한국시뮬레이션학회 논문지
Vol. 27, No. 2, pp. 101-114 (2018. 6)

http://doi.org/10.9709/JKSS.2018.27.2.101
ISSN 1225-5904

Repast기반 진화 알고리즘을 통한 무인 비행체의 동적 경로계획 모델링 및 시뮬레이션

김용호[†]

Modeling and Simulation of Evolutionary Dynamic Path Planning for Unmanned Aerial Vehicles Using Repast

Yong-Ho Kim[†]

ABSTRACT

Several different approaches and mechanisms are introduced to solve the UAV path planning problem. In this paper, we designed and implemented an agent-based simulation software using the Repast platform and Java Genetic Algorithm Package to examine an evolutionary path planning method by implementing and testing within the Repast environment. The paper demonstrates the life-cycle of an agent-based simulation software engineering project while providing a documentation strategy that allows specifying autonomous, adaptive, and interactive software entities in a Multi-Agent System. The study demonstrates how evolutionary path planning can be introduced to improve cognitive agent capabilities within an agent-based simulation environment.

Key words : Evolutionary path planning, agent-based simulation, Genetic algorithm, Repast Simulator,

요 약

무인 비행체의 실시간 경로계획 생성 시 최적의 경로를 찾기 위한 다양한 연구가 진행되어 왔다. 본 논문에서는 진화알고리즘을 통한 무인비행체의 경로계획 생성을 수행하고, 이를 에이전트 기반 시뮬레이션 환경에서 구현 및 테스트가 가능함을 검증하였다. 이를 위해, Repast toolkit에 JGAP 패키지를 탑재하여 Java 기반의 유전 알고리즘 프로그래밍을 통한 무인비행체의 경로 계획을 생성하였고, 해당 결과를 에이전트 기반으로 시뮬레이션을 수행하였다. 본 논문에서는 에이전트 기반 시뮬레이션 소프트웨어를 소프트웨어 공학 개발 생명주기에 맞춰 문서화하여 설계 및 구현되었으며, 에이전트 모델링 설계는 자동화, 적응성 및 에이전트 간의 상호 작용에 초점을 맞추었다. 또한, 시뮬레이션을 통해 에이전트 기반 환경에서 설계한 모델 및 시나리오를 검증하여 다수의 비행 에이전트에 내재된 동적 경로계획 알고리즘이 실시간으로 자율적인 경로 생성이 가능함을 증명하였다.

주요어 : 진화알고리즘, 에이전트 기반 시뮬레이션, 유전 알고리즘, Repast 시뮬레이터,

1. Introduction

Autonomous vehicles such as unmanned aerial vehicles, robots, and unmanned ground vehicles are on demand by military and civilian operations (Cruz, et al., 2008). Such vehicles are capable of conducting high-level missions of tasks, as well as low-level actions without direct human control. Using such autonomous vehicles hazardous missions, such as exploration of planets in the universe, battlefield missions, and nuclear operations can be achieved safely. New simulation models, command and control mechanism and simulation tools have been developed to tackle issues for different aspects of the autonomous vehicle control problems. Currently, a UAV is operated by at least one ground operator. However,

Received: 22 February 2018, **Revised**: 17 April 2018, **Accepted**: 11 June 2018

[†] **Corresponding Author**: Yong Ho Kim
E-mail: yzk0302@gmail.com
Agency for Defense Development, Daejeon, Korea

the objective is to improve autonomy of such vehicles. To facilitate autonomic behavior, one of the challenge problems in autonomous vehicle development involves development of advanced dynamic path planning methods.

Path planning can be divided into two components, online and offline path planning. Offline path planning requires complete knowledge of the environment. So, all UAVs know locations of targets, obstacles, and geometry. With that information, they can generate optimal paths; however, it calculation of such paths is time consuming (Nikolos, et al., 2003). Many proposed path planning mechanisms are based on offline methods. In online path planning, UAVs fly with only limited information such as target's location, or mission tasks. Since online path planning mechanism does not have complete information, it is challenging to compute optimal path; on the other hand, computation cost is lower. Genetic algorithms mimic evolutionary biology, which involves the study of evolutionary process of human nature. Genetic algorithm is selected because research shows that evolutionary path planning could be an efficient and effective solution to path planning for swarm of unmanned vehicles (Besada-Portas, et al., 2010). In this paper, a genetic algorithm is presented to generate potential paths for UAVs in the presence of unknown obstacles in an environment. So, the objective is to develop an online path planning strategy for a group of UAVs. The strategy views path planning as an optimization problem; however, the solution is not completely optimal but rather near-optimal. For large-scale applications, we recommend combining the proposed solution with graph search algorithms. But, the main contribution of this paper is to show a software engineering solution that integrates agent-based simulation with a genetic algorithm package to support evolutionary path planning.

After implementing the genetic algorithm solution to the UAV path planning problem, the Repast agent-based modeling and simulation platform is used to demonstrate and validate its utility. In this paper, we demonstrate, verify, and validate the solution using this simulator to check whether the actual problem has been solved. Agent-based modeling and simulation involves specifying each individual agent and its interactions with other agents and the environment.

Overall, this paper involves designing and simulation of evolutionary path planning for UAVs while avoiding collision avoidance. So, we aim to demonstrate the simulation software engineering lifecycle from the inception of the basic concept statement of the problem to actual implementation of the selected UAV path planning method. A prototype in the form of a Repast simulation program is implemented. In Section 2, we overview existing methods in the extant literature. Besides, we discuss three critical elements of this project: (1) agent-based modeling, (2) the Repast toolkit, and (3) evolutionary path planning. In section 3, the UAV mission scenario is introduced as a problem statement. Analysis, design, and implementation of the solution are discussed in section 3 as well. Screenshots of the Repast simulator is provided to demonstrate how the simulated mission is executed. In section 4, using the simulation program, we illustrate the verification and validation of the solution. In section 5, we conclude by providing a summary of the paper along with a discussion on potential avenues of future research.

