

Path Planning for Autonomous Information Collecting Vehicles

Jun-young Kwak
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, U.S.A.
Email: junyoung.kwak@cs.cmu.edu

Paul Scerri
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA, U.S.A.
Email: pscerri@cs.cmu.edu

Abstract—In many environments where autonomous air or ground vehicles are used to collect information, there will be a known prioritization of areas of the environment where most valuable information will be found. Over time, priorities may change with areas losing value or suddenly becoming important. In this paper, we present an approach to planning paths for vehicles collecting information in such environments, such that they maximize the overall system information gain over time. A key feature of this path planning problem is that there is not a single or small set of goal points to which the vehicles should try to reach, instead information is collected over the entire path without a particular goal in mind. We present a planning approach, which rapidly expands a search tree, inspired by an RRT planner by choosing promising nodes to expand and expanding them randomly. Genetic algorithms are used to learn sets of configuration parameters for the planner, i.e., how to expand which nodes. Results show that the learned planner gets more substantially information than pre-defined paths in a variety of domains.

Keywords: Path planning under uncertainty, Rapidly-exploring Random Tree, Genetic algorithm, Randomized methodology.

I. INTRODUCTION

In a variety of important domains, autonomous, unmanned systems are either being considered or already being used as mobile sensor platforms, capable of collecting large amounts of information about an environment. In a large environment, the platforms need to move around, using their sensors to sense their immediate surroundings. Typically, information from all parts of the environment is not of equal value, at all times. For example, in a disaster response, EO sensing of an area that was very recently searched with IR, is of lower value than sensing of an area that has not been searched at all. Moreover, information requirements and areas where most valuable information can be sensed changes dynamically and unpredictably. A key task for an autonomous vehicle in such an environment is to plan paths that allow it to utilize its sensors to maximize the value of information it collects over time.

While path planning has been a very active research area for a long time [1] [3] [4], most planners are not well suited to this particular problem. Two specific features of this particular problem make it problematic. First, there is no “goal” location for the planner to head toward, the vehicle might go in any direction and end up in any location. Second, value is

accumulated along the path of the vehicle, meaning that paths that back-track, cross themselves or spiral around an area might be optimal in some situations. Typically, path planners are designed to avoid such paths and hence are not appropriate, e.g., RRTs [6] and Annotated and hierarchical graph-based path planning [5]. Moreover, pre-defined, “optimal” search patterns are not appropriate since information needs change dynamically [8].

The algorithm proposed in this paper adapts the concept of a Rapidly-exploring Random Tree (RRT) planner to the constraints of this problem. Specifically, an initial node is randomly expanded in a range of directions. The new nodes are inserted into a priority queue based on the estimated value of additional expansion of that node. The top node in the priority queue is removed and again randomly expanded, with new nodes again being inserted into the priority queue. This process is repeated until a time limit is reached, at which point the nodes representing the best path is returned. The key intuition behind the algorithm is that most promising nodes are always being expanded and that expansion can occur very quickly, allowing many possibilities to be evaluated. Clearly, the keys to the algorithm are the prioritization of the expanded nodes for insertion in the queue and the effectiveness of the expansion of the promising nodes.

Hand-tuning of these important functions might be possible for a specific scenario and vehicle, however, in general different types of expansion functions are desirable. We applied a genetic algorithm (GA) to the task of learning parameters for these functions. Given a characterization of the type of environment for which plans are required and a specification of the characteristics of the vehicle traversing the paths, the GA was able to fairly quickly find a parameterization that would lead to good plans in many specific instances. However, the GA was prone to falling into local minima which necessitated the application of several techniques for improving the GA search.

Within a given environment, different types of expansion might be appropriate in different locations. For example, in an environment where useful information is only found in one particular area, expanding the path in long, straight pieces is appropriate until the vehicle is in the area where most information can be collected is most effective and then expanding in

short pieces, with potentially high turn rates when in the area where information can be collected becomes most effective. To allow for this, the planner is given multiple expansion and prioritization functions which it uses alternatively. With this technique the planner is able to rapidly find plans in a wide range of situations.

The proposed algorithm, the adapted RRT planning with GA learning, showed the better performance than pre-defined path planning and the RRT with a random configuration over the various scenarios. The experimental results were investigated using specific performance criteria including the combined reward and planning time.

II. PROBLEM STATEMENT

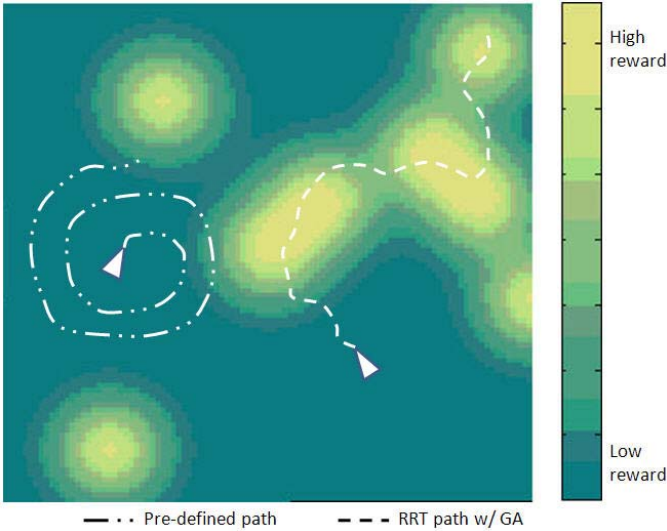


Figure 1. Comparison over a mixed Gaussian value map: Pre-defined path (on left) vs RRT path with a configuration set obtained from GA learning (on right)

