Leading the World with Asia's Best

Term Project report for CS6244

# Particle - Rapidly Exploring Random Tree

**Submitted by** :
Astha Jain (A0123438J)
Harshul Gandhi (A0123448H)

## TABLE OF CONTENT

# 1. INTRODUCTION

Path planning is an important aspect for autonomous mobile robots. It has been a major field of research among enthusiastic researcher. Planning a path for an autonomous robot doesn't just refers to finding a path, but it should be an optimal path and the one that avoid obstacles. There has been many research papers published describing different path planning algorithms like Dijkstra's, A*, D* and RRT (Rapidly exploring random tree).

Out of these algorithms, RRT is one of the most popularly used algorithm for path planning with kinodynamic constraints. As the name suggested, in RRT involves building a rapid tree in the search space by selecting random states (state here refers to a 'node' on the tree). Selected random state is added to the nearest node in the tree built so far, provided there exists a collision free connection between this selected state and state nearest to it in the tree.

There is one drawback in RRT. n case of RRT, each node in the search tree represents one state. But there might be a case where robot does not successfully reach that exact state while traversing the path. There are some uncertainties in the environment that may cause the robot to end up in another state as compared to the one identified by the tree. This can jeopardize the execution of path.

So, it does not provide any concrete way of tackling some of the environmental uncertainties for example coefficient of friction and angle of elevation of a slant terrain. To overcome this drawback, there is another algorithm known as particle RRT (pRRT). pRRT has provision to plan the path taking under consideration the uncertain factors of the environment. In this algorithm, we consider all the possible states where robot might end up. These are called particles. Similar particles (corresponding to one node of the tree) are grouped to form clusters. These clusters then represent one node in the tree formed via pRRT algorithm. We'll look at this algorithm in detail in later sections.

## 1.1. Motivation

We wanted to identify a solution to handle uncertain factors of environment. pRRT provides that solution. But in pRRT, we'll have to consider these uncertainties while creating the tree, i.e., before actual execution of path. This involves a lot of approximation in terms of uncertainty factors which can be a little less precise at times. Our work is motivated by the idea of modifying pRRT in a way that robot takes these uncertainties into consideration on the fly. This would provide better approximation of uncertain factors of the environment as robot will the in the field.

# 2. OBJECTIVE

The objective of our project is to:

- study the existing p-RRT algorithm
- find path for an autonomous robot through an unknown terrain to reach from a specified source to goal by implementing RRT
- implement p-RRT over the path provided by RRT and hence handle the environmental uncertainties
- define a cost vs safety path for the robot so that more efficient path is prefered while execution of algorithm

# 3.ALGORITHM

**ALGORITHM (p-RRT)**

## 3.1. Sampling

### 3.1.1. Sampling Algorithm:

```
//image <- input image of the environment
SAMPLE(p[][2])
1: number <- number of samples
2: n <- 0
3: I <- image
4: while (n <= number)
5:        {x,y} = GENERATE_SAMPLE(I)
6:        if (x.R, x.G, x.B != 0 AND y.R, y.G, y.B != 0) //static obstacle
7:                p[n][2] <- {x,y}
8:                n = n+1

GENERATE_SAMPLE(I)
1: RETURN RANDOM_NUMBER {(x, y) ∈ I}
```

### 3.1.2. Explanation:

For sampling the search space efficiently, we have implemented following steps based on the above algorithm.

- Input for sampling code is an image of the search space where static obstacles are marked as solid black color.

