CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Bachelor Thesis

Jan Vakula

Escape behavior in swarms of unmanned helicopters

Department of Cybernetics

Thesis supervisor: **Dr. Martin Saska**

In Prague on May 25, 2012

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Jan Vakula

Study programme: Cybernetics a Robotics

Specialisation: Robotics

Title of Bachelor Project: Escape Behavior in Swarms of Unmanned Helicopters

Guidelines:

The aim of this thesis is to design, implement and verify an algorithm of escape behavior, which decreases possibility of collisions of dynamics obstacles with swarms of unmanned helicopters (UAV).
Work plan:

1. To design and implement an extension of the escape behavior, presented in [1], to 3D.
2. To integrate a model of UAV movement from [2] and constraints given by relative localization of swarm particles.
3. To integrate the developed method into the control system of autonomous swarms.
4. To verify the implemented system with simulations of movement of the 3D swarm.
5. To demonstrate the functionality of the method with the system of autonomous mobile robots, SyRoTek [3].

Bibliography/Sources:

[1] Min, H.; Wang, Z.: Design and analysis of Group Escape Behavior for distributed autonomous mobile robots. Robotics and Automation (ICRA), 2011 IEEE International Conference on , vol., no., pp.6128-6135, 9-13 May 2011.
[2] Lee, T.; Leok, M.; McClamroch, N. H.: Geometric tracking control of a quadrotor UAV on SE(3). Decision and Control (CDC), 2010 49th IEEE Conference on , pp.5420-5425, 15-17 Dec. 2010.
[3] Kulich, M.; Košnar, K.; Chudoba, J.; Faigl, J.; Přeučil, L.: On a Mobile Robotics E-learning System. In Proceedings of the Twentieth European Meeting on Cybernetics and Systems Research. Vienna: Austrian Society for Cybernetics Studies, 2010, p. 597-602.

Bachelor Project Supervisor: Ing. Martin Saska, Dr. rer. nat.

Valid until: the end of the winter semester of academic year 2012/2013

L.S.

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 9, 2011

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jan V a k u l a

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Únikové chování navržené pro roje bezpilotních helikoptér

Pokyny pro vypracování:

Cílem práce je navrhnout, implementovat a verifikovat algoritmus skupinového únikového chování, které sníží riziko kolize roje bezpilotních helikoptér a dynamických překážek.

Plán prací:
1. Navrhnout a implementovat rozšíření metody „Escape behavior" [1] do 3D.
2. Zakomponovat do algoritmu model pohybu helikoptéry [2] a omezení daná relativní lokalizací členů roje, což umožní reálné aplikace bezpilotních helikoptér.
3. Integrovat výslednou metodu do celkového systému řízení autonomních rojů.
4. Ověřit implementovaný systém simulacemi pohybu roje ve 3D.
5. Demonstrovat funkci algoritmu na systému SyRoTek [3].

Seznam odborné literatury:

[1] Min, H.; Wang, Z.: Design and analysis of Group Escape Behavior for distributed autonomous mobile robots. Robotics and Automation (ICRA), 2011 IEEE International Conference on , vol., no., pp.6128-6135, 9-13 May 2011.
[2] Lee, T.; Leok, M.; McClamroch, N. H.: Geometric tracking control of a quadrotor UAV on SE(3). Decision and Control (CDC), 2010 49th IEEE Conference on , pp.5420-5425, 15-17 Dec. 2010.
[3] Kulich, M.; Košnar, K.; Chudoba, J.; Faigl, J.; Přeučil, L.: On a Mobile Robotics E-learning System. In Proceedings of the Twentieth European Meeting on Cybernetics and Systems Research. Vienna: Austrian Society for Cybernetics Studies, 2010, p. 597-602.

Vedoucí bakalářské práce: Ing. Martin Saska, Dr. rer. nat.

Platnost zadání: do konce zimního semestru 2012/2013

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 12. 2011

Declaration

I hereby declare that I have completed this thesis independently and that I have used only the sources (literature, software, etc.) listed in the enclosed bibliography.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on May 25, 2012 .....

Aknowledgements

I would like to thank all the people who helped me in any way with the bachelor thesis. Especially, I would like to thank my supervisor Ing. Martin Saska, Dr. rer. nat. for showing me the right path. Further I have to thank my room-mates and my family for supporting me.

Abstract

This thesis deals with the design, implementation and verification an algorithm of escape behaviour, which decreases the possibility of collision of a swarm of unmanned helicopters (UAV) with dynamic obstacles. The model of the quad-rotor movement and constraints given by relative localization of swarm particles is also implemented in the algorithm, which is crucial for real deployment of the method. Controller for the real model of the quad-rotor is designed and it calculates the speed of each propeller of quad-rotors depending on a prescribed trajectory. The algorithm for swarm control has been inspired by a paper [1], where the algorithm of escape behaviour for 2D robots with differential drive is described. The functionality of 2D algorithm is performed using simulations and testing on a robotic platform SyRoTek. The functionality of extended algorithm for UAV is verified using simulations in Matlab.

Abstrakt

Tato práce se zabývá návrhem, implementací a testováním algoritmu skupinového únikového chování, které sníží riziko kolize roje bezpilotních helikoptér a dynamických překážek. Pro reálné využití algoritmu je zakomponován model pohybu helikoptéry a omezení daná relativní lokalizací členů roje. Pro potřeby modelu je navržen regulátor, který počítá rychlosti jednotlivých vrtulí helikoptéry v závislosti na trajektorii, kterou má sledovat. Jako základ algoritmu je použit článek [1], který zkoumá skupinové únikové chování robotů, kteří se pohybují ve 2D. Ověření funkčnosti 2D algoritmu je provedeno pomocí simulací a implementací do robotické platformy SyRoTek. Funkčnost rozšířeného algoritmu pro bezpilotní helikoptéry je ověřena pomocí simulací v programu Matlab.

Contents

List of Figures

List of Tables

1 Introduction

The main goal of my Bachelor thesis is to understand, design, implement and verify the algorithm of escape behaviour, which decreases the possibility of collisions of dynamic obstacle with swarms of unmanned helicopters.

The motivation of this thesis is to integrate methods of control of a group of quad-rotors and natural behaviours of animals, like birds or fish. To find a way to move a large number of quad-rotors from one point to another. During the flight they have to fly in a compact swarm and be able to avoid obstacles.

For my thesis, an algorithm based on flocking behaviour was chosen. This algorithm is inspired by natural behaviour of fish, insect or birds. Animals have this algorithm *"programmed in their heads"*, that means they use flocking behaviour without thinking. It helps them to solve their cooperative tasks in their daily life. When looking at the motion of a flock of birds it seems as though there is some kind of a centralised control. The flock appears to move as one fluid object, it moves cooperatively. Nevertheless the movement decisions of individuals are decentralised and based upon the local surroundings.

One of the basic models of this behaviour may be controlled by three simple rules [2].

- Separation - avoid crowding neighbours (short range repulsion).
- Alignment - steer towards average heading of neighbours.
- Cohesion - steer towards average position of neighbours (long range attraction).

Such algorithm based on flocking behaviour has some advantages. For example:

1. Real-time Algorithm
   This algorithm can run real-time. There is nothing to be computed before robots start. Every program loop calculates with actual position of robots, obstacles and goal. The approach is also suitable for dynamic obstacles.
2. Distributed computation
   Every individual in quad-rotor flock computes his own program, which accelerates solving next step.

The biggest disadvantage of this algorithm is that it is based on constants, which are difficult to set.

I chose the algorithm described in "Design and analysis of Group Escape Behaviour for distributed autonomous mobile robots" [1] for unnamed ground vehicles (UGV) working in planar environment. In this paper the authors describe control paradigm for robots, that moves in two-dimensional (2D) space. At the beginning I used this approach for flocking and for obstacle avoidance and made its modifications. This paper helped me to understand advantages, disadvantages and principles of this algorithm. This experience helps me to build a novel method for quad-rotors, which can fly in three-dimensional space.

1.1 SyRoTek

The whole algorithm for 2D robots was developed and tested in Matlab. It has been rewroted to SyRoTek (System for robotic e-learning) [3] for better illustration of feasibility and validity. It is a multi-robot platform, which consists of an arena with real autonomous mobile robots, communication infrastructure and main control computer accessible from the Internet. SyRoTek experiments are the evidence that these algorithm can be employed for real robots. For use in SyRoTek the algorithm has to be rewritten in C++ language.

1.2 3D

Modification of the original UGV algorithm for using in 3D applications will be investigated. This algorithm is suitable for using with quad-rotors like "Parot AR.Drone" [4]. Model of quad-rotor's movement from the paper "Geometric Tracking Control of a Quadrotor UAV on SE(3)" [5] will be integrated to enable extensions to 3D. In [5] the authors describe the theoretical model of quad-rotor that is based on a real model and contains constraints. To simplify the implementation of escape behaviour algorithm to 3D, it was decided to solve this matter by dividing into two sub-problems. At first a controller for one quad-rotor is implemented. And then a behaviour model for a flock is developed. Equations for controller are used from the paper [5].

Matlab is used for the simulation of the developed method. For given input parameters Matlab calculates the forces that need to be generated by propellers. The algorithm will be visualised in Blender, which is a free open source 3D content creation suite.

2 Control equations for 2D

In this section the control equations for 2D, which is inspired by the paper [1], are explained. In this paper, planar robots with three degrees of freedom are used. Meaning of all variables and constants used in this section is described in Table 2 and in Table 1 for better readability. The equations

$$m_i \frac{\mathrm{d}\mathbf{v}_i}{dt} = \mathbf{F}_p + \sum_{j,j\neq i} e_{ij}\mathbf{F}_{KH_{ij}} - \gamma \mathbf{v}_i, \tag{1}$$

$$I_i \frac{\mathrm{d}^2\alpha_i}{dt^2} = \sum_{j,j\neq i} \left(e_{ij}M_{c_{ij}} + e_{ij}M_{d_{ij}}\right) + M_{rb_i} - D_n \frac{\mathrm{d}\alpha_i}{dt}, \tag{2}$$

describe the dynamics of robot's flock. There are two flocking equation, where equation (1) is Newton's second law of motion. On the left side of the equation there is described the dynamics of the i-th individual and on the right side there is a sum of all the forces acting on the i-th robot. Euler's second law for 2D, equation (2), represents the dynamics of rotation. The sum of the moments, which are acting on the i-th robot is on the right side.

$M_{c_{ij}}$ is the moment for keeping the same heading direction of the robots. This moment is important for faster reaction to obstacle avoidance. If any robot detects an obstacle or a predator, it will start to change it's direction. The other robots, which don't detect the obstacle and have local interaction, will change direction by moment the $M_{c_{ij}}$ as

$$M_{c_{ij}} = K_t\alpha_{ji} + C_t \frac{\mathrm{d}\alpha_{ji}}{\mathrm{d}t}. \tag{3}$$

$\mathbf{F}_{K_{ij}}$ denotes the force, that provides separation and cohesion. This force depends on the relative distance between two robots and on derivation of relative distance. This force is resolved into two components, one directed in the direction of heading vector of the robot and the other is perpendicular to the heading direction. $\mathbf{F}_{KH_{ij}}$ accelerates the robot and directly moves robots to required relative distance. $\mathbf{F}_{K\perp ij}$ rotates the robot. This is necessary for shortening or enlarging the relative distance between the robots if they are too far or too close.

The schematic figure with forces $\mathbf{F}_{KH_{ij}}$ and $\mathbf{F}_{K\perp ij}$ is shown in Figure 1. The size of $\mathbf{F}_{K_{ij}}$ is computed as

Table 1: Table of constants used in 2D

constant	description
K_t, C_t	Constants used in the equation (3). These constants tune the speed of reaction when α_{ij} is not 0.
K_d, D_d	Constants used in the equation (4). These constants tune speed of reaction when the robots divert from their optimal relative distance between them
k_{F-M}	Constant that transfers force into moment in the equation (6).
L_{d_i}	Constant, which sets required relative distance between the robots.
U	Potential function generating repulsion effect from the obstacle.
d	Constant used in the equation (8). This constant set magnitude of function δ_{rb_i} in dependence on angle.
a, b, c	Constants, which sets distance function. This constant is used in the equation (9) and (10).

$$\| \mathbf{F}_{K_{ij}} \| = K_d \left(L_{d_i} - \| \mathbf{L}_{ji} \| \right) + D_d \frac{\mathrm{d}\mathbf{L}_{ji}}{\mathrm{d}t}, \tag{4}$$

$$\mathbf{F}_{K_{ij}} = \mathbf{F}_{KH_{ij}} + \mathbf{F}_{K\perp ij}. \tag{5}$$

$\mathbf{F}_{K\perp ij}$ creates the moment $M_{d_{ij}}$ with the equation (6). The force is converted to moment by the constant k_{F-M} as

$$M_{d_{ij}} = \operatorname{sign}(\alpha_i - \sigma) \cdot k_{F-M} \cdot \mathbf{F}_{K\perp ij}. \tag{6}$$

The angle $(\alpha_i - \sigma)$ has to be in the interval $\langle -\pi, \pi)$.

The whole swarm also performs the obstacle avoidance by reshaping the swarm by incorporating the following local obstacle avoidance control law to each robot:

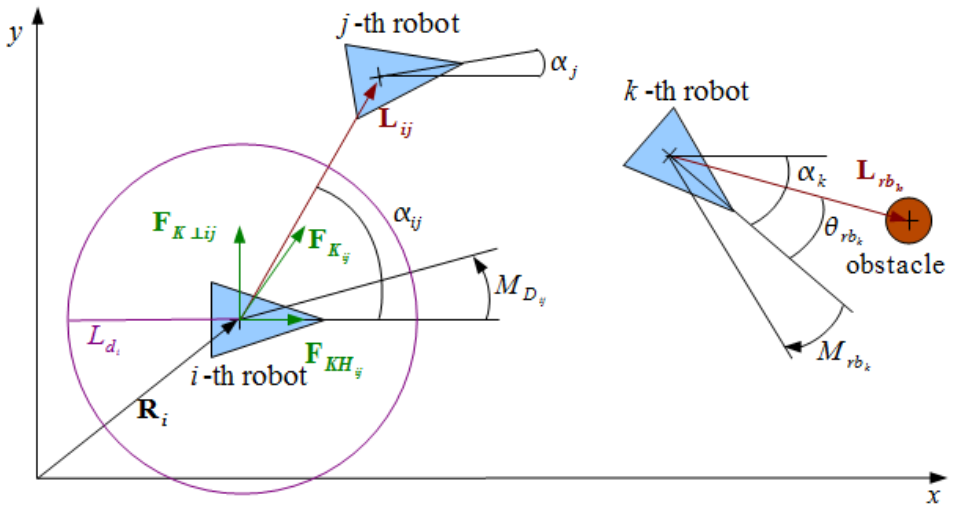

Figure 1: Schematically illustrated forces and moments acting on robots

$$M_{rb_i} = \text{sign}(\Theta_{rb_i}) \cdot \delta_{rb_i} \cdot e_{rb_i} \cdot U, \tag{7}$$

$$\delta_{rb_i} = (1 + d\cos(\alpha_i - \Theta_{rb_i})). \tag{8}$$

M_{rb_i} will generate repulsion rotational motion from direction towards the detected obstacle. Magnitude of this moment is decided by both the direction dependence function δ_{rb_i} and the potential function U generating repulsion effect from the obstacle. The dependence function δ_{rb_i} generates the repulsion according to the relative angle $(\alpha_i - \Theta_{rb_i})$ between heading direction vector and the direction vector to the obstacle $\mathbf{L}_{rb_i}$.

We should mention that all these local interactive forces and moments only work between the robots and obstacles inside their sensor range by using the distance functions e_{ij} and e_{rb_i}.

$$e_{ij} = \frac{1}{e^{a\cdot\mathbf{L}_{ij}-b}+c} + \frac{1}{e^{0.5\cdot a\cdot\mathbf{L}_{ij}-b}+c} \tag{9}$$

$$e_{rb_i} = \frac{1}{e^{a\cdot\mathbf{L}_{rb_i}-b}+c} + \frac{1}{e^{0.5\cdot a\cdot\mathbf{L}_{rb_i}-b}+c} \tag{10}$$

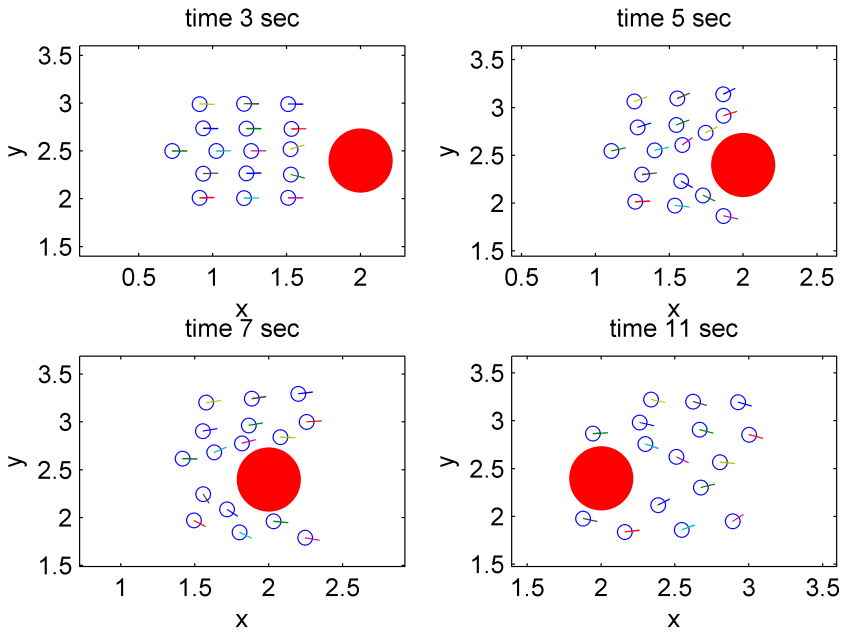

Figure 2: Sub-figures of simulation with 16 robots and static obstacle

2.1 Simulations

In order to simulate the algorithm and for setting constants mentioned in Table 1, a simulation platform was developed. The platform was written in Matlab. Input parameters for simulations are the starting position and the angle of the heading direction vector of the robots, the constants and the position of the obstacle. The simulation computes the trajectory, forces, speed and shows the movement of the swarm in a video sequence. An example of the video sequence is shown in Figure 2. In this sequence the robots start in front of the obstacle. The angles of heading vectors of all robots were set to 0 degree. At the beginning the robots move like a flock. After a few steps they detect an obstacle and the flock splits into two groups to avoid the obstacle. After avoiding the obstacle the group is merged back into one flock.

The trajectories of robots are shown in Figure 3. This figure illustrates that regrouping takes more time then splitting. It depends on the distance function e_{ij}, equation (9).

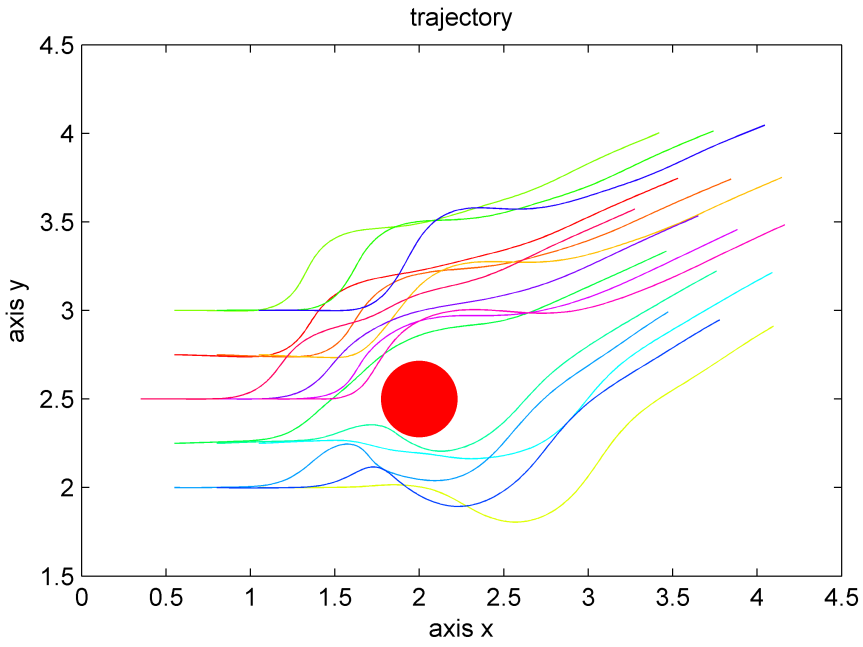

Figure 3: Trajectory of the simulation with static obstacle

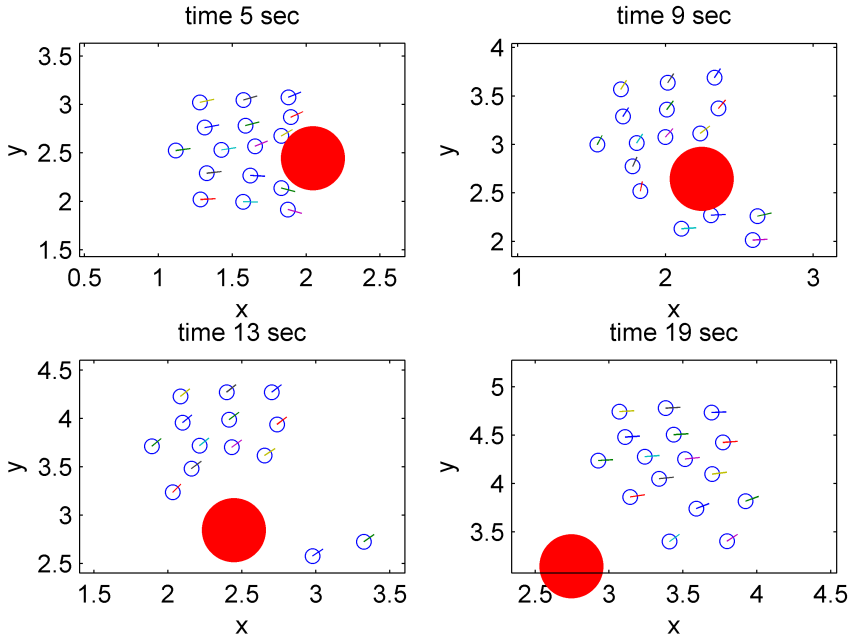

Figure 4: Sub-figures of the simulation with 16 robots and moving obstacle

The next simulation in Figure 4 shows the flock behaviour dealing with avoiding a dynamic obstacle. The obstacle moves during the simulation. This situation doesn't cause any troubles since the program calculates real-time all forces. Problems might emerge, if the obstacle is moving too fast and the time step of the simulation is too long.

Table 2: Table of variables used in 2D

variable	description
m_i	Mass of the i-th individual.
$\mathbf{v}_i$	Speed of the i-th robot.
γ	Viscosity coefficient of fluid, which represents resistance of the environment. If γ were 0, the robot could reach infinite speed.
I_i	Moment of inertia of the i-th individual.
α_i	The angle of heading direction vector in world coordinates.
σ	The angle between the force vector, which points to other robot, and the heading direction vector
$\mathbf{R}_i$	Denotes the position vector of the i-th individual
D_n	Rotation viscosity coefficient has the same purpose for rotation, like the viscosity coefficient γ for translation
$\mathbf{F}_p$	Propulsion force acting in the direction of the heading vector.
α_{ji}	$\alpha_{ji} = \alpha_j - \alpha_i$ denotes angular difference of heading direction from the j-th robot to the i-th robot.
$\mathbf{L}_{ji}$	$\mathbf{L}_{ji} = \mathbf{R}_j - \mathbf{R}_i$ denotes relative position vector from the j-th to the i-th robot
$\mathbf{L}_{rb_i}$	Relative position vector from the i-th robot to the position of the obstacle if it exists
Θ_{rb_i}	Angle between the vector $\mathbf{L}_{rb_i}$ and the heading direction vector of i-th robot

3 SyRoTek

Six mobile robots were used to show the performance of the developed algorithm. The robots have differential drive, which corresponds to the model used in section 2. On the top of the robot there is a label for its localization. Global localization from the top camera is used, that is more suitable for long time simulations. The achieved accuracy of the global localization is about 3 mm in position estimation and 5 degrees in robot orientation [6], which is enough for verification experiments. SyRoTek robots also have odometry localization, which is more accurate, but has a cumulative error.

The experiments were conducted in SyRoTek Arena. This is an enclosed space, which is 3.5 meters length and 3.8 meters width. The robot working space is a flat area with an outer barrier 18 cm tall. Additional 13 cm tall obstacles are placed inside. Some obstacles can be remotely retracted, while the rest of them is fixed. To get more space in experiments, the fixed obstacles were manually removed and only dynamic ones were used.

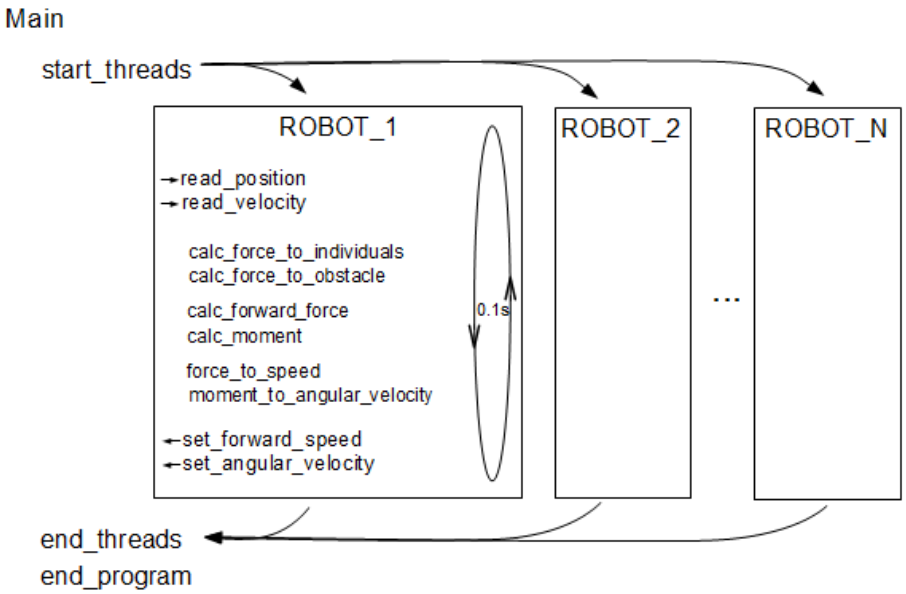

Figure 5: Program schema

The algorithm described in section 2 was developed and tested in Matlab. For testing in SyRoTek, I rewrote the code to C++. The Main class of the algorithm starts one thread for each robot. The threads run in a infinite loop. At the beginning of the loop the algorithm reads global positions of the robots. Afterwards it calculates the equation (1), equation (2) and finally desired forward speed and angular velocity. The scheme of the algorithm is illustrated in Figure 5. One loop lasts about 0.1 sec. The algorithm runs real-time, there is nothing to be calculated before the main program starts. The evaluation of SyRoTek system is described in Appendix A.

3.1 Experiments

1. All six robots start close to the border of the arena. The obstacle is extruded in the middle of the arena. According to the expectation the flock of robots splits into two groups of three members, as in the simulation in Matlab using the same algorithm. The video from experiments is on the enclosed CD and sub-figures are shown in Figure 6.

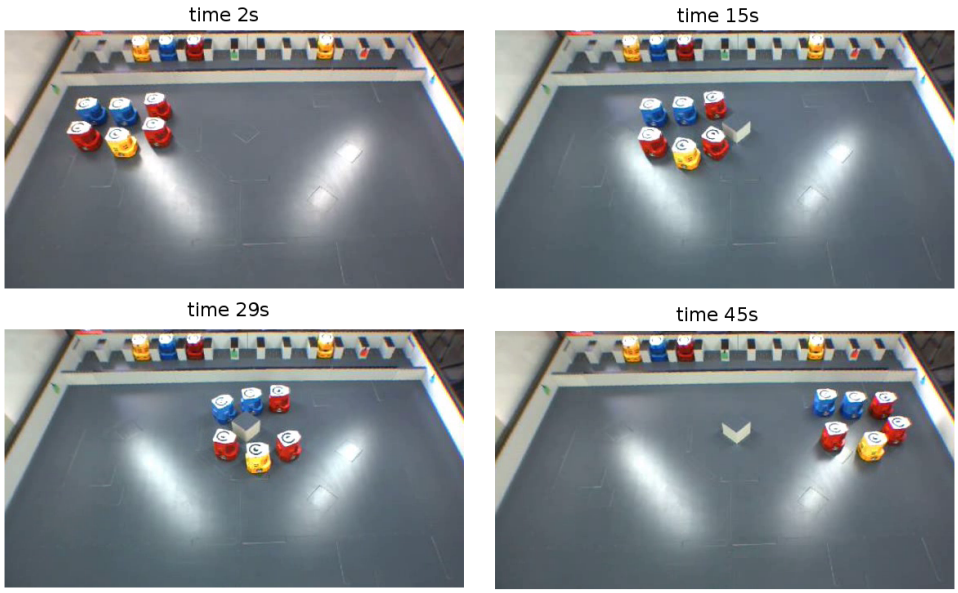

Figure 6: Sub-figures of experiments with real robots and one obstacle

2. Robots start in the corner and go to the opposite one. There are two dynamic obstacles extruded. The robots avoid the first obstacle, like in the previous experiment. The second obstacle is left closed to the planed trajectory and the flock avoids it together by turning right as shown in Figure 7.

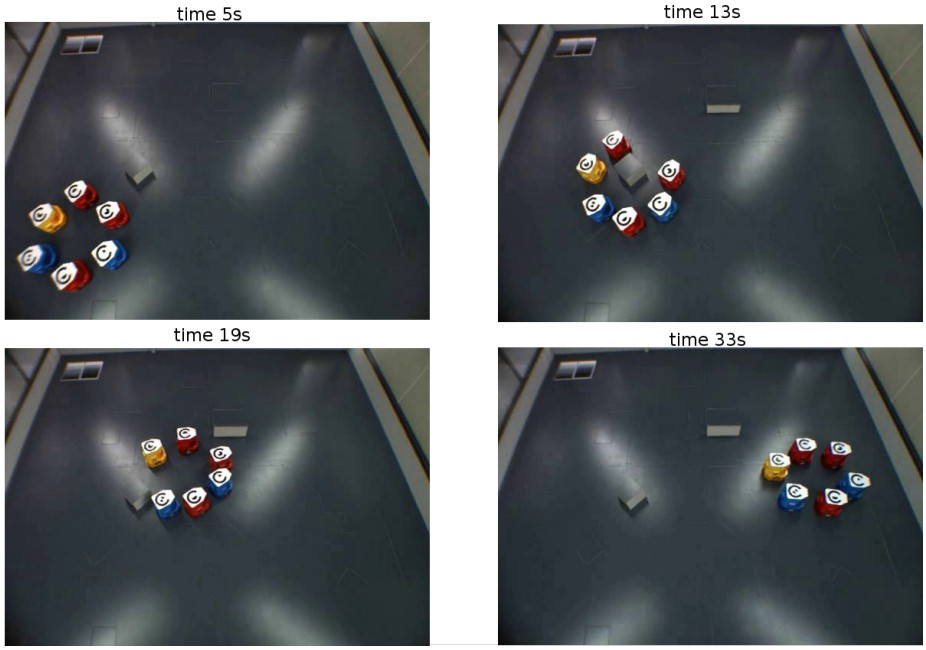

Figure 7: Sub-figures of experiments with real robots and two obstacles

4 3D

A quad-rotor unmanned aerial vehicle (UAV) is a multicopter that is lifted and propelled by four rotors, see Figure 8. Quad-rotors are classified as rotorcraft, as opposed to fixed-wing aircraft, because their lift is generated by a set of revolving narrow-chord airfoils. Two of the propellers spin clockwise and the other two counter-clockwise. Control of the machine can be achieved by varying relative speed of the propellers.
Quad-rotor has six degrees of freedom, where four of them are used for asymptotic tracking control, namely three position variables for the vehicle center of mass and the direction of one vehicle body-fixed axis.

Figure 8: Photo of quadrotor Parot AR.Drone

There are several advantages of quad-rotors over other flying vehicles.

- Simple mechanical structure does not require complex mechanical linkages such as swashplates or teeter hinges, which is common for helicopters. It incorporates only four electric motors and fixed-pitch rotors.

- Using four rotors allows each rotor to have smaller diameter. Small diameter rotor stores less kinetic energy than an equivalent single rotor, this reduces the seriousness of consequences of contact with other objects [7].
- Quad-rotor is easy to stabilize and control.
- It is capable of vertical take-off landing (VTOL).

