Proyecto Final

1st Juan Carlos Garduño Gutiérrez 157302
Ingeniería en Telecomunicaciones
Instituto Tecnológico Autónomo de México
jgarduo6@itam.mx

2nd Humberto Martínez Barrón y Robles 166056
Ingeniería en Mecatrónica
Instituto Tecnológico Autónomo de México
hmartin3@itam.mx

3rd Sebastián Aranda Bassegoda 157465
Ingeniería en Computación e Ingeniería Industrial
Instituto Tecnológico Autónomo de México
sarandab@itam.mx

***Resumen*—Se reporta el trabajo realizado en este proyecto que consiste en implementar un sistema mecatrónico, un robot diferencial con dos actuadores, uno por rueda. Tenía que ser capaz de realizar un trayecto desde una posición inicial a una final preestablecida. Para alcanzar este objetivo se combinó** $Arduino$**,** ROS**, y** $XBee$**.**

***Index Terms*—**$Arduino$**,** ROS**,** $XBee$**,** PID

I. Introducción

El objetivo de este proyecto fue implementar un sistema mecatrónico. Este robot tenía que ser capaz de realizar un trayecto desde un punto inicial a uno final, con y sin obstáculos. La figura 1 muestra el planteamiento del problema.

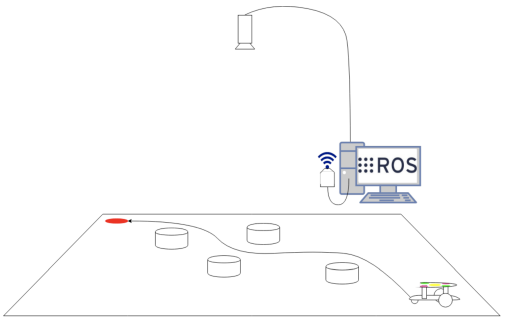

Figura 1. Diagrama del experimento

Un sensor, una cámara en el techo, era la que nos iba a permitir ubicar a nuestro robot en relación al punto final y los obstáculos. Esta información era necesario interpretarla con la ayuda de ROS para poder realizar el recorrido.

El proyecto requiere implementar diferentes herramientas que hemos visto a lo largo del curso como ROS, $Arduino$, etc. Algunas nos permitirán mover al robot, como los motores, PID, puente H, etc. Otras nos permitan comunicarnos con él, como los $XBees$. Es la combinación de todos estos elementos lo que hace posible enfrentar el problema.

II. Marco teórico

En esta sección se describen los dispositivos utilizados, otras tecnologías y conceptos aplicados. Es decir, muestra los fundamentos teóricos y matemáticos del porqué las cosas funcionan, así como las suposiciones y las restricciones.

Los robots son máquinas automáticas programables. Estas máquinas son capaces de realizar determinadas operaciones de manera autónoma: manipulan objetos, se mueven, y realizan trabajos al interactuar con su entorno. Lo que le permite a un robot el poder interactuar con su entorno son los sensores y actuadores con los que cuenta.

Un sensor es un dispositivo capaz de captar magnitudes físicas y demás alteraciones en su entorno. Un actuador por otra parte, es un dispositivo mecánico que tiene como función proporcionar fuerza para mover o hacer actuar a otro dispositivo mecánico. En el robot de nuestro proyecto, el sensor principal es una cámara encargada de recibir la posición del robot mientras que los actuadores son dos motores eléctricos que mueven las llantas.

Se utiliza ROS *(Robot Operating System)*, que es una herramienta que proporciona bibliotecas para ayudar a los desarrolladores de software a crear aplicaciones de robótica. Proporciona abstracción de $hardware$, controladores de dispositivo, bibliotecas, visualizadores, paso de mensajes, administración de paquetes y más. ROS es completamente de código abierto (BSD) y es gratuito para que otros lo usen, lo modifiquen y lo comercialicen. [1]

Como interfaz se utilizan $Xbees$. De acuerdo a $Digi$(fabricante), los módulos XBee son soluciones integradas que brindan un medio inalámbrico para la interconexión y comunicación entre dispositivos. Estos módulos utilizan el protocolo de red llamado IEEE 802.15.4 para crear redes FAST POINT-TO-MULTIPOINT (punto a multipunto); o para redes PEER-TO-PEER (punto a punto). Fueron diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible. Por lo que básicamente XBee es propiedad de Digi basado en el protocolo Zigbee. En términos simples, los XBee son módulos inalámbricos fáciles de usar.

III. Desarrollo

III-A. Armado del Robot

Se empezó por armar al robot para el cual utilizamos las siguientes piezas.

Cantidad	Pieza
1	Base
2	Llanta
2	Conexión de Llanta
2	Hub para Motor DC
2	Motor DC
1	Arduino Mega 2560
1	Arduino wireless proto shield
1	XBee S1
2	Opto Interrupter ITR8102
1	Encoder de 18 orificios
1	Protoboard
1	Batería LIPO

En la figura 2 podemos observar el robot completmante armado y cableado.

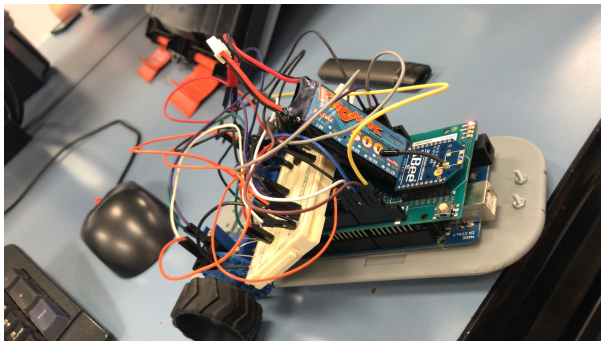

Figura 2. Robot armado.

III-B. Conexión del Puente H

Tras armar el robot fue necesario realizar la conexión del puente H. Esto es lo que le permite al Arduino poder controlar los motores. Se utilizó el diagrama de la figura 3 en el cableado del puente H, que por excepción de los pines en el arduino y el uso del circuito integrado equivalente, SN75441ONE, fue idéntico.

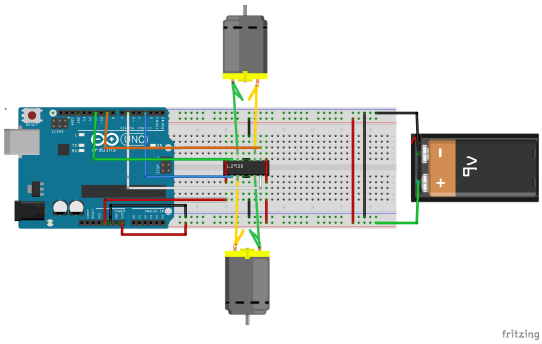

Figura 3. Diagrama mostrando las conexiones del puente H.

III-C. Configuración de los optointerruptores

El siguiente paso del proyecto fue conectar los optointerruptores ilustrados en la figura 4. La salida de estos fue alambrada hasta las entradas analógicas del Arduino. Es gracias a los

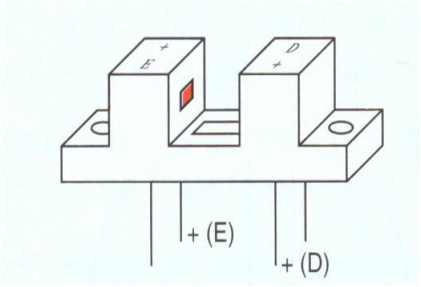

Figura 4. Diagrama de un optointerruptor.

optointerruptores que es posible medir la velocidad de las llantas y que posteriormente el controlador PID la pueda estabilizar.

III-D. Controlador PID

Con la intención de que convergiera el movimiento de los motores, se tuvo que implementar un PID. Es gracias a este que la trayectoria del robot podía controlarse.

Para lograr esto, se implementó un programa en Arduino que se suscribía a un nodo de ROS, llamado `escucha_pose_node` en el código. De ahí, se leía una velocidad angular deseada para cada motor. Posteriormente, se tomaron las siguientes medidas (nótese que se hizo lo mismo para dos motores, por lo que fue necesario implementar dos controladores PID prácticamente idénticos):

1. Se configuraron los pines necesarios para leer el estado de los fototransistores colocados sobre los $encoders$ de las ruedas.
2. Se indicó que cada vez que había un cambio en el estado del fototransistor una variable de cuenta debía incrementar en 1.
3. Cada veinte milisegundos, se programó un paso de control, en donde se medía la cuenta que se llevaba hasta el momento, c y se efectuaba la siguiente operación:
$$v_i = \frac{c}{36} \cdot \frac{1000}{20} \cdot 2\pi$$
donde v_i es la velocidad del motor en radianes por segundo i, con $i = 1, 2$.
4. Se toma v_i y la velocidad deseada (también en radianes por segundo) leída del nodo de ROS y a ésta se le resta aquélla para obtener un error.
5. Pasamos este error por el control, integrándolo a la suma de errores y derivándolo (restándole el error anterior). Con esto, obtenemos un control PID.

III-E. Configuración de los $XBees$

Para lograr que funcionen los $XBee$, primeramente fue necesario hacer las configuraciones necesarias en el programa $XCTU$, en donde se especifica el baudaje y las direcciones propias y de destino de cada dispositivo. Después de llevar a cabo este paso, se tuvo que configurar cada $XBee$ para trabajar con ROS. Para esta tarea existe un paquete de ROS llamado `rosserial_xbee`, dentro del repositorio

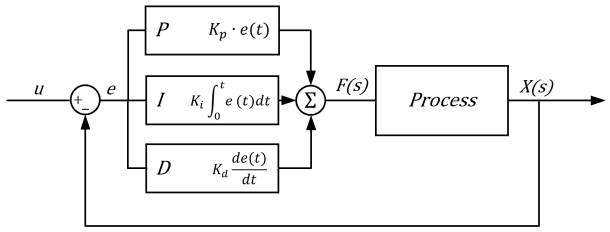

Figura 5. Implementación del PID.

`rosserial` que lleva a cabo la configuración de un $XBee$ para ROS como configurador con el comando:

```
rosrun rosserial_xbee setup_xbee.py -C
/dev/ttyUSBn 0
```

donde n es el número del puerto en donde está conectado el $XBee$. Para configurar un $Xbee$ que no es configurador, se corre:

```
rosrun rosserial_xbee setup_xbee.py
/dev/ttyUSBn 1
```

donde, una vez más, n es el número de puerto en el que está conectado el $XBee$.

Una vez configurados los $XBee$, es necesario que ROS establezca la red en la que se van a comunicar. Para esto, el comando es el siguiente:

```
rosrun rosserial_xbee xbee_network.py
/dev/ttyUSBn 1
```

Para correr este comando, `roscore` debe estar corriendo en alguna terminal. También, un $XBee$ (el configurador) debe estar conectado a la computadora y el otro al robot (en este caso, al $shield$ que se monta sobre el $Arduino$). Posteriormente, se pueden observar mensajes indicando a qué tópicos está suscrito y cuáles está publicando.

III-F. Navegación

Para la navegación, se resolvieron dos problemas distintos: navegación con obstáculos y navegación sin obstáculos. En ambos casos, todo el código para resolver estos problemas se escribió en ROS.

III-F1. Sin obstáculos: El caso más simple de la navegación simplemente requirió de un control de la posición del robot. Para lograr que se moviera adecuadamente, se requirió elaborar un nodo que se suscriba al tópico que contiene la posición del objetivo (llamado `/ball`) y al que contiene la posición del robot (llamado `/y_r0`). Una vez que se leen estas posiciones, se necesita calcular una diferencia en el eje x y en el eje y (pues se trata de navegación bidimensional solamente). Con esta diferencia, se pueden obtener dos variables de mucho interés: la distancia entre el robot y su objetivo y la diferencia en ángulos entre el ángulo deseado y el ángulo del robot.

Con la distancia entre el robot y su objetivo, se calcula una velocidad con un control proporcional, multiplicándola por una constante (en este caso, $\frac{1}{10}$). Esto se puede implementar así porque la distancia es un error que se quiere llevar a cero. Esto se puede lograr con un control. De manera analógica, se resolvió la diferencia de ángulos multiplicando la diferencia entre el ángulo de orientación del robot y el ángulo al que se encuentra el objetivo (calculado con la función `atan2` para que se respete el cuadrante en el que se encuentra) para implementar un control proporcional (esta vez, se usó una constante de $\frac{1}{5}$). Estos dos datos se pasan por una matriz que, según la especificación del proyecto, obtiene las velocidades angulares de los motores dados estos dos parámetros.

$$\begin{bmatrix}\phi_l \\ \phi_r\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & -\frac{L}{2} \\ 1 & \frac{L}{2}\end{bmatrix}\begin{bmatrix}v \\ \omega\end{bmatrix}$$

Midiendo el robot, se encontró que $r = 21$ mm y $L = 115$ mm. Estos datos fueron consistentes con las especificaciones de la práctica. Publicando estos datos, tenemos nuestra trayectoria.

III-F2. Con obstáculos: Para navegar con obstáculos, se agregó un $script$ de $Python$ que publica punto por punto de una trayectoria dadas las posiciones de los obstáculos.

Lo primero que se debe hacer es leer las posiciones del robot, del objetivo final y de cada obstáculo. Una vez logrado esto, se utiliza la biblioteca $Numpy$ para generar puntos aleatorios entre el robot y el objetivo (si hay pocos obstáculos, y se usa todo el mapa para generar los puntos si hay muchos objetivos). Posteriormente, se intenta conectar a cada punto a todos los puntos en una vecindad de 500 mm para saber si se pueden considerar como nodos conectados. Si es así, se va generando un grafo con nodos: cada nodo tiene su identificador único (índice que comienza en 1), su posición, la lista de identificadores de nodos a los que está conectado, el costo acumulado hasta ese instante (inicializado en cero), la distancia del nodo al objetivo y el identificador del nodo anterior en el camino (inicializado en cero).

Después de generar estos nodos, se implementa el algoritmo A* para encontrar la ruta óptima. Una vez que se obtiene este camino, se van mandando los objetivos uno por uno al robot (publicados con el nombre de tópico `/objetivo`). En la figura 7 se puede ver un ejemplo del resultado de la navegación simulada para 100 obstáculos cuadrados de longitud 20 mm.

IV. Resultados

Los resultados no son conclusiones, si fallaste, describe por que fallaste y que se puede hacer para solucionar el error. Describe los valores o las configuraciones utilizadas así como los resultados de cada una de ellas.

Al intentar implementar el proyecto, la configuración de los $Xbee$, que había funcionado casi a la perfección en el laboratorio, no se pudo lograr en la presentación. Se intentó volver a configurar cada $XBee$ incontables veces, pero no se tuvo éxito al final. Este resultado fue extremadamente frustante, pues ya había sucedido antes que los $Xbee$ no se pudieran conectar, pero siempre se había podido arreglar con una reconfiguración de cada dispositivo. Sin embargo, en esta ocasión el error que arrojaba el $XBee$ era consistente y era que ROS no era capaz de sincronizar ambos $XBee$. Esto se pudo deber a una configuración errónea que no había notado el equipo hasta ese momento. Para aún lograr presentar, se

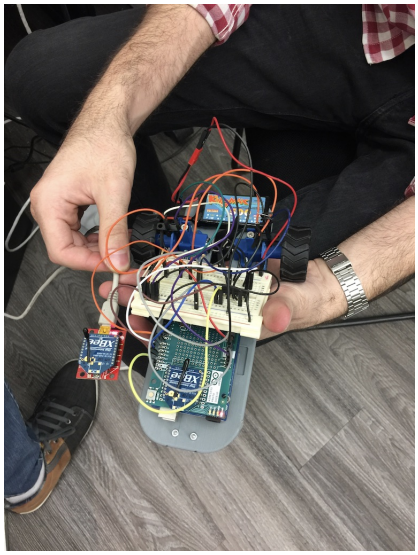

Figura 6. Xbees. Uno conectado a la computadora y otro en el robot.

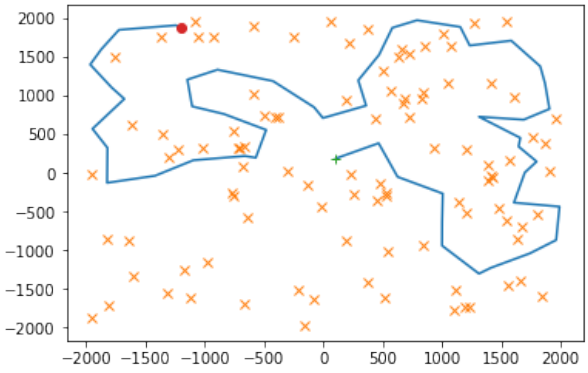

Figura 7. Ejemplo de ruta de navegación con obstáculos (los obstáculos están representados por las 'x' mientras que el objetivo está representado como un círculo grande rojo y el robot como una cruz azul)

decidió presentar el proyecto con el robot conectado a una computadora para poder correr el código implementado.

Después de muhas fallas, se logró un resultado postivo, tanto en la neavegación sin obstáculos como en la navagación sin obtaáculos. La estretegia fue primero solucionar el problema sin obstáculos y luego con ellos.

El tiempo de recorrido en la neavegación sin obstáculos fue de 12 segundos y el tiempo en el recorrido sin obstáculos fue de de 8 segundos.

Para la navegación con onstáculos se logró que el robot implementado identificara una forma satisfactoria para pasar en medio de ellos. Aunque todo funcionó como se esperaba, el robot puede ser mejorado bastante; por ejemplo, se pueden soldar los cables para evitar cualquier posible desconexión, se puede reacomadar de manera efeciente todo lo que la *chancla* contenía; batería, *protoboard*, tarjeta *Arduino* y cables, se puede mejorar la velocidad de los motores para que ambos trayectos se hagn en menor tiempo, etc. Las mejoras dependen de donde vayamos a usar el carrito.

Los valores y configuraciones utilizadas en cada sección del proyecto, así como los resultados de cada una de ellas se encuentran explicadas detalladamente en la seccion de *Desarrollo*.

V. Conclusiones

V-A. *Humberto Martínez*

Este proyecto fue demandante porque integró conocimientos de electrónica, computación, mecánica y control. Estas cuatro disciplinas componen la mecatrónica según el instituto Politécnico Renssealer. El aprendizaje obtenido en este proyecto podrá servir en el futuro como experiencia para crear más sistemas mecatrónicos y estar más en contacto con la robótica en particular.

Por otro lado, una habilidad no relacionada con el conocimiento técnico y teórico que se necesitó en este proyecto fue la perseverancia y la paciencia. No fue poco común encontrarse con discrepancias, complicaciones y errores en general, sobre todo al implementar el código en *hardware*. Como gran parte de estas tecnologías siguen siendo poco familiares para el equipo, aprender a integrarlas todas en una sola aplicación tomó mucho tiempo y, de hecho, la comunicación por *XBee* tampoco se logró implementar en la presentación final, a pesar de que todo funcionaba perfectamente en las pruebas.

Al respecto de hacer pruebas, el último aprendizaje que vale la pena resaltar de este proyecto fue que las pruebas son poco confiables para predecir el comportamiento de una aplicación si no se hacen exactamente en el ambiente a la que se va a enfrentar la solución diseñada. Por lo tanto, algo que se pudo haber mejorado mucho fue la forma en la que se probó el robot pues, seguramente, de haber tenido más tiempo (o quizá de haber tenido el conocimiento suficiente acerca de este tipo de problemas) se hubiera podido dar más importancia a probar el dispositivo en donde verdaderamente tenía que funcionar, en vez de suponer que tenía que funcionar simplemente porque funcionaba correctamente en el laboratorio.

V-B. *Juan Carlos Garduño*

El proyecto resultó ser más complejo de lo que pensé. Leí el papel donde estaba explicado el proyecto, y en plabras cotidianas, sonaba relativamente fácil. Conforme fuimos avanzando me dí cuenta que no iba a ser así y que ibamos a tener que poner nuestro mejor esfuerzo y mucho tiempo.

Durante el proceso de implementación del robot, me dí cuenta que los problemas que iban surgiendo eran un réplica de algunos problemas resueltos en prácticas pasadas, sin embargo, mucho más complejos. Entonces, en más de una ocasión, nos tuvimos que regresar a las prácticas pasadas para retomar esas bases y poderlas aplicr ahora a este proyecyo. También me dí cuenta de que este proyecto es la integración de muchos temas vistos en las prácticas pasadas.