In this section, we formally describe the problem addressed in this paper. The expected value of information available at a location x, y at time t , given a history of previous sensor readings H is captured by the function $I(x, y, t, H) \rightarrow \mathcal{R}$. While I varies dramatically from environment to environment and even instance to instance, it will typically exhibit some basic properties. First, typically $|I(x, y, t, H) - I(x, y, t + 1, H)|$ will be small, i.e., most of the time the value of information at a particular location will not change dramatically in a short period of time. For example, in a military scenario, certain places will be inherently more important to know about, but as a battle progresses the key locations to know information will change. Second, typically, if a sensor reading was taken at a location at time t , there will be little to no value to taking a sensor reading at that location at time $t + 1$. The exception to this is when a sensor reading turns out to be ambiguous. Third, typically, $|I(x, y, t, H) - I(x \pm \epsilon, y \pm \epsilon, t, H)|$ will be small, i.e., nearby locations will have similar value. For example, all points along a road may be of approximately the same importance, with importance falling off away from the

road. Finally, for domains of interest, there will be significant variance in the value of I over the environment, i.e., some locations will be substantially more or less important.

Figure 1 shows an example of the type of domain of interest in this paper. Notice that large areas are of significantly higher or lower value than other areas. The paths of the two vehicles indicate how a well designed path (on right) can collect significantly more valuable information than a simple pattern (on left).

Each vehicle, $v_i \in V$, must follow a path P_i in the environment taking sensor readings to collect information. Those paths should maximize the information gain of the vehicle. Information gain, IG , is computed as:

$$IG_i(P_i) = \int_{t=0}^{\infty} I(P_i.x, P_i.y, P_i.t, H) dt$$

When multiple vehicles are in environment, the total information gain due to all paths, P , is simply $IG(P) = \sum_{P_i \in P} IG_i(P_i)$. Notice that H is the union of the paths flown by all vehicles up to the current time. The optimization problem is thus to find P^* such that:

$$P^* = \max_P IG(P)$$

It is assumed that vehicles can communicate to coordinate their paths, but that bandwidth is limited. I is assumed to be thrown and stored as a “costmap”.

III. RAPIDLY-EXPLORING RANDOM TREE PLANNER

The original Rapidly-exploring Random Tree (RRT) algorithm was introduced by Lavelle and Kuffner [6]. RRT is a widely-used algorithm for motion planning in high-dimensional spaces with kinodynamic constraints. In this work, we use an alternative RRT that has a similar concept to the original RRT algorithm, but differs in an important way. The basic RRT algorithm is composed of two main procedures: *Select* and *Extend*. The *Select* function draws its sample (x_{sample}) from a uniform distribution spread over the problem space, and finds the nearest neighbor (x_{near}) from the selected sample. The *Extend* function takes this selected pair of nodes (x_{sample}, x_{near}), and generates an action from the node towards the sample (u_{new}) and the resultant state (x_{new}). If u_{new} is executable, x_{new} is added in the existing tree [6].

In the proposed RRT, the algorithm does not have an explicit *Select* step. Importantly, by excluding this step, a nearest neighbor calculation is not required, thus removing the most computationally expensive part of an RRT.

Instead, at each step, the node having the maximum combined reward is selected to extend without picking a random point. With given the cost map, vehicle properties and configurations specifying several aspects including the capability of movement, a node is randomly (but reasonably) extended considering given parameters. More specifically, configuration properties used in RRT planning contain maximum number of iterations, number of branches per expansion, maximum

Algorithm 1 RRTPlan(): Generate a RRT plan with given init configuration, costmap, environment, vehicle information, and etc...

Require: $startPoint, initState, configuration, costmap, startTime$

Ensure: $path$

- 1: $config \leftarrow setConfiguration(configuration)$
- 2: $env \leftarrow setEnvironment()$
- 3: $vehicle \leftarrow setVehicle()$
- 4: $queue \leftarrow initQueue()$
- 5: $startNode \leftarrow newNode(startPoint, initState)$
- 6: $putNodeToQueue(startNode, queue)$
- 7: **for** $i = 0$ to $config.noExpansions$ **do**
- 8: **for** $i = 0$ to $config.branchesPerExpansion$ **do**
- 9: $node \leftarrow getBestNode(queue)$
- 10: $expNode \leftarrow expandRRTTree(node, cms, config, env, vehicle)$
- 11: $putNodeToQueue(expNode, queue)$
- 12: **end for**
- 13: **end for**
- 14: $path \leftarrow initPath()$
- 15: $node \leftarrow getBestNode(queue)$
- 16: $path.score \leftarrow -node.totCost$
- 17: **while** $node \neq NULL$ **do**
- 18: $path \leftarrow appendWaypoint(node, timeStep)$
- 19: $node \leftarrow node.prev$
- 20: **end while**
- 21: **return** $path$

angle between two branches horizontally and vertically, and maximum and minimum distance that branch can be as ratio of map width.