To simplify implementation of the algorithm of escape behaviour to 3D, it was decided to solve this matter by dividing it into two sub-problems.

1. Extend the approach in [1] for a flock of quad-rotors.
   Actual position of all flock members as an input parameter for one program loop. Output is the required position of all quad-rotors, which is given as the input for controllers of each of the robots.
2. Implement the controller for one quad-rotor.
   Input parameter is the required position and output is the angular velocity of propellers. The controller is based on the paper [5].

At the beginning of the program loop every quad-rotor in the group calculates the required positions. Required positions are determined by

- the positions of all members which have local interaction,
- the positions of obstacles, if obstacles are in robot's sensor range,
- the position of the goal.

The equations are described and explained in section 5. The required position is the input parameter for the controller. The controller calculates lifting force and the moments, which quad-rotor needs to achieve the required position. The Lifting force and the moments can be transformed to speeds of rotors. The controller is described and explained in section 6.

5 Flocking equations in 3D

In this section the method of calculating the required position depending on surrounding of the i-th quad-rotor is developed. The surrounding acts on quad-rotors via forces. The forces are summed and then transformed to the required position. The form of the equations is developed intentionally to be similar to the equations in section 2.
The importance of all variables and constants used in this section is expressed in Table 3 and in Table 4.

Table 3: Table of variables used in 3D

variable	description
$\mathbf{R}_i$	Denotes the position vector of the i-th quad-rotor
H_i	Denotes the heading direction vector of i-th quad-rotor
$\mathbf{L}_{ji}$	$\mathbf{L}_{ij} = \mathbf{R}_i - \mathbf{R}_j$ denotes the relative position vector from the i-th to the j-th quad-rotor
$\mathbf{L}_{ig}$	The relative position vector from the i-th quad-rotor to the position of the goal
$\mathbf{L}_{oi}$	The relative position vector from the position of the obstacle, if the obstacle exists, to the position of i-th quad-rotor
Ψ_{io}	The angle between the vector $\mathbf{L}_{rb_i}$ and direction of the heading direction vector of i-th quad-rotor

5.1 Other Individuals effect

The effect that provides separation and cohesion is expressed by the equation (11). It represents their mutual interactions and ensures proper grouping of all individuals. In section 2 the effect is represented by the force $\mathbf{F}_{K_{ij}}$. The effect is expressed by

$$\mathbf{F}_{ind_i} = \sum_{j,j\neq i}^{N} e_{ij} \cdot \mathbf{F}_{ind_{ij}}, \tag{11}$$

where e_{ij} is the distance weight function. The magnitude of this function depends on the relative distance between quad-rotors L_{ij}. For the 3D algorithm the same weight function is chosen as the one used in section 2:

$$e_{ij} = \frac{1}{e^{a \cdot \mathbf{L}_{ij} - b} + c} + \frac{1}{e^{0.5 \cdot a \cdot \mathbf{L}_{ij} - b} + c}. \tag{12}$$

Example of this weight function is shown in Figure 9.

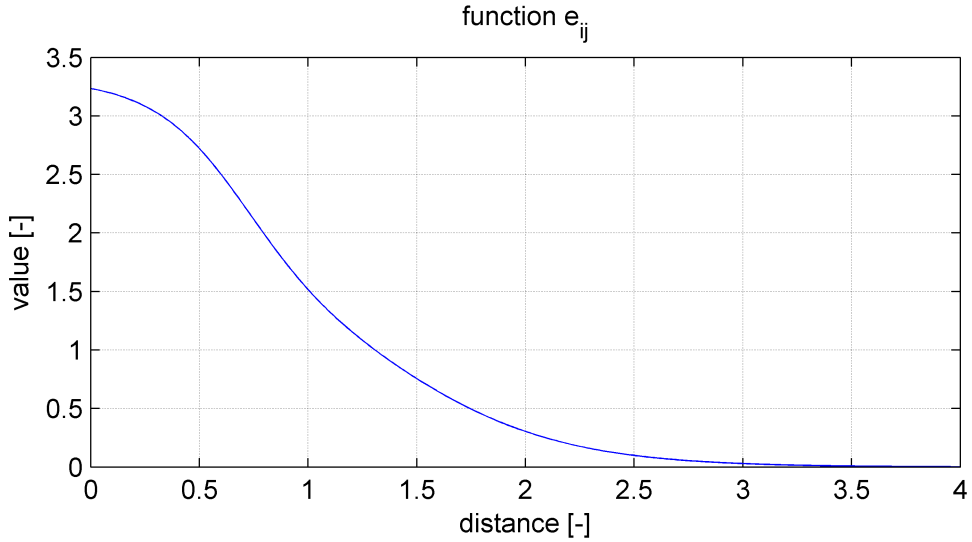

Figure 9: Graph for e_{ij}, where $a = 5$, $b = 4$, $c = 0.6$, $\mathbf{L}_{ij} \in < 0, 4 >$

The interactive force

$$\mathbf{F}_{ind_{ij}} = K_d(\|\mathbf{L}_{ij}\| - L_r)\mathbf{L}_{ij} + D_d\frac{\mathrm{d}\mathbf{L}_{ij}}{\mathrm{d}t} \tag{13}$$

is designed as a spring-damper model to let $\mathbf{L}_{ij}$ be fixed distance L_r.

5.2 Goal effect

The force that pushes the swarm into the goal isn't mentioned in the paper [1]. We upgraded this original algorithm by incorporating the goal effect. For practical usage it is convenient to be able to assign the quad-rotors a specific position, that should be reached. The interactive force pointing to the goal

$$\mathbf{F}_{goal_i} = K_g \cdot \mathbf{L}_{ig} + D_g \frac{\mathrm{d}\mathbf{L}_{ig}}{\mathrm{d}t} \tag{14}$$

is designed as a spring-damper model to let $\mathbf{L}_{ig}$ be distance 0. It is like a PD regulator that is set by constants K_g and D_g. The problem of this proposed solution is that $\mathbf{F}_{goal_i}$ has too large magnitude if quad-rotors are too far from the goal position. Therefore $\mathbf{F}_{goal_i}$ is set to constant magnitude and direction is set to the direction of the vector to the goal $\mathbf{L}_{ig}$ if the magnitude of $\mathbf{L}_{ig}$ is too large. $\mathbf{F}_{goal_i}$ is then expressed by

$$\mathbf{F}_{goal_i} = W_g \frac{\mathbf{L}_{ig}}{\|\mathbf{L}_{ig}\|}. \tag{15}$$

5.3 Obstacle effect

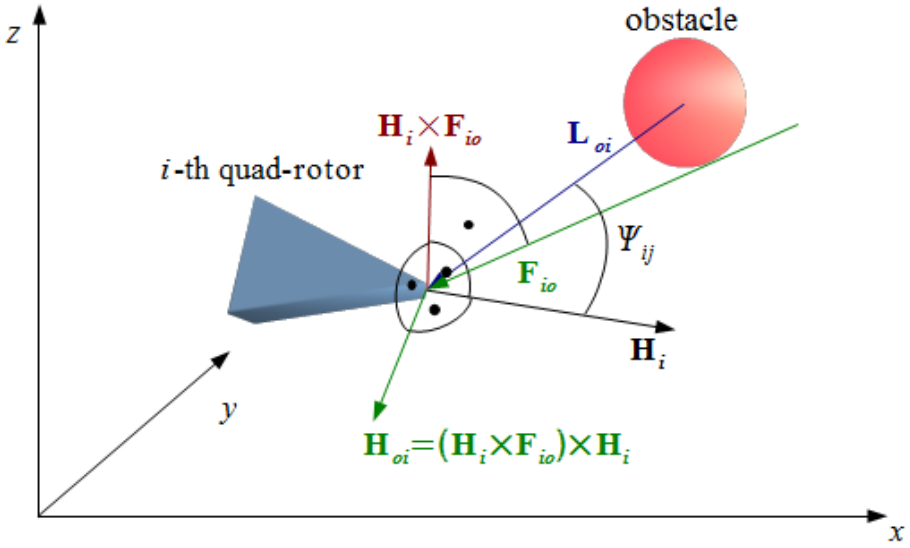

Figure 10: Schematically illustrated vectors near the obstacle

The swarm of quad-rotors performs the obstacle avoidance by the reshaping swarm by incorporating the equation

$$\mathbf{F}_{obs_i} = \delta \cdot e_{oi} \cdot \frac{\mathbf{H}_{oi}}{\|\mathbf{H}_{oi}\|} \tag{16}$$

to each quad-rotor. The magnitude of this force is decided by both the dependence function δ and the exponential distance function e_{oi}. The distance function is designed as

$$e_{oi} = b_o e^{a_o \|\mathbf{L}_{oi}\|} \tag{17}$$

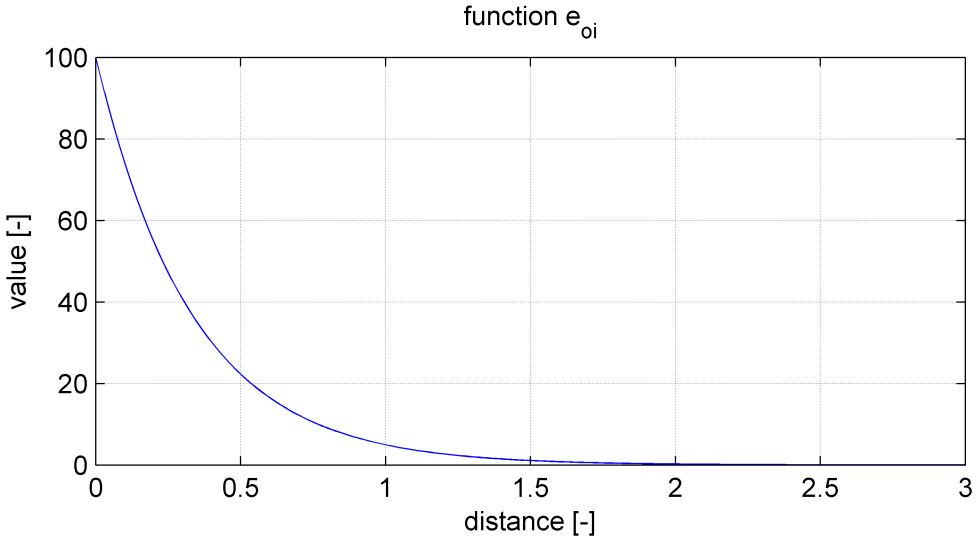

Figure 11: Graph for e_{oi}, where $a_o = -3$, $b_o = 100$, $\|\mathbf{L}_{oi}\| \in < 0, 3 >$

and the example of a graph is shown in Figure 11. The dependence function is designed as in section 2 as

$$\delta = (1 + d_o \cos(\Psi_{io})) \tag{18}$$

and the example of graph is shown in Figure 12.

