Proyecto Final
Laboratorio Principios de Mecatrónica

Arcadio Alexis Calvillo Madrid 159702
Arlet Díaz Méndez 154840

Resumen—El presente escrito describe la implementacion de un robot autónomo diferencial que realiza un trayecto desde una posición inicial a una meta fijada en un ambiente controlado a partir de la recepción de las posiciones, del robot, los obstaculos y la meta, desde un sitema de visión a través de tópicos ROS. Para su realización, se utilizó una tarjeta Arduino Mega 2560 con una lógica programada que crea la mejor ruta para llegar de la posición inicial a la meta evitando los obstaculos. Para la realización de este proyecto es necesario el uso de tecnologías de telecomunicaciones, mecánicas y eléctricas. La finalidad de este proyecto es reunir todos los elementos vistos en las prácticas anteriores en un solo proyecto.

1. INTRODUCCIÓN

En la actualidad, la presencia de robots es cada vez más frecuente en muchos aspectos de la vida cotidiana y de las diferentes industrias. La movilidad autónoma de dispositivos mediante la dirección de información recibida desde otro dispositivo es una práctica común que sirve para la automatización de estos procesos. Este tipo de sistemas es muy interesante no solo por sus aplicaciones en la industria sino también por los procesos de ingeniería que existen detrás. El control de robots de cualquier tipo utiliza técnicas especializadas de mecatrónica, electrónica y robótica. En el caso de robots que establecen comunicación, en especial comunicación inalámbrica, hacen uso de las prácticas de telecomunicaciones. El presente proyecto se concentra en desarrollar un sistema que se mueva de manera autónoma en un campo límitado para llegar de una posición inicial a una meta a través de los datos recibidos desde un sistema de visión.

Para la realización de este sistema autónomo se usó una tarjeta programable Arduino Mega2560, esta controla la lógica completa del sistema. A partir de ella se crean comunicaciones con ROS y en ella se encuentra el controlador PID. A esta tarjeta se transfieren los datos del sistema de visión y de los optointerruptores con los que se controla la velocidad. La transferencia de estos datos se realiza de forma inalámbrica a través de dispositivos XBee que transmiten la información mediante tópicos de ROS.Para que el robot se desplace se usan dos motores de corriente continua. Cada uno se maneja mediante un control PID. Los optointerruptores mandan la información de la velocidad a la que se está moviendo cada motor y así esta puede ser modificada mediante el algortimo de manejo que se implementó en Arduino.

Este documento tiene la siguiente organización: un Marco Teórico en el que se explican las tecnologías utilizadas, así como los conocimientos teóricos que las respaldan, un Desarrollo en el que se explica cómo contribuyen los componentes a la solución, una sección de Resultados en la que se habla de los valores obtenidos durante la práctica, luego se presentan las Conclusiones de la práctica, los Roles de cada integrante del equipo y las Referencias utilizadas durante el proceso.

2. MARCO TEÓRICO

Arduino Mega Es una placa de Arduino (compañía de desarrollo abierto de hardware) que dispone de un microcontrolador AVR Atmel de 8 bits. Se considera la placa más potente de Arduino debido a que es el que más pines i/o tiene, es apto para trabajos más complejos que el que soportan otras placas. El Arduino Mega usa el microcontrolador Atmega2560. [1]

IDE Arduino El entorno de desarrollo integrado de Arduino es una aplicación disponible para varios sistemas operativos desarrollado en Java que se utiliza para escribir y cargar programas en las placas Arduino. Este IDE permite el uso de los lenguajes C y C++ pero se utilizan reglas especiales de estructura de códigos.[2]

Lenguaje C Es un lenguaje de programación imperativo, de propósito general, que admite programación estructurada, alcance de variable léxica y recursión, lo que se define como lenguaje de alto nivel. Sin embargo, cuenta con un sistema de tipo estático que evita muchas operaciones no intencionadas en bajo nivel. Lo anterior se debe a que es un lenguaje orientado a la implementación del sistema operativo Unix.[3]

XBee Los 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 punto a multipunto o para redes punto a punto. Fueron diseñados para aplicaciones que requieren de un alto tráfico de datos, baja latencia y una sincronización de comunicación predecible. [4]

ROS Robotic Operating System (Sistema Operativo Robótico) es un framework para el desarrollo de softare especializado en robots. Provee los servicios estándar de un sistema operativo: abtracción de hardware, control de

dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos y mantenimiento de paquetes. Funciona bajo una estructura de gráfos en la cual los distintos nodos pueden recibir, mandar y multiplexar mensajes. Las aplicaciones de Ros son las siguientes:

- Un nodo principal de coordinación.
- Publicación o subscripción de flujos de datos: imágenes, estéreo, láser, control, actuador, contacto, etc.
- Multiplexación de la información.
- Creación y destrucción de nodos.
- Los nodos están perfectamente distribuidos, permitiendo procesamiento distribuido en múltiples núcleos, multiprocesamiento, GPUs y clústeres.
- Login.
- Parámetros de servidor.
- Testeo de sistemas.

