17. Ciencia, Tecnología e Innovación

Vehículos Autónomos De Superficie: Estudio y Comparación de Planificadores Locales de Ruta

Peralta, Federico; fperalta@ing.una.py ;

Universidad Nacional de Asunción

Resumen

En el desarrollo de vehículos autónomos de superficie, los planificadores locales de ruta son importantes, ya que permiten calcular o recalcular rutas cuando hay obstáculos presentes en el entorno. Una evaluación del desempeño de las diferentes técnicas de planificadores locales se presenta, usando un simulador que permite verificar, comparar, medir y visualizar las soluciones que entregan las diferentes técnicas. Las técnicas seleccionadas incluyen A*, Potential Fields, y RRT*. Luego, los resultados son comparados explicando las ventajas y desventajas de cada técnica.

Palabras clave: vehículo autónomo de superficie, planificador local de ruta, planificador de movimiento, navegación autónoma.

Resumen:

En el desarrollo de vehículos autónomos de superficie, los planificadores locales de ruta son importantes, ya que permiten calcular o recalcular rutas cuando hay obstáculos presentes en el entorno. Una evaluación del desempeño de las diferentes técnicas de planificadores locales se presenta, usando un simulador que permite verificar, comparar, medir y visualizar las soluciones que entregan las diferentes técnicas. Las técnicas seleccionadas incluyen A*, Potential Fields, y RRT*. Luego, los resultados son comparados explicando las ventajas y desventajas de cada técnica.

1. Introducción

Numerosos trabajos son realizados con el uso de vehículos no tripulados, incluyendo recolección de datos, transporte, vigilancia y otros. (Liu, 2016). En específico, los ASV (vehículo autónomo de superficie) son vehículos no tripulados con sistemas autónomos de control usados en ríos, lagos y mares para tareas como monitoreo ambiental (Fraga, 2004), vigilancia (Yaakob, 2012), batimetría (Ferreira, 2009), también como base móvil para otros vehículos autónomos (Pinto, 2014). Además, ellos pueden actuar como repetidoras o extensoras en grupos de vehículos como el trabajo de Busquets (2013).

Un vehículo es autónomo si posee sensores y actuadores que responden de forma coherente y planificada. La coherencia se refiere a que los actuadores se despliegan y actúan de la misma manera para las mismas situaciones, y la planificación a las formas y procedimientos que el robot desarrolla para cumplir con un objetivo definido.

Respecto a la planificación de movimiento, Mac (2016) afirma que las técnicas se clasifican en diferentes niveles de grupos. En el primer nivel, se describe el planificador global (offline) y el planificador local (online). Estas dos técnicas pueden ser consideradas complementarias y deberán ser utilizadas en la implementación de planificación de rutas para un ASV real.

Este trabajo evalúa las técnicas más recientes de planificadores de ruta locales. Este estudio forma parte del proyecto de desarrollo e implementación de un ASV para monitoreo ambiental en Lagos.

2. Objetivos

2.1. Objetivo General

- Comparar modelos actuales de planificadores locales de ruta en ambientes acuáticos.

2.2. Objetivos Específicos

- Implementar algoritmos de planificadores de ruta recientes.
- Evaluar, mediante simulaciones experimentales, variables de tiempo y distancia relacionadas al manejo de un ASV.

3. Materiales y métodos

3.1. Trabajo Relacionado

Conforme a Liu (2016), para un ASV se deben tener en cuenta aspectos como el modelo de control, las propiedades del vehículo y las técnicas de navegación. Esta última hace referencia a la planificación de movimiento.

La planificación de movimiento incluye el *planificador local de ruta*, definido, según Sedighi (2004), como un algoritmo para planificar una ruta libre de colisiones, y satisfacer ciertos criterios de optimización, entre 2 puntos específicos. En estos días, existen numerosos métodos y algoritmos de planificación local de ruta bien estudiados como el Dijkstra (1959), RRT (Adiyatov, 2017), Potential Fields (Barraquand, 1992).

La diferencia entre los métodos se puede concluir en cómo se utilizan los datos mínimos para encontrar una ruta óptima. Estos datos incluyen un mapa, las posiciones finales e iniciales, y las restricciones de movimiento. Sedighi (2004). De manera a proveer a los planificadores los datos mínimos, un complejo simulador se desarrolló en el Laboratorio de Sistemas Distribuidos. Se puede considerar este trabajo como la continuación del trabajo de Arzamendia (2017), en donde se definen 60 puntos de control, en el perímetro interno del Lago Ypakarai y un orden para visitarlos de tal manera a cubrir la mayor área posible. Esta es la ruta global, y se usa en el simulador, que posee un mapa escalado del Lago Ypakarai. Luego un planificador local es

seleccionado y ejecutado para obtener una ruta local. Esta ruta es recorrida por el ASV simulado, mientras se toman mediciones para la comparación.

3.2. Implementación de planificadores locales de ruta.

En esta sección se implementan los planificadores en el simulador. Las simulaciones fueron realizadas usando una computadora con Procesador Intel Core i7-7700HQ @ 2.8Ghz y memoria RAM 8GB.

Las simulaciones usan el mismo set de puntos, y el objetivo del ASV es el de visitar cada punto una vez, mientras se toma en cuenta el tiempo tardado en obtener una ruta entre puntos, el tiempo total tardado, incluyendo la simulación, y la distancia total recorrida.

3.2.1. Persecución Pura

Para tener un modelo ideal con que comparar, el algoritmo de persecución pura (PP) es implementado, en donde no se evitan los obstáculos presentes, si los hubiere, entonces, el ASV maneja directamente hacia cada punto de referencia. La ecuación (1) muestra cómo se calcula la velocidad v, teniendo una ganancia k, una $meta$, y una $posicion$ actual, estas últimas dos variables son definidas por las posiciones $\{x, y\}$ en una región definida $\mathbb{R}^2$, usando además una función para obtener la norma de la diferencia de posiciones. Esta ecuación es utilizada en el Algoritmo (1), que maneja de forma autónoma al ASV de una meta a la siguiente.

$$v = k\frac{meta - posicion}{norm(meta - posicion)} \quad (1)$$

```
1. inicializar();
2. beacons = obtener_posiciones_de_metas();
3. meta = 1;
4. Mientras (meta < 60) {
5.     pose = simulator.obtener_pose();
6.     Si (pose - beacons(meta) < error_permitido), meta++;
7.     dx = ganancia*(beacons(meta)-pose) /
   norm(beacons(meta)-pose);
8.     simulator.definir_velocidades(dx);
9.     simulator.paso();}
```

Algoritmo 1: Persecución Pura.

En donde, después de la inicialización, el simulador corre mientras que no se visiten todas las boyas, obteniendo la posición actual del ASV y la meta y luego calculando la velocidad deseada para llegar a la meta actual.

La figura 1 muestra la visualización del simulador, mostrando el ASV en rosado, las boyas como asteriscos verdes, las rutas planificadas como líneas rojas y las metas a visitar como cuadrados rojos. El PP maneja el ASV direccionándolo directamente de boya a boya como es de esperarse.

Figura 1: Simulador usando PP.

3.2.2. A*

A* es un planificador local de ruta que agrega una función heurística al algoritmo de Dijkstra, el cual busca una ruta entre dos puntos, examinando los vecinos inmediatos de una posición inicial, luego estos vecinos son considerados como iniciales y se examinan sus vecinos, hasta encontrar la posición de la meta, uniendo el camino existente entre la posición inicial y la final con los nodos examinados.

En lugar de examinar en todas las direcciones al agregar un vecino nuevo, A* utiliza una fila, que prioriza la examinación de nodos mas cercanos a la meta, o que poseen unos menores *costos de movimiento*, en este caso, el costo es definido por las restricciones de movimiento del robot. El programador puede, por ejemplo, asignar mayores costos a nodos que están en las cercanías de los limites u obstáculos. Estos costos son sumados y asignados como la prioridad de los nodos, obteniendo nodos de menores costos con mayores prioridades (ya que el nodo con prioridad 1 es el nodo con mayor prioridad). De esta manera, existe una probabilidad mayor de llegar en menor tiempo a la meta, manteniendo los niveles de seguridad y convergencia que el Dijkstra ofrece.

