MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL

DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL

ROBOTS AUTÓNOMOS

PRÁCTICA 1: NAVEGACIÓN DE ROBOTS Y ARQUITECTURAS DE CONTROL

JAVIER MORENO VEGA

---

5 de noviembre de 2017

Índice

1. Introducción

Navegación de robots es la metodología que permite guiar el curso de un robot móvil a través de un entorno con obstáculos. [3]

Como bien definía esta práctica, era una introducción al software V-Rep para la navegación con robots sobre el robot Pioneer-3DX, empezando con un controlador reactivo e intentando trabajar con aprendizaje con refuerzo.

[Imagen 1: Robot Pioneer-3DX]

La práctica se empezó con un controlador muy sencillo que solo evitaba los obstáculos, y se fue iterando sobre él hasta conseguir uno "más avanzado"que seguía un corredor y terminaba al detectar el final, tal y como pedía el enunciado de esta práctica. Se han creado varios mundos para experimentar con éste controlador, como se enseñará en los experimentos.

Para el final de la práctica se ha investigado sobre el aprendizaje por refuerzo [2] con el fin de que fuera el propio agente el que creara su controlador al explorar el entorno. Se presenta una versión preliminar con una idea para trabajar en mundos continuos usando Q-learning [4] [1] y el Proceso de Decisión de Markov (MDP).

2. Instalación y arquitectura creada

La instalación del software no tuvo ningún problema complicado, lo único que costó mas detectar fue que había que darle permisos de ejecución al script Python client. Esta instalación se ha realizado sobre un sistema Xubuntu 16.0xx con kernel xxx, V-Rep xxx y python xxx. Además sobre un portatil Macbook Pro 2017 macOS High Sierra 10.13, V-Rep Pro Edu 3.4.0 y python 2.7.10.

Acerca de la arquitectura seguida, se ha intentado modularizar todo lo posible para evitar scripts demasiado largos y confusos. Se ha creado una clase llamada RemoteConnection que es la encargada de gestionar la conexión con V-Rep, recibe en el constructor los argumentos de llamada (puerto, los manejadores de los motores y el robot, etc) y crea una instancia RemoteConnection. Esta clase es la que inicia el controlador, el cuál es pasado a través de otro script como argumento del método de RemoteConnection run". Por lo tanto desde el controlador no es necesario llamar a ninguna función de V-Rep, sólo se llaman a métodos ya creados de RemoteConnection, y esta es la que llama a V-Rep. Además se encarga de la gestión de errores y de escribir mensajes de log en la consola de V-Rep.

Cada controlador o versión de este se escribe en archivos diferentes sin hacer ninguna llamada y esta se realiza desde client.py el cual ejecuta el import del controlador a utilizar, llama al constructor y al método run.

Aunque en la versión de prueba dada por el profesor los sensores se pasaban como argumentos desde el script del servidor de V-Rep, he preferido llamar a los sensores desde los script remotos en python.

Para los experimentos se han creado tres mundos, a parte del dado por el profesor (y modificado un poco).

Clase RemoteConnection

Método	Devuelve	Descripción
init	Instancia	Constructor
run(controller)	void	Inicializador de la simulación
printMessage(message)	int	Mensaje en consola V-Rep
readASensor(sensor)	float	Valor de proximidad de un sensor
readAllSensor(numberOfSensors)	array	Valores de proximidad
getConnectionId()	int	Obtiene la conexión
setLeftMotorVelocity(velocity)	void	Define la velocidad
setRightMotorVelocity(velocity)	void	Define la velocidad
getAngle()	float	Ángulo del robot
getPosition()	array	Posición del robot
getSensorAngle(sensorName)	float	Ángulo de un sensor
setAngle(angle)	void	Define el nuevo ángulo del robot
isConnectionEstablished()	bool	Conexión establecida

3. Controlador reactivo

Un controlador reactivo usado sobre un agente es aquel que dependiendo de una entrada (sensor) decide una acción u otra, sin aprender de ella. Es decir, acción-reacción, siendo la reacción la proximidad a algún obstáculo.

Se han desarrollado tres versiones de un controlador reactivo, siendo la primera la más simple, y la última una funcional que cumple con los requisitos de la práctica. Voy a explicar las tres versiones y enseñar los tres mundos usados para los experimentos.

3.1. Primera versión

Este controlador es el más sencillo, la base es un límite de proximidad definido. Además se definen: una velocidad base, una velocidad de giro y una variable de control de rotación.

En cada iteración del controlador, y con un tiempo de pausa, se obtienen los valores de todos los sensores delanteros (sensor 1 a 8). Sobre estos valores se comprueba si hay alguno que detecte una distancia inferior al límite de proximidad definido, y sí es así se genera un giro al lado contrario de donde esté ese sensor, es decir, si es un sensor de la izquierda (1 a 4) gira a la derecha, y si es un sensor de la derecha gira a la izquierda. Cuando se define el giro se activa la variable de rotación, la cuál no se desactivara hasta que ningún sensor tenga una proximidad inferior al límite, mientras esta variable esté activada el robot estará girando.

3.2. Segunda versión

En la segunda versión del controlador reactivo se ha intentado añadir algunas mejoras: eliminar giros muy grandes, y añadir aleatoriedad sobre el ángulo.

Aunque la mejora principal es el método "setAngle", cuando empecé a trabajar en la primera versión me di cuenta que definir los giros era algo muy "prueba y error", así que para esta segunda versión me puse a trabajar en alguna forma de poder definir el giro exacto que quería y así evitar obstáculos con un giro bien definido. También implementé métodos para obtener el

ángulo actual del robot y los ángulos de los sensores, y así poder girar hacia el sensor que quisiera. Esta es la base de esta versión y la final.

Como en la versión anterior se definen las mismas variables y se añade una velocidad máxima, una mínima y un límite de proximidad para la parte frontal. Este controlador se ejecuta también mientras la conexión esté activa y además añade una variable de control por si se termina la simulación desde el propio controlador (llega al final).

En cada iteración se obtienen los valores de los sensores delanteros y se comprueba que ninguno devuelva una proximidad inferior al límite, si es así se observa si el sensor que tenga esta proximidad pertenezca a los delanteros frontales (4 o 5); si no se obtiene el ángulo de este sensor y una uniforme entre -10 y +10, definiendo esta variación en el ángulo del robot (ángulo del sensor +(-) 10).

En el caso que ningún sensor supere el límite pero que los sensores frontales superen el límite proximidad frontal, se obtiene el sensor que tenga mayor proximidad (más distancia a cualquier obstáculo). Si esta proximidad es 1 (proximidad que se devuelve cuando el sensor supera su límite de detección): se selecciona aleatoriamente un sensor de todos los que tengan proximidad 1, obtiene el ángulo del sensor seleccionado y se genera una uniforme entre 0 y el ángulo, definiendo en el robot este nuevo ángulo.

Por último tenemos el caso de que no se supere ningún límite. Entonces si la velocidad actual es menor que la máxima y ningún sensor detecta obstáculo (todos devuelven 1) se aumenta la velocidad en 0.1. En cambio si la velocidad es mayor que la velocidad mínima y algún sensor detecta obstáculo (el menor no es 1) se reduce la velocidad en 0.1.

3.3. Versión final

Esta versión final es una modificación de la anterior y que cumple con los requisitos de la práctica, navegar por un corredor y pararse al detectar el final.

La modificación es eliminar aún más los giros grandes e ir girando lentamente si fuera necesario. Por lo tanto se ha eliminado el segundo caso de la versión anterior (cuando superaba el límite de proximidad frontal), y el pri-

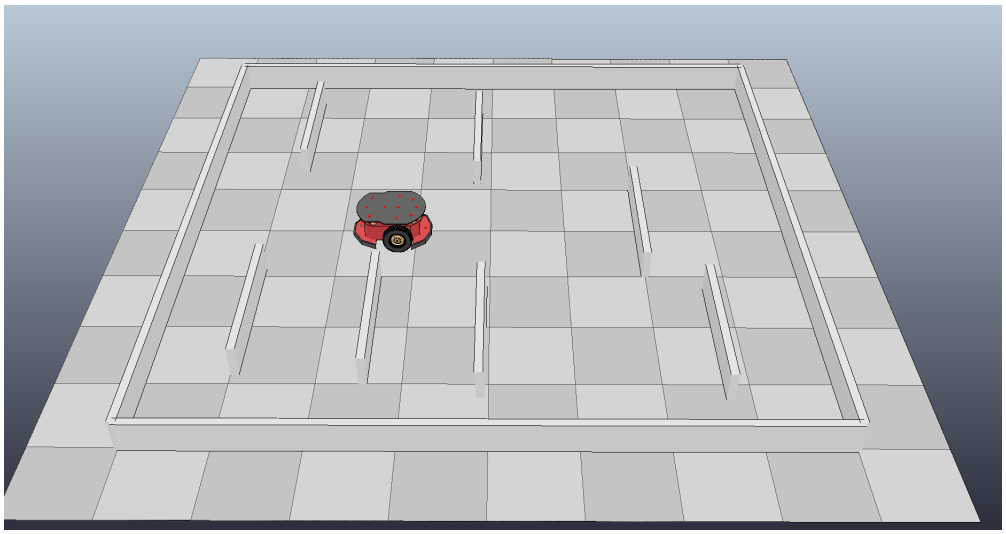
mer caso se mantiene igual excepto porque ahora se ha definido un intervalo de giro, si es un sensor de la izquierda -5 y +5 si es un sensor de la derecha; este intervalo se calcula además con una uniforme entre 0 y el mismo, y es el ángulo que se define.