Each iteration of the algorithm, as specified in Algorithm 1, begins with the vehicle in a particular state. At the first step, it initializes a queue, environment and configurations (lines 1-4). The initial tree (i.e. queue) only consists of the initial state of the vehicle. A new state is expanded stochastically from the best node in the tree, but biased by the parameters in the configuration (lines 7-13). This continues for a given number of iteration, after which the best node is retrieved, and defines the path to be followed (lines 14-20). The configuration used in this algorithm is obtained by learning technique based on genetic algorithm (GA). We will discuss the detailed procedure to get this parameter in the next section.

Each iteration of the algorithm calls $expandRRTTree$ shown in Algorithm 2 to actually expand RRT tree from a given node. In main routine, the node is selected based on the quality value, the total cost of reaching the selected node from the root of the tree. Once the best node is picked, the algorithm expands randomly from the selected one. For each extension added to the tree, several states are computed via the parameters in the configuration. The resulting states are added after a verifying process (lines 10-11), which checks feasibility of them.

In Algorithm 2, the node is expanded by the parameters

Algorithm 2 $expandRRTTree()$: Expand RRT Tree from a given node with costmaps

Require: $start, cms, config, env, vehicle$

Ensure: $expand$

- 1: $rand \leftarrow getRandomValue(0, 1)$ // Get a random value between 0 and 1
- 2: $\theta \leftarrow start.theta + (config.maxTheta \times rand)$
- 3: $\psi \leftarrow start.psi + (config.maxPsi \times rand)$
- 4: $(\theta, \psi) \leftarrow setAngle(0, 360)$ // Set angles between 0 and 360.0
- 5: $randInt \leftarrow getRandomIntValue(config.rrtBranch)$
- 6: **if** $start.prev = NULL$ **then**
- 7: $expand \leftarrow start + randInt - \frac{config.rrtBranch}{2}$
- 8: **else**
- 9: $dist \leftarrow env.mapSize \times (rand \times (config.maxLength + config.minLength) + config.minLength)$
- 10: $(expand.x, expand.y, expand.z) \leftarrow start + dist \times (\cos(degreeToRad(\theta)), \sin(degreeToRad(\theta)), \cos(degreeToRad(\psi)))$
- 11: **end if**
- 12: $start \leftarrow incExpansions(1)$
- 13: $expand.prev \leftarrow start$
- 14: $expand.depth \leftarrow start.depth + 1$
- 15: $expand.totCost \leftarrow calcTotalCost(expand, start, cms, config)$
- 16: $verifyNode(expand)$
- 17: **return** $expand$

in the configuration such as θ, ψ , minimum and maximum length. First, we set the parameter values randomly from a range of permissible values (lines 1-5). Next the planner actually expands from the starting node using the selected parameter values (lines 6-11). After getting the expansion node, lines 12-14 set the hierarchy between two nodes, and change the related values including depth and total cost. Total cost for the newly expanded node is calculated by the following equations:

$$\begin{aligned}
 Cost_{total} &= Cost_{near} + Cost_{costmaps} \\
 &\quad + Penalty_{direction_change} + Penalty_{height_change} \\
 &\quad + Penalty_{distance} \\
 Penalty_{direction_change} &= \\
 &\quad smoothFactor^{(scaleFactor \times \Delta Vel)} - penalizeFactor \\
 Penalty_{height_change} &= heightFactor \times \Delta Height \\
 Penalty_{distance} &= e^{\frac{(\Delta Distance - prefPathLength)}{scaleFactor}}
 \end{aligned}$$

$Cost_{near}$ is a penalty for being near any previous nodes on this path. The concept behind this cost is that if the distance between the current node and previous ones is close to that of the path distance then the path cannot be turning on itself too much. $Cost_{costmaps}$ is for adding all cost of any terrain covered. Also, we calculate a bonus or penalty for straight and jagged movement and an inherent cost for going either up or down to penalize excess changes in direction,

$Penalty_{direction_change}$ and $Penalty_{height_change}$. For the last step, we also consider the expanded distance metric to avoid nodes having too long distance and promote a length otherwise, $Penalty_{distance}$. The numbers used in these equations were experimentally determined.

IV. LEARNING APPROACH

Algorithm 3 GALearning(): GA Learning to get a configuration set

Require: $popSize, maxGens, sizeOfQueue, threshold$

Ensure: $queue$

```

1:  $env \leftarrow generateEnvironmentConf()$ 
2:  $vehicle \leftarrow generateVehicleConf()$ 
3:  $queue \leftarrow generateInitQueue(sizeOfQueue)$ 
4:  $initPlanner(env, vehicle)$ 
5:  $cms \leftarrow initCostmap()$ 
6:  $pop \leftarrow generatePopulation(popSize)$ 
7:  $evaluateFitness(pop, cms)$ 
8: if  $preEvaluate(pop) < fitnessThreshold$  then
9:   Stop this process and generate population, again.
10: end if
11: while  $diff > threshold$  and  $curIteration < maxGens$ 
do
12:    $cms \leftarrow addMoreCostmaps()$  // if we need
13:    $candidate \leftarrow selectTwoIndsByRWS(pop)$ 
14:    $ops \leftarrow createOffsprings(candidate, mode)$ 
15:    $diff \leftarrow updatePopulation(pop, ops)$  // Replace the
      worst individual with a generated offspring
16:    $updateQueue(queue)$ 
17:    $curIteration \leftarrow curIteration + 1$ 
18: end while
19: return  $queue$ 

