



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERIA



EXPERIMENTACIÓN EN TIEMPO REAL SOBRE DEFORMACIÓN REACTIVA DE TRAYECTORIAS PARA ROBÓTICA MÓVIL

TESIS
Que para obtener el título de
INGENIERO MECATRÓNICO

PRESENTA
ALEJANDRO MÉNDEZ SANDOVAL

DIRECTOR DE TESIS
DR. VICTOR JAVIER GONZÁLEZ VILLELA

MÉXICO DF NOVIEMBRE 2011



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Dedicatorias

A mis papás Eduardo y Virginia por su amor, confianza y consejo a lo largo de mi vida. Ustedes son la razón de éste y todos mis proyectos.

A mis hermanos Lalo y Memo, el amor y amistad que me ofrecen es invaluable.

A mis amigos, quienes me han aconsejado y apoyado incondicionalmente.

Alejandro Méndez Sandoval
Septiembre 2011

Agradecimientos

Al Dr. Víctor Javier González Villela, por la oportunidad, paciencia y apoyo a través de la realización de este proyecto.

A los compañeros y profesores del cubículo por todo el conocimiento compartido en este proceso.

A la Universidad por la excelencia de la preparación que me ofrecieron, dentro y fuera del aula.

A DGAPA, UNAM, por el apoyo brindado a través del proyecto PAPIIT IN115811, con título: "Investigación y desarrollo en sistemas mecatrónicos: robótica móvil, robótica híbrida y teleoperación" durante la realización de este trabajo.

Índice

| | |
|---|----|
| Resumen | 5 |
| Introducción | 7 |
| 1. Aspectos generales (contexto general) | 9 |
| 2. Trabajo previo | 11 |
| 3. Planteamiento del problema | 11 |
| 4. Objetivo | 12 |
| 5. Contribuciones | 12 |
| 6. Organización | 13 |
| II. Planeación del movimiento y control del robot | 15 |
| 1. Deformación reactiva de trayectorias | 17 |
| 2. Robótica móvil | 19 |
| 2.1 Introducción a la robótica móvil | 19 |
| 2.2 Tipos de robots móviles | 20 |
| 2.3 Robot usado para la prueba | 25 |
| 2.4 El robot móvil por diferencial eléctrico | 26 |
| 2.4.1 Postura del robot | 26 |
| 2.4.2 Modelo cinemático | 27 |
| 3. Esquema de Control | 29 |
| III. Implementación del sistema | 31 |
| 1. Descripción general del sistema | 33 |
| 2. Diseño de la prueba o experimento (Simulink) | 35 |
| 1 Deformación de trayectorias | 35 |
| 2 ReactIVision | 36 |
| 3 Comunicación con ReactIVision | 37 |
| 4 Acoplamiento de datos | 41 |
| 5 Implementación de visión sobre la deformación de trayectorias | 44 |
| 6 Funcionamiento en tiempo real | 46 |
| 7 Comunicación con el robot | 48 |
| 8 Comunicación con los motores | 50 |
| IV. Experimentos y resultados | 53 |
| 1. Entorno de las pruebas | 55 |
| 2. Pruebas en lazo semicerrado | 58 |
| 1 Obstáculos fijos | 58 |
| 2 Obstáculo fijo y obstáculo móvil | 63 |
| 3 Obstáculos móviles | 69 |



| | |
|---|-----|
| 3. Pruebas en lazo cerrado | 74 |
| 4. Discusión | 78 |
| V. Conclusiones y trabajo futuro | 83 |
| 1. Conclusiones | 85 |
| 2. Trabajo futuro | 86 |
| VI. Apéndices | 87 |
| 1. ReactIVision | 89 |
| 2. Código de function S de visión | 90 |
| 3. Acoplamiento de velocidades | 94 |
| 4. Correspondencia de velocidad real y comando MD25 | 98 |
| 5. Arduino BT | 99 |
| 6. Tarjeta drivers MD25 | 102 |

Tabla de figuras

Figura 1. Diagrama objetivo del sistema que realizará el experimento

Figura 2. Deformación de trayectorias

Figura 3. Robot y su interacción con el entorno

Figura 4. Llanta fija

Figura 5. Llanta centrada orientable

Figura 6. Llanta descentrada orientable

Figura 7. Arquitecturas usuales de robots móviles

Figura 8. Tipos de robots móviles

Figura 9. Robot usado en la prueba

Figura 10. Esquemático de la postura de un robot móvil

Figura 11. Postura del robot tipo (2,0)

Figura 12. Diagrama esquemático del sistema

Figura 13. Captura de pantalla del sistema en lazo semi-cerrado

Figura 14. Ejemplos de símbolos fiducial

Figura 15. Reconocimiento de fiducial

Figura 16. Entorno ReactIVision

Figura 17. Aplicación ReactIVision

Figura 18. Símbolos fiducial utilizados para la prueba

Figura 19. Función para obtener datos de visión

Figura 20. Funciones para acoplar las señales de control

Figura 21. Comparativa de comandos de velocidad real y enviada

Figura 22. Bloque para trabajar en tiempo real (Real Time Pacer)

Figura 23. Propiedades del bloque Real Time Pacer

Figura 24. Arduino BT

Figura 25. Módulo WT11 Bluegiga

Figura 26. Esquemático de comunicación Computadora-Robot

Figura 27. Motor EMG30 y tarjeta MD25

Figura 28. Sistema real

Figura 29. Captura de pantalla para la prueba de lazo cerrado

Figura 30. Prueba 1 Obstáculos fijos

Figura 31. Prueba 2 Obstáculos fijos

Figura 32. Prueba 3 Un obstáculo fijo y uno móvil

Figura 33. Prueba 4 Un obstáculo fijo y uno móvil

Figura 34. Prueba 5 Dos obstáculos móviles

Figura 35. Prueba 6 Dos obstáculos móviles

Figura 36. Prueba 7 Seguimiento de senoidal positiva

Figura 37. Prueba 8 Seguimiento de senoidal negativa

Resumen

En esta tesis se presenta la experimentación en tiempo real para probar la teoría acerca de Deformación Reactiva de Trayectorias para robótica móvil.

El sistema está basado en 2 obstáculos y un robot con direccionamiento diferencial, este último sigue una trayectoria predefinida mediante ecuaciones parametrizadas en el tiempo evadiendo los 2 obstáculos. Los datos del robot y de los obstáculos se obtienen mediante un software llamado ReactIVision, éstos ingresan mediante un cliente TUIO en Java a Matlab donde se realiza el algoritmo de deformación de trayectorias y un control no lineal mediante Simulink. Utilizamos envío de datos por comunicación serial inalámbrica por Bluetooth mediante ArduinoBT y protocolo I2C para la comunicación con la tarjeta MD25, encargada de controlar los motores acoplados a cada una de las llantas.

Se presentan pruebas en lazo "semicerrado", llamado así pues es un lazo cerrado pero con datos provenientes del modelo cinemático del robot, con obstáculos fijos y móviles y en lazo totalmente cerrado, realimentando datos del sistema de visión (datos reales) sólo para seguimiento de una trayectoria.

I. Introducción

1. Aspectos generales

Mecatrónica, como quiera que sea, se refiere exclusivamente a una integración multidisciplinaria en el diseño de sistemas de manufactura y de productos en general. Esta representa la nueva generación de máquinas, robots y mecanismos expertos necesarios para realizar trabajo en una variedad de ambientes, principalmente en la automatización de las fábricas, de las oficinas, y de las casas [1].

En particular hablar de robots implica hablar de su estudio, la robótica. El término robótica surge al momento en el que surge el de robot, en 1920 precedido por la palabra checa robot, que significa "siervo" o "trabajo obligatorio" en ruso (Capek, 1920). Ahora bien, la robótica es la técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales [2]. Basándonos en las definiciones presentadas, podemos enfatizar en la importancia que tiene este campo de estudio, y en particular, estos sistemas en nuestra vida cotidiana.

Como ya se mencionó, los robots surgen como siervos, y su aplicación, en sus orígenes y tiempo después se limitaba a la industria. En nuestros días no nos resulta ajena la idea de ver robots en establecimientos, o incluso en nuestras casas. Los robots cada día se extienden a secciones donde antes no se concebían. A pesar de esta expansión de su aplicación, es de resaltar que la aplicación industrial de los robots sigue siendo la principal, donde hay más inversión en el área y se obtienen mayores beneficios.

Por esta razón el desarrollo de funciones distintas a nivel industrial implica desarrollo en las funciones que son capaces de realizar los robots. La evolución en las teorías de flujo de materiales y manejo de los mismos dentro de los procesos industriales, por ejemplo, impulsa y exige el desarrollo de sistemas capaces de cumplir tareas sobre medios de transporte eficaz de materiales, es aquí donde surgen los primeros robots móviles. En principio los robots fueron simplemente considerados como máquinas programables capaces de realizar aquellas tareas repetitivas para las que el ser humano no requiere cierta destreza y por tanto de su inteligencia [3], sin embargo, la tendencia hoy en día es generar sistemas totalmente autónomos capaces de realizar estas tareas repetitivas mencionadas pero en entornos desconocidos o dinámicos, es decir, que esté provisto de algoritmos que, a partir de la información que se tiene del entorno y de la tarea, el robot la desempeñe sin contratiempos. Esto implica sistemas sensoriales para darle al robot la información que necesita para trabajar sin necesidad de intervención humana, ya sean sensores montados sobre el robot o fuera de él.

El avance tecnológico y la creciente complejidad de los sistemas y tareas a realizar nos lleva a diseñar robots y algoritmos capaces de realizar tareas más complejas que sólo seguir



trayectorias preestablecidas, como es el caso de los vehículos guiados automáticamente (AGV, por sus siglas en inglés) que fueron los primeros vehículos que aparecieron para resolver este problema. Esto implica que es requerido cierto flujo de información, del entorno hacia un sistema de procesamiento y finalmente al sistema de actuación del robot. Bajo este esquema de funcionamiento del sistema podemos clasificar a las arquitecturas de movimiento de la siguiente manera:

- Deliberativa
- Reactiva

La aproximación deliberativa utiliza técnicas de planeación de movimientos [4] donde se requiere el modelo del entorno tan completo como sea posible, sin embargo el problema radica en la complejidad de su aplicación para ambientes desconocidos o dinámicos.

Por otro lado, la aproximación reactiva puede operar en cualquier tipo de ambientes, sea desconocido o dinámico. La arquitectura reactiva opera mediante la obtención de información sobre el entorno en cada iteración y determina la señal de actuación solamente para el siguiente paso. Estas aproximaciones son aplicables para evasión de obstáculos o exploración de ambientes sin colisionar con objetos.

El trabajo matemático realizado nos lleva a ecuaciones aplicables al robot para calcular la trayectoria a seguir, o por lo menos el siguiente punto. Esto generalmente es probado mediante simulaciones con las cuales podemos concluir acerca del funcionamiento de la teoría. El siguiente paso es claro, implementación. Las pruebas reales son de gran importancia para la comprobación completa de alguna teoría ya que de esta manera se pueden observar detalles fuera de la parte matemática, ahora se considera qué tan factible es su implementación, qué tan robusto es el modelo y el sistema en general para soportar variaciones que probablemente no se hayan considerado. Los experimentos en tiempo real proveen de información de la implementación que, aunada a la información que ya se tiene de simulación, pueden ser premisas para una conclusión más completa sobre el funcionamiento del sistema.

La robótica móvil hoy en día, teniendo en consideración todos los aspectos mencionados para su diseño, fabricación y control, tienen un gran campo de aplicación, ya sea como robots de servicio transportando medicinas o comida en el área médica, limpiando automáticamente supermercados, aeropuertos, como asistentes para el campo, utilización en ambientes peligrosos, en aplicaciones espaciales o militares, entre otras. Por esta razón se resalta la importancia del desarrollo de este tipo de teorías y la generación de pruebas consistentes.



En este trabajo se realiza la experimentación para comprobar una teoría acerca de deformación de trayectorias reactiva [5], esto quiere decir que se aplican algoritmos de deformación no sólo del lugar geométrico de la trayectoria, sino del perfil de velocidad del robot para que éste se aleje de ella lo menos posible, en espacio y en tiempo, considerando, claro está, sistemas estáticos y dinámicos.

2. Trabajo previo

Podemos observar aproximaciones similares al trabajo realizado en [9] y [10], donde se realizan experimentos en tiempo real con dos robots (Dala y CyCab) y sobre un robot tipo tráiler respectivamente, sobre deformación de caminos. Esto es un paso anterior al trabajo presente puesto que se deforma el lugar geométrico a seguir sin considerar la parte temporal. Obtienen datos de los obstáculos mediante sensores laser y la posición del robot con odometría. La implementación del sistema es distinta sin embargo obtienen resultados satisfactorios para la evasión en ambos casos.

Un paso importante de la evolución de esta teoría sobre deformación se da en [11], donde se pasa de la deformación de caminos (path deformation) a la deformación de trayectorias (trajectory deformation), la cual la realizan en 2 etapas, evasión de colisión y conectividad entre puntos. El paso dos con la finalidad de que se mantenga posible la deformación realizada en el paso uno. Los resultados se presentan a nivel simulación y pruebas a computadora.

En [12] se plantea un problema más similar al planteado en el presente trabajo, donde se realiza deformación reactiva de trayectorias para recorrer ambientes dinámicos con resultados de simulación satisfactorios, generando un algoritmo capaz de ser parte de un sistema de navegación en sistemas robóticos. El aporte es que se presenta una aproximación de una sola etapa, a diferencia de [11].

Como continuación del trabajo de [10] se encuentra publicado [13], donde se realiza la misma evolución que en [11], las pruebas se realizan mediante sensores láser que encuentran en el plano los obstáculos y calculan la posición y velocidad del robot mediante odometría. Los resultados son favorables.

3. Planteamiento del problema

La necesidad de experimentación en sistemas de control para robótica móvil a nivel laboratorio ha impulsado el desarrollo de sistemas tipo banco de pruebas para comprobar teorías. El problema a resolver es el desarrollo de pruebas experimentales para una teoría sobre deformación de trayectorias en robótica móvil. No sólo esto, sino el desarrollo de un protocolo de pruebas capaz de realizar simulaciones en tiempo real utilizando visión artificial y algoritmos programados en Simulink.



4. Objetivo

El objetivo del presente trabajo es el desarrollo de pruebas experimentales para la teoría de deformación de trayectorias reactiva presentada en el capítulo VII de [5]. Básicamente es el desarrollo de pruebas en lazo cerrado de control (con datos experimentales y de simulación) y tomando los datos reales de los obstáculos puestos en el espacio de trabajo (fijos y móviles) para generar la deformación de la trayectoria y por ende la evasión de los obstáculos en tiempo real.

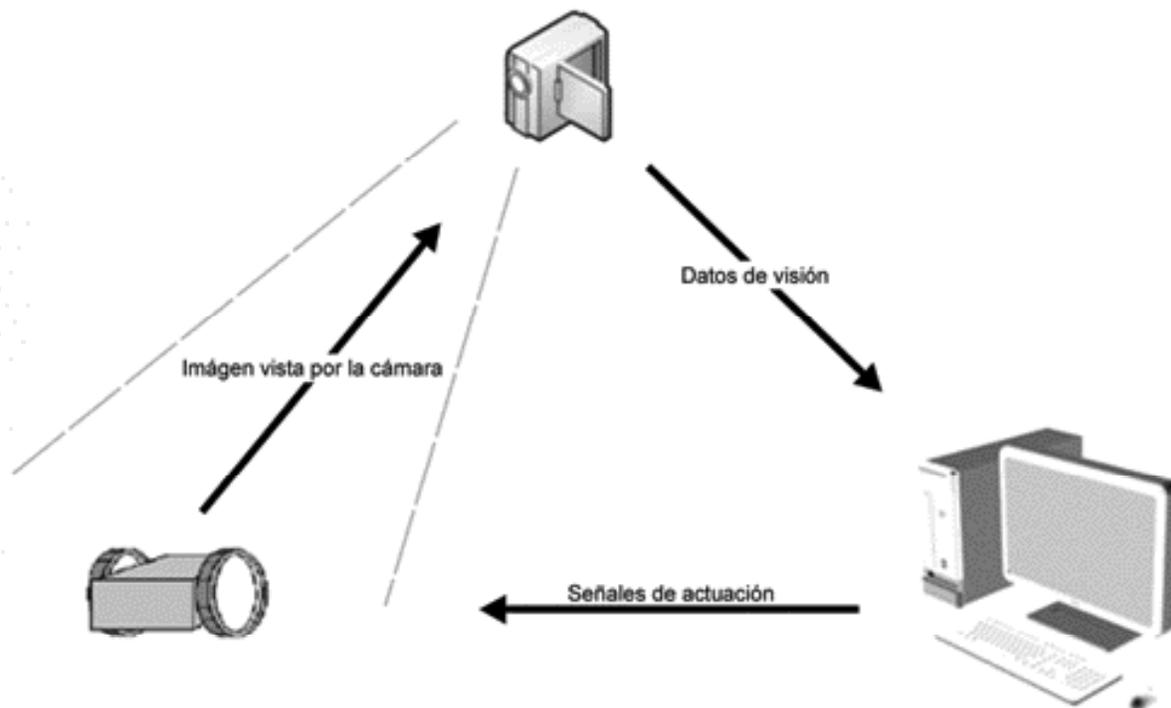


Figura 1: Diagrama objetivo del sistema que realizará el experimento

5. Contribuciones

Las contribuciones de este trabajo son amplias. En primera instancia y siendo la más directa, la comprobación de la teoría de deformación reactiva de trayectorias de [5]. Este experimento proveerá de resultados prácticos que prueben el funcionamiento del algoritmo de deformación de trayectorias y de control del robot móvil.

En segundo plano, establecemos una comunicación de software (ReacTIVision y Simulink) con la cual se obtienen datos operables para cualquier tipo de algoritmo o prueba. Esto implica un banco de pruebas, similar al esquematizado en la *Figura 1*, en el cual se pueden montar diversos tipos de robots,

mientras las restricciones físicas lo permitan claro está, para probar seguimientos de trayectorias, cerrar lazos de control, o alguna otra aplicación donde datos de posición de algún sistema, obtenidos mediante visión, puedan ser útiles.



Al haber encontrado ciertas restricciones propias del sistema al momento de realizar las pruebas, contribuimos en plantear el camino a seguir para el desarrollo futuro. Cabe recalcar que este trabajo no fue el primero en el campo, está basado en tesis anteriores y, de igual forma que esos trabajos nos dieron pauta y orientaron para abordar los problemas que se presentaron en el desarrollo del proyecto, esta tesis servirá de base para ser usada en aplicaciones futuras, incluso en áreas fuera de la robótica móvil, como robótica paralela o serial.

6. Organización

El presente trabajo está dividido en 5 capítulos:

I INTRODUCCIÓN. Se da un marco general del problema a abordar, del área donde se desarrolla el trabajo, el objetivo de la tesis, los trabajos previos y la importancia del desarrollo del presente.

II PLANEACIÓN DEL MOVIMIENTO Y CONTROL DEL ROBOT. Presenta la teoría desarrollada en [5], la cual es la base de este trabajo. Ésta define los parámetros del experimento desarrollado y los resultados esperados del mismo.

III IMPLEMENTACIÓN DEL SISTEMA. Describe a detalle la manera en que se aborda el problema y el desarrollo de las soluciones propuestas para generar el producto final.

IV EXPERIMENTOS Y RESULTADOS. Se presentan los parámetros bajo los cuales se desarrollan las pruebas de la teoría y del sistema a la vez, y los resultados de estas pruebas. Contiene además la discusión de estos resultados.

V CONCLUSIONES Y TRABAJO FUTURO. Plantea la conclusión sobre el trabajo realizado y da pauta para la continuación del trabajo presentado.

VI APÉNDICES.

1. ReactIVision
2. Código de function S de visión
3. Acoplamiento de velocidades
4. Correspondencia de velocidad real y comando MD25
5. Arduino BT
6. Tarjeta drivers MD25



II. Planeación de movimiento y control del robot

1. Deformación reactiva de trayectorias

La navegación en espacios cerrados no es una tarea trivial. El objetivo en general de cualquier tipo de navegación, incluida la de robots, es seguir una trayectoria deseada con la finalidad de llegar a un punto determinado. Esta tarea es secundada por la evasión de los obstáculos a través del seguimiento para evitar colisiones indeseadas. La conjunción de estas dos tareas genera la idea básica de la navegación de robots en espacios cerrados, es decir las etapas necesarias para cumplir la tarea: el seguimiento de una trayectoria, evitando colisiones, con la finalidad de alcanzar un objetivo.

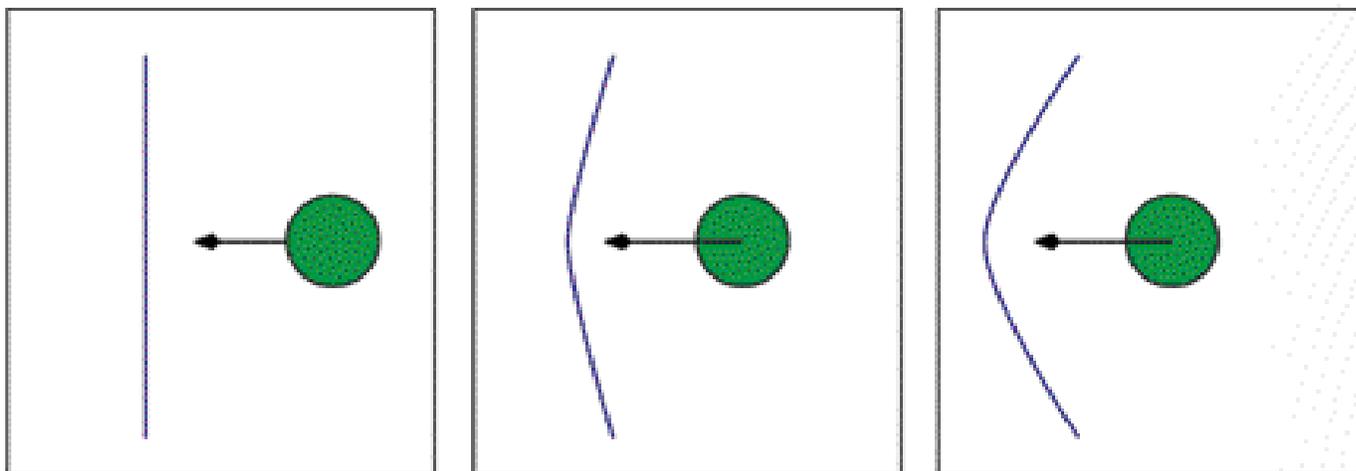


Figura 2: Representación de la deformación de trayectorias

La *Figura 2* representa la deformación de trayectorias, con la finalidad de evadir obstáculos. El problema fundamental es generar una trayectoria distinta de manera reactiva con base en 2 elementos: el obstáculo (entiéndase posición y tamaño) y la trayectoria inicial (original) que se busca seguir.

Este principio está basado en una función potencial, la cual es construida con la suma de un campo potencial atractivo y otro repulsivo. El campo potencial atractivo tiene un mínimo local en el objetivo y va creciendo entre más lejos nos encontremos del mismo. El campo potencial repulsivo tiene un mínimo cuando la distancia entre el obstáculo y el robot es mayor a cierto parámetro (radio de influencia del robot + radio de influencia del obstáculo) y va creciendo mientras sea menor.

Al ser un marco de referencia linealizado por una realimentación de estados, los vectores del campo potencial pueden ser interpretadas como fuerzas. Estas “fuerzas artificiales” se pueden cambiar por “metas artificiales” definidas en el “espacio de las metas”, donde ahora no interpretamos vectores sino posiciones, posiciones virtuales las cuales tenemos que seguir y recalcular en cada iteración para alcanzar el objetivo sin colisionar con los obstáculos.



Con esto se elimina el problema de encontrar la función potencial correspondiente, ahora sólo es necesario definir las metas atractivas y repulsivas.

En el dominio de las metas el problema de guiar al robot de un punto inicial a uno final se descompone en 2 partes:

- Alcanzar una meta
- Evitar los obstáculos

La suma de los efectos de ambas define la meta virtual que debe ser alcanzada en la siguiente iteración. Esta suma mencionada se reduce a la suma de dos posiciones, la atractiva y la repulsiva. La atractiva definido por dos puntos, el actual y la meta, y la repulsiva mediante una sumatoria, ya que puede existir más de un obstáculo.

Traduciendo esto a ecuaciones matemáticas obtenidas de [5] obtenemos lo siguiente:

Los vectores de los puntos inicial y final virtuales son obtenidos mediante:

$$\bar{Z}_d = \frac{X_d - Z_f}{\|X_d - Z_f\|}$$

$$\bar{Z}_{oi} = \frac{X_{oi} - Z_f}{\|X_{oi} - Z_f\|}$$

La parte de alcanzar una meta:

$$\bar{U}_n = G_a(Z_f, X_d)\bar{Z}_d - Z_f = X_d - Z_f$$

Donde X_d es la trayectoria de referencia y Z_f es el punto actual del robot.

La evasión de los obstáculos como:

$$G_{ri}(Z, X_{oi})\bar{Z}_{oi} = \begin{cases} \mu_i \left(\frac{1}{\rho_{ci} - \rho_{ri}} - \frac{1}{\rho_{oi}} \right) \bar{Z}_{oi} & \text{si } \rho_{ci} \leq \rho_{ri} + \rho_{oi} \\ 0 & \text{si } \rho_{ci} > \rho_{ri} + \rho_{oi} \end{cases}$$

Donde μ_i es una ganancia constante, ρ_{oi} representa el radio de influencia del robot. ρ_{ri} el radio de influencia del obstáculo, todos para el obstáculo i .



Al final, la trayectoria deformada se expresa, como en un principio se menciona, como la suma de las acciones de seguimiento y evasión, traduciendo a funciones matemáticas:

$$\overline{G}_{ref} = \overline{G}_d(Z_f, X_d) + \overline{G}_r(Z_f, X_{01}, \dots, X_{0n})$$

Cabe mencionar que la finalidad de este trabajo es probar esta teoría y no su desarrollo. Como bien se define anteriormente la teoría se obtiene de [5] y se describe lo que se cree necesario para el entendimiento del trabajo en su totalidad.

2. Robótica móvil

2.1 Introducción a la robótica móvil

La robótica móvil es la parte de la robótica encargada del estudio de los robots móviles. En general, el esquema de cualquier robot se puede describir mediante la *Figura 3* [6]:

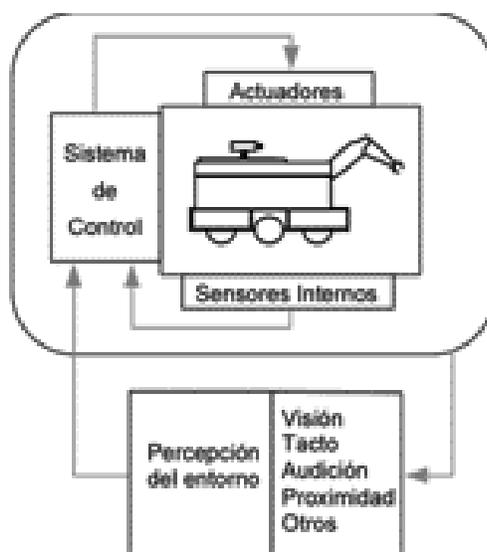


Figura 3: Robot y su interacción con el entorno

Donde podemos distinguir al robot y la manera en la que interactúa con su entorno, sea esta interacción como salida a actuadores o entrada a sensores. El diagrama sugiere distintas maneras de percibir el comportamiento del entorno, visión, tacto, audición, proximidad, u otros. Esta decisión dependerá en gran parte del tipo de entorno en el que estemos trabajando y del tipo de tarea que el robot tiene que realizar. Existen en el diagrama sistemas implícitos como por ejemplo el de comunicación.

Estos robots muchas veces realizan su procesamiento en otro dispositivo, o no trabajan solos, sino en un ambiente con otros robots, por lo que es de gran importancia el envío y recepción de datos a otros sistemas para un mejor desempeño.



Todas las partes que conjuntan al robot tienen una tarea esencial para su correcto funcionamiento y el desarrollo de la tarea programada.

Al ser un robot una "máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas sólo a las personas" [7] entre más versátil sea el sistema mayor será la gama de tareas para las que se pueda programar. Por lo tanto el diseño de estos sistemas debe considerar siempre el tipo de tarea, el ambiente donde operará, etcétera.

Hablando de manera particular de la robótica móvil, ésta estudia la cinemática y dinámica de los robots con capacidad de desplazarse dentro de un espacio de trabajo considerando que no hay vínculos mecánicos entre el robot y la base de referencia fija/inercial [8].

Esta definición es claramente sobre el estudio de robots móviles, los cuales deben cumplir con ciertas características como son:

- Movilidad: se refiere a la capacidad de movilidad relativa con el ambiente.
- Cierta nivel de autonomía: interacción humana limitada.
- Habilidad de percepción: sensar y reaccionar dentro de un ambiente de trabajo.

El sistema robótico móvil se expresa mediante la posición en el plano de un punto, llamado punto P, (x,y) , y la orientación del robot móvil, (θ) , generando una terna de datos que definen al robot en el plano, (x,y,θ) . Este punto P se encuentra sobre el robot y se mueve con él, y es precisamente este punto el que se referencia a la base inercial (fija) mencionada anteriormente. Con base en estas relaciones y las relaciones considerando la arquitectura del robot es como se consigue un modelado matemático del sistema dentro de su ambiente.

2.2 Tipos de robots móviles

Como podemos suponer existen diversos tipos de robots móviles, esto dependiendo de la configuración física del mismo, y, siendo más específicos, existen también distintos modelos matemáticos de estos robots, dependiendo de las dimensiones del robot, tipo de configuración de llantas, posición del marco de referencia del robot, definición del marco de referencia de las llantas, etcétera.

Al ser el tipo de rueda un factor importante para definir el tipo de robot con el que estamos trabajando, es necesario definir antes los tipos de ruedas con las que generalmente están equipados los robots móviles.



Para definir a las ruedas se plantea la siguiente clasificación:

- Convencionales.
- Suecas.

En ambos casos tenemos la consideración de que el contacto entre la llanta y el suelo es en un sólo punto. En el caso específico de las convencionales, existe rodamiento puro, sin deslizamiento.

De las llantas convencionales, las cuales son las de nuestro interés, podemos subdividir las en:

- Fijas
- Centradas orientables
- Descentradas orientables

Las llantas fijas son aquéllas que no cambian su posición angular con respecto al marco de referencia del robot.

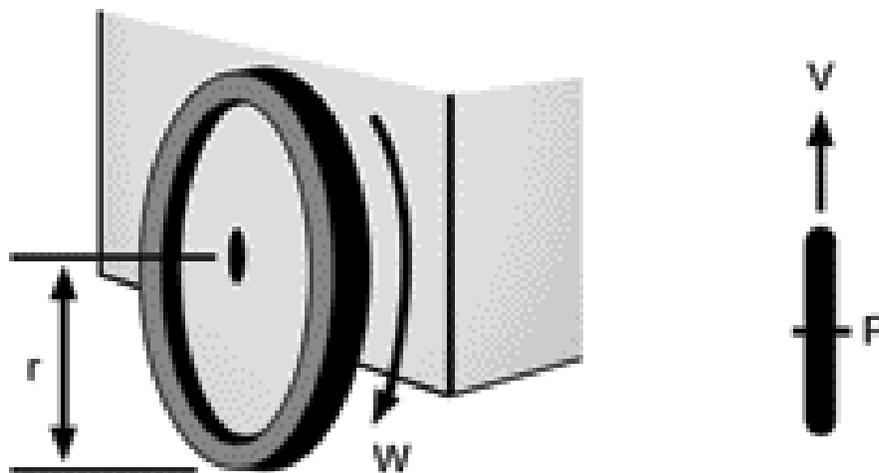


Figura 4: Llanta fija

En la vista superior y referenciada la posición de la llanta con respecto al marco de referencia del robot mediante posición y orientación, (x,y,α) , el ángulo α se mantiene constante, por lo que ahora el punto donde se encuentre la llanta está definido sólo por dos coordenadas, llamadas en este caso (d,b) .

Las centradas orientables se comportan de igual manera que las fijas sólo que ahora el ángulo α es variable. El eje de giro de la orientación de la llanta (punto P en la Figura 5) atraviesa al eje de giro de la rueda.



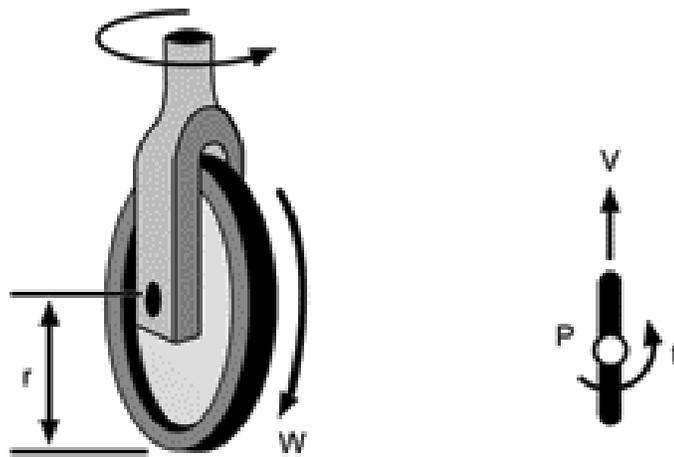


Figura 5: Llanta centrada orientable

Las descentradas orientables tienen las mismas características que las centradas orientables sólo que ahora el eje de giro de la orientación de la llanta no cruza con el eje de giro de la rueda, como se puede observar en la Figura 6.

