sin \phi_i]'$$

Simplificando la ecuación:

$$\dot{J}_{ix} = A'_{ix} - B'_{ix} \quad (\text{A.41})$$

Donde A'_{ix} :

$$\begin{aligned}
 A'_{ix} &= [\cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \cos \phi_i]' \\
 A'_{ix} &= \cos \phi_i * [\cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i}]' \\
 A'_{ix} &= \cos \phi_i * \left[[\cos'(\theta_{1i} + \theta_{2i}) \sin \theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) \sin' \theta_{3i}] \right] \\
 A'_{ix} &= \cos \phi_i * \left[[-\sin(\theta_{1i} + \theta_{2i}) * (\theta_{1i} + \theta_{2i})' * \sin \theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) * \cos \theta_{3i} * \theta_{3i}'] \right] \\
 A'_{ix} &= \cos \phi_i * \left[[-\sin(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin \theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) * \cos \theta_{3i} * \dot{\theta}_{3i}] \right]
 \end{aligned}$$

Donde B'_{ix} :

$$\begin{aligned}
 B'_{ix} &= [\cos \theta_{3i} \sin \phi_i]' \\
 B'_{ix} &= \sin \phi_i * [\cos \theta_{3i}]' \\
 B'_{ix} &= \sin \phi_i * [-\sin \theta_{3i} * \dot{\theta}_{3i}]
 \end{aligned}$$

Derivando J_{iy} respecto al tiempo:

$$\dot{J}_{iy} = [\cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \sin \phi_i]' + [\cos \theta_{3i} \cos \phi_i]'$$

Simplificando la ecuación:

$$\dot{J}_{iy} = A'_{iy} + B'_{iy} \quad (\text{A.42})$$

Donde A'_{iy} :

$$\begin{aligned}
 A'_{iy} &= [\cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i} \sin \phi_i]' \\
 A'_{iy} &= \sin \phi_i * [\cos(\theta_{1i} + \theta_{2i}) \sin \theta_{3i}]' \\
 A'_{iy} &= \sin \phi_i * \left[[\cos'(\theta_{1i} + \theta_{2i}) \sin \theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) \sin' \theta_{3i}] \right] \\
 A'_{iy} &= \sin \phi_i * \left[[-\sin(\theta_{1i} + \theta_{2i}) * (\theta_{1i} + \theta_{2i})' * \sin \theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) * \cos \theta_{3i} * \dot{\theta}_{3i}] \right] \\
 A'_{iy} &= \sin \phi_i * \left[[-\sin(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin \theta_{3i}] + [\cos(\theta_{1i} + \theta_{2i}) * \cos \theta_{3i} * \dot{\theta}_{3i}] \right]
 \end{aligned}$$

Donde B'_{iy} :

$$\begin{aligned}
 B'_{iy} &= [\cos \theta_{3i} \cos \phi_i]' \\
 B'_{iy} &= \cos \phi_i * [\cos \theta_{3i}]' \\
 B'_{iy} &= \cos \phi_i * [-\sin \theta_{3i} * \dot{\theta}_{3i}]
 \end{aligned}$$

Derivando J_{iz} respecto al tiempo:

$$\dot{J}_{iz} = [\sin(\theta_{1i} + \theta_{2i}) \sin \theta_{3i}]'$$

$$\dot{J}_{iz} = [\sin'(\theta_{1i} + \theta_{2i}) \sin \theta_{3i}] + [\sin(\theta_{1i} + \theta_{2i}) \sin' \theta_{3i}]$$

$$\dot{J}_{iz} = \left[\cos(\theta_{1i} + \theta_{2i}) * (\dot{\theta}_{1i} + \dot{\theta}_{2i}) * \sin \theta_{3i} \right] + \left[\sin(\theta_{1i} + \theta_{2i}) * \cos \theta_{3i} * \dot{\theta}_{3i} \right] \quad (\text{A.43})$$

Calculando las derivadas de los elementos de la matriz J_θ donde se sabe desde la cinemática de velocidad que:

$$J_{i\theta} = a \sin \theta_{2i} \sin \theta_{3i}, i = \{1, 2, 3\}$$

Derivando $J_{i\theta}$ respecto al tiempo:

$$\begin{aligned}\dot{J}_{i\theta} &= [a \sin \theta_{2i} \sin \theta_{3i}]' \\ \dot{J}_{i\theta} &= a [\sin \theta_{2i} \sin \theta_{3i}]' \\ \dot{J}_{i\theta} &= a [[\sin' \theta_{2i} \sin \theta_{3i}] + [\sin \theta_{2i} \sin' \theta_{3i}]] \\ \dot{J}_{i\theta} &= a [[\cos \theta_{2i} * \dot{\theta}_{2i} * \sin \theta_{3i}] + [\sin \theta_{2i} * \cos \theta_{3i} * \dot{\theta}_{3i}]]\end{aligned}\quad (\text{A.44})$$

Las derivadas de los ángulos interiores θ_{2i} y θ_{3i} se determinarán derivando las ecuaciones:

$$\begin{aligned}\theta_{3i} &= \cos^{-1} \frac{c_{yi}}{b} \\ \theta_{2i} &= \cos^{-1} \left(\frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2ab \sin \theta_{3i}} \right)\end{aligned}$$

Donde:

$$\begin{aligned}c_{xi} &= \cos \phi_i * p_x + \sin \phi_i * p_y + h - r \\ c_{yi} &= -\sin \phi_i * p_x + \cos \phi_i * p_y \\ c_{zi} &= p_z\end{aligned}$$

Derivando θ_{3i} respecto al tiempo:

$$\begin{aligned}\dot{\theta}_{3i} &= \left[\cos^{-1} \frac{c_{yi}}{b} \right]' \\ \dot{\theta}_{3i} &= \left[\frac{-1}{\sqrt{1 - \left(\frac{c_{yi}}{b}\right)^2}} \right] * \left[\frac{c_{yi}}{b} \right]' \\ \dot{\theta}_{3i} &= \left[\frac{-1}{\sqrt{1 - \left(\frac{c_{yi}}{b}\right)^2}} \right] * \left[\frac{c'_{yi}}{b} \right]\end{aligned}\quad (\text{A.45})$$

La derivada de c_{yi} es:

$$\begin{aligned}c'_{yi} &= [-\sin \phi_i * p_x + \cos \phi_i * p_y]' \\ c'_{yi} &= [-\sin \phi_i * \dot{p}_x + \cos \phi_i * \dot{p}_y] \\ c'_{yi} &= [-\sin \phi_i * v_x + \cos \phi_i * v_y]\end{aligned}$$

Derivando θ_{2i} respecto al tiempo:

$$\dot{\theta}_{2i} = \left[\cos^{-1} \left(\frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2absin \theta_{3i}} \right) \right]'$$

Se nombra a:

$$K = \frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2absin \theta_{3i}}$$

Reemplazando:

$$\dot{\theta}_{2i} = \left[\frac{-1}{\sqrt{1 - (K)^2}} \right] * [K'] \quad (\text{A.46})$$

Calculando K' :

$$\begin{aligned} K' &= \left[\frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2absin \theta_{3i}} \right]' \\ K' &= \left[\frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{2absin \theta_{3i}} \right]' \\ K' &= \left[\frac{1}{2ab} \right] \left[\frac{c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2}{sin \theta_{3i}} \right]' \end{aligned}$$

Abreviando la fórmula:

$$K' = \left[\frac{1}{2ab} \right] * \left[\frac{A_{11} - A_{12}}{B_1} \right]$$

Donde:

$$\begin{aligned} A_{11} &= [c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2]' * [sin \theta_{3i}] \\ A_{11} &= [2c_{xi}c_{xi}' + 2c_{yi}c_{yi}' + 2c_{zi}c_{zi}'] * [sin \theta_{3i}] \\ A_{12} &= [c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2] * [sin \theta_{3i}]' \\ A_{12} &= [c_{xi}^2 + c_{yi}^2 + c_{zi}^2 - a^2 - b^2] * [cos \theta_{3i} * \dot{\theta}_{3i}] \\ B_1 &= [sin \theta_{3i}]^2 \end{aligned}$$

La derivada de c_{xi} es:

$$\begin{aligned} c_{xi}' &= [\cos \phi_i * p_x + \sin \phi_i * p_y + h - r]' \\ c_{xi}' &= \cos \phi_i * \dot{p}_x + \sin \phi_i * \dot{p}_y \\ c_{xi}' &= \cos \phi_i * v_x + \sin \phi_i * v_y \end{aligned}$$

La derivada de c_{zi} es:

$$c_{zi}' = \dot{p}_z = v_z$$

Finalmente se tienen todas las ecuaciones para determinar la aceleración con la ecuación A.40.

A.1.4. Modelación dinámica

A.1.4.1. Restricciones

Se requiere una función de restricción $f_l^{(h)}$ para resolver los valores de torque. Se utilizará la restricción formada por la junta de paralelogramos, ya que su longitud L_2 será igual en cualquier momento.

Entonces, las restricciones para el eslabón $\vec{s} = \vec{L}_2$ se formula de la siguiente manera:

$$\vec{P} = \vec{r}_a + \vec{L}_1 + \vec{s} + \vec{r}_B$$

$$\vec{s} = \vec{P} - (\vec{r}_a + \vec{L}_1 + \vec{r}_B)$$

Para el motor i

$$\vec{s} = \begin{bmatrix} X_P \\ Y_P \\ Z_P \end{bmatrix} - Rot(\phi_i) \left(\begin{bmatrix} r_a \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} L_1 \cos \theta_i \\ 0 \\ L_1 \sin \theta_i \end{bmatrix} + \begin{bmatrix} -r_b \\ 0 \\ 0 \end{bmatrix} \right)$$

Donde:

$$Rot(\phi_i) = \begin{bmatrix} \cos \phi_i & -\sin \phi_i & 0 \\ \sin \phi_i & \cos \phi_i & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Donde $\phi_1 = 0^\circ$, $\phi_2 = 120^\circ$, $\phi_3 = 240^\circ$. Reemplazando:

$$\vec{s} = \begin{bmatrix} X_P \\ Y_P \\ Z_P \end{bmatrix} - \begin{bmatrix} \cos \phi_i & -\sin \phi_i & 0 \\ \sin \phi_i & \cos \phi_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L_1 \cos \theta_i + r \\ 0 \\ L_1 \sin \theta_i \end{bmatrix}$$

Donde $r = r_a - r_b$

Entonces \vec{s} es:

$$\vec{s} = \begin{bmatrix} X_P \\ Y_P \\ Z_P \end{bmatrix} - \begin{bmatrix} L_1 \cos \theta_i \cos \phi_i + r \cos \phi_i \\ L_1 \cos \theta_i \sin \phi_i + r \sin \phi_i \\ L_1 \sin \theta_i \end{bmatrix} = \begin{bmatrix} X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i \\ Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i \\ Z_P - L_1 \sin \theta_i \end{bmatrix}$$

Las restricciones son sobre el largo del antebrazo, ya que es constante en todo momento en magnitud. La ecuación que representa lo anterior para $i=1,2,3$ es:

$$\|\vec{s}\| - \vec{L}_2 = 0$$

Por lo tanto, siguiendo la nomenclatura de restricciones de Lagrange, se tiene 3 restricciones por cada antebrazo los cuales son para $i=1,2,3$:

$$\begin{aligned} f_i^{(h)}(\theta_i, \phi_i) = & (X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i)^2 + \\ & (Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i)^2 + (Z_P - L_1 \sin \theta_i)^2 - L_2^2 = 0 \end{aligned}$$

A.1.4.2. Lagrangiano, energía cinética y energía potencial

Se sabe que la energía cinética de un cuerpo sólido queda expresada por la siguiente ecuación:

$$E = E_{lineal} + E_{rotacional} = \frac{1}{2}mv_{cm}^2 + \frac{1}{2}I_{cm}\omega^2 \quad (\text{A.47})$$

Donde v_{cm} es la velocidad centro de masa, m es la masa del cuerpo, I_{cm} es la inercia rotacional respecto al centro de masa y ω es la velocidad angular.

Se sabe que la energía potencial de una partícula queda expresada por:

$$E_{potencial} = m * g * h \quad (\text{A.48})$$

Donde m es la masa del cuerpo, g es la aceleración de gravedad y h es altura respecto al sistema de referencia.

A partir de las ecuaciones A.47 y A.48, la energía cinética del robot delta a partir de la simplificación de masas en partículas m_p , m_1 , m_2 es:

$$\begin{aligned} T &= [E_{lineal, (m_p)}] + \left[\sum_{i=1}^3 E_{rotacional, m_1} \right] + 2 \left[\sum_{i=1}^3 E_{lineal, \frac{m_2}{2}} + E_{rotacional, \frac{m_2}{2}} \right] \\ T &= \left[\frac{1}{2}m_P \|\dot{p}\|^2 \right] + \left[\sum_{i=1}^3 \left(\frac{1}{2} \right) \left(\frac{1}{3}m_1(L_1)^2 \right) (\dot{\theta}_i)^2 \right] + \\ &\quad 2 \left[\sum_{i=1}^3 \left(\frac{1}{2} \right) \left(\frac{m_2}{2} \right) \|\dot{p}\|^2 + \left(\frac{1}{2} \right) \left(\frac{m_2}{2} \right) (L_1\dot{\theta}_i)^2 \right] \\ T &= \left[\frac{1}{2}m_P \|\dot{p}\|^2 \right] + \left[\sum_{i=1}^3 \left(\frac{1}{2} \right) \left(\frac{1}{3}m_1(L_1)^2 \right) (\dot{\theta}_i)^2 \right] + \\ &\quad 2 \left[\sum_{i=1}^3 \left(\frac{1}{2} \right) \left(\frac{m_2}{2} \right) \left(\|\dot{p}\|^2 + (L_1\dot{\theta}_i)^2 \right) \right] \end{aligned}$$

Reemplazando $\|\dot{p}\|^2 = \dot{X}_p^2 + \dot{Y}_p^2 + \dot{Z}_p^2$

$$T = \left[\frac{1}{2}m_P (\dot{X}_p^2 + \dot{Y}_p^2 + \dot{Z}_p^2) \right] + \left[\left(\frac{1}{6}m_1 L_1^2 \right) \sum_{i=1}^3 \dot{\theta}_i^2 \right] + \frac{m_2}{2} \left[\sum_{i=1}^3 (\dot{X}_p^2 + \dot{Y}_p^2 + \dot{Z}_p^2) + (L_1^2 \dot{\theta}_i^2) \right] \quad (\text{A.49})$$

La energía potencial del robot delta se puede calcular en relación a la base fija y con la simplificación de masas de la siguiente manera:

$$\begin{aligned} V &= [E_{potencial, m_p}] + \left[\sum_{i=1}^3 E_{potencial, m_1} \right] + 2 \left[\sum_{i=1}^3 E_{potencial, \frac{m_2}{2}} \right] \\ V &= -[m_p g Z_p] - \left[m_1 g \sum_{i=1}^3 \left(\frac{\sin \theta_i L_1}{2} \right) \right] - 2 \left[g \sum_{i=1}^3 \left(\frac{m_2}{2} \right) (\sin \theta_i * L_1) + \left(\frac{m_2}{2} \right) (Z_p) \right] \end{aligned}$$

Simplificando la ecuación para los siguientes cálculos:

$$V = -[m_p g Z_p] - \left[\frac{1}{2}m_1 g L_1 \sum_{i=1}^3 \sin \theta_i \right] - \left[m_2 g L_1 \sum_{i=1}^3 \sin \theta_i \right] - [3m_2 g Z_p] \quad (\text{A.50})$$

A.1.4.3. Multiplicadores de Lagrange

Para obtener los 3 valores de los multiplicadores de Lagrange λ_l es necesario establecer 3 ecuaciones de la forma de la ecuación 4.56, más 3 ecuaciones de restricciones de la forma 4.63. Además, se tiene en cuenta que el término $Q_j^{(NU)}(F_{px}, F_{py}, F_{pz})$ son las fuerzas externas que se aplican al efecto final. Por lo tanto, derivando respecto a las coordenadas generalizadas $q_j \in \{X_p, Y_p, Z_p\}$ se obtienen las 3 ecuaciones dinámicas:

$$\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{X}_p} \right) - \frac{\delta L}{\delta X_p} = \sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta X_p} + F_{px} = \lambda_1 \frac{\delta f_1^{(h)}}{\delta X_p} + \lambda_2 \frac{\delta f_2^{(h)}}{\delta X_p} + \lambda_3 \frac{\delta f_3^{(h)}}{\delta X_p} + F_{px} \quad (\text{A.51})$$

$$\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Y}_p} \right) - \frac{\delta L}{\delta Y_p} = \sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta Y_p} + F_{py} = \lambda_1 \frac{\delta f_1^{(h)}}{\delta Y_p} + \lambda_2 \frac{\delta f_2^{(h)}}{\delta Y_p} + \lambda_3 \frac{\delta f_3^{(h)}}{\delta Y_p} + F_{py} \quad (\text{A.52})$$

$$\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Z}_p} \right) - \frac{\delta L}{\delta Z_p} = \sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta Z_p} + F_{pz} = \lambda_1 \frac{\delta f_1^{(h)}}{\delta Z_p} + \lambda_2 \frac{\delta f_2^{(h)}}{\delta Z_p} + \lambda_3 \frac{\delta f_3^{(h)}}{\delta Z_p} + F_{pz} \quad (\text{A.53})$$

Reordenando matricialmente para determinar los multiplicadores:

$$\begin{pmatrix} \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{X}_p} \right) - \frac{\delta L}{\delta X_p} - F_{px} \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Y}_p} \right) - \frac{\delta L}{\delta Y_p} - F_{py} \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Z}_p} \right) - \frac{\delta L}{\delta Z_p} - F_{pz} \end{pmatrix} = \begin{pmatrix} \frac{\delta f_1^{(h)}}{\delta \dot{X}_p} & \frac{\delta f_2^{(h)}}{\delta \dot{X}_p} & \frac{\delta f_3^{(h)}}{\delta \dot{X}_p} \\ \frac{\delta f_1^{(h)}}{\delta \dot{Y}_p} & \frac{\delta f_2^{(h)}}{\delta \dot{Y}_p} & \frac{\delta f_3^{(h)}}{\delta \dot{Y}_p} \\ \frac{\delta f_1^{(h)}}{\delta \dot{Z}_p} & \frac{\delta f_2^{(h)}}{\delta \dot{Z}_p} & \frac{\delta f_3^{(h)}}{\delta \dot{Z}_p} \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} \quad (\text{A.54})$$

Calculando los valores relacionados con la coordenada generalizada $q_j = X_p$:

$$\begin{aligned} \frac{\delta L}{\delta \dot{X}_p} &= \frac{\delta T}{\delta \dot{X}_p} - \frac{\delta V}{\delta \dot{X}_p} = \frac{\delta T}{\delta \dot{X}_p} - [0] = \frac{\delta T}{\delta \dot{X}_p} \\ \frac{\delta L}{\delta \dot{X}_p} &= \left[2 * \frac{1}{2} * m_p * \dot{X}_p \right] + [0] + \left[\frac{1}{2} * m_2 * \sum_{i=1}^3 2 * \dot{X}_p \right] \\ \frac{\delta L}{\delta \dot{X}_p} &= [m_p \dot{X}_p] + [3m_2 \dot{X}_p] \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{X}_p} \right) &= \frac{d}{dt} ([m_p \dot{X}_p] + [3m_2 \dot{X}_p]) \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{X}_p} \right) &= [m_p \ddot{X}_p] + [3m_2 \ddot{X}_p] \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{X}_p} \right) &= \ddot{X}_p [3m_2 + m_p] \\ \frac{\delta L}{\delta X_p} &= 0 \end{aligned}$$

$$\sum_{l=1}^3 \lambda_l \frac{\delta f_l^{(h)}}{\delta X_p} = \sum_{l=i=1}^3 \lambda_l * 2 (X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i)$$

Reemplazando en la ecuación A.51

$$\begin{aligned}
 \ddot{X}_p [3m_2 + m_p] - 0 &= F_{px} + \sum_{l=i=1}^3 \lambda_l * 2 (X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i) \\
 [3m_2 + m_p] \ddot{X}_p &= F_{px} + \sum_{l=i=1}^3 \lambda_l * 2 (X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i) \\
 (m_p + 3m_2) \ddot{X}_p - 2 \sum_{l=i=1}^3 \lambda_l (X_P - r \cos \phi_i - L_1 \cos \theta_i \cos \phi_i) &= F_{px} \tag{A.55}
 \end{aligned}$$

La fuerza F_x representa la fuerza externa en dirección x sobre la plataforma móvil. Calculando los valores relacionados con coordenada generalizada $q_j = Y_p$:

$$\begin{aligned}
 \frac{\delta L}{\delta \dot{Y}_p} &= \frac{\delta T}{\delta \dot{Y}_p} - \frac{\delta V}{\delta \dot{Y}_p} = \frac{\delta T}{\delta \dot{Y}_p} - [0] = \frac{\delta T}{\delta \dot{Y}_p} \\
 \frac{\delta L}{\delta \dot{Y}_p} &= \left[\frac{1}{2} * 2 * m_P * \dot{Y}_p \right] + \left(\frac{m_2}{2} \right) * \left[\sum_{i=1}^3 2 * \dot{Y}_p \right] \\
 \frac{\delta L}{\delta \dot{Y}_p} &= \left[m_P \dot{Y}_p \right] + \left[3m_2 \dot{Y}_p \right] \\
 \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Y}_p} \right) &= \frac{d}{dt} \left(\left[m_P \dot{Y}_p \right] + \left[3m_2 \dot{Y}_p \right] \right) \\
 \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Y}_p} \right) &= \left[m_P \ddot{Y}_p \right] + \left[3m_2 \ddot{Y}_p \right] \\
 \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Y}_p} \right) &= [m_P + 3m_2] \ddot{Y}_p \\
 \frac{\delta L}{\delta Y_p} &= 0
 \end{aligned}$$

$$\sum_{l=1}^3 \lambda_l \frac{\delta f_l^{(h)}}{\delta Y_p} = \sum_{l=i=1}^3 \lambda_l * 2 (Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i)$$

Reemplazando en la ecuación A.52:

$$\begin{aligned}
 [m_P + 3m_2] \ddot{Y}_p - 0 &= F_{py} + \sum_{l=i=1}^3 \lambda_l * 2 (Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i) \\
 [m_P + 3m_2] \ddot{Y}_p &= F_{py} + \sum_{l=i=1}^3 \lambda_l * 2 (Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i) \\
 (m_P + 3m_2) \ddot{Y}_p - 2 \sum_{l=i=1}^3 \lambda_l (Y_P - r \sin \phi_i - L_1 \cos \theta_i \sin \phi_i) &= F_{py} \tag{A.56}
 \end{aligned}$$

Calculando los valores relacionados con la coordenada generalizada $q_j = Z_p$

$$\begin{aligned}
\frac{\delta L}{\delta \dot{Z}_p} &= \frac{\delta T}{\delta \dot{Z}_p} - \frac{\delta V}{\delta \dot{Z}_p} = \frac{\delta T}{\delta \dot{Z}_p} - [0] = \frac{\delta T}{\delta \dot{Z}_p} \\
\frac{\delta T}{\delta \dot{Z}_p} &= \left[\frac{1}{2} * 2 * m_P * \dot{Z}_p \right] + \left(\frac{m_2}{2} \right) * \left[\sum_{i=1}^3 2 * (\dot{Z}_p) \right] \\
\frac{\delta T}{\delta \dot{Z}_p} &= [m_P \dot{Z}_p] + [3m_2 \dot{Z}_p] \\
\frac{\delta T}{\delta \dot{Z}_p} &= [m_P + 3m_2] \dot{Z}_p \\
\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Z}_p} \right) &= [m_P + 3m_2] \ddot{Z}_p \\
\frac{\delta L}{\delta Z_p} &= -\frac{\delta V}{\delta Z_p} = [m_p g] + [3m_2 g] \\
\frac{\delta L}{\delta Z_p} &= g [m_p + 3m_2]
\end{aligned}$$

$$\sum_{l=1}^3 \lambda_l \frac{\delta f_l^{(h)}}{\delta Z_p} = \sum_{l=i=1}^3 \lambda_l * (2 * (Z_P - L_1 \sin \theta_i))$$

Reemplazando en la ecuación A.53

$$\begin{aligned}
[[m_P + 3m_2] \ddot{Z}_p] - [g [m_p + 3m_2]] &= \sum_{l=i=1}^3 \lambda_l * (2 * (Z_P - L_1 \sin \theta_i)) + F_{pz} \\
[[m_P + 3m_2] \ddot{Z}_p] - \sum_{l=i=1}^3 \lambda_l * (2 * (Z_P - L_1 \sin \theta_i)) - [g [m_p + 3m_2]] &= F_{pz} \\
[m_P + 3m_2] \ddot{Z}_p - 2 \sum_{l=i=1}^3 \lambda_l (Z_P - L_1 \sin \theta_i) - [m_p + 3m_2] g &= F_{pz} \quad (\text{A.57})
\end{aligned}$$

