Planificación

Dr José Martinez Carranza
carranza@inaoep.mx

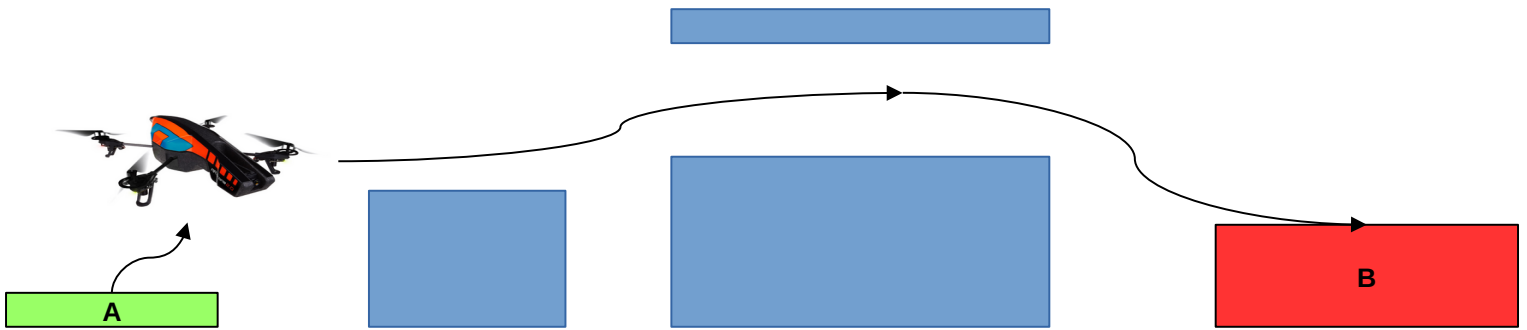
Dr Alejandro Gutiérrez Giles
alejandro.giles@inaoep.mx

Otoño 2024

Coordinación de Ciencias Computacionales, INAOE

Planificación

- Capacidad importante que permite a un robot móbil autónomo el encontrar el camino (más corto u óptimo) entre dos puntos.
- Un camino óptimo puede ser aquel que minimize ciertas maniobras, que ahorre recursos o tiempo. Por ejemplo, un camino óptimo es aquel que busca minimizar:
  - El número de vueltas.
  - El número de veces que hay que frenar al robot.
  - El uso de batería.
  - El tiempo de recorrido.

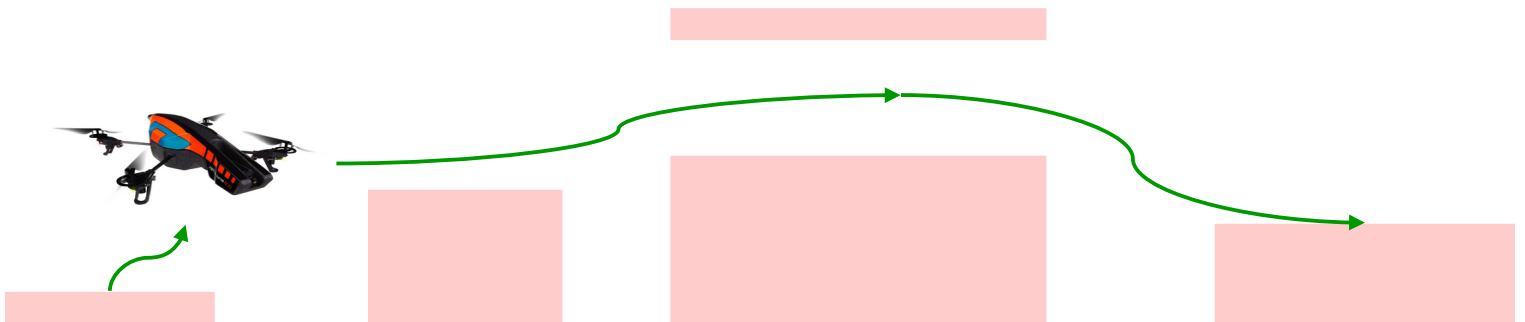


Planificación

- Requerimientos:
  - Mapa del Ambiente

Planificación

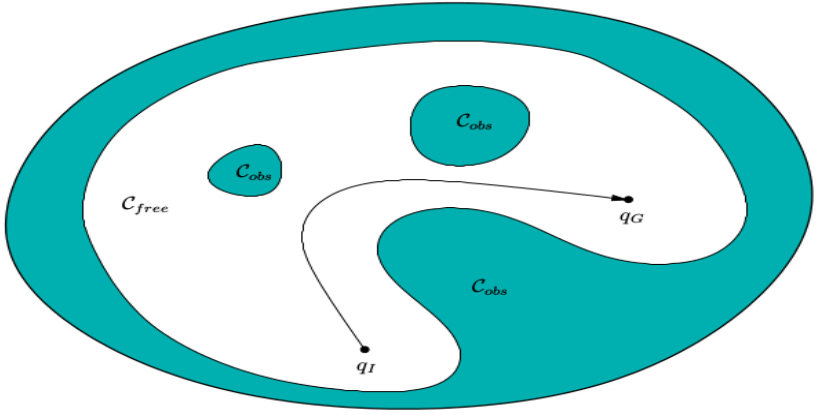
- Requerimientos:
  - Mapa del Ambiente
  - Posición del Robot con respecto al ambiente (ubicación en el mapa)



