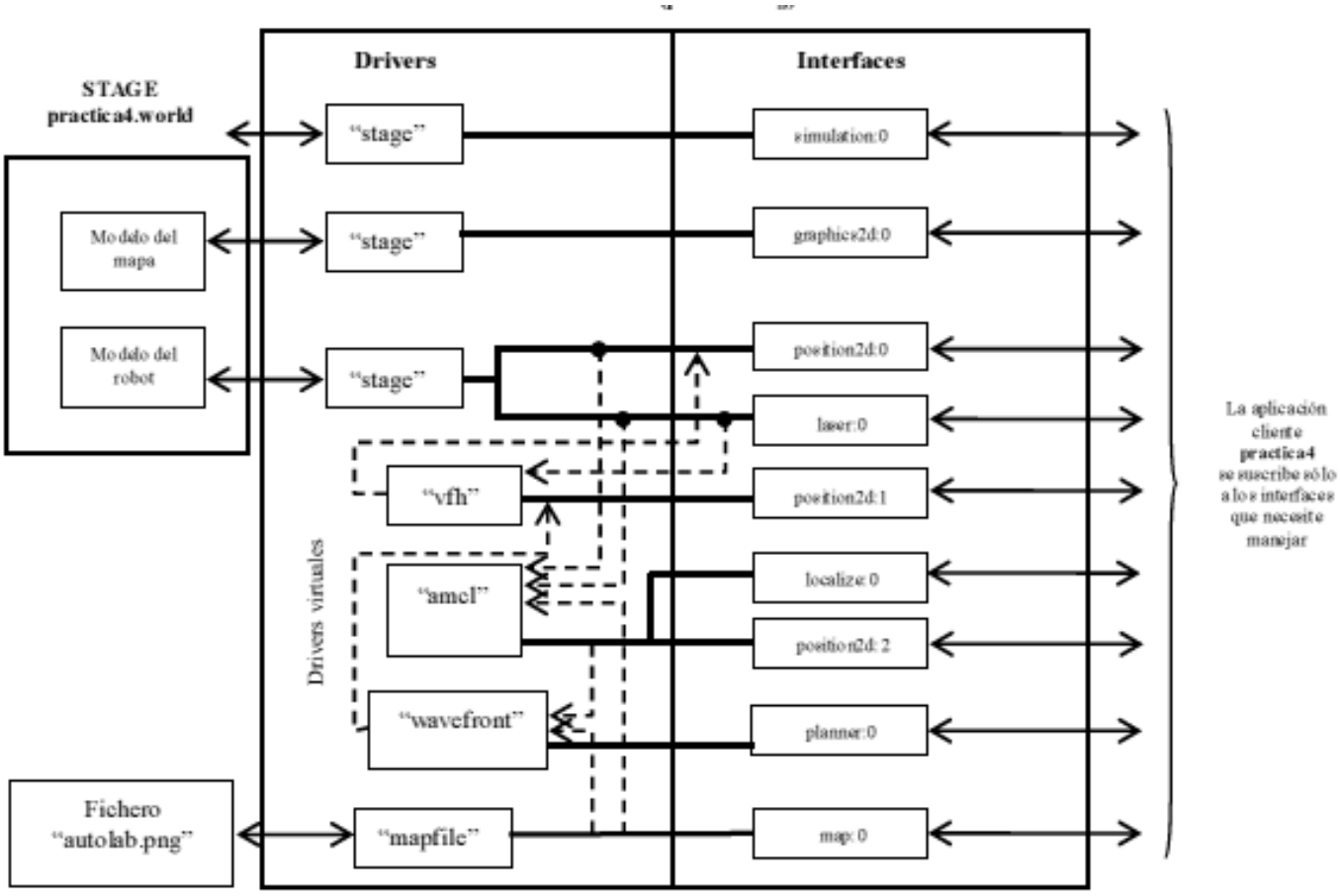
Sistemas de Control para Robots
PRÁCTICA 4

Planificación Global

Pedro Barquín Ayuso
Miguel Ballesteros García

4.1-Planificacion global con “wavefront”

Hemos implementado el código necesario siguiendo el siguiente esquema:




```
driver
(
  name "wavefront"
  provides ["planner:0"]
  requires ["output:::position2d:1" "input:::position2d:2" "map:0"]
  safety_dist 0.15
  distance_epsilon 0.5
  angle_epsilon 10
)
```

Si miramos la documentación de wavefront vemos que existen más parámetros que no estan contemplado en el driver introducido. Dentro de estos drivers cabe destacar dos:

-replan_dist_thresh (length): por defecto a distancia para volver a planificar la ruta es cada 2 metros y ayuda en entornos dinámicos. Se puede poner el valor a -1 no replanificar. Dado que no hemos podido probarlo suponemos que al evitar ciertos obstáculo que hagan desviarse mucho al robots de la ruta original, es decir, un desvío mayor de la distancia introducida en este parámetro, el robot planificara una nueva ruta hasta el destino.