```

The RRT algorithm presented in the previous section relies on the configuration parameters for best performance. In this Section, we introduce a learning algorithm based on a genetic algorithm (GA) that is adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic. [9]. The overall idea is specified in Algorithm 3. First, the algorithm initializes the environment, costmap and vehicle properties and generates initial population having random configuration values with a given population size (lines 1-6). In each generation, the fitness of every individual in the population is evaluated by the simulation in the path planner, and multiple individuals are selected from the current population based on their fitness by *Roulette Wheel Selection* (RWS) method [10]. The selected individuals are modified by crossover and random mutation - to form a new population. Specifically, for implementing *crossover*, we generate a new configuration based on parents' configurations with calculated weight factors, which are obtained by their fitness. The new individual is then used to replace the worst ranked individual in population. In this algorithm, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for

the population. After finishing iterations, the obtained best solutions are returned (lines 11-18).

While there are many different types of selection, we choose the most common type - roulette wheel selection [10]. This method is a genetic operator used in genetic algorithms for selecting potentially useful solutions for recombination. In roulette wheel selection, as in all selection methods, the fitness function assigns fitness to possible solutions. This fitness level is used to associate a probability of selection with each individual's configuration. If f_i is the fitness of individual i in the population, its probability of being selected is $p_i = \frac{f_i}{\sum f_N}$, where N is the number of individuals in the population. Two individuals are then chosen randomly based on these probabilities to produce offspring.

However, GA has a tendency to converge towards local optima or even arbitrary points rather than the global optimum of the problem. To escape the local minimum, Diversity Maintenance Module (DMM) is used in this algorithm. Specifically, "Shrinking Window Mutation (SWM)" [11] and "Simulated Annealing (SA)" [12] procedures are used in the *updatePopulation* routine. For SWM, the perturbation window shrinks as the optimization progresses. In SA, acceptance probability is calculated as:

$$P(fitness, fitness_{neighbor}, T) = \frac{1}{1 - e^{-\|fitness_{neighbor} - fitness\| \times (scaleFactor \times k + \frac{1}{k_{max}})}}$$

Furthermore, to improve the overall performance, we also consider the *preCalculate* module. Because GA procedure randomly generates the initial population, we cannot guarantee the quality of the configuration set. Thus, we stochastically retrieve the small set of population, about 10% of the original size, and pre-test via GA algorithms. If the result set cannot return the satisfactory solution set, the algorithm discard the population set and re-iterate the procedure.

More specifically, the *preCalculate* module randomly selects the small set among the existing population. As a main routine specified in Algorithm 3, in each generation, the fitness of every individual in the selected set is evaluated, multiple individuals are stochastically selected, and modified to form a new population. The algorithm is terminated when a maximum number of generations has been reached, and very small number is used for a maximum number of generations. The solution set is retrieved by this procedure and the average fitness value is used for evaluating the original population set.

V. RESULTS

A. Overview of Results

In this section, we demonstrate the performance of the proposed algorithm, the RRT path planning with GA learning technique. We performed experiments over various scenarios and investigate the results using specific performance criteria including the combined reward and planning time.

For the simulation, the challenge is to find the reasonable parameter values such as threshold values, population size and a max generation number in GA, an initial window size

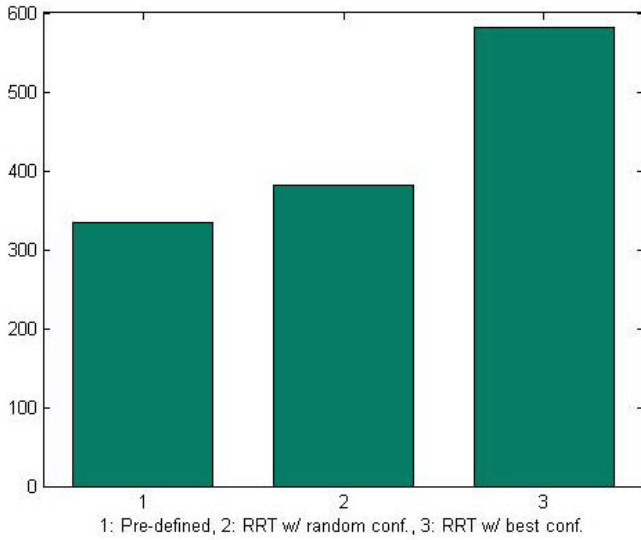


Figure 2. Overview of reward comparison

for Shrinking Window Mutation and a max trial number for Simulated Annealing. As we mention briefly in the previous section, we experimentally selected those values. Details are shown in Table I.

Table I
PART OF EXPERIMENTAL PARAMETERS

Parameter	Value	Note
# of UAVs	1-4	
iteration # in RRT	1,000	
population size	100	
max generations	10,000 & 200,000	
# of configurations	1-5	
iteration # in GA	100	
α	-200.0	Threshold for <i>preCal</i>
window size	3	Initial window size in <i>SWM</i>
K_{max}	20	Max trial # in <i>SA</i>
ϵ	10^{-3}	Stop condition in <i>GA</i>

As depicted in Figure 2, we compared average reward values for three different algorithms - Pre-defined path planning, RRT with a random configuration and RRT with GA. For a pre-defined case, we simply extend the path from the current position with the fixed distance and angle to mainly generate a spiral path. To perform experiments, 4 UAVs and 5 configurations are used, and at each step, the best configuration was dynamically selected among a configuration set obtained from GA. Figure 2 shows that the proposed algorithm showed much better results than other ways in terms of the average reward. Specifically, when we use RRT with GA to generate the path, the performance of the suggested algorithm was improved by 75.79% over the pre-defined case and by 53.68% over the RRT with a random configuration. Details are discussed in the following.

B. The RRT algorithm with a genetic algorithm

a) *Reward comparison over various costmaps*: In this part, we mainly focus on examining the RRT performance

over various scenarios in terms of the reward. To compare the RRT performance to the pre-defined path, we did experiments over seven different cases with various costmaps. Each case in Figure 3 and 4 represents the dynamic situations having different costmaps. Detailed information for each case is shown in Table II and III. As shown in Table II and III, we generated a combined costmap in a simulator using several simple static, random and mixed Gaussian costmaps to perform experiments.

Table II
CHANGING COSTMAPS IN GA: GETTING A CONFIGURATION SET

Case# in fig.3	Costmaps in GA
GA Case1	1 simple static costmap
GA Case2	1 simple static costmap + 3 random gaussian
GA Case3	1 static + 5 random gaussian
GA Case4	1 static + 5 random gaussian + 3 mixed gaussian
GA Case5	1 static + 5 random gaussian + 5 mixed gaussian
GA Case6	1 static + 7 random gaussian + 5 mixed gaussian
GA Case7	1 static + 10 random gaussian + 5 mixed gaussian
Testing environment	1 static + 5 random gaussian + 3 mixed gaussian + other vehicle's costmaps

Table III
CHANGING COSTMAPS IN RRT: TESTING OVER DIFFERENT COSTMAPS

Case# in fig.4	Costmaps in RRT planner
Testing Case1	1 simple static costmap + other vehicle's costmaps
Testing Case2	1 static + 3 random gaussian + other vehicle's costmaps
Testing Case3	1 static + 5 random gaussian + other vehicle's costmaps
Testing Case4	1 static + 5 random gaussian + 3 mixed gaussian + other vehicle's costmaps
Testing Case5	1 static + 5 random gaussian + 5 mixed gaussian + other vehicle's costmaps
Testing Case6	1 static + 7 random gaussian + 5 mixed gaussian + other vehicle's costmaps
Testing Case7	1 static + 10 random gaussian + 5 mixed gaussian + other vehicle's costmaps
GA Env1	1 simple static costmap
GA Env2	1 static + 5 random gaussian
GA Env3	1 static + 5 random gaussian + 3 mixed gaussian

Specifically, for getting the result in Figure 3, we used seven different cases, specified in Table II, to get a configuration set via the GA. Each case's costmap is applied to GA learning to get five best configurations. After getting the configurations we tested those seven sets over the same testing environment, also specified in Table II. The percentage of gain was calculated as $\% \text{ of gain} = \frac{\text{reward}_{RRT} - \text{reward}_{pre}}{\text{reward}_{pre}} \times 100$. As shown in Figure 3, the gain of RRT plan with GA over the pre-defined path was at least about 30% for any cases, and the gain was over 75% for the best case.

For Figure 4, we evaluated two different approaches. One approach is learning a configuration set from one group of costmaps, GA Env3 in Table III. Another approach is learning a configuration set from three different groups of costmaps, GA Env1 through 3. Because we select five configurations to make a set, we get one configuration from GA Env1, two configurations from GA Env2, and two more configurations from GA Env3. Once we get a configuration set, we test those parameters in RRT over various testing cases shown in Table III.

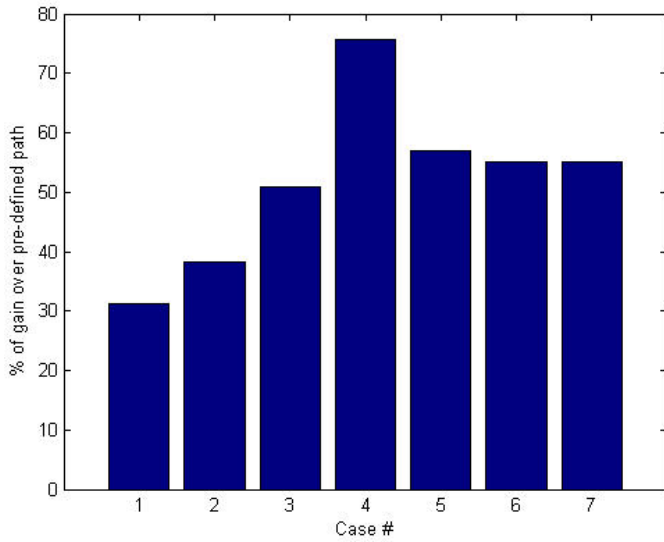


Figure 3. RRT: % of gain over pre-defined path ⇒ Change costmaps in GA.

