Percepción y Control
PRÁCTICA Tema 3

Diseño de algoritmos de control clásico para la navegación de robots móviles II

Ingeniería de Computadores

Pedro Barquín Ayuso
Miguel Ballesteros García

1.-Diseño de un control de posición

•Se pide el diseño de controladores de tipo proporcional (P) y también controladores de tipo proporcional-integral (PI).

Adjuntamos el los bucles de control de cada controlador, en los archivos de la entrega se puede ver el código completo.

P:

```
//BUCLE DE CONTROL:
while (1){
	playerc_client_read(client);
	//Leemos la posición actual del robot
	Iw=position2d->pa;
	Ix=position2d->px;
	Iy=position2d->py;
	//Calculamos los errores lateral y de orientación
	de=sqrt(((Ix-xp)*(Ix-xp))+((Iy-yp)*(Iy-yp)));
	oe=atan2((yp-Iy),(xp-Ix))-Iw;
	if(oe>PI){
		oe=oe-PI*2;
	}else if(oe<(-PI)){
		oe=oe+PI*2;
	}
	//Calculamos las señales de control (velocidades V y W)
	V=Kde*de;
	W=Koe*oe;
	//Enviamos las velocidades al robot (angular y lineal)
	playerc_position2d_set_cmd_vel(position2d,V,0.0,W,1);
	//playerc_position2d_set_cmd_pose(position2d,xp,yp,DTOR(45),1);
	//Dibujamos un punto en la posición actual del robot
	playerc_client_read(client);
	puntos->px=position2d->px;
	puntos->py=position2d->py;
	playerc_graphics2d_draw_points (graficos, puntos, 1);
	//Si llegamos cerca de los bordes del mapa, terminamos el programa
	if (position2d->px>20 | position2d->px<-20 | position2d->py>20 | position2d->py< -20)
		break;
}
```

Pi:

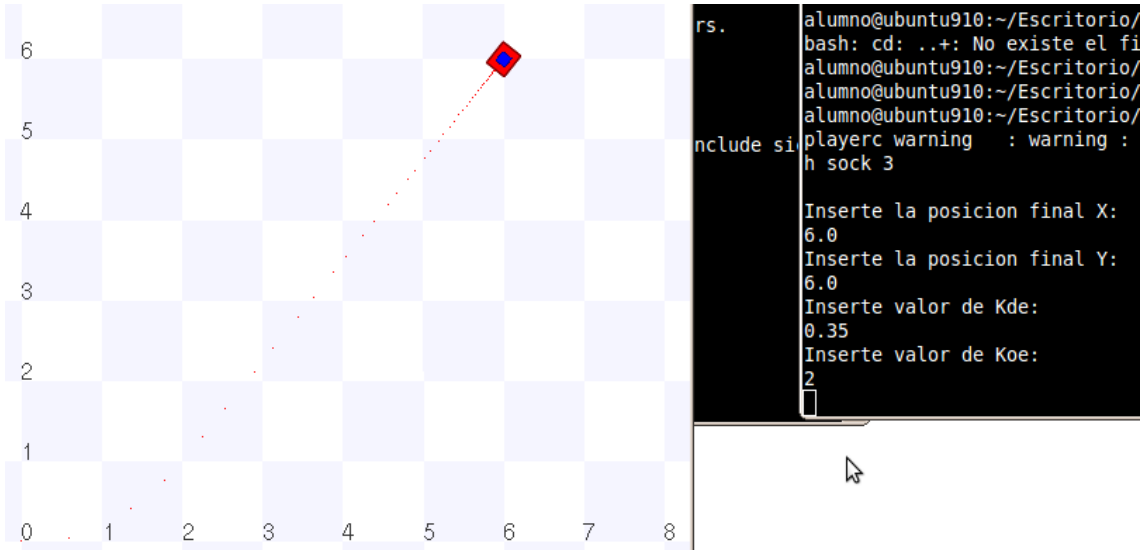
```
//BUCLE DE CONTROL:
while (1){
	playerc_client_read(client);
	//Leemos la posición actual del robot
	Iw=position2d->pa;
	Ix=position2d->px;
	Iy=position2d->py;
	//Calculamos los errores lateral y de orientación
	de=sqrt(((Ix-xp)*(Ix-xp))+((Iy-yp)*(Iy-yp)));
	oe=atan2((yp-Iy),(xp-Ix))-Iw;
	if(oe>PI){
		oe=oe-PI*2;
	}else if(oe<(-PI)){
		oe=oe+PI*2;
	}
	//Calculamos las señales de control (velocidades V y W)
	error=error+oe;
	V=Kde*de;
	W=(Koe*oe)+(Kip*error);
	//Enviamos las velocidades al robot (angular y lineal)
	playerc_position2d_set_cmd_vel(position2d,V,0.0,W,1);
	//Dibujamos un punto en la posición actual del robot
	playerc_client_read(client);
	puntos->px=position2d->px;
	puntos->py=position2d->py;
	playerc_graphics2d_draw_points (graficos, puntos, 1);
	//Si llegamos cerca de los bordes del mapa, terminamos el programa
	if (position2d->px>20 | position2d->px<-20 | position2d->py>20 | position2d->py< -20)
		break;
}
```

•El valor de las ganancias de los controladores debe ser ajustado experimentalmente mediante simulación para conseguir un comportamiento adecuado del sistema con una respuesta rápida y sin grandes oscilaciones en el movimiento hacia el punto de destino.

