Improving Surveillance Using Cooperative Target Observation

by

Rashi Aswani, Sai Krishna Munnangi, Praveen Paruchuri

in

Thirty-First AAAI Conference on Artificial Intelligence
(*AAAI-17*)

Hilton San Francisco, San Francisco, California, USA

Report No: IIIT/TR/2017/-1

Centre for Data Engineering
International Institute of Information Technology
Hyderabad - 500 032, INDIA
February 2017

Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Improving Surveillance Using Cooperative Target Observation

Rashi Aswani, Sai Krishna Munnangi and **Praveen Paruchuri**
Machine Learning Lab, Kohli Center on Intelligent Systems
International Institute of Information Technology - Hyderabad, India
{rashi.aswani, krishna.munnangi}@research.iiit.ac.in, praveen.p@iiit.ac.in

Abstract

The Cooperative Target Observation (CTO) problem has been of great interest in the multi-agents and robotics literature due to the problem being at the core of a number of applications including surveillance. In CTO problem, the observer agents attempt to maximize the collective time during which each moving target is being observed by at least one observer in the area of interest. However, most of the prior works for the CTO problem consider the targets movement to be Randomized. Given our focus on surveillance domain, we modify this assumption to make the targets strategic and present two target strategies namely Straight-line strategy and Controlled Randomization strategy. We then modify the observer strategy proposed in the literature based on the K-means algorithm to introduce five variants and provide experimental validation. In surveillance domain, it is often reasonable to assume that the observers may themselves be a subject of observation for a variety of purposes by unknown adversaries whose model may not be known. Randomizing the observers actions can help to make their target observation strategy less predictable. As the fifth variant, we therefore introduce Adjustable Randomization into the best performing observer strategy where the observer can adjust the expected loss in reward due to randomization depending on the situation.

Introduction

Performing surveillance of targets (Chakrabarty et al. 2002) (Tang and Ozguner 2005) is an important task that security agencies across the world perform on regular basis. The key difficulty here is that there are typically limited security resources for performing the Cooperative Target Observation (referred to as CTO in the literature). The difficulty is exacerbated since targets themselves may have a variety of behaviors. It is therefore important that the security resources are used in an efficient fashion. (Microdrones 2016) presents an application where micro-drones are useful to perform surveillance. In particular, with large gatherings and protests (or even when monitoring a set of individuals or wildlife), it is possible that there would be a number of people (or events) who could possibly be considered suspicious (called targets). Here, the aim of micro-drone would be to perform surveillance of the maximum number of targets (i.e. maintain an overview) instead of doing deep probe of a target i.e. spend all the time on a specific target who may not turn out to be problematic or dangerous.

(Microdrones 2016) also discusses the notion of risk free surveillance. The motors of UAV micro-drones typically allow for a discrete and almost silent surveillance unlike helicopters and planes which are noisy. In addition, micro-drones can use camouflage colors to minimize the risk of being sighted. In this paper, we develop a strategic component to obfuscate the aim of micro-drone wherein the surveillance strategy is inherently randomized with the amount of randomization being adjustable according to the situation. The Cooperative Target Observation problem has been of great interest in both multi-agents (Hu et al. 2013) (Jacobi et al. 2005) and robotics (Rysdyk 2006) (Werger and Matarić 2000) (Parker 1999) literature due to the problem being at the core of many applications. (Parker 2002) introduces the Cooperative Multi-Robot Observation of Multiple Moving Targets (CMOMMT) formalism which is similar to CTO problem. The paper focuses on developing on-line distributed control strategies that allow the robot team to attempt to minimize the total time in which targets escape observation by some robot team member in the area of interest. (Luke et al. 2005) studies the CTO problem and compares the performance of K-means and Hill Climbing algorithms on centralized, partly-decentralized and fully decentralized agent strategies under different levels of target speeds, sensor ranges and update rates. The paper shows that K-means performs quite well over a large part of the parameter space tested and is pretty robust to the degree of decentralization.

(Kolling and Carpin 2007) presents an algorithm for the CMOMMT problem which utilizes information from sensors, communication and a mechanism to predict the minimum time before a robot loses a target. (Jung and Sukhatme 2006) proposes an algorithm that accounts for densities of observers and targets as properties of the environment. By manipulating these densities, a control law for each observer is proposed. In this paper, we build upon the CTO problem formulation presented in (Luke et al. 2005) and adapt it for Surveillance domain which needs us to focus on the following aspects: (a) Model realistic behavior for targets (b) Develop strategy for the observers to perform optimal target observation and (c) Randomize the observers actions to make their target observation strategy less predictable.