2. Related Works

Several different approaches and mechanisms are introduced to solve the UAV path planning problem. Methods in the extant literature aim to find optimal solutions depending on mission objectives and environmental components. Finding an optimal path often requires substantial computational time (Trovato, 1996). In some applications where the environment is not volatile and uncertainty is minimal, it is acceptable to allocate substantial computational resources offline prior to mission (Sanders, et al,. 2007). However, UAV's path planning method should be fast enough to support real-time planning and adaptable solutions for complex dynamic and evolving missions. To solve this problem, adaptive path planning (Chen 1995) has been introduced as a basis for a potential solution. Adaptive path planning uses past experience and learning mechanisms to improve future performance. In other words, it generates adaptive solutions based on previous data and environmental feedback. Researchers demonstrated the use of Evolutionary Algorithms as a viable and effective approach to solve path planning problems (Nokolos, et al., 2007).

Evolutionary models provide a useful and robust technique to deal with complex problems (Nokolos, et al., 2007). It has already been used to solve different types of the UAV problems. Based on the conviction that evolutionary path planning could be a candidate of solving complex path planning problem, many different evolutionary paths planning methods have been proposed. In (Kragelund, et al., 2002), authors present the application of evolution-based path planning within a field of obstacles distributed at uncertain location. In (Fu, et al., 2012), a new method based on genetic algorithm is also used in the presence of uncertainty with respect to locations of the obstacles in the environment. In addition, the authors implemented this solution using a simulated environment to determine its performance using the Matlab SimulLink tool. In (Bortoff, 2000), the authors proposed an evolutionary path planner that can be used in a realistic and risky scenario. Other researchers suggested evolutionary UAV path planning using B-splice trajectory calculations to implement realistic UAV flying trajectories (May, et al., 2010).

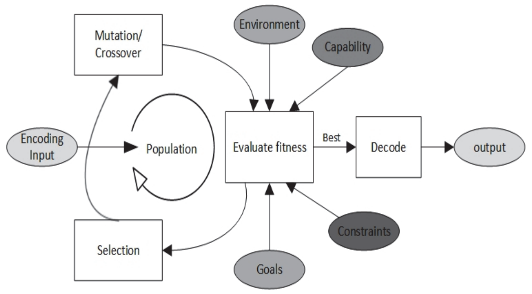


Fig. 1. Basic genetic algorithm process

All evolutionary path planning methods are integrated with the GA algorithm. EA-based path planning algorithms use a population of solutions and random modifications to those solutions to form a structured, stochastic search. GAs works iteratively on a population of candidate solutions, which are encoded to obtain a candidate corresponding to better solution. So, most GA based path planning methods have similar structure based on the GA architecture (Rathbun, et al,. 2002) and show the genetic algorithm that consists of several steps. So, starting from the initial population, each step for the GA algorithm, such as mutation, crossover, selection, and evaluation provides a basic building block. The differences between evolutionary path planning methods are based on how they implement those basic building blocks, environments, constraints, and goals.

The functional units of the GA algorithm shown in Figure 1 indicate the data components such as goals, constraints, and the path. Based on how we set up the path, environment, UAV capability, constraints, and goals, implementation aspects of algorithm change. Earlier work indicates that there is ample evidence on the viability and feasibility of the use of Genetic Algorithms for dynamic path planning. None of these studies were conducted in an agent-based modeling framework. This study demonstrates how evolutionary path planning can be introduced to improve cognitive agent capabilities within an agent-based simulation environment.

Repast is an agent based modeling and simulation platform. Agent based modeling and simulation is a new method to modeling a complex system composed of autonomous agents. Agents have behaviors, often described by simple rules, and interactions with other agents, which in turn influence their behaviors. By modeling agents individually, the full effects of the diversity that exists among agents in their attributes and behaviors can be observed as it gives rise to the behaviors of the system as a whole. So, this method is selected to develop the solution because of its ability to model complex adaptive system relatively easily. Ability to act autonomously is one of the most important reasons that we choose this framework and integrate with the actual evolutionary path planning algorithm.

3. Solution Analysis

3.1 Objective

The objective of this project is to demonstrate the efficacy of the evolutionary path planning method in an environment with multiple UAVs, targets, and obstacles using an agent-based simulator. Furthermore, collision avoidance and finding optimal paths are the major performance criteria.

3.2 Tools & Software Libraries

As a whole, the project needs a base toolkit which can allow integration of agent models with path planning algorithm into a coherent composite model to run and simulate the dynamics of mission. Repast is a free and open source agent-based modeling and simulation platform. Repast supports both the Java Programming Language and ReLogo APIs. In addition, since the project is aimed for agent-based simulation, Repast is a good match for the purpose of the study.

Since UAVs generate the path based on evolutionary mechanism, genetic algorithms and related functions are integrated with the cognitive components of agents. We used the Java based genetic algorithms package, which is also open-source. The Java Genetic Algorithm Package(JGAP) is a Genetic Algorithms Package, and genetic programming components are provided as a Java framework. Therefore, it is interoperable with the Repast platform and hence easy to import, because both libraries are built over the Java framework.

3.3 Simulation Environment & Constraints

For problem formulation, a simulation environment that is available for proper simulation and testing is required. It is impossible to create exact same simulation environment as real environment. Therefore, it is important to build a simulation environment by considering abstraction constraints. In this project, we used a two-dimensional airspace. Repast supports three-dimensional space and grid environment. However, we chose to use two-dimensional space and grid because of its complexity. By limiting the airspace, this would save the time to develop evolutionary algorithm, and the idea that if an algorithm works in two dimensional spaces it should be extended for use in a three-dimensional space with adjustment to genetic algorithm. In addition, even though the project is focused on UAVs, this constraint would work for autonomous ground vehicles, which use two-dimensional spaces.

To simplify the solution and focus on the implementation of the GA, conditions such as weather condition, air resistance are excluded from the problem. Since the project concentrates on the path planning mechanism, we disregarded the environment facts. Furthermore, motion of UAV is also limited by grid based. In case of aircrafts it needs a turning angle. So, because of this limitation, UAVs are not expected to make sudden change in their direction. However, since we would like to focus more on evolutionary path planning as well as entire autonomous vehicle, grid based movement of the vehicle is sufficient for the purpose of this project.

3.4 Problem Formulation

Currently, Autonomous Vehicles play an important role in military and civilian operations. They are often used for reconnaissance or video recording for accessing human unreachable areas. Since an AV still needs at least one ground operator, the ability to control by ground operators is important to conduct AV missions. However, depending on the weather condition or limitation of human ability, sometimes it is challenging to sustain an AV in a safe mode. Because of the limitations of human control, autonomous AV support systems are becoming more desirable. So, this project is focused on unmanned aerial vehicles that are able to fly automatically by generating paths dynamically on their own. Therefore, we assume an UAV that flies with only limited information such as locations of targets, and hazardous areas. Other than that UAV should be able to fly automatically without any human control and information starting from departure to landing. While UAV flies, it should also able to manage collision avoidance.

3.5 Scenario Analysis

We assumed that each AV has targets in order of user defined sequence. The only input that the AV knows is the location of the target. Starting from an initial position, AVs move toward the first target in their mission list. As soon as they reach first target, they conduct surveillance and then move toward the next target location. AVs continue to conduct their mission, until they arrive at last target's location. In this scenario, we also assumed that AVs have sensors that are able to detect collisions. The sensor has a maximum threshold for the sensor range, so AVs can only detect obstacles only within the sensor range. This limitation also causes AVs to generate waypoints with maximum of certain range from current position. Collisions with other UAV are also avoided.

With the environmental representation and scenario above, we transformed the abstract scenario into the simulator as a model. The model is composed of three major components: Objects, Path Generations, and the Evolutionary Algorithm (EA) model. All three models are interdependent and are integrated within the Repast Platform. From here, we use UAV as representative of AVs since the scenario is based on UAVs.

The Repast platform embraces all objects. Objects are the visible components of the simulation, and they can be divided into UAVs, Targets, and Obstacles. Another main part of the model is Path Generator. UAVs are able to generate the path on their own, so path component should be designed as part of the cognitive architecture of the UAV agent. As a result, whenever UAVs need to get new path for safe flight, it can be done in a real-time and without depending on external source. Last major component is the Evolutionary Algorithm (EA). EA takes a role of deliberation mechanism within the cognitive architecture of the UAV. It is fundamentally based on the genetic algorithm. EA is used to define the parameters of a path for UAV. Therefore, the EA component is comprised in the Path Generator component. Whenever UAV needs to create a path, the EA is used to create the optimal path for the UAV.

3.6 UAV Analysis

The main component of this project is an autonomous vehicle especially the Unmanned Aerial Vehicle. UAV must implement correct path planning method to conduct a successful mission. It must be able to detect possible collisions while they are flying over the area, and it should have an ability of generating optimal path to the target for itself.

Initially, when the UAV is launched, a mission is assigned to the UAV. Following the mission assignment, all the geographic information, target's location, and input variables are received from the based station. As soon as the UAV receives its inputs, it creates the initial path to the target and heads toward that target. If the UAV detects obstacles or another UAV using its sensors, it will generate new path to the target while considering collision avoidance. After the UAV reaches its target, it will finish the current task in the mission and start engaging with the next task. For more detail explanation, therefore, process activity diagram could be used to show the activities within the core control system of the UAV. To detect obstacles in the space, we assumed that each UAV has built-in sensors. UAV sensors have sensor ranges that limit their perception to certain amount of distance. Four arrows demonstrate range length of x-axis and y-axis. Black large square indicates the entire range of UAV's perception.

4. Solution Design

4.1 Overview

With the knowledge of high-level structures that we discussed above, this design phase is intended to implement from high-level concepts into actual program by developing the simulation software. So, the purpose is to understand and demonstrate how well evolutionary path planning for Unmanned Aerial Vehicle works in an environment that has multiple UAVs and targets.

4.2 State Variables and Scales

The models, Object, Path, and EA, consist of the following entities, which are described in Table 1. Parameters may change during the simulation or settled when initializing the simulator. Length of time in the simulation is composed of time ticks. One tick is same as one step of time in the simulation.

Table 1. model Entities

entity	Parameters	Descriptions
UAV	CurrentPosition	X and Y coordinate of current Vehicle's location
	Number of vehicle	The ID of the vehicles in space
	Speed	How fast vehicles are moving
	Direction	Current direction of the vehicle in space
	arrived	Check UAV finishes mission
Obstacle	Position	X and Y coordinates of current Obstacle's location
	Number of obstacles	The number of obstacles in space
Target	Position	X and Y coordiates of current Target's location
	Number of Target	The number of targets in space
Path	Number of waypoints	The number of waypoints that make a path
	Map boundaries	size of the space
EA	population size	A group size of chromosome
	Fitness value	A value that decides optimal candidates of chromosome
	The number of generation	the number of population evolve
	operators	crossover OR mutation
	Length of chromosome	The number of genes in the chromosome

4.3 Process Overview and Scheduling

The UAV model and the path model are updated in every time step. This could be explained by UAV activity diagram of the UAV agent.

During the initialization phase, the UAV receives targets as an input. Then, it retrieves first target from target list and gets direction to target. Before UAV moves toward target, it needs to complete several steps. The UAV first checks its current position with respect to the current target position. If two positions are same, check this target is the last target in the list in order to finish the mission. If two positions are not same or the target is not the last one, the UAV senses obstacles within the sensor range. Based on the result from detection, UAV decides whether it should move toward the current target, generates a path using the UAV fitness function.

UAV moves at least one grid point from current position depending on its speed and direction. Within every time tick, the UAV tries to detect obstacles as well as other UAVs that are apart at most five grid points from it. Until then, UAVs will keep going to the first ordered target point. To assist the UAV, the path model retains the latest path information for the UAV. Path model keeps record of the starting position, target position, and lists of waypoints. Then, as soon as a UAV detects a possible collision, another model, which is the EA model, proceeds to compute a new set of waypoints. EA model will start calculating new path for UAV based on the genetic algorithm, then send it to the path model to update the current path.

