Online Path Planning for Autonomous Underwater Vehicles in Unknown Environments

Juan David Hernandez, Eduard Vidal, Guillem Vallicrosa, Enric Galceran, and Marc Carreras

ICRA 2015

Presented by Youngki Kwon

KAIST

Results

- Geometric Planner reaches the nail but ignores the required dynamics (reaches the sides)

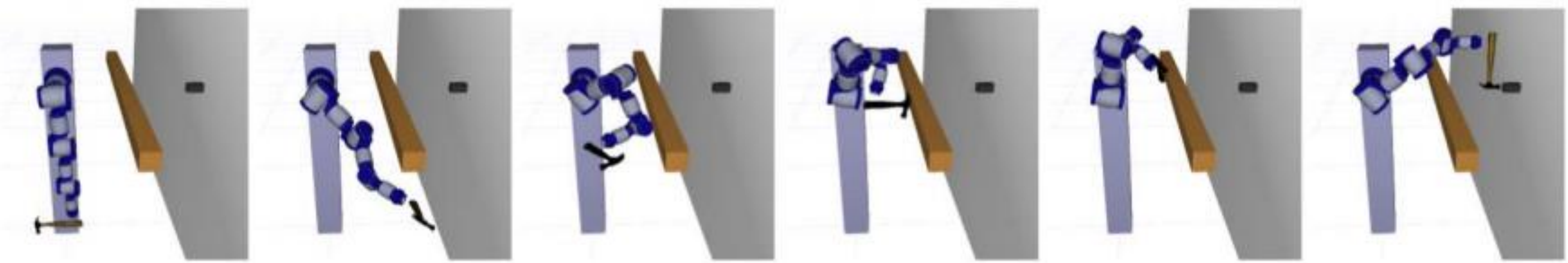


Fig. 6. The DIMT-RRT planner hits the nail at the desired velocity

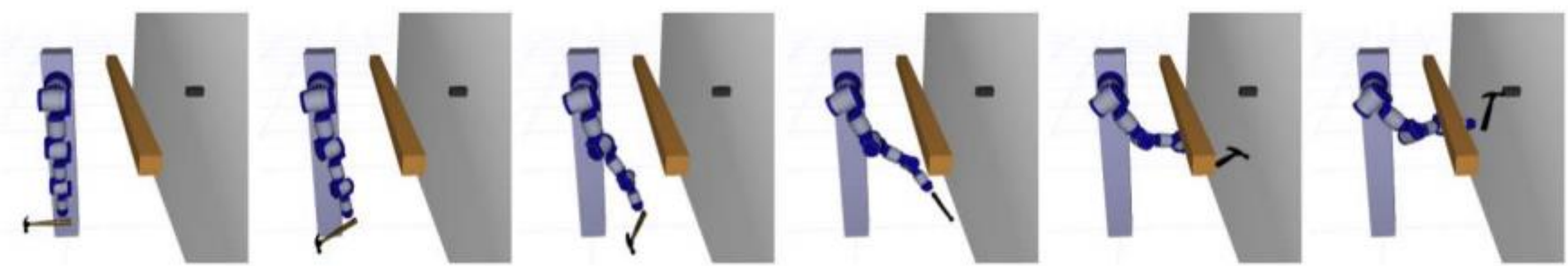


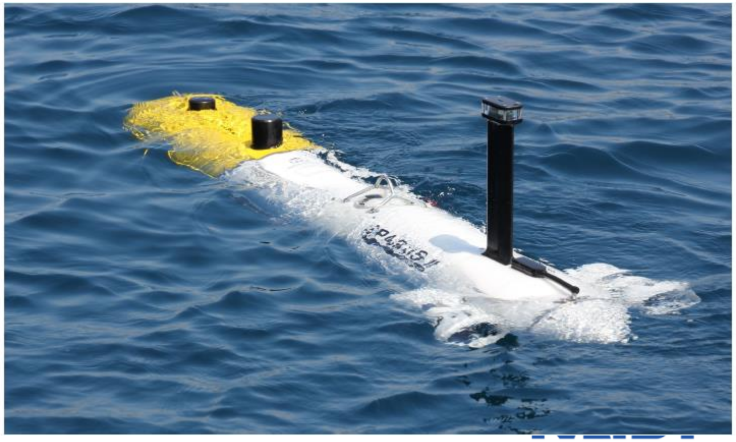
Fig. 7. The geometric RRT planner reaches the nail but not at the desired velocity

Contents

INTRODUCTION

Introduction

- AUVs must operate in unknown, cluttered and dynamic environments
  - AUVs are more exposed to collisions
  - Drift effects on the position estimated by navigation system
- Online capabilities of path planner make AUVs overcome global position inaccuracy



MAIN CONTRIBUTION

Main Contribution

- Combination of ideas from lazy collision evaluation and anytime path planning algorithm
  - For online path planning
  - incrementally built octree-based representation of the environment

- The experimental evaluation of the resulting planning framework using the SPARUS-II AUV in a real-world setting

BACKGROUND

Anytime Path Planning

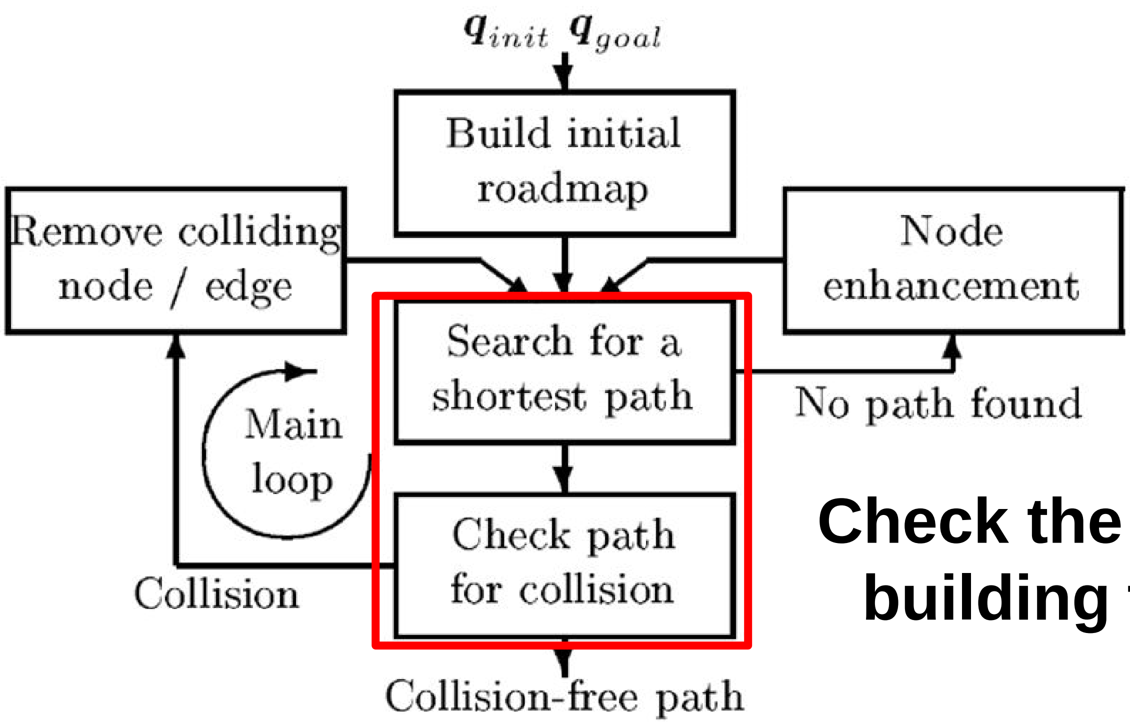
- Overall approach
  - Quickly find a solution that is feasible, but not necessarily optimal
  - Exploit execution time to incrementally improve towards optimal solution

- Desired properties
  - Form of completeness guarantees
  - Asymptotic optimality given more computation time

Lazy Collision Evaluation

- Only check for collisions when we have to

Lazy PRM



Check the collision after building the roadmap

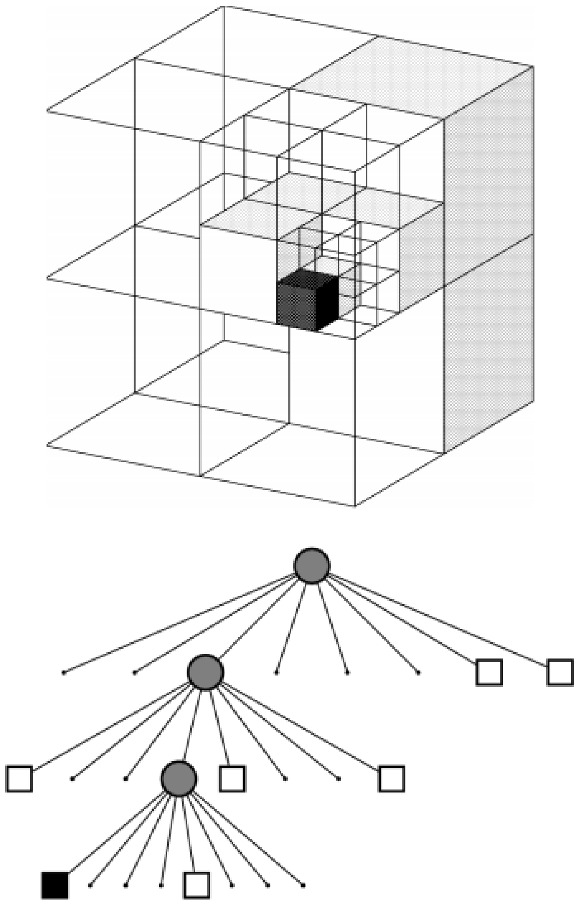
Octree-based Map

Map Representations

Octrees

- Tree-based data structure
- Recursive subdivision of space into octants
- Volumes allocated as needed

➡ Smart 3D grid



Techniques for 3d mapping - Wolfram burgard's slide

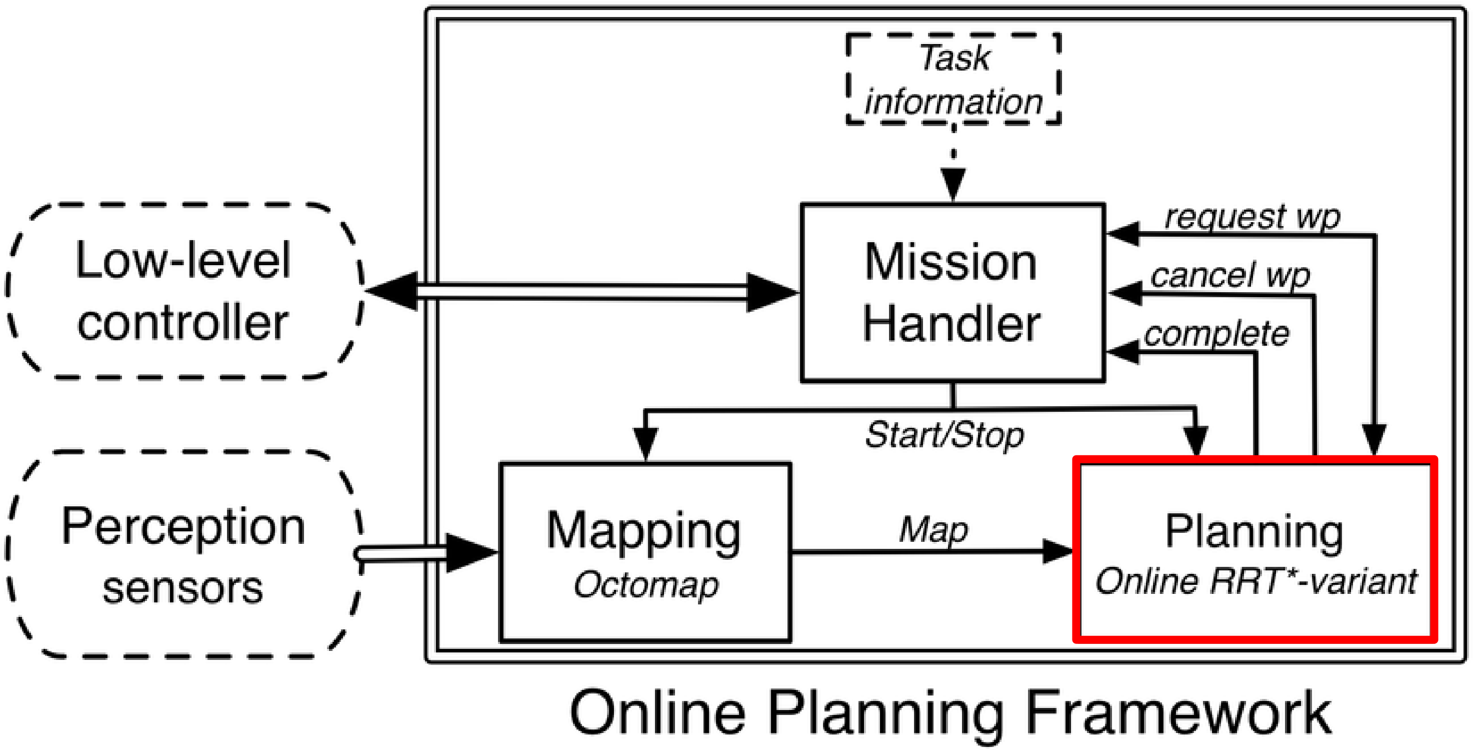
Octree-based Map

- **OctoMap Framework**
  - Based on octrees
  - Probabilistic representation of occupancy including unknown
  - Supports multi-resolution map queries
  - Memory efficient
  - Compact map files
  - Optimized for runtime
  - Open source implementation as C++ library available at http://octomap.sf.net

MAIN APPROACH

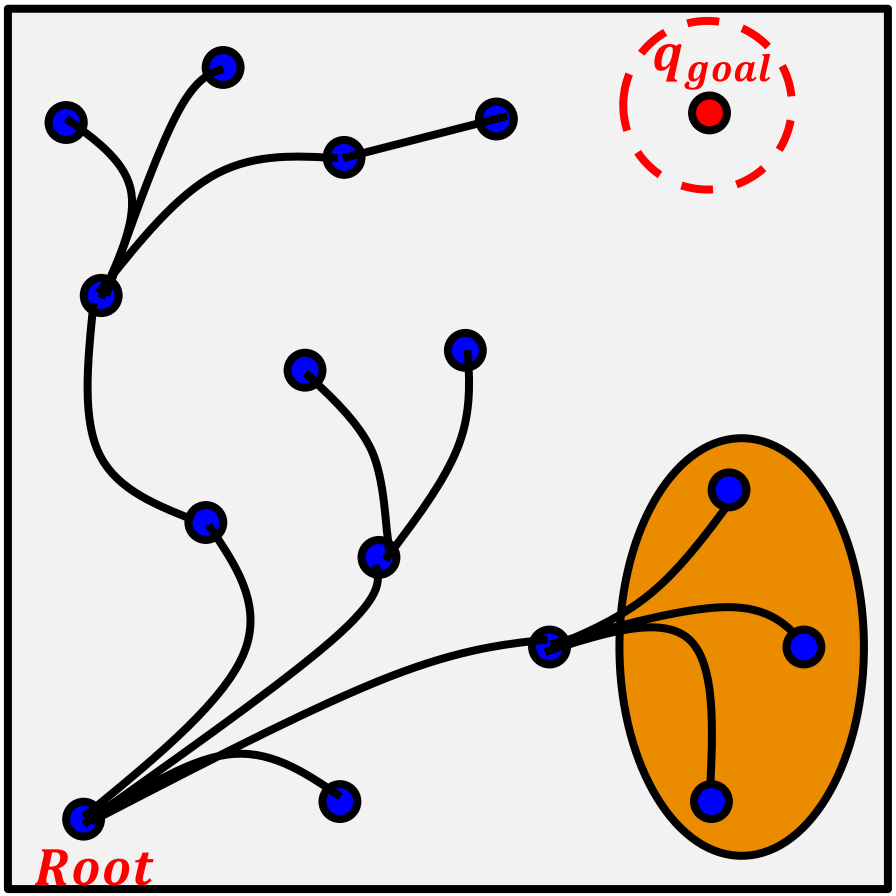
Framework

- To solve start-to-goal queries for an AUV in an unknown environment
- Framework for path planning online for AUVs



Anytime Approach

- Under the time constraint, current T is left image
- Assume T is valid



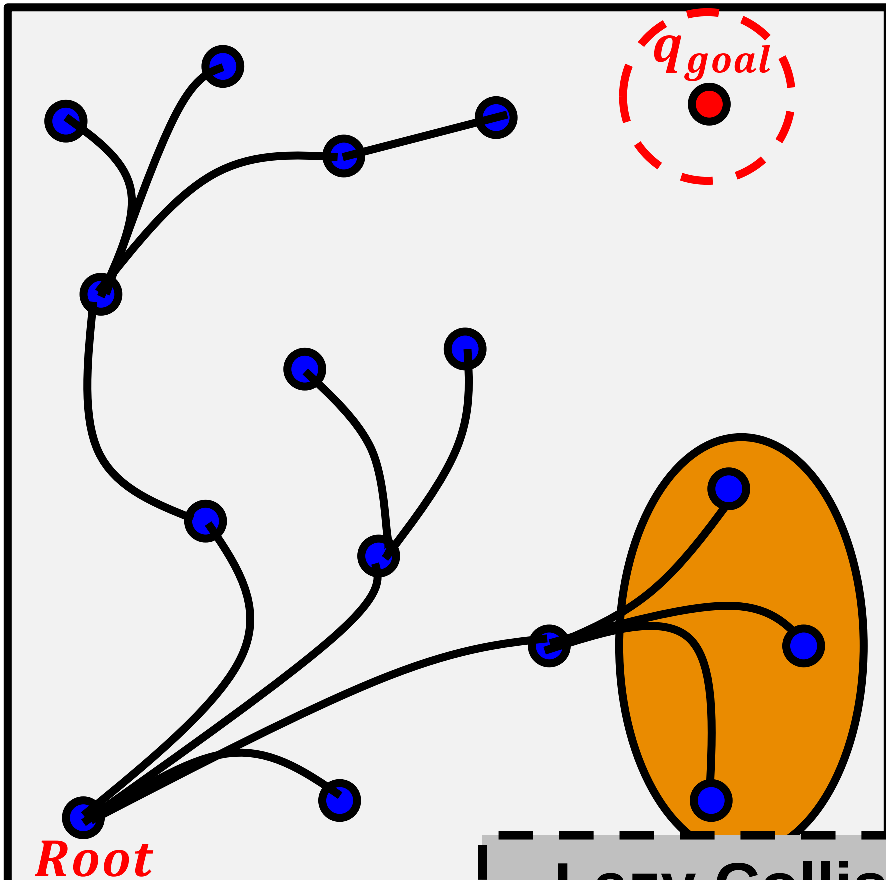
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

begin
    $\boxed{\texttt{updateTree()}}$ Time limit
    while not $\boxed{stop\_condition}$ **do**
        $q_{rand} \leftarrow$ `sampleConf()`
        $result, q_{new} \leftarrow$ `extendRRT(`T, q_{rand}`)`
        if $result \neq TRAPPED$ **then**
            if `dist(`q_{new}, q_{goal}`)` $< \epsilon_{goal}$ **then**
                `addSolution(`q_{new}`)`
                $solution\_found \leftarrow true$
    if $solution\_found$ **and** wp_req **then**
        $result\_path \leftarrow$ `getBestSolution()`
        $new\_root \leftarrow result\_path[1]$
        `sendWaypoint(`new_root`)`
        `pruneTree(`new_root`)`
end

Anytime Approach

- Traverse T, check the collision, cut the subtree in collision



Algorithm 1: buildRRT(T)

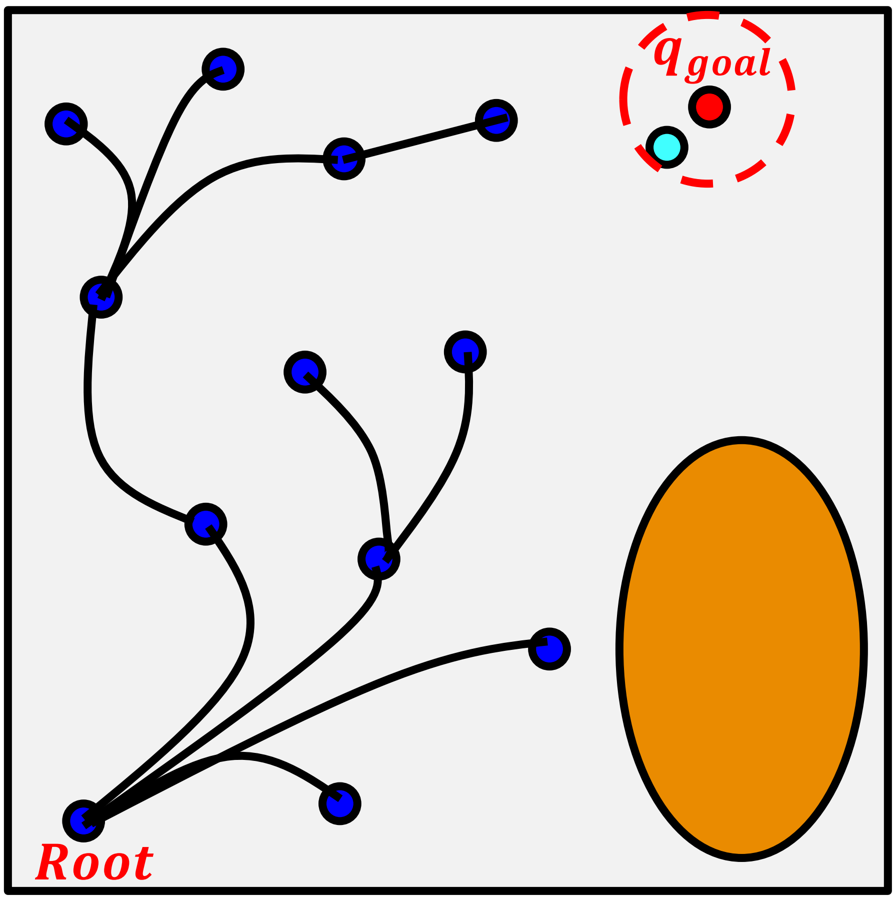
Input:
T: configurations tree (RRT).

begin
  $\boxed{\texttt{updateTree()}}$
  while not *stop_condition* **do**
    $q_{rand} \leftarrow$`sampleConf()`
    $result, q_{new} \leftarrow$`extendRRT`$(T, q_{rand})$
    if $result \neq TRAPPED$ **then**
      if `dist`$(q_{new}, q_{goal}) < \epsilon_{goal}$ **then**
        `addSolution`(q_{new})
        $solution\_found \leftarrow true$
  if *solution_found* **and** *wp_req* **then**
    $result\_path \leftarrow$`getBestSolution()`
    $new\_root \leftarrow result\_path[1]$
    `sendWaypoint`(new_root)
    ew_root)

Lazy Collision Evaluation

Anytime Approach

- Sample configuration



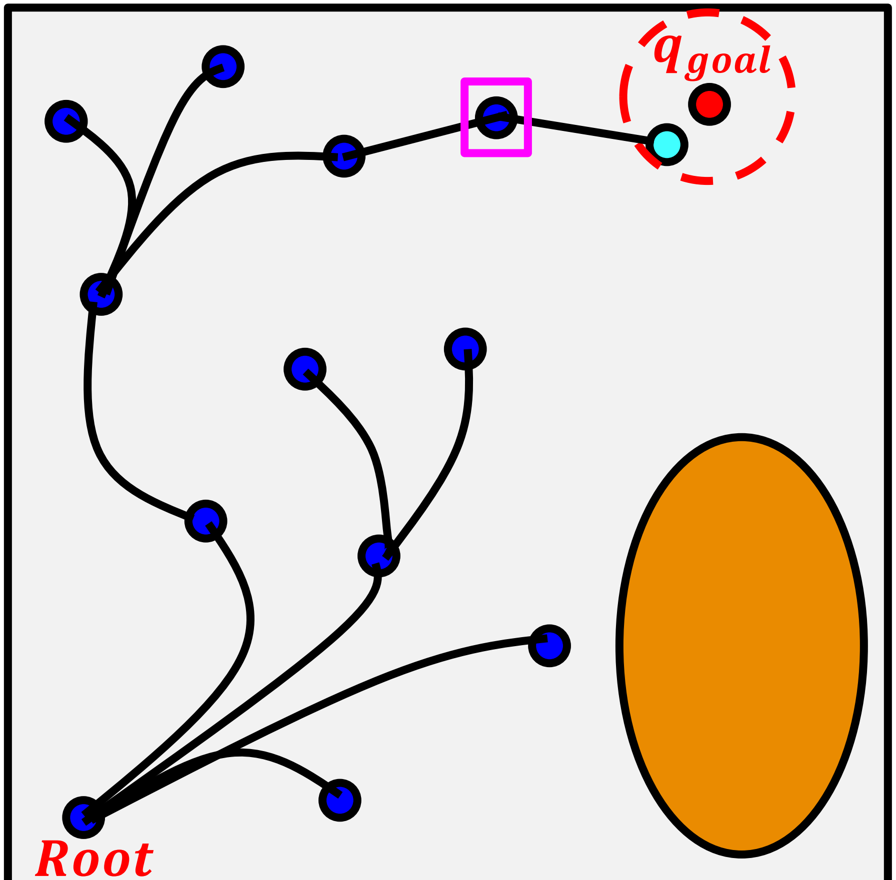
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

begin
  `updateTree()`
  while not *stop_condition* **do**
    $\boxed{q_{rand} \leftarrow \texttt{sampleConf()}}$
    result, $q_{new} \leftarrow$ `extendRRT`(T, q_{rand})
    if *result* $\neq$ *TRAPPED* **then**
      if `dist`$(q_{new}, q_{goal}) < \epsilon_{goal}$ **then**
        `addSolution`(q_{new})
        solution_found $\leftarrow$ *true*
  if *solution_found* **and** *wp_req* **then**
    result_path $\leftarrow$ `getBestSolution()`
    new_root $\leftarrow$ *result_path*[1]
    `sendWaypoint`(*new_root*)
    `pruneTree`(*new_root*)
end

Anytime Approach

- Extension in RRT*



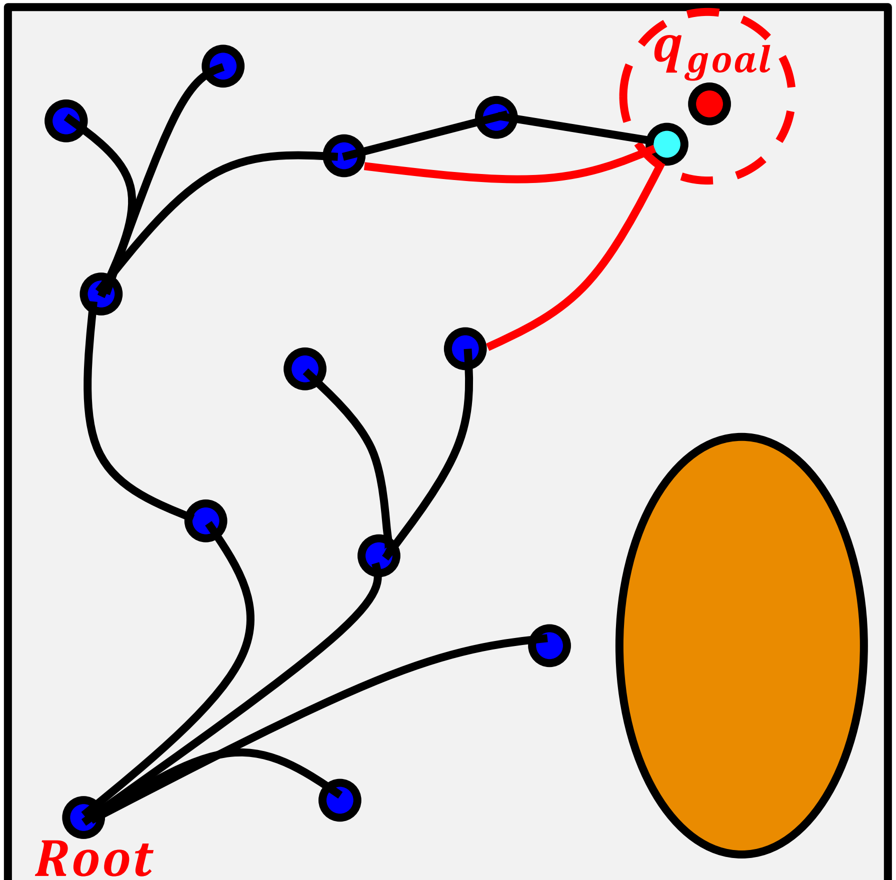
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

begin
  `updateTree()`
  while not *stop_condition* **do**
    q_{rand} ← `sampleConf()`
    $\boxed{result, q_{new} \leftarrow \texttt{extendRRT}(T, q_{rand})}$
    if $result \neq TRAPPED$ **then**
      if `dist`$(q_{new}, q_{goal}) < \epsilon_{goal}$ **then**
        `addSolution`(q_{new})
        $solution\_found \leftarrow true$
  if *solution_found* **and** *wp_req* **then**
    result_path ← `getBestSolution()`
    $new\_root \leftarrow result\_path[1]$
    `sendWaypoint`(new_root)
    `pruneTree`(new_root)
end

Anytime Approach

- Extension in RRT*



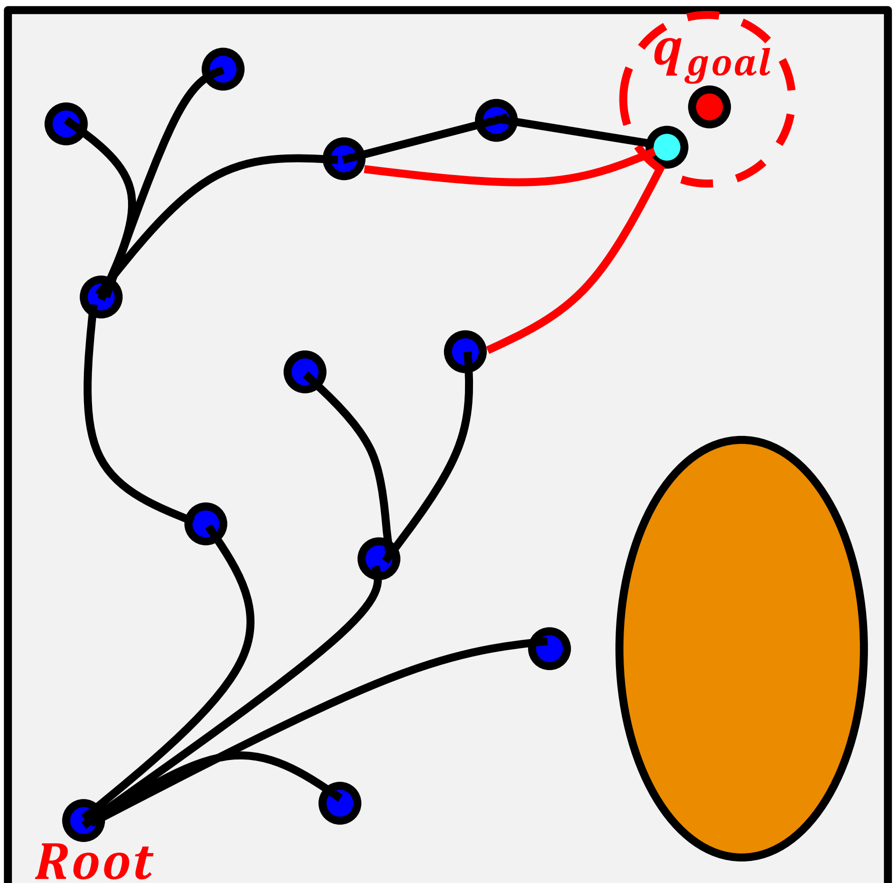
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

begin
  `updateTree()`
  while not *stop_condition* **do**
    q_{rand} ← `sampleConf()`
    $\boxed{result, q_{new} \leftarrow \texttt{extendRRT}(T, q_{rand})}$
    if $result \neq TRAPPED$ **then**
      if `dist`$(q_{new}, q_{goal}) < \epsilon_{goal}$ **then**
        `addSolution`(q_{new})
        solution_found ← *true*
  if *solution_found* **and** *wp_req* **then**
    result_path ← `getBestSolution()`
    new_root ← *result_path*[1]
    `sendWaypoint`(*new_root*)
    `pruneTree`(*new_root*)
end

Anytime Approach

- **Extension in RRT***



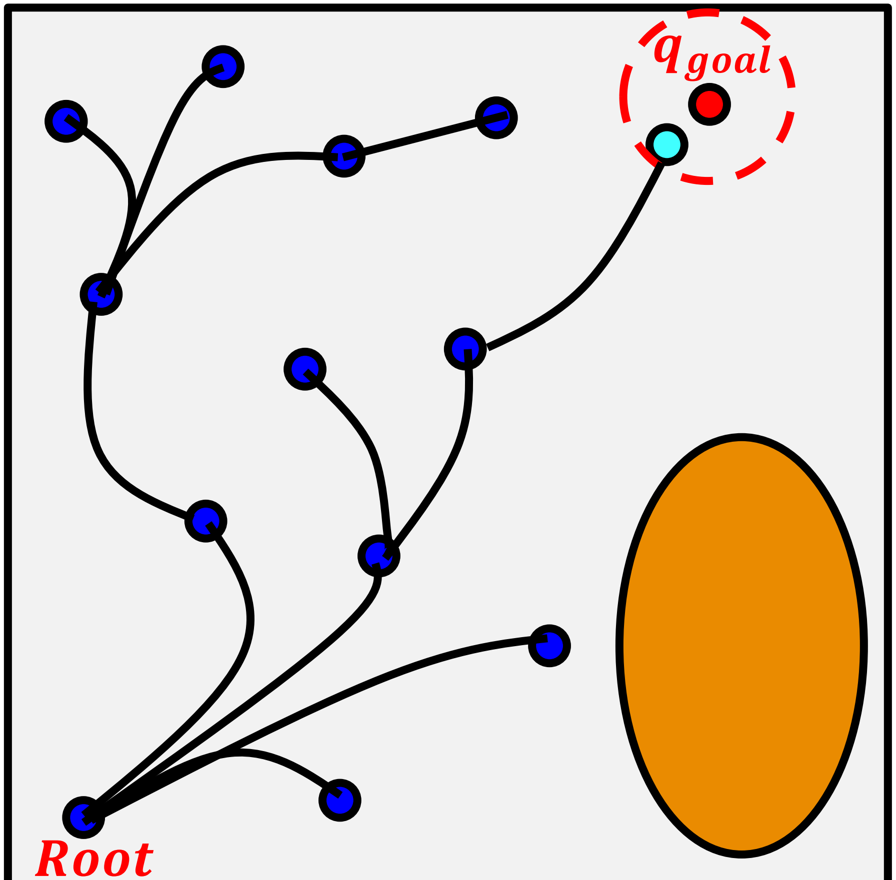
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

begin
  `updateTree()`
  while not *stop_condition* **do**
    q_{rand} ←`sampleConf()`
    $\boxed{result, q_{new} \leftarrow \texttt{extendRRT}(T, q_{rand})}$
    if $result \neq TRAPPED$ **then**
      if `dist`$(q_{new}, q_{goal}) < \epsilon_{goal}$ **then**
        `addSolution`(q_{new})
        solution_found ← *true*
  if *solution_found* **and** *wp_req* **then**
    result_path ←`getBestSolution()`
    new_root ← *result_path*[1]
    `sendWaypoint`(*new_root*)
    `pruneTree`(*new_root*)
end

Anytime Approach

- **Extension in RRT***



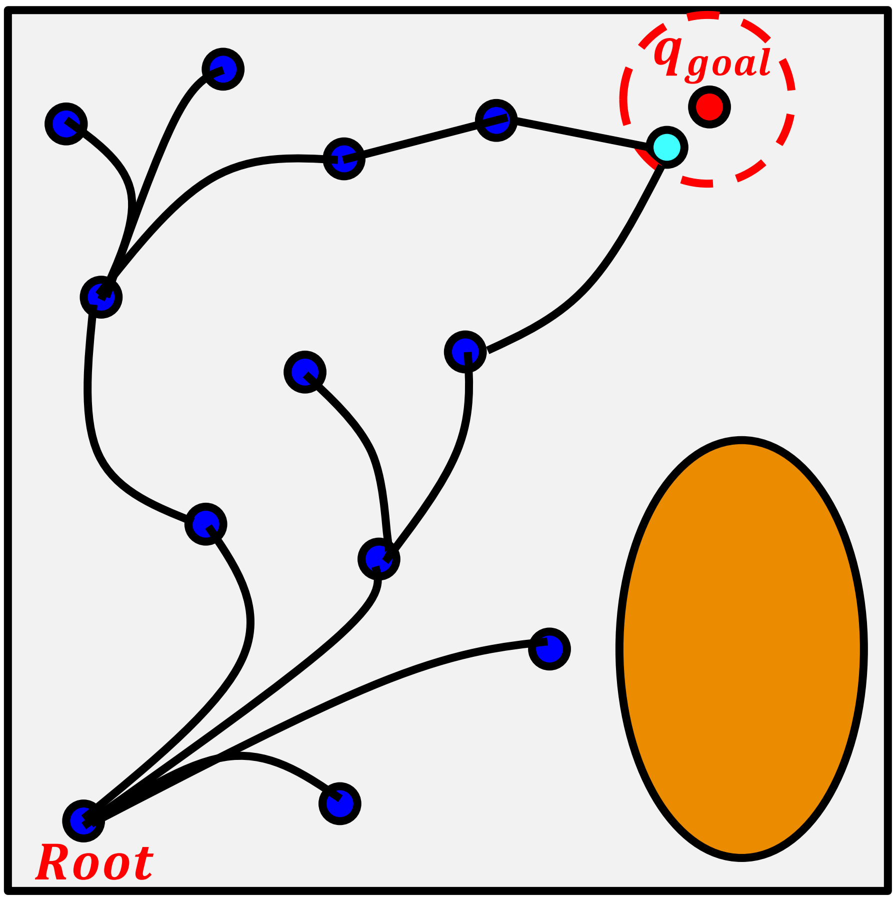
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

begin
    `updateTree()`
    while not *stop_condition* **do**
        $q_{rand} \leftarrow$ `sampleConf()`
        $\boxed{result, q_{new} \leftarrow \texttt{extendRRT}(T, q_{rand})}$
        if *result* $\neq$ *TRAPPED* **then**
            if `dist`$(q_{new}, q_{goal}) < \epsilon_{goal}$ **then**
                `addSolution`(q_{new})
                solution_found $\leftarrow$ *true*
    if *solution_found* **and** *wp_req* **then**
        result_path $\leftarrow$ `getBestSolution()`
        new_root $\leftarrow$ *result_path*[1]
        `sendWaypoint`(*new_root*)
        `pruneTree`(*new_root*)
end

Anytime Approach

- Extension in RRT*



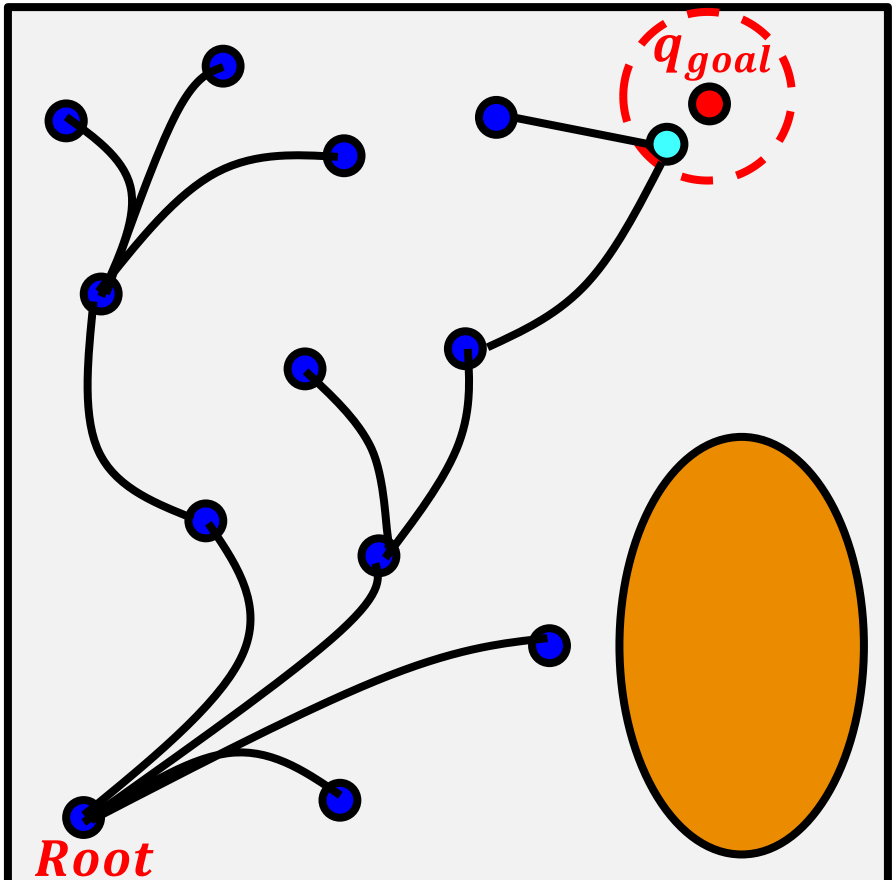
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

begin
    updateTree()
    while not *stop_condition* **do**
        q_{rand} ← sampleConf()
        $\boxed{result, q_{new} \leftarrow \texttt{extendRRT}(T, q_{rand})}$
        if $result \neq TRAPPED$ **then**
            if dist(q_{new}, q_{goal}) $< \epsilon_{goal}$ **then**
                addSolution(q_{new})
                $solution\_found \leftarrow true$
    if *solution_found* **and** *wp_req* **then**
        $result\_path$ ← getBestSolution()
        $new\_root \leftarrow result\_path[1]$
        sendWaypoint(new_root)
        pruneTree(new_root)
end

Anytime Approach

- Find solution



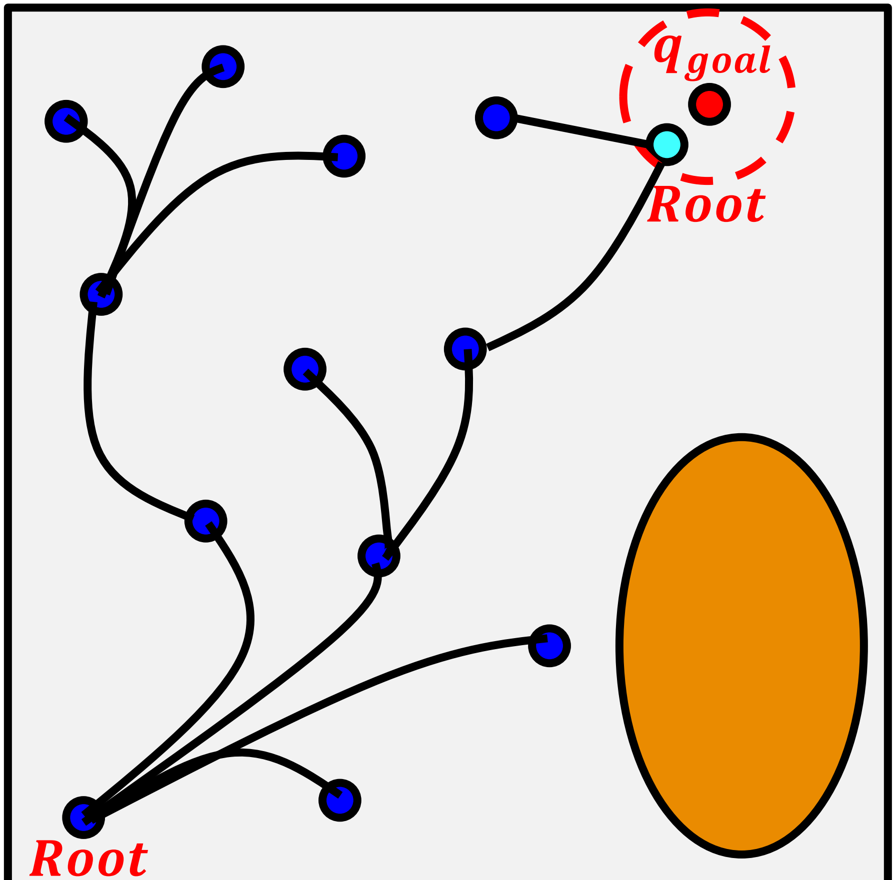
Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

```
begin
    updateTree()
    while not stop_condition do
        q_rand ←sampleConf()
        result, q_new ←extendRRT(T, q_rand)
        if result ≠ TRAPPED then
            if dist(q_new, q_goal) < ε_goal then
                addSolution(q_new)
                solution_found ← true
    if solution_found and wp_req then
        result_path ←getBestSolution()
        new_root ← result_path[1]
        sendWaypoint(new_root)
        pruneTree(new_root)
end
```

Anytime Approach

- **After Satisfying stop condition**



Algorithm 1: buildRRT(T)

Input:
T: configurations tree (RRT).

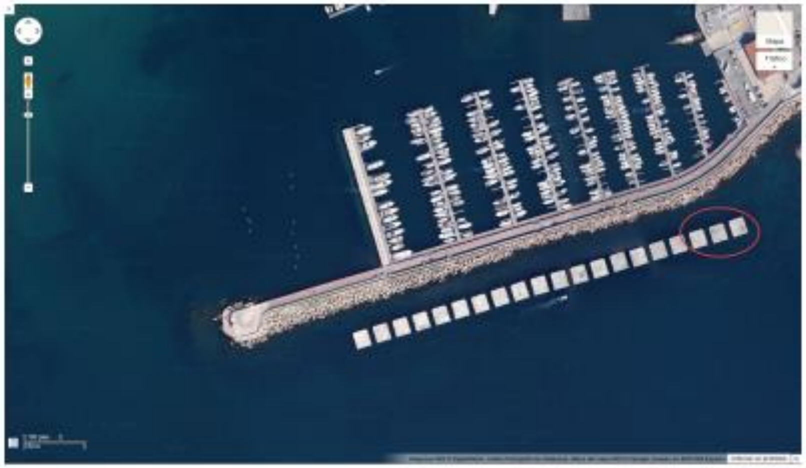
begin
    updateTree()
    while not *stop_condition* **do**
        q_{rand} ←sampleConf()
        $result, q_{new}$ ←extendRRT(T, q_{rand})
        if $result \neq TRAPPED$ **then**
            if dist(q_{new}, q_{goal}) $< \epsilon_{goal}$ **then**
                addSolution(q_{new})
                $solution\_found \leftarrow true$
    if *solution_found* **and** *wp_req* **then**
        $result\_path$ ←getBestSolution()
        $new\_root \leftarrow result\_path[1]$
        sendWaypoint(new_root)
        pruneTree(new_root)
end

EXPERIMENT & RESULT

Experiment & Result

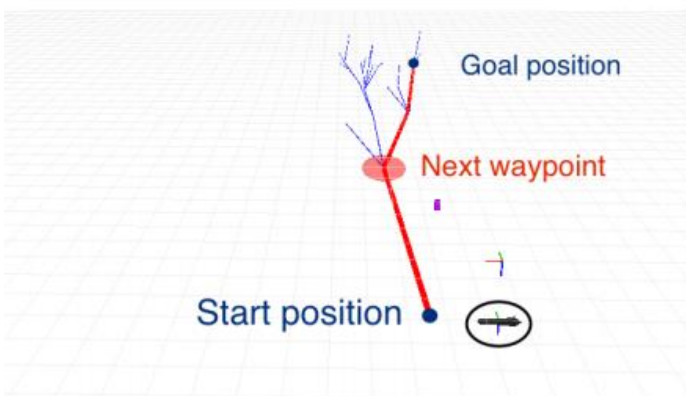
- AUV use
  - Pressure sensor, Doppler velocity log, IMU measurement, GPS
  - Five Echosounders

- Experiments in both simulation and real world

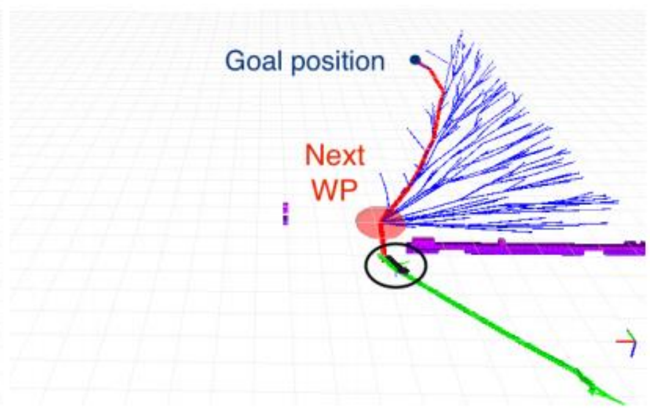


Experiment & Result

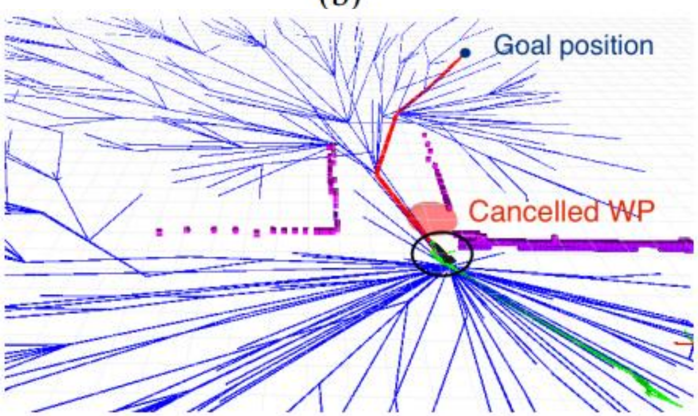
- **Experiment in simulation**



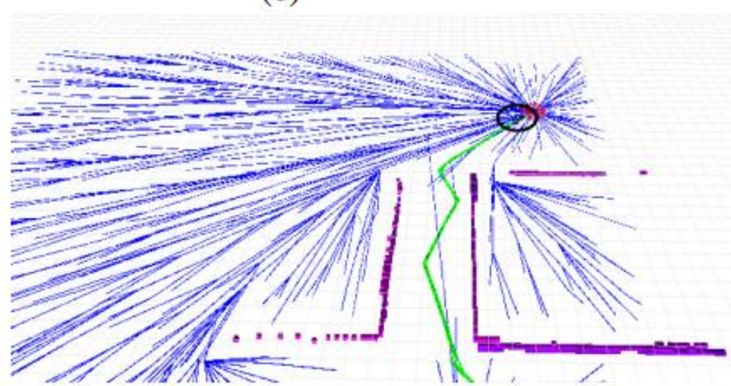
(b)



(c)



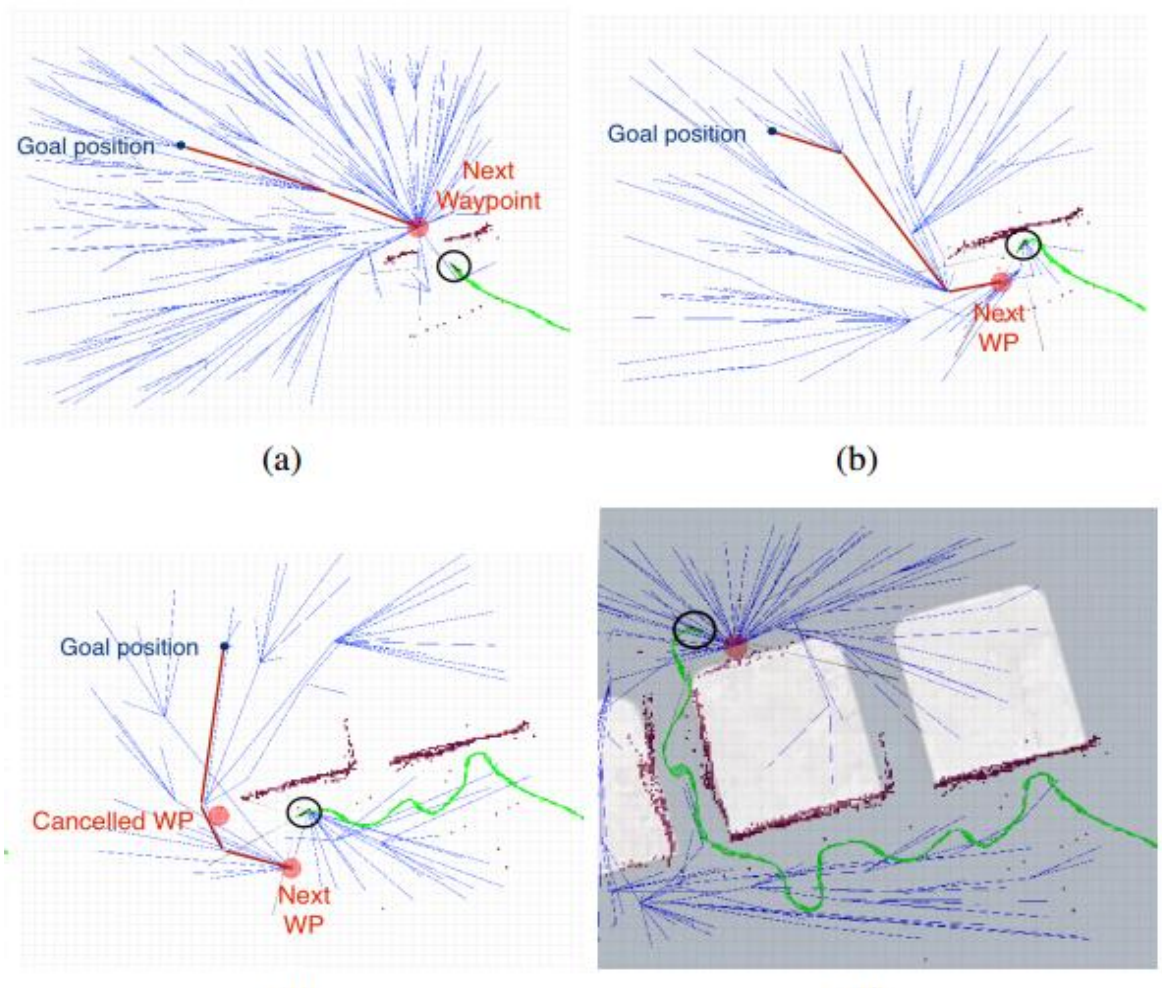
(d)



(e)

Experiment & Result

- **Experiment in real-world**



KAIST

SUMMARY

Summary

- Online capabilities of path planner make AUVs overcome global position inaccuracy
- Combination of ideas from lazy collision evaluation and anytime path planning algorithm

KAIST

Q & A