Department of Computer Science
Umeå Universitet

October 8, 2018

5DV121
Fundamentals of Artificial Intelligence
Follow the path

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

Teachers
Carl-Anton Anserud, Anders Broberg
Adam Dahlgren Lindström, Monika Jingar
Thomas Johansson, Jenny Nilsson, Ola Ringdahl

Emil Hallberg id16ehg@cs.umu.se
Jonas Sjödin id16jsn@cs.umu.se

Contents

1 Introduction **1**

2 Path Tracking Algorithm **1**
2.1 Essential Functions 1
2.2 Follow The Carrot:ish 2

3 Controller Program **3**
3.1 Prerequisites 3
3.1.1 Input 3
3.1.2 Arguments 3
3.2 Instructions 4
3.3 Output 5
3.3.1 Example Run 5

4 Discussion **5**

Attachments **7**

1 Introduction

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.

Figure 2 – Simulated Environment in Microsoft Robotics Developer Studio 4

2 Path Tracking Algorithm

We used our own version of the "Follow the carrot" algorithm (see Figure 3). First we gets 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.

2.1 Essential Functions

`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 trigonometics 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. See Figure 4.

2.2 Follow The Carrot:ish

```
pos = get_pos()
laser = get_laser()
[speed, look_dist, rot] = get_speed(laser)

index = get_passed_nodes()
index = get_next_goal(path, ind, pos, look_dist)
direction = set_heading(path[ind], pos)

post_speed(direction*rot, speed)
```

Figure 3 – Psuedo Code Of Our Follow The Carrot:ish

```
//Takes a goal position and the robots position and sets which way the robot
//should rotate or if it should not rotate at all.
function set_heading(goal_pos, robot_pos) {
    //Get the heading-direction of the robot
    robot_head = get_heading()

    //Calculate angle between robot x and y and robot and destination
    robot_angle = -1 * arctan(robot_head.getY(), robot_head.getX())
    dest_angle = -1 * arctan(goal_pos.getY() - robot_pos.getY(), goal_pos.getX() - robot_pos.getX())

    //If degree is negative, make it positive
    if (dest_degree < 0)
        dest_degree += 2 * pi
    if (robot_angle < 0)
        robot_angle += 2 * pi

    //Calculate rotation way with a wrong angle acceptance of 0.2/(pi*2) which is around 3% wrong angle.
    If (abs(robot_angle - dest_angle) > 0.2)
        if (robot_angle > dest_degree)
            if (abs(robot_angle - dest_degree) > pi)
                return -1
            return 1
        if (abs(robot_angle - dest_degree) > pi)
            return 1
        return -1
    return 0
}
```

Figure 4 – Psuedo Code Of Our Main Algorithm Set Heading

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 5.

```
[
   {
      "Pose":{
         "Orientation":{
            "W":0.9999483227729741,
            "X":-0.0000012598708510098447,
            "Y":4.0935409816933006e-8,
            "Z":0.010499421505498067
         },
         "Position":{
            "X":-0.003833770751953125,
            "Y":-0.00782213173806673,
            "Z":0.07760075479745865
         }
      },
      "Status":4,
      "Timestamp":21523
   },
   {
      "Pose":{
         "Orientation":{
            "W":0.9999484419822693,
            "X":-0.0000012508626241469756,
            "Y":4.456115831885654e-8,
            "Z":0.010499733500182629
         },
         "Position":{
            "X":-0.003833770751953125,
            "Y":-0.00782213918864727,
            "Z":0.07760075479745865
         }
      },
      "Status":4,
      "Timestamp":21757
   }
]
```


Figure 5 – 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 6 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

```
Starting robot
Running
Finished! Lap time: 44.6851s
```

Figure 6 – 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 togheter 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 depedning on its current heading and next point int the path. We solved it by going through lecture slides and drawing up simple scenarios which we could apply math an logic thinking to.

We also had some problems on how to optimize the combination of speed, rotation and obstacle avoidence 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

```
# robot.py
# Authors: Emil Hallberg & Jonas Sj din
# Date: September 2018
# Course: Fundamentals of Artificial Intelligence
#
# Description: This python program controlls a lokarria robot by sending
# different http-requests through its interface. It takes a path as an argument
# and follows the path with a look ahead distance using an interpretation of
# the "Follow the carrot algorithm". It records its lap time and prints it on finish.


import http.client, json, time, sys
from math import pi, atan2, sqrt, pow

MRDS_URL = 'localhost:50000'
HEADERS = {"Content-type": "application/json", "Accept": "text/json"}


class UnexpectedResponse(Exception):
    pass


def get_laser():
    """
    returns the laser of the robot at this moment
    :return: the laser of the robot at this moment
    """
    mrds = http.client.HTTPConnection(MRDS_URL)
    mrds.request('GET', '/lokarria/laser/echoes')
    response = mrds.getresponse()
    if response.status == 200:
        laser_data = response.read()
        response.close()
        return json.loads(laser_data.decode())
    else:
        return response


def rotate(q, v):
    """
    rotation taking a q and a v
    :param q: the q
    :param v: the v
    :return: the rotation
    """
    return vector(qmult(qmult(q, quaternion(v)), conjugate(q)))


def quaternion(v):
    """
    Returns the quaternion of v
    :param v: the v that should become a queaternion
    :return: the quaternion of v
```

```python
    """
    q = v.copy()
    q['W'] = 0.0
    return q


def vector(q):
    """
    returns a dict version of q
    :param q: the q
    :return: dict version of q
    """
    return {"X": q["X"], "Y": q["Y"], "Z": q["Z"]}


def conjugate(q):
    """
    returns the conjugate of q
    :param q: the q
    :return: the q copy
    """
    qc = q.copy()
    qc["X"] = -q["X"]
    qc["Y"] = -q["Y"]
    qc["Z"] = -q["Z"]
    return qc


def qmult(q1, q2):
    """
    Returns the qmult
    :param q1: value 1
    :param q2: value 2
    :return: the qmult
    """
    return {
        "W": q1["W"] * q2["W"] - q1["X"] * q2["X"] - q1["Y"] * q2["Y"] - q1["Z"] * q2["Z"],
        "X": q1["W"] * q2["X"] + q1["X"] * q2["W"] + q1["Y"] * q2["Z"] - q1["Z"] * q2["Y"],
        "Y": q1["W"] * q2["Y"] - q1["X"] * q2["Z"] + q1["Y"] * q2["W"] + q1["Z"] * q2["X"],
        "Z": q1["W"] * q2["Z"] + q1["X"] * q2["Y"] - q1["Y"] * q2["X"] + q1["Z"] * q2["W"]
    }


def get_heading():
    """Returns the XY Orientation as a heading unit vector"""
    return rotate(get_pose()['Pose']['Orientation'], {'X': 1.0, 'Y': 0.0, "Z": 0.0})


class Path:
    """
    Initializes the Path class and reads a path file
    """

    def __init__(self):
        self.path = []
        self.load_path(sys.argv[1])
        self.vecPath = self.vectorize_path()

```

```
    def load_path(self, file_name):
        """
        Loads the path taken as an argument and saves it as json data
        :param file_name: the file that should be read.
        :return: the path as json data
        """
        with open(file_name) as path_file:
            data = json.load(path_file)
        self.path = data

    def vectorize_path(self):
        """
        Makes the path into a vector
        :return: the path as an vector
        """
        return [{'X': p['Pose']['Position']['X'],
                 'Y': p['Pose']['Position']['Y'],
                 'Z': p['Pose']['Position']['Z']}
                for p in self.path]


def post_speed(angular_speed, linear_speed):
    """
    Sets the speed of the robot by taking an angular speed and a linear speed
    as arguments
    :param angular_speed: the rotational speed
    :param linear_speed: the speed in the robots direction
    :return: the response status of the robot
    """
    mrds = http.client.HTTPConnection(MRDS_URL)
    params = json.dumps({'TargetAngularSpeed': angular_speed, 'TargetLinearSpeed': linear_speed})
    mrds.request('POST', '/lokarria/differentialdrive', params, HEADERS)
    response = mrds.getresponse()
    status = response.status
    if status == 204:
        return response
    else:
        raise UnexpectedResponse(response)


def get_pose():
    """
    Gets the pose of the robot by sending a http request
    :return: the pose if eveything went right
    """
    mrds = http.client.HTTPConnection(MRDS_URL)
    mrds.request('GET', '/lokarria/localization')
    response = mrds.getresponse()
    if response.status == 200:
        pose_data = response.read()
        response.close()
        return json.loads(pose_data.decode())
    else:
        return UnexpectedResponse(response)


def get_dist(robot_t, goal):
    """
```

```
    Returns the distance between two positions
    :param robot_t: The robots position
    :param goal: The goal position
    :return: the distandce between the two positions
    """
    return sqrt(pow(robot_t['X'] - goal['X'], 2) + pow(robot_t['Y'] - goal['Y'], 2))


def get_next_goal(path_t, ind_t, robot_t, look_dist_t):
    """
    Gets the next goal on the given path that the robot should aim for
    :param path_t: The given path that the robot should follow
    :param ind_t: The current index that the robot last looked for
    :param robot_t: The robots position
    :param look_dist_t: the look ahead distance of the robot
    :return: index of the position the robot should look at
    """
    try:
        while get_dist(path_t[ind_t + 1], robot_t) < look_dist_t:
            ind_t += 1
    except IndexError:
        return ind_t - 1

    return ind_t


def set_heading(goal, robot_t):
    """
    Sets the heading direction of the robot
    :param goal: the goal position in this state
    :param robot_t: the robots position in this state
    :return: Returns if the robot should rotate to the right or to the left
    """
    robot_head = get_heading()
    robot_angle = -1 * atan2(robot_head['Y'], robot_head['X'])

    dest_degree = -1 * atan2(goal['Y'] - robot_t['Y'], goal['X'] - robot_t['X'])

    if dest_degree < 0:
        dest_degree += 2 * pi
    if robot_angle < 0:
        robot_angle += 2 * pi

    if abs(robot_angle - dest_degree) > 0.2:
        if robot_angle > dest_degree:
            if abs(robot_angle - dest_degree) > pi:
                return -1
            return 1
        if abs(robot_angle - dest_degree) > pi:
            return 1
        return -1
    return 0


def set_speed(laser_t):
    """
    Sets the next speed, look ahead distance and rotation speed.
    :param laser_t: the given laser containing info about the surroundings
```

```
    :return: speed, look ahead distance and rotation speed
    """
    smallest = min(laser_t['Echoes'][95:270 - 95])
    if smallest < 0.3:
        return [0.1, 0.4, 1]
    if smallest < 0.4:
        return [0.3, 0.6, 1]
    if smallest < 0.5:
        return [0.5, 0.6, 1]
    if smallest < 0.6:
        return [0.7, 0.6, 1]
    if smallest < 0.7:
        return [0.8, 0.6, 1]
    if smallest < 0.8:
        return [1, 0.6, 1]
    if smallest < 2:
        return [1, 1, 1]
    return [1, 2, 0.8]


def get_passed_nodes(robot_t, path_t, ind_t):
    """
    Returns the index of the last position in the path that the robot has passed
    :param robot_t: The robots position containig a x-pos and a y-pos
    :param path_t: The given path
    :param ind_t: The last passed index of the path
    :return: the index of the last position passed.
    """

    nearest = get_dist(robot_t, path_t[ind_t])
    near_ind = ind_t

    for i in range(ind_t, len(path_t)):
        dist2 = get_dist(robot_t, path_t[ind_t])
        if dist2 > nearest:
            nearest = dist2
            near_ind = i
        elif ind_t + 400 < i:
            break
    return near_ind


if __name__ == '__main__':
    """
    Main function for the robot controller program, which runs everything.
    """
    print("Starting robot")
    path = list(Path().vecPath)
    ind = 0
    look_dist = 1.4
    robot = dict
    laser = dict
    print("Running")
    t0 = time.time()
    while 1:

        try:
            robot = get_pose()['Pose']['Position']
```

```
        except UnexpectedResponse as ex:
            print('Unexpected response from server when reading position:', ex)

        try:
            laser = get_laser()
        except UnexpectedResponse as ex:
            print('Unexpected response from server when reading laser data:', ex)

        [speed, look_dist, rot] = set_speed(laser)

        if get_dist(robot, path[len(path) - 1]) < 0.2:
            try:
                post_speed(0, 0)
            except UnexpectedResponse as ex:
                print('Unexpected response from server when sending speed commands:', ex)
            break

        ind = get_passed_nodes(robot, path, ind)
        if ind == -1:
            break
        ind = get_next_goal(path, ind, robot, look_dist)

        direction = set_heading(path[ind], robot)

        try:
            post_speed(direction * rot, speed)
        except UnexpectedResponse as ex:
            print('Unexpected response from server when sending speed commands:', ex)

        time.sleep(0.05)
    t1 = time.time()
    print("Finished! Lap time:", "{:.4f}s".format(t1 - t0))
```