UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema de Visión Aplicado al Mejoramiento en el Seguimiento de Trayectorias en un Robot Delta Plano 3RRR

T E S I S

Que para obtener el título de

INGENIERO MECATRÓNICO

P R E S E N T A

CRISTHIAN GÓMEZ GONZÁLEZ

DIRECTOR DE TESIS:

DR. VÍCTOR JAVIER GONZÁLEZ VILLELA

MEXICO, D.F. **NOVIEMBRE 2012**

Dedicatorias

A mi madre Alicia.

Por creer en mí, por su confianza, por haberme apoyado incondicionalmente en todo momento, pero sobre todo, por quererme mucho, este logro también es tuyo.

A mi padre Esteban

Por haberme infundado los valores que lo caracterizan, perseverancia, constancia y responsabilidad, por sus consejos y motivaciones, por su apoyo y por su amor.

Todo este trabajo ha sido posible gracias a ellos, porque han sido la fuente de inspiración para lograr esta meta.

A mi hermano Jaime y mi hermana Marlet

Por estar siempre conmigo, por haberme impulsado a seguir adelante, por esos momentos padres que hemos pasado y por enseñarme a que debemos estar siempre unidos, los quiero mucho.

A mis sobrinos para que vean en mí un ejemplo a seguir.

A mis amigos, Pedro Enrique, Roberto, Pedro Mariano, Alejandro, Luz, Santiago, Arturo, Shael, Eulises, Ronald, Irving, Emmanuel, que me brindaron su ayuda, sus consejos, su atención y lo más importante su amistad.

Agradecimientos

Al Dr. Víctor Javier González Villela, por su asesoramiento, por su paciencia y sobre todo por su apoyo en mi proceso de titulación.

A la Universidad Nacional Autónoma de México por permitir mi formación como profesional, la Máxima casa de estudios.

A los profesores por compartirme sus conocimientos, sus enseñanzas y su dedicación.

Agradezco en lo que le corresponde a la DGAPA, UNAM, por el apoyo brindado para la realización de este trabajo, a través de los proyectos PAPIIT IN115811: "Investigación y desarrollo en sistemas mecatrónicos: robótica móvil, robótica paralela, robótica híbrida y teleoperación" durante la realización de este trabajo.

Índice

Capítulo 4

Lista de Figuras

Capítulo 1

Introducción

Durante la historia de la robótica, investigadores han desarrollado una gran variedad de robots, principalmente utilizados en cuestiones de tareas peligrosas, monótonas e imposibles para el ser humano. Por otra parte, la robótica se ha encargado de optimizar los sistemas robóticos a través del tiempo, buscando diferentes métodos para resolver los problemas. Una de las soluciones para mejorar la eficiencia y guiado de los sistemas robóticos, ha sido la implementación de sistemas de visión, siendo uno de los temas de interés de los investigadores por más de dos décadas.

Un sistema robótico sin la función de visión puede operar correctamente aunque con ciertos inconvenientes. Para ello, necesitan de un conocimiento total de su espacio de trabajo o de los objetos a manipular. Si alguna de estas características varía, el robot no es capaz de trabajar correctamente y es necesario la intervención del hombre para corregir los errores de posición o fallas de operación. Por ejemplo, algunas aplicaciones en robots industriales para solucionar dicho problema, realizan una serie de secuencias de posición pre-enseñadas a partir de la geometría de la zona de trabajo y la localización de los objetos, esto comúnmente se lleva a cabo en robots ensambladores o en funciones de "pick-and-place" [1].

En cambio, la visión en sistemas robóticos, les aporta una mayor flexibilidad operacional, proporcionando una respuesta automatizada sobre objetos y ambientes variables, logrando así trabajos más eficientes e inteligentes. Con esto, ciertas aplicaciones como trasporte de objetos (en procesos de manufactura y automatización), ensamblado a precisión o robots en la medicina, entre otros, requieren de robots con visión debido a la importancia de las tareas que realizan.

El tema de investigación sobre el estudio de visión en robots, comúnmente se le conoce en inglés como "Visual Servoing" o "Vision-Based Robot Control". Este método se refiere principalmente a la utilización de información extraída del sistema de visión para el control de movimiento del robot [2].

En términos generales, "Visual Servoing" se ha clasificado en "Image Based" (IBVS) propuesto por Weiss y Sanderson, "Position Based" (PBVS) y una tercera que es la

combinación de ambos “Hybrid Approach”. La diferencia entre IBVS y PBVS, es que el primero, utiliza el control de posición en base al error de la imagen en el plano y el segundo se basa en técnicas de modelado sobre la imagen capturada, debido a que se estima la posición del objeto de interés con respecto a la posición de la cámara en un ambiente en 3D, siendo así un control del robot en el espacio [3].

Esta técnica se ha aplicado tanto en robots seriales como en robots paralelos. Para ello, existen algunas formas de montar la cámara sobre el robot, el primero se le llama “eye-at-hand” el cual, se fija la cámara de modo que capture al robot y su espacio de trabajo **Figura 1.1 (b, d)**, el segundo llamado “eye-in-hand”, el cual se monta la cámara sobre el efector final de modo que la cámara se mueve junto con él **Figura 1.1 (a, c)** y por último una combinación de ambos **Figura 1.1 (e)** [1].

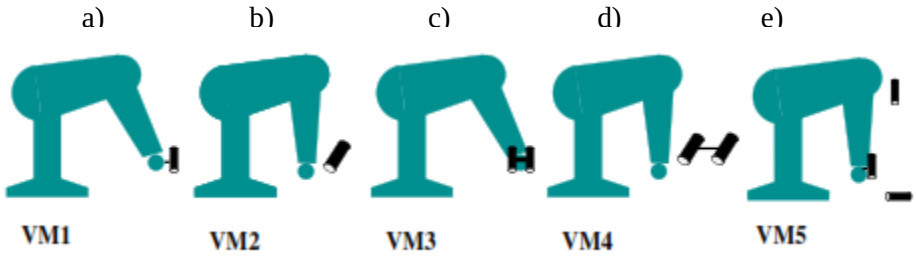


Figura 1.1: *Ejemplos de estrategias para montar la cámara sobre el robot.*

En cuanto a los problemas sobre la visión en robots manipuladores, se tienen por ejemplo, ciertas investigaciones que se enfocan en el problema de la orientación y posición del robot mediante visión, otros en el guiado de sistemas robóticos utilizando este método o en el mejoramiento de algoritmos más versátiles para el procesamiento de imágenes.

Algunas investigaciones han tratado de resolver dichos problemas con diferentes teorías, obteniendo buenos resultados. Aunque, continúa su amplia investigación ya que se desea tener robots más eficientes.

En el tema de los manipuladores con visión, se tienen investigaciones desarrolladas en robots seriales por ejemplo, Eve Coste-Maniere et ál. (1993) [4] donde utilizó un robot SCARA de seis grados de libertad para el seguimiento de contornos de objetos desconocidos, basándose en el método “Visual Servoing” **Figura 1.2**. La cámara fue montada en el modo “eye-in-hand” para capturar la superficie del objeto y seguir los contornos en tiempo real, con la captura de características relevantes (en este caso

puntos) procesaban la información para controlar la posición del efector final y así corregir la orientación y traslación de la cámara.

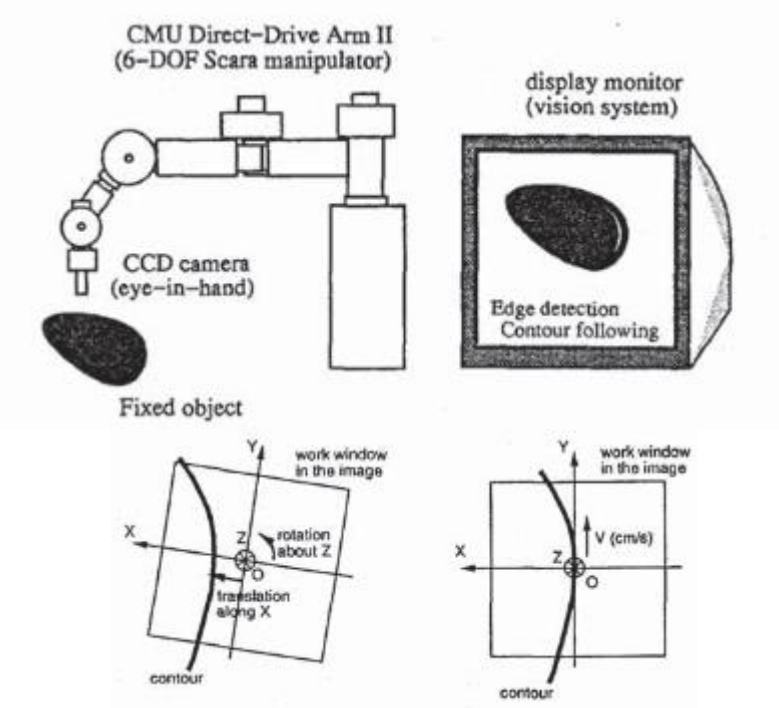


Figura 1.2: *Robot SCARA de 6 GDL con configuración "eye-in-hand", aplicado al seguimiento de contornos vía visión.*

La aplicación de este robot tendría tareas de inspección de superficies, en telerobótica, detección de defectos de contornos, soldadura de superficies uniformes o seguimiento de movimientos guiados. Los resultados fueron aceptables aunque el sistema de visión no detectaba automáticamente los objetos, se tenía que especificar el área y colocar objetos obscuros para un buen contraste de los contornos.

Por lo que, otra investigación desarrollada en la Universidad de Alicate para el seguimiento de trayectorias fue realizada por G.J. García et ál. [5], creando un método sobre el seguimiento intemporal de trayectorias basado en el control visual dinámico. En este caso utilizaron un robot serial de 2 GDL **Figura1.3**, al igual que en [4] utilizaron la configuración "eye-in-hand".

El objetivo era evitar obstáculos fijos a partir de una trayectoria descrita. Para ello, desarrollaron un controlador visual dinámico, que toma en cuenta la dinámica del robot para obtener los pares de fuerza necesarios y posicionar al robot a partir de la información

visual. Le llaman seguimiento intemporal de trayectoria, al cambio de referencia deseada en cada instante de movimiento, con el objetivo de ir corrigiendo la trayectoria.

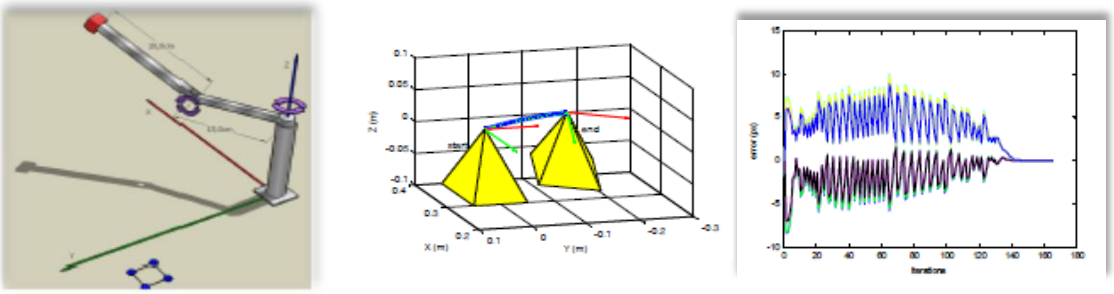


Figura 1.3: *Robot de 2 GDL para el seguimiento de trayectorias intemporal basado en el control visual dinámico.*

También se tienen los robots con movimiento en el plano en este tema, por ejemplo en el estudio hecho por E.C. Dean-León et ál (2004) [6], utilizó un robot plano de 2 GDL tomando en cuenta la configuración "eye-at-hand" **Figura 1.4**. A este robot le aplicaron un control de posición por medio de visión, al igual que [5] proponen un control dinámico basado en imagen.

Con la ayuda de métodos predictivos basados en la imagen, obtienen la velocidad y posición del robot. Algo importante de su sistema de control fue la utilización de dos computadoras, uno específicamente para el procesamiento de imágenes y otro para los cálculos del algoritmo de control.

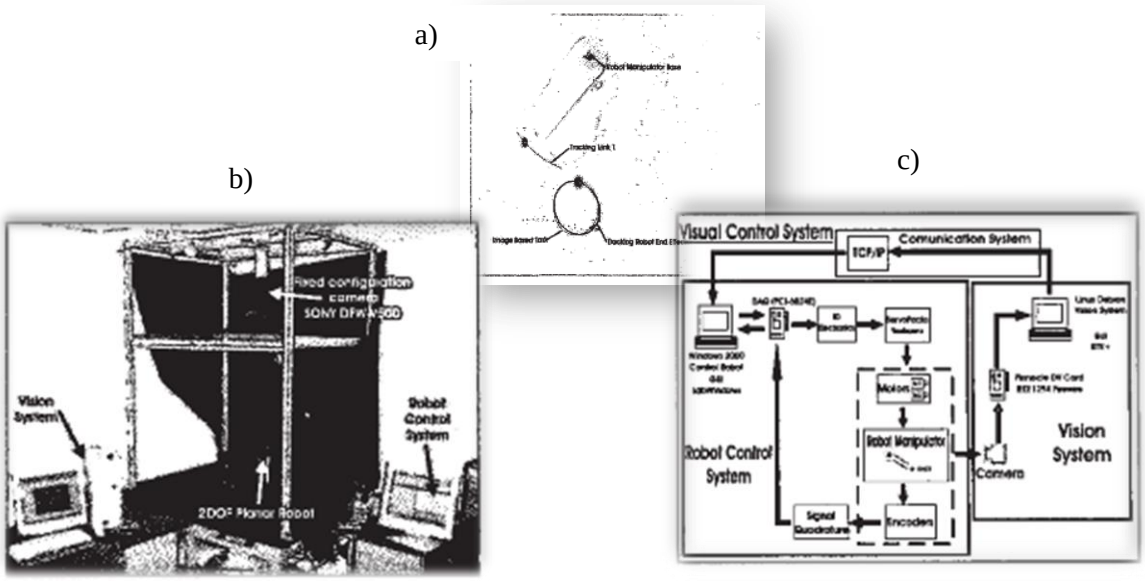


Figura 1.4: *a) Manipulador serial plano de 2 GDL describiendo una trayectoria, b) Lugar de trabajo del manipulador, c) Esquema general del sistema de control visual.*

Los robots paralelos por su parte, han sido de gran interés por investigadores debido a las ventajas que presentan en comparación con un robot serial. Por lo tanto, controlar la postura del efector final vía visión, es lo que también se busca en este tipo de robots.

Por ejemplo, en la investigación de M.A Trujano et ál (2010) [7] utiliza un robot paralelo plano del tipo RRR **Figura 1.5**, el objetivo de la investigación era mover el efector final del robot a una posición deseada, por medio de la captura de imágenes constante ("eye-at-hand"). Para ello, desarrollaron un algoritmo de control proporcional-derivativo logrando corregir la posición del efector final en base a la imagen, con esto obtenían los torques de las juntas activas reales.

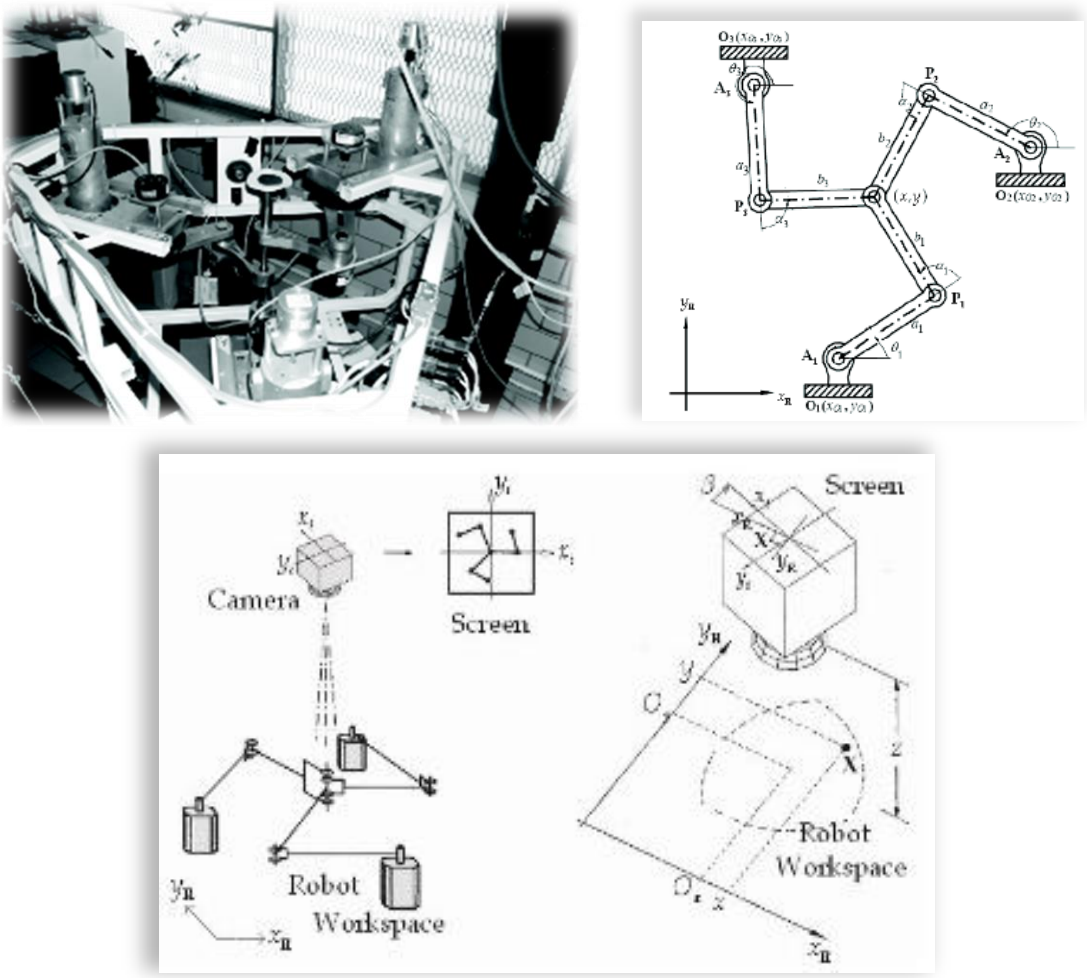


Figura 1.5: *Manipulador paralelo de 2 GDL, control vía visión para su estabilización y corrección.*

Por otra parte, en la Universidad Técnica de Brunswick en Alemania [17] están enfocados en la investigación de robots manipuladores, principalmente encaminado a aplicaciones de micro-ensamblado y manipulación a precisión. Por lo que, el Instituto de Herramientas de máquinas y tecnologías de fabricación (IWF sus siglas en alemán), están en constante desarrollo sobre métodos aplicados a robots paralelos de 3 a 6 GDL.

Algunos de los robots desarrollados, alcanzan a tener una repetitividad de menos de 1 µm de posicionamiento en el efector final. Para lograr tal precisión, hacen uso de sistemas de procesamiento de imágenes, esto les permite obtener la orientación y posición de varios objetos al mismo tiempo (por medio de un único sensor). Por consiguiente, la retroalimentación de los datos de visión en los sistemas de control del robot, garantizan una precisión de posicionamiento muy pequeño de menos de 1 µm. Como es el caso del robot para micro-ensamble de estructura plana trípode **Figura 1.6**.

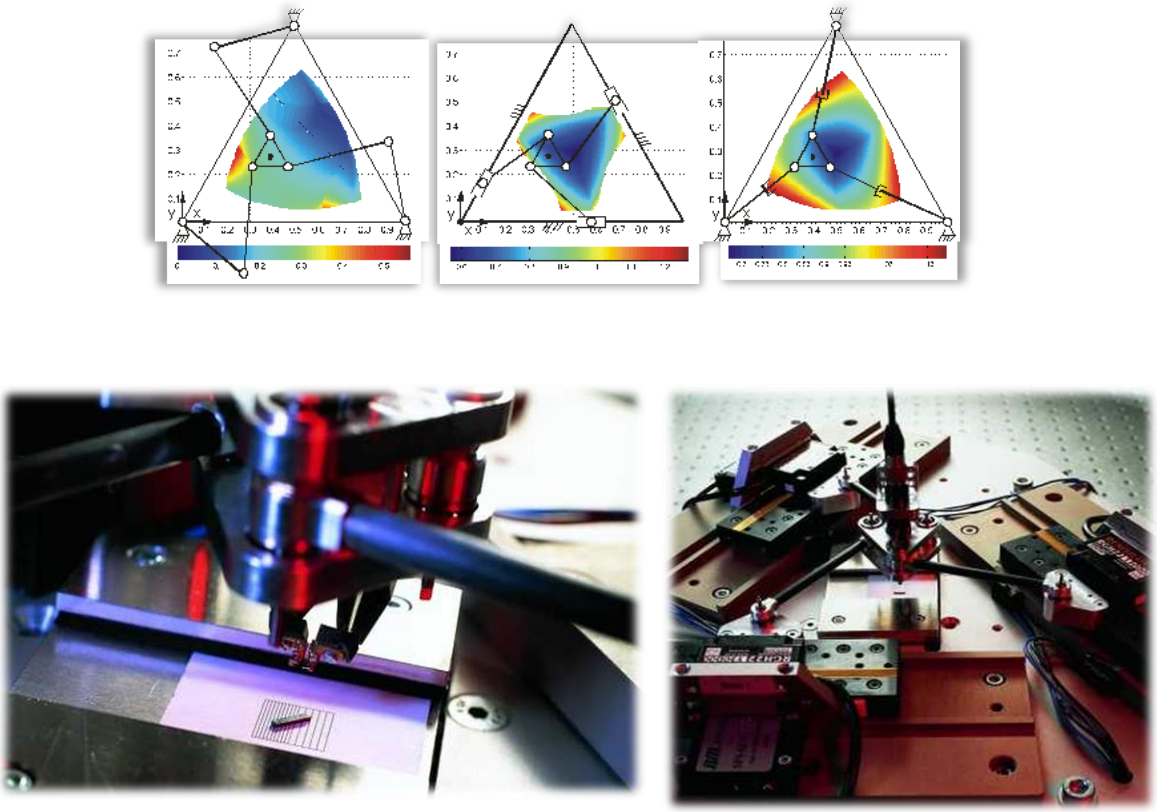


Figura 1.6: *Manipulador paralelo plano 3PRR desarrollado en la Universidad Técnica de Brunswick con aplicaciones de micro-ensamblado (repetitividad de 1µm).*

Dentro de las investigaciones, también se tienen a los robots paralelos con movimiento en el espacio como es el robot Gough-Stewart [8], que por medio de la información de visión controlan la posición de la plataforma móvil (sin sensores en sus juntas), otros autores muestran el control del robot paralelo I4R [9] utilizando la retroalimentación de visión.

También se tiene el desarrollo del robot delta de 3GDL con sistema de visión, por ejemplo A. Traslosheros et ál, ha publicado una variedad de investigaciones con este tipo de robot, tratando de mejorar su control mediante visión. Recientemente, en el 2011 [10] desarrollaron un nuevo algoritmo para el control de la postura del efector final, utilizaron estrategias especialmente diseñadas para el seguimiento y reconocimiento de objetos en el espacio en 3D.

Figura 1.7: *Manipulador paralelo inspirado en el robot delta, conocido como "robotenis" de 3 GDL. Desarrollado para el seguimiento de una pelota de ping-pong vía visión.*

Específicamente el robot reconoce una pelota de ping pong, de allí le atribuye el nombre de "Robotenis". Uno de los objetivos que tiene es el de cachar y golpear la pelota teniendo la cámara en el modo "eye-in-hand" **Figura 1.7**.

Estas investigaciones y otras, han obtenido buenos resultados, sólo que el tema de la visión en la robótica es muy amplio. Además, la integración de un sistema de visión en robots, ha dado buenos resultados, motivo por el cual aún continúa desarrollándose, esto se debe a que, actualmente existen varias aplicaciones donde los robos requieren de sistemas de visión. También, esto conlleva a la implementación en diferentes tipos de robots manipuladores y desarrollo de teorías más eficientes en cuanto al control y reconocimiento de objetos.

Aunado a esto, el surgimiento de los nuevos avances tecnológicos está directamente ligado con el desarrollo de estas investigaciones, ya que se requiere de dispositivos con niveles de procesamiento muy grandes que actualmente son la limitante de algunas investigaciones.

El presente trabajo llevado a cabo en el MRC (Mechatronic Research Center) de la Facultad de Ingeniería de la UNAM, pretende implementar y probar un sistema de visión por computadora basado en ReacTIVision (open source), fungiendo como sistema de sensado externo para la obtención de la postura del efector final, aplicando a un manipulador delta plano (MDP) del tipo 3RRR, de esta forma se pretende mejorar el rendimiento del robot en cuanto al seguimiento de trayectorias predefinidas en base a la retroalimentación de información.

La meta que se busca con el desarrollo de este trabajo, es el de implementar un sistema de visión sobre un manipulador paralelo plano y a su vez que sea capaz de mejorar el seguimiento de trayectorias predefinidas utilizando esta técnica, todo esto en un ambiente de simulación y experimentación. Por una parte, contribuye con resultados prácticos provenientes de los experimentos realizados e información relacionada a esta área de investigación.

Capítulo 2

Análisis cinemático de un manipulador delta plano 3RRR

Introducción

El capítulo describe un manipulador delta plano (MDP) 3RRR y su modelado cinemático directo, donde los brazos del MDP son modelados individualmente. Así mismo, se obtiene el modelado inverso de cada cadena cinemática que lo conforma, para posteriormente unirlo y obtener la posición y orientación del efector final, además incluye el análisis del espacio de trabajo del robot.

2.1 Descripción del manipulador delta plano 3RRR

El manipulador (paralelo) delta plano 3RRR se muestra geométricamente en la **Figura 2.1**. Está formado por tres pivotes fijos O_1, O_2 y O_3 los cuales están sobre una base fija, los pivotes B_1, B_2 y B_3 definen el efector final en movimiento o plataforma móvil, cada conjunto de pivotes están acomodados de tal manera que definen un triángulo equilátero.

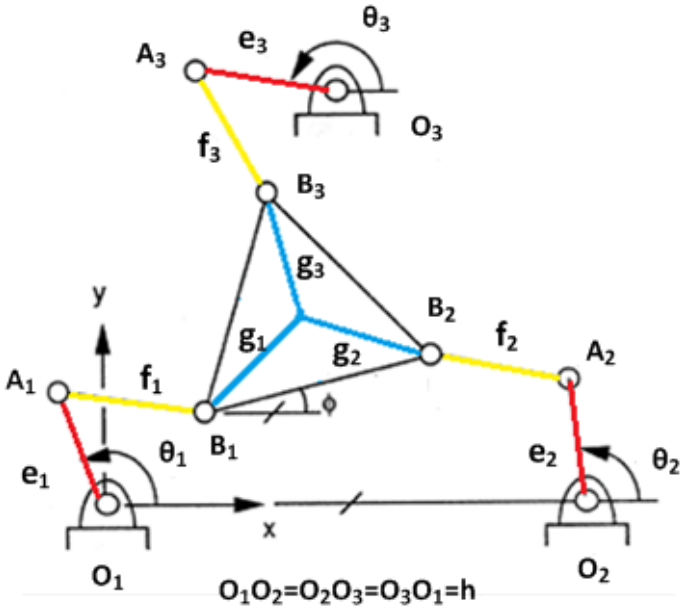


Figura 2.1: *Manipulador delta plano 3RRR.*

La plataforma móvil a su vez está conectado por tres cadenas cinemáticas que van de los puntos B_1, B_2 y B_3 hacia la base fija donde se localizan los puntos O_1, O_2 y O_3 respectivamente, conectados por juntas del tipo rotatorio, cada brazo tiene dos eslabones binarios (cada extremo tiene un nodo) unidos también por juntas rotatorias.

Para determinar los grados de libertad del robot, el cual se refieren al número de parámetros independientes que fijan la situación del órgano terminal [11], se puede hacer uso de la ecuación definida por Grübler –kutzbach ecuación **2.1**.

$$GDL = 3\,(L-1) - 2\,J_1 - 1\,J_2 \qquad (2.1)$$

Donde:

GDL : grados de libertad.

L : número total de eslabones, incluyendo la tierra.

J_1 : número de juntas completas.

J_2 : número de semijuntas.

En una junta completa (juntas con un grado de libertad) dos eslabones unidos pierden dos grados de libertad. Por ejemplo, en una junta rotacional o en una junta corredera.

En una semijunta (juntas con dos grados de libertad) dos eslabones unidos pierden un grado de libertad. Por ejemplo, una junta perno y ranura.

Para juntas múltiples (tres eslabones unidos o más) se cuenta el número de eslabones unidos a la junta y se resta uno, agregandolo a la categoría de "completas" (J_1).

$$J_1 = \#\,eslabones - 1$$

La configuración del mecanismo en conjunto está compuesto por ocho cuerpos y nueve juntas rotatorias (juntas completas) y ninguna semijunta (porque no hay ninguna junta que tenga dos grados de libertad), sustituyendo estos valores tenemos:

$$GDL = 3\,(8-1) - 2\,(9) - 1\,(0) = 3$$

El mecanismo tiene 3 grados de libertad, los cuales hacen referencia a la posición y orientación de la plataforma móvil [x, y, ϕ]. Por otra parte, el movimiento de la plataforma móvil, se considera como el eslabón de salida y los eslabones $O_1 A_1$, $O_2 A_2$ y $O_3 A_3$ como de entrada, dado a que la unidad motriz se encuentra conectada sobre los primeros eslabones.

2.2 Cinemática directa del manipulador plano de 3GDL

La cinemática directa nos permite conocer la posición del efector final en el espacio a partir de los valores de sus coordenadas articulares. Mediante el método de Denavit-Hartenberg se obtiene la cinemática directa de cada brazo del MDP (el análisis individual de los brazos del MDP permite tener expresiones menos extensas para la simulación). La cinemática directa principalmente se utilizó para poder simular el movimiento del MDP 3RRR en un software de simulación.

Si analizamos uno de los brazos del MDP 3RRR, el brazo se convierte en un manipulador plano de 3 GDL **Figura 2.2**. Viéndolo de esta forma, tiene tres juntas rotatorias ubicadas en los puntos O_i, A_i y B_i respectivamente **Figura 2.2**. Para su análisis, se le asigna a cada eslabón su sistema de referencia en base a las siguientes definiciones:

Primero enumeramos los cuerpos empezando de *0* a *n* y las juntas de *1* a *n* [12]. Por lo tanto, el sistema de coordenadas inercial está definido por $[x_0, y_0, z_0]$.

a) *El eje z_i se hace coincidir a lo largo del eje de la articulación $(i+1)$. La dirección de rotación o translación positiva puede ser seleccionada arbitrariamente.*

b) *El eje x_i es la perpendicular común a los ejes z_i y z_{i-1}.*

c) *El eje y_i completa el sistema de coordenadas dextrógiro según se requiera.*

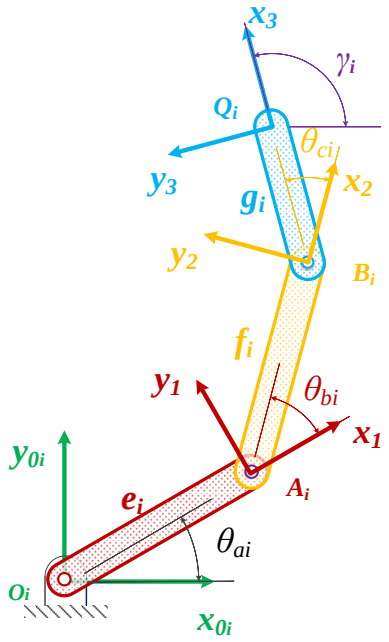


Figura 2.2: *Manipulador plano de 3gdl.*

Además, el método plantea que la configuración cinemática puede describirse definiendo cuatro magnitudes para cada articulación, representando las relaciones de traslación y rotación entre los enlaces adyacentes [11]. Estos parámetros determinan la estructura del elemento (a_i, α_i) y los parámetros de la articulación (d_i, θ_i) determinan la posición relativa de los elementos vecinos. Definidos de la siguiente forma:

a_i *Es la distancia de separación desde la intersección del eje z_{i-1} con el eje x_i hasta el origen del sistema i-ésimo medido a lo largo del eje x_i*

d_i *Es la distancia desde el origen del sistema de coordenadas $(i-1)$-ésimo hasta la intersección del eje z_{i-1} con el eje x_i medido a lo largo del eje z_{i-1}.*

α_i *Es el ángulo del eje z_{i-1} al eje z_i respecto del eje x_i usando la regla de la mano derecha.*

θ_i *Es el ángulo de la articulación de eje x_{i-1} al eje x_i respecto del eje z_{i-1} usando la regla de la mano derecha.*

Después, se crea la matriz de trasformación homogénea de 4x4, el cual integra los elementos de un sistema de coordenadas con respecto al sistema de coordenadas del elemento anterior, obteniendo una matriz compuesta por cuatro submatrices:

$$ {}^{A}_{B}T = \begin{bmatrix} {}^{A}_{B}R_{(3x3)} & {}^{A}q_{(3x1)} \\ \gamma_{(1x3)} & \rho_{(1x1)} \end{bmatrix} $$

Esta matriz expresa de una forma compacta la rotación *(submatriz 3x3)*, la traslación (vector columna 3x1), la transformación de perspectiva γ y el factor escalado ρ. Siendo el factor escalado una unidad y la matriz de trasformación de perspectiva *(1x3)* cero.

En base a las transformaciones básicas para el movimiento de los ejes coordenados [12], obtenemos la matriz de trasformación homogénea ${}^{i-1}A_i$, que relaciona los sistemas de coordenadas *i* con el sistema de coordenadas *i-1*, dado por:

$$ {}^{i-1}A_i = T(z,d)T(z,\theta)T(x,a)T(x,\alpha) \tag{2.2} $$

Sustituyendo, se obtiene:

$$ {}^{i-1}A_i = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3} $$

La ecuación **(2.3)** es llamada *matriz de transformación de Denavit-Hartenberg*. Ahora aplicando lo anteriormente establecido en cada uno de los brazos del MDP, se obtienen sus correspondientes parámetros. Dado a que todos los ejes de las juntas del manipulador plano son paralelos entre sí, se obtendrán valores de cero en los ángulos α_i y las distancias d_i de la traslación.

Los parámetros obtenidos en base a los sistemas de coordenadas establecidos en la **Figura 2.2**, da como resultado la tabla **2.1**.

Tabla 2.1: *Parámetros generales de Denavit-Hartenberg para el manipulador plano.*

i	α_i	a_i	d_i	θ_i
1	0	e_i	0	θ_{ai}
2	0	f_i	0	θ_{bi}
3	0	g_i	0	θ_{ci}

Sustituyendo los parámetros de la tabla **2.1,** en la matriz de transformación de D-H, de las tres articulaciones, se obtiene:

$$ {}^{0}A_1 = \begin{bmatrix} c\theta_{ai} & -s\theta_{ai} & 0 & e_i c\theta_{ai} \\ s\theta_{ai} & c\theta_{ai} & 0 & e_i s\theta_{ai} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}A_2 = \begin{bmatrix} c\theta_{bi} & -s\theta_{bi} & 0 & f_i c\theta_{bi} \\ s\theta_{bi} & c\theta_{bi} & 0 & f_i s\theta_{bi} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}A_3 = \begin{bmatrix} c\theta_{ci} & -s\theta_{ci} & 0 & g_i c\theta_{ci} \\ s\theta_{ci} & c\theta_{ci} & 0 & g_i s\theta_{ci} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Multiplicando las tres matrices de transformación, obtenemos la matriz de transformación que va del sistema de referencia {3} al sistema de referencia {0} inercial.

$$ {}^{0}A_3 = {}^{0}A_1 \ {}^{1}A_2 \ {}^{2}A_3 $$

Sustituyendo y desarrollando, se tiene:

$$ {}^{0}A_3 = \begin{bmatrix} c\theta_{aibici} & -s\theta_{aibici} & 0 & e_i c\theta_{ai} + f_i c\theta_{aibi} + g_i c\theta_{aibici} \\ s\theta_{aibici} & c\theta_{aibici} & 0 & e_i s\theta_{ai} + f_i s\theta_{aibi} + g_i s\theta_{aibici} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

El vector de posición del origen de Q expresado en el sistema de coordenadas del efector final está dado por ${}^{3}\mathrm{q}_{i} = [0,0,0,1]^{\mathrm{T}}$. Dejando el vector de posición de Q con respecto al sistema de coordenadas de la base es ${}^{0}\mathrm{q}_{i} = [\mathrm{x}_{q_i}, \mathrm{y}_{q_i}, \mathrm{z}_{q_i}, 1]^{\mathrm{T}}$. Posteriormente podemos relacionar ${}^{3}\mathrm{q}$ a ${}^{0}\mathrm{q}$ con la siguiente trasformación:

$$\begin{bmatrix} \mathrm{x}_{q_i} \\ \mathrm{y}_{q_i} \\ \mathrm{z}_{q_i} \\ 1 \end{bmatrix} = {}^{0}A_{3} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} e_i c\theta_{ai} + f_i c\theta_{aibi} + g_i c\theta_{aibici} \\ e_i s\theta_{ai} + f_i s\theta_{aibi} + g_i s\theta_{aibici} \\ 0 \\ 1 \end{bmatrix}$$

Por lo tanto, para conocer la posición del punto Q de cada brazo sólo es necesario el vector de posición (x, y) de la matriz anterior, el cual se obtiene:

$$\begin{aligned} \mathrm{x}_{qi} &= e_i c\theta_{ai} + f_i c\theta_{aibi} + g_i c\theta_{aibici} \\ \mathrm{y}_{qi} &= e_i s\theta_{ai} + f_i s\theta_{aibi} + g_i s\theta_{aibici} \end{aligned} \qquad (2.4)$$

La ecuación (2.4) representa la posición del punto Q para cualquier brazo del MDP dado a que el análisis es el mismo para cada brazo. Si se sustituye el subíndice *i=1, 2, 3* para hacer referencia al número de brazo se obtiene la siguiente matriz:

$$\begin{bmatrix} \mathrm{x}_{q_1} \\ \mathrm{y}_{q_1} \\ \mathrm{x}_{q_2} \\ \mathrm{y}_{q_2} \\ \mathrm{x}_{q_3} \\ \mathrm{y}_{q_3} \end{bmatrix} = \begin{bmatrix} e_1 c\theta_{a1} + f_1 c\theta_{a1b1} + g_1 c\theta_{a1b1c1} \\ e_1 s\theta_{a1} + f_1 s\theta_{a1b1} + g_1 s\theta_{a1b1c1} \\ e_2 c\theta_{a2} + f_2 c\theta_{a2b2} + g_2 c\theta_{a2b2c2} \\ e_2 s\theta_{a2} + f_2 s\theta_{a2b2} + g_2 s\theta_{a2b2c2} \\ e_3 c\theta_{a3} + f_3 c\theta_{a3b3} + g_3 c\theta_{a3b3c3} \\ e_3 s\theta_{a3} + f_3 s\theta_{a3b3} + g_3 s\theta_{a3b3c3} \end{bmatrix}$$

Cuando se usa este método para el manipulador paralelo plano son necesarias seis variables articulares del robot, para lo cual se emplea la cinemática inversa que se explica a continuación.

2.3 Análisis cinemático inverso del MDP

El modelo cinemático inverso nos permite obtener las variables articulares del robot en base a la posición y orientación deseada del efector final, al igual que en la cinemática directa, los brazos del MDP se resolverán individualmente y no en conjunto.

Esto último, se debe a que los métodos que existen para resolver la cinemática inversa del MDP como cadena cerrada, obtienen expresiones de las variables articulares con diversas funciones no lineales (senos y cosenos), el cual conseguirían alentar los cálculos de la cinemática inversa. Esto se debe a que algunos software resuelven estas funciones por medio de iteraciones, aumentando así el tiempo de cálculo.

