Ejercicio 5 – Operaciones morfológicas y Calibración

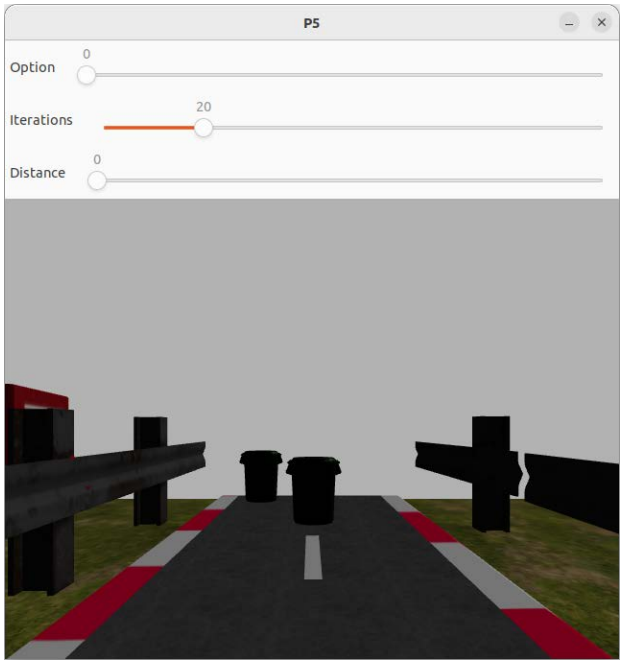
Este ejercicio tiene como objetivo aplicar los conceptos aprendidos en el Tema 6: Operaciones morfológicas y Tema 7: Calibración.

La defensa del ejercicio se hará en clase, y hay que entregar un archivo **cv_node_p5.cpp** con el código generado que deberás subir al Aula Virtual.

Puntos totales posibles del ejercicio: 10

Instrucciones

Utilizando el simulador con Tiago, se pide crear un programa que trabaje con la imagen y muestre en la parte superior varios controles deslizantes (sliders) como los de la figura.



Se pide que en el escenario **aws_racetrack** en cada una de las 3 opciones se haga lo siguiente con la imagen **BGR** del Tiago:

- **Opción 0**: mostrar la **imagen original sin procesar**. En caso de haber puntos pintados de las opciones 2 y 3, **borrar los puntos** y dejar únicamente la imagen original sin procesar.
- **Opción 1**: Identificar el **esqueleto** de las líneas del circuito (laterales blancas y rojas, y blancas centrales). Proyectar líneas de **3D a 2D** a diferentes distancias, y calcular las proyecciones de **2D a 3D** de los puntos seleccionados con el ratón teniendo en cuenta la imagen de profundidad.
- **Opción 2**: Mostrar la **imagen de profundidad** correctamente aplicando las técnicas estudiadas de **realzado de la imagen**, y calcular las proyecciones de **2D a 3D** de los puntos seleccionados con el ratón teniendo en cuenta la imagen de profundidad como en la opción 2.

Tanto en la opción 1 como en la opción 2, los puntos se tienen que mantener. Con cada click, se capturará la pulsación del ratón sobre la imagen y se realizará una proyección de 2D a 3D, tan solo se borrarán cuando se seleccione la opción 0. Para ello, se recomienda que dichos puntos sean almacenados de forma correcta en alguna estructura de datos.

Para añadir un evento que controle los click de ratón, es necesario crear un Callback:

```
// create mouse callback
cv::setMouseCallback( "window_name", on_mouse, 0 );
```

Donde “window_name” es el nombre de la ventana que controlará este evento, y la función on_mouse es la función a la que se llama cuando ocurra una pulsación, la cual se define como sigue:

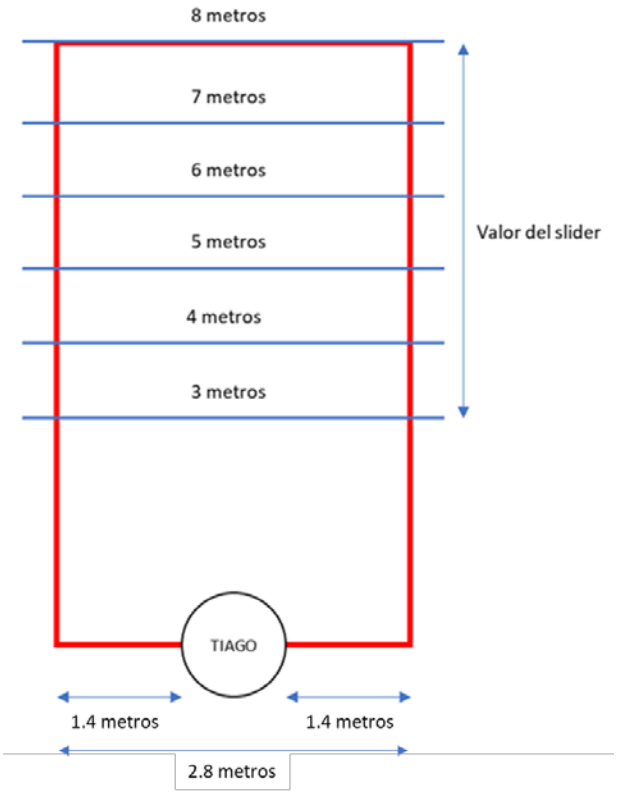
```
// create mouse callback
void on_mouse(int event, int x, int y, int, void*)
{
    switch (event){
      case cv::EVENT_LBUTTONDOWN:
        std::cout << "Left button" << std::endl;
        break;
      case cv::EVENT_RBUTTONDOWN:
        std::cout << "Right button" << std::endl;
        break;
      default:
        std::cout << "No button" << std::endl;
        break;
    }
}
```

Proyección de líneas 3D a 2D:

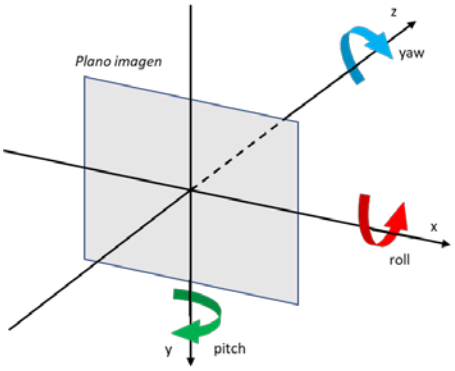
En primer lugar, se pide que se proyecten líneas de diferentes colores distanciadas 1 metro entre ellas en la realidad (mundo 3D), hasta el valor máximo seleccionado en el slider de **Distance**. Este slider irá de **0 a 8**, y junto a cada línea se indicará el valor de la distancia en ese punto.

Esta zona tendrá además 2.8 metros de ancho, es decir, 1.4 metros a cada lado del Tiago, y estará situada a los pies de este, por lo que cada línea nos indicará la distancia de los objetos situados en el suelo en dicho punto (¡ojo! La cámara no está situada en el mismo eje que la base del robot, por lo que hay que tenerlo en cuenta a la hora de generar los puntos).

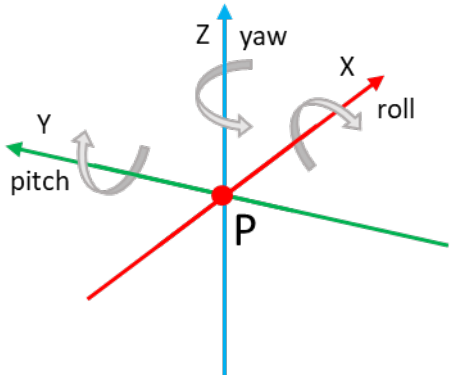
Para ello, puede verse el esquema del mundo 3D en la siguiente figura, junto al resultado final.



