Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Puebla

TE3003B.501

Integración de robótica y sistemas inteligentes

Reto Semanal 7 | Manchester Robotics

Frida Lizett Zavala Pérez A01275226

Diego Garcia Rueda A01735217

Alejandro Armenta Arellano A01734879

24 de Mayo del 2024

Índice

Resumen

En este reto se realizó la calibración de la cámara para su correcto funcionamiento en la Jetson en conjunto con ROS, para realizar la tarea de detección de ArUco Markers.

Objetivos

Con este reto se busca añadir un eslabón más de los requerimientos para el reto final, el cual es la navegación y localización a partir de la detección de señales visuales, por lo que se hizo uso de la visión por computadora y ArUco markers.

Introducción

Se está desarrollando un robot móvil capaz de llegar a un punto deseado de manera autónoma, aplicando técnicas de navegación y localización que se complementan entre sí. En el reto actual se incorpora al sistema la localización basada en la detección de marcadores ArUco. Para esta solución es indispensable identificar la posición de los marcadores con respecto a un sistema de coordenadas común.

Como punto de partida se generó el algoritmo de detección de ArUcos en un nodo de Ros, usando OpenCv, con la finalidad de ser añadido al sistema general de navegación que se ha construido con los retos previos.

Aruco Markers

Un marcador ArUco es un marcador sintetico cuadrangular compuesto por un borde negro y una matriz binaria interna que determina un identificador. El tamaño del marcador determina el tamaño de la matriz interna. Existen diferentes diccionarios ArUco que agrupan marcadores con distintos identificadores.

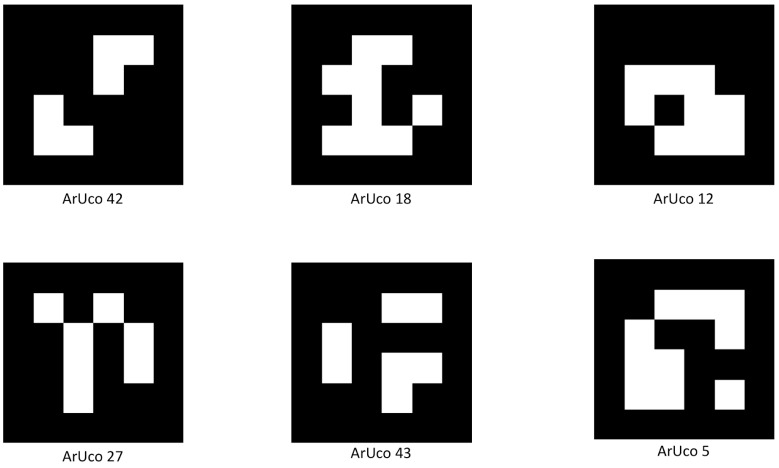


Fig 1. Marcadores ArUco.

OpenCv

Para esta solución se usó la librería OpenCv, la cual es una librería de computación visual para el procesamiento de imágenes en Python. Esta biblioteca proporciona herramientas para realizar operaciones de procesamiento de imágenes, como el filtrado, la detección de bordes, el reconocimiento de características, el seguimiento de objetos, etc. La librería nos ayuda para la calibración de la cámara y la detección de los marcadores ArUco.

Calibración de la cámara

La calibración de la cámara es el proceso de obtención de los parámetros básicos de la cámara. Estos parámetros nos permiten determinar la posición del punto 3D en el espacio proyectado sobre el sensor de la cámara. Sus parámetros se dividen en internos y externos. Los primeros describen el funcionamiento de la cámara, los segundos determinan la posición y orientación del marco de referencia de la cámara con respecto al mundo real, es decir, determinan la orientación externa de la cámara.

Las lentes de las cámaras distorsionan la escena, variando las distancias de los objetos con respecto al centro de la imágen. Es por ello que sí se desea tener con exactitud la proyección

de los diferentes puntos de la imágen se deben de contemplar los coeficientes de distorsión. Esta distorsión consta de dos componentes, uno radial y otro tangencial, ambos dependientes del ángulo y de la distancia focal. Esto se considera en los coeficientes de distorsión calculados en la calibración. Por otro lado sí lo que se busca es saber la proyección de un punto referido a un sistema de referencia arbitrario es necesario hacer uso de los parámetros extrínsecos. Estos son las rotaciones 3D, y las translaciones, las cuales son necesarias para traducir el sistema de referencia de la cámara al sistema arbitrario.

