Reporte del proyecto final: Robot
(Mayo 25, 2019)

Thomas, Manuel & Ruz, Pablo. *Principios de Mecatrónica, ITAM*

***Abstract*— Durante aproximadamente cuatro semanas, se buscó implementar un robot que llegase a un destino determinado por medio de una conexión con ROS, Xbee, Arduino, sensores y un PID.**

***Index Terms*—ROS, Arduino, PID, Xbee, C, sensores.**

I. Introducción

Habiendo realizado una serie de prácticas a lo largo del curso que involucraron, por separado, la implementación de ROS, protocolos de comunicación por medio de XBees, publicadores y controladores, y PID, entre otras cosas, el proyecto final tuvo como finalidad el conjuntar todos esos conocimientos y hacer una conexión estrecha entre ellos.

En concreto, se buscaba que el coche:

a. Alcanzara una velocidad constante por medio de un PID, dada una velocidad deseada.
b. Pudiese medir su propia velocidad lineal por medio de la velocidad angular proveniente de sensores ópticos de herradura.
c. Fuese capaz de establecer comunicación con un publicador de ROS, que pasaba las coordenadas del robot.
d. Adaptar su velocidad de acuerdo con la posición relativa tanto a obstáculos como a la meta.

Cada uno de los puntos enlistados en la parte superior se describirá a detalle en las secciones siguientes.

II. Marco teórico

El marco teórico que se presentará en esta sección se enfocará en los componentes nuevos que se implementaron para el proyecto, y una muy breve recapitulación de los marcos de las prácticas anteriores.

Velocidad angular

Antes de definir matemáticamente la velocidad angular, es importante recordar que la velocidad es un vector (y que por tanto tiene dirección y sentido), mientras que la rapidez es la magnitud de dicho vector. Para el caso concreto del automóvil, lo que más importaba era la rapidez, para luego poder traducirla a una lineal. Recuérdese, pues, que la rapidez (denotada aquí por v) está definida como:

$$v = \frac{d_2 - d_1}{t_2 - t_1}$$

Así pues, la siendo que la rapidez angular debe de seguir la misma lógica, es posible expresarla de la siguiente forma:

$$\omega = \frac{d\theta}{dt}$$

Donde:

- $\omega = velocidad\ angular$
- $\frac{d\theta}{dt} = deriv\ de\ posición\ angular\ con\ respecto\ al\ t.$

Evidentemente, dicha fórmula es para el caso continuo, pero puede ser discretizada siguiendo el mismo formato que el de la velocidad lineal. Habiendo presentado las dos fórmulas, y sabiendo que la velocidad angular está medida en rad/s, mientras que la lineal está medida en m/s, queda claro que para hacer una transformación de una a otra se debe de cumplir que:

$$v = rw \ldots (1)$$

Donde:

- $v = velocidad\ lineal$
- $r = radio$
- $w = velocidad\ angular$

Nótese que la conciliación de unidades es correcta, en tanto que el radián no es medida de una magnitud física, por lo que es posible eliminarla al momento de realizar la conciliación sin temor a perder el significado físico de la ecuación o violar las leyes matemáticas.

Para poder explicar de mejor manera las fórmulas siguientes, obsérvese la siguiente figura.

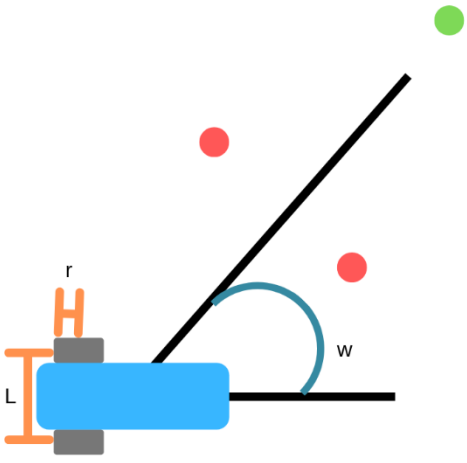

Fig.2
Obsérvense las distintas variables que estarán involucradas en el movimiento del carro.

Teniendo de manera más clara las relaciones existentes, es más sencillo explicar la fórmula que se muestra en los documentos de la práctica, misma que se reproduce aquí:

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

Fig.2
Fórmula matricial del cálculo de la velocidad lineal para cada una de las ruedas del coche.

Para proceder con la explicación de la fórmula presentada en la Fig.1, se procederá a expresar una de ambas fórmulas contenidas por las matrices:

$$\emptyset_l = \frac{1}{r} * (v - \frac{L}{2} w)$$

Donde:

- $\emptyset_l = velocidad\ angular\ de\ la\ rueda\ izquierda$
- $r = radio\ de\ la\ rueda$
- $w =$ ángulo formado entre el perpendicular al eje de las ruedas y el punto destino al que el robot tiene que ir.
- $L = distancia\ entre\ las\ ruedas$
- $v = velocidad\ lineal\ del\ automóvil$

Procedemos ahora con la explicación de la fórmula, para lo cual será especialmente útil la conciliación de unidades. Como se ha explicado antes, el radián no mide una magnitud física, por lo que es posible hacer uso del mismo para que las unidades de las ecuaciones sean correctas. Sabemos que como $\emptyset$ está medido en $\frac{rad}{s}$, entonces:

$$\begin{aligned} \frac{rad}{s} &= \frac{1}{\frac{m}{rad}} * \left(\frac{m}{s} - \frac{m}{rad} * rad\right) \\ &= \frac{rad}{m} * \left(\frac{m}{s} - m\right) \\ &= \frac{rad}{m} * \left(\frac{m}{s}\right) - \frac{rad}{m} * m \\ &= \frac{rad}{s} - dmnl \\ &= \frac{rad}{s} \end{aligned}$$

Como puede observarse, se ha usado el radián como "comodín" para poder llevar a cabo la conciliación de las unidades. Ahora bien, para poder traducir dicha velocidad angular a una lineal, simplemente basta operar con la fórmula (1).

Rosbags

En el sentido más claro, una bolsa o *bag* en ROS no es otra cosa que un formato de archivo en ROS que sirve para almacenar datos. Para poder acceder o manipular los datos existen diversos comandos, como los que se muestran a continuación:

- *rosbag record -a:* graba todos los tópicos.
- *rosbag info "bag-name":* despliega información sobre la bolsa previamente grabada.
- *rosbag play "bag-name":* pone a funcionar los datos guardados.

Sensores ópticos u optoelectrónicos

El principio básico de un sensor optoelectrónico es el efecto fotoeléctrico, que no es otra cosa que la liberación de cargas al momento en el que el sensor presenta una incidencia con la luz. Existen varios tipos de sensores optoelectrónicos, dentro de los cuales se encuentran los de:

- Fotoconducción: varía la conductividad (medida en $\frac{1}{\Omega m}$) dependiendo de la incidencia de luz.
- Efecto fotovoltaico: genera un voltaje con la incidencia de luz.
- Efecto fotoemisivo: emite electrones al incidir la luz.

El sensor que fue utilizado para el proyecto es el *itr8102*, que es del tipo fotoconductor. En efecto, el principio de los fototransistores es la generación de una corriente de base que es proporcional a la potencia óptica. Esto es, obedece a la siguiente fórmula:

$$i_c = \beta_t * i_{fp}$$

Donde:

- i_c = corriente total generada, medida en A.
- β_t = multiplicador adimensional
- i_{fp} = corriente que fluye a la terminal del fototransistor, medida en A.

Ahora bien, como en el caso de cualquier sensor, es importante ver la hoja de datos para asegurarse de que se está trabajando en el rango lineal del sensor porque, de lo contrario, se presentarían comportamientos erráticos. En la siguiente figura se presentan partes de las hojas de datos del *itr82.*

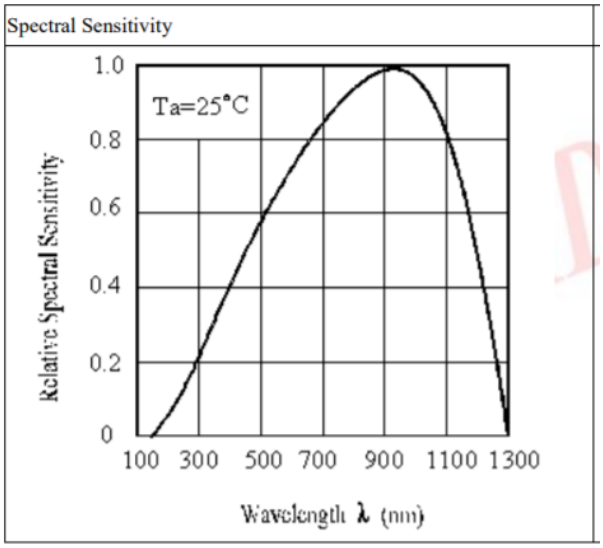

Fig.3
Nótese que para una temperatura ambiente de 25°C, el comportamiento de sensibilidad es bastante lineal para valores de longitud de onda de entre 100 y 900 nm. Esto quiere decir que para la luz visible, cuyo valor máximo de longitud de onda es de 550 nm, funciona perfectamente.

Recapitulación de conceptos pasados:

ROS

ROS (Robot Operating System) es un meta sistema operativo que es usado para controlar robots y que tiene estructura modular, lo que permite controlar muchos motores del robot de manera simultánea. Algunos de los principales componentes de ROS son los mensajes, nodos y tópicos.

Un nodo es una especie de módulo que conjunta una serie de funciones que le permiten realiza funciones dentro de sí mismo, pero también comunicarse y colaborar con otros nodos.

Fig. 4
Estructura básica de un nodo. Obsérvense los diversos componentes que lo forman: publicaciones, suscripciones, etc.

Como su nombre lo indica, un "topic" es el nombre que se le otorga a una serie de procesos que permiten identificarlo y, por ende, que exista comunicación entre los nodos. Es decir, cuando un nodo realiza una función determinada, puede publicarlo a un topic y, a su vez, un segundo nodo puede estar suscrito a este topic para así poder recibir el mensaje tras la publicación por parte del nodo uno.

Por su parte, un mensaje es el paquete de datos que un nodo publicador envía a través de un tópico para que el suscriptor pueda leerlo.

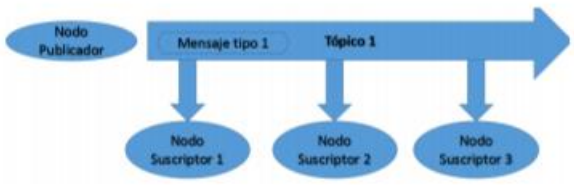

Fig. 5
Véase que el nodo publicador manda el mensaje a través del tópico uno, de modo que los nodos suscriptores pueden tener acceso a él y, por ende, al mensaje.

