Bachelor thesis

Algorithms for navigation of swarm of unmanned helicopters and following a dynamic target in a complex environment

Tomáš Sekanina

April 2014

Thesis supervisor: **Ing. Martin Saska, Dr.rer.nat.**

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Tomáš S e k a n i n a
Study programme:	Cybernetics and Robotics
Specialisation:	Robotics
Title of Bachelor Project:	Algorithms for Navigation of Swarm of Unmanned Helicopters and Following a Dynamic Target in a Complex Environment

Guidelines:

The aim of this thesis is to design, implement and verify an algorithm for stabilization and navigation of a swarm of MAVs that enables to follow a moving target in an environment with complex obstacles.
Work plan:

- To study the method [1] (for MAV adjusted in [2]) and to extend it for utilization in the task of following a moving target by a swarm of MAVs
- To study a method, which enables to represent arbitrary obstacles as a set of convex polygons and to effectively compute distance between MAVs and obstacles [5,6]
- To integrate this method into the swarm control system
- To verify the implemented system with simulations of movement of the MAV swarm in an office-like environment; to study the swarm behavior in narrow corridors and to improve the developed algorithm based on results of the simulations (to avoid mutual collisions between MAVs)

Bibliography/Sources:

[1] H. Min, Z. Wang: Design and analysis of Group Escape Behavior for distributed autonomous mobile robots. IEEE International Conference on Robotics and Automation (ICRA), 2011.
[2] M. Saska, J. Vakula, L. Preucil: Swarms of micro aerial vehicles stabilized under a visual relative localization. IEEE International Conference on Robotics and Automation (ICRA), 2014.
[3] T. Lee, M. Leok, N.H. McClamroch: Geometric tracking control of a quadrotor UAV on SE(3). IEEE Conference on Decision and Control (CDC), 2010.
[4] S. M. LaValle: Planning algorithms. Cambridge University Press, 2006.
[5] P. Jiménez, T. Federico, T. Carme: 3D collision detection: a survey. Computers & Graphics, 25(2), 269-285, 2001.
[6] M. C. Lin, D. Manocha: Collision and proximity queries. CiteSeer, 2003.

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

Valid until: the end of the summer semester of academic year 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
Head of Department	**Dean**

Prague, January 10, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Tomáš S e k a n i n a
Studijní program:	Kybernetika a robotika (bakalářský)
Obor:	Robotika
Název tématu:	Algoritmy pro navigaci roje bezpilotních helikoptér a sledování dynamického cíle v komplexním prostředí

Pokyny pro vypracování:

Cílem práce je navrhnout, implementovat a experimentálně verifikovat algoritmus pro stabilizaci a navigaci roje umožňující sledovat pohybující se cíl v prostředí se složitými překážkami.
Plán prací:

- Nastudovat metodu [1] (pro MAV upravenou v [2]) a rozšířit ji o možnost sledování pohybujícího se cíle rojem MAV.
- Nastudovat metodu umožňující reprezentovat libovolně složité prostředí množinou konvexních polygonů a efektivně počítat vzdálenost MAV a překážek [5,6].
- Integrovat tuto metodu do systému řízení roje.
- Ověřit výsledný systém simulací pohybu roje v kancelářském prostředí; studovat chování roje v úzkých koridorech a na základě výsledků simulací uzpůsobit navržený algoritmus tak, aby se zabránilo vzájemným kolizím mezi letouny.

Seznam odborné literatury:

[1] H. Min, Z. Wang: Design and analysis of Group Escape Behavior for distributed autonomous mobile robots. IEEE International Conference on Robotics and Automation (ICRA), 2011.
[2] M. Saska, J. Vakula, L. Preucil: Swarms of micro aerial vehicles stabilized under a visual relative localization. IEEE International Conference on Robotics and Automation (ICRA), 2014.
[3] T. Lee, M. Leok, N.H. McClamroch: Geometric tracking control of a quadrotor UAV on SE(3). IEEE Conference on Decision and Control (CDC), 2010.
[4] S. M. LaValle: Planning algorithms. Cambridge University Press, 2006.
[5] P. Jiménez, T. Federico, T. Carme: 3D collision detection: a survey. Computers & Graphics, 25(2), 269-285, 2001.
[6] M. C. Lin, D. Manocha: Collision and proximity queries. CiteSeer, 2003.

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

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic	prof. Ing. Pavel Ripka, CSc.
vedoucí katedry	**děkan**

V Praze dne 10. 1. 2014

Acknowledgement

I would like to thank all the people who helped me during work on this thesis. In the first place to my thesis supervisor Ing. Martin Saska, Dr. rer. nat. for professional guidance and recommendations and secondly to Ing. Zdeněk Kasl for helping me to overcome all problems in the beginning. Further I have to thank my girlfriend Eva for big support as the deadline was approaching and my thanks also belongs to Bc. Martin Bláha for providing me with his algorithm.

Abstract

Cílem této bakalářské práce bylo nastudovat metodu [2] escape behaviour a využítj i pro sledování pohyblivého cíle rojem MAV. Řídící model použitý v této metodě byl původně navrhnut pro fungování s překážkami ve tvaru koule a tak má být do tohoto algoritmu integrována metoda pro počítání vzdálenosti mezi MAV a libovolnou překážkou. Na závěr má být ověřena tato implementace simulacemi pohybu roje v kancelářském prostředí a prozkoumáno chování roje v úzkých uličkách. Vyvinutý algoritmus má být upraven tak, aby nedocházelo ke kolizím mezi jednotlivými letouny.

Abstract

Aim of this bachelor thesis was to study method [2] of escape behaviour and utilized it in a task of following a moving target by a swarm of MAVs. The control model used by this method was originally designed only for obstacles in shape of sphere and so into the algorithm is to be integrated a library for computing distance between the MAV and arbitrary obstacles. And finally to verify this implementation with simulations of movement of the swarm in an office environment and study behaviour of the swarm in narrow corridors. The developed algorithm should be improved to avoid collisions between individual quad-rotors.

Contents

1 Introduction

There was made a bachelor thesis by Jan Vakula [1] in 2012 describing amongst other things control model for swarm of unmanned aerial vehicles (UAV) - quadrocopters to be precise. This model was also implemented so it can be used for simulating of behaviour of the swarm. It is designed to be a distributed algorithm, which means that each quad-rotor does his own computations according to its surroundings and does not communicate with rest of the swarm. The control model is inspired by behaviour of schooling fish. It means that each individual in the swarm reacts on the movement of others visible to it.

The algorithm as it stands can simulate movement of the swarm towards defined goal across obstacles in 3D space. It is possible to configure number of quad-rotors in the swarm, coordinates of obstacles, initial position of the swarm and goal. Important limitation of this algorithm is, that it can work only with obstacles in shape of sphere.

This thesis is based on the mentioned algorithm, and its main aim is to make it more flexible, so it can be used in real devices one day. Obviously, such application needs to be able to work with more than only sphere obstacles. For this purpose ability to represent complex environment and compute distances from the obstacles is needed. There are various libraries already implemented able to this. As the most suitable for this project seem to be GJK library.

For the drones to be able to move between complex obstacles, especially indoor where are smaller spaces with lot of objects, it is often not possible to use static goal. It is so because even though the UAVs are able to detect obstacle in front of them, if it is large, they might not be able to effectively avoid it, and possibly not at all. Apart from that, it is sometimes desirable that the UAVs move along predefined path. That is why implementing methods for moving target is another goal of this thesis. Furthermore, to be efficient, the algorithm has to be also able to find possible trajectory trough set of control points.

To even start planning, the environment firstly needs to be represented as a graph. Afterwards, using some planning method can be found a path. There are different methods designed for this but their results still vary a lot. It is desirable that the route keeps distance from obstacles rather than being just shortest possible. Suitable graph for this application is a structure called Voronoi diagram. Edges of this graph are created in a way that they have same distance from the neighbouring points. It means that if the path was bounded by two obstacles, it would lead straight in between them.

Another step is choosing right planning algorithm. There are more those that would suffice but the best choice seems to be A* algorithm. It is not difficult to implement and yet fast, compared to other common planning algorithms such as BFS.

Library combining both generation of Voronoi diagram and path planning using A* was created by Bc. Martin Bláha as a part of his dissertation. This library is used in this thesis for computing route for the goal.

Firstly the swarm behaviour model had to be rewritten from Matlab to C++ because the other libraries are written in this language. And after this, all these three parts had to be adjusted so they can be merged together. Also functions for easier setting of the environment were implemented. There were made a series of simulations testing behaviour of this algorithm in different situations. Especially on movement in corridors because there appeared some difficulties at the beginning. So this thesis also focuses on enhancing the control algorithm to avoid collisions which occurred in some cases before.

As the thesis had been written there was added one more point as a request from another teacher interested in this field. And that is to explore influence of width of the corridor and number of MAVs in it on shape of the swarm. In ideal case, find some specific function describing it. These data could be used later for deciding which path should the swarm take when choosing between several corridors or whether it would no be efficient to split.

2 Swarm

The thesis mentioned in Introduction [1] contains implementation of control model of physics of the UAVs and also model for its behaviour as part of the swarm. Both these models are described in [2] and briefly introduced in the following paragraphs.

2.1 Quadrocopter

Experiments with quadrocopters are quite popular nowadays and many research teams are working with them. Even though its control is one of the more difficult speaking of aerial vehicles, it has several positives:

- It has no limitation on minimum flying speed as a plane does.
- It is capable of precise movement (depending on its controller and hardware). And also can be stabilized on certain position.
- It has a good maneuverability because it can change directions almost instantly and is capable fo vertical take-off and landing.
- It has simple mechanical structure compared to typical helicopter.

It is suitable for many applications such as cooperative surveillance, reconnaissance and monitoring tasks, search and rescue missions, searching for sources of pollution, sensory data acquisition and various military applications [2]. Because of its maneuverability it is also usable indoors where it can encounter narrow corridors or acute turns.

It usually has four identical propellers aligned into square where one pair perpendicular to the other one spins clock-wise and the other pair in opposite direction. This UAV has 6 degrees of freedom: three translational and three rotational(yaw, pitch and roll). These movements are controlled by 4 inputs - propellers motors which can be independently controlled. Model of such quad-rotor is defined by following differential equations and illustrated in figure 1. All symbols are described in table 1.

$$\dot{x} = v$$

$$m\dot{v} = mge_3 - fRe_3$$

$$\dot{R} = R\hat{\Omega}$$

$$J\dot{\Omega} + \Omega \times J\Omega = M$$

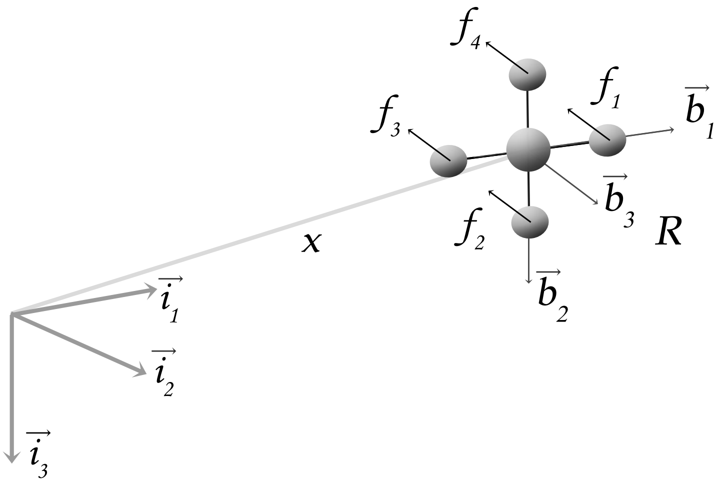


Figure 1 Quadrotor model

Symbol	Description
$\{\vec{i}_1, \vec{i}_2, \vec{i}_3\}$	reference frame
$\{\vec{b}_1, \vec{b}_2, \vec{b}_3, \}$	the body-fixed frame
$f_i \in R^3$	the thrust of the i-th propeller along the $-\vec{b}_3$
$f = \sum_{i=1}^{4} f_i$	the total thrust of all motors
$x \in R$	the location of centre of the mass m
$M \in R$	the total moment in the body-fixed frame
$R \in \mathtt{SO(3)}$	the rotation matrix from the body-fixed frame to the inertial frame
$J \in R^{3\times3}$	the inertia matrix with respect to the body-fixed frame
hat map $\hat{\cdot}$	$R^3 \to \mathfrak{so}(3)$ is defined so that: $\hat{x}y = x \times y$ for all $x, y \in R^3$
$\Omega \in R^3$	the angular velocity in the body-fixed frame

Table 1 Table of symbols used in equations in the UAV control model.

2.2 Flocking equations

Another thing to be discussed is stabilization and control of the whole swarm. The equations stated below are based on a control technique proposed in [2] which is inspired by BOID model [3]. The BOID model was developed to simulate schooling fish behaviour. The technique from [2] is based on visual relative localization between members of the swarm. It means that each individual is controlling itself and moves according to movement of its neighbours which is the same as can be seen in fish schools. This behaviour enables the quad-rotors to react really quick on changes in the environment, because they do not event need to see it. They only moves according to their neighbour. Moreover control system discussed in [2] also consists of obstacle avoidance ability, though only for sphere obstacles. It is thoroughly described in the article so in the next paragraphs is only light outline.

