ITESO, Universidad Jesuita de Guadalajara

Práctica de Navegación Autónoma Híbrida (ArUco + LIDAR)

Control híbrido basado en visión cenital y evasión LIDAR

Luis Eduardo Diaz Macias, Pablo Perez Sanchez, Miguel de Jesus Flores Gonzalez & Jesus Alejandro Ocegueda Melin

Título de la práctica	Práctica: Navegación Híbrida
Autor o Responsable	Luis Eduardo Diaz Macias, Pablo Perez Sanchez, Miguel de Jesus Flores Gonzalez & Jesus Alejandro Ocegueda Melin
Asignatura	PAP
Fecha de entrega	13 de noviembre de 2025
Objetivo	Implementar un controlador autónomo híbrido en el cual la navegación global se obtiene con ArUco y la evasión inmediata se basa en un sensor LIDAR comunicado vía Firebase.
Materiales y Recursos	1. **Equipo de Cómputo y Visión:** 2. Cámara web con cable USB 3. Trípode o estructura de montaje 4. Códigos ArUco (4 para esquinas, 1 para robot) 5. Computadora (PC) con Python 6. Código de interfaz (`rebote.py`) 7. **Componentes del Robot:** 8. Carrito del laboratorio (omnidireccional o diferencial) 9. Sensor LIDAR (ej. RPLIDAR) 10. Microcontrolador con WiFi (ej. Raspberry Pi, ESP32) 11. Script de robot (para leer LIDAR y conectarse a Firebase)
Duración Estimada	8 horas
Lugar o Modalidad	Laboratorio de Mecatrónica

Resumen— Esta práctica introduce al estudiante en la configuración de un sistema de navegación autónoma híbrido. Se centra en un movimiento continuo (rebote) dentro de un área delimitada por ArUco, con una capa de evasión de obstáculos en tiempo real proporcionada por un sensor LIDAR y comunicada a través de Firebase.

---

1. Introducción

A diferencia de la navegación por objetivos (Práctica 02), este sistema explora la navegación autónoma continua. El objetivo es crear un robot que pueda patrullar un área de forma indefinida sin intervención humana. Para lograrlo, se implementa un controlador híbrido que fusiona dos fuentes de datos:

- **Visión Global (ArUco):** Una cámara cenital define un çorral"virtual, actuando como un sistema de posicionamiento absoluto que evita que el robot se escape del área.
- **Percepción Local (LIDAR):** Un sensor a bordo del robot detecta obstáculos dinámicos e imprevistos (personas, objetos) que la cámara no puede ver.

El script `rebote.py` actúa como el cerebro central, recibiendo datos del LIDAR vía Firebase y fusionándolos con los datos de la visión ArUco para enviar comandos de movimiento.

2. Capítulo 0: Configuración del Sistema

El sistema requiere dos componentes de software que se comunican a través de Firebase:

- **Interfaz de Control (`rebote.py`):** Se ejecuta en la computadora (PC) y actúa como el çerebroçentralizado.
- **Código del Carrito (en el robot):** Se ejecuta en el microcontrolador del robot.

Configuración de entorno para la interfaz (PC)

Es necesario instalar las dependencias de Python que requiere `rebote.py`:

Instalación de librerías (PC)

```
pip install opencv-python numpy pillow firebase-admin
```

Estas librerías cumplen las siguientes funciones:

- **opencv-python:** Procesamiento de imágenes, detección de ArUco y homografía.
- **numpy:** Cálculos numéricos y matriciales.
- **pillow:** Manejo de imágenes para la interfaz gráfica (Tkinter).
- **firebase-admin:** Conexión con la base de datos en tiempo real de Firebase.

