A Pursuit-evasion Game Between Unmanned Aerial Vehicles

Alexander Alexopoulos, Tobias Schmidt and Essameddin Badreddin
Automation Laboratory, Institute for Computer Engineering, Heidelberg University, B6, Mannheim, Germany

Keywords: Pursuit-evasion Games, Dynamic Non-cooperative Games, Unmanned Aerial Vehicles, Quad-rotors, Real-time Applications.

Abstract: In this paper the problem of two-player pursuit-evasion games with unmanned aerial vehicles (UAVs) in a three-dimensional environment is solved. A game theoretic framework is presented, which enables the solution of dynamic games in discrete time based on dynamic programming. The UAV agents taking part in the pursuit-evasion game are two identical quad-rotors with the same non-linear state space model, while the evaders' absolute velocity is smaller than the pursuers'. The convergence of the pursuit-evasion game is shown in numerical simulations. Finally, the approach is simulated on an embedded computer and tested for real-time applicability. Hence, the implementation and real-time execution on a physical UAV system is feasible.

1 INTRODUCTION

In recent years, pursuit-evasion games (PEGs) are highly challenging problems in the research area of optimal control theory and robotics. Generally, in PEGs a pursuer (or a team) are supposed to capture an evader (or a team) that is trying to escape. Many applications and areas of operations are conceivable, e.g., search and rescue missions, cops and robber games, patrolling, surveillance, and warfare. In robotics, there exist two primary approaches for solving PEGs: combinatorial and differential. The former requires the environment being represented either geometrically (e.g., with polygons) or topologically (e.g., by a graph). The *Lion and Man* problem is a famous example of PEGs. According to (Nahin, 2012), it is deemed to be one of the first (unpublished) mathematically formulated PEGs, defined by R. Rado in the 1920s. The problem was extensively studied, e.g., by Littlewood (Littlewood, 1986) and Sgall (Sgall, 2001). The former tackles the problem in continuous time and space, while, on the contrary, the latter analyzes it in discrete time.

The authors of (Chung et al., 2011) summarized different approaches for solving PEGs, which are applicable in robotics. They aimed to survey methods that are based on combinatorial approaches.

Earlier, LaValle and Hutchinson surveyed (LaValle and Hutchinson, 1993) various applications in robotics, which are applicable for game theoretic formulation. The focus of their survey is how game theory can be applied to robot navigation, high-level strategy planning, information gathering through manipulation and/or sensor planning, and pursuit-evasion scenarios.

Game-theoretic approaches consider that the solution of a problem does not only depend on the own decisions but on the decisions of each agent involved. Those problems are solved assuming rational decision making by all players. PEGs can be formulated as *dynamic non-cooperative games*, while the evolution of the game state depends on the dynamic constraints of each agent. Such dynamic PEGs (*differential games*) where introduced by Isaacs (Isaacs, 1965), e.g., *the Homicidal Chauffeur Game*. In this game a more agile but slower evader shall avoid to become run over by a faster but curvature-bound pursuer. The agents of such dynamic games are described by differential equations, which characterize the agents' dynamics.

The authors of (Vieira et al., 2009) published an implementation of a PEG on mobile robots, where a group of pursuers are supposed to catch a group of evaders. They use game theory to solve the PEG. Unfortunately, the PEG is solved off-line and the cost of the robots' motions are stored as weights on a mathematical graph. The motions of the robots are based on the best path according to the edges' weights between the graphs' vertices. This approach provides an *open-loop* solution, because the agents cannot respond to unpredictable events.

As far as is known, there is hardly anything done in research area of PEGs with UAVs in three-

Alexopoulos A., Schmidt T. and Badreddin E..
A Pursuit-evasion Game Between Unmanned Aerial Vehicles.
DOI: 10.5220/0005038600740081
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 74-81
ISBN: 978-989-758-040-6

dimensional environment, thus, an implementation of a PEG on real UAV systems seems not to be carried out, yet. In this paper, a framework is proposed, which provides a *closed-loop* solution of a PEG with two UAVs based on game-theoretic methods in a three-dimensional environment. Since the solutions (control actions of the UAVs) shall be calculated locally, the approach was implemented on an on-board embedded computer, running a real-time operating system (RTOS). This set-up assures that the predefined real-time specifications are satisfied. Hence, the foundations for an implementation on a real UAV system are laid.

In the next section the problem formulation is stated and the corresponding solution approach is presented. In section 3, a brief system description of the controlled UAV model is given. Section 4 introduces the framework with which *N-player discrete-time deterministic dynamic games* can be solved. Then, the two-player UAV PEG formulation is given in section 5. After that, the implementation of the PEG on the embedded computer is described in section 6. Finally, the simulation results and some interesting aspects and remarks are discussed in sections 7 and 8.

2 PROBLEM STATEMENT AND SOLUTION APPROACH

2.1 Problem Statement

Two UAV agents (pursuer and evader) with the same dynamic constraints are given. The pursuer is able to move faster through the three-dimensional environment than the evader. Furthermore, both UAVs have an attitude and velocity controller implemented. The agents are in a conflict situation called PEG. PEGs describe a problem, in which an agent tries to catch an adversarial agent, while the meaning of catch is the fulfilment of one or more conditions.

A solution to this game is sought that fulfils the following requirements:

- Considering that the solution depends on decisions of the antagonist, while each agent is aware of that.
- Being able to react to unexpected behavior of the adversarial agent (*closed-loop* solution).
- Computational time has to satisfy the determined real-time specifications.

2.2 Solution Approach

Therefore, the following points were processed:

- Game-theoretical problem formulation as *two-player discrete-time deterministic dynamic zero-sum game*.
- Consideration of feedback (perfect state) information structure and on-line computation of optimal strategies by calculating the *closed-loop* Nash equilibrium in mixed strategies in each discrete time step.
- Implementation of the approach on an embedded computer with RTOS.

3 SYSTEM DESCRIPTION

3.1 Dynamical Model

For modeling the quad-rotor dynamics, the mechanical configuration depicted in Figure 1 was assumed. The body fixed frame and the inertial frame are denoted by $\mathbf{e^B}$ and $\mathbf{e^I}$, respectively. The UAV is defined as a point mass. To derive the equations of motions, the following notations are necessary. $\mathbf{P^I} = (x,y,z)^T$ is the position vector of the quad-rotors' center of gravity in the inertial frame, $\mathbf{P^B} = (x_B,y_B,z_B)^T$ is the position vector of the quad-rotors' center of gravity in the body fixed frame, $\mathbf{v} = (u,v,w)^T$ are the linear velocities in the body fixed frame, $\omega = (p,q,r)^T$ are the angular rates for roll, pitch and yaw in the body fixed frame and $\Theta = (\phi,\theta,\psi)^T$ is the vector of the Euler angles. A key component of the quad-rotor model is the transformation between inertial and body frames. Rigid body dynamics are derived with respect to the body frame that is fixed in the center of gravity of the quad-rotor. However, to simulate the motion of the quad-rotor in the inertial frame, a transformation of the coordinates is needed. If the quad-rotors' attitude is parameterized in terms of Euler angles, the transformation can be performed using the rotation matrix $R(\Theta)$, which is a function of roll, pitch and yaw angles. Using s and c as abbreviations for $\sin(\cdot)$ and

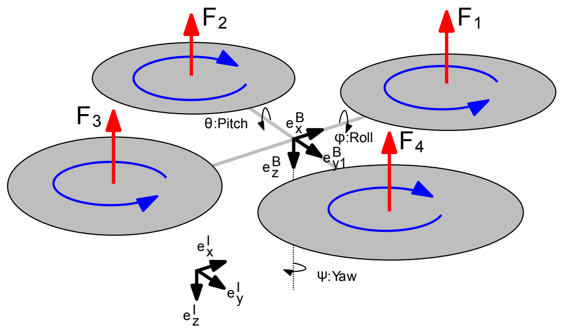


Figure 1: Mechanical configuration of a quadrocopter with body fixed and inertial frame.

$\cos(\cdot)$, respectively, the linear velocities defined in the inertial frame can be obtained as follows:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi - c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}. \quad (1)$$

The transformation of positions defined in the body frame into the corresponding positions in the inertial frame can be obtained by

$$\begin{bmatrix} \mathbf{P^I} \\ 1 \end{bmatrix} = \begin{bmatrix} R(\Theta) & \mathbf{P^I_{B,org}} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P^B} \\ 1 \end{bmatrix}. \quad (2)$$

The equations of motion are derived from the first principles (Newton-Euler laws (Beatty, 2006)) to describe both the translational and rotational motion of the quad-rotor, leading to the following discrete-time non-linear state space model with the state vector $\mathbf{x} = \left[x^k\, y^k\, z^k\, u^k\, v^k\, w^k\, \phi^k\, \theta^k\, \psi^k\, p^k\, q^k\, r^k\right]^T = \left[x_1^k\, x_2^k\, x_3^k\, x_4^k\, x_5^k\, x_6^k\, x_7^k\, x_8^k\, x_9^k\, x_{10}^k\, x_{11}^k\, x_{12}^k\right]^T$, while s and c are abbreviations for $\sin(\cdot)$ and $\cos(\cdot)$, respectively:

$$\begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \\ x_3^{k+1} \\ x_4^{k+1} \\ x_5^{k+1} \\ x_6^{k+1} \\ x_7^{k+1} \\ x_8^{k+1} \\ x_9^{k+1} \\ x_{10}^{k+1} \\ x_{11}^{k+1} \\ x_{12}^{k+1} \end{bmatrix} = \begin{bmatrix} x_1^k \\ x_2^k \\ x_3^k \\ x_4^k \\ x_5^k \\ x_6^k \\ x_7^k \\ x_8^k \\ x_9^k \\ x_{10}^k \\ x_{11}^k \\ x_{12}^k \end{bmatrix} + \begin{bmatrix} x_4^k \\ x_5^k \\ x_6^k \\ -(cx_7^k sx_8^k cx_9^k + sx_7^k sx_9^k)\frac{\upsilon_1^k}{m} \\ -(cx_7^k sx_8^k sx_9^k + sx_7^k cx_9^k)\frac{\upsilon_1^k}{m} \\ g - cx_7^k cx_8^k \frac{\upsilon_1^k}{m} \\ x_{10}^k \\ x_{11}^k \\ x_{12}^k \\ \frac{I_y - I_z}{I_x} x_{11}^k x_{12}^k + \frac{L}{I_x}\upsilon_2^k - \frac{I_R}{I_x} x_8^k g(\upsilon) \\ \frac{I_z - I_x}{I_y} x_{10}^k x_{12}^k + \frac{L}{I_y}\upsilon_3^k - \frac{I_R}{I_y} x_7^k g(\upsilon) \\ \frac{I_x - I_y}{I_z} x_{10}^k x_{11}^k + \frac{1}{I_z}\upsilon_4^k \end{bmatrix} \Delta t, \quad (3)$$

with $\upsilon = (\upsilon_1, \upsilon_2, \upsilon_3, \upsilon_4)^T, \upsilon \in \Upsilon$ being the inputs for altitude, roll, pitch and yaw, I_x, I_y, I_z are the inertia around x,y,z-axes, I_r is the rotor moment of inertia, L is the length between the center of gravity of the UAV and the center of one rotor, g is the gravitation constant, $g(\upsilon)$ is a function of υ depending on the rotors' angular velocities and Δt is the sampling time. The derivation of the model cannot be handled here in detail. For more details on quad-rotor modeling (Bouabdallah and Siegwart, 2007) can be consulted. A closer look at the state space model reveals that the angular accelerations depend only on the angular rates and the input vector υ and the linear accelerations depend on the Euler angles and υ. Hence, the state space model can be divided into two interlinked sub-models $M1$ and $M2$. Table 1 lists the chosen parameters based on (Voos, 2009). In this paper, all values without unit are normalized to the SI units.

Table 1: Model parameters.

Parameter	Value
m	0.5
L	0.2
$I_x = I_y$	$4.85 \cdot 10^{-3}$
I_z	$8.81 \cdot 10^{-3}$
I_R	$3.36 \cdot 10^{-5}$
thrust factor	$2.92 \cdot 10^{-6}$
air drag factor	$1.12 \cdot 10^{-7}$

3.2 Attitude and Velocity Control

The model structure is suitable for a cascaded attitude and velocity controller. The attitude controller, controlling subsystem $M1$, is ordered in the (faster) inner loop and the velocity controller, controlling $M2$ in the (slower) outer loop. The control of attitude and velocity of quad-rotors are not part of this work; therefor, refer to (Voos, 2009) and (Krstic et al., 1995) for more details about the present controller. More than sufficient reference reaction with the given control structure were derived in simulations (Alexopoulos et al., 2013).

4 GAME-THEORETICAL SOLUTION APPROACH

Game theory is an approach for strategic decision-making, considering that the solution depends on the decision of other agents, while everybody is aware of that. This makes the solution process very complex, especially if the number of players rises. Since PEGs are highly competitive games, only *non-cooperative games* are considered in this paper. *Non-cooperative games* treat a conflict situation where increasing the pay-off of one player results in decreasing that of another. The following definition describes the class of games considered in this work.

4.1 N-player Discrete-time Deterministic Dynamic Games

A *N-player discrete-time deterministic dynamic game* with a non-fixed terminal time can be defined by the octuplet $\{\mathbf{N}, \mathbf{K}, X, U, f, \iota, \Gamma, L\}$ with:

- A set of players $\mathbf{N} = \{1, \ldots, N\}$.
- A set $\mathbf{K} = \{1, \ldots, K\}$ denoting the stage of the game.

- An infinite set X, being the state space with the states $\mathbf{x}^k \in X, \forall\, k \in \mathbf{K} \cup \{K+1\}$.
- A set U_i^k, with $k \in \mathbf{K}$ and $i \in \mathbf{N}$ being the action space of player i in stage k, where the elements $\mathbf{u}_i^k$ are all admissible actions of player i in stage k.
- A difference equation $f_k : X \times U_1^k \times U_2^k \times \cdots \times U_N^k \to X$, defined for each $k \in \mathbf{K}$, so that
$$\mathbf{x}^{k+1} = f^k(\mathbf{x}^k, \mathbf{u}_1^k, \ldots, \mathbf{u}_N^k), k \in \mathbf{K} \qquad (4)$$
with $\mathbf{x}^1 \in X$ as the initial state, describing the evolution of a decision process.
- A finite set ι_i^k, defined for each $k \in \mathbf{K}$ and $i \in \mathbf{N}$, is the information structure of each player, while the collection of all players information structures ι is the information structure of the game.
- A class Γ_i^k of functions $\gamma_i^k : X \to U_i^k$ defined for each $k \in \mathbf{K}$ and $i \in \mathbf{N}$ are the strategies of each player i in stage k. The class Γ_i is the collection of all such strategies and is the strategy space of player i.
- A functional $L_i : (X \times U_1^1 \times \cdots \times U_1^K) \times (X \times U_2^1 \times \cdots \times U_2^K) \times \cdots \times (X \times U_N^1 \times \cdots \times U_N^K) \to \Re$ defined for each $i \in \mathbf{N}$ and is called cost functional of player i.

The game stops as soon as the terminal set $\Xi \subset X \times \{1, 2, ...\}$ is reached, meaning for a given N-tuple of actions in stage k, k is the smallest integer with $(\mathbf{x}^k, k) \in \Xi$. With this definition it is possible to describe the dynamic game in normal form (matrix form). Each fixed initial state $\mathbf{x}^1$ and each fixed N-tuple of admissible strategies $\{\gamma_i \in \Gamma_i; i \in \mathbf{N}\}$ yield a unique set of vectors $\{\mathbf{u}_i^k \triangleq \gamma_i^k(\iota_i^k), \mathbf{x}^{k+1}; k \in \mathbf{K}, i \in \mathbf{N}\}$, due to the causality of the information structure and the evolution of the states according to a difference equation. Inserting this vector in L_i $(i \in \mathbf{N})$ yields a unique N-tuple of numbers, reflecting the costs of each player. This implicates the existence of the mapping $J_i : \Gamma_1 \times \cdots \times \Gamma_N \to \Re$ for all $i \in \mathbf{N}$, being also the cost functional of player i with $i \in \mathbf{N}$. According to that, the spaces $(\Gamma_1, \ldots, \Gamma_N)$ and the cost functional $(J_1, \ldots, J_N)$ built the normal-form description of the dynamic game with a fixed initial state $\mathbf{x}^1$.

Since this class of games can be described in normal form, all solution concepts for *non-cooperative games*, e.g., found in (Başar and Olsder, 1999), can be used directly. For solving the later termed PEG with UAV agents, the solution concept of Nash equilibrium in mixed strategies (Nash, 1950) was used. Due to the fact that the PEG in this work is formulated as a *two-player zero-sum game* (Thomas, 1984), the saddle-point equilibrium in mixed strategies is sought, while being equivalent to the Nash equilibrium in *zero-sum games*.

4.2 Saddle-point Equilibrium

A tuple of action variables $(\mathbf{u}_1^*, \mathbf{u}_2^*) \in U, U = U_1 \times U_2$ in a two-player game with cost functional L is in saddle-point equilibrium, if

$$L(\mathbf{u}_1^*, \mathbf{u}_2) \le L(\mathbf{u}_1^*, \mathbf{u}_2^*) \le L(\mathbf{u}_1, \mathbf{u}_2^*),\ \forall (\mathbf{u}_1, \mathbf{u}_2) \in U. \qquad (5)$$

This means that the order of the maximization and minimization done is irrelevant:

$$\min_{\mathbf{u}_1 \in U_1} \max_{\mathbf{u}_2 \in U_2} L(\mathbf{u}_1, \mathbf{u}_2) = \max_{\mathbf{u}_2 \in U_2} \min_{\mathbf{u}_1 \in U_1} L(\mathbf{u}_1, \mathbf{u}_2) = L(\mathbf{u}_1^*, \mathbf{u}_2^*) =: L^* \qquad (6)$$

Note that if a value exists (a saddle-point exists), it is unique, meaning if another saddle-point $(\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2)$ exists, $L(\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2) = L^*$ applies. Moreover $(\mathbf{u}_1^*, \hat{\mathbf{u}}_2)$ and $(\hat{\mathbf{u}}_1, \mathbf{u}_2^*)$ constitute also a saddle-point. This feature does not hold for Nash equilibria (*non-zero-sum games*). If there is no value in a *zero-sum game*,

$$\min_{\mathbf{u}_1 \in U_1} \max_{\mathbf{u}_2 \in U_2} L(\mathbf{u}_1, \mathbf{u}_2) > \max_{\mathbf{u}_2 \in U_2} \min_{\mathbf{u}_1 \in U_1} L(\mathbf{u}_1, \mathbf{u}_2) \qquad (7)$$

holds. Hence, there is no saddle-point solution. Therefore, we consider the saddle-point equilibrium in mixed strategies with the following property:

Theorem 1 (Minimax-Theorem) *Each finite* two-player zero-sum game *has a saddle-point equilibrium in mixed strategies (von Neumann and Morgenstern, 2007).*

4.2.1 Saddle-point Solution in Mixed Strategies

If there is no saddle-point solution in pure strategies, the strategy space is extended, thus, the players can choose their strategies based on random events, leading to the so called mixed strategies. That means, a mixed strategy for a player i is a probability distribution $\mathbf{p}_i$ over the action space U_i. This holds also for general games having no Nash equilibrium. To get a solution in mixed strategies, L_i is replaced by its expected value, according to the chosen mixed strategies, denoted by $J_i(\mathbf{p}_1, \mathbf{p}_2)$. A 2-tuple $(\mathbf{p}_1^*, \mathbf{p}_2^*)$ is a saddle-point equilibrium in mixed strategies of a two-player game, if

$$J(\mathbf{p}_1^*, \mathbf{p}_2) \le J(\mathbf{p}_1^*, \mathbf{p}_2^*) \le J(\mathbf{p}_1, \mathbf{p}_2^*),\ \forall (\mathbf{p}_1, \mathbf{p}_2) \in P,\ P = P_1 \times P_2 \qquad (8)$$

holds, with $J(\mathbf{p}_1, \mathbf{p}_2) = E_{\mathbf{p}_1, \mathbf{p}_2}[L(\mathbf{u}_1, \mathbf{u}_2)]$. Thus, $J^* = J(\mathbf{p}_1^*, \mathbf{p}_2^*)$ is called the value of the *zero-sum game* in mixed strategies.

4.3 Discrete-time Dynamic Zero-sum Games

4.3.1 Information Structure

It is assumed that a feedback information structure is available to all agents during the game $\iota_i^k = \{\mathbf{x}^k\}, k \in \mathbf{K}, i \in \mathbf{N}$.

4.3.2 Stage-additive Cost Functional

The cost functional for the *discrete-time dynamic game* is formulated as follows:

$$L(\mathbf{u}_1,\dots,\mathbf{u}_N)=\sum_{k=1}^{K} g_i^k(\mathbf{x}^{k+1},\mathbf{u}_1^k,\dots,\mathbf{u}_N^k,\mathbf{x}^k), \qquad (9)$$

with $\mathbf{u}_j=(\mathbf{u}_j^{1'},\dots,\mathbf{u}_j^{K'})'$. This cost functional for player i is called “stage-additive” and implies the existence of a $g_i^k: X\times X\times U_1^k\times\cdots\times U_N^k\to\Re, k\in\mathbf{K}$.

4.3.3 Dynamic Programming for Discrete-time Dynamic Zero-sum Games

Since a stage-additive cost functional and a feedback information structure is assumed, dynamic programming and the *Principle of Optimality* (Bellman, 1957) can be applied. Hence, the set of strategies $\{\gamma_i^{k*}(\mathbf{x}^k);k\in\mathbf{K},i=1,2\}$ is for a *two-player discrete-time dynamic zero-sum game* a feedback-saddle-point solution if, and only if a function $V(k,\cdot):\Re^n\to\Re, k\in\mathbf{K}$ exists, thus the following recursion is satisfied:

$$\begin{aligned}
V_i(k,\mathbf{x}) &= \min_{\mathbf{u}_1^k\in U_1^k}\max_{\mathbf{u}_2^k\in U_2^k}\Big[g_i^k\left(f^k(\mathbf{x},\mathbf{u}_1^k,\mathbf{u}_2^k),\mathbf{u}_1^k,\mathbf{u}_2^k,\mathbf{x}\right)\\
&\qquad +V\left(k+1,f^k(\mathbf{x},\mathbf{u}_1^k,\mathbf{u}_2^k)\right)\Big]\\
&= \max_{\mathbf{u}_2^k\in U_2^k}\min_{\mathbf{u}_1^k\in U_1^k}\Big[g_i^k\left(f^k(\mathbf{x},\mathbf{u}_1^k,\mathbf{u}_2^k),\mathbf{u}_1^k,\mathbf{u}_2^k,\mathbf{x}\right)\\
&\qquad +V\left(k+1,f^k(\mathbf{x},\mathbf{u}_1^k,\mathbf{u}_2^k)\right)\Big]\\
&= g_i^k\left(f^k(\mathbf{x},\gamma_1^{k*}(\mathbf{x}),\gamma_2^{k*}(\mathbf{x})),\gamma_1^{k*}(\mathbf{x}),\gamma_2^{k*}(\mathbf{x}),\mathbf{x}\right)\\
&\qquad +V\left(k+1,f^k(\mathbf{x},\gamma_1^{k*}(\mathbf{x}),\gamma_2^{k*}(\mathbf{x}))\right);\\
V(K+1,\mathbf{x}) &= 0.
\end{aligned} \qquad (10)$$

The value function is found by calculating the saddle-point equilibria in mixed strategies recursively for each stage of the game as described above.

5 PURSUIT-EVASION GAME FORMULATION

The PEG between the two UAV systems is defined with following characteristics:

- A set of two players $\{e,p\}$.
- A set $\mathbf{K}=\{1,\dots,K\}$ with variable number of stages K. K is the time p needs to capture e, i.e., to minimize the distance d_ε to player e (e reaches the terminal set Ξ). Thus, K depends on the initial states of e and p.
- The terminal set $\Xi\subset X\times Y\times Z\times\{1,2,\dots\}$ is the set of all elements $\xi\in\Xi$ of a sphere around the pursuers' position (x_p,y_p,z_p) with radius d_ε in stage k.
- A set $\mathbf{X}=X\times Y\times Z\times U\times V\times W\times\Phi\times\Theta\times\Psi\times P\times Q\times R$ being the state space.
- Two finite discrete action spaces $U_p=U_e\subset U\times V\times W$. U_p and U_e are steady during each stage k of the game. They are defined as $U_p=\left\{\mathbf{u}_{pu,1}+i\frac{\mathbf{u}_{pu,2}-\mathbf{u}_{pu,1}}{s},\mathbf{u}_{pv,1}+j\frac{\mathbf{u}_{pv,2}-\mathbf{u}_{pv,1}}{s},\mathbf{u}_{pw,1}+l\frac{\mathbf{u}_{pw,2}-\mathbf{u}_{pw,1}}{s}\right\}$, with $i=0,\dots,s$; $j=0,\dots,s$; $l=0,\dots,s$ and $U_e=U_p$, while $(s+1)^3$ is the number of strategies available for each player and $[\mathbf{u}_{pu,1},\mathbf{u}_{pu,2}]=[\mathbf{u}_{pv,1},\mathbf{u}_{pv,2}]=[\mathbf{u}_{pw,1},\mathbf{u}_{pw,2}]=[\mathbf{u}_{eu,1},\mathbf{u}_{eu,2}]=[\mathbf{u}_{ev,1},\mathbf{u}_{ev,2}]=[\mathbf{u}_{ew,1},\mathbf{u}_{ew,2}]=[-1,1]$ are the continuous action spaces. $\mathbf{u}_p$ and $\mathbf{u}_e$ are elements of the sets U_p and U_e, while $\mathbf{u}\in U_p\times U_e$.
- The state of the PEG between two UAVs in the pursuers reference frame is defined as

$$\mathbf{x}^k=\mathbf{x}_e^k-\mathbf{x}_p^k=\begin{bmatrix}x^k\\y^k\\z^k\\u^k\\v^k\\w^k\\\phi^k\\\theta^k\\\psi^k\\p^k\\q^k\\r^k\end{bmatrix}=\begin{bmatrix}x_e^k-x_p^k\\y_e^k-y_p^k\\z_e^k-z_p^k\\u_e^k-u_p^k\\v_e^k-v_p^k\\w_e^k-w_p^k\\\phi_e^k-\phi_p^k\\\theta_e^k-\theta_p^k\\\psi_e^k-\psi_p^k\\p_e^k-p_p^k\\q_e^k-q_p^k\\r_e^k-r_p^k\end{bmatrix}=\begin{bmatrix}x_1^k\\x_2^k\\x_3^k\\x_4^k\\x_5^k\\x_6^k\\x_7^k\\x_8^k\\x_9^k\\x_{10}^k\\x_{11}^k\\x_{12}^k\end{bmatrix} \qquad (11)$$

  with the difference function $\mathbf{x}^{k+1}=f\left(\mathbf{x}^k,h(\mathbf{w}^k)\right)$ defined in equation 3, while $\mathbf{w}^k=\mathbf{u}_e^k-\mathbf{u}_p^k$ and $h:U\times V\times W\to\Upsilon$ provides an input vector $\upsilon=(\upsilon_1,\upsilon_2,\upsilon_3,\upsilon_4)^T$. Note that this state space model describes the evolution of the PEG state relative to the pursuer p.
- A feedback perfect state information structure $\iota_e^k=\iota_p^k=\{\mathbf{x}^k\},\forall k\in\mathbf{K}$.
- The strategy spaces $\Gamma_p=U_p$ und $\Gamma_e=U_e$.
- A cost functional

$$J(\mathbf{p}_p^k,\mathbf{p}_e^k)=E\left[\sum_{k=1}^{K}\mathrm{D}(f\left(\mathbf{x}^k,h(\mathbf{w}^k)\right),\mathbf{x}^k)\right], \qquad (12)$$

  with $\mathrm{D}(\cdot)$ being a function describing the change in distance between p and e in one stage k, playing the control action $(\mathbf{u}_p^k,\mathbf{u}_e^k)$.
- The value function

$$V(k,\mathbf{x}^k)=\min_{\mathbf{p}_p^k}\max_{\mathbf{p}_e^k}J(\mathbf{p}_p^k,\mathbf{p}_e^k). \qquad (13)$$

- $\mathbf{p}^{k*} = (\mathbf{p}_p^{k*}, \mathbf{p}_e^{k*})$ is the optimal solution of the game in stage k. It is calculated by solving the closed-loop saddle-point equilibrium in mixed strategies. The optimal probability distributions $\mathbf{p}^{k*} = (\mathbf{p}_p^{k*}, \mathbf{p}_e^{k*})$ over the action space $U_p \times U_e$ in stage k is given by
$$\mathbf{p}^{k*} = argV(k, \mathbf{x}^k), \forall k \in \mathbf{K}. \qquad (14)$$
- The optimal control actions $\mathbf{u}^{k*} = (\mathbf{u}_p^{k*}, \mathbf{u}_e^{k*})$ are those where the probabilities $\mathbf{p}_p^{k*}$ and $\mathbf{p}_e^{k*}$ are maximal. The reference velocities for the pursuers' and evaders' velocity controller are given by
$$\mathbf{v}_p^{r,k} = (u_p^k, v_p^k, w_p^k)^T + (\mathbf{u}_p^{k*})^T \qquad (15)$$
and
$$\mathbf{v}_e^{r,k} = (u_e^k, v_e^k, w_e^k)^T + (\mathbf{u}_e^{k*})^T. \qquad (16)$$

Since the solution of the above described problem shall be computed in real-time, the embedded computer and the implementation of the PEG are presented in the following section.

6 IMPLEMENTATION

The implementation of the pursuit-evasion problem defined above on an embedded computer is briefly described in this section. Firstly, the utilized embedded computer is described. Then, an algorithm is described, which enables the determination of Nash equilibria in mixed strategies of N-player games. Finally, a pseudo code is given describing the overall solution process of the PEG.

6.1 Embedded Computer

There are many low-power and small-size computers available, e.g., Raspberry Pi (Raspberry Pi Foundation, 2014), Cubieboard (CubieTech Ltd., 2014), BeagleBoard (Texas Instruments Inc., 2014a), and some variants. Many of those single-board computers are open-source hardware, assembled with a low-frequency processor. For this work a BeagleBone Black (Texas Instruments Inc., 2014b) was utilized, a community-supported development platform with a TI Sitara AM335x 1GHz ARMCortex A8 processor and 512MB DDR3 RAM. The embedded computer runs with QNX 6.5, a RTOS enabling the implementation and execution of real-time applications written in C programming language.

6.2 Algorithm for N-player Nash-equilibrium in Mixed Strategies

As described above, an optimal control action tuple $\mathbf{u}^{k*} = (\mathbf{u}_p^{k*}, \mathbf{u}_e^{k*})$ for the agents p and e in stage k of the PEG is derived by the determination of the Nash (saddle-point) equilibrium. The MATLAB-function NPG (Chatterjee, 2010) is able to solve an *N-player finite non-cooperative game* by computing one Nash equilibrium in mixed strategies. Thereby, the optimization formulation of a *N-player non-cooperative game* according to (Chatterjee, 2009) is used for computation. The function uses the *sequential quadratic programming* based *quasi Newton method* to solve a non-linear minimization problem with non-linear constraints.

Since it is not feasible to generate C code of the NPG function automatically, the algorithm to compute one Nash equilibrium was implemented from scratch in C to be applicable on the embedded computer. Therefor, the NLopt package (Johnson, 2013) was utilized to solve the non-linear minimization problem, more precisely the *SLSQP* (Kraft, 1988; Kraft, 1994) algorithm included there.

Algorithm 1 describes the overall solution process of the PEG defined above. Firstly, in stage k each control action $\mathbf{u} = (\mathbf{u}_p, \mathbf{u}_e)$ is simulated to calculate the resulting states $\mathbf{x}_p^{k+1}$ and $\mathbf{x}_e^{k+1}$ of both the pursuer and the evader playing its control action $\mathbf{u}_p$ and $\mathbf{u}_e$, respectively. The function call $\text{D}(f(\mathbf{x}^k, h(\mathbf{w}^k), \mathbf{x}^k)$ en-

Algorithm 1: PEG between two UAVs with recursive call of the *Value* function f.

```
 1: function PEG($\mathbf{x}^1$)
 2:     $(value^K, K) \leftarrow$ VALUE$(1, \mathbf{x}^1)$
 3:     return $(value^K, K)$
 4: end function

 5: function VALUE($k, \mathbf{x}^k$)
 6:     if NORM$((x^k, y^k, z^k)) \leq d_\varepsilon$ then
 7:         return $0, k$
 8:     else
 9:         for all $\mathbf{u} = (\mathbf{u}_p, \mathbf{u}_e) \in U_p \times U_e$ do
10:             $\mathbf{w} \leftarrow \mathbf{u}_e - \mathbf{u}_p$
11:             $L(\mathbf{u}) \leftarrow$ D$(f(\mathbf{x}^k, h(\mathbf{w})), \mathbf{x}^k)$
12:         end for
13:         $(\mathbf{p}_p^{k*}, \mathbf{p}_e^{k*}) \leftarrow$ NPG$(L, U_p \times U_e)$
14:         Select $(\mathbf{u}_e^*, \mathbf{u}_p^*)$ with MAX$(\mathbf{p}_p^{k*})$ and MAX$(\mathbf{p}_e^{k*})$
15:         $(val^{k+1}, \kappa) \leftarrow$ VALUE$(k+1, f(\mathbf{x}^k, h(\mathbf{w}^*)))$
16:         $val^k \leftarrow L(\mathbf{u}^*) + val^{k+1}$
17:         return $(val^k, \kappa)$
18:     end if
19: end function
```

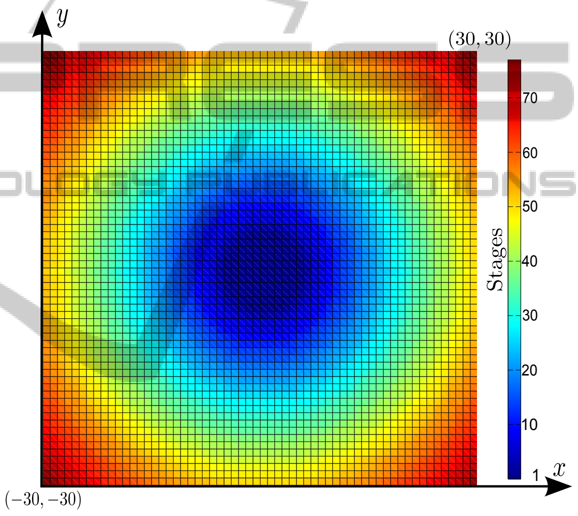
capsulates each of this steps and returns the change of the Euclidean norm of each guessed position difference, according to equation 12. Those distance changes are set as pay-offs $L(\mathbf{u})$ of the regarding control action $\mathbf{u}$. Then, one Nash equilibrium in mixed strategies is computed with the previously calculated pay-offs, according to equation 14. Lastly, the optimal control action $\mathbf{u}^* = (\mathbf{u}_p^*, \mathbf{u}_e^*)$ having the highest probability within the resulting probability distribution, is executed by the agents.

7 SIMULATION RESULTS

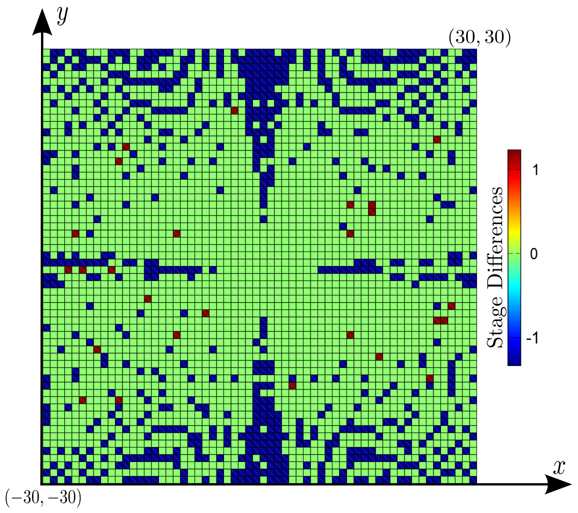
To be able to analyze the implementation on the embedded computer, a comparison with the solutions in MATLAB has to be carried out. Therefore, following assumptions were made for both simulations:

- Since the chosen optimal control actions represent a velocity change in three linear directions of p and e, a maximum velocity $\mathbf{v}_{max}$ with $\mathbf{v}_{max}^p = \begin{bmatrix}15 & 15 & 3.5\end{bmatrix}^T$ and $\mathbf{v}_{max}^e = \frac{\mathbf{v}_{max}^p}{1.5}$ and a maximal absolute value of $v_{maxA}^p = 15$ for the pursuer and $v_{maxA}^e = 10$ for the evader was defined.
- The numerical solution of the PEG is computed by solving it for each initial positions $(x^1, y^1, z^1) \in X \times Y \times Z$, while x^1 and y^1 take integer values in a 61x61 grid, with $X = [-30, 30]$ and $Y = [-30, 30]$ in the pursuers' reference frame (pursuers' position is the origin). In each simulation, the initial altitude of both UAVs is 20, i.e., $z^1 = -20$. This was necessary for the visualization of the value function.
- $s = 6$ was chosen, i.e., each player has 7^3 strategies available in each time step k.
- The stage duration was chosen to be $\Delta T = 0.1$, while the velocity control is sampled with $\Delta t = 0.005$. The real-time specification to be satisfied by the embedded computer was $\Delta T = 0.1$s for one stage k.
- A capture distance $d_\varepsilon = 5$ was chosen, since it is the maximum change in distance, which can be achieved in $\Delta t = 1$ regarding $v_{maxA}^p = 15$ and $v_{maxA}^e = 10$.

Figure 2(a) depicts the value function over the regarded discretized state space computed by the embedded computer. Regarding this solutions the convergence of the PEG in three dimensions is given everywhere, meaning that in this configuration the evader can never avoid to be captured by the pursuer. Figure 2(b) depicts the difference of the value of stages between the MATLAB simulation and the simulation on the embedded computer. The differences are slightly in the whole state space. Moreover, due to the very small differences (caused by possible rounding errors and varieties in the minimization algorithm implementation) between the MATLAB and the embedded computer solution, the implementation on the BeagleBone Black was accomplished successfully. The next important point was to check the real-time applicability of the approach. The demanded computational time of $\Delta T = 0.1$s for one stage of the game was successfully satisfied. By configuring the algorithm for the saddle-point computation of one stage k, such that it stops after maximal 0.09s, the minimization algorithm was still able to maintain the demanded absolute tolerance of 10^{-6} for the mini-



(a) Value of stages needed for capture.



(b) Difference of value of stages needed for capture in MATLAB and on the embedded computer.

Figure 2: Simulation Results.

mum function value. The use of an RTOS assures that the algorithm yields a solution within $\Delta T = 0.1$s, thus the real-time specifications are satisfied.

8 CONCLUSIONS

The goal of this paper was to present a framework, which enables the solution of a PEG with UAVs in three dimensions. This framework, formulated in a game-theoretical manner, does not only provide a solution approach for the present problem, but for all problems which can be formulated as *N-player discrete-time deterministic dynamic games*. By applying this approach the convergence of the PEG in a three-dimensional environment with UAV agents having dynamic constraints was shown successfully. Furthermore, the approach was implemented on an embedded computer providing results equal to the MATLAB implementation. Finally, the real-time applicability of the approach was shown successfully in simulations. This paper forms the basis for a real UAV system implementation of the presented approach, which will be carried out next on the quad-rotor system *L4-ME* of HiSystems GmbH (MikroKopter, 2014).

REFERENCES

Alexopoulos, A., Kandil, A. A., Orzechowski, P., and Badreddin, E. (2013). A comparative study of collision avoidance techniques for unmanned aerial vehicles. In *SMC*, pages 1969–1974.

Başar, T. and Olsder, G. J. (1999). *Dynamic Noncooperative Game Theory (Classics in Applied Mathematics)*. Soc for Industrial & Applied Math, 2 edition.

Beatty, M. (2006). *Principles of Engineering Mechanics: Volume 2 Dynamics – The Analysis of Motion*. Mathematical Concepts and Methods in Science and Engineering. Springer.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition.

Bouabdallah, S. and Siegwart, R. (2007). *Advances in Unmanned Aerial Vehicles*, chapter Design and Control of a Miniature Quadrotor, pages 171–210. Springer Press.

Chatterjee, B. (2009). An optimization formulation to compute nash equilibrium in finite games. In *Methods and Models in Computer Science, 2009. ICM2CS 2009. Proceeding of International Conference on*, pages 1–5.

Chatterjee, B. (2010). n-person game. www.mathworks.com/matlabcentral/fileexchange/27837-n-person-game.

Chung, T. H., Hollinger, G. A., and Isler, V. (2011). Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316.

CubieTech Ltd. (2014). Cubieboard – A series of open ARM miniPCs. www.cubieboard.org.

Isaacs, R. (1965). *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. John Wiley and Sons, Inc., New York.

Johnson, S. G. (2013). The nlopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt.

Kraft, D. (1988). A software package for sequential quadratic programming. Technical Report DFVLR-FB 88-28, DFVLR, Cologne, Germany.

Kraft, D. (1994). Algorithm 733: Tompfortran modules for optimal control calculations. *ACM Transactions on Mathematical Software*, 20(3):262–281.

Krstic, M., Kokotovic, P. V., and Kanellakopoulos, I. (1995). *Nonlinear and Adaptive Control Design*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

LaValle, S. M. and Hutchinson, S. A. (1993). Game theory as a unifying structure for a variety of robot tasks. In *Proceedings of 8th IEEE International Symposium on Intelligent Control*, pages 429–434. IEEE.

Littlewood, J. E. (1986). *Littlewood's Miscellany*. Cambridge University Press.

MikroKopter (2014). MK-QuadroKopter / L4-ME. http://www.mikrokopter.de/ucwiki/MK-Quadro.

Nahin, P. J. (2012). *Chases and Escapes: The Mathematics of Pursuit and Evasion (Princeton Puzzlers)*. Princeton University Press.

Nash, J. F. (1950). *Non-cooperative Games*. PhD thesis, Princeton University, Princeton, NJ.

Raspberry Pi Foundation (2014). Raspberry Pi. www.raspberrypi.org.

Sgall, J. (2001). Solution of david gale's lion and man problem. *Theoretical Computer Science*, 259(1-2):663–670.

Texas Instruments Inc. (2014a). Beagleboard.org. www.bealgeboard.org.

Texas Instruments Inc. (2014b). BeagleBone Black. www.beagleboard.org/Products/BeagleBone Black.

Thomas, L. C. (1984). *Games, Theory and Applications*. Dover Books on Mathematics. Dover Publications.

Vieira, M., Govindan, R., and Sukhatme, G. (2009). Scalable and practical pursuit-evasion. In *Robot Communication and Coordination, 2009. ROBOCOMM '09. Second International Conference on*, pages 1–6.

von Neumann, J. and Morgenstern, O. (2007). *Theory of Games and Economic Behavior (60th-Anniversary Edition)*. Princeton University Press.

Voos, H. (2009). Entwurf eines flugreglers für ein vierrotoriges unbemanntes fluggerät (control systems design for a quadrotor uav). *Automatisierungstechnik*, 57(9):423–431.