Horizontal Collision Avoidance Resolutions for Unmanned Quadcopters Using ACAS Xu

Leonardo Mouta Pereira Pinheiro*, Jean-Baptiste Chaudron†

Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
*Email: leonardo.MOUTA-PEREIRA-PINHEIRO@student.isae-supaero.fr
†Email: jean-baptiste.CHAUDRON@isae-supaero.fr

***Abstract*—ACAS X is a probabilistic-based collision avoidance system which aims to replace the current standard ACAS II. The original concept has been branched into several variants, including ACAS Xu, intended for unmanned aerial systems. Modeling the dynamics of an encounter between two quadcopters via a Markov Decision Process, whose optimal policy was calculated through value iteration, results in an acceptable number of near midair collisions and alerts, thus showing that ACAS Xu could be potentially used in quadcopter drones, an application thus far neglected. Possible improvements that could bring the simulation closer to real life scenarios were then detailed for use in future research.**

I. Introduction

This work concerns ACAS X technologies and consists of two main lines. In the first one, the general formulation of ACAS X, its current state of the art, its advantages when compared to legacy ACAS II, and the branching of the original concept into ACAS Xa/Xu are briefly described. Afterwards, the article focus on the problem of adapting standard ACAS Xu logic for quadcopter use, which behaves differently than other aircraft currently covered by this system.

Section II contains a brief description of how the general ACAS X concept works. Section III explores some particularities of ACAS Xu - intended for unmanned aerial systems (UAS) - when compared to the baseline concept. Section IV pertains to the specific problem of issuing resolution advisories (RAs) in the horizontal plane for UAS - as opposed to the traditionally adopted climb/descend instructions. Section V details the procedures involved in modeling and simulating ACAS Xu logic for quadcopters. Sections VI and VII discuss the results for two different encounter scenarios, respectively aircraft with zero and non-zero horizontal speeds. Section VIII details some possible improvements to the model used in this article in order to better account for real life operation. Finally, Section IX concludes this paper with some final remarks about the feasibility of implementing ACAS Xu logic in quadcopters.

II. Context and State of the Art

As was stated by [1], current ACAS II technology, although largely successful, is starting to show some limitations, such as its deterministic logic, unnecessary advisories, heavy dependence on transponders and its bad response to non-compliant aircraft (which became clear after the Überlingen collision in 2002).

In order to deal with these shortcomings, ACAS X - a new airborne collision avoidance (CA) paradigm - is being developed, which relies on probabilistic modeling of aircraft flight and use of equipment other than the traditional transponder (such as GPS, for example) [1].

Grosso modo, ACAS X logic works as follows [2]: in offline development, aircraft movement is modeled via a Markov decision process (MDP). Through the use of a computing technique called dynamic programming, the best course of action - based upon the MDP - is transformed into a numerical lookup table. The reason for using dynamic programming - instead of other techniques - as well as the baseline MDP formulation for ACAS X has been put forth by [3]. Also, solving an MDP by means of dynamic programming is a well known problem, with the detailed process described in works such as [4]. On the other hand, during real-time operations the measurements obtained from sensors available to the aircraft are used to generate a state distribution, which is compared against the lookup table. If it is determined that the aircraft is in a route that will result in a violation of its safety volume, the system issues a resolution advisory (RA) to the pilot.

That said, some particularities arise when implementing ACAS X for each type of aircraft, resulting in a branching of the original concept. ACAS Xa, for example, is the general purpose implementation, to be installed in commercial airplanes. Another existing variant is ACAS Xu, for example, which deals with unmanned aerial systems (UAS). There are other specific implementations, but they are not relevant for this work, which shall limit itself to the contrast between ACAS Xa and ACAS Xu.

III. ACAS Xa vs. ACAS Xu

In terms of specifications, some points of difference in which ACAS Xu departs from baseline ACAS Xa are [5]:

- The need to incorporate non-cooperative traffic. ACAS Xa deals exclusively with aircraft carrying cooperative sensors (transponders, ADS-B, etc.). Since ACAS Xu will need to sense non-cooperative traffic, it will also need to include non-cooperative sensors in its design (such as radar or even optical mounts);

- Automatic pilot overrides. In ACAS Xu logic, the autopilot may take over command of the aircraft if the remote pilot fails to heed an issued RA;
- Horizontal resolutions. In order to avoid collisions, ACAS Xa only issues resolutions in the vertical plane (climb, descend). When using ACAS Xu, horizontal resolutions (turn left/right) are sometimes better to avoid collisions.

Economically, [5] also states that the delay in integrating unmanned aircraft into the general airspace costs an estimate 10 billion dollars per year. The same reference warns that this value is probably overestimated, but it offers some perspective on the market size of ACAS Xu regardless. Small drones in particular - such as civilian-operated quadcopters - could also be integrated into the economy, thus generating significant revenue, but not before their operational safety is ensured.

The theoretical groundwork has been laid out by studies such as [6], which covers the performance of ACAS Xu algorithms. Furthermore, proof-of-concept of ACAS Xu logic has already been established by [7], meaning the practical feasibility of the system has already been established.

Thus, it becomes clear that ACAS Xu technology is a field with a lot of potential. Of the issues mentioned above, this work focuses on the problem of horizontal RAs.

IV. Horizontal Resolutions for Collision Avoidance

As explained in Section III, ACAS Xa logic - when confronted with a collision avoidance - only issues RAs on the vertical plane, that is level-off, climb or descend instructions, with varying degrees of intensity for the last two. The work done in [3], for example, explores the results of issuing climb and descend instructions of $\pm 1500\ ft/min$ - with two different acceleration options - and $\pm 2500\ ft/min$.

ACAS Xu also issues horizontal RAs, meaning ACAS Xu can instruct the aircraft to turn left/right instead of climbing/descending if it is a better option for avoiding collisions.

Evidently, the first thing to determine is when to choose between vertical or horizontal conflict resolution. A study on that subject has been published in [8].

The study in [8] (along with many others), however, assumes a turn model for the UAS similar to that of airplanes, thus modeling horizontal advisories as "turn $x°/s$ left/right". While that may be a reasonable description for larger UAS, smaller ones - such as quadcopters - do not behave like that. Instead, a quadcopter generally changes its direction by banking, thus creating a lateral speed. The difference between large UAS and quadcopter movement is illustrated in Fig. 1.

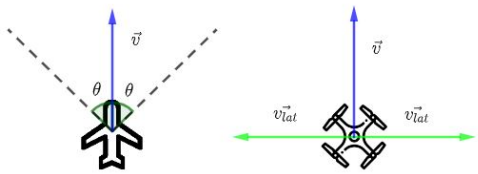


Fig. 1: While a plane or large UAS changes its heading by gradually altering the angle of its speed, a quadcopter usually develops a lateral speed to change its trajectory

What this means is that the previous research models for studying horizontal resolutions in ACAS Xu logic have to be revisited if they are to be adapted for quadcopter operation.

V. Problem Formulation and MDP Modeling

A qualitative description of the problem of horizontal separation is as follows: suppose two aircraft are moving in such a manner that they will have the same altitude after a certain time τ. Suppose also that a near midair collision (NMAC) occurs if their horizontal separation is smaller than a certain critical value ρ_{crit} when they are at the same altitude. Given the dynamics of the aircraft on the horizontal plane, what measures can be taken to avoid an NMAC?

A geometrical representation of this problem, along with the relevant quantities involved, was put forth by [8], which is illustrated in Fig. 2.

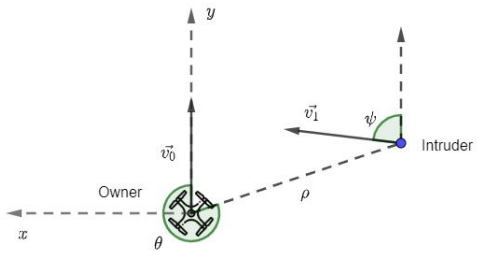


Fig. 2: Geometrical description of horizontal separation problem. Axes x and y are constructed based on $\vec{v_0}$

In order to model the problem as a Markov Decision Process, it is necessary to define the state variables, which can be derived from Fig. 2 and are registered in I. Since an MDP works with discrete states, these state variables will need to be discretized. Any state s will thus be defined by the tuple $s = (\rho, \theta, \psi, v_0, v_1, \tau, a_{prev})$.

TABLE I: State variables used in MDP formulation

Variable	Description	Commentary
ρ	Range to intruder	
θ	Relative intruder bearing	Counterclockwise
ψ	Relative intruder heading	Counterclockwise
v_0	Owner ground speed	
v_1	Intruder ground speed	
τ	Time to loss of vertical separation	
a_{prev}	State of previously issued advisory	Already discrete

The possible actions, which defines the action state, are listed in TABLE II. In order to keep the lookup table as small as possible, no strengthenings or reversals are possible.

TABLE II: Possible actions at each state

Action	Description
coc	No maneuvering necessary
left	Speed v_{lat} oriented towards positive x
right	Speed v_{lat} oriented towards negative x

For this article, the value of v_{lat} is taken to be $1\ m/s$. This is not based on any real quadcopter, which can sometimes reach up to $15\ m/s$ in lateral speed, but rather a small, realistic value that is easy to implement and control in the event of a physical simulation.

As for the rewards at each state s, the values were defined in an *ad hoc* manner and are analogous to the ones used in [9]. These reward values are listed in TABLE III. The collision radius ρ_{crit} was assumed equal to equal $1.5\ m$. This value is a bit high given typical quadcopter dimensions, but is useful for visualization purposes.

TABLE III: Reward values for each state s

Situation	Reward	Condition
NMAC	-1	When $\tau = 0$ and $\rho \leq \rho_{crit}$
Alert	-0.01	When $a_{prev} =$ left or $a_{prev} =$ right
Clear	0.0001	When $a_{prev} =$ coc

Since the size of the final lookup table is an important constraint, for the storage space in quadcopters is usually very limited, it is better to choose different discretization schemes for the state variables based on the expected type of encounter. For that, two situations will be studied, in both of them the intruder also being a small quadcopter drone. These two situations are:

- There are no horizontal speeds involved. In other words, $v_0 = v_1 = 0$. This situation serves as a benchmark: since almost all ACAS X algorithms are made by MIT Lincoln Lab and are unavailable to the general public, it will be necessary to develop most tools from scratch. Thus a simple encounter will serve as a benchmark to ascertain the soundness of the logic implemented, for it is analogous to the simplest "head-on collision" described by [3] and can be qualitatively compared to the results in [9].
- The intruder is also a small quadcopter drone but the initial speeds v_0 and v_1 are non-zero. This is a more general case that will serve to better analyze the particularities of quadcopter encounters. Since there will be more states involved, this scenario is also more computationally costly.

It is not essential to separate these cases for, as shall be seen, the general equations involved apply to both of them. The advantages in doing so, however, consist in having a more clear understanding of the conditions involved, having a "quicker" situation (the first scenario) which can be run and changed without much work, and keeping the number of states involved small, both to generate small lookup tables and to keep dynamic programming as an effective tool for solving MDPs (for an indiscriminately large state space would require other solving methods ir order to be processed in a reasonable amount of time).

Regardless of the encounter type studied, which will influence only the discretization of the variables, there are some hypotheses common to both scenarios, namely:

- The collision avoidance system has perfect state information. In other words, sensors are not involved and there is no noise when determining the state. Relaxation of this hypothesis would transform the MDP into a Partially Observable Markov Decision Process (POMDP).
- The intruder aircraft is not equipped with ACAS and will not have its behavior altered during the course of the encounter. Since quadcopters are not usually equipped with ACAS, this is a reasonable assumption.
- Decisions will be taken at time intervals $\Delta t = 1\ s$. While this is quite a large interval for quadcopters, this is currently the standard for ACAS X.
- If a "left"/"right" action has been been issued and $a_{prev} =$ coc, it takes Δt time for the owner aircraft to respond, after which the lateral speed immediately assumes value v_{lat}. Clear of conflict (coc) actions are executed immediately and all lateral speeds are instantly nullified.

Regarding the dynamic model, the variables ψ, v_0 and v_1 are assumed to remain constant throughout the encounter. As a consequence, updating ρ and θ becomes a simple 2D kinematics problem based on the other parameters: after passing the system to Cartesian coordinates, it is easy to determine the new position of the intruder relative to the owner. Returning these coordinates to a polar frame yields the new ρ' and θ'. Thus, given a state s and action a, the new position x' after time Δt is given by:

$$x' = \begin{bmatrix} \rho' \\ \theta' \\ \psi' \\ v_0' \\ v_1' \\ \tau' \\ a_{prev}' \end{bmatrix} = \begin{bmatrix} ||(x_i(s, v_{lat}, a, \Delta t), y_i(s, v_{lat}, a, \Delta t))|| \\ \arg_\theta \left(x_i(s, v_{lat}, a, \Delta t), y_i(s, v_{lat}, a, \Delta t)\right) \\ \psi \\ v_0 \\ v_1 \\ \tau - \Delta t \\ a \end{bmatrix} \tag{1}$$

