

Utilización de Statecharts para el Modelamiento del Comportamiento de un Robot Móvil

Ignacio Garrido Z. y Danilo Bassi A.

*Departamento de Ingeniería Informática
Universidad de Santiago de Chile
Santiago, CHILE
dbassi@ieee.org, igarrido@vtr.net*

Abstract

Este paper presenta la utilización de statecharts para el modelamiento del comportamiento de un robot móvil, como una metodología sencilla y practica. Se presenta una tarea completa modelada e implementada según este enfoque.

Keywords: Robots Móviles, Diseño, Control, Tareas, Arquitectura.

1. MOTIVACIÓN

Una arquitectura de control define una descomposición modular del sistema de control de un robot y la interacción entre dichos módulos [Maes 94]. Por lo general, las arquitecturas dan énfasis a la implementación del sistema de control, pero no proporcionan facilidades para la posterior utilización del robot en tareas específicas. Esto es valido cuando se desea implementar un robot que cumplirá solo ciertas tareas específicas, como por ejemplo un robot planetario o un robot de vigilancia.

Sin embargo, en el ámbito educativo y académico comúnmente se implementan muchas tareas distintas con un mismo robot, por lo que una arquitectura que soporte una metodología para la implementación de tareas es de suma utilidad. Dicha metodología debe ser clara, sencilla y general, sistematizando y simplificando la implementación de tareas.

2. MODELAMIENTO DE TAREAS UTILIZANDO STATECHARTS

2.1 Enfoque

Los diagramas StateChart [Harel 87] forman parte de la notación UML, y sus principales ventajas radican en su expresividad, sencillez y modularidad. Como menciona su autor, son apropiados para la especificación y diseño de sistemas complejos de eventos discretos, es decir, aquellos que deben

reaccionar continuamente a eventos externos inesperados (como un robot).

En el enfoque desarrollado cada estado esta asociado a una acción, que se define como un comportamiento significativo, idealmente atómico. Las transiciones se generan al finalizar dichas acciones, o al suceder algún evento externo o interno. Las transiciones pueden ser internas a un estado; aquellas que comienzan con la cláusula “do” se ejecutan continuamente mientras se permanezca en el estado. La Figura 1 muestra un ejemplo sencillo para una tarea de transporte de objetos.

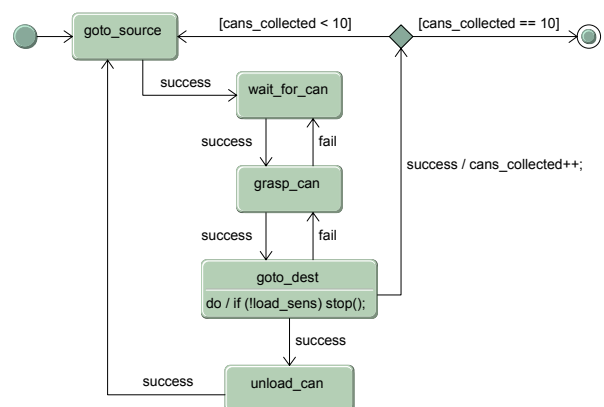


Figura 1. Tarea de transporte modelada mediante un StateChart.

Para que las acciones puedan ser reutilizadas en diversas tareas, es conveniente parametrizarlas

apropiadamente; de este modo, es posible crear una librería de acciones que permite generar el comportamiento del robot solamente parametrizando los estados y agregando las transiciones apropiadas. Por ejemplo, el estado `goto_source` de la Figura 1 podría recibir como parámetro la posición (x,y) de la fuente.

En cuanto a la implementación de los estados, no se ha adoptado ninguna formalidad, ya que esta depende en alto grado de la arquitectura y el robot particulares que se utilicen, y por general será mas cómodo para el usuario el programar cada estado de la forma que considere más apropiada.

Un punto importante de mencionar es el hecho de que, por simplicidad, para el modelamiento de tareas no es necesario considerar todas las situaciones posibles. Así, en el ejemplo anterior, en el estado `goto_dest` se asume que el robot será capaz de llegar a su destino, condición perfectamente válida en un experimento de laboratorio. De todas formas, si así se requiriese, sería posible considerar todas las situaciones posibles.

Cabe notar que las capacidades de concurrencia y jerarquía de los StateCharts no son utilizadas en la implementación actual, aunque para modelar tareas complejas en robots con más recursos probablemente serian de utilidad.

2.2 Implementación

El enfoque propuesto forma parte integral de la arquitectura SARA [Bassi 04], que se muestra en la Figura 2^a.

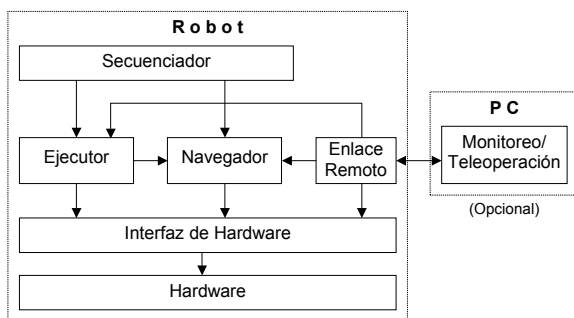


Figura 2. Arquitectura de SARA.

Básicamente, aquí el Ejecutor es un modulo que contiene a los estados, bajo la forma de funciones. El secuenciador, como su nombre lo indica, se encarga de ejecutar los estados en la secuencia establecida por el statechart implementado, es decir, implementa las transiciones. De este modo, se separa la implementación de los estados y las transiciones y se facilitan las pruebas y la reutilización código. Esta

arquitectura fue implementada en el robot Gbot, que se muestra en la Figura 3.

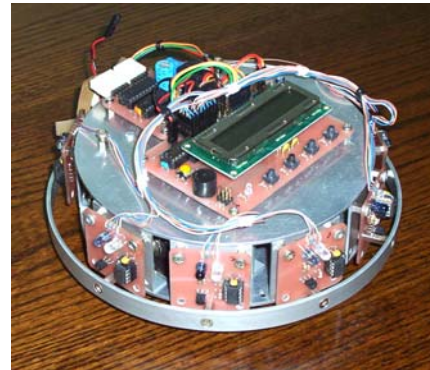


Figura 3. Robot Gbot.

3. EXPERIMENTOS

En esta sección se describe un experimento que valida el enfoque descrito, y que consiste básicamente en la recolección de latas.

3.1 Descripción de la Tarea

El robot parte en una posición inicial y debe primero encontrar un faro de luz que se encuentra en uno de los bordes de un escenario. Una vez hecho esto, debe buscar 5 latas y llevarlas, una por una, junto al faro. Al finalizar debe retornar a la posición inicial. Las condiciones del experimento son las siguientes:

- El escenario consiste en un rectángulo de 1.2x1,5 metros, bordeado por listones de madera de 5 cm de altura.
- Al escenario no llega la luz del sol directamente ni tampoco hay lámparas muy cercanas.
- Las latas deben ser metálicas, de 5 cm de diámetro y 8 cm de altura (vacías).
- El robot no debe chocar, en lo posible, con las latas o con el borde del escenario.
- Las latas son 5, y pueden estar en cualquier punto del escenario, a no menos de 5 cm del borde.
- No hay otros objetos dentro del escenario.

Es importante considerar que debido a que los sensores del robot utilizado tienen un alcance máximo de 13 cm, este debe realizar una búsqueda exhaustiva dentro del escenario para encontrar el faro. Una vez que lo ha encontrado, debe guardar su posición, y no depender más del faro.

3.2 Modelamiento

La Figura 4 muestra el statechart desarrollado para la tarea descrita.

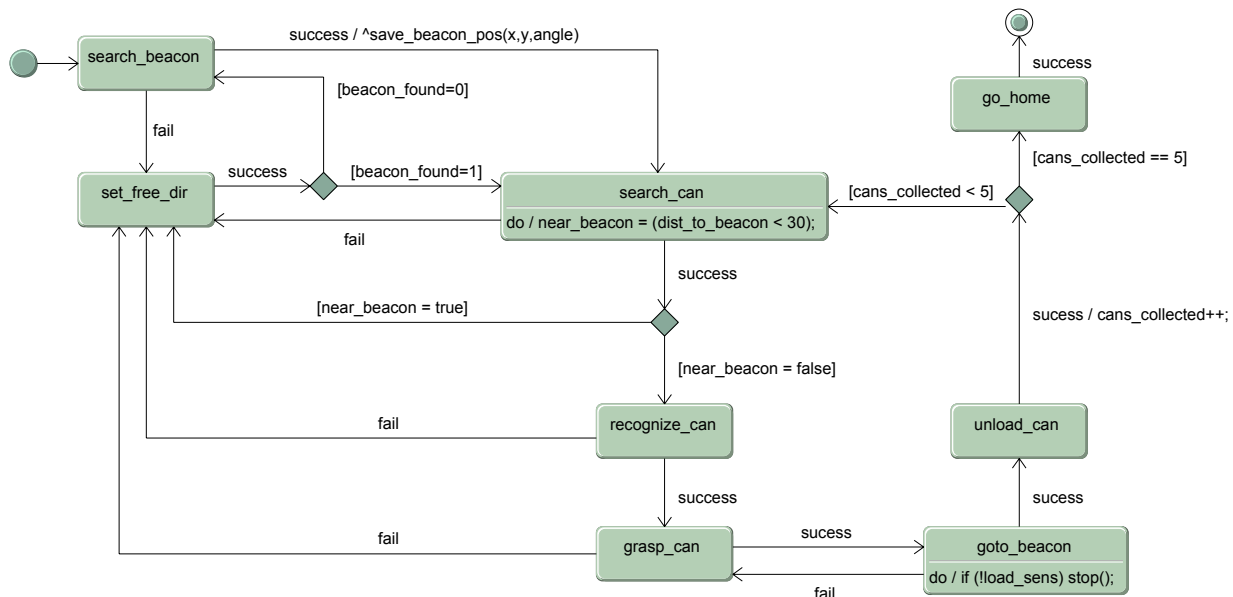


Figura 4. Statechart para tarea de recolección de latas.

A continuación se describen cada uno de los estados:

- search_beacon:** aquí el robot se mueve en línea recta 70 cm, y si en el inter tanto detecta una fuente de luz, se acerca a ella hasta que detecta un obstáculo, se detiene y pasa al estado **search_can** guardando la posición aproximada del faro y poniendo la variable **beacon_found** a verdadero. Si después de recorrer 70 cm no encuentra una fuente de luz o se topa con algún obstáculo, se detiene y pasa al estado **set_free_dir**.
- set_free_dir:** aquí el robot escoge una dirección libre de obstáculos (el algoritmo actual selecciona simplemente entre 4 alternativas). Luego, si es que previamente ya se había encontrado el faro (**beacon_found=1**), entonces se pasa al estado al estado **search_can**; de lo contrario se pasa al estado **search_beacon**.
- search_can:** aquí el robot se mueve en línea recta hasta que detecta un objeto en alguno de sus sensores. Si los sensores contiguos no detectan nada, asume que puede ser una lata y pasa al estado **recognize_can**. En caso contrario, o si no detecto ningún obstáculo después de recorrer 70 cm, pasa al estado **set_free_dir**. Mientras este estado se esta ejecutando, se monitorea la distancia al faro, y si esta es inferior a 30 cm, se pone la variable **near_beacon** a verdadero. Para evitar coger las latas que ya se han recolectado, se pasa al estado **recognize_can** solo si es que el robot no se encuentra cerca del faro (**near_beacon=0**).
- recognize_can:** aquí el robot determina si el objeto detectado en el estado anterior es realmente una lata. Para ello gira hasta quedar en frente del objeto detectado, y si ninguno de los dos sensores frontales marca nada, asume que se trata de una lata (puesto que una lata quedaría en el ángulo muerto frontal). Luego avanza y retrocede dos centímetros para dejar un espacio entre la lata y el robot, por si es que la lata estuviese en contacto con el robot. Finalmente, gira en 180°, y pasa al estado **grasp_can**. Si es que el objeto no es identificado como una lata (como podría suceder con un muro o con objetos de dimensiones mayores que una lata) se pasa al estado **set_free_dir**.
- grasp_can:** aquí el robot retrocede hasta que el sensor en su asidor trasero indique la presencia de un objeto; entonces activa el electroimán para sujetar la lata y pasa al estado **goto_beacon**. Si retrocede 7 centímetros sin que este sensor se active asume que no se trataba de una lata, y se pasa al estado **set_free_dir** para continuar buscando.
- goto_beacon:** aquí el robot se dirige hacia el faro evitando obstáculos y una vez ahí pasa al estado **unload_can**. Si en el trayecto la lata se suelta, pasa al estado **grasp_can** para tratar de sujetarla de nuevo. El robot trata de acercarse lo más posible al faro, pero debido a que es posible que hallan otras lata ahí, una vez que se ha acercado a 10 cm se acerca solo hasta que detecta un obstáculo (es decir, tiene una tolerancia de 10 cm para acercarse al faro).

- **unload_can:** aquí el robot gira de modo de posicionar la lata lo más cerca del faro, y desactiva el imán, incrementando el conteo de latas recogidas (`cans_collected`). Si ha completado cinco (u otro número), entonces pasa al estado `go_home`, de lo contrario pasa a `search_can`.
- **go_home:** aquí el robot se devuelve al punto de origen (0,0) en el sistema de coordenadas.

En general, el diagrama es bastante autoexplicativo, por lo que no es necesario ahondar más en su descripción. Desde luego existen detalles no mencionados de la implementación de cada estado, aunque estos no son muy relevantes para el modelamiento de la tarea.

3.3 Resultados

El experimento fue repetido varias veces con distintas configuraciones, y en todas el robot logro cumplir su cometido. De todas las pruebas, se selecciono una, cuya traza se muestra en la Figura 5. La grafica se obtuvo con el software de monitoreo del robot; no se trata de una simulación (las latas y el faro se agregaron manualmente). Las posiciones originales de las latas han sido marcadas con círculos rojos, mientras que las posiciones finales se indican con círculos verdes. El faro es representado mediante un círculo amarillo.

En general, los resultados obtenidos fueron satisfactorios, por cuanto demuestran el la validez del enfoque utilizado.

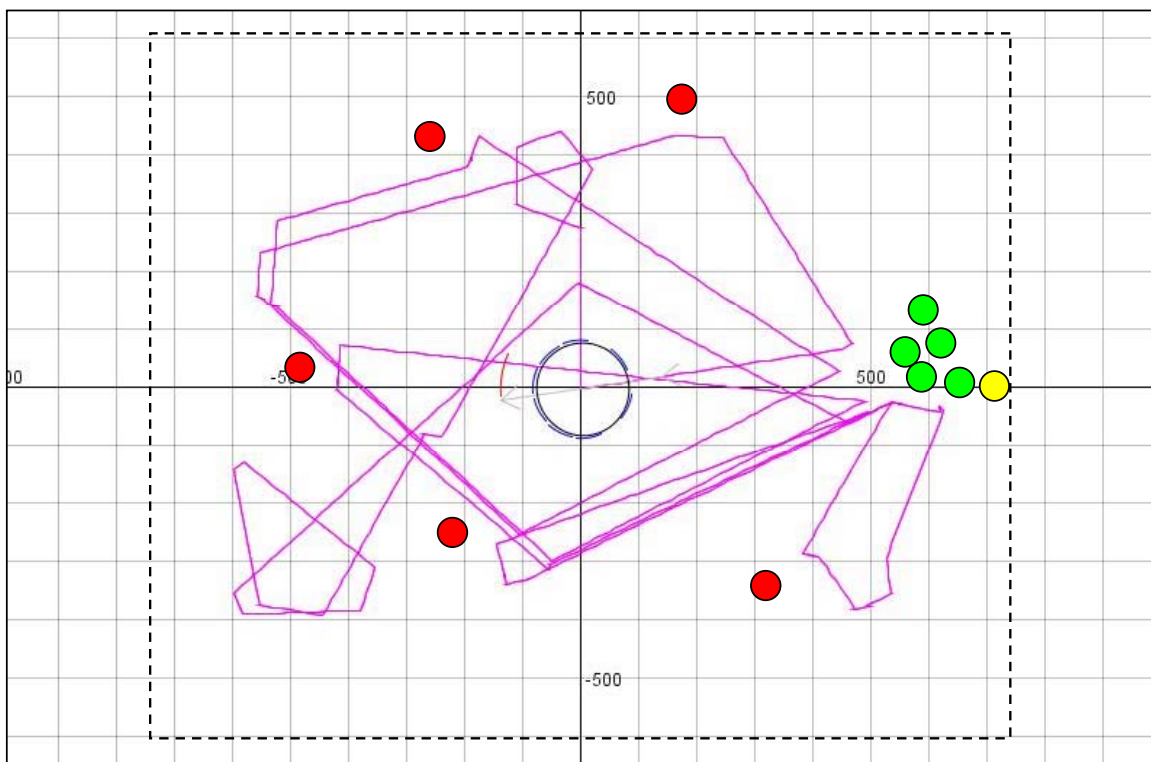


Figura 5. Traza de la tarea de recolección de latas.
Tiempo = 4:08 min

4. TRABAJOS RELACIONADOS

No se conocen antecedentes de la utilización de statecharts para el modelamiento del comportamiento de un robot, aunque sí de diagramas de estado comunes.

En [Gini 96] se utiliza, aunque no de manera formal, un diagrama de estados para modelar una tarea similar a la descrita. En [Arkin & MacKenzie 94] se utiliza un diagrama de estados para modelar el comportamiento

de un robot en forma muy similar a la propuesta, aunque de manera tangencial; el objetivo de este trabajo es la utilización de autómatas finitos, pero para la coordinación perceptual. Quizás el caso más cercano sea [Kosecka & Christensen 95], en donde se utilizan autómatas finitos con bastante formalismo, aunque a un nivel de abstracción más bajo que el utilizado en aquí. Así, por ejemplo, en dicho trabajo los comportamientos “evitar obstáculos” y “seguir objeto” se han modelado ambos mediante autómatas finitos, mientras que en el enfoque propuesto ambos corresponderían a estados.

