MTL Robustness for Path Planning with A*

Robotics Track†

Sarra Alqahtani
Tandy School of Computer Science
University of Tulsa
Tulsa, OK USA
sarra-alqahtani@utulsa.edu

Ian Riley
Tandy School of Computer Science
University of Tulsa
Tulsa, OK USA
ian-riley@utulsa.edu

Samuel Taylor
Tandy School of Computer Science
University of Tulsa
Tulsa, OK USA
smt506@utulsa.edu

Rose Gamble
Tandy School of Computer Science
University of Tulsa
Tulsa, OK USA
gamble@utulsa.edu

Roger Mailler
Tandy School of Computer Science
University of Tulsa
Tulsa, OK USA
roger-mailler @utulsa.edu

ABSTRACT

Maintaining the safety of an autonomous drone while it executes a mission is a primary concern in presence of fixed and mobile enemies. Path planning using A* fails to deliver a feasible, safe plan when a drone has resource limitations in such environments. Enhancing A* with constraint optimization techniques may improve outcomes, but significantly increases path determination time. We define Robust A* (RA*) that introduces the use of a safety margin to maximize the robustness of the drone to meet mission requirements while managing resource restrictions. We rely on a theory of robustness based on Metric Temporal Logic (MTL) as applied to offline verification and online control of hybrid systems. By satisfying the predefined MTL constraints, RA* dynamically defines a safety margin between the drone and an enemy, while constraining the margin size given the drone's resources. The safety margin creates a robust neighborhood around the dynamically generated path. The robust neighborhood holds all valid trajectories within the current world state. When the world state changes, RA* first examines the robust neighborhood to find a valid trajectory before initiating the path re-planning. We evaluate RA* using the Rassim simulator. The results show that the algorithm generates faster and safer paths than the classical A* in the presence of moving enemies.

KEYWORDS

Path planning; metric temporal logic; robustness; A*

ACM Reference format:

S. Alqahtani, I. Riley, S. Taylor, R. Gamble, and R. Mailler. 2018. In Proc. of the 17th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2018), July 10-15, 2018, Stockholm, Sweden, ACM, New York, NY, 9 pages

1 INTRODUCTION

The goal of autonomous drone path planning is to generate risk free paths that account for the geometric characteristics of obstacles. The planning methods commonly used include A*, genetic algorithms, simulated annealing, artificial neural networks, Dijkstra's algorithm, dynamic programming algorithms, particle swarm optimization, ant colony algorithms etc. Many of these approaches handle the autonomous planning problem as a pure path planning problem without considering mission objectives, constraints, and enemy mobility. For a drone to become an autonomous vehicle, the path planning algorithm must provide intelligence to avoid obstacles, threats, and no-fly zones, while satisfying the performance requirements. The autonomous drone must be able to quickly act when enemies move. If the drone relies only on pure path planning, it is probable that the excess time spent to find the shortest path will place it within an enemy's range, which could influence its safety. Without path planning and with only reactive enemy avoidance, the potential for the drone to reach its target location is greatly reduced.

In this paper, we address the problem of an autonomous drone, with resource restrictions, traveling to a target in a dynamic, adversarial environment. The target has a range called *terminal range*. For each problem instance, the goal is for the drone to be located eventually within terminal range of its assigned target. The environment is non-deterministic, such that the drone has partial information about enemies prior the mission, including their geometry and capabilities, but the distribution of enemies is unknown until the drone traverses the world and uses its sensors for detection. The drone can perceive the environment around it within a circular region centered at the drone location where the radius of the circle is its vision range. The environment under investigation has three types of static and mobile enemies with different risk ranges:

1. **Radars**: able to see and detect the drone from a distance with what we call *vision range* (VR).
2. **Jammers**: able to jam the drone's communications if the drone gets in its *jamming range* (JR).
3. **Killers**: able to shoot missiles when the drone gets inside its *weapon range* (WR).

The A* algorithm is perhaps the most popular path planning search algorithm using both a heuristic function and a cost

function to reach the target, i.e., $f(n) = h(n) + g(n)$. The algorithm finds the cheapest path by trying (expanding) the node with the lowest f [1]. For real-time planning, where computational speed is a priority, the D* algorithm can perform fast rerouting when new obstacles are detected in the environment [2]. D* is substantially faster than A*, but at the cost of sub-optimal solution paths [3].

Both A* and D* consider path length as the only optimality criterion. Additional optimality criteria introduce the problem of multi-objective path planning (MOPP). As a solution to such problems, the Multi-objective A* (MOA*) Algorithm [4] and MOD* [5] were developed, based on similar principles to A* and D*, respectively. In the presence of safety risks imposed by moving enemies, the aforementioned algorithms cannot guarantee they will find a feasible path. The shortest path may not be the safest, and could be the most hazardous path. The safest path may be the longest one, but requires consideration of limited resources. Therefore, an algorithm incorporating path feasibility into risk-minimization is needed.

In this paper, we develop a robustness function using the robustness theory of Metric Temporal Logic (MTL) [6] to maximize the drone safety while satisfying mission constraints. The MTL robustness can be defined as the upper-bounded perturbation that the drone can tolerate without changing its Boolean truth value with respect to its mission specification expressed in MTL [6]. In detail, if an MTL specification φ valuates to positive robustness ε, then the specification is true, i.e., satisfied and, moreover, the path points can tolerate perturbations up to ε and still satisfy the specification. Similarly, if ε is negative, then the path point does not satisfy φ and all the other points that remain within the open tube of radius $|\varepsilon|$ also do not satisfy φ.

Our approach to address the reach-while-avoid-when-possible problem has two main steps. First, the mission constraints are simply and concisely expressed using MTL specifications [10]. Secondly, we create RA* by extending A* with two modifications: (i) a soft modification of the objective function to include the MTL robustness function, and (ii) a hard modification of the algorithm logic to exclude the non-robust positions from the search space. The robustness function is used to guide the node expansion in RA* and dynamically create a safety margin around adversarial assets using the drone resources. In addition, RA* creates a robust neighborhood around the generated path using the robustness degrees of the path points [6, 11]. The robust neighborhood provides a set of valid trajectories that can be robust enough for the drone to autonomously react to moving enemies or fuel leak without conducting any re-planning process.

The next section further discusses the related work of the MTL robustness and its application to path planning problems. Section 3 discusses the proposed approach, and Section 4 presents the results and shows examples.

2 RELATED WORK

Researchers have applied genetic algorithms to MOPP problems [12, 13], but these have a major issue of computational complexity. Combining the optimization criteria into a single objective function is a common approach [14, 15] often with tools, such as penalty functions and weights for linear combinations of attribute values. These methods are problematic as the generated paths are very sensitive to small adjustments in the penalty function coefficients and weighting factors [16].

The Limited damage A* algorithm [15] considers UAV damage and distance criteria. An upper bound is predefined for maximum tolerated damage. Two heuristic functions are computed, one for the distance and one for the damage, each of which is multiplied by a factor that depends on the environmental parameters that must be examined experimentally to find their optimal values. The algorithm finds suboptimal solutions with a reasonable time performance compared to MOA*.

The path planning algorithm in [16] uses A* with a key modification. Rather than computing the cost function f by summing cost criteria, it calculates the Pareto front of the cost criteria. However, the algorithm deals with fixed obstacles as impassable terrains, which affects the flexibility of the generated paths when resources are limited.

Considering the issues of using predefined weights for path planning algorithms, there is a need for more robust multi-objective optimization functions. Recent papers [8, 9] develop path planning algorithms using the MTL robustness theory. Unlike the problem domain explained in this paper, the problem domains in [8, 9] only contain of static obstacles that the robot or drone must avoid in order for the mission to succeed using the specification of reach-while-avoid problem.

In [8], a path planner framework is designed to produce global paths using a linear temporal logic (LTL) specification and local trajectories using an MTL specification. The global path is only updated if new obstacles appear, while the MTL specification is updated if new obstacles appear and if mission requirements change. It is unclear how they handle a change to mission requirements that relax risk constraints. In addition, incorporating mobile enemies would require the global planner to recalculate a global path every time a known obstacle moves.

In [9], a reach-while avoid specification using MTL was defined and used by a robustness estimation function to find the most robust path. Their mixed-integer linear programming (MILP) algorithm calculates the trajectory that minimizes the performance objective function while satisfying the robustness measure. It does not support moving enemies or risk awareness.

To maximize the robustness of a MTL specification, authors in [7] apply a gradient descent method using Sequential Quadratic Programming (SQP). Their results show that the proposed optimization technique is significantly faster and the functional output is arguably much better than the output of MILP algorithm [9]. Although the investigated environment is fully known with only one static obstacle, the performance of the developed technique is not time efficient for online path planning problems. To our knowledge, our work is the only work that utilizes the MTL robustness with the A* algorithm for online path planning in non-deterministic environments with mobile enemies.

3 MTL ROBUSTNESS MAXIMIZATION FOR PATH PLANNING

The RA* generates the robust neighborhood and Robust_Heuristic_Search (RHS) to adjust the path when new information is detected. Figure 1 displays a closed-loop process that ends when the drone reaches the target. The RA* creates a safety margin around the closest enemy to set the minimum accepted risk at each path step. The safety margin size is computed using the robustness function to maximize the satisfaction of the mission constraints along the planned path. RA* keeps track of a robust neighborhood around the optimal trajectory to reduce re-planning attempts and increase drone resilience against moving enemies. The robustness measure returns positive values if the trajectory satisfies the specification and negative values otherwise. Intuitively, the robustness degree of a feasible path is the largest distance the drone can independently perturb and still maintain the feasibility of its current path. This defines a neighborhood around the original path such that any trajectory within this neighborhood is guaranteed to satisfy the specification but with a lower degree of robustness. When the drone detects a violation of its constraints, the RHS utilizes the current robust neighborhood to find a valid replaceable trajectory. When the whole neighborhood becomes invalid, re-planning needs to be executed by RA*. The objective is to find a neighborhood with a set of valid trajectories and an optimal path at the neighborhood's center instead of planning for a single path. However, when the algorithm can only find one feasible path then the robust neighborhood is collapsed into a single trajectory. A collapsed neighborhood occurs when the available resources approach their limits and, consequently, the drone must take some allowable risk to reach its target.

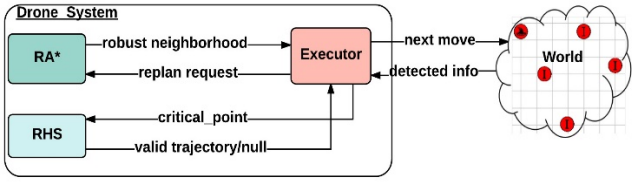

Figure 1: The proposed approach

Definition 1 (MTL Syntax). Let AP be the set of atomic propositions and I be a time interval of R. The MTL φ formula is recursively defined using the following grammar [10]:

$$\varphi := T|p|\neg\varphi|\ \varphi_1 \vee \varphi_2|\varphi_1 \wedge \varphi_2|\ \varphi_1 \mathcal{U}_I \varphi_2 \quad (1)$$

T is the Boolean True, p ∈ AP, ¬ is the Boolean negation, ∨ and ∧ are the logical OR and AND operators, respectively. $\mathcal{U}_I$ is the *timed until* operator and the interval I imposes timing constraints on the operator. Informally, $\varphi_1 \mathcal{U}_I \varphi_2$ means that φ_1 must hold until φ_2 holds, which must happen within the interval I. The implication (⟹), Always (□), and Eventually (◊) operators can be derived using the above operators.

Using the MTL syntax (**Definition 1**), we define the MTL specification of our problem of reach-while-avoid-when-possible as follows.

$$\varphi = \Diamond_{[0,timeConstraint]}\, q \wedge \Box_{[0,timeConstraint]}\,(\neg unsafe\,) \wedge \Box_{[0,timeConstraint]}\big((\, constrain_1 > threshold_1 \wedge \ldots \wedge constrain_n > threshold_n\,) \Longrightarrow \neg semiSafe\big) \quad (2)$$

This formula requires the drone to reach the target terminal q (i.e., liveness property) while always avoiding being inside *unsafe* areas (i.e., safety property). When the available $constraint_{1..}$ $constraint_n$ are above their predefined thresholds, it must always stay away from the semi-safe areas (i.e., conditional safety property). Otherwise, the semi-safe areas need to be gradually receded to free up some space for the drone to maneuver in to reach its target. We define a path trajectory that satisfies the specification given in (2) to be a feasible trajectory. Otherwise, it is infeasible. For our problem domain, this specification would be:

$$\varphi = \Diamond_{[0,deadline]}\, q \wedge \Box_{[0,deadline]}\,(\neg WR \wedge \Box_{[0,deadline]}\big((\, f > f_{min} \wedge\ t < t_{max}\,) \Longrightarrow \neg VR \wedge \neg JR\big) \quad (3)$$

To precisely capture the MTL formula, each predicate $p \in AP$ is mapped to a subset of the metric space S. Let $\mathcal{O}: AP \longrightarrow \mathcal{P}(S)$ be an observation map for the atomic propositions. The Boolean truth value of a formula φ with respect to the trajectory s at time t is defined recursively using the MTL semantics directly reproduced as stated in [10]:

$$(s,t) \coloneqq T \iff T$$

$$\forall\, p \in AP, (s,t) \coloneqq \mathcal{O}\, p \iff s_t \in \mathcal{O}(p)$$

$$(s,t) \coloneqq \mathcal{O}\, \neg\varphi \iff \neg(s,t) \coloneqq \mathcal{O}\, \varphi$$

$$(s,t) \coloneqq \mathcal{O}\, \varphi_1 \vee \varphi_2 \iff (s,t) \coloneqq \mathcal{O}\, \varphi_1 \vee (s,t) \coloneqq \mathcal{O}\, \varphi_2$$

$$(s,t) \coloneqq \mathcal{O}\, \varphi_1 \wedge \varphi_2 \iff (s,t) \coloneqq \mathcal{O}\, \varphi_1 \wedge (s,t) \coloneqq \mathcal{O}\, \varphi_2$$

$$(s,t) \coloneqq \mathcal{O}\, \varphi_1 \mathcal{U}_I \varphi_2 \iff \exists t' \in t + I.\, (s,t') \coloneqq \mathcal{O}\, \varphi_2 \wedge \forall\, t'' \in (t,t'), (s,t'') \coloneqq \mathcal{O}\, \varphi_1$$

In our problem domain, we have 6 atomic propositions including time, fuel, risk ranges (VR, JR, and WR), and the target. To properly use the observation map semantics in the problem domain, we compute time and fuel in terms of distance metric d, which is the Euclidian distance between points p_1, p_2, as:

$$d(p_1,p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (4)$$

By using the drone's velocity v, and the fuel consumption rate crt (fuel per distance), we calculate the time and fuel in terms of distance as follows:

$$t = \frac{d}{v} \quad (5)$$

$$f = d \times crt \quad (6)$$

To formally measure the robustness degree of φ (2) at the trajectory point s at time t, the robustness semantics of φ is recursively defined as taken directly from [6]:

$$[\![T]\!](s,t) \coloneqq +\infty$$

$$[\![p]\!](s,t) \coloneqq Dist_d(s(t), \mathcal{O}(p))$$

$$[\![\neg\varphi]\!](s,t) \coloneqq \neg[\![\varphi]\!](s,t)$$

$$[\![\varphi_1 \vee \varphi_2]\!](s,t) \coloneqq [\![\varphi_1]\!](s,t) \sqcup [\![\varphi_2]\!](s,t)$$

$$[\![\varphi_1 \wedge \varphi_2]\!](s,t) \coloneqq [\![\varphi_1]\!](s,t) \sqcap [\![\varphi_2]\!](s,t)$$

$$[\![\varphi_1 \mathcal{U}_{[l,u]} \varphi_2]\!](s,t) \coloneqq \bigsqcup_{j=t+1}^{t+u} \Big([\![\varphi_2]\!](s,j) \sqcap \bigsqcap_{k=t}^{t-1} [\![\varphi_1]\!](s,k)\Big)$$

where ⊔ stands for maximum, ⊓ stands for minimum, $p \in AP$, and $l, u \in \mathbb{N}$. The robustness is a real-valued function of the trajectory point s with the following important property stated in Theorem 1.

Theorem 1 [6]: For any $s \in S$ and MTL formula φ, if $[\![\varphi]\!](s,i)$ is negative, then s does not satisfy the specification φ at time *i*. If it is positive, then s satisfies φ at *i*. If the result is zero, then the satisfaction is undefined.

By maximizing the robustness degree $[\![\varphi]\!]$, we can compute the control inputs (direction, velocity) over the finite set of input sequences which would provide us with a path solution to a given problem instance, assuming that there is at least one feasible path. The generated sequence of inputs can be simply considered as the sequence of path points, i.e., trajectory (s,t) to the target that satisfy φ by having positive robustness degree $[\![(s,t)]\!] > 0$. The larger $[\![(s,t)]\!]$, the more robust is the trajectory to a perturbation of φ. In other words, trajectory s can be disturbed at time t while $[\![\varphi]\!]$ decreases but remains positive. Consequently, the robustness degree $[\![\varphi]\!]$ of each path trajectory creates a robust neighborhood around the trajectory to make available a set of trajectories with less $[\![\varphi]\!]$ but still satisfy the original φ.

The Signed Distance, $Dist_d$, is a domain-specific function that must be defined to reflect the domain properties [6]. In this paper, we define three functions to measure the distance from the propositions of the target, the enemy set, and the resource limits (Figure 2). The target symbol represents the target terminal while the blue circle is the drone. The red circle surrounded by multiple circles represents the enemy, where cyan, orange, and red circles are the VR, JR, and WR, respectively.

Definition 2 (Target $dist$ function): Given that q is the target terminal of drone p and r is its range, the $dist$ between q and p at time *i* is defined as

$$dist(p_i, q) = d(p_i, q) - r \tag{7}$$

Definition 3 (Enemy $dist$ function): Let p be the drone, X be the enemy set, and $\theta \geq 0$ be the enemy speed. Then $dist$ between p and X at time i is defined for each risk range as follows:

$$dist_{VR}(p_i, X) = \min_{0 \leq j \leq |X|} \mathrm{d}(p_i, x_j) - (x_j.VR + \theta) \tag{8}$$

$$dist_{JR}(p_i, X) = \min_{0 \leq j \leq |X|} \mathrm{d}(p_i, x_j) - (x_j.JR + \theta) \tag{9}$$

$$dist_{WR}(p_i, X) = \min_{0 \leq j \leq |X|} \mathrm{d}(p_i, x_j) - (x_j.WR + \theta) \tag{10}$$

With respect to the target q, $dist$ is defined as the distance from the drone to the closest edge of the region defined by the target's terminal range. On the other hand, the enemy $dist$ function is evaluated with respect to the drone p and the set of enemies X to a triple that represents the distances to the range of the closest enemy x to p (Figure 2). In addition, we use the velocity of a given enemy θ such that each risk range is extended with the enemy's velocity as represented by dotted circles around colored enemy ranges (Figure 2). The terminal range for a moving target must be shrunk, rather than extended, by the velocity of the target, assuming that the target is running away from the drone.

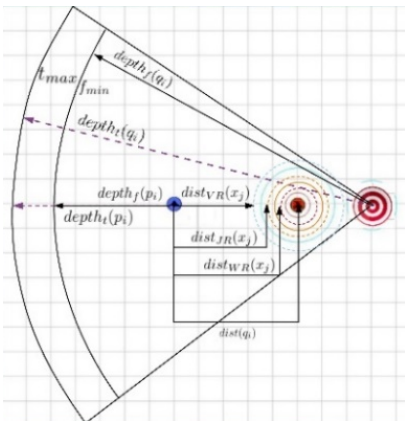

Figure 2: The structure of the problem domain

Lastly, we define a depth function to measure the distance between the current position of the drone and its resource limit. Each drone has a pre-specified amount of fuel and time to reach its target. We assume that each mission starts at time 0 with fuel f_{max} and each drone has time t_{max}, the deadline, to reach its target. To calculate the robustness of a given configuration of drones with allotted fuel capacities, fuel consumption rates, and deadlines, we redefine each constraint in terms of distance using equations 5 and 6. Given that a drone moves with velocity v, we define two regions centered at the target q with radius $v \times f_{min}$ and $v \times t_{max}$ to define the farthest positions that the drone could travel while still being able to reach its target. With these regions defined, we can define the function depth.

Definition 4 (Resource $depth$ function): Given that f_{min}, t_{max} are the resource thresholds, $depth_f$ and $depth_t$ functions for the drone p at time *i* are defined as:

$$depth_f(p_i) = \begin{cases} (f_i - f_{min}) - (d(p_i, q) - r) \times \mathrm{crt} & if\ p_i \notin \mathcal{O}(f_{min}) \\ 0 & otherwise \end{cases} \tag{11}$$

$$depth_t(p_i) = \begin{cases} (t_{max} - t_i) - \frac{(d(p_i,q)-r)}{v} & if\ p_i \notin \mathcal{O}(t_{max}) \\ 0 & otherwise \end{cases} \tag{12}$$

The function depth measures the distance to the closest edge of the region defined by a constraint centered on the target. It should be noted that the defined regions are 3D with respect to time. Therefore, the pizza slice shown in Figure 2 would be shrunk from the outer edge over time.

Using the $dist$ and $depth$ functions, the MTL robustness degree of φ in equation 3 can be point-wise computed at each position in the world state to solve the following path planning problem:

$$\rho(s_0, q) = min \sum c(s_i, s_{i+1}) - [\![(s_i, s_{i+1})]\!] \tag{13-1}$$

$$s.t. \quad 0 \leq i < t_{max} \tag{13-2}$$

$$f_{min} \leq f \leq f_{max} \tag{13-3}$$

$$\epsilon_{min} \leq [\![\rho]\!] \leq \epsilon_{max} \tag{13-4}$$

$$\begin{aligned}\square_{[0,deadline]}\Big(\neg WR_X(pos)\Big) &= \neg\Big(T\,\mathcal{U}_{[0,deadline]}\,\neg WR_X(pos)\Big)\\ &\overset{[1,1]}{\Longrightarrow} = \neg\Big(\bigsqcup_{j=i+1}^{i+1}\Big([\![T]\!](pos,i)\sqcap\prod_{k=1}^{i}\neg[\![WR_X]\!](pos,k)\Big)\Big)\\ &= min\Big(\infty, dist_{WR}(pos,X)\Big) = dist_{WR}(pos,X)\end{aligned} \tag{14}$$