As soon as the Path model requests a new path, the EA model starts to generate new waypoints. It randomly generates paths into population. The number of population is set at first. Having multiple possible paths in the population, each path gets a fitness value from the Fitness Evaluator. Different fitness function is used depending on whether UAV or Obstacles are detected by UAV. Basically, calculating shortest length and intersection rates are the main components of the fitness function. After getting fitness value, all path candidates, which are chromosomes in the Evolutionary Algorithm, apply the crossover and mutation operators. Both operators are used in the genetic algorithm to maintain solution diversity. So, they are able to generate new path candidates into population by retaining diverse set of solution. Finally, the EA model chooses the best current candidate path in population based on the fitness value. This circulation lasts until the number of evolution reaches to threshold, and then it returns current best path to the Path model.

4.4 Design concepts

4.4.1 Fitness

The model seeks fitness model explicitly in the evaluation function. Each potential individual paths calculates its fitness value using the evaluate function.

The method of measuring fitness will be discussed in a submodel.

4.4.2 Sensing

UAVs are assumed to know their status, position, target position and sensing range. Each vehicle will sense obstacles or other vehicles if the objects are within their sensing range.

4.5 Initialization

A two-dimensional grid space is initially generated by locating multiple UAVs, targets, and obstacles. When a UAV is created, it will head for the target following a straight line. The path consists of an array of two integer values. Each integer value will be encoded to IntegerGene. After that, two IntegerGenes will be encoded by a CompositeGene. Therefore, the CompositeGene finally becomes a waypoint. Each CompositeGene waypoint will be the member of the chromosome, denoting the path. Also, location of the target and the obstacle area are determined either by the user through parameterization or predetermined randomly.

4.6 Input

The number of UAVs is an input variable that is varied to test the scalability of the simulation as well as the efficacy of the collision avoidance strategy. The environmental change is also an input. As the locations of obstacles change the environment also changes. The path model will take the start position and target position as inputs. In the EA model, genetic algorithm has important aspects such as how to produce offspring and set the population size. Mutation and crossover rates are used in this model to produce optimal path solutions for each UAV.

4.7 Submodels

4.7.1 Genetic algorithm

The general process for the application of the genetic algorithm is as follows: (1) Sample set of potential solutions are generated randomly. (2) Within the sample set, poor solutions are removed while better solutions are retained. (3) Survived samples perform some of the operations like crossover and mutation to derive new possible solutions. (4) Based on the fitness function, the best possible solution is selected. (5) Step 2 to step 4 repeats until a defined criterion is satisfied. (6) Return best possible solution as a result.

To make the genetic algorithm work effectively, a few conditions should be met. It should be able to evaluate how good potential solution is relative to other potential solutions that are not optimal. The critical components in a Genetic Algorithm are the design of gene, the application of the evolutionary operators, and the fitness function. The Gene represents distinct aspects of the solution, and genes are gathered together to define the chromosome. This chromosome representation is important to determine how the problem is structured in the algorithm and the genetic operators are applied. Genetic Algorithm has two building block mechanics as a whole, Crossover and Mutation. In Genetics, Crossover swaps genes between two chromosomes. Mutation randomly alters the gene in a chromosome. Internal mechanisms are different based on the topic of the project. Finally, all the decisions are made in Fitness Function. After chromosomes pass through the previous step, all they left to do is passing the fitness function. However, to adapt genetic algorithm into real world project, we need to make fitness function to select a good potential solution.

4.7.2 Chromosome structure

Chromosome consists of multiple genes. Each chromosome is based on sequence of genes from certain value. To build a chromosome, genes must be defined first. JGAP (Java Genetic Algorithm Package) supports the creation of chromosomes. The genes could be binary, floating point, integers, or symbols. Since we are using a two-dimensional grid space, we decided to create value of gene in Integer. JGAP has already built an IntegerGene class and it is ready to use. So, we put x coordinate value into that gene. However, because IntegerGene only can contain one integer value, we created another IntegerGene for the y coordinate value.

JGAP also supports the combination of two simple genes into a combined gene, named the CompositeGene shown in Figure 2.

CompositeGene plays a role as a waypoint of the UAV. Therefore, chromosome is composed of multiple of CompositeGenes which combined with two Integer Genes x and y. then the chromosome becomes a possible path for UAV.

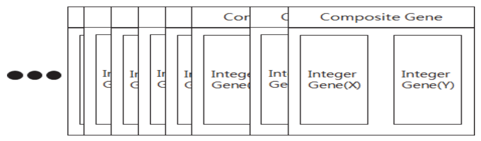


Fig. 2. The Design of the chromosome

4.7.3 The Fitness Function

Within possible chromosomes, the algorithm should be able to identify the best possible chromosome. To distinguish every possible chromosome, chromosome in the population will be assigned a value, which is the fitness value. Increasing values for the fitness indicate better performance in our design. To determine each fitness value of chromosome, every possible chromosome must be processed by the fitness function. In this project, the fitness function evaluates the performance of a path in two aspects: distance from the obstacle and closeness to the target. These aspects are, defined in terms of equations (1) and (2).

$$Obstacle(p) = \sum_{i=0}^{n} iSect(wayP_i, wayP_{i+1}) \qquad (1)$$

Where obstacle(p) denotes the total penalty value p, iSect(wayP$_i$,wayP$_{i+1}$) is the function that returns a value if a line between two way points intersects with detected obstacle.

$$Distance(p) = \sum_{i=0}^{n} dwt(wayP_i, T) \qquad (2)$$

Where T denotes location of target and dwt(wayP$_i$,T) denotes distance between waypoint and target. So, distance(p) contains total length of each way Points to the target. In addition to these aspects, another equation (equation (3)) is needed in case detected obstacle is a UAV.

$$UAV(p) = \sum_{i=0}^{n} iSect(wayP_i, wayP_{i+1}, \alpha\, UAV) \qquad (3)$$

Where UAV(p) denotes the total penalty value p, is the function that returns a value if a line between two waypoints intersects with detected another UAV. Since each encountered UAVs generates new path simultaneously, the collision avoidance between UAVs is expected from this calculation.

Therefore, the fitness is computed in (4) or (5)

$$\text{Fitness Value} = (\text{obstacle}(p) + \text{distance}(p))^{-1} \qquad (4)$$

$$\text{Fitness Value} = (\text{obstacle}(p) + \text{distance}(p) + \text{UAV}(a))^{-1} \qquad (5)$$

The return values from obstacle(p), UAV(p), and distance(p) is optimal when both values are as lowest as possible. However, fitness value will choose best possible chromosome with highest value. Therefore, final equation remains reciprocal number of sum of all outcomes like (4) or (5).

4.7.4 Evolutionary Operators

In this paper, two operators are used the crossover operator and the mutation operator. These two operators play an integral role in the genetic algorithm. They retain good candidates in the population. Operators can be modified in many ways, but we used two methods named Best Chromosome Crossover and All Mutation methods. These operators operate immediately after the fitness function. After each chromosome is assigned a fitness value by the fitness function, Best Chromosome Crossover will find two best fitness chromosomes from population. Two selected chromosomes randomly choose one of their genes and crossover all of other genes to the end as shown in Figure 3.

In Figure 3, Gene3 is chosen randomly, then all the remaining genes after Gene3 crossover until the last gene. Therefore, existing genes and newly generated genes are retained in the population for the next generation. All Mutation picks a chromosome from the population, and then randomly chooses one of the genes. With selected gene, BCM will generate totally new gene, new X and Y points in this project, and replace old gene with new one. above figure provides graphical explanation.

Figure 3 shows that the third gene randomly selected and is replaced by new gene. So, they both become one of the possible candidates.

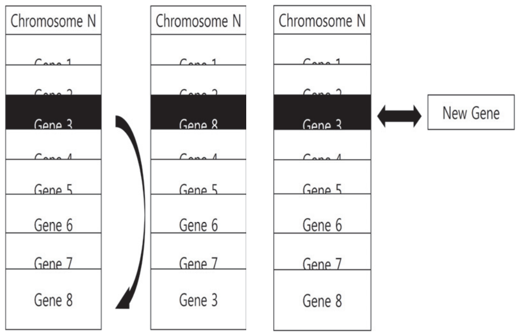


Fig. 3. The Crossover Operator & The Mutation Operator

4.7.5 Evolution

Evolution is a process that generates new generation of the path. It will evolve until the number of times user specified. One generation cycle needs several steps. All chromosomes in the population pass through fitness function, and receive new fitness values. After that, those chromosomes are transformed by two operators: crossover and mutation. After all chromosomes are processed, newly generated population will merge into old Population. In the meantime, if population size is limited, worst chromosome candidates will be dropped from the population. With the new generated population, another cycle is performed until the maximum number of generation has been reached.

4.8 Implementation

Besides development of the source code of the program, several steps must be established to run a simulation in this Platform. The metadata that the Repast Symphony runtime uses to help create displays and other runtime components should be updated. The context.xml file under the PathPlanning.rs folder should be edited as Table 2.

Table 2. context.xml

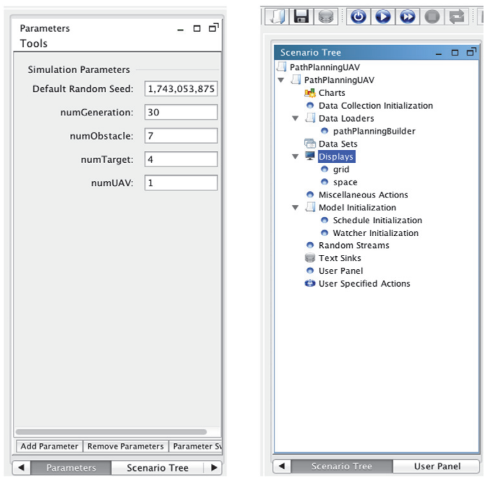
```
<context id="PathPlanningUAV"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="http://repast.org/scenario/
context">
            <projection type="continuous space"
id="space"></projection>
            <projection type="grid" id="grid"></projection>
</context>
```

4.8.1 The Scenario Tree

Before running the simulation, we had to setup Scenario Tree which is left side panel of the simulator in Figure 4. The Scenario Tree is able to connect my context builder to simulator by specifying the data loader. (1) In the Scenario Tree, right click on the Data Loaders and click “Set Data Loader”. (2) In the “Select Data Source Type” window, click on “Custom ContextBuilder Implementation. Click Next. (3)Choose “PathPlanningUAV.pathPlanningBuilder”. Click Next (4) Click Finish

These steps create “pathPlanningBuilder” as the name of the Data Loader and continue to create display. Steps are following: (1) In the Scenario Tree, Right click on Displays and click “Add Display” (2) In the Display configuration dialog, type Space Display for name. Leave 2D as the type because we are using 2D space. (3)Select space projection in the “Projection and Value Layers” section and then click the green arrow. The projections on the right are the ones will be displaying and those on the left are all the possible projections to display. (4)Click next. (5)Select UAV, Obstacle, and Target agents. Move them to right by clicking arrow. Click Next. (6) In this panel, we can configure what we want the UAV, Obstacle, and Target look like. (7) Click each agent, and open 2D shape Editor to change colors and shapes. After setting all three of agents, Click next (8) Click Next. Click Finish

These will create Space display under Display node in the Scenario Tree. Save this by clicking save button on the toolbar. Finally, we can now run our model.


