ISSN 1870-4069

Diseño y desarrollo de un robot de servicio con un control de velocidad PID

Rosario Baltazar, Martín Sánchez, Miguel Ángel Casillas,
Liliana Sánchez, Arnulfo Alanis

Tecnológico Nacional de México,
Instituto Tecnológico de León,
México

{rosario.baltazar, 15241180, liliana.sanchez,
miguel.casillas}@leon.tecnm.mx,
alanis@tectijuana.edu.mx

Resumen. En este trabajo se muestra el diseño y desarrollo de un Robot de Servicio con velocidad controlada a través de un Control PID (Proporcional Integral Derivativo) para ser aplicado en terapias para niños con capacidades diferentes o adultos mayores con Alzheimer. El robot se diseñó con base en el microcontrolador ESP 32 y se aprovechó la capacidad de su doble núcleo para que en uno de los núcleos se realizara el control de la rueda izquierda y en el otro núcleo se programó el control de la rueda derecha. El robot también cuenta con diferentes sensores, entre ellos un acelerómetro que permite conocer la orientación del robot y su desplazamiento. Se cuenta con un encoder para conocer la velocidad del robot y lograr el control a través de las revoluciones por minuto RPM. También se incluye un sensor GPS para tener la localización del robot en tiempo real. Finalmente se agregaron 6 sensores ultrasónicos para evitar colisiones. Una de las grandes ventajas es que todo el diseño del hardware y los programas son de elaboración propia, es resultado del trabajo de los miembros de grupo de ambientes inteligentes, esto permitirá en un futuro, agregar más propiedades al robot.

Palabras clave: Control PID, ESP 32, acelerómetro, encoder, GPS.

Design and Development of a Service Robot with PID Velocity Control

Abstract. The design and development of a Service Robot with controlled speed through a PID Control to be applied in therapies for children with different capacities or to the elderly people with Alzheimer's is presented. The robot was designed based on the ESP 32 microcontroller and the capacity of its dual core was used so that the left wheel was controlled in one of the cores and the right wheel control was programmed in the other core, as well. The robot shown in this work also has different sensors, including an accelerometer that allows knowing the orientation of the robot and its displacement. An encoder was implemented to know the speed of the robot and achieve control through the RPM. A GPS sensor is also included to have the location of the robot in real time and thus facilitate its

control. Finally 6 ultrasonic sensors were added to avoid collisions. One of the great advantages is that all the hardware design and the programs are self-made, it is the result of the work of the members of the intelligent environments group, this will allow in the future to add more properties to the robot.

Keywords: PID Control, ESP 32, acelerometer, encoder, GPS.

1. Introducción

Desde hace algún tiempo se ha investigado el efecto que tienen los robots en las personas. Un ejemplo de ello es la investigación acerca del uso de robots de servicio para la asistencia a personas de la tercera edad como lo hizo Ariño [1] en sus tesis.

Paro es un robot con forma de foca que se ha usado para terapias con adultos mayores en el trabajo [2] muestra los resultados de los efectos psicológicos y sociales de la actividad asistida por robot durante un año en personas mayores.

Por otro lado, se ha investigado que los niños autistas son pacientes más receptivos cuando las teorías de aprendizaje o terapias son implementadas vía tecnológica como lo han revelado estudios previos; autores como [3, 4], evidencian la presencia de un claro interés en robots y en dispositivos electrónicos de diversa naturaleza.

Otros autores muestran terapias asistidas por robots para niños con autismo, que buscan provocar un patrón de comportamiento repetitivo y secuencial para desarrollar alguna habilidad y obtener retroalimentación del comportamiento en el infante [5]. Diehl [6], hace un extenso estudio de los robots y su uso en niños con autismo desde el punto de vista clínico.

También se ha desarrollado investigación con respecto a los robots sociales como herramientas para ayudar durante la terapia y la educación de niños con discapacidades intelectuales, sin embargo, hay pocos estudios que muestren el uso por largos periodos de tiempo, sin embargo, en [7] se analizó el impacto de una exposición prolongada a robots de asistencia social en un contexto educativo con estudiantes de secundaria con discapacidad intelectual y autismo. En donde los participantes respondieron positivamente ante el uso de los robots en las escuelas.

Por está razón nace la idea de crear un robot de servicio simple y de bajo costo como primera aproximación al desarrollo de un robot que permita una interacción con niños con capacidades diferentes.

Este robot es para lograr posteriormente agregar más módulos, en los cuales ya se está trabajando por el equipo de investigación, que permitan realizar terapias y otras actividades con los niños, teniendo un código propio y un hardware propio.

Este trabajo se centró principalmente en el diseño del control de la velocidad de los motores del robot, esto debido a que la velocidad de un robot de servicio o de un robot que interactúa con niños debe ser adecuada, si la velocidad es muy alta puede lastimar a los niños o personas que están interactuando con el robot. Además se están aprovechando las características del doble núcleo del ESP 32, para que la base del control recaiga en un dispositivo de muy bajo costo y fácil programación.