Estas aplicaciones se utilizan para el correcto funcionamiento de los programas, pues ROS funciona principalmente por la comunicación de mensajes entre distintos dispositivos. Los programas pueden ser escritos en distintos lenguajes de alto nivel. Para esta práctica utilizaremos C/C++. [5]

MatLab es un entorno de cálculo técnico de altas prestaciones para cálculo numérico y visualización. Integra análisis numérico, cálculo matricial, procesamiento de señales y gráficos en un entorno fácil de usar, donde los problemas y las soluciones son expresados como se escriben matemáticamente. El nombre MATLAB proviene de "MATrix LABoratory" (Laboratorio de Matrices). El sistema de computo numérico de MatLab se utiliza en el IDE MatLab mediante el lenguaje M.[6]

Motor DC El motor de corriente continua (motor DC) es una máquina que convierte la energía eléctrica en mecánica, provocando un movimiento rotatorio. Se compone principalmente de dos partes, un estator que da soporte mecánico al aparato y tiene un hueco en el centro generalmente de forma cilíndrica. En el estator además se encuentran los polos, que pueden ser de imanes permanentes o devanados con hilo de cobre sobre núcleo de hierro. El rotor es generalmente de forma cilíndrica, también devanado y con núcleo, al que llega la corriente mediante dos escobillas. [7]

Controlador PID Un Controlador Proporcional, Integral y Derivativo es un mecanismo de control simultaneo por realimentación. Calcula la desviación o error entre un valor medido y un valor deseado. El algoritmo del control PID consiste de tres parámetros distintos: el **proporcional**, el **integral** y el **derivativo**. El valor Proporcional depende del error actual. El Integral depende de los errores pasados y el Derivativo es una predicción de los errores futuros. La suma de estas tres acciones es usada para ajustar al proceso por medio de un elemento de control. [8]

Puente H Para controlar el movimiento de un motor DC es necesario implementar algun circuito eléctrico que dirija el movimiento. Un puente H es un circuito electrico que funciona a traves de dos interrupciones de la correinte electrica y la tierra. Al crear cortos circuitos ocasióna que el motor se mueva en un sentido o en otro o frene. [9]

Sensor Optointerruptor Este tipo de sensor funciona con la emisión de luz infrarroja desde un led y una fototransistor que solo es sensible a luz infrarroja. El led se encuentra de frente al fototransistor, por lo que detecta inmediatamente si un objeto fue interpuesto entra ambos.

Bateria LIPO La batería de polímero de litio (comunmente LIPO) es una pila recargable compuesta de distintas celdas en paralelo que aumenta la capacidad de la corriente de descarga. Funcionan mediante el intercambio de electrones entre el material del electrodo negativo y el material del electrodo positivo mediante un medio conductor.

3. DESARROLLO

Para la realización de este proyecto primero se realizaron de manera individual las distintas partes que al final se unierían para que el robot lograra llegar de una posición inicial a una meta sin encontrar obstaculos. A continuación, se muestra el desarrollo de estas partes en el orden en que se llevaron a cabo.

3.1. Estructura física

Para la estructura física del motor utilizamos un chasis, dos encoders unidos a las bases del motor, dos sensores optointerruptores, un balín, dos ruedas pequeñas y una protoboard. Los componentes se unieron como se muestra en la figura 1. La protoboard sirvió para conectar los componentes electrónicos necesarios para el funcionamiento del sistema.