$$\begin{aligned}\square_{[0,deadline]}\Big(\big(f(pos)>f_{min}\wedge t(pos)<t_{max}\big)\implies\big(\neg VR_X(pos)\wedge\neg JR_X(pos)\big)\Big)\\ = \neg\Big(T\,\mathcal{U}_{[0,deadline]}\Big(\big(f(pos)>f_{min}\wedge t(pos)<t_{max}\big)\wedge\big(\neg VR_X(pos)\vee\neg JR_X(pos)\big)\Big)\Big)\\ \overset{[1,1]}{\Longrightarrow} = \Big(\prod_{j=i+1}^{i+1}\Big([\![F]\!](pos,i)\sqcup\bigsqcup_{k=1}^{i}\big(\neg[\![f]\!](pos,k)\sqcup\neg[\![t]\!](pos,k)\big)\sqcup\big([\![VR_X]\!](pos,k)\sqcap[\![JR_x]\!](pos,k)\big)\Big)\Big)\\ = max\Big(\neg depth_f(pos),\neg depth_t(pos),min\big(dist_{VR}(pos,X),dist_{JR}(pos,X)\big)\Big)\end{aligned} \tag{15}$$

where $c(s_0, s_i)$ is a cost function similar to the f function in A*. Equations 13-2 and 13-3 represent the resource limitation. ϵ_{min} and ϵ_{max} are the desired minimum and maximum robustness which make equation 13-4 an optional constraint in the problem. When $\epsilon_{min} > 0$, it enforces a minimum safety margin around enemies, while $\epsilon_{max} > 0$ attempts to limit the path length when the available resources are extremely large.

3.1 RA* Algorithm

A* represents the world state as a grid map divided into squares. In this paper, each square is evaluated either as a passable (safe and reachable), impassable when it is either occupied by one or more enemies, or unreachable when the drone does not have enough resources to reach it. However, the impassable squares that are occupied by the VR or JR of enemies can become passable when the drone cannot afford to avoid them given its limited resources.

RA* uses the same logic as the standard A*. The modified functions are presented in Algorithm 1. In line 3 of the *neighbors* function, the node is included into the search space only when the robustness is zero or positive. Our algorithm uses the MTL robustness to classify nodes in the gird into passable or impassible based on their satisfaction of the mission safety constraints. Only passable nodes are used to feed the open queue in RA*. The robustness of the liveness property is used in the h function to encourage the algorithm to expand its search towards positions closer to the target.

Assumption 1: If drone velocity is v and the max velocity of the enemy set is $v' = max_{x\in X} v_x$, then the drone is assumed to be faster $v > v'$ by at least 20%.

Assumption 2: If the drone vision range is VR and the enemy set's max vision range is $VR' = max_{x\in X} VR_x$, then the drone has bigger vision range $VR > VR'$.

Assumption 3: Given target q and enemy set X, q is not part of the enemy set $q \notin X$.

The robustness of the safety property ($\square_{[0,deadline]}(\neg \mathrm{WR})$) is measured at each position of the search space since it must hold at all path points. To measure the robustness of the safety property for *pos*, we use the MTL robustness semantic with duration of [1, 1] and enemy set X is found in equation 14. In order to apply the robustness semantic, the always operator $\square$ is converted into the *Until* operator using the conversion rules in [11].Then, the robustness becomes a minimum function of the robustness of True value and the *dist* function in equation 10. Since the robustness of True by semantic is positive infinity, the robustness function becomes about the *dist* function of WR.

Line 4 in robustness_safety in Algorithm 1 defines the dist function using equation 10 to measure the distance from the WR of the enemy set. The positions inside the WR would have negative robustness, preventing the drone from taking paths through them. Positions with positive numbers are considered passable. Their robustness degree depends on how far they are from the boundary of the WR.

The conditional safety property $\Big(\square_{[0,deadline]}\big((f > f_{min}\wedge t < t_{max})\implies \neg VR\wedge\neg JR\big)\Big)$ evaluates the ability of the drone to avoid being seen or jammed by enemies considering its current time and fuel. It avoids enemy JR and VR only when it has sufficient resources to do so, otherwise these areas are included in the search space as passable positions for the drone. The $depth$ functions, in equations 11 and 12, measure how far away the drone is from being out of time or fuel if it chooses to pass through position *pos*. The safety margin, i.e., the minimum robustness ϵ_{min}, is dynamically computed using the results of $depth_f$ and $depth_t$ (line 1 in *robustness_safety)*. The safety margin is decreased gradually with the time and fuel and becomes negative when either one of the resources starts approaching its limits. In lines 2 and 3 of function *robustness_safety*, the updated safety margin would be subtracted from the VR and JR to allow the drone pass through semi-safe areas. The *dist* functions in lines 2 and 3 return positive if the drone obeys the constraint of the minimum robustness and returns negative otherwise. The robustness of the conditional safety is computed in equation 15 using the MTL robustness semantics for the time duration of [1, 1] and enemy set *X*. The

robustness function becomes about finding the maximum values of the negative of $depth_f$ and $depth_t$ and the minimum of dist functions of equations 8 and 9.

The MTL robustness semantic in equation 15 is mapped into line 5 of *robustness_safety* function to decide if position *pos* is passable or impassable. Equation 15 would return negative values if the point *pos* is inside the jamming and vision ranges (JR, VR) of the closest enemy and the drone's time and fuel are above thresholds. On the other hand, it would return positive if and only if *pos* is outside the VR and JR of all enemies. In case the fuel or time are approaching their limits, equation 15 would always return zero when *pos* is inside VR or JR of enemies. According to Theorem 1, zero is undefined robustness, which RA* would accept only when there are insufficient resources to reach the target while avoiding areas with conditional safety property. By using this technique, the MTL robustness allows runtime risk assessment of the VR and JR which completely depends on the availability of resources.

Line 6 in *robustness_safety* computes the final robustness degree of *pos* as the min value of the safety and conditional safety properties. A negative value means the robustness is negative and *pos* is not inserted into the search space (line 3 of the *neighbors* function). Otherwise, the robustness degree is either positive or zero and *pos* is passable and considered for path planning search. By preferring paths with larger safety robustness in line 5 of the *neighbors* function, the algorithm generates robust paths to risky areas.

In RA^*, $\Diamond_{[0,deadline]}\, q$ evaluates the reachability of the target q from position *pos* in the *robustness_liveness* function. It depends on the *dist* function in equation 7, which returns a positive real number if *pos* is outside the terminal range of q and returns negative otherwise. Then, h in line 6 of *neighbors* function would have negative value when *pos* is inside the target area which decreases f in line 7. To guarantee that the algorithm expands positions with shorter distances to the target, the OPEN queue of A* is ordered based on the lowest (most robust) f values of searchable positions.

Algorithm 1 Functions needed for RA^* Search

function neighbors(p, ϵ_{min}, $\epsilon_{\max}$)
1- neighbors = neighbors_of(p,1)
2- **for** n ∈ neighbors
3- **if**(robustness_safety(n, X, ϵ_{min}, $\epsilon_{\max}$)>=0)
4- n.r=robustness_safety(n, X, ϵ_{min}, $\epsilon_{\max}$)
5- n.g = d(n,p)- n.r
6- n.h= robustness_liveness(neighbor, q)
7- n.f = n.g + n.h
8- n.parent = p
9- **return** neighbors
function robustness_safety(pos, X, ϵ_{min}, $\epsilon_{\max}$)
1- $\epsilon_{min} = \min(\epsilon_{min}, depth_f(pos) - \epsilon_{min}, depth_t(pos) - \epsilon_{min})$
2- $dist_{VR}(pos, X) = dist_{VR}(pos, X) - \epsilon_{min}$
3- $dist_{JR}(pos, X) = dist_{JR}(pos, X) - \epsilon_{min}$
4- safety $= dist_{WR}(pos, X)$
5-con_safety=
$$\max\begin{pmatrix} depth_f(pos), depth_t(pos), \\ \min\left(dist_{VR}(pos, X), dist_{JR}(pos, X)\right) \end{pmatrix}$$
6-**return** min (safety,con_safety, ϵ_{max})
function robustness_liveness(pos, q)
1- **return** $dist(pos, q)$

3.2 RHS

The loop function (Algorithm 2) of the drone (agent) moves the drone on the path and monitors robustness simultaneously. The drone's sensor detects enemies when they are within its vision. The *violate_robustness* function (line 9 of loop) checks if the robustness of the current path is violated by newly detected information. It iterates over the current path points, re-evaluates the robustness of the safety property given available enemy information and the world state, and returns the first invalid path point, i.e., critical point *cp*. A violation calls RHS (line 12) to find a replaceable trajectory inside the robust neighborhood. To make a quick decision about the validation of the robust neighborhood, *RHS* checks the validity of the neighbors of *cp* at the edges of the neighborhood using the *cp*'s robustness degree to identify these edges (lines 1-6 in RHS). The RHS is a pure heuristic search for finding a valid trajectory inside the robust neighborhood with a min robustness of zero. Essentially, the heuristic search concentrates on quickly and effectively finding valid trajectory as an immediate reaction against moving enemies. However, when there is no valid trajectory inside the robust neighborhood, RA* is recalled, generating a new path with its robust neighborhood considering all enemies that can be seen (line 14).

Algorithm 2 Functions needed for MTL-Robustness Monitoring

function loop()
1- X =sensor_results
2- **If** ρ=∅
3- ρ ←path generated by MTL_Robustness_Based_A*
4- **If** ρ=∅
5- Agent.Stop() // no path
6- **Else**
7- **If** sensor_results != ∅
8- X= X ∪ sensor_results
9- cp=violate_robustness(ρ,X,0, $\epsilon_{\max}$)
10- **If** cp≠null
11- s_0= drone_position
12- ρ= RHS(s_0,cp, ρ, 0, $\epsilon_{\max}$)
13- **If** ρ=null
14- ρ ←re-plan by RA*
15- Agent.move(ρ)
function violate_robustness(ρ,X, $\epsilon_{\min}$, $\epsilon_{\max}$)
1- **for** pos ∈ ρ
2- **if** robustness_safety(pos, X,0, $\epsilon_{\max}$)<0
3- **return** pos
4- **return** null

function RHS(s_0,cp, ρ, $\epsilon_{\min}$, $\epsilon_{\max}$)
1- neighbors = neighbors_of(cp, cp.r)
2- **for** each pos ∈ neighbors
3- **if** robustness_safety(pos, X,0, $\epsilon_{\max}$)<0
4- neighbors.remove(pos)
5- **if** neighbors=null
6- **return** null
7- p= s_0
8- open←p
9- **while** *open* ≠ ∅
10- p= pop(open)
11- **if**(dist(p,q)==r)
12- return construct_path(p)
13- closed←p
14- **for** n ∈ neighbors(p,0, $\epsilon_{\max}$)
15- **if** n ∉ open
16- n.h= robustness_liveness(n, q)-n.r
16- open←(n)
17- **return** null

By using MTL robustness to plan and monitor paths, the generated paths satisfy the initial mission constraints under all circumstances. However, when the drone has smaller vision than the enemies, it might find itself inside a vision of one or more enemies before it sees them. This case can be dealt with as a special case by considering these areas temporarily passable allowing the drone to escape the immediate risk towards the target. This obviously violates the safety property but that is because of the physical capabilities of the drone and not related to the path planning and monitoring processes.

Figure 4 illustrates the approach using the Rassim simulator where the drone has unlimited time and fuel to reach its target (deadline=∞, fuel=∞, v =100, $\epsilon_{\min} = v$, $\epsilon_{\max} = v$):

$$\varphi = \Diamond_{[0,\infty]}\, q \wedge \Box_{[0,\infty]}\,(\neg \mathrm{WR}\,) \wedge \Box_{[0,\infty]}\big((\infty > f_{min} \wedge\ t < \infty) \Longrightarrow \neg VR\ \wedge \neg JR\big)$$

Pink represents the planned path while green represents the path traversed by the drone to reach its current position. The robust neighborhood is shown in blue around the path. At time t_0, the drone detects enemy E_1 and updates its specification to become: $X = E_1$. RA^* finds a path that avoids enemies in X. At t_1, it sees E_2, which is a mobile enemy, and X becomes $\{E_1, E_2\}$. Since E_2 invalidates the whole robust neighborhood, RA^* is recalled to find another path with a new robust neighborhood. The drone sees E_3 and X becomes $\{E_1, E_2, E_3\}$, but part of the neighborhood is still valid. Here, *RHS* finds a valid trajectory to the target without doing replanning.

The case of limited resources is shown in Figure 5. Here, the mission specification is (deadline=$25ms$, fuel=$50g$, crt=0.1 g/ms, $f_{min} = 5g$, v =100, $\epsilon_{\min} = v$, $\epsilon_{\max} = v$):

$$\varphi = \Diamond_{[0,25]}\, q \wedge \Box_{[0,25]}\,(\neg \mathrm{WR}\,) \wedge \Box_{[0,25]}\big((f > 5 \wedge\ t < 25\,) \Longrightarrow \neg VR\ \wedge \neg JR\big)$$

Obviously, the drone cannot reach its target without accepting some risk. At t_0, the drone accepts some risk of VR of enemy E_1. At t_1, it tries to avoid E_2, but it continues moving toward its path. Then, the drone evaluates the robustness of its current path. Since the deadline at t_1 is approaching, the path becomes robust despite the moving enemy and any further detected enemies. Therefore, the drone stops attempting to avoid E_2 and takes the direct path to the target without replanning when it saw enemy E_4.

4 EVALUATION

The main motivation of this paper is to increase the possibility of the mission completion in adversarial world when the drone does not have sufficient resources to completely avoid all hazardous areas in presence of mobile enemies. Enhancing A* with MTL robustness helps to find a balance between risk avoidance and resource depletion. To show the effectiveness of the approach, we compare the average of planning time, travel time (i.e., path length), and the time spent inside risky areas of the A* algorithm and RA* in 100 randomly generated scenarios with static and moving enemies. We have built a random scenario generator on top of Rassim to test our algorithm. The worlds are setup as 3000 × 1500 cell grids. The scenario generator randomly places the target, drone, and enemy assets using a normal distribution with multiple values of the standard deviation and mean using the world width and height. Killers are selected with a 60% probability, Jammers with a 20% probability, and Radars with a 20% probability. We initially set the min and max accepted robustness, $\epsilon_{\min}$ and $\epsilon_{\max}$, to the drone velocity. The experiments are conducted on a quad-core Intel i7 3.4GHz processor with 16GB RAM. We run the same scenarios with unlimited, limited (100 seconds), insufficient (50 seconds) time. The results are shown in Table 1.

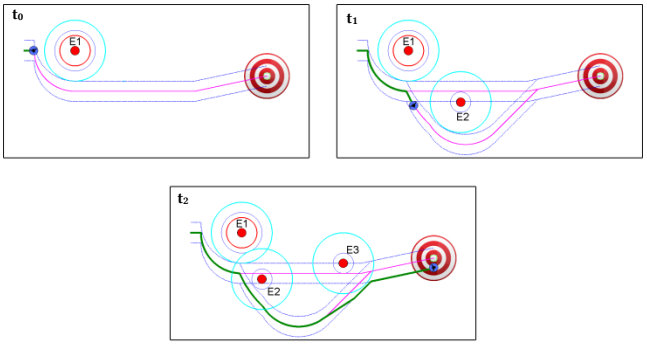

Figure 3: An example of RA* utilizes the robust neighborhood

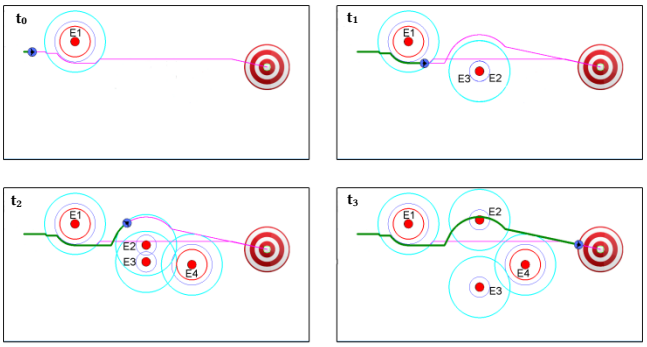

Figure 4: RA* path planning with limited resources

With unlimited resources, RA* successfully accomplishes the mission in all scenarios without accepting risk except in two cases, where enemies construct a virtual wall in front of the target. The drone was unable to reach its target without breaking through the VR and JR of enemies. With 50 seconds, RA* was unable to find feasible paths in 4% of the scenarios. The failed missions occurred because of the distribution of the enemies with WR. Since the drone must avoid WRs regardless of its resource condition, it ran out of time before reaching the target.

Table 1: Comparison between A* and RA* in seconds

	Avg. Planning Time	Avg. Travel Time	Avg. Accepted Risk	Mission Completion
A* (Unlimited Resources)	13.18	29.56	1.8676	98%
RA* (Unlimited Resources)	5.91	26.86	0.1022	100%
A* (Limited Resources)	10.36	26.71	1.227	90%
RA* (Limited Resources)	3.798	25.48	0.4948	100%
A* (Insufficient Resources)	7.129	25.01	0.6444	84%
RA* (Insufficient Resources)	2.499	23.43	0.509	96%

The average planning time of RA* in all cases is less than A*, because it prioritizes the drone safety by avoiding paths in narrow passages between enemies, while A* allows the drone to pass between enemies. This sometimes traps the drone inside moving enemy ranges, consequently increasing replanning time. On the other hand, the RA* reduces replanning attempts by adjusting the trajectory of the current path from the robust neighborhood. Thus, RA* generates safer, shorter, and faster paths overall than A* with and without limited resources.