Hay que tener en cuenta que el sistema de coordenadas de la cámara difiere del sistema de coordenadas del simulador Gazebo (ver imagen), y que una unidad en el eje z de la imagen (X/Y en Gazebo), equivale a 1 metro en la realidad.



Ejes en la imagen



Ejes en Gazebo

Para calcular la transformada de 3D (X,Y,Z) a 2D (x,y), necesitamos conocer los **parámetros intrínsecos** y **extrínsecos** del sistema para generar las matrices correspondientes.

Para valores de los **parámetros intrínsecos**, hay que observar la **matriz de proyección** que nos facilita el simulador a través de los siguientes comandos vistos en clase:

1. Ver el tipo del mensaje enviado en ROS para la información de la cámara:

```
ros2 topic info /head_front_camera/rgb/camera_info
```

2. Una vez sabemos el tipo del mensaje, utilizaremos el siguiente comando para ver la ayuda del mensaje y observar los parámetros que nos interesaría ver:

```
ros2 interface show <msg>
```

3. Observadas las variables que necesitamos, tenemos que ejecutar el siguiente comando para ver qué valores tiene la cámara del simulador y así poder generar la **matriz *K*** correspondiente a los **parámetros intrínsecos** del sistema.

```
ros2 topic echo <topic>
```

4. Para la obtención de los **parámetros extrínsecos**, es necesario conocer la transformada desde la base del robot a la cámara. Para ello, podemos utilizar las tfs que nos proporciona el modelo del robot. Y ejecutando el siguiente comando, conocer la traslación que sufre la cámara con respecto a la base del robot.

```
ros2 run tf2_ros tf2_echo <frameid_from> <frameid_to>
```

Se valorará positivamente que la creación de las matrices de parámetros intrínseca y extrínseca se haga de forma automática.

Para la matriz de parámetros intrínsecos, hay que crear un nuevo suscriptor con el tipo de mensaje correcto, que se suscribirá al topic:

```
/head_front_camera/rgb/camera_info
```

Una vez dentro del Callback de dicho suscriptor, se generará un modelo del tipo **image_geometry::PinholeCameraModel**, mediante el cuál se puede acceder a través del método **intrinsicMatrix()** a la matriz de parámetros intrínsecos **K**.

En el caso de la matriz de parámetros extrínsecos, es necesario obtener de forma automática los valores de la transformada del punto 4 mediante las funciones de **lookupTransform**.

Crear un esqueleto de las líneas del circuito:

En este apartado, se pide hacer una **identificación las líneas del circuito (los bordes blancos/rojos y la línea central)**, para ello, se puede aplicar cualquier técnica vista en clase para identificar únicamente aquellos objetos que pertenecen a estas líneas.

Además, se pide que únicamente se identifiquen los esqueletos de las líneas que aparecen a partir de la distancia máxima seleccionada en el slider **Distance**, dentro de los 2.8 metros de ancho, salvo en el caso de que la distancia sea inferior a 3 metros y por consiguiente no se proyecte ningún punto en el plano de la imagen. En este caso, se obtendrá el esqueleto de todos los objetos que aparecen en la imagen.

Distancia inferior a 3

Distancia inferior a 3

Distancia a 3

Distancia inferior a 5

Una vez identificados los objetos en la zona deseada, se pide crear el esqueleto de la imagen a partir de operaciones morfológicas.

El algoritmo que hay que implementar para conseguir un esqueleto a partir de operaciones morfológicas es el siguiente:

1. Crear una imagen **esqueleto** vacía.
2. Realizar una **apertura** de la imagen. La llamaremos **open**.
3. **Restar** esta nueva imagen open a la imagen original. La llamaremos **temp**.
4. **Erosionar** la imagen original, y redefinir la imagen esqueleto calculando la **unión** entre el **esqueleto** actual y la imagen **temp**.
5. **Repetir los pasos 2-4** tantas veces como se haya elegido en el slider **Iterations**.

