UMEÅ UNIVERSITET

Institutionen för Datavetenskap

Rapport obligatorisk uppgift

Fundamentals of Artificial Intelligence 7.5 Hp

5DV121 HT 16

Assignment 1

MRDS4 - Rapport

Namn		**Användarnamn**
Kristoffer Karlsson		kv14kkn@cs.umu.se
Jonatan Gustavsson		Kv14jgn@cs.umu.se
Handledare	Alexander Sutherland	

Innehållsförteckning

Problemspecifikation

Uppgiften gick ut på att implementera ett program som interagerar med programmet MRDS4 genom en lokal server. Programmet ska styra roboten så att den följer en redan given väg av kordinater som läses in från en .json-fil.

Det är även tillåtet att använda robotens lasersensorer för att förbättra robotens prestation.

Det rekommenderas även att använda någon av teknikerna; Follow the Carrot, eller Pure Pursuit.

Vi valde Follow the Carrot.

Originalinstruktion kan hittas här:
https://www8.cs.umu.se/kurser/5DV121/HT15/assignment1/index.html

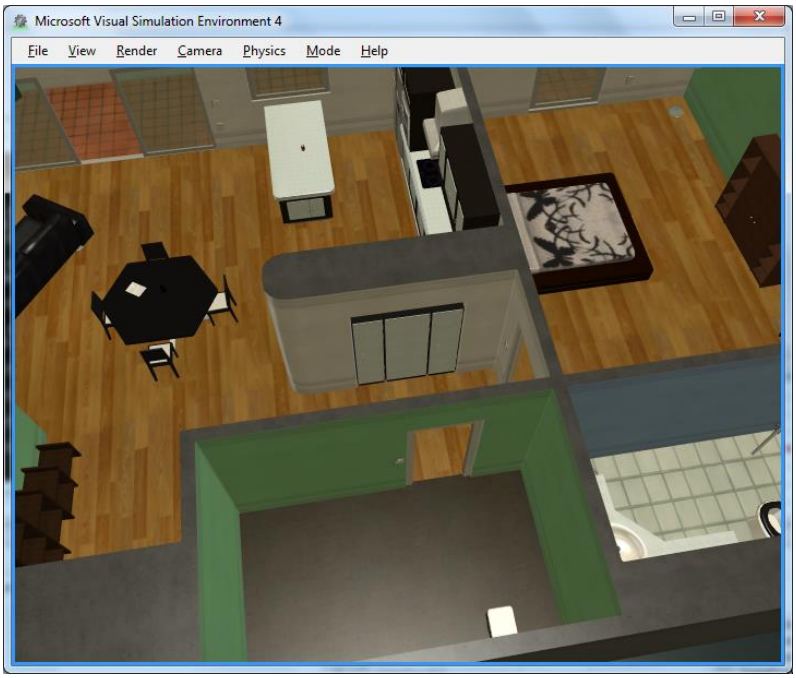
Notera:

En del av koden i lösningen är hämtad från tillgängliggjord kod för studenterna på kursen.

Användarhandledning

För att starta programmet behövs en Python IDE som är inställd på Python 2.7. Tryck på ”Kör”-knappen. Du får nu välja vilken Path du vill köra, eller skriva in ett eget filnamn. Innan du gör detta valet måste du ha startat upp MRDS4s filen StartLokarria.bat som hittas under MRDS4\store\launchers. Du får nu upp programmet och bland annat ett fönster med en lägenhetsmiljö, med en robot i. Det är denna robot som kommer navigera i rummet.

```
Select path:
 1. Path around table and back.
 2. Path to bed.
 3. Type in filename.
...:
```



Nu kan du gå tillbaka till din Python IDE och välja Path. När du trycker ”Enter” kommer roboten att köra den här Pathen.

Det går till så att programmet börjar med att välja en lämplig punkt en bit framför roboten. Det gör programmet genom att räkna ut distansen till varje punkt tills det hittar en punkt som är lagom långt bort. Det läser sedan av robotens position, dess vinkel och jämför denna information med den valda punkten. Programmet säger åt roboten att svänga mot punkten samtidigt som den kör framåt. Programmet ”sover” en kort stund, och sedan görs proceduren om.

Vidare har roboten lasersensorer som läser av dess omgivning och om den stöter på ett hinder tar dess över och stannar roboten, samt svänger åt motsats håll.

När Roboten har kört genom alla kordinater stannar programmet den och skriver ut tiden det tagit.

Algoritmbeskrivning

1. Programmet skriver ut en meny där användaren får ange vilken Path de vill att roboten ska köra. De kan även skriva in ett eget filnamn.
2. Programmet tar användarens val, läser in filen och indexerar koordinaterna i Path:en.
3. Programmet startar en timer.
4. Om det finns kordinater att köra till gör programmet följande:
    a. Programmet läser in robotens kordinater.
    b. Programmet räknar ut skillnaden mellan mål-koordinat X och Robot-koordinat X. Sedan gör den samma sak med Y koordinaterna.
    c. Programmet räknar ut skillnaden i vinkel mellan de båda X:en och Y:en. Detta är vinkeln mellan robotkroppen och punkten.
    d. Programmet räknar ut distansen till punkten, och om punkten inte är 1,4 distanser bort, kollar programmet efter en annan punkt. Om punkten är minst 1,4 distanser bort, fortsätter den med följande:
        i. Programmet räknar ut robotens vinkel i förhållande till X-axeln (robotens riktning).
        ii. Programmet lägger ihop vinkel mellan robot och punkt samt robotens riktning för att få en total vinkel roboten behöver svänga.
        iii. Om vinkeln i grader är större än 180 och mindre än -180. Svänger roboten i en hastighet som är robotens riktning minus vinkeln till punkten. Detta kommer bli en negativ vinkel och roboten svänger vänster.
        iv. Om vinkeln är något annat en ovan svänger roboten höger genom att svänga i hastigheten vinkeln till punkten minus robotens riktning.
        v. Programmet läser också av lasersensorerna.
            1. Programmet väljer ut de laserstrålar som pekar framåt på ett sådant sätt att de kännerav hinder snett framåt åt båda sidorna.
            2. Om de uppfattar ett hinder väldigt nära på någon sida stannar roboten upp, och svänger åt motsatt håll innan den fortsätter sin färd.
5. Programmet stoppar timern och stannar roboten, samt skriver ut hur lång tid det tog.

Flödesschema: Se bilaga 1.

Systembeskrivning

- Inläsning från fil
    - Programmet börjar med att köra funktionen *path_menu* som skriver ut en meny där användaren får välja fil att läsa in kordinater från genom att skriva in en integer, eller skriva in ett filnamn själv.
    - Informationen skickas till funktionen give_file som tar användarens input och associerar den med ett filnamn. Filnamnet returneras till path_menu som returnerar till huvudprogrammet.
    - Filnamnet skickas till *get_path()*, vilken tar ett filnamn. Filnamnet skickas till *open_json()* som också tar ett filnamn och öppnar filen med hjälp av json.load-modulen. Filinnehållet läggs i en lista, vilken returneras tillbaka till get_path.
    - Här tas alla path-kordinater ut och läggs i en lista. Varje koordinat är en dictionary som innehåller X, Y och Z. Listan returneras till huvudprogrammet.
- Värden specificeras
    - Counter = 1. Används för att kontrollera kommande whileloop.
    - Forward_speed = 1. Robotens hastighet framåt.
    - Sleep_time = 0.07. Tiden roboten får åka mellan varje kordinatavsökning,
    - Start_time = time.time(). Klockar av roboten initialt genom att ta ett timestamp i UTC.
- Styrdelen i programmet startas. Denna är konstruerad i en whileloop som körs så länge counter inte är större än längden på listan med kordinater
    - Robotens position läses in via funktionen *getPose()*
    - Programmet plockar ut en X-koordinat genom att ta aktuell X-koordinat från världen och aktuell X-koordinat från roboten och subtrahera dessa. Likadant med Y-koordinaterna för att få en Y-koordinat. (världens väljs genom att indexeras med counter)
    - Dessa båda skickas till funktionen *calc_angle()* för att få vinkeln mellan robotens center och punkten.
        - *Clac_angle* tar ett Y och ett X. I denna funktion körs den inbyggda modulfunktionen atan2, vilken ger tillbaka skillnaden i vinkel mellan de båda i radianer. Funktioner returnerar sedan detta.
    - Sedan körs funktionen *calc_distance()*.

        - I *calc_distance* räknas distans mellan robot och koordinat ut genom en formel som återfinns i bilaga 2. Resultatet returneras.
    - Programmet kollar om resultatet av calc_distance är mindre än inställd look-ahead-distance, 1.4. Om den är det adderas counter på med 1 och whileloopen körs igen. Om distansen däremot är större än 1.4 körs koden vidare. Vi har valt denna lösning för att undvika att roboten väljer punkter som är bakom den, eller för nära.
    - Programmet räknar ut robotens riktning i förhållande till X-axeln genom funktionen *get_robot_angle()*.
        - *get_robot_angle* Använder sig av funktionen getBearing för att plocka ut en punkt framför roboten och robotens X-axel(när roboten är rak). Sedan används funktionen *calc_angle()* för att räkna ut vinkeln mellan dessa två. Vinkeln returneras.
    - Programmet subtraherar sedan robotens riktning och vinkeln mellan robotens center och världskordinaten. Denna vinkel görs även om till grader genom att ta vinkeln * (180/pi)
    - Beroende på denna vinkel reagerar roboten olika. Om vinkeln är större än eller lika med 180 och mindre än -180 svänger roboten i en hastighet som är robotens riktning – vinkel mellan robotens center och världskoordinaten. Informationen om denna skillnad skickas till funktionen *turn()* vilken tar en radian, samt en integer som bestämmer hastighet framåt, i detta fall forward:speed – variabeln.
        - *Turn* använder i sin tur funktionen *postSpeed()*.
    - Annars svänger den i en hastighet som är vinkel mellan robotens center och världskordinaten – robotens riktning. Även detta sker med funktionen *turn*.
    - Nästa steg är lasercheck. Detta börjar med funktionen *get_laser_info()*.
        - *Get_laser_info* läser in robotens laserinformation via funktionen *getLaser()*. Sedan plockar den ut alla ”echoes” och rensar bort onödigt.
        - Dessa echoes loopas igenom från 100 till 180, för att endast få information om hinder framifrån och lätt åt sidorna. De från 100 till 140 läggs i listan som representerar höger sida och resten i listan som representerar vänster sida. De båda listorna läggs i en lista, vilken returneras

    - Programmet plockar ut minsta värdet ur de båda laserlistorna, vilket gör att de fungerar som två stora strålar istället för många små.
    - Om det minsta värdet i högerlistan är mindre än 0.5 stannar roboten och girar åt vänster med hjälp av *turn*.
    - Om det minsta värdet i vänsterlistan är mindre än 0.5 stannar roboten och girar åt höger med hjälp av *turn*.
- När alla världskoordinater loopats igenom beordras roboten att stanna och en ny time.time() tas.
- Tiden för åkturen tas genom att ta sista time.time – första time.time.

Problem och reflektioner

Vårt största problem var att vi saknar kunskap inom den matematik som används. Därför var det svårt för oss att bygga upp en algoritm med nödvändiga funktioner för att styra roboten. Lyckligtvis fick vi handledning som ledde oss genom problemen.

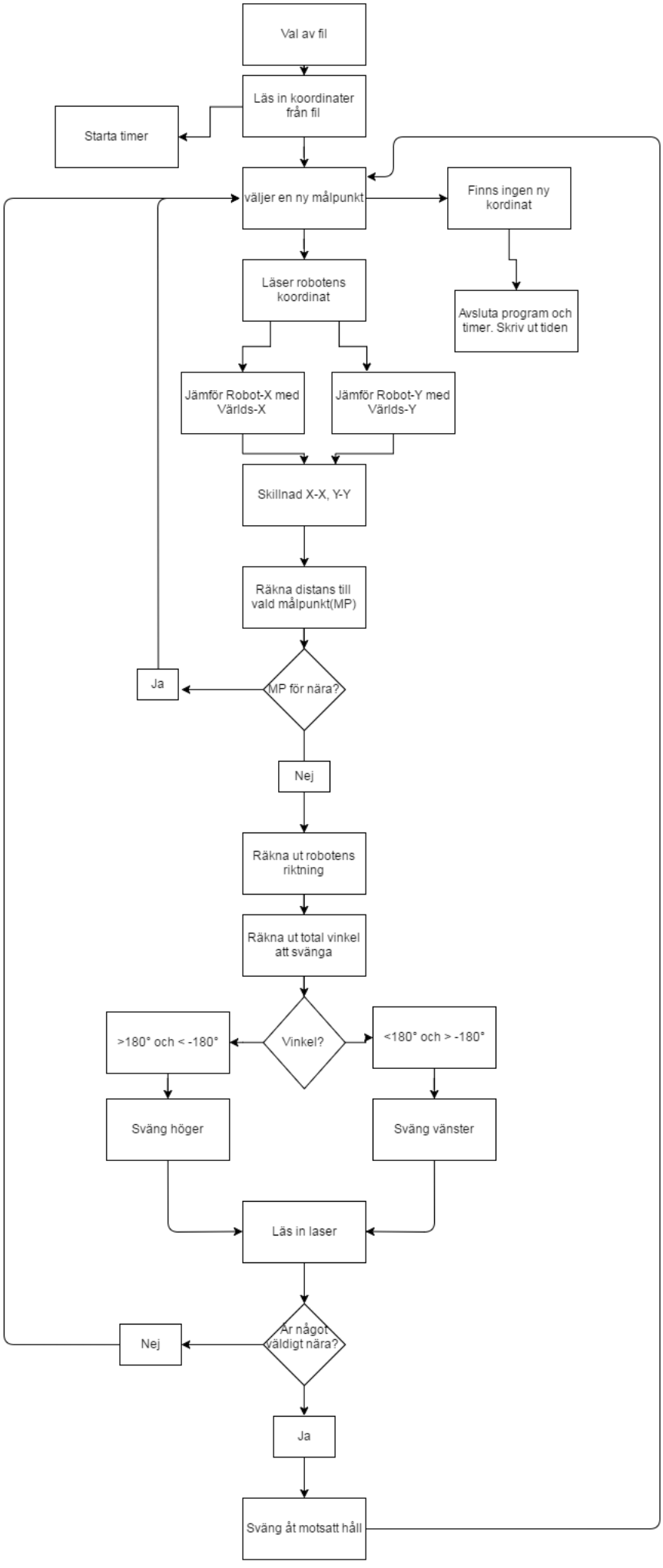
Ett annat problem var att förstå hur json-filen med koordinater, MRDS4 och vårt pythonprogram skulle integrera. Detta löste sig dock när vi kom igång med koden, samt fick lite handledning.

Ett problem som kvarstår är att roboten inte alltid lyckas undvika all hinder som finns. Ett stort problem är att ta sig förbi en stol vid bordet, samt att ta sig in i sovrummet från hallen.

Vi implementerade lasersensorer för att lösa detta. Problemet med vår lösning är att den är lite för enkel. Om roboten ska ta en snäv kurva genom en dörr, kan sensorerna reagera så att den svänger åt motsatt håll.

Bilagor

Bilaga 1



Bilaga 2

$$distance = \sqrt{dx^2 + dy^2}$$