VOLUMEN 6, N°2 / JULIO - DICIEMBRE 2024 / e - ISSN: 2661

IMPLEMENTACIÓN DE UN CONTROLADOR PARA ROBOT DE TIPO PÉNDULO INVERTIDO

Implementation of a controller for an inverted pendulum type robot

Josepablo Cruz Baas [1]	jcruzbaas@gmail.com
Jorge Ríos Martínez [1]	jorge.rios@correo.uady.mx
Ernesto Olguín Díaz [2]	ernesto.olguin@cinvestav.edu.mx
Francisco Moo Mena [1]	mmena@correo.uady.mx

[1] Universidad Autónoma de Yucatán (Facultad de Matemáticas).
[2] CINVESTAV (Unidad Saltillo).

RESUMEN

En la robótica móvil de interiores sobresalen los robots con ruedas porque su mecánica es más simple y requiere menor trabajo de mantenimiento comparados con los robots articulados de piernas. En este trabajo se describen las fases para la implementación de un software de control para un robot péndulo invertido usando el algoritmo regulador cuadrático lineal (*LQR*, por sus siglas en inglés). La arquitectura del software se puede descomponer por grandes bloques de abstracción, combinando bibliotecas de software embebido y piezas de software desarrolladas especialmente para el proyecto. La gran mayoría de proyectos con el péndulo invertido utilizan un control PID, principalmente debido a la simplicidad de su implementación. Este método es funcional, pero poco escalable, consume tiempo y, aunque algunas implementaciones presentan un excelente rechazo a perturbaciones externas, no existe una garantía de estabilidad si algún parámetro del robot cambia. El prototipo desarrollado podría servir como base de experimentación para futuras investigaciones sobre robótica móvil o control automático.

Palabras Clave: Péndulo invertido; software de control; LQR

ABSTRACT

In indoor mobile robotics, wheeled robots stand out due to their simpler mechanics and lower maintenance requirements compared to legged articulated robots. This work describes the phases for implementing control software for an inverted pendulum robot using the Linear Quadratic Regulator (LQR) algorithm. The software architecture can be decomposed into large abstraction blocks, combining embedded software libraries and software components developed specifically for the project. Most inverted pendulum projects use a PID control, primarily due to the simplicity of its implementation. This method is functional but lacks scalability, consumes time, and, although some implementations exhibit excellent disturbance rejection, there is no guarantee of stability if any robot parameter changes. The developed prototype could serve as an experimental foundation for future research in mobile robotics or automatic control.

Keywords: Inverted pendulum; control software; LQR

I. Introducción

La robótica se ha convertido en una herramienta ubicua en varios campos de la industria y la sociedad, como una tecnología que brinda enormes beneficios económicos y *sociales. En la robótica móvil de interiores sobresalen los* robots con ruedas porque su mecánica es más simple y requiere menor trabajo de mantenimiento

Fecha de Recepción: 23/nov/2023. **Fecha de Aceptación:** 01/mar/2024 **DOI:** https://doi.org/10.47187/perspectivas.6.2.222

comparados con los robots articulados de piernas.

El robot péndulo invertido, robot balancín o *two wheeled inverted pendulum robot*, en inglés, es un robot con dos ruedas que se puede mover de forma independiente alrededor del mismo eje de rotación, obteniendo así una alineación de sentido paralelo. El modelo del péndulo invertido (fig. 1) permite analizar el problema de equilibrio que presenta ese robot porque el cuerpo del robot puede abstraerse como el péndulo, y las ruedas del robot como el eje de rotación del péndulo, de ahí el nombre del robot [1].

Métodos clásicos de control bastan para estabilizar el ángulo de inclinación (o balanceo) del robot, pero no para estabilizarlo mientras se controla. Es un sistema sub-actuado porque hay más variables a controlar que acciones de control. Para conseguir un control preciso tanto de traslación como de inclinación, es necesario un controlador de retroalimentación de múltiples salidas. En este trabajo se describen las fases para la implementación de un software de control para un robot péndulo invertido usando el algoritmo regulador cuadrático lineal (*LQR*, por sus siglas en inglés). El método *LQR* proporciona ganancias de retroalimentación controladas de manera óptima para permitir el diseño de sistemas de alto rendimiento y estabilidad en circuito cerrado.

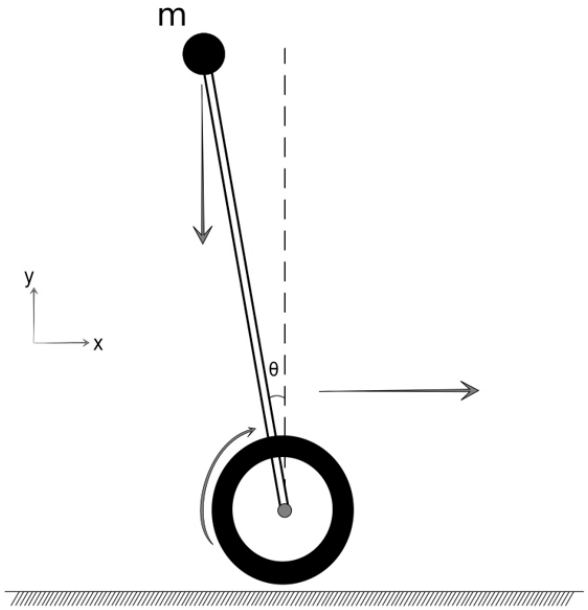

Fig. 1. *Modelo del péndulo invertido sobre una rueda o monociclo.*

II. Antecedentes

La mayoría de los proyectos que controlan de manera automática un mecanismo de tipo péndulo invertido utilizan un algoritmo de control clásico, el proporcional, integral y derivativo (PID, por sus siglas en inglés). El PID es simple de implementar y no requiere un modelo dinámico para sintonizar sus ganancias permitiendo estabilizar al robot a través de prueba y error. Este método es funcional, pero poco escalable, consume tiempo y, aunque algunas implementaciones presentan un excelente rechazo a perturbaciones externas, no existe una garantía de estabilidad si algún parámetro del robot cambia (por ejemplo: reemplazar las ruedas, o añadir/quitar algún peso). Para dar garantías de estabilidad se requiere analizar un modelo matemático, por ejemplo, el *modelo matemático simplificado de 2 grados de libertad* para el péndulo invertido (fig. 1) reduce el problema a un sistema holonómico cuyo espacio de configuraciones puede ser descrito usando una variable de posición unidimensional y una variable de inclinación del cuerpo del robot.

Según los trabajos revisados, las ecuaciones que describen la dinámica de esos sistemas se obtienen de los fundamentos de mecánica de Newton y de la mecánica de Lagrange. El libro [2] postula que el modelado a través de coordenadas generalizadas de la mecánica Lagrangiana es mucho más simple cuando la cantidad de cuerpos rígidos en el modelo es considerable. En la tabla I, se encuentran los trabajos revisados clasificados por grados de libertad y fundamentos de mecánica clásica.

Existen propuestas nuevas de control utilizando redes neuronales que muestran buenos resultados [12],[16]. Sin embargo, se requiere de un proceso de entrenamiento para representar la dinámica del robot sin certeza de la precisión del modelo, y la capacidad de cómputo necesaria para ejecutar el control en tiempo real supera las capacidades de un microcontrolador aumentando el costo del robot.

De igual forma, existen métodos de control no lineal que aumentan considerablemente la precisión del controlador incluso para el sistema

de 4 grados de libertad [3],[5],[6], pero requieren inevitablemente un filtro tipo Kalman para garantizar un control robusto.

Los controladores que se pueden implementar en un microcontrolador son: el controlador PID, el controlador LQR y el controlador de lógica difusa [17].

Tabla I. Comparación de modelos dinámicos

Fundamentos de mecánica	Modelo de 4 grados de libertad	Modelo de 2 grados de libertad
Lagrange	[3],[4],[5],[6],[7]	[8],[1],[9]
Newton/Euler	[10]	[11],[12],[13],[14],[15]

Con respecto a los sistemas embebidos de control en la literatura se encontraron trabajos basados en un procesador tipo ARMTM como la tarjeta Beagle BoneTM o la tarjeta Raspberry PiTM [5],[14],[16]. Existen ejemplos de sistemas corriendo en un ArduinoTM en [12],[13],[15],[18]-[20], en Lego MindstormsTM [21] o incluso en tarjetas FPGA [9],[10].

III. Diseño del robot

El robot RAP (fig. 2), construido para este proyecto, posee hardware electrónico que emplea componentes profesionales de bajo costo. Es un prototipo de robot móvil compuesto por un módulo de control (MC) y un módulo de inteligencia básica (MIB). El módulo de control, como su nombre lo indica, es la placa de control, un sistema embebido basado en un microcontrolador de arquitectura *ARM*™ que interactúa directamente con los sensores y actuadores esenciales del robot (acelerómetro, giroscopio, magnetómetro y encoders). Dicho sistema tiene la obligación de estabilizar el balanceo del robot y convertir comandos en movimiento. El módulo de inteligencia básica es el módulo que gobierna al MC considerando que el sistema es estable y controlable. Está basado en un microprocesador de arquitectura *ARM*™ con sistema operativo Linux. Envía comandos de acción para que el robot *RAP* se traslade. Además, el módulo de inteligencia básica se puede comunicar con otros sistemas computacionales para escalar las habilidades del robot. Este diseño divide los requerimientos en módulos independientes y autónomos, siendo el pilar del sistema el módulo de control; sin él, no es posible convertir el sistema físico en un robot móvil.

El presente trabajo se enfoca en la primera parte del proyecto, el desarrollo del controlador automático para el MC del robot *RAP*. Se sintetiza un modelo simplificado para describir la dinámica de la posición, la velocidad y el ángulo de inclinación del *RAP*, linealizándolo para un rango seguro de operación y estabilizándolo con un controlador *LQR* de tiempo discreto. Al final, se desarrolla un prototipo robótico péndulo invertido de dos ruedas diferenciales con un sistema de control embebido, que podrá servir como base de experimentación para futuras investigaciones sobre robótica móvil o control automático.

A. Diseño mecánico del robot

El robot *RAP* se compone de un carro, un cuerpo y las ruedas. El carro es la estructura que sujeta las ruedas, la batería, la electrónica de potencia y la electrónica de control. Está conformado por elementos de acrílico cortados a láser. Las ruedas del robot son ruedas de patín o scooter, con un diámetro de 144 milímetros, un grosor de 29 milímetros y un peso de 235 gramos. Cada una se encuentra acoplada al eje del reductor de un motor de corriente directa de 12 volts. Ambos motores comparten especificaciones: el reductor tiene una razón de 30:1, poseen un encoder digital de efecto hall de 64 unidades por revolución, acoplado al eje del motor antes del reductor. Ambos motores sin algún tipo de carga alcanzan una velocidad de 330 rpm con un consumo de 200 miliamperios, [mA]. Por otro lado, los motores entregan un par de fuerza de hasta 14 kilogramos por centímetro con un consumo de 5.5 amperios (esta carga es el límite máximo de los motores, el cual podría dañarlos), los motores son de la marca Pololu [22].

