2004

Comparison of partially decoupled and combined methods of path planning and task allocation

Jennifer Beth Hazelton
West Virginia University

Comparison of Partially Decoupled and Combined Methods of Path Planning and Task Allocation

Jennifer Beth Hazelton

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

Marcello Napolitano, Ph. D., Chair
Gary Morris, Ph. D.
Jacky Prucz, Ph. D.

Department of Mechanical and Aerospace Engineering

Morgantown
West Virginia
2004

KEYWORD: UNMANNED AERIAL VEHICLES

Abstract

Comparison of Partially Decoupled and Combined Methods of Path Planning and Task Allocation

Jennifer Beth Hazelton

Developing autonomous unmanned aerial vehicles (UAVs) reduces the risks to which soldiers are subjected by enabling the UAVs to make efficient decisions, regardless of the situation. This requires each group of UAVs to be proficient in planning their own paths and assigning tasks in a way that minimizes the total cost of the mission. Two methods are presented for doing this, the partially decoupled approach and the combined approach. After comparing two methods, the partially decoupled approach costs an average of 3.0 meters less than the combined approach, while taking an average of 0.327 seconds longer to complete. This indicates that the partially decoupled method should be chosen if the main concern is the cost of the mission and the combined approach should be chosen if computational time is the main concern.

Table of Contents

List of Tables

List of Figures

Nomenclature

Variable	Description	Units
A	constraint coefficient matrix, state space system matrix	---
B	state space input matrix	---
b	constraint values vector	---
c	objective function coefficients vector	---
dist	distance	m
ε	weighting factor	---
f	force, force vector	N
g	number of times a target must be visited	---
J	objective function utility	---
M	number of segments	---
m	index of M	---
N	number of	---
R	relaxation factor	---
s	state vector	---
t	time, time step	s ---
$\bar{t}$	overall mission completion time step	---
v	velocity	m/s
W	target matrix	---
x	x-location, decision variable vector	m
$\dot{x}$	velocity in x-direction	m/s

y	y-location	m
$\dot{y}$	velocity in y-direction	m/s
Z	no-fly zone matrix	m

Subscript	**Description**	**Units**
i	target designation	---
j	no-fly zone designation	---
k	direction $(1 = +X, 2 = -X, 3 = +Y, 4 = -Y)$	---
max	maximum	---
p	vehicle designation	---
t	time, time step designation	s ---
v	vehicles	---
w	targets	---
z	no-fly zones	---
0	time step = 0	---
1	first of a set, x-position	---
1:4	locations 1 through 4 in relation to the no-fly zones	---
2	second of a set, y-position	---

Acronym	**Description**	**Units**
AMPL	A Modeling Language for Mathematical Programming	---
CPU	central processing unit	---
MILP	mixed-integer linear program	---

MMKP	multi-dimensional multiple-choice knapsack problem	---
POK	probability of kill	---
UAV(s)	unmanned air vehicle(s)	---
VRT	virtual reality toolbox	---

Chapter 1: Introduction

1.1 Thesis Objective

The United States' military has assumed a more physical role in the world's affairs, with Operation Enduring Freedom and Operation Iraqi Freedom. These operations are intended to make the world safer; however, soldiers' lives are still at risk. One way to decrease the risks to which soldiers are subjected is increasing the use of unmanned air vehicles (UAVs). Currently, UAVs only exist as remotely piloted platforms. The amount of manpower required for operation decreases as the level of autonomous control increases.[9]

In the future, it is desirable to have UAVs capable of providing reconnaissance information and delivering ordnance to specified targets independently. Autonomous UAVs are capable of making decisions efficiently. They do not suffer from the stresses that affect humans and their decision making abilities. This enables the vehicles to respond quickly to rapidly changing environments. Each UAV task force will be capable of cooperative path planning and task allocation. Ideally, UAVs will make cooperative, optimal decisions to minimize the total cost of the mission.

This thesis discusses and compares two methods that provide the UAVs with the tools necessary for path planning and task allocation. The paths must be known before any tasks can be assigned. However, the paths cannot be planned unless the UAV knows which task it must complete. The task allocation refers to assigning each vehicle a specific target to visit. Solving these problems in a partially decoupled manner involves, first, generating the cheapest paths from each UAV to every target. Then, the tasks are allocated to minimize the overall cost of the mission while ensuring every target is

visited. The costs of the paths may include fuel cost, related to distance traveled, and threat risk cost, associated with the threats on the battlefield. A second approach to solving the path planning and task allocation problem maintains the combined relationship between them. The paths are planned and tasks are assigned simultaneously.

Five scenarios, each involving two UAVs, two targets, and one no-fly zone are investigated. Each scenario is evaluated with the partially decoupled approach and the combined approach. The two approaches have the same input data and produce comparable plots indicating the paths the UAVs will take to complete their tasks. The results will indicated which method costs more to complete and/or is more computationally intensive.

1.2 Survey of Previous Work

Many approaches for path planning and task allocation have been investigated. These include different types of trajectory generators, such as using visibility grids and Voronoi diagrams, and different types of solution methods, including hierarchical, partially decoupled, and combined methods. The results from previous efforts indicate that completely autonomous UAVs are not far in the future. For the purposes of this thesis, one trajectory generation approach and three complete approaches were found to be most applicable.

The trajectory generation approach presented here involves work by Timothy W. McLain and Randal W. Beard at Brigham Young University and McLain with Phillip R. Chandler, Steven Rasmussen and Meir Pachter at Wright-Patterson Air Force Base. McLain and Beard present a method for generating flyable trajectories that minimize the risk to the UAVs. To minimize the risk further, the targets are attacked simultaneously.

The locations of the UAVs (initially), threats, and targets are known. Tasks are assigned prior to the trajectory generation; specifically, each UAV knows the target it is to attack. A Voronoi diagram, based on threat locations, provides paths that minimize the threat risk to the UAVs as they travel to the targets. However, the Voronoi diagram does not have an internal mechanism to attach the UAVs, at their initial locations, or the targets to the diagram. Therefore, these UAV positions and target locations are connected to the respective three closest nodes of the diagram. Costs, based on threat risk and fuel consumption, are assigned to each edge of the Voronoi diagram. Dijkstra's algorithm searches the Voronoi diagram to determine the least expensive paths from each UAV to its particular target.[16]

The UAVs are given the same, constant velocities. These velocities are the maximum that each UAV can achieve, enabling the UAV to reach the target as quickly as possible. Because each UAV has the same velocity, the length of each path can determine the order of target visitations. To accurately compare the lengths of the initial paths, they are divided into fixed length segments. Adding an appropriate number of segments to the shorter paths equalizes the path lengths. These segments modify the costs of the paths. However, they do not change the mission completion time. Instead, they organize the target visitations to all occur at the same time.[16]

Then, the paths need to be made flyable. The sharp corners in the paths are smoothed until the vehicles' dynamics are capable of making the turns. The paths are smoothed with a method similar to straightening a chain. Each path segment is represented by a link in the chain. The threats apply repulsive forces to the chain ensuring that the path does not approach too close to the threats. Internal forces in the

chain act to straighten it. This results in a minimal risk, smooth path for the UAVs to follow when attacking their targets.[16]

The McLain, Chandler, et al approach is based on the McLain and Beard method of trajectory generation. Instead of using Dijkstra's cheapest path algorithm, this approach utilizes Eppstein's *k* shortest paths algorithm. It extends the mission of the UAVs by requiring them to travel to a specified location after simultaneously attacking the targets. An estimated time until arrival at the rendezvous location is cooperatively decided upon in a manner that minimizes the threat risk to the entire team. This method may result in one or more UAVs being dismissed from the mission and sent to a predefined location away from the battlefield. A UAV is only sent home when the remaining UAVs can complete all of the tasks in a way that reduces the overall mission time. This method provides a cooperative control algorithm for UAVs attacking predefined targets and rendezvousing at a predetermined location.[15]

When the tasks are not assigned prior to the trajectory generation, a complete approach must be used. The first complete approach uses a hierarchical approach for task allocation and searches for objects, such as targets, within an assigned area, as researched at the Wright-Patterson Air Force Base by Chandler and Pachter. This approach divides the path planning and task allocations into three levels. The top-level team agent must ensure that the mission objectives are met. It must define the objectives and assign the tasks for each sub-team. A sub-team consists of the number of UAVs to accomplish a particular task. The mid-level sub-team agent assigns tasks to individual UAVs. These tasks may include target verification, attacking the target, battle damage assessment, and rendezvous coordination, among others. The lower level vehicle agent contains

information about the terrain, threats, and targets. The vehicle agent is responsible for path planning and trajectory generation for each vehicle, independently. After an assignment algorithm has defined the sub-teams, new targets and vehicles are allocated to these teams using a market analogy based assignment algorithm.[5]

This approach has no specified leader. The algorithms must be contained on each vehicle, enabling all vehicles to arrive at the same decisions using the same, shared information. This redundancy makes the system fault tolerant; if one vehicle is destroyed, the others can continue with the mission.[5]

The simulation for this hierarchical method begins with the vehicles in formation, following a serpentine search pattern. Tasks are assigned as objects are detected. When a vehicle attacks a target, it is destroyed. The UAVs are essentially flying bombs. After targets are attacked and assessed, the remaining vehicles continue the search pattern. The results of these simulations indicate that the sub-team agents reduce the amount of necessary communication, although with some reduction in the optimality of the results. Also, large numbers of vehicles benefit from the market based analogy algorithm when the assignments remain decoupled.[5]

The second complete approach involves partially decoupling the path planning and task allocation of the problem. John Bellingham, Michael Tillerson, Arthur Richards, and Jonathan P. How have developed this approach at Massachusetts Institute of Technology. The basic problem requires a team of UAVs to visit a set of targets while avoiding specified no-fly zones. Path planning and task allocation are strongly coupled. It is difficult to assign tasks without knowing the UAV to target assignments. Similarly, paths cannot be planned until each UAV knows which tasks it is to perform at

which targets. Path planning and task allocation can be partially decoupled. First, rough paths from each UAV's initial location to every target are created. Then, tasks are allocated based on these paths. [3]

The initial paths from the UAV's starting position to every target, while avoiding no-fly zones, are found with a visibility grid. This visibility grid provides straight-line paths from the starting positions through obstacle vertices to the targets. Many permutations exist for a single UAV to visit multiple targets, especially when the visitation order is not specified. To find the shortest paths within the visibility grid, a shortest path algorithm, such as the Floyd-Warshall All-Pairs Shortest Path algorithm, is applied.[3]

These shortest paths are then provided to the task allocation portion of this approach. A multi-dimensional multiple-choice knapsack problem (MMKP) provides a clear, useful method for assigning tasks. One path must be chosen for each vehicle from the available permutations from UAV to targets. The combination of chosen paths must minimize the cost for the mission and follow the appropriate conditions placed on the ordering of targets. These conditions force the MMKP to be solved as a mixed-integer linear program (MILP). The resulting information indicates which target, and in what order, the UAVs are to attack. After the tasks are assigned, the paths become flyable, detailed trajectories.[3]

This approach has the ability to react to a dynamic environment in two ways. The first is a local repair, where the vehicle that detects the change adapts to the change. If a new target is discovered, then the target is added to the vehicle's list of tasks. If a new no-fly zone is discovered, then the vehicle modifies its trajectory to avoid the obstacle.

The second reaction is a sub-team solution, where vehicles capable of adapting to the change are included in the reassignment of tasks and/or modification of trajectories. Although this approach provides suboptimal solutions to the path planning and task allocation problem, the partially decoupled approach is not computationally intensive and can quickly provide solutions. It may also exist on multiple vehicles, providing fault tolerant systems.[3]

The third complete approach involves a combined solution for path planning and task allocation, as developed by Richards, How, et al. The problem is formulated in a manner that allows one MILP, using a branch-and-bound algorithm, to solve the path planning and task allocation problems simultaneously. The vehicles are assumed to be flying at a constant altitude with constant speed. The locations of the UAVs' starting points, the no-fly zones, and the targets must be known. Constraints limiting the vehicles paths and capabilities must be defined using linear equations. These constraints include vehicle dynamics, maximum vehicle velocities, maximum forces the vehicle can withstand, collision avoidance, and minimum time trajectories. After carefully defining the constraints, the problem is ready to be solved. The results of the MILP are optimal trajectories to complete the assigned tasks. However, this approach is extremely computationally intensive. [3,18,19,20]

Chapter 2: Partially Decoupled Approach

The partially decoupled approach is investigated due to the results presented in "Coordination and Control of Multiple UAVs[20]" and "Multi-task Allocation and Path Planning for Cooperating UAVs[3]." The basic outline for creating a partially decoupled simulation for path planning and task allocation is described by these papers. This approach generates paths prior to allocating tasks. After allocating the tasks, the paths are improved to reduce the overall mission cost. The process followed to generate paths and assign tasks using a partially decoupled method is explained in this chapter.

2.1 Generating Initial Paths

The first step in this approach is to determine the initial paths to be used when assigning tasks. To do this, certain values must be known: UAV number, starting locations, altitudes, velocities; no-fly zone number, locations, and radii; target number, locations, values, and states; and threat number, locations, effective ranges, probabilities of kill (POK), and states. The simulation user supplies all of this information. The UAVs' altitudes and velocities do not change in this simulation. At this point in the simulation, the states of the targets and threats indicate whether the object is static or dynamic. A static object is known at the beginning of the simulation. A dynamic object appears at a specified time during the simulation. The effective ranges and POK for the threat are dependent on its type. The threats in the partially decoupled approach include[17]:

- KS-19 100mm Antiaircraft Artillery - Range 4000 meters, 40% POK
- SA-7 Grail - Man-Portable SAM - Range 5000 meters, 50% POK
- Crotale SAM - Range 10,000 meters, 80% POK
- SA-2 - Range 30,000 meters, 80% POK

The path planning portion requires an initial path generation. These initial paths need to completely avoid the no-fly zones, and minimize risk associated with the threats. A Voronoi diagram, as shown in McLain and Beard's work, provides a simple way to create paths while avoiding threats and no-fly zones. To create a Voronoi diagram, a set of nodes to be avoided are located on a plane[22]. Polygons are then created around each node using Delaunay triangulation[22]. The vertices of the polygon are closer to its central node than to any other node[22]. This produces Voronoi edges that are halfway between the two closest nodes[22]. The Voronoi diagram used in the initial path planning is based on the known threat locations, no-fly zone locations[3,16], and nodes placed around the outer edge of the battlefield. There are four nodes, equally spaced, on each edge of the rectangular battlefield. These nodes are incorporated because the number of threats and no-fly zones do not provide enough nodes to generate a useful Voronoi diagram. These nodes also guarantee a path that does not intersect any threats or no-fly zones exists around the outer edge of the battlefield. The UAV starting locations and the target locations must then be connected to the diagram. These locations are connected to the three respective closest nodes[16]. Figure 1 shows a Voronoi diagram, with the vehicles and threats connected to it.

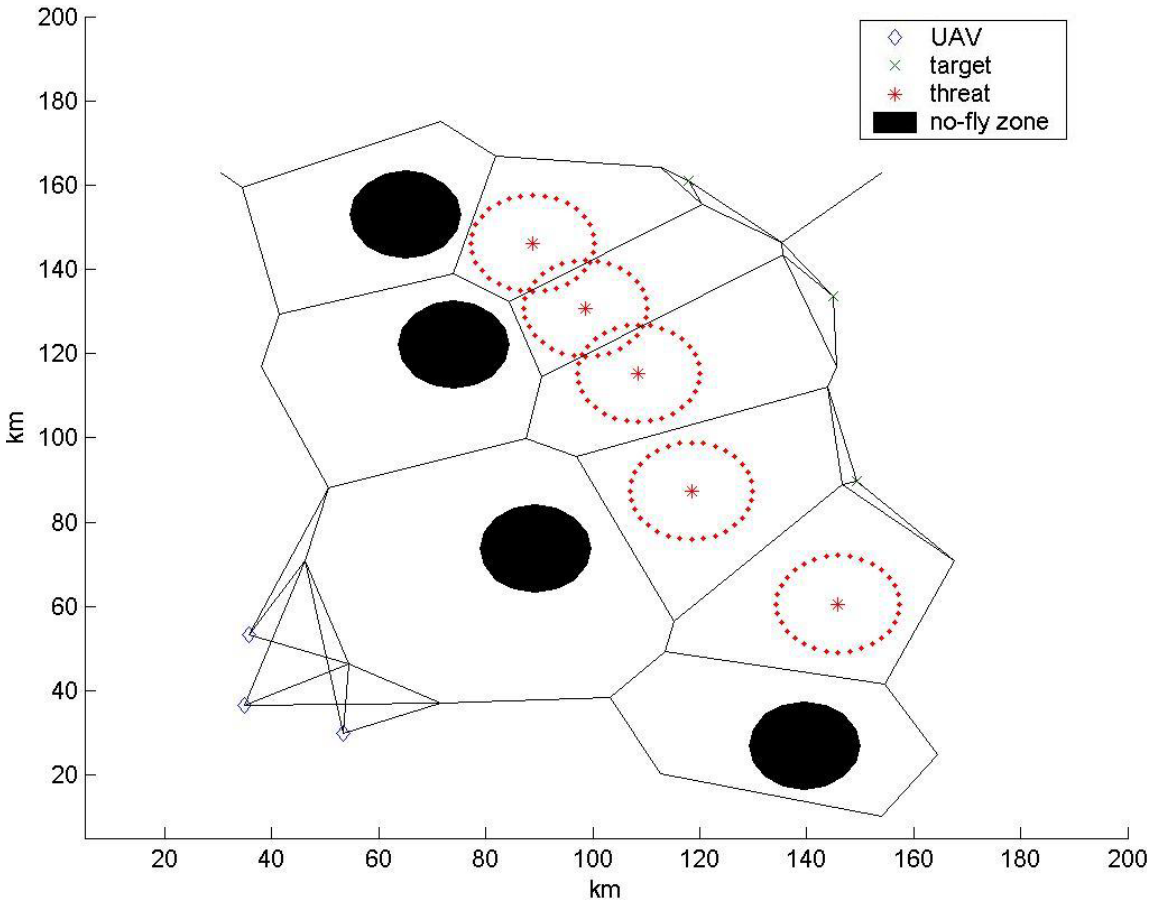


Figure 1: Voronoi diagram based on UAVs, targets, threats, and no-fly zones

Next, the cheapest paths for all of the permutations of UAVs to targets need to be determined. Several shortest path algorithms were reviewed, including Floyd-Warshall All-Pairs Shortest Path algorithm, Eppstein's *k* shortest paths algorithm, Bellman-Ford algorithm, and Dijkstra's shortest path algorithm. Dijkstra's algorithm is chosen for this simulation due to its simplicity and the availability of code for use with MATLAB, written by Michael G. Kay[12]. This algorithm requires a directed graph with positive weights, or costs, associated with each segment[4,10]. The starting node (UAV) and finishing node (target) must be specified[4,10]. The path's cost is the summation of each segment's weight[10]. The algorithm works by beginning at a UAV node, selecting the cheapest path segment, moving to the next vertex, selecting the cheapest path segment, and repeating

until the appropriate target node is reached[4,9,10]. Before Dijkstra's algorithm can be applied, the direction and cost for each path segment must be defined. This direction indicates the tail and head of each segment. The vehicles only travel from the tail to the head.

A simple example of Dijkstra's algorithm is illustrated by Figure 2. The starting node is A, while the finishing node is D. The cheapest path from A to D is determined by looking at the path from A directly to D, with a cost of 10, and the path from A to B to C to D, with a cost of 6. The resulting cheapest path is from A to B to C to D.

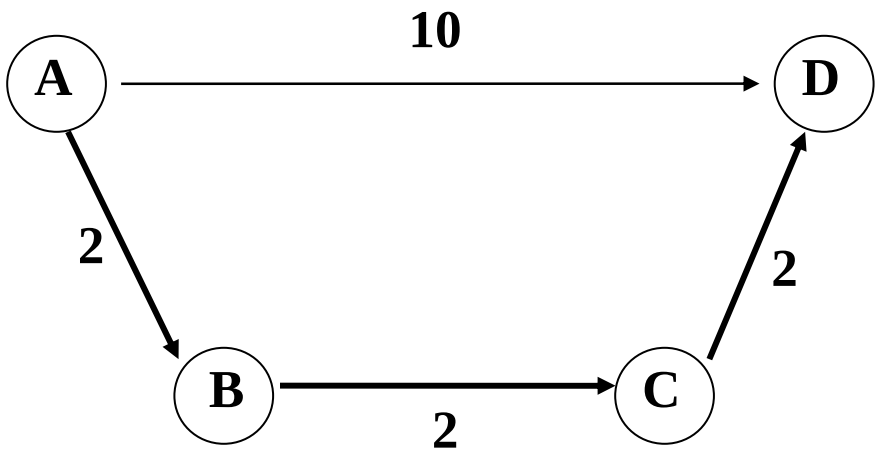


Figure 2: Example of Dijkstra's algorithm

For the UAV path planning, the vehicles may travel in either direction along a segment. Therefore, all edges are specified in both directions. Since all of the UAVs are assumed to be flying at constant speeds, the length of each edge represents how long it will take to travel that distance and represents fuel consumption. Therefore, the costs

representing time and fuel consumption are comparable. In this approach, costs are generated for each Voronoi edge, representative of fuel consumption. Each edge's cost is determined using Equation 1,

$$\text{cost} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

where x_1 and y_1 indicate the x- and y-location of the beginning of the edge and x_2 and y_2 indicate the x- and y-location of the end of the edge.

Then, the costs of the edges are updated to include the costs for intersecting no-fly zones and threats. The cost due to a collision with a no-fly zone is described in Equation 2.

$$\text{cost}_{\text{NFZ}} = 1 \times 10^{30} * \text{cost}_{\text{fuel}} \tag{2}$$

This makes the cost of traveling along a segment that intersects a no-fly zone so expensive that it is never chosen. The cost for intersecting a threat and/or its effective range is related to the POK, as shown in Equation 3.

$$\text{cost}_{\text{threat}} = \text{POK} * 100 + \text{cost}_{\text{fuel}} \tag{3}$$

The POK is a percentage value associated with the particular type of threat. The total cost for each segment is then defined as Equation 4.

$$\text{cost}_{\text{total}} = \text{cost}_{\text{fuel}} + \text{cost}_{\text{NFZ}} + \text{cost}_{\text{threat}} \tag{4}$$

At this point, Dijkstra's algorithm is implemented to find the cheapest paths for each permutation from UAV to targets. Figure 3 contains the cheapest paths from the Voronoi diagram.

