ROBOTICS
AND
MECHATRONICS

Optimal Event Handling by Multiple UAVs

M. (Martijn) de Roo

MSc Report

Committee:
Prof.dr.ir S. Stramigioli
Dr. R. Carloni
Dr. P. Frasca
Dr. W.S. Rossi

July 2015

018RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE.

MIRA
BIOMEDICAL TECHNOLOGY
AND TECHNICAL MEDICINE

CTIT

PREFACE

This MSc report in front of you is the result of a graduation project at the research group Robotics and Mechatronics of the University of Twente.

Several persons have contributed both academically and practically to this thesis, I am very thankful to them.

> “
>
> *The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!', but 'That's funny ...'*
>
> *- Isaac Asimov* ”

Martijn de Roo
Enschede, July 2015

CONTENTS

I Introduction 1
I-A Group System . . . 1
I-B Individual System . . . 1

II Control Architecture 2

III Group System 2
III-A Optimal coverage problem . . . 2
III-B Gradient descent optimization . . . 3
III-C Implementation of the control law . . . 4

IV Individual System 4
IV-A UAV . . . 4
IV-B Path planning . . . 5
IV-C Controller . . . 6
IV-C1 Position control . . . 6
IV-C2 Path following . . . 6
IV-D Collision avoidance . . . 7

V Simulations 7
V-A Hardware . . . 7
V-B Results 20-sim . . . 8
V-C Results ROS . . . 8
V-D Results MATLAB . . . 9

VI Experiments 9
VI-A Hardware . . . 9
VI-B Results . . . 10

VII Conclusion 11

VIII Recommendations 11

Appendix A: Algorithm group system 12

Appendix B: Generalization Metric Manifold 14
B-A Gradient descent optimization . . . 14

Appendix C: Path Following 14
C-A Radius . . . 14
C-B Error radius . . . 14
C-C Line of sight . . . 16
C-D Curvature based path following . . . 16

Appendix D: Circumcircle calculation 17

Appendix E: Manual 17
E-A Prerequisites . . . 17
E-B Installation . . . 18
E-C Usage . . . 18

References 19

Optimal Event Handling by Multiple UAVs

Martijn de Roo, Paolo Frasca, and Raffaella Carloni

***Abstract*—In this paper a control architecture is presented, designed for event handling by a UAV (Unmanned Aerial Vehicle) network with an optimal approach. The architecture allows the calculations for the placement of UAVs in an area for tasks that need to be served by more than one UAV at the same time. The feasibility of this method is shown in simulations and is made ready for a test setup consisting out of several quadcopters.**

***Index Terms*—UAV, quadcopters.**

I. Introduction

EVENT handling can be important, depending on the nature of the events. An example of this is the SHERPA project [1], which aims to develop a robotic platform with UAVs and ground robots to support search and rescue activities. Being able to manage these activities as optimal as possible is of high interest, because it will offer the possibility to save time in situations where time is not enough. Four subtopics are identified that combined can achieve optimal event handling: (1) optimal control of the UAVs, (2) achieving optimal paths through an area, (3) optimal coverage of an area such that UAVs are placed in positions where they can serve the most events, (4) optimal event handling such that events can be serviced as fast as possible, with the main focus being on the optimal coverage.

The contribution that will be delivered is a system that calculates the position of the UAVs based on where events are likely to occur and rearranges the positions of the UAVs to meet this coverage, allowing the response of UAVs to events.

We continue by splitting our approach, one part directed at the coverage and event handling, which effects all UAVs, the *Group System*, and the other part directed to control and paths, the *Individual System*.

A. Group System

The group system delivers the distribution of the UAVs and selects which UAV goes where. The coverage of an area by UAVs needs to be solved; ideally the solution takes in account a certain probability distribution of events occurring in an area and the possibility of requiring more than one UAV for an event. The dynamic property of the UAV makes the problem rather complicated and does not add much merit under the assumption that the UAV has a sufficient fast turning rate, like a quadcopter. A common solution to the coverage problem are Voronoi tessellations [2]–[4]. Voronoi tessellations divide space in a number of regions based on an set of generators, on which the resulting regions give the area for which that generator is closer than any other generator. While this works in Euclidean space, it is harder to achieve in non-convex space, as argued in [3], which gives a solution to the non-convex Voronoi problem by discretizing it and solving it with Lloyd's algorithm. The downside of this is that the problem can be a computationally heavy approach to solve. The distribution of UAVs can be solved offline if the amount of UAVs and geography is known, so in some applications computational heavy solutions are allowed.

While the research in [3] is of interest for us, it does not cover for the possibility of requiring more than one UAV for an event, thus it will be extended for multiple density functions that depend on the amount of required UAVs.

The other part of the group system is the selection of which UAV has to go where. Ideally the time required for an event is known and new events near the currently occupied UAV might be able to be processed faster by waiting for the occupied UAV to finish. This is a rather extended topic but might be interesting for extending the framework. The time for an event is assumed to be unknown, therefore no sequential events are considered and the system will work on a first come first served process.

For the selection of the UAV that has to respond to the appearance of an event, the path planning can be used; by calculating for each UAV the length of the path to the event and taking the shortest one of them, the UAV can be selected that has to go. For this it is assumed that the longer the distance is, the more time it takes before an event can be serviced.

B. Individual System

The individual system manages the movement of the UAV and ensures that no collisions occur. Ideally a solution is used that is able to take in account all the UAVs at the same time, takes in account the paths they follow and avoids possible collisions beforehand by changing the paths they follow. A decentralized approach for this is used, otherwise one central configuration has to be solved, which is a NP-complete problem [5] and is hard to solve in real-time.

Because the system is decoupled, the path planning will be done separately from the control action, which means that separate obstacle avoidance is required. As the chance of a collision occurring is rather small, because UAVs will tend to serve events close to them, it might only be useful in close spaces.

Collision avoidance is a topic in which a lot of research has been done [6]–[8]. Solutions that are available are suitable, but are too advanced for a simple integration, therefore a basic collision avoidance will be proposed to solve this.

This work has been partially funded by the European Commission's Seventh Framework Programme as part of the project SHERPA under grant no. 600958. The authors are with the Faculty of Electrical Engineering, Mathematics and Computer Science, CTIT Institute, University of Twente, The Netherlands. Email: m.deroo-1@student.utwente.nl, {p.frasca, r.carloni}@utwente.nl.

The individual system will be addressed with the following individual topics: control, path planning and collision avoidance.

II. Control Architecture

An overview of the control architecture is shown in Fig. 1. The architecture consists of an *Individual System* and a *Group System* that act based on *Events*.

The *Group System* determines which UAV goes where, the *Coverage* module determines the placement of the UAV(s) based on the area and a density function for each amount of UAV(s) that is needed for events in the area, the *Event handling* module determines which UAV(s) is(are) closest to the event and then forwards the event location to the *Individual System* of the UAV(s).

The *Individual system* takes the UAV from its original position to the position of the event. The *Path Planning* module determines the path that needs to be travelled and takes into account the area, the *Control and Avoidance* module then applies a control on the path to make sure that it is followed as close as possible, without collisions occurring with other objects. The output of the *Control and Avoidance* module is sent to the UAV to control it.

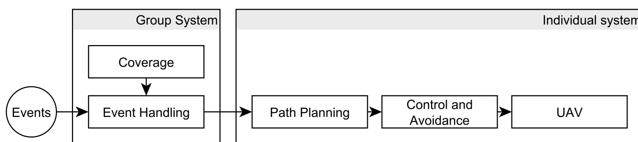

Fig. 1. Control Architecture, shows the global overview of the system on how events are handled, first by the group system, and then by the individual system.

III. Group System

The goal of the group system is to optimally assign UAVs to the events. In our system, optimality is defined as minimizing the average waiting time incurred by the events. This goal is achieved by deploying the group of UAVs in an optimal configuration (via the *Coverage* module) and by assigning, upon the appearance of an event, the closest UAVs to service it (via the *Event handling* module). The key part here is optimizing the configuration (i.e., the positions) of the UAVs while they wait for the appearance of the events. In this section, we provide the formal description and a solution for this optimization problem.

A. Optimal coverage problem

Let Ω be a bounded connected domain of $\mathbb{R}^N$, where in our applications $N \in \{1, 2, 3\}$. In this domain, events take place according to a random spatial process. The events are heterogeneous in nature and thus require different numbers of UAVs to be serviced: consequently, we define for each $m \in \{1, \dots, M\}$ a smooth density function $\phi_m : \Omega \to \mathbb{R}_{>0}$, representing at each point in Ω the density of events requiring m UAVs. Let $p \in \Omega^n$ be the vector of the UAV positions, where the position of the ith UAV is given by p_i and let $||q - p_i||$ denote the Euclidean distance between p_i and a generic point q in Ω. For the sake of generality and in order to account for heterogeneities among the UAVs in terms of speed, we define for each $i \in \{1, \dots, n\}$ a smooth strictly increasing convex function $f_i : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$: consequently, $f_i(||p_i - q||)$ will represent the travelling time of UAV i from its current position to q. Whenever m UAVs are required by an event, we assume that the event cannot be serviced until it has been reached by all m UAVs: hence, the waiting time of the event is the travelling time of its m-th closest UAV. To formalize this fact, we shall write $m\text{-}\min S$ to denote the m-th smallest element in a finite set of real numbers S. With this notation in place, we are ready to define the following integral cost function, which represents the *expected service time of an event*,

$$\mathscr{J}_{exp}(p) = \sum_{m=1}^{M} \int_{\Omega} \underset{i \in \{1,\dots,n\}}{m\text{-}\min} f_i(||q - p_i||)\phi_m(q)\,\mathrm{d}q \quad (1)$$

and the corresponding optimization problem

$$\min_{p \in \Omega^n} \mathscr{J}_{exp}(p).$$

In the special case when $M = 1$, this problem boils down to optimizing

$$\int_{\Omega} \min_{i \in \{1,\dots,n\}} f_i(||q - p_i||)\phi(q)\,\mathrm{d}q,$$

which is extensively studied for instance in [9] with the use of Voronoi tessellations of the domain Ω. In fact, also the cost (1) can be conveniently rewritten by using a suitable decomposition of Ω into sub regions, in the following way:

Let W be a collection of subsets of Ω such that each region W_i^m is indexed by $i \in \{1, \dots, n\}$ and $m \in \{1, \dots, M\}$ and for each m the sub-collection $\{W_i^m\}_{i\in\{1,\dots,n\}}$ is a *tessellation* of Ω: that is, $\cup_{i=1}^n W_i^m = \Omega$ for every m and $W_i^m \cap W_j^m$ is a set of measure zero if $i \neq j$. Note that the regions W_i^m are not assumed to be connected, i.e., a region can consist out of several disjoint sub-regions. Given such a collection W, we can define the cost

$$\mathscr{J}(p, W) = \sum_{i=1}^{n}\sum_{m=1}^{M} \int_{W_i^m} f_i(||q - p_i||)\phi_m(q)\,\mathrm{d}q. \quad (2)$$

Writing this cost, we take each UAV i as responsible for servicing events in the region $\cup_m W_i^m$: in region W_i^1 it shall service all events, in region W_i^2 it shall service all events that require at least two UAVs, and so forth.

In order to link this generic cost with the expected cost (1) we need to define one specific collection Z. Given a configuration p with no coincident positions, let us then define the region Z_i^m as the region of Ω such that i is the mth closest UAV to the points in Z_i^m. More formally, let $Z_i^m = \{q \in \Omega : \exists H \subset \{1 \dots n\}$ such that $|H| = m - 1,\ i \notin H$ and $f_h(||q - p_h||) \leq f_i(||q - p_i||) \leq f_\ell(||q - p_\ell||)$ for all $h \in H,\ \ell \notin H,\ \ell \neq i\}$. Note that the sub-collection $\{Z_i^1\}_{i\in\{1,\dots,n\}}$ is just the classical Voronoi tessellation generated by p.

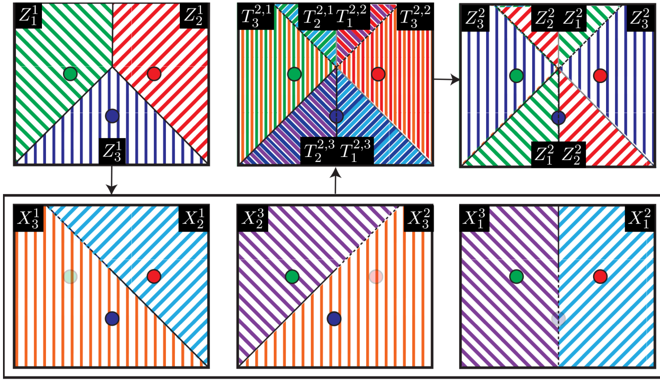

Fig. 2. Example of the construction of the regions Z_i^m with $n = 3$ and $m = 2$.

By virtue of its definition, Z minimizes $\mathscr{J}(p, W)$ as a function of W for fixed p. In fact, we have

$$\mathscr{J}_{exp}(p) = \mathscr{J}(p, Z) = \min_{W} \mathscr{J}(p, W)$$

for every $p \notin C$, where $C = \{p \in \Omega^n \, : \, \exists i, j \text{ s.t. } p_i = p_j\}$. This fact also permits to show –along the lines of [9, Theorem 2.16]– that the function $\mathscr{J}_{exp}$ is Lipschitz continuous on its domain and continuously differentiable outside the set of coincident configurations C.

Constructing the regions Z_i^m: For the sake of clarity and concreteness, we now illustrate with the help of Fig. 2 how to construct such regions Z_i^m for an example with $n = 3$ and $m = 2$. First, we construct the Voronoi regions Z_i^1 for each i. Next, to get the second closest tessellation of the Voronoi region Z_i^1, we leave out UAV i from the generators and construct the Voronoi tessellation generated by $\{1, \ldots, n\} \setminus \{i\}$, which we denote X_k^i; note that $\cup_{k \neq i} X_k^i = \Omega$. By taking the intersection of the new regions X_k^i and the regions Z_i^1 for all i we get $T_k^{2,i} = Z_i^1 \cap X_k^i$. Finally, we define $Z_k^2 = \cup_i T_k^{2,i}$.

The same process can be repeated for the $m = 3$ case using the regions $T_k^{2,i}$ as the starting point. Special care has to be taken to make sure that, for the creation of X_j^k we not only leave out k, but also the generators of all the parent tessellations $T_k^{2,i}$ is based on, in this case only i, this is also why we cannot use Z_k^2 because it is based on two different values of i. We give a more formal definition for the set of generators to also allow higher values of m, $\{1, \ldots, n\} \setminus (\{k\} \cup K)$, where $K = \{\text{all } j \text{ for which } T_k^{m,i} \cap Z_j^l \neq \emptyset \; \forall l < m-1\}$.

B. Gradient descent optimization

The minimization of the coverage function can be sought by following its gradient by the dynamics

$$\dot{p}_k = -\kappa \frac{\partial \mathscr{J}(P)}{\partial p_k} \quad \forall \, i \in \{1, \ldots, n\}, \tag{3}$$

where $\kappa \geq 0$. However, the coverage function is not convex, so a gradient descent will not in general reach a global minimum. We shall now compute the gradient under the simplifying assumption that $f_i(x) = x^2$: this assumption, according to [3], is often adequate in practice and can be relaxed at the cost of more involved notation. We then have

$$\frac{\partial \mathscr{J}(P)}{\partial p_k} = \frac{\partial}{\partial p_k} \sum_{m=1}^{M} \sum_{i=1}^{n} \int_{Z_i^m} ||p_i - q||^2 \phi_m(q) \, \mathrm{d}q \tag{4a}$$

$$= \sum_{m=1}^{M} \sum_{i=1}^{n} \int_{Z_i^m} \frac{\partial}{\partial p_k} ||p_i - q||^2 \phi_m(q) \, \mathrm{d}q \tag{4b}$$

$$= 2 \sum_{m=1}^{M} \sum_{i=1}^{n} \int_{Z_i^m} ||p_i - q|| \frac{\partial}{\partial p_k} ||p_i - q|| \phi_m(q) \, \mathrm{d}q \tag{4c}$$

$$= 2 \sum_{m=1}^{M} \int_{Z_k^m} ||p_k - q|| \frac{\partial}{\partial p_k} ||p_k - q|| \phi_m(q) \, \mathrm{d}q, \tag{4d}$$

where for step (4a) to (4b) it is allowed to take the derivative under the integral according to [4, Proposition 2] and for step (4c) to (4d) the summation is removed because $\frac{\partial}{\partial p_k} ||p_i - q|| = 0, \; \forall k \neq i$. Since $\frac{\partial}{\partial p_i} ||p_i - q|| = \frac{p_i - q}{||p_i - q||}$, Equation (3) becomes

$$\dot{p}_i = -2\kappa \sum_{m=1}^{M} \int_{Z_i^m} (p_i - q) \phi_m(q) \, \mathrm{d}q \tag{5}$$

which corresponds to steering each UAV towards a weighted "centre of mass" of its own responsibility region $\cup_m Z_i^m$.

Since the change in the objective function

$$\frac{\mathrm{d}\mathscr{J}(p(t))}{\mathrm{d}t} = \sum_{i=1}^{n} \frac{\partial \mathscr{J}(p(t))}{\partial p_i} \dot{p}_i = -\kappa \sum_{i=1}^{n} \left(\frac{\partial \mathscr{J}(p(t))}{\partial p_i} \right)^2$$

is negative semidefinite and the trajectories $p(t)$ stay in the bounded set Ω^n, we are in the position to apply a LaSalle invariance principle to prove convergence to the critical points of $\mathscr{J}(p)$. Actually, special care must be taken because $\mathscr{J}(p)$ is not continuously differentiable in C, hence the LaSalle principle should be applied in the set $\Omega^n \setminus C$, which is not closed and is dense in Ω^n. By using the generalized invariance principle in [9, E1.8], we thus obtain the following convergence result.

Proposition 1. *The trajectories $p(t)$ of the dynamics (3) converge to a set contained in the closure of*

$$\{q \in \Omega^n \, : \, \nabla \tilde{\mathscr{J}}(q) = 0\} \cup C.$$

In words, we have proved that the gradient dynamics converges to either the critical points of the cost or to configurations with coincident positions. Note that convergence to coincident positions is not a weakness of our analysis but instead an inherent feature of this optimization problem. In Fig. 3, we show that local minima of $\tilde{\mathscr{J}}$ can in fact involve coincident positions. Clearly, the implementation of such minima would be approximate, thus preventing collisions between the UAVs. The convergence of (3) is illustrated in Fig. 4.

In Appendix B we show the generalization to Ω in a metric manifold, along the lines of [3].

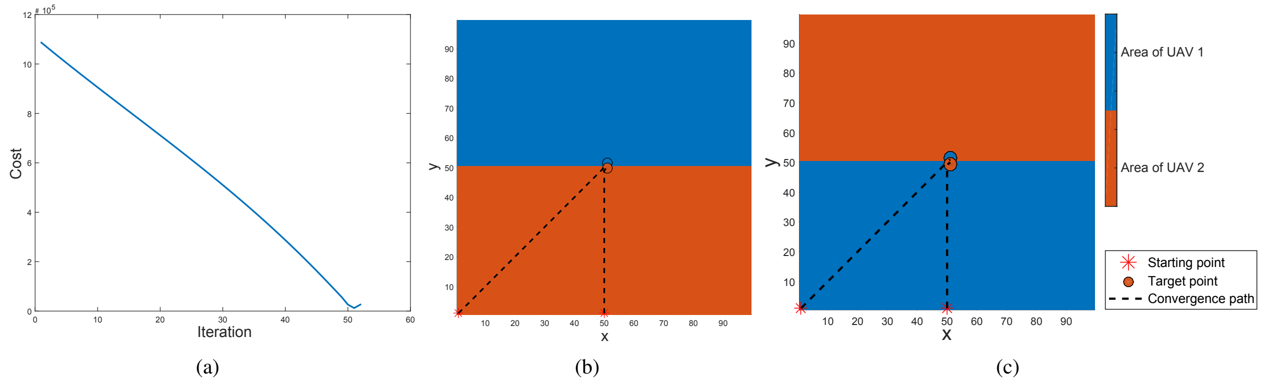

Fig. 3. Simulation of the group system for $n = 2$, $m = 2$ on a 100x100 grid.

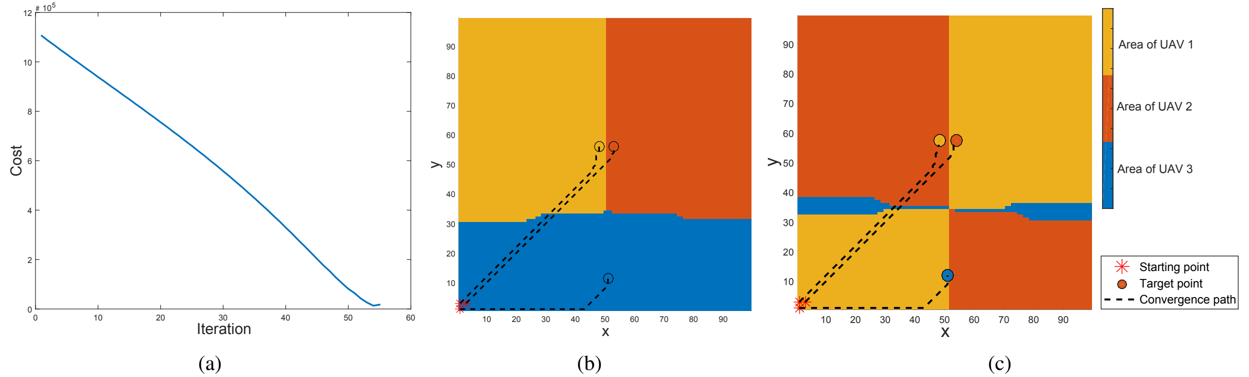

Fig. 4. Simulation of the group system for $n = 3$, $m = 2$ on a 100x100 grid.

C. *Implementation of the control law*

For the implementation of the control law in equation (5) the same approach as [10] is used, utilizing a discrete graph-search for achieving the implementation, resulting in an approximation of the algorithm. The pseudocode for the algorithm can be found in Appendix A.

Based on this pseudocode, an implementation is made in MATLAB and two simulations using this are executed. Both simulations assume a uniform density function for each m and $m = 2$ possible required UAVs on a grid of 100x100. It is assumed that there are no obstacles, this to show a clear working of the algorithm. The results are shown in Fig. 3 for $n = 2$ UAVs and Fig. 4 for $n = 3$ UAVs. The colours show which area belongs to which UAV and the dashed lines show the progress of the algorithm, for which in each iteration of the algorithm the UAV slowly converges to a final point that minimizes the cost function, as can be seen in Fig. 3(a) and Fig. 4(a).

The behaviour that each UAV is steered towards a "centre of mass" can be clearly seen in Fig. 3, if the regions of both the $m = 1$ (Fig. 3(b)) and $m = 2$ (Fig. 3(c)) are combined for the responsible UAVs, it can be seen that both UAVs have the same combined area, this is due to the fact that the second closest will always be the other UAV in the case of $n = 2$, resulting in the same regions.

The second closest area for each UAV can be clearly seen in Fig. 4(c). It shows that the distribution of the area for the second closest is also distributed in an even way yet shows that Area of UAV 1 and Area of UAV 2 are clearly larger, because these areas are based on a more central situated position.

While UAV 3 has less coverage in Fig. 4(c), it can be seen that it has a larger area in Fig. 4(b), which shows that the closest area influences the shape of the second closest and vice versa.

IV. INDIVIDUAL SYSTEM

The goal of the individual system is to guide the UAV to the event, which is achieved by implementing several controllers, a path planning system and obstacle avoidance.

The type of UAV that will be used for the individual system is a quadcopter, in particular the Crazyflie 2.0 from Bitcraze [11], which has a built-in attitude control available. No internal software changes will be done to the quadcopter, therefore a controller that uses the available attitude control will have to be created.

According to our decoupled approach the subjects of UAV, path planning, control and avoidance will be separately discussed.

A. *UAV*

The UAV that we consider is a quadcopter, schematically depicted in Fig. 5. The dynamics of a quadcopter are based on the model in [12]; in our analysis we add a drag force for each velocity in the inertial frame to get a more realistic behaviour.

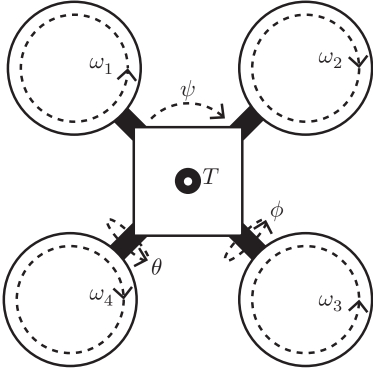

Fig. 5. Quadcopter with the angular velocities of the rotors, ω_1, ω_2, ω_3 and ω_4. The roll, pitch and yaw are shown with ϕ, θ and ψ, the thrust is shown with T.