Un pseudo-código es presentado (Algoritmo 2), en donde se crea una fila con prioridad, y en ella se agregan los próximos nodos a ser examinados. Luego de examinar el nodo de la meta, una ruta es generada recorriendo desde el nodo final hasta el inicial, moviéndose hacia el nodo de menor costo en cada ciclo.

```
1.  pos_node = Node(pos_pose, prioridad, parent_node);
2.  frontera = PriorityQueue(pos_node);
3.          nodosExaminados = pos_node;
4.          Mientras (frontera_no_vacia){
5.                  nodoActual = frontera.pop();
6.                  Si (nodoActual.position == meta.position}{
7.                          nodosExaminados.add(nodoActual);
8.                          break;}
10.                 Por Cada (sgteNodo en vecinosDe(nodoActual,
    mapa)) {
11.                         priodidad = obtener_costo(sgteNodo);
12.                         frontera.push(sgteNodo, prioridad)
13.                         nodosExaminados.add(sgteNodo);
14.                 }
15.         }
```

Algoritmo 2: A*.

Para esta implementación y las siguientes, se realiza una modificación al algoritmo PP, en donde se agrega la planificación de ruta, y esta planificación es llamada cada vez que el ASV llega a una de las 60 metas, Algoritmo 3. El resultado de utilizar este algoritmo en conjunto con la planificación de ruta A*, en el simulador, es observado en la Figura 2. Es importante notar que puntos a visitar adicionales se crearon con esta planificación, debido a la naturaleza de algoritmos basados en grillas con búsquedas en ángulos definidos.

```
1. inicializar();
2. beacons = obtener_posiciones_de_metas();
3. meta = 1;
4. pose = simulator.obtener_pose();
5. path = p_planner(mapa, pose, beacons(meta), 'Astar');
6. Mientras (meta < 60){
7. pose = simulator.obtener_pose();
8. objetivo = obtener_obj(path, pose, meta, 'Astar');
9. Si(pose - beacons(meta) < error_permitido){
10. meta++;
11. path = p\_planner(mapa, pose, beacons(meta));}
12. dx =  ganacia*(objetivo-pose)/norm(objetivo-pose);
13. simulator.definir_velocidades(dx);
14. simulator.paso();}
```

Algoritmo 3: Ejecución del simulador con Planificador de ruta

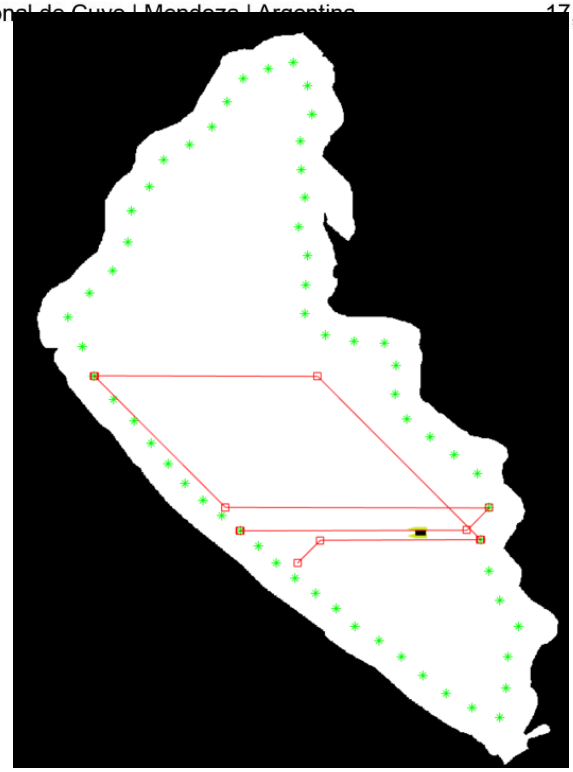

Figura 2: A* en ejecución.

3.2.3. Potential Fields

Planificadores de ruta con campos potenciales, o, en inglés, Potential Fields (PF), son utilizados por los beneficios y las mejoras que ofrece. La idea del PF consiste en obtener valores potenciales para cada punto dentro de la región de manejo, teniendo en cuenta dos tipos de campos: los repulsivos, y los atractivos. Uno de las primeras investigaciones (Barraquand, 1992) define 2 principales funciones: ecuación 2, y ecuación 3, como las ecuaciones de atracción y repulsión que posee un punto, por lo tanto, se deben sumar y asignar como valor potencial del punto p.

$$U_{atr} = \frac{1}{2} * k_a * pdist(p, meta) \quad (2)$$

$$U_{rep} = \frac{k_r}{pdist\,(p, obstaculo_i)} \quad (3)$$

En estas ecuaciones, k_a y k_r son constantes de atracción y repulsión, y $pdist$ es la distancia euclidiana entre la *posición* examinada y la *meta* actual, o el *obstáculo* i, según sea el caso. Una

ventaja del PF es que se controla específicamente zonas de peligro (obstáculos, zonas no visitables, etc.) con la ecuación 3. Un mapa potencial es calculado para cada meta, como es observado en la Figura 3., y es usado para determinar en qué dirección debe ir el ASV, haciéndolo moverse al punto vecino de menor potencial en cada iteración, haciéndolo viajar hacia la meta, que posee potencial ($U_{atr} = 0$). En esta figura, se observa que, para la primera situación (la letra S en cian indica el punto inicial y la letra amarilla G, la meta), la zona de la meta es más negra indicando menor potencial, y los obstáculos o zonas que son menos favorables están son de color más blanco. La Figura 4. es generada luego de implementar este algoritmo, y utilizar el mapa potencial para indicar la dirección de viaje del ASV, cuando este llega a una meta, el siguiente mapa potencial es generado y se vuelve a repetir hasta visitar los 60 puntos. En este algoritmo, no se generan puntos de visita intermedios como sucedía en A*.

Figura 3: Mapa potencial inicial.

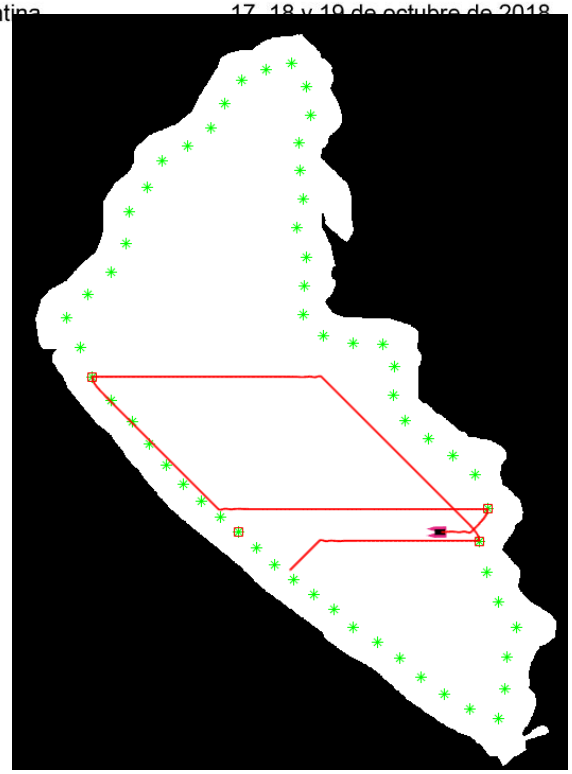

Figura 4: PF en ejecución.

3.2.4. Rapid-Exploring Random Trees *

Uno de los planificadores basados en muestras más populares es el Rapid-Exploring Random Tree (RRT), mencionado por Adiyatov (2017), o árbol aleatorio de exploración rápida, este método no solo provee una ruta rápidamente, sino que también requiere de muy poca energía computacional en comparación con otros planificadores. RRT hace crecer un árbol, con raíz en la posición inicial, con nodos y ramas, estos nodos son creados aleatoriamente y agregados al árbol hasta que un nodo aleatorio se conecte con la posición final. De forma a mejorar el tiempo de cálculo, RRT* es una modificación que agrega funciones de costos y elecciones de nodos padre.

Los algoritmos principales de RRT* se muestran en los siguientes algoritmos siendo el Algoritmo 4., el principal, el Algoritmo 5., la función *steer*, y Algoritmo 6., la función para elegir un nodo padre. En RRT*, un árbol τ es inicializado, conteniendo como raíz al nodo inicial, luego una posición aleatoria en obtenida dentro de la región, luego se adapta esta posición por medio una función *steer*, para ubicar este nodo nuevo lo más cerca posible de un nodo del árbol, luego como Adiyatov (2017) sugiere, el nuevo nodo adaptado obtiene un costo y se conecta al mejor