Parameters
Tools
Simulation Parameters
Default Random Seed: 1,743,053,875
numGeneration: 30
numObstacle: 7
numTarget: 4
numUAV: 1

Scenario Tree
- PathPlanningUAV
  - PathPlanningUAV
    - Charts
    - Data Collection Initialization
    - Data Loaders
      - pathPlanningBuilder
    - Data Sets
    - Displays
      - grid
      - space
    - Miscellaneous Actions
    - Model Initialization
      - Schedule Initialization
      - Watcher Initialization
    - Random Streams
    - Text Sinks
    - User Panel
    - User Specified Actions

Fig. 4. parameters

Figure 4 is depicted final Scenario Tree.

There are five parameters in this project shown in Figure 4. *numGeneration* parameter is able to set up maximum number of generation for Genetic algorithm. The default value is 30. It is very important parameter because it directly affects to UAV path creation. *numObstacle* parameter literally creates zero or more squared obstacles as user specified in the space. The default value is 7. *numTarget* parameter creates one or more targets that UAVs have to arrive during their mission. The default is 4. Lastly, *numUAV* parameter sets the number of UAV in the simulator. The default value is 1.

When the program executes, it launches Repast simulator as shown in Figure 5.The User panel on the left (Scenario Tree, Figure 4) reveals the runtime configuration of the model. It must be done before running the simulation

As shown in Figure 5, we can monitor the simulation in the middle of simulator display. We can run the simulation by clicking the Run button at which upper side of problem, step through each timestep with the Step button, stop the simulation with the Stop button, and reset it for another run with the Reset Button. I will not cover rest of the buttons because they are not relevant in this project.

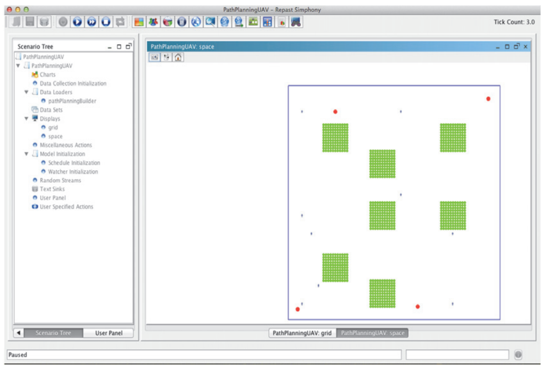


Fig. 5. Demo Screenshot of Simulation

5. Solution Validation and verification

5.1 Verification

To ensure that the program has been built according to the requirements and design specification, we verified the project using three testing method; Module, Integration, and System test. The module test shown in Tables 3 verifies individual methods. Each function has a pre-condition and post-condition. Pre-condition is used for evaluating the parameters before the function is invoked. Post-condition is used after the function executes. Integration testing is required after all methods tested in the module testing phase. Integration testing combines and tests individual modules as a group. Integration testing verifies functional, performance, and reliability requirements placed on the critical feature items of this project. we assumed path creation using the genetic algorithm and correctness of the GA as critical items for the project.

5.1.1 Module test

Functions in the table below are selected as part of module testing because they are repeatable, consistent, and fast. Repeatable means we can rerun the same test as many times as we needed. Consistent means every time we run with same parameter, we get the same result. That is why we used BlackBox testing for this testing. By only examining pre-condition and post-condition, we can be assured that a function is working with respect to its functional specification.

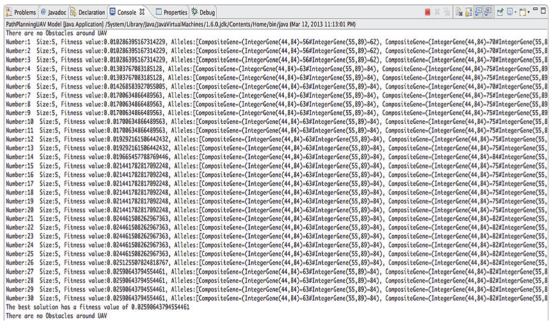
5.1.2 Integration Test

(1) Path Creation using a Genetic Algorithm

To illustrate integration testing, we choose path creation mechanism that involves integration of the simulator with the genetic algorithm functionality. To generate a path, multiple methods are combined and orchestrated, and it can be done via System Sequence Diagram. In addition, the Genetic Algorithm requires computational time to calculate fitness values and to evolve candidate paths.

To check whether the system actually generates the best possible path as an output, we configured the system to display the path information during every each generation. Figure 6 shows the output trace. Detail explanation of Figure 6 is following: *Number* indicates the number of evolution in Genetic Algorithm. *Size* total length of chromosome. *Fitness Value* the value that determines best chromosome. *Alleles* List of waypoints.

By checking the output information above as well as monitoring the visualization, the system was tested with respect to its functional, performance and reliability aspects.



```
PathPlanningUAV Model [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Mar 12, 2013 11:13:01 PM)
There are no Obstacles around UAV
Number:1  Size:5, Fitness value:0.010286395167314229, Alleles:[CompositeGene-(IntegerGene(44,84)-56#IntegerGene(55,89)-62), CompositeGene-(IntegerGene(44,84)-70#IntegerGene(55,8
Number:2  Size:5, Fitness value:0.010286395167314229, Alleles:[CompositeGene-(IntegerGene(44,84)-56#IntegerGene(55,89)-62), CompositeGene-(IntegerGene(44,84)-70#IntegerGene(55,8
Number:3  Size:5, Fitness value:0.010286395167314229, Alleles:[CompositeGene-(IntegerGene(44,84)-56#IntegerGene(55,89)-62), CompositeGene-(IntegerGene(44,84)-70#IntegerGene(55,8
Number:4  Size:5, Fitness value:0.01303767083185128, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,89
Number:5  Size:5, Fitness value:0.01303767083185128, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,89
Number:6  Size:5, Fitness value:0.014268583927055005, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-70#IntegerGene(55,8
Number:7  Size:5, Fitness value:0.01700634866489563, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,89
Number:8  Size:5, Fitness value:0.01700634866489563, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,89
Number:9  Size:5, Fitness value:0.01700634866489563, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,89
Number:10  Size:5, Fitness value:0.01700634866489563, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,8
Number:11  Size:5, Fitness value:0.01700634866489563, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,8
Number:12  Size:5, Fitness value:0.019292161506442432, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:13  Size:5, Fitness value:0.019292161506442432, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:14  Size:5, Fitness value:0.019665457788769446, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-84#IntegerGene(55,
Number:15  Size:5, Fitness value:0.021441782817092248, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:16  Size:5, Fitness value:0.021441782817092248, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:17  Size:5, Fitness value:0.021441782817092248, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:18  Size:5, Fitness value:0.021441782817092248, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:19  Size:5, Fitness value:0.021441782817092248, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:20  Size:5, Fitness value:0.021441782817092248, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-75#IntegerGene(55,
Number:21  Size:5, Fitness value:0.02446150826296736З, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,
Number:22  Size:5, Fitness value:0.024461508262967363, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,
Number:23  Size:5, Fitness value:0.024461508262967363, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,
Number:24  Size:5, Fitness value:0.024461508262967363, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,
Number:25  Size:5, Fitness value:0.024461508262967363, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,
Number:26  Size:5, Fitness value:0.025125507824818767, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,
Number:27  Size:5, Fitness value:0.02590643794554461, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,8
Number:28  Size:5, Fitness value:0.02590643794554461, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,8
Number:29  Size:5, Fitness value:0.02590643794554461, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,8
Number:30  Size:5, Fitness value:0.02590643794554461, Alleles:[CompositeGene-(IntegerGene(44,84)-63#IntegerGene(55,89)-84), CompositeGene-(IntegerGene(44,84)-82#IntegerGene(55,8
The best solution has a fitness value of 0.02590643794554461
There are no Obstacles around UAV
```

Fig. 6. Best candidate chromosome information on console output

(2) Correctness of Genetic algorithm

In this task we checked the correctness of our implementation of the Genetic Algorithm. Even though paths are generated within the system as discussed before, we cannot be sure whether the algorithm is actually producing an optimal path. To verify correctness of the algorithm, we used the time step which translates into a time tick in the simulator. There are two reasons for using the time tick:

- Based on the theory of the Genetic Algorithms, the more the number of generations in a Genetic

Table 3. UAV & Genetic algorithm & Fitness Function conditions

Name of function	pre condition	post condition
getDirection	receive target location and knows current location	returns direction to target in degrees
radar	list of gridcells where obstacles possibly exist	returns the number of detected obstacles
almostEqual	receives two gridpoints	returns true if two points are almost equal, false otherwise
Forward	list of multiple waypoints	moves UAV next waypoint
createPath	UAV detects obstacle	generate new path
createPathUAV	UAV detects other UAV	generate new path
GAsetup	UAV calls createPath method	Initialize GA model
GAconfiguration	GA algorithm initialized	settings for genetic algorithm
pathToGene	GA model instantiated	Perform genetic algorithm and return waypoints
generatePath	function receives best possible chromosome	Transform chromosome to waypoints
IsInterSect	Two waypoints are given and one gridpoint to check collision	Returns true if gridpoints lies on line between two waypoints, false otherwise
nearFromObstacle	Receives a candidate chromosome	returns fitness value of the chromosome
findshortest	receives a candidate chromosome	returns fitness value of the chromosome
Distancebetweenpoint	Two points are given	returns distance between two points in double
getPositionGene	Function receives a chromosome and index number to get	returns a gene of specified index in chromosome

Algorithm, the higher the possibility of generating an optimal path

- The UAVs make at least one move for the target in every time step.

For UAV, optimal path in free space would be a straight line to the target. If the distance between UAV and target is 10 in grid space, UAV will need 10 tick of time to reach the target point because UAV makes a move in every time step. However, when the UAV encounters an obstacle and invokes the genetic algorithm, a path comprised of multiple waypoints is created. As we described before, the larger the number of evolutionary generations in genetic algorithm, the higher the fitness of the generated path. An optimal path is one that is close target, able to avoid obstacle, and short. Therefore, the less time ticks are expected with large number of evolutionary generations. Based on these reasons, the testing checks the time tick to verify "how long does UAV take to get to all the targets". Following Figure 7 and Table 4 show the observed results.

The above results are showing that as the number of generations increases, UAV's path get close to near optimal. Therefore, we can verify that our genetic algorithm is functioning correctly.

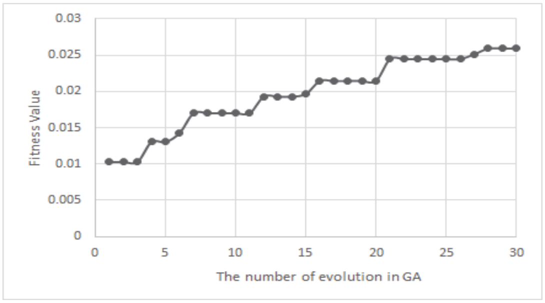


Fig. 7. Fitness value of the generated path

Table 4. Observed generation result per Time Tick

The Number of Evolutionary Generations	Tick Count
10	720
20	559
30	513
50	449

5.2 Validation

We compared specified requirements, which are proposed during initial phase, with finished product in order to ensure that the product actually meets the user's requirements.

- The program should be able to show visual display to user.
- The simulation must be able to test path planning mechanism of UAV in the simulator
- Each UAV must avoid collision with obstacles and other UAVs
- UAV must reach the assigned targets in the correct order

5.2.1 validation Testing

In order to validate the overall operation of the simulation, we tested with 3 UAVs, 7 squared obstacles, and 4 targets as shown in Figure 8. we only concentrate on the path of one UAV (blue colored, pointed by black arrow in Figure 8). The order of targets for this UAV is also numbered in Figure 8. In addition, Path to target #1 is expected similar to Red and Blue paths in Figure 8.

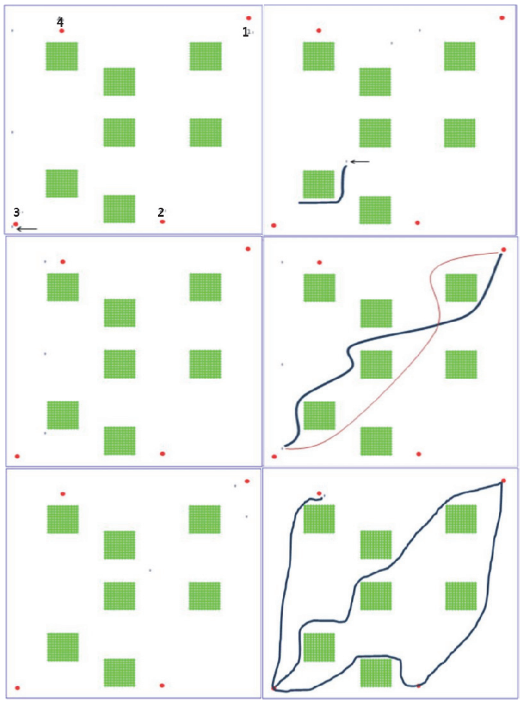


Fig. 8. Simulation Results

Figure 8 shows that UAV is detecting the first obstacle (on the Left) and avoiding from it (on the Right). This also demonstrates that Path is almost near optimal as we expected in Figure 8. Circumstance that is depicted in Figure 8 demonstrates UAV collision avoidance. Two UAVs are aware of another UAV's movement. So, they immediately calculate new path. The new path is avoiding from collision as well as towards to next target. Lastly, UAV visited all targets and arrived last target location shown in Figure 8.

6. Conclusion

In this paper, a new simulation method based on Genetic Algorithms is presented to realize dynamic UAV path planning in an environment with obstacle and multiple target points. The path planning model is based on a two-dimensional grid map. Adaptive evolutionary planning is adopted based on a set of criteria to generate path to avoid obstacles as well as other UAVs. The simulation model is implemented using the Repast platform to visualize and test the simulated scenario. The modeling and simulation study of dynamic path planning is conducted in accordance with the software engineering life cycle.

As simulation results indicate, evolutionary path planning using genetic algorithms has potential to improve adaptivity of autonomous vehicle. Even though the algorithm does not return the best path, it demonstrates that collisions can be avoided from obstacles as well as moving obstacles using the near optimal path. This path planning mechanism demonstrates that it can be used in volatile environments as depicted by the scenarios used in this project.

Since the evolutionary dynamic path planning enables us the ability of collision avoidance while generating near optimal paths in real time, future research is warranted. The scenarios can be simulated under more realistic environment with accurate turning angles, speed, and altitude of vehicle. This presents additional challenges in design and implementation because of its computational complexity, but it would give more accurate results than the current simulation does. Once the algorithm is verified with acceptable results, the final step would be testing with real AVs in real world with appropriate missions.

References

Besada-Portas E, de la Torre L, de la Cruz J.M, & de Andrés-Toro, B. (2010). Evolutionary Trajectory Planner for Multiple UAVs in Realistic Scenarios. Robotics, IEEE Transactions. 619-634. doi: 10.1109 /TRO.2010.2048610

Bortoff, S. A. (2000). Path planning for UAVs. In Proc. of the American Control Conference, pages 364– 368, Chicago, IL.

Ioannis K. Nokolos, Eleftherios S. Zografos, and Athina N. Brintaki. UAV Path Planning Using Evolutionary Algorithm. p77-78 (2007).

J.M de la Cruz, E. Besada-Portas, L Torre-Cubillo. (2008). Evolutionary Path Planner for UAVs in Realistic Environments.

Kan Ee May, Ho Jiun Sien, Yeo Swee Ping, & Shao Zhen Hai. (2010). An evolutionary algorithm for multiple waypoints planning with B-spline trajectory generation for Unmanned Aerial Vehicles (UAVs). Computational Problem-Solving (ICCP). 77-81, 3-5 Dec. 2010

McLain, T. W. and Beard, R. W. (2000). Trajectory planning for coordinated rendezvous of unmanned air vehicles. In Proc. of AIAA Guidance, Navigation and Control Conference, Denver, CO.

Nikolos I.K, Valavanis K.P, Tsourveloudis N.C, & Kostaras A.N. (2003). Evolutionary algorithm based offline/online path planner for UAV navigation. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions. 898- 912, doi: 10.1109/TSMCB. 2002.804370

Pang C. Chen. (1995). Adaptive Path Planing: Algorithm and Analysis. IEEE international conference on Robotics and automation.

Rathbun, D.; Kragelund, S.; Pongpunwattana,A.; Capozzi, B (2002).,.Digital avionics Systems Conference, vol.2, 8D2-1-8D2-12 doi: 10.1109/DASC.2002.1052946

Rathbun D. Kragelund S, Pongpunwattana A, & Capozzi,

B. (2002). An evolution based path planning algorithm for autonomous motion of a UAV through uncertain environments, Digital Avionics Systems Conference, 2002. Proceedings. The 21st. 8D2-1- 8D2-12 doi: 10.1109/DASC.2002.1052946

Sanders G, & Ray T. (2007). Optimal offline path planning of a fixed wing unmanned aerial vehicle (UAV) using an evolutionary algorithm. Evolutionary Computation.4410-4416, 25-28. doi: 10.1109/CEC. 2007.4425048

Si-Yao Fu, Li-Wei Han, Yu Tian, & Guo-Sheng Yang. (2012). Path planning for unmanned aerial vehicle based on genetic algorithm. Cognitive Informatics & Cognitive Computing. 2012 IEEE 11th International Conference. 140-144, 22-24 doi:10.1109/ICCI-CC. 2012.6311139

Trovato, K.I. 1996. A* Planning in Discrete Configuration Spaces of Autonomous Systems. PhD thesis, Amsterdam University, 1996.

김 용 호 (yzk0302@gmail.com)

2011 미국 Auburn University, Wireless Software Engineering, BSE
2013 미국 Auburn University, Software Engineering, MSwE
2013.09 ~ 현재 국방과학연구소 연구원

관심분야 : 국방시스템, 소프트웨어 공학, 실시간 임베디드 시스템, 모델링&시뮬레이션