Model Description

The CTO problem as stated in (Luke et al. 2005), has a set of mobile agents called observers, which collectively attempt to observe as many targets as possible. Targets are assumed to move randomly in the environment. Observers have a limited sensor range and can observe targets which fall within a circle of radius R centered at each of the observer. The environment is a non-toroidal rectangular continuous 2D field free of obstacles. The paper presents performance comparisons of centralized, partly-decentralized and fully decentralized CTO algorithms for a variety of parameter settings. The decentralized version retains the key characteristics of the centralized CTO problem with the main distinction that each observer takes its decision independently with the help of its local knowledge.

In this paper, we model the surveillance domain as a decentralized CTO problem. Given our focus on surveillance a realistic model for target would be to assume that they are strategic. Furthermore no assumption was made on the targets sensor range earlier since targets were assumed to move randomly and do not consider inputs from environment. We modify this assumption to make targets strategic with a sensor range similar to observers. Targets can therefore identify the presence of observer(s) within their sensor range. In particular, we model targets as intelligent agents, whose aim is to minimize their chances of being observed while observers aim is to maximize their target observation. Prior work on decentralized CTO shows that a K-means based solution (Luke et al. 2005) produces a high quality movement behavior for the observers. We now describe how we build upon the K-means solution to introduce five variant strategies for the observer that consider the strategic behavior of the targets.

Observers Movement

As described in (Luke et al. 2005), a K-means based algorithm can provide a high quality movement behavior for observers. The K-means based algorithm computes a destination point for each observer. The observer then moves towards this destination for γ time-steps or steps (with default value of 10). If the observer reaches its destination before γ steps, it waits until a new destination is computed. There is no communication involved among observers (or targets). Observers do not send any information but are synchronized in decision making due to the waiting time of γ steps(as modeled in (Luke et al. 2005)).

K-means based Algorithm For Decentralized CTO Problem K-means is one of the simplest unsupervised learning algorithm that is used to perform clustering. Clustering is the process of partitioning a group of data points into clusters. It takes N data points and divides them into K clusters. Each data point is assigned to the cluster closest to it. In our context, the set of targets that fall within the range of an observer can be considered as a cluster. Note that clusters need not be of equal sizes. The goal of the algorithm would be to identify a new destination for each observer so that the cluster associated with an observer is more uniformly spread around it. This will improve the possibility of retaining the maximum number of targets over time. Following are key steps of the K-means based algorithm as presented in (Luke et al. 2005):

- Firstly, the algorithm obtains the number of clusters to pick which is same as the number of observers $|O|$ in our domain and then initializes the cluster centers with the observers initial positions.
- For each of the clusters $c1, c2, ..., c|O|$, it identifies the targets whose distance is less than the sensor range of the observer associated with that cluster. Observers observe all the targets that fall into their cluster.
- The algorithm then computes the mean of the targets present in each cluster ci and sets the destination position of the corresponding observer i closer to the mean using the following equation:

$$K_i = (1 - \alpha)K_i + \alpha M_i \quad (1)$$

  where K_i is the initial position of observer i, α is the weighing factor and M_i is the mean position of targets present in the cluster ci.

The algorithm iterates over the above mentioned steps till the time-steps (set as 1500 in our experiments) are exhausted. The destination for each observer is updated every γ steps. Therefore each update step for the observer consists of γ steps and is called update rate of the algorithm. At the start of new update step the algorithm sets a new (next) destination for each observer and allows the observers to act (i.e. move or wait) for γ steps that would help them reach their destination. Hence each observer keeps adjusting its current position that would enable it to retain more targets in its sensor range over time. We now build upon the K-means based algorithm for observer to develop: **(a) Explore-Exploit (b) Memorization (c) One Step Prediction (d) k-step Prediction (e) Explore-Exploit with Adjustable Randomization**

Explore-Exploit In the Explore-Exploit strategy an observer has two specific actions namely Explore and Exploit. An observer can perform an Exploit action using the K-means clustering i.e. Eqn. 1 and can perform an Explore action by setting its destination position at random. The destination of observer is computed using the following formula:

$$K_i = (1 - \alpha)K_i + \alpha(W_{explore} * RP_i + W_{exploit} * M_i)$$
$$W_{explore} + W_{exploit} = 1 \quad (2)$$

Here, K_i denotes the initial observer position, $W_{explore}$ denotes weight factor for Explore component, $W_{exploit}$ denotes weight factor for Exploit component, M_i denotes the mean position of targets present in i^{th} cluster and RP_i denotes a random point. Taking cue from prior work(Luke et al. 2005) on the method to pick a random destination for target, we bound the randomly picked destination point RP_i for the observer to be within a quarter of the environment width and height centered on the observer. The intuition for picking dimensions for the rectangle in this fashion is to avoid observers getting clustered near the center of the environment while moving to their intended destinations. Note that the algorithm still follows the steps of K-means algorithm and hence is iterative i.e. the observer would take γ steps (either random or towards a destination) and then tries

to identify a new destination point. Observer has autonomy to set values for the parameters $W_{explore}$ and $W_{exploit}$. We now present three settings for the Explore-Exploit strategy, generated by changing the weights for Explore and Exploit actions:

- **Model 1 (0-1 model)**: In this model, if the observer is not observing any target currently it performs an Explore action in search of targets i.e. $W_{explore}$ = 1. On encountering one or more target(s), it stops Explore and takes Exploit action of observing the target(s) using K-means clustering algorithm. Hence, if there is no target under observation, $W_{explore}$ = 1, otherwise $W_{explore}$ = 0.
- **Model 2 (0-.5-1 model)**: Model 2 sets $W_{explore}$ to either 0, 1 or 0.5. In case of no target under observation, $W_{explore}$ = 1, if more than two targets are being observed $W_{explore}$ = 0 and if either one or two targets are being observed, $W_{explore}$ = 0.5.
- **Model 3 ($(\frac{1}{|targets_i|+1})^2$ model)**: Similar to the above models, if the observer i is not observing any target it performs an Explore action. Upon encountering target(s), it assigns weight to the Explore component depending upon the number of targets being observed. In particular, the weight for Explore is set as $(\frac{1}{|targets_i|+1})^2$ i.e., as the number of targets being observed increases, the weight for Explore gets smaller and vice versa.

Memorization In Memorization strategy, we improve the 0-1 Explore-Exploit strategy in the following fashion (reason for picking 0-1 Model explained in Experiments): At the start of every update step ($us(i)$: update step i), each observer finds targets in its sensor range. If targets are present it performs an Exploit action as earlier i.e. $W_{exploit}$ = 1. However, the key difference is that, it first sets value for last stored target position ($lstp$) variable. The value for $lstp$ is set as closest target position in the sensor range before starting the Exploit action and then it starts executing the Exploit steps. If at the start of update step ($us(i)$) no targets are present, the observer performs an Explore action by setting its destination as the position in $lstp$ variable (set at $us(i-1)$). The reason is that it hopes that the target may not have gone far from earlier and maybe able to capture it. It then updates the $lstp$ variable for $us(i)$ as $null$. In the next update step ($us(i+1)$) if the observer still does not have a target in range, $lstp$ will be $null$ and hence performs a random explore (true for any step where $lstp$ is $null$). As earlier, random explore sets destination as random point within one quarter of environment width and height centered on it.

One Step Prediction One Step Prediction has a similar structure as Memorization except that it performs a prediction step. The key difference due to this happens in the Explore action wherein instead of setting destination (at $us(i)$) as the last stored target position ($lstp$), if no target is present in observers range, we perform a prediction as where the target would be (at end of us(i)) using the $lstp$. To do this, we use the $lstp$ variable and $ocop$ variable (observers current origin position i.e. before taking first step at $us(i)$). Given that $2 * \gamma$ time-steps would have elapsed by the time the observer reaches its destination set in $us(i)$ (counting from when $lstp$ would be set, which is at $us(i-1)$), we predict the expected distance traveled would be time $(2 * \gamma) *$ target speed (1 unit/time-step) = $2 * \gamma$ units. We assume that target would travel the computed distance on the $ocop$ to $lstp$ line and the expected target position is set as the destination for observer. After reaching the destination, if the observer still does not find any target in range, it sets $lstp$ to $null$. If $lstp$ is $null$ at the start of an Explore action, the target does random explore as in Memorization.