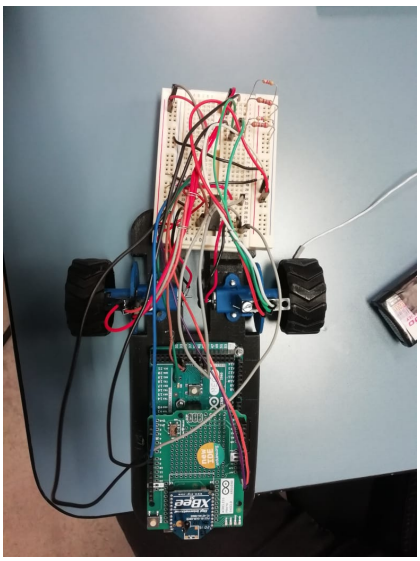

Figura 1: Estructura del robot

3.2. Circuitos electrónicos

En la protobord se implemento un puente H con uso del circuito integrado L293D. Como se menciona en el marco teórico este circuito sirve para que el motor gire en ambos sentidos o se detenga. Los sensores optointerruptores integrados usados se conectan a la protobord y todos los inputs y outputs que van al Arduino también pasan a través del protoboard.

3.3. Control PID

Para cada uno de los dos motores DC se implementó un control PID que se encarga de controlar la velocidad a la que gira cada rueda. Estos controles se programaron en C++. Para poder integrarlo al sistema físico, fue necesario establecer qué pines servirían como inputs y cuáles como outputs. A continuación se muestra el código correspondiente al PID del motor izquierdo. Se está omitiendo la declaración de variables y la inclusión de librerías. Este controlador se realizó en C en el IDE de Arduino.

```
double PIDIzq(double e){
 double ki;
 double kp;
 double kd;
 double u;
 double integ;
 double di;
 //double d0;

 ki=0.8;
 kp=2.3;
 kd=.7;
 errTI=errTI+e;
 integ=errTI;
 di=e-eI;
 eI=e;
 u=kp*e+ki*integ+kd*di;
 di=u/10;
 Serial.println(di);
 Serial.println(" Der ");
 Serial.println( u);
 return u;
 }
```

El PID de la rueda derecha funciona de manera análoga, sin embargo, con diferentes ganancias de la parte proporcional, integral y diferencial. En el método void de Arduino se encuentra las instrucciones necesarias para hacer girar al motor correspondiente a cierta velocidad.

3.4. Conexión ROS - XBee - Arduino

Para conocer la posición actual del robot, de la meta y de los obstáculos es neceario establecer una conexión entre el sistema de visión y el robot. Estas posiciones son publicadas en tópicos de ROS que deben ser comunicadas al Arduino mediante XBees. También. mediante un nodo de ROS se determina la velocidad a la que debe ir el robot. Esta conexión establece la relación entre los elementos a bordo del robot, y los elementos que recibe del exterior para cumplir su objetivo. Mediante los Xbees se establece una red de comunicación para poder leer e interpretar los datos de posicion del robot, los obstaculos y la meta, al igual que la velocidad. Los Xbees son configurados por medio de ROS usando la librería *rosserial-xbee* de tal forma que el Xbee conectado al arduino pudiera publicar los resultados al otro Xbee, por medio de un callback, y poder graficarlos. Para establecer esto se usaron los siguientes comandos en una terminal al mismo tiempo que en otra terminal se ejecuta ROS:

```
rosrun rosserial xbee setup xbee .py  C  /dev/
    ttyUSB0 0
rosrun rosserial xbee setup xbee .py /dev/ttyUSB0 1
    rosrun rosserial xbee xbee network.py /dev/
    ttyUSB 01
```

3.5. Algoritmo de búsqueda de ruta

Para resolver el problema de llegar a un punto indicado desde un punto de inicio, se creo un algoritmo cuyo objetivo es llegar a la meta de una forma eficiente. El primer paso para esto fue modelar la ruta en MatLab para así poder crear simulaciones de la ruta creada con el algoritmo. En la figura 2 se muestra la primera aproximación, se crea una líne recta y se identifican los obstaculos que interfieren con esta ruta inicial.

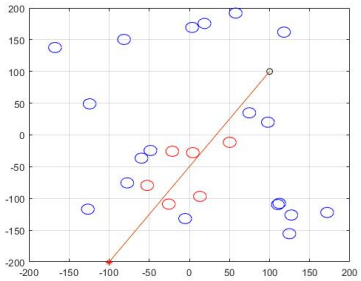

Figura 2: Ruta inicial e identificación de obstaculos

A partir de la información de la ubicación de los obstaculos en la ruta, se hacen ajustes para no crear una nueva ruta que se vuelva a encontrar con ellos. La figura 3 ilustra este procedimiento.

