This is a repository copy of *Collision Avoidance of Two Autonomous Quadcopters*.

Article:

Collision Avoidance of Two Autonomous Quadcopters

Michalis Smyrnakis[1], Jonathan M. Aitken[1], and Sandor M. Veres[1] [*]

Department of Automatic Control and Systems Engineering, University of Sheffield
m.smyrnakis, jonathan.aitken, s.veres@sheffield.ac.uk

Abstract. Traffic collision avoidance systems (TCAS) are used in order to avoid incidences of mid-air collisions between aircraft. We present a game-theoretic approach of a TCAS designed for autonomous unmanned aerial vehicles (UAVs). A variant of the canonical example of game-theoretic learning, fictitious play, is used as a coordination mechanism between the UAVs, that should choose between the alternative altitudes to fly and avoid collision. We present the implementation results of the proposed coordination mechanism in two quad-copters flying in opposite directions.

1 Introduction

In this paper we present a Traffic Collision and Avoidance System (TCAS), motivated by interaction between two quadcopters. Crucially this novel approach does not require any communication between the agents controlling the copters, cooperation is generated by a game-theoretic approach using physical observations and predictions of the action of the other vehicle, obtained solely by visual monitoring of the environment.

Unmanned aerial vehicles (UAVs) present a unique challenge in robots. A ground robot can hold position indefinitely, simply by stopping. For flying vehicles this is a very different preposition. Even though a quadcopter can hover in the air this consumes battery power, so prolonged processing is not an efficient solution. As technology develops UAVs will be used for a wide range of different tasks including inspection work, for example hard to access structures [13]. The focus will be on operating the quadcopters autonomously to minimise human workload.

Naturally, as part of this task, the quadcopters will need to avoid each other as they move around in the environment. Communication between each quadcopter provides a potential solution, but creates an exponentially increasing overhead as the number of quadcopters increases, and vastly increases power considerations on-board. By reducing communication overhead, we can conserve power to lengthen the time spent on the task. Therefore each should be able to take decisions itself based on what it observes - game theory provides the potential for this by allowing each agent to predict the actions of others without requiring communication.

TCAS [16] is a system designed to prevent collision between aircraft. Should two aircraft be on a collision course, a warning is given to one pilot with a recommendation to climb or descend, and an equivalent warning with an opposite instruction given to the

other pilot. This is an efficient system enabling collision avoidance, but requires communication of information between both parties, if contradicting information is given, for example in the case of the Überlingen mid air collision where one aircraft acted on TCAS and the other on instructions from a ground controller, then the probability of an accident is significantly increased [1].

In this work we present a very different solution based on the game theoretic approach. Whilst similar techniques use off-line learning [5] to establish behaviours, this approach takes decisions on-line, requiring no a priori training, no communication between the players and presents a system which generates agreement online; allowing the process of cooperation itself to be observed.

2 Problem definition

In this section we define the TCAS problem as a symmetric game where the UAVs are the players of the game and the altitudes that they can fly are the actions available to each player. We start by briefly providing some basic game-theoretic definitions that we will use throughout this paper and then we show how the TCAS problem can be cast as a game.

2.1 Game-theoretic definitions

The strategic game used in this paper has a set of players, $i = 1, \ldots \mathscr{I}$, who choose their actions from their actions sets S^i simultaneously. We will often write $-i$ in order to refer to all players but i. Each player i chooses his action, $s^i \in S^i$. We will refer to the actions that all the players chose in a game as joint action $s = (s^1, \ldots, s^{\mathscr{I}}) \in S = \times_{i=1}^{\mathscr{I}} S^i$. The rewards that players gain is a mapping from the joint action space to the real numbers, $r^i(s) : S \to \mathbb{R}$.

The players of a game choose their actions using mixed strategies. A mixed strategy of a player i is defined as $\sigma^i \in \Delta^i$, where Δ^i is the set of all probability distributions over the action space S^i of player i. Similarly to actions, a joint mixed strategy $\sigma = (\sigma^1, \ldots \sigma^{\mathscr{I}}) \in \Delta = \times_{i=1}^{\mathscr{I}} \Delta^i$, is the set of all the mixed strategies players use in a game. Special cases of mixed strategies are pure strategies. A player that chooses an action with probability one chooses his actions using a pure strategy.

Players can use any decision rule in order to choose their mixed strategy in the game. A common decision rule in game-theoretic literature is Best Response (*BR*). When players are using *BR*, they choose to play the mixed strategy that maximises their expected reward. More formally we can write:

$$\hat{\sigma}^i_{BR}(\sigma^{-i}) = \underset{\sigma^i \in \Delta^i}{\operatorname{argmax}} \quad r^i(\sigma^i, \sigma^{-i}) \tag{1}$$

Nash in [10] proved that every game has at least one equilibrium which is a fixed point in the best response correspondence. A joint mixed strategy $\hat{\sigma}$ is a Nash equilibrium iff

$$r^i(\hat{\sigma}^i, \hat{\sigma}^{-i}) \geq r^i(\sigma^i, \hat{\sigma}^{-i}) \qquad \forall \sigma^i \in \Delta^i, i = 1, \ldots \mathscr{I}. \tag{2}$$

Equation (2) implies that no players can increase their reward if they unilaterally change their strategy. A special case of Nash equilibria are pure Nash equilibria. A joint action, pure strategy, where players cannot do better by unilaterally changing their actions, is called a pure strategy Nash equilibrium.

2.2 The TCAS problem as a symmetric game

The goal of a TCAS for aircraft is to avoid collision. Since all the UAVs have the same goal, it is natural to use coordination games in order to describe the task. In coordination games all players share the same reward that depends to their joint action. Thus the reward function of the game that represent a TCAS is defined as:

$$r(s^i, s^{-i}) = \begin{cases} a \ if s^i \neq s^j \ \forall j \in -i \\ 0 \ otherwise \end{cases} \tag{3}$$

If we define the reward function of the UAVs using (3), all the joint actions that are not collision free, and therefore all the cases which the UAVs fail to coordinate, are penalised and only the joint actions that solve the TCA problem produce some reward for the agents. Nevertheless there are more than one possible joint actions that lead to a positive reward, and consequently more than one pure Nash equilibria. This is due to the fact that the UAVs don't have a preference of the altitude that they want to fly at and therefore (3) defines a symmetric game. In symmetric games someone can change the identities of the agents without changing the reward of the game, i.e in a two player game $r(s^i, s^{-i}) = r(s^{-i}, s^i)$. Thus a coordination mechanism is needed in order for the UAVs to perform their assigned task autonomously.

3 Coordination Mechanism

3.1 Fictitious play

Game-theoretic learning algorithms are suitable as a coordination mechanism between the UAVs because of their small communication cost and reliability. In fictitious play all agents choose the actions to maximise their expected reward, which is based on their estimates of their opponent's strategies, using (1). These estimates are updated in each iteration of the game based on the observations of the other agents' actions.

In order to compute (1) a agent i needs to know the strategies of all the other agents. But agent i can only observe the actions of his opponents and not their strategies. Under the assumption that agents use a unique strategy throughout the game, opponents' strategies can be estimated using a multinomial distribution, whose parameters can be updated using the observed actions of the other agents.

In particular, before the initial iteration of the game, each agent maintains some random non-negative weights, κ_0 for each of his opponents' actions. Then in each iteration of the game each agent i updates his weights for the agent $j \in -i$ as follows:

$$\kappa_t^j(s^j) = \kappa_{t-1}^j(s^j) + I_{s_t^j = s^j}(s^j), \ t = 0, 1, \ldots, \tag{4}$$

where s_t^j is the observation at time t and $I_{s_t^j=s^j}(s^j) = \begin{cases} 1 \ if s_t^j = s^j \\ 0 \ otherwise. \end{cases}$ The mixed strategy of opponent j is estimated then as:

$$\sigma_t^j(s^j) = \frac{\kappa_t^j(s^j)}{\Sigma_{s^j \in S^j} \kappa_t^j(s^j)}$$
$$= (1 - \tfrac{1}{t})\sigma_{t-1}^j(s^j) + \tfrac{1}{t}I_{s_t^j=s^j} \qquad (5)$$

Fictitious play converges to the Nash equilibrium in various categories of games, including 2×2 games with generic payoffs [7], zero sum games [12], games that can be solved using iterative dominance [9] and potential games [8]. Nevertheless there are also games where fictitious play does not converge, and becomes trapped in limit-cycles. An example of such a game is Shapley's game [14]. In addition in symmetric games, that we are interested in, agents may choose a mixed strategy that is a Nash equilibrium but they will actually become trapped in a cycle where their rewards will be not maximised [4]. In the next section we introduce a new version of estimating opponents strategies $\sigma_t^j(s^j)$ using the Boltzmann formula.

3.2 Extended Kalman Filter Fictitious Play

A variant of fictitious play, which addresses the problem of the classic algorithm in symmetric games, is the EKF-based (Extended Kalman Filter based) fictitious play [15].

In order to overcome difficulties that arise from the fuse of probability distributions, agents predict the unconstrained propensities [17] that their opponents have for their actions. A crucial assumption of this algorithm is that, throughout the game, agents adapt their propensities to choose an action and based on these propensities they update their strategies and choose their actions [17]. Figure 1 depicts how propensities of agent
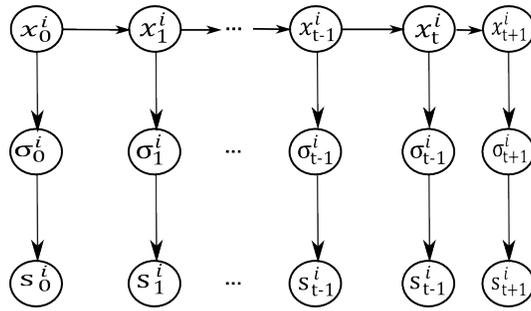

Fig. 1. Propensities propagation throughout the game. The Propensity at time t depends only to the propensity at time $t - 1$. Moreover the strategy of a agent at time t depends only in the propensity of the same iteration, and the action that a agent chooses depends only at his strategy at the same iteration as well.

i change through the iterations of the game and how they are related to strategies and actions. Based on the fact that agents have no prior knowledge about their opponents'

strategies, an autoregressive model is chosen in order to propagate the propensities [17]. In addition, inspired from the sigmoid functions that are used in neural networks to connect the weights and the observations, a Boltzman formula (7) is used to relate the propensities with opponents' strategies [18]. Thus the following state space model is used to describe the fictitious play process:

$$x_t^j = x_{t-1}^j + \xi_{t-1}^j$$
$$I_{s_t^j = s^j}(s^j) = h(x_t^j) + \zeta_t^j, \ j \in \{1, ..., \mathscr{I}\} \backslash \{i\} \qquad (6)$$

where the components of h are

$$h_k(x) = \frac{exp(x_k/\tau)}{\sum_k exp(x_k/\tau)}, \ k \in S^j \qquad (7)$$

$\xi_{t-1}^j \sim N(0, \Xi)$, is the noise of the propensity process which comprises the internal states and $\zeta_t \sim N(0, Z)$ is the error of the observation of propensities by the indicator function with zero mean and covariance matrix Z, which occurs because we observe a discrete 0-1 process, such as the best response in (1) through the continuous Boltzmann formula $h(\cdot)$ in (7) in which τ is a "temperature parameter".

agent i then evaluates his opponents strategies using his estimates as:

$$\sigma_{tk}^j = \frac{exp(\bar{x}_{kt}^j/\tau)}{\sum_k exp(\bar{x}_{kt}^j/\tau)}. \qquad (8)$$

where $\bar{x}_{kt}^j$ is agent i's prediction of the propensities of opponent j to choose action k based the state equations in (6) and using observations up to time $t - 1$. Agent i then uses the estimates of its opponents strategies in (8) to choose an action by best response (1) evaluation. The EKF estimation is done by any standard textbook procedure [19].

Table 1 summarises the fictitious play algorithm when EKF is used to predict opponents strategies.

4   Experimental evaluation

The scenario presented in this paper requires autonomous control of a pair of quadcopters, in this case the model selected is the Parrot AR.Drone 2.0. The quadcopter has one forward-facing and one downward-facing camera. This camera will be used for location of the parter quadcopter, and for navigation to maintain position within the room.

4.1   Quadcopter Driver

The autonomy package[1] for the AR.Drone developed for the Robot Operating System [11] provides the main harness for communicating with the quadcopter, and provides an interface to the quadcopter navigation data.

---

[1] `https://github.com/AutonomyLab/ardroneautonomy`

1. At time t agent i maintains estimates of its opponent's propensities up to time $t-1$, $\hat{x}^j_{t-1}$,, with covariance P^j_{t-1} of its distribution.
2. Agent i predicts its estimates about its opponents' propensities to $\bar{x}_t k^j$, $j \in \{1,...,\mathscr{I}\}\backslash\{i\}$, $k \in S^j$ using the state equations in (6).
3. Based on the propensities in 2 each agent updates its beliefs about its opponents' strategies using $\sigma^j_{tk} = \frac{exp(\bar{x}^j_{tk}/\tau)}{\sum_k exp(\bar{x}^j_{tk}/\tau)}$, $k \in S^j$.
4. Agent i chooses an action based on the beliefs in 3 and applies best response decision rule.
5. The agent i observes its opponents' actions s^j_{tk}, $j \in \{1,...,\mathscr{I}\}\backslash\{i\}$.
6. The agent update its estimates of all of its opponents' propensities using extended Kalman Filtering to obtain $\hat{x}^j_t$, $j \in \{1,...,\mathscr{I}\}\backslash\{i\}$.

Table 1. EKF based fictitious play algorithm.

4.2 Maintaining Quadcopter Position

In order to maintain position this paper utilises the Technical University of Munich (TUM) AR.Drone package [2,3]. This algorithm provides a monocular Simultaneous Location and Mapping (SLAM) based on the Parallel Tracking and Mapping (PTAM) algorithm [6]. This algorithm uses a collection of keypoints identified within a stream of images to track the position of a camera relative to a scene. Typically these keypoints are object edges or corners which helps build a picture of the environment as shown in Figure 2.

By coupling the tracking of these keypoints to information coming from the onboard sensors, The TUM AR.Drone package can produce a 3D map of the world locating these keypoints in space by tracking them between frames, using a pair of initialisation frames using monocular SLAM. Once these keypoints are known, the quadcopter position can be fixed in space, relative to the keypoints. This provides a very robust method for positioning the quadcopter. Incorporated within the package is an autopilot function that takes these 3D coordinates as demand positions for the quadcopter.

By incorporating the autopilot and the SLAM routine each quadcopter can be initialised into position to begin the experiment. Appropriate commands can then be passed to the autopilot as a result of the decision making process.

4.3 Quadcopter Recognition

Quadcopter recognition is performed using inbuilt functionality of the Parrot AR.Drone 2.0. The front facing cameras processes images and has been configured to locate a colour pattern, blue-orange-blue. As part of the navigation data, the quadcopter reports the presence of the colour pattern in the current field of view. Each quadcopter has the colour pattern affixed to their front-facing side, adjacent to the forward-facing camera. Therefore since each quadcopter knows its own altitude, by observing or not observing the other quadcopter it can infer its action.
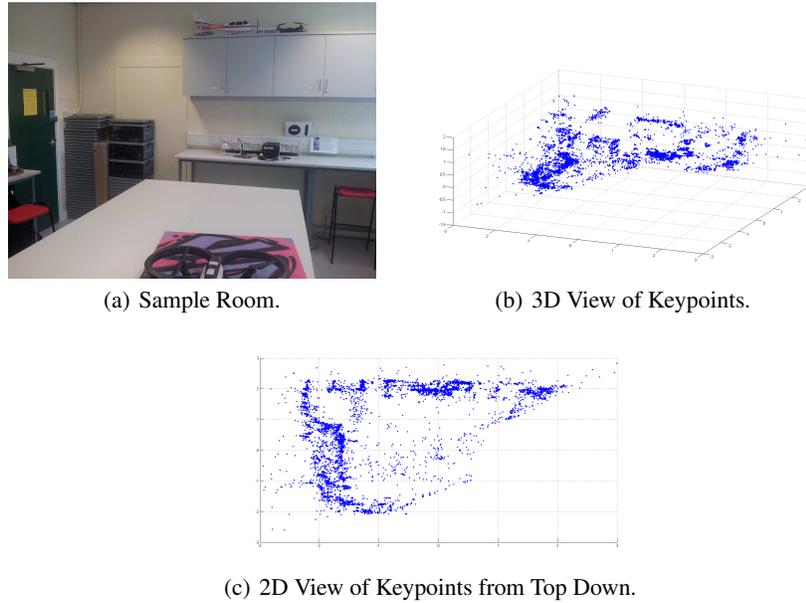
(a) Sample Room.

(b) 3D View of Keypoints.

(c) 2D View of Keypoints from Top Down.

Fig. 2. Keypoints Identified within a Room Showing Distribution Across a Scene.

4.4 Mission Management

A mission management node contains the intelligence for the mission. Figure 3 shows the connection into the complete ROS graph for the system. The mission management node takes decisions based on the observed environment, therefore interfaces to the navigation data to receive tag information, and information on the predicted pose to understand attitude and orientation provided by the SLAM system outlined in Section 4.2. At the beginning of each experiment both quadcopters have their SLAM systems initialised separately to ensure that they do not capture the other as part of the background. The mission node controls takeoff and landing directly through connected topics within the driver, but uses the TUM AR.Drone command channel to communicate with the autopilot to adjust quadcopter position.

The mission management node controls the safe execution of the manoeuvre. The quadcopter starts by observing whether the other player is visible. Every 8s a decision is taken based on the algorithm outlined in Section 4.5, the result of this decision is passed to the autopilot to change the quadcopter position appropriately; either rising 1m if in the low position, descending 1m if in the upper position or holding station. During this 8s window, if the quadcopter observers the absence of the other player for a continuous 4s period it begins the passing procedure by travelling forwards, adjusting the longitudinal position supplied to the autopilot moving it forwards whilst maintaining the current altitude.
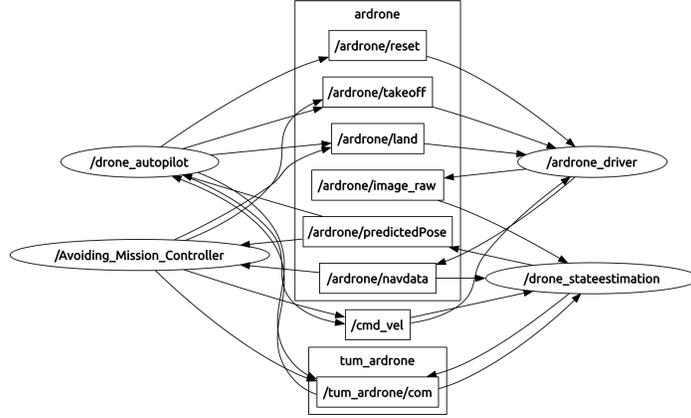
Fig. 3. Underlying ROS Connection Graph.

4.5 A symmetric Game and the Decision Making Process

In our implementation we allowed the quadcopters to fly in opposite directions using two altitudes, high and low altitude hereafter. When (3) is used, players share the rewards of the game which is presented in Table 2.

	High altitude	Low altitude
Low altitude	1	0
High altitude	0	1

Table 2. The reward that the two quadcopters share.

We arbitrarily chose $a = 1$, but the presented results will be valid for any other value of constant a.

The rest of this section provides details about the implementation of EKF fictitious play in the proposed TCAS framework. We consider the case where the above mentioned symmetric game has been played for $t - 1$ iterations and the players use EKf fictitious play to update their estimations of their opponent's strategy.

We will present the updates of only one robot since the updates of the other quadcopter will be identical. In our implementation we used the following parameters for Ξ and Z and τ:

$$\Xi = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix}, \quad Z = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.3 \end{bmatrix}, \quad \tau = 2$$

From the previous iterations of the game the UAV will have an updated estimation of $p(x_{t-1}|s_{1:t-1})$ in the light of joint action s_{t-1}. Therefore since it will approximate $p(x_{t-1}|a_{1:t-1})$, using $\hat{x}_{t-1}$, the variance of this approximation will be P_{t-1}. For the rest of this paper the i_{th} element of a vector y will be denoted as $y[i]$ and the $(i,j)_{th}$ element of a matrix y as $y[i][j]$. Without loss of generality we assume that $\hat{x}[1]$ is the approximation

of the propensity the other UAV has to fly in high altitude and $\hat{x}[2]$ is the approximation of the propensity the other UAV has to fly in low altitude. Consequently $P[1][1]$ and $P[2][2]$ is the variance of the approximation of the propensity the other UAV has to fly in high and low altitude altitude respectively. The covariance between the two actions propensities is estimated in $P[1][2] = P[2][1]$.

Then at time t before it makes a decision it needs to approximate $p(x_t|s_{1:t-1})$. The approximation of the propensities and its variance is:

$$\hat{x}_t^-[1] = \hat{x}_{t-1}[1]$$
$$\hat{x}_t^-[2] = \hat{x}_{t-1}[2]$$

$$P_t^-[1][1] = P_{t-1}[1][1] + \Xi[1][1] + d$$
$$P_t^-[1][2] = P_{t-1}[1][2]$$
$$P_t^-[2][1] = P_{t-1}[2][1]$$
$$P_t^-[2][2] = P_{t-1}[2][2] + \Xi[2][2] + d$$

where $d = 0.1 + 0.0001|\mathbf{n}|$, $\mathbf{n} \sim N(0, 0.0001)$ The estimation of of the other UAV's strategy is

$$\sigma_t[1] = \frac{\exp(\hat{x}_t^-[1]/\tau)}{\exp(\hat{x}_t^-[1]/\tau) + \exp(\hat{x}_t^-[2]/\tau)}$$
$$\sigma_t[2] = \frac{\exp(\hat{x}_t^-[2]/\tau)}{\exp(\hat{x}_t^-[1]/\tau) + \exp(\hat{x}_t^-[2]/\tau)}$$

Then based of this estimation the UAV make a decision about the altitude that it will choose. Because of the symmetry of the game and the fact that Best responce (1), the UAV chooses the action which has smaller to be selected by the other UAV, and thus $s_t^i = argmin_{j=1,2}\sigma_t[j]$.

After both UAVs choosing their actions then they should update their estimations by estimating $p(x_t|s_{1:t})$. Without loss of generality we will assume that the UAV which we estimate its strategy chose to fly in high altitude, action 1. The approximation of $p(x_t|s_{1:t})$ and its covariance, based on the update step of the EKF process are as follows:

The difference between the estimation and the observed action is evaluated as:

$$v_t[1] = \sigma_t[2]$$
$$v_t[2] = -\sigma_t[2]$$

The Jacobian of the transformation function h is computed as:

$$H_t = \begin{bmatrix} \sigma_t[1]\sigma_t[2] & -\sigma_t[1]\sigma_t[2] \\ -\sigma_t[1]\sigma_t[2] & \sigma_t[1]\sigma_t[2] \end{bmatrix}$$

The variance of the approximation process is

$$S_t = c_1 \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} + \begin{bmatrix} (1/t) & 0 \\ 0 & (1/t) \end{bmatrix}$$

where $c_1 = \sigma_t[1]\sigma_t[2](P_t^-[1][1] + P_t^-[2][2]) - 2P_t^-[1][2])$. The Kalman gain is computed as:

$$K_t = c_2 \begin{bmatrix} P[1][1] - P[1][2] & P[1][2] - P[1][1] \\ P[1][2] - P[2][2] & P[2][2] - P[1][2] \end{bmatrix}$$

where $c_2 = \frac{1}{tc_1} \frac{1}{P_t^-[1][1]+P_t^-[2][2])-2P_t^-[1][2]} \frac{1}{c_1(1+c_1)^2}$. Finally the updates for $\hat{x}_t$ and P_t are:

$$\hat{x}_t[1] = \hat{x}_t^-[1] + 2(P[1][1] - P[1][2])\sigma_t[2]c_2$$
$$\hat{x}_t[2] = \hat{x}_t^-[2] - 2(P[1][1] - P[1][2])\sigma_t[2]c_2$$

