**−Table of Contents**

* [MapHelper](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment8#maphelper)
* [Tree Build](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment8#tree_build)
* [Path Construction](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment8#path_construction)
* [Constructed Graph](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment8#constructed_graph)
* [Goto\_Pose video](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment8#goto_pose_video)
* [Move base as global planner](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment8#move_base_as_global_planner)
* [Move base video](http://robot.cmpe.boun.edu.tr/~cmpe565/doku.php?id=mehmetalpsarkisla:assignment8#move_base_video)

**MapHelper**

I added a psudo 0.2 inflatation by sending true if 0.1(10\*0.01 resolution) radius of the pixel is occupied :   
MapHelper.cpp:

bool MapHelper::isOccupied(int rowIndex, int colIndex)

{

int linearIndex = rowIndex \* width + colIndex;

int occupancy = (int)occupancyGrid.data[linearIndex];

for(int i = -10; i < 11 ; i++) {//0.2 inflaiton

for(int j = -10; j < 11 ; j++) {

linearIndex = (rowIndex+j) \* width + (colIndex+i);

occupancy = (int)occupancyGrid.data[linearIndex];

if (occupancy > 0 || occupancy == -1)

{

return true;

}

}

}

return false;

}

Formun Üstü

Formun Altı

**Tree Build**

My algorithm is as follows:

* Get random point between (4.5 / -4.5 , 7.5 / -7.5)
* Bring that random point to step size radius of the its neighbor Vertex(direction is not changed) , if the point is already in the radius this step is ignored.

**My step\_size is 0.5 , I started with 2 and lower the size until a good path is constructed without taking too much time.**

* Draw a line from new point to its neighbor Vertex

**I used Bresenham's line algorithm(given below) to construct line and check each point with isOccupied function given above**

* Check if line passes any occupied points. If it passes ignore rest and start again. If it is not add new point to tree.
* Draw a line between new point and TargetVertex. Check if line passes any occupied points. Check if line lenght is smaller than targetCloseEnough.
* If line does not colline with occupied points and leng is smaller than targetCloseEnough. End algorithm. If not expand tree until it does.

MyRRTSolver::build() :

void MyRRTSolver::build(){

// TODO: implement random tree building function

// NOTE: you are supposed to add vertices and edges to graph data structure

// create initial vertex at initial position

initialVertex = new MyVertex(initialx, initialy);

targetVertex = new MyVertex(targetx, targety);

// add initial vertex to graph

graph.addVertex(initialVertex);

bool Nonvalid ;

//check validity of the target

MyVertex\* newValidVertex = new MyVertex(targetx, targety);

Nonvalid = (\*mapHelper).isOccupied(newValidVertex->x , newValidVertex->y);

if(Nonvalid){ std::cout << "Target Non valid " << std::endl; return ;}else {std::cout << "Target valid " << std::endl;}

//check validity of the inital

newValidVertex = new MyVertex(initialx, initialy);

Nonvalid = (\*mapHelper).isOccupied(newValidVertex->x , newValidVertex->y);

if(Nonvalid){ std::cout << "Initial Non valid " << std::endl; return ;}else {std::cout << "Initial valid " << std::endl;}

int i = 0 ;

// DO: construct the RRT tree until the newly created node is close enough the target position and can have valid edge

while ((MyEdge::calDistance(targetVertex,newValidVertex) > targetCloseEnough) || bhm\_line((\*mapHelper).getColIndex(newValidVertex->x), (\*mapHelper).getRowIndex(newValidVertex->y),(\*mapHelper).getColIndex(targetVertex->x), (\*mapHelper).getRowIndex(targetVertex->y)) ){

//first get a random vertex

newValidVertex = getRandomConf() ;

double step\_size = 0.5 ;

if(MyEdge::calDistance(newValidVertex , graph.getNearestVertex(newValidVertex)) > step\_size){

double m = ((graph.getNearestVertex(newValidVertex)->y) - newValidVertex->y )/((graph.getNearestVertex(newValidVertex)->x) - newValidVertex->x) ;

double stepX = step\_size / sqrt(1 + m \* m) ;

newValidVertex = new MyVertex(stepX , m \* stepX) ; //index corrected with smaller step same direction

}

//check if edge is valid (dont go outside map) Bresenhams-Line-Algorithm (it also checks the point itself)

