



Campus Puebla

Materia

Integración de Robótica y Sistemas Inteligentes TE3003B

Challenge VII

Integrantes

José Jezarel Sánchez Mijares A01735226

Antonio Silva Martínez A01173663

Dana Marian Rivera Oropeza A00830027

Fecha: 24 de mayo de 2024

Contenidos

Resumen	2
Objetivos	2
Videos	2
Conclusiones	2
Referencias	2

Resumen

En este reporte hablaremos sobre el proceso para detectar los Aruco markers y obtener la posición a la que nos encontramos de estos, algo que, en entregas futuras nos ayudarán a establecer filtros de Kalman y tener un mejor mapeo del mapa. Para esto se realizó un proceso de calibración de la cámara para poder realizar la detección de los arucos en el robot real

Objetivos

- Calibración de la cámara
- El usuario debe desarrollar un nodo de información de la cámara para publicar los parámetros de la cámara obtenidos mediante el proceso de calibración
- Detección de aruco markers

Introducción

Un marcador ArUco es un marcador sintético cuadrado compuesto por un borde negro ancho y una matriz binaria interna que determina su identificador (id). El borde negro facilita su rápida detección en la imagen y la codificación binaria permite su identificación y la aplicación de técnicas de detección y corrección de errores. El tamaño del marcador determina el tamaño de la matriz interna. Por ejemplo, un marcador de tamaño 4x4 está compuesto por 16 bits.

Los ArUco markers pueden tener distintas aplicaciones en el ámbito de la robótica. En la robótica móvil, pueden ser colocados en puntos clave para ayudar a los robots a determinar una posición más precisa y mejorar su navegación autónoma. Estos marcadores actúan como referencias visuales que permiten a los robots corregir su trayectoria y evitar obstáculos con mayor exactitud. Además, se utilizan en la calibración de cámaras y otros sensores, proporcionando un patrón de referencia conocido que facilita el ajuste de los parámetros de los sensores.

En el contexto de mapeo y localización (SLAM - Simultaneous Localization and Mapping), los ArUco markers son extremadamente útiles. Actúan como puntos de referencia fiables que ayudan a construir y actualizar mapas del entorno en tiempo real. Los robots pueden detectar estos marcadores para corregir desviaciones en su trayectoria y mejorar la precisión de su mapeo y localización. En entornos colaborativos, donde varios robots trabajan juntos, los ArUco markers permiten coordinar y compartir información de mapeo, asegurando que todos los robots tengan una comprensión común del entorno.