With:

$$x_i(s, v_{lat}, a, \Delta t) = \rho \sin\theta + (v_1 \sin\psi - \delta(a, a_{prev}) v_{lat})\Delta t \tag{2}$$

$$y_i(s, v_{lat}, a, \Delta t) = \rho \cos\theta + (v_1 \cos\psi - v_0)\Delta t \tag{3}$$

$$\delta(a, a_{prev}) = \begin{cases} 1, \text{if } a \neq \text{coc and } a_{prev} = \text{left} \\ 0, \text{if } a = \text{coc or } a_{prev} = \text{coc} \\ -1, \text{if } a \neq \text{coc and } a_{prev} = \text{right} \end{cases} \tag{4}$$

It is worth noting that x' is not the new state s'. While variables ψ', v'_0, v'_1 and a'_{pref} will, because of the very construction of the modeling, fall within the values stipulated by discretization; and while the correct combination of a discretization scheme for τ and Δt will also ensure that τ' also falls within the permitted values, the results yielded by the operations over $x_i(s, v_{lat}, a, \Delta t)$ and $y_i(s, v_{lat}, a, \Delta t)$ may very well result in values of ρ' and θ' that fall outside the discretization scheme.

It is thus necessary to create a certain "mapping" of the continuous variable x' to the discrete states s'. The method chosen is to assign a certain probability $p(s'|x')$ that x' will be mapped to s' based on the coefficients involved in the bilinear interpolation of ρ' and θ'. This procedure is detailed and used with good results in [10].

The process goes as follows: suppose ρ' falls between two points belonging to the possible discrete values ρ_1 and ρ_2, with $\rho_1 \leq \rho' < \rho_1$. The probability $p(\rho_1|\rho')$ that ρ' is mapped to state variable ρ_1 is:

$$p(\rho_1|\rho') = \frac{\rho_2 - \rho'}{\rho_2 - \rho_1} \tag{5}$$

In an analogous manner:

$$p(\rho_2|\rho') = \frac{\rho' - \rho_1}{\rho_2 - \rho_1} \tag{6}$$

The exact same methodology can be applied to θ'. Suppose x' is surrounded by 4 states, as shown in Fig. 3. Assuming independence between the mapping of ρ' and θ' to the possible states, the mapping probabilities are given by equation 7.

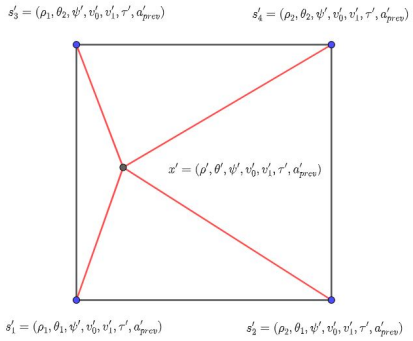


Fig. 3: Position x' (continuous on ρ' and θ') can be mapped to the surrounding states s'_1 to s'_4. Probabilities are based on the coefficients of bilinear interpolation.

$$\begin{cases} p(s'_1|x') = p(\rho_1|\rho')p(\theta_1|\theta') \\ p(s'_2|x') = p(\rho_2|\rho')p(\theta_1|\theta') \\ p(s'_3|x') = p(\rho_1|\rho')p(\theta_2|\theta') \\ p(s'_4|x') = p(\rho_2|\rho')p(\theta_2|\theta') \end{cases} \tag{7}$$

After this discussion, and combining it with the theoretical methodology put forth by [4], it is now possible to fully describe the situation as a Markov Decision Process. To sum up all that has been discussed, the discretization of the state variables in TABLE I results in the grid points upon which the state space $\mathcal{S}$ is built. For each state $s \in \mathcal{S}$, the possible action space $\mathcal{A}$ and the reward space $\mathcal{R}$ are defined in TABLES II and III respectively. Furthermore, under the hypothesis that the reward function depends only on the state s, TABLE III directly leads to the definition of the reward function $R(s)$.

Equations 1-7 lead to the definition of the state transition function $T(s'|s, a) = p(s'|x')$, with x' deterministically defined by equations 1-4, thus guaranteeing all necessary parameters for fully describing the MDP.

Based on a policy $\pi(s)$, assumed to be stationary and which specifies which action a should be taken at stage s, and a value function $U^{\pi}(s)$, which is the expected utility of executing policy π at state s, the objective of the MDP is to find an optimal policy $\pi^*(s)$ which maximizes the expected utility for all states. It is this optimal policy that will be ultimately converted to the lookup table. It can be shown that the expected reward $U^*(s)$ for each state when following an optimal policy satisfies the Bellman equation:

$$U^*(s) = R(s) + \max_a \sum_{s'} T(s'|s, a) U^*(s') \tag{8}$$

It is worth noting that this equation is different from its original formulation, which includes a discount value γ, but is identical to the one used in [9]. This adaptation is made because the MDP created deals with a finite horizon. In real applications, however, it could be a good idea to reintroduce the discount factor in order to prioritize the present over the future, thus balancing any propagated errors.

To find the value of U^* the method of value iteration, an application of dynamic programming, can be used. This is an iterative method that consists in updating the the value of $U(s)$ with each passage until convergence. The lack of a discount factor would imply the need of exact convergence. In practice, since computers tend to introduce error when dealing with large floating point representations, the values were rounded to the 9th decimal place. This rounding was found by trial an error, based upon checking if the optimal policy found were nonsensical or not. Implementation of this value iteration method is shown in Algorithm 1.

Algorithm 1: Value iteration in order to find $U^*(s)$

$k \leftarrow 0$;
$U_0(s) \leftarrow 0$, for all states s;
$\delta \leftarrow 1$;
while $\delta \neq 0$ **do**
  $U_{k+1}(s) \leftarrow$
   $R(s) + \max_a[round(\sum_{s'} T(s'|s,a)U_k(s'), 9)]$,
   for all states s;
  $k \leftarrow k + 1$;
  $\delta \leftarrow \max_s ||U_k(s) - U_{k-1}(s)||$;
end
return U_k

From the value of $U^*(s)$, it is straightforward to obtain the optimal policy $\pi^*(s)$ (again removing the discount factor) [4]:

$$\pi^*(s) = \arg\max_a \left(\sum_{s'} T(s'|s,a)U^*(s') \right) \tag{9}$$

The simulation of encounters to evaluate the optimal policy works as follows: one million encounters will be generated such that the starting state has components $\tau = \tau_{max}$, $a_{prev} =$ coc and variables ρ, θ, ψ, v_0 and v_1 are chosen from a uniform distribution over their permitted values. This is similar to the methodology described in [9].

After the definition of the starting state, the simulation will follow the action proposed by the policy and will transition to next state s' based on the probabilities in the transition model $T(s'|s,a)$ until the simulation reaches a state with $\tau = 0$. During the simulation, the number of alerts issued is counted and the program will verify if the end state represents a violation of the safety volume. Because most values chosen are random, this will enable the exploration of different types of encounters, which in turn allows for a good assessment of the optimal policy.

VI. First Encounter Scenario: Zero Horizontal Speeds

As discussed in the previous section, the first example consists of a simple scenario where no horizontal speeds are involved. By setting setting the horizontal speeds $v_0 = v_1 = 0$, it is also possible to also set $\psi = 0$, thus greatly reducing the state space, which allows for a fast simulation that can be quickly changed.

The discretization scheme for this scenario is listed in TABLE IV, resulting in a total of 16,848 states, which can be computed fairly quickly.

TABLE IV: Discretization of state variables for scenario 1

Variable	Grid Points
ρ	0, 0.2, 0.4, ..., 5 m
θ	0, 10, 20, ..., 350 degrees
ψ	0 degrees
v_0	0 m/s
v_1	0 m/s
τ	0, 1, ..., 5 s
a_{prev}	Already discrete

The optimal policy obtained after running the MDP simulation is illustrated in Fig. 4. Evidently, since the problem is intrinsically three-dimensional in (ρ, θ, τ), the results were plotted considering slices of τ. Also, it is assumed that $a_{prev} =$ coc, in other words that there were no previously issued actions. Also, since the plots at time slices $\tau = 4$ s and $\tau = 5$ s do not bring a lot of information, they were not shown in Fig. 4

At first sight, it would appear that no information can be gained from the empty plot in Fig. 4a, but this is incorrect. In fact, it gives vital insight into the reasoning of the MDP formulation: since a turn action takes one second to be applied, if detection occurs at $\tau = 1$ s there is no possibility to avoid the collision if $\rho \leq \rho_{crit}$, hence the empty plot, for it offers a higher reward not to issue an alert at all.

As for Fig. 4b, it is possible to see a "hole" inside the action points. This is an analogous result to the conclusions reached concerning Fig. 4a: any action issued at $\tau = 2$ s would only be implemented at $\tau = 1$ s, thus giving the drone only a second to avoid the collision. For the chosen values of v_{lat} and ρ_{crit}, this time might not suffice to avoid the collision, hence the empty region. As for the action points, the symmetry observed is expected and is similar to the results obtained in [9].

Finally, Fig. 4c shows that collision can be avoided in the region neglected by Fig. 4b if the threat is detected earlier. The system, however, chooses not to issue warnings at points farther from the center at this time, preferring to deal with them when the time to collision is lower. Also in Fig. 4c it is possible to notice that the previous symmetry has been lost. This is due to programming reasons: when the expected value derived from two different actions are the same, the program tends to prioritize the "bank left" action simply because it is evaluated first.

As for the simulation to ascertain the efficiency of the policy, total NMACs and alerts are listed in TABLE V. While the number of alerts is high when compared to works such as [9], the number of NMACs equaling zero certainly raises some questions. Of course, this result could be brushed aside based on the simplicity of the model, the scenario and the simulation, and should not be taken at face value. However it at least serves to show that the methodology implemented has some potential.

TABLE V: Policy simulation results for scenario 1

Case	Occurrences
NMACs	0
Alerts	765,908

Given this discussion, it is reasonable to assume that the programmed model and simulation comply well with the requirements - both technical and "common sense" - desired to ensure collision avoidance, allowing the migration to the more general case with non-zero horizontal speeds. And, since storage space is an important constraint in quadcopters, it is worth noting that the policy file has a size smaller than 1 MB for this scenario.

VII. Second Encounter Scenario: Non-Zero Horizontal Speeds

For this scenario, the drones will have non-zero horizontal speeds. To keep the number of possible states as small as possible, v_0 and v_1 can only assume two values each. The discretization for the state variables is shown in TABLE VI. The variable ρ has its range increased in relation to the previous scenario in order for the chance of a collision to be higher. In this scheme, a total of 1,959,552 million states were created.

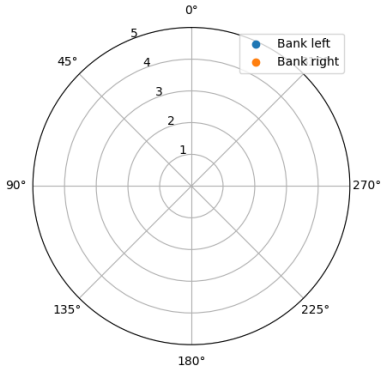
TABLE VI: Discretization of state variables for scenario 2

Variable	Grid Points
ρ	0, 0.5, 1, ..., 10 m
θ	0, 10, 20, ..., 350 degrees
ψ	0, 10, 20, ..., 350 degrees
v_0	1, 2 m/s
v_1	1, 2 m/s
τ	0, 1, ..., 5 s
a_{prev}	Already discrete

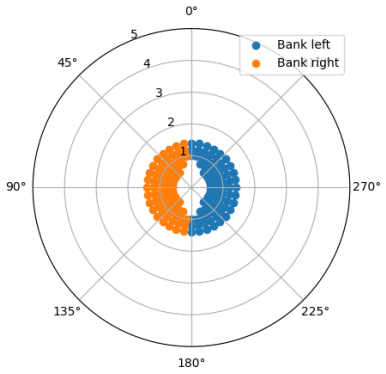
Differently from Section VI, it is not enough just to specify τ in order to specify a policy slice, for there are different combinations of ψ, v_0 and v_1 which should also be specified. Some examples are shown in Fig. 5, Fig. 6 and Fig. 7. Again, it is assumed no action has been issued previously, so $a_{prev} =$ coc

Fig. 5, in which the intruder is heading directly towards the owner aircraft, again shows some of the trends already visible in scenario 1, such as the "hole" in Fig. 5a, meaning no collision prevention is possible for these points, and the tendency to prefer "bank left" action in Fig. 5b.

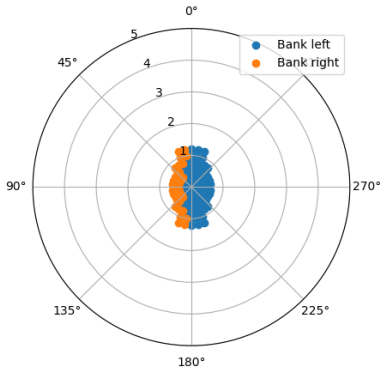
Fig. 6 on the other hand illustrates the effect of an "oblique" encounter, with the owner and intruder having different speeds. It is interesting to note that the expected action varies depending on the θ position of the intruder, completely suppressing the previous existing symmetry. This is expected when compared to the base ACAS Xu implementations, such as shown in [8], which also show the actions necessary for the intruder approaches from an oblique angle. Another example of an oblique approach is shown in Fig. 7, reinforcing this discussion. Fig. 7 also raises an interesting discussion: for this combination of intruder heading and speeds, instead of