PID

Recuérdese que un controlador PID tiene como función principal el poder acercar el valor de una variable a uno de referencia, como se muestra en la figura siguiente:

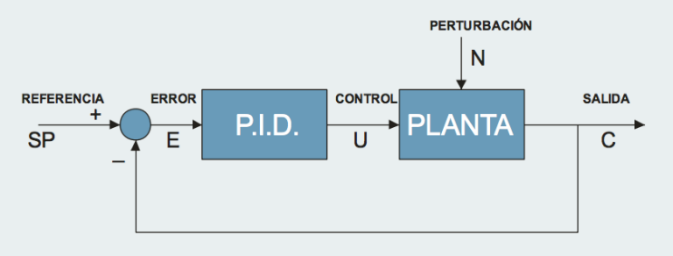

Fig.
Véase que el PID tiene como esencia que, tras el cálculo del valor, este es retroalimentado.

Para poder lograrlo, una vez que el error ha sido calculado, se ejecuta una función que cuenta con tres partes:

- Proporcional: multiplica el error por una constante kp
- Integradora: acumula el error de la vez pasada y lo pondera por una constante ki.
- Derivativa: permite que el acercamiento al valor deseado sea de la manera más leve posible, multiplica dicho valor por otra constante kd.

$$u(t) = \underbrace{K_p e(t)}_{P} + \underbrace{K_i \int e(t)dt}_{I} + \underbrace{K_d \frac{de(t)}{dt}}_{D}$$

Fig. 6
Aquí se observan las tres partes del controlador PID, a saber: la proporcional (P), la integradora (I) y la derivativa (D)

Protocolos de comunicación

La transmisión de datos de un aparato electrónico a otro es algo por demás común. No obstante, llevarlo a cabo no es tan sencillo como podría parecer. Para lograrlo, es necesario contar con protocolos de comunicación e interfaces. Por medio de la interfaz, es posible definir a uno de los aparatos como el *emisor* (el que transmite el mensaje) y a otro como el *receptor* (el que recibe el mensaje).

La interfaz mediante la cual operan los protocolos está basada en estándares que permiten establecer el canal de comunicación entre los aparatos en cuestión. La obediencia a estos estándares es lo que permite establecer la comunicación entre aparatos electrónicos, desde pequeños XBee hasta impresoras y hardware más complejo.

Existen varios tipos de comunicación, entre los cuales destacan la comunicación en serie y la comunicación en paralelo. La primera de ellas se caracteriza por la transmisión de datos a través de un único canal, mientras que la que es en paralelo se lleva a cabo mediante múltiples canales de manera simultánea.

A continuación se muestran dos figuras que ejemplifican la comunicación serial y en paralelo:

Fig. 7
Como puede observarse, los datos son enviados de un punto A a uno B uno tras otro.

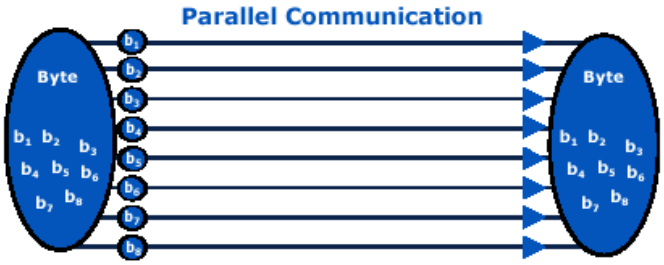

Fig. 8
En la comunicación paralela, los datos pueden ser enviados del punto A al punto B por medio de los diversos canales, permitiendo así que se haga de manera simultánea.

III. Desarrollo

El proyecto final consistió en desarrollar tres pilares fundmentales que se pudieron hacer por separado y que, posteriormente, se unieron para crear un solo sistema conjunto.
Para ello, fue necesario construir un robot en forma de carrito al cual llamamos chancla, sobre el que se montó el programa y cuya estructura debió de contener las siguientes piezas:

1. 1 chancla
2. 2 llantas
3. 2 sujetadores para los servomotores y los opto interruptores
4. 2 sujetadores de las llantas para los encoder
5. 2 servomotores
6. 2 opto interruptores
7. 1 canica

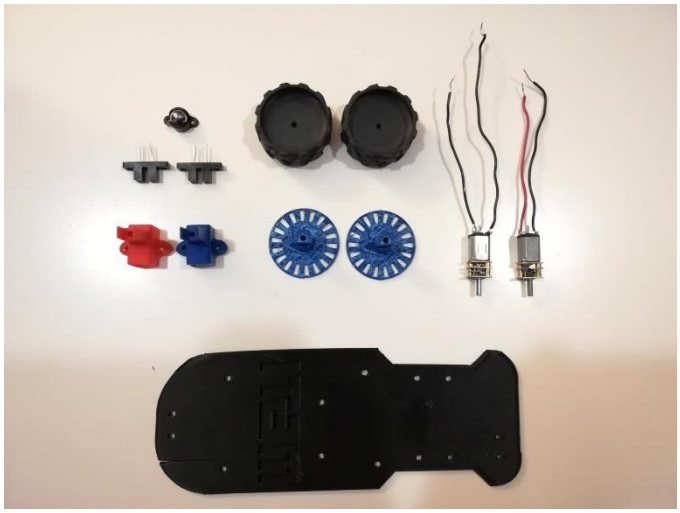

Fig.9
Piezas del robot.

A la lista anterior se agrega un Arduino Mega 2560 y un Xbee Serie 1 con shield para Arduino.

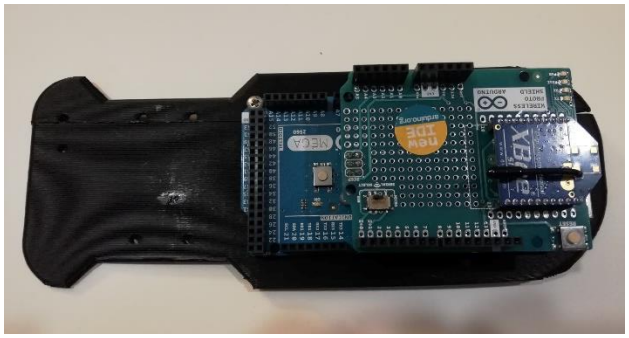

Fig.10
Chancla con Arduino y Xbee sobre shield integrados.

Debido a faltas de impresiones de un par de piezas, fue en la semana posterior a la asignación del proyecto cuando cuando el robot quedó montado, incluyendo el cableado de los servomotores con puente H y el cableado correspondiente a los Encoders con sus respectivos opto interruptores.
Finalmente, el robot tenia el aspecto a continuación mostrado:

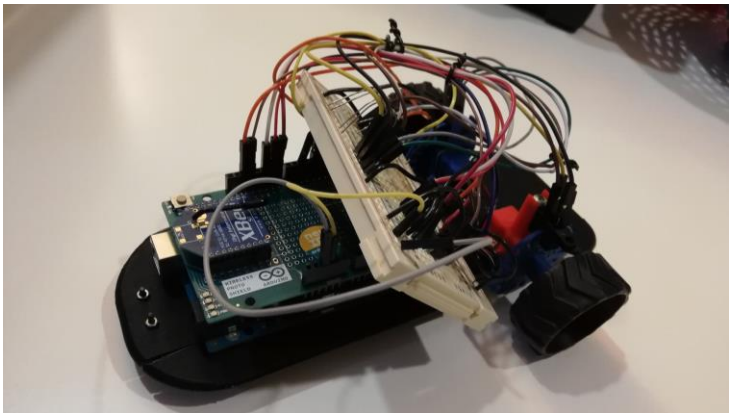

Fig.11
Robot. Toma 1

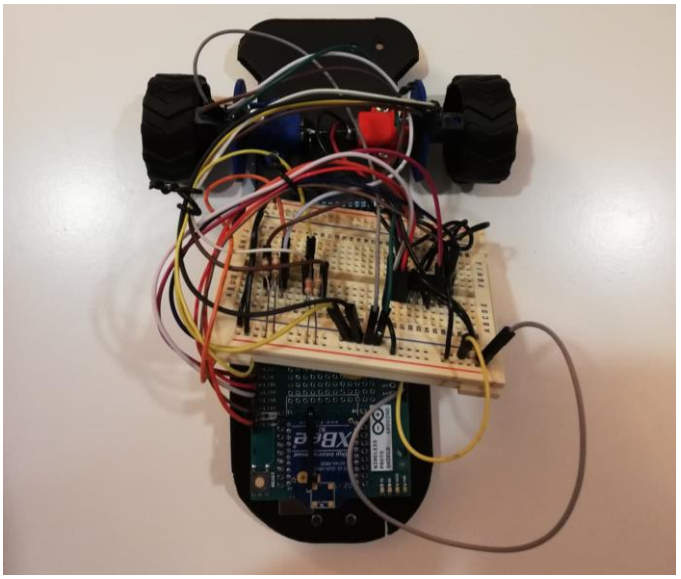

Fig.12
Robot. Toma 2

Fig.13
Robot por la parte de abajo. Toma 3

PARTE 1

Comenzamos implementando el puente H L293, idéntico al que se presentó en una ocasión en la práctica 3 bajo el siguiente diagrama y sin agregar código en Arduino más que para probar que el puente estuviera bien configurado, con la intención de dejar el alambrado para utilizarlo posteriormente.

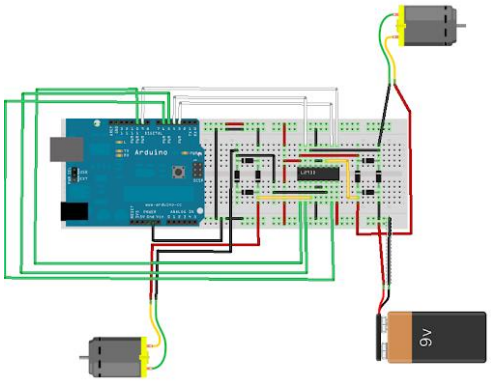

Fig.14
Circuito utilizado para puente H.

EL motivo de utilizar el puente H fue que , debido a las ecuaciones planteadas y el objetivo de girar el carrito, era necesario que las llantas pudieran hacer revoluciones en ambas direcciones de manera independiente una de otra.

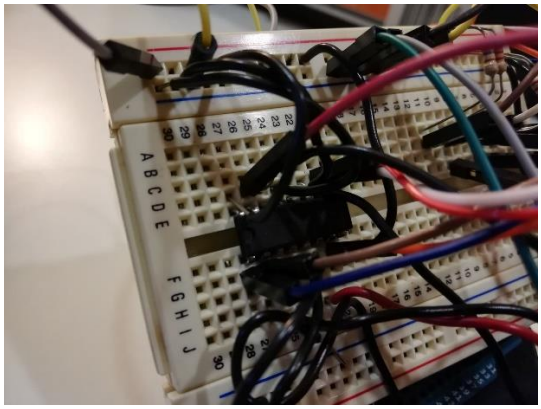

Fig.15
Circuito utilizado para puente H.

PARTE 2

La segunda parte que construimos fueron los encoder, que consistían en un par de opto interruptores montados sobre la parte de la llanta con su respectivo sujetador con 18 orificios cada uno. Estos sirvieron para calcular las velocidades reales de cada llanta de manera independiente al contar interrupciones cada vez que una parte del sujetador de la llanta interrumpía la señal del opto interruptor. Para eso utilizamos un alambrado de este estilo:

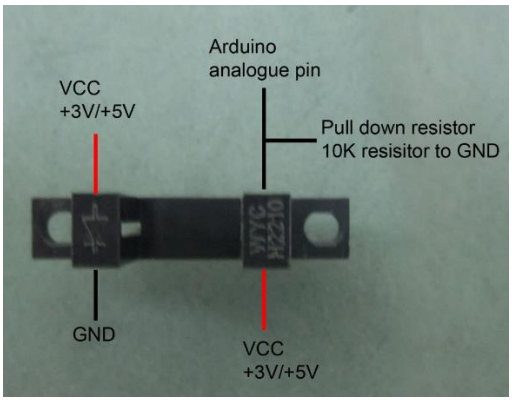

Fig.16
Referencias de cableado para opto interruptor.

El encoder funcionaba perfectamente bien y lo integramos a la chancla sin ninguna dificultad. El problema con esta parte surgió cuando lo utilizamos para calcular la velocidad. El problema en sí fue comprender el concepto de interrupciones con el comando de Arduino *attachInterrupt()* pues resultaba algo complejo cuando nos lo explicó el profesor Edgar. Una vez leida la documentación en Arduino y analizado unos cuantos ejemplos, logramos crear interrupciones de tal modo que nos guiara a una función específica encargada de contar las veces que se interrumpía una señar en un lapso de tiempo denominado *deltaT*.

Debido a que el sujetador de la llanta tenía 18 agujeros, las revoli¿uciones se calculaban cada 18 intrrupciones sobre un *deltaT*, luego se hacía la transformación de Rps a velocidad lineal para utilizar esa velocidad de la llanta en el controlador PID.

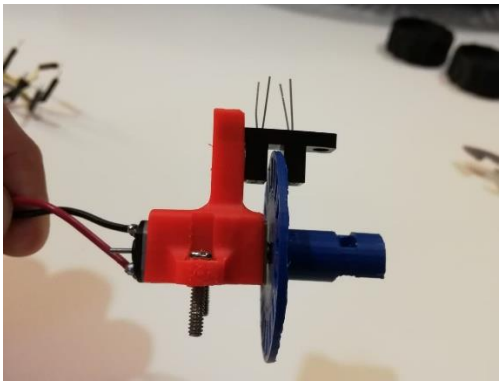

Fig.17
Encoder con servomotor integrado.

PARTE 3

Continuando con el proyecto pasamos a la parte de implementar el controlador PID. En teoría, conocíamos muy bien estos sistemas de control de estabilidad, pues habíamos hecho controladores en Matlab y simulink. Las nociones teóricas eran claras; sin embargo, como suele

suceder en todos los casos, la práctica fue distinta y tuvo complicaciones.
En esta sección nuestro mayor problema fue entender que velocidad cera cual; es decir, teníamos las velocidades deseadas, las velocidades que inyectaríamos a las llantas y además las velocidades reales calculadas por las llantas. Logramos hacer el código de un PID para los respectivos motores de la chancla, pero algo no funcionaba, el valor aumentaba siempre hasta legar al límite físico del motor. Insisto, el motivo era que no utilizábamos de manera correcta las fórmulas y que, además, le dimos parámetros erróneos a la función del PID.
La forma en la que logramos resolver el problema fue concentrándonos en cada línea de código para poder entender que proceso sucedía, nos apoyamos imprimiendo en la consola cada resultado computado para comprobar que fuesen correctos. Una vez que hallamos el error del código (el tema de las velocidades) fue cuestión de cambiar los parámetros de tal modo que hicieran justo lo que deseábamos. Al hacerlo, logramos implementar el PID y unirlo a los componentes descritos en las primeras partes del proyecto. El paso siguiente era hacer programación en ROS utilizando Sublime text.

PARTE 4.

Esta parte del trabajo consistió en poder suscribirse a ROS por medio de un archivo cpp, recibiendo las coordenadas de una bolsa, en donde se estaban publicando constantemente. No obstante, previo a crear el archivo cpp, se ejecutaron los siguientes comandos en la terminal para verificar el funcionamiento adecuado de la bolsa, y conocer la información que estaba siendo publicada:

- *$roscore:* inicia el núcleo central de ROS.
- *$git clone https://*github.com/garygra/PM_proyecto_fial.git: este comando clona el repositorio junto con los archivos en el directorio bajo el que se ejecute en la terminal.
- *$cd PM_proyecto_final*: cambio de directorio.
- *$catkin_make*: crea los nuevos paquetes de acuerdo con la dirección anterior.
- *$source devel/setup*.bash: encuentra los ejecutables dentro del paquete. Es importante que este comando se ejecute en cada una de las terminales que se abran.
- *$roslaunch graphical_client graphical_client*.launch: este commando daba como resultado la apertura de una nueva ventana en la cual se veía la "cancha", misma que representaba el escenario bajo el cual el robot tendría que moverse.
- *$rosbag play rosbags/no_obstacles_2*.bag: como se vio en la parte introductoria teórica, este comando pone a funcionar una rosbag, en este caso la ruta sin obstáculos número dos. El resultado de esto era que en la pantalla apareciera la posición inicial y el punto al cual tenía que ser dirigido el robot.