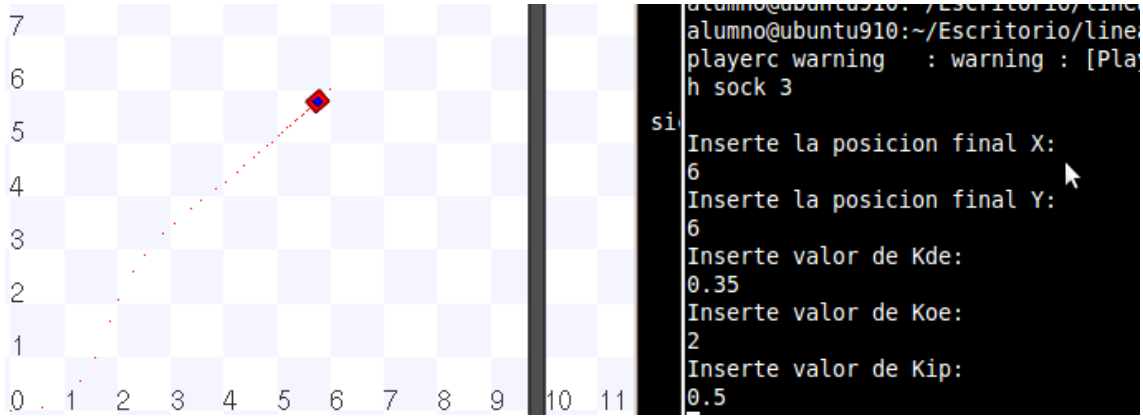
•Observe de que manera influye e valor de las ganancias del controlador, tanto P como Pi en el movimiento del robot hacia el punto de destino (valores grandes y pequeños). Documéntelo con varios ejemplos y justifique la respuesta.

Hemos ajustado los valores para conseguir un comportamiento adecuado del robot, obteniendo los siguientes resultados.

P:

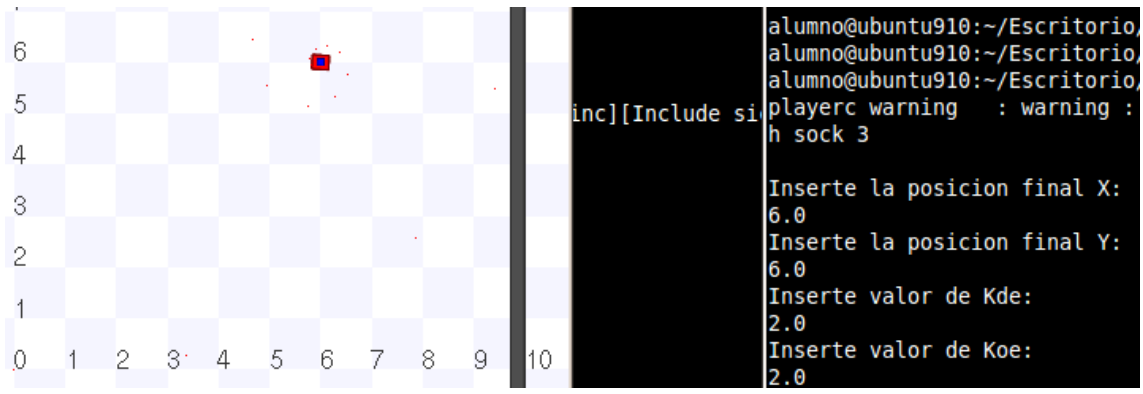


Pi:

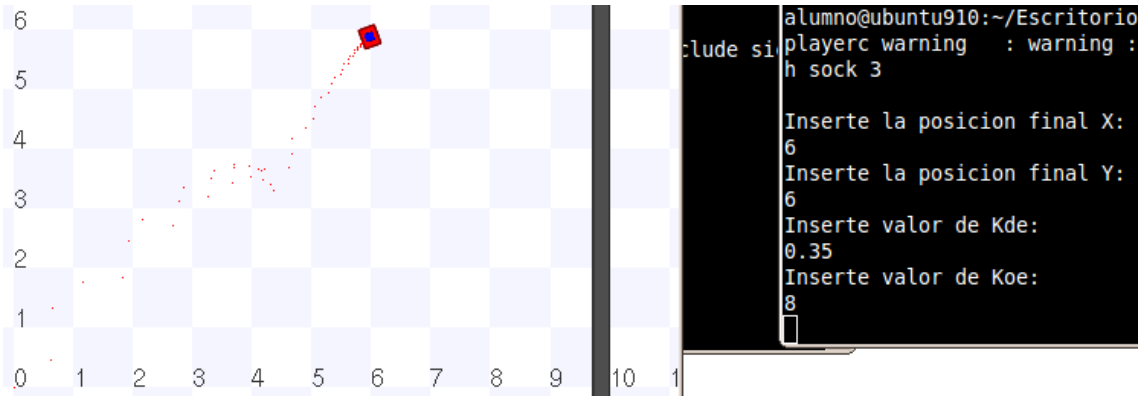


Si ponemos los valores de las variables demasiado altos dentro del controlador P obtenemos los siguientes resultados:

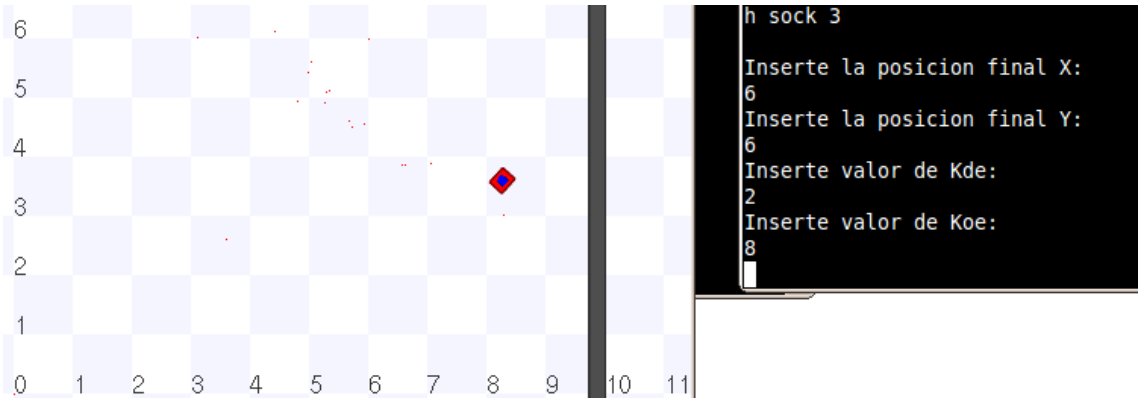
Kde: la velocidad de movimiento es demasiado elevada por lo que no le da tiempo a corregir bien el ángulo y describe una espiral.



Koe: al ser el parámetro del ángulo demasiado elevado cambia constantemente de dirección hasta que se va aproximando al punto de destino.

