h_da HOCHSCHULE DARMSTADT UNIVERSITY OF APPLIED SCIENCES fbi FACHBEREICH INFORMATIK	Prof. Horsch R. Scheitler Robotics Lab	

Practical Exercise 6: Motion Planning

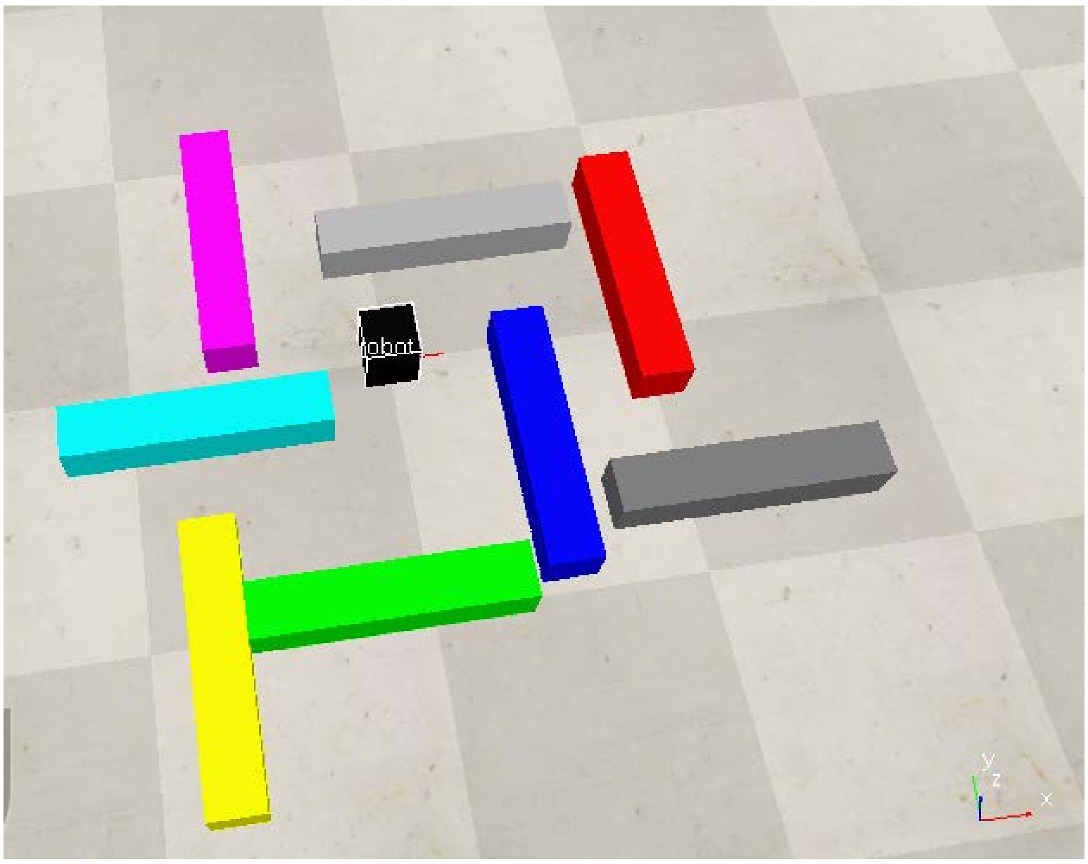
Objective:

In this practical exercise you implement a motion planning algorithm for a robot with two translational degrees of freedom moving in a 2-D maze. As a baseline you will use the configuration space (c-space) calculated in the previous exercise.

For this purpose you implement in Python (here we provide a source code framework) or C++ (here, we **do not provide** a source code framework) and evaluate your results by visual inspection using CoppeliaSim.

Task:

In the scene *DOF2_cartesian.ttt* you see a black robot with two translational degrees of freedom moving in a maze (see figure).



You should implement a motion planning algorithm based on Breadth-First Search (BFS).

Starting from a 2-dimensional c-space, which has been stored in a bitmap file, a collision-free path in the c-space should be calculated. Each pixel in the bitmap corresponds to a small 2-dimensional area. If the color of the pixel is white, this small area belongs to the freespace. Use the breadth-first search for calculation a collision free path.

The following pseudo code can be used:

```
1.  Insert start configuration into the empty queue
2.  Mark start configuration as visited and set distance level to 0
3.  Take the first configuration from the queue ( queue.pop(0) )
4.  If the current configuration is the goal configuration, the algorithm terminates
5.  For each neighboring configuration, which is not marked visited and is collision-free
    and lies within the bounds of the work-space increment the distance level and insert
    into the queue
6.  Jump to step 3, until the algorithm terminates or the queue is empty (no path found)
```

After termination you need to move backwards from the goal configuration to the start configuration based moving “downward” decreasing the distance levels down to 0.

Evaluate your algorithm with different start and goal configurations.
To do so, start your python script and move the robot to the desired start configuration (saving this configuration by pressing ‘s’), then move the robot to the desired goal configuration (saving this configuration by pressing ‘g’). Then press ‘q’ to quit the endless loop.

Also change between N4 and N8 neighborhood (set the appropriate variable to 4 or 8).
You can draw your own configuration space (for example with the program *paint*) and evaluate with own examples.

Color configurations (pixels), which have been explored by BFS with a different color and color pixels of the returned path as black.

Hints:

- You can use the sample code P6_Task1.py. There you need to implement the method *def FindPathBFS(cspace, nStart, nGoal)*

Preparation:

- Make yourself familiar with the given code example
- Make yourself familiar with the methods to use (BFS)
- Have a scaffold of your program ready

Report:

The report should contain
- the task description itself (you can reuse this document, at the top your name(s) must be mentioned)
- a comprehensive description of the implemented algorithm
- Sketches and explaining figures
- source code with result output
- you deliver one zipped file. It contains the report as PDF, the .ttt scene files and the solution python files including all additional project files necessary, to run the program. Your program must start out of the box.
- You can provide one solution for one group consisting of max. 2 students.

Naming Conventions: as in exercise 2

The solution should be similar to the following picture (explored space in brown rgb=[128,0,0] and the path in black).

Observe, that the collision free path is always close to obstacles, since the BFS returns the shortest path.