PRÁCTICA 2. SENSORES EN PLAYER/STAGE

El objetivo de esta práctica es utilizar diferentes sensores que están disponibles en el servidor Player y poder comprobar su funcionamiento sobre el simulador Stage. Se emplearán diversos interfaces como el *position2d*, *sonar*, *graphics2d* y *laser*, con el fin de estudiar sus ventajas e inconvenientes en una simulación realista.

1.1 Introducción

Empleando el mismo entorno que se describió en la primera práctica (ver Figura 1).

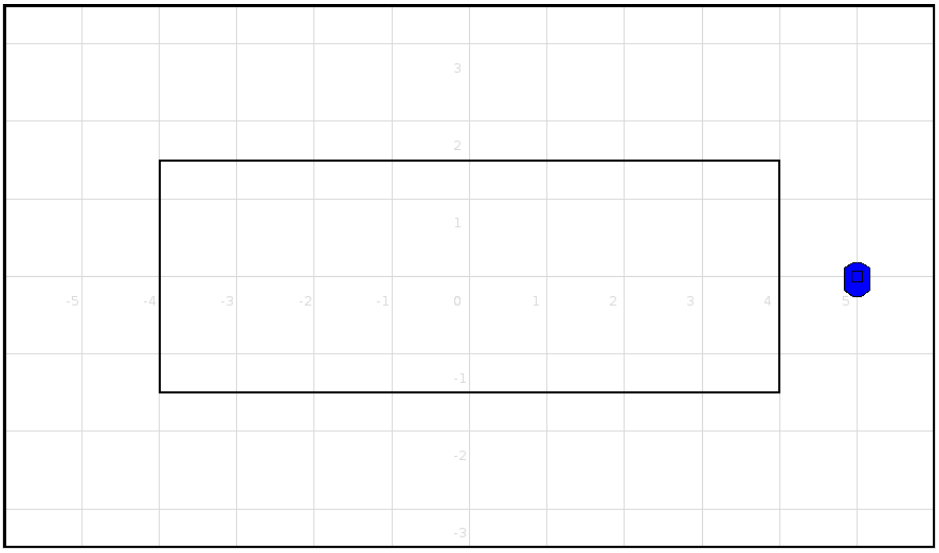

Figura 1. Entorno de simulación.

Y siguiendo la misma estructura de archivos (programa cliente y archivos de configuración)

