UNIVERSIDAD CENTROCCIDENTAL LISANDRO ALVARADO
DECANATO DE CIENCIAS Y TECNOLOGÍA
MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN
MENCIÓN INTELIGENCIA ARTIFICIAL

ALGORITMO DE NAVEGACIÓN ALEATORIA SOBRE VREP SIMULATOR

BARQUISIMETO, 2014

JORGE PARRA SAÚL PIÑA

RESUMEN

En el presente informe se describe de manera detallada el proceso de instalación del Simulador VREP, así como también de la aplicación desarrollada. Además, se exponen las características resaltantes del robot, la descripción del simulador y el algoritmo propuesto para la arquitectura con control reactivo.

1. Problema

Se desea desarrollar un algoritmo de navegación con una arquitectura de control reactivo que permita a un robot Pioneer 3DX recorrer un espacio de trabajo, evadiendo obstáculos o evitando colisiones usando como ambientes de simulación virtual VREP Simulator.

2. Descripción del Pioneer P3DX

El pionner P3DX es un robot ideal para usar en el interior de las infraestructuras y cuenta con las siguientes dimensiones: 45,5 cm de largo por 38,1 cm de ancho por 23,7 cm de alto. El robot cuenta con 2 ruedas, las cuales se encuentran una a cada lado y cuenta cada una con un motor y un encoder. Además, tiene una rueda en la parte posterior que le ayuda a mantener el equilibrio.

Igualmente, el pionner P3DX posee sensores de ultrasonido o sonares, los cuales sirven para detectar obstáculos en el ambiente.

Especificaciones del Pioneer P3DX

Construcción

Cuerpo: 1,6 mm de aluminio (recubrimiento de polvo)

Neumáticos: de caucho relleno de goma

Operación

Peso del Robot: 9 kg

Carga operativa: 17 kg

Potencia

Tiempo de ejecución: 8-10 horas con 3 baterías (sin accesorios)

Tiempo de carga: 12 horas o 2,4 horas (con cargador de alta capacidad)

Baterías: Soporta hasta 3 a la vez.

Tensión: 12 V

Capacidad: 7,2 Ah (cada uno)

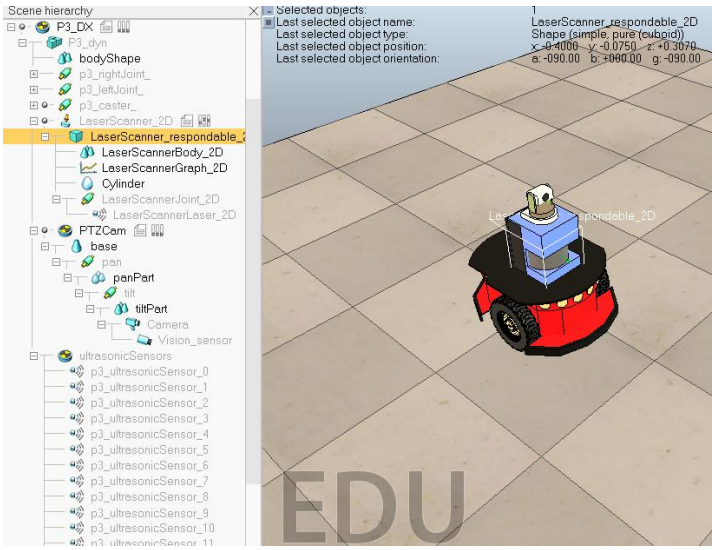
3. VREP Simulator

VREP (Virtual Robot Experimentation Platform) es simulador de propósito general con entorno de desarrollo integrado. Se basa en una arquitectura de control distribuido: cada objeto (modelo) puede ser controlado individualmente a través de una secuencia de comandos incrustada (script), un plugin, ROS, un cliente de API remota o una solución personalizada. Esto hace a a VREP muy versátil e ideal para aplicaciones multi-robot. Los controladores pueden ser escritos en C/C++, Python, Java, Lua, Matlab, Octave o Urbi. VREP se utiliza para el desarrollo rápido de algoritmos, simulaciones de automatización de fábrica, prototipado rápido, educación, simulación de procesos de producción, etc.