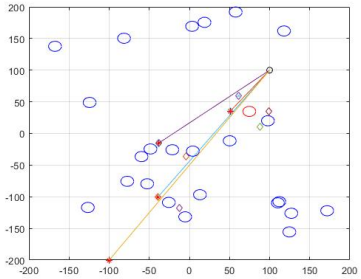

Figura 3: Ajustes en la ruta

Finalmente, teniendo la ubicación de los obstaculos y los posibles ajustes se crea una ruta que llegará de un punto de inicio a una meta rodeando los obstaculos. En la figura 4 se muestra el resultado para el campo y los obstaculos usados en las figuras anteriores.

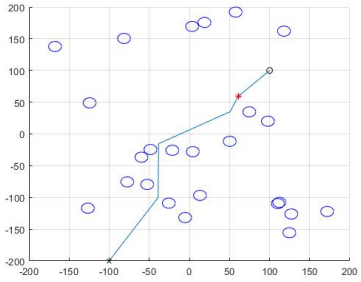

Figura 4: Ruta final

Para la implementación del algoritmo en este proyecto

fue necesario traducir el algoritmo a Python para que pudiera recibir los datos de ubicación de los tópicos de ROS. A continuación,se muestra el código correspondiente a la búsqueda de los obstaculos que interfieren en la ruta inicial (línea recta).

```
import numpy as np
def calculaObstaculos  (ini, fin, Obs, numObs):
    #Recta y parametros para determinar si hay
        obstaculos en ella
    m = (fin(1)-ini(1))/(fin(0)-ini(0))
    b = ini(1)-m*ini(1)
    numInter=0

    for i in range(1,numObs):
        d=abs(Obs(i,2)-m*(Obs(i,1))-b)/sqrt(m^2+1)
        d2=abs(ini(0)-fin(0))-max(abs([ini(0)-(Obs(i
            ,0)+4),fin(0)-(Obs(i,0)+4)]))
        d3=abs(ini(1)-fin(1))-max(abs([ini(1)-(Obs(i
            ,1)+4),fin(1)-(Obs(i,1)+4)]))
        if (d2>0 &d3>0)&d<40:
            numInter=numInter+1
            #Arreglo que nos indica cuales circulos del
                arreglo interfieren
            circuloN(numInter)=i
     if (circuloN(0)!=0):
        d=(Obs(circuloN(0),0)-ini(0))^2+(Obs(circuloN
            (0),1)-ini(1))^2
        for i in range(1,num):
            dn=(Obs(circuloN(i),0)-ini(0))^2+(Obs(
                circuloN(i),1)-ini(1))^2
            if dn<d:
                d=dn
                circuloN(1)=circuloN(i)
     return circuloN
```

Teniendo la localización de los obstaculos entonces se procede a crear una ruta. Mediante los ajustes que se simularon en la figura 3 se crea la ruta final que será seguida por el programa de Arduino gracias a que existe una comunicación constante con el sistema de visión. En este código se pueden observar pequeños cambios respecto al código de MatLab principalmente porque en Python no existen las matrices, sino que se utilizaron areglos de arreglos.

```
def calculaRuta  (ini, fin, Obs):
    numObs= len(Obs)
    m=(fin(1)-ini(1))/(fin(0)-ini(0))
    circuloN = alculaObstaculos(ini,fin,Obs,numObs)
    #Vector con los puntos a los que debe llegar
    numPuntos=1
    if circuloN(0)==0:
        numInter=0
    else
        numInter=len(circuloN)

    puntos(1)=[ini, math.atan(m)]
    i=1
    r=16
    numI=0
    while(numInter>=1 & numI<200):
          p1=[r*sin(puntos(numPuntos,2)-math.pi)+Obs(
              circuloN(i),0),r*cos(puntos(numPuntos,2)
              )+Obs(circuloN(i),1)]
          p2=[-r*sin(puntos(numPuntos,2)+math.pi)+Obs(
              circuloN(i),0),-r*cos(puntos(numPuntos
              ,2))+Obs(circuloN(i),1)]
          cipi=calculaObstaculos(ini,p1,Obs,numObs)
          if(cipi(0)<numInter|cipi(0)==0):
              numPuntos=1+numPuntos
              puntos(numPuntos)=[p1,math.atan(((p1(1)-
                  ini(1))/(p1(0)-ini(0))))]
              ini=p1
              circuloN=calculaObstaculos(ini,fin,Obs,
                  numObs)
              r=r+4
          else:
              cipi=calculaObstaculos(ini,p2,Obs,numObs
                  )
              if(cipi(1)<numInter|cipi(1)==0):
                  numPuntos=1+numPuntos
                  puntos(numPuntos)=[p2,math.atan(((p2
                      (1)-ini(1))/(p2(0)-ini(0))))];
                  ini=p2
                  circuloN=calculaObstaculos(ini,fin,
                      Obs,numObs)
                  r=r+4
              else:
                  r=r+32
          if(circuloN(1)==0):
              numInter=0
          else:
              numInter=len(circuloN)
          numI=numI+1
    puntos(numPuntos+1)=[fin,math.atan((fin(1)-ini(1)
        )/(fin(0)-ini(0)))]
    ruta=puntos
     return ruta
```

Al estar constantemente recibiendo datos de ROS el robot debe moverse en el campo para lograr su objetivo de manera satisfactoria.