Solución del challenge

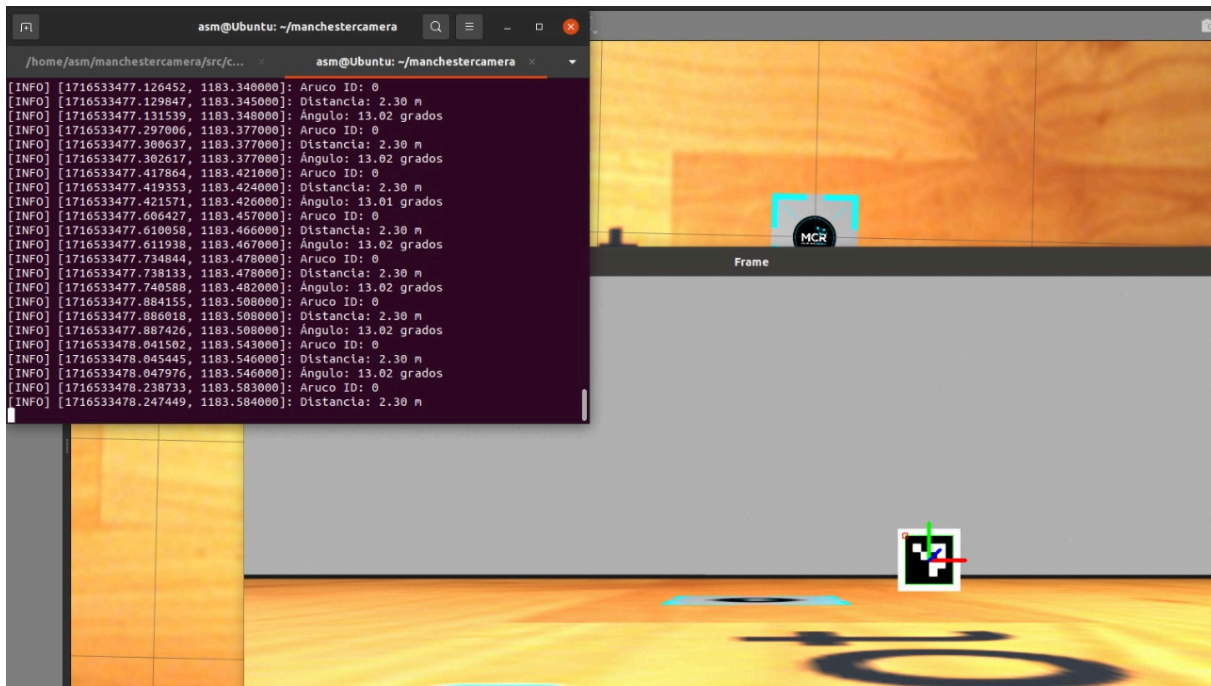


Imagen 1. Visualización de ArUco markers usando Gazebo

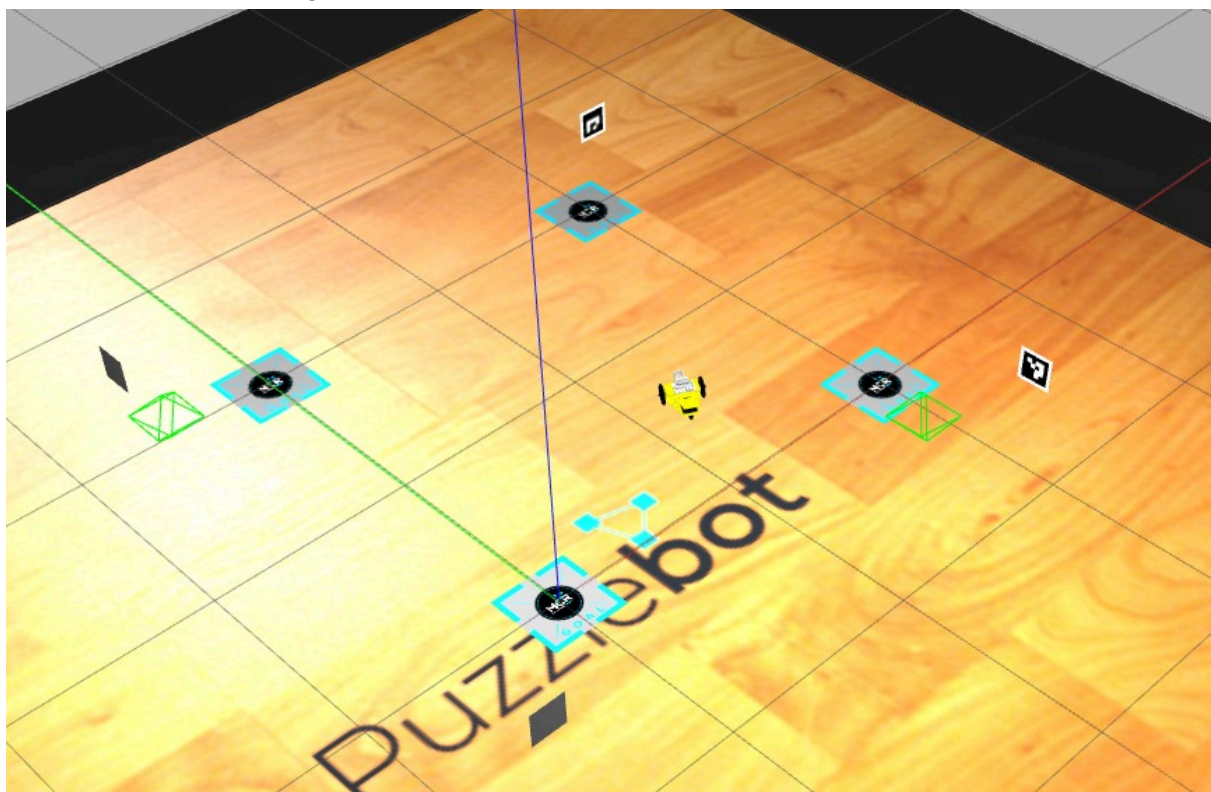
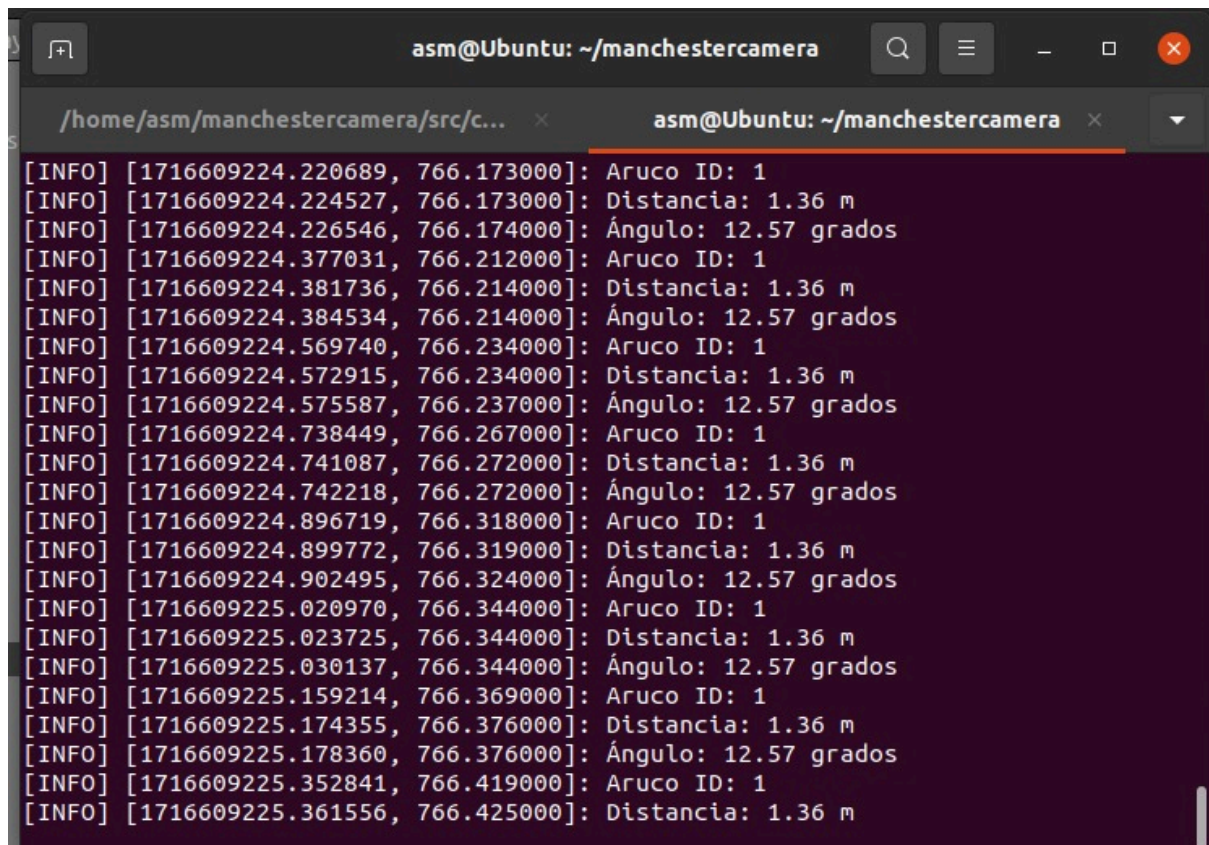


Imagen 2. Visualización del robot en el escenario con ArUcos

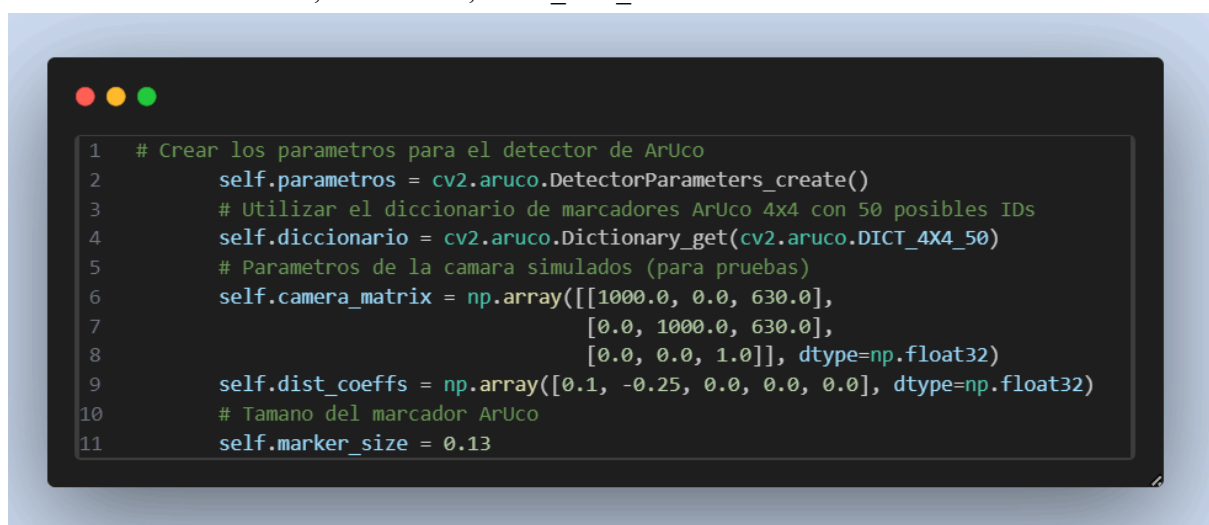
A terminal window titled 'asm@Ubuntu: ~/manchestercamera' displays a series of log messages. Each message is an '[INFO]' line containing a timestamp, a pair of coordinates, and a label. The labels alternate between 'Aruco ID: 1' and distance/angle measurements. The coordinates are in the format [x, y] where x is a long integer and y is a float. The distance measurements are 'Distancia: 1.36 m' and the angle measurements are 'Ángulo: 12.57 grados'.

```
[INFO] [1716609224.220689, 766.173000]: Aruco ID: 1
[INFO] [1716609224.224527, 766.173000]: Distancia: 1.36 m
[INFO] [1716609224.226546, 766.174000]: Ángulo: 12.57 grados
[INFO] [1716609224.377031, 766.212000]: Aruco ID: 1
[INFO] [1716609224.381736, 766.214000]: Distancia: 1.36 m
[INFO] [1716609224.384534, 766.214000]: Ángulo: 12.57 grados
[INFO] [1716609224.569740, 766.234000]: Aruco ID: 1
[INFO] [1716609224.572915, 766.234000]: Distancia: 1.36 m
[INFO] [1716609224.575587, 766.237000]: Ángulo: 12.57 grados
[INFO] [1716609224.738449, 766.267000]: Aruco ID: 1
[INFO] [1716609224.741087, 766.272000]: Distancia: 1.36 m
[INFO] [1716609224.742218, 766.272000]: Ángulo: 12.57 grados
[INFO] [1716609224.896719, 766.318000]: Aruco ID: 1
[INFO] [1716609224.899772, 766.319000]: Distancia: 1.36 m
[INFO] [1716609224.902495, 766.324000]: Ángulo: 12.57 grados
[INFO] [1716609225.020970, 766.344000]: Aruco ID: 1
[INFO] [1716609225.023725, 766.344000]: Distancia: 1.36 m
[INFO] [1716609225.030137, 766.344000]: Ángulo: 12.57 grados
[INFO] [1716609225.159214, 766.369000]: Aruco ID: 1
[INFO] [1716609225.174355, 766.376000]: Distancia: 1.36 m
[INFO] [1716609225.178360, 766.376000]: Ángulo: 12.57 grados
[INFO] [1716609225.352841, 766.419000]: Aruco ID: 1
[INFO] [1716609225.361556, 766.425000]: Distancia: 1.36 m
```

Imagen 3. Visualización de los datos de posición y id enviados por el nodo

Funcionamiento del código:

El código implementa un detector de marcadores ArUco utilizando ROS y OpenCV. Primero, se define la clase `ArucoDetector`, en donde se encarga de inicializar y gestionar la detección de los marcadores, posteriormente se configuran las herramientas necesarias para la conversión de imágenes de ROS a OpenCV mediante `CvBridge`, y se establecen los parámetros del detector de ArUco y el diccionario de marcadores, en este caso, `DICT_4X4_50`.

A code editor window shows the initialization of the `ArucoDetector` class. The code includes comments in Spanish and sets up parameters for the Aruco detector, including the dictionary, camera matrix, distortion coefficients, and marker size.

```
1 # Crear los parametros para el detector de Aruco
2 self.parametros = cv2.aruco.DetectorParameters_create()
3 # Utilizar el diccionario de marcadores Aruco 4x4 con 50 posibles IDs
4 self.diccionario = cv2.aruco.Dictionary_get(cv2.aruco.DICT_4X4_50)
5 # Parametros de la camara simulados (para pruebas)
6 self.camera_matrix = np.array([[1000.0, 0.0, 630.0],
7                                [0.0, 1000.0, 630.0],
8                                [0.0, 0.0, 1.0]], dtype=np.float32)
9 self.dist_coeffs = np.array([0.1, -0.25, 0.0, 0.0, 0.0], dtype=np.float32)
10 # Tamano del marcador Aruco
11 self.marker_size = 0.13
```

Se define la matriz intrínseca de la cámara (`camera_matrix`) y los coeficientes de distorsión (`dist_coeffs`), ambos obtenidos a partir de un proceso de calibración de la cámara. Además, se especifica el tamaño físico de los marcadores ArUco a detectar (`marker_size`). El nodo de ROS se

inicializa con el nombre `aruco_detector`, y se suscribe al tópic `puzzlebot_1/camera/image_raw` para recibir imágenes. El método `rospy.spin()` mantiene el nodo activo y en espera de nuevas imágenes.

```
1 # Convertir la imagen a escala de grises
2 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
3 # Detectar los marcadores ArUco en la imagen
4 esquinas, ids, _ = cv2.aruco.detectMarkers(gray, self.diccionario, parameters=self.
  parametros)
5 if ids is not None:
6     # Dibujar los marcadores detectados en la imagen
7     frame = cv2.aruco.drawDetectedMarkers(frame, esquinas)
8     # Estimar la pose de los marcadores
9     rvecs, tvecs, _ = cv2.aruco.estimatePoseSingleMarkers(esquinas, self.marker_size,
  self.camera_matrix, self.dist_coeffs)
10    # Iterar sobre cada marcador detectado
11    for i in range(len(ids)):
12        aruco_id = ids[i][0] # ID del marcador ArUco
13        rvec = rvecs[i][0]   # Vector de rotacion
14        tvec = tvecs[i][0]   # Vector de traslación
```

El método `image_callback` se encarga de procesar cada imagen recibida. Primero, convierte la imagen del formato ROS al formato OpenCV y la convierte a escala de grises. Utilizando el diccionario y los parámetros previamente configurados, se detectan los marcadores ArUco en la imagen. Si se detectan marcadores, se dibujan los recuadros alrededor de ellos y se estima su pose, obteniendo vectores de rotación (`rvecs`) y traslación (`tvecs`). A partir de los vectores de traslación, se calcula la distancia y el ángulo hacia cada marcador, mostrando esta información en la consola. Además, se dibujan los ejes del marcador en la imagen para finalmente mostrar la imagen procesada en una ventana de OpenCV.

```
1 # Calcular la distancia al marcador usando la norma del vector de traslación
2 distance = np.linalg.norm(tvec)
3 # Calcular el ángulo del marcador usando arctan2
4 angle = np.arctan2(tvec[0], tvec[2])
5 angle_degrees = np.degrees(angle) # Convertir el ángulo a grados
6 # Mostrar información del marcador en la consola
7 rospy.loginfo(f"Aruco ID: {aruco_id}")
8 rospy.loginfo(f"Distancia: {distance:.2f} m")
9 rospy.loginfo(f"ángulo: {angle_degrees:.2f} grados")
10
11 # Dibujar los ejes del marcador en la imagen
12 cv2.aruco.drawAxis(frame, self.camera_matrix, self.dist_coeffs, rvec, tvec, 0.1)
13 else:
14     # Mostrar un mensaje si no se detectaron marcadores
15     rospy.loginfo("No se detectaron Arucos")
16 # Mostrar la imagen resultante en una ventana de OpenCV
17 cv2.imshow('Frame', frame)
18 cv2.waitKey(1)
```

Video de resultados:

https://youtu.be/kkREf1rz3_8?feature=shared

Conclusiones

Durante este mini challenge nos dedicamos en paralelo en el desarrollo de los dos distintos tipos de enfoques, tanto en el desarrollo de un código que funciona en opencv, cómo en el desarrollo del nodo de ros para detección de arucos, el desarrollo de este challenge utilizamos el aproach uno ya que fue más fácil la visualización e implementación de estos landmarks. Sin embargo el aproach 2 facilita la implementación de transformadas lo que posteriormente nos ayudaría bastante cuando implementemos los filtros de Kalman, lo que a su vez nos permitiría tener una mejor localización de nuestro robot, es decir una implementación de SLAM

Referencias

OpenCV: Detection of ArUco Markers. (s. f.).

https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html