Quadrotor Implementation of the Distributed Reactive Collision Avoidance Algorithm

Andrew Melander

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science in Aeronautics and Astronautics

University of Washington

2010

Program Authorized to Offer Degree: Aeronautics and Astronautics

University of Washington
Graduate School

This is to certify that I have examined this copy of a master's thesis by

Andrew Melander

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

____

Kristi A. Morgansen

____

Juris Vagners

Date: ______________________________

University of Washington

Abstract

Quadrotor Implementation of the Distributed Reactive Collision Avoidance Algorithm

Andrew Melander

Chair of the Supervisory Committee:
Professor Kristi A. Morgansen
Aeronautics and Astronautics

The Distributed Reactive Collision Avoidance (DRCA) algorithm was implemented on quadrotors at the Boeing Vehicle Swarm Technology Lab (VSTL). The DRCA algorithm guarantees collision avoidance for n vehicles while attempting to follow a desired control. DRCA avoids a centralized approach, scales to *O(n)* calculations on each vehicle, and explicitly accounts for maximum acceleration restrictions. The prescribed deconfliction maneuver was modified to an "all-turn-left" rules-of-the-road approach for simplicity. DRCA requires position and velocity for each vehicle involved. The Boeing VSTL uses a VICON motion-capture system to provide full-state feedback for each vehicle. This quadrotor implementation successfully demonstrated the functionality of the algorithm in two and four vehicle scenarios.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to University of Washington Department of Aeronautics and Astronautics for providing the funding for this research, Juris Vagners and Kristi Morgansen for providing guidance throughout my time here, Emmett Lalish for his previous work and assisting me with much of this project, and Christopher Lum for being extremely gracious and answering many of my questions.

DEDICATION

To my parents and my wife-to-be.

Chapter 1

INTRODUCTION

As vehicles on the ground, air, and sea increase in their level of autonomy, collision avoidance is an area of primary concern. Conflict resolution is one of the most important elements in autonomous systems, allowing multiple vehicles to perform their missions safely. The Distributed Reactive Collision Avoidance (DRCA) algorithm developed in [10] is one such conflict resolution scheme, guaranteeing collision avoidance for n nonholonomic vehicles. The DRCA algorithm is designed to work with vehicles that have limited control authority and complex dynamics (such as aircraft, which have low acceleration compared to speed and must bank to turn). One example of conflict resolution on a massive scale is Air Traffic Control (ATC), which directs air traffic all over the United States. ATC is highly centralized, and an increasing number of aircraft are directed by relatively few ATC controllers. This system has been in place for decades and has proven to be safe, but inefficient. Current air corridors account for a mere five percent of available airspace [4]. Many in the aviation industry have begun to promote a concept called “free flight”, which distributes some control authority from ATC to aircraft. This concept allows pilots to deviate from their original flight plans to utilize winds, avoid bad weather, save fuel, and decrease the time of flight [7]. Although free flight would save the airlines several billion dollars each year, it cannot be implemented until a viable conflict resolution strategy is proposed and thoroughly tested.

Many conflict resolution strategies have been proposed using varying degrees of automation. Some of these strategies are designed specifically for ATC applications, while others are more general in nature. An overview and classification of many collision avoidance algorithms can be found in [8]. According to this particular collision avoidance classification, DRCA is a nominal (projects current states into the future along a single trajectory), horizontal plane, global algorithm that combines turning and speed changes and detects conflict. It is most closely associated with a force field approach, although it does not strictly meet this definition because DRCA does not treat vehicles as charged particles.

It is imperative that any avoidance algorithm used for automating air or shipping traffic guarantees collision avoidance. For a collision avoidance guarantee to be valid for real vehicles, it must also restrict the maximum acceleration. Any collision avoidance scheme that does not meet this criteria can be ruled out for such applications. Many of the approaches previously proposed guarantee avoidance, but only for a limited number of vehicles [14], [2]. Because traffic in the air and on the sea is rapidly increasing, there is a higher likelihood of a collision (or at least a conflict) involving multiple vehicles. Thus, *n* vehicles should be accounted for. The DRCA algorithm is ideal for such applications because it guarantees collision avoidance (including a restriction on maximum acceleration) for *n* vehicles simultaneously.

Furthermore, a centralized avoidance scheme should be avoided because of the high computational load required by the central node, lack of robustness (what happens if it breaks?), and inability to respond quickly to emergency situations. The DRCA algorithm distributes computation among the entire group because each vehicle accounts only for its own interactions. This distribution makes for *O(n)* calculations on each vehicle, which should be reasonable in most applications. It is not a centralized algorithm, but it is not strictly decentralized since states of all other vehicles are required (not just the nearest neighbors). Since it is not centralized or decentralized, the term “distributed” is used in naming the algorithm.

The purpose of this project was to proceed beyond the theoretical and simulation work developed in [10] by implementing the DRCA algorithm on actual hardware. The DRCA algorithm was implemented as a planar method, although extensions in three dimensions have been developed, proven, and demonstrated in simulation [9]. The tests were conducted on autonomous quadrotors at the Boeing Vehicle Swarm Technology Lab (VSTL). Quadrotors are small, agile vehicles with four rotors. Figure 1 shows a picture of the quadrotor vehicle.

Literature also includes numerous examples of implementing collision avoidance on quadrotors. Much of this work is focused strictly on obstacle avoidance (static objects) rather than multi-vehicle collision avoidance, as in [12], [5],[1]. One work of interest is [6], which also describes a decentralized, computationally efficient algorithm that guarantees collision avoidance for n vehicles. This approach uses a switching control law in which vehicles compute avoid sets with respect to other vehicles. Although the approach is much different than the DRCA, it presents an interesting example for implementing collision avoidance in quadrotors.

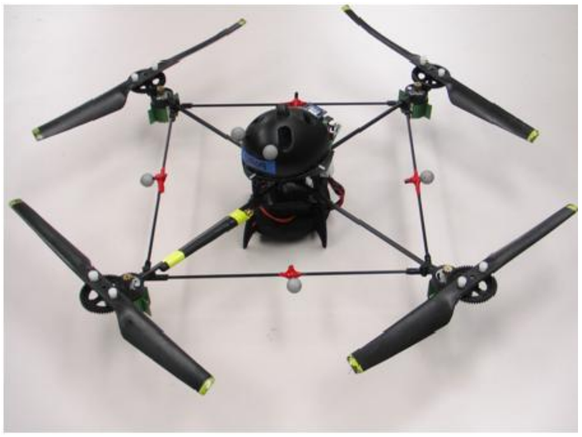


Figure 1.1: Picture of autonomous quadrotor used at Boeing VSTL.

This thesis is organized as follows. The theoretical work developed by Lalish in [10],[11],[9] is introduced in Chapter 2. It describes the model and the DRCA algorithm. The Boeing VSTL testbed is described in Chapter 3. Chapter 4 details the Boeing-developed simulation environment and gives simulation results. The initial flight test results are included in Chapter 5, along with a discussion of discrepancies between simulation and experiment. Results for the final flight test are given in Chapter 6, which returns expected results between simulation and experiment. Finally, concluding remarks and possible future work are presented in Chapter 7.

Chapter 2

THEORETICAL WORK

The development of the DRCA algorithm is presented in [10]. The authors discuss the algorithm as it applies to n unicycle-type vehicles. Most of this theoretical work translates directly to the quadrotor, which is itself a unicycle-type vehicle (at least in a two-dimensional sense). The theoretical work below will be presented as a two-dimensional analysis, since the work was implemented on the quadrotors in two dimensions. Each vehicle has a desired control input, $\mathbf{u}_d(t) \in \mathbb{R}^3$, which is an acceleration vector given by some outer-loop controller. The DRCA algorithm is mission independent, meaning the controller could make the vehicle do any number of desired tasks such as waypoint navigation, target tracking, or area searching. The desired control input is passed to the DRCA algorithm, which modifies the control to guarantee collision avoidance.

2.1 Quadrotor Model

Position and velocity are the only states of concern to the DRCA algorithm; therefore, these states must be observable for each vehicle at all times. A quadrotor's orientation is not needed to guarantee collision avoidance. The notation throughout this paper will use bold face for vectors, script capital letters for sets, standard capital letters for matrices and functions, and everything else is assumed scalar. Vector quantities subscripted with t and n are the components in the tangent and normal direction, respectively. The vehicle model is a double integrator model that is concerned only with position and velocity:

$$\frac{d}{dt}\begin{bmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{bmatrix} = \begin{bmatrix} \mathbf{v}_i \\ \mathbf{u}_i \end{bmatrix} \tag{2.1}$$

In [10], examples for constraint sets that work well for quadrotor applications are set out. These examples are used below. The input is constrained by using an arbitrarily varying constraint set, $\mathbf{u}_i \in \mathcal{C}_i$. The only requirement is that the set contains the origin, so the vehicle is allowed to have a control input of zero. Quadrotors are limited by maximum acceleration and velocity, so an appropriate constraint set is

$$\mathcal{C}_i = \left\{ \mathbf{u}_i \in \mathbb{R}^3 \middle| \, \|\mathbf{u}_i\| \leq u_{max}, \, \|\mathbf{v}_i\| \geq v_{max} \implies \mathbf{u}_i^{\mathsf{T}} \mathbf{v}_i \leq 0 \right\}. \tag{2.2}$$

For the quadrotors, the maximum acceleration is limited by their maximum bank angle (30 degrees). This angle corresponds to a maximum acceleration of $u_{max} = 5.6638 m/s^2$.

The DRCA algorithm requires that constraints be given in terms of a rectangular box description that encloses $\mathcal{C}$. For the quadrotors, an appropriate $\mathcal{R}$ is

$$\mathcal{R}_i = \left\{ \mathbf{u}_i \in \mathbb{R}^3 \middle| - u_{max_i} \leq u_{t_i} \leq u_{max_i}, \dots \right\} \tag{2.3}$$

with similar restrictions in the normal direction.

Finally, a continuous saturation function is needed, $S : \mathcal{R} \to \mathcal{C}$. This saturation function must also become the identity map for any $\mathbf{u} \in \mathcal{C}$, and must preserve the sign of each component of $\mathbf{u}$ when decomposed in the $\mathbf{t}$ and $\mathbf{n}$ directions. The saturation function for the quadrotors is

$$S_i = \begin{cases} \mathbf{u}_i \frac{u_{max}}{\|\mathbf{u}_i\|}, & \|\mathbf{u}_i\| > u_{max} \\ \mathbf{u}_i - \frac{\mathbf{v}_i \mathbf{u}_i^{\mathsf{T}} \mathbf{v}_i}{v_{max}}, & \|\mathbf{v}_i\| \geq v_{max}, \mathbf{u}_i^{\mathsf{T}} \mathbf{v}_i \geq 0 \\ \mathbf{u}_i, & \text{otherwise.} \end{cases} \tag{2.4}$$

2.2 Collision Cones

The DRCA algorithm uses the collision cone concept, first developed in [3], extensively. Thus, a good understanding of collision cones is necessary to fully understand the DRCA algorithm. A collision cone is defined for each vehicle in relation with each of the others. The relative position vector from vehicle i to vehicle j is denoted $\tilde{\mathbf{r}}_{ij} \equiv \mathbf{r}_j - \mathbf{r}_i$, while the relative velocity vector is defined in the opposite sense: $\tilde{\mathbf{v}}_{ij} \equiv \mathbf{v}_i - \mathbf{v}_j$. Note that these definitions imply that $\dot{\tilde{\mathbf{r}}}_{ij} = -\tilde{\mathbf{v}}_{ij}$, and $\dot{\tilde{\mathbf{v}}}_{ij} = \mathbf{u}_i - \mathbf{u}_j$. Figure 2.1 gives a visual depiction of the collision cone concept.

Each collision cone is defined by the relative position between vehicles i and j, $\tilde{\mathbf{r}}_{ij}$ (the axis of the cone), and the minimum allowed separation distance between vehicles i and j, d_{sep} (the edges of the cone). Careful attention should be paid to selecting the minimum separation distance, since it

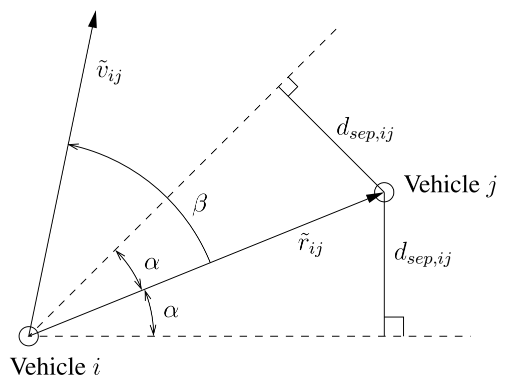


Figure 2.1: A visual depiction of the collision cone concept.

is defined from the centers of the vehicles rather than the edges of the vehicles. For the quadrotors, this minimum allowed separation distance is $d_{sep} = 1$ m. The following definitions are provided in [11].

Two concepts fundamental to collision avoidance are collision and conflict. These terms need to be strictly defined to avoid confusion.

Definition 1 (Collision) *A collision occurs between two vehicles when*

$$\|\mathbf{r}\| < d_{sep}, \tag{2.5}$$

where d_{sep} is the minimum allowed separation distance between the vehicles' centers. This distance can be different for each pair of vehicles in order to account for heterogeneity in the system.

If two vehicles are not in a collision, it is important to ascertain if they will be at some future point in time if they continue with their current trajectories. This first–order prediction of a collision will be called a conflict.

Definition 2 (Conflict) *A conflict occurs between two vehicles (i and j) if they are not currently in a collision, but with null control inputs (i.e. constant velocity) they will at some future point in time enter a collision:*

$$\min_{t>0} \|\mathbf{r}(t)\| < d_{sep}. \tag{2.6}$$

Lemma 1 *A necessary and sufficient condition for there to be no conflict is $|\beta| \geq \alpha$, where $\beta = \angle \mathbf{v} - \angle \mathbf{r}$ and $\alpha = \arcsin\left(\frac{d_{sep}}{\|\mathbf{r}\|}\right)$.*

This lemma states that there is no conflict if the tip of the relative velocity vector is not in the collision cone. The following proof for this lemma is given in [11].

Proof 1 *First, define $\tilde{\mathbf{r}}_{min}$ as the position vector corresponding to the closest approach of one vehicle to another in (2.6). By definition, at $\tilde{\mathbf{r}}_{min}$ the derivative of $\|\tilde{\mathbf{r}}\| = 0$. Therefore:*

$$\frac{d}{dt}\|\tilde{\mathbf{r}}\| = -\frac{\tilde{\mathbf{v}}^{\mathsf{T}}\tilde{\mathbf{r}}}{\sqrt{\tilde{\mathbf{r}}^{\mathsf{T}}\tilde{\mathbf{r}}}} = 0$$

$$\implies \tilde{\mathbf{v}}^{\mathsf{T}}\tilde{\mathbf{r}}_{min} = 0 \tag{2.7}$$

Next, note that for constant velocity, $\tilde{\mathbf{v}}$:

$$\tilde{\mathbf{r}}_{min} = \tilde{\mathbf{r}}_0 - \tilde{\mathbf{v}}T \tag{2.8}$$

where T is the time to closest approach. To find T, multiply by $\tilde{\mathbf{v}}^{\mathsf{T}}$ on both sides of (2.8), apply (2.7) and solve:

$$\tilde{\mathbf{v}}^{\mathsf{T}}\tilde{\mathbf{r}}_{min} = \tilde{\mathbf{v}}^{\mathsf{T}}\tilde{\mathbf{r}}_0 - \tilde{\mathbf{v}}^{\mathsf{T}}\tilde{\mathbf{v}}T$$

$$0 = \|\tilde{\mathbf{v}}\|\,\|\tilde{\mathbf{r}}_0\|\cos\beta - \|\tilde{\mathbf{v}}\|^2 T$$

$$T = \frac{\|\tilde{\mathbf{r}}_0\|}{\|\tilde{\mathbf{v}}\|}\cos\beta \tag{2.9}$$

Now using (2.7), (2.8), and (2.9), a concise expression for the closes approach distance is given by

$$\begin{aligned}
d_{min} &= \sqrt{\tilde{\mathbf{r}}_{min}^{\mathsf{T}}\tilde{\mathbf{r}}_{min}} \\
&= \sqrt{\left(\tilde{\mathbf{r}}_0 - \tilde{\mathbf{v}}T\right)^{\mathsf{T}}\tilde{\mathbf{r}}_{min}} \\
&= \sqrt{\tilde{\mathbf{r}}_0^{\mathsf{T}}\left(\tilde{\mathbf{r}}_0 - \tilde{\mathbf{v}}T\right)} \\
&= \sqrt{\|\tilde{\mathbf{r}}_0\|^2 - \|\tilde{\mathbf{v}}\|\,\|\tilde{\mathbf{r}}_0\|\cos\beta\left(\frac{\|\tilde{\mathbf{r}}_0\|}{\|\tilde{\mathbf{v}}\|}\cos\beta\right)} \\
&= \sqrt{\|\tilde{\mathbf{r}}_0\|^2\left(1-\cos^2\beta\right)} \\
&= \|\tilde{\mathbf{r}}_0\|\,|\sin\beta|\,.
\end{aligned}$$

For no conflict to occur, the converse of (2.6) must be true:

$$\begin{aligned}
d_{sep} &\geq d_{min} \\
&= \|\tilde{\mathbf{r}}_0\|\,|\sin\beta|\,. \qquad (2.10)
\end{aligned}$$

Therefore, to remain free of conflict it is necessary that:

$$|\beta| \geq \arcsin\left(\frac{d_{sep}}{\|\tilde{\mathbf{r}}_0\|}\right) = \alpha. \qquad (2.11)$$

For sufficiency, if $|\beta| \geq \alpha$, then (2.11) and (2.10) still hold, thus implying that no conflict exists.

Because DRCA is reactive one can, without loss of generality, denote $\tilde{\mathbf{r}}_0$ as $\tilde{\mathbf{r}}$.

2.3 DRCA Algorithm

The DRCA algorithm consists of a deconfliction maneuver and a deconfliction maintenance phase that operate according to Figure 2.2. If the vehicles are separated by a predetermined bound, γ, the desired controls will pass through unaffected. Otherwise, the DRCA algorithm determines if vehicles are in conflict and modifies the desired control accordingly. The deconfliction maneuver is designed to get the vehicles out of conflict and must guarantee safety for the vehicles, given certain bounds on the initial conditions.

The DRCA algorithm works by checking the state of each vehicle against every other vehicle. It first checks to see if vehicles are close enough to each other to be of concern, referred to as being

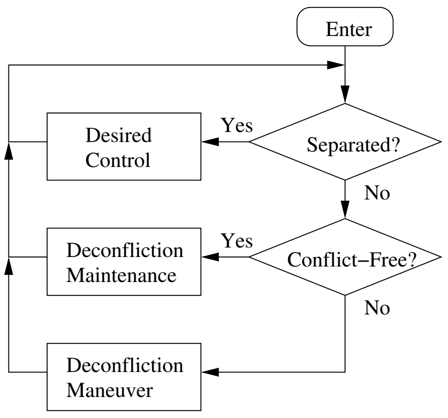


Figure 2.2: DRCA algorithm flow.

“in bound”. If vehicles are in bound, the algorithm determines if a vehicle is in conflict using the collision cone concept [3]. If the vehicle is in bound and in conflict, a deconfliction maneuver is initiated to get the vehicles out of conflict. Figure 2.2 shows a flow chart of the algorithm.

A deconfliction maneuver must involve some sort of acceleration change, which means either changing speed or turning. Since quadrotors are limited by a maximum speed and can accelerate faster laterally, a turning approach is preferable. One simple way to implement a turning approach is to have all vehicles in the conflict turn a specified direction (either all left or all right) at maximum rate until a conflict-free state is reached–similar to a rules-of-the-road approach. For the quadrotor application of the DRCA algorithm, an all-turn-left deconfliction maneuver was chosen. As the name implies, vehicles in conflict turn left at maximum rate to get out of conflict. This basic maneuver serves to demonstrate the algorithm on the quadrotors, although more sophisticated and efficient deconfliction maneuvers have been developed and proven in [9]. The all-turn-left deconfliction maneuver is defined by

$$\mathbf{u}_i = L\mathbf{v}_i \tag{2.12}$$

$$L = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}.$$

In [11] it is proven that even when the system cannot reach a conflict-free state, the all-turn-left maneuver simply becomes a loiter pattern. Thus, this deconfliction maneuver does guarantee safety.

Theorem 1 *If the initial separation of vehicles in an n-vehicle system satisfies*

$$\|\tilde{\mathbf{r}}\| \geq 2\frac{s_i}{u_{n_{i,max}}} + 2\frac{s_j}{u_{n_{j,max}}} + d_{sep_{ij}}, \tag{2.13}$$

where s is the vehicle's speed, then all vehicles will remain collision free for all time if each vehicle constantly turns at its maximum rate while maintaining constant speed.

Proof 2 *The trajectory each vehicle follows using constant turning is a circle of radius $\frac{s_i}{u_{n_{i,max}}}$. As long as a pair of vehicles is separated by at least the sum of the diameters of their loiter patters and the minimum separation distance, then they can never collide.*

The purpose of the deconfliction maintenance process is to keep the vehicles free of conflict. The desired control input is used unless such a control will bring the vehicles back into conflict. To make this determination and transition smoothly, it is first necessary to know how close a vehicle's velocity vector is to causing a conflict. Figure 2.3 shows the collision cone augmented with new vectors that will be defined below. These new vectors are used to construct the deconfliction maintenance algorithm. The entire process is described in detail in [11].

Start by defining a unit vector $\mathbf{c}$ on the side of a collision cone closest to conflict:

$$\mathbf{c} = R\left(\operatorname{sgn}\beta\alpha\right)\frac{\tilde{\mathbf{r}}}{\|\tilde{\mathbf{r}}\|},$$

where $\operatorname{sgn}\gamma$ is the signum function, which takes on the value of 1 when $\gamma \geq 0$ and -1 for $\gamma < 0$. $R\left(\gamma\right)$ is a rotation matrix defined as

$$R\left(\gamma\right) = \begin{bmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{bmatrix}.$$

A vector, $\mathbf{e}$, is then defined from the near side of the collision cone to the relative velocity vector, $\tilde{\mathbf{v}}$. If $\mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} > 0$, the vehicles are heading toward each other, and $\mathbf{e}$ can be defined from the following two geometric relations:

$$\mathbf{e} + k\mathbf{c} = \tilde{\mathbf{v}}$$
$$\mathbf{c}^{\mathsf{T}}\mathbf{e} = 0,$$

where k is an appropriate scalar. These equations can be rewritten in matrix notation:

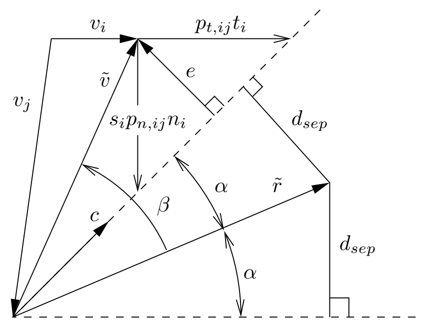


Figure 2.3: An augmented depiction of the collision cone used in DRCA.

$$\begin{bmatrix} I & \mathbf{c} \\ \mathbf{c}^{\mathsf{T}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{e} \\ k \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{v}} \\ 0 \end{bmatrix}. \tag{2.14}$$

If instead, $\mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} \leq 0$ (the vehicles are headed away from each other), no normal exists. The closest point on the collision point is the apex, which implies $\mathbf{e} = \tilde{\mathbf{v}}$. Using (2.14), this new $\mathbf{e}$ becomes

$$\mathbf{e} = \begin{cases} \tilde{\mathbf{v}}, & \mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} \leq 0 \\ R\left(\frac{\pi}{2}\right)\mathbf{c}\mathbf{c}^{\mathsf{T}}R^{\mathsf{T}}\left(\frac{\pi}{2}\right)\tilde{\mathbf{v}}, & \mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} > 0, \end{cases}$$

where $R\left(\frac{\pi}{2}\right) c$ is a vector orthogonal to $\mathbf{c}$.