El **esqueleto** se mostrará sobre la imagen original con las líneas del apartado anterior, y el **esqueleto** calculado superpuesto será de color **verde**.

Proyección de líneas 2D a 3D:

Tanto en la opción 1 y 2, es necesario calcular las proyecciones de los puntos seleccionados en la imagen. Dado que al transformar de 3D a 2D la distancia en profundidad no se puede calcular, es necesario utilizar la imagen de profundidad que nos proporciona el Tiago.

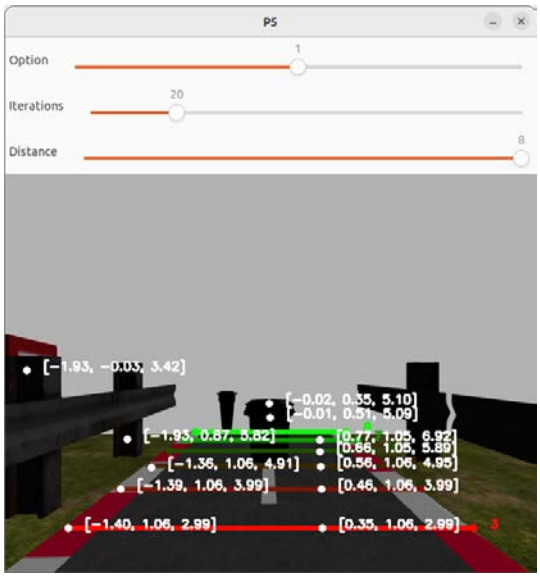
Para ello, es necesario crearse otro suscriptor, el cuál obtendrá la imagen del topic:

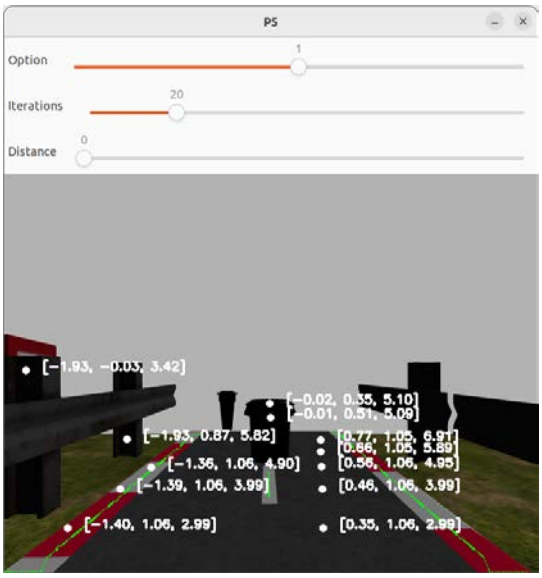
/head_front_camera/depth_registered/image_raw

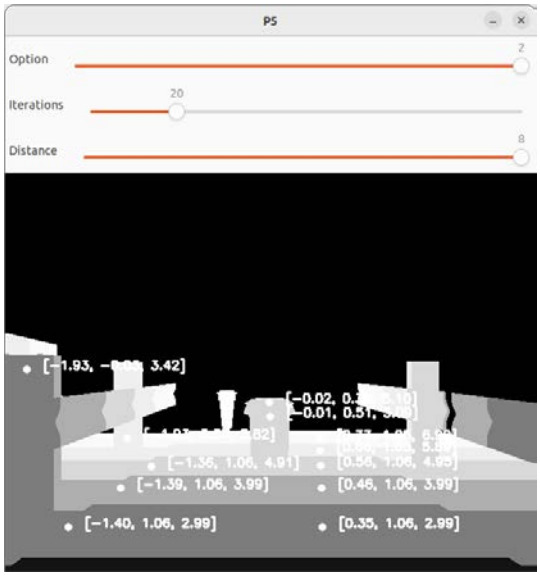
Hay que tener en cuenta que esta imagen al tratarse de una imagen de profundidad, los valores que contiene no son enteros, sino decimales, y equivalen a la distancia en metros calculada al punto sobre el que se está midiendo.