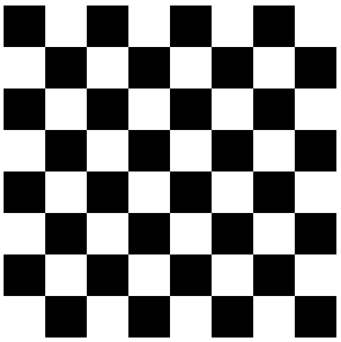


Fig 2. Tablero de calibración.

Para hacer esta calibración es necesario imprimir un tablero de calibración como el que se muestra en la figura 2. Una vez que se ha impreso el tablero, es necesario medir con la mayor precisión posible las dimensiones del tablero y se procede a tomar fotos desde diferentes ángulos y posiciones. Se toman la mayor cantidad de imágenes donde se vea el tablero completo. Tras haber tomado las fotos se ejecutan los programas para hacer la calibración y los datos se almacenan en un archivo de tipo .yaml.

Solución del problema

A continuación se explica de manera general el funcionamiento del único nodo implementado, debido a que es una solución simple es un único algoritmo. Podemos visualizar el código en anexos al final del documento.

Importación de bibliotecas: Importa las bibliotecas necesarias, incluyendo ROS para la comunicación entre nodos, OpenCV para el procesamiento de imágenes y NumPy para operaciones matemáticas.

Definición de parámetros de ArUCo: Define el tipo de marcador ArUCo a detectar y un diccionario de nombres asociados a los tipos de marcadores ArUCo soportados por OpenCV.

Inicialización de la detección de ArUCo: Verifica que el tipo de marcador ArUCo especificado esté soportado por OpenCV, carga el diccionario de ArUCo y configura los parámetros del detector.

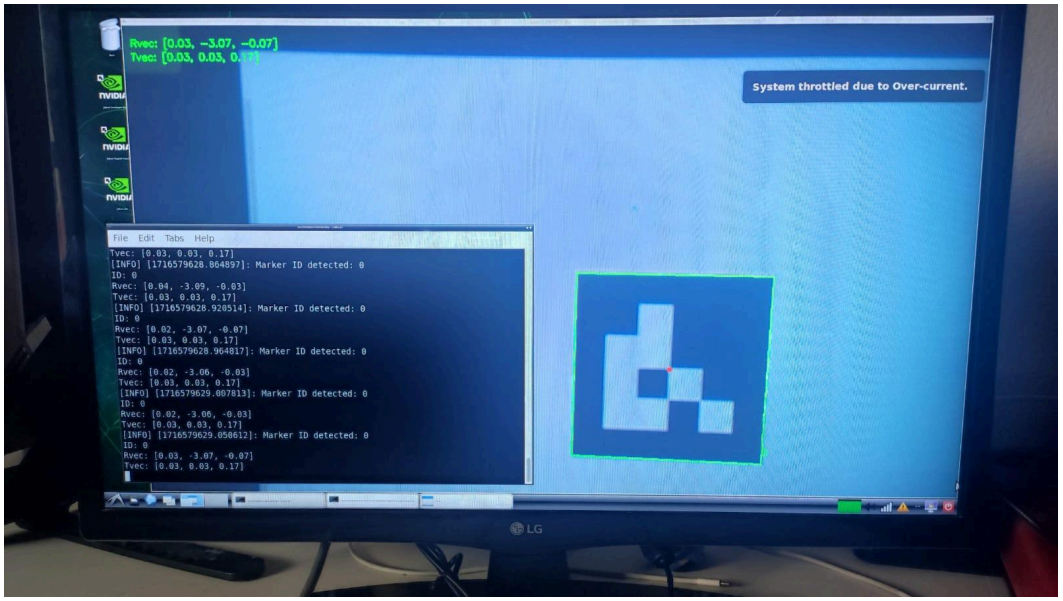
Inicialización de la clase ArucoDetector: Define una clase que encapsula la lógica de detección de ArUCo. En su inicialización, crea un objeto CvBridge para convertir entre mensajes de ROS y matrices de OpenCV, y se suscribe al tópico de la imagen de la cámara.

Método de devolución de llamada (callback): Define el método callback, que se ejecuta cada vez que se recibe una imagen de la cámara. Convierte la imagen de ROS a una matriz de OpenCV, detecta los marcadores ArUCo en la imagen, estima sus poses en el espacio 3D y visualiza los resultados en la imagen.

Bucle principal: Inicializa el nodo de ROS, crea una instancia de la clase ArucoDetector y entra en un bucle que permite que ROS maneje las suscripciones y publicaciones. Este bucle se ejecuta hasta que se interrumpe el programa (por ejemplo, con Ctrl+C).

Limpieza: Al finalizar, cierra todas las ventanas de visualización creadas por OpenCV.

Resultados



Video detectando el ArUco
https://youtu.be/FJ9IzQZ6PAY

Conclusiones

Este reto tiene una implementación que en principio no es tan complicada, sin embargo al ser la primera vez que se utiliza la cámara en la Jetson nano, las configuraciones previas pueden resultar tediosas y tardadas, además de que pueden surgir errores y problemas inesperados.

Una vez que se logró generar la detección de los marcadores esto es de suma utilidad para la localización que estamos implementando, ya que al unirlo al filtro de Kalman, el error de la posición y distancia va a ayudar a contener la covarianza en la elipse de confidencialidad.

Referencias

Rafael, R. J. B., Ramiro, J. R. M. D., & Suárez, A. S. F. M. (2018). *Sistema de seguimiento de objetos usando OpenCV, ARUCO y filtro de Kalman*

extendido [Proyecto Fin de grado Grado en Ingeniería Electrónia, Robótica y Mecatrónica]. Escuela Técnica Superior de Ingeniería Universidad de Sevilla.

Roos, S. R. H. T., Fernández, A. F. G., Álvarez, I. Á. G., & Corsino, R. C. G. R. (2019). *LOCALIZACIÓN DE ROBOTS MÓVILES EN ENTORNOS INDUSTRIALES USANDO UN ANILLO DE CÁMARAS* [Universidad de Oviedo Campus Universitario de Gijón]. https://ruc.udc.es/dspace/bitstream/handle/2183/23714/2019_Roos-Sara_Localizacion-robots-moviles-entornos-industriales-usando-anillo-camaras.pdf