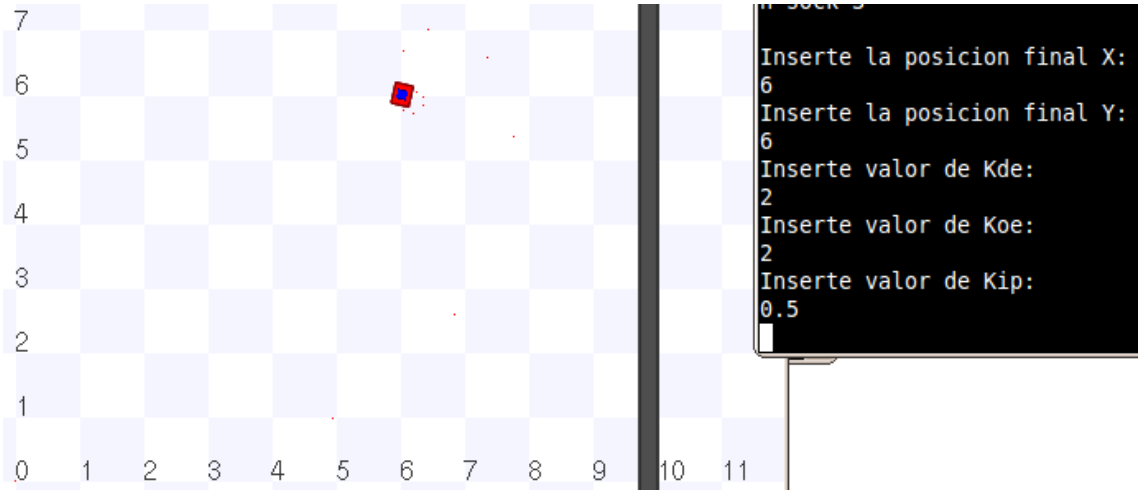


Kde y Koe: se vuelve impredecible

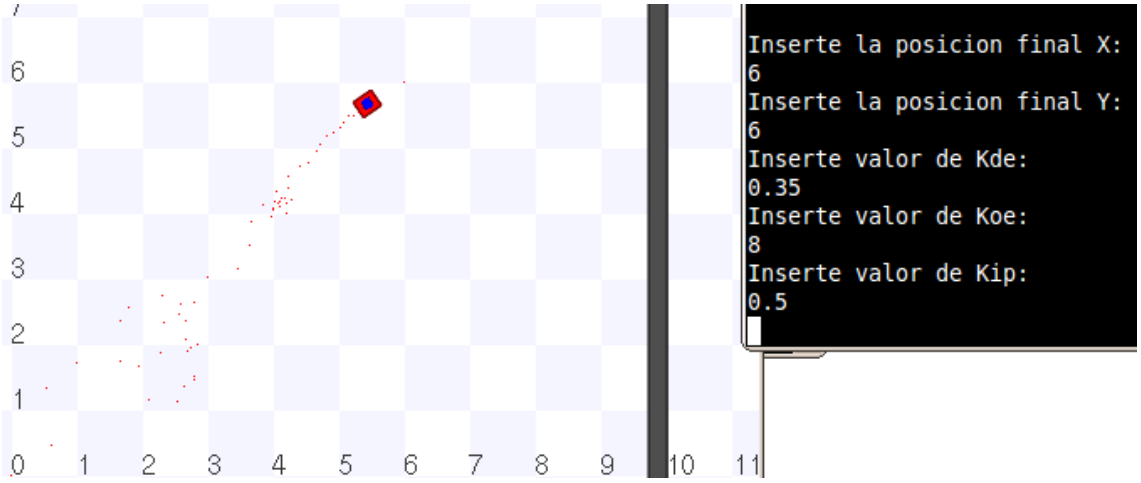


Efectuamos el mismo procedimiento para explicar el controlador Pi

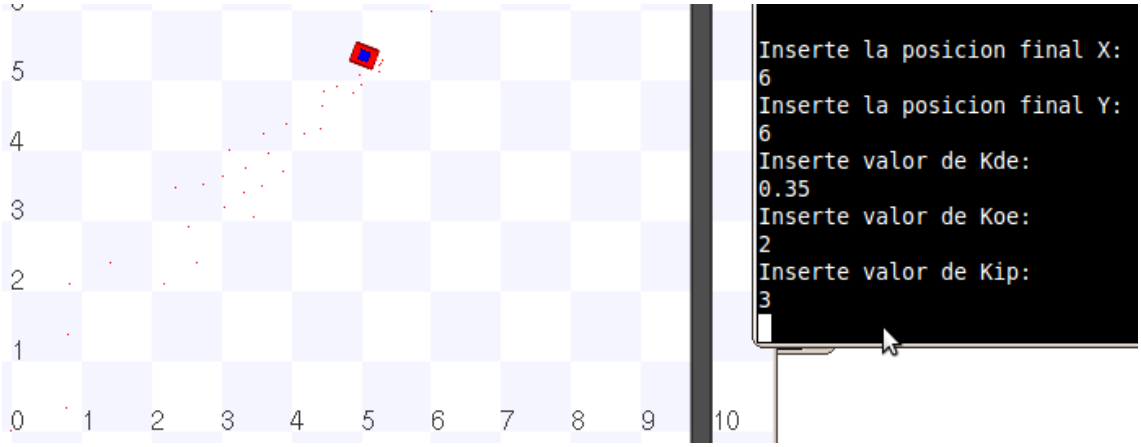
Kde: realiza un movimiento similar al del P



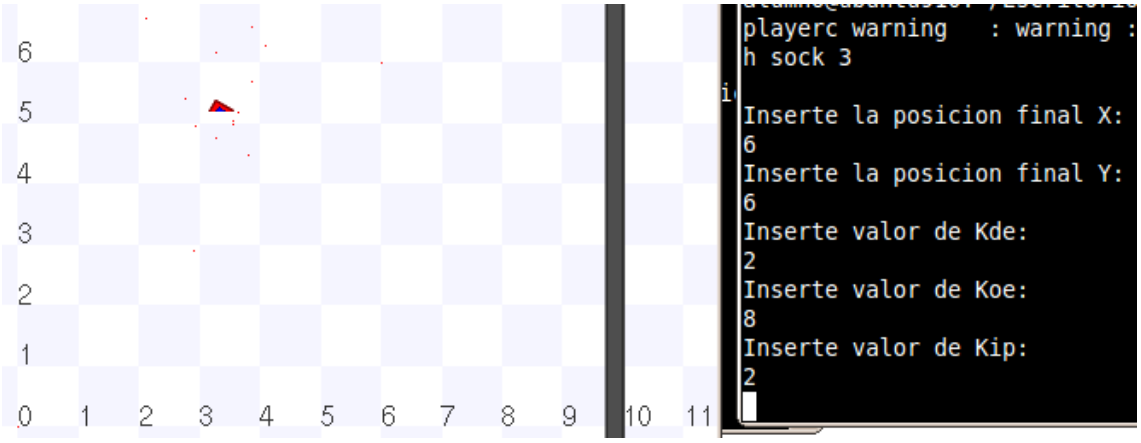
Koe: movimiento similar al controlador P



Kip:



Kde, Koe y Kip: se vuelve inestable.