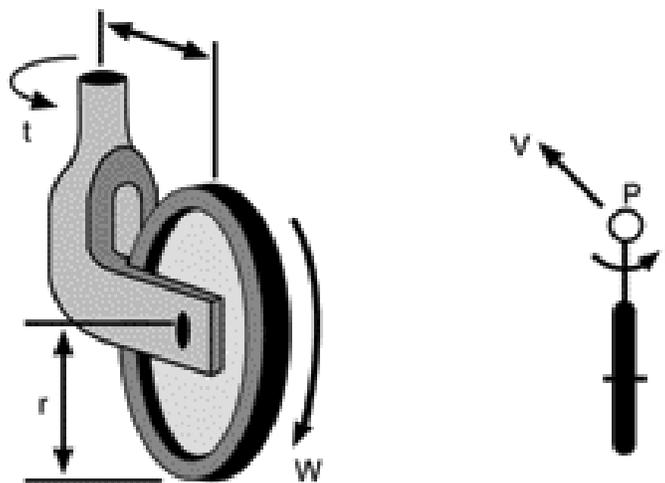


Figura 6: Llanta descentrada orientable

Es claro que los distintos tipos de llantas generan distintos tipos de modelos matemáticos y de restricciones cinemáticas de los robots. Dependiendo del tipo de ruedas utilizadas en el robot podemos describir ciertas arquitecturas que son las generalmente utilizadas para la robótica móvil:

- Direccionamiento diferencial
- Ackerman
- Triciclo
- Síncronos
- Omnidireccionales (ruedas suecas)



El robot de direccionamiento diferencial (*Figura 7a*) se basa sólo en dos llantas que proveen tracción y, mediante velocidad diferencial entre ellas, logra su direccionamiento. El Ackerman (*Figura 7b*) se basa en ruedas de direccionamiento (tipo automóvil) en las cuales la rueda interna gira de distinta manera que la externa, basado el giro en parámetros como el centro instantáneo de rotación (ICR, por sus siglas en inglés). El triciclo (*Figura 7c*) utiliza tres ruedas, dos traseras y una delantera. La delantera provee la direccionabilidad y, dependiendo de la configuración, la potencia puede ser provista por la delantera o por las traseras. El síncrono (*Figura 7d*) es capaz de direccionar e impulsar cada una de sus ruedas. Generalmente se usan en configuración de triángulo (3 ruedas claramente) y de rectángulo (4 ruedas). Los omnidireccionales tienen ruedas suecas, con las cuales pueden desplazarse hacia cualquier lado del plano sin necesidad de rotación del sistema definido en el robot.

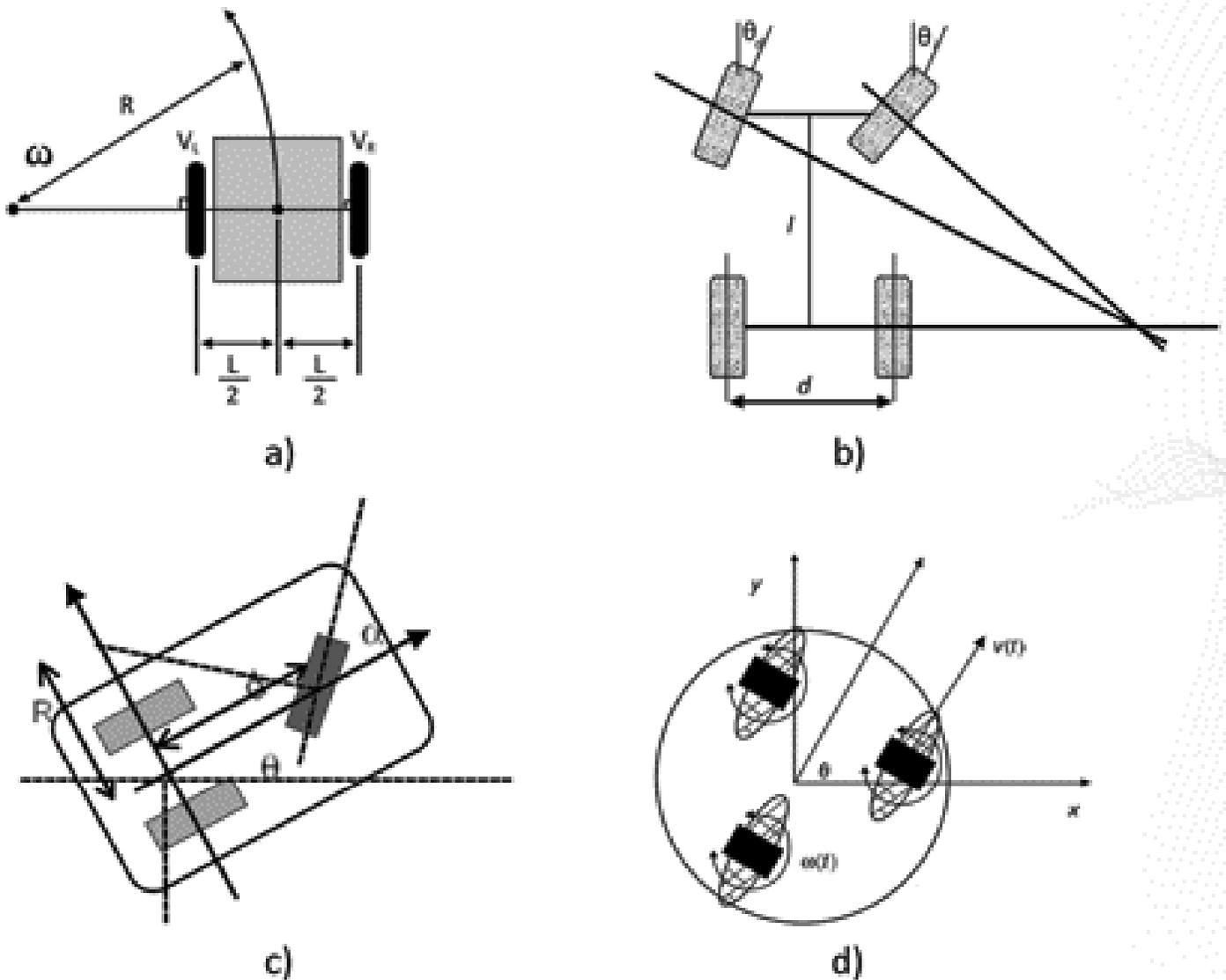


Figura 7: Arquitecturas usuales de robots móviles



A partir de las matrices asociadas con las restricciones cinemáticas ortogonales al plano de las ruedas y las matrices asociadas a las restricciones cinemáticas sobre el plano de las ruedas se puede definir una matriz de la cinemática interna del robot [5]. Con el análisis de esta matriz (por supuesto con los cálculos adecuados) podemos definir propiedades como:

- Grado de movilidad
- Grado de direccionabilidad
- Grado de maniobrabilidad

El grado de movilidad define el número de variables que son libres para controlar, el grado de direccionabilidad define cuántas llantas centradas orientables están libres para controlar y el grado de maniobrabilidad define el número de grados de libertad totales que el robot puede manipular.

Con los dos primeros (movilidad y direccionabilidad) es posible definir el tipo de robot con el que estamos trabajando. Cabe recalcar que el grado de maniobrabilidad es la suma de los grados de movilidad y de direccionabilidad.

El tipo de robot está definido mediante (r_m, r_s) , donde r_m es el grado de movilidad, r_s es el grado de direccionabilidad y r_M es el grado de maniobrabilidad. No es suficiente r_M para definir el tipo del robot, ya que pueden existir combinaciones de r_m y r_s que nos darían el mismo grado de maniobrabilidad.

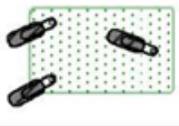
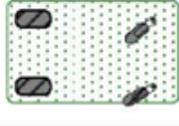
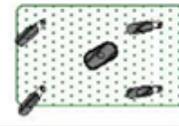
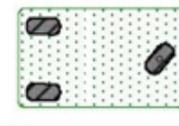
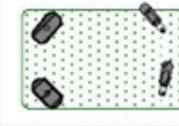
| | | Type(r_m, r_s) | | | | | |
|---------------------|-------|---|---|---|---|---|---|
| | | (3,0) | (2,0) | (2,1) | (1,1) | (1,2) | |
| Degree | r_m | 3 | 2 | 2 | 1 | 1 | |
| | r_s | 0 | 0 | 1 | 1 | 2 | |
| | r_M | 3 | 2 | 3 | 2 | 3 | |
| Robot configuration | |  |  |  |  |  | |
| Motorized wheels | | Fixed |  | Centred |  | Off-centred |  |
| Unmotorized wheels | | | |  | Passive off-centred | Passive centred |  |



Figura 8[5]: Tipos de robots móviles

El grado de direccionabilidad está restringido a $r_s = 2 - r_f$ donde $r_f \leq 1$ es el número de ejes comunes de las llantas fijas.

Los robots con $r_m = 3$ son considerados omnidireccionales ya que pueden definir el centro instantáneo de rotación en cualquier posición con libertad, por lo que pueden moverse en cualquier dirección en el plano.

Los de $r_m \leq 2$ tienen movilidad restringida puesto que sus movimientos están restringidos a ciertas direcciones en el plano. Los robots móviles más usuales, definidos por tipos se presentan en la *Figura 8*.

2.3 Robot usado para la prueba

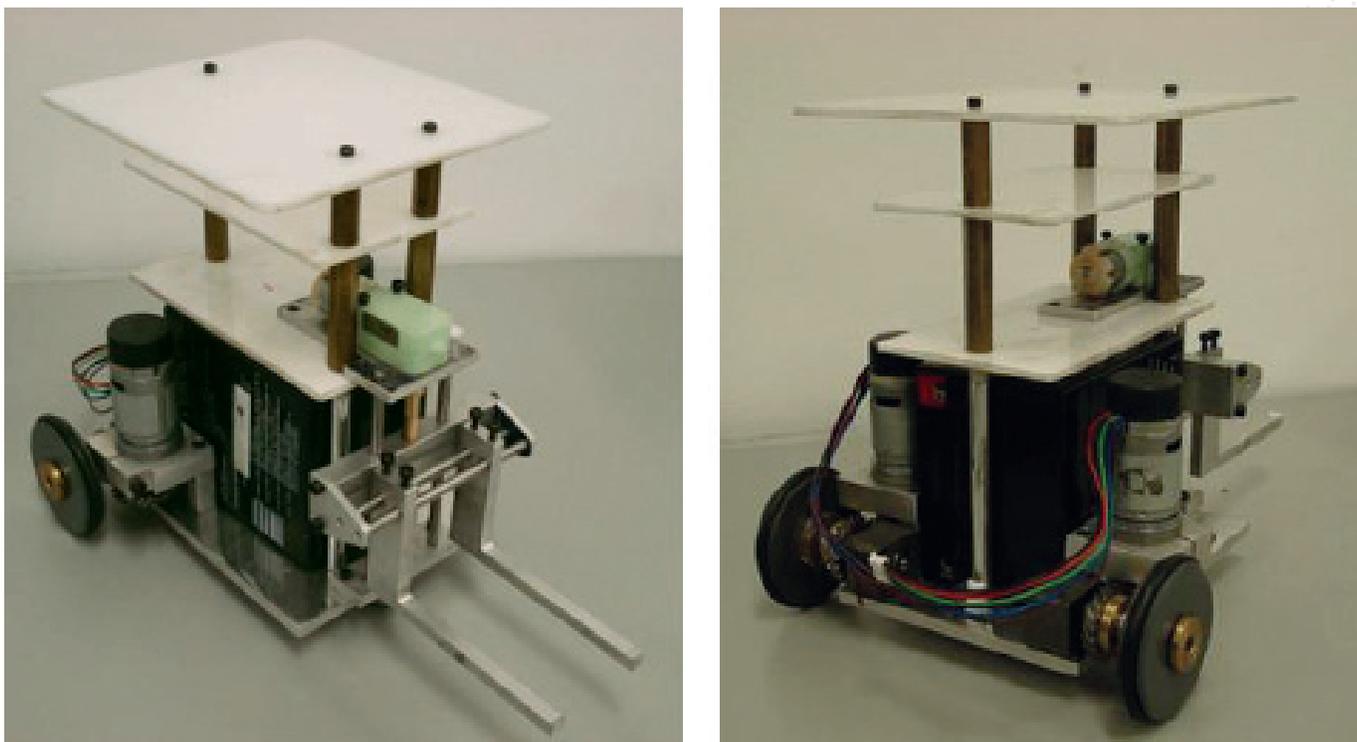


Figura 9: Robot usado en la prueba

El robot de la *Figura 9* es utilizado para la prueba del presente trabajo. Es claro que es un robot por diferencial eléctrico, tipo (2,0) utilizando los conceptos mencionados con anterioridad.

El robot es tipo montacargas, con un balín en la parte de enfrente (análogo a una rueda loca no direccionable) con el objetivo de soporte. Cuenta con pisos de acrílico para la colocación de cualquier circuito necesario para controlar las velocidades de las llantas y manejar la comunicación con la computadora. Los motores son de corriente directa, modelo EMG-30, que contienen encoders de cuadratura internos de 360 pulsos por vuelta acoplados al eje. La batería utilizada para estos robots es de 12 [V].



El montacargas está acoplado al sistema, sin embargo para este caso no se utilizará.

Las dimensiones necesarias para el modelado del robot son las siguientes [metro]:

| | |
|--|-------|
| Distancia entre llantas: | 0.112 |
| Distancia más corta del eje de las llantas al punto P: | 0.075 |
| Radio de las llantas: | 0.03 |

2.4 El robot móvil por diferencial eléctrico

Como ya vimos anteriormente, el robot a utilizar para la prueba es de tipo (2,0). Esto quiere decir que tiene dos llantas fijas sobre el mismo eje y no tiene llantas direccionables. Su direccionamiento está basado en la velocidad diferencial entre la llanta derecha e izquierda.

2.4.1 Postura del robot

Asumiendo que el robot móvil es rígido, no deformable, las llantas no se deslizan y se mueve sobre un plano horizontal se pueden introducir las siguientes definiciones. Cabe recalcar que el trabajo es tomado de [5] para finalidad del experimento y no fue desarrollado en el presente trabajo.

Un robot móvil tiene dos marcos de referencia utilizados para su análisis.

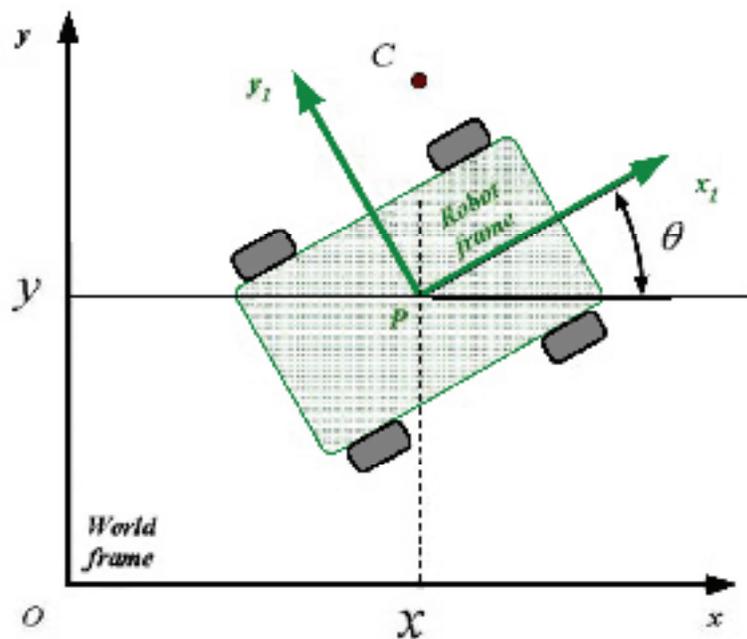


Figura 10: Esquemático de la postura de un robot móvil



El primero es la base inercial, la cual es fija y se representa en este caso por (x,y) en la *Figura 10*. El segundo es móvil y relativo a la posición del punto P del robot, descrito por (x_1,y_1) . Por lo tanto la postura del robot se representa por:

$$\xi = [x \quad y \quad \theta]$$

Donde ξ describe la posición del punto P y la orientación del marco de referencia que está sobre el robot con respecto al inercial. Este ángulo θ está medido a partir del eje X de la base inercial al eje X del marco de referencia móvil.

Con estas relaciones se puede definir la matriz ortonormal de rotación como sigue:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

La cual nos sirve para mapear todos los vectores referenciados a la base inercial, ahora con el marco que está sobre el robot y viceversa, y no sólo de posición, sino que con esta misma matriz de rotación podemos mapear vectores de velocidad.

A partir de este planteamiento y con el desarrollo matemático necesario se obtiene la representación en variables de estado de la cinemática de la postura del robot:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & d * \sin(\theta) \\ \sin(\theta) & -d * \cos(\theta) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

2.4.2 Modelo cinemático

El modelo que se utilizará para la prueba se describe a continuación. Al igual que el apartado anterior el modelado fue tomado de [5] para su uso en el experimento, no fue desarrollado en este trabajo.

Se propone la siguiente configuración representada en la *Figura 11*: el robot tipo (2,0) tiene dos llantas fijas y ruedas pasivas, sea una o dos las cuales se excluyen del análisis.



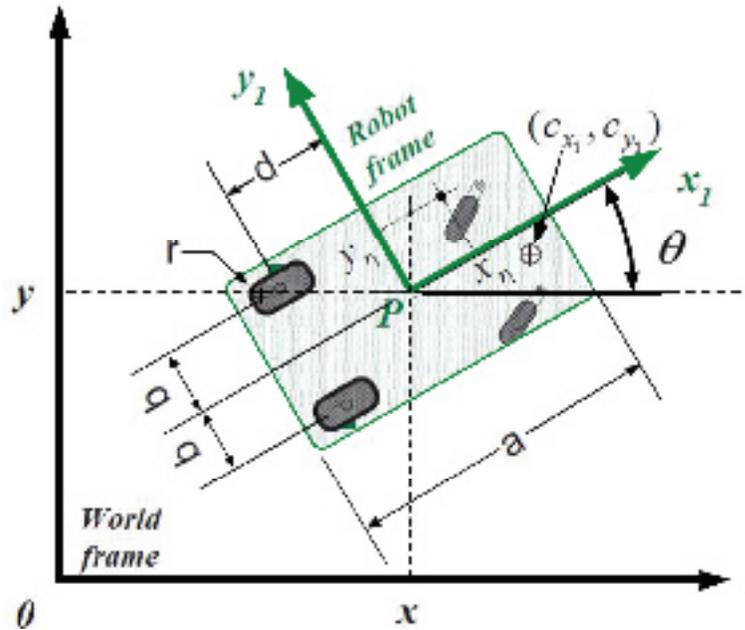


Figura 11: Postura del robot tipo (2,0)

Las variables utilizadas para el modelo del robot están descritas en el siguiente vector:

$$q = [x \quad y \quad \theta \quad \Phi_r \quad \Phi_l]$$

Donde x y y denotan la posición del punto P , θ la orientación del marco de referencia montado sobre el robot $\{x_1, y_1\}$ con respecto a la base inercial. Las variables Φ_r y Φ_l son los ángulos de rotación de las llantas derecha e izquierda alrededor de su eje. El radio de la llanta está definido por r (el cual ya se mencionó su valor numérico para nuestro caso en particular).

Con el desarrollo matemático necesario, desarrollado en [5], a partir de las condiciones presentadas anteriormente y el uso de las restricciones cinemáticas necesarias, se obtiene la representación en variables de estado del robot móvil (2,0) como sigue:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\Phi}_r \\ \dot{\Phi}_l \end{bmatrix} = \begin{bmatrix} \cos(\theta) & d * \sin(\theta) \\ \sin(\theta) & -d * \cos(\theta) \\ 1 & 1 \\ 1/r & b/r \\ 1/r & -b/r \end{bmatrix}$$



3. Esquema de control

El sistema contiene un controlador para llevar al robot a la posición deseada, sea ésta el objetivo final o los objetivos virtuales. Está basado en un controlador no lineal por retroalimentación de estados. Esto significa realimentar ciertas variables (variables de estado) de tal manera que, al hacer la realimentación, se cancelen las no linealidades del modelo matemático y se pueda tratar como un modelo lineal. De esta manera no es necesario linealizar alrededor de ningún punto de equilibrio, sino que la realimentación linealiza para cualquier punto, haciendo más fácil la tarea de controlar en todo el espacio de trabajo. De otra manera, la linealización del modelo cinemático cambia, y el algoritmo debería ser capaz de realizar esta tarea dependiendo del valor de las variables en ese preciso momento. Se requiere realimentación de variables de posición y orientación del robot, además de la velocidad de las mismas para la linealización y el control.



III. Implementación del sistema

1. Descripción general del sistema

El sistema consta de diferentes etapas de flujo de información:

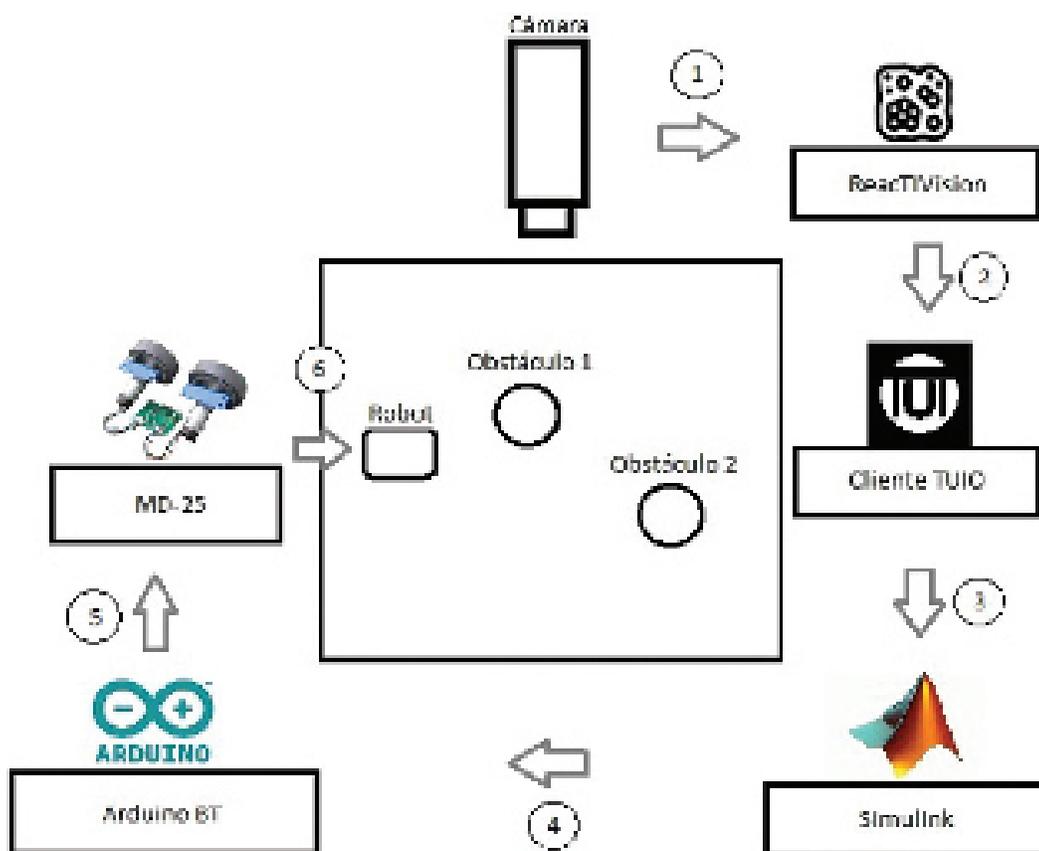


Figura 12: Diagrama esquemático del sistema

Estas etapas, esquematizadas en el diagrama presentado en la *Figura 12*, son subsistemas trabajados independientemente y luego conjuntados para su correcto funcionamiento.

El sistema está hecho para trabajar con 2 obstáculos y el robot en el campo de trabajo. Esta información es captada por la cámara, enviada por FireWire y captada mediante ReacTIVision, software "open source" para detección y envío de información sobre las figuras previamente diseñadas (amebas o fiduciales). La cámara se encuentra por encima del área de trabajo, captando las amebas que están encima de los obstáculos y del robot.

ReacTIVision no es capaz de enviar información directo a algún software, pero mediante un cliente TUIO (explicación más adelante) podemos acceder a la información, toda la que puede proveer (apéndice 1), y utilizarla en el procesamiento.

Una vez obtenida la información es procesada por el algoritmo de deformación de trayectorias y por el controlador programados, enviando comandos vía Bluetooth al robot.



El robot, provisto de un microcontrolador ArduinoBT capta las señales de control y las reenvía por protocolo I2C a la tarjeta MD-25 para llegar, como último eslabón, a los actuadores finales. Todo este proceso se realiza en tiempo real y como resultado se obtiene al robot siguiendo una trayectoria y evadiendo los obstáculos que se colocan en su camino.

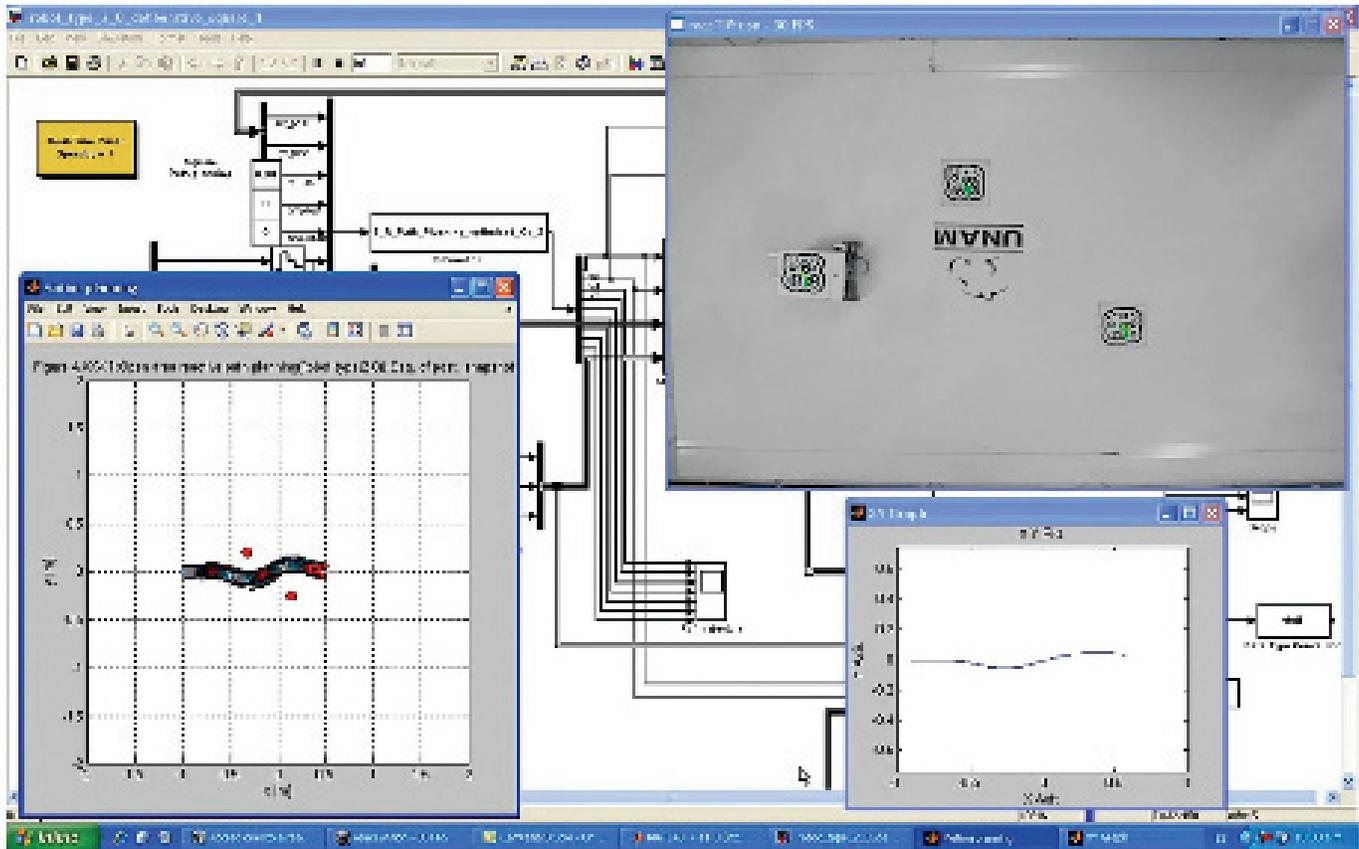


Figura 13: Captura de pantalla del sistema en lazo semi-cerrado

Como resultado de la conjunción de todos los sistemas presentados anteriormente obtenemos las distintas aplicaciones corriendo al mismo tiempo, *Figura 13*, para lograr la deformación. La cámara obtiene los datos de los símbolos fiduciales y los del robot, en este caso en particular, se despliegan en una gráfica pues no están siendo realimentados, solo mostrados. La ventana "Motion Planning" presenta, en tiempo real, los datos de visión de los obstáculos y los datos del robot provenientes del modelo cinemático, siguiendo la trayectoria deformada. Teóricamente la deformación presentada en ésta ventana debería ser exactamente igual a la obtenida por la cámara.



2. Diseño de la prueba

2.1 Deformación de trayectorias

Las ecuaciones mencionadas anteriormente (sección II.1) para la deformación de trayectorias se encuentran programadas en una función tipo S de Simulink. Como entradas necesitamos la trayectoria en ese instante, la posición del robot y la posición de los obstáculos. Con esta información obtenemos la nueva posición virtual deseada, es decir, a donde el controlador tiene que llevar al robot para ese instante. La función tipo S de Simulink nos permite temporizar esta función, aunque en este caso está siendo llamada en cada una de las iteraciones que hace el programa dada la importancia de sus resultados para terminar el recorrido. Dentro de la función podemos definir radios de influencia para los obstáculos y para el robot ya que, como se explicará adelante, los símbolos fiducial junto con ReactIVision nos envía información sobre el centroide de la figura, para conocer su posición, sin embargo no nos habla de las dimensiones del obstáculo. De esto depende qué tan cerca pasará el robot de los obstáculos. Teóricamente se podrían definir estos radios de influencia del tamaño mínimo necesario para que rodeen al obstáculo y al robot y de igual forma no habría colisión, sin embargo por ciertas consideraciones en particular de este trabajo se definen de un radio mayor para tener un mejor resultado.

Considerando la temporización de las funciones S, como ya se mencionó la que contiene las ecuaciones de deformación reactiva de trayectorias está corriendo cada iteración del programa, sin embargo depende de los datos de entrada del sistema. Estos datos provienen de realimentaciones del mismo programa y de nuevas entradas de visión, programadas en otra función S, la cual sí se encuentra temporizada por los procesos que se llevan a cabo en ella. Estos procesos requieren más recursos computacionales, por lo que si se corre cada iteración del programa no nos permite trabajar en tiempo real. La frecuencia de funcionamiento de esta función en particular fue determinada mediante experimentos. Se buscó la frecuencia específica para que la computadora tuviera tiempo para procesar la información proveniente de la cámara y enviarla al programa en Simulink, sin poner en riesgo el funcionamiento en tiempo real. Además, toda la información sobre posición, velocidad, comandos, posición de los obstáculos, entre otras variables, se guardan en Matlab para su análisis posterior.

La deformación de trayectorias trabaja claramente sobre una trayectoria, y aunque sea obvio vale la pena mencionarlo, esa trayectoria es la que se planea deformar. Estos datos deben ser programados también en el algoritmo de tal forma que se obtenga el punto real en el que tiene que estar el robot en ese instante y el objetivo virtual que se calcula mediante la deformación. En este caso se programa la trayectoria inicial mediante una parametrización en el tiempo con ciertas constantes modificables dependiendo del tipo de prueba que se quiera realizar.



2.2 ReactIVision

Se presenta el software utilizado para la visión artificial del sistema.

Por definición, la visión es la imagen que, de manera sobrenatural, se percibe por el sentido de la vista o por representación imaginativa [14]. En nuestro caso, al ser visión artificial quiere decir que la imagen percibida por un medio artificial y no por el sentido de la vista. Este medio artificial al que refiero es una cámara. Esta desempeña las funciones de ojo y de tal forma obtenemos información del área de trabajo para después ser procesada en el programa.

La adición de sistemas de visión artificial generalmente se relaciona con procesos complejos y de alta demanda de recursos computacionales, sin embargo ahora existen aplicaciones para facilitarnos la tarea y aprovechar los grandes beneficios de su uso.

ReactIVision es software libre de visión para rastrear símbolos fiducials (*Figura 14*), que surge como sensor de un sintetizador modular para aplicaciones multi-touch.



Figura 14: Ejemplos de símbolos fiducial

Es una aplicación que manda mensajes TUIO a cualquier cliente en la misma computadora mediante el puerto UDP 3333. Los clientes TUIO están de igual forma libres para descarga y en varios lenguajes de programación listos para utilizarse.