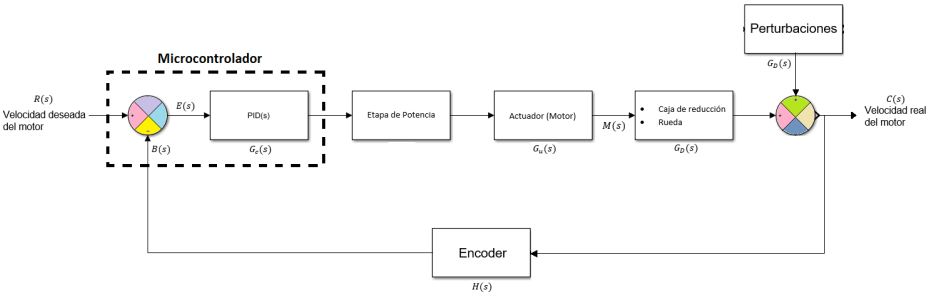

Fig. 1. Sistema de control de lazo cerrado para la velocidad del motor A.

2. Diseño del robot

En esta sección se muestran los componentes del diseño del robot y sus características.

2.1. Sensores

Para lograr que el robot tenga una cierta autonomía para su desplazamiento, se requieren diferentes sensores que permitan conocer su ubicación en tiempo real, por lo que se agregaron los siguientes componentes:

Acelerómetro. Este sensor es capaz de medir la vibración y aceleración de una estructura, en sus tres ejes conocidos en inglés como Roll, Pitch y Yaw. Este sensor es necesario para conocer la orientación del robot, es decir, si se desplaza o incluso si se cae.

Se utilizó como acelerómetro el circuito MPU-6050, este módulo tiene también un giroscopio de tres ejes con el que podemos medir velocidad angular y un acelerómetro también de 3 ejes con el que medimos los componentes x, y y z de la aceleración.

Para programarlo se definieron como variables la aceleración en g y la aceleración angular en °/s. Se obtuvieron los valores raw, se trabajo en el escalamiento de las lecturas y se aplicó un filtro. Finalmente se probó en conjunto con el ESP 32 físicamente.

Encoder. Se utilizó un encoder rotativo incremental óptico para calcular las rpm, el cual se acopló al motor y a las ruedas por medio de una banda dentada. El encoder se encarga de enviar las pulsaciones generadas por las ruedas al microcontrolador a través de interrupciones para calcular las RPM. Esto se programó para comprobar que el motor giraba exactamente 30 vueltas emitiendo 64 pulsos por cada revolución y cada pulso enviado por el encoder se efectuaba aproximadamente cada 5.6°.

Sensor de Posición GPS. En el robot se utilizó un Sistema de Posicionamiento Global (GPS; en inglés, Global Positioning System), modelo NEO-6. Estos elementos disponen de interface de comunicación UART Universal Asynchronous Receiver-Transmitter, en español:

Transmisor-Receptor Asíncrono Universal, SPI del inglés Serial Peripheral Interface o en español bus de interfaz de periféricos, Inter-Integrated Circuit (I2C) y USB (en inglés: Universal Serial Bus) o Bus Universal en Serie.

```
/*Declaración de variables PID*/
unsigned long lastTime;
double Input, Output, Setpoint;
double errSum, lastErr;
double Kp = 1, Ki = 0.6, Kd = 0.08;
int SampleTime = 1000;
double error = 0;
/*Variables para calculo de RPM*/
int in1 = 32;
int in2 = 33;
int motor_ = 16;
volatile int motor = 0;
volatile int rpm = 0;
#define channel      0
#define frecuencia   10000
#define resolucion   8
unsigned long tAnterior;
```

Fig. 2. Código para la declaración de variables.

```
void loop() {
  ledcWrite (channel, Output);          //Función
       cycle
  delay(1000);
  rpm = (motor*60/64)*1000/(millis()-tAnterior);
  tAnterior = millis();                 //Se igua
       cada segundo
  Input = (double)rpm;                  //Igualam
  unsigned long now = millis();         //se Igua
  double timeChange = double (now - lastTime);//T
  /*Calcula todos los errores*/
  double error = Setpoint - Input;
  errSum += (error * timeChange);
  double dErr = (error - lastErr) / timeChange;
  /*Calculamos la función de salida del PID*/
  Output = Kp * error + Ki * errSum + Kd * dErr;
  /*Guardamos el valor de algunas variables para
  lastErr = error;                      //Igualam
  lastTime = now;                       //Igualam
  motor = 0;                            //Se rein
}
/*Función de la interripción*/
void motor_sen()
{
  motor++;                              //Sumator
}
```

Fig. 3. Código para el cálculo del error del control PID en el ESP32.

Soportan los protocolos NMEA (National Marine Electronics Association) y RTCM que son protocolos de Internet. Además, se proporciona la localización, la fecha, la velocidad y la latitud entre otros parámetros. Se realizó la conexión para que los parámetros de ubicación se enviaran a la plataforma de IoT para transformar los datos recibidos en coordenadas en un mapa y así tener la localización del robot en tiempo real.

Sensor ultrasónico. Finalmente se usaron seis sensores ultrasónicos HC-SR04, con la finalidad de detectar obstáculos y conocer la distancia hacia ellos, se agregó un led rojo para avisar que el robot tiene un obstáculo a menos de 30 cm.

Actuadores. Este modelo del robot solo se desplaza en el plano, de tal manera que se le acoplaron un par de llantas a los motores, así mismo se incorporó una rueda que no tiene ningún control (rueda loca).

Para mover las ruedas se seleccionaron motores de corriente continua y se realizó el cálculo para conocer el par dependiendo del peso que iba a cargar nuestro robot, después de realizar el cálculo se determinó el uso de dos motoreductores metálicos 37D57L. Para lograr la conexión de estos motores con el microcontrolador se utilizaron dos puentes H de 2A.

3. Control

3.1. Controlador PID básico

Para lograr una velocidad adecuada en el desplazamiento del robot se desarrollo el diseño de un controlador PID básico como primera opción, capaz de controlar la velocidad del motor y evitar las variaciones y compensar las perturbaciones en el sistema como se muestra en la figura 1. El sistema de lazo cerrado de la figura 1 se compone de los siguientes elementos:

Fig. 4. Derivative kick. Recuperado de: [8].

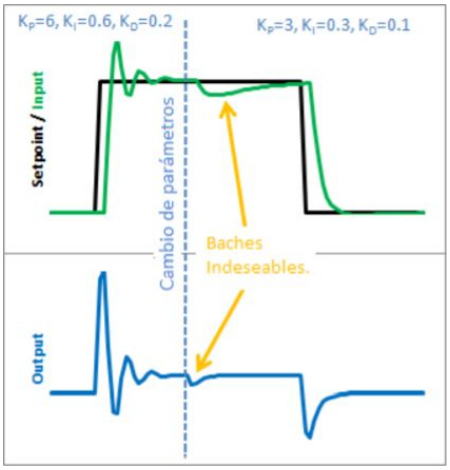

Fig. 5. Cambio de sintonización. Recuperado de: [8].

- $R(s)$ - Variable de entrada.
- $E(s)$ - Error.
- $B(s)$ - Valor medido por el sensor.
- $G_C(s)$ - Controlador.
- $G_v(s)$ - Elemento final de control.
- $M(s)$ - Variable manipulada.
- $G_P(s)$ - Planta.
- $G_D(s)$ - Perturbaciones.
- $H(s)$ - Sensor.
- $C(s)$ - Variable controlada.

Cada una de estas partes del sistema de control de lazo cerrado se denomina por una variable ya definida por la teoría de control. Este sistema de control comienza cuando se ingresa una variable de entrada $R(s)$ al sistema, en este caso la variable de entrada es "velocidad deseada del motor", después se hace un cálculo del error el cual se calcula de la siguiente manera: $E(s) = R(s) - B(s)$, es decir, el valor deseado menos el valor medido por el sensor $H(s)$, después de calcular el error, la señal entra al bloque del controlador $G_C(s)$, en este caso un controlador PID, la señal de salida del controlador PID ingresa por el bloque de la etapa de potencia para después ingresar al bloque del actuador $G_v(s)$ para reestablecer la variable controlada en su valor de referencia, después de esto la variable manipulada $M(s)$ entra en un punto de suma y se suma con las perturbaciones externas del sistema $G_D(s)$, al final del sistema de control se encuentra la variable controlada o de salida $C(s)$, la cual se retroalimenta hacia el sensor para calcular nuevamente el error del sistema.

```
30 void loop() {
31   ledcWrite (channel, Output);          //Función
         cycle
32   delay(1000);
33   rpm = (motor*60/64)*1000/(millis()-tAnterior);
34   tAnterior = millis();                 //Se igua
         cada segundo
35   Input = (double)rpm;                  //Igualam
36   unsigned long now = millis();         //se Igua
37   double timeChange = double (now - lastTime);//T
38   /*Calcula todos los errores*/
39   double error = Setpoint - Input;
40   errSum += (error * timeChange);
41   double dErr = (error - lastErr) / timeChange;
42   /*Calculamos la función de salida del PID*/
43   Output = Kp * error + Ki * errSum + Kd * dErr;
44   /*Guardamos el valor de algunas variables para
45   lastErr = error;                      //Igualam
46   lastTime = now;                       //Igualam
47   motor = 0;                            //Se rein
48 }
49 /*Función de la interripción*/
50 void motor_sen()
51 {
52   motor++;                              //Sumator
53 }
```

Fig. 6. Código para el cálculo del error del control PID con tiempos regulares en el ESP32.

```
52 /*Función para los tunnings de los valores KP, KI, KD*/
53 void SetTunings(double Kp_, double Ki_, double Kd_)
54 {
55   double SampleTimeInSec = ((double)SampleTime) / 1000;
56   Kp = Kp_;
57   Ki = Ki_ * SampleTimeInSec;
58   Kd = Kd_ / SampleTimeInSec;
59 }
60 /*Función para colocar el tiempo de muestreo*///Función que te
61 //permite cambiar a un nuevo tiempo de muestreo mientras el program
62 void SetSampleTime(int NewSampleTime)
63 {
64   if (NewSampleTime > 0)
65   {
66
67     double ratio = (double)NewSampleTime / (double)SampleTime;
68     Ki *= ratio;
69     Kd /= ratio;
70     SampleTime = (unsigned long)NewSampleTime;
71   }
72 }
73 /*Función de la interrupción*/
74 void motor_sen()
75 {
76   motor++;
77 }
```

Fig. 7. Código para el cálculo del error del control PID con tiempos regulares segunda parte en el ESP32.