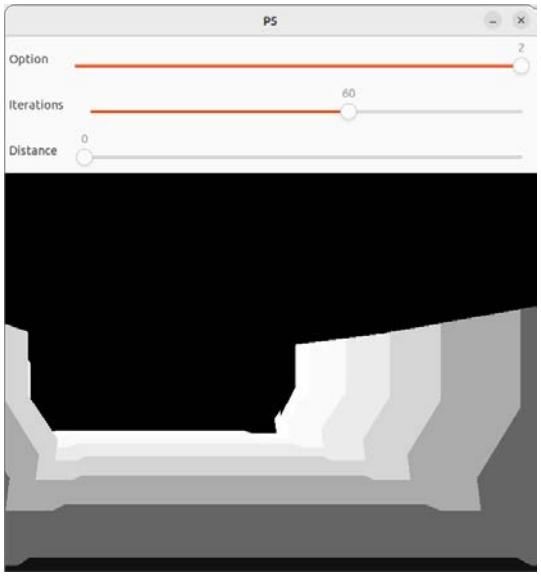
Así pues, haciendo uso de los parámetros intrínsecos de la cámara y de la distancia en profundidad obtenida de dicha imagen, se puede calcular la proyección de un punto 2D (x,y) de la imagen en el mundo real 3D (X,Y,Z).

Capturas:

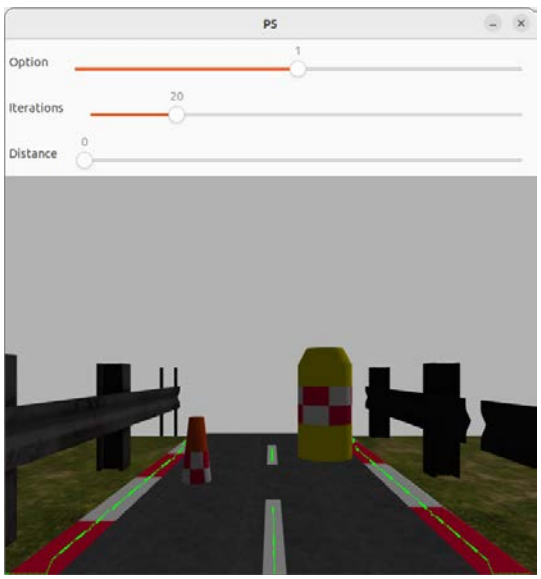

PS
Option
Iterations
Distance
[−1.93, −0.03, 3.42]
[−0.02, 0.35, 5.10]
[−0.01, 0.51, 5.09]
[−1.93, 0.87, 5.82]
[−1.36, 1.06, 4.91]
[−1.39, 1.06, 3.99]
[0.46, 1.06, 3.99]
[−1.40, 1.06, 2.99]
[0.35, 1.06, 2.99]