4. Características de VREP

a. Es multiplataforma, puede instalarse en sistemas operativos Linux, Windows y Mac OS.
b. Encapsulamiento y portabilidad en modelos
c. Arquitectura cliente-servidor
d. Interoperabilidad con otras plataformas y lenguajes
e. Entorno visual para el diseño de modelos y escenarios
f. Script embebido Lua
g. Excelente para la enseñanza

5. Modelo Virtual de Pioneer P3DX



a. Motores: Derecho e izquierdo

b. Sensores:

   i. Sensor de Visión

   ii. 16 Sensores de ultrasonido

   iii. Sensor de proximidad laser

6. Instalación de VREP

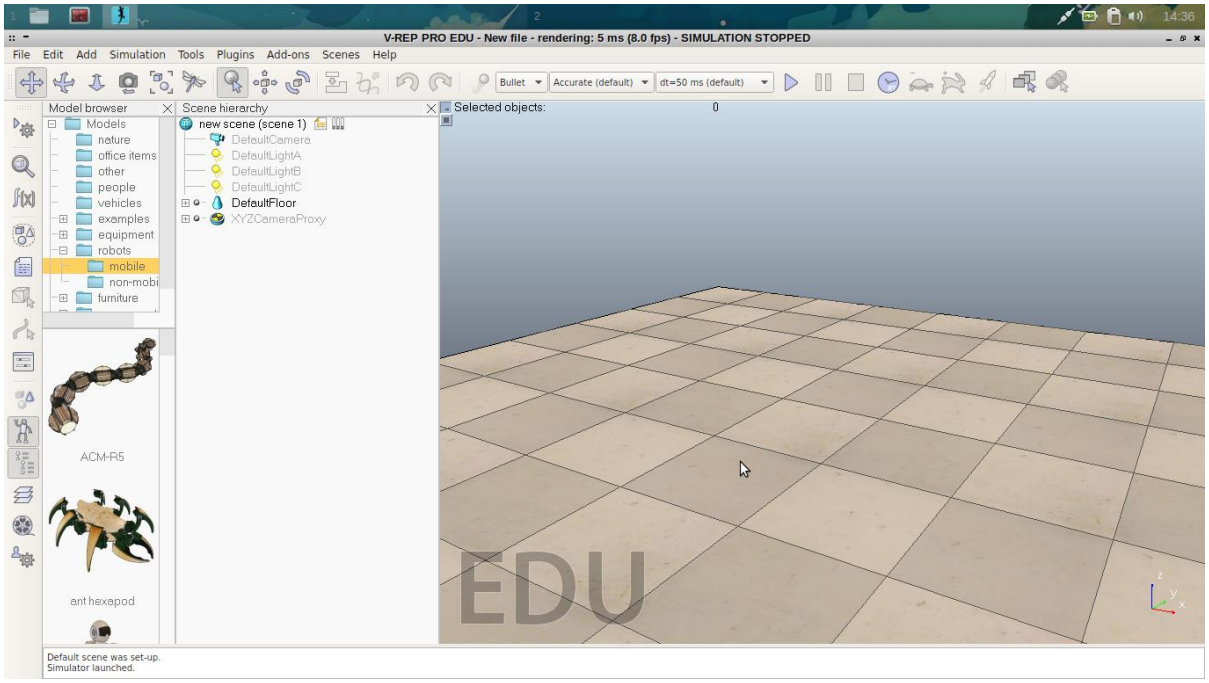
Para efecto del presente informe se explicara la instalación sobre Linux 32Bits:

a. Descargue VREP Simulator EDU. Para la fecha de elaboración de este informe la versión de VREP es 3.1.1. Url de descarga http://www.coppeliarobotics.com/downloads.html,

b. Descomprimir archivo

c. En el terminal ubicarse en el directorio de VREP y ejecutar "/vrep.sh"