- Also, to specify a sloped structure (similar to a hill), we given solid red color to one side (uphill) and solid blue color to other side (down hill).
- After reading the image in Step 1, we call GENERATE_SAMPLE(image) function to return randomly picked pixel values of this image. This random selection is uniform and is achieved by using “ranlux48”[2] and “uniform_int_distribution”[1] classes.
- For the random pixel value (x,y) generated, we check for the color region in which it lies. If its black that means it overlaps with the obstacle and we ignore this random selection.
- If not, we consider it as a valid sample and it is stored in array ‘p’. (Step 6 & 7)

## 3.2. RRT

### 3.2.1. RRT Algorithm

```
BUILD RRT($x_{init}$)
1: T .init($x_{init}$)
2: while ($x_{goal}$ !∈ T ) do
3: {p, q} ← SELECT EXTENSION(T );
4: EXTEND(T , p, q)
5: Return T

SELECT EXTENSION(T )
1: p ← RANDOM STATE();
2: q ← NEAREST NEIGHBOUR(p, T );
3: Return {p, q}

EXTEND(T , p, q)
1: if (not EDGE_COLLISION(p,q)) then
2: T .add vertex(p);
3: T .add edge(q, p);
```

### 3.2.2. Explanation

- BUILD_RRT function generates a random tree rooted at $x_{init}$. The tree is built until goal node is reached. For each iteration, it calls SELECT_EXTENSION function (Step 3).
- SELECT_EXTENSION randomly picks up a node from sampled points (Step 1). Then it finds the nearest neighbour in the tree to this randomly picked node (Step 2).
- Once the nearest node to the newly picked node has been found, EXTEND function is called which establishes an edge between these two nodes.
- Before establishing this edge, collision check is applied on the edge being created (Step 1). If the edge is collision free, node ‘p’ is added to the tree (Step 2) and edge (q, p) is added to the tree (Step 3).

## 3.3 p-RRT :

While executing the path returned by RRT when we encounter any environmental change, p-RRT

algorithm is executed on the fly.
To generate Particle-Node, as mentioned in [1], multiple simulations are used while generating the tree from RRT. But for our project we have introduced on the fly creation of particles. We believe that the best estimate of the vibrant environmental factors at the next node can be observed only from the current node, when a path consists of nodes and path is followed in the nodal sequence only.

Intuitively, this does not add much additional cost as on the fly re-planning already happens for dynamic obstacles. The particle node generation algorithm is as follows:

```
CALCULATE_PATH
1: path[] <- RRT path
2: p = CURRENT_NODE
2: $\mu_{01}$ = most probable friction at current node
3: while (p != Goal)
        $\mu_{p1}$ = most probable friction at next node
        if ($\mu_{01}$ != $\mu_{p1}$)
                NEXT NODE = RANDOM NEW MEAN STATE()
        $\mu_{01} = \mu_{p1}$
        p = NEXT NODE
4: RETURN path[]

RANDOM NEW MEAN STATE
1: p ⟵CURRENT NODE
2: $\mu_p$[] <- all probable friction values at next node
3: PARTICLE[] = CREATE_PARTICLE(p, prob{$\mu_p$})
4: MEAN STATE = CLUSTER(PARTICLE[])
5: RETURN MEAN STATE

CREATE_PARTICLE(p , prob{$\mu_p$})
1: $\mu$ = friction at next node
2: p($\mu$) = prob{$\mu_p$}
3: for all ($\mu_p$):
        PARTICLE_STATE = CALC_DISTANCE(p , p($\mu$))
        PARTICLE[] <- PARTICLE_STATE
4: RETURN (PARTICLE[])

CALC_DISTANCE(p , p($\mu$))
1: RETURN SAMPLE at EUCLIDEAN DISTANCE FROM p in direction of NEXT NODE

CLUSTER(p'[])
1: for all (p_particle $\in$ p'[] ):
        if (dist(p',cluster) < threshold)
                cluster[] <- p'
        else    NEW CLUSTER ⟵p'
2: NEW MEAN STATE = CALC_MEAN(cluster[])
3: RETURN NEW MEAN STATE

CALC_MEAN(cluster[])
```

```
1: sum_x = 0
2: sum_y = 0
3: for all (particle ∈ cluster[])
        particle_x = X coordinate of particle
        particle_y = Y coordinate of particle
        sum_x = sum_x + ($\mu_{particle}$) * particle_x
        sum_y = sum_y + ($\mu_{particle}$) * patricle_y
4: MEAN = (sum_x, sum_y)
```

**A. Generating Particle Nodes**

Once the robot starts to follow the path returned by RRT, it can be comfortably assumed that it is always aware of its current state. By state, it is clear that it knows about its own position, orientation and environmental variables, that, in our case is friction of the ground.