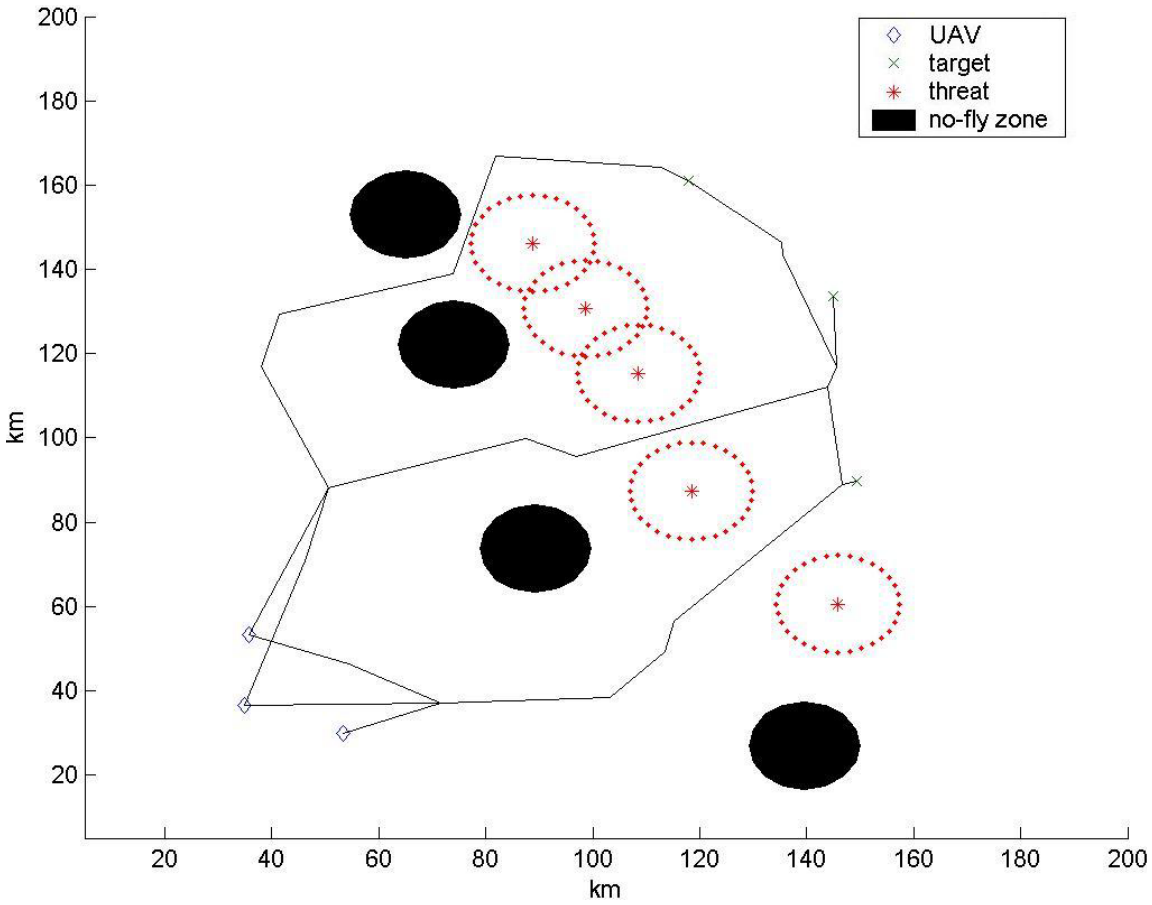


Figure 3: The selected paths from the Voronoi diagram

Dijkstra's algorithm produces the cheapest, but not necessarily the best, paths that a UAV could take to reach its target. Upon inspection, the Voronoi diagram gives paths that tend to have many unnecessary turns. To eliminate these excess turns, line-of-sight paths are investigated. The Voronoi edges are first divided into ten segments, to provide more nodes used for path shortening. The nodes are located at the vertices of the path segments. Starting from a UAV and going directly to a target, a straight line is drawn and checked for intersections with threats and no-fly zones. If the line does not intersect any threats or no-fly zones, then it becomes the new path for this particular permutation. If a threat or no-fly zone is intersected, then a straight line from the UAV to the node prior to the target is drawn and checked for intersections. If no intersections exist, this

straight line is the new path. Otherwise, a straight line is drawn from the UAV to the next previous node. This process is repeated until a straight line is found without any intersections. The node corresponding to the end of this straight line becomes the new starting point. Straight lines from this new starting point to the target are checked for intersections following the same logic as from the UAV to the target. This process is complete when the target is reached. The results are shorter, simpler paths from the UAVs to the targets.

These shortened paths may still have some sharp corners that violate the minimum turning radius of the UAV, making them impossible for the UAVs to follow. To make the paths flyable, fillets are placed in the corners that are too sharp. These fillets have a radius equivalent to the minimum turning radius of the UAV. The fillet is placed as close to the vertex as possible, using the law of cosines to determine where the fillet intersects the existing path. The fillet then becomes part of the path, replacing the corner. The UAV must correct its heading angle to follow the changes in direction from these paths. The simulation ensures that the heading angle is recorded and properly modified to allow for all turns required by the mission. Figure 4 shows the shortened paths from each UAV to each target.

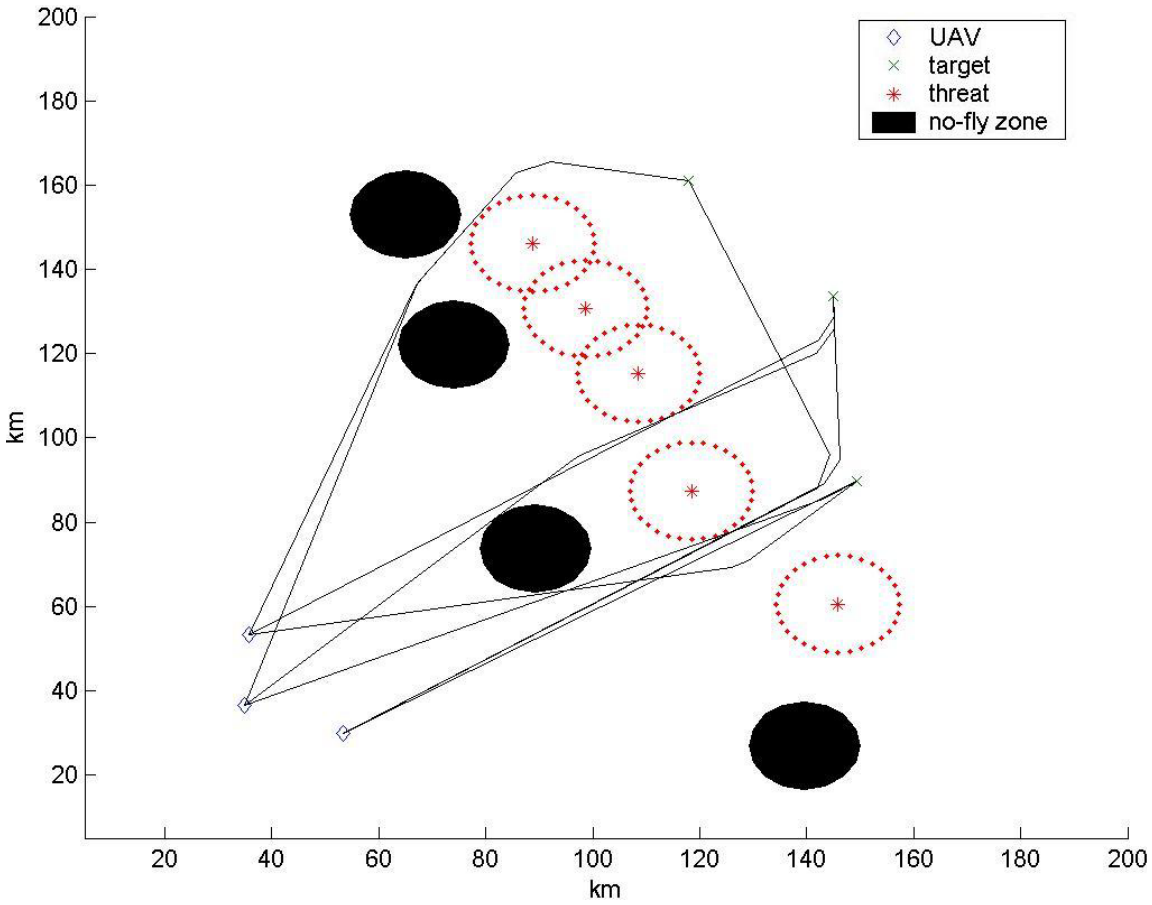


Figure 4: Shortened paths for each UAV to target permutation

Most paths change significantly from those produced by the Voronoi diagram. For this reason, the costs of the paths are updated at this point to accurately reflect the shortened, smoothed paths. Knowing the costs for the best paths from UAVs to targets, the tasks can now be assigned.

2.2 Task Allocation

In [3], the partially decoupled problem formulates its task allocation problem as a multi-dimensional multiple-choice knapsack problem (MMKP). The MMKP is a classic problem that requires one item, having a value and a resource requirement, to be chosen from each group. Several groups exist, each containing several items. The value

indicates the benefit of choosing a particular item, while the resource indicates a restriction on which combination of items may be chosen. The combined value of the items chosen from the groups must be maximized while adhering to the resource constraints.[2,3,13]

The UAV task allocation problem presented here has a number of groups equal to the number of UAVs. Each group consists of the paths corresponding to a particular UAV to each of the targets. A constraint on this problem says that a vehicle may only have one path assigned to it. Also, every target must have a vehicle assigned to it. Every target is visited, even if the importance of visiting a particular target is much greater than that of another target. The goal of the problem is to minimize the total cost of the mission.

A simple example of how the MMKP works involves three vehicles and three targets. The data given to the MMKP is displayed in Table 1.

Table 1: MMKP example costs, in meters, for each vehicle to visit each target

	UAV 1	UAV 2	UAV 3
Target 1	100m	110m	150m
Target 2	280m	225m	250m
Target 3	500m	550m	575m

Each target must be visited, and each vehicle must have a task assigned to it. These constraints must be met, while still finding the cheapest combination of paths and tasks. The resulting combination that provides the lowest cost has UAV 1 visiting target 3, UAV 2 visiting target 1, and UAV 3 visiting target 2. The total cost of this mission is

860 meters. Any other combination of vehicles and targets results in a higher mission cost.

In this simulation, the minimum mission cost is determined by finding all of the permutations from UAV starting locations to targets. Then, the total cost for each permutation is calculated. The permutation with the lowest cost indicates which set of paths will allow the UAVs to accomplish their mission of destroying all targets while minimizing risk and cost. The selected paths with the lowest mission cost are shown in Figure 5.

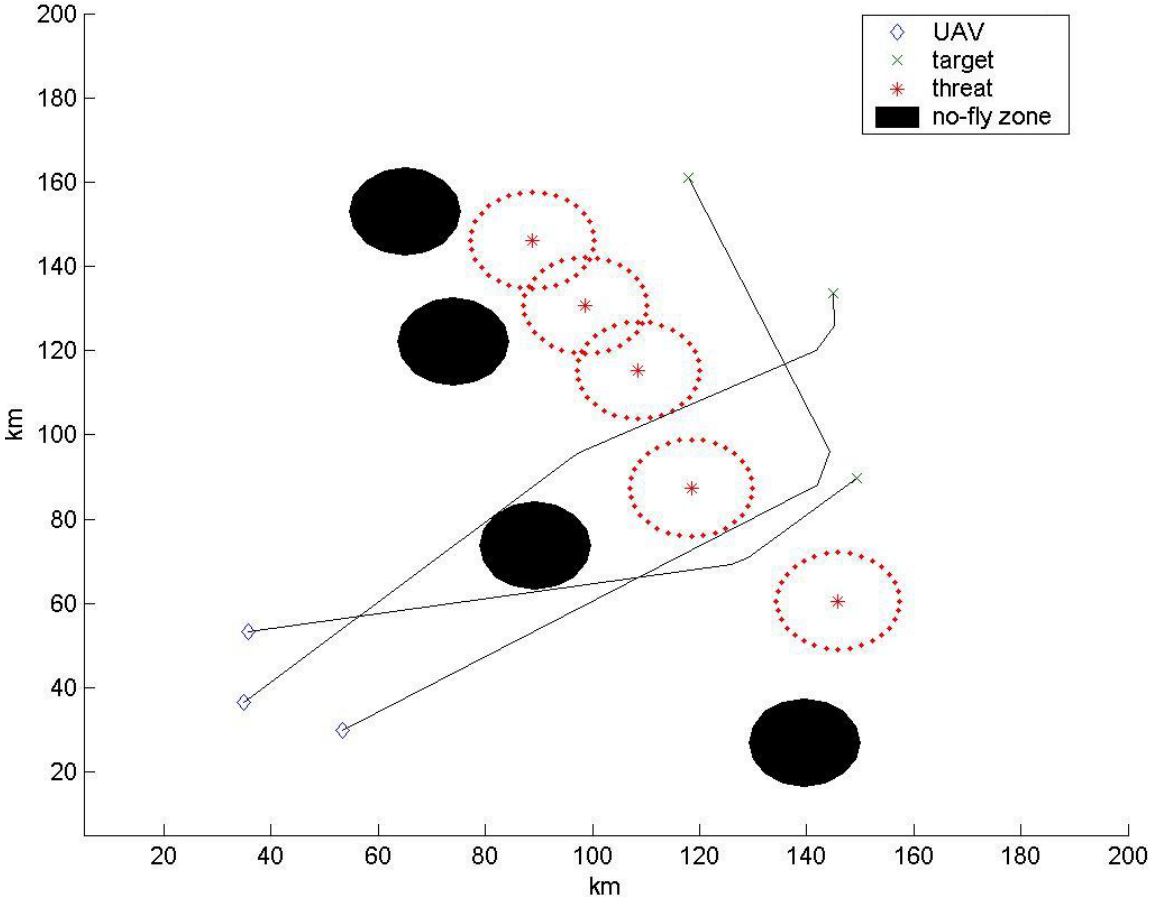


Figure 5: Selected paths with the lowest mission cost

All of the steps up to this point assume that the number of UAVs equals the number of targets and the environment is static. When the number of UAVs is greater

than the number of targets, the simulation places a duplicate target at the same location of the target with the greatest value. The duplicate target is assigned a value of zero, and cannot be classified or destroyed. This target causes a second UAV to visit the most valuable target to help ensure its destruction. Duplicate targets are created until the number of targets equals the UAVs. With each new duplicate target, the values of the original targets are halved. This allows a target that is more than twice as valuable as the next valuable target to have three vehicles sent to it. By doing this, the extra UAVs help destroy the targets in the order of value.

When the number of targets is greater than the number of UAVs, the excess targets with the lowest values are hidden. This is done until the number of targets equals the number of UAVs. When a visible target is destroyed, the most valuable target of those hidden is displayed and involved in the simulation.

The change in visible targets causes a replan to be signaled. When a replan is signaled, the entire program is repeated to recalculate paths and reallocate tasks based on the new information. Other changes in the simulation environment also cause replans. These changes include a UAV crash, a change in the state of a target, and a target or threat appearing.

A UAV crash occurs when it flies into a no-fly zone. A vehicle may also be removed from the simulation when it intersects a threat's effective range. When a UAV enters a threat's range, the threat always fires. The UAV's chance of survival is random, based on a value supplied by MATLAB. The UAV survives when its chance of survival is greater than the threat's POK. Otherwise, it is shot down. Once a threat fires, it is

considered no longer active, and removed from the simulation. If a particular threat is known to fire more than once, the simulation can be modified accordingly.

A target state begins as unclassified. The first time a UAV passes over a target, the target is considered classified. There is a ten percent chance, based on a random number from MATLAB, the target has been previously misidentified and should not be destroyed. If this occurs, then that target is dismissed from the active environment, signaling a replan. If the target is perceived to be real, then its state changes to classified. This also signals a replan. The UAVs know that they are allowed to attack the classified targets. There is a fifteen percent chance, based on a random number from MATLAB, that the target is not destroyed when it is attacked. This is detected during the battle damage assessment of the target. The assessment of the target signals a replan. After a UAV verifies that a target has been destroyed, the target is removed from the active environment.

UAVs that complete the mission of destroying all targets return to the origin of the battlefield. These return paths must still avoid threats and no-fly zones. This represents a specified rendezvous location that the UAVs must reach when they complete their tasks.

This simulation was originally created in MATLAB. Then, when the MATLAB files worked properly, the simulation was transferred to a SIMULINK environment. During the simulation, a list of the events signaling replans and the time at which they occur are displayed in the MATLAB command window. Plots, displaying the locations of the UAVs, threats, no-fly zones, and targets, can appear while the simulation is running. These plots are created each time a replan occurs. The data is also saved so the

simulation may be played, paused, and viewed more than once, upon completion. Data necessary to view the simulation using MATLAB's Virtual Reality Toolbox (VRT) is also saved. After the simulation is complete, the VRT can be used to display it. The main MATLAB codes are included in Appendix A, while the main SIMULINK block diagrams are included in Appendix B.

Chapter 3: Combined Approach

The combined path planning and task allocation problem was approached from the direction of Richards, How, et al's work[18,19,20]. The combined problem, when formulated as a mixed integer linear program, produces optimal solutions. The paths and tasks are planned and allocated concurrently. These solutions can serve as a benchmark to evaluate the performance of suboptimal routines. MILP problems are solved using a variety of methods, including simplex and branch and bound[1,7]. These solution methods apply to many different applications of linear programming, including scheduling and vehicle routing[1]. The simplex solution method is used in this research because it is the method employed by the free student version of AMPL/CPLEX[11].

3.1 Classic Mixed Integer Linear Program (MILP)

A classic example of a simple MILP problem has an objective function that must be minimized (or maximized) subject to several constraints. Equation 5 is an example of an objective function.

$$\min J = c * x \tag{5}$$

J is the utility to be minimized, c is a row vector containing the objective function coefficients, and x is a column vector containing the decision variables for which MILP is solving. The constraints are in the form of Equation 6.

$$A * x \leq b \tag{6}$$

A is the coefficients matrix of the constraints and b is the column vector containing the values that limit the constraints.

The decision variables are constrained by upper and lower bounds, thereby limiting the number of possible solutions. This decreases the amount of computational time required to reach the solution. The path planning and task allocation problem must be formulated in this classical way before it can be solved.

A simple MILP example involves two vehicles and two targets. The specific data given is shown in Table 2.

Table 2: MILP example costs, in meters, for each vehicle to visit each target

	UAV 1	UAV 2
Target 1	10m	20m
Target 2	15m	10m

The specific objective function for this example is shown in Equation 7,

$$\min J = 10x_{11} + 15x_{12} + 20x_{21} + 10x_{22} \tag{7}$$

where $x_{uav\ target}$. The constraints placed on the decision variables are in Equations 8 and 9. Equation 8 enables only one path to be chosen for each vehicle. Equation 9 requires each target to be attacked.

$$\begin{aligned} x_{11} + x_{12} &\le 1 \\ x_{21} + x_{22} &\le 1 \end{aligned} \tag{8}$$

$$\begin{aligned} x_{11} + x_{21} &= 1 \\ x_{12} + x_{22} &= 1 \end{aligned} \tag{9}$$

Equations 8 and 9 are written in such a way that the decision variables, x, are binary variables. This creates a simple integer linear problem, as opposed to a mixed integer linear problem. A mixed integer problem builds on this basic integer problem, by adding

decision variables whose values will not be integer. The solution to this example has UAV 1 visiting target 1 and UAV 2 visiting target 1, with a cost of 20 meters.

3.2 Path Planning and Task Allocation MILP

The goal of the path planning and task allocation MILP is to minimize the overall mission completion time. Before the objective function can be explicitly written, the constraints on the system must be defined. The problem begins with the following items[18,19,20] being specified by the simulation user:

- number of vehicles, N_v
- number of targets, N_w
- number of time steps, N_t
- mass of each vehicle, *mass*
- maximum internal force of the vehicles, f_{max}
- maximum velocity of the vehicles, v_{max}
- maximum size of the battlefield, $dist_{max}$
- starting locations of the UAVs, (x_{p0}, y_{p0})
- initial vehicle velocities, $(\dot{x}_{p0}, \dot{y}_{p0})$
- target locations, (W_{i1}, W_{i2})
- size and location of the no-fly zones, $(Z_{j1}, Z_{j2}, Z_{j3}, Z_{j4})$

The first constraint on this system represents the vehicle dynamics. The dynamics determine the positions, (x_{pt}, y_{pt}), and velocities, $(\dot{x}_{pt}, \dot{y}_{pt})$, of vehicle *p* at time *t*. These values are incorporated into a state vector, s_{pt}, as shown in Equation 10[18,19,20].

$$s_{pt} = \left[x_{pt}\ y_{pt}\ \dot{x}_{pt}\ \dot{y}_{pt}\right]^T \tag{10}$$

The inputs that affect the state vector are control forces, f_{pt}, represented by Equation 11[18,19,20].

$$f_{pt} = \left(f_{x_{pt}}, f_{y_{pt}}\right) \tag{11}$$

Together, the state vector and the force vector create the state space model of the system. This constraint is shown in Equation 12[18,19,20].

$$\begin{aligned} &\forall p \in [1 \ldots N_v], \forall t \in [0 \ldots N_t + 1] \\ &s_{p(t+1)} = As_{pt} + Bf_{pt} \end{aligned} \tag{12}$$

The vehicles are approximated as point masses moving at constant altitudes. The matrices *A* and *B* are known from previous work on point mass dynamics[14,18]. They are represented in Equation 13[14,18] for the continuous system.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{mass} & 0 \\ 0 & \frac{1}{mass} \end{bmatrix} \tag{13}$$

For use in Equation 12, and this simulation, the matrices of Equation 13 must be discretized. The discretization is done using the MATLAB command "c2d." The decision variables in Equation 12 are s_{pt} and f_{pt}. The initial conditions, s_{p0}, must be defined by the user.[18]

The aircraft's velocities and the forces on it are limited by their maximum values. These constraints should be nonlinear equations. However, to maintain the linear formulation, some approximations are made. To determine the magnitude of the total velocity, or force, it is necessary to combine the x and y components. The magnitude constraint should be a complete, continuous circle. A necessary approximation requires the circle to be divided into *M* segments. The locations of the segment vertices around the circle are represented with sine and cosine terms. The velocity and force magnitude constraints can now be written as in Equations 14[16,18,19] and 15[16,18,19].

$$\forall t \in [1 \ldots N_t], \forall p \in [1 \ldots N_v], \forall m \in [1 \ldots M]$$
$$\dot{x}_{pt} \sin\left(\frac{2\pi m}{M}\right) + \dot{y}_{pt} \cos\left(\frac{2\pi m}{M}\right) \leq v_{\max} \tag{14}$$

$$\forall t \in [0 \ldots N_t - 1], \forall p \in [1 \ldots N_v], \forall m \in [1 \ldots M]$$
$$f_{x_{pt}} \sin\left(\frac{2\pi m}{M}\right) + f_{y_{pt}} \cos\left(\frac{2\pi m}{M}\right) \leq f_{\max} \tag{15}$$

The sine and cosine terms represent nonlinearities. To avoid this problem, the sine and cosine terms are reduced to constant numerical values prior to writing the constraints in the model file.[16,18,19]

Next, the UAVs must be concerned with avoiding no-fly zones. The UAV's position must not be within the boundaries of a no-fly zone at any time *t*. For simplicity, the no-fly zones are modeled as rectangles. The lower left vertex and upper right vertex are represented by (Z_{j1}, Z_{j2}) and (Z_{j3}, Z_{j4}), respectively. The no-fly zone avoidance constraint is described by Equation 16[18,20].

$$\begin{aligned}
&\forall t \in [1 \ldots N_t], \forall p \in [1 \ldots N_v], \forall j \in [1 \ldots N_z] \\
&x_{pt} - Z_{j3} \geq -Rd_{j1pt} \\
&Z_{j1} - x_{pt} \geq -Rd_{j2pt} \\
&y_{pt} - Z_{j4} \geq -Rd_{j3pt} \\
&Z_{j2} - y_{pt} \geq -Rd_{j4pt} \\
&\sum_{k=1}^{4} d_{jkpt} \leq 3
\end{aligned} \tag{16}$$

In addition to x_{pt} and y_{pt}, d_{jkpt} is a decision variable. Unlike the previous decision variables, d_{jkpt} is a binary variable equaling one if the vehicle is potentially within the boundaries of a no-fly zone. The k represents the four directions: $(1 = +X, 2 = -X, 3 = +Y, 4 = -Y)$. The R in Equation 16 helps relax the constraint if d_{jkpt}

equals one. The value of R is a positive number much greater than any position or velocity to be encountered in the problem.[18] The summation of the d_{jkpt}'s ensures that no more than three of the constraints are relaxed.[18,20]

The UAVs' mission is to visit known targets. To do this, the position of a UAV must exactly equal that of a target at a particular time step. The location of a target is defined as (W_{i1}, W_{i2}). Equation 17[18,19,20] shows how this target constraint is defined.

$$\begin{aligned} &\forall p[1\ldots N_v], \forall t \in [1\ldots N_t], \forall \in [1\ldots N_z] \\ &x_{pt} - W_{i1} \le R(1 - b_{ipt}) \\ &x_{pt} - W_{i1} \ge -R(1 - b_{ipt}) \\ &y_{pt} - W_{i2} \le R(1 - b_{ipt}) \\ &y_{pt} - W_{i2} \ge -R(1 - b_{ipt}) \end{aligned} \tag{17}$$

Similar to the partially decoupled method, each UAV is capable of visiting every target. Another binary decision variable, b_{ipt}, is introduced. If vehicle p visits target i at time step t, then b_{ipt} equals one. R is the same number as used in the no-fly zone constraint, Equation 16. This target constraint requires the vehicles to pass directly over the target. If desired, this constraint may be relaxed to allow the vehicle to pass within a certain distance of the target.[18,19,20]

While Equation 17 indicates when a target has been reached, it does not explicitly require this visitation. Equation 18 specifies that each target must be visited.

$$\begin{aligned} &\forall p \in [1\ldots N_v], \forall t \in [1\ldots N_t], \forall i \in [1\ldots N_w] \\ &\sum_{p=1}^{N_v} \sum_{t=1}^{N_t} b_{i,p,t} = g \end{aligned} \tag{18}$$

The integer value g indicates the number of times the targets must be visited, as required to complete the mission. When g equals one, the targets are simply visited. To classify, attack, and assess a target, g is set to three.

Next, it is necessary to know at which time step a target is visited. To do this, a constraint as shown in Equation 19[18,20] is added.

$$\forall p \in [1 \ldots N_v], \forall i \in [1 \ldots N_w]$$
$$t_p \geq \sum_{t=1}^{N_t} t b_{ipt} \tag{19}$$

The additional variable, t_p, represents the time step at which vehicle p visits target i. This constraint ensures that t_p is the time at which the last target is visited.[18,20]

The overall mission complete time, $\bar{t}$, must be determined. This value must be greater than or equal to the longest completion time among the vehicles. Equation 20[18] shows this constraint.[18,20]

$$\forall p \in [1 \ldots N_v]$$
$$\bar{t} \geq t_p \tag{20}$$

After formulating these constraints, the objective function is written. The goal of the objective function is to minimize the overall mission completion time. Equation 21[18,20] explicitly states the objective function.

$$\min_{s,f,b,d} J = \bar{t} \tag{21}$$

The decision variables *s*, *f*, *b*, and *d* are the values that determine t_p, which directly affects $\bar{t}$.[18,20]

Solution aids can be added to this objective function as shown in Equation 22[18,20].

$$\min_{s,f,b,d} J = \bar{t} + \varepsilon_1 \sum_{p=1}^{N_v} t_p \tag{22}$$

The small weighting factor ε_1 decreases the computation time. This weighting factor requires the minimum time path to be chosen for each vehicle. If this factor is omitted, then only the vehicle with the longest finishing time will be directly minimized.[18,20]

After specifying the upper and lower bounds for all of the decision variables, the MILP is ready to be solved. The resulting values for the decision variables indicate the paths and tasks assigned to the UAVs. The objective function value is calculated based on the values of the decision variables.

Following the recommendations of [3], [6], [19], and [20], the combined problem is formulated for use with AMPL/CPLEX. The student version of AMPL/CPLEX has limitations of 300 decision variables and 300 constraints. The constraints, upper and lower bounds, and objective function are written in a model file. The input data, such as N_v, N_w, N_t, N_z, and locations, are included in a data file. These files are developed following the examples and descriptions provided by *AMPL, A Modeling Language for Mathematical Programming*[8]. The results are plotted in MATLAB to visualize the assigned paths and tasks. The code for the model file is included in Appendix C and the data file is included in Appendix D.

Chapter 4: Comparison between the Partially Decoupled and Combined Approaches

4.1 Simulation Parameters Defined

A comparison of the partially decoupled approach and the combined approach yields information on how the approximations used to decouple the path planning and task allocation degrade the optimality of the solution. For an accurate comparison, the input data provided to each simulation must be the same. The common input data consists of the number and starting positions of the UAVs; the number and locations of the targets; and the number, locations, and sizes of the no-fly zones. Additionally, the partially decoupled simulation requires a threat, its effective range, probability of kill, state, and location, and the targets' values and states. Because threats do not exist in the combined approach due to constraint limitations, a single threat is located outside the edge of the battlefield and given a POK of zero. This is done to minimize the effect the threat has on the scenario. It influences the Voronoi diagram slightly, but not the costs of the path segments. The combined simulation is not capable of giving targets any type of value or precedence; therefore, all target values are set to 100 in the partially decoupled simulation. The partially decoupled simulation is stopped after the first path planning and task allocation assignment, due to the inability of the combined approach to react to dynamic situations. Accordingly, the targets are only visited once.

The combined approach additionally requires the UAVs' starting velocities, maximum velocities, maximum force loadings, and masses to be inputs. The combined approach is based on no-fly zones that are rectangular, rather than the circles used in the partially decoupled approach. To match the circular no-fly zones of the partially

decoupled method, three rectangles are layered on top of each other to approximate a circle. Equation 16 was not modified to represent a circle due to the excessive increase in constraints. Figure 6 shows how rectangles placed inside a circle can approximate it.

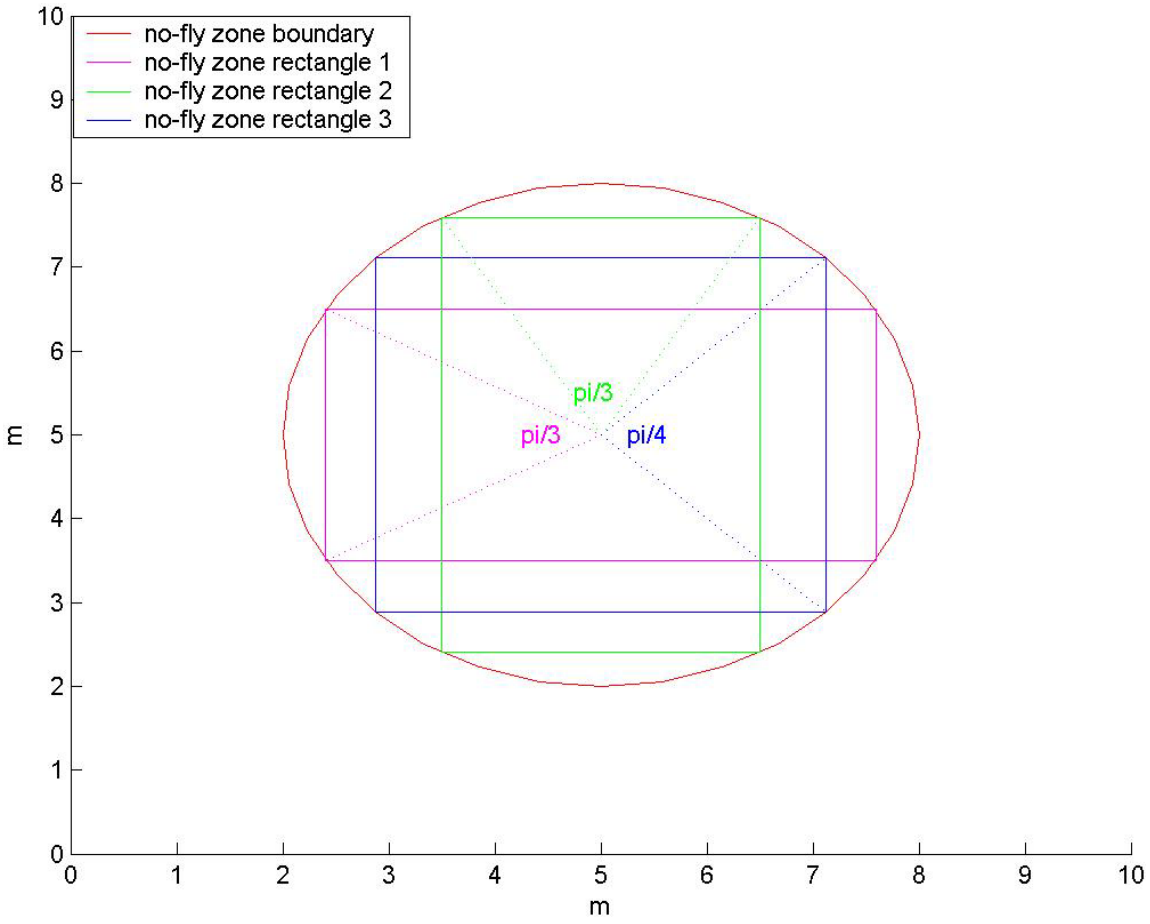


Figure 6: Circular no-fly zone approximated with rectangles

These rectangles have angles chosen such that the difference in area covered by the rectangles is about 10% less than the area covered by the circle.

4.2 Results from the Comparison

For this comparison, five scenarios are investigated. They involve altering the initial locations of the vehicles. The limits imposed by the student version of AMPL/CPLEX restrict the scenarios to two vehicles, two targets, and one no-fly zone.

Additionally, in the combined method, only three time steps are available. The battlefield must be small enough to allow the vehicles to avoid the no-fly zones and reach the targets in three time steps, each time step equaling one second. The scenarios are designed for a 50 m x 50 m battlefield. The initial locations for the first scenario can be seen in Figure 7. The partially decoupled simulation additionally has a threat located at (55, 55) m.

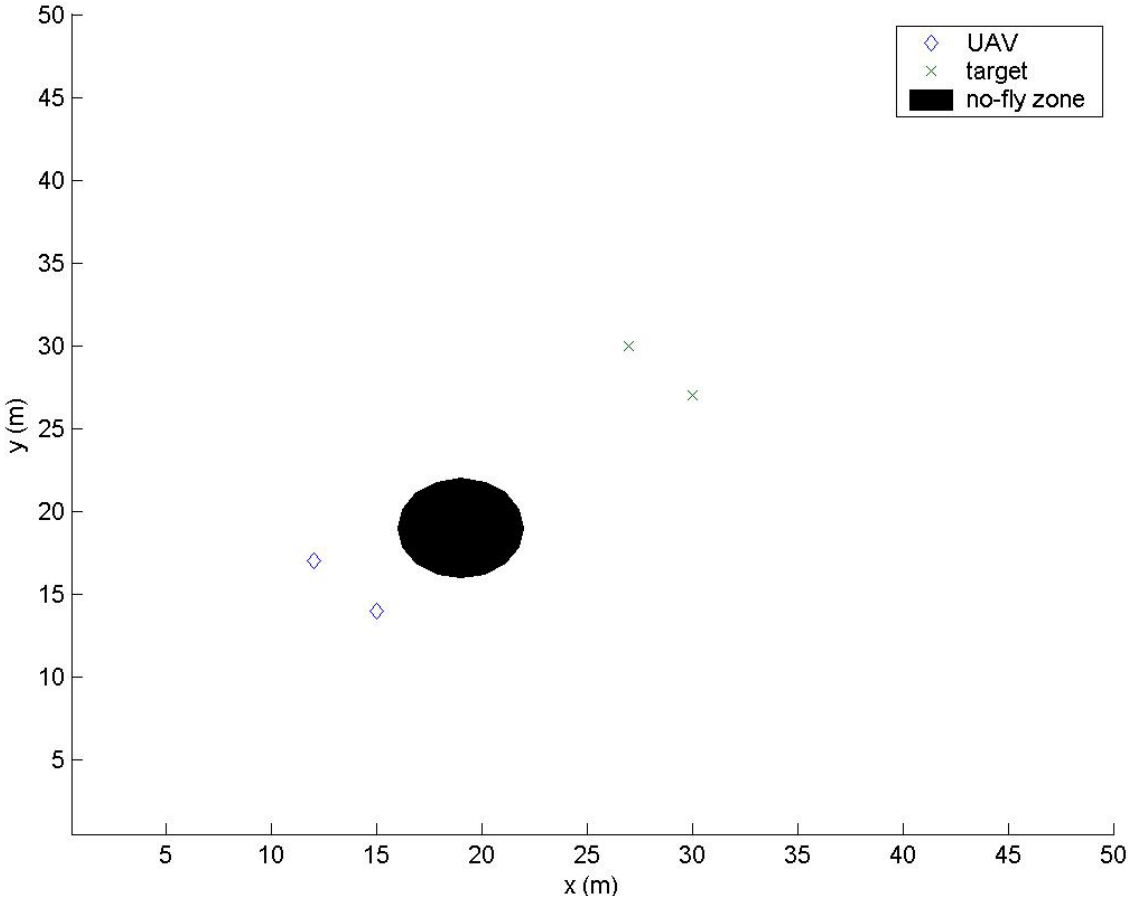


Figure 7: Initial setup of the battlefield, scenario 1

This method has a constant vehicle speed of 13.41 m/s, and a minimum turning radius of 6.5 m. The combined approach has the UAVs' limiting velocities set at ± 13.41 m/s, limiting allowable forces of ± 2.35 N, and masses of 0.085 kg[17].

After entering the data using the graphical user interface for the partially decoupled approach, the simulation is executed until the first path planning and task

allocations are complete. At this point, the data is saved and plotted. The chosen paths and assigned tasks are shown in Figure 8.

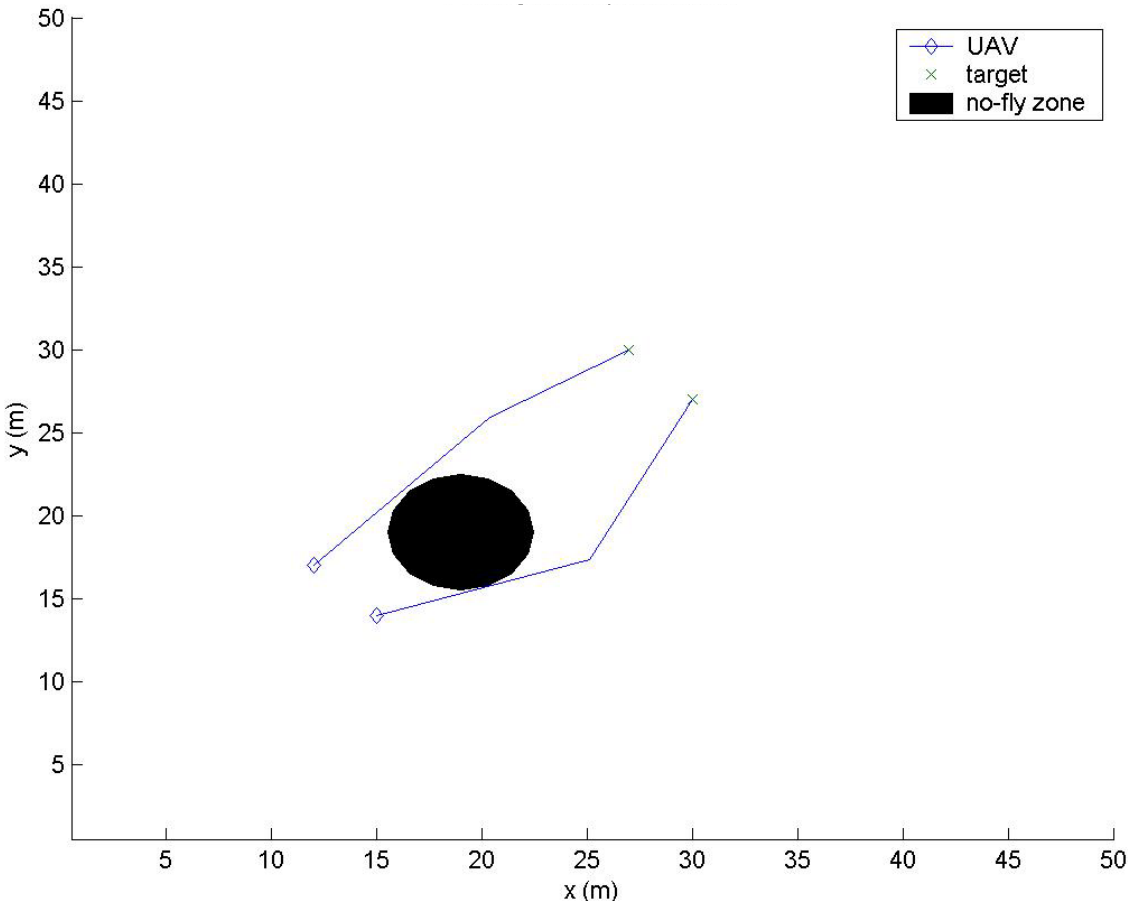


Figure 8: Results for scenario 1 using the partially decoupled method

A data file containing the input information is created for the combined approach. The input information is the same as used for the partially decoupled approach. Also, the number of segments, *M*, from Equations 14 and 15, for approximating a circle, is set to ten. The relaxation constant, *R*, from Equations 16 and 17 has a value of 100,000. The weighting factor, ε_1, equals 0.001. These values are specified in the data file. Upon the completion of the combined simulation, a plot is created using the locations of the vehicles at each time step. Figure 9 displays the paths the vehicles take to reach the targets.

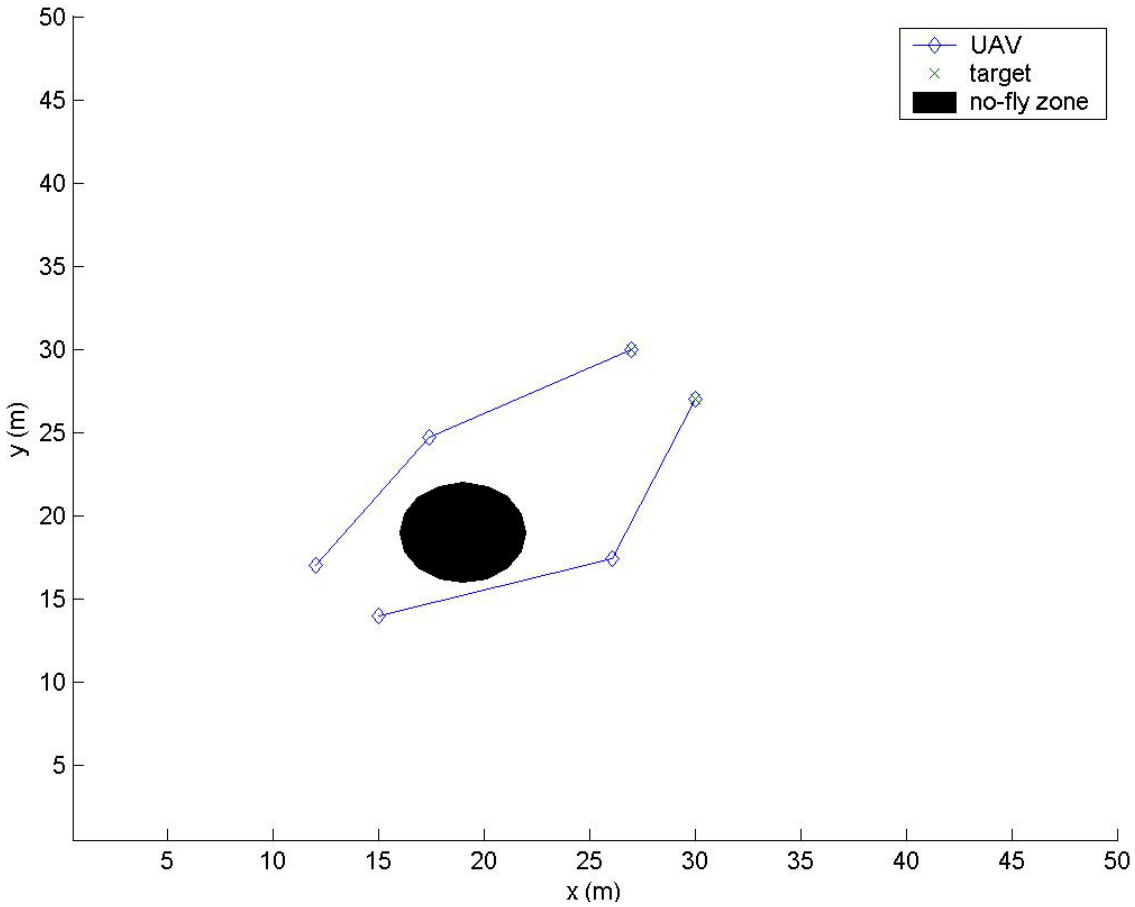


Figure 9: Results for scenario 1 using the combined method

Scenarios two through five have the same parameters as scenario one, except for the initial vehicle location and velocity. The changes in the locations affect the Voronoi diagrams, and the initial direction the UAVs need to be going, hence, modifying the initial velocity. The initial locations for scenario two are shown in Figure 10.

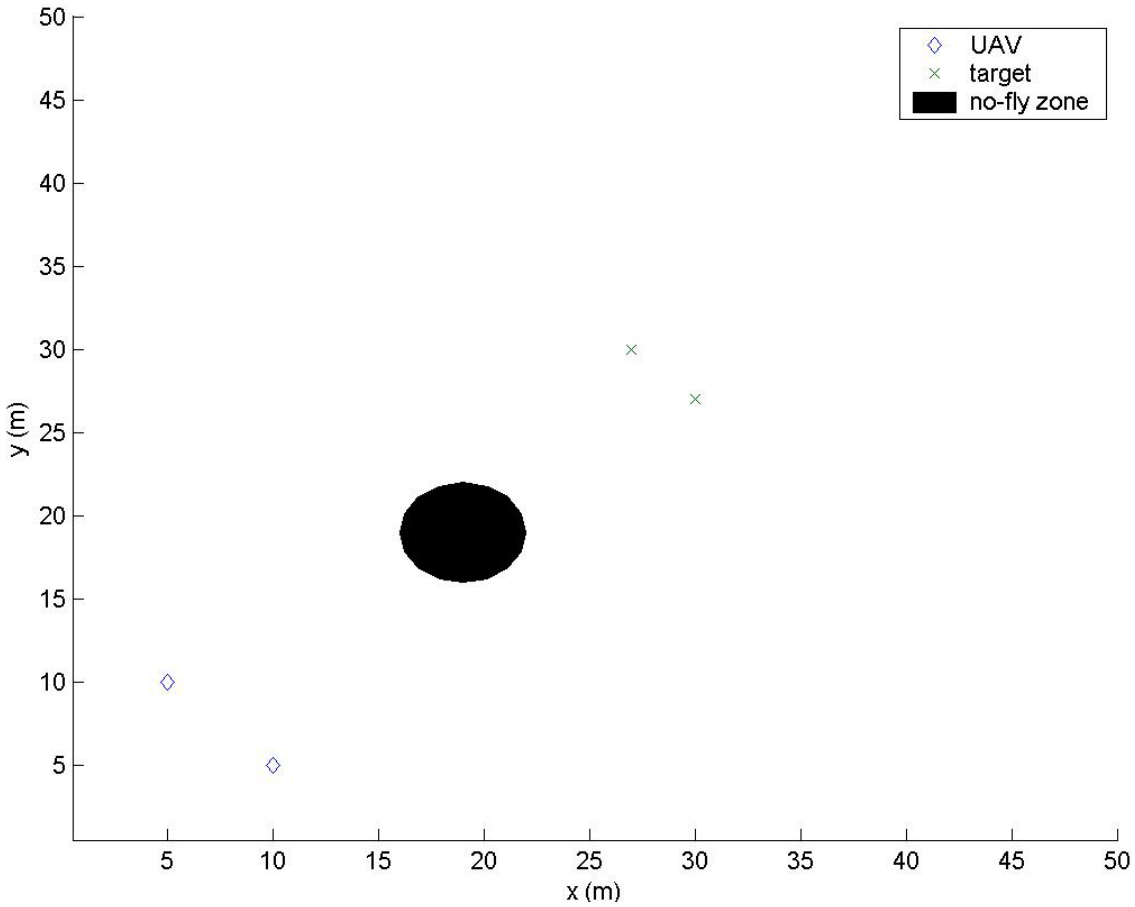


Figure 10: Initial setup of the battlefield, scenario 2

The results from the partially decoupled method are displayed in Figure 11, while the combined method results are in Figure 12.