Configuración del Código del Carrito

El script `rebote.py` es un **controlador**. Espera recibir datos del LIDAR y envía comandos de motor a través de Firebase. El script del carrito (no proporcionado en esta práctica) debe realizar dos tareas:

1. **Escuchar Instrucciones:** Debe suscribirse a la ruta `.../instrucciones` para recibir comandos de movimiento (`vx`, `vy`, `w`) y detención (`parar`).
2. **Publicar Datos del LIDAR:** Debe leer su sensor LIDAR y publicar los resultados en la ruta `.../lidar` en el siguiente formato JSON:

```
{
        "obstaculo_detectado": true,
        "distancia_minima": 350,
        "direccion_libre": "left"
}
```

Sincronización de Rutas

El script `rebote.py` y el script del carrito deben compartir la **misma ruta base** en Firebase (ej. `robots/123456`). Si las rutas no coinciden, la comunicación fallará.

3. Capítulo 1: Primer acercamiento a la interfaz

Al iniciar `rebote.py`, se despliega una ventana principal dividida en dos secciones. A diferencia de la interfaz de la Práctica 02, esta es más simplificada y enfocada en el monitoreo autónomo.

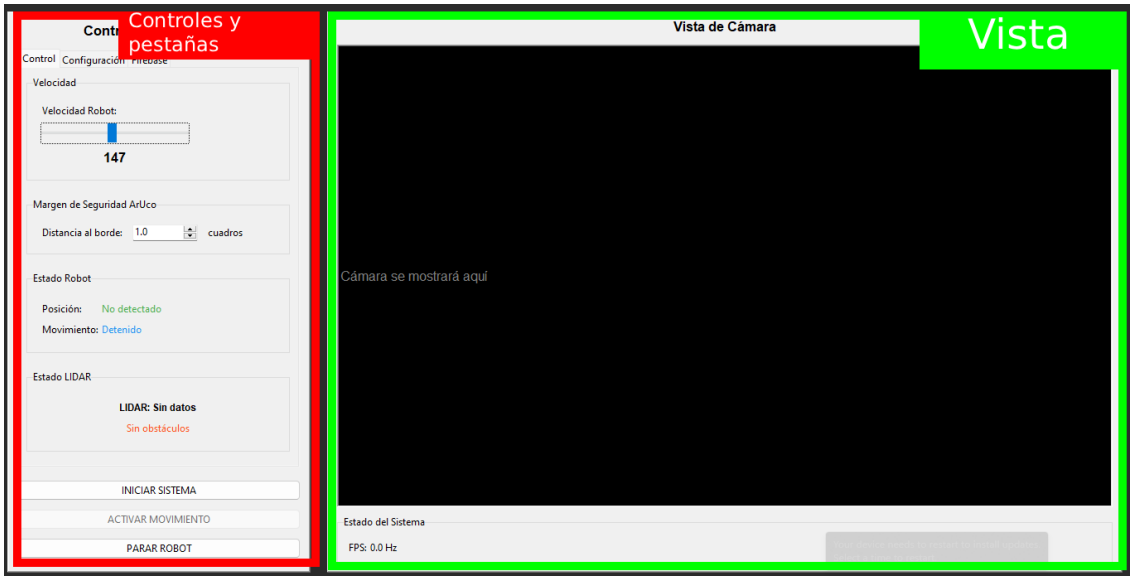

Figura 1: Interface.

Panel Izquierdo (Control y Configuración)

Contiene los botones de acción principales y tres pestañas para la configuración:

- **Botones Principales:**
  - `INICIAR SISTEMA`: Activa la cámara y el procesamiento de visión.
  - `ACTIVAR MOVIMIENTO`: Inicia la lógica de rebote autónomo.
  - `PARAR ROBOT`: Envía un comando de detención de emergencia.
- **Pestañas:**
  - `Control`: Ajustes en tiempo real (velocidad, margen).
  - `Configuración`: Ajustes de hardware (cámara, IDs de ArUco).
  - `Firebase`: Parámetros de conexión a la base de datos.

Panel Derecho (Vista de Cámara)

Dedicado a la visualización en tiempo real. Aquí se observa la imagen de la cámara, la malla verde (si las 4 esquinas son detectadas), el borde de seguridad naranja, y la detección del robot (círculo y flecha).

4. Capítulo 2: Pestañas de Configuración

Pestaña “Control”

Agrupa los parámetros de comportamiento en tiempo real.

Velocidad

- **Velocidad Robot**: Deslizador (50-250) que define la velocidad base del robot durante el movimiento autónomo.

Margen de Seguridad ArUco

- **Distancia al borde**: Define cuántos “cuadros” de la malla (0.5-3.0) se usarán como búfer. Si el robot entra en esta zona, se activará el “rebote”.

Estado Robot y LIDAR

- **Estado Robot**: Monitorea la posición (x, y) y el ángulo del robot en la malla, y si el movimiento está "Activo" o "Detenido".
- **Estado LIDAR**: Muestra los datos recibidos desde Firebase (Sin datos, Camino libre, Obstáculo) y la acción de evasión sugerida.

Pestaña "Configuración"

Define los parámetros físicos y de hardware del sistema.

Cámara

- **Índice, Ancho, Alto, FPS**: Parámetros estándar de captura de video (ej. Índice 1, 1920×1080, 30 FPS).

Marcadores ArUco

- **Robot ID**: El ID del marcador sobre el robot (ej. 4).
- **Esquinas**: Los 4 IDs que definen el área (ej. 1, 2, 5, 3).
- **Tamaño Malla**: La granularidad de la cuadrícula virtual (ej. 10 para una malla de 10×10).

Archivo de Configuración

- **Guardar / Cargar**: Guarda todos estos parámetros en un archivo `config.json` para no tener que reconfigurarlos en cada sesión.

Pestaña Firebase

Gestiona la conexión con la base de datos en la nube.

Configuración de Conexión

- **URL**: La dirección URL de tu base de datos Firebase.
- **Ruta**: La ruta base del robot (ej. `robots/123456`).
- **Credenciales**: La ruta al archivo `cred.json` descargado de Firebase.

Estado de Conexión

- **Botón Conectar / Desconectar**: Inicia o detiene la comunicación.
- **Indicador de Estado**: Muestra Desconectado, Conectado o Error.

5. Capítulo 3: Acomodo del escenario

Previo a la ejecución, es indispensable preparar el entorno de trabajo.

1. Monte la cámara en una posición elevada (cenital), utilizando el trípode o una estructura, apuntando directamente hacia el suelo.
2. Coloque los cuatro códigos ArUco de **Esquinas** en el suelo, formando un rectángulo o cuadrado que defina el área de operación.
3. Asegúrese de que los cuatro marcadores sean claramente visibles en la vista de la cámara.
4. Coloque el código ArUco de **Robot ID** sobre el centro del chasis del carrito, visible para la cámara.