d. Posteriormente si podrá visualizar la aplicación



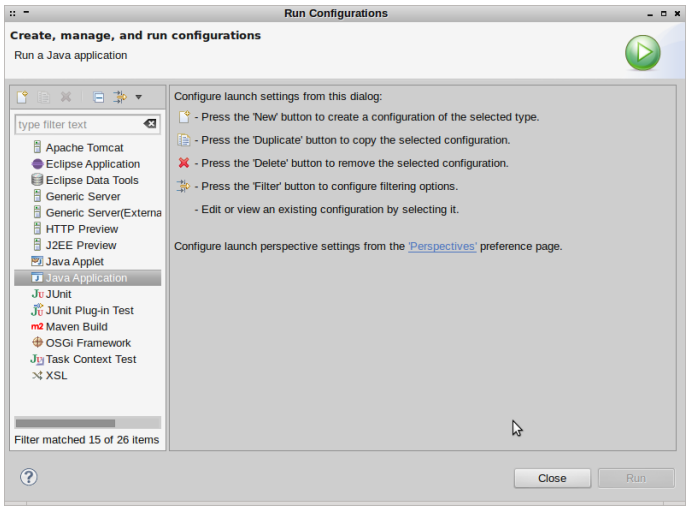
7. La aplicación desarrollada

Se desarrolló una aplicación cliente usando el API de VREP Simulator, posee las siguientes características:

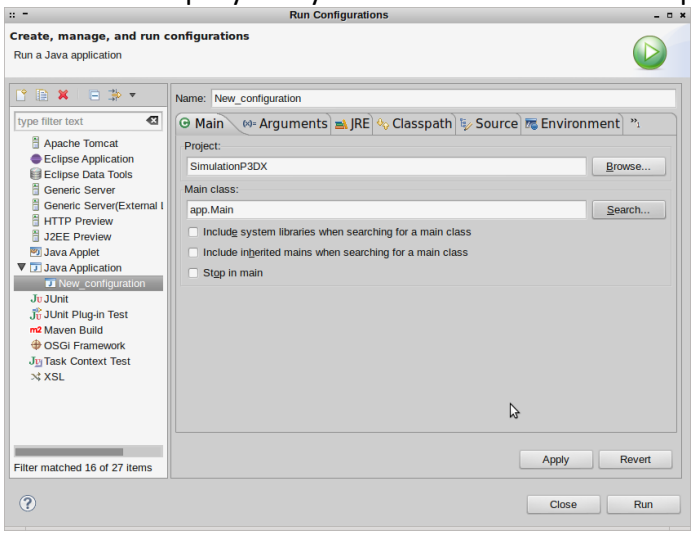
a. Lenguaje de programación Java 7
b. Sistemas operativos probados Windows 7, Debian 7, Ubuntu 13
c. Posee una interfaz gráfica agradable

8. Ejecutar simulación

a. Instalar y abrir Eclipse IDE
b. Importar Proyecto “SimulationP3DX”
c. Abrir menú “Run -> Run Configurations”



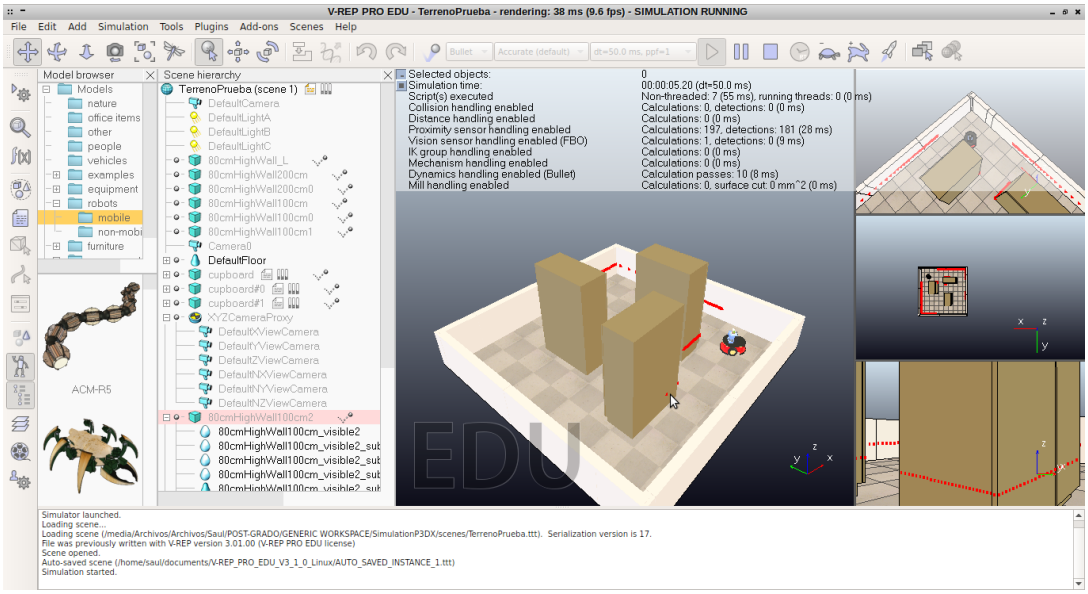
d. Seleccionar “Java Application”
e. Hacer clic en nuevo
f. Seleccionar el proyecto y escribir en “Main class”: “app.Main”