El robot con el cuerpo acoplado tiene una altura de 67 centímetros, sin el cuerpo tiene una altura de 20 centímetros; ambas medidas a partir del piso. El centro de masa del sistema con cuerpo tiene una altura de 17.27 centímetros a partir del eje de las ruedas, con un peso de 3.071 kilogramos.

Tabla II. Variables del modelo matemático con valores medidos y estimados		
Variable	Nombre	Valor
m_b	Masa del cuerpo del robot	3.071kg
m_w	Masa de ambas ruedas	0.470 kg
I_b	Inercia del cuerpo	0.091593 kg m^2
I_w	Inercia total de ambas ruedas	0.056870 kg m^2
R	Radio exterior de ambas ruedas	0.072 m
l_m	Altura del centro de masa del cuerpo	0.1727 m
μ_d	Coeficiente de fricción cinética del piso	0.8
g	Aceleración de la gravedad	9.8 m/s^2

B. Diseño eléctrico y electrónico del robot.

El sistema *RAP* necesita 3 tensiones diferentes de corriente directa para funcionar: 3.3, 5 y 12 volts. La electrónica de potencia consume 12 volts. Para alimentar el sistema se utiliza una batería LiPo de la marca Wild Scorpion. La batería es de 4 celdas, con una capacidad de 4200 miliamperios hora, [mAh]. El voltaje nominal de la batería es de 14.8 volts, sin embargo, el voltaje con carga completa es de 16 volts, y 14 volts cuando se ha descargado. La capacidad de carga es de entre 5 y 10 C, la capacidad de descarga es de hasta 40C [1C = 4.2 A]. Para alimentar la electrónica digital se tiene un regulador de tipo conmutado que baja la tensión de 12 volts a 5 volts, y en la placa de control se cuenta con un regulador extra de tipo lineal, para bajar la tensión de 5 volts a 3.3 volts.

Para gobernar ambos motores se requieren dos puentes H. de la marca *Freescale Semiconductor*™ modelo *MC33926*. Estos circuitos integrados soportan una salida de hasta 5 amperios cada uno. Cuentan con un monitor de corriente para que el microcontrolador, a través del ADC (convertidor de analógico a digital), pueda sensar la corriente que es entregada a los motores.

El control de potencia se hace con dos canales PWM, uno para cada dirección de giro. La frecuencia máxima de estas señales, que el circuito integrado soporta, es de 20 kHz.

La electrónica digital de control del robot *RAP*, consiste en un microcontrolador de la marca *Texas Instruments*™, modelo *Tiva C TM4C123GH6PM*, un sensor de inercia con acelerómetro, giroscopio y magnetómetro de la marca *InvenSense*™ modelo *MPU-9250*, y un módulo puente de comunicación UART a bluetooth, de la marca *Guangzhou HC Information Technology*™, modelo *HC-06*. El microcontrolador *TM4C123G* es un microcontrolador de arquitectura *ARM*™ *Cortex-4MF*, de 32 bits con unidad de coma flotante, acompañado del sistema operativo en tiempo real *TI-RTOS*, propiedad de *Texas Instruments*™ y de libre uso para los microcontroladores de su marca.

IV. Diseño del controlador

Esta sección pretende ilustrar el proceso de diseño del controlador, sin embargo, por falta de espacio, importantes detalles del análisis matemático no se incluyeron, pero están disponibles enviando un correo al autor correspondiente. El modelo matemático para trabajar es un sistema lineal de ecuaciones diferenciales conocido como representación en el espacio de estado.

A. Modelo dinámico

A partir del modelo lineal del sistema se definen las siguientes variables para simplificar el análisis.

$$H_1 \triangleq ((m_w + m_b)\, R^2 + I_w)$$
$$H'_2 \triangleq m_b\, l_m\, R$$
$$H_3 \triangleq l_b$$
$$D_1 \triangleq 2R\mu_d$$
$$g_2 \triangleq -m_b\, g l_m$$

Sean $\vec{x}$ el vector de estado y $\vec{y}$ el vector de salida. La representación del modelo dinámico en el espacio de estado es la ecuación (1).

$$\vec{x} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-g_2 H'_2}{h_1} & \frac{H_3 D_1}{h_1} & 0 \\ 0 & \frac{g_2 H_1}{h_1} & \frac{-H'_2 D_1}{h_1} & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \theta \\ \dot{\alpha} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{H_3}{h_1} \\ \frac{H'_2}{h_1} \end{bmatrix} u$$

$$\vec{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \theta \\ \dot{\alpha} \\ \dot{\theta} \end{bmatrix} \quad \vec{y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \theta \\ \dot{\alpha} \\ \dot{\theta} \end{bmatrix}$$

En la tabla II se muestran las variables obtenidas mediante mediciones físicas del robot y sus piezas. Excepto el coeficiente de fricción, el cual es un valor estimado por tablas y condiciones relativas.

El análisis se restringió a un espacio de configuraciones de dos grados de libertad (fig. 1), cuyas variables son: la posición unidimensional del robot *RAP*, y el ángulo de inclinación del cuerpo del robot *RAP*. Para que dichas restricciones tengan sentido, se asume que las ruedas diferenciales del robot RAP rotan exactamente a la misma velocidad.

