PRÁCTICA 3. NAVEGACIÓN CLÁSICA EN PLAYER/STAGE

El objetivo de esta práctica es utilizar algunos de los conceptos principales de navegación clásica estudiados durante el tema 3.

1.1 Introducción

Como punto de partida se empleará el entorno con obstáculo que se utilizó en la práctica 2 (ver Figura 1).

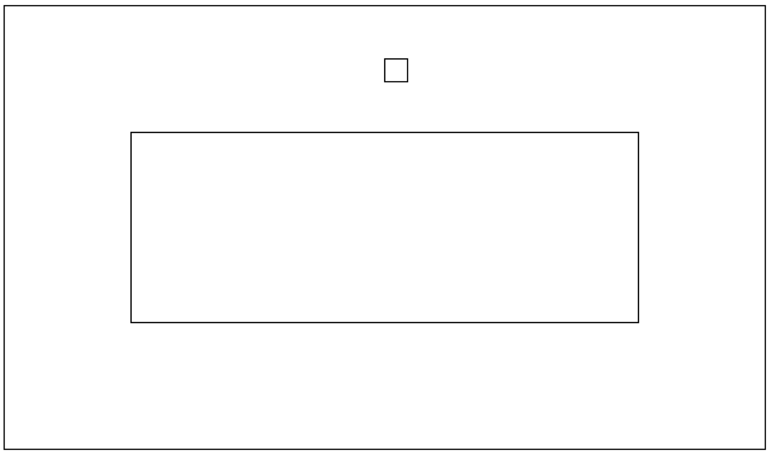


Figura 1. Entorno de simulación.

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

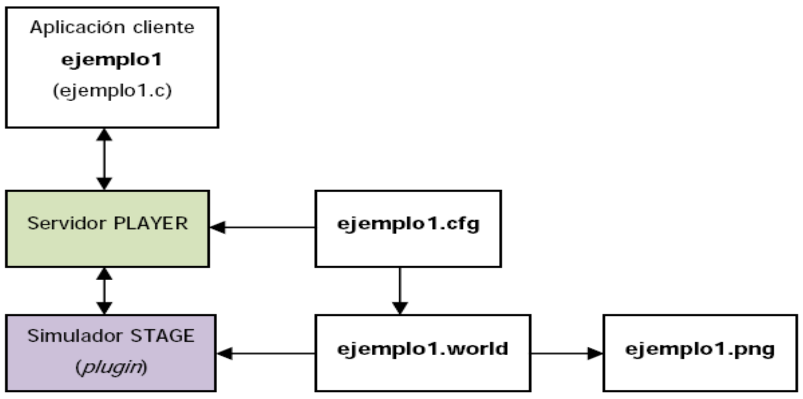


Figura 2. Estructura de archivos

Se partirá de una aplicación capaz de mover el robot alrededor del entorno. Para ello se proporciona la misma aplicación cliente básica (esqueleto.c) de la práctica 2.

```
#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"

1.2 Pruebas

Mapa

En este apartado se pretende comprobar la funcionalidad del mapeado del entorno. Para ello, haciendo uso del proxy *map*, se pide:

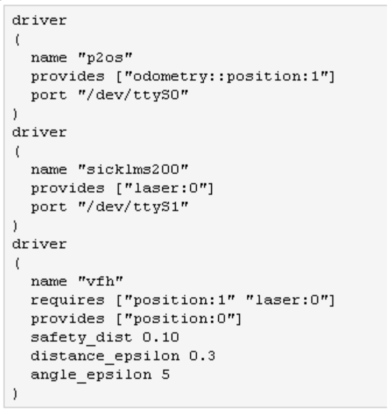
- Obtener la información del proxy *map*.
- ¿Qué tipo de representación del entorno utiliza player/stage a través del proxy *map*?

Evitación de obstáculos

Se pide diseñar una maniobra de evitación del obstáculo para conseguir realizar un recorrido completo sin chocarse con el mismo.

Algoritmo de evitación de obstáculos VFH+ (Vector Field Histogram)

En esta apartado de la práctica se empleará el algoritmo de evitación de obstáculos VFH+ propuesto por Borenstein. Para ello será necesario utilizar el driver *vfh* que proporciona el player (ver Figura 4), adaptando los parámetros del mismo para conseguir navegar por el entorno de forma segura.