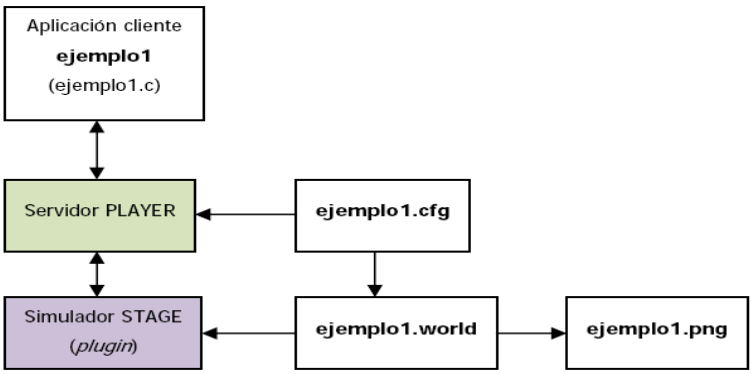

Figura 2. Estructura de archivos

Se partirá de una aplicación capaz de mover el robot alrededor del entorno. Para ello se proporciona una aplicación cliente básica (esqueleto.c) donde además se hace uso, a modo de ejemplo, del interfaz *graphics2d* para representar la posición del robot a lo largo del recorrido.

```
#include <stdio.h>
#include <math.h>

#include <libplayerc/playerc.h>

int
main(int argc, const char **argv)
{
  int i;
  playerc_client_t *client;
  playerc_position2d_t *position2d;
  player_pose2d_t position2d_target;
  double arrayx[5]={ 5.0 ,  -5.0 , -5.0 ,  5.0 ,  5.0};
  double arrayy[5]={ 2.5 ,   2.5 , -2.5 , -2.5 ,  0.0};
  double arraya[5]={3.14 , -1.57 ,  0.0 , 1.57 , 1.57};

  //sonar

  //laser

  //drawing
  playerc_graphics2d_t *graficos;
  player_point_2d_t *puntos;
  player_color_t color;
  puntos=(player_point_2d_t *)malloc(sizeof(player_point_2d_t)*(1)); //(1) punto
  color.red=255; color.green=0; color.blue=0;


  // Create a client and connect it to the server.
  client = playerc_client_create(NULL, "localhost", 6665);
  if (playerc_client_connect(client) != 0)
    {
      fprintf(stderr, "error: %s\n", playerc_error_str());
      return -1;
    }


  // Create and subscribe to a position2d device.
  position2d = playerc_position2d_create(client, 0);
  if (playerc_position2d_subscribe(position2d, PLAYER_OPEN_MODE) != 0)
    {
      fprintf(stderr, "error: %s\n", playerc_error_str());
      return -1;
    }

  // Fixing initial position
  playerc_position2d_set_odom(position2d,5.0,0.0,1.57);

  // Create and subscribe to a sonar device

  // Create and subscribe to a graphics device
  graficos = playerc_graphics2d_create(client, 0);
  if (playerc_graphics2d_subscribe(graficos, PLAYER_OPEN_MODE) != 0)
    {
      fprintf(stderr, "error: %s\n", playerc_error_str());
      return -1;
```

```
    }

  // Fix colour
  playerc_graphics2d_setcolor (graficos, color);

  // Clear screen
  playerc_graphics2d_clear(graficos);

  // Enable motors
  playerc_position2d_enable(position2d,1);

  for ( i=0 ; i<5 ; i++ )
    {
      position2d_target.px = arrayx[i];
      position2d_target.py = arrayy[i];
      position2d_target.pa = arraya[i];

      // Move to pose
      playerc_position2d_set_cmd_pose(position2d,        position2d_target.px        ,        position2d_target.py,
position2d_target.pa , 1);

      // Stop when reach the target
      while          (sqrt(pow(position2d->px     -     position2d_target.px,2.0)     +     pow(position2d->py     -
position2d_target.py,2.0)) > 0.05 )
         {
           // Wait for new data from server
           playerc_client_read(client);

           // Print current robot pose
           printf("position2d  :  x  %f  y  %f  th  %f  stall  %d\n",position2d->px,  position2d->py,  position2d->pa,
position2d->stall);
           // What does mean stall?
           // x, y, th, world frame or robot frame?

           // Draw current robot pose
           puntos[0].px=position2d->px;
           puntos[0].py=position2d->py;
           playerc_graphics2d_draw_points (graficos, puntos, 1);

           // Print sonar readings

         }
    }


  // Unsuscribe and Destroy
  // position2d
  playerc_position2d_unsubscribe(position2d); playerc_position2d_destroy(position2d);
  // sonar

  // graphics2d
  playerc_graphics2d_unsubscribe(graficos); playerc_graphics2d_destroy(graficos);
  // client
  playerc_client_disconnect(client); playerc_client_destroy(client);

  // End
  return 0;
}
```

Figura 3. Programa fuente “esqueleto.c”

Aunque se proporciona el ejemplo de uso de la interfaz graphics2d se recomienda consultar la ayuda sobre el uso de la misma en la documentación oficial de Player/Stage.

Classes

struct	**playerc_graphics2d_t** *Graphics2d device data. More...*

Functions

playerc_graphics2d_t *	**playerc_graphics2d_create** (**playerc_client_t** *client, int index) *Create a graphics2d device proxy.*
void	**playerc_graphics2d_destroy** (**playerc_graphics2d_t** *device) *Destroy a graphics2d device proxy.*
int	**playerc_graphics2d_subscribe** (**playerc_graphics2d_t** *device, int access) *Subscribe to the graphics2d device.*
int	**playerc_graphics2d_unsubscribe** (**playerc_graphics2d_t** *device) *Un-subscribe from the graphics2d device.*
int	**playerc_graphics2d_setcolor** (**playerc_graphics2d_t** *device, **player_color_t** col) *Set the current drawing color.*
int	**playerc_graphics2d_draw_points** (**playerc_graphics2d_t** *device, **player_point_2d_t** pts[], int count) *Draw some points.*
int	**playerc_graphics2d_draw_polyline** (**playerc_graphics2d_t** *device, **player_point_2d_t** pts[], int count) *Draw a polyline that connects an array of points.*
int	**playerc_graphics2d_draw_polygon** (**playerc_graphics2d_t** *device, **player_point_2d_t** pts[], int count, int filled, **player_color_t** fill_color) *Draw a polygon.*
int	**playerc_graphics2d_clear** (**playerc_graphics2d_t** *device) *Clear the canvas.*

