Modelling and Implementation of Unmanned Aircraft Collision Avoidance

Weizhi Feng[1,2], Cheng-Chao Huang[3], Andrea Turrini[1,3][0000−0003−4343−9323], Yong Li[1][0000−0002−7301−9234]

[1] State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences – Beijing, China
[2] University of the Chinese Academy of Sciences – Beijing, China
[3] Institute of Intelligent Software, Guangzhou – Guangzhou, China
{fengwz,turrini,liyong}@ios.ac.cn, huangchengchao@gziis.org

Abstract. With the increasing application of unmanned aircraft in civil airspace, collision avoidance systems for unmanned aircraft are becoming more and more important and valuable. An ideal collision avoidance system gives the aircraft an optimal strategy for choosing flight actions to avoid collision risks when it detects other aircraft nearby. Currently the general approach to generating collision avoidance logics is to model the problem as a partially observable Markov decision process (POMDP), and then synthesize an optimal policy. However, the existing systems require the precise position information of the intruder aircraft to generate the avoidance actions and ignore the effects of the flight path changes, which may result in its lower robustness or a wasting of flying resources. In this paper, we construct a collision avoidance system based on limited information that reduces the variations from the original flight path. We use POMDPs to model the collision avoidance system with only the destination information of our own aircraft and rough information about the intruder position and generate the collision resolution logic. We implement the collision avoidance module, embed it into the real unmanned aircraft system over PX4 flight control platform and demonstrate the effectiveness of our system by flight simulation.

1 Introduction

Safety is a core requirement for aviation. In order to increase safety, modern aviation involves pilots that are periodically trained on the airplane types they are qualified for, air-traffic controllers that instruct pilots about what route and altitude to keep and what ground movements to perform, and more and more advanced computerized control systems on board of the aircraft to help pilots to control their airplane. One of the most dangerous situations during a flight is the possibility to have a mid-air collision. The standard way for avoiding a collision is to implement a strict and effective air traffic control, where pilots adapt route and altitude as instructed by the air-traffic controller. When this is not possible, e.g., in the middle of a trans-oceanic flight, pilots are assisted by aircraft on-board automation.

1.1 Airborne Collision Avoidance System

As air travel became commonplace, the risk of a mid-air collision increased; the research on automatic aircraft collision avoidance systems gained attraction. In the late 1960s and early 1970s, several manufacturers developed prototype aircraft collision avoidance systems, like the Beacon Collision Avoidance System (BCAS). BCAS uses reply data from the Air Traffic Control Radar Beacon System transponders to determine an intruder's range and altitude. In 1978, FAA started the development of Traffic alert and Collision Avoidance System (TCAS) [9, 10] widely used nowadays. As an improvement over BCAS, TCAS can issue a resolution advisory, including a suggested "climb" or "descent" rate, to the pilot to reduce the collision risk when other aircraft are detected nearby. After decades of development, TCAS is currently mandated on board of all large passenger and cargo aircraft worldwide.

TCAS uses an on-board beacon radar to monitor the local air traffic and logic to determine when to alert pilots to potential near mid-air collision (NMAC). The logic is specified using pseudocode that has been validated through simulation studies to ensure the system safety while remaining operationally acceptable. This decision logic was the result of decades of experience which, however, also limits its robustness. For instance, it does not consider the scenario where pilots do not follow the suggested collision avoidance manoeuvre. Due to the high complexity of the TCAS pseudocode, it is hard to fully understand the overall TCAS logic, let alone modify it. See [9] for more details about TCAS and the development of airborne collision avoidance systems.

1.2 Towards Collision Avoidance in Unmanned Aircraft

In recent years, a new collision avoidance system—Airborne Collision Avoidance System X (ACAS X) [8]—has been proposed and started to be investigated by many researchers. The approach currently implemented in ACAS X is to model the aircraft nearby aviation environment as a partially observable Markov decision process (POMDP), and use dynamic programming algorithms to solve the POMDP model to get the optimal control policy ensuring as much as possible that a collision does not occur. The collision avoidance system then stores the policy as a lookup table, queries the table and chooses the optimal action to avoid the collision when other aircraft are detected. It has been shown in [8] that ACAS X outperforms TCAS in terms of the robustness, efficiency and flexibility of the generated threat resolution logic.

ACAS Xu, the version of ACAS X specifically tailored for unmanned aircraft, is still under development. In fact, in recent years aviation safety needed to consider not only human-piloted airplanes, but also unmanned ones. Unmanned aircraft are becoming more and more common in several scenarios: from military service where they are used in several types of missions due to their lightness, hiding ability, and safety of the pilot, to civil service where unmanned aircraft can be used for aerial photography, resource exploration, express delivery, plant protection, and so on. With the increasing number of unmanned aircraft used in public airspace, the safety of unmanned aircraft has begun to attract attention. In particular, for mid-air collision avoidance, unmanned aircraft need to have the ability to sense other aircraft and be able to enforce the flight corrections needed to preserve the aircraft integrity.

Compared to the passenger and cargo aircraft, the design of a collision avoidance system for unmanned aircraft faces a lot of new challenges, such as more complex flying environments, weaker sensor capabilities and tight time limit to react to avoid other aircraft, which make it difficult to ensure the correctness of the collision avoidance systems for unmanned aircraft. In particular, differently from large transport aircraft [12], unmanned aircraft collision avoidance has to consider the following challenges: 1) unmanned aircraft need to sense non-cooperative traffic and tend to use cheaper, lighter, but noisier sensors due to sensor cost and payload; 2) aircraft flying in controlled airspace have similar performances while unmanned aircraft have a larger range of performances. The resolution logic has to support more complex situations; 3) ACAS X only provides resolution in the vertical plane for manned aviation, but unmanned aircraft need both vertical and horizontal resolutions for safety reasons.

There are various previous works on collision avoidance systems for unmanned aircraft [1, 5, 20]. [20] modeled the collision avoidance system using POMDPs under limited observation capabilities, applied algorithms in discrete state space to get the policy, and performed experiments in a simulator with parameters taken from the unmanned aircraft Global Hawk. [1] directly solved continuous-sate POMDPs to handle high-dimensional state space situations. [5] used a deep neural network to learn an approximation of the lookup table. However they do not implement the obtained avoidance logic on real flight systems, as we do in this work. Also, their simulation results usually do not consider the impact of the generated logic on the flight path, which we also analyze.

These works are usually rather dependent on the precise detection information of the intruder aircraft (e.g. position, distance from own aircraft, etc.), as obtained from the sensors, to generate the avoidance actions. In practice, however, due to the weak sensor capabilities and complex flying environments, it is often very difficult to obtain such a detailed information about the intruder, as well as about our own aircraft, which may result in lower robustness of the system or waste of flying resources.

1.3 Contribution of the Paper

To address the problems described above, in this paper we construct a collision avoidance system working with limited information and minimizing the corrections to the original flight path in order to save flying resources. We use POMDPs to model the collision avoidance system with only the destination information of our own aircraft and rough direction information of the intruder aircraft and then generate the collision resolution logic. We evaluate the effectiveness of our proposal by implementing the collision avoidance module and running it in a Pixhawk simulator, as well as embedding it into the Pixhawk unmanned aircraft system running the PX4 flight control platform, which has been widely used for both customer drones and industrial applications, such as boats and under water vehicles. As our experiment on flying a real Pixhawk unmanned aircraft shows (cf. Section 4.4), the generated avoidance logic performs well in avoiding the intruder also in real world situations.

2 Partially Observable Markov Decision Processes

In this section we briefly recall POMDP formulation and solution methods for unmanned aircraft collision avoidance as given in [1].

Consider an agent (for example, a robot in the robotic navigation problem, or the unmanned aircraft in our setting) operating in a stochastic environment. The agent together with the environment represent our system, which keeps information about the state of the agent and of the environment. At each time step the agent, based on its view of the system state, decides what action to perform; this results in a change of the agent (and system) state, change that can be governed by a probability measure to model uncertainty in the outcome of the action. When the agent has a perfect knowledge of the state of the system, it can be modeled by means of a Markov decision process [15]; when the knowledge is only partial, partially observable MDPs allow for a better modeling of the system: in the context of unmanned aircraft equipped with noisy sensors, the exact position of an intruder is not known exactly, but only a coarse observation about its position (like: being on the left/front/right) is available.

POMDPs augment MDPs by hiding the actual states and providing an observation process that generates observations probabilistically based on the underly state. A POMDP models an agent decision process taking a sequence of actions to maximize the total reward obtainable from the underlying system. The agent, however, is not able to know precisely the underlying system state, but only gets some observations about it; different underlying system states can result in the same observation. The agent keeps a belief over the system state according to the actions performed and the observations seen so far, where a belief is usually a probability measure over the system's states.

2.1 POMDP Formulation

Formally, a POMDP is a 7-tuple $(S, A, O, T, Z, R, \gamma)$ where S, A, O denote the state, action, and observation spaces of the system; $T\colon S \times A \times S \to [0, 1]$ and $Z\colon S \times A \times O \to [0, 1]$ are the transition and observation functions, respectively, such that for each $s \in S$ and $a \in A$, and $s' \in S$ and $o \in O$, $T(s, a, s') = p_S(s'|s, a)$ and $Z(s, a, o) = p_O(o|s, a)$ for appropriate probability measures p_S and p_O modelling the system's dynamics and observations, respectively; $R\colon S \times A \to \mathbb{R}$ is the reward function; and $\gamma \in [0, 1)$ is the discount factor balancing the trade-off between immediate and future rewards. In the remainder of the paper, we assume that both A and O are finite sets.

At each step, the POMDP evolves as follows: when the system is in state $s \in S$ and action $a \in A$ is performed, the system 1) gets the reward $r = R(s, a)$ for having performed action a in s, 2) moves to state s' with probability $T(s, a, s')$, 3) and generates the observation $o \in O$ for s' with probability $Z(s', a, o)$. Note that the observation depends on the states reached after the transition.

Since the agent has only a belief over the system state, it needs to update its belief according to the performed action a and the observed observation o. Let $\mathcal{B}$ denote the set of beliefs, i.e., the set of probability measures over S. Given $b \in \mathcal{B}$, after taking action a and observing o, the agent updates its belief about the system's next state s' as follows:

$$b_a^o(s') = \eta Z(s', a, o) \int_{s \in S} T(s, a, s') b(s) \, \mathrm{d}s, \tag{1}$$

where η is a normalization factor ensuring $\int_{s \in S} b_a^o(s) = 1$. To simplify the notation, we may just write b' instead of b_a^o when a and o are clear from the context. This process of updating the belief state follows directly from Bayes' rule and is known as state estimation or filtering [7].

2.2 Solution Methods

POMDPs, like MDPs, exhibit both probabilistic and nondeterministic behaviors. The nondeterminism existing between the different actions available in a system state is resolved by a policy π representing the agent's choice, i.e., a function $\pi \colon \mathcal{B} \to A$ that specifies which action to perform based on the current belief.

The execution process of the agent alternates between action selection and belief update. Given π and the agent's current belief b, the control of the agent takes the action $a = \pi(b)$. Then the belief is updated according to Eqn. (1).