We thus have

$$m\ddot{\xi} = -mge_z + Re_zT - C\dot{\xi} \tag{6}$$

$$I\dot{\Omega} = -\Omega \times I\Omega - G + \tau \tag{7}$$

$$T = b\sum_{i=1}^{4}\omega_i^2$$

$$\tau = \begin{bmatrix} db(\omega_2^2 - \omega_4^2) \\ db(\omega_1^2 - \omega_3^2) \\ \kappa(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) \end{bmatrix} \tag{8}$$

$$G = \sum_{i=1}^{4} I_r(\Omega \times e_z)(-1)^{i+1}\omega_i, \tag{9}$$

where $m \in \mathbb{R}$ is the mass, $\xi \in \mathbb{R}^3$ the position of the body-fixed frame in the inertial frame, $g \in \mathbb{R}$ the acceleration due to gravity, $R \in SO(3)$ the rotation of the airframe, $T \in \mathbb{R}$ the thrust applied to the airframe and $C \in \mathbb{R}^{3x3}$ the diagonal drag force matrix. $I \in \mathbb{R}^3$ denotes the diagonal inertial matrix of the airframe with respect to the base, $\Omega \in \mathbb{R}^3$ the angular velocity of the airframe in the body-fixed frame and G the gyroscopic forces and τ the external torque.

The constants required to calculate the thrust T, gyroscopic forces G and the external torque τ are $b \in \mathbb{R}$ (the proportionality constant of the rotor), $d \in \mathbb{R}$ (the distance of rotor to the centre of mass of the airframe) and $\kappa \in \mathbb{R}$ (the rotor thrust factor).

For R we take the Z_ψ - Y_θ - X_ϕ (yaw - pitch - roll) Euler angles for the mapping between the inertial and body-fixed frame, resulting in

$$R = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta c\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}, \tag{10}$$

where $s = \sin$, $c = \cos$ and $t = \tan$.

For simulations we want to express the dynamics of the quadcopter in the inertial frame. To transform the angular velocity Ω of the body fixed frame to the angular velocity $\dot{\eta}$ in the inertial frame, $\Omega = W\dot{\eta}$ or $\dot{\eta} = W^{-1}\Omega$, we use the same mapping of (yaw - pitch - roll). Assuming small changes, the matrix will first have to be rotated before it can be transformed, requiring two transformations for the first angle, one for the second and none for the last, resulting in

$$\Omega = Z_\psi Y_\theta \begin{bmatrix} 0 \\ 0 \\ \dot{\phi} \end{bmatrix} + Z_\psi \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \dot{\psi} \\ 0 \\ 0 \end{bmatrix} = W\dot{\eta},$$

which can be solved for W, resulting into

$$W = \begin{bmatrix} 1 & 0 & -s\theta \\ 0 & c\phi & c\theta s\phi \\ 0 & -s\phi & c_\phi c_\theta \end{bmatrix}, \; W^{-1} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix}.$$

To transform the derivative of the angular velocity of the body fixed frame to the inertial frame we require the derivative of the transformation.

$$\ddot{\eta} = \frac{d}{dt}(W^{-1}\Omega) = \frac{d}{dt}(W^{-1})\Omega + W^{-1}\dot{\Omega} \tag{11}$$

where $\frac{d}{dt}(W^{-1})$ is given by

$$\frac{d}{dt}(W^{-1}) = \begin{bmatrix} 0 & \dot{\phi}c\phi t\theta + \dot{\theta}\frac{s\phi}{c^2\theta} & -\dot{\phi}s\phi t\theta + \dot{\theta}\frac{c\phi}{c^2\theta} \\ 0 & -\dot{\phi}s\phi & -\dot{\phi}c\phi \\ 0 & \dot{\phi}\frac{c\phi}{c\theta} + \dot{\theta}\frac{s\phi t\theta}{c\theta} & -\dot{\phi}\frac{s\phi}{c\theta} + \dot{\theta}c\phi\frac{t\theta}{c\theta} \end{bmatrix}$$

The simulation can then be solved by using equations, (6), (7) and (11), resulting in

$$\ddot{\xi} = -ge_z + \frac{1}{m}RT - \frac{1}{m}C\dot{\xi} \tag{12}$$

$$\ddot{\eta} = \frac{d}{dt}(W^{-1})\Omega + W^{-1}I^{-1}(-\Omega \times I\Omega - G + \tau) \tag{13}$$

where the equations (12) and (13) represent the UAV in the inertial frame with $\Omega = W\dot{\eta}$.

B. Path planning

A path has to be calculated on which the UAV has to travel, which has to be done rather quickly because the UAV has to travel the moment a destination is known. Therefore the calculation is a trade-off between optimality and speed, as an optimal solution tends to take more time to calculate.

Several methods exist to create a path, [13] gives an overview of the methods available for path planning.

Our group system is solved using a graph approach, due to this we also employ a graph approach to generate the path. To find the path on a grid several methods can be used. The A* algorithm [14] or the Dijkstra's algorithm [15] are well known algorithms for a graph approach; both ensure that the shortest path will result. In some cases the A* algorithm can be faster, because Dijkstra's algorithm visits the closest vertex that it has not visited yet to see if it is the destination, while the A* algorithm allows heuristics and can prefer a direction. For this we employ the A* search algorithm which will generate a set of points that represent the path that will have to be taken by the UAV.

Fig. 6 shows the result of the A* algorithm in MATLAB; the algorithm gives the closest path for each target, all possible

9 paths are calculated and, as is said for the event handling in Section I-A, the combination that is shortest is selected. In case several combinations of paths have the same length, the combination that is shortest and, if possible, has no overlapping paths that can cause collisions, is selected.

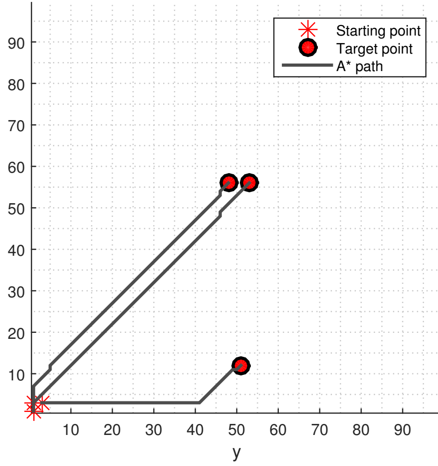

Fig. 6. A* shortest path for a 100x100 grid.

C. Controller

A controller is required to control the quadcopter, in particular we are interested in a controller that makes it possible to follow a path and stay in one position. The Crazyflie 2.0 has a built-in attitude control available. No internal software changes will be done to the quadcopter, therefore a controller using the available attitude PID control will have to be created.

A quadcopter can be controlled by changing the thrust of the four propellers, as can be seen in equation (6), which also shows that in order to make the quadcopter move, the roll and pitch have to be controlled to cause a change in the rotational matrix, which then causes a change in the direction of the thrust and, assuming small angles, displacement to the direction it is leaning to.

The control inputs of the quadcopter are shown in equation (8). By making sure that $\omega_1 \neq \omega_3$, a pitch is achieved, the same goes for $\omega_2 \neq \omega_4$ which will cause roll. A change in the yaw can be achieved by setting $\omega_1 + \omega_3 \neq \omega_2 + \omega_4$

If small angles are assumed for the roll and pitch, the setpoints of the roll and pitch can be linearized and be used as the desired velocity, meaning that the more the quadcopter leans towards a direction, the faster it will go into that direction, assuming that the altitude control responds faster and keeps the quadcopter levelled.

Movement through an area can be done in several ways; using just the destination as a set-point and apply PID position control to go there is possible, but can easily lead to collisions or situations where the destination cannot be reached, additionally position control with waypoints cause the quadcopter to converge to each waypoint that it has to visit, thus slowing it down. Solutions to this are possible but not elegant, as is shown in Appendix C. Therefore we split the type of control into *Position control* to stay at a position and *Path following control* to follow a path. An overview of the complete controller is shown in Fig. 7.

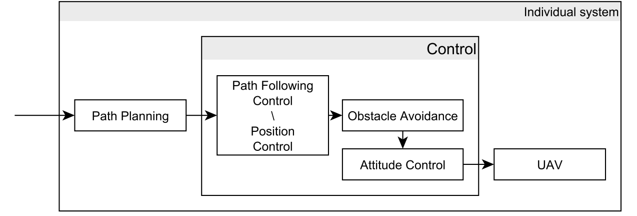

Fig. 7. Control scheme of the system, the quadcopter has an input $\omega_{[1...4]}$ which is controlled by an attitude controller that controls based on the angle of the quadcopter. Depending on the selection of the position controller or the path following controller, a desired control action is generated, the resulting desired control action is then processed by the obstacle avoidance and send to the attitude controller.

1) Position control: For the position control a PID controller is used; it is easy to implement and only needs to make sure that the quadcopter stays in one position. Because we assumed that for small angles the change of angles can be considered as setting a velocity, the position control becomes rather straightforward. The error will be the difference of the current position and the desired position and the output will be a velocity in the direction of the error. The output will have to be transformed to take in account the yaw of the quadcopter, thus

$$\begin{bmatrix} v'_x \\ v'_y \end{bmatrix} = \begin{bmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}. \tag{14}$$

2) Path following: To follow the path we use a similar method as is described in [16] and [17]: we set a position error that we want to minimize normal to the line that goes from the previous waypoint to the next waypoint, and a desired velocity in the tangent direction of the path. This allows us to move through a set of waypoints without stopping or converging to each of them, which happens if position control is used for the waypoints. The definition of the path as described in [17] is used, $P \in N \times \mathbb{R}^3$, a sequence of N waypoints, x_i and desired velocity v_i along the path segment P_i connecting waypoint i to $i+1$, and the current position $x(t)$. The cross-track e_{ct} error of this method can be defined as

$$e_{ct} = (x_i - x(t))\tilde{n} + (x_i - x(t))\tilde{b} \tag{15}$$

where $\tilde{n}$ is the normal and $\tilde{b}$ the unit vector to the track. This way we can minimize the cross-track error and still move along it, because the position control does not take in account the tangent direction. No separate controller is required for tangent direction, setting the velocity in the tangent direction corresponds to sending a setpoint to the built-in controller directly. Transition to the next path segment occurs when the UAV is c_{error} distanced from the plane normal to the path at the end of the segment. The path following is schematically depicted in Fig. 8.

The controller requires a desired velocity. To get this velocity we propose to make the velocity dependant on the curvature of the path. The more curved the path is, the slower

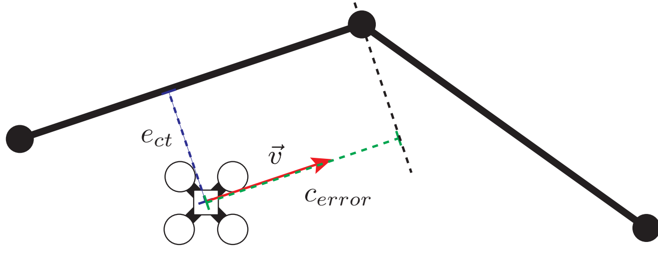

Fig. 8. Graphical overview of the path following system.

the desired velocity is. That means that v_{max} would be a straight line and v_{min} a 90 degree turn. To get the curvature for our waypoints we approximate it using a circumcircle, using 3 sequent waypoints to find a unique radius that can represent the curvature. A large radius would indicate an almost straight line, while a small radius would indicate a sharp turn.

The circumcircle of three points can be determined by solving the following determinant

$$det \begin{vmatrix} x^2+y^2 & x & y & 1 \\ x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_2 & y_2 & 1 \\ x_3^2+y_3^2 & x_3 & y_3 & 1 \end{vmatrix} = 0 \tag{16}$$

where (x, y) is the set of points of the circumcircle and (x_i, y_i) the i-th point. The determinant can be written in the form $(x-x_0)^2 + (y-y_0)^2 = r^2$, resulting in a radius r from the three points, as is shown in Appendix D.

The resulting radius r will result into ∞ when all the points are on a straight line, to make use of this the desired velocity is based on the inverse of the radius, making it zero, the following application is proposed

$$f(r) = \begin{cases} (v_{max} - v_{min}) - c\frac{1}{r} & \text{if } (v_{max} - v_{min}) - \frac{1}{r} > 0 \\ 0 & \text{if } (v_{max} - v_{min}) - \frac{1}{r} \leq 0 \end{cases}$$
$$v_{des}(r) = f(r) + v_{min},$$

where $f(r)$ is a function that scales the velocity based on the inverse radius, and makes sure that v_{max} can be achieved and that possible results of $(v_{max} - v_{min}) - \frac{1}{r}$ are 0 for values lower then 0, this to ensure that v_{min} is achieved. The constant $c \in \mathbb{R}$ can be used to tune the function.

An example of the method is shown in Fig. 9, the z axis describes how many degrees are desired for the velocity in the tangent direction, degrees is used due to the linearization of the velocity for the roll/pitch of the quadcopter.

D. Collision avoidance

To ensure that no crashes occur an avoidance algorithm is proposed, that has to ensure that the UAVs are not able to hit each other. For this the following safety system is used:

The system works on the fact that when UAVs are near each other and have no velocity direction towards each other, that they will not crash. No movement towards other UAVs means that it is not possible to collide based on the movement direction. The avoidance strategy checks all UAVs in a radius r around each UAV, r should be large enough for the UAV to be

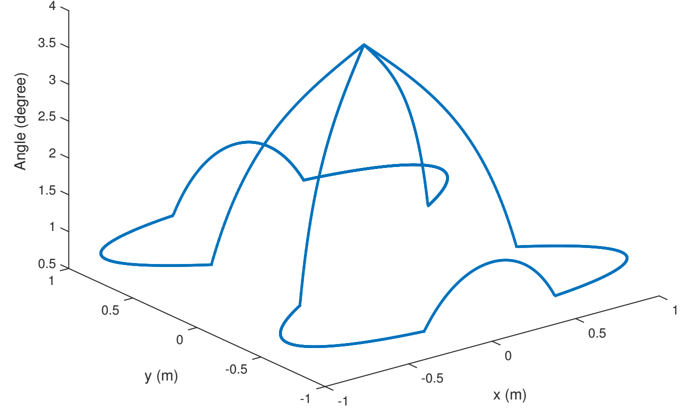

Fig. 9. Example of the velocity profile generation method, with $v_{max} = 0.065$ rad, $v_{min} = 0.015$ rad and $c = 0.15$, of a Lissajous figure.

able to get to a standstill. For each surrounding UAV the velocity vector is decomposed based on the direction vector between the UAV and each surrounding UAV, and saves the resulting tangent vector as the new desired velocity if the normal vector is positive; a negative normal vector would mean that the UAVs are moving away from each other. An example of this is shown in Fig. 10 and the algorithm is shown in Algorithm 1.

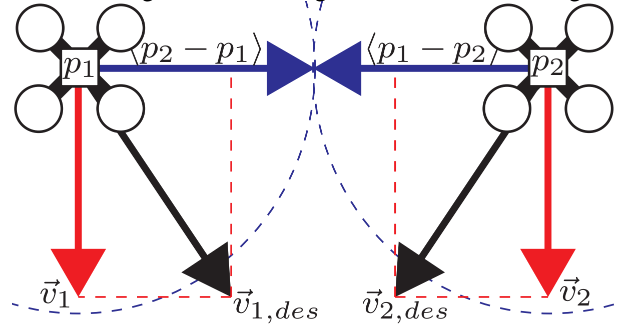

Fig. 10. Representation of the avoidance system: The desired velocity is shown as $\vec{v}_{\#,des}$. $\vec{v}_{\#}$ shows the new velocity, which is based on the normal vector of the desired velocity decomposed into the unit vector that points towards the other UAV within the collision circle, which is shown with the dotted circle.

V. Simulations

Simulations are used to confirm the proposed architecture.

A. Hardware

The hardware that is simulated is the Crazyflie 1.0 from Bitcraze [11]. At the time of the simulations the Crazyflie 2.0 was not yet available, the Crazyflie 1.0 is however similar, therefore the parameters of the Crazyflie 1.0 are being used. A research into the parameters has been done in [18]. For the dynamic simulation of the system 20-sim [19] is used, which allows 3D simulations of the quadcopter equations. In addition, it is also used to simulate the collision avoidance algorithm and path following controller. Simulations of the Group system and Individual system are made using ROS [20],

Algorithm 1 Avoidance

Input: $\vec{v}_{N,des}$, p_N, ψ_N, r
Output: $\vec{v}_N$

1: **for** all $n \in N$ **do** // $\vec{v}_{n,des}$ to the inertial frame
2: $\quad \vec{v}_{n,inertial} = Rot(-\psi_n)\vec{v}_{n,des}$
3: **end for**
4: **for** all $n \in N$ **do** // Remove conflicting parts of $\vec{v}$
5: $\quad$ **for** all $i \in N \setminus n$ **do**
6: $\quad\quad$ **if** $(||p_i - p_n|| < r)$ **then**
7: $\quad\quad\quad \hat{p} = \frac{p_i - p_n}{||p_i - p_n||}$ // Get the normal unit vector
8: $\quad\quad\quad \hat{p}_{tan} = Rot(\frac{\pi}{2})\hat{p}$ // Get the tangent unit vector
9: $\quad\quad\quad v_{norm} = \hat{p}'\vec{v}_{n,inertial}$ // Decompose the velocity
10: $\quad\quad\quad v_{tan} = \hat{p}'_{tan}\vec{v}_{n,inertial}$
11: $\quad\quad\quad$ **if** $v_{norm} > 0$ **then**
12: $\quad\quad\quad\quad \vec{v}_{n,inertial} = v_{tan}\vec{p}_{tan}$
13: $\quad\quad\quad$ **end if**
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: **end for**
17: **for** all $n \in N$ **do** // $\vec{v}_{n,inertial}$ to the body-fixed frame
18: $\quad \vec{v}_n = Rot(\psi_n)\vec{v}_{n,inertial}$
19: **end for**
20: **return** $\vec{v}_N$

Gazebo [21] and MATLAB. ROS is a robotic aimed framework that allows, in combination with the simulator Gazebo, an easy transfer from simulation code to experimental code. MATLAB is used for the Group system and the MATLAB ROS I/O [22] package is used for the communication with ROS.

The simulation was split between 20-sim and Gazebo because the initial design was done in 20-sim. The designed system was tested and then implemented in ROS, Gazebo was used to simulate a quadcopter to test that the ROS package was implemented correctly. For the collision avoidance a radius of 50 centimetres was used. A Lissajous figure is used to generate a path, this allows the velocity profile generation to be used effectively in the corners, to ensure that the quadcopter does not overshoot the path. The Lissajous figure also makes it possible to recreate the path easily, which ensures that the same path is used for all simulations and experiments. For the simulations in Gazebo a quadcopter implementation of [23] was used and for the velocity profile of the simulations the same parameters as Fig. 9 were used.

B. Results 20-sim

Two simulations are made, one is made with a quadcopter following a Lissajous figure, showing the effectiveness of the path following controller and velocity generation, and one with two identical quadcopters that follow a Lissajous figure, having starting positions opposite to each other moving the opposite directions, this to ensure that a collision occurs.

Fig. 11 shows the simulation result of a quadcopter following a Lissajous figure: The trajectory is closely followed except for the corners. Fig. 12 shows the cross-track error along the path: The error slightly exceeds six centimetres in the corners, a lower error can be achieved by changing the velocity profile.

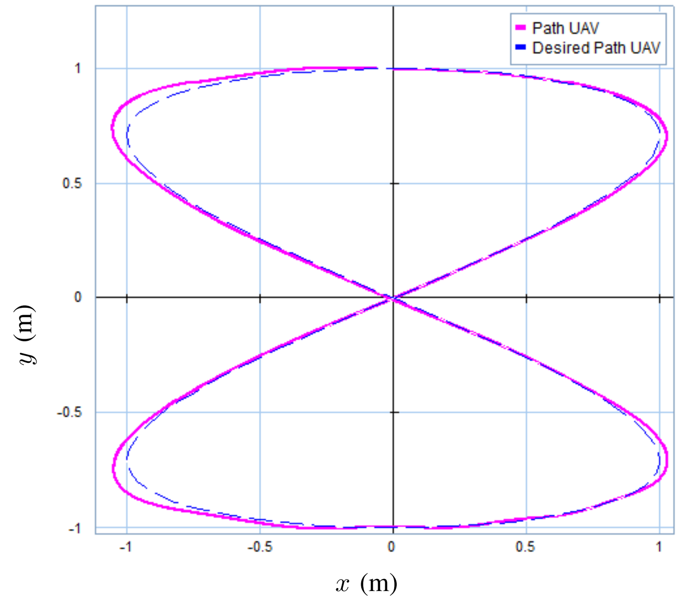

Fig. 11. Quadcopter following a Lissajous figure, starting at $x = 0$, $y = 1$ and moving clockwise. Simulated in 20-sim

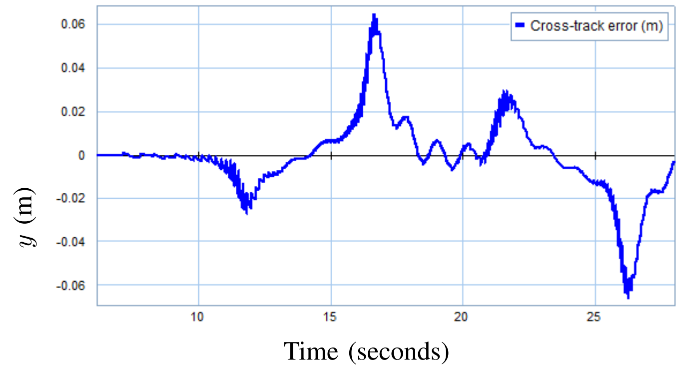

Fig. 12. Error of the quadcopter following a Lissajous figure. Simulated in 20-sim.

Fig. 13 shows the simulation result of the quadcopter following a trajectory. The trajectory is shown, two quadcopters, on opposite ends, are started at the same time and meet each other in the middle, at this point the obstacle avoidance takes control due to the quadcopters heading into a collision and ensures that the quadcopters do not collide.

The cross-track error is shown in Fig. 14. It can be seen that the error is closely followed, except for when the two UAVs are in collision. The saw like response is due to the approximation of the desired velocity, which consists out of 150 points of Fig. 9.

C. Results ROS

The system that was modelled in 20-sim is implemented in ROS. To test that the system does indeed show similar behaviour a simulation of it was made using Gazebo, for which a quadcopter implementation of [23] was used.

Fig. 15 shows the results of the implementation of the 20-sim model applied to a Lissajous figure. The results are similar to the ones from the 20-sim model in Fig. 11 and Fig. 12 but

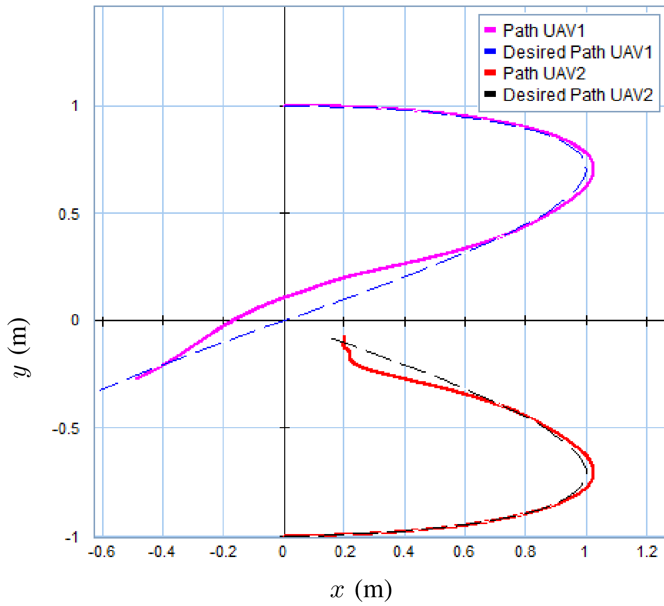

Fig. 13. Path of the quadcopter following a Lissajous figure while avoiding another quadcopter, both quadcopters start at $x = 0$. Simulated in 20-sim.

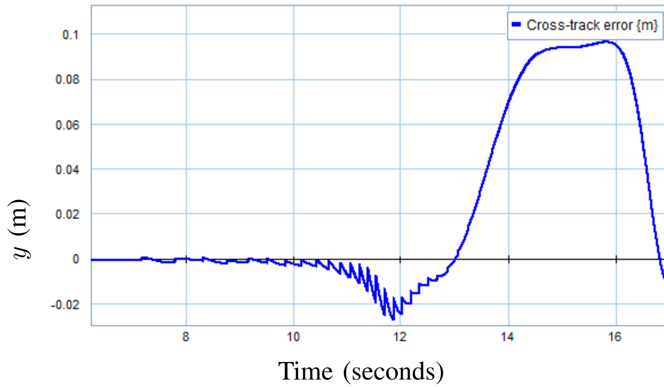

Fig. 14. Error of the quadcopter following a Lissajous figure while avoiding another quadcopter. Simulated in 20-sim.

does show a slightly larger error around four seconds which corresponds to the first corner. This is due to the attitude controller that is different for the 20-sim model and Gazebo model. The UAV is having too much acceleration after it starts from no movement, causing it to overshoot.

The implementation of the avoidance using ROS is shown in Fig. 16. No collision occurs and the system shows the same behaviour as the 20-sim model with a similar cross-track error.

D. Results MATLAB

The implementation of the group system was realized using MATLAB and is shown in Fig. 17. There is no difference for the system for simulation application or experimental because it only uses the position data to generate paths. The desired position where an event appears is due to user action.

Fig. 17 shows an application of the system, all optimal distributions were calculated beforehand using the results of the optimal coverage problem. The background shows the

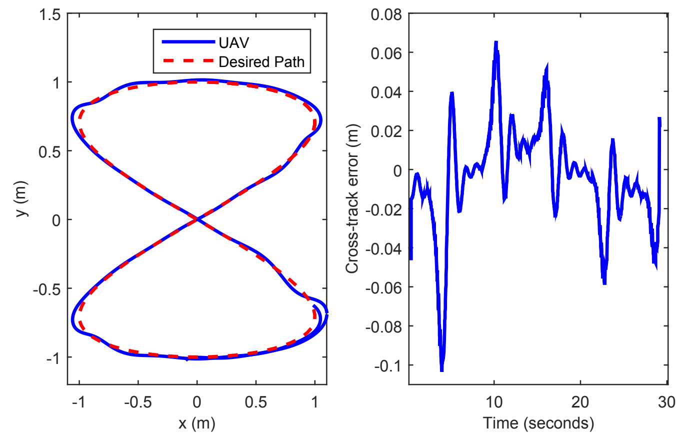

Fig. 15. Path following of a Lissajous figure, showing both the path and the cross-track error. Simulated in ROS/Gazebo

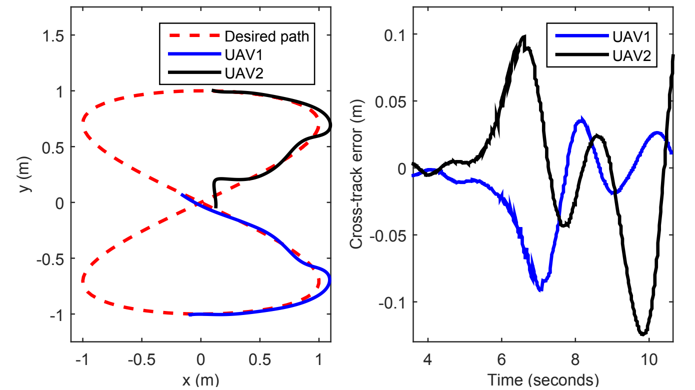

Fig. 16. Path following of a Lissajous figure while avoiding another quadcopter, showing both the path and the cross-track error, both quadcopters start at $x = 0$. Simulated in ROS/Gazebo.

regions of each UAV. At Fig. 17(a) the system is enabled, from the current positions of the UAVs a path is generated towards the optimal placing of the UAVs according to the optimal coverage for 3 available UAVs and the desired positions are achieved in Fig. 17(b). In Fig. 17(c) two points in the area are selected, for which paths are generated for the two closest UAVs and the available UAV 3 is sent to the optimal coverage position, the centre. In Fig. 17(d) UAV 1 reaches the desired position, at which the system sends it back to the optimal coverage for 2 UAVS, which requires both available UAVs in the middle. When UAV 2 reaches its desired position in Fig. 17(e), 3 UAVs are available, for which the systems generates paths to move the UAVs, the final positioning is then achieved Fig. 17(f), which is similar to Fig. 17(b).

VI. Experiments

For practical confirmation of the proposed architecture, experiments are performed based on the simulations.

A. Hardware

For the actual experiment ROS is used; the same code as for the simulation in Gazebo can be used due to the nature of ROS. An optical tracking system from Naturalpoint [24] is used to

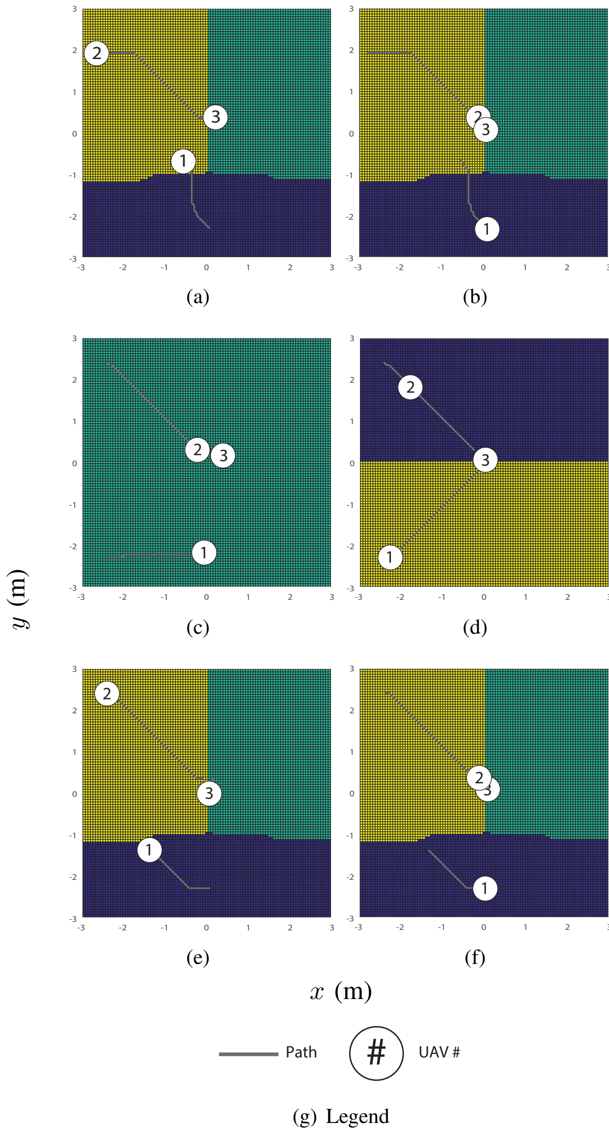

Fig. 17. Example of the group system showing the UAVs and the paths they take.

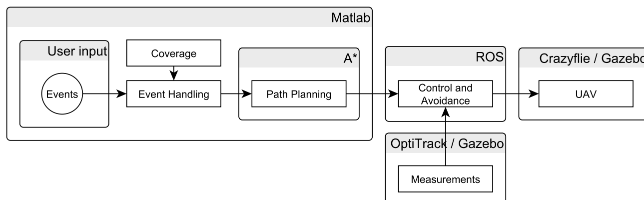

Fig. 18. Overview of the experimental setup.

keep track of the quadcopter; it allows a 100 Hz position and attitude tracking. An overview of the setup is shown in Fig. 18. For the actual experiment the Crazyflie 2.0 was used, fitted with retroreflective markers as is shown in Fig. 19, carbon tubes were used to keep additional weight as low as possible due to the maximum recommended payload of 15g.

Fig. 19. Photograph of the Crazyflie 2.0 that are used.

B. Results

The results of the experimental setup are shown in Fig. 20. The Lissajous figure is tracked by the UAV but has quite a large error when compared to the simulations. The main cause of this is due to an improperly balanced UAV and low-level controller. The balancing issue is due to tracking markers that are required to be fitted on the UAV, adding unbalance to it, as is shown in Fig. 19. Additional experiments to check if the avoidance system works were not made. The error of the path following control on the Lissajous figure can go up to 25 centimetres, which is a substantial error compared to the size of the UAV (WxHxD) 92x92x29mm. Due to the large error the avoidance would not be shown in a close proximity, nor would it ensure that no collisions occur. It could be tested if a radius r for the collision circle is chosen such that the error is small compared to r, but this would require a rather large experimental area.

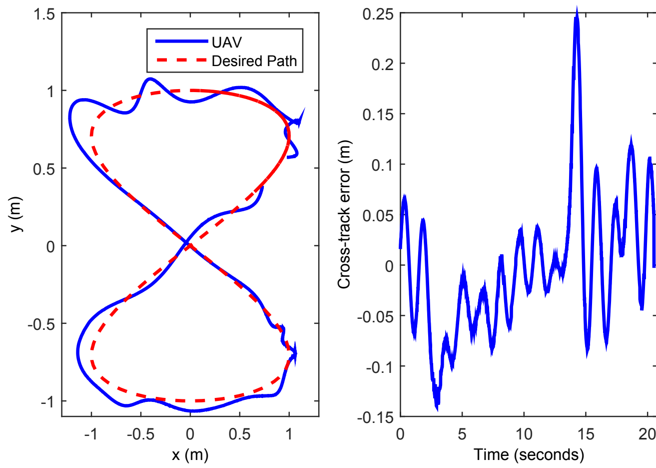

Fig. 20. Path following of a Lissajous figure, showing both the path and the cross-track error.

To get a better understanding of the large error in the system the position control error is also looked at, which is shown in Fig. 21. For the control the x-direction and y-direction used the same control settings. It can be seen that for the x-direction the error is within a band of 15 centimetres and shows a slight offset to the negative side, the y direction has a larger error band of 25 centimetres, and shows less control actions. The markers were attached in the y-direction, which explains the different results. A better result might be gained if the attitude control is tuned to take in account the additional mass due to the markers, which affects the roll.

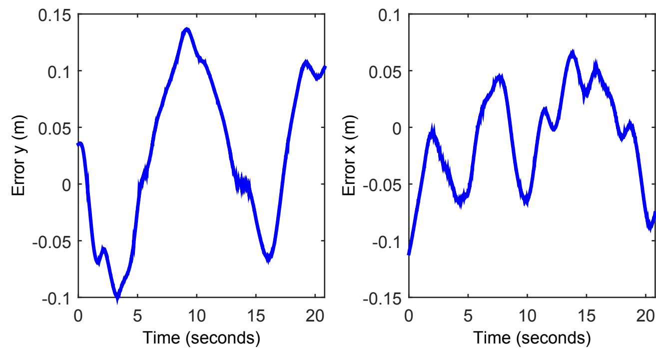

Fig. 21. Error of the position control for the x-direction and y-direction.

VII. Conclusion

A working framework is proposed which allows the handling of events based on the minimization of costs over an area. The framework is designed and simulated, a practical implementation was tried but showed a large error, due to which the collision avoidance was not tested in practice.

For the optimal coverage problem that takes in account that events can require multiple UAVs, a control function is created which allows the calculation of optimal positions in an area based on density functions. The result of this allows the deployment of a group of UAVs on positions that are calculated such that events can be serviced as optimal as possible.

The simulations of the UAV in both ROS/Gazebo and 20-sim achieved a low error while being able to follow a path using the generated velocity profile that is based on the angle the path makes, while the proposed collision avoidance ensured that no collisions occurred.

VIII. Recommendations

Several recommendations can be made for the framework. The event handling is currently managed on a first-come, first-served basis, which works by taking the closest UAV for the event. A better solution is to take in account that UAVs might be able to manage multiple events sequential and optimize for this.

A velocity controller might be able to improve the results of the set-up. Position control does not take in account the velocity, so while the desired position is achieved, it will overshoot due to the UAV still having a velocity.

It is not possible to compensate for the markers with just the current path following and position control, this requires tuning of the attitude control because it is a body-fixed problem.

The proposed collision avoidance is straightforward and does not allow for applications where multiple UAVs collide, when two UAVs are in a collision course the avoidance will stop the collision, but for a straight, head on collision, it means that both UAVs will stop and hover while not being able to continue. This is not desirable for a situation where events have to be serviced.

Code optimization might be possible, such as improvements to the implementations of the A* algorithm, the Coverage calculation and the MATLAB group system.