Con el modelo lineal del sistema en representación de estado obtenido para una región de operación conocida, se calcula el rango de la matriz de controlabilidad, para saber si el sistema es controlable. La región de operación del robot RAP, consiste en definir el rango admisible del ángulo de inclinación del robot, el cual se define en la vecindad alrededor del cero con variaciones muy pequeñas.

Se pretende ahorrar la mayor cantidad de recursos de cómputo en el sistema embebido para poder construir sobre él otra clase de servicios además del control de la planta, por ejemplo, un muestreador en tiempo real y un sistema de configuración. El proyecto mínimo viable consistió en estabilizar la posición, velocidad y ángulo de inclinación del cuerpo, así que, ponderando los requerimientos, el sistema de control cuya implementación en software embebido es más simple y ligera, es el sistema de control *LQR*, pues consiste en una única multiplicación del vector de estado $\vec{x}$ y la matriz de control K en cada periodo de control. Utilizando *LQR* y definiendo dos matrices Q y R para la función de costo, de forma que se maximice el tiempo de respuesta y se minimice la energía de la señal de control, se encuentra el valor de la matriz de control K que estabiliza el sistema dinámico linealizado.

Posterior a esto, utilizando el software *Octave*, se codifica una simulación para el sistema lineal del robot RAP. Esto es, ingresar las ecuaciones del sistema en representación de estado a *Octave* (tanto en lazo abierto (LA) como en lazo cerrado (LC)), asignar un vector de estado inicial donde exista un ángulo de inclinación diferente de cero, graficar las respuestas del sistema sin controlador y con controlador, y comparar las respuestas.

Antes de implementar el sistema en el *RAP*, es necesario discretizarlo para observar los efectos de una frecuencia de control baja y garantizar que el sistema será estable a frecuencias determinadas por el programador. Para discretizar el sistema de control, se hace uso de la relación entre la variable x y la variable z, propuesta por el método bilineal (también conocido como Tustin). Se asigna un periodo de muestreo adecuado, y se programa al robot *RAP* para emitir por el puerto serial los valores de las variables de estado en tiempo real, capturadas en dicho periodo de control. Esta información produce las gráficas del comportamiento dinámico del sistema real previo al estabilizador.

Para obtener el vector de estado del sistema dinámico, se hace uso de un sensor de inercia MPU9250 digital (que cuenta con un acelerómetro, un giroscopio y un magnetómetro) y un encoder en cada motor del robot RAP. La estimación del estado se consigue a través de un filtro complementario, para la estimación del ángulo de inclinación del cuerpo del robot RAP y el promedio de la velocidad de las ruedas del robot RAP, para la estimación de la posición y velocidad del robot.

La intención del controlador es estabilizar las variables α, $\dot{\alpha}$, θ y $\dot{\theta}$, correspondientes a la posición y velocidad del carro, y la inclinación del péndulo. Por lo tanto, el sistema de control se simplifica dado que la entrada del sistema es el origen del espacio de estado.

La arquitectura del sistema de control se presenta en la figura 3.

En el diagrama se considera que el sistema ya está discretizado, por lo tanto, las entradas y las salidas de cada bloque son digitales, con un periodo de muestreo de T_s = 0.010 segundos. El bloque denominado *muestreador*, es el bloque que almacena las variables de estado y la salida

del sistema en tiempo real para compartirlas con el sistema de comunicación. El control de torque de los motores convierte la señal de torque T_u resultado del controlador *LQR* y produce un torque efectivo U_k que cambia el estado dinámico de la planta. El control de motores se asume lineal para este proyecto, de forma que $T_u = U_k$.

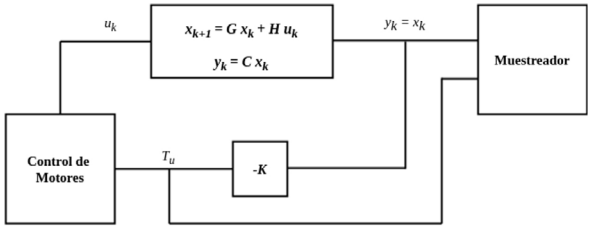

Fig. 3. *Diagrama a bloques de la propuesta de control discreto.*

El microcontrolador *Tiva C TM123* cuenta con dos unidades de medición para encoders, lo que permite independizar al CPU principal del proceso de muestreo de las ruedas. A su vez, de entre todos los periféricos interesantes que posee, sobresalen para el proyecto, un conjunto de buses IIC y buses UART. La arquitectura del µC permite implementar un sistema operativo en tiempo real *Texas Instruments* ha implementado sobre esta arquitectura su propio RTOS de nombre *Ti-RTOS*, que además de proveer interfaces para la administración lógica de tiempos y recursos, también posee controladores para todos los periféricos internos, éstos controladores son parte de una biblioteca llamada *TivaWare*, la cual viene instalada en una ROM dedicada en el propio µC, lo que acelera el proceso de desarrollo de prototipos.

En este proyecto se ha utilizado tanto la biblioteca como el *RTOS* para agilizar el prototipado, compilando el firmware desde el entorno de desarrollo *Code Composer Studio*}, el cual también es propiedad de *Texas Instruments*.

V. Arquitectura de software

Como se puede apreciar en la fig. 4, el software se puede descomponer por grandes bloques de abstracción. Los dos bloques más bajos son todas las bibliotecas de software embebido provistas por el fabricante, mientras que las capas superiores, son piezas de software desarrolladas especialmente para el proyecto.