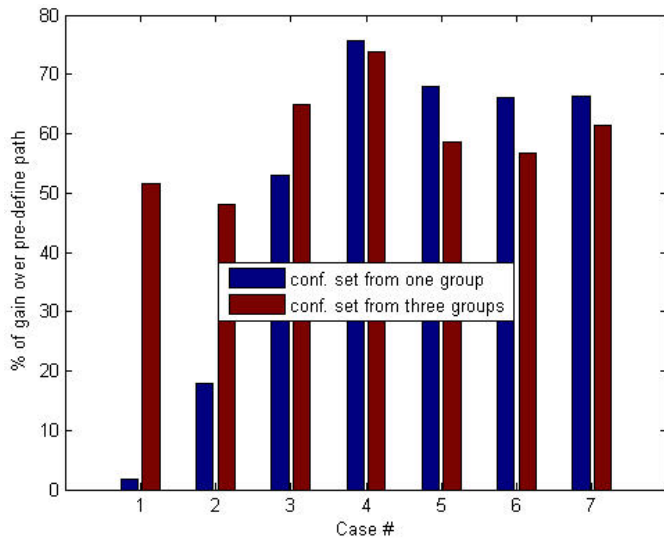


Figure 4. RRT: % of gain over pre-defined path ⇒ Change costmaps in testing environment.

As shown in Figure 3 and 4, the performance of case 4 was dramatically improved (about 75%). This is because case 4 has a very similar testing environment to a situation that a configuration set is obtained by GA. Thus, as we can expect easily, when we apply the best configuration set to generate a plan in the RRT planner, it generate much better results than a pre-defined case.

Also, for Figure 4, when we obtain a configuration set from three different groups of costmap, % of gain in Figure 4 has a smaller variance for the performance than the case that a configuration set is obtained from one group of costmap. This means if we get configurations from various environment and apply them dynamically, we can get fairly good performance for any general cases.

b) Reward comparisons with different configurations: In this part, we perform the experiments to show the performance of the suggested algorithm from more criteria: the static vs. dynamic method, different number of vehicles, different types of costmaps, and different number of configurations in a set.

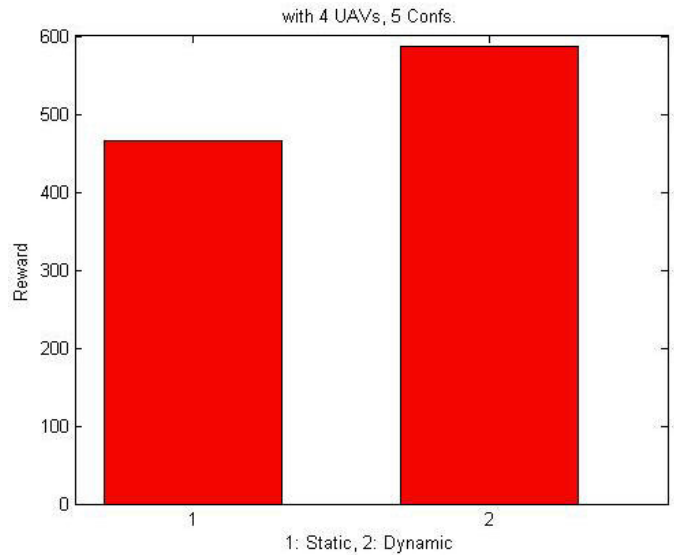


Figure 5. Static vs Dynamic (Reward)

The static method chooses the best configuration in the set at the initial time based on their fitness, whereas the dynamic method considers all configurations in the set to generate the path at each step, and chooses the best one dynamically. As shown in Figure 5, dynamic method showed 26.08% better performance than static case in terms of reward. This is because dynamic method evaluates all possible paths with configurations in the set, and selects the best case at each step. The interesting fact here is because we are using 5 configurations for the experiments, it takes approximately five times longer to generate a path when we use a dynamic method. This might look worse for the overall performance, but when we actually tested a dynamic case in a simulator, it was not a critical point because the planner practically does not have to generate the path frequently and planning still takes only about 3.5 seconds.

Figure 6 represents the average reward with different number of vehicles. We used one through four UAVs to perform the experiments, and compare the results. UAVs start at random positions. As shown in Figure 6, the reward difference is not large, and is caused by the RRT path planner considering other vehicles' costmaps to generate the path. Usually if we use more vehicles to collect information, we can get more information from other vehicles. The lower performance of case 3 might be caused by not enough experiment number and fundamental properties of randomized methods.

We also tested over two different cases specified in Figure 7. Case 1 initially sets three random Gaussian costmaps, and case 2 uses three different types of costmaps, and dynamically add more costmaps as the iteration number increases to get a