Por tal motivo, se decidió resolver la cinemática inversa del MDP de esta forma, llegando a expresiones con menos términos y reduciendo el tiempo de cálculo, aunado a esto, se tiene la flexibilidad de resolver sólo las variables articulares de las juntas activas (los pivotes motorizados).

De los métodos existentes para resolver la cinemática inversa en un manipulador serial, los más comunes son; el método analítico o el método geométrico. El método analítico obtiene expresiones que resuelven ecuaciones no lineales trigonométricas, resulta en igualar la posición y orientación del efector final referido a la base.

La otra alternativa consiste en estudiar geométricamente el manipulador en cuestión para formular las ecuaciones, obteniendo un conjunto de subproblemas geométricos en el plano. Eligiendo este último, por la simplicidad que presenta el robot y sobre todo para tener un mejor entendimiento del análisis.

2.3.1 Cinemática inversa del MDP por el método geométrico

El modelo de la cadena cinemática se observa en la **Figura 2.3**, donde se puede formar un triángulo con los eslabones e_i , f_i y el vector $\overline{O_iB_i}$. Así mismo, las líneas punteadas representan la postura opuesta de los eslabones e_i y f_i, siendo ambas posturas, las dos posibles configuraciones que puede tener el manipulador para la localización de un mismo punto, generalmente conocido como codo arriba (CA) y codo abajo (CB)

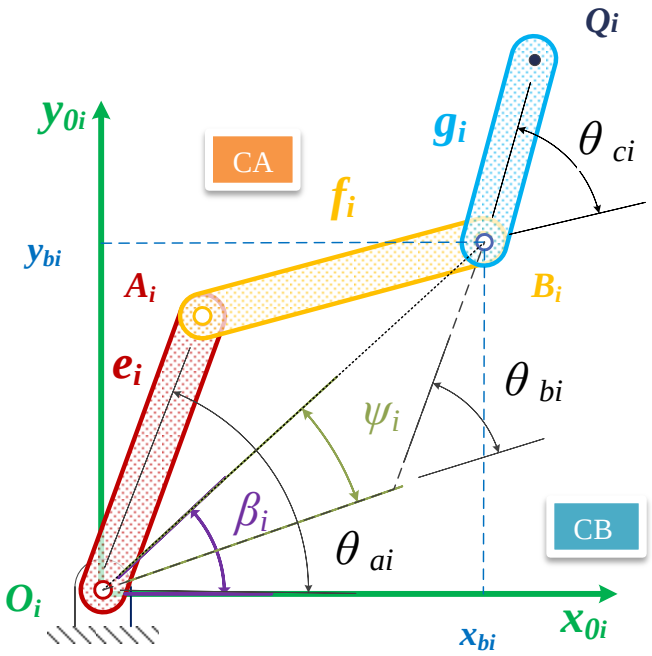


Figura 2.3: *Geometría de un manipulador plano de 3-gdl.*

Considerando el triángulo de la configuración codo arriba (CA) formado por los eslabones e_i , f_i , se aplica la "ley de los cosenos" obteniendo la siguiente expresión:

$$x_{bi}{}^2 + y_{bi}{}^2 = e_i{}^2 + f_i{}^2 - 2e_i\, f_i \cos(180 + \theta_{bi}) \tag{2.5}$$

De la ecuación **(2.5)** se despeja a θ_{bi}, siendo la segunda variable articular del manipulador:

$$\cos(180 + \theta_{bi}) = -\cos(\theta_{bi})$$

$$\theta_{bi} = \text{Acos}\left[\frac{x_{bi}{}^2 + y_{bi}{}^2 - e_i{}^2 - f_i{}^2}{2e_i f_i}\right] \tag{2.6}$$

Por otra parte, debe cumplirse la siguiente condición para que el triángulo exista:

$$\sqrt{x_{bi}{}^2 + y_{bi}{}^2} < e_i + f_i$$

Esta condición sirve para verificar si existe la solución al punto deseado. La condición no se satisface si el punto de destino está fuera del alcance del manipulador. Si existe una solución, esta ecuación se resuelve para aquel valor de θ_{bi} que se encuentre entre [0 a -180] grados en codo arriba y viceversa para codo abajo [0 a 180] grados.

Para obtener θ_{ai}, primero se encuentran las expresiones de los ángulos β_i y ψ_i. En el caso de β_i puede estar en cualquier cuadrante dependiendo de los signos de x_{bi} y y_{bi} . Por lo tanto se usa la función arco tangente de dos argumentos, quedando la ecuación **(2.7)**.

$$\beta_i = Atan2(y_{bi}, x_{bi}) \tag{2.7}$$

Del triángulo que forma ψ_i, se aplica de nuevo la ley de los cosenos, despejando el ángulo tenemos:

$$\psi_i = Acos\ \frac{{x_{bi}}^2 + {y_{bi}}^2 + {e_i}^2 - {f_i}^2}{2e_i\sqrt{{x_{bi}}^2 + {y_{bi}}^2}} \tag{2.8}$$

Estableciendo la condición para que el triángulo exista, el arco coseno debe resolverse en $0 \leq \psi_i \leq 180°$. Finalmente, con la suma de los ángulos tenemos:

$$\theta_{ai} = \beta_i \pm \psi_i \tag{2.8}$$

El signo positivo (+) si $\theta_{bi} < 0$ (CA) y el signo negativo (–) si $\theta_{bi} > 0$ (CB). Esta variable articular es la que requieren los actuadores.

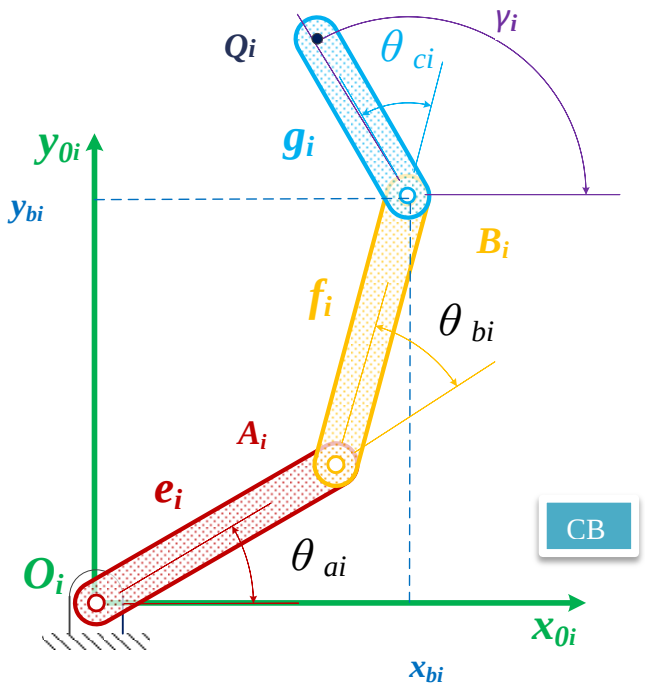


Figura 2.4: *La suma de las tres variables articulares del manipulador determinan la posición angular del último eslabón (γ) medido con respecto al sistema inercial.*

De la suma de los ángulos de las variables articulares **Figura 2.4** se obtiene θ_{ci}:

$$\theta_{ci} = \gamma - \theta_{ai} - \theta_{bi} \tag{2.9}$$

2.3.2 Integración de los tres brazos del MDP

Para Integrar los tres brazos del manipulador y que sigan el mismo punto, se hizo el siguiente análisis.

Primero definimos un sistema de coordenadas, que nos permita describir el desplazamiento de la plataforma móvil en su universo, como se muestra en la **Figura 2.5**.

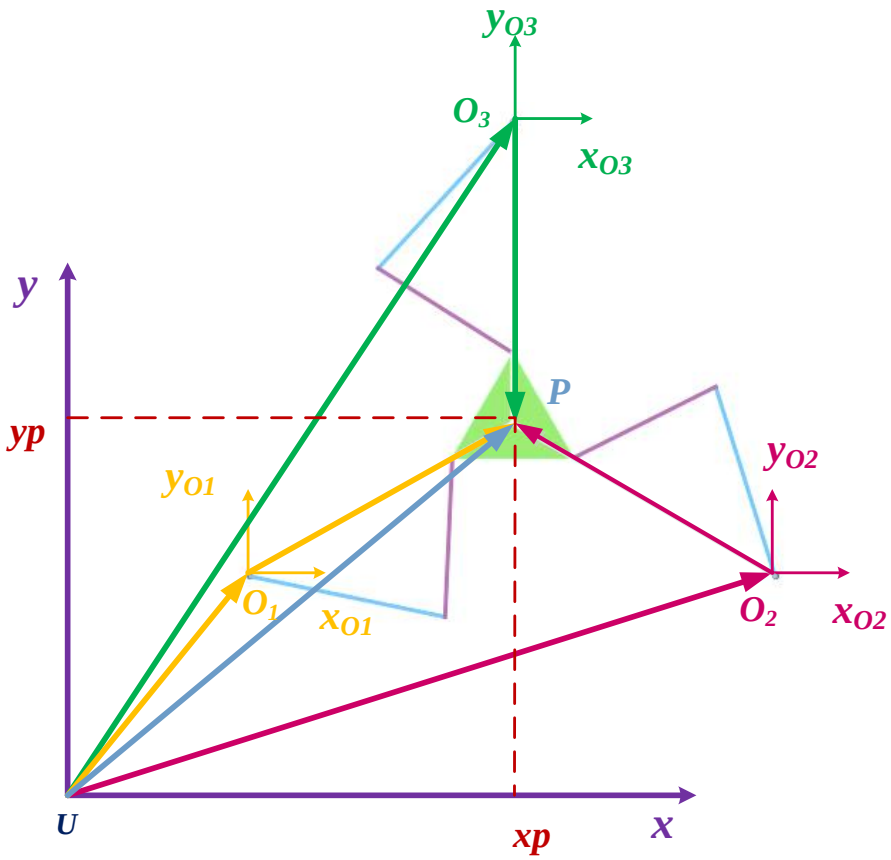


Figura 2.5: *Obtención de la posición de la plataforma móvil mediante la relación de vectores.*

En el sistema de referencia general establecemos el punto P. Este punto determina la posición del centro de la plataforma móvil en el sistema de referencia general y la unión de los brazos del manipulador.

Primero debemos encontrar las coordenadas del punto Q_i para encontrar las coordenadas del punto B_i (*i =1, 2, 3*) necesarias para la cinemática inversa de cada brazo, el cual las coordenadas del punto B_i incluyan los términos del sistema de referencia general.

Mediante la relación vectorial correspondiente a cada brazo, podemos obtener las coordenadas del punto Q_i. Basándonos en los vectores creados en la **Figura 2.3,** tenemos la siguiente relación vectorial:

$$\overline{UP} = \overline{UO_i} + \overline{O_iP}$$

Despejando el vector deseado $\overline{O_iP}$, el cual tiene las coordenadas de Q_i :

$$\overline{O_iP} = \overline{UO_i} - \overline{UP} \qquad (2.10)$$

Dónde:

$\overline{O_iP} = (x_{qi}, y_{qi})$

$\overline{UO_i} = (x_{Oi}, y_{Oi})$

$\overline{UP} = (x_p, y_p)$

Expresando la ecuación **(2.10)** en las coordenadas establecidas, tenemos:

$$x_{qi} = x_{Oi} - x_p \qquad (2.11)$$

$$y_{qi} = y_{Oi} - y_p$$

La ecuación correspondiente para cada brazo se obtiene sustituyendo el subíndice *i* = 1, 2, 3 en la ecuación **(2.11)**:

Brazo i=1

$$x_{q1} = x_{O1} - x_p$$
$$y_{q1} = y_{O1} - y_p$$

Brazo i= 2

$$x_{q2} = x_{O2} - x_p$$
$$y_{q2} = y_{O2} - y_p$$

Brazo i=3

$$x_{q3} = x_{O2} - x_p$$
$$y_{q3} = y_{O2} - y_p$$

Estas ecuaciones relacionan las coordenadas del sistema de referencia general al sistema de referencia inercial de cada brazo, en otras palabras, estas coordenadas permitirán que la cinemática inversa de cada brazo resuelva para el mismo punto *P,* pero con el origen en diferente posición.

Para resolver la cinemática inversa de los brazos, se necesitan las coordenadas de B_i $\{x_{bi}, y_{bi}\}$. Sin embargo, la cinemática inversa debe resolverse para el punto Q_i, que es el centro de la plataforma móvil.

Para ello, se debe restar la longitud del tercer eslabón y que la cinemática inversa contemple dicha magnitud. Resolviendo de forma vectorial, en la **Figura 2.3** se puede observar la relación del vector $\overline{O_iQ_i}$. Despejando el vector $\overline{O_iB_i}$, el cual contiene las coordenadas del punto B_i, obtenemos la ecuación (2.12):

$$\overline{O_iQ_i} = \overline{O_iB_i} + \overline{B_iQ_i}$$
$$\overline{O_iB_i} = \overline{O_iQ_i} - \overline{B_iQ_i} \qquad (2.12)$$

Ahora, representado la ecuación (2.12) en coordenadas cartesianas tenemos:

$$x_{bi} = x_{qi} - g_i \cos(\gamma_i)$$
$$y_{bi} = y_{qi} - g_i \sin(\gamma_i)$$

Sustituyendo la ecuación (2.11) de Q_i en (2.12), las coordenadas del punto B_i quedan de la siguiente forma:

$$x_{bi} = x_{0i} - x_p - g_i \cos(\gamma_i)$$
$$y_{bi} = y_{0i} - y_p - g_i \sin(\gamma_i)$$

Para definir la variable de orientación [ϕ] de la plataforma móvil, debemos fijar la posición angular del tercer eslabón g_i, para que en conjunto formen el triángulo equilátero.

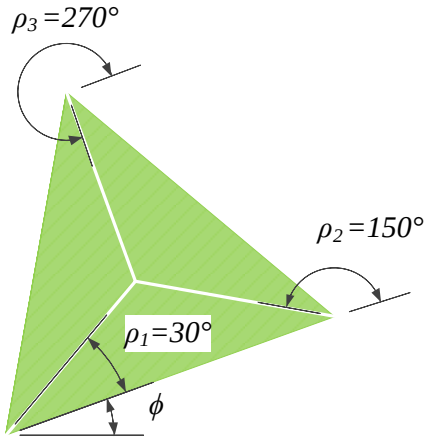


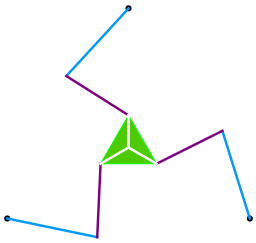
Figura 2.6: *Posición angular de los tres eslabones y ϕ como la orientación de la plataforma móvil.*

De acuerdo con la **Figura 2.6** se puede observar que: $\gamma_i = \rho_i + \phi$, por lo que, con esta igualdad la única variable a manipular es: (ϕ).

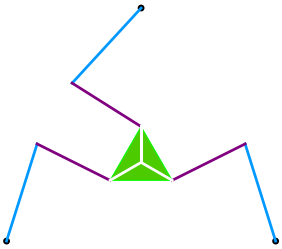
Dónde:
ρ_i: Posición angular fija del eslabón g_i para formar la plataforma móvil.
ϕ: Variable de orientación de la plataforma móvil.
γ_i: Posición angular del eslabón g_i medido con respecto al eje x_{0_i}

2.4 Posturas del MDP 3RRR

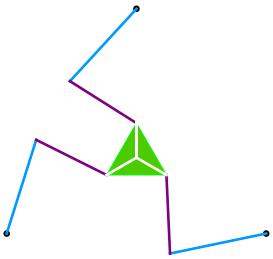
De la cinemática inversa, se concluye que existen dos configuraciones en cada brazo para la localización de un mismo punto. Haciendo combinaciones de los mismos, da un total de ocho posibles posturas en el MDP 3RRR, mostradas en las siguientes imágenes:



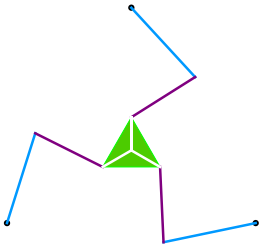
a) B-B-B



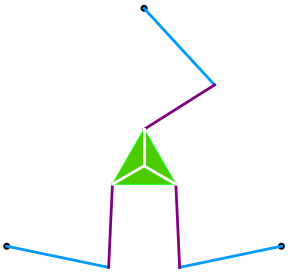
b) A-B-B



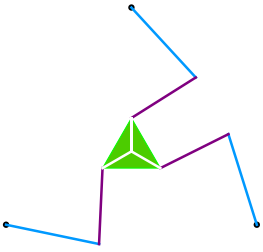
c) A-A-B



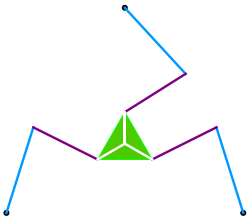
d) A-A-A



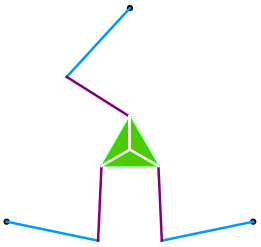
e) B-A-A



f) B-B-A



g) A-B-A



h) B-A-B

La configuración "B" se refiere al brazo en codo abajo y "*A*" a codo arriba. Basándonos en la investigación de [14], las posturas que presenta un mayor espacio de trabajo útil y las singularidades no interfieren demasiado al desplazarse sobre el espacio de trabajo, son las posturas *A-A-A* y *B-B-B*. De esta manera y junto con pruebas experimentales, se eligió la postura *B-B-B* tanto para las pruebas como para el análisis del espacio de trabajo.

2.5 Espacio de Trabajo del MDP 3RRR

El espacio de trabajo en un manipulador paralelo, se entiende como el conjunto de todas las posiciones y orientaciones que la plataforma móvil puede alcanzar. Una de las desventajas que tiene un manipulador paralelo en comparación con un manipulador serial, es su limitado espacio de trabajo. Sin embargo, la desventaja dependerá de la aplicación que se le dé al mecanismo, en nuestro caso sólo es importante conocer el área de trabajo con el que cuenta el modelo funcional y así evitar comportamientos indeseados durante las pruebas.

Si los brazos del MDP individualmente se extienden por completo, se logra observar el espacio máximo que puede alcanzar cada uno (generando un círculo). La intersección de las tres circunferencias dibujadas por los brazos, representa el área de trabajo ideal del manipulador, como lo muestra la **Figura 2.7**.

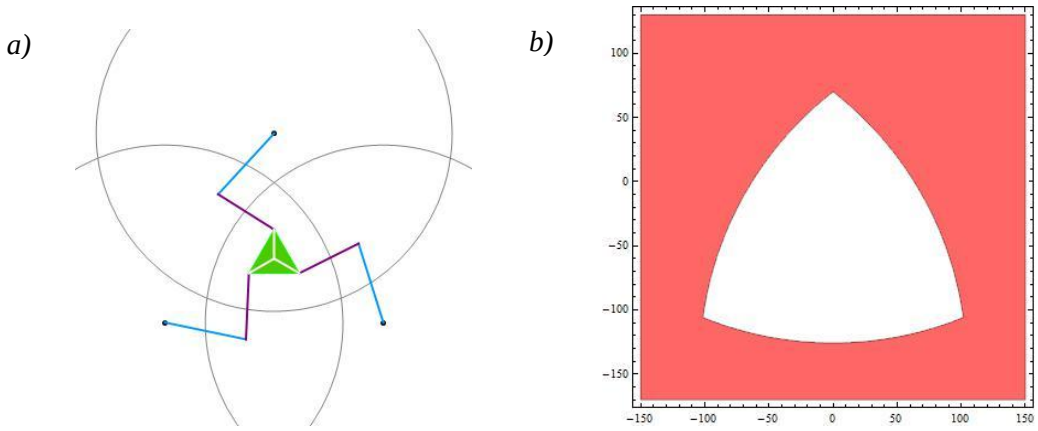


Figura 2.7: *a) Espacio de trabajo máximo ideal del MDP 3RRR, b) El triángulo ovalado color blanco, representa el espacio de trabajo ideal del modelo funcional.*

Ahora, si se toma en cuenta la orientación de la plataforma móvil, por ejemplo $\phi = 0$ y mantenga fija esa orientación, dibujando el área de trabajo, podemos observar que no hubo mucho cambio comparado con el área de trabajo ideal, mostrado en la **Figura 2.8**.

La **Figura 2.8** muestra el espacio de trabajo ideal, representado por la intersección de los círculos azules y el espacio de trabajo teniendo una orientación fija en la plataforma móvil $\phi = 0$, representado por la intersección de los círculos rojos.

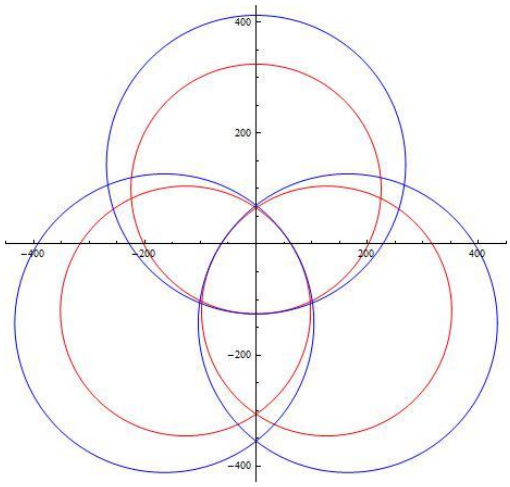
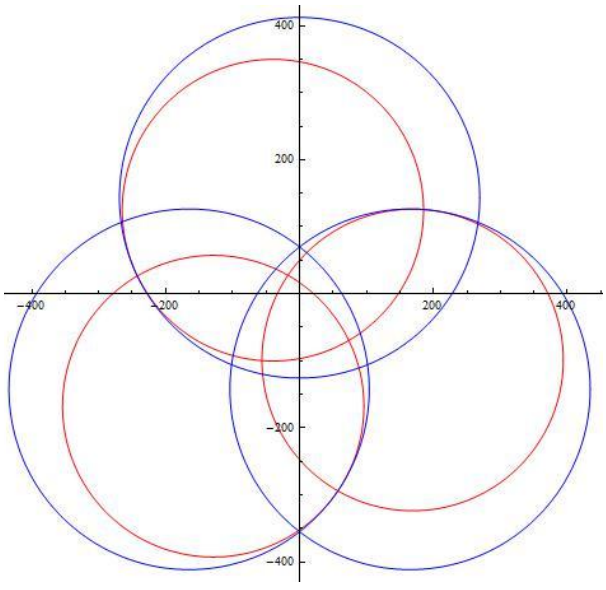


Figura 2.8: *Comparación del espacio de trabajo de la plataforma móvil ideal (círculos azules) y con una orientación fija ϕ=0 (círculos rojos).*

Por otra parte, si la plataforma móvil permanece con una orientación diferente a cero, el espacio de trabajo cambia. Tomando en cuenta, por ejemplo la orientación máxima y mínima que puede alcanzar el MDP estando en la postura del tipo *B-B-B,* podemos afirmar que efectivamente existe una disminución en el área de trabajo. Estas dos condiciones se pueden observar en la **Figura 2.9**.

a) *Orientación mínima ϕ=-65°*



b) Orientación máxima $\phi = 115°$

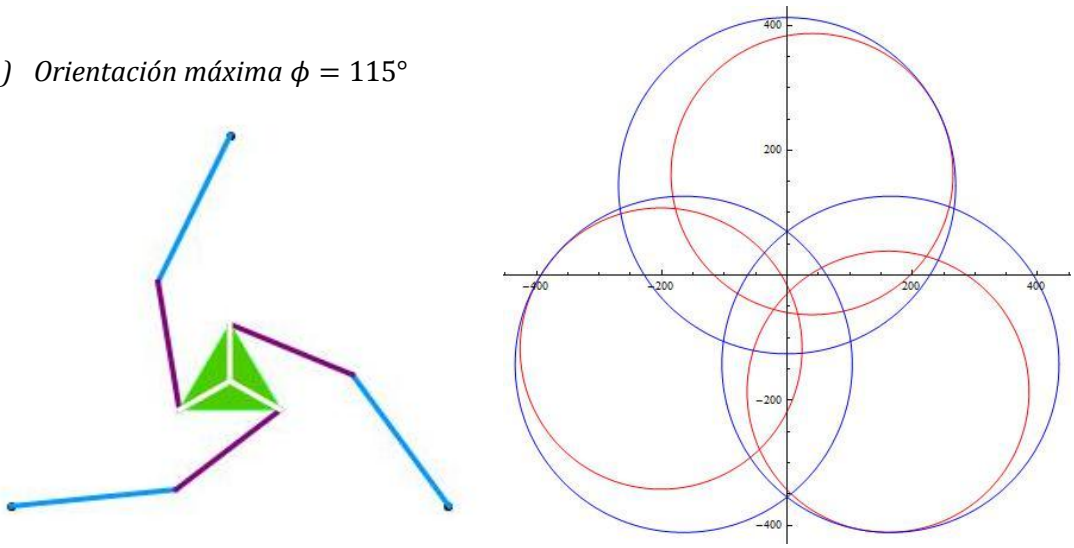


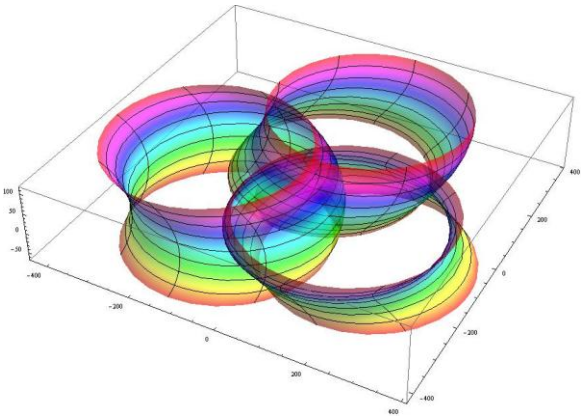
Figura 2.9: *Espacio de trabajo del MDP con orientación fija de ϕ = -65° y ϕ = 115°, representado por la intersección de los círculos rojos.*

El espacio de trabajo disminuye porque la plataforma móvil se mantiene en una orientación fija y los brazos se limitan entre sí por estar en cadena cerrada.

Los valores de orientación mínima y máxima se establecen del modelo funcional, ya que en estas posiciones impedimos que se colisionen los eslabones en el manipulador.

Para observar la evolución del espacio de trabajo variando la orientación de ϕ en su rango completo [-65° a 115°] se obtienen las figuras de **2.10**, donde muestran en forma de volumen el área de trabajo del MDP representado por intersección de los tres círculos que forman los brazos.

a) Vista en isométrico



b) Vista inferior

c) Vista superior

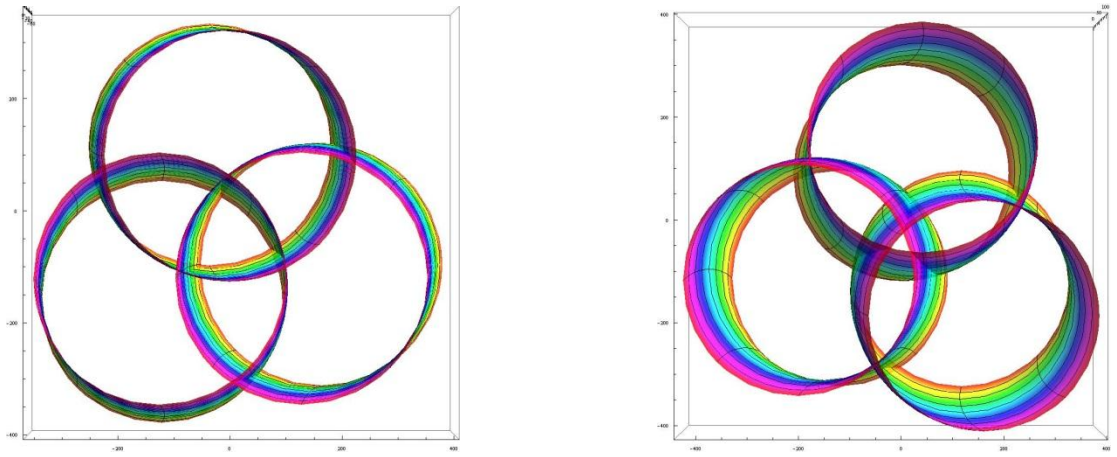


Figura 2.10: *Espacio de trabajo variando la orientación ϕ [-65° a 115°]*

La importancia de analizar el espacio de trabajo del MDP con orientación, es porque en las pruebas se incluye la orientación de la plataforma móvil (PM), interés por el cual se debe conocer el área de trabajo con el que cuenta nuestro robot para las pruebas. Sobre todo nos ayuda a observar el limitado espacio de trabajo que se genera si se utiliza la orientación de la PM, puesto que no muchos estudios lo mencionan referente a este manipulador.

2.6 Planeación de Trayectoria del MDP 3RRR

Una de las razones de asignarle al MDP un perfil de trayectoria, es para suavizar el movimiento sobre un camino predefinido, si lo vemos desde el punto de vista a una aplicación, serviría para trasportar cargas útiles con seguridad, realizar cortes a precisión o para el micro-ensamblado. Al incluir un perfil de trayectoria en el robot, nos permitirá probar el comportamiento del modelo funcional estando en estas condiciones.

Una manera de restringir los valores de las variables cinemáticas en ciertos instantes, es mediante el uso de polinomios, donde la variable independiente es el tiempo y la variable dependiente es la posición. El grado del polinomio dependerá del número de restricciones menos uno. Suponiendo que la plataforma móvil se mueva en línea recta donde se

conozca la posición inicial y final y además se quiera tanto la velocidad como la aceleración sean nulas al final del recorrido, nos da un total de seis restricciones, siendo así un polinomio de quinto grado para la posición.

El polinomio que corresponde a la posición, para un cuerpo que se desplace a lo largo de un eje *p* desde una posición p_i hasta una posición p_f en un intervalo de tiempo, con una duración de t_f, es:

$$p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

Suponiendo: $t_0 = 0$

Posición inicial: $p(t_0) = p_0$

En este caso, suponiendo lo anterior, las condiciones son:

$$p_0 = a_0$$

$$p_f = a_0 + a_1 t_f + a_2 {t_f}^2 + a_3 {t_f}^3 + a_4 {t_f}^4 + a_5 {t_f}^5$$

$$p'_0 = a_1$$

$$p'_f = a_1 + 2a_2 t_f + 3a_3 {t_f}^2 + 4a_4 {t_f}^3 + 5a_5 {t_f}^4$$

$$p''_0 = 2a_2$$

$$p''_f = 2a_2 + 6a_3 t_f + 12a_4 {t_f}^2 + 20a_5 {t_f}^3$$

Las condiciones representan un sistema lineal de seis ecuaciones con seis incógnitas, cuya solución es:

$$a_0 = p_0$$

$$a_1 = p'_0$$

$$a_2 = \frac{p''_0}{2}$$

$$a_3 = \frac{20p_f - 20p_0 - \left(8p'_f + 12p'_0\right) t_f - (3p''_0 - p''_f) {t_f}^2}{2{t_f}^3}$$

$$a_3 = \frac{30p_0 - 30p_f + \left(14p'_f + 16p'_0\right) t_f + \left(3p''_0 - 2p''_f\right) {t_f}^2}{2{t_f}^4}$$

$$a_3 = \frac{12p_f - 12p_0 - \left(6p'_f + 6p'_0\right) t_f - (p''_0 - p''_f) {t_f}^2}{2{t_f}^5}$$

Ahora tomando en cuenta las condiciones iniciales y finales deseadas:

$$t = 0 \qquad t = t_f$$
$$p_0 = 0 \qquad p_f = 1$$
$$p'_0 = 0 \qquad p'_f = 0$$
$$p''_0 = 0 \qquad p''_f = 0$$

Sustituyendo las condiciones deseadas en las ecuaciones obtenidas anteriormente, tenemos:

$$a_0 = 0 \qquad a_3 = \frac{10}{t_f{}^3}$$
$$a_1 = 0 \qquad a_3 = \frac{-15}{t_f{}^4}$$
$$a_2 = 0 \qquad a_3 = \frac{6}{t_f{}^5}$$

Finalmente, las variables se sustituyen en el polinomio quíntico de posición y en sus derivadas (velocidad y aceleración instantánea):

$$p(t) = 10\left(\frac{t}{t_f}\right)^3 - 15\left(\frac{t}{t_f}\right)^4 + 6\left(\frac{t}{t_f}\right)^5$$

$$p'(t) = 30\frac{t^2}{t_f{}^3} - 60\frac{t^3}{t_f{}^4} + 30\frac{t^4}{t_f{}^5}$$

$$p''(t) = 60\frac{t}{t_f{}^3} - 180\frac{t^2}{t_f{}^4} + 120\frac{t^3}{t_f{}^5}$$

Ejemplificando gráficamente la función de quinto grado, para un tiempo $t_f = 60$, la evolución de la posición, velocidad y aceleración con respecto al tiempo se muestra en la F**igura 2.11**.

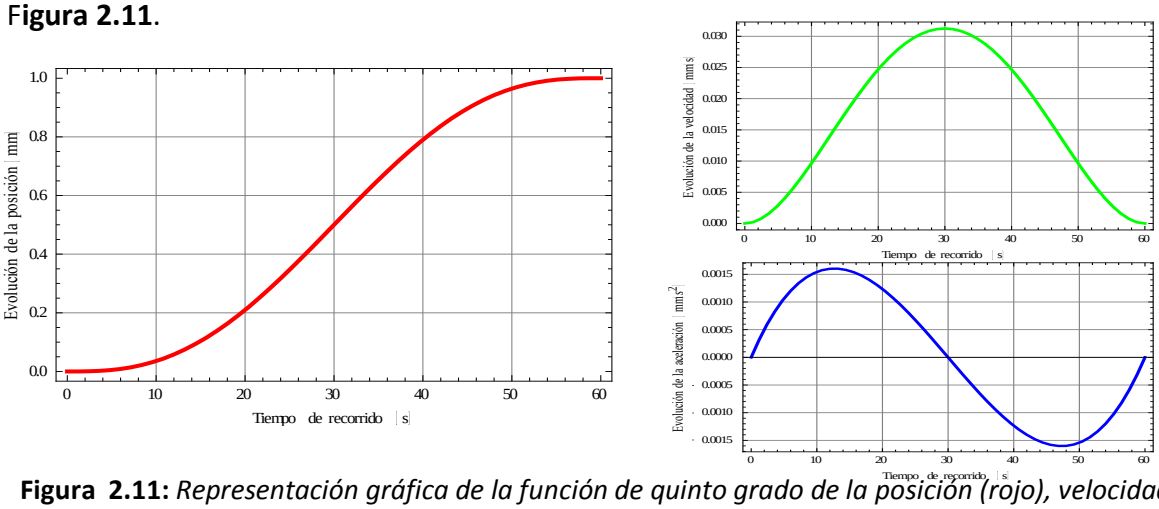


Figura 2.11: *Representación gráfica de la función de quinto grado de la posición (rojo), velocidad (verde) y aceleración (azul) de la trayectoria.*

2.7 Giro de la plataforma móvil.

Con la finalidad de probar la orientación del efector final, uno de los grados de libertad del MDP, se propone que durante el desplazamiento de la trayectoria predefinida, la PM al mismo tiempo esté girando continuamente. Para referirnos a este movimiento le llamaremos penduleo, justamente porque tiene un comportamiento de "va y viene" similar a un péndulo.

Una vez más, este movimiento se incluye con el objetivo de observar el comportamiento del MDP teniendo una tarea de este tipo sobre el modelo funcional. La plataforma móvil girará alrededor del eje de su centro, girando de una posición angular inicial a una final, como se muestra en la **Figura 2.12**.

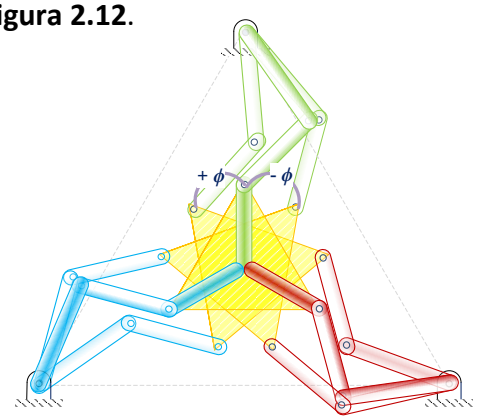


Figura 2.12: *Movimiento del penduleo de la PM.*

Para tener un movimiento oscilatorio, tomamos la función seno o coseno (dependiendo de la posición inicial que se desee). En la **Figura 2.13** se muestra gráficamente la función senoidal, donde la curva representa la evolución de la orientación de la plataforma móvil.

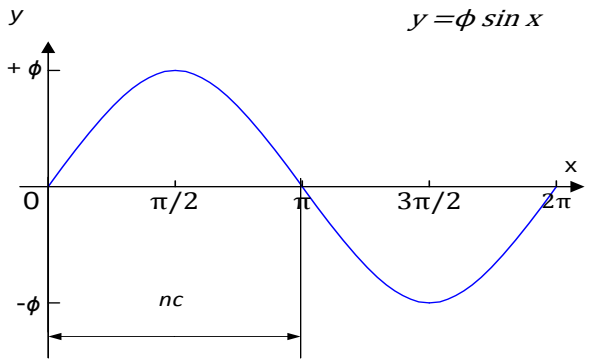


Figura 2.13: *Función senoidal para el movimiento del penduleo.*

Con la función senoidal obtenemos la ecuación para resolver el movimiento del penduleo, obteniendo la variable ϕ con respecto al tiempo, quedando de la siguiente forma:

$$\phi(t) = \phi\ sin(\pi * nc * t)$$

Dónde:

nc: *número de veces que ira de una posición angular inicial a la final la PM.*
φ: *Variable de orientación del PM.*
t: *Tiempo instantáneo en el recorrido.*

Durante su desplazamiento la PM puede realizar cierto número de péndulos, por ejemplo si la PM describe una línea recta en un determinado tiempo, este en su recorrido puede ir penduleando una, dos o *n* veces durante su trayecto.

Capítulo 3

Arquitectura del sistema

Introducción

Este capítulo describe las partes que conforman al sistema real y su implementación para las pruebas experimentales sobre el modelo funcional. Además, se detalla el sistema de visión, las herramientas de simulación y el sistema de control.

3.1 Esquema general del sistema

Ya que se tiene el análisis del manipulador delta plano (MDP) 3RRR, es necesario describir los elementos del sistema que lo conforman, puesto que fueron parte importante para las pruebas experimentales sobre el modelo funcional.

Las partes del sistema para el modelo funcional, se divide en el sistema de visión, el sistema de control y la comunicación entre el robot y la computadora. El esquema de la **Figura 3.1**, describe gráficamente la circulación de la información de forma general.