Finalmente, se calcula con álgebras de matrices los multiplicadores $\lambda_l, l = 1, 2, 3$ reemplazando las ecuaciones calculadas anteriormente en el sistema de ecuaciones matricial A.54:

$$\begin{aligned}
\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{X}_p} \right) - \frac{\delta L}{\delta X_p} - F_{px} &= (m_p + 3m_2) \ddot{X}_p - F_{px} \\
\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Y}_p} \right) - \frac{\delta L}{\delta Y_p} - F_{py} &= (m_P + 3m_2) \ddot{Y}_p - F_{py} \\
\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{Z}_p} \right) - \frac{\delta L}{\delta Z_p} - F_{pz} &= (m_P + 3m_2) \ddot{Z}_p - (m_p + 3m_2) g - F_{pz} \\
\frac{\delta f_1^{(h)}}{\delta X_p} &= 2 (X_P - r \cos \phi_1 - L_1 \cos \theta_1 \cos \phi_1)
\end{aligned}$$

$$\frac{\delta f_2^{(h)}}{\delta X_p} = 2(X_P - r\cos\phi_2 - L_1 \cos\theta_2 \cos\phi_2)$$

$$\frac{\delta f_3^{(h)}}{\delta X_p} = 2(X_P - r\cos\phi_3 - L_1 \cos\theta_3 \cos\phi_3)$$

$$\frac{\delta f_1^{(h)}}{\delta Y_p} = 2(Y_P - r\sin\phi_1 - L_1 \cos\theta_1 \sin\phi_1)$$

$$\frac{\delta f_2^{(h)}}{\delta Y_p} = 2(Y_P - r\sin\phi_2 - L_1 \cos\theta_2 \sin\phi_2)$$

$$\frac{\delta f_3^{(h)}}{\delta Y_p} = 2(Y_P - r\sin\phi_3 - L_1 \cos\theta_3 \sin\phi_3)$$

$$\frac{\delta f_1^{(h)}}{\delta Z_p} = 2(Z_P - L_1 \sin\theta_1)$$

$$\frac{\delta f_2^{(h)}}{\delta Z_p} = 2(Z_P - L_1 \sin\theta_2)$$

$$\frac{\delta f_3^{(h)}}{\delta Z_p} = 2(Z_P - L_1 \sin\theta_3)$$

A.1.4.4. Torque

Para determinar el torque de los actuadores $\tau_i, i = 1, 2, 3$ se utilizan 3 ecuaciones de la forma de la ecuación de Lagrange 4.56 con respecto las coordenadas generalizadas $q_j \in \{\theta_1, \theta_2, \theta_3\}$. Entonces:

$$\frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) - \frac{\delta L}{\delta \theta_i} = \sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta \theta_i} + \tau_i$$

Reordenando y despejando τ_i

$$\tau_i = \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) - \frac{\delta L}{\delta \theta_i} - \sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta \theta_i} \quad (\text{A.58})$$

Calculando el primer término de la ecuación A.58:

$$\begin{aligned} \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) &= \frac{d}{dt} \left(\frac{\delta T}{\delta \dot{\theta}_i} - \frac{\delta V}{\delta \dot{\theta}_i} \right) \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) &= \frac{d}{dt} \left(\frac{\delta T}{\delta \dot{\theta}_i} - 0 \right) \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) &= \frac{d}{dt} \left(\left[\left(\frac{1}{6} m_1 L_1^2 \right) * \sum_{i=1}^3 2\dot{\theta}_i \right] + \left(\frac{m_2}{2} \right) * \left[\sum_{i=1}^3 (2L_1^2 \dot{\theta}_i) \right] \right) \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) &= \frac{d}{dt} \left(\left(\frac{1}{3} m_1 L_1^2 \right) * \sum_{i=1}^3 \dot{\theta}_i + m_2 L_1^2 * \sum_{i=1}^3 \dot{\theta}_i \right) \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) &= \frac{d}{dt} \left(\left(\frac{1}{3} m_1 + m_2 \right) L_1^2 \sum_{i=1}^3 \dot{\theta}_i \right) \\ \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{\theta}_i} \right) &= \left(\frac{1}{3} m_1 + m_2 \right) L_1^2 \ddot{\theta}_i \end{aligned}$$

Calculando el segundo término de la ecuación A.58:

$$\begin{aligned} \frac{\delta L}{\delta \theta_i} &= \frac{\delta T}{\delta \theta_i} - \frac{\delta V}{\delta \theta_i} \\ \frac{\delta L}{\delta \theta_i} &= 0 - \frac{\delta V}{\delta \theta_i} \\ \frac{\delta L}{\delta \theta_i} &= -\frac{\delta V}{\delta \theta_i} \\ \frac{\delta L}{\delta \theta_i} &= \left[\frac{1}{2} m_1 g L_1 \cos \theta_i \right] + [m_2 g L_1 \cos \theta_i] \\ \frac{\delta L}{\delta \theta_i} &= \left(\frac{1}{2} m_1 + m_2 \right) g L_1 \cos \theta_i \end{aligned}$$

Calculando el tercer termino de la ecuación A.58:

$$\sum_{l=1}^{K^{(h)}} \lambda_l \frac{\delta f_l^{(h)}}{\delta \theta_i} = \lambda_1 \frac{\delta f_1^{(h)}}{\delta \theta_i} + \lambda_2 \frac{\delta f_2^{(h)}}{\delta \theta_i} + \lambda_3 \frac{\delta f_3^{(h)}}{\delta \theta_i}$$

Si $l \neq i$:

$$\frac{\delta f_l^{(h)}}{\delta \theta_i} = 0$$

Si $l = i$

$$\begin{aligned} \frac{\delta f_l^{(h)}}{\delta \theta_i} &= 2(X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i) (X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i)' + \\ &\quad 2(Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i) (Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i)' + \\ &\quad 2(Z_P - L_1 \sin \theta_i) (Z_P - L_1 \sin \theta_i)' \\ \frac{\delta f_l^{(h)}}{\delta \theta_i} &= 2(X_P - L_1 \cos \theta_i \cos \phi_i - r \cos \phi_i) (L_1 \sin \theta_i \cos \phi_i) + \\ &\quad 2(Y_P - L_1 \cos \theta_i \sin \phi_i - r \sin \phi_i) (L_1 \sin \theta_i \sin \phi_i) + \\ &\quad 2(Z_P - L_1 \sin \theta_i) (-L_1 \cos \theta_i) \\ \frac{\delta f_l^{(h)}}{\delta \theta_i} &= 2L_1 \left(X_P \cos \phi_i - L_1 \cos \theta_i \cos^2 \phi_i - r \cos^2 \phi_i \right) (\sin \theta_i) + \\ &\quad 2L_1 \left(Y_P \sin \phi_i - L_1 \cos \theta_i \sin^2 \phi_i - r \sin^2 \phi_i \right) (\sin \theta_i) + \\ &\quad 2L_1 (-Z_P \cos \theta_i + L_1 \sin \theta_i \cos \theta_i) \\ \frac{\delta f_l^{(h)}}{\delta \theta_i} &= 2L_1 \sin \theta_i \left(X_P \cos \phi_i - L_1 \cos \theta_i \cos^2 \phi_i - r \cos^2 \phi_i + \right. \\ &\quad \left. Y_P \sin \phi_i - L_1 \cos \theta_i \sin^2 \phi_i - r \sin^2 \phi_i \right) + \\ &\quad 2L_1 (-Z_P \cos \theta_i + L_1 \sin \theta_i \cos \theta_i) \end{aligned}$$

Utilizando la igualdad trigonométrica $\cos^2 \phi_i + \sin^2 \phi_i = 1$:

$$\begin{aligned} \frac{\delta f_l^{(h)}}{\delta \theta_i} &= 2L_1 \sin \theta_i (X_P \cos \phi_i - L_1 \cos \theta_i - r + Y_P \sin \phi_i) + \\ &\quad 2L_1 (-Z_P \cos \theta_i + L_1 \sin \theta_i \cos \theta_i) \end{aligned}$$

$$\begin{aligned} \frac{\delta f_l^{(h)}}{\delta \theta_i} &= 2L_1 [(X_P \cos \phi_i - L_1 \cos \theta_i - r + Y_P \sin \phi_i) \sin \theta_i + \\ &\quad (-Z_P \cos \theta_i + L_1 \sin \theta_i \cos \theta_i)] \end{aligned}$$

$$\frac{\delta f_l^{(h)}}{\delta \theta_i} = 2L_1 [(X_P \cos \phi_i - r + Y_P \sin \phi_i) \sin \theta_i + (-Z_P \cos \theta_i)]$$

$$\frac{\delta f_l^{(h)}}{\delta \theta_i} = 2L_1 [(X_P \cos \phi_i + Y_P \sin \phi_i - r) \sin \theta_i - Z_P \cos \theta_i]$$

Finalmente, el toque para los motores $i = 1, 2, 3$ se calcula con la siguiente formula:

$$\begin{aligned}\tau_i = & \left(\frac{1}{3}m_1 + m_2 \right) L_1^2 \ddot{\theta}_i - \left(\frac{1}{2}m_1 + m_2 \right) g L_1 \cos \theta_i - \\ & 2\lambda_i L_1 [(X_P \cos \phi_i + Y_P \sin \phi_i - r) \sin \theta_i - Z_P \cos \theta_i]\end{aligned}\quad (\text{A.59})$$

A.2. Método B

A.2.1. Modelación cinemática de aceleración

Desde la modelación cinemática de velocidad se tiene que:

$$\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix} \vec{\dot{P}}_0 + \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix} \vec{\dot{\theta}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Reordenando:

$$\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix} \vec{\dot{P}}_0 = - \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_1 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_1 \end{bmatrix} \vec{\dot{\theta}}$$

Derivando matricialmente la expresión anterior:

$$\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' \vec{\ddot{P}}_0 + \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix} \vec{\ddot{P}}_0 = - \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' \vec{\ddot{\theta}} - \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix} \vec{\ddot{\theta}}$$

Despejando el término de aceleración $\vec{\ddot{P}}_0$:

$$\vec{\ddot{P}}_0 = \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \left[- \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' \vec{\dot{P}}_0 - \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' \vec{\ddot{\theta}} - \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix} \vec{\ddot{\theta}} \right]$$

Reordenando:

$$\begin{aligned} \vec{\ddot{P}}_0 = & - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' \vec{\dot{P}}_0 \\ & - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' \vec{\ddot{\theta}} \\ & - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix} \vec{\ddot{\theta}} \end{aligned}$$

Se sabe que el jacobiano se calcula con la siguiente expresión:

$$J = - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}$$

Reemplazando y factorizando:

$$\vec{\dot{P}}_0 = - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' \vec{P}_0 - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' \vec{\theta} + J \vec{\theta}$$

Reordenando:

$$\vec{\dot{P}}_0 = - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} * \left[\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' \vec{P}_0 + \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' \vec{\theta} \right] + J \vec{\theta}$$

Se sabe que:

$$\vec{P}_0 = J \vec{\theta}$$

Reemplazando:

$$\vec{\dot{P}}_0 = - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} * \left[\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' J \vec{\theta} + \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' \vec{\theta} \right] + J \vec{\theta}$$

Factorizando:

$$\vec{\dot{P}}_0 = - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} * \left[\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' J + \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' \vec{\theta} \right] * \vec{\theta} + J \vec{\theta}$$

Agrupando términos para facilitar la codificaciones:

$$\vec{\dot{P}}_0 = - \begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}^{-1} * \left(\begin{bmatrix} \vec{s}_1^T \\ \vec{s}_2^T \\ \vec{s}_3^T \end{bmatrix}' J + K \right) * \vec{\theta} + J \vec{\theta} \quad (\text{A.60})$$

Donde:

$$K = \begin{bmatrix} \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 \end{bmatrix}' = \begin{bmatrix} \vec{s}_1^T \vec{b}_1 + \vec{s}_1^T \vec{b}_1 & 0 & 0 \\ 0 & \vec{s}_2^T \vec{b}_2 + \vec{s}_2^T \vec{b}_2 & 0 \\ 0 & 0 & \vec{s}_3^T \vec{b}_3 + \vec{s}_3^T \vec{b}_3 \end{bmatrix}$$

$$\vec{\dot{b}_i} = \begin{bmatrix} L_A \cos(\theta_i) \\ 0 \\ -L_A \sin(\theta_i) \end{bmatrix} \vec{\dot{\theta}_i}$$

$$\vec{\dot{s}_i} = \begin{bmatrix} \dot{P}_{0x} \\ \dot{P}_{0y} \\ \dot{P}_{0z} \end{bmatrix} + R_i^R * \begin{bmatrix} L_A \sin(\theta_i) \\ 0 \\ L_A \cos(\theta_i) \end{bmatrix} \dot{\theta}_i = \vec{\dot{P}_0} + \vec{b}_i \dot{\theta}_i$$

Para calcular la aceleración angular de los actuadores $\vec{\ddot{\theta}}$, se despeja desde la ecuación A.60 resultando:

$$\vec{\ddot{\theta}} = J^{-1} \left[\vec{\ddot{P}_0} + \begin{bmatrix} \vec{s_1}^T \\ \vec{s_2}^T \\ \vec{s_3}^T \end{bmatrix}^{-1} * \left(\begin{bmatrix} \vec{\dot{s}_1}^T \\ \vec{\dot{s}_2}^T \\ \vec{\dot{s}_3}^T \end{bmatrix} J + K \right) * \vec{\dot{\theta}} \right]$$

A.2.2. Modelación dinámica

A.2.2.1. Parámetros dinámicos

El modelo dinámico del robot consta de varios parámetros dinámicos. Los términos intermedios se definen para simplificar la representación del modelo dinámico.

Primero, la posición del centro de la masa del brazo viene dada por:

$$CoM = L_A \frac{\frac{1}{2} m_a + m_{codo} + 2 * r * m_b}{m_a + m_{codo} + 2 * r * m_b} \quad (\text{A.61})$$

En A.61, se elige que $r = 1/2$ para una distribución equitativa de la masa de los antebrazos como se explica en la hipótesis.

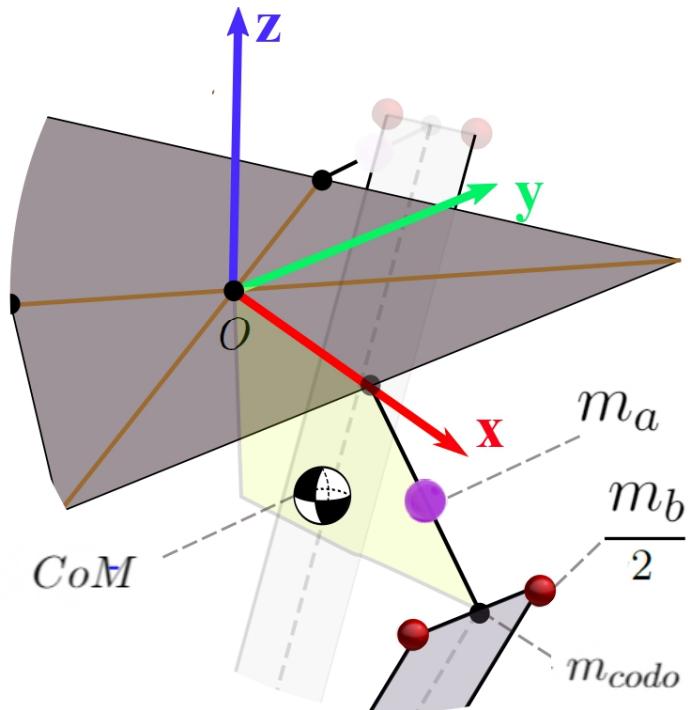


Figura A.10: Representación gráfica del centro de masas CoM

El segundo parámetro dinámico importante es la matriz de inercia en el espacio articular, dado en A.62:

$$I_b = \begin{bmatrix} I_{b1} & 0 & 0 \\ 0 & I_{b2} & 0 \\ 0 & 0 & I_{b3} \end{bmatrix} \quad (\text{A.62})$$

En A.62, I_{bi} es la inercia de cada brazo que es la suma de la inercia del motor I_m y la del brazo, como se expresa a continuación:

$$I_{bi} = I_m + L_A^2 \left(\frac{m_a}{3} + m_{codo} + 2 * r * m_b \right) \quad (\text{A.63})$$

Estos parámetros dinámicos se utilizan para derivar el modelo dinámico del robot paralelo delta.

A.2.2.2. Dinámica inversa

La parte del efecto final consta de la base móvil, la carga útil y las masas concentradas en las extremidades de los antebrazos. La masa de esta porción, m_{nt} , se expresa como:

$$m_{nt} = m_c + m_{payload} + 3 * 2 * (1 - r) m_b$$

donde, $m_{payload}$ es la masa de la carga útil, m_c es la masa de la base móvil y m_b es la masa de un par de barras paralelas y r es la relación de división de masas del antebrazo.

En el robot paralelo Delta, los únicos dos tipos de fuerzas que actúan sobre el efecto final son; la fuerza de gravedad \vec{F}_g y la fuerza de inercia \vec{F}_{in} debida a la aceleración:

$$\vec{F}_g = m_{nt} [00 - g]^T$$

$$\vec{F}_{in} = m_{nt} \vec{P}_0$$

Debido a la arquitectura del robot delta, la orientación del marco del efecto final es siempre paralela al marco de referencia $\{O - xyz\}$. Por lo tanto, se ignoran los términos de movimiento rotacional para el efecto final.

Como se explicó anteriormente, el principio del trabajo virtual se basa en el supuesto de que la contribución de todas las fuerzas iniciales debe ser igual a la contribución de todas las fuerzas no iniciales. El principio de la ecuación de trabajo virtual para el efecto final se puede escribir en su forma vectorial como:

$$\left(\vec{F}_g - m_{nt} \vec{P}_0 \right) * \delta r_E = 0 \quad (\text{A.64})$$

donde, g es la aceleración debida a la gravedad, \vec{P}_0 es el vector de aceleración del efecto final y δr_E es el desplazamiento virtual del efecto final.

Para resolver la dinámica del robot, el brazo es formado por un enlace de masa m_a y en sus extremidades se agrega las masas puntuales concentradas de las barras paralelas ($\frac{m_b}{2} + \frac{m_b}{2}$). Tres pares actúan sobre los brazos en cualquier momento: el par $\vec{\tau}_{Gb}$ debido a la gravedad que actúa sobre el centro de masa, el par $\vec{\tau} = [\tau_1, \tau_2, \tau_3]^T$ debido al actuador y el par $\vec{I}_b \vec{\theta}$ debido al momento de inercia I_b sobre el eje de rotación. La vector de aceleración angular de los actuadores se representa matricialmente $\vec{\ddot{\theta}} = [\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3]^T$. La ecuación de trabajo virtual del brazo se expresa de la siguiente manera:

$$\left(\vec{\tau} - I_b \vec{\ddot{\theta}} + \vec{\tau}_{Gb} \right) * \delta \theta = 0$$

Para obtener el para debido a la gravedad, primero se debe calcular el centro de masa CoM del brazo, que incluye las masas puntuales concentradas de las barras paralelas. Una vez determinado el centro de masa, se calcula el torque debido a las fuerzas de gravedad en el brazo con la siguiente expresión:

$$\vec{\tau}_{Gb} = g * CoM * (m_a + m_codo + 2 * r * m_b) [\cos(\theta_1) \cos(\theta_2) \cos(\theta_3)]^T \quad (\text{A.65})$$

Con los parámetros dinámicos para los componentes individuales del manipulador calculados, se puede desarrollar la dinámica completa del robot. Recordando que la suma de todo el trabajo virtual realizado en el sistema por todas las fuerzas y pares externos debe ser igual a cero, se suman las ecuaciones A.64 y A.65 para determinar el torque en los actuadores $\vec{\tau}$:

$$\left(\vec{F}_g - m_{nt}\vec{P}_0\right) * \delta r_E + \left(\vec{\tau} - I_b \vec{\dot{\theta}} + \vec{\tau}_{Gb}\right) * \delta\theta = 0$$

Reemplazando la contribución del desplazamiento $\delta r_E = J^T \delta\theta$ en los actuadores:

$$\left(\vec{F}_g - m_{nt}\vec{P}_0\right) * J\delta\theta + \left(\vec{\tau} - I_b \vec{\dot{\theta}} + \vec{\tau}_{Gb}\right) * \delta\theta = 0$$

Reordenando matricialmente para simplificar la ecuación anterior se llega a que:

$$J^T \vec{F}_g - J^T m_{nt} \vec{P}_0 + \vec{\tau} + -I_b \vec{\dot{\theta}} + \vec{\tau}_{Gb} = 0$$

$$\vec{\tau} = I_b \vec{\dot{\theta}} + J^T m_{nt} \vec{P}_0 - J^T \vec{F}_g - \vec{\tau}_{Gb}$$

Sustituyendo $\vec{P}_0 = J \vec{\dot{\theta}} + \vec{J} \vec{\ddot{\theta}}$

$$\vec{\tau} = I_b \vec{\dot{\theta}} + J^T m_{nt} \left(J \vec{\dot{\theta}} + \vec{J} \vec{\ddot{\theta}} \right) - J^T \vec{F}_g - \vec{\tau}_{Gb}$$

$$\vec{\tau} = \left(I_b + J^T m_{nt} J \right) \vec{\dot{\theta}} + \left(J^T m_{nt} \vec{J} \right) \vec{\ddot{\theta}} + \left(-J^T \vec{F}_g - \vec{\tau}_{Gb} \right)$$

A partir de (), podemos identificar fácilmente la matriz de masa $M(\theta)$, la matriz $C(\theta, \dot{\theta})$ de coeficiente de Coriolis y centrífuga, y el vector de términos de gravedad $\vec{G}(\theta)$ comparándolo con la dinámica estándar:

$$\vec{\tau} = M(\theta) \vec{\dot{\theta}} + C(\theta, \dot{\theta}) \vec{\ddot{\theta}} + \vec{G}(\theta)$$

Dónde

$$M(\theta) = I_b + J^T m_{nt} J$$

$$C(\theta, \dot{\theta}) = J^T m_{nt} \vec{J}$$

$$\vec{G}(\theta) = -J^T \vec{F}_g - \vec{\tau}_{Gb}$$

Apéndice B

Carpetas, comandos y códigos

Se muestra en la figura B.1 todas las carpetas donde esta la configuración en ROS y RVIZ. Los archivos creados en esta tesis que están contenidos en estas carpetas principalmente: son los algoritmos en lenguaje python de los metodos A, metodo B, espacio de trabajo y trayectorias, los mensajes que se utilizan en los temas de ROS, la configuración URDF del robot delta y la configuracion de la vizualizacion en Rviz. Los archivos se pueden descargar del siguiente repositorio en GitHub: https://github.com/IvanFernandezGracia/delta_robot_tesis.

B.1. Carpetas creadas por catkin make

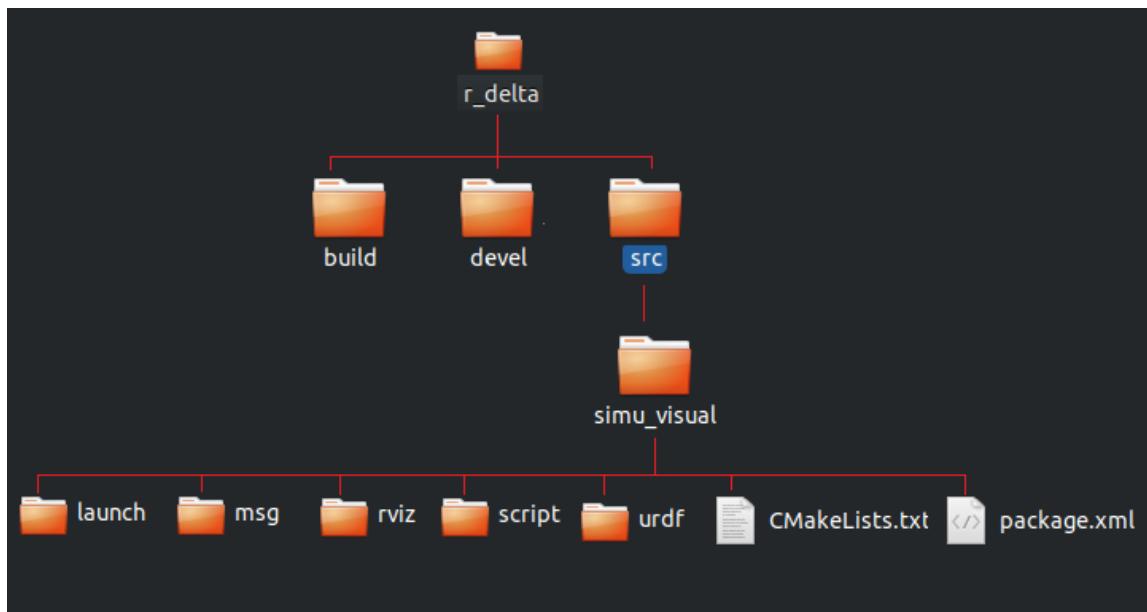


Figura B.1: Espacio de trabajo creado por catkin_make para los algoritmos de los métodos A y B

B.2. Comandos para compilar nodos en shell

- Transformar archivos de nodos a archivos python ejecutables

```
chmod +x nombre_py_nodo_ejecutable.py
```

- Archivos python que son nodos

Trayectoria y Torques: path_tm1_v2_adams.py

Espacio de trabajo: workspace_v2.py

Visualizacion: posicionador_rviz_realtime_tm1_adams.py

- Shell y comandos necesarios para calcular torques de trayectorias

```
----- [shell 0]: Para todos los shell
cd r_delta
source devel/setup.bash

----- [shell 1]: ROS Master
roscore

----- [shell 2]: Abrir Rviz
roslaunch simu_visual rviz_tm1_adams.launch

----- [shell 3]: Nodo visualizacion
rosrun simu_visual posicionador_rviz_realtime_tm1_adams.py

----- [shell 4]: Herramienta que grafica nodos iniciados
rosrun rqt_graph rqt_graph

----- [shell 5]: Nodo de Trayectoria y Torques
rosrun simu_visual path_tm1_v2_adams.py

----- [shell 6]: Mensaje para el nodo torques y trayectorias (1)
rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: -300.0, yo: 0.0, zo: -450.0,
xf: 300.0, yf: 150.0, zf: -750.0,
vmax: 2000.0, amax: 40000.0,
paso1: 50, paso2: 150,
ls_fin: true, idcall: 1, num_tray: 1}""

----- [shell 6]: Mensaje para el nodo torques y trayectorias (2)
rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: -300.0, yo: -150.0, zo: -450.0,
xf: 300.0, yf: 150.0, zf: -750.0,
vmax: 2000.0, amax: 40000.0,
paso1: 50, paso2: 150,
```

```

ls_fin: true, idcall: 1, num_tray: 2}"
```

----- [shell 6]: Mensaje para el nodo torques y trayectorias (3)

```

rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: 300.0, yo: -150.0, zo: -450.0,
xf: -300.0, yf: 150.0, zf: -750.0,
vmax: 2000.0, amax: 40000.0,
paso1: 50, paso2: 150,
ls_fin: true, idcall: 1, num_tray: 3}"
```

----- [shell 6]: Mensaje para el nodo torques y trayectorias (4)

```

rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: 400.0, yo: 150.0, zo: -450.0,
xf: -400.0, yf: 150.0, zf: -450.0,
vmax: 2000.0, amax: 40000.0,
paso1: 50, paso2: 150,
ls_fin: true, idcall: 1, num_tray: 4}"
```

----- [shell 6]: Mensaje para el nodo torques y trayectorias (5)

```

rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: -300.0, yo: 0.0, zo: -450.0,
xf: 300.0, yf: 150.0, zf: -750.0,
vmax: 200.0, amax: 10000.0,
paso1: 50, paso2: 150,
ls_fin: true, idcall: 1, num_tray: 5}"
```

----- [shell 6]: Mensaje para el nodo torques y trayectorias (6)

```

rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: -300.0, yo: -150.0, zo: -450.0,
xf: 300.0, yf: 150.0, zf: -750.0,
vmax: 200.0, amax: 10000.0,
paso1: 50, paso2: 150,
ls_fin: true, idcall: 1, num_tray: 6}"
```

----- [shell 6]: Mensaje para el nodo torques y trayectorias (7)

```

rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: 300.0, yo: -150.0, zo: -450.0,
xf: -300.0, yf: 150.0, zf: -750.0,
vmax: 200.0, amax: 10000.0,
paso1: 50, paso2: 150,
ls_fin: true, idcall: 1, num_tray: 7}"
```

----- [shell 6]: Mensaje para el nodo torques y trayectorias (8)

```

rostopic pub -1 /input_ls_final simu_visual/linear_speed_xyz
"{"xo: 400.0, yo: 150.0, zo: -450.0,
xf: -400.0, yf: 150.0, zf: -450.0,
vmax: 200.0, amax: 10000.0,
paso1: 50, paso2: 150,
```

```
ls_fin: true, idcall: 1, num_tray: 8}"
```

- Espacio de Trabajo

```
----- [shell 0]: Para todos los shell
cd r_delta
source devel/setup.bash
----- [shell 1]: Nodo espacio de trabajo
rosrun simu_visual workspace_v2.py
----- [shell 2]: Mensaje para tema de nodo espacio de trabajo
rostopic pub -1 /input_workspace simu_visual/parameter_ws
"graficar_realtime: true
step: 10
idcall: 1"
```

B.3. CMakeLists.txt (/simu_visual)

```
cmake_minimum_required(VERSION 2.8.3)
project(simu_visual)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS
    roscpp
    rospy
    std_msgs
    sensor_msgs
    message_generation
    genmsg
    urdf)

## Generate messages in the 'msg' folder
add_message_files(
    FILES
    parameter_ws.msg
    linear_speed_xyz.msg
    matriz_path_ls.msg
)

## Generate added messages and services with any dependencies listed here
generate_messages(
    DEPENDENCIES
    std_msgs
)

catkin_package(
    INCLUDE_DIRS include
    LIBRARIES simu_visual
    CATKIN_DEPENDS roscpp rospy std_msgs message_runtime
    #DEPENDS system_lib
)

include_directories(
# include
    ${catkin_INCLUDE_DIRS}
)
```

B.4. Package.xml (/simu_visual)

```
<?xml version="1.0"?>
<package format="2">
  <name>simu_visual</name>
  <version>0.0.0</version>
  <description>The simu_visual package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <maintainer email="ivan.fernandez.g@usach.cl">ivan</maintainer>

  <!-- Commonly used license strings: -->
  <license>BSD</license>

  <!-- Use build_depend for packages you need at compile time: -->
  <build_depend>roscpp</build\_\_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>message_generation</build_depend>

  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <build_export_depend>roscpp</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>std_msgs</build_export_depend>
  <build_export_depend>message_generation</build_export_depend>

  <!-- Use buildtool_depend for build tool packages: -->
  <buildtool_depend>catkin</buildtool_depend>

  <!-- Use exec_depend for packages you need at runtime: -->
  <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>sensor_msgs</exec_depend>
  <exec_depend>message_runtime</exec_depend>

  <export>
  </export>
</package>
```

B.5. Mensajes (/msg)

B.5.1. linear_speed_xyz.msg

```
float32 xo
float32 yo
float32 zo
float32 xf
float32 yf
float32 zf
float32 vmax
float32 amax
int64 paso1
int64 paso2
bool ls_fin
int64 idcall
int64 num_tray
```

B.5.2. matriz_path_ls.msg

```
bool permiso
int64 id_call
float32[] x
float32[] y
float32[] z
float32[] th1
float32[] th2
float32[] th3
float32[] tiempo
```

B.5.3. parameter_ws.msg

```
bool graficar_realtime
int64 step
int64 idcall
```

B.6. robot_urdfm1_adams.urdf (/urfd)

```
<?xml version="1.0"?>

<robot name="Delta_robot">

    <!--BASE SUPERIOR: ----- -->
    <link name="base_link">
        <visual>
            <geometry>
                <cylinder length="0.005" radius="0.210"/> <!--f/2-->
            </geometry>
            <material name="gris">
                <color rgba="0.55 0.55 0.55 0.5"/>
            </material>
        </visual>
    </link>

    <!--BRAZOS SUPERIORES: ----- -->
    <!--BRAZO_1-->
    <link name="brazo1">
        <visual>
            <origin xyz="0.310 0 0" rpy="0 0 0"/> <!--la = 150-->
            <geometry>
                <box size="0.620 0.003 0.003"/> <!--la = 150-->
            </geometry>
            <material name="blanco">
                <color rgba="0.9 0.9 0.9 1"/>
            </material>
        </visual>
    </link>

    <joint name="base_brazo1" type="revolute">
        <axis xyz="0 1 0"/>
        <limit effort="100.0" lower="0.0" upper="1.8" velocity="0.5"/>
        <parent link="base_link"/>
        <child link="brazo1"/>
        <origin xyz="-0.181865335 0.105 0" rpy="0 0 2.61799"/>
    </joint>

    <!--BRAZO_2-->
    <link name="brazo2">
        <visual>
            <origin xyz="0.310 0 0" rpy="0 0 0"/>
            <geometry>
```

```

        <box size="0.620 0.003 0.003"/> <!--la = 150-->
    </geometry>
    <material name="blanco"/>
</visual>
</link>

<joint name="base_brazo2" type="revolute">
    <axis xyz="0 1 0"/>
    <limit effort="100.0" lower="0.0" upper="1.8" velocity="0.5"/>
    <parent link="base_link"/>
    <child link="brazo2"/>
    <origin xyz="0 -0.210 0" rpy="0 0 -1.57075"/>
</joint>

<!--BRAZO_3-->
<link name="brazo3">
    <visual>
        <origin xyz="0.310 0 0" rpy="0 0 0"/>
        <geometry>
            <box size="0.620 0.003 0.003"/> <!--la = 150-->
        </geometry>
        <material name="blanco"/>
    </visual>
</link>
<joint name="base_brazo3" type="revolute">
    <axis xyz="0 1 0"/>
    <limit effort="100.0" lower="0.0" upper="1.8" velocity="0.5"/>
    <parent link="base_link"/>
    <child link="brazo3"/>
    <origin xyz="0.181865335 0.105 0" rpy="0 0 0.523599"/>
</joint>

<!--BRAZOS INFERIORES: ----- -->

<!--Ante_BRAZO_1-->
<link name="barra1_a"/>
<joint name="codo1_a" type="revolute">
    <axis xyz="0 -1 0"/>
    <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
    <parent link="brazo1"/>
    <child link="barra1_a"/>
    <origin xyz="0.620 0 0" rpy="0 0 0"/>
</joint>

<link name="barra1_b">
    <visual>
        <origin xyz="-0.440 0 0" rpy="0 0 0"/> <!--0.175-->
        <geometry>

```

```

        <box size="0.880 0.003 0.003"/> <!--lb = 205-->
    </geometry>
    <material name="blanco"/>
</visual>
</link>

<joint name="codo1_b" type="revolute">
<axis xyz="0 0 1"/>
<limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
<parent link="barra1_a"/>
<child link="barra1_b"/>
<origin xyz="0 0 0" rpy="0 0 0"/>
</joint>

<!--Ante_BRAZO_2-->

<link name="barra2_a"/>
<joint name="codo2_a" type="revolute">
<axis xyz="0 -1 0"/>
<limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
<parent link="brazo2"/>
<child link="barra2_a"/>
<origin xyz="0.620 0 0" rpy="0 0 0"/> <!--la = 150-->
</joint>

<link name="barra2_b">
<visual>
<origin xyz="-0.440 0 0" rpy="0 0 0"/> <!--lb = 205 / 2 -->
<geometry>
<box size="0.880 0.003 0.003"/> <!--lb = 205-->
</geometry>
<material name="blanco"/>
</visual>
</link>

<joint name="codo2_b" type="revolute">
<axis xyz="0 0 1"/>
<limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
<parent link="barra2_a"/>
<child link="barra2_b"/>
<origin xyz="0 0 0" rpy="0 0 0"/> <!--02 = 62.0652 -107.5000 0-->
</joint>

<!--Ante_BRAZO_3-->
<link name="barra3_a"/>
<joint name="codo3_a" type="revolute">
<axis xyz="0 -1 0"/>
<limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>

```

```

<parent link="brazo3"/>
<child link="barra3_a"/>
<origin xyz="0.620 0 0" rpy="0 0 0"/> <!--02 = 62.0652 -107.5000 0-->
</joint>

<link name="barra3_b">
  <visual>
    <origin xyz="-0.440 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.880 0.003 0.003"/> <!--1b = 205-->
    </geometry>
    <material name="blanco"/>
  </visual>
</link>

<joint name="codo3_b" type="revolute">
  <axis xyz="0 0 1"/>
  <limit effort="100.0" lower="-3.14" upper="3.14" velocity="0.5"/>
  <parent link="barra3_a"/>
  <child link="barra3_b"/>
  <origin xyz="0 0 0" rpy="0 0 0"/> <!--02 = 62.0652 -107.5000 0-->
</joint>

<!--ACTUADOR: ----- -->

<link name="actuador">
  <visual>
    <origin xyz="0 0 0" rpy="0 -1.57075 0"/>
    <geometry>
      <cylinder length="0.001" radius="0.050"/>
    </geometry>
    <material name="colorte">
      <color rgba="0.15 0.55 0.95 0.3"/>
    </material>
  </visual>
</link>

<link name="aux1"/>
<link name="aux2"/>

<joint name="act_x" type="prismatic">
  <parent link="base_link"/>
  <child link="aux1"/>
  <limit effort="100.0" lower="-1" upper="1" velocity="0.5"/>
  <origin rpy="0 0 0" xyz="0 0 0"/>
</joint>
```

```

<joint name="act_y" type="prismatic">
  <parent link="aux1"/>
  <child link="aux2"/>
  <limit effort="100.0" lower="-1" upper="1" velocity="0.5"/>
  <origin rpy="0 0 1.57075" xyz="0 0 0"/>
</joint>

<joint name="act_z" type="prismatic">
  <parent link="aux2"/>
  <child link="actuador"/>
  <limit effort="100.0" lower="0" upper="1" velocity="0.5"/>
  <origin rpy="0 -1.57075 0" xyz="0 0 0"/>
</joint>
</robot>

```

B.7. rviz_tm1_adams.launch (/launch)

```

<launch>
  <!-- comentarios -->
  <arg name="model"/>
  <arg name="gui" default="true"/>
  <param name="robot_description"
        textfile="$(find simu_visual)/urdf/robot_urdf_tm1_adams.urdf"
        />
  <param name="use_gui" value="$(arg gui)"/>
  <node name="estados_robot_pub"
        pkg="robot_state_publisher" type="robot_state_publisher"/>
  <node name="rviz" pkg="rviz" type="rviz"
        args="d $(find simu_visual)/rviz/tfm_simulacion.rviz"
        required="true"/>
</launch>

```

B.8. Python (/script)

B.8.1. pd_tm1_adams.py

```
1  # Nombre Creador: Ivan Alejandro Fernandez Gracia
2  # Universidad: Universidad de Santiago de Chile
3  # Mail: ivan.fernandez.g@usach.cl
4  # Objetivo: Ingresar parametros para calcular el
5  # torque de los actuadores de un robot delta
6
7  # Importar Librerias
8  import math
9
10 #####
11 ##### Cambio Unidades #####
12 #####
13 #####
14 def dtr():
15     valor = math.pi / 180.0  # Grados a Radianes
16     return valor
17
18
19 def rtd():
20     valor = 180.0 / math.pi  # Radianes a Grados
21     return valor
22
23
24 def mm2m():
25     valor = ((10.0) ** (-3))  # Milimetros a Metros
26     return valor
27
28
29 def mt2mm():
30     valor = ((10.0) ** (3))  # Metros a Milimetros
31     return valor
32
33
34 def kgm2tgrmm2():
35     valor = ((10.0) ** (3 + 6))  # Kilogramo metro^2 a gramo mm^2
36     return valor
37
38
39 #####
40 ##### Trigonometria #####
41 #####
42 def sqrt3():
43     valor = math.sqrt(3.0)
44     return valor
```

```

45
46
47 def pi():
48     valor = math.pi # [rad]
49     return valor
50
51
52 def sin120():
53     valor = sqrt3() / 2.0
54     return valor
55
56
57 def cos120():
58     valor = -0.5
59     return valor
60
61
62 def tan60():
63     valor = sqrt3()
64     return valor
65
66
67 def sin30():
68     valor = 0.5
69     return valor
70
71
72 def tan30():
73     valor = 1.0 / sqrt3()
74     return valor
75
76
77 def cos30():
78     valor = (sqrt3()) / 2.0
79     return valor
80
81
82 ##### Dimensionamiento Geometrico Robot Delta #####
83 ##### Dimensionamiento Geometrico Robot Delta #####
84 ##### Dimensionamiento Geometrico Robot Delta #####
85 def l2():
86     valor = (880.0) * ((10.0) ** (-3)) # Longitud Antebrazo [m]
87     return valor
88
89
90 def l1():
91     valor = (620.0) * ((10.0) ** (-3)) # Longitud Brazo [m]
92     return valor

```

```

93
94
95 def rb():
96     valor = (50.0) * ((10.0) ** (-3)) # Radio plataforma móvil [m]
97     return valor
98
99
100 def ra():
101    valor = (210.0) * ((10.0) ** (-3)) # Radio base fija [m]
102    return valor
103
104
105 def e():
106    valor = (2.0 * rb()) / (tan30()) # [m]
107    return valor
108
109
110 def f():
111    valor = (2.0 * ra()) / (tan30()) # [m]
112    return valor
113
114
115 def hf():
116    valor = math.sqrt(0.75 * (f() ** 2)) # [m]
117    return valor
118
119
120 def he():
121    valor = math.sqrt(0.75 * (e() ** 2)) # [m]
122    return valor
123
124
125 ##### Masas e Inercias #####
126 ##### Masa del Brazo #####
127 #####
128 def m1():
129    valor = (2213.0) * ((10.0) ** (-3)) # Masa Brazo [kg]
130    return valor
131
132
133 def m_elbow():
134    valor = (0.0) * ((10.0) ** (-3)) # Masa juntas [kg]
135    return valor
136
137
138 def m2():
139    valor = (1315.0 / 2.0) * ((10.0) ** (-3)) # Masa de una varilla del antebrazo [kg]
140    return valor

```

```

141
142
143 def mp():
144     valor = (510.0) * ((10.0) ** (-3)) # Masa plataforma móvil [kg]
145     return valor
146
147
148 def inercia_m():
149     valor = (0.0) * ((10.0) ** (-3)) # Inercia actuadores [kg * m2]
150     return valor
151
152
153 def gg():
154     valor = (9.81) # Gravedad [m/s2]
155     return valor
156
157
158 def r_mass():
159     valor = (0.5) # Relación de masas del antebrazo
160     return valor
161
162
163 def com():
164     valor = (l1()) * (((0.5 * (m1())) + (m_elbow()) + ((2) * (r_mass()) * (m2())))) /
165                         (((1.0 * (m1())) + (m_elbow()) + ((2) * (r_mass()) * (m2()))))
166     # Centro de masas brazo
167     return valor
168
169 #####
170 ##### Restricciones Espacio de Trabajo #####
171 #####
172 #####
173 # Restricciones límites de ángulos de motores [grados]
174 def res_ang_min():
175     valor = int(-90)
176     # valor=int(-30)
177     return valor
178
179
180 def res_ang_max():
181     valor = ((res_ang_min()) * -1) + 1
182     # valor=90
183     return valor
184
185
186 # Restricción ángulos 2 y 3 de cada brazo [grados]
187 def theta2i_min():

```

```

188     valor = 5
189     return valor
190
191
192     def theta2i_max():
193         valor = 180 - theta2i_min()
194         return valor
195
196
197     def theta3i_min():
198         valor = 45
199         return valor
200
201
202     def theta3i_max():
203         valor = 180 - theta3i_min()
204         return valor
205
206
207 # Restricciones Singularidad Jacobiano
208 def err_jxx():
209     valor = (6) * (10 ** (-1))
210     return valor
211
212
213 def err_jinv():
214     valor = (4) * (10 ** (-3))
215     return valor
216
217
218 # Restriccion del fabricante (caja) [mm]
219 def px_max_ws():
220     valor = 400
221     return valor
222
223
224 def px_min_ws():
225     valor = (-1) * px_max_ws()
226     return valor
227
228
229 def py_max_ws():
230     valor = 400
231     return valor
232
233
234 def py_min_ws():
235     valor = (-1) * py_max_ws()

```

```
236     return valor
237
238
239 def pz_max_ws():
240     valor = -300
241     return valor
242
243
244 def pz_min_ws():
245     valor = -750
246     return valor
```

B.8.2. path_tm1_v2_adams.py

```
1  #!/usr/bin/env python
2  # Nombre Creador: Ivan Alejandro Fernandez Gracia
3  # Universidad: Universidad de Santiago de Chile
4  # Mail: ivan.fernandez.g@usach.cl
5  # Objetivo: Comparar los torques calculados por
6  # los algoritmos del metodo A, metodo B y en ADAMS.
7
8  ##### [Importar Librerias] #####
9
10 #####
11 import numpy as np
12 import random
13 import plot_delta
14 import rospy
15 import trans_rot_ls_adams
16 import linear_speed_f_adams
17 import delta_kinematics_t1m_adams
18 import delta_kinematics_Paderborn_tm1_adams
19 import jacobian_tm1_adams
20 import jacobian_Paderborn_tm1_v2_adams
21 import torque_m1_adams
22 import torque_m1_Paderborn_v2_adams
23
24 #####
25 ##### [Importar mensajes] #####
26 #####
27 from simu_visual.msg import matriz_path_ls
28 from simu_visual.msg import linear_speed_xyz
29
30
31 #####
32 ##### [ Nodo ] #####
33 #####
34 def nodo():
35     ##### {{ callback_linear_speed_xyz(data) }} #####
36     global permiso_2
37     global id_call_2
38     global call_xo_2
39     global call_yo_2
40     global call_zo_2
41     global call_xf_2
42     global call_yf_2
43     global call_zf_2
44     global call_vmax_2
45     global call_amax_2
46     global call_pas1_2
```

```

47 global call_pas2_2
48 global num_tray_2
49
50 ##### {{ Variables verifica mensaje entrante }} #####
51 permiso_2 = False
52 id_call_2 = 0
53 id_permiso_2 = 0
54
55 ##### {{ Iniciar Nodo ROS }} #####
56 rospy.init_node("torque_metodo_1", anonymous=False)
57 rate = rospy.Rate(7.8125) # Hz
58
59 ##### {{ Publisher }} #####
60 pub1 = rospy.Publisher("m_txyzth123", matriz_path_ls, queue_size=10)
61
62 while not rospy.is_shutdown():
63 ##### {{ Topics }} #####
64 rospy.Subscriber("input_ls_final", linear_speed_xyz, callback_linear_speed_xyz)
65
66 if (permiso_2 == True and id_call_2 != id_permiso_2 and (
67     call_xo_2 != call_xf_2 or call_yo_2 != call_yf_2 or call_zo_2 != call_zf_2) and (
68     call_vmax_2 > 0 and call_amax_2 > 0)):
69 rospy.loginfo("Creando Torque M1 Trayectoria i=1,2,3!")
70
71 # ****
72 # ***** {{ Crear Trayectoria }} *****
73 # ****
74
75 ##### {{ Rotacion punto final e inicial }} #####
76 resultados_2 = trans_rot_ls_adams.path_linear_speed(call_xo_2, call_yo_2, call_zo_2,
77     call_xf_2, call_yf_2, call_zf_2)
78 ##### {{ Perfil trapezoidal de velocidad }} #####
79 resultados_3 = linear_speed_f_adams.ls_v_a_total(0, resultados_2[0],
80     call_vmax_2, call_amax_2,
81     call_pas1_2, call_pas2_2)
82 ##### {{ Rotacion inversa punto final, inicial y trayectoria}} #####
83 resultados_4 = trans_rot_ls_adams.path_linear_speed_inv(resultados_3[0],
84     resultados_3[1], resultados_3[2], resultados_3[3],
85     resultados_2[1], resultados_2[2],
86     resultados_2[3], resultados_2[4],
87     resultados_2[5])
88 ##### {{ Guardar en Matriz Trayectoria espacio cartesiano XYZ }} #####
89 res_1 = [resultados_4[0],
90         resultados_4[1], resultados_4[4], resultados_4[7],
91         resultados_4[2], resultados_4[5], resultados_4[8],
92         resultados_4[3], resultados_4[6], resultados_4[9]]
93
94 # ****

```

```

95  # ***** {{ Metodo A }} ****
96  # ****
97
98  ##### {{ Cinematica Inversa }} #####
99  res_2 = delta_kinematics_t1m_adams.inverse_m(res_1[1], res_1[4], res_1[7])
100 ##### {{ Visualizacion Trayectoria Rviz}} #####
101 enviar = matriz_path_ls()
102 enviar.x = res_1[1]
103 enviar.y = res_1[4]
104 enviar.z = res_1[7]
105 enviar.th1 = res_2[0]
106 enviar.th2 = res_2[1]
107 enviar.th3 = res_2[2]
108 enviar.tiempo = res_1[0] * 10.0
109 enviar.permiso = bool(1)
110 enviar.id_call = random.randrange(10)
111 pub1.publish(enviar)
112 n_1 = len(res_1[1])
113 ##### {{ Guardar Trayectoria }} #####
114 path_root = "/home/ivan/r_delta/src/simu_visual/script/"
115 path_root_tray = "resultado_trayectorias/Tray_"
116 mi_path1 = path_root + path_root_tray + str(num_tray_2) + "_XYZ.txt"
117 mi_path2 = path_root + path_root_tray + str(num_tray_2) + "_Theta.txt"
118 mi_path3 = path_root + path_root_tray + str(num_tray_2) + "_Tiempo.txt"
119 fic1 = open(mi_path1, "w")
120 data1 = np.append(np.append(res_1[1], res_1[4]), res_1[7])
121 for line in data1:
122     fic1.write(str(line * (0.001)))
123     fic1.write("\n")
124 fic1.close()
125 fic2 = open(mi_path2, "w")
126 data2 = np.append(np.append(res_2[0], res_2[1]), res_2[2])
127 for line in data2:
128     fic2.write(str(line * ((np.pi) / 180)))
129     fic2.write("\n")
130 fic2.close()
131 fic3 = open(mi_path3, "w")
132 for line in res_1[0]:
133     fic3.write(str(line))
134     fic3.write("\n")
135 fic3.close()
136 ##### {{ Velocidad y Aceleracion Angular }} #####
137 res_3 = jacobian_tm1_adams.jacobian_total_macel(res_1[1], res_1[4], res_1[7],
138                                                 res_1[2], res_1[5], res_1[8],
139                                                 res_1[3], res_1[6], res_1[9],
140                                                 res_2[0], res_2[1], res_2[2])
141 ##### {{ Fuerza Externa en el Efecto }} #####
142 res_4 = [(res_1[1]) * 0, (res_1[1]) * 0, (res_1[1]) * 0]