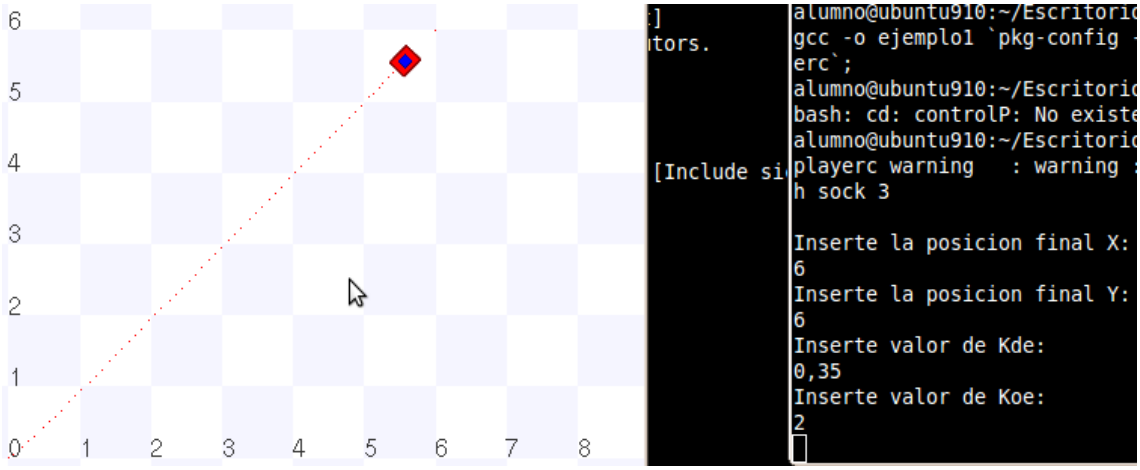


•Compare el comportamiento de este sistema con el obtenido mediante la simulación con la herramienta Matlab/Simulink. Discuta el efecto de la discretizacion de los controladores.

Al utilizar los mismos valores tanto en las pruebas anteriores como en la práctica de Matlab, vemos que el comportamiento del robot es prácticamente el mismo.

•Compare el funcionamiento de este controlador con el utilizado con player mediante el comando set_cmd_pose suponiendo que el ángulo objetivo en set_cmd_pose se fija mediante la pendiente de la recta que une el objetivo inicial del robot con la posición final.

Adjuntamos la captura de las pruebas realizadas con set_cmd_pose



En el caso de st_cmd_pose primero corrige el ángulo y después avanza hasta punto de destino por lo que vemos que su trayectoria es totalmente recta. Sin embargo, en el caso del controlador P corrige el ángulo mientras avanza por lo que se un ligera curva en el inicio del movimiento.