So, when it moves ahead and estimates variation in environmental parameters at the next node, p-RRT algorithm is called. Under such event, we split the next node into a cluster of probable particles where the robot might land up taking the uncertainty of variable estimation in account.

For our project, the variable environmental factors that we have considered are friction and the slope of hills that could be encountered on the terrain. Friction coefficient values are chosen from the set $F \in \{0.2, 0.4, 0.6, 0.8\}$ and slope is chosen as $\theta \in \{\pi/4 , \pi/3\}$

The result of each Particle Generation step is a particle node. Term is coined because each node consists of a bunch of particles. After the particle node is created, to facilitate further extension, we calculate a MEAN STATE that can represent all the particles of that particle node. There are two ways to generate the MEAN STATE as proposed in [1].

a. Calculate the mean for all particles
b. Discrete Sampling

For our experiments we have chosen MEAN STATE as the mean of all particles.

Let, $P(F_i \in F)$ denote the probability of each discrete friction value. Then the MEAN STATE for all $q_i \in$ **q (particles)** can be represented as

$q_{mean} = 1/F_{sum}(\sum q_i * F_i)$

Moreover, we are generating particles only where uncertainty is high, the estimation could be more precise. This reduces the computational time, as in original p-RRT , particles are calculated for each node of the tree.

B. Clustering [1]

Clustering of closer particles involves grouping of closely distant particles in a particle node to find the MEAN STATE , to minimize the extreme variations caused by reading errors.

Although, we have considered all particles for each node to find the MEAN STATE for experiments.

C. Node and Path creation
Once the MEAN STATE is generated for the particle node, it is checked for collision with obstacles. If it is collision free, it replaces the original NEXT NODE in the path, otherwise most probable node from the samples is chosen. The path from MEAN NODE to the CURRENT NODE is calculated from the existing tree, until a direct path is found recursively.

# 4.EXPERIMENTAL RESULTS

**4.1 Input -**

**1. Image of the test environment**

For the experiments a 2-D image of the terrain is given as input, in which obstacles are colored black, hills are color coded as blue and red. Either of these color could represent uphill or downhill depending upon the CURRENT STATE of the robot
Size of the image: 500 X 500 px

2. Start Node coordinates
3. Goal Node coordinates

**4.2 Parameters -**

- maximum velocity (v) of the robot is considered and maintained throughout the experiments to be 1 m/s
- The acceleration a is maintained to keep the velocity of robot as 1 m/s depending on the frictional force applicable on the it
- The acceleration due to gravity is considered as $g = 10 m/s^2$
- The slope of the hills is taken from the set $\theta \in \{\pi/4 , \pi/3\}$
- The friction is taken from the set $F \in \{0.2, 0.4, 0.6, 0.8\}$
- number of samples = 1000
- number of iterations in RRT = 1000
- number of nodes in tree - RRT = 1000
- scale : 170px = ½ m