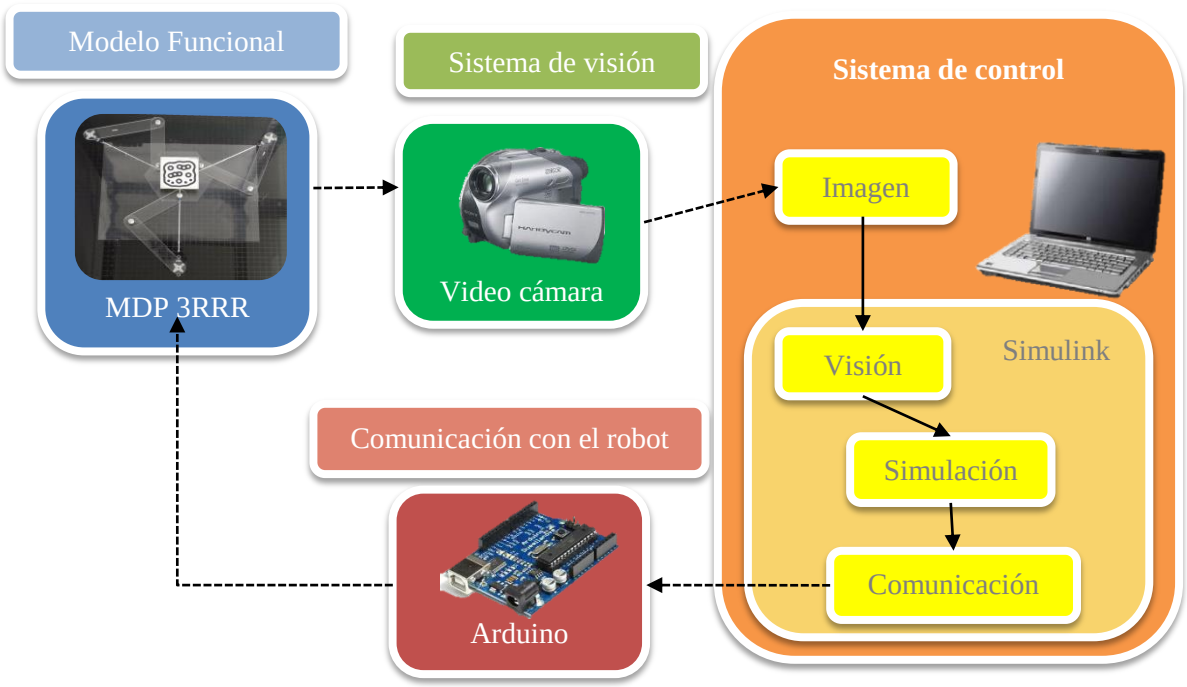


Figura 3.1: *Diagrama general del sistema implementado*

La cámara y el manipulador se montaron sobre una estructura hecha con perfil de estantería **Figura 3.2**, construido con la finalidad de fijar la cámara y aprovechar completamente el enfoque, capturando únicamente el espacio de trabajo del MDP.

El sistema cuenta con elementos indispensables que permiten la comunicación entre ellos, instrumentos que dan como resultado el control sobre el robot delta plano para el seguimiento de trayectorias predefinidas. El primer elemento se refiere al sistema de visión por computadora, que consta de un software para procesar la imagen capturada y posteriormente enviar los datos al sistema de control, donde es procesada esta información para obtener las posiciones angulares de las juntas activas del robot y finalmente ser enviadas a los servomotores por medio de comunicación serial, el resultado que se tiene es el desplazamiento de la plataforma móvil.

Figura 3.2: *Estructura de perfil de estantería para montar la cámara y el modelo funcional.*

Es necesario detallar los elementos del sistema para comprender el funcionamiento en su conjunto. Por lo que, a continuación se explican dichos elementos.

3.1.1 Modelo funcional

El modelo funcional se refiere al MDP 3RRR **Figura 3.3**, construido de material de acrílico. Los cortes de las piezas fueron mandados a hacer a láser, sus dimensiones pueden ser consultadas en el **Apéndice F**, estas dimensiones se tomaron a partir de otro modelo que servía para pruebas. El material se eligió con la finalidad de tener un modelo funcional rápido, económico y de fácil acceso, el cual lo cubría en ese momento el acrílico.

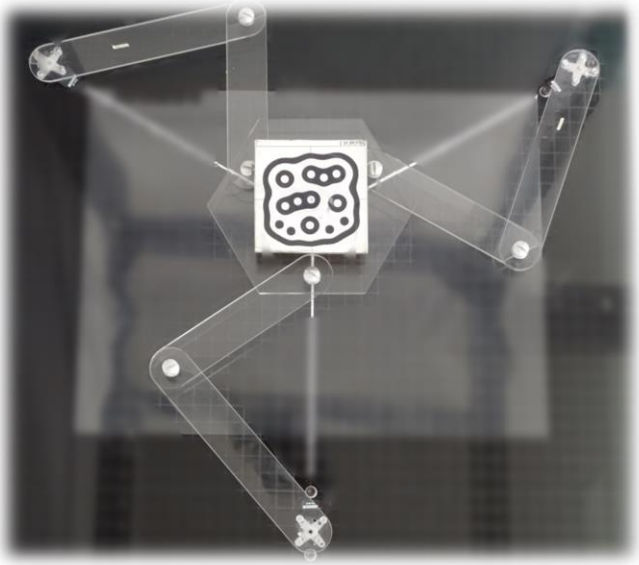


Figura 3.3: *Foto del manipulador delta plano real*

En la unión entre los eslabones, se utilizaron pequeños sujetadores de aluminio. Sobre la base se hicieron espacios para colocar los servomotores, la ubicación de éstos fue diseñada de tal manera que no perdiera simetría el robot.

3.1.2 Servomotor

La parte motriz del MDP, está formado por servomotores, los cuales se caracterizan por tener la capacidad de posicionarse de forma inmediata en cualquier posición dentro de su intervalo de operación. Están generalmente formados por un motor, un sistema reductor de engranes y un circuito de control **Figura 3.4**. Sobre el eje del motor del servo tiene conectado un potenciómetro, este le permite a la circuitería de control, supervisar el ángulo actual del servomotor.

Los servomotores utilizados son VS-3 [20], su voltaje de operación es de 4.8 V ~ 6.0 V de corriente directa, el ancho de pulso mínimo y máximo es de 800 a 2200 μsec respectivamente. En nuestro caso se controlaron mediante un microcontrolador (arduino) para posicionarlos dentro de su margen de operación [0° a ~180°]. En la **Tabla 3.1** se muestran sus datos técnicos.

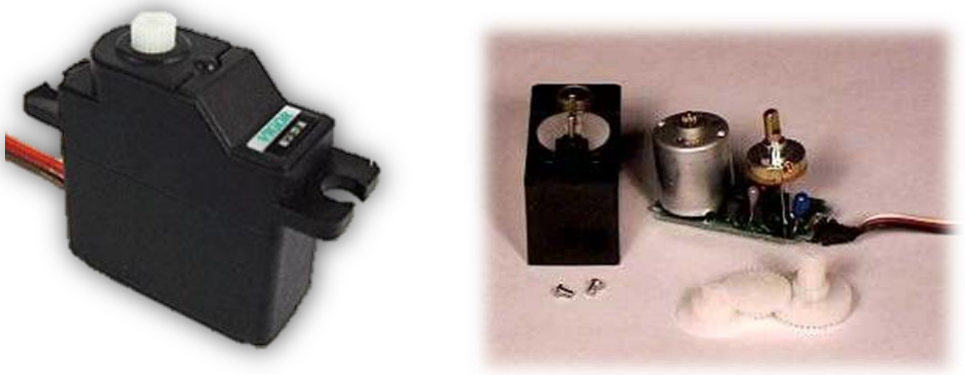


Figura 3.4: *Servomotor VS-3*

Tabla 3.1: *Especificaciones técnicas del servomotor VS-3*

Control System	Pulse width control 1500**µs** neutral	
Operating Voltage	4.8 V ~ 6.0 V (DC)	
STD Direction	Counter clockwise/pulse traveling 800 a 2200 **µs**	
Operating Speed	0.16 sec/60° no load	0.12sec/60° no load
Stall Torque	≥2.2 kgf.cm(30.55 oz/in)	≥2.5 kgf.cm(34.72 oz/in)
Runing Current	~0.2A	~0.25A
Output angle	≥ **170°**	

Este tipo de actuadores se utilizaron por su facilidad en cuanto a su funcionamiento, ofreciendo un alto torque y respondiendo rápido para posicionarse, además de esto son servos de bajo costo. Por otro lado, nos proporcionaron información de su implementación en un manipulador paralelo plano.

3.2 Sistema de Visión

El sistema de visión para su funcionamiento, requiere de una cámara y el software, este último contiene el algoritmo para el procesamiento de imágenes. De forma general, la cámara funge como sensor para observar al robot desde una vista aérea y mediante la localización de los identificadores dentro de su campo de visión (delimitado por la cámara), el software procesa la imagen para arrojarnos los datos necesarios (posición, orientación, velocidad, etc.) los cuales se traducen como datos de la plataforma móvil, ya que el marcador es montado sobre éste. Por último, estos datos son enviados al sistema de control, el encargado de utilizar dichos datos.

3.2.1 Cámara

La cámara que se utilizó fue una *Sony Handycam DCR-TRV361,* su conexión con la computadora es alámbrica a través del puerto firewire. El software de visión cuenta con soporte para Windows, por lo que puede aceptar cualquier cámara que utilice puerto USB o firewire. Un aspecto importante, es que la conexión firewire permite tener una transferencia de imágenes rápida, aproximadamente 30 fps (frames por segundo), siendo éste una característica a tomar en cuenta, puesto que ayuda a evitar el retraso de los datos necesarios para el sistema de control. Entre más rápido sea la obtención de los datos de visión, el sistema de control será más rápido para tener los movimientos del robot casi en tiempo real.

3.2.2 ReacTIVision

Se utilizó un software llamado *reacTIVision* de código abierto, el cual funge como rastreador de imágenes, principalmente orientado al reconocimiento de marcadores *fiducial* especialmente diseñados **Figura 3.5**, estos son capturados a través del flujo de información mediante el video en tiempo real. Este software fue desarrollado originalmente como componente sensorial del Reactable **Figura 3.6**, un sintetizador modular para aplicaciones multi-touch [21].

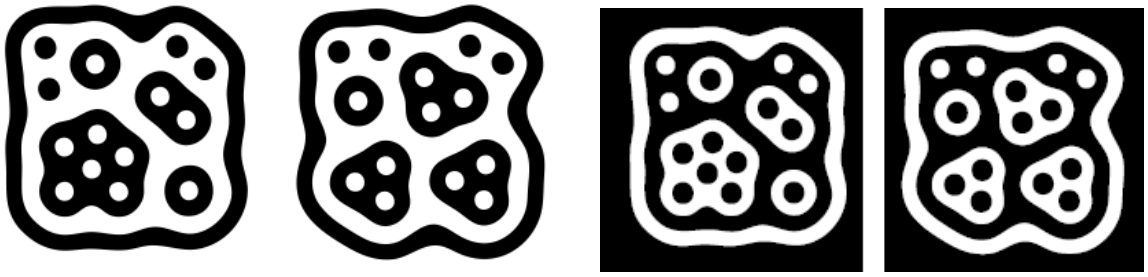


Figura 3.5: *Ejemplos de marcadores fiducial*

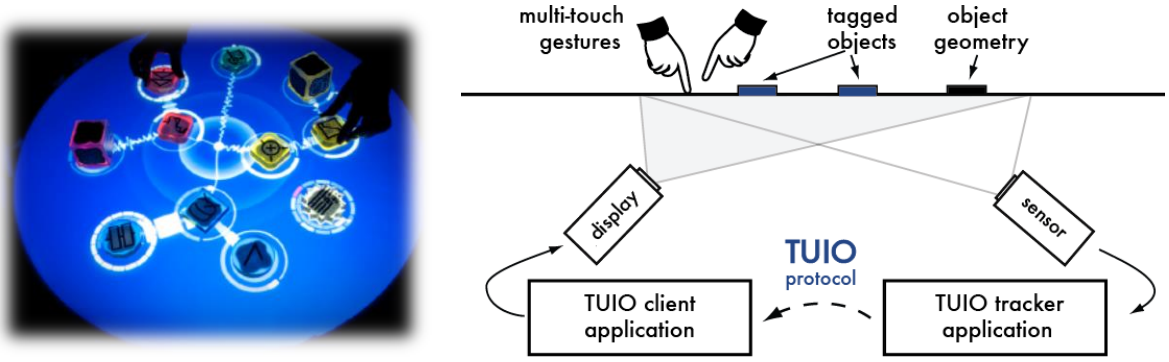


Figura 3.6: *Sintetizador Reactable*

El funcionamiento del sistema de procesamiento de imágenes de reacTIVision trabaja de la siguiente forma: una vez que encuentra el marcador fiducial del video en tiempo real, convierte la imagen en binarizado (en blanco y negro) mediante un algoritmo de umbral adaptativo, posteriormente la imagen es segmentada en un árbol de regiones negras y blancas cambiantes (grafica de región adyacente). En las gráficas se buscan secuencias específicas que se codifican en un símbolo fiducial. Por último, las secuencias de los arboles encontrados se hacen coincidir con una base de datos para asignarle un número de identificación único. El diseño del marcador le permite obtener fácilmente el cálculo del centroide, así como su orientación.

Los mensajes OSC implementados en el protocolo TUIO codifican la presencia del fiducial, su ubicación, orientación y su número de identificación para ser transmitidos a las aplicaciones cliente.

ReacTIVision es una aplicación independiente, que envía mensajes TUIO, a través del puerto UDP 333, a cualquier aplicación que tenga habilitado el cliente TUIO en la misma computadora. Estos clientes TUIO se encuentran en varios lenguajes de programación

(C++, Java, C#, Processing, Pure Data, Max/MSP, Quartz Composer) de libre acceso para su uso [22].

Después del algoritmo descrito anteriormente sobre la detección de los fiducial, el software realiza un proceso con el cual calcula la posición y orientación del fiducial, la secuencia de esto se puede ver gráficamente en la **Figura 3.7**.

Lo primero que hace, es calcular el centroide del marcador a partir de todos los puntos negros y blancos **Figura 3.7 (b)**, luego con solo los puntos negros vuelve a calcular un nuevo centroide **Figura 3.7 (c)**, finalmente estos dos centroides generan un vector para conocer la orientación del fiducial **Figura 3.7 (d)**. Con esta información, el software es capaz de obtener los demás parámetros que ofrece ReacTIVision y posteriormente los envía por el protocolo TUIO.

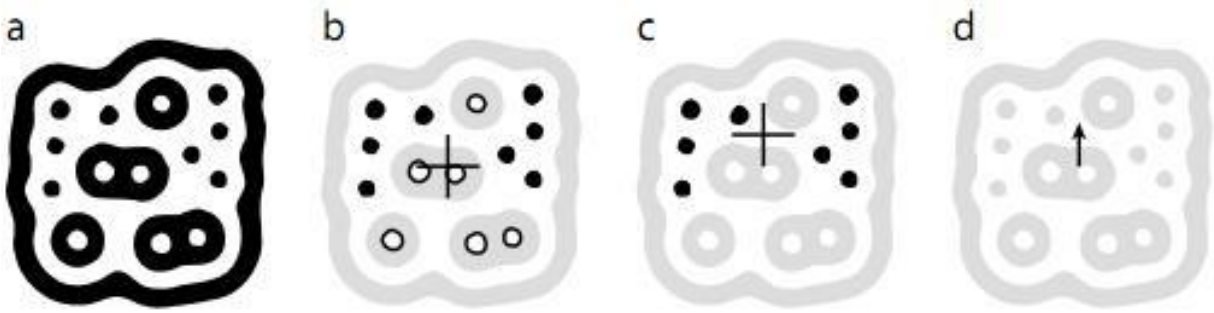


Figura 3.7: *Secuencia para la obtención de los datos del fiducial*

La información que ofrece ReacTIVision es muy útil para nuestra aplicación, montando uno de los marcadores fiducial sobre el centro de la plataforma móvil, se logra obtener su posición y orientación. Los datos arrojados por ReacTIVision, servirán para dibujar la trayectoria trazada por el centro de la plataforma móvil y de esta forma corroborar el rendimiento del robot durante la prueba, por otra parte los datos de visión son aprovechados para corregir los errores (relativamente en tiempo real) de la trayectoria ejecutada por el manipulador.

3.3 Comunicación con el robot

Para la comunicación entre la computadora y los actuadores, así como el control de estos, se utilizó Arduino. Es una plataforma de código abierto basado en software y hardware flexible y fácil de usar. Arduino cuenta con distintos tipos de tarjetas, los cuales trabajan con microcontroladores de la compañía Atmel.

Estos microcontroladores sobre la tarjeta Arduino, se programan mediante su propio lenguaje de programación Arduino (basado en wiring), éste es un lenguaje de programación similar a C. El software lo proporciona la misma compañía donde puede ser adquirido de la página de Arduino [23]. El software cuenta con varias librerías que sirven para salidas a PWM, comunicación serial, comunicación I^2C, interrupciones, entre otras funciones útiles.

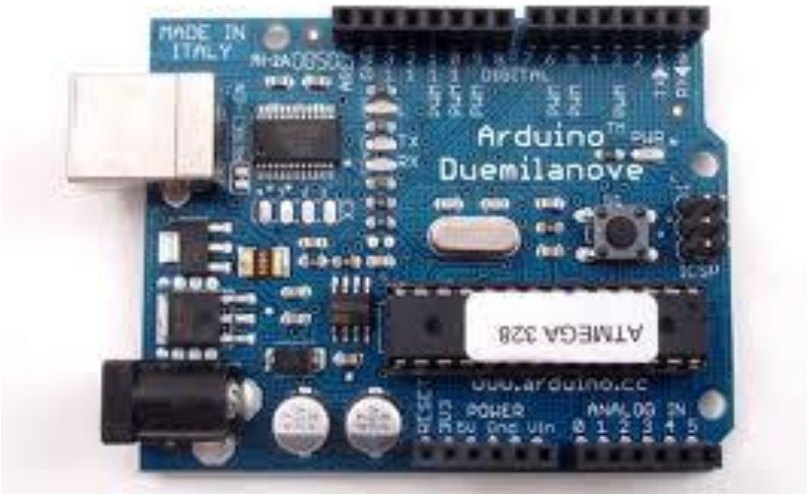


Figura 3.8: *Arduino Duemilanove con microcontrolador Atmega328.*

En particular, la tarjeta que se ocupo tiene el nombre de Arduino Duemilanove **Figura 3.8**, el cual cuenta con un microcontrolador Atmega328 (sus datos técnicos en el **Apéndice E2**. La tarjeta tiene el soporte para conectarse a la computadora por medio de USB, tanto para cargar los programas como para tener comunicación serial.

La consideración de esta tarjeta fue por su flexibilidad y facilidad en el manejo de la programación y por su comunicación, aunado a esto cuenta con librerías PWM útiles para el control de los servomotores.

3.4 Sistema de control

Esta es la sección encargada de adquirir los datos provenientes del sistema de visión y de la ejecución de todos los cálculos correspondientes, también realiza la simulación del movimiento del robot durante las pruebas y por último envía los datos de las posiciones angulares al microcontrolador. Todos los procesos son realizados por una computadora mediante el software MATLAB, que cuenta con una paquetería llamada Simulink de la compañía MathWorks, dicha paquetería es la utilizada para la programación del sistema de control.

La razón de utilizar Simulink, es por su gran variedad de herramientas, tanto de cálculo, operaciones lógicas, sistemas de control y de simulación, además el entorno de programación para el desarrollo de algoritmos es de fácil entendimiento, el cual permite seguir una secuencia lógica del programa por su fácil implementación.

Por otra parte, Simulink es un entorno de simulación multidominio con diseño basado en modelos para sistemas dinámicos y embebidos. Tiene un entorno grafico interactivo y personalizable en librerías de bloques, los cuales permiten diseñar, simular, implementar y probar sistemas variables con el tiempo, por ejemplo en comunicación, control, procesamiento de señales, video e imagen. Este está integrado con MATLAB, el cual ofrece un acceso inmediato a una amplia gama de herramientas [24].

Un esquema general del proceso del flujo de información dentro del sistema de control en simulink, se puede observar en la **Figura 3.9.** Donde muestra los dos algoritmos utilizados para las pruebas, resaltando por un lado el programa en lazo abierto (recuadro anaranjado) y el programa en lazo cerrado, donde este último, los datos de visión son aprovechados para corregir el error de posición de la plataforma móvil basado en un control proporcional-integral.

El sistema de control de lazo cerrado utilizado para controlar las variables del proceso, en nuestro caso la posición (x, y), se eligió por ser un control robusto, fácil de implementar y a su vez no requiere de conocer el sistema. Además, es un método de control muy utilizado en las industrias por las ventajas ya antes mencionadas. No obstante, tiene sus desventajas, no es sencilla la sintonización o calibración de las ganancias, el sistema puede hacerse inestable y puede no ser óptimo para todos los sistemas.

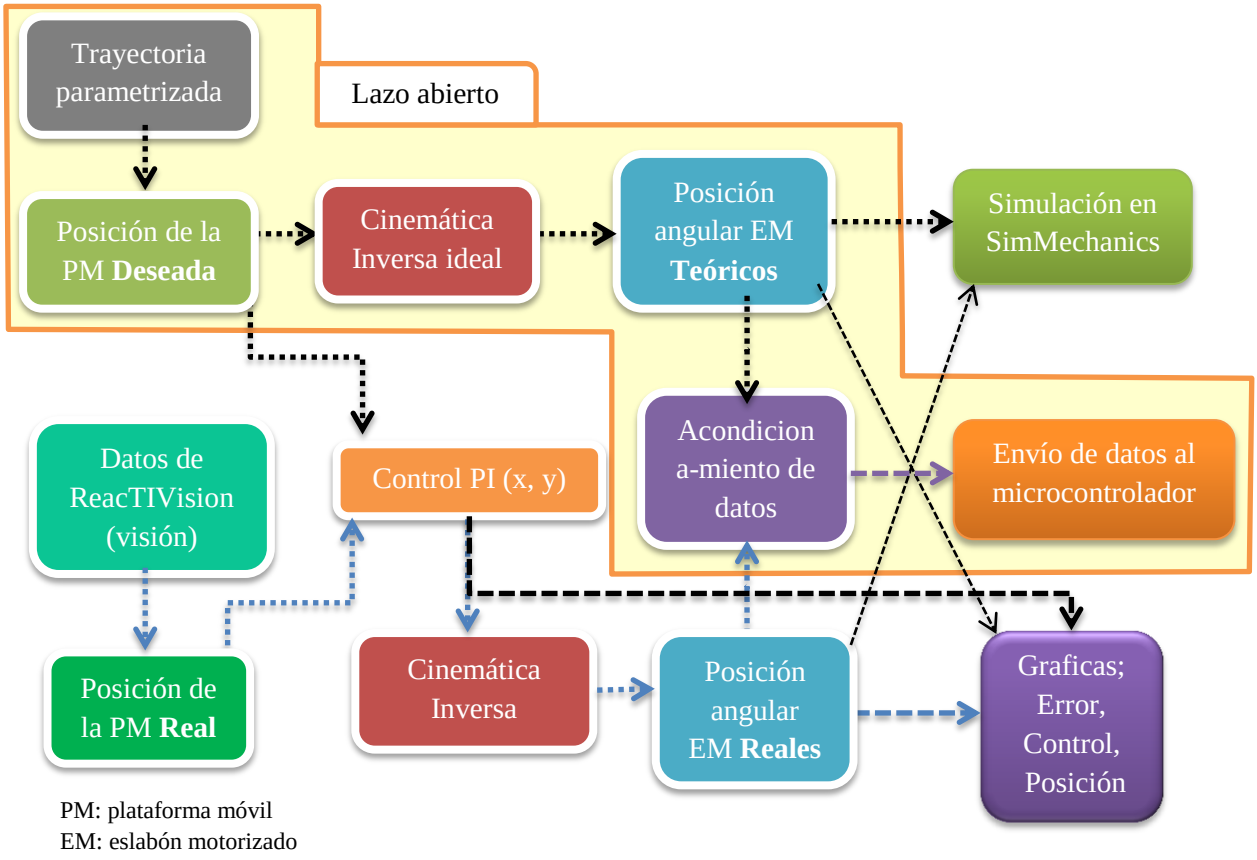


Figura 3.9: *Diagrama general del sistema de control*

La elección del control PI (proporcional-integral) se basó en la experimentación con el modelo funcional. Las pruebas del sistema de visión junto con el modelo funcional demostraron en conjunto un buen desempeño aplicado un controlador con una acción proporcional e integral, sobre la señal del error. Puesto que, este controlador se caracteriza de la siguiente forma:

Acción de control Proporcional (P): da una salida del controlador que es proporcional al error, es decir: $u(t) = Kp\, e(t)$

En su función de transferencia queda: $U(s) = Kp\, E(s)$

Dónde: *Kp* es una guanacia proporcional ajustable. Posee un desempeño limitado y error en estado permanente (off-set).

Acción de control Integral (I): da una salida del controlador que es proporcional al error acumulado, lo que implica que es un modo de control lento:

$$u(t) = ki \int e(t)\, dt \qquad U(s) = \frac{ki}{s} E(s)$$

Dada una referencia constante, el error en estado permanente es cero, siendo una ventaja comparada con la acción proporcional. Sin embargo, la acción derivativa proporciona la siguiente característica, el cual no es conveniente para nuestro sistema:

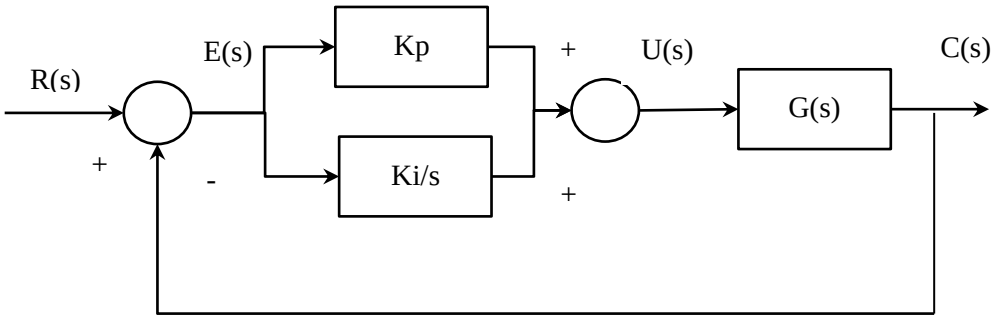
Acción de control Derivativa (D): entrega una señal proporcional a la velocidad de cambio de la señal de error.

La razón de no incluir una acción derivativa en el controlador es por su característica, esto se debe a que en las pruebas, el error está cambiando continuamente. Por lo cual, al incluir esta acción en los experimentos, la salida del control cambiaba rápidamente y no le daba tiempo al sistema de visión en actualizar la posición nueva del marcador fiducial, como resultado el robot se hacía inestable.

Combinando las dos acciones (P, I) se tiene un controlador PI:

$$\frac{U(s)}{E(s)} = \frac{kp\, s + ki}{s}$$

Su esquema queda de la siguiente forma:



Finalmente el bloque en Simulink donde calcula el control PI se muestra en la **Figura 3.10**

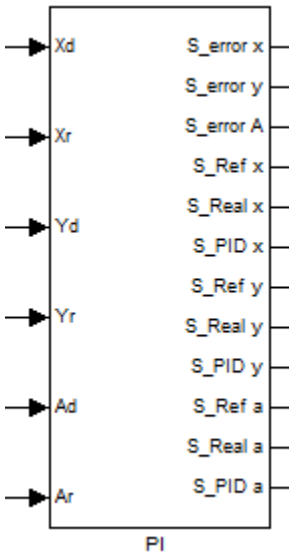


Figura 3.10: *Bloque de Simulink del control PI para controlar las variables del proceso*

Para la sintonización o calibración de las ganancias tanto de la acción integral como de la acción proporcional, fueron obtenidas con base a la experimentación con el robot. Finalmente el manipulador alcanzó a tener un buen desempeño con las ganancias obtenidas. El motivo de no haber realizado el modelo del sistema para poder diseñar una salida del controlador deseada, fue porque en principio sólo se deseaba aplicar alguno de los sistemas de control existentes (redes neuronales, lógica difusa, algoritmos genéticos, etc.) para poder corregir el error de posición de la PM, y uno de los más sencillos de implementar (en nuestro caso) fue el control PI, además de las ventajas mencionadas anteriormente.

El programa en diagrama de bloques en simulink de la **Figura 3.9** se puede consultar en el **Apéndice C**. Este programa se compone de tres partes, uno de ellos es la obtención de los datos de ReacTIVision para su utilización en simulink, la simulación en SimMechanics y el acondicionamiento de los datos para ser enviados al microcontrolador. A continuación estas tres partes se explican con más detalle.

3.4.1 Obtención de los datos de visión en Simulink

ReacTIVision proporciona una interfaz muy sencilla y fácil de usar **Figura 3.11**, éste permite variar algunos parámetros disponibles para el usuario. Por ejemplo, los parámetros que se pueden editar, es invertir el incremento de la posición del fiducial en el eje x, y o el ángulo de orientación del fiducial (cambiarlo en sentido horario o antihorario), calibrar el área de trabajo para la detección de los marcadores, ajustar la sensibilidad de la luz ambiente y entre otras funciones disponibles. Si se desea, la misma interfaz permite mostrar continuamente la posición, orientación y los demás parámetros del fiducial, casi en tiempo real.

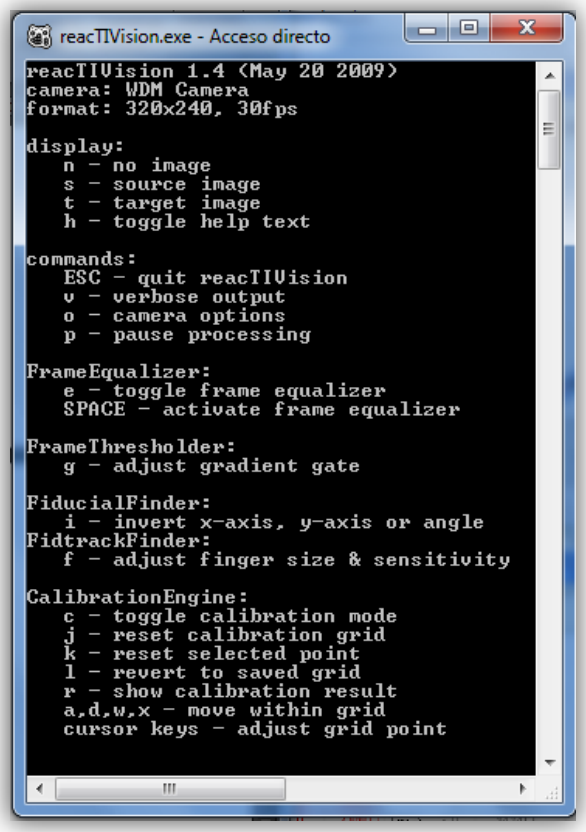


Figura 3.11: *Interfaz de ReacTIVision.*

Es importante editar estos parámetros antes de utilizar o enviar los datos en alguna otra plataforma, evitando de esta manera la incongruencia en los cálculos.

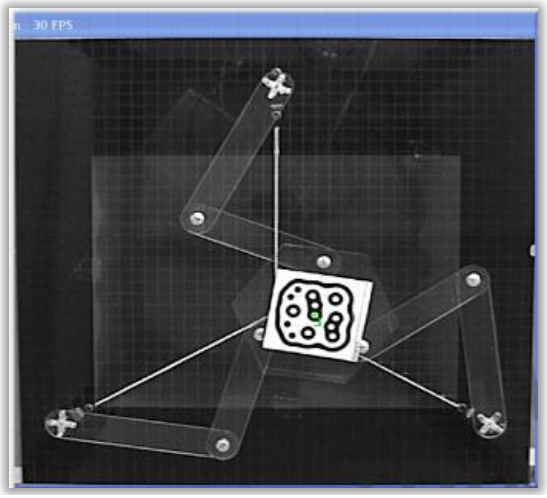


Figura 3.12: *Imagen de ReacTIVision detectando el fiducial.*

Además de la interfaz, ReacTIVision también muestra la captura de la cámara, marcando el centroide y el número de identificación del fiducial **Figura 3.12** en tiempo real. De esta forma, se puede ir observando el movimiento del robot, apreciando visualmente sus irregularidades.

La comunicación entre Simulink y ReacTIVision se da a partir del cliente TUIO como se mencionó anteriormente, para ello es necesario incluir dentro de las librerías de MATLAB el cliente TUIO Java, eso se debe a que MATLAB trabaja con ese tipo de lenguaje. Para instalar el cliente TUIO se puede ver el **Apéndice E4**.

Con la instalación del cliente TUIO Java, MATLAB es capaz de ejecutar ciertos comandos para adquirir la información de ReacTIVision, en el **Apéndice E1** se encuentran dichos comandos. A partir de esto se pueden programar objetos de esta clase dentro de una función *S* en Simulink, declarando los siguientes comandos:

```
Import TUIO.*;
tc = TuioClient();
tc.conect();
```

Con esto el objeto esta creado y se pueden adquirir los parámetros de ReacTIVision. Nuestra aplicación en particular requiere sólo de la posición en x, y y la posición angular del fiducial, por lo tanto se utilizan los siguientes comandos:

```
x=tc.getTuioObjects().get(#).getPosition().getX();
y=tc.getTuioObjects().get(#).getPosition().getY();
a=tc.getTuioObjects().get(#).getAngle();
```

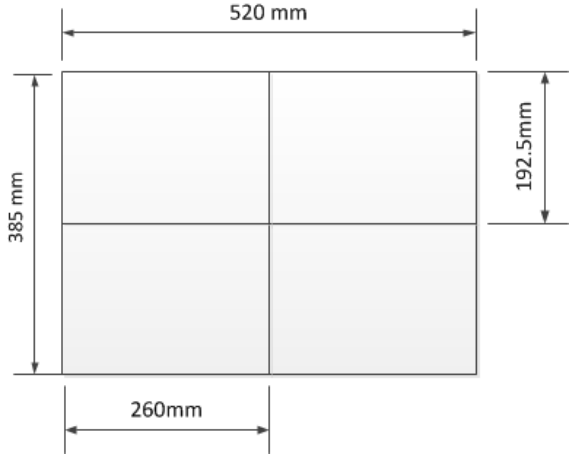
El símbolo #, se refiere al orden de aparición de los símbolos fiducial, por ejemplo si se pone 1, se guarda la información del primer fiducial que encuentre. Por último, se debe terminar la comunicación con el software del objeto creado, a partir del comando:

```
tc.disconnect();
```

Los comandos anteriores, dentro de una función *S* en Simulink, permite la comunicación con el sistema de visión. Por otro lado, ya estando en la función *S,* debe existir una pequeña pausa entre el dato anterior y el dato actual para que se actualice la nueva posición del fiducial, sin esto la información permanece constante y todo el tiempo se obtiene el mismo dato. Aunque al incluir esta pausa, la velocidad de muestreo se reducía, generando lentitud en la adquisición de los datos del sistema de visión. Para ello, se hicieron pequeñas pruebas tratando de encontrar una pausa muy pequeña que no afectara mucho a la velocidad de muestreo para la función *S* y al mismo tiempo evitar pérdidas de información del marcador.

Antes de utilizar la información de ReacTIVision, primero se hizo un acoplamiento de los datos. Porque la posición en (x, y) esta escalado de 0 a 1 y el cero del sistema de visión se localiza por default en la esquina inferior izquierda o dependiendo de cómo se configuren las direcciones de los ejes en la interfaz, además el ángulo esta dado de 0 a 2π.

El escalamiento de los datos se solucionó con simples operaciones en el caso de los ejes, tomando en cuenta las dimensiones del área capturada por la cámara **Figura 3.13** y restando la mitad de cada lado para obtener el centro del sistema justo en el centro del área de trabajo del robot.



$$x = x * 520 - 260$$
$$y = y * 385 - 192.5$$

Figura 3.13: *Dimensiones del área capturada por la cámara*

Para el ángulo, solamente se cambió de radianes a grados y además se cambió la referencia del marcador fiducial (su cero), así como el sentido del ángulo, mostrado en la **Figura 3.12**.

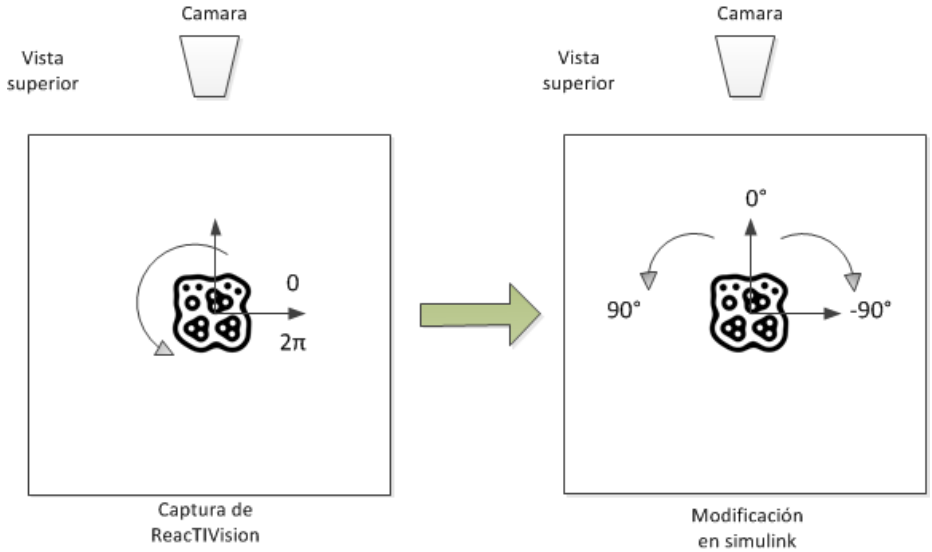


Figura 3.14: *Acoplamiento de la posición angular del fiducial en la función S*

En el **Apéndice B1** se muestra el programa de la función S donde incluye todo lo comentado anteriormente, y una vez creado esto, dentro de Simulink se manda a llamar dicha función utilizando un bloque, mostrado en la **Figura 3.15**.

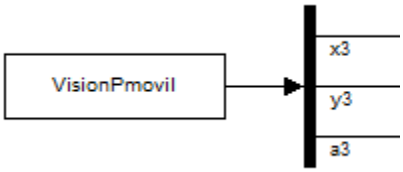


Figura 3.15: *Bloque en Simulink que llama a la función S.*

3.4.2 Simulación en SimMechanics

El propósito de hacer simulaciones nos permite corroborar de manera rápida si los resultados son coherentes o poder diseñar pruebas antes de ser aplicados al modelo real, además aportan una representación visual del comportamiento del robot. Por lo tanto, esto nos ayuda a observar movimientos previstos o no esperados del robot, ocasionados algunas veces por errores en los cálculos o limitantes del robot.

En la simulación se decidió utilizar un software llamado SimMechanics, es una herramienta con el cual se puede modelar y simular sistemas mecánicos de forma fácil y eficiente dentro de la interfaz gráfica de Simulink, ofreciendo distintas herramientas que permiten especificar las propiedades de un cuerpo, por ejemplo, la masa, la inercia, los grados de libertad o los ejes coordenados de los cuerpos. Por lo tanto, esto nos permite unir todo en un solo entorno de programación y no tener que recurrir a la integración de otro tipo de software.

De manera más detallada, SimMechanics es un ambiente de simulación en 3D para sistemas mecánicos multicuerpos, por ejemplo robots, suspensiones de vehículos, equipo de construcción o máquinas de aeronaves, entre otros. En el modelado de sistemas multicuerpos, este software utiliza diagramas de bloques que representan los cuerpos, las articulaciones, las restricciones cinemáticas o los elementos de fuerza, por lo que, posteriormente SimMechanics formula y resuelve las ecuaciones de movimiento del sistema mecánico en conjunto [25].

3.4.2a Construcción del MDP 3RRR en SimMechanics

La construcción de sistemas mecánicos se realiza por medio de diagramas de bloques iguales a los de la **Figura 3.16**, la lógica de programación es similar a la de simulink. Los bloques se encuentran en diferentes bibliotecas y organizados por SimMechanics e incluidos dentro de la paquetería de Simulink.