g. Hacer clic en Run

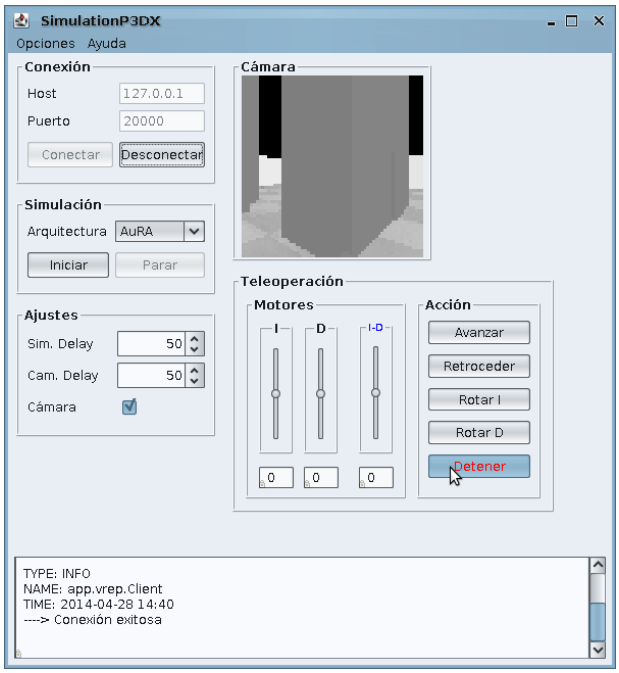
h. En el simulador previamente abierto ir al menú “File -> Open Scene” y seleccionar el archivo “SimulationP3DX/scenes/TerrenoPrueba.ttt”

i. Posteriormente ir al menú “Simulation -> Start Simulation”



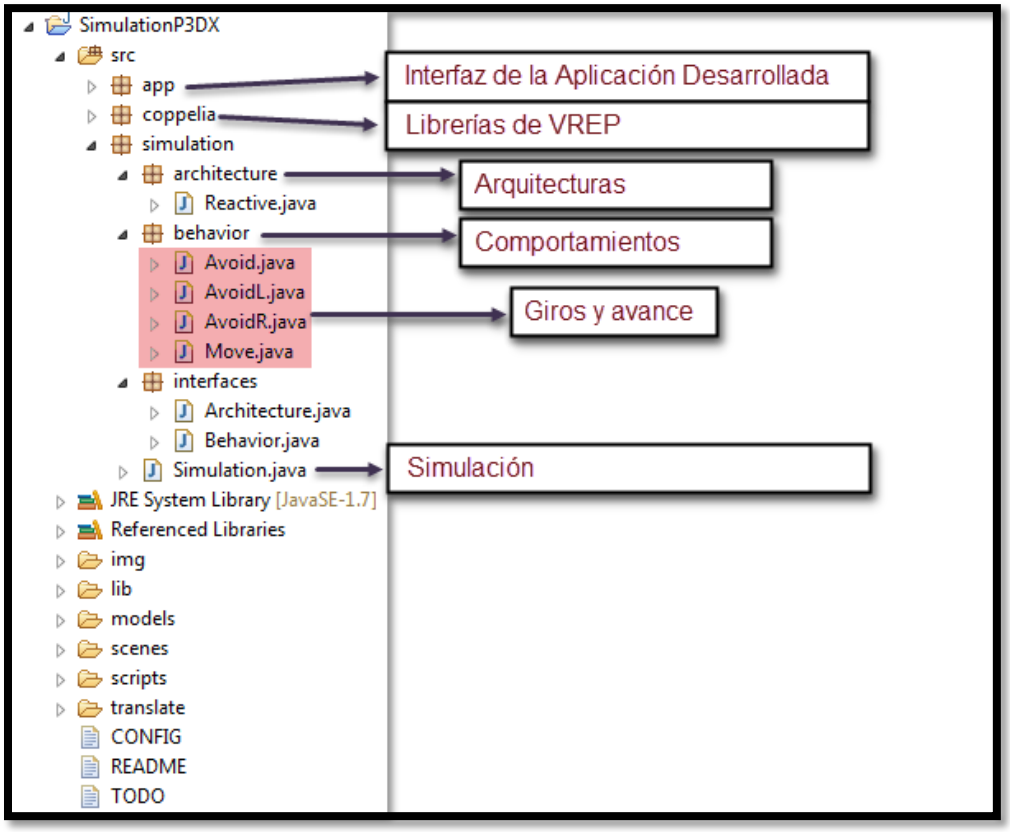
j. En el cliente java seleccionar la arquitectura “Reactive”, e ingresar el Host y Puerto

