Prácticas

Robótica

(3º Ingeniería Telemática)

Rodrigo Pérez Rodríguez (rodrigo.perez@urjc.es)

José Miguel Guerrero Hernández (josemiguel.guerrero@urjc.es)

Miguel Ángel de Miguel Paraíso (miguelangel.demiguel}@urjc.es)

1. Práctica 1: Manejo del Kobuki
2. Práctica 2: Avance y parada del Kobuki
3. Práctica 3: Avance y giro del Kobuki
4. Práctica 4: Acciones
5. Práctica 5: Búsqueda y seguimiento de una pelota
6. Práctica 6: Seguimiento de una persona

Práctica 1: Manejo del Kobuki

El objetivo de esta práctica es usar los comandos de ROS 2 en la terminal para examinar el robot Kobuki.

Debes completar el siguiente cuestionario para realizar la práctica. Incluye los comandos que has usado en cada una:

1. ¿Has tenido que hacer algún setup en tu ordenador? Indica los pasos

   [Respuesta]

2. ¿Qué comando has usado para lanzar el Kobuki en ROS 2?

   [Respuesta]

3. ¿Qué nodos se lanzan?

   [Respuesta]

4. ¿Qué topics están disponibles?

   [Respuesta]

5. Analiza los topics (tipo y QoS) de /cmd_vel, /events/bumper y /scan_filtered

   [Respuesta]

6. ¿Qué servicios o acciones están disponibles?

   [Respuesta]

Práctica 2: Avance y parada del Kobuki

En esta práctica debes crear un nodo de ROS 2 que cuando se lance haga avanzar el robot Kobuki a una velocidad moderada (0.2-0.4) hasta que:

1. el bumper del robot detecte la colisión con un obstáculo, ó
2. hayan pasado 5 segundos.

También se propone que cuando el bumper esté presionado suene un sonido, y otro cuando se suelte.

Práctica 3: Avance y giro del Kobuki

En esta práctica debes crear un nodo de ROS 2 que, aplicando un FSM, haga que el robot ejecute dos movimientos de forma consecutiva:

1. Avanzar 1 metro

2. Girar PI radianes

Entonces el robot se parará y el nodo finalizará su ejecución.

Debes usar TFs (en particular odom->base_footprint) para determinar cuando el robot ha girado o avanzado suficiente.

Práctica 4: Acciones

En esta práctica debes crear 2 paquetes:

1. En un paquete debes definir los interfaces necesarios.
2. En otro paquete debes crear dos nodos:

    - **Nodo servidor:** Este nodo implementa una acción por el que se le envían comandos de avance o de giro (uno u otro, no ambos), indicando la distancia a recorrer o los radianes a girar. El nodo está latente hasta que recibe la petición. Cuando ha acabado, se para y espera una nueva petición. Si recibe una petición mientras está ejecutando otro, el nuevo comando se empieza a ejecutar inmediatamente, expulsando el anterior. El robot notifica a través de *feedback*:

        - distancia avanzada o giro realizado

        - lo que le queda para completar la petición

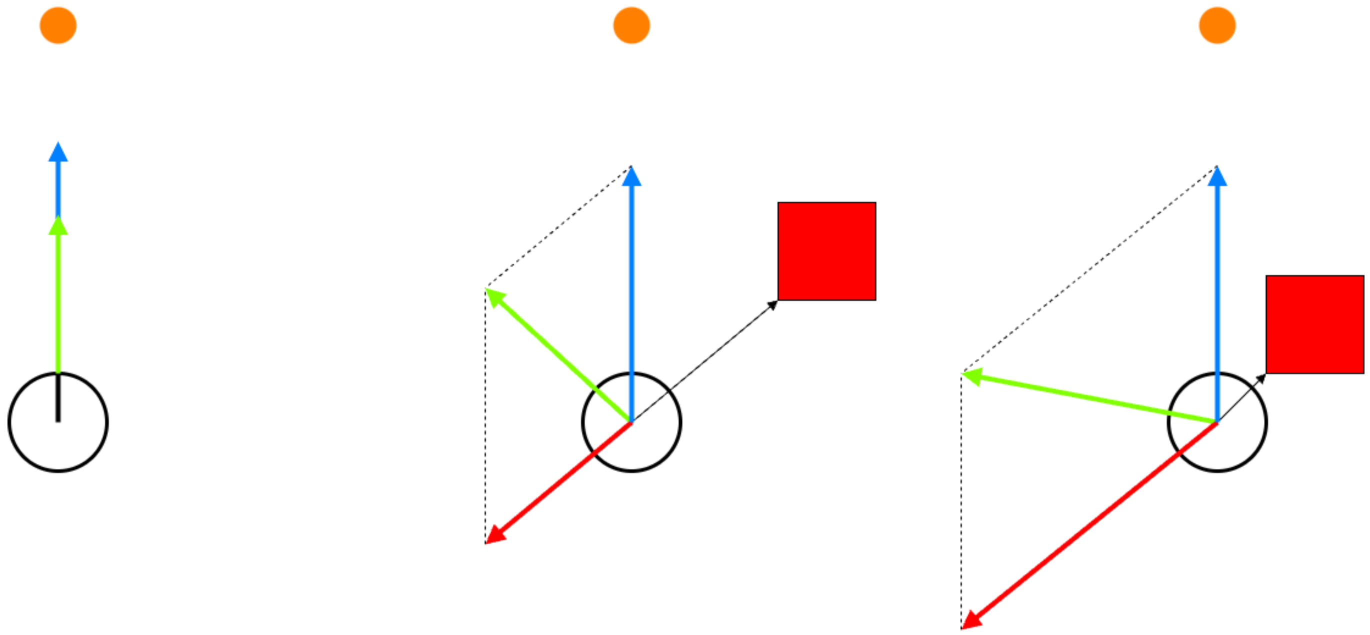
        - y el tiempo que le está llevando realizarlo