Para el caso de rastreo de símbolos fiducials funciona de la siguiente manera: la imagen original se convierte a una blanco y negro mediante un algoritmo de umbrales adaptativos. Esta imagen se segmenta en un árbol de las varias alternativas de regiones en blanco y negro, generando una gráfica. En esta gráfica es en donde se busca y se compara la secuencia de cada uno de los símbolos fiducial. Cuando

las secuencias, en el algoritmo de búsqueda en el árbol de alternativas, coinciden se le asigna su número de identificación único. Una vez detectado, la forma única del fiducial permite calcular su centroide y su orientación. Una vez obtenidos los datos pueden ser enviados mediante el protocolo TUIO.



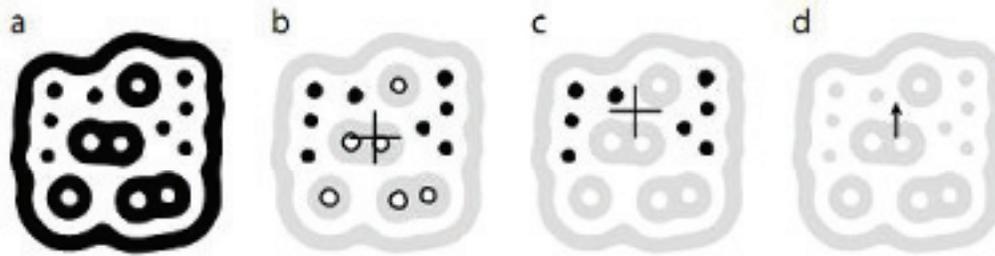


Figura 15: Reconocimiento de símbolos fiducial

El proceso de reconocimiento se realiza primero detectando al fiducial con el algoritmo descrito anteriormente (Figura 15), después se calcula el centroide con todos los círculos (blancos y negros) y ese es el centroide del símbolo fiducial. Con sólo los círculos negros se calcula un nuevo centroide y, al generar un vector de centroide a centroide se puede conocer la orientación del fiducial. Esta información es la básica que se procesa para calcular otros parámetros y se envía mediante protocolo TUIO.

2.3 Comunicación con ReactIVision

Mediante un cliente TUIO java se realiza la comunicación de Simulink con ReactIVision.

El entorno de ReactIVision (Figura 16) detecta los centroides con el algoritmo previamente explicado.



Figura 16: Entorno ReactIVision



Esta información mediante la aplicación (Figura 17) es enviada a algún cliente TUIO que se encuentre instalado y listo para recibir la información. La aplicación, como se muestra en la figura es sencilla (no en funcionamiento sino en interfaz) y nos permite variar distintos parámetros dependiendo de la aplicación. Con estos parámetros podemos definir de manera correcta el espacio de trabajo y el orden de las variables (dirección en la que crece X, Y o el ángulo) para que coincidan los datos obtenidos de visión con los datos de simulación, parte crucial para que el controlador funcione correctamente.

Teniendo disponibles los datos ya en memoria de la computadora el problema es acceder a ellos mediante simulink, para procesarlos en la deformación de trayectorias.

Este problema se resolvió con el código presentado en el apéndice 2. El primer paso es importar el cliente TUIO a utilizar. Como anteriormente se menciona ReactIVision permite utilizar clientes TUIO para diferentes lenguajes de programación (todos disponibles para su descarga gratuita en la página oficial de ReactIVision [16]) como los siguientes:

- C++
- Java
- C#
- Processing
- Pure Data
- Max/MSP
- Quartz Composer

```

C:\Users\Alejandro\Documents\Alejandro\Tesis...
reactIVision 1.4 (May 20 2009)
camera: WDM Camera
format: 640x480, 24fps

display:
n - no image
s - source image
t - target image
h - toggle help text

commands:
ESC - quit reactIVision
v - verbose output
o - camera options
p - pause processing

FrameEqualizer:
e - toggle frame equalizer
SPACE - activate frame equalizer

FrameThresholder:
g - adjust gradient gate

FiducialFinder:
i - invert x-axis, y-axis or angle
FidtrackFinder:
f - adjust finger size & sensitivity

CalibrationEngine:
c - toggle calibration mode
j - reset calibration grid
k - reset selected point
l - revert to saved grid
r - show calibration result
a,d,w,x - move within grid
cursor keys - adjust grid point
  
```

Figura 17: Aplicación ReactIVision



Matlab soporta Java, por lo que se utiliza este cliente TUIO para tener acceso a los datos. Para poder utilizar la biblioteca incluida con este cliente y a la vez todas las propiedades de los objetos a crear debemos agregar al "path" de Matlab las bibliotecas necesarias del cliente TUIO Java. Este proceso es sencillo y se describe a continuación:

1. Descargar el archivo de TUIO_JAVA de la página de ReactIVision.
2. Salvar la carpeta que extraemos en alguna dirección de donde no la cambiemos.
3. En la ventana de comandos de MatLab (Command Window) abrimos el archivo de "classpath.txt" para editarlo mediante el comando "edit classpath.txt", recordar ejecutar como administrador para que nos permita salvar este archivo.
4. Una vez abierto el archivo de texto debemos agregar dos líneas: las direcciones de los archivos "TuioDemo.jar" y "libTUIO.jar", los cuales se encuentran en la carpeta descargada. La dirección se tiene que colocar con toda la ruta para el acceso a los mismos.
5. Guardamos el archivo y cerramos.

Este proceso hace a Matlab capaz de ejecutar ciertos comandos para tener acceso a los datos de ReactIVision mediante el cliente TUIO Java.

Dentro de la función para obtener los datos de visión se siguen varios pasos para poder utilizar los comandos definidos para ReactIVision. Esto es: agregar la biblioteca a esa función (diferente a agregarla al "path" de Matlab), crear un objeto para que obtenga las propiedades definidas en apéndice 1, de las cuales claramente no se utilizarán todas. Al final se conecta este objeto creado a la aplicación de ReactIVision para obtener los datos requeridos. Esto se traduce a los comandos siguientes, incluidos por supuesto en la función de visión (apéndice 2):

- import TUIO.*;
- tc = TuioClient();
- tc.conect();

El objeto está creado y conectado a la ReactIVision. Dependiendo de la aplicación los datos necesarios pueden ser diversos. Para este caso en particular se requieren solamente posiciones: X y Y de ambos obstáculos y del robot y la posición angular del robot.

Antes de obtener estos datos es necesario diferenciar los símbolos fiducial (si es del robot, del obstáculo 1 o del obstáculo 2). Esto se hace mediante el número de fiducial. Existen 215



diferentes símbolos fiducial, con un número de identificación (ID) distinto (archivo disponible para descarga de la página de ReactIVision), con el cual podemos diferenciar cada uno de nuestros símbolos. En este caso se utilizaron los símbolos 1, 3 y 4 para el robot y los dos obstáculos respectivamente como se presenta en la *Figura 18*. Este número (1, 3 y 4) es el que identifica al fiducial, es decir, identifica si es robot u obstáculo.



Figura 18: Símbolos fiducial utilizados para la prueba

Abordaremos primero el caso de los obstáculos por orden de dificultad. Para ambos sólo necesitamos su posición en el campo de trabajo. Al momento de detectar el fiducial, sea el 3 o el 4, obtenemos la posición en X y en Y del centroide del fiducial con los comandos siguientes:

- `X = tc.getTuioObjects().get(#).getPosition().getX();`
- `Y = tc.getTuioObjects().get(#).getPosition().getY();`

El símbolo “#” denota un orden o secuencia de aparición de símbolos fiducial. Si se coloca 1 quiere decir que se guardará la información del primer símbolo fiducial en encontrar. Esto depende de la colocación en el área de trabajo, por esta razón este número debe coincidir con el de la aparición al momento de solicitar el número de símbolo fiducial.

De igual manera se obtiene la información para el segundo obstáculo.

Para el robot es necesaria mayor información. En nuestro caso requerimos 3 posiciones (2 lineales y una angular) y 2 velocidades (lineal y angular del robot) para hacer las pruebas. Al trabajar con

velocidades tenemos diversas maneras de obtenerlas, directamente de ReactIVision, con datos de velocidad en X y en Y de ReactIVision y manipulación algebraica, derivación de la posición y manipulación algebraica, entre otros, sin embargo es difícil definir qué



método nos arrojará los mejores datos. Esta es la razón principal por la que se obtienen los datos de todos los métodos para compararlos en pruebas experimentales y tomar una decisión fundamentada en resultados. Se obtienen las posiciones (3) y las velocidades (4) mediante los comandos siguientes:

- `A = tc.getTuioObjects().get(#).getAngle();`
- `X = tc.getTuioObjects().get(#).getPosition().getX();`
- `Y = tc.getTuioObjects().get(#).getPosition().getY();`
- `V = tc.getTuioObjects().get(#).getMotionSpeed();`
- `VX = tc.getTuioObjects().get(#).getXSpeed();`
- `VY = tc.getTuioObjects().get(#).getYSpeed();`
- `VA = tc.getTuioObjects().get(#).getRotationSpeed();`

Después de realizar la comunicación mediante el objeto creado debemos terminar el enlace entre software que creamos usando:

1. `tc.disconnect();`

2.4 Acoplamiento de datos

Los datos obtenidos del proceso explicado en la sección anterior no necesariamente son los que requerimos para el procesamiento de la deformación reactiva de trayectorias y los obtenidos en el procesamiento tampoco pueden ser enviados directamente. Existen ciertos detalles a resolver antes de poder usar estos datos, se mencionan a continuación:

1. Los datos de posición (en X y en Y) están escalados de 0 a 1.
2. El valor del ángulo sólo va de 0 a 2π .
3. El cero del sistema (0,0) se encuentra en una esquina
4. Las señales de control deben ser enviadas en una escala distinta por el tipo de driver de los motores
5. Las señales de control (velocidad angular a las llantas) pueden superar los permitidos por los motores
6. Los valores de velocidad provenientes de derivadas no pueden ser utilizados directamente por la temporización de la función de visión



Los problemas de escalamiento son fácilmente corregibles y no serán parte de la explicación de este trabajo.

El valor del ángulo de giro del robot debe considerar valores negativos y superiores a 2π , esto se traduce a conocer si el robot ha dado o no una vuelta. Por las condiciones de las pruebas donde la velocidad es muy baja, las diferencias grandes en el ángulo entre el dato anterior y el actual sólo pueden provenir de una vuelta (sea sentido horario o antihorario). Bajo esta suposición la función se programa realimentando el ángulo anterior guardado en memoria y restándolo con el ángulo actual. Dependiendo del valor absoluto de la diferencia podemos suponer que giró (pasó de 0 a 2π o viceversa) y con el signo obtenemos el sentido de giro. Esta información se guarda en un contador de vueltas el cual también se realimenta cada vez que llamamos a la función y se le suma o resta la unidad dependiendo si realizó el giro y en qué sentido. El valor del contador se multiplica por 2π y se suma al valor que tenemos de ReacTIVision, obteniendo así el valor real del ángulo del robot.

La función con las correcciones mencionadas se presenta en la *Figura 19* y su código en el apéndice 2.

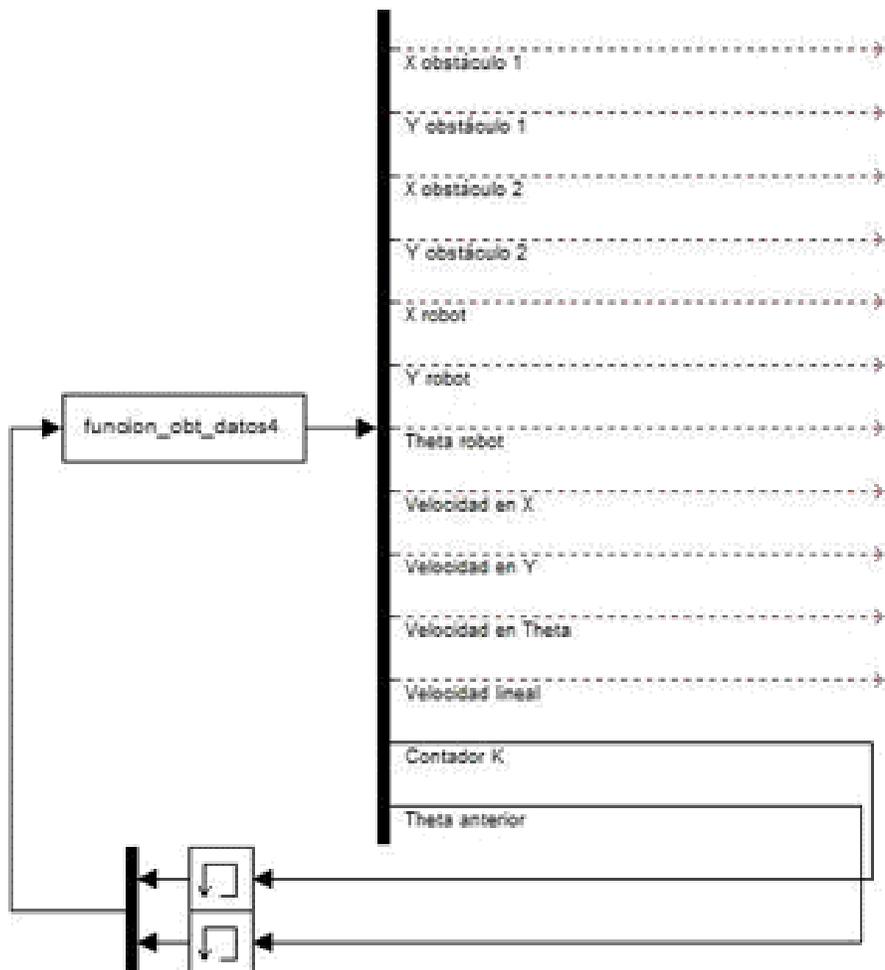


Figura 19: Función para obtener datos de visión



Para el problema del escalamiento de las señales de control a los motores del robot móvil es, de igual forma que para la visión, relativamente sencillo y no será abordado. Sólo se menciona la recta a la cual se aproximó la relación de primer orden entre los parámetros de velocidad de salida del algoritmo y la velocidad que maneja la tarjeta de MD25:

$$y = 6.4081x + 127.67$$

Obtenida de manera experimental. Los datos y la aproximación mediante regresión lineal se presentan en el apéndice 4.

El mantener estas señales dentro de rango es una tarea más exigente y se hace por la siguiente razón, la comunicación serial se está haciendo para datos no signados de 8 bits, es decir, hasta 255 en decimal, por el funcionamiento de la tarjeta MD-25 (driver para los motores) es suficiente ya que el 0 es la velocidad máxima en reversa, el 128 detiene a los motores y el 255 es la velocidad máxima hacia el frente. Este inconveniente se abordó de dos maneras: una simple limitación con rangos superior e inferior, manteniendo el dato máximo o mínimo en caso de que saliera de rango y otra donde se mantiene constante la relación entre las velocidades puesto que para el robot usado en la prueba la dirección está dada por velocidad diferencial. En el segundo caso la velocidad de la llanta que esté fuera de rango se mantiene en el extremo y mediante una constante calculada con el cociente entre las velocidades reales se escala al valor necesario para que esa relación se mantenga constante. De esta forma se lleva al extremo a los motores manteniendo la velocidad diferencial entre ellos, es decir, la dirección del robot no se ve afectada, sólo la velocidad en la que llega a esa posición. La primera aproximación a la solución del problema (simples límites) está programada en bloques de Simulink, y la segunda (manteniendo constante la relación entre velocidades) se programó en una función S, considerando todas las posibilidades que podrían presentarse. Ambos códigos se presentan en el apéndice 3.

Aunque las funciones están programadas de manera distinta (bloques y código) el resultado final en ambas es similar, un bloque que requiere como entradas las velocidades de ambas llantas en radianes sobre segundo y de salida entrega los comandos para la tarjeta MD-25 en valores desde 0 a 255 (Figura 20).

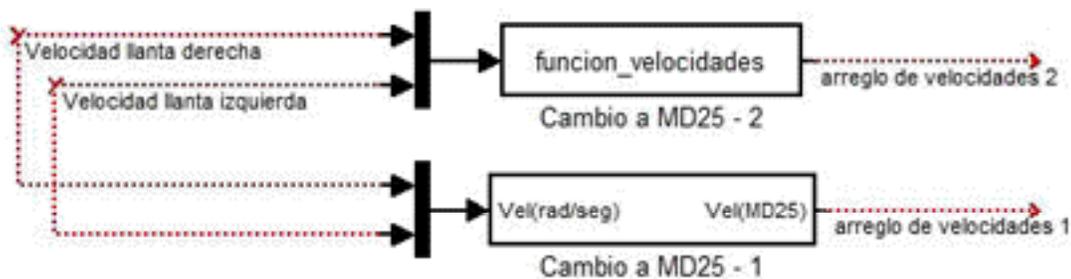


Figura 20: Funciones para acoplar las señales de control



Basándonos en pruebas de laboratorio se concluye que la función S que contiene el algoritmo para mantener constante la relación entre velocidades es la que mejor se comporta y es la que utilizamos. Cabe recalcar que para las velocidades en las que se necesitaría mantener constante la relación son muy altas, teóricamente para una prueba con los parámetros con los que trabajamos no debería ser necesaria esta parte, se programa como precaución. Su objetivo se completa con éxito como se muestra en la *Figura 21*.

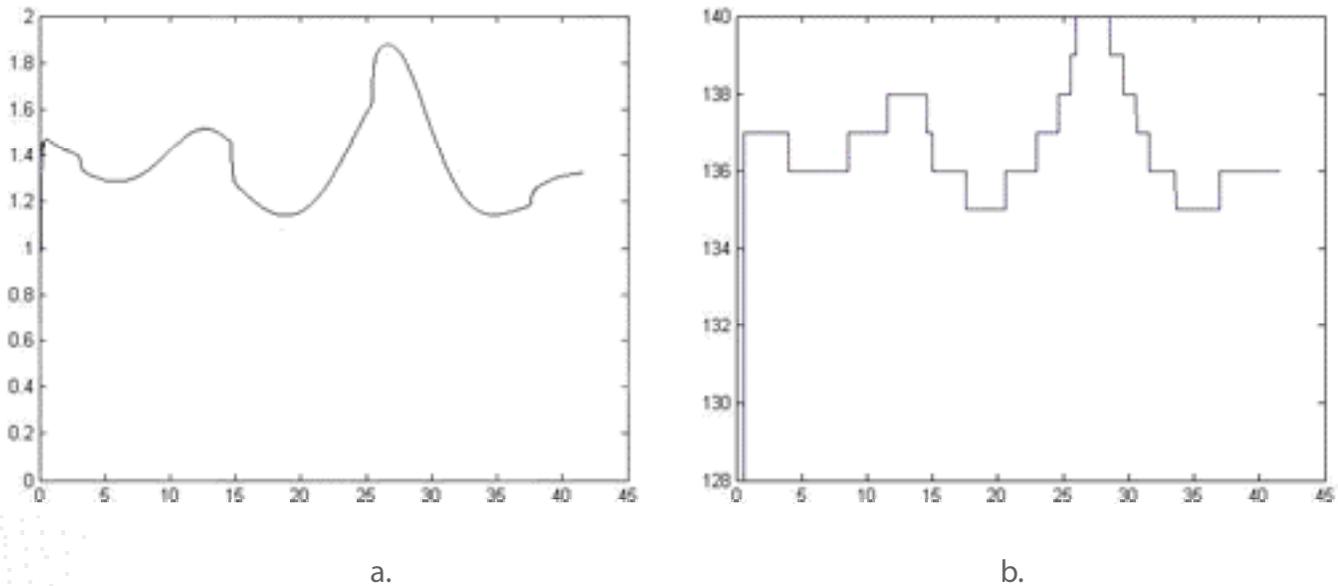


Figura 21: Comparativa de comandos de velocidad real y enviada. a. Comandos de velocidad reales, salida del controlador, antes de la función de acoplamiento. b. Comandos de velocidad enviados al robot, después de la función de acoplamiento.

Abordamos el problema de las velocidades obtenidas mediante derivadas de posición pues la función de visión está discretizada a una frecuencia bastante baja, por lo que necesita un proceso de suavizado de la curva para que los datos no generen por instantes comandos que no correspondan a la realidad.

2.5 Implementación de visión sobre la deformación de trayectorias

La información obtenida en la función de visión es crucial para el funcionamiento del experimento, se requiere con actualización rápida y con veracidad. Los datos se utilizan en 2 etapas del algoritmo, en la deformación de la trayectoria y en el controlador del robot móvil de la siguiente manera:

- No. De datos de obstáculos (4) y del robot (2) - Deformación de trayectoria
- No. De datos de robot (7) - Controlador



Aquí radica la diferencia entre los dos experimentos realizados, el lazo semicerrado y cerrado, en el uso de los datos del robot para la realimentación del controlador.

Los datos del robot que entran a la deformación de trayectorias no son los obtenidos directamente, pasan por una función que realiza una proyección para acoplarlos.

Para los datos de los obstáculos y del robot la función se programó de tal manera que se mantenga el dato actual hasta que llegue el siguiente. Un problema común es que para velocidades relativamente altas del robot móvil o de los obstáculos ReactIVision pierde el símbolo fiducial, por lo que la función está programada para mantener el último dato y refrescarlo hasta que llegue uno nuevo. Además, como la función está temporizada puesto que requiere bastantes recursos computacionales y nos afecta al momento de ejecutar el programa en tiempo real, el dato debe mantenerse todo el tiempo que la función no esté trabajando, para proveer de información y que el algoritmo trabaje correctamente. Un error de este tipo no provocaría una deformación de trayectorias o un seguimiento de trayectorias fallido sino errores que no permitan al programa continuar ya que no manda datos falsos, simplemente no manda datos y si una función trata de operar cadenas vacías es evidente que no lo logrará.

En el primero de los casos, el lazo semicerrado, se utilizan los datos de los obstáculos (4) para que se realice la deformación de trayectorias. Los datos del robot son solamente para registrarlos y generar las gráficas de error. Al controlador se realimentan los datos calculados por el modelo cinemático del robot. Es claro que no podemos llamarle lazo cerrado como tal, es cierto que se cierra el lazo, pero la información que se realimenta no es del sistema real, proviene de un modelo matemático que, como sabemos, no es más que una aproximación. Esto genera un comportamiento excelente para ciertas condiciones, como por ejemplo que no haya deslizamiento entre las llantas y la superficie, que no haya picos de velocidad que los motores no puedan alcanzar, entre otros factores o perturbaciones que pudieran afectar al sistema. Ahora bien, es claro que este experimento es suficiente para probar la teoría de deformación de trayectorias, puesto que la evasión de obstáculos se realiza en simulación y físicamente en tiempo real, bajo parámetros que faciliten su correcto funcionamiento como velocidad de la trayectoria a seguir o los radios de influencia tanto para el robot como para el obstáculo.

Para lograr el lazo cerrado completo se realimentan los valores de posición y velocidad del robot para ser procesados por el algoritmo y con base en ellos generar los comandos de control. Como se mencionó anteriormente tenemos varias maneras de obtener estos datos, en particular los de velocidad. La elección de qué datos se utilizan para la realimentación se hace mediante experimentos en el laboratorio, probando cada uno de los medios de obtención



de los valores y llegando a la conclusión de que los que se calculan mediante la derivada de la posición (lineal y angular) son los mejores para la prueba.

2.6 Funcionamiento en tiempo real

El funcionamiento en tiempo real se refiere a hacer que todo el sistema funcione en concordancia con la realidad. Esto es de gran importancia pues se requiere coordinar al sistema en la realidad (robot y obstáculos en el campo de trabajo) con el algoritmo en la computadora.

Como bien sabemos, generalmente las simulaciones de computadora son más rápidas que la realidad, lo cual en este caso nos favorece, puesto que lo que necesitamos es retardarla para coordinarse con la realidad, en caso contrario esta coordinación resultaría imposible a menos que tuviéramos más recursos computacionales para que el tiempo de cómputo se reduzca y de esta manera estar en el caso anterior.

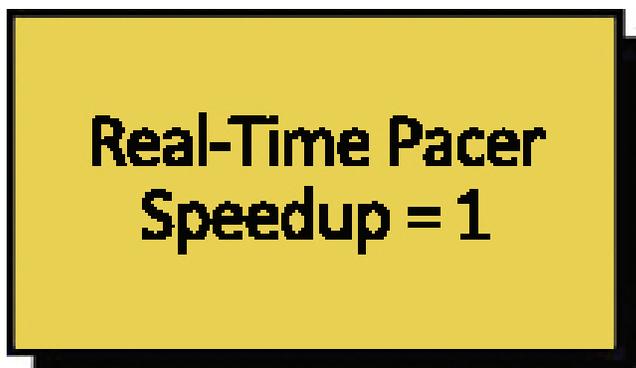


Figura 22: Bloque para trabajar en tiempo real (Real Time Pacer)

Para sincronizar se utilizó un bloque llamado Real Time Pacer, o Marcador de pasos en tiempo real mostrado en la *Figura 22*. Su funcionamiento consiste en retrasar la simulación para hacer coincidir el tiempo de simulación con el tiempo real o para mantener una relación constante entre estos dos tiempos. Como podemos observar en la *Figura 23* esta relación puede ser distinta de 1, sin embargo por cuestiones de la prueba para este caso en particular se utiliza la unidad. Esto significa que un segundo en simulación computacional consiste en un segundo en tiempo real, que es precisamente lo que estamos buscando.



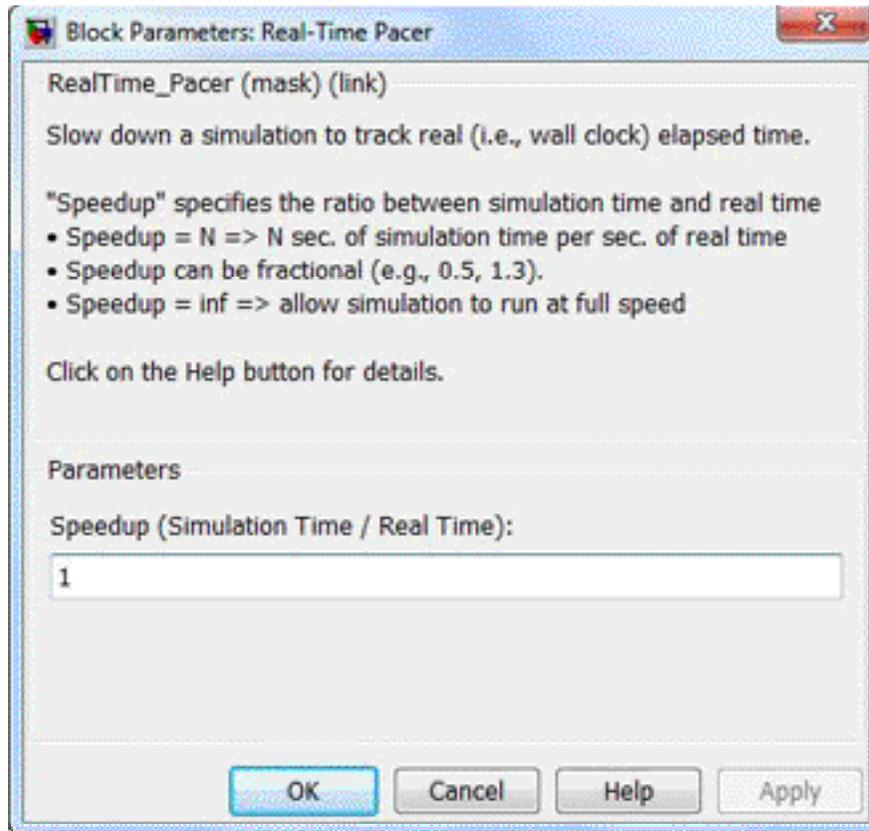


Figura 23: Propiedades del bloque Real Time Pacer

Nuestra simulación, sin el uso de Real Time Pacer, está casi sobre el tiempo real. Esto generalmente no sucede y es consecuencia de la naturaleza del sistema diseñado para la prueba. Desde el enlace entre software para la función de visión, la relativa complejidad de los algoritmos de control y de deformación de trayectorias, hasta el tipo de comunicación y la dificultad que aparentemente tiene Simulink para realizarla, todas son causantes de consumo de recursos computacionales y, por tanto, de aumento en el tiempo de procesamiento. Aquí es donde radica la importancia de las funciones S de Simulink. La capacidad para temporizarse nos genera grados de libertad con los que podemos trabajar para reducir el tiempo lo necesario para mantenernos en rango de tiempo real. Sin embargo lo crucial es saber cuáles son las funciones que podemos temporizar y cuáles debemos dejar que corran cada iteración del programa. Una mala elección en este sentido podría generar errores más grandes y resultados no satisfactorios, además, una vez conocidas las funciones con las que podemos trabajar, en lo que a temporización se refiere, la tarea se traduce a probar, mediante experimentos en el laboratorio, frecuencias para las funciones y comparar resultados para saber cuál es el mejor.



Para este caso las funciones que se decidió temporizar fueron la de visión y el bloque de envío de datos por puerto serial. Este último en realidad debe ser temporizado por defecto como parte de los requerimientos del mismo.

Para poder utilizar Real Time Pacer se deben de seguir ciertos pasos, similares a la utilización del cliente TUIO Java en el caso de la parte de visión:

1. Descargar Real Time Pacer (disponible en la red)
2. Agregar al Path de Matlab el directorio descargado mediante el comando "ADDPATH dir"
3. Si se realizó correctamente, en Simulink habrá un menú llamado Real Time Pacer con el bloque presentado en la *Figura 22*

Cabe recalcar que estas explicaciones de pasos a seguir para poder utilizar los recursos usados en este proyecto son con la finalidad de que no sólo sea repetible, sino que la estructura básica del sistema se pueda utilizar para probar otras teorías mediante un sistema de visión artificial con ReactIVision y comunicación serial desde Simulink.

2.7 Comunicación con el robot

La comunicación inalámbrica entre la computadora y el robot es crucial para esta prueba. El sistema debe ser capaz de enviar los datos necesarios para mover al robot en la dirección correcta y lo suficientemente rápido para que el robot responda como debe de hacerlo. Para la aplicación se utilizó Arduino BT, *Figura 24*, provisto de un microprocesador ATMEGA328 de montaje superficial y un módulo Bluetooth Bluegiga WT11 integrado a la tarjeta (entre otras características véase apéndice 5).



Figura 24: Arduino BT



La tecnología Bluetooth es tecnología de comunicación de rango corto, simple, segura y en cualquier lugar. Las características de la tecnología Bluetooth son la robustez, el bajo consumo de energía y su bajo costo. El rango de alcance de un dispositivo Bluetooth varía entre los 3 y 10 metros, dependiendo del sistema integrado a él.

La comunicación inalámbrica vía Bluetooth opera en la banda de aplicaciones industriales, científicas y médicas sin licencia (ISM por sus siglas en inglés) de 2.4 a 2.485 GHz utilizando un espectro disperso, brincos de frecuencia y señales dobles (datos y voz) a una velocidad nominal de 1600 brincos/segundo (hop/sec).

En lo que a interferencia se refiere se utiliza una tecnología adaptativa de brincos de frecuencia (AFH por sus siglas en inglés) precisamente para evitar interferencia entre dispositivos o con ruido. Se logra detectando otros dispositivos en el mismo espectro y evitando las frecuencias que los demás utilizan para la comunicación. El sistema de saltos adaptativos de frecuencia, entre 79 frecuencias con intervalos de 1 MHz provee al sistema un alto grado de inmunidad contra la interferencia y permite mayor eficiencia en la transmisión dentro del espectro.

Existe cierta clasificación para los dispositivos dependiendo del alcance:

- Clase 3 – alcance de hasta 1 metro
- Clase 2 – frecuentemente utilizados, alcance de hasta 10 metros
- Clase 1 – utilizados para aplicaciones industriales, alcance de hasta 100 metros

Dependiendo de la clase varios parámetros del dispositivo cambian. Uno de los importantes para mencionar es el consumo de energía. Para los clase 2, que son los que más se utilizan y se encuentran en el mercado, el consumo es de 2.5 mW [15].

Para este experimento las características del dispositivo a utilizar son favorables, por el tamaño, el consumo energético, la velocidad de transmisión, entre otras variables. El dispositivo Bluetooth integrado a la tarjeta (*Figura 25*), WT11 de Bluegiga, cumple con las características y nos facilita la tarea de comunicar la computadora con el robot.

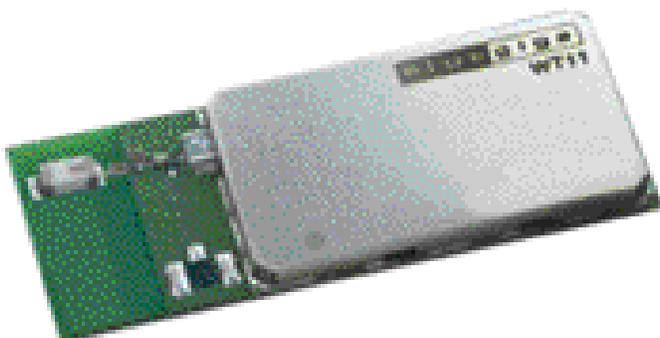


Figura 25: Módulo WT11 Bluegiga



Cabe recalcar que la comunicación no se realiza directamente mediante Bluetooth, es un proceso facilitado por Arduino con el cual la comunicación se programa de manera serial (RS232), y mediante un convertidor, instalado automáticamente cuando se enlaza con Arduino BT. Este convertidor es similar al instalado para cualquier Arduino, USB Serial Converter, sólo que ahora convierte a comunicación vía Bluetooth. De igual manera la placa ArduinoBT convierte los datos entrantes y salientes para poder procesarlos como datos por puerto serial, sin necesidad de modificar la programación para otros protocolos. Este proceso se esquematiza en la *Figura 26*.

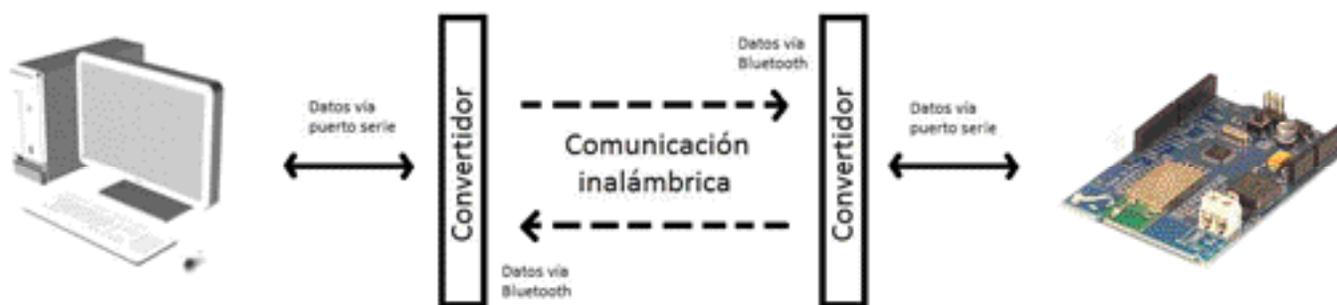


Figura 26: Esquemático de comunicación Computadora-Robot

Esta comunicación, por el dispositivo de la computadora y el del microcontrolador Arduino BT, soporta 8.5 metro, lo cual es suficiente para hacer las pruebas de laboratorio.

Como se explicó, con los recursos computacionales disponibles logramos que el problema se redujera a comandos de comunicación serial RS232. Por las características del convertidor integrado en Arduino BT ésta sólo se puede realizar a una velocidad de 115200 baudios.

2.8 Comunicación con los motores

El flujo de datos tiene como etapa final a los actuadores. Estos motores son los que mueven al robot controlados por los comandos provenientes del algoritmo. Estos comandos son valores de velocidad angular para cada una de las llantas.



El robot está provisto con motores EMG-30 de corriente directa y la tarjeta driver para 2 de ellos (*Figura 27*).

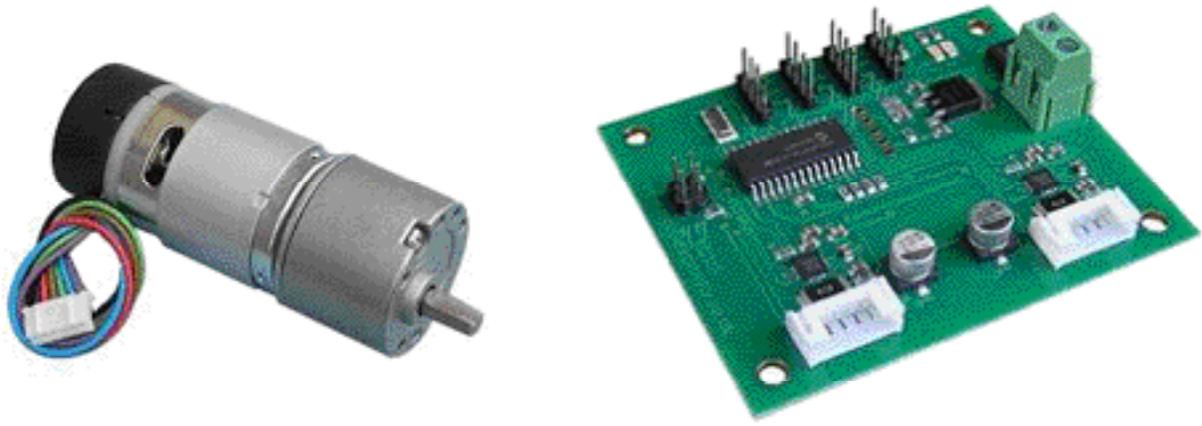


Figura 27: Motor EMG30 y tarjeta MD25

Cada motor es de 12 VCD totalmente equipado con codificadores de cuadratura, caja reductora de 30:1 y capacitor para suprimir el ruido.

La tarjeta de drivers para los motores que se presenta es la MD25. Con ella se pueden controlar dos motores EMG30 mediante comandos independientes, que provienen del algoritmo en Simulink. Requiere alimentación de 12 [Volt] y cuenta con etapa de regulación para alimentar el circuito interno con 5 [Volt]. Este voltaje se utiliza también para alimentar el microcontrolador ArduinoBT. Contiene un microcontrolador PIC-877A y las etapas de potencia puente H para los motores. Pueden controlarse por medio de comunicación serial o con I2C. Por facilidad se eligió controlarlos mediante I2C puesto que el puerto serie que tiene ArduinoBT está utilizado para comunicarse con la computadora y, aunque se podría agregar otro puerto serie mediante programación, implicaría complicar el código de manera innecesaria.

Tenemos registros dentro de la tarjeta que nos dan información adicional, los encoders, el consumo de energía, el nivel de la batería y aceleración (véase apéndice 6) y un controlador interno para la velocidad de los motores.

Todas estas funciones y dispositivos conjuntan el sistema utilizado para las pruebas. Fue un proceso de selección basado en experimentación y trabajos anteriores para lograr el sistema final.



IV. Experimento y resultados

1. Entorno de las pruebas

Las pruebas realizadas son a nivel laboratorio. El sistema descrito en los capítulos anteriores ahora se detalla. Se utilizó una cámara Sony HandyCam colocada en el techo del laboratorio, a 2.3 [metro] del piso aproximadamente. Se encuentra conectada por puerto FireWire 1394 a la computadora. La cámara, a esta altura, nos genera un espacio de trabajo de aproximadamente 1.9 [metro] en el eje X y 1.4 [metro] para el eje Y. Con estas constantes de proporción se hace el escalamiento de los datos de visión. Los símbolos fiducial son, como ya se mencionó, los número 1, 3 y 4. Son cuadrados de 14 [cm] por lado, con su centroide coincidiendo con el punto P del robot (véase II.2.4.1). Este sistema se muestra en la *Figura 28*.

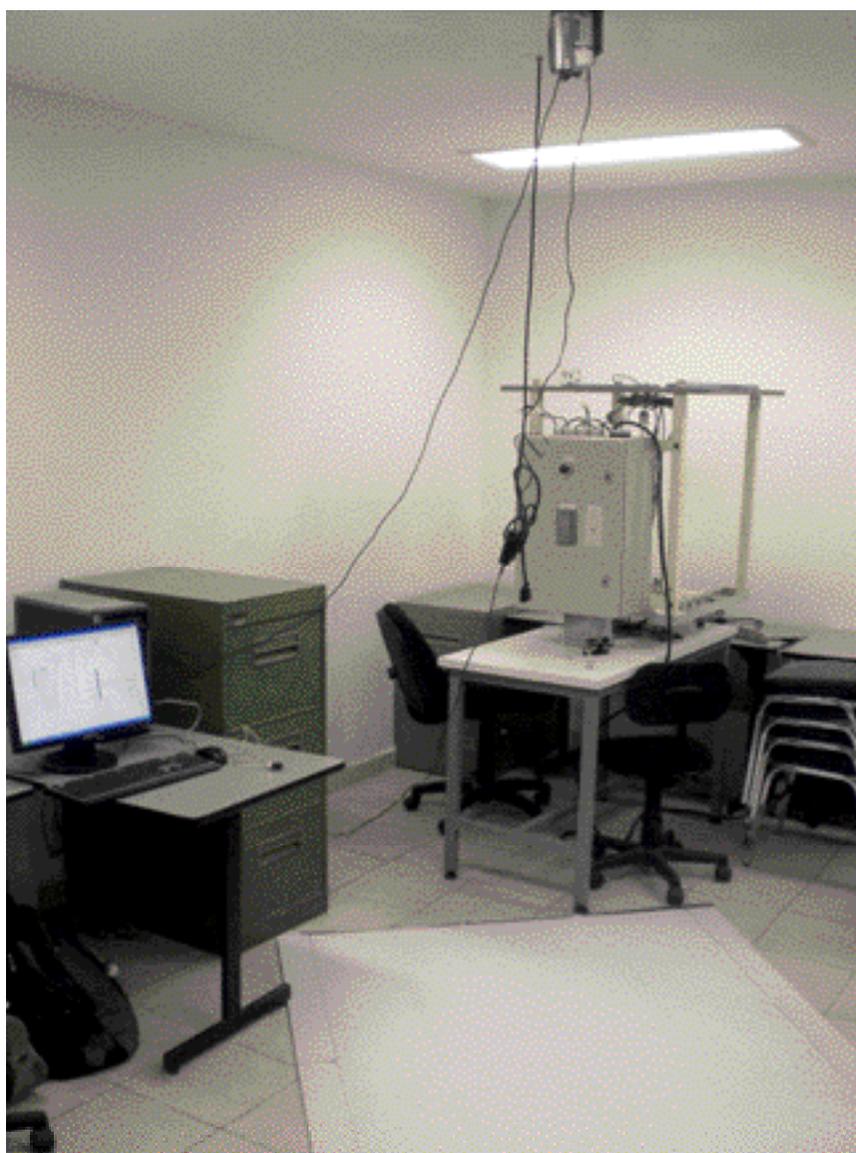


Figura 28. Sistema real



Diferenciamos dos tipos de pruebas:

Pruebas en lazo semicerrado

Pruebas en lazo cerrado

La diferencia está en la realimentación de los datos del sistema real (visión) al algoritmo. Se hace esta distinción puesto que ambas teorías funcionan sin embargo encontramos problemas al unificarlas, lo cual se tratará más adelante. Para la mayoría de las pruebas se utiliza una trayectoria predefinida, la que se deformará:

$$xd = xd_0 + vel * t;$$

$$yd = yd_0 + ampl * \sin(2 * \pi * frec * t);$$

La cual es una senoidal parametrizada con el tiempo, parámetro (t) que ofrece Simulink obtenido desde los datos que utiliza para las operaciones internas del programa. En esta función paramétrica, en el caso del lazo semicerrado, la amplitud (ampl) es cero, de esta manera se genera una recta en el eje X a velocidad constante (vel). Para el caso cerrado la amplitud (ampl) es distinta de cero, al igual que la frecuencia, para nuestra prueba:

$$vel = \frac{1}{20} \left[\frac{m}{seg} \right];$$

$$ampl = 0.2[m];$$

$$frec = \frac{1}{150} [Hz];$$

Estos son los parámetros utilizados para las pruebas de laboratorio realizadas. La obtención de las mismas fue mediante mediciones, cálculos y experimentos sobre el sistema para utilizar las óptimas. La única prueba donde la trayectoria es distinta es la Prueba 1 con los 2 obstáculos móviles. En esta prueba se toma de igual forma una recta pero con un perfil de velocidades diferente, tomado de [5]. El objetivo de este perfil en particular es evitar demandas altas de torque (como usamos motores eléctricos la demanda de torque es directamente proporcional a la demanda de corriente eléctrica) al momento del arranque.

Los radios de influencia se manejaron, tanto para el robot como para los obstáculos, de 15 [cm] para todas las pruebas excepto la Prueba 1 y Prueba 2 de ambos obstáculos móviles, la prueba 2 de 1 obstáculo fijo y uno móvil y las de lazo cerrado, donde se definieron de 30 [cm] para la Prueba 1 de ambos obstáculos móviles, 20 [cm] para la prueba 2 y de 0.1 [cm] para las de lazo cerrado. En la *Figura 13* (Página 34) podemos observar el campo virtual de trabajo para la prueba



de lazo abierto. Tenemos la pantalla de ReactIVision obteniendo datos en tiempo real del campo, la deformación reactiva de trayectorias en la simulación de Matlab obtenida mediante el modelo cinemático del robot y el algoritmo en Simulink en la parte del fondo. La *Figura 29* presenta la captura para la prueba de lazo cerrado. En este caso se configura la simulación para no presentar los datos anteriores animados, solo el dato presente, para facilitar el análisis del funcionamiento al momento de la prueba y, como los datos de posición y velocidad del robot ya están realimentados se obvia la parte de presentarlos en una gráfica al momento de la prueba.

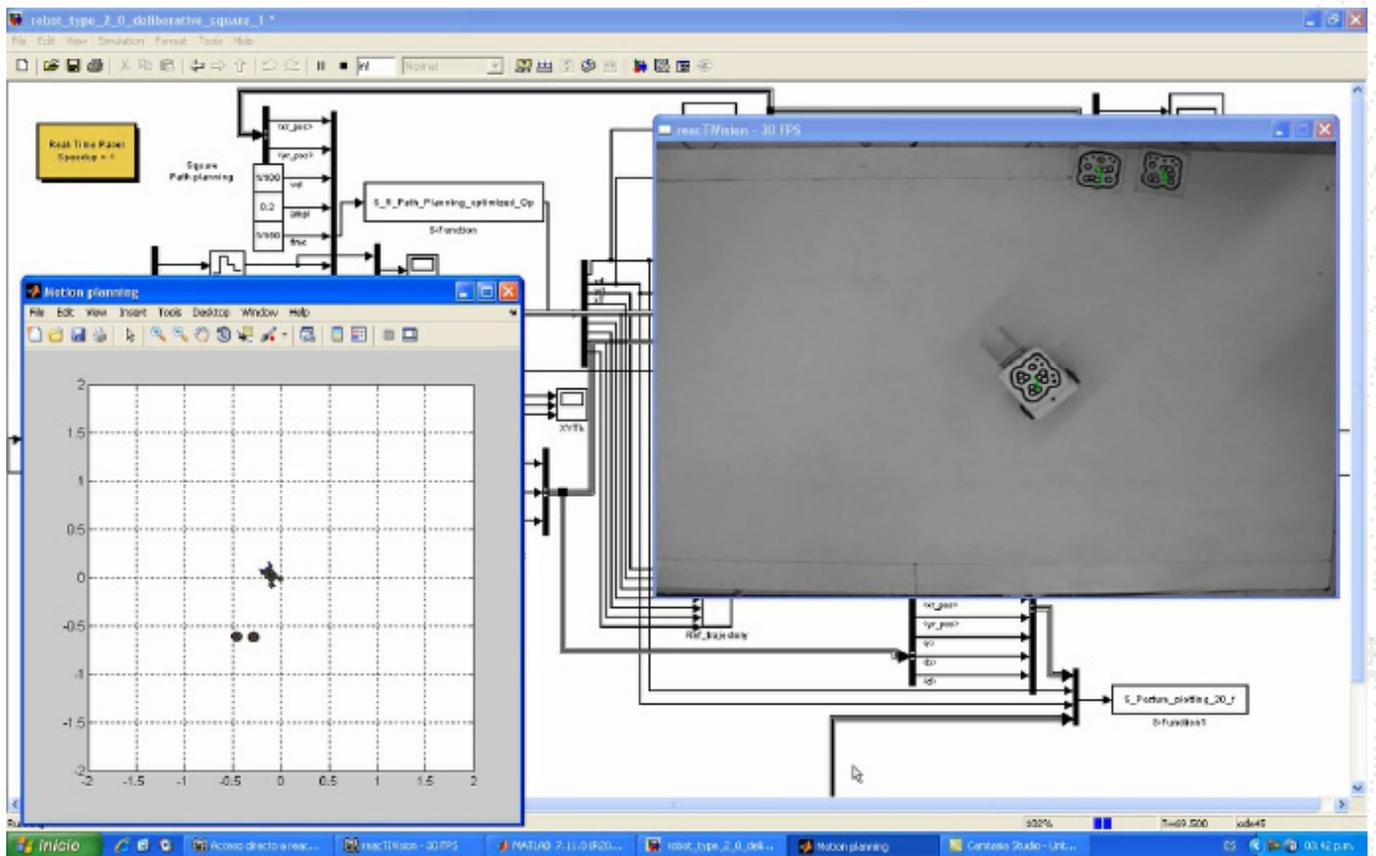


Figura 29: Captura de pantalla para la prueba de lazo cerrado.

Para todas las pruebas se presenta el mismo orden de figuras:

- a. Trayectoria final
- b. Posición del obstáculo 1
- c. Posición del obstáculo 2
- d. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo
- e. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo



f. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot

g. Variables de postura del robot (x,y,θ)

h. Error de seguimiento del punto de referencia del robot

i. Velocidad de las variables de postura del robot $(\dot{x},\dot{y},\dot{\theta})$

j. Velocidades de las llantas $(\dot{\phi}_r,\dot{\phi}_l)$

k. Velocidades del sistema $(\dot{x}_1,\dot{\theta})$ lineal y angular

l. Aceleraciones del sistema $(\ddot{x}_1,\ddot{\theta})$ lineal y angular

2. Pruebas en lazo semicerrado

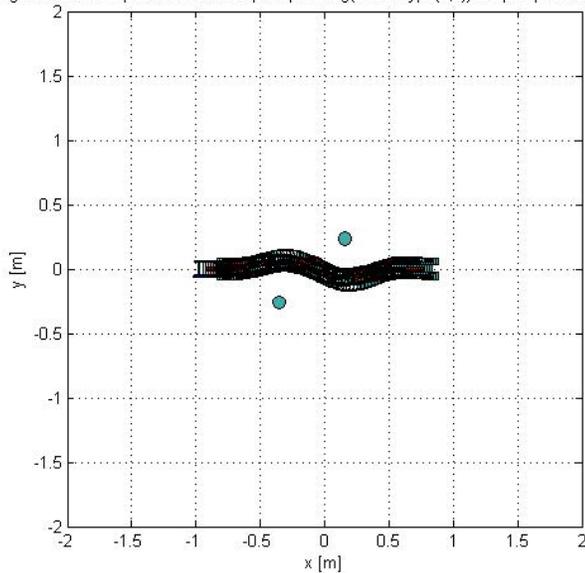
Éstas se realizaron sin realimentar datos de visión al controlador sino datos de cálculos mediante el modelo cinemático del robot.

2.1.2 Obstáculos fijos

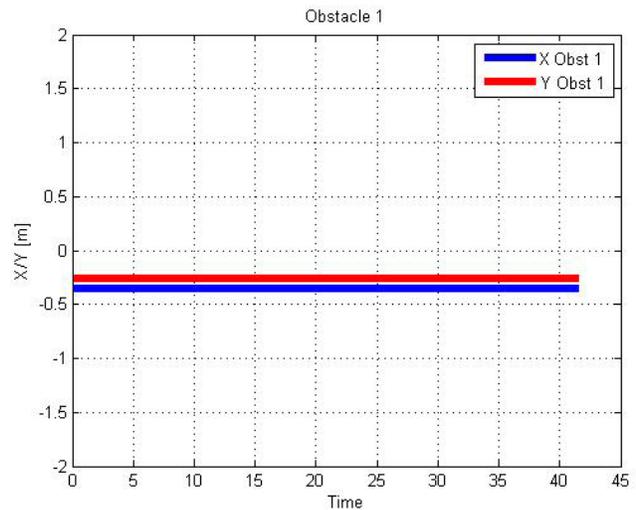
Se presentan dos experimentos con obstáculos fijos en diferentes configuraciones:

Prueba 1:

Figure A.XXV.1: Open area reactive path planning (Robot type(2,0)): Seq. of post. snapshot:

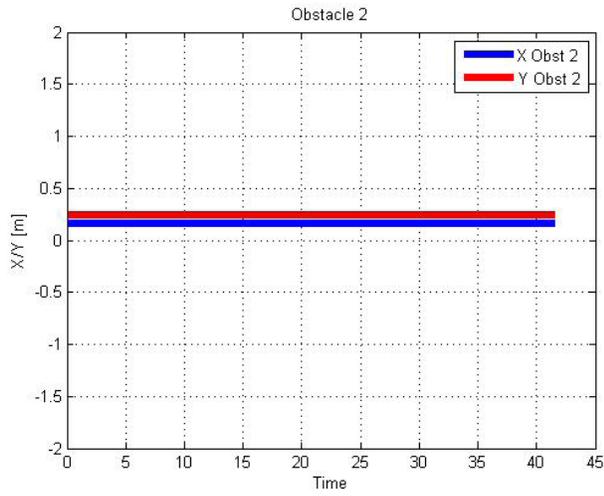


a

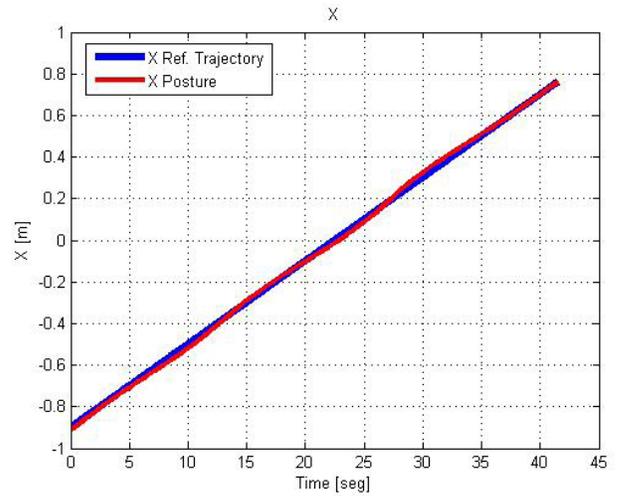


b

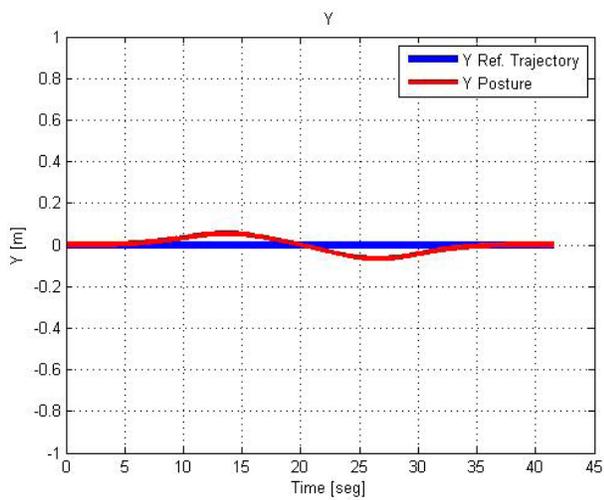




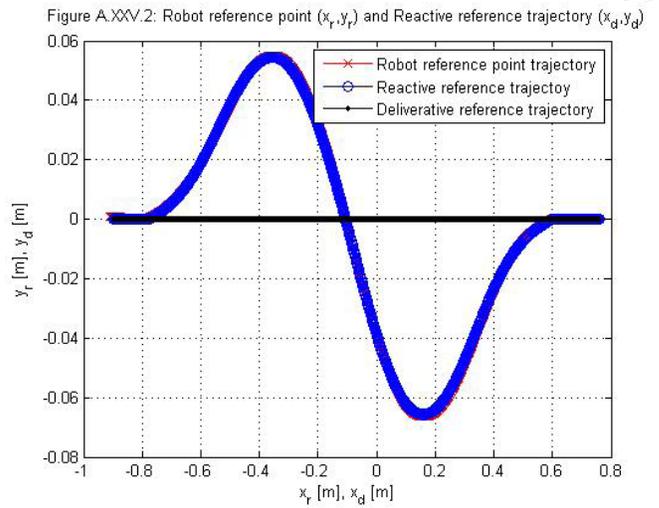
c



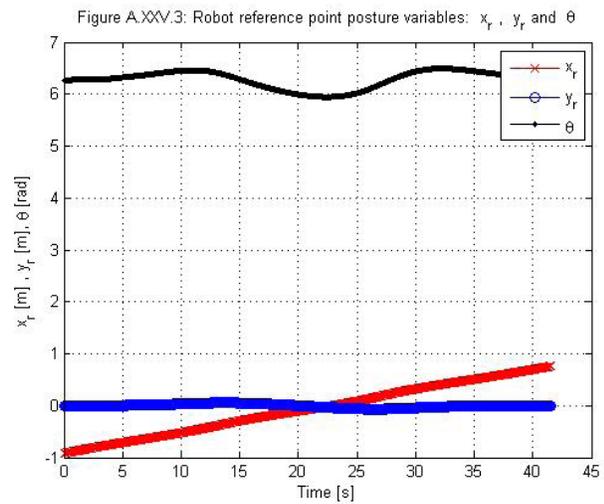
d



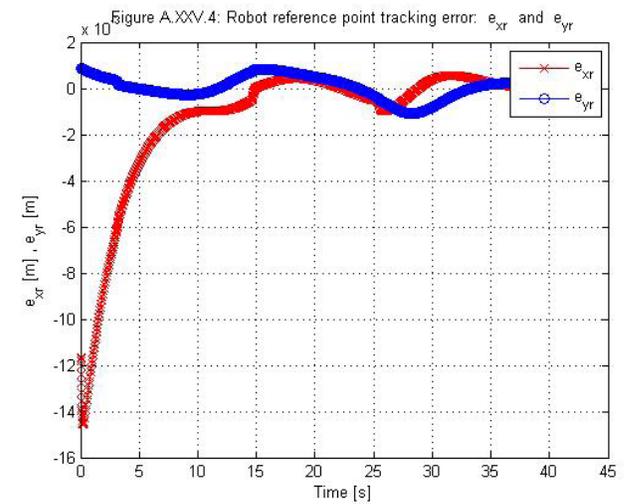
e



f



g



h



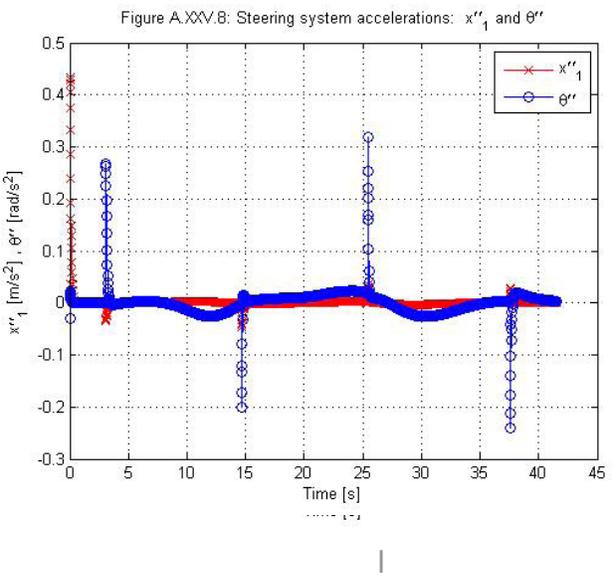
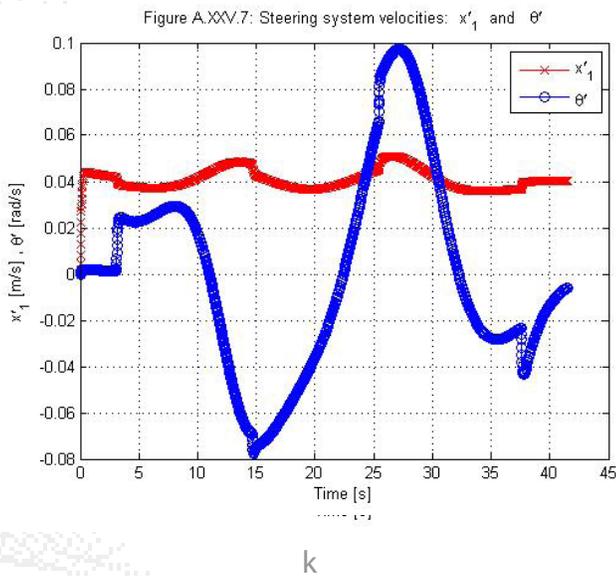
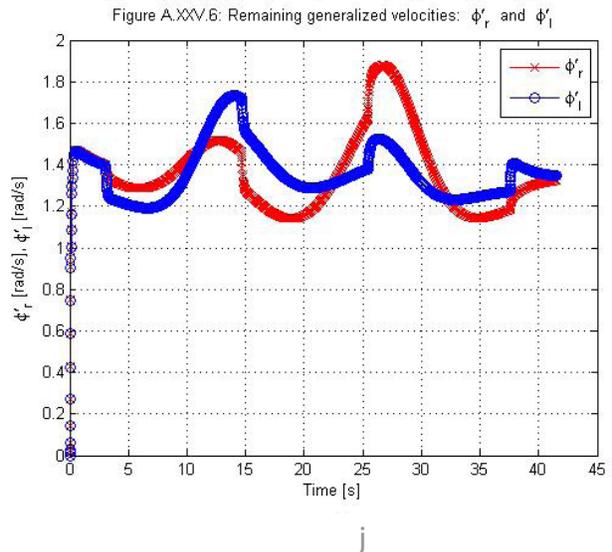
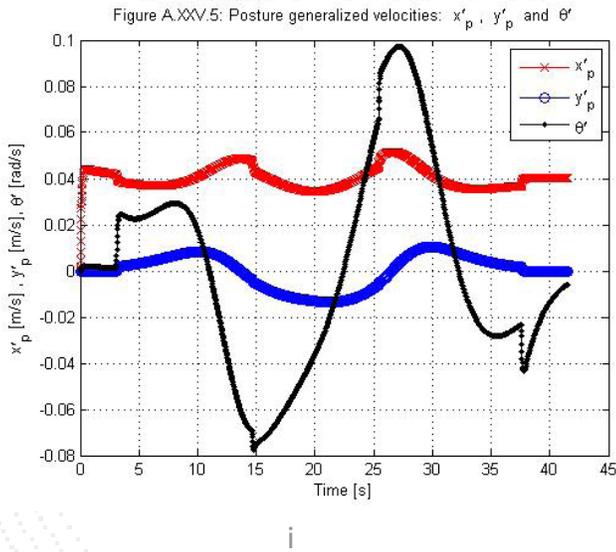
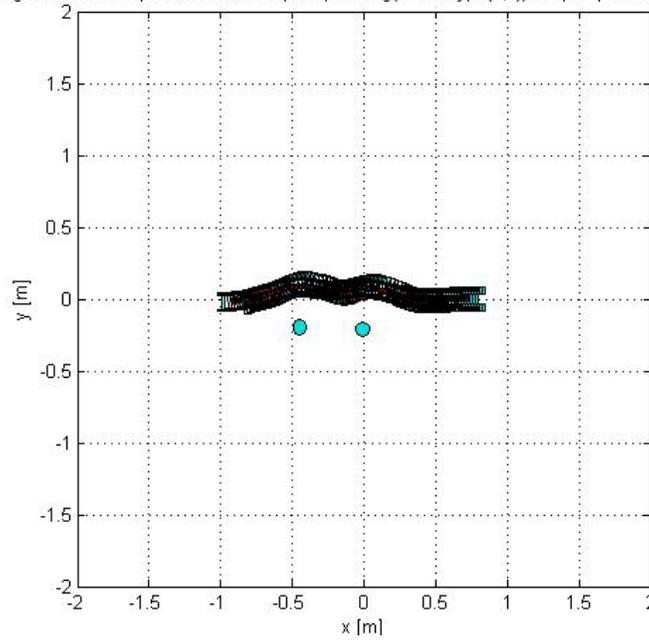


Figura 30: Resultados prueba 1: lazo semicerrado 2 obstáculos fijos 1. a. Trayectoria final recorrida. b. Posición del obstáculo 1. c. Posición del obstáculo 2. d. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. e. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. f. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. g. Variables de postura del robot (x, y, θ). h. Error de seguimiento del punto de referencia del robot. i. Velocidad de las variables de postura del robot (x', y', θ'). j. Velocidades de las llantas (ϕ'_r, ϕ'_l). k. Velocidades del sistema (x'_1, θ') lineal y angular. l. Aceleraciones del sistema (x''_1, θ'') lineal y angular.

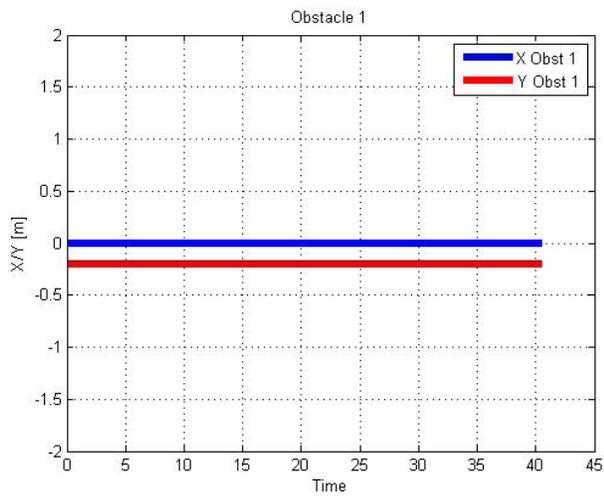


Prueba 2:

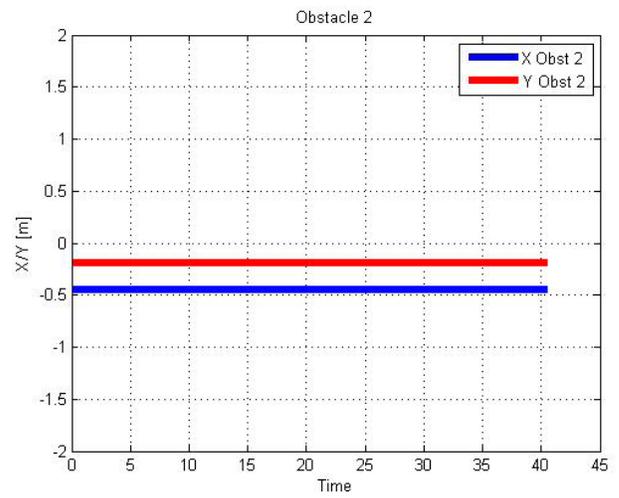
Figure A.XXV.1: Open area reactive path planning(Robot type(2,D)):Seq. of post. snapshot:



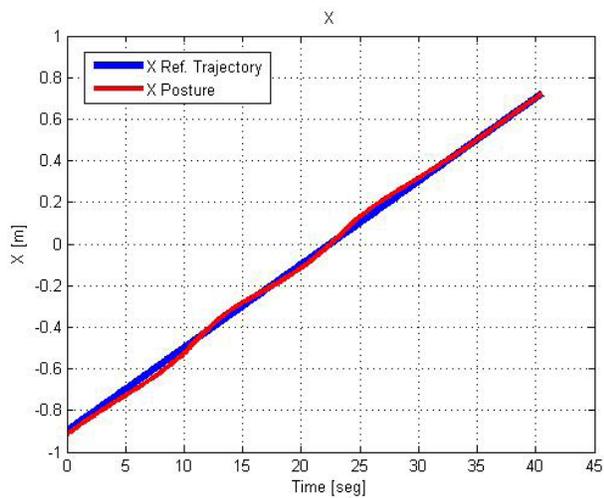
a



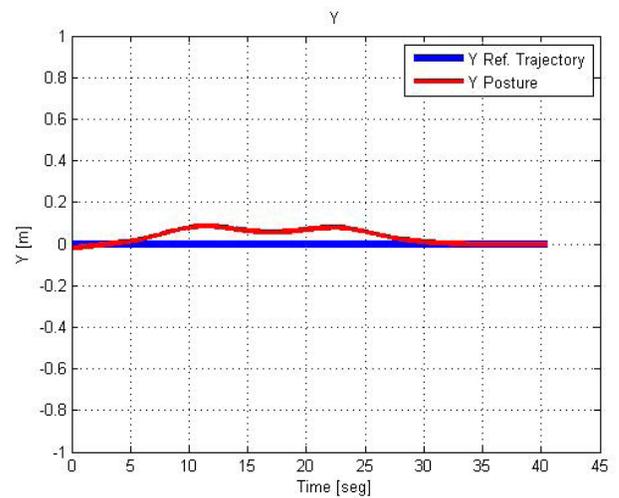
b



c

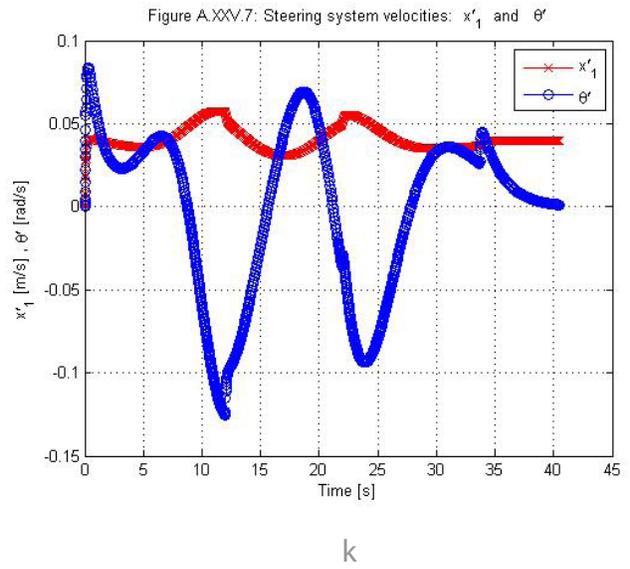
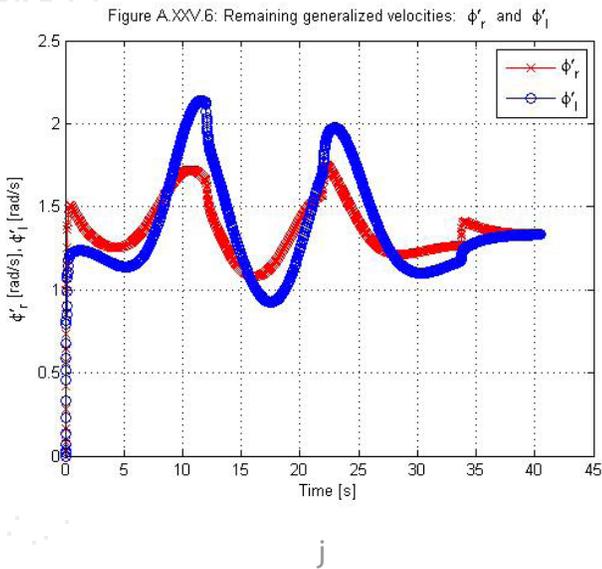
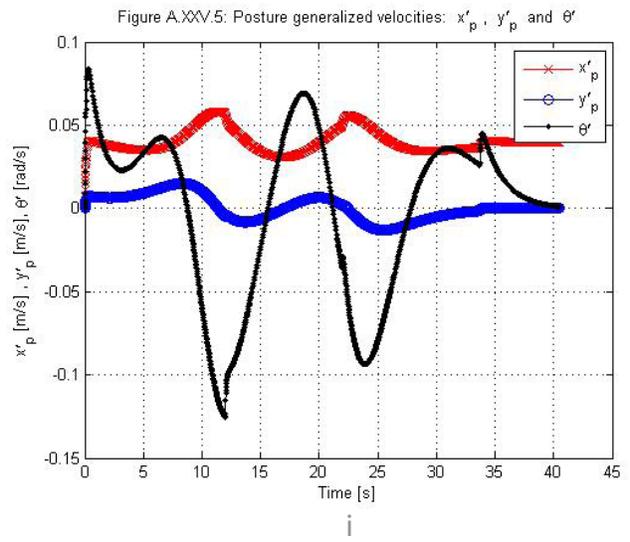
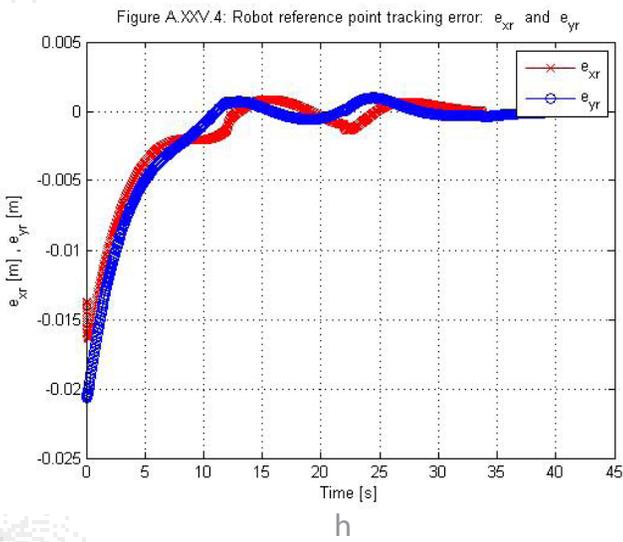
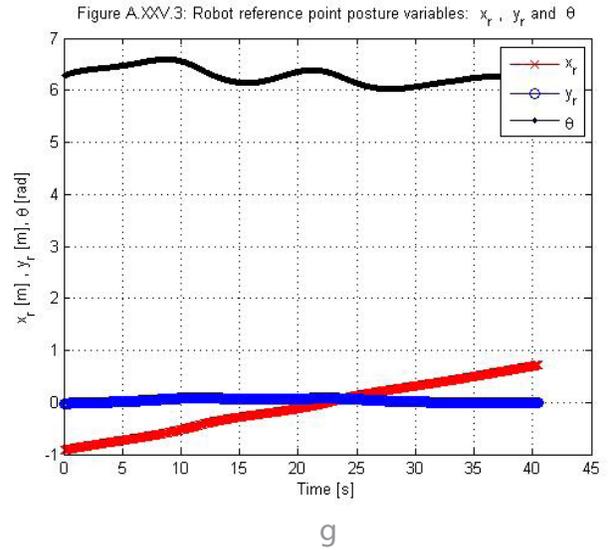
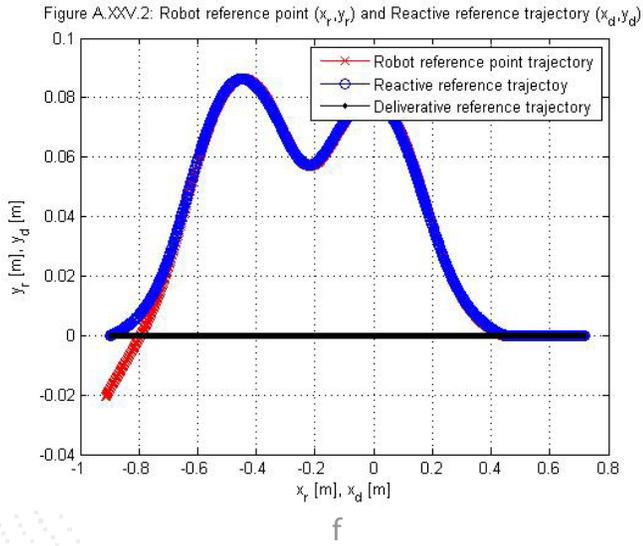


d



e





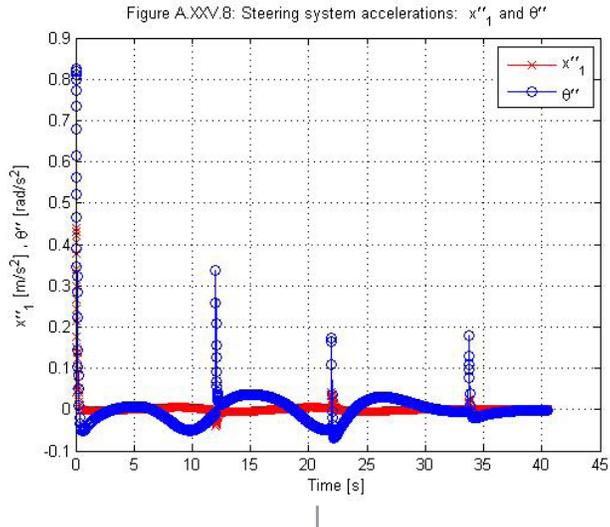
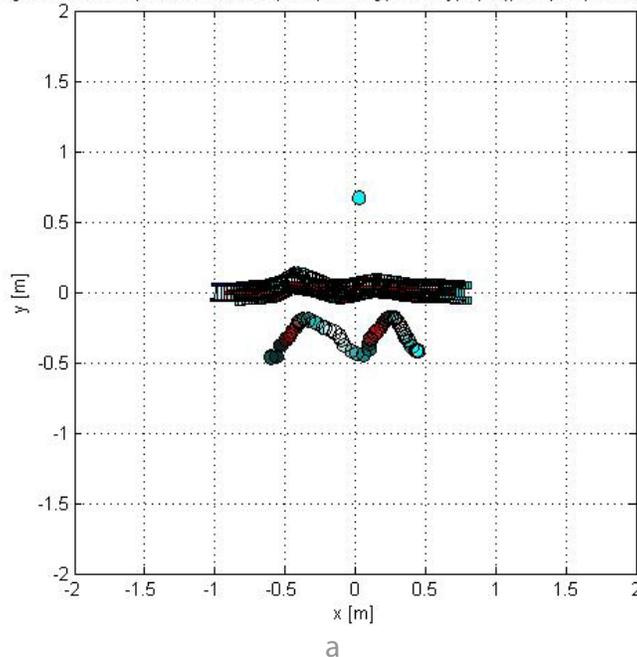


Figura 31: Resultados prueba 2: lazo semicerrado 2 obstáculos fijos 2. a. Trayectoria final recorrida. b. Posición del obstáculo 1. c. Posición del obstáculo 2. d. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. e. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. f. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. g. Variables de postura del robot (x,y,θ) . h. Error de seguimiento del punto de referencia del robot. i. Velocidad de las variables de postura del robot $(\dot{x},\dot{y},\dot{\theta})$. j. Velocidades de las llantas $(\dot{\phi}_r,\dot{\phi}_l)$. k. Velocidades del sistema $(\dot{x}_1,\dot{\theta})$ lineal y angular. l. Aceleraciones del sistema $(\ddot{x}_1,\ddot{\theta})$ lineal y angular.

2.2. 1 Obstáculo fijo y 1 obstáculo móvil

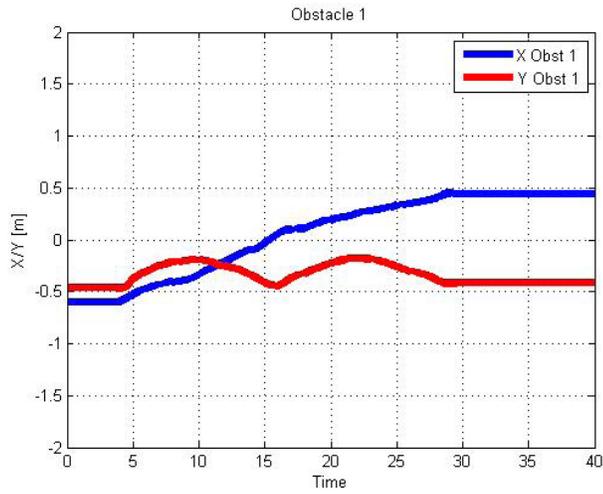
Prueba 3:

Figure A.XXV.1: Open area reactive path planning(Robot type(2,0)):Seq. of post. snapshot:

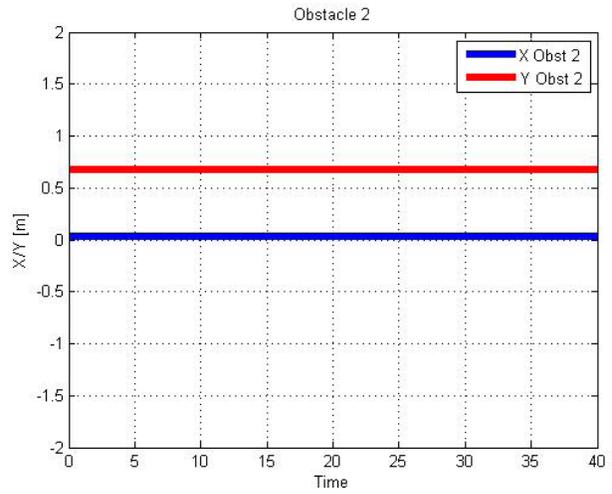


a

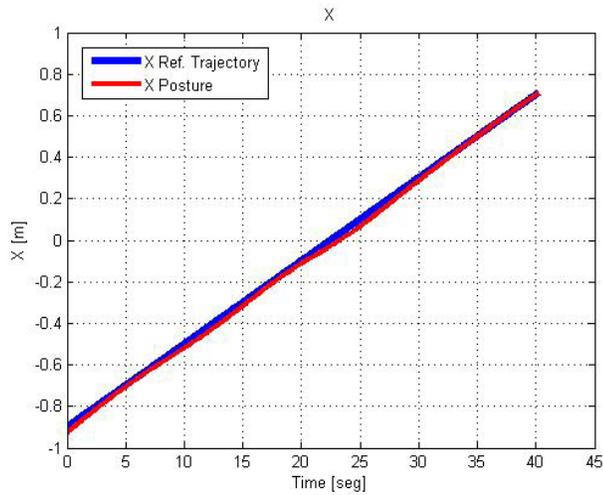




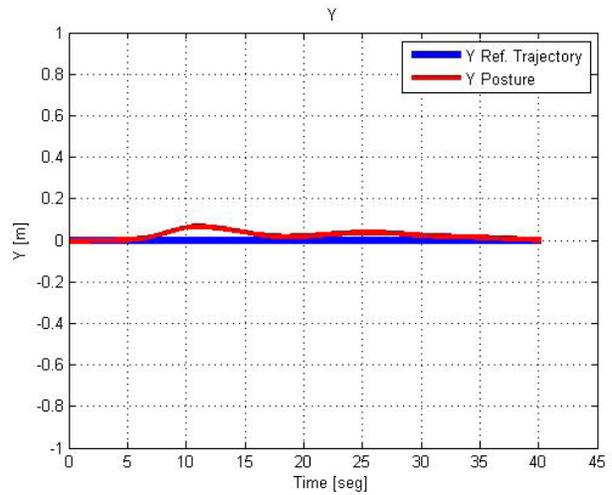
b



c

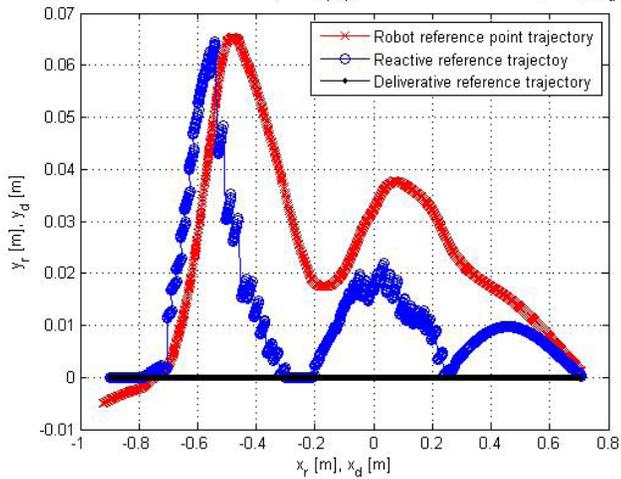


d



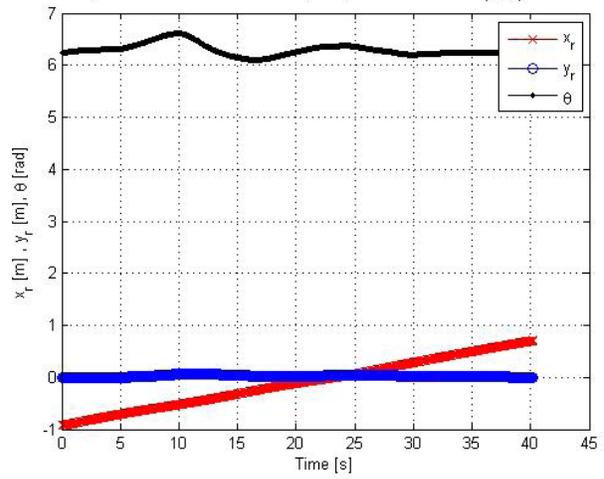
e

Figure A.XV.2: Robot reference point (x_r, y_r) and Reactive reference trajectory (x_d, y_d)



f

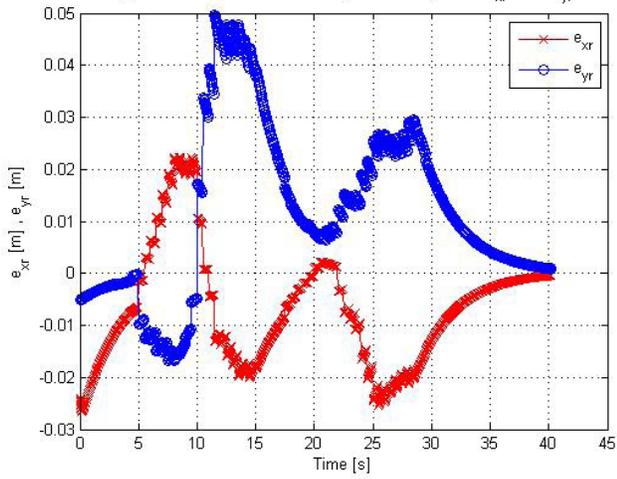
Figure A.XV.3: Robot reference point posture variables: x_r , y_r and θ



g

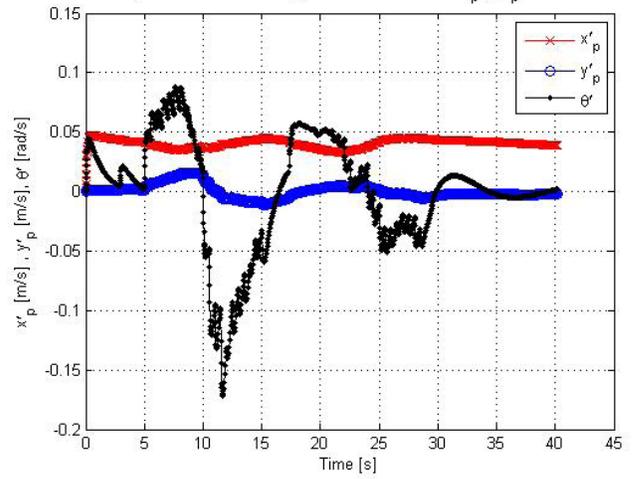


Figure A.XXV.4: Robot reference point tracking error: e_{xr} and e_{yr}



h

Figure A.XXV.5: Posture generalized velocities: x'_p , y'_p and θ'



i

Figure A.XXV.6: Remaining generalized velocities: ϕ'_r and ϕ'_l

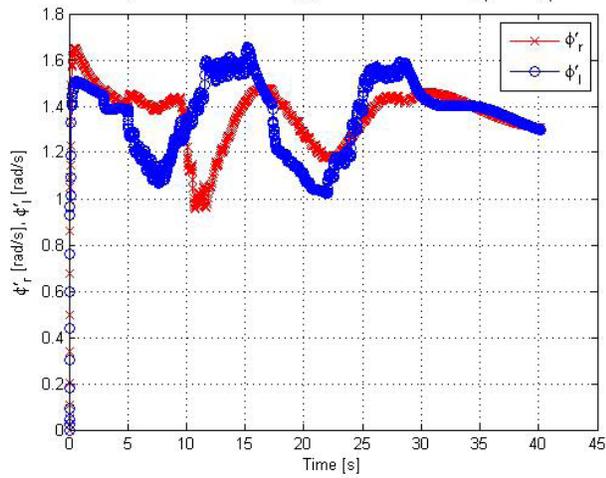


Figure A.XXV.7: Steering system velocities: x'_1 and θ'

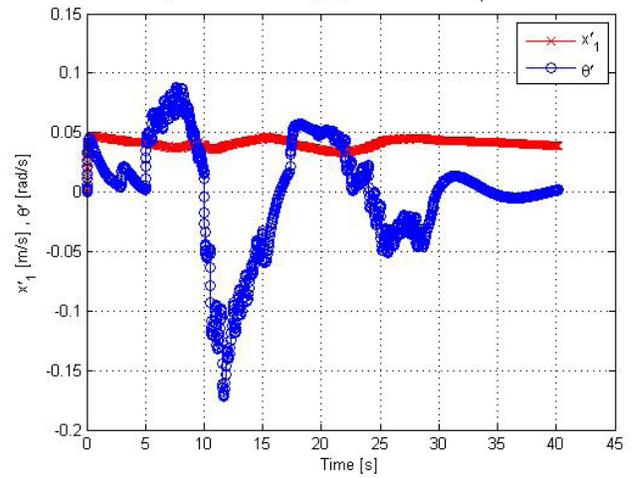


Figure A.XXV.8: Steering system accelerations: x''_1 and θ''

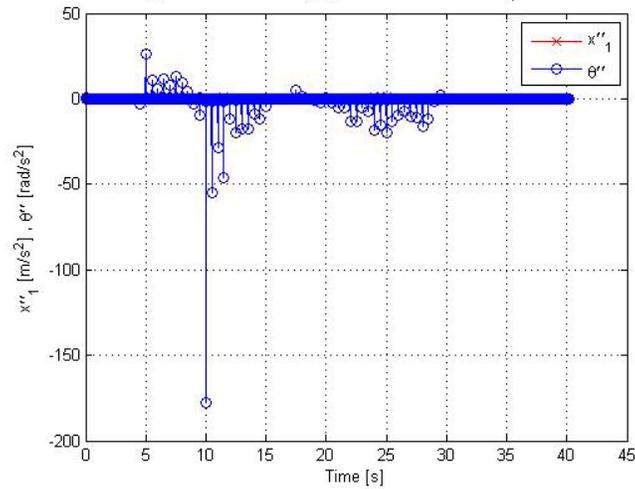
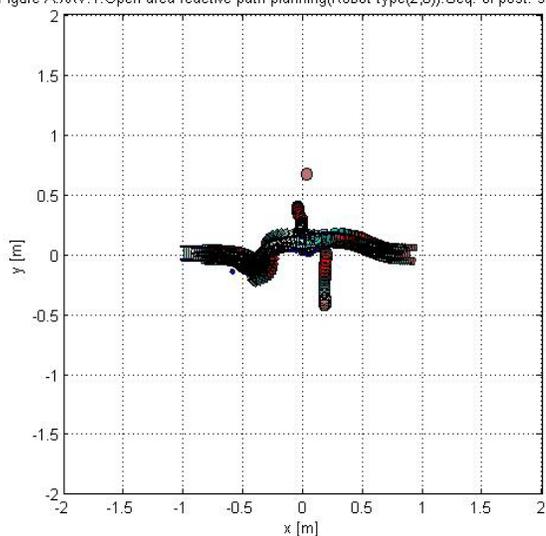


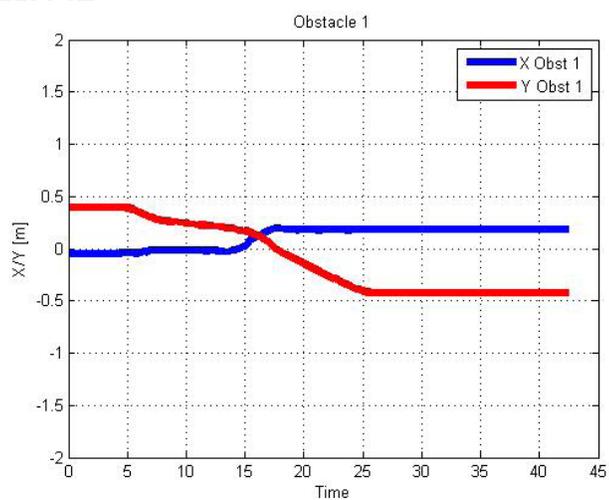
Figura 32: Resultados prueba 3: lazo semicerrado 1 obstáculo fijo y 1 móvil 1. a. Trayectoria final recorrida. b. Posición del obstáculo 1. c. Posición del obstáculo 2. d. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. e. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. f. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. g. Variables de postura del robot (x,y,θ) . h. Error de seguimiento del punto de referencia del robot. i. Velocidad de las variables de postura del robot $(\dot{x},\dot{y},\dot{\theta})$. j. Velocidades de las llantas $(\dot{\phi}_r,\dot{\phi}_l)$. k. Velocidades del sistema $(\dot{x}_1,\dot{\theta}_1)$ lineal y angular. l. Aceleraciones del sistema $(\ddot{x}_1,\ddot{\theta}_1)$ lineal y angular.

Prueba 4:

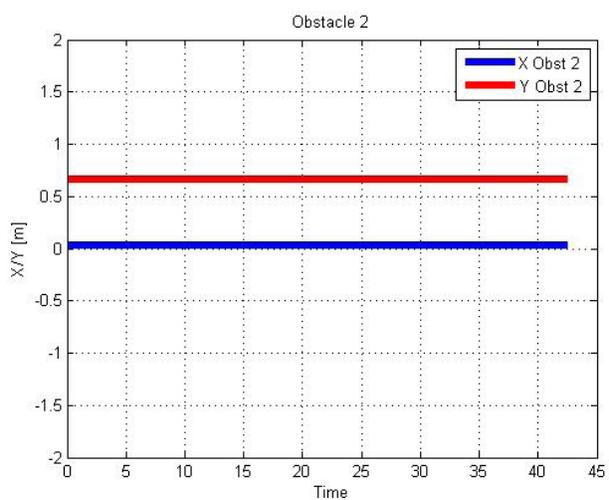
Figure A.XXV.1: Open area reactive path planning(Robot type(2,0)); Seq. of post. snapshot



a

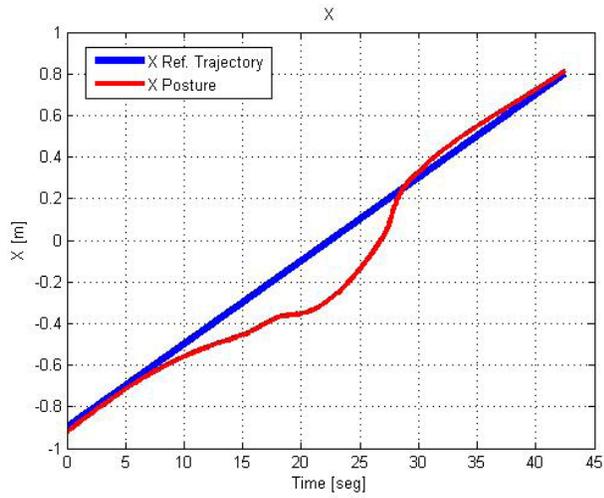


b

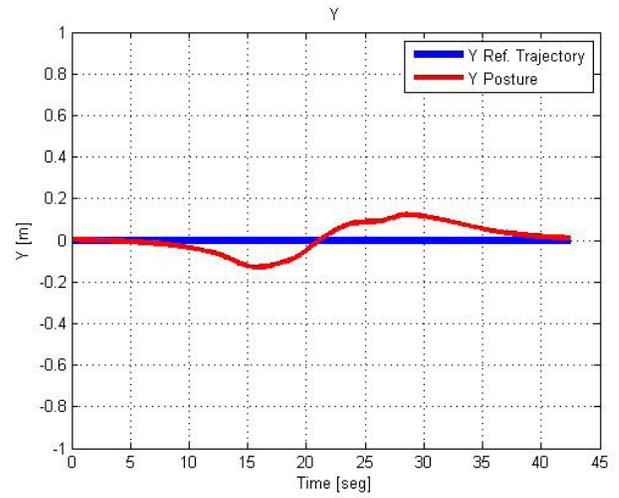


c



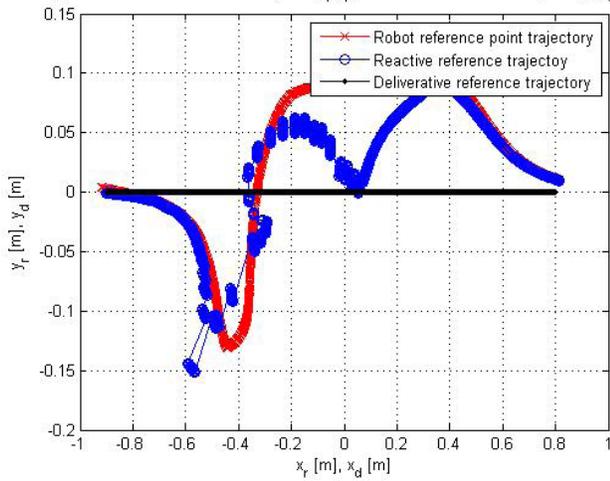


d



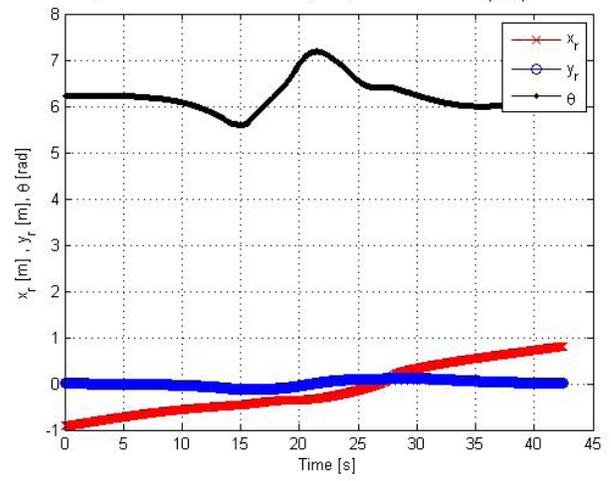
e

Figure A.XXV.2: Robot reference point (x_r, y_r) and Reactive reference trajectory (x_d, y_d)



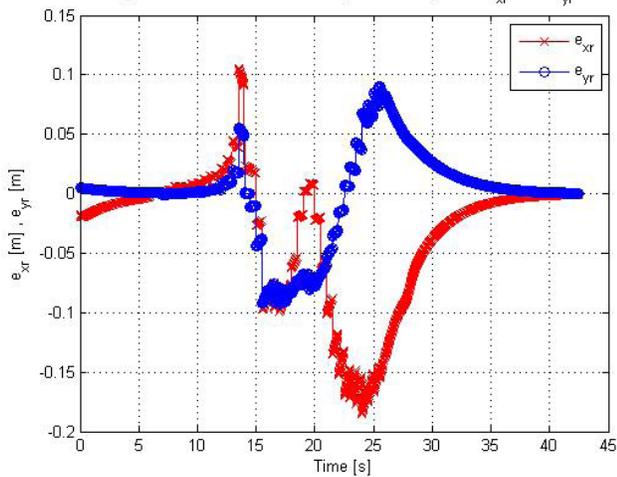
f

Figure A.XXV.3: Robot reference point posture variables: x_r , y_r and θ



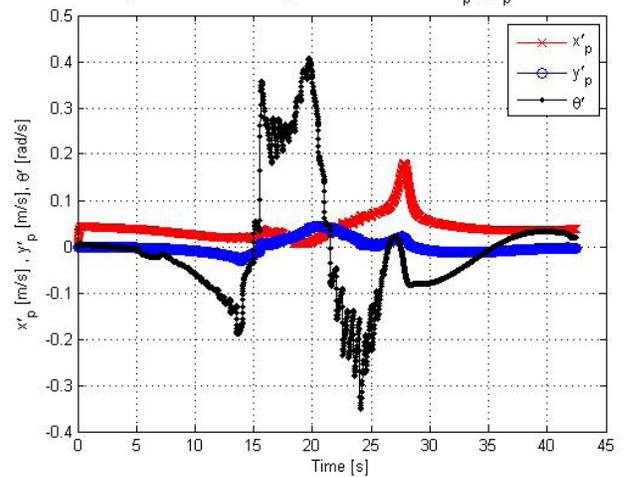
g

Figure A.XXV.4: Robot reference point tracking error: e_{xr} and e_{yr}



h

Figure A.XXV.5: Posture generalized velocities: x'_p , y'_p and θ'



i



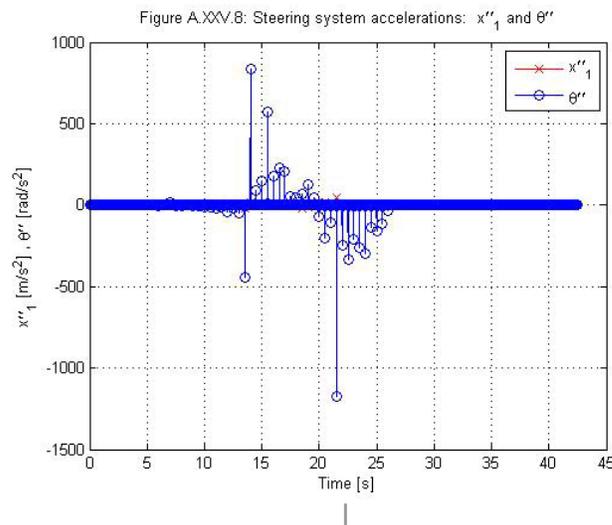
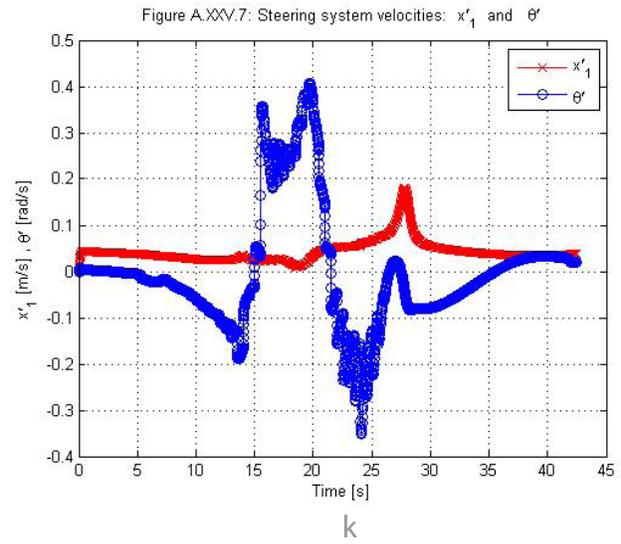
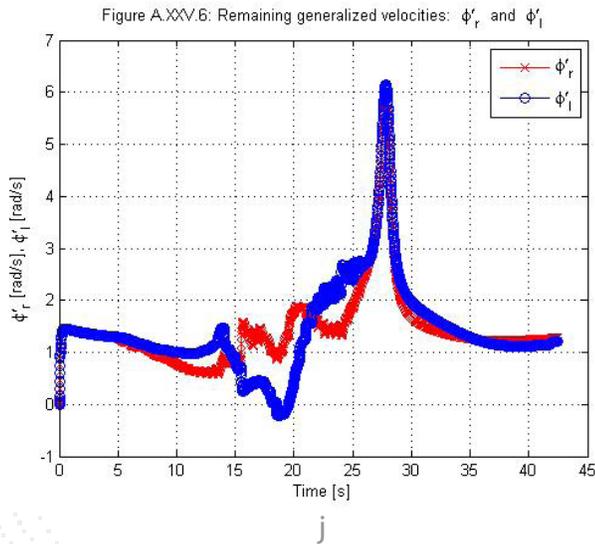


Figura 33: Resultados prueba 4: lazo semicerrado 1 obstáculo fijo y 1 móvil 2. a. Trayectoria final recorrida. b. Posición del obstáculo 1. c. Posición del obstáculo 2. d. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. e. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. f. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. g. Variables de postura del robot (x,y,θ). h. Error de seguimiento del punto de referencia del robot. i. Velocidad de las variables de postura del robot (x',y',θ'). j. Velocidades de las llantas (ϕ'_r,ϕ'_l). k. Velocidades del sistema (x'_1,θ') lineal y angular. l. Aceleraciones del sistema (x''_1,θ'') lineal y angular.

