Computing trajectories in 3D space to avoid obstacles using systematic and stochastic search algorithms

MSc thesis by Vasileios Markou

09/21/2021

Artificial Intelligence & Data Analytics
University of Macedonia
2020-21

Main topics discussed

1. Problems tackled
2. Previous work
3. Simulation software
4. World of simulation
5. Algorithms
6. Implementation of algorithms
7. Storing information
8. Experiments

Problems tackled

Construction of a Jenga tower

Construction of a “Π” that consists of Jenga tiles

Implementation of RRT, Dijkstra, A*

Planning using RRT, Dijkstra, A*

World setup for experimentation and measurements

- The Jenga tower is meant to reach 3 levels of height
- The simulated physics tend to limit our ability for further constructing it
- We mainly exploit the use of a distance and a camera sensor on a custom-made robot with 6 DOF
- Feed from the camera is in real-time
- Distance sensor was picked in order to avoid the load of a force sensor as in other similar projects
- The “Π” is built by the same robot with minor changes and consists of 3 Jenga tiles

Various questions emerged during the authoring of this dissertation, such as;

- pros/cons of robotics
- application of robotics to our lives
- simplicity of controller **VS** automation of process
- costs and their reduction when compared to different sensors

However, as in many aspects of IT the answer is that it depends upon each problem.

Previous work

Jenga game by a manipulator with multi-articulated fingers (IEEE 2011)

Main goal; create a robot capable of playing jenga using artificial fingers

Tools; omnidirectional camera, degree of danger

Results; the robot was able to compete against a human and create 4-level jenga tower due to physical limitations

Conclusion; sophisticated strategies and a single camera could provide very accurate predictions to create the game environment for the robot

Citation; Yoshikawa, Tsuneo, et al. "Jenga game by a manipulator with multiarticulated fingers." 2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). IEEE, 2011.

Greedy Stone Tower Creations with a Robotic Arm (IJCAI-18)

Main goal; a robot is tasked with stacking stones of unknown shape on top of the other

Tools; PCL framework, clustering algorithms, a RGB camera and a force/torque sensor

Results; 100% in only 2 of 11 tries at 4-level creation

Conclusion; the actual results did not meet the scientists' expectations, however they gave us useful insight on understanding their goals

Citation; Wermelinger, Martin, et al. "Greedy stone tower creations with a robotic arm." Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18). Lawrence Erlbaum Associates, 2018.

Similar problems in the last decade

- Wermelinger, Martin, et al. "Greedy stone tower creations with a robotic arm." Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18). Lawrence Erlbaum Associates, 2018.
- Yoshikawa, Tsuneo, Tatsuya Sugiura, and Seiji Sugiyama. "Development of a Jenga game manipulator having multi-articulated fingers." 2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI). IEEE, 2012.
- Fazeli, Nima, et al. "See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion." Science Robotics 4.26 (2019).
- Yoshikawa, Tsuneo, et al. "Jenga game by a manipulator with multiarticulated fingers." 2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM). IEEE, 2011.
- Hernández, Juan David, et al. "Increasing robot autonomy via motion planning and an augmented reality interface." IEEE Robotics and Automation Letters 5.2 (2020): 1017-1023.

Simulation software

- Robotic systems require a lot of testing before we proceed, a huge part of which is done in a simulation environment
- In this way it becomes easier to find bugs and flaws before they rise in real-life scenarios
- It is cost-efficient, since even for premium software it is usually cheaper than constructing the robot
- One of the main disadvantages is that it is common that in real cases the actual result varies dramatically from the simulation

Free simulation software

- Webots
- V-Rep
- Gazebo

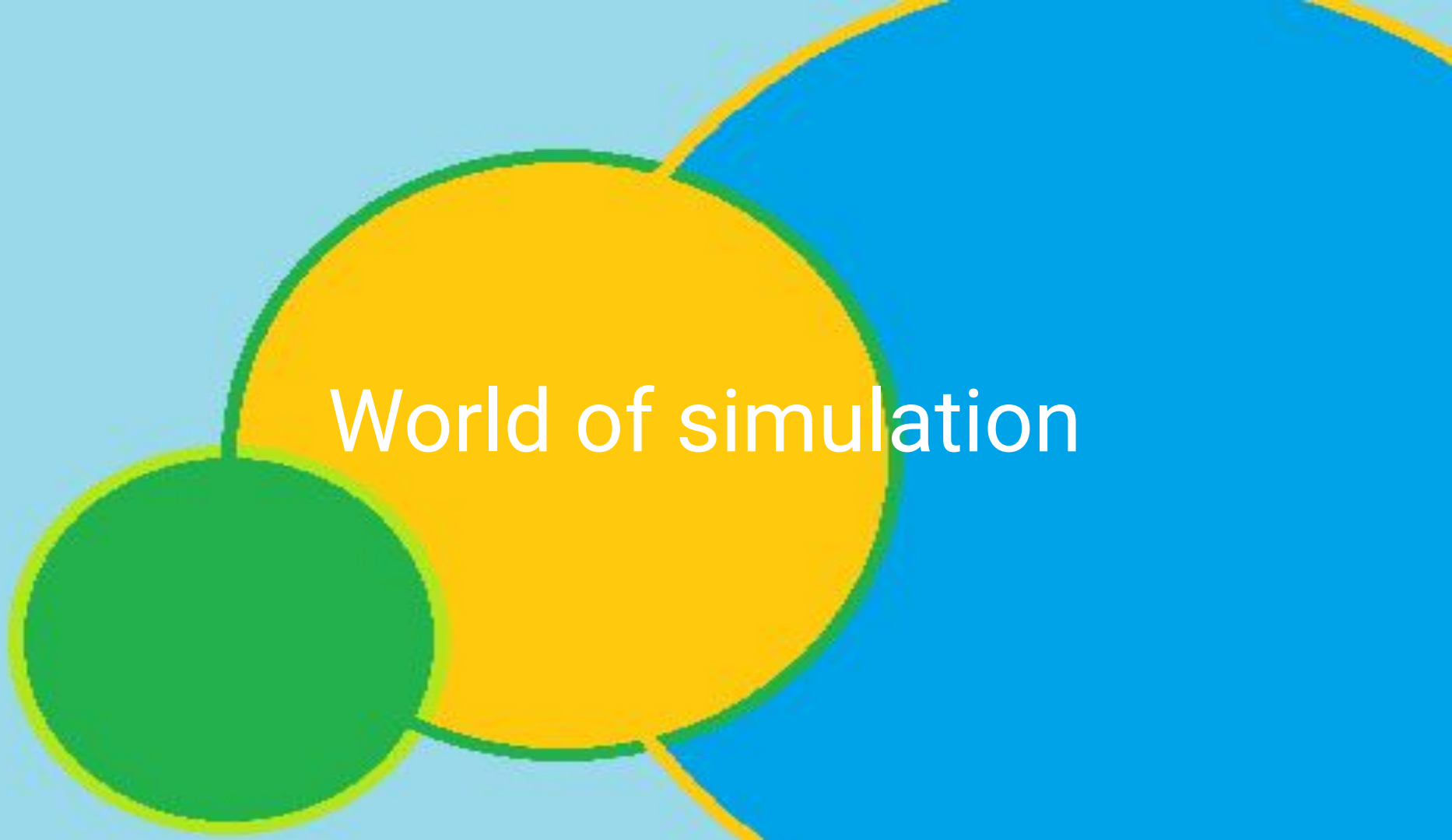
Commercial simulation software

- RoboDK (1 month trial)
- Octopuz (demo presentation after mutual agreement)

In this dissertation, all experiments are run on *Webots* and from when we use the “world” we refer to the environment setup in this world.

Main advantages

1. various 3D model formats supported
2. wide range of industrial robots
3. variety of languages to program controllers in, such as *python, java, c, c++, matlab,* including supported documentation
4. active community


World of simulation

The problems discussed previously are all tackled in 3D

Three main entities to construct our world;

Floor

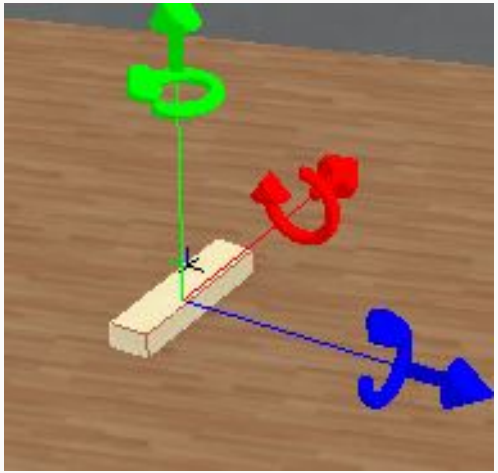
Tiles

Obstacles

Tiles

Our robotic arm will have to pick 9 of those up to construct a Jenga tower

Measurement unit is in *cm*



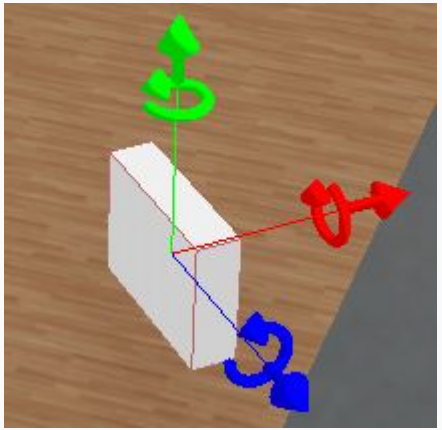
Obstacles

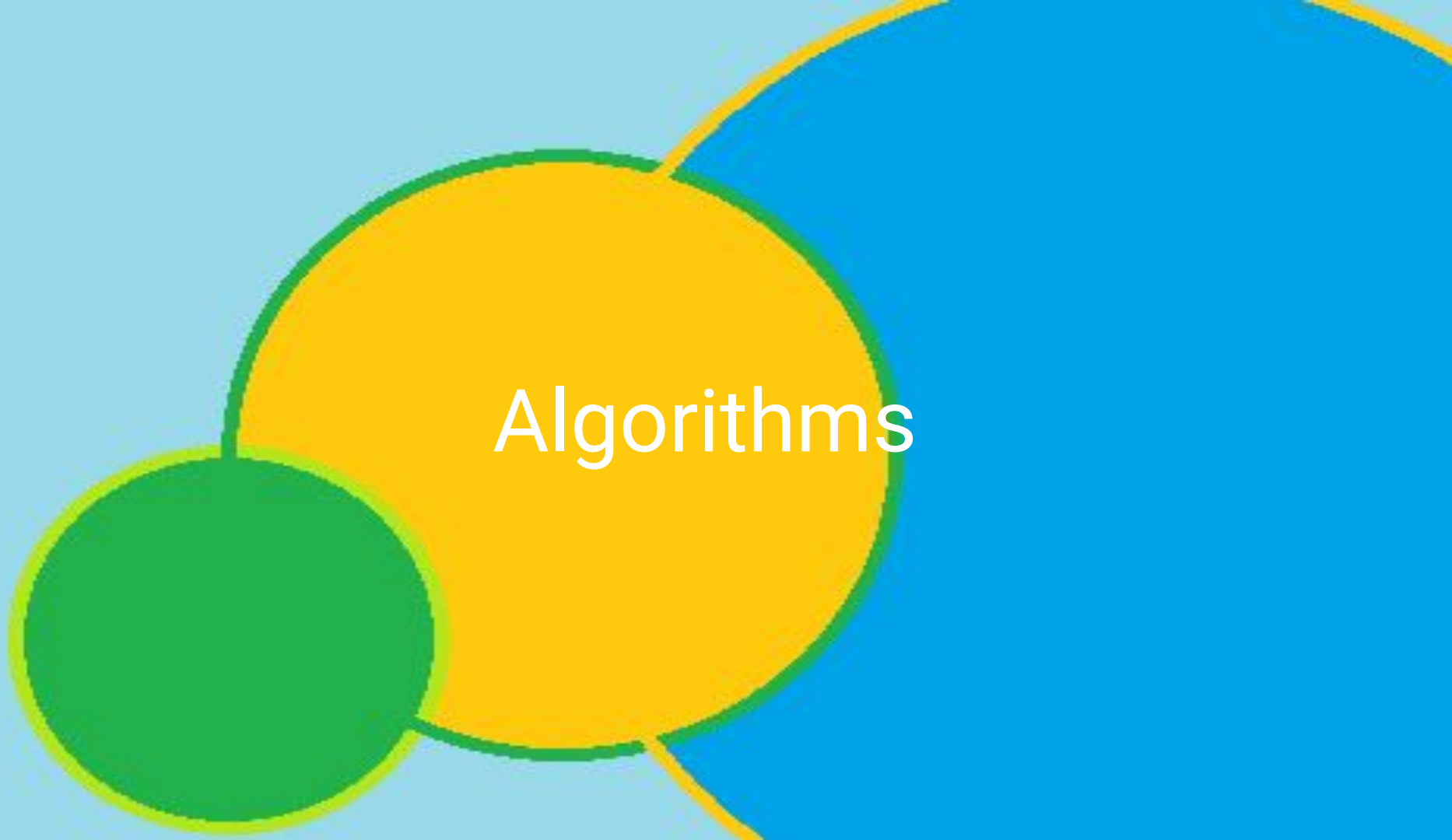
Our robotic arm is required to avoid all of them

Avoidance is considered valid when the grip of the robot is far from an obstacle based on some given distance

Their number is not standard as it is based on all we encounter

Measurement unit is in *cm*




Algorithms

RANDOM-BASED	SYSTEMATIC
RRT	A*
RRT*	Dijkstra
Probabilistic Roadmaps	
Artificial Potential Fields	
Monte Carlo	

RRT

- explores configuration space no matter the dimensionality
- two main factors affect the process
- maximum step growth to try and even branch sizes
- number of iterations before the algorithm completes
- certain researchers can end algorithm simply if goal criteria are met
- first implementation of RRT*

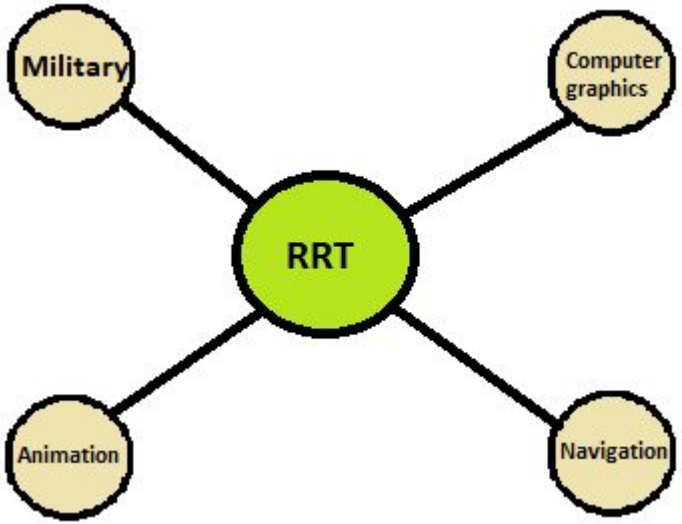
RRT

Path Planning with RRT

RRT Algorithm

```
g = Initialize algorithm
for  i = 1 to K
      next = random_move()
      g.add(next)

return solution
```



RRT

Variations of **RRT**

- **RRT* FND** - extension to RRT* for dynamic environments
- **CERRT** - variation of RRT to include uncertainty
- **TB-RRT** - variation of RRT to include time in constraints
- **RRT*-AR** - variation of RRT* dealing with alternate routes
- **RRG** - (Rapidly-exploring Random Graph) variation of RRT for optimal solution convergence