    - **Nodo cliente:** Este nodo se encarga de recibir como argumento la distancia a recorrer o los radianes a girar y hará la petición al servidor, mostrando por la consola el *feedback* recibido

3. Se deberán de crear 3 ejecutables distintos, uno para lanzar el servidor, otro para lanzar un cliente que solicite un determinado avance, y otro para lanzar un cliente con un giro dado.

Práctica 5: Búsqueda y seguimiento de una pelota

Crea una aplicación en ROS 2 que haga que un robot busque y siga una pelota de un color característico, esquivando los obstáculos que se encuentre en su camino, usando VFF.



- Debes crear un paquete que tenga un nodo que **use** la salida de los nodos que ya existen en https://github.com/URJC-teaching/asr-clase/tree/main/sensors, con ligeras modificaciones. No debes copiar ni cheros ni código a tu paquete nuevo, solo modificarlos en su ubicación original.

- La modificación de los nodos consiste en que cada uno publique los componentes necesarios de un vector atractivo, en el caso de hsv_filter o un vector repulsivo, en el caso de ObstacleDetector. Puedes usar geometry_msgs/msg/Vector3 o cualquier interfaz que te parezca adecuada (tendrás que crear una nueva interfaz de mensaje en un paquete).

- El nodo desarrollado debe tener dos estados:

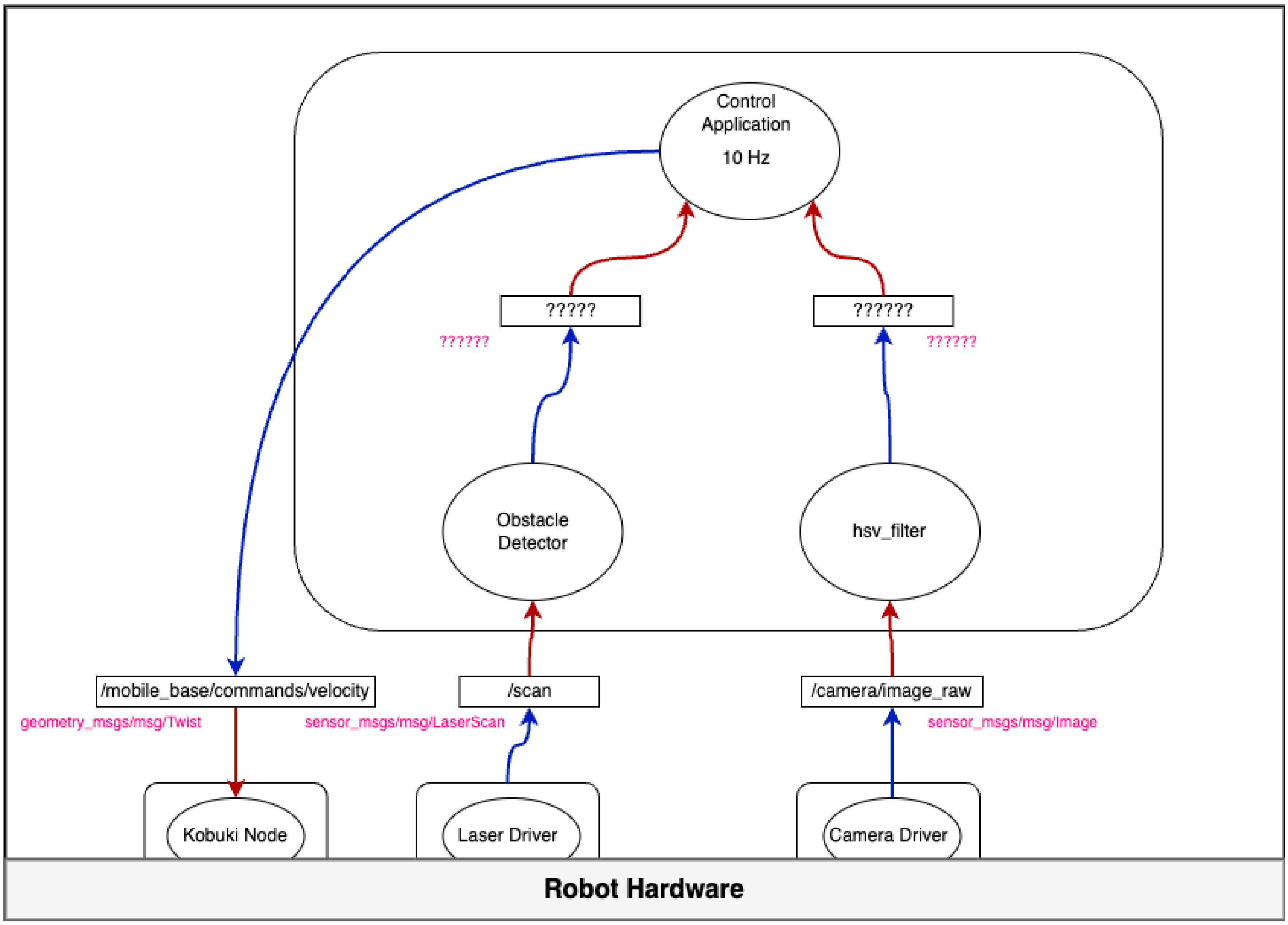
    - Uno para cuando no percibe el objeto, en el que gira hacia un lado. Por ejemplo, podría girar siempre hacia el mismo lado, o hacia el lado donde estaba la pelota antes de desaparecer.