De la documentación existente sobre sistemas de control, podemos destacar la ecuación 1, esta ecuación muestra la representación matemática de los 3 controladores que componen al controlador PID:

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(t)dt + K_p T_d \frac{d\,e(t)}{dt}, \tag{1}$$

donde u(t) es la salida del controlador y entrada de control al proceso, e(t) es el error de salida, K_p es la ganancia proporcional, T_i es la constante de tiempo integral y T_d es la constante de tiempo derivativa.

3.2. Código de programación del controlador PID básico

Este controlador PID se programó en el IDE de Arduino y se bajó al microcontrolador ESP 32. En el ESP 32 se tienen dos núcleos y en el robot se tienen dos ruedas, entonces se hizo uso de esa propiedad y se hizo un controlador en cada uno de los núcleos.

El programa se presenta a continuación y muestra la programación de un controlador PID básico, en el cual primeramente se definen las variables para el controlador, para el PWM y para las interrupciones del encoder.

Las revoluciones por minuto (RPM) son calculadas por medio de una ecuación que se programó en el código y que incluye los pulsos generados, los pulsos por revolución del encoder y algunas variables del tiempo.

Como primer código de programación del controlador PID tenemos en la siguiente imagen 2 el código para la declaración de variables: Posteriormente, el programa genera los cálculos para determinar los errores y la salida del controlador PID.

```
void loop() {
  ledcWrite (channel, Output);  //Función para escribir el PWM la cual
  unsigned long now = millis(); //se iguala la variable now(actual) a
  int timeChange = (now - lastTime);//Tiempo de cambio entre cálculos
  //Determina si hay que ejecutar el PID o retornar de la función//
  if (timeChange>= SampleTime)
  {
    rpm = (motor*60/64)*1000/(millis()-tAnterior);  //60 = 60s(1min),
    tAnterior = millis();             //Se iguala tiempo anterior a millis
    Input = (double)rpm;              //Igualamos las RPM de salida a la e
    /*Calcula todos los errores*/
    double error = Setpoint - Input;
    errSum += error;
    double dInput = (Input - lastInput);//Resta del Input actual menos
    /*Calculamos la función de salida del PID*/
    Output = Kp * error + Ki * errSum - Kd * dInput;
    /*Guardamos el valor de algunas variables para el próximo recalcul
    lastInput = Input;                //Igualamos el Input anterior al Input
    lastTime = now;                   //Igualamos el último tiempo de cálcul
    motor = 0;                        //Se reinicia el contador de los pulso
  }
}
/*Función para los tunnings de los valores KP, KI, KD*/
void SetTunings(double Kp_, double Ki_, double Kd_)
{
  double SampleTimeInSec = ((double)SampleTime) / 1000;
  Kp = Kp_;
  Ki = Ki_ * SampleTimeInSec;
  Kd = Kd_ / SampleTimeInSec;
}
```

Fig. 8. Código para el cálculo del error del control PID con derivative kick en el ESP32.

```
/*Función para colocar el tiempo de muestreo*/

//Función que permite cambiar al nuevo tiempo de muestreo mientras
void SetSampleTime(int NewSampleTime)
{
  if (NewSampleTime > 0)
  {

    double ratio = (double)NewSampleTime / (double)SampleTime;
    Ki *= ratio;
    Kd /= ratio;
    SampleTime = (unsigned long)NewSampleTime;
  }
}
/*Función de la interrupción*/
void motor_sen()
{
  motor++;                          //Sumatoria de los pulsos
}
```

Fig. 9. Código para el cálculo del error del control PID con derivative kick segunda parte en el ESP32.

Por último, la salida del controlador la ingresa en la función **ledcWrite()** para generar la señal PWM del motor. Esto se repite para la otra rueda. En la figura 3 está el código para el cálculo del error del controlador PID.

3.3. Código de programación del controlador PID con tiempos regulares

Para la segunda versión del código de programación del controlador PID se aseguró que la función que ejecuta el PID lo haga regularmente.

Se basa en un tiempo de ejecución predeterminado, el PID decide si debe hacer cálculos o retornar a la función. Una vez que el PID se ejecuta a intervalos regulares, los cálculos correspondientes a la parte de la derivada e integral se simplifican.

A continuación, se muestra la versión 2 del código de programación del controlador PID. Figura 6 y 7. En el código mostrado en la Figura 7 se puede observar como varía el error actual tomando como base el error anterior y como se modifica el tiempo de muestreo. La función que se encarga de calcular el PID será evaluada a tiempos regulares y el tiempo de muestreo es constante.