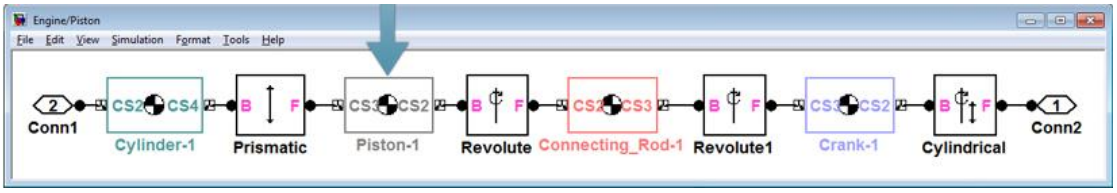


Figura 3.16: *Representación de un diagrama de bloques en SimMechanics*

La animación en SimMechanics permite visualizar los cuerpos en figuras geométricas o por medio de la importación de objetos realizados en CAD **Figura 3.17**. Esto último se puede lograr mediante un convertidor de SimMechanics o ensamblando las piezas manualmente. Los editores gráficos que soportan dicho add-in tool para guardar el ensamble CAD a un archivo XML e importarlo a SimMechanics son; SolidWorks, Pro/ENGINEER y Autodesk Inventor.

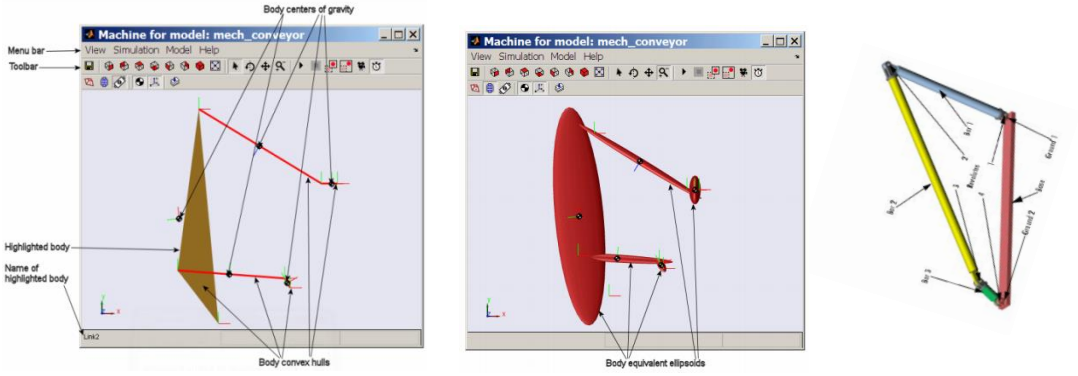


Figura 3.17: *Formas para visualizar la simulación*

Para entender cómo modelar un mecanismo con este tipo de software, se explicara brevemente la construcción del MDP 3RRR para obtener el diagrama de bloques en SimMechanics, así mismo, se muestran las dos formas de obtener su ensamble.

La forma más fácil de obtener el diagrama de bloques del MDP fue por medio del ensamble realizado en CAD, en este caso se utilizó SolidWork para el diseño de las piezas del manipulador y obviamente para el ensamble incluyendo sus restricciones cinemáticas **Figura 3.18.**

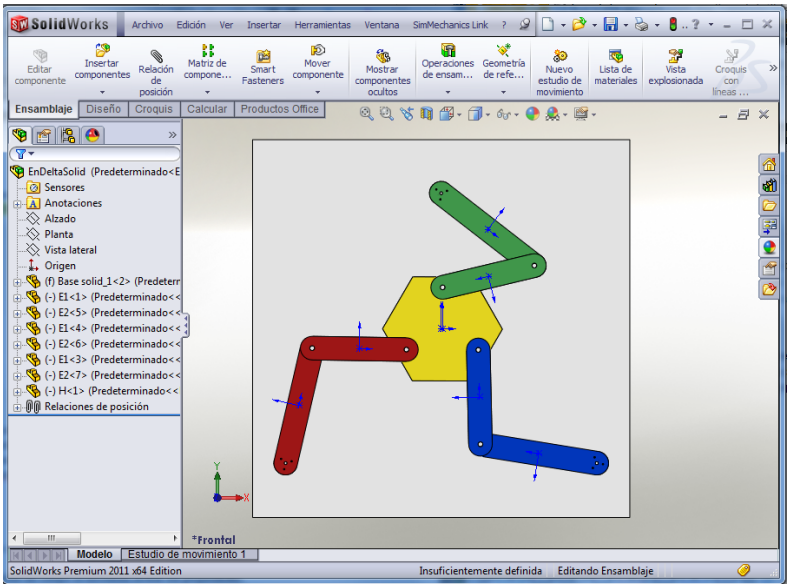


Figura 3.18: *Ensamble realizado en SolidWorks*

Después se instaló el add-in tool en SolidWorks, para instalarlo ver el **Apéndice E5**. Una vez que instalado el add-in tool debe aparecer la opción "SimMechanics Link" en la parte superior de la barra de SolidWorks **Figura 3.19 (a)**, esto indica que se puede guardar el ensamble en el formato XML **Figura 3.19 (b).**

a)

b)

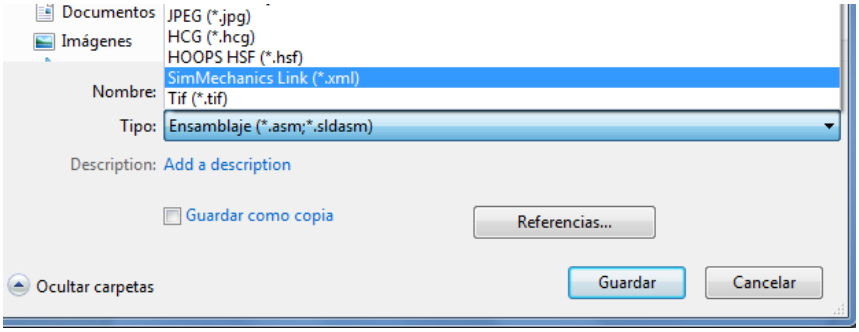


Figura 3.19: *a) Add-in tool instalado en SolidWorks, b) Opción para guardar el ensamble en XML*

Ya realizado lo anterior, el software genera un archivo en XML y varios archivos en formato STL **Figura 3.20,** el número de estos archivos corresponde al número de cuerpos que tiene el manipulador. Por último se guardan los archivos generados por SolidWorks en una nueva carpeta y desde MATLAB se abre la carpeta para poder ejecutar el comando "mech_import", de esta manera aparece una nueva ventana donde se pude elegir el archivo XML y convertirlo en un diagrama de bloques de SimMechanics.

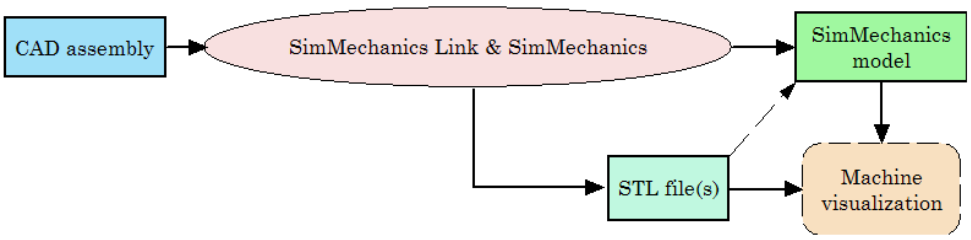


Figura 3.20: *Diagrama de la conversión del ensamble en CAD*

Finalmente se obtiene el diagrama de bloques **Figura 3.21**. Aunque, realizar este ensamble por este método requiere de un buen ensamblado desde el CAD o definir bien las propiedades de los cuerpos

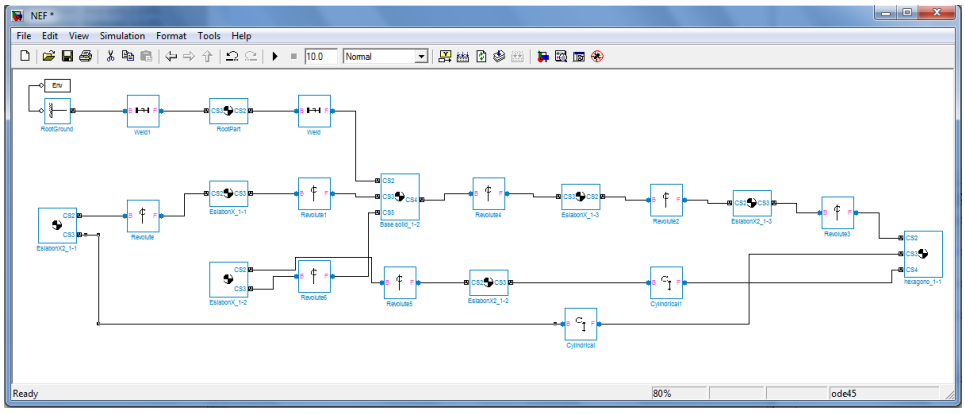


Figura 3.21: *Diagrama del MDP generado por medio del CAD*

La forma más tediosa pero fácil a su vez para obtener el ensamble en SimMechanics es la forma manual. Sin embargo, se debe conocer bien los movimientos del robot para evitar errores al ejecutar la simulación.

La otra opción fue ensamblar el MDP manualmente, con esto se garantizó un buen ensamble, con propiedades correctas y posiciones iniciales deseadas, para ello fue necesario conocer las propiedades de los bloques que ofrece SimMechanics. En este proceso solo es necesario crear los archivos STL de cada pieza del robot o mecanismo en CAD y guardarlos en un archivo donde se desea crear el diagrama de bloques en SimMechanics.

SimMechanics cuenta con una clasificación de librerías para construir un sistema mecánico, entre ellas una librería para cuerpos, juntas, elementos de fuerza, sensores y actuadores, y algunas otras herramientas **Figura 3.22.**

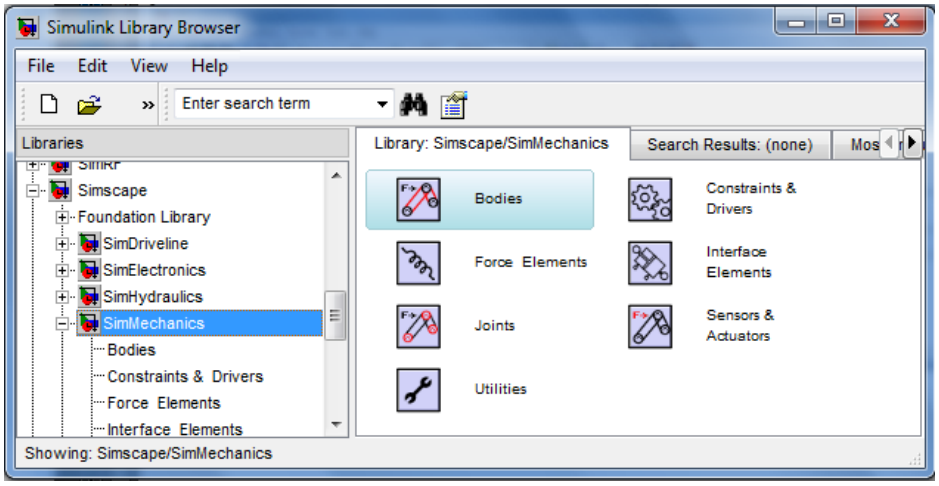


Figura 3.22: *Librerías de SimMechanics*

Dentro de la librería de cuerpos, están los elementos importantes para comenzar a crear un diagrama, por ejemplo está el bloque para un cuerpo, la tierra y las propiedades de animación. En el MDP se utilizaron los boques de la **Figura 3.23**, los cuales se refieren a la animación, la tierra y a una primera junta con cero grados de libertad, fija a la tierra. En cada bloque se deben editar ciertas propiedades necesarias para la simulación.

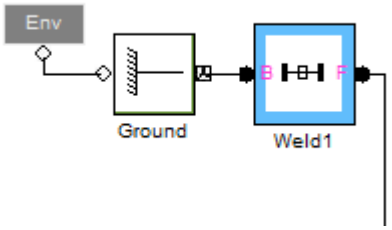


Figura 3.23: *Bloques importantes para simular el mecanismo*

Posteriormente se ensamblaron los cuerpos con un orden consecutivo, como si se estuviera conectando el robot físicamente **Figura 3.24.** Para unir dos cuerpos se colocaron juntas y sobre este mismo se pueden acoplar actuadores o sensores para mover el cuerpo o monitorear el movimiento.

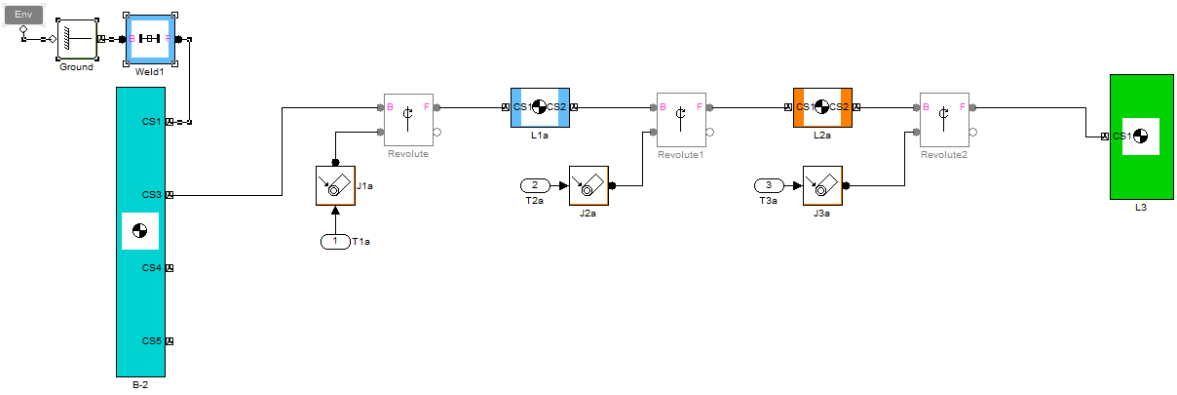


Figura 3.24: *Orden de los cuerpos en el diagrama de bloques.*

El bloque que representa a un cuerpo contiene los parámetros para editar su posición, orientación y su visualización en la simulación. En la visualización de los cuerpos puede ser por medio de figuras geométricas (basado en líneas), elipsoides equivalentes con masas o archivos gráficos externos (CAD), en este último se refiere a importar los archivos STL del diseño en CAD **Figura 3.25**.

Al momento de crear las piezas en CAD se debe tomar en cuenta el centro del cuerpo (centro u origen de su sistema de coordenadas), dado a que ese centro se adjunta al centroide del cuerpo en SimMechanics.

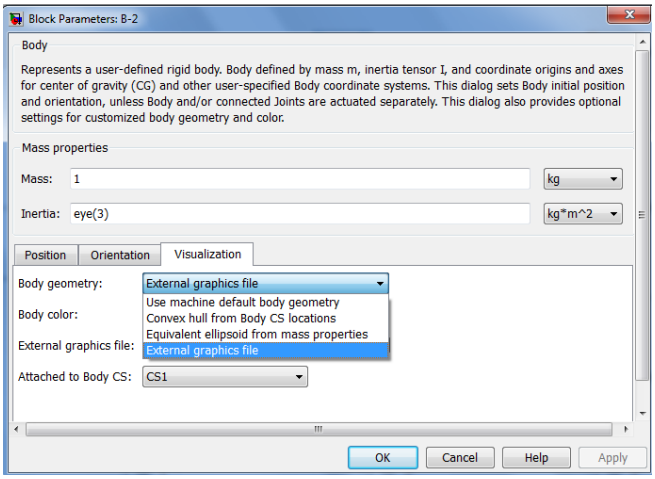


Figura 3.25: *Opciones para visualizar el cuerpo en la simulación*

En este caso las posiciones del MDP se editaron pieza por pieza para evitar errores y tener un buen ensamble. Por último se colocaron los actuadores para generar el movimiento en la simulación, en cada junta fue necesario un actuador para su correcto movimiento. El bloque cuenta con varias opciones para actuar un cuerpo **Figura 3.26**, por torque, por velocidad, por rpm o por posición angular. Este último fue el utilizado sobre el MDP ya que a partir de la solución de la cinemática inversa se obtienen las posiciones angulares del MDP.

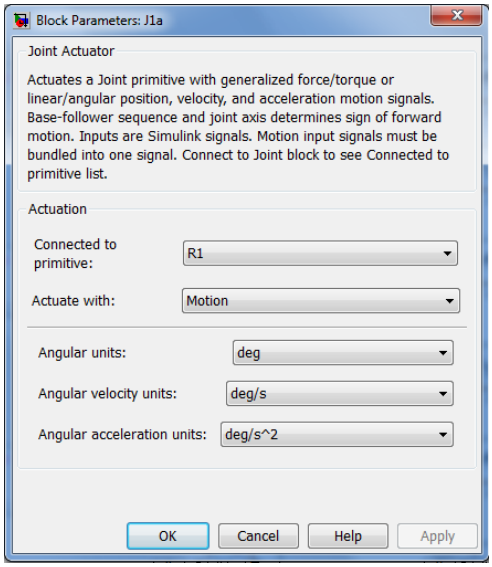


Figura 3.26: *Parámetros del bloque de actuación*

Al final se logra simular el movimiento del mecanismo si se tiene un buen ensamble de las piezas y si los datos enviados a los actuadores son los correctos. La simulación se da en 3D con ciertos comandos para manipular la animación. En la **Figura 3.27** se muestra el MDP 3RRR dentro del ambiente de SimMechanics.

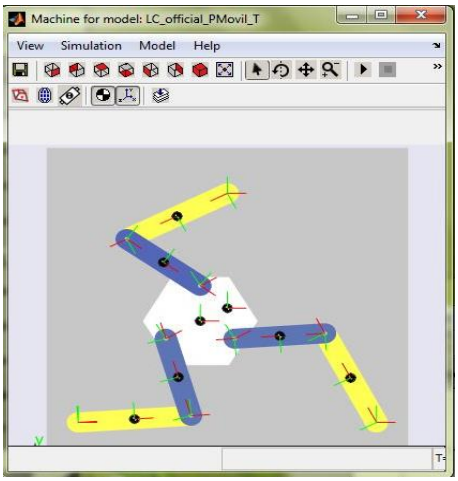


Figura 3.27: *La animación del MDP 3RRR en SimMechanics*

3.4.3 Acondicionamiento de los datos para la comunicación

Dentro del sistema de control, una vez realizado los cálculos, es necesario enviar los datos a los actuadores. Sin embargo, antes de enviar la información se requiere acondicionarlos para que el microcontrolador los reciba.

Durante algunas pruebas, se observó que era necesario este proceso de acondicionamiento de los datos, para reducir algunos errores de posición de los servomotores. El primer paso, fue caracterizar los servomotores debido a errores de posición angular y al rango de operación que presentaban. Con la ayuda del sistema de visión, se logró observar los errores en los servomotores, por tanto se decidió tratar de mejorarlos en base al acondicionamiento de los datos.

La prueba con el sistema de visión consistió en montar un fiducial sobre cada servo, para ir observando su posición real después de enviarle una posición angular deseada. Los resultados que se obtuvieron se muestran en la **Figura 3.28**. Aquí se puede ver la comparación del ángulo deseado con el ángulo real de los servos, de esta manera se aprecia que los servos no son totalmente lineales y no tienen un rango de operación de 0° a 180°.

a)

b)

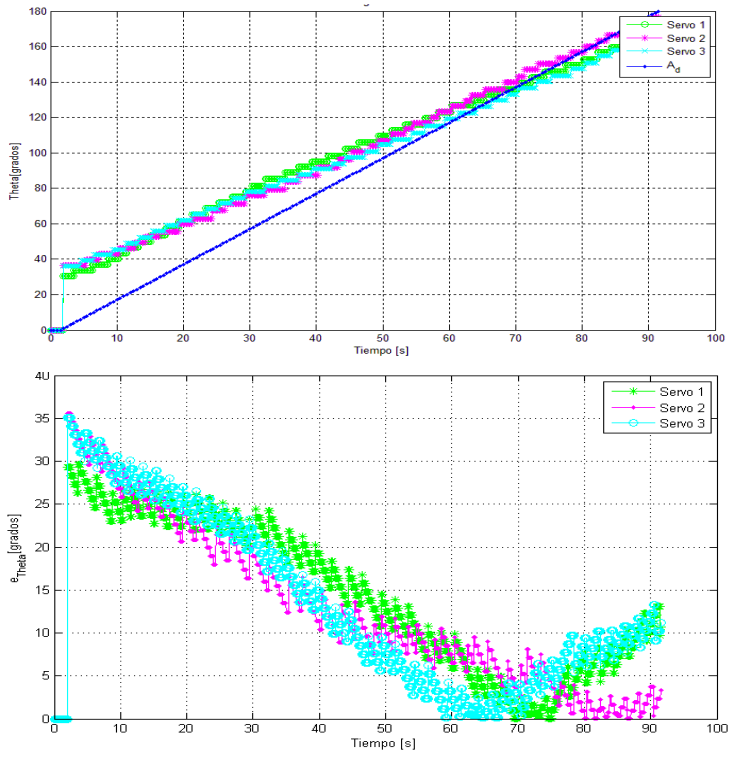


Figura 3.28: *a) Posición angular real y deseada de los servos, b) Error absoluto de cada servo.*

Algo importante a tomar en cuenta, es que existía mucho error en los servomotores como lo muestra la **Figura 3.28 (b)**, alcanzando a tener errores máximos de 30 a 36 grados. Estos errores afectaban demasiado sobre la posición de la plataforma móvil.

Otra cuestión importante del acondicionamiento de los datos, es que dentro de los resultados de la cinemática inversa se obtienen posiciones angulares con punto decimal, por lo que Simulink en su comunicación serial solo permite enviar números enteros sin signo, lo cual es un inconveniente porque se perdían datos al momento de redondear las posiciones angulares.

Como sabemos los servomotores se controlan por modulación de ancho de pulso (PWM), el cual consiste en generar una onda cuadrada en la que se varía el tiempo del pulso para lograr el movimiento del servo según se desee, pero manteniendo el mismo periodo. De acuerdo con los datos técnicos del fabricante los servos trabajan con un ancho de puso mínimo y máximo de 800 a 2200 μs. Pero dichos datos fueron diferentes al corroborarlos con un osciloscopio, de esta forma se verifico cual era su ancho de pulso real para tener un margen de operación de 0° a 180°, como se muestra en la **Figura 3.29**.

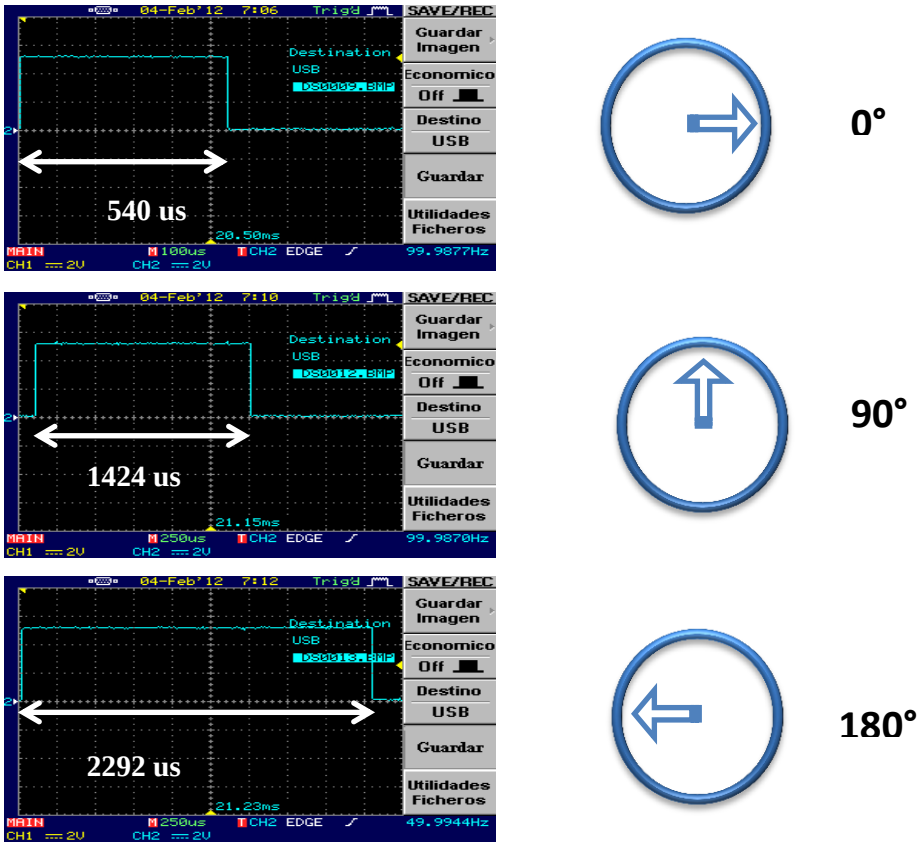


Figura 3.29: *Ejemplo del ancho de pulso mínimo, medio y máximo real de un servomotor*

El ancho de pulso mínimo y máximo de los otros dos servomotores se pueden ver en el **Apéndice E3**.

El método para reducir los errores de posición debido a su no linealidad de los servomotores, se fue iterando la posición angular, tomando como referencia el ancho de pulso mínimo, medio y máximo de cada uno, con esto se logró que los servos mejoraran en cuanto a su posición angular. En la **Figura 3.30 (a)** se observa la mejora de la posición angular de los servos. El error máximo con este método se redujo hasta ocho grados como se muestra en la **Figura 3.30 (b)**.

a)

b)

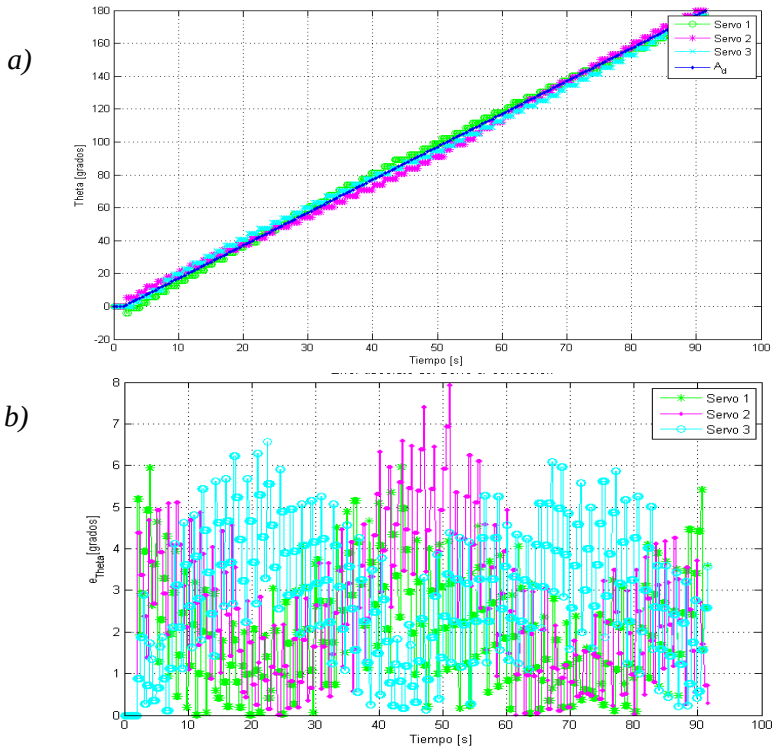


Figura 3.30: *Grafica con mejoras en la posición angular de los servomotores*

En cuanto al problema sobre el redondeo de los datos para la comunicación, lo que se hizo fue convertir los grados a microsegundos y por ejemplo en vez de enviar 0° o 180° enviar 580 μsec o 2250 μsec, de este modo no se pierde demasiada información y aumentamos el rango de posiciones del servo. Para ello, lo que se hizo fue separar los números en dos dígitos debido a que simulink solo envía bytes de 8 bits, lo cual un número en microsegundos es demasiado grande para trasmitirlo en un solo byte.

El programa en Simulink lo que hace es separar el número en dos dígitos y posteriormente en el programa del microcontrolador juntar estos dígitos y así obtener el dato correcto. En la **Figura 3.31** se observa un diagrama general del acondicionamiento de los datos y en el **Apéndice C3** se muestra el diagrama en Simulink**.**

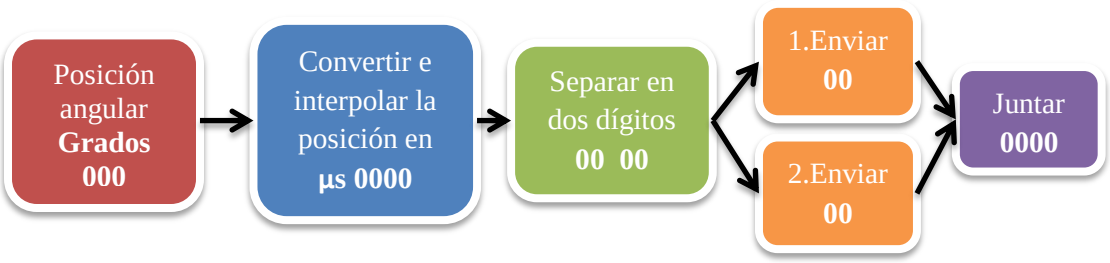


Figura 3.31: *Diagrama general del acondicionamiento de los datos*

Capítulo 4

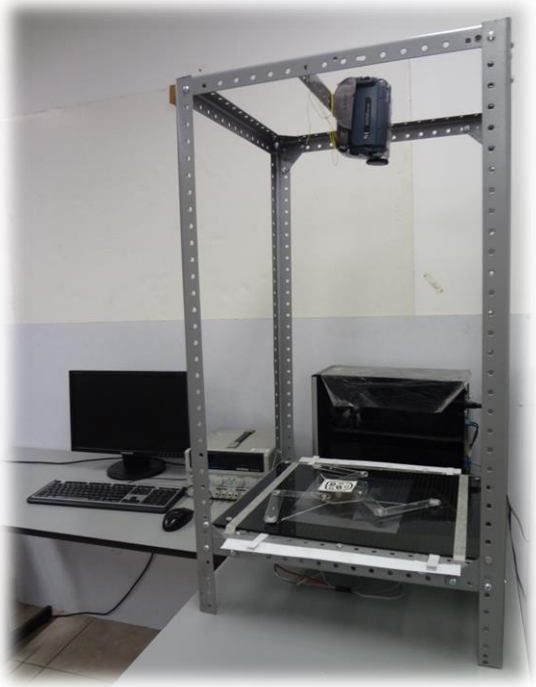
Pruebas y Resultados

Introducción

Este capítulo muestra y describe las pruebas más representativas del seguimiento de trayectorias predefinidas del robot, tanto en lazo abierto como en lazo cerrado. Del mismo modo, se tienen los datos de los resultados del manipulador.

4.1 Entorno de pruebas

Para las pruebas, no se requirió de algún lugar en específico, ya que se construyó la estructura especialmente para fijar la cámara y el robot **Figura 4.1**, solo fue necesario el uso de una computadora con el puerto firewire (proporcionado por el laboratorio). Algunas de las ventajas que presenta la estructura, es evitar inconvenientes externos que puedan mover la base del robot o la descalibración de la cámara. Por otra parte, se requirió controlar la luz ambiental cubriendo la parte superior de la estructura, esto fue porque la luz de las lámparas del laboratorio afectaba al software de visión (reflejaba la luz sobre la base).



La altura entre la cámara y el robot situados sobre la estructura, es de aproximadamente 70 [cm], de esta forma la cámara tenía un campo de visión de aproximadamente 60 [cm] en el eje *x* y 54 [cm] sobre el eje *y*, cubriendo es su totalidad al modelo funcional.

El tamaño del fiducial colocado sobre el centro de la plataforma móvil fue un cuadrado de 7[cm], con este tamaño la cámara alcanzaba a detectar e identificar el marcador sin perderlo.

Figura 4.1: *Área de trabajo del sistema real*

4.2 Descripción general de las pruebas

Las pruebas son pensadas para apreciar visualmente el comportamiento real del manipulador con o sin la retroalimentación del sistema de visión. No obstante, el principal objetivo es observar la mejora del rendimiento de la plataforma móvil utilizando el sistema de visión en un entorno real.

Para ello, se utilizaron trayectorias que representan los movimientos básicos de cualquier combinación sobre el plano, la primera es una línea recta con inclinación, el cual combina el desplazamiento tanto en el eje x como en el eje y siendo un movimiento lineal, la segunda es una trayectoria circular, así podemos observar el desplazamiento no lineal.

Como se había mencionado anteriormente, además del desplazamiento, se incluirá la rotación de la PM (el penduleo) y el perfil de trayectoria, habiendo así varias pruebas ordenadas de la siguiente forma:

- Seguimiento de una Línea recta diagonal
  - En lazo abierto
    - LRD SS: Prueba 1
    - LRD PP: Prueba 2
  - En lazo cerrado
    - LRD SS: Prueba 3
    - LRD PP: Prueba 4
- Seguimiento de una trayectoria Circular
  - En lazo abierto
    - C SS: Prueba 5
    - C PP: Prueba 6
  - En lazo cerrado
    - C SS: Prueba 7
    - C PP: Prueba 8

SS: Sin penduleo y sin perfil de trayectoria.
PP: Con penduleo y con perfil de trayectoria.

Un sistema en lazo abierto es aquel sistema en que solo actúa el proceso sobre la señal de entrada y da como resultado una señal de salida independiente a la señal de entrada, pero basada en la primera.

Por lo tanto, los experimentos de lazo abierto realizados por el robot no se tomara en cuenta la retroalimentación de la información del sistema de visión, únicamente se

enviara la posición articular de las juntas motorizadas y se registrara la posición y orientación de la PM. Estas pruebas nos permitirán apreciar los errores de posición del robot, que posiblemente son ocasionados por problemas mecánicos (fricción, huelgo, etc.) o por perturbaciones externas al robot.

Por otro lado, las pruebas en lazo cerrado, se utiliza la retroalimentación de información del sistema de visión y con esto podremos comprobar la reducción del error de posición de la plataforma móvil.

Sin embargo, es importante mencionar que durante las pruebas con el sistema de visión, se observó que había un inconveniente con respecto a la resolución de la posición angular del fiducial, el problema que tiene, es que solo detecta un cambio mínimo de 2.5 a 3 grados, imposibilitando el control en la orientación ϕ en el caso de las pruebas en lazo cerrado.

Cabe mencionar, que se realizaron algunas pruebas con este inconveniente, tratando de contrarrestar dicho problema pero no se obtuvieron buenos resultados, motivo por el cual se descartaron las pruebas para el control en la orientación ϕ.

Por otra parte, con base a las pruebas experimentales con el modelo funcional, se lograron obtener las ganancias del control PI para las variables (x , y). De la misma forma, aclaro que con dichas ganancias experimentales el manipulador tuvo errores mínimos y solo son propias del modelo funcional, tomando en cuenta las características como medidas dimensionales del robot en general, a los actuadores y al sistema de visión. Con lo cual, las ganancias del controlador PI tal vez no sirvan si alguno de estos parámetros cambia.

Una vez aclarada la obtención de las ganancias experimentales, finalmente se encontraron las siguientes ganancias que fueron ingresados en los bloques de Simulink.

$kp = 1.3$
$ki = 1.6$

Estas ganancias fueron utilizadas en todas las pruebas realizadas, así también es importante aclarar, que en todas las pruebas se utilizó una configuración codo abajo para los brazos del MDP, tipo *B-B-B* visto en el capítulo 2.

La realización de cada prueba con el modelo funcional se describe a continuación.

4.3 Seguimiento de una Línea Recta Diagonal

La prueba 1 consiste en que el centro de la plataforma móvil (PM) siga una trayectoria rectilínea con inclinación de 45° con respecto al sistema $\{x, y\}$, sin giro de la plataforma y sin perfil de trayectoria. Logrando así observar un movimiento básico del robot sobre el plano, tanto en lazo abierto como en lazo cerrado. La magnitud de la línea recta es de aproximadamente 85 [mm], las ecuaciones de la trayectoria parametrizadas en función del tiempo resultan ser las siguientes:

$$x = x1 + (x2 - x1) * t$$
$$y = y1 + (y2 - y1) * t$$
$$x1 = 30;\ x2 = -30;\ y1 = -16;\ y2 = -76;$$

El tiempo establecido para el total del recorrido del punto inicial al punto final es de 45[s] (tiempo de simulink) en esta trayectoria.

4.3.1 Prueba 1 en lazo abierto

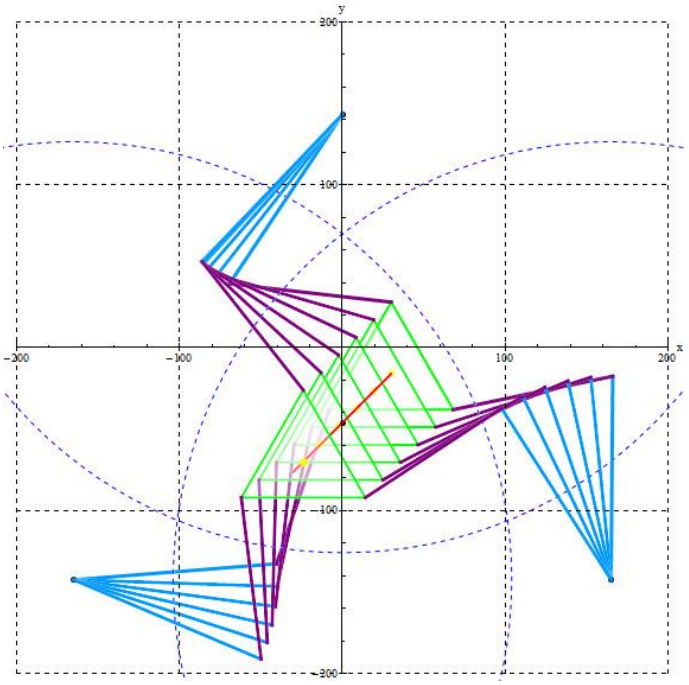


Figura 4.2: *Simulación del MDP describiendo la línea recta diagonal SS.*

En la **Figura 4.2** se muestra la simulación del movimiento de la plataforma describiendo la línea recta diagonal de manera teórica. Realizando este movimiento sobre el modelo funcional, se obtuvieron los resultados mostrados gráficamente dentro de la **Figura 4.3.**

Línea Recta Diagonal (LRD SS) LA

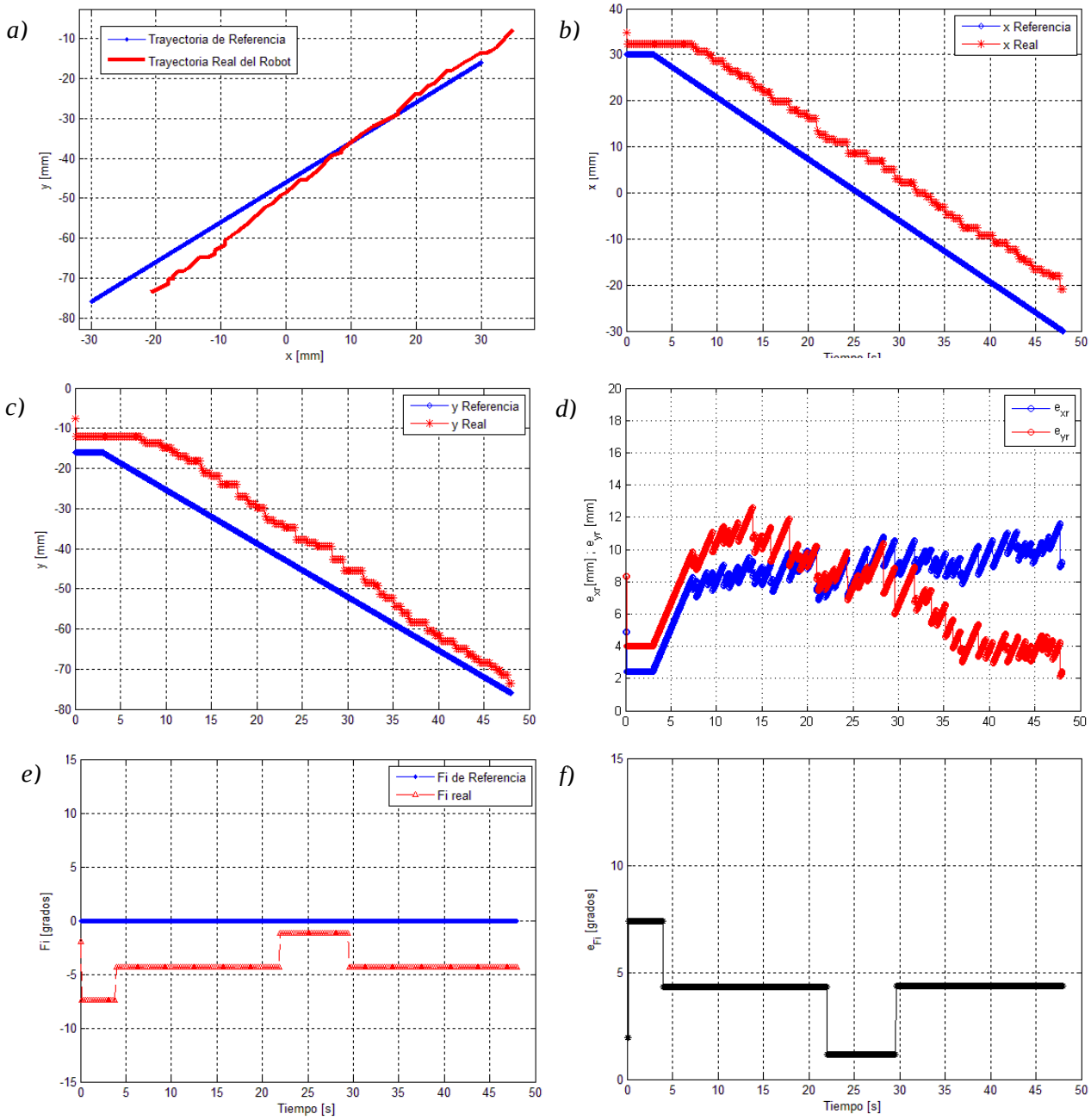


Figura 4.3: *Prueba 1, Línea recta diagonal SS LA,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* x *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* y *con respecto al tiempo, deseado y real.* ***d)*** *Error absoluto del desplazamiento en el eje* x *y en el eje* y *.****e)*** *Rotación de la PM deseado y real.* ***f)*** *Error absoluto de la rotación.*

En esta prueba básica de la línea recta de 45° en lazo abierto, los datos de las gráficas demuestran que el manipulador como era de esperarse no realizo la trayectoria correctamente, sin embargo trato de realizar la trayectoria con respecto a si mismo mas no con respecto al universo. Por ejemplo, las desviaciones del inicio y durante el recorrido se debieron posiblemente a la inercia del robot o huelgo en las juntas entre los eslabones.

