

Allocation of Radar Resources to Maximize Tracker Information

15 August 2002

Kevin J. Sullivan
Craig S. Agate
Harry L. Burger

Toyon Research Corporation

75 Aero Camino, Suite A
Goleta, CA 93117-3139

Abstract

This paper describes the development and evaluation of a dynamic sensor-tasking module that plans future sensor schedules such that the expected amount of information contained within a track database is maximized. The amount of information is quantified using the concept of entropy for each target classification vector in each track in the database. At each processing event, all tracks are extrapolated forward in time to match the time of the candidate future sensor tasks. This allows for the calculation of the expected entropy given a candidate sensor task. Many potential tasks are evaluated to select the resulting schedule. Moving as well as stationary targets are considered. Possible radar modes include GMTI/LRR (Low Range Resolution), GMTI/HRR (High Range Resolution), or spotlight SAR (Synthetic Aperture Radar). The module is evaluated and demonstrated in a simple scenario using the SLAMEM simulation.

1 INTRODUCTION

The United States is procuring many new intelligence, surveillance, and reconnaissance (ISR) assets that may provide future commanders with a strategic advantage. New platforms such as the Global Hawk will provide the ability to position sensors close to the battlefield. New sensor systems, such as agile radars, will provide greater flexibility in terms of the modes and beam positions that can be selected during a given time period. Along with this new flexibility comes a challenge to develop algorithms that effectively control the sensors.

Sensor tasking algorithms have been researched for many years. Much of the focus of these algorithms has been aimed at improving or maintaining kinematic state information on a given number of targets (e.g., [1], [2]). Less work has been done in the area of developing tasking algorithms that support the buildup and maintenance of identification information. Musick and Kastella [3] presented an analysis of several different techniques for selecting between sensor modes and pointing locations. A technique they referred to as “discrimination gain” showed the most potential. This technique selects the best sensor task based on the largest change in the expected entropy of the classification vector of a number of stationary targets. In the research effort described here, we have applied this concept to the problem of locating and identifying ground targets that may be moving or stationary. We have created a dynamic sensor tasking module that works with a tracking module to plan sensor schedules that maintain and buildup identification information using radars capable of operating in a synthetic aperture (SAR) or ground-moving-target-indicator (GMTI) mode.

Our approach seeks to maximize the amount of information that is contained in a track database containing the estimated location and identity of ground vehicles operating in an area of interest. We quantify the amount of information in the database using entropy. Our approach is further described in Section 2. In order to develop, test, and demonstrate our dynamic tasking module, we have created two testbeds. These testbeds are described in Section 3. In order to demonstrate the operation of the algorithms developed in this effort, we have quantified the performance of them on a simple example problem that is described in Section 4. We finish this paper with a summary in Section 5.

2 TECHNICAL DESCRIPTION OF TASKING MODULE

2.1 ISR Architecture and Concept of Operations (CONOPS)

Dynamic tasking systems are strongly influenced by the way in which the elements of a tasking, processing, exploitation, and dissemination (TPED) architecture operate. For example, if each asset in an architecture is “stove-piped”, then the dynamic tasking algorithm can operate independently for each asset and the problem to be solved is much simpler. If, on the other hand, the assets share information and can be tasked in light of this shared information, then the tasking algorithms can consider the coupled effects of each platform. This is a much more challenging problem for sensor tasking algorithms, but should result in improved performance [4]. We plan to focus on the latter case of shared information and discuss our expected CONOPS below.

The elements of a surveillance architecture that shares information across several platforms are shown in Figure 1. The figure shows a few examples of sensors and their platforms including manned and unmanned aircraft and satellites equipped with Synthetic Aperture Radar (SAR)/Moving Target Indicator (MTI) radars. We expect that each platform will send its output to a local exploitation cell that will process the received signals and make detections, and possibly

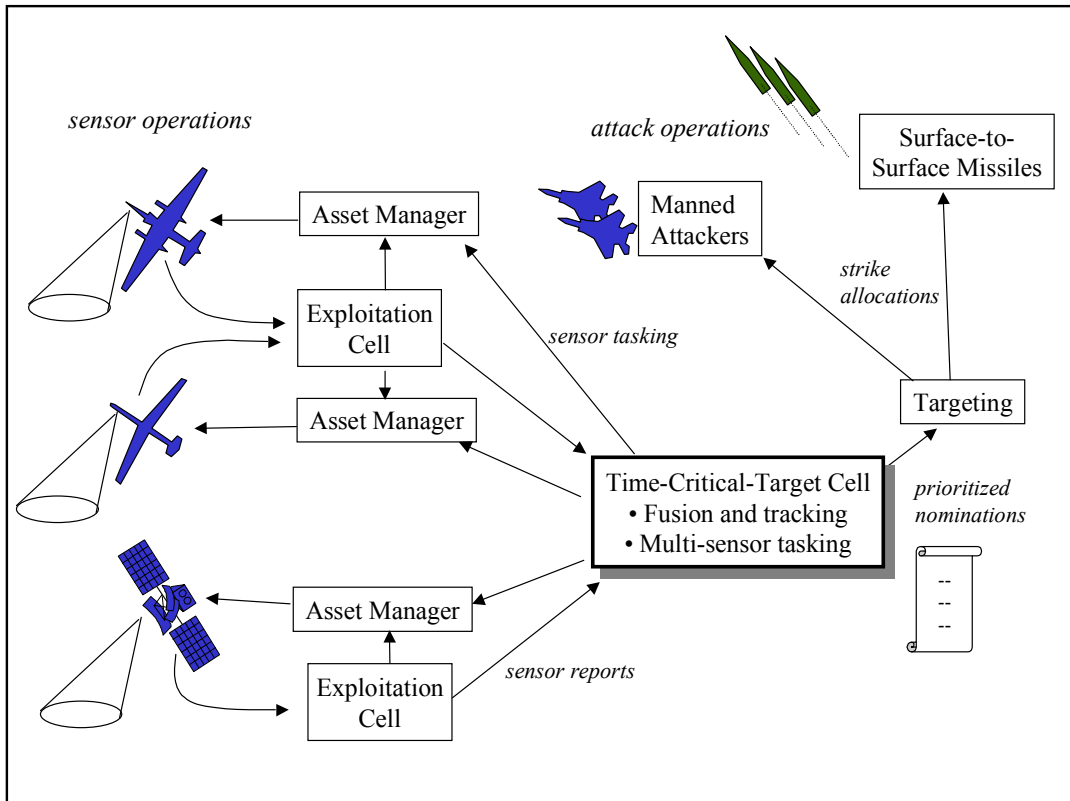


Figure 1. ISR Architecture

classifications of vehicles. These local exploitation cells may exist onboard the aircraft (e.g., JSTARS), or they may exist as detached units (e.g., the Contingency Airborne Reconnaissance System (CARS)). More than one platform may report to a single local exploitation cell.

Each local exploitation cell reports its detections to a unit that we generically refer to as a fusion center or time-critical-target cell. Tracking algorithms and the dynamic tasking algorithms exist at the fusion center. Here, the detections collected from the multiple platforms are fused and associated to form tracks of vehicles in the area. These tracks include information regarding the estimated class or identity of each vehicle. This information is used to decide where the sensors should look next.

The location of the fusion center is not important as long as it is able to rapidly receive the detections output by the local exploitation cells. In some cases, the fusion center may be collocated with the local exploitation functions. For example, a JSTARS operating with an

Unmanned Air Vehicle (UAV) may have both a local exploitation cell and the fusion center onboard the JSTARS. For other sensors, the fusion center is likely to be ground-based.

The dynamic tasking algorithms use the detections and resulting tracks from all of the sensors to request additional sensor tasks. The task requests are sent to the local asset managers that do the actual scheduling of the individual sensors. At some point in this process, the dynamic task requests must be deconflicted with other tasks that have previously been scheduled for missions such as Battle Damage Assessment (BDA). This can be handled by operating with a sensor schedule that has been created prior to the execution of rapid dynamic tasking. This schedule should contain a priority for each task. The priority of a dynamic task must exceed the priority of the tasks already in the schedule in order to displace them. The difficulty with this process lies in the method used to quantify the priorities given the needs of many commanders operating at different echelons.

We envision a process where the fusion center interacts with the local asset managers when planning schedules. The fusion center will contain a module that we will refer to as the global asset manager. The global asset manager optimizes tasking across all of the sensors that report to the fusion center. The interaction between the global asset manager and the local asset managers could take place by having a copy of each asset manager's software resident in the fusion center or by communicating with the asset managers over a secure computer network. The interaction will consist of queries in which the global asset manager asks for the amount of time required to perform various sensor tasks. The time computed by the local asset manager should include any slewing required by the sensor and any time required for rerouting the sensor platform. This time can then be used by the global asset manager to create ordered lists of tasks for each sensor based on the timing provided by the individual asset managers. This allows the details of each sensor to be handled by the appropriate asset manager without any loss of generality in planning the global schedule.

We envision that the tasking algorithms must support a timeline that consists of a regular sequence of processing events. This is driven by several factors, including the fact that the situation on the battlefield is very dynamic and, when multiple types of sensors are being used to track large numbers of vehicles, it would be computationally prohibitive to develop a sensor schedule prior to the start of the battle that takes into account all of the possible events that may occur. This prevents the use of what has been referred to as "closed-loop" [5] sensor tasking algorithms. We envision only open-loop feedback approaches will be feasible. With the open-loop feedback approach, sensor tasks are computed at regular intervals after receiving new information from recent sensor measurements. Although the sensor schedules are computed multiple times, at each computation the scheduling horizon needs only be past the time of the next processing event. This process can be better understood by considering Figure 2. The figure shows three timelines. The top line represents the collection of measurements by the sensors. The middle line represents the processing of each batch of sensor measurements by the exploitation and fusion systems, and the bottom line represents the processing of the exploitation and fusion output by the dynamic tasking algorithms. The label on the top of the sensor timeline indicates when the sensor schedule that is being used was created. At startup, a default schedule (D) is employed until sufficient sensor measurements are collected to make decisions about future tasks. The sensor measurements created by running this default schedule can be grouped into batches (bn) that are labeled on the bottom of the sensor timeline. Each sensor batch is sent to the exploitation and fusion processors (note down-arrow for batch 1). The labels below the

exploitation and fusion timeline indicate the sensor batch that is being processed. When exploited and fused, the data is sent to the dynamic tasking processor (down-arrow for batch 1). The dynamic tasking processor will create new schedules for the future. At the end of a processing interval, the new sensor schedules (S_n) are sent to the sensors for implementation (up-arrow for schedule created in response to batch 1 sensor measurements). After a brief startup period during which the default schedule is employed, the dynamic tasker continues to create sensor schedules for the future using sensor measurements from the recent past.

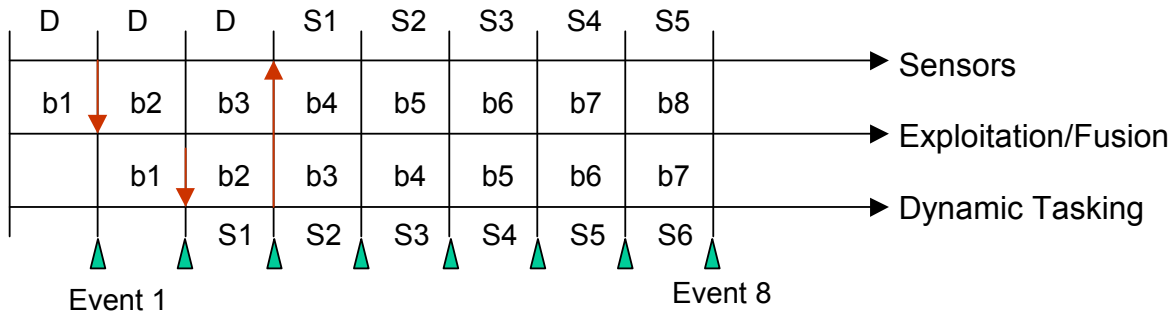


Figure 2. Notional Processing Timeline

The amount of time selected between fusion center processing events will be driven by the speed of the dynamic tasking algorithms at the global asset manager, the exploitation times required, the speed of the fusion algorithms, and the speed of the individual asset managers. A large processing interval is better for tracking algorithms because they perform best if all of the detections can be considered in one big batch. On the other hand, a small processing interval is desired so that up-to-date information is available about the current location of certain vehicles. Once a good interval of time is selected, this will set a requirement for the speed of the real-time tasking algorithms since they must finish before the next processing event arrives. Of course, since the dynamic scheduling algorithm is submitting a new schedule at each processing event, it only needs to compute a new schedule that has a time horizon that is just past the time of the next processing event.

2.2 Tasks and Schedules

The dynamic sensor-tasking problem can be formulated many different ways, and no clearly best way is evident. A first step in formulating the problem is to define the notion of a sensor task and establish the domain of all sensor tasks of interest. For a Ground Moving Target Indicator (GMTI) sensor, a sensor task could simply be a location to which the beam is pointed and the length of time required to collect sufficient signal-to-noise at that location. In other words, it could be a single dwell. The domain of tasks that are defined in this way could contain the continuous space of all beam positions within the sensor field-of-regard. Developing an algorithm that chooses from amongst this continuous set would be difficult to make robust and fast. An alternative is to discretize the possible beam position locations so that a tasking algorithm only needs to select from amongst a finite set of possibilities. We have taken this approach. Furthermore, we have lumped the set of beam positions into groups so that the number of tasks in the domain of sensor tasks is reduced. Our algorithm includes a parameter that defines the size of

the region that will be considered for a GMTI sensor task; a large value provides faster run times at the expense of less fidelity in the resulting sensor schedule.

For mechanically scanned radars, large groupings of sensor dwells into tasks are typically more optimal because the optimum schedule will tend to look like this anyways. This is because of the large slew times required to cover non-adjacent portions of the sensor's field-of-regard.

Instead of dividing the sensor's field-of-regard into a set of possible tasks, the tasks could be discretized by dividing the battlespace into a set of desired tasks. The location(s) of the desired tasks on the battlefield could be determined using information such as the expected location(s) of targets in track. The advantage to this approach is that the number of possible tasks from which to choose will be much less in sparse target situations. In dense target situations, however, the number of possibilities becomes very large and it will be better to discretize the field-of-regard of the sensor. An additional advantage to discretizing the field-of-regard of the sensor is that all of the tasks within this domain are feasible while some tasks in the battlespace discretization may not be feasible.

When considering spot SAR tasks, we work with the battlespace and consider locations of stopped tracks. Since the number of stopped tracks will tend to be fairly sparse, we elected to work with this domain rather than discretize the field-of-regard of the sensor. For the GMTI sensor we expect the density of objects of interest to be much greater and we thus search through the field-of-regard of the sensor.

Once the domain of tasks has been defined, a schedule can be made that is simply composed of a set of tasks that occur over a time window. Each task, including dwell, sensor slewing, and possibly platform routing, will require a certain amount of time. The length of time required to perform a task will be a function of the current states of the sensor and the platform. Since these states vary with time, the time required to perform a task will be a function of time. The dynamic scheduling problem can thus be defined as finding ordered sets of tasks for a set of sensors that minimizes or maximizes an objective function.

2.3 Objective Function

A key element of the dynamic tasking problem is the selection of a meaningful objective function. For scenarios in which we seek to keep track of ground targets, it is desired to know the location and identity of all vehicles on the battlefield at all times. Knowledge of the location of all vehicles can be quantified using the state covariance matrix of each track sampled at a regular time interval. These samples could be averaged over time to arrive at a single number that quantifies target location accuracy.

The quantification of the knowledge of vehicle identity, however, is not so straightforward. The identity of a vehicle will be determined using a set of measurements that have been collected on the vehicle over time. These measurements will be fused to compute a likelihood vector that quantifies the likelihood that the vehicle in track belongs to a given number of classes, possibly including an unknown class. A vector that contains one element with a large value is one whose identity is fairly certain based on the measurements and the fusion process. One objective is thus to get all tracks into this situation. Another objective is that once tracks are in this situation, it would be good to maintain this knowledge by correctly associating the tracks with future

measurements. In other words, the objective function should be able to encourage both the accumulation and maintenance of Identification (ID).

We have developed an objective function that encourages both the accumulation and maintenance of ID. This objective function uses the concept of entropy. Entropy is a means of quantifying the amount of information contained within a track. Entropy for a track's classification vector can be defined as

$$H = -\sum_{\omega_i} p_{\omega_i} \log p_{\omega_i}$$

where p_{ω_i} is the probability that the track belongs to class ω_i . This probability vector is computed by accumulating evidence using each measurement that is associated with a track. It is provided to the sensor tasker by a separate fusion and tracking process. The entropy will be largest for classification vectors that have all of their elements equal to one divided by the number of elements. This represents no information regarding classification. Entropy will be zero for a classification vector that has one element equal to one and all of the other elements equal to zero. This represents complete classification information. Our objective function is simply the sum of the expected entropy of each track *after* the candidate schedule has been implemented. This requires that we estimate the future states of all of the tracks as well as the sensor. The expected entropy of each track is weighted by the track's classification vector. This can be used to give more weight to tracks that are classified as targets of interest. Mathematically, this is given by

$$Obj = \sum_{i=1}^{\# \text{ of tracks}} \hat{H}_i \sum_{j \in \text{Targets}} p_{ij}$$

where \hat{H}_i is the expected entropy of track i after a candidate schedule has been applied and the summation provides a weighting based on the classification vector of the track.

The expected entropy of each track is influenced by the application of new classification information resulting from the candidate sensor schedule, the misassociation of tracks, and the dropping of tracks. First, consider the application of new classification information. For this case, we estimate the expected entropy by considering all of the possible classification outcomes weighted by their probability of occurring. This produces

$$\hat{H} = \sum_j \left[\left(-\sum_i \frac{p_{\omega_i} C_{ij}}{\sum_k p_{\omega_k} C_{kj}} \log \frac{p_{\omega_i} C_{ij}}{\sum_k p_{\omega_k} C_{kj}} \right) \left(\sum_i p_{\omega_i} C_{ij} \right) \right]$$

where C_{ij} is the value of the confusion matrix given truth target i and classification outcome j . A confusion matrix is one of many means of quantifying the likelihood of getting a particular measurement (or ATR outcome) given that the target belongs to a particular class. Other techniques for representing this likelihood, such as probability density functions, could be employed if available.

We estimate the future positions of all tracks by extrapolating them forward in time using the state estimated by the tracker. We extrapolate vehicles on roads down the road network. Vehicles that are offroad are extrapolated by using their velocity vector and performing dead-reckoning. If the tracker has multiple motion models (e.g., an interacting multiple model (IMM) tracker), the future state estimate can be made using a weighted average of the estimates of each motion model. Tracks that approach each other in the future, risk misassociation.

Given the expected trajectories of all of the tracks and the sensor schedules, the potential for misassociation can be computed. If tracks are misassociated, then the information contained in their classification vectors can become incorrect. The entropy metric assumes that the classification vectors are correct. To account for the errors introduced by misassociation, the classification vector should be altered. We account for this by computing the number of vehicles within a gate of each track. The gate is computed by taking into account the expected error in the velocity estimate of the tracker and the length of time since the track was last updated. Any tracks that lie within the gate of another track risk misassociation. We estimate the resulting entropy by assuming that the classification vectors of all tracks within the gate will be averaged. The entropy of this averaged classification vector is then used as the expected entropy of each track. If all of the targets within the gate are exactly the same type, then no information will be lost and the expected entropy will not change. If the targets within the gate are classified as significantly different vehicles, then the entropy level will increase.

By capturing the expected entropy considering misassociations in this manner, more sensor updates are encouraged for vehicles that are classified and traveling in denser traffic regions. This is because if the revisit rate is high on a particular vehicle, then the number of other vehicles in its association gate will be small. By minimizing the total number of vehicles in all gates, vehicles that are operating in isolation will receive fewer updates because there are no surrounding vehicles while vehicles in crowded situations will receive more updates. This automatically applies resources to where they are needed the most.

This objective function also captures the desire to minimize target location error because, if the location error is large, then the association gate will be large and more vehicles will be contained in the gate. This should be adequate in situations where the objective is to maintain tracks. If the objective is to provide targeting information, then a different objective function that places more importance on location accuracy is required. Our focus here is on track maintenance and we thus believe that additional accounting for errors in the kinematic state is not needed.

Tracks that are expected to be dropped will result in the loss of information contained in their classification vector. We account for this by giving these tracks an expected entropy level produced by complete ignorance of their classification. For example, a track that has missed several GMTI detections is labeled by our tracking model as a “stopped track”. This means that we suspect we are not detecting it anymore because it has stopped. If we illuminate its expected location with spotlight SAR, then we expect to be able to pick it up again. If we do not illuminate it with spotlight SAR, we expect that the track will be dropped and we will thus lose the information contained in the classification vector. If the classification vector had little information in it, little will be added to the objective function by not illuminating it. If the classification vector indicated the track was most certainly a target and it is not illuminated, then a large value will be added to the objective function because it is assumed that the track will be dropped.

2.4 Solution Approach

Once a suitable objective function and sensor schedule have been defined, attention can be turned toward a solution approach. The only approach that is sure to create optimum schedules is an exhaustive search of all of the different possible schedules for each asset. This is only computationally tractable for small problems with few assets and short time horizons. At the other end of the computational spectrum is a greedy heuristic. We have developed a heuristic that is a combination of a greedy approach and an exhaustive search.

Our solution to the dynamic tasking problem involves two phases. In the first phase, we task the sensors based on areas of interest (AOIs). This is done because prior to the collection of any detections, there will be no tracks to use for predicting the best schedules. Once the number of new tracks stabilizes using the AOI-based sensor plan, a track-based approach is employed. The next two subsections discuss more of the details of each of these two phases.

2.4.1 *AOI-based Tasking*

In this phase of the automated tasking process, sensor schedules are created based solely on a priori information. The a priori information that is used is a list of AOIs that have a specified sensor mode, priority, and revisit interval. The scheduling algorithm attempts to best meet the requirements of each AOI with preference given to AOIs with higher priority and shorter revisit times. A flowchart of the AOI-based tasking algorithm is shown in Figure 3. The process begins by first creating a set of grid points that cover the AOIs. Next, the possible viewing times for each sensor to view each AOI are computed and stored as viewing opportunities (VOPs). For example, an aircraft that enters a turn would end a viewing opportunity for an AOI if the AOI leaves the field-of-regard of the sensor while it is turning. The VOPs that overlap in time are grouped into sets called macroVOPs.

Once the macroVOPs have been created, they are cycled through in time sequence until the schedules of all sensors are filled. This is done by looping over time, visible AOIs, and tasks. The selection of a task is done by computing the total value of the area that is covered by a sensor beamprint on the ground. The value of the area changes depending on the priority placed on the AOI, the revisit time specified for the AOI, and the time since a task was last scheduled for the AOI. Once a task is selected, the value of the covered region is adjusted to account for the selection. This reduces the likelihood of placing another task in this region until after a revisit interval has passed.

2.4.2 *Track-based Tasking*

Once the number of tracks has stabilized in our track database, we switch our approach to a dynamic mode that plans schedules based on the track information. This approach is outlined in Figure 4. The algorithm begins with an outer loop over time. Within the time loop is a loop over assets. Within the asset loop is a loop over all feasible tasks for that

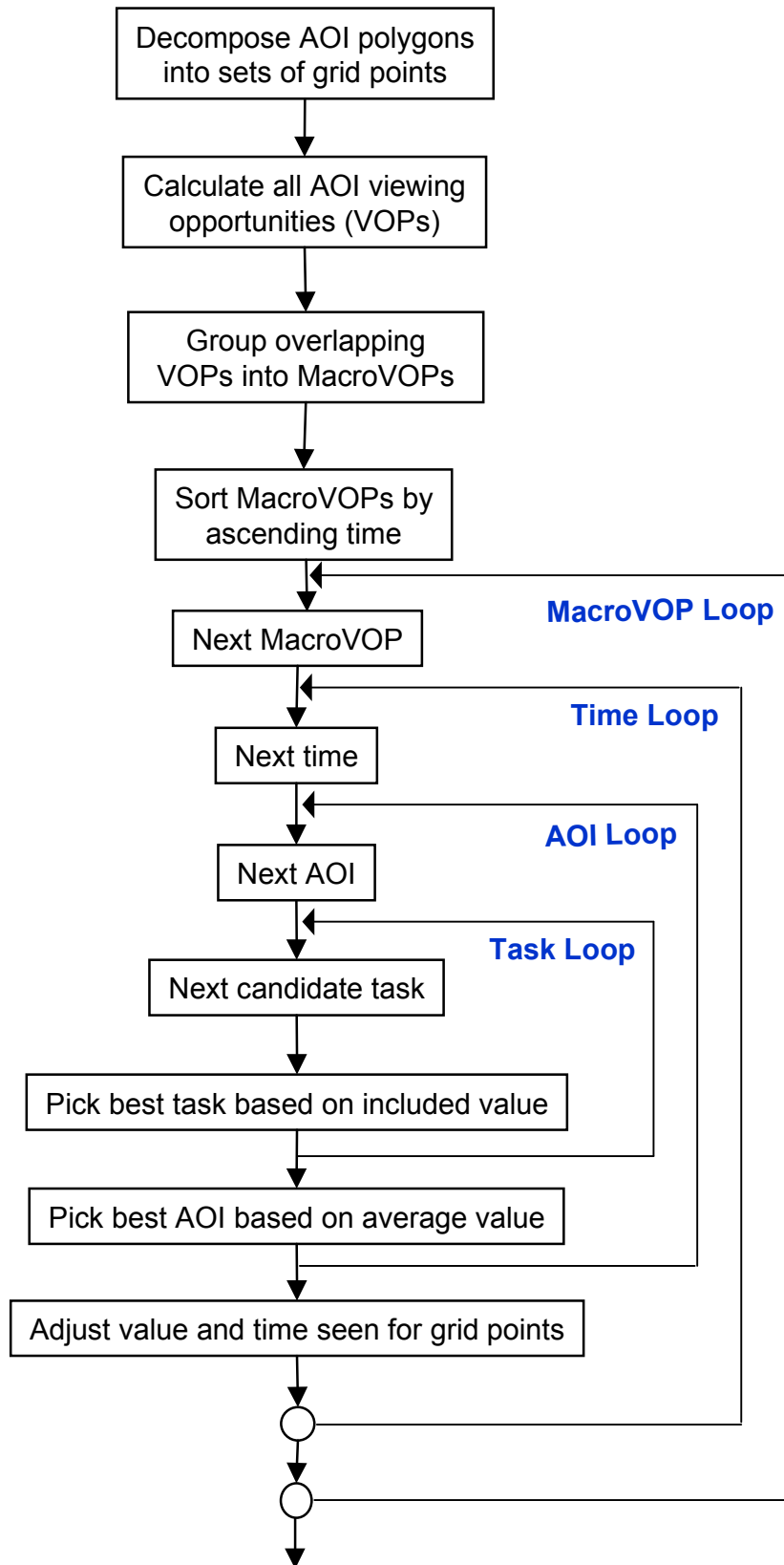


Figure 3. AOI-based Tasking

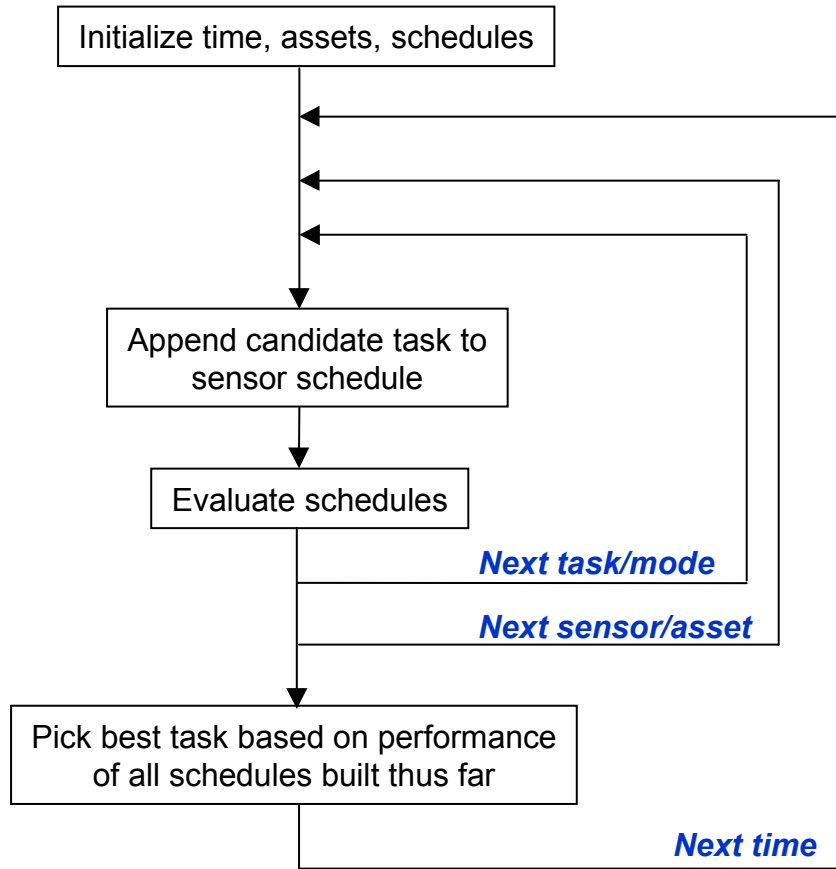


Figure 4. Track-based Dynamic Tasking

asset considering tasks in several possible modes (low- and high-range resolution GMTI and spot SAR). This portion of the algorithm is an exhaustive search. The best task for the asset at this time is selected and placed on the asset's schedule. This process is repeated for all of the other assets taking into consideration the tasks previously assigned to the first assets. The algorithm is greedy in the sense that once a task is chosen for an asset, other assets cannot influence its removal, but it is not greedy in the sense that the other assets can still inspect the same region if that improves the objective function. Once all of the assets have been assigned a task, the assets are looped over again, this time considering new tasks that will begin at the time that the previously-assigned tasks end. This process continues until all of the assets have tasks that extend past the time of the next processing event.

This algorithm does not find optimal schedules, but it runs fairly fast and is able to create good schedules. Its performance is quantified on an example problem in Section 4.

3 DESCRIPTION OF DYNAMIC TASKING TESTBEDS

We have developed a global asset manager prototype that works with aircraft operating GMTI and SAR sensors. In order to develop, test, and demonstrate this asset manager, we have built two testbeds. The testbeds were built using the existing SLAMEM™ and UAV Simulator (UAVsim) simulations. One testbed is a standalone version of SLAMEM™ that contains the dynamic tasking module. This testbed operates on a PC. The second testbed includes a version of SLAMEM™ operating on a PC and a version of UAVsim operating on a Sun. With this testbed, SLAMEM™ models the ground vehicles, fusion and tracking, and the dynamic sensor tasking process. UAVsim models the aircraft and the sensors. SLAMEM™ and UAVsim communicate via a Distributed Interactive Simulation (DIS) interface.

Under this effort, new aircraft and sensor models were added to SLAMEM™. Additionally, a DIS (Distributed Interactive Simulation) interface was added to UAVsim and SLAMEM™.

A block diagram showing the interaction of each module is shown in Figure 5. First consider the ground truth model on the left of the figure. The state of each vehicle is used to determine if it is detected. The type of each vehicle is used to determine the possible classification outcomes in the exploitation cell. The sensor and exploitation cells are modeled in UAVsim in one of the testbeds. This requires sending DIS PDUs to inform UAVsim of the state of each vehicle and detection PDUs (protocol description unit) to SLAMEM™ to inform the tracker model in SLAMEM™.

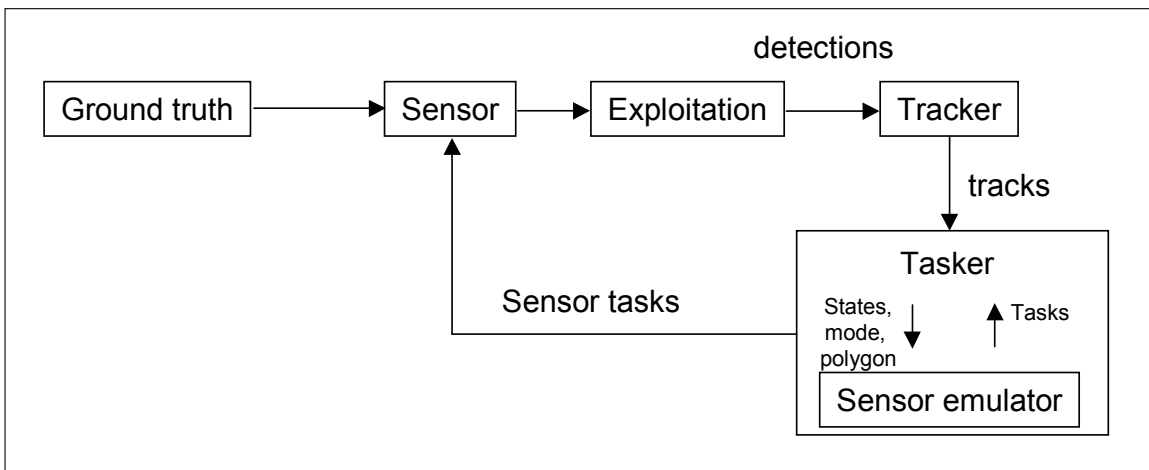


Figure 5. Module Interactions

The tracks are sent to the tasking module that resides within SLAMEM™ and there they are processed to create future sensor schedules as described in Section 2. The tasker interacts with a sensor emulator so that the future state of the aircraft and sensor can be computed for each candidate schedule that is considered. This required the creation of a UAVsim emulator within SLAMEM™ for the testbed that interacts with UAVsim. The sensor tasks are sent out to the sensor model. For the UAVsim – SLAMEM™ testbed this required sending an additional DIS PDU.

The results presented in this paper were created using the SLAMEM testbed. A description of the UAVsim can be found in [6].

3.2 SLAMEM™

3.2.1 Aircraft Model

There are two types of aircraft models in SLAMEM™. The motion of the first type is controlled by specifying a series of line segments that define the route of the aircraft along with an aircraft speed. The line segments are connected by computing a set of circular arcs that minimize the flight time between straight line segments. The radius of curvature of the circular arcs is computed using the specified speed in addition to an input acceleration.

The second aircraft model in SLAMEM™ mimics the flight model in UAVsim. This model was built by cutting and pasting the same code contained within UAVsim, but instead of simulating the motion of the aircraft while the simulation is running, the entire trajectory is run ahead of time and stored. This allows for the ability to predict where the aircraft will be in the future. This was needed in order to allow for the creation of future dynamic sensor schedules.

3.2.2 Radar Model

SLAMEM™ includes two radar models. One models GMTI and SAR, while the other mimics the GMTI model in UAVsim. The GMTI model assumes a fixed dwell time regardless of the azimuth and elevation angle being viewed. The slew logic for this model considers electronic and mechanical slew limits and slew rates specified in azimuth and elevation. The radar first attempts to perform a sensor task by steering the beam electronically. If the task is outside the electronic scan limits, mechanical steering is employed. The radar is slewed mechanically based on the change in azimuth and elevation angle between the previous location of the boresight and the new location.

The SAR model in SLAMEM™ accounts for the required integration time to achieve a desired resolution at a specified location. Additionally, the required slew time is computed if the target is outside the electronic field-of-regard.

The GMTI model in SLAMEM™ that mimics UAVsim operates in the same manner as described above for UAVsim. This model needed to be built so that dynamic sensor tasking could be performed with UAVsim.

3.2.3 Exploitation

Sensor data is sent from the surveillance assets to the exploitation center, where the data is transformed into information that can be employed for data fusion, target classification, and tracking. The first step in exploiting sensor data is to determine if a target has been detected. This is done for GMTI by checking to see that the target's radial velocity is above a minimum detectable velocity. If this test passes, the target is checked for foliage or terrain obscuration. If this test passes, the target must pass an input probability of detection. If it passes this final test, it is detected and passed on to the tracker.

Exploitation of classification information can be performed manually by an image analyst, or by a computer through automatic target recognition (ATR) algorithms. For GMTI/HRR (High Range Resolution) measurements we assume an automated classification procedure such as is described

in [7]. For SAR measurements, we assume an image analyst exploits the data with the aid of an automated tool such as SAIP (Semi-Automated IMINT Processing). The product of the exploitation center is information about potential targets that are detected in the sensors' fields of view, as well as any false alarms that may be generated. This information includes position and velocity estimates and target classification outcomes.

The performance of the ATR algorithms and/or image analysts can be represented using confusion matrices. A confusion matrix models ATR performance by assigning probabilities to the classification outcomes. These probabilities can be derived from statistics compiled when testing the system. The inputs to the confusion matrix are the true target types, and the outcomes are the perceived results of the ATR algorithm. When a vehicle is detected, the row corresponding to the true vehicle type is pulled out of the confusion matrix to simulate the classification decision. The row is used to segment the real number line between zero and one. A random number is drawn between zero and one and the segment of the number line that it falls into determines the classification outcome. The column from the confusion matrix that is specified by the classification decision is used by the tracker as described in Section 3.2.4.2.

3.2.4 Fusion and Tracking

Our dynamic tasking module must work closely with a tracker that feeds it the estimated future location and identity of all targets in a region. The actual tracker that is employed can be of any particular type so long as it supports our general interface. Since the focus of this effort was on the development of a dynamic sensor-tasking module, we used a model of a tracker to evaluate and develop it.

Tracking Model

We have modeled the tracking function using something that we call the parametric tracker. This model does not operate as an actual tracker, but rather models these operations using ground truth and random number draws. The methodology used for this process is described in [8]. A key parameter for this tracking model is the tracking quality factor. This quality factor is an approximation to the probability of correct association. Experiments have been performed (e.g., [9]) in order to estimate this parameter using actual trackers.

An approach to modeling the performance of a tracker that uses SAR and GMTI detections to continuously track a vehicle over multiple move-stop cycles was developed here. This model does not assume perfect association of the detections, but rather allows for misassociations in ambiguous situations. The extent to which misassociations occur is controlled by a parameter that can be varied between zero and one. When this parameter is set to one, the tracker behaves with perfect association. When the parameter is set to zero, the tracker will still correctly associate in unambiguous situations, but will incorrectly associate in ambiguous situations.

We assume the tracker operates at a fusion and tracking center where batches of SAR and GMTI detections are sent. Each detection has a collection time associated with it. The collection time records the time at which the detected object was illuminated by the SAR or GMTI radar.

The tracker maintains a list of tracks. Each track is composed of a set of previous detections that have been associated and, possibly, an estimate of the current state of the object. At regular

update intervals, the tracker considers a new batch of detections for association with the current set of tracks.

Track Drops

Before a track is dropped, it must meet two sets of criteria. First, it must not have been updated with a detection even though it was illuminated by a sensor for a specified number of illuminations. For instance, we typically choose one SAR illumination for stationary tracks and several MTI illuminations for tracks that are expected to be moving. Secondly, after the illumination criterion has been met, a period of time must elapse before the track will be dropped. This has been employed to keep tracks from being dropped in situations where we are waiting for a vehicle to exit an obscured region or we are waiting for a spotlight SAR image to be formed and exploited so that the moving to stationary transition can be observed. The track drop times and number of missed illuminations are different for each target type.

ID Fusion

The goal of the evidence accumulation process is to correctly classify targets with high confidence. Since any one sensor may not provide classification with the required degree of confidence, we require fusion of data from multiple sensors or from the same sensor using multiple measurements. Additionally, we incorporate a priori information about the targets into the confidence estimate.

Consider a simple case where objects detected by the sensors are divided into two categories, targets of interest (TOIs) and non-TOIs (a 2×2 confusion matrix). A “good” sensor would have a high probability of correctly classifying a TOI, and a low probability of misclassifying it as a non-TOI. Occasionally, this sensor may also incorrectly identify non-TOIs as TOIs. Since the non-TOIs, or “confuser” objects, often outnumber the actual targets of interest, even minor probabilities of misclassification can lead to a large number of misclassified targets. Updating the classification of detected objects with subsequent sensor detections can increase the confidence in the target classification.

Target likelihoods (i.e., the likelihood that a target is of a particular type) for objects in the target database can be updated with each subsequent sensor measurement through Bayesian evidence accumulation, as described in [10]. This method involves updating the target class likelihoods through repeated applications of Bayes rule,

$$P(\omega_j | Z^k) = \frac{P(z_k | \omega_j)P(\omega_j | Z^{k-1})}{\sum_{i=1}^{n_i} P(z_k | \omega_i)P(\omega_i | Z^{k-1})}$$

where $P(\omega_j | Z^k)$ is the probability that the target is of type j given all measurements up to and including the measurement collected at time k , $P(z_k | \omega_j)$ is the probability of getting the measurement z_k given that the target belongs to class ω_j , and n_i is the number of possible classification outcomes (i.e., number of classes). For the first measurement, initial target likelihoods are set *a priori* to a value that is typically based on the expected density of that target type, but may also be based on other factors such as the location of the first detection.

4 EXAMPLE ANALYSIS PROBLEM

In this section we apply the dynamic tasking module to an example problem and quantify its performance. We compare its performance to the performance that would be obtained if the same sensor is applied, but with a non-dynamic sensor tasking strategy. To develop this non-dynamic sensor tasking strategy, we use the area-based sensor-tasking algorithm for the entire scenario instead of allowing the track-based approach to take over at some point.

4.1 Location

We have defined an area of interest (AOI) that is located in North Korea. This AOI is used to create the area-based sensor tasking. Terrain in the eastern portion of this region is rugged, making terrain-masking likely.

4.2 Ground Vehicles

We have placed ten target vehicles in this region. The target vehicles travel as a unit. The target vehicles have been selected to be tanks, but other applications such as a mobile surface-to-air missile unit should provide similar results.

The target vehicles begin the scenario in a stationary location and remain stationary for the first five minutes. After five minutes, the targets move from their initial site to another site in the area that is selected at random from a set of uniformly-spaced sites. The typical travel time is approximately ten to fifteen minutes. When the targets reach their new site, they stop and remain stationary for about thirty minutes. After this time, they select a new site and travel to it. Once reaching the second site, they stop and remain stationary for the remainder of the simulation (two hours total simulation time).

In addition to the target vehicles, there are 200 background vehicles operating in the area. These background vehicles are moving nearly all of the time.

4.3 Blue Assets

The Blue architecture designed to locate the targets consists of one advanced Global Hawk, an exploitation cell, and a time-critical targeting cell. The Global Hawk carries a radar that is capable of searching with GMTI/HRR at a rate of one degree per second. It can also create spotlight SAR imagery, but not at the same time it's performing GMTI. The sensor data from the radar is processed in the exploitation cell to produce detections. GMTI detections are produced with a minimal delay, but each SAR detection is assumed to require an image analyst about 90 seconds to process. The actual time required to process an image is based on a fixed search time plus an additional amount of time based on the number of vehicles in the image. The time-critical targeting cell contains the fusion and tracking algorithms as well as the dynamic sensor tasking algorithms described in Sections 2 and 3. The tracking quality parameter used was 0.95. This value has been shown to be reasonable for an existing tracker in a similar scenario [9].

The performance of the exploitation cell when classifying vehicles is modeled using the confusion matrices in Tables 1 and 2 for GMTI/HRR and spot SAR, respectively. The assumed

performance estimates are conservative and are only intended for the use of evaluating the tasking algorithm.

Table 1. GMTI/HRR Confusion Matrix

!! True Category	Pdetect	TEL	MTT	CRANE	POL	SAM_LAU	RADAR	TANK	TRUCK	BUS	CAR
TEL	0.9	0.65	0.05	0.05	0.02	0.05	0.03	0.05	0.05	0.05	0
MTT	0.9	0.05	0.5	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.1
CRANE	0.9	0.05	0.05	0.5	0.05	0.05	0.05	0.05	0.05	0.05	0.1
POL	0.9	0.05	0.05	0.05	0.4	0.05	0.05	0.05	0.05	0.05	0.2
SAM_LAUNCHER	0.9	0.05	0.05	0.05	0.05	0.5	0.05	0.05	0.05	0.05	0.1
RADAR	0.9	0.05	0.05	0.05	0.05	0.05	0.5	0.05	0.05	0.05	0.1
TANK	0.9	0.05	0.05	0.05	0.05	0.05	0.05	0.5	0.05	0.05	0.1
TRUCK	0.9	0.1	0.1	0.1	0.1	0.1	0.05	0.05	0.25	0.05	0.1
BUS	0.9	0.1	0.1	0.1	0.1	0.1	0.05	0.05	0.05	0.25	0.1
CAR	0.7	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.55

Table 2. Spot SAR Confusion Matrix

!! True Category	Pdetect	TEL	MTT	CRANE	POL	SAM_LAU	RADAR	TANK	TRUCK	BUS	CAR
TEL	1	0.85	0.05	0.05	0	0.05	0	0	0	0	0
MTT	1	0.05	0.7	0.05	0.05	0.05	0	0.05	0.05	0	0
CRANE	1	0.05	0.05	0.7	0.05	0.05	0.05	0.05	0	0	0
POL	1	0.05	0.05	0.05	0.6	0.05	0.05	0.05	0.05	0.05	0
SAM_LAUNCHER	1	0.05	0.05	0.05	0.05	0.7	0.05	0.05	0	0	0
RADAR	1	0.05	0.05	0.05	0.05	0.05	0.7	0.05	0	0	0
TANK	1	0.05	0.05	0.05	0.05	0.05	0.05	0.7	0	0	0
TRUCK	1	0	0.05	0.1	0.1	0.1	0.05	0.05	0.45	0.05	0.05
BUS	1	0.05	0.05	0.1	0.1	0.1	0.05	0.05	0.05	0.45	0
CAR	1	0	0	0	0.05	0.05	0.05	0	0.05	0.05	0.75

Table 3. Track Drop Criteria

!!		Drop stationary tracks		Drop moving tracks		Declare as stopped	
!! Target name	P(i) declaration	Time (sec)	Illuminations	Time (sec)	Illuminations	Time (sec)	Illuminations
TEL	0.5	3600	2	600	4	250	3
MTT	0.5	3600	2	600	4	250	3
CRANE	0.5	3600	2	600	4	250	3
POL	0.5	600	1	600	4	250	3
SAM_LAUNCHER	0.5	3600	2	600	4	250	3
RADAR	0.5	3600	2	600	4	250	3
TANK	0.5	3600	2	600	4	250	3
TRUCK	0.8	600	1	300	3	200	3
BUS	0.8	600	1	300	3	200	3
CAR	0.8	600	1	300	3	200	3
UNCLASSIFIED	N/A	600	1	300	4	200	4

The track drop criteria and the stopped track declaration criteria used by the tracking module are provided in Table 3. These values were selected based on an estimate of what might work best. An exploration of the changes in performance when using other values might produce significantly better results.

4.4 Analysis Results

One measure of the performance of the Blue architecture is the length of time that the target locations are known. We have computed this measure for (1) the case in which the dynamic sensor tasking algorithms are employed and (2) the case in which only the area-based sensor tasking algorithm is employed. The area-based sensor tasking is representative of what could be preplanned without knowledge of the target locations. These results are shown in the bar chart in Figure 7. Note that the dynamic sensor tasking track length is significantly longer than when the preplanned sensor-tasking algorithm is employed. Much of this benefit is due to the fact that the targets are lost when they transition to a stationary state. They are lost by the preplanned sensor tasking strategy because only GMTI/HRR is employed. We could have created a preplanned

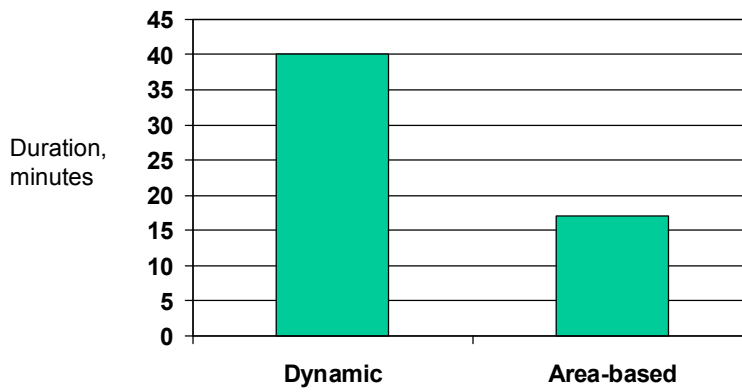


Figure 6. Comparison of Average Track Length

sensor search strategy employing a stripmap search of the area, but this would have provided a revisit rate of about 2½ hours. With this revisit rate, the targets could stop and restart before ever being imaged and there would be no HRRGMTI coverage to catch them when they are moving. The power of the dynamic tasking algorithm lies in its ability to make use of the fast revisit rate of the GMTI sensor in addition to a few selected applications of the SAR sensor to catch the targets when they are stationary.

Figure 8 shows the number of real and false tracks as a function of time during the scenario when

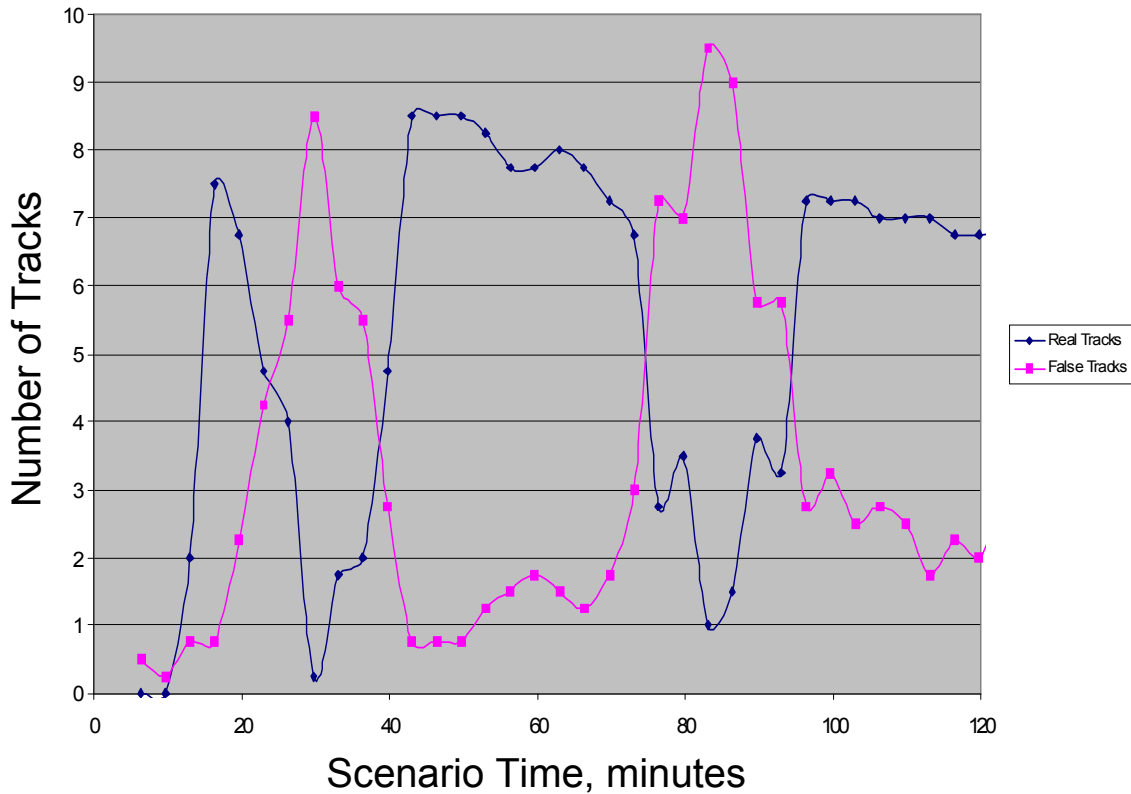


Figure 7. Number of Real and False Tracks versus Time – *Dynamic Tasking*

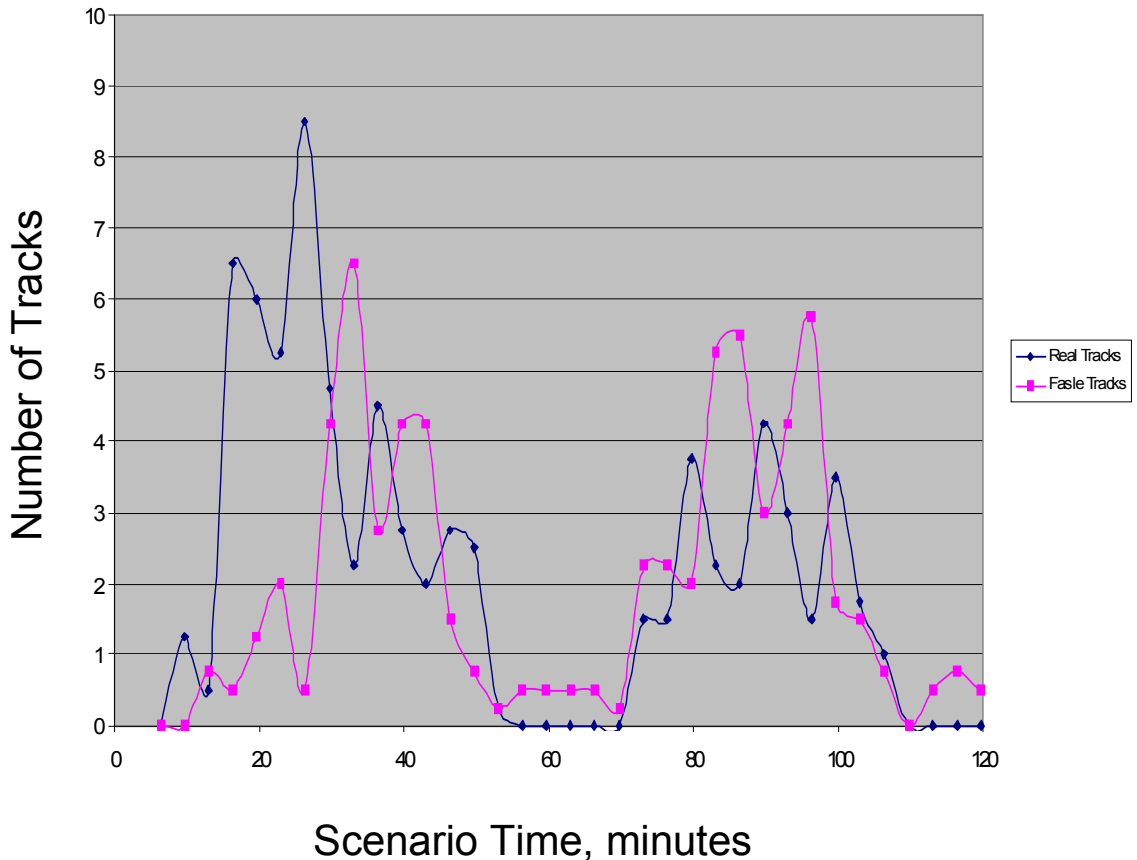


Figure 8. Real and False Tracks versus Time – *Preplanned Tasking*

using the dynamic sensor-tasking algorithm. A real track is defined as a track that is labeled as a target of interest and there is in fact a target of interest within a specified distance of the track’s estimated position. The distance specified for this case was 800 m. A false track is defined as a track that is labeled as a target of interest without a true target within the specified distance of the track position. Note that when the number of real tracks drops, the number of false tracks often increases. This happens when a track has not been updated for a while and the position information becomes old. In this scenario, it is a temporary situation that happens when the vehicles are transitioning from a moving state to a stopped state. Once the spot SAR images are made, the vehicle positions are updated and the number of real tracks rises and the number of false tracks decreases.

Figure 9 shows the number of real and false tracks when the preplanned sensor tasking strategy is employed. Note that the number of real tracks drops significantly at around 60 and 120 minutes. This is because the targets are stopping shortly before these time periods and the preplanned GMTI/HRR tasking is unable to detect the targets when they are stationary. If the vehicles remain stationary for a long period of time, their position will be unknown.

5 SUMMARY

We have created a dynamic sensor-tasking algorithm that can follow targets through multiple move-stop cycles. This algorithm employs the concept of entropy to maximize the amount of information collected and maintained by a multi-target fusion and tracking process. The algorithm employs GMTI/LRR (low Range Resolution), GMTI/HRR, or spot SAR at the time and location that minimizes the total entropy expected for all of the tracks.

In order to develop, test, and demonstrate this algorithm, we have created two testbeds. The first testbed employs both the UAVsim and SLAMEM™ simulations as a networked system. This testbed is primarily intended for demonstration purposes and can operate only with the GMTI/LRR mode. The second testbed employs SLAMEM™ as a standalone application on a PC with the UAVsim flight and sensor dwell time models incorporated. This testbed supports all three modes of dynamic tasking.

We have evaluated the performance of the dynamic-tasking module in a scenario involving ten targets and 200 background vehicles. The module demonstrated good performance in comparison to a pre-planned sensor tasking strategy.

6 REFERENCES

1. Musick, S., Kastella, K., "Comparison of Sensor Management Strategies for Detection and Classification", Ninth National Symposium on Sensor Fusion, March 1996.
2. Sullivan, K., et al., "Surveillance and Targeting Studies," August 2000, Toyon Research Corporation.
3. Washburn, R., Chao, A., Castanon, D., Bertsekas, D., Malhotra, R., "Stochastic Dynamic Programming for Far-Sighted Sensor Management," IRIS National Symposium on Sensor and Data Fusion, 1997.
4. Brandstadt, J., Kozak, M., Redmond, P., Jones, J., "A UAV Flight and Sensor Simulator for Generating Radar and Infrared Detections as Stimuli for Track and Fusion Processors," Proceedings of the IRIS Sensor and Data Fusion Symposium, 1999.
5. Ressler, M., et al., "Robust Automatic Target Recognition (ATR) Using HRR Profiles," August 2000, Toyon Research Corporation.
6. Sullivan, K., "A Parametric Model of Tracker Performance", Toyon Research Corporation, February 1999.
7. Meloon, M., Sullivan, K., "Evaluation of the AlphaTech Tracker for Discoverer II," October 2000, Toyon Research Corporation (T301C0401).
8. Rich, E., Knight, K., *Artificial Intelligence*, 2nd ed., McGraw-Hill, Inc., 1991.