3.4. Código de programación del controlador PID eliminando el fenómeno "Derivative Kick"

La modificación que se presenta a continuación cambiará levemente el termino derivativo con el objetivo de eliminar el fenómeno "Derivative Kick", el cual consiste en sobrepicos que se generan en la señal de salida.

Este fenómeno, se produce por variaciones rápidas en la señal de referencia que es del tipo función escalón, debido a la presencia del término derivativo en la acción de control, la señal de control $u(t)$ contendrá una función impulso ya que, en el momento el cambio finito de la referencia, la derivada del error se hace infinita.

```
void loop() {
  ledcWrite (channel, Output);//Función para escribir el PWM
  unsigned long now = millis(); //se Iguala la variable now(a
  int timeChange = (now - lastTime);//Tiempo de cambio entre
  //Determina si hay que ejecutar el PID o retornar de la fun
  if (timeChange >= SampleTime)
  {
     rpm = (motor*60/64)*1000/(millis()-tAnterior);  //60 = 6
     tAnterior = millis();      //Se iguala el tiempo anterior
          segundo
     Input = (double)rpm;       //Igualamos las RPM de salida a
     /*Calcula todos los errores*/
     double error = Setpoint - Input;
     ITerm += (Ki * error);
     double dInput = (Input - lastInput);
     /*Calculamos la función de salida del PID*/
     Output = Kp * error + ITerm - Kd * dInput;
     /*Guardamos el valor de algunas variables para el próximo
     lastInput = Input;         //Igualamos el Input anterior al
     lastTime = now;            //Igualamos el último tiempo de c
     motor = 0;                 //Se reinicia el contador de los
  }
}
```

Fig. 10. Código para el cálculo del error del control PID eliminando cambios de sintonización en el ESP32.

```
/*Función para los tunnings de los valores KP, KI, KD*/
void SetTunings(double Kp_, double Ki_, double Kd_)
{
   double SampleTimeInSec = ((double)SampleTime) / 1000;
   Kp = Kp_;
   Ki = Ki_ * SampleTimeInSec;
   Kd = Kd_ / SampleTimeInSec;
}
/*Función para colocar el tiempo de muestreo*///Función que te
//permite cambiar a un nuevo tiempo de muestreo mientras el progr
void SetSampleTime(int NewSampleTime)
{
   if (NewSampleTime > 0)
   {
      double ratio = (double)NewSampleTime / (double)SampleTime;
      Ki *= ratio;
      Kd /= ratio;
      SampleTime = (unsigned long)NewSampleTime;
   }
}
/*Función de la interrupción*/
void motor_sen()
{
   motor++;                    //Sumatoria de los pulsos
}
```

Fig. 11. Código para el cálculo del error del control PID eliminando cambios de sintonización en el ESP32 segunda parte.

La Figura 4 ilustra el problema, siendo el $\text{error} = \text{setpoint} - \text{entrada}$, cualquier cambio en el error. Esto provoca que se visualice un sobrepico que se puede distinguir fácilmente en la salida y que se puede corregir de una manera sencilla:

$$\frac{d\,\text{Error}}{dt} = \frac{d\,\text{Setpoint}}{dt} - \frac{d\,\text{Input}}{dt}, \tag{2}$$

$$\frac{d\,\text{Error}}{dt} = -\frac{d\,\text{Input}}{dt}. \tag{3}$$

Como podemos observar en las Ecuaciones (2) y (3) la derivada del error es igual a la derivada negativa de la entrada en situaciones donde el setpoint es constante, excepto cuando el setpoint está cambiando, esto acaba siendo de gran ayuda para mejorar el código de programación. Ahora en lugar de añadir $Kd \times \text{error derivado}$, se le restará $Kd \times \text{valor de entrada derivado}$.

Esto se conoce como el uso de “derivada de la medición”. A continuación, se muestra el código de programación. En este código solo se reemplazó la derivada positiva del error con la derivada negativa de la entrada y se sustituye el último valor del error por el último valor que tomó la entrada (lastInput).

3.5. Código de programación del controlador PID eliminando cambios de sintonización

Se puede cambiar la sintonización del PID, mientras el sistema se encuentra en funcionamiento por lo que la señal de salida puede tener la siguiente forma. Como se puede observar en la Figura 5 el cambio de valores de sintonización afecta a nuestro sistema PID, el culpable del bache de la Figura 5 es el término integral; es el único

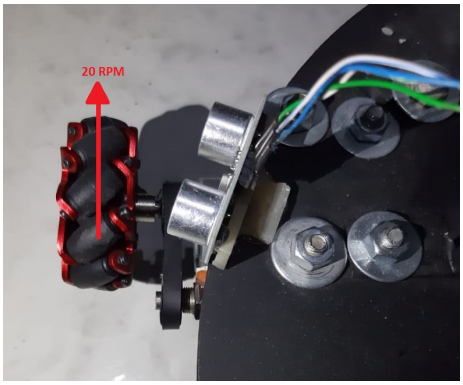

Fig. 12. Rueda izquierda.