Las piezas más importantes, son los controladores de periféricos externos del robot. Éstos son los drivers que mueven a ambos motores y permiten codificar y decodificar los datos del sensor IMU. Estos controladores se implementan a través de los drivers internos provistos por el fabricante.

La siguiente capa consiste en el sistema de control de lazo cerrado o simplemente sistema de control, y el sistema de medición de variables de ambiente o entorno, también llamado *environment*. Estas piezas de software comparten el mismo nivel porque interactúan directamente con los drivers del robot.

La capa más alta del sistema está dedicada al sistema de comunicación, aquí los procesos sólo consisten en intercambio de información entre el microcontrolador y la computadora externa.

A continuación, se describen brevemente las 3 piezas de software que dan vida al proyecto.

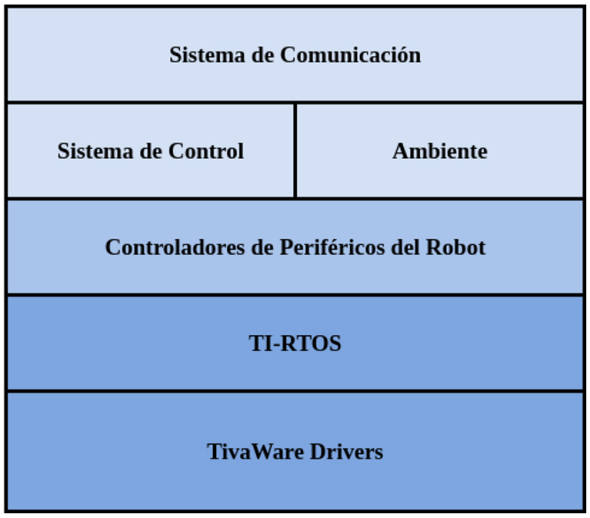

Fig. 4. *Bloques lógicos de la arquitectura del software, la posición de abajo hacia arriba, indica el nivel de abstracción, donde el nivel más alto corresponde a flujo de datos, y el nivel más bajo, a comunicación y control del hardware.*

A. Sistema de medición del entorno

Este sistema se encarga de leer cada sensor con una frecuencia de 2 milisegundos, que posteriormente procesa para construir una estructura privada denominada variables de entorno. Para que

cualquier proceso, externo al sistema de entorno, pueda acceder a esta estructura, se realiza una petición a través de la interfaz *Environment_read(variableID, ptrBuffer)*, misma que desactiva momentáneamente las interrupciones del CPU para copiar dichos datos desde la estructura hasta el buffer. La función de lectura es de tipo no reentrante[1] y, de no ser por el bloqueo de interrupciones, podría incurrir en una condición de carrera[2] que dañaría la estructura de datos, afectando al sistema de control.

B. Sistema de control de lazo cerrado

El sistema de control realiza una petición al sistema de entorno para leer las variables de estado estimadas, realiza la corrección de signo en el ángulo de inclinación y hace un llamado al *controlLoop*} y al sistema de muestreo.

El controlLoop, es una función sin argumento y que no retorna nada, su propósito es actualizar la variable local del sistema de control *u*, la cual almacena la señal que deberá ser enviada a cada motor. El cálculo de la señal, *u*, es el resultado de multiplicar el vector de estado por la matriz de ganancias (resultado de la propuesta de control) y acondicionarlo usando el factor de conversión $Km = 0.00291566$.

El sistema de muestreo consiste en rellenar una trama y agregarla a una cola a través del sistema de comunicación. Este proceso se activa o desactiva con una petición desde el sistema de comunicación.

Tanto el *controlLoop* como el sistema de muestreo se ejecutan en el periodo de control; esto es cada 10 milisegundos. La prioridad de este proceso es la más alta de todo el firmware.

VI. Sistema de comunicación

Es el proceso de menor prioridad del firmware. En él, se configura un puerto UART a 115200 baudios con una codificación de 8 bits y 1 bit de paro. El sistema de comunicación no tiene periodo de ejecución, sino que se ejecuta bajo demanda y únicamente cuando el sistema tiene tiempo libre.

A través del driver del periférico interno UART, implementado por el fabricante, se construye un hilo para recibir datos, y un hilo para transmitirlos. El hilo receptor, verifica que los bytes recibidos tienen el formato adecuado, en cuyo caso almacena una copia de estos datos y hace un llamado al despachador.

El despachador es una función que identifica el endpoint[3] donde va dirigido el paquete, dentro de una tabla de funciones. Luego hace el llamado al procesador del endpoint y espera su respuesta.

El procesador del endpoint es una función que se almacena en una lista denominada, lista de endpoints o lista de puertos. Recibe como argumento el puntero a un buffer de datos y la longitud de dichos datos. El desarrollador puede definir el formato de los datos que puede recibir (dentro de las capacidades de la especificación del sistema de comunicación) y las respuestas que se envían a través de una trama.

Por su parte, el hilo emisor se activa cuando se ha recibido una trama para enviar y sólo si el CPU tiene tiempo muerto. Este hilo empaqueta la trama en un buffer dedicado y envía el puntero al driver UART del RTOS para su transmisión. El driver UART del RTOS implementa transacciones DMA (del inglés Direct Memory Access) las cuales agilizan las transacciones entre la RAM y un periférico interno.