3.6. Implementación final

Para el correcto funcionamiento del sistema, es necesario unir la estructura física a los circuitos electrónicos y a Arduino. Con esto se logra la funcionalidad mínima del robot, pues con ello ya puede moverse. Con esto, gracias al PID, el robot podrá moverse de manera autónoma. Con los puentes H, se podrá lograr que las llantas giren en un sentido u otro o que no se muevan. Por otra parte, es necesario crear la conexión entre Arduino y ROS a través de XBee para que el robot obtenga los datos del sistema de visión y cree la ruta necesaria para poder llegar de un punto inicial a la meta. En la figura 5 se muestran los componentes que integran el robot final.

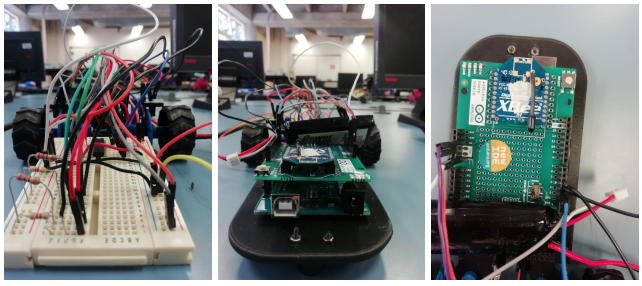

Figura 5: Implementación final del robot

4. RESULTADOS

La implementación del PID se llevó a cabo de manera exitosa para ambas llantas. Cuando el sistema era alimentado por la computadora, este corría en excelentes condiciones respetando las velocidades programadas para cada una de las llantas. Sin embargo, al conectar la bateria LIPO el robot no respetaba estas velocidades e incluso algunas veces no giraba en el sentido adecuado. Por lo anterior, creemos que el error se debe a algun elemento faltante para cerrar de manera correcta el circuito electrico y que la correinte se distribuya de la manera correcta en el robot, y no a una falla

en la programación. No pudimos encontrar dicho error de conexión para corregirlo.

Por otra parte, sabemos que el algoritmo para encontrar la ruta funciona de manera correcta pues fue probado con distintos números de objetos, así como distintos puntos de inicio y distintas metas. El programa en MatLab mostraba un funcionamiento apropiado con hasta 75 obstaculos. Sin embargo, para un número mayor de obstáculos su identificación en la ruta inicial requiere mucho más tiempo de cómputo y en algunas ocasiones la ruta encontrada presentaba fallos.

En cuestión de la comunicación del sistema solo logramos crear comunicación entre la consola de Arduino y ROS mediante la terminal. La conexión de los XBees no se pudo lograr debido a que no pudimos encontrar los parámetros de configuración necesarios para crear la red. Debido a esta falla, tampoco pudimos acceder a los datos arrojados por el sistema de visión para poder probar los distintos componentes del robot.

Debido a lo anterior, la ruta en Python nunca se llegó a probar, por lo que no sabemos si funciona de la manera correcta. Sabemos que es posible que existan errores, pues la lógica detrás de este lenguaje es muy distinta a la de Python. Sin embargo, en caso de haber fallos solo sería necesario hacer correcciones de adaptación, pues el algoritmo de búsqueda de ruta debe ser el mismo.