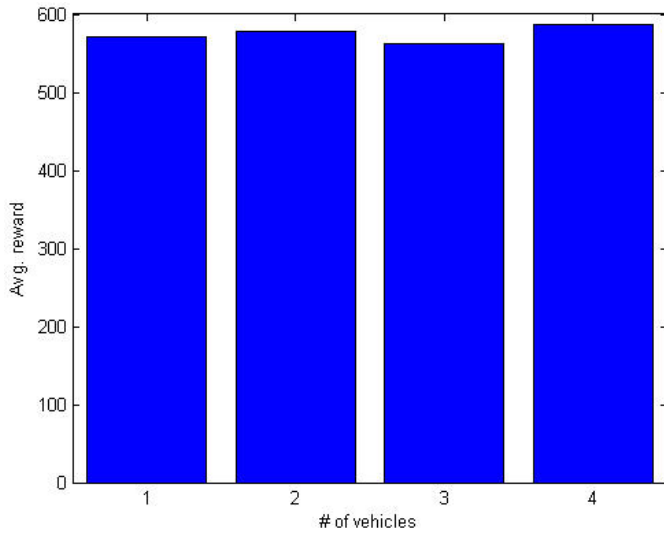


Figure 6. Different # of vehicles

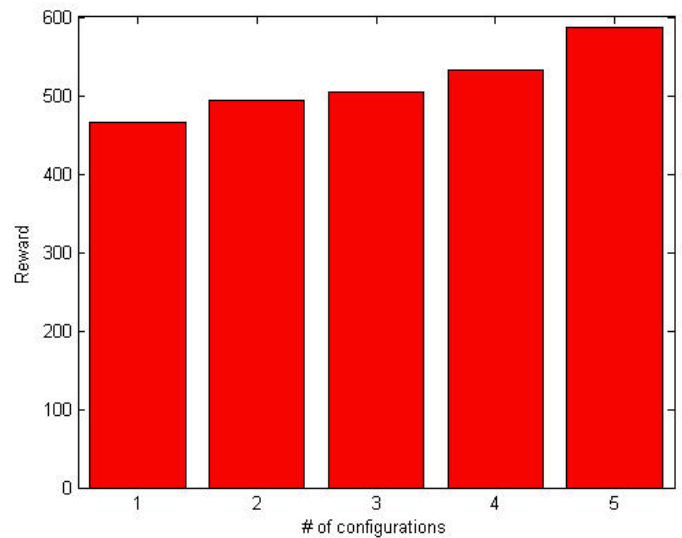


Figure 8. Different # of configurations in a set

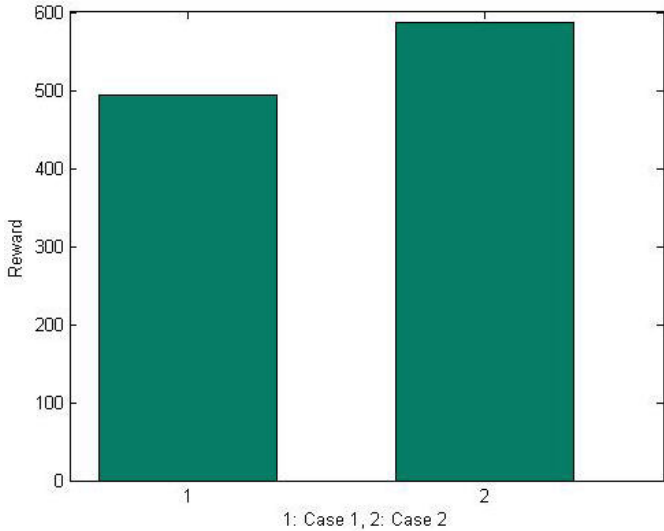


Figure 7. Different types of costmap to get a configuration set in the GA

configuration set via the GA. Case 2 starts with a simple static costmap and adds more random and mixed Gaussian and maps as the iteration number goes on. We also used 4 UAVs, 5 configurations, and applied a dynamic method to evaluate the path. As shown in Figure 7, case 2 showed the better result by 18.71% than case 1 because configurations in case 2 are trained with more complicated environment, and thus more applicable and flexible to a general environment.

For the last part, we also performed experiments with different number of configurations and measured the average reward. Because we used a dynamic method, if the planner has more configurations, it is highly likely to get higher reward. As depicted in Figure 8, the overall performance with five configurations was improved by 26.08%.

C. GA Learning

In this part, we demonstrate the performance of GA learning in terms of fitness value and time. As shown in Figure 9, we performed the experiments over eight different scenarios. Details about each scenario are specified in Table IV. In Table IV, *preCal* is the module for an optimization. We select a small set of population and apply GA with that and pre-evaluate it with the specific threshold value. *DMM* stands for Diversity Maintenance Module, and this module is used to escape the local optima - SWM: Shrinking Window Mutation, SA: Simulated Annealing.

Table IV
EXPERIMENTS SETUP

Experiment#	Description
Experiment1	Without any GA
Experiment2	Basic GA without update population
Experiment3	GA without preCal & DMM
Experiment4	GA without preCal & with DMM-SWM
Experiment5	GA without preCal & with DMM-SA
Experiment6	GA with preCal & without DMM
Experiment7	GA with preCal & DMM-SWM
Experiment8	GA with preCal & DMM-SA

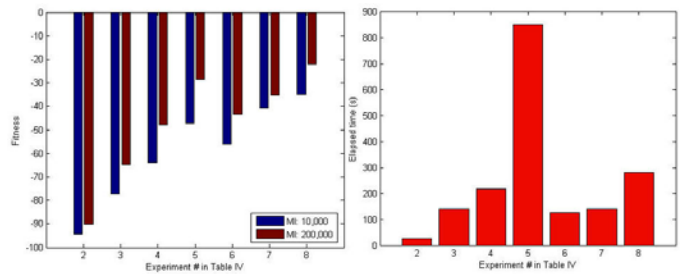


Figure 9. Fitness & time

We tested with two different maximum numbers of gen-

erations - 10,000 and 200,000 - over eight cases. For both numbers, experiment 8 showed the best result. This implies SA is better than SWM and the *preCal* module also helps to improve the average fitness. For the elapsed time, when maximum number of generations is 200,000 and SA without *preCal* is used, the algorithm showed the worst performance in terms of time. However, if we use *preCal* together, we can significantly reduce the elapsed time to get a configuration set.

