

**ENGINNERING FACULTY - COMPUTER ENGINEERING DEPARTMENT**

**AUTONOMUS MOBILE ROBOTS**

**2022-2023 SPRING**

**FINAL PROJECT REPORT**

**INSTRUCTOR** : Assoc. Prof. Dr. Ahmet Çağdaş SEÇKİN

**STUDENT NAME** : Buse Latife Beker

**STUDENT ID** : 181805067

**OTHER GROUP MEMBERS** 181805057 – Kardelen Gel

## ***TABLE OF CONTENTS***

[1.1 TABLE OF CONTENTS 1](#_Toc136460552)

[2 HOUSE/MAZE PLAN 2](#_Toc136460553)

[3 METHOD 3](#_Toc136460554)

[4 SIMULATION RESULTS 12](#_Toc136460555)

# HOUSE/MAZE PLAN

**Indoor Model in the Coppeliasim Simulator**

ekran görüntüsü, kare, dikdörtgen, harita içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Mobile Robot**

**çocukların yaptığı resimler, sanat içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**2D Map of the House Plan**

**taslak, diyagram, plan, kare içeren bir resim

Açıklama otomatik olarak oluşturuldu**

# METHOD

**Python Code for Mapping**

Necessary imports

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Codes were written for robot connection. Object definitions were made using the **Vrep.simxGetObjectHandle** function. These objects can include sensors, motors, and other items.

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

First, the empty lists "px" and "py" are created. These lists will be used to store the x and y coordinates of the data received from the lidar sensor.

An empty string variable named "statuskey" is created. This variable will be used to hold the state of the key pressed from the keyboard.

A function named "key\_press" is defined. This function will be called when a key is pressed from the keyboard and will update the "statuskey" variable with the value of the pressed key. This function is passed to "keyboard.Listener" which is used to listen for keyboard input.

A function called "rotate\_vector" is defined. This function is used to rotate a given data vector by a specified angle. The function returns the incoming data vector at the specified angle and returns the returned vector.

The element "keyboard.Listener" is connected to the "key\_press" function with the "on\_press" parameter. This starts a listener to receive key input from the keyboard.

The "time.sleep(1)" command causes the program to wait for 1 second. This gives the listener some time to receive key input from the keyboard.

An empty list named "lidarmap" is created. This list will be used to store the map of the data received from the lidar sensor.

This piece of code provides the necessary constructs to process data from the lidar sensor and receive keyboard input from the user. The "rotate\_vector" function can be used to map the data received from the lidar sensor and the key input from the keyboard can be tracked via the "statuskey" variable.

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

First, the `if(activate):` statement checks the value of the `activate` variable. If `activate` is True, the rest of the code is executed. This allows the code to continue if the connection was successful.

Calling `plt.close('all')` closes all existing graphic windows.

Interactive mode is enabled by calling `plt.ion()`. This allows the graphics to be updated instantly.

An infinite loop is started (`while(1)`). This loop continues to control the robot's movement and visualize it by receiving data from the lidar sensor.

With the `vrep.simxGetObjectPosition` function, the position of the object named `hokuyo` is taken and assigned to the variable `pos`.

With the `vrep.simxGetObjectPosition` function, the position of the object named `vsensor` is taken and assigned to the variable `pos\_dir`.

The variables `dr\_x` and `dr\_y` are calculated. These variables represent the x- and y-directed distance between `vsensor` and `hokuyo` objects.

Using the `math.atan2` function, the angle (`dangle`) between `dr\_y` and `dr\_x` is calculated and converted to degrees.

The values `pos[0]` and `pos[1]` are added to the `px` and `py` lists. This stores the x and y positions of the robot in the `px` and `py` lists.

The last inserted position is printed as the last elements of the lists `px` and `py` (`px[-1]` and `py[-1]`).

If the first elements of the lists `px` and `py` are equal to zero, the first elements (`px[0]` and `py[0]`) are omitted. This clears the zero position in lists that represent the track of the moving robot.

With the `vrep.simxGetStringSignal` function, a signal named "measuredDataAtThisTime" is received and assigned to the `data` variable.

