Universidad de los Andes Colombia | Departamento de Ingeniería Eléctrica y Electrónica

Planeación para Manipulación

Informe Taller 3
ROBOTICS LEARNING - INTERSEMESTRAL 202319

Santiago Muñoz
Nicolás Rincón Sánchez

Para el desarrollo de este taller, se hizo uso de la Librería Robowflex, compatible con ROS, la cual contiene un conjunto de modelos de robots manipuladores con altos grados de libertad. En particular, el robot utilizado es el UR5, el cual fue programado en simulación mediante comandos de ROS y empleando el lenguaje de programación C++

1. Punto 1

En primer lugar, se programó un script en C++ que cargaba una escena generada en YAML que representaba una escena con tres cubos dispuestos sobre una mesa. Para este script nos basamos en uno preexistente que planeaba y simulaba el agarre de uno de los bloques desde arriba. Ahora, se cambió para que la planeación resultara en un agarre frontal de otro de los bloques en la mesa. A continuación, se muestra la escena modificada simulada mediante el software de Ubuntu, RVIZ. Los ejes coordenados son x (rojo), y (verde), z (azul):

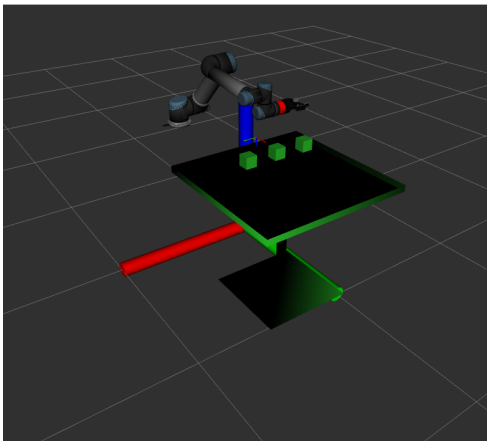

Figura 1: Escena original generada y marco de referencia global

Al ejecutar el planeador, el movimiento del manipulador es el siguiente:

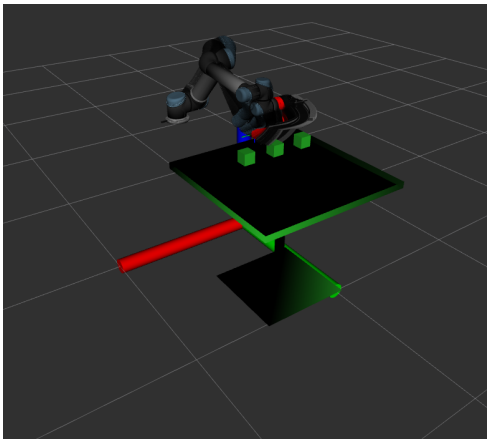

Figura 2: Trayectoria generada mediante algoritmos de planeación para agarre frontal

Finalmente, en la siguiente imagen se puede apreciar el efector final del UR5 ubicado de forma tal que el agarre sea frontal. En esta figura, se puede apreciar mejor los ejes coordenados del efector. El eje x positivo apunta hacia adelante en dirección al cubo que se va a agarrar, el eje y apunta hacia la izquierda y el eje z apunta hacia abajo. La transformación realizada en cuaterniones fue de (0.0, 0.707, 0.707, 0.0), o en ángulos de Euler en orden xyz fue de (-180, 0, -90). La rotación de -180 en x se decidió para que el robot tuviera menos probabilidades de estrellarse con la mesa debido a la composición de los últimos links del robot.

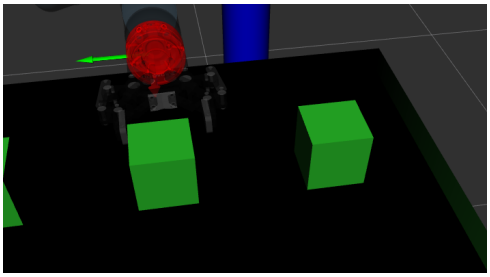

Figura 3: Posición final y marco de referencia transformado del efector final

2. Punto 2

En segundo lugar, se desarrolló un script que calculaba a partir de la trayectoria generada en planeación del punto anterior, la distancia que se desplazaba el efector final. Para ello, se calcularon dos distancias: la distancia en el espacio de configuración, donde el efector final se representa de manera puntual y la distancia en el espacio de trabajo, donde se determina a nivel físico la distancia desplazada.

Para calcular la distancia en el espacio de configuración, se utilizó la siguiente instrucción:

```
trajectory->fromYAMLFile(*ur5->getScratchState(),"ur5_block.yml");
auto length = trajectory->getLength();
std::cout << "CSpace Length: "<< length << "rad" << std::endl;
```

En el comando anterior, la instrucción ->getScratchState() arroja las configuraciones en el espacio de estados. Luego, la instrucción ->getLength() arroja la longitud de toda la trayectoria. Esta se expresa en radianes.

Luego, para calcular la distancia en el espacio de trabajo, era necesario aplicar Cinemática Directa (FK) a las configuraciones. Para ello, se aplicaron las instrucciones de traslación para obtener un vector xyz ->getFrameTransform(END EFFECTOR).translation() para

cada configuración. Luego, se restaron vectores adyacentes para obtener la diferencia entre configuraciones y se utilizó NORM para obtener la magnitud de la distancia entre dos estados. Estas se sumaron para obtener la distancia real en el espacio de trabajo. Se expresa en metros.

Al ejecutar el archivo desde la terminal de Linux, se obtiene la siguiente respuesta:

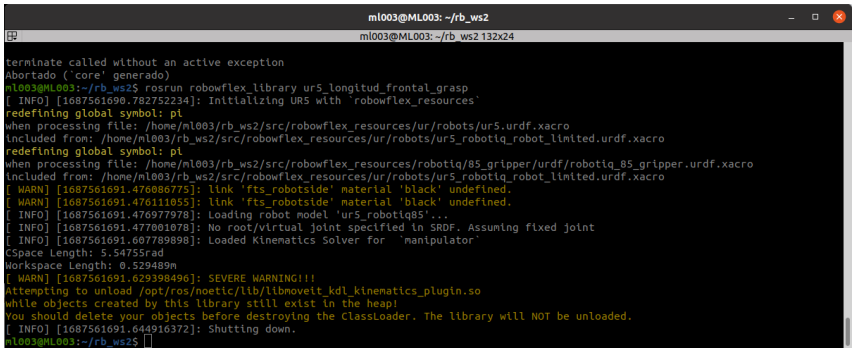

Figura 4: Desplazamiento total aproximado y longitud de la trayectoria en el espacio de configuración

Como se puede apreciar en la imagen, las distancias son:

- Distancia en espacio de configuración: 5.54755 radianes
- Distancia en espacio de trabajo: 0.529489 metros

¿Si una trayectoria tiene una menor longitud en el espacio de configuración que otra, también tiene una menor longitud de desplazamiento de su efector final?

Si bien parece intuivo que existe una relación directa entre ambas longitudes, esto no necesariamente es cierto. La razón por la cual ocurre es porque, en un manipulador de múltiples grados de libertad, una misma posición ubicada en el espacio se puede lograr a través de más de una configuración de los links del robot. En ese orden de ideas, por la naturaleza de la cinemática inversa de un robot manipulador, una trayectoria corta en el espacio de configuración puede implicar mayores movimientos en el espacio de trabajo. Al final de cuentas, los algoritmos de planeación sólo buscan resolver el problema de planeación mediante heurísticas aproximadas. Sin embargo, la longitud de una trayectoria en el espacio de trabajo no necesariamente será optimizada, sólo importa llegar a la posición final deseada y eso se puede lograr mediante múltiples configuraciones que pueden ser cercanas entre sí, por lo cual, se tendría un recorrido corto en el espacio de configuración.

3. Punto 3

En tercer lugar, se generó un nuevo script para determinar la cinemática inversa del robot a partir de las poses generadas en la planeación. Es decir, en esta nueva oportunidad, se utilizaron las poses en el espacio de trabajo (coordenadas xyz) y se enviaron Queries de Cinemática Inversa (IK) por cada pose para determinar una configuración posible para llevar al robot a la posición final deseada.