The controller computes position of each quad-rotor separately. The next position of the i-th quad-rotor is computed from a force $\mathbf{F}_i$ defined in equation (7). Which is force by which the surroundings acts on the UAV. This force consists of three elements:

1) Effect of other individuals, 2) goal effect and 3) obstacle effect. These three forces are described in the next paragraphs. Variables used in equations in this section are described in table 2.

Symbol	Description
$\mathbf{R}_i$	is the position vector of the i-th quadrocopter
H_i	is the heading direction vector of i-th quadrocopter
$\mathbf{L}_{ij}$	$\mathbf{L}_{ij} = \mathbf{R}_i - \mathbf{R}_j$Is vector from the i-th to the j-th quadrocopter
$\mathbf{L}_{ig}$	is vector from the i-th quadrocopter to the goal
$\mathbf{L}_{oi}$	is vector from obstacle to i-th quadrocopter
Ψ_{io}	is the angle between $\mathbf{L}_{rb_i}$ and H_i

Table 2 Table of symbols used in the flocking equations

2.2.1 Other individuals forces

Since the UAVs are supposed to behave like a swarm, the individuals have to keep the swarm compact but on the other hand must keep enough distance from each other. That is a reason why the force between i-th and j-th quad-rotor is designed as a spring-damper model. This force is described by differential equation:

$$\mathbf{F}_{ind_i j} = K_d(\|\mathbf{L}_{ij}\| - L_r)\mathbf{L}_{ij} + D_d \frac{\mathrm{d}\mathbf{L}_{ij}}{\mathrm{d}t}. \tag{1}$$

Where D_d and K_d are constants of the regulator and L_r is the desired distance. The final force is defined as a sum of forces generated by other individuals. Magnitude of these forces is set by a distance weight function e_{ij}. Formula for the final force representing effect of other individuals in the swarm is than:

$$\mathbf{F}_{ind_i} = \sum_{i,j \neq i}^{N} e_{ij} \mathbf{F}_{ind_i j}. \tag{2}$$

Where the distance weight function serves for emulation of a range of visibility of the UAV, that means that only close neighbours which can be visually localized have effect on the resulting force. This is important for a realistic swarm behaviour. Function e_{ij} is described by formula:

$$e_ij = \frac{1}{e^{a\mathbf{L}_{ij} - b} + c} + \frac{1}{e^{0.5a\mathbf{L}_{ij} - b} + c}. \tag{3}$$

2.2.2 Goal effect

In order to be able set some point which is the swarm supposed to reach, there is a force which attracts it towards this point called goal effect. It is designed as spring-damper model and is defined by differential equation:

$$\mathbf{F}_{goal_i} = K_g \cdot \mathbf{L}_{ig} + D_g \frac{\mathrm{d}\mathbf{L}_{ig}}{\mathrm{d}t}, \tag{4}$$

Where D_g and K_g are constants of the regulator and $\mathbf{L}_{ig}$ is the wanted distance from goal. But this equation would mean that with distance from goal rises also magnitude of the attractive force. Which would mean that when the swarm si far from the goal this force would be too large. So for cases where magnitude of $\mathbf{L}_{ig}$ is too big there is another formula:

$$\mathbf{F}_{goal_i} = W_g \frac{\mathbf{L}_{ig}}{\|\mathbf{L}_{ig}\|}. \tag{5}$$

2.2.3 Obstacle effect

Since in the model has to be also accounted avoiding obstacles there is obstacle effect. It is repulsive force defined by its magnitude and vector from centre of the obstacle to the quad-rotor. Reaction from set of all visible obstacles $\mathcal{O}$ for i-th quadrocopter is:

$$\mathbf{F}_{obs_i} = \sum_{o \in \mathcal{O}} \delta \cdot e_{oi} \cdot \frac{\mathbf{H}_{oi}}{\|\mathbf{H}_{oi}\|}. \tag{6}$$

Where $e_{oi} = b_o e^{a_o} \|\mathbf{L}_{oi}\|$ is the distance function, $\delta = (1 + \cos(\Psi_{io}))$ is the direction dependency function, $\mathbf{H}_{oi} = (\mathbf{H}_i \times \mathbf{F}_{io}) \times \mathbf{H}_i$ is vector perpendicular to the direction vector of the i-th UAV, where $\mathbf{H}_i$ is the direction vector of the UAV.

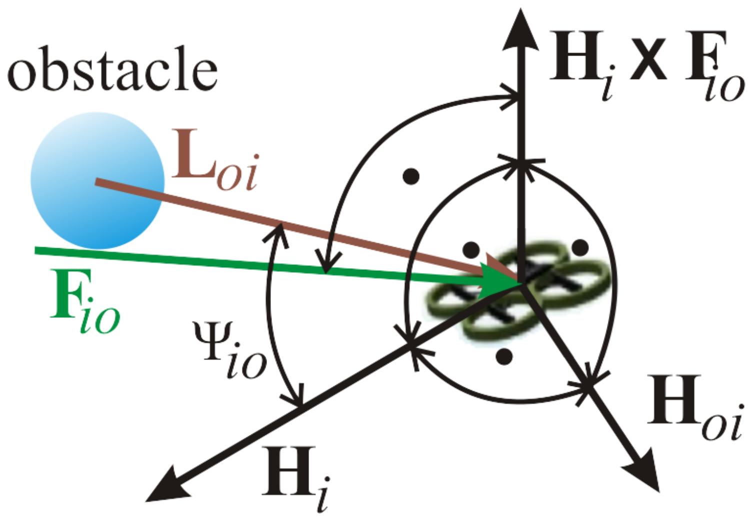


Figure 2 Graphical representation of variables used in computation of the obstacles effect. (source: [2])

2.2.4 Conclusion

Finally, the total force that acts on the i-th UAV is sum of the effects mentioned above:

$$\mathbf{F}_i = W_{ind} \cdot \mathbf{F}_{ind_i} + \mathbf{F}_{goal} + W_{obs} \cdot \mathbf{F}_{obs_i}. \tag{7}$$

Prerequisite of this algorithm is the obstacles being spheres and knowledge of their radius and centre coordinates. So this project is focused on modifying this algorithm for its better utilization, by enabling usage in environment composed of polyhedrons.

3 Distance computation

Since the algorithm for simulation of behaviour of swarm from Jan Vakula [1] works only for sphere obstacles, its applicability is very limited. For more realistic scenarios it is crucial to obtain information on distance of MAV from complex obstacles. For such purpose, a suitable library will be chosen and implemented into the current algorithm.

Most libraries used for operations with three dimensional objects are designed for collision detection. But the mechanisms for computing other thing such as intersection depth are similar. Which goes also for minimal distance computation. As two possible choices able of finding the closest points between two 3D objects appeared to be SWIFT++ and GJK.

3.1 SWIFT++

SWIFT++ [4] is a large package for collision detection, capable of detecting intersection, computing approximate and exact distance and has also other features. It is able to work with a scene composed of general rigid polyhedral models. It is implemented in C++ as the name suggests. It supports different formats of description of the obstacles. It is also possible to use non-convex objects because it has decomposer. Decomposer is a function for preprocessing of objects and one of it tasks is decompose it into convex parts.

3.2 GJK

Apart from SWIFT++, GJK is name for an algorithm named by initials of its authors Gilebert – Johnson – Keerthi. Depending on its implementation it is able of computing intersection depth, detect collisions or compute minimal distance between pair of objects in 2 or 3-dimensional space. If used with Qhull [5] library it can even work with non-convex objects.

The original algorithm was described in 1988 and it went through various developments until now. There had been made progress in computational time and different methods are used since then. This is possible thanks to better hardware and more sophisticated code. Once grasped it is very easy to implement and therefore there are many variations made by different people now. Most of definitions and facts about gjk in this section is taken from [6].

3.2.1 Introduction

It is an iterative method computing euclidean distance between objects in m-dimensional space and depending on particular implementation can find minimum distance, detect intersection between objects or count penetration depth. The algorithm is fast because the computation time is linearly dependent on number of vertices of the given objects and can be even reduced to almost constant time. Main reason behind its quickness is that gjk uses so-called support mapping functions for describing geometrical objects and reducing the whole task of computing distance between two objects into simpler one consisting only of one object and the origin. Finding the point closest to origin goes through iterative constructing of simplexes inside the object and finding closer ones every iteration until the result is found. All of these steps will be explained in following subsections.

3.2.2 Basic Concepts

In following paragraphs are explained basic concepts used.

Convex object

Object is convex if every pair of its vertices can be connected with a straight line which is completely contained within the object.

Simplex

M-simplex [7] is object consisting of m-dimensional polytopes which are the convex hull of m+1 vertices in m-dimensional space. It this case the most used are triangle and tetrahedron. Other examples can be seen in Figure 3.

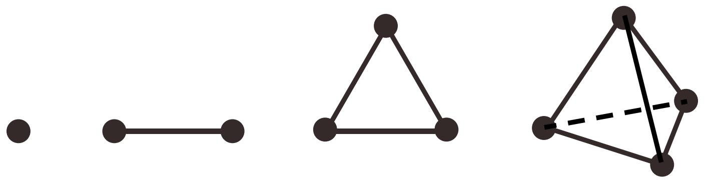


Figure 3 Simplices: from left to right: 0-simplex: point, 1-simplex: line, 2-simplex: triangle, 3-simplex: tetrahedron

Minkowski Difference

One of the GJKs features that make distance computation much easier is its possibility to compute only distance between one object and origin instead of two objects. That is possible using Minkowski difference which is variation of more known concept called Minkowski sum. The euclidean distance between two convex sets A and B is defined:

$$d(A, B) = \min\{\|\mathbf{x} - \mathbf{y}\| : \mathbf{x} \in A, \mathbf{y} \in B\}, \tag{8}$$

and according to [8], definition of Minkowski difference C is as follows:

$$C = A - B = \{\mathbf{x} - \mathbf{y} : \mathbf{x} \in A, \mathbf{y} \in B\}. \tag{9}$$

Hence the desired point $v(C) \in C$ is closest to the origin according the following equation:

$$d(A, B) = \|v(A - B)\| = \|v(C)\| = \min\{\|\mathbf{x}\| : \mathbf{x} \in C\}. \tag{10}$$

Example of Minkowski difference for two non-intersecting polygons can be seen in Figure 4. The Minkowski difference simplifies the problem a lot from two reasons:

1. Searching for shortest distance between polygon and point is much easier than between two polygons.

2. As can be seen in the example Figure 4 some of the points are inside of the polygon, and thus the hull of the new object A-B consists of less vertices than the two original polygons. This means that less points need to be computed.

The same method can be used for detecting intersection of two objects. In such case the origin is inside the resulting shape.

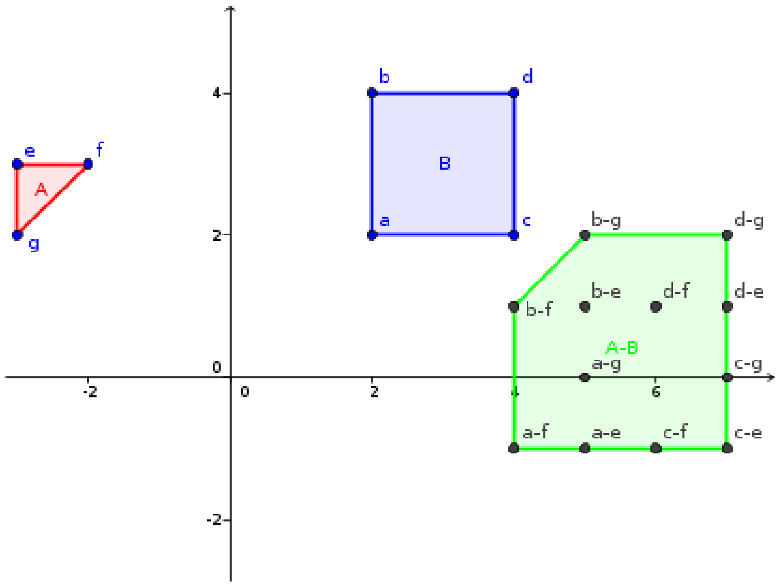


Figure 4 Minkowski difference of two polygons A and B. Source: [9]

Support Mapping Function

Usage of support mapping in GJK is another mechanism employed for reduction of computational time. It is a way of description of geometrical objects. Output of this function for convex set C is a point within this set which is the most extreme in direction of specified vector v. In this case where the purpose is finding the minimum distance to origin the vector v aims toward it. Since gjk is usable with convex objects of arbitrary shapes and number of dimensions there are needed different mappings for each individual classes of objects. Computation of support points for most common geometric primitives can be described with simple algebra.

Polytope is a geometrical object with flat sides in space of any number of dimensions. Basic polytopes are points, line segments, triangles and tetrahedron but the polytope can be also any convex polygon or polyhedra. Support function $h_C(v)$ with support point $s_C(v)$ for polytope C and vector v is defined as:

$$h_C(v) = s_C(v) \cdot v = \max\{v \cdot \mathbf{x} : \mathbf{x} \in C\}. \tag{11}$$

It means that computational time of support point for polytope linearly depends on number of vertices. Moreover, in some cases the task can even be reduced to almost constant time. To achieve such time dependency is needed adjacency graph of all vertices. Where each edge of the polytope corresponds with edge of this graph. Having the adjacency graph, searching for the next support point takes much less time because the next point is most probably near the previous one. This technique of using local search is called hill climbing.

Graphical demonstration of result of support function of polytope C with vector v aiming to origin is in Figure 5. Mapping function for spherical extensions such as sphere, cone or cylinder are defined as well, it makes more convenient treating of some non-convex shapes which are a union of convex polytope and its spherical extension i.e. pillar with hemispheres at its ends. Spherical extensions can also represent shell of safety around some object.

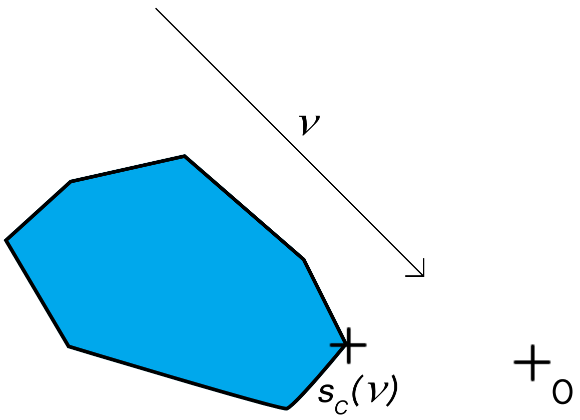


Figure 5 Support function of polytope C with vector v aiming to origin

3.2.3 The Algorithm

The implementation used in this thesis serves for computation of distance a coordinates of a couple of support points for two convex sets $A \subset R^m$ and $B \subset R^m$.

Main Algorithm

As stated above, GJK first computes the Minkowski difference $C = A - B = \{\mathbf{x} - \mathbf{y} : \mathbf{x} \in A, \mathbf{y} \in B\}$ and then searches for only one support point $v(C)$ closest to origin. In order to find the $v(C)$, it constructs sequence of simplexes which converge to $v(C)$. Before run of the algorithm, there is need to define some assumptions:

- Let $V_k \subset C$ be list of vertices of simplex in k-th *iteration* k>=0.
- Let $v_k \subset V_k$ be the point which is closest to the origin.
- Let function $g(x)$ be defined as:

$$g_C(x) = |x|^2 + k_C(-x), \tag{12}$$

  where following statements are true for $x \in C$[1]:
  1. if $g_C(x) > 0$ there is a point z in the line segment formed by co $\{x, s_C(-x)\}$ satisfying $|z| < |x|$. Where notation co { } is defined in [6] and means convex hull of points of the set.
  2. $x = v(C)$ if and only if $g_C(x) = 0$.
  3. $|x - v(C)|^2 \leq g_C(x)$.

There follows the algorithm:

1. Initialization

[1]Proof can be found in [6]

- k=0;
- Set of vertices of the simplex in k-th iteration is $V_k = \emptyset$;
- Set v_k to be an arbitrary point within C ;

2. Determine support point $v_k = s_C(-v_k)$ in direction $-v_k$;
3. If $g_C(v_k) = 0$, set $v(C) = v_k$ and stop;
4. Set $V_{k+1} = \hat{V_k} \cup \{s_C(-v_k)\}$, where $\hat{V_k} \in V_k$ has m elements or less and satisfies $v_k \in \text{co}\,\hat{V_k}$;
5. Increment k and proceed to step 2;

Figure 6 illustrates example of the main algorithm for polytope $C = \text{co}\,\{z_1, \dots, z_5\}$ in R^2

Sub-algorithm

The main algorithm, requires computation of new $v(\text{co }Y)$,$Y = \{y_1, \dots, v\} \in R^m$, where $Y = V_k$ every iteration. Johnson [10] originated a procedure for doing this. This method is described in next paragraphs. It is efficient when v is small and yields a representation:

$$v(\text{co}\,Y) = \sum_{i \in I_s} \lambda^i y_i, \tag{13}$$

$$\sum_{i \in I_s} \lambda^i = 1, \lambda^i > 0, i \in I_s \subset \{1, \dots, v\}, \tag{14}$$

$$Y_s = \{y_i : i \in I_s\} \text{ is affinely independent,} \tag{15}$$

where s indicates a particular member of the family of all non-empty subsets of Y. In step 4) of the main algorithm, stated above, Y_s becomes $\hat{V_k}$ and since it is affinely independent it has minimal number of elements. This simplifies computation in the next iteration. Because v is small, it is effective to check all possible combinations of subsets Y in order to find one fitting to equations (13) to (15). Number of combinations can be obtained from:

$$\sigma = \sum_{k=1}^{v} [\frac{v!}{k!(v-k)!}] \tag{16}$$

Geometrical explanation is that all open subsets of the polytope co Y need to be checked whether they contain $v(\text{co}\,Y)$. The following theorem characterizes the representation (15). Let I'_s be the complement of I_s in I and $Y_s, s = 1, \dots, \sigma$ be an ordering of the subsets of Y. Real numbers $\Delta_i(Y_s), i \in I_s$, and $\Delta(Y_s)$ are defined:

$$\Delta_i(\{y_i\}) = 1, i \in I,$$

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i \in I_s} \Delta_i(Y_s)(y_i \cdot y_k - y_i \cdot y_j), \text{ where}$$

$$k = \min i, i \in I_s, j \in I'_s.$$

$$\Delta_j(Y_s) = \sum_{i \in I_s} \Delta_i(Y_s). \tag{17}$$

Number of operations for all subsets of Y is relatively low, i.e. $v = 3$ requires 6 inner product evaluations and 12 multiplies. There are 3 conditions which must be fulfilled for (15) to hold:

1. $\Delta(Y_s) > 0 \implies Y_s$ is affinely independent
2. $\Delta_i(Y_s) > 0$ for each $i \in I_s \implies v(\text{co } Y_s)$ is in relative interior of co Y_s
3. $\Delta_j(Y_s \cup \{y_j\}) \leq 0 \implies v(\text{co } Y_s) = \text{co } Y$

And for coefficients λ^i from applies[2]:

$$\lambda^i = \Delta_i(Y_s)/\Delta(Y_s) \tag{18}$$

So for a finite set $Y = \{y_1, \ldots, y_v\} \subset R^m$, and ordering $Y_s, s = 1, \ldots, \sigma$, of all subsets of Y the algorithm processes like this:

1. $s = 1$;
2. if $\Delta(Y_s) > 0$ and $\Delta_j(Y_s) > 0, j \in I_s$, and $\Delta_j(Y_s \cup \{y_j\} \leq 0), j \in I'_s$, define $v(\text{co } Y)$ by equations (13) and (18) and stop;
3. if $s < \sigma$, increment s and go to step 1
4. stop and indicate failure

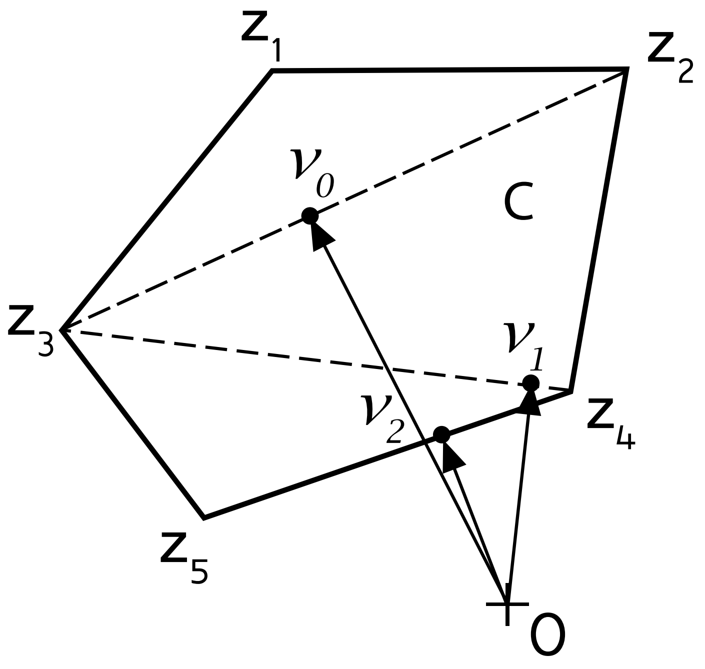


Figure 6 Graphical example of run of the Main algorithm of GJK

This implementation of sub-algorithm comes from [10] written in 1987 but with modern hardware and different authors come other methods for the sub-algorithm. The enhanced version of gjk used in this thesis made by Dr. Stephen Cameron [11] uses so-called hill climbing when computing support points. It means that results from previous iteration are used for quick finding of new support point. This method is effective while computing large convex hulls because it searches first points adjacent to the previous support point.

3.2.4 Implementation

In order to use the GJK in the current algorithm one of them had to be rewritten to different programming language because implementation of gjk by Stephen Cameron

[2]Proof of this theorem can be found in Appendix II of [6].

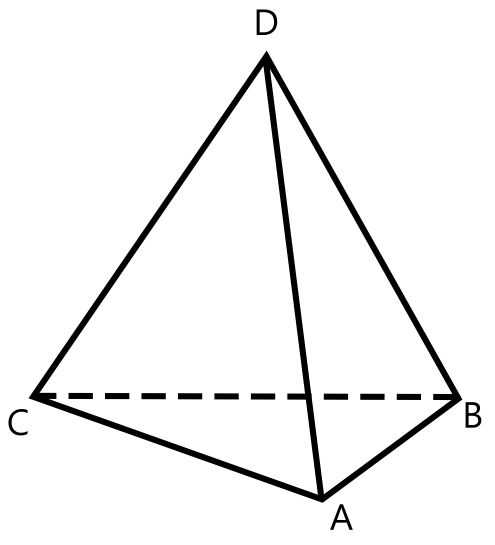


Figure 7 Tetrahedron

is in C++ and the swarm behaviour was implemented in MATLAB. It was decided to use C++ mostly because if the code should be used in real device some day, it would probably be C or C++.

Obstacle definition

GJK library uses unusual concept for defining obstacles. Because of its support function and need to maintain graph of adjacent vertices, the object structure is defined by three attributes:

1. Number of vertices - integer specifying number of vertices
2. Vertices - two-dimensional array, where rows are coordinates of individual vertices
3. Rings - one-dimensional array specifying which vertices are connected by edge

The third parameter may be little harder to understand. For obstacle of n vertices, it is an array of numbers, where the first n numbers, define indices at which starts list of adjacent vertices for each vertex in this array. Order of the vertices corresponds with the *vertices* array. Each list of the connected vertices ends with negative number. So for tetrahedron in figure 7 the variables are: number of vertices= 4, vertices= $\{A, B, C, D\}$ and rings= $\{4, 8, 12, 16, \quad i_B, i_C, i_D, -1, \quad i_A, i_C, i_D, -1, \quad i_A, i_B, i_D, -1, \quad i_A, i_B, i_C, -1\}$.

Where i_k is index of k-th vertex in *vertices* array. In this way it is possible to define any convex object. In case of the object is point the rings=0.

Representation of complex environment

Input of GJK must be only convex objects. The MAVs are considered to be points, so there is no problem. In case of obstacles, it is more complicated. But it is usually no problem to divide a non-convex object into few convex ones and less vertices means much smaller rings variable. Probably the easiest and fastest way of creating the environment is to represent every obstacle by one or more arbitrarily distorted blocks. It also makes specifying *rings* less difficult. Using the same *rings* "template" while changing only coordinates of individual vertices is really fast and convenient. But it is still possible to use convex objects of any complexity.

4 Moving target

After making the swarm able to move between convex obstacles, there is need to define its target. Using only one static goal point is not usable with this kind of controller because it would lead the swarm right through the obstacles and it would have difficulties bypassing huge ones, especially walls perpendicular to its direction. Such situations is sketched in figure 8. So the target must guide the swarm along some defined route in between the obstacles. Target moving along a path is good in many ways. It leads the swarm around obstacles while keeping specified distance from them. This means that the drones should not encounter situations where they can not get through the environment. Thanks to this would be also possible assign some other vehicle as the goal so it would serve as a leader of the swarm.

To be able to use moving goal, its route must be firstly found. Path can be manually defined by points through which the swarm must go, and in some cases it may be even the best way, but it is not efficient in more complicated environment. More usable is to use method for path planning which takes start and end point or some checkpoints and finds short and safe path.

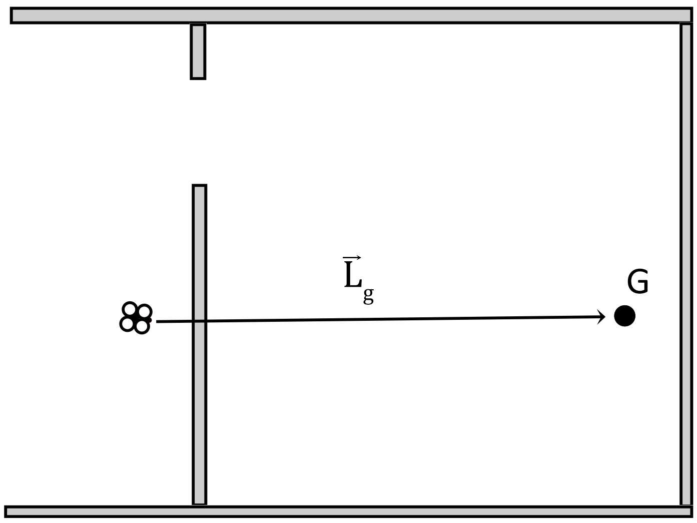


Figure 8 Example of case where static goal is not usable, because the MAV (Micro aerial vehicle) is pulled straight into a wall.

4.1 Graph

In order to plan path, the environment has to be represented as a graph. There are various ways to do that and each method has its pros and cons. The first solution that anyone could come up with is to just divide the space by a 3D grid but that is not the best solution for this application. It would be needed to remove edges which go through obstacles. And even though it would probably work good for finding the shortest path, it would not be suitable path for following by the swarm. Truth is the shortest route is actually not the best one in most cases, because it would lead too close to obstacles sometimes. Possibly even through places where not even a single quad-rotor fits.

Since it is highly desirable for the target to maintain a distance from the obstacles there is graph which is commonly used in robotics and fulfils all requirements for this algorithm.

4.1.1 Voronoi Diagram

This text is based on survey [12]. Voronoi diagram is a one of the most fundamental structures in the computational geometry which is used in many scientific branches. It takes a set of points in space or plane and divides it accordingly to the *nearest-neighbour-rule*. That means every point is assigned with a region closest to it. In two dimensional space voronoi diagram for few points can look like in figure 9.

This definition is easy to understand but for completeness the usual generic definition given in [12] is mentioned there. Let S be a set of n points (called sites). For two distinct sites $p, q \in S$, the *dominance* of p over q is defined as the subset of the plane being at least as close to p as to q.

$$\mathrm{dom}(p, q) = \{s \in R^m | d(x, p) \leq d(x, q)\} \tag{19}$$

As can be seen $\mathrm{dom}(p, q)$ is a closed cell bounded by bisector or plane perpendicular to segment $|pq|$. This so-called separator divides the space into points closer to p and those closer to q. The *region* of site $p \in S$ is the portion of the space lying in all of the dominances of p over the remaining sites in S. Mathematical definition is:

$$\mathrm{reg}(p) = \bigcap_{q \in S - \{p\}} \mathrm{dom}(p, q). \tag{20}$$

As can be seen, the Voronoi diagram is designed to work with sets of points. But environment in this application consists of convex polyhedrons. The algorithm made by Martin Bláha solves it by representing the each face of the polyhedra with dense grid of points. In result, it works as if it was continuous surface, because the algorithm afterwards removes most of edges which go through the obstacles and the planning algorithm does not use them.

4.2 Planning algorithm

Having the graph of the map, all that remains is to choose and implement some suitable algorithm. There are many algorithms and their variations able to find the shortest trajectory but they are can differ a lot. Even though they would eventually find the same result some are much faster because they expand less nodes or require less mathematical operations. The algorithms can also differ in used metrics but than even the same algorithm would return different result while using i.e. Euclidean and Manhattan distance.

Figure 10 serves for illustration of how much depends on choice of the algorithm. BFS - breadth-first search algorithm is used only for comparison so there is not an explanation but it can be found in [13]. Let just say it uses less sophisticated way of choosing nodes to be expanded. As can be seen both algorithms found same result (white line) but number of expanded nodes (not white or black) differs a lot.

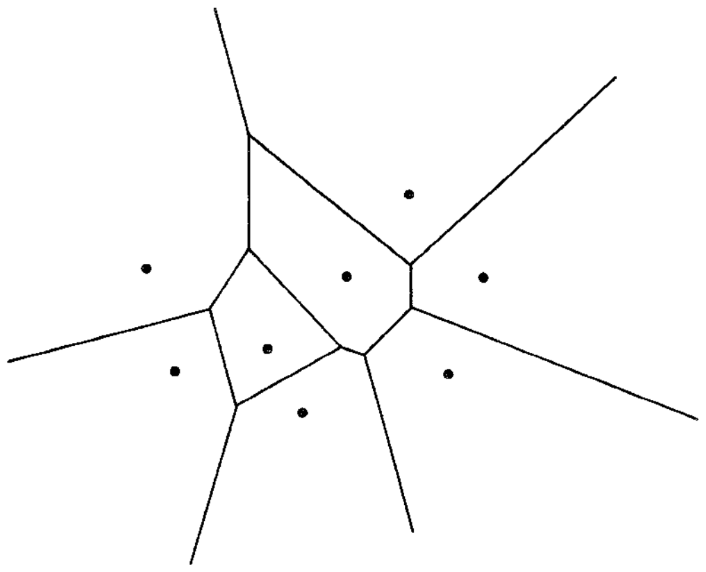


Figure 9 Voronoi diagram in 2-D [12]

Algorithm made by Martin Bláha used in following simulations is A* which is not hard to implement and efficient.

4.2.1 A*

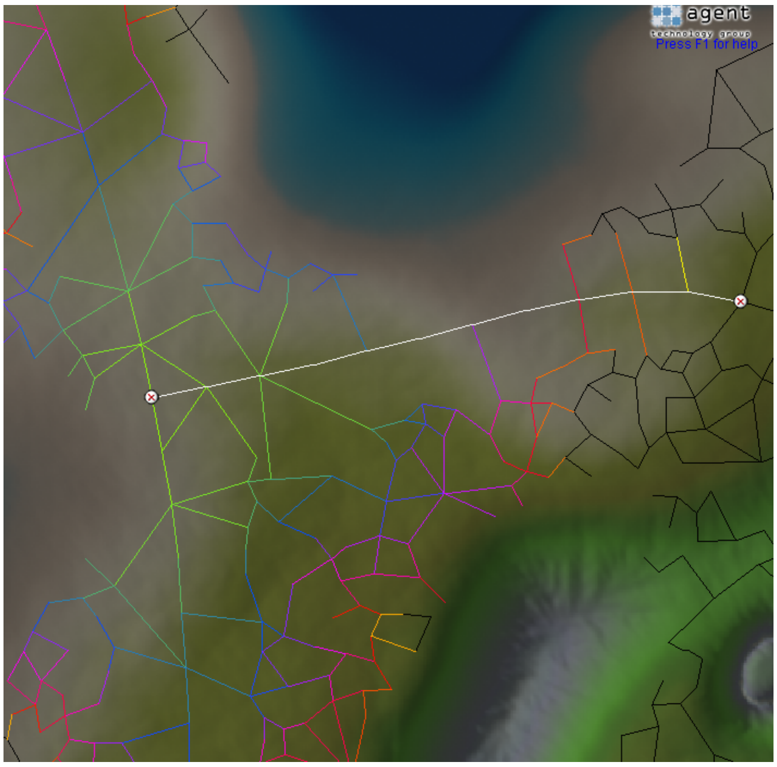
The A* is an algorithm using heuristic function to expand as least nodes as possible. This function serves for deciding which node should be expanded next by estimating cost of the path if it would go through this node. Little altered definition of the one in [13] is given in next paragraphs. All symbols used there are defined in table 3.

Computing the optimal distance from start to the current node $C^*(x)$ can be done somewhere during the run of the program, when there is no possibility of finding lesser cost, but there is no way to compute $G^*(x)$ and so its value must be reasonably underestimated by some heuristics. That means at least the euclidean distance, but the aim is to compute cost $\hat{G}^*(x)$ as close as possible to $G^*(x)$.

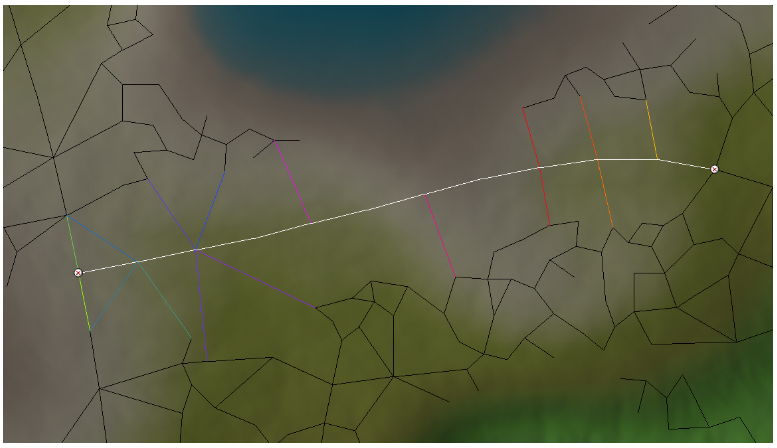
At first $Q = x_I$ and $C^*(x_I) = 0$. The next steps then are always according to same pattern:

1. From Q is chosen site x with the highest priority and removed from this set.
2. For all neighbouring sites of x are computed $C(x'_i) = C^*(x) + l(e_i)$ and $\hat{G}^*(x'_i)$ where x'_i are the neighbouring sites and e_i cost of edge from x to x'_i .
3. If these sites are not in Q yet they are put in according sum of their cost-to-go and cost-to-come. If they already are in the list and have higher value than the new one, then they are replaced.
4. If the goal has not been found yet the process goes all over again.

No matter what kind of heuristic function is used the A* algorithm always (if possible) finds the shortest route to goal. But choosing concurrently fast to evaluate and also efficient heuristic function might matter a lot if computational time is important.



a) BFS - Breadth-first search algorithm



b) A*

Figure 10 Illustration of how much can differ number of expanded nodes between two path planning algorithms. White line is the found route and coloured edges surround expanded nodes

Symbol	Description
X	is a non-empty state space
$x_i \in X$	is one state(node) where x_1 is the start state and x' is the next generated state
$X_G \subset X$	is a goal set.
$C(x)$	denotes the cost-to-come from start which is distance from x_I to x and $C^*(x)$ is *optimal* value thus the shortest possible
$G(x)$	denotes the cost-to-go from x to some set in X_G and $G*(x)$ is optimal cot-to-go
Q	is sorted priority queue of unexpanded states
$l(e)$	is cost of edge

Table 3 Table of symbols used in description of A* algorithm

4.3 The target

Having the path for the goal planned there is need to define movement of the target along this path. Firstly because output of the algorithm is polyline defined by its vertices which is not enough points, it has to be sampled so the goal can move in smaller steps. Since speed of the swarm is not constant the target must move according to position of the swarm so it does not move too much ahead. Because of this, there must be decided important parameter and that is distance which should the goal keep from the swarm. Let centre of the swarm be average of positions of all MAVs. Than l can be denoted as a minimum distance from centre of the swarm to the goal. There are two options now:

1. Make size of l dynamic, meaning that it would be radius of the swarm + some constant. This would mean that the MAVs can never reach the goal unless it stops. Positive of this would be that the MAVs do not have to slow down but on the other hand, the more spread the swarm would be, the further would be also the goal. This could eventually cause that the MAVs find themselves in the situation outlined in Figure 12.
2. Make l constant. Than the MAVs in front which get to the goal slows down which lets the others to catch up and the swarm is not so widespread.

Both of these methods were tested and were found successful. Only in some cases appeared the negative effect of dynamic l mentioned above. Yet still, togetherness of the swarm is important and so was used constant l. Last attribute of the moving target is that it can move only forward along its route.

Setting of l depends on the specific application. If high consistence of the swarm is desired then l should be smaller. This is because when the goal is near the swarm, than the force that attracts the MAVs towards the goal, pulls them more together. But since the MAVs are in dense group they are not so flexible and slower than when having more freedom. On the contrary when l is bigger the swarm can spread wider and more easily avoid the obstacles and thus move faster. Of course, strict following of the goal can be the only key to better performance in some specific environment, but speaking of some common office space, there should not appear so complicated cases. Figure 11 illustrates effect which can have choice of l.

$l = 2$ is probably the lowest possible value because it almost touches the swarm in its standard shape. It can be seen that the distance from the centre is similar for all

MAVs except of few exceptions. But with this size of l the swarm never reaches the goal in measured time apart from the other two cases. The last graph, where $l = 5$, shows greater dispersion of the swarm but reaches the goal at least twice faster. $l = 5$ is quite high because the swarm did not follow the target along the prescribed trajectory much in this case. $l = 3.5$ seems like a good compromise in this case and also other simulations proved that it is suitable value.

However l remains an adjustable parameter and is to be set according to the particular case. Even though its value must be within some bounds. If it was too low the swarm would move slowly and the forces between individual MAVs could collide against each. On contrary, if it was the other extreme and goal was two far, it would lost its purpose of moving target, because if the MAV is in front of an obstacle and the goal is already behind it the goal effect $\vec{F}_G$ would pull it straight into the obstacle. Figure 12 is example of such situation.

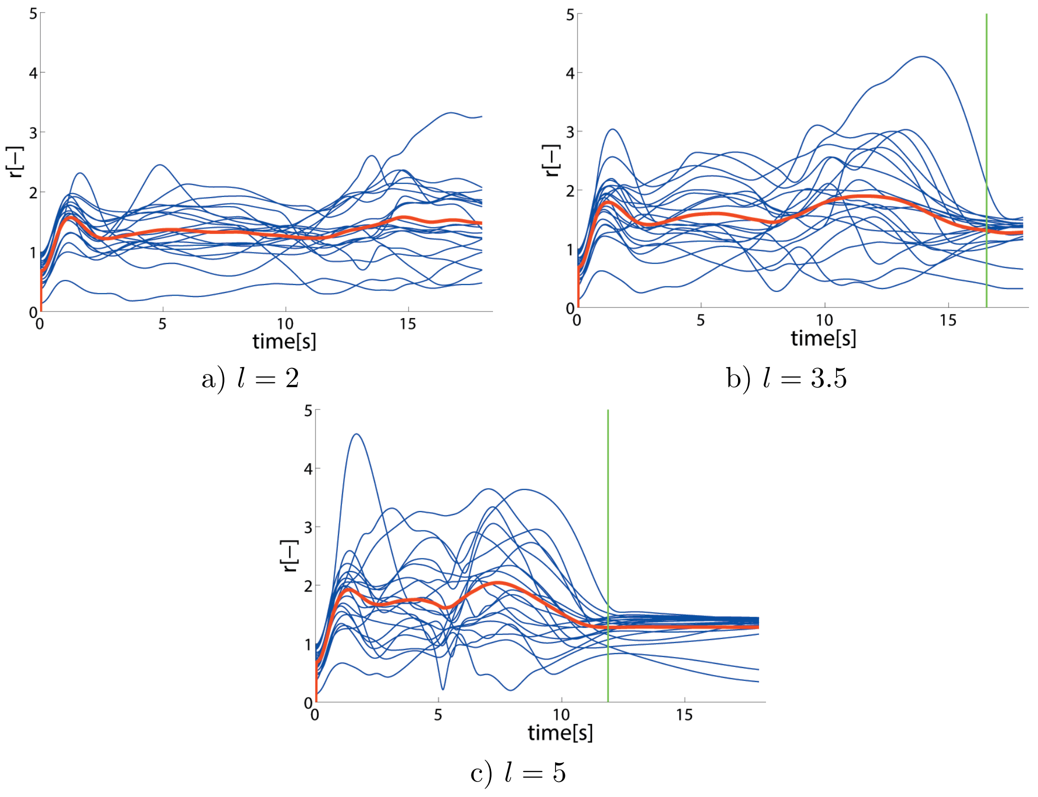


Figure 11 Illustration of effect of the distance l between centre of the swarm and goal. Axis r shows distance from the centre of the swarm. Blue colour represents individual MAVs and red is average value. Green line symbolizes reaching of the end of the path.

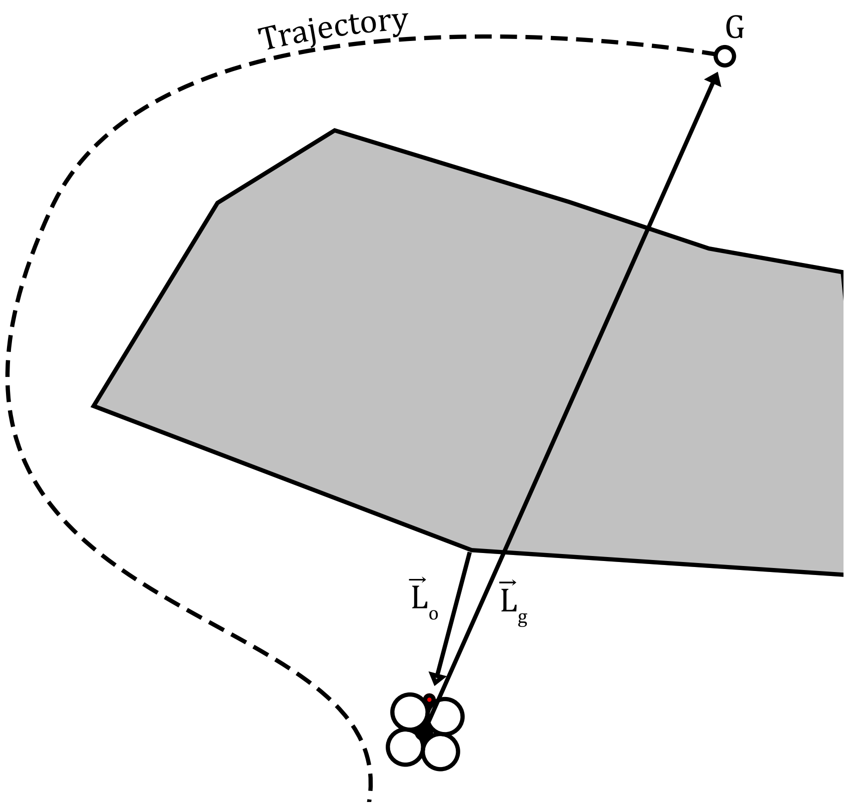


Figure 12 Illustration of case, where right setting of the distance between goal and swarm l is crucial for its functionality. $\vec{\mathbf{L}}_g$ is vector from the MAV to goal and $\tilde{\mathbf{L}}_o$ is the vector from the obstacle to the MAV.

5 Simulations

The swarm controller was rewritten into C++ and modified for use with the GJK. Algorithm for generating Voronoi diagram and path planning using A* was prepared for usage and other scripts were implemented for more convenient usage. With all the needed algorithms ready there is need to practically test its function.

Assignment of this thesis is to confirm functionality in office environment. For this purpose was created model of two connected rooms containing common office equipment represented by geometric primitives. In ....can be seen the test room.

Without any adjustments has the algorithm worked well in most cases, but in some situations especially when the swarm was flying in narrow spaces occurred collisions. Most of it solved setting maximum value for obstacle effect because when the quad-rotor got too close to obstacle even though not colliding the $\vec{F_o}$ send it completely away across whole map through obstacles. Rest of the usual collisions solved little adjustment of few constants, especially increasing the required distance between individual MAVs.

5.1 Configuration of the simulation

Setting up the simulation requires few steps going in logical order:

5.1.1 Defining obstacles

The environment can be defined in Matlab file "obstacles.m". There was made set of functions to ease creating of the environment because defining each vertex is unnecessary labour. For a better notion of the environment while designing it, this function also generates preview in form of Matlab figure. The *colour* parameter serves only for better distinction of the obstacles in the plot in Matlab. It also automatically converts the obstacles into format used by GJK and Tunnel. There are four functions for creating obstacles:

- "plot_block($[x_1, x_2, y_1, y_2, z_1, z_2]$,*colour*)". This function creates block with proportions $(x_2 - x_1) \times (y_2 - y_1) \times (z_2 - z_1)$.
- "plot_hexahedron($[x_1, \dots, x_8]$,$[y_1, \dots, y_8]$,$[y_1, \dots, y_8]$,$[z_1, \dots, z_8]$,*colour*)". This is more adjustable function for creating any convex hexahedron defined by its vertices. Order of the vertices describes figure 13.
- "plot_table($[x, y, z], x_{width}, y_{width}, colour$)". Since table appears quite often in office environment there is function for creating it. $[x, y, z]$ are coordinates of vertex which would have number 6 if the table top was the block in figure 13.
- "plot_chair([x,y,z],orientation,colour)". Same rules as for table applies for function for chair, except in this case it is not a table top but the seat. Parameter *orientation* has four possible values $\{x+, x-, y+, y-\}$ and specifies which direction is the chair facing.

There is of course still the possibility to define any custom shapes as long as they are convex. In such cases one more step is needed and that is generating (if not done manually) of the *rings* parameter for gjk. For this purpose was implemented method "path_to_gjk.py", which given set of vertices and faces computes the "rings".

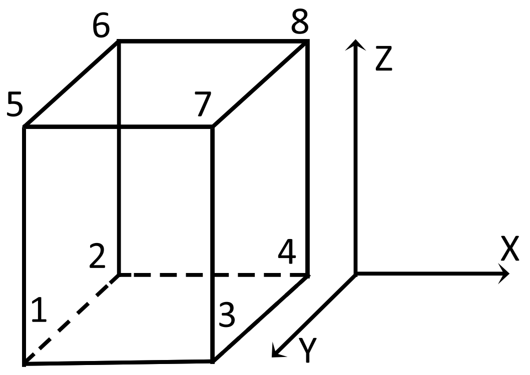


Figure 13 Illustration of order of vertices for function "plot_hexahedron.m"

5.1.2 Planning trajectory

Generating Voronoi diagram and planning of the path is done in one step by script "Tunel". Three things are there to set: *start, end* and *offset.* Parameter *offset* describes how large space around the obstacles should be included in the Voronoi diagram. Useful feature of this algorithm is that given arbitrary points *start* and *end* it finds the nearest vertices of the Voronoi diagram. It means that the user does not have to think about whether the points are in the graph.

In this step may occur problems with generating of the Voronoi diagram. This error probably happens during transformation of the cells when using obstacles which have one dimension many times bigger or smaller than the other two. It can by solved by splitting the scene into individual rooms and omitting the walls, floor and ceiling, because these are the critical shapes. It can be done because the diagram is created in defined bounds, so it only requires setting small *offset.* This deficit needs to be solved in the feature and will be discussed with the author or another solution will be found.

When the path is planned, the function "create_path.m" must be run, because it samples the path and saves it into file for the GJK.

5.1.3 Run the Simulation

At last the main step. Running the simulation is done through "UAV_swarm". In order to speed up the algorithm, there were added a method which decides which obstacles need to be computed. The function must be simple if it should take less time than computing of all obstacles. Let r_s be called radius of the swarm which is distance between centre of the swarm and the furthest individual. And the same parameter is declared for obstacles, where r_o is a radius between average of its vertices and the furthest vertex. The centre of the obstacle and r_o are computed during initialization of the scene and there is no need to compute it again because the obstacles does not change either shape or location. Than each loop of the algorithm, new set of obstacles that are passed to GJK is computed. The obstacle is add to the set if distance between centre of the swarm and centre of the obstacles is less than $r_o + r_s + s$, where s is predefined constant. Depending on setting of s, number of obstacles in radius around the swarm will be computed by GJK. For successful avoiding of obstacles the s should be at least 2.

Before the run itself there can be defined these four options:

1. Number of MAVs in the swarm (1 - 25) - "swarm_size".
2. Number of steps where one step is equal to 0.015 second - "pocet_kroku".
3. Distance l between the centre of the swarm and the target - "target_distance" (differs with size of the swarm but should be at least 2 and 3.5 is proven to be suitable value).
4. Distance s, deciding how far obstacles will be computed by the gjk - "computed_distance" (minimal value is 2).

When the simulation is done, the data are save into file "output.m". The result can be drawn in Matlab by "drawSimulation.m". There is also possibility to set up graphs displaying data of interest.

5.2 Results

Fusing of the GJK with the former algorithm was successful and after some corrections there no longer appear collisions if the goal path leads through enough space. Example of successful avoidance of obstacle can be seen in figure 14. It can be seen that the swarm is behaving as it is supposed and after passing the obstacle it merges back together. Simulation in more complicated office-like environment is shown in figure 15.

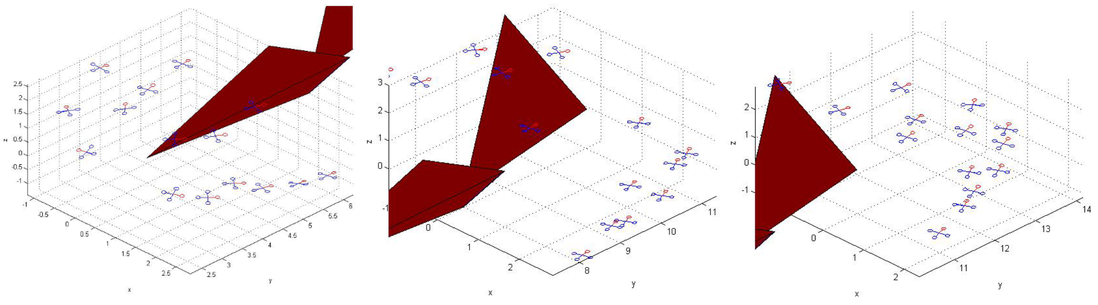


Figure 14 Example of successful avoidance of two polyhedra obstacles

5.2.1 Corridors

There was made a request for research of effect of width of corridor on shape of the swarm. This particular topic is interesting because knowledge of behaviour of the swarm in corridors can help when planning route for the swarm. Before that the only mean of evaluating some path was by its length. With data from following experiments, there could be taken in account also i.e. minimum width of the corridor. Another thing to be explored is effect of bend or change of width of the corridor.

The shape of the swarm is described by an ellipse fitted into set of points, where each point is defined by X and Y coordinates of one of the MAVs. The ellipse is estimated by function [14] using the Least-Squares criterion. The estimation is done for the conic representation of an ellipse (with a possible tilt). But only the proportions are of interest there. Since the shape of an ellipse can be described by its two semi-axis

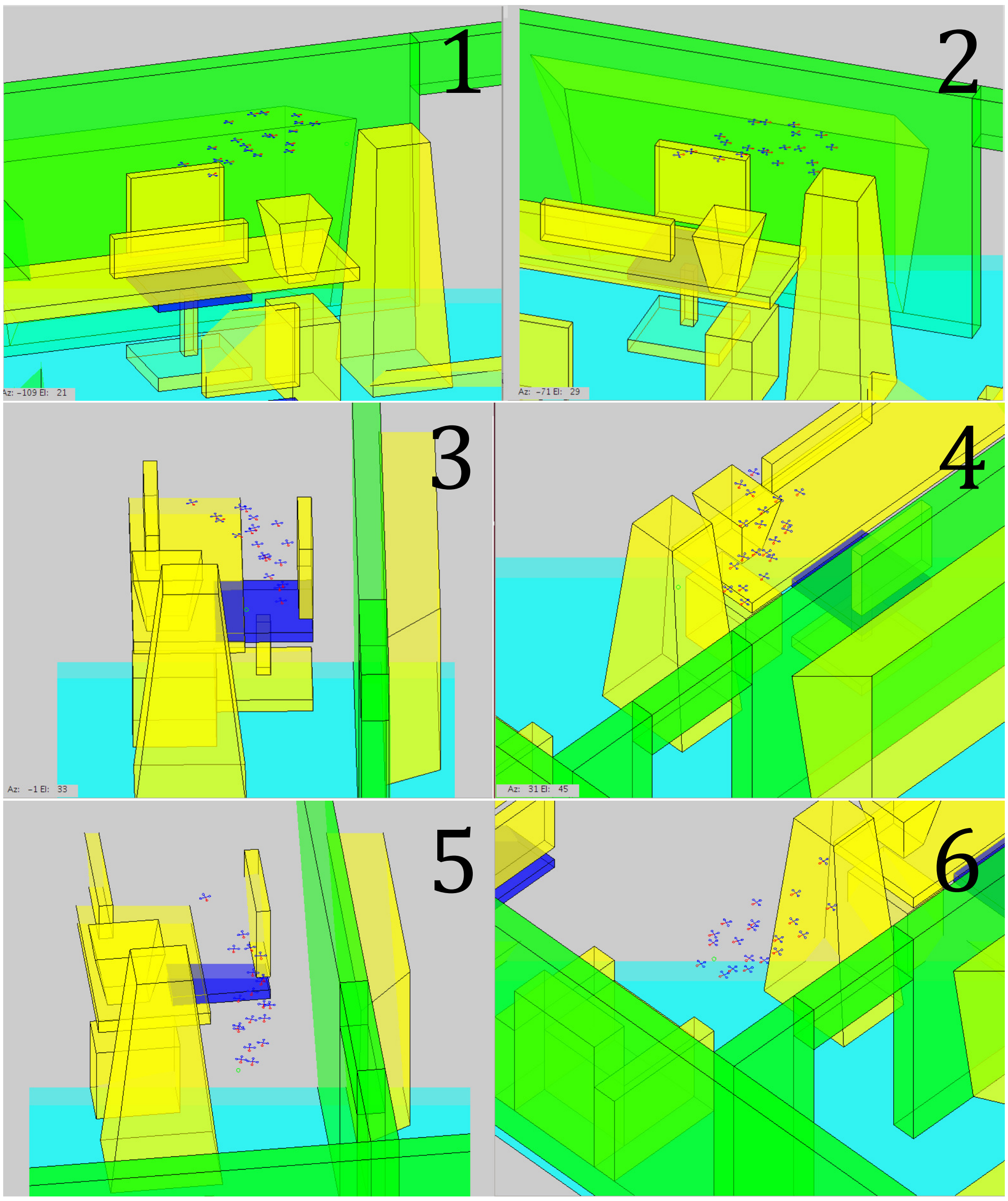


Figure 15 Example of run of the algorithm in office-like environment.

a and b (see scheme in figure 16), these are the parameters which were compared during the simulations.

The more points to fit in, the more accurate is the ellipse. Because of that, there were only done test with swarms larger or equal to 15 MAVs. Even for swarm of 15 quad-rotors is the grap waving instead of being almost straight line. It was often not possible to construct the ellipse with smaller swarms and so these data would not have much value. The ellipse is also not possible to construct when the MAVs are aligned into vertical plane. It is caused the by computing only X and Y coordinates, and so the algorithm sees the swarm as a line. When the corridor is too narrow and so that only one row of MAVs fit into it, the ellipse can no longer be constructed. The computed data are displayed in graphs in the subsections below this text.

For each size of the swarm were made graph showing dependency of the semi-axes of the ellipse, which represents shape of the swarm, on time for different widths of the corridor. It turned out that behaviour of the swarm is very similar for different sizes of the swarm. The simulations showed that minimum width of the corridor where can be constructed the ellipse is 5.5. and the maximum value is 7.5. From 7.5 and more is shape of the swarm always the same. It is visible from the graphs, that even values of a and b for $d = 7$ and $d = 7.5$ are close. From the gathered data is also visible that the swarm even in empty space moves in shape of an ellipse, because values for $d = 7.5$ are the same as if no obstacles were present.

During the simulations appeared an interesting phenomenon. All the data looks according expectations, except for values for $d = 6$. With this specific scenario the swarm behave different and the ellipse was then computed according it. But when the swarm was moved a little and went through the same corridor, it worked well. Because of this anomaly were the values for $d = 6$ not taken in account in the graphs for dependency of size of the semi-axis on d.

As can be seen in figures 29 and 30 the results for different number of MAVS does not vary much. Important fact to consider there, is that precision of the fitted ellipse depends on number of MAVs and even 25 is not much for a precise fit. This can be visible especially in graphs for 15 MAVs, because the characteristics are the most wavy.

From the current data it seems that for these sizes of swarm, does not the fitted ellipse differ much for different numbers of MAVs. The more numerous swarm only spreads wider vertically when flying through narrower corridor, but the width is almost the same.

There has been also done test with corridors, which were bent in some place. But nothing unusual appeared there. The swarm took the turn without problem and its shape changed only if the next part was wider or narrower.

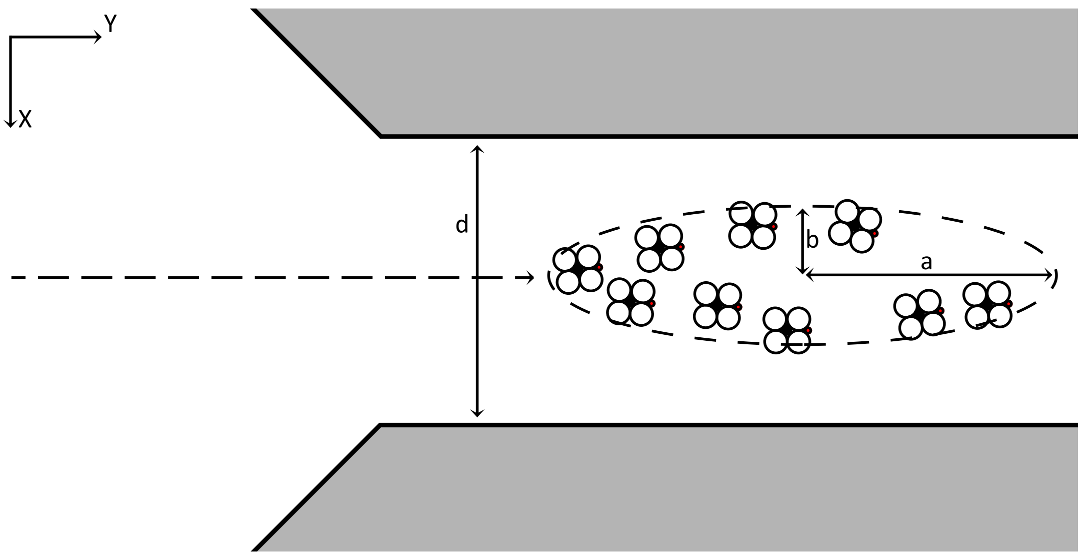


Figure 16 Scheme of measurement of dependence of semi-axis a and b on d. The dashed line symbolizes direction of the swarm.

15 MAVs

The results for swarm of 15 MAVs are:

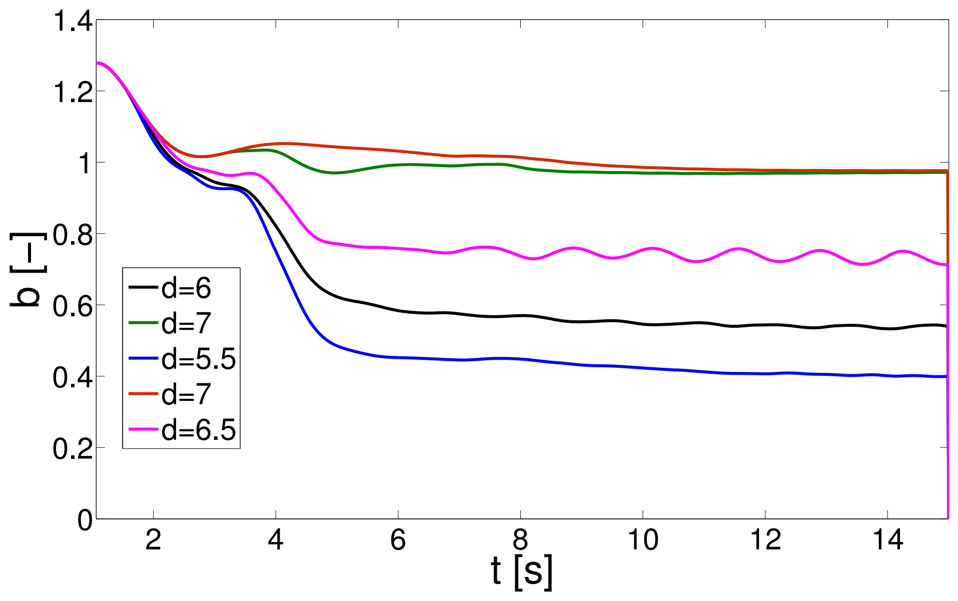


Figure 17 Graph of size of the semi-minor axis b for different widths of the corridor, for swarm of 15 MAVs

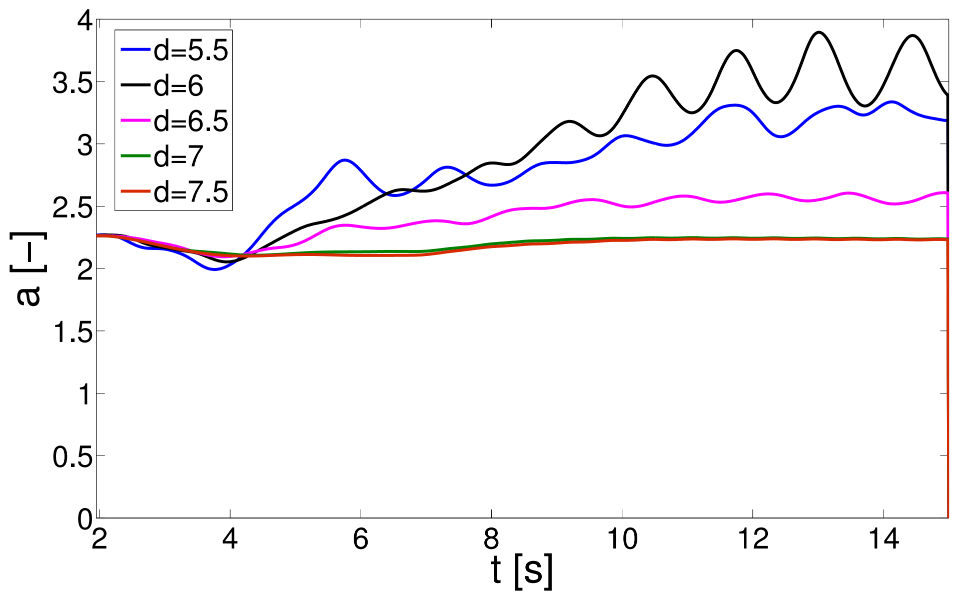


Figure 18 Graph of size of the semi-major axis a for different widths of the corridor, for swarm of 15 MAVs

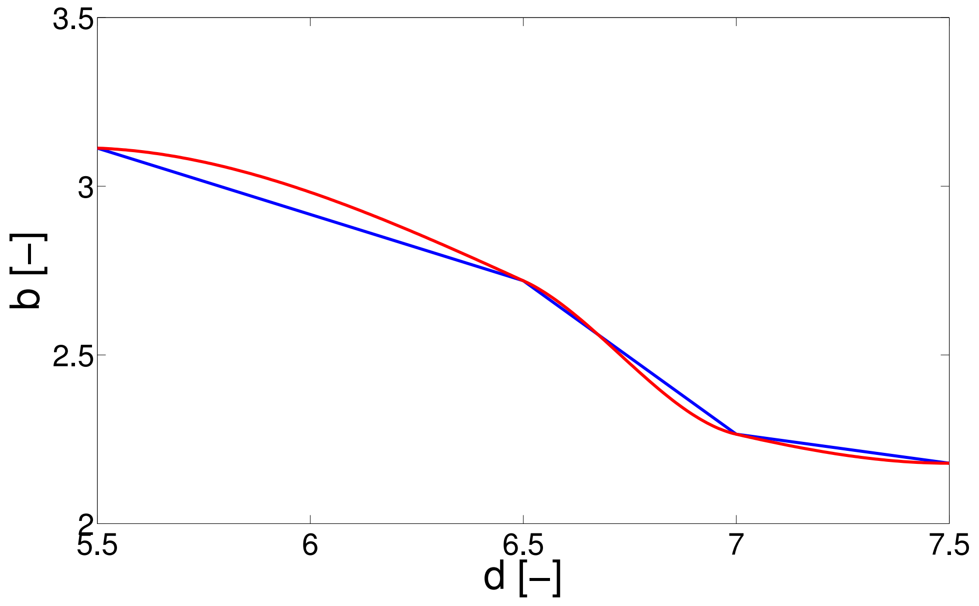


Figure 19 Graph of dependency of size of the semi-minor axis b on width of the corridor d for swarm of 15 MAVs. The red line is a polyline fitted on the measured data.

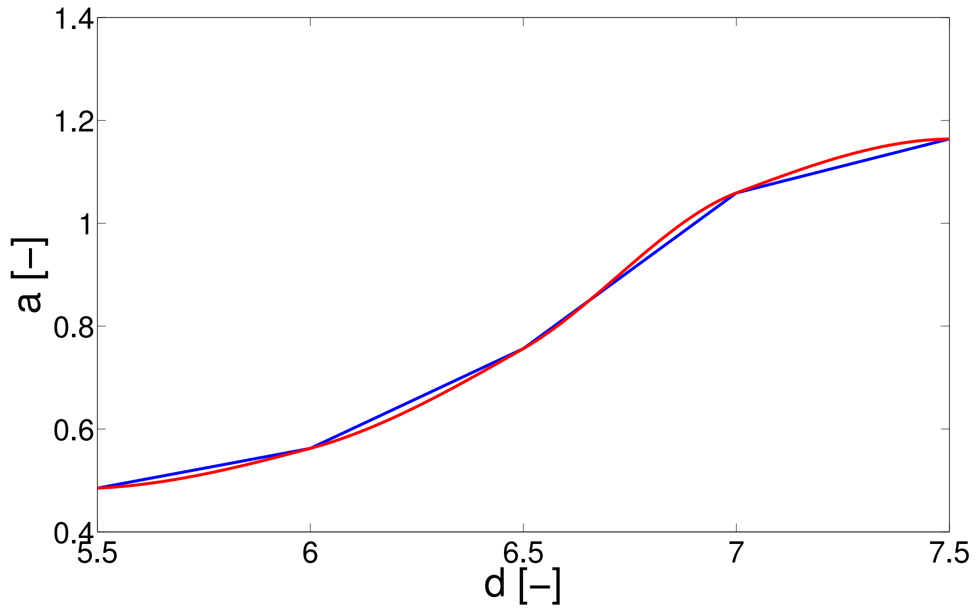


Figure 20 Graph of dependency of size of the semi-major axis a on width of the corridor d for swarm of 15 MAVs. The red line is a polyline fitted on the measured data.

20 MAVs

The results for swarm of 20 MAVs are:

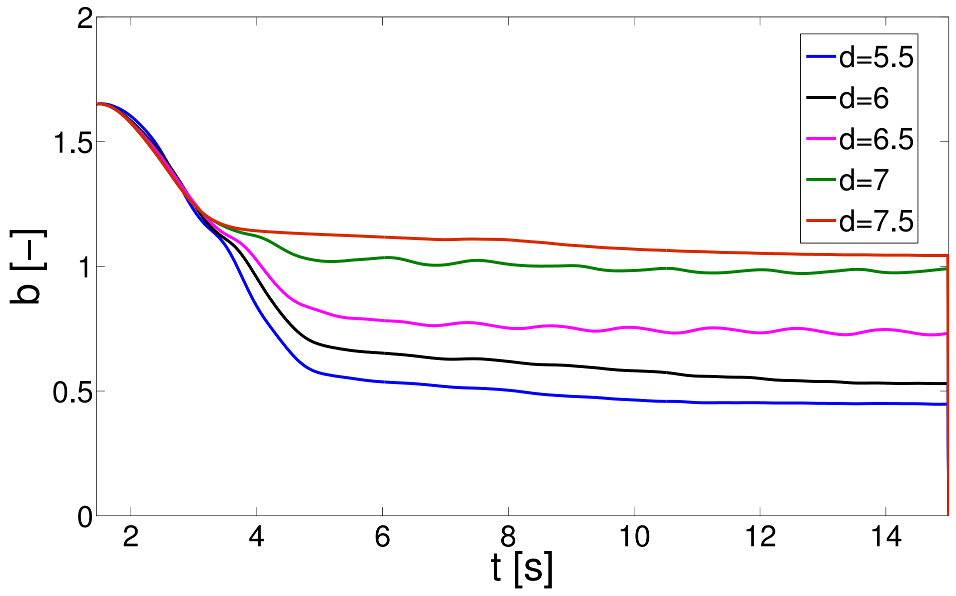


Figure 21 Graph of size of the semi-minor axis b for different widths of the corridor, for swarm of 20 MAVs

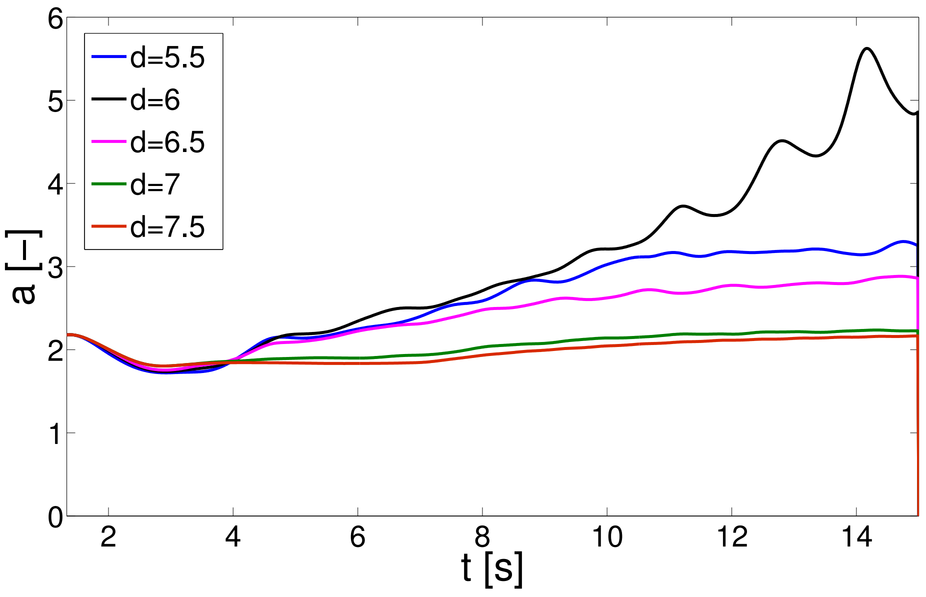


Figure 22 Graph of size of the semi-major axis a for different widths of the corridor, for swarm of 20 MAVs

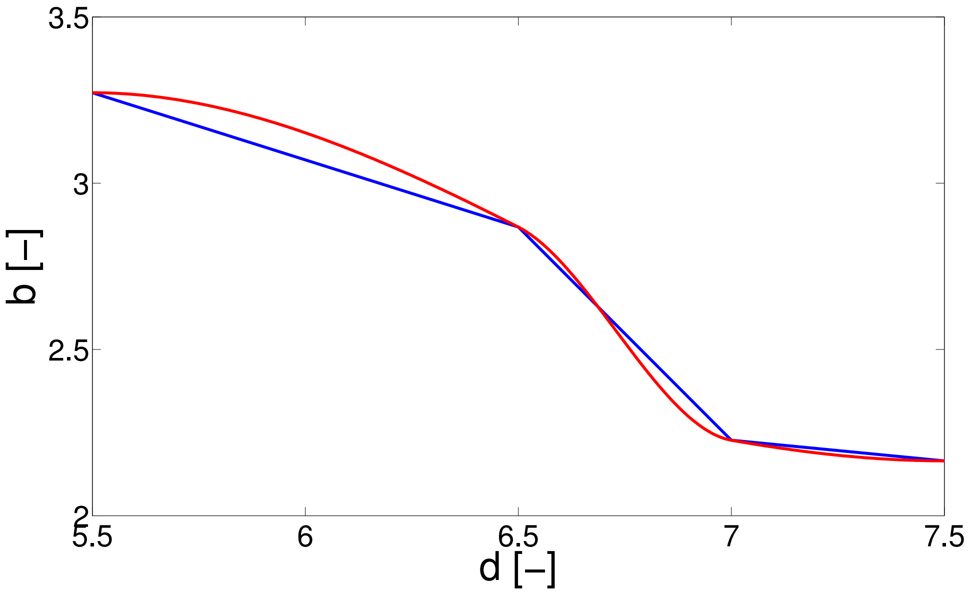


Figure 23 Graph of dependency of size of the semi-minor axis b on width of the corridor d for swarm of 20 MAVs. The red line is a polyline fitted on the measured data.

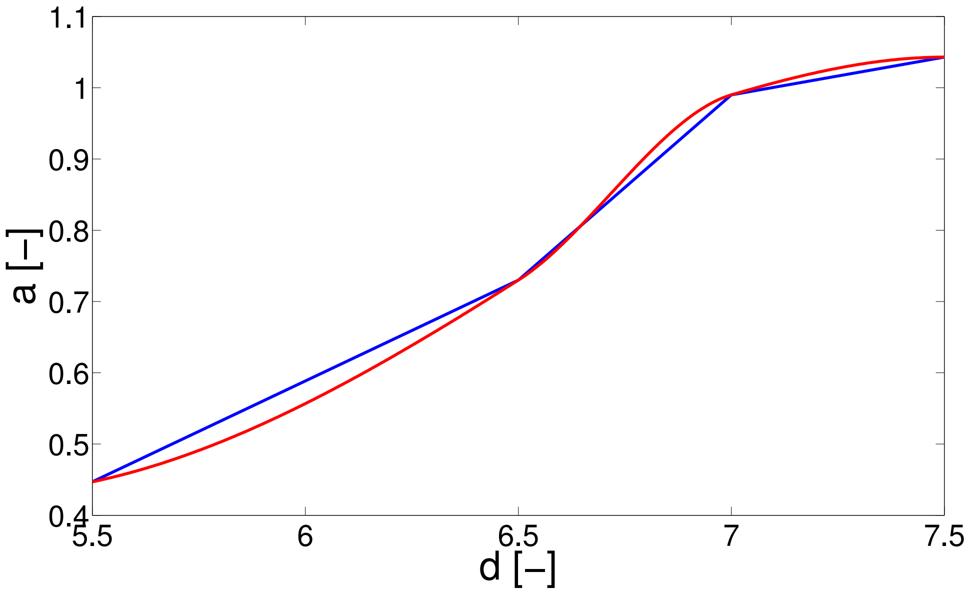


Figure 24 Graph of dependency of size of the semi-major axis a on width of the corridor d for swarm of 20 MAVs. The red line is a polyline fitted on the measured data.

25 MAVs

The results for swarm of 25 MAVs are:

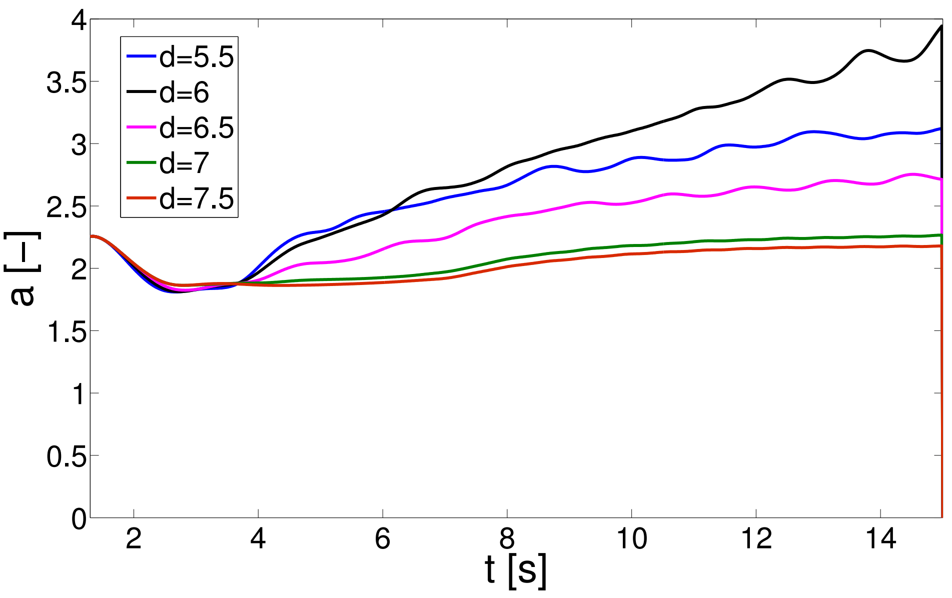


Figure 25 Graph of size of semi-minor axis b for different widths of the corridor, for swarm of 25 MAVs

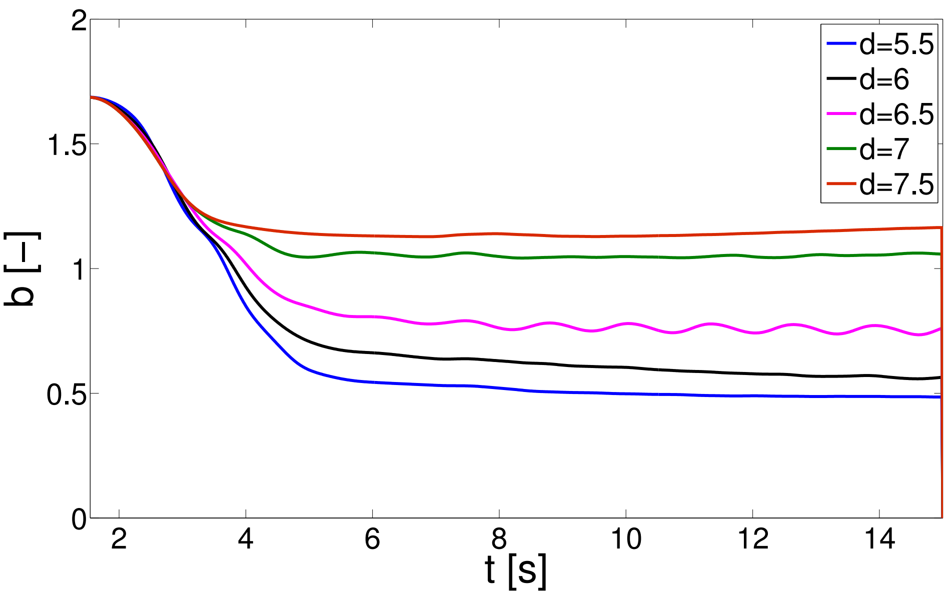


Figure 26 Graph of size of the semi-major axis a for different widths of the corridor, for swarm of 25 MAVs