k-step Prediction k-step Prediction has similar structure as One Step Prediction but makes k prior observations of the target instead of one $lstp$ variable. As earlier, at the start of every update step (say $us(i)$), each observer finds targets in its sensor range. If targets are present it performs an Exploit action. The key difference arises in the data collected: The observer collects ID of the nearest target (say target t_j) before taking the first step of Exploit action in $us(i)$. Collecting the ID allows the observer to track the same target afterwards (and avoids mixup of different targets and their locations). Thereafter for each of the next γ steps of $us(i)$, the observer maintains the location of target t_j. If t_j is out of sensor range, it stores $null$ for that step. It then identifies the start and end positions when t_j was in continuous observation, called the $uStart$ and $uEnd$. To predict target position, we assume the target to be moving away from $uStart$ in a straight line connecting $uStart$ to $uEnd$. Given that γ time-steps elapse by the time observer reaches its destination in $us(i)$, we predict the distance traveled by the target from $uEnd$ as time multiplied by speed of target, i.e., $(\gamma - uEnd + \gamma) *$ target speed (1 unit/time-step) = $(2 * \gamma - uEnd)$ units. We set this expected position of target as the observer destination (for $us(i)$). After reaching the destination, if the observer still does not find any target in range, it sets all the location values to $null$. If no prior target information is available or if $uStart$ equals $uEnd$ for $us(i-1)$ at the start of an Explore action, the target does random explore as in Memorization.

Explore-Exploit with Adjustable Randomization One of the key contributions of this paper is to introduce controlled randomization into the observer strategy to make it less predictable. As shown in experiments section, the Explore-Exploit strategy was found to be the best strategy for observer across a variety of settings. Hence we pick Explore-Exploit as the strategy into which we introduce adjustable randomization. Intentionally introducing randomness into agent strategy has been used as a technique in the literature (Bryant and Miikkulainen 2006) to make the agent strategy less predictable. We adapt the BRLP algorithm (Paruchuri et al. 2006) introduced in the context of randomization in MDP based planning to our setting. To adapt this into our solution, we first define the notion of reward for observer in a CTO problem as the mean number of targets being observed by the observer in one simulation run (which as mentioned in our experimental setup has been set as 1500 time-steps).

Randomness in observer strategy will be computed over the weighing factor (i.e. α variable) in Eqn 1. In the ad-

justable randomization strategy, α value is discretized into units of 0.1 varying from 0.1 to 1. A randomized solution would suggest a probability distribution over α vector instead of picking a specific α value that will maximize the observer expected reward i.e. maximize the expected number of targets observed. Randomness in strategy is measured using entropy. However an optimization program involving entropy as the objective function is non-linear in nature and will be intractable. The BRLP algorithm (Paruchuri et al. 2006) introduced in the context of randomization in MDP based planning (BRLP-MDP), is a heuristic developed to address the problem related to non-linearity of the entropy function. We adapt the BRLP-MDP algorithm for our purposes here to develop the BRLP-CTO algorithm.

At each update step of the Explore-Exploit algorithm, the observer needs to compute the optimal α to use. It therefore calls BRLP-CTO algorithm to return the probability distribution over α and one particular value of α is picked based on the distribution returned. A key step involved in BRLP-CTO is the computation of the following Linear Program (3). This LP computes the optimal probability distribution over α i.e. $p(\alpha)$ given a $r(\alpha)$ vector, a template probability distribution vector $p(\bar{\alpha})$ and a value for β as input to the LP. The template probability vector can be any distribution that has a high entropy and is useful to enforce some level of randomness into the solution since the objective function is reward maximization. One such high entropy probability distribution vector is the uniform distribution vector. The amount of randomness that is reflected in the solution from the template vector is controlled by $\beta \in [0, 1]$.

The core issue with adapting BRLP-MDP (Paruchuri et al. 2006) to the CTO problem, is with modeling of $r(\alpha)$. Unlike an MDP where the reward function is input to the problem, we need to perform a real-time simulation over all the anticipated scenarios to construct the reward function. Following are the steps involved in this construction: (1) Each observer $o \in O$ notes the position, speed and direction of all targets $t_o \in T_o$ in its sensor range over the γ time-steps in current update step ($us(i)$). Each target added to T_o may be in sensor range at different time-steps and durations as long as they fall within the window from end of first step in $us(i)$ to the start of first step in $us(i+1)$. (2) Using information from the γ sensing steps, at the start of $us(i+1)$ all the observers $o \in O$ estimate the destination position $dp^o_{t_o}$ for each $t_o \in T_o$ at start of $us(i+2)$. This is called an estimation step and can take place only at the start of update step. (3) Using the estimated values, at the start of $us(i+1)$, every observer o computes the estimated mean position of targets that will lie in its sensor range at $us(i+2)$. Then, every observer o for each α value uses Eqn. 2 (with $W_{explore}$ = 0) to compute its own possible destination position dp^o_α and checks whether each of the targets in $dp^o_{t_o}$ falls within the sensing range of dp^o_α. (4) For each T_o that o predicts will be in sensing range of dp^o_α at $us(i+2)$, it adds $+1$ to its reward at α. Hence the reward vector $r(\alpha)$ for $us(i+1)$ gets estimated.

