

ABSTRACT

The Center for Information Technology (CIT) of the University of South Carolina (USC) participated in the Autonomous Negotiating Teams (ANTS) program funded by DARPA since its inception. The USC team was involved in two efforts. The first was to develop a resource allocation architecture called **TargetShare**, in which agents used utility to negotiate and allocate resources. That effort was led by Dr. Jose Vidal. The second effort, led by Dr. Juan E. Vargas, was focused on providing a multi-target tracking system, called the **SCTracker**, compatible with the program's challenge problem infrastructure. This report describes the two efforts.

TABLE OF CONTENTS

| | |
|---|----|
| A. TARGETSHARE | 1 |
| 1. Introduction..... | 1 |
| 1.1 Task Allocation | 2 |
| 1.2 Service Allocation..... | 2 |
| 2. A Formal Model for Service Allocation..... | 3 |
| 2.1 Search Algorithms | 3 |
| 3. Simulations..... | 7 |
| 4. Test Results..... | 8 |
| 5. Related Work | 9 |
| 6. Conclusions..... | 9 |
| B. SC TRACKER | 11 |
| 1. Introduction..... | 11 |
| 2. Sensor Model..... | 11 |
| 3. Process Model..... | 12 |
| 4. Time Frames | 14 |
| 5. Location Model | 15 |
| 6. Amplitude Handler | 16 |
| 7. Frequency Handler..... | 17 |
| 8. Motion Model..... | 19 |
| 9. Target Location | 21 |
| 10. Results..... | 23 |
| 11. Multiple Target Tracking..... | 23 |
| 12. Conclusions..... | 24 |
| References | 26 |

TABLE OF FIGURES

| | |
|--|-----------|
| <i>Figure 1: Interactions Between the State and Process Models.....</i> | <i>14</i> |
| <i>Figure 2: Time Frames.....</i> | <i>15</i> |
| <i>Figure 3: Target Probability Distribution From Amplitude Measuremens.</i> | <i>18</i> |
| <i>Figure 4: Relation Between the Sensor Location and the Target Velocity</i> | <i>19</i> |
| <i>Figure 5: Motion Model</i> | <i>22</i> |
| <i>Figure 6: Results Obtained with the SC Tracker During the First Demonstration</i> | <i>23</i> |
| <i>Figure 7: Results Obtained with the SC Tracker During the Final Demonstration.....</i> | <i>24</i> |

A. TARGETSHARE

1. INTRODUCTION

TargetShare is an architecture based on a method for solving service allocation problems in which a set of services must be allocated to a set of agents so as to maximize a global utility. The method is completely distributed so it can scale to any number of services without degradation. In this report, we formalize the service allocation problem and then present a simple hill-climbing, a global hill climbing, and a bidding-protocol algorithm for solving it. We then analyze the expected performance of these algorithms as a function of various problem parameters such as the branching factor and the number of agents. Finally, we use the sensor allocation problem, an instance of a service allocation problem, to show the bidding protocol at work. The simulations also show that phase transition on the expected quality of the solution exists as the amount of communication between agents increases.

The problem of dynamically allocating services to a changing set of consumers arises in many applications. For example, in an e-commerce system, the service providers are always trying to determine which service to provide to whom, and at what price [5]; in an automated manufacturing for mass customization scenario, agents must decide which services will be more popular/profitable [1]; and in a dynamic sensor allocation problem, a set of sensors in a field must decide which area to cover, if any, while preserving their resources.

While these problems might not seem related, they are instances of a more general service allocation problem in which a finite set of resources can be allocated by a set of autonomous agents so as to maximize some global measure of utility. A general approach to solving these types of problems has been used in many successful systems, such as [2] [3] [11] [9]. The approach involves three general steps: 1. Assign each resource that needs to be preserved to an agent responsible for managing the resource. 2. Assign each goal of the problem domain to an agent responsible for achieving it. Achieving these goals requires the consumption of resources. 3. Have each agent take actions so as to maximize its own utility, but implement a coordination algorithm that encourages agents to take actions that also maximize the global utility. In this report we formalize this general approach by casting the problem as a search in a global fitness landscape which is defined as the sum of the agents' utilities. We show how the choice of a coordination/communication protocol disseminates information, which in turn "smooths" the global utility landscape. This smooth global utility landscape allows the agents to easily find the global optimum by making selfish decisions to maximize their individual utility. We also present experiments that pinpoint the location of a phase transition in the time it takes for the agents to find the optimal allocation. The transition can be seen when the amount of communication allowed among agents is manipulated. It exists because communication allows the agents to align their individual landscapes with the global landscape. At some amount of communication, the alignment between these landscapes is good enough to allow the agents to find the global optimum, but less communication drives the agents into a random behavior from which the system cannot recuperate.

1.1 Task Allocation

The service allocation problem we discuss is a superset of the well known task allocation problem. A task allocation problem is defined by a set of tasks that must be allocated among a set of agents. Each agent has a cost associated with each subset of tasks, which represents the cost the agent would incur if it had to perform those tasks. Coordination protocols are designed to allow agents to trade tasks so that the globally optimal allocation (the one that minimizes the sum of all the individual agent costs) is reached as soon as possible. It has been shown that this globally optimal allocation can be reached if the agents use the contract-net protocol [9] with Original Cluster Swap Multi (OCSM) contracts [8]. These OCSM contracts make it possible for the system to transition from any allocation to any other allocation in one step. As such, a simple hill-climbing search is guaranteed to eventually reach the global optimum. We consider the service allocation problem, which is a superset of the task allocation because it allows for more than one agent to service a “task”. The service allocation problem we study also has the characteristic that every allocation cannot be reached from every other allocation in one step.

1.2 Service Allocation

In a service allocation problem there are a set of services, offered by service agents, and a set of consumers who use those services. A server can provide any one of a number of services and some consumers will benefit from that service without depleting it. A server agent incurs a cost when providing a service and can choose not to provide any service. For example, a server could be an agent that sets up a website with information about cats. All the consumer agents with interests in cats will benefit from this service, but those with other interests will not benefit. Since each server can provide, at most, one service, the problem is to find the allocation of services that maximizes the sum of all the agents’ utilities, that is, an allocation that maximizes the global utility.

1.2.1 SENSOR ALLOCATION

Another instance of the service allocation problem is the sensor allocation problem, which we will use as an example throughout this report. In the sensor allocation problem we have a number of sensors placed in fixed positions in a two-dimensional space. Each sensor has a limited viewing angle and distance but can point in any one of a number of directions. For example, a sensor might have a viewing angle of 120 degrees, viewing distance of 3 feet, and be able to look in three directions, each one 120 degrees apart from the others. That is, it can “look” in any one of three directions. In each direction it can see everything that is in the 120 degree and 3 feet long view cone. Each time a sensor looks in a particular direction it uses energy. There are also targets that move around in the field. The goal is for the sensors to detect and track all the targets in the field. However, in order to determine the location of a target, two or more sensors have to look at it at the same time. We also wish to minimize the amount of energy spent by the sensors.

We consider the sensor agents as being able to provide three services, one for each sector,

but only one at a time. We consider the target agents as consuming the services of the sensors.

2. A FORMAL MODEL FOR SERVICE ALLOCATION

We define a service allocation problem SA as a pair $SA = \{C, S\}$ where C is the set of consumer agents $C = \{c_1, c_2, \dots, c_{|C|}\}$, and c_i has only one possible state, $c_i = 0$. The set of service agents is $S = \{s_1, s_2, \dots, s_{|S|}\}$ and the value of s_i is the value of that service. For the sensor domain in which a sensor can observe any one of three 120-degree sectors or be turned off we have $s_i \in \{0, 1, 2, \text{off}\}$. An allocation is an assignment of states to the services (since the consumers have only one possible state we can ignore them). A particular allocation is denoted by $a = \{s_1, s_2, \dots, s_{|S|}\}$, where the s_i have some value taken from the domain of service states, and $a \in \mathcal{A}$, where \mathcal{A} is the set of all possible allocations. That is, an allocation tells us the state of all agents (since consumers have only one state they can be omitted). Each agent also has a utility function. The utility that an agent receives depends on the current allocation a , where we let $a(s)$ be the state of service agent s under a . The agent's utilities will depend on their state and the state of other agents. For example, in the sensor problem we define the utility of sensor s as $U_s(a)$, where

$$U_s(a) = \begin{cases} 0 & \text{if } a(s) = \text{off} \\ -k_1 & \text{otherwise} \end{cases} \quad [\text{eq 1}]$$

That is, a sensor receives no utility when it is off and must pay a penalty of $-k_1$ when it is running. The targets are the consumers, and each target's utility is defined as U_c

$$U_c(a) = \begin{cases} 0 & \text{if } fc(a) = 0 \\ k_2 & \text{if } fc(a) = 1 \\ k_2 + n - 2 & \text{if } fc(a) = n \end{cases} \quad [\text{eq 2}]$$

where $fc(a) =$ number of sensors s that see c given their state $a(c)$. [eq 3]

Finally, given the individual agent utilities, we define the global utility $GU(a)$ as the sum of the individual agents' utilities:

$$GU(a) = \sum_{c \in C} U_c(a) + \sum_{s \in S} U_s(a) \quad [\text{eq 4}]$$

The service allocation problem is to find the allocation a that maximizes $GU(a)$. In the sensor problem, there are $4^{|S|}$ possible allocations, which would make a simple generate-and-test approach take exponential amounts of time. We wish to find the global optimum much faster than that.

2.1 Search Algorithms

Our goal is to design an interaction protocol whereby an allocation a that maximizes the global utility $GU(a)$ is reached in a small number of steps. In each step of our protocol

one of the agents will change its state or send a message to another agent. The messages might contain the state or utilities of other agents. We assume that the agents do not have direct access to the other agents' states or utility values. A simpler algorithm could involve having each consumer, at each time, changing the state of a randomly chosen service agent so as to increase the consumer's own utility. That is, a consumer c will change the current allocation a into a_0 by changing the state of some sensor s such that $U_c(a_0) > U_c(a)$. If the sensor's state cannot be changed so as to increase the utility, then the consumer does nothing. In the sensor domain this amounts to a target picking a sensor and changing its state so that the sensor can see the target. We refer to this algorithm as individual hill-climbing. The individual hill-climbing algorithm is simple to implement and the only communication needed is between the consumer and the chosen server. This simple algorithm makes every consumer agent increase its individual utility at each turn. However, the new allocation a_0 might result in a lower global utility, since a_0 might reduce the utility of several other agents. Therefore, it does not guarantee that an optimal allocation will be eventually reached. Another approach is for each agent to change state so as to increase the global utility. We call this a global hill-climbing algorithm.

In order to implement this algorithm, an agent would need to know how the proposed state change affects the global utility as well as the states of all the other agents. That is, it would need to be able to determine $GU(a')$ which requires it to know the state of all the agents in a_0 as well as the utility functions of every other agent, as per the definition of global utility in equation 4. In order for an agent to know the state of others, it would need to somehow communicate with all other agents. If the system implements a global broadcasting method then we would need for each agent to broadcast its state at each time. If the system uses more specialized communications such as point-to-point, limited broadcasting, etc., then more messages will be needed. Any protocol that implements the global hill-climbing algorithm will reach a locally optimal allocation in the global utility. This is because it is always true that, for a new allocation a_0 and old allocation a , $GU(a') \geq GU(a)$. Whether or not this local optimum is also a global optimum will depend on the ruggedness of the global utility landscape. That is, if it consists of one smooth peak then it is likely that any local optimum is the global optimum. On the other hand, if the landscape is very rugged then there are likely many local peaks.

Studies in NK landscapes [4] tell us that smoother landscapes result when an agent's utility depends on the state of smaller number of other agents. Global hill-climbing is better than individual hill-climbing since it guarantees that we will find a local optima. However, it requires agents to know each others' utility function and to constantly communicate their state. Such large amount of communication is often undesirable in multi-agent systems. We need a better way to find the global optimum.

One way of correlating the individual landscapes to the global utility landscape is with the use of a bidding protocol in which each consumer agent tells each service the marginal utility the consumer would receive if the service switched its state to so as to maximize the consumer's utility. The service agent can then choose to provide the service with the highest aggregate demand. Since the service is picking the value that maximizes the utility of everyone involved (all the consumers and the service) without decreasing the utility of

anyone else (the other services) this protocol is guaranteed to never decrease the global utility. This bidding protocol is a simplified version of the contract-net [9] protocol in that it does not require contractors to send requests for bids. However, in order for a consumer to determine the marginal utility it will receive from one sensor changing state, it still needs to know the state of all the other sensors. This means that a complete implementation of this protocol will still require a lot of communication (namely, the same amount as in global hill-climbing). We can reduce this number of messages by allowing agents to communicate with only a subset of the other agents and making their decisions based on only this subset of information. That is, instead of all services telling each consumer their state, a consumer could receive state information from only a subset of the services and make its decision based on this (assuming that the services chosen are representative of the whole). This strategy shows a lot of promise but its performance can only be evaluated on an instance-by-instance basis. We explore this strategy experimentally in Section 3 using the sensor domain.

2.1.1 THEORETICAL TIME BOUNDS OF GLOBAL HILL-CLIMBING

We now know that global hill-climbing will always reach a local optimum, the next questions we must answer are:

1. How many local optima are there?
2. What is the probability that a local optimum is the global optimum?
3. How long does it take, on average, to reach a local optimum?

Let a be the current allocation and a_0 be a neighboring allocation. We know that a is a local optimum if

$$\forall_{a' \in N(a)} GU(a) \geq GU(a') \quad [\text{eq 5}]$$

where

$$N(a) = \{x \mid x \text{ is a Neighbor of } a\} \quad [\text{eq 6}]$$

We define a Neighbor allocation as an allocation where one, and only one, agent has a different state.

The probability that some allocation I is a local optimum is simply the probability that equation 5 is true. If the utility of all pairs of neighbors is not correlated, then this probability is

$$\Pr[\forall_{a' \in N(a)} GU(a) > GU(a')] = \Pr[GU > GU(a')]^b \quad [\text{eq 7}]$$

where b is the branching factor. In the sensor problem $b = 3 \cdot |S|$ where S is the set of all sensors. That is, since each sensor can be in any of four states it will have three neighbors from each state. In some systems it is safe to assume that the global utilities of a 's neighbors are independent. However, most systems show some degree of correlation.

Now we need to calculate the $Pr[GU(a) > GU(a_0)]$, that is, the probability that some allocation a has a greater global utility than its neighbor a_0 , for all a and a_0 . This could be calculated via an exhaustive enumeration of all possible allocations. However, often we can find the expected value of this probability. For example, in the sensor problem each sensor has four possible states. If a sensor changes its state from sector x to sector y the utility of the target agents covered by x will decrease while the utility of those in y will increase. If we assume that, on average, the targets are evenly spaced on the field, then the global utilities for both of these are expected to be the same. That is, the expected probability that the global utility of one allocation is bigger than the other is $1/2$. If, on the other hand, a sensor changes state from “off” to a sector, or from a sector to “off,” the global utility is expected to decrease and increase, respectively. However, there are an equal number of opportunities to go from “off” to “on” and vice-versa. Therefore, we can also expect that for these cases the probability that the global utility of one allocation is bigger than the other is $1/2$. Based on these approximations, we can declare that for the sensor problem

$$Pr[\forall_{a' \in N(a)} GU(a) > GU(a')] = \frac{1}{2^b} = \lambda \quad [\text{eq 8}]$$

If we assume an even distribution of local optima, the total number of local optima is simply the product of the total number of allocations times the probability that each one is a local optimum. That is,

$$\text{Total number of local optima} = \lambda |A| \quad [\text{eq 9}]$$

For the sensor problem, $\lambda = 1/2^b$, $b = 3 \cdot |S|$ and $|A| = b^{|S|}$, so the expected number of local optima is $b^{|S|} / 2^{3|S|}$.

$$Pr[a \text{ Local Optimum is Also Global}] = \frac{1}{\lambda |A|} = \frac{1}{2^b} \quad [\text{eq 10}]$$

We can find the expected time the algorithm will take to reach a local optimum by determining the maximum number of steps from every allocation to the nearest local optimum. This gives us an upper bound on the number of steps needed to reach the nearest local optimum using global hill-climbing. Notice that, under either individual hill-climbing or the bidding protocol it is possible that the local optimum is not reached, or is reached after more steps, since these algorithms can take steps that lower the global utility. In order to find the expected number of steps to reach a local optimum, we start at any one of the local optima and then traverse all possible links at each depth d until all possible allocations have been visited. This occurs when

$$\lambda |A| \cdot bd > |A| \quad [\text{eq 11}]$$

Solving for d , and remembering that $\lambda = 1/2^b$, we get

$$d > b \log_b 2. \tag{eq 12}$$

The expected worst-case distance from any point to the nearest local optimum is, therefore, $b \log_b 2$ (this number only makes sense for $b \geq 2$ since smaller number of neighbors do not form a searchable space). That is, the number of steps to reach the nearest local optima in the sensor domain is proportional to the branching factor b , which is equal to $\beta \cdot |S|$. We can expect search time to increase linearly with the number of sensors in the field.

3. SIMULATIONS

While the theoretical results above give us some bounds on the number of iterations before the system is expected to converge to a local optimum, the bounds are rather loose and do not tell us much about the dynamics of the executing system. Also, we cannot show mathematically how changes in the amount of communication change the search. Therefore, we have implemented a service allocation simulator to answer these questions. It simulates the sensor allocation domain described in the introduction. The simulator is written in Java and the source code is available upon request. It gathers and analyzes data from any desired number of runs. The program can analyze the behavior of any number of target and sensor agents on a two-dimensional space, and the agents can be given any desired utility function. The program is limited to static targets. That is, it only considers the one-shot service allocation problem.

Each new allocation is completely independent of any previous one. In the tests we performed, each run has seven sensors and seven targets, all of which are randomly placed on a two-dimensional grid. Each sensor can only point in one of three directions or sectors. These three sectors are the same for all sensors (specifically, the first sector is from 0 to 120 degrees, the second one from 120 to 240, and the third one from 240 to 360). All the sensors use the same utility function which is given by equation 1, while the targets use equation 2. After a sensor agent receives all the bids it chooses the sector that has the highest aggregate demand, as described by the bidding protocol. During a run, each of the targets periodically sends a bid to a number of sensors asking them to turn to the sector that faces the target. We set the bid amount to a fixed number for these tests. Periodically, the sensors count the number of bids they have received for each sector and turn their detector (such as a radar) to face the sector with the highest aggregate demand.

We assume that neither the targets nor the sensors can form coalitions. We vary the number of sensors to which the targets send their bids in order to explore the quality of the solution that the system converges upon as the amount of communication changes. For example, at one extreme if all the targets send their bids to all the sensors, then the sensors would always set their sector to be the one with the most targets. This particular service allocation should, usually, be the best. However, it might not always be the optimal solution. For example, if seven targets are clustered together and the eighth is on another part of the field, it would be better if six sensor agents pointed towards the cluster of targets while the remaining two sensor agents pointed towards the stray target rather than having all sensor agents point towards the cluster of targets. At the other extreme, if all the targets send their bids to

only one sensor then they will minimize communications but then the sensors will point to the sector from which they received a message, i.e., an allocation which is likely to be suboptimal.

These simulations explore the ruggedness of the system’s global utility landscape and the dynamics of the agents’ exploration of this landscape. If the agents were to always converge on a local (non-global) optimum then we would deduce that this problem domain has a very rugged utility landscape. On the other hand, if they usually manage to reach the global optimum then we could deduce a smooth utility landscape.

4. TEST RESULTS

In each of our tests we set the number of agents that each target will send its bid to, that is, the number of neighbors, to a fixed number. Given this fixed number of neighbors, we then generated 100 random placements of agents on the field and ran our bidding algorithm 10 times on each of those placements. Finally, we plotted the average solution quality, over the 10 runs, as a function of time for each of the 100 different placements. The solution quality is given by the ratio

$$\alpha = \frac{\text{CurrentUtility}}{\text{GlobalOptimalUtility}} \quad [\text{eq 13}]$$

when $\alpha = 1$ the run has reached the global optimum. Since the number of agents is small, we were able to calculate the global optimum using a brute-force method. Specifically, there are $3^7 = 2187$ possible configurations times 100 random placements leads to 218700 combinations that we had to check for each run in order to find the global optimum using brute-force.

Due to the large number of configurations, using more than 7 sensors to run the simulation was not practical. Notice, however, that our algorithm is much faster than this brute-force search which we perform only to confirm that our search does find the global optimum. In our tests there were always seven target agents and seven sensor agents. We varied the number of neighbors from 1 to 7. If the target can only communicate with one other sensor, the sensors will likely have very little information for making their decision, while if all targets communicate with all seven sensors, then each sensor will generally be able to point to the sector with the most targets. However, because these decisions are made in an asynchronous manner, it is possible that some sensors may not always receive all the bids before decisions are due. The targets always send their bids to the sensors that are closest to them.

The results from our experiments are shown in Figure 1 where we can see that there is a transition in the system’s performance as the number of neighbors goes from three to five. That is, if the targets only send their bids to three sensors then it is almost certain that the system will stay in a configuration that has a very low global utility. However, if the targets send their bids to five sensors, then it is almost guaranteed (98% of the time) that the

system will reach the globally optimal allocation.

5. RELATED WORK

There is ongoing work in the field of complexity that attempts to study the dynamics of complex adaptive systems [4]. Our approach is based on ideas borrowed from the use of NK landscapes for the analysis of co-evolving systems. As such, we are using some of the results from that field. However, complexity theory is more concerned with explaining the dynamic behavior of existing systems, while we are more concerned with the engineering of multi-agent systems for distributed service allocation. The Collective Intelligence (COIN) framework [12] shares many of the same goals of our research. They start with a global utility function from which they derive the rewards functions for each agent. The agents are assumed to use some form of reinforcement learning. They show that the global utility is maximized when using their prescribed reward functions. They do not, however, consider how agent communication might affect the individual agent's utility landscape. The task allocation problem has been studied in [7], but the service allocation problem we present in this paper has received very little attention. There is also work being done on the analysis of the dynamics of multi-agent systems for other domains such as e-commerce [5] and automated manufacturing [6]. It is possible that extensions to our approach will shed some light into the dynamics of these domains.

6. CONCLUSIONS

We have formalized the service allocation problem and examined a general approach to solving problems of this type. The approach involves the use of utility-maximizing agents that represent the resources and the services. A simple form of bidding is used for communication. An analysis of this approach reveals that it implements a form of distributed hill-climbing, where each agent climbs its own utility landscape and not the global utility landscape. However, we showed that increasing the amount of communication among the agents forces each individual agent's landscape to become increasingly correlated to the global landscape.

These theoretical results were then verified in our implementation of a sensor allocation problem, which is an instance of a service allocation problem. Furthermore, the simulations allowed us to determine the location of a phase transition in the amount of communication needed for the system to consistently arrive at the globally optimal service allocation. More generally, we have shown how a service allocation problem can be viewed as a distributed search by multiple agents over multiple landscapes. We also showed how the correlation between the global utility landscape and the individual agent's utility landscape depends on the amount and type of inter-agent communication. Specifically, we have shown that increased communications leads to a higher correlation between the global and individual utility landscapes, which increases the probability that the global optimum will be reached. Of course, the success of the search still depends on the connectivity of the search space, which will vary from domain to domain.

We expect that our general approach can be applied to the design of any multi-agent systems whose desired behavior is given by a global utility function but whose agents must act selfishly. Our future work includes the study of how the system will behave under perturbations. For example, as the target moves it perturbs the current allocation and the global optimum might change. We also hope to characterize the local to global utility function correlation for different service allocation problems and the expected time to find the global optimum under various amounts of communication.

B. SC TRACKER

1. INTRODUCTION

This part of the report describes a multi-target tracking system developed to support the ANTS challenge problem (CP). The tracking system is based on a Bayesian approach [18] that estimates target locations and velocities by fusing amplitude and frequency measurements obtained from Doppler Radar sensors. The tracking system was designed to operate in conjunction with Doppler Radar sensors developed by the BAE Sanders [26], or their simulation version, called RADSIM, developed at the Air Force Research Laboratory [25]. The simulated and hardware sensors are devices that return values of amplitude and frequency. The amplitude is proportional to the distance between the sensor and the target, and the frequency is proportional to the angular velocity of the target, as seen by the sensor. Equations 4 and 5 describe the amplitude and frequency models; Figure 4 shows the geometry involved to determine amplitude and frequency given the target velocity, its location, and the location of the sensor.

The tracker system fuses data into a probability space to obtain an estimate of a target's state at any given time [15, 18, 21, 22, 23]. The state of a target, represented by its location and velocity, are not directly observable from the sensors data. Instead, the sensors return amplitude and frequency measurements that are related to the target location, its velocity, and sensor parameters, including sensor location, sector selection, gain, type of measurement, beam width, number of pulses within a measurement, and power.

2. SENSOR MODEL

In general, a target moving in a two-dimensional plane can be represented by a vector T_s given by,

$$T_s = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \quad [\text{eq 1}]$$

Where x and y are the target's coordinates, and v_x and v_y are the target's velocity in the x and y direction, respectively, in a fixed reference frame. Given that the i^{th} sensor returns a measurement m_p , then a sensor model for that measurement, m_p , can be expressed as

$$m_i = M_{m,i}(T_s, \theta_i) \quad [\text{eq 2}]$$

The sensor model $M_{m,i}$ is a function of the target state T_s and the parameters θ_i of the model. The model could be based on fundamental principles of physics or be formulated from empirical observations [20, 22, 24]. In either case it is likely that the functional

model $M_{m,i}$ might not explain the measurement m_i completely. Errors may occur because the model might be insufficient and may not take into consideration all the possible interactions within the system or because the experimental conditions may go beyond control. Hence a better representation of the model is,

$$m_i = M_{m,i}(T_s, \theta_i) + \varepsilon_{m,i} \quad [\text{eq 3}]$$

Where $\varepsilon_{m,i}$ is the error term in the m^{th} measurement of the i^{th} sensor. During the development of the tracking systems two sensor models were used. The first sensor model was given by the system of equations:

$$\begin{aligned} A_k &= K \exp \left(- \frac{\alpha_k^2}{\left(\frac{\pi}{3}\right)^2} \right) \\ f_k &= C \left| \dot{R}_k \right| \end{aligned} \quad [\text{eq 4}]$$

Where A_k is a value of amplitude and f_k is a value of frequency given the range R_k (the distance between the sensor and the target). K , C , and α , are parameters of the sensor model. This model had a correction factor given by equation 5

$$\begin{aligned} A_{k,n} &= A_k(1 + 0.05N(0,1)) + B_{k,i}(1 + 0.05N(0,1)) \\ f_{k,n} &= \frac{f_k}{1 + \varepsilon_f f_k} \end{aligned} \quad [\text{eq 5}]$$

i.e., the value of amplitude was affected by Gaussian noise compounded at a level of 5%, while the value of frequency had an error factor of ε_f .

The first sensor model was replaced by a model that contained a noise factor given by a uniform distribution around $[-0.2, 0.2]$. The second model considered amplitude as a function of frequency, which in turn was a function of an interpolation function C_f .

$$A = \frac{KC_f(f) \exp \left(- \frac{\alpha^2}{BW^2} \right) [1 + U(-0.2, 0.2)]}{R^\gamma} \quad [\text{eq 6}]$$

3. PROCESS MODEL

In addition to the sensor model, the tracker uses a process model to predict target location and velocity [14, 16, 17, 18, 19, 25, 26]. The process model contains data structures and algorithms that were specifically designed to cope with the asynchronous nature of the

CP and the incompatible measurement spaces of the data sources. The process model relates the history of the target states to potential future states. Let $H(t)$ denote the history of target states until time t . If $T_s(t)$ denotes the target state at time t , then

$$H(t) = [T_s(1) \quad \dots \quad T_s(t)]^T \quad [\text{eq 7}]$$

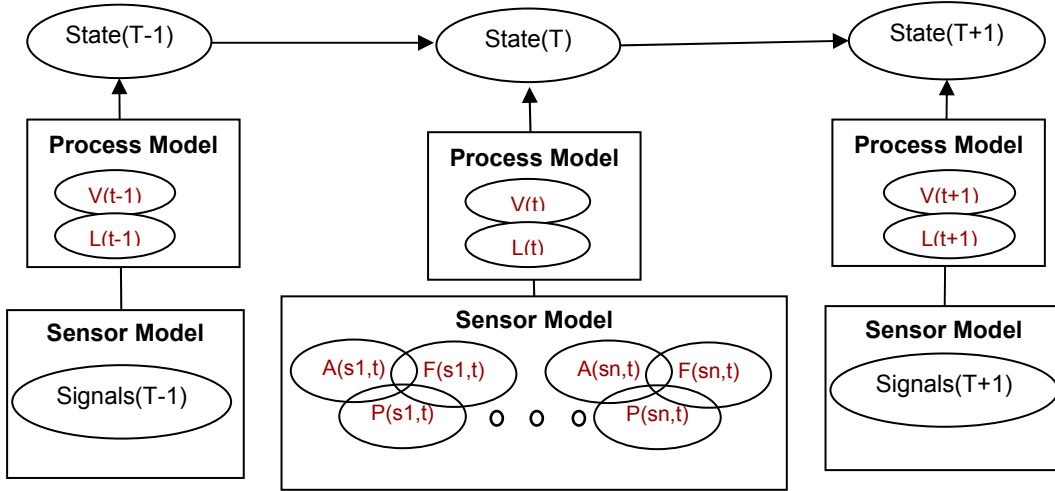
i.e., $H(t)$ is the set of all target states until time t . A general motion model of the target's trajectory can be ascertained from the history of the target states. Given the current target state and the distribution of the velocity obtained from that history, it is feasible to project that information into the future and obtain a distribution of the most likely target locations. This information can be used with the measurements obtained at time $t+1$ to locate the target.

$$T_s(t+1) = T_s(m_1, \dots, m_N, H(t)) \quad [\text{eq 8}]$$

Where m_1, \dots, m_N are the measurements from N sensors at time $t+1$.

The process model assumes that the clocks on all the sensors are synchronized. However the hardware implementation of the challenge problem is not compatible with this assumption. One of the constraints of the CP architecture is that a sensor returns one single measurement during a time t . Yet, to estimate the target state at any time, it is necessary to have as many measurements as possible during that period. Another constraint is that measurements from a single sensor are not enough to track a target with a high degree of accuracy.

To overcome these limitations, the process model assumes that the motion of a target follows the laws of inertia. This assumption is implemented using the concept of motion through time, which is implemented by two structures, called the time frames and the motion model, described in the following sections.



where

$V(t-1), V(t), V(t+1)$ are the estimates of target velocity,
 $L(t-1), L(t), L(t+1)$ are the estimates of target location,
 $A(s1,t).. A(sn,t)$ are the amplitude measurements from sensors $s1 .. sn$, at time= t .
 $F(s1,t).. F(sn,t)$ are the frequency measurements from sensors $s1 .. sn$, at time= t .
 $P(s1,t).. P(sn,t)$ are the operational parameters of sensors $s1 .. sn$, at time= t .

Figure 1: Interactions Between the State and Process Models

4. TIME FRAMES

Given the asynchronous nature of the CP infrastructure, information from the sensors is never guaranteed to arrive at the tracker on time. In fact, data arriving with significant delay are the norm rather than the exception. To cope with this constraint, it was hypothesized that changes in the target state are very small during the period ΔT_f , and that all sensor measurements received in that period can be fused to obtain an approximate estimate of that target state. This assumption led to the discretization of time into a series of time frames of length ΔT_f . It is implied that the target location in a time frame ΔT_f is constant. Theoretically the best results are obtained by making ΔT_f approach zero. However it is not feasible to obtain sufficient measurements to estimate target state under that condition.

The tracker uses time frames to accommodate incoming data based on the time stamp at which data was produced by the sensors. These data, which consist of amplitude and frequency, are used to update the estimation of target location and velocity at the end of each time frame. Due to this scheme, data belonging to the past, received later because of communication or processor delays, is not ignored. Instead, those data are used to update the tracker at the current time frame, or at times $T-1, T-2, T-3$, etc., if data for those frames arrive during the current time frame. Since measurements are continuously added

under a timed-queued scheme, whenever the tracker updates location and velocity for the time-frame at time T , the tracker first checks if there is any data for previous time frames. If new data indeed came for previous time frames, then the estimates of target state for those frames are updated, and the new predictions are propagated forward. The tracker ignores measurements whose time stamp is delayed more than 4 time frames. Although this number is easily set up as an operational parameter, the number is consistent with theoretical results from hidden Markov models, and constitutes a good compromise that has little impact on performance [18].

Figure 2 shows the sequence of data update that would take place at frame 4000, as new “old” data from previous frames are detected. The circles are color coded: Green means that the data item arrived on time, yellow that it arrived one time frame later, blue two times later, red three time frames later. The location in the time frame is such as to indicate approximately the time at which the item arrived. For example, in time frame 1000, data items were produced at the sensors at times 1000, 1250, 1500, and 1800. Of these, only the first two items arrived on time, the data item for $t=1500$ arrived one frame later, and the data item for $t=1800$ arrived three frames later. Similarly, the data item for $t=2800$ arrived at that frame two frames later, etc.

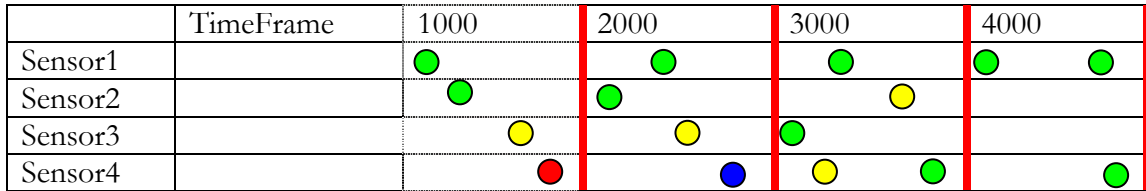


Figure 2 Time Frames

In a given time frame some sensors may return more than one measurement. This would be illustrated in the last time frame, where Sensor1 returns measurements at times $t=4100$ and $t=4800$. Two approaches were taken to deal with this issue. The first approach considered only the most recent measurement and ignored the previous measurements within the frame. Each measurement has an intrinsic background noise that is not constant. If the assumption is made that background noise fluctuates around zero, then averaging the measurements actually diminishes the effect of fluctuations, and the result is a more accurate representation of the target state. Therefore the approach adopted by the tracker is to average the measurements within the same frame.

Although some of these concepts may not be applicable *as-is* to other tracking problems, the time frame concept offers a reasonable insight that is applicable to many tracking platforms.

5. LOCATION MODEL

The measurements obtained from the hardware sensors or from RADSIM are

stored in a time-stamped queue of time frames. Each time frame has a start time and an end time. The end-time of frame $T-1$ is the start time of frame T . A measurement belongs to a time frame if that measurement's time is between the start and end time of that time frame.

The tracker uses an internal clock to keep track of time. As time progresses, the clock reaches a point at which the tracker time is greater than the end time of the current time frame. When that occurs, a method, called *updateTracker*, is invoked. The method first checks if new data has just arrived for previous time frames. If new data exist in those frames then the frames are updated before updating the current time frame, and the update proceeds forward. The method uses two data handlers to manage amplitude and frequency measurements from the sensors. The amplitude handler uses the sensor model to fuse information and obtain a probabilistic distribution of the target location. This distribution is updated using the motion model, which is in turn an expression of the history of target states. The frequency handler uses the frequency measurements to estimate the target's velocity.

6. AMPLITUDE HANDLER

The two-dimensional target domain is divided into a set of $m \times n$ grid cells [13, 18]. The measurements from the sensors are used to compute the probability of a target being in each grid cell. The grid cell is mapped to the center of the cell as a single point.

Consider two amplitude measurements A_1, A_2 obtained from two different sensors S_1, S_2 . A distribution of target location in each cell of the grid can be obtained by fusing these measurements. The probability of the target being in each cell (x_i, y_j) where $i = 1 \dots m$ and $j = 1 \dots n$ is computed given the measurements A_1, A_2 . Let this probability be denoted by $P(X = x_i, Y = y_j | M_1 = A_1, M_2 = A_2)$. The measurements A_1, A_2 are assumed to be independent; thus measurements from one sensor do not have any affect on the measurements of another sensor. This implies that,

$$P(X = x_i, Y = y_j | M_1 = A_1, M_2 = A_2) = P(X = x_i, Y = y_j | M_1 = A_1) \times P(X = x_i, Y = y_j | M_2 = A_2) \quad [\text{eq 9}]$$

From Bayes theorem we can write each term in the right hand side of the equation as,

$$P(X = x_i, Y = y_j | M_1 = A_1) = \frac{P(M_1 = A_1 | X = x_i, Y = y_j) P(X = x_i, Y = y_j)}{\sum_{i=1}^m \sum_{j=1}^n P(M_1 = A_1 | X = x_i, Y = y_j) P(X = x_i, Y = y_j)} \quad [\text{eq 10}]$$

$$P(X = x_i, Y = y_j | M_2 = A_2) = \frac{P(M_2 = A_2 | X = x_i, Y = y_j) P(X = x_i, Y = y_j)}{\sum_{i=1}^m \sum_{j=1}^n P(M_2 = A_2 | X = x_i, Y = y_j) P(X = x_i, Y = y_j)}$$

The term $P(X = x_i, Y = y_j)$ is the *a-prior* probability associated with the target being in location (x_i, y_j) and $P(X = x_i, Y = y_j | M_1 = A_1)$ is the *posteriori* probability after the

measurement A_1 was obtained. Assuming a model for the amplitude that is a function of the target location,

$$A = M_A(X, Y, \theta) + \varepsilon_A \quad [\text{eq 11}]$$

in equation 11 θ is a sensor-specific parameter vector and ε_A is the measurement error. If ε_A is assumed to be distributed normally with a standard deviation σ_A then,

$$P(X = x_i, Y = y_j | M_1 = A_1) \propto \frac{1}{\sqrt{2\pi\sigma_{1,A}^2}} \exp\left(-\frac{(M_{1,A}(x_i, y_j, \theta_1) - A_1)^2}{2\sigma_{1,A}^2}\right) P(X = x_i, Y = y_j) \quad [\text{eq 12}]$$

Using equation 12 the posterior probability distribution of the target location given measurements from N different sensors can be obtained. If the measurement error for each sensor is assumed to be normal, the equation above becomes

$$P(X = x_i, Y = y_j | M_1 = A_1, \dots, M_N = A_N) \propto \frac{1}{\sqrt{\prod_{k=1}^N \sigma_{k,A}^2}} \exp\left(-\sum_{k=1}^N \frac{(M_{k,A}(x_i, y_j, \theta_k) - A_k)^2}{2\sigma_{k,A}^2}\right) \quad [\text{eq 13}]$$

$$P(X = x_i, Y = y_j | M_1 = A_1, \dots, M_N = A_N) = \frac{1}{\sqrt{\prod_{k=1}^N \sigma_{k,A}^2}} \exp\left(-\sum_{k=1}^N \frac{(M_{k,A}(x_i, y_j, \theta_k) - A_k)^2}{2\sigma_{k,A}^2}\right) \frac{1}{C}$$

Where $\sigma_{k,A}^2$ is the variance in amplitude measurements, θ_k is the parameter vector, $M_{k,A}(x_i, y_j, \theta_k)$ is the amplitude model of the k^{th} sensor, A_k is the measurement received from that sensor, and C is a normalization constant.

An example that illustrates the fusion of two amplitude measurements to obtain a probability distribution of target location is shown in Figure 3.

Clearly, it is not possible to track a target with just one or two amplitude measurements; the region of high probability is too wide. To resolve this issue, either more measurements are needed, or a motion model that could resolve the ambiguities is needed. Ideally, the motion model would use the history of target states to estimate velocity.

7. FREQUENCY HANDLER

A frequency model for the sensor relates the velocity of the target to the sensor's frequency measurement. It is assumed that frequency is a linear function of the radial velocity with respect to the sensor, i.e.,

$$f_k = C_{k,f} \left| \dot{R}_k \right| \quad [\text{eq 14}]$$

Where f_k is the frequency measurement, $C_{k,f}$ is a constant, \dot{R}_k is the radial velocity with respect to k^{th} sensor.

A single frequency measurement is not sufficient to solve for target's velocity; measurements from at least two sensors are required. Even in the presence of two measurements, a technique to realize the radial direction (whether the target is moving towards or away from the sensor) is required.

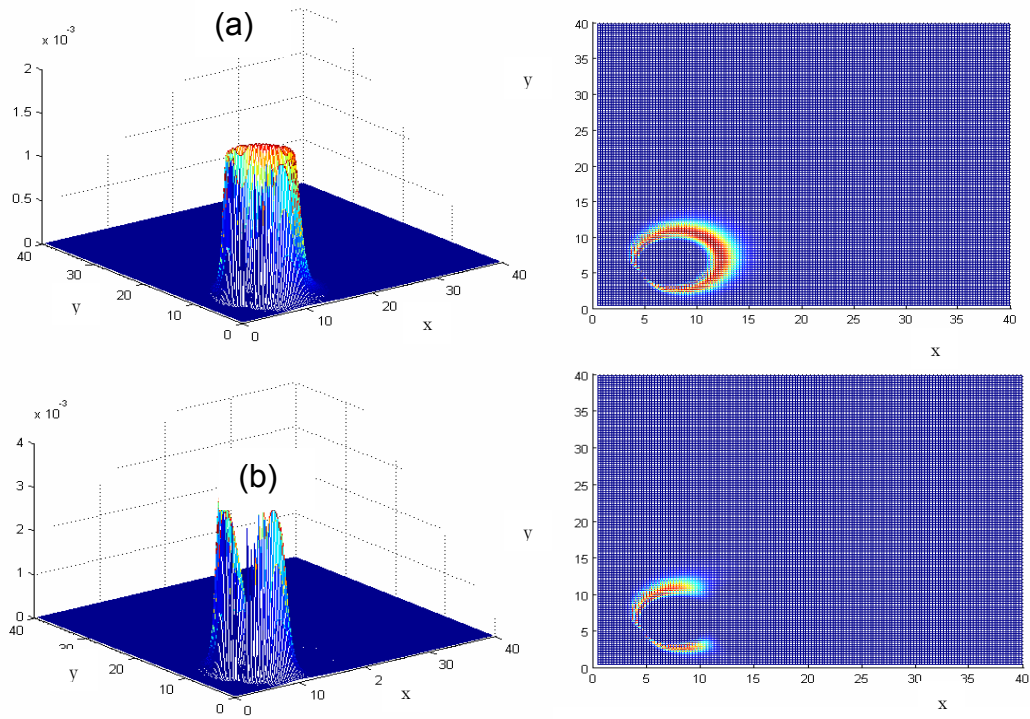
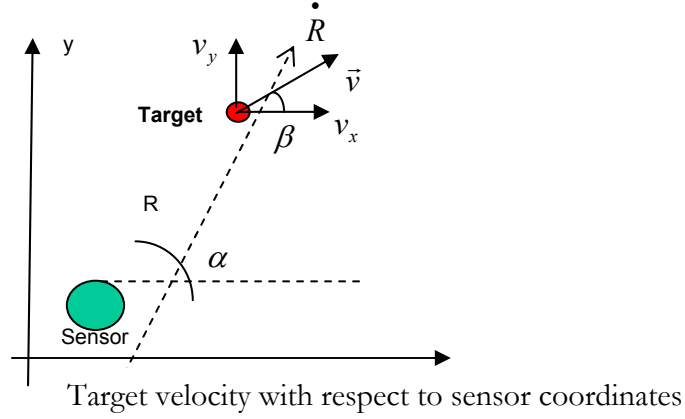


Figure 3 (a) Target Probability Distribution From Single Amplitude Measurement.
 (b) Target Probability Distribution From Fusing two Amplitude Measurements

Figure 4 shows the relation between the sensor location and the target velocity.



\vec{v} : Target velocity

\dot{R} : Radial velocity with respect to the sensor

α : Angle between target and sensor

v_x, v_y : Velocity of the target in x and y direction respectively

(x_s, y_s) : Sensor coordinates

(x_T, y_T) : Target coordinates

R : Target distance from the sensor

Figure 4: Relation Between the Sensor Location and the Target Velocity

The frequency measurements from the sensors can be converted into radial velocities using the equation for frequency. Let \dot{R}_i denote the radial velocity of a target computed from the frequency of the i^{th} sensor. The target velocity components v_x, v_y are related to \dot{R}_i as,

$$v_x \cos \alpha_i + v_y \sin \alpha_i = \dot{R}_i \quad [\text{eq 15}]$$

Where α_i denotes the angle between the i^{th} sensor and target coordinates. The tracker solves this equation simultaneously for v_x and v_y whenever frequency measurements are obtained from more than two sensors within the same time frame.

8. MOTION MODEL

A motion model was developed using recent history of target states [14, 18, 20]. This can be viewed as a learning technique of the target's motion, based on the assumption that the target is subjected to the laws of inertia. This assumption is consistent with the CP; the targets being detected have a constrained pattern of motion. Other "common-sense" constraints, like the fact that the target cannot change its direction abruptly, or cannot move faster than a pre-specified maximum velocity, are also assumed. Some

of the advantages of the motion model are,

- Since the motion model captures the velocity distribution, using the model in conjunction with the amplitude handler results in more accurate distributions of target location. As discussed in a previous section, the amplitude handler sometimes gives two clusters as the most likely target locations. Using a probabilistic weighted mean in such circumstances to predict target location may lead to erroneous estimations. Multi-modal distributions of target location can be avoided most of the times by using the motion model.
- The motion model can help to predict future target states and hence optimize the use of CP resources (sensors) from this information.
- Estimates of target location can be obtained even when the sensors fail to report measurements.

The motion model builds a probabilistic estimate from previous target states. There are two techniques to compute target velocity. One uses the previous target locations to find the fraction of distance that the target has moved in time t . The second uses frequency measurements from the sensors to compute the target velocity [14, 18, 20].

A fixed number of time frames N_f in the past are used to build the motion model.

The velocity of the target has two components, the magnitude and the direction of motion. Hence the model is split into two parts, one for the distance model and one for the direction model. An average value of the speed and the direction can be computed from the previous time frames.

Let $\bar{v}, \bar{\beta}$ be the average speed and direction computed from N_f time frames. Let (x_p, y_p) be the previous target location estimated at time t_p . Assume that the amplitude model gives the target location distribution from the amplitude measurements. The distribution is updated using the distance and the direction models.

- A new target location (x_c, y_c) at time t_c is computed using

$$\begin{aligned} x_c &= x_p + (\bar{v} \cos \bar{\beta})(t_c - t_p) \\ y_c &= y_p + (\bar{v} \sin \bar{\beta})(t_c - t_p) \end{aligned} \quad [\text{eq 16}]$$

- *Distance Model:* The target domain is divided into $m \times n$ grid cells. Let (x_i, y_j) represent a cell in the grid. If d_{ij} is the distance between the points (x_i, y_j) and (x_c, y_c) , the probability of a target being in cell (x_i, y_j) is computed using a Gaussian model

$$P(x_i, y_j | H(t_c)) \propto \frac{1}{\sqrt{2\pi\sigma_d^2}} \exp\left(-\frac{d_{ij}^2}{\sigma_d^2}\right) \quad [\text{eq 17}]$$

Where $H(t_c)$ is the target history considered, σ_d is the standard deviation

assumed for the distance model.

- *Direction Model:* Let β_{ij} denote the angle between (x_i, y_j) and (x_c, y_c) . Assume a standard deviation σ_β in the direction estimate $\bar{\beta}$. The probability estimate for the location (x_i, y_j) using the direction model is given by

$$P(x_i, y_j | H(t_c)) \propto \frac{1}{\sqrt{2\pi\sigma_\beta^2}} \exp\left(-\frac{(\beta_{ij} - \bar{\beta})^2}{\sigma_\beta^2}\right) \quad [\text{eq 18}]$$

The joint distribution of the distance and direction model gives the target location distribution given the history $H(t_c)$, expressed as:

$$P(x_i, y_j | H(t_c)) \propto \frac{1}{\sqrt{\sigma_\beta^2 \sigma_d^2}} \exp\left(-\left(\frac{d_{ij}^2}{\sigma_d^2} + \frac{(\beta_{ij} - \bar{\beta})^2}{\sigma_\beta^2}\right)\right) \quad [\text{eq 19}]$$

9. TARGET LOCATION

A joint probability distribution for amplitude measurements and motion model is computed to estimate target's location. Combining the equations of the target model and the motion model we obtain the joint probability distribution

$$P(X = x_i, Y = y_j | A_1, \dots, A_N, H) \propto \exp\left(-\left(\sum_{k=1}^N \frac{(M_{k,A}(x_i, y_j, \theta_k) - A_k)^2}{2\sigma_{k,A}^2}\right) - \left(\frac{d_{ij}^2}{\sigma_d^2} + \frac{(\beta_{ij} - \bar{\beta})^2}{\sigma_\beta^2}\right)\right) \quad [\text{eq 20}]$$

The target location (\bar{X}_T, \bar{Y}_T) is given by,

$$(\bar{X}_T, \bar{Y}_T) = \left(\frac{\sum_{i=0}^{i=m} \sum_{j=0}^{j=n} P(x_i, y_j | A_1, \dots, A_N, H) x_i}{\sum_{i=0}^{i=m} \sum_{j=0}^{j=n} P(x_i, y_j | A_1, \dots, A_N, H)}, \frac{\sum_{i=0}^{i=m} \sum_{j=0}^{j=n} P(x_i, y_j | A_1, \dots, A_N, H) y_j}{\sum_{i=0}^{i=m} \sum_{j=0}^{j=n} P(x_i, y_j | A_1, \dots, A_N, H)} \right) \quad [\text{eq 21}]$$

Figure 5 illustrates the joint of the two equations.

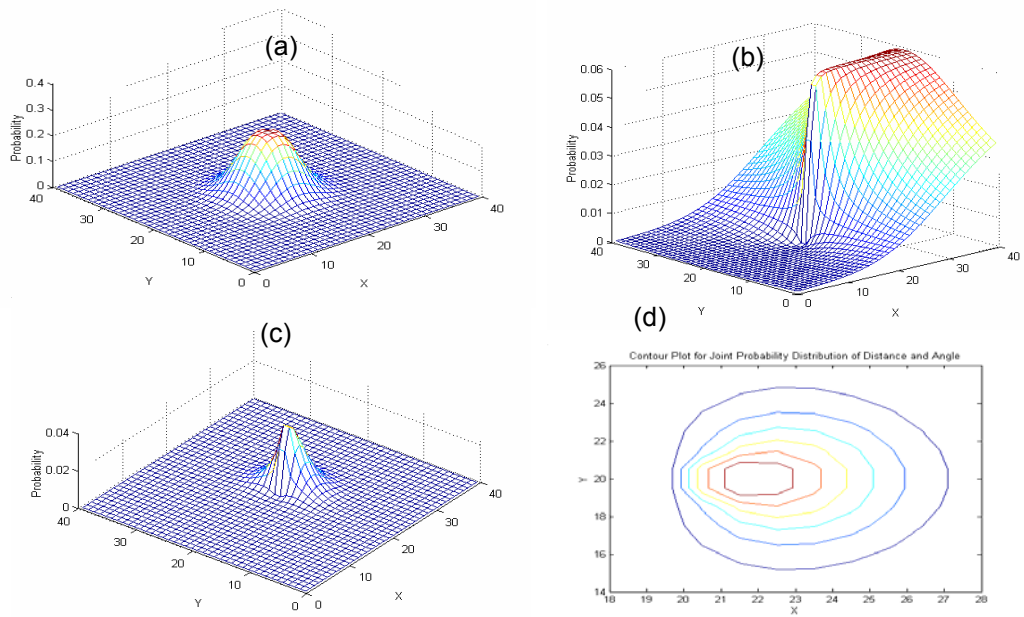


Figure 5 Motion model: (a) Distance Probability Model
 (b) Angle Probability Model
 (c) Joint Probability Distribution From Distance and Angle Probability
 (d) Contour Plot of the Joint Probability Distribution

10. RESULTS

The tracker was tested using several target configurations. Figure 6 shows a typical set of results, obtained using the tracker under an “omniscient” controller. Since this controller knew where the target was, the controller instructed the tracker to get measurements from the “right sectors,” providing a means to establish a base-line estimation of the tracker performance. Even under these conditions it should be noted that the measurements were not assured to arrive at the right time. The diamond shaped dots are the target’s predictions. The square dots are the true locations. As the figure shows, there is an insignificant difference between the two sets of data. The Root Mean Square error (RMS) was 0.4 ft.

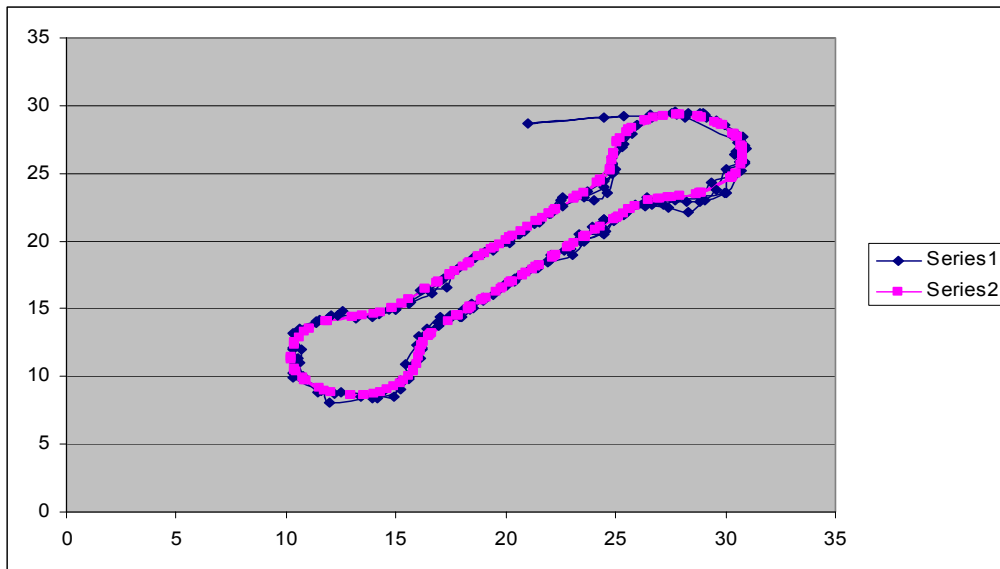


Figure 6 Results Obtained with the SC Tracker During the First Demonstration

11. MULTIPLE TARGET TRACKING

The multiple target version was grown from the original single-target version. In essence, instead of maintaining target state for a single target, the multi-target version maintained state for N targets. Since the allocation of sensor resources was decided by the controllers, the association of target to sensors was not considered in the multi-target version.

The final version of the multiple target tracker was further adapted so that instead of keeping multiple states within the tracker, multiple instances of the tracker were spawned by their controller agents. The resulting architecture is that shown in Figure 7. Since the controller agent was the piece of software in which negotiation was to be made, the controller determined which resources were needed, and based on those decisions, sensor measurements were feed into the appropriate instance of a tracker thread, where

predictions of target state were made and communicated to the controller.

As in previous demos, the performance of the tracker was quite acceptable, as shown in Figure 7.

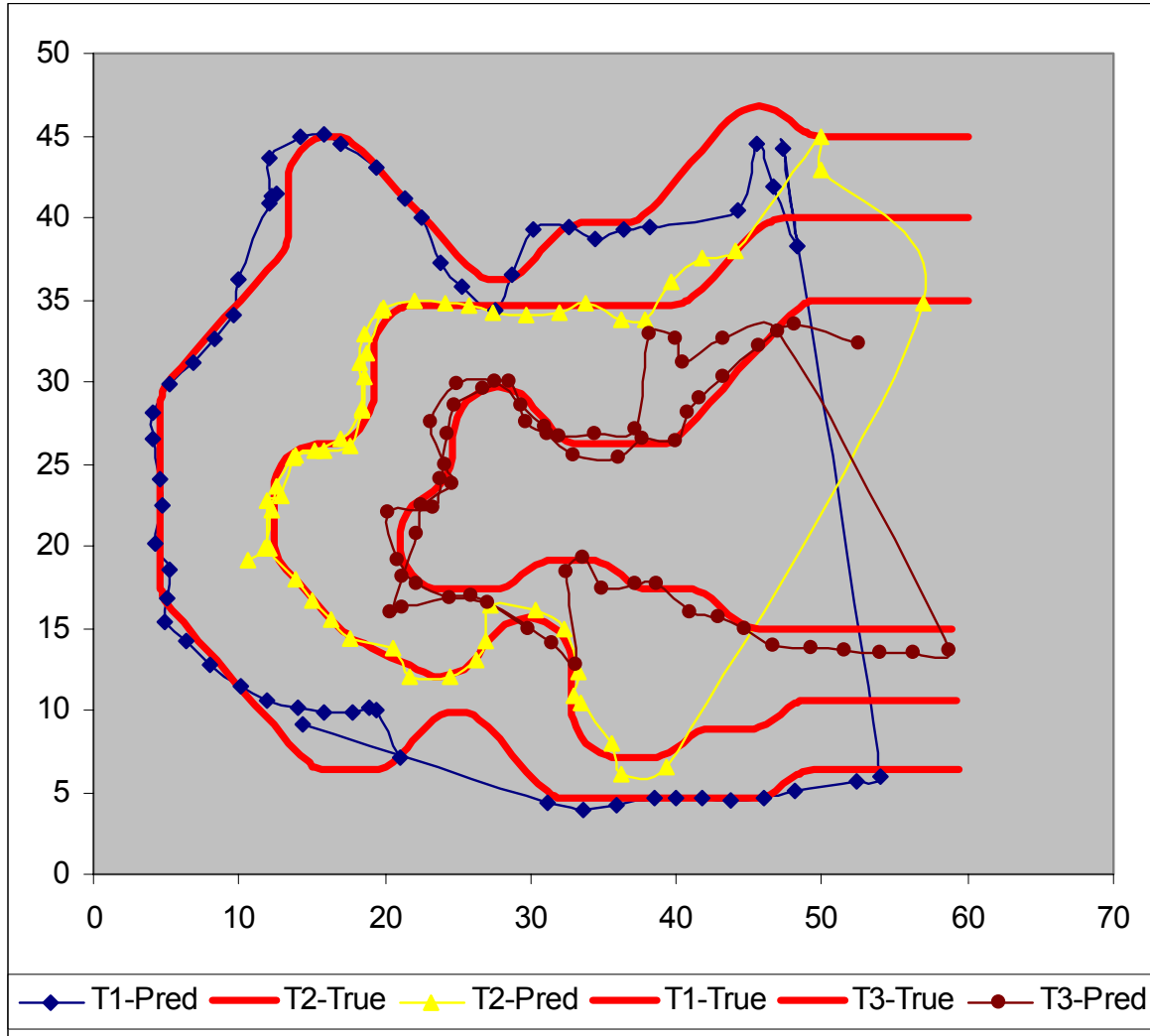


Figure 7 Results Obtained with the SC Tracker During the Final Demonstration

12. CONCLUSIONS

We have presented a multiple-target tracking system that uses probabilistic data fusion to estimate target location and velocity from sensor data. The system's architecture provides a high degree of separation between the data sources, the tracking logic, and the state transitions, which could be adapted to other tracking and data fusion scenarios.

The two major components of the tracker, the state model and the process model, are based on Bayesian estimation theory. This makes it possible to fuse data from different

measurement spaces into a rich, unified, representation scheme, and offers the following advantages:

- Robust operational behavior: Multi-sensor data fusion has an increased robustness when compared to single sensor data fusion. When one sensor becomes unavailable or is inoperative, other sensors can provide information about the environment.
- Extended spatial and temporal coverage: Some parts of the environment may not be accessible to some sensors due to range limitations. This occurs especially when the environment being scanned is vast. In such scenarios, multiple sensors that are mounted at different locations can maximize the regions of scanning. Multi-sensor data fusion provides increased temporal coverage as some sensors can provide information when others cannot.
- Increased confidence: The confidence in detection of targets is increased in multi-sensor data fusion. Single target location can be confirmed by more than one sensor and this increases the users confidence in target detection.
- Reduced ambiguity: Joint information from multiple sensors can reduce the set of beliefs about the target.
- Decreased costs: Multiple, inexpensive sensors can replace expensive single sensor architecture at a significant reduction of cost.
- Improved detection: Integrating measurements from multiple sensors can reduce signal to noise ratio, which ensures improved detection.

REFERENCES

- [1] A. D. Baker, H. V. Parunak, and K. Erol. Agents and the internet: Infrastructure for mass customization. *IEEE Internet Computing*, 3(5):62–69, September-October 1999.
- [2] E. H. Durfee, T. Mullen, S. Park, J. M. Vidal, and P. Weinstein. The dynamics of the UMDL service market society. In M. Klusch and G. Weiß, editors, *Cooperative Information Agents II*, LNAI, pages 55–78. Springer, 1998.
- [3] J. M. Epstein and R. L. Axtell. *Growing Artificial Societies : Social Science from the Bottom Up*. Brookings Institute, 1996.
- [4] S. Kau_man. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.
- [5] J. O. Kephart, J. E. Hanson, and A. R. Greenwald. Dynamic pricing by software agents. *Computer Networks*, 32(6):731–752, 2000.
- [6] H. V. D. Parunak. “go to the ant”: Engineering principles from natural agent systems. *Annals of Operation Research*, 75:69–101, 1997.
- [7] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter*. The MIT Press, Cambridge, MA, 1994.
- [8] T. W. Sandholm. Necessary and sufficient contract types for optimal task allocation. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1997.
- [9] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1981.
- [10] G. Weiss, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.
- [11] M. P. Wellman. Market-oriented programming: Some early lessons. In S. Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.
- [12] D. H. Wolpert and K. Tumer. An introduction to collective intelligence. Technical report, ACM Computing Research Repository, 1999. cs.LG/9908014.
- [13] Alberto Elfes, Using Occupancy Grids for Mobile Robot Perception and Navigation. *IEEE Computer*, 22(6): 46-57 (1989).
- [14] Bar-Shalom, Y. and T.E. Fortmann, “Tracking and Data Association”. Academic Press: New York, 1987.
- [15] Berler, A. and Shimony, S. E., “Bayes Networks for Sonar Sensor Fusion”, *Proceedings of the 13th Conference on Uncertainty in AI*, 1997.
- [16] Gregory Francis Welch, “Incremental tracking with incomplete information”, PhD thesis, UNC Chapel hill, 1996.
- [17] Hughes, T.J. “Sensor Fusion in a Military Avionics Environment.” *Measurement and Control*. Sept. 1989: (203-205)..
- [18] L. D. Stone, C.A. Barlow, and T. L. Corwin, “Bayesian Multiple Target Tracking”, Artech House, Norwood, MA, 1999.
- [19] L. Y. Pao. "A Measurement Reconstruction Approach for Distributed Multisensor Fusion," *J. Guidance, Control, and Dynamics*, 19(4): 842-847, July-Aug. 1996.
- [20] M. M. Kokar, M. D. Bedworth and K. B. Frankel. “[A Reference Model for Data Fusion Systems](#)”, In *Sensor Fusion: Architectures, Algorithms and Applications IV*, 191-202, SPIE, 2000.
- [21] M. M. Kokar, J. A. Tomasik and J. Weyman. [A Formal Approach to Information Fusion](#). Proceedings of the Second International Conference on Information Fusion (Fusion'99), Vol.I, 133-140, July 1999.
- [22] N Okello and D. Tang and D W McMichael, “Tracker: A Sensor Fusion Emulator for Generalised Tracking”, Proceedings of Information Decision and Control 99, 359—364, February. 1999.

- [23] Waltz, E. and J. Llinas. *Multisensor Data Fusion*. Artech House, Norwood, MA 1990.
- [24] Wen, W. and H.F. Durrant-Whyte. "Model-based Multi-sensor Data Fusion.", Proceedings. *IEEE International Conference on Robotics and Automation*. 12-14 May 1992: Nice, France. IEEE: Los Alamitos, CA, 1992. Vol. 2: (17206).
- [25] Lawton, J.L. *The RADSIM Simulator*, in Distributed Sensor Networks, Ed. By V. Lesser, C. Ortiz & M. Tambe, Kluwer Academic Press, 2003.
- [26] Zeman, P. and Gaughan, M. *Challenge Problem Testbed*, in Distributed Sensor Networks, Ed. By V. Lesser, C. Ortiz & M. Tambe, Kluwer Academic Press, 2003.