

5DV121
Fundamentals of Artificial Intelligence
Follow the path



Figure 1 – Robosoft Kompai Robot in Microsoft Robotics Developer Studio 4

Contents

1	Introduction	1
2	Path Tracking Algorithm	1
2.1	Essential Functions	1
2.2	Follow The Carrot:ish	2
3	Controller Program	2
3.1	Prerequisites	2
3.1.1	Input	2
3.1.2	Arguments	2
3.2	Instructions	3
3.3	Output	4
3.3.1	Example Run	4
4	Discussion	4
	Attachments	6

This task is about gaining in-depth knowledge within the basics of artificial intelligence by writing a controller program that can guide a robot (see Figure 1) in a simulated environment (see Figure 2), such that it follows a pre-specified path. The Robot must follow the given path within 5 minutes.



2 Path Tracking Algorithm

We used our own version of the "Follow the carrot" algorithm (see section 2.2). First we get the position of our robot and the laser containing the surroundings of our robot. Then we calculate the speed, rotationspeed and look ahead distance that should be set next based on the robots surroundings. Then we check which nodes we have passed on the path to see which is closest and returns the value. Then we sets our next goal on the path by using our look ahead distance. Afterwards we get the rotation direction. Then we set the speed and rotation direction and rotation speed.

get_passed_nodes(): Checks the how many indexes of the path that the robot has passed and returns the new index. The index is used later on to set a new look ahead distance if it has changed.

`get_next_goal()`: checks where the furthest index of the path is that is still within the look ahead distance and returns the index.

set_heading: Using trigonometrics this functions calculates the robots angle and the angle between the robot and its goal. It then makes them both positive if necessary and then returns the result depending on the following. If the robots angle is less then 0.2 radians differentiated from the goal then it returns that the robot shouldn't rotate. Else if checks if the robots angle is more than the destinations angle and if the difference is also larger than pi it returns -1 else 1. If the robot angle is

less than the destinations angle and it is also larger than pi it returns 1 else -1. The return value can later be used to set the way of rotation for the robot.

2.2 Follow The Carrot:ish

```
1 pos = get_pos()
2 laser = get_laser()
3 [speed, look_dist, rot] = get_speed(laser)
4
5 index = get_passed_nodes()
6 index = get_next_goal(path, ind, pos, look_dist)
7 direction = set_heading(path[ind], pos)
8
9 post_speed(direction*rot, speed)
```

Figure 3 – Psuedo Code Of Our Algorithm

3 Controller Program

The Controller Program is written in Python 3 and communicates with a Robosoft Kompai robot in the simulated environment (see Figure 2) through a HTTP interface.

3.1 Prerequisites

To run the controller program you must have a computer running on Windows and that have Python 3 and Microsoft Robotics Developer Studio 4 installed. To see the full soucre code see Attachment 1.

3.1.1 Input

- robot.py
- *.json

3.1.2 Arguments

The program takes a argument in form of a JSON file (see section 3.1.1) which contains the path for the robot to follow. The path is given as a sequence of coordinates. Example of the path is given in Figure 4.

```
1  [
2    {
3      "Pose":{
4        "Orientation":{
5          "W":0.9999483227729741,
6          "X":-0.0000012598708510098447,
7          "Y":4.0935409816933006e-8,
8          "Z":0.010499421505498067
9        },
10       "Position":{
11         "X":-0.003833770751953125,
12         "Y":-0.007822131738066673,
13         "Z":0.07760075479745865
14       }
15     },
16     "Status":4,
17     "Timestamp":21523
18   },
19   {
20     "Pose":{
21       "Orientation":{
22         "W":0.9999484419822693,
23         "X":-0.0000012508626241469756,
24         "Y":4.456115831885654e-8,
25         "Z":0.010499733500182629
26       },
27       "Position":{
28         "X":-0.003833770751953125,
29         "Y":-0.00782213918864727,
30         "Z":0.07760075479745865
31       }
32     },
33     "Status":4,
34     "Timestamp":21757
35   }
36 ]
```

Figure 4 – Path as formatted JSON string.

3.2 Instructions

If you have fulfilled the prerequisites in section 3.1 follow these instructions to tun the Controller Program.

