

Modelling, simulation, and planning for the MoleMOD system

Michaela Brejchová¹, Miroslav Kulich²[0000-0002-0997-5889], Jan Petrš³, and Libor Přečil²

¹ Faculty of Electrical Engineering

Czech Technical University in Prague, Prague, Czech Republic
brejmi3@fel.cvut.cz

² Czech Institute of Informatics, Robotics, and Cybernetics,
Czech Technical University in Prague, Prague, Czech Republic
kulich@cvut.cz, preucil@cvut.cz

<http://imr.ciir.cvut.cz>

³ Faculty of Architecture

Czech Technical University in Prague, Prague, Czech Republic
petrjan@fa.cvut.cz

Abstract. MoleMOD is a heterogeneous self-reconfigurable modular robotic system to be employed in architecture and civil engineering. In this paper we present two components of the MoleMOD infrastructure - a test environment and a planning algorithm. The test environment for simulation and visualization of active parts as well as passive blocks of MoleMOD is based on Gazebo - a powerful general-purpose robotic simulator. The key effort has been put into preparation of realistic models of passive and active components taking into account their physical characteristics. Moreover, given a starting configuration of the MoleMOD system and a final configuration an approach to plan collision-free trajectories for a fleet of active parts is introduced.

Keywords: modelling, simulation, planning, self-reconfiguration, modular robotic systems

1 Introduction

MoleMOD is a unique self-reconfigurable modular robotic system developed at Czech Technical University in Prague [4, 3]. The system is heterogeneous as it consists of active robots and passive modular building blocks and it is inspired by colonies like termites, ants or bees, which permanently rebuild and adapt their “houses” to surroundings and current conditions.

The robots are flexible, can rotate, and are able to connect to the passive blocks as well as pick up and carry them along a given trajectory. This approach offers extensive possibilities of reconfiguration and adaption due to separated mobile and passive units. The two parts separation gave the system its acronym Mole (animal) + MOD (module). The passive part can be imagined as units



Fig. 1. The MoleMOD system.

of regular 3D lattice, just like individual crystals or voxels, respectively, in a crystalline lattice or virtual digitized volume.

The active part, as it is now, is further decomposed to three essential parts: soft/flexible body, revolving heads and a rotator in the centre. The head is primarily used for screwing the passive units to hold together and the secondary function is to ride over the construction site¹. The flexible body allows for the peristaltic movement, through the trajectories within the passive block conglomerate. A secondary function of the body, not least important, is the manipulation with the blocks, so picking up and carrying. Finally, the rotator allows rotation of the blocks as is typical for majority of modular robotic systems.

The MoleMOD system is very adaptable and can be used in many various situations, especially at locations that are not safe for people, or there is a problem to build – places like deserts, mountains or polar regions, which cannot be inhabited, but it can be necessary to build there. MoleMOD may not be used only for building houses, but also for bridges, pylons or research stations.

The system does not need cranes or other external construction machines. Therefore it is quite easy to transport it to the building site. It will even be possible to transfer only robots; building modules will be created by a 3D printer from local materials. This way the system could be used for the colonization of another planet.

Other uses may be for example temporary constructions. Tribunes for sports events, such as the Olympic Games or the World Cup races, markets, exhibitions, festivals, events that last only a few days or weeks. No less important is the possibility of using the system in case of a disaster. It can be building of bridges after a flood, shelters for people who have lost their homes due to a catastrophe and so on. Also, it can be used after a nuclear disaster, when the presence of humans is not possible because of radiation.

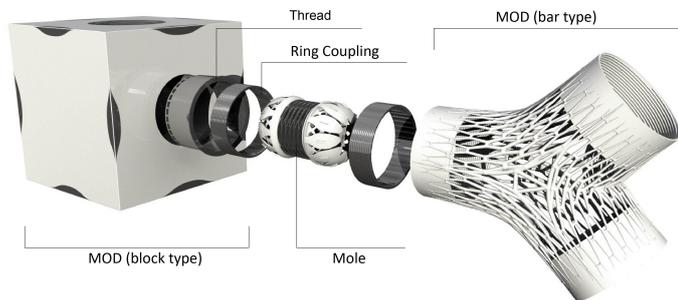


Fig. 2. Structure of the system.

The rest of the paper is organized as follows. Section 2 is dedicated to the description of the simulation environment for MoleMOD based on the Gazebo simulator, while the planning approach is presented in Section 3. The final comments and future work are described in Section 4.

2 Simulation environment

This section provides a description of the Gazebo simulator and design of models (both passive blocks and active parts) for testing the MoleMOD system in Gazebo. Gazebo is an open-source robotics simulator, which can be used to design robots, test algorithms and artificial intelligence systems using realistic scenarios. The simulator offers indoor and outdoor environments with the possibility of setting several properties, such as wind, gravitation, friction and so on. Gazebo includes multiple physics engines (ODE, Bullet, Simbody and DART), a library of robot models and environments, several types of sensors and functional graphical and programmatic interfaces [1].

A simulation environment in Gazebo is described in so called world files, which include specification of elements such as robots, lights, sensors or static objects. The files use SDF (Simulation Description Format), an XML format originally developed for Gazebo.

Model files are similar to world files but contain only specifications for a model. The model created by this file can be included in a world file, so it is possible to use one model several times without rewriting the entire code. Also, there is the online model database.

SDF models can be just simple shapes but also complex robots. Basically, a model consists of links, joints, sensors, collision objects, visuals and plugins [2]. A **link** contains the physical properties. It is a body of the model or its part. It may have many collision and visual elements. A **collision** element is a geometry that is used to check collisions. A link can contain many collisions. A **visual** element visualize parts of a link. A link can have many visuals or none. A **joint** connects two links. Each joint has a parent and a child, an axis of rotation and

some other properties. A **sensor** collects data from the world and these are then used by plugins.

2.1 Modelling passive blocks

As passive blocks do not move, only their shape needs to be defined in the form of a triangular mesh – a collection of triangles. Even simplest blocks, cubes with two straight circular tunnels, consist of many triangles, when a precise approximation is needed, see Fig. 3 (left). We, therefore, assume square-shaped tunnels for which only a fraction of triangles is needed, Fig.3 (right).

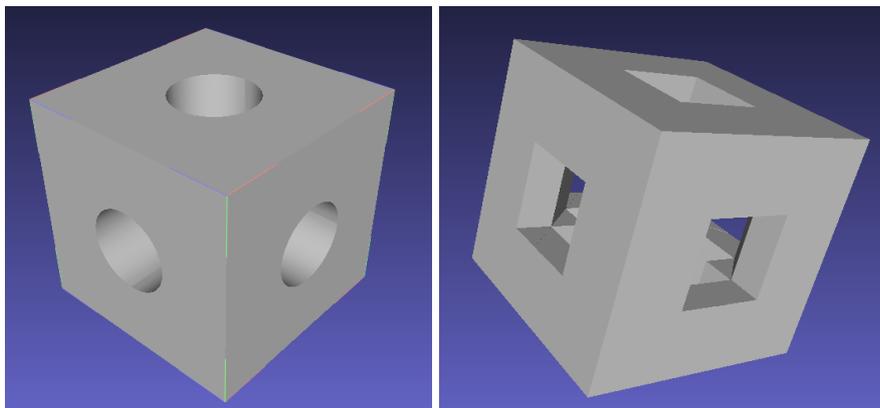


Fig. 3. A cube-shaped block with two tunnels. Precise approximation of a cube-shaped block with circular tunnels contains 6174 triangular faces (left), while a description of a block with squared tunnels contains 352 faces.

Triangular meshes can be defined in Collada (Collaborative Design Activity) files (.dae), which can be created either manually or using some modelling tool like MeshLab (www.meshlab.net/) or Blender (www.blender.org). This can be imported into a world file similarly to Listing 1.1.

```
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <model name="my_mesh">
      <pose>0 0 .2 0 0 0</pose>
      <link name='tunnel'>
        <visual name='visual'>
```

```

        <transparency>0.5</transparency>
        <geometry>
            <mesh>
                <uri>file://1.dae</uri>
                <scale>1 1 1</scale>
            </mesh>
        </geometry>
    </visual>
    <collision name='collision'>
        <geometry>
            <mesh>
                <uri>file://1.dae</uri>
                <scale>1 1 1</scale>
            </mesh>
        </geometry>
    </collision>
</link>
</model>
</world>
</sdf>

```

Listing 1.1. A world file with mesh importing

2.2 Modelling active parts

Active parts – robots – consist of three components: a soft body, revolving heads and a rotator. Precise modeling of these will be unnecessarily complex which will significantly slow down simulation of the whole MoleMOD system. Gazebo has, moreover, no ways how to simulate soft components. We thus model the robots making use of components available in Gazebo – links and joints. To describe the modelling process, we start with a simple model with limited functionality and make the model more complex and powerfull subsequently in several steps.

The basic model The basic model is constructed from two cubes connected by a prismatic joint, see Fig. 4. The model is limited to move only in one direction forwards or backwards by expanding and contracting the joint and changing the frictions of the links.

The forward motion of the model consists of four parts:

1. setting frictions
2. the joint expansion
3. setting frictions
4. the joint contraction

The frictions of the links are set in a model plugin. Before moving itself, it is necessary to lower the friction of the front cube (the first cube in the direction

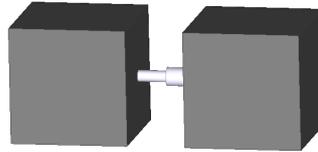


Fig. 4. The basic model

of the movement). For expanding the joint, a positive velocity is set to the joint in the plugin which leads to movement of the front cube.

The simulator updates in fixed intervals. When a predefined number of iterations is done, the plugin changes the sign of the velocity and swaps frictions higher for the front cube and lower for the back cube. After repeating the same number of iterations, the joint position is set to the initial (contracted) state.

The rotating model For turning, it is needed to add a revolute joint in the middle of the model. Because the simulator cannot connect two joints, it is necessary to add two links and one more prismatic joint. Now the model is made of two head cubes, two small middle cubes, two prismatic joints and one revolute joint, Fig. 5.

The first prismatic joint connects the first head cube with the first middle cube. This cube is joined with the second middle cube by the revolute joint. These both cubes are immaterial and they are placed on themselves. The second middle cube is connected to the second cube with the second prismatic joint.

After these changes, the code of the plugin has to be adapted. For translation, both prismatic joints expand and contract at the same time.

Rotation is similar to translation. The four parts of motion are:

1. setting frictions
2. the joint rotation to the direction, where we want to turn
3. setting frictions
4. the joint rotation to the different direction

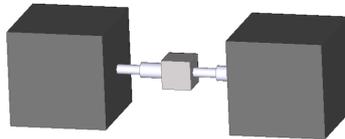


Fig. 5. The rotating model

At first, the friction of the cube to be shifted is reduced. The plugin then sets an input velocity to the revolutive joint. This velocity can be positive or negative depending on the direction of the turn. Finally, frictions are swapped and velocity with an opposite sign is set to the joint. In contrast to the moving ahead, this motion stops after one step and will never be repeated, because the model would spin on the same place.

Translation/rotation controller Unfortunately, the movements of the previous models are not precise enough. The joints move for the same time, but it does not guarantee that their final position is the same. To make the motion more accurate, a simple controller has been designed. The input to the controller is a position of the joint - the length of the prismatic joint or the angle of the rotation of the revolutive joint.

```

1     if(position < required_value - accuracy) {
2         setVel(vel);
3     } else if(position > required_value + accuracy) {
4         setVel(-vel);
5     } else {
6         setVel(0.0);
7     }

```

Listing 1.2. A joint controller

Listing1.2 shows a primitive controller that sets a positive velocity to the joint, if its position is lower than the required one, negative if larger or zero if it is in the interval determined by the deviation.

For better control, we can divide the possible positions into more intervals. The result will be similar to the example above; it will only contain more conditions. The controller used in the simulator is divided into five intervals. At the beginning of the motion, the joint is set to an initial speed. When the joint position is close to the required position, the speed is decreased to half the value.

The controller accuracy is 50 micrometres. In this range, the joint is set to zero velocity. If the position is larger or smaller (depending on the direction of the motion) the plugin sets a negative speed to the joint. Thanks to this condition there is no need to set the reverse speed for the contracting, the controller will solve it. Also, it is not necessary to count updates; one step consists of expanding to input length and contracting back to the initial state.

The second input for ahead motion is a number of iterations, but in this case, it is not the number of updates, but the number of calls of the controller (the total sum of all expanding and contracting). The distance the model moves is equal to the product of one step length and the number of iterations.

The final model The model with just one revolutive joint is not sufficient. To turn in a tunnel or lift a building unit, at least two revolutive joints are needed. It is also not convenient to use a simple revolutive joint, because it can rotate

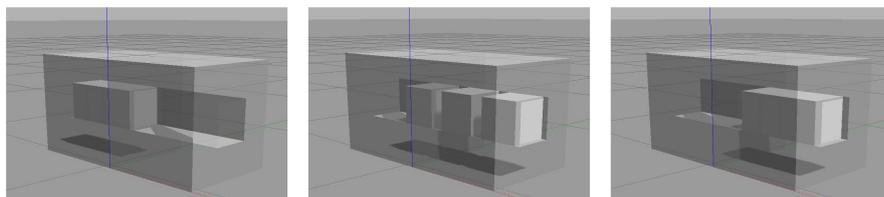


Fig. 6. Moving in a tunnel - the final model

only around one axis. With the joints, the robot could rotate sideways or up and down, but could not do both of these operations. A solution to this problem is to use another joint. The joint is called universal in the simulator and it can rotate around two axes.

The final model consists of three main cubes, four prismatic joints, two universal joints and four little immaterial cubes that are between joints as depicted in Fig. 7.

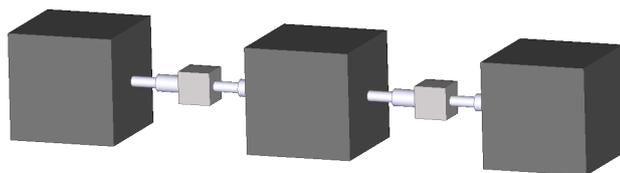


Fig. 7. The final model

2.3 Experiments

The models for the simulator have been made; the next step is to test them. In the previous chapter, the model of the MoleMOD robot was introduced and two simple moves (forward moving and rotation) were described.

Forward movement works on the same principle that has been described previously, see Fig. 8. The only difference is that the final model has four prismatic joints instead of two. The model could use all four joints when moving, but it is simpler to use only two, for example, the first and the fourth joint. That will spare us larger changes in the code and also the motion will be more accurate.

The rotation remains practically the same with only two minor changes. The last model contains two universal joints, so it is needed to decide which joint will rotate. Then the rotation axis has to be set because the universal joint can rotate around two axes. An example of rotation is depicted in Fig. 9.

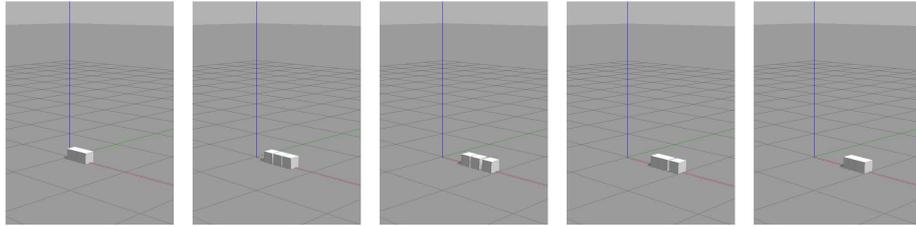


Fig. 8. Forward moving

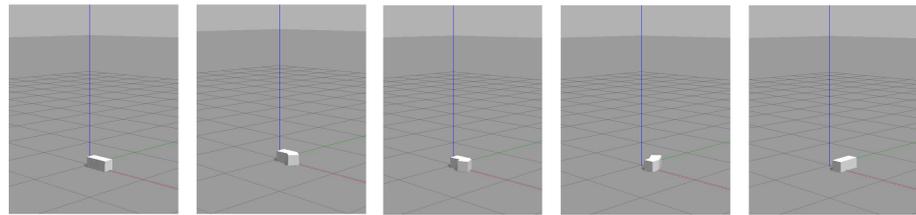


Fig. 9. Turning right

However, for simulating the work of the system, these two motions are not sufficient. For the basic version of the planning, it is essential to add lifting (putting a module on a next block or lifting a module just up), shifting modules and moves of robots in tunnels.

Robot movement in a straight tunnel is exactly the same as the forward moving. The length of the motion is adjusted according to the size of the block, see Fig. 10.



Fig. 10. Moving in the straight tunnel

Turning in a tunnel is more complicated. Because the robot is only a little bit smaller than the tunnel, it is impossible to turn around at once. It is needed to combine both types of moves - rotation and translation. This can be achieved by using a joint controller, see an example in Fig. 11

Lifting is, similarly to turning, a combination of translational and rotational motion, see Fig. 12.

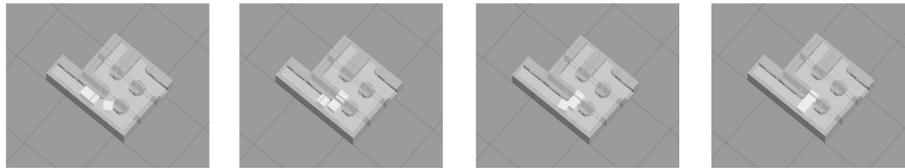


Fig. 11. Turning in the module

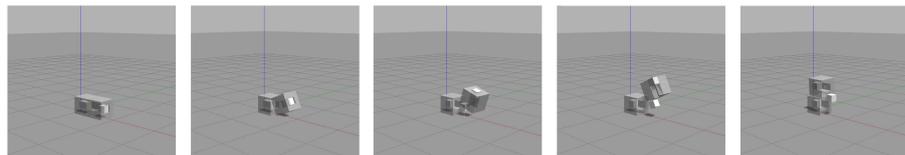


Fig. 12. Lifting

3 Planning

Planning in MoleMOD is based on the A* algorithm – an informed method for state-space search. A state is represented by an arrangement of passive blocks and robots positions. We assuming two-dimensional space, which can be described as a two-dimensional matrix, where each cell stores the information whether the corresponding space is empty, it contains a block, or a block with a robot.

An action is performed by movement of robots, which can relocate some blocks. The simplest action is moving from one block to another. The natural condition for this action is that the position, where the robot moves to, is neighboring to the current robot position, lies inside a given area and also it contains an empty block. The robot can move to the right, to the left, up and down this way.

Another type of movement is lifting of a block. In reality, the model expands and partially inserts into the blocks beside. After that, the robot lifts one block up and place it on the top of the second one. Finally, the model retracts into one of the two blocks. To reduce the number of possibilities, we assume that the robot picks up the block, where it originally was, and remains in the block after the movement. To the motion can be executed, there has to be a free space around the moving block.

Putting down is similar to lifting. The robot expands into the block under its position and then contracts with the top one. For simplicity, the model starts and finishes again in the moved block.

The last motion that is possible with only one robot is moving a block to the right or the left. The model is in the block that we want to shift. If the place next to the block is free and under it, there is another block, the robot expands

to this block. Then it moves the block to the empty position and contracts. The moving finishes again in the shifted block.

All movements, which were mentioned above, are valid also for the case with more robots. The advantage is that they can be performed in parallel, so the entire construction is done faster.

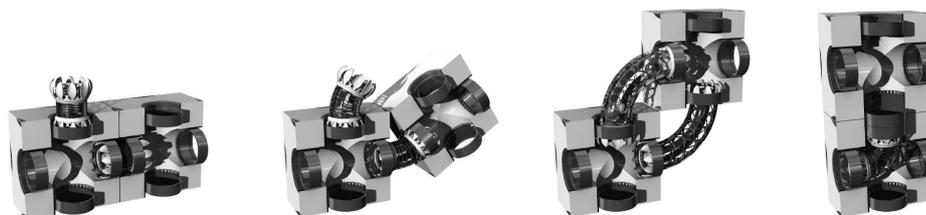


Fig. 13. Cooperation of two robots

Besides, robots can work on a movement together. A block is lifted by one robot to a certain position, where a second robot takes it and completes the move as shown in Fig.13.

The planning algorithm applies one action, where cooperation of two robots is used. When the block is lifted by one robot, it is like stairs, the block moves not only up but also to a side. In some cases, however, it is necessary to move just upwards. The movement starts identically as lifting. The robot in the block expands to the block beside and then elevates the block one position up. The second robot has to be in the block that lies on the block where the robot stretched to before. The second robot expands and “catches” the block and the first robot contracts to the underlying block.

3.1 Cost calculation

The evaluation function of A* has a standard form $f(x) = g(x) + h(x)$, where $g(x)$ is the cost of the path from the start node to x , and $h(x)$ is a heuristic function that estimates the cost of the cheapest path from x to the goal.

One possibility of computing a value of the cost g is based on distance of the start and current states: the distance of a robot or a block moving is equal to $|i' - i| + |j' - j|$, where (i, j) are coordinates of the robot or the robot in the first state and (i', j') are its coordinates in the second state. However, computing the distance in every step means to find the robot or block that has just been moved, and calculate how far has shifted. That is senselessly complicated. Because the number of movements is limited and each has a specific distance that never changes, it is much simpler to assign a value d to every movement m :

$$g(x_n) = g(x_{n-1}) + d(m).$$

While the case of one robot is simple as the robot can perform only one movement in one step, the case with more robots is more complicated. For example, two robots can shift two blocks at the same time or gradually in two steps. The second option does not have any advantages, it only prolongs the building, so it is necessary to obviate it. That can be done by adding 1 to the distance in every step:

$$g(x_n) = g(x_{n-1}) + d(m) + 1,$$

where $d(m)$ is the sum of costs of the motions that have been made to get from the state x_{n-1} to the state x_n .

Counting the distance to the goal state is more difficult. A robot moves to a block, shifts it, moves to another block, shifts it and so over and over again until the goal state is reached. It is almost impossible to calculate the real cost.

When a block moves between two states, the cost is equal to $|i' - i| + |j' - j|$. If more blocks are moved, the cost is $\sum_{n=1}^N |i'_n - i_n| + |j'_n - j_n|$, where N is the number of blocks that have been moved, and $[i_n, j_n]$ and $[i'_n, j'_n]$ are coordinates of n -th block before and after the movement.

Costs of robots movement have to be added to the total cost, but it is hard to reckon them in advance. It depends on the number of robots and blocks and their positions. Every time a robot moves a block to a correct location, it has to move to the position next to the block. The minimal cost of the path from a block to another is one and the total cost of the way between blocks is equal to n_{wrong} , where n_{wrong} is the number of the blocks in the wrong positions which do not contain an robot. Therefore, the estimation of the cost of the path from the current state x to the goal state x_G is:

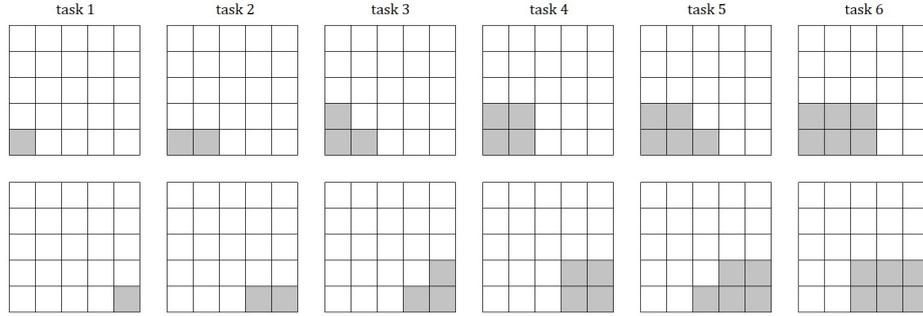
$$h(x) = \sum_{n=1}^{n_{wrong}} (|i_{Gn} - i_n| + |j_{Gn} - j_n|) + n_{wrong} - n_{robot},$$

where $[i_{Gn}, j_{Gn}]$ is a goal position of n -th block.

3.2 Experiments

Several experiments have been performed to demonstrate feasibility and time complexity of the planning algorithm. Six planning tasks depicted in Fig. 14 were run. In each pair, the upper state represents the initial layout of the blocks and the state below the goal arrangement. All tasks have been successively solved for different numbers of robots and for all cases the time of the planning was measured.

Table 1 shows that the time needed to find a solution, if the system contains more robots, is significantly higher than the case of one robot. However, the ratio of blocks and robots is also important. Robots need sufficient amount of space to move, so in the case that most blocks are occupied, the number of options is reduced. The other case is, if the quantity of blocks is quite higher than the number of robots, so the robots have plenty of space for moving. Each robot

**Fig. 14.** Planning tasks for time measurement**Table 1.** Time of planning

	time [ms]					
	1 robot	2 robots	3 robots	4 robots	5 robots	6 robots
task 1	0.24					
task 2	1.87	2.02				
task 3	3.08	18.23	5.10			
task 4	9.00	75.60	117.95	9.91		
task 5	15.50	224.10	614.93	332.15	18.63	
task 6	35.64	1212.75	4390.14	4000.69	706.35	22.48

can perform some movements, the number of moves depends on the specific conditions. One more robot adds its motions and combinations of its moves and moves of others. The larger space and the more blocks it contains, the more movements will be possible.

4 Conclusion

In this paper we presented MoleMOD – heterogeneous self-reconfigurable modular robotic system and two software parts of it – the simulation environment and the planning algorithm. The system is still in the first phase of development and thus only first results were presented which can be further improved. Especially the planning algorithm has a potential for next improvements. First, better heuristics, which more precisely estimates the cost to the goal, needs to be designed. This will guide the A* algorithm to search the state space more effectively by expanding less states. Secondly, we will investigate more advanced planning algorithms (e.g., hierarchical) which will further improve computational complexity of the planning. Finally, extension of the algorithm into the 3D case will be done. Regarding the simulation environment, we would like to equip robots with sensors to accelerate development of control algorithms for robots.

Acknowledgement

This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 688117, and by the European Regional Development Fund under the project Robotics for Industry 4.0 (reg. no. CZ.02.1.01/0.0/0.0/15 003/0000470).

References

1. Open Source Robotics Foundation: Gazebo. <http://gazebo.org/> (2018)
2. Open Source Robotics Foundation: SDF. <http://sdformat.org/> (2018)
3. Petrš, J.: MoleMOD. <http://www.studioflorian.com/projekty/347-jan-petrš-molemod> (2017)
4. Petrš, J., Havelka, J., Florián, M., Novák, J.: MoleMOD - on design specification and applications of a self-reconfigurable constructional robotic system. In: Fioravanti, A., Cursi, S., Elahmar, S., Gargaro, S., Loffreda, G., Novembri, G., Trento, A. (eds.) ShoCK! - Sharing Computational Knowledge! - Proceedings of the 35th eCAADe Conference. vol. 2, pp. 159–166. Fioravanti, A, Cursi, S, Elahmar, S, Gargaro, S, Loffreda, G, Novembri, G, Trento, A (eds.) (2017)