We now present the LP (3) which takes β value as input (which as we will see the BRLP-CTO provides):

$$\begin{aligned} &\text{Maximize} \sum_{\alpha=0.1}^{1.0} p(\alpha) r(\alpha) \\ &s.t. \sum_{\alpha=0.1}^{1.0} p(\alpha) = 1, \forall p(\alpha) >= \beta p(\bar{\alpha}) \end{aligned} \tag{3}$$

BRLP-CTO identifies the appropriate β value by doing the following: It performs a binary search over the β space to attain a policy with expected reward $E(\beta)$ which lies between $\{\bar{E}, E^*\}$, by adjusting the parameter β. For $\beta = 0$ this problem reduces to a LP without any constraint on $p(\alpha)$ and hence returns the highest expected reward of E^*. For $\beta = 1$ it returns the template policy $p(\bar{\alpha})$ which acts as a lower bound on the expected reward ($\bar{E}$). The binary search exploits this inverse relationship between β and $E(\beta)$ to identify the appropriate β (and hence $p(\alpha)_\beta$) that guarantees the threshold amount of reward E_{min} that a user requests for. BRLP-CTO has fast convergence (average in milliseconds), hence it is suited for dynamic environments.

Algorithm 1 BRLP-CTO(E_{min}, $p(\bar{\alpha})$)

```
Set β_l = 0, β_u = 1 and β = 1/2.
Solve Problem (3), let p(α)_β and E(β) be the optimal solution and expected reward value obtained.
if E_min > Ē then
    while |E(β) − E_min| > ε do
        if E(β) > E_min then
            Set β_l = β
        else
            Set β_u = β
        end if
        β = (β_l + β_u)/2
        Solve Problem (3), let p(α)_β and E(β) be the optimal solution and expected reward value returned
    end while
end if
return p(α)_β
```

Targets Movement

Prior work in the literature assumes that targets move randomly. However in a surveillance domain it is reasonable to assume that targets would be strategic and try to minimize their observation. Targets have a sensor range similar to observers in which they can see observers. Like for observers, we define targets movement by setting a destination point and moving a fixed γ_{target} number of steps (with default value of 100, note that it was γ steps for observers) towards it. If the target reaches its destination position before γ_{target} steps, it immediately calculates its new destination similar to the target movement in (Luke et al. 2005). Targets destination position is computed by using following algorithms: **(a) Randomized Movement (b) Straight-line Movement (c) Controlled Randomization**

Randomized Movement In the randomized approach, targets destination positions are chosen at random from within a local region (one quarter of the environment height and width) centered on the target as targets have a local view of environment. Targets then move to the destination point for utmost γ_{target} time-steps whose default value is set to 100.

Straight-line Movement We propose our first heuristic namely Straight-line Movement wherein a target attempts to escape out of the observers sensor range by following a straight line trajectory in the direction that maximizes the distance of the target from the observer. This direction is computed by joining a line segment between the observer current position and the target in the direction of target. Target takes γ_{target} steps in this direction after which it again calculates the new direction depending on the observer observing it then. If there is more than one observer observing the target, it calculates the mean position of observers and uses it to identify the direction which leads to the maximum distance from the mean.

Controlled Randomization The Straight-line Movement has a downfall that it might increase observers suspicion towards target. To reduce suspicion, we propose Controlled Randomization which adds randomization to the Straight-line Movement. Randomization decreases the suspicion towards a target while Straight-line Movement helps target to escape out of the sensor range of observer. In Controlled Randomization, targets follow the Straight-line Movement for η time-steps and Randomized Movement for the remaining time-steps. We experimented with various values for η and a value of 0.7 was found to perform the best. Hence, the target moves in the direction of destination for .7 * γ_{target} time-steps and follows a random movement henceforth.

Experiments

For purposes of experiments, we assume that the observers and targets are operating in a rectangular field with a width and height of 150 x 150 units (we refer to as units for generalization). We performed a variety of experiments by varying the density, speed and sensor range of agents, where density is a function of number of observers and targets.

- To vary density, the number of observers was picked from $\{2, 6, 10, 14, 18\}$ and the number of targets from $\{3, 9, 15, 21, 27\}$.
- Six possible target speed values were picked among $\{0.2, 0.5, 0.8, 1.0, 1.2, 1.5\}$ measured as units per time-step. The speed of observers was fixed to 1 unit per time-step.
- Five possible sensor range values picked from $\{5, 10, 15, 20, 25\}$ where a value 5 represents 5 units.
- In total there are 5*5 i.e. 25 settings for densities. They were tested against six possible values for target speeds and five different values for sensor ranges resulting in a testbed involving 5*5*6*5 = 750 different settings.
- Each experiment was simulated 30 times (number of runs) and each simulation run consists of a total of 1500 time-steps i.e. each observer and target takes 1500 steps (can stay at same position or move in a step).
- While performing the experiments, for each simulation run we gathered the number of time-steps each target is under observation. The mean across the 30 runs gives the mean number of steps each target is under observation per experiment. If a target was observed by multiple observers, we counted the target only once.

All our experiments were performed on MASON simulation toolkit (Luke et al. 2004). MASON is a fast discrete-event multi-agent simulation library core developed in Java. We implemented our model over MASON and used its internal threading mechanism to run the agents in parallel.

Model	3	9	15	21	27
Model 1	2.3	5.48	7.77	9.72	11.47
Model 2	1.76	4.58	6.94	8.96	10.9
Model 3	2.21	5.28	7.65	9.51	11.41

Table 1: Comparison of Explore-Exploit models

Optimal α value

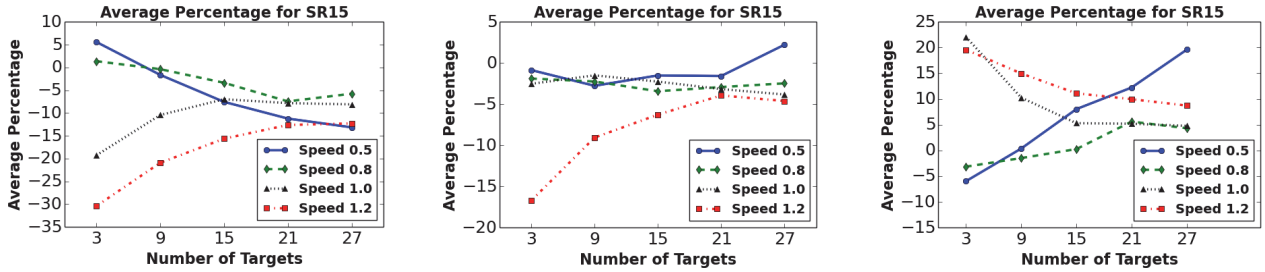
Our first experiment identifies the best α value for observers using K-means algorithm. We set the number of observers as 18, number of targets as $\{3, 9, 15\}$, sensor range as 15, target strategy as Randomized strategy, α discretized using interval of 0.1 from 0.1 to 1 and obtained the following vector of average number of targets observed: <4.39, 4.7, 5.57, 6.52, 6.69, 6.63, 6.66, 6.69, 6.73, 6.82> (corresponding to each value of alpha from 0.1 to 1). Henceforth, we set α to 1 in all our experiments since it performs the best. Prior work on k-means (Luke et al. 2005) used α as 0.25 which we found to be significantly worse for our setup.

Explore-Exploit Model

Our second experiment identifies the best weight vector to use for the Explore and Exploit components in Observer strategy. Table 1 compares the performance measured as the mean number of targets observed per experiment for the three Explore-Exploit models. Each row of the table corresponds to a model and the columns represent the number of targets. For purposes of this experiment we set the number of observers to 10, target speed to 1.0, sensor range to 15 and targets following Randomized strategy. Based on these experiments, Model 0-1 was found to perform consistently better than the rest, as the maximum mean number of targets were observed with the Model 0-1 on an average over 30 runs with 1500 time-steps in each run. Hence, we use weight vectors defined in Model 0-1 for further experimentation.

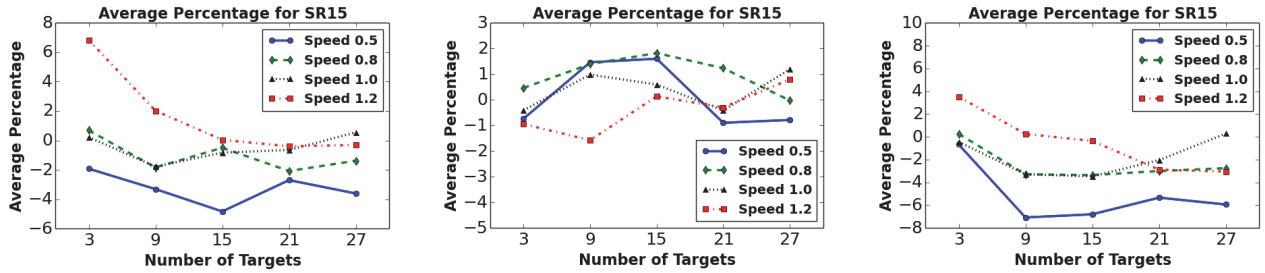
Proportions	3	9	15	21	27
70-30	2.57	5.67	7.88	9.97	11.62
50-50	2.62	6	8.11	10.03	11.93
30-70	2.6	5.83	7.93	10.07	11.69