nodo padre que se encuentra en la vecindad de este nodo, este nodo padre es elegido mediante la función *elegirPadre*, por último, este nodo es conectado al árbol y luego se comienza de nuevo la iteración.

```
1.  τ = inciarArbol();
2.  insertarNodo(pose_inicial, τ);
3.  Para (i=1; i<maxiter; i++) {
4.    p_aleat= muestra_aleatoria_de_mapa();
5.    [p_aleat, p_cercano] =
                  obtenerCercanosySteer(τ, p_aleat);
6.    Si (caminoPosible(p_aleat, p_cercano)) {
7.      n_cercanos = obtenerCercanos(p_aleat, τ);
8.      p_min = elegirPadre(p_aleat, n_cercanos);
9.      insertarNodo(p_aleat,p_min, τ);
10.     Si (p_aleat == meta), break;}
11. }
12. calc_ruta(τ, meta);
```

Algoritmo 4: Algoritmo principal para RRT*.

```
1.  pos = p_aleat;
2.  Si (dist(p_cercano, p_aleat) > maxDist){
3.      angulo = atan(p_aleat, p_cercano);
4.      pos = p_cercano +
        signo(p_aleat-p_cercano)*maxDist *sin_y_cos(angulo); }
5.  return(pos);
```

Algoritmo 5: Función steer.

```
1. mejor_pd = p_cercano;
2. costo_m = p_cercano.costo + dist(p_cercano.position –
   p_aleat);
3. Para (i=1; i<size(n_cercanos); i++){
4.     costo_n = n_cercanos(i).costo +
           dist(n_cercanos(i).position – p_aleat)\;
5.     Si (costo_n < costo_m){
6.         [costo_m, mejor_pd] = [costo_n, n_cercanos(i)];}
7. }
8. return(mejor_pd);
```

Algoritmo 6: Función elegirPadre.

RRT es uno de los planificadores que puede calcular una ruta en direcciones de cualquier ángulo, significando que el robot es instruido de viajar en cualquier dirección al contrario de los anteriores planificadores. La Figura 5. muestra la ruta generada para las primeras metas. Es muy claro que RRT* agrega muchos nodos innecesarios, pero, por lo observado, provee una ruta más corta entre los puntos de cálculo.

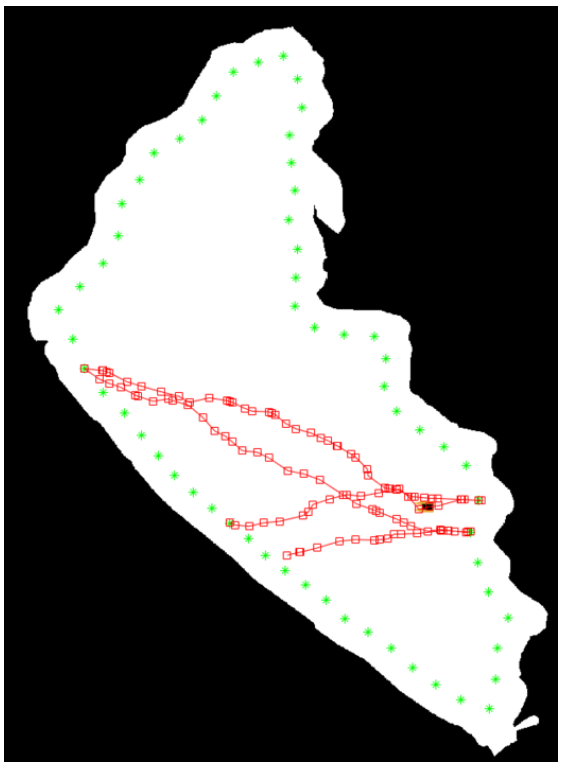

Figura 5: RRT en ejecución.

4. Resultados y discusiones

En la Tabla I se muestra una comparación de las distintas técnicas de planificación de ruta, en donde se indican el método utilizado, el tiempo promedio de cálculo de rutas, el tiempo total de simulación, la distancia total viajada, donde 1 pixel es aproximadamente 10.33 metros, y por último un nivel de seguridad o confianza, teniendo en cuenta los giros y cambios de direcciones, las cercanías a obstáculos, entre otros criterios.

Planificador	T. prom. de C. (s)	T. total (s)	Distancia v. (m)	N. de Seg.
A*	0.4266	71.67	402 784	☑☑☑☑☑
PF	0.9609	104.51	402 788	☑☑☑☑
RRT*	0.4184	66.51	390 386	☑☑☑
PP	—	43.66	382 310	—

Tabla I: Comparación de métodos.

Entre los métodos que incluyen detección de obstáculos, A* es el más seguro por la forma de análisis de los nodos, la distancia recorrida es 5,2% veces mayor comparando con PP, recordando que PP es el caso idealizado. En el caso de PF, se obtiene una ruta simular a las de A* pero requiriendo mayores potencias y tiempos de procesamiento. Finalmente, considerando RRT*, y teniendo en cuenta que este provee diferentes rutas en cada iteración, en promedio se observa que la distancia es menor a la recorrida por A* (3,1% menor), y el tiempo promedio de cálculo es también menor.

La Figura 6. presenta una comparación de los tiempos de cálculo de rutas, en el eje de las abscisas. Cada asterisco representa el tiempo tardado para obtener una de las 60 rutas. Con esta figura y la Tabla I se encuentra que el método PF tiene una pequeña desviación estándar de solo $\sigma = 0.1$, mientras que A* y RRT* poseen desviaciones mayores, $\sigma = 0.42$ y $\sigma = 0.58$, respectivamente, esto sucede debido que el tiempo de cálculo aumenta con la distancia entre nodos, para estos últimos dos, mientras que PF siempre calcula un mapa completo, por lo que es de esperar que posea una pequeña desviación.

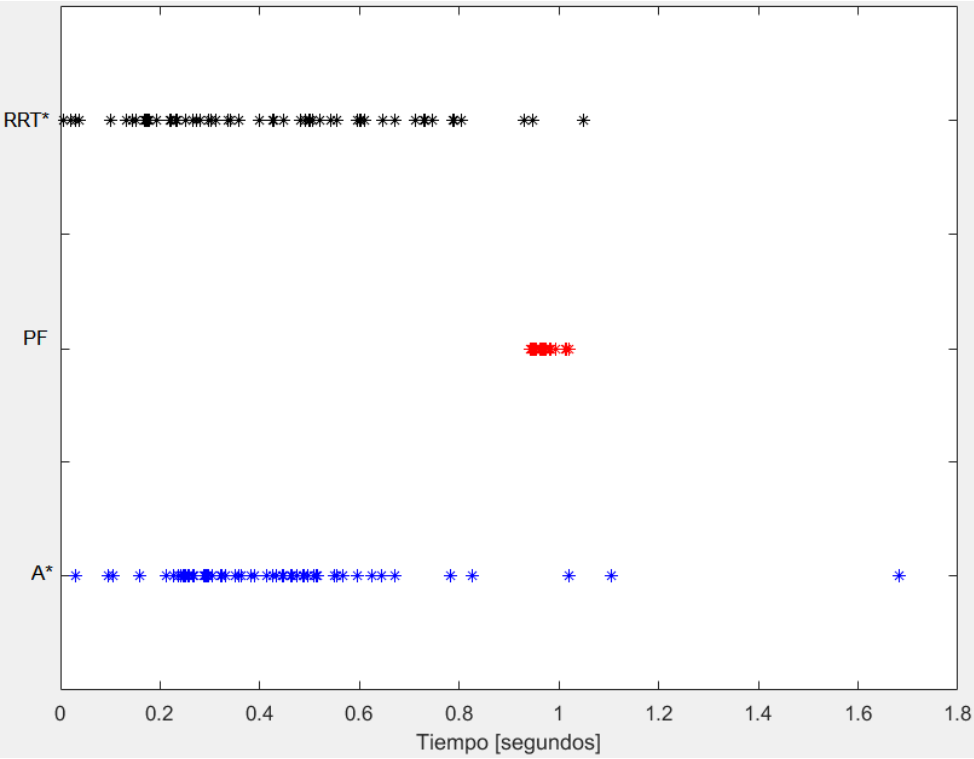

Figura 6: Tiempo de cálculo de las 60 rutas.