A. Requerimientos

El sistema de comunicación debe programarse con un protocolo de transmisión que provoque una baja carga de procesamiento por parte del RTOS. También se debe transmitir y recibir bytes sin formato definido, con especial atención a variables de tipo flotante de doble precisión. Al mismo tiempo, el sistema de comunicación debe ser capaz de transmitir muestras entre el RAP y la

[1] Una función reentrante, es una función que puede ser interrumpida y vuelta a ejecutar sin repercusiones en el resultado.

[2] Una condición de carrera sucede cuando dos procesos cuyo orden de ejecución es desconocido comparten un mismo recurso que debe consumirse en un orden determinado.

[3] En software, un endpoint es una abstracción para identificar canales de comunicación entre procesos, los canales pueden ser buffers, puertos, interfaces, etc.

computadora mientras recibe comandos de acción.

B. Tramas

Para que lo anterior sea posible, el sistema de comunicación se basa en la transmisión de paquetes de datos denominados tramas.

Una trama se compone por 2 bytes dedicados a la cabecera, de 1 a 255 bytes destinados a los datos crudos y de 1 a 5 bytes opcionales, para la suma de verificación.

La cabecera de la trama dedica 1 byte al símbolo de inicio de trama, y 1 byte para indicar la longitud de la carga de datos (payload en inglés, o datos crudos).

La suma de verificación, por su parte, dedica 5 bytes opcionales para una suma de verificación tipo CRC (no implementada por el momento) y 1 byte no opcional, para el símbolo de final de trama.

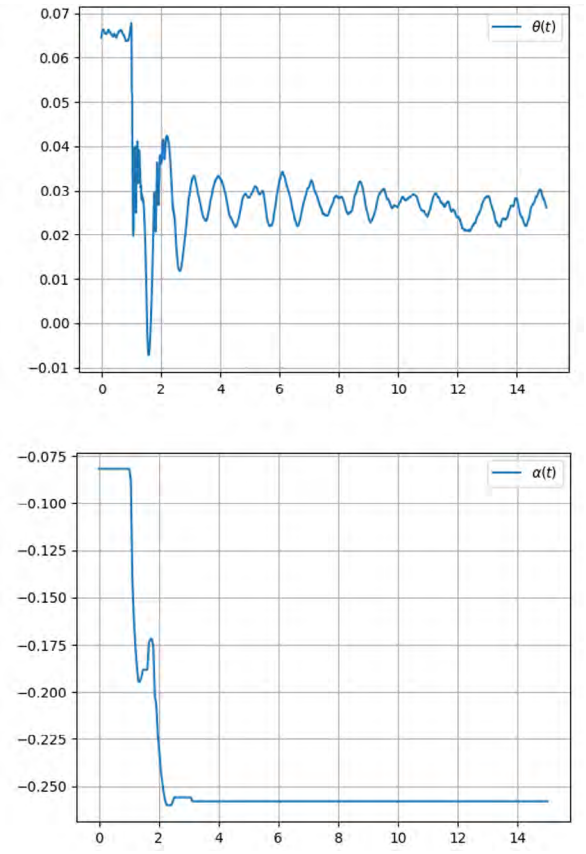

Fig. 5. *Coordenadas generalizadas muestreadas en tiempo real desde el sistema de comunicación del RAP (discretización bilineal). En la figura de arriba se encuentra la inclinación del robot en radianes. Abajo, la posición angular de la rueda virtual, también medida en radianes*

C. Canales

El sistema de comunicación no admite tramas cuyo *payload* tenga longitud 0, debido a que el primer byte del *payload* es destinado a identificar el canal o *endpoint* de comunicación. Es decir, el primer byte del *payload* indica la función que decodificará los datos de la trama. El desarrollador puede implementar sus propias funciones y definir los argumentos codificados en la trama.

El firmware del *RAP* tiene instalados 6 *endpoints*, desarrollados especialmente para el proyecto.

- **Ping:** con identificador 0 y sin recibir datos. Este *endpoint* transmite una trama de respuesta con su propio identificador.

- **startSampler:** con identificador 1, recibe 4 bytes en formato entero sin signo para solicitar una cantidad fija de muestras, o una cantidad indefinida de muestras si la cantidad es 0; las muestras son enviadas en forma de tramas con identificador 1. La codificación consiste en 5 valores en formato flotante de doble precisión cuyo orden es [variable $\propto$, variable θ, variable $\dot{\alpha}$, variable $\dot{\theta}$, variable u], la variable u, es el valor de torque aplicado a la rueda virtual del robot $\equiv[\vec{x}, u]$.

- **stopSampler:** con identificador 2 y sin recibir datos. Este *endpoint* detiene el proceso de muestreo independientemente de la cantidad de tramas restante.

- **startControl:** con identificador 3 y sin recibir datos. Activa la ejecución del *controlLoop.*

- **stopControl:** con identificador 4 y sin recibir datos. Detiene la ejecución del *controlLoop.*

- **setK:** con identificador 5, recibe 4 valores en formato flotante de doble precisión cuyo orden es: $[K_{\alpha},K_{\theta},K_{\dot{\alpha}},K_{\dot{\theta}}]$. Estos valores se cargan como la matriz de ganancias del *controlLoop.*