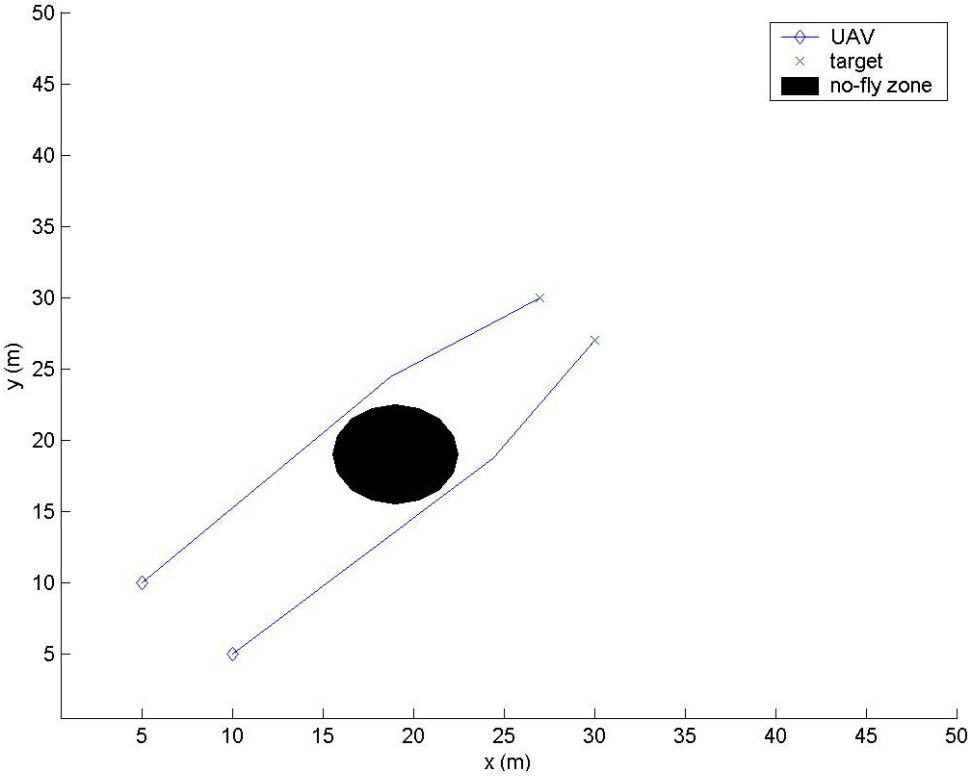


Figure 11: Results for scenario 2 using the partially decoupled method

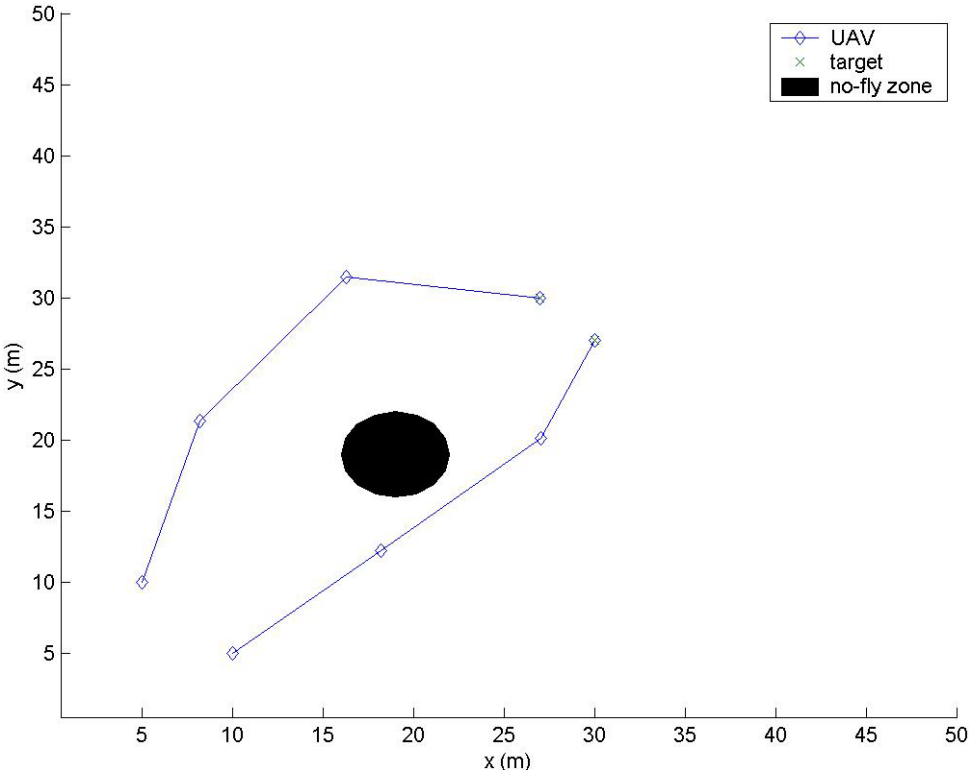


Figure 12: Results for scenario 2 using the combined method

The initial locations for scenario three are shown in Figure 13.

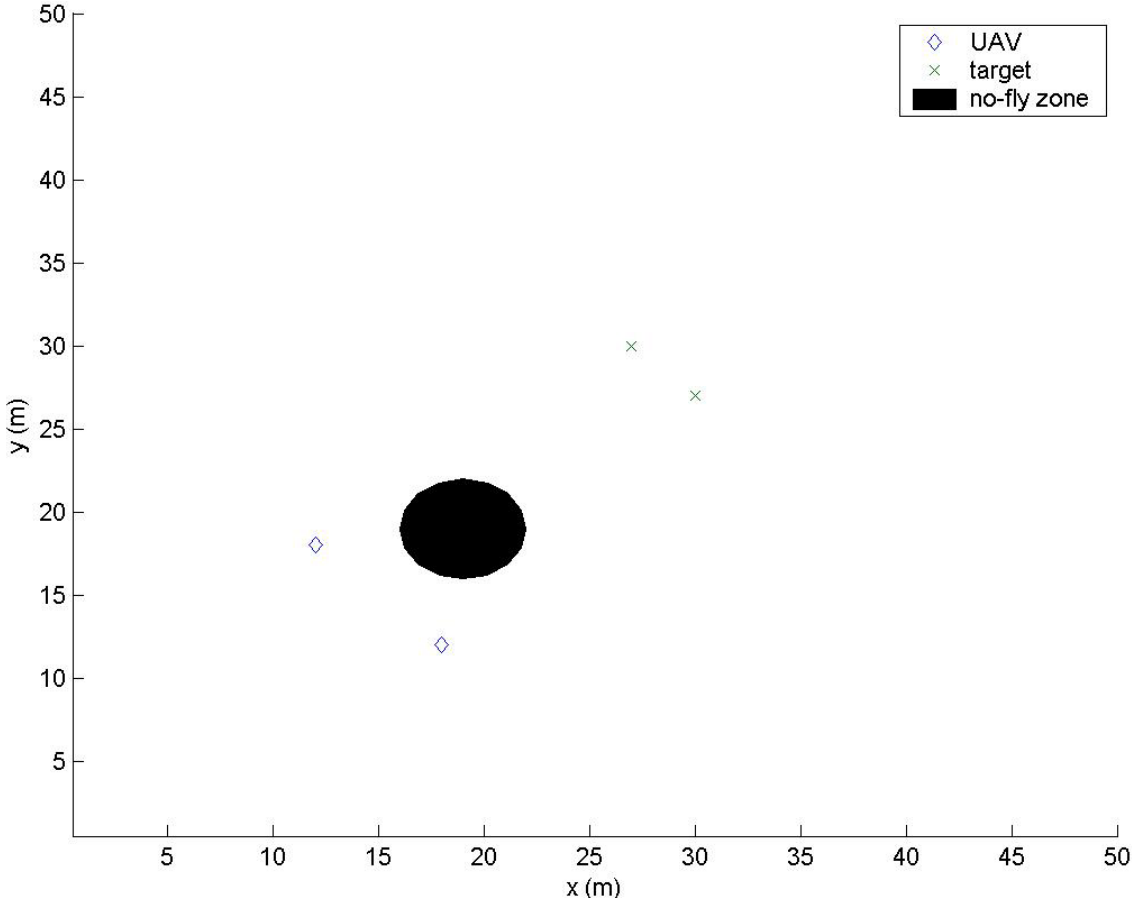


Figure 13: Initial setup of the battlefield, scenario 3

The results from the partially decoupled method are displayed in Figure 14, while the combined method results are in Figure 15.

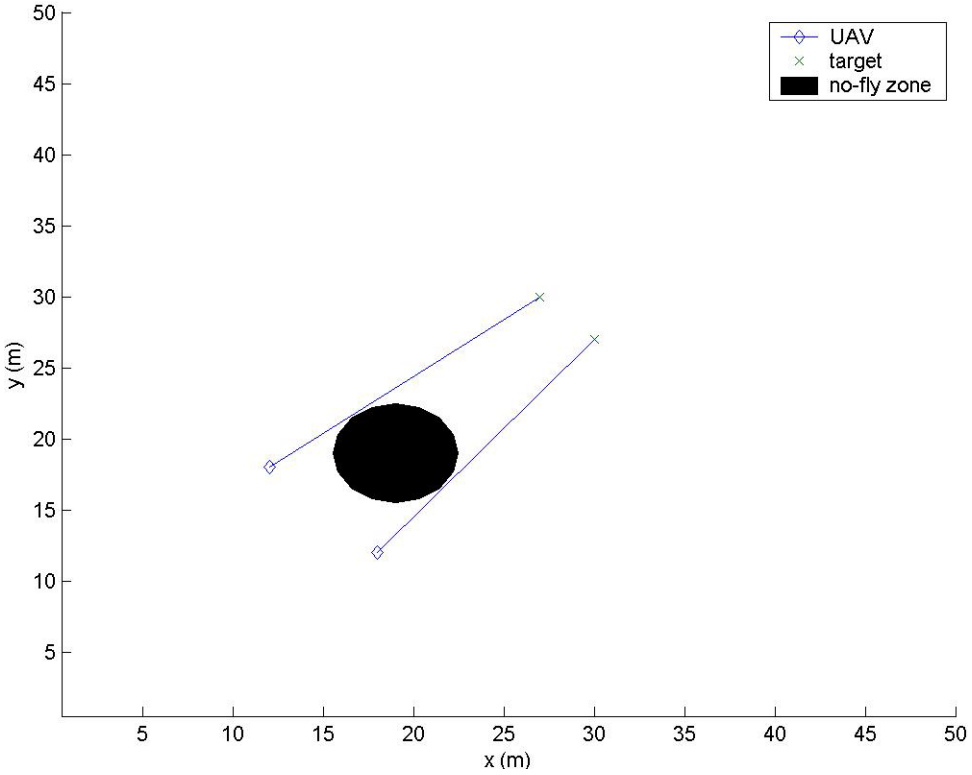


Figure 14: Results for scenario 3 using the partially decoupled method

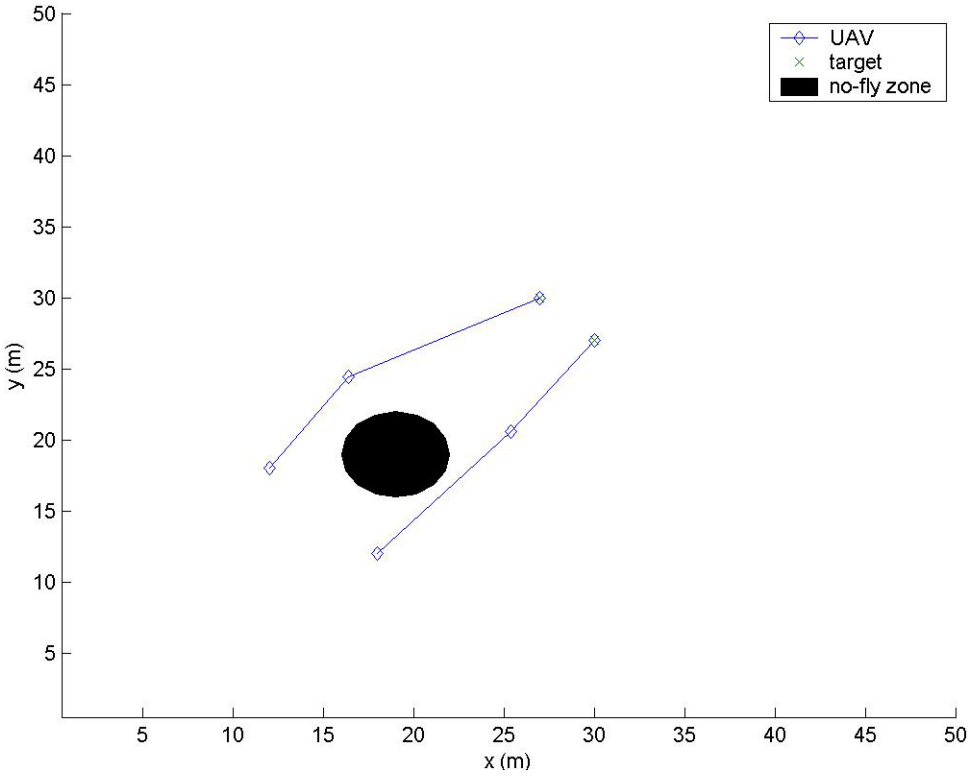


Figure 15: Results for scenario 3 using the combined method

The initial locations for scenario four are shown in Figure 16.

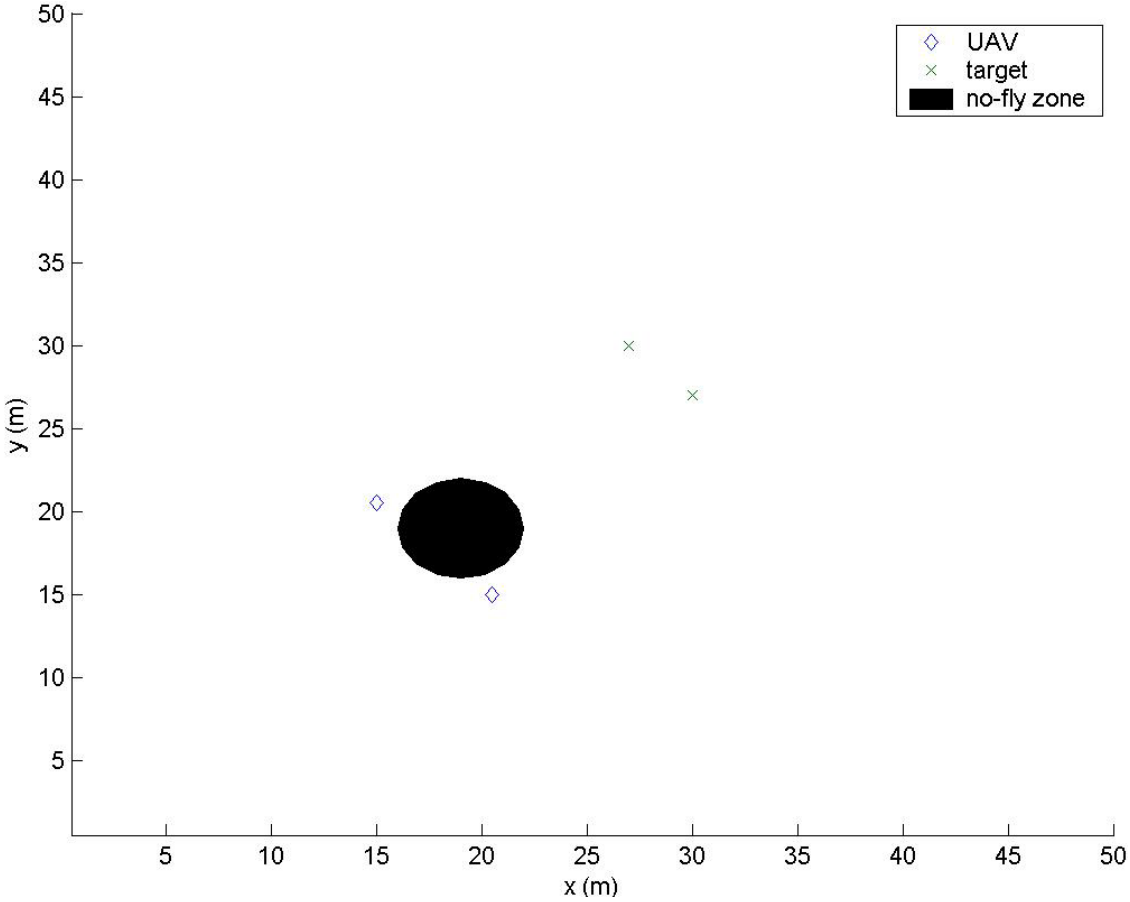


Figure 16: Initial setup of the battlefield, scenario 4

The results from the partially decoupled method are displayed in Figure 17, while the combined method results are in Figure 18.

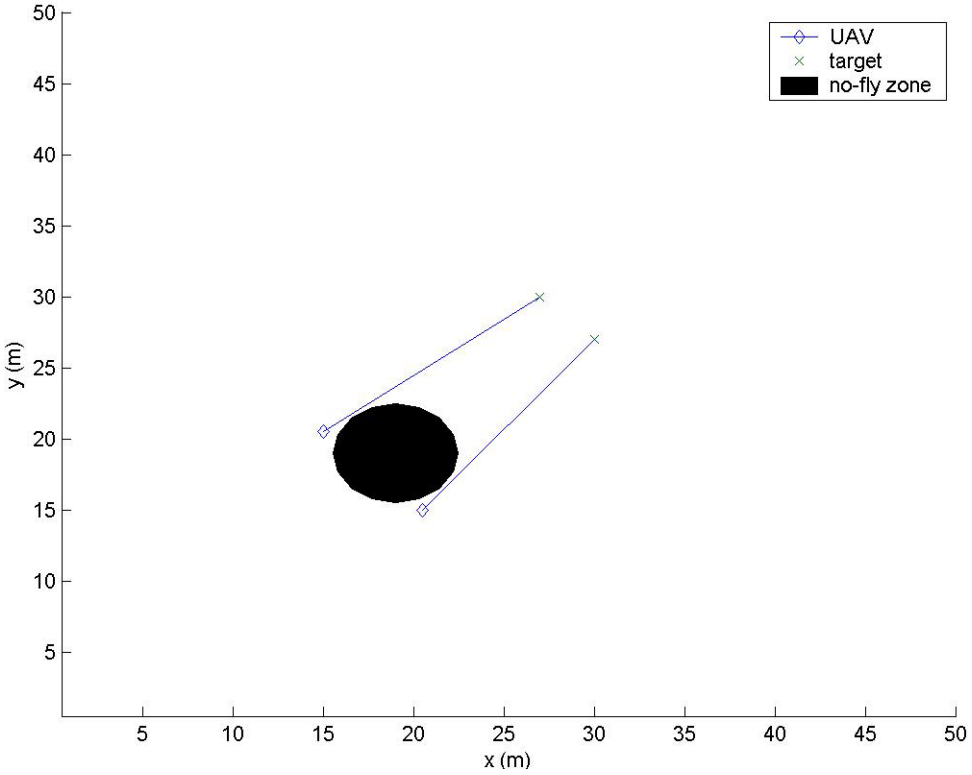


Figure 17: Results for scenario 4 using the partially decoupled method

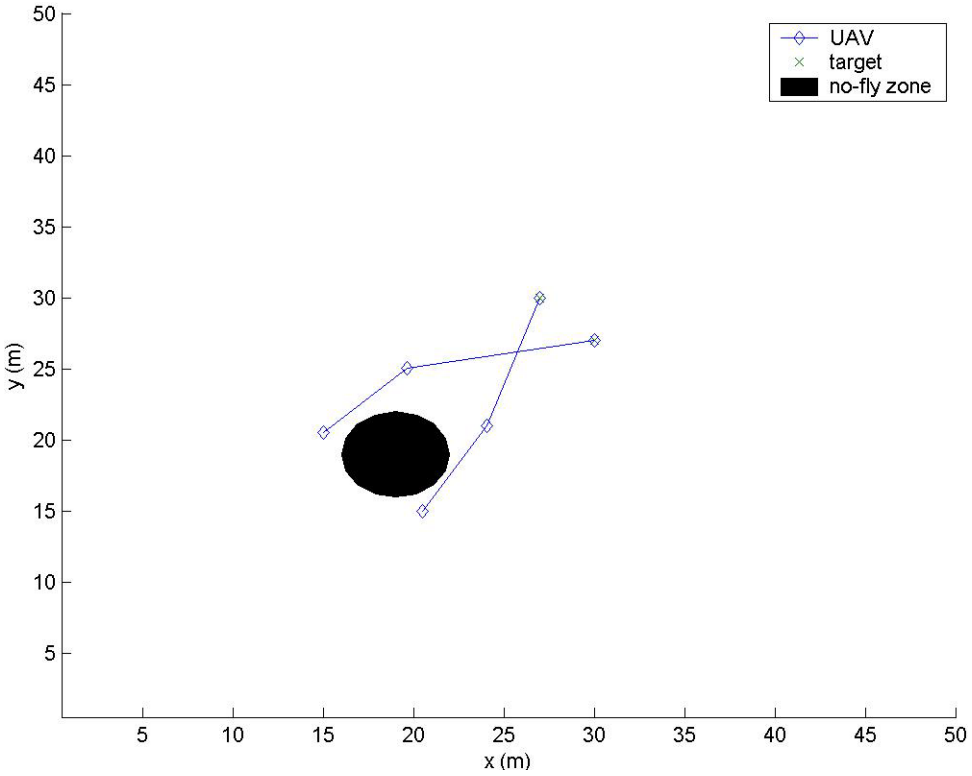


Figure 18: Results for scenario 4 using the combined method

The initial locations for scenario five are shown in Figure 19.

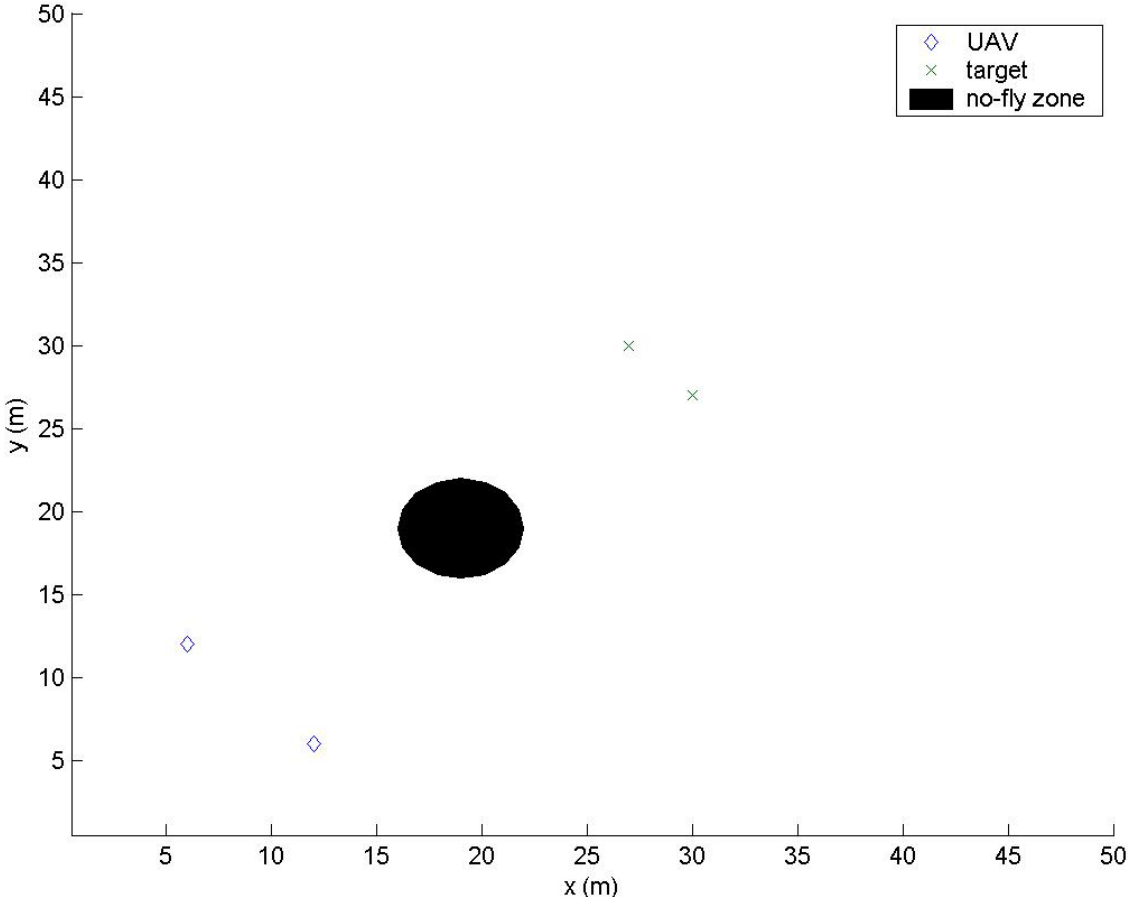


Figure 19: Initial setup of the battlefield, scenario 5

The results from the partially decoupled method are displayed in Figure 20, while the combined method results are in Figure 21.

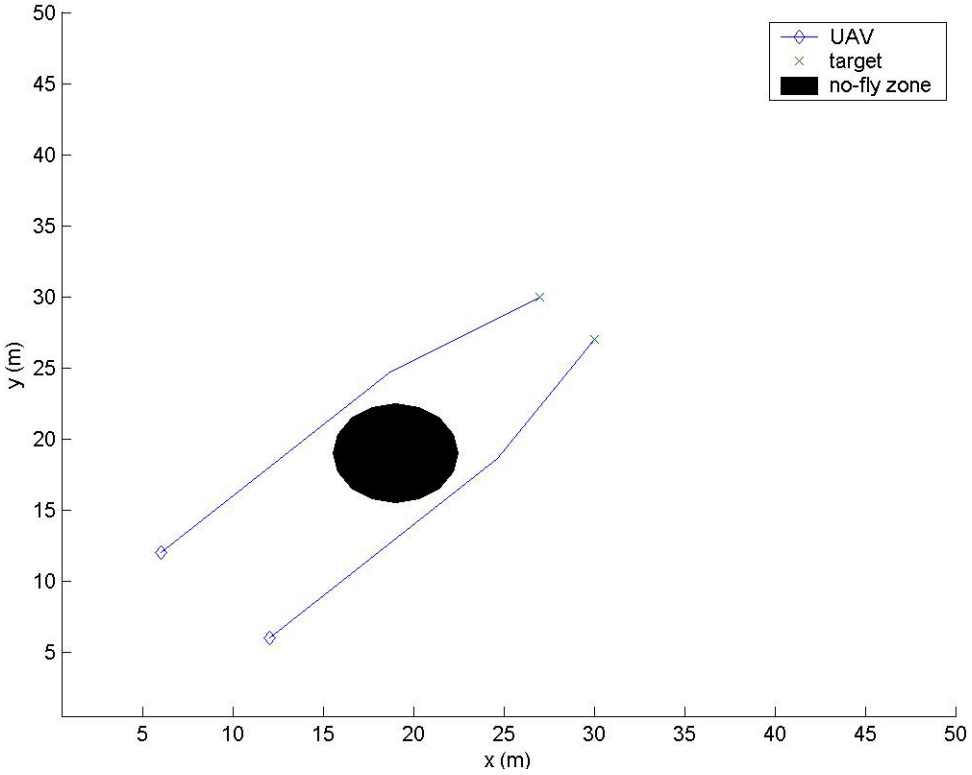


Figure 20: Results for scenario 5 using the partially decoupled method

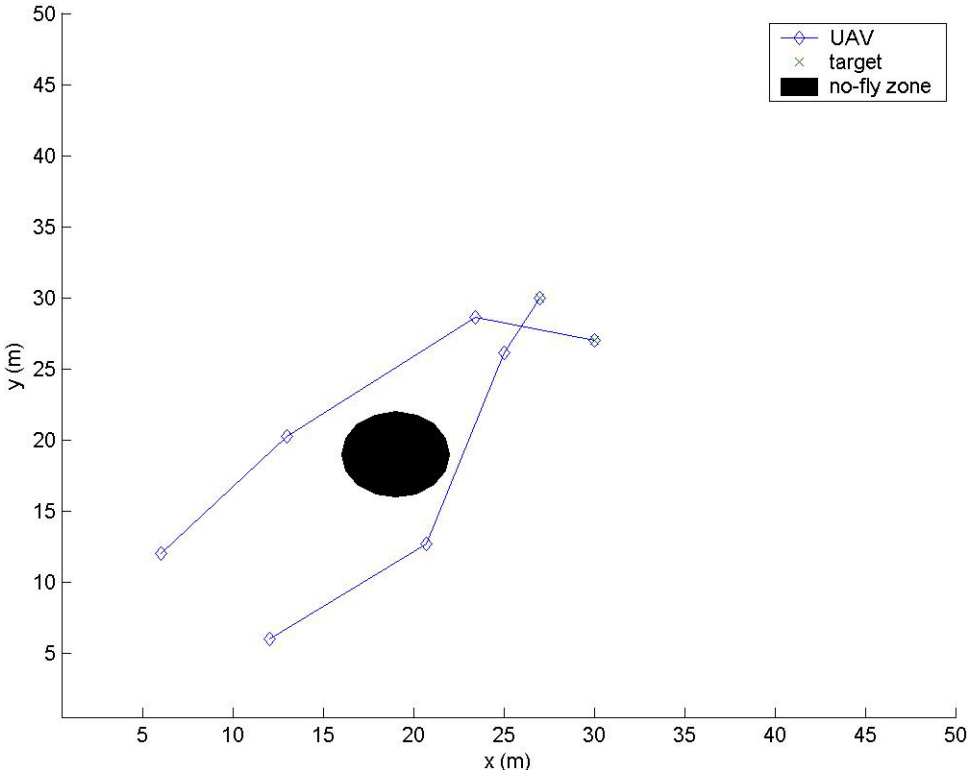


Figure 21: Results for scenario 5 using the combined method

The total cost for each scenario and method is found during the respective simulations. The cost for each method is the fuel cost, calculated using Equation 1, the distance formula. The results are shown in Table 3.

Table 3: Mission cost, in meters, for each method and scenario

	partial (m)	combined (m)	difference (m)
scenario 1	41.5	42.3	-0.9
scenario 2	59.8	65.8	-6.0
scenario 3	38.4	39.0	-0.6
scenario 4	30.6	33.5	-2.9
scenario 5	55.6	60.4	-4.8

The results in Table 3 indicate that the partially decoupled method paths are less expensive than the combined method paths. Table 4 contains the central processing unit (CPU) times necessary to solve each scenario by each method. This time does not include the initial setup of the simulations, nor the time necessary to generate plots.

Table 4: CPU time, in seconds, for each method and scenario

	partial (s)	combined (s)	difference (s)
scenario 1	0.391	0.050	0.341
scenario 2	0.401	0.050	0.351
scenario 3	0.360	0.050	0.310
scenario 4	0.361	0.060	0.301
scenario 5	0.381	0.050	0.331

The results in Table 4 show the combined method to take significantly less time to solve than the partially decoupled method. The computer being used to run these simulations is a Pentium® 4 CPU, 2.00 GHz, with 512 MB of RAM.

Chapter 5: Conclusions and Recommendations

Creating software that enables UAVs to attack specific targets while minimizing the associated risks is a goal for many researchers. Presented in this thesis are two methods for the desired path planning and task allocation. The first is a partially decoupled method. The second is a combined method. The two scenarios evaluated in Chapter 4 indicate that the partially decoupled method costs an average of 3.0 m less than the combined method, while taking an average of 0.327 s longer.

The cost results are due to the limitations of the student version of AMPL/CPLEX. More available times steps would allow the locations and velocities of the vehicles to be checked more frequently. This would allow more opportunities to change the velocities and provide smoother, more efficient paths to the targets.

Additionally, a full version of AMPL/CPLEX would allow for a larger battlefield, with more vehicles, targets, and no-fly zones. The no-fly zones would no longer have to be approximated with three rectangles, and the number of segments, M, for restricting forces and velocities could be expanded. As the number of constraints increases, the solution time is expected to increase.

The availability of additional variables and constraints provides the opportunity to add options to the combined method so it better parallels the partially decoupled method. For example, constraints defining the role of threats could be added. The tasks the UAVs are required to complete could be expanded to include classifying and assessing targets. Precedence, or importance, could be assigned to the targets. Rendezvous locations may also be defined to ensure the vehicles safely exit the combat zone. Vehicles of varying size and requirements may be incorporated.

In conclusion, as currently modeled, the partially decoupled method has many advantages over the combined method, while solving slightly slower. These advantages include larger scenarios, inclusion of threats, reactions to dynamic situations, final rendezvous locations, and available software.

References

1: "A Tutorial in Integer Programming." Ed. Michael A. Trick. 14 June 1998. The Operations Research Faculty of GSIA. 1 Sept 2003 <http://mat.gsia.cmu.edu/orclass/integer/integer.html>

2: Akbar, Mostofa Md., Eric G. Manning, Gholamali C. Shoja, and Shahadat Khan. "Heuristic Solution for Multi-dimensional Multiple Choice Knapsack Problem." 5 Mar. 2004 <http://www.panda.uvic.ca/papers/storage/HeuMCMDKP.pdf>

3: Bellingham, John, Michael Tillerson, Arthur Richards, and Jonathan P. How. "Multi-task Allocation and Path Planning for Cooperating UAVs."

4: Black, Paul E. "Dijkstra's Algorithm." National Institute of Standards and Technology. 5 Mar. 2004 <http://www.nist.gov/dads/HTML/dijkstraslgo.htm>

5: Chandler, P. R., and M. Pachter. "Hierarchical Control for Autonomous Teams," in the proceedings of the *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA Paper 2001-4149. AIAA, Reston, VA, 2001.

6: Chandler, Phillip R., Meir Pachter, Steven R. Rasmussen, and Corey Schumacher. "Distributed Control for Multiple UAVs with Strongly Coupled Tasks," in the proceedings of the *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA Paper 2003-5799. AIAA, Reston, VA, 2003.

7: Fourer, Robert (4er@iems.nwu.edu). "Linear Programming Frequently Asked Questions." 2000. Optimization Technology Center of Northwestern University and Argonne National Laboratory. 1 Sept 2003 <http://www-unix.mcs.anl.gov/otc/Guide/faq/ linear-programming-faq.html>

8: Fourer, Robert, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. 2nd ed. Toronto: Curt Hinrichs, 2003

9: Hazelton, Jennifer B., Matthew C. Lechliter, and Zachary W. Spritzer. "Path Planning and Task Allocation for Unmanned Air Vehicles (UAVs)," as presented at the *2003 AIAA Region I Mid-Atlantic Student Conference.* April, 2003.

10: Helwig, Catherine. "Dijkstra's Shortest Path Algorithm." 1997. 17 Mar. 2003 <http://renoir.vill.edu/~helwig/grafpage.html>

11: ILOG. *ILOG AMPL CPLEX System Version 8.0 Users Guide*. France: 2002. 28 Mar 2004. <http://netlib.bell-labs.com/netlib/ampl/solvers/cplex/ampl80.pdf >

12: Kay, Michael G. *Matlog software.* <http://www.ie.ncsu.edu/Kay/matlog>

13: Khan, Shahadatullah, Md. "Quality Adaptation in a Multisession Multimedie System: Model, Algorithms, and Architecture." Diss. University of Victoria, 1998. 5 Mar. 2004 <http://www.lapis.ece.uvic.ca/WWW_LAPIS/theses>

14: Kosecka, Jana. "CS685 – Lecture Notes." 16 Mar. 2004 <http://cs.gmu.edu/~kosecka/cs685/dynamicalSystems.pdf>

15: McLain, Timothy W., Phillip R. Chandler, Steven Rasmussen, and Meir Pachter. "Cooperative Control of UAV Rendezvous."

16: McLain, Timothy W., and Randal W. Beard. "Trajectory Planning for Coordination and Rendezvous of Unmanned Air Vehicles." AIAA Paper 2000-4369. AIAA, Reston, VA, 2000.

17: Pike, John. *Global Security*. 23 Apr 2004 <http://www.globalsecurity.org>

18: Richards, Arthur George. "Trajectory Optimization using Mixed-Integer Linear Programming." Thesis, Massachusetts Institute of Technology, June 2002.

19: Richards, Arthur, and Jonathan P. How. "Aircraft Trajectory Planning with Collision Avoidance Using Mixed Integer Linear Programming," in the proceedings of the *American Control Conference*, AIAA Paper 2002-1057. AIAA, Reston, VA, 2002.

20: Richards, Arthur, John Bellingham, Michael Tillerson, and Jonathan How. "Coordination and Control of Multiple UAVs," in the proceedings of the *AIAA Guidance, Navigation, and Control Conference and Exhibit*, AIAA Paper 2002-4588. AIAA, Reston, VA, 2002.

21: Weisstein, Eric W. "Voronoi Diagram." From *MathWorld*—A Wolfram Web Resource. 26 Apr 2004 <http://mathworld.wolfram.com/VoronoiDiagram.html>

Appendix A: Main MATLAB files for the Partially Decoupled Simulation

define_battlefield.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file is representative of the information necessary to input to the partially decoupled scheme. This information may also be entered using graphical user interface menus.

```
function[UAVS,TARGETS,THREATS,ZONES,n_uav,n_targ,n_zones,n_threats]=
        define_battlefield

UAVS=zeros(4,9);
TARGETS=zeros(4,9);
THREATS=zeros(4,15);
ZONES=zeros(3,10);

n_uav=menu('Enter the number of UAVs for this simulation','                    1
',...
   '2','3','4','5','6','7','8','9');
n_targ=menu('Enter the number of TARGETs for this simulation','
1                              ',...
   '2','3','4','5','6','7','8','9');
n_zones=menu('Enter the number of NO-FLY ZONEs for this simulation','
1                              ',...
   '2','3','4','5','6','7','8','9','10');
n_threats=menu('Enter the number of THREATs for this simulation','
1                              ',...
   '2','3','4','5','6','7','8','9','10','11','12','13','14','15');

Vel_UAV=0.13;
menu('Using the crosshairs and clicking on the plot','Place UAVs at desired positions');
axis([5 200 5 200]);
grid on;

for i=1:n_uav
   [UAVS(1,i),UAVS(2,i)]=ginput(1);
   plot(UAVS(1,i),UAVS(2,i),'bd');
   text(UAVS(1,i)+5,UAVS(2,i),{i},'FontSize',12,'Color','b');
   axis([5 200 5 200]);
   grid on;
   UAVS(3,i)=2;
   UAVS(4,i)=Vel_UAV ;
   hold on;
end
```

```
hold on;

menu('Using the crosshairs and clicking on the plot','Place TARGETs at desired positions');
for i=1:n_targ
   tar=menu('Select Target Value - Scale 10-100','10','20','30','40','50','60','70','80','90','100');
   TARGETS(3,i)=10*tar;
   TARGETS(4,i)=1;
   [TARGETS(1,i),TARGETS(2,i)]=ginput(1);
   plot(TARGETS(1,i),TARGETS(2,i),'x','Color',[0,.4,0]);
   text(TARGETS(1,i)+5,TARGETS(2,i),{i},'FontSize',12,'Color',[0,0.4,0]);
   axis([5 200 5 200]);
   grid on;
   hold on;
end

hold on;

menu('Using the crosshairs and clicking on the plot','Place NO-FLY ZONEs at desired positions');

for i=1:n_zones
   ZONES(3,i)=9;
   [ZONES(1,i),ZONES(2,i)]=ginput(1);
   axis([5 200 5 200]);
   grid on;
   t_nfz = (1/16:1/16:1)'*2*pi;
   x_nfz = ZONES(3,i)*sin(t_nfz)+ZONES(1,i);
   y_nfz = ZONES(3,i)*cos(t_nfz)+ZONES(2,i);
   fill(x_nfz,y_nfz,'k');
end

menu('Using the crosshairs and clicking on the plot','Place THREATs at desired positions');
hold on;

for i=1:n_threats
   thr=menu('Select Threat Type','KS-19 100mm AntiAircraft Artillery - Range 4000 meters, 40% Probability of Kill',...
      'SA-7 Grail - Man-Portable SAM - Range 5000 meters, 50% Probabilty of Kill',...
      'Crotale SAM - Range 10,000 meters, 80% Probability of Kill',...
      'SA-2 - Range 30,000 meters, 80% Probabilty of Kill');
   if thr == 1
      THREATS(3,i)=4;
```

```
    THREATS(4,i)=.4;
  end
  if thr == 2
    THREATS(3,i)=5;
    THREATS(4,i)=.5;
  end
  if thr == 3
    THREATS(3,i)=10;
    THREATS(4,i)=.8;
  end
  if thr == 4
    THREATS(3,i)=30;
    THREATS(4,i)=.8;
  end
  [THREATS(1,i),THREATS(2,i)]=ginput(1);
  plot(THREATS(1,i),THREATS(2,i),'r*');
  text(THREATS(1,i)+5,THREATS(2,i),{i},'FontSize',12,'Color','r')
  axis([5 200 5 200]);
  grid on;
  t_threat = (1/32:1/32:1)'*2*pi;
  x_threat = THREATS(3,i)*sin(t_threat)+THREATS(1,i);
  y_threat = THREATS(3,i)*cos(t_threat)+THREATS(2,i);
  plot(x_threat,y_threat,'r.');
  hold on;
end
```

path_planning.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file is the main file that takes the input data and determines the paths and allocates tasks.

```
function [out]=path_planning(in)

UAVS_long=in([1:36],1);
UAVS_long=reshape(UAVS_long,4,9);
TARGETS_long=in([37:72]);
TARGETS_long=reshape(TARGETS_long,4,9);
ZONES_long=in([73:102]);
ZONES_long=reshape(ZONES_long,3,10);
THREATS_long=in([103:162]);
THREATS_long=reshape(THREATS_long,4,15);
TIME=in(163);
n_plots=in(164);
HEADING_ANGLE=in([165:173]);
```

```
uavs_existing=zeros(1,9);
for i=1:9
   if abs(sum(UAVS_long(:,i)))>0 & abs(sum(UAVS_long(:,i)))~=0.26
      uavs_existing(1,i)=1;
  end
end
[UAVS]=filter_zeros(UAVS_long,9);
n_uav=size(UAVS,2);

targ_existing=zeros(1,9);
for i=1:9
   if TARGETS_long(3,i)~=0,
      targ_existing(1,i)=1;
  end
end
[TARGETS_temp]=filter_zeros(TARGETS_long,9);
TARGETS=[TARGETS_temp(1,:);TARGETS_temp(2,:)];
n_targ=size(TARGETS,2);

[ZONES]=filter_zeros(ZONES_long,10);
n_zones=size(ZONES,2);

threats_existing=zeros(1,15);
for i=1:15
   if THREATS_long(3,i)~=0
      threats_existing(1,i)=1;
  end
end
[THREATS]=filter_zeros(THREATS_long,15);
n_threats=size(THREATS,2);

ZONES_REAL=ZONES;
THREATS_REAL=THREATS;

ZONES(3,:)=1.15*ZONES_REAL(3,:);
THREATS(3,:)=1.15*THREATS_REAL(3,:);

split_seg=10;
min_turn=1;
[all_pos,all_lines_x,all_lines_y,all_costs]=vrn_diag_gen(UAVS,TARGETS,ZONES,
        THREATS);
[stored_paths,totalcost]=cheapest_paths(all_pos,all_lines_x,all_lines_y,all_costs,UAVS,
        TARGETS,ZONES,THREATS);
[Shortened_Paths_x,Shortened_Paths_y,totalcost]=path_shrtng(stored_paths,all_pos,
        ZONES,THREATS,min_turn,split_seg,n_uav,n_targ,HEADING_ANGLE);
[Selected_Paths_x,Selected_Paths_y,mincost]=mmkp_task_allocation(totalcost,
```

```
        Shortened_Paths_x,Shortened_Paths_y,n_uav);
[uav_path_x,uav_path_y,time_uav,altitude_uav]=vrt_sim_convert(Selected_Paths_x,
        Selected_Paths_y,UAVS,min_turn*2);
keyboard
if n_plots~=0,
   plot_uav(UAVS_long,TARGETS_long,ZONES_REAL,THREATS_long,uav_path_x,
        uav_path_y,n_plots,uavs_existing,targ_existing,threats_existing);
end

disp(sprintf('Path Planning ran at time %d. \n',round(TIME)));

bestcomb=zeros(1,9);
for i=1:n_uav,
   for j=1:n_targ,
      if round(Selected_Paths_x(end,i)*10)==round(TARGETS(1,j)*10) &
         round(Selected_Paths_y(end,i)*10)==round(TARGETS(2,j)*10)
         bestcomb(1,i)=j;
         break
      end
   end
end


%Making into vector
uav_x=zeros(9,100);
uav_y=zeros(9,100);
uav_time=zeros(9,100);
uav_alt=zeros(9,100);
selected_targets=zeros(9,1);
szpath=size(uav_path_x,2);
counter=1;
for i=1:9,
   if uavs_existing(1,i)==1
      selected_targets(i,1)=bestcomb(1,counter);
      uav_x(i,[1:szpath])=uav_path_x(counter,:);
      uav_y(i,[1:szpath])=uav_path_y(counter,:);
      uav_time(i,[1:szpath])=time_uav(counter,:)+TIME;
      uav_alt(i,[1:szpath])=altitude_uav(counter,:);
      counter=counter+1;
   end
end
sys_temp=[];
for i=1:9;
   sys_temp=[sys_temp,uav_x(i,:),uav_y(i,:),uav_alt(i,:),uav_time(i,:)];
end
out=[sys_temp,selected_targets'];
```

vrn_diag_gen.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file generates the Voronoi diagram based on the locations of the threats and no-fly zones.

```
function[all_pos,all_lines_x,all_lines_y,all_costs]=vrn_diag_gen(UAVS,TARGETS,
        ZONES,THREATS)
%INPUTS:
%
%UAVS - is a 4xn matrix where n is number of UAVs, the first row is the
%initial x position of the UAVs, the second row is the initial y position
%of the UAVs, the third row is the initial altitude of the UAVs, and
%the fourth row is the initial Velocity of the UAVs.
%
%TARGETS - is a 2xn matrix where n is the number of Targets, the first row
%is the x position of the targets and the second row is the y position of
%the targets.
%
%ZONES - is a 3xn matrix where n is the number of no-fly zones, the first
%row is the x position of the no-fly zones, the second row is the y
%position of the no-fly zones, and the third row is the radius or range of
%the no-fly zones.
%
%THREATS - is a 4xn matrix where n is the number of threats, the first row
%is the x position of the threats, the second row is the y position of the
%threats, the third row is the range of the threats, and the fourth row is
%the level of danger of the threats.
%
%OUTPUTS:
%
%all_pos - is a 2xn matrix where n is the number of unique Voronoi points,
%uav points, and target points.  Where the first row is the x position and
%the second row is the y position of all of these unique points.
%
%all_lines_x - is a 2xn matrix where n is the number of all of the lines
%for the Voronoi, UAVs, and targets.  The first row is the ending point's
%x position for the nth line and the second row is the starting point's
%x position for the nth line.
%
%all_lines_y - is a 2xn matrix where n is the number of all of the lines
%for the Voronoi, UAVs, and targets.  The first row is the ending point's
%y position for the nth line and the second row is the starting point's
%y position for the nth line.
%
```

```
%all_costs - is a 1xn row where n is the number of all of the lines
%for the Voronoi, UAVs, and targets.  This row is the costs for all of the
%lines of all_lines_x and all_lines_y

max_x=max([TARGETS(1,:),UAVS(1,:),ZONES(1,:),THREATS(1,:)])+25;
min_x=min([TARGETS(1,:),UAVS(1,:),ZONES(1,:),THREATS(1,:)])-25;
max_y=max([TARGETS(2,:),UAVS(2,:),ZONES(2,:),THREATS(2,:)])+25;
min_y=min([TARGETS(2,:),UAVS(2,:),ZONES(2,:),THREATS(2,:)])-25;

VRNPTS=[ZONES([1,2],:) THREATS([1,2],:) ...
   [(min_x)*ones(1,4);(((max_y-min_y)*[1:4]/4)+min_y)] ...
   [(max_x)*ones(1,4);(((max_y-min_y)*[1:4]/4)+min_y)] ...
   [(((max_x-min_x)*[1:4]/4)+min_x);(min_y)*ones(1,4)] ...
   [(((max_x-min_x)*[1:4]/4)+min_x);(max_y)*ones(1,4)]];

[vx,vy] = voronoi(VRNPTS(1,:),VRNPTS(2,:));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Taking unique numbers from vx and vy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[vxyn]= 1e-6*unique(round(1e6*[vx(:),vy(:)]),'rows');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Connecting UAV's into Voronoi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[line_cost_uav,uavx,uavy]=connect_vrn(vxyn,UAVS([1,2],:));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Connecting the targets into the Voronoi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[line_cost_targ,targx,targy]=connect_vrn(vxyn,TARGETS([1,2],:));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generation for Voronoi line costs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nvlines=size(vx,2);
line_cost_vrn=zeros(1,nvlines);
for i=1:nvlines,
   line_cost_vrn(1,i)=sqrt((vx(1,i)-vx(2,i))^2+(vy(1,i)-vy(2,i))^2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Stacking unique positions, lines for x and y, and costs of those lines
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
all_pos=[UAVS([1,2],:) vxyn(:,[1,2])' TARGETS([1,2],:)];
all_lines_x=[uavx([1,2],:) vx([1,2],:) targx([1,2],:)];
all_lines_y=[uavy([1,2],:) vy([1,2],:) targy([1,2],:)];
all_costs=[line_cost_uav(1,:) line_cost_vrn(1,:) line_cost_targ(1,:)];
```

connect_vrn.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file is called in the vrn_diag_gen.m file to connect the UAVs and the targets to the Voronoi diagram.

```
function [line_cost_uav,uavx,uavy]=connect_vrn(vxyn,UAVS)

%Inputs:
%vxyn - is a nx2 matrix with first column defining all of the unique x
%positions of the Voronoi diagram or grid and the second column defining
%all of the unique y positions of the Voronoi diagram or grid.
%
%UAVS - is a 2xn matrix with the first row defining the x position of the
%UAV and the second row defining the y position of the UAV.
%
%Outputs:
%
%line_cost_uav - is a vector containing the cost of the lines of connecting
%the UAV's into the Voronoi diagram or grid
%
%uavx - is a 2xn matrix with first row defining ending point and second row
%defining starting point for the x coordinates.
%
%uavy - is a 2xn matrix with first row defining ending point and second row
%defining starting point for the y coordinates.
%
nuav=size(UAVS,2);
nvxynpts=size(vxyn,1);
du=zeros(1,nvxynpts-1);
uavx=zeros(2,nuav*3);
uavy=zeros(2,nuav*3);
line_cost_uav=zeros(1,nuav*3);
for k=1:nuav,
   for j=2:nvxynpts,
      du(1,j-1)=sqrt((UAVS(1,k)-vxyn(j,1))^2+(UAVS(2,k)-vxyn(j,2))^2);
   end
   mdu=sort(du,2);
   for i=1:3,
      mdu_loc=find(du==mdu(1,i));
      uavx(1,3*(k-1)+i)=vxyn(mdu_loc+1,1);
      uavy(1,3*(k-1)+i)=vxyn(mdu_loc+1,2);
      uavx(2,3*(k-1)+i)=UAVS(1,k);
      uavy(2,3*(k-1)+i)=UAVS(2,k);
      line_cost_uav(1,3*(k-1)+i)=mdu(1,i);
   end
end
```

cheapest_paths.m

Authored by Michael G. Kay, [H]

This file finds the cheapest paths for each permutation of UAV to target. It uses Dijkstra's algorithm.

```
function[stored_paths,totalcost]=cheapest_paths(all_pos,all_lines_x,all_lines_y,all_costs,
        UAVS,TARGETS,ZONES,THREATS)
%
%INPUTS:
%
%all_pos - is a 2xn matrix where n is the number of unique Voronoi points,
%uav points, and target points.  Where the first row is the x position and
%the second row is the y position of all of these unique points.
%
%all_lines_x - is a 2xn matrix where n is the number of all of the lines
%for the Voronoi, UAVs, and targets.  The first row is the ending point's
%x position for the nth line and the second row is the starting point's
%x position for the nth line.
%
%all_lines_y - is a 2xn matrix where n is the number of all of the lines
%for the Voronoi, UAVs, and targets.  The first row is the ending point's
%y position for the nth line and the second row is the starting point's
%y position for the nth line.
%
%all_costs - is a 1xn row where n is the number of all of the lines
%for the Voronoi, UAVs, and targets.  This row is the costs for all of the
%lines of all_lines_x and all_lines_y.
%
%UAVS - is a 4xn matrix where n is number of UAVs, the first row is the
%initial x position of the UAVs, the second row is the initial y position
%of the UAVs, the third row is the initial altitude of the UAVs, and
%the fourth row is the initial Velocity of the UAVs.
%
%TARGETS - is a 2xn matrix where n is the number of Targets, the first row
%is the x position of the targets and the second row is the y position of
%the targets.
%
%ZONES - is a 3xn matrix where n is the number of no-fly zones, the first
%row is the x position of the no-fly zones, the second row is the y
%position of the no-fly zones, and the third row is the radius or range of
%the no-fly zones.
%
%THREATS - is a 4xn matrix where n is the number of threats, the first row
%is the x position of the threats, the second row is the y position of the
%threats, the third row is the range of the threats, and the fourth row is
%the level of danger of the threats.
```

```
%
%OUTPUTS:
%
%stored_paths - is a mxn matrix where m is the number of UAVs times the
%number of targets and n is the length of the longest path.  The first row
%being the first path for the first UAV and the last row being the last
%path for the last UAV. The paths are output by node numbers coming from
%the implementation of Dijkstra's algorithm [H].
%
%totalcost - is a mxn matrix where m is the number of UAVs and n is the
%number of possible paths for each UAV.  The element (m,n) of this matrix
%is the cost for the mth UAV to take the nth path.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Making THC matrix for Dijkstra's algorithm [H]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[THC]=set_THC(all_pos,all_lines_x,all_lines_y,all_costs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Cost Assignment for all lines
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[THC]= c_assign(all_pos,THC,ZONES,THREATS);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Adding the reverse of the THC matrix onto the end, so that the
%reverse of the lines is possible
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
THC=[THC(:,[1,2,3]); THC(:,[2,1,3])];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Implementing Dijkstra's algorithm [H]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
nuav=size(UAVS,2);
ntarg=size(TARGETS,2);
A = list2adj(THC);
totalcost=zeros(nuav,ntarg);
for i=1:nuav,
   for j=1:ntarg,
      [totalcost(i,j),path] = dijk(A,i,size(all_pos,2) - j + 1);
      stored_paths((i-1)*ntarg+j,[1:size(path,2)])=path(1,[1:size(path,2)]);
   end
end
```

c_assign.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file assigns the costs for each segment of the Voronoi diagram. These costs must be assigned before the cheapest paths can be determined.

```
function [THC]= c_assign(all_pos,THC,ZONES,THREATS)
%
%INPUTS:
%
%all_pos - is a 2xn matrix where n is the number of unique Voronoi points,
%UAV points, and target points.  Where the first row is the x position and
%the second row is the y position of all of these unique points.
%
%THC - is a nx3 matrix where n is the number of possible lines to be chosen
%the first column is the tail of the line or starting point, the second
%column is the head of the line or the ending point, and the third column
%is the cost of the line.
%
%ZONES - is a 3xn matrix where n is the number of no-fly zones, the first
%row is the x position of the no-fly zones, the second row is the y
%position of the no-fly zones, and the third row is the radius or range of
%the no-fly zones.
%
%THREATS - is a 4xn matrix where n is the number of threats, the first row
%is the x position of the threats, the second row is the y position of the
%threats, the third row is the range of the threats, and the fourth row is
%the level of danger of the threats.
%
%OUTPUTS:
%
%THC - is a nx3 matrix where n is the number of possible lines to be chosen
%the first column is the tail of the line or starting point, the second
%column is the head of the line or the ending point, and the third column
%is the cost of the line. With updated costs due to no-fly zones and
%threats.

szthc=size(THC,1);
nzones=size(ZONES,2);
nthrts=size(THREATS,2);

for i=1:szthc,
   start=THC(i,1);finish=THC(i,2);
   SF=sqrt(((all_pos(1,finish)-all_pos(1,start))^2)+((all_pos(2,finish)-
all_pos(2,start))^2));                    % cost associated with length of each segment
   for j=1:nzones,
      SC=sqrt(((ZONES(1,j)-all_pos(1,start))^2)+((ZONES(2,j)-all_pos(2,start))^2));
```

```
      FC=sqrt(((ZONES(1,j)-all_pos(1,finish))^2)+((ZONES(2,j)-all_pos(2,finish))^2));
      SN=(SC^2+SF^2-FC^2)/(2*SF);
      if SN<SF & SN>0,PC=sqrt(SC^2-SN^2);
      else
         if SC<FC,PC=SC;
         else
            PC=FC;
         end
      end
      if PC < ZONES(3,j),THC(i,3)=1e30*THC(i,3);
          % cost associated with intersecting a no-fly zone
      end
   end
   for j=1:nthrts,
      SC=sqrt(((THREATS(1,j)-all_pos(1,start))^2)+((THREATS(2,j)-
all_pos(2,start))^2));
      FC=sqrt(((THREATS(1,j)-all_pos(1,finish))^2)+((THREATS(2,j)-
all_pos(2,finish))^2));
      SN=(SC^2+SF^2-FC^2)/(2*SF);
      if SN<SF & SN>0,PC=sqrt(SC^2-SN^2);
      else
         if SC<FC,PC=SC;
         else
            PC=FC;
         end
      end
      if PC < THREATS(3,j),THC(i,3)=(THREATS(4,j)*100)+THC(i,3);
         % cost associated with intersecting a threat and/or its effective range
      end
   end
end
```

path_shrtng.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file looks for line-of-sight paths to reduce the costs of the mission. It also provides fillets for sharp corners and the proper heading angles for the entire path. After these elements are taken care of, the costs are updated to reflect the changes.

```
function[Shortened_Paths_x,Shortened_Paths_y,totalcost]=path_shrtng(stored_paths,
        all_pos,ZONES,THREATS,min_turn,split_seg,nuav,ntarg,HEADING_ANGLE)

%INPUTS:
%
%stored_paths - is a mxn matrix where m is the number of UAVs times the
%number of targets and n is the length of the longest path.  The first row
```

```
%being the first path for the first UAV and the last row being the last
%path for the last UAV. The paths are output by node numbers coming from
%the implementation of Dijkstra's algorithm.
%
%all_pos - is a 2xn matrix where n is the number of unique Voronoi points,
%UAV points, and target points.  Where the first row is the x position and
%the second row is the y position of all of these unique points.
%
%ZONES - is a 3xn matrix where n is the number of no-fly zones, the first
%row is the x position of the no-fly zones, the second row is the y
%position of the no-fly zones, and the third row is the radius or range of
%the no-fly zones.
%
%THREATS - is a 4xn matrix where n is the number of threats, the first row
%is the x position of the threats, the second row is the y position of the
%threats, the third row is the range of the threats, and the fourth row is
%the level of danger of the threats.
%
%min_turn - minimum turning radius for the UAVs
%
%split_seg - number of segments to Split the Voronoi lines into for the
%purpose of a more near-optimal solution
%
%nuav - number of UAVs
%
%ntarg - number of targets

%OUTPUTS:
%
%Shortened_Paths - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs multiplied by the number of targets.
%The element (nxmx1) x position of the mth UAV at point n.  The element
%(nxmx2) y position of the mth UAV at point n.
%
%totalcost - is a mxn matrix where m is the number of UAV and n is the
%number of possible paths for each UAV.  The element (m,n) of this matrix
%is the cost for the mth UAV to take the nth path.
%
%Stored_Pos - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs multiplied by the number of targets.
%The element (nxmx1) x position of the mth UAV at point n.  The element
%(nxmx2) y position of the mth UAV at point n.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Splitting the Voronoi lines into more segments for the purpose of a more
%near-optimal solution
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
szpths=size(stored_paths,2);
 split_vect=[(0:(1/split_seg):(1- 1/split_seg))]';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Finding the corresponding x and y coordinates of the smaller segments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Stored_Pos_x=ones(szpths,nuav*ntarg);
Stored_Pos_y=ones(szpths,nuav*ntarg);
stored_paths(:,szpths+1)=0;
for i=1:nuav*ntarg,
    mnz=min(find(stored_paths(i,:)==0));
    Stored_Pos_x(1:mnz-1,i)=all_pos(1,stored_paths(i,1:mnz-1))';
    Stored_Pos_y(1:mnz-1,i)=all_pos(2,stored_paths(i,1:mnz-1))';
    Stored_Pos_x(mnz:end,i)=ones((szpths-mnz+1),1)*all_pos(1,stored_paths(i,mnz-1))';
    Stored_Pos_y(mnz:end,i)=ones((szpths-mnz+1),1)*all_pos(2,stored_paths(i,mnz-1))';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Stored_Pos_x_new=ones((((szpths-1)*split_seg)+1),nuav*ntarg);
Stored_Pos_y_new=ones((((szpths-1)*split_seg)+1),nuav*ntarg);
for k=1:nuav*ntarg,
        j=1;
    for i=1:(szpths -1),
          Stored_Pos_x_new([j:(j + (split_seg -1))],k)=
           ones(split_seg,1)*Stored_Pos_x(i,k)+split_vect*(Stored_Pos_x(i+1,k)-
          Stored_Pos_x(i,k));
          Stored_Pos_y_new([j:(j + (split_seg -1))],k)=
           ones(split_seg,1)*Stored_Pos_y(i,k)+split_vect*(Stored_Pos_y(i+1,k)-
          Stored_Pos_y(i,k));
          j=j+ split_seg;
    end
    Stored_Pos_x_new((((szpths-1)*split_seg)+1),k)=Stored_Pos_x(szpths,k);
    Stored_Pos_y_new((((szpths-1)*split_seg)+1),k)=Stored_Pos_y(szpths,k);
end

Shortened_Paths_x_end=ones(500,1)*Stored_Pos_x(szpths,:);
Shortened_Paths_y_end=ones(500,1)*Stored_Pos_y(szpths,:);
Shortened_Paths_x=[Stored_Pos_x_new;Shortened_Paths_x_end];
Shortened_Paths_y=[Stored_Pos_y_new;Shortened_Paths_y_end];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Shortening the paths
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:nuav*ntarg,
```

```
[Shortened_Paths_x(:,i),Shortened_Paths_y(:,i)]=shorten_paths(Shortened_Paths_x(:,i),S
hortened_Paths_y(:,i),ZONES,THREATS,Stored_Pos_x(:,i),Stored_Pos_y(:,i));
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Putting fillets into the shortened paths
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:nuav*ntarg,
[Shortened_Paths_x(:,i),Shortened_Paths_y(:,i)]=fillet_path([Shortened_Paths_x(:,i),Shor
tened_Paths_y(:,i)],min_turn);
end

Shortened_Paths_x_old=Shortened_Paths_x;
Shortened_Paths_y_old=Shortened_Paths_y;
Shortened_Paths_x=[];
Shortened_Paths_y=[];
for j=1:size(Shortened_Paths_x_old,1)-1,
   if Shortened_Paths_x_old(j,:)==Shortened_Paths_x_old(j+1,:) &
      Shortened_Paths_y_old(j,:)==Shortened_Paths_y_old(j+1,:),
      Shortened_Paths_x(j,:)=Shortened_Paths_x_old(j,:);
      Shortened_Paths_y(j,:)=Shortened_Paths_y_old(j,:);
      break
   else
     Shortened_Paths_x(j,:)=Shortened_Paths_x_old(j,:);
     Shortened_Paths_y(j,:)=Shortened_Paths_y_old(j,:);
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Updating the Costs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
szsp_perm=size(Shortened_Paths_x,2);
permcost=zeros(nuav*ntarg,1);
for z=1:szsp_perm,
[permcost(z,1)]=update_cost([Shortened_Paths_x(:,z),Shortened_Paths_y(:,z)],
THREATS);
end
totalcost=reshape(permcost,ntarg,nuav)';
```

shorten_paths.m

Authored by Zachary Spritzer, and Matthew Lechliter

This file actually shortens the paths.

```
function [shr_x,shr_y]=shorten_paths(sp_x,sp_y,Z,T,spo_x,spo_y)
```

```
%INPUTS:
%
%sp - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs.  The element (nxmx1) x position of the
%mth UAV at point n.  The element (nxmx2) y position of the mth UAV at
%point n.
%
%Z - is a 3xn matrix where n is the number of no-fly zones, the first
%row is the x position of the no-fly zones, the second row is the y
%position of the no-fly zones, and the third row is the radius or range of
%the no-fly zones.
%
%T - is a 4xn matrix where n is the number of threats, the first row
%is the x position of the threats, the second row is the y position of the
%threats, the third row is the range of the threats, and the fourth row is
%the level of danger of the threats.
%
%spo - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs.  The element (nxmx1) x position of the
%mth UAV at point n.  The element (nxmx2) y position of the mth UAV at
%point n. This matrix is the original matrix without the Voronoi segments
%split up.
%
%OUTPUTS:
%
%shr - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs.  The element (nxmx1) x position of the
%mth UAV at point n.  The element (nxmx2) y position of the mth UAV at
%point n.
spo=[spo_x,spo_y];
sp=[sp_x,sp_y];
SC=0;FC=0;SF=0;SN=0;
for j=1:size(T,2),
   PC=[];
   for i=1:size(spo,1)-1,
      SC=sqrt(((T(1,j)-spo(i,1))^2)+((T(2,j)-spo(i,2))^2));
      FC=sqrt(((T(1,j)-spo(i+1,1))^2)+((T(2,j)-spo(i+1,2))^2));
      SF=sqrt(((spo(i+1,1)-spo(i,1))^2)+((spo(i+1,2)-spo(i,2))^2));
      SN=(SC^2+SF^2-FC^2)/(2*SF);
      if SN<SF & SN>0
         PC(i)=sqrt(SC^2-SN^2);
      else
         if SC<FC
            PC(i)=SC;
         else
```

```
            PC(i)=FC;
         end
      end
      mPC=min(PC);
      if  mPC< T(3,j),
         T(3,j)=mPC*.995;
      end
   end
end

ZT=[Z([1:3],:) T([1:3],:)];
szzt=size(ZT,2);
szsp=size(sp,1);
shr=ones(szsp,2);
for i=1:2,
   shr(:,i)=sp(szsp,i);
end
shr(1,:)=sp(1,:);
a=1;
PC=zeros(1,szzt);
while shr(a,:)~=sp(szsp,:),
   for i=1:szsp,
      if shr(a,:)==sp(i,:)
         pck=i;
         break
      end
   end
   for i=szsp:-1:pck+1,
      SF=sqrt(((shr(a,1)-sp(i,1))^2)+((shr(a,2)-sp(i,2))^2));
      for j=1:szzt,
         SC=sqrt(((ZT(1,j)-shr(a,1))^2)+((ZT(2,j)-shr(a,2))^2));
         FC=sqrt(((ZT(1,j)-sp(i,1))^2)+((ZT(2,j)-sp(i,2))^2));
         SN=(SC^2+SF^2-FC^2)/(2*SF);
         if SN<SF & SN>0
            PC(1,j)=sqrt(SC^2-SN^2);
         else
            if SC<FC
               PC(1,j)=SC;
            else
               PC(1,j)=FC;
            end
         end
      end
      if PC(1,:)>ZT(3,:),
         a=a+1;
         shr(a,:)=sp(i,:);
```

```
        break
      end
   end
end
shr_x=shr(:,1);
shr_y=shr(:,2);
```

fillet_path.m

Authored by Jennifer Hazelton, and Matthew Lechliter

This file places fillets in the corners of the paths.

```
function[Shortened_Paths_fillet_x,Shortened_Paths_fillet_y]=fillet_path
        (Shortened_Paths,min_turn)

%INPUTS:
%
%Shortened_Paths - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs multiplied by the number of targets.
%The element (nxmx1) x position of the mth UAV at point n.  The element
%(nxmx2) y position of the mth UAV at point n.
%
%min_turn - minimum turning radius for the UAVs

%OUTPUTS:
%
%Shortened_Paths_fillet - is a nxmx2 matrix where n is the length of the
%longest path with the addition of fillets ((2*old size)-1) and m is the
%number of UAVs multiplied by the number of targets.  The element (nxmx1)
%x position of the mth UAV at point n.  The element (nxmx2) y position of
%the mth UAV at point n.

Shortened_Paths_fillet=Shortened_Paths*0;
Shortened_Paths_fillet(:,1)=Shortened_Paths(size(Shortened_Paths,1),1);
Shortened_Paths_fillet(:,2)=Shortened_Paths(size(Shortened_Paths,1),2);
Shortened_Paths_fillet(1,:)=Shortened_Paths(1,:);

fillet_counter=2;
for j=2:size(Shortened_Paths,1)-1,
   if Shortened_Paths(j,:)==Shortened_Paths(j+1,:),
      break
   end
   start=Shortened_Paths(j-1,:);
   middle=Shortened_Paths(j,:);
   finish=Shortened_Paths(j+1,:);
   SM=sqrt(sum((middle-start).^2));
```

```
    MF=sqrt(sum(((finish-middle).^2)));
    SF=sqrt(sum(((finish-start).^2)));
    alpha=acos((SM^2+MF^2-SF^2)/(2*SM*MF));
    Fillet=min_turn/tan(alpha/2);
    if Fillet>=SM
       Shortened_Paths_fillet(fillet_counter,:)=Shortened_Paths(j-1,:);
    else
       Shortened_Paths_fillet(fillet_counter,:)=Shortened_Paths(j-
          1,:)+(Shortened_Paths(j,:)-Shortened_Paths(j-1,:))*((SM-Fillet)/SM);
    end
    if Fillet>=MF,
       Shortened_Paths_fillet(fillet_counter+1,:)=Shortened_Paths(j+1,:);
    else
          Shortened_Paths_fillet(fillet_counter+1,:)=Shortened_Paths(j,:)+
          (Shortened_Paths(j+1,:)-Shortened_Paths(j,:))*(Fillet/MF);
    end
    fillet_counter=fillet_counter+2;
end
Shortened_Paths_fillet_x=Shortened_Paths_fillet(:,1);
Shortened_Paths_fillet_y=Shortened_Paths_fillet(:,2);
```

mmkp_task_allocation.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file calls the functions necessary to assign tasks to each of the UAVs in a manner that minimizes the overall mission cost.

```
function [Selected_Paths_x,Selected_Paths_y,mincost]=mmkp_task_allocation(totalcost,
        Shortened_Paths_x,Shortened_Paths_y,nuav)

%INPUTS:
%
%totalcost - is a mxn matrix where m is the number of UAVs and n is the
%number of possible paths for each UAV.  The element (m,n) of this matrix
%is the cost for the mth UAV to take the nth path.
%
%Shortened_Paths - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs multiplied by the number of targets.
%The element (nxmx1) x position of the mth UAV at point n.  The element
%(nxmx2) y position of the mth UAV at point n.
%
%nuav - number of UAVs

%OUTPUTS:
%
%Selected_Pos - is a nxmx2 matrix where n is the length of the longest
```

```
%path and m is the number of UAVs.  The element (nxmx1) x position of the
%mth UAV at point n.  The element (nxmx2) y position of the mth uav at
%point n.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%MMKP algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[bestcomb,mincost]=mmkp_new(totalcost);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Taking the results from mmkp
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Selected_Paths_x=zeros(size(Shortened_Paths_x,1),nuav);
Selected_Paths_y=zeros(size(Shortened_Paths_x,1),nuav);
for i=1:nuav,
   Selected_Paths_x(:,i)=Shortened_Paths_x(:,(nuav)*(i-1)+bestcomb(1,i));
   Selected_Paths_y(:,i)=Shortened_Paths_y(:,(nuav)*(i-1)+bestcomb(1,i));
end
```

mmkp_new.m

Authored by Jennifer Hazelton, Zachary Spritzer, Matthew Lechliter, and Elena Lucchi