En cuanto al error en el desplazamiento en el eje x obtuvo un error mínimo de aproximadamente 7 [mm] y un máximo de 12 [mm], en el eje y alcanzo un error máximo de aproximadamente 13 [mm] y un mínimo de 2[mm], por lo que los errores fueron variando significativamente. Además la orientación de la PM no logro mantenerse fija, oscilando durante su recorrido con un error de cinco grados en promedio.

4.3.2 Prueba 2 en lazo abierto

En esta prueba, la trayectoria es igual a la anterior, sin embargo el movimiento que tiene la plataforma móvil es más complejo, puesto que se combina el desplazamiento con la rotación en la PM, aunado a esto se incluye un perfil de trayectoria de quinto grado, el objetivo de incluir estos dos comportamientos permite aprovechar las capacidades de desplazamiento del robot.

En la **Figura 4.4** se muestra la simulación de dicho movimiento, y posteriormente se muestran los resultados del modelo funcional de los movimientos realizados **Figura 4.5**.

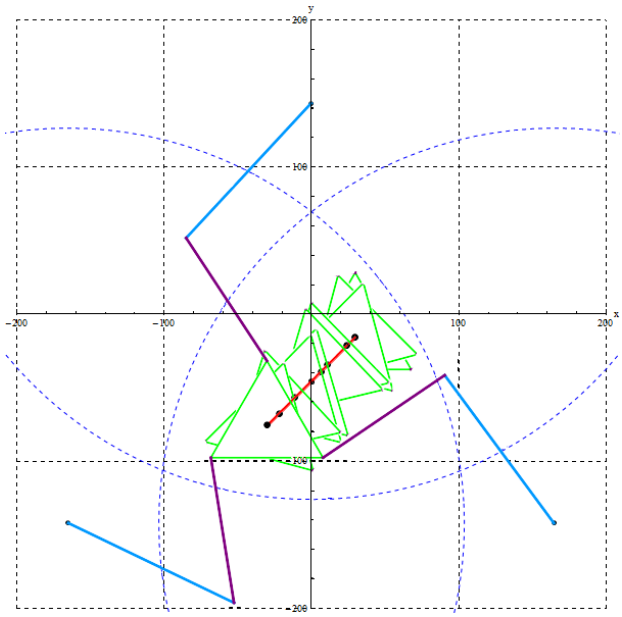


Figura 4.4: *Simulación de la trayectoria del MDP describiendo una línea recta, incluyendo el perfil de trayectoria y el giro de la PM.*

Línea Recta Diagonal (LRD PP) LA

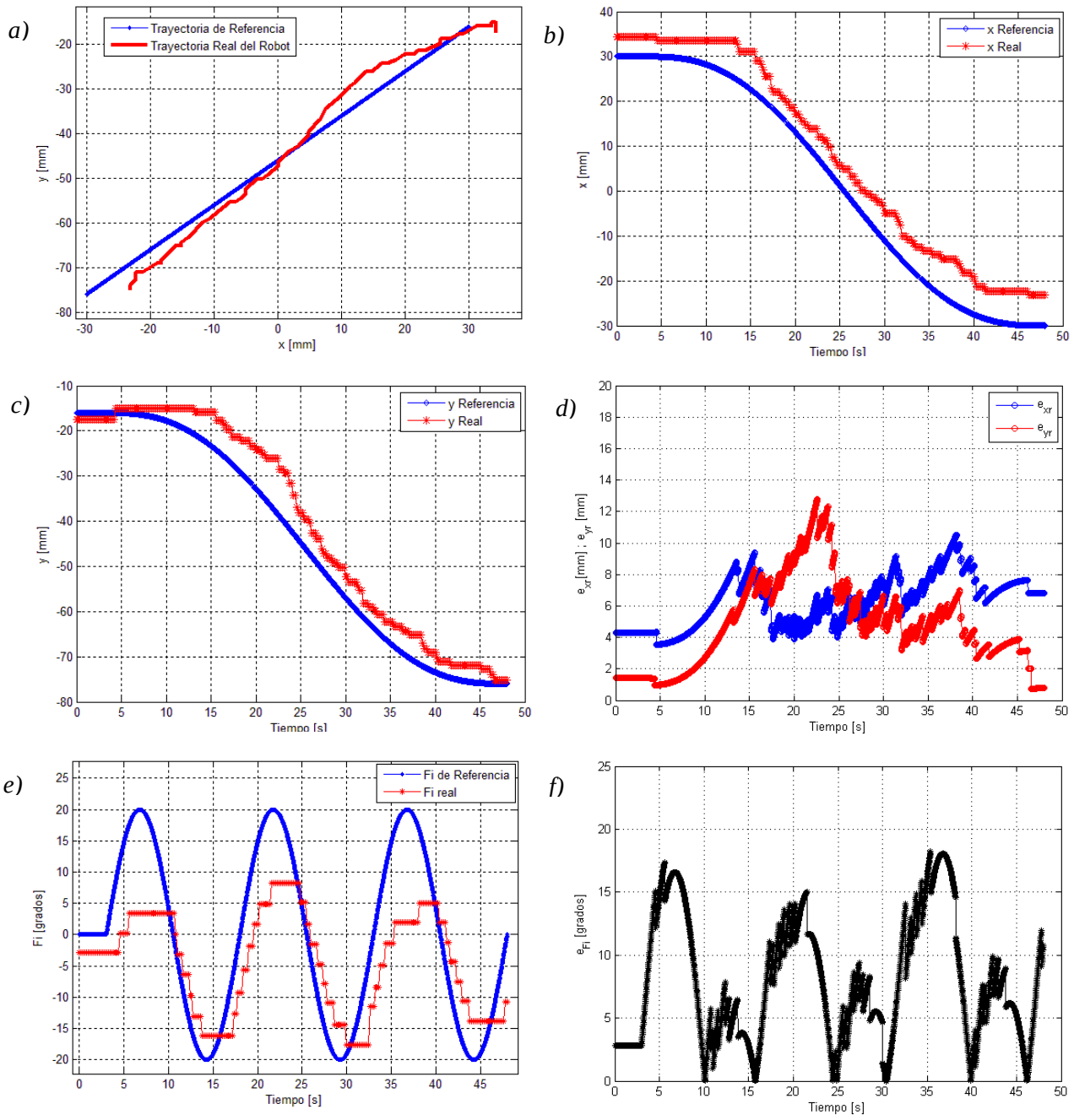


Figura 4.5: *Prueba 2, Línea recta diagonal PP LA,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* x *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* y *con respecto al tiempo, deseado y real.* ***d)*** *Error absoluto del desplazamiento en el eje* x *y en el eje* y *.****e)*** *Rotación de la PM deseado y real.* ***f)*** *Error absoluto de la rotación.*

Como podemos observar en la **Figura 4.5 (a)**, la PM no pudo mantener la linealidad de la recta al inicio, esto debido a la rotación. Cabe destacar que gracias al perfil de trayectoria, al inicio del recorrido disminuyeron los errores comparado con la prueba 1 entre 1 a 3[mm], costándole menos romper la inercia del robot. Los errores de posición tanto en el eje x como en el eje y oscilaron más **Figura 4.5 (d)**, manteniendo en algunos puntos los errores máximos similares a la prueba 1, pero los errores no fueron tan constantes durante la trayectoria, logrando errores mínimos de 4 a 1[mm].

En cuanto a la rotación de la PM, no alcanzo a girar hasta el ángulo establecido y hubo un poco de retraso en el caso de la rotación cuando iba en sentido positivo $+\phi$, en estas circunstancias se obtuvieron errores muy grandes entre 18 a 19 grados, en cambio cuando la rotación era en sentido negativo $-\phi$ los errores disminuían, teniendo errores máximos de 10 grados, posiblemente esto sea consecuencia de la configuración del robot, el cual utilizamos una configuración codo abajo, dicha postura del brazo se le complica realizar esta rotación positiva.

Si nos imaginamos el movimiento de la rotación en sentido positivo (antihorario), es como si los brazos del manipulador estuvieran desenroscando una tapa, por lo que dicha postura en los brazos no le favorece el torque. Sin embargo, si la rotación es en sentido negativo (horario), es como si enroscáramos la tapa, donde la postura de los brazos del manipulador logran tener un mayor torque entre sí.

4.3.3 Prueba 3 en lazo cerrado

Una vez que hemos visto el comportamiento del manipulador en lazo abierto, ahora el objetivo principal es utilizar la información del sistema de visión para corregir los errores de posición en $\{x, y\}$.

En este caso se requirio colocar al robot cerca del punto de inicio para evitar que el control no oscilara demasiado y que el robot se estabilizara rapidamente. Esto fue porque durante en los experimentos se observo que el sistema de vision no le daba tiempo de capturar la posición del fiducial dado a la continua oscilación de la PM, provocando que el control no se se estabilizara nunca. Por lo cual, para evitar este problema se propuso un tiempo de espera de dos segundos, así la PM lograría estabilizarse y colocarce en su posición inicial.

En la **Figura 4.6** se muestra la secuencia real del MDP realizando este movimeinto y en la **Figura 4.7** se muestran los resultados obtenidos de la prueba.

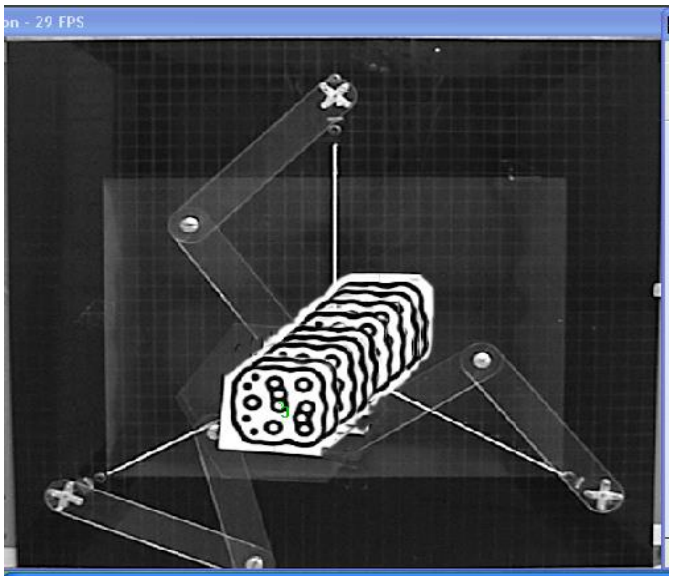
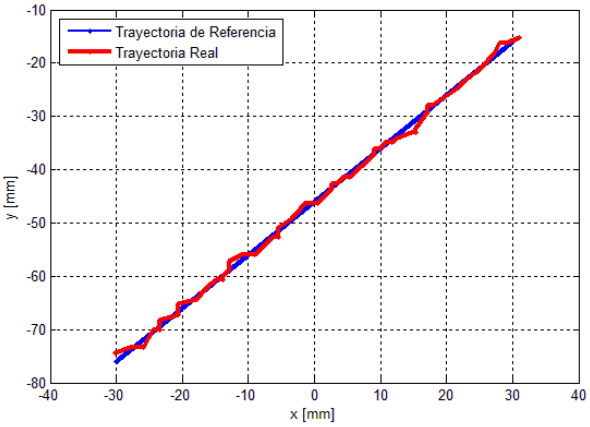


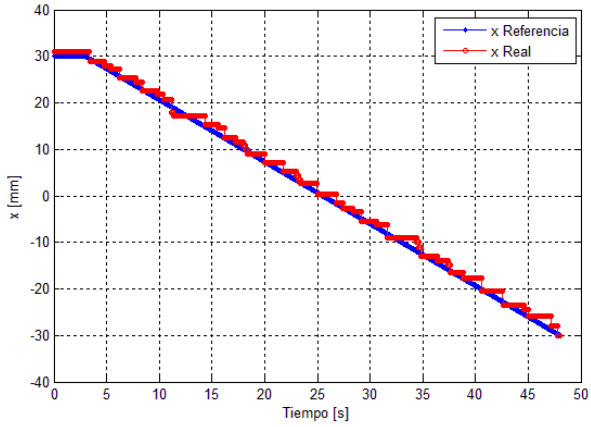
Figura 4.6: *Imagen secuencial de la prueba sobre el modelo funcional.*

Línea Recta Diagonal (LRD SS) LC

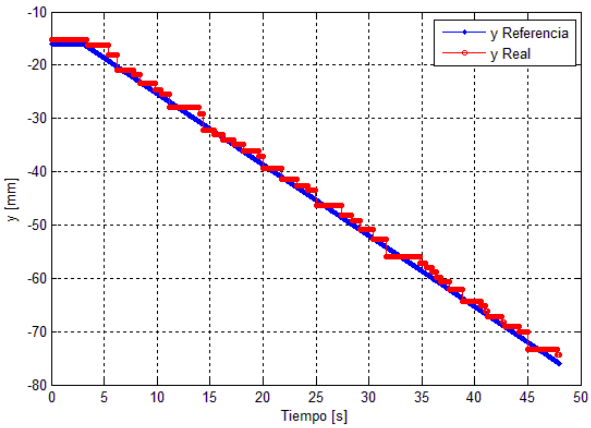
a)



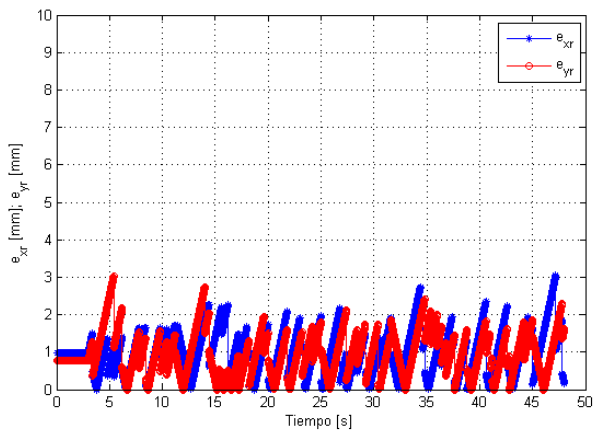
b)



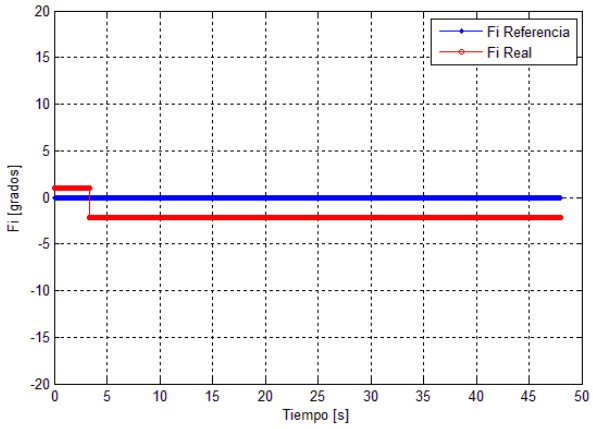
c)



d)



e)



f)

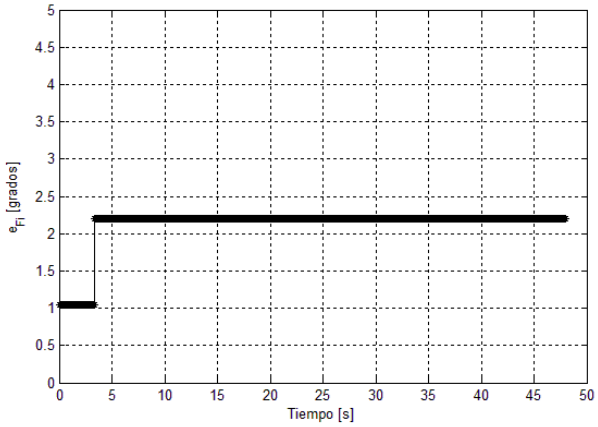


Figura 4.7: *Prueba 3, Línea recta diagonal SS LC,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* $\boldsymbol{x}$ *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* $\boldsymbol{y}$ *con respecto al tiempo, deseado y real.* ***d)*** *Error absoluto del desplazamiento en el eje* $\boldsymbol{x}$ *y en el eje* $\boldsymbol{y}$ *.****e)*** *Rotación de la PM deseado y real.* ***f)*** *Error absoluto de la rotación.*

Con los resultados de la **Figura 4.7**, se demuestra que hubo un gran mejoramiento en cuanto al seguimiento de la trayectoria, si se compara con la prueba 1. El desempeño fue aceptable, aunque con pequeños picos durante su trayecto pero iniciando y terminando en su posición deseada con errores mínimos. Dichos errores oscilan entre un máximo y un mínimo de 3 a 0 [mm] respectivamente. En general, la variación del error fue casi repetitiva, ocasionado principalmente por el controlador.

La discontinuidad en los desplazamientos de (x, y) de la **Figura 4.7 (b, c),** es algo importante a tomar en cuenta, porque las causas de dicho problema pueden ser varias, como vemos hay puntos donde la posición tanto en x como en y permanecen constantes.

Lo anterior se debe, a que mientras la trayectoria de referencia cambia de posición constantemente con respecto al tiempo, el controlador también corrige el error de posición constantemente, sin embargo los servos al corregir su posición angular, no tienen la suficiente resolución mínima para mover el eslabón o no son capaces de romper la inercia de la PM y por consiguiente se va acumulando el error de posición de la plataforma. Por eso, podemos ver en la gráfica del error que existen picos máximos de 3[mm] y cuando el controlador logra tener una corrección significativa para el actuador, este logra moverse. No obstante, puede que el actuador acumule demasiada energía y sobrepase la posición requerida, dando como resultado errores muy grandes en la PM. En la orientación mantuvo un error constante de aproximadamente 2.5 grados, posiblemente debido al huelgo entre las juntas.

4.3.4 Prueba 4 en lazo cerrado

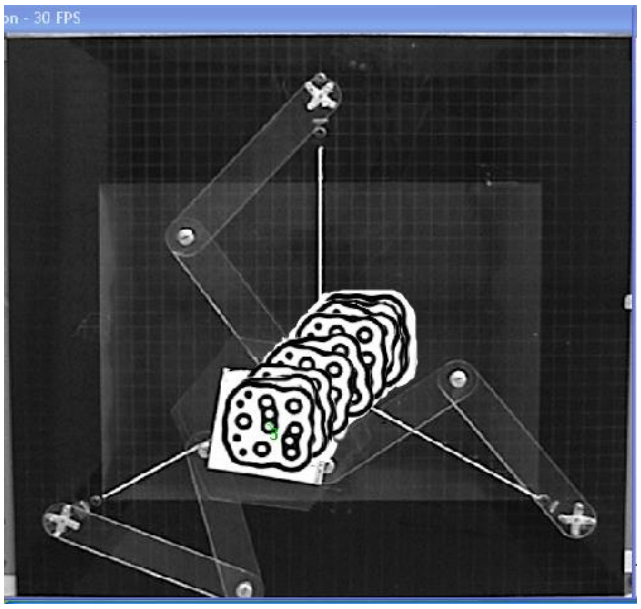
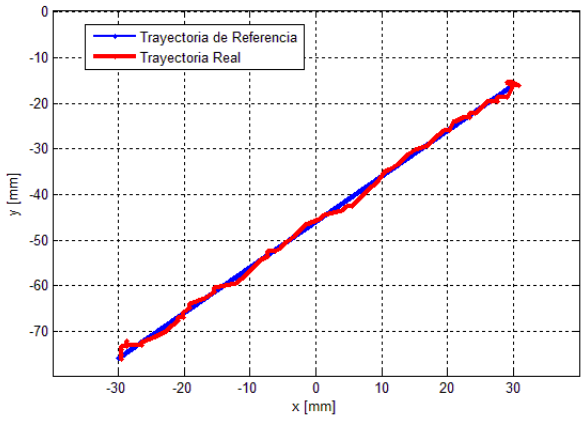


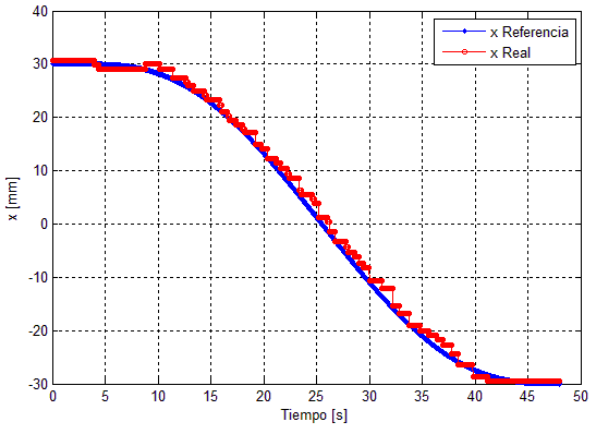
Figura 4.8: *Secuencia del modelo funcional describiendo el movimiento de la línea recta PP*

Línea Recta Diagonal (LRD PP) **LC**

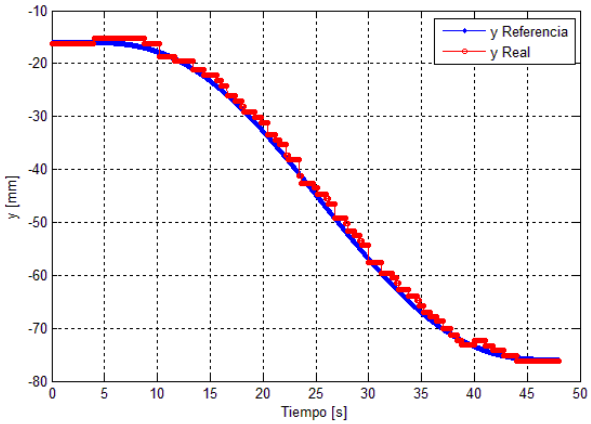
a)



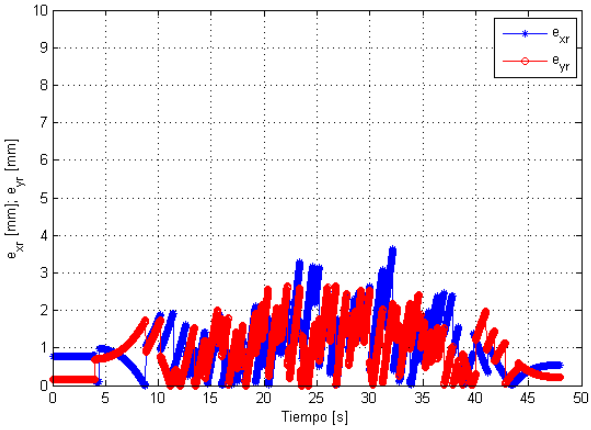
b)



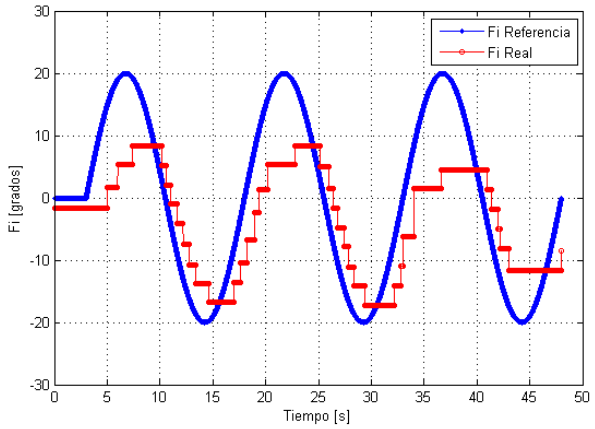
c)



d)



e)



f)

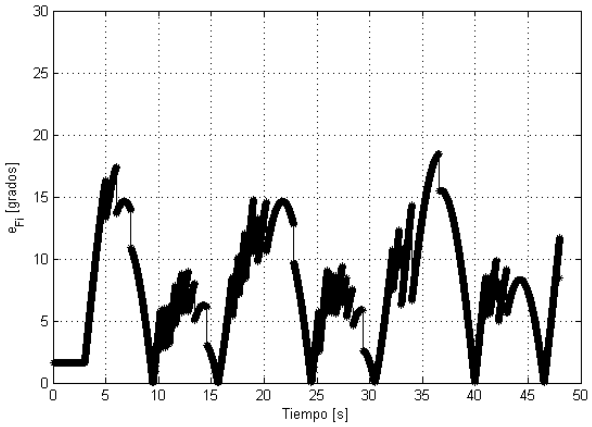


Figura 4.9: *Prueba 4, Línea recta diagonal PP LC,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* x *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* y *con respecto al tiempo, deseado y real.* ***d)*** *Error absoluto del desplazamiento en el eje* x *y en el eje* y *.****e)*** *Rotación de la PM deseado y real.* ***f)*** *Error absoluto de la rotación.*

Los movimientos de esta prueba es uno de los más complejos, como ya se había mencionado anteriormente en la prueba de lazo abierto. Podemos ver, que la información del sistema de visión ayudo considerablemente para corregir los errores de posición, en lazo abierto se obtuvieron errores muy grandes, en cambio el error máximo en esta prueba fue de aproximadamente 4[mm]. Dado a que el movimiento a realizar es más complejo, era de esperarse que el error tendiera a aumentar en comparación con la prueba 3. No obstante, aumento muy poco, cerca de 1 a 2 [mm] en algunos puntos y otros disminuyo el error.

En cuanto al perfil de trayectoria, ayudó al inicio y al final del recorrido a reducir los errores. En cambio, el error en la orientación mantuvo las mismas condiciones que en las prueba en lazo abierto.

4.4 Seguimiento de una trayectoria circular

Para las pruebas del seguimiento de una trayectoria circular, el diámetro del circulo que describe es de 60[mm], el fin de observar dicho desplazamiento es para ver el movimiento la plataforma móvil en una trayectoria no lineal, como ya se había mencionado anteriormente. El tiempo para su recorrido es de 45[s] (tiempo de simulink). Las ecuaciones de la trayectoria parametrizadas en función del tiempo son las siguientes:

$$x = r * cos(2\pi * t)$$
$$y = r * sin(2\pi * t) - 46 \qquad r = 30\ [mm]$$

4.4.1 Prueba 5 en lazo abierto

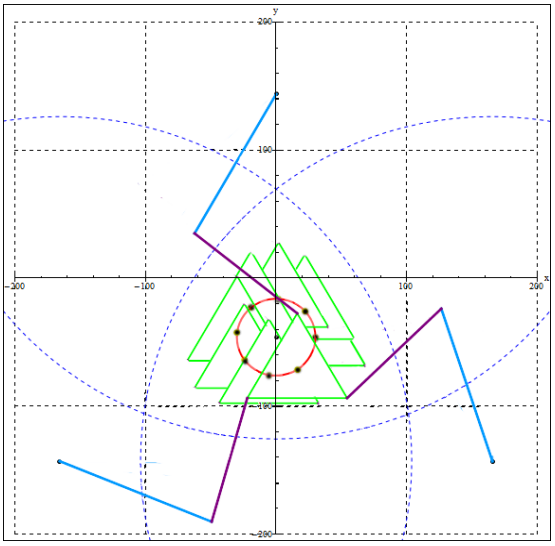
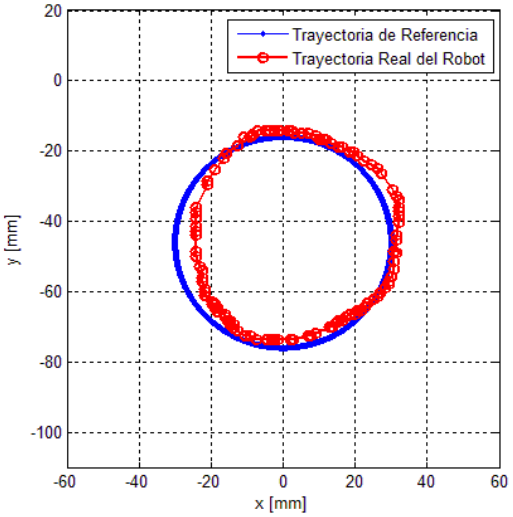


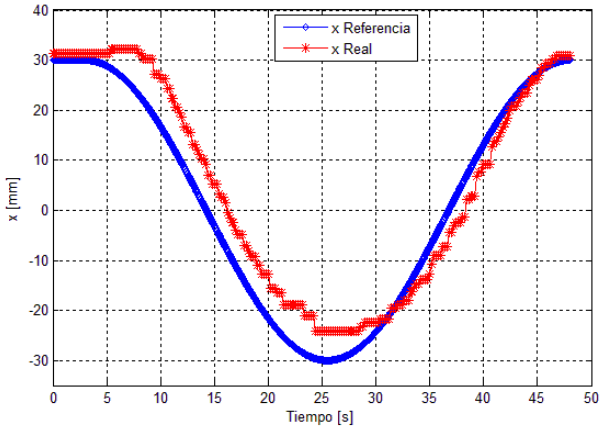
Figura 4.10: *Simulación de la PM describiendo el circulo SS*

Circulo (C SS) LA

a)



b)



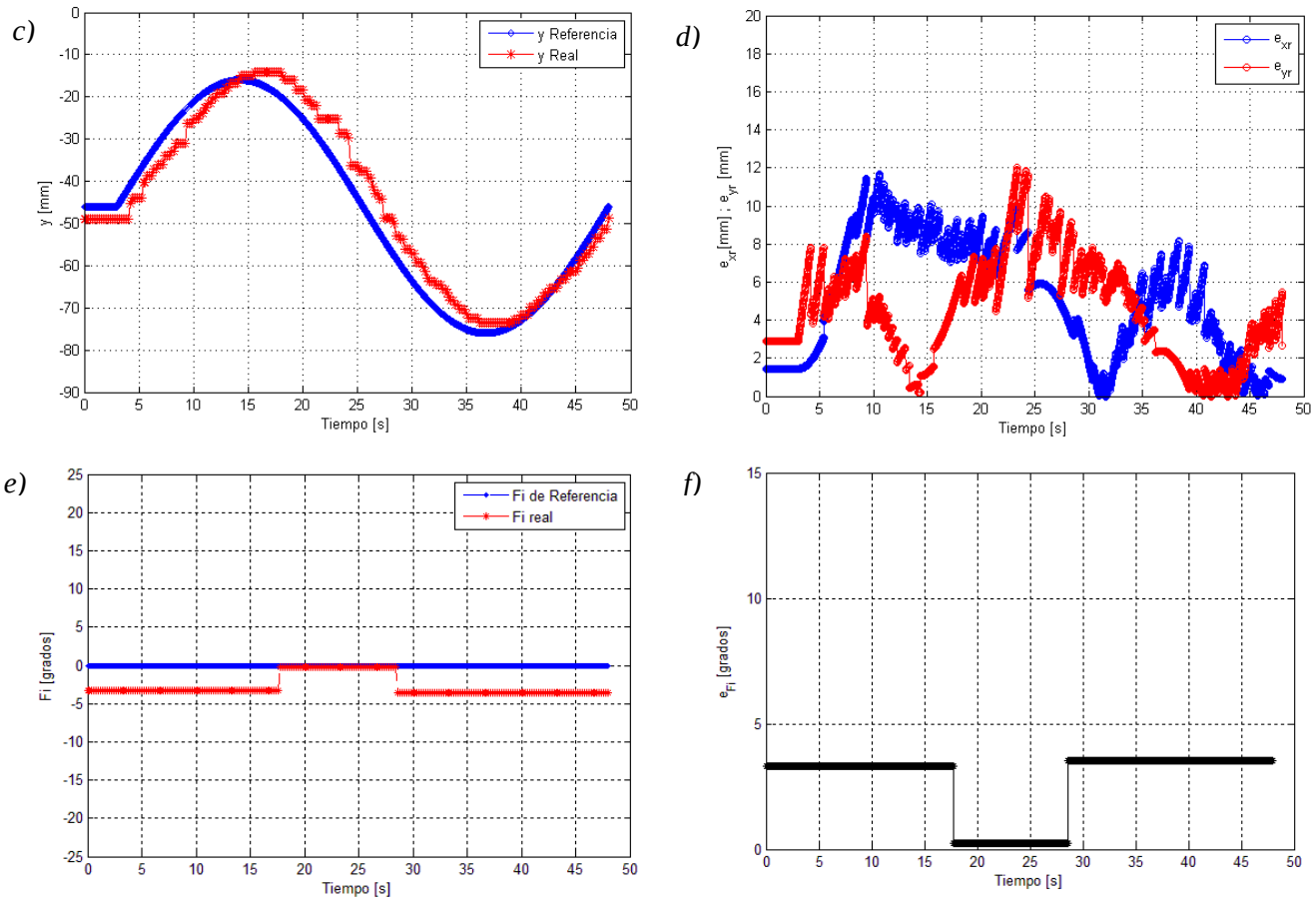


Figura 4.11: *Prueba 5, Circulo SS LA,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje x con respecto al tiempo, esperado y real .****c)*** *Desplazamiento en el eje y con respecto al tiempo, esperado y real.* ***d)*** *Error absoluto del desplazamiento en el eje x y en el eje y .****e)*** *Rotación de la PM esperado y real.* ***f)*** *Error absoluto de la rotación.*

Como era de esperarse, para este movimiento en lazo abierto tuvo muchos errores. En el caso del desplazamiento en el eje x **Figura 4.11 (b)**, tuvo un error casi constante hasta la mitad de su recorrido, tomando en cuenta que comenzó en las coordenadas (0, -46) con un movimiento en sentido contrario a las manecillas del reloj. Dicho error posiblemente se debió a problemas de los eslabones en el modelo funcional, causado por fricción o por la inercia.

En este caso, el movimiento realizado tuvo los errores máximos cuando la plataforma móvil se desplazaba en el sentido de su propio eje, por ejemplo en las gráficas de desplazamiento de x y y, se puede observar en las curvas como aumentaba el error cuando la posición cambiaba continuamente, por lo que, los errores para una trayectoria circular alcanzaron a tener un máximo de 12[mm]. La orientación tuvo un ligero error de aproximadamente 4 grados.

4.4.2 Prueba 6 en lazo abierto

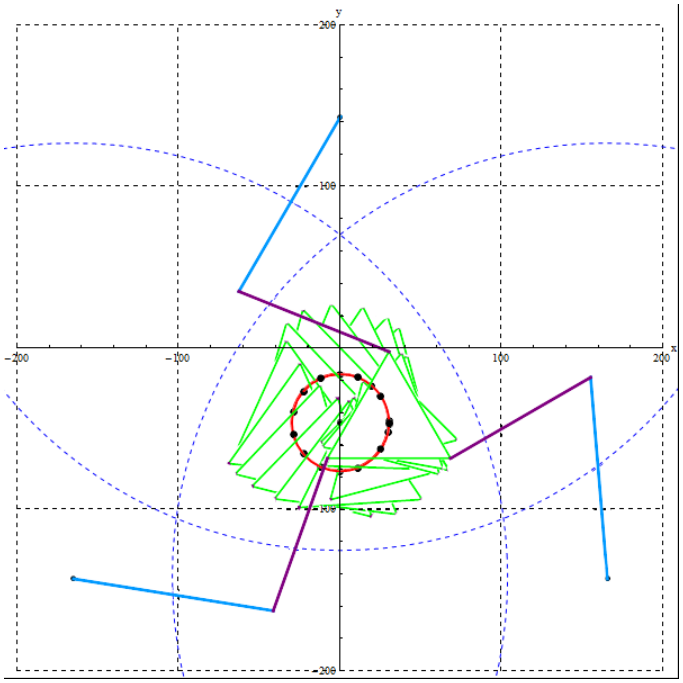
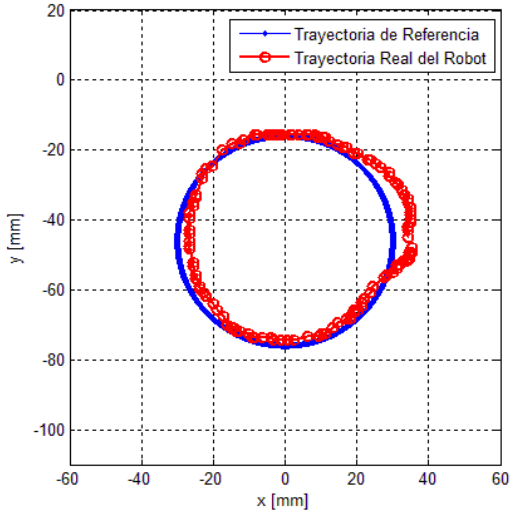


Figura 4.12: *Simulación del movimiento de la PM describiendo la trayectoria circular PP*

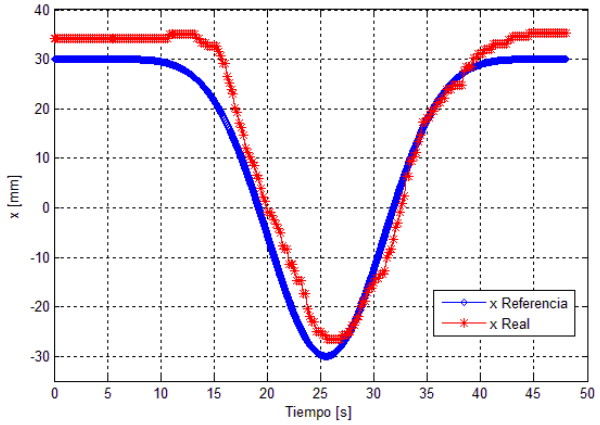
En la **Figura 4.12** muestra claramente la simulación del desplazamiento de la PM para dicha trayectoria, incluyendo la rotación (penduleo) y el perfil de trayectoria. En la **Figura 4.13**, se muestran los resultados de la prueba sobre el modelo funcional.