VI. RELATED WORK

Much path planning work assumes that the environment is completely known before the robot begins its traverse [6]. The optimal algorithms in this search a state space (e.g., visibility graph, grid cells) using the distance transform [4] or heuristics [13] to find the lowest cost path from the robot's start state to the goal state. Cost can be defined to be distance travelled, energy expended, time exposed to danger, etc.

The stochastic or random approach was first introduced by Barraquand and Latombe [14], and later used by Overmars [15], and more recently by Kavraki [16]. The main idea behind these algorithms is to build a graph in the configuration space. The graph is obtained incrementally as follows: a local planner is used to try to reach the goal. Should the motion stop at a local minimum, a new node (or landmark) is created by generating a random motion starting from that local minimum. The method iterates these two steps until the goal configuration has been reached from one of these intermediary positions by a gradient descent motion. These algorithms work with a discretized representation of the configuration space.

Previous work in path planning has taken several approaches to planning with uncertainty. One of the most common approaches is to ensure proper operation in the worst case scenario. For example, if uncertainty is considered only in actuation, but not in sensing or modeling, Hait and Sim'eon [17] consider the range of possible rover poses and test for impact with the terrain. Related work by Esposito in the domain of plan validation [18] samples several possible values of the uncertain parameter from a given distribution and repeats planning for each value.

Another tactic for dealing with uncertainty is presented by Gonzalez and Stentz [19]. This work considers only actuation uncertainty, which is modeled as a zero-mean symmetric Gaussian. However, they are able to compute resolution-optimal paths for a point robot using the grid-based A* planner [13].

VII. CONCLUSIONS

This paper presented a novel path-planning algorithm for vehicles performing sensor readings in an environment where value of information from all parts of the environment is not equal. The approach built on ideas from RRT planners, but prioritized nodes from which to expand on the basis of their likelihood of leading to the best path. Key parameters of the planner were successfully learned via a genetic algorithm. The resulting planner was shown to be successful at finding paths that collected almost double the useful information as pre-defined paths.

In the near future, this planner will be applied to several physical vehicles as well as simulations of a range of different vehicle types. This usage will press the performance of the planner and, particularly, the learning algorithms. In real-world domains, additional constraints such as particular points the vehicle must go to or the need to refuel at specific locations will mean that the random expansion will need to be more carefully controlled. In addition, the proposed method will be tuned for automated learning of appropriate prioritization.

REFERENCES

- [1] J. Carsten, A. Rankin, D. Ferguson and A. Stentz, "Global Path Planning On-board the Mars Exploration Rovers," in *Proc. of the IEEE Aerospace Conference*, 2007.
- [2] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz and S. Thrun, "Anytime Dynamic A*: an Anytime, Replanning Algorithm," in *Proc. of the Int. Conf. on Automated Planning and Scheduling*, 2005.
- [3] K. Iagnemma, H. Shibly, A. Rzepniewski, and S. Dubowsky, "Planning and control algorithms for enhanced rough-terrain rover mobility," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2001.
- [4] Jarvis, R. A., "Collision-Free Trajectory Planning Using the Distance Transforms," *Mechanical Engineering Trans. of the Institution of Engineers*, Australia, Vol. ME10, No. 3, September, 1985.
- [5] Cipriano Galindo, Juan-Antonio Fernandez-Madriral, and Javier Gonzalez, "Improving Efficiency in Mobile Robot Task Planning Through World Abstraction," in *IEEE TRANSACTIONS ON ROBOTICS*, Vol. 20, No. 4, August, 2004
- [6] S. M. Lavalle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378-400, May 2001.
- [7] Faverjon, B. Tournassoud, P., "A Local Based Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom," in *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, 1152-1159, 1987.
- [8] Russell, S. J., Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice Hall, pp. 97-104, 2003.
- [9] Falkenauer, Emanuel, *Genetic Algorithms and Grouping Problems*, Chichester, England: John Wiley & Sons Ltd., 1997.
- [10] Goldberg D., *Genetic Algorithms*, Addison Wesley, 1988.
- [11] K. Rasheed and H. Hirsh, "Using Case-Based Learning to Improve Genetic-Algorithm-Based Design Optimization," in *Proceedings of the 1997 International Conference on Genetic Algorithms*, 1997.
- [12] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol 220, Number 4598, pages 671-680, 1983.
- [13] Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Publishing Company, 1980.
- [14] Barraquand, J. Latombe, J., "A Monte Carlo Algorithm for Path Planning with Many Degrees of Freedom," in *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, 1712-1717, 1990.
- [15] Overmars, M., "A Random Approach to Motion Planning," *Technical Report RUU-CS-92-32*, Department of Computer Science, Utrecht University, 1992.
- [16] Kavraki, L., Svestka, P., Latombe, J., Overmars, M., "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," *IEEE Transactions on Robotics and Automation*, 14(4), 566-580, 1996.
- [17] A. Hait and T. Sim'eon, "Motion planning on rough terrain for an articulated vehicle in presence of uncertainties," *IEEE/RSJ International Symposium on Intelligent Robots and Systems*, pp. 1126.1133, 1996.
- [18] J. M. Esposito, J. Kim, and V. Kumar, "Adaptive RRTs for validating hybrid robotic control systems," *International Workshop on the Algorithmic Foundations of Robotics*, July 2004.
- [19] J. P. Gonzalez and A. T. Stentz, "Planning with uncertainty in position: An optimal and efficient planner," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS'05)*, pp. 2435-2442, August 2005.