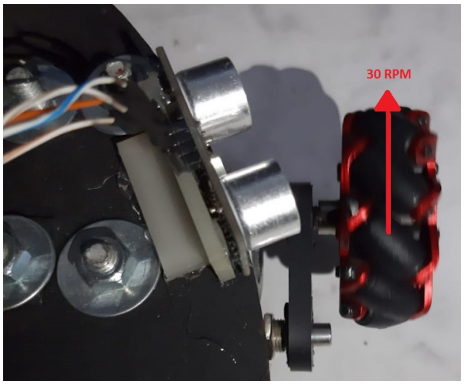

Fig. 13. Rueda derecha.

valor que cambia drásticamente la señal de salida cuando se reajusta el parámetro K_I. Esto sucede debido a la interpretación de la integral:

$$K_I \int e\, dt \approx K_{I_n} \left[e_n + e_{n-1} + ... \right]. \tag{4}$$

La interpretación de 4 funciona muy bien hasta que K_I cambia. Supongamos que en un tiempo X al operario se le ocurre cambiar el valor de K_I, lo que pasaría sería que la suma de todos los errores se multiplicaría con el valor de K_I, esto no es lo que se necesita.

Nosotros solo queremos que al cambiar de sintonización solo afecte a los valores que estén por delante. La solución a este error consiste en reescalar la suma del error, doblando el valor de K_I o cortando la suma de los errores a la mitad. esto quita el bache del término integral, solucionando el problema:

$$K_I \int edt = \int K_I e\, dt, \tag{5}$$

$$K_I \int edt \approx K_{I_n} e_n + K_{I_{n-1}} e_{n-1} + ... \tag{6}$$

En lugar de tener el término K_I fuera de la integral, lo introducimos dentro del cálculo. Hasta este punto no se ha realizado nada extraño, pero en la práctica está acción resulta en una gran diferencia en la función de salida del PID. Ahora se toma el error y se multiplica por el valor de K_I en ese momento, luego se almacena la suma de los diferentes errores multiplicados por la constante K_I.

Esto resulta en una función de salida, suave y sin sobresaltos, con la ventaja de no tener que utilizar matemática adicional para ello. A continuación se muestra el código de programación para eliminar efectos de cambios de sintonización.

4. Resultados

4.1. Resultados de la conexión del GPS

Los resultados del robot se mostrarán por partes, primero se muestra el resultado de la conexión del GPS, se verifica que el GPS establezca la conexión con los satélites, luego se comienzan a recibir datos y se envían al microcontrolador y este enviaba a su vez cada uno de los datos a la plataforma de Ubidots, entonces el microcontrolador envía una alerta por medio de un led verde y lo enciende por 500ms indicando que el envío de datos había sido un éxito.

Cuando los datos se envían satisfactoriamente, el sistema despliega la información por el monitor serial proporcionando fecha, hora, localización, el número de satélites encontrados y adicionalmente la altitud.

Cabe destacar que este tipo de GPS no solo puede brindar estos datos sino también tiene posibilidad de enviar la velocidad en nudos, la rotación en grados y la variación magnética en grados, pero estos datos se omitieron ya que solo se requieren los datos más elementales.

Ya creado y configurado el mapa en la página de Ubidots, cada vez que la página recibe un valor proveniente del microcontrolador, el mapa marca una localización aproximada de donde se encuentra el robot.

4.2. Resultados del control de velocidad

Cálculo de las revoluciones del motor DC. Como en el monitor serial del IDE de Arduino se registró una velocidad aproximada de 10 RPM, esta velocidad se calcula por medio de la ecuación en el código del cálculo de revoluciones del motor y adicionalmente también se utilizan las interrupciones que el encoder óptico emite cada vez que el motor gira.

La velocidad programada para la rueda derecha fue de 30 RPM y para la rueda izquierda de 20 RPM, en la Figura 12 y Figura 13 se muestran las llantas izquierda y derecha respectivamente con su velocidad de cada una.

Esta claro que no se puede observar el movimiento del robot, por eso en las figuras anteriores se muestra una flecha con la dirección del movimiento de las llantas y la magnitud de la velocidad. La velocidad del motor se encuentra en un intervalo de 9-11 RPM.

Este programa funciona perfectamente cuando sólo se quiere mantener el motor a una velocidad dada pero como la aplicación en este proyecto requiere que el robot tenga un control de la velocidad, para ello fue necesario la implementación de un PID que controlara la velocidad del motor.

4.3. Resultados del controlador PID básico

Después de poner en marcha el controlador PID Básico se revisó el funcionamiento pero seguía teniendo algunas variaciones en la salida, En la Figura 14 se puede observar las RPM de salida del motor utilizando la primera versión del controlador PID.

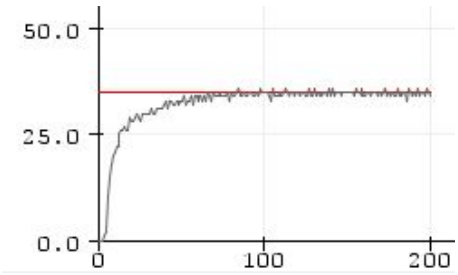

Fig. 14. Velocidad de salida utilizando el controlador PID Básico (PID_V1).