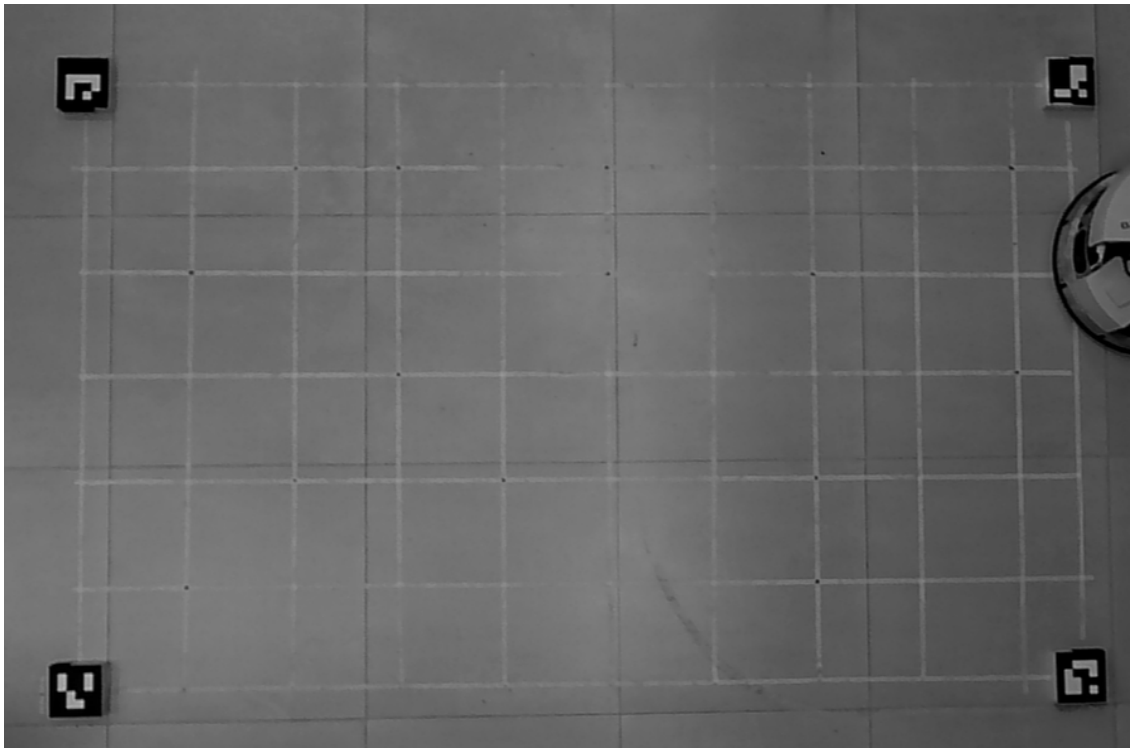

Figura 2: Códigos en escenario.

6. Capítulo 4: Secuencia de Inicio del Sistema Híbrido

Para operar el sistema, se debe seguir una secuencia de arranque específica para asegurar que todos los componentes se comuniquen. El orden es crítico.

Conexión a Firebase y Verificación del LIDAR

El primer paso es establecer la comunicación.

1. Ejecute el script `rebote.py` en la PC.
2. Navegue a la pestaña **Firebase** y verifique que todos los campos (URL, Ruta, Credenciales) sean correctos.
3. Presione el botón **Conectar**. El estado debe cambiar a Conectado.
4. **Ahora, encienda el carrito** y ejecute su script (el que lee el LIDAR y se conecta a Firebase).

Verificación Crucial del LIDAR

Observe la pestaña **Control** en la PC. El **Estado LIDAR** debe cambiar de LIDAR: Sin datos a LIDAR: Camino libre.
Pruebe el LIDAR: Acerque una mano al sensor del robot. El estado en la PC debe cambiar a "LIDAR: Obstáculo...". Si esto no ocurre, la comunicación no está funcionando y no debe proceder.

Inicio del Sistema de Visión (Cámara)

Una vez confirmada la comunicación del LIDAR, active la cámara.

1. Navegue a la pestaña **Configuración** y verifique que los IDs de ArUco y los parámetros de la cámara sean correctos.
2. Presione el botón principal **INICIAR SISTEMA**.

3. **Verificación Visual:** El panel derecho (Vista) debe mostrar el video. Si los 4 marcadores de esquina son detectados, se dibujará una **malla verde** y un **borde de seguridad naranja**.
4. Coloque el robot dentro del área. Un círculo y una flecha deben aparecer sobre él, y el campo **Posición** (en la pestaña Control) debe mostrar sus coordenadas.

7. Capítulo 5: Prueba de Movimiento Autónomo Híbrido

El objetivo de esta práctica es validar la lógica de control jerárquica: la evasión de LIDAR (Prioridad 1) debe anular el rebote de ArUco (Prioridad 2).

Activación del Movimiento Autónomo

Una vez que la cámara está activa, Firebase está conectado y el robot es detectado, presione el botón **ACTIVAR MOVIMIENTO**.