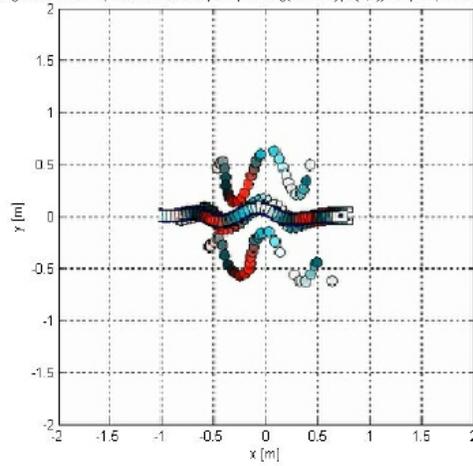


2.3. 2 Obstáculos móviles

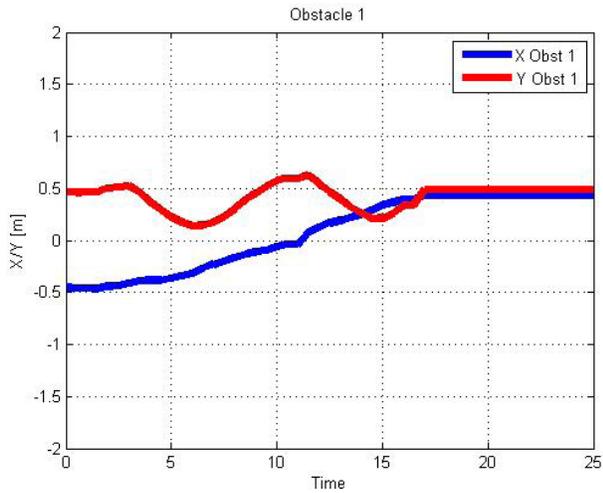
Se presentan dos experimentos con obstáculos móviles en diferente configuración:

Prueba 5:

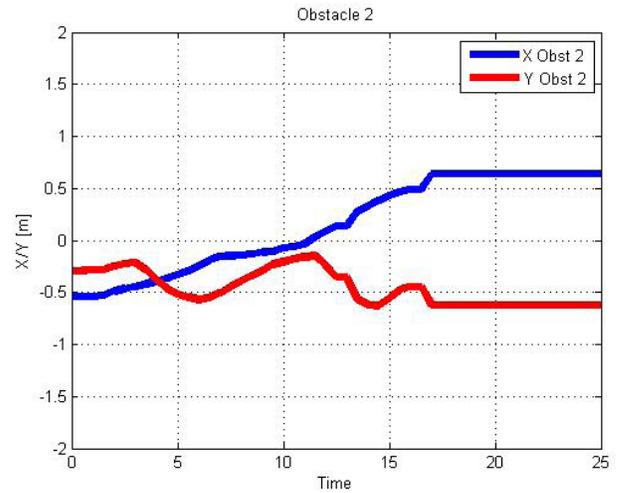
Figure A.XXV.1: Open area reactive path planning (Robot type(2,D)): Seq. of post. snapshot:



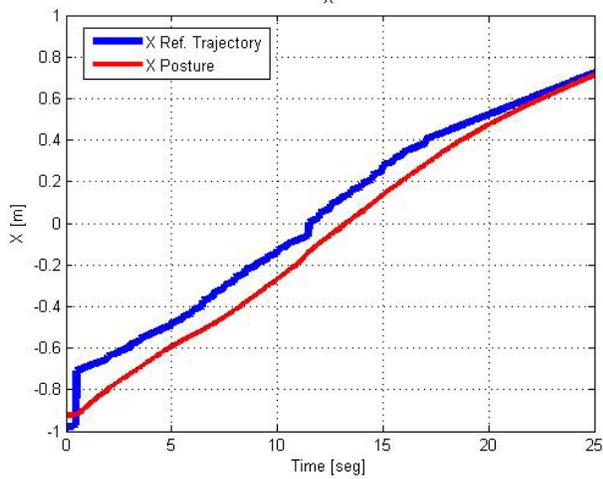
a



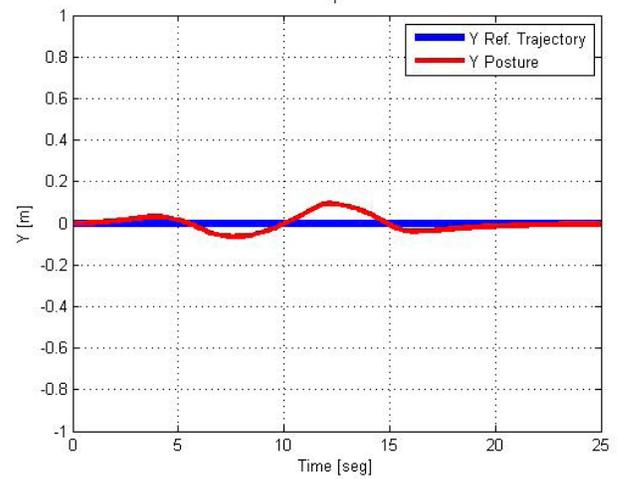
b



c



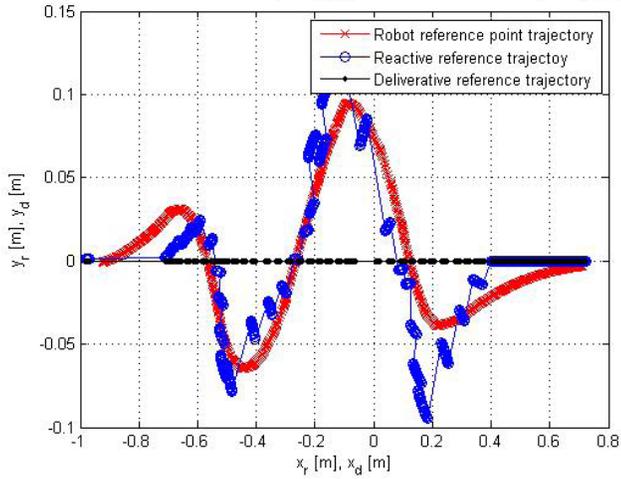
d



e

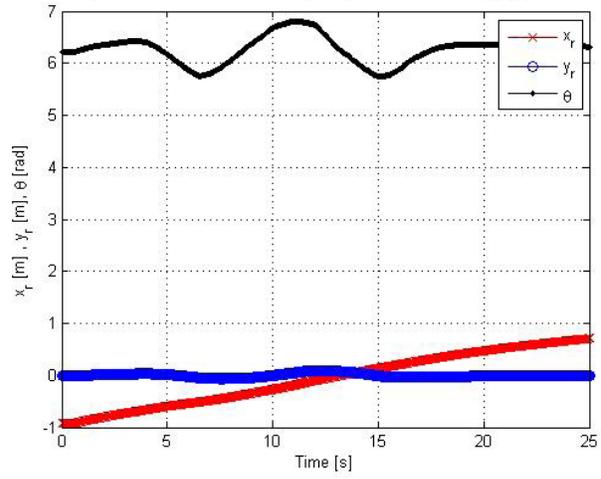


Figure A.XXV.2: Robot reference point (x_r, y_r) and Reactive reference trajectory (x_d, y_d)



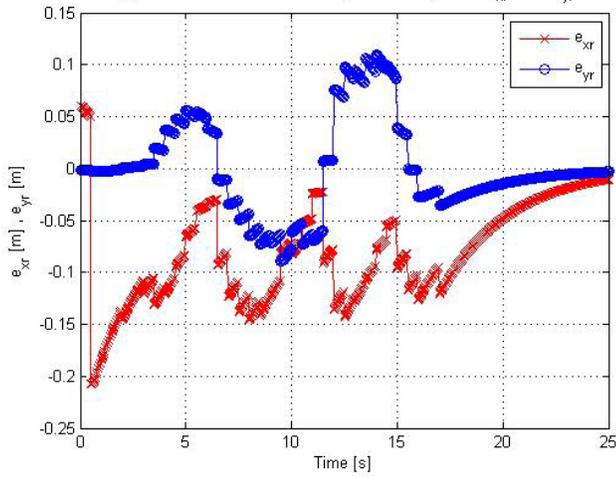
f

Figure A.XXV.3: Robot reference point posture variables: x_r , y_r and θ



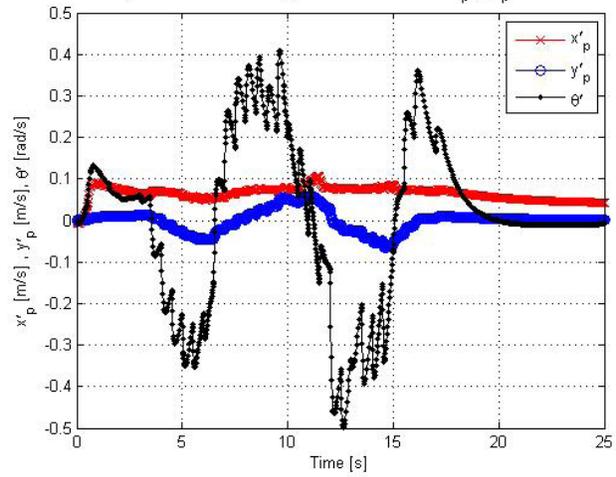
g

Figure A.XXV.4: Robot reference point tracking error: e_{x_r} and e_{y_r}



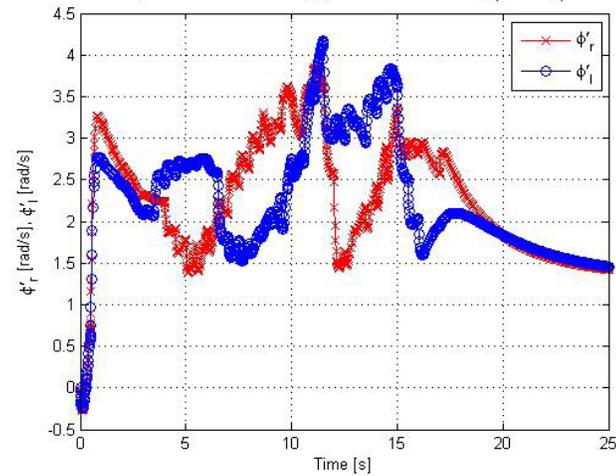
h

Figure A.XXV.5: Posture generalized velocities: x'_p , y'_p and θ'



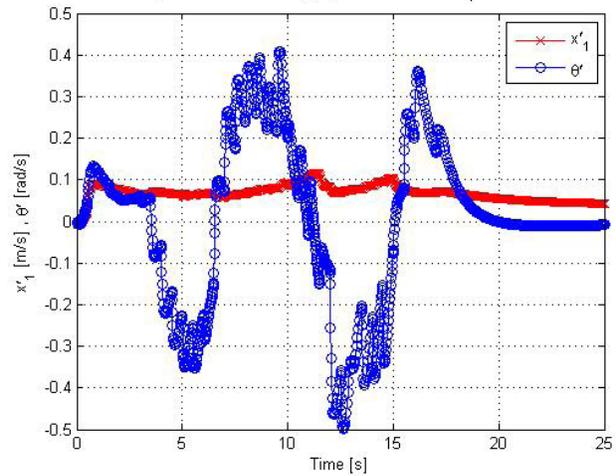
i

Figure A.XXV.6: Remaining generalized velocities: ϕ'_r and ϕ'_l



j

Figure A.XXV.7: Steering system velocities: x'_1 and θ'



k



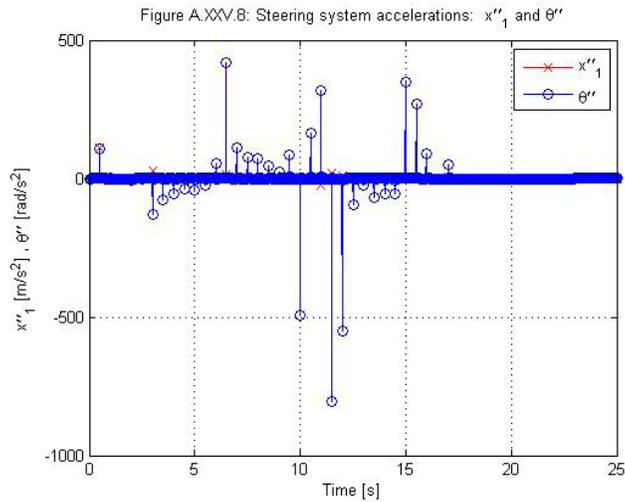
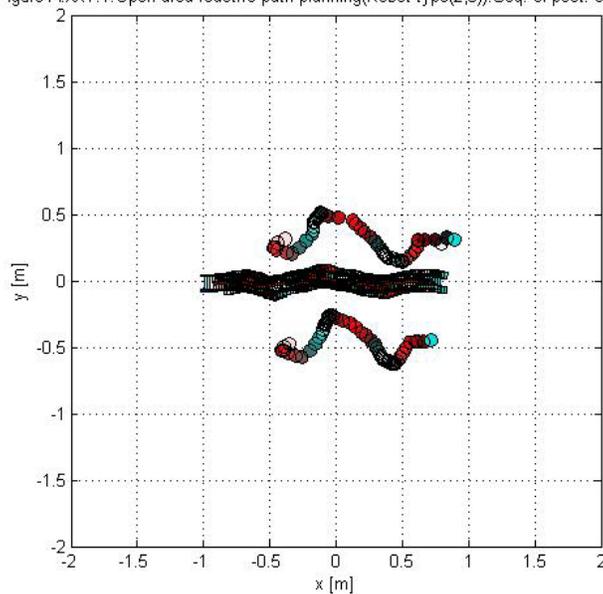


Figura 34: Resultados prueba 5: lazo semicerrado 2 obstáculos móviles 1. a. Trayectoria final recorrida. b. Posición del obstáculo 1. c. Posición del obstáculo 2. d. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. e. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. f. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. g. Variables de postura del robot (x,y,θ). h. Error de seguimiento del punto de referencia del robot. i. Velocidad de las variables de postura del robot (x',y',θ'). j. Velocidades de las llantas (ϕ'_r,ϕ'_l). k. Velocidades del sistema (x'_1,θ') lineal y angular. l. Aceleraciones del sistema (x''_1,θ'') lineal y angular.

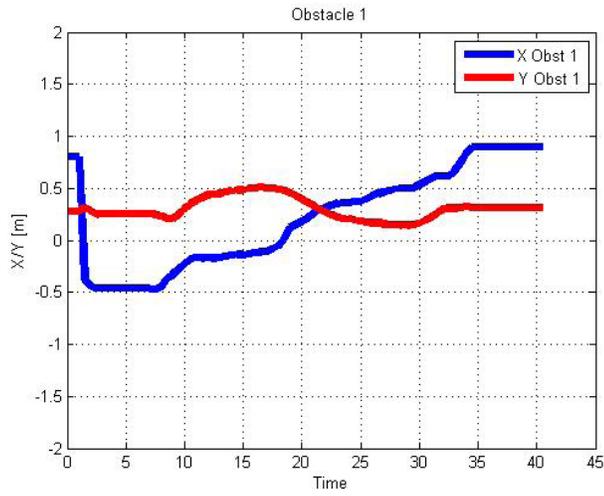
Prueba 6:

Figure A.XXV.1: Open area reactive path planning(Robot type(2,0)):Seq. of post. snapshot:

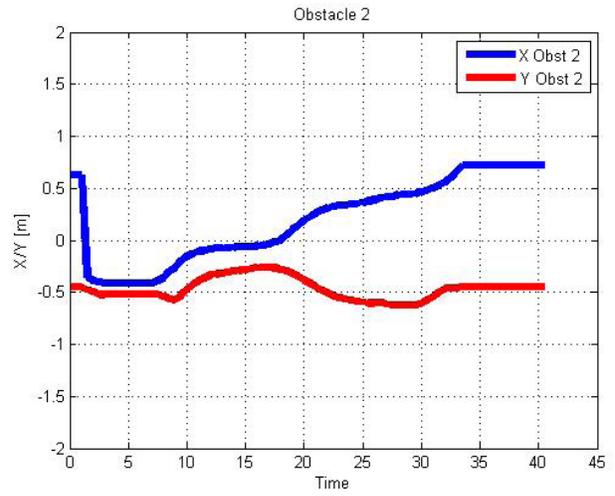


a

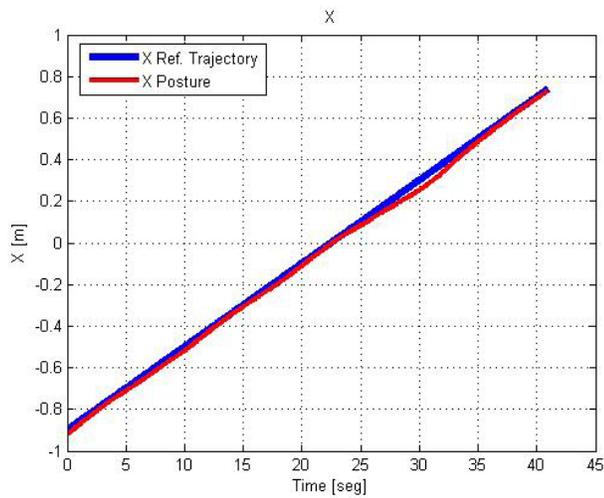




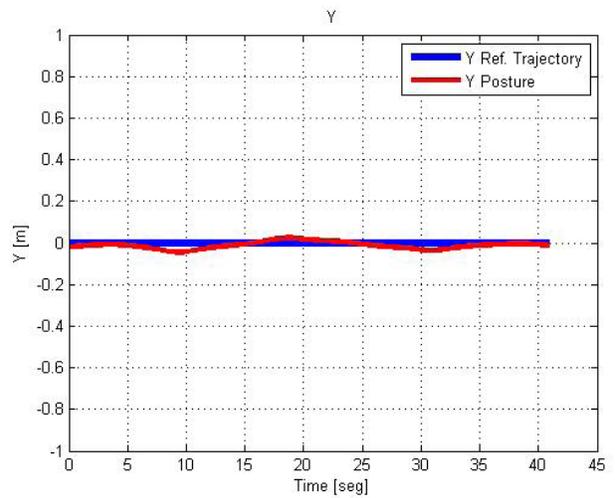
b



c

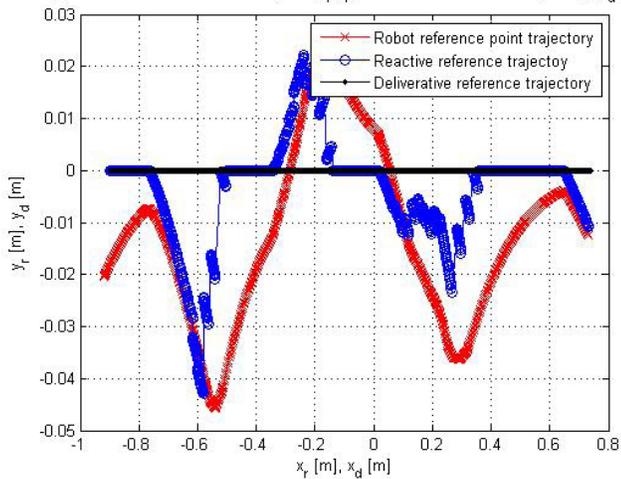


d



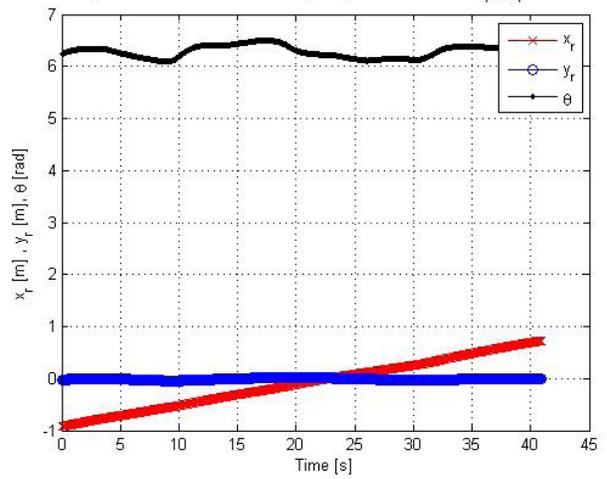
e

Figure A.XXV.2: Robot reference point (x_r, y_r) and Reactive reference trajectory (x_d, y_d)



f

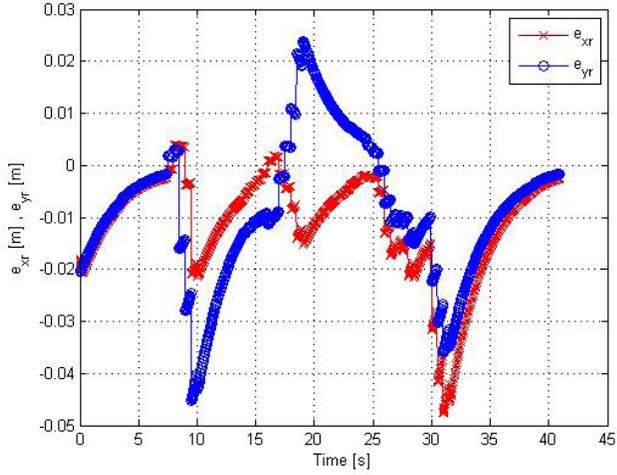
Figure A.XXV.3: Robot reference point posture variables: x_r , y_r and θ



g

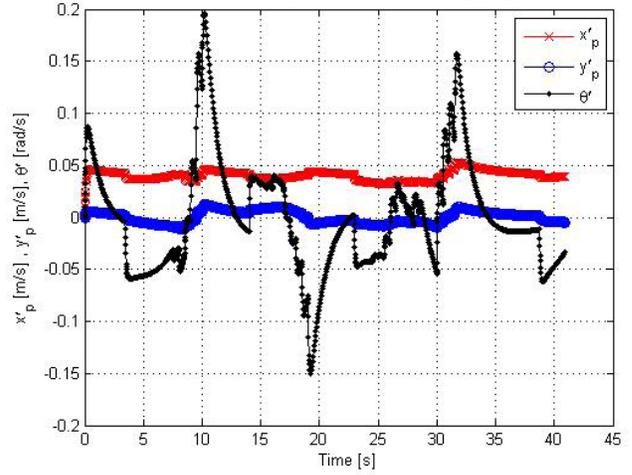


Figure A.XXV.4: Robot reference point tracking error: e_{xr} and e_{yr}



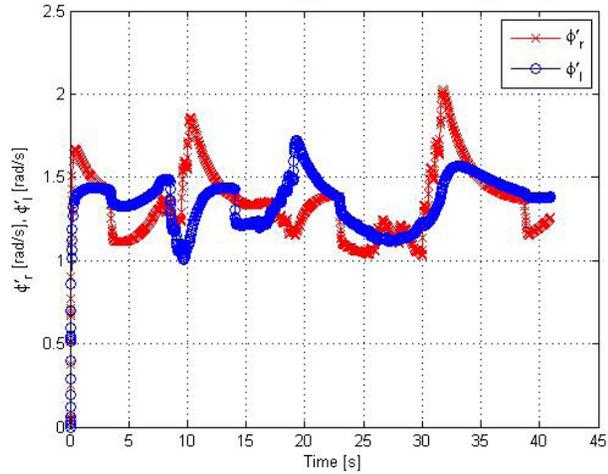
h

Figure A.XXV.5: Posture generalized velocities: x'_p , y'_p and θ'



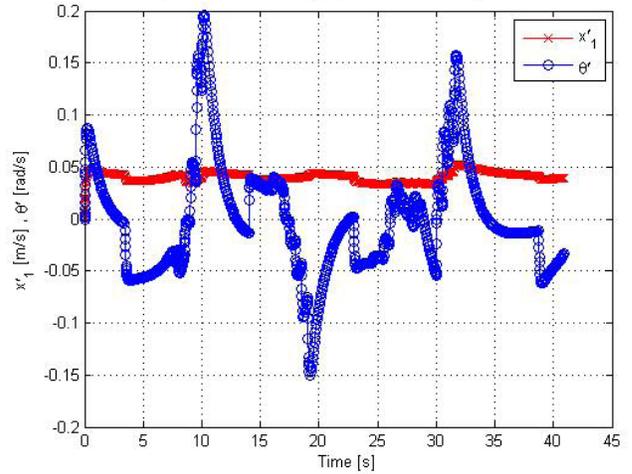
i

Figure A.XXV.6: Remaining generalized velocities: ϕ'_r and ϕ'_l



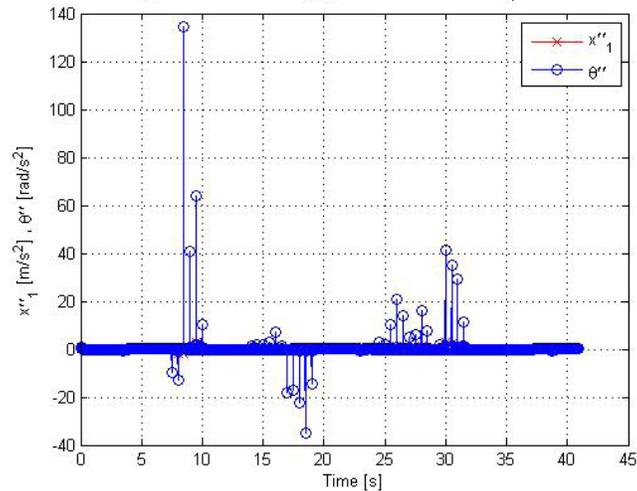
j

Figure A.XXV.7: Steering system velocities: x'_1 and θ'



k

Figure A.XXV.8: Steering system accelerations: x''_1 and θ''



l



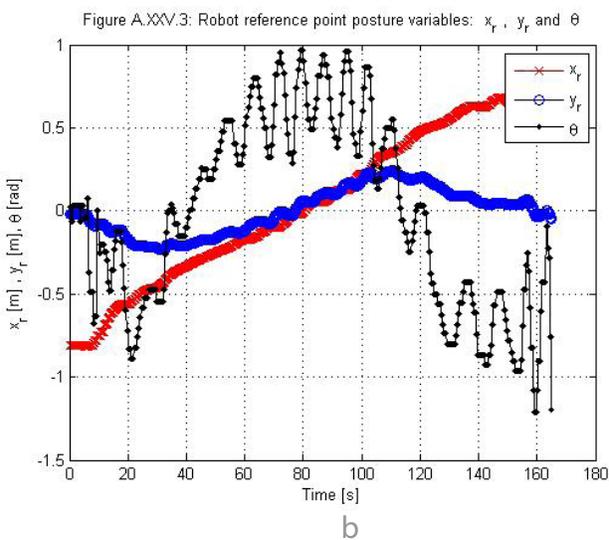
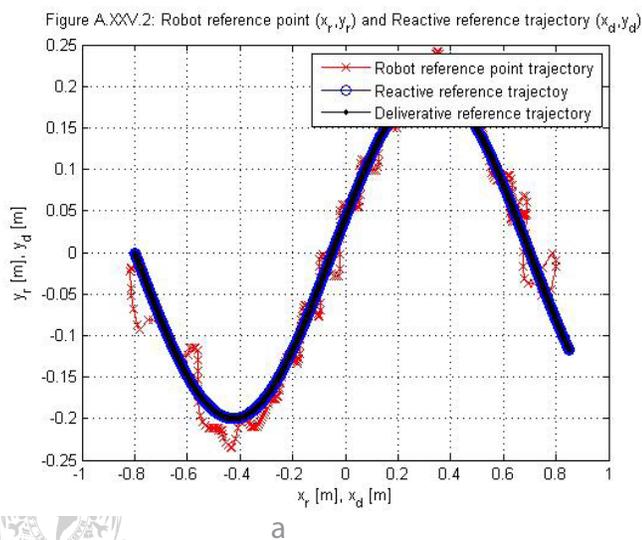
Figura 35: Resultados prueba 6: lazo semicerrado 2 obstáculos móviles 2. a. Trayectoria final recorrida. b. Posición del obstáculo 1. c. Posición del obstáculo 2. d. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. e. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. f. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. g. Variables de postura del robot (x,y,θ). h. Error de seguimiento del punto de referencia del robot. i. Velocidad de las variables de postura del robot ($\dot{x},\dot{y},\dot{\theta}$). j. Velocidades de las llantas ($\dot{\phi}_r,\dot{\phi}_l$). k. Velocidades del sistema ($\dot{x}_1,\dot{\theta}$) lineal y angular. l. Aceleraciones del sistema ($\ddot{x}_1,\ddot{\theta}$) lineal y angular.