$\mathbf{H}_{oi}$ is the direction vector, that points away from the obstacle and is perpendicular to the heading direction vector of i-th quad-rotor. The schematic figure with vectors is shown in Figure 10. The perpendicularity is ensured by crossproducts:

$$\mathbf{H}_{oi} = (\mathbf{H}_i \times \mathbf{F}_{io}) \times \mathbf{H}_i. \tag{19}$$

The vector acting away from the obstacle $\mathbf{F}_{io}$ is designed as

$$\mathbf{F}_{io} = K_o \cdot \mathbf{L}_{oi} + D_o \frac{\mathrm{d}\mathbf{L}_{oi}}{\mathrm{d}t}. \tag{20}$$

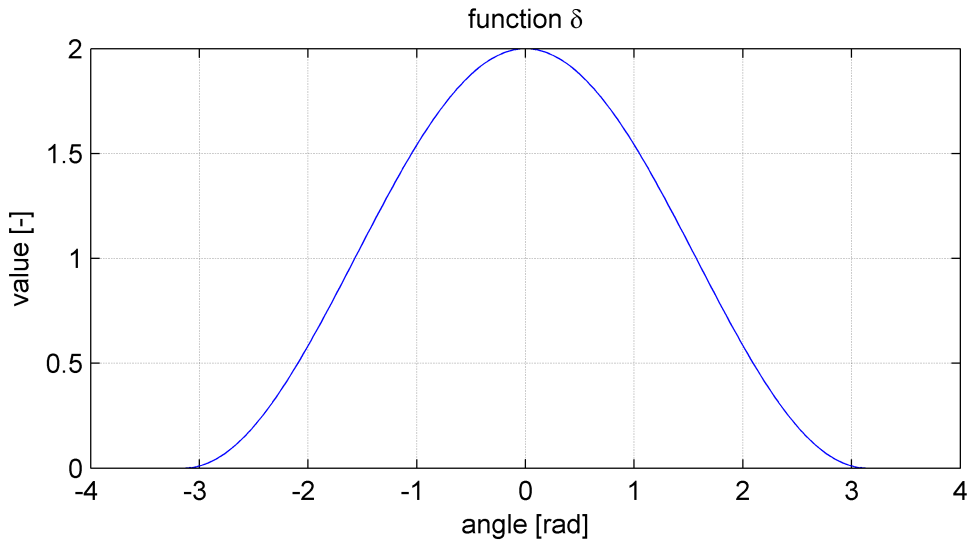

Figure 12: Graph for δ, where $d = 1$, $\Psi \in < -\pi, \pi >$

It's direction doesn't point exactly from the obstacle to the quad-rotor, but it deviates of this course. This deviation is caused by derivation, that predicts the position of the obstacle.

5.4 Sum of all effects

The total force, that acts on the i-th individual is determined by the weighted sum

$$\mathbf{F}_i = W_{ind} \cdot \mathbf{F}_{ind_i} + \mathbf{F}_{goal_i} + W_{obs} \cdot \mathbf{F}_{obs_i}. \tag{21}$$

The required position for the i-th quad-rotor is

$$\mathbf{X}_g = \mathbf{X}_i + C_{FtoD} \cdot \mathbf{F}_i, \tag{22}$$

where $\mathbf{X}_i$ denotes the actual position of the i-th quad and C_{FtoD} is a constant, that transforms force $\mathbf{F}_i$ to the vector.

The result of these equations is the required position that should be reached by the i-th quad-rotor. The required position is given to the regulator and the regulator sets the speed of the propellers to reach it. This algorithm runs simultaneously in all robots.

Table 4: Table of constants used in 3D

constant	description
K_d, D_d	Constants used in the equation (11). These constants tune the speed of reaction when quad-rotors divert from their optimal relative distance between quad-rotors L_r.
K_g, D_g	Constants used in the equation (14). These constants tune the speed of reaction when quad-rotors divert from the position of the goal.
K_o, D_o	Constants used in equation (14). These constants tune the speed of reaction when quad-rotors detect an obstacle.
W_g	Constant that set the constant magnitude to the force that pushes the swarm into the goal.
W_{ind}, W_{obs}	Weight constants used in the equation (21).
L_{d_i}	Constant, which represent the ideal distance between i-th quad-rotor and j-th quad-rotor.
C_{FtoD}	Constant, which transforms force to distance.
d	Constant used in the equation (18). This constant sets the magnitude of dependence function δ depending on angle.
a, b, c	Constants, which set distance function (12)
a_o, b_o	Constants, which set distance function (17)

6 Controller for 3D

In this section we elaborated quad-rotor vehicle model in Figure 13.

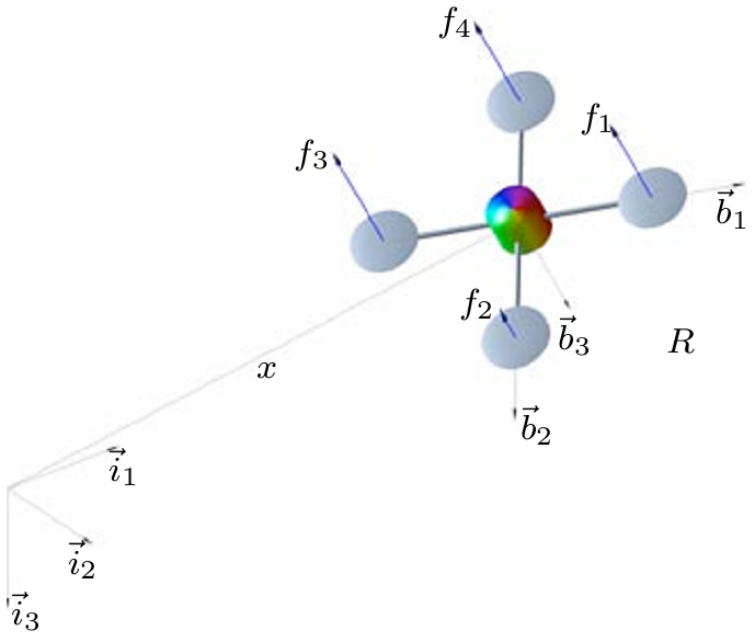

Figure 13: Illustrated model of quad-rotor [2]

This is a system of four identical rotors located at vertices of a square, which generates thrust and torque normal to a plane of this square. The body-fixed frame is chosen as $(\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3})$. The origin of the body-fixed frame is located in the center of the mass of this vehicle. $\mathbf{b_1}$ and $\mathbf{b_2}$ lie in the plane defined by centres of the four rotors. The body-fixed axis $\mathbf{b_1}$ is normal to this plane and points downwards, opposite to the direction of the total thrust. To develop the controller the dynamic of quad-rotors needs to be further elaborated.

6.1 Dynamic of quad-rotor

The dynamics of the quad-rotor is represented by two equations. The first describes translation, the second solves rotation.

1. Newton's second law of motion,

$$\mathbf{F} = m\mathbf{a}, \tag{23}$$

is used for translation. m denotes the mass of quad-rotor and $\mathbf{a} = \dot{\mathbf{v}} = \ddot{\mathbf{r}}$ denotes acceleration. $\mathbf{F} = -mg\mathbf{e_3} + f\mathrm{R}\mathbf{e_3}$ denotes the sum of all forces acting on the quad-rotor. R denotes the rotation matrix from the body-fixed frame to the inertial frame. f denotes total thrust generated by propellers.

2. Euler's second law is used for rotation:

$$\mathbf{J}\frac{\mathrm{d}\mathbf{\Omega}}{\mathrm{d}t} + \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega} = \mathbf{M}. \tag{24}$$

$\mathbf{J}$ denotes the inertia matrix with respect to the body-fixed frame. $\mathbf{\Omega}$ denotes the angular velocity in the body fixed-frame and $\mathbf{M}$ is the sum of all moments acting on the quad-rotor.

The rotation matrix R can be found by converting it from $\mathbf{\Omega}$ as

$$\frac{\mathrm{dR}}{\mathrm{d}t} = \mathrm{R}\hat{\mathbf{\Omega}}, \tag{25}$$

where

$$\hat{\mathbf{\Omega}} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{26}$$

The equation (25) is not appropriate for numerical computing, because after few steps the rotation matrix R will not be orthogonal. This problem is therefore solved using another method.

6.1.1 Numerical computing

Imagine a solid object, which rotates around the x, y and z axes, the angular velocities around these axes are ω_x, ω_y and ω_z. This rotation can also be represented by a single rotation around the axis $\omega = (\omega_x, \omega_y, \omega_z)$ (see Figure 14).

The angular speed is $\frac{\mathrm{d}\Psi}{\mathrm{d}t} = \omega = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$.

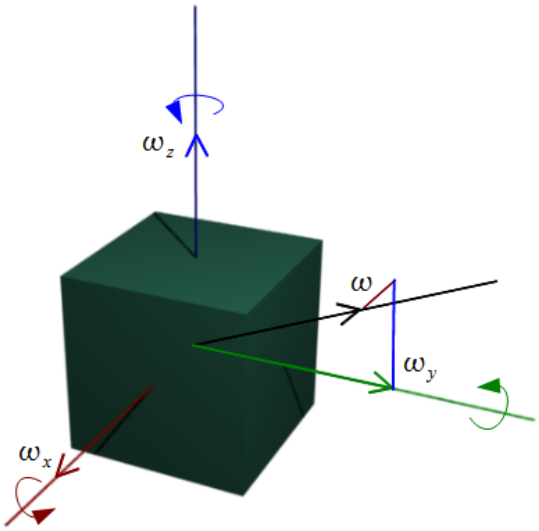

Figure 14: Rotation axes

Now we can calculate the angle of rotation Ψ, because ω is known. If the angle of rotation around axis ω is known, we can express the rotation matrix $\mathrm{R_w}$ around arbitrary axis ω [8]. The vectors ω_x, ω_y and ω_z in this matrix are normalised to a unit vector. The rotation matrix can be expressed as

$$\mathrm{R_w} = \begin{pmatrix} (1-c)\omega_x^2 + c & \omega_z s + \omega_x\omega_y(1-c) & -\omega_y s + \omega_x\omega_z(1-c) \\ -\omega_z s + \omega_x\omega_y(1-c) & (1-c)\omega_y^2 + c & \omega_x s + \omega_y\omega_z(1-c) \\ \omega_y s + \omega_x\omega_z(1-c) & -\omega_x s + \omega_y\omega_z(1-c) & (1-c)\omega_z^2 + c \end{pmatrix}, \quad (27)$$

where $c = \cos(\Psi)$ and $s = \sin(\Psi)$. The new rotation matrix is calculated from the old rotation matrix and $\mathrm{R_w}$.

$$\mathrm{R_{new}} = \mathrm{R_w} \cdot \mathrm{R_{old}} \quad (28)$$

6.2 Geometric tracking control on SE(3)

The controller is used to follow the prescribed trajectory of the location of the center of mass, and given direction of the heading vector. The controller is the same as used in [5].

6.2.1 Tracking errors

At the beginning it is convenient to define tracking errors for $\mathbf{x}, \mathbf{v}, \mathrm{R}, \boldsymbol{\Omega}$. In my case they are defined as follows:

The tracking errors for the position and the velocity are given by

$$\mathbf{e_x} = \mathbf{x} - \mathbf{x_d} \tag{29}$$

$$\mathbf{e_v} = \mathbf{v} - \mathbf{v_d} \tag{30}$$

The tracking errors for the rotation matrix and the angular velocity are given by

$$\mathrm{e_R} = \frac{1}{2}\left(\mathrm{R_d^T R} - \mathrm{R^T R_d}\right)^\vee, \tag{31}$$

$$\mathbf{e_\Omega} = \boldsymbol{\Omega} - \mathrm{R^T R_d}\boldsymbol{\Omega_d}, \tag{32}$$

where the *vee map* $^\vee : \mathbf{SO(3)} \to \mathbb{R}^3$ is the inverse of the hat map that is defined by condition $\hat{x}y = x \times y$ for all $x, y \in \mathbb{R}$

6.2.2 Tracking controller

The next thing to be defined is the desired rotation matrix $\mathrm{R_d}$ depending on $\mathbf{x_d}$ and the heading direction vector $\mathbf{b_{1_d}}$ The rotation matrix $\mathrm{R_d}$ also depends on the position error $\mathbf{e_x}$ and the velocity error $\mathbf{e_v}$, because if quad-rotors are not in the required position it has to be tilted to move. The tilt is contained in rotation matrix $\mathrm{R_d}$ that depends on the body-fixed axis $\mathbf{b_{3_d}}$. $\mathbf{b_{3_d}}$ is defined by

$$\mathbf{b_{3_d}} = \frac{-k_x\mathbf{e_x} - k_v\mathbf{e_v} - mg\mathbf{e_3} + m\ddot{\mathbf{x}}_\mathbf{d}}{\| - k_x\mathbf{e_x} - k_v\mathbf{e_v} - mg\mathbf{e_3} + m\ddot{\mathbf{x}}_\mathbf{d}\|}, \tag{33}$$

where

$$\mathbf{b_{3_d}} = \| - k_x\mathbf{e_x} - k_v\mathbf{e_v} - mg\mathbf{e_3} + m\ddot{\mathbf{x}}_\mathbf{d}\| \neq 0. \tag{34}$$

For given $\mathbf{b_{1_d}}$ and calculated $\mathbf{b_{3_d}}$, $\mathbf{b_{2_d}}$ is expressed from the definition of the Cartesian coordinate system. $\mathbf{b_{2_d}}$ has to be a unit vector of the crossproduct of two perpendicular axes $\mathbf{b_{3_d}}$ and $\mathbf{b_{1_d}}$.

$$\mathbf{b_{2_d}} = \frac{\mathbf{b_{3_d}} \times \mathbf{b_{1_d}}}{\|\mathbf{b_{3_d}} \times \mathbf{b_{1_d}}\|} \tag{35}$$

The desired rotation matrix is given by

$$\mathrm{R_d} = (\mathbf{b_{1_d}}, \mathbf{b_{2_d}}, \mathbf{b_{3_d}}) \,. \tag{36}$$

Once we calculate all the errors and the desired position, we can determine forces and moments that the propellers have to create to reach the required position as

$$f = -\left(-k_x \mathbf{e_x} - k_v \mathbf{e_v} - mg\mathbf{e_3} + m\ddot{\mathbf{x}}_\mathbf{d}\right) \cdot \mathrm{R}\mathbf{e_3}, \tag{37}$$

$$M = -k_R \mathrm{e_R} - k_\Omega + \mathbf{\Omega} \times \mathbf{\Omega} - J\left(\hat{\mathbf{\Omega}}\mathrm{R^T R_d}\mathbf{\Omega_d} - \mathrm{R^T R_d}\dot{\mathbf{\Omega}}_\mathbf{d}\right). \tag{38}$$

Furthermore, we can calculate the force, that has to be generated by each propeller. We assume that the total thrust f and the total moment M can be written as

$$\begin{bmatrix} f \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -d & 0 & d \\ d & 0 & -d & 0 \\ -c_{\tau f} & c_{\tau f} & -c_{\tau f} & c_{\tau f} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}, \tag{39}$$

where f_i is the force which generates the i-th propeller and d is the distance from the center of mass to the center of propeller.
The propeller generates torque and $c_{\tau f}$ is constant, which transforms kinetic energy of the propeller to moment M_3.

When we have thrust of each propeller we can calculate the rotation speed of the propeller. If the speed comes out too high we set it to maximum.

7 Experiments and simulations in 3D

7.1 Controller

The controller is tested only for one quad-rotor. At the beginning of the experiment the quad-rotor is placed to the origin of the coordinate system and the heading vector is directed to $(1, 0, 0)$. The goal position is set to $(2, -10, 1)$ and the goal direction of heading vector is set to $(0, -1, 0)$. In Figure 15 there is a quad-rotor illustrated during the experiment. Figure 16 shows the graphs of the x,y,z positions, euler angles, velocities and

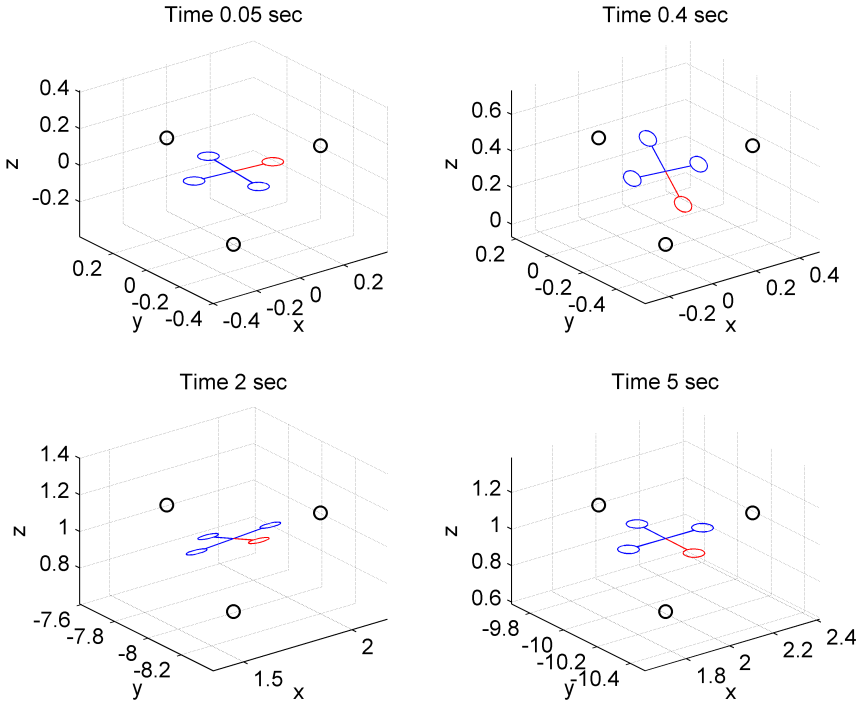

Figure 15: Sub-figures of the simulation

angular velocities. In the graph of positions, there is shown that the quad-rotor reached the goal position. The graph of euler angles is the evidence that quad-rotor rotated to the required direction. Figure 17 shows graphs of the moments, the forces generated by each rotor and the total force. In the graph of forces there is shown, that f_1,f_2,f_3 and f_4 are limited to the value of 0 from below and to the value of 30 from above. The properties of

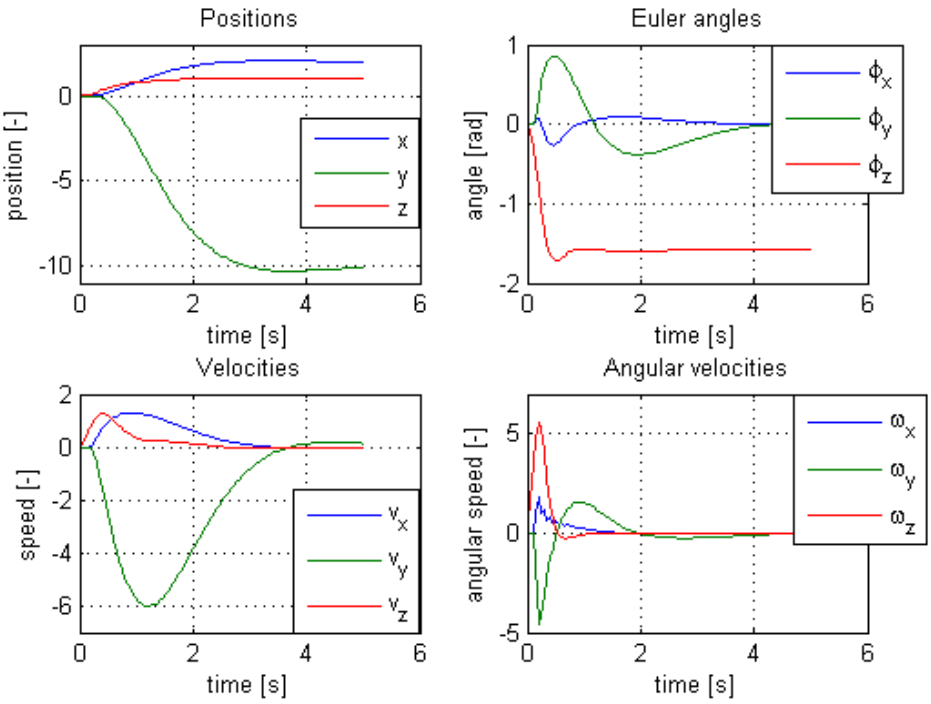

Figure 16: Graphs of positions, velocities, euler angles and angular velocities

the model such as mass, moment of inertia etc. is set as in the paper [2]. The limits are set to this value to show the functionality of the system in real applications. The actual value can't be measured because real quad-rotors are not available. The setting of values of the controller's constants are inspired by the paper [2]. Some of them are a little increased for shorter rise time and smaller overshoot.

7.2 Swarm experiments

1. In the first experiment, we used 27 quad-rotors that start in a formation around the origin of the coordinating system. There is one obstacle placed in the position $(5, 0, 0)$. The goal position is at $(10, 0, 0)$. At the beginning of the simulation the quad-rotors increase their relative distance to the pre-set value and move like a flock. After a few steps they detect an obstacle and the flock splits into groups to avoid the obstacle. The distance to the obstacle of all the quad-rotors is shown in the graph in Figure 18. The graph of distance of all quad-rotors to the closest neighbour is shown

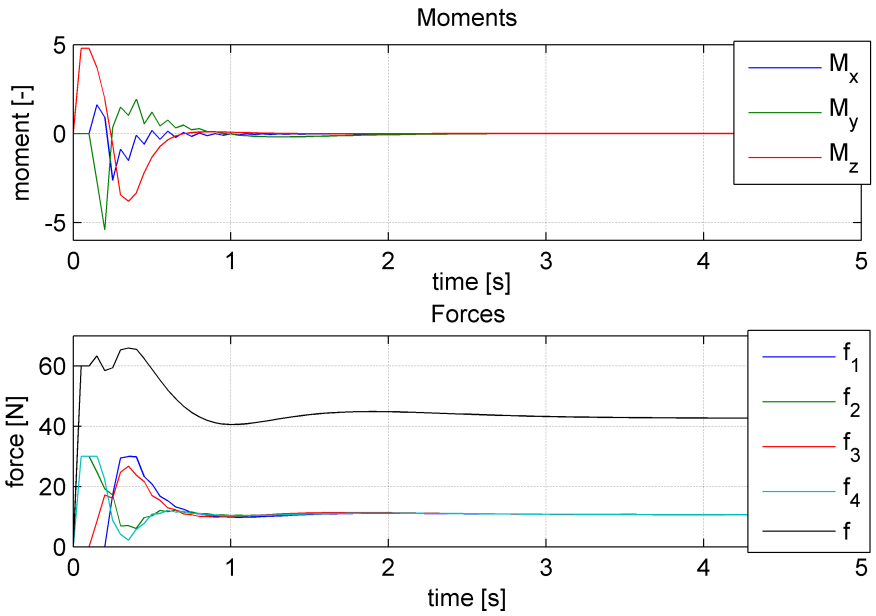

Figure 17: Graphs of moments and forces

in Figure 19. The simulation in sub-figures is shown in Figure 20.

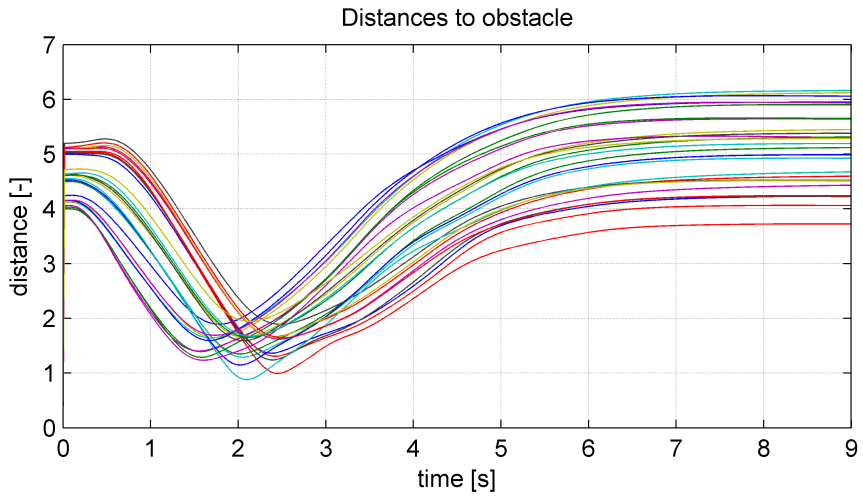

Figure 18: Graph of distances to the obstacle of experiment 1.

2. In the next simulation we use the same number of quad-rotors, 27. The start and the goal position is set to the same value as in the previous experiment. The difference

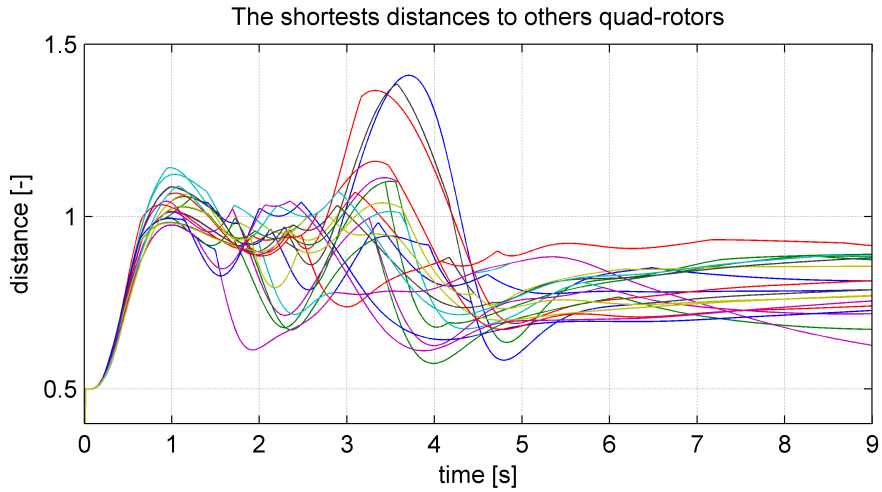

Figure 19: Graph of distances to the closest neighbour of experiment 1.

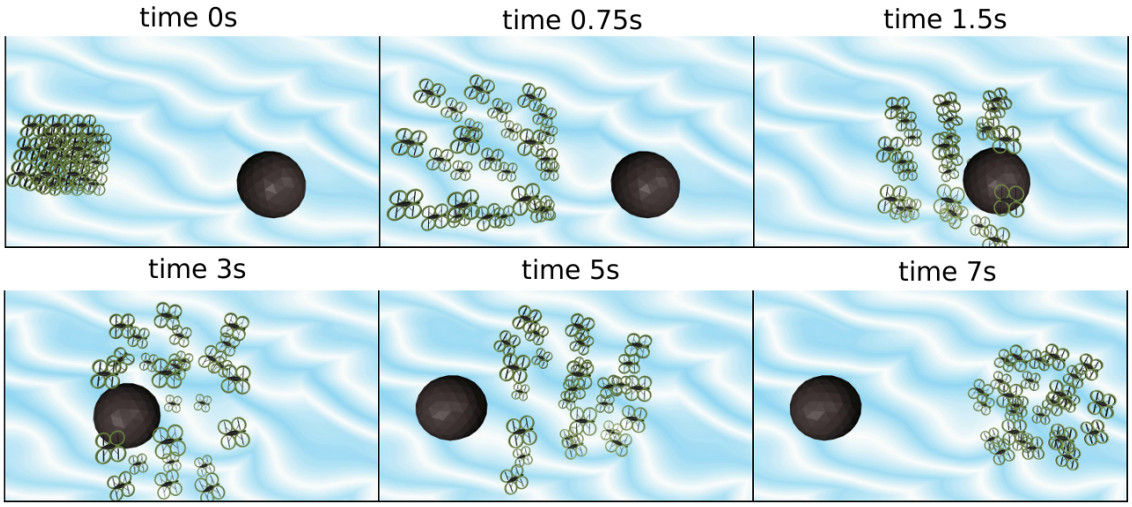

Figure 20: Sub-figures of experiment 1.

is that there are three obstacles. The first one is placed at $(5, 0, 0)$, the second one at $(5, 0, 2)$ and the third one at $(7, 2, 0)$. In this experiment the quad-rotors avoid the obstacles as expected. The sub-figures from experiment is shown in the Figure 23. The distance to the obstacle of all the quad-rotors is shown in graph in Figure 21. The graph of distance of all quad-rotors to the closest neighbour is shown in Figure 22.

3. The next experiment deals with 10 quad-rotors. The start, the goal and the obstacles

positions are set to the same value as in the previous experiment. The experiment shows that the algorithm can control different number of swarm members. The sub-figures from the experiment is shown in Figure 25. The graph of the distance of all quad-rotors to the closest neighbour is shown in Figure 24.

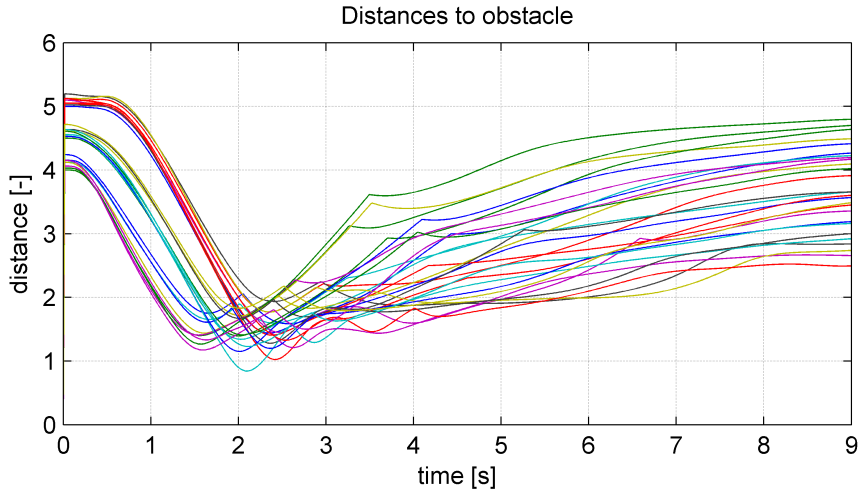

Figure 21: Graph of distances to the obstacle of experiment 2.

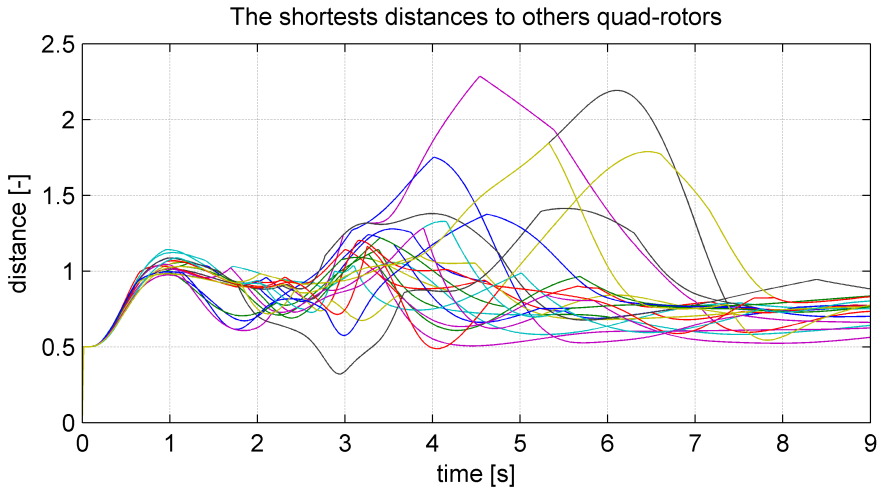

Figure 22: Graph of distances to the closest neighbour of experiment 2.

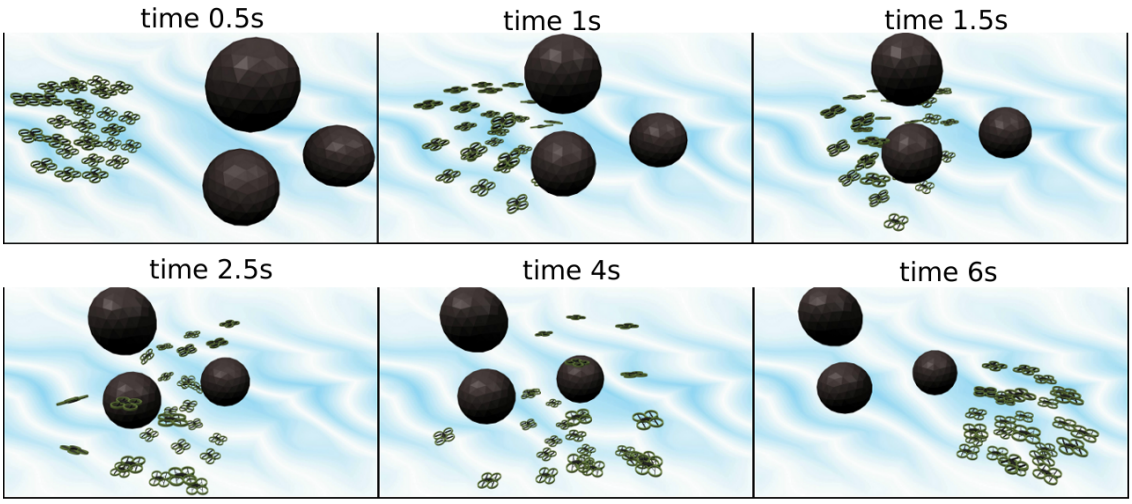

Figure 23: Sub-figures of experiment 2.