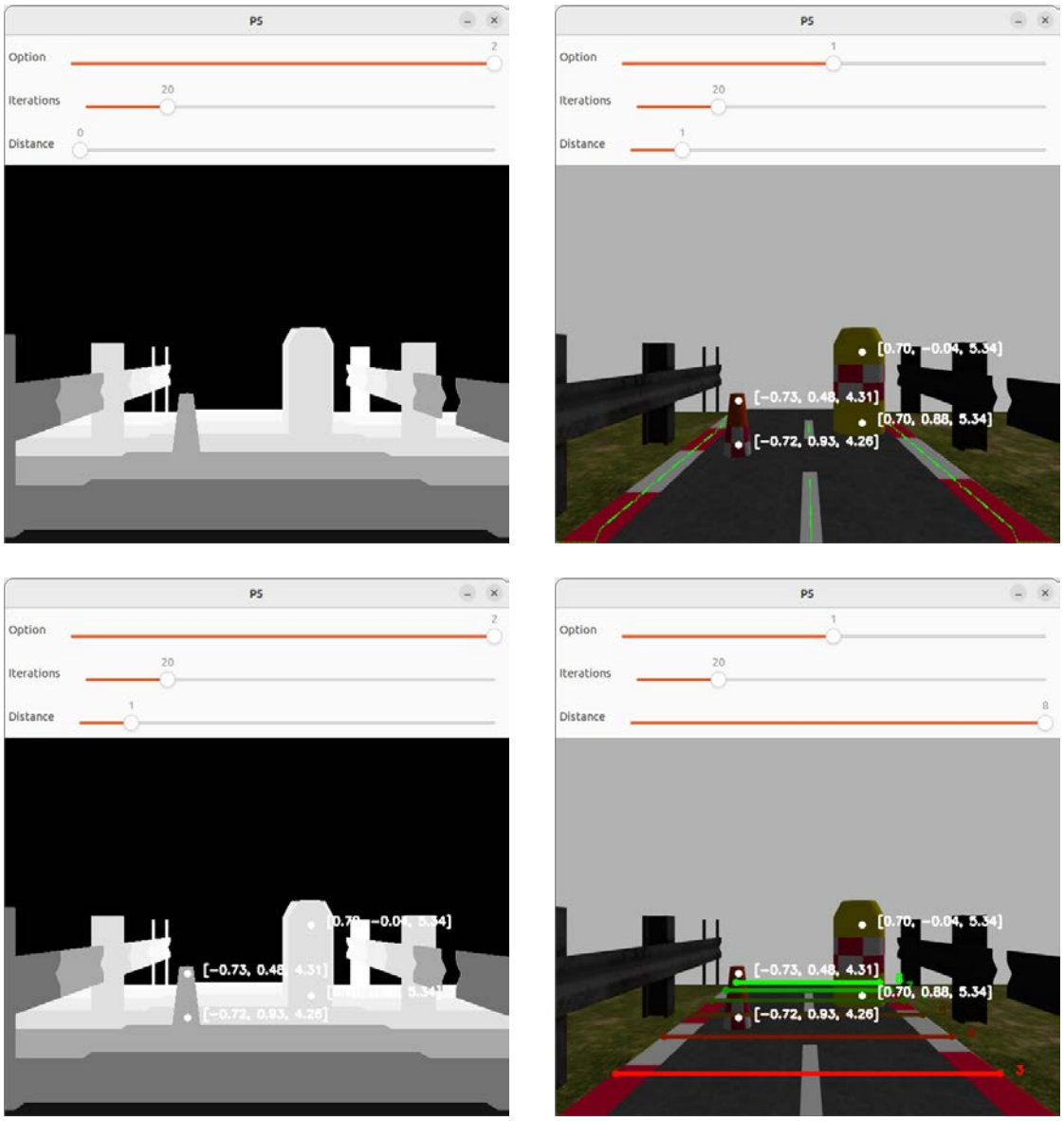

PS
Option
Iterations
Distance
[−1.93, −0.03, 3.42]
[−0.02, 0.35, 5.10]
[−0.01, 0.51, 5.09]
[−1.93, 0.87, 5.82]
[−1.36, 1.06, 4.90]
[−1.39, 1.06, 3.99]
[0.46, 1.06, 3.99]
[−1.40, 1.06, 2.99]
[0.35, 1.06, 2.99]


PS
Option
Iterations
Distance
[−1.40, 1.06, 2.99]
[0.35, 1.06, 2.99]


PS
Option
Iterations
Distance


PS
Option
Iterations
Distance


PS
Option
Iterations
Distance



Ayuda

lookupTransform:
https://docs.ros.org/en/foxy/Tutorials/Intermediate/Tf2/Writing-A-Tf2-Listener-Cpp.html

PinHole model Camera:
https://docs.ros.org/en/api/image_geometry/html/c++/classimage__geometry_1_1PinholeCameraModel.html