```

```

143 ##### {{ Torques }} #####
144 res_5 = torque_m1_adams.ti_matriz(res_3[3], res_3[4], res_3[5],
145     res_2[0], res_2[1], res_2[2],
146     res_1[1], res_1[4], res_1[7],
147     res_1[3], res_1[6], res_1[9],
148     res_4[0], res_4[1], res_4[2],
149     0.0)
150 ##### {{ Guardar Torque Metodo A }} #####
151 path_root_t = "resultados_torque_metodo_a/Tray_"
152 mi_path11 = path_root + path_root_t + str(num_tray_2) + "_Torque1.txt"
153 mi_path22 = path_root + path_root_t + str(num_tray_2) + "_Torque2.txt"
154 mi_path33 = path_root + path_root_t + str(num_tray_2) + "_Torque3.txt"
155 fic11 = open(mi_path11, "w")
156 for line in res_5[0]:
157     fic11.write(str(line))
158     fic11.write("\n")
159 fic11.close()
160 fic22 = open(mi_path22, "w")
161 for line in res_5[1]:
162     fic22.write(str(line))
163     fic22.write("\n")
164 fic22.close()
165 fic33 = open(mi_path33, "w")
166 for line in res_5[2]:
167     fic33.write(str(line))
168     fic33.write("\n")
169 fic33.close()
170
171 # ****
172 # ***** {{ Metodo B }} ****
173 # ****
174
175 ##### {{ Cinematica Inversa }} #####
176 res_6 = delta_kinematics_Paderborn_tm1_adams. \
177     inverse_Paderborn_m(res_1[1], res_1[4], res_1[7])
178 ##### {{ Visualizacion Trayectoria Ruiz}} #####
179 enviar = matriz_path_ls()
180 enviar.x = res_1[1]
181 enviar.y = res_1[4]
182 enviar.z = res_1[7]
183 enviar.th1 = res_6[0]
184 enviar.th2 = res_6[1]
185 enviar.th3 = res_6[2]
186 enviar.tiempo = res_1[0] * 1.0
187 enviar.permiso = bool(1)
188 enviar.id_call = random.randrange(10)
189 pub1.publish(enviar)
190 ##### {{ Velocidad y Aceleracion Angular }} #####

```

```

191     res_10 = jacobian_Paderborn_tm1_v2_adams.\n
192         jacobian_total_macel(res_1[1], res_1[4], res_1[7],\n
193             res_1[2], res_1[5], res_1[8],\n
194             res_6[0], res_6[1], res_6[2],\n
195             res_1[3], res_1[6], res_1[9])\n
196     ##### {{ Fuerza Externa en el Efecto }} #####\n
197     res_8 = [(res_1[1]) * 0, (res_1[1]) * 0, (res_1[1]) * 0]\n
198     ##### {{ Torques }} #####\n
199     res_11 = torque_m1_Paderborn_v2_adams.ti_matriz(res_10[5], res_10[6], res_10[7],\n
200         res_6[0], res_6[1], res_6[2],\n
201         res_1[1], res_1[4], res_1[7],\n
202         res_8[0], res_8[1], res_8[2],\n
203         res_10[3], 0.0, res_10[4],\n
204         res_10[0], res_10[1], res_10[2])\n
205\n
206     # ****\n207     # *** {{ Compara Torques Metodo A ,Metodo B y ADAMS}} ***\n208     # ****\n209\n210     ##### {{ Leer Torque ADAMS }} #####\n211     path_root_t_adams="resultados_torque_adams/adams_\n212     mi_path44 = path_root + path_root_t_adams + str(num_tray_2) + ".txt"\n213     data_numpi = np.loadtxt(mi_path44, delimiter='\t')\n214     ##### {{ Plotear posicion, velocidad y aceleracion angular }} #####\n215     plot_delta.comparar_angles(\n216         [res_1[0],\n217             res_2[0], res_3[0], res_3[3],\n218             res_2[1], res_3[1], res_3[4],\n219             res_2[2], res_3[2], res_3[5],\n220             res_6[0], res_10[0], res_10[5],\n221             res_6[1], res_10[1], res_10[6],\n222             res_6[2], res_10[2], res_10[7]])\n223     ##### {{ Plotear Torques }} #####\n224     plot_delta.torque_adams(res_1[0],\n225         res_5[0], res_5[1], res_5[2],\n226         res_11[0], res_11[1], res_11[2],\n227         data_numpi[:, 1] * (-0.001), data_numpi[:, 2] * (-0.001),\n228         data_numpi[:, 3] * (-0.001),\n229         data_numpi[:, 0])\n230\n231     # ****\n232     # *** {{ Reset mensaje entrante}} ***\n233     # ****\n234     permiso_2 = False\n235     id_permiso_2 = id_call_2\n236\n237     rate.sleep()\n238     return

```

```

239
240
241 ##### [ Callback ] #####
242 ##### [ Callback ] #####
243 #####
244 def callback_linear_speed_xyz(data):
245     global permiso_2
246     global id_call_2
247     global call_xo_2
248     global call_yo_2
249     global call_zo_2
250     global call_xf_2
251     global call_yf_2
252     global call_zf_2
253     global call_vmax_2
254     global call_amax_2
255     global call_pas1_2
256     global call_pas2_2
257     global num_tray_2
258
259     call_xo_2 = data.xo
260     call_yo_2 = data.yo
261     call_zo_2 = data.zo
262     call_xf_2 = data.xf
263     call_yf_2 = data.yf
264     call_zf_2 = data.zf
265     call_vmax_2 = data.vmax
266     call_amax_2 = data.amax
267     call_pas1_2 = data.paso1
268     call_pas2_2 = data.paso2
269     permiso_2 = data.ls_fin
270     id_call_2 = data.idcall
271     num_tray_2 = data.num_tray
272
273
274 ##### [ Main Funtion] #####
275 ##### [ Main Funtion] #####
276 #####
277 if __name__ == "__main__":
278     try:
279         nodo()
280     except rospy.ROSInterruptException:
281         pass
282
283 ##### FIN #####

```

B.8.3. linear_speed_f_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo: Linear segment with parabolic blend (LSPB)
5 # Objetivo: Perfil trapezoidal de velocidad
6
7 ##### [Importar Librerias] #####
8 ##### [Parametros] #####
9 #####
10 import math
11 import pd_tm1_adams
12 import numpy as np
13 from numpy.linalg import inv
14 import matplotlib.pyplot as plt
15 from mpl_toolkits.mplot3d import Axes3D
16
17 ##### [Linear Speed LSPB (Total)] #####
18 ##### [Parametros Curva] #####
19 #####
20 mtmm = pd_tm1_adams.mtmm()
21 mmtm = pd_tm1_adams.mmtm()
22
23 #####
24 ##### [Linear Speed LSPB (Total)] #####
25 ##### [Parametros Curva] #####
26 #####
27 # q0 y q1 en [metros]
28 # vmax [metros/segundos]
29 # amax [metros / segundos^2]
30 # pas 1 y pas 2 son numero enteros
31 def ls_v_a_total(q0, q1, vmax, amax, pas_1, pas_2):
32     ##### {{ Cambio unidades SI }} #####
33     q0 *= mmtm
34     q1 *= mmtm
35     vmax *= mmtm
36     amax *= mmtm
37
38     if (pas_1 <= 0):
39         pas_1 = 2
40     if (pas_2 <= 0):
41         pas_2 = 2
42
43     ##### {{ Parametros Curva }} #####
44     tau = vmax / amax
45     s = np.sign((q1 - q0))
46     T = (s * (((q1) - (q0)) / (vmax))) + (tau)
```

```

47  paso1 = (tau) / pas_1
48  paso2 = (T - (2 * tau)) / pas_2
49  pas_total = pas_1 + pas_2 + pas_1 + 1
50
51  Tf = 2 * (math.sqrt(((q1) - (q0)) / (amax)))
52  vmax_acel = amax * (Tf / 2)
53  if (vmax_acel <= vmax):
54      ##### {{ Parametros Curva Amax triangular }} #####
55      vmax = vmax_acel
56      tau = Tf / 2
57      T = Tf
58      paso1 = (tau) / pas_1
59      paso2 = 0
60      pas_2 = 0
61      pas_total = pas_1 + pas_2 + pas_1 + 1
62
63  ##### {{ Creacion Matrices }} #####
64  pos = np.zeros((1, pas_total))
65  vel = np.zeros((1, pas_total))
66  acel = np.zeros((1, pas_total))
67  tiempo = np.zeros((1, pas_total))
68
69  ti = -1 * paso1
70  for i in range(0, pas_1 + 1):
71      ti = ti + paso1
72      x = ls_v_a_puntual(q0, q1, vmax, amax, ti)
73      pos[0, i] = x[0]
74      vel[0, i] = x[1]
75      acel[0, i] = x[2]
76      tiempo[0, i] = ti
77  for i in range(pas_1 + 1, pas_1 + pas_2 + 1):
78      ti = ti + paso2
79      x = ls_v_a_puntual(q0, q1, vmax, amax, ti)
80      pos[0, i] = x[0]
81      vel[0, i] = x[1]
82      acel[0, i] = x[2]
83      tiempo[0, i] = ti
84  for i in range(pas_1 + pas_2 + 1, pas_1 + pas_2 + pas_1):
85      ti = ti + paso1
86      x = ls_v_a_puntual(q0, q1, vmax, amax, ti)
87      pos[0, i] = x[0]
88      vel[0, i] = x[1]
89      acel[0, i] = x[2]
90      tiempo[0, i] = ti
91  for i in range(pas_1 + pas_2 + pas_1, pas_1 + pas_2 + pas_1 + 1):
92      ti = T
93      x = ls_v_a_puntual(q0, q1, vmax, amax, ti)
94      pos[0, i] = x[0]

```

```

95     vel[0, i] = x[1]
96     acel[0, i] = x[2]
97     tiempo[0, i] = ti
98
99 ##### {{ Plot }} #####
100 ax = plt.subplot(121)
101
102 plt.subplot(1, 3, 1)
103 plt.plot(tiempo[0, :], mtmm * pos[0, :], 'o-')
104 plt.title('Posicion vs time')
105 plt.xlabel('time [t]')
106 plt.ylabel(' [mm]')
107 plt.grid(True)
108
109 plt.subplot(1, 3, 2)
110 plt.plot(tiempo[0, :], mtmm * vel[0, :], '.-')
111 plt.title('vel vs time')
112 plt.xlabel('time [t]')
113 plt.ylabel(' [mm/s]')
114 plt.grid(True)
115
116 plt.subplot(1, 3, 3)
117 plt.plot(tiempo[0, :], mtmm * acel[0, :], '.-')
118 plt.title('acel vs time')
119 plt.xlabel('time [t]')
120 plt.ylabel(' [mm/s2]')
121 plt.grid(True)
122
123 plt.ion()
124 plt.show()
125 plt.pause(0.3)
126 plt.ioff()
127 plt.close("all") # 00x0
128 return [tiempo, mtmm * pos, mtmm * vel, mtmm * acel]
129
130
131 ##### [ Linear Speed (PUNTUAL) ] #####
132 ##### [ Linear Speed (PUNTUAL) ] #####
133 ##### [ Linear Speed (PUNTUAL) ] #####
134 # q0 y q1 en [metros]
135 # vmax [metros/segundos]
136 # amax [metros / segundos^2]
137 # tactual [segundos]
138 def ls_v_a_puntual(q0, q1, vmax, amax, tactual):
139     tau = vmax / amax
140     s = np.sign((q1 - q0))
141     T = (s * (((q1) - (q0)) / (vmax))) + (tau)
142     tramo1 = tau

```

```

143     tramo2 = T - tau
144     tramo3 = T
145     if ((0 <= tactual) and (tactual <= tramo1)):
146         q_actual = (q0) + ((s) * (amax / 2) * (tactual ** 2))
147         v_actual = s * amax * tactual
148         a_actual = s * amax
149         res = [q_actual, v_actual, a_actual, tramo1, tramo2, tramo3]
150         return res
151     elif ((tramo1 < tactual) and (tactual <= tramo2)):
152         q_actual = (q0) - ((s) * ((vmax ** 2) / (2 * amax))) + (s * vmax * tactual)
153         v_actual = s * vmax
154         a_actual = 0
155         res = [q_actual, v_actual, a_actual, tramo1, tramo2, tramo3]
156         return res
157     elif ((tramo2 < tactual) and (tactual <= tramo3)):
158         q_actual = (q1) + \
159                     ((s) * ((-1 * ((amax * (T ** 2)) / (2))) \
160                             + (amax * T * tactual) \
161                             + (-1 * (amax / 2) * (tactual ** 2))))
162         v_actual = s * ((amax * T) - (amax * tactual))
163         a_actual = s * (-1 * amax)
164         res = [q_actual, v_actual, a_actual, tramo1, tramo2, tramo3]
165         return res
166     else:
167         res = [0, 0, 0, tramo1, tramo2, tramo3]
168         return res
169
170 ##### FIN #####

```

B.8.4. trans_rot_ls_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo: Calcular angulos de rotacion, matrices de rotacion
5 # y distancia entre un punto inicial y final la trayectoria lineal
6 # de perfil de velocidad trapezoidal
7
8 ##### [Importar Librerias] #####
9 #####
10 #####
11 import math
12 import pd_tm1_adams
13 import numpy as np
14 from numpy.linalg import inv
15 import matplotlib.pyplot as plt
16 from mpl_toolkits.mplot3d import Axes3D
17
18 ##### [Parametros] #####
19 #####
20 #####
21 rtd = pd_tm1_adams.rtd()
22 mmtm = pd_tm1_adams.mmtm()
23 mtmm = pd_tm1_adams.mtmm()
24 dtr = pd_tm1_adams.dtr()
25
26
27 ##### [ Traslacion y Rotacion Directa ] #####
28 #####
29 #####
30
31 # ****
32 # ***** {{ Funcion Principal }} *****
33 # ****
34 def path_linear_speed(xo, yo, zo, xf, yf, zf):
35     ##### {{ Cambio unidades internacional }} #####
36     xoo = xo * mmtm
37     yoo = yo * mmtm
38     zoo = zo * mmtm
39     xff = xf * mmtm
40     yff = yf * mmtm
41     zff = zf * mmtm
42
43     res_final_1 = system_linear(xoo, yoo, zoo, xff, yff, zff)
44     return res_final_1
45
46
```

```

47 # ****
48 # **** {{ Rotaciones XYZ a X'Y'Z' }} ****
49 # ****
50 def system_linear(xo, yo, zo, xf, yf, zf): # ,theta_z,theta_y) :
51 ##### {{ Creacion de Matrices }} #####
52 rot_z = np.zeros((3, 3))
53 rot_y = np.zeros((3, 3))
54 rot_tras = np.zeros((4, 4))

55
56 pf = np.zeros((3, 1))
57 pf_trans = np.zeros((4, 1))

58
59 ##### {{ Traslacion }} #####
60 # Traslacion (Xo,Yo,Zo)
61 rot_tras[0, 0] = 1
62 rot_tras[1, 1] = 1
63 rot_tras[2, 2] = 1
64 rot_tras[3, 3] = 1
65 rot_tras[0, 3] = -xo
66 rot_tras[1, 3] = -yo
67 rot_tras[2, 3] = -zo
68 # Traslacion (Xf,Yf,Zf)
69 pf_trans[0, 0] = xf
70 pf_trans[1, 0] = yf
71 pf_trans[2, 0] = zf
72 pf_trans[3, 0] = 1

73
74 x_trans = (rot_tras).dot(pf_trans)

75
76 ##### {{ Angulos theta-y theta-z }} #####
77 NN = vector_unitario(0, 0, 0, x_trans[0, 0], x_trans[1, 0], x_trans[2, 0])
78 angles = angulos_rot(NN[0], NN[1], NN[2])
79 theta_y = angles[1]
80 theta_z = angles[0]

81
82 cos_axisz = math.cos(theta_y)
83 sin_axisz = math.sin(theta_y)

84
85 cos_axisy = math.cos(theta_z)
86 sin_axisy = math.sin(theta_z)

87
88 ##### {{ Rotacion Z }} #####
89 # Plano xy ya trasladado
90 pf[0, 0] = x_trans[0, 0]
91 pf[1, 0] = x_trans[1, 0]
92 pf[2, 0] = x_trans[2, 0]
93 # Rotacion respecto a eje Z
94 rot_z[0, 0] = cos_axisz

```

```

95     rot_z[0, 1] = sin_axisz
96     rot_z[1, 0] = -1 * sin_axisz
97     rot_z[1, 1] = cos_axisz
98     rot_z[2, 2] = 1
99
100    m_rot1 = rot_z.dot(pf)
101
102    ##### {{ Rotacion Y }} #####
103    # Rotacion respecto a eje Y
104    rot_y[0, 0] = cos_axisy
105    rot_y[0, 2] = sin_axisy
106    rot_y[1, 1] = 1
107    rot_y[2, 0] = -sin_axisy
108    rot_y[2, 2] = cos_axisy
109
110    m_rot2 = rot_y.dot(m_rot1)
111
112    return [(m_rot2[0, 0]) * mtmm, rot_z, rot_y, theta_y * rtd, theta_z * rtd, rot_tras]
113
114
115    # ****
116    # ***** {{ Vector Unitario }} *****
117    # ****
118    def vector_unitario(xo, yo, zo, xf, yf, zf):
119        delta_x = xf - xo
120        delta_y = yf - yo
121        delta_z = zf - zo
122        modulo = math.sqrt(((delta_x) ** 2) + ((delta_y) ** 2) + ((delta_z) ** 2))
123        nx = delta_x / modulo
124        ny = delta_y / modulo
125        nz = delta_z / modulo
126        return [nx, ny, nz]
127
128
129    # ****
130    # ***** {{ Theta Y ; Theta Z }} *****
131    # ****
132    def angulos_rot(nx, ny, nz):
133        if (nz < 0):
134            theta_z = (360 * dtr) + math.asin(nz)
135        else:
136            theta_z = math.asin(nz)
137
138        if (nx < 0):
139            theta_y = (180 * dtr) + (math.atan(ny / nx))
140        elif (nx == 0):
141            if (ny < 0):
142                theta_y = (270 * dtr)

```

```

143     elif (ny == 0):
144         theta_y = 0
145     else:
146         theta_y = (90 * dtr)
147     else:
148         if (ny < 0):
149             theta_y = (360 * dtr) + (math.atan(ny / nx))
150         else:
151             theta_y = (math.atan(ny / nx))
152     return [theta_z, theta_y]
153
154
155 ##### [ Traslacion y Rotacion Inversa ] #####
156 ##### [ Traslacion y Rotacion Inversa ] #####
157 ##### [ Traslacion y Rotacion Inversa ] #####
158
159 # ****
160 # ***** {{ Funcion Principal }} *****
161 # ****
162 def path_linear_speed_inv(m_tiempo,
163     m_pos, m_vel, m_acel,
164     rot_z, rot_y,
165     theta_y, theta_z,
166     m_trans):
167     ##### {{ Cambiar unidades a SI }} #####
168     m_tiempo = m_tiempo[0]
169     m_pos = mmtm * m_pos[0]
170     m_vel = mmtm * m_vel[0]
171     m_acel = mmtm * m_acel[0]
172     theta_y = theta_y * dtr
173     theta_z = theta_z * dtr
174
175     res_final_2 = system_linear_matrix(m_tiempo,
176         m_pos, m_vel, m_acel,
177         rot_z, rot_y,
178         theta_y, theta_z,
179         m_trans)
180     return res_final_2
181
182
183 # ****
184 # ***** {{ Trayectoria Lineal XYZ }} *****
185 # ****
186 def system_linear_matrix(m_tiempo,
187     m_pos, m_vel, m_acel,
188     rot_z, rot_y,
189     theta_y, theta_z,
190     m_trans):

```

```

191 tamano = len(m_tiempo)
192 ##### {{ Crear Matrices }} #####
193 m_pos_x = np.zeros((1, tamano))
194 m_pos_y = np.zeros((1, tamano))
195 m_pos_z = np.zeros((1, tamano))
196 m_vel_x = np.zeros((1, tamano))
197 m_vel_y = np.zeros((1, tamano))
198 m_vel_z = np.zeros((1, tamano))
199 m_acel_x = np.zeros((1, tamano))
200 m_acel_y = np.zeros((1, tamano))
201 m_acel_z = np.zeros((1, tamano))
202 for i in range(0, tamano):
203     ##### {{ Posicion xyz }} #####
204     xyz_ls = system_linear_inv(m_pos[i],
205         rot_z, rot_y,
206         theta_y, theta_z,
207         m_trans)
208     m_pos_x[0, i] = xyz_ls[0, 0]
209     m_pos_y[0, i] = xyz_ls[1, 0]
210     m_pos_z[0, i] = xyz_ls[2, 0]
211     ##### {{ Velocidad xyz }} #####
212     xyz_ls_vel = system_linear_inv(m_vel[i],
213         rot_z, rot_y,
214         theta_y, theta_z,
215         m_trans)
216     m_vel_x[0, i] = xyz_ls_vel[0, 0]
217     m_vel_y[0, i] = xyz_ls_vel[1, 0]
218     m_vel_z[0, i] = xyz_ls_vel[2, 0]
219     ##### {{ Aceleracion xyz }} #####
220     xyz_ls_acel = system_linear_inv(m_acel[i],
221         rot_z, rot_y,
222         theta_y, theta_z,
223         m_trans)
224     m_acel_x[0, i] = xyz_ls_acel[0, 0]
225     m_acel_y[0, i] = xyz_ls_acel[1, 0]
226     m_acel_z[0, i] = xyz_ls_acel[2, 0]
227 return [m_tiempo,
228         mtmm * m_pos_x[0], mtmm * m_pos_y[0], mtmm * m_pos_z[0],
229         mtmm * m_vel_x[0], mtmm * m_vel_y[0], mtmm * m_vel_z[0],
230         mtmm * m_acel_x[0], mtmm * m_acel_y[0], mtmm * m_acel_z[0]]
231
232
233 # ****
234 # ***** {{ Rotacion Inversa}} *****
235 # ****
236 def system_linear_inv(xprima, rot_z, rot_y, theta_y, theta_z, m_trans):
237     ##### {{ Creacion de Matrices }} #####
238     rot_tras_inv = np.zeros((4, 4))

```

```

239 pf_inv = np.zeros((3, 1))
240 pf_trans_inv = np.zeros((4, 1))
241 m_trans_inv = np.zeros((4, 4))
242
243 ##### {{ Rotacion Y }} #####
244 pf_inv[0, 0] = xprima
245 pf_inv[1, 0] = 0
246 pf_inv[2, 0] = 0
247
248 rot_y_tras = rot_y.transpose()
249
250 m_rot1_inv = rot_y_tras.dot(pf_inv)
251
252 ##### {{ Rotacion Z }} #####
253 rot_z_tras = rot_z.transpose()
254 m_rot2_inv = rot_z_tras.dot(m_rot1_inv)
255
256 ##### {{ Traslacion}} #####
257 pf_trans_inv[0, 0] = m_rot2_inv[0, 0]
258 pf_trans_inv[1, 0] = m_rot2_inv[1, 0]
259 pf_trans_inv[2, 0] = m_rot2_inv[2, 0]
260 pf_trans_inv[3, 0] = 1
261
262 m_trans_inv[0, 3] = -1 * m_trans[0, 3]
263 m_trans_inv[1, 3] = -1 * m_trans[1, 3]
264 m_trans_inv[2, 3] = -1 * m_trans[2, 3]
265 m_trans_inv[0, 0] = 1
266 m_trans_inv[1, 1] = 1
267 m_trans_inv[2, 2] = 1
268 m_trans_inv[3, 3] = 1
269
270 xyz_res = (m_trans_inv).dot(pf_trans_inv)
271 return xyz_res
272
273
274 # ****
275 # *** {{ Plot trayectoria linear XYZ}} ***
276 # ****
277 def plot_ls(x_m, y_m, z_m):
278     fig = plt.figure()
279     lsx = fig.add_subplot(111, projection='3d')
280     lsx.scatter3D(x_m, y_m, z_m, c=z_m, cmap='hsv')
281     lsx.set_xlabel('X [mm]')
282     lsx.set_ylabel('Y [mm]')
283     lsx.set_zlabel('Z [mm]')
284     lsx.set_title('Linear Speed')
285     lsx.text(x_m[0], y_m[0], z_m[0], "Inicio", color='red')
286     plt.ion()

```

```
287     plt.show()
288     plt.pause(0.1)
289     plt.ioff()
290
291 ##### FIN #####
```