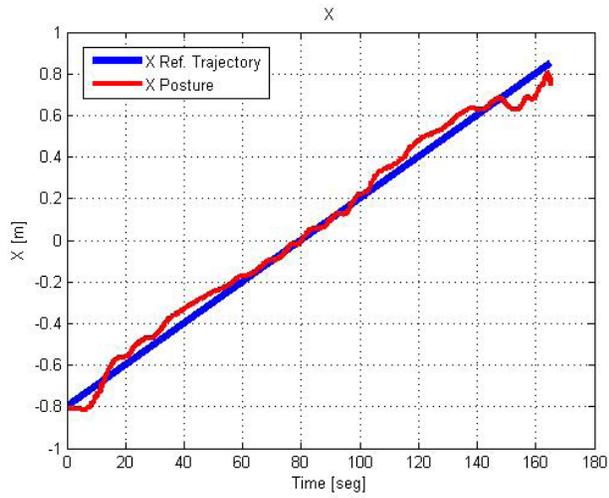
3. Pruebas en lazo cerrado

Estas pruebas son realimentando los datos (posición y velocidad del robot y posición de ambos obstáculos) al controlador del algoritmo. Se presentan seguimientos de trayectorias senoidales para probar su funcionalidad, sin embargo acoplar la deformación de trayectorias, con la capacidad de procesamiento computacional disponible, no resulta factible. Esta es la razón principal de la separación de resultados de lazo semicerrado y cerrado.

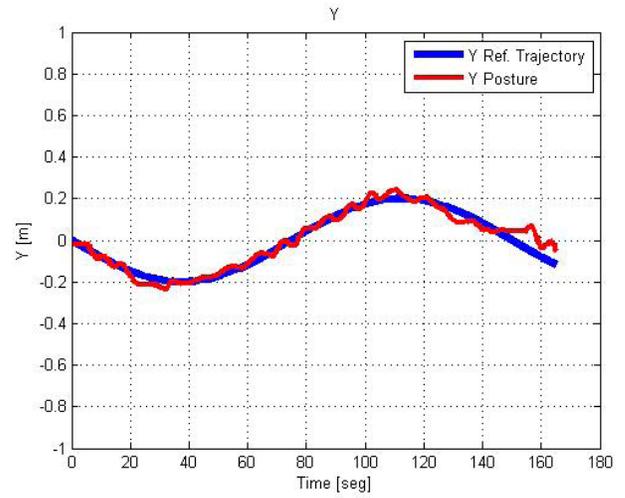
Se presentan dos seguimientos de trayectoria distintos. En estos casos la primera imagen, la captura inicial de los obstáculos, no se presenta pues no se aplica la deformación de trayectoria. Se debe hacer notar que los obstáculos están en el campo de trabajo, pues para poder arrancar la función de visión deben estar presentes los símbolos fiducial, sin embargo se encuentran en la parte de abajo y los parámetros que definen los radios de influencia son muy pequeños, lo suficiente para que no influyan en el seguimiento de la trayectoria.

Prueba 7:



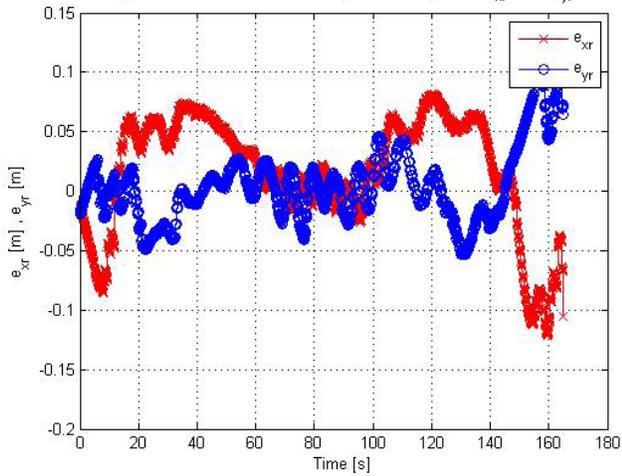


c



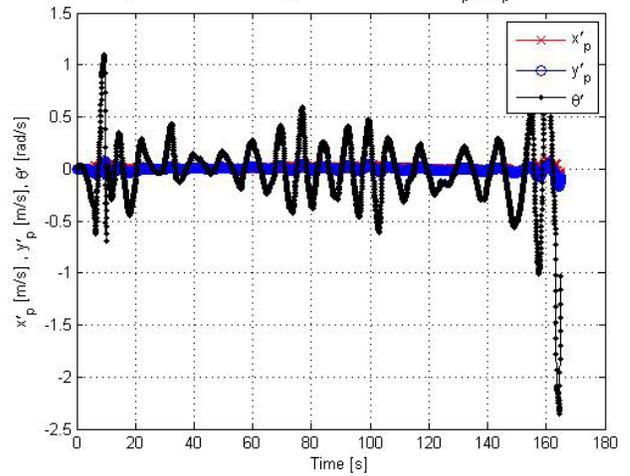
d

Figure A.XXV.4: Robot reference point tracking error: e_{xr} and e_{yr}



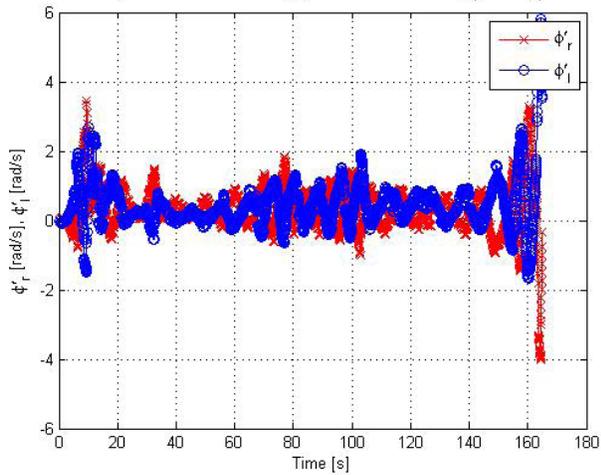
e

Figure A.XXV.5: Posture generalized velocities: x'_p , y'_p and θ'



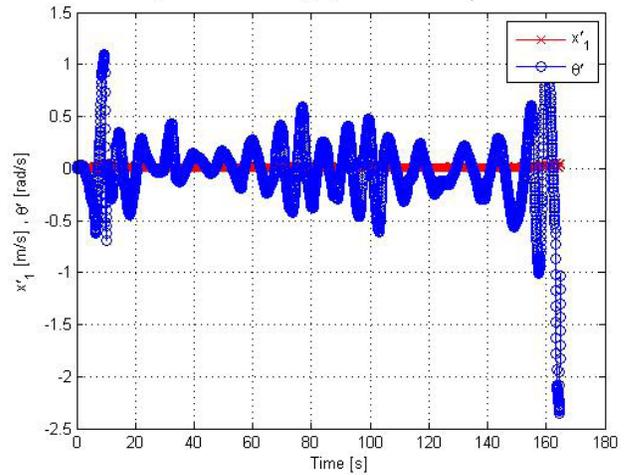
f

Figure A.XXV.6: Remaining generalized velocities: ϕ'_r and ϕ'_l



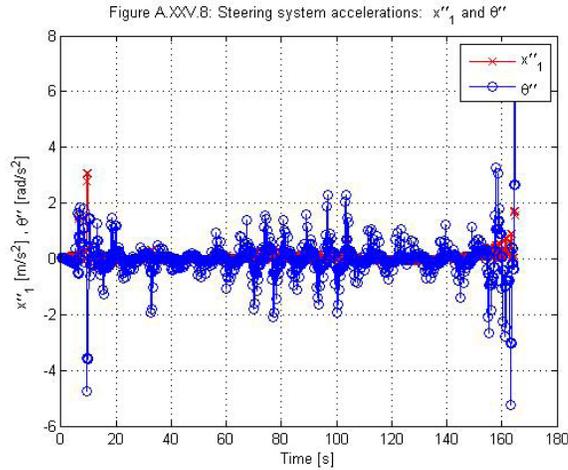
g

Figure A.XXV.7: Steering system velocities: x'_1 and θ'



h

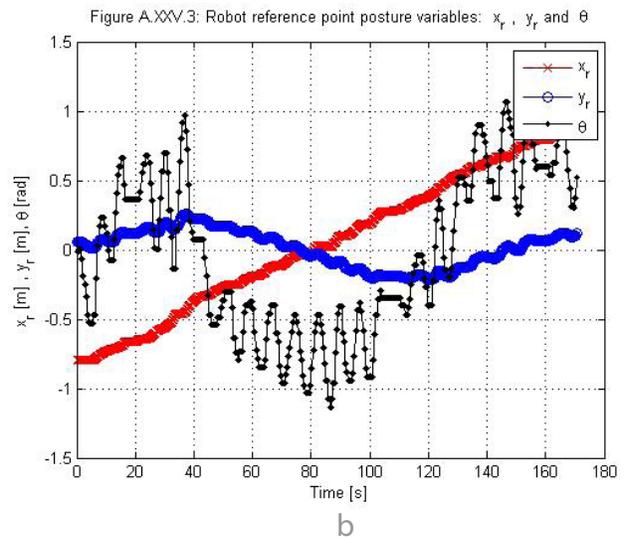
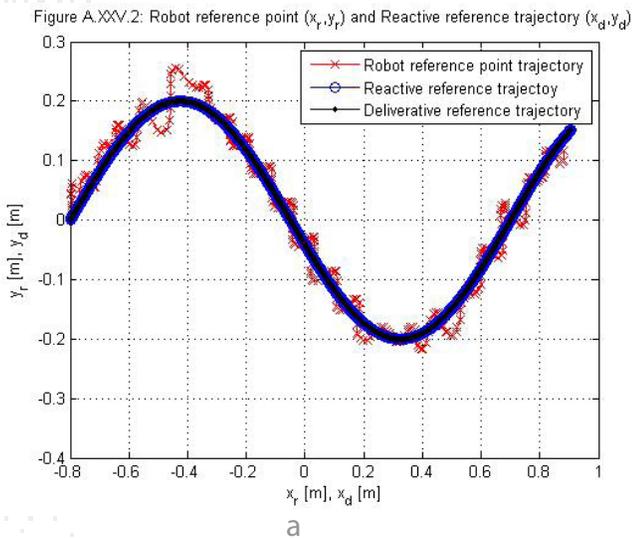


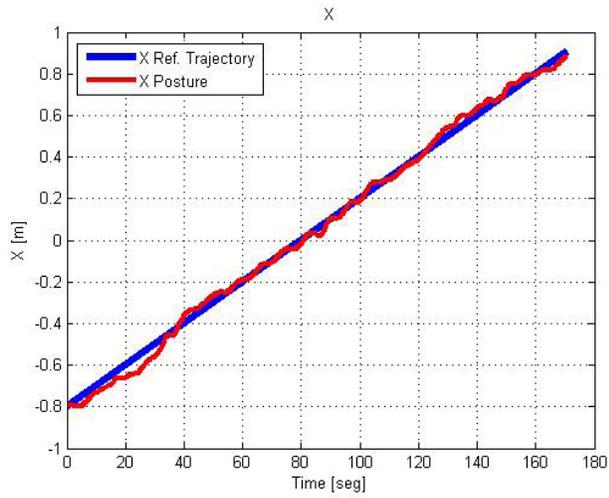


i

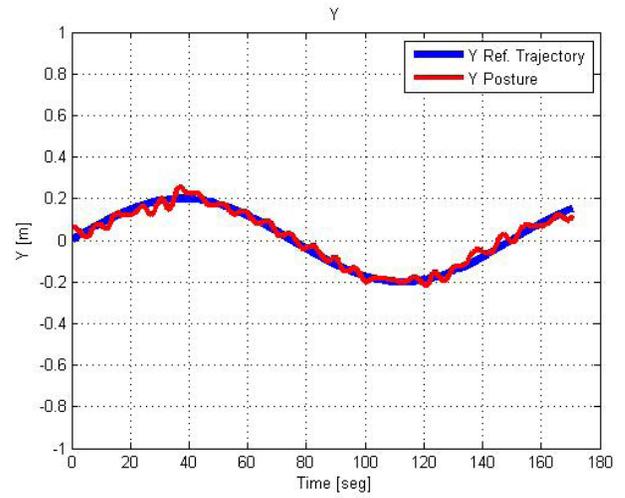
Figura 36 Resultados prueba 7: lazo cerrado seguimiento de trayectoria senoidal negativa a. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. b. Variables de postura del robot (x, y, θ). c. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. d. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. e. Error de seguimiento del punto de referencia del robot. f. Velocidad de las variables de postura del robot ($\dot{x}, \dot{y}, \dot{\theta}$). g. Velocidades de las llantas ($\dot{\phi}_r, \dot{\phi}_l$). h. Velocidades del sistema ($\dot{x}_1, \dot{\theta}$) lineal y angular. i. Aceleraciones del sistema ($\ddot{x}_1, \ddot{\theta}$) lineal y angular.

Prueba 8:



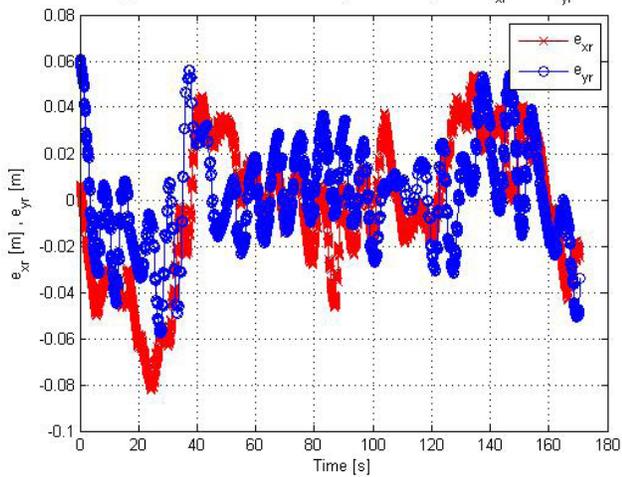


c



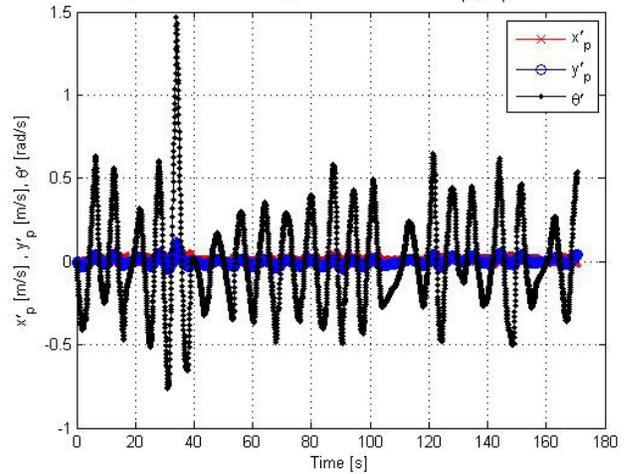
d

Figure A.XXV.4: Robot reference point tracking error: e_{xr} and e_{yr}



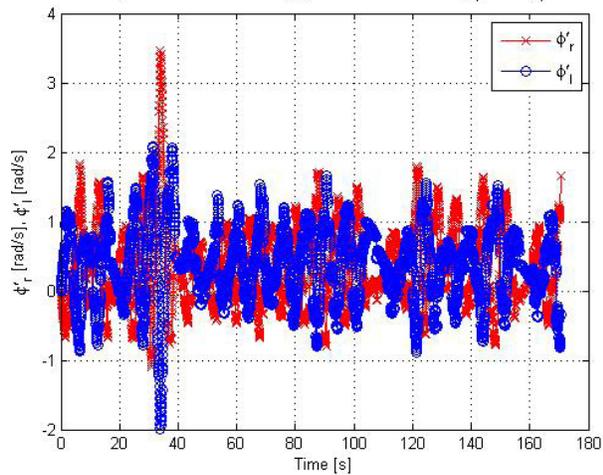
e

Figure A.XXV.5: Posture generalized velocities: x'_p , y'_p and θ'



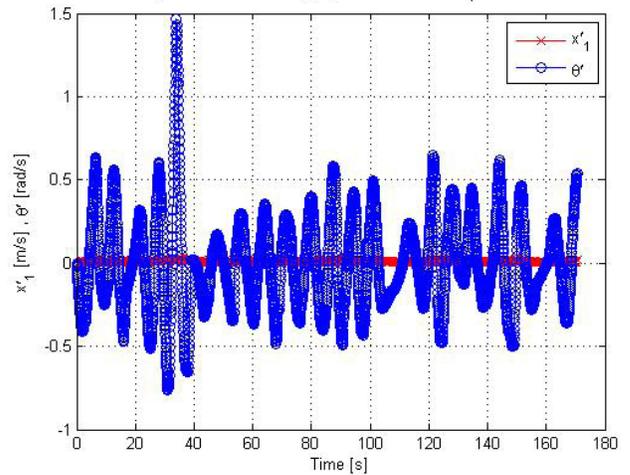
f

Figure A.XXV.6: Remaining generalized velocities: ϕ'_r and ϕ'_l



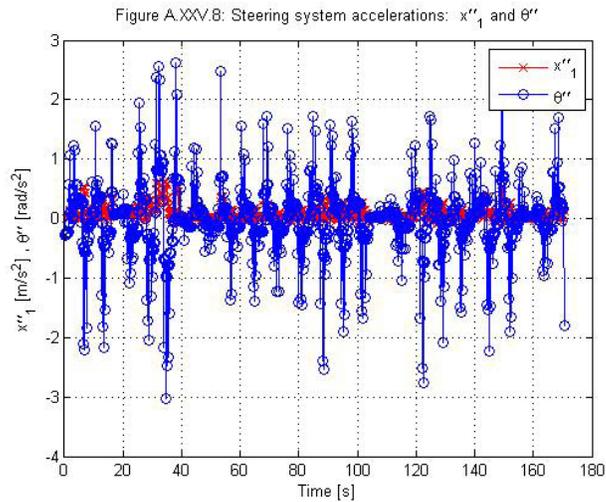
g

Figure A.XXV.7: Steering system velocities: x'_1 and θ'



h





i

Figura 37. Resultados prueba 8: lazo cerrado seguimiento de trayectoria senoidal positiva. a. Comparativa trayectoria predefinida-trayectoria deformada-trayectoria del robot. b. Variables de postura del robot (x,y,θ). c. Comparativa de X en la trayectoria de referencia y la real con respecto al tiempo. d. Comparativa de Y en la trayectoria de referencia y la real con respecto al tiempo. e. Error de seguimiento del punto de referencia del robot. f. Velocidad de las variables de postura del robot ($\dot{x},\dot{y},\dot{\theta}$). g. Velocidades de las llantas ($\dot{\phi}_r;\dot{\phi}_l$). h. Velocidades del sistema ($\dot{x}_1,\dot{\theta}$) lineal y angular.i. Aceleraciones del sistema ($\ddot{x}_1,\ddot{\theta}$) lineal y angular.

4. Discusión de resultados

Podemos observar el funcionamiento de la deformación de trayectorias de manera muy clara en los dos primeros experimentos, las pruebas con obstáculos fijos (Figuras 30 y 31). Los resultados de estas pruebas muestran la deformación en posición y velocidad (Figuras 30 g. i. y 31 g. i.). En posición, el camino programado (hablando especialmente puesto que no está explícito el parámetro del tiempo en las primeras gráficas) es una línea recta, y por la posición de los obstáculos se hacen 2 curvas, una hacia arriba y otra hacia abajo. Existe una diferencia sustancial entre las 2 pruebas, en la primera se deforma hacia ambos sentidos del eje Y y en la segunda sólo hacia el sentido positivo. Esto depende de la posición de los obstáculos, en el primer caso existe un espacio entre ambos (considerando el radio de influencia de los obstáculos y del robot) en el cual el robot regresa a su trayectoria inicial (la línea recta con velocidad constante) y el segundo obstáculo se encuentra ligeramente arriba de la horizontal por lo que la deformación de la trayectoria es óptima hacia abajo. El algoritmo toma partida para elegir el rumbo de la deformación basándose en los parámetros



del sistema, colocación de los obstáculos, trayectoria definida, objetivo final, rangos de influencia del robot y de los obstáculos, como bien se menciona en la explicación teórica de la deformación de trayectorias (véase II.1). En la velocidad (*Figuras 30 i. k. y 31 i. k.*) se nota la deformación en sus cambios, la trayectoria predefinida está con una velocidad constante por lo tanto los cambios en ella son muestra y consecuencia de los obstáculos interfiriendo con el robot. Como se puede observar el error es pequeño (*Figuras 30 h. y 31 h.*), en el rango de milímetros, consecuencia de ciertos deslizamientos de las llantas con el piso, el redondeo de los datos de salida (comandos de control), entre otros. El error inicial es grande puesto que la posición inicial virtual (algoritmo en computadora) está definido mediante constantes en Simulink, y la posición del robot en el espacio de trabajo es una aproximación a la programada. Estos errores pequeños no se repiten en las pruebas subsecuentes (crecen), en este primer experimento los errores pequeños son consecuencia de que no existe movimiento de los obstáculos, por lo tanto no se generan picos muy grandes de velocidad que puedan provocar ciertos deslizamientos indeseados.

En el caso del experimento 3, un obstáculo fijo y uno móvil, la deformación se basa de manera sustancial en el móvil (*Figura 32 a.*). El fijo, por el parámetro de radio de influencia y de su posición en el espacio de trabajo, pasa desapercibido. Los errores en esta prueba son mayores (*Figura 32 h.*), ahora en rango de centímetros, precisamente en consecuencia del obstáculo móvil. Mientras se acerca el obstáculo al robot existen ciertos picos de velocidad y aceleración (*Figuras 32 i. k. l.*) que son demandados mediante los comandos de control a las llantas del robot (*Figura 32 j.*). El error, como bien mencionamos, crece debido al obstáculo móvil, ahora la velocidad de aproximación aumenta puesto que el obstáculo se mueve, en algunos casos hacia el robot o alrededor de él, y esto es una gran influencia en los comandos de control. Haciendo una comparativa de las *Figuras 32 j. y 33 j.* podemos observar que los comandos de control no son parecidos, esto es consecuencia del tipo de aproximación del obstáculo hacia el robot, el obstáculo de alguna manera puede "retrasar" al robot de su trayectoria, generando errores más grandes (Comparativa *Figuras 32 h. y 33 h.*) y como consecuencia comandos de control mayores para compensar. Algo interesante de mencionar de la Prueba 4 (*Figura 33 a.*) es el comportamiento del robot ante un obstáculo que se acerca directamente a él. Por el algoritmo de deformación reactiva de trayectorias el robot es capaz de modificar no sólo su posición en el plano, sino su velocidad en caso de ser necesario. En este caso al momento de que se aproxima el obstáculo intenta rodearlo por arriba, como el obstáculo no cambia su velocidad el robot disminuye la propia para poder rodearlo por el otro lado, y acelera después para alcanzar el punto de la trayectoria deseada. Este



comportamiento se muestra en la *Figura 33 i.* observando con especial atención las variables de velocidad de x, θ , como vemos, al momento de la desaceleración en θ existe una desaceleración (pequeña) en x , y cuando cambia el robot de dirección, x comienza a acelerarse para alcanzar a la trayectoria programada. Este comportamiento ejemplifica la gran funcionalidad de la deformación de trayectorias y, en parte, la diferencia con la deformación de caminos.

Las pruebas con los 2 obstáculos móviles son las que muestran la deformación más compleja, ya que es un ambiente totalmente dinámico (considerando que sólo existen 2 obstáculos en el campo de trabajo) y en ningún momento se detienen, están en constante movimiento alrededor del robot. Las pruebas mostradas son las de mayor parecido con la simulación de donde partimos este trabajo, en la cual los obstáculos seguían una trayectoria senoidal (con diferente amplitud y frecuencia) junto al robot. Es una aproximación a la situación donde el robot está desplazándose entre dos personas caminando. Es compleja no por la trayectoria deformada en sí, sino porque el algoritmo ahora está procesando datos dinámicos tanto para el robot como para ambos obstáculos. La Prueba 5 (*Figura 34*) es menos satisfactoria que la Prueba 6 (*Figura 35*) sólo en el sentido del seguimiento de la trayectoria (*Figuras 34 h. y 35 h.*), puesto que la evasión de obstáculos se realiza en ambas pruebas (*Figuras 34 a. y 35 a.*) exitosamente. La primera diferencia, y la más clara, es la trayectoria predefinida, la primera es con perfil de velocidades variante y la segunda constante. Los errores tan grandes que observamos se presentan en su mayoría por la mayor complejidad del sistema, lo que nos obliga a hacer más grande la temporización de la función de visión, manteniendo el dato constante por un mayor periodo de tiempo, además de que se acompleja la solución de las ecuaciones puesto que ya no es constante la velocidad.

Estos errores son grandes, es claro por la *Figura 34 h.*, sin embargo se compensa por el radio de influencia tan grande que se manejó en esta prueba, siendo esto la razón principal por la que, a pesar de los errores tan grandes capturados en las gráficas la evasión de obstáculos se logra. Esta fue una de las primeras pruebas realizadas, por eso tiene una trayectoria, radios de influencia, entre otros parámetros, distintos a las demás, sin embargo consideramos incluirla en los resultados para dejar en claro el funcionamiento del algoritmo bajo circunstancias distintas, no sólo de los obstáculos, sino del robot mismo.

La Prueba 6 (*Figura 34 a.*) se realizó con todos los parámetros iguales que en las demás pruebas, sólo cambiando ahora a los obstáculos. De igual manera que las anteriores, ahora los errores de

seguimiento se reducen y la evasión de obstáculos por medio de deformación reactiva de trayectorias se realiza satisfactoriamente. En este caso, por la trayectoria predefinida, no existen cambios tan grandes en la velocidad y aceleración (*Figuras 35 i. k. l.*), en



comparación con la Prueba 5 (*Figuras 34 i. k. l.*). Esto se puede observar de manera más clara haciendo una comparativa de las *Figuras 34 j. y 35 j.* donde podemos observar el grado de diferencia entre los comandos de control hacia las llantas del robot.

Las últimas dos pruebas son de seguimiento de una trayectoria sin obstáculos y en lazo de control totalmente cerrado. La razón por la que no hay obstáculos es precisamente el lazo cerrado de control. Los resultados confirman (*Figuras 36 a. y 37 a.*) que el seguimiento de las trayectorias senoidales (positiva y negativa respectivamente) se genera de manera correcta. Esto quiere decir que el robot siguió la trayectoria predefinida, sin embargo los errores (*Figuras 36 h. y 37 h.*) en cada una de las pruebas no son precisamente pequeños ni despreciables. Esto se debe a ciertos parámetros de la prueba que nos resultó imposible cambiar y se tratarán más a fondo en la conclusión del trabajo. Algo importante de notar es el comportamiento del ángulo θ (*Figuras 36 g. i. y 37 g. i.*), que se debe a que el robot está siguiendo al punto de la trayectoria preestablecida y que, por cuestiones de la capacidad y velocidad de procesamiento de la computadora, el robot generaba ciertos sobrepasos, no consecuencia del controlador sino de la velocidad de recepción y manipulación de datos.

Las gráficas restantes (*Figuras 36 j. k. l. y 37 j. k. l.*), correspondientes a velocidades de las llantas, del sistema y aceleraciones del sistema, muestran el comportamiento del robot alrededor de la prueba. Es de notar que existen muchos picos de velocidad y aceleración, los datos están muy perturbados y no precisamente por ruido, es consecuencia de que el robot aceleraba y desaceleraba constantemente con giros poco suaves. Esta es la razón principal por la que los obstáculos no aparecieron en la prueba (virtualmente, físicamente ahí estaban pero con radios de influencia muy cercanos a cero), la evasión de obstáculos generaría velocidades y aceleraciones que, si bien el robot podía cumplir, la cámara no podía percibir, provocando que se perdiera el símbolo fiducial, mantuviera el último dato y, por el controlador, los comandos de control crecieran aún más, generando un experimento fallido.

En general los experimentos presentados se realizaron para probar la teoría de deformación de trayectorias bajo distintas circunstancias y con comportamiento distinto de los obstáculos, análogos a los que se podrían presentar en una aplicación real en particular.

En general, las gráficas d. y e. de todas las pruebas nos muestran la deformación de trayectorias en sí, pues manejan una comparativa entre la trayectoria predefinida y la final con el factor del tiempo de manera explícita, que es la gran diferencia de esta teoría con las presentadas en trabajos previos. Podemos observar que en todas las gráficas se genera una deformación de la trayectoria y cómo corresponden en espacio y tiempo al final de cada experimento.



V. Conclusiones y trabajo futuro

1. Conclusiones

En general la teoría sobre deformación reactiva de trayectorias se considera probada mediante los experimentos en tiempo real descritos en el presente documento. Esta tarea se realiza no sólo para una situación en particular, de hecho la variedad de resultados presentados implica la cantidad de experimentos realizados con la diversidad de situaciones, consideradas desde la más sencilla (obstáculos fijos) hasta la más compleja (ambos obstáculos móviles) creando así una gran gama de posibilidades de evasión.

Es notable que la deformación en una situación real sería con mayor número de obstáculos y mayor tamaño para ellos y para el robot por la misma naturaleza de la aplicación, la forma de detección de los obstáculos y obtención de información de los robots sería más complicada de implementar en una aplicación real, sin embargo, es importante mencionar que la teoría es escalable sin cambios drásticos, por lo que las modificaciones recaen totalmente en el sistema encargado de la obtención de información de campo y el envío y actuación de los comandos de control.

La división de experimentos en lazo semicerrado y lazo cerrado es consecuencia de los resultados obtenidos en lazo cerrado. Al realizar la prueba, incluyendo a la deformación reactiva de trayectorias (no sólo el seguimiento de una trayectoria preestablecida), se obtienen respuestas no satisfactorias consecuencia de la temporización de la función de visión. En lazo semicerrado los datos provienen del modelo cinemático y se refrescan en cada una de las iteraciones del programa, a diferencia del lazo cerrado donde los datos son refrescados a una frecuencia ligeramente mayor a las 2 veces por segundo, esto debido a la velocidad y capacidad de procesamiento de la computadora. Esta inconsistencia es probada mediante los dos últimos experimentos, donde es claro que el controlador funciona y sigue a la trayectoria deseada con cierto sobrepaso consecuencia de la temporización.

Con base en los resultados obtenidos y, de cierta manera en un principio de separación, podemos llegar a la conclusión de que el sistema funciona en su totalidad en lazo cerrado. El algoritmo de deformación de trayectorias genera de manera correcta objetivos virtuales para llegar de un punto inicial a un punto final evadiendo obstáculos y siguiendo una trayectoria predefinida (geométrica y temporalmente); en otras palabras, deforma de manera reactiva la trayectoria inicial para generar una nueva que evada los obstáculos que se encuentren dentro de ciertos radios de influencia, lo cual podemos observar claramente en las pruebas en lazo semicerrado (*Figuras 27, 28, 29, 30, 31, 32*), en donde el algoritmo de control funciona al enviar los comandos de control para el seguimiento de una trayectoria predefinida (llámese trayectoria programada inicial o trayectoria



deformada de manera reactiva) con ciertos errores y sobrepasos indeseados, lo que podemos observar en las pruebas en lazo cerrado (*Figuras 33, 34*), por lo tanto es importante apuntar que al ser dos algoritmos (el de deformación reactiva de trayectorias y el de controlador no lineal) que no están ligados (es decir, que pueden funcionar de manera independiente) y aunado a que en las pruebas de lazo semicerrado el controlador estaba en funcionamiento sólo que con datos provenientes del modelo cinemático del robot, podemos concluir con certeza que, tal como se menciona inicialmente, el sistema funciona en su totalidad en lazo cerrado.

Ahora bien, en este trabajo no se alcanza ese punto de funcionalidad y la causa más probable que se encontró es la capacidad de procesamiento en materia de cómputo en general, el procesamiento matemático, la visión artificial, el sistema de enlace entre software y la comunicación computadora-robot.

2. Trabajo futuro

Este trabajo deja una base sólida sobre deformación reactiva de trayectorias para robótica móvil. Como primera aproximación podríamos decir que el funcionamiento en lazo semicerrado es suficiente para ciertas aplicaciones a nivel laboratorio. El trabajo subsecuente es probar la hipótesis sobre las causas de los inconvenientes con la deformación en lazo cerrado. Una vez determinada y solucionada, las aplicaciones resultan vastas.

Cualquier sistema en robótica móvil requiere de sistemas de navegación con evasión de obstáculos, sistemas que tengan como finalidad no sólo la evasión, sino que cumplan con diferentes funciones o aplicaciones donde la evasión de obstáculos sea crucial para realizar la tarea para lo que fueron programados de manera correcta. Robots manipuladores tipo serial o paralelos podrían aprovechar las ventajas de algoritmos de este tipo para aumentar su rango de manipulabilidad al ser capaces de desplazarse en su entorno evitando colisiones.

En general, la base teórica probada en este trabajo es utilizable en diversos rubros a nivel laboratorio con el sistema propuesto o incluso en otras aplicaciones a nivel industrial, con los cambios que conlleva, por supuesto, el cambio de una aplicación de laboratorio a una a nivel industrial.