2. Diseño de un control para seguimiento de paredes

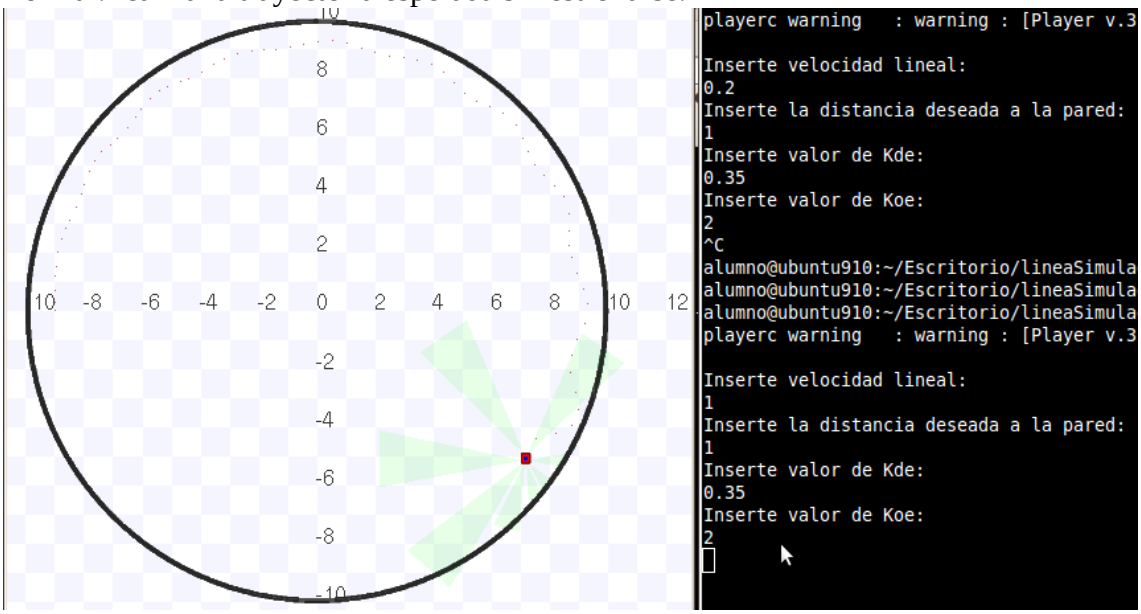
Código completado:

```
// Create and subscribe to a sonar device
sonar = playerc_sonar_create(client, 0);
if (playerc_sonar_subscribe(sonar, PLAYER_OPEN_MODE) != 0)
        return 1;

// Pedir por teclado la trayectoria velocidad lineal (V) y la distancia deseada a pa
printf("Inserte velocidad lineal:\n");
scanf("%lf",&vl);
printf("Inserte la distancia deseada a la pared:\n");
scanf("%lf",&dp);

// Pedir por teclado las ganancias del controlador Kde y Koe
printf("Inserte valor de Kde:\n");
scanf("%lf",&Kde);
printf("Inserte valor de Koe:\n");
scanf("%lf",&Koe);

// Leer la primera medida del sonar 0 y de la posición del robot (para almacenarlas
// para la primera iteración del bucle de control)
playerc_sonar_get_geom(sonar);
playerc_client_read(client);
posicionXAnterior=position2d->px;
posicionYAnterior=position2d->py;
anguloOAnterior=position2d->pa;
medidaSAnterior=sonar->scan[0];

//cambiamos color
playerc_graphics2d_setcolor(graficos,color);

//BUCLE DE CONTROL:
while (1){
```

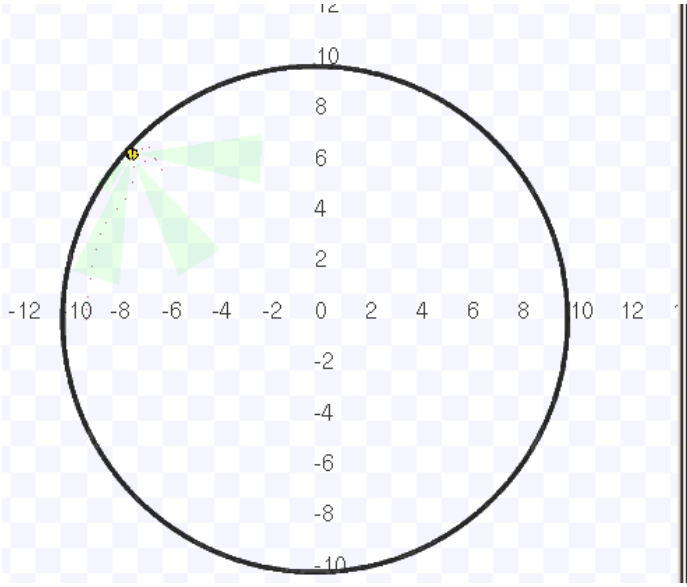
```
        playerc_client_read(client);
        //Leemos la posición actual del robot y la medida actual del sensor 0
        posicionX=position2d->px;
        posicionY=position2d->py;
        anguloO=position2d->pa;
        medidaS=sonar->scan[0];
        //Calculamos los errores lateral y de orientación
        distanciaAvanzada=sqrt(((posicionX-posicionXAnterior)*(posicionX-posicionXAnterior))+((posicionY-posicionYAnterior)*(posicionY-
posicionYAnterior)));
        oe=atan2((medidaS-medidaSAnterior),distanciaAvanzada);
        de=(medidaS+OFFSETY)*cos(oe)-dp;
        //Calculamos la señal de control (velocidad angular)
        va=Kde*de+Koe*oe;
        //Enviamos las velocidades al robot (angular y lineal)
        playerc_position2d_set_cmd_vel(position2d,vl,0.0,va,1);
        //Dibujamos un punto verde en la posición actual del robot
        playerc_client_read(client);
        puntos->px=position2d->px;
        puntos->py=position2d->py;
        playerc_graphics2d_draw_points (graficos, puntos, 1);
        //actualizamos las variables para la proxima iteracion

        posicionXAnterior=posicionX;
        posicionYAnterior=posicionY;
        anguloOAnterior=anguloO;
        medidaSAnterior=medidaS;
        }
        // Shutdown
```

•Observe de que manera influye el valor de las ganancias del controlador en el siguiente de la trayectoria (valores grandes y valores pequeños). Documéntelo con varios ejemplos y justifique la respuesta.

Normal: realiza la trayectoria esperada sin estrellarse.

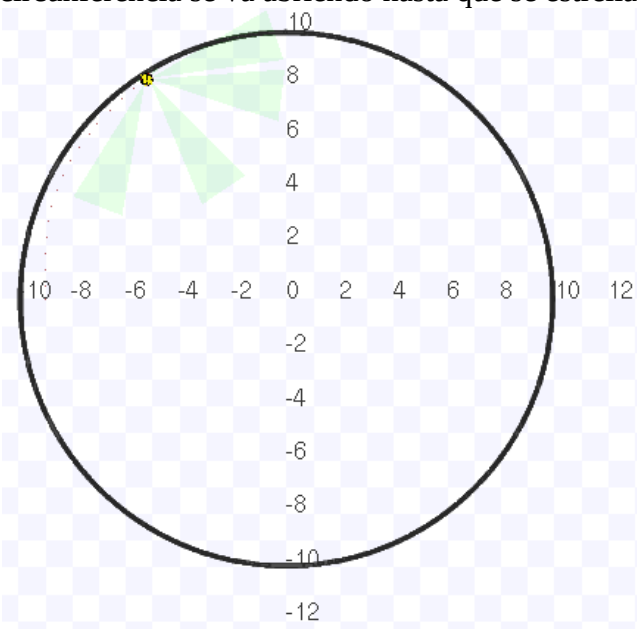


Valores altos: la corrección de la trayectoria es muy alta por lo que oscila muy rápido hasta estrellarse.



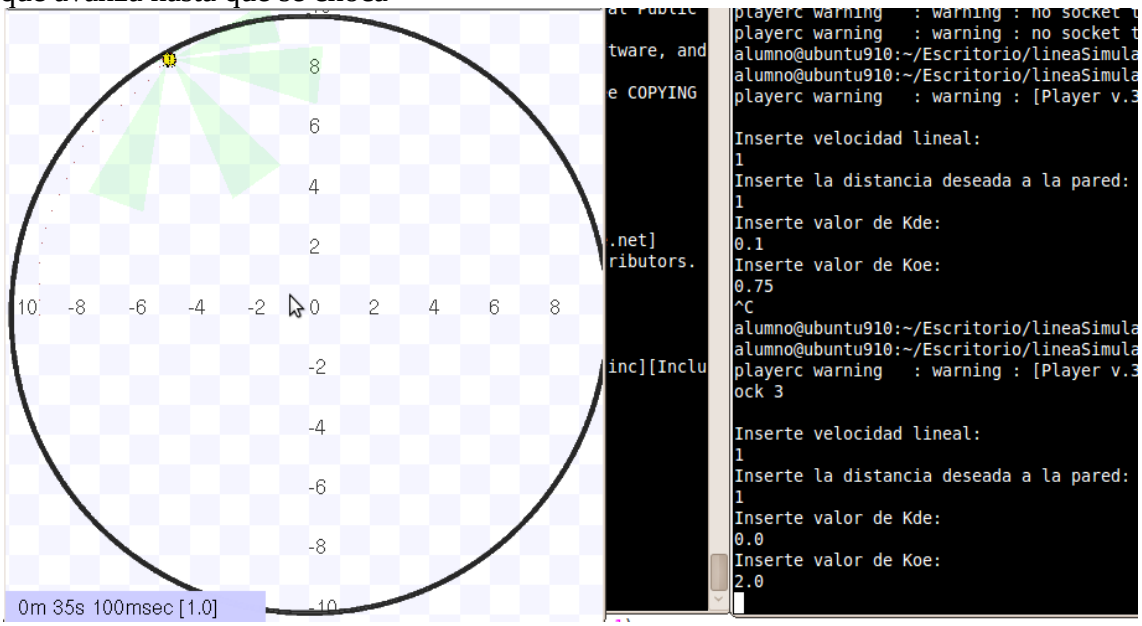

```
Inserte velocidad lineal:
1
Inserte la distancia deseada a la pared:
1
Inserte valor de Kde:
1
Inserte valor de Koe:
4
```

Valores pequeños: la corrección de la trayectoria es muy pequeña por lo que la circunferencia se va abriendo hasta que se estrella.