Figura 4. Proxy del interfaz *graphics2d*.

1.2 Pruebas

Odometría

En este apartado se pretende comprobar la funcionalidad de la odometría como sensor básico de navegación. Para ello, haciendo uso del comando *set_cmd_pose* en el interfaz position2d y analizando los valores de la posición que nos proporciona dicho interfaz, se pide:

- Analiza el efecto de tener un error de odometría nulo durante diferentes recorridos.
- Introduce diferentes errores de odometría (tanto en X e Y como en Th) mediante el parámetro *odom_error [errorX errorY errorTh]* dentro del archivo "*.world". Asegúrate de modificar el modo de localización del pioneer (pioneer.inc), cambiando el modo de "gps" a "odom". Analiza el efecto de dicho error para diferentes valores y representa el error que se obtiene en el simulador stage.
- ¿Qué efecto tiene eliminar el comando *set_odom* en el funcionamiento del robot?
- ¿Sería útil la odometría si no se le proporciona la posición inicial de partida al robot?
- ¿Para qué sería útil el parámetro *stall* que nos proporciona el interfaz *position2d*? ¿Qué valor tomaría este parámetro en aquellas situaciones en las que el robot se traslade por suelos resbaladizos y tropiece con algún obstáculo?

Sensores de ultrasonidos

Se desea emplear los sensores de ultrasonidos con la finalidad de obtener información de distancia sobre el entorno en el que se mueve el robot. Para ello será necesario emplear el interfaz *sonar* (ver Figura 4). Consultar la documentación de la página oficial para utilizar este interfaz.

Classes

struct	**playerc_sonar_t** *Sonar proxy data. More...*

Functions

playerc_sonar_t *	**playerc_sonar_create** (**playerc_client_t** *client, int index) *Create a sonar proxy.*
void	**playerc_sonar_destroy** (**playerc_sonar_t** *device) *Destroy a sonar proxy.*
int	**playerc_sonar_subscribe** (**playerc_sonar_t** *device, int access) *Subscribe to the sonar device.*
int	**playerc_sonar_unsubscribe** (**playerc_sonar_t** *device) *Un-subscribe from the sonar device.*
int	**playerc_sonar_get_geom** (**playerc_sonar_t** *device) *Get the sonar geometry.*

Figura 5. Proxy del interfaz *sonar*

Una vez creado y suscrito el dispositivo, se pide:

- Comprobar la información que proporcionan los ultrasonidos sobre el simulador Stage.
- Obtener la geometría de los sensores de ultrasonidos sobre el pioneer 2DX
- Obtener la lectura de cada uno de los 16 sensores por pantalla a medida que se va moviendo el robot por el entorno
- ¿Esta lectura está referida al sistema de coordenadas del entorno, del robot o está referida a la posición que ocupa cada uno de los sensores del robot?
- Dibujar el punto de impacto de los ultrasonidos sobre el simulador Stage.
- Trasladar los puntos de impacto a coordenadas x-y del mundo (entorno) en el que se mueve el robot. Para comprobar la exactitud de la medida que proporcionan los sensores de ultrasonidos.

Área de seguridad

En este apartado se pretende establecer un área de seguridad alrededor del robot.

Se debe tener en cuenta que la mayoría de los robots móviles trabajan en entornos dinámicos y además suelen encontrarse personas en el mismo. Por ello se debe evitar colisionar con obstáculos y y personas del entorno.