VII. Resultados

El modelo dinámico del péndulo invertido sobre ruedas diferenciales, reducido a un modelo discretizado, holonómico y lineal e invariante en el tiempo, resultó en una excelente aproximación a la dinámica del RAP, y permitió la implementación de un controlador LQR de tiempo discreto que resultó satisfactorio, pues es capaz de estabilizar las variables críticas del RAP, las cuales son: el ángulo de las ruedas α y su velocidad $\dot{\alpha}$, relacionadas con la posición del RAP, así como el ángulo θ de inclinación del péndulo. Para comprobar la efectividad del estabilizador se perturba al robot RAP con ligeras desviaciones del ángulo de inclinación de su cuerpo y, empujando a éste para obligarlo a moverse de lugar. Por un lado, las gráficas resultantes de esta actividad deben mostrar que las variables de estado convergen, por otro, el robot RAP debe presentar resistencia perceptible al movimiento y evitar desplomarse, con lo cual, se puede afirmar que el sistema físico es estable en lazo cerrado y el controlador funciona. Se probaron tres controladores distintos: Controlador *LQR* de tiempo continuo, Controlador *LQR* del sistema discretizado por el método ZOH y Controlador *LQR* del sistema discretizado por el método bilineal.

Todos los controladores fueron instalados a través del sistema de comunicación utilizando un script de python versión 3, solicitando al muestreador 1500 capturas, equivalente a 15 segundos de operación. El entorno de ejecución es una máquina corriendo el sistema operativo Arch Linux y el puerto serial identificado como */dev/rfcomm0*. El script de muestreo se corre con una redirección del *stdout* hacia un archivo de texto con formato CSV. El comando correspondiente es: *python3 muestreador.py $>$ samples.csv*. En esta sección solamente se muestran las gráficas del controlador que se obtuvo por el método ZOH, para el cual se aplicó una ganancia de acondicionamiento de 0.0245 unidades y la matriz instalada fue:

$$K = [-0.022051\ -1.904296\ -0.034785\ -0.353189]$$

En la fig. 5 se observa que el ángulo de inclinación del RAP es muy cercano a cero y que el ángulo de las ruedas se estabiliza, lo que físicamente se traduce en un robot RAP erguido sobre sus ruedas y estacionado en una zona cercana al origen.

Aunque cada método de discretización produjo una matriz de ganancias diferente, la implementación de ésta, para todos los casos, requirió de una ganancia de acondicionamiento para evitar saturar la señal de los actuadores. Al final, este factor, diferente para cada controlador, conllevó a una matriz de ganancias K muy similar. Las ganancias del controlador discretizado por el método de bilineal requirieron el factor de reducción menos significativo. Es por esta razón que se prefirió este resultado.

Es importante mencionar que la planta física presenta un efecto de giro sobre el acimut cuando el *RAP* retorna al origen. Esto se debe al controlador de lazo abierto implementado en cada motor, pues a pesar de que comparten especificaciones, estos presentan un desgaste y una respuesta diferente a la señal de control. Por lo que usando este método no es posible garantizar que las ruedas giren exactamente igual. Además, el control de posición también es impreciso, pues el *RAP* no retorna al origen real, sino a una región próxima al origen.

VIII. Conclusión

En este trabajo se presentó un método para implementar un controlador en el caso de un modelo dinámico para un robot péndulo invertido con dos variables de configuración, la posición unidimensional del robot y el ángulo de inclinación del cuerpo. Para linealizar el modelo dinámico del *RAP* se utilizó el punto de equilibrio inestable del sistema y la suposición de que el ángulo del cuerpo del robot tendrá variaciones despreciables.

Posteriormente, se reorganizaron las ecuaciones en forma de *representación de estado* para analizar la controlabilidad y la estabilidad del sistema, lo que permitió confirmar que el sistema dinámico linealizado del *RAP* es controlable.

Se utilizó el software *Octave* y los paquetes *control* y *signal* para comprobar que es posible controlar el RAP con un controlador *LQR* calculando una matriz de control K de tiempo continuo y

validando su estabilidad de forma numérica.

En la implementación, se propuso una frecuencia de control de 100 Hertz, y, usando el software *Octave*, se discretizó el sistema dinámico del *RAP* con el método bilineal. Posteriormente, se calculó una nueva matriz de control *K* pero esta vez de tiempo discreto, misma que pudo validarse de forma numérica al producir un comportamiento estable. La ejecución del controlador *LQR* consiste en multiplicar la matriz de control *K* por el vector de estado $\vec{x}$ en cada periodo de control.

Por otro lado, discretizar el sistema permitió observar el comportamiento del controlador para una frecuencia conveniente, elegida por el programador, asegurando la estabilidad de la planta en frecuencias bajas de control. Además, usar frecuencias bajas permitió implementar el controlador en un microcontrolador y dividir el tiempo de ejecución en múltiples tareas, sin sobrecargar el sistema embebido. Como resultado, fue más sencillo garantizar que el proceso de control se ejecute en su respectivo periodo con el menor desface posible, añadiendo mayor funcionalidad al firmware del *RAP*.

Con el fin de probar la estabilidad del *RAP*, se programó el firmware en la tarjeta de control del *RAP* usando el puerto de programación y se instaló la matriz de control *K* utilizando el sistema de comunicación. Este prototipo de robot se construyó desde cero y se documentó el proceso del controlador para que sirva como una plataforma de experimentación de control automático en un hardware real. El software de control es abierto lo que permite implementar estrategias distintas o modificar parámetros para comparar resultados.

A. Problemas abiertos

Una importante actualización al proyecto debe ser añadir el control de velocidad y considerar el ángulo de giro (el acimut del *RAP*), para lo cual, el modelo propuesto no es suficiente, porque el giro del robot y el efecto de las llantas individuales introducen nuevos fenómenos sobre la dinámica del sistema inercial, además de convertirse en un sistema, conocido en robótica como no holonómico.