APPENDIX A
ALGORITHM GROUP SYSTEM

The algorithm consists out of two parts, Algorithm 2 shows the part that calculates the Voronoi distribution based on a position, as is done in [10], Algorithm 3 uses the results of Algorithm 2 to calculate a new position that can be used for the next iteration, which takes in account the optimization for multiple UAVs that is proposed in Section III.

Algorithm 2 $\{\tau, I\}$ = Tessellation.Computation$\big(G, \{p_k\}, \{R_k\}, \bar{\phi}, \eta\big)$

Input: graph G (with vertex set $\mathscr{V}(G)$, edge set $\mathscr{E}(G) \subseteq \mathscr{V}(G) \times \mathscr{V}(G)$, and cost function $\mathscr{C}(G) : \mathscr{E}(G) \to \mathbb{R}^+$).
UAV locations $p_k \in \mathscr{V}(G), k = 1, 2, \cdots, N$.
UAV weight $R_k \in \mathbb{R}^+, k = 1, 2, \cdots, N$.
Discretized density function $\bar{\phi} : \mathscr{V}(G) \to \mathbb{R}$.
The metric tensor $\eta : \mathscr{C} \to \mathbb{R}^{D \times D}$.
Output: The tessellation map $\tau : \mathscr{V}(G) \to \{1, 2, \cdots, N\}$
The control integral $I \in \mathbb{R}^{D \times D}$

1: Initiate g: Set $g(v) := \infty \; \forall v \in \mathscr{V}(G)$ // Shortest distances
2: Initiate ρ: Set $\rho(v) := \infty \; \forall v \in \mathscr{V}(G)$ // Power distances
3: Initiate τ: Set $\tau(v) := -1 \; \forall v \in \mathscr{V}(G)$ // Tessellation
4: Initiate v: Set $v(v) := \emptyset \; \forall v \in \mathscr{V}(G)$ // Pointer to robot neighbor. $v : \mathscr{V}(G) \to \mathscr{V}(G)$
5: **for each** $k \in \{1, 2, \cdots, N\}$ **do**
6: Set $g(p_k) = 0$
7: Set $\rho(p_k) = -R_k^2$
8: Set $\tau(p_k) = k$
9: Set $I_k = 0$
10: **for each** $q \in \mathscr{N}_G(p_k)$ **do** // For each neighbor of p_k
11: Set $v(q) = q$
12: **end for**
13: **end for**
14: Set $Q := \mathscr{V}(G)$ // Set of un-expanded nodes
15: **while** $Q \neq \emptyset$ **do**
16: $q := argmin_{q' \in Q} \rho(q')$
17: **if** $(g(q) == \infty)$ **then**
18: **break**
19: **end if**
20: Set $Q = Q - q$ // Remove q from Q
21: Set $l := \tau(q)$
22: Set $s := v(q)$
23: **if** $s! = \emptyset$ **then** // Equivalently, $q \notin \{p_k\}_{k=1,2,\cdots,N}$
24: Set $z = \frac{P(s) - P(p_l)}{||P(s) - P(p_l)||_2}$ // Unit tangent vector
25: $M = \eta(P(p_l))$ // Metric tensor at p_l as a matrix
26: $I(q) = g(q) \times \frac{Mz}{\sqrt{z^T M z}} \times \bar{\phi}(q) \times \sqrt{detM}$
27: **end if**
28: **for each** $w \in \mathscr{N}_G(q)$ **do** // For each neighbor of q
29: Set $g' := g(q) + \mathscr{C}(G)(q, w)$
30: Set $\rho' :=$ **PowerDist**(g', R_l)
31: **if** $\rho' < \rho(w)$ **then**
32: Set $g(w) = g'$
33: Set $\rho(w) = \rho'$
34: Set $\tau(w) = l$
35: **if** $s \neq \emptyset$ **then** // Equivalently, $q \notin \{p_k\}_{k=1,2,\cdots,N}$
36: Set v(w) = s
37: **end if**
38: **end if**
39: **end for**
40: **end while**
41: **return** τ, I

The algorithm shown in Algorithm 2 is roughly the same as the algorithm that is proposed in [10], the main change is that the calculation of the position for the next iteration is removed and that instead the control integral is used as output. Algorithm 3 uses this control integral by calculating all control integrals for the UAVs of each m required UAVs to be serviced, each m required UAVs consists out of a combination of possible UAVs that can be used for this requirement. By calculating the integral for each regions we can combine the control integrals of the tessellations and receive a global control integral, which takes in account all the combinations such that the UAVs are steered in the direction that has the most effect on the global cost.

Algorithm 3 $\{p'_k\}$ = Position.Computation$\big(G, \{p_k\}, \{R_k\}, \bar{\phi}_m, \eta\big)$

Input: graph G (with vertex set $\mathscr{V}(G)$, edge set $\mathscr{E}(G) \subseteq \mathscr{V}(G) \times \mathscr{V}(G)$, and cost function $\mathscr{C}(G) : \mathscr{E}(G) \to \mathbb{R}^+$).
UAV locations $p_k \in \mathscr{V}(G), k = 1, 2, \cdots, N$.
UAV weight $R_k \in \mathbb{R}^+, k = 1, 2, \cdots, N$.
Discretized density function $\bar{\phi}_m : \mathscr{V}(G) \to \mathbb{R}\ \forall m \in \{1, 2, \cdots, M\}$.
The metric tensor $\eta : \mathscr{C} \to \mathbb{R}^{D \times D}$.

Output: The next position of each UAV, $p'_k \in \mathscr{N}_G(p_k), k = 1, 2, \cdots, N$ // p'_k is a neighbour of p_k

1: $\{\tau_1, I'\} =$ Tessellation.Computation$\big(G, K, \{R_k\}, \bar{\phi}_m, \eta\big)$ // Calculate the initial Voronoi tessellations
2: **for each** $i \in \{1, \cdots, N\}$ **do**
3: $Q_{T_i^1} =$ find$(\tau_1 == i)$ // Get the initial area per UAV
4: $I_i = sum(I'(Q_{T_i^1}))$ // Calculate the control action of UAV i
5: **end for**
6: $c(1) = N$ // Set the amount of entries in T_i^1, the maximum value for i
7: **for each** $m \in \{2, \cdots, M\}$ **do**
8: $l = 0$ // Set the initial amount of entries
9: **for each** $i \in \{1, \cdots, c(m-1)\}$ **do** // For each region of the previous $m-1$ distribution we want to calculate the sub regions to create the m distribution
10: $K = \{1, 2, \cdots, N\}$
11: $q = random(Q_{T_i^{m-1}})$ // A point in the region such that we can find the UAVs of the parent regions, random because any point is in the same parent region
12: **for each** $j \in \{1, \cdots, m-1\}$ **do**
13: $K = K \setminus \tau_{T^j}(q)$ // Get the UAV on which the parent directory j is based on and remove it from the set
14: **end for**
15: $\{\tau', I'\} =$ Tessellation.Computation$\big(G, K, \{R_k\}, \bar{\phi}_m, \eta\big)$ // Calculate the tessellation of the UAVs that could be the m-th closest
16: **for each** $k \in \{1, 2, \cdots, length(K)\}$ **do** // Loop through each UAV and calculate the sub region it takes from the parent region
17: $l = l + 1$
18: $Q_{T_l^m} =$ find$(\tau' == k) \cap Q_{T_i^{m-1}}$ // Get the new sub-region
19: $\tau_{T^m}(Q_{T_l^m}) = K_k$ // Keep a map which UAV takes which area
20: $I_{K_k} = I_{K_k} + sum(I'(Q_{T_l^m}))$ // Add to the control action of UAV K_k
21: **end for**
22: **end for**
23: $c(m) = l$
24: **end for**
25: **for each** $k \in \{1, 2, \cdots, N\}$ **do**
26: Set $p'_k := argmax_{u \in \mathscr{N}_G(p_k)} = \frac{P(u) - P(p_k)}{||P(u) - P(p_k)||_2} I_k$
27: **end for**
28: **return** p'_k

APPENDIX B
GENERALIZATION METRIC MANIFOLD

The proposed theory can also be generalized for any N-dimensional manifold that is equipped with a metric tensor. We can use the same methodology as is used in Section III and define a cost function based on a distance function $d(q, p_i)$.

$$\mathscr{J}(p, W) = \sum_{i=1}^{n} \sum_{m=1}^{M} \int_{W_i^m} f_i(d(q, p_i))\phi_m(q)\,\mathrm{d}q. \quad (17)$$

We have to redefine our Z_i^m regions to allow for a generic distance function, thus let $Z_i^m = \{q \in \Omega : \exists H \subset \{1 \ldots n\}$ such that $|H| = m-1,\ i \notin H$ and $f_h(d(q, p_h)) \leq f_i(d(q, p_i)) \leq f_\ell(d(q, p_\ell))$ for all $h \in H,\ \ell \notin H,\ \ell \neq i\}$,

By definition Z minimizes $\mathscr{J}(p, W)$ as a function of W for fixed p, so we have

$$\mathscr{J}_{exp}(p) = \mathscr{J}(p, Z) = \min_W \mathscr{J}(p, W)$$

for every $p \notin C$, where $C = \{p \in \Omega^n : \exists i, j \text{ s.t. } p_i = p_j\}$.

A. Gradient descent optimization

As in Section III-B we follow the gradient of the dynamics of the cost function to minimize it using $f_i(x) = x^2$.

$$\frac{\partial \mathscr{J}(P)}{\partial p_k} = \frac{\partial}{\partial p_k} \sum_{m=1}^{M} \sum_{i=1}^{n} \int_{Z_i^m} d(q, p_i)^2 \phi_m(q)\,\mathrm{d}q \quad (18a)$$

$$= \sum_{m=1}^{M} \sum_{i=1}^{n} \int_{Z_i^m} \frac{\partial}{\partial p_k} d(q, p_i)^2 \phi_m(q)\,\mathrm{d}q \quad (18b)$$

$$= 2 \sum_{m=1}^{M} \sum_{i=1}^{n} \int_{Z_i^m} d(q, p_i) \frac{\partial}{\partial p_k} d(q, p_i) \phi_m(q)\,\mathrm{d}q \quad (18c)$$

$$= 2 \sum_{m=1}^{M} \int_{Z_k^m} d(q, p_k) \frac{\partial}{\partial p_k} d(q, p_k) \phi_m(q)\,\mathrm{d}q \quad (18d)$$

where for step (18a) to (18b) it is allowed to take the derivative under the integral according to [4, Proposition 2] and for step (18c) to (18d) the summation is removed because $\frac{\partial}{\partial p_k} d(q, p_i) = 0,\ \forall k \neq i$.

Equation 3 then becomes

$$\dot{p}_i = -2\kappa \sum_{m=1}^{M} \int_{Z_k^m} d(q, p_k) \frac{\partial}{\partial p_k} d(q, p_k) \phi_m(q)\,\mathrm{d}q \quad (19)$$

which corresponds to steering each UAV towards a weighted "centre of mass" of its own responsibility region $\cup_m Z_i^m$. We can then take

$$\frac{\partial}{\partial p_i} d(q, p_i) = \frac{\sum_j \eta_{ij}(p_i) z_{qp_i}^j}{\sqrt{\sum_m \sum_n \eta_{mn}(p_i) z_{qp_i}^m z_{qp_i}^n}} \ [3] \quad (20)$$

where $\eta_{ij}(p_k)$ is the metric tensor at point p_k and $z_{qp_k}^i$ is the ith component of the tangent vector at p_k to the geodesic connecting q to p_k. Since the objective function

$$\frac{\mathrm{d}\mathscr{J}(p(t))}{\mathrm{d}t} = \sum_{i=1}^{n} \frac{\partial \mathscr{J}(p(t))}{\partial p_i} \dot{p}_i = -\kappa \sum_{i=1}^{n} \left(\frac{\partial \mathscr{J}(p(t))}{\partial p_i} \right)^2$$

is the same as in Section III-B we can use Proposition 1, thus the gradient dynamics converges to either the critical points of the cost or to configurations with coincident positions.

APPENDIX C
PATH FOLLOWING

A path following controller is required to ensure that the UAV is following the path, the path already avoids the obstacles so the path should be followed as good as possible.

For any simulations in this appendix we consider the behaviour of the UAV as a point mass, $F = m \times a$, with a mass of 1 kg and F_{max} of 100 N. The path that we consider is a Lissajous curve, which is easy to generate and requires some action to follow it properly, all simulations are for 10 seconds.

A. Radius

Following a path can be done in several ways, a rather straightforward one is use a radius around the UAV, the waypoint that is the farthest away inside the radius is selected and is then used as a setpoint, as is shown in Fig. 22.

