Práctica 1 de robótica

Componentes, Webots y un simple robot

1. Introducción

En esta práctica se trabajó con **RoboComp** y el simulador **Webots** para construir un sistema robótico modular. Los objetivos principales fueron:

- Configurar el entorno de desarrollo.
- Integrar un robot simulado en Webots con RoboComp.
- Utilizar un sensor láser virtual (**Lidar3D**) y un controlador de movimiento (**OmniRobot**).
- Desarrollar un componente personalizado llamado **chocachoca** que consuma estos servicios.
- Controlar el robot mediante un **joystick físico**.

Todo el trabajo se realizó en máquinas virtuales del laboratorio (máquinas Beta), donde RoboComp y Webots ya estaban preinstalados.

2. Preparación del entorno

Se realizaron ajustes en el sistema para evitar conflictos con dispositivos de entrada y garantizar una compilación estable:

sudo dkms remove xone/v0.3-57-g29ec357 --all

sudo apt remove xone-dkms

sudo apt upgrade

Estos comandos eliminan el driver **xone** (usado para mandos Xbox antiguos) y actualizan el sistema.

Además, se clonó el repositorio del grupo para gestionar el código:

git clone https://github.com/JairoFarfanC/ROBOTICAG12.git

cd ROBOTICAG12

3. Configuración del Lidar3D

El **Lidar3D** simula un escáner láser 3D compatible con el robot del simulador.

3.1 Instalación del driver de RoboSense

cd ~/software

git clone https://github.com/RoboSense-LiDAR/rs_driver

cd rs_driver

cmake .

make

sudo make install

3.2 Configuración del componente Lidar3D

cd /robocomp/components/robocomp-robolab/components/hardware/laser/lidar3D

nano src/CMakeLists.txt # Se eliminó la dependencia de Open3D

cmake .

make -j23

3.3 Ejecución del sensor

bin/Lidar3D etc/config_helios_webots

El Lidar comenzó a publicar datos del entorno en la red de RoboComp.

4. Creación del componente chocachoca

Se creó un componente personalizado llamado chocachoca en el directorio del grupo:

cd ~/ROBOTICAG12

mkdir actividad_1

cd actividad_1

4.1 Definición del componente

Se generó el archivo de contrato chocachoca.cdsl:

robocompdsl chocachoca.cdsl .

Se editó para incluir las interfaces necesarias:

```
import "Lidar3D.idsl";
import "OmniRobot.idsl";
Component chocachoca
{
  Communications
  {
```

```
    requires Lidar3D, OmniRobot;
  };
  language Cpp11;
  gui Qt(QWidget);
};
```

4.2 Generación y compilación

robocompdsl chocachoca.cdsl .

cd ..

cmake .

cd actividad_1

make

4.3 Implementación

En src/specificworker.cpp, se añadió el código para leer el láser:

```
auto data = lidar3d_proxy->getLidarDataWithThreshold2d("helios", 5000, 1);
qInfo() << "Puntos detectados:" << data.points.size();
```

4.4 Ejecución

bin/chocachoca etc/config

El componente mostraba periódicamente en consola el número de puntos detectados por el Lidar dentro de un radio de 5 metros.

5. Integración con Webots

Se utilizó el **puente webots-bridge** para conectar el simulador con RoboComp.

- Abrir Webots y cargar el mundo:
  File → Open World → ~/robocomp/components/webots-shadow/worlds/Shadow.wbt
- Ejecutar el puente en una terminal:

cd ~/robocomp/components/webots-bridge

bin/Webots2Robocomp etc/config

El robot simulado quedó accesible desde RoboComp a través de las interfaces OmniRobot y Lidar3D.

6. Control mediante joystick

Se compiló y ejecutó el componente JoystickPublish para teleoperar el robot:

cd ~/robocomp/components/robocomp-robolab/components/hardware/external-control/joystickpublish

cmake .

make

bin/JoystickPublish etc/config_shadow

El robot respondió en tiempo real al mover el joystick físico conectado. Se verificó que etc/config_shadow contenía el ID correcto del robot (robot_0 por defecto).

7. Ejecución del sistema completo

Secuencia recomendada en terminales separadas:

1. webots ~/robocomp/components/webots-shadow/worlds/Shadow.wbt
2. cd webots-bridge && bin/Webots2Robocomp etc/config
3. cd lidar3D && bin/Lidar3D etc/config_helios_webots
4. cd joystickpublish && bin/JoystickPublish etc/config_shadow
5. cd ~/ROBOTICAG12/actividad_1 && bin/chocachoca etc/config

El robot era controlable mediante joystick mientras **chocachoca** mostraba los obstáculos detectados.

8. Interfaces IDSL y comunicación con el simulador

La comunicación entre componentes en RoboComp se realiza mediante interfaces definidas en archivos .idsl.

8.1 Interfaz OmniRobot.idsl

Permite controlar el movimiento del robot y obtener su estado:

- setSpeedRobot(float adv, float rot)
  Establece la velocidad de avance (adv, mm/s) y giro (rot, rad/s). ⚠️ No es control por posición, sino por velocidad.
- getBaseState(float &x, float &z, float &alpha)
  Devuelve la pose actual: x, z (mm, plano 2D), alpha (rad).

8.2 Interfaz Lidar3D.idsl

Aunque es un sensor 3D, en esta práctica se usa la proyección 2D:

- TData getLidarData(string name, float start, float len, int decimation)
    - name: dispositivo ("helios")
    - start = 0, len = 2π: escaneo completo
    - decimation = 1: usar todos los puntos

Las llamadas son **síncronas**, esperando la respuesta del componente remoto.

9. Implementación avanzada del componente chocachoca

Se amplió la funcionalidad para:

- Leer datos del Lidar 3D.
- Filtrarlos a 2D (plano horizontal).
- Visualizar puntos en una ventana gráfica Qt.
- Detectar obstáculos cercanos.
- Tomar decisiones básicas de movimiento.

9.1 Configuración de la interfaz gráfica

En SpecificWorker.h:

```
#include <abstract_graphic_viewer/abstract_graphic_viewer.h>


private:
  QRectF dimensions;
  AbstractGraphicViewer *viewer;
  const int ROBOT_LENGTH = 400;
  QGraphicsPolygonItem *robot_polygon;


public slots:
  void new_target_slot(QPointF);
```

9.2 Inicialización de la ventana gráfica

En SpecificWorker.cpp:

```
this->dimensions = QRectF(-6000, -3000, 12000, 6000); // 12m x 6m
```

```
viewer = new AbstractGraphicViewer(this->frame, this->dimensions);
this->resize(900, 450);
viewer->show();

// Añadir representación del robot
const auto rob = viewer->add_robot(ROBOT_LENGTH, ROBOT_LENGTH, 0, 190, QColor("Blue"));
robot_polygon = std::get<0>(rob);

// Conectar eventos del ratón
connect(viewer, &AbstractGraphicViewer::new_mouse_coordinates,
    this, &SpecificWorker::new_target_slot);
```

9.3 Visualización del Lidar

Método draw_lidar():

```
void SpecificWorker::draw_lidar(const auto &points, QGraphicsScene* scene)
{
  static std::vector<QGraphicsItem*> draw_points;
  for (const auto &p : draw_points) {
    scene->removeItem(p);
    delete p;
  }
  draw_points.clear();

  const QColor color("LightGreen");
  const QPen pen(color, 10);
  for (const auto &p : points) {
    auto dp = scene->addRect(-25, -25, 50, 50, pen);
    dp->setPos(p.x, p.y);
```

```
        draw_points.push_back(dp);
    }
}
```

9.4 Lógica en el método compute()

```
// 1. Leer datos del Lidar 3D
auto lidar_data = lidar3d_proxy->getLidarData("helios", 0, 2*M_PI, 1);

// 2. Filtrar a 2D (|z| < 200 mm)
std::vector<QPointF> points_2d;
for (const auto &p : lidar_data.points) {
    if (std::abs(p.z) < 200)
        points_2d.emplace_back(p.x, p.y);
}

// 3. Dibujar en la ventana
draw_lidar(points_2d, &viewer->scene);

// 4. Buscar obstáculo más cercano
float min_dist = std::numeric_limits<float>::max();
for (const auto &p : points_2d) {
    float dist = std::hypot(p.x(), p.y());
    if (dist < min_dist) min_dist = dist;
}
// 5. Decisión básica
if (min_dist < 500) // <50 cm
    omnirobot_proxy->setSpeedRobot(0, 0); // detenerse
else
    omnirobot_proxy->setSpeedRobot(200, 0); // avanzar
```