- **Observación:** El estado de Movimiento cambiará a Activo. El robot debe comenzar a moverse en una dirección aleatoria inicial.

Prueba 1: Evasión de Límites (ArUco / Rebote")

Verifica que el robot permanece dentro del corral virtual.

1. Deje que el robot navegue libremente hasta que se aproxime a uno de los bordes.
2. Observe el video: cuando el robot toque la **línea naranja** (el margen de seguridad), se activa la lógica de rebote.
3. **Comportamiento Esperado:** El robot debe calcular una nueva dirección aleatoria apuntando *lejos* de ese borde y continuar su movimiento.

Prueba 2: Evasión de Obstáculos (LIDAR / .ᴱᵛasión")

Verifica que el LIDAR tiene prioridad sobre el movimiento de rebote.

1. Mientras el robot se mueva por el **centro** del área (lejos de los bordes naranjas), coloque un obstáculo físico (ej. un libro) en su trayectoria.
2. **Observación (GUI):** El **Estado LIDAR** debe cambiar a Obstáculo y mostrar la dirección de evasión (ej. "Girar left").
3. **Comportamiento Esperado:** El robot debe **ignorar** su trayectoria de rebote actual e iniciar un "modo de evasión". Girará en la dirección indicada por el LIDAR durante un tiempo fijo (aprox. 1.5s) para esquivar el objeto.
4. Una vez completada la maniobra, el robot reanudará su navegación aleatoria.

Prueba Clave: Jerarquía de Control

Esta prueba es la más importante. Demuestra que el sistema fusiona los datos correctamente: el sensor local (LIDAR) tiene prioridad sobre el plan de navegación global (ArUco) para garantizar la seguridad.

Botón de Emergencia

En cualquier momento, presione **PARAR ROBOT**. Esto debe enviar un comando de detención inmediata a Firebase, y el robot debe detenerse por completo.

8. Capítulo 6: Conclusiones y Desafíos Futuros

Conclusiones del Aprendizaje

Al finalizar esta práctica, se han integrado con éxito los componentes de un sistema de navegación autónoma híbrido y robusto.

- **Integración de Subsistemas:** Se comprobó que una arquitectura de controlador centralizado (PC) puede fusionar datos de visión (ArUco) y datos de sensores a bordo (LIDAR) utilizando Firebase como puente de comunicación en tiempo real.
- **Percepción Híbrida:** Se comprendió la diferencia entre un sistema de **posicionamiento global** (la cámara ArUco, que sabe dónde está el robot en el "mapa") y un sistema de **percepción local** (el LIDAR, que detecta obstáculos inmediatos no mapeados).
- **Control Jerárquico:** Se validó una lógica de control de dos niveles. La **Prioridad 1 (Seguridad/LIDAR)** anula a la **Prioridad 2 (Navegación/ArUco)**, asegurando que el robot responda primero a amenazas inmediatas antes de continuar con su tarea.

Desafíos y Próximos Pasos

- **Fusión con Movimiento por Objetivo:** ¿Cómo modificaría el código `rebote.py` para combinar esta evasión LIDAR con el "movimiento por selección de objetivo"(control PI) de la práctica anterior? El desafío es que el robot navegue a un punto específico *mientras* esquiva obstáculos dinámicos en su camino.
- **Robustez de Conexión:** El script `rebote.py` tiene un `lidar_timeout` de 2.0 segundos. Pruebe qué sucede si apaga el robot (dejando de enviar datos LIDAR) mientras está en movimiento autónomo. ¿Cómo reacciona la PC pasados 2 segundos?
- **Control Descentralizado:** Actualmente, la PC toma todas las decisiones. ¿Cómo se modificaría el sistema si el robot tomara sus propias decisiones de evasión LIDAR (control local) y la PC solo le diera comandos de alto nivel como "patrulla el área¿