El último caso es igual que en la version anterior.

3.4. Experimentos

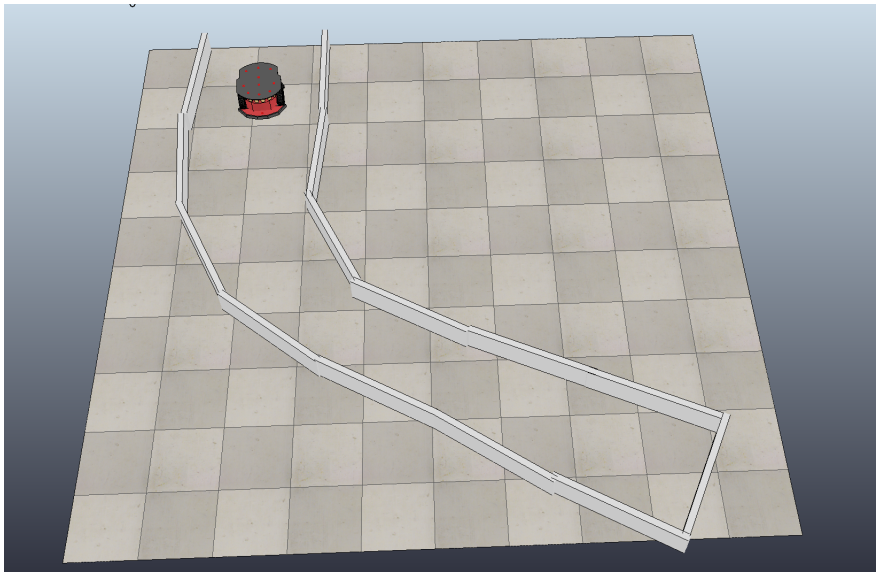
Se han realizado experimentos sobre tres escenarios, uno el dado por el profesor modificado añadiendo más obstáculos; y dos nuevos diseñados por mi como corredores con giros y una pared final para determinar el final del corredor.

Sobre el primer escenario funcionan todos los controladores, es decir no chocan con los obstáculos, y en el caso del último controlador en cuanto va a chocar decide que ha finalizado la simulación; pero con los otros dos seguiran ejecutándose debido a que sus giros son de mayor grado.



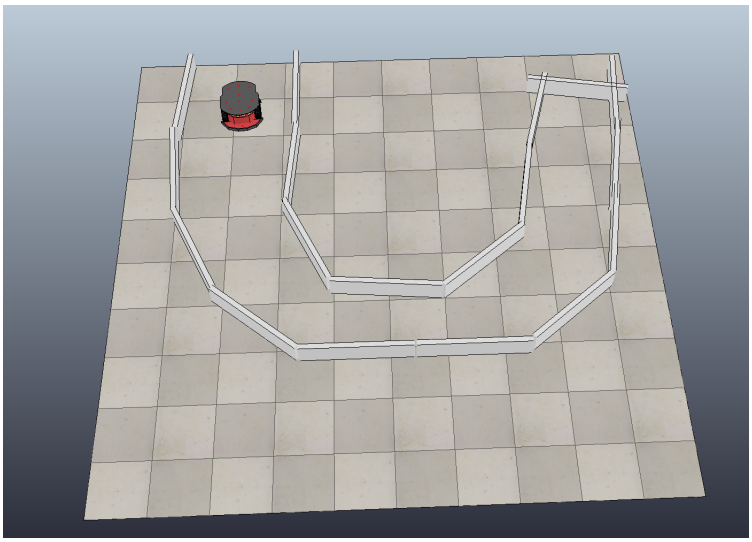
[Imagen 2: Escenario 1]

En el segundo escenario se ha diseñado un corredor muy sencillo con un ligero giro, y funciona correctamente el controlador final, cumpliendo con el requisito de la práctica.



[Imagen 3: Escenario 2]

Por último en el escenario tres se añadió un giro más pronunciado para ver el funcionamiento del controlador final, y funcionó correctamente.



