Curso Doctorado

Práctica de la sesión 1: Filtro de Kalman

Domingo Gallardo

1. Problema

En la sesión de prácticas de hoy debes implementar un programa MATLAB con el que se pueda probar el estimador $\hat{\eta}$ de una longitud η visto en las transparencias.

Recordemos que este estimador se construye mediante una combinación lineal de las medidas realizadas por los distintos sensores y que tiene como propiedades ser no sesgado (su media coincide con la longitud) y tener la mínima varianza. Aquí están las transparencias:

We measure an object of length η with n instruments of varying accuracies. The results of the measurements are n random variables

$$x_i = \eta + \nu_i \qquad E[\nu_i] = 0 \qquad Var[\nu_i] = E[\nu_i^2] = \sigma_i^2$$

where ν_i are the measurement errors which we assume independent with zero mean. Our problem is to determine the unbiased, minimum variance, linear estimation of η.

We wish to find n constants α_i such that the sum

$$\hat{\eta} = \alpha_1 x_1 + \cdots + \alpha_n x_n$$

is a random variable with mean $E[\hat{\eta}] = \alpha_1 E[x_1] + \cdots + \alpha_n E[x_n] = \eta$ and its variance $V = \alpha_1^2\sigma_1^2 + \cdots + \alpha_n^2\sigma_n^2$ is minimum, subject to the constraint:

$$\alpha_1 + \cdots + \alpha_n = 1$$

To solve this problem, we note that

$$V = \alpha_1^2\sigma_1^2 \ldots \alpha_n^2\sigma_n^2 - \lambda(\alpha_1 \ldots \alpha_n - 1)$$

for any λ (Lagrange multiplier). Hence V is minimum if

$$\frac{\delta V}{\delta \alpha_i} = 2\alpha_i\sigma_i^2 = 0 \qquad \alpha_i = \frac{\lambda}{2\sigma_i^2}$$

Inserting into the constraint and solving for λ, we obtain

$$\frac{\lambda}{2} = V = \frac{1}{1/\sigma_1^2 + \cdots + 1/\sigma_n^2}$$

Hence

$$\hat{\eta} = \frac{x_i/\sigma_1^2 + \cdots + x_n/\sigma_n^2}{1/\sigma_1^2 + \cdots + 1/\sigma_n^2}$$

2. Implementación

Debes implementar unos comandos MATLAB que permitan probar este estimador, asi como visualizar las longitudes realizadas por los distintos instrumentos. Por ejemplo, podrías implementar las funciones `measurelength`, `estimatelength` y `plotlength`. La primera función realiza las medidas con distintas varianzas, la segunda la estimación de la longitud y la tercera muestra una figura con los resultados.

Ejemplos MATLAB

Función que construye un array de estructuras

```
function result=puntos(n)
% Devuelve un array de estructuras con los
% campos: x y error
function result=puntos(n)
for i=1:n
   result(i).x = 5+i;
   result(i).error = 0.5;
end
```

Función que dibuja una gráfica mostrando datos y errores

```
function plotpuntos(puntos)
% plotpuntos dibuja una gráfica con los valores x de
% los puntos y sus errores, ajustando el eje Y a los límites
% de los valores mostrados

% cálculo valores máximo y mínimo
min=puntos(1).x; imin=1;
max=puntos(1).x; imin=1;
for i=1:length(puntos)
   if (puntos(i).x < min)
      min=puntos(i).x;
      imin=i;
   else
      if (puntos(i).x > max)
         max=puntos(i).x;
         imax=i;
      end
   end
end
```

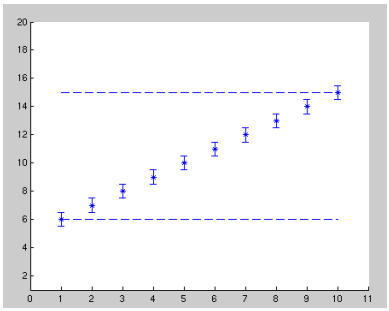
```
% límites a visualizar
sup=zeros(1,length(puntos))+max;
inf=zeros(1,length(puntos))+min;

% dibujo de la gráfica
hold on;
axis([0,length(puntos)+1,min-5,max+5]);
errorbar([puntos.x],[puntos.error],'*');
plot(1:length(puntos),sup,'--');
plot(1:length(puntos),inf,'--');
hold off;
```

La gráfica resultante de la evaluación de los comandos

```
>> p=puntos(10);
>> plotpuntos(p);
```

es la siguiente



3. Documentación a entregar

Debes entregar un informe de la práctica, que contenga el código de los programas MATLAB y algunos ejemplos que muestren gráfica y analíticamente la corrección del estimador. Es recomendable que los ejemplos tengan un número considerable de mediciones (más de 50).