Nonvalid = bhm\_line((\*mapHelper).getColIndex(newValidVertex->x), (\*mapHelper).getRowIndex(newValidVertex->y),(\*mapHelper).getColIndex(graph.getNearestVertex(newValidVertex)->x), (\*mapHelper).getRowIndex(graph.getNearestVertex(newValidVertex)->y)) ;

if(Nonvalid){

std::cout << "Vertex not valid: (" <<newValidVertex->x << "'"<< newValidVertex->y << ")" << std::endl;

continue;}

//no collision now add it to graph

graph.addVertex(newValidVertex);

graph.addEdge(graph.getNearestVertex(newValidVertex),newValidVertex);

std::cout << "Vertex created in: (" << newValidVertex->x << "," <<newValidVertex->y << ")"<< std::endl ;

}

graph.addVertex(targetVertex);

graph.addEdge(newValidVertex,targetVertex);

std::cout << "Tree constructed " << std::endl;

}

Bresenham's line algorithm (draw a line and check for occupied points):

bool MyRRTSolver::bhm\_line(int x1,int y1,int x2,int y2)// y rowIndex x Colindex

{

std::cout << "1 : " << x1 << "," << y1 << std::endl;

std::cout << "2 : " << x2 << "," << y2 << std::endl;

int x,y,dx,dy,dx1,dy1,px,py,xe,ye,i;

bool pixel\_collision = false ;

dx=x2-x1;

dy=y2-y1;

dx1=fabs(dx);

dy1=fabs(dy);

px=2\*dy1-dx1;

py=2\*dx1-dy1;

pixel\_collision = (\*mapHelper).isOccupied( y1 , x1) ; //check collision

if(pixel\_collision){return pixel\_collision;}

pixel\_collision = (\*mapHelper).isOccupied( y2 , x2) ; //check collision

if(pixel\_collision){return pixel\_collision;}

if(dy1<=dx1) { //x orianted

if(dx>=0){//start 1 or 2

x=x1;

y=y1;

xe=x2;

}

else{

x=x2;

y=y2;

xe=x1;

}

for(i=0;x<xe;i++){ //now increasse x and y

x=x+1;

if(px<0){

px=px+2\*dy1;

}

else {

if((dx<0 && dy<0) || (dx>0 && dy>0)) {

y=y+1;

}

else {

y=y-1;

}

px=px+2\*(dy1-dx1);

}

pixel\_collision = (\*mapHelper).isOccupied(y , x) ; //check collision

if(pixel\_collision){return pixel\_collision;}

}

}

else { //y orianted

if(dy>=0) {//start 1 or 2

x=x1;

y=y1;

ye=y2;

}

else {

x=x2;

y=y2;

ye=y1;

}

for(i=0;y<ye;i++){

y=y+1;

if(py<=0){

py=py+2\*dx1;

}

else{

if((dx<0 && dy<0) || (dx>0 && dy>0)) {

x=x+1;

}

else {

x=x-1;

}

py=py+2\*(dx1-dy1);

}

pixel\_collision = (\*mapHelper).isOccupied(y , x) ; //check collision

if(pixel\_collision){return pixel\_collision;}

}

}

return pixel\_collision;

}

Formun Üstü

Formun Altı

**Path Construction**

To construct path first I started with the TargetVertex and iterate through its source Vertex. After finding sourceVertex = InitialVertex path is created.   
Matchvertex returns edge itarator whose target is TargetVertex.   
constructPath() :

struct MatchVertex

{

MatchVertex(const MyVertex\* v) : v\_(v) {}

bool operator()(const MyEdge\* obj) const

{

return obj -> target == v\_;

}

private:

const MyVertex\* v\_;

};

void MyRRTSolver::constructPath()

{

MyVertex\* target = targetVertex ;

while(target != initialVertex){

//we only have one source node for each target node

std::vector<MyEdge\*>::iterator it = std::find\_if(graph.edges.begin(), graph.edges.end(), MatchVertex(target ) );

path.push\_back(target) ;

target = it[0] -> source ;

}

path.push\_back(target) ;

// print the calculated path for debugging purpose

std::cout << "----path----- " << std::endl;

for (int i = 0; i < path.size(); i++)

{

std::cout << "(" << path[i]->x << ", " << path[i]->y << ")" << std::endl;

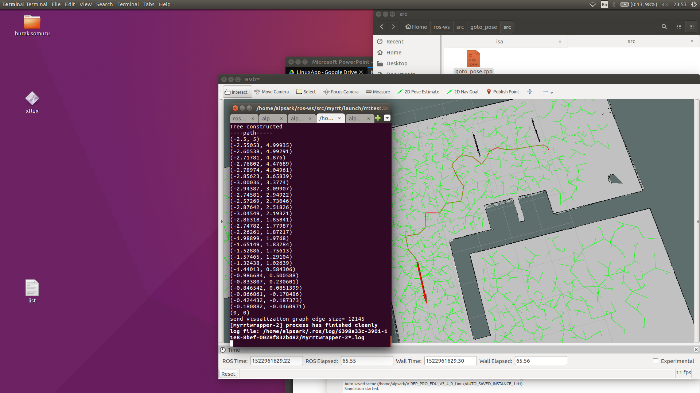
}

}

Formun Üstü

Formun Altı

**Constructed Graph**



Formun Üstü

Formun Altı

**Goto\_Pose video**

[**https://youtu.be/4cYMZMgFjhc**](https://youtu.be/4cYMZMgFjhc)

Formun Üstü

Formun Altı

**Move base as global planner**

I used this guide <http://wiki.ros.org/navigation/Tutorials/Writing%20A%20Global%20Path%20Planner%20As%20Plugin%20in%20ROS> to construct globalplanner.

So i created global\_planner.h , global\_planner.cpp in nav\_stack\_tuning/src and global\_planner\_plugin.xml in nav\_stack\_tuning locations.

I also modified the Cmake, package xml, move\_base in the assignment 7:   
CmakeList.txt

[find\_package](http://www.cmake.org/cmake/help/cmake2.6docs.html#command:find_package)(catkin REQUIRED COMPONENTS

amcl

move\_base

roscpp

tf

std\_msgs

tf\_conversions

visualization\_msgs

)

[add\_library](http://www.cmake.org/cmake/help/cmake2.6docs.html#command:add_library)(global\_planner\_lib src/global\_planner.cpp src/graph/MyVertex.cpp src/graph/MyEdge.cpp src/graph/MyGraph.cpp src/MyRRTSolver.cpp src/MapHelper.cpp)

package.xml:

<buildtool\_depend>catkin</buildtool\_depend>

<build\_depend>amcl</build\_depend>

<build\_depend>move\_base</build\_depend>

<build\_depend>roscpp</build\_depend>

<build\_depend>tf</build\_depend>

<build\_export\_depend>amcl</build\_export\_depend>

<build\_export\_depend>move\_base</build\_export\_depend>

<build\_export\_depend>roscpp</build\_export\_depend>

<build\_export\_depend>tf</build\_export\_depend>

<exec\_depend>amcl</exec\_depend>

<exec\_depend>move\_base</exec\_depend>

<exec\_depend>roscpp</exec\_depend>

<exec\_depend>tf</exec\_depend>

<build\_depend>nav\_core</build\_depend>

<exec\_depend>nav\_core</exec\_depend>

<build\_depend>"std\_msgs"</build\_depend>

<build\_depend>"sensor\_msgs"</build\_depend>

<build\_depend>"tf\_conversions"</build\_depend>

<build\_depend>"visualization\_msgs"</build\_depend>

<exec\_depend>"std\_msgs"</exec\_depend>

<exec\_depend>"sensor\_msgs"</exec\_depend>

<exec\_depend>"tf\_conversions"</exec\_depend>

<exec\_depend>"visualization\_msgs"</exec\_depend>

<!-- The export tag contains other, unspecified, tags -->

<export>

<!-- Other tools can request additional information be placed here -->

<nav\_core plugin="${prefix}/global\_planner\_plugin.xml" />

</export>

move\_base.launch :  
Deleted global\_planner\_params.yaml and global\_costmap\_params.yaml lines.   
Added <param name=“base\_global\_planner” value=“global\_planner/GlobalPlanner”/> line

global\_planner.h and global\_planner\_plugin.xml are the same as the link .

I added the code in MyRRTWrapper.cpp to global\_planner.cpp.   
I also copied every c and cpp file to nav\_stack\_tuning directory except goto\_pose and MyRRTWrapper.cpp,

I didnw used first two vertexes since they are always too close, I also navigated to two vertex after the correct vertex. That rsut in smoother path and better results . global\_planner.cpp:   
Poses constructed with RRT at the start . Code activates when 2D nav goal is entered. Then in each time robot voles to next vertex. After it reach the vertex(tolerances are same as base\_local\_planner\_params.yaml) it jumps to next vertex.

