THE NEOTIA UNIVERSITY

ज्ञानम् आत्म प्रदीपाय

DEPARTMENT OF ROBOTICS & AUTOMATION

Motion Planning

LAB MANUAL

THE NEOTIA UNIVERSITY
DEPARTMENT OF ROBOTICS & AUTOMATION

EXPERIMENT NO.: 1

NAME OF THE EXPERIMENT: Two assignments on Programming the Robot for applications.

OBJECTIVE: To study the Robot programming for industrial applications.

THEORY:

PROGRAM 1:

Palletizing application using AL. Begin 'Palletizing sample program' Frame in-pallet, out-pallet, part; Comment

The (1, 1) positions of the pallets and grasping position of parts;

Vector del-r1,del-c1;

Vector del-r2,del-c2;

COMMENT relatives' displacements along the rows and columns;

Scalar r1, c1, ir1, ic1;

Scalar r2, c2, ir2, ic2;

COMMENT COUNTERS;

EVENT in-pallet-empty, in-pallet-replaced;

EVENT out-pallet-full, out-pallet-replaced;

COMMENT

Here insert the frame definition for IN-PALLET and OUT-PALLET and the vector value for displacements along and recorded using robot

PROCEDURE PICK;

BEGIN

FRAME Pick-frame; Ir1=ir1+1;

```
IF ir1 =c1Tr1
    THEN
    BEGI
    N
     ir1;=1
    ic1; =
    ic1+1 IF
    ic1c1Tc1
    THEN
    BEGIN
    SINGAL     in-pallet-
    empty;   WAIT   in-
    pallet-replaced;
    ic1;=1;
    END
    ;
    END
    ;
    Pick-frame; = in-pallet+ (ir1-1)*del-r1+( ic1-
    1)*del-c1 MOVE BEHIND TO PICK
    FRAME;
    CENTRE BARM;
    AFFIX PART TO BARM;
```

```
END
PROCEDURE
PLACE BEGIN
FRAME PLACE-
frame ir2 = ir2+1;
IF ir2=c1Tr2
THEN BEGIN
ir2; =1;
ic2; = ic2+1 IF
ic2=c1Tc2
THEN BEGIN
SINGALOUT-Pallet-empty;
WAIT OUT-Pallet-replaced;
ic2=1;
END;
END;
Place-frame; out-Pallet+(ir2-1)*del-r2+( ic2-
1)*del-r2. MOVE Part To Place-frame.
OPEN BHAND To
3.0*IN UNFIX PART
FROM BARM;
END;
COMMENT THE main
```

Program, OPEN BHAND

To 3.0*IN; WHILE TRUE

DO

BEGIN

PICK;

PLACE;

END;

END;

PROGRAM 2:

Palletizing application using KAREL

PROGRAM PALLET

---Transfer workpieces from one pallet to another.

---Variables for the input pallet,

BASE 1; position—(1,1) position on

pallet. IR 1, IC 1: integer.

NR1, NC1: integer.

DR 1, DC 1:

vector. 1S1G1,

0S1G1; intger.

--Variables for the output

pallet. BASE 2; Position

IR 2, IC 2: integer.

NR2, NC2: integer.

DR 2, DC2: vector.

1S1G2, 0S1G2;

intger. ROUTINE

PICK

--Pick a workpiece from the input

Pallet. TARGET: POSITION—

targetpose BEGIN

IR1=IR1+

1 If

IR1>NR1

Then

I R1=1

IC1=IC1+

1

If IC1>NC1

Then,

IC1=1

dout (0S1G1)=true

wait for din

(1S1G1) + dout

(0S1G1)=false.

End

if

```
End
if
TARGET = BASE1
Shift (TARGET, (1R1-1)*DR1+(1C1-1)*DC1
Move near TARGET
by 50 Move to
TARGET
Close hand 1
Move away 50
and Pick
ROUTINE PLACE
--Place a workpiece on the output
pallet var TARGET:POSITION.
BEGIN IR2=
IR2+1
IfIR2>NR2
Then, IR2=1
IC2=IC2+1 If
IC2>NC2
Then,
IC2=1
dout (0SIG2)=true.
Wait for din
```

(1SIG2)+1 dout

(0SIG2)=false.

End

if

End

if

TARGET=BASE 2

Shift (TARGET, (1R2-1)*DR2+(1C2-1)*DC2

Mover near TARGET

by 50 Move to

TARGET

Close hand 1

Move away 50

end Place.

MAIN PROBLEM

BEGIN

IR1=0;

IC1=0

IR2=0;

IC2=0

--initialize other variable

--BASE 1, NR1, NC1, DR1, DC1, IS1G1,0S1G1.

--BASE 2, NR2, NC2, DR2, DC2, IS1G2,0S1G2.

--numerical pose definition

omitted here. Opened hand 1

While true do—

loop. PICK

PLACE
and while
and PALLET.

CONCLUSION: Thus, we have studied the Robot programming for industrial applications.

THE NEOTIA UNIVERSITY
DEPARTMENT OF ROBOTICS & AUTOMATION

EXPERIMENT NO.: 2

NAME OF THE EXPERIMENT: Two assignments on Programming the Robot for applications in VAL-II.

OBJECTIVE: To study the Robot programming application in VAL II.

THEORY:

PROGRAM:

Palletizing application in VAL II:

In the VAL II version of Palletizing application, the program transfer parts between two Pallets using the external binary I/O signals to request additional pallets. It communicates with the user via the system terminal asking questions and providing information on system operation. A Pallet location is taught by instructing the operator to move the robot to the corners of pallet. The program then computes all locations in the pallet.

```
PROGRAM Main ( );
    ABSTRACT:- This is the top level program to move
    ; Thee parts between two Pallets. It allows the operator
    ; To teach the Pallet locations if desired & then
    ; moves parts from one pallet to next
    ; DATA STRUCTURE
    ; in. Pallet [.]=An array of location for items on the pallet to be unloaded.
    ; in. height =approach/depart height for input pallet.
    ; in. max= The no. of items on a full input pallet.
    ; in. count= The no. of items left on this I/P pallet.
```

; out. pallet [ ] = An array of location for items on pallet to be loaded,

; out height = approach/dipalt height for O/P pallet.

; out. max= The no. of items on a full O/P pallet.

; out. count= The no. of items left on this O/P pallet

; #Safe=soft robot location reachable from

both pallets LOCAL sans in count, out count

Define binary signal no. used to control pallets transfer =1001; input signal TRUE when transfers permitted

in.ready =1002; I/P Signal TRUE when output Pallet ready

out ready =1003; I/P Signal TRUE when O/P

Pallet ready in. Change=4; O/P Signal requests

new I/P Pallet

out. Change=5; O/P Signal requests new O/P Pallet

; Ask operator about set up and teach new pallets

ifdesired PROMPT "Do you want to define the

pallet (Y/N)"Sans IF Sans =="Y"THEN

DETACH ( ); Detach robot from program control

TYPE "Use the PENDANT to teach the I/P Pallet

location" CALL set up. Pallet (in. count, in pallet [

], out. height)

TYPE "Press the comp. button on the PENDANT to

continue " ATTACH ( )

END

```
; Initialize transfer
data transfer count
=0
in.
count=0
out
count=0
; wait for transfer signal; then start the pallet
transfer MOVES # Safe
TYPE “waiting fortransfersignal”/S
WAIT SIG (transfer)

TYPE “ starting transfer”,/c2
Main, loop transferring one pallet to another, requesting;
;new pallets as necessary; Quit when
transfer signal becomes FALSE
WHILE. SIG (transfer)
DO IF in. count< = 0
THEN SIGNAL in.
change
WAIT SIG (-
in.ready) WAIT
SIG (in. ready) in.
count= in. max
```

```
END
IF out. Count < 0
THEN SIGNAL
OUT. Change WAIT
SIG (-out. ready)
WAIT SIG (out.
ready) out. count=
out. max END
OPEN
APPROS in Pallet [in. count],
in.height. SPEED 20
MOVES in. Pallet [in. count]
CLOSE I
DEPARTS in.
height in. count =
in. count-1
; Place output part
APPROS
out SPEED
20
MOVES out. Pallet [out.
count] OPEN I
DEPARTS out.
```

```
height out. count =
out. count-1
; Count transfer and display it.
transfer. Count = transfer.
Count+1
TYPE N, "Number of parts transferred," / I8,
 transfer. count END;
; All done transferring parts, move robots to safe
place and quit MOVES # Safe
END
PROGRAM set up pallet (count. Array [ ],approach)
; INPUT PARM ; NONE;
;OUTPUT PARM ; Count = No. of items on their
pallet. array [ ]= Array containing the pallet
location
approach = The approach height for their
Pallet. Local r1; 11, 1r, ap, t [], ncol,
nrow
Local row, COL.cs, rs, i, frame
Ask operator to teach pallet location
CALL teach. Point ("Upper left pallet
location",Ll) CALL teach. Point ("Lower left
pallet position", Ll) CALL teach. Point
```

```
("Lower right pallet position", Lr)
CALL teach. Point ("approach height above
pallet", ap) PROMPT "Enter the no. of columns
(left to right);" ncol PROMPT "Enter the no. of
columns (top to bottom);" nrow Count =
ncol*nrow
; set up to compute pallet
location cs = 0
IF ncol>1 THEN
cs =DISTANCE
(l11lr)/(ncol=1) END
; Compute frame values
SET frame = FRAME
(ll,lr,l11,l11) approach = DZ [
INVERSE (frame);ap]
DECOMPOSE t [1]=l1L
LOOP to compute array
value i =1
For row=0 to
nrow-1 For col=0
to ncol-1
SET array [i]= frame.;TRANS ( row*rs,
 col*cs,0,t[4],t[5],t[6] i = i+1
```

END

END

RETUR

N END.

CONCLUSION: Thus, we have studied the Robot programming for application in VAL II.

THE NEOTIA UNIVERSITY
DEPARTMENT OF ROBOTICS & AUTOMATION

EXPERIMENT NO.: 3

NAME OF THE EXPERIMENT: Exercise on Robotic Simulation Software.

OBJECTIVE: To study the Robot path planning using Robotic simulation software.

THEORY:

The locus of points along the path defines the sequence of position through which the robot will move its wrist. In most applications, an end effector is attached to the wrist and program can be considered to be the path in space through which the end effector is to be moved by the robot.

Since, the robot consists of several joint (axes) linked together, the definition of the path in space in effect requires that the robot move its axes through various positions in order to follow that path for a robot with six axes, each point in the path consists of six coordinates value corresponds to the position of one joint. There are basic robot anatomies; Polar, Cylindrical, Cartesian and Jointed Arm.

Each one of three axes associated with the arm and body configuration and two or three additional joints are associated with wrist. The arm and body joint determines the general position in space of the end effector and the wrist determines its orientation. If we think of a joint in space in the robot program as a position and orientation of the end effector, there is usually more than one possible set of joint coordinate values that can be used for the robot to reach that point.

For example, there are two alternative axis configurations that can be used by the jointed arm shown in figure to achieve the target point indicated.

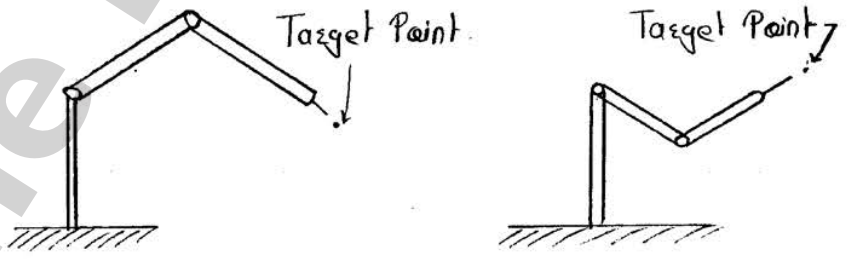


fig ②:→ Two alternative axis configurations with end effector located at desired target point.

As shown in figure (a) that; although the target point has been reached by both of alternative axis configurations, there is a difference in the orientation of the wrist with respect to the point. We must conclude from this that the specification of the joint coordinates of the robot does define only one point in a space that corresponds to that set of coordinate values. Point specified in this fashion are said to be joint coordinates. Accordingly, an advantage of defining robot program in this way is that is simultaneously specifies the position and orientation of the end effector at each point in the path.

Let's consider the problem of defining a sequence of points in space. We will assume that these points are defined by specifying the joint coordinates as described above. Although, this method of specification will not affect the issue we are discussing here for a sake of simplicity, lets assume that we are programming a point-to-point Cartesian robot with only two axes and only two addressable point is one of the available points (as determine by the control revolution) that can be commended to go to that point. Figure (b) shows the four points (possible points) in the robot's rectangular space. A program of this robot to start in lower left hand corner and traverse the perimeter of the rectangle could be written as follows;

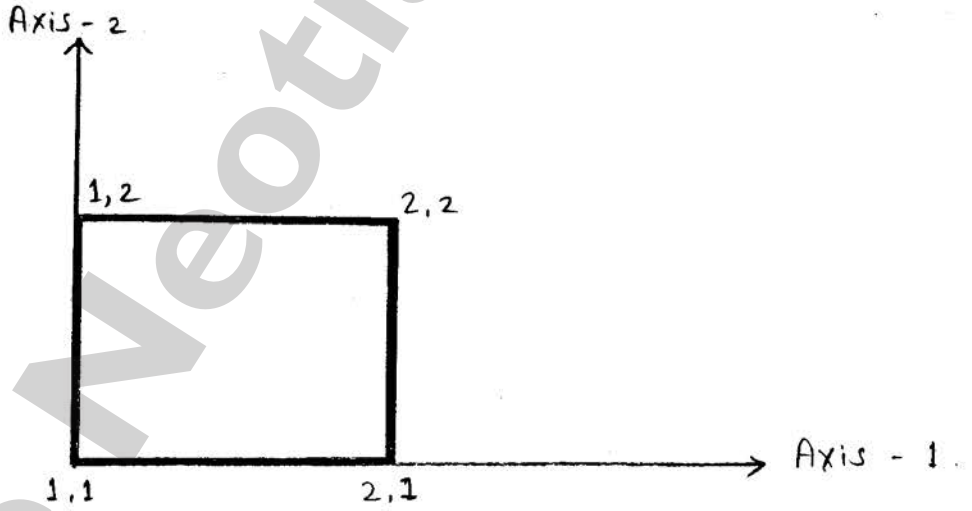


fig (b) - Robot Workspace.

STEP	MOVE	COMMENTS
1	1, 1	Move to lower left corner.
2	2, 1	Move to lower right corner.
3	2, 2	Move to upper right corner.
4	1, 2	Move to upper left corner.
5	1, 1	Move back to start position.

The point designation corresponds to the x, y- coordinates positions in the Cartesian axis system. In this example, using a robot with two orthogonal slides and only two addressable points per axis, the definition of points in space corresponds exactly with joint coordinate's values.

CONCLUSION: Thus, we have studied the robot path planning using simulation control software.

THE NEOTIA UNIVERSITY
DEPARTMENT OF ROBOTICS & AUTOMATION

EXPERIMENT NO.: 4

NAME OF THE EXPERIMENT: Plan Mobile Robot Paths using RRT

OBJECTIVE: This experiment shows how to use the rapidly-exploring random tree (RRT) algorithm to plan a path for a vehicle through a known map. Special vehicle constraints are also applied with a custom state space. You can tune your own planner with custom state space and path validation objects for any navigation application.

THEORY:

Functions:

plannerRRT	Create an RRT planner for geometric planning
plannerRRTStar	Create an optimal RRT path planner (RRT*)
plannerAStarGrid	A* path planner for grid map
plannerHybridAStar	Hybrid A* path planner

Load Occupancy Map:

Load an existing occupancy map of a small office space. Plot the start and goal poses of the vehicle on top of the map.

Code:

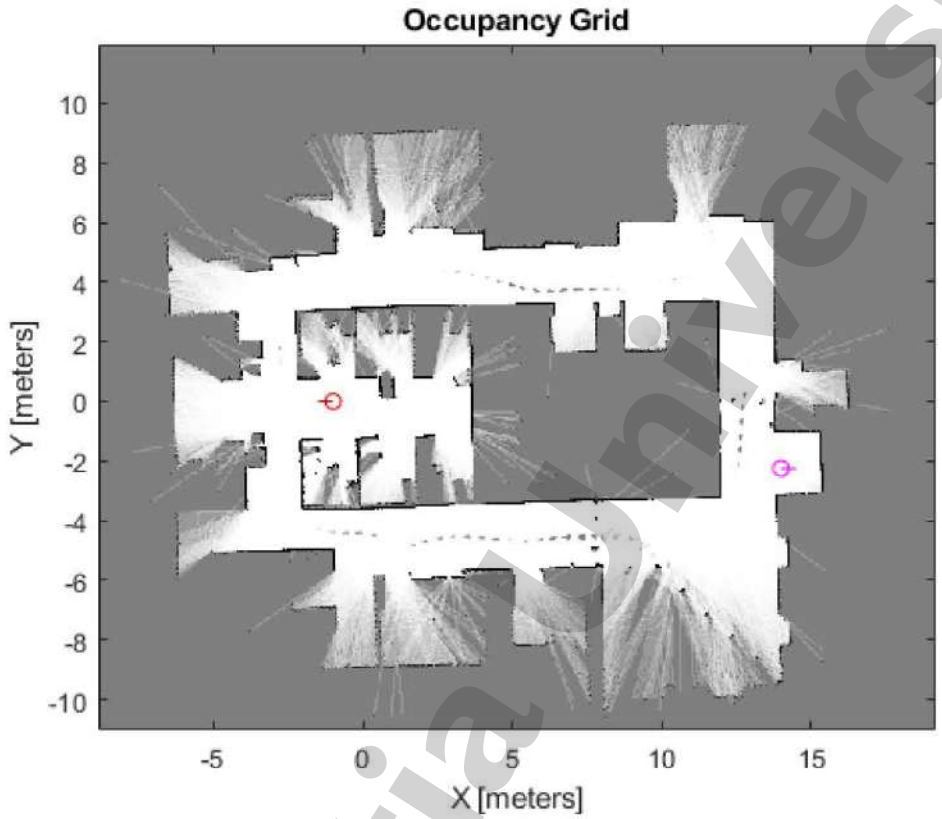
```
load("office_area_gridmap.mat", "occGrid")
show(occGrid)

% Set the start and goal poses
start = [-1.0, 0.0, -pi];
goal = [14, -2.25, 0];

% Show the start and goal positions of the robot
hold on
plot(start(1), start(2), 'ro')
plot(goal(1), goal(2), 'mo')

% Show the start and goal headings
r = 0.5;
plot([start(1), start(1) + r*cos(start(3))], [start(2), start(2) + r*sin(start(3))],
'r-' )
```

```
plot([goal(1), goal(1) + r*cos(goal(3))], [goal(2), goal(2) + r*sin(goal(3))], 'm-' )
hold off
```



Define State Space:

Specify the state space of the vehicle using a state Space Dubins object and specifying the state bounds. This object limits the sampled states to feasible Dubins curves for steering a vehicle within the state bounds. A turning radius of 0.4m allows for tight turns in this small environment.

```
bounds = [occGrid.XWorldLimits; occGrid.YWorldLimits; [-pi pi]];

ss = stateSpaceDubins(bounds);
ss.MinTurningRadius = 0.4;
```

Plan the Path:

To plan a path, the RRT algorithm samples random states within the state space and attempts to connect a path. These states and connections need to be validated or excluded based on the map constraints. The vehicle must not collide with obstacles defined in the map.

Create a validator Occupancy Map object with the specified state space. Set the Map property to the loaded occupancy Map object. Set a Valdiation Distance of 0.05m. This distance discretizes the path connections and checks obstacles in the map based on this.

```
stateValidator = validatorOccupancyMap(ss);
stateValidator.Map = occGrid;
stateValidator.ValidationDistance = 0.05;
```

Create the path planner and increase the max connection distance to connect more states. Set the maximum number of iterations for sampling states.

```
planner = plannerRRT(ss, stateValidator);
planner.MaxConnectionDistance = 2.0;
planner.MaxIterations = 30000;
```

Customize the Goal Reached function. This example helper function checks if a feasible path reaches the goal within a set threshold. The function returns true when the goal has been reached, and the planner stops.

```
planner.GoalReachedFcn = @exampleHelperCheckIfGoal;
    function isReached = exampleHelperCheckIfGoal(planner, goalState, newState)

        isReached = false;

        threshold = 0.1;

        if planner.StateSpace.distance(newState, goalState) < threshold

            isReached = true;

        end

    end
```

Plan the path between the start and goal. Because of the random sampling, this example sets the rng seed for consistent results.

```
rng(0,'twister')

[pthObj, solnInfo] = plan(planner, start, goal);
```

Plot the Path:

Show the occupancy map. Plot the search tree from the soln Info. Interpolate and overlay the final path.

Code:

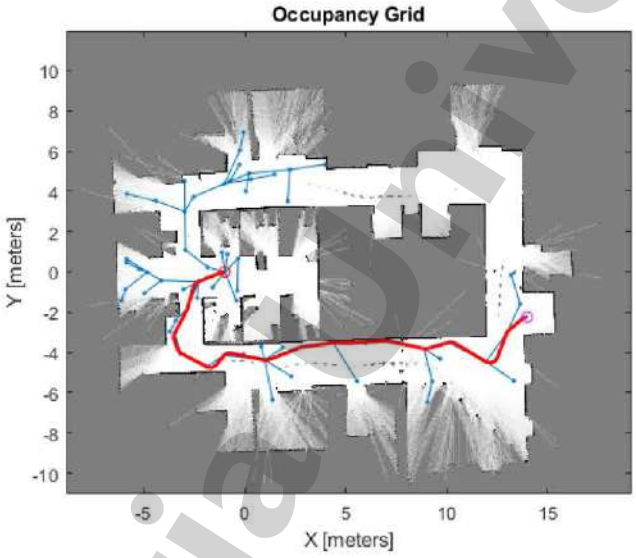
```
show(occGrid)
hold on

% Search tree
```

```
plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-');

% Interpolate and plot path
interpolate(pthObj,300)
plot(pthObj.States(:,1), pthObj.States(:,2), 'r-', 'LineWidth', 2)

% Show the start and goal in the grid map
plot(start(1), start(2), 'ro')
plot(goal(1), goal(2), 'mo')
hold off
```



Customize Dubins Vehicle Constraints:

To specify custom vehicle constraints, customize the state space object. This example uses Example Helper State Space One Sided Dubins, which is based on the state Space Dubins class. This helper class limits the turning direction to either right or left based on a Boolean property, Go Left. This property essentially disables path types of the dubins Connection object uses (see dubins Connection. Disabled Path Types).

