# Experimental Project

## *"Trajectory Panning for the TurtleBotII Robot"*

Written by Barak Or,

Under the guidance of Professor Daniel Zelazo

November, 2015

DEPARTMENT OF AEROSPACE ENGINEERING | TECHNION Israel Institute of Technology

# Table of contents

## Introduction

This document summarizes the work on the experiment project over the last eight months, since April until these days.

My task at the beginning of the project was to study the robot "TurrtleBot2" system, starting from zero: open boxes and assembly of the robot, writing instructions to use for other users and assembly of units to other students. In addition, I knew the R.O.S and worked with the robot through the 'Matlab' environment, as detailed in this document.

Once I knew the system, I started working on the implementation of a known problem- Vehicle Routing Problem (V.R.P) with a certain constraint. I assumed assumptions that simplify the model and wrote an algorithm for solving the problem. At first I wrote an algorithm specific to an individual case, and then I wrote a generic solution. After writing the code and I've simulation at 'Matlab', I connect to robot into.

I made an experiment for checking the algorithm by implement it on the 'TurtleBot' robot. The experiment was held at the 'CASY-Cooperative Autonomous SYstems' lab.

Enjoy your reading.

# Summary report of TurtleBot2 hardware specification



## 0. Introduction

In the document I will give an overview on the robot components, I will describe each component functionality and how it connected to the robot.

At general, we have 3 main platforms: Robot, Laptop and Control computer. This summary deals with the Robot, the Laptop and the relationship between them.

## 1. Platforms overview

The Robot:

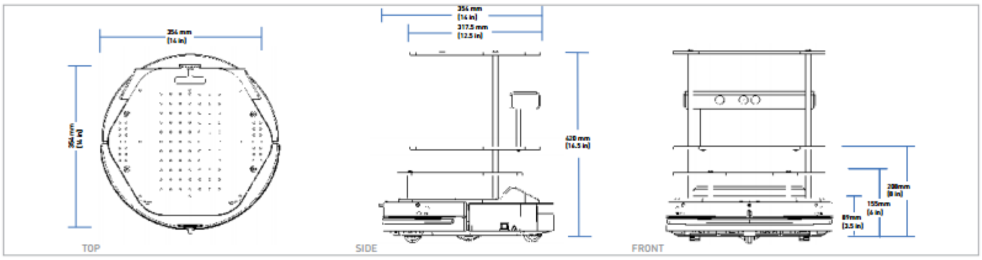



Image 1.1 " sketch 3 views"

| | |
|---|---|
| **SIZE AND WEIGHT** | |
| **EXTERNAL DIMENSIONS** (L x W x H) | 354 x 354 x 420 mm (14.0 x 14.0 x 16.5 in) |
| **WEIGHT** | 6.3 kg (13.9 lb) |
| **WHEELS** (Diameter) | 76 mm (3 in) |
| **GROUND CLEARANCE** | 15 mm (0.6 in) |
| **SPEED AND PERFORMANCE** | |
| **MAX. PAYLOAD** | 5 kg (11 lb) |
| **MAX. SPEED** | 65 cm/s (25.6 in/s) |
| **MAX. ROTATIONAL SPEED** | 180°/S |
| **BATTERY AND POWER SYSTEM** | |
| **STANDARD BATTERY** | 2200 mAh Li-Ion |
| **EXTENDED BATTERY** | 4400 mAh Li-Ion |
| **USER POWER** | 5 V and 19V (1A), 12 V (1.5A), 12V (5A) |

Image 1.2 " technical specification-Robot"

The Laptop:

| COMPUTER (subject to change) | |
|---|---|
| MEMORY | 4 GB |
| SCREEN | 11.6in (1366x768) |
| PROCESSOR | Intel Celeron 1007U dual-core @ 1.5GHz |
| GRAPHICS | Intel® HD Graphics |
| INTERNAL HARD DRIVE | 500 GB |
| WIFI | 802.11n |
| OPTICAL DRIVE | Not Applicable |

Image 1.3 " technical specification-Laptop"

The Sensor:

| SENSORS | | |
|---|---|---|
| 3D VISION SENSOR (Microsoft Kinect) | Color Camera: 640px x 480px, 30 fps. | Depth Camera: 640px 489px, 30 fps |
| ENCORDERS | 25700 cps | 11.5 ticks/mm |
| RATE GYRO | 100 deg/s Factory Calibrated | |
| AUXILIARY SENSORS | 3x forward bump, 3x cliff, 2x wheel drop | |

Image 1.4 " technical specification-Kinect"

2. How each component is connected to the Robot

-Place the Laptop on his shelf

-Connect the Laptop by USB cable to the Robot: one side into the front panel and the second into the USB exit of the Laptop.

-Take the XBOX 360 cable, as shown below, and connect the orange USB into the Kinect exit. Connect the "normal" USB into the Laptop.


-Connect the Kinect to the Robot Power:  12V, 1.5A, as pictured below:


-Pay attention that you are on operation mode

-if everything seems OK, you can turn on the Robot by turning the button at the bottom.


-Your Robot is ready to use.

3. TurtleBot Index

We have 12 unit of "TurtleBot". The entire system called "Tribes of Israel", where each robot has a number and a name of one of the tribes.
Each platform has 5 labels on it, R=Robot, C=Computer, K-Kinect, P-Power, X-XBOX Cable.

| | | | | | |
|---|---|---|---|---|---|
| R1 | C1 | K1 | P1 | X1 | Dan |
| R2 | C2 | K2 | P2 | X2 | Reuben |
| R3 | C3 | K3 | P3 | X3 | Simeon |
| R4 | C4 | K4 | P4 | X4 | Levi |
| R5 | C5 | K5 | P5 | X5 | Judah |
| R6 | C6 | K6 | P6 | X6 | Naphtali |
| R7 | C7 | K7 | P7 | X7 | Gad |
| R8 | C8 | K8 | P8 | X8 | Asher |
| R9 | C9 | K9 | P9 | X9 | Issachar |
| R10 | C10 | K10 | P10 | X10 | Zebulun |
| R11 | C11 | K11 | P11 | X11 | Joseph |
| R12 | C12 | K12 | P12 | X12 | Benjamin |

4. Sources:

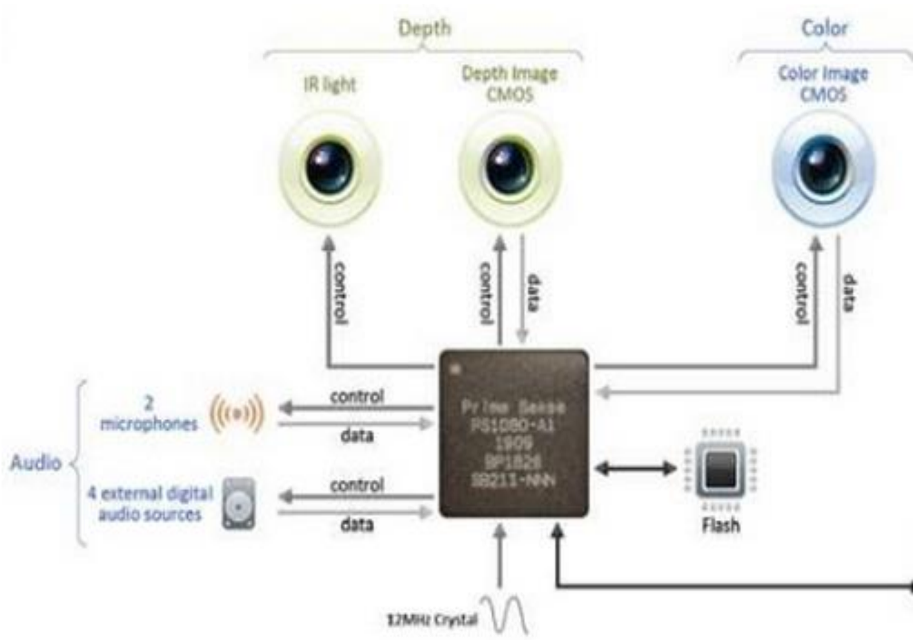
In this task, I used the following sources:
http://www.clearpathrobotics.com/turtlebot_2/downloads/
Packing list CLEARPATH. Kobuki quick guide

*Recommend Video: http://learn.turtlebot.com/2015/02/01/3/

## XBOX Short Review


3D Depth sensor- 3D sensors tracks the body within the "play space"

RGB Camera- 640*480 pixels, help identify and takes in-game pictures and video.



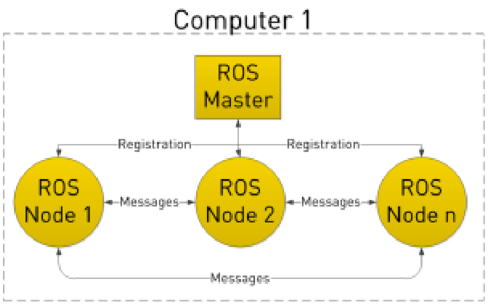
# Robot Operating System- R.O.S

In this document, we will discuss in R.O.S- the Robot Operating system and it role in operating our robot- the TurtleBot2.

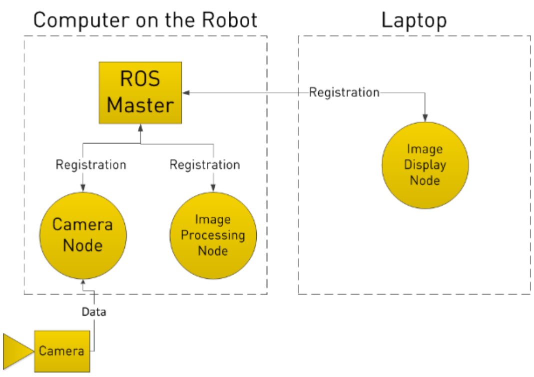
What is it?

R.O.S is a system for controlling robotic components from a computer. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish messaging model. For example, a particular sensor's driver might be implemented as a node, which publishes sensor data in a stream of messages. These messages could be consumed by any number of other nodes, including filters, loggers, and higher-level systems such as guidance, etc.

The concept: we can simply tell Node 1 to send messages to Node 2. As shown:




The nodes publishing and subscribing.




This link is very helpful for the beginning: [http://www.clearpathrobotics.com/blog/how-to-guide-ros-101/](http://www.clearpathrobotics.com/blog/how-to-guide-ros-101/)

# Quick Guide to initialize the robot

We have to define a "Master" and "workstation". The "Master" will be the Acer Laptop which we put on the "TurtleBot" and the workstation will be the

1. Turn on the Laptop. (Acer)
2. Open a new terminal window by pressing "Ctrl+Alt+t"
3. Discover your I.P by write "*ifconfig*". The I.P will show up at the line which is start with the words "*inet addr*".
4. Define as a master by the command: *echo export ROS_MASTER_URI=http://IP_OF_TURTLEBOT:11311 >> ~/.bashrc* . where *IP_OF_TURTLEBOT* is the I.P from step 3.
5. Define the "Master" host-name by the command, at the same terminal window "*echo export ROS_HOSTNAME=IP_OF_TURTLEBOT >> ~/.bashrc*"
6. Now, open a new terminal window and operate ROS by the command: "*roscore*"
7. Check the system is working properly and no fault in one of the components. Open a new terminal window and give the command: "*roslaunch turtlebot_bringup minimal.launch*"
   At this step you can play with the keyboards key for controlling the 'TurtleBot'
8. We have to establish a network between the Acer laptop and the workstation. After we make sure there is available Wi-Fi in the room we have to create a new Wi-Fi Network between the two. At first, open the window "Create New Wi-Fi Network" by pressing on the connection icon at the top menu. A new window will show up:

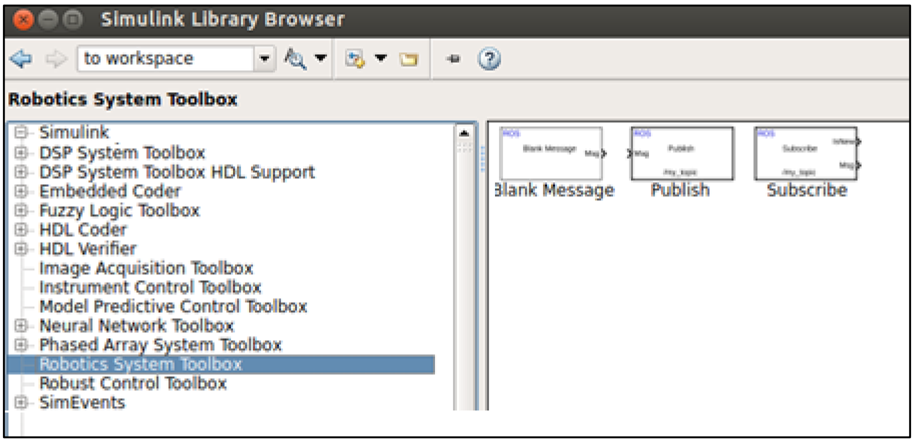

   At the "Network name" insert Any name you want (and remember it). I recommend on "TurtleBot".

9. Now we have the new network as one of the option to connect on among the other. From now and go on we move to deal with the workstation. We need to find the I.P address, at the same way we did in steps 1-3.
10. Define the Master on the workstation (for they know to recognize each other). As written in step 4, give a command: *echo export ROS_MASTER_URI=http://IP_OF_TURTLEBOT:11311 >> ~/.bashrc* . where *IP_OF_TURTLEBOT* is the I.P from step 3.
11. Define the workstation as an "host": *echo export ROS_HOSTNAME=IP_OF_WORKSTATION >> ~/.bashrc*
12. "Connection Testing". For checking if we actually succeed we can give a command on the workstation: *rostopic pub -r10 /hello std_msgs/String "hello"*. Now on the Laptop we give the command *rostopic echo /hello*. If the word "hello" show over and over than we are fine. If not, return the steps- there is a problem. To stop, you can press "Ctrl+c".

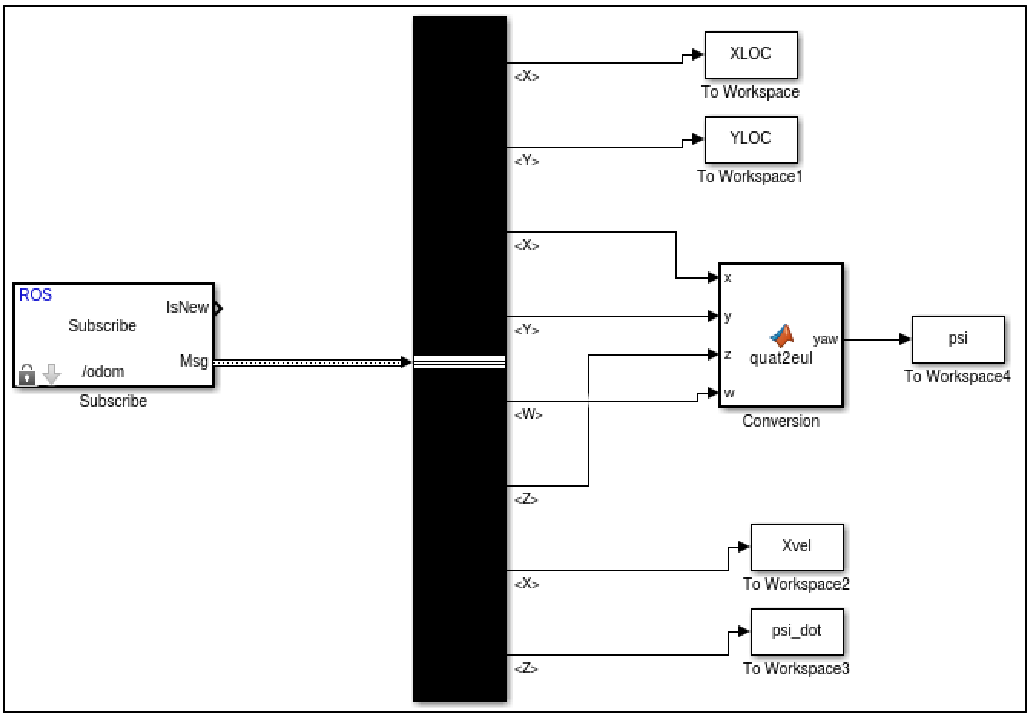
# Controlling the robot by using 'Matlab'

Using "Robotic System Toolbox" package make it easier to control the robot. There are 3 common operators:

-Blank Message
-Publish
-Subscribe

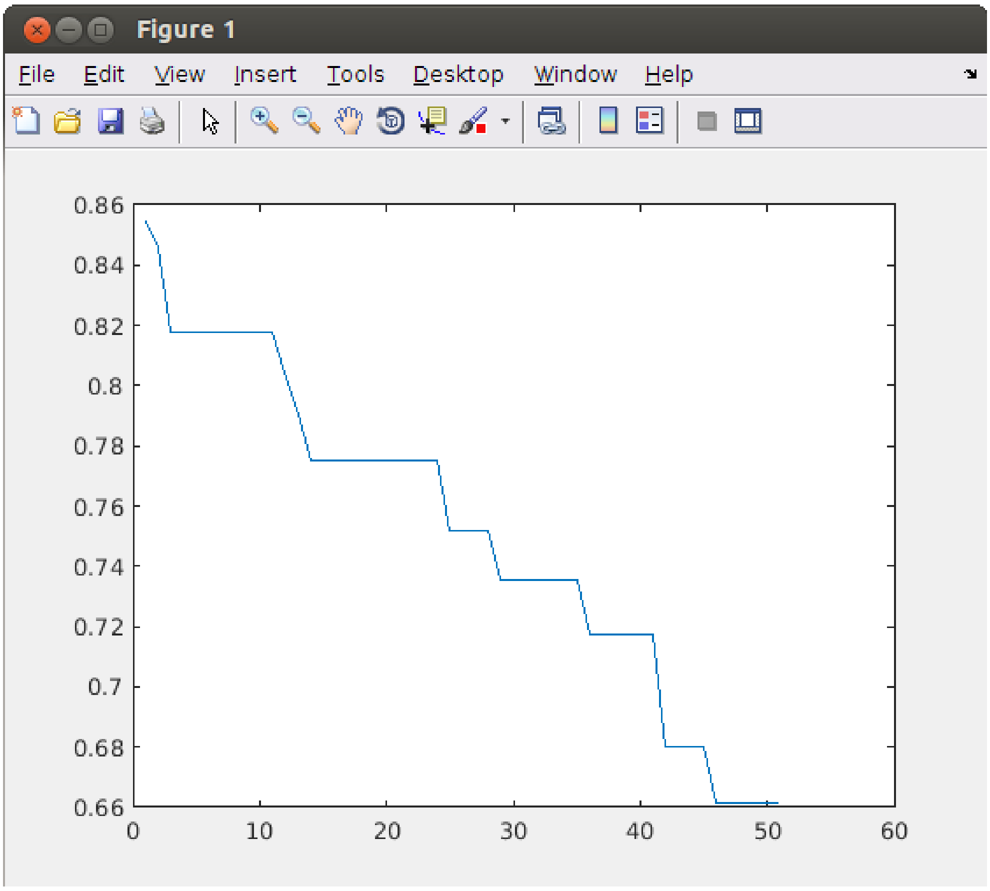



we can build the following "blocks diagram". We use the "Subscribe" and we can get out data about the robot: velocity, place, rotation angel and more.



For example, if we want to get a graph of the Yaw angle ("psi") we can import the data form the robot into the work space on 'Matlab', as shown above. The graph we get:

"Yaw (rad) by time (sec)"



## Trajectory panning- Vehicle Routing Problem

Goal:

Find the optimal time& distance solution for the Vehicle Routing Problem. Meaning- minimizing the total route cost function by finding the shortest path and minimum time.

The V.R.P is a combinatorial optimization problem which asks "what is the optimal set of routes for a fleet of vehicles to traverse in order to deliver for a given set of customers", in our case we have only 1 vehicle, "TurtleBot" Robot and N customers. We will see first a simple case for N=4 which give us basic understanding about the optimization process and then we will generalize to every possible case without regard for the complexity of the process.

Assumption:

-One vehicle (TurtleBot)

-At least one charging station (depot)

-The vehicle must visit a set of points ("customers")

-After visiting all customers return to a depot

-The vehicle allowed returning to a depot during a tour and must do it after 2 visits for "refuel energy".

The mission:

1. Visits each customer once
2. Minimizes the total distance travelled
3. Minimizes the total time used

## Algorithm a, 4 stations

The idea is to optimize the path between 4 stations. Visit at the 'docking' station after 2 visits (for charging, "refuel energy").

Define docking station at $O(0,0)$

Get 4 coordinates and insert into $2\times4$ matrix (Point_val):

$$\textbf{Point value} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} 1 & 3 & 2 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix}$$

Calculate the distance of each station relative to docking station (distance):

$$\textbf{distance}(i) = \sqrt{x_i^{\,2} + y_i^{\,2}} \rightarrow \textbf{distance} = \begin{pmatrix} \sqrt{10} & \sqrt{13} & \sqrt{5} & \sqrt{32} \end{pmatrix}^T$$

Let r* will be the sum of distance for each station relative to docking station

$$\mathbf{r}^* = \sum_i \textbf{distance}(i) = \sqrt{10} + \sqrt{13} + \sqrt{5} + \sqrt{32} = 14.66$$

Building a matrix of all distances between the stations:

$$\bar{\bar{a}} \equiv \underbrace{\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}}_{General\ Structure} \rightarrow \underbrace{\begin{pmatrix} 0 & a_{12} & a_{13} & a_{14} \\ 0 & 0 & a_{23} & a_{24} \\ 0 & 0 & 0 & a_{34} \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{Calculation} \rightarrow \underbrace{\begin{pmatrix} 0 & a_{12} & a_{13} & a_{14} \\ a_{12} & 0 & a_{23} & a_{24} \\ a_{13} & a_{23} & 0 & a_{34} \\ a_{14} & a_{24} & a_{34} & 0 \end{pmatrix}}_{final\ structure}$$

$$a_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j^{\,2}\right)}$$

$$\bar{\bar{r}}\ symetric = \begin{pmatrix} 0 & 2.2361 & 2.2361 & 3.1623 \\ 2.2361 & 0 & 1.4142 & 2.2361 \\ 2.2361 & 1.4142 & 0 & 3.6056 \\ 3.1623 & 2.2361 & 3.6056 & 0 \end{pmatrix}$$

Define min distance (d_min) =10000

define z=time that min distance found, initial at z=1

Loop for finding the minimum distance:

running all over the matrix and check which combination give the minimum distance. When find, save the order and the total distance. One step in the loop is shown:

$$if\ \{(j \neq k) \cap (j \neq l) \cap (j \neq m) \cap (k \neq l) \cap (k \neq m) \cap (l \neq m)\}$$

$$sum = r_{jk} + r_{lm}$$

$$if\ (sum \leq d_{\min})$$

$$(d_{\min} := sum)$$

$$(total_{distance} = d_{\min} + r^{*})$$

$$order = [j \quad k \quad l \quad m \quad d_{\min}]$$

$$z = z + 1$$

$$end$$

$$end$$

$$final\ d_{\min} = order(5)$$

For example let run on the situation when $j = 1, k = 2, l = 3, m = 4$

$$sum = r_{12} + r_{34}\ \{2.2361 + 3.6056 = 5.8417\}$$

$$if\ (sum \leq d_{\min})\ \{5.8417 \leq 10000\}$$

$$(d_{\min} := sum)\{d_{\min} := 5.8417\}$$

$$(total_{distance} = d_{\min} + r^{*})\ \{total_{distance} = 5.8417 + 14.66 = 20.5017\}$$

$$order = [j \quad k \quad l \quad m \quad d_{\min}]\{[1 \quad 2 \quad 3 \quad 4 \quad 5.8417]\}$$

$$z = z + 1$$

$$end$$

$$end$$

$$final\ d_{\min} = order(5)$$

Calculate permutations and print each of them with the total minimum energy (distance).

Matlab Code

```
clc
close all
clear all

i               = 4; %Number of stations
docking         = [0,0]; %Coordinate of initial position
Point_val       = [1 3 2 4; 3 2 1 4]; %First row for the values of X and second row for the the values of Y
vector_of_letters = cellstr(char('A', 'B' ,'C' ,'D')); %Station names
vector_of_letters = vector_of_letters';%Rotate

for j=1:1:i
    distance(j) = sqrt( (Point_val(1,j) )^2 + (Point_val(2,j) )^2 );  %Calculating the distance of each station relative to docking station
  for k=(j+1):1:i
        r(j,k) = sqrt( (Point_val(1,j)-Point_val(1,k))^2 +(Point_val(2,j)-Point_val(2,k))^2);
  end
end

r_star          = sum(distance);  %Sum of the distances of all stations relative to the docking station
r(i,:)          = zeros;%Adding more row
z               = 1;%Initial conditions
r_sym           = r+triu(r,1)';% Makes the matrix symmetry
d_min           = 1000000;%Initial conditions

%% The next section refers to the situation where there are 4 stations
for j=1:1:i
   for k=1:1:i
      for l=1:1:i
         for m=1:1:i

            if ((j~=k)&&(j~=l)&&(j~=m)&&(k~=l)&&(k~=m)&&(l~=m))%Ensures that the stations are not the same and we visit them all
               vector    = [r_sym(j,k) r_sym(l,m)];
               d_compere = sum(vector);

               if d_compere<=d_min; % A comparison between the current value and the minimum value
                   d_min        = d_compere;% Placing the minimum value
```

```
                total_distance = d_min + r_star; %Total distance including the
distances to docking station
                order(z,:)     = [ j k l m d_min]; % Saving order in which visiting at
stations and the min distance
                z              = z+1;
             end

           end
         end
      end
   end
end
p = 1;
final_d_min = order(z-1,5); %the min distance

for j=1:1:z-1
 if order(j,5)==final_d_min
   result(p,:) = order(j,:);
   p = p+1;%Number of permutations
 end
end

fprintf('There are %0.d of optimal trajectory between 4 stations \n',p-1)

for j=1:1:p-1
   fprintf('O%s%sO%s%sO
\n',vector_of_letters{result(j,1)},vector_of_letters{result(j,2)},vector_of_letters{result(
j,3)},vector_of_letters{result(j,4)})
end

fprintf('The total energy is %0.2f',final_d_min)
```

Command Window

```
There are 8 of optimal trajectory between 4 stations
OACOBDO
OACODBO
OBDOACO
OBDOCAO
OCAOBDO
OCAODBO
ODBOACO
ODBOCAO
fx The total energy is 4.47>>
```

| Name | Value | Min | Max |
|---|---|---|---|
| Point_val | [1,3,2,4;3,2,1,4] | 1 | 4 |
| d_compere | 5.8416 | 5.8416 | 5.8416 |
| d_min | 4.4721 | 4.4721 | 4.4721 |
| distance | [3.1623,3.6056,2.2361,... | 2.2361 | 5.6569 |
| docking | [0,0] | 0 | 0 |
| final_d_min | 4.4721 | 4.4721 | 4.4721 |
| i | 4 | 4 | 4 |
| j | 8 | 8 | 8 |
| k | 4 | 4 | 4 |
| l | 4 | 4 | 4 |
| m | 4 | 4 | 4 |
| order | <10x5 double> | 1 | 5.8416 |
| p | 9 | 9 | 9 |
| r | <4x4 double> | 0 | 3.6056 |
| r_star | 14.6608 | 14.6608 | 14.6608 |
| r_sym | <4x4 double> | 0 | 3.6056 |
| result | <8x5 double> | 1 | 4.4721 |
| total_distance | 19.1329 | 19.1329 | 19.1329 |
| vector | [3.6056,2.2361] | 2.2361 | 3.6056 |
| vector_of_letters | <1x4 cell> | | |
| z | 11 | 11 | 11 |

## Algorithm b , N stations

Here we let the user choose for number of station "N". For each station we have to enter the x,y coordinates. After we have all these data we send it to VRP function which returns us the best order of the station for optimal route and the distance we have to do (the "energy").

-main-

By given 'N' (num. of station) and 'Point_val' (coordinates):

$$\textbf{Point value} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & \cdots & x_N \\ y_1 & y_2 & \cdots & y_N \end{pmatrix}_{2\times N}$$

Running function 'VRP' (return the optimal route and the energy)

-VRP function-

Define docking station at $O(0,0)$

Calculate the distance of each station relative to docking station (distance):

$$\textbf{distance}(i) = \sqrt{x_i^{\,2} + y_i^{\,2}} \rightarrow \textbf{distance} = \begin{pmatrix} d_1 & d_2 & \cdots & d_N \end{pmatrix}$$

Building a matrix of all distances between the stations:

$$\bar{\bar{a}} \equiv \underbrace{\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{pmatrix}}_{General\ Structure} \rightarrow \underbrace{\begin{pmatrix} 0 & a_{12} & \cdots & a_{1N} \\ 0 & 0 & \cdots & a_{2N} \\ 0 & 0 & 0 & \vdots \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{Calculation} \rightarrow \underbrace{\begin{pmatrix} 0 & a_{12} & \cdots & a_{1N} \\ a_{12} & 0 & \cdots & a_{2N} \\ \vdots & \ddots & 0 & \vdots \\ a_{1N} & a_{2N} & \cdots & 0 \end{pmatrix}}_{final\ structure}$$

$$a_{ij} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j^{\,2}\right)}$$

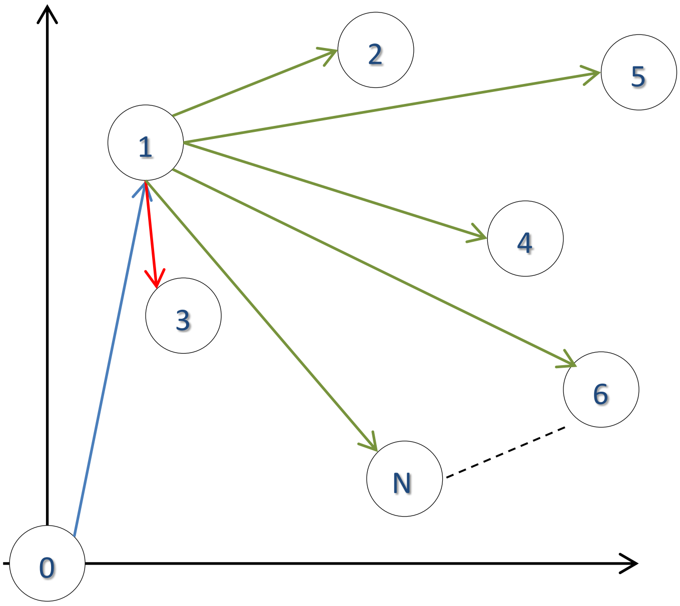
Define *final vector* and initialize by '-1' values.

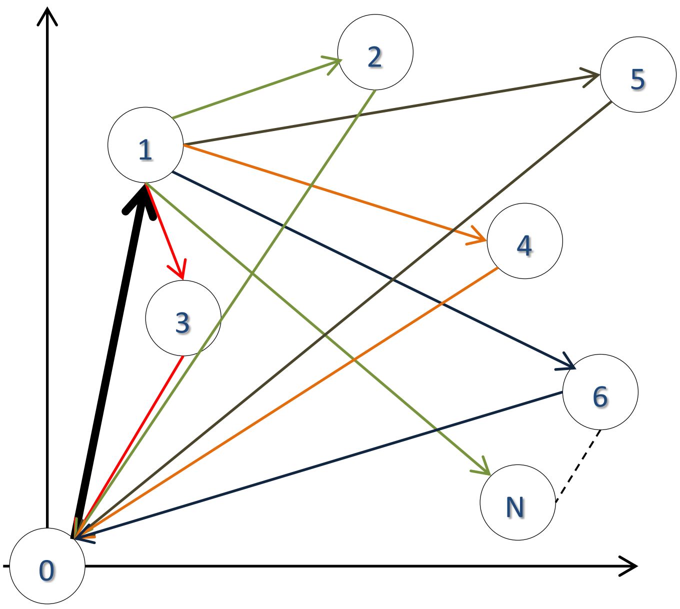
Define *min distance vector* (d_min_vector) =<u>0</u>

Loop for finding the minimum distance:

Running over all the values at upper triangular matrix for finding combinations which give the minimum distance. When find, save the next station who gave the minimum distance (as show below) into *station_min*.

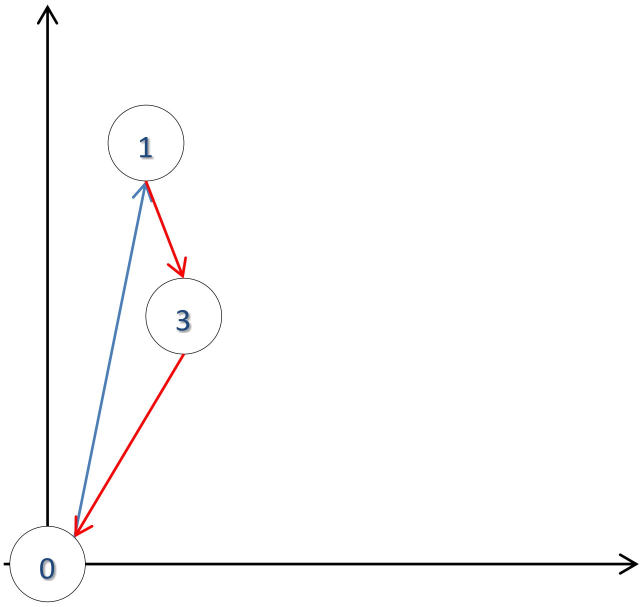
At the figure we can see the dynamic process:
we start with station *j*='*1*' and run all over the other N-1 station for finding the shortest sub path. At the figure below we can see half of the process "the searching". Each station search for the next station *i* to move. The next step is checking the shortest path we get when we choose station *k*. We can describe the distance with the formula: $d_{0\to j\to k\to 0} = d_j + d_{jk} + d_k$ .When we go from zero. Visit on two stations and return to the zero point



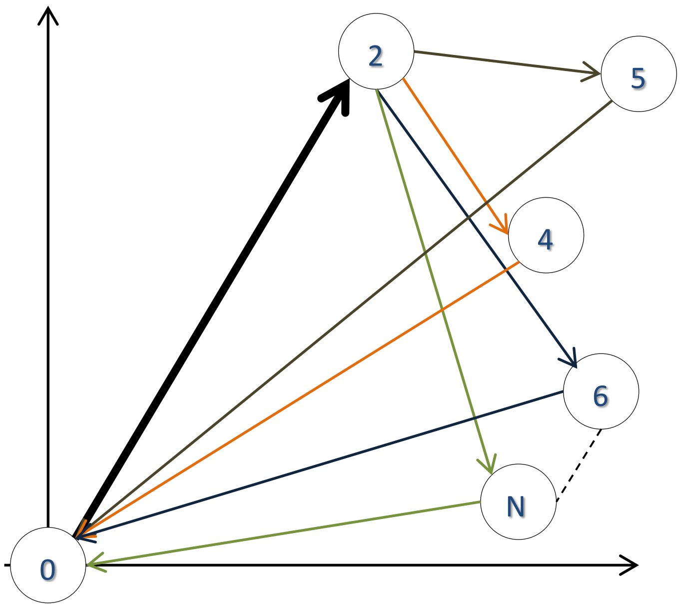




When we found the optimal sub path (as shown at the next figure: *j=1* and *k=3)* we save it in a new vector (*final*) at place j we put *k* and in place k we put *j* instead the *-1* value. This presentation was intended so that you can to erase the stations who "we handled" of them.

So the vector *final* is now: $final = (1,-1,3,-1,-1,...,-1)_{1\times N}$ . We save the distance into `d_min_vec(j)=d_min` and a sub path is getting the form:




Now the "map" is looking:




And the process continues the same until it finish handle all the station.
There is a case when we have odd number of station and we have to match the last station a partner. The solution is by match it for himself. The searching for this station is by passing all over vector *final* and find when we have a cell with '-1'. When it found we place *final(j)=j*.

So at the end of the section we have $final = \left( \underset{1}{4}, \underset{2}{7}, \underset{3}{6}, \underset{4}{1}, \underset{5}{5}, \underset{6}{3}, \underset{7}{2} \right)_{1\times 7} \quad N = 7$ .

Now we want to arrange it in a logical vector. So we define a new vector *order* which initialize with zeros. It put at place *j* the value from *final (j)* and in place *i=final (j)* place '*-1*'.

$$order = \left( \underset{1}{4}, \underset{2}{7}, \underset{3}{6}, \underset{4}{-1}, \underset{5}{5}, \underset{6}{-1}, \underset{7}{-1} \right)_{1\times 7} \quad , N = 7$$

We can see that if *final (i)=i then order(i)=i* . (Tip case for odd num. of stations). The next step is "to prepare", it means we have to consider a constraint which states that after 2 visits to the stations, we must return to the docking station.

We build *output*= $\begin{bmatrix} 0 & x_1 & x_2 & 0 & x_3 & x_4 & 0 & \cdots \\ 0 & y_1 & y_2 & 0 & y_3 & y_4 & 0 & \cdots \end{bmatrix}^T_{(2*N)\times 2}$ .

We can see that *output* get the x and y values only if *order* has a valid value. The places it puts the values are divide into groups of two double cells:

The condition for placing $\begin{bmatrix} x_i & y_i \end{bmatrix}^T$ is as shown:

$$
\begin{aligned}
& k = 2 \ \left(start\ at\ \underline{0}\right) \\
& for\ j = 1,\ j \le N \\
& \qquad if\ \ order(j) > 0 \\
& \qquad\qquad output(k,1) = \text{Point\_val}(1,j) \\
& \qquad\qquad output(k,2) = \text{Point\_val}(1,j) \\
& \qquad\qquad if\ \left(order(j) \ne j\right) \\
& \qquad\qquad\qquad output(k,1) = \text{Point\_val}(1,j) \\
& \qquad\qquad\qquad output(k,2) = \text{Point\_val}(1,j) end \\
& \qquad\qquad\qquad k = k + 1 \\
& \qquad\qquad end \\
& \qquad\qquad k = k + 1 \\
& \qquad end \\
& end
\end{aligned}
$$

There is the tip case: odd number of station. In this case we have to check if *final(i)=i.* if so, the variable *last=i.*

For calculation the optimal distance (the minimum energy path) we have to sum all the minimum distances (*d_min_vec)* which we are already found at the first section (when we built up *final* vector). If the N is odd then we have to put in *d_min_vec(last)=2*distance(last),* so at the end of the route, the vehicle will be straight to him and back.

Example:

$$\textbf{Point\_val} = \begin{bmatrix} 3 & 7 & 2 & 1 & 6 & 7 & 2 \\ 1 & 5 & 7 & 3 & 9 & 1 & 4 \end{bmatrix} \quad N = 7$$

| | |
|---|---|
| 0 | 0 |
| 3 | 1 |
| 1 | 3 |
| 0 | 0 |
| 7 | 5 |
| 2 | 4 |
| 0 | 0 |
| 2 | 7 |
| 7 | 1 |
| 0 | 0 |
| 6 | 9 |
| 0 | 0 |

The minimum energy is 92.9104

```
%%main of VRP
clc
close all
clear all

N = 7; %Number of stations
Point_val = [3 7 2 1 6 7 2; 1 5 7 3 9 1 4]; %1st row X values 2st row
Y values
[out,opt_dist] = VRP(Point_val,N);



function [ output, opt_dist ] = VRP( Point_val,N )

docking = [0,0]; %Coordinate of initial position
for j=1:1:N
      distance(j) = sqrt( (Point_val(1,j) )^2 + (Point_val(2,j) )^2
);  %Calculating the distance of each station relative to docking
station
   for k=(j+1):1:N
           r(j,k) = sqrt( (Point_val(1,j)-Point_val(1,k))^2
+(Point_val(2,j)-Point_val(2,k))^2); %building Matrix of distance
between stations
   end
end

r_star             = sum(distance);  %Sum of the distances of all
stations relative to the docking station
r(N,:)             = zeros;%Adding more row
z                  = 1;%Initial conditions
r_sym              = r+triu(r,1)';% Makes the matrix symmetry

for j=1:1:N    %%initializion of final vector
    final(j)=-1;
end

d_min_vec(N) = zeros; %%initializion of minimun distance vector


%%calculate the optimal path
for j=1:1:N
    d_min    = 10000; % max value

    for k=1:1:N
          if ((j<k) && (final(k) < 0))%only values on upper
triangular matrix
              temp = (distance(j)+r_sym(j,k)+distance(k)); %calculate
partial path
                if (temp < d_min)
                d_min= temp;
                station_min = k;%saving the optimal next station
                end
          end
    end

    if (d_min<10000) %%taking the real min value
        d_min_vec(j)=d_min; %%saving min distance for optimal path
    end

    if ((final(j)<0) && (final(station_min) < 0))
        final(station_min) = j;  %symetry
```

```
        final(j)= station_min;   %here we place the next station to
go
    end

end


%taking the last final and make it its own value. (solving tip case).
for j=1:1:N
    if (final(j)==-1)
        final(j)=j;
    end
end

%initialize vector of station order for making Comparison
for j=1:1:N
    order(j)=0;
end

for j=1:1:N
    if (order(j)==0)
        order(j) = final(j);
        if (final(j)~=j)
            order(final(j))=-1;
        end
    end
end
output(2*N,2)= zeros;

k=2; %start from docking station

for j=1:1:N
    if (order(j)>0)  %%valid point
        output(k,1)=Point_val(1,j);
        output(k,2)=Point_val(2,j);
        if (order(j)~=j)
            output(k+1,1)=Point_val(1,order(j));
            output(k+1,2)=Point_val(2,order(j));
            k=k+2;
        end
        k=k+1;
    end
end

%finding the last station when we have odd number of N stations
for i=1:1:N
     if (final(i)==i)
         last=i;
     end
 end

%calculate total optimal distance
d_min_vec(last)=2*distance(last);

opt_dist=0;
for i=1:1:N
     opt_dist=opt_dist+d_min_vec(i);
end
end
```

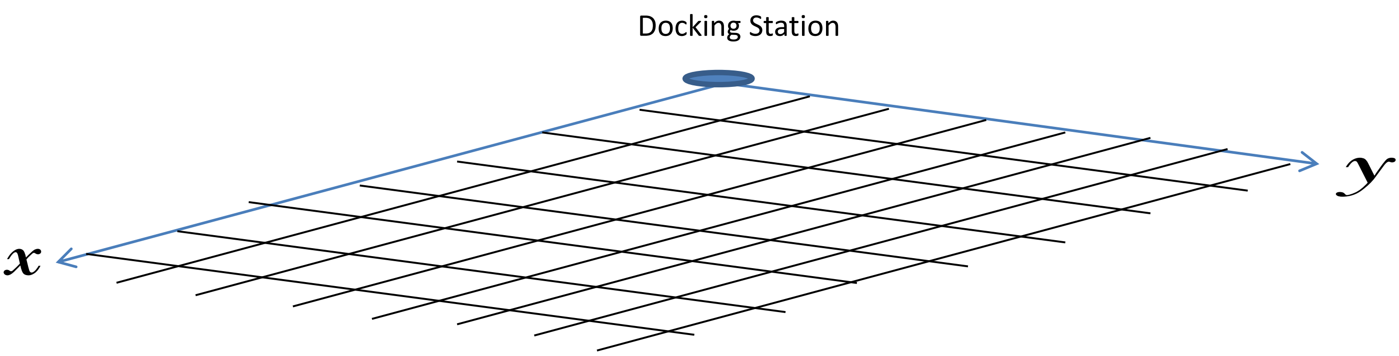
## Experiment by using 'TurtleBot' robot

The experiment took a place at the 'CASY-Cooperative Autonomous SYstems' lab. Date of experiment: Wednesday 11.11.2015

The vehicle 'TurtleBot' make its route by passing all over the points, according the values from the vector *output*.

In the link you can watch the video of the experiment:
https://www.youtube.com/watch?v=IlHwWoQK_1g

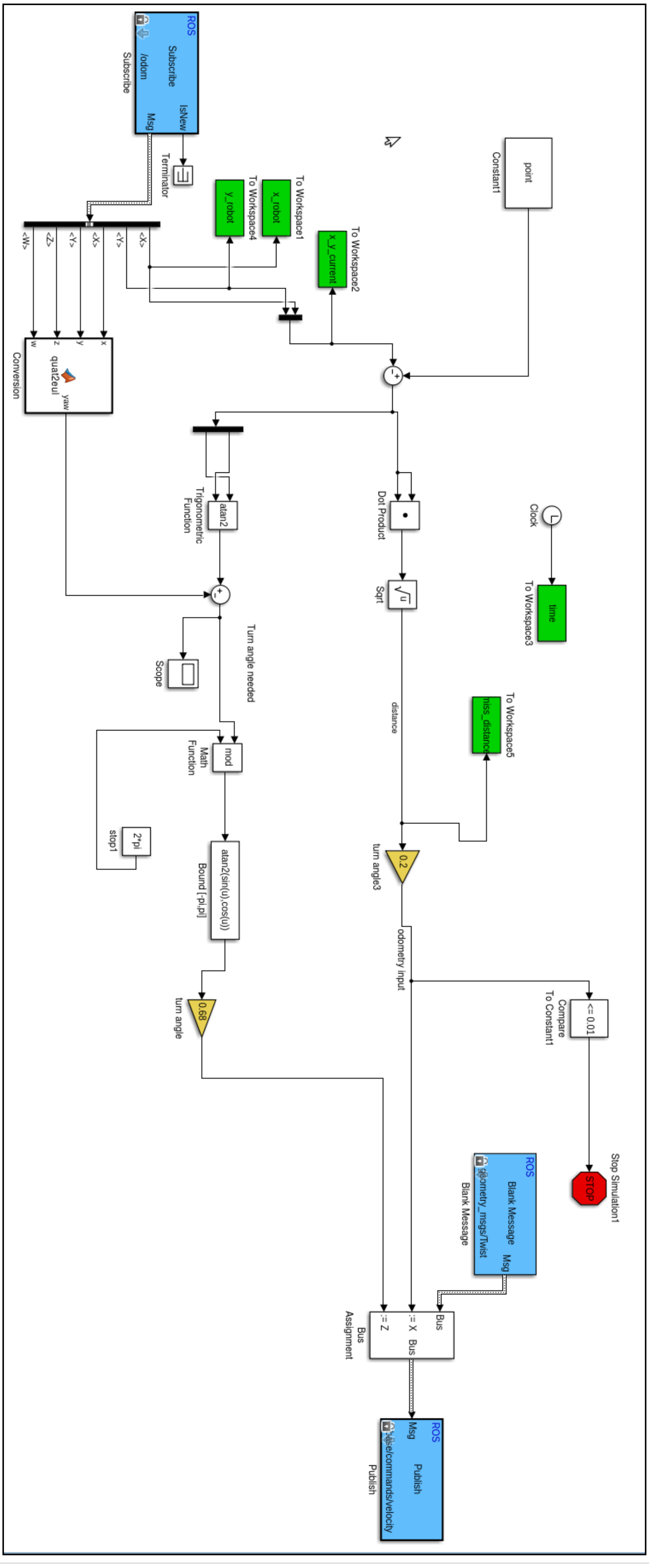
Note: turn on the subtitles for description.



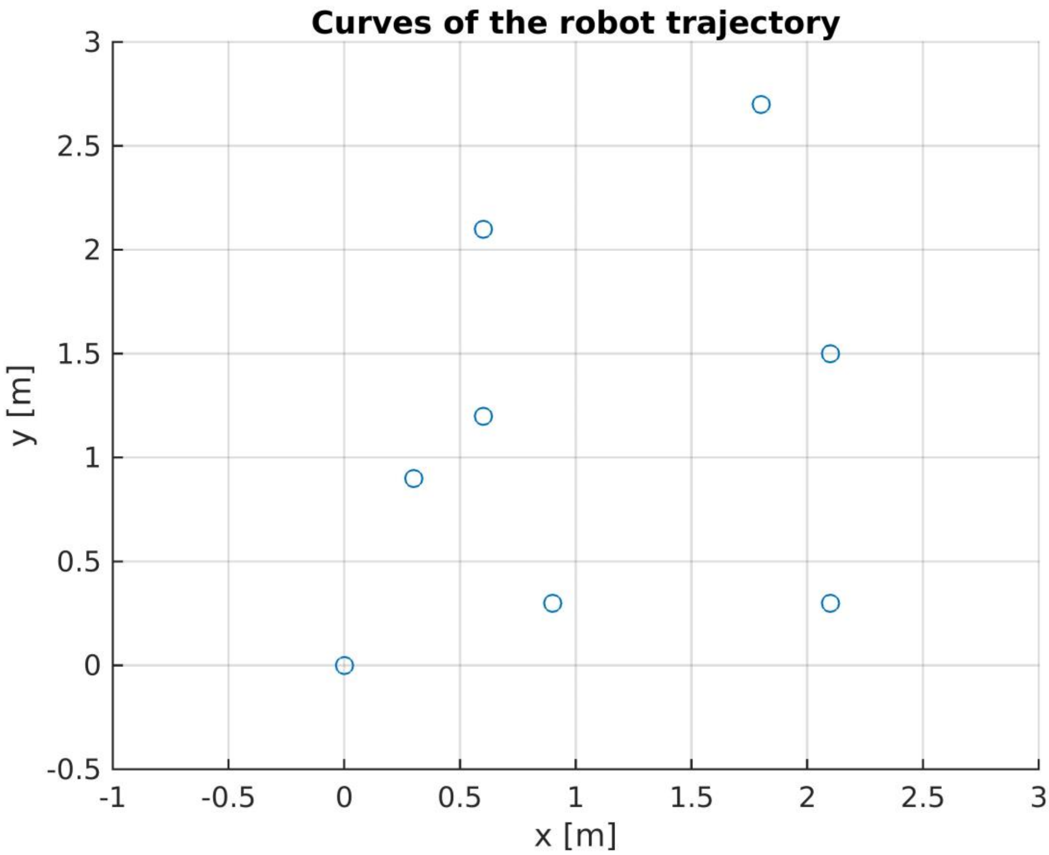
The area we chose for making the experiment was a square 3*3 meters. We normalized the data (the coordinates) by multiply by factor 0.3, so the values are:

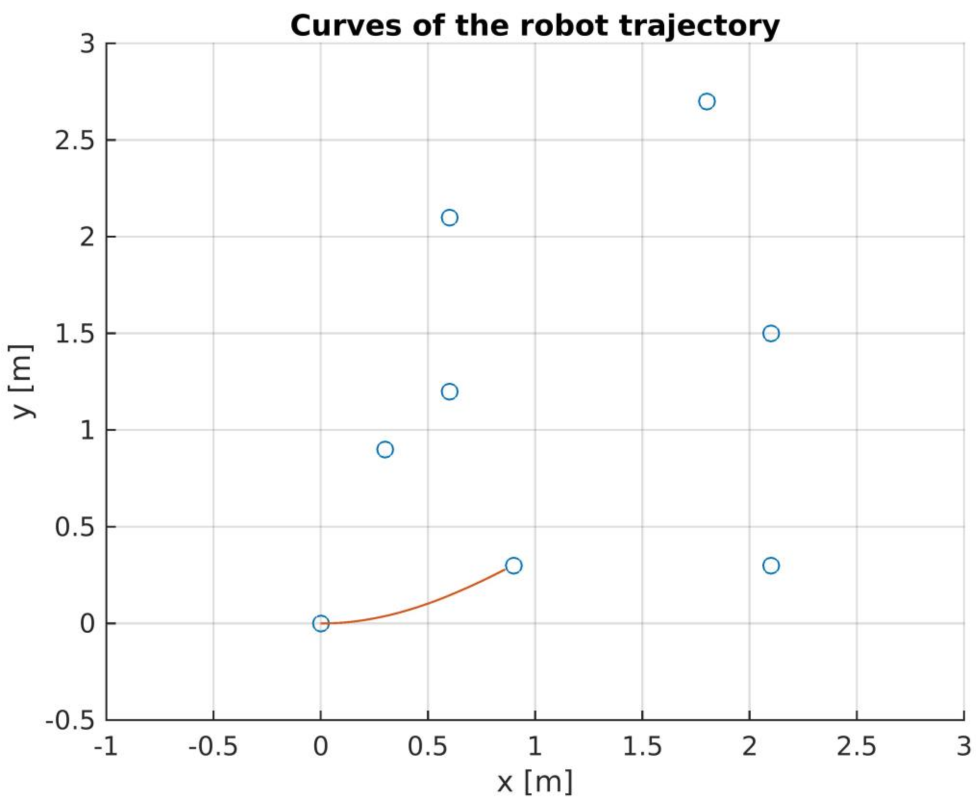
$$\begin{bmatrix} station & x & y \\ docking & 0 & 0 \\ 1 & 3 & 1 \\ 2 & 1 & 3 \\ docking & 0 & 0 \\ 3 & 7 & 5 \\ 4 & 2 & 4 \\ docking & 0 & 0 \\ 5 & 2 & 7 \\ 6 & 7 & 1 \\ docking & 0 & 0 \\ 7 & 6 & 9 \\ docking & 0 & 0 \\ end & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} station & x & y \\ docking & 0 & 0 \\ 1 & 3 & 1 \\ 2 & 1 & 3 \\ docking & 0 & 0 \\ 3 & 7 & 5 \\ 4 & 2 & 4 \\ docking & 0 & 0 \\ 5 & 2 & 7 \\ 6 & 7 & 1 \\ docking & 0 & 0 \\ 7 & 6 & 9 \\ docking & 0 & 0 \\ end & 0 & 0 \end{bmatrix} * 0.3 \Rightarrow \begin{bmatrix} station & x & y \\ docking & 0 & 0 \\ 1 & 0.9 & 0.3 \\ 2 & 0.3 & 0.9 \\ docking & 0 & 0 \\ 3 & 2.1 & 1.5 \\ 4 & 0.6 & 1.2 \\ docking & 0 & 0 \\ 5 & 0.6 & 2.1 \\ 6 & 2.1 & 0.3 \\ docking & 0 & 0 \\ 7 & 1.8 & 2.7 \\ docking & 0 & 0 \\ end & 0 & 0 \end{bmatrix}$$

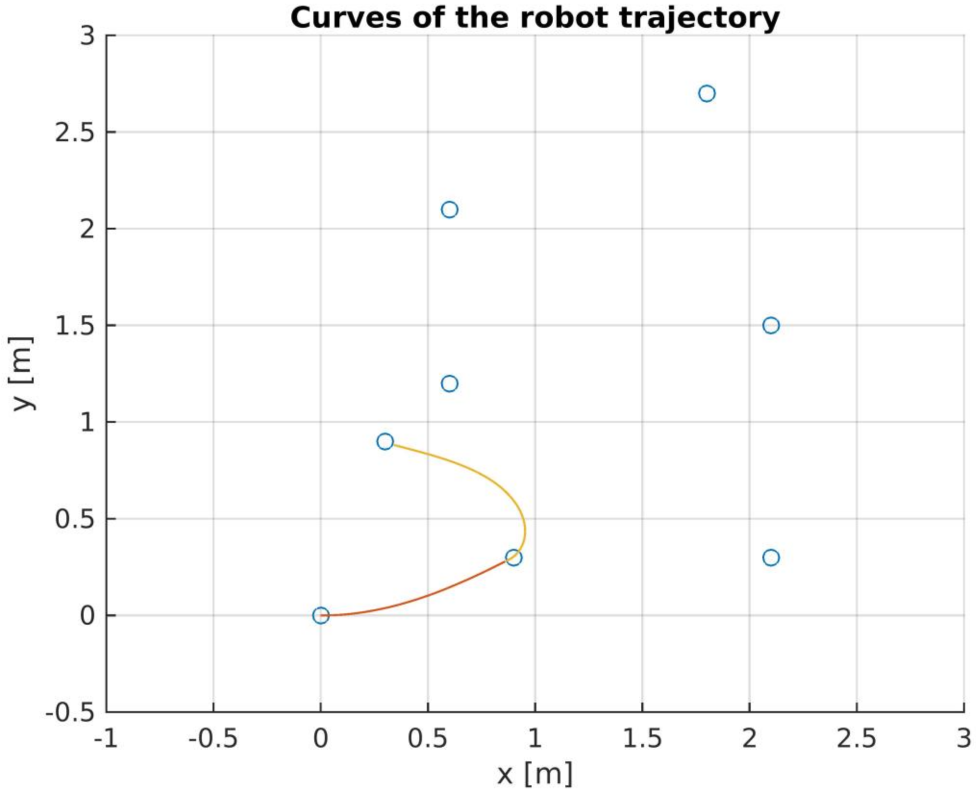
At the next page there is a Simulink blocks diagram which describes the commands for the TurtleBot. By using a non-linear controller we can examine the result of the VRP algorithm.

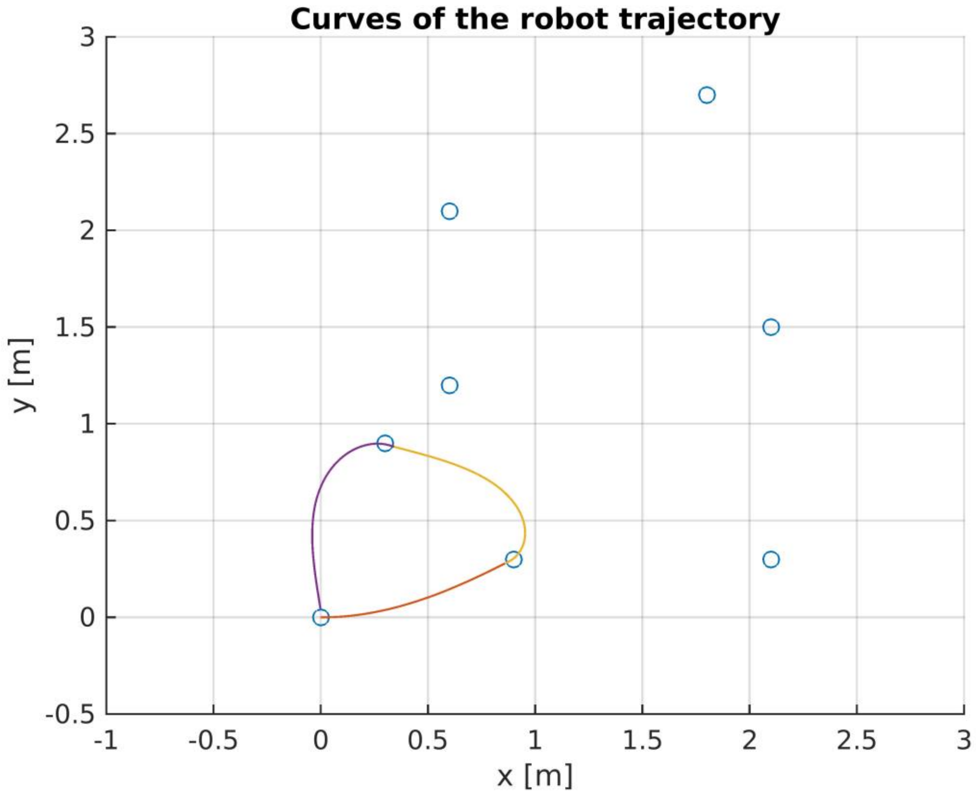

ROS
Subscribe
/odom
IsNew
Msg
Subscribe
Terminator
To Workspace1
x_robot
To Workspace4
y_robot
To Workspace2
x_y_current
Constant1
point
<X>
<Y>
<X>
<Y>
<Z>
<W>
x
y
z
w
quat2eul
yaw
Conversion
Clock
To Workspace3
time
Dot Product
Sqrt
Trigonometric Function
atan2
Turn angle needed
Scope
distance
To Workspace5
miss_distance
mod
Math Function
2*pi
stop1
atan2(sin(u),cos(u))
Bound [-pi,pi]
0.2
turn angle3
odometry input
0.68
turn angle
<= 0.01
Compare To Constant1
STOP
Stop Simulation1
ROS
Blank Message
geometry_msgs/Twist
Msg
Blank Message
Bus
:= X
:= Z
Bus
Bus Assignment
ROS
Publish
Msg
/cmd_vel/commands/velocity
Publish

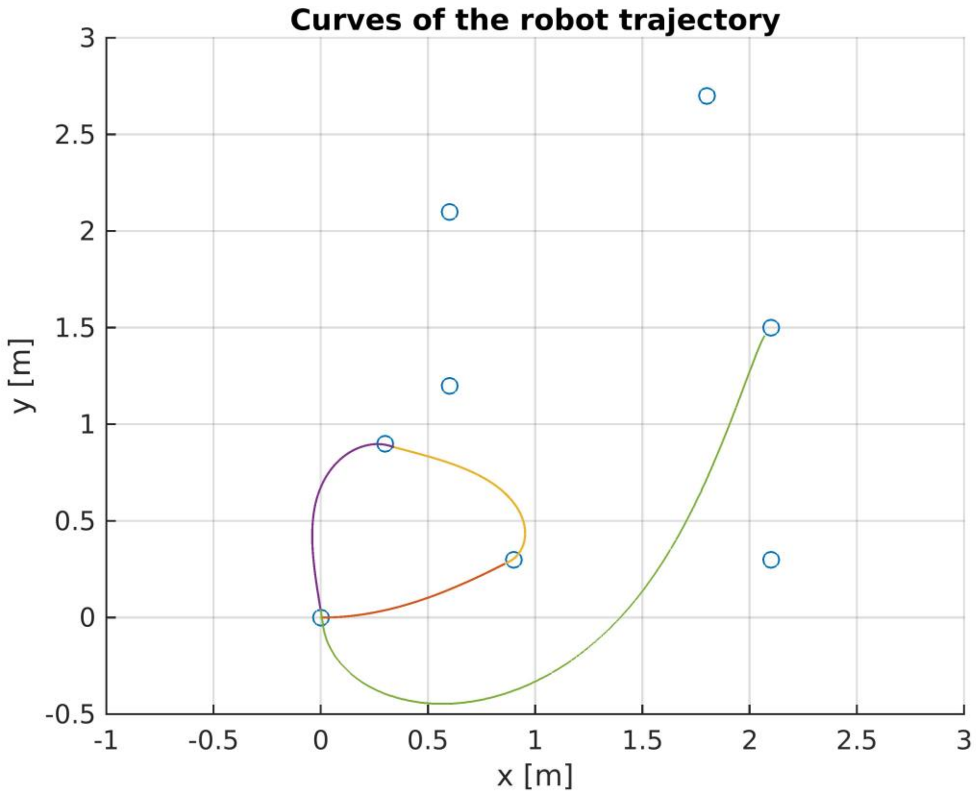
Plots-the curves that the robot makes, "step by step":

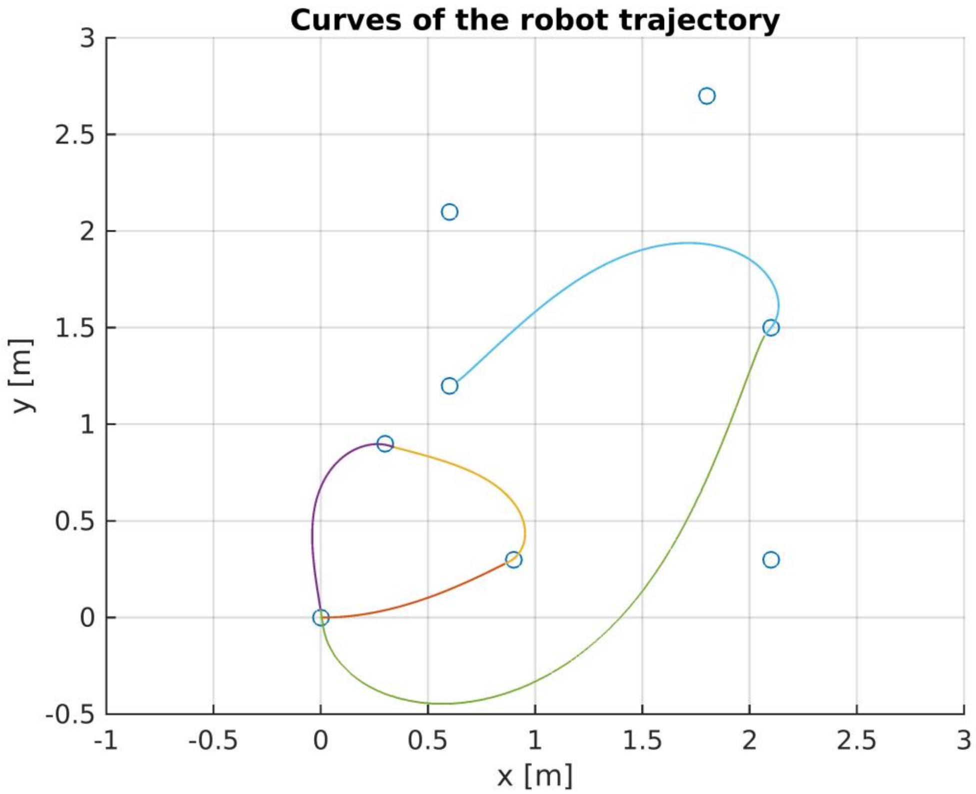


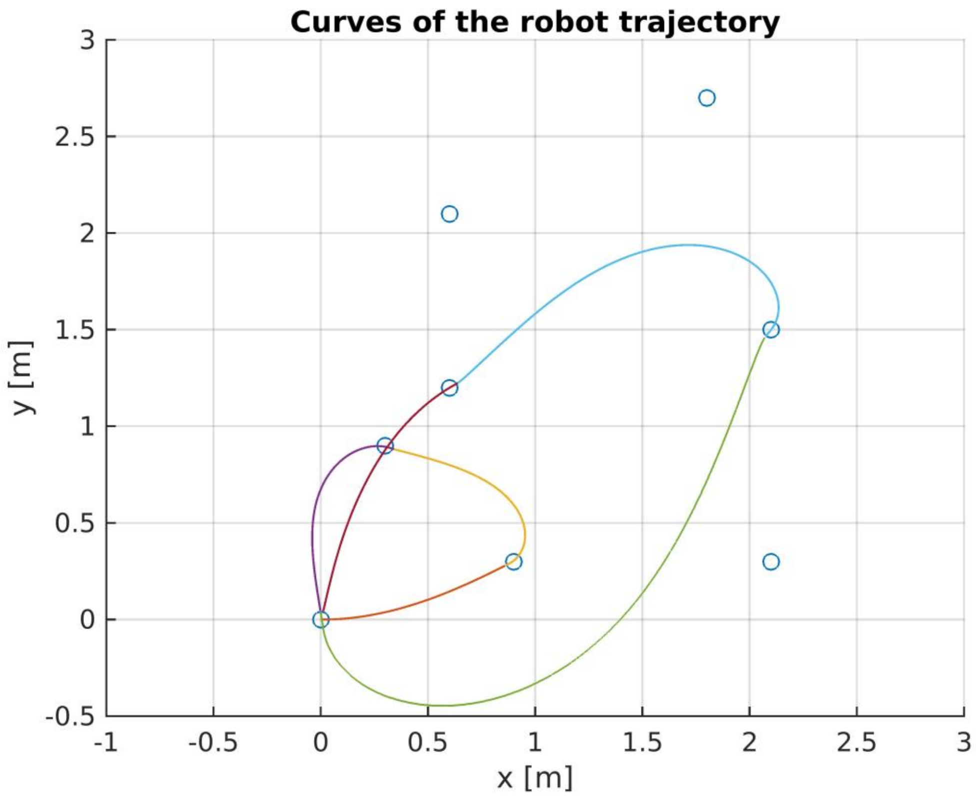



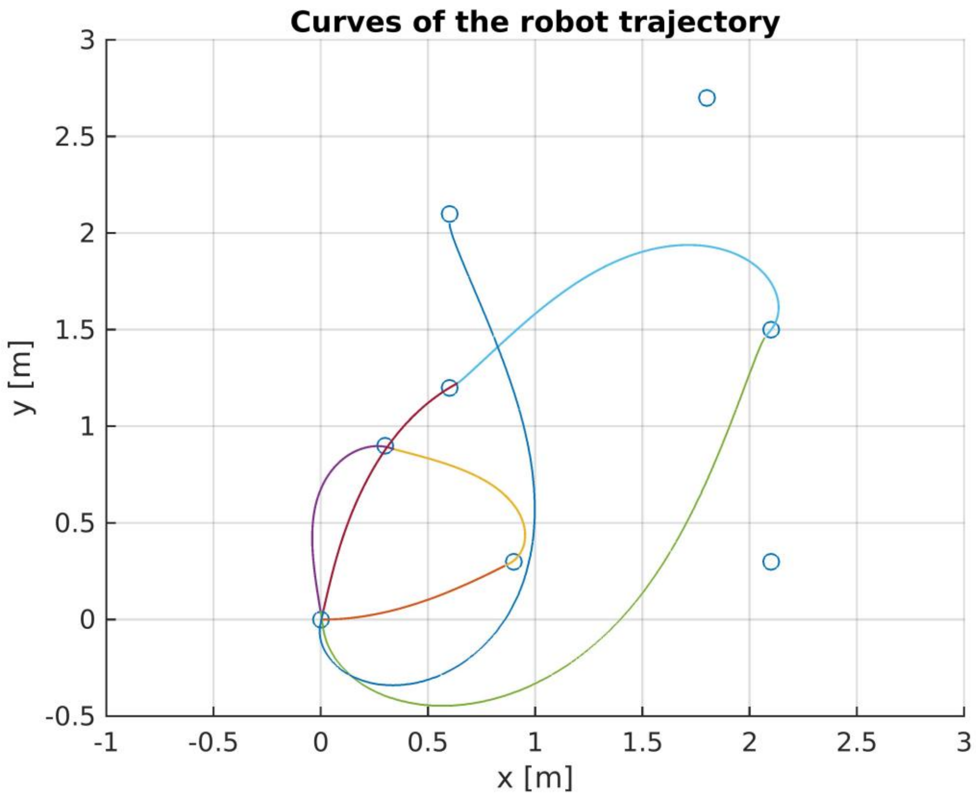

Curves of the robot trajectory
3
2.5
2
1.5
1
0.5
0
-0.5
y [m]
-1
-0.5
0
0.5
1
1.5
2
2.5
3
x [m]

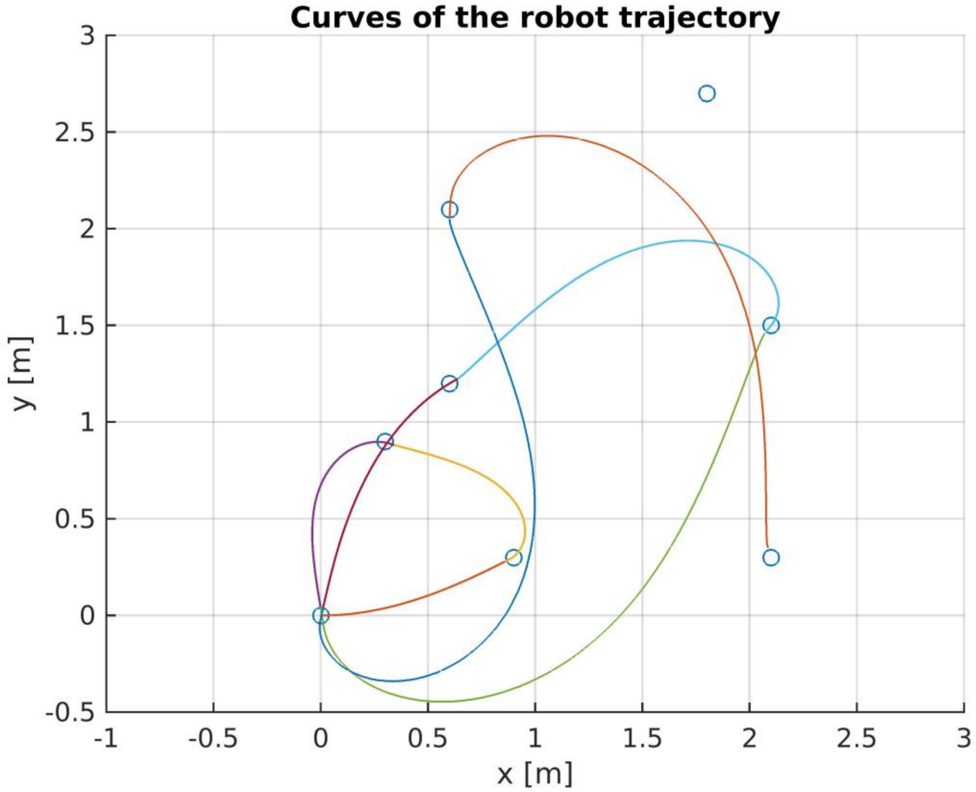

Curves of the robot trajectory
3
2.5
2
1.5
1
0.5
0
-0.5
y [m]
-1
-0.5
0
0.5
1
1.5
2
2.5
3
x [m]

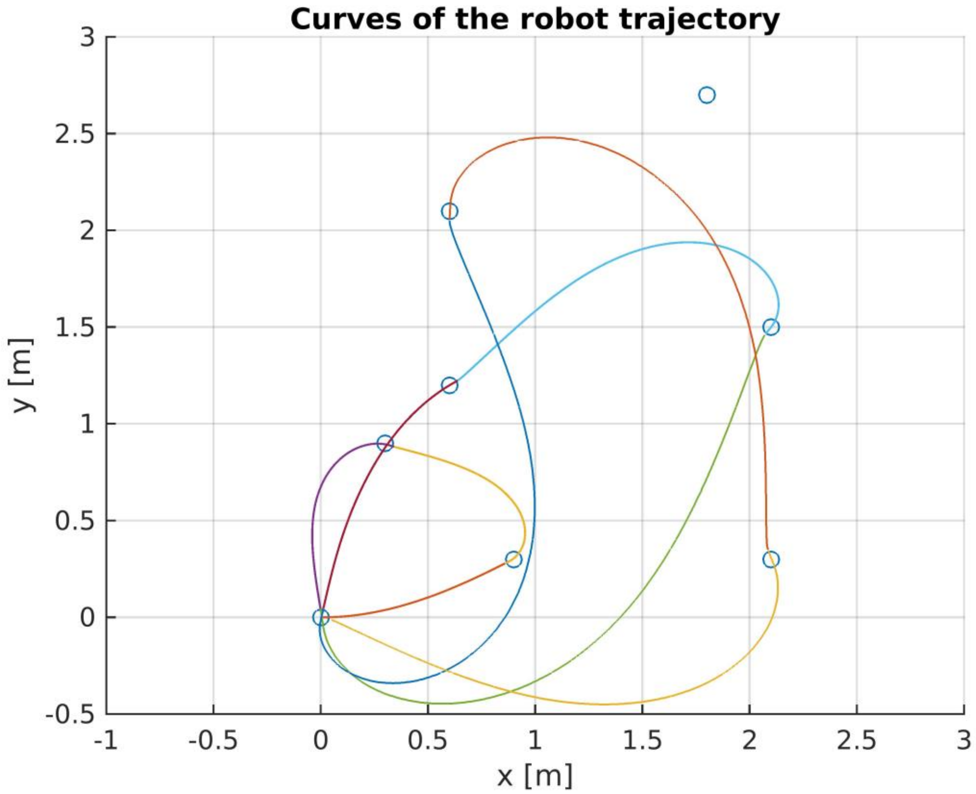

Curves of the robot trajectory
y [m]
x [m]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3

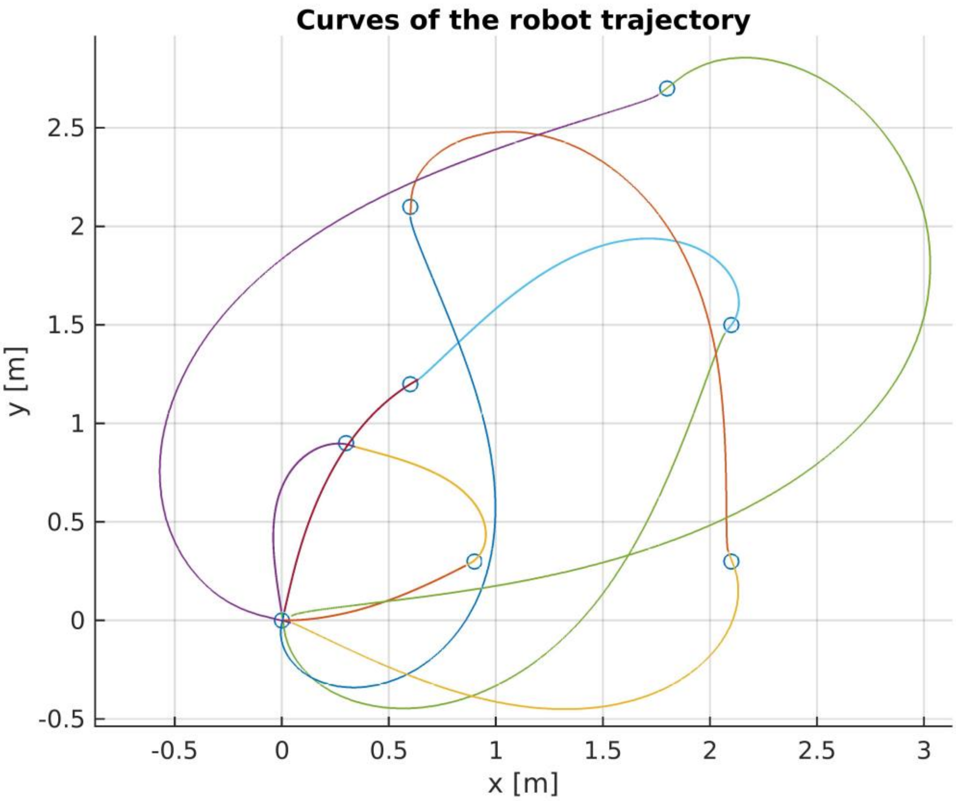

Curves of the robot trajectory
y [m]
x [m]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3


Curves of the robot trajectory
y [m]
x [m]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3


Curves of the robot trajectory
y [m]
x [m]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3


Curves of the robot trajectory
y [m]
x [m]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3


Curves of the robot trajectory
y [m]
x [m]
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-0.5
0
0.5
1
1.5
2
2.5
3




The last plot is the route from docking station to 7 and back to the docking station.

## 'Matlab' Code:

```
rosshutdown

clear
close all
clc
rosinit ('10.42.0.26')
%%main of VRP-Experiment

N = 7; %Number of stations
Point_val = 0.3*[3 7 2 1 6 7 2; 1 5 7 3 9 1 4]; %1st row X values 2st
row Y values
[out,opt_dist] = VRP(Point_val,N);

%Reset odometry - Ruben
odomresetpub = rospublisher('/mobile_base/commands/reset_odometry');
% Reset odometry
odomresetmsg = rosmessage(rostype.std_msgs_Empty);
send(odomresetpub,odomresetmsg)

soundpub = rospublisher('/mobile_base/commands/sound');
soundmsg = rosmessage(rostype.kobuki_msgs_Sound);
soundmsg.Value = 5;% Any number 0-6
out=out(2:end,:);
w=1;
i=1;

figure(1)
scatter(out(:,1),out(:,2),'o')
ylim([-0.5 3])
xlim([-1 3])
grid on
title('Curves of the robot trajectory')
xlabel('x [m]')
ylabel('y [m]')

while w>0

if i>2*N
    w=-1;
    break
end
point=out(i,:);
sim ('go_to_points');

if i>2*N
    w=-1;
end
figure(1)
hold all
plot(x_robot,y_robot)


i=i+1;
  send(soundpub,soundmsg);
end
```