Planificación

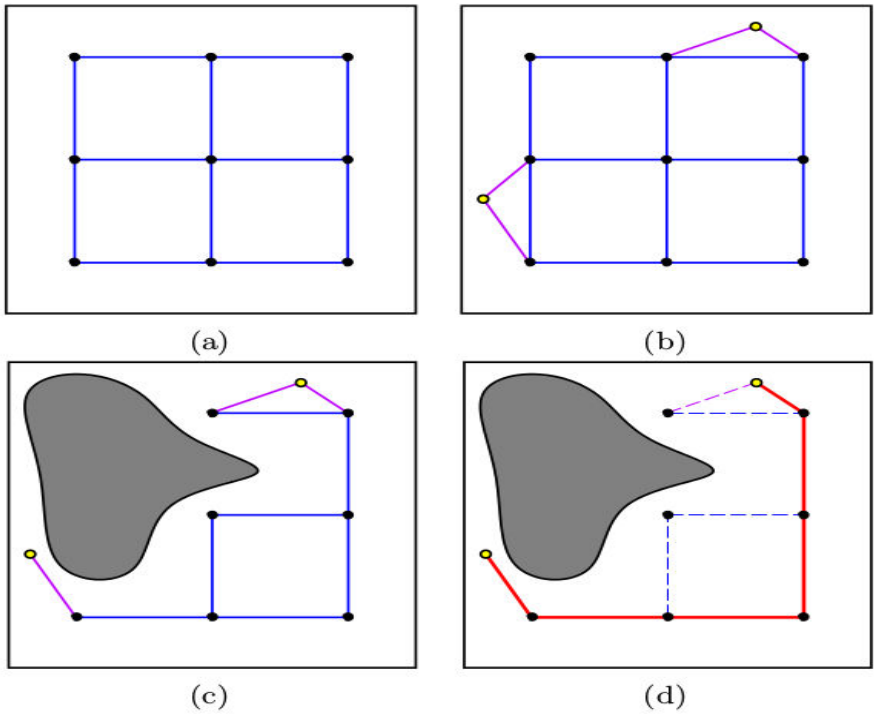
- Dados:
  - Espacio libre C_{free}
  - Configuración Inicial $\mathbf{q}_I$
  - Configuración Final $\mathbf{q}_G$
- Meta:
  - Encontrar un camino continuo que conecte la configuració inicial con la configuración final

Espacio de configuraciones



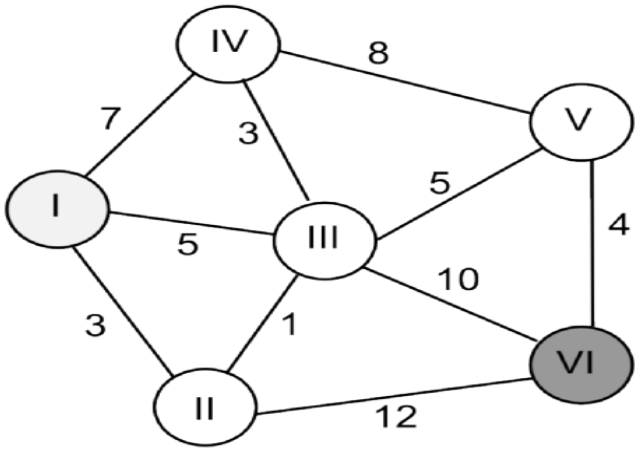
Discretización del Espacio de Configuraciones

- Discretización:
  - Creación de rejillas de ocupación sobre el espacio de configuraciónes.
  - Modelado de la rejilla a través de un grafo.
  - Las celdas de la rejilla son los nodos.
  - Las aristas representan que se puede mover de una celda a otra.
- Planificación:
  - Elegir un nodo como *inicio* y otro como *meta*.
  - Si los nodos de inicio y meta no estan en el grafo entonces conectarlos a los *k* nodos más cercanos.
  - Encontrar un camino mediante algoritmos de búsqueda en grafos.



Algoritmos Populares

- Algoritmos completos (Problema modelado como un grafo)
  - Dijkstra
  - A*
  - D*
  - Depth-first-search (Búsqueda por profundidad primero)
  - Breadth-first-search (Búsqueda por niveles primero)



- **Algoritmo Completo:** Una algoritmo es considerado completo si para cualquier entrada, éste indica de manera correcta *si existe una solución o no*, y si existe, si se puede obtener en una cantidad finíta de tiempo (Lavalle, 2006).

Dijkstra

```
        procedure Dijkstra;
           { Dijkstra computes the cost of the shortest paths
            from vertex 1 to every vertex of a directed graph }
        begin
(1)         S := {1};
(2)         for i := 2 to n do
(3)             D[i] := C[1, i]; { initialize D }
(4)         for i := 1 to n-1 do begin
(5)             choose a vertex w in V-S such that
                  D[w] is a minimum;
(6)             add w to S;
(7)             for each vertex v in V-S do
(8)                 D[v] := min(D[v], D[w] +
C[w, v])
           end
        end; { Dijkstra }
```

Tomado de [1]

A*

Search Algorithm A:*

1) Mark s "open" and calculate $\hat{f}(s)$.

2) Select the open node n whose value of $\hat{f}$ is smallest. Resolve ties arbitrarily, but always in favor of any node $n \in T$.

3) If $n \in T$, mark n "closed" and terminate the algorithm.

4) Otherwise, mark n closed and apply the successor operator Γ to n. Calculate $\hat{f}$ for each successor of n and mark as open each successor not already marked closed. Remark as open any closed node n_i which is a successor of n and for which $\hat{f}(n_i)$ is smaller now than it was when n_i was marked closed. Go to Step 2.

$$f(n) = g(n) + h(n)$$

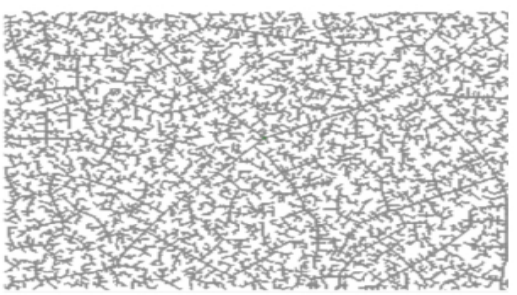
Tomado de [2]

Limitaciones en el uso de rejillas

- Si la solución existe será encontrada pero:
  - No escalan bien para espacios de configuración con varias dimensiones
  - Las soluciones no siempre son completas, la mayoría de las veces estarán limitadas por la resolución de la rejilla
    - Una celda (nodo) en el mapa (grafo), tomará el tamaño que más se adecue a la aplicación, por ejemplo: en un mapa bidimensional una celda podría ser de 1 cm^2.
    - Una solución encontrada en un mapa de alta resolución se aproximará a la solución completa, pero entre mas resolución se tenga más tiempo se llevará la búsqueda.
    - La resolución del mapa podría dejar fuera a la solución completa fuera del espacio de búsqueda.

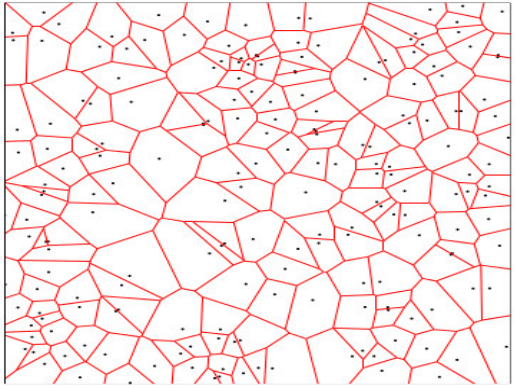
Planeación Basada en Muestreo

- Idea principal: agregar nodos de manera aleatoria a un árbol hasta que una solución sea encontrada o el tiempo expire.
- Solución *probabilisticamente completa,* la probabilidad de encontrar un camino (solución) se acerca a 1 conforme pasa el tiempo.
- Ejemplos:
  - Probabilistic Random Maps (PRM)
  - Rapidly-exploring Random Trees (RRT)

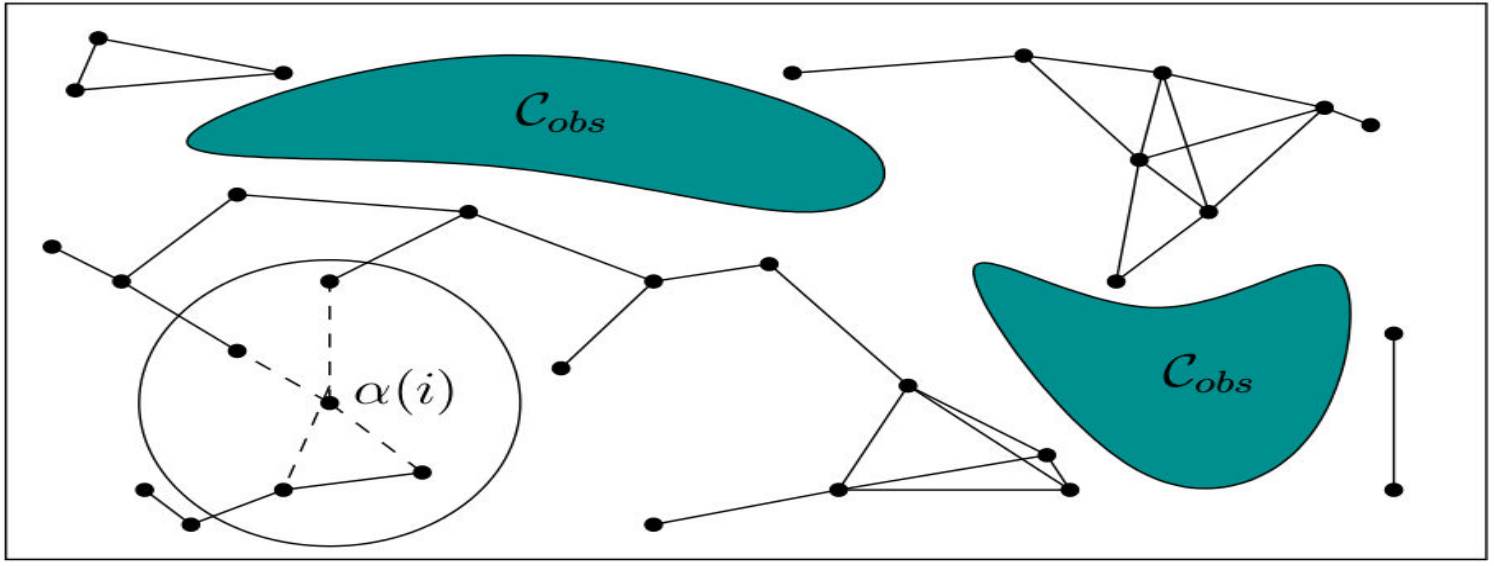


Probabilistic Road Maps PRM

- Solución basada en grafos.
- Vertices: muestras aleatorias del espacio de configuraciones las cuales se validan para saber si estan en espacio libre.
- Aristas: conectan dos vertices vecinos con un segmento de línea si dicha linea no colisiona con el espacio de obstáculos.
- IDEA: conectar vertices libres de colisión y agregarlos al mapa de caminos hasta que se alcance una densidad suficiente.



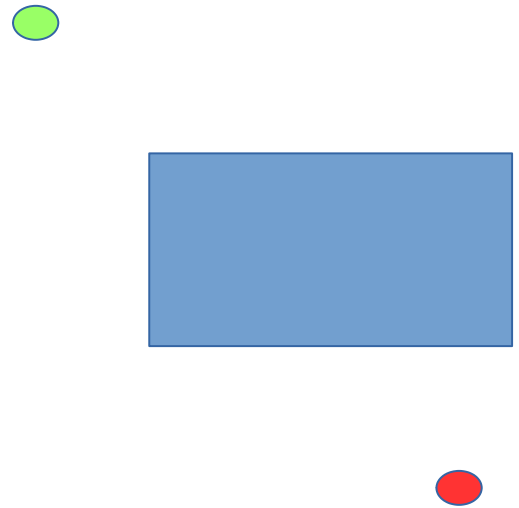
Probabilistic Road Maps



¿Una vez construido el mapa o los mápas, como se encuentra el camino?

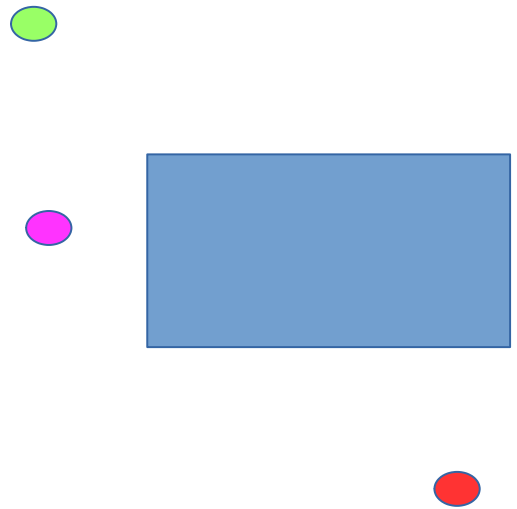
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



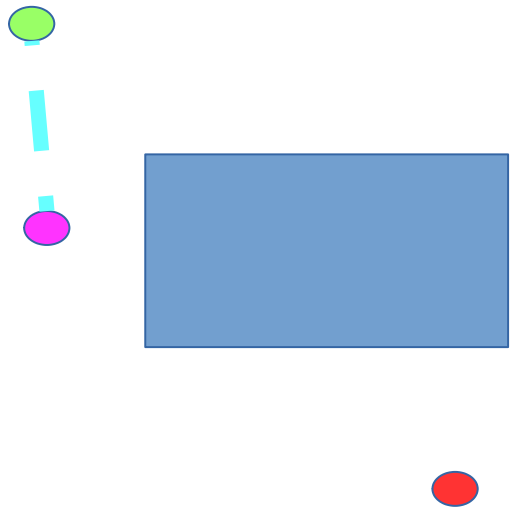
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