#include <pluginlib/class\_list\_macros.h>

#include "global\_planner.h"

#include "math.h"

#include <ros/ros.h>

#include <visualization\_msgs/MarkerArray.h>

#include <visualization\_msgs/Marker.h>

#include <geometry\_msgs/Point.h>

#include <geometry\_msgs/Pose.h>

#include <nav\_msgs/OccupancyGrid.h>

#include <nav\_msgs/GetMap.h>

#include <std\_msgs/Bool.h>

#include <tf/transform\_listener.h>

#include <vector>

#include "MapHelper.h"

#include "MyRRTSolver.h"

#include "graph/MyVertex.h"

#include "graph/MyGraph.h"

#include "graph/MyEdge.h"

//register this planner as a BaseGlobalPlanner plugin

PLUGINLIB\_EXPORT\_CLASS(global\_planner::GlobalPlanner, nav\_core::BaseGlobalPlanner)

//using namespace std;

bool gotoPoseCompleted = true;

static int counter = 0; // required to draw with different indexes

bool isTesting;

bool path\_created = false;

double initialx;

double initialy;

double targetx;

double targety;

std::vector<geometry\_msgs::Pose> poses ;

visualization\_msgs::Marker createMarkerFromEdge(MyEdge\* edge, int counter, float r, float g, float b, const char\* \_ns, float lineWidth)

{

visualization\_msgs::Marker marker;

marker.header.frame\_id = "odom";

marker.header.stamp = ros::Time::now();

// Set the namespace and id for this marker. This serves to create a unique ID

// Any marker sent with the same namespace and id will overwrite the old one

marker.ns = \_ns;

marker.id = counter;

// Set the marker type. Initially this is CUBE, and cycles between that and SPHERE, ARROW, and CYLINDER

marker.type = visualization\_msgs::Marker::LINE\_STRIP;

// Set the marker action. Options are ADD and DELETE

marker.action = visualization\_msgs::Marker::ADD;

// Set the pose of the marker. This is a full 6DOF pose relative to the frame/time specified in the header

double x = edge->source->x;

double y = edge->source->y;

geometry\_msgs::Point p1;

p1.x = x;

p1.y = y;

marker.points.push\_back(p1);

x = edge->target->x;

y = edge->target->y;

geometry\_msgs::Point p2;

p2.x = x;

p2.y = y;

marker.points.push\_back(p2);

marker.pose.position.x = 0; //edge->source->x;

marker.pose.position.y = 0; //edge->source->y;

marker.pose.position.z = 0;

marker.pose.orientation.x = 0.0;

marker.pose.orientation.y = 0.0;

marker.pose.orientation.z = 0.0;

marker.pose.orientation.w = 1.0;

// Set the scale of the marker -- 1x1x1 here means 1m on a side

marker.scale.x = lineWidth;

marker.scale.y = 0.005;

marker.scale.z = 0.005;

// Set the color -- be sure to set alpha to something non-zero!

marker.color.r = r;

marker.color.g = g;

marker.color.b = b;

marker.color.a = 1.0;

// markers are visible only for 10 minutes

marker.lifetime = ros::Duration(600.0);

return marker;

}

void sendVisualizationMessage(ros::Publisher& visualPublisher, MyRRTSolver& solver)

{

std::cout << "send visualization graph edge size= " << solver.graph.edges.size() << std::endl;

counter = 0;

visualization\_msgs::MarkerArray graphMarkerArray;

for (int i=0; i < solver.graph.edges.size(); i++)

{

visualization\_msgs::Marker marker = createMarkerFromEdge(solver.graph.edges[i], counter, 0.0f, 1.0f, 0.0f, "rrt", 0.01);

graphMarkerArray.markers.push\_back(marker);

counter++;

}

visualPublisher.publish(graphMarkerArray);

visualization\_msgs::MarkerArray pathMarkerArray;

// draw path

for (int i = 0; i < solver.path.size()-1; i++)

{

MyEdge tempEdge(solver.path[i], solver.path[i+1]);

visualization\_msgs::Marker marker = createMarkerFromEdge(&tempEdge, counter, 1.0f, 0.0f, 0.0f, "path", 0.02);

pathMarkerArray.markers.push\_back(marker);

counter++;

}

visualPublisher.publish(pathMarkerArray);

}

//Default Constructor

namespace global\_planner {

GlobalPlanner::GlobalPlanner (){

}

GlobalPlanner::GlobalPlanner(std::string name, costmap\_2d::Costmap2DROS\* costmap\_ros){

initialize(name, costmap\_ros);

}

void GlobalPlanner::initialize(std::string name, costmap\_2d::Costmap2DROS\* costmap\_ros){