Circulo (C PP) LA

a)



b)



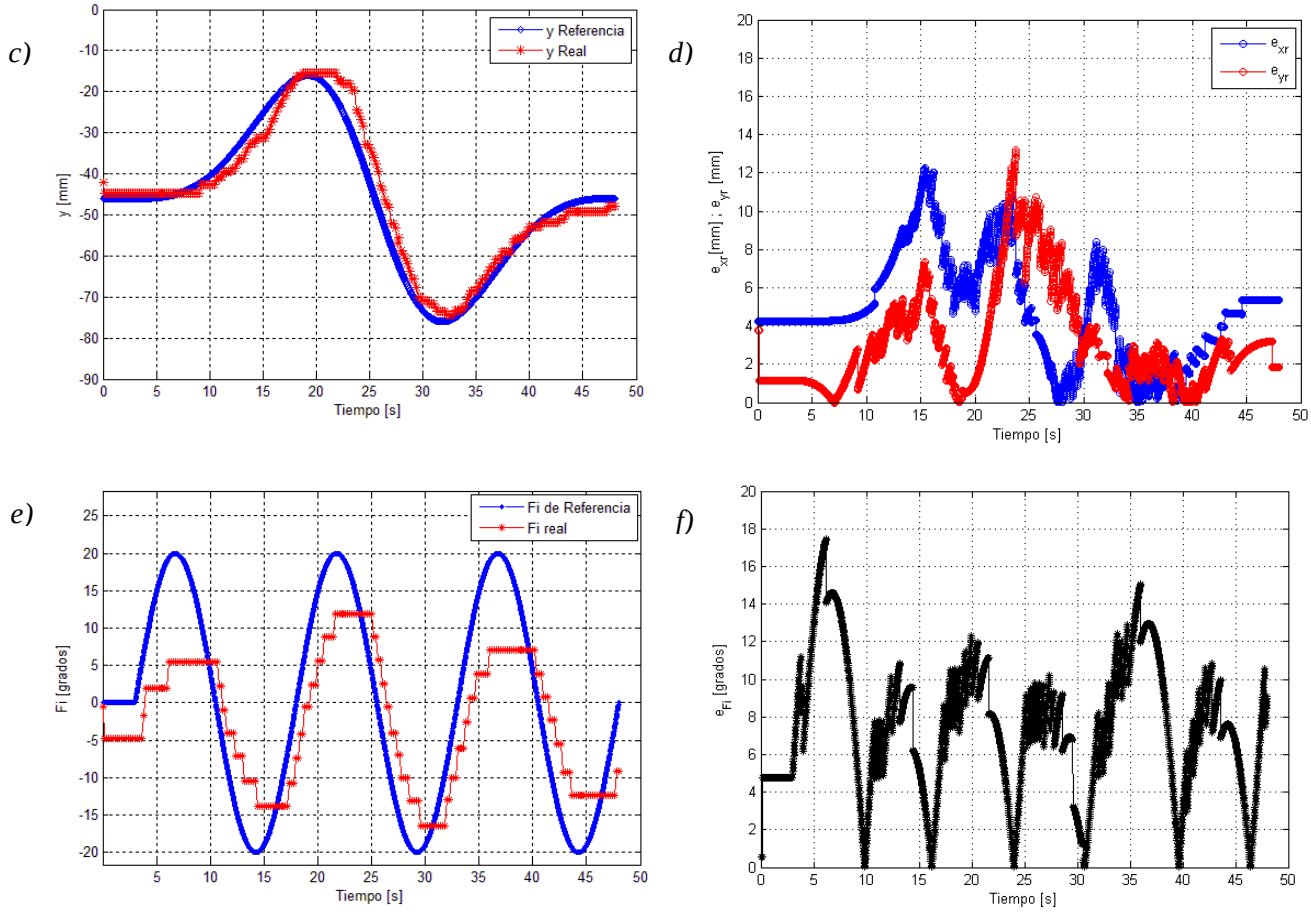


Figura 4.13: *Prueba 6, Circulo PP LA,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje x con respecto al tiempo, esperado y real .****c)*** *Desplazamiento en el eje y con respecto al tiempo, esperado y real.* ***d)*** *Error absoluto del desplazamiento en el eje x y en el eje y .****e)*** *Rotación de la PM esperado y real.* ***f)*** *Error absoluto de la rotación.*

En esta prueba es claro que aumentaron un poco los errores de posición, en ambos desplazamientos (x, y) su error máximo llego casi a los 13 [mm] aumentando tan solo un milímetro en comparación con la anterior prueba, lo cual se esperaban errores más grandes por la dificultad del movimiento.

Algo diferente en comparación con la prueba de la línea recta con el perfil de trayectoria, es que en este caso no mejoro mucho al incluir el perfil de trayectoria, ya que al inicio y al final del recorrido se puede apreciar el desfasamiento que hizo el centro de la PM, mostrado en la **Figura 4.11 (a)**.

En cuanto a la orientación de la PM los errores siguen teniendo el mismo comportamiento que las pruebas de la línea recta. Aunque es claro, que aumentaron unos milímetros en el sentido negativo de la rotación de la PM.

4.4.3 Prueba 7 en lazo cerrado

Para la prueba del seguimiento de la trayectoria circular, al igual que las pruebas anteriores, se mejora el desempeño del robot utilizando el sistema de visión y el método para controlar el error de posición.

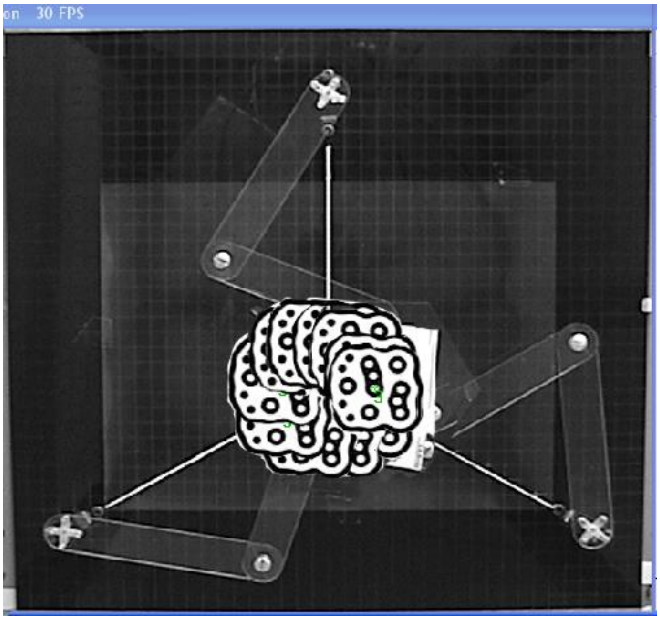
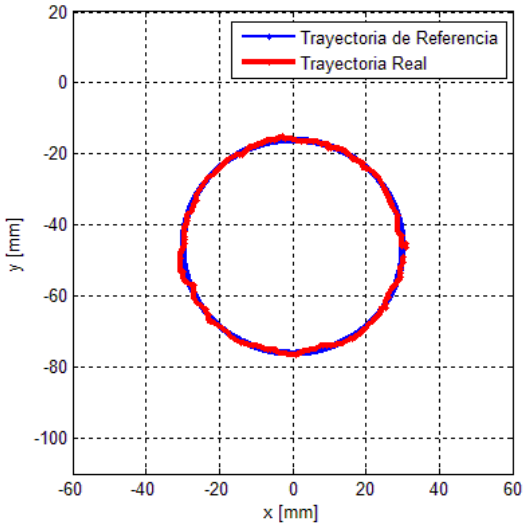


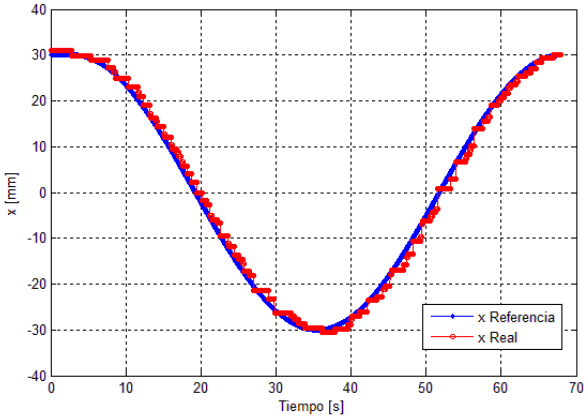
Figura 4.14: *Secuencia del modelo funcional describiendo el movimiento circular SS*

Circulo (C SS) LC

a)



b)



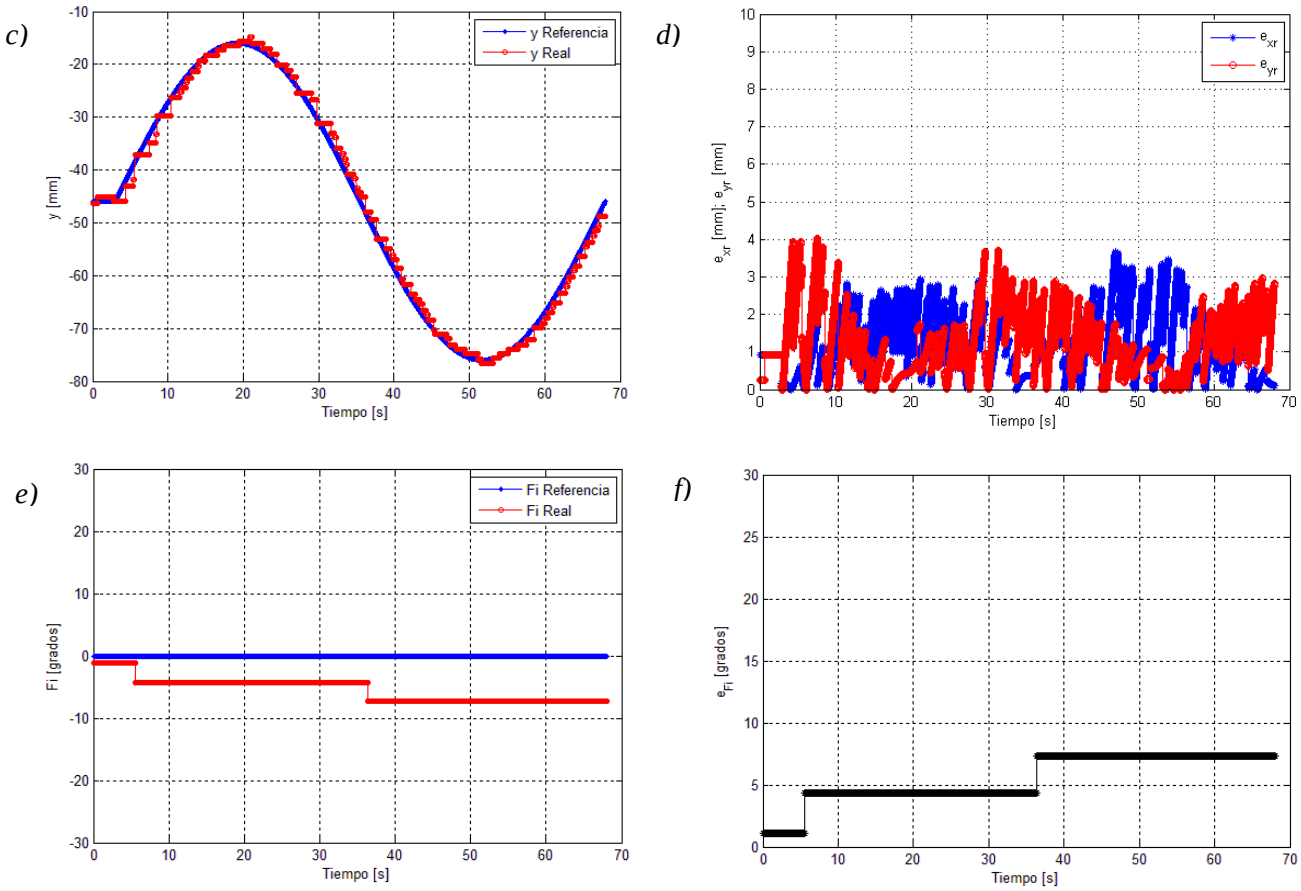


Figura 4.15: *Prueba 7, Circulo SS LC,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje x con respecto al tiempo, esperado y real .****c)*** *Desplazamiento en el eje y con respecto al tiempo, esperado y real.* ***d)*** *Error absoluto del desplazamiento en el eje x y en el eje y .****e)*** *Rotación de la PM esperado y real.* ***f)*** *Error absoluto de la rotación.*

Es claro que la plataforma móvil es capaz de corregir los errores con la ayuda del sistema de visión y el método para controlar dichas variables estando en trayectorias no lineales. Durante su trayecto, logro reducir considerablemente los errores de posición llegado a tener errores de 4[mm] como máximo, si lo comparamos con el movimiento en lazo abierto la corrección es aceptable.

La orientación como en las demás pruebas mantuvo los mismos errores, concluyendo que esto se debe a los problemas del modelo funcional y a los actuadores, como se ha mencionado anteriormente.

4.4.4 Prueba 8 en lazo cerrado

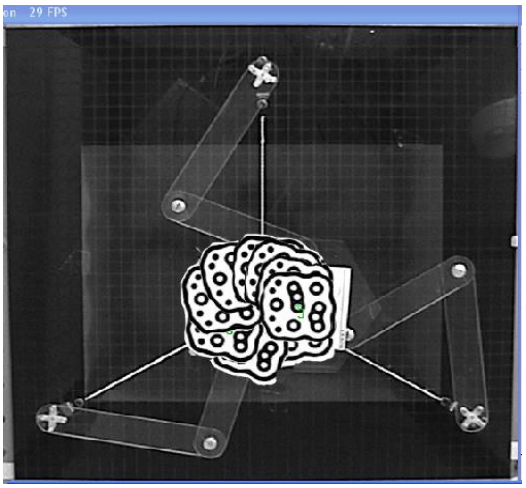
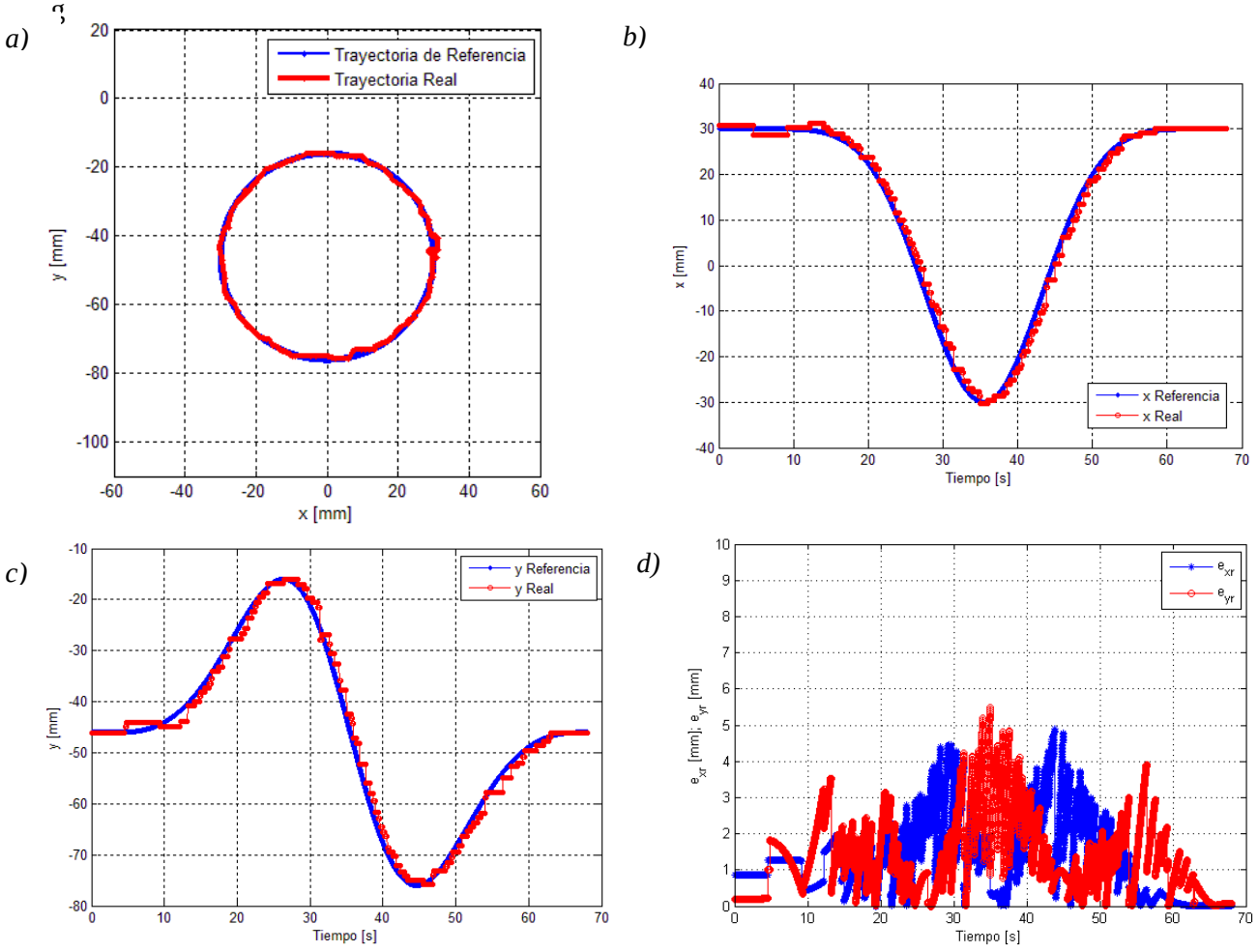
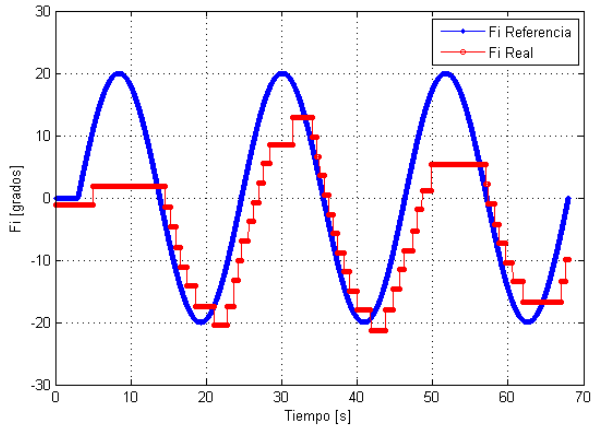


Figura 4.16: *Secuencia del modelo funcional describiendo el movimiento circular PP.*

Circulo (C PP) LC



e)



f)

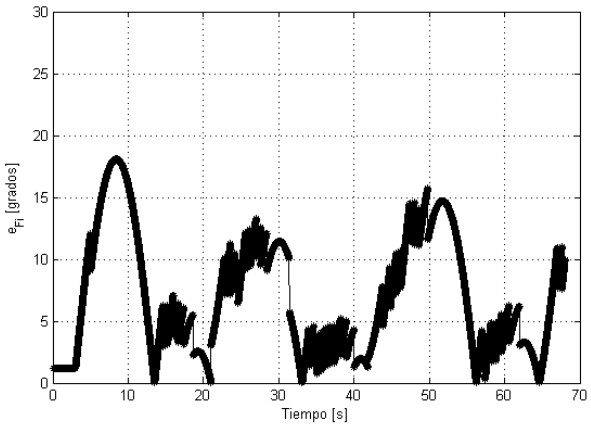


Figura 4.17: *Prueba 8, Circulo PP LC,* ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje x con respecto al tiempo, esperado y real .****c)*** *Desplazamiento en el eje y con respecto al tiempo, esperado y real.* ***d)*** *Error absoluto del desplazamiento en el eje x y en el eje y .****e)*** *Rotación de la PM esperado y real.* ***f)*** *Error absoluto de la rotación.*

Con esta prueba siendo una de las más completas por el conjunto de movimientos que debe realizar la PM, los resultados nos muestran la efectividad tanto del sistema de visión como el método para controlar las variables del proceso. En cuanto al seguimiento de la trayectoria la PM lo siguió muy bien, los errores más grandes se debieron al perfil de trayectoria, ya que a la mitad de su recorrido aumenta la velocidad por la evolución del perfil de trayectoria. En la orientación, los errores mantuvieron el mismo patrón que las pruebas anteriores, aunque al inicio y al final le costó más trabajo orientarse.

Cabe mencionar que en todas las pruebas de lazo cerrado hubo un retraso en el tiempo de simulación y el tiempo real del modelo funcional, mientras que el tiempo de simulación era de 45 [s] en tiempo real al robot le tomaba aproximadamente de 3 a 4 [min] en realizar la trayectoria.

En el **Apéndice A**, se muestran los resultados de otras trayectorias realizadas por el modelo funcional, entre ellas está una línea horizontal, vertical y un ocho.

Conclusiones y Trabajo a Futuro

Conclusiones

Se implementó un sistema de visión sobre un MDP 3RRR mediante la configuración "eye-at-hand" para el seguimiento de trayectorias. Con la implementación del sistema de visión el modelo funcional redujo el error de posición de la plataforma móvil, realizando las trayectorias casi cercanas a la referencia. Cabe destacar que en las pruebas con movimientos complejos hubo errores menores a cinco milímetros.

El sistema de visión cumplió con su función, se obtuvo la posición y orientación del robot, se pudo controlar la posición, redujo la cantidad de sensores y la información proporcionada fue fácil de manipular. Aunque, se concluyen varias características en cuanto a su funcionamiento, en primera instancia, la resolución mínima que alcanza a detectar la orientación de un fiducial es de aproximadamente tres grados. Este inconveniente evito reducir el error en la orientación (ϕ) de la plataforma móvil, ya que con esta resolución causaba inestabilidad en el robot debido a que no convergía al valor deseado. Por lo tanto, el sistema de visión no permite tener una resolución menor a los tres grados, esto genera un problema para aplicaciones donde requiera precisión en esta variable.

Por otro lado, el sistema fue capaz de detectar movimientos mínimos de aproximadamente 0.7 a 1 [mm] en la resolución de los ejes (x, y), esto con base al acercamiento que tenía la cámara y el robot. La resolución permitió aplicarle el control a las variables (x, y).

En las pruebas hubo un retraso entre el tiempo de la simulación y el tiempo del modelo funcional al ejecutar las trayectorias. Por un lado, este retraso de tiempos se debió a la cantidad de cálculos realizados por el sistema de control y el sistema de visión (procesamiento de imágenes) al ser procesados por una sola computadora y por el otro, posiblemente fue ocasionado por la comunicación entre ReacTIVision y Simulink, ya que existia un retraso en el envió de los datos. Para evitar esto, se recomendaría utilizar dos sistemas de procesamiento (dos computadoras), uno para la visión y otro para los cálculos.

En la obtención de los datos entre el puerto de comunicación de ReacTIVision a Simulink se pudo tener una velocidad de muestreo de cinco muestras (datos de visión) por segundo (tiempo de Simulink). No se logró tener más muestras por segundo debido a que el tiempo

del sistema de procesamiento aumentaba así como también el tiempo de realización de la trayectoria en el modelo funcional. En el caso contrario, al disminuir las muestras por segundo el tiempo de procesamiento disminuía pero se corría el riesgo de perder información de alguna variable, esto afectaba directamente al control y la desestabilización del robot. Por lo tanto, una mejora sería la solución del párrafo anterior y también sería conveniente probar una cámara con mayor cantidad de frames por segundo (fps) posiblemente aumentaría la velocidad de muestreo.

Con las pruebas se pudo observar físicamente los problemas del modelo funcional, uno de los problemas más notables fue el movimiento de rotación de la plataforma móvil, el cual, dicha relación de dimensiones entre los eslabones impedía que girara correctamente la plataforma móvil, generando de esta manera los errores en la orientación. Aunado a esto, el huelgo entre las juntas y el tipo de material pudieron haber sido también causa de dicho problema.

Uno de los elementos importantes fueron los actuadores, en este caso los servomotores nos ofrecieron su fácil uso y su rápida controlabilidad, aunque en este caso se requirió de un mayor torque para moverlo en posiciones muy pequeñas, ya que los que se utilizaron no tenían esa capacidad, por lo que carecían de resoluciones muy pequeñas. Por otro lado, se percató que estos servomotores no tenían mucha precisión puesto que se tuvo que caracterizar a los servos para tener su rango completo y reducir sus errores de posición angular.

El control PI utilizado funciono acorde al robot, la respuesta del control era rápida de acuerdo a las condiciones del proceso, un poco inestable en algunos puntos y los errores fueron mínimos. Tanto las ganancias de la acción proporcional e integral fueron con base en los experimentos, no se incluyó la acción derivativa porque tiene la propiedad de entregar una señal proporcional a la velocidad de cambio de la señal del error, el cual, el sistema de visión no le daba tiempo de capturar las nuevas posiciones del fuducial teniendo como resultado inestabilidad en el robot.

Finalmente, con las ventajas y desventajas presentes en este trabajo, podemos concluir que efectivamente el sistema de visión ayudó a reducir los errores de posición del efector final del MDP 3RRR. Esto claramente se ve reflejado en los resultados de las pruebas realizadas, comprobándolo con una variedad de pruebas en condiciones y movimientos que aprovechan las capacidades del robot.

La ventaja de tener un sistema de visión es que se puede reducir la cantidad de sensores, mejorar ciertos aspectos que se deseen en un robot y hasta darle cierta autonomía. No

obstante, el costo de tener un sistema de visión es muy grande, puesto que requiere de una alta capacidad de procesamiento y rapidez.

Trabajo a Futuro

Este proyecto mostró una forma más de corregir los errores de posición en un robot delta plano 3RRR con base a la información de un sistema de visión, desde el punto de vista cinemático. Posiblemente esto de lugar a trabajos posteriores, tomando en cuenta la dinámica y el desarrollo o búsqueda de nuevos sistemas de visión.

La velocidad del sistema de visión podría funcionar adecuadamente si los recursos de los sistemas de procesamiento mejoraran u ocupando dos computadoras, así como también el dispositivo de captura (videocámara), teniendo una mejor calidad de video y una mayor cantidad de fsp.

El modelo funcional por su parte, se le diseñaría primero, la relación de las dimensiones de los eslabones para tener una adecuada rotación en la plataforma móvil, esto se haría con base en la utilización de simulaciones para evitar la manufactura de varias piezas. Mejorar también la manufactura de las juntas, reduciendo el huelgo y la fricción, colocar servomotores con mayor toque o utilizar otro tipo de actuadores.

Diseñar un controlador con base al modelo del robot o probar otras teorías o métodos de control por ejemplo, redes neuronales, sistemas difusos, algoritmos genéticos o filtros de kalman.

Por otra parte, sería interesante probar esta técnica de visión en manipuladores que tienen movimiento en el espacio, ya que son los manipuladores más comunes en la industria y en las aplicaciones.

Referencias

Bibliografía

[1] Anup Kale, Zenon Chaczko, Imre Rudas. ***Parallel robot vision using genetic algorithm and object centroid. University of technology Sydney***. 15 Broadway.

[2] Kragic, D. Christensen, H. I. (2002) ***Survey on Visual Servoing for manipulation, Centre for Autonomus System, Numerical Analysis and Computer Science***. Citeseerx.

[3] Chaumette, F. Hutchinson S. (2006) ***Visual Servo Control, Part I: Basic Approaches," IEEE Robotics and Automation Magazine***. Págs. 82-90.

[4] Coste-Manière, E. Convignous, P. Khosla K. (1993) ***Robotic Contour Following based on Visual Servoing***. IEEE International Conference on Intelligent and Systems, Japan.

[5] García, G.J. Pomares, J. Gil, P. Torres, F. (2009) ***Seguimiento intemporal de trayectorias en la imagen basado en control visual dinámico***. XXX Jornada de Automática, Universidad de Alicate.

[6] Dean-Leon, E.C. Parra-Vega, V. et al. (2004) ***Dynamical image-based PID uncalibrated visual servoing fixed camera for tracking of planar robots with a heuristical predictor***. IEEE International Conference, Págs. 339-345.

[7] Trujano, M.A. Garrido, R. Soria, A. (2010) ***Stable visual servoing of an overactuated planar parallel robot***. IEEE Conference on Electrical Engineering Computing Science and Automatic Control, Págs. 182-187.

[8] Andreff, N. Martinet, P. ***Visual servoing of a Gough-Stewart Parallel Robot without proprioceptive sensor.*** Université Blaise Pascal/IFMA.

[9] Dallej, T. Andreff, N. Martinet, P. (2007) ***Image-Based Visual Servoing of the I4R parallel robot without proprioceptive sensor***. IEEE International Conference on Robotics and Automation, Italy.

[10] Traslosheros, A. Sebastian, et al. (2011) ***Visual servoing for the Robotenis system: A strategy for a 3 DOF parallel robot to hit a Ping-Pong ball***. IEEE Conference on Decision and Control. Págs. 5695-5701.

[11] Ollero Baturone A. ***Robótica: manipuladores y robots móviles.*** Ed. Alfaomega Marcobo, Barcelona, 2007.

[12] Lung-Wen Tsai. ***Robot Analysis: The mechanics of serial and parallel manipulators.*** Ed. John Wiley & Sons, USA, 1999.

[13] John J. Craig, ***Robótica.*** Ed. Pearson Pentrice Hall, 3ra. Edición, México, 2006.

[14] Yañez Valdez R. ***"Resolución de mecanismo paralelo planar 3RRR impulsado por actuadores eléctricos".*** IPN, Querétaro, 2007.

[15] Dabney, J. et al. ***SIMULINK 4***. Ed. Prentice Hall, New Jersey, 2001.

Fuentes electrónicas

[16] Última revisión 17/07/2012, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.167.1459 &rep=rep1&type=pdf

[17] Última revisión 18/07/2012, http://www.iwf.tu-bs.de/En/f+e/fawm/Allgemein.html

[18] Última revisión 22/06/2012, http://www.preciseautomation.com/VisualServoing.html

[19] Última revisión 22/06/2012, http://www.irisa.fr/lagadic/visp/visual-servoing.html

[20] Última revisión 19/07/2012, http://www.servodatabase.com/servo/vigor/vs-3

[21] Última revisión 5/08/2012, http://reactivision.sourceforge.net/

[22] Última revisión 5/08/2012, http://www.tuio.org/

[23] Última revisión 6/08/2012, http://www.arduino.cc/

[24] Última revisión 8/08/2012, http://www.mathworks.es/products/simulink/index.html

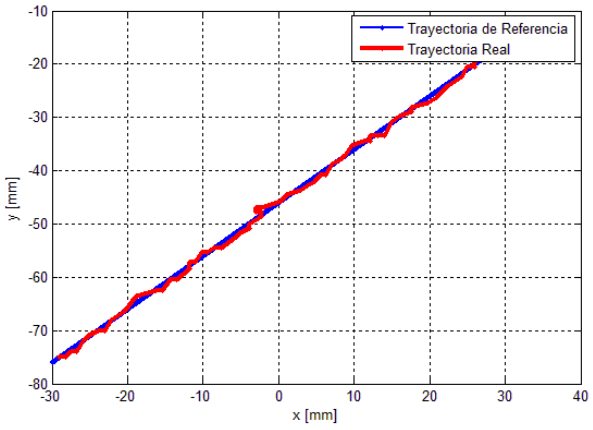
[25] Última revisión 8/08/2012, http://www.mathworks.es/products/simmechanics/index.html

Apéndices

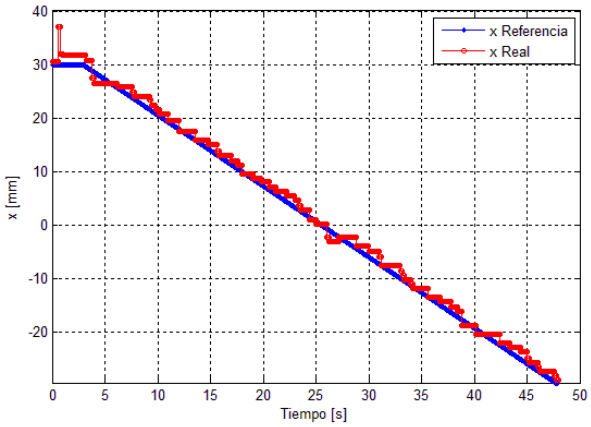
Apéndice A

1. Resultado de la Línea Recta Diagonal SP (sin perfil/con penduleo), LC.

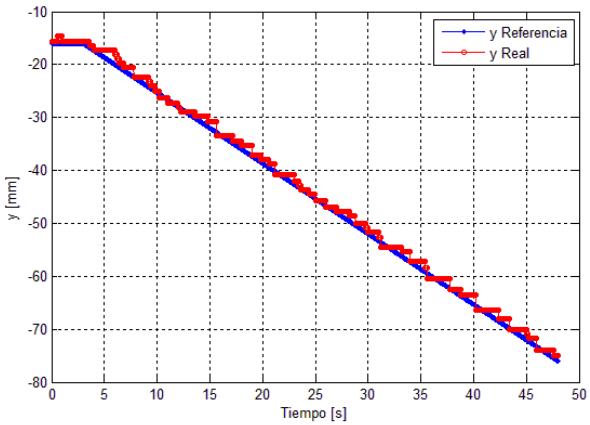
a)



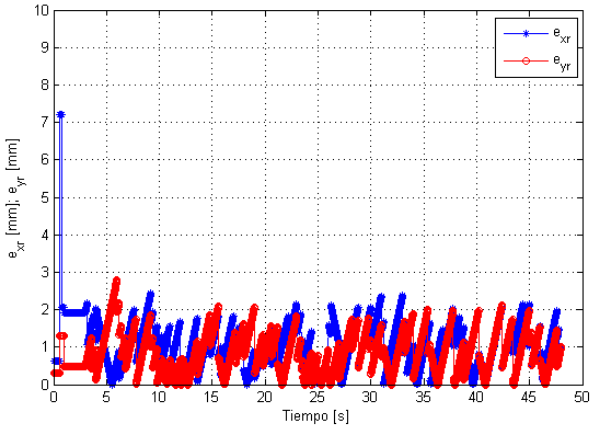
b)



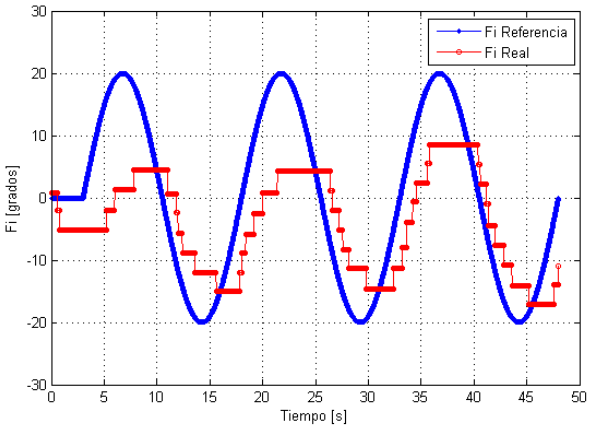
c)



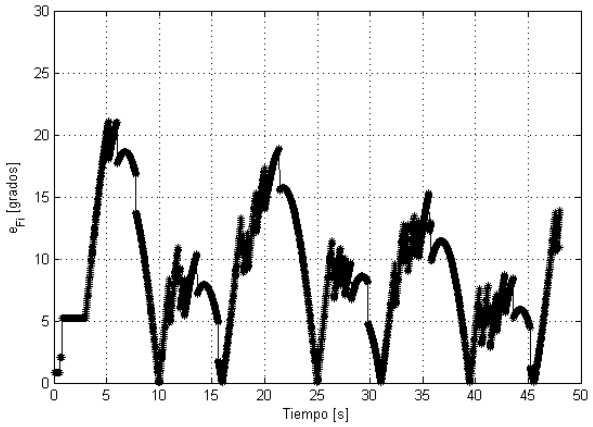
d)



e)



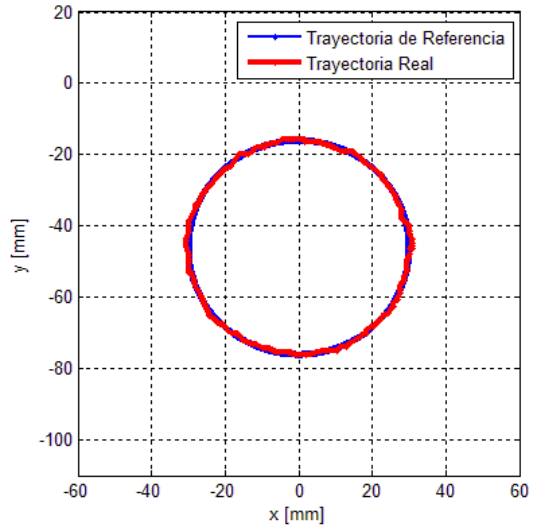
f)



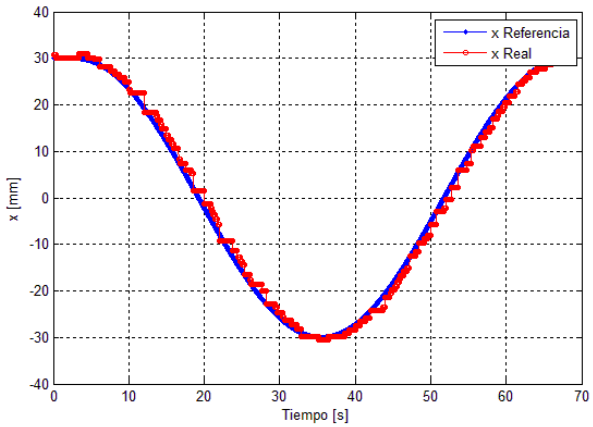
Línea recta diagonal SP LC, ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* ***x*** *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* ***y*** *con respecto al tiempo, deseado y real.* ***d)*** *Error absoluto del desplazamiento en el eje* ***x*** *y en el eje* ***y*** *.****e)*** *Rotación de la PM deseado y real.* ***f)*** *Error absoluto de la rotación.*

2. Resultado de la Trayectoria circular, SP, LC.

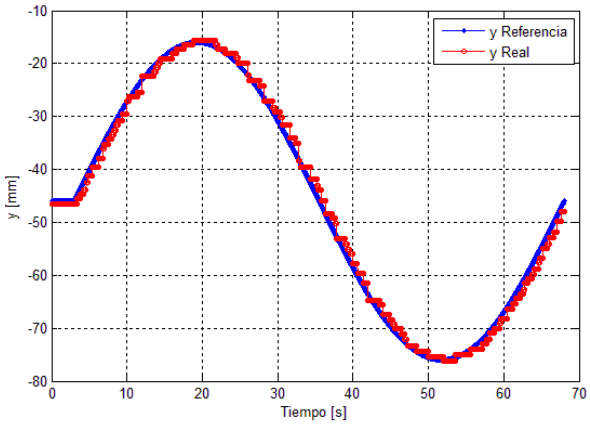
a)



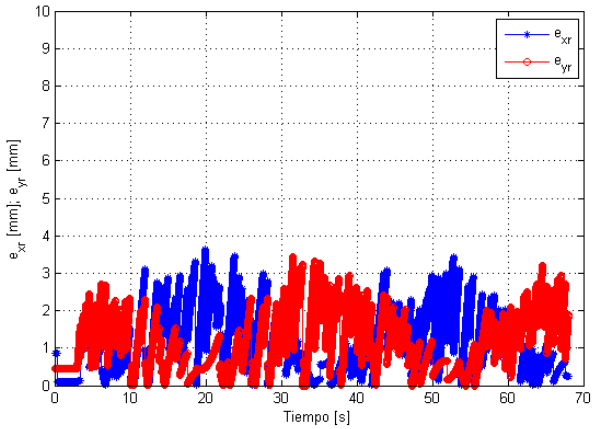
b)