Con un control de velocidad y giro, será posible construir un sistema externo de navegación a través del sistema de comunicación, añadiendo al prototipo *RAP* el módulo de inteligencia básica o *MIB*. *El MIB* se programará sobre el estándar *Robot Operating System* (ROS2) con lo cual será posible construir rutinas de evasión de obstáculos y rutinas de servicios y aplicarse como plataforma de experimentación en robótica móvil.

VII. Referencias

[1] P. Frankovský, L. Dominik, A. Gmiterko, and I. Virgala, “Modelling of two-wheeled self-balancing robot driven by gearmotors” International Journal of Applied echanics and Engineering, vol.22, 2017.

[2] E. Olguín-Díaz, “3D Motion of Rigid Bodies: A Foundation for Robot Dynamics Analysis”, ser. Studies in Systems, Decision and Control. Springer International Publishing, 2018. [Online]. Available: https://books.google.com.mx/books?id=9vt9DwAAQBAJ.

[3] G. Rigatos, K. Busawon, J. Pomares, and M. Abbaszadeh, “Nonlinear optimal control for the wheeled inverted pendu-lum system” Robotica, vol. 38, no. 1, pp. 29–47, 2020.

[4] S. Kim and S. Kwon, “Dynamic modeling of a two-wheeled inverted pendulum balancing mobile robot” International Journal of Control, Automation and Systems, vol. 13, no. 4, pp. 926–933, 2015.

[5] C.-F. Hsu and W.-F. Kao, “Double-loop fuzzy motion control with cog supervisor for two-wheeled self-balancing assistant robots” International Journal of Dynamics and Control, pp. 1–16,2020.

[6] H. Ahmadi Jeyed and A. Ghaffari, “A nonlinear optimal control based on the sdre technique for the two-wheeled self-balancing robot” Australian Journal of Mechanical Enginee-ring, pp.1–9, 2020.

[7] R. D. Huerta Gil, “Control de un p´endulo invertido sobre dos ruedas de tres grados de libertad” Thesis, Universidad Nacional

Autónoma de México, 2014.

[8] M. Fauziyah, Z. Amalia, I. Siradjuddin, D. Dewatama, R. P. Wicaksono, and E. Yudaningtyas, "Linear quadratic regulator and pole placement for stabilizing a cart inverted pendulum system" Bulletin of Electrical Engineering and Informatics, vol. 9, no. 3, pp. 914–923, 2020.

[9] M. R. Bageant, "Balancing a two-wheeled segway robot" Thesis, Massachusetts Institute of Technology, 2011.

[10] F. Grasser, A. D'arrigo, S. Colombi, and A. C. Rufer, "Joe: a mobile, inverted pendulum" IEEE Transactions on industrial electronics, vol. 49, no. 1, pp. 107–114, 2002.

[11] V. Mudeng, B. Hassanah, Y. T. K. Priyanto, and O. Saputra, "Design and simulation of two-wheeled balancing mobile ro-bot with pid controller" International Journal of Sustainable Transportation, vol. 3, no. 1, pp. 12–19, 2020.

[12] H. Nabil, "Supervised neural network control of real-time two wheel inverted pendulum" Journal of Advanced Engineering Trends, vol. 38, no. 2, pp. 131–146, 2020.

[13] O. E. Aburas and A. H. Ahmed, "A guide to implement a two wheeled robot using pole-placement on arduino" The International Journal of Engineering and Information Technology (IJEIT), vol. 6, 2020.

[14] M. Velazquez, D. Cruz, S. Garcia, and M. Bandala, "Velocity and motion control of a self-balancing vehicle based on a cascade control strategy" International Journal of Advanced Robotic Systems, vol. 13, Apr. 2016.

[15] H. Hellman and H. Sunnerman, "Two-wheeled self-balancing robot," Thesis, KTH Royal Institute of Technology, 2015.

[16] C. Dengler and B. Lohmann, "Adjustable and adaptive control for an unstable mobile robot using imitation learning with trajectory optimization" Robotics, vol. 9, no. 2, p. 29, 2020.

[17] R. P. M. Chan, K. A. Stol, and C. R. alkyard, "Review of modelling and control of two-wheeled robots" Annual Reviews in Control, vol. 37, pp. 89–103, Apr. 2013.

[18] J. Morantes, D. Espitia, O. Morales, R. Jiménez, and O. Avi-les, "Control system for a segway" International Journal of Applied Engineering Research (IJAER), vol. 13, 2018.

[19] D. M Abd-elaziz, "Conventional fuzzy logic controller for ba-lancing two-wheel inverted pendulum" Journal of Advanced Engineering Trends, vol. 38, no. 2, pp. 107–119, 2020.

[20] M. Moness, D. Mahmoud, and A. Hussein, "Real-time mamdani-like fuzzy and fusion-based fuzzy controllers for balancing two-wheeled inverted pendulum" Journal of Am-bient Intelligence and Humanized Computing, pp. 1–17, 2020.

[21] T. Johnson, S. Zhou, W. Cheah, W. Mansell, R. Young, and S. Watson, "Implementation of a perceptual controller for an inverted pendulum robot," Journal of Intelligent & Robotic Systems, pp. 1–10, 2020.

[22] Pololu, "Gearmotor 4752" Accedido en 01-10-2019 a https://www.pololu.com/product/4752/specs, 2019.