Memetic technique for multiple unmanned aerial vehicles

Doan Van Khanh

2011

Doan, V. K. (2011). Memetic technique for multiple unmanned aerial vehicles. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/47560>

<https://doi.org/10.32657/10356/47560>

Memetic Technique for Multiple Unmanned Aerial Vehicles Tasks Allocation

Doan Van Khanh

School of Electrical & Electronic Engineering

Nanyang Technological University

Singapore

A thesis submitted to

Nanyang Technological University

in partial fulfillment of the requirement for the degree of

Master of Engineering

2011

Abstract

This thesis focuses on a memetic computation approach for area coverage involving multiple unmanned aerial vehicles (UAVs). In some real-life scenarios, such as a battlefield or an earthquake disaster area, access using manned vehicles may present severe challenges for practical reasons. In scanning a region, each UAV will typically follow a certain pattern of combing through a region. To our knowledge, there has been no attempt to quantitatively model the turning time cost for UAV scanning. Such quantitative modeling allows for reliable estimation of the scanning time for the different scanning strategies.

We describe the formulation of a turning time model for estimating the time cost for UAVs changing direction during flight. Based on this model, we explore two strategies of scanning as a basis for a memetic computing search; the raster and circular scanning strategies. For raster scanning, the UAV flies in parallel to one edge of a polygon while for circular scanning, the UAV flies in cyclical path defined by the boundary starting from the outer to the inner region or vice versa. The objective of the memetic approach is to minimize scanning time through polygon decomposition as a form of tasks division between UAVs and for each sub-polygon, a pattern of scanning is specified such that the overall turning time is reduced accordingly. Validation through flight tests conducted at Boeing's Vehicles Swarm Technology Laboratory (VSTL) showed performance consistent with the estimation model in the algorithm.

Acknowledgement

No words of gratitude are enough to thank my supervisor Associate Professor Lim Meng Hiot (School of Electrical Electronic Engineering - Nanyang Technological University, Singapore), whose untiring support, motivation and supervision made this thesis. He always gives me timely and constructive feedback and thoughtful suggestions for my research ideas, paper and thesis drafts. I am particularly thankful for his patience and encouragement. His guidance has been indispensable in my academic, professional, and personal development. I feel lucky and honored to work with him.

I am indebted to Matthew Vavrina and Jang Jung Soon, the project team members from Boeing Research & Technology for their help during the time I work on the project. Without their help, the project could not finish successfully. I am also indebted to Dr. John Vian and Dr. James Barhorst, managers at Boeing Research & Technology for their financial support and guidance.

I also like to thank my friends especially to Chen Xianshun who is currently PhD candidate at Nanyang Technological University who gave me a lot of ideas and guided during my time at Intelligent System Centre. Thanks to Nora who had been helpful, especially during the time when I need to travel to conferences and to Boeing Research & Technology in Seattle.

Thanks to the School of Electrical Electronic Engineering and Research Techno plaza, all its administrative staff and IT support group who provided me with all the required facilities and took care of my needs during my stay at the school.

I express my gratitude to all of my teachers who showed me divine light of knowledge. I would like to express gratitude to all my fellows in the campus. Particularly, I would like to thank Mr. Xuan Anh, Mr. Minh Chau and Ms. Ee May for sharing their time and extending their friendship to me.

Finally I would like to thank my parents and my sisters for their unwavering support, guidance, encouragement, and emphasis on the value of education that allowed me to get where I am today. I am forever grateful to them.

Dedication

To My Family

List of Publications

Conference Articles

1. V.K. Doan, M.H. Lim, Y.S. Ong, J. S. Jang, M. A. Vavrina, J. Vian and J. Barhorst, “Turning Time Estimation Model for Region Coverage”, International Conference on Computational Problem Solving, Lijiang, China, 2010 (Best Student Paper Award)
2. P. Pandey, S. Hemant, and V.K. Doan, “An Agent Based Trading Game for Risk Adversity Level Estimation”, Proc. of the International Conference on Soft Computing and Pattern Recognition, Malaysia, 2009
3. E. M Kan., V.K. Doan and S. P. Yeo “Contour Based Path Planning for Unmanned Aerial Vehicles (UAVs) over Hostile Terrain” , Proc. of the International Conference on Soft Computing and Pattern Recognition, Malaysia, 2009

Contents

Abstract	2
Acknowledgement.....	3
List of Publications.....	5
List of Figures.....	8
List of Tables	10
Chapter 1 Introduction	11
1.1 Unmanned Aerial Vehicles Introduction	11
1.2 Applications of UAVs	12
1.3 Types of UAVs.....	13
1.4 Report Organization	14
Chapter 2 Polygon Decomposition.....	15
2.1 Quad-tree Approach.....	16
2.2 Divide-and-Conquer Sweep Line Algorithm	19
2.3 Random Sweep Line Algorithm	22
Chapter 3 Turning Time Model	24
3.1 Turning Time Model Formulation.....	24
3.2 Validation Based on Boeing's Model of UAV Dynamics	25
Chapter 4 Scanning Strategies.....	29
4.1 Scanning Triangles	29
4.2 Scanning Rectangles	30
4.3 Scanning Random Polygons.....	31
Chapter 5 Memetic Approach to Optimize Scanning Time.....	34
5.1 Representation and Decoding	36
5.2 Crossover and Mutation Operators.....	37
5.3 Local Search	39
5.4 ADEP Program.....	40
Chapter 6 Simulation.....	41
6.1 Simulation Using UAV Model	41
6.2 Effect of Scanning Strategies on Total Scanning Time.....	45

6.3	Integration with VSTL and Testing with real UAVs	49
Chapter 7	Re-Planning.....	52
7.1	First Approach to Re-planning	52
7.2	A Feasible Re-planning Approach	55
7.3	Interface Script	56
Chapter 8	Conclusions and Future Research Directions	59
8.1	Summary and Conclusions.....	59
8.2	Future Research Directions	59
References		61
Appendix A	- Simulator Screen Shots	63
Appendix B	- Syntax to Interact with MMF	65
Appendix C	- Swarm View Command	68

List of Figures

Figure 1: The Predator drone UAV developed by General Atomics GNAT.....	12
Figure 2: A quad-rotor UAV at Boeing Research & Technology	14
Figure 3: A polygonal plane and its bases.	15
Figure 4: An arbitrarily-shaped polygon region	17
Figure 5: Apply quad-tree algorithm for the given region.	17
Figure 6: Illustration of quad-tree for task allocation problem.....	18
Figure 7: Using quad-tree method to divide tasks to UAVs. We can see blue region assigned to one UAV spread to two different locations which are not contiguous to each other.	19
Figure 8: Polygon division using divide-and-conquer, sweep line algorithm	20
Figure 9: Divide-and-conquer, sweep line algorithm	20
Figure 10: A robot during turning	24
Figure 11: Validating the turning time model	25
Figure 12: Turning time curve for quad-rotor UAV.....	27
Figure 13: UAV flying a circular trajectory.	28
Figure 14: Scanning a triangle	29
Figure 15: Spiral pattern	29
Figure 16: Scanning a rectangle	30
Figure 17: A polygonal planar region for scanning.....	31
Figure 18: Determining reference edge for raster scanning	31
Figure 19: Circular scanning	32
Figure 20: Estimation of number of flight cycles.....	32
Figure 21: Memetic Technique.....	34
Figure 22: Task allocation for UAVs	36
Figure 23: Chromosome Decoding.....	37
Figure 24: An illustration of a 2-Opt swap operation.....	39
Figure 25: ADEP interface	40
Figure 26: Polygon decomposition.....	41
Figure 27: Assigned regions for all the UAVs	42
Figure 28: Plot of fitness versus generations.....	42
Figure 29: Optimization of scanning time.....	43
Figure 30: Testing with actual UAVs in VSTL	44
Figure 31: The performance of the algorithm observe in SwarmView.....	44
Figure 32: 3D visualization of the tasks allocation algorithm.....	45
Figure 33: Polygon decomposition.....	45
Figure 34: Assigned regions to UAVs.....	46
Figure 35: Fitness versus generations using both scanning method	47
Figure 36: Fitness versus generations using raster scanning method.....	47
Figure 37: Fitness versus generations using circular scanning method	47
Figure 38: Compare results between three cases	48
Figure 39: Fitness versus generations using random scanning method	48

Figure 40: Integration of tasks allocation algorithm with Mission Management Framework	49
Figure 41: Testing the tasks allocation algorithm with 6 quad-rotor UAVs	49
Figure 42: Actual flight paths of UAVs during mission execution.	50
Figure 43: Separation distance among the vehicles and number of active pairs on the local conflict resolution.	51
Figure 44: Some failure scenarios and proposed solutions	53
Figure 45: System integration involving Mission Management Framework, VSTL and NTU app showing the protocol for data communication.	54
Figure 46: System integration showing data communication protocol for re-planning	55
Figure 47: Tasks allocation program interface	63
Figure 48: Mission Management Framework.....	63
Figure 49: Swarm View is used to display information and send command to UAVs	64
Figure 50: Interactive Bezier curve	70
Figure 51: Bezier curve setting chart.....	70

List of Tables

Table 1: Data acquired from simulation	26
Table 2: Turning time information extracted from the experiment. The last column shows the estimated time using the values of a and b found using the least square error regression method. 26	
Table 3: Data extracted from circular trajectory simulation of Fig. 13.	28
Table 4: Crossover with two crossover sites	38
Table 5: One example of mutation operator	38
Table 6: Information of base stations in Fig. 26	41
Table 7: Information of base stations in Fig. 33	46

Chapter 1 Introduction

This report focuses on the computational techniques for missions involving unmanned vehicles. The recent disaster involving damaged nuclear reactors in Fukushima as a result of a 9.2 magnitude on the Richter scale highlights the necessity for unmanned missions. This is because of the high-level of radioactivity making it close to impossible for operations without endangering human lives. Japan has created a 20-kilometre no-go zone around the plant, where workers were dousing reactors and fuel rod pools with water to prevent overheating and melt-down [23]. The Obama administration also sent a squad of robots to Japan to help efforts to regain control over the Fukushima nuclear plant [24]. From this issue, we can see the importance of using unmanned vehicles to perform some tasks in dangerous environment. This project sponsored by Boeing Research & Technology attempts to explore features of unmanned aerial vehicles in performing region coverage through premeditated flight patterns. Such flight maneuvers are useful for certain missions, such as search and rescue, insecticide spraying, etc.

1.1 Unmanned Aerial Vehicles Introduction

Both helicopters and spy planes have a serious weakness; they must be piloted by human beings - that is to say, human beings must be on board to operate them. They cannot be lost with impunity and hence should not be deployed in situations that endanger the life of the pilots, as in the case of toxic environments. One must avoid both the purely negative consequences of crashes (death and property destruction), but also the consequences that are positive for one's enemies, such as the taking of prisoners of war, hostages and other potential sources of sensitive information. Therefore, unmanned aerial vehicles (UAVs) have been developed during the past few decades to perform tasks which may not be appropriately achieved using manned aerial planes.

UAVs can use wireless, radio technology or system of satellites to communicate with their ground stations, making them flexible and potentially useful in surveillance, search and rescue tasks or even used for tracking and combat. UAVs equipped with good image sensors can provide precise and real-time imagery to a ground control operator, who would then disseminate that information to facilitate decision-making. Additionally, some new UAVs such as the Predator (Figure 1) used in the recent wars in Afghanistan and Iraq can fly for more than 30 hours without refueling, compared with a helicopter's average flight time of just over a few hours [7]. The ability of UAVs to loiter for prolonged period of time has important operational advantages over manned aircrafts. The longer flight time of UAVs means that they can be used to handle difficult tasks like search and rescue missions in the open sea which normally require prolonged time of operation.

UAVs can be equipped with thermal detection sensors to detect humans hiding under a tree or a house; further extending the UAV's usefulness for different application scenarios.

Figure 1: The Predator drone UAV developed by General Atomics GNAT¹

1.2 Applications of UAVs

Unmanned Aerial Vehicles have been developed for a few decades. However they have found many uses in police, military, and in some cases, civilians' applications. Currently, UAVs are most often used for the following tasks:

- Aerial Reconnaissance – UAVs can be equipped with high resolution still video and even infrared cameras which can capture video of a remote location, especially in situations whereby the level of risk to the pilot of a manned aircraft is deemed unacceptable. Information obtained by the UAV can be streamed back to the control center in real time.
- Scientific Research – In many cases, scientific research necessitates obtaining data from hazardous or remote locations. A good example is hurricane research, which often involves sending a large manned aircraft into the center of the storm to obtain meteorological data. It would be desirable if an UAV can be used to gather data, with no risk to a human pilot.
- Logistics and Transportation – UAVs can be used to carry and deliver a variety of payloads. Helicopter types UAVs are well suited for this purpose, because payloads can be suspended from the bottom of the airframe, with minimal aerodynamic penalty.

¹ <http://science.howstuffworks.com/predator.htm>

1.3 Types of UAVs

There are many different types of UAVs, designed for different purposes. Based on the size and flying endurance, we can divide UAVs into several groups as follow:

- Micro UAVs – small, extremely portable units;
- Low altitude, long endurance UAVs;
- High altitude, long endurance UAVs employing a conventional design;
- High altitude, long endurance UAVs using a low observable design.

Some UAVs use a blimp design, and are well-suited for carrying large cargo. In the early days of UAVs, they were called drones as they are not autonomous, because they required constant control input from a remote human pilot. Computer technology now allows UAVs to make their own decisions, or fly autonomously.

Generally, autonomous flight involves the following operations:

- Interpreting sensor inputs, and merging the inputs of multiple sensors;
- Communicating with ground stations, satellites, and other UAVs and aircrafts;
- Determining the ideal course to fly for a given mission, based on sensors' input;
- Determining the best maneuvers to perform for a given task;
- In some cases, cooperating with other UAVs to accomplish a common task.

In this project, we use quad-rotor UAVs, type of aircrafts whose lift is generated by four rotors. Control of such a craft is accomplished by varying the speed of the four motors relative to each other. Quad-rotor crafts naturally demand a sophisticated control system in order to allow for a balanced flight. Uncontrolled flight of a quad-rotor would be virtually impossible by an operator, as the dynamics of such a system demand constant adjustment of the four motors simultaneously. Fig. 2 is a photograph showing the basic structure of a quad-rotor UAV at Boeing's research laboratory.

Figure 2: A quad-rotor UAV at Boeing Research & Technology

1.4 Report Organization

The remainder of the thesis is structured as follow. Chapter 2 describes algorithms to divide a polygon to multiple pieces; the number of sub-regions is according to the number of base stations defined for the problem. Chapter 3 describes the turning time model to estimate the turning time which depends on the turning angle and turning radius. Chapter 4 compares two scanning methods; raster scanning and circular scanning. Chapter 5 introduces the memetic approach for optimizing the total scanning time of all UAVs in order to minimize the time of scanning the whole region. Some simulation results are presented. Chapter 6 describes the integration of the algorithm into the system at Vehicles Swarm Technology Laboratory (VSTL) and experiment with real UAVs. In real-life scenario, there is always the possibility of one or several UAVs failing during the mission. Therefore, the requirement for re-planning becomes important and the solution is described in Chapter 7. Finally, the last chapter concludes this thesis.

Chapter 2 Polygon Decomposition

The main objective of this work is to study the tasks allocation problem associated with UAVs for region coverage. We consider the problem which involves a number of UAVs to scan a region represented as a convex polygon, an example being shown in Fig. 3. In the figure, S_1 to S_4 represent the base locations where the UAVs are stationed prior to being deployed in a mission.

Figure 3: A polygonal plane and its bases.

Each base contains a number of UAVs. In this project, we assume that there are several types of UAVs (with differing capabilities) to be used for the scanning tasks. The capability C of a UAV is the scanning capacity in terms of area it can cover and is calculated as the product of speed, flying endurance and footprint:

$$C = \text{footprint} * \text{speed} * \text{endurance}$$

The first step involves dividing the polygon into a number of pieces such that the coverage area assigned to each base is proportional to the base's overall capability. A base's capability is the sum of all the individual UAVs' capabilities located at this base. After assigning each sub-region to one base, a memetic technique is used to configure a working region and the best scanning method for each UAV in order to minimize the overall scanning time. Chapter 3 will introduce turning time model which shows that different scanning methods result in different scanning cost, essentially the time of turning. As a general rule, the method which requires the minimum number of turns normally results in smallest scanning time and is generally preferred.

Consider a given two-dimensional planar polygonal surface which corresponds to the area or region of interest. The objective of polygon decomposition is to divide the given polygon p into m polygonal pieces. In the context of tasks allocation, m refers to the number of base stations where

the UAVs are located. A decomposed polygon p will result in m polygons; the area of each piece is according to the pre-specified proportion of overall UAVs capabilities at each of the base stations. Let's assume we have one polygon with n vertices to be scanned using unmanned aerial vehicles located at different bases on the polygon's boundary (Figure 3). The team of UAVs has a mission to fly and scan the whole region, and to do it effectively, the polygon p should be divided among the bases according to their relative capabilities. With each assigned area, each UAV will execute its scanning strategy to comb the region. This will be discussed with greater details in Chapter 4.

In our work, we consider only convex polygons with no holes. Assume that base station i possesses capability denoted as C_i which is the sum of the capabilities of all UAVs situated in this base and there are m bases on the polygon. Our approach is to divide the workspace into m parts, and assign each part to one base. Let's assume that the area of the part P_i is $Area(P_i)$. A solution is considered to be feasible if the following equation is satisfied:

$$Area(P_1): Area(P_2): \dots : Area(P_m) = C_1: C_2: \dots : C_m \quad (1)$$

Furthermore, the total area of all the parts should equal the area of the given polygon:

$$\sum_{i=1}^m Area(P_i) = Area(p)$$

We first present the quad-tree method to divide tasks for multiple UAVs followed by a divide and conquer sweep line algorithm.

2.1 Quad-tree Approach

A quad-tree is a tree data structure in which each internal node having exactly four leaf nodes. Quad-trees are often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular, or may have arbitrary shapes. All forms of quad-trees share some common features listed as follow:

- They decompose space into adaptable cells;
- Each cell has a maximum capacity. When maximum capacity is reached, the bucket splits;
- The tree directory follows the spatial decomposition of the quad-tree.

Consider a region represented as a polygon as shown in Fig. 4.

Figure 4: An arbitrarily-shaped polygon region

We can construct a rectangle which bounds the region inside it. Applying the quad-tree algorithm to this rectangle, we achieve a decomposed rectangle as shown in Fig. 5.

Figure 5: Apply quad-tree algorithm for the given region.

The shaded rectangular cells are those located inside the region and the unshaded ones are located outside the region. The construction of quad-tree follows the steps below:

1. The algorithm first subdivides the region AABB into four rectangular quadrants (regions of equal size).
2. For each region, check if it is empty. A region is considered empty if the area overlapping between this region and the interested region is lower than the minimum coverage threshold. A node that contains the empty region is considered as a leaf node which is not highlighted in Fig. 5. A non-empty leaf node is then highlighted as a shaded region.
3. For each region, check if it is fully occupied. A region is considered fully occupied if the area overlapping between this region and the region of interest is higher than the threshold for full coverage. A node that is fully occupied region is considered a leaf node and is highlighted in Figure 5.

4. For each region, if it is not empty and not fully occupied, the algorithm will recursively subdivide this region by repeating steps 1-4.

The algorithm stops when a region reaches a minimum size limit or the quad-tree reaches certain number of levels.

The method using quad-tree for allocating tasks for UAVs can only achieve one requirement, that is all UAVs will be assigned to cells but cannot satisfy the second requirement of contiguity. This is because when a cell is assigned to one UAV, and the capacity of this UAV reaches zero, it will be assigned to another UAV. Therefore, one UAV may be assigned to different cells located on different quadrants, making the cells assigned to be non-contiguous.

Figure 6: Illustration of quad-tree for task allocation problem

For example, consider the process of assigning cells to a UAV T . Here, we consider a searching region bounded by a rectangle. The quad-tree algorithm will be applied to the bounded rectangle and first divide it into four equal small rectangles named as top-left (TL), top-right (TR), bottom-left (BL) and bottom-right (BR) as shown in Fig.6. When the algorithm is applied to cell X as shown in Fig. 6 above, UAV T is assigned to X . Then, the algorithm moves to TR branch, UAV T which has capacity above zero is still used to assign to cells in the TR branch. However, cells in TR may not be adjacent with cell X ; hence not able to achieve the contiguity requirement. Figure 7 further illustrates the issue of contiguity as the light blue regions are divided into two different quadrants which are not adjacent.

Figure 7: Using quad-tree method to divide tasks to UAVs. We can see blue region assigned to one UAV spread to two different locations which are not contiguous to each other.

Because of this limitation of quad-tree, we introduce another method to divide polygons to UAVs based on a divide-and-conquer sweep line algorithm.

2.2 Divide-and-Conquer Sweep Line Algorithm

We first describe the notations used in the algorithm. Consider a polygon p with n number of vertices from V_1 to V_n , and m bases on its boundary from S_1 to S_m . We need to divide the polygon into two small polygons first before further dividing each of the sub- polygon recursively.

Each polygon p has three data lists $V(p)$, $S(p)$, and $W(p)$

- $V(p)$ - the vertices of the polygon in counter clockwise order;
- $S(p)$ - the bases of the polygon in counter clockwise order;
- $W(p)$ - the vertices and bases of the polygon in counter clockwise order.

This problem is studied in [1] with divide-and-conquer, sweep line algorithm. The main idea of the algorithm is to use line L with two end points L_s and L_e to cut the polygon into two pieces. The algorithm tries to adapt L_s and L_e to form two polygons which satisfy the area constraint.

Figure 8: Polygon division using divide-and-conquer, sweep line algorithm

Let L denote the sweep line L_s to L_e such that L_s and L_e are both on the polygon's boundary. Let $poly1$ be the polygon on the right of L and $poly2$ to be the polygon on the left of L . In the above Figure, $poly1$ has $V(poly1) = \{L_s, V_1, V_2, V_3, L_e\}$, $S(poly1) = \{S_1, S_2\}$ and $poly2$ has $V(poly2) = \{L_e, V_4, V_5, L_s\}$ and $S(poly2) = \{S_3, S_4\}$.

A polygon p with its state of area completeness is categorized as follow:

- Area-complete if area required by $S(p)$ equals the polygon's area;
- Area-incomplete if area required by $S(p)$ is greater than the polygon's area;
- Base-incomplete if area required by $S(p)$ is less than polygon's area.

Figure 9: Divide-and-conquer, sweep line algorithm

Pseudo code for divide-and-conquer, sweep line algorithm

1. **Input:** convex polygon p with $W(p) = \{w_1, w_2, \dots, w_m\}$ all are counter clockwise.
(For the above polygon shown in Fig. 9, $S(p) = \{S_1, S_2, S_3\}$, $S_1 = w_3, S_2 = w_5, S_3 = w_7$)
2. **Output:** two well-divided regions
3. **Implementation** - Convex_Polygon_Decomposition
 - a. Form a line $L = (L_s, L_e) = (w_1, w_3)$
 - b. $Poly1 \leftarrow S_1$ we assume that $Poly1$ or a polygon of (w_1, w_2, w_3) contain S_1
 - c. While $Area(Poly1) < AreaRequired(S(Poly1))$ and $L_e \neq S_n$ do
 *keep running this loop when this condition still happen
 * $AreaRequired(S(Poly1))$ means the sum of all capabilities of sites inside the polygon $Poly1$, and S_n here means the last site in the sequence which is S_3
 1. Move L_e to the next point, for example from w_3 to w_4
 2. If w_4 is a site then add it to our list $S(Poly1)$, $Poly1$ now includes a new site.
 - d. After the above loop, there are three possibilities:
 1. If $L_e = S_1$, and $Area(Poly1) > AreaRequired(S(Poly1))$, then we can move L_s counter-clockwise so that $Area(Poly1) = AreaRequired(S(Poly1))$
 2. If $L_e = S_n$, it means S_3 in Fig. 9, and $Area(Poly1) < AreaRequired(S(Poly1))$ then we move L_s clockwise, to $w_{11} \rightarrow w_{10} \dots$ For each movement of L_s , we have a new $Area(Poly1)$ and we keep moving until $Area(Poly1) = AreaRequired(S(Poly1))$
 3. L_e has moved through some S_i . For example L_e has passed by S_2 and at $L_e = w_6$ we have $Area(Poly1)$ greater than $AreaRequired(S(Poly1))$
 However, at $L_e = w_5$ we still have $Area(Poly1) < AreaRequired(S(Poly1))$.
 It means that the desired point L_e should belong to the line (w_5, w_6) . We do interpolation to find a desired point on that line which gives
 $Area(Poly1) = AreaRequired(S(Poly1))$

The polygon has m bases, therefore the procedure should be called exactly $m-1$ times to partition a convex, m -base area-complete polygon into m convex, 1-base area-complete polygons. The computation of the full partition of the polygon with n vertices and m bases has a $O(m-1)(m+n)$ complexity in the worst case[1]. One drawback of the algorithm is that it gives only one solution for the given polygon. In fact, there can be many solutions which satisfy the requirements of the problem. To achieve this, we present a new algorithm called “random sweep line (RSL) algorithm” to divide a convex polygon.

The main idea of this algorithm is to divide from the group of bases $SG = (S_1, S_2, \dots, S_m)$ into two sub-groups SG_1, SG_2 such that the area covered by each group is the required area of that group.

2.3 Random Sweep Line Algorithm

Consider the polygon p with $S(p) = (S_1, S_2, \dots, S_m)$. The algorithm divides $S(p)$ into two sub-groups $S(poly1)$ and $S(poly2)$ such that their capabilities correspond to the areas of $poly1$ and $poly2$ respectively. The random sweep line procedure is described by the pseudo-codes below.

Random sweep line algorithm

Input – convex polygon p with n vertices

The set of bases $S(p) = \{S_1, S_2, \dots, S_m\}$ in counter-clockwise (CCW) order;

– Divide each edge into Max number of points.

Output - Two polygons $Poly1, Poly2$ and their bases.

Procedure - ConvexPolygonDivide

1. Divide each edge to Max points and form new polygon $polyMax$;
2. CalculateIndexOfBase() //function used to find index number of each base in $polyMax$;
3. L_s = random number between S_1 index and S_2 index;
4. $Poly1.vertices[] \leftarrow$ vertices of p from V_1 up to vertices with index less than L_s/Max , and point at L_s ;
5. $Poly1.base[] \leftarrow S_1$.

If ($Poly1.area < areaRequired$ by S_1)

$Half = \max(0.5 * Max * n, \text{index of } S_2)$;

L_e = random between $Half$ and $Max * n$;

$Poly1.vertices[] \leftarrow$ vertices of p from V_1 up to vertices with index less than L_s/Max , points at L_s, L_e , and vertices of p from L_e/Max up to V_n ;

$Poly1.base[] \leftarrow$ all bases located between 0 to L_s and L_e to $Max * n$;

If ($Poly1.area < areaRequired$) move L_e clockwise (CW) until L_e corresponds to a base in p ;

If condition still not satisfied, move L_s CCW until $Poly1.area = areaRequired$;

If ($Poly1.area > areaRequired$) move L_e CCW until L_e correspond to a base in p ;

If condition still not satisfied, move L_s CW until $Poly1.area = areaRequired$;

$Poly2$ is constructed here.

If ($Poly1.area > areaRequired$ by S_1)

Let $L_e = L_s$; let $L_s = 0$;

$Poly1$ is re-constructed here;

If ($Poly1.area > areaRequired$) move L_s CCW to reduce area of $Poly1$ until $L_s \equiv S_1$;

If still not satisfied move L_e CW to reduce area of $Poly1$;

$Poly2$ is constructed here.

The complexity of RSL algorithm is $O(Max*m*n)$. Both algorithms are fast with reasonable computation time. However, the RSL algorithm presented here can give us many solutions for a given search region, making it more flexible and useful for some applications.

Chapter 3 Turning Time Model

Here, we present the model to estimate the time cost of a UAV making a turn. We study the case by considering a robot making a turn in order to formulate the model. This model is also applicable to UAV in which frictional force can be replaced by force generated by the UAV's rotor.

3.1 Turning Time Model Formulation

Consider the situation whereby a robot has to make a turn with turning angle α . Let v be the velocity which is considered to be constant during a turn. During a turn, frictional force must be equal to the centrifugal force in order to keep the robot turning without sliding:

$$mv^2/R = kmg \quad \text{or} \quad ma_n = kmg$$

Therefore, $a_n = kg = \text{const}$ (if there is no sliding). Here, m is the robot's mass,

- a_n is centripetal acceleration, k is a frictional coefficient and g is a gravitational acceleration

Figure 10: A robot during turning

For a small incremental time dt , the robot moves a distance $ds = vdt = R d\theta$ where $R = v^2/a_n$. This implies that:

$$dt = \frac{v}{a_n} d\theta$$

Therefore, the total time taken to make a turn of angle α is:

$$\int_0^t dt = \int_0^\alpha \frac{v}{a_n} d\theta \quad (1)$$

Let v_0 be the speed of a robot on a straight line. To make a turn, it has to reduce the speed to $v = v_0 - h\alpha$. Here, h (unit m/s) is a constant which can be measured through experimentation.

In order to effect a turning without sliding, $v \leq \sqrt{kgR}$. If $v_0 \leq \sqrt{kgR}$, the robot can turn safely without reducing its speed. However, in real life, this condition is usually not applicable. The larger the turning radius, the smaller is the value of h and the larger is the speed achievable during turning without sliding. From Eq. 1, we have:

$$t_{turning} = \int_0^\alpha \frac{(v_0 - h\alpha)}{a_n} d\theta = \frac{(v_0 - h\alpha) * \alpha}{a_n} \quad (2)$$

So, if the turning angle is α , then the time it takes to make a turn can be represented by the following equation:

$$t_{turning} = a * \alpha - b * \alpha^2 \quad (3)$$

In Eq. 3, a and b are two constants which depend on the physical characteristics of the turning condition.

3.2 Validation Based on Boeing's Model of UAV Dynamics

We use the UAV model developed by Boeing to test and validate the correctness of the above turn cost model. We simulated a UAV flying in a curved trajectory in Swarm View (Boeing's vehicle simulation platform) as shown in Fig. 11. For more information on how to execute commands in Swarm View to control UAVs flight, see Appendix C.

Figure 11: Validating the turning time model

Based on the simulation, the acquired information about the flight and trajectory are given in the Table 1. The timing information is based on the average of five measurements so as to reduce the error in estimating the flight time.

Table 1: Data acquired from simulation

	Length (cm)	Time (s)
From A to B	AB=12.2 cm	$t_{AB}=14.5$ s
From B to C	Arc BC=15.5 cm	$t_{BC}=21.6$ s
From C to K	Arc CK= 5 cm	$t_{CK}=4.9$ s
From H to D	Arc HD=2 cm	$t_{HD}=3.3$ s
From H to T	Arc HT=2.2 cm	$t_{HT}=6.4$ s
From T to E	Arc TE=3.3 cm	$t_{TE}=4.7$ s

Turning angle from C to K is 105° or 1.8326 radians, from H to D is 63° or 1.1 radians, from H to T is 163° or 2.845 radians, and from T to E is 98° or 1.71 radians. From the experiment, we can calculate average velocity on straight line AB and on curve BC

$$v_{AB} = 12.2/14.52 = 0.8402 \text{ cm/s}$$

$$v_{BC} = 15.5/21.63 = 0.7166 \text{ cm/s}$$

Clearly, the velocity of the UAV along a curve is less than the velocity on a straight line path. This is consistent with the basic assumption specified by $v = v_0 - h\alpha$. The simulation shows that from C to K, the UAV cannot follow the path of the arc. It changed abruptly from C to a point before K, stopping for a while and then moving forward to K.

From Equation 3, we have $t_{\text{turning}} = a\alpha - b\alpha^2$ for $\alpha \leq \pi$. In order to estimate a and b , we apply the least square error regression approximation.

Table 2: Turning time information extracted from the experiment. The last column shows the estimated time using the values of a and b found using the least square error regression method.

Angle (x)	Measured time (y)	Estimated time by equation
1.8326	4.9	4.92
1.1	3.3	3.30
2.845	6.4	6.4
1.71	4.7	4.69

We need to minimize the error:

$$R = \sum (y - ax + bx^2)^2 \quad (4)$$

Using the least square regression approximation method, we have: $\delta R / \delta a = 0$ and $\delta R / \delta b = 0$

Taking the derivative of equation (4) with the respective values of a and b , we have

$$\begin{cases} \sum xy - a \sum x^2 + b \sum x^3 = 0 \\ \sum x^2 y - a \sum x^3 + b \sum x^4 = 0 \end{cases}$$

By substituting the value of x and y from Table 2 into the above equations, we have:

$$\begin{cases} -15.587a + 35.51b = -38.91 \\ -35.51a + 86.807b = -86.1 \end{cases}$$

Solving the above equations, we have $a = 3.477$ and $b = 0.4305$. Therefore, the turning time cost of the UAV can be approximated using: $t = 3.477\alpha - 0.4305\alpha^2$

With t measured in seconds and α (turning angle) in radians, Fig. 12 below shows the turning curve with these values of a and b .

Figure 12: Turning time curve for quad-rotor UAV

Comparing with the measured values, the calculated values of turning time are very close to the values measured. This shows that the second order polynomial equation gives a very close approximation of the turning time cost incurred by the UAV.

We consider another simulation scenario as shown in Fig. 13 to validate the turning time model.

Figure 13: UAV flying a circular trajectory.

Table 3: Data extracted from circular trajectory simulation of Fig. 13.

	Length(cm)	Time (s)	Average velocity (cm/s)
From A to B	AB=12.2	$t_{AB}=14.1$	0.865
From B to C	BC=10.5	$t_{BC}=12.2$	0.861
From D to E	Arc DE =11.7	$t_{DE}=16.28$	0.719
From E to F	Arc EF=5.8	$t_{EF}=7.91$	0.733
From F to G	Arc FG=5.8	$t_{FG}=7.84$	0.74
From G to H	Arc GH=5.6	$t_{GH}=7.75$	0.72
From H to K	Arc HK=5.5	$t_{HK}=7.3$	0.75

Based on the data extracted as tabulated in Table 3, we can calculate the average velocity of the UAV on different segments of the flight path as shown in the last column. From the experiments, we can conclude that velocity of UAV during turning is less than velocity on a straight line. The velocity on a curve depends on the turning radius. If turning radius is too small for the UAV, it cannot follow the correct path, so it has to fly abruptly or be blown out of the course.

Chapter 4 Scanning Strategies

We compare two methods of scanning, raster and circular scanning to study the difference in terms of the number of turns it takes and total time-cost of scanning. The scanning time combines the time a UAV takes to execute the turns and total time on straight paths. For a UAV to make a turn, it first reduces its speed, executes the turning and then increases its speed to the desired level. Additionally, during turning, a UAV consumes more fuel compared to flying on a straight path. Therefore, the number of turnings can be an important factor in choosing the scanning pattern. We first analyze the case of triangles, followed by rectangles and then random polygons.

4.1 Scanning Triangles

First, we consider the case where the polygon is a triangle. Assume that a , b and c are the length of edges opposite to vertex A , B and C respectively and h_a , h_b and h_c are the corresponding heights of the triangle, we have $a \cdot h_a = b \cdot h_b = c \cdot h_c = 2 \cdot \text{area of the triangle} = 2S$ (let S is the area of the triangle). If $a > b > c$, then it implies that $h_a < h_b < h_c$.

If a UAV does a raster scan in the direction parallel to edge a , the number of turns required is

$N_a = h_a / \Delta = 2S / a\Delta$. (Here, Δ denotes footprint of an image sensor)

Figure 14: Scanning a triangle

Let " t_o " be the time the UAV takes to turn 180° and v is its velocity. The total time it takes to scan the whole region is:

$$T_a = 2St_o / a\Delta + S/v\Delta \quad (1)$$

If the UAV scans in the direction parallel to edge b or c , the number of turns required is h_b / Δ or h_c / Δ respectively. Since $h_a < h_b < h_c$, this implies that the number of turns on the direction parallel to the edge a is minimum.

If the UAV scans in a spiral pattern, the total number of turns will be $N_s = 3r / \Delta$, r being the radius of the circle within the triangle (see Fig. 15).

We can deduce that:

$$S = (a+b+c) r / 2 \Rightarrow r = 2S / (a+b+c)$$

If the triangle is not an equilateral triangle, then the turning angles are not the same. Let the average time a UAV takes to make a turn be Ωt_o . Therefore, the total time a UAV takes to scan the given region is:

$$T_s = \Omega 6St_o / (a+b+c)\Delta + S/v\Delta \quad (2)$$

Figure 15: Spiral pattern

Here the turning angle is less than 180° , hence $0.5 < \Omega < 1$

- If $a > b > c \Rightarrow a + b + c < 3a$
 $\Rightarrow T_s > \Omega 2St_o/a\Delta + S/v\Delta$
- If $a = b = c \Rightarrow T_s = \Omega 2St_o/a\Delta + S/v\Delta < T_a$

From the above analysis, we can deduce that it takes less time to scan in a spiral pattern than a raster scan in a direction parallel to an edge of the triangle.

For the case of a triangular region, the best direction of scanning is to scan in parallel to an edge corresponding to the longest edge of the triangle or to scan in a spiral pattern. In general, it is not possible to ascertain which method is better if we do not have *a priori* knowledge about the value Ω .

4.2 Scanning Rectangles

If the UAV scans in the direction parallel to edge “a”, the number of turns required is b/Δ .

If the UAV scans in the direction parallel to edge “b”, the number of turns required is a/Δ . We can therefore deduce total scanning time as follow:

$$T_a = bt_o/\Delta + S/v\Delta < T_b = at_o/\Delta + S/v\Delta \quad (\text{If } a > b)$$

If the UAV scans in a spiral pattern, the total number of turns will be $2b/\Delta$. Here, the turning angles are 90° compared to the above strategy where the turning angles are 180° . Hence, the time required to scan the rectangle is

$$T_s = 2b \Omega t_o/\Delta + S/v\Delta$$

Depending on the value of Ω , T_s may be larger or smaller than T_a . If $\Omega < 0.5$ (it means turning 90° requires less than half the time compared to turning 180°) then $T_s < T_a$. If $\Omega > 0.5$ then $T_a < T_s$.

Figure 16: Scanning a rectangle

4.3 Scanning Random Polygons

Consider the case where a UAV scans a random polygon as shown in Fig. 17 below and assume that scanning is either raster or circular. Raster scanning strategy has also been studied in [3]. However in [3], they only considered the simple case of three UAVs located at different locations on a search region's boundary. In this thesis, we study the tasks allocation problem associated with UAVs combing a region of interest using multiple scanning patterns.

Figure 17: A polygonal planar region for scanning.

In raster scanning, the UAV will scan parallel to one edge of the polygon. As with the case of rectangles, it is appropriate to consider which edge of the polygon is to be referenced in order to reduce the number of turnings? We show that the total number of turnings is minimized if the UAV scans parallel to the edge with the smallest corresponding height h as illustrated in Fig. 18.

Figure 18: Determining reference edge for raster scanning

Here, the total number of turns is h/Δ with Δ denoting the image sensor width or footprint. In raster scan, all turning angles are 180° so the total turning cost is:

$$t_{raster} = h/\Delta * t_{turning_180} \quad (3)$$

If the UAV scans in spiral manner, we need to estimate the total number of cycles required to complete the coverage. We consider the polygon after the UAV scans one cycle from outer to inner as shown in Fig. 19.

Figure 19: Circular scanning

After one cycle, the dimension of the polygon's edges will reduce accordingly. For example, consider edge AB for illustration.

Figure 20: Estimation of number of flight cycles

Referring to Fig. 20, we have $IM=IN=\Delta$ where I is located on the distributed ray vision from angle B . Therefore,

$$IB = \Delta / \sin (B/2)$$

$$BM = IB \cos (B/2) = \Delta \cot (B/2) \quad (4)$$

Similarly, when turning at angle A , edge AB reduces by an amount of $\Delta \cot (A/2)$. Therefore, after one cycle, edge AB has an adjusted length of A_1B_1

$$A_1B_1 = AB - \Delta [\cot (A/2) + \cot (B/2)]$$

After n scanning cycles, its length reduces to:

$$A_nB_n = AB - n\Delta [\cot (A/2) + \cot (B/2)] \quad (5)$$

From Eq. 5, as n increases, there is a tendency that some of the shorter edges will approach zero faster than the longer edges. When this happens, the geometrical shape of the polygon will change accordingly. We can estimate the total number of cycles required to cover the entire region to be:

$$N_{\text{cycle}} = 2 * \text{Area} / (\text{Perimeter} * \Delta)$$

Therefore, the total turning time to cover the polygon is:

$$t_{\text{cycle}} = N_{\text{cycle}} * \sum t_{\text{turning at each corner}} \quad (6)$$

Comparing the values calculated from (3) and (6) will enable us to determine which scanning method is better for a given polygon.