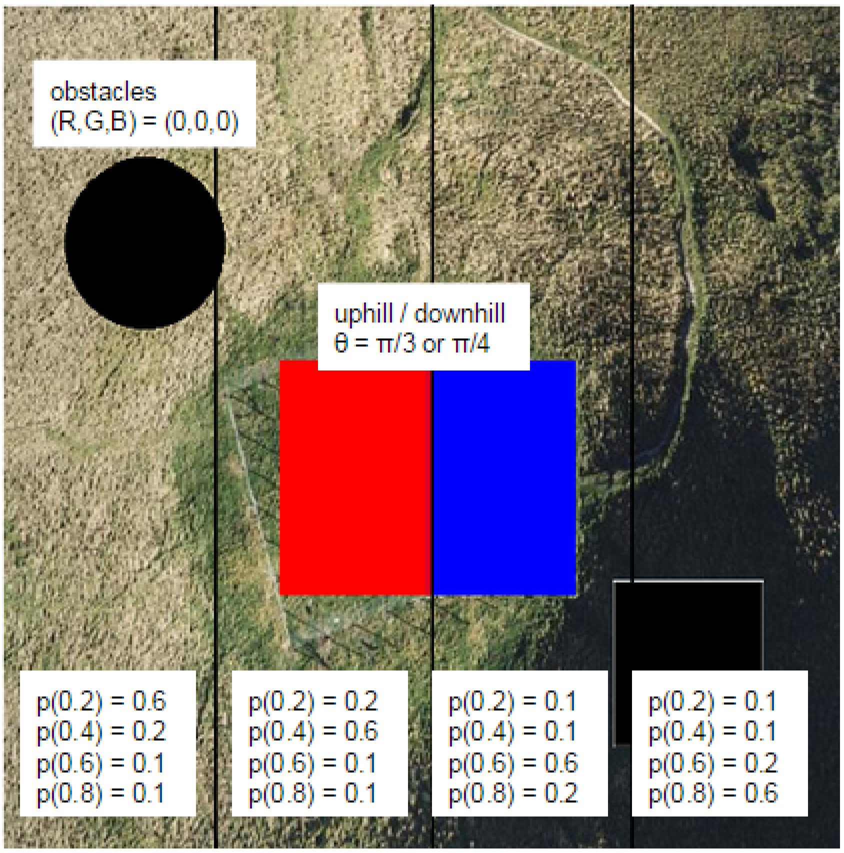
**4.3 Assumptions**

- obstacles are circular or polygonal

- Robot is a point robot

**4.4 Test environment**

- The experiments are carried out in a 2-D environment
- The environment is divided into 4 zones each representing a different probability distribution from F