A*

-used in graph search primarily

-features a heuristic function

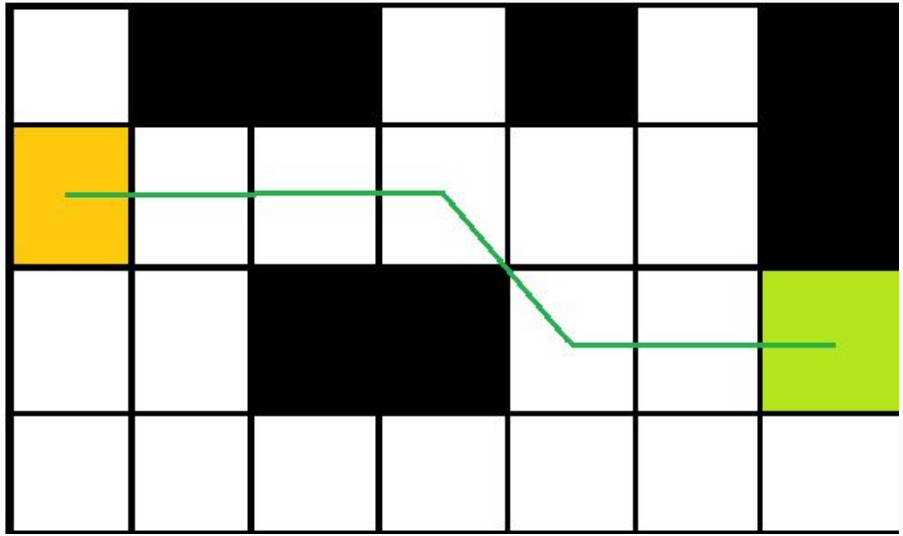
-considered a complete algorithm

-used in motion/path planning when it comes to robotics

A*

A* heuristic function

- expressed as $f(n) = g(n) + h(n)$
- n refers to the next node to be processed
- $f(n)$ refers to the quality of this node
- $g(n)$ refers to the distance covered so far
- $h(n)$ refers to the heuristic value
- Euclidean distance as a common metric in path planning



Dijkstra

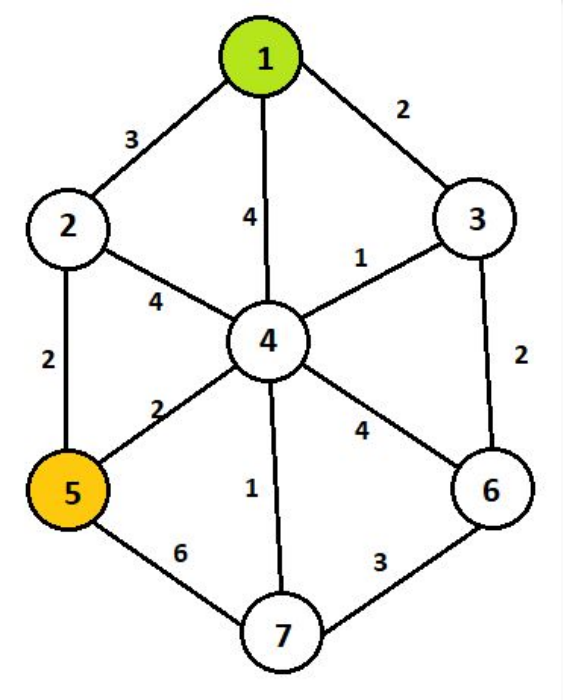
I. used in graph theory & traversal
II. aims to find minimum cost of a path between nodes
III. poses as a special case of A* (without the heuristic function)
IV. weights to be present between interconnected nodes
V. it is behind OSPF (Open Shortest Path First - internet connecting to gates)
VI. it is a greedy approach

Dijkstra

In this example, we can see that this algorithm can even be used in non-directed graphs

We still have the weights

We only need a start and end nodes



Implementation of algorithms

Implementing **RRT**

Input

- starting position
- target position
- list of obstacle positions
- radius for obstacle avoidance
- step size to limit how much we can proceed in a single motion
- number of different branches to keep
- threshold to consider that goal has been reached

Implementing **RRT**

```
for   j = 1 to B
    G(j).initialize(Cinit)

for n = 1 to N
    for   b = 1 to B
        do
            nRandom = random_move()
        while nRandom collides with any o in Os

        nClose = nearest_neighbor(G, nRandom)
        nChosen = limit_distance(G, Md, nClose)

        G(b).add(nClose, nChosen)

        if distance(nChosen, target) < threshold
            break

return G
```

```
G.initialize(Cinit)
for n = 1 to N
    nRandom = random_move()
    nClose = nearest_neighbor(G, nRandom)
    nChosen = limit_distance(G, Md, nClose)
    G.add(nClose, nChosen)
return G
```

- ➔ collision checks
- ➔ branch number
- ➔ target threshold

Implementing **Dijkstra**

Input

- starting position
- target position
- list of obstacle positions
- radius for obstacle avoidance

We also have added a threshold to help with marking certain motions as explored

Implementing **Dijkstra**

Implemented VS Original

```
while Q is not empty
    for every node n in Q
            best_step.add(compute_best_step(n))
    t = min_distance(Q, distance, obstacles, best_step)
    if step for motion < threshold
        Q.remove(t)

    for every neighbor n of t
        newPath = distance[t] + length(t, n)
        if newPath < distance[n]
            distance[n] = newPath
            previous[n] = t

return (distance, previous)
```

```
while Q is not empty
    t = min_distance(Q, distance)
    Q.remove(t)

    for every neighbor n of t
        newPath = distance[t] + length(t, n)
        if newPath < distance[n]
            distance[n] = newPath
            previous[n] = t

return (distance, previous)
```

Implementing **A***

Input

- starting position
- target position
- list of obstacle positions
- radius for obstacle avoidance
- minima
- media

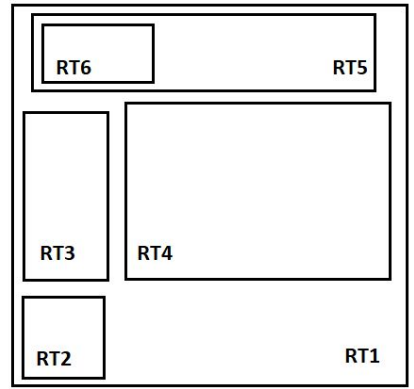
Implementing **A***

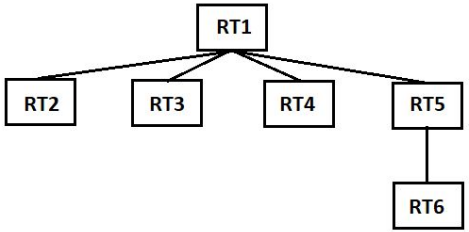
- ⏩ We use min/mid/**best** steps to perform branching of movement
- ⏩ Min/mid are provided by the user
- ⏩ **Best** is calculated as the farthest a single motion can take the robot avoiding collision and closing to the target
- ⏩ Euclidean distance as the metric to measure movement progress
- ⏩ Explored areas are store in **R-Tree**

Storing information

In **A***, in order to cope with the growing need for information storage we are using a R-Tree

- Emerged as a concept in 1984 by Antonin Guttman
- Multi-dimensional boxes can be sorted through advanced indexing
- MBR helps provide greater accuracy to correctly piece spatial indexes
- Uses pages in order to split data storage after exceeding a given threshold
- Deletion sometimes is more resource-demanding than keeping unused information



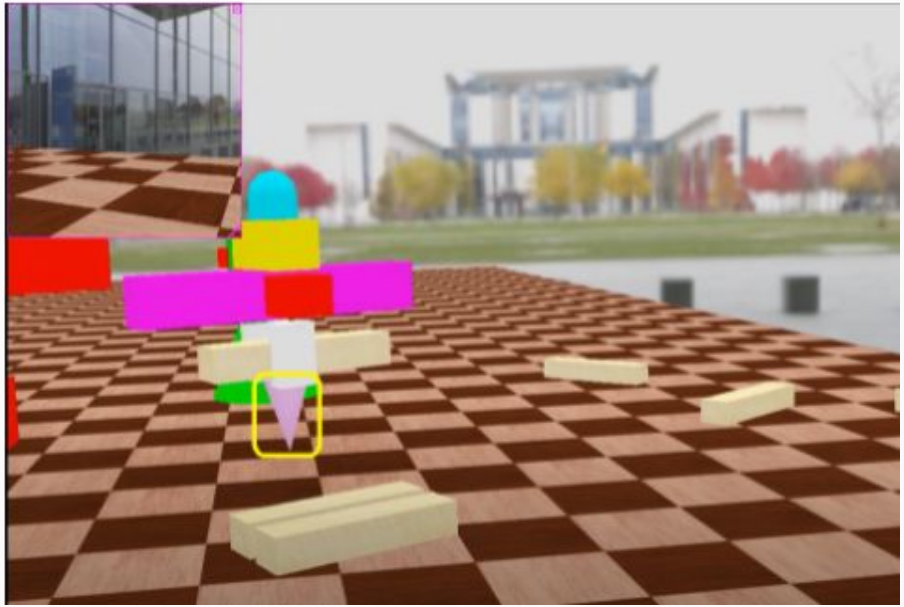


➔ The (*id, MBR*) help identify each node

➔ Every R-Tree follows a tree structure

Experiments

- Distance sensor offered simplicity compared to force/torque sensors, however in real-case scenarios it could sacrifice accuracy significantly
- Custom-made robots offer a more specific approach to one’s needs
- The controller logic could easily adapt to create a “Π” instead of a Jenga tower



- Overall informed algorithms were able to approach closer to the target every time compared to RRT
- Minor differences between Dijkstra, A* probably due to rounding errors
- All values expressed in meters

RRT	Dijkstra	A*
0.16	0.13	0.12
0.16	0.14	0.12
0.16	0.13	0.13

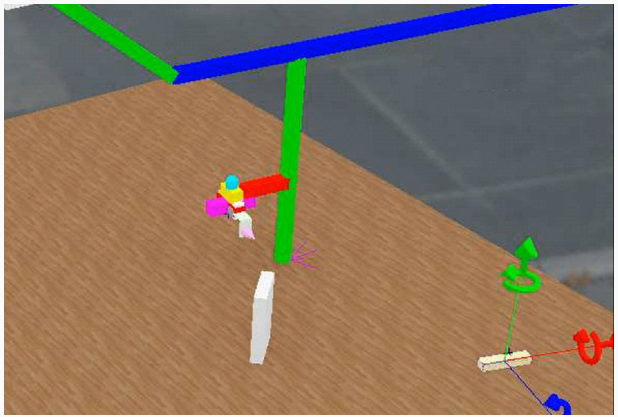
- Informed algorithms performed faster in terms of reaching the goal position compared to RRT
- Values are measured in *msec*

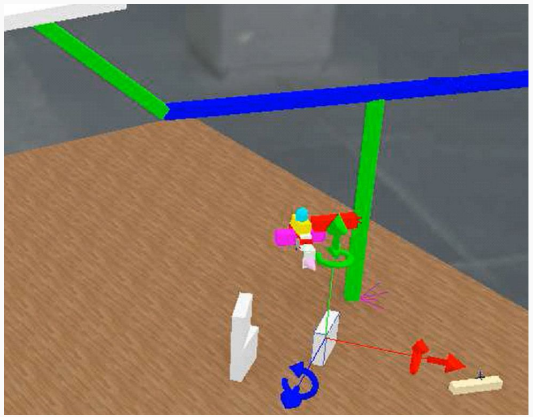
RRT	Dijkstra	A*
13.97	3.08	1.44
13.23	3.83	3.04
15.56	8.30	6.99

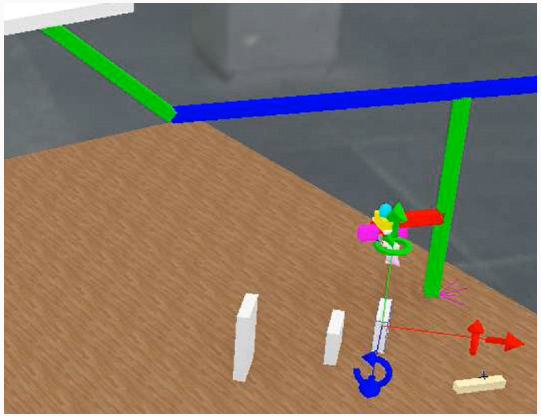
- Informed algorithms were able to go through fewer states and reach their goal
- States refer to the number of actual motions that need to be performed

RRT	Dijkstra	A*
42	33	30
114	15	13
137	58	46

We decided to run more experiments to test our A* implementation, by gradually adding more obstacles







Thank you for your time!

Supervised by

Ioannis Refanidis, Professor