```
driver
(
  name "p2os"
  provides ["odometry::position:1"]
  port "/dev/ttyS0"
)
driver
(
  name "sicklms200"
  provides ["laser:0"]
  port "/dev/ttyS1"
)
driver
(
  name "vfh"
  requires ["position:1" "laser:0"]
  provides ["position:0"]
  safety_dist 0.10
  distance_epsilon 0.3
  angle_epsilon 5
)
```

Figura 4. Ejemplo de uso del driver *vfh*.

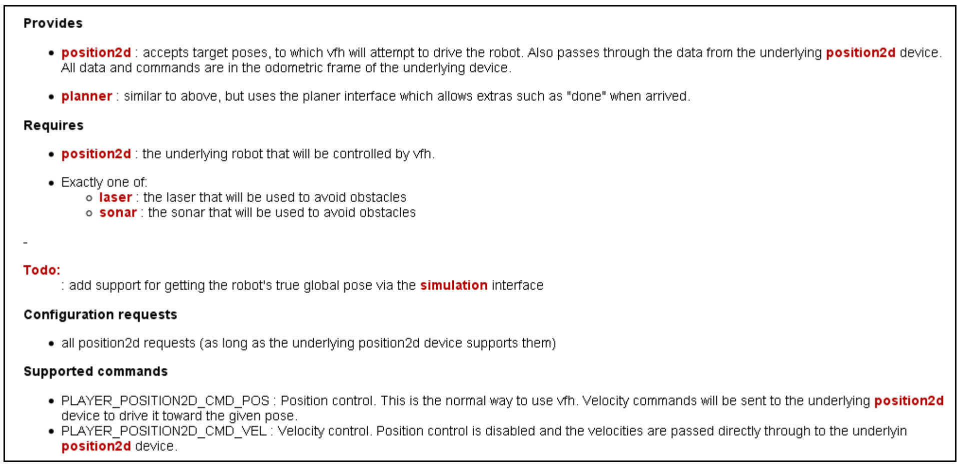
A continuación se muestran los parámetros que se pueden modificar en el driver *vfh*:

- `cell_size (length)`
  - o `Default: 0.1 m`
  - o `Local occupancy map grid size`
- `window_diameter (integer)`
  - o `Default: 61`
  - o `Dimensions of occupancy map (map consists of window_diameter X window_diameter cells).`
- `sector_angle (integer)`
  - o `Default: 5`

  - Histogram angular resolution, in degrees.
- safety_dist_0ms (length)
  - Default: 0.1 m
  - The minimum distance the robot is allowed to get to obstacles when stopped.
- safety_dist_1ms (length)
  - Default: safety_dist_0ms
  - The minimum distance the robot is allowed to get to obstacles when travelling at 1 m/s.
- max_speed (length / sec)
  - Default: 0.2 m/sec
  - The maximum allowable speed of the robot.
- max_speed_narrow_opening (length / sec)
  - Default: max_speed
  - The maximum allowable speed of the robot through a narrow opening
- max_speed_wide_opening (length / sec)
  - Default: max_speed
  - The maximum allowable speed of the robot through a wide opening
- max_acceleration (length / sec / sec)
  - Default: 0.2 m/sec/sec
  - The maximum allowable acceleration of the robot.
- min_turnrate (angle / sec)
  - Default: 10 deg/sec
  - The minimum allowable turnrate of the robot.
- max_turnrate_0ms (angle / sec)
  - Default: 40 deg/sec
  - The maximum allowable turnrate of the robot when stopped.
- max_turnrate_1ms (angle / sec)
  - Default: max_turnrate_0ms
  - The maximum allowable turnrate of the robot when travelling 1 m/s.
- min_turn_radius_safety_factor (float)
  - Default: 1.0
  - ?
- free_space_cutoff_0ms (float)
  - Default: 2000000.0
  - Unitless value. The higher the value, the closer the robot will get to obstacles before avoiding (while stopped).
- free_space_cutoff_1ms (float)
  - Default: free_space_cutoff_0ms
  - Unitless value. The higher the value, the closer the robot will get to obstacles before avoiding (while travelling at 1 m/s).
- obs_cutoff_0ms (float)
  - Default: free_space_cutoff_0ms
  - ???
- obs_cutoff_1ms (float)
  - Default: free_space_cutoff_1ms
  - ???
- weight_desired_dir (float)
  - Default: 5.0
  - Bias for the robot to turn to move toward goal position.
- weight_current_dir (float)
  - Default: 3.0
  - Bias for the robot to continue moving in current direction of travel.
- distance_epsilon (length)
  - Default: 0.5 m

    - Planar distance from the target position that will be considered acceptable.
- angle_epsilon (angle)
    - Default: 10 deg
    - Angular difference from target angle that will considered acceptable.
- Stall escape options. If the underlying position2d device reports a stall, this driver can attempt a blind escape procedure. It does so by driving forward or backward while turning for a fixed amount of time. If the escape fails (i.e., the stall is still in effect), then it will try again.
    - escape_speed (length / sec)
        - Default: 0.0
        - If non-zero, the translational velocity that will be used while trying to escape.
    - escape_time (float)
        - Default: 0.0
        - If non-zero, the time (in seconds) for which an escape attempt will be made.
    - escape_max_turnrate (angle / sec)
        - Default: 0.0
        - If non-zero, the maximum angular velocity that will be used when trying to escape.
    - synchronous (int)
        - default: 0
        - If zero (the default), VFH runs in its own thread. If non-zero, VFH runs in the main Player thread, which will make the server less responsive, but prevent nasty asynchronous behaviour under high CPU load. This is probably only useful when running demanding simulations.

Nota: prestar especial atención a los dispositivos que requiere y a los que proporciona.



Provides

- **position2d** : accepts target poses, to which vfh will attempt to drive the robot. Also passes through the data from the underlying **position2d** device. All data and commands are in the odometric frame of the underlying device.
- **planner** : similar to above, but uses the planer interface which allows extras such as "done" when arrived.

Requires

- **position2d** : the underlying robot that will be controlled by vfh.
- Exactly one of:
    - **laser** : the laser that will be used to avoid obstacles
    - **sonar** : the sonar that will be used to avoid obstacles

-

Todo:
: add support for getting the robot's true global pose via the **simulation** interface

Configuration requests

- all position2d requests (as long as the underlying position2d device supports them)

Supported commands

- PLAYER_POSITION2D_CMD_POS : Position control. This is the normal way to use vfh. Velocity commands will be sent to the underlying **position2d** device to drive it toward the given pose.
- PLAYER_POSITION2D_CMD_VEL : Velocity control. Position control is disabled and the velocities are passed directly through to the underlyin **position2d** device.

1.3 Ampliación

Análisis del driver de planificación global *wavefront*. Player proporciona un driver de planificación global basado en el popular algoritmo NF1 o WaveFront, para utilizar este driver es necesario utilizar una tarea de evitación de obstáculos subyacente que normalmente suele llevarse a cabo mediante el algoritmo VFH.

En este apartado de ampliación se pide:

- Analizar los parámetros que pueden configurarse en el driver para utilizar este planificador.
- ¿Qué dispositivos son necesarios para utilizarlo? ¿Qué dispositivos proporciona?
- ¿Qué algoritmo de evitación de obstáculos utiliza por defecto?
- ¿Cómo se debe instanciar en los archivos de configuración?