Thus, the vector $\mathbf{e}$ shows how close a vehicle's velocity vector is to conflict. Next, the control inputs needed to produce said conflict must be found. The control inputs can be decoupled into tangential and normal directions to simplify this process. To reemphasize, these directions are with respect to the body frame (which also happens to be fixed to the inertial coordinate system because there is no heading change, and the vehicle's orientation does not matter to the algorithm). The tangential control, $\mathbf{u}_t$, is a change in speed, and the normal control, $\mathbf{u}_n$ is a change in direction. The signed distance to the collision cone in the $\mathbf{t}_i$ direction gives the speed change needed to produce a conflict. Similarly, a signed distance to the collision cone in the $\mathbf{n}_i$ direction gives the directional change needed to produce a conflict. These signed distances are defined as follows:

$$p_{t,ij} = \frac{\|\mathbf{e}_{ij}\|^2}{\mathbf{e}_{ij}^{\mathsf{T}}\mathbf{t}_i} \tag{2.15}$$

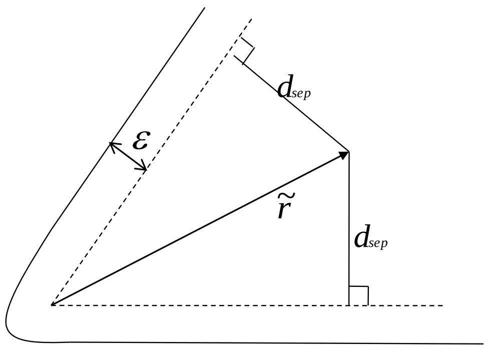


Figure 2.4: Visual depiction of ϵ, the length of a spring-like buffer in velocity space.

$$p_{n,ij} = \frac{\|\mathbf{e}_{ij}\|^2}{\mathbf{e}_{ij}^{\mathsf{T}}\mathbf{n}_i}. \tag{2.16}$$

Note that when $\mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} < 0$, these scalars measure the distance to the apex of the collision cone. In this circumstance, there is a possibility that both components may never intersect the cone. To circumvent this problem, the signed distance when $\mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} < 0$ is from $\tilde{\mathbf{v}}$ to the plane defined by the normal vector $\mathbf{e}$ (defined as $\tilde{\mathbf{v}}$ for this case). This scenario is actually encompassed by (2.15) and (2.16), but it is explained geometrically in Figure 2.3.

Now define a threshold, $\epsilon > 0$, that dictates when a conflict is far enough away to be ignored for the purposes of deconfliction maintenance. This parameter can also be decoupled into $\epsilon_t, \epsilon_n > 0$ such that a conflict can be ignored in the tangential direction if $|p_t| > \epsilon_t$ and in the normal direction if $|p_n| > \epsilon_n$. Another way to understand ϵ is as the length of a spring-like buffer (in velocity space) that surrounds the collision cone (Figure 2.4). If a vehicle tries to push against this buffer towards a conflict, the buffer will push back. The faster a vehicle is accelerating toward the collision cone, the harder the buffer will push back.

The DRCA algorithm runs the deconfliction maintenance controller on each vehicle separately and computes a p_t and p_n for every other vehicle within the predetermined separation bound, γ. It then must find the closest conflict in each direction:

$$\begin{aligned} p_{t_i}^{+} &= \min_j \{p_{t,ij} > 0, \epsilon_{t_i}\} \\ p_{t_i}^{-} &= -\max_j \{p_{t,ij} < 0, -\epsilon_{t_i}\}, \end{aligned} \tag{2.17}$$

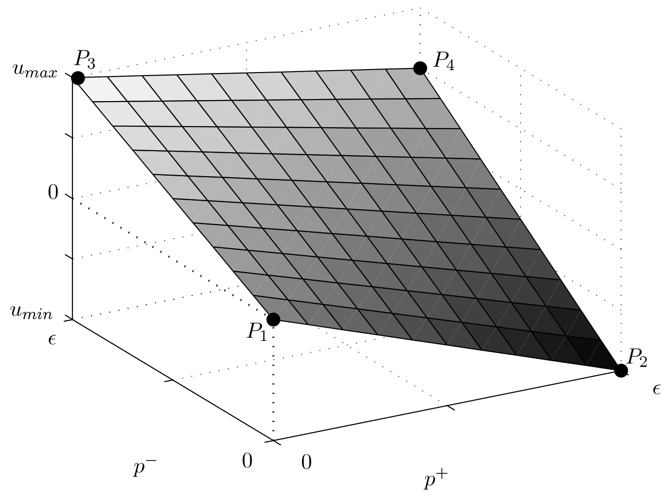


Figure 2.5: Example of the control function, F. Note that P_4 moves up and down with changing u_d.

and likewise for p_n. Also, (2.17) takes into account that p can be positive or negative. Note that by definition $0 < p^{\pm} \leq \epsilon$. It is defined this way because deconfliction maintenance will not occur outside of the ϵ threshold.

In [10] a control function, F, is chosen to modify the desired control input such that in each direction $\mathbf{u} = F\left(p^+, p^-\right)$. It is defined as follows:

$$F(p^+, p^-) = \frac{u_{min}}{\epsilon}p^+ + \frac{u_{max}}{\epsilon}p^- + \frac{u_d - u_{max} - u_{min}}{\epsilon^2}p^+p^-. \tag{2.18}$$

Note that when $p^+ = p^- = \epsilon$, $F\left(p^+, p^-\right) = \mathbf{u}_d$. This control function is chosen because it is a bilinear interpolation of the following ordered triples of the form (p^+, p^-, u):

$$P_1 = (0, 0, 0) \qquad P_2 = (\epsilon, 0, u_{min})$$
$$P_3 = (0, \epsilon, u_{max}) \qquad P_4 = (\epsilon, \epsilon, u_d).$$

Figure 2.5 illustrates an example of this control function. Of course, the desired control must be saturated so that

$$\mathbf{u}_{min} \leq \mathbf{u}_d \leq \mathbf{u}_{max} \tag{2.19}$$

in order for F to function properly. The saturation function $\mathcal{S}$ must be used to map $\mathcal{R}$ to $\mathcal{C}$ (see (2.2) - (2.3)).

The selection of ϵ is important because it determines when deconfliction maintenance is run. An intuitive way to choose ϵ is to treat it like a gain parameter,

$$k = \frac{u_{max} - u_{min}}{\epsilon}.$$

If ϵ is akin to the length of the spring-like buffer zone around the collision cone, k is like the spring constant. Note it has units of inverse seconds.

The proof that this avoidance scheme will indeed work for n vehicles rests on the following theorem from [10].

Theorem 2 *The deconfliction maintenance controller described above, when implemented on n vehicles with dynamics (2.1) and inputs constrained by $\mathcal{C}_i$, will keep the system collision free for all time if the system starts conflict-free.*

Proof 3 *To measure the distance to a collision, define m as a signed version of $\|\mathbf{e}\|$ (in terms of $\tilde{\mathbf{v}}$ from the geometry in Figure 2.3):*

$$m = \begin{cases} \|\tilde{\mathbf{v}}\|, & \mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} \le 0 \\ \|\tilde{\mathbf{v}}\| \sin(|\beta| - \alpha), & \mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} > 0. \end{cases} \tag{2.20}$$

Note that m is negative during conflict and positive outside of conflict. Vehicles will remain conflict-free if they stop approaching collision cones as they get closer to collision. Thus,

$$\lim_{m \to 0^+} \dot{m} \ge 0, \tag{2.21}$$

for every pair of vehicles. This ensures that $m > 0$ for all time

$$\dot{m} = \frac{\mathbf{e}^{\mathsf{T}}\dot{\tilde{\mathbf{v}}}}{m} = \hat{\mathbf{e}}^{\mathsf{T}}\dot{\tilde{\mathbf{v}}}, \tag{2.22}$$

where $\hat{\mathbf{e}} = \frac{\mathbf{e}}{\|\mathbf{e}\|}$.

For $\mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} > 0$, the derivative of (2.20) becomes

$$\dot{m} = \sin(|\beta| - \alpha)\frac{d\,\|\tilde{\mathbf{v}}\|}{dt} + \|\tilde{\mathbf{v}}\| \cos(|\beta| - \alpha)\frac{d}{dt}(|\beta| - \alpha). \tag{2.23}$$

The derivative exists because $m > 0$ implies $\|\tilde{\mathbf{v}}\| \neq 0$, $|\beta| \ge \alpha > 0$, and $\|\tilde{\mathbf{r}}\| \ge d_{sep} > 0$. Taking these derivatives and simplifying (2.23) yields

$$\dot{m} = \hat{\mathbf{e}}^{\mathsf{T}}\dot{\tilde{\mathbf{v}}} + \cos(|\beta| - \alpha)\frac{\|\tilde{\mathbf{v}}\|^2}{\|\tilde{\mathbf{r}}\|}(|\sin\beta| - \cos\beta\tan\alpha).$$

Since

$$\frac{m}{\|\tilde{\mathbf{v}}\|\cos\alpha} = \frac{\sin|\beta|\cos\alpha - \sin\alpha\cos|\beta|}{\cos\alpha}$$
$$= |\sin\beta| - \cos\beta\tan\alpha,$$

the expression simplifies to

$$\dot{m} = \hat{\mathbf{e}}^{\mathsf{T}}\dot{\tilde{\mathbf{v}}} + m\frac{\|\tilde{\mathbf{v}}\|\cos(|\beta|-\alpha)}{\|\tilde{\mathbf{r}}\|\cos\alpha}. \tag{2.24}$$

The second term is always positive because $\mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}} > 0$ *implies that* $\cos(|\beta|-\alpha) > 0$*, and* $\alpha \leq \pi/2$ *by definition. Combining this result with (2.22) implies that*

$$\dot{m} \geq \hat{\mathbf{e}}^{\mathsf{T}}\dot{\tilde{\mathbf{v}}} \tag{2.25}$$

for any value of $\mathbf{c}^{\mathsf{T}}\tilde{\mathbf{v}}$*.*

Expanding $\mathbf{u}_i$ *into its components yields* $\mathbf{u}_i = [u_{t_i}\mathbf{t}_i + u_{n_i}\mathbf{n}_i]$*. Following the logic in [11], (2.25) can be rewritten as (using the* ij *notation again briefly for clarity)*

$$\dot{m}_{ij} \geq m_{ij}\left(\frac{u_{t_i}}{p_{t,ij}} + \frac{u_{n_i}}{p_{n,ij}} + \frac{u_{t_j}}{p_{t,ji}} + \frac{u_{n_j}}{p_{n,ji}}\right). \tag{2.26}$$

Thus, as long as the controller ensures that u_{t_i} *has the same sign as* $p_{t,ij}$*, etc. then* $\dot{m} \geq 0$ *for that pair of vehicles. Because each vehicle calculates its control from its own point of view, all vehicles will necessarily cooperate in avoiding conflicts.*

Combining this result with the definitions (2.17), any continuous control function that satisfies

$$\begin{aligned} &\lim_{p^+_{t_i}\to 0^+} u_{t_i} \geq 0, \quad \lim_{p^+_{n_i}\to 0^+} u_{n_i} \geq 0, \quad \lim_{p^+_{b_i}\to 0^+} u_{b_i} \geq 0, \\ &\lim_{p^-_{t_i}\to 0^+} u_{t_i} \leq 0, \quad \lim_{p^-_{n_i}\to 0^+} u_{n_i} \leq 0, \quad \lim_{p^-_{b_i}\to 0^+} u_{b_i} \leq 0, \end{aligned} \tag{2.27}$$

also ensures that

$$\lim_{m_{ij}\to 0^+} \dot{m}_{ij} \geq 0,$$

guaranteeing the system cannot enter a conflicted state.

The control function used in this implementation (2.18) satisfies (2.27), so the deconfliction maintenance controller will cause the n*-vehicle system to remain conflict-free for all time, assuming it started that way.*

Note the results will still hold after saturating $\mathbf{u}$ with $\mathcal{S}$ as long as $\mathcal{S}$ preserves the quadrant of $\mathbf{u}$ so that (2.27) is still satisfied.