B.8.5. delta_kinematics_t1m_adams.py

```
1 #Nombre Creador: Ivan Alejandro Fernandez Gracia
2 #Universidad: Universidad de Santiago de Chile
3 #Mail: ivan.fernandez.g@usach.cl
4 #Objetivo: Cinematica directa e inversa robot delta
5
6 ##### [Importar Librerias] #####
7 ##### [Parametros] #####
8
9 import math
10 import numpy as np
11 from numpy.linalg import inv
12 import matplotlib.pyplot as plt
13 import pd_tm1_adams
14
15 ##### [Constantes Trigonometricas]
16 ##### [Transformacion unidades]
17
18 e=pd_tm1_adams.e()
19 f=pd_tm1_adams.f()
20 re=pd_tm1_adams.l2()
21 rf=pd_tm1_adams.l1()
22
23 # Constantes Trigonometricas
24 sqrt3=pd_tm1_adams.sqrt3()
25 pi=pd_tm1_adams.pi()
26 sin120=sqrt3/2.0
27 cos120=pd_tm1_adams.cos120()
28 tan60=sqrt3
29 sin30=pd_tm1_adams.sin30()
30 tan30=pd_tm1_adams.tan30()
31
32 # Transformacion unidades
33 dtr=pd_tm1_adams.dtr()
34 mtmm=pd_tm1_adams.mtmm()
35 mmmtm=pd_tm1_adams.mmmtm()
36 rtd= pd_tm1_adams.rtd()
37
38 ##### [Cinematica Directa] #####
39
40
41
42 # ****
43 # ***** {{ Directa }} ****
44 # ****
45 def forward(theta1,theta2,theta3):
46     theta1*=dtr
```

```

47     theta2*=dtr
48     theta3*=dtr
49
50     x0=0.0
51     y0=0.0
52     z0=0.0
53
54     t=(f-e)*tan30/2.0
55
56     y1=-(t+rf*math.cos(theta1))
57     z1=-rf*math.sin(theta1)
58
59     y2=(t+rf*math.cos(theta2))*sin30
60     x2=y2*tan60
61     z2=-rf*math.sin(theta2)
62
63     y3=(t+rf*math.cos(theta3))*sin30
64     x3=-y3*tan60
65     z3=-rf*math.sin(theta3)
66
67     dnm=(y2-y1)*x3-(y3-y1)*x2
68
69     w1=y1*y1+z1*z1
70     w2=x2*x2+y2*y2+z2*z2
71     w3=x3*x3+y3*y3+z3*z3
72
73     # x = ( a1 * z + b1 ) / dnm
74     a1=(z2-z1)*(y3-y1)-(z3-z1)*(y2-y1)
75     b1=-((w2-w1)*(y3-y1)-(w3-w1)*(y2-y1))/2.0
76
77     # y = ( a2 * z + b2 ) / dnm
78     a2=-(z2-z1)*x3+(z3-z1)*x2
79     b2=((w2-w1)*x3-(w3-w1)*x2)/2.0
80
81     # a * z ~2 + b * z + c = 0
82     a=a1*a1+a2*a2+dnm*dnm
83     b=2.0*(a1*b1+a2*(b2-y1*dnm)-z1*dnm*dnm)
84     c=(b2-y1*dnm)*(b2-y1*dnm)+b1*b1+dnm*dnm*(z1*z1-re*re)
85
86     # discriminante
87     d=b*b-4.0*a*c
88     if d<0.0:
89         print("discriminante - directo error")
90         return [1,0,0,0]
91
92     z0=-0.5*(b+math.sqrt(d))/a
93     x0=(a1*z0+b1)/dnm
94     y0=(a2*z0+b2)/dnm

```

```

95
96     y0*=mtmm
97     x0*=mtmm
98     z0*=mtmm
99     return [0,x0,y0,z0]
100
101 ##### [ Cinematica Inversa ] #####
102 ###### [ Cinematica Inversa ] #####
103 ###### [ Cinematica Inversa ] #####
104
105 # ****
106 # ***** {{ Inversa }} ****
107 # ****
108 def inverse(x0,y0,z0):
109     y0*=mmtm
110     x0*=mmtm
111     z0*=mmtm
112
113     theta1=0
114     theta2=0
115     theta3=0
116
117     #Rotacion(0, 120, 240)
118     status=angle_yz(x0,y0,z0)
119
120     if status[0]== 0:
121         theta2=status[1]
122         status=angle_yz(x0*cos120+y0*sin120,y0*cos120-x0*sin120,z0,theta2)
123
124     if status[0]== 0:
125         theta3=status[1]
126         status=angle_yz(x0*cos120-y0*sin120,y0*cos120+x0*sin120,z0,theta3)
127
128     theta1=status[1]
129
130     return [status[0],theta1,theta2,theta3]
131
132 # ****
133 # ***** {{ Plano YZ }} ****
134 # ****
135 def angle_yz(x0,y0,z0,theta=None):
136     y1=-0.5*tan30*f
137     y0=y0-0.5*tan30*e
138     # z = a + b*y
139     a=(x0*x0+y0*y0+z0*z0+rf*rf-re*re-y1*y1)/(2.0*z0)
140     b=(y1-y0)/z0
141
142     # discriminante

```

```

143     #d=-(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf)
144     d=(-1)*(a+b*y1)*(a+b*y1)+(rf*rf*(b*b+1.0))
145
146     if d<0:
147         print("discriminante - angle_yz")
148         return [1,0] # error inversa
149
150     yj=((y1-a*b)-math.sqrt(d))/(b*b+1.0)
151     zj=a+b*yj
152
153     if ((y1-yj)!=0.0):
154         theta=math.atan(((1)*zj)/(abs(y1-yj)))
155         if (yj<y1):
156             theta=theta*(rtd)
157         else:
158             if zj<0:
159                 theta=theta*(180.0/pi)
160                 theta=180.0-theta
161             else:
162                 theta=theta*(180.0/pi)
163                 theta=(-1.0*180.0)+theta
164         else:
165             if zj<0:
166                 theta=90.0
167             else:
168                 theta=-90.0
169
170     return [0,theta]
171
172 # *****
173 # ***** {{ Inversa Matricial }} *****
174 # *****
175 def inverse_m(p0x_m,p0y_m,p0z_m):
176     tamano=len(p0x_m)
177     theta1_m=np.zeros((1,tamano))
178     theta2_m=np.zeros((1,tamano))
179     theta3_m=np.zeros((1,tamano))
180     for i in range(0,tamano):
181         theta_m=inverse(p0x_m[i],p0y_m[i],p0z_m[i])
182         #print(theta_m)
183         theta1_m[0,i]=theta_m[1]
184         theta2_m[0,i]=theta_m[2]
185         theta3_m[0,i]=theta_m[3]
186     return [theta1_m[0],theta2_m[0],theta3_m[0]]

```

B.8.6. delta_kinematics_Paderborn_tm1_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo: Cinematica Directa y Inversa robot delta
5
6 ##### [Importar Librerias] #####
7 ##### [Parametros] #####
8
9 import math
10 import numpy as np
11 from numpy.linalg import inv
12 import matplotlib.pyplot as plt
13 import pd_tm1_adams
14
15 ##### [Constantes Trigonometricas]
16 ##### [Cinematica Directa] #####
17
18 e = pd_tm1_adams.e()
19 f = pd_tm1_adams.f()
20 re = pd_tm1_adams.l2()
21 rf = pd_tm1_adams.l1()
22
23 # Constantes Trigonometricas
24 sqrt3 = pd_tm1_adams.sqrt3()
25 pi = pd_tm1_adams.pi()
26 sin120 = sqrt3 / 2.0
27 cos120 = pd_tm1_adams.cos120()
28 tan60 = sqrt3
29 sin30 = pd_tm1_adams.sin30()
30 tan30 = pd_tm1_adams.tan30()
31 cos30 = pd_tm1_adams.cos30()
32
33 dtr = pd_tm1_adams.dtr()
34 mtmm = pd_tm1_adams.mtmm()
35 mmtm = pd_tm1_adams.mmtm()
36
37
38 ##### [Cinematica Directa] #####
39 ##### [Inversa] #####
40
41
42 # ****
43 # ***** {{ Directa }} ****
44 # ****
45 def forward_Paderborn(theta1, theta2, theta3):
46     theta2 *= dtr
```

```

47 theta1 *= dtr
48 theta3 *= dtr
49
50 x0 = 0.0
51 y0 = 0.0
52 z0 = 0.0
53
54 t = (f - e) * tan30 / 2.0
55
56 # j2prima_x=0
57 j2prima_y = -(t) - (rf * math.cos(theta2))
58 j2prima_z = -rf * (math.sin(theta2))
59
60 j3prima_y = (t + (rf * math.cos(theta3))) * sin30
61 j3prima_x = (t + (rf * math.cos(theta3))) * cos30
62 j3prima_z = -rf * (math.sin(theta3))
63
64 j1prima_y = (t + (rf * math.cos(theta1))) * sin30
65 j1prima_x = (-1) * (t + (rf * math.cos(theta1))) * cos30
66 j1prima_z = -rf * (math.sin(theta1))
67
68 # x1=j2prima_x
69 y1 = j2prima_y
70 z1 = j2prima_z
71
72 y2 = j3prima_y
73 x2 = j3prima_x
74 z2 = j3prima_z
75
76 y3 = j1prima_y
77 x3 = j1prima_x
78 z3 = j1prima_z
79
80 dnm = (y2 - y1) * x3 - (y3 - y1) * x2
81
82 w1 = y1 * z1 + z1 * z1
83 w2 = x2 * x2 + y2 * y2 + z2 * z2
84 w3 = x3 * x3 + y3 * y3 + z3 * z3
85
86 # x = ( a1 * z + b1 ) / dnm
87 a1 = (z2 - z1) * (y3 - y1) - (z3 - z1) * (y2 - y1)
88 b1 = -((w2 - w1) * (y3 - y1) - (w3 - w1) * (y2 - y1)) / 2.0
89
90 # y = ( a2 * z + b2 ) / dnm
91 a2 = -(z2 - z1) * x3 + (z3 - z1) * x2
92 b2 = ((w2 - w1) * x3 - (w3 - w1) * x2) / 2.0
93
94 # a * z ^2 + b * z + c = 0

```

```

95     a = a1 * a1 + a2 * a2 + dnm * dnm
96     b = 2.0 * (a1 * b1 + a2 * (b2 - y1 * dnm) - z1 * dnm * dnm)
97     c = (b2 - y1 * dnm) * (b2 - y1 * dnm) + b1 * b1 + dnm * dnm * (z1 * z1 - re * re)
98
99     # discriminante
100    d = b * b - 4.0 * a * c
101    if d < 0.0:
102        print("discriminante - directa Paderborn")
103        return [1, 0, 0, 0]  # error
104
105    z0 = -0.5 * (b + math.sqrt(d)) / a
106    x0 = (a1 * z0 + b1) / dnm
107    y0 = (a2 * z0 + b2) / dnm
108
109    y0 *= mtmm
110    x0 *= mtmm
111    z0 *= mtmm
112    return [0, x0, y0, z0]
113
114
115 ##### [ Cinematica Inversa ] #####
116 ##### [ Cinematica Inversa ] #####
117 ##### [ Cinematica Inversa ] #####
118 # ****
119 # ***** {{ Inversa }} ****
120 # ****
121 def inverse_Paderborn(p0x, p0y, p0z):
122     p0x *= mmtm
123     p0y *= mmtm
124     p0z *= mmtm
125
126     theta1 = 0
127     theta2 = 0
128     theta3 = 0
129     status = angle_yz_Paderborn(p0x, p0y, p0z)
130
131     if status[0] == 0:
132         theta2 = status[1]
133         status = angle_yz_Paderborn(p0x * cos120 + p0y * sin120,
134             p0y * cos120 - p0x * sin120,
135             p0z,
136             theta3)
137
138     if status[0] == 0:
139         theta3 = status[1]
140         status = angle_yz_Paderborn(p0x * cos120 - p0y * sin120,
141             p0y * cos120 + p0x * sin120,
142             p0z,

```

```

143     theta1)
144
145     theta1 = status[1]
146
147     return [status[0], theta1, theta2, theta3]
148
149
150 # *****
151 # ***** {{ Plano YZ }} *****
152 # *****
153 def angle_yz_Paderborn(p0x, p0y, p0z, theta=None):
154     f2y = -1 * (f / (2 * math.sqrt(3)))
155
156     # Centros y Radios de los 2 circulos
157     h1 = (p0y) - ((e) / (2 * math.sqrt(3)))
158     k1 = p0z
159     h2 = f2y
160     k2 = 0
161     R1 = math.sqrt((re) ** 2 - ((p0x) ** 2))
162     R2 = (rf)
163     dist_centros = math.sqrt(((h1 - h2) ** 2) + ((k1 - k2) ** 2))
164
165     # Casos de interseccion
166     if dist_centros > R1 + R2:
167         print("non intersecting KI")
168         return [1, 0]
169     elif dist_centros < abs(R1 - R2):
170         print("One circle within other KI")
171         return [1, 0]
172     elif dist_centros == 0 and R1 == R2:
173         print("coincident circles KI")
174         return [1, 0]
175     else:
176         a = (R1 ** 2 - R2 ** 2 + dist_centros ** 2) / (2 * dist_centros)
177         h = math.sqrt(R1 ** 2 - a ** 2)
178
179         x2 = h1 + a * (h2 - h1) / dist_centros
180         y2 = k1 + a * (k2 - k1) / dist_centros
181
182         x3 = x2 + h * (k2 - k1) / dist_centros
183         y3 = y2 - h * (h2 - h1) / dist_centros
184
185         x4 = x2 - h * (k2 - k1) / dist_centros
186         y4 = y2 + h * (h2 - h1) / dist_centros
187         # Intersection (x3, y3) (x4, y4)
188
189         if x3 < x4:
190             j2y = x3

```

```

191     j2z = y3
192 else:
193     j2y = x4
194     j2z = y4
195
196 if (f2y - j2y) != 0:
197     theta = math.atan(((-1) * (j2z)) / (abs(f2y - j2y)))
198     if (j2y < f2y):
199         theta = theta * (180.0 / pi)
200     else:
201         if j2z < 0:
202             theta = theta * (180.0 / pi)
203             theta = 180 - theta
204         else:
205             theta = theta * (180.0 / pi)
206             theta = (-1 * 180) + theta
207     else:
208         if j2z < 0:
209             theta = 90
210         else:
211             theta = -90
212 return [0, theta] # error
213
214
215 # ****
216 # ***** {{ Inversa Matricial }} *****
217 # ****
218 def inverse_Paderborn_m(p0x_m, p0y_m, p0z_m):
219     tamano = len(p0x_m)
220     theta1_m = np.zeros((1, tamano))
221     theta2_m = np.zeros((1, tamano))
222     theta3_m = np.zeros((1, tamano))
223     for i in range(0, tamano):
224         theta_m = inverse_Paderborn(p0x_m[i], p0y_m[i], p0z_m[i])
225         theta1_m[0, i] = theta_m[1]
226         theta2_m[0, i] = theta_m[2]
227         theta3_m[0, i] = theta_m[3]
228     return [theta1_m[0], theta2_m[0], theta3_m[0]]

```

B.8.7. jacobian_tm1_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo: Determinar Jacobiano Metodo A
5
6 ##### [Importar Librerias] #####
7 #####
8 #####
9 import math
10 import pd_tm1_adams
11 import numpy as np
12 from numpy.linalg import inv
13 import time
14
15 ##### [Parametros] #####
16 #####
17 #####
18 e = pd_tm1_adams.e()
19 f = pd_tm1_adams.f()
20 re = pd_tm1_adams.l2()
21 rf = pd_tm1_adams.l1()
22
23 # Constantes Trigonometricas
24 sqrt3 = pd_tm1_adams.sqrt3()
25 pi = pd_tm1_adams.pi()
26 sin120 = sqrt3 / 2.0
27 cos120 = pd_tm1_adams.cos120()
28 tan60 = sqrt3
29 sin30 = pd_tm1_adams.sin30()
30 tan30 = pd_tm1_adams.tan30()
31
32 dtr = pd_tm1_adams.dtr()
33 mtmm = pd_tm1_adams.mtmm()
34 mmtm = pd_tm1_adams.mmtm()
35 rtd = pd_tm1_adams.rtd()
36
37
38 ##### [Jacobian Total] #####
39 #####
40 #####
41 def jacobian_total(px, py, pz,
42                     theta11, theta12, theta13):
43     # Cambio unidades Internacional
44     pxx = px * mmtm
45     pyy = py * mmtm
46     pzz = pz * mmtm
```

```

47
48     # Ordenar segun sistema LOCAL
49     punto = [-pyy, -pxx, -pz]
50     phi_orden = [0, 120, 240]
51     thetai_orden = [theta12, theta11, theta13]
52
53     jxx = np.zeros((3, 3))
54     jinv = np.zeros((3, 3))
55
56     # Angulo theta3i
57     theta31 = calculo_theta3i(phi_orden[0], punto[0], punto[1])
58     theta32 = calculo_theta3i(phi_orden[1], punto[0], punto[1])
59     theta33 = calculo_theta3i(phi_orden[2], punto[0], punto[1])
50
51     # Angulo theta2i
52     theta21 = calculo_theta2i(phi_orden[0], punto[0], punto[1], punto[2], theta31)
53     theta22 = calculo_theta2i(phi_orden[1], punto[0], punto[1], punto[2], theta32)
54     theta23 = calculo_theta2i(phi_orden[2], punto[0], punto[1], punto[2], theta33)
55
56     # Jacobiano directo (jxx)
57     jxx[0, 0] = calculo_jix(phi_orden[0], thetai_orden[0], theta21, theta31)
58     jxx[1, 0] = calculo_jix(phi_orden[1], thetai_orden[1], theta22, theta32)
59     jxx[2, 0] = calculo_jix(phi_orden[2], thetai_orden[2], theta23, theta33)
60
61     jxx[0, 1] = calculo_jiy(phi_orden[0], thetai_orden[0], theta21, theta31)
62     jxx[1, 1] = calculo_jiy(phi_orden[1], thetai_orden[1], theta22, theta32)
63     jxx[2, 1] = calculo_jiy(phi_orden[2], thetai_orden[2], theta23, theta33)
64
65     jxx[0, 2] = calculo_jiz(phi_orden[0], thetai_orden[0], theta21, theta31)
66     jxx[1, 2] = calculo_jiz(phi_orden[1], thetai_orden[1], theta22, theta32)
67     jxx[2, 2] = calculo_jiz(phi_orden[2], thetai_orden[2], theta23, theta33)
68
69     # Jacobiano inverso (jinv)
70     jinv[0, 0] = calculo_jtheta1(phi_orden[0], theta21, theta31)
71     jinv[1, 1] = calculo_jtheta2(phi_orden[1], theta22, theta32)
72     jinv[2, 2] = calculo_jtheta3(phi_orden[2], theta23, theta33)
73
74     Jtotal = np.dot(inv(jxx), ((1.0) * jinv))
75
76     return [jinv, jxx, theta31, theta32, theta33, theta21, theta22, theta23, Jtotal]
77
78
79 ##### [ Angulos theta2i ,theta3i ] #####
80 ##### [ Angulos theta2i ,theta3i ] #####
81 ##### [ Angulos theta2i ,theta3i ] #####
82
83     # /-----/
84     # /----- theta3i -----/

```

```

95 # /----- /
96 def calculo_theta3i(phi_deg,
97                      px, py):
98     phi = phi_deg * dtr
99
100    cyi = ((-px) * (math.sin(phi))) + ((py) * (math.cos(phi)))
101    b = re
102    theta3i_rad = math.acos(cyi / b)
103
104    theta3i = theta3i_rad / dtr
105    return theta3i
106
107
108 # /----- /
109 # /----- theta2i ----- /
110 # /----- /
111 def calculo_theta2i(phi_deg,
112                      px, py, pz,
113                      theta3i_deg):
114     phi = phi_deg * dtr
115
116     r_efector = (e / 2.0) * (tan30)
117     R_fija = (f / 2.0) * (tan30)
118
119     cxi = ((px) * (math.cos(phi))) + ((py) * (math.sin(phi))) + (r_efector) - (R_fija)
120     cyi = ((-px) * (math.sin(phi))) + ((py) * (math.cos(phi)))
121     czi = pz
122
123     aa = rf
124     bb = re
125
126     theta3i = theta3i_deg * dtr
127
128     A = ((cxi ** 2.0) + (cyi ** 2.0) + (czi ** 2.0)) - ((aa ** 2.0) + (bb ** 2.0))
129     B = 2.0 * aa * bb * (math.sin(theta3i))
130
131     theta2i_rad = math.acos(A / B)
132     theta2i = theta2i_rad / dtr
133
134     return theta2i
135
136
137 ##### [ Jacobian cinematica directa Ji(xyz) ] #####
138 ##### [ Jacobian cinematica directa Ji(xyz) ] #####
139 ##### [ Jacobian cinematica directa Ji(xyz) ] #####
140
141 # /----- /
142 # /----- Jix ----- /

```

```

143 # /-----/
144 def calculo_jix(phi_deg,
145         theta1i_deg, theta2i_deg, theta3i_deg):
146     phi = phi_deg * dtr
147     theta1i = theta1i_deg * dtr
148     theta2i = theta2i_deg * dtr
149     theta3i = theta3i_deg * dtr
150
151     A = (math.cos(theta1i + theta2i)) * (math.sin(theta3i)) * (math.cos(phi))
152     B = (math.cos(theta3i)) * (math.sin(phi))
153     jix = (A) - (B)
154
155     return jix
156
157 # /-----/
158 # /----- Jiy -----/
159 # /-----/
160 def calculo_jiy(phi_deg,
161         theta1i_deg, theta2i_deg, theta3i_deg):
162     phi = phi_deg * dtr
163     theta1i = theta1i_deg * dtr
164     theta2i = theta2i_deg * dtr
165     theta3i = theta3i_deg * dtr
166
167     A = (math.cos(theta1i + theta2i)) * (math.sin(theta3i)) * (math.sin(phi))
168     B = (math.cos(theta3i)) * (math.cos(phi))
169     jiy = (A) + (B)
170
171     return jiy
172
173 # /-----/
174 # /----- Jiz -----/
175 # /-----/
176 def calculo_jiz(phi_deg,
177         theta1i_deg, theta2i_deg, theta3i_deg):
178     phi = phi_deg * dtr
179     theta1i = theta1i_deg * dtr
180     theta2i = theta2i_deg * dtr
181     theta3i = theta3i_deg * dtr
182
183     jiz = (math.sin(theta3i)) * (math.sin(theta1i + theta2i))
184
185     return jiz
186
187 ##### [ Jacobian cinematica inversa angular Jtheta] #####
188 ##### #####
189 ##### #####
190

```

```

191 # /-----/
192 # /----- jtheta1 -----/
193 # /-----/
194 def calculo_jtheta1(phi_deg,
195                         theta2i_deg, theta3i_deg):
196     phi = phi_deg * dtr # Motor 1
197     theta2i = theta2i_deg * dtr
198     theta3i = theta3i_deg * dtr
199     aa = rf
200
201     jtheta1 = (aa) * (math.sin(theta2i)) * (math.sin(theta3i))
202     return jtheta1
203
204
205 # /-----/
206 # /----- jtheta2 -----/
207 # /-----/
208 def calculo_jtheta2(phi_deg,
209                         theta2i_deg, theta3i_deg):
210     phi = phi_deg * dtr # Motor 2
211     theta2i = theta2i_deg * dtr
212     theta3i = theta3i_deg * dtr
213     aa = rf
214     jtheta2 = (aa) * (math.sin(theta2i)) * (math.sin(theta3i))
215     return jtheta2
216
217
218 # /-----/
219 # /----- jtheta3 -----/
220 # /-----/
221 def calculo_jtheta3(phi_deg,
222                         theta2i_deg, theta3i_deg):
223     phi = phi_deg * dtr # Motor 3
224     theta2i = theta2i_deg * dtr
225     theta3i = theta3i_deg * dtr
226     aa = rf
227
228     jtheta3 = (aa) * (math.sin(theta2i)) * (math.sin(theta3i))
229     return jtheta3
230
231
232 ##### [ Jacobian Derivadas Total ] #####
233 ##### [ Jacobian Derivadas Total ] #####
234 ##### [ Jacobian Derivadas Total ] #####
235
236 # /-----/
237 # /----- jacobian_total_der -----/
238 # /-----/

```

```

239 def jacobian_total_der(px, py, pz,
240                         theta11, theta12, theta13,
241                         px_p, py_p, pz_p):
242     # Cambio unidades Internacional
243     pxx = px * mmtm
244     pyy = py * mmtm
245     pzz = pz * mmtm
246
247     pxx_p = px_p * mmtm
248     pyy_p = py_p * mmtm
249     pzz_p = pz_p * mmtm
250
251     # Ordenar segun sistema LOCAL
252     punto = [-pyy, -pxx, -pzz]
253     punto_p = [-pyy_p, -pxx_p, -pzz_p]
254     phi_orden = [0, 120, 240]
255     thetai_orden = [theta12, theta11, theta13]
256
257     jxx = np.zeros((3, 3))
258     jinv = np.zeros((3, 3))
259     jxx_der = np.zeros((3, 3))
260     jinv_der = np.zeros((3, 3))
261
262     # Angulo theta3i
263     theta31 = calculo_theta3i(phi_orden[0], punto[0], punto[1])
264     theta32 = calculo_theta3i(phi_orden[1], punto[0], punto[1])
265     theta33 = calculo_theta3i(phi_orden[2], punto[0], punto[1])
266
267     # Angulo theta2i
268     theta21 = calculo_theta2i(phi_orden[0], punto[0], punto[1], punto[2], theta31)
269     theta22 = calculo_theta2i(phi_orden[1], punto[0], punto[1], punto[2], theta32)
270     theta23 = calculo_theta2i(phi_orden[2], punto[0], punto[1], punto[2], theta33)
271
272     # Jacobiano directo (jxx)
273     jxx[0, 0] = calculo_jix(phi_orden[0], thetai_orden[0], theta21, theta31)
274     jxx[1, 0] = calculo_jix(phi_orden[1], thetai_orden[1], theta22, theta32)
275     jxx[2, 0] = calculo_jix(phi_orden[2], thetai_orden[2], theta23, theta33)
276
277     jxx[0, 1] = calculo_jiy(phi_orden[0], thetai_orden[0], theta21, theta31)
278     jxx[1, 1] = calculo_jiy(phi_orden[1], thetai_orden[1], theta22, theta32)
279     jxx[2, 1] = calculo_jiy(phi_orden[2], thetai_orden[2], theta23, theta33)
280
281     jxx[0, 2] = calculo_jiz(phi_orden[0], thetai_orden[0], theta21, theta31)
282     jxx[1, 2] = calculo_jiz(phi_orden[1], thetai_orden[1], theta22, theta32)
283     jxx[2, 2] = calculo_jiz(phi_orden[2], thetai_orden[2], theta23, theta33)
284
285     # Jacobiano inverso (jinv)
286     jinv[0, 0] = calculo_jtheta1(phi_orden[0], theta21, theta31)