1. Open the program "Run".
2. Start "Microsoft Robotics Developer Studio 4" by executing:
C:/MRDS4/store/launchers/StartLokarria.bat
3. Open the "Command Prompt" and navigate to the folder which contains the Controller Program.
4. Run the Controller Program by executing the following lines:
python robot.py path.json

3.3 Output

The Controller Program gives the user information about when the Robot starts the lap, finishes the lap and if something went wrong along the way. See Figure 5 for a example run of the Controller Program.

To interpret the output you can use the guide below:

- **Starting robot**
The Robot has started the lap.
- **Running**
The Robot is currently following the path.
- **Finished! Lap time: 44.6851s**
The Robot has reached the goal in the time given.
- **Unexpected response from server when sending speed commands: "Exception"**
The Robot could not change speed since the request to the HTTP interface gave a bad response.
- **Unexpected response from server when reading position: "Exception"**
The Robot could not get its current position since the request to the HTTP interface gave a bad response.
- **Unexpected response from server when reading laser data: "Exception"**
The Robot could not get data from the laser sensor since the request to the HTTP interface gave a bad response.

3.3.1 Example Run

```
1 Starting robot
2 Running
3 Finished! Lap time: 44.6851s
```

Figure 5 – Successful Example Run

4 Discussion

Overall the assignment went well. We had good communication within the group which helped us to organize and delegate work between us. Most of the time we worked together on the same computer and sometimes we worked on distance via Skype. We faced some trouble since we both don't have Windows as our main operating systems, but we solved it and gained good knowledge along the way.

We choose to work in Python. A language we never used in the group. So there was a bit of learning curve before we could start programming for real. During the programming we faced some issues when we were developing the function for turning the robot. We had problems to make the robot decide if it should rotate left or right depending on its current heading and next point in the path. We solved it by going through lecture slides and drawing up simple scenarios which we could apply math and logic

thinking to.

We also had some problems on how to optimize the combination of speed, rotation and obstacle avoidance to make the robot move as fast as possible around the giving path. Since we ran out of time we decided to focus on safety instead of getting the best time.

Bilagor

Attachment 1 Source Code

```
1 # robot.py
2 # Authors: Emil Hallberg & Jonas Sjödin
3 # Date: September 2018
4 # Course: Fundamentals of Artificial Intelligence
5 #
6 # Description: This python program controls a lokarria robot by sending
7 # different http-requests through its interface. It takes a path as an argument
8 # and follows the path with a look ahead distance using an interpretation of
9 # the "Follow the carrot algorithm". It records its lap time and prints it on finish.
10
11
12 import http.client, json, time, sys
13 from math import pi, atan2, sqrt, pow
14
15 MRDS_URL = 'localhost:50000'
16 HEADERS = {"Content-type": "application/json", "Accept": "text/json"}
17
18
19 class UnexpectedResponse(Exception):
20     pass
21
22
23 def get_laser():
24     """
25     returns the laser of the robot at this moment
26     :return: the laser of the robot at this moment
27     """
28     mrds = http.client.HTTPConnection(MRDS_URL)
29     mrds.request('GET', '/lokarria/laser/echoes')
30     response = mrds.getresponse()
31     if response.status == 200:
32         laser_data = response.read()
33         response.close()
34         return json.loads(laser_data.decode())
35     else:
36         return response
37
38
39 def rotate(q, v):
40     """
41     rotation taking a q and a v
42     :param q: the q
43     :param v: the v
44     :return: the rotation
45     """
46     return vector(qmult(qmult(q, quaternion(v)), conjugate(q)))
47
48
49 def quaternion(v):
50     """
51     Returns the quaternion of v
52     :param v: the v that should become a quaternion
53     :return: the quaternion of v
```



```
54     """
55     q = v.copy()
56     q['W'] = 0.0
57     return q
58
59
60 def vector(q):
61     """
62     returns a dict version of q
63     :param q: the q
64     :return: dict version of q
65     """
66     return {"X": q["X"], "Y": q["Y"], "Z": q["Z"]}
67
68
69 def conjugate(q):
70     """
71     returns the conjugate of q
72     :param q: the q
73     :return: the q copy
74     """
75     qc = q.copy()
76     qc["X"] = -q["X"]
77     qc["Y"] = -q["Y"]
78     qc["Z"] = -q["Z"]
79     return qc
80
81
82 def qmult(q1, q2):
83     """
84     Returns the qmult
85     :param q1: value 1
86     :param q2: value 2
87     :return: the qmult
88     """
89     return {
90         "W": q1["W"] * q2["W"] - q1["X"] * q2["X"] - q1["Y"] * q2["Y"] - q1["Z"] * q2["Z"],
91         "X": q1["W"] * q2["X"] + q1["X"] * q2["W"] + q1["Y"] * q2["Z"] - q1["Z"] * q2["Y"],
92         "Y": q1["W"] * q2["Y"] - q1["X"] * q2["Z"] + q1["Y"] * q2["W"] + q1["Z"] * q2["X"],
93         "Z": q1["W"] * q2["Z"] + q1["X"] * q2["Y"] - q1["Y"] * q2["X"] + q1["Z"] * q2["W"]
94     }
95
96
97 def get_heading():
98     """Returns the XY Orientation as a heading unit vector"""
99     return rotate(get_pose()['Pose']['Orientation'], {'X': 1.0, 'Y': 0.0, 'Z': 0.0})
100
101
102 class Path:
103     """
104     Initializes the Path class and reads a path file
105     """
106
107     def __init__(self):
108         self.path = []
109         self.load_path(sys.argv[1])
110         self.vecPath = self.vectorize_path()
111
```

```
112     def load_path(self, file_name):
113         """
114         Loads the path taken as an argument and saves it as json data
115         :param file_name: the file that should be read.
116         :return: the path as json data
117         """
118         with open(file_name) as path_file:
119             data = json.load(path_file)
120             self.path = data
121
122     def vectorize_path(self):
123         """
124         Makes the path into a vector
125         :return: the path as an vector
126         """
127         return [{'X': p['Pose']['Position']['X'],
128                 'Y': p['Pose']['Position']['Y'],
129                 'Z': p['Pose']['Position']['Z']}
130                 for p in self.path]
131
132
133 def post_speed(angular_speed, linear_speed):
134     """
135     Sets the speed of the robot by taking an angular speed and a linear speed
136     as arguments
137     :param angular_speed: the rotational speed
138     :param linear_speed: the speed in the robots direction
139     :return: the response status of the robot
140     """
141     mrds = http.client.HTTPConnection(MRDS_URL)
142     params = json.dumps({'TargetAngularSpeed': angular_speed, 'TargetLinearSpeed': linear_speed})
143     mrds.request('POST', '/lokarria/differentialdrive', params, HEADERS)
144     response = mrds.getresponse()
145     status = response.status
146     if status == 204:
147         return response
148     else:
149         raise UnexpectedResponse(response)
150
151
152 def get_pose():
153     """
154     Gets the pose of the robot by sending a http request
155     :return: the pose if eveything went right
156     """
157     mrds = http.client.HTTPConnection(MRDS_URL)
158     mrds.request('GET', '/lokarria/localization')
159     response = mrds.getresponse()
160     if response.status == 200:
161         pose_data = response.read()
162         response.close()
163         return json.loads(pose_data.decode())
164     else:
165         return UnexpectedResponse(response)
166
167
168 def get_dist(robot_t, goal):
169     """
```

```
170     Returns the distance between two positions
171     :param robot_t: The robots position
172     :param goal: The goal position
173     :return: the distandce between the two positions
174     """
175     return sqrt(pow(robot_t['X'] - goal['X'], 2) + pow(robot_t['Y'] - goal['Y'], 2))
176
177
178 def get_next_goal(path_t, ind_t, robot_t, look_dist_t):
179     """
180     Gets the next goal on the given path that the robot should aim for
181     :param path_t: The given path that the robot should follow
182     :param ind_t: The current index that the robot last looked for
183     :param robot_t: The robots position
184     :param look_dist_t: the look ahead distance of the robot
185     :return: index of the position the robot should look at
186     """
187     try:
188         while get_dist(path_t[ind_t + 1], robot_t) < look_dist_t:
189             ind_t += 1
190     except IndexError:
191         return ind_t - 1
192
193     return ind_t
194
195
196 def set_heading(goal, robot_t):
197     """
198     Sets the heading direction of the robot
199     :param goal: the goal position in this state
200     :param robot_t: the robots position in this state
201     :return: Returns if the robot should rotate to the right or to the left
202     """
203     robot_head = get_heading()
204     robot_angle = -1 * atan2(robot_head['Y'], robot_head['X'])
205
206     dest_degree = -1 * atan2(goal['Y'] - robot_t['Y'], goal['X'] - robot_t['X'])
207
208     if dest_degree < 0:
209         dest_degree += 2 * pi
210     if robot_angle < 0:
211         robot_angle += 2 * pi
212
213     if abs(robot_angle - dest_degree) > 0.2:
214         if robot_angle > dest_degree:
215             if abs(robot_angle - dest_degree) > pi:
216                 return -1
217             return 1
218         if abs(robot_angle - dest_degree) > pi:
219             return 1
220         return -1
221     return 0
222
223
224 def set_speed(laser_t):
225     """
226     Sets the next speed, look ahead distance and rotation speed.
227     :param laser_t: the given laser containing info about the surroundings
```

```
228         :return: speed, look ahead distance and rotation speed
229         """
230         smallest = min(laser.t['Echoes'][95:270 - 95])
231         if smallest < 0.3:
232             return [0.1, 0.4, 1]
233         if smallest < 0.4:
234             return [0.3, 0.6, 1]
235         if smallest < 0.5:
236             return [0.5, 0.6, 1]
237         if smallest < 0.6:
238             return [0.7, 0.6, 1]
239         if smallest < 0.7:
240             return [0.8, 0.6, 1]
241         if smallest < 0.8:
242             return [1, 0.6, 1]
243         if smallest < 2:
244             return [1, 1, 1]
245         return [1, 2, 0.8]
246
247
248 def get_passed_nodes(robot.t, path.t, ind.t):
249     """
250     Returns the index of the last position in the path that the robot has passed
251     :param robot.t: The robots position containig a x-pos and a y-pos
252     :param path.t: The given path
253     :param ind.t: The last passed index of the path
254     :return: the index of the last position passed.
255     """
256
257     nearest = get_dist(robot.t, path.t[ind.t])
258     near_ind = ind.t
259
260     for i in range(ind.t, len(path.t)):
261         dist2 = get_dist(robot.t, path.t[i])
262         if dist2 < nearest:
263             nearest = dist2
264             near_ind = i
265         elif ind.t + 400 < i:
266             break
267     return near_ind
268
269
270 if __name__ == '__main__':
271     """
272     Main function for the robot controller program, which runs everything.
273     """
274     print("Starting robot")
275     path = list(Path().vecPath)
276     ind = 0
277     look_dist = 1.4
278     robot = dict
279     laser = dict
280     print("Running")
281     t0 = time.time()
282     while 1:
283
284         try:
285             robot = get_pose()['Pose']['Position']
```

```
286     except UnexpectedResponse as ex:
287         print('Unexpected response from server when reading position:', ex)
288
289     try:
290         laser = get_laser()
291     except UnexpectedResponse as ex:
292         print('Unexpected response from server when reading laser data:', ex)
293
294     [speed, look_dist, rot] = set_speed(laser)
295
296     if get_dist(robot, path[len(path) - 1]) < 0.2:
297         try:
298             post_speed(0, 0)
299         except UnexpectedResponse as ex:
300             print('Unexpected response from server when sending speed commands:', ex)
301             break
302
303     ind = get_passed_nodes(robot, path, ind)
304     if ind == -1:
305         break
306     ind = get_next_goal(path, ind, robot, look_dist)
307
308     direction = set_heading(path[ind], robot)
309
310     try:
311         post_speed(direction * rot, speed)
312     except UnexpectedResponse as ex:
313         print('Unexpected response from server when sending speed commands:', ex)
314
315     time.sleep(0.05)
316     t1 = time.time()
317     print("Finished! Lap time:", "{:.4f}s".format(t1 - t0))
```