The goal of solving a POMDP is to synthesize an optimal policy π^* maximizing the agent's expected total reward $\mathbb{E}(\sum_{n=0}^{\infty} \gamma^n R(s_n, a_n))$, where s_n and a_n denote the system's state and chosen action at step n, respectively. A policy π induces a value function $V_\pi \colon \mathcal{B} \to \mathbb{R}$ representing the expected total reward. The value of the total reward obtained by following π from the belief b is

$$V_\pi(b) = \mathbb{E}\big(\sum_{n=0}^{\infty} \gamma^n R(s_n, a_n) | \pi, b\big). \tag{2}$$

In practice, solving POMDPs exactly is challenging due to the high computational complexity, even for relatively small-sized problems [16]. A variety of exact solution methods can be found in previous literature, see, e.g., the witness algorithm [6]. However, these exact methods are not suitable for large-sized problems. To synthesize a policy for large POMDPs, approximate solution methods have been developed [3]. Among them, point-based POMDP algorithms have achieved special success in computing approximate solutions to large discrete POMDPs [13, 16]. Due to the fact that the optimal value function V^* satisfies the Bellman equation [4]

$$\begin{aligned} V^*(b) &= H(V^*(b)) \\ &= \max_{a \in A} \Big\{ \int_{s \in S} R(s, a) b(s) \, \mathrm{d}s + \gamma \sum_{o \in O} p(o|b, a) V^*(b_a^o) \Big\} \end{aligned} \tag{3}$$

where H is called the *backup* operator, most of the solution methods use it to compute the optimal V^*. Value iteration is a commonly used dynamic programming algorithm based on the Bellman equation for solving MDP problems [19]; its basic structure has also been used for developing POMDP algorithms [16].

In the value iteration process, an initial value function V is chosen so to represent the initial known information about the value; if no information is known in advance, then V is taken to be 0 for all beliefs. Then it performs backup operation H on V by repeatedly computing the Bellman equation until the iteration process converges to a fixed point. Once the iteration procedure terminates, the corresponding policy is extracted by taking, for each belief b, the action a satisfying the equality in Eqn. 3.

When implemented naively, the value iteration algorithm needs to search the entire belief space $\mathcal{B}$, which will face the curse of dimensionality and exponential time complexity in large real-world-sized problems [16]. Two main ideas of point-based algorithms have brought about significant progress in improving computational efficiency [2]. First, a small set of points is sampled from the belief space $\mathcal{B}$ as an approximate representation of $\mathcal{B}$. The algorithm performs backup operations over the sampled belief points rather than the entire belief space $\mathcal{B}$ to compute the approximately optimal value function. Second, the α-vectors [11] are used to approximate the value function $V_\pi \colon \mathcal{B} \to \mathbb{R}$ representing the expected total reward of executing policy π starting from b. The optimal value function can be arbitrarily closely approximated by a piece-wise linear and convex function [18] of the form $V^*(b) = \max_{\alpha \in \mathit{\Gamma}} \alpha \cdot b$, where $\mathit{\Gamma} = \{\alpha_0, \alpha_1, \ldots, \alpha_m\}$ is a set of m vectors called α-vectors. Intuitively, each α-vector can be associated with an action and for the current belief, the solution policy selects the action corresponding to the best α-vector. Thus the policy can be represented by the set $\mathit{\Gamma}$ of α-vectors [11]. By using α-vectors, we can use partial policies calculated at one belief point for other parts of the belief space $\mathcal{B}$ under appropriate conditions, thereby significantly improving the computational efficiency [16].

Point-based algorithms differ in how they sample the belief space and perform backup operations. PBVI [13] has been the first point-based algorithm to work well on large POMDPs. HSVI2 [17] uses heuristics to guide sampling from the belief space, which can reduce the gap between the upper and lower bounds of the optimal value function, thus speeding up convergence. SARSOP [11] is related to HSVI2, but it attempts to sample the optimally reachable space $\mathcal{R}^*(b_0)$ from the initial belief b_0 through learning-enhanced exploration and bounding techniques.

Though point-based algorithms have made great progress in solving discrete POMDPs, progress on continuous state space POMDPs is rather limited. Continuous POMDP models have more powerful representation capabilities for real-world systems; however, they make it hard to deal with the "curse of dimensionality". One common way is to place a regular grid over S and construct a discrete POMDP model first. Then one can solve the latter by using existing discrete POMDP algorithms. The difficulty with this approach is that the discretization of the state space often results in a number of states that is still too large. One idea to avoid this is to use a particular parametric form to represent beliefs and value functions of states, e.g., a linear combination of Gaussian distributions [14]. Another idea is to use particle filters to represent beliefs, like in MC-POMDP [21]. MCVI (Monte Carlo Value Iteration) [2] combines the method of particle-based belief representation and the key idea of discrete point-based POMDP algorithms: the α-vectors. MCVI represents the solution policy by means of a policy graph G [2], that is, a directed graph that encodes the optimal policy by labeling each node of the graph with an action and each edge with an observation; beliefs are not stored explicitly in the graph. Given a belief b, MCVI synthesizes the policy from a policy graph by starting from some appropriate node v of G. The action chosen by the policy is the one labeling v; then the agent receives an observation o and moves to the successor node v' by following the edge (v, v') labeled by o. This process is repeated to get the policy.

For each node v of the policy graph G, we can define the α-function α_v: $\alpha_v(s)$ is the expected total reward of executing $\pi_{G,v}$ from the current state s:

$$\alpha_v(s) = \mathbb{E}\big(\sum_{n=0}^{\infty} \gamma^n R(s_n, a_n) | \pi_{G,v}, s\big). \tag{4}$$

Then the value of b with $\pi_{G,v}$ is defined by the α-functions associated with G:

$$V_G(b) = \max_{v \in G} \int_{s \in S} \alpha_v(s) b(s) \, \mathrm{d}s. \tag{5}$$

The policy graph is constructed by approximate dynamic programming techniques and MCVI performs Monte Carlo (MC) simulation to sample the state space and the corresponding belief space. The optimal POMDP value function V^* is computed iteratively according to the backup operation in Eqn. (3), i.e., by the Bellman equation, on each belief b in the reachable belief set. Each backup can choose an action that maximizes the sum of the expected immediate reward and the expected total reward at the next belief. Then the action is added to the policy graph to make it closer to the optimal policy. The reachable belief set is sampled from the whole belief space as an approximate representation of the belief space. The MCVI method performs well in dealing with integral calculations in continuous state problems, thus avoiding to discretize the state space as a grid and improving its efficiency. For more details, see [2].

3 Collision Avoidance Models

To the best of our knowledge, the algorithms and models used in the latest versions of ACAS X and ACAS Xu are not publicly accessible. Therefore, we define our model based on the early published version of ACAS X [8] and ACAS Xu [1]. In our implementation, the aircraft model is parametric and can be easily modified to fit different types of aircraft. In this work, we use the parameter values from Pixhawk[1] and build the model with PX4[2].

[1] `https://pixhawk.org/`

[2] `https://dev.px4.io/master/en/index.html`, PX4 Development Guide.

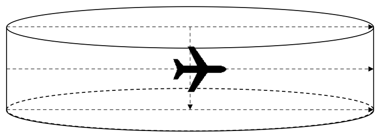


Fig. 1. Threshold range for starting the system in 3D

In our model, we consider a simple encounter situation between two aircraft, our own aircraft and an intruder aircraft. Our own aircraft has no prior knowledge about the flight path of the intruder aircraft, when an encounter happens. During this period, our own aircraft can only get information about the intruder aircraft's elevation and bearing by sensors onboard. The collision avoidance system is responsible to control our own aircraft so to keep a safe distance from the intruder by choosing appropriate actions to avoid collision.

We will describe the state-transition, observation and reward modeling in our model in detail, similarly to the modeling proposed in [1].

3.1 State-Transition Modeling

The position of a flying aircraft can be represented as a point in the 3-dimensional space. The collision avoidance system needs to intervene when the horizontal and vertical distances of two aircraft are less than the given thresholds, which depend on the type of the aircraft and other parameters; this can be represented as the cylinder shown in Fig. 1.

Similarly to other works on collision avoidance [8,20], in order to simplify the model's state space and make verification more practical, we omit the altitude and consider only the latitude/longitude of the aircraft. So when an intruder is detected, the system tries to avoid the collision by banking left or right, i.e., by changing our aircraft's direction.

State space. Since we ignore the aircraft altitude in our model, we let (x, y) be the position of the aircraft with respect to the Earth coordinate system, where the positive x-direction points East and the positive y-direction points North. Let θ be the heading angle of the aircraft with respect to East and u the aircraft's horizontal forward speed. The state information of each aircraft is then (x, y, θ, u). By combining the information about our aircraft and the intruder, a state in the POMDP representing our model is a 7-tuple $(x_1, y_1, \theta_1, x_2, y_2, \theta_2, u)$, where (x_1, y_1, θ_1, u) stands for the state of our own aircraft while (x_2, y_2, θ_2, u) is the intruder's state, by assuming that both aircraft have the same speed.

The state space of the POMDP is continuous, given the nature of the aircraft information. In order to compute the next action for our own aircraft, traditional methods have to first discretize the state space by means of a grid, where each grid cell becomes a state of the discrete POMDP [9]. In our model, we will directly work in the continuous state space instead. In this way, we can apply Monte Carlo methods to improve efficiency of the computation of the policy.

Action space. Now we introduce our modeling of actions. In order to avoid a collision in an encounter situation, we can change our aircraft direction by changing its forward angle, that is, we bank the aircraft. Thus we can represent the angular velocity $\omega \in \{-\omega_m, 0, \omega_m\}$ as an action in our model, where ω_m is the maximum input value for the banking rate. Consequently, if at step t the heading angle is θ_t and we take the action ω, then the heading angle will become $\theta_{t+1} = \theta_t + \omega \Delta t$ at step $t + 1 = t + \Delta t$ after a small duration of time Δt.

Transition dynamics. Now we are ready to give the state-transition dynamics of an aircraft in an encounter situation. Recall that in our model, a state consists of the information about both aircraft. Let s_t be the state at step t and s_{t+1} the new state at step $t + 1$ after a small duration of time Δt, where Δt is the sum of two small durations Δt_h and Δt_f, i.e., $\Delta t = \Delta t_h + \Delta t_f$. In order to avoid a collision, our own aircraft first changes the heading angle by banking ω degrees for a small amount of time Δt_h and then goes forward at the constant speed u. After a small amount of time Δt_u, our own aircraft will reach the new position (x_{t+1}, y_{t+1}) from (x_t, y_t). This transition dynamics is formalized as follows.

$$\begin{aligned} \theta_{t+1} &= \theta_t + \omega \Delta t_h, \\ x_{t+1} &= x_t + u \Delta t_f \cos \theta_{t+1}, \\ y_{t+1} &= y_t + u \Delta t_f \sin \theta_{t+1}. \end{aligned} \tag{6}$$

Note that at the same time, the intruder will also change its flight state (x_2, y_2, θ_2, u) in a similar way by following the action $\omega_{intruder}$. Since our own aircraft does not know the action of the intruder, we can model it in different ways: for instance, the action $\omega_{intruder}$ can be chosen between bank-left, go-forward, and bank-right with equal probability of $\frac{1}{3}$. Other policies can be used to represent different flight plans for the intruder, such as always banking left or right, to represent an intruder flying in circle; we can also consider policies where the intruder tries to intercept our aircraft. We leave the analysis of such policies to the extended version of this paper.

3.2 Sensor Modeling

The threat resolution logic behind a collision avoidance system relies on the aircraft on-board sensors to detect intruders. Electro-optical/infrared (EO/IR) sensors and passive radars are two types of sensors commonly used on large aircraft. On smaller aircraft, the on-board sensors usually have limited accuracy; to model this scenario, we use as possible observations $\{N, L, R\}$, which stand for the fact that no intruder is detected, it is detected on the left of the aircraft, or on the right, respectively. Fig. 2 shows the observation values corresponding to the possible positions of the intruder with respect to our own aircraft.

By following the state modeling introduced in Section 3.1, we can let (x_1, y_1) be the coordinates of our aircraft in the Earth coordinate system and θ its heading angle with respect to East. Let (x_2, y_2) be the coordinates of the intruder

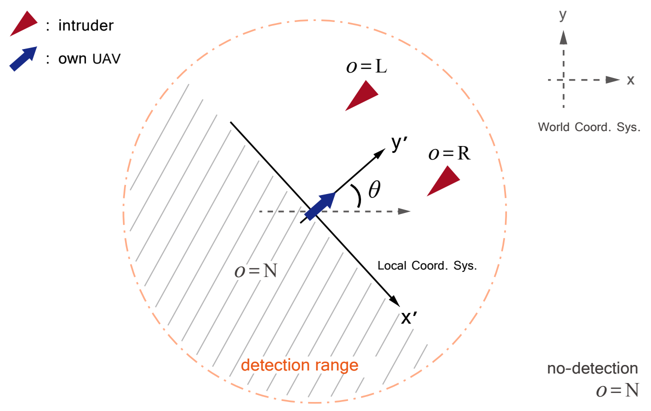


Fig. 2. Observation Model

in the Earth coordinate system and (x'_2, y'_2) be the corresponding coordinates in the local coordinate system of our own aircraft, which is represented in Fig. 2 by the $x'y'$-coordinate system whose origin coincides with our aircraft. In order to get the position of the intruder with respect to our own aircraft, we convert the coordinates of the intruder into our local coordinate system as follows:

$$\begin{aligned} x'_2 &= (x_2 - x_1)\sin\theta - (y_2 - y_1)\cos\theta, \\ y'_2 &= (x_2 - x_1)\cos\theta + (y_2 - y_1)\sin\theta. \end{aligned} \tag{7}$$

We let d denote the horizontal relative distance of two aircraft and o denote the observation value of the intruder; the value d can be computed as

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = \sqrt{{x'_2}^2 + {y'_2}^2}, \tag{8}$$

while the value for o, for a given detection range r, is

$$o = \begin{cases} N & \text{if } y'_2 < 0 \lor d > r, \\ L & \text{if } y'_2 \geq 0 \land d \leq r \land x'_2 < 0, \text{ and} \\ R & \text{if } y'_2 \geq 0 \land d \leq r \land x'_2 \geq 0. \end{cases} \tag{9}$$

3.3 Reward Modeling

As we have seen in Section 2.2, a solution for a POMDP is a policy that maximizes the expected total reward. In order to synthesize a policy piloting our own aircraft to its destination while avoiding collisions, we use rewards to compute a policy that fulfills our requirements.

When at state s and the policy chooses to take action a, a penalty value (i.e., a negative value) $r = R(s, a)$ is computed: it depends on the distance between our aircraft and the intruder, as well as the chosen action.

Table 1. Aircraft action rewards

condition on distance d and banking angle ω	reward value
$d \leq$ NMAC	-100
$d >$ NMAC, $\omega \neq 0$	-1
$d >$ NMAC, $\omega = 0$	0
$d >$ NMAC, at destination	100

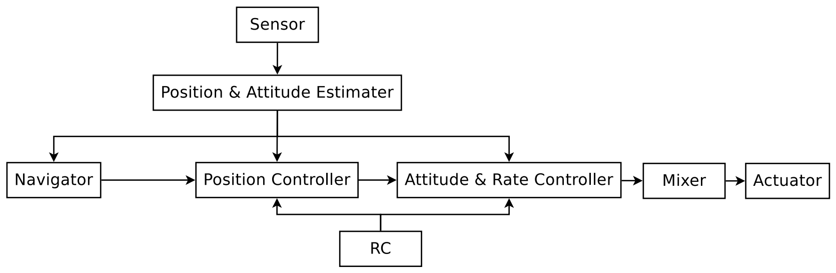


Fig. 3. Overview of the PX4 flight stack

Since we only consider the 2-D space, we use NMAC to represent the minimum horizontal distance between the two aircraft so that they are considered to be near collision. We give the large penalty of -100 to each action performed when the two aircraft are closer than the NMAC distance, as shown in Table 1. When the aircraft are enough far away, we give a penalty of -1 when the action ω of our own aircraft is not zero, i.e., the aircraft changes direction, and of 0 if it goes straight. This allows us to reduce non-necessary actions taken by our own aircraft. Finally, when our aircraft reaches its destination, a reward of 100 is given to each action, as long as the intruder is not closer than NMAC.

Regarding the discount factor for the expected reward (cf. Eqns. (3) and (4)), we use $\gamma = 0.95$ as discount factor. This values ensures that it is better to immediately avoid the intruder than eventually reach the destination; however, reaching the destination is still taken into account rather strongly (cf. [19]).

4 Experimental Evaluation

We based our experiments on the implementation of MCVI [1] publicly available in `https://github.com/bhy/mcvi`: it takes as input a POMDP model (coded in C++) and produces as output a policy graph file. We have created our own POMDP model of the collision avoidance problem by extending functions of the MCVI source code and generated the corresponding collision avoidance policy graph. We then extended the PX4 flight stack by adding a collision avoidance module implementing the policy synthesized by MCVI. The performance of the collision avoidance module has then been evaluated on two scenarios.

PX4 is an open source flight control platform that is supported by the Pixhawk flight control board, an independent open-hardware project for drones and other unmanned vehicles developed by Computer Vision and Geometry Laboratory at ETH Zurich. Due to the clear structure of PX4 flight stack, shown in Fig. 3. it is easy for developers to add their own flight mode on Pixhawk hardware board with PX4 software, or in the jMAVSim tool that is a simulator for PX4-controlled unmanned aircraft. We integrated the collision avoidance module we developed into the PX4 flight stack by adding it in the position controller, the PX4 component responsible for the flight trajectory.