In this chapter, the effect of different scanning methods on total turning time for a UAV to scan a given region is presented. Based on the UAV model used in our study, it takes less time to scan a triangular shaped region using circular scanning strategy than raster scanning method. For rectangular cases, the best scanning strategy is raster scanning and UAV has to fly parallel to the longer edge of the rectangle. For a large polygonal plane to be assigned to a number of UAVs, the allocation of sub-region of different geometrical shapes will result in different total scanning time for all the UAVs to accomplish the mission. As such, it is important to derive efficient tasks allocation solutions which also specify the appropriate scanning strategy for each UAV so that the overall scanning is minimized. Besides reducing the overall mission time, there are benefits in terms of cost-saving and flexibility in adapting to changes in the mission. To achieve the objective stated, we employ a memetic computing technique to derive solutions for efficient tasks allocation. In the next chapter, details on the memetic computational approach will be presented.

Chapter 5 Memetic Approach to Optimize Scanning Time

In the context of memetic computation, memetic algorithms (MAs) is one of the most widely used technique for optimization. They are evolutionary algorithms (EAs) that apply a separate local search process or methods to refine individuals. Under different contexts and situations, MAs are also known as hybrid EAs, genetic with local searchers. Combining global and local search is a strategy used by many successful optimization approaches, and MAs have in fact been recognized as a powerful algorithmic paradigm for evolutionary computing. The idea of the algorithm is representing solutions in term of chromosomes and applying operations that simulate selection, crossover and mutation to adapt the chromosomes. As the population of chromosomes evolves, the overall fitness improves, finally converging towards the solutions that are either optimal or close to optimal. For more details of memetic algorithm, readers can refer to [4, 5]. Memetic algorithms have been applied to optimization problems like logistic management, portfolio optimization [8], database optimization [9], etc. Here, the memetic approach is used to optimize total scanning time for a swarm of UAVs deployed to comb a region defined by a given polygonal plane. The general flow of a canonical memetic approach is as given in Fig. 21 [4].

Figure 21: Memetic Technique

Genetic algorithms are population-based metaheuristics. They have been successfully applied to many optimization problems. However, premature convergence is an inherent characteristic of such classical genetic algorithms, making them incapable of producing high quality solutions. A memetic algorithm is an extension of the traditional genetic algorithm. It uses a local search

technique to extend the search capability, hence enhancing the likelihood of delivering high quality solutions.

Various forms of genetic algorithms may exist in literature. However, the basic form of a GA follows a few standard steps and is as described below:

1. **[Start]** Initial population is generated by randomly generating many individuals representing potential solutions to the problem. These individuals are referred to as chromosomes.
2. **[Fitness]** Evaluate the fitness of each chromosome in the population.
3. **[New population]** New population is created based on the following steps:
 1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the higher the fitness, the higher the chance of being selected)
 2. **[Crossover]** Crossover is performed on the selected parents to create a new offspring (child). If no crossover is performed, offspring is an exact copy of parents.
 3. **[Mutation]** Offspring is being mutated (randomly changing the value of a gene in the chromosome) based on a certain preset mutation probability.
4. **[Replace]** Replace the current population of chromosomes with the newly generated population samples.
5. **[Test]** If the end condition is satisfied, the algorithm terminates and returns the best solution in the current population. Otherwise, return to step 2 and repeat.

Memetic algorithm adds in local search which is applied to individuals in the population at each generation in the recombination step of a genetic algorithm. This addition makes memetic algorithm able to refine individuals and the result is normally better than those obtained by a standard genetic algorithm [21].

For our problem, we start off with a given polygon. After dividing the given polygon to sub-regions according to the number of base stations, we further divide each region to a number of UAVs in each base as illustrated in Fig. 22.

Figure 22: Task allocation for UAVs

Figure 22 depicts three dotted lines which divide the polygon into four pieces meaning that the point where all the dotted lines converge has four UAVs. Each UAV has different capability and hence needs to be assigned with different area. The order of UAV assigned from left to right will affect the assigned shape to each UAV and therefore affects the scanning time of this UAV. The total scanning time is the sum of the time required by UAVs to complete its task. The objective is to use MA to minimize the total scanning time of all UAVs.

5.1 Representation and Decoding

In memetic algorithm, solution representation plays an important role and can directly affect the performance of the algorithm. We make use of an example to illustrate the chromosome representation using integer permutation. Let us assume that there are three base stations on the edges of the polygon. Base 1 has 5 UAVs, base 2 has 6 UAVs, and base 3 has 4 UAVs. The chromosome has a total of $5+6+4=15$ genes and each gene can take on values from 0 to 14. An example of a solution representing the plan for deployment of UAVs is as shown by the permutation string below. We illustrate the process of decoding based on this example.

1	5	0	9	10	14	2	13	8	11	4	12	6	3	7
---	---	---	---	----	----	---	----	---	----	---	----	---	---	---

If a gene has value less than 5, it belongs to base 1. For the above chromosome, the assigned order of UAVs in base 1 is:

1	0	2	4	3
---	---	---	---	---

The assigned order of UAVs in base 2 is:

5	9	10	8	6	7
---	---	----	---	---	---

The assigned order of UAVs in base 3 is:

14	13	11	12
----	----	----	----

Using this representation, a memetic algorithm is applied to search for a permutation which results in minimum total scanning time. Based on the sub-region assigned to a UAV, the cost is calculated by comparing the time incurred by scanning in raster and circular manner.

Assume that one base has three UAVs as shown in Fig. 23 and the order of the UAVs assigned is “2, 1, 3”. First, the algorithm checks the area of polygon $KABC$. If the area of polygon $KABC$ is greater than the area that should be allocated to UAV 2 based on the area constraint, we need to find point M on edge BC such that area of polygon $KABM$ of equal area should be assigned to UAV 2. The procedure is repeated for UAV 1 and UAV 3.

Figure 23: Chromosome Decoding

For each UAV with the assigned area, the algorithm will estimate the scanning time based on raster and circular scanning, and choose the scanning method which leads to smaller turning time. The objective is to minimize total turning time for all UAVs used in the mission.

5.2 Crossover and Mutation Operators

Crossover selects genes from parent chromosomes and creates a new offspring. The simplest way to do this is to choose randomly a crossover point and everything before this point copied from the first parent and then everything after the crossover point copied from the second parent. In this problem, we use 2-point crossover operator. The operation is illustrated in Table 4 (where “|” represents a crossover site):

Table 4: Crossover with two crossover sites

Chromosome 1	11011 0010011 0110
Chromosome 2	11011 1100001 1110
Offspring 1	11011 1100001 0110
Offspring 2	11011 0010011 1110

Crossover probability affects how often the offspring will be changed due to crossover. Crossover is made in hope that new chromosomes will have desirable features of the old chromosomes. However, it is beneficial to retain some individuals to survive into the next generation.

Mutation operator is performed after crossover operator. Mutation helps to prevent solutions in population from converging into a local optimum. Mutation changes randomly some values (or bits values) in chromosome to make a new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. An illustration of mutation operation is shown in Table 5.

Table 5: One example of mutation operator

Original offspring 1	1101111000011110
Original offspring 2	1101100100110110
Mutated offspring 1	1100111000011110
Mutated offspring 2	1101101100110110

Similar to crossover, the exact mode of carrying out mutation depends very much on the method of solution's coding. For example, with permutation string coding, mutation could be effected by exchanging two genes. The frequency or extent of mutation is governed by the mutation probability. Mutation probability affects the frequency of mutation to be performed on a chromosome. The higher the mutation probability, the more frequency mutation is applied to the offspring. If there is no mutation, offspring is taken after crossover (or copied) without any change. If mutation is performed, part of the chromosome is changed. Mutation is applied to prevent GA from falling into local extremes. However, it should not occur too frequently; doing so

is a kin to a random search. If mutation probability is 100% then all bits in chromosome are mutated which cause the chromosome string to be inverted. That means we have inversion and no mutation, such drastic manipulation of the string have the effect of rapid destruction of good solutions that have been built up. In practice, we set up mutation and crossover possibility (which depends on each solved problem) to achieve a suitable level of inheritance and random perturbations.

5.3 Local Search

An important component in memetic search is the local search method. This allows the algorithm to refine a solution so as to speed up convergence towards the optimum. In local search, it is also common to include domain-specific knowledge to make the search more effective. In our current work, we have not incorporated any domain-specific knowledge about the problem to speed-up the computation of the local search. Instead, we employed a 2-opt mutation swap operation to carry out localized solution refinement. The 2-opt is executed with a probability of 0.2. If a swap results in a better solution, it is used to replace the existing current chromosome. This continues up to a pre-set limit of 120 times.

Figure 24: An illustration of a 2-Opt swap operation.

For our evolutionary algorithm, we set the crossover probability to 0.5. The crossover implemented is a form of random two-point crossover; the two points defining the starting and ending gene whereby crossing over between two chromosomes occur. In our current implementation, we set a termination criterion based on the number of generations which is currently set at 600.

5.4 ADEP Program

Algorithm Development Environment for Problem Solving (ADEP) is a Problem Solving Environment for configuring meta-heuristics for solving real-life combinatorial optimization problems [22]. It was developed to address the need for rapid configuration of efficient algorithms for real-life optimization problems. We used ADEP for configuring the properties and operators used in our memetic approach, such as adding local search, parent selection method, etc. ADEP reduces the development cycle time and is a convenient platform to test out different methods/operators for a given problem.

Figure 25: ADEP interface

Chapter 6 Simulation

In this chapter, we present the results of experimentation using the UAV model developed by Boeing Research Technology, and demonstrate the effect of different scanning strategies on the total scanning time. Finally, we show the results using real UAVs at Boeing's laboratory in Seattle Washington to demonstrate our approach.

6.1 Simulation Using UAV Model

We create a test case as shown in Fig. 26. The bases are shown by crosses and the capability of the UAVs located at each base is shown in Table 6. The first step involves dividing the polygon into a number of sub-regions corresponding to the number of bases. Fig. 26 shows one solution for polygon decomposition using random sweep line algorithm. The area of each sub-region is proportionate to the relative capability of the base.

Figure 26: Polygon decomposition.

Table 6: Information of base stations in Fig. 26

Base	UAV capability					Total Capability
Base1	10	12	15	18	20	75
Base2	12	15	16	18		61
Base3	22	20	24	19		85
Base4	20	22	24	26	25	117
Base5	18	19	20	22		79

After dividing the polygon accordingly, a memetic approach is applied to optimize scanning time. An example of the result derived is shown in Fig. 27. We carried out two simulation runs and the plots of fitness against the number of generations are given in Fig. 28.

Figure 27: Assigned regions for all the UAVs

Figure 28: Plot of fitness versus generations

Clearly, the order of UAVs assignment as dictated by the chromosome string will affect the scanning time. Based on the above problem scenario, the best solution found requires a total of 340 seconds for turning to scan the whole region. Figure 27 shows the partitioning of the polygon into sub-regions according to the number of base stations and with each sub-region, the division of the sub-region into smaller polygons according to number of UAVs within the base station. The figure does not include information on the scanning method for each UAV. The assignment of scanning is derived by the memetic computing module as shown in Figure 28, which shows the results for two simulation runs. Each plot in Fig. 28 represents a single run of the memetic algorithm. It can be seen that the best tasks allocation solution of minimum total turning time at

each generation reduces as the number of generations increases. After around 400 generations, the solution converges towards a best tasks allocation cost of 340 seconds of total scanning time.

The algorithm is developed to test with actual UAV model developed by Boeing. In the experiment, we consider 7 UAVs being deployed. The algorithm assigns task to each UAV and determine the best scanning method for the UAVs as shown in Figure 29. Here, base 1 has 3 UAVs, base 2 and base 3 each has 2 UAVs.

Figure 29: Optimization of scanning time

The algorithm can send results to the management system framework which then transmits commands to control the actual flights of the UAVs. For more information about mission management framework and VSTL (Boeing Vehicle Swarm Technology Lab), readers are referred to [8] and [10].

Figures 30 and 31 show the graphical visualization of VSTL and SwarmView developed by Boeing respectively. It enables us to test and observe UAV operations based on the output results of the memetic approach. The results are indicative that our algorithm can be deployed in real-life scenarios. It further illustrates the capacity to incorporate new scanning strategy in dependence on the resources available to carry out a mission. The algorithm is scalable to handle large number of bases and UAVs and is potentially valuable to reduce operational cost by reducing the overall time to complete area coverage.

Figure 30: Testing with actual UAVs in VSTL

Figure 31: The performance of the algorithm observe in SwarmView

We also developed a 3D visualization platform to simulate our tasks allocation algorithm. The advantage of using 3D simulation is that it gives a more realistic depiction of the actual flight scenarios. In this version, we use several UAVs to scan a region which has several robots moving on the ground. While scanning the region assigned, the UAVs also try to detect the presence

robots. If the presence of a robot is detected, the UAV fires with intention to destroy. Figure 32 illustrates the simulation.

Figure 32: 3D visualization of the tasks allocation algorithm

6.2 Effect of Scanning Strategies on Total Scanning Time

The objective is to study the effect of scanning methods on the total scanning time. We test with a scenario as shown in Fig. 33.

Figure 33: Polygon decomposition

Table 7: Information of base stations in Fig. 33

Base	UAV capability				Total Capability
Base1	14	20	19	22	75
Base2	18	15	20	18	71
Base3	22	20	24	19	85
Base4	20	22	24	18	84
Base5	25	24			49

After dividing the polygon accordingly, a memetic approach is applied to optimize the scanning time. An example of the result derived is shown in Fig. 34.

Figure 34: Assigned regions to UAVs

Clearly, the order of UAVs assignment as dictated by the chromosome string will affect the scanning time. Based on the above problem scenario, the best solution found incurs a turning time cost of 929 seconds to scan the whole region. Fig.35 shows the plot of fitness (total turning time of all UAVs) versus the number of generations using memetic computation.

Figure 35: Fitness versus generations using both scanning method

We also compare with the scenario whereby all the UAVs scan using raster or circular scanning method. Fig. 36 depicts the fitness versus the number of generations for the case where all the UAVs scan using the raster method. Fig. 37 shows the result using only circular scanning method for all UAVs.

Figure 36: Fitness versus generations using raster scanning method

Figure 37: Fitness versus generations using circular scanning method

Clearly, we can see that by integrating both scanning methods in memetic computation give us the best results compared to using either raster or circular scanning alone. Fig. 38 shows the comparison between these methods. Simulation 0 is for the case using both scanning methods, simulation 1 for using only circular scanning and simulation 2 for using only raster scanning.

Figure 38: Compare results between three cases

For the case of a UAV using only random scanning method, the results are as shown in Fig. 39.

Figure 39: Fitness versus generations using random scanning method

From these plots, we can see that using memetic approach with both scanning methods gives minimum total turning time of 929 seconds compared with using only raster scanning of 1286 seconds, circular scanning alone of 997 seconds, and for the random case, 964 seconds.

The results are indicative that our approach is capable of producing cost-effective results. Although we have considered only two scanning strategies in this work, further strategies for scanning a region can be easily incorporated to enrich the tasks allocation framework. The algorithm is scalable to handle large number of bases and UAVs, and is potentially valuable to reduce operational cost by reducing the overall time to complete area coverage.

6.3 Integration with VSTL and Testing with real UAVs

We use socket as a bridge to link between our programs with the mission management framework. One script opens a socket and listens to this socket. Our program sends data to the socket, which will be processed by the mission management framework to control all UAVs to follow a series of points. The diagram in Fig. 40 shows how our system is integrated with the mission management framework:

Figure 40: Integration of tasks allocation algorithm with Mission Management Framework

We tested our system with real UAVs at Boeing Research Technology laboratory and Figure 41 is a picture of the facility for flight testing.

Figure 41: Testing the tasks allocation algorithm with 6 quad-rotor UAVs

In the experiment carried out at Boeing's VSTL, we perform testing with 6 UAVs. The picture shows two UAVs at the top, two on the left, one at the bottom and one at the right of the rectangle. The results of actual path scanning for each UAV is shown in Fig. 42

Figure 42: Actual flight paths of UAVs during mission execution.

From the experiment, we can see that the UAVs deployed execute the tasks plan derived by the tasks allocation algorithm. However, due to the effect of collision avoidance routine being activated when two UAVs are in close proximity, the UAVs at times were observed to have deviate away from the actual intended paths.

Figure 43 shows the corresponding separation distance among the vehicles and number of active pairs involved in local conflict resolution. One nice feature of the algorithm is an inherent guarantee of the separation due to the partitioning of the polygon. In reality, however, the local conflicts occurred on the boundary of partitioned space which is partly due to incorrect sensor footprint. The situation is further aggregated by the physical limitation constraint of the flight space. In actual scenarios, space constraint may be less of an issue. Furthermore, secondary path planning to reduce the effect of UAVs flying close to each other, especially at the boundary of sub-regions can be incorporated. This will serve to reduce the frequency of activation of the collision avoidance routine.

Figure 43: Separation distance among the vehicles and number of active pairs on the local conflict resolution.

From observation during experimentation, we realized that by letting the UAVs' flights to start from the inner region and working its way toward the outer region in a circular manner, the number of conflicts can be reduced. There are times when it was observed that one or more UAVs may fail doing the mission. However, in most of the cases the UAV failed at the beginning of the task. In such scenarios, the disabled UAVs are removed from the mission and the tasks allocation algorithm is invoked to derive new tasks plan for the UAVs. For other modes of failure during the mission, we present a re-planning scheme to handle the problem in the next chapter.

Chapter 7 Re-Planning

In this chapter, we address the issue of UAV failures during a mission. Dealing with UAV failures is an important consideration in real-life mission execution. We introduce two methods to handle the problem.

7.1 First Approach to Re-planning

The first approach to handle re-planning is based on the following rules:

- UAVs in the same base will handle the remaining tasks of failed UAVs.
- If all the UAVs in one base fail, then UAVs from the adjacent base will be assigned to handle the remaining tasks of the failed base.
- UAVs are required to complete their tasks first, before flying over to the unexplored region meant for the failed UAVs to complete the added task.

Some scenarios of leftover region due to a failed UAV and the approach to handle it are shown in Fig. 44. The shaded region in the polygon refers to area which has been successfully covered by a UAV.

(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

(e) Scenario 5

Figure 44: Some failure scenarios and proposed solutions

We consider five scenarios and refer to each as scenario 1, scenario 2, etc. In scenario 1, only UAV 2 fails. UAV 1 or UAV 3 which finishes its task first will handle the remaining task for UAV 2. If the remaining area of UAV 2 requires more than the remaining capability of UAV 1 or UAV 3, then both UAVs will be used to accomplish the unfinished task. In scenario 2, two UAVs fail then UAV 3 will handle the remaining task for UAV 2 and UAV 4 will handle the remaining

task for UAV 1. In scenario 3, we have two UAVs failing. If UAV 5 finishes its task first, we can assign it to handle remaining task for UAV 4. If UAV 3 finishes its task first, we assign it to handle remaining task for UAV 2. If UAV 1 finishes its task first, followed by UAV 5, we can assign UAV 1 to handle the remaining task for UAV 2 and UAV 5 to handle the remaining task for UAV 4. Otherwise, we assign UAV 1 and UAV 3 to pick up the slack for UAV 4 and UAV 2 respectively. In scenario 4, UAV 1 has sufficient capability to handle the remaining tasks for UAV 2 and UAV 3. In last scenario, all UAVs in base b3 fail. UAV in bases b1 and b2 will be deployed to handle the remaining tasks for all UAVs in base b3. For example, UAV 3 and UAV 4 handle the remaining tasks for UAVs in base b3. If they have insufficient capability, then another UAV, such as UAV 7 may be required.

Fig. 45 shows the protocol in the overall integration of the various subsystems during flight testing. The module "NTU app" refers to the tasks allocation algorithm for region coverage.