[Imagen 4: Escenario 3]

4. Aprendizaje por refuerzo

Finalizando la práctica se ha intentado un modelo de aprendizaje por refuerzo. A continuación explico los problemas que han surgido, las ideas iniciales de lo que se ha querido desarrollar y una primera versión implementada.

4.1. Ideas iniciales

El aprendizaje por refuerzo consiste en que el agente aprenda a partir del entorno y genere un controlador para futuras acciones.

Según el proceso de decisión de Markov (MDP) se tiene un conjunto finito de estados (S), un conjunto de acciones (A) y las interacciones entre ellos. Aquí surgió el primer problema ya que había que modelar este método en un espacio continuo, es decir, una forma de poder decidir que el robot está en un estado cuando este esté en determinado rango del espacio.

La idea inicial para resolver este problema fue un modelo de estados por rango dirigido por los 16 sensores del robot, limitando el rango al sensor que detectara menor proximidad en cada lado del robot.

Tendriamos definido el estado actual del robot el conjunto de acciones, y su probabilidad que dependerá de la distancia a un obstáculo, a las que se puede acceder desde el estado sería la dirección (el ángulo) de cada sensor. Por lo tanto el funcionamiento sería el siguiente: se define el estado actual, se elige una dirección y el robot empieza a andar hasta que sale del estado (rango de este); entonces vuelve a definir el nuevo estado y a repetir el proceso.

La recompensa de un estado será la suma de las distancias a obstáculos de los sensores en ese estado.

El siguiente problema que se encontró fue al intentar trabajar con Q-Learning ya que había que inicializar el conjunto de estados, y en este caso los estados eran desconocidos (así como su número), por lo tanto simplemente se eliminó éste paso del algoritmo y ahora cada vez que sale de un estado comprueba si el estado al que ha llegado ya existía o es nuevo.

4.2. Primera versión

La estructura de esta versión para implementar Q-Learning es la siguiente:

- S: Estados, array dinámico
- A: Acciones, matriz $|S|$ x 16
- Q: Q(s, a) Función que devuelve la probabilidad de elegir a estando en el estado s.

Se han definido cuatro funciones para realizar el aprendizaje:

- getState: devuelve el estado actual a partir de la posición
- addNewState: añade un nuevo estado a S
- addActionsToQ: añade las probabilidades de las acciones
- calculateDistance: calcula la distancia entre dos puntos del mapa
- setReward: obtiene la recompensa de un estado actual a partir de los sensores
- getAnglesOfSensors: devuelve los ángulos de las direcciones de los sensores
- addActionsToState: añade las acciones a un estado
- createNewState: Crea un nuevo estado y añade todas las posibles acciones con sus probabilidades
- normalizeValue: Normaliza un valor entre 0 y 1

La idea para esta primera implementación ha sido ir ejecutando el controlador sin determinar un estado terminal, se irán detectando los diferentes estados actuales y se moverá entre estados, haciendo uso de las funciones explicadas arriba. Para añadir un nuevo estado se ha definido el número -1, cuando se detecta que estamos en un estado -1, es que es un nuevo estado.

He basado la implementación en Q-Learning, y una visión general del algoritmo procedería de la siguiente forma: Iniciariamos en un estado sin definir, obtendríamos la posición actual y los sensores de proximidad, y si estamos

en un estado diferente al anterior (el estado inicial sería una excepción) se procedería a cambiar de estado y definir las recompensas.

Además se ha añadido una probabilidad alta para que tienda a mantener la misma dirección en vez de elegir una nueva, e intentar así que el robot siga una dirección más estable.

4.3. Experimentos: Primera versión

La versión de aprendizaje por refuerzo se ha utilizado sobre los tres escenarios usados por la versión reactiva, pero no se ha conseguido un funcionamiento correcto. El robot terminaba por no elegir la mejor acción y acababa chocando o quedándose atrapado en un giro continuo.

Habría que mejorar la función de recompensa y la probabilidad de dirección.

5. Bibliografía

Referencias

[1] Alex Irpan. Q-learning in continuous state action spaces. https://www.alexirpan.com/public/research/281areport.pdf. [Internet; descargado 31-octubre-2017].

[2] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. http://incompleteideas.net/sutton/book/bookdraft2017june19.pdf. [Internet; descargado 31-octubre-2017].

[3] UMA. Capítulo 2: Navegación de robots móviles. http://webpersonal.uma.es/~VFMM/PDF/cap2.pdf. [Internet; descargado 31-octubre-2017].

[4] CHRISTOPHER J.C.H. WATKINS. Q-learning. https://link.springer.com/content/pdf/10.1007%2FBF00992698.pdf. [Internet; descargado 31-octubre-2017].