En primer lugar se deberá modificar el entorno para añadir un objeto estático que se encuentre dentro de la trayectoria del robot. A continuación se muestra un ejemplo de objeto estático en el entorno de desarrollo de la práctica.

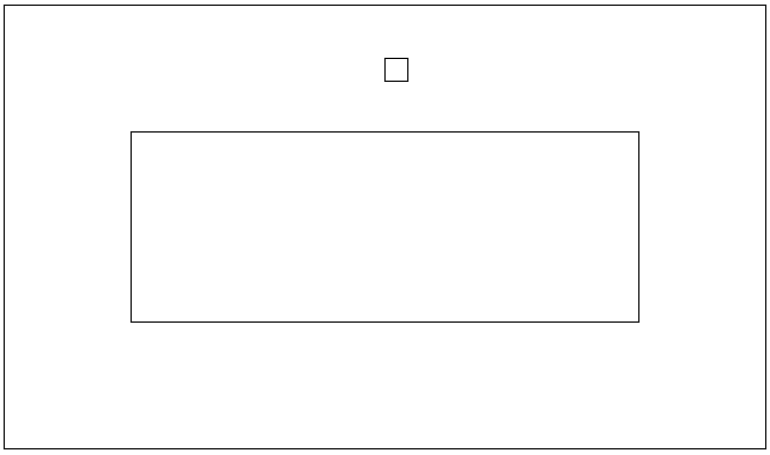

Figura 6. Ejemplo de objeto estático en la trayectoria del robot.

Se pide:

- Modificar el programa de los apartados anteriores para que, empleando la lectura de ultrasonidos, se pueda definir un área de seguridad de diferentes valores alrededor del robot (10cm, 20cm, 30cm, ...). Cuando el robot detecte un objeto dentro de su área de seguridad realizará la maniobra conocida como *Stop&Go*, permaneciendo detenido hasta que el objeto desaparezca del área de seguridad.
- ¿Cómo afecta esta maniobra sobre el parámetro *stall* del interfaz *position2d*?
- ¿Qué ocurre si se define un área de seguridad muy grande? Por ejemplo, del orden de 1.5m.

1.3 Ampliación.

Se propone como ampliación de la práctica utilizar un nuevo sensor como es el sensor Láser.

Los Láser se están convirtiendo en uno de los principales sensores utilizados en bases robóticas ya que proporcionan una medida muy exacta de distancia, con una cobertura grande (del orden de 180º) y, además, sus precios se están reduciendo considerablemente debido al amplio uso que se hace de ellos.

En este apartado se pide: instanciar y suscribir un sensor Láser, para ello hacer uso del proxy *laser* (ver figura 7) y consultar la ayuda de la página oficial de Player/Stage.

Classes

struct	**playerc_laser_t** *Laser proxy data. More...*

Functions

playerc_laser_t *	**playerc_laser_create** (**playerc_client_t** *client, int index) *Create a laser proxy.*
void	**playerc_laser_destroy** (**playerc_laser_t** *device) *Destroy a laser proxy.*
int	**playerc_laser_subscribe** (**playerc_laser_t** *device, int access) *Subscribe to the laser device.*
int	**playerc_laser_unsubscribe** (**playerc_laser_t** *device) *Un-subscribe from the laser device.*
int	**playerc_laser_set_config** (**playerc_laser_t** *device, double min_angle, double max_angle, double resolution, double range_res, unsigned char intensity) *Configure the laser.*
int	**playerc_laser_get_config** (**playerc_laser_t** *device, double *min_angle, double *max_angle, double *resolution, double *range_res, unsigned char *intensity) *Get the laser configuration.*
int	**playerc_laser_get_geom** (**playerc_laser_t** *device) *Get the laser geometry.*
void	**playerc_laser_printout** (**playerc_laser_t** *device, const char *prefix) *Print a human-readable summary of the laser state on stdout.*

Figura 7. Proxy del interfaz *laser*.

Comprobar la información que proporciona el sensor Láser:

- Medida de distancias
- Geometría
- Valores máximos y mínimos

Definir un área de seguridad utilizando la medida del Láser. ¿Resulta más eficiente utilizar las medidas del sensor Láser o las medidas del sensor de ultrasonidos para esta tarea?