La posición inicial del manipulador es la siguiente:

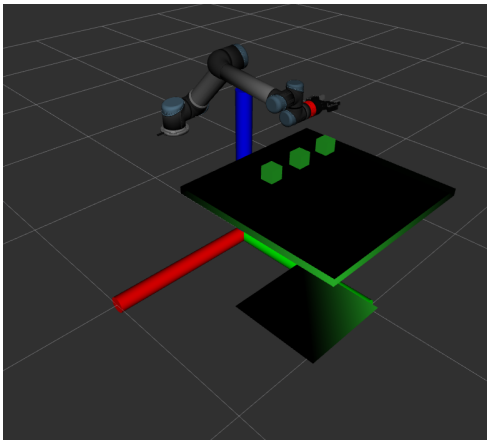

Figura 5: Configuración inicial Cinemática Inversa

Se utilizó el siguiente comando

```
const auto query = Robot::IKQuery(GROUP, {pos_mano}, {END_EFFECTOR}, scene);
```

A partir de este, se obtiene una configuración para cada pose del robot, que forma el siguiente trayecto en simulación:

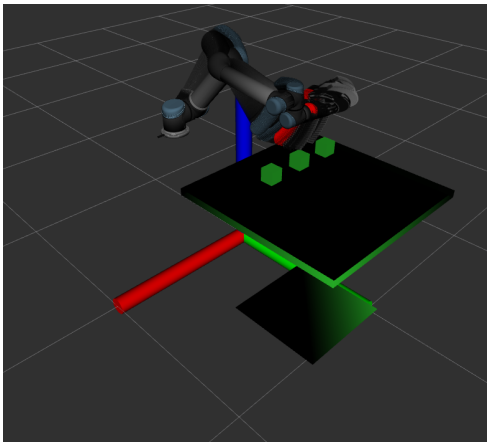

Figura 6: Configuración de agarre Cinemática Inversa

A nivel cualitativo, la diferencia con la trayectoria generada mediante planeación en el Punto 1 es que el brazo se levanta un poco más, lo cual resulta en un ascenso más amplio del manipulador para luego descender y acercar el efector a su posición frontal.

4. Punto 4

A continuación, se modificó el script `"ur5_benchmark.cpp"` para ejecutar la evaluación de la solución por algoritmos de planeación del Punto 1. Se programó para que se ejecuten un total de 100 intentos de solución cada uno con duración de 5 segundos. Pasado este timeout, si no encuentra una solución correcta y/o aproximada, aborta la operación. La siguiente imagen muestra el resultado en términos del tiempo de ejecución para cuatro algoritmos: RRT Connect, RRT, PRM y EST:

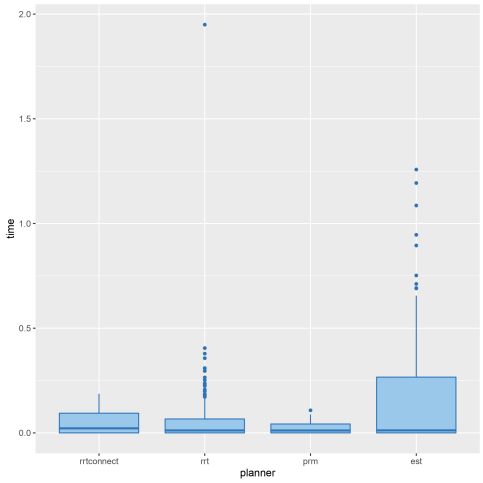

Figura 7: Tiempo de planeación

A partir de esta gráfica, es posible dar cuenta de que los algoritmos lograron resolver siempre el problema de planeación. Adicionalmente, el tiempo promedio de todos los algoritmos fue de aproximadamente 0 segundos. Es decir, el problema de planeación inicialmente planteado en el Punto 1 no es un problema difícil de resolver. De hecho, los algoritmos no tardan en encontrar la solución para este. En el peor caso, el algoritmo que es menos eficiente es el EST y el más eficiente es el PRM.

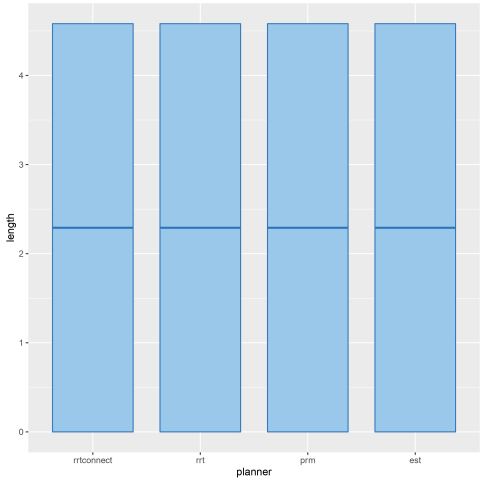

Figura 8: Longitud de la trayectoria

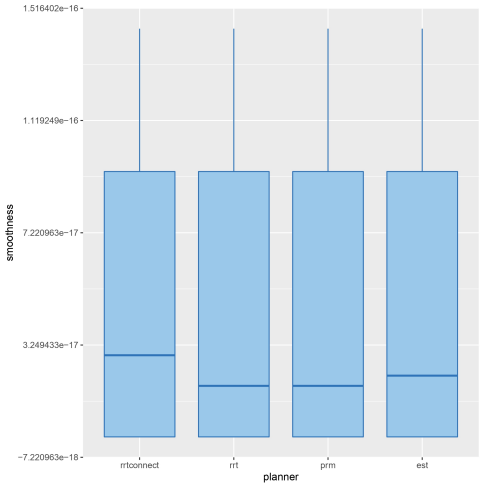

Figura 9: Suavidad de la trayectoria

Con respecto a la longitud y la suavidad de las trayectorias encontradas, los cuatro algoritmos arrojan valores prácticamente equivalentes. La longitud de la trayectoria generada en planeación se encuentra siempre entre 0 y un poco más de 4.5, con una media aproximada de 2.25, mientras que la suavidad se encuentra en el orden de 10^{-17} así como la media de este criterio para los cuatro algoritmos.

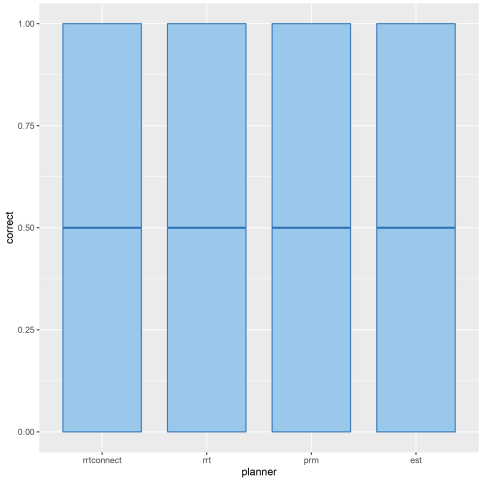

Figura 10: Trayectorias correctas

Finalmente, respecto al criterio de Correctitud, los cuatro algoritmos arrojan la misma distribución. El diagrama de cajas se encuentra entre 0 y 1, con una media de 0.5. Esto implica que, para cada uno de los algoritmos, se encontró la solución correcta de planeación en la mitad de las oportunidades. Así mismo, hubo casos en los cuales fallaron los cuatro, pero de igual manera, todos los algoritmos son capaces de resolver el problema de forma correcta.

5. Punto 5

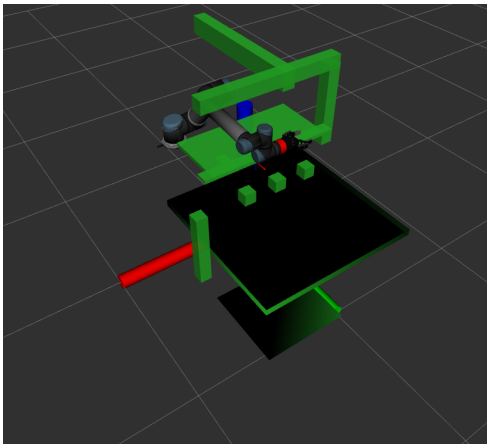

Figura 11: Vista isométrica obstáculos adicionales

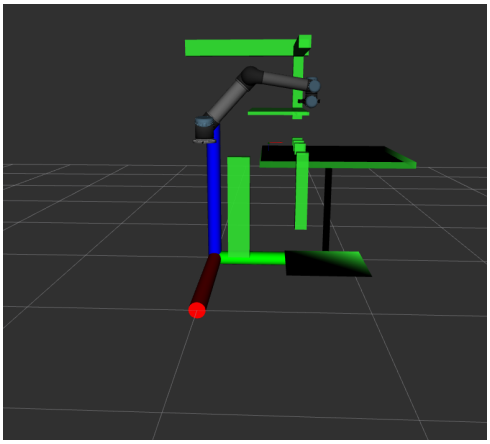

Figura 12: Vista lateral obstáculos adicionales

Para el último punto, se utilizó la escena `"blocks_table_hard.yml"` que contiene los obstáculos adicionales, como se observa en las figuras 11 y 12. Para realizar el benchmarking satisfactoriamente, fue necesario utilizar parte del código `"ur5_frontal_grasp_hard.yml"`. Específicamente, la parte que carga la pose inicial del robot a partir de la escena y la pose final, del cubo a agarrar. Esto debido a que, al intentar correr el algoritmo de benchmarking empleando las configuraciones iniciales y finales de las joints, el robot se estrellaba con algún obstáculo o el algoritmo no encontraba una solución con ningún planeador, independientemente del tiempo (> 60 segundos) o el número de iteraciones (> 100 intentos). Las siguientes imágenes evidencian el aumento de dificultad para los planeadores en solucionar el problema en comparación con el punto anterior, al haber agregado los obstáculos anteriores.

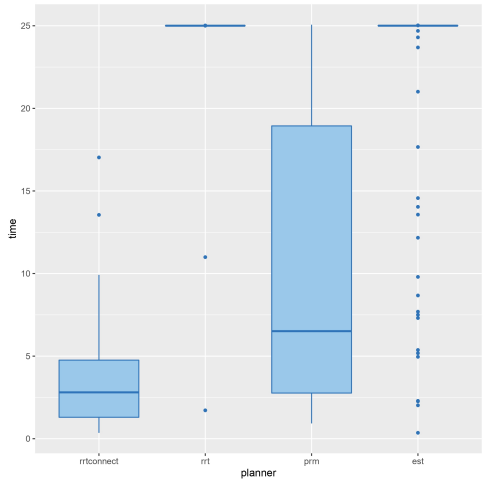

Figura 13: Tiempo de planeación

Como podemos observar, para todos los planeadores el tiempo de ejecución supera el segundo, y aumenta en algunos planeadores hasta alcanzar el máximo tiempo establecido, que corresponde a 25 segundos. Así mismo, podemos observar que en la mayoría de casos, para el rrt y el est dicho tiempo no es suficiente y no alcanzan la solución como se puede evidenciar en la figura 16, que muestra las trayectorias correctas alcanzadas según cada planeador. Por otro lado, también podemos observar en la figura 14 que la longitud para todos los planeadores fue mayor a 10, a excepción de los casos en los que no se alcanzó la solución que se muestran en 0.

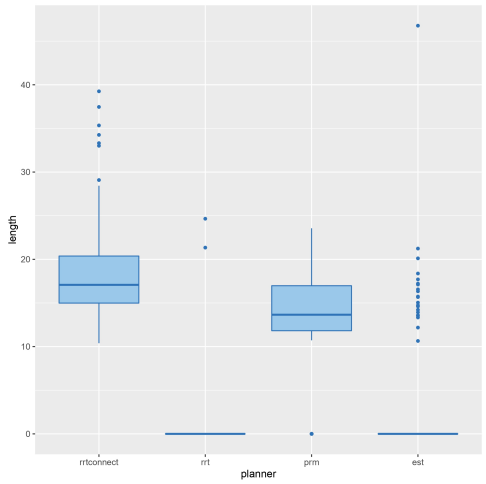

Figura 14: Longitud de la trayectoria

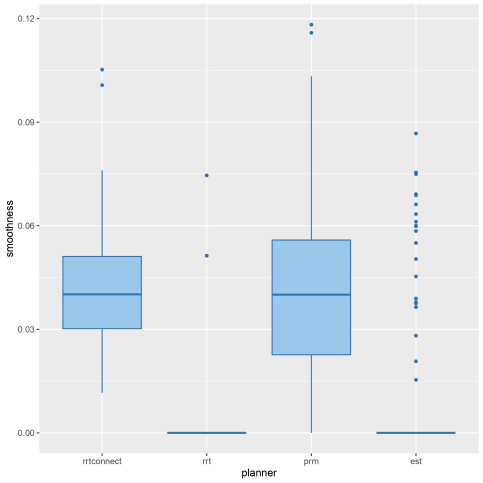

Figura 15: Suavidad de la trayectoria

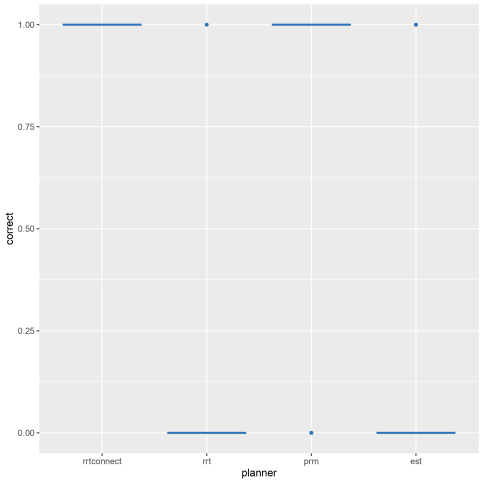

Figura 16: Trayectorias correctas