Create the state space object using the example helper. Specify the same state bounds and give the new Boolean parameter as true (left turns only).

```
% Only making left turns
goLeft = true;

% Create the state space
ssCustom = ExampleHelperStateSpaceOneSidedDubins(bounds, goLeft);
ssCustom.MinTurningRadius = 0.4;
```

Plan The Path:

Create a new planner object with the custom Dubins constraints and a validator based on those constraints. Specify the same GoalReached function.

```
stateValidator2 = validatorOccupancyMap(ssCustom);
stateValidator2.Map = occGrid;
stateValidator2.ValidationDistance = 0.05;

planner = plannerRRT(ssCustom, stateValidator2);
planner.MaxConnectionDistance = 2.0;
planner.MaxIterations = 30000;
planner.GoalReachedFcn = @exampleHelperCheckIfGoal;
```

Plan the path between the start and goal. Reset the rng seed again.

```
rng(0, 'twister')
[pthObj2, solnInfo] = plan(planner, start, goal);
```

Plot the path:

Draw the new path on the map. The path should only execute left turns to reach the goal. This example shows how you can customize your constraints and still plan paths using the generic RRT algorithm.

Code:

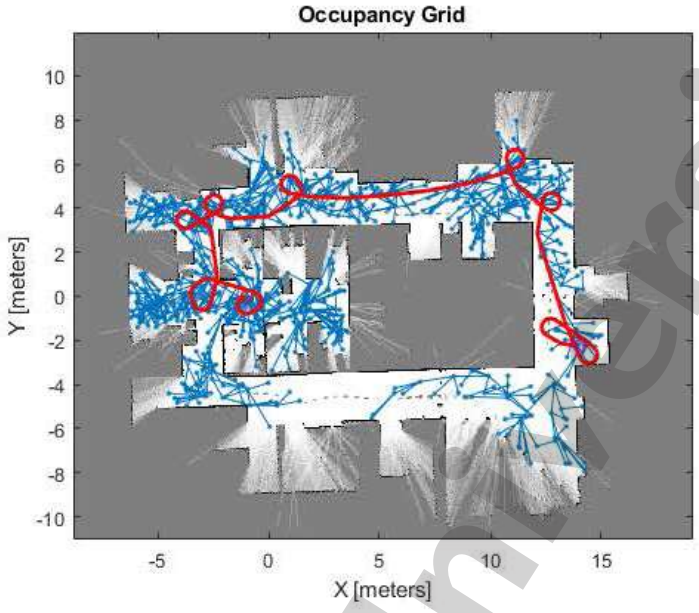
```
figure
show(occGrid)

hold on

% show the search tree
plot(solnInfo.TreeData(:,1), solnInfo.TreeData(:,2), '.-'); % tree expansion

% draw path (after the path is interpolated)
pthObj2.interpolate(300)
plot(pthObj2.States(:,1), pthObj2.States(:,2), 'r-', 'LineWidth', 2)

% Show the start and goal in the grid map
plot(start(1), start(2), 'ro')
plot(goal(1), goal(2), 'mo')
hold off
```



CONCLUSION: Thus, we have studied how to use the rapidly-exploring random tree (RRT) algorithm to plan a path for a vehicle through a known map. Special vehicle constraints are also applied with a custom state space. You are able to tune your own planner with custom state space and path validation objects for any navigation application.

THE NEOTIA UNIVERSITY
DEPARTMENT OF ROBOTICS & AUTOMATION

EXPERIMENT NO.: 5

NAME OF THE EXPERIMENT: Path Following with obstacle avoidance in Simulink.

OBJECTIVE: To study about Path Following with obstacle avoidance in Simulink.

THEORY:

This example uses a model that implements a path following controller with obstacle avoidance. The controller receives the robot pose and laser scan data from the simulated robot and sends velocity commands to drive the robot on a given path. You can adjust parameters while the model is running and observe the effect on the simulated robot.

Sensor Models

Accelerometers

accelparams	Accelerometer sensor parameters

Gyroscopes

allanvar	Allan variance
gyroparams	Gyroscope sensor parameters

Magnetometers

magparams	Magnetometer sensor parameters
magcal	Magnetometer calibration coefficients

Satellites

gnssconstellation	Satellite locations at specified time
lookangles	Satellite look angles from receiver and satellite positions
pseudoranges	Pseudoranges between GNSS receiver and satellites
receiverposition	Estimate GNSS receiver position and velocity
skyplot	Plot satellite azimuth and elevation data

Other

gnssSensor	Simulate GNSS to generate position and velocity readings
altimeterSensor	Altimeter simulation model
gpsSensor	GPS receiver simulation model
imuSensor	IMU simulation model
insSensor	Inertial navigation system and GNSS/GPS simulation model
rangeSensor	Simulate range-bearing sensor readings
wheelEncoderUnicycle	Simulate wheel encoder sensor readings for unicycle vehicle
wheelEncoderBicycle	Simulate wheel encoder sensor readings for bicycle vehicle
wheelEncoderDifferentialDrive	Simulate wheel encoder sensor readings for differential drive vehicle
wheelEncoderAckermann	Simulate wheel encoder sensor readings for Ackermann vehicle
kinematicTrajectory	Rate-driven trajectory generator
timescope	Display time-domain signals
waypointTrajectory	Waypoint trajectory generator
nmeaParser	Parse data from standard NMEA sentences sent from GNSS receivers
gpsdev	Connect to a GPS receiver connected to host computer

Localization and Pose Estimation

Multisensor Positioning

ahrsfilter	Orientation from accelerometer, gyroscope, and magnetometer readings
ahrs10filter	Height and orientation from MARG and altimeter readings
complementaryFilter	Orientation estimation from a complementary filter
ecompass	Orientation from magnetometer and accelerometer readings
imufilter	Orientation from accelerometer and gyroscope readings
insfilter	Create inertial navigation filter
insfilterAsync	Estimate pose from asynchronous MARG and GPS data
insfilterErrorState	Estimate pose from IMU, GPS, and monocular visual odometry (MVO) data
insfilterMARG	Estimate pose from MARG and GPS data
insfilterNonholonomic	Estimate pose with nonholonomic constraints
tunerconfig	Fusion filter tuner configuration options
tunerPlotPose	Plot filter pose estimates during tuning

Particle Filter State Estimation

stateEstimatorPF	Create particle filter state estimator
getStateEstimate	Extract best state estimate and covariance from particles
predict	Predict state of robot in next time step
correct	Adjust state estimate based on sensor measurement

Scan Matching

matchScans	Estimate pose between two laser scans
matchScansGrid	Estimate pose between two lidar scans using grid-based search
matchScansLine	Estimate pose between two laser scans using line features
transformScan	Transform laser scan based on relative pose
lidarScan	Create object for storing 2-D lidar scan

Monte Carlo Localization

monteCarloLocalization	Localize robot using range sensor data and map
lidarScan	Create object for storing 2-D lidar scan
getParticles	Get particles from localization algorithm
odometryMotionModel	Create an odometry motion model
likelihoodFieldSensorModel	Create a likelihood field range sensor model
resamplingPolicyPF	Create resampling policy object with resampling settings

Pose Graphs

poseGraph	Create 2-D pose graph
poseGraph3D	Create 3-D pose graph
addPointLandmark	Add landmark point node to pose graph
addRelativePose	Add relative pose to pose graph
edgeNodePairs	Edge node pairs in pose graph
edgeConstraints	Edge constraints in pose graph
edgeResidualErrors	Compute pose graph edge residual errors
findEdgeID	Find edge ID of edge
nodeEstimates	Poses of nodes in pose graph
optimizePoseGraph	Optimize nodes in pose graph
removeEdges	Remove loop closure edges from graph
show	Plot pose graph
trimLoopClosures	Optimize pose graph and remove bad loop closures

Wheel Encoder Odometry

wheelEncoderOdometryAckermann	Compute Ackermann vehicle odometry using wheel encoder ticks and steering angle
wheelEncoderOdometryBicycle	Compute bicycle odometry using wheel encoder ticks and steering angle
wheelEncoderOdometryDifferentialDrive	Compute differential-drive vehicle odometry using wheel encoder ticks
wheelEncoderOdometryUnicycle	Compute unicycle odometry using wheel encoder ticks and angular velocity

Mapping

binaryOccupancyMap	Create occupancy grid with binary values
occupancyMap	Create occupancy map with probabilistic values
occupancyMap3D	Create 3-D occupancy map
mapLayer	Create map layer for *N*-dimensional data
multiLayerMap	Manage multiple map layers

Other

buildMap	Build occupancy map from lidar scans
checkOccupancy	Check locations for free, occupied, or unknown values
exportOccupancyMap3D	Import an octree file as 3D occupancy map
getOccupancy	Get occupancy value of locations
getMapData	Retrieve data from map layer
importOccupancyMap3D	Import an octree file as 3D occupancy map
inflate	Inflate each occupied grid location
insertRay	Insert ray from laser scan observation
insertPointCloud	Insert 3-D points or point cloud observation into map
mapClutter	Generate map with randomly scattered obstacles
mapMaze	Generate random 2-D maze map
move	Move map in world frame
occupancyMatrix	Convert occupancy grid to double matrix
raycast	Compute cell indices along a ray
rayIntersection	Find intersection points of rays and occupied map cells
setOccupancy	Set occupancy value of locations
setMapData	Assign data to map layer
syncWith	Sync map with overlapping map
show	Show grid values in a figure
updateOccupancy	Integrate probability observations at locations

SLAM

lidarSLAM	Perform localization and mapping using lidar scans
addScan	Add scan to lidar SLAM map
buildMap	Build occupancy map from lidar scans
removeLoopClosures	Remove loop closures from pose graph
scansAndPoses	Extract scans and corresponding poses
show	Plot scans and robot poses

Motion Planning

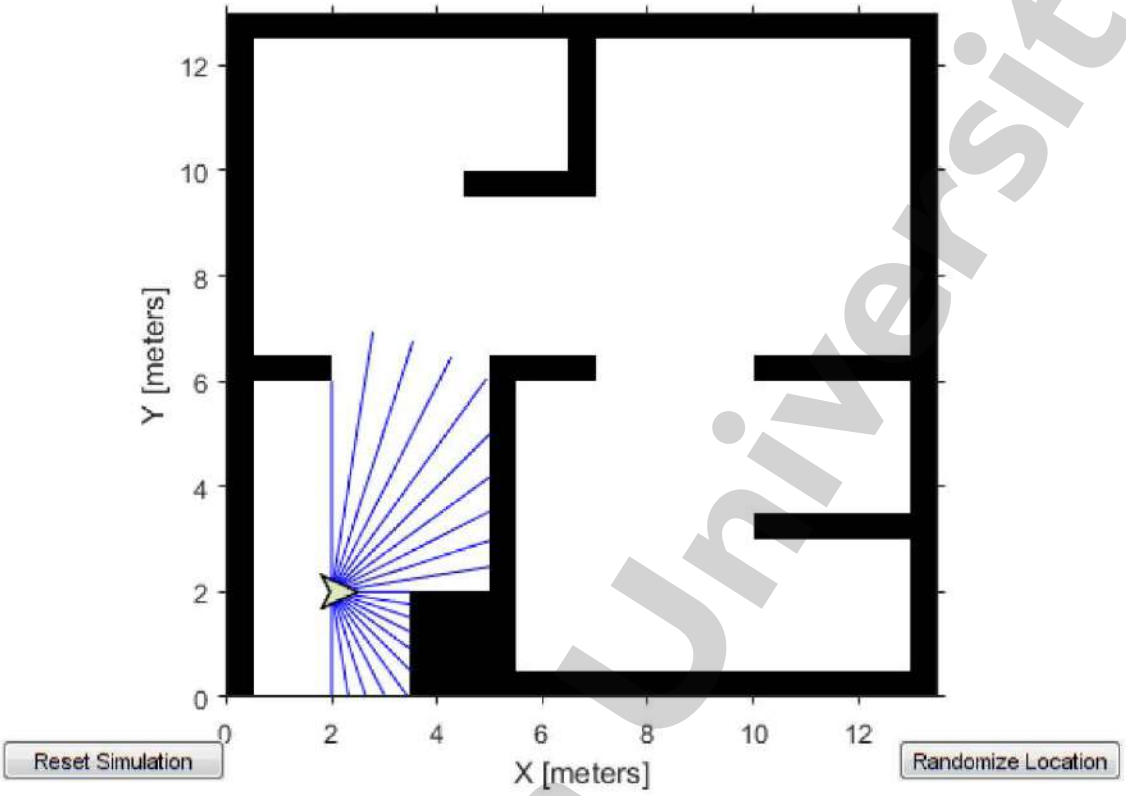
Paths and Path Segments

navPath	Planned path
dubinsConnection	Dubins path connection type
dubinsPathSegment	Dubins path segment connecting two poses
reedsSheppConnection	Reeds-Shepp path connection type
reedsSheppPathSegment	Reeds-Shepp path segment connecting two poses

START A ROBOT SIMULATOR:

Start a simple MATLAB-based simulator:

- Type rosinit (ROS Toolbox) at the MATLAB command line. This creates a local ROS master with network address (URI) of http://localhost:11311.
- Type Example Helper Simulink Robot ROS ('Obstacle Avoidance') to start the Robot Simulator. This opens a figure window:



This MATLAB-based simulator is a ROS-based simulator for a differential-drive robot. The simulator receives and sends messages on the following topics:

- It receives velocity commands, as messages of type geometry_msgs/Twist, on the /mobile_base/commands/velocity topic
- It sends ground truth robot pose information, as messages of type nav_msgs/Odometry, to the /ground_truth_pose topic
- It sends laser range data, as messages of type sensor_msgs/LaserScan, to the /scan topic

CONCLUSION: Thus have studied about path Following with obstacle avoidance in Simulink.

Courtesy: https://in.mathworks.com/& Google