$$P_t[1][1] = P_{t-1}^-[1][1] - (2 + \tfrac{1}{t})c_2^2 c_1(c_3^2 - c_3 c_4)$$
$$P_t[1][2] = P_{t-1}^-[1][2] - (2 + \tfrac{1}{t})c_2^2 c_1(-c_3^2 - c_3 c_4)$$
$$P_t[2][1] = P_{t-1}^-[2][1] - (2 + \tfrac{1}{t})c_2^2 c_1(-c_3^2 - c_3 c_4)$$
$$P_t[2][2] = P_{t-1}^-[2][2] - (2 + \tfrac{1}{t})c_2^2 c_1(c_4^2 - c_3 c_4)$$

where $c_3 = P[1][1] - P[1][2]$ and $c_4 = P[2][2] - P[1][2]$.

Based on the above described process the UAVs made the decisions that are depicted in Figure 4. Initially both UAVs choose to flight in low altitude. They need 2 iterations to learn the other UAV's action and then they change their action which is the best response to low altitude. Nevertheless they both change altitude and thus the should keep play the symmetric game. In the next iteration of the EKF fictitious play they change together their actions to low altitude. This time quadcopter 2 adapt its estimation of quadcopter 1's propensity slower than quadcopter 1 does and only quadcopter 1 change altitude. In the new altitudes they don't observe the other UAV and they choose to complete their task to fly in opposite directions. Figures 5(a), 5(b), 5(c) and 5(d) depict snapshots of the implementation of the above algorithm using two Parrot AR.Drone 2.0.
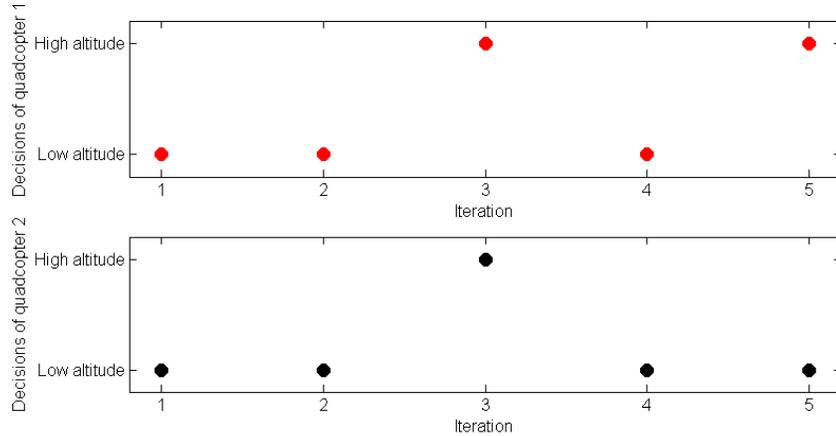

Fig. 4. Decisions of the quadcopters in the TCAS implementation.

5   Conclusions

A new TCAS principle has been introduced for unmanned aerial vehicles represented by agents capable of decisions about the flight of their planes, based on a variant of
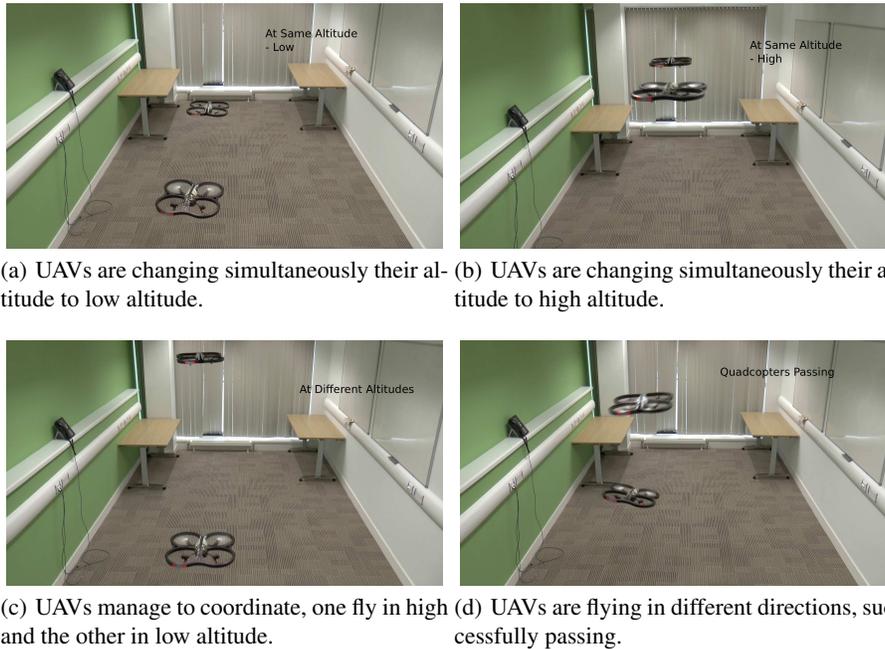
(a) UAVs are changing simultaneously their altitude to low altitude.