Table 2: Comparison of Controlled Randomization ratios



(a) Straight-line vs Random (b) Controlled Random vs Random (c) Controlled Random vs Straight-line

Figure 1: Target Strategies Comparison



(a) Memorization vs Explore-Exploit (b) One Step vs Explore-Exploit (c) k-Step vs Explore-Exploit

Figure 2: Observer Strategies Comparison

Mixing Ratio for Controlled Randomization

Our next experiment aims to identify the best ratio of Straight-line and Randomized steps to take for Controlled Randomization. We tested three different mixing ratios: 50-50, 70-30 and 30-70, where 30-70 implies 30% steps in Straight-line and 70% Random. Table 2 illustrates the results measured as the mean number of targets observed. Each row of the table corresponds to one particular mixing ratio while the columns correspond to number of targets. Other parameters were set as follows: number of observers as 10, target speed as 1.0, sensor range as 15 and observers following Explore-Exploit strategy. The table shows that the minimum number of targets were observed when targets use a 70:30 mixing ratio and hence set as default mixing ratio for Controlled Randomization strategy.

Target Strategies Comparison

In this experiment, we study the performance of Straight-line and Controlled Randomization (with 70:30 mix) and compare with the Randomized strategy. Our study shows a significant decrement in the mean number of targets observed due to their ability to avoid the observer. The experiments were performed using a setting of sensor range as 15, number of observers as $\{2, 6, 10, 14, 18\}$, number of targets as $\{3, 9, 15, 21, 27\}$, target speeds as $\{0.5, 0.8, 1.0, 1.2\}$ and observers following Explore-Exploit strategy. The percentage difference between strategies in terms of mean number of targets observed is calculated as $\frac{T_a - T_b}{T_b} \times 100$, where T_a denotes mean number of targets observed by strategy a and T_b denotes mean number of targets observed by strategy b.

Figure 1(a) shows the number of targets on x-axis and depicts the percentage difference in the mean number of targets being observed by targets Straight-line (strategy a) with respect to the Randomized (strategy b) on y-axis. The 4 lines correspond to the 4 different target speed settings indicated. The figure shows a trend wherein as the target speeds get higher from 0.5 to 1.2 there is a marked decrease in the percentage difference which shows that the Straight-line strategy at higher target speeds starts becoming effective.

Figure 1(b) shows the percentage difference in the mean number of targets being observed by targets following Controlled Randomization (strategy a) having 30% randomization with respect to the Randomized (strategy b). While lower on an average than figure 1(a), the figure shows that the percentage of mean targets being observed is entirely in the negative y-axis which shows the Controlled randomization strategy dominates.

Figure 1(c) compares the Controlled Randomization strategy (strategy a) with Straight-line (strategy b). Most points lie on positive y-axis which implies Straight-line strategy is more effective than Controlled Randomization. This is because the random part of the path decreases the distance between observer and target and leads to increase in target observation. While we leave mathematical quantification of suspicion for future work, our goal of introducing randomness is to reduce the suspicion that target is always trying to move away and will be used as default strategy for targets.

Observer Strategies Comparison

In this experiment, we study the performance of Memorization, One Step and k-step Prediction in comparison to

the Explore-Exploit strategy. The experiments were performed using a setting having a sensor range of 15, number of observers as $\{2, 6, 10, 14, 18\}$, number of targets as $\{3, 9, 15, 21, 27\}$ and target speeds as $\{0.5, 0.8, 1.0, 1.2\}$. Targets were modeled to follow Controlled Randomization strategy. The x-axis of each of the subfigures in Fig 2 shows the number of targets while the y-axis shows the percentage difference in the mean number of targets being observed. The four lines correspond to four different target speed settings corresponding to $\{0.5, 0.8, 1.0, 1.2\}$. As earlier, each point in the graph is an average over 30 runs across all the settings for number of observers.

Figure 2(a) compares the Memorization strategy (strategy a) against Explore-Exploit (strategy b). As we can see from the figure for the highest speed setting almost all the points lie on positive y-axis implying Memorization outperforms Explore-Exploit. However, apart from the highest speed setting Explore-Exploit outperforms for all the other settings. The lower performance for Memorization is possibly due to the fact that the observer is only moving to the last position of the nearest target and not searching for other targets.