The length of the `data` list is checked. If `data` is not empty, that is, `len(data) > 0` it is entered in the if block. This check is to check whether the data from the lidar sensor is being received. If `data` is empty, that is, no data has been received, the if block is skipped and the program continues.

The current chart is cleared by calling `plt.clf()`.

Using the `vrep.simxUnpackFloats` function, the `data` variable is unpacked and assigned to the `measuredData` variable. This enables data from the lidar sensor to be analyzed.

Two new arrays named `lx` and `ly` are created. These arrays take data in x and y coordinates from the `measuredData` array.

An array named `ldata` is created. This array is formed by combining `lx` and `ly` strings.

Using the `rotate\_vector` function, the `ldata` array is rotated with a `dangle` angle and assigned to the variable `rotated\_data`.

In a `try-except` block inside the loop, each point in the `rotated\_data` array is checked. If the point is not in the `lidarmap` list, it is added to the `lidarmap` list.

The `lidarmap` list is converted to a numpy array and visualized with a scatter plot using the matplotlib library.

The `px` and `py` lists are also visualized with a scatter plot.

Using the `plt.arrow` function, an arrow is drawn indicating the direction of movement of the robot.

Graphic boundaries are set, labels are added, and a title is assigned.

The graph and axes are shown.

A wait time is added by calling `plt.pause(0.0001)`. This takes a while for the graphics to update and the interface to react.

The robot's right and left wheel speeds are assigned to the variables `wR` and `wL`. These speeds are updated as the key is pressed.

By controlling the `statuskey` variable, movement speeds are adjusted according to the key input.

The `statuskey` variable is reset.

Speed values are sent to the motors using the `vrep.simxSetJointTargetVelocity` function.

When `Key.esc` is pressed, the loop is terminated and the program is completed.

The columns of the `lmnp` array are saved in the `lmnpx.npy` and `lmnpy.npy` files.

This piece of code controls the robot's motion, receives data from the lidar sensor, visualizes the data, and allows the user to control the robot's motion with keyboard keys. The lidar data and the robot's position are displayed graphically using the matplotlib library. In this way, the user can detect obstacles around the robot and monitor the robot's movement in real time. Lidar data is added to the 'lidarmap' list and this data is displayed on the graph as a scatter plot. The robot's track is stored in the `px` and `py` lists and is indicated by a red line on the graph. The movement of the robot is controlled by keyboard input: it can be moved up, left, right or down. At the end of the code, the `lmnp` string is saved in the `lmnpx.npy` and `lmnpy.npy` files with numpy's `np.save` function. These files hold the x and y coordinates of the lidar data.metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Start (Cycle) and End (X) Points**

**taslak, diyagram, plan, kare içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Python Code for Path Planning**

**Shortest Paths Between the Points A and B**

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Start and end points, grid size and robot radius importing

**metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Loading map data

****

**Shortest Path Between the Points A and B with A\* Algorithm**

This piece of code performs a planning process using the A\* algorithm and graphically visualizes the result. The `show\_animation` variable must be set to `True` for the code to run.

First, it needs the obstacle coordinates (`ox` x coordinates,`oy` y coordinates) and cell size (`grid\_size`) and robot radius (`robot\_radius`) stored in lists named `ox` and `oy`.

An object (`a\_star`) of class `AStarPlanner` is created and initialized with the values `ox`, `vote`, `grid\_size` and `robot\_radius`.

Then, a path is made between the starting (`sx`, `sy`) and destination (`gx`, `gy`) points using the `planning` method of the `a\_star` object. As a result of the planning, the x and y coordinates of the planned path are assigned to the `rx` and `ry` lists.

Using the `interpolate.splprep` function, a smoothing operation is performed on the lists `rx` and `ry` and a function `f` representing the smoothed curve and control points `u` are returned.

Using the `np.linspace` function, 100 points are created on the flattened curve and the x and y coordinates of these points are assigned to the `xint` and `yint` lists.

Finally, using the `plt.plot` function, obstacle points (`ox`, `oy`), starting point (`sx`, `sy`), target point (`gx`, `gy`) and smoothed curve (` xint`, `yint`) are plotted on the chart.

