Navegación en vehículos no tripulados acuáticos de superficie

Taller de sistemas ciber-físicos 2019

Guillermo Trinidad

Introducción

Objetivo

El objetivo de este proyecto es resolver que un barco no tripulado realice una trayectoria entre dos puntos indicados por sus coordenadas en latitud y longitud. Este deberá ser capaz de esquivar las condiciones de cartografía del lago/laguna/río donde se encuentre y evitar obstáculos que se le puedan presentar dinámicamente en el trayecto.

Alcance

Se trabajó sobre el bote Hidrodata, un emprendimiento uruguayo para el relevamiento de información de las aguas; el cual cuenta con una placa Ardupilot[1], un sensor de GPS, un motor brushless, un timón manejado por un motor servo y un transmisor/receptor de radiofrecuencia para comunicación. Con esto, la placa es capaz de resolver la navegación en línea recta entre dos puntos dados.
En el proyecto se busca:

- Generar rutas que respondan a la cartografía del lugar.
- Mostrar al usuario dichas rutas.
- Integrar la planificación con el bote.
- Que el bote sea capaz de adaptarse a las condiciones dinámicas de su entorno y pueda replanificar su ruta en consecuencia

Para esto se definieron dos hitos en el marco de este desarrollo:

- Hito 1: Planificación de rutas seguras
    - Planificador de ruta
    - Integración con el hardware

- Hito 2: Actualización del mapa

Planificación de rutas seguras

Generación del mapa

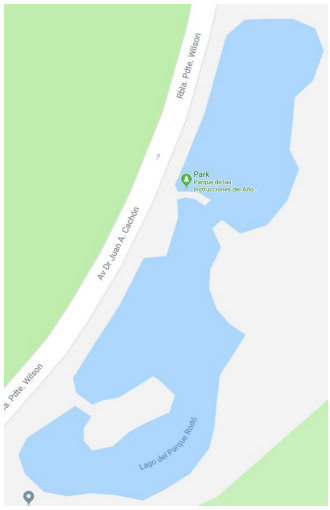
Para poder planificar la mejor ruta entre los dos puntos dados, se necesita tener un mapa del lugar donde se desplazará el bote.
Para integrar un mapa al entorno ROS[2] se utilizó el modelo de grillas de ocupación.

Obtención de los bordes

El primer paso para tener un mapa donde planificar, es crear una imagen en escala de grises a partir de la cual generar la grilla de ocupación. Para esto se probaron diferentes alternativas que automatizan el proceso, como Qgis.[3] Luego se decidió realizar el proceso a mano:

- Encontrar el lago en google maps
- Tomar una captura de la imágen
- Abrirla con un editor (p.e. gimp) y convertirla a blanco y negro.

Con esto logramos pasar de la imagen de la izquierda a la de la derecha, consiguiendo así determinar la forma del lago a navegar.

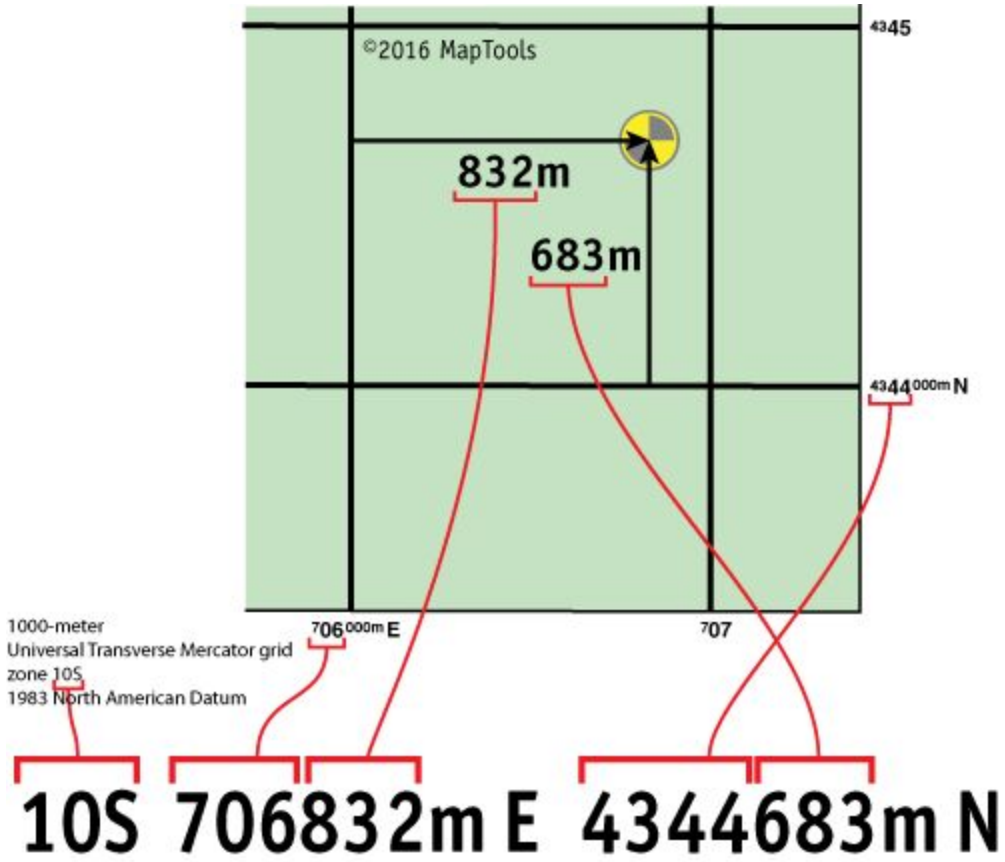


Colocar el mapa en el mundo

El segundo desafío es representar la ubicación del mapa en el mundo real, esto es porque necesitamos que las indicaciones de destino marcadas sobre el mapa obtenido se correspondan con sus coordenadas en el sistema de latitud longitud para que Ardupilot pueda guiarlo de un punto a otro.

Otra vez se decidió guiarse por el standard en ROS, que es manejar coordenadas métricas. Para esto utilizamos el sistema UTM (Universal Transverse Mercator)[4] el cual se basa en dividir el globo en 60 husos y 20 bandas que conforman zonas rectangulares dentro de las cuales un punto puede ser referenciado por la distancia al origen de su zona en metros. Esto nos permite utilizar un sistema métrico dentro de ROS y transformar las coordenadas a sus latitudes y longitudes correspondientes a la hora de enviarlas al bote.

Ejemplo de utilización del sistema UTM:

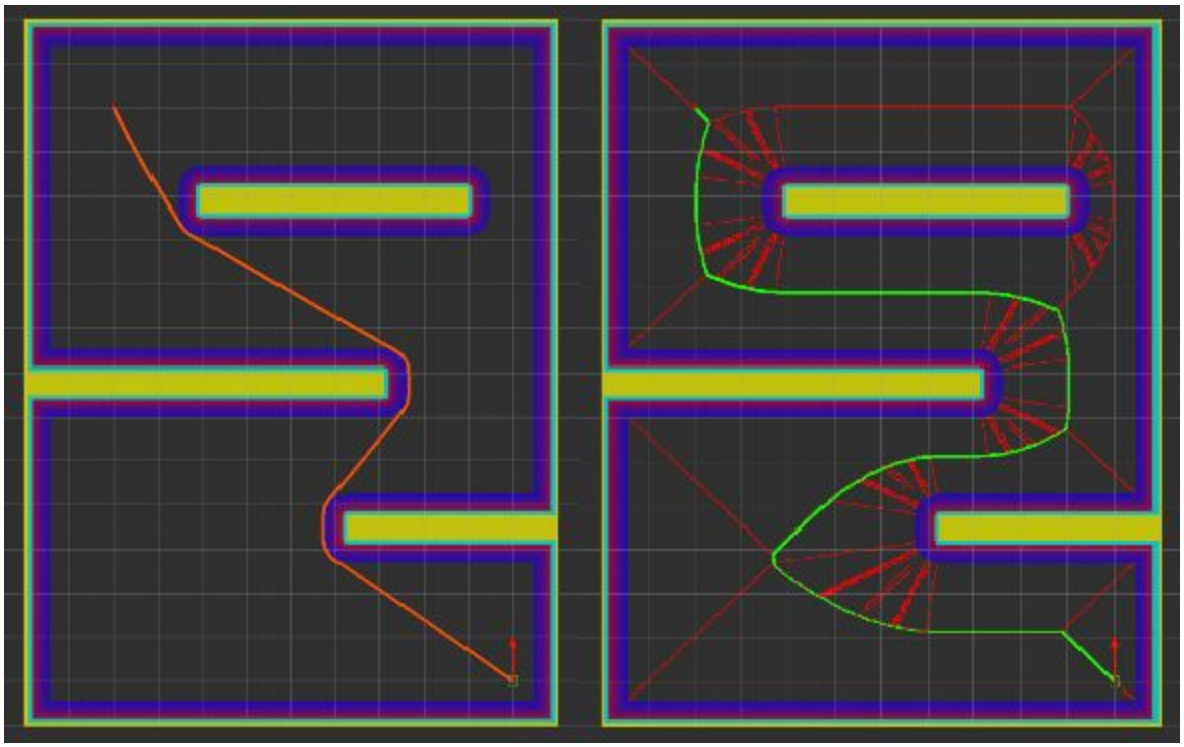


Planificador seguro

En ROS existen muchos paquetes que resuelven la planificación de rutas, teniendo cada uno sus propias virtudes y defectos. Para este proyecto se evaluaron las características necesarias, decidiendo, por ejemplo, que tenía sentido hacer rutas un poco más largas si éstas nos mantenían alejados de los obstáculos y bordes, ya que una corriente podría desviarnos y acercarnos más de lo deseado, generando atascos en la navegación.

Voronoi [5]

El planificador Voronoi tiene como característica principal generar rutas que se encuentran equidistantes de todos los obstáculos del mapa. Esto es particularmente útil cuando no se cuenta con un hardware capaz de realizar maniobras bruscas o simplemente se quiere transitar de manera más segura.



Ejemplo de planificación. A la izquierda un planificador standard (A*), a la derecha la planificación generada por Voronoi.

Si bien en principio puede parecer una buena idea utilizar este planificador, en espacios abiertos puede que la ruta generada sea innecesariamente larga.
Por esto y por simplicidad de la implementación, se decidió seguir otra alternativa.

Alteración del costmap para simular Voronoi

Los paquetes de planificación utilizan grillas de ocupación para encontrar los mejores caminos entre el punto A y el B. Estas grillas pueden considerarse como una matriz donde cada celda tiene una probabilidad de ocupación o un costo y el algoritmo debe encontrar un camino donde la suma de estos valores sea mínima.

Teniendo en cuenta los objetivos de planificación considerados en la parte anterior, se buscó generar un costmap que tuviese genere una planificación similar a Voronoi, pero sin necesariamente estar equidistante a los obstáculos.

Para lograr esto se modificaron dos parámetros en la generación del costmap:

- **Inflation radius,** esto hace que los obstáculos se "inflen" en el mapa, generando una zona de costo elevado a su alrededor.
- **Cost scaling factor,** indica que las celdas infladas tengan un costo inversamente proporcional a la distancia al obstáculo.

Con esto se logra a partir del mapa de la izquierda generar el costmap de la derecha, donde la cercanía a los bordes tiene un costo elevado.

Navfn[6]

Para generar las rutas sobre el nuevo costmap se utilizó uno de los planificadores más utilizados en ROS (Navfn). Este se basa en el algoritmo de Dijkstra[2] para encontrar el camino de costo mínimo.

Ejemplo de planificación en el lago

Integración con el Hardware

MAVRos[7]

Para establecer la conexión entre ROS y Ardupilot (Software utilizado en el bote) se decidió utilizar el paquete MAVRos. Este implementa el protocolo MAVLink[8], comunicándose con la placa y poniendo a disposición varios tópicos donde publicará información de Ardupilot y otros donde podremos colocar información a enviar a la misma.

WayPoints

Ardupìlot maneja la navegación en línea recta de un punto a otro, para especificar los puntos a recorrer, se le especifica una lista de WayPoints, los que recorrerá uno a uno.

Por limitaciones físicas no se puede enviar una ruta arbitrariamente larga de puntos, visto esto se decidió manejar un subconjunto de puntos de la ruta, enviándolos poco a poco a medida que el bote reportaba que había llegado al último de la lista que tenía.

En la práctica, se limitó la cantidad de puntos a 50. Esto representa una porción lo suficientemente grande de recorrido para que la comunicación no se dé con una regularidad innecesariamente grande y se comprobó que era un número de puntos soportado por la placa.

Aquí se ve en amarillo los primeros 50 waypoints de la ruta de los ejemplos anteriores, que está en verde

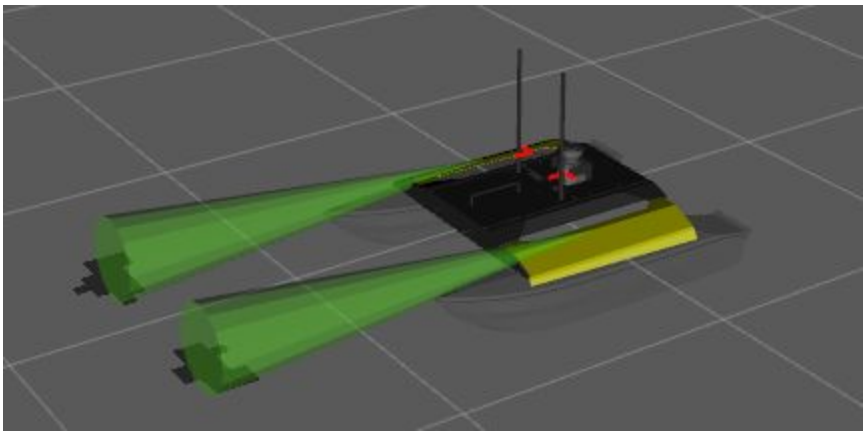
Actualización del mapa

Para ser capaz de adaptarse a cambios en el entorno, el robot deberá contar con sensores que le permitan detectar obstáculos nuevos e integrarlos al mapa, para volver a planificar teniendo en cuenta esos datos.

Integración de sensores de distancia

Para dicha tarea se integraron los sensores ultrasónicos de distancia HC-SR04.
Estos se conectan a una placa Arduino Uno[9], que se encargará de tomar los valores de distancia leídos por los mismos y reportarlos por serial. Se implementó un nodo ROS que se encarga de leer dicha información y transformarla al tipo de mensaje RangeSensor.

Esto último se debe a que se utilizó el plugin Range Sensor Layer[10] de ROS, que interpreta estos mensajes y se encarga de integrar mediante un modelo probabilístico esta información al costmap.



Sensores de distancia integrados al robot HERON. Se observa en verde el cono leído por el sensor y en negro las celdas marcadas como ocupadas en el costmap.

Trabajos a futuro

Integrar sensor sonar

Este tipo de sensor aporta más información sobre el suelo del lago y posibles obstáculos en el medio.

Colocar sensores en el modelo

Las pruebas de integración de sensores se hicieron sobre el modelo del robot HERON, se busca colocarlos en un modelo del robot real.

Probar en exteriores

Las condiciones climatológicas dificultaron las pruebas en exterior, dejándolo como punto pendiente.

Referencias

1. http://ardupilot.org/
2. https://www.ros.org/
3. https://gis.stackexchange.com/questions/241814/downloading-polygon-files-like-lake-or-municipal-borders-from-openstreetmap-in
4. https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system
5. http://wiki.ros.org/voronoi_planner
6. http://wiki.ros.org/navfn
7. http://wiki.ros.org/mavros
8. https://en.wikipedia.org/wiki/MAVLink
9. https://es.wikipedia.org/wiki/Arduino_Uno
10. http://wiki.ros.org/range_sensor_layer