Curso Doctorado

Práctica de la sesión MultiRobots

María Isabel Alfonso

1. Problema

En la sesión de prácticas de hoy debes implementar un programa con Player/Stage que permita simular un sistema multirobot con coordinación centralizada.

El sistema (ver figura 1) está compuesto por un entorno cerrado, 4 alarmas (objetos de color rojo, verde, azul claro y amarillo) situadas en determinadas posiciones fijas del entorno, un robot controlador (robot rojo) situado en el centro del entorno y 6 robots (azules) situados aleatoriamente en distintas posiciones iniciales.

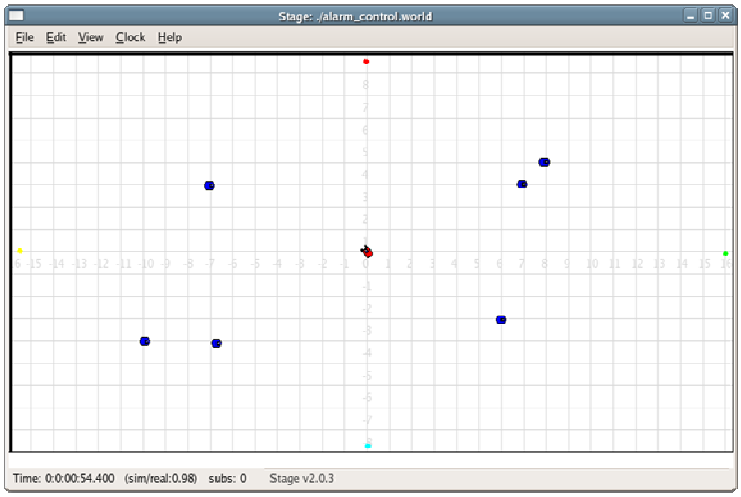


Figura 1. Estado inicial del sistema.

Inicialmente todos los robots (excepto el controlador) se mueven aleatoriamente por el entorno simplemente intentando evitar obstáculos. Cuando el controlador detecte que una alarma se ha activado, rotará sobre sí mismo y se parará en la dirección de la alarma activada. En ese momento, solicitará al resto de robots que le comuniquen sus posiciones respectivas con el fin de averiguar cuál es el que está más cerca de dicha alarma. Después el controlador enviará una orden al robot más cercano a la alarma para que se dirija a ella.

El estado final del sistema está reflejado en la figura 2, en donde las 4 alarmas han sido activadas y 4 robots se han dirigido a ellas. El controlador queda señalando a la última alarma activada y los otros 2 robots siguen moviéndose aleatoriamente por el entorno.

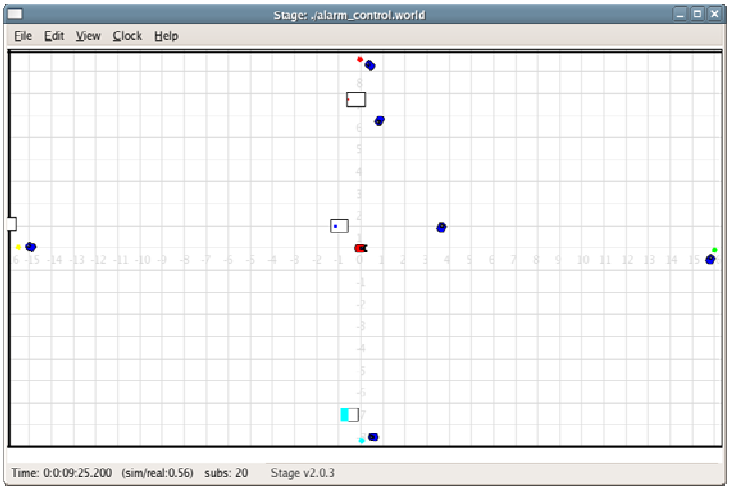


Figura 2. Estado final del sistema.

2. Implementación

Debes implementar un programa en C++ utilizando player/stage que permita simular el sistema descrito anteriormente. Para ello, dispones de la estructura general del programa principal y de un API de funciones que te facilitarán la labor.

La aplicación proporcionada implementa una arquitectura de control basada en comportamientos que utiliza un autómata de estados finitos con transiciones activadas principalmente mediante comunicación por paso de mensajes entre robots. En concreto se proporcionan los siguientes ficheros:

- alarm_control.cc : cuerpo principal del programa y autómata de estados finitos.
- alarm_control.h : declaración de tipos, constantes y variables globales.
- initial_pose.h : funciones para leer del fichero de configuración del entorno la pose inicial de los robots y las alarmas.
- locomotion.h : funciones para determinar el movimiento de un robot en el entorno evitando obstáculos.
- communication.h: funciones para la comunicación entre robots usando sockets.

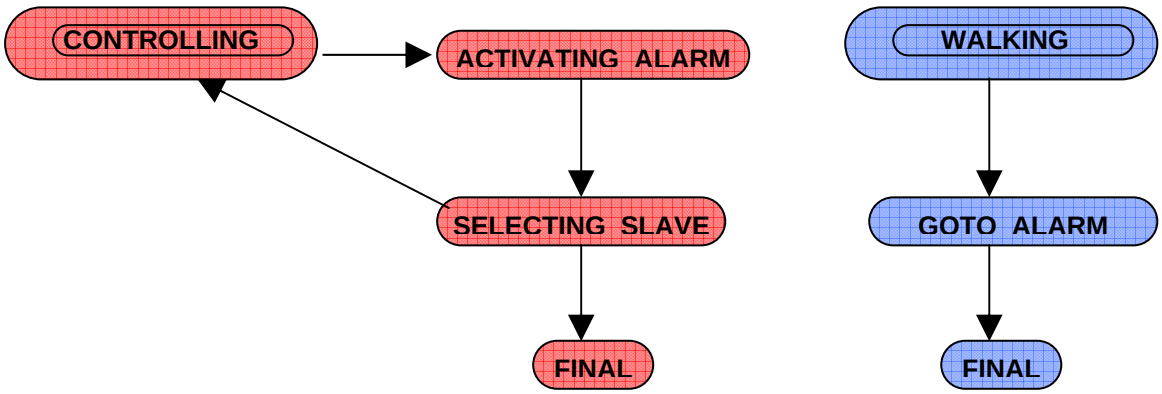
También se proporcionan distintos ficheros de configuración del entorno para player/stage y para la compilación del programa.

A continuación se muestra un ejemplo de implementación de envío y recepción de mensajes entre robots:

```
// Envío de mensajes
sprintf(msg_send, "%c%c%d", MSG_TYPE, CHAR_SEPARATOR, id_robot);
Send_Message_OneToMany(sock_send, msg_send);  // envío al resto de robots
Send_Message_OneToOne(sock_send, msg_send, id_receiver);  // envío a otro robot

// Recepción de mensajes
Receive_Message(sock_recv, msg_recv);
msg_code = msg_recv[0];
if (msg_code == MSG_TYPE)
   sscanf(msg_recv, "%*c %*c %d", &id_robot); // si sólo interesa el id del emisor
```

Se sugiere la implementación de dos autómatas con los siguientes estados y transiciones:



ROBOT CONTROLADOR **ROBOT ESCLAVO**

3. Entregables

Debes entregar el/los fichero/s fuentes que contengan el código del programa C++ que implementa el sistema propuesto. También se debe entregar una breve documentación que describa los autómatas diseñados, indicando las acciones que se llevan a cabo en cada estado y las reglas de activación de las transiciones. Por último, y en su caso, una breve documentación que describa si se ha mejorado el código o se han incorporado nuevas ideas al problema propuesto.