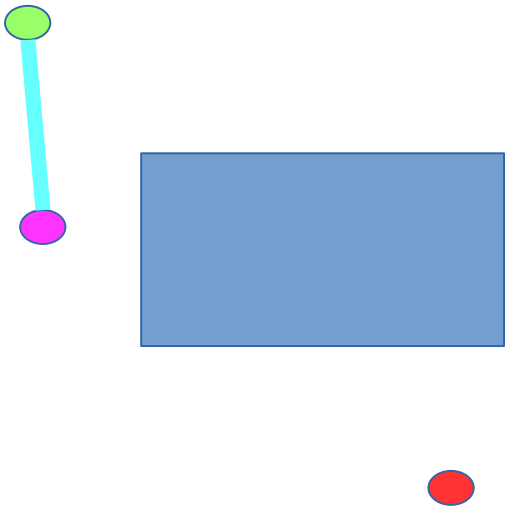
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



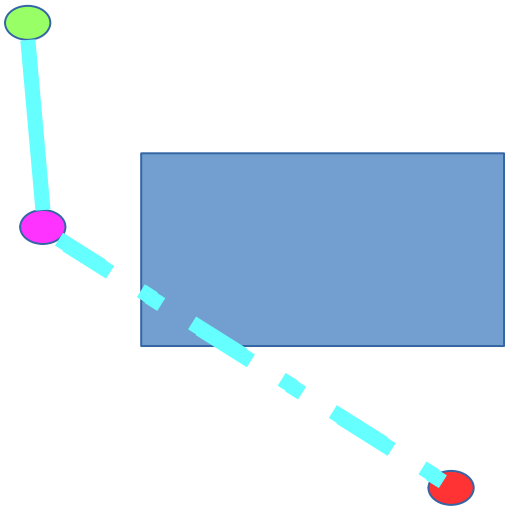
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



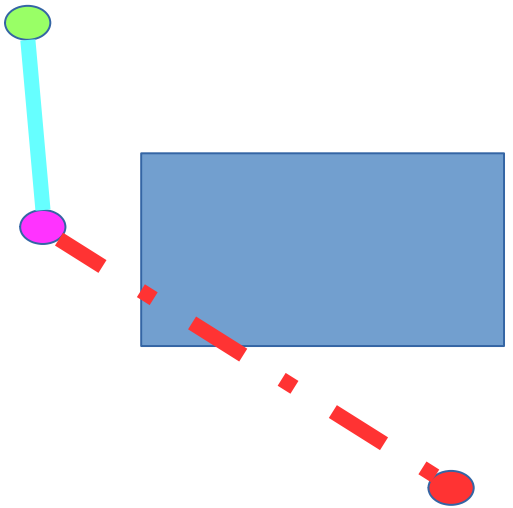
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



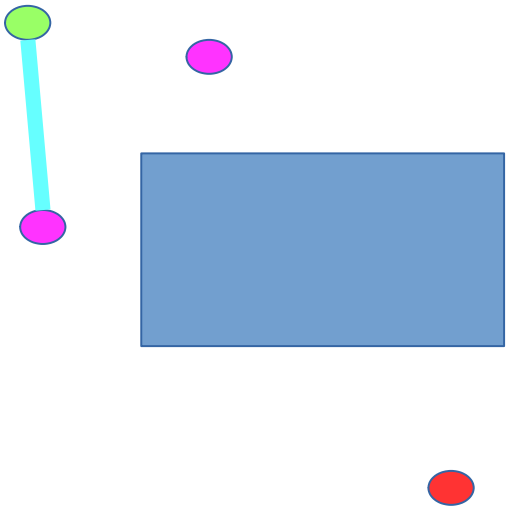
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



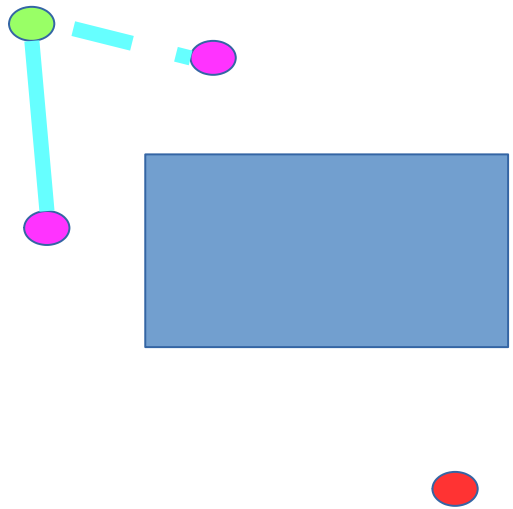
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



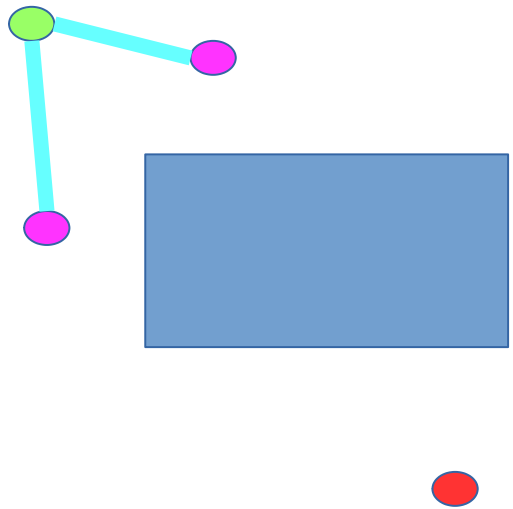
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



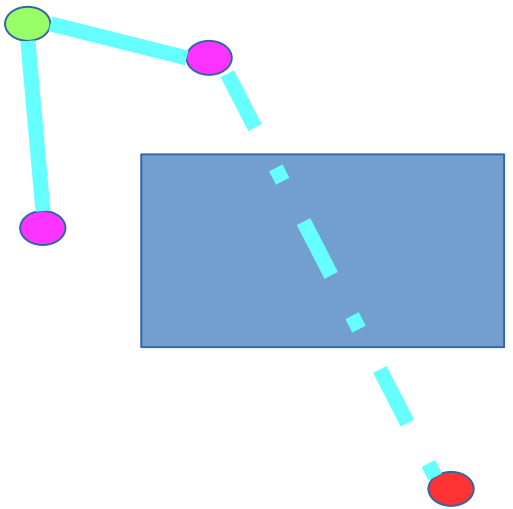
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



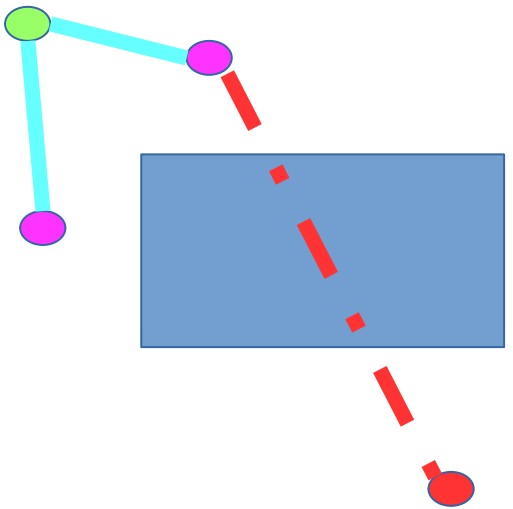
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



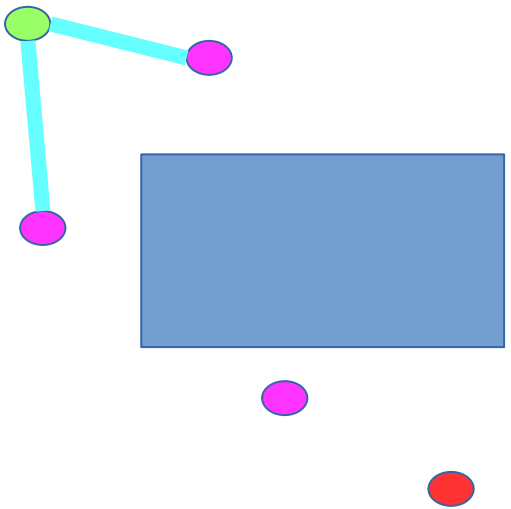
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



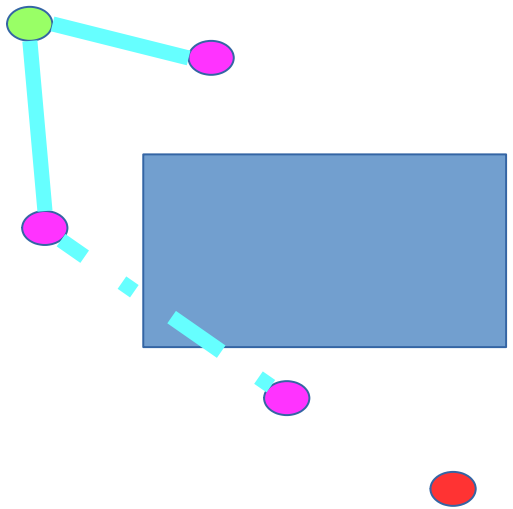
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



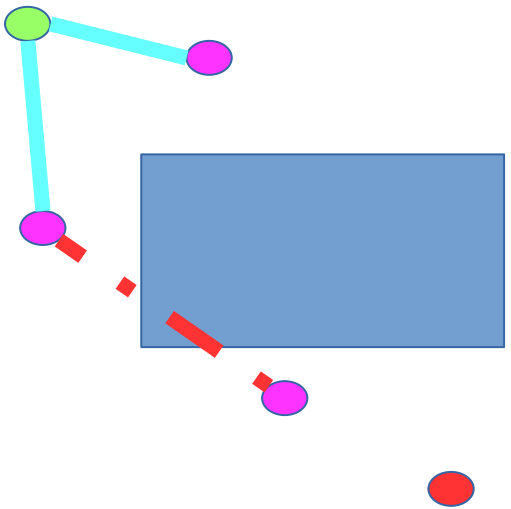
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



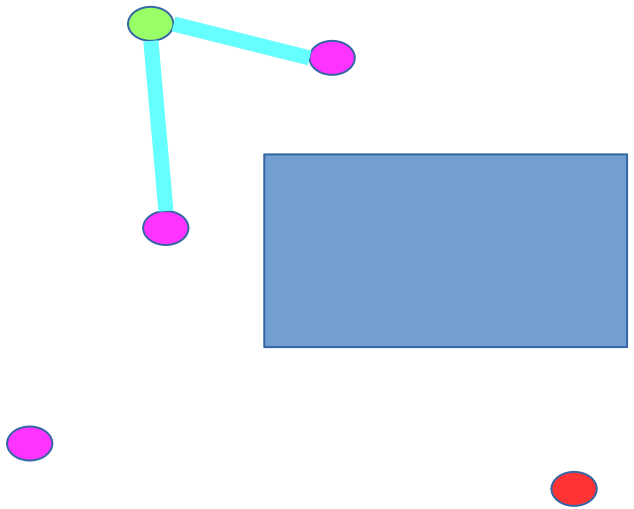
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



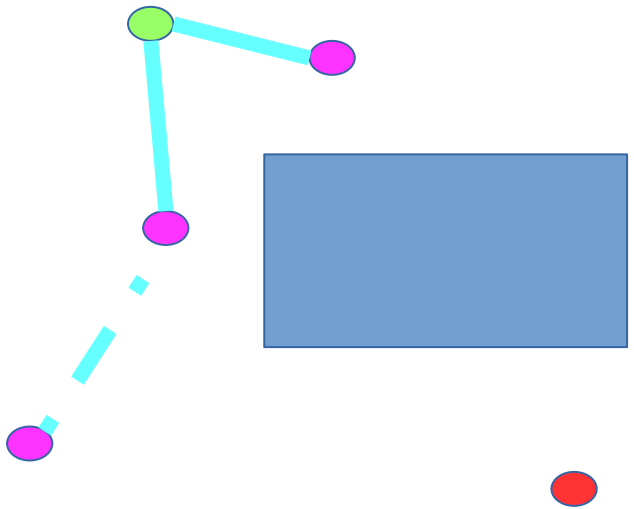
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



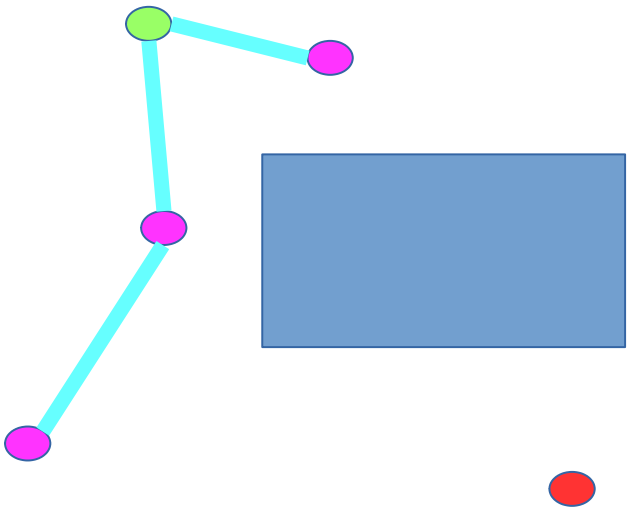
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



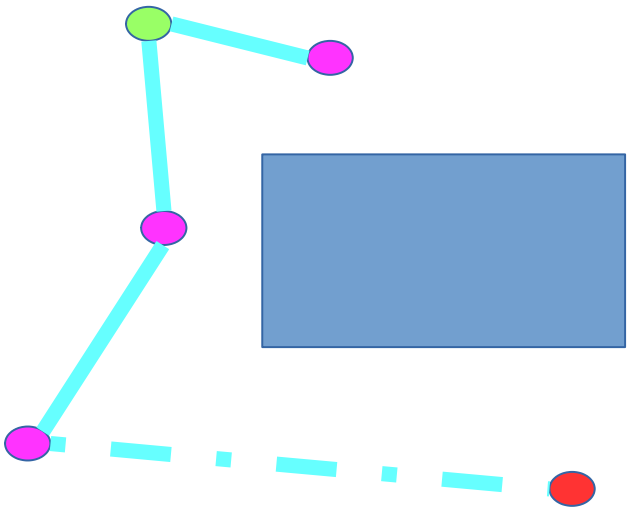
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



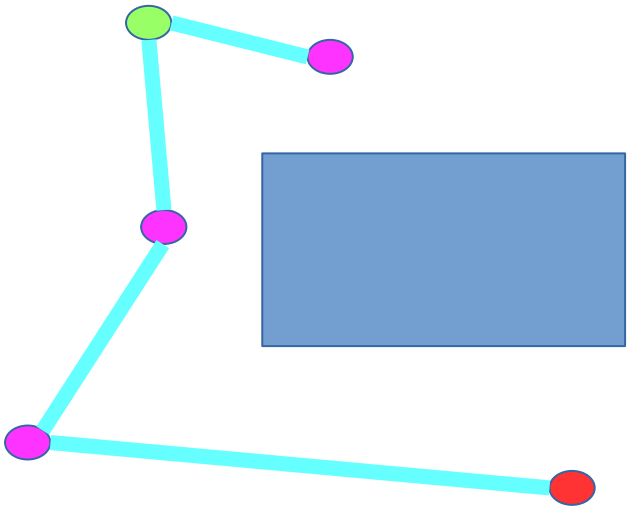
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



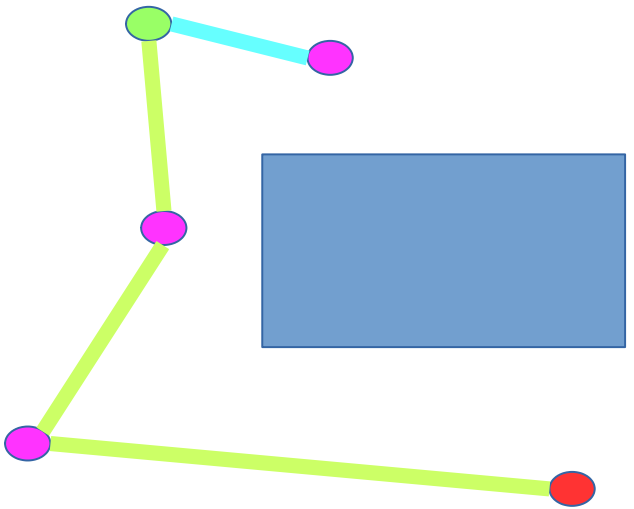
Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado

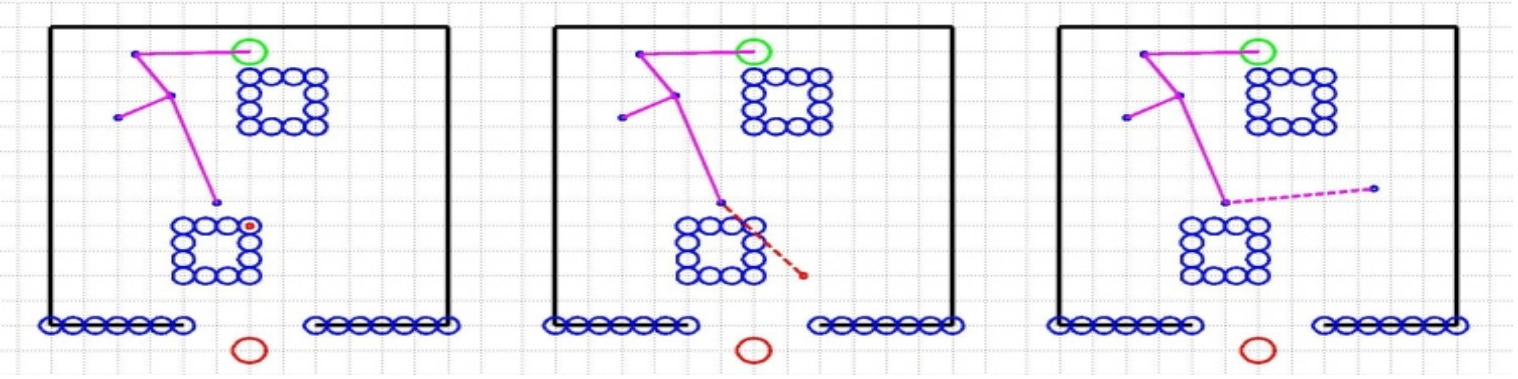


Rapidly-exploring Random Trees (RRT)

- Algortimo:
  1. Iniciar con el número de iteraciones igual a 1
  2. Crear un árbol con nodo inicial igual a la posición inicial en el espacio de configuraciones
  3. Si el número de iteraciones es mayor a N reportar fallo y salir
  4. Incrementar el número de iteraciones
  5. Seleccionar un punto **p** de manera aleatoria en el espacio de configuraciones
  6. Evaluar si **p** colisiona con el espacio de obstáculos
  7. Si colisiona volver a 3
  8. Si no, entonces crear una línea entre **p** y aquel nodo en el árbol cuya distancia sea la menor
  9. Evaluar si la línea colisiona con el espacio de obstáculos
  10. Si colisiona volver a 3
  11. Si no, entonces agregar **p** al árbol donde el padre será el nodo con el que se generó la menor distancia
  12. Calcular la distancia entre el último nodo agregado y la meta
  13. Si la distancia es menor que D entonces crear una línea entre el útimo nodo y la meta
  14. Evaluar si la línea colisiona con el epacio de obstáculos
  15. Si colisiona volver a 3
  16. Si no entonces devuelve el camino encontrado