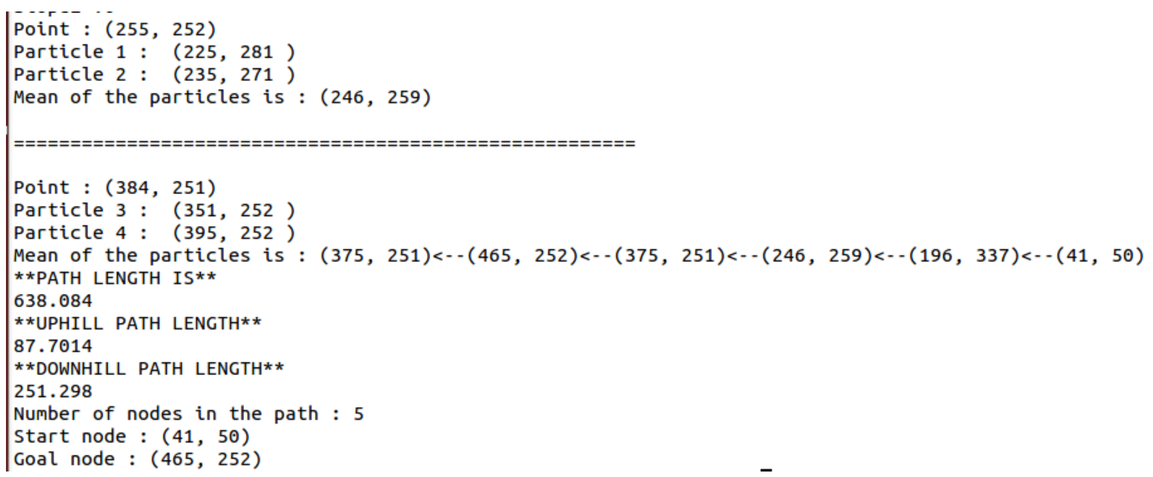
Sample Test Environment

**4.5 Results**

**<u>4.5.1 Test Case 1</u>**

P(0.4) = 0.7
P(0.5) = 0.2
P(0.6) = 0.1
Uphill/ Downhill theta = 0 degrees

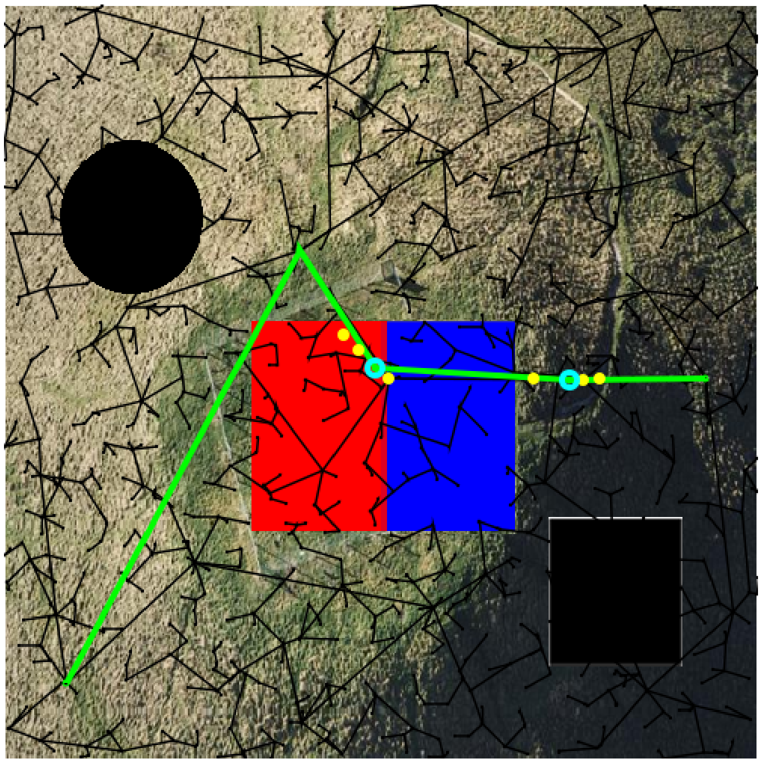
Result : Path Length : 638.084



```
Point : (255, 252)
Particle 1 :  (225, 281 )
Particle 2 :  (235, 271 )
Mean of the particles is : (246, 259)

=======================================================

Point : (384, 251)
Particle 3 :  (351, 252 )
Particle 4 :  (395, 252 )
Mean of the particles is : (375, 251)<--(465, 252)<--(375, 251)<--(246, 259)<--(196, 337)<--(41, 50)
**PATH LENGTH IS**
638.084
**UPHILL PATH LENGTH**
87.7014
**DOWNHILL PATH LENGTH**
251.298
Number of nodes in the path : 5
Start node : (41, 50)
Goal node : (465, 252)
```

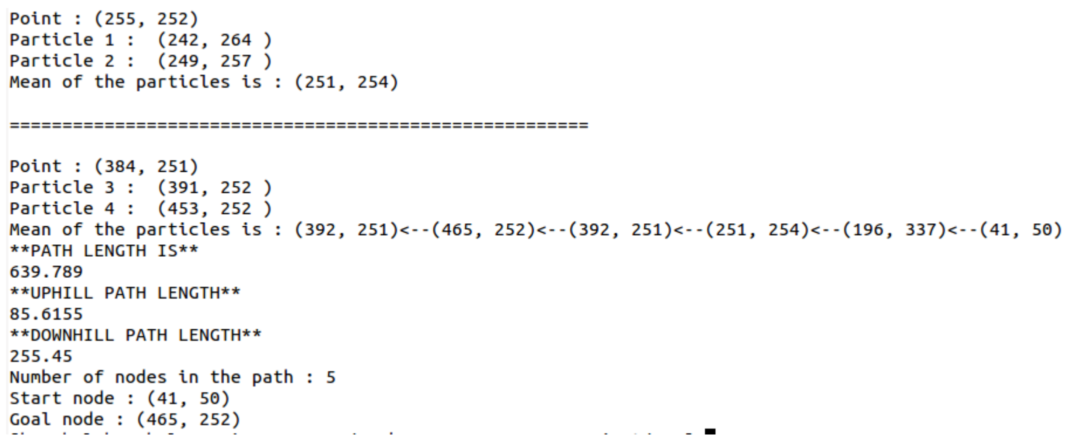


**4.5.2 Test Case 2**

p(0.4) = 0.7
p(0.5) = 0.2
p(0.6) = 0.1

Uphill/downhill theta = 45 degrees

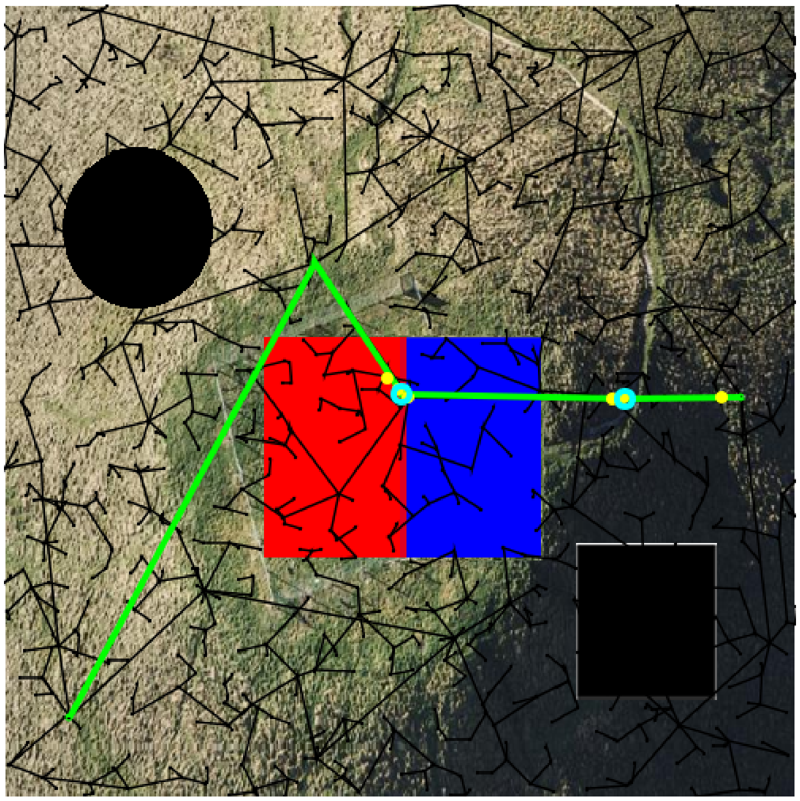
**Result: Path length : 639.789**



```
Point : (255, 252)
Particle 1 :  (242, 264 )
Particle 2 :  (249, 257 )
Mean of the particles is : (251, 254)

=======================================================

Point : (384, 251)
Particle 3 :  (391, 252 )
Particle 4 :  (453, 252 )
Mean of the particles is : (392, 251)<--(465, 252)<--(392, 251)<--(251, 254)<--(196, 337)<--(41, 50)
**PATH LENGTH IS**
639.789
**UPHILL PATH LENGTH**
85.6155
**DOWNHILL PATH LENGTH**
255.45
Number of nodes in the path : 5
Start node : (41, 50)
Goal node : (465, 252)
```



## 4.5.3 Test Case 3

p(0.4) = 0.3

p(0.5) = 0.1

p(0.6) = 0.6

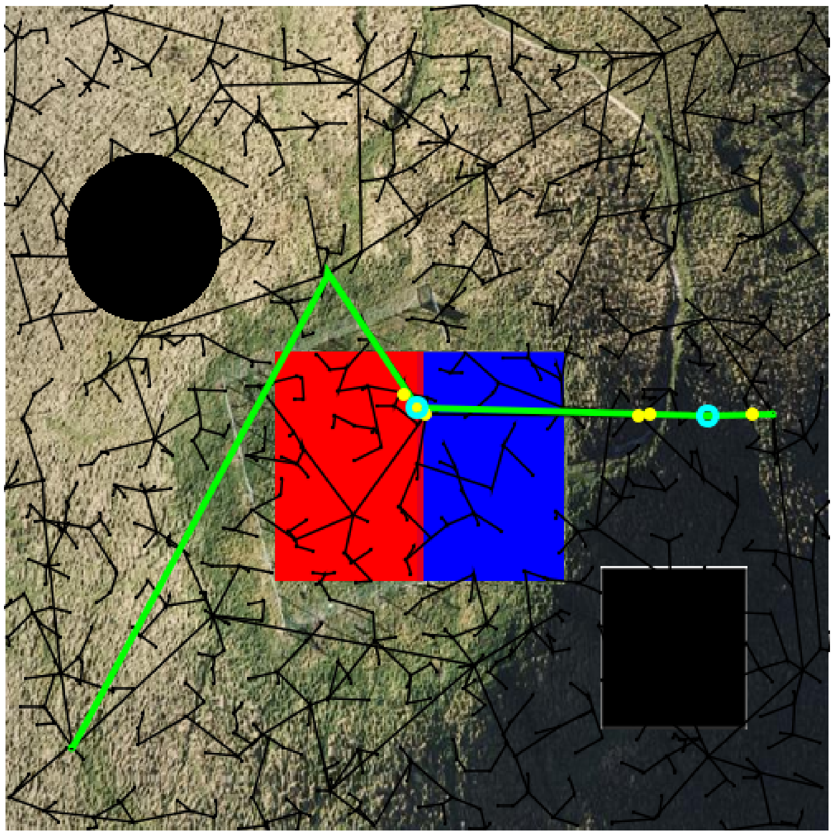
Uphill/downhill theta = 45 degrees

**Result: Path length : 689.615**

```
Point : (255, 252)
Particle 1 :  (242, 264 )
Particle 2 :  (249, 257 )
Mean of the particles is : (250, 256)

=======================================================

Point : (384, 251)
Particle 3 :  (391, 252 )
Particle 4 :  (453, 252 )
Mean of the particles is : (426, 251)<--(465, 252)<--(426, 251)<--(250, 256)<--(196, 337)<--(41, 50)
**PATH LENGTH IS**
638.615
**UPHILL PATH LENGTH**
86.0335
**DOWNHILL PATH LENGTH**
254.118
Number of nodes in the path : 5
Start node : (41, 50)
Goal node : (465, 252)
```



## 5.CONCLUSION

- The observed paths after using pRRT are more safe as compared to RRT path, as the risk of climbing steep hills or overshooting the estimated distance after downhill are minimized. These results are shown as MEAN STATE after the creation of particles.
- The MEAN STATE represents the next NODE so the acceleration values can be set so as to reach this state without breaking the device.

## 6. APPLICATIONS

- The system can be used to provide information about the terrain in rainy and hilly areas to autonomous vehicles
- Similar system can be combined with applications like Google Maps to provide more information about the environment
- p-RRT can be applied to home robots for tasks like mowing the lawn, walking on the floor with variable friction etc.

## 7. FUTURE WORK

- The path finding algorithm by using RRT can be optimized by using hRRT*
- Vision modules to retrieve better information about the terrain dimensions will be helpful

## 8. REFERENCES

[1] Melchior, N. A., & Simmons, R. (2007). Particle RRT for path planning with uncertainty. In Proc. IEEE International Conference on Robotics and Automation.
[2]http://www.cplusplus.com/reference/random/
[2] http://www.cplusplus.com/reference/random/ranlux48_base/