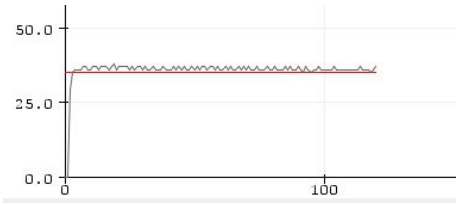

Fig. 15. Velocidad de salida utilizando el controlador PID (PID_V2).

Como se puede observar en la Figura 14, la señal de color negro representa las RPM de salida y estas se mantenían casi iguales al Setpoint (señal roja) el cual se programó de 35 RPM, el sistema se mantenía muy estable ya que no había variaciones muy grandes en la señal de salida de las RPM, pero seguía habiendo la problemática de los intervalos regulares para simplificar los cálculos.

4.4. Resultados del controlador PID con tiempos regulares

Al igual que la versión 1 del controlador PID en esta versión del controlador PID con tiempos regulares se tiene una salida en RPM del motor que se puede obtener a través del Arduino y la cual se puede observar en la Figura 15. Como se puede observar en la Figura 15, la señal de salida de la velocidad en RPM se comporta cada vez mejor, esto se debe a las modificaciones que se le hicieron al primer programa del controlador PID.

En esta figura la señal de salida de la velocidad se mantiene por debajo de la señal Setpoint pero conforme transcurre el tiempo la señal se va sobreponiendo a la señal Setpoint, lo cual refleja que las modificaciones del programa mejoraron el control de la velocidad.

4.5. Resultados del controlador PID eliminando fenómeno Derivative Kick

La versión 3 del controlador PID se encarga de eliminar el fenómeno Derivative Kick. En la Figura 16 se observa que el sistema responde cada vez mejor si se compara con la señal de la Figura 14, ya que la señal de salida de las RPM es más estable y se sobrepone sobre la señal Setpoint por lo tanto, cada vez más se tiene una versión más adecuada del control de la velocidad del robot.

4.6. Resultados del controlador PID eliminando cambios de sintonización

Para visualizar la señal de salida de la versión 4 del controlador PID de velocidad se observa la Figura 17, en ella se muestra la señal de salida de las RPM del motor DC.

Como se observa en la Figura 17, la señal de salida se comporta cada vez más estable ya que, aunque se tienen algunas pequeñas perturbaciones en la señal de salida, el sistema se estabiliza en 35 RPM desde que el controlador se pone en marcha.

Fig. 16. Velocidad de salida utilizando el controlador PID (PID_V3).

Fig. 17. Velocidad de salida utilizando el controlador PID (PID_V4).

4.7. Pruebas del GPS y el acelerómetro

Después de que se desarrolló la placa PCB, se procedió a cargar el programa donde se envían los datos de la localización obtenidos por el GPS y los datos de rotación del robot de servicio.

El programa se encarga de desplegar los datos de los puertos de transmisión y recepción, la dirección IP del internet y los datos proporcionados por el GPS (Satélites, Fecha, Hora, Latitud, Longitud), los datos de Roll y Pitch fueron proporcionados por el acelerómetro.

Cuando se despliega la información anterior el programa esperaba la conexión del GPS, esto debido a que duraba un tiempo en conectarse a los satélites para obtener la localización, después de establecer la conexión con el microcontrolador, el programa establecía conexión con Ubidots y comenzaba a enviar datos cada minuto.

Después de realizar todas las conexiones necesarias para los motores, los sensores ultrasónicos y la PCB procedimos a comprobar el funcionamiento del controlador PID con los sensores ultrasónicos.

Primeramente, se probó la placa PCB para verificar su funcionamiento con el controlador PID, en la Figura 18 se muestra la placa en funcionamiento junto con los módulos L298N y la placa del GPS.

Finalmente se pretende realizar el marketing de este robot. En este momento todos y cada uno de los sectores han tenido que evolucionar con la actualización del marketing al marketing digital, han surgido redes sociales, aplicaciones móviles, que han permitido a muchos organismos posicionar sus productos o servicios con grandes beneficios económicos.

En el giro dela Robótica no es la excepción, esta inventiva: “Robot de servicio con control de velocidad PID”, ha de introducirse también en este mundo comercial virtual, con el fin de cubrir algunas necesidades de personas con capacidades diferentes y adultos mayores, quienes deben ser cuidadosamente tratados.

5. Conclusiones

Se presentó el diseño y desarrollo de un robot de servicio para el cual se desarrollaron los cálculos para lograr el control de cada una de las ruedas. Se agregaron sensores para detección de obstáculos, de la ubicación y para la detección de giro o aceleración.

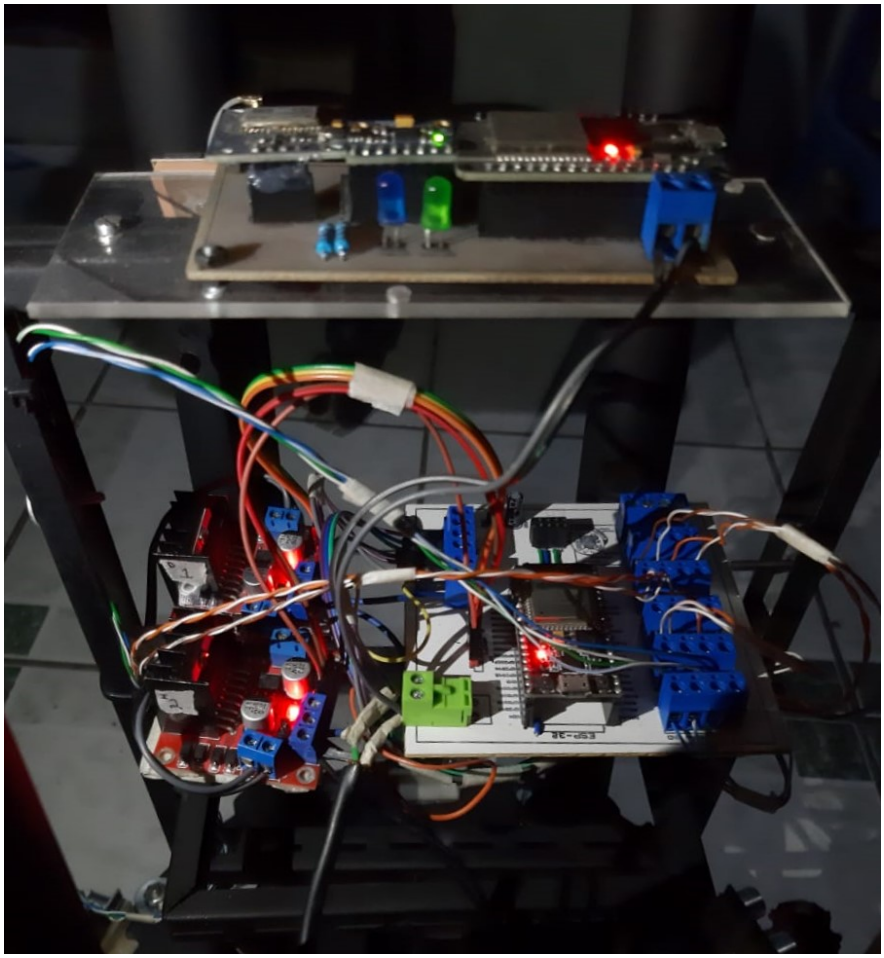

Fig. 18. Placa PCB del controlador PID en funcionamiento.

Se codificaron los encoder para el cálculo de las vueltas y esto convertirlo al control de velocidad. Se diseñó el control de la velocidad por medio de un PID digital que se programó en cada uno de los núcleos del ESP y se muestran los resultados.

La importancia de dar a conocer este producto de innovación tecnológica en esta era digital es que posibilite al segmento de personas con capacidades diferentes y adultos mayores, en cuanto a obtener una herramienta didáctica que promueva desarrollar habilidades y actitudes para mejorar su calidad de vida. Es por ello que ante ésta revolución digital se diseñan e implementan estrategias en medios digitales para su logro.

Agradecimientos. Los autores agradecen al TecNM por el apoyo al Proyecto Integración de agentes inteligentes, asistentes inteligentes e interfaces BCI para la atención de personas con capacidades diferentes.

Referencias

1. Ariño, J.: Desarrollo de un robot de servicio para la asistencia a personas de la tercer edad. Doctoral dissertation, Universitat Politècnica de València (2014)
2. Wada, K., Shibata, T., Saito, T., Sakamoto, K., Tanie, K.: Psychological and social effects of one year robot assisted activity on elderly people at a health service facility for the aged. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp. 2785–2790 (2005) doi: 10.1109/ROBOT.2005.1570535
3. Dautenhan, K.: Robots as social actors: Aurora and the case of autism. In: Proceedings of the Third International Cognitive Technology Conference, vol. 359, pp. 374 (1999)
4. Robins, B., Dautenhahn, K., Te Boerkhorst, R., Billard, A.: Robots as assistive technology - does appearance matter? In: 13th IEEE International Workshop on Robot and Human Interactive Communication, pp. 277–282 (2004) doi: 10.1109/ROMAN.2004.1374773
5. Ravindra, P., De Silva, S., Tadano, K., Saito, A., Lambacher, S. G., Higashi, M.: Therapeutic-assisted robot for children with autism. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3561–3567 (2009) doi: 10.1109/IROS.2009.5354653
6. Diehl, J. J., Schmitt, L. M., Villano, M., Crowell, C. R.: The clinical use of robots for individuals with autism spectrum disorders: A critical review. Research in Autism Spectrum Disorders, vol. 6, no. 1, pp. 249–262 (2012) doi: 10.1016/j.rasd.2011.05.006
7. Silvera-Tawil, D., Yates, C. R: Socially-assistive robots to enhance learning for secondary students with intellectual disabilities and autism. In: 27th IEEE International Symposium on Robot and Human Interactive Communication, pp. 838–843 (2018) doi: 10.1109/ROMAN.2018.8525743
8. Beauregard, B.: Improving the beginner's pid–introduction (2011) http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/comment-page-1/comments