This file finds the permutation from UAVs to target associated with the minimum cost.

```
function [bestcomb,mincost]=mmkp_new(totalcost)

%Inputs:
%
%totalcost - is a nxm matrix where n is the total number of UAVs and m is
%the total number of targets or paths. Where the element nxm is the cost
%associated with UAV "n" choosing target or path "m".
%
%Outputs:
%
%bestcomb - is a 1xn row with n equal to the number of UAVs where each
%element of the row represents which path the UAV should select to give the
%optimal solution.
%
%mincost - is a scalar number which is sum of the optimal costs for all
%the UAVs paths.

nuav=size(totalcost,1);
mincost=inf;
C_new=perms(1:nuav);
for j=1:size(C_new,1),
```

```
   sc=0;
   for i=1:nuav,
      sc=sc+totalcost(i,C_new(j,i));
   end
   if sc < mincost
      bestcomb=C_new(j,:);
      mincost = sc;
   end
end
```

vrt_sim_convert.m

Authored by Zachary Spritzer

This file converts all of the data necessary for creating a VRT simulation.

```
function[uav_path_x,uav_path_y,time_uav,altitude_uav]=vrt_sim_convert(shr_x,shr_y,
        UAVS,distpast)
%
%INPUTS:
%
%shr - is a nxmx2 matrix where n is the length of the longest
%path and m is the number of UAVs.  The element (nxmx1) x position of the
%mth UAV at point n.  The element (nxmx2) y position of the mth UAV at
%point n.
%
%UAVS - is a 4xn matrix where n is number of UAVs, the first row is the
%initial x position of the UAVs, the second row is the initial y position
%of the UAVs, the third row is the initial altitude of the UAVs, and
%the fourth row is the initial Velocity of the UAVs.
%
%
%OUTPUTS:
%
%uav_path_x - is a mxn matrix where m is the number of UAVs and m is the
%length of the longest path. These are the x coordinates of the paths.
%
%uav_path_y - is a mxn matrix where m is the number of UAVs and m is the
%length of the longest path. These are the y coordinates of the paths.
%
%time_uav - is a mxn matrix where m is the number of UAVs and m is the
%length of the longest path. These values correspond to the time at which
%the uavs are at coordinates x and y in uav_path_x and uav_path_y.
%
%altitude_uav - is a mxn matrix where m is the number of UAVs and m is the
%length of the longest path. These values correspond to the altitudes that
%the UAVs are at when they are at coordinates x and y in uav_path_x and
```

```
%uav_path_y.
%
%Threat_range_vrt - is a 1xn vector where n is the number of threats, where
%the first row is the range of the threats at the altitude where the UAVs
%are flying.
%
%Zone_range_vrt - is a 1xn vector where n is the number of zones, where
%the first row is the range of the zones at the altitude where the UAVs
%are flying.

nuav=size(shr_x,2);
szshrpth=size(shr_x,1);
shr_x=[[shr_x];[shr_x(szshrpth,:)]];
shr_y=[[shr_y];[shr_y(szshrpth,:)]];
uav_path_x=zeros(nuav,szshrpth+1);
uav_path_y=zeros(nuav,szshrpth+1);
for i=1:nuav,
   for j=1:szshrpth,
      if [shr_x(j+1,i),shr_y(j+1,i)]==[shr_x(j,i),shr_y(j,i)] | j==szshrpth,
         lst_pnt_x=shr_x(j,i);
         nxtlst_pnt_x=shr_x(j-1,i);
         lst_pnt_y=shr_y(j,i);
         nxtlst_pnt_y=shr_y(j-1,i);
         dist_pnts=sqrt(((lst_pnt_x-nxtlst_pnt_x)^2)+((lst_pnt_y-nxtlst_pnt_y)^2));
         last_x=lst_pnt_x+((lst_pnt_x-nxtlst_pnt_x)*(distpast/dist_pnts));
         last_y=lst_pnt_y+((lst_pnt_y-nxtlst_pnt_y)*(distpast/dist_pnts));
         uav_path_x(i,[j+1:szshrpth+1])=last_x;
         uav_path_y(i,[j+1:szshrpth+1])=last_y;
         uav_path_x(i,j)=shr_x(j,i);
         uav_path_y(i,j)=shr_y(j,i);
         break
      else
         uav_path_x(i,j)=shr_x(j,i);
         uav_path_y(i,j)=shr_y(j,i);
      end
   end
end

%Initializing matrixes
time_uav_temp=zeros(nuav,szshrpth+1);
time_uav=zeros(nuav,szshrpth+1);
altitude_uav=zeros(nuav,szshrpth+1);

%Time matrix
for i=1:nuav,
   for j=1:szshrpth,
```

```
      shr_dist(i,j)=sqrt((uav_path_x(i,j)-uav_path_x(i,j+1))^2+(uav_path_y(i,j)-
         uav_path_y(i,j+1))^2);
      time_uav_temp(i,j+1)=shr_dist(i,j)/UAVS(4,i);
   end
   time_uav(i,[2:szshrpth+1])=sum(time_uav_temp(i,:));
   for j=2:szshrpth+1,
      time_uav(i,j)=time_uav(i,j-1)+time_uav_temp(i,j);
   end
end

time_uav=time_uav*1.01;

%Altitude matrix
for i=1:nuav,
   for j=1:szshrpth+1,
      altitude_uav(i,j)=UAVS(3,i);
   end
end
```

plot_uav.m

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

This file will plot the locations of the UAVs and their environment each time the method replans, during the simulation, if the plots are enabled.

```
functionplot_uav(UAVS,TARGETS,ZONES,THREATS,uav_path_x,uav_path_y,
         n_plots,uavs_existing,targ_existing,threats_existing)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Plotting results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(n_plots);
hold on;
for i=1:2,
   subplot(1,2,i),
   for i=1:size(UAVS,2)
      if uavs_existing(1,i)==1
         plot(UAVS(1,i),UAVS(2,i),'bd');
         text(UAVS(1,i)+5,UAVS(2,i),{i},'FontSize',12,'Color','b');
         axis([5 200 5 200]);
         hold on;
      end
   end
   for i=1:size(TARGETS,2)
      if targ_existing(1,i)==1
         plot(TARGETS(1,i),TARGETS(2,i),'x','Color',[0,.4,0]);
         text(TARGETS(1,i)+5,TARGETS(2,i),{i},'FontSize',12,'Color',[0,0.4,0]);
```

```
            axis([5 200 5 200]);
            hold on;
        end
    end
    for i=1:size(THREATS,2)
        if threats_existing(1,i)==1
            plot(THREATS(1,i),THREATS(2,i),'r*');
            text(THREATS(1,i)+5,THREATS(2,i),{i},'FontSize',12,'Color','r')
            axis([5 200 5 200]);
            hold on;
        end
    end
    hold on;
end

%Plotting threats and range
for i=1:size(THREATS,2)
    if threats_existing(1,i)==1
        t_threat = (1/32:1/32:1)'*2*pi;
        x_threat = THREATS(3,i)*sin(t_threat)+THREATS(1,i);
        y_threat = THREATS(3,i)*cos(t_threat)+THREATS(2,i);
        for i=1:2,
            subplot(1,2,i),plot(x_threat,y_threat,'r.');hold on;
        end
    end
end

%Plotting No fly Zones
for i=1:size(ZONES,2)
    t_nfz = (1/16:1/16:1)'*2*pi;
    x_nfz = ZONES(3,i)*sin(t_nfz)+ZONES(1,i);
    y_nfz = ZONES(3,i)*cos(t_nfz)+ZONES(2,i);
    for i=1:2,
        subplot(1,2,i),fill(x_nfz,y_nfz,'k');hold on;
    end
end

%Plotting shortened paths
for i=1:size(uav_path_x,1)
    subplot(1,2,2),plot(uav_path_x(i,:),uav_path_y(i,:),'b-');hold on;
end
subplot(1,2,1),title('Initial Positions');hold on;
subplot(1,2,2),title('Shortened Selected Paths');hold on;
for i=1:2,
    subplot(1,2,i),axis([-25 250 -25 250]);hold on;
end
```

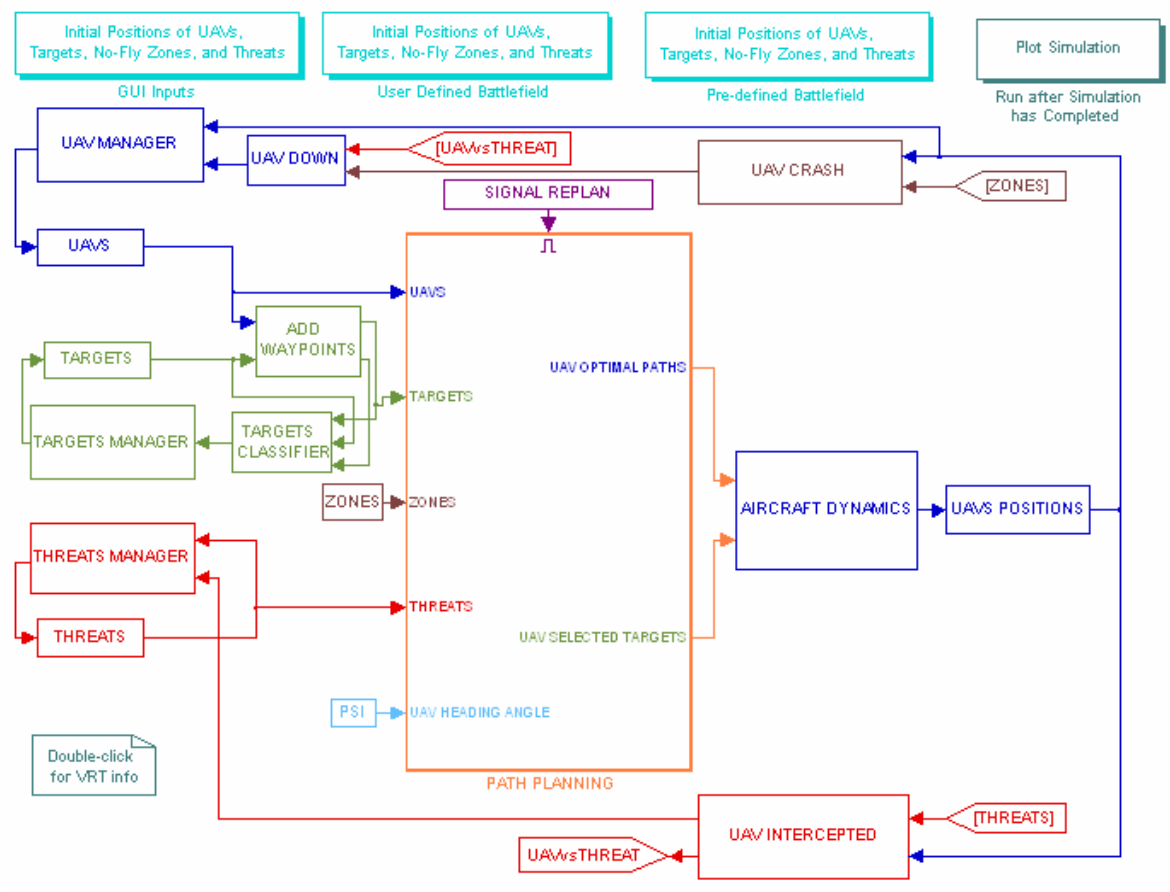
Appendix B: Main SIMULINK diagrams for the Partially Decoupled Simulation

This model is the top level for the partially decoupled simulation. The GUI Inputs, User Defined Battlefield, and Pre-defined Battlefield all offer different ways to input the necessary data. The Plot Simulation button will show the paths the vehicles actually take in their quest to fulfill the mission.

pathplan.mdl

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

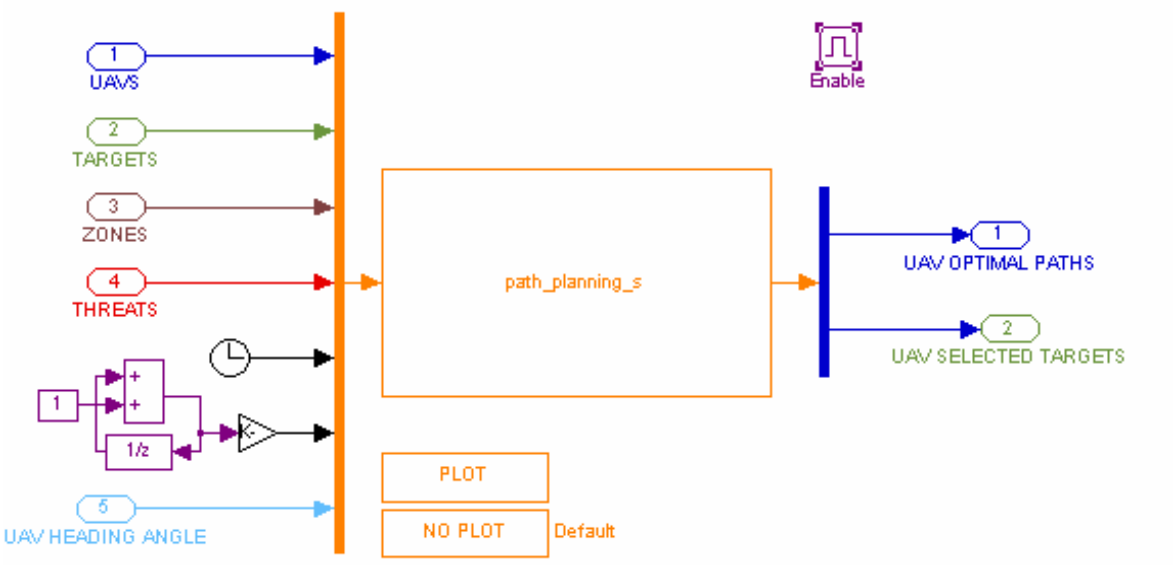
The main model for the path planning and task allocation simulation of the partially decoupled system.



pathplan/ PATH PLANNING

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

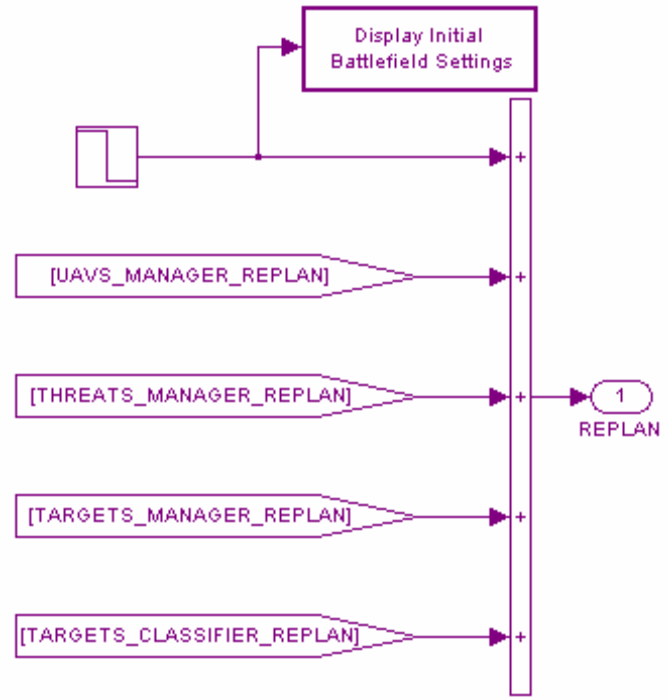
The path planning subsystem calls the path_planning.m file.



pathplan/SIGNAL REPLAN

Authored by Zachary Spritzer, and Matthew Lechliter

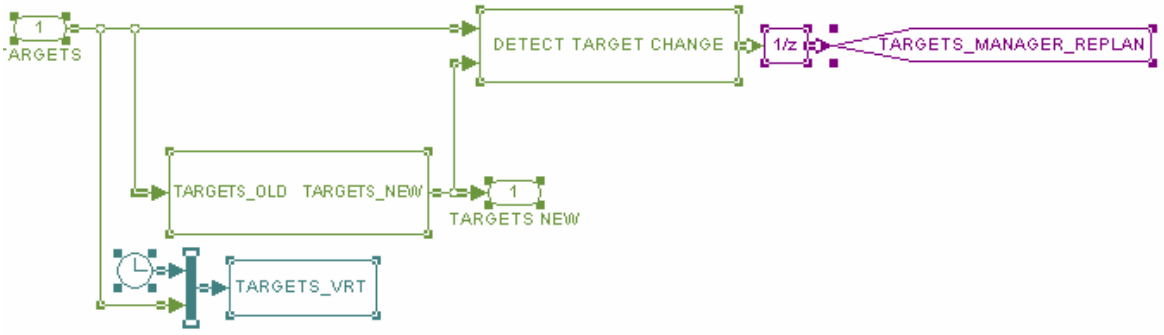
This code is called every time a replan is signaled.



pathplan/TARGETS MANAGER

Authored by Zachary Spritzer, and Matthew Lechliter

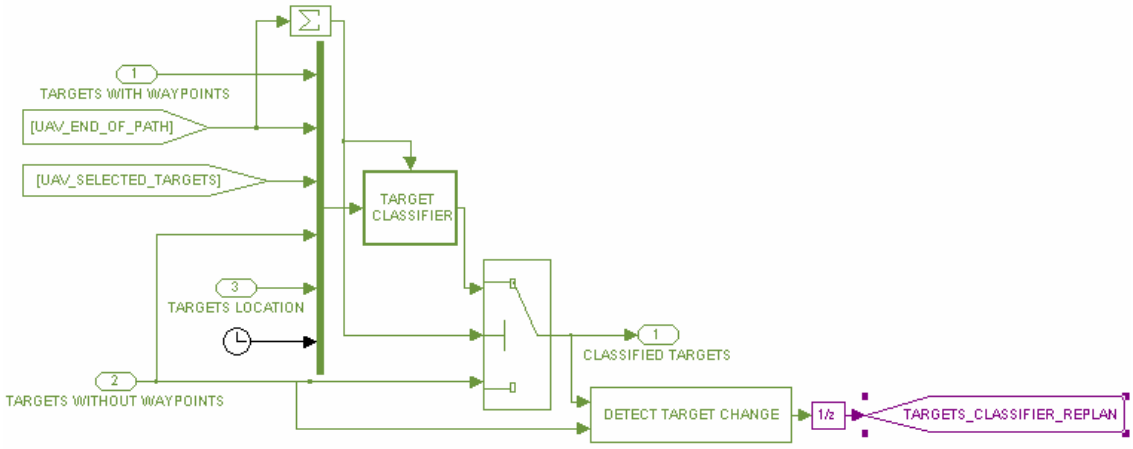
The target manager subsystem is responsible for identifying when a target changes state and signaling an appropriate replan.



pathplan/TARGETS CLASSIFIER

Authored by Zachary Spritzer, and Matthew Lechliter

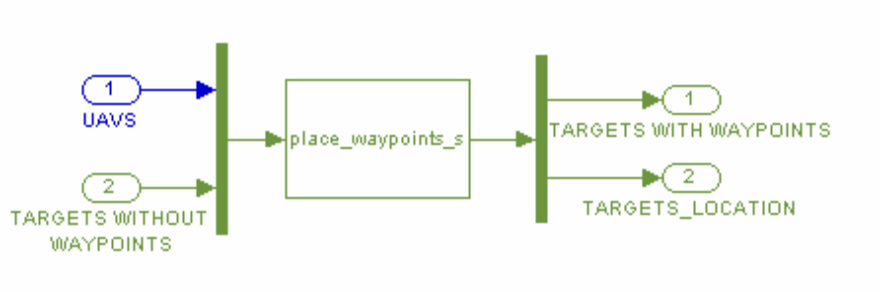
The targets classifier subsystem determines when a target is classified and signals appropriate replans.



pathplan/ADD WAYPOINTS

Authored by Jennifer Hazelton, Zachary Spritzer, and Matthew Lechliter

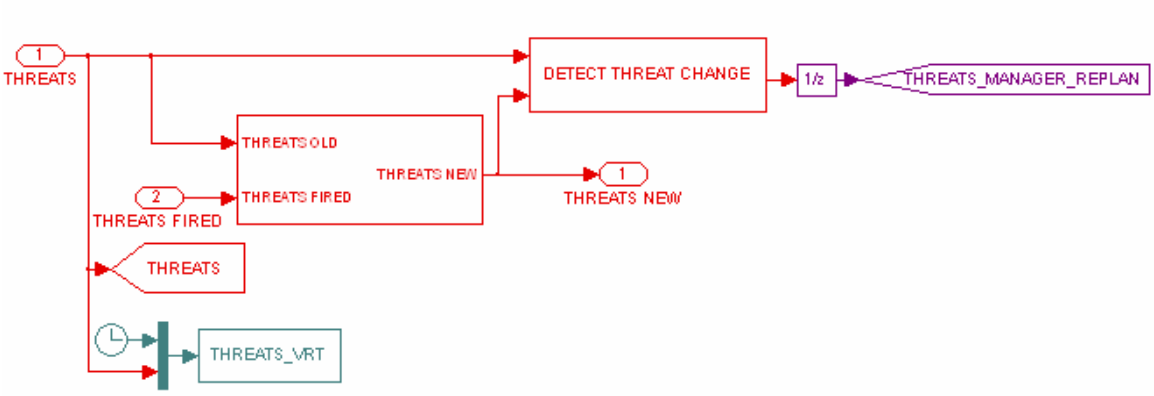
When the number of targets is less than the number of vehicles, the ADD WAYPOINTS subsystem adds targets with a value of zero until the targets equal the vehicles in number.



pathplan/THREATS MANAGER

Authored by Zachary Spritzer, and Matthew Lechliter

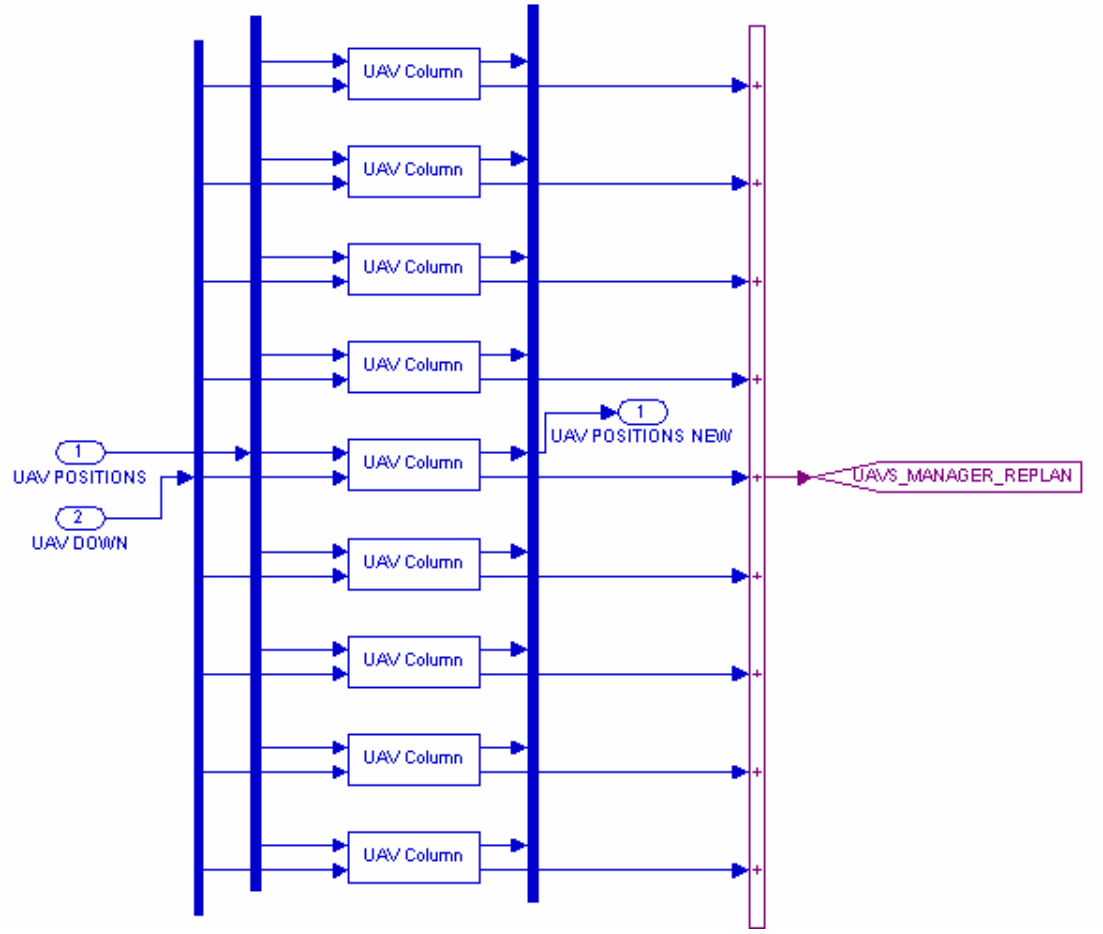
The threats manager subsystem, in Figure 15, determines when a threat fires and signals a appropriate replans.



pathplan/UAV MANAGER

Authored by Zachary Spritzer, and Matthew Lechliter

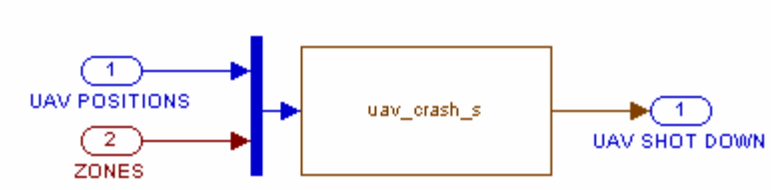
The UAV Manager subsystem determines when a vehicle has been destroyed and signals appropriate replans.



pathplan/UAV CRASH

Authored by Zachary Spritzer, and Matthew Lechliter

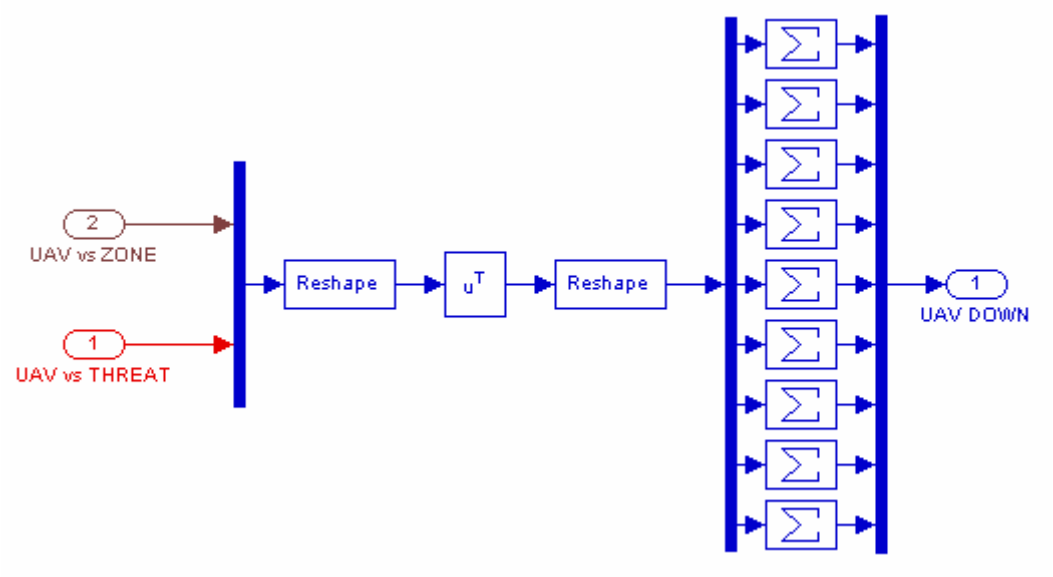
The UAV crash subsystem determines if a UAV's path intersects a no-fly zone. If it does, then the vehicle is considered lost.



pathplan/UAV DOWN

Authored by Zachary Spritzer, and Matthew Lechliter

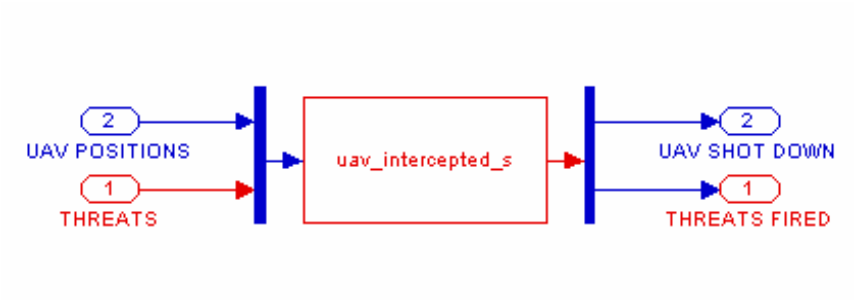
The UAV down subsystem detects whether a vehicle has been shot down by a threat.



pathplan/UAV DOWN

Authored by Zachary Spritzer, and Matthew Lechliter

The UAV intercepted subsystem determines whether a threat hits a vehicle.



Appendix C: Model file for the Combined Method

Authored by Jennifer Hazelton

```
### Combined path planning and task allocation UAV problem ###
### used for each scenario ###

# sets of data, corresponding to indices
set Nv;         # number of vehicles
set Nw;         # number of targets
set Nz;         # number of no-fly zones
set Nk;         # number of corners on a no-fly zone
set Pos;        # x-position, y-position of targets

# parameters, data for the model
param Nt      >0;     # number of time steps
param posmax >0;      # maximum position of vehicle from origin
param vmax    >0;     # maximum velocity of vehicle
param fmax    >0;     # maximum force on vehicle
param sigma1 >0;      # weighting factor
param sigma2 >0;      # weighting factor
param x0 {Nv};        # initial values of x-location
param y0 {Nv};        # initial values of y-location
param xdot0 {Nv};     # initial values of x-velocity
param ydot0 {Nv};     # initial values of y-velocity
param Z {Nz,Nk};      # locations of NFZ
param R;              # relaxation value in NFZ and target constraints
param W {Nw,Pos};     # locations of targets
param attack;         # number of times to attack each target

# decision variables
var tp {p in Nv} integer >=0, <=30;                    # vehicle completion time
var tbar integer >=0, <=35;                            # overall mission completion time
var b {i in Nw, p in Nv, t in 1..Nt} binary;           # target state indicator
var d {j in Nz, k in Nk, p in Nv, t in 1..Nt} binary;  # no-fly zone indicator
var x {p in Nv, t in 0..Nt} >= 0, <= posmax;           # x-location of vehicle
var y {p in Nv, t in 0..Nt} >= 0, <= posmax;           # y-location of vehicle
var xdot {p in Nv, t in 0..Nt} >= -vmax, <= vmax;      # x-velocity of vehicle
var ydot {p in Nv, t in 0..Nt} >= -vmax, <= vmax;      # y-velocity of vehicle
var fx {p in Nv, t in 0..Nt} >= -fmax, <= fmax;        # x-force on vehicle
var fy {p in Nv, t in 0..Nt} >= -fmax, <= fmax;        # y-force on vehicle

# objective function
minimize mission_time: tbar+ sigma1*(sum {p in Nv} tp[p]);
```

```
# constraints

# initial condition
subject to initial_cond1 {p in Nv}: x[p,0]    = x0[p];
subject to initial_cond2 {p in Nv}: y[p,0]    = y0[p];
subject to initial_cond3 {p in Nv}: xdot[p,0] = xdot0[p];
subject to initial_cond4 {p in Nv}: ydot[p,0] = ydot0[p];

# constraint 1
subject to state_vector_1 {p in Nv, t in 0..Nt-1}:
        x[p,t+1]   = x[p,t]+xdot[p,t]+5.8824*fx[p,t];
subject to state_vector_2 {p in Nv, t in 0..Nt-1}:
        y[p,t+1]   = y[p,t]+ydot[p,t]+5.8824*fy[p,t];
subject to state_vector_3 {p in Nv, t in 0..Nt-1}: xdot[p,t+1] = xdot[p,t]+11.765*fx[p,t];
subject to state_vector_4 {p in Nv, t in 0..Nt-1}: ydot[p,t+1] = ydot[p,t]+11.765*fy[p,t];

# constraint 2, one line for every m in 1..M
subject to max_force_1  {p in Nv, t in 0..Nt}:
        fx[p,t]*0.58779    + fy[p,t]*0.80902          <= fmax;
subject to max_force_2  {p in Nv, t in 0..Nt}:
        fx[p,t]*0.95106    + fy[p,t]*0.30902          <= fmax;
subject to max_force_3  {p in Nv, t in 0..Nt}:
        fx[p,t]*0.95106    + fy[p,t]*(-0.30902)       <= fmax;
subject to max_force_4  {p in Nv, t in 0..Nt}:
        fx[p,t]*0.58779    + fy[p,t]*(-0.80902)       <= fmax;
subject to max_force_5  {p in Nv, t in 0..Nt}:
        fx[p,t]*0          + fy[p,t]*(-1)             <= fmax;
subject to max_force_6  {p in Nv, t in 0..Nt}:
         fx[p,t]*(-0.58779) + fy[p,t]*(-0.80902)      <= fmax;
subject to max_force_7  {p in Nv, t in 0..Nt}:
        fx[p,t]*(-0.95106) + fy[p,t]*(-0.30902)      <= fmax;
subject to max_force_8  {p in Nv, t in 0..Nt}:
        fx[p,t]*(-0.95106) + fy[p,t]*0.30902         <= fmax;
subject to max_force_9  {p in Nv, t in 0..Nt}:
        fx[p,t]*(-0.58779) + fy[p,t]*0.80902         <= fmax;
subject to max_force_10  {p in Nv, t in 0..Nt}:
        fx[p,t]*0          + fy[p,t]*1               <= fmax;

# constraint 3
subject to max_velocity_1  {p in Nv, t in 0..Nt}:
        xdot[p,t]*0.58779    + ydot[p,t]*0.80902          <= vmax;
subject to max_velocity_2  {p in Nv, t in 0..Nt}:
        xdot[p,t]*0.95106    + ydot[p,t]*0.30902          <= vmax;
subject to max_velocity_3  {p in Nv, t in 0..Nt}:
        xdot[p,t]*0.95106    + ydot[p,t]*(-0.30902)       <= vmax;
```

```
subject to max_velocity_4  {p in Nv, t in 0..Nt}:
        xdot[p,t]*0.58779    + ydot[p,t]*(-0.80902)          <= vmax;
subject to max_velocity_5  {p in Nv, t in 0..Nt}:
        xdot[p,t]*0          + ydot[p,t]*(-1)                <= vmax;
subject to max_velocity_6  {p in Nv, t in 0..Nt}:
         xdot[p,t]*(-0.58779) + ydot[p,t]*(-0.80902)         <= vmax;
subject to max_velocity_7  {p in Nv, t in 0..Nt}:
        xdot[p,t]*(-0.95106) + ydot[p,t]*(-0.30902)         <= vmax;
subject to max_velocity_8  {p in Nv, t in 0..Nt}:
        xdot[p,t]*(-0.95106) + ydot[p,t]*(0.30902)          <= vmax;
subject to max_velocity_9  {p in Nv, t in 0..Nt}:
        xdot[p,t]*(-0.58779) + ydot[p,t]*0.80902            <= vmax;
subject to max_velocity_10 {p in Nv, t in 0..Nt}:
        xdot[p,t]*0          + ydot[p,t]*1                  <= vmax;

# constraint 5
subject to noflyzone_1 {t in 1..Nt, p in Nv, j in Nz}: x[p,t] - Z[j,'xr'] >= -R*d[j,'xl',p,t];
subject to noflyzone_2 {t in 1..Nt, p in Nv, j in Nz}: Z[j,'xl'] - x[p,t] >= -R*d[j,'yl',p,t];
subject to noflyzone_3 {t in 1..Nt, p in Nv, j in Nz}: y[p,t] - Z[j,'yr'] >= -R*d[j,'xr',p,t];
subject to noflyzone_4 {t in 1..Nt, p in Nv, j in Nz}: Z[j,'yl'] - y[p,t] >= -R*d[j,'yr',p,t];
subject to noflyzone_5 {t in 1..Nt, p in Nv, j in Nz}:  sum {k in Nk} d[j,k,p,t] <= 3;

# constraint 6
subject to target_1 {p in Nv, t in 1..Nt, i in Nw}: x[p,t] - W[i,'xpos'] <= R*(1-b[i,p,t]);
subject to target_2 {p in Nv, t in 1..Nt, i in Nw}: x[p,t] - W[i,'xpos'] >= -R*(1-b[i,p,t]);
subject to target_3 {p in Nv, t in 1..Nt, i in Nw}: y[p,t] - W[i,'ypos'] <= R*(1-b[i,p,t]);
subject to target_4 {p in Nv, t in 1..Nt, i in Nw}: y[p,t] - W[i,'ypos'] >= -R*(1-b[i,p,t]);

# constraint ensure targets are hit, right hand side indicates how many times the vehicle
#must be hit
subject to hit_target1: sum{p in Nv} (sum {t in 1..Nt} b['targ1',p,t]) = attack;
subject to hit_target2: sum{p in Nv} (sum {t in 1..Nt} b['targ2',p,t]) = attack;

# constraint 7
subject to vehicle_completion_t {p in Nv, i in Nw}: tp[p] >= sum {t in 1..Nt} t*b[i,p,t];


# constraint 8
subject to mission_completion_t {p in Nv}: tbar >= tp[p];
```

Appendix D: Data file for the Combined Method

Authored by Jennifer Hazelton

Scenario 1

Data file for path planning and task allocation

```
# sets of data, or the indices explicitly
set Nv:= uav1 uav2;             # number of UAVs
set Nz:= nfz1 nfz2 nfz3;        # number of no-fly zones
set Nk:= xl yl xr yr;           # corners of a no-fly zone
set Nw:= targ1 targ2;           # number of targets
set Pos:= xpos ypos;            # x- and y-location of targets

param Nt       := 3;            # number of time steps
param posmax:= 70;              # m, maximum position value of a UAV
param vmax     := 13.41;        # m/s, maximum velocity Roskam (max=267m/s)
param fmax     := 2.35;         # N, Roskam
param sigma1  := 0.001;         # weighting factor
param attack   := 1;            # number of times to attack each target
param R        := 100000;       # relaxation value in NFZ and target constraints

param :        x0      y0      xdot0   ydot0   :=
 uav1   15     14      10      1
 uav2   12     17      5       3       ;

param Z:       xl      yl      xr      yr      :=
  nfz1         16.4    17.5    21.6    20.5
  nfz2         17.5    16.4    20.5    21.6
  nfz3         16.88   16.88   21.12   21.12;

param W:       xpos    ypos    :=
  targ1        30      27
  targ2        27      30      ;
```

Scenario 2

Data file for path planning and task allocation

```
# sets of data, or the indices explicitly
set Nv:= uav1 uav2;             # number of UAVs
set Nz:= nfz1 nfz2 nfz3;        # number of no-fly zones
set Nk:= xl yl xr yr;           # corners of a no-fly zone
set Nw:= targ1 targ2;           # number of targets
set Pos:= xpos ypos;            # x- and y-location of targets

param Nt       := 3;            # number of time steps
param posmax:= 70;              # m, maximum position value of a UAV
```

```
param vmax   := 13.41;      # m/s, maximum velocity Roskam (max=267m/s)
param fmax   := 2.35;       # N, Roskam
param sigma1 := 0.001;      # weighting factor
param attack := 1;          # number of times to attack each target
param R      := 100000;     # relaxation value in NFZ and target constraints

param :      x0     y0     xdot0  ydot0  :=
 uav1        10     5      12     1
 uav2        5      10     2      9.25   ;

param Z:     xl     yl     xr     yr     :=
  nfz1       16.4   17.5   21.6   20.5
  nfz2       17.5   16.4   20.5   21.6
  nfz3       16.88  16.88  21.12  21.12;

param W:     xpos   ypos   :=
  targ1      30     27
  targ2      27     30     ;
```

Scenario 3

```
#### Data file for path planning and task allocation ####

# sets of data, or the indices explicitly
set Nv:= uav1 uav2;         # number of UAVs
set Nz:= nfz1 nfz2 nfz3;    # number of no-fly zones
set Nk:= xl yl xr yr;       # corners of a no-fly zone
set Nw:= targ1 targ2;       # number of targets
set Pos:= xpos ypos;        # x- and y-location of targets

param Nt     := 3;          # number of time steps
param posmax := 70;         # m, maximum position value of a UAV
param vmax   := 13.41;      # m/s, maximum velocity Roskam (max=267m/s)
param fmax   := 2.35;       # N, Roskam
param sigma1 := 0.001;      # weighting factor
param attack := 1;          # number of times to attack each target
param R      := 100000;     # relaxation value in NFZ and target constraints

param :      x0     y0     xdot0  ydot0  :=
 uav1        18     12     6      7
 uav2        12     18     1      4      ;

param Z:     xl     yl     xr     yr     :=
  nfz1       16.4   17.5   21.6   20.5
  nfz2       17.5   16.4   20.5   21.6
  nfz3       16.88  16.88  21.12  21.12;
```

```
param W:      xpos   ypos   :=
 targ1        30     27
 targ2        27     30     ;
```

Scenario 4

```
#### Data file for path planning and task allocation ####

# sets of data, or the indices explicitly
set Nv:= uav1 uav2;             # number of UAVs
set Nz:= nfz1 nfz2 nfz3;        # number of no-fly zones
set Nk:= xl yl xr yr;           # corners of a no-fly zone
set Nw:= targ1 targ2;           # number of targets
set Pos:= xpos ypos;            # x- and y-location of targets

param Nt       := 3;            # number of time steps
param posmax:= 70;              # m, maximum position value of a UAV
param vmax     := 13.41;        # m/s, maximum velocity Roskam (max=267m/s)
param fmax     := 2.35;         # N, Roskam
param sigma1 := 0.001;          # weighting factor
param attack   := 1;            # number of times to attack each target
param R        := 100000;       # relaxation value in NFZ and target constraints

param :        x0     y0     xdot0  ydot0   :=
 uav1          20.5   15     13.41  0
 uav2          15     20.5   0      13.41   ;

param Z:       xl     yl     xr     yr      :=
 nfz1          16.4   17.5   21.6   20.5
 nfz2          17.5   16.4   20.5   21.6
 nfz3          16.88  16.88  21.12  21.12;

param W:      xpos   ypos   :=
 targ1        30     27
 targ2        27     30     ;
```

Scenario 5

```
#### Data file for path planning and task allocation ####

# sets of data, or the indices explicitly
set Nv:= uav1 uav2;             # number of UAVs
set Nz:= nfz1 nfz2 nfz3;        # number of no-fly zones
set Nk:= xl yl xr yr;           # corners of a no-fly zone
set Nw:= targ1 targ2;           # number of targets
set Pos:= xpos ypos;            # x- and y-location of targets

param Nt       := 3;            # number of time steps
```

```
param posmax := 70;          # m, maximum position value of a UAV
param vmax   := 13.41;       # m/s, maximum velocity Roskam (max=267m/s)
param fmax   := 2.35;        # N, Roskam
param sigma1 := 0.001;       # weighting factor
param attack := 1;           # number of times to attack each target
param R      := 100000;      # relaxation value in NFZ and target constraints

param :      x0     y0     xdot0  ydot0  :=
 uav1        6      12     2      10
 uav2        12     6      13     0      ;

param Z:     xl     yl     xr     yr     :=
  nfz1       16.4   17.5   21.6   20.5
  nfz2       17.5   16.4   20.5   21.6
  nfz3       16.88  16.88  21.12  21.12;

param W:     xpos   ypos   :=
  targ1      30     27
  targ2      27     30     ;
```