Figure 45: System integration involving Mission Management Framework, VSTL and NTU app showing the protocol for data communication.

This scheme has some inherent complications from an implementation point of view:

- The need to keep track of adjacency between UAVs in a same base, and adjacency between the bases as well;
- The need to handle polygon division involving non-convex polygons.

7.2 A Feasible Re-planning Approach

Clearly, the first approach while feasible for many scenarios has some serious limitations, especially from an implementation point of view. We consider an alternative method to address the issue. The second method can be described as follow:

- Keep track of all failed UAV during the mission including *id* and location
- Compute the capability required to handle the unfinished task for each failed UAV.
- Check which UAV finishes its task first, and note the remaining capability of this UAV. If it has sufficient capability to accomplish the remaining task of the failed UAV, then this UAV is assigned to handle the task remaining.

As a simple illustration, consider the case of UAV_r finishing its task and has sufficient capability to handle the remaining area for UAV_k. UAV_r will fly to the location of UAV_k and continues with the unfinished task.

Figure 46: System integration showing data communication protocol for re-planning

In summary, the tasks allocation algorithm for re-planning can be described as follow:

- NTU app sends an information packet to MMF <UAV #*id*, series of points to scan>
- VSTL records which UAV fails during the mission (UAV #*id*, current location) and sends information back to MMF.
- MMF computes the remaining area for each failed UAV, capabilities of remaining UAVs. It also computes which UAV finishes its task earlier than the other UAVs.
- MMF allocates remaining tasks to the remaining UAVs based on the remaining capability of UAVs and remaining area to be covered.

These solution approaches also take into account the adjacency between UAVs. If one UAV fails, the algorithm will check all available UAVs and find the nearest available and assigns the new task to this UAV.

If no UAV has remaining capability to handle the remaining area of the failed UAV, then there is a difficulty for MMF to handle the tasks division for multiple UAVs since it does not know the shape; it only know the remaining series of points. In this case, we can divide the remaining series of points into 2 parts and allocate each part to one UAV. However, the problem of collision avoidance may affect the performance since it is likely that the presence of more than one UAV in the leftover region by the failed UAV gives rise to greater occurrence of collision avoidance due to the proximity of the UAVs.

7.3 Interface Script

The interface written in Python language is used as a bridge to connect our tasks allocation application with the mission management framework (MMF) developed by Boeing to monitor and control the UAVs. We first start all modules of the MMF, then open the socket and listen to receive data from our application.

The syntax to start MMF's modules is as follow:

```
init(sizeOfVolume=(6.0,10.0,6.0),commIP='155.69.52.142',commPort=20000,
posPort=9999)
```

The syntax to read data and launch all UAVs in the MMF:

```
# get initial heli objects

helis = getObjects(type=Type.HEL1)

# activate and take off all UAV here

for j in helis:

    comm_bindings.launchVehicle(j)
```

The syntax to open socket:

```
HOST = '155.69.52.142' # This is the ip address of a pc

PORT = 1003          # Arbitrary non-privileged port

maServerSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

maServerSock.bind((HOST, PORT))

maServerSock.listen(1)
```

```
conn, addr = maServerSock.accept()
```

The interface opens a socket and listens to receive data. Once data is sent to the socket, the interface will capture and interpret data. The data correspond to the series of waypoints that define the scanning path for each UAV.

```
# get packet from NTU app
```

```
format_strI = ('4H' + str(maxwp*3) + 'f' + str(2*maxvert) + 'f')*maxveh
```

```
# format string for incoming packet structure
```

```
print "incoming packet size:", calcsiz(format_strI)
```

```
print ('Connected by', addr)
```

```
while receiving:
```

```
data = conn.recv(27360)
```

```
print "incoming packet size: ", len(data)
```

```
recv_packet = unpack(format_strI, data)
```

```
receiving = False
```

```
conn.close()
```

Once data is received, the interface will close the socket and decode the data to extract the necessary information. Then, it sends command to the mission management framework to control all UAVs that execute the scanning.

The syntax to command a UAV following a series of points is as follow:

```
comm_bindings.flyToWaypoint(helis[j], x, y, z, 0, 0, veh_speed)
```

Here, *helis[j]* is the *id* of the UAV; (*x, y, z*) is the coordinate of target point and *veh_speed* is the speed allowed for UAV to fly to the target point. The interface then listens to MMF to detect if any UAV fails during the mission so that it can start re-planning if necessary.

The code to check if a UAV crashes, and then detects the *id* and location of all crashed UAVs stored as *crash_id* and *crash_pos* respectively is given below:

```
vehsCrashed = getObjects(state=State.CRASHED)
```

```
crash_pos = list()
```

```
crash_id = list()
```

```
for veh in vehsCrashed:
```

```
crash_pos.append([veh.x, veh.y])
```

```
crash_id.append(veh.id)
```

The code to check if there is any available UAV which is free or had finished its tasks is as follow:

```
vehTaskable = getObjects(state=State.FLYING_TASKABLE)
```

If there is a UAV available to deal with the slack, the interface will determine whether this UAV has sufficient capability to handle the remaining area left behind by the failed UAV. If so, this UAV will be assigned to scan the remaining area left by the failed UAV. The interface now continues to listen to check if other UAVs fail during the mission.

Chapter 8 Conclusions and Future Research Directions

This chapter briefly summarizes the research carried out in this thesis. It also discusses the findings and conclusions, and indicates some possible future research directions.

8.1 Summary and Conclusions

The thesis gives a detailed account of the implementation of a memetic tasks allocation algorithm that generates a plan to cooperatively comb a given area (convex polygon) by a team of UAVs parked at different base stations, taking into account their different sensing and range capabilities. Two methods for polygon decomposition are introduced and a memetic technique is used to optimize total scanning time for all UAVs. The method presented in this report can be easily extended to the cooperation of autonomous aerial and ground vehicles, and can be useful in some real-life applications, such as search and rescue missions. To the best of our knowledge, our work reported here represents the first attempt to incorporate quantitative turning time model of unmanned aerial vehicles. The thesis also provides detailed analytical comparison between two strategies of scanning. The need for re-planning to deal with UAV failures during mission execution is an important issue. To address this, we present a re-planning scheme based on heuristics. The memetic tasks allocation approach was tested with UAVs deployed in a Boeing's laboratory setting and the results are indicative of the algorithm's viability. The re-planning scheme implemented will be the focus of testing in our next meeting at Boeing's VSTL facility.

8.2 Future Research Directions

The current research can be extended in a number of different ways.

- The current project is based on a central command and control (CCC) methodology; all the processing pertaining to tasks division and re-planning is centralized and integrated into a system or framework running on one or multiple computing platforms. We can use de-centralized approach to handle the task. With this approach, each UAV behaves as one intelligent agent with capability to send and receive data from other UAVs and can automatically divide tasks among them. This approach is advantageous as it is more reactive for suitable dynamic and chaotic environment. It is potentially more robust since it is less likely to be constrained by the bandwidth capacity of the command centre. A CCC approach requires that all the UAVs fly within a range to ensure that communication channel remains intact. Furthermore, a communication bottleneck at the command centre is a potential issue to consider.

- The current project only compares the turning time between raster and circular scanning strategies. In fact, there are more scanning strategies which can be considered. Provisions for more diverse scanning strategies can be incorporated to enrich the search capacity of the memetic tasks allocation technique.
- Although not demonstrated, it is clear that the approach considered in our work here is extensible to scenarios involving a multitude of unmanned vehicles. For example, tasks allocation in a heterogeneous setting involving multiple UAVs and ground autonomous vehicles (GAV) can be further explored and validated.

References

- [1] S. Hert, V. Lumelsky, "Polygon area decomposition for multiple-robot workspace division", International Journal of Computational Geometry and Applications, Volume 8, Number 4, August 1998, pages 437-466
- [2] A. Agarwal, MH Lim, M. J. Er & T. Nghia, "Rectilinear workspace partitioning for parallel coverage using multiple UAVs", Advanced Robotics, Vol 21, No. 1, January 2007
- [3] I. Maza, A. Ollero, "Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms", Proceedings of the 7th International Symposium on Distributed Autonomous Robotic Systems, France, 2004.
- [4] M. Mitchell, "An Introduction to Genetic Algorithms", the MIT Press, 5th Edition 1999
- [5] Holland JH, "Adaptation in natural and artificial systems", Ann Arbor: University of Michigan Press, 1975.
- [6] M. Kreveld and I. Reinbacher, "Good NEWS: Partitioning a Simple Polygon by Compass Directions", Proc. of the 19th Annual ACM Symposium on Computational Geometry, pp 78-87, San Diego, USA, 2003.
- [7] C. Bolkcom, "Homeland Security: Unmanned Aerial Vehicles and Border Surveillance", Foreign Affairs, Defense, and Trade Division, Domestic and Social Policy Division, Federation of American scientists, July 2010.
- [8] P. Pandey, S. Hemant, and V.K. Doan "An Agent Based Trading Game for Risk Adversity Level Estimation", Proc. of the International Conference on Soft Computing and Pattern Recognition, Malaysia, 2009.
- [9] M.A. Bayir, H. Toroslu, and A. Cosar, "Genetic Algorithm for the Multiple-Query Optimization Problem", IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews, Vol. 37, No. 1, January 2007.
- [10] B. Bethke, J.P. How, A. Frank, D. Dale and J. Vian, "Real-Time Indoor Autonomous Vehicle Test Environment", IEEE Control Systems Magazine, April 2008, pp 51-64.
- [11] J. Tang, M.H. Lim and Y.S. Ong, "Parallel Memetic Algorithm with Selective Local Search for Large Scale Quadratic Assignment Problems", International Journal of Innovative Computing, Information and Control, Vol. 2, No 6, pp. 1399-1416, Dec 2006.
- [12] B.H. Gwee & M.H. Lim, "Polynominoes Tiling by a Genetic Algorithm," Computational Optimization and Applications Journal, Vol.6, No.3, Nov 1996, pp 273-291.
- [13] Y.S. Ong, M.H. Lim and X. Chen, "Memetic Computing – Past, Present and Future", Research Frontier Article, IEEE Computational Intelligence Magazine, June 2010.
- [14] B. Bethke, J. P. How, and J. Vian, "Group Health Management of UAV Teams With Applications to Persistent Surveillance", 2008 American Control Conference Seattle, Washington, USA, June 11-13, 2008.

- [15] M.A. Bayir, I.H. Toroslu, and A. Cosar, “Genetic Algorithm for the Multiple-Query Optimization Problem”, IEEE Transactions on Systems, Man, and Cybernetics-part C: Applications and Reviews, Vol. 37, No. 1, January 2007.
- [16] M. Valenti, B. Bethke, J. P. How, J. Vian, “Embedding Health Management into Mission Tasking for UAV Teams”, Proceedings of the 2007 American Control Conference, New York, NY, June 2007.
- [17] R. Meuth, M.H. Lim, Y.S. Ong and D. Wunsch, “A Proposition for Memes and Meta-memes in Computing for Higher-Order Learning”, Memetic Computing, Vol. 1:2, June 2009
- [18] M.H. Lim, S. Gustafson, N. Krasnogor and Y.S. Ong, Editorial – Inaugural Issue, Memetic Computing, Vol 1:1, March 2009.
- [19] Z. Yueqin, L. Jinfeng, D. Fu and R. Jing, "Genetic Algorithm in Vehicle Routing Problem", In Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2007, Vol. 2, pp. 578-581.
- [20] A. H. Wright, "Genetic Algorithms for Real Parameter Optimization", G. J. E. Rawlins editor, “Foundations of Genetic Algorithms”, pages 205-218, Morgan Kaufmann Publishers, 1999.
- [21] Z. Michalewicz, “Genetic algorithms + data structures = evolution programs”, New York: Springer, 1994.
- [22] Y. L. Xu and M. H. Lim, "Solving constraint satisfaction optimization problem using an integrated algorithm development environment", In Proceedings of 10th International Symposium on Integrated Circuits, Singapore, 2004.
- [23] Shingo Ito, “Japan says Fukushima different from Chernobyl”, <http://blogs.forbes.com/oshadavidson/2011/04/26/fukushima-and-chernobyl-similar-but-different/>, 21 April 2011
- [24] S. Goldenberg and J. McCurry, “<http://www.guardian.co.uk/world/2011/mar/29/japan-nuclear-plant-us-robots>”, 29 March 2011

Appendix A - Simulator Screen Shots

Figure 47: Tasks allocation program interface

Figure 48: Mission Management Framework

Figure 49: Swarm View is used to display information and send command to UAVs

Appendix B- Syntax to Interact with MMF

(Source from Boeing Mission Management Framework API provided by Boeing Research & Technoly)

Initialization Function	Description
init(sizeOfVolume =(6.0,10.0,6.0), commIP = '127.0.0.1', commPort =20000, posPort =9999)	Initialize all modules
initTask()	Initialize the Task Manager
initPersist()	Initialize the Persistence Manager
initMission()	Initialize the Mission Manager
initPos(port)	Initialize the Position Bus
initComm(ip , port)	Initialize the Communication Bus
initCommHandlers()	Initialize all Communication Handlers
Object Functions	Description
getObjects([name, id, state, type, near])	Get objects filtered by criteria. A call to this function with no arguments returns all objects.
getObject([name, id, state, type, near])	Get first object as filtered by criteria.
getChargers([chargeid, vehid, near])	Get chargers filtered by criteria. A call to this function with no arguments returns all chargers.
getCharger([chargeid, vehid, near])	Get first charger as filtered by criteria.

Mission Functions	Description
getMissionMgr()	Returns the global mission manager. This may be None if not initialized.
addElement(trigger , element , clearPrevious=False)	Add the element to the mission.
setStartFunction(func)	Set the function that runs at the start of the mission.
setStopFunction(func)	Set the function that runs upon completion of the mission.
addIterationFunction(func)	Add a function to run on every iteration of the global mission manager.
addStopCondition(func)	Add a function as a stop condition to the global mission manager.
advanceSequence()	Advance the current sequence number.
getSequenceNumber()	Get the current sequence number.
Tasking Function	Description
getTaskMgr()	Returns the global task manager. This may be None if not initialized.
delegateTask(task , type=Type.HELIX)	Delegate the task to a vehicle of the given type.
assignTask(task , vehicle)	Assign the task to the given vehicle.
clearTask(task)	Remove the task from the global task queue and sends a clearTasks message to the vehicle if the task is being worked.
Persistence Functions	Description
getPersistMgr()	Returns the global persistence manager. This may be None if not initialized.
requestVehicles(n , type=Type.HELIX , wait=False)	Request vehicles of the given type from the global persistence manager, optionally waiting for them to become active. The number passed in is added to the total number of requested vehicles.
releaseVehicles(n , type=Type.HELIX)	Release vehicles of the given type from the global persistence manager. The number passed in is subtracted from the total number of requested vehicles. This is the opposite of requestVehicles.

Communication Bus Functions	Description
getCommListener()	Returns the global comm listener. This may be None if not initialized.
getCommHandlers()	Returns the global comm handler dictionary.
addCommHandler(messageId , handlerFunc)	Adds a new handler to the global comm handler dictionary.
getCommBroadcaster()	Returns the global comm broadcaster. This may be None if not initialized.
broadcast(packet)	Broadcast the given packet on the global Comm Bus.
Position Bus Functions	Description
getPositionListener()	Returns the global position listener. This may be None if not initialized.
Miscellaneous Functions	Description
start()	Start the Mission.
stop()	Stop the Mission.
isCompleted(o)	Test if the given object is complete (usually task or element).

Appendix C - Swarm View Command

(Source from Swarm View manual provided by Boeing Research & Technology)

This document describes the SwarmView Ground Station Operator Interface software. This software is one component of a development test-bed for health-adaptive control and monitoring of a large number of heterogeneous autonomous vehicles.

Tasks for a vehicle or group are specified using one of the following key-words (this may be a volatile list). There are up to eight floating point arguments for each task, depending on the type. See the appendix B for a description of the required arguments for each task type. The standard SwarmView installation comes with a template file (available via the Startup menu folder) that more fully describes all of the options available with each command.

- Internal Use Only:
 - WAIT

- Operational Commands:
 - ACTIVATE
 - STANDBY
 - TAKEOFF
 - PREPARE_TO_LAND
 - LAND
 - REFUEL
 - MAINTENANCE
 - DEACTIVATE
 - SET_FLIGHT_CONTROL_MODE
 - SET_VEHICLE_MODE
 - CLEAR
 - SET_YAW_CONTROL_MODE

- Tasking Commands:
 - FLY_TO_WAYPOINT
 - HOVER_AT_LOCATION
 - LOITER_AT_LOCATION
 - SEARCH_LOCATION
 - RETURN_TO_BASE
 - FOLLOW_OBJECT
 - TRACK_OBJECT

- TARGET_ASSESSMENT
 - TARGET_ENGAGEMENT
 - SYSTEM_ID
 - LAND_AT_LOCATION
 - ENABLE_JOYSTICK
- Health Commands:
 - SYSTEM_EMERGENCY
 - HEALTH_STATUS_REQUEST
 - SIM_COMPONENT_FAILURE
- Group Commands:
 - GROUP_ADD
 - GROUP_REMOVE

For all information of each tasking command, please read the manual for Swarm View. Here, we only describe a command to let UAV follow a Bezier curve which is used to find the turning time model for UAVs.

This option is only available when exactly one vehicle is selected. Selecting this option appends a “rubber-band” line to the end of the vehicles’ planned path, allowing the user to interactively create a sequence of Bezier waypoints to form a smoothly curving path. Each successive left-click while in this “creation” mode appends a waypoint with its associated control point automatically placed at a default relative position.

The following figure shows the interactive Bezier editor in the “creation” mode, with the rubber-band line extending from the most recently-placed waypoint, ready to place the next waypoint. The waypoints are shown as small red squares with straight red lines between them as a visual reference. The control point for each waypoint is shown as a small green square, connected to its waypoint by a green line. As can be seen in the figure, the relative location (both direction and distance) of the control point determines the curvature and direction of the Bezier path as it passes through the waypoint. If the control point is co-located with the waypoint, the path makes an instantaneous change in direction to the next waypoint. The greater the relative distance of the control point, the more gently the path curves.

Figure 50: Interactive Bezier curve

When all the desired waypoints have been placed, press the “Escape” key (ESC) or double-left-click to exit the “creation” mode to enter the “editing” mode. The rubber-band line disappears and a mode-less “Settings” dialog like the one shown below appears. Move the dialog to a convenient out-of-the-way corner of the screen in order to complete editing the Bezier curve.

Figure 51: Bezier curve setting chart

In the “editing” mode, the red waypoints and the green control points can be selected individually by left-clicking with the mouse, and dragged to new locations to modify the shape of the curve. The selected waypoint or control point is highlighted with a larger bold square around it. Selected waypoints (but not control points) can be deleted one at a time by pressing the “Delete” (DEL) key.

Also, while in “editing” mode, the “Settings” dialog is available to change the characteristics of the secondary “terminating” Bezier curve (if any). The “Settings” dialog also has an Append button which puts the Bezier editor back into “creation” mode (hiding the “Settings” dialog temporarily).

The secondary “terminator” Bezier curve can be used for vehicles that are unable to remain motionless in-place, requiring a path for continuous motion when not otherwise engaged. The available Termination Types are as follows:

- **None:** No termination – the final waypoint control point is directly on top of the waypoint.
- **Repeat:** The final waypoint returns smoothly to the first waypoint.
- **Circle:** Appends a clockwise circular path at the location of the last waypoint. Must also specify a radius in meters.
- **Racetrack:** Appends a clockwise racetrack shape with two half-circles connected by two parallel straight legs. Must also specify a radius in meters for the half-circles, a length in meters for the straight legs, and a counter-clockwise angle in degrees for the straight legs to make with the positive x-axis.
- **Figure-8:** Appends a figure-8 shape that is similar to the racetrack, except that the straight legs cross in the middle such that the two half-circles are traversed in opposite directions. Required input parameters are the same as for the racetrack.

To apply changes in the “Settings” dialog for viewing without the dismissing the dialog, click the “Apply” button. To discard changes made in the dialog along with any edits (movements, additions and/or deletions) made to control points, click the “Cancel” button or press the “Escape” (ESC) key. To save changes, and dismiss the dialog and the Bezier editor, press the “OK” button or press the “Return” (RET) key.