Ahora bien, para que el robot pudiera ser dirigido, era necesario conocer sus coordenadas, mismas que serían pasadas posteriormente al mismo robot a través del XBee. Dicha información estaba contenida en el tópico de nombre /y_r0, y era posible acceder a él bajo los siguientes comandos:

- *$rostopic list -v:* este comando despliega todos los tópicos que están siendo publicados y a los que es posible acceder mediante los suscriptores. Aquí era donde se encontraba el /y_r0, junto con otros, tales como /ball, que contenía las coordenadas de la meta.
- *$rosbag info /y_r0:* este comando permitía conocer la información de las constantes publicaciones que se estaban llevando a cabo en la bolsa. Como se trataba de un objeto tipo Pose2D, los datos venían en la forma:

X:	Coordenada
Y:	Coordenada
Theta:	Valor

Una vez que dichos comandos fueron ejecutados y se comprobó la información contenida en la bolsa, se procedió a la elaboración del archivo cpp y de su correspondiente ejecutable. Para esto, fue necesario tomar en consideración que para poder ejecutar un archivo cpp por medio de ROS, es menester que se modifique también el archivo CMakeLists.txt.

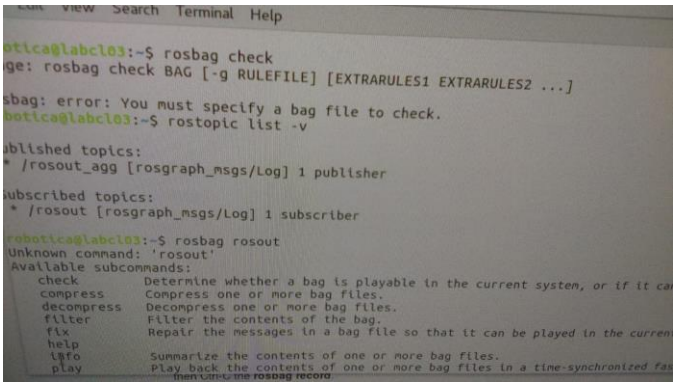

Fig.18

Nótese que aquí ocurre que no se encuentra el /y_r0. Esto es de importancia porque se trató de uno de los errores que nos retrasaron. No habíamos terminado de comprender que para que pudiese aparecer, era necesario primero correr los comandos que abrían la pantalla de la cancha y situaban las coordenadas.

Para poder modificar el archivo CMakeLists.txt, se agregaron las dos líneas siguientes:

- *add_executable(exec_subscriber src/subscriber_pf.cpp):* este comando lo que hace es crear un archivo de nombre exec_subscriber a partir del que se encuentra en la dirección src/subscriber_pf.cpp. Aquí también ocurrió un error, puesto que el archivo no se encontraba en el directorio src. Después fue corregido.
- *target_link_libraries (exec_subscriber ${catkin_LIBRARIES}):* esta línea lo que hace es ligar las librerías con el ejecutable.

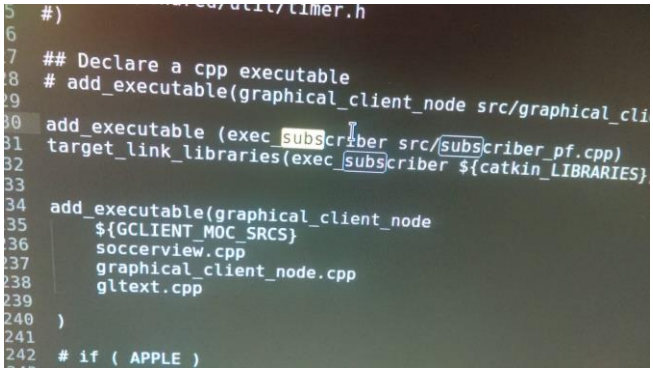

Fig.19

Aquí se puede observar el par de líneas que fueron agregadas. En el archivo que se ha adjuntado al reporte de este proyecto, ya aparece con la dirección adecuada (sin el src).

Para que el CMakeLists.txt tuviese sentido, era necesario que el cpp tuviese el nombre *subscriber_pf.cpp*. Este archivo, del cual se mostrará una imagen a continuación, tuvo las siguientes funciones:

- Suscribirse al tópico /yr_0 en donde se estaban publicando las coordenadas del robot.
- Publicar información que sería enviada al robot (esta parte es tratada en secciones posteriores).

```
float x;
float y;
float theta;
geometry_msgs::Pose2D singlePos;
geometry_msgs::Pose2D ballPos;

float xBall;
float yBall;
float thetaBall;

void get_msg(const geometry_msgs::Pose2D& msg) {
        x = msg.x;
        y = msg.y;
        theta = msg.theta;
        singlePos.x = x;
        singlePos.y = y;
        singlePos.theta = theta;
}

void get_ball(const geometry_msgs::Pose2D& mensaje){
        xBall = mensaje.x;
        yBall = mensaje.y;
        thetaBall = mensaje.theta;
        ballPos.x = xBall;
        ballPos.y = yBall;
        ballPos.theta = thetaBall;
}


int main(int argc, char **argv){
        ros::init(argc,argv,"ejemplo_sub_node");
        ros::NodeHandle nh;
        ROS_INFO_STREAM("ejemplo_sub_node initialized");
        ROS_INFO_STREAM(ros::this_node::getName());

        ros::Subscriber sub_vel = nh.subscribe("y_r0", 1000, &get_msg);
```

Fig.20

Obsérvese que la última línea de código mostrada en la figura es la que se suscribe al tópico y_r0. Cuando esta línea es ejecutada, se manda llamar a la primera función de la imagen, es decir, a get_msg(.). Esta función lo que hace es asignar a las variables x, y & theta lo que recibe el mensaje, que es de tipo geometry_msgs::Pose2D.

Después de varios intentos, este código fue ejecutado con éxito desde la terminal, dando como resultado la recepción adecuada de los valores de posición del robot.

Ahora bien, para que el robot funcionara era necesario también establecer la conexión ROS-Arduino. Para ello, se siguieron las guías proporcionadas en la práctica 5, además de algunas otras. A grandes rasgos, establecer la conexión requiere descargar la librería de ROS para Arduino y posteriormente realizar la instalación, e incluirla en el archivo que se está trabajando con #include<ros.h>. En otro de los archivos adjuntos al presente trabajo se puede observar más a detalle el uso de ROS dentro del Arduino que, en realidad, era útil para hacer callbacks al momento en el que se recibía un mensaje proveniente del XBee maestro, conectado a la computadora (a ser tratado en otra sección).

PARTE 5.

Una vez programada la parte de ROS y estructurada la parte mecánica del robot; es decir, la parte programada del Arduino. Necesitamos utilizar Xbee para hacer una conexión entre la tarjeta Arduino colocada en la chancla y el ordenador donde corría ROS.
Esta fue la parte de la práctica donde fracasamos.
Iniciamos utilizando XCTU para configurar ambos Xbee entre ellos.
Ya logrado lo anterior, de la misma manera que se hizo en la práctica 3, utilizamos ROS para programar los Xbee como maestro y esclavo, que resulta equivalente a publicador y subscriptor, para que de un ordenador con ROS en ejecución se pudieran enviar las velocidades de las llanta, calculadas y determinadas a su vez, con base en los datos que ROS tomaba de su subscripción al nodo proporcionado por el profesor, mientras que el otro Xbee recibía como parámetros estas velocidades y tenía que hacer su ajuste mediante el PID,.
Por una parte, en La terminal de ROS indicaba que estaba transmitiendo datos a través del Xbee, pero el de Arduino jamás indicó que estuviese recibiendo respuesta alguna, el único error que nos indicaba era la carencia de la librería *#include <ros/ros.h>* una vez agregada la librería solicitada por el programa, dejó de marcar errores. El código cargaba de manera adecuada y los controladores funcionaban, pero sin lograr en ningún momento la conexión.
Por cuestiones de tiempo al momento de tener que presentar, tuvimos que dejar el proyecto hasta este punto, motivo por el cual nos fue imposible hacer las pruebas con o sin obstáculos; sin embargo, el resto del proyecto estaba en condiciones de ser ejecutado para probarse de manera independiente

IV. Resultados

El proyecto no pudo ser implementado con éxito. Aunque por separado, las cinco partes parecían funcionar adecuadamente, lo cierto fue que el no poder conectar de manera exitosa los XBees por medio del archivo cpp no permitió coordinar las diversas facetas del proyecto y, por ende, no se pudieron realizar las pruebas, ni sin obstáculos ni con ellos. En concreto:

1. Fue posible realizar un PID que funcionara para una velocidad deseada dada en cada uno de los motores.
2. Se logró realizar un movimiento diferencial de acuerdo con las fórmulas teóricas presentadas en la primera parte del trabajo.
3. Fue posible establecer una comunicación entre los XBees pero sin lograr el paso correcto de la información.
4. Se implementó un pequeño algoritmo para conocer la velocidad del robot transformando las velocidades angulares en velocidades lineales.
5. Se intentó implementar un algoritmo para esquivar los obstáculos, pero nunca pudo ser probado por la limitación de comunicación con Xbee mencionada anteriormente.
6. Se logró que la velocidad angular fuera calculada correctamente a través de los optointerruptores, por medio de los contadores correspondientes.

Así mismo, las fallas que se presentaron e impidieron el funcionamiento del robot fueron las siguientes:

1. Nunca se pudo realizar el paso de información de manera correcta entre los XBees. Se configuraron adecuadamente y se logró su detección mutua, pero no fue posible transmitir la información.
2. Se falló en recorrer una ruta de manera satisfactoria, tanto sin, como con obstáculos.