En general, en ninguno de estos trabajos el modelamiento mediante estados forma parte integral de una arquitectura, ni tampoco se da énfasis a su utilización como metodología de modelamiento.

5. CONCLUSIÓN

Los statecharts constituyen una notación poderosa y expresiva para la modelación de tareas, por cuanto permiten su división en unidades discretas y significativas a la vez que especifican una secuencia de pasos. El hecho de que los statecharts formen parte de la notación UML facilita además la comprensión por terceros.

La descomposición de las tareas en unidades discretas y poco acopladas es de suma importancia, por cuanto permite la implementación y prueba de módulos en forma independiente, reduciendo así la complejidad del sistema y facilitando la reutilización de código.

Es importante que la utilización de statecharts este apoyada por una arquitectura de control que soporte su implementación en forma directa; de lo contrario el statechart pasa a ser nada más que una ayuda para el programador robots. En este sentido, una de las principales ventajas de la arquitectura utilizada es la separación, a nivel de código, entre estados y transiciones.

Aunque en la presente implementación no se utilizaron, futuros trabajos podrían explotar las capacidades de jerarquía y concurrencia de los statecharts. La jerarquía sería útil en tareas con una gran cantidad de estados, ya que permitiría agrupar varios estados en “super estados”, mientras que la concurrencia permitiría modelar tareas complejas con procesos simultáneos.

REFERENCIAS

[Arkin & MacKenzie 94] R. Arkin, D. MacKenzie, *Temporal Coordination of Perceptual Algorithms for Mobile Robot Navigation*, IEEE Transactions on Robotics and Automation, Vol 10, No. 3, pp 276-286, Junio 1994.

[Bassi 04] Danilo Bassi, Ignacio Garrido, *SARA: Una Arquitectura para Robots Móviles Minimales*, Anales de XV Congreso de la ACCA, Santiago de Chile, 2004.

[Gini 96] Maria Gini, *Design and Building Autonomous Minirobots*, Department of Computing Science, University of Minnesota, 1996.

[Harel 87] D. Harel, *Statecharts: A Visual Formalism for Complex Systems*, in Science of Computer Programming, Vol.8, pp. 231-274, Junio 1987.

[Kosecka & Christensen 95] J. Kosecka, H. Christensen, *Discrete event systems for autonomous mobile agents*, Workshop on Intelligent Control, Julio 1993.

[Maes 91] Pattie Maes, *The agent network architecture*, SIGART Bulletin, Vol 2, N°4, pp. 115-120, 1991.