In Figures 2(b) and 2(c), we compare One Step and k-step Prediction against Explore-Exploit. The figures show no clear winner among Explore-Exploit and One Step but show Explore-Exploit being better than k-step. The reason is possibly similar to Memorization where observer tries to follow the last nearest target in case where no target is in range instead of searching for more targets. The 0-1 Explore-Exploit will therefore be used as the default observer strategy.

Reward vs. Entropy Tradeoff for BRLP-CTO

For this experiment, we vary the percentage of optimal reward (i.e. threshold) that the user would like to obtain using the BRLP-CTO to increase randomization. We performed this experiment with sensor range as 15, target speed as $\{0.8, 1.2\}$, number of observers as $\{2, 10, 18\}$, number of targets as $\{3, 15, 27\}$ and targets following Controlled Randomization. We obtained the following vector of average entropy values (averaged across all the settings for speeds, number of observers, targets and runs): $<$3.321, 3.319, 3.312, 3.26, 3.041, 2.137$>$ for threshold reward ranging from 50% to 100% in increments of 10%. Note that as expected it is a entropy vector with decreasing values with 100% threshold giving the lowest entropy. However, the lowest entropy is still a significant value here since there are many instances where irrespective of α, observer gets same reward and hence the template policy $p(\bar{\alpha})$ gets picked.

Conclusions

We adapted the CTO problem to Surveillance domain. We then improved upon the K-means algorithm to develop five variant strategies for the observer. We changed the assumption related to targets to make them strategic via development of two heuristics. We then performed a series of experiments to show that the 0-1 Explore-Exploit strategy performs best for the observers. We build upon this strategy to introduce user adjustable randomization into the surveillance plan. In future, we plan to study inclusion of realistic constraints in the environment in which the agents operate.

References

Bryant, B. D., and Miikkulainen, R. 2006. Evolving stochastic controller networks for intelligent game agents. In *2006 IEEE International Conference on Evolutionary Computation*, 1007–1014. IEEE.

Chakrabarty, K.; Iyengar, S. S.; Qi, H.; and Cho, E. 2002. Grid coverage for surveillance and target location in distributed sensor networks. *IEEE transactions on computers* 51(12):1448–1453.

Hu, J.; Xie, L.; Lum, K.-Y.; and Xu, J. 2013. Multiagent information fusion and cooperative control in target search. *IEEE Transactions on Control Systems Technology* 21(4):1223–1235.

Jacobi, S.; Madrigal-Mora, C.; León-Soto, E.; and Fischer, K. 2005. Agentsteel: An agent-based online system for the planning and observation of steel production. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 114–119. ACM.

Jung, B., and Sukhatme, G. S. 2006. Cooperative multi-robot target tracking. In *Distributed Autonomous Robotic Systems 7*. Springer. 81–90.

Kolling, A., and Carpin, S. 2007. Cooperative observation of multiple moving targets: an algorithm and its formalization. *The International Journal of Robotics Research* 26(9):935–953.

Luke, S.; Cioffi-Revilla, C.; Panait, L.; and Sullivan, K. 2004. Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 swarmfest workshop*, volume 8, 44.

Luke, S.; Sullivan, K.; Panait, L.; and Balan, G. 2005. Tunably decentralized algorithms for cooperative target observation. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 911–917. ACM.

Microdrones. 2016. https://www.microdrones.com/en/applications/areas-of-application/monitoring/.

Parker, L. E. 1999. Cooperative robotics for multi-target observation. *Intelligent Automation & Soft Computing* 5(1):5–19.

Parker, L. E. 2002. Distributed algorithms for multi-robot observation of multiple moving targets. *Autonomous robots* 12(3):231–255.

Paruchuri, P.; Tambe, M.; Ordóñez, F.; and Kraus, S. 2006. Security in multiagent systems by policy randomization. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 273–280. ACM.

Rysdyk, R. 2006. Unmanned aerial vehicle path following for target observation in wind. *Journal of guidance, control, and dynamics* 29(5):1092–1100.

Tang, Z., and Ozguner, U. 2005. Motion planning for multi-target surveillance with mobile sensor agents. *IEEE Transactions on Robotics* 21(5):898–908.

Werger, B. B., and Matarić, M. J. 2000. Broadcast of local eligibility for multi-target observation. In *Distributed autonomous robotic systems 4*. Springer. 347–356.