k. Hacer clic en Iniciar



9. Desarrollo del Algoritmo

La estructura del proyecto es la siguiente:



El paquete **app** contiene las clases para preparar la interfaz gráfica del cliente (vistas) y el paquete **coopelia** contiene todas las librerías utilizadas para la comunicación con el VREP. Dentro del paquete **simulation**, además de la clase **Simulation**, existen 3 paquetes: interfaces, architecture y behavior.

El paquete **interfaces** contiene una **interface Architecture**, la cual tiene los métodos que deben ser implementados por las arquitecturas que se construyan; y la **interface Behavior**, la cual provee la estructura para definir un comportamiento determinado del robot.

El paquete **architecture** contiene las diferentes clases que implementan la **interface Architecture**, en este caso se encuentra implementada la arquitectura reactiva dentro de la **clase Reactive**.

Dentro del paquete **behavior** se encuentran las clases que implementan la **interface Behavior**, en la cual se han implementado las clases **Avoid** (Girar), **AvoidL** (Girar a la izquierda), **AvoidR** (Girar a la derecha) y **Move** (Mover hacia adelante).

La clase **Simulation** crea un hilo en el cual se repite el llamado al método simulate() de la arquitectura seleccionada, mientras exista conexión con el cliente y no se haya detenido la ejecución en la interfaz de la aplicación.

Simulation.java

```
62
63      @Override
64      public void run() {
65          while (Client.isConnect() && !stop) {
66              architecture.simulate();
67              try {
68                  Thread.sleep(delay);
69              } catch (InterruptedException e) {
70                  e.printStackTrace();
71              }
72          }
73          if (!Client.isConnect())
74              stopSimulation();
75      }
76
```

En esta oportunidad, se implementó la arquitectura de control Reactivo, en la clase **Reactive**, específicamente en el método simulate().

Reactive.java

```
51
52      @Override
53      public void simulate() {
54          getDetectionSensorUltrasonicBuffer();
55          if (detectionSensorUltrasonicR0.getValue()
56                  || detectionSensorUltrasonicR1.getValue()
57                  || detectionSensorUltrasonicR2.getValue()
58                  || detectionSensorUltrasonicR3.getValue()
59                  || detectionSensorUltrasonicL0.getValue()
60                  || detectionSensorUltrasonicL1.getValue()
61                  || detectionSensorUltrasonicL2.getValue()
62                  || detectionSensorUltrasonicL3.getValue()) {
63              System.out.println("GIRAR");
64              step = (int) (Math.random() * 10.0);
65              avoid();
66          } else if (avoiding || forwarding) {
67              System.out.println("CONTINUAR");
68              step--;
69              System.out.println(step);
70              if (step <= 0) {
71                  avoiding = false;
72                  forwarding = false;
73              }
74          } else {
75              avoiding = false;
76              forwarding = true;
77              step = (int) (Math.random() * 10.0);
78              move.simulate();
79              System.out.println("AVANZAR");
80          }
81      }
82
```

Allí se evalúan los sensores ultrasónicos y si se detecta algún obstáculo se manda a **Girar** entre 0 y 10 pasos. En caso que no se haya detectado obstáculos, se verifica que si se encuentra **girando o avanzando**, se continúe realizando dicha acción y se vayan descontando la cantidad de pasos. Pero si no se encuentra ni girando ni avanzando se envía la orden para **avanzar hacia adelante** entre 0 y 10 pasos. Sin embargo, si se detecta algún obstáculo en cualquier momento, se activará el comportamiento **Girar**, sin importar si se está ejecutando alguna otra acción.