trayectoria30

Class ControladorRobot

INICIO

self.contador_joint_states = 0

INICIALIZAR nodo ROS 'modo_control'

CREAR objeto MoveGroupCommander para 'niryo_arm'

ESTABLECER planificación con "PTP"

SUSCRIBIRSE al tópico 'pos_dy' para obtener posición de motores

PUBLICAR en el tópico 'cord_dy' para enviar comandos de posición

SUSCRIBIRSE al tópico 'rutina' para recibir órdenes de ejecución

PUBLICAR en el tópico 'rutina'

SUSCRIBIRSE al tópico 'joint_states' para monitorear estados articulares

self.estado_actual = None

INICIALIZAR posición del robot en "home"

self.ejecutando_rutina = False

self.rutina_corriendo = False

robot = RobotCommander()

CONECTAR con el servidor de acciones para ejecutar trayectorias

DESUSCRIBO de joint_states

FUNCIÓN `rad_bit(rad)`

CONVERTIR `rad` a valores entre 0 y 4095

RETORNAR lista de valores en bits

FUNCIÓN `grados_rad(grad)`

CONVERTIR `grad` a radianes

RETORNAR lista de valores en radianes

FUNCIÓN ‘ejecutar_trayectoria(self, joint_angles)’

Activar la bandera rutina_corriendo=True

ESTABLECER planificación "PTP"

Setear posición a moverse (joint_angle)

PLANIFICAR trayectoria con MoveIt

SI la planificación es exitosa:

SUSCRIBIRSE a `joint_states`

ENVIAR trayectoria al servidor de acciones

ESPERAR finalización de ejecución

DESUSCRIBIRSE de `joint_states`

Desactivar la bandera de rutina corriendo

FUNCIÓN `ejecutar_rutina(data) `

F = Bool(False)

SI `data` no es nulo y `ejecutando_rutina` es `False`:

Activar la bandera rutina_corriendo=True

PUBLICAR en ‘rutina’ false

OBTENER lista de puntos desde la base de datos (puntps)

Guardo los puntos en un array “lista_puntos” (FOR 155 a 169)

Creo ‘MotionSequenceRequest’ ‘sequence_request’

COMIENZO a armar “motionSequenceRequest” que tiene los datos para relizar la secuencia

INGRESO el primer punto que va a ser la posición actual del robot

CREO ‘MotionSequendeItem’ ‘item’

CREO ‘JointTrajectory’ ‘traj’

Ingreso los nombre de las articulaciones ‘traj.joint_names’

CREO ‘JointTrayectoryPoint’ ‘point ’

INGRESO la posición actual en ‘point’ (point.positions)

ASIGNO Velocidad en ese punto (point).Vacío para que tome el del controlador

ASIGNO Aceleración en ese punto (point) Vacío para que tome el del controlador

AGREGO ‘point’ en ‘traj’

CREO ‘GenericTrajectory’ ‘generic_traj’

CREAR ‘Constrainsts’ (restricciones) ‘contraint’

Creo lista vacia ‘joint_contrants’

FOR Itera sobre la lista de nombres de articulaciones

CREAR un nuevo objeto ‘JointConstraint’ ‘joint_containt’, que representará la restricción de una articulación específica

ASIGNAR el nombre de la articulación (joint_name) a la restricción

ASIGNAR la posición objetivo de la articulación (point.positions[i]).

DEFINIR tolerancia alrededor de la posición deseada(+-0.01

AGREGAR ‘joint_containt’ a la lista ’joint_contrants’

ASIGNAR la lista de restricciones ’joint_contrants’ al objeto ‘containt’

AGREGAR las restricciones (constraint) al goal_constraints del ítem

AGREGAR generic_traj como referencia en reference_trajectories.

ESPECIFICAR el planificador de movimiento a usar (PTP - "Point-To-Point").

DEFINIR el grupo de articulaciones que se moverán en esta secuencia. (‘niryo_arm’)

ESCALAR la velocidad máxima del movimiento usando ’punto['vel_esc']‘.

ESTABLECER el escalar de la aceleración (1)

INDICA la suavidad entre puntos (0, osea no hay suavidad) (‘ blend_radiun’).

AGREGAR ‘item’ a la lista ‘sequence_request’

INGRESO el resto de puntos de la rutina con un FOR

FOR con respecto a ‘lista_puntos’

Repito lo anterior pero en ves de asignar el valor actual paso los valores de los puntos de la rutina a radianes(‘angles’) (línea 231) y luego lo ingreso en ‘trah.points’(línea 232)

CREO ‘MoveGroupSequenceGoal’ ‘goal’ que representa la meta de la secuencia de movimiento del robot

Verifica si el servidor está activo (if)

Muestra un mensaje en la consola indicando que el servidor no está disponible(si entro en el if)

Detiene la ejecución de la función y sale de ella, ya que el servidor no está disponible

Asigna la solicitud de movimiento ‘ sequence_request ‘ al objetivo ‘ goal ‘.

SUSCRIBIR al tópico ‘ joint_satates ’

Cancelar cualquier objetivo previo en ejecución(en el servidor).

Enviar la nueva meta al servidor ‘goal’

Esperar a que la ejecución termine.

Desuscribirse del tópico `joint_states`.

Mostrar mensaje de finalización.

Marcar que la rutina ha finalizado ejecutando_rutina=false

FUNCIÓN controlar_publicacion_cord_dy con el parámetro data.

Incrementar el contador de mensajes recibidos en joint_states.

Si el contador cumple es múltiplo del valor indicado (en este caso 1), continuar:

Convertir las posiciones de las articulaciones de radianes a bits.

Crear un mensaje ROS con los valores convertidos.

Publicar el mensaje en el tópico correspondiente. cord_dy

FUNCIÓN ejecutar_cord_ros con el parámetro data.

Crear la variable F con el valor False. F=false

Si la rutina ya está ejecutándose, salir de la función.

Marcar que la rutina ha comenzado. Ejecutar_rutina=true

Publicar el mensaje F en el tópico rutina_pub.

Inicia un bloque `try-except` para capturar errores

Convertir los valores recibidos de grados a radianes.

Ejecutar la trayectoria con los valores convertidos.

Mostrar en consola los valores usados para la trayectoria.

Marcar la rutina como finalizada. Ejecutar_rutina=false

Captura interrupciones del nodo de ROS. except

Registrar el error en consola.

Marcar la rutina como finalizada. Ejecutar_rutina=false

Comprueba si el script se está ejecutando directamente 'main'

Inicia un bloque `try-except` para capturar errores

Inicializa ROS en C++ dentro de Python.

CREAR 'ControladorRobot' 'controlador'

SUSCRIBIRSE al tópico 'cord_ros' con el callback ejecutar_cord_ros.

Mantener el nodo en ejecución con rospy.spin().

Captura interrupciones del nodo de ROS. except

Registrar en consola que la ejecución del nodo se ha interrumpido

Modo_lectura12

Class ModoLectura: