26th International Conference on Computer Systems and Technologies
CompSysTech'25, 27-28 June 2025, University of Ruse, Bulgaria

Motion Planning with Obstacle Avoidance Within Complex Environment for Redundant Manipulators

Kaloyan Yovchev

Faculty of Mathematics and Informatics,
Sofia University, Bulgaria

Institute of Robotics,
Bulgarian Academy of Sciences, Bulgaria

Motivation

- Real-world environments are dynamic and partially unknown
- Robots often face unexpected obstacles
- Traditional planners assume full environment knowledge

Research Goal

- Compare different algorithms for planning in unknown environments
- Application to a planar redundant manipulator
- Evaluate based on:
  - Planning time
  - Path smoothness
  - Collision avoidance

Planning in Unknown Environments

- Challenges:
  - No full map beforehand
  - Must build tree incrementally
  - Real-time obstacle detection
  - Continuous collision checking

Redundant Manipulator Model

- 3D printed robot with 4 revolute joints
- Link lengths: 150mm, 100mm, 100mm
- Joint limits: $[-\pi/2, \pi/2]$
- End-effector position in 2D plane
- 3 revolute joints used (4th link = 0)
- Kinematics resolved in joint space
- Redundancy: 3 joints to reach 2D target

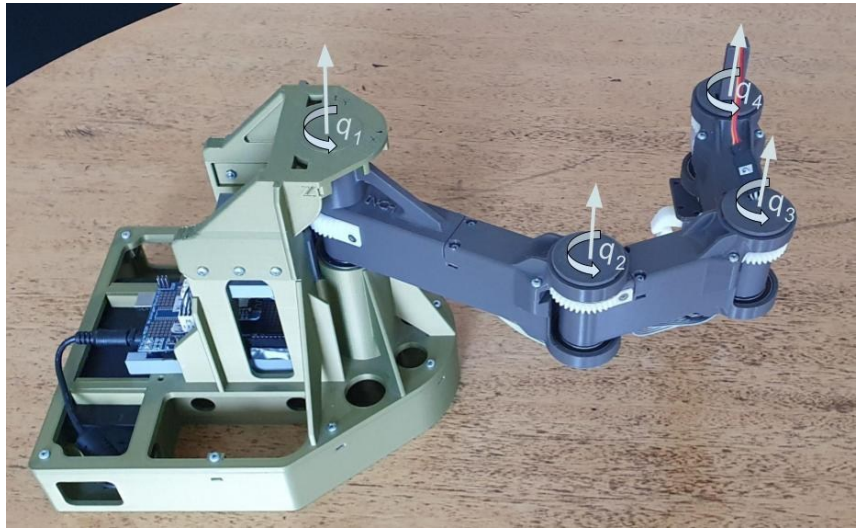


Fig. 1. 3D printed redundant planar redundant robotic manipulator.

Configuration Space

- Workspace Division
- 4 configuration types: LL, LR, RL, RR

$$
\begin{aligned}
RR: &\quad q_1 \leq 0, q_2 \leq 0 \\
RL: &\quad q_1 \leq 0, q_2 \geq 0 \\
LR: &\quad q_1 \geq 0, q_2 \leq 0 \\
LL: &\quad q_1 \geq 0, q_2 \geq 0
\end{aligned}
$$

- Obstacle splits workspace
- Changing configuration might be necessary

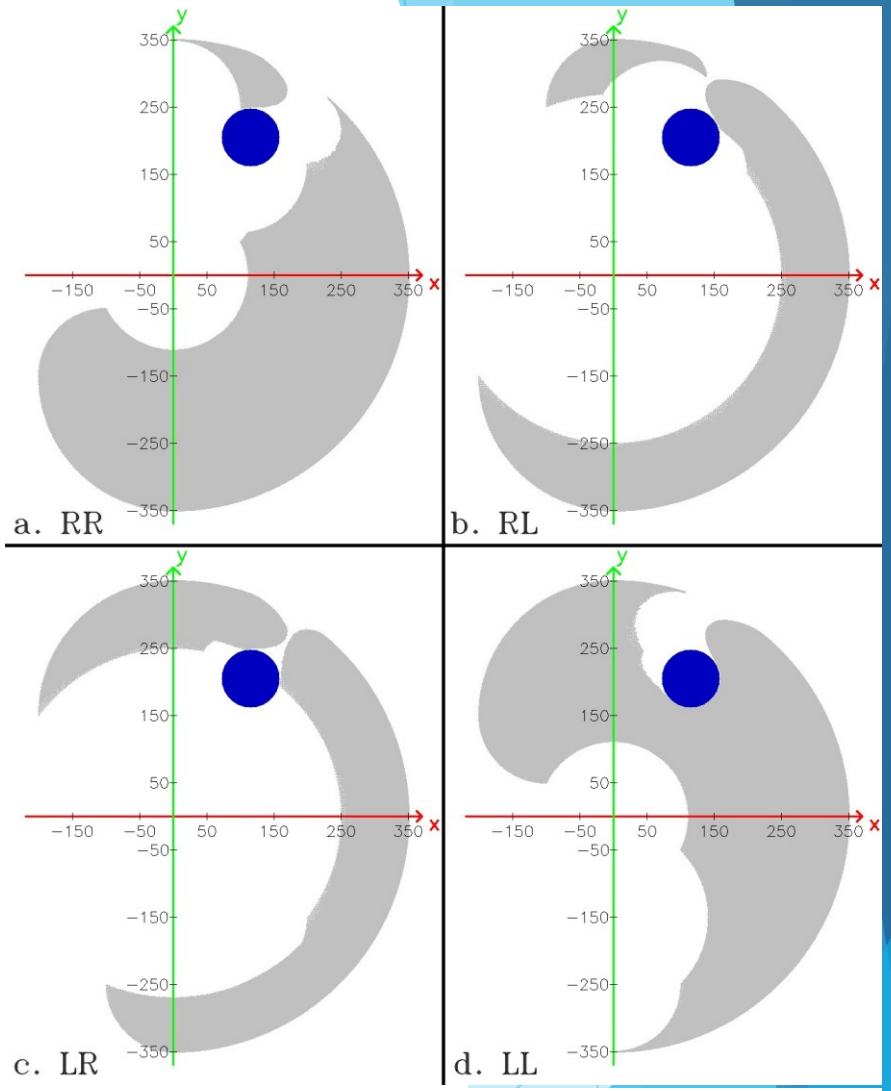


Fig. 2. Separation of the workspace of the robot for the different types of solutions as defined in (1) when there is an obstacle (denoted with a blue circle): a. RR; b. RL; c. LR; d. LL

Motion Planning Algorithms

- **Two categories:**
  - **Classical:**
    - Depth-First Search (DFS),
    - Iterative Deepening Depth-First Search (IDDFS),
    - Breadth-First Search (BFS)
  - **Sampling-based:**
    - Rapidly-exploring Random Trees (RRT),
    - RRT*

How RRT and RRT* Work?

- Random node generation
- Nearest neighbor connection
- Path built incrementally
- RRT*: rewires to optimize path
- Safety threshold δ for fine-tuning and for limiting the number of the edges

RRT and RRT*: Algorithm Flow

- **RRT Steps:**
  - Sample random configuration
  - Find nearest node
  - Generate safe edge
  - Add node if valid
  - Repeat until goal
- **RRT* additions:**
  - Rewire neighbors
  - Choose optimal parent
  - Minimize path cost

RRT Algorithm

```
Function rrt(start, end, delta=0.05):
    Initialize the random number generator
    prev ← empty dictionary (stores parent nodes)
    vertices ← list containing the start configuration
    found ← False
    While not found:
        randomQ ← random configuration
                  from getRandomQ()
        closest ← Find the vertex from vertices which is
                  closest to randomQ with respect to norm2
        newQ ← Adjust the coordinates of randomQ so that it is
               at distance delta from the closest vertex and
               lies on the line containing the closest vertex
               and randomQ
        If checkCollision(solveForward(newQ)):
            Continue to the next iteration
        Store newQ as a child of the closest vertex in prev
        Add newQ to vertices
        If norm2(newQ, end) < delta^2:
            Store end as a child of newQ in prev
            found ← True
    path ← genPath(prev, start, end)
    Return path and prev
```

Fig. 4. Pseudo-code of the RRT algorithm when redundant manipulator is considered.

Experimental Setup

- Start: $(-1, -1, -1)$ radians
- Goal: $(1, 1, 1)$ radians
- Obstacle at (115, 205) mm, safety radius 45 mm

- Evaluate the existence of a connection (edge) between two graph (tree) nodes u and v which are corresponding to configurations (q_1^u, q_2^u, q_3^u) and (q_1^v, q_2^v, q_3^v) is given by the condition where δ is an external parameter of the algorithms:

$$\sqrt{(q_1^u - q_1^v)^2 + (q_2^u - q_2^v)^2 + (q_3^u - q_3^v)^2} \leq \delta.$$

- Tested δ values: 0.15, 0.10, 0.05

Results – Generated Trees

- Generated trees with their respective nodes and edges for IDDFS, BFS, RRT and RRT* are compared
- RRT and RRT* algorithms create more sparce trees than BFS and especially IDDFS
- RRT and RRT* require a lower computation time
- For lower values of δ the graphs contain more nodes and edges.

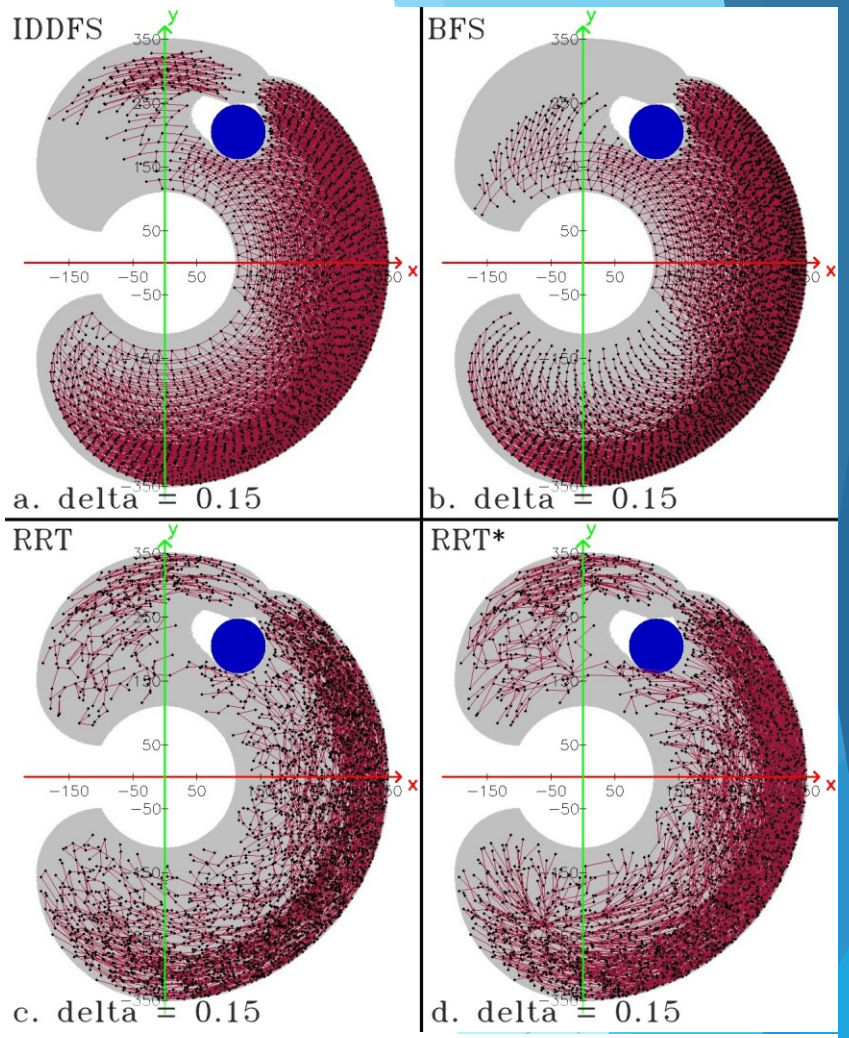
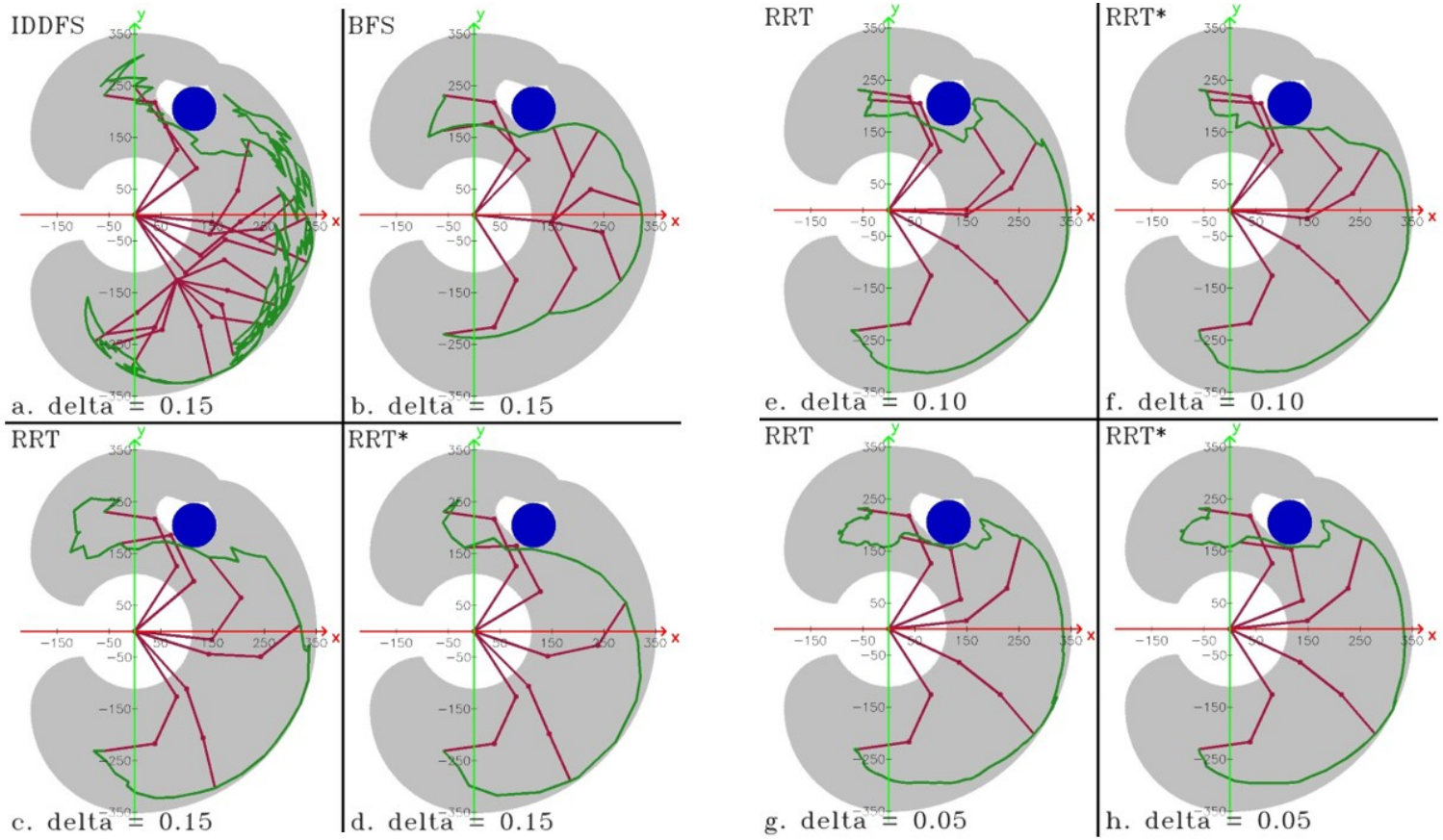


Fig. 6. Generated tree when $\delta = 0.15$ from algorithm: a. IDDFS; b. BFS; c. RRT; d. RRT*.

Results – Generated Trajectories

- The found point-to-point trajectory (dark green line) from algorithm: a. IDDFS with $\delta = 0.15$; b. BFS with $\delta = 0.15$; c. RRT with $\delta = 0.15$; d. RRT* with $\delta = 0.15$; e. RRT with $\delta = 0.10$; f. RRT* with $\delta = 0.10$; g. RRT with $\delta = 0.05$; h. RRT* with $\delta = 0.05$.



Discussion – Tradeoffs and Use Cases

Algorithm	Speed	Path Quality	Resource Usage	Suitable for
BFS	Medium	Good	High	Known Maps
RRT	High	Acceptable	Low	Real-time
RRT*	Medium	Excellent	Medium-High	Repetitive Tasks

Conclusions

- RRT and RRT* fastest and most efficient
- BFS produced valid but slower results
- DFS and IDDFS failed with low δ
- RRT: best balance of speed and feasibility
- RRT*: optimal paths, good for repetitive tasks
- BFS: acceptable but slower
- DFS/IDDFS: not practical for fine resolution

- For the specific task and the specific robotic manipulator, the chosen planning algorithm is RRT, so that fewer computational resources can be used.

Future Work

- Other designs of a manipulation robot with additional degrees of freedom will be considered, for which additional translational and/or rotational joints are added.

The research presented in this paper was supported by the Bulgarian National Science Fund under contract no. KP-06-H87/5 from 06.12.2024.

Thank you for your attention!