}

bool GlobalPlanner::makePlan(const geometry\_msgs::PoseStamped& start, const geometry\_msgs::PoseStamped& goal, std::vector<geometry\_msgs::PoseStamped>& plan ){

plan.push\_back(start);

//BEGIN: ROS

//ros::init(argc,argv,"global\_planner");

ros::NodeHandle nh;

ros::Rate rate(10);

if(!path\_created){

initialx = start.pose.position.x;

initialy = start.pose.position.y;

targetx = goal.pose.position.x;

targety = goal.pose.position.y;

//frame\_id = "odom";

//useInitialPosition = false;

//isTesting = false;

std::cout << "initialx=" << initialx << " initialy=" << initialy << std::endl;

std::cout << "targetx=" << targetx << " targety=" << targety << std::endl;

std::cout << "frame\_id= " << "odom" << std::endl;

std::cout << "use\_initial " << "false" << std::endl;

}

// create graph publisher

ros::Publisher graphPublisher = nh.advertise<visualization\_msgs::MarkerArray>("/mygraph", 10);

// create service client to get map from the map\_server package

ros::ServiceClient mapClient = nh.serviceClient<nav\_msgs::GetMap>("static\_map");

nav\_msgs::GetMap srvMap;

nav\_msgs::OccupancyGrid myMap;

bool mapLoaded = false;

while (!mapLoaded) {

// get the map

if (mapClient.call(srvMap))

{

std::cout << "map is retrieved successfully" << std::endl;

const nav\_msgs::OccupancyGrid& map(srvMap.response.map);

myMap = map;

mapLoaded = true;

} else {

ROS\_ERROR("map retrieve error");

}

usleep(500000);

}

// create map helper class

MapHelper mapHelper(myMap);

// get the current pose if we do not provide initial position

tf::TransformListener transformListener;

tf::StampedTransform robotTransform;

// BEGIN: RRT solver

// implement your own RRT algorithm by using class template MyRRTSolver

// you are free to modify in any way you wanted to

if(!path\_created){

MyRRTSolver solver;

solver.setMapHelper(&mapHelper);

solver.solve(initialx, initialy, targetx, targety);

//END: RRT Solver

// get pose commands from the RRT solver

poses = solver.getWayPoints();

//BEGIN: RVIZ VISUALIZATION

sendVisualizationMessage(graphPublisher, solver);

//END: RVIZ VISUALIZATION

path\_created = true;

}

geometry\_msgs::PoseStamped new\_goal = goal;

// wait for 3 seconds to be make sure that connection is complete and messages will be sent successfully

usleep(3000000);

// the goto pose loop will run we do not test the rrt algorithm

geometry\_msgs::Pose pose ;

// get pose information from transform tf

transformListener.lookupTransform("odom","base\_link", ros::Time(0),robotTransform);

float robotX=robotTransform.getOrigin().x();

float robotY=robotTransform.getOrigin().y();

float robotYaw=tf::getYaw(robotTransform.getRotation());

// if goto pose completed we can send new goto pose command

if (gotoPoseCompleted && (poses.size() > 0)) {

// remove the first two elements

gotoPoseCompleted = false;

//delete vertexes 2 by two

if(poses.size() > 3){

pose = \*(poses.begin()+2);

poses.erase(poses.begin());

poses.erase(poses.begin());

}

else {

pose = \*(poses.begin()+2);

poses.erase(poses.begin());

}

new\_goal.pose.position.x = pose.position.x ; //pose.position.x ;

new\_goal.pose.position.y = pose.position.y;

new\_goal.pose.orientation.z = pose.orientation.z;

plan.push\_back(new\_goal);

}

std::cout<<"robot pose: x,y,yaw: "<<robotX<<","<<robotY<<","<<robotYaw <<std::endl;

std::cout << "Semi-goal pos: " << new\_goal.pose.position.x << " pose.y=" << new\_goal.pose.position.y << " number of semi-goals left " << poses.size()-2 << std::endl;

if(0.04 < (robotX- new\_goal.pose.position.x) \* (robotX- new\_goal.pose.position.x) + (robotY- new\_goal.pose.position.y) \* (robotY- new\_goal.pose.position.y)) {

if( (robotYaw < new\_goal.pose.orientation.z + 0.3)|| (robotYaw > new\_goal.pose.orientation.z - 0.3) ) {

gotoPoseCompleted = true;

}}

return true;

}

};

Formun Üstü

Formun Altı

**Move base video**

<https://youtu.be/jTliQy8ekGI>