-replan_min_time (float): por defecto el tiempo para planificar una nueva ruta seria de 2 segundos. Como en el caso anterior, cada 2 segundo se realizara una replanificación de la ruta.

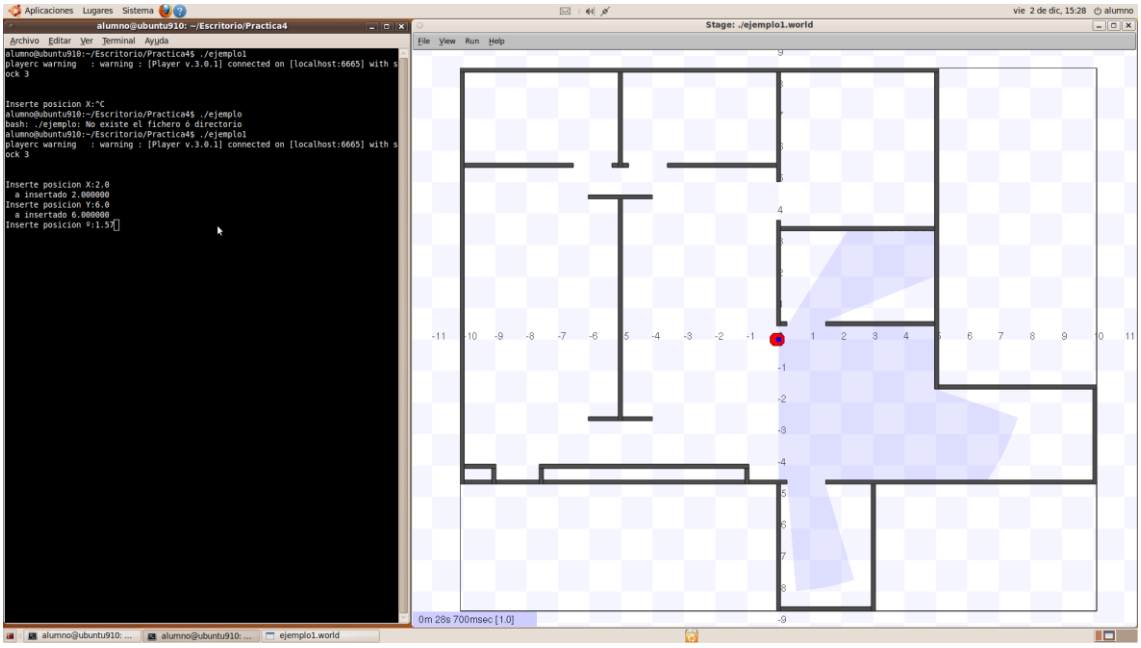
También hemos modificado el fichero .c para que solicite la posición final por pantalla, y que una vez aceptado ese destino obtenga la ruta y mande al robot a la misma, durante el recorrido se va mostrando información.

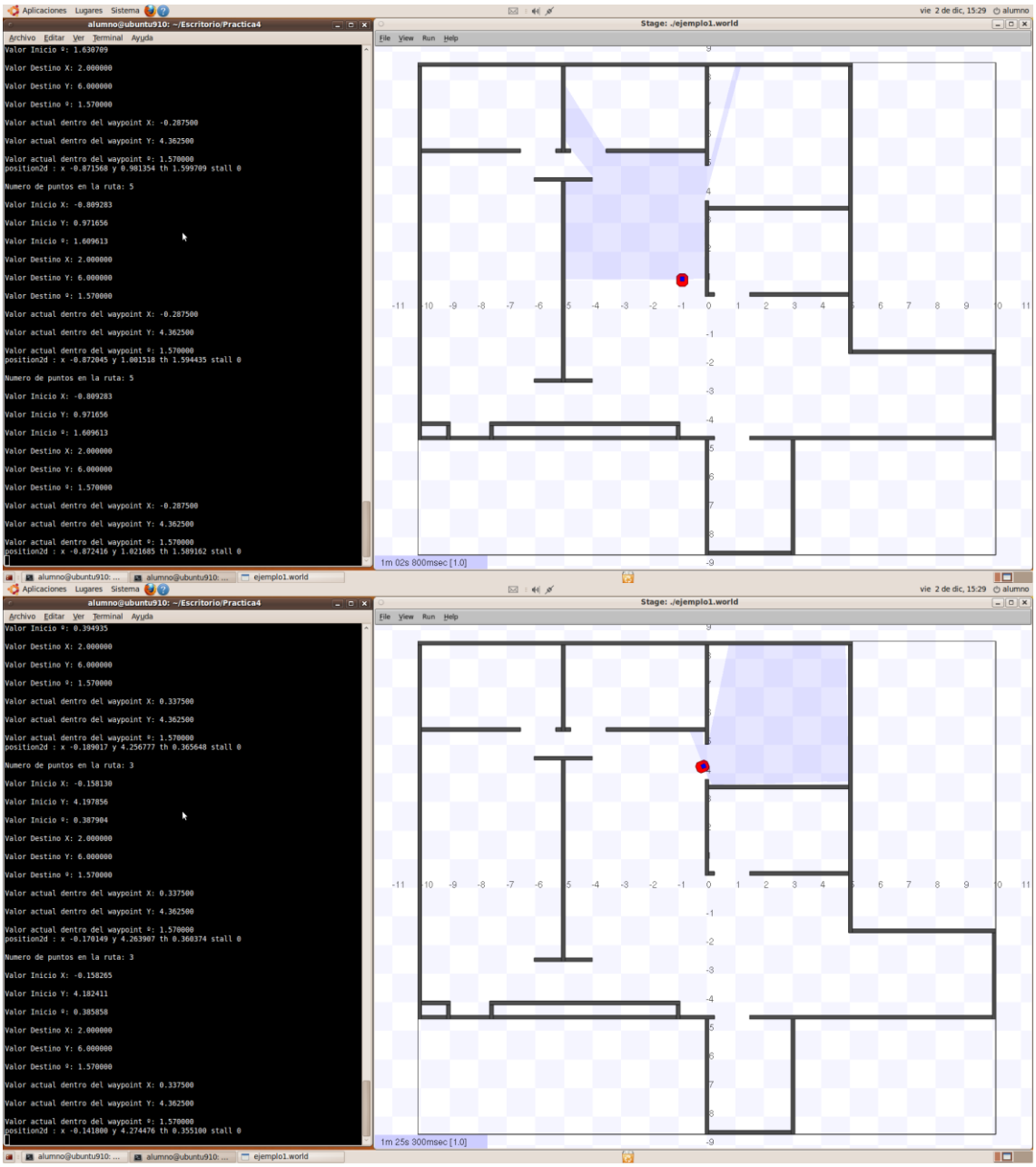
```
//planner
  playerc_planner_enable(planned,1);
printf("\nInserte posicion X:");
  scanf("%lf",&PFX);
  printf("  a insertado %lf",PFX);
  printf("\nInserte posicion Y:");
  scanf("%lf",&PFY);
  printf("  a insertado %lf",PFY);
printf("\nInserte posicion º:");
  scanf("%lf",&PFG);
  printf("  a insertado %lf",PFG);
  playerc_planner_set_cmd_pose(planned,PFX,PFY,PFG);

//hasta que no se localice no empieza a planificar creo ya que las primeras iteraciones no localiza nada
playerc_planner_get_waypoints(planned);

//Informacion planner
  printf("\nNumero de puntos en la ruta: %d\n",planned->waypoint_count);
  printf("\nValor Inicio X: %f\n",planned->px);
  printf("\nValor Inicio Y: %f\n",planned->py);
  printf("\nValor Inicio º: %f\n",planned->pa);
  printf("\nValor Destino X: %f\n",planned->gx);
  printf("\nValor Destino Y: %f\n",planned->gy);
  printf("\nValor Destino º: %f\n",planned->ga);
  printf("\nValor actual dentro del waypoint X: %f\n",planned->wx);
  printf("\nValor actual dentro del waypoint Y: %f\n",planned->wy);
printf("\nValor actual dentro del waypoint º: %f\n",planned->wa);


  while(sqrt(pow(position2d->px - PFX,2.0) + pow(position2d->py - PFY,2.0)) > 0.5){
        // 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);

        //leer particulas y dibujar
        playerc_localize_get_particles(localize);

        //PLANNER
```

```
        //playerc_planner_get_waypoints(planned);
printf("\nNumero de puntos en la ruta: %d\n",planned->waypoint_count);
            printf("\nValor Inicio X: %f\n",planned->px);
            printf("\nValor Inicio Y: %f\n",planned->py);
            printf("\nValor Inicio º: %f\n",planned->pa);
            printf("\nValor Destino X: %f\n",planned->gx);
            printf("\nValor Destino Y: %f\n",planned->gy);
            printf("\nValor Destino º: %f\n",planned->ga);
            printf("\nValor actual dentro del waypoint X: %f\n",planned->wx);
            printf("\nValor actual dentro del waypoint Y: %f\n",planned->wy);
          printf("\nValor actual dentro del waypoint º: %f\n",planned->wa);
}
```

Ejemplo de recorrido:

Para este recorrido utilizamos el wavefront con el cual mandamos una posición por consola al robot (de 0.0 , 0.0 a 2.0 , 6.0) y automáticamente el sistema traza el recorrido a seguir y los puntos intermedios, gracias al resto de componentes el robot navega evitando los obstáculos hasta el destino.