V. Conclusiones

Pablo A. Ruz Salmones: el proyecto final resultó muy bueno para poder aprender a conjuntar los diversos conocimientos que se fueron adquiriendo a lo largo de las sesiones del laboratorio. Los archivos adjuntos son muestra patente de que dicho conocimiento existió y de que, además, se implementó en su mayor parte. En concreto, aprendí:

1. La utilidad de saber conectar diversas partes para que trabajen para lograr un fin determinado.
2. A calcular bien velocidades angulares y establecer una relación directa entre la parte teórica y la parte práctica.
3. Realizar un PID de manera exitosa: este punto es muy importante porque durante la práctica de los OpAms, fue

algo que nunca pude hacer, por más que lo intenté, así que el haber logrado hacerlo para este caso representa un avance y es muestra del aprendizaje continuo.

En términos no personales, se puede concluir que:

1. El robot no funcionó por la falta de comunicación de los XBees.
2. Que las fórmulas teóricas se aplican bien, aunque siempre, en el hacer empírico, existen diferencias que pueden llegar a resultar significativas. Por ejemplo, el hecho de que uno de los sensores al principio no funcionara bien, hizo que las fórmulas resultaran poco útiles.
3. Se pudo concluir de manera medianamente satisfactoria el objetivo de este proyecto, aunque nunca pudo ser puesto a prueba.

Victor M. Thomas Ruiz: Sin duda, el proyecto final resultó mucho más complejo de lo que en un principio, y en forma teórica parecía. Desde un inicio las limitaciones para comprender el funcionamiento de este y el modo de desarrollarlo se hicieron notar, y a pesar de que recurrimos en repetidas ocasiones al profesor Edgar para recibir ayuda, lográbamos sacar algunas dudas y entrar en otras. El motivo fue que lográbamos entender que se tenía que hacer en cada sección, pero surgía la problemática del ¿Cómo?

Quizás nuestra forma de organizarnos no fue la mejor e invertimos muy mal nuestro tiempo, ya que tareas medianamente sencillas nos tomaron varias horas de trabajo pues desde un inicio no se nos explicó a fondo que, y cómo se haría el proyecto, por lo que tuvimos que investigar y preguntar muchísimo para llegar a una etapa en la que pudiéramos empezar con las funciones importantes a implementar sobre el robot.

Hablo por los dos, mi compañero Pablo y yo, cuando digo que aprendimos bastante, pues a pesar de no lograr con éxito el proyecto, trajimos a cuenta los temas vistos en las 5 prácticas anteriores de un modo en el que entendimos por completo como se desarrollaban. Con la excepción de la comunicación Xbee, ahora somos capaces de implementar nuestro conocimiento del laboratorio para hacer interrupciones para correr procesos mientras que otros están en ejecución, para utilizar muchísimos comandos de arduino en diversos hábitos, para crear controladores que utilicen partes móviles, entre otras cosas. Por esa parte, nos vamos contentos.

Mientras avanzábamos en las diversas etapas del proyecto, nos vimos en la necesidad de leer cantidad de manuales y acceder a decenas de foros para adquirir conocimiento y cumplir con los requerimientos especificados en el proyecto, cosa que si bien, nos hizo invertir muchísimo tiempo, nos hizo seguir el método de prueba y error de un modo tal que al tenr una versión final de cada parte, entendíamos perfectamente que significaba cada línea de código y cada comando y con ello, logramos concluir con una versión casi terminada del proyecto. Fue muy difícil; sin embargo, por las habilidades adquiridas al término del proyecto, considero que ha valido la pena el esfuerzo; y aun más contemplando que se hizo el mejor intento que se pudo de nuestra parte.

VI. Roles

Pablo A. Ruz Salmones: como se trató de un proyecto muy pesado, y dado que nos cuesta algo de trabajo, intentamos realizar todas las partes en conjunto para ir entendiendo mejor cada una de las acciones que estábamos tomando y las consecuencias de las mismas para el robot. Hubo una carga de trabajo de Arduino un poco más del lado de Manuel y una mía del lado de ROS, pero colaboramos en todo, como siempre, para poder sacar el proyecto adelante.

Manuel Thomas: Dado que en este caso no se trató de una práctica más, sino del proyecto final, nos vimos en la necesidad de invertir muchas más horas de trabajo para desarrollar el robot. En primera instancia tanto Pablo como yo, trabajamos al a par en las secciones de alambrado e implementación del PID, con forme pasaron los días fuimos distribuyendo el trabajo de tal modo que mi compañero se encargó prácticamente de toda la parte de ROS, mientras que mi trabajo se enfocó más en los programas de Arduino. Finalmente, unos días de la entrega, colaboramos sobre las partes faltantes para concluir el proyecto, trabajamos de forma equitativa enfocando prioritariamente nuestros esfuerzos particularmente en aquellos temas que dominamos mejor cada uno.

VII. Referencias

http://cosweb1.fau.edu/~jordanrg/phy2048/chapter_9/notes_9.pdf
https://hackernoon.com/unicycle-to-differential-drive-courseras-control-of-mobile-robots-with-ros-and-rosbots-part-2-6d27d15f2010
https://sceweb.sce.uhcl.edu/harman/A_CRS_ROS_I_SEMINAR/3_4_ROSBAG.pdf
http://robertocapobianco.com/wp-content/uploads/teachpress/course_3/introduction-to-ros.pdf
http://www.everlight.com/file/ProductFile/ITR8102.pdf
http://ocw.uc3m.es/tecnologia-electronica/instrumentacion-electronica-i/material-de-clase-1/tema8_MOD.pdf
http://files.microjpm.webnode.com/200002996-a274aa370d/ITR8102%20Datasheet.pdf
https://www.electronics-tutorials.ws/transistor/tran_1.html