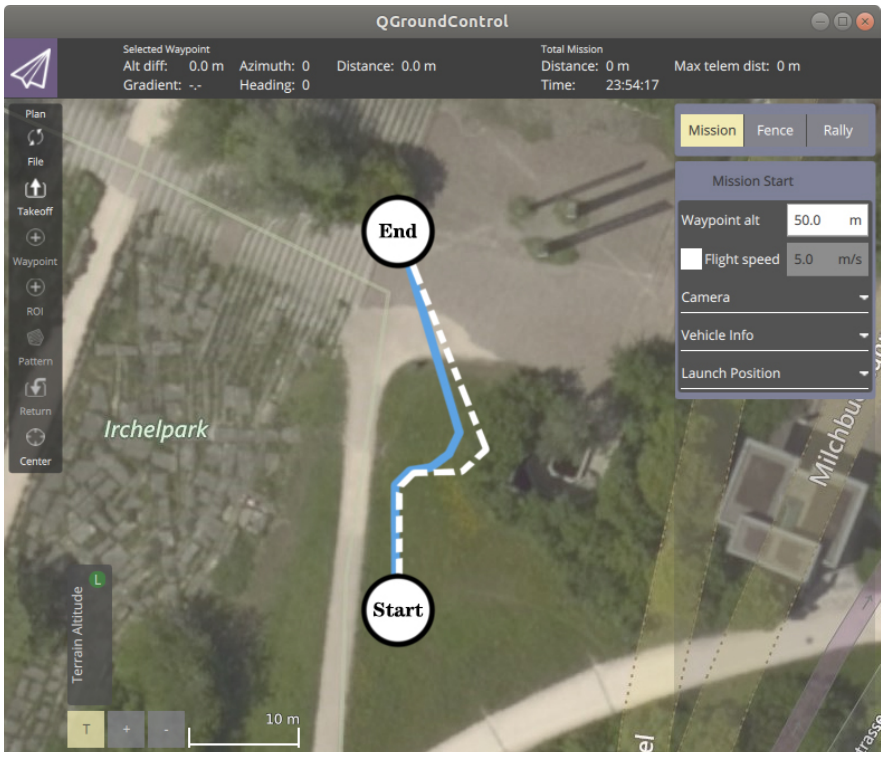
4.1 Experiments Configuration

In the experiments, since our Pixhawk drone has no sensors for detecting an intruder, we have created synthetic encounter situations where our own aircraft detects an intruder while performing a flight mission, so there may be a collision. The flight mission of our own aircraft is to reach the destination placed at 30 meters North of the initial position. The intruder aircraft starts from our destination and its goal is to reach our initial position. When an encounter occurs, the intruder aircraft is designed to fly according to two predefined policies, introduced later in Sections 4.2 and 4.3. By following the sensor modeling given in Section 3.2, our own aircraft makes a detection every second, and the detection radius is set to 10 meters. That is, our own aircraft gets one observation per second about the presence and position of an intruder aircraft; it can be sensed when the distance between the two aircraft is below 10 meters. After getting the observation about the intruder aircraft, our own aircraft will react according to the two different collision avoidance methods introduced below.

Simple rule-based collision avoidance scenario. In the simple rule-based collision avoidance scenario, our own aircraft reacts to observations as follows: if L or R is observed (i.e., the intruder aircraft is in the detection range on the left or right of our aircraft), then bank to the opposite direction by $\frac{\pi}{6}$ and fly forward by 1 meter; otherwise N is observed, so fly towards to destination.

MCVI-based collision avoidance scenario. In the collision avoidance scenario using the MCVI synthesized policy, our own aircraft reacts to observations as follows: as soon as an intruder aircraft is detected (observations L and R), bank according to the policy as long as the intruder is detected or the graph terminal node is reached. If the intruder aircraft is not detected or the policy graph terminal node is reached, fly towards the destination. The banking angle and step length are the same as those of the simple rule-based method. Note that after the controller stops to follow the policy, if the intruder aircraft is detected again, the controller restarts to follow the policy.

Experiments setting. Finally, we implemented the above two collision avoidance methods in the PX4 platform. Supported by the Software In the Loop (SITL) simulation of the PX4 platform, we used jMAVSimto simulate our experiments. jMAVSim supports the multirotor simulation and allows us to fly drones running PX4 in a synthetic world. To compare the effectiveness of two collision avoidance



Selected Waypoint
Alt diff: 0.0 m Azimuth: 0 Distance: 0.0 m
Gradient: -.- Heading: 0

Total Mission
Distance: 0 m Max telem dist: 0 m
Time: 23:54:17

Waypoint alt 50.0 m
Flight speed 5.0 m/s

Fig. 4. One flight simulation for the opposite flight experiment. Blue line: MCVI flight path; white line: simple flight path

methods, we recorded the following measure data (with the meter as the length unit) to indicate the performance: the number *actNum* of banking performed to reach the destination, the minimum distance *minDis* between our own aircraft and the intruder aircraft, the maximum deviation distance *maxDvtDis* from the designed mission flight path, and the increased flight distance *incFltDis* from the original mission flight distance.

4.2 Simulated Intruder's Opposite Flight Experiment

In the first simulated experiment, we assume that the intruder aircraft starts from the destination of our own aircraft (viz. 30 meters North of the starting place of own aircraft) and flies straight towards the starting place of our own aircraft. The flight paths of the mission while avoiding a collision are shown in Fig. 4 and the average experimental results are summarized in Table 2.

Based on the jMAVSim simulation, depicted in Fig. 4, we can see the flight path of own aircraft in the simulated map, where the blue solid line represents the flight path induced by the MCVI method while the white dashed line tracks the flight path given by the simple rule-based method. It can be intuitively seen

Table 2. Opposite flight results; in bold, the best results

	rule-based	MCVI-based
actNum	**4**	5
minDis	2.51	**2.95**
maxDvtDis	9.69	**5.88**
incFltDis	14.83	**9.67**

Table 3. Random interfering flight results; in bold, the best results

	average value		median value	
	rule-based	MCVI-based	rule-based	MCVI-based
actNum	**3.82**	4.96	**2.5**	4
minDis	5.54	**6.06**	5.88	**6.04**
maxDvtDis	8.85	**5.40**	7.50	**5.60**
incFltDis	9.14	**7.10**	**6.67**	6.87

from the figure that the MCVI method can reduce the useless avoidance distance of our own aircraft, and avoid the collision risk more effectively.

Table 2 gives more details about the flight paths: we note that although the logic generated by MCVI based on the POMDP model takes 5 actions instead of 4 to reach the destination while avoiding the intruder aircraft, it does not results in an increase of the flight distance. On the contrary, as shown in Fig. 4 by the MCVI method's blue solid line being closer to the straight line between the terminal points than the simple method's white dashed line, the flight distance is increased by just 9.67 meters, about 5 meters less than the 14.83 meters of the simple method. Furthermore, the MCVI-based approach increases the minimum distance (2.95 vs. 2.51 meters) from the intruder aircraft while deviating less (5.88 vs. 9.69 meters) from the optimal path without intruders than the simple method; this guarantees a higher safety level while decreasing the maximum deviation distance from the new flight path to the original mission flight path.

4.3 Simulated Intruder's Random Interfering Flight Experiment

In the second simulated experiment, we assume that the intruder aircraft takes off from the midway between the start place and destination of our own aircraft (viz. 15 meters North of the starting place of our own aircraft). This time the intruder aircraft chooses randomly an action between bank-left, go-forward and bank-right, i.e., each one with probability $\frac{1}{3}$. Since the flight behavior of the intruder aircraft in this experiment is randomized, we repeated the simulation 50 times and then considered the average outcomes of the two scenarios against the same intruder's random choices, which can be reproduced by fixing the same random seed for the two scenarios.

The statistics for this experiment are presented in Table 3; as we can see, we have a behavior similar to the opposite flight experiment: although the logic generated by MCVI on the POMDP model banks on average one time more to avoid the intruder than the simple rule-based method, it is able to reduce the increased flight distance. Furthermore, it increases the minimum distance between the two aircraft, which implies that the MCVI method gives a higher safety level when dealing with the random behavior of the intruder aircraft. It is particularly noteworthy that when compared with the simple rule-based method, MCVI-based collision avoidance is able to save about 40% of the maximum deviation distance between the actual flight path and the original mission flight path, which is a straight line between the designed takeoff and landing coordinates.

We now give some more details about the safety level achieved by the two collision avoidance scenarios introduced in Section 4.1. Out of the 50 simulations for each of the two scenarios, both scenarios ensure that there is no collision since in none of the simulations the minimum distance was below 2 meters, the value we set for NMAC; the minimum distance has been at least 2.5 meters. There have been few simulations where the distance was about 3 meters and the majority above 4 meters, with the MCVI-based scenario usually keeping a larger distance. This is reflected also by the average *minDis* values reported in Table 3, where the rule-based *minDis* value is lower than the one for MCVI, which means that the former control policy got closer to the intruder than the latter one.

From these experiments, we obtain that the MCVI-based collision avoidance controller provides a better safety level, since it keeps a larger distance between our aircraft and the intruder than the rule-based controller. This comes at the cost of banking the aircraft few more times, which still allows for reducing the increased flown distance. The flown distance is also an aspect to be considered for drones and other unmanned aircraft, since the increased flight distance reduces the actual operative range of the aircraft, given the limited amount of energy stored in the batteries that can be used to power the aircraft.

While the experiments give a good indication about the safety level provided by the MCVI-based collision avoidance controller, there is no guarantee that it always avoids collisions. We leave the formal analysis of its safety as future work.

4.4 Actual Pixhawk Drone Flight Experiment

In the third experiment, we have taken the PX4 MCVI-based collision avoidance module we developed and uploaded it on a real Pixhawk drone we assembled ourselves. The flight mission is the same as for the previous experiments. The Pixhawk drone communicates to its ground station several data about the position and flight attitude. Since the Pixhawk drone has no sensors for detecting intruders, we used the ground station to inject in the drone's PX4 control system the synthetic intruder from the opposite flight experiment presented in Section 4.2. We show in Fig. 5 the actual flight path flown by our drone as registered by the ground control station and superimposed on the map.

As we can see, the MCVI-based collision avoidance module works well not only in a synthetic environment, but also in the real world. This suggests that the policy synthesis for specifically crafted POMDPs can improve the safety of unmanned aircraft equipped with a collision avoidance module basing its decisions on the synthesized policy.

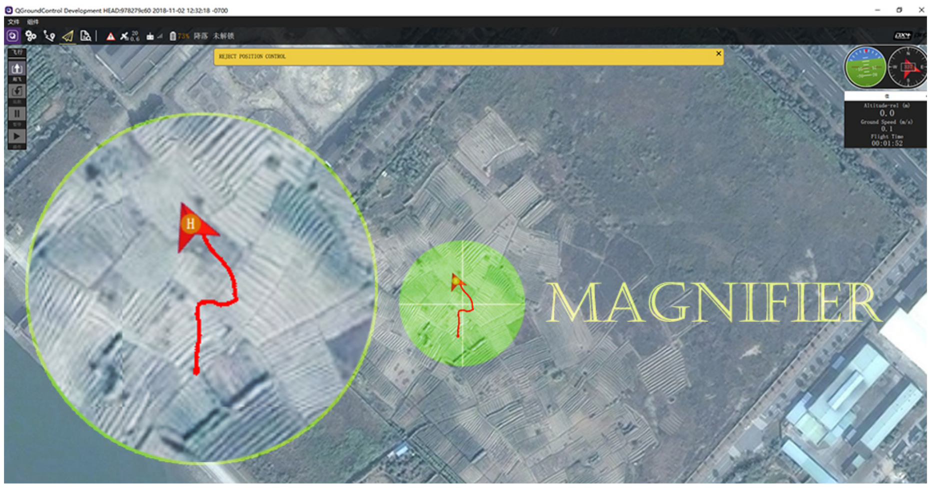


Fig. 5. The actual Pixhawk drone flight path registered by the ground control station

5 Conclusion

We have constructed a collision avoidance system working with limited information, which also tried to save flying resources by means of reducing the change of the original flight path. Our approach used a POMDP to model the collision avoidance system with only the destination information of our own aircraft and rough direction information of the intruder and then generated the collision resolution logic that maximizes the expected sum of rewards of selected actions. We have implemented the collision avoidance module and embedded it into the PX4 flight control platform that can control real unmanned aircraft systems. The effectiveness of our system is witnessed by the experimental evaluation in both a simulated environment and a real world scenario using a Pixhawk drone.

As future work, we consider to study the scalability of the MCVI approach to more detailed flight scenarios, such as the ones with the 3D environment, different aircraft speeds, more accurate observations given by better sensors, and so on. Also, we plan to apply machine learning algorithms to improve the efficiency of the algorithm for searching an optimal policy. Moreover, we also consider the use of rigorous methods, such as model checking, to ensure the correctness of the collision avoidance system.

Acknowledgments. The authors are very thankful to Xuechao Sun and Junwen Li for helpful discussion and the assembly of the Pixhawk drone. This work has been supported by the Guangdong Science and Technology Department (grant no. 2018B010107004), the Natural Science Foundation of Guangdong Province (grant no. 2019A1515011689), and the National Natural Science Foundation of China (grant nos. 61761136011, 61532019, 61836005).

References

1. Bai, H., Hsu, D., Kochenderfer, M.J., Lee, W.S.: Unmanned aircraft collision avoidance using continuous-state POMDPs. In: Durrant-Whyte, H.F., Roy, N., Abbeel, P. (eds.) Robotics: Science and Systems VII, University of Southern California, Los Angeles, CA, USA, June 27-30, 2011 (2011). https://doi.org/10.15607/RSS.2011.VII.001, `http://www.roboticsproceedings.org/rss07/p01.html`
2. Bai, H., Hsu, D., Lee, W., Vien, N.: Monte Carlo value iteration for continuous-state POMDPs. Algorithmic Foundations of Robotics IX **68**, 175–191 (2010). https://doi.org/10.1007/978-3-642-17452-0_11
3. Hauskrecht, M.: Value-function approximations for partially observable Markov decision processes. Journal of Artificial Intelligence Research **13**, 33–94 (2000). https://doi.org/10.1613/jair.678
4. Hazeghi, K., Puterman, M.: Markov decision processes: Discrete stochastic dynamic programming. Journal of the American Statistical Association **90**, 392 (1995). https://doi.org/10.2307/2291177
5. Julian, K.D., Lopez, J., Brush, J.S., Owen, M.P., Kochenderfer, M.J.: Policy compression for aircraft collision avoidance systems. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC). pp. 1–10 (2016)
6. Kaelbling, L., Littman, M., Cassandra, A.: Planning and acting in partially observable stochastic domains. Artificial Intelligence **101**, 99–134 (1998). https://doi.org/10.1016/S0004-3702(98)00023-X
7. Kamen, E.W., Su, J.K.: Introduction to Optimal Estimation (1999). https://doi.org/10.1007/978-1-4471-0417-9_3
8. Kochenderfer, M., Chryssanthacopoulos, J.: Robust airborne collision avoidance through dynamic programming (2011)
9. Kochenderfer, M., Holland, J., Chryssanthacopoulos, J.: Next generation airborne collision avoidance system. Lincoln Laboratory Journal **19**, 17–33 (2012)
10. Kuchar, J., Drumm, A.: The traffic alert and collision avoidance system. Lincoln Laboratory Journal **16**(2), 277–296 (2007)
11. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: Robotics: Science and Systems IV (2008)
12. Manfredi, G., Jestin, Y.: An introduction to ACAS Xu and the challenges ahead. In: DASC. pp. 1–9 (2016). https://doi.org/10.1109/DASC.2016.7778055
13. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: An anytime algorithm for POMDPs. In: IJCAI. pp. 1025–1032 (2003)
14. Porta, J., Vlassis, N., Spaan, M., Poupart, P.: Point-based value iteration for continuous POMDPs. Journal of Machine Learning Research **7**, 2329–2367 (2006)
15. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. No. 594 in Wiley Series in Probability and Statistics, John Wiley & Sons, Inc. (2005)
16. Shani, G., Pineau, J., Kaplow, R.: A survey of point-based POMDP solvers. Auton Agent Multi-Agent Syst **27**, 1–51 (2013). https://doi.org/10.1007/s10458-012-9200-2
17. Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation. In: UAI. pp. 542–549 (2005)
18. Sondik, E.J.: The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. Oper. Res. **26**(2), 282–304

(1978). https://doi.org/10.1287/opre.26.2.282, `https://doi.org/10.1287/opre.26.2.282`
19. Sutton, R.S., Barto, A.G.: Reinforcement learning - an introduction. Adaptive computation and machine learning, MIT Press (1998), `http://www.worldcat.org/oclc/37293240`
20. Temizer, S., Kochenderfer, M., Kaelbling, L., Lozano-Perez, T., Kuchar, J.: Collision avoidance for unmanned aircraft using Markov decision processes. In: AIAA Guidance, Navigation, and Control Conference (2010). https://doi.org/10.2514/6.2010-8040
21. Thrun, S.: Monte Carlo POMDPs. In: NIPS. pp. 1064–1070 (1999)