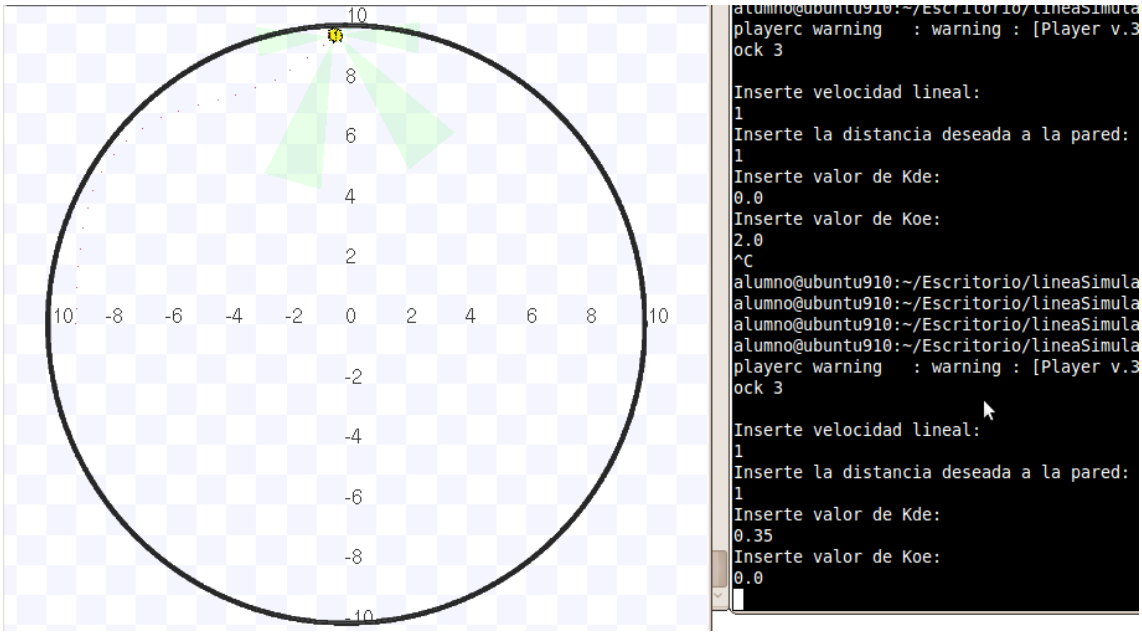



```
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
playerc warning   : warning : no socket t
alumno@ubuntu910:~/Escritorio/lineaSimula
alumno@ubuntu910:~/Escritorio/lineaSimula
playerc warning   : warning : [Player v.3

Inserte velocidad lineal:
1
Inserte la distancia deseada a la pared:
1
Inserte valor de Kde:
0.1
Inserte valor de Koe:
0.75
```

•**¿Qué sucede si anulamos Kde?** Al no corregir la distancia se va cerrando a medida que avanza hasta que se choca



•**¿Qué sucede si anulamos Koe?** Al no corregir el ángulo empieza a oscilar y se estrella.



•¿De qué manera influye el valor de la velocidad lineal V del robot en el seguimiento de la trayectoria?

Para un valor dado de velocidad lineal se deben calcular los valores de Kde y Koe para que el comportamiento sea el deseado.

Ampliaciones (opcional):

•Modifique la aplicación para que el robot siga la pared más cercana (izquierda o derecha).

Lo primero que hacemos es leer de los dos sonar laterales para ver que pared está más cerca. Una vez obtenida la medida nos colocamos a la distancia introducida por teclado de la pared más cercana y avanzamos. En cada interacción efectuamos estas operaciones por lo que si en algún momento está más cercana la otra pared corregimos la trayectoria hasta ponernos a la distancia deseada de la otra pared y continuamos avanzando.
Dependiendo de loa parámetros introducidos el robot oscila más o menos al cambiar de una pared a otra pero siempre intenta mantener la distancia a la pared sin estrellarse.

•Modifique la aplicación para que, en caso de detectarse dos paredes, el robot se desplace centrado respecto a ambas (seguimiento de pasillos).

En este caso no introducimos por teclado la distancia a una pared sino que la calculamos en todo momento para que vaya centrado por el pasillo. Por lo tanto en vez de introducir una distancia constante, calculamos la distancia sumando las dos medidas de los sonar y dividendo ésta entre dos para ver donde esta e centro del pasillo. Una vez obtenida esta distancia a ambas pareces el robot avanza o corrige su posición para mantenerse en el medio.

•Amplíe la aplicación anterior para que, en caso de detectarse una pared frontal, el robot se pare a una determinada distancia de la misma.

En este caso no pedimos la velocidad por teclado, la calculamos en función de la distancia a la que se encuentre el robot de la pared. Incluimos las medidas de los dos sonar delanteros y calculamos la distancia frontal del robot a la pared. Según se va acercando a la pared vamos reduciendo la velocidad hasta detenerse. Para ver claramente que se detiene al estar a cierta distancia de la pared igualamos a cero la velocidad angular al tener una velocidad lineal de cero porque sino el robot se da la vuelta y sigue avanzando por todos los sitios.