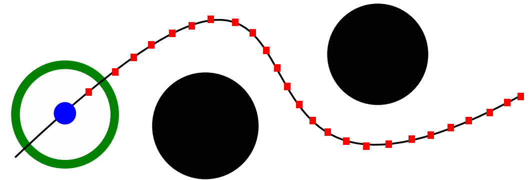

Fig. 22. Path following using a radius around the UAV and selecting the farthest away waypoint inside the circle schematically depicted.

Fig. 23 shows the result of the simulation using a Lissajous curve as path around four obstacles, a simulation is shown for different radii. It can be seen that for small radii the simulation time is not enough but the path is followed, while for large radii more distance is travelled but the mass cuts-off corners, which causes it to collide with obstacles.

B. Error radius

A different approach to the radius following is to put a minimum radius on the error, this ensures that a minimum error has to be achieved before the next setpoint can be chosen. The results of this are shown in Fig. 24.

Fig. 24 shows the result of the simulation for different radii on the error. The radius that is used is the radius that the error has to be in before it can go to the next waypoint. It can be seen that for a large radius, the mass cuts off corners as was the same with the previous method. On the other side for low radii the system slows down because it first has to get closer to the setpoint before it can go to the next one.

Based on the above simulations it can be seen that the selection of the radius depends on the allowed error and trajectory. There is not a lot of difference between both methods except for that in the first simulation the error never

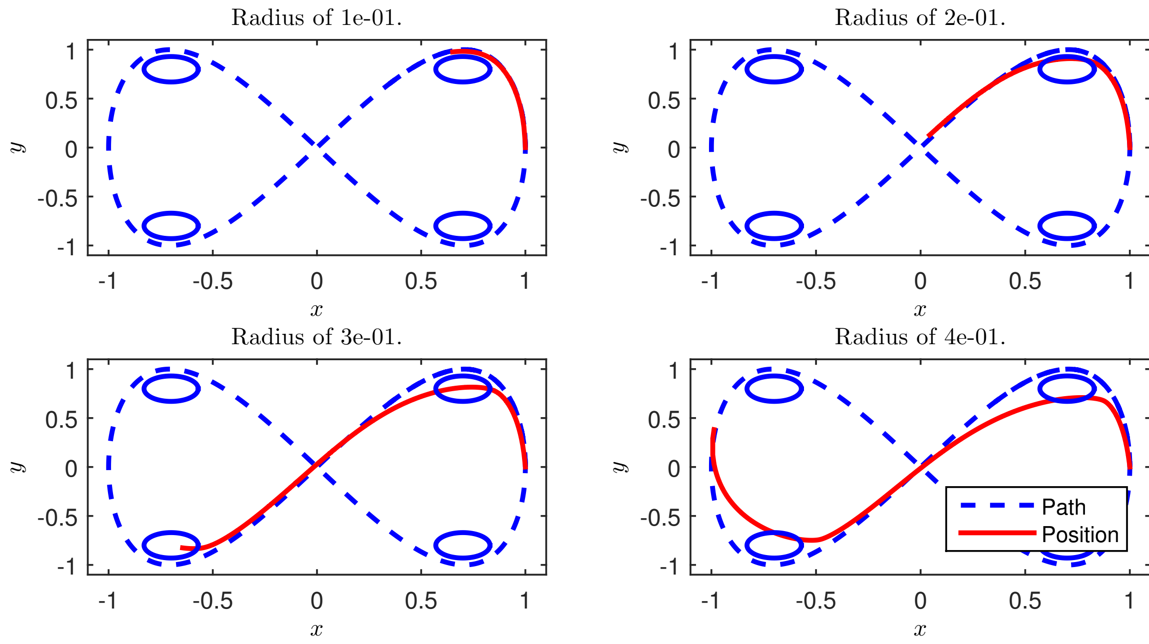

Fig. 23. Path following with obstacles, select waypoint based on furthest waypoint in the radius.

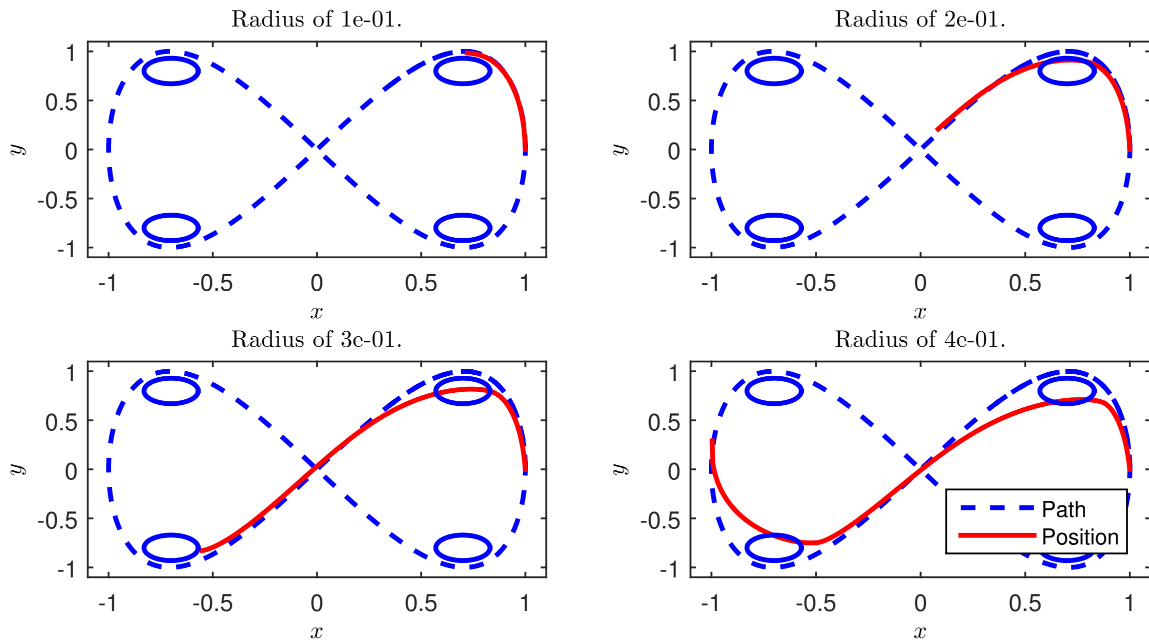

Fig. 24. Path following with obstacles, next waypoint is set when error is within the radius.

is greater than the set radius, while for the second simulation it is slightly larger in general. While for a straight trajectory the radius does not matter (except for the final point), it does matter for trajectories with curves, it is expected that for sharp curves a small radius is required to ensure that there are no corners cut-off.

The methods mentioned are usable but will require that the obstacle is not within the radius, thus the path always has to be around an obstacle at a distance at least equal to the radius. This indicated that a method has to be found which takes this in account and compensates for it, in case of the method that was used for the simulations this would mean that the radii have to be variable and change depending on the trajectory/change in direction.

While it is probably possible to come up with a method that changes the radius based on the trajectory and/or current state of the mass, a more simple method that takes in account the obstacle can be used.

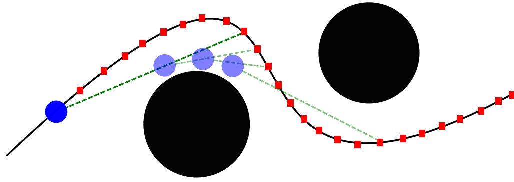

Fig. 25. Path following line of sight schematically depicted.

C. Line of sight

Fig. 25 shows an example of this method, by selecting the waypoint that is in the line of sight (LOS) of the mass, we ensure that the obstacle is not crossed, we can do this because we know the obstacles on which the path is based on.

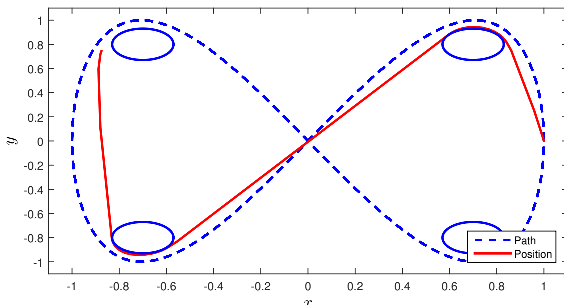

Fig. 26. Path following with obstacles using line of sight.

Fig. 26 shows result of using LOS, it is quite better than the previous two methods, half the trajectory is finished and no obstacles are crossed, this situation was not possible previously. The downside of this method is that the path is not followed, this is no problem in our case because it is shorter and faster, thus more optimal.

The problem with path following is that on a straight line we can go as fast as possible, while on a curved line we have to slow down to ensure that we do not overshoot. The previous method is a simplification of this, when a path is curved it goes around an object, thus by selecting a waypoint that is seen we automatically compensate for the curve because we have to slow down because we do not see the next point yet.

The downside of this method is that we cannot control the speed of the UAV because it fully depends on the waypoint that is in sight.

D. Curvature based path following

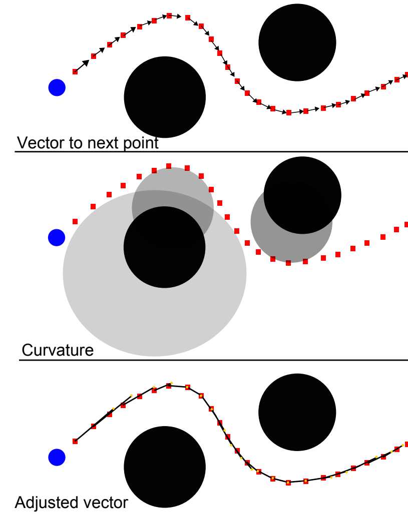

Fig. 27. Vector based path following schematically depicted.

To get a method that compensates for the curvature we opt to change the waypoints of the path based on the curvature, as shown in Fig. 27. A good way to get the curvature is to estimate a circle/sphere through 3/4 points (depending on the dimension). This circle/sphere will have a radius, which we can use to change the waypoints. We move the waypoint in the direction of the path, thus the vector pointing to the next waypoint is lengthened based on the radius of the sphere/circle that goes through the points.

For the simulation the length of the vector was increased by a factor that was based on the curvature, the curvature $\frac{1}{r}$ of the circle that was intersection the waypoint and the next and previous waypoint was mapped, resulting in a value between 0 and 1. This radius was then used to calculate the factor which lengthens the vector pointing to the next waypoint, thus our set point. We used $c = k(1 - \frac{1}{r})$, instead of using r the inverse was used, using r would result in an infinite radius when there is no curvature, this curvature $\frac{1}{r}$ is then subtracted from 1 to get a scaling where a small curve results in 0 and a straight line in 1. k is a constant which can be tweaked. The resulting factor c is then used to lengthen the vector pointing to the next waypoint.

Fig. 28 shows the result of using the curvature based path following, the result is quite better than the previous methods but requires some tweaking for it to work. in the simulation $k = 15$ was used, it can be increased even further but at some point it will end up crossing the obstacles.

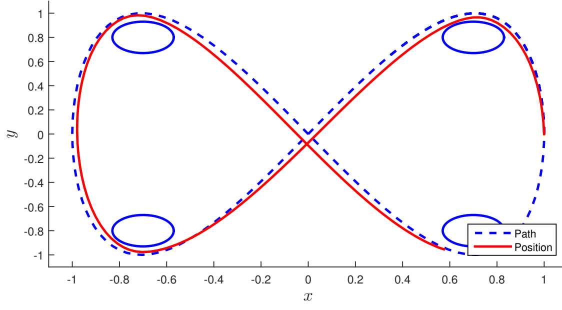

Fig. 28. Vector adjusted path following with obstacles.

Appendix D
Circumcircle Calculation

There are 3 points in space that we want to connect using one circle, also known as a circumcircle, from which we want to know the radius $r \in \mathbb{R}$,

$$\begin{aligned} P_1 &= [x_1, y_1] \in \mathbb{R}^2 \\ P_2 &= [x_2, y_2] \in \mathbb{R}^2 \\ P_3 &= [x_3, y_3] \in \mathbb{R}^2, \end{aligned}$$

these 3 points can be connected trough one circle, for this the following constriction holds,

$$|P_1 - P_0| = r^2 \tag{21}$$

$$|P_2 - P_0| = r^2 \tag{22}$$

$$|P_3 - P_0| = r^2, \tag{23}$$

where $P_0 = [x_0, y_0] \in \mathbb{R}^2$ is the centre point of the circumcircle, the set of points $P = [x, y] \in \mathbb{R}^2$ of the circumcircle can then be described by $|P - P_0| = r^2$.