En la consola de la izquierda podemos ver la información del recorrido que se está realizando.

4.2-Manejo de la utilidad “playernav”

Para esta parte introducimos un nuevo robot lo que nos permite realizar dos recorridos en la misma ejecución. Uno de ellos se inicia como en las prácticas anteriores y el nuevo robot se lanza en otro puerto median la instrucción:

```
player ejemplo2.cfg –p 6666
```

También hay que modificar el fichero ejemplo2.cfg para indicar el puerto en el que esta cada driver.

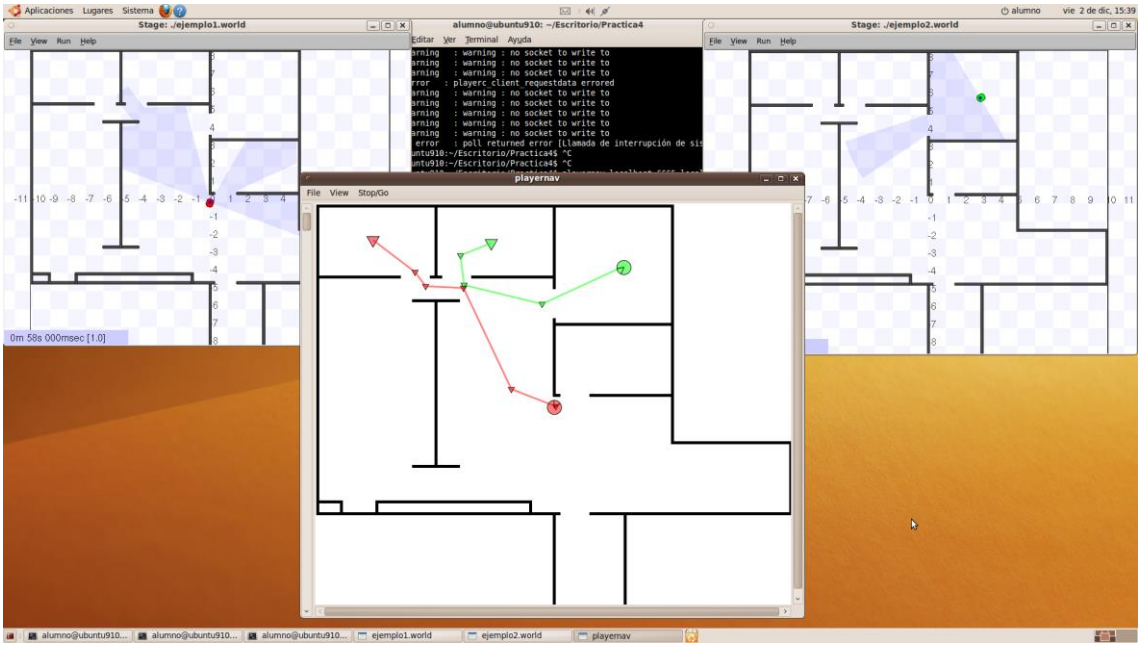
```
driver
(
  name "vfh"
  provides [ "6666:position2d:1" ]
  requires [ "position2d:0" "laser:0" ]
)

Driver
(
  name “amcl”
  provides [“6666:localize:0” “6666:position2d:2”]
  requires ["odometry:::position2d:0" "laser:::map:0" "laser:0"]
init_pose [3.0 6.0 -1.57]
init_pose_var [1 1 6.28]
pf_min_samples 100
  pf_max_samples 10000
)
```

Para lanzar la ejecución debemos seguir estos pasos:

```
player ejemplo1.cfg
player ejemplo2.cfg –p 6666
playernav localhost:6665 localhost:6666
```

Una vez abierta la interfaz habría que insertar los puntos de destino de los robots, para ellos hemos de pinchar con el botón secundario y arrastrar el triángulo hasta la posición de destino al soltar aparece una línea que nos permite definir la orientación final también.

Ejemplo de simulación con playernav:



Problemas

Utilizando el wavefront, al cambiar la posición del robot no encuentra bien el camino. Alguna vez saltan fallos de conexión entre las aplicaciones y se deben volver a realizar todos los pasos.

Vemos que el problema se soluciona cambiando odom por gps en la localización world, ya que las referencias de las posiciones del mapa no eran las correctas dado que al usar odom la posición de partida se considera siempre en 0.0 , 0.0