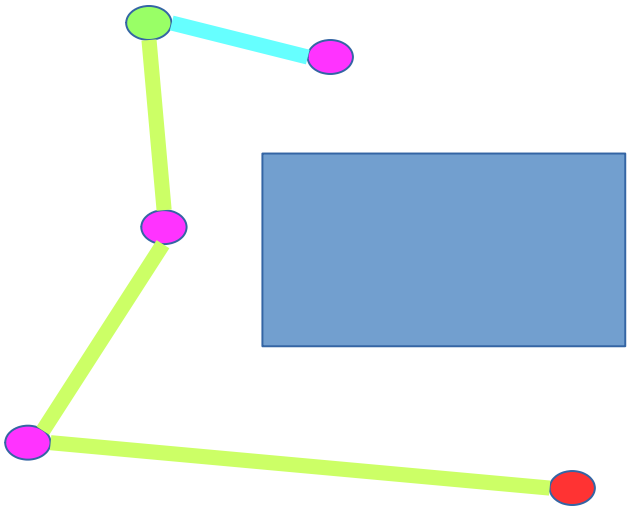
RRT Demo



Referencia y Código Fuente: http://coecsl.ece.illinois.edu/ge423/spring13/RickRekoskeAvoid/rrt.html

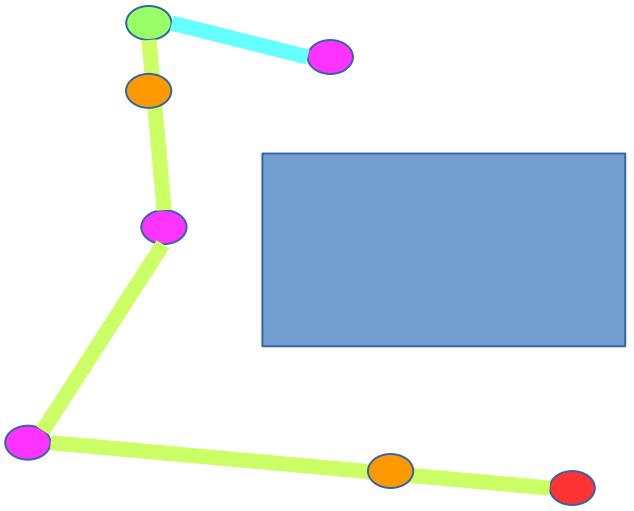
RRT

- **Suavizado de la trayectoria**
  - **IDEA**: elegir puntos aleatorios entre los segmentos del camino encontrado y validar si colisionan con el espacio de obstáculos



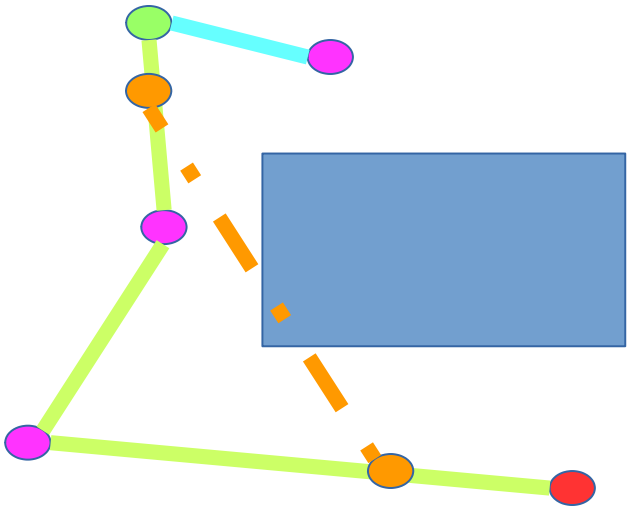
RRT

- **Suavizado de la trayectoria**
  - **IDEA**: elegir puntos aleatorios entre los segmentos del camino encontrado y validar si colisionan con el espacio de obstáculos



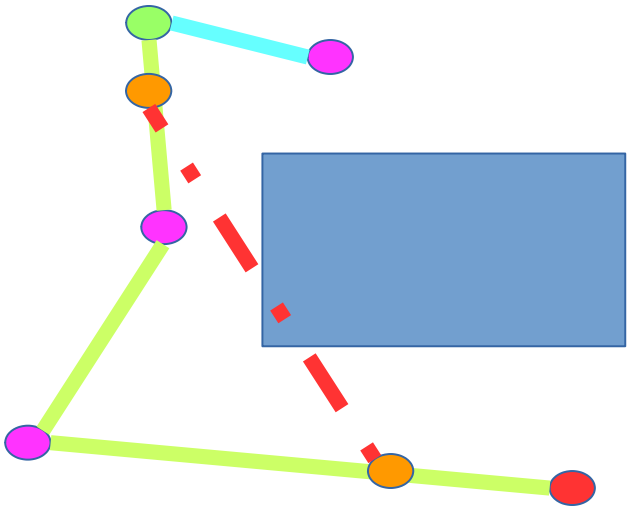
RRT

- **Suavizado de la trayectoria**
  - **IDEA**: elegir puntos aleatorios entre los segmentos del camino encontrado y validar si colisionan con el espacio de obstáculos



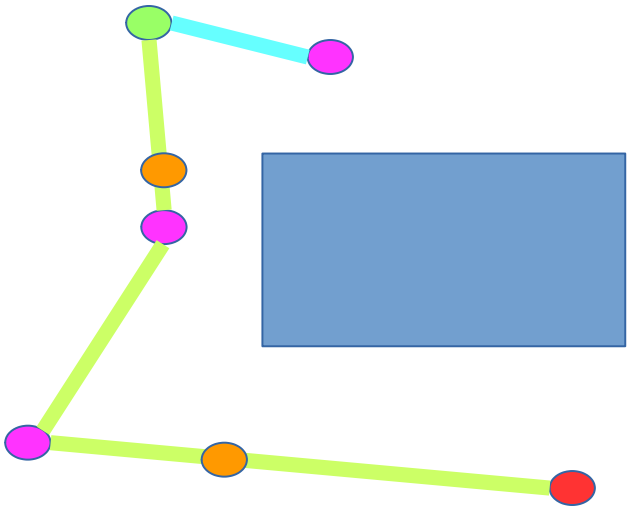
RRT

- **Suavizado de la trayectoria**
  - **IDEA**: elegir puntos aleatorios entre los segmentos del camino encontrado y validar si colisionan con el espacio de obstáculos



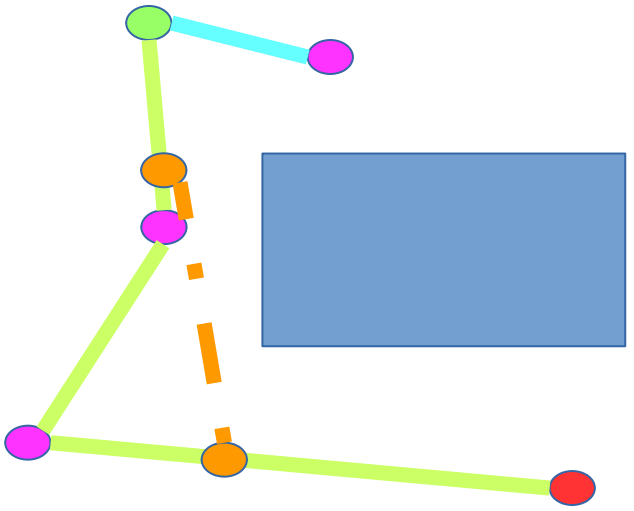
RRT

- **Suavizado de la trayectoria**
  - **IDEA**: elegir puntos aleatorios entre los segmentos del camino encontrado y validar si colisionan con el espacio de obstáculos



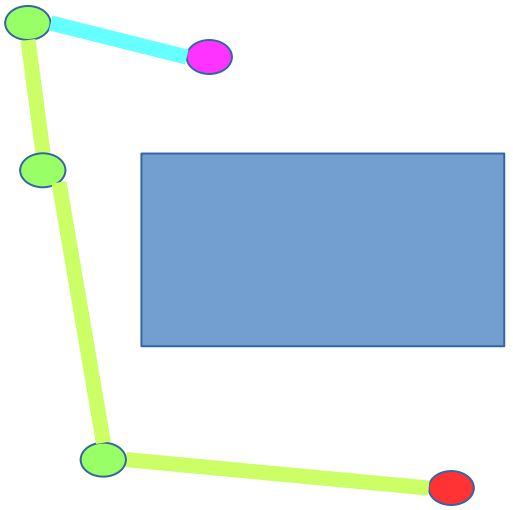
RRT

- **Suavizado de la trayectoria**
  - **IDEA**: elegir puntos aleatorios entre los segmentos del camino encontrado y validar si colisionan con el espacio de obstáculos



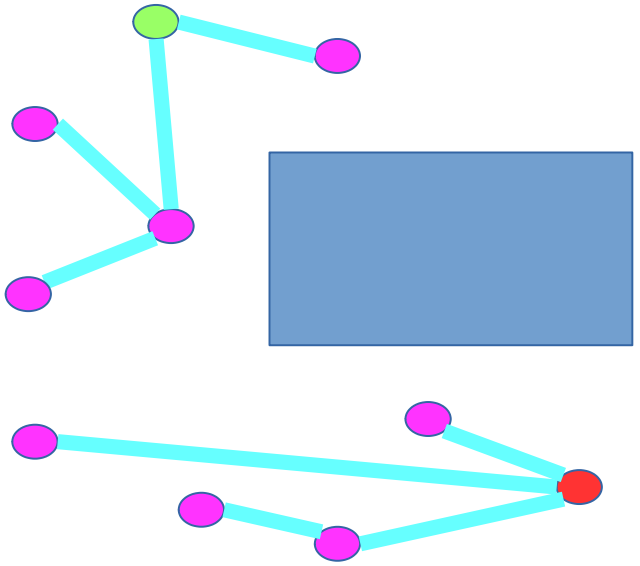
RRT

- **Suavizado de la trayectoria**
  - **IDEA**: elegir puntos aleatorios entre los segmentos del camino encontrado y validar si colisionan con el espacio de obstáculos



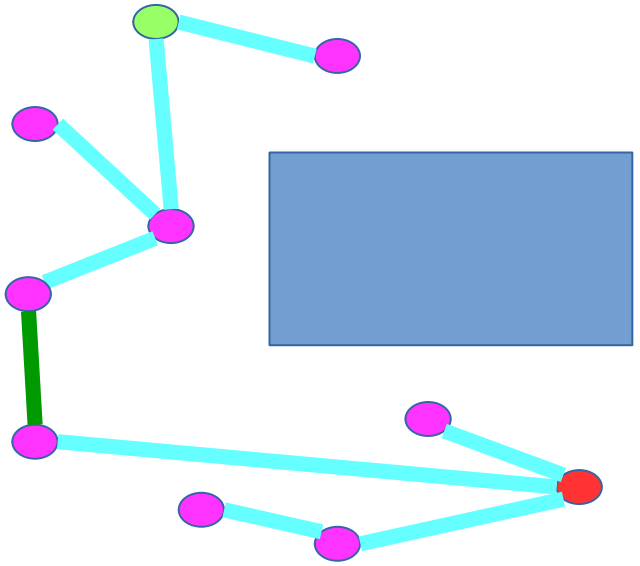
RRT con dos árboles

- **IDEA:** crecer árboles que partan desde punto inicial y desde la meta y para cuando se conecten.



RRT con dos árboles

- **IDEA:** crecer árboles que partan desde punto inicial y desde la meta y para cuando se conecten.



Comentarios Finales Acerca de los RRT

- Ventajas
  - Ofrece un balance entre un algoritmo glotón y uno de búsqueda exahustiva.
  - Converge a la distribución de muestreo en el límite.
  - Implementación fácil y sencilla.
  - Pueden incorporar fácilmente restricciones dinámicas, por ejemplo: dinámica de robotos no-holonómicos.
- Desventajas
  - No garantiza devolver el camino más eficiente.
  - La eficacia del algoritmo decae en espacios de configuración donde el espacio libre es estrecho.

Referencias

1. Hopcroft, J. E., Ullman, J. D., & Aho, A. V. (1983). Data structures and algorithms (Vol. 175). Boston, MA, USA:: Addison-wesley.
2. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), 100-107.
3. LaValle, Steven M. (2006). Planning algorithms. Cambridge university press.
4. Correl, Nikolaus (2014). Introduction to Autonomous Robots, 1st edition. Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.
5. College of Engineering Control Systems Laboratory. University of Illinois at Urbana-Champaign. http://coecsl.ece.illinois.edu/ge423/spring13/RickRekoskeAvoid/rrt.html