Chapter 3
TESTBED

The Boeing Vehicle Swarm Technology Lab (VSTL) was developed to provide low-cost alternatives for the rapid prototyping of mission algorithms, vehicle hardware, and health management [13]. The indoor testbed, shown in Figure 3, is a 100'x50'x20' volume that allows for testing a heterogeneous mixture of autonomous air and ground vehicles. The testbed has full-state feedback capability through a VICON high-accuracy, low-latency, vision-based, motion capture position reference system. The VICON system pulses visible light that bounces off reflective markers attached to the vehicles, using multiple high-resolution digital cameras to triangulate the vehicles' position and attitude. The markers are arranged in unique patterns to distinguish vehicles. Position accuracy is sub-millimeter and angular accuracy is sub-degree. Data for multiple vehicles is provided to the central data processing hub at 100 frames per second with approximately 10 milliseconds of latency [13].

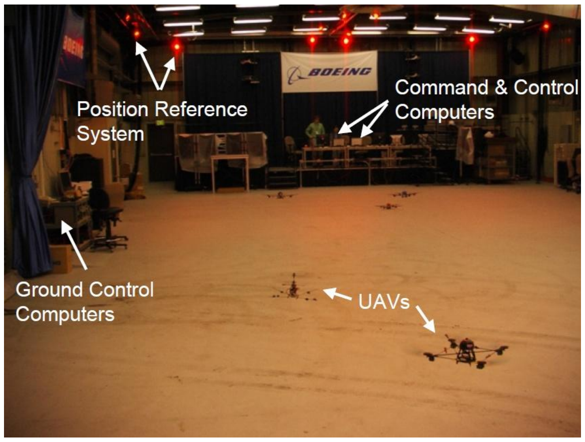


Figure 3.1: Boeing Vehicle Swarm Technology Lab (VSTL).

The vehicles used for testing the DRCA algorithm were modified Draganflyer quadrotors (VTOL vehicles lifted and propelled by four rotors). Each quadrotor is designed primarily for autonomous control and is equipped with its own on-board controller, health-monitoring, and communication sensor payload. The health-monitoring system assists with vehicle safety, as the vehicle will land and deactivate if something is wrong. The quadrotors can be controlled simultaneously in a Boeing-developed operator interface known as SwarmView. Other testbed software includes a vehicle position re-formatting and broadcasting application and a common ground-based vehicle control application. Two data buses allow software elements to commmunicate via User Datagram Protocol (UDP) Ethernet packets. One is used for transmitting vehicle position and attitude data, and the other for transmitting vehicle health/capability data and vehicle commands [13].

Although this testbed is adequate for testing the DRCA algorithm, it is not ideal. The DRCA algorithm was designed using “velocity space.” Since rotorcraft are continually changing velocity while hovering, the quadrotors are constantly going in and out of conflict even if they are commanded to be stationary a safe distance apart. Additionally, the algorithm is designed with the hope that there will be space to move that is free of collision cones. This concept guarantees collision avoidance in open airspace, but presents problems in enclosed environments. In certain scenarios, quadrotors may be presented with the choice of a collision or of going out of bounds, so that collision avoidance can no longer be guaranteed. Nevertheless, demonstrating that the DRCA algorithm works in this environment is a testament to its robustness.

Chapter 4
SIMULATION

The DRCA algorithm was first implemented in [9] using Matlab–a numerical computing environment and fourth generation programming language standard in the engineering industry. Matlab is an ideal programming language for this application because it allows matrix manipulation; however, a lower level, general-purpose programming language is needed to implement the DRCA algorithm beyond simulation. The algorithm was rewritten in C++ with functions that emulate the functions developed in Matlab, serving to simplify coding and allowing greater continuity. This process requires writing code in C++ that emulates Matlab commands. Boeing provided a matrix class to perform matrix operations, and other classes were coded to emulate various Matlab commands. These classes are used to port the DRCA algorithm into C++. The C++ code was then debugged function by function, ensuring that for a common input, outputs are identical in Matlab and C++.

This code was then ported into the Boeing-provided programming construct, which gives position and velocity for each vehicle (along with other information that is extraneous for this application). The construct does not, however, give a desired control, which is a vitally necessary component of the DRCA algorithm. The entire purpose of the algorithm is to modify the desired control as necessary to avoid collisions. Additionally, since the Boeing VSTL mostly tests potential-based collision avoidance algorithms, the construct uses additive velocity, in which the velocity output from the collision avoidance algorithm is added to the vehicle's current velocity. An illustration of this construct is shown in Figure 4.1.

The Boeing VSTL decided that they would not change the programming architecture to output a desired control; however, they did slightly modify it to reveal the formerly hidden additive velocity so that its effects could be negated (by subtracting off the additive velocity). To get the DRCA algorithm functioning within this framework, the vehicles' desired velocity was multiplied by a gain called dt to get, in essence, a desired control. Note that dt is not meant to describe an increment of time. The control output from the algorithm is then converted back to a velocity–the expected

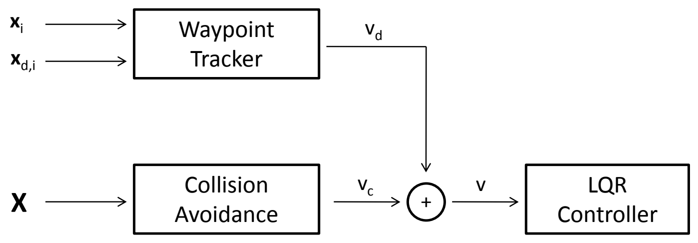


Figure 4.1: Original Boeing-provided control sequence. Note the velocity output from the collision avoidance algorithm is added to the vehicle's desired velocity.

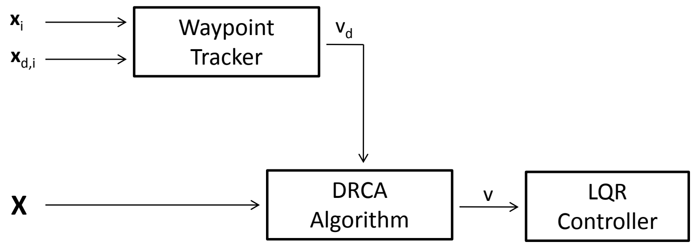


Figure 4.2: New control sequence without additive velocity.

output within the Boeing-provided programming construct. This new process is illustrated in Figure 4.2.

Boeing does not provide an algorithm to keep the vehicles within the bounds of the VICON system. Thus, a simple algorithm was developed so that the vehicles remain within the bounds. If the vehicles encounter a certain boundary in the grid, the desired velocity in that boundary's direction is set to zero. The boundaries used within the algorithm are conservative to ensure that the vehicle will remain visible to the VICON system even if it approaches the grid boundary with a good deal of momentum.

The DRCA algorithm was tested in a Boeing-developed simulation environment. The model uses Matlab Simulink to simulate a specified number of vehicles at 50 Hz. Models in Simulink describe the dynamics, onboard processing, and health of each vehicle. Boeing also developed a user-interface for the simulation, called SwarmView. SwarmView allows one to command and control multiple vehicles real-time, both in simulation and in the actual VSTL testbed. This flow of information is illustrated in Figure 4.3.

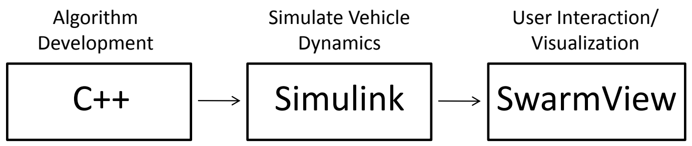


Figure 4.3: Code flow of the DRCA algorithm in simulation.

Several parameters or gains can be adjusted within the DRCA algorithm to produce desired behavior in simulation: d_{sep}, s_{max}, u_{max}, ϵ, γ, and dt. The parameters are adjusted to produce desirable collision avoidance performance. Since d_{sep} is defined for safety and s_{max} and u_{max} are generally properties of the vehicle, the three adjustable parameters are ϵ, $bound$, and dt. The parameters decided upon in simulation prior to the flight tests are given in Table 4.1.

Table 4.1: Parameters used in DRCA simulation.

Parameter	d_{sep}	s_{max}	u_{max}	ϵ	dt	$bound$
Value	1.0	1.25	5.6638	0.8	$0.4 \times \frac{s_{max}}{u_{max}}$	6.0

Four main scenarios are designed to be evaluated both in simulation and in flight test: two-vehicle, head-on conflict; two-vehicle scenario with one vehicle hovering directly in the path of the other vehicle; two-vehicle scenario with one vehicle hovering directly in the path of the other vehicle and collision avoidance turned off for the hovering vehicle, so that it acts as a stationary obstacle; four-vehicle, head-on conflict. In order to provide ample space for the quadrotors to accelerate and avoid each other without encountering the boundaries, their initial position is diagonally offset at a distance greater than γ.

Mission scripts are developed to generate the series of vehicle commands needed to run each scenario. Using mission scripts allows the vehicles to perform their specified tasks simultaneously, which is much better than commanding each vehicle manually (with the operator using a mouse to click waypoints in SwarmView) for two reasons. First, the vehicles cannot be commanded fast enough manually to perform every desired test (particularly the four-vehicle scenario). Second, using mission scripts allows for repeatable results.

The mission scripts were originally written as text files and converted to comma-separated value files. These files are compatible with the simulation, but the version of SwarmView at the VSTL did not support this format. Collaborators at Boeing first suggested the mission scripts be rewritten using Python. However, this process turned out to be complicated, even for the relatively simple commands needed to generate each scenario. Thus, the mission scripts were redesigned using Perl. Each vehicle was commanded to fly to their waypoints at 1.0 m/s, with the exception of vehicles three and four in scenario four. These vehicles are commanded to fly at 0.3 m/s so that all four vehicles reach a common point at approximately the same time.

4.1 Simulation Results

4.1.1 Scenario 1

The results from simulating the first scenario are included in Figures 4.4(a)-4.4(c). Figure 4.4(a) displays the position of the vehicles as they maneuver around one another. Far from looking like a potential function collision avoidance method, where vehicles run into an invisible wall and awkwardly scoot around each other, the paths taken by the vehicles seem more intuitive, similar to humans trying to avoid each other. They follow their desired controls, ignoring the conflict, until they come in bound (within γ). Figure 4.4(b) shows that the vehicles are in conflict until they enter into that boundary and then quickly resolve the conflict (by both turning left). The deconfliction maintenance process is functional as the vehicles do not come back into conflict after initial deconfliction.

The controls in the x and y direction (u_x and u_y) are basically mirror images, since the vehicles are headed directly toward each other, and their prescribed deconfliction maneuver is all-turn-left. After the vehicles take off, they accelerate toward their waypoints. They both turn hard left after coming within a distance of γ. Hard left for vehicle one equates to a positive increase in u_x; for vehicle two it is a negative increase in u_x. After deconflicting, both vehicles accelerate in the direction opposite their deconfliction maneuvers to reorient themselves toward their waypoints. There are not many overshoots, and the controls occasionally saturate at $5.66m/s^2$, which equates to the max acceleration at maximum bank. The vehicles succeed in avoiding each other while remaining in close proximity to their desired paths. Figure 4.4(c) shows how close the vehicles come to their defined minimum separation distance, d_{sep} over time. Most importantly, no collision occurs.

4.1.2 Scenario 2

Figures 4.5(a)-4.5(c) show the performance of the DRCA algorithm in the case where a moving vehicle and a stationary vehicle are trying to avoid one another. They succeed in deconflicting immediately after coming within a distance of γ and remain out of conflict the remainder of the time they are in this bound. The origin lies in the center of a straight-line path between vehicle two's initial position and its waypoint. Vehicle two is able to fly close to the origin because vehicle one is cooperating in collision avoidance, moving out of its way. Although the vehicles do not come back into conflict, the oscillations in u_x show that the vehicles are constantly readjusting (they are unable to come back into conflict because of ϵ). While the controls do not look quite as optimal as the first scenario (it is more oscillatory), the algorithm does work well.

4.1.3 Scenario 3

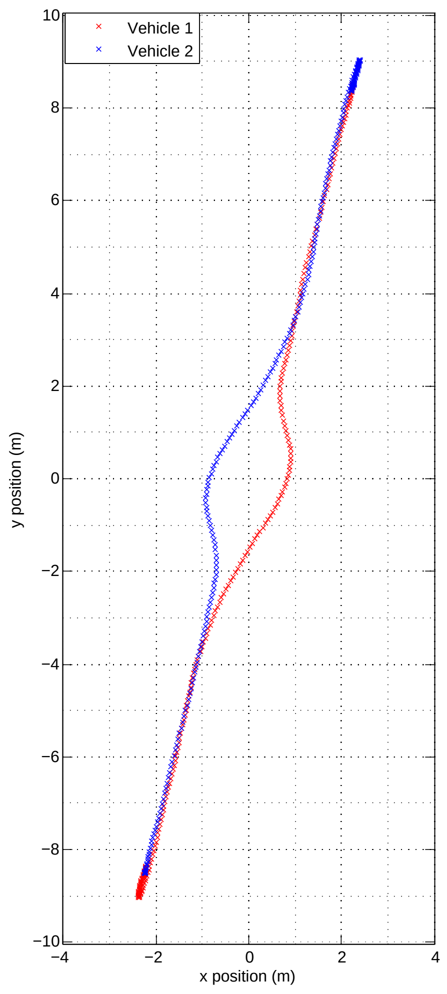
A good collision avoidance algorithm should have no problem avoiding static obstacles. This third scenario was set up to demonstrate how the DRCA handles this case. The simulation results are shown in Figures 4.6(a)-4.6(c). Figure 4.6(a) shows how vehicle two maneuvers smoothly around vehicle one. Once vehicle two comes within a distance of γ, it immediately maneuvers left to exit conflict and remains out of conflict (4.6(b)). Figure 4.6(c) shows that the vehicles remain collision-free (they do not come within d_{sep}).

4.1.4 Scenario 4

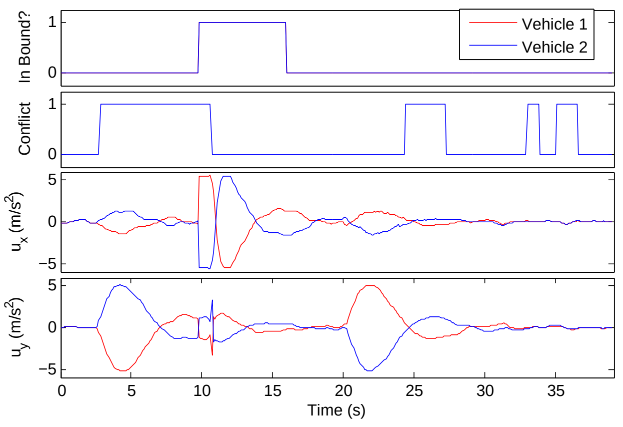
The final scenario demonstrates the DRCA algorithm using four vehicles. Figures 4.7(a)-4.7(d) show the results. The vehicles are diagonally offset so that no vehicle is within a distance of γ of the other vehicles. Both pairs of vehicles are instructed to trade places, but the vehicles closer to the origin are commanded to fly slower so that all four vehicles would reach the origin at approximately the same time (causing a four-vehicle collision with no avoidance running). Figure 4.7(a) shows the flight path each vehicle takes during the simulation. While vehicle one and two smoothly avoid each other, vehicle three and four oscillate frequently as they encounter the $epsilon$ boundary while trying to realign themselves with their waypoints.

Figures 4.7(b) and 4.7(c) show that the vehicles remain conflict free the entire time they are within the bound γ. The controls for vehicles one and two are mirror images (as are the controls for vehicles three and four) since they are each turning left to avoid each other. As expected from observing their flight paths, vehicles three and four have numerous control oscillations. However, the algorithm upholds its guarantee of collision avoidance, and both the deconfliction maneuver and deconfliction maintenance work properly. The oscillations can likely be reduced by changing the parameters, which is done prior to the final flight test.

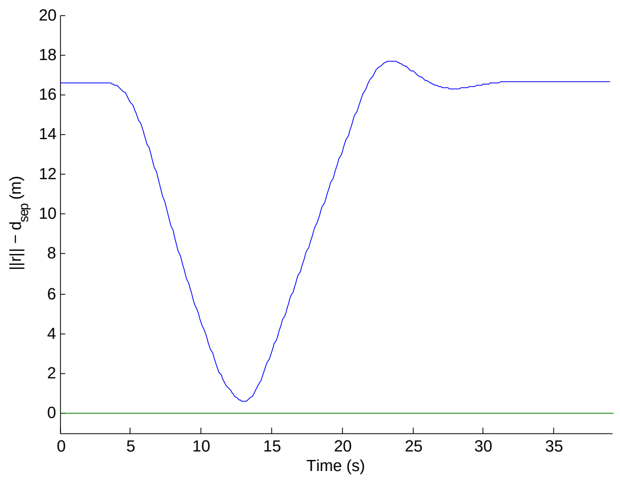
Figure 4.7(d) demonstrates that the system remains collision-free. Notice that vehicles one and four and vehicles two and three remain almost exactly the same distance apart for all time, showing that the system has good symmetry–an expected characteristic of an all-turn-left collision avoidance algorithm. Vehicles one and two do not come as close together because they travel around vehicles three and four.



(a) Trajectories

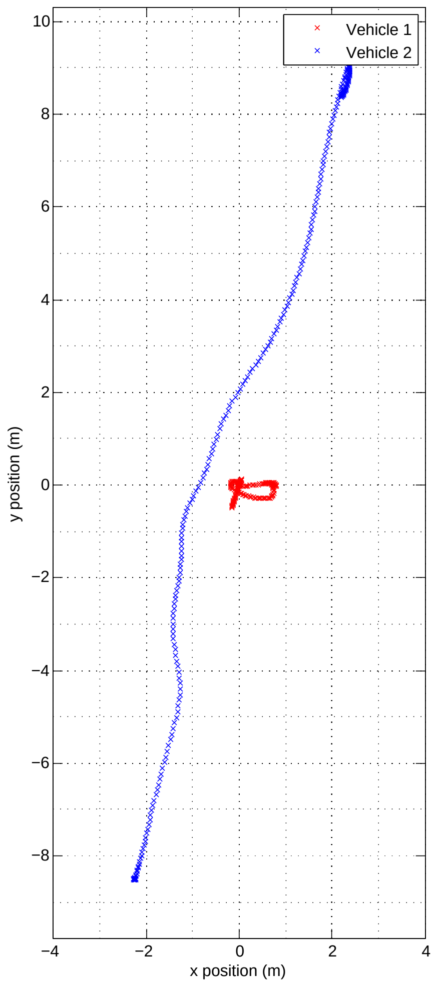


(b) Algorithm performance

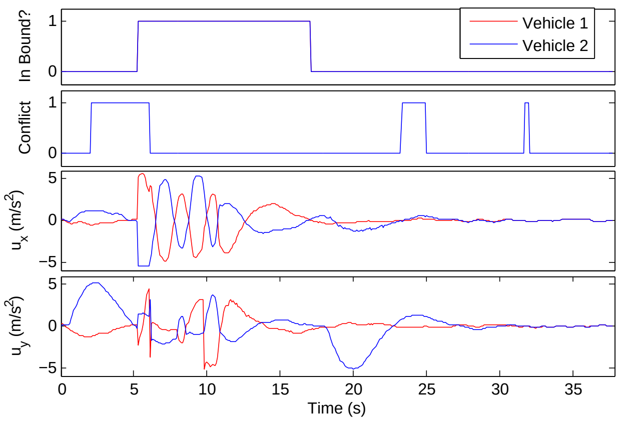


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

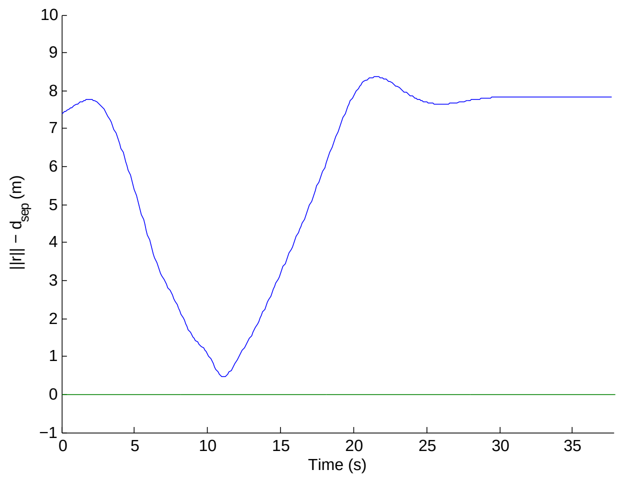
Figure 4.4: Scenario 1 simulation.



(a) Trajectories

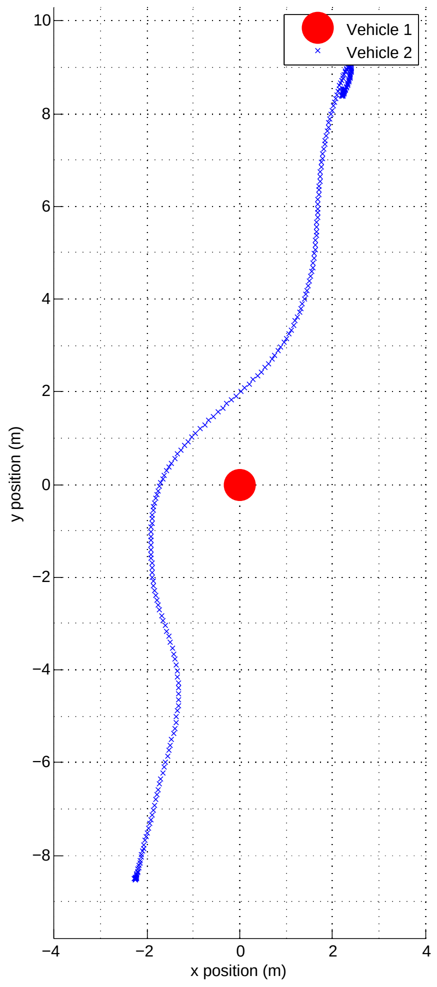


(b) Algorithm performance

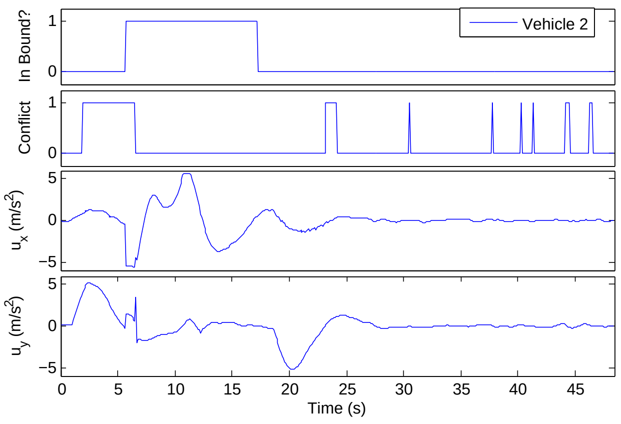


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

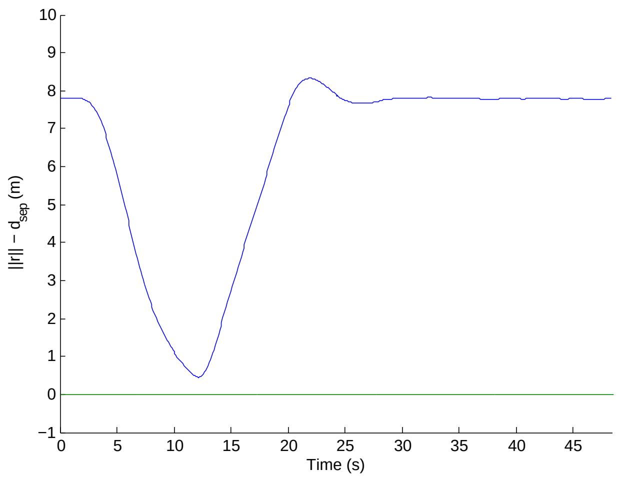
Figure 4.5: Scenario 2 simulation.



(a) Trajectories

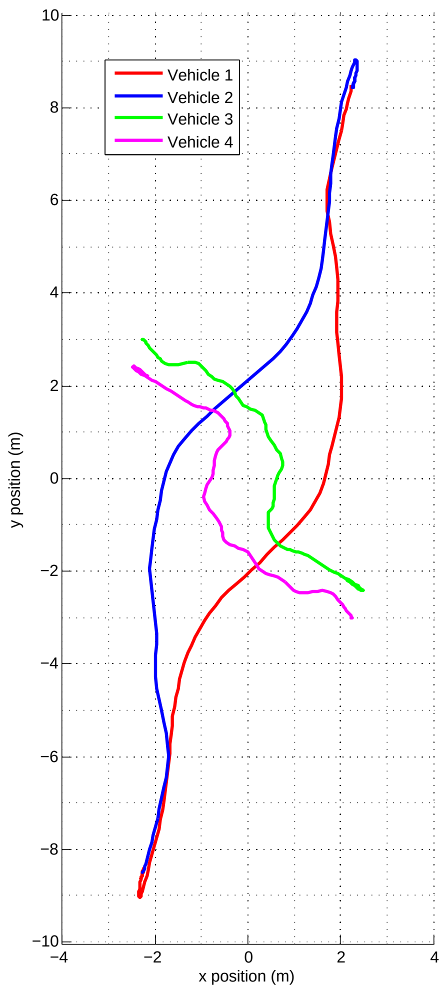


(b) Algorithm performance

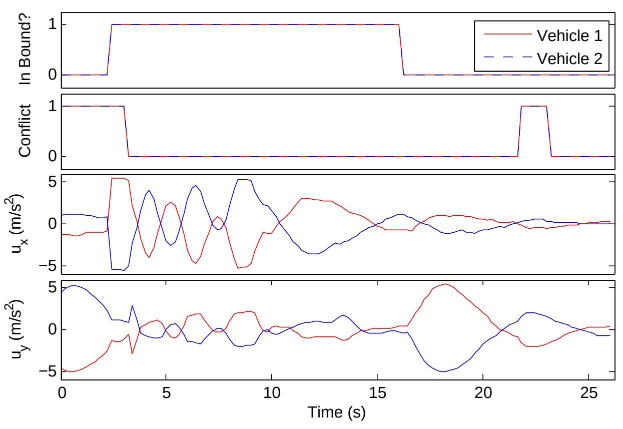


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

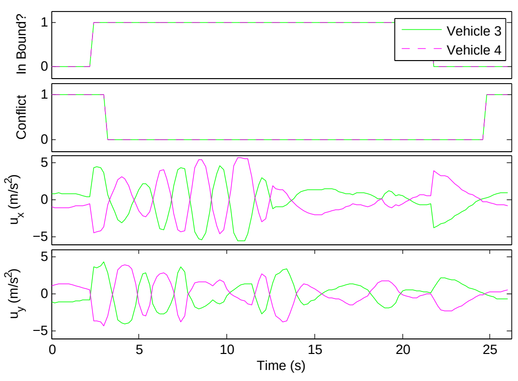
Figure 4.6: Scenario 3 simulation.



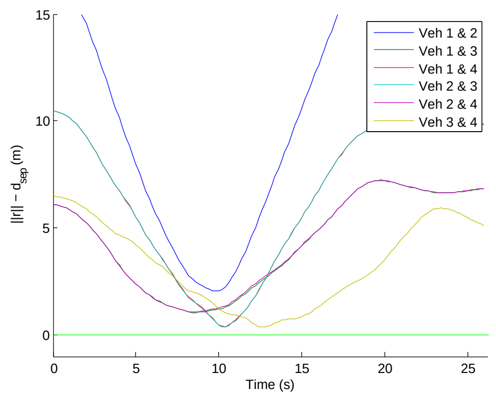
(a) Trajectories



(b) Algorithm performance for vehicles 1 and 2



(c) Algorithm performance for vehicles 3 and 4



(d) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

Figure 4.7: Scenario 4 simulation.

Chapter 5
INITIAL FLIGHT TESTS

5.1 Fight Test 1

The first flight test was both a success and a failure. The vehicles did succeed in avoiding one another; however, they did not navigate as expected from simulation. The first problem encountered was that the version of SwarmView at the VSTL did not support the mission scripts used to run the scenarios in simulation. The inability of the system to support mission files meant that each vehicle had to be commanded manually with unfamiliar default values for certain parameters such as the speed of the waypoint tracker, which turned out to be an issue.

There were also hardware problems with the quadrotors. After a long day of testing, there were only two quadrotors left in working condition. During the course of the DRCA flight tests, another vehicle ceased to operate properly, leaving only one working vehicle. These hardware difficulties made the flight tests much longer and arduous than anticipated.

The DRCA algorithm did enable the vehicle to avoid collisions, but the vehicles did not navigate as expected from simulation. The two-vehicle head-on collision (scenario one) was the first scenario fight tested. As expected, the vehicles followed their desired controls until entering the specified bound between them. There was then a sort of oscillatory behavior from the vehicles. It was as if one vehicle was trying to avoid the other by an unreasonable amount–so much so that it would try to exit the test space. The algorithm would prevent it from doing so by zeroing the velocity in the escaping direction. The vehicle then oscillated around the boundary.

A simpler two-vehicle scenario (scenario two) with one stationary vehicle (that is also running the DRCA algorithm) was then flight tested. The same behavior was observed, and the vehicle would not fly to the waypoint on the opposite side of the grid. Finally, scenario three was tested, which is the same as scenario two except that collision avoidance is turned off for the stationary vehicle. In this flight test, the vehicle finally made it to the waypoint, but its path traveled along the outside of the grid.

Unfortunately, data was only collected for one of the flight tests. Figures 5.1(a)-5.1(c) show the results from flight testing scenario three and compares it to simulation results. One source of the flight path error seen in Figure 5.1(a) comes from the lower default speed of the vehicles in manual control ($0.5 m/s$) compared to their speed using mission scripts ($1.0 m/s$).

The gains and parameters in the DRCA algorithm were designed for the vehicles to operate with mission scripts, and the different speed in manual mode was unknown until after flight test. This different speed caused a problem because

$$\epsilon > s_{max} \tag{5.1}$$

meaning the velocity barrier, ϵ, was actually a larger velocity than the vehicle could attain. The vehicle tries to avoid this barrier until approaching the edge of the grid. It slides up the artificial "wall" included in the algorithm to keep vehicles from going out of bounds, keeping the maximum distance possible from the other vehicle. Once it passes the static vehicle, it maintains maximum separation until it is outside the distance γ. The vehicle then proceeds directly to the prescribed waypoint.

Figure 5.1(b) shows that the vehicle exits conflict as soon as it comes within distance γ. However, the vehicle does reenter conflict while still within distance γ at about t = 17 s, which corresponds to the time when the vehicle tries to exit the grid but instead is forced the opposite way. Notice that u_x spikes in the positive direction at this same time. Thus, it can be assumed that without the boundary control in the algorithm, the vehicle would have remained conflict free for all time. Figure 5.1(c) shows that the vehicles remain collision-free, but they are also separated by more than two meters beyond d_{sep}.

5.2 Flight Test 2

For the second flight test, Perl mission scripts were used to command the vehicles. Better results were expected after eliminating the inconsistency in waypoint tracker speed. Nevertheless, the second flight test results were no better than the first, although much more data was collected. All four scenarios were run and compared to the simulation results. The first parameters tested are included in Table 4.1. After obtaining poor results using these parameters, ϵ and dt were changed on-site endeavoring to produce better flight test results. Several tests were run varying these parameters, and the results that follow are the best results obtained for each scenario. Simulations were run retroactively to reflect the new parameters, thus every comparison seen in the following plots is between flight tests and simulations that use the same parameters.

Because of a data processing error, the internal logging that was output from the DRCA algorithm at 50 Hz was only collected for vehicle one in each scenario. The internal logging reports x and y position for each vehicle, a counter to keep track of time, u_x and u_y for each vehicle, and boolean variables that report if each vehicle is within the bound, γ, or conflicted. Fortunately, the data from the VICON system (100 Hz) exports the position data for each vehicle and keeps a counter for time, so most of the plots could be reconstructed. The flight path and vehicle separation data comes from the VICON position reference system. The conflict and control information use data from internal logging within the DRCA algorithm itself, so it is shown for vehicle one only.

5.2.1 Scenario 1

The results for scenario one were extremely close to simulation results. The parameters for this test are those in Table 4.1, but ϵ and dt were modified to be 0.2 and $1.0 \times \frac{s_{max}}{u_{max}}$ respectively. Using these parameters, the simulation and flight test results had very similar paths (Figure 5.2(a)), algorithm performance (Figure 5.2(b)), and collision avoidance curves (Figure 5.2(c)). The only significant difference is the higher acceleration (u_y) vehicle one experiences in flight test compared to simulation. Simulation and flight test results both exit conflict immediately when the vehicles come in bound and do not reenter conflict while within a distance of γ, indicating that the deconfliction maneuver and the deconfliction maintenance both function as intended.

Figure 5.3(d) shows that the speed of the vehicles (a cause for concern in the first flight test) are certainly on the same order in flight test and simulation. The flight test data in Figures 5.3(a), 5.3(c), and 5.3(d) are constructed from the VICON system logging (100 Hz), while Figure 5.3(b) presents data from vehicle one using internal logging within the DRCA algorithm itself (50 Hz).

5.2.2 Scenario 2

The parameters for this test were those in Table 4.1, but ϵ and dt are modified to be 0.2 and $1.4 \times \frac{s_{max}}{u_{max}}$ respectively. Figures 5.3(a)-5.3(c) show that the flight test results are quite different from the simulation results. Figures 5.3(a), 5.3(c), and 5.3(d) use data from the VICON system (100 Hz), while Figure 5.3(b) captures the conflict and control information of vehicle one using data from internal logging within the DRCA algorithm itself (50 Hz). Note that in simulation, the static vehicle tries to avoid the moving vehicle. The static vehicle moves because the vehicles come closer together in simulation; in flight test the vehicles are so far apart that the static vehicle does not need to move to avoid the other vehicle (Figure 5.3(c)).

The static vehicle's movement can also be observed from the static vehicle's controls in Figure 5.3(b). Also note that in simulation, the vehicles come back into conflict after initially deconflicting, indicating deconfliction maintenance is not functioning properly, and thus the modified parameter values are unacceptable in simulation. The moving vehicle has similar speeds in flight test and simulation until it comes within γ. After coming in bound, the vehicle flies quite a bit faster in flight test than simulation. The vehicle should never be flying that fast because $s_{max} = 1.25$ m/s, and the algorithm saturates the speed at s_{max}.

5.2.3 Scenario 3

The third scenario involves collision avoidance of one vehicle with a static obstacle (a stationary vehicle not running any collision avoidance). The vehicle had trouble navigating around the obstacle using the parameters in Table 4.1, so ϵ and dt were changed to 0.3 and $1.4 \times \frac{s_{max}}{u_{max}}$ respectively. The resulting flight path is shown in Figure 5.4(a). Various trials of this scenario using other parameters caused the moving vehicle to pass on the right instead of the left. The moving vehicle should never pass the static vehicle on the right since the prescribed deconfliction maneuver is all-turn-left.

The simulation results for vehicle two shown in Figure 5.4(b) indicate that the vehicle comes back into conflict, and deconfliction maintenance is not functioning in simulation. The flight test results are not shown because internal logging failed for vehicle two, and vehicle one is not running DRCA in this scenario. Figure 5.4(c) shows that vehicle two goes around vehicle one by almost three meters more than d_{sep}. Figure 5.4(d) shows that, once again, the speed of vehicle two is very similar in flight test and simulation until coming within γ. After that the results differ, although they are on the same order.

5.2.4 Scenario 4

The four-vehicle scenario was designed to cause all vehicles to converge at the origin almost simultaneously. This scenario was tested twice and resuled in collisions between vehicles each time. The parameters for this test are those in Table 4.1, but ϵ and dt were modified to be 0.3 and $1.0 \times \frac{s_{max}}{u_{max}}$ respectively. Although this scenario works in simulation (even better with all the original parameters from Table 4.1), the results in flight test are much different (Figures 5.5(a)-5.5(d)).

The vehicles deadlocked when they came near the origin and began oscillating (see the controls of vehicle one in Figure 5.5(b)). This erratic and unstable oscillatory motion caused vehicles to reenter conflict on multiple occasions (Figure 5.5(b)). It ultimately resulted in two different collisions–between vehicles two and four and vehicles one and three (see Figure 5.5(c)). Employing these same parameters in simulation caused vehicles to come back into conflict (Figure 5.5(b)), but did not cause collisions (Figure 5.5(c)). Finally, looking at the speed of the vehicles in 5.5(d) shows that there is not much difference between the maximum speeds of flight test and simulation but they both exceed s_{max}.

5.2.5 Flight Test 2 Analysis

Analyzing the data from flight test two revealed significant differences between simulation and flight test results, with the exception of scenario one. Scenario one reinforced the fact that the vehicles were indeed running DRCA in flight test, and that the algorithm actually worked. However, without access to the quadrotors' controller algorithm (Boeing proprietary information), it was very difficult to troubleshoot these problems. The best troubleshooting mechanism available was the simulation, which was frustrating because of the obvious discrepancies between simulation and flight test.

Nevertheless, hypotheses were developed and pursued in seeking to resolve simulation and flight test differences. The first hypothesis is that a time delay at some point in the system could be causing problems. Although this hypothesis is not intuitive (for instance, how can a time delay induce a vehicle to maneuver right instead of left), it is beneficial to rule out this hypothesis experimentally. A time delay is introduced into the algorithm by storing each vehicle's velocity command for a finite time before commanding the vehicle to fly that velocity. Time delays were introduced in simulation for scenario three. The delays are implemented from zero to three seconds in increments of 0.2 seconds. Figure 5.6 shows how the time delay in simulation changes the way vehicle two maneuvers around vehicle one. Note that it never produces a trajectory like that observed in flight test, nor does it ever make vehicle two maneuver to the right of vehicle one.

A second hypothesis was that the parameters are somehow differently scaled in simulation to flight test. Thus, if parameters can be found that replicate flight test trajectories when implemented in simulation, a gain can be found to account for the difference in the parameters linearly. If this theory is sound, the gain can then be inverted and used to force flight test results to behave like the simulations. In order to find such parameters (if they exist), it is first useful to do a parameter sensitivity study.

After simulating scenario three many times varying one parameter at a time, the following generalizations can be made. Increasing ϵ will cause the moving vehicle to avoid the stationary vehicle by a larger distance (i.e. it will go further left to maneuver around the vehicle). Additionally, increasing ϵ will cause the vehicle to maneuver sooner. Increasing dt will cause the vehicle to make larger oscillations in its trajectory. Decreasing dt will cause smaller oscillations and will cause the moving vehicle to pass closer and closer to the stationary vehicle until dt reaches a certain magnitude. After this point, the vehicle will start to fly much slower and will actually make a right turn maneuver around the stationary vehicle. Decreasing s_{max} and u_{max} has the same effect as decreasing dt. It is unclear why, since s_{max} and u_{max} are designed to saturate the speed and acceleration of the vehicle. Finally, d_{sep} and γ perform as they were designed. Increasing d_{sep} causes the moving vehicle to stay further away from the static vehicle; increasing γ causes the moving vehicle to perform its deconfliction maneuver further away from the stationary vehicle.

After performing the sensitivity analysis, some parameters were chosen to try to emulate a flight test trajectory in simulation. The parameters that produced this result are shown in Table 5.1.

Table 5.1: Parameters in simulation that best emulate flight test trajectory for scenario 3.

Parameter	d_{sep}	s_{max}	u_{max}	ϵ	dt	$bound$
Value	1.0	1.25	5.6638	2.9	$2 \times \frac{s_{max}}{u_{max}}$	6.0

Figure 5.7 shows the resulting simulation flight path compared to the flight test trajectory. Now

that parameters have been found to emulate flight test results in simulation, a linear gain can be used to describe the relationship between flight test and simulation parameters. The hypothesis will be tested by inverting the gain and multiplying it by each of the original simulation parameters in Table 4.1 to get new parameters to use in flight test. The hypothesis will be validated if these flight test results with these new parameters match the simulation results shown in Chapter 4. This theory is tested in the third scenario of flight test three. The third hypothesis was that perhaps the controllers on the quadrotors prefer the vehicles to turn right, which would help explain why the quadrotors have a tendency to go right around a stationary vehicle. Testing this hypothesis only involves a sign change on the transformation matrix L in (2.12). This theory was investigated in the next flight test.

5.3 Flight Test 3

The third and final Boeing flight test is designed to investigate hypotheses developed after the second flight test, to acquire flight tests for every scenario using the original parameters (Table 4.1), and to validate previous results. Better results are not expected since no significant changes are made, but the tests are helpful in troubleshooting the inconsistencies between flight test and simulation.

5.3.1 Scenario 1

The results of flight testing the two vehicle head-on scenario using parameters in Table 4.1 are shown in Figures 5.8(a)-5.9(a). Figure 5.8(a) shows the flight paths of the vehicles. Though the deconfliction maneuver commands all vehicles to turn left, they turn right in flight test, which indicates a serious problem. The vehicles land before reaching their waypoints because the mission scripts only allot a certain amount of time to travel to their waypoints before the command to land is given. Figure 5.8(b) gives the control and algorithm information for flight test and simulation.

Figure 5.8(c) shows that the vehicles avoid each other by 3 meters more than necessary. It also shows that the mission takes longer in flight test than in simulation, which is confirmed by the fact that the vehicles did not reach their waypoints before landing. This longer flight time makes the speed of the vehicles in simulation versus flight test of particular interest. Figure 5.9(a) shows the speed of the vehicles in flight test are about half that of the simulation speeds. In fact, dividing the flight test time vector by 2 shows how close the new speed (since speed is calculated as $\frac{\Delta position}{\Delta time}$)

matches with simulation results (Figure 5.9(b)).

Since previous flight testing of the DRCA algorithm sometimes resulted in vehicles maneuvering to the right around other vehicles rather than left, L is made negative to force the vehicles to all turn right. Figure 5.10 shows the resulting flight paths compared to the all-turn-left maneuver. Note that the two tests are virtually identical, indicating a serious problem with the Boeing controller.

5.3.2 Scenario 2

The second scenario is flight tested using the original parameters from Table 4.1. Figures 5.11(a)-5.11(d) show the results. Figure 5.11(a) shows the flight test results are despicable and bear no semblance to the simulation results. Figure 5.11(b) gives the control and algorithm information for flight test and simulation. The controls are erratic, and the vehicles frequently enter and exit conflict. Figure 5.11(c) shows that once again, the vehicle maneuvers around by much more than necessary in flight test. Finally, just as in scenario one, the flight test speed for vehicle two is roughly half the magnitude of the simulation speed (Figure 5.11(d)).

5.3.3 Scenario 3

The third scenario is the last scenario flight tested using parameters from Table 4.1. Figures 5.12(a)-5.12(d) show the results. Figure 5.12(a) shows that vehicle two's trajectory in flight test bears no resemblance to the simulation trajectory once it enters a distance of γ. Figure 5.12(b) gives the control and algorithm information for flight test and simulation. The vehicle frequently enters and exits conflict after coming in bound. The controls are erratic from the moment the vehicle enters γ until the vehicle lands. Figure 5.12(d) shows that the flight test speed is once again about half the magnitude of simulation speed.

The third scenario was run again using the inverted parameters discussed previously. In summary, a gain, h, was multiplied by each parameter to make simulation results look as close as possible to flight test results. Thus, $\epsilon_{simulation} = h\epsilon_{FT}$. Theoretically, inverting this process would make flight test results look like simulation results, so that $\epsilon_{FT} = \frac{1}{h}\epsilon_{simulation}$. This concept holds for all parameters changed, not just ϵ. Table 5.2 shows the new parameters used for this flight test utilizing the method above.

Table 5.2: Parameters in simulation that best emulate flight test trajectory.

Parameter	d_{sep}	s_{max}	u_{max}	ϵ	dt	$bound$
Value	1.0	1.25	5.6638	0.031	$0.98 \times \frac{s_{max}}{u_{max}}$	6.0

Figure 5.13 shows the resulting trajectory in flight test, which does not correspond to simulation results. Therefore, the hypothesis that proposes a different scaling between simulation and flight test parameters is invalid.

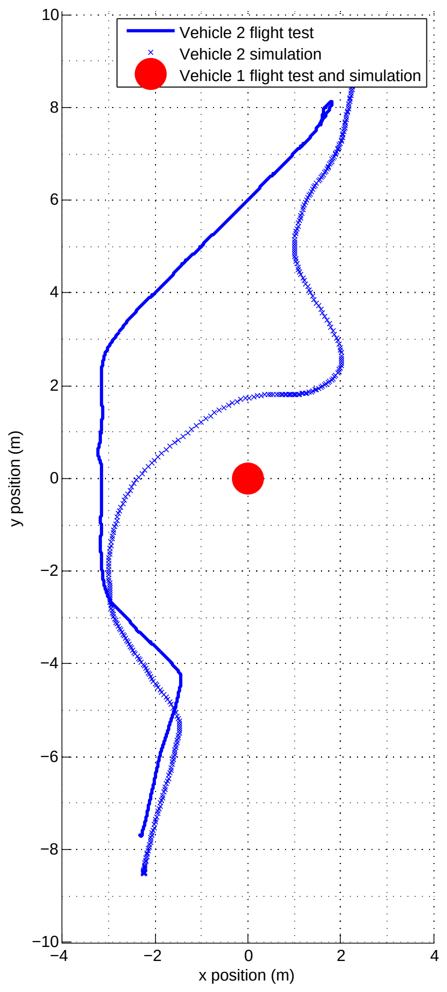
5.3.4 Scenario 4

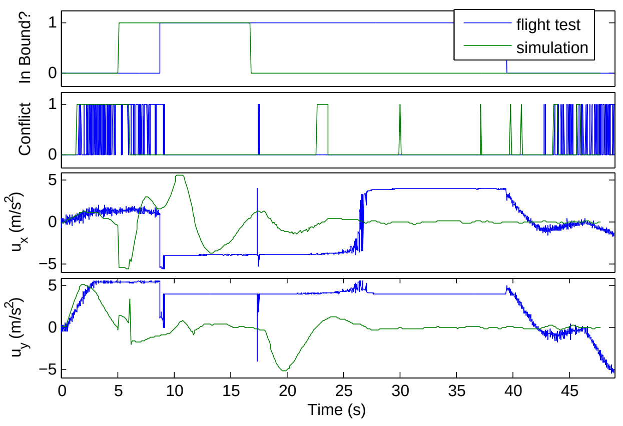
The four-vehicle scenario was not flown in this flight test. The differences between simulation and flight test were not yet reconciled, and disastrous results were expected, as in the second flight test.

5.4 Source of Errors

After months of trying to troubleshoot the differences in flight test and simulation without access to Boeing's controller algorithm, the Boeing VSTL found a bug in their controller that caused the flight test errors. The bug caused some issues with self-vehicle velocities, which is not surprising considering the results of Flight Test 3. Correcting this bug fixed the discrepancies between flight test and simulation results, producing a successful flight test for the four-vehicle scenario. The results of the final round of flight tests will be discussed in the next chapter.

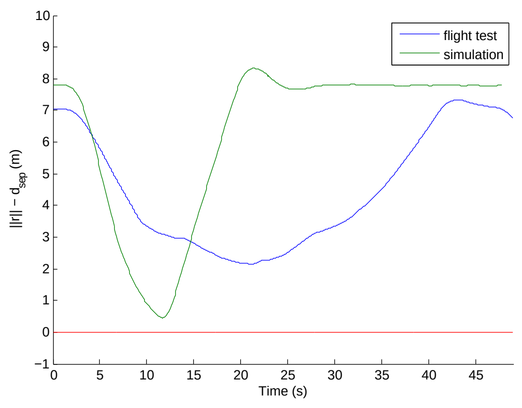
Before flight testing, the Boeing VSTL found another minor error. The mode selected in the Perl scripts enabled both the DRCA algorithm and another collision avoidance algorithm that the Boeing VSTL had previously tested. The other collision avoidance algorithm was developed by Massachusetts Institute of Technology (MIT). Disabling the MIT avoidance algorithm in simulation shows that it did not make a significant difference in the results. Figures 5.14(a)-5.14(b) portray the similar results for the most complex case tested–scenario four.





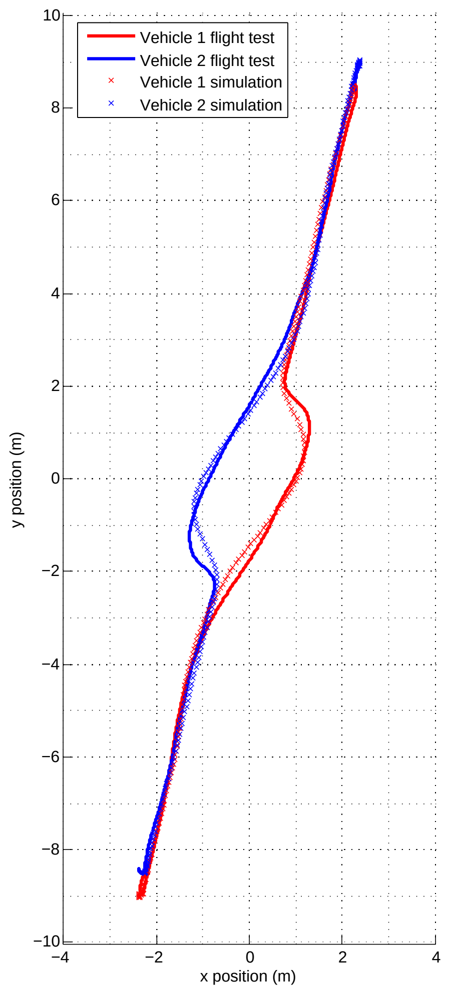
(a) Trajectories

(b) Algorithm performance

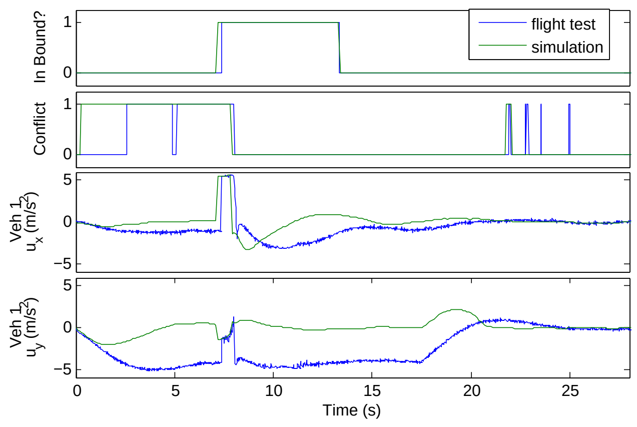


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

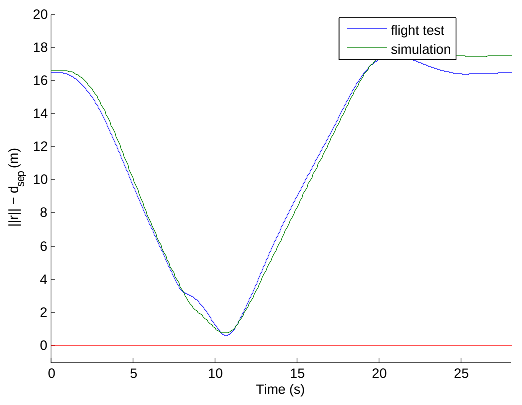
Figure 5.1: Flight test 1, scenario 3.



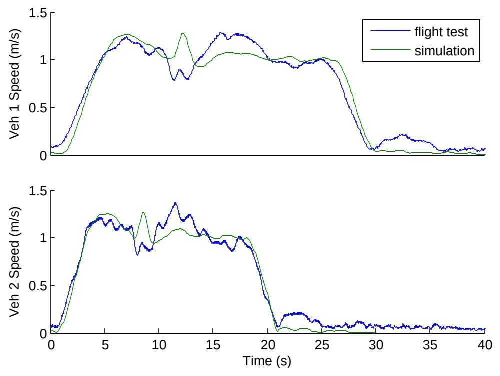
(a) Trajectories



(b) Algorithm performance (Vehicle 1)

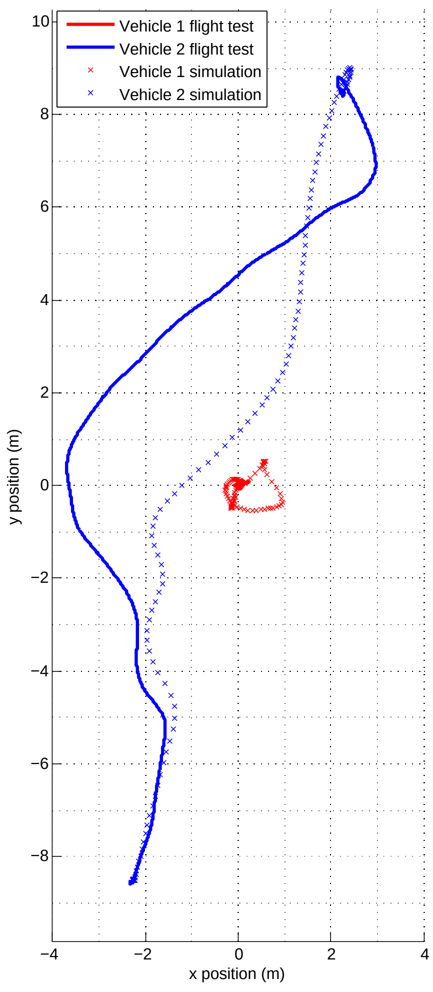


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

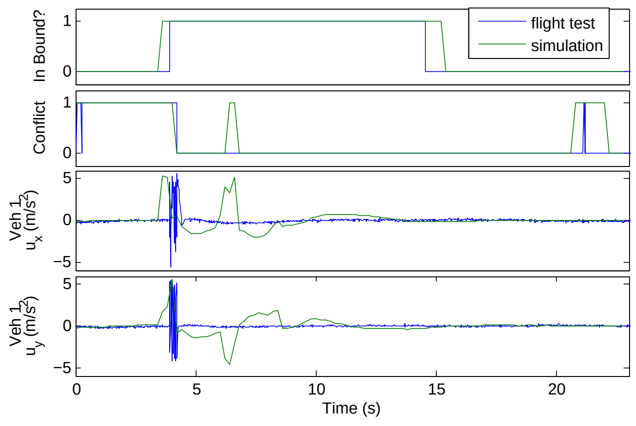


(d) Vehicle speed

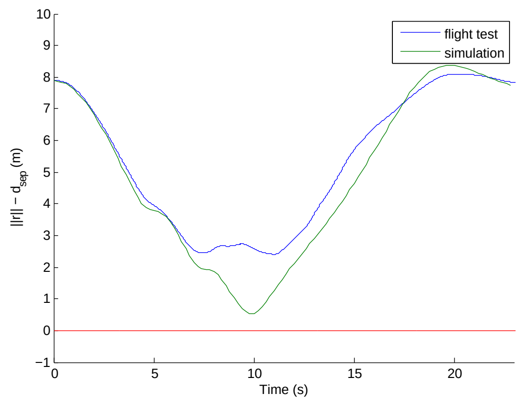
Figure 5.2: Flight test 2, scenario 1.



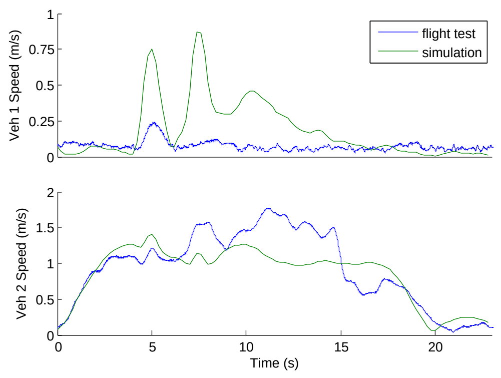
(a) Trajectories



(b) Algorithm performance

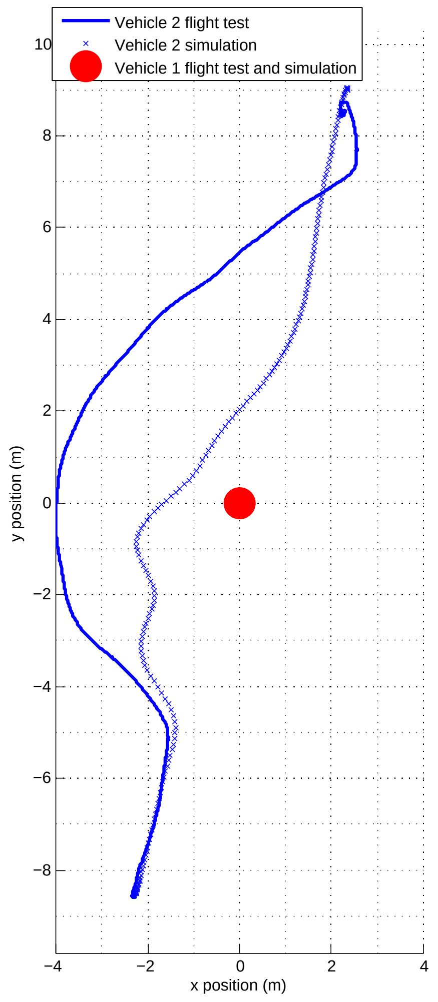


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

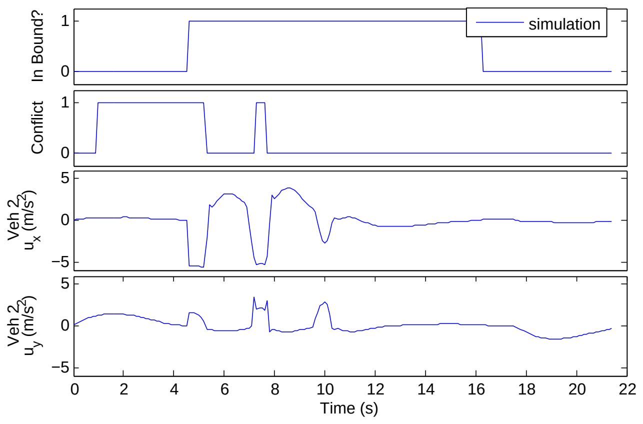


(d) Vehicle speed

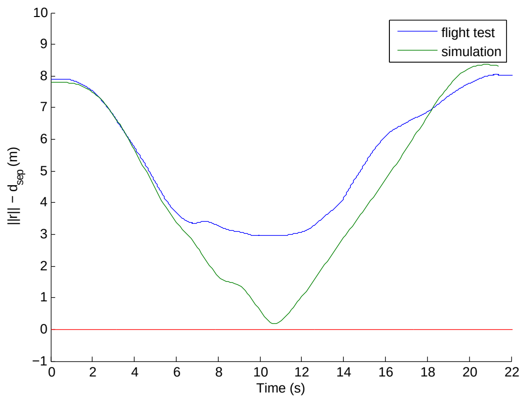
Figure 5.3: Flight test 2, scenario 2.

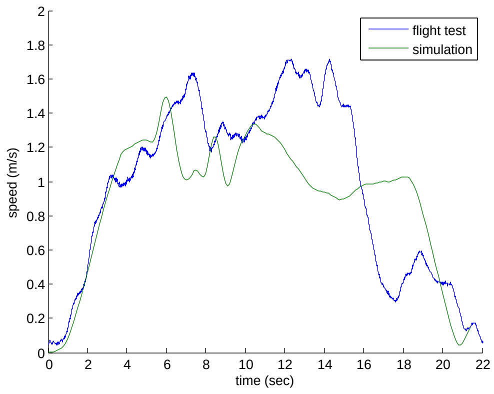


(a) Trajectories



(b) Algorithm performance

