Algoritmo Bug2

Luis Alberto Ballado Aradias
Cinvestav Unidad Tamaulipas
luis.ballado@cinvestav.mx

30 de abril de 2023

Resumen

El presente trabajo describe la implementación del algoritmo bug2, con ayuda de los módulos desarrollados anteriormente como el de odometría para un robot móvil de tipo diferencial y su implementación bajo el lenguaje NXC (Not eXactly C).
El algoritmo Bug2 es un algoritmo de navegación en robótica que se utiliza para que un robot encuentre su camino hacia un destino en un ambiente desconocido, sin la necesidad de utilizar un mapa previamente construido. Este algoritmo se basa en la idea de que el robot puede seguir la pared del obstáculo que se encuentra en su camino hacia el destino.

I. Introducción

El algoritmo Bug2 es un algoritmo de navegación en robótica que se utiliza para guiar a un robot hacia un destino en un ambiente desconocido sin la necesidad de tener un mapa previamente construido. Este algoritmo se basa en la idea de que el robot puede seguir la pared del obstáculo que se encuentra en su camino hacia el destino, es una mejora del algoritmo Bug1, el cual tenía la limitación de que el robot podía quedar atrapado en situaciones donde no había un camino claro hacia el destino. El algoritmo Bug2 resuelve este problema al permitir que el robot se aleje temporalmente del obstáculo para encontrar una forma de continuar hacia el destino.

Este algoritmo es especialmente útil en aplicaciones de robótica móvil, donde el robot debe navegar en un ambiente desconocido y no se dispone de un mapa previamente construido. Además, es una técnica simple y fácil de implementar, lo que lo hace adecuado para aplicaciones en las que se requiere una navegación autónoma en tiempo real. En esta técnica, el robot utiliza sensores para detectar obstáculos y para determinar la dirección y la distancia al destino. A partir de esta información, el algoritmo guía al robot para que siga la pared del obstáculo en su camino hacia el destino.

También se ha utilizado en aplicaciones de seguridad, como la vigilancia y el control de fronteras, y en aplicaciones de rescate, como la búsqueda y rescate en zonas de desastre.

En general, el algoritmo Bug2 es una técnica importante en el campo de la robótica móvil que ha permitido el desarrollo de robots autónomos capaces de navegar en ambientes desconocidos. Su simplicidad y facilidad de implementación lo hacen adecuado para una amplia gama de aplicaciones, y las mejoras y extensiones continuas lo mantienen relevante y valioso en la investigación y el desarrollo de robótica móvil.

II. Vehículos de Braitenberg

Los vehículos de Braitenberg son robots simples y autónomos diseñados para demostrar los conceptos de comportamiento emergente y robótica comportamental. Estos robots fueron desarrollados por el neurocientífico italiano Valentino Braitenberg en la década de 1980 como un medio para explorar la relación entre los sistemas sensoriales y motores en los seres vivos y los robots.

Se componen de uno o varios sensores y motores que están interconectados de manera específica para generar diferentes comportamientos. Estos comportamientos pueden ser desde simples movimientos en respuesta a la estimulación sensorial hasta comportamientos más complejos, como la evasión de obstáculos o la búsqueda de fuentes de luz.

Los vehículos de Braitenberg han sido utilizados en la investigación y la educación en robótica debido a su simplicidad y facilidad de construcción y programación. Además, han demostrado ser una herramienta valiosa para entender cómo los sistemas simples pueden dar lugar a comportamientos complejos y emergentes, también han sido utilizados en la investigación en neurociencia para estudiar los principios de la percepción y el movimiento en los seres vivos. Esto se debe a que los vehículos de Braitenberg se asemejan a los sistemas sensoriales y motores de los animales y pueden utilizarse para investigar cómo estos sistemas interactúan y generan comportamientos complejos.

En general, los vehículos de Braitenberg son una herramienta valiosa en la investigación y educación en robótica y neurociencia. Su simplicidad y facilidad de construcción y programación permiten a los estudiantes y los investigadores explorar los principios fundamentales de la robótica y la biología sin necesidad de conocimientos especializados en programación o ingeniería.

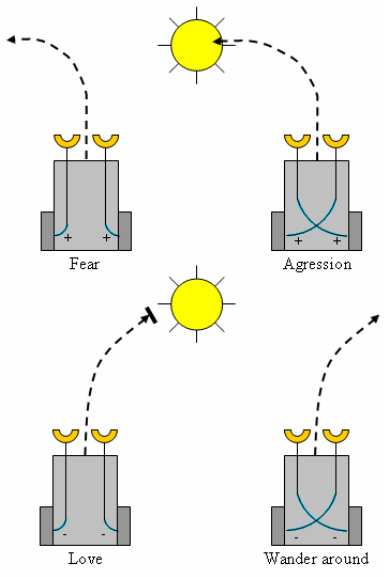

Figura 1: *Vehículos de Braitenberg*

III. Algoritmo Bug2

La relación entre el algoritmo Bug2 y los vehículos de Braitenberg radica en el hecho de que ambos utilizan principios simples para generar comportamientos complejos y emergentes. En el caso de los vehículos de Braitenberg, esto se logra mediante la interconexión de sensores y motores de manera específica. En el caso del algoritmo Bug2, esto se logra mediante el uso de un sensor y motores para permitir que el robot interactúe con su entorno de manera autónoma.

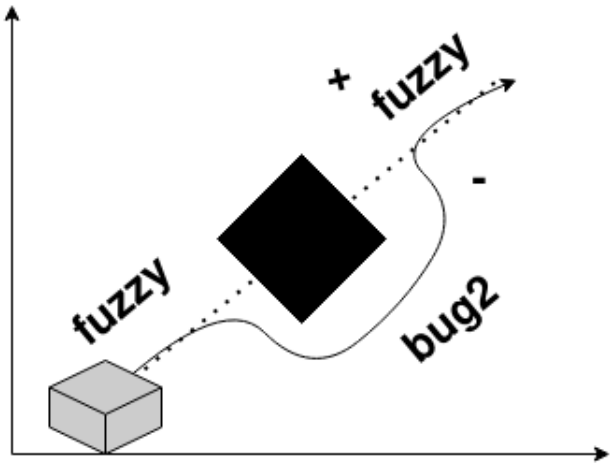

Figura 2: *Ejemplo de implementación*

El algoritmo bug2 se describe a continuación

while *no en el objetivo* **do**
  seguir navegando ; `/* si aún no llegamos al objetivo */`
  if *detectamos un obstáculo* **then**
    seguir el obstáculo por su borde hasta encontrar el punto más cercano que nos regrese al control de navegacion;
  end
  else
    apagar_motores;
    llegué al objetivo;
  end
end

Algorithm 1: Algoritmo Bug2

Partiendo de la sencillez del algoritmo, lo que falta es poder identificar la región donde pueda cambiar hacia el control difuso nuevamente. Para ello seguimos haciendo uso de los valores que nuestro odometro nos brinda en todo momento.

I. Cambio de control

Una de los objetivos son los cambios de control.

Partiendo de la navegación, al detectar un obstáculo.

1. Guardar la posición cuando se detecta un obstáculo.
2. Calcular el vector x,y respecto a la distancia entre el punto donde se detectó el obstáculo y el punto final.
3. Rotar el vector x,y 90°
4. Efectuar la diferencia entre las lecturas del odometro con la posición donde se dectó el obstáculo.
5. Multiplicar la diferencia con los elementos del vector para obtener el vector resultante
6. Identificar el cambio de signo del resultado anterior

IV. Implementación

Nuestra implementación es la combinación de las practicas anteriores como odometría y el control difuso para ir del origen a un punto x_f, y_f.

Partimos de la necesidad de cambiar de controles respecto a la situación que se presenta.

- Control de lógica difusa: Navegar de punto $A \rightarrow B$
- Control Boundary Following: Rodear el obstáculo
- Cálculo para identificar si rodie el obstáculo

A continuación se describe el pseudocódigo que describe el comportamiento de la implementación

```
Comenzar con fuzzy_control();
while true do
    seguir_con_control_fuzzy ;                  /* si aún no llegamos al objetivo */
    if a >4 then
        if sensor_frente <30 then
            Apagar_Motores ;   /* si el sensor de enfrete detecta un obstáculo */
            definir x0, y0 ;            /* guardar x,y donde encontre un obstáculo */
            goto follow_b ;                  /* cambiar al control follow_boundary */
        end
    end
    else
        apagar_motores;
        llegué al objetivo;
    end
end
```

Algorithm 2: Cambio entre controles

Para el control de follow boundary se tiene

```
calculos_signo = 0 ;              /* variable para detectar los cambios de signo */
Rotar_robot_30Grados() ; /* rotar robot para que el sensor de izquierda obtenga
 lecturas */
while true do
    if sensor_izquiera <10 then
        error_izquierda ← 0
    end
    else if sensor_izquierda <100 then
        error¡zquierda ← (30 − us_izquierda) ∗ 2
    end
    if sensor_izquierda >100 then
        error_izquierda = -10;
    end
    potencia_motor_izquierda = 50 + error_izquierda;
    potencia_motor_derecha = 50 - error_izquierda;
    vx = (x_f − x_0) ;                                  /* calcular vector x_0x_f */
    vy = (y_f − y_0) ;                                  /* calcular vector y_0y_f */
    n[0] = -vy ;                                            /* rotar 90 grados */
    n[1] = vx;
    ODO_X_0 = x-x_0;
    ODO_Y_0 = y-y_0;
    calculos_signo = (ODO_X_0 ∗ n[0]) + (ODO_Y_0 ∗ n[1]);  /* identificar el cambio de
     signo */
    if (calculos_signo >= 0 && prev_ <0) || (calculos_signo <0 && prev_ >= 0) then
        goto fuzzy_control ;  /* cambiar al control difuso para seguir al objetivo
         */
    end
    prev_ = calculos_signo;
end
```

Algorithm 3: Rodear obstáculo

V. Resultados

Al tener el ladrillo lego lleno de programas anteriores, no pensé que la carga del programa pudiera afectar la carga. Teniendo que eliminar código muerto sin ejecutar, faltando incluir la vuelta de <90 grados al momento de cambiar el mando de control del difuso a control follow boundary.

En los videos se puede mostrar el funcionamiento. Nuevamente nos faltó un ajuste a la potencia de los motores que parte de la matriz de conocimientos del control difuso.
ver video bug2

ver video boundary following

Ver código fuente Github

VI. Conclusiones

Sin embargo, el algoritmo Bug2 presenta algunas limitaciones que deben tenerse en cuenta. Una de las limitaciones más significativas es que puede no encontrar la ruta más corta hacia el destino. Esto se debe a que el algoritmo sigue la pared del obstáculo, lo que puede llevar al robot por un camino más largo hacia el destino en lugar de un camino más directo.

Otra limitación del algoritmo Bug2 es que existe la posibilidad de quedar atrapado en un ciclo infinito si el robot no puede encontrar una forma de salir de una situación de obstáculo. Esto puede ocurrir si el robot no puede encontrar una ruta que lo lleve al destino después de alejarse del obstáculo.

A pesar de estas limitaciones, el algoritmo Bug2 sigue siendo una técnica valiosa en robótica móvil debido a su simplicidad y facilidad de implementación. Además, su capacidad para navegar en ambientes desconocidos lo hace especialmente útil en aplicaciones donde no se dispone de un mapa previamente construido.

A lo largo del desarrollo de la práctica se hizo uso de controles para el salto entre las funciones y controles, que aunque no se hicieron de una forma elegante. Se puede observar el buen funcionamiento y cambio entre los dos controles.

Referencias

[1] Intalación NXC en LINUX, http://ubuntudaily.blogspot.com/2011/03/using-lego-mindstorms-nxt-with-ubuntu.html

[2] Repositorio Compilador NBC utilizado, https://github.com/pierre-24/nbc-compiler

[3] Documento para evitar sudo en NXC, https://bricxcc.sourceforge.net/nbc/doc/nxtlinux.txt

[4] Comando para instalar libusb-dev, https://howtoinstall.co/en/libusb-dev

[5] Presentación Odometría, http://www.kramirez.net/Robotica/Material/Presentaciones/Odometria.pdf

[6] Modelo Cinemático de un robot móvil tipo diferencial y navegación a partir de la estimación odométrica,https://www.redalyc.org/pdf/849/84916680034.pdf VALENCIA V., JHONNY A.; MONTOYA O., ALEJANDRO; RIOS, LUIS HERNANDO

[7] Modelo cinemático de un robot móvil implementado con LEGO NXT para un sistema de localización indoor diseñado en Labview, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiu0_So28_9AhViDkQIHRlXDtcQFnoECBQQAQ&url=https%3A%2F%2Frevistas.udistrital.edu.co%2Findex.php%2FTecnura%2Farticle%2Fdownload%2F6810%2F8394%2F30717&usg=AOvVaw2PsCrkFGkk_nGN-G084B11

[8] Notas de clase, Robótica Móvil Inteligente, Dr. José Gabriel Ramírez Torres, Enero-Abril 2023

[9] Braitenberg Vehicles as Computational Tools for Research in Neuroscience, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7525016/, Danish Shaikh and Ignacio Rañó, 16 09 2020