5. CONCLUSIONES

Arlet: La elaboración de cada uno de los componentes resultó relativamente fácil pues la mayoría de ellos ya se habían implementado en prácticas anteriores. Sin embargo, la comunicación central ROS-Arduino-Xbee no se logró debido en gran parte a que en la práctica de protocolos de comunicación se llevaban a cabo de forma independiente. Por otra parte, era necesario utilizar algunas librerías de ROS para lograr la conexión con XBee que no pudimos agregar de forma correcta. Creo que con un poco más de tiempo hubieramos podido investigar la forma de lograr el sistema de comunicación. En lo que respecta a la implementación de los circuitos electricos, el más importante fue el alambrado de los dos puentes H para controlar la dirección de los motores. De ahí en fuera, el alambrado para el circuito general fue relativamente sencillo pues en todas las prácticas era necesario crear algún circuito. En cuanto al control PID, lo más relevante fue la búsqueda de los valores para la relación de ganancia necesaria. En la práctica en la que se simuló con MatLab esto se hizo de manera más o menos empírica, por lo que en esta ocasión también hicimos la búsqueda a prueba y error. Para el algoritmo de la búsqueda se tomaron en cuenta algoritmos estudiados en otras clases como Estructuras de Datos Avenzadas e Inteligencia Artificial. Sin embargo, al final mi compañero hizo una investigación más extensa de soluciones con lo que logramos construir el algoritmo final. En esta solución nos dimos cuenta que en las clases antes mencionadas generalmente no se tiene un plano de referencia por lo que los desplazamientos tienen un grado de restricción menor. Durante la realización de esta práctica aprendí que en proyectos de mediana escala como este es importante crear valor a partir de la implementación de las partes aisladas. Sin embargo, también me di cuenta que la conexión de todos los elementos independientes puede ser a veces más importante, pues sin esto el objetivo final puede llegar a no ser cumplido, como en nuestro caso.

Alexis: Pese a que cada una de las partes por separado funcionaba de manera correcta no se logró unir todo de forma que el robot fuera totalmente autónomo. Uno de los aspectos fundamentales del correcto funcionamiento y que era prácticamente el pegamento de las piezas es la comunicación ROS-XBee-Arduino.Implementar el alambrado del robot autónomo en realidad no fue muy difícil, una de las partes más complicadas fue el uso de los encoders y el tiempo de muestreo, que era fundamental, para calcular las velocidades que se debían enviar a los controladores PID. En específico respecto a los controladores PID se tienen que tener diversas consideraciones con las constantes relacionadas a cada uno de los motores ya que estas, a pesar de ser muy parecidas, definen comportamientos totalmente diferentes, además, dado que no se usó ninguna librería, el cálculo de las constantes fue totalmente empírico.

En lo que respecta a la búsqueda y formación de una ruta entre dos puntos con obstáculos se pensaron diversos algoritmos: A*, cálculo de centroides, diagramas de Voronoi, cuadriculación del espacio, puntos aleatorios y nuestra propuesta mediante la generación de puntos perpendiculares al obstáculo más cercano. Cabe resaltar que existen muchas otras opciones, pero no siempre son las más eficientes en términos de tiempo y capacidad de cómputo.

En general aprendí que una de las cosas con las que más se tiene que tener cuidado es la comunicación entre componentes y la obtención de datos mediante los sensores.

6. ROL O PAPEL

Arlet: Desarrollo del algoritmo de búsqueda. Modificación de los archivos para el establecimiento de la comunicación entre nodos ROS. Programación en Python. Programación del PID en Arduino. Implementación del hardware para la elaboración del robot.

Alexis: Desarrollo del algoritmo de búsqueda. Modificación de los archivos para el establecimiento de la comunicación entre nodos ROS. Programación en MatLab. Programación del PID en Arduino. Implementación del hardware para la elaboración del robot.

7. FUENTES CONSULTADAS

[1] Arduino. Arduino Mega 2560. [En línea]. Disponible en: http://arduino.cl/arduino-mega-2560/

[2] SparkFun. What is an arduino? [Online]. Available: https: //learn.sparkfun.com/tutorials/what- is- an- arduino/all

[3] Bonet, E. Lenguaje C. [En línea]. Disponible en: https://informatica.uv.es/estguia/ATD/apuntes/laboratorio/LenguajC.pdf

[4] XBee Cl. ¿Qué es Xbee? [En línea]. Disponible en: https://xbee.cl/que-es-xbee/

[5] Wikipedia. Sistema Operativo Robótico. [En línea]. Disponible en: https://es.wikipedia.org/wiki/SistemaOperativoRobótico

[6] Departamento de Informática de la Universidad de La Laguna. Introducción a MatLab. Descripción de Matlab.

[En línea]. Disponible en: http://nereida.deioc.ull.es/ pcgu-ll/ihiu01/cdrom/matlab/contenido/node2.html

[7] Motor DC. [En línea]. Disponible en: http://www.geekbotelectronics.com/motores-de-dc/

[8] Benites, H. ANEXO 1: SINTONI-ZACION A TRAVÉS DEL METODO DE ZIEGLER-NICHOLS. [En línea]. Disponible en: http://tesis.pucp.edu.pe/repositorio/bitstream/handle/123456789/6185