VI. Apéndice

| 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. |



Apéndice 2

Código de function S de visión

%1 -> Obstáculo 1 (FD 3)

%2 -> Obstáculo 2 (FD 4)

%3 -> Robot (FD 1)

```
global a3 x1 x2 x3 y1 y2 y3 c f1 f2 f3 vx3 vy3 vth3 v3 a3ant k kant a3s;
```

```
kant=u(1);
```

```
a3ant=u(2);
```

```
import TUIO.*;
```

```
tc = TuioClient();
```

```
tc.connect();
```

```
for i = 1 : 2
```

```
    c=tc.getTuioObjects().size();
```

```
    if (c > 0)
```

```
        f1 = tc.getTuioObjects().get(0).getSymbolID();
```

```
        switch f1
```

```
            case 1
```

```
                a3 = tc.getTuioObjects().get(0).getAngle();
```

```
                x3 = tc.getTuioObjects().get(0).getPosition().getX();
```

```
                y3 = tc.getTuioObjects().get(0).getPosition().getY();
```

```
                vx3 = tc.getTuioObjects().get(0).getXSpeed();
```

```
                vy3 = tc.getTuioObjects().get(0).getYSpeed();
```

```
                vth3 = tc.getTuioObjects().get(0).getRotationSpeed();
```



```
v3 = tc.getTuioObjects().get(0).getMotionSpeed();
```

```
case 3
```

```
x1 = tc.getTuioObjects().get(0).getPosition().getX();
```

```
y1 = tc.getTuioObjects().get(0).getPosition().getY();
```

```
case 4
```

```
x2 = tc.getTuioObjects().get(0).getPosition().getX();
```

```
y2 = tc.getTuioObjects().get(0).getPosition().getY();
```

```
otherwise
```

```
end
```

```
if(c > 1)
```

```
f2 = tc.getTuioObjects().get(1).getSymbolID();
```

```
switch f2
```

```
case 1
```

```
a3 = tc.getTuioObjects().get(1).getAngle();
```

```
x3 = tc.getTuioObjects().get(1).getPosition().getX();
```

```
y3 = tc.getTuioObjects().get(1).getPosition().getY();
```

```
vx3 = tc.getTuioObjects().get(1).getXSpeed();
```

```
vy3 = tc.getTuioObjects().get(1).getYSpeed();
```

```
vth3 = tc.getTuioObjects().get(1).getRotationSpeed();
```

```
v3 = tc.getTuioObjects().get(1).getMotionSpeed();
```

```
case 3
```

```
x1 = tc.getTuioObjects().get(1).getPosition().getX();
```

```
y1 = tc.getTuioObjects().get(1).getPosition().getY();
```



case 4

```
x2 = tc.getTuioObjects().get(1).getPosition().getX();
```

```
y2 = tc.getTuioObjects().get(1).getPosition().getY();
```

otherwise

end;

if(c > 2)

```
f3 = tc.getTuioObjects().get(2).getSymbolID();
```

switch f3

case 1

```
a3 = tc.getTuioObjects().get(2).getAngle();
```

```
x3 = tc.getTuioObjects().get(2).getPosition().getX();
```

```
y3 = tc.getTuioObjects().get(2).getPosition().getY();
```

```
vx3 = tc.getTuioObjects().get(2).getXSpeed();
```

```
vy3 = tc.getTuioObjects().get(2).getYSpeed();
```

```
vth3 = tc.getTuioObjects().get(2).getRotationSpeed();
```

```
v3 = tc.getTuioObjects().get(2).getMotionSpeed();
```

case 3

```
x1 = tc.getTuioObjects().get(2).getPosition().getX();
```

```
y1 = tc.getTuioObjects().get(2).getPosition().getY();
```

case 4

```
x2 = tc.getTuioObjects().get(2).getPosition().getX();
```

```
y2 = tc.getTuioObjects().get(2).getPosition().getY();
```

otherwise

end;



```

    end;
end;
end;

pause(0.1);

end;

if(a3-a3ant>5)
    k=kant-1;
elseif(a3-a3ant<-5)
    k=kant+1;
else
    k =kant;
end

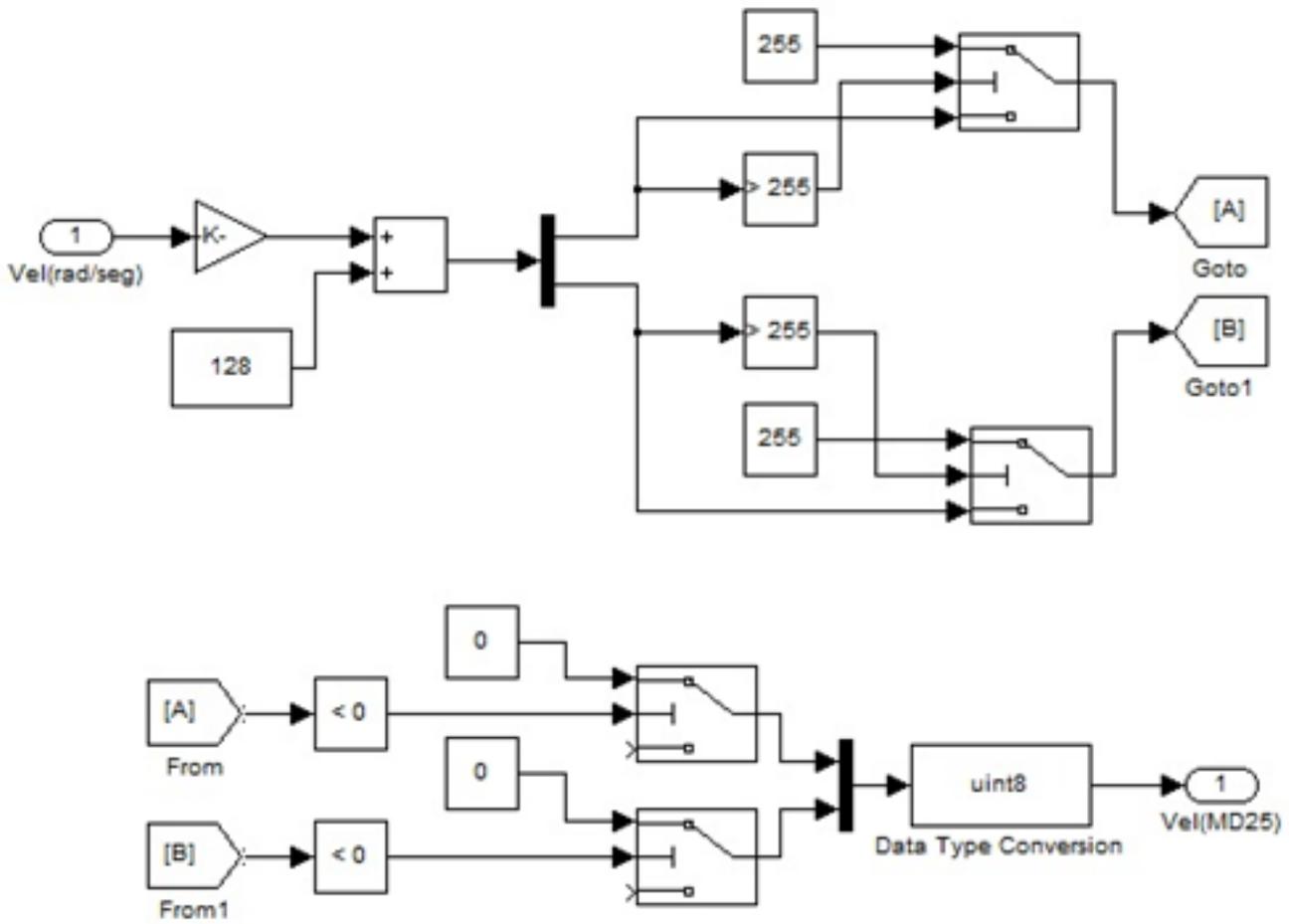
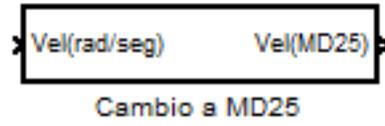
a3s=a3+k*2*pi;

tc.disconnect();

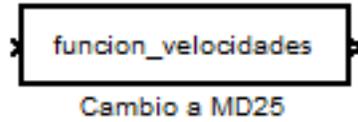
sys = [(x1*1.90-0.95) (y1*1.4-0.7) (x2*1.9-0.95) (y2*1.4-0.7) (x3*1.9-0.95) (y3*1.4-0.7) a3s vx3 vy3
vth3 v3 k a3];

```





Función 2



global a velizq velder k;

%%Entradas

velder=u(1);

velizq=u(2);

if(velizq==0||velder==0)

 k=10000000000000000000;

elseif(abs(velizq) > abs(velder))

 k=abs(velder/velizq);

else

 k=abs(velizq/velder);

end

%%En rango

if(velder>=-1.872629953 && velder<=1.872629953 && velizq>=-1.872629953 && velizq<=1.872629953)

 a=[velder*6.4081+128 velizq*6.4081+128];

%%velizq en rango, velder sale por arriba

elseif(velder>1.872629953 && velizq>=-1.872629953 && velizq<=1.872629953)

 a=[(1.872629953)*6.4081+128 (velizq*k)*6.4081+128];

%%velizq en rango, velder sale por abajo



```
elseif(velder<-1.872629953 && velizq>=-1.872629953 && velizq<=1.872629953)
```

```
    a=[(-1.87269953)*6.4081+128 (velizq*k)*6.4081+128];
```

```
%%velder en rango, velizq sale por arriba
```

```
elseif(velder>=-1.872629953 && velder<=1.872629953 && velizq>1.872629953)
```

```
    a=[(velder*k)*6.4081+128 (1.872629953)*6.4081+128];
```

```
%%velder en rango, velizq sale por abajo
```

```
elseif(velder>=-1.872629953 && velder<=1.872629953 && velizq<-1.872629953)
```

```
    a=[(velder*k)*6.4081+128 (-1.872629953)*6.4081+128];
```

```
%%velder sale por arriba, velizq sale por arriba
```

```
elseif(velder>1.872629953 && velizq>1.872629953)
```

```
    if(velder>=velizq)
```

```
        a=[(1.872629953)*6.4081+128 (velizq*k)*6.4081+128];
```

```
    else
```

```
        a=[(velder*k)*6.4081+128 (1.872629953)*6.4081+128];
```

```
    end
```

```
%%velder sale por abajo, velizq sale por abajo
```

```
elseif(velder<-1.872629953 && velizq<-1.872629953)
```

```
    if(velder>=velizq)
```

```
        a=[(-1.872629953)*6.4081+128 (velizq*k)*6.4081+128];
```

```
    else
```

```
        a=[(velder*k)*6.4081+128 (-1.872629953)*6.4081+128];
```

```
    end
```



```
%%velder sale por arriba, velizq sale por abajo
```

```
elseif(velder>1.872629953 && velizq<-1.872629953)
    if(abs(velder)>=abs(velizq))
        a=[(1.872629953)*6.4081+128 (velizq*k)*6.4081+128];
    else
        a=[(velder*k)*6.4081+128 (-1.872629953)*6.4081+128];
    end
end
```

%%velder sale por abajo, velizq sale por arriba

```
elseif(velder<-1.872629953 && velizq>1.872629953)
    if(abs(velder)>=abs(velizq))
        a=[(-1.872629953)*6.4081+128 (velizq*k)*6.4081+128];
    else
        a=[(velder*k)*6.4081+128 (1.872629953)*6.4081+128];
    end
end
end
```

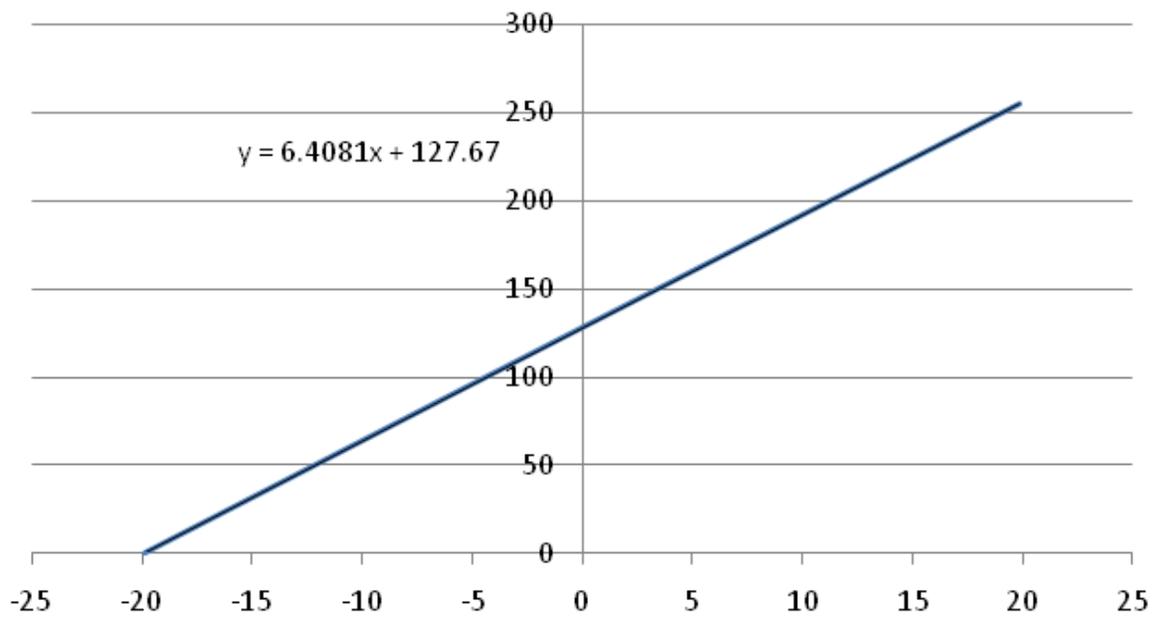
```
sys = [a(1) a(2)];
```

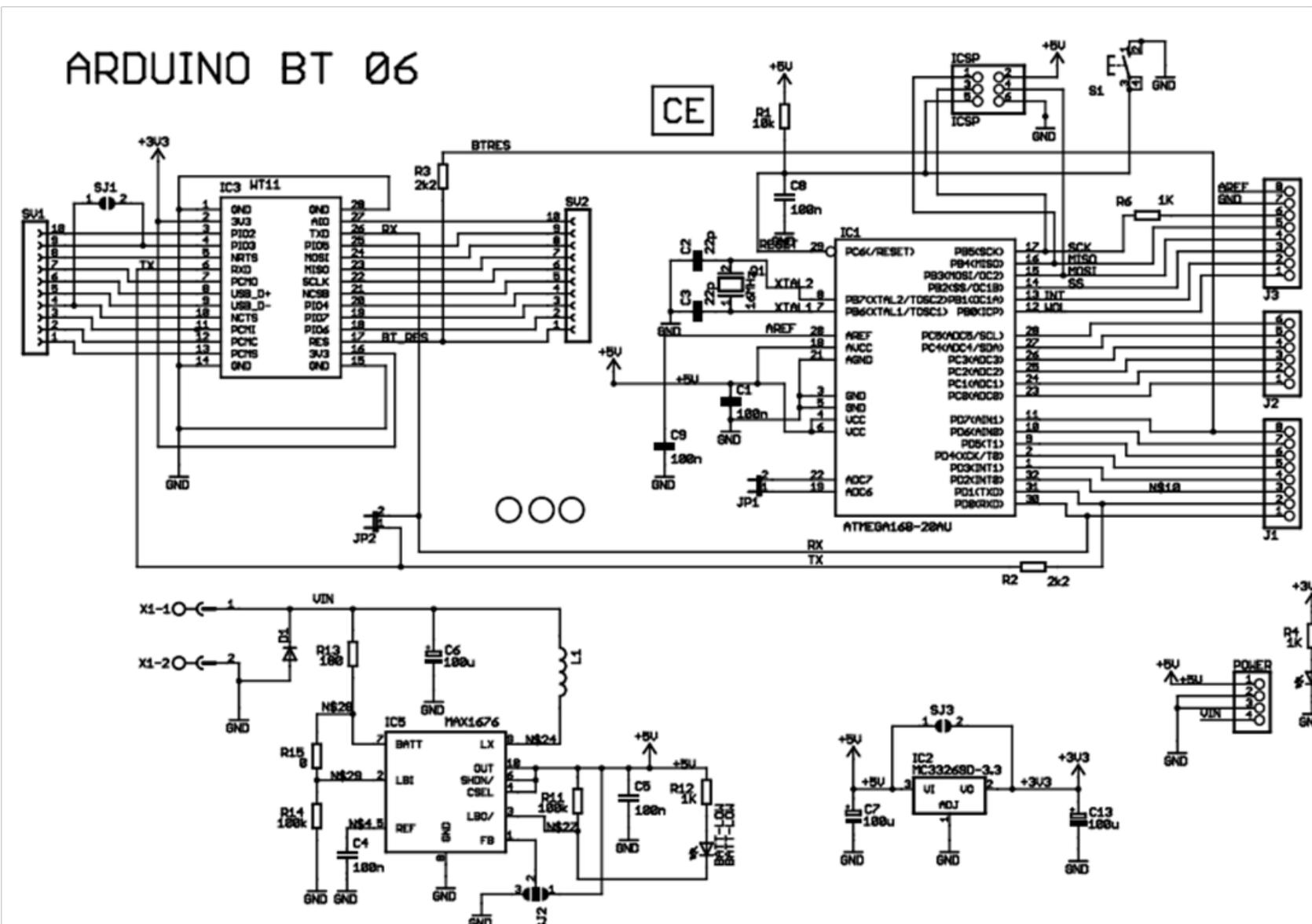


Apéndice 4

Correspondencia de velocidad real y comando MD25

| RPM | Velocidad tarjeta | Rad/seg | Velocidad tarjeta |
|------|-------------------|--------------|-------------------|
| 190 | 255 | 19.89675347 | 255 |
| 110 | 200 | 0 | 128 |
| 50 | 160 | -19.89675347 | 0 |
| -190 | 0 | | |





DESCRIPTION

WT11 is a next-generation, class 1, Bluetooth® 2.0+EDR (Enhanced Data Rates) module. It introduces three times faster data rates compared to existing Bluetooth® 1.2 modules even with lower power consumption! WT11 is a highly integrated and sophisticated Bluetooth® module, containing all the necessary elements from Bluetooth® radio to antenna and a fully implemented protocol stack. Therefore WT11 provides an ideal solution for developers who want to integrate Bluetooth® wireless technology into their design with limited knowledge of Bluetooth® and RF technologies. By default WT11 module is equipped with powerful and easy-to-use iWRAP firmware. iWRAP enables users to access Bluetooth® functionality with simple ASCII commands delivered to the module over serial interface - it's just like a Bluetooth® modem.

FEATURES:

- Fully Qualified Bluetooth system v2.0 + EDR, CE and FCC
- Class 1, range up to 300 meters
- Integrated chip antenna or UFL connector
- Industrial temperature range from -40 °C to +85 °C
- Enhanced Data Rate (EDR) compliant with v2.0.E.2 of specification for both 2Mbps and 3Mbps modulation modes
- RoHS Compliant
- Full Speed Bluetooth Operation with Full Piconet
- Scatternet Support
- USB interface (USB 2.0 compatible)
- UART with bypass mode
- Support for 802.11 Coexistence
- 8Mbits of Flash Memory



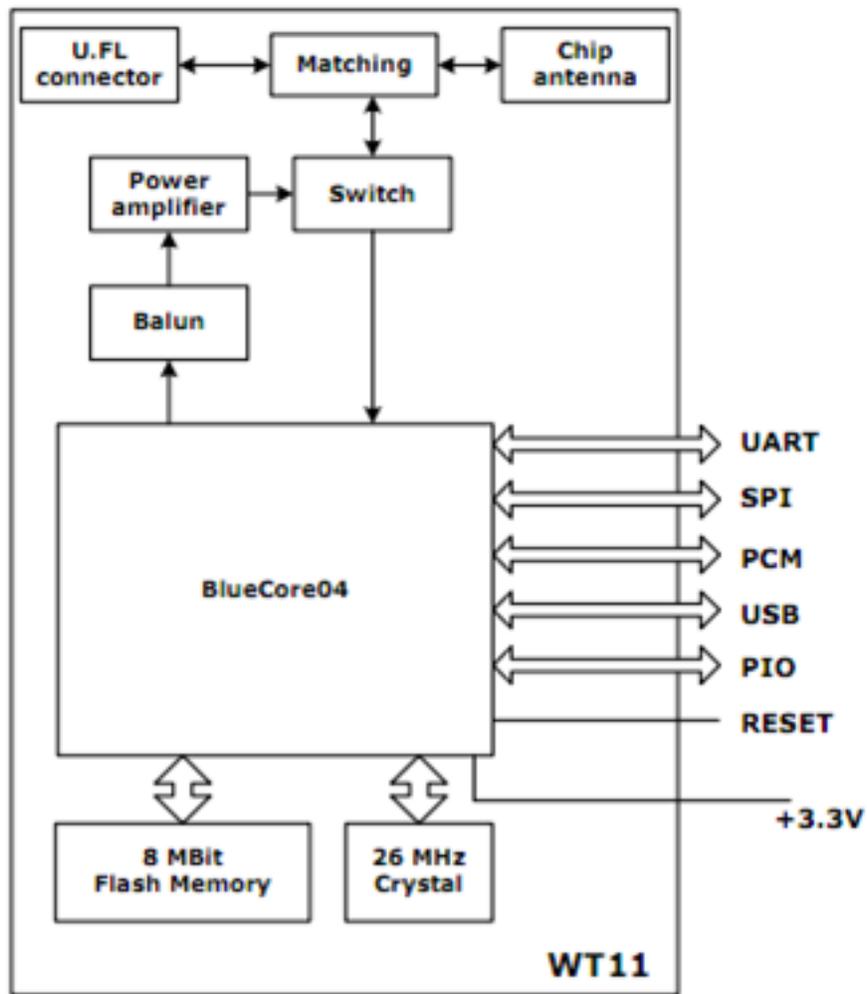


Diagrama de bloques WT11

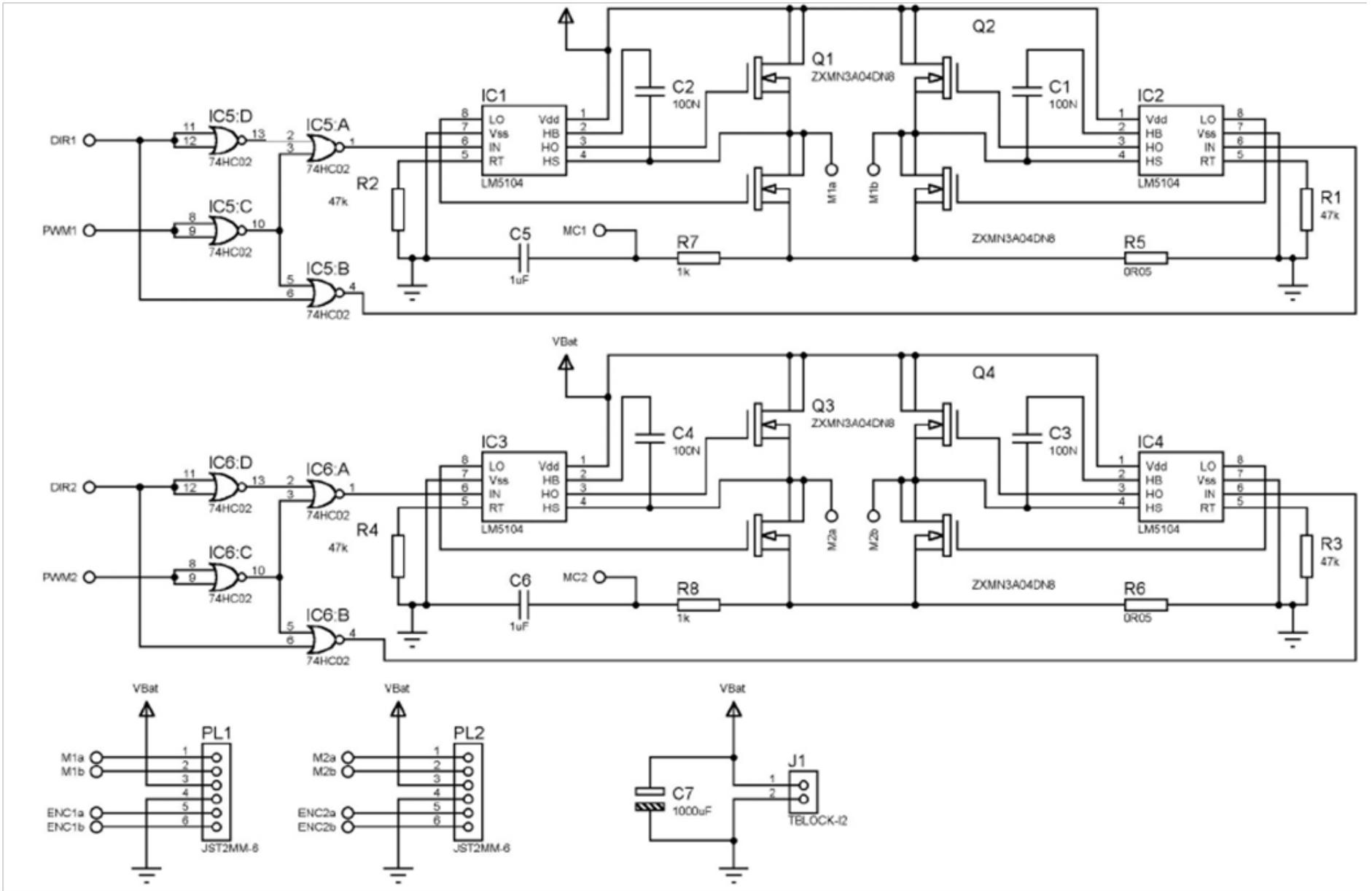


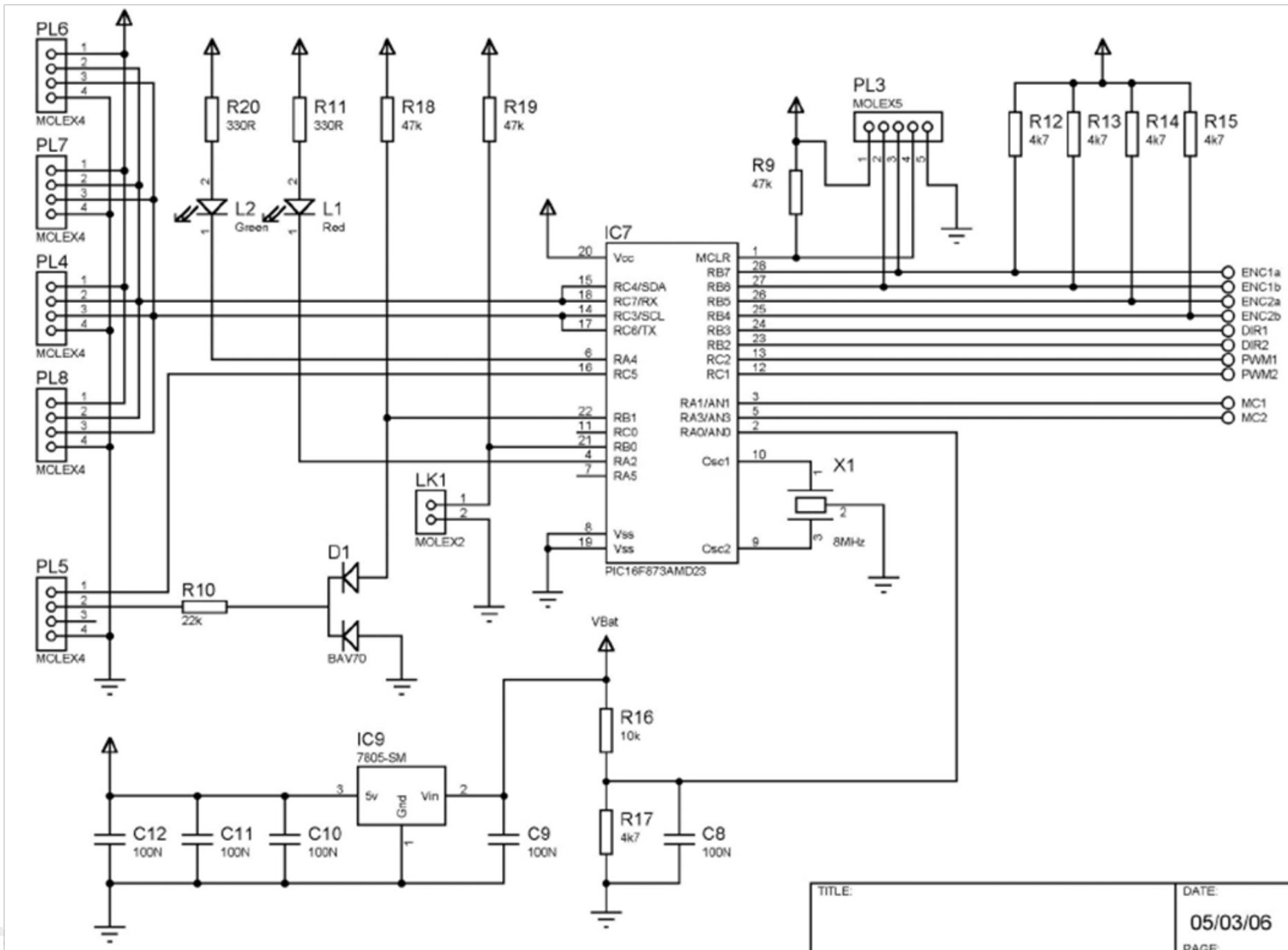
Apéndice 6

Tarjeta Drivers MD25

| Reg.Nº | Nombre | Tipo | Descripción |
|--------|--------------------|-------------------|---|
| 0 | Velocidad 1 | Lectura/Escritura | Ajusta la velocidad del motor 1 en los modos 0 y 1 o la velocidad de ambos en los modos 2 y 3 |
| 1 | Velocidad 2 / Giro | Lectura/Escritura | Ajusta la velocidad del motor 2 en los modos 0 y 1 o el giro en los modos 2 y 3 |
| 2 | Codificador 1 a | Sólo lectura | Posición del codificador 1. 1er. byte (mas peso) |
| 3 | Codificador 1 b | Sólo lectura | Posición del codificador 1, 2º byte |
| 4 | Codificador 1 c | Sólo lectura | Posición del codificador 1, 3er. Byte |
| 5 | Codificador 1 d | Sólo lectura | Posición del codificador 1, 4º byte (menos peso) |
| 6 | Codificador 2 a | Sólo lectura | Posición del codificador 2. 1er. byte (mas peso) |
| 7 | Codificador 2 b | Sólo lectura | Posición del codificador 2, 2º byte |
| 8 | Codificador 2 c | Sólo lectura | Posición del codificador 2, 3er. Byte |
| 9 | Codificador 2 d | Sólo lectura | Posición del codificador 2, 4º byte (menos peso) |
| 10 | Batería | Sólo lectura | Refleja la tensión actual de la batería de alimentación |
| 11 | Consumo Motor 1 | Sólo lectura | Refleja el consumo actual del motor 1 |
| 12 | Consumo Motor 2 | Sólo lectura | Refleja el consumo actual del motor 2 |
| 13 | Versión | Sólo lectura | Indica la versión del firmware interno |
| 14 | Aceleración | Lectura/Escritura | Registro opcional para la aceleración |
| 15 | Modo | Lectura/Escritura | Establece los modos de trabajo del MD25 |
| 16 | Comando | Lectura/Escritura | Se introduce el comando a ejecutar por el MD25 |







TITLE: _____

DATE: 05/03/06
PAGE: _____

Referencias

- [1] Portal de Ingeniería Mecatrónica de la Universidad Nacional Autónoma de México (16/Junio/2011)
<http://mecatronica.unam.mx/mecatronica.html>
- [2] Portal de la Real Academia de la Lengua Española. Definición de Robótica (16/Junio/2011)
http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=rob%F3tica
- [3] Lahoz-Beltrá R. (2004). Bioinformática: simulación, vida artificial e inteligencia artificial. Madrid: Editorial Díaz de Santos.
- [4] Lavelle S. M. (2006). Planning Algorithms. Cambridge University Press.
- [5] Gonzalez-Villela V. J. (2006). Research on a semiautonomous mobile robot for loosely structures environments focused on transporting mail trolleys. Loughborough: Loughborough University PhD.
- [6] Ollero Baturone A. (2001). Robótica: manipuladores y robots móviles. Barcelona: Editorial Marcombo
- [7] Portal de la Real Academia de la Lengua Española. Definición de Robot (17/Junio/2011)
http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=robot
- [8] Díaz Hernández O. (2010). Experimentación con una configuración de robot móvil de llantas frontales independientes en su direccionabilidad. Tesis de Maestría en Ingeniería. Universidad Nacional Autónoma de México.
- [9] Olivier, Lefebvre, Lamiroux, Florent. Pradalier, Cédric. Fraichard, Thierry. Obstacles Avoidance for Car-Like Robots. Integration and Experimentation on Two Robots. Francia.
- [10] Lamiroux, Florent. Bonnafous, David. Lefebvre, Olivier. (2004). Reactive Path Deformation for Nonholonomic Mobile Robots. IEEE Transactions on Robotics. Vol. 20, No. 6.
- [11] Kurniawati, Hanna. Fraichard, Thierry. From Path to Trajectory Deformation. (2007). INRA. Francia.
- [12] Fraichard, Thierry. Delsart, Vivien. (2009). Navigating Dynamic Environments with Trajectory Deformation. INRA, LIG-CNRS and Grenoble University (Francia).
- [13] Lamiroux, Florent. Lefebvre, Olivier. (2010). Sensor-based trajectory deformation: application to reactive navigation of nonholonomic robots. "Visual Servoing via Advanced Numerical Methods" Chesi, Graziano; Hashimoto, Koichi. (Ed.)(2010). P. 315-334.
- [14] Portal de la Real Academia de la Lengua Española. Definición Visión (1/Julio/2011)
http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=visi%F3n
- [15] Página dedicada a comunicación Bluetooth (27/Julio/2011)
<http://www.bluetooth.com/Pages/Basics.aspx>
- [16] Página oficial de ReactIVision (29/Julio/2011)
<http://reactivision.sourceforge.net/>

