Actividad 4

Control reactivo del movimiento

En este actividad vamos a realizar la tarea de evitación de obstáculos realizada en la actividad 3 aunque aumentando la complejidad del control de las acciones del robot. En esta ocasión vamos a trabajar con controles reactivos, en este tipo de controles existe una relación directa entre percepción y acción. Para que podáis realizar cómodamente esta y otras actividades os vamos a recomendar un IDE y a proporcionaros una plantilla para trabajar en proyectos sencillos con la librería Aria.

4.1. Evitación de obstáculos

Para trabajar sobre controles reactivos vamos a crear un vehículo de Braitenberg para la evitación de obstáculos. También podéis encontrar información en esta presentación.

Este vehículo va a usar una red neuronal cableada muy simple para llevar a cabo la tarea asignada. Con cableada queremos decir que los pesos están fijados desde el principio, no hay posibilidad de aprendizaje, no tendremos que tratar con esa clase de algoritmos, lo que proponemos es mucho más simple. El esquema de la red sería una capa de entrada, conectada a los sensores, y una de salida, conectada a los motores, solo con conexiones hacia adelante. Esto último implica que cada neurona de entrada puede estar conectada (sinapsis) a cada neurona de salida, pero no hay sinapsis entre neuronas de la misma capa.

El mundo para probar la actividad debe ser cuadrado10x10-obs.map.

4.1.1. Implementación del control

La forma simplificada en la que vamos a implementar la red neuronal va a ser la siguiente.

4.1.1.1. Variables de la clase Control

La clase Control deberá tener al menos las siguientes variables:

- Una lista de valores para guardar las entradas de los sensores, normalizadas entre 0 y 1. Esta lista normalizada va a representar las salidas de las neuronas de la capa entrada

de la red o, si no queremos verlo como una red neuronal, simplemente las entradas del control.

- Una lista de valores umbral para cada entrada. Dado que el objetivo de este control es la evitación de obstáculos, solo nos interesan los cercanos, por lo que conviene fijar una medida umbral en mm. por encima de la cual la entrada se considera 0. Este umbral puede ser distinto para cada entrada en función de la posición de su sensor, por ejemplo si no movemos hacia adelante un objeto frontal cercano es mucho más peligroso que uno lateral a la misma distancia.
  El valor umbral puede servir para normalizar la medida entre 0 y 1 sin más que dividir la distancia entre él, para medidas inferiores a él, también se podría usar la sigmoide.

- Una matriz bidimensional de pesos. Los pesos indican la importancia de una determinada entrada para un motor (rueda), pueden tomar cualquier valor, 0 indica la inexistencia de sinapsis, valores positivos son sinapsis facilitadoras y los negativos inhibidoras. Con un mismo valor de entrada pesos pequeños variarán poco la salida y pesos grandes la variarán significativamente.

- Una lista de valores para guardar las salidas del control, normalizadas entre -1 y 1, representarían las salidas de las neuronas de la capa de salida. -1 significará máxima velocidad hacia atrás, 1 significará máxima velocidad hacia adelante.

Para la implementación de las listas y la matriz bidimensional se pueden usar arrays básicos de C++, teniendo en cuenta las diferencias con Java, los contenedores e iteradores de la STL o los contenedores e iteradores de Qt.

4.1.1.2. Métodos a implementar

Hay que implementar los métodos protegidos de la Clase control, ver 4.2.3, en las parte indicadas en el código con // TODO:

- init(), básicamente dar valores a los pesos y a otras variables que se puedan necesitar.

- input(), recoger los valores de los sensores que se vayan a utilizar usando los métodos oportunos de ArLaser y ArSonarDevice o sus clases antecesoras, como ArRangeDevice. En nuestro caso solo vamos a usar el láser dividiendo su haz en 5 sectores iguales, o sea, tendremos 5 sensores de entrada, consideraremos cada valor que devuelve ArRangeDevice::currentReadingPolar() como una medida de entrada. A continuación esos valores se normalizan entre 0 y 1 como se ha comentado más arriba.

- proccess(), hallar las salidas y normalizarlas. Si llamamos $\vec{e} = (e_1, \ldots, e_n)$ al vector de las n entradas, $\vec{o} = (o_1, \ldots, o_m)$ al vector de las m salidas, $\vec{\theta} = (\theta, \ldots, \theta)$ una constante y

$$W = \begin{pmatrix} w_{11} & \ldots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \ldots & w_{nm} \end{pmatrix}$$

a la matriz de pesos, se tiene que $\vec{o} = \vec{e}W + \vec{\theta}$. También podemos escribirlo así:

$$o_j = \sum_{i=1}^{n} e_i w_{ij} + \theta \quad j = 1, \ldots m$$

w_{ij} representa la contribución de e_i en o_j, valores positivos contribuyen a mover la rueda hacia adelante y negativos hacia atrás. El valor de la constante θ representa la salida en ausencia de obstáculos cercanos, digamos la velocidad crucero, fijaremos esa constante en 0.5, con lo que la velocidad crucero será la mitad de la velocidad máxima del robot. Una vez calculadas las salidas hay que normalizarlas entre -1 y 1. En este caso puede bastar un truncamiento de los valores fuera del intervalo $[-1, 1]$ o también se podría usar la tangente hiperbólica.

- output(), establecer las velocidades de las ruedas, ArRobot::setVel2(), haciendo que las salidas del control representen proporciones (con signo) de la velocidad máxima del robot, ArRobot::getTransVelMax().

No olvidar bloquear y desbloquear el robot al invocar sus métodos (lock() y unlock()).

4.1.2. ¿Cuál debe ser el comportamiento del robot?

Lo más importante no debe chocar con obstáculos, pero también es deseable que el robot no se quede parado o atrapado por obstáculos. Por ejemplo si el peso del sensor central es el mismo para cada motor, el robot se quedará parado frente a obstáculos frontales, para evitarlo una estrategia sencilla es hacer esos pesos distintos, con lo que el robot tendrá una tendencia a evitar los obstáculos dirigiéndose hacia un determinado lado. Otro comportamiento deseable es pasar por pasillos estrechos, como el que hay en la esquina inferior izquierda del mundo cuadrado10x10-obs.map.

Para conseguir estos objetivos debéis ajustar los umbrales de las entradas y en especial los pesos de la red, a base de observar los comportamientos y corregirlos.

En esta actividad no se pide el uso de sensores traseros para evitar golpes por detrás, habrá que ajustar los pesos para evitar esta contingencia. Como actividad extra, no evaluable, podéis usar los sensores traseros del sonar para evitar golpes traseros.

4.2. IDE y plantilla de proyectos

4.2.1. El IDE QtCreator

Vamos a recomendaros es uso de QtCreator. De esta manera dispondréis de:

- Un gestor de proyectos, para poder trabajar con varios archivos fuentes y organizar archivos de inclusión y librerías.
- Un editor de textos orientado a la programación, con el que podréis usar la función de completado de código con las funciones de Aria y resaltado de sintaxis entre otras.

- Un depurador gráfico, para atrapar los bugs.
- Un diseñador de interfaces gráficas, con el que poder pasar datos y órdenes al robot en ejecución de una forma cómoda. El uso de interfaces gráficas de Qt no será pedido en las actividades, es opcional y en una próxima actividad daremos un ejemplo de su uso.
- Acceso predeterminado a las funciones de la librería Qt. Se pueden usar sin restricciones las clases y funciones de esta librería, aunque las actividades pueden realizarse sin necesidad de usarlas. La ventaja de su uso radica en que Qt nació como una herramienta para llevar la facilidad de programación de Java a C++, por ello las funciones de acceso a listas tiene una versión de sintaxis al estilo Java, ver Qt Containers.

No es necesario usar el IDE para realizar las actividades del curso, éstas se pueden realizar al estilo de los ejemplos de la carpeta examples de Aria, todo en un solo archivo colocado en ese directorio, pero es muy recomendable usarlo.

4.2.2. Instalación de Qt y QtCreator

En las distribuciones de Linux se suelen encontrar los paquetes de las librerías Qt, de QtCreator y de otras herramientas Qt. Normalmente basta con abrir el manejador de paquetes del sistema y seleccionar para instalar el paquete qtcreatorxxx. Al seleccionarlo para instalar el manejador debería indicar los paquetes de Qt necesarios para instalar, si es que no están ya instalados. Podemos usar cualquier versión de Qt de la 4.x o 5.x y su correspondiente QtCreator, lo más recomendable es usar la que viene con el sistema y no descargar la última versión disponible en la red e instalarla localmente.

Una vez instalado QtCreator arrancadlo y en el menú Help->Contents y leed la sección Getting Started para lanzar el programa de prueba, por ahora no vamos a usar programas con GUI (Qt GUI Application) sino de consola (Qt Console Application). A continuación leed las secciones Coding y la subsección Debugging dentro Debugging and Analyzing, para que veáis las posibilidades que en estas cuestiones dispone QtCreator.

Para terminar crear un proyecto nuevo a partir de la plantilla que se comenta en la subsección siguiente 4.2.3.

4.2.3. Plantilla de proyectos no GUI

La plantilla de proyecto está disponible en la carpeta Software en Alf, plantilla_nogui.tar.gz. Consiste en un directorio, plantilla_nogui, con cuatro archivos:

- plantilla_nogui.pro, el archivo del manejador de proyectos para un proyecto Qt de consola con acceso a directorios de Aria (usa /usr/local/Aria, si lo tenéis en otro tendréis que cambiarlo).
- main.cpp, archivo de inicio de ejecución donde se carga el robot con láser y sonar. Se crea un objeto de clase Control y se invoca el método de ejecución del Control, a partir

de aquí la ejecución pasa a la clase Control, solo se vuelve a main() tras pulsar Esc o Ctrl-C y se sale del programa.

- control.h y control.cpp, archivos de declaración y definición de la clase Control. Hemos diseñado esta clase con un constructor, en el que se pasa un puntero al robot, una solo función pública, execute(), que contiene el bucle de ejecución. Tiene varios métodos protegidos no implementados: init(), para la inicialización y input(), proccess() y output(), que forman en este orden el bucle de ejecución. input() lee y preprocesa las entradas de los sensores. proccess() procesa estas entradas para obtener las salidas. output() convierte las salidas del control en órdenes motoras.

Las variables de la clase Control robot, láser, y sonar son punteros, hay que acceder a sus miembros con el operador "->" no con el ".".

Para usar la plantilla debéis:

- Copiar el directorio plantilla_nogui con su contenido al directorio donde se desee tener el nuevo proyecto.
- A continuación cambiar el nombre del directorio plantilla_nogui por el del nuevo proyecto.
- Cambiar el nombre del archivo plantilla_nogui.pro por el del nuevo proyecto.
- Editar el nuevo archivo .pro y cambiar el valor de TARGET (plantilla_nogui) por el del nuevo proyecto.
- Abrir el nuevo proyecto en QtCreator.
- Añadir las variables necesarias a la clase Control e implementar los métodos protegidos.