```

```

287     jinv[1, 1] = calculo_jtheta2(phi_orden[1], theta22, theta32)
288     jinv[2, 2] = calculo_jtheta3(phi_orden[2], theta23, theta33)
289
290     Jtotal_inv = np.dot(inv(jinv), ((1.0) * jxx))
291
292     theta1i_p = Jtotal_inv.dot([[punto_p[0]], [punto_p[1]], [punto_p[2]]])
293     theta1i_p = theta1i_p * (rtd)
294
295     # Velocidad theta3i
296     theta31_p = calculo_theta3i_der(phi_orden[0],
297                                     punto[0], punto[1], punto_p[0], punto_p[1])
298     theta32_p = calculo_theta3i_der(phi_orden[1],
299                                     punto[0], punto[1], punto_p[0], punto_p[1])
300     theta33_p = calculo_theta3i_der(phi_orden[2],
301                                     punto[0], punto[1], punto_p[0], punto_p[1])
302
303     # Velocidad theta2i
304     theta21_p = calculo_theta2i_der(phi_orden[0],
305                                     punto[0], punto[1], punto[2],
306                                     theta31, theta31_p, punto_p[0],
307                                     punto_p[1], punto_p[2])
308     theta22_p = calculo_theta2i_der(phi_orden[1],
309                                     punto[0], punto[1], punto[2],
310                                     theta32, theta32_p, punto_p[0],
311                                     punto_p[1], punto_p[2])
312     theta23_p = calculo_theta2i_der(phi_orden[2],
313                                     punto[0], punto[1], punto[2],
314                                     theta33, theta33_p, punto_p[0],
315                                     punto_p[1], punto_p[2])
316
317     # Jacobiano directo derivada (jix)
318     jxx_der[0, 0] = calculo_jix_der(phi_orden[0],
319                                     thetai_orden[0], theta21, theta31,
320                                     theta1i_p[0, 0], theta21_p, theta31_p)
321     jxx_der[1, 0] = calculo_jix_der(phi_orden[1],
322                                     thetai_orden[1], theta22, theta32,
323                                     theta1i_p[1, 0], theta22_p, theta32_p)
324     jxx_der[2, 0] = calculo_jix_der(phi_orden[2],
325                                     thetai_orden[2], theta23, theta33,
326                                     theta1i_p[2, 0], theta23_p, theta33_p)
327
328     jxx_der[0, 1] = calculo_jiy_der(phi_orden[0],
329                                     thetai_orden[0], theta21, theta31,
330                                     theta1i_p[0, 0], theta21_p, theta31_p)
331     jxx_der[1, 1] = calculo_jiy_der(phi_orden[1],
332                                     thetai_orden[1], theta22, theta32,
333                                     theta1i_p[1, 0], theta22_p, theta32_p)
334     jxx_der[2, 1] = calculo_jiy_der(phi_orden[2],

```

```

335                                     theta1i_orden[2], theta23, theta33,
336                                     theta1i_p[2, 0], theta23_p, theta33_p)
337
338     jxx_der[0, 2] = calculo_jiz_der(phi_orden[0],
339                                         theta1i_orden[0], theta21, theta31,
340                                         theta1i_p[0, 0], theta21_p, theta31_p)
341     jxx_der[1, 2] = calculo_jiz_der(phi_orden[1],
342                                         theta1i_orden[1], theta22, theta32,
343                                         theta1i_p[1, 0], theta22_p, theta32_p)
344     jxx_der[2, 2] = calculo_jiz_der(phi_orden[2],
345                                         theta1i_orden[2], theta23, theta33,
346                                         theta1i_p[2, 0], theta23_p, theta33_p)
347
348     # Jacobiano inverso derivada (jinv)
349     jinv_der[0, 0] = calculo_jtheta1_der(phi_orden[0],
350                                         theta21, theta31,
351                                         theta21_p, theta31_p)
352     jinv_der[1, 1] = calculo_jtheta2_der(phi_orden[1],
353                                         theta22, theta32,
354                                         theta22_p, theta32_p)
355     jinv_der[2, 2] = calculo_jtheta3_der(phi_orden[2],
356                                         theta23, theta33,
357                                         theta23_p, theta33_p)
358
359     return [jinv, jxx, jinv_der, jxx_der,
360             theta31, theta32, theta33,
361             theta21, theta22, theta23]
362
363 #####
364 ##### [ Derivada theta2i ,theta3i ] #####
365 #####
366
367 # /-----/
368 # /----- Derivada theta3i -----/
369 # /----- -/
370 def calculo_theta3i_der(phi_deg,
371                         px, py,
372                         px_p, py_p):
373     phi = phi_deg * dtr
374
375     cyi = ((-px) * (math.sin(phi))) + ((py) * (math.cos(phi)))
376     b = re
377
378     cyi_p = ((-px_p) * (math.sin(phi))) + ((py_p) * (math.cos(phi)))
379
380     A = (cyi_p) / (b)
381     B = math.sqrt((1.0) - (((cyi) / (b)) ** 2.0))
382

```

```

383     theta3i_p_rad = (-1.0) * ((A) / (B))
384
385     theta3i_p = theta3i_p_rad / dtr
386     return theta3i_p
387
388
389 # /----- /
390 # /----- Derivada theta2i ----- /
391 # /----- - /
392 def calculo_theta2i_der(phi_deg,
393                         px, py, pz,
394                         theta3i_deg, theta3i_p_deg,
395                         px_p, py_p, pz_p):
396     phi = phi_deg * dtr
397
398     r_efector = (e / 2.0) * (tan30)
399     R_fija = (f / 2.0) * (tan30)
400
401     cxi = ((px) * (math.cos(phi))) + ((py) * (math.sin(phi))) + (r_efector) - (R_fija)
402     cyi = ((-px) * (math.sin(phi))) + ((py) * (math.cos(phi)))
403     czi = pz
404
405     cxi_p = ((px_p) * (math.cos(phi))) + ((py_p) * (math.sin(phi)))
406     cyi_p = ((-px_p) * (math.sin(phi))) + ((py_p) * (math.cos(phi)))
407     czi_p = pz_p
408
409     aa = rf
410     bb = re
411
412     theta3i = theta3i_deg * dtr
413     theta3i_p = theta3i_p_deg * dtr
414
415     A11 = ((2.0 * ((cxi * cxi_p) + (cyi * cyi_p) + (czi * czi_p))) * (math.sin(theta3i)))
416     A12 = (((cxi ** 2.0) + (cyi ** 2.0) + (czi ** 2.0)) -
417             ((aa ** 2.0) + (bb ** 2.0))) * (math.cos(theta3i)) * (theta3i_p)
418     B1 = (math.sin(theta3i)) ** 2.0
419
420     kprima = (1.0 / (2.0 * aa * bb)) * (((A11) - (A12)) / (B1))
421
422     A2 = ((cxi ** 2.0) + (cyi ** 2.0) + (czi ** 2.0)) - ((aa ** 2.0) + (bb ** 2.0))
423     B2 = 2.0 * aa * bb * (math.sin(theta3i))
424     k = A2 / B2
425
426     A = kprima
427     B = math.sqrt((1.0) - ((k) ** 2.0))
428
429     theta2i_p_rad = (-1.0) * ((A) / (B))
430

```

```

431     theta2i_p = theta2i_p_rad / dtr
432
433     return theta2i_p
434
435
436 ##### [ Derivada  $J_i(xyz)$ ] #####
437 ##### [ Derivada  $J_i(xyz)$ ] #####
438 ##### [ Derivada  $J_i(xyz)$ ] #####
439
440 # /-----/
441 # /----- Derivada  $J_{ix}$  -----/
442 # /-----/
443 def calculo_jix_der(phi_deg,
444                     theta1i_deg, theta2i_deg, theta3i_deg,
445                     theta1i_deg_p, theta2i_deg_p, theta3i_deg_p):
446     phi = phi_deg * dtr
447     theta1i = theta1i_deg * dtr
448     theta2i = theta2i_deg * dtr
449     theta3i = theta3i_deg * dtr
450
451     theta1i_p = theta1i_deg_p * dtr
452     theta2i_p = theta2i_deg_p * dtr
453     theta3i_p = theta3i_deg_p * dtr
454
455     A1 = (-1.0) * (math.sin(theta1i + theta2i)) * \
456             (math.sin(theta3i)) * (theta1i_p + theta2i_p)
457     A2 = (math.cos(theta1i + theta2i)) * (math.cos(theta3i)) * (theta3i_p)
458     A = (math.cos(phi)) * ((A1) + (A2))
459     B = (math.sin(phi)) * ((-1.0) * (math.sin(theta3i)) * (theta3i_p))
460     jix = (A) - (B)
461     return jix
462
463
464 # /-----/
465 # /----- Derivada  $J_{iy}$  -----/
466 # /-----/
467 def calculo_jiy_der(phi_deg,
468                     theta1i_deg, theta2i_deg, theta3i_deg,
469                     theta1i_deg_p, theta2i_deg_p, theta3i_deg_p):
470     phi = phi_deg * dtr
471     theta1i = theta1i_deg * dtr
472     theta2i = theta2i_deg * dtr
473     theta3i = theta3i_deg * dtr
474
475     theta1i_p = theta1i_deg_p * dtr
476     theta2i_p = theta2i_deg_p * dtr
477     theta3i_p = theta3i_deg_p * dtr
478

```

```

479     A1 = (-1.0) * (math.sin(theta1i + theta2i)) *\n480         (math.sin(theta3i)) * (theta1i_p + theta2i_p)\n481     A2 = (math.cos(theta1i + theta2i)) * (math.cos(theta3i)) * (theta3i_p)\n482     A = (math.sin(phi)) * ((A1) + (A2))\n483     B = (math.cos(phi)) * ((-1.0) * (math.sin(theta3i)) * (theta3i_p))\n484     jiy = (A) + (B)\n485     return jiy\n486\n487\n488 # /-----/\n489 # /----- Derivada Jiz -----/\n490 # /-----/\n491 def calculo_jiz_der(phi_deg,\n492                     theta1i_deg, theta2i_deg, theta3i_deg,\n493                     theta1i_deg_p, theta2i_deg_p, theta3i_deg_p):\n494     phi = phi_deg * dtr\n495     theta1i = theta1i_deg * dtr\n496     theta2i = theta2i_deg * dtr\n497     theta3i = theta3i_deg * dtr\n498\n499     theta1i_p = theta1i_deg_p * dtr\n500     theta2i_p = theta2i_deg_p * dtr\n501     theta3i_p = theta3i_deg_p * dtr\n502\n503     A = (math.cos(theta1i + theta2i)) * (math.sin(theta3i)) * ((theta1i_p) + (theta2i_p))\n504     B = (math.sin(theta1i + theta2i)) * (math.cos(theta3i)) * (theta3i_p)\n505     jiz = (A) + (B)\n506     return jiz\n507\n508 ##### [ Derivada Jtheta ] #####\n509 ##### [ Derivada Jtheta1 ] #####\n510 ##### [ Derivada Jtheta2 ] #####\n511 ##### [ Derivada Jtheta3 ] #####\n512\n513 # /-----/\n514 # /----- Derivada jtheta1 -----/\n515 # /-----/\n516 def calculo_jtheta1_der(phi_deg,\n517                         theta2i_deg, theta3i_deg,\n518                         theta2i_deg_p, theta3i_deg_p):\n519     phi = phi_deg * dtr # Motor 1\n520     theta2i = theta2i_deg * dtr\n521     theta3i = theta3i_deg * dtr\n522     theta2i_p = theta2i_deg_p * dtr\n523     theta3i_p = theta3i_deg_p * dtr\n524     aa = rf\n525\n526     A = (math.cos(theta2i)) * (math.sin(theta3i)) * (theta2i_p)

```

```

527     B = (math.sin(theta2i)) * (math.cos(theta3i)) * (theta3i_p)
528     jtheta1 = (aa) * (A + B)
529
530     return jtheta1
531
532
533 # /-----/
534 # ----- Derivada jtheta2 -----/
535 # /-----/
536 def calculo_jtheta2_der(phi_deg,
537                         theta2i_deg, theta3i_deg,
538                         theta2i_deg_p, theta3i_deg_p):
539     phi = phi_deg * dtr # Motor 2
540     theta2i = theta2i_deg * dtr
541     theta3i = theta3i_deg * dtr
542     theta2i_p = theta2i_deg_p * dtr
543     theta3i_p = theta3i_deg_p * dtr
544     aa = rf
545
546     A = (math.cos(theta2i)) * (math.sin(theta3i)) * (theta2i_p)
547     B = (math.sin(theta2i)) * (math.cos(theta3i)) * (theta3i_p)
548     jtheta2 = (aa) * (A + B)
549
550     return jtheta2
551
552
553 # /-----/
554 # ----- Derivada jtheta3 -----/
555 # /-----/
556 def calculo_jtheta3_der(phi_deg,
557                         theta2i_deg, theta3i_deg,
558                         theta2i_deg_p, theta3i_deg_p):
559     phi = phi_deg * dtr # Motor 3
560     theta2i = theta2i_deg * dtr
561     theta3i = theta3i_deg * dtr
562     theta2i_p = theta2i_deg_p * dtr
563     theta3i_p = theta3i_deg_p * dtr
564     aa = rf
565
566     A = (math.cos(theta2i)) * (math.sin(theta3i)) * (theta2i_p)
567     B = (math.sin(theta2i)) * (math.cos(theta3i)) * (theta3i_p)
568     jtheta3 = (aa) * (A + B)
569
570     return jtheta3
571
572
573 ##### [ Jacobian Total Aceleracion Matricial ] #####
574

```

```

575 ######
576 def jacobian_total_macel(m_px, m_py, m_pz,
577                         m_pxp, m_pyp, m_pzp,
578                         m_pxpp, m_pypp, m_pzpp,
579                         m_theta11, m_theta12, m_theta13):
580     global dtr
581     global mmmtm
582     global rtd
583
584     tamano = len(m_px)
585
586     # Guarda Acleraciones thetas
587     m_acelth1 = np.zeros((tamano))
588     m_acelth2 = np.zeros((tamano))
589     m_acelth3 = np.zeros((tamano))
590
591     # Guarda velocidad thetas
592     m_th1_p = np.zeros((tamano))
593     m_th2_p = np.zeros((tamano))
594     m_th3_p = np.zeros((tamano))
595
596     # Para multiplicar matricialmente, son momentaneas
597     m_acelth = np.zeros((3, 1))
598     m_thetas = np.zeros((3, 1))
599     m_velxyz = np.zeros((3, 1))
600     m_acelxyz = np.zeros((3, 1))
601
602     for i in range(0, tamano):
603         # Jacobiano
604         [jinv, jxx, jinv_der, jxx_der,
605          theta31, theta32, theta33,
606          theta21, theta22, theta23] = jacobian_total_der(m_px[i], m_py[i], m_pz[i],
607          m_theta11[i], m_theta12[i], m_theta13[i],
608          m_pxp[i], m_pyp[i], m_pzp[i])
609
610         # Cambiar a SI
611         # Cambiar sistema de referencia LOCAL
612         m_velxyz[0, 0] = (-m_pyp[i]) * (mmmtm)
613         m_velxyz[1, 0] = (-m_pxp[i]) * (mmmtm)
614         m_velxyz[2, 0] = (-m_pzp[i]) * (mmmtm)
615
616         # Matriz Velocida Angular Thetas
617         # Cambiar a SI
618         # Velocidad tetas en sistema de referencia LOCAL
619         m_thetas = (inv(jinv)).dot(jxx.dot(m_velxyz))
620
621         # Cambiar a SI
622         m_acelxyz[0, 0] = (-m_pypp[i]) * (mmmtm)

```

```

623     m_acelxyz[1, 0] = (-m_pxpp[i]) * (mmtm)
624     m_acelxyz[2, 0] = (-m_pzpp[i]) * (mmtm)
625
626     # Vxyz=J*Vthetas
627     m_acelth = (inv(jinv)).dot(((jxx_der).dot(m_velxyz)) +
628                               ((jxx).dot(m_acelxyz)) -
629                               ((jinv_der).dot(m_thetas)))
630
631     # Guardar thetas velocidad
632     # Cambiar sistema de referencia GLOBAL
633     m_th1_p[i] = m_thetas[1, 0]
634     m_th2_p[i] = m_thetas[0, 0]
635     m_th3_p[i] = m_thetas[2, 0]
636
637     # Cambiar a sistemas de Referencia Global XYZ
638     m_acelth1[i] = (+1) * m_acelth[1, 0]
639     m_acelth2[i] = (+1) * m_acelth[0, 0]
640     m_acelth3[i] = (+1) * m_acelth[2, 0]
641
642     return [m_th1_p * rtd, m_th2_p * rtd, m_th3_p * rtd,
643             m_acelth1 * rtd, m_acelth2 * rtd, m_acelth3 * rtd]
644
645 #####      FIN      #####

```

B.8.8. jacobian_Paderborn_tm1_v2_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo: Determinar Jacobiano Metodo B
5
6 ##### [Importar Librerias] #####
7 ##### [Parametros] #####
8 ##### [Jacobiano Total] #####
9 import math
10 import numpy as np
11 from numpy.linalg import inv
12 import pd_tm1_adams
13
14 ##### [Constantes Trigonometricas]
15 ##### [Jacobiano Paderborn] #####
16
17 e = pd_tm1_adams.e()
18 f = pd_tm1_adams.f()
19 re = pd_tm1_adams.l2()
20 rf = pd_tm1_adams.l1()
21 ra = pd_tm1_adams.ra()
22 rb = pd_tm1_adams.rb()
23 R = ra - rb
24
25 # Constantes Trigonometricas
26 sqrt3 = pd_tm1_adams.sqrt3()
27 pi = pd_tm1_adams.pi()
28 sin120 = sqrt3 / 2.0
29 cos120 = pd_tm1_adams.cos120()
30 tan60 = sqrt3
31 sin30 = pd_tm1_adams.sin30()
32 tan30 = pd_tm1_adams.tan30()
33 cos30 = pd_tm1_adams.cos30()
34
35 dtr = pd_tm1_adams.dtr()
36 mtmm = pd_tm1_adams.mtmm()
37 mmtm = pd_tm1_adams.mmtm()
38 rtd = pd_tm1_adams.rtd()
39
40 ##### [Jacobiano Total] #####
41 ##### [Jacobiano Paderborn] #####
42
43 # /-----/
44 # /----- jacobian Paderborn -----/
45 # /-----/
```

```

47 def jacobian_calculo_Paderborn(pox, moy, poz,
48                               thetai1, thetai2, thetai3):
49     # Cambio unidades Internacional
50     pxx = pox * mmtm
51     pyy = moy * mmtm
52     pzz = poz * mmtm
53
54     # Ordenar segun sistema LOCAL
55     punto = [-pyy, pxx, pzz]
56     phi_orden = [0 * dtr, 120 * dtr, 240 * dtr]
57     thetai_orden = [thetai2 * dtr, thetai3 * dtr, thetai1 * dtr]
58
59     s1 = si(punto[0], punto[1], punto[2], phi_orden[0], thetai_orden[0])
60     s2 = si(punto[0], punto[1], punto[2], phi_orden[1], thetai_orden[1])
61     s3 = si(punto[0], punto[1], punto[2], phi_orden[2], thetai_orden[2])
62
63     s1t = np.transpose(s1)
64     s2t = np.transpose(s2)
65     s3t = np.transpose(s3)
66
67     b1 = bi(phi_orden[0], thetai_orden[0])
68     b2 = bi(phi_orden[1], thetai_orden[1])
69     b3 = bi(phi_orden[2], thetai_orden[2])
70
71     jx = np.zeros((3, 3))
72     jtheta = np.zeros((3, 3))
73
74     jx[0, 0] = s1t[0, 0]
75     jx[0, 1] = s1t[0, 1]
76     jx[0, 2] = s1t[0, 2]
77
78     jx[1, 0] = s2t[0, 0]
79     jx[1, 1] = s2t[0, 1]
80     jx[1, 2] = s2t[0, 2]
81
82     jx[2, 0] = s3t[0, 0]
83     jx[2, 1] = s3t[0, 1]
84     jx[2, 2] = s3t[0, 2]
85
86     jtheta[0, 0] = np.dot(s1t, b1)
87     jtheta[1, 1] = np.dot(s2t, b2)
88     jtheta[2, 2] = np.dot(s3t, b3)
89
90     jtheta = jtheta * (-1.0)
91
92     Jtotal = (np.linalg.inv(jx)).dot(jtheta)
93
94     return [jx, jtheta, Jtotal]

```

```

95 #####
96 ##### [ Funciones interiores de Jacobian Total] #####
97 #####
98 # /-----/
99 # /----- Si -----/
100 # /-----/
101 # /-----/
102 # Matriz si, devuelve matriz 3x1
103 def si(pox, poy, poz,
104         phi,
105         thetai):
106     angle_brazo = thetai
107     si_1 = np.zeros((3, 1))
108     si_2 = np.zeros((3, 1))
109
110     si_1[0, 0] = pox
111     si_1[1, 0] = poy
112     si_1[2, 0] = poz
113
114     si_2[0, 0] = (R) + (rf * math.cos(angle_brazo))
115     si_2[1, 0] = 0.0
116     si_2[2, 0] = (-1.0) * rf * math.sin(angle_brazo)
117
118     si = (si_1) - (np.dot((matri_rot(phi)), si_2))
119
120     return si
121
122 # /-----/
123 # /----- Bi -----/
124 # /-----/
125 # Matriz bi devuelve matriz 3x1
126 def bi(phi, thetai):
127     angle_brazo = thetai
128     bi_1 = np.zeros((3, 1))
129
130     bi_1[0, 0] = (-1.0) * rf * math.sin(angle_brazo)
131     bi_1[1, 0] = 0.0
132     bi_1[2, 0] = (-1.0) * rf * math.cos(angle_brazo)
133
134     bi_matrix = np.dot(matri_rot(phi), bi_1)
135
136     bi_matrix = (-1.0) * bi_matrix
137     return bi_matrix
138
139 # /-----/
140 # /----- Matriz de Rotacion -----/
141 # /-----/
142 def matri_rot(phi):

```

```

143     angle_motor = phi
144     Rot = np.zeros((3, 3))
145
146     Rot[0, 0] = math.cos(angle_motor)
147     Rot[0, 1] = (-1.0) * math.sin(angle_motor)
148     Rot[1, 0] = math.sin(angle_motor)
149     Rot[1, 1] = math.cos(angle_motor)
150     Rot[2, 2] = (1.0)
151
152     return Rot
153
154 ##### [Jacobian Total Matricial] #####
155 #####
156 def jacobian_total_mvel(m_px, m_py, m_pz,
157                         m_theta11, m_theta12, m_theta13,
158                         m_der_theta11, m_der_theta12, m_der_theta13):
159     global dtr
160     global mmtm
161
162     tamano = len(m_px)
163     m_velx = np.zeros((tamano))
164     m_vely = np.zeros((tamano))
165     m_velz = np.zeros((tamano))
166
167     m_vel = np.zeros((3, 1))
168     m_thetas = np.zeros((3, 1))
169
170     for i in range(0, tamano - 1):
171         # NO Cambiar a SI
172         m_theta1d = m_theta11[i] * (1)
173         m_theta2d = m_theta12[i] * (1)
174         m_theta3d = m_theta13[i] * (1)
175
176         # Jacobiano
177         [jx, jtheta, Jtotal] = jacobian_calculo_Paderborn(m_px[i], m_py[i], m_pz[i],
178                                                       m_theta1d, m_theta2d,
179                                                       m_theta3d)
180
181         # Matriz Velocida Angular Thetas
182         # Cambiar a SI
183         m_thetas[0, 0] = (m_der_theta12[i] * (+1)) * (dtr)
184         m_thetas[1, 0] = (m_der_theta13[i] * (+1)) * (dtr)
185         m_thetas[2, 0] = (m_der_theta11[i] * (+1)) * (dtr)
186
187         # Vxyz=J*Vthetas
188         m_vel = Jtotal.dot(m_thetas)
189
190         # Cambiar a sistemas de Referencia Global XYZ

```

```

190     m_velx[i] = (+1) * m_vel[1, 0]
191     m_vely[i] = (-1) * m_vel[0, 0]
192     m.velz[i] = (+1) * m.vel[2, 0]
193
194     # Cambiar salida [mm/s]
195     return [m.velx * mtmm, m.vely * mtmm, m.velz * mtmm]
196
197 ##### [ Jacobian Derivadas Total ] #####
198 ##### [ Jacobian Derivadas Total ] #####
199 ##### [ Jacobian Derivadas Total ] #####
200 # /-----/
201 # /--- jacobian_total_tm1 (PUNTUAL) --/
202 # /-----/
203 def jacobian_total_tm1(pox, poy, poz,
204                         thetai1, thetai2, thetai3,
205                         pox_p, poy_p, poz_p):
206     # Cambio unidades Internacional
207     pxx = pox * mmtm
208     pyy = poy * mmtm
209     pzz = poz * mmtm
210
211     pxx_p = pox_p * mmtm
212     pyy_p = poy_p * mmtm
213     pzz_p = poz_p * mmtm
214
215     # Ordenar segun sistema referencia LOCAL
216     punto = [-pyy, pxx, pzz]
217     phi_orden = [0 * dtr, 120 * dtr, 240 * dtr]
218     thetai_orden = [thetai2 * dtr, thetai3 * dtr, thetai1 * dtr]
219     punto_p_i = [-pyy_p, pxx_p, pzz_p]
220
221     s1 = si(punto[0], punto[1], punto[2], phi_orden[0], thetai_orden[0])
222     s2 = si(punto[0], punto[1], punto[2], phi_orden[1], thetai_orden[1])
223     s3 = si(punto[0], punto[1], punto[2], phi_orden[2], thetai_orden[2])
224
225     s1t = np.transpose(s1)
226     s2t = np.transpose(s2)
227     s3t = np.transpose(s3)
228
229     b1 = bi(phi_orden[0], thetai_orden[0])
230     b2 = bi(phi_orden[1], thetai_orden[1])
231     b3 = bi(phi_orden[2], thetai_orden[2])
232
233     jx = np.zeros((3, 3))
234     jtheta = np.zeros((3, 3))
235
236     sitt = np.zeros((3, 3))
237

```

```

238     sitt[0, 0] = s1t[0, 0]
239     sitt[0, 1] = s1t[0, 1]
240     sitt[0, 2] = s1t[0, 2]
241
242     sitt[1, 0] = s2t[0, 0]
243     sitt[1, 1] = s2t[0, 1]
244     sitt[1, 2] = s2t[0, 2]
245
246     sitt[2, 0] = s3t[0, 0]
247     sitt[2, 1] = s3t[0, 1]
248     sitt[2, 2] = s3t[0, 2]
249
250     jx[0, 0] = s1t[0, 0]
251     jx[0, 1] = s1t[0, 1]
252     jx[0, 2] = s1t[0, 2]
253
254     jx[1, 0] = s2t[0, 0]
255     jx[1, 1] = s2t[0, 1]
256     jx[1, 2] = s2t[0, 2]
257
258     jx[2, 0] = s3t[0, 0]
259     jx[2, 1] = s3t[0, 1]
260     jx[2, 2] = s3t[0, 2]
261
262     jtheta[0, 0] = np.dot(s1t, b1)
263     jtheta[1, 1] = np.dot(s2t, b2)
264     jtheta[2, 2] = np.dot(s3t, b3)
265
266     jtheta = jtheta * (-1.0)
267
268     Jtotal = ((np.linalg.inv(jx)).dot(jtheta))
269
270     Jtotal_inv = np.linalg.inv(Jtotal)
271     theta_i_p = Jtotal_inv.dot([[punto_p_i[0]], [punto_p_i[1]], [punto_p_i[2]]])
272
273     punto_p = np.zeros(3)
274     punto_p[0] = punto_p_i[0]
275     punto_p[1] = punto_p_i[1]
276     punto_p[2] = punto_p_i[2]
277
278     s1_p = si_p(punto_p[0], punto_p[1], punto_p[2],
279                  phi_orden[0], theta_i_orden[0], theta_i_p[0])
280     s2_p = si_p(punto_p[0], punto_p[1], punto_p[2],
281                  phi_orden[1], theta_i_orden[1], theta_i_p[1])
282     s3_p = si_p(punto_p[0], punto_p[1], punto_p[2],
283                  phi_orden[2], theta_i_orden[2], theta_i_p[2])
284
285     s1t_p = np.transpose(s1_p)

```

```

286     s2t_p = np.transpose(s2_p)
287     s3t_p = np.transpose(s3_p)
288
289     b1_p = bi_p(phi_orden[0], thetai_orden[0], thetai_p[0])
290     b2_p = bi_p(phi_orden[1], thetai_orden[1], thetai_p[1])
291     b3_p = bi_p(phi_orden[2], thetai_orden[2], thetai_p[2])
292
293     sit_p = np.zeros((3, 3))
294     K_p = np.zeros((3, 3))
295
296     sit_p[0, 0] = s1t_p[0, 0]
297     sit_p[0, 1] = s1t_p[0, 1]
298     sit_p[0, 2] = s1t_p[0, 2]
299
300     sit_p[1, 0] = s2t_p[0, 0]
301     sit_p[1, 1] = s2t_p[0, 1]
302     sit_p[1, 2] = s2t_p[0, 2]
303
304     sit_p[2, 0] = s3t_p[0, 0]
305     sit_p[2, 1] = s3t_p[0, 1]
306     sit_p[2, 2] = s3t_p[0, 2]
307
308     K_p[0, 0] = (np.dot(s1t_p, b1)) + (np.dot(s1t, b1_p))
309     K_p[1, 1] = (np.dot(s2t_p, b2)) + (np.dot(s2t, b2_p))
310     K_p[2, 2] = (np.dot(s3t_p, b3)) + (np.dot(s3t, b3_p))
311
312     return [jx, jtheta, Jtotal, sit_p, K_p, thetai_p, sitt]
313
314 ##### [ Funciones interiores de Jacobian Derivada Total ] #####
315 # /-----/
316 # /----- Si Punto -----/
317 # /-----/
318 # /----- Si -----/
319 # /-----/
320 # Derivada de Si
321 def si_p(px_p, py_p, pz_p,
322         phi,
323         thetai, thetai_p):
324     angle_brazo = thetai
325     vel_brazo = thetai_p
326     si_1 = np.zeros((3, 1))
327     si_2 = np.zeros((3, 1))
328
329     si_1[0, 0] = px_p
330     si_1[1, 0] = py_p
331     si_1[2, 0] = pz_p
332
333     si_2[0, 0] = (rf * math.sin(angle_brazo)) * (vel_brazo)

```

```

334     si_2[1, 0] = 0.0
335     si_2[2, 0] = (rf * math.cos(angle_brazo)) * (vel_brazo)
336
337     si = (si_1) + (np.dot((matri_rot(phi)), si_2))
338     return si
339
340 # /-----/
341 # /----- Bi Punto -----/
342 # /-----/
343 # Derivada de Bi
344 def bi_p(phi, thetai, thetai_p):
345     angle_brazo = thetai
346     vel_brazo = thetai_p
347     bi_1 = np.zeros((3, 1))
348
349     bi_1[0, 0] = (1.0) * (rf * math.cos(angle_brazo)) * (vel_brazo)
350     bi_1[1, 0] = 0.0
351     bi_1[2, 0] = (-1.0) * (rf * math.sin(angle_brazo)) * (vel_brazo)
352
353     bi_matrix = np.dot(matri_rot(phi), bi_1)
354
355     bi_matrix = (1.0) * bi_matrix
356     return bi_matrix
357
358 ##### [ Jacobian Total Aceleracion Matricial ] #####
359 ##### [ Jacobian Total Aceleracion Matricial ] #####
360 ##### [ Jacobian Total Aceleracion Matricial ] #####
361 def jacobian_total_macel(m_px, m_py, m_pz,
362                         m_px_p, m_py_p, m_pz_p,
363                         m_thetai1, m_thetai2, m_thetai3,
364                         m_pxpp, m_pypp, m_pzpp):
365     global dtr
366     global mmtm
367     global rtd
368
369     tamano = len(m_px)
370     # Para Guardar Aceleraciones thetas
371     m_acelth1 = np.zeros((tamano))
372     m_acelth2 = np.zeros((tamano))
373     m_acelth3 = np.zeros((tamano))
374
375     # Para Guardar velocidad thetas
376     m_th1_p = np.zeros((tamano))
377     m_th2_p = np.zeros((tamano))
378     m_th3_p = np.zeros((tamano))
379
380     # Para Guardar Jacobianos
381     m_jac = np.zeros((tamano, 9))

```

```

382
383     # Para multiplicar matricialmente, son momentaneas
384     m_acelth = np.zeros((3, 1))
385     m_thetas = np.zeros((3, 1))
386     m_velxyz = np.zeros((3, 1))
387     m_acelxyz = np.zeros((3, 1))
388
389     # Para Guardar Jacobianos
390     m_jac_der = np.zeros((tamano, 9))
391
392     for i in range(0, tamano):
393         # Jacobiano
394         [jx, jtheta, Jtotal,
395          si_p, K_p, thetai_p, si] = jacobian_total_tm1(m_px[i], m_py[i], m_pz[i],
396          m_theta11[i], m_theta12[i], m_theta13[i],
397          m_px_p[i], m_py_p[i], m_pz_p[i])
398
399         # Matriz Velocida Angular Thetas
400         m_thetas = thetai_p
401
402         # Guardar jacobiano para torque
403         m_jac[i, 0] = Jtotal[0, 0]
404         m_jac[i, 1] = Jtotal[0, 1]
405         m_jac[i, 2] = Jtotal[0, 2]
406         m_jac[i, 3] = Jtotal[1, 0]
407         m_jac[i, 4] = Jtotal[1, 1]
408         m_jac[i, 5] = Jtotal[1, 2]
409         m_jac[i, 6] = Jtotal[2, 0]
410         m_jac[i, 7] = Jtotal[2, 1]
411         m_jac[i, 8] = Jtotal[2, 2]
412
413         # Cambiar a SI
414         m_acelxyz[0, 0] = (-m_pypp[i]) * (mmtm)
415         m_acelxyz[1, 0] = (m_pxpp[i]) * (mmtm)
416         m_acelxyz[2, 0] = (m_pzpp[i]) * (mmtm)
417
418         # aceleracion thetas
419         A = inv(Jtotal)
420         B1 = m_acelxyz
421         C1 = inv(jx)
422         C2 = ((si_p).dot(Jtotal)) + (K_p)
423         B2 = ((C1).dot(C2)).dot(m_thetas)
424         B = (B1 + B2)
425         m_acelth = (A).dot(B)
426
427         # Jacobiano Derivada
428         Jder = (-1) * (inv(si)).dot(((si_p).dot(Jtotal)) + (K_p))
429         # guardar jacobiano

```

```

430     m_jac_der[i, 0] = Jder[0, 0]
431     m_jac_der[i, 1] = Jder[0, 1]
432     m_jac_der[i, 2] = Jder[0, 2]
433     m_jac_der[i, 3] = Jder[1, 0]
434     m_jac_der[i, 4] = Jder[1, 1]
435     m_jac_der[i, 5] = Jder[1, 2]
436     m_jac_der[i, 6] = Jder[2, 0]
437     m_jac_der[i, 7] = Jder[2, 1]
438     m_jac_der[i, 8] = Jder[2, 2]

439
440     # Cambiar a sistemas de Referencia Global XYZ
441     m_acelth1[i] = (+1) * m_acelth[2, 0]
442     m_acelth2[i] = (+1) * m_acelth[0, 0]
443     m_acelth3[i] = (+1) * m_acelth[1, 0]

444
445     # Guardar theta velocidad resultado ya esta ordenado
446     # desde jacobian total para el sistema Metodo B
447     m_th1_p[i] = m_thetas[2]
448     m_th2_p[i] = m_thetas[0]
449     m_th3_p[i] = m_thetas[1]

450
451     # Cambiar salida [mm/s] Y grados
452     return [m_th1_p * rtd, m_th2_p * rtd, m_th3_p * rtd,
453             m_jac, m_jac_der,
454             m_acelth1 * rtd, m_acelth2 * rtd, m_acelth3 * rtd]

455
456 #####      FIN      #####

```

B.8.9. torque_m1_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo: Determinar torque metodo A
5
6 ##### [Importar Librerias] #####
7 ##### [Parametros] #####
8
9 import math
10 import pd_tm1_adams
11 import numpy as np
12 from numpy.linalg import inv
13
14 ##### [Constantes Trigonometricas]
15 ##### [Trigonometric constants]
16
17 ra = pd_tm1_adams.ra()
18 rb = pd_tm1_adams.rb()
19 l1 = pd_tm1_adams.l1()
20 l2 = pd_tm1_adams.l2()
21 rdif = (ra - rb)
22
23 # Constantes Trigonometricas
24 m1 = pd_tm1_adams.m1()
25 m_elbow = pd_tm1_adams.m_elbow()
26 m2 = pd_tm1_adams.m2()
27 i_motor = pd_tm1_adams.inercia_m()
28 g = pd_tm1_adams.gg()
29 mp = pd_tm1_adams.mp()
30
31 # Trigonometric constants
32 sqrt3 = pd_tm1_adams.sqrt3()
33 pi = pd_tm1_adams.pi()
34 sin120 = sqrt3 / 2.0
35 cos120 = pd_tm1_adams.cos120()
36 tan60 = sqrt3
37 sin30 = pd_tm1_adams.sin30()
38 tan30 = pd_tm1_adams.tan30()
39
40 dtr = pd_tm1_adams.dtr() # degrees to radians
41 mtmm = pd_tm1_adams.mtmm() # Metros a Milimetros
42 mmmtm = pd_tm1_adams.mmmtm() # Milimetros a Metros
43 kgm2tgrmm2 = pd_tm1_adams.kgm2tgrmm2()
44 rtd = pd_tm1_adams.rtd() # degrees to radians
45
46 #####
```

```

47 ##### [Torque Menu Input] #####
48 #####
49 # /-----/
50 # /----- Torque TOTAL (puntual) -----/
51 # /-----/
52 def ti(theta11_pp_deg, theta12_pp_deg, theta13_pp_deg,
53         theta11_deg, theta12_deg, theta13_deg,
54         xp_mm, yp_mm, zp_mm,
55         xp_pp_mm, yp_pp_mm, zp_pp_mm,
56         fpx, fpy, fpz,
57         m_payload):
58     # Cambiar unidades a SI
59     # Cambiar orden angulos y coordenadas cartesianas a sistema referencia LOCAL
60     phi = [0 * dtr, 120 * dtr, 240 * dtr]
61     theta1i_pp = [theta12_pp_deg * dtr, theta11_pp_deg * dtr, theta13_pp_deg * dtr]
62     theta1i = [theta12_deg * dtr, theta11_deg * dtr, theta13_deg * dtr]
63     punto = [-yp_mm * mmtm, -xp_mm * mmtm, -zp_mm * mmtm]
64     punto_pp = [-yp_pp_mm * mmtm, -xp_pp_mm * mmtm, -zp_pp_mm * mmtm]
65     fp = [-fpy, -fpx, -fpz]
66
67     lambdai = system_tq(phi, theta1i, punto, fp, punto_pp, m_payload)
68
69     t1 = ti_puntual(phi[0], theta1i_pp[0], theta1i[0],
70                      punto[0], punto[1], punto[2], lambdai[0, 0])
71     t2 = ti_puntual(phi[1], theta1i_pp[1], theta1i[1],
72                      punto[0], punto[1], punto[2], lambdai[1, 0])
73     t3 = ti_puntual(phi[2], theta1i_pp[2], theta1i[2],
74                      punto[0], punto[1], punto[2], lambdai[2, 0])
75
76     # Cambio orden sistema referencia GLOBAL
77     return [t2, t1, t3]
78
79 #####
80 ##### [Torque puntual motor i] #####
81 #####
82 def ti_puntual(phi,
83                 theta1i_pp, theta1i,
84                 xp, yp, zp,
85                 lambdai):
86     A = (((m1 / 3.0) + (m2)) * (l1 ** 2.0) * (theta1i_pp))
87     B = (((m1 / 2.0) + (m2)) * (g * l1) * (math.cos(theta1i))) # g
88
89     C1 = ((xp) * (math.cos(phi))) + ((yp) * (math.sin(phi))) - (rdif) *
90           (math.sin(theta1i))
91     C2 = zp * (math.cos(theta1i))
92     C = C1 - C2
93
94     torq_2 = (A) # Inercia BRazo      #0x00

```

```

94     torq_3 = (-B)  # Peso Brazo-Antebrazo
95     torq_1 = ((-1) * 2.0 * lambdai * 11 * C)  # Peso efecto +  #0x00
96
97     # Efecto + Inercia BRazo + Peso brazo-antebrazo
98     torq = (torq_1 * (1)) + (torq_2 * (1)) + (torq_3 * (1))
99     return torq
100
101 ##### [Multiplicadores de Lagrange] #####
102 ##### /-----/
103 ##### /-----Sistema Ecuaciones -----/
104 ##### /-----/
105
106
107 def system_tq(phi, thetaii, punto, fp, punto_pp, m_payload):
108     # Creacion Matrices
109     m_A = np.zeros((3, 3))
110     m_B = np.zeros((3, 1))
111     m_lamda = np.zeros((3, 1))
112
113     fpx = fp[0]
114     fpy = fp[1]
115     fpz = fp[2]
116     xp_pp = punto_pp[0]
117     yp_pp = punto_pp[1]
118     zp_pp = punto_pp[2]
119
120     # Matriz A
121     # x
122     m_A[0, 0] = maxx(phi[0], thetaii[0], punto[0])
123     m_A[0, 1] = maxx(phi[1], thetaii[1], punto[0])
124     m_A[0, 2] = maxx(phi[2], thetaii[2], punto[0])
125     # y
126     m_A[1, 0] = mayy(phi[0], thetaii[0], punto[1])
127     m_A[1, 1] = mayy(phi[1], thetaii[1], punto[1])
128     m_A[1, 2] = mayy(phi[2], thetaii[2], punto[1])
129     # z
130     m_A[2, 0] = mazz(thetaii[0], punto[2])
131     m_A[2, 1] = mazz(thetaii[1], punto[2])
132     m_A[2, 2] = mazz(thetaii[2], punto[2])
133
134     # Matriz B
135     m_B[0, 0] = ((mnt(m_payload) + (3.0 * m2)) * ((1) * (xp_pp))) - (fpx)
136     m_B[1, 0] = ((mnt(m_payload) + (3.0 * m2)) * ((1) * (yp_pp))) - (fpy)
137     m_B[2, 0] = ((mnt(m_payload) + (3.0 * m2)) * ((1) * (zp_pp - g))) - (fpz)
138
139     m_lamda = (inv(m_A)).dot(m_B)
140     return m_lamda
141

```

```

142 # /-----/
143 # /----- M_AX -----/
144 # /-----/
145 def maxx(phi, theta1i, xp):
146     valor = 2.0 * ((xp) - (rdif * (math.cos(phi))) -
147                     (l1 * (math.cos(theta1i)) * (math.cos(phi))))
148     return valor
149
150 # /-----/
151 # /----- M_AY -----/
152 # /-----/
153 def mayy(phi, theta1i, yp):
154     valor = 2.0 * ((yp) - (rdif * (math.sin(phi))) -
155                     (l1 * (math.cos(theta1i)) * (math.sin(phi))))
156     return valor
157
158 # /-----/
159 # /----- M_AZ -----/
160 # /-----/
161 def mazz(theta1i, zp):
162     valor = 2.0 * (zp - (l1 * (math.sin(theta1i))))
163     return valor
164
165 ##### [Torque Matricial] #####
166 ##### [Torque TOTAL] #####
167
168 # /-----/
169 # /----- Torque TOTAL -----/
170 # /-----/
171 def ti_matriz(theta11_pp_deg, theta12_pp_deg, theta13_pp_deg,
172                 theta11_deg, theta12_deg, theta13_deg,
173                 xp_mm, yp_mm, zp_mm,
174                 xp_pp_mm, yp_pp_mm, zp_pp_mm,
175                 fpx, fpy, fpz,
176                 m_payload):
177     tamano = len(xp_mm)
178     tm1 = np.zeros(tamano)
179     tm2 = np.zeros(tamano)
180     tm3 = np.zeros(tamano)
181
182     for i in range(0, tamano):
183         t231 = ti(theta11_pp_deg[i], theta12_pp_deg[i], theta13_pp_deg[i],
184                    theta11_deg[i], theta12_deg[i], theta13_deg[i],
185                    xp_mm[i], yp_mm[i], zp_mm[i],
186                    xp_pp_mm[i], yp_pp_mm[i], zp_pp_mm[i],
187                    fpx[i], fpy[i], fpz[i],
188                    m_payload)
189         tm1[i] = t231[0]

```

```
190     tm2[i] = t231[1]
191     tm3[i] = t231[2]
192     return [tm1, tm2, tm3]
193
194 # /-----/
195 # /----- masa efecto TOTAL -----/
196 # /-----/
197 # masa efecto + masa objeto a levantar + masa dividida antebrazo Lb
198 def mnt(m_payload):
199     mass_mnt = mp + m_payload
200     return mass_mnt
201
202 ##### FIN #####
```

B.8.10. torque_m1_Paderborn_v2_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo: Determinar torque metodo B
5
6
7 ##### [Importar Librerias] #####
8 ##### [Parametros] #####
9 #####
10 import math
11 import numpy as np
12 from numpy.linalg import inv
13 import pd_tm1_adams
14
15 ##### [Constantes Trigonometricas]
16 ##### [Masas e inercias]
17 #####
18 ra = pd_tm1_adams.ra()
19 rb = pd_tm1_adams.rb()
20 la = pd_tm1_adams.l1()
21 lb = pd_tm1_adams.l2()
22 rdif = (ra - rb)
23
24 # Masas e inercias
25 ma = pd_tm1_adams.m1()
26 m_elbow = pd_tm1_adams.m_elbow()
27 mb = pd_tm1_adams.m2()
28 i_motor = pd_tm1_adams.inercia_m()
29 g = pd_tm1_adams.gg()
30 mc = pd_tm1_adams.mp()
31 r_mass = pd_tm1_adams.r_mass()
32 com = pd_tm1_adams.com()
33
34 # Constantes Trigonometricas
35 sqrt3 = pd_tm1_adams.sqrt3()
36 pi = pd_tm1_adams.pi()
37 sin120 = sqrt3 / 2.0
38 cos120 = pd_tm1_adams.cos120()
39 tan60 = sqrt3
40 sin30 = pd_tm1_adams.sin30()
41 tan30 = pd_tm1_adams.tan30()
42
43 dtr = pd_tm1_adams.dtr()
44 mtmm = pd_tm1_adams.mtmm()
45 mmtm = pd_tm1_adams.mmtm()
46 kgm2tgrmm2 = pd_tm1_adams.kgm2tgrmm2()
```

```

47 rtd = pd_tm1_adams.rtd()
48
49
50 ##### [Torque Menu Input] #####
51 #####
52 #####
53 # /-----/
54 # ----- Torque TOTAL puntual -----/
55 # /-----/
56 def ti(theta11_pp_deg, theta12_pp_deg, theta13_pp_deg,
57     theta11_deg, theta12_deg, theta13_deg,
58     xp_mm, yp_mm, zp_mm,
59     fpx, fpy, fpz,
60     JTotal_m, m_playload, JTotal_m_p,
61     theta11_p_deg, theta12_p_deg, theta13_p_deg):
62     # Cambiar unidades a SI
63     # Cambiar orden angulos y coordenadas cartesianas a sistema referencia LOCAL
64     phi = [0 * dtr, 120 * dtr, 240 * dtr]
65     theta1i_pp = [[theta12_pp_deg * dtr], [theta13_pp_deg * dtr], [theta11_pp_deg * dtr]]
66     theta1i = [theta12_deg * dtr, theta13_deg * dtr, theta11_deg * dtr]
67     punto = [[-yp_mm * mmtm], [xp_mm * mmtm], [zp_mm * mmtm]]
68     fp = [[-fpy], [fpx], [fpz + ((-1) * (g) * (mnt(m_playload)))]]
69
70     theta1i_p = [[theta12_p_deg * dtr], [theta13_p_deg * dtr], [theta11_p_deg * dtr]]
71
72     torques = ti_puntual(theta1i, theta1i_pp, JTotal_m, fp,
73     m_playload, JTotal_m_p, theta1i_p)
74
75     return [torques[0, 0], torques[1, 0], torques[2, 0]]
76
77
78 # /-----/
79 # ----- Torque puntual motor i -----/
80 # /-----/
81 def ti_puntual(theta1i, theta1i_pp,
82     JTotal, fg, m_playload, Jtotal_der,
83     theta1i_p):
84     # Torque Mass matrix
85     M1 = (inercia_b()).dot(theta1i_pp)
86     M2 = (((np.transpose(JTotal)).dot(JTotal)) * (mnt(m_playload))).dot(theta1i_pp)
87
88     # Torque Centrifugal and Coriolis coefficente matrix o force inerciales
89     C = (((np.transpose(JTotal)).dot(Jtotal_der)) * (mnt(m_playload))).dot(theta1i_p)
90
91     # Torque Fuerza gravitacional efector
92     G1 = (np.transpose(JTotal)).dot(fg)
93
94     # Torque fuerzas gravitacionales de los brazos

```