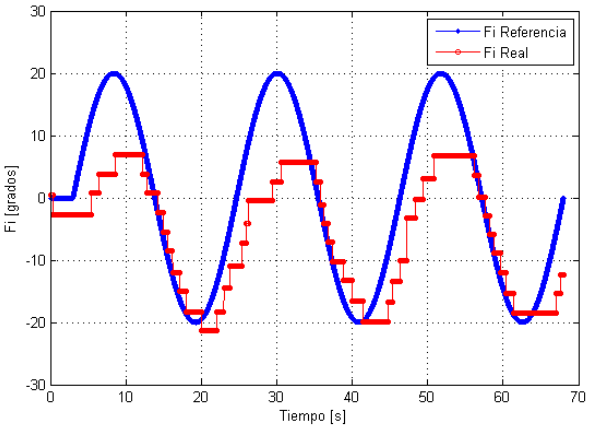
c)



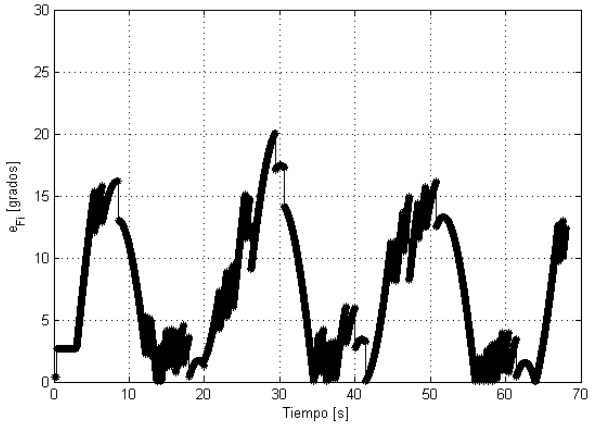
d)



e)



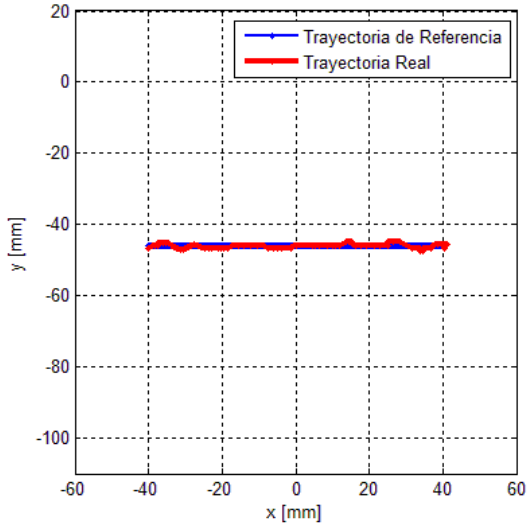
f)



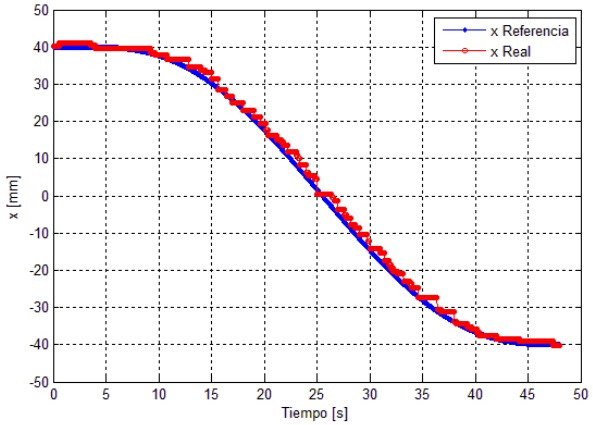
Trayectoria circular SP LC, ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* x *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* y *con respecto al tiempo, deseado y real.* ***d)*** *Error absoluto del desplazamiento en el eje* x *y en el eje* y *.****e)*** *Rotación de la PM deseado y real.* ***f)*** *Error absoluto de la rotación.*

3. Resultado de una Línea Recta Horizontal, PP, LC.

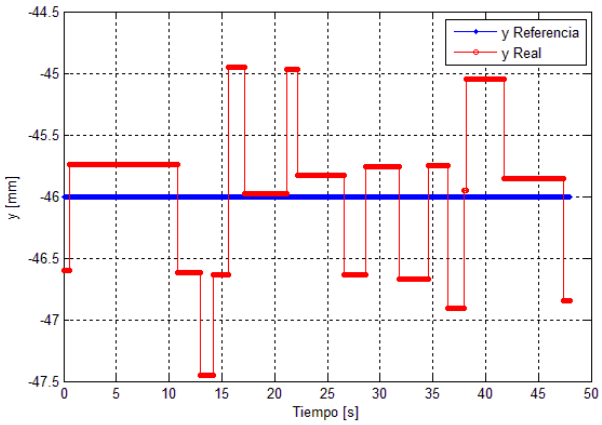
a)



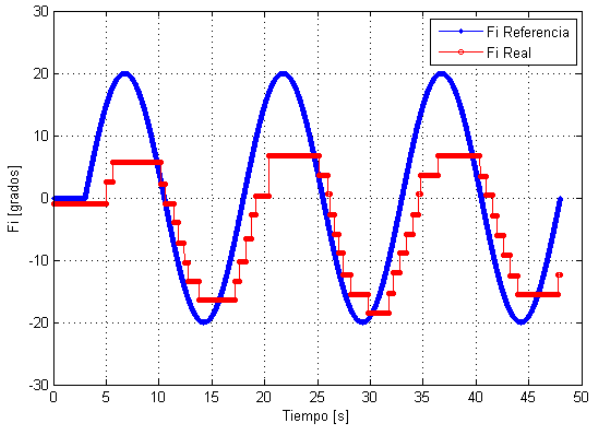
b)



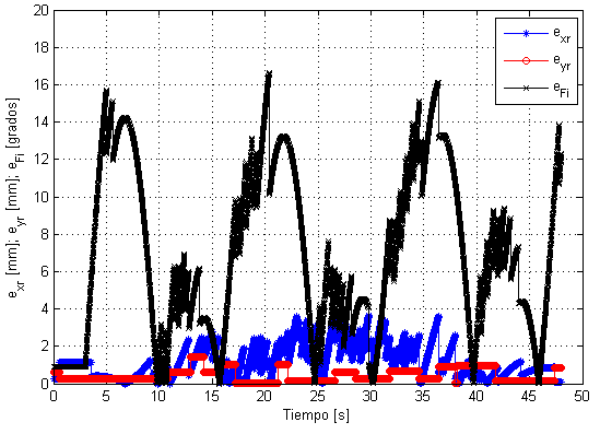
c)



d)



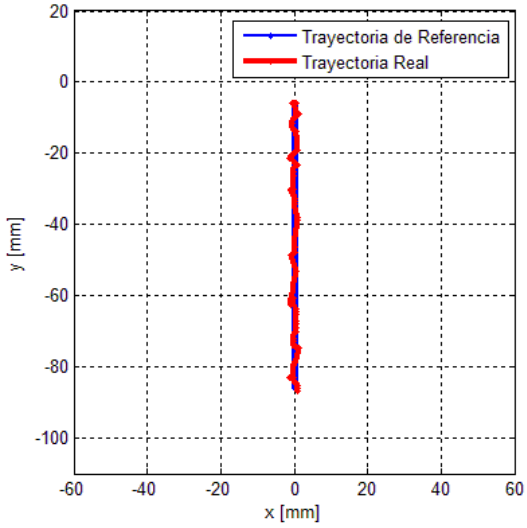
e)



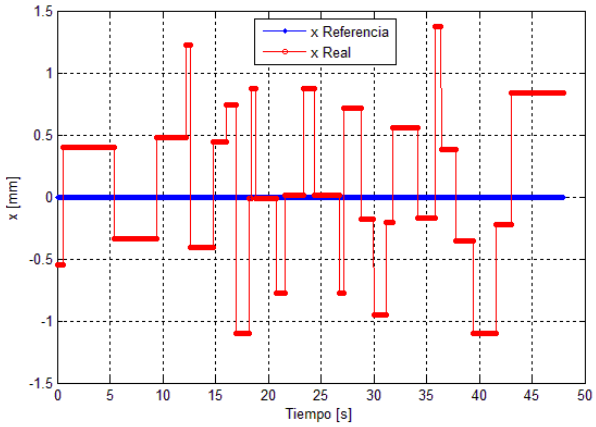
Línea Recta Horizontal PP LC, ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* x *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* y *con respecto al tiempo, deseado y real.* ***d)*** *Rotación de la PM deseado y real.* ***e)*** *Error absoluto del desplazamiento en* x*, en* y *y en* ϕ*.*

4. Resultado de una Línea Recta Vertical, PP, LC

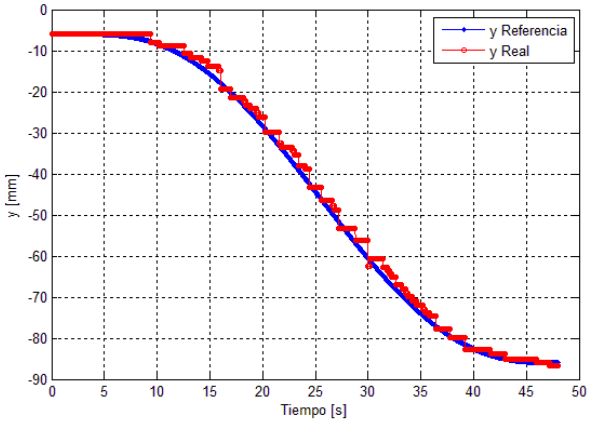
a)



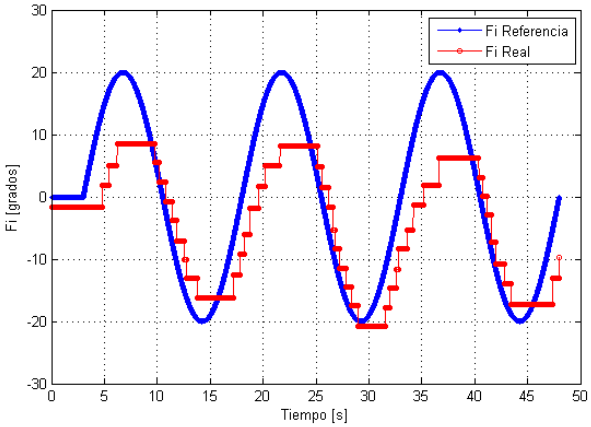
b)



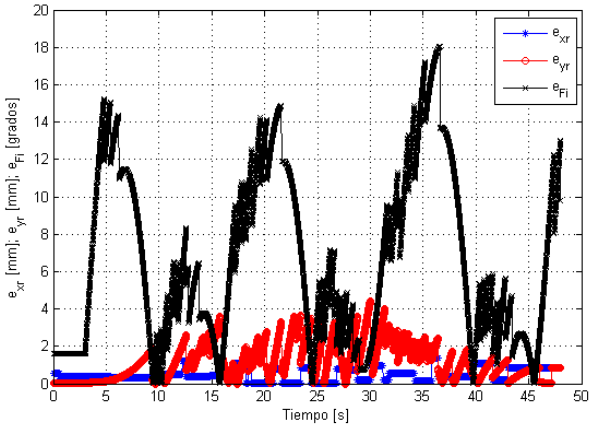
c)



d)



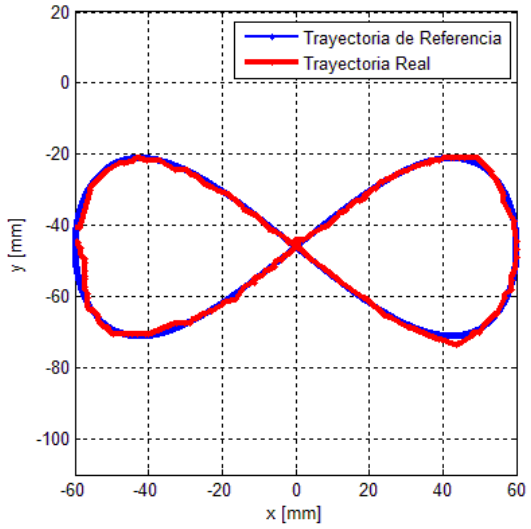
e)



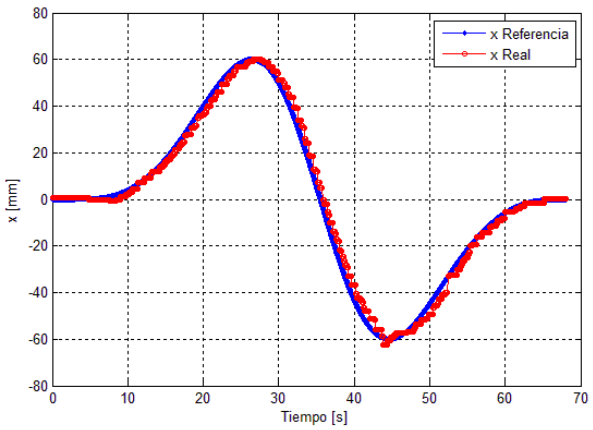
Línea Recta Vertical PP LC, ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* ***x*** *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* ***y*** *con respecto al tiempo, deseado y real.* ***d)*** *Rotación de la PM deseado y real.* ***e)*** *Error absoluto del desplazamiento en* ***x,*** *en* ***y*** *y en ϕ.*

5. Resultados de la trayectoria de un ocho, PP, LC.

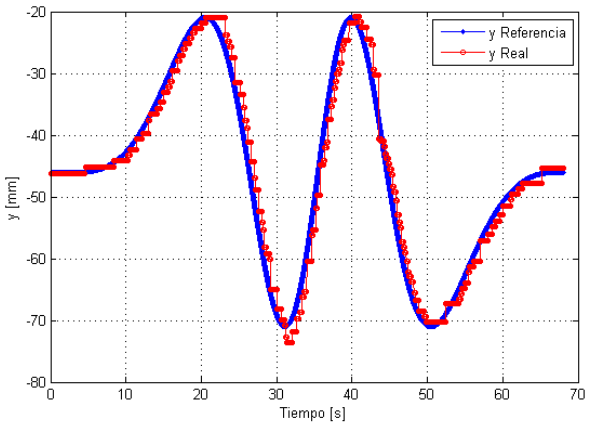
a)



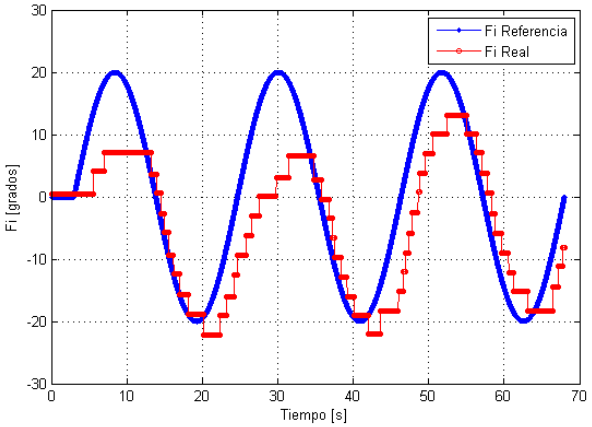
b)



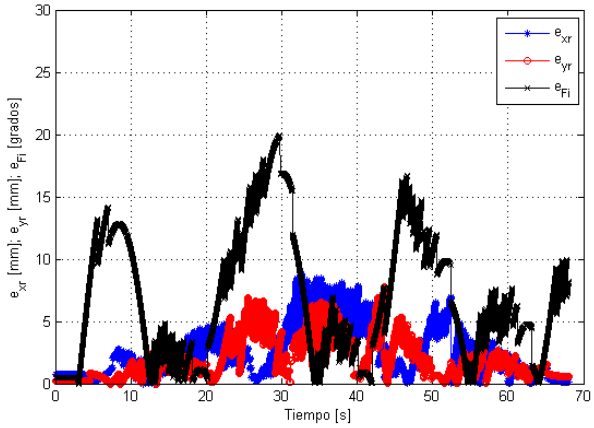
c)



d)



e)



Ocho horizontal PP LC, ***a)*** *Comparativa de la trayectoria deseada y la trayectoria real del robot.* ***b)*** *Desplazamiento en el eje* ***x*** *con respecto al tiempo, deseado y real .****c)*** *Desplazamiento en el eje* ***y*** *con respecto al tiempo, deseado y real.* ***d)*** *Rotación de la PM deseado y real.* ***e)*** *Error absoluto del desplazamiento en* ***x,*** *en* ***y*** *y en ϕ.*

Apéndice B

1. Código de la función S de visión

```
%Programa del sistema de vision
%Abre la comunicación entre ReacTIVision y Simulink. Con esto,
%se obtienen los datos de ReacTIVision (x,y,fi) y se acondicionan los
datos
%para usarlos en el programa de simulink.

function [sys,x0,str,ts] = VisionPmovil(t,x,u,flag)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u,flag);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
%=============================================================================
% mdlInitializeSizes
%=============================================================================
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0  = [];
str = [];     %ts=0.9 es rapido pero se pierden datos y afecta al control
ts  = [0.2 0];%ts=0.25 cuatro muestras/segundo
              %ts=0 es lento y el control no funciona bien
%=============================================================================
% mdlDerivatives
%=============================================================================
function sys=mdlDerivatives(t,x,u)
sys = [];
%=============================================================================
% mdlUpdate
%=============================================================================
function sys=mdlUpdate(t,x,u)
sys = [ ];
```

```
%================================================================================
% mdlOutputs
%================================================================================
function sys=mdlOutputs(t,x,u, flag)
global a3 c x3 y3 f0 a3r;   %Se declaran las variables
import TUIO.*;
tc = TuioClient();  %Se crea el objeto del Cliente TUIO
tc.connect();       %Se conecta con ReacTIVision
for i = 1 : 2   %Se abre un ciclo for, para actualizar los datos
                %constantemente.
                %otra opcion(c8,2)
    c=tc.getTuioObjects().size(); %Guarda el numero de fiducials
    if (c > 0)  %Condición en caso de que exista un fiducial
                %Guarda en una variable la ID del fiducial
      f0 = tc.getTuioObjects().get(0).getSymbolID();
         switch f0
           case 3 %Si es el fiducial 3 obtiene sus parametros.
                a3 = tc.getTuioObjects().get(0).getAngle();
                x3 = tc.getTuioObjects().get(0).getPosition().getX();
                y3 = tc.getTuioObjects().get(0).getPosition().getY();
           otherwise
         end
     end     %Pausa para poder actualizar la posición anterior
             %a la posicion actual
    pause(0.1);   % otras opciones que se probaron;0.03,0.1
end
tc.disconnect();  %Se desconecta de ReacTIVision

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Para convertir de rad a grados de 0 a 360°
 a3=(a3*180)/pi;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Cambiando la referencia del ángulo
if a3>=0 && a3<=270
    a3=a3-90;
end
if a3>270 && a3<=360
    a3=a3-450;
end
 sys = [(x3-0.504312)*520 (y3-0.502353)*385 a3];
%================================================================================
% mdlGetTimeOfNextVarHit
%================================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sys = [];
%================================================================================
% mdlTerminate
%================================================================================
function sys=mdlTerminate(t,x,u)
sys = [];
```

2. Código de la función S para la selección de las trayectorias

```
%Programa de trayectorias
%Con este programa se seleccionan las trayectorias, el perfil de
%trayectoria y el penduleo. Se obtienen las coordenadas de la
%trayectoria en funcion del tiempo.

function [sys,x0,str,ts] = Trayectorias_T(t,x,u,flag)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u,flag);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
%=============================================================================
% mdlInitializeSizes
%=============================================================================
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 5;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [0 0];    %Define cada cuanto tiempo se ejecuta
%=============================================================================
% mdlDerivatives
%=============================================================================
function sys=mdlDerivatives(t,x,u)
sys = [];
%=============================================================================
% mdlUpdate
%=============================================================================
function sys=mdlUpdate(t,x,u)
sys = [ ];
%=============================================================================
% mdlOutputs
%=============================================================================
function sys=mdlOutputs(t,x,u, flag)
global tf k xx yy Rot ki select PP; %Declara las variables
```

```
%=========================================================================
%Función s con todas las trayectorias
%=========================================================================
tf=u(1);     %Tiempo que toma la trayectoria
k=u(2);      %Valor del contador de entrada
select=u(3);%Variable para seleccionar la trayectoria
te=3;        %Tiempo de espera para posicionarse en su inicio
ti=t-te;
PP=ti/tf;    %Parámetro con respecto al tiempo sin perfil de
              trayectoria
%Puntos para una línea con pendiente:
x_1=30;
y_1=-16;
x_2=-30;
y_2=-76;
%Puntos para una línea horizontal:
% x_1=30;
% y_1=-46;
% x_2=-30;
% y_2=-46;
%Puntos para una línea vertical:
% x_1=0;
% y_1=-16;
% x_2=0;
% y_2=-76;
switch select
%=========================================================================
%1.-Linea: Contiene la combinacion entre penduleo y perfil
%=========================================================================
    case 1
        if t>=te
         %Estas dos líneas sirven para agregar el perfil 5to
        %[ft]=Perfil5to_T(ti,tf);
        %[xx yy]=Línea(x_1,y_1,x_2,y_2,ft);
         %Sirve para agregar el penduleo durante la trayectoria
        %[Rot]=Penduleo_T(ti,tf);
         %Si se habilita el perfil se debe deshabilitar esta línea
        [xx yy]=Linea(x_1,y_1,x_2,y_2,PP);
        %Si se habilita el penduleo se deshabilita esta otra
        %Rot=0*(pi/180);
        else
            %Pendiente
            Rot=0;
            xx=30;
            yy=-76;
            %horizontal
            %Rot=0;
            %xx=30;
            %yy=-46;
            %vertical
            %Rot=0;
            %xx=0;
            %yy=-16;
        end
```

```
%=========================================================================
%2.-Cuadrado: Contiene la combinación entre penduleo y perfil 5to
%=========================================================================
    case 2
        if t>=te
        %Sirve para agregar el penduleo durante la trayectoria
         %[Rot]=Penduleo_T(ti,tf);
        %Si se habilita el perfil se debe deshabilitar esta línea
        [xx yy]=Cuadrado_T(ti,tf);
        %Si se habilita el penduleo se deshabilita esta otra
        Rot=0*(pi/180);
        else
            Rot=0;
            xx=30;
            yy=-16;
        end
%=========================================================================
%3.-Circulo: Contiene la combinación entre penduleo y perfil 5to
%=========================================================================
    case 3
        if t>=te
        %Estas dos líneas sirven para agregar el perfil 5to
       [ft]=Perfil5to_T(ti,tf);
       [xx yy]=Circulo(ft);
        %Sirve para agregar el penduleo durante la trayectoria
       [Rot]=Penduleo_T(ti,tf);
        %Si se habilita el perfil se debe deshabilitar esta línea
         %[xx yy]=Circulo(PP);
        %Si se habilita el penduleo se deshabilita esta otra
         %Rot=0*(pi/180);
        else
            Rot=0;
            xx=30;
            yy=-46;
        end
%=========================================================================
%4.-Ocho: Contiene la combinación entre penduleo y perfil 5to
%=========================================================================
    case 4
        if t>=te
        %Estas dos líneas sirven para agregar el perfil 5to
       [ft]=Perfil5to_T(ti,tf);
       [xx yy]=Ocho(ft);
        %Sirve para agregar el penduleo durante la trayectoria
       [Rot]=Penduleo_T(ti,tf);
        %Si se habilita el perfil se debe deshabilitar esta línea
         %[xx yy]=Ocho(PP);
        %Si se habilita el penduleo se deshabilita esta otra
         %Rot=0*(pi/180);
        else
            Rot=0;
            xx=0;
            yy=-46;
        end
```

```
%=========================================================================
%5.-Lisajoux s/penduleo, s/perfil una sola vez
%=========================================================================
    case 5
         if t>=te
        %Estas dos líneas sirven para agregar el perfil 5to
         %[ft]=Perfil5to_T(ti,tf);
         %[xx yy]=Lisajoux(ft);
        %Sirve para agregar el penduleo durante la trayectoria
         %[Rot]=Penduleo_T(ti,tf);
        %Si se habilita el perfil se debe deshabilitar esta línea
        [xx yy]=Lisajoux(PP);
        %Si se habilita el penduleo se deshabilita esta otra
        Rot=0*(pi/180);
        else
            Rot=0;
            xx=0;
            yy=-46;
        end
%=========================================================================
%6.-Lemniscata s/penduleo, s/perfil una sola vez
%=========================================================================
    case 6
        if t>=te
        %Estas dos líneas sirven para agregar el perfil 5to
       [ft]=Perfil5to_T(ti,tf);
       [xx yy]=Lemniscata(ft);
        %Sirve para agregar el penduleo durante la trayectoria
       [Rot]=Penduleo_T(ti,tf);
        %Si se habilita el perfil se debe deshabilitar esta línea
         %[xx yy]=Lemniscata(PP);
        %Si se habilita el penduleo se deshabilita esta otra
         %Rot=0*(pi/180);
        else
            Rot=0;
            xx=0;
            yy=14;
        end
%=========================================================================
%7.-Cardioide s/penduleo, s/perfil una sola vez
%=========================================================================
    case 7
        if t>=te
        %Estas dos lineas sirven para agregar el perfil 5to
       [ft]=Perfil5to_T(ti,tf);
       [xx yy]=Cardioide(ft);
        %Sirve para agragar el penduleo durante la trayectoria
       [Rot]=Penduleo_T(ti,tf);
        %Si se habilita el perfil se debe deshabilitar esta línea
         %[xx yy]=Cardioide(PP);
        %Si se habilita el penduleo se deshabilita esta otra
         %Rot=0*(pi/180);
        else
            Rot=0;
            xx=25;
            yy=-46;
        end
```

```
%=============================================================================
%En caso de no haber ninguna elección
%=============================================================================
    otherwise
        xx=0;
        yy=0;
        Rot=0;
        ki=0;
        PP=0;
end
%El contador es solo para ver cuántas veces hizo la CI
if t==0
    ki=0;
end
%Contador para ver las veces que entro a la función S
if t>=te
    ki=k+1;
end
sys=[xx yy Rot ki PP];

%=============================================================================
% mdlGetTimeOfNextVarHit
%=============================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sys = [];
%=============================================================================
% mdlTerminate
%=============================================================================
function sys=mdlTerminate(t,x,u)
sys = [];
```

3. Código del cálculo de la Cinemática Inversa en una función S

```
%Programa de la cinemática inversa
%Resuelve la cinemática inversa del MDP3RRR y obtiene las
%posiciones angulares de las juntas activas del manipulador

function [sys,x0,str,ts] = Cin_Inv_LC(t,x,u,flag)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u,flag);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
```

```
end
%=============================================================================
% mdlInitializeSizes
%=============================================================================
function [sys,x0,str,ts,T_samp]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 3;
sizes.NumInputs      = 3;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;   % at least one sample time is needed
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [-1 0];
%=============================================================================
% mdlDerivatives
%=============================================================================
function sys=mdlDerivatives(t,x,u)
sys = [];
%=============================================================================
% mdlUpdate
%=============================================================================
function sys=mdlUpdate(t,x,u)
sys = [ ];
%=============================================================================
% mdlOutputs
%=============================================================================
function sys=mdlOutputs(t,x,u, flag)
global Xp Yp RT;
%Dimensiones de los eslabones
l1=125;
l2=100;
l3=44;
%Obtención de 3 variables para la solución de la cinemática inversa
Xp=u(1);
Yp=u(2);
RT=u(3);
%Ángulos de L3
an1=pi/6;   %30°
an2=5*pi/6; %150°
an3=3*pi/2; %270°
%Translación del origen de cada brazo
o1=[-165 -142.5];
o2=[165 -142.5];
o3=[0 143.3];
%La distancia máxima de los eslabones L1 y L2
LT1=l1+l2;
%Angulo del triangulo
Fi1=an1+RT;
Fi2=an2+RT;
Fi3=an3+RT;
%Formación del vector L3
CO1=[l3*cos(Fi1),l3*sin(Fi1)];
CO2=[l3*cos(Fi2),l3*sin(Fi2)];
CO3=[l3*cos(Fi3),l3*sin(Fi3)];
```

```
ptray=[Xp,Yp];
P1=ptray-o1;
P2=ptray-o2;
P3=ptray-o3;
PC1=P1-CO1;
PC2=P2-CO2;
PC3=P3-CO3;
L1=sqrt((PC1((1)))^2+(PC1((2)))^2);
L2=sqrt((PC2((1)))^2+(PC2((2)))^2);
L3=sqrt((PC3((1)))^2+(PC3((2)))^2);
%Condicion que cumple el triángulo entre los dos eslabones
 if L1 > LT1
    L1=LT1;
end
if L2 > LT1
    L2=LT1;
end
if L3 > LT1
    L3=LT1;
End

h1=((L1)^2+l1^2-l2^2)/(2*l1*L1);
h2=((L2)^2+l1^2-l2^2)/(2*l1*L2);
h3=((L3)^2+l1^2-l2^2)/(2*l1*L3);
g1=((L1)^2-l1^2-l2^2)/(2*l1*l2);
g2=((L2)^2-l1^2-l2^2)/(2*l1*l2);
g3=((L3)^2-l1^2-l2^2)/(2*l1*l2);

beta1=atan2(PC1((2)),PC1((1)));
beta2=atan2(PC2((2)),PC2((1)));
beta3=atan2(PC3((2)),PC3((1)));

yi1=acos(h1);
yi2=acos(h2);
yi3=acos(h3);

%Cuando esta en - es codo abajo y cuando esta en + codo arriba
T1a=(beta1-yi1)*180/pi;
T1b=(beta2-yi2)*180/pi;
T1c=(beta3-yi3)*180/pi;

%Modo inverso al anterior, + codo abajo y en - codo arriba
T2a=(acos(g1))*180/pi;
T2b=(acos(g2))*180/pi;
T2c=(acos(g3))*180/pi;

T3a=(Fi1)*180/pi-T1a-T2a;
T3b=(Fi2)*180/pi-T1b-T2b;
T3c=(Fi3)*180/pi-T1c-T2c;


%Convierte los ángulos de la cinemática inversa a un solo rango de
%de 0° a 180° ya que son enviados a los servomotores.

T1a=T1a+60;
```

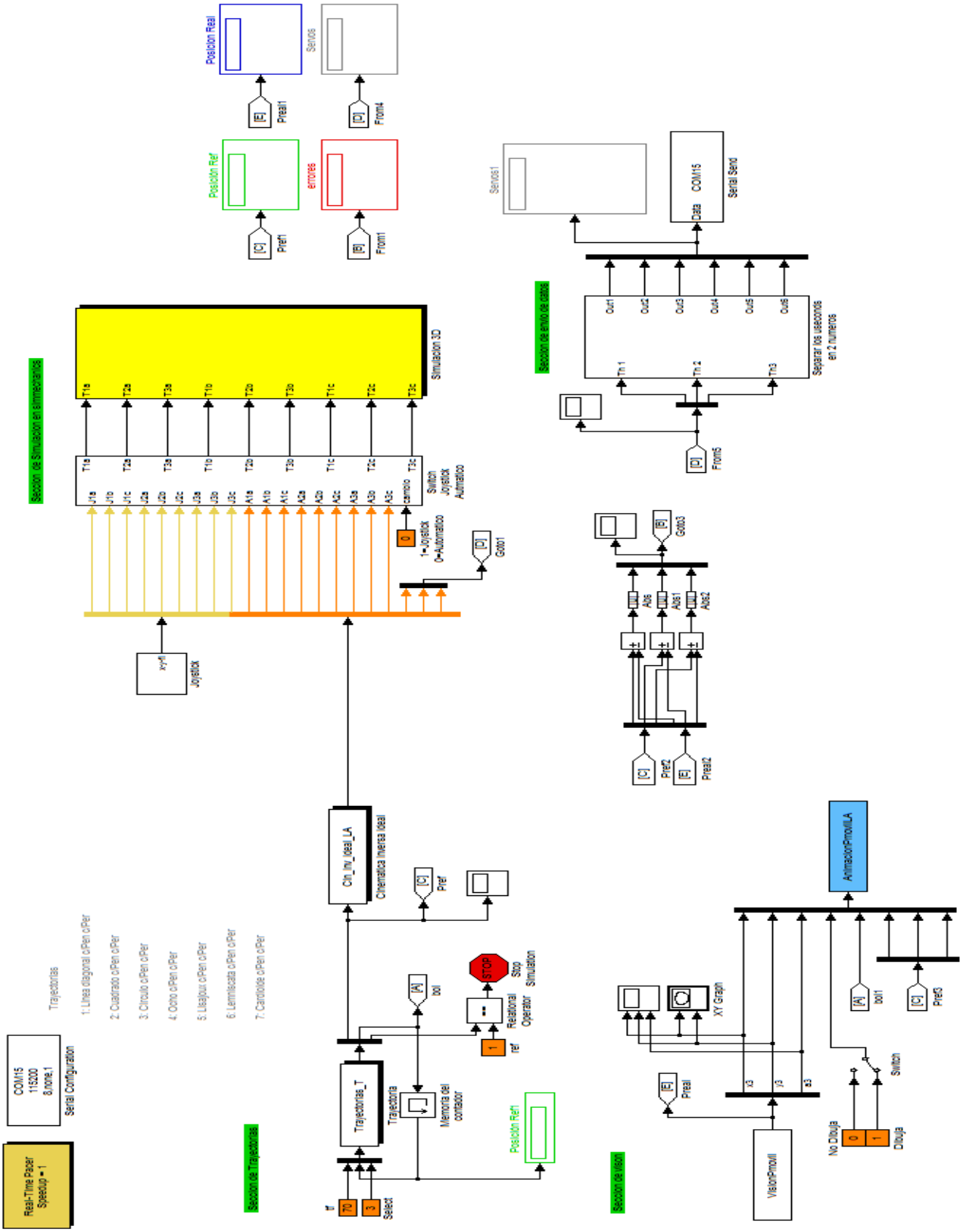
```
if T1a<=0
    T1a=T1a+360;
end

T1b=T1b-60;
if T1b<0
    T1b=T1b+420;
end

if T1c<0
    T1c=T1c+180;
else T1c=-T1c+360;
end


sys = [T1a T1b T1c];

%=============================================================================
% mdlGetTimeOfNextVarHit
%=============================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sys = [];
%=============================================================================
% mdlTerminate
%=============================================================================
function sys=mdlTerminate(t,x,u)
sys = [];
```

4. Código de los Servomotores en la plataforma Arduino

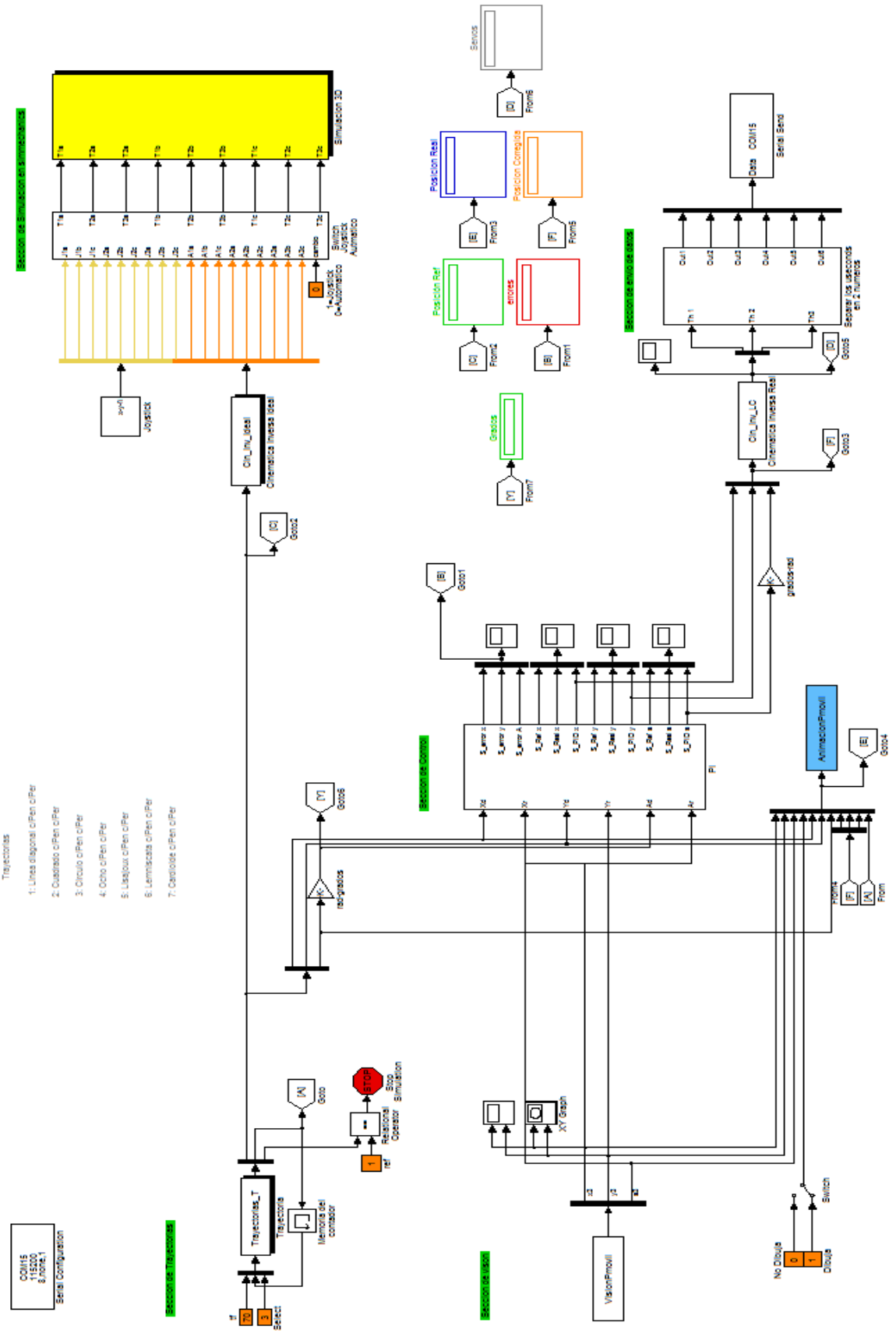
```
////////////////////////////////////////////////////////////////////////////////////
///Manipulador Paralelo Plano tipo 3RRR (Delta)
///Cristhian Gómez González
///Descripción:
///Este programa recibe 6 bytes del puerto serial, estos son ocupados para mover 3 servos.
///Cada par de byte obtenido contiene la información de la posición de cada servo.
///Ejemplo: Desde simulink se envía un numero entero de 4 dígitos "1234", enviado en dos pares
///un par es 12 y el otro 34, arduino los recibe por separado y lo que hace es juntarlos.
///Y los envía los servomotores, de esta manera se tiene un rango de 0 a 9999 (us) y no un rango 0 a
255 (posición)
////////////////////////////////////////////////////////////////////////////////////
//Incluir librerías y declaración de variables

#include <Servo.h>
Servo servo1;
Servo servo2;
Servo servo3;
int ser1a, ser1b, ser2a, ser2b, ser3a, ser3b, ser1, ser2, ser3;
```

```
///////Configuración
void setup()
{
  Serial.begin(115200);            //Ancho de pulso min y max, de la caracterización
  servo1.attach(9,530,2310);       //Asignado al M1, vértice izquierdo del triángulo de la base
  servo2.attach(10,520,2300);      //Asignado al M2, vértice derecho del triangulo
  servo3.attach(11,530,2310);      //Asignado al M3, vértice superior
                                   //Ancho de pulso del fabricante; min 530, max 2450
}
//////Algoritmo
void loop()
{

    if(Serial.available()>5)
    {
    ser1a=Serial.read();           //Recibe 1 byte y lo almacena en una variable
    ser1b=Serial.read();
    ser2a=Serial.read();
    ser2b=Serial.read();
    ser3a=Serial.read();
    ser3b=Serial.read();

    alfa=ser1a*100 + ser1b;        //Suma los 2 bytes de cada servo,
    beta=ser2a*100 + ser2b;         //para tener los microsegundos
    gama=ser3a*100 + ser3b;

    if(alfa > 2292){alfa=2292;}   //Condiciones para evitar que salga
    if(alfa < 540){alfa=540;}      //del rango de operación de cada servo
    if(beta > 2250){beta=2250;}
    if(beta < 520){beta=520;}
    if(gama > 2290){gama=2290;}
    if(gama < 530){gama=530;}

    servo1.writeMicroseconds(alfa);       //Envía los microsegundos a los servomotores
    servo2.writeMicroseconds(beta);
    servo3.writeMicroseconds(gama);

    delay(25);     //Pausa entre el dato anterior y el dato actual
    }              //para que el servo pueda cambiar de posición
}

//Nota: sincronizar los tiempos tanto del envío (software que envía los datos)
//como el tiempo de espera (delay())o pausa que le da el tiempo para que cambie el servo
//a un nueva posición.
```

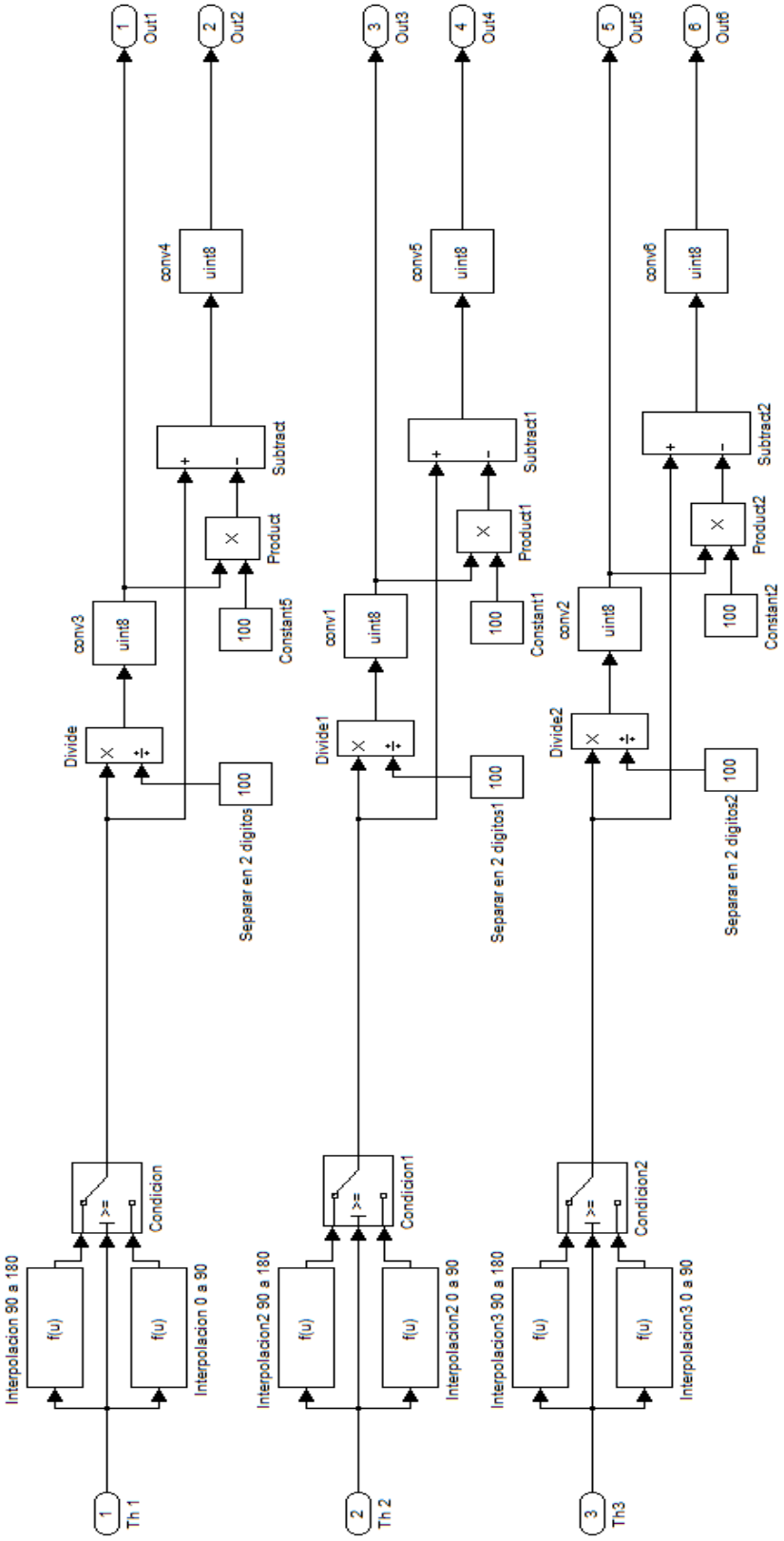
1. Diagrama de bloques de Simulink del sistema en lazo abierto.