`plt.grid(True)` displays the grid lines and `plt.axis("equal")` equalizes the scale of the x and y axes.

Finally, the graph is displayed with `plt.show()`.metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Shortest Path Between the Points A and B with Dijkstra Algorithm**

This piece of code performs a planning process using the Dijkstra algorithm and graphically visualizes the result. The `show\_animation` variable must be set to `True` for the code to run.

First, the obstacle coordinates (`ox` x coordinates, `oy` y coordinates) and cell size (`grid\_size`) and robot radius (`robot\_radius`) stored in lists named `ox` and `oy` are needed.

An object of class `dijkstra` (`dijkstra`) is created and initialized with the values `ox`, `oy`, `grid\_size` and `robot\_radius`.

Next, a path plan is made between the starting (`sx`, `sy`) and destination (`gx`, `gy`) points using the `planning` method of the `dijkstra` object. As a result of the planning, the x and y coordinates of the planned path are assigned to the `rx` and `ry` lists.

Using the `interpolate.splprep` function, a smoothing operation is performed on the lists `rx` and `ry` and a function `f` representing the smoothed curve and control points `u` are returned.

Using the `np.linspace` function, 100 points are created on the flattened curve and the x and y coordinates of these points are assigned to the `xint` and `yint` lists.

Finally, using the `plt.plot` function, obstacle points (`ox`, `oy`), starting point (`sx`, `sy`), target point (`gx`, `gy`) and smoothed curve (` xint`, `yint`) are plotted on the chart.

`plt.grid(True)` displays the grid lines and `plt.axis("equal")` equalizes the scale of the x and y axes.

Finally, the graph is displayed with `plt.show()`.

**metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Shortest Path Between the Points A and B with D\* Algorithm**

# Numbers must be bigger than 0 and they must be integer in this algorithm

This piece of code performs a planning process using the D\* Lite algorithm and graphically visualizes the result. The `show\_animation` variable must be set to `True` for the code to run.

First, a `Map` object (`m`) is created from the `d\_star\_module` module. This `Map` object represents the position of obstacles and target in a grid. When building, the grid dimensions are set to 100x100.

Next, the obstacle coordinates (`n\_ox` x coordinates, `n\_oy` y coordinates) in the `n\_ox` and `n\_oy` lists are assigned to the `ox` and `vote` lists. However, a transformation is performed on these coordinates. Each coordinate is multiplied by 10, rounded up, and added by 52. This operation moves the coordinates to a new origin.

Using the `set\_obstacle` method of the `m` object, the obstacle points are set with `[(i, j) for i, j in zip(ox, oy)]`.

The starting and target points (`sx`, `sy`) and (`gx`, `gy`) are determined.

If the variable `show\_animation` is `True`, then using the `plt.plot` function obstacle points (`ox`, `oy`), starting point (`sx`, `sy`), target point (`gx`, `gy`) `) is plotted on the chart and `plt.axis("equal")` equalizes the scale of the x and y axes.

The `Dstar` algorithm is run on the `m` object. This is the part where the D\* Lite algorithm works. The `run` method is called using the start and destination points and the x and y coordinates of the planned path are assigned to the lists `rx` and `ry`.

If the `show\_animation` variable is `True`, the planned path (`rx`, `ry`) is plotted using the `plt.plot` function and the graph is displayed with `plt.show()`.

The `rx` and `ry` lists are saved using the `np.save` function.

**metin, ekran görüntüsü, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu**

# SIMULATION RESULTS

**2D Map of the House Plan**

**taslak, diyagram, plan, kare içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Shortest Path Between the Points A and B with A\* Algorithm**

**dikdörtgen, taslak, diyagram, kare içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**diyagram, plan, harita içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Shortest Path Between the Points A and B with Dijkstra Algorithm**

****

**diyagram, dikdörtgen, plan, kare içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**Shortest Path Between the Points A and B with D\* Algorithm**

metin, ekran görüntüsü, dikdörtgen, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, diyagram, harita, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu