UMEÅ UNIVERSITET
Institutionen för Datavetenskap
Rapport obligatorisk uppgift

Fundamentals of Artificial Intelligence

5DV121

Obligatorisk uppgift nr

1

Namn	Amine Balta, Martin Willman
E-post	id14aba@cs.umu.se, id14mwn@cs.umu.se
CAS	amba0020 mawi0214
Datum	18-09-2017
Lärare, Handledare	Thomas Johansson, Lennart Jern och Jonathan Westin

Inledning

Uppgiftens syfte är att skriva ett program som kan guida en robot att följa en specifik bana i en simulerad miljö. Banan är given i en sekvens med koordinater och programmet ska använda sig av robotens sensorer, alltså dennes position och riktning för att kunna gå banan.

Uppgift

Ett program ska implementeras som kan interagera med Kompai roboten i MRDS, funktioner som kan läsa robotens sensorer och även kontrollera robotens manövrar. Den ska som nämnt följa en given väg i form av koordinater, om roboten går ifrån sin bana måste den kunna hitta tillbaka. Den ska följa banan utan att slå in i en vägg eller köra in i andra hinder. Tiden bör även tas från att roboten startar till att den är klar med banan. Algoritmen som programmet följer bör vara *Follow the carrot* eller *Pure Pursui*t, denna uppgiftet löstes med follow the carrot algoritmen.

Material

De program som använts är Microsoft Robotics Developer Studio för robot miljön och Wing Python IDE, programmet är skrivet i Python 3. När laborationen skrevs så använde vi oss av exempelkoden, *lokarriaexample3.py*, som fanns under kursmaterial.

Programbeskrivning

En robot beskrivs med hjälp av sina sex frihetsgrader(DOF) som är positionen vilket är definierad av x, y och z. Samt inställningen som är definierad av rullning, lutning och girning(heading i detta fall), alltså phi, theta och psi. Rullning, lutning och girning betecknas med Eulers vinklar. Roboten har sitt egna koordinatsystem, och positionen beskrivs med hjälp av relationen mellan robotens och världens koordinatsystem. I denna laboration har det antagits att roboten rör sig i en platt 2D värld.

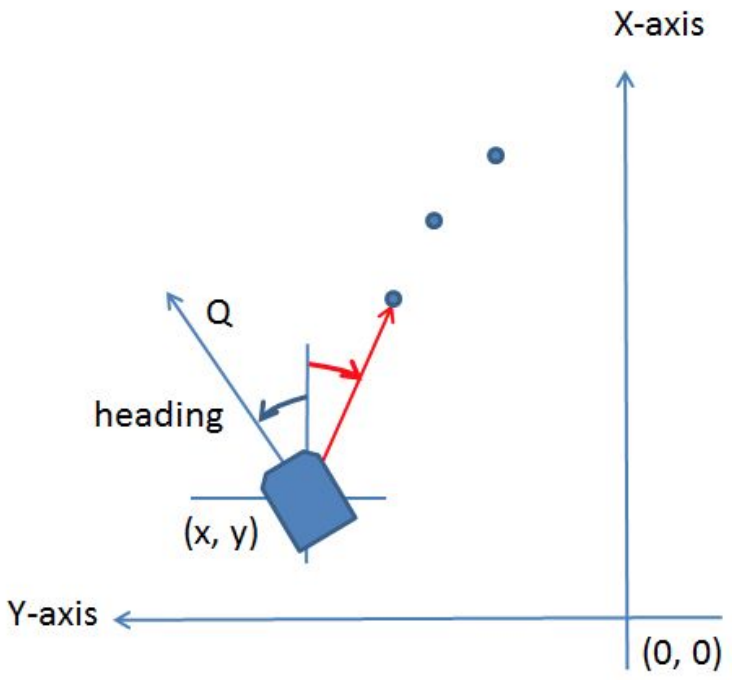
Follow the carrot algoritm

För att följa vägen har Follow the carrot algoritmen implementerats. Filosofin är att för varje "steg" roboten tar mot en punkt ska styrningen vara riktad mot en punkt längre bort i banan utan att bekymra sig om hur resten av banan ser ut.

För att beräkna vägen som ska köras, implementeras först funktioner för att beräkna avstånd mellan två punkter med hjälp av Pythagoras sats, vinkel mellan robotens position och "carrot point"(detta benämns som bearing), vinkeln mellan robotens riktning mot norr(detta benämns

som heading). För att beräkna en vinkel mellan två riktningar används inversen av tangens. Sedan räknas skillnaden mellan dessa vinklar vilket ger ett error, det är den som bestämmer vilket håll roboten ska styra mot. För att undvika att roboten tar extra varv när den har en punkt som är över 180 grader eller under -180 läggs sätts ett villkor för hur stort roboten kan svänga. En funktion för att hämta banan skapas, samt lägger in alla koordinater i en lista.

När programmet sedan körs så skapas denna lista, man tilldelar en hastighet, en vinkelhastighet och en look-a-head. Look-a-head parametern är något som kan modifieras beroende på hur långt man vill att roboten ska titta. Är den för stor kan det resultera i att roboten har svårt att stanna i banan, är det för liten tar det längre tid för den att avsluta banan. För denna lösning är det rekommenderade look-a-head värdet ungefär runt 1. När alla parametrar är satta behövs ett villkor som säger att så länge det finns en bana så ska en position, riktning och "carrot point" hämtas, om nästa position finns hämtas avståndet, vinkeln till "carrot point"(bearing) samt robotens vinkel(heading). Med dessa beräknas error eller felmarginalen mellan bearing och heading, därefter kör roboten med vinkel för error med satta vinkelhastigheten och den vanliga satta hastigheten.



Figur 1. Robotens vinklar

I figur 1 visas relationen mellan robotens riktning och positionen till nästa punkt.

Köra programmet

För att köra programmet bör man ha ett program som stödjer körning av Python3, programmet förväntar sig även att man en MRDS körs, samt en localhost: 50000 webbserver som körs. Det senare kan ändras genom att ge ett nya argument för IP och port. Man behöver även ha en sökväg till en vägfil. Programmet kommer inte köras utan dessa argument. Nedan visas hur en körning med en given bana kan se ut.

```
Angle between head and nextpos 0.9749230186450104
This is nex point: {'X': 0.4779653549194336, 'Y': 0.5894299745559692,
Distance to next point 1.00776225422938
This is heading: -0.9998480296327382
Angle between head and nextpos 0.8549954298294375
This is nex point: {'X': 0.4779653549194336, 'Y': 0.5894299745559692,
Distance to next point 1.0765388147399513
This is heading: -0.9924104162331937
Angle between head and nextpos 0.8166011323661908
This is nex point: {'X': 0.4443182945251465, 'Y': 0.48953506350517273,
Distance to next point 1.0096419609037781
This is heading: -0.9531048788741019
Angle between head and nextpos 0.6944918570695817
This is nex point: {'X': 0.4443182945251465, 'Y': 0.48953506350517273,
Runtime was: 52.25755476951599
```

Figur 2. En körning med roboten.

Diskussion

Att göra denna uppgift gick rätt smidigt, det hjälpte oss mycket att vi började med uppgiften i tid då det tog lång tid för oss att förstå hur vi skulle göra. De största problemen vi hade var förmodligen att ta steget mellan beräkningarna och implementationen. Sedan var att förstår sambandet mellan heading och bearing något som tog tid, vi fick tillsut rätt på det och då fungerade roboten. Eftersom att vi inte tidigare programmerat i Python så var det en sak att vänja sig vid. Annars har det gått bra att arbeta med detta laboration och vi är nöjda med resultatet.

Bilagor

Fakta:
https://www.cambro.umu.se/access/content/group/57205HT17-1/Lecture%20slides/Path%20tracking.pdf

Exempelkod:
https://www8.cs.umu.se/kurser/5DV121/HT17/utdelat/MRDS-Assignment1/lokarriaexample3.py

Källkod

```
#!/usr/bin/env python3
"""
Example demonstrating how to communicate with Microsoft Robotic
Developer
Studio 4 via the Lokarria http interface.

Author: Erik Billing (billing@cs.umu.se)

Updated by Ola Ringdahl 2014-09-11
Updated by Lennart Jern 2016-09-06 (converted to Python 3)
Updated by Filip Allberg and Daniel Harr 2017-08-30 (actually
converted to Python 3)
Modifyed by Martin Willma(id14mwn) and Amine Balta(id14aba)
2017-09-27.
"""

MRDS_URL = 'localhost:50000'

import http.client, json, time, sys
from math import *

HEADERS = {"Content-type": "application/json", "Accept":
"text/json"}

class UnexpectedResponse(Exception): pass

def postSpeed(angularSpeed,linearSpeed):
    """Sends a speed command to the MRDS server"""
    mrds = http.client.HTTPConnection(MRDS_URL)
    params =
json.dumps({'TargetAngularSpeed':angularSpeed,'TargetLinearSpeed':
linearSpeed})
```

```
mrds.request('POST','/lokarria/differentialdrive',params,HEADERS)
    response = mrds.getresponse()
    status = response.status
    #response.close()
    if status == 204:
        return response
    else:
        raise UnexpectedResponse(response)

def getLaser():
    """Requests the current laser scan from the MRDS server and
parses it into a dict"""
    mrds = http.client.HTTPConnection(MRDS_URL)
    mrds.request('GET','/lokarria/laser/echoes')
    response = mrds.getresponse()
    if (response.status == 200):
        laserData = response.read()
        response.close()
        return json.loads(laserData.decode())
    else:
        return response

def getLaserAngles():
    """Requests the current laser properties from the MRDS server
and parses it into a dict"""
    mrds = http.client.HTTPConnection(MRDS_URL)
    mrds.request('GET','/lokarria/laser/properties')
    response = mrds.getresponse()
    if (response.status == 200):
        laserData = response.read()
        response.close()
        properties = json.loads(laserData.decode())
        beamCount =
int((properties['EndAngle']-properties['StartAngle'])/properties['
AngleIncrement'])
        a = properties['StartAngle']#+properties['AngleIncrement']
        angles = []
        while a <= properties['EndAngle']:
            angles.append(a)
            a+=pi/180 #properties['AngleIncrement']

#angles.append(properties['EndAngle']-properties['AngleIncrement']
/2)
        return angles
    else:
```

```
        raise UnexpectedResponse(response)

def getPose():
    """Reads the current position and orientation from the MRDS"""
    mrds = http.client.HTTPConnection(MRDS_URL)
    mrds.request('GET','/lokarria/localization')
    response = mrds.getresponse()
    if (response.status == 200):
        poseData = response.read()
        response.close()
        return json.loads(poseData.decode())
    else:
        return UnexpectedResponse(response)

def heading(q):
    return rotate(q,{'X':1.0,'Y':0.0,"Z":0.0})

def rotate(q,v):
    return vector(qmult(qmult(q,quaternion(v)),conjugate(q)))

def quaternion(v):
    q=v.copy()
    q['W']=0.0;
    return q

def vector(q):
    v={}
    v["X"]=q["X"]
    v["Y"]=q["Y"]
    v["Z"]=q["Z"]
    return v

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

def qmult(q1,q2):
    q={}

q["W"]=q1["W"]*q2["W"]-q1["X"]*q2["X"]-q1["Y"]*q2["Y"]-q1["Z"]*q2[
"Z"]
```

```
q["X"]=q1["W"]*q2["X"]+q1["X"]*q2["W"]+q1["Y"]*q2["Z"]-q1["Z"]*q2[
"Y"]

q["Y"]=q1["W"]*q2["Y"]-q1["X"]*q2["Z"]+q1["Y"]*q2["W"]+q1["Z"]*q2[
"X"]

q["Z"]=q1["W"]*q2["Z"]+q1["X"]*q2["Y"]-q1["Y"]*q2["X"]+q1["Z"]*q2[
"W"]
    return q

def getHeading():
    """Returns the XY Orientation as a heading unit vector"""
    return heading(getPose()['Pose']['Orientation'])

def getPosition():
    """Returns the XYZ position"""
    return getPose()['Pose']['Position']

def getDistance(x, y):
    """Returns the distance between currentPosition and nextpos"""
    return sqrt((x ** 2) + (y ** 2))

def getBearingAngle(currentPosition, PositionP):
    """Returns the bearing angle between currentPosition and
nextpos"""
    x1, y1 = currentPosition
    x2, y2 = PositionP
    bearingAngle = atan2(y2-y1, x2-x1)
    return bearingAngle

def getHeadingAngle(x, y):
    """Returns the heading angle"""
    angleX = atan2(y, x)
    return(angleX)

def errorAngle(angleHeading, angleBearing):
    """Returns the difference between bearing angle and heading
angle"""
    errorA = angleBearing - angleHeading

    """Make sure that the robot does not circulate when angle is
    bigger/smaller than +-pi"""
    if (errorA < -pi):
        errorA = errorA + 2*pi
        return(errorA)
```

```
    elif (errorA > pi):
        errorA = errorA - 2*pi
        return(errorA)
    else:
        return(errorA)

def createPath():
    """Puts the positions of the path in a reversed list and
returns the list"""
    vecArray = []

    with open('D:\exam2017.json') as path_file:
        data = json.load(path_file)

        for i in range (len(data)):
            vecArray.append(data[i]['Pose']['Position'])
        vecArray.reverse()
        return vecArray


def carrotPoint(path, pose, lookAhead):
    """Finds the Carrotpoint and returns it"""

    if path:

        for i in range (len(path)):
            nextPos = path[len(path)-1]


            distX = nextPos['X'] - pose['X']
            distY = nextPos['Y'] - pose['Y']

            dist = getDistance(distX, distY)

            """If the carrotPoint is smaller than the distance to
the next
            point the list is poped and new carrotpoint is
searched"""
            if dist < lookAhead:
                path.pop()
            else:
                return nextPos

if __name__ == '__main__':
    print('Sending commands to MRDS server', MRDS_URL)
```

```
    """""""
    path = createPath()
    """Fixed stats for finished all paths"""
    speed = 0.7
    aSpeed = 1.3
    lookAhead = 1
    """Start the timer"""
    startTime = time.time()

    try:

        while path:
            pose = getPosition()
            head = getHeading()
            nextPos = carrotPoint(path, pose, lookAhead)

            """Checks if the robots have a next position, new
carrotpoint,
            else it stops until the robot finds one"""
            if(nextPos):

                """Calculate the headingAngle and bearingAngle"""
                dist = getDistance((nextPos['X'] - pose['X']),
                                   (nextPos['Y'] - pose['Y']))
                headingAngle = getHeadingAngle(head['X'],
head['Y'])
                bearingAngle = getBearingAngle((pose['X'],
pose['Y']),

(nextPos['X'],nextPos['Y']))
                angleError = errorAngle(headingAngle,
bearingAngle)
                response = postSpeed(angleError*aSpeed, speed)
                time.sleep(0.1)

            response = postSpeed(0, 0)
            endTime = time.time()
        """Calculates the runtime by take endtime - startime"""
        runTime = endTime - startTime
        print ('Runtime was:', runTime)

        response = postSpeed(0,0)

    except UnexpectedResponse as ex:
        print ('Error')
```