(a) $\tau = 1\ s$

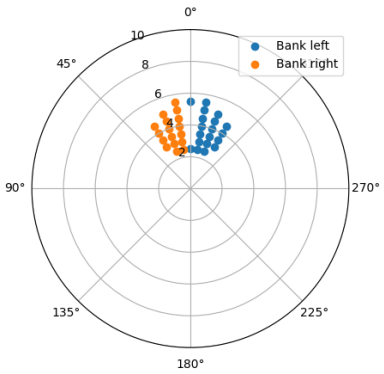


(b) $\tau = 2\ s$

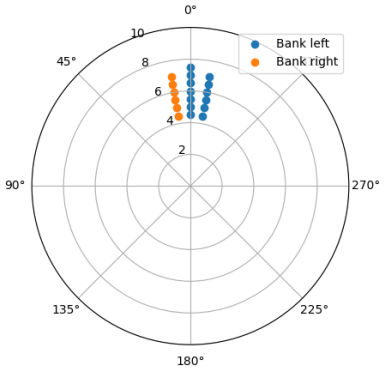


(c) $\tau = 3\ s$

Fig. 4: Optimal action plots for the first scenario. The plots are polar with ρ and θ defined as in Fig. 2. Each plot indicates a different τ slice.

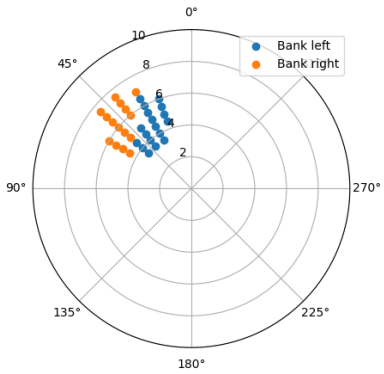


(a) $\tau = 2\ s$

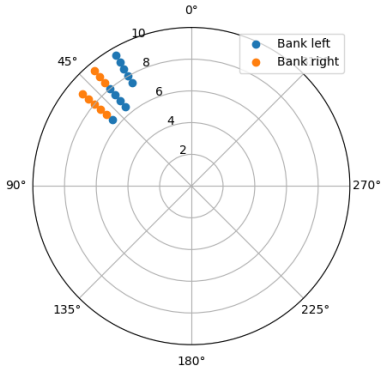


(b) $\tau = 3\ s$

Fig. 5: Optimal action plots for the second scenario. Each plot indicates a different τ slice. In all plots, $\psi = 180°$, $v_0 = v_1 = 1\ m/s$



(a) $\tau = 2\ s$



(b) $\tau = 3\ s$

Fig. 6: Optimal action plots for the second scenario. Each plot indicates a different τ slice. In all plots, $\psi = 240°$, $v_0 = 1\ m/s$ and $v_1 = 2\ m/s$

a collision because of "opposite" paths, the aircraft converge on the same position from the same direction.

Concerning the trajectory simulations, results are registered in TABLE VI. The number of NMACs increases significantly, showing that it is harder to avoid collisions when both aircraft are moving. On the other hand, the reduction in the number of alerts means that it is rarer for an encounter to happen in this scenario. This is evident, for the risk of collision, given v_0 and v_1, involves certain combinations of (ρ, θ, ψ) that may not always be the case. Averaging the values of NMACs and alerts of the two scenarios, the result is in the same ballpark as those shown in [9].

TABLE VII: Policy simulation results for scenario 2

Case	Occurrences
NMACs	9
Alerts	66,900

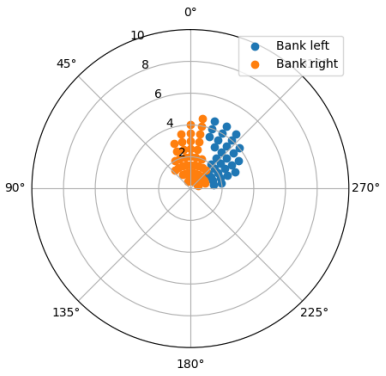
Finally, in terms of storage space, the policy file generated by this encounter has a size of approximately 56 MB, which is not exceedingly high.

VIII. Possible Model Improvements

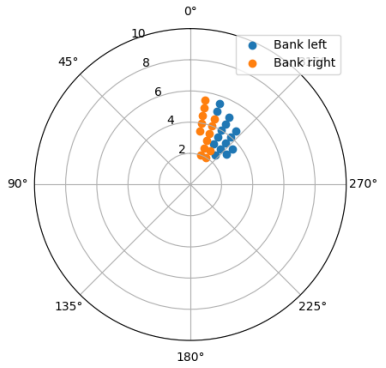
Evidently, some assumptions and hypotheses made during the development of the model are quite naive. Starting with the dynamic equations, which could include random noise accelerations, meaning that ψ, v_0 and v_1 could change from update to update. This type of approach has been described in [3] and would result in altering the transition model calculated.

Still concerning the dynamical model, some other constraints might be relaxed. For instance, the drone could react faster and with higher speeds if a collision is imminent, or even could apply the lateral speed increasingly instead of instantaneously. Also, the value chosen for v_{lat} is actually quite low if some commercially available quadcopters are considered and could be increased, meaning collisions would be less likely.

Another possible model improvement is to change the reward function $R(s)$, dependent only on the state, into a function $R(s, a)$, which also depends on the action. This could eliminate the prioritizing of the first evaluated action by adding a small additional negative reward if the action taken results in the relative intruder bearing θ "changing sides" (eg. goes from a value $\theta < 180°$ to $\theta > 180°$).



(a) $\tau = 2\ s$



(b) $\tau = 3\ s$

Fig. 7: Optimal action plots for the second scenario. Each plot indicates a different τ slice. In all plots, $\psi = 30°$, $v_0 = 2\ m/s$ and $v_1 = 1\ m/s$

Of the assumptions listed, the most controversial probably is that the system possesses perfect state information. During real operation, the current state is determined by measurements acquired by the drone sensors, which undoubtedly include errors and noises, not to mention the problem of even calculating some variables, such as τ [5]. Bearing and heading measurements specially are known to be imprecise. These uncertainties added by the sensors can be included in the modeling by transforming the MDP into a POMDP, which can be done by adding an observation model $O(o, s)$, which is the probability of observing o given a state s [4]. One possible example of observation model, for example, comes from assuming that o follows a Gaussian distribution with mean s and some chosen standard deviation.

Finally, given that small quadcopters tend to fly in low-altitude, packed airspace, this simple one intruder encounter may not be realistic. Works such as [11] offer some insight into more realistic encounter modeling, for example. Indeed, [2] cites the question of packed airspace as one of the problems surrounding ACAS Xu, for the higher number of states involved would require larger lookup tables that might be too big to be stored even inside high performance drones, much less a quadcopter. One possible solution is to make sure that the quadcopter stays in contact with some ground station that stores the lookup tables, but that could mean an increase in response time.

IX. Conclusions

All in all, and even considering the simplicity of the formulated model and simulation, it is safe to assume that there is potential in applying ACAS Xu logic to quadcopters. Though better models might give a more precise idea of behavior during real life scenarios, the general sense obtained in this article is that all important constraints and results were kept within acceptable values.

Further investigation into this topic, however, will help fine-tune the methodology in here described. Some possible pathways for future research include simulation which take the sensor into account and the description of situations with multiple intruders during and encounter.

References

[1] Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. *Next-generation airborne collision avoidance system*. Tech. rep. Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[2] Dimitra Giannakopoulou, Dennis Guck, and Johann Schumann. "Exploring model quality for ACAS X". In: *International Symposium on Formal Methods*. Springer. 2016, pp. 274–290.

[3] Mykel J Kochenderfer and JP Chryssanthacopoulos. "Robust airborne collision avoidance through dynamic programming". In: *Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371* 130 (2011).

[4] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[5] Guido Manfredi and Yannick Jestin. "An introduction to ACAS Xu and the challenges ahead". In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE. 2016, pp. 1–9.

[6] Michael P Owen, Adam Panken, Robert Moss, et al. "ACAS Xu: Integrated Collision Avoidance and Detect and Avoid Capability for UAS". In: *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE. 2019, pp. 1–10.

[7] Tatsuya Kotegawa. "Proof-of-concept airborne sense and avoid system with ACAS-X U flight test". In: *IEEE Aerospace and Electronic Systems Magazine* 31.9 (2016), pp. 53–62.

[8] Michael P Owen and Mykel J Kochenderfer. "Dynamic logic selection for unmanned aircraft separation". In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE. 2016, pp. 1–8.

[9] Mykel J Kochenderfer and James P Chryssanthacopoulos. "A decision-theoretic approach to developing robust collision avoidance logic". In: *13th International IEEE Conference on Intelligent Transportation Systems*. IEEE. 2010, pp. 1837–1842.

[10] Mykel J Kochenderfer, James P Chryssanthacopoulos, Leslie P Kaelbling, et al. *Model-based optimization of airborne collision avoidance logic*. Tech. rep. MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, 2010.

[11] Mykel J Kochenderfer, Matthew WM Edwards, Leo P Espindle, et al. "Airspace encounter models for estimating collision risk". In: *Journal of Guidance, Control, and Dynamics* 33.2 (2010), pp. 487–499.