Nos pasó que el carrito funcionaba perfectamente en el laboratorio, pero a la hora de hacer las prubas de verdad, había algo que hacía que fallara. Entonces, la lección que me llevo de esto es que en realidad no sabemos que agentes externos puedan interferir y hay que preveer la mayor cantidad

posible. Un ejemplo de esto, fue que los $XBee$ al final no salieron y estoy seguro que no fue el código implementado, sino la interferencia entre los demás que había en el cuarto de pruebas o quizás la computadora donde estabamos corriendo el programa le faltaba un paquete o algo por el estilo.

Me siento satisfecho con los resultados del proyecto, desde mi punto de vista se logró el objetivo principal: aprender cosas nuevas, aplicar lo visto en teoría y ganar experencia en cuanto a la superación de problemas y la implementación de un proyecto que al inicio se veía muy retador.

V-C. Sebastián Aranda

Lo que hace esta práctica tan valiosa es que requiere juntar todo el conociemiento y utilizar varias herramientas que hemos visto a lo largo de todo el curso. Combinando lo que hemos visto en teoría y lo que se ha implementado en el laboratorio. Es por esta razón que es tan completo este proyecto.

El proyecto nos demandó utilizar varios lenguajes diferentes y requiere armar el robot. Cada equipo acababa con algo diferente al ser conformado por tantas partes y ser tan complejo. Es la complejidad del proyecto lo que lo llevo a ser tan frustrante y por lo que hubo varios momentos a lo largo de la práctica que como equipo no creímos que fuéramos capaces de acabar a tiempo. Finalmente sí fue posible, con sus deficiencias, pero de eso no enorgullecemos los tres.

Es una bonita conlusión esta práctica al curso. Como mencioné antes combinó todas las herramientas que habíamos adquirido a lo largo del semestre y es increíble llegar a algo tangible.

VI. Rol o Papel

VI-A. Humberto Martínez

En la práctica, Humberto tomó posesión del teclado para realizar el código necesario en la práctica.

En las demás secciones, se encargó del código, mientras sus compañeros colaboraban con dudas de código que iban surgiendo.

En cuanto al reporte Humberto elaboró o colaboró las siguientes secciones:

- Introducción.
- Desarrollo.
- Sus respectivas conclusiones.

VI-B. Juan Carlos Garduño

En la práctica, Juan Carlos se encargó junto con Sebastián a ir alambrando los componentes necesarios. En momentos de la práctica, iba dictando algunas secciones del código a Humberto que había investigado y podían ser de utilidad para la compilación del programa. De igual manera, iba revisando que la sintaxis de Humberto en el código, fuera la correcta. En cuanto al reporte Juan Carlos elaboró las siguientes secciones:

- Marco teórico.
- Resumen.
- Sus respectivas conclusiones.

VI-C. Sebastián Aranda

En la práctica, Sebastián se encargo del alambrado junto con Juan Carlos, así como de ir colaborando con los demás integrantes en la elaboración del código. En cuanto al reporte él elaboro las secciones siguientes:

- Resultados.
- Rol o Papel.
- Sus respectivas conclusiones.

Referencias

[1] Willow Garage. (2011). ROS. abril 24,2019, de Willow Garage Sitio web: http://www.willowgarage.com/pages/software/ros-platform

[2] Cplusplus. (2019). C++. abril 24,2019, de cplusplus Sitio web: http://www.cplusplus.com/

[3] DIE. (2019). Linux man pages. abril 24,2019, de die.net Sitio web: https://linux.die.net/man/

[4] Gary. (2018). ROS Cheat Sheet. abril 24,2019, de Departamento Académico de Sistemas Digitales, ITAM Sitio web: https://github.com/garygra/ROS_cheat_sheet/blob/master/main.pdf

[5] ROS.Official Page (2019). abril 20,2019, de ROS Sitio web: http://www.ros.org/

[6] ROS Wiki. (2019). turtlesim. abril 24,2019, de ROS Sitio web: http://wiki.ros.org/turtlesim