```

95     modulo = ((-g) * (com) * ((ma) + (m_elbow) + (2 * r_mass * mb)))
96     G2 = np.zeros((3, 1))
97     G2[0, 0] = modulo * math.cos(theta1i[0])
98     G2[1, 0] = modulo * math.cos(theta1i[1])
99     G2[2, 0] = modulo * math.cos(theta1i[2])
100
101    ti_m_2 = (M1) # Inercia Brazo
102    ti_m_3 = (G2) # Gravedad Brazo-Antebrazo
103    ti_m_5 = (-G1) # Gravedad efecto
104    ti_m_1 = (C)
105    ti_m_4 = (M2)
106
107    # Efecto + Inercia Brazo + Gravedad brazo-antebrazo
108    ti_m = (ti_m_1 * (1)) + (ti_m_4 * (1)) +\
109          (ti_m_5 * (1)) + (ti_m_2 * (1)) + (ti_m_3 * (1))
110
111    return ti_m
112
113
114 #####
115 ##### [Matrices Puntuales ] #####
116 #####
117 # /-----/
118 # /--- Matriz de inverica Brazo superior -----/
119 # /-----/
120 def inercia_b():
121     ib = np.zeros((3, 3))
122     i_rot_la = (la ** 2.0) * ((ma) / (3.0))
123     i_elbow = (la ** 2.0) * (m_elbow)
124     i_rot_lb = (la ** 2.0) * (2.0 * r_mass * mb)
125     valor = (i_motor) + (i_rot_la) + (i_elbow) + (i_rot_lb)
126     ib[0, 0] = valor
127     ib[1, 1] = valor
128     ib[2, 2] = valor
129     return ib
130
131
132 # /-----/
133 # /--- Masa efecto TOTAL -----/
134 # /-----/
135 # masa efecto + masa objeto a levantar + masa dividida antebrazo Lb
136 def mnt(m_payload):
137     mass_mnt = mc + m_payload + ((3.0) * (2.0) * (1.0 - r_mass) * (mb))
138     return mass_mnt
139
140
141 #####
142 ##### [Torque Matricial] #####

```

```

143 #####/#####
144 # /-----/
145 # /----- Torque TOTAL -----/
146 # /-----/
147 def ti_matriz(theta11_pp_deg, theta12_pp_deg, theta13_pp_deg,
148             theta11_deg, theta12_deg, theta13_deg,
149             xp_mm, yp_mm, zp_mm,
150             fpx, fpy, fpz,
151             m_jac, m_payload, m_jac_der,
152             theta11_p_deg, theta12_p_deg, theta13_p_deg):
153     tamano = len(xp_mm)
154     tm1 = np.zeros(tamano)
155     tm2 = np.zeros(tamano)
156     tm3 = np.zeros(tamano)
157
158     for i in range(0, tamano):
159         Jtotal = np.zeros((3, 3))
160
161         Jtotal[0, 0] = m_jac[i, 0]
162         Jtotal[0, 1] = m_jac[i, 1]
163         Jtotal[0, 2] = m_jac[i, 2]
164         Jtotal[1, 0] = m_jac[i, 3]
165         Jtotal[1, 1] = m_jac[i, 4]
166         Jtotal[1, 2] = m_jac[i, 5]
167         Jtotal[2, 0] = m_jac[i, 6]
168         Jtotal[2, 1] = m_jac[i, 7]
169         Jtotal[2, 2] = m_jac[i, 8]
170
171     Jtotal_der = np.zeros((3, 3))
172
173     Jtotal_der[0, 0] = m_jac_der[i, 0]
174     Jtotal_der[0, 1] = m_jac_der[i, 1]
175     Jtotal_der[0, 2] = m_jac_der[i, 2]
176     Jtotal_der[1, 0] = m_jac_der[i, 3]
177     Jtotal_der[1, 1] = m_jac_der[i, 4]
178     Jtotal_der[1, 2] = m_jac_der[i, 5]
179     Jtotal_der[2, 0] = m_jac_der[i, 6]
180     Jtotal_der[2, 1] = m_jac_der[i, 7]
181     Jtotal_der[2, 2] = m_jac_der[i, 8]
182
183     t231 = ti(theta11_pp_deg[i], theta12_pp_deg[i], theta13_pp_deg[i],
184                theta11_deg[i], theta12_deg[i], theta13_deg[i],
185                xp_mm[i], yp_mm[i], zp_mm[i],
186                fpx[i], fpy[i], fpz[i],
187                Jtotal, m_payload, Jtotal_der,
188                theta11_p_deg[i], theta12_p_deg[i], theta13_p_deg[i])
189
190     # Ordenar torques segun sistema referencia GLOBAL

```

```
191     tm1[i] = t231[2]
192     tm2[i] = t231[0]
193     tm3[i] = t231[1]
194
195     return [tm1, tm2, tm3]
196
197 ##### FIN #####
```

B.8.11. workspace_v2.py

```
1  #!/usr/bin/env python
2  # Nombre Creador: Ivan Alejandro Fernandez Gracia
3  # Universidad: Universidad de Santiago de Chile
4  # Mail: ivan.fernandez.g@usach.cl
5  # Objetivo: Crear espacio de trabajo del robot delta a partir de restricciones
6
7  ##### [Importar Librerias] #####
8  #####
9  #####
10 import math
11 import pd_tm1_adams
12 import delta_kinematics_t1m_adams
13 import jacobian_tm1_adams
14 import rospy
15 import numpy as np
16 import matplotlib.pyplot as plt
17 from mpl_toolkits.mplot3d import Axes3D
18 import plot_delta
19 #####
20 ##### [Importar mensajes] #####
21 #####
22 from simu_visual.msg import parameter_ws
23
24 #####
25 ##### [Parametros] #####
26 #####
27 e = pd_tm1_adams.e()
28 f = pd_tm1_adams.f()
29 re = pd_tm1_adams.l2()
30 rf = pd_tm1_adams.l1()
31
32 pi = pd_tm1_adams.pi()
33 dtr = pd_tm1_adams.dtr()
34
35
36 ##### [Funcion Principal] #####
37 #####
38 #####
39 # /-----/
40 # /----- NODO -----/
41 # /-----/
42 def nodo():
43     global resultados
44     global permiso
45     global step_global
46     global id_call
```

```

47
48 ##### {{ Inicio Nodo }} #####
49 rospy.init_node("workspace_delta", anonymous=False)
50 rate = rospy.Rate(7) # Hz
51
52 ##### {{ Varibales para comprobar mensaje entrante }} #####
53 permiso = False
54 step_global = 0
55 id_call = 0
56 id_permiso = 0
57 while not rospy.is_shutdown():
58     rate.sleep()
59
60 ##### {{ Topics }} #####
61 rospy.Subscriber("input_workspace", parameter_ws, callback) # listener
62
63 if (permiso == True) and (step_global != 0) and (id_call != id_permiso):
64     rospy.loginfo("Creando Trayectoria!")
65     # Array de puntos XYZ del espacio de trabajo
66     resultados = espaciotrabajo(step_global)
67     rospy.loginfo("Workspace Creado, numero puntos:" +
68                 str(resultados[1]) + " ID: " + str(id_call))
69     rospy.loginfo("Workspace Creado, numero puntos JXX:" +
70                 str(resultados[3]) + " ID: " + str(id_call))
71     rospy.loginfo("Workspace Creado, numero puntos JINV:" +
72                 str(resultados[5]) + " ID: " + str(id_call))
73
74     # Graficar puntos anteriores
75     plot_delta.graphws(resultados[0], resultados[2], resultados[4], resultados[6])
76     rospy.loginfo("Grafico Workspace Listo!")
77
78 ##### {{ Reset variable comprobar mensaje entrante}} #####
79 permiso = False
80 step_global = 0
81 id_permiso = id_call
82
83
84 # /-----/
85 # /----- Calcular Workspace -----/
86 # /-----/
87 def espaciotrabajo(deltaangulo):
88     ##### {{ Direccion txt para guardar ws }} #####
89     mi_path = "/home/ivan/r_delta/src/simu_visual/script/workspace.txt"
90
91     ##### {{ Crear matrices ws }} #####
92     size_rows = int(((180.0 / deltaangulo) + 1) ** 3)
93     espTra = np.zeros([size_rows, 6], dtype=float)
94     espTra_singu = np.zeros([size_rows, 6], dtype=float)

```

```

95 espTra_singu_inv = np.zeros([size_rows, 6], dtype=float)
96 espTra_real = np.zeros([size_rows, 6], dtype=float)
97
98 ##### {{ Restricciones limines de angulos 1i }} #####
99 res_ang_min = pd_tm1_adams.res_ang_min()
100 res_ang_max = pd_tm1_adams.res_ang_max()
101
102 cont = 0
103 cont_singu = 0
104 cont_singu_inv = 0
105 for theta11 in range(res_ang_min, res_ang_max, deltaangulo):
106     for theta12 in range(res_ang_min, res_ang_max, deltaangulo):
107         for theta13 in range(res_ang_min, res_ang_max, deltaangulo):
108             ##### {{ Cinematica Directa }} #####
109             Pnew = delta_kinematics_t1m_adams.forward(theta12, theta13, theta11)
110
111             # Punto a Guardar proximamente
112             px = Pnew[1] # [mm]
113             py = Pnew[2] # [mm]
114             pz = Pnew[3] # [mm]
115
116             ##### {{ Jacobiano Metodo A}} #####
117             [jinv, jxx,
118              theta31, theta32, theta33,
119              theta21, theta22, theta23,
120              Jtotal1] = jacobian_tm1_adams.jacobian_total(px, py, pz,
121              theta11, theta12, theta13)
122
123             # Determinantes para Singularidades
124             det_jinv = np.linalg.det(jinv)
125             det_jxx = np.linalg.det(jxx)
126
127             ##### {{ Restricciones angulos y Singularidades }} #####
128             theta2i_min = pd_tm1_adams.theta2i_min()
129             theta2i_max = pd_tm1_adams.theta2i_max()
130             theta3i_min = pd_tm1_adams.theta3i_min()
131             theta3i_max = pd_tm1_adams.theta3i_max()
132
133             ##### {{ Restricciones de angulo 2i y 3i para cada brazo }} #####
134             if (theta2i_min < theta21 < theta2i_max and
135                 theta2i_min < theta22 < theta2i_max and
136                 theta2i_min < theta23 < theta2i_max):
137                 if (theta3i_min < theta31 < theta3i_max and
138                     theta3i_min < theta32 < theta3i_max and
139                     theta3i_min < theta33 < theta3i_max):
140
141             ##### {{ Restricciones Singularidad Jacobiano }} #####
142             # Error singularidad

```

```

143     err_jxx = pd_tm1_adams.err_jxx()
144     err_jinv = pd_tm1_adams.err_jinv()
145
146     # ****
147     # ***** {{ Jx=0 y Jinv=0 }} *****
148     # ****
149     if (det_jxx > err_jxx or det_jxx < (-1 * err_jxx)) and (
150         det_jinv > err_jinv or det_jinv < (-1 * err_jinv)):
151
152         espTra[cont, 0] = px
153         espTra[cont, 1] = py
154         espTra[cont, 2] = pz
155         espTra[cont, 3] = theta11
156         espTra[cont, 4] = theta12
157         espTra[cont, 5] = theta13
158         cont = cont + 1
159
160     # ****
161     # ***** {{ Restriccion Cubo}} *****
162     # ****
163     if (((px < pd_tm1_adams.px_max_ws()) and
164             (px > pd_tm1_adams.px_min_ws()))
165         and ((py < pd_tm1_adams.py_max_ws()) and
166                 (py > pd_tm1_adams.py_min_ws()))
167         and ((pz < pd_tm1_adams.pz_max_ws()) and
168                 (pz > pd_tm1_adams.pz_min_ws()))):
169         espTra_real[cont, 0] = px
170         espTra_real[cont, 1] = py
171         espTra_real[cont, 2] = pz
172         espTra_real[cont, 3] = theta11
173         espTra_real[cont, 4] = theta12
174         espTra_real[cont, 5] = theta13
175
176     # ****
177     # ***** {{ espTra_singu }} *****
178     # ****
179     # Jx=0
180     if (not (det_jxx > err_jxx or
181             det_jxx < (-1 * err_jxx))):
182         espTra_singu[cont_singu, 0] = px
183         espTra_singu[cont_singu, 1] = py
184         espTra_singu[cont_singu, 2] = pz
185         espTra_singu[cont_singu, 3] = theta11
186         espTra_singu[cont_singu, 4] = theta12
187         espTra_singu[cont_singu, 5] = theta13
188         cont_singu = cont_singu + 1
189
190     # ****

```

```

191     # ***** {{ espTra_singu_inv }} ****
192     # ****
193     # Jinv=0
194     if (not (det_jinv > err_jinv or
195             det_jinv < (-1 * err_jinv))):
196         espTra_singu_inv[cont_singu_inv, 0] = px
197         espTra_singu_inv[cont_singu_inv, 1] = py
198         espTra_singu_inv[cont_singu_inv, 2] = pz
199         espTra_singu_inv[cont_singu_inv, 3] = theta11
200         espTra_singu_inv[cont_singu_inv, 4] = theta12
201         espTra_singu_inv[cont_singu_inv, 5] = theta13
202         cont_singu_inv = cont_singu_inv + 1
203
204 ##### {{ Guardar WS }} #####
205 # Guardar WS txt
206 f = open(mi_path, 'w')
207 for i in range(0, cont, 1):
208     f.write('\t'.join(str(v) for v in espTra[i, 0:6]) + "\n")
209 f.close()
210 return [espTra, cont - 1,
211          espTra_singu, cont_singu - 1,
212          espTra_singu_inv, cont_singu_inv - 1,
213          espTra_real, cont - 1]
214
215
216 ##### [ Callback ] #####
217 ##### [ Main Funtion ] #####
218
219 def callback(data):
220     global permiso
221     global step_global
222     global id_call
223
224     permiso = data.graficar_realtime
225     step_global = data.step
226     id_call = data.idcall
227
228
229 ##### [ FIN ] #####
230
231
232 if __name__ == "__main__":
233     try:
234         nodo()
235     except rospy.ROSInterruptException:
236         pass
237
238 ##### FIN #####

```


B.8.12. posicionador_rviz_realtime_tm1_adams.py

```
1  #!/usr/bin/env python
2  # Nombre Creador: Ivan Alejandro Fernandez Gracia
3  # Universidad: Universidad de Santiago de Chile
4  # Mail: ivan.fernandez.g@usach.cl
5  # Objetivo: Trayectorias en tiempo real Rviz
6
7  ##### [Importar Librerias] #####
8  ##### [Importar mensajes] #####
9  #####
10 import pd_tm1_adams
11 import codos_tm1_adams
12 import roslib
13 import rospy
14 import std_msgs.msg
15
16 #####
17 ##### [Parametros] #####
18 #####
19 from sensor_msgs.msg import JointState
20 from simu_visual.msg import angulo
21 from simu_visual.msg import posicionxyz
22 from simu_visual.msg import matriz_path_ls
23
24 #####
25 ##### [Funcion Principal] #####
26 #####
27 pi = pd_tm1_adams.pi()
28 dtr = pd_tm1_adams.dtr()
29 mmtm = pd_tm1_adams.mmtm()
30
31 #####
32 ##### [ Funcion Principal] #####
33 #####
34 #####
35 # /-----/
36 # /----- Nodo -----/
37 # /-----/
38 def nodo():
39     ##### {{ Callback Recibe datos }} #####
40     global permiso
41     global id_call
42     global x_m
43     global y_m
44     global z_m
45     global theta1_m
46     global theta2_m
```

```

47     global theta3_m
48     global t_m
49
50     juntas = JointState()
51     permiso = False
52     id_call = 0
53     id_permiso = 0
54
55     ##### {{ Inicio Nodo }} #####
56     rospy.init_node("posicionador_rviz_realtime_tm1", anonymous=False)
57     rate = rospy.Rate(7.8125) # Hz
58
59     ##### {{ Publisher }} #####
60     pub = rospy.Publisher("joint_states", JointState, queue_size=10)
61
62     while not rospy.is_shutdown():
63         ##### {{ Topics }} #####
64         rospy.Subscriber("m_txyzth123", matriz_path_ls, callback)
65
66         if permiso == True and id_call != id_permiso:
67             rospy.loginfo("Creando Trayectoria Linear RVIZ !")
68             largo = len(t_m)
69
70             # Punto inicial
71             juntas = angulos_eulerianos(1,
72                                         x_m[0], y_m[0], z_m[0],
73                                         theta1_m[0], theta2_m[0], theta3_m[0])
74             delta = 1
75             rospy.sleep(delta)
76             pub.publish(juntas)
77
78             for i in range(1, largo):
79                 juntas = angulos_eulerianos(t_m[i],
80                                             x_m[i], y_m[i], z_m[i],
81                                             theta1_m[i], theta2_m[i], theta3_m[i])
82                 delta = t_m[i] - t_m[i - 1]
83                 rospy.sleep(delta)
84                 pub.publish(juntas)
85
86             ##### {{ Reset variable mensaje entrante }} #####
87             permiso = False
88             id_permiso = id_call
89
90             rate.sleep()
91
92
93 ##### [ Otras Funciones ] #####
94

```

```

95 ######
96 # /-----/
97 # /----- Angulos Juntas -----/
98 # /-----/
99 def angulos_eulerianos(ti,
100                 xi, yi, zi,
101                 th1, th2, th3):
102     # Rviz angulos interiores en Radianes
103     joint = JointState()
104     punto = [-yi * mmtm, -xi * mmtm, -zi * mmtm]
105
106     c2 = codos_tm1_adams.punto_codo(th2)
107     p2 = codos_tm1_adams.punto_ee(punto, 2)
108     [a2_a, a2_b] = codos_tm1_adams.angulos_codo(c2, p2, 2)
109
110     c3 = codos_tm1_adams.punto_codo(th3)
111     c3 = codos_tm1_adams.rotacion120(c3)
112     p3 = codos_tm1_adams.punto_ee(punto, 3)
113     [a3_a, a3_b] = codos_tm1_adams.angulos_codo(c3, p3, 3)
114
115     c1 = codos_tm1_adams.punto_codo(th1)
116     c1 = codos_tm1_adams.rotacion120(c1)
117     c1 = codos_tm1_adams.rotacion120(c1)
118     p1 = codos_tm1_adams.punto_ee(punto, 1)
119     [a1_a, a1_b] = codos_tm1_adams.angulos_codo(c1, p1, 1)
120
121     # Datos para publicar en Rviz
122     joint.header = std_msgs.msg.Header()
123     joint.header.stamp = rospy.Time.now()
124
125     joint.name = ["base_brazo1", "base_brazo2", "base_brazo3",
126                   "codo1_a", "codo1_b",
127                   "codo2_a", "codo2_b",
128                   "codo3_a", "codo3_b",
129                   "act_x", "act_y", "act_z"]
130
131     joint.position = [th1 * dtr, th2 * dtr, th3 * dtr,
132                       th1 * dtr + a1_a, a1_b,
133                       th2 * dtr + a2_a, a2_b,
134                       th3 * dtr + a3_a, a3_b,
135                       xi / 1000, yi / 1000, zi / 1000]
136
137     joint.velocity = []
138     joint.effort = []
139
140     return joint
141
142

```

```

143 ##### [ Callback] #####
144 ##### [ Callback] #####
145 #####
146 # /-----/
147 # /----- Recibe datos matrices -----/
148 # /-----/
149 def callback(data):
150     global permiso
151     global id_call
152     global x_m
153     global y_m
154     global z_m
155     global theta1_m
156     global theta2_m
157     global theta3_m
158     global t_m
159
160     permiso = data.permiso
161     id_call = data.id_call
162     x_m = data.x
163     y_m = data.y
164     z_m = data.z
165     theta1_m = data.th1
166     theta2_m = data.th2
167     theta3_m = data.th3
168     t_m = data.tiempo
169
170
171 ##### [ Main Funtion] #####
172 ##### [ Main Funtion] #####
173 #####
174 if __name__ == "__main__":
175     try:
176         nodo()
177     except rospy.ROSInterruptException:
178         pass
179
180 ##### FIN #####

```

B.8.13. codos_tm1_adams.py

```
1 # Nombre Creador: Ivan Alejandro Fernandez Gracia
2 # Universidad: Universidad de Santiago de Chile
3 # Mail: ivan.fernandez.g@usach.cl
4 # Objetivo:
5 # Fuentes: Calcular angulos interiores de cada cadena cinematica
6 # para simular Rviz uan trayectoria
7
8 ##### [Importar Librerias] #####
9 #####
10 #####
11 import math
12 import pd_tm1_adams
13
14 #####
15 ##### [Parametros] #####
16 #####
17 e = pd_tm1_adams.e()
18 f = pd_tm1_adams.f()
19 rf = pd_tm1_adams.l1()
20 hf = pd_tm1_adams.hf()
21 he = pd_tm1_adams.he()
22
23 dtr = pd_tm1_adams.dtr()
24
25 # Puntos iniciales
26 A1 = [hf / 3, 0, 0]
27 A2 = [-A1[0] * math.cos(60 * dtr), -A1[0] * math.sin(60 * dtr), 0]
28 A3 = [A2[0], -A2[1], 0]
29
30
31 ##### [ punto_codo (J 1,2,3) ] #####
32 #####
33 #####
34 # /-----/
35 # /----- punto_codo -----/
36 # /-----/
37 # Plano XZ
38 def punto_codo(theta):
39     theta *= dtr
40     b1 = [A1[0] + rf * math.cos(theta), 0.0, rf * math.sin(theta)]
41     return b1
42
43
44 # /-----/
45 # /----- rotacion120 -----/
46 # /-----/
```

```

47     sin120 = math.sin(120 * dtr)
48     cos120 = math.cos(120 * dtr)
49
50
51     def rotacion120(ent):
52         sal = [0.0, 0.0, 0.0]
53         sal[0] = cos120 * ent[0] + sin120 * ent[1]
54         sal[1] = -sin120 * ent[0] + cos120 * ent[1]
55         sal[2] = ent[2]
56
57         return sal
58
59     # /-----/
60     # /----- rotacion240 -----/
61     # /-----/
62     sin240 = math.sin(240 * dtr)
63     cos240 = math.cos(240 * dtr)
64
65
66     def rotacion240(ent):
67         sal = [0.0, 0.0, 0.0]
68         sal[0] = cos240 * ent[0] + sin240 * ent[1]
69         sal[1] = -sin240 * ent[0] + cos240 * ent[1]
70         sal[2] = ent[2]
71
72         return sal
73
74 ##### [ punto_ee EE1,2,3 ] #####
75 ##### [ punto_ee EE1,2,3 ] #####
76 ##### [ punto_ee EE1,2,3 ] #####
77     # /-----/
78     # /----- punto_ee -----/
79     # /-----/
80     # Distancia de Punto centrar efector al punto EE y girar por motor
81
82     vhe2 = [he / 3, 0.0, 0.0]
83     vhe3 = rotacion120(vhe2)
84     vhe1 = rotacion120(vhe3)
85
86     def punto_ee(ee, brazo):
87         sal = [0.0, 0.0, 0.0]
88         if brazo == 2:
89             sal = sumav(ee, vhe2)
90         elif brazo == 3:
91             sal = sumav(ee, vhe3)
92         elif brazo == 1:
93             sal = sumav(ee, vhe1)
94
95         return sal

```

```

95
96 # /-----/
97 # /----- sumav -----/
98 # /-----/
99 def sumav(v1, v2):
100     s = [0.0, 0.0, 0.0]
101     s[0] = v1[0] + v2[0]
102     s[1] = v1[1] + v2[1]
103     s[2] = v1[2] + v2[2]
104     return s
105
106
107 ##### [ angulos_codo ] #####
108 #####
109 #####
110 # /-----/
111 # /----- angulos_codo -----/
112 # /-----/
113 def angulos_codo(codo, ee, brazo):
114     if brazo == 3:
115         codo = rotacion240(codo)
116         ee = rotacion240(ee)
117     elif brazo == 1:
118         codo = rotacion120(codo)
119         ee = rotacion120(ee)
120
121     if (codo[0] - ee[0]) != 0:
122         ang_a = math.atan((ee[2] - codo[2]) / (codo[0] - ee[0]))
123         if ((codo[0] - ee[0]) < 0):
124             ang_a = ang_a + (180 * dr)
125     else:
126         print("entra")
127         ang_a = 1.570796326794897
128
129     # prep
130     codo = rotacion_y(codo, ang_a)
131     ee = rotacion_y(ee, ang_a)
132
133     # calc
134     if (codo[0] - ee[0]) != 0:
135         ang_b = math.atan((ee[1] - 0) / (codo[0] - ee[0]))
136         if ((codo[0] - ee[0]) < 0):
137             ang_a = ang_a + (180 * dr)
138     else:
139         ang_b = 0
140     return [ang_a, ang_b]
141
142

```

```
143 # /-----/
144 # /----- rotacion_y -----/
145 # /-----/
146 def rotacion_y(ent, ang):
147     sal = [0.0, 0.0, 0.0]
148     sal[0] = ent[0] * math.cos(ang) - ent[2] * math.sin(ang)
149     sal[1] = ent[1]
150     sal[2] = -ent[0] * math.sin(ang) + ent[2] * math.cos(ang)
151     return sal
```