5. Conclusiones y trabajos futuros

Técnicas de planificación de rutas recientes se implementaron y simularon luego de estudiarlas en profundidad durante el desarrollo de un ASV para el monitoreo y control de la calidad del agua en lagos, como el Lago Ypakarai. Los resultados muestran que en el diseño se debe de elegir entre obtener un planificador seguro, o un planificador de bajos requerimientos de energía y tiempo. De las simulaciones, RRT* demuestra ser la mejor técnica, debido a la rápida convergencia, pero se necesitan mejoras para obtener rutas más seguras y confiables. Por otro lado, A* posee el mejor nivel de confianza y las maniobras entregadas son posibles, pero se tarda más tiempo en determinar rutas. Mayor investigación puede realizarse en el simulador desarrollado, siempre teniendo en cuenta que la idea final consiste en desarrollar un ASV real.

Bibliografía

1. Liu, Z., Zhang, Y., Yu, X., & Yuan, C. (2016). Unmanned surface vehicles: An overview of developments and challenges. *Annual Reviews in Control*, *41*, 71-93.
2. Fraga, J., Sousa, J., Cabrita, G., Coimbra, P., & Marques, L. (2014). Squirtle: An asv for inland water environmental monitoring. In *ROBOT2013: First Iberian Robotics Conference* (pp. 33-39). Springer, Cham.
3. Yaakob, O., Mohamed, Z., Hanafiah, M. S., Suprayogi, D., Abdul Ghani, M., Adnan, F., ... & Din, J. (2012, October). Development of unmanned surface vehicle (USV) for sea patrol and environmental monitoring. In *Proceedings of the International Conference on Marine Technology, Kuala Terengganu, Malaysia* (pp. 20-22).
4. Ferreira, H., Almeida, C., Martins, A., Almeida, J., Dias, N., Dias, A., & Silva, E. (2009, May). Autonomous bathymetry for risk assessment with ROAZ robotic surface vehicle. In *Oceans 2009-Europe* (pp. 1-6). IEEE.
5. Pinto, E., Marques, F., Mendonça, R., Lourenço, A., Santana, P., & Barata, J. (2014, December). An autonomous surface-aerial marsupial robotic team for riverine environmental monitoring: Benefiting from coordinated aerial, underwater, and surface level perception. In *Robotics and Biomimetics (ROBIO), 2014 IEEE international conference on* (pp. 443-450). IEEE.
6. Busquets, J., Zilic, F., Aron, C., & Manzoliz, R. (2013, June). AUV and ASV in twinned navigation for long term multipurpose survey applications. In *OCEANS-Bergen, 2013 MTS/IEEE* (pp. 1-10). IEEE.
7. Mac, T. T., Copot, C., Tran, D. T., & De Keyser, R. (2016). Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, *86*, 13-28.
8. Sedighi, K. H., Ashenayi, K., Manikas, T. W., Wainwright, R. L., & Tai, H. M. (2004, June). Autonomous local path planning for a mobile robot using a genetic algorithm. In *Evolutionary Computation, 2004. CEC2004. Congress on* (Vol. 2, pp. 1338-1345). IEEE.
9. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269-271.
10. Adiyatov, O., & Varol, H. A. (2017, August). A novel RRT*-based algorithm for motion planning in Dynamic environments. In *Mechatronics and Automation (ICMA), 2017 IEEE International Conference on* (pp. 1416-1421). IEEE.
11. Barraquand, J., Langlois, B., & Latombe, J. C. (1992). Numerical potential field techniques for robot path planning. *IEEE transactions on systems, man, and cybernetics*, *22*(2), 224-241.
12. Arzamendia, M., Gregor, D., Reina, D. G., & Toral, S. L. (2017). An evolutionary approach to constrained path planning of an autonomous surface vehicle for maximizing the covered area of Ypacarai Lake. *Soft Computing*, 1-12.

Financiamiento

Este proyecto está financiado por el Consejo Nacional de Ciencia y Tecnología del Paraguay (CONACYT). Los autores agradecen por el apoyo para el desarrollo del proyecto PINV15-177.

Agradecimiento

El autor de este trabajo desea agradecer plenamente al Dr. Investigador Derlis Gregor, al Investigador en Formación Ing. Kevin Cikel, ambos del Laboratorio de Sistemas Distribuidos de la FIUNA, al Dr. Daniel Gutierrez Reina, investigador de la Universidad de Loyola de Andalucía y al Dr. Serio Toral de la Universidad de Sevilla, España por la incansable ayuda recibida en términos técnicos, administrativos y académicos.