With A*, the drone was unable to accomplish its mission in 10% and 16% of the scenarios with limited and insufficient resources, respectively. One possible reason for this occurrence when the drone is trapped inside a mobile enemy range for long time while trying to find an optimal path, causing resource depletion before reaching the target. Another reason may be that when the drone consumes its resources to completely avoid all enemies, it has no fuel or time left to accomplish the mission. In addition, the average path length (i.e., travel time) for paths generated by A* in both cases is surprisingly longer than the average length for RA* paths. In fact, A* attempts to find the shortest path, which makes its paths very tight around enemies. With mobile enemies, the tight paths face replanning very frequently, increasing their overall lengths.

Although A* attempts to avoid all risky areas, the drone takes more risk with A* than with RA*, with and without limited resources. This situation is apparent when scenarios surround the drone with multiple enemies, trapping it. One scenario with moving enemies is shown in Figure 6. At time t_1 and t_2, A* spends significant time finding a valid path and stays in its position until enemies E_2 and E_3 move away from the optimal path. In similar scenarios, if enemies E_2 or E_3 move toward the drone, the drone will be trapped inside the risky area until the enemy moves away, which may cost the drone its life.

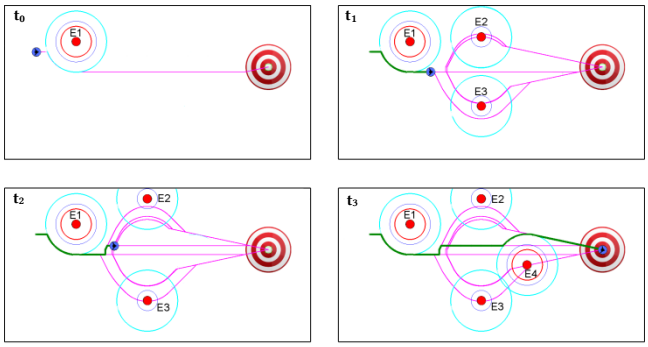

Figure 5: A*path planning in presence of mobile enemies

Figure 7 shows how RA* reacts to the same scenario in Figure 6 with unlimited resources. RA* can find a valid path without accepting any risk. The same scenario with limited resources has been shown in the previous section (Figure 5).

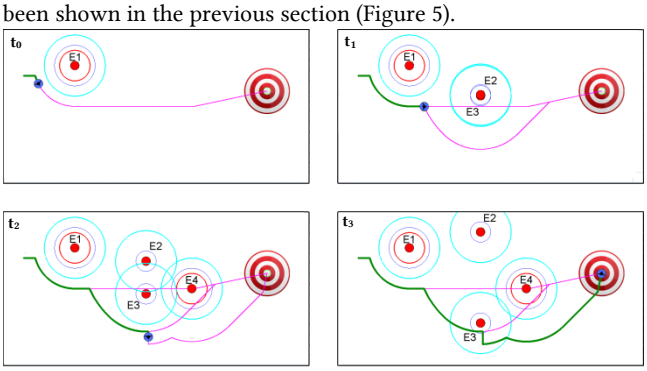

Figure 6: RA* plans with unlimited resources

5 CONCLUSIONS

The presented algorithm, RA*, dynamically and flexibly determines the risk avoidance based on the latest information about the environment and mission constraints. It addresses the "reach-while-avoid-when-possible" path planning problem by using the MTL robustness theory. The experiments showed that RA* is able to avoid mobile enemies better than A*. Moreover, A* is not able to plan for feasible paths under limited resources.

ACKNOWLEDGMENTS

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-16-1-0253. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government.

REFERENCES

[1] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[2] A. Stentz. The focussed D* algorithm for real-time replanning. *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Canada, 1995.

[3] D. T. Wooden. *Graph-based Path Planning for Mobile Robots*. Ph.D. Dissertation, Georgia Institute of Technology, 2006.

[4] B. S. Stewart and C. Chelsea. Multiobjective A*. ACM,2(1991), 775-814.

[5] T. Oral and F. Polat. MOD* Lite: An incremental path planning algorithm taking care of multiple objectives. *IEEE Transactions on Cybernetics*, 1(2016), 245-257.

[6] G. Fainekos and G.J. Pappas. Robustness of temporal logic specifications. *In Proceedings of the First Combined International Conference on Formal Approaches to Software Testing and Runtime Verification*. Springer-Verlag, Seattle, 2006

[7] Y. V. Pant, H. Abbas, and R. Mangharam. Smooth operator: Control using the smooth robustness of temporal logic. In *Proceedings of the IEEE Conference on Control Technology and Applications (CCTA)*, HI, 2017

[8] B. Hoxha, and G. Fainekos. Planning in dynamic environments through temporal logic monitoring. *In Proceedings of the Workshops at the Thirtieth Conference on Artificial Intelligence AAAI*,2016

[9] S. Saha and A.A. Julius. An MILP approach for real-time optimal controller synthesis with metric temporal logic specifications. In *Proceedings of the American Control Conference* ,2016.

[10] Koymans, R. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 4 (1990), 255-299

[11] A. Dokhanchi, B. Hoxha, and G, Fainekos. On-Line monitoring for temporal logic robustness. *In Runtime Verification*. Springer, pp. 1–20,2014

[12] H. Jun and Z. Qingbao. Multi-objective mobile robot path planning based on improved genetic algorithm. *In Proceedings of the International Conference on Intelligent Computation Technology and Automation*, 2(2010).

[13] M. Samadi and F. Othman. Global path planning for autonomous mobile robot using genetic algorithm. In *Proceedings of the International Conference on Signal-Image Technology & Internet-Based Systems*, Japan, 2013.

[14] F. Guo, H. Wang, and Y. Tian. Multi-objective path planning for unrestricted mobile. *In Proceedings of the IEEE International Conference on Automation and Logistics*, Shenyang, 2009.

[15] S. Bayili and F. Polat. Limited-Damage A*: A path search algorithm that considers damage as a feasibility criterion. *Knoweldge-Based System*, 24 (2011), 501-512.

[16] A. Lavin. A Pareto front-based multiobjective path planning algorithm. *Computing Resource Respository*, 2015.