(b) UAVs are changing simultaneously their altitude to high altitude.

(c) UAVs manage to coordinate, one fly in high and the other in low altitude.

(d) UAVs are flying in different directions, successfully passing.

Fig. 5. Snapshots of the Implementation of Two Quadcopters Safely Passing Avoiding a Collision.

fictitious-play-based symmetric game with no communication requirements between the agents. The agents can infer all the necessary information they need by observing the other agent's actions. They use a variant of fictitious play, which combines the classical algorithm with extended Kalman filters, as coordination mechanism in order to avoid collision and accomplish their task.

References

1. J. M. Aitken, R. Alexander and T. Kelly (2010). A case for dynamic risk assessment in NEC systems of systems. In 5th International Conference on System of Systems Engineering (SoSE) 1-6.
2. J. Engel, J. Sturm and D. Cremers (2012). Camera-based navigation of a low-cost quadrocopter. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2815-2821.
3. J. Engel, J. Sturm, and D. Cremers (2014). Scale-aware navigation of a low-cost quadrocopter with a monocular camera. Robotics and Autonomous Systems.
4. Fudenberg, D. and Levine, D. (1998) *The theory of Learning in Games*. The MIT Press
5. C. Johnson, A. J. Gonzalez (2014), Learning collaborative team behavior from observation, Expert Systems with Applications. 41(5):2316-2328.
6. G. Klein and David Murray (2007). Parallel tracking and mapping for small AR workspaces. 6th IEEE and ACM International Symposium on Mixed and Augmented Reality.

7. K. Miyasawa (1961). On the convergence of learning process in a 2x2 non-zero-sum two person game. *Research Memorandum No 33, Princeton Unniversity*.
8. D. Monderer and L. Shapley (1996). Potential Games. *Games and Economic Behavior*, **14**, 124–143.
9. J. Nachbar (1990). "Evolutionary" selection dynamics in games: Convergence and limit properties. *International Journal of Game Theory*, **19**, 58–89.
10. J. Nash (1950). Equilibrium points in n-person games. Proceedings of the National Academy of Sciences. 36(1):48-49.
11. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Y. Ng (2009). ROS: an open-source Robot Operating System. In ICRA Workshop on Open Source Software.
12. J. Robinson (1951). An iterative Method of solving a game. *Annals of Mathematics*, **54**, 269–301.
13. I. Sa and P. Corke (2014). Vertical infrastructure inspection using a quadcopter and shared autonomy control. Field and Service Robotics. 219-232.
14. L. Shapley (1964). *In advances in Game theory*. Prinston, Prinston University
15. M. Smyrnakis and S. Veres (2014). Coordination of control in robot teams using game-theoretic learning. International Federation of Automatic Control (IFAC) 19th World Congress. Cape Town South Africa. 1194-1202.
16. T. Williamson and N. A. Spencer (1989). Development and operation of the traffic alert and collision avoidance system (TCAS). Proceedings of the IEEE, 77(11):1735-1744.
17. Smyrnakis, M., and Leslie, D. S. (2010). Dynamic opponent modelling in fictitious play. The Computer Journal, bxq006.
18. Bishop, C. M. (1995). Neural networks for pattern recognition.
19. Sakka, S. (2013). Bayesian Filtering and Smoothing. Cambridge University Press.