A solution to these equations can be found by putting them into a matrix form $Ax = 0$, and solving the determinant of A for $det(A) = 0$, as long as the kernel of A is non-zero.

The matrix form can be found by expanding the equations

$$\begin{aligned} |P - P_0| - r^2 &= (x - x_0)^2 + (y - y_0)^2 - r^2 \\ &= x^2 - 2xx_0 + x_0^2 + y^2 - 2yy_0 + y_0^2 - r^2 \\ &= |P|^2 - 2xx_0 - 2yy_0 + |P_0|^2. \end{aligned} \tag{24}$$

Doing the same for equations (21), (22) and (23) results into

$$\begin{aligned} |P|^2 - 2xx_0 - 2yy_0 + |P_0|^2 &= 0 \\ |P_1|^2 - 2x_1x_0 - 2y_1y_0 + |P_0|^2 &= 0 \\ |P_2|^2 - 2x_2x_0 - 2y_2y_0 + |P_0|^2 &= 0 \\ |P_3|^2 - 2x_3x_0 - 2y_3y_0 + |P_0|^2 &= 0, \end{aligned}$$

which can be written in the matrix form $Ax = 0$ as

$$\begin{bmatrix} |P|^2 & -2x & -2y & 1 \\ |P_1|^2 & -2x_1 & -2y_1 & 1 \\ |P_2|^2 & -2x_2 & -2y_2 & 1 \\ |P_3|^2 & -2x_3 & -2y_3 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_0 \\ y_0 \\ |P_0|^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{25}$$

which has a non-zero kernel. By solving the $det(A) = 0$ we can gain an expression for the radius r of the circumcircle. For this we apply column operations to the matrix A to get to the following expression

$$det \begin{bmatrix} |P|^2 & x & y & 1 \\ |P_1|^2 & x_1 & y_1 & 1 \\ |P_2|^2 & x_2 & y_2 & 1 \\ |P_3|^2 & x_3 & y_3 & 1 \end{bmatrix} = 0. \tag{26}$$

Solving the determinant leads to

$$D_1|P|^2 + D_2x + D_3y + D_4 = 0 \text{ with} \tag{27}$$

$$D_1 = det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \qquad D_2 = -det \begin{bmatrix} |P_1|^2 & y_1 & 1 \\ |P_2|^2 & y_2 & 1 \\ |P_3|^2 & y_3 & 1 \end{bmatrix}$$

$$D_3 = det \begin{bmatrix} |P_1|^2 & x_1 & 1 \\ |P_2|^2 & x_2 & 1 \\ |P_3|^2 & x_3 & 1 \end{bmatrix} \qquad D_4 = -det \begin{bmatrix} |P_1|^2 & x_1 & y_1 \\ |P_2|^2 & x_2 & y_2 \\ |P_3|^2 & x_3 & y_3 \end{bmatrix}.$$

Equation (27) can then be rewritten as

$$\begin{aligned} D_1(x + \frac{D_2}{2D_1})^2 + D_1(y + \frac{D_3}{2D_1})^2 - \frac{D_2^2}{4D_1} - \frac{D_3^2}{4D_1} + D_4 &= 0 \\ D_1(x + \frac{D_2}{2D_1})^2 + D_1(y + \frac{D_3}{2D_1})^2 &= \frac{D_2^2}{4D_1} + \frac{D_3^2}{4D_1} - D_4 \\ D_1(x + \frac{D_2}{2D_1})^2 + D_1(y + \frac{D_3}{2D_1})^2 &= \frac{D_2^2 + D_3^2 - 4D_1D_4}{4D_1} \\ (x + \frac{D_2}{2D_1})^2 + (y + \frac{D_3}{2D_1})^2 &= \frac{D_2^2 + D_3^2 - 4D_1D_4}{4D_1^2}. \end{aligned} \tag{28}$$

Equation (28) corresponds to a circle of the form $(x - x_0)^2 + (y - y_0)^2 = r^2$, with the central point $x_0 = -\frac{D_2}{2D_1}$. $y_0 = \frac{D_3}{2D_1}$ and radius $r = \frac{\sqrt{D_2^2 + D_3^2 - 4D_1D_4}}{2D_1}$, which is the circumcircle of 3 points.

Appendix E
Manual

This appendix elaborates on how to use the software package that was developed based on the proposed architecture. The package is available through the Robotics and Mechatronics group of the University of Twente.

A. *Prerequisites*

The software package is written for ROS, at the date of writing the current version of ROS is Indigo, which currently only supports Saucy (13.10) and Trusty (14.04) for Debian packages. The software was written based on ROS Indigo.

The following prerequisites are assumed for this manual:

1) Ubuntu 14.04 or any derivative based on this (Xubuntu 14.04, Lubuntu 14.04, etc).
2) ROS Indigo installation according to http://wiki.ros.org/indigo/Installation/Ubuntu with a Desktop-Full Install.

3) Matlab R2013b with MATLAB I/O package.
4) Crazyflie client (https://github.com/bitcraze/crazyflie-clients-python)
5) Clean catkin workspace.
6) Knowledge on how to use Optitrack.

B. Installation

The package requires three main external packages, hector_quadrotor (http://wiki.ros.org/hector_quadrotor) for the simulation of a quadcopter, crazyflie_ros (http://wiki.ros.org/crazyflie) to connect to the crazyflie and mocap_optitrack (http://wiki.ros.org/mocap_optitrack) to read the Optitrack data in ROS. The package that is available works with the currently available versions of these packages, using the latest version of the packages might be interesting but can be incompatible, therefore the current version will be included in the package.

After unpacking the package in the catkin workspace execute the following command in the workspace to solve dependencies.

```
$ rosdep install --from-path src
      --ignore-src
```

This will install all required packages, the installation will request confirmation of installing the packages several times. The catkin workspace can now be compiled using the command.

```
$ catkin_make
```

It might be required to execute the command several times before the package compiles completely.

C. Usage

The interface can be started by executing the command

```
$ rosrun ram_crazy interface.py
```

An example on what the interface looks like is shown in Fig. 29, the toggle button *Simulation* allows for the option to start the system with or without simulation, depending on the selection the ROS publisher and subscriber topics are changed. The *Add Drone* button adds drones, the response of this is shown in Fig. 30.

The table shows several columns, the *Address* shows the radio address that is used to connect to the drone, currently it is not possible to use more than one drone per radio. The first number shown is the radio ID, the second the drone ID and the third the bandwidth. The current configuration starts with the first drone at zero and increments by 60 for each additional drone, this can be changed in the interface code. The *Prefix* column is used by ROS to distinguish each drone. The *Trackable ID* is used by Optitrack, this ID will have to be set in the Optitrack interface to send the data to ROS. The *Active* column shows if a controller is active.

If the simulation is selected Gazebo will have to be started manually, this can be done by executing the following command.

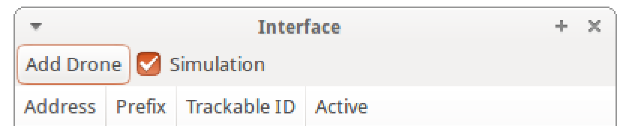

Fig. 29. Example of the interface after start up.

Interface

Add Drone | Simulation

Address	Prefix	Trackable ID	Active
radio://0/0/2M	drone0	0	No
radio://1/60/2M	drone60	60	No
radio://2/120/2M	drone120	120	No

Fig. 30. Example of the interface with 3 drones added.

```
$ roslaunch ram_crazy
      quadrotor_empty_world.launch
```

Double clicking on a row will activate the controller for that drone, a new window will appear. The interface that will appear is shown in Fig. 31. In this interface the following things can be done. The buttons on top will allow for the drone to take-off, land, or get the current setpoint of the drone. In case a simulation is used, the *Spawn Simulation* toggle will allow the deployment of the drone in the simulation, as shown in Fig. 32, when no simulation is used the *Toggle Radio* toggle can be used to enable the transmission to the drone using the radio dongle. In all cases the *Publish Setpoint* toggle will have to be enabled before any data is send. Data can be saved using the rosbag package, the *Save Data* toggle enables this and saves it to the standard *~/.ros/* folder. A Lissajous path test will be executed if the *Lissajous test* toggle is enabled.

The interface allows the ability to set a position setpoint using the sliders, the offset thrust can also be set. The PID values for the Position control and Path Following control can be set, and send to the controller by pressing the *Set gains* button. An offset is available to compensate for a static error, the *Get Current* button allows the current I-action to be read and send to the interface, this way the offset can be determined using the I-action of the PID controller.

It is not possible to change the collision radius in the interface, nor the settings for the velocity profile generation based on the curvature, these settings will have to be changed in the code directly.

Visualization is available in RViz, as shown in Fig. 33, the available topics will show in RViz when the controller is active.

The group system control can be started from the MATLAB directory of the package, executing *control.m* will show an interface, as is shown in Fig. 34. The interface has three buttons, the *Start Node* button, starts the ROS node that communicates with the ROS core, this button only needs to be activated once per MATLAB session. The *Enable System* button enables the group system, pressing on the graph will send a path towards that spot to the controller for the nearest UAV and will regroup the available UAVs. The *Save Figure* button saves the current graph.

Fig. 31. Example of the interface after start up.

Fig. 32. Example of the interface with 3 drones added.

Fig. 33. Visualization of the system using RViz, the path is shown, the orientation of the UAV and the velocity vector.

Fig. 34. MATLAB group system.

REFERENCES

[1] Sherpa introduction. [Online]. Available: "http://www.sherpa-project.eu/

[2] J. Cortes, S. Martinez, and F. Bullo, "Spatially-distributed coverage optimization and control with limited-range interactions," *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 11, no. 04, pp. 691–719, 2005.

[3] S. Bhattacharya, R. Ghrist, and V. Kumar, "Multi-robot coverage and exploration in non-Euclidean metric spaces," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 245–262.

[4] L. Pimenta, V. Kumar, R. Mesquita, and G. Pereira, "Sensing and coverage for a network of heterogeneous robots," in *47th IEEE Conference on Decision and Control*, Cancún, Mexico, Dec 2008, pp. 3947–3952.

[5] P. Surynek, "An optimization variant of multi-robot path planning is intractable." in *AAAI*, 2010.

[6] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1928–1935.

[7] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.

[8] D. M. Stipanović, P. F. Hokayem, M. W. Spong, and D. D. Šiljak, "Cooperative avoidance control for multiagent systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 129, no. 5, pp. 699–707, 2007.

[9] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*, ser. Applied Mathematics Series. Princeton University Press, 2009.

[10] S. Bhattacharya, R. Ghrist, and V. Kumar, "Relationship between gradient of distance functions and tangents to geodesics," Technical report, University of Pennsylvania, http://subhrajit.net/wiki/index.php, Tech. Rep.

[11] Bitcraze: The Crazyflie 2.0 product page. [Online]. Available: "https://www.bitcraze.io/crazyflie-2/

[12] T. Hamel, R. Mahony, R. Lozano, and J. Ostrowski, "Dynamic modelling and configuration stabilization for an x4-flyer." *a a*, vol. 1, no. 2, p. 3, 2002.

[13] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Access, IEEE*, vol. 2, pp. 56–77, 2014.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.

[15] M. Sniedovich, "Dijkstra's algorithm revisited: the dynamic programming connexion," *Control and cybernetics*, vol. 35, no. 3, p. 599, 2006.

[16] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro-UAV testbed," *IEEE Robotics Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[17] G. M. Hoffmann, S. L. Wasl, and C. J. Tomlin, "Quadrotor helicopter trajectory tracking control," in *AIAA Guidance, Navigation, and Control Conference*, 2008.

[18] W. Hanna, "Modelling and control of an unmanned aerial vehicle," Ph.D. dissertation, Charles Darwin University, 2014.

[19] 20-sim, *version 4.5*. Enschede, The Netherlands: Controllab Products B.V., 2015.

[20] ROS: An open-source robot operating system. [Online]. Available: "http://www.ros.org/

[21] Gazebo: Robot simulation made easy. [Online]. Available: "http://gazebosim.org/

[22] Matlab ROS I/O, A Downloadable MATLAB Add-On. [Online]. Available: "http://www.mathworks.com/ros

[23] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2012, p. to appear.

[24] NaturalPoint, Inc. - Optical Tracking Solutions optitrack: Motion capture systems. [Online]. Available: "https://www.naturalpoint.com/