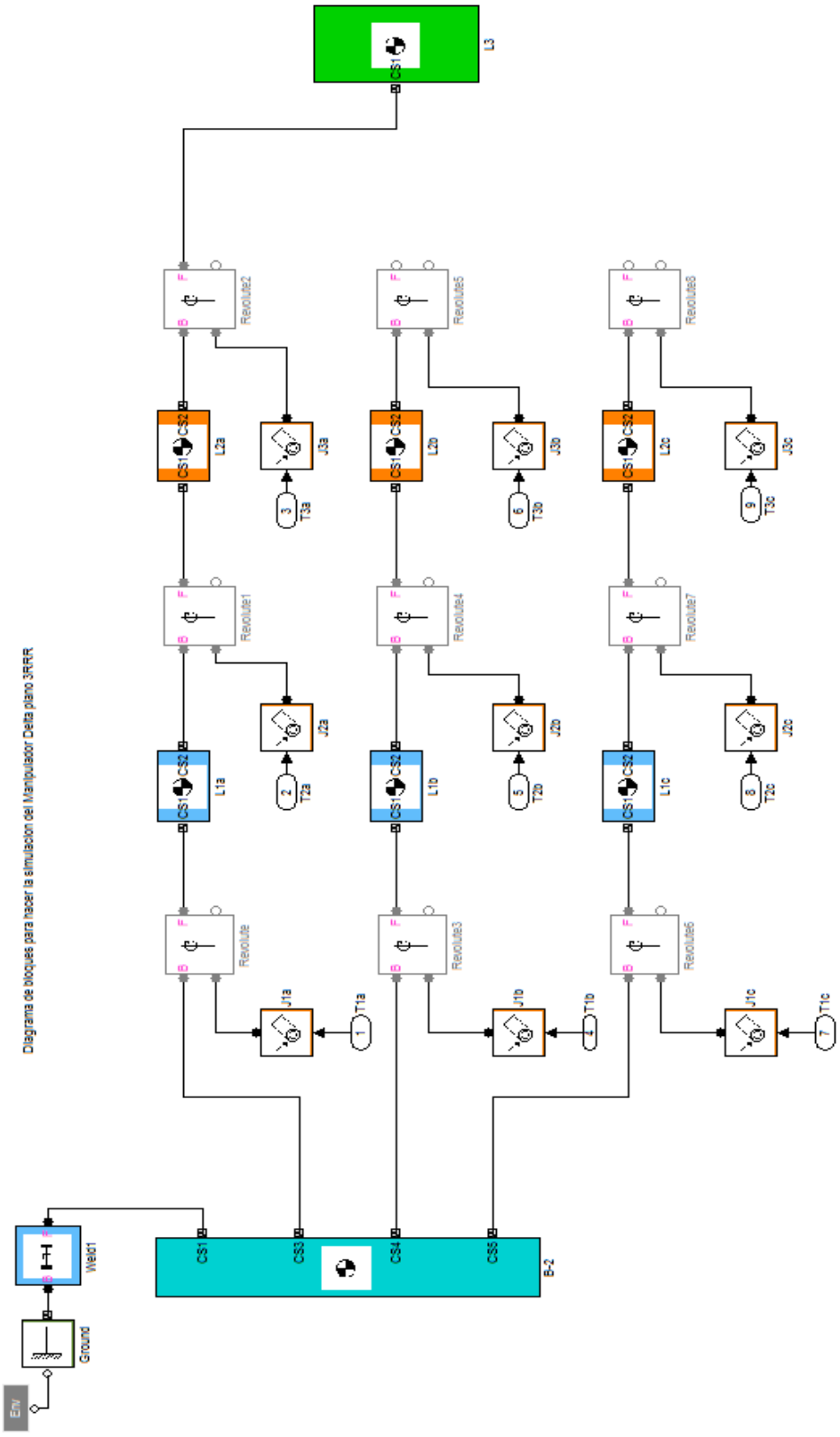
2. Diagrama de bloques en Simulink del sistema en lazo cerrado.



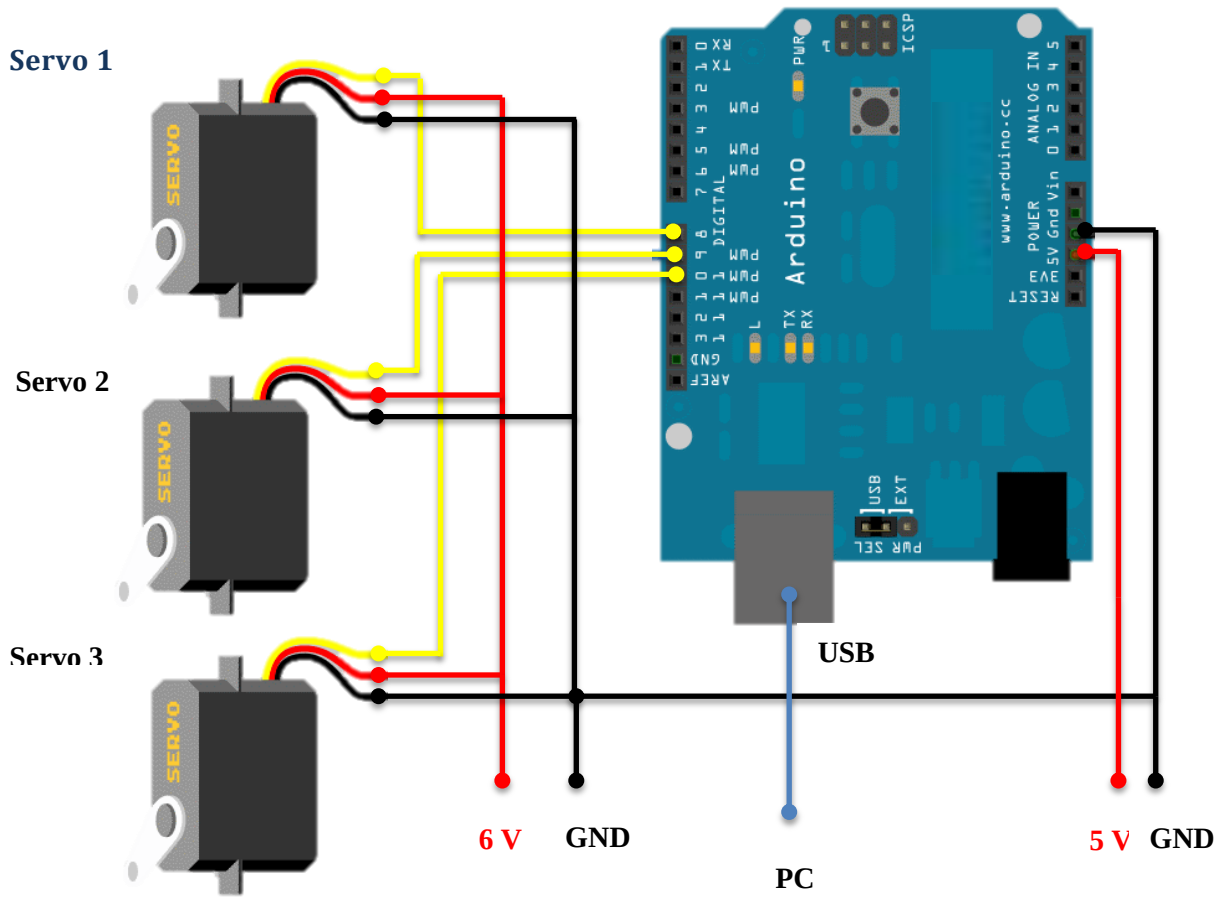
3. Diagrama de acondicionamiento de los datos a enviar.



4. Diagrama de bloques de la simulación en SimMechanics.



5. Diagrama eléctrico de los servomotores



Apéndice E

1. ReacTIVision.

Method Summary

float	***getAngle***() Returns the rotation angle of this TuioObject.
float	***getAngleDegrees***() Returns the rotation angle in degrees of this TuioObject.
float	***getMotionAccel***() Returns the motion acceleration of this TuioContainer.
float	***getMotionSpeed***() Returns the motion speed of this TuioContainer.
java.util.Vector <*TuioPoint*>	***getPath***() Returns the path of this TuioContainer.
TuioPoint	***getPosition***() Returns the position of this TuioContainer.
float	***getRotationAccel***() Returns the rotation acceleration of this TuioObject.
float	***getRotationSpeed***() Returns the rotation speed of this TuioObject.
long	***getSessionID***() Returns the Session ID of this TuioContainer.
int	***getSymbolID***() Returns the symbol ID of this TuioObject.
int	***getTuioState***() Returns the TUIO state of this TuioContainer.
float	***getXSpeed***() Returns the X velocity of this TuioContainer.
float	***getYSpeed***() Returns the Y velocity of this TuioContainer.
boolean	***isMoving***() Returns true of this TuioObject is moving.

2. Datos técnicos de Arduino Duemilanove.

Microcontroller	*ATmega328*
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

3. Ancho de pulso de los servos en base a la caracterización.

No. Servo	*Mínimo*	*Máximo*
Servo 1	540	2292
Servo 2	520	2250
Servo 3	530	2290

4. Instalación del cliente TUIO java

Dado a que Matlab soporta Java, hacemos uso del cliente TUIO en Java. Instalando el cliente podremos tener acceso a los datos de ReacTIVision, así podremos utilizar la biblioteca y las propiedades para crear los objetos.

- Bajar el archivo TUIO_JAVA de la página de ReacTIVision.
- Extraer el archivo y guardarlo dentro de una nueva carpeta (preferiblemente colocar la carpeta nueva dentro de la carpeta de MATLAB).
- Abrimos Matlab como administrador y dentro de la ventana de comandos escribimos “edit classpath.txt”.

- Dicho archivo se abrirá y debemos agregar las direcciones de los archivos que guardamos en la nueva carpeta, “TuioDemo.jar” y “libTUIO.jar”.
- Finalmente se guarda el archivo y se cierra.

Con estos pasos tendremos acceso a las propiedades de ReacTIVision.

5. Instalación del SimMechanics link software y el add-in tool en SolidWorks

Con la instalación de este add-in tool en SolidWorks se podrán guardar los ensambles hechos en CAD en el formato XML, formato que requiere Matlab para crear automáticamente los ensambles en diagramas de bloques de SimMechanics. Para ello, también es necesario la instalación del SimMechanics link software en Matlab, útil para la conversión del archivo xml al diagrama de bloques.

Matlab también proporciona el add-in tool de otras paqueterías como Autodesk Inventor, y Pro/ENGINER. Sin embargo, para SolidWorks se hace lo siguiente:

- Primero descargamos los archivos de SimMechanics link software de la página: http://www.mathworks.com/products/simmechanics/download_smlink.html.
- Debemos descargar el archivo que corresponda a la versión de Matlab que se tenga instalado y la versión de 32bits o 64bits.
- Una vez descargado y guardado los archivos (“*install_addon.m*” y por ejemplo: “*smlink.r2011a.win64.zip*”) en una carpeta nueva (preferiblemente dentro del escritorio), abrir Matlab como administrador y en el *current folder* cambiar la dirección a donde se encuentran los archivos descargados.
- En la ventana de comandos de Matlab escibir el siguiente comando para la instalación del archivo: **install_addon(‘zip_file_name’).** En el zip_file_name corresponde al nombre del archivo que se descargó en este ejemplo sería “*smlink.r2011a.win64.zip*”. Con esto ya se tiene instalado el software de conversión.

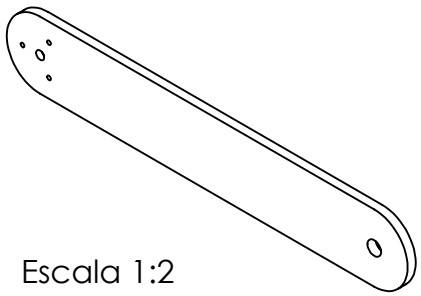
Por último debemos registrar el SimMechanics link con la plataforma CAD, en otras palabras es activar el add-in tool a SolidWorks.

- En la ventana de comandos de Matlab escribir el comando ***smlink_linksw,*** al terminar muestra un mensaje de que el enlace fue un éxito, además escribir ***regmatlabserver*** para adjuntar el registro de SimMechanics a Matlab.

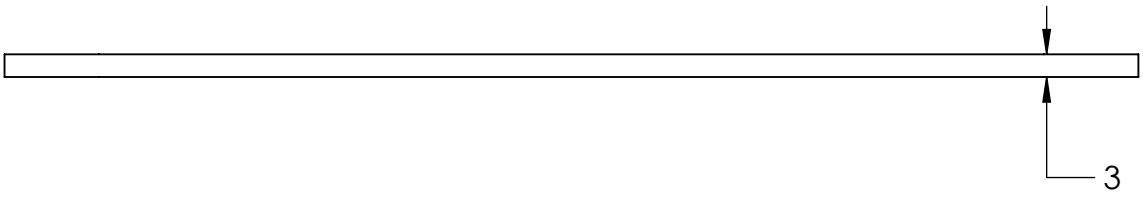
- Finalmente en SolidWorks debemos activar el complemento haciendo lo siguiente: en la barra de menú de SolidWorks seleccionar *tool>add-Ins* y dentro del cuadro de Add-Ins seleccionar la casilla ***SimMechanics Link.***

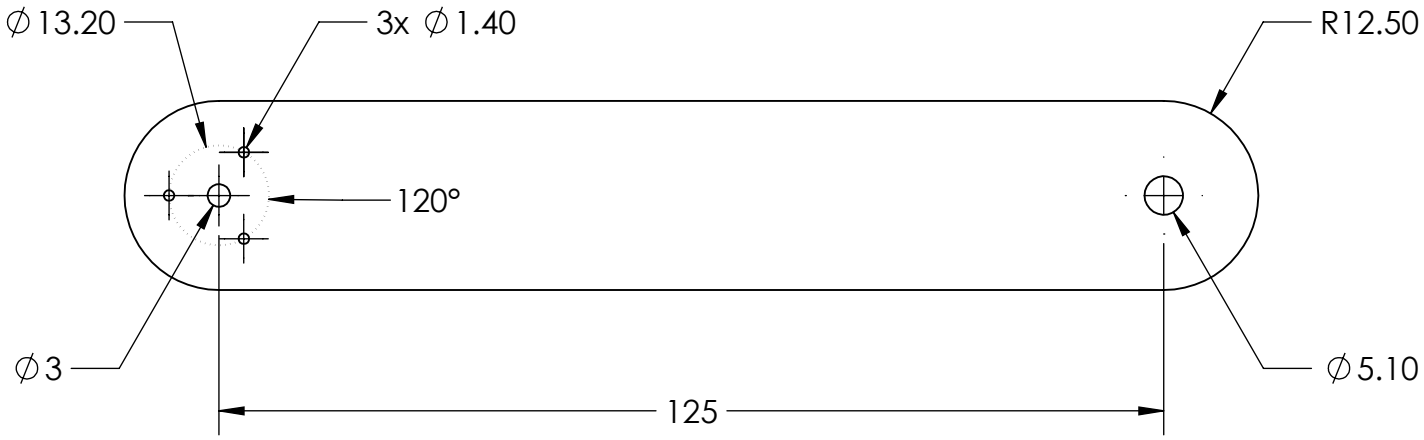
Apéndice F

Planos del manipulador delta y de la estructura.

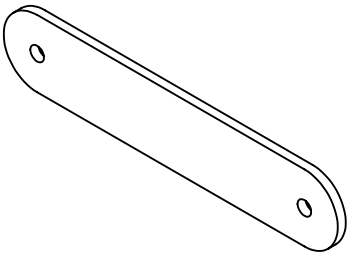


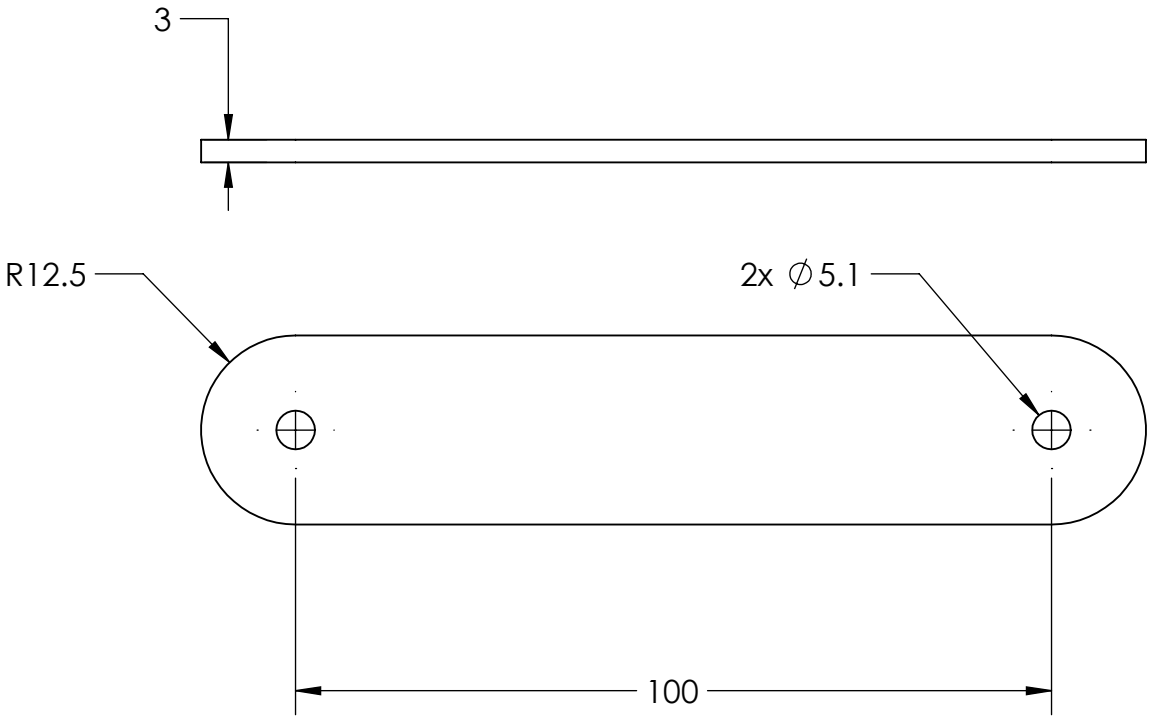
Escala 1:2



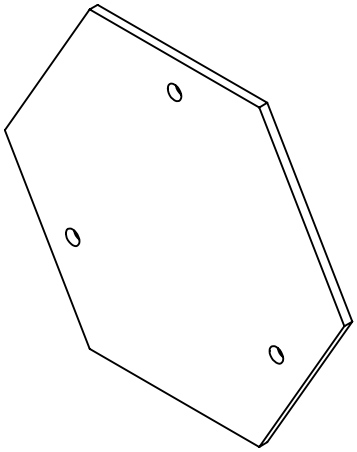


Acot: mm	Universidad Nacional Autónoma de México Facultad de Ingeniería
Material: Acrilico	
ANSI	Dibujó: Cristhian G. Glz. / ESCALA:1:1
	TÍTULO: Planos MDP
	Nombre del Dibujo: E1 / A4
	20/02/2012 / Cantidad: 3 / HOJA 1 DE 6

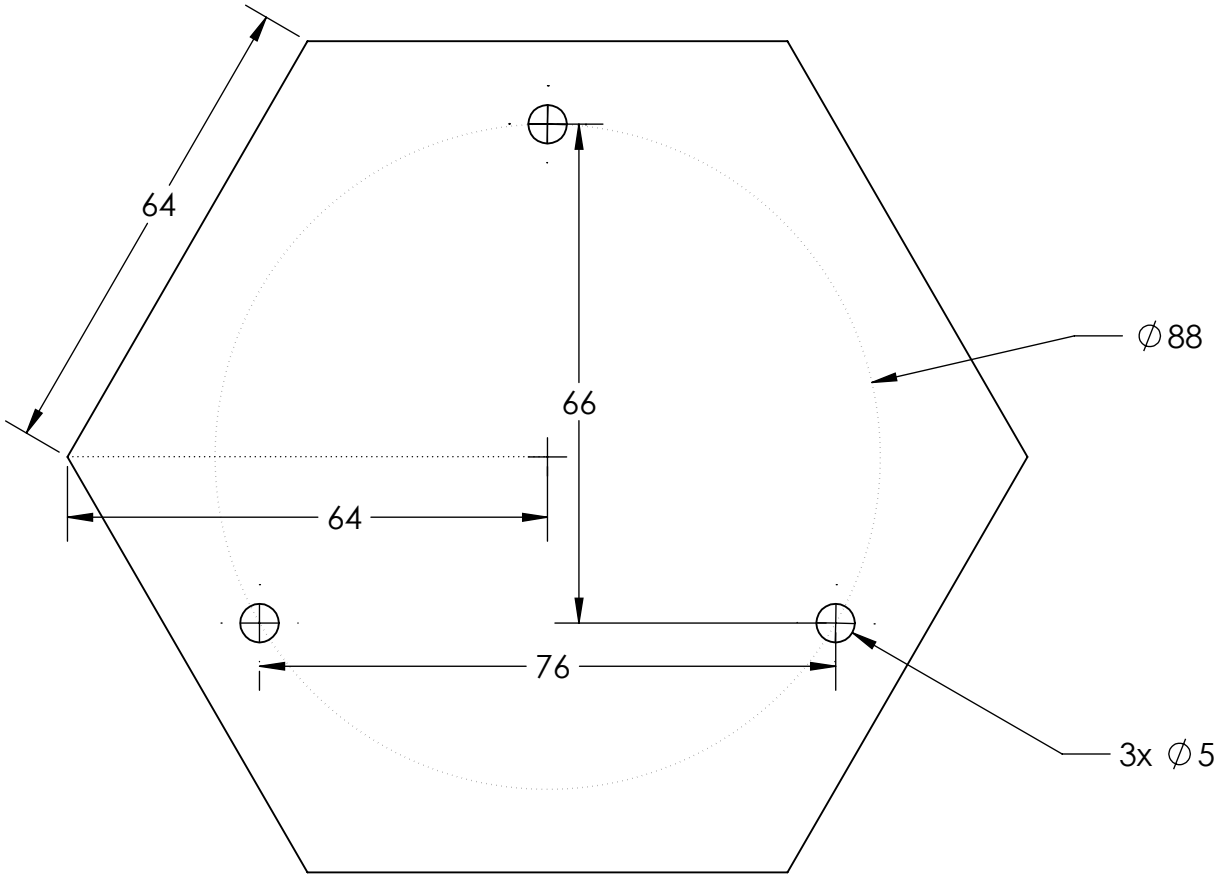




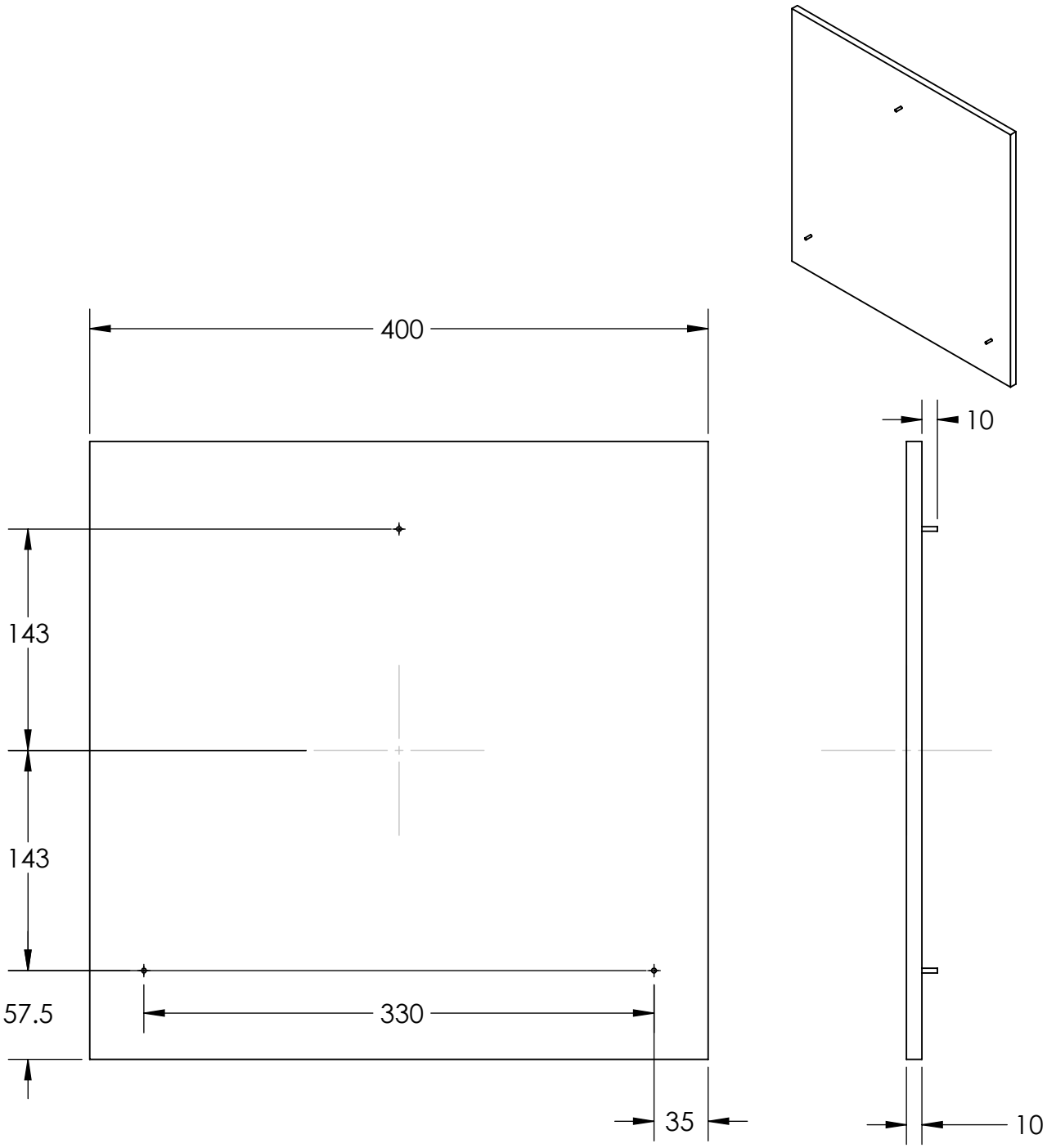
Acot: mm	Universidad Nacional Autónoma de México Facultad de Ingeniería
Material: Acrilico	
ANSI	Dibujó: Cristhian G. Glz. / ESCALA:1:1
	TÍTULO: Planos MDP
	Nombre del Dibujo / E2 / A4
	20/02/2012 / Cantidad: 3 / HOJA 2 DE 6



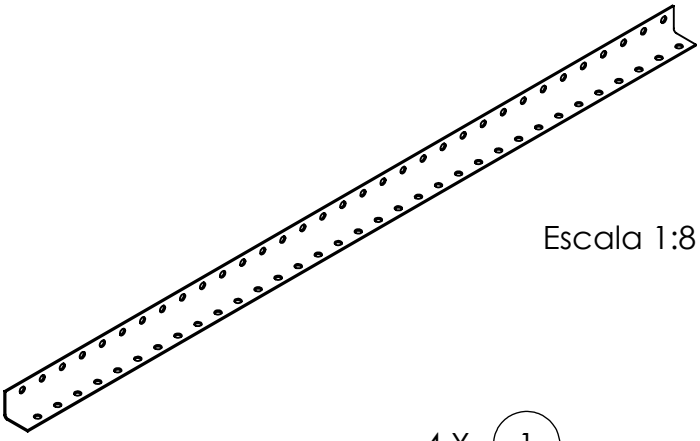
Escala 1:2



Acot: mm	Universidad Nacional Autónoma de México Facultad de Ingeniería
Material: Acrilico	
ANSI	Dibujó: Cristhian G. Glz. / ESCALA:1:1
	TÍTULO: Planos MDP
	Nombre del Dibujo: Hex / A4
	20/02/2012 / HOJA 3 DE 6

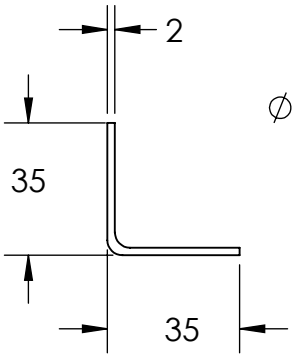


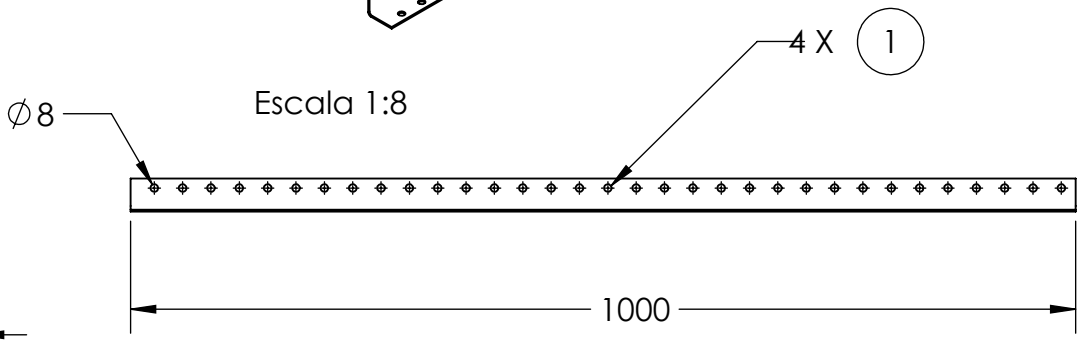
Acot: mm	Universidad Nacional Autónoma de México Facultad de Ingeniería
Material: Acrilico	
ANSI	Dibujó: Cristhian G. Glz. — ESCALA:1:1
	TÍTULO: Planos MDP
	Nombre del Dibujo: Base — A4
	20/02/2012 — HOJA 4 DE 6



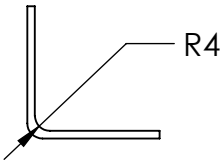
Escala 1:8

Escala 1:2

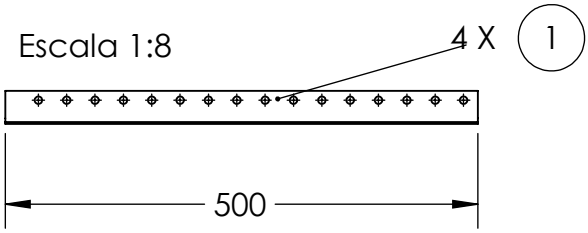




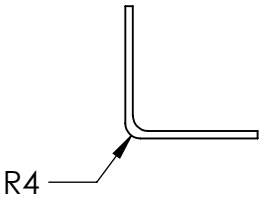
Escala 1:8



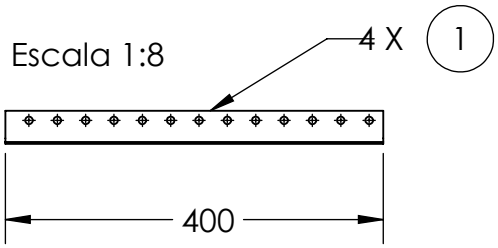
Escala 1:2



Escala 1:8

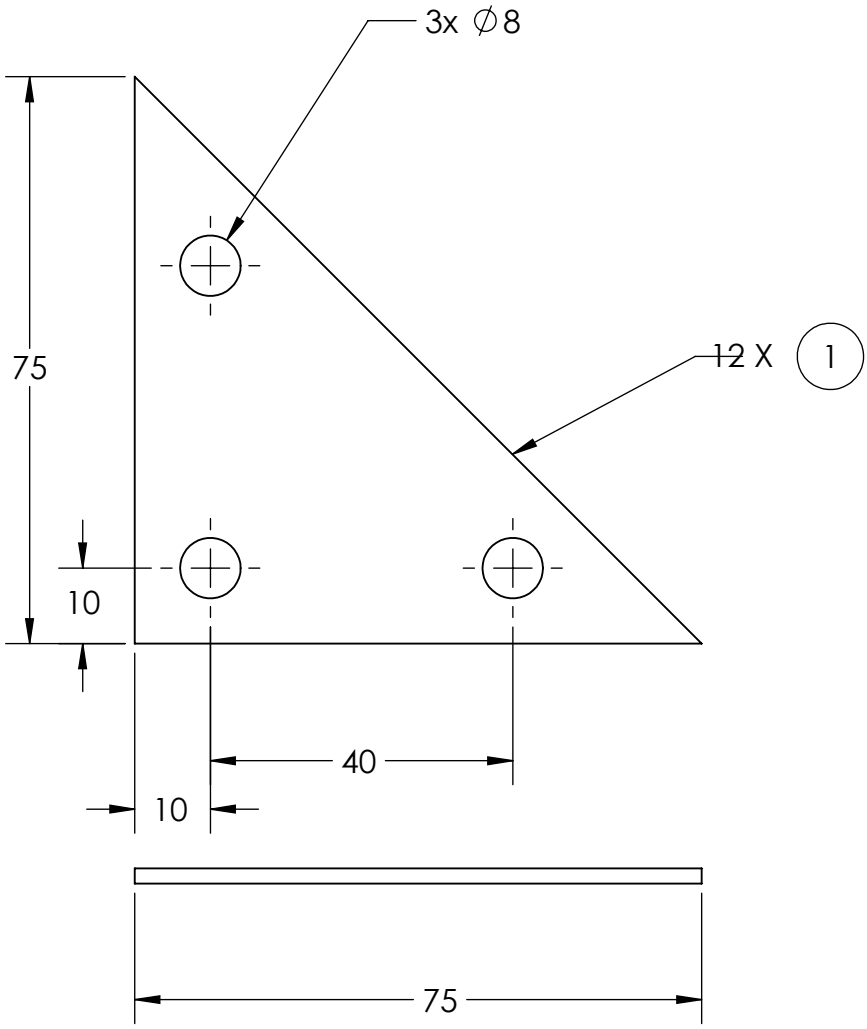


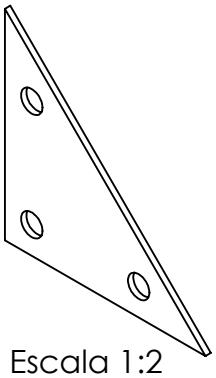
Escala 1:2



Escala 1:8

Acot: mm	Universidad Nacional Autónoma de México Facultad de Ingeniería
Material: Fierro	
ANSI	Dibujó: Cristhian G. Glz.
	TÍTULO: Planos MDP
	Nombre del Dibujo: Barras — A4
	20/02/2012 — HOJA 5 DE 6





Escala 1:2

Acot: mm	Universidad Nacional Autónoma de México Facultad de Ingeniería
Material: Fierro	
ANSI	Dibujó: Cristhian G. Glz. — ESCALA:1:1
	TÍTULO: Planos MDP
	Nombre del Dibujo: — Esc — A4
	20/02/2012 — HOJA 6 DE 6