2020

Trabajo Final de la Asignatura

MEMORIA

LÓPEZ GARCÍA, IGNACIO
MAYORAL MARÍN, SARA

UAH - GIC | Percepción y Control

Índice

Introducción

En esta práctica se va a desarrollar un algoritmo por el cual el robot, mediante el uso de los sónares y el láser navega a través de un laberinto, haciendo un mapeado del mismo y escapando de él.

Esta práctica la hemos enfocado de manera que el robot sigue dos fases principales: la fase de mapeo y la de salida del laberinto, y durante ambas el robot se guiará por los sensores de odometría, el láser y los actuadores del motor. Los topics serán utilizados como variables globales accesibles a todas las funciones de nuestro programa, para así hacerlo más sencillo.

1 - Fase de mapeo

Durante la fase de mapeo, como el nombre indica, reuniremos información sobre el laberinto en cuestión para construir un mapa de este y crear una ruta de escape durante el proceso.

Tendremos una estructura llamada casilla que almacenará información sobre las casillas entre las que navegaremos, tal como las paredes (una subestructura que contendrá las cuatro direcciones de las casillas), las cuales se reconocerán de manera independiente a la orientación en cada casilla del robot, y si las casillas han sido visitadas, esta estructura se puede ver en la *figura 1*. Almacenaremos toda esta información en un array de casillas que representará el mapa.

Para empezar el bucle de mapeo reconoceremos primero la casilla en la que estamos mediante la lectura de los sensores de odometría aplicando algunas transformaciones para dividir el mapa en casillas de 2x2. Tomaremos la distancia de la odometría, la dividiremos entre 2 esta será nuestra casilla, la cual dependiendo de si el resto es mayor que cero, nos indicará que invadimos la siguiente casilla.

```
% Usaremos una estructura mapa que contiene: casillas y en estas
% casilla, si ha sido visitada
paredes.arriba=0;
paredes.abajo=0;
paredes.derecha=0;
paredes.izquierda=0;
casilla.paredes = paredes;
casilla.visitada = 0;
mapa=[casilla casilla casilla; casilla casilla casilla; casilla c
```

Figura 1. Estructuras de datos para nuestro mapa

1.2 - Lectura de paredes

A continuación, si la casilla en la que nos encontramos no ha sido visitada, haremos una lectura de las paredes que rodean al robot en la casilla actual. Como utilizaremos el láser, sabremos a qué orientación responde el rayo frontal (200 de los 400 del láser), calcularemos el ángulo complementario a la orientación del robot y lo transformaremos mediante una regla de tres a su correspondencia en rayos del láser, sabiendo que un giro de (π/2) radianes corresponde a 100 rayos (ver *figura 2*). Una vez tenemos este desplazamiento dependiendo del cuadrante en el que estemos aplicaremos distintos desplazamientos para calcular los rayos correspondientes a las direcciones de las paredes. La lectura dejará un margen de error de 30 centímetros por si el movimiento del robot no fuera lo suficientemente preciso.

Tras la lectura de paredes comprobaremos las casillas mapeadas para saber si hemos visitado todas las casillas (3x7=21) para acabar la fase de mapeo. En caso de lograrlo acabaremos la fase, de lo contrario, calcularemos la siguiente casilla a la que avanzaremos.

```
% Sacamos los rayos que ocupa el complementario
rayos_compl = floor(100*complementario/(pi/2));
valor = 200 + rayos_compl;

%% Indicación de rayos según el cuadrante de la orientación
% Primer cuadrante
if(0<=yaw<=pi/2)
    arriba = mod(valor,401);
    izq = mod(arriba+100,401);
    der = mod(arriba-100,401);
    abajo = mod(arriba-200,401);

% Segundo cuadrante
elseif(pi/2<yaw<=pi)
    izq = mod(valor,401);
    arriba = mod(izq-100,401);
    der = mod(izq-200,401);
    abajo = mod(izq-300,401);
% Tercer cuadrante
elseif(-pi/2>yaw>=-pi)
    abajo = mod(valor,401);
    izq = mod(abajo-100,401);
    der = mod(abajo+100,401);
    arriba = mod(abajo+200,401);
% Cuarto cuadrante
else
    der = mod(valor,401);
    arriba = mod(der+100,401);
    izq = mod(der+200,401);
    abajo = mod(der-100,401);
end
```

Figura 2. Código del apartado de selección de rayos para la lectura de paredes

1.3 – Toma de decisiones

Esta es una parte clave de la navegación que nos indicará a qué casilla desplazarnos de las adyacentes.

Esta función priorizará siempre los movimientos a casillas no visitadas marcando un valor numérico en una variable llamada dirección, donde el valor 1 es desplazamiento a la derecha, el valor 2 es desplazamiento hacia arriba, el valor 3 es el desplazamiento hacia abajo y el valor 4 hacia la izquierda. En caso de que todas las casillas adyacentes hayan sido visitadas se activará una variable a modo de flag indicador de que necesitamos tomar medidas teniendo en cuenta la casilla visitada anteriormente para evitar que el robot de vueltas en círculos, a menos que se encuentre en un callejón sin salida.

Esta función se divide en bloques de código que indican las acciones del robot según su posición con respecto a los bordes del mapa.

Si no se encuentra en los bordes priorizará el movimiento vertical, si está en los bordes laterales igual, a menos que esté en una esquina, la cual condicionará el movimiento, y si está en los bordes superior o inferior priorizará el movimiento horizontal.

1.4 – Trazado de salida

Tras la lectura de la casilla se comprobará si es la salida del laberinto, esto se hará, aunque ya se haya visitado la salida, por si se da el caso de encontrar una segunda salida que sea más cercana. En ese momento se buscará la casilla que corresponde al exterior del laberinto.

Después de tomar la decisión de movimiento y, si se ha encontrado la salida se trazará una ruta de salida que se corregirá en cada ciclo, introduciendo en un array las posiciones que deberemos seguir para salir del laberinto.

Como se traza sobre la marcha, el algoritmo que utilizamos para hacer una ruta de salida corregirá la ruta si volvemos a una casilla que ya hemos visitado para evitar que el robot de vueltas en círculos y la ruta sea lo más corta posible, en la *figura 3* se puede ver parte del código para la toma de decisiones y la corrección de la ruta de salida.

```
if(puntos>0)
    coincidencia = 0; %Para saber si ha funcionado el proceso
    % Bucle desde puntos que decrementa de 1 en 1 hasta 0
    for index = puntos:-1:1
        % Si la casilla en la que estamos ya esta dentro de la rut
        % escape ignoramos todas las demás para ahorrar vueltas
        if(ruta(index,1)== actual.X && ruta(index,2)==actual.Y)
            puntos=index;
            coincidencia=1;
            break;
        end
    end
    % Si no coincide con la ruta de salida se añade
    if(coincidencia==0)
        puntos = puntos+1;
        ruta(puntos,1)=actual.X;
        ruta(puntos,2)=actual.Y;
    end

else
    puntos = puntos+1;
    ruta(puntos,1)=actual.X;
    ruta(puntos,2)=actual.Y;
end

resultado.puntos = puntos;
resultado.ruta = ruta;
```

Figura 3. Código de la formación y corrección de ruta de salida

1.5 – Navegación entre casillas

Esta es otra parte clave, ya que utilizaremos un controlador de posición proporcional como el del primer apartado de la práctica 2.

Como nuestro robot se va a mover en horizontal o vertical, empezamos haciendo una corrección de la orientación del robot para facilitar el movimiento y evitar choques con las paredes. Dependiendo de si el movimiento será horizontal o lateral, el robot se colocará en una orientación objetivo que obtendremos utilizando un controlador proporcional base para alcanzarla, ver figura 4.

Tras corregir la orientación comenzaremos a utilizar el controlador de posición, el cual utilizará el error de orientación y el de posición para asignar una velocidad lineal y angular para nuestro desplazamiento, ver *figura 5*.

```
while( abs(error) >= 0.05 )

    ori = odom.LatestMessage.Pose.Pose.Orientation;
    yaw = quat2eul([ori.W ori.X ori.Y ori.Z]); % Transformació
    yaw = yaw(1); % Solo nos interesa esa orientación
    error = obj - yaw;

    % Definición de cuadrante
    if(error>pi)
        error = error - 2*pi;
    elseif(error<-pi)
        error = error + 2*pi;
    end
    % Para limitar la velocidad de giro a 1 rad/s
    if(abs(error)>2)
        error=error/abs(error);
    end
    msg_vel.Angular.Z = error*0.65;
    send(pub,msg_vel);
    waitfor(r);
end
msg_vel.Angular.Z = 0;
send(pub,msg_vel);
```

Figura 4. Código del controlador de corrección de orientación

```
while((eje>-1) && (abs(Pe)>=0.11))
    % Variables actuales
    pos = odom.LatestMessage.Pose.Pose.Position;
    %% Cálculo de error
    % Error de posición y orientación
    Pe = sqrt((pos.X-destino.X)^2+(pos.Y-destino.Y)^2);
    Oe = atan2(destino.Y-pos.Y,destino.X-pos.X)-yaw;

    % Contolamos los cambios de cuadrante del error de orient
    if(Oe>pi)
        Oe = Oe - 2*pi;
    elseif(Oe<-pi)
        Oe = Oe + 2*pi;
    end
    %% Consignas de velocidades
    consigna_vel_ang = 0.45*Oe;
    consigna_vel_lin = 0.60*Pe;

    if(consigna_vel_ang>1)
        consigna_vel_ang=1;
    end
    if(consigna_vel_lin>1)
        consigna_vel_lin=1;
    end

    % Controlamos que no de vueltas en círculos atascado
    if(consigna_vel_ang>consigna_vel_lin)
        consigna_vel_ang=consigna_vel_lin-0.1;
    elseif(consigna_vel_ang<-consigna_vel_lin)
        consigna_vel_ang=-consigna_vel_lin+0.1;
    end
    %% Aplicamos consignas de control
    msg_vel.Linear.X= consigna_vel_lin;
    msg_vel.Angular.Z= consigna_vel_ang;
    send(pub,msg_vel);

    waitfor(r);
end
```

Figura 5. Código del controlador de posición

2 - Fase de salida

Durante esta fase, simplemente, utilizaremos el array que rellenamos durante la fase de mapeo.

Utilizaremos un bucle para avanzar de casilla en casilla por el recorrido que hemos marcado en el array y utilizando el índice “puntos”, que también corregimos junto con la ruta durante la etapa de mapeo, ver *figura 6*.

Una vez nuestro índice se ha reducido a cero, solo debemos enviar a nuestro robot a la casilla salida que calculamos y habrá acabado el programa.

```
%% Bucle de salida
while(fuera==0)
    % Cargamos la casilla objetivo
    obj.X = ruta_salida(puntos,1);
    obj.Y = ruta_salida(puntos,2);
    % Nos movemos
    avanzar(obj);
    % Decrementamos el indicador de casilla
    puntos=puntos-1;

    % Si ya hemos recorrido todos los puntos para salir
    if(puntos==0)
        avanzar(casilla_salida);
        fuera=1;
    end
end
```

Figura 6. Código del bucle de salida

Conclusión

Durante el desarrollo de esta practica nos hemos enfrentado a diferentes desafíos. Hay muchas posibles soluciones para resolver el mismo problema. En este trabajo final se nos pedía realizar el mapeado de un laberinto para posteriormente salir de él, todo eso realizado dentro de un tiempo límite y con una velocidad máxima para el robot permitida.

Con esta premisa hay que buscar soluciones que no solo sean precisas, si no que además sean eficientes, debido a la limitación temporal. Para poder llegar a la solución se requiere de varias fases, construyendo la solución poco a poco. Un claro ejemplo de esto es la decisión de dividir el mapa en casillas. Inicialmente, era un método lento, que requería pararse en cada casilla y analizarla; pero, por otro lado, este diseño nos permite garantizar que el robot ha realizado un correcto mapeado del laberinto. Avanzar de casilla en casilla, además, nos permite minimizar posibles errores tanto de rotación como de posición. Este diseño inicial sacrificaba la velocidad por la precisión, es decir, garantizaba que el robot recorría el mapa, pero no lo hacía dentro del tiempo límite.

Para arreglar el problema de la velocidad, se mejoró el algoritmo de la toma de decisiones para elegir la siguiente casilla que visitar además de hacer que el robot guardase en memoria la ruta inversa una vez llega a la casilla de salida, si todavía no se ha realizado el mapeado completo. Esto mejoró enormemente el tiempo que tardaba en recorrer el laberinto y llegar a la salida.

Hay que tener en cuenta que este controlador esta diseñado para un tipo de mapa muy específico, las paredes son paralelas o perpendiculares entre sí y los pasillos tiene siempre el mismo ancho, si nuestro laberinto no cumple esas condiciones el robot no podría navegar de forma adecuada porque el algoritmo no se ha diseñado para ese tipo de mapas.

Como conclusión a este trabajo, se puede decir que en este tipo de problemas de percepción y control hay que diseñar las soluciones para el problema especifico y, aunque hay directrices y estándares, estos no garantizan la solución al problema.