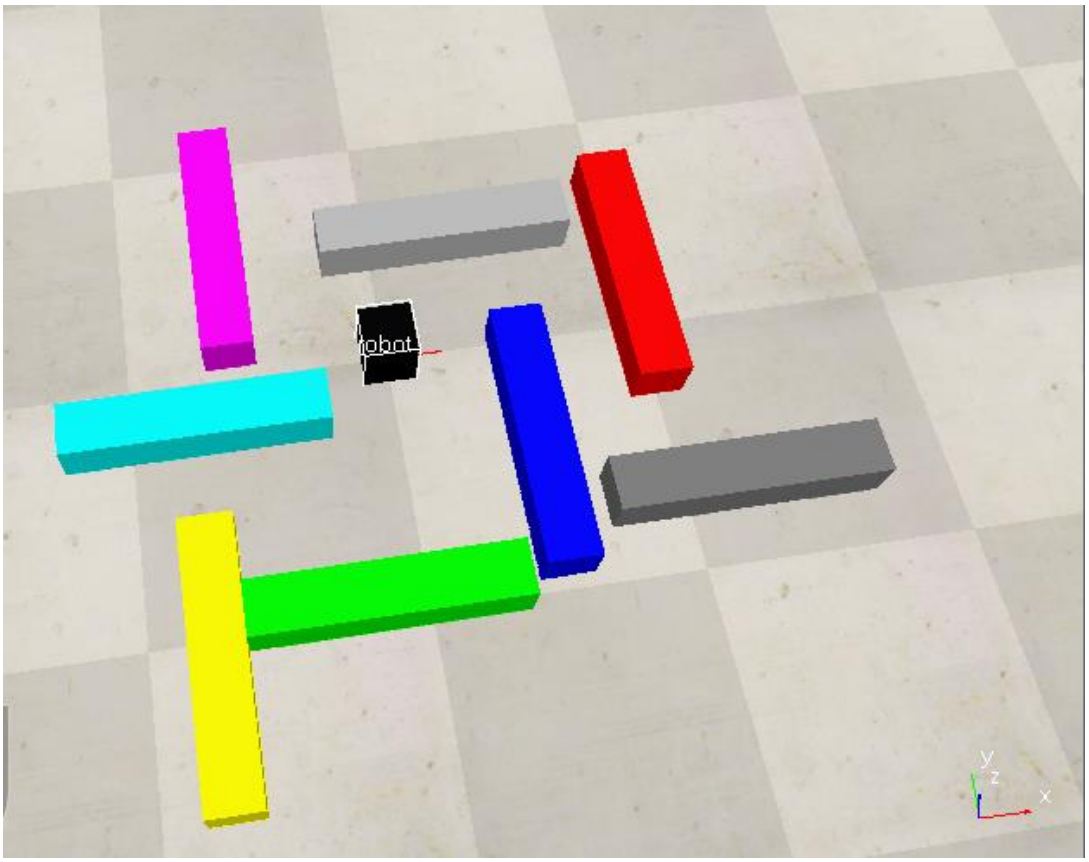
Practical Exercise 5: Configuration Space

Objective:

In this practical exercise you should learn how you can generate a 2-dimensional configuration space (c-space) including its configuration space obstacles for a robot with two translational degrees of freedom moving in a maze. (Hindernisse)
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).



In the scene are 8 differently colored obstacles. Build the configuration space of this environment as a 2-dimensional array and visualize it as a bitmap file. In this bitmap file the configuration space obstacles should have the same color. The freespace should be colored in white.

The size of the c-space is 1000 x 1000 (although you can start with a smaller resolution to develop faster) covering the workspace of the robot [-1m, 1m] x [-1m, 1m]. So each pixel is 2mm x 2mm = 4 mm^2.

Hints:

- The 8 obstacles in the scene have the same dimensions (0.1m x 0.5m x 0.1m) and are axis aligned. Some of them are rotated by 90 degrees.
- These obstacles are already processed in the file P5_Task1.py and stored in a list of obstacles. Each element of the list has the format: [x_min, x_max, y_min, y_max].
- See bitmap.py as a reference how to save and load bitmap images and convert between arrays and bitmap images.

Preparation:

- Make yourself familiar with the given code example
- Implement an algorithm to check collision between two **axis aligned** boxes in 2D.
  - This algorithm should be used to generate the configuration space obstacles
  - This routine should get two workspace obstacles as parameters and return a boolean value (True: collision, False: no collision). So you need to transform pixel coordinates into workspace coordinates before calling his routine.
  - Keep the code simple. The optimal solution checks 4 conditions. You can start thinking about the cases, where are no collisions.
  - This algorithm should be used to determine a collision free configuration (or not) by setting the robot in all possible configurations (brute-force)
- 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

Outlook for next exercise: you will reuse the c-space (bitmap) file for path planning purposes.

Proof of correctness:
The solution should be similar to the left picture based on the right work space: