

Objects Manipulation

Project Report

Alex Lavriv and Vladimir Shargorodsky
Advisor: Prof. Ronen Brafman
20.06.2019

The problem:

Pick up a desired object with the robot arm after clearing the way from obstacle objects.

Program description

The program controls the robotic arm such that it moves obstacle objects from an initial board state to a target board state.
In the target state the robotic arm can grip and pick up the desired object.

Program implementation

Tools:

- ROS (Robot Operating System)
 - Gazebo
 - Manipulator-H library
 - C++
 - Python
- Nodejs
 - JavaScript

Nodejs Server and gui client:

We represented the desired object and the obstacles in a matrix.
Using the web client you can insert the initial board state, the target state and send them to the server.
The server will find the set of instructions that will bring the board from the initial state to the target state (if possible), and send them to the client.

Mov_boxes node in ROS:

The node will receive the instructions and perform each in order until the board is in the target state.
The arm will then move to the desired object, grab it and pick it up.

Main Files

Mov_box_node.cpp

Converts string instructions to robotic arm movement using the manipulator-h library.

functions:

Perform_instruction

- Perform a single instruction e.g. *(1,4),left*.

The following functions move the head in alignment to the matrix cell:

Mov_head_to_y_then_x

- Move the arm head to specific y coordinate and then x.

Mov_head_to_x_then_y

- Move the arm head to specific x coordinate and then y.

These 4 functions move the **head** one cell at a specified direction:

Mov_forward

Mov_backward

Mov_left

Mov_right

Mov_fingers

- Moves the gripper's fingers to the desired position.

waitForEndTrajectory

- Pauses execution of instructions until the previous instruction is completed.

Main.py

Receives an initial and target board states and returns a set of instructions required to transform the initial to the target state (if one exists)

Server.js

Provides a web server for the web client - using nodejs and express.

Running the code

In order to run the simulation in gazebo, run the following commands in terminal:

```
roslaunch manipulator_h_gazebo manipulator_h_gripper.launch
```

Launches the gazebo simulator with the arm and the gripper attached.

```
roslaunch manipulator_h_manager manipulator_h_manager_gazebo.launch
```

Launches the arm manipulator.

```
roslaunch mov_box mov_box_node < solution >
```

Moves the boxes according to the solution string.

For example:

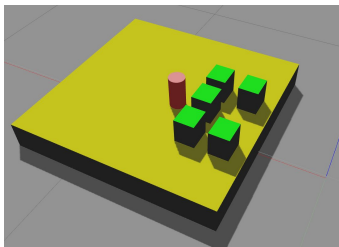
```
roslaunch mov_box mov_box_node [((1, 4), 'up', 1), ((3, 4), 'up', 1), ((2, 4), 'right', 1)]
```

```
roslaunch manipulator_h_gui manipulator_h_gui
```

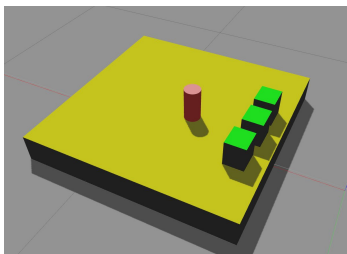
Optional - Runs the manipulator-h gui that allows controlling the arm manually.

Provided are two world configurations to use in the gazebo simulation:

Black_m_boxes1.world -



black_m_boxes2.world



Both located in the directory:

src/ROBOTIS - MANIPULATOR - H/manipulator_h_gazebo/worlds

Web client

Install nodejs and npm express and run the following command:

nodejs server.js

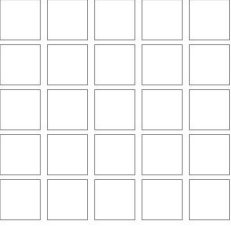
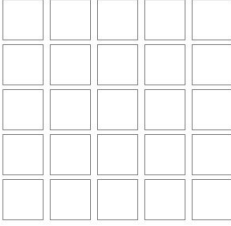
Open a web browser and enter the following link:

<http://localhost:3000/main.html>

Choose the desired board size

Enter dimention N x N

5



After choosing the desired board size, you may insert obstacles and the target objects.

Note that the robotic arm is located to the right of the board.

The green squares are obstacles.

The red square is the goal.

Choosing between green and red squares is done by clicking the switch.

Enter dimention N x N

5



Hitting the send button will return the instructions string if one exists.

Output String:

`[[(1, 4), 'left', 1), ((1, 3), 'up', 1), ((3, 4), 'right', 1), ((3, 3), 'up', 1), ((2, 3), 'left', 1)]`