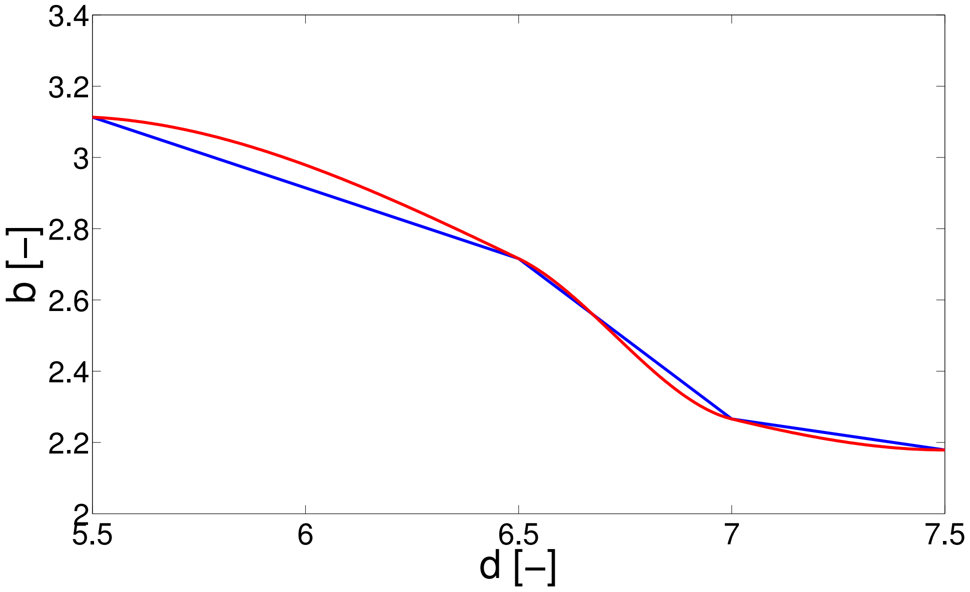


Figure 27 Graph of dependency of size of the semi-minor axis b on width of the corridor d for swarm of 25 MAVs. The red line is a polyline fitted on the measured data.

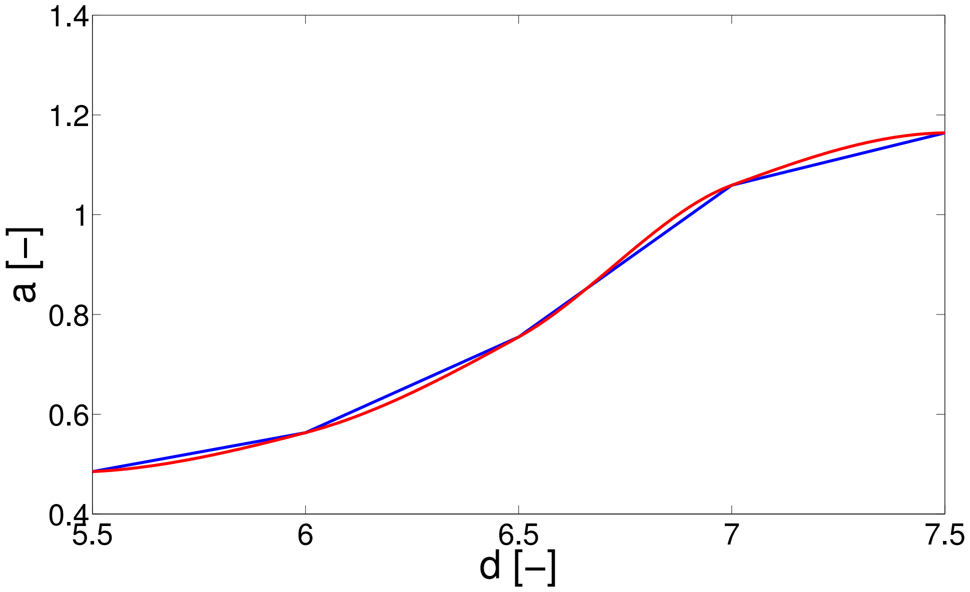


Figure 28 Graph of dependency of size of the semi-major axis a on width of the corridor d for swarm of 25 MAVs. The red line is a polyline fitted on the measured data.

Comparison

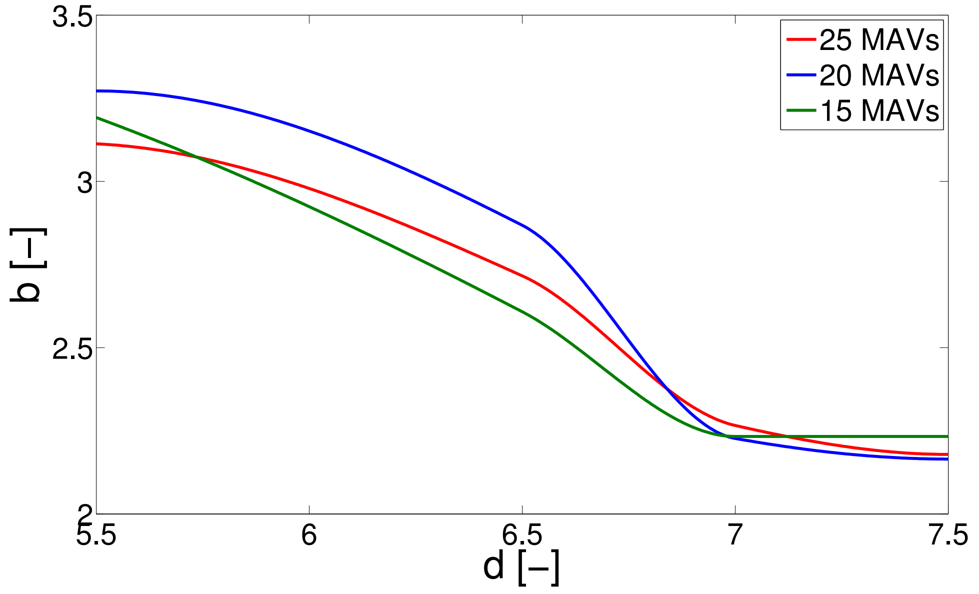


Figure 29 Graph comparing dependencies of sizes of the semi-minor axes b on width of the corridor for different sizes of the swarm.

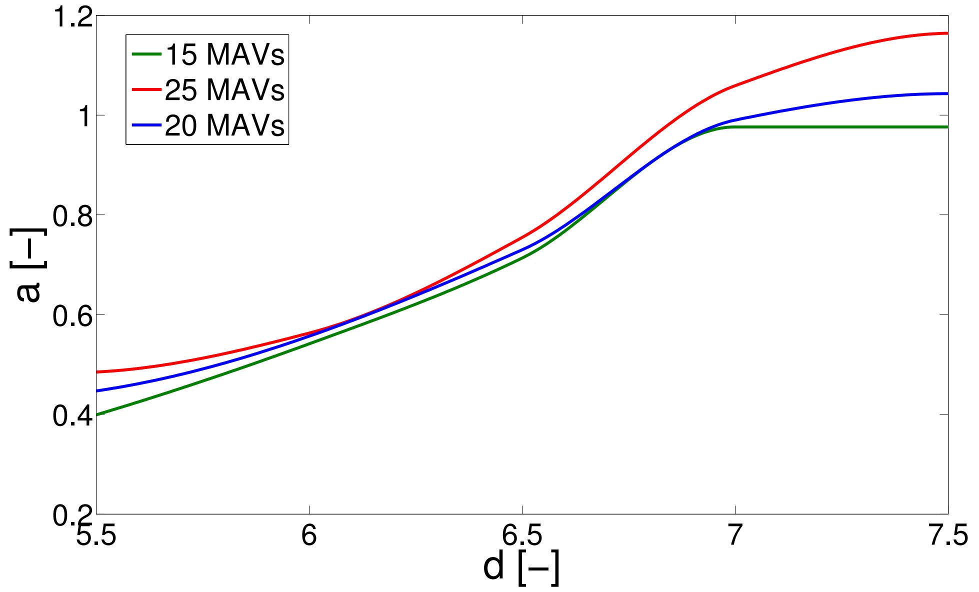


Figure 30 Graph comparing dependencies of sizes of the semi-major axes a on width of the corridor for different sizes of the swarm.

6 Conclusion

There were created functions for more convenient setting of the scenario, so it is possible to configure the simulation in few steps. Possible changes in the configuration were described in chapter 5. Results of the implementation are showed and discussed

The aim of this thesis was to make the implementation of the control model of swarm behaviour more variable so it could be one day used in a real quad-rotors. The original algorithm for simulating behaviour of swarm of an unmanned aerial vehicles was enhanced of features which should contribute to this vision. The individual enhancements are listed in next paragraphs.

There was added a possibility to use obstacles consisting of convex polygons using GJK library. The MAVs are now able to avoid obstacles made of polygons and any environment can be represented by polygons so the algorithm should be quite flexible in this matter. The swarm should now be capable of moving without any collisions in complex environment such as inside of a building.

It is now also possible for the swarm to follow some prescribed trajectory. Using static goal in complex environment was not possible, because the swarm cannot avoid big obstacles if they are in its trajectory. This feature could be also usable in many tasks i.e. for monitoring of some area.

In case the path is not defined,the algorithm was made compatible with "Tunel", which is script for creating Voronoi diagram of 3D environment and path planning using A* algorithm. It is now possible to set start and end point in the environment and the algorithm finds a followable path. But the algorithm needs to be a little, because does not work with obstacles that has one dimension many times bigger or smaller than the other two dimensions. This problem will be either solved with the author or some other algorithm will be used.

There were also added functions for easier configuration of the scenario. So the environment can be set up using function for creating blocks, custom hexahedrons or tables and chairs. But there still remains possibility of defining custom obstacles as long as they are convex.

In section 5.1 was made a list of possible changes in configuration of the simulation and also step by step manual for creating a scenario for simulating.

Finally in section 5.2 were made experiments with the algorithm. And as can be seen from the pictures of simulation the swarm behaves according to expectations. Also research of behaviour of the swarm according to width of a corridor in which it moves was done there. Its results are documented by a series of graphs in this section. From the gathered data it seems that width of the swarm does not matter on size of it, because it adapts to the corridor and spread vertically.

All parts of the thesis assignment were fulfilled:

- Method for moving target was implemented.
- Two libraries - GJK and Swift++ , which are capable of computing distance between convex obstacles was studied and GJK was chosen.

- The GJK library was implemented into the original algorithm.
- There were made many simulations in different scenarios and experimenting with corridors of different width and the gathered data were analysed.

The algorithm as it stands is still not ready for usage in real devices, but it is few steps closer now. There is still space for improvements in the path planning algorithm, because it is still not fully working and the whole algorithm could be made quicker by right optimization. Before tests with the real quad-rotors the constants of the control model would have to configured for some particular device. The MAVs must be also firstly equipped with sensors or cameras sufficient for reliable localization of the neighbouring individuals and obstacle detection. Other thing that is needed is some kind of localization system, so the location of the MAVs can be controlled in unknown environment. Hopefully, this project will some day reach its final stage.

Bibliography

[1] Jan Vakula. *Escape Behavior in Swarm of Unmaned Helicopters.* English. 2012.

[2] Libor Přeučil Martin Saska Jan Vakula. “Swarms of micro aerial vehicles stabilized under a visual relative localization”. In: ().

[3] C. W. Reynolds. “Flocks, herds, and schools: A distributed behavioral model”. In: *Computer Graphics.* 1987, pp. 25–34.

[4] Stephen Ehmann. *Speedy Walking via Improved Feature Testing for Non-Convex Objects.* English. GAMMA Research Group. 2001. URL: `http://gamma.cs.unc.edu/SWIFT++/`.

[5] C. B. Barber. *Ghull.* URL: `http://www.qhull.org`.

[6] Emer G. Gilbert et al. *IEEE Journal of Robotics and Automation.* English. Vol. 4. 2. 1988. Chap. A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space.

[7] *Simplex.* English. 2014. URL: `http://mathworld.wolfram.com/Simplex.html`.

[8] Gino van den Bergen. “A Fast and Robust GJK Implementation for Collision Detection of Convex Objects”. Paper. Department of Mathematics and Computing Science, Eindhoven University of Technology, 1999.

[9] *GJK Algorithm.* 2011. URL: `http://entropyinteractive.com/2011/04/gjk-algorithm/`.

[10] D. W. Jowhnson. “The optimization of robot motion in presence of obstacles”. Ph.D dissertation. University of Michigan, 1987.

[11] DR. Cameron Stephen. *Dr.* English. Version 2.4. 1998. URL: `http://www.cs.ox.ac.uk/stephen.cameron/distances`.

[12] Franz Aurenhammer. “Voronoi Diagrams - A Survey of a Fundamental Deometric Data Structure”. Survey. Institute für Informationsverarbeitung Technische Universität Graz, 1991.

[13] Steven M. LaValle. *PLanning Algorithms.* Cambridge University Press, 2006. Chap. 2. URL: `http://planning.cs.uiuc.edu/`.

[14] Ohad Gal. *fit_ellipse.* 2003. URL: `http://www.mathworks.com/matlabcentral/fileexchange/3215-fit-ellipse`.