    - Otro cuando el objeto es percibido, en el que genera velocidades usando VFF.

- Lanza los nodos utilizando launchers (no tienes por qué lanzar todos los nodos en un launcher, sino llegar a una solución que satisfaga el problema propuesto).

- Usa parameters para cualquier valor parametrizable: velocidades, valores del filtro, etc.

- Usa topics con nombre genérico y conéctalo con remapeos en el launcher. Usa launchers y/o configuraciones de parámetros diferentes cuando estés en el simulador o en el robot real.



Consejos de implementación

- El módulo del vector repulsivo no debería de ser equivalente a la distancia del obstáculo, ya que eso implicaría que los obstáculos más lejanos, causan mayor repulsión que los más cercanos. Una posible estrategia es hacerlo proporcional a K/d^2 (la constante K se puede determinar de manera experimental). Así, un objeto lejano apenas causaría repulsión, mientras que uno cercano causaría mucha repulsión (recuerda utilizar std:clamp() para limitar la velocidad de salida).

- Si quieres que obstáculos muy lejanos no causen repulsión, puedes establecer un umbral a partir del cual el obstáculo detectado no se tiene en cuenta.

- Ya que no estamos utilizando imagen de profundidad, la distancia de la pelota no es conocida, por lo que tampoco conocemos el módulo del vector atractivo. Puedes asumir que es 1 y luego multiplicarlo por una constante que adapte la velocidad de salida.

- En el robot real:

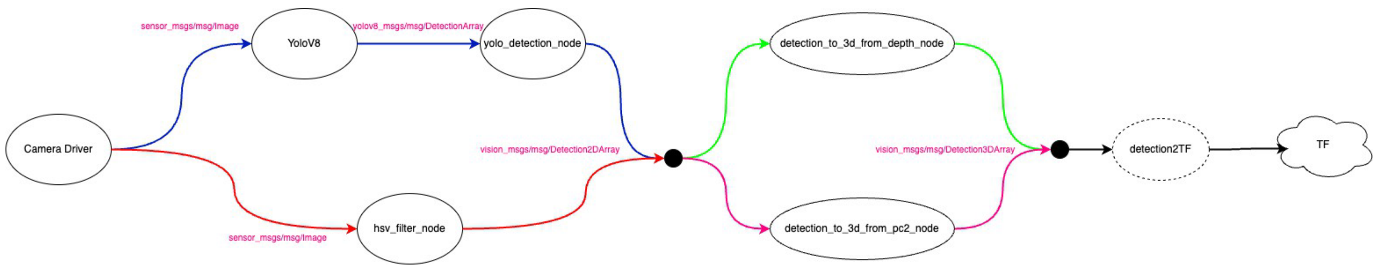
    - utiliza el topic /scan_filtered ya que tiene en cuenta algunos obstáculos fijos en el chasis del robot.

    - ten en cuenta que, debido a su colocación, los ejes del láser se encuentran invertidos. Es posible que, después de calcular las coordenadas (x,y) del obstáculo tengas que cambiarlas de signo para saber dónde está realmente.

Práctica 6: Seguimiento de una persona

Implementa un comportamiento para un robot en ROS 2 de manera que este siga a una persona (o cualquier otra entidad según un parámetro) a un metro:

1. El robot debe acercarse y mantener la distancia de 1 metro de la entidad a seguir
2. Siempre ha de orientarse hacia la entidad
3. Cuando no vea a la entidad, ha de girar hasta que la encuentra
4. Una vez encontrada, usa un PID para aproximarse a la TF que haya producido la detección
5. Opcionalmente, se puede incluir el algoritmo VFF para esquivar obstáculos



Completa el subsistema de percepción visto en clase con un nuevo nodo que produzca TFs (odom2entity) a partir de un vision_msgs/msg/Detection3DArray. Puedes usar el nodo de detección 3D que prefieras (basado en PointCloud2 o en imagen de profundidad).