Figure 24: Graph of the distances to the obstacle of experiment 3.

7.3 Blender

Blender is used to create visualisation of the developed algorithm for a flock of quad-rotors [9]. Blender is the 3D visualisation software for creating 3D scenes. The trajectories of the quad-rotors are generated in Matlab and transferred into Blender by Python script. The script mapped generated trajectories to 3D model of quad-rotors. The Blender renders a video sequences of the scene.

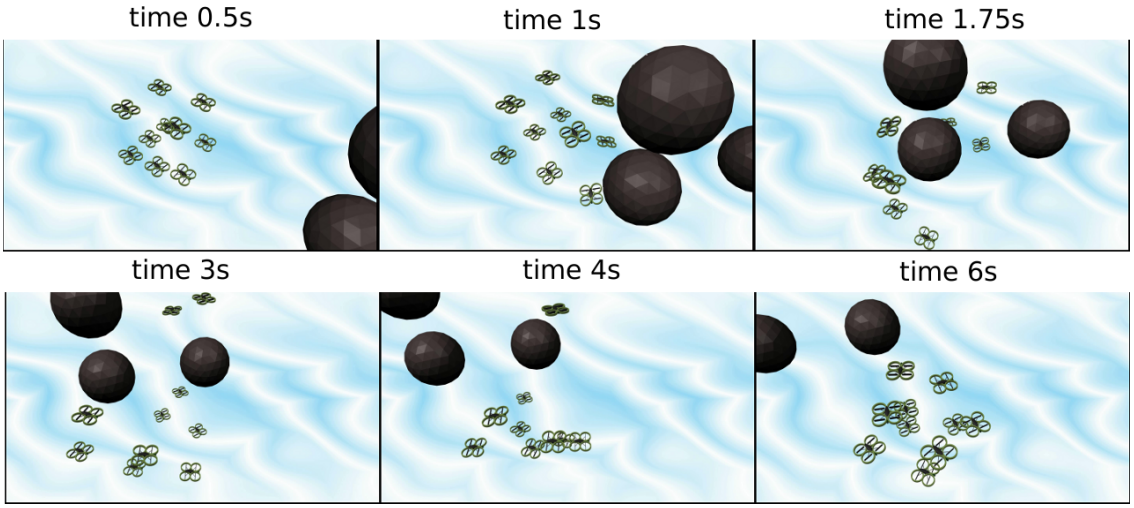

Figure 25: Sub-figures of experiment 3.

8 Conclusion

In this thesis the algorithm of escape behaviour for swarm of UAV was successfully developed. The algorithm decreases the possibility of collision of dynamic obstacles with the swarm of the quad-rotors. This algorithm could be implemented into controllers of real quad-rotors and used for shifting the whole swarm from point A to point B. The functionality and the design of the algorithm were theoretically tested in 2D with models of UGVs before developing the algorithm for quad-rotors. Theoretical tests showed that the algorithm can be used for the problem of dynamic obstacle avoidance. The algorithm for UGVs was implemented into SyRoTek for test with real hardware.

The experience gained during developing and testing the algorithm for UGVs was used to develop the algorithm for UAVs. Before implementation the algorithm for swarms the tracking controller for one quad-rotor had to be developed. One restriction for real robots was implemented in the controller. The restriction affects lifting force generated by the rotors. The controller was tested in a simulation in Matlab and worked well. The whole algorithm for swarms was simulated and tested in Matlab as well. The Blender was used for visualisations. The sequences from simulations are shown in section 7.

To sum up, all the thesis objectives were successfully accomplished:

- The extension of escape behaviour, [1], to 3D was designed and implemented in section 6
- The model of UAV from [5] was integrated in section 5.
- The developed method was prepared to integrate into the control system of autonomous swarm.
- The implemented system was verified with simulation of movements of 3D swarm in section 7.
- The functionality of the method of 2D algorithm was demonstrated with a system of autonomous robots SyRoTek [3] in section 3.

The algorithm for 3D is fully functional and it is prepared for real deployment. It has fulfilled all expectations, but I have a few ideas that could be further elaborated.

- The algorithm considers only point obstacles. This is sufficient for simple scenarios but not for real-life situations. Improvement of the algorithm might be developing of an algorithm for obstacles of real size and shape.
- To implement robot's reaction to the change of the heading vectors of the others quad-rotors, like $M_{c_{ij}}$ in section 2. This could improve the speed of reaction for obstacle avoidance.
- Real parameters of quad-rotors couldn't be measured, because they were not available.

I hope that the algorithm is robust and it will be used in real quad-rotors for shifting quad-rotors swarm.

References

[1] H. Min and Z. Wang, "Design and analysis of group escape behavior for distributed autonomous mobile robots.," in *ICRA*, pp. 6128–6135, IEEE, 2011.

[2] M. LaLena, "Flocking behavior simulator," *http://www.lalena.com/AI/Flock/*, 1996-2012.

[3] J. Faigl, J. Chudoba, K. Košnar, M. Kulich, M. Saska, and L. Přeučil, "Syrotek-a robotic system for education," *In Proceedings of Robotics in Education*, 2010.

[4] P. SA, "Ar.drone," *http://ardrone.parrot.com/parrot-ar-drone/en*, 2012.

[5] T. Lee, M. Leoky, and N. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 5420–5425, IEEE, 2010.

[6] J. Chudoba, J. Faigl, M. Kulich, T. Krajnık, K. Košnar, and L. Preucil, "A technical solution of a robotic e-learning system in the syrotek project,"

[7] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *Proceedings of the AIAA guidance, navigation, and control conference*, vol. 4, p. 44, 2007.

[8] M. Chen, S. Mountford, and A. Sellen, "A study in interactive 3-d rotation using 2-d control devices," in *ACM SIGGRAPH Computer Graphics*, vol. 22, pp. 121–129, ACM, 1988.

[9] G. G. P. License, "Blender," *http://www.blender.org/*, 2012.

Appendix A : Evaluation of SyRoTek

After eliminating all the problems I evaluate working with SyRoTek fairly well. I commend that I was able to test my algorithms from home via the Internet. The schematic view on SyRoTek web page was sufficient for primitive orientation. The reservation system also performed well, except of the button for inviting others students to my time slot, which didn't work. I have some comments about the SyRoTek system.

- I had problems installing the required software to my own distribution of Linux. The instructions posted on SyRoTek web pages didn't work for me. I tried to find a solution for my problems on Player-Stage web pages, but it took me a lot of time and it didn't work. Finally I used a pre-installed Linux distribution on a virtual machine. The disadvantage of this solution is that the virtual machine was running too slow, especially when I started Stage. In the virtualized Linux, I had also problems with display resolution and font, which was difficult to read. While working on the computers at school It was necessary to use a virtual machine to run Player/Stage, because students do not have the privileges to install new software.
  I suggest to install the required software for SyRoTek onto the computers at school or write more detailed user manual for installation onto my distribution of Linux operation system.

- Another helpful thing for beginners could be a prepared template of an easy program with basic functions. For example the program should be able to move robots straight forward. The communication between the robots should be solved as well. It would be good to post this template on the SyRoTek web site. We got the program from one Ph.D. student, but it was too complicated with lot of unnecessary functions for us.

- At the beginning of work I had problems with the preparation of six robots that I used for my programs. Sometimes one of the robots would stay at the charging station and would report malfunction. Over the time this problem was solved.

- The next problem, which we solved during our work, was controlling the moving obstacle. Teachers gave me the privileges and provided me a functional and simple sample C++ code. It would be good to post this sample code on SyRoTek web site too.

- To develop a system which would enable students to download video sequences that are recorded during their time slot. Now students have to go to the teachers and ask for the records. It is annoying for teachers as well as for students. All records could be deleted after some time, for example a week.

I would like to thank all the teachers and students who helped me with SyRoTek, especially to Ing. Jan Chudoba, who was very helpful and always ready to fix the problems with the arena.

Appendix B : Contents of the enclosed CD

- `/`
  - `bachelor_thesis.pdf` ..... complete Bachelor Thesis in PDF
  - `2D_simulations/`
    - `simulation.m` ..... main m-file, which starts the 2D simulation
    - `animate.m, create.m, plot_ind.m` ..... supporting m-files for the visualisation of the simulation
  - `3D_simulations/`
    - `controller/` ..... this directory holds m-files for the simulation of the controller
      - `sim_movement_quad.m` ..... main m-file, which starts the controller simulation
      - `SetParameters.m` ..... m-file with the set parameters
      - `plotGraphs.m, drawQuad.m, createQuadModel.m` ..... supporting m-files for visualisation of the simulation
      - `create3D.m, calculateRw.m, derivation.m, saveAllVariables.m, RtoE.m, inverseVee.m, vee.m` ..... supporting m-files
      - `BoundedControl.m` ..... m-file with the set restrictions
      - `regulator.m` ..... m-file computing regulator calculations
    - `swarm/` ..... this directory holds m-files for the simulation of swarm movement, some of them are the same as in the controller folder
      - `sim_movement_swarm.m` ..... main m-file, which starts the controller simulation
      - `forceToObst.m` ..... function calculating the repulsion force from the obstacles
      - `forceToInd.m` ..... function calculating the force to the others robots
      - `forceToGoal.m` ..... function calculating attractor force to goal
      - ...
  - `SyRoTek/` ..... this directory holds files for SyRoTeK
    - `robot.cc` ..... main file of the algorithm
    - ...

- `videos/` ..... this directory holds videos of the experiments
  - `2D_matlab.avi` ..... video from the experiment in section 2
  - `controller.avi` ..... video from the experiment with the controller from section 6
  - `swarm_10i_1obs.avi` ..... video from experiment 3 in section 7 with 10 quad-rotors and one obstacle
  - `swarm_14i_1obs.avi` ..... video with 14 quad-rotor and one obstacle
  - `swarm_27i_1obs.avi` ..... video from experiment 2 in section 7 with 10 quad-rotor and one obstacle
  - `swarm_27i_3obs.avi` ..... video from experiment 1 in section 7 with 27 quad-rotor and three obstacles
  - `syrotek_experiment_1.avi` ..... video from SyRoTek experiment 1
  - `syrotek_experiment_2.avi` ..... video from SyRoTek experiment 2