Anexos

```python
#!/usr/bin/env python

import rospy
import cv2
import numpy as np
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

# Definimos el tipo de ArUCo tag a detectar
arUCo_type = "DICT_4X4_250"

# Definimos el diccionario de nombres de los ArUCo tags soportados por OpenCV
ARUCO_DICT = {
    "DICT_4X4_50": cv2.aruco.DICT_4X4_50,
    "DICT_4X4_100": cv2.aruco.DICT_4X4_100,
    "DICT_4X4_250": cv2.aruco.DICT_4X4_250,
    "DICT_4X4_1000": cv2.aruco.DICT_4X4_1000,
    "DICT_ARUCO_ORIGINAL": cv2.aruco.DICT_ARUCO_ORIGINAL,
}

# Verificamos que el ArUCo tag especificado exista y sea soportado por OpenCV
if ARUCO_DICT.get(arUCo_type, None) is None:
    rospy.loginfo("ArUCo tag '{}' no es soportado".format(arUCo_type))
    exit(0)
```

```
# Cargamos el diccionario de ArUCo
arucoDict = cv2.aruco.getPredefinedDictionary(ARUCO_DICT[arUCo_type])
arucoParams = cv2.aruco.DetectorParameters_create()

# Definimos los parámetros de la cámara (obtenidos de la calibración de
la cámara)
# Esto es solo un ejemplo, necesitas usar los valores obtenidos de tu
calibración
camera_matrix = np.array([[1000, 0, 640], [0, 1000, 360], [0, 0, 1]],
dtype=np.float32)
dist_coeffs = np.zeros((5, 1), dtype=np.float32)

class ArucoDetector:
    def __init__(self):
        self.bridge = CvBridge()
        self.image_sub =
rospy.Subscriber("/puzzlebot_1/camera/image_raw", Image, self.callback)
        rospy.loginfo("Aruco Detector Node Initialized")

    def callback(self, data):
        try:
            frame = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
            rospy.logerr("CvBridge Error: {0}".format(e))

        # Detectamos marcadores ArUCo en el frame de entrada
        corners, ids, rejected = cv2.aruco.detectMarkers(frame,
arucoDict, parameters=arucoParams)

        # Verificamos que *al menos* un marcador ArUCo haya sido
detectado
        if ids is not None and len(corners) > 0:
            # Aplanamos la lista de IDs de ArUCo
            ids = ids.flatten()

            # Estimamos la pose de cada marcador detectado
            rvecs, tvecs, _ =
cv2.aruco.estimatePoseSingleMarkers(corners, 0.05, camera_matrix,
dist_coeffs)

            # Loop sobre las esquinas detectadas de ArUCo
            for i in range(len(ids)):
                markerID = ids[i]
                rvec = rvecs[i]
                tvec = tvecs[i]
```

```
                # Imprime el ID del marcador detectado
                rospy.loginfo(f"Marker ID detected: {markerID}")

                # Extraemos las esquinas del marcador (siempre se
devuelven
                # en el orden superior-izquierda, superior-derecha,
inferior-derecha e inferior-izquierda)
                markerCorner = corners[i].reshape((4, 2))
                (topLeft, topRight, bottomRight, bottomLeft) =
markerCorner

                # Convertimos cada par de coordenadas (x, y) a enteros
                topRight = (int(topRight[0]), int(topRight[1]))
                bottomRight = (int(bottomRight[0]), int(bottomRight[1]))
                bottomLeft = (int(bottomLeft[0]), int(bottomLeft[1]))
                topLeft = (int(topLeft[0]), int(topLeft[1]))

                # Dibujamos el cuadro delimitador de la detección de
ArUCo
                cv2.line(frame, topLeft, topRight, (0, 255, 0), 2)
                cv2.line(frame, topRight, bottomRight, (0, 255, 0), 2)
                cv2.line(frame, bottomRight, bottomLeft, (0, 255, 0), 2)
                cv2.line(frame, bottomLeft, topLeft, (0, 255, 0), 2)

                # Calculamos y dibujamos las coordenadas del centro (x,
y) del marcador ArUCo
                cX = int((topLeft[0] + bottomRight[0]) / 2.0)
                cY = int((topLeft[1] + bottomRight[1]) / 2.0)
                cv2.circle(frame, (cX, cY), 4, (0, 0, 255), -1)

                # Dibujamos el ID del marcador ArUCo en el frame
                cv2.putText(frame, str(markerID),
                            (topLeft[0], topLeft[1] - 15),
                            cv2.FONT_HERSHEY_SIMPLEX,
                            0.5, (0, 255, 0), 2)

                # Extraemos la posición y la orientación
                rvec_str = "Rvec: [{:.2f}, {:.2f},
{:.2f}]".format(rvec[0][0], rvec[0][1], rvec[0][2])
                tvec_str = "Tvec: [{:.2f}, {:.2f},
{:.2f}]".format(tvec[0][0], tvec[0][1], tvec[0][2])

                print("ID: {}".format(markerID))
                print(rvec_str)
                print(tvec_str)
```

```
                # Dibujamos la posición y la orientación en el frame

                cv2.putText(frame, rvec_str, (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
                cv2.putText(frame, tvec_str, (10, 50),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

        # Mostramos el frame de salida
        cv2.imshow("Frame", frame)
        cv2.waitKey(3)

def main():
    rospy.init_node('aruco_detector_node', anonymous=True)
    ad = ArucoDetector()
    try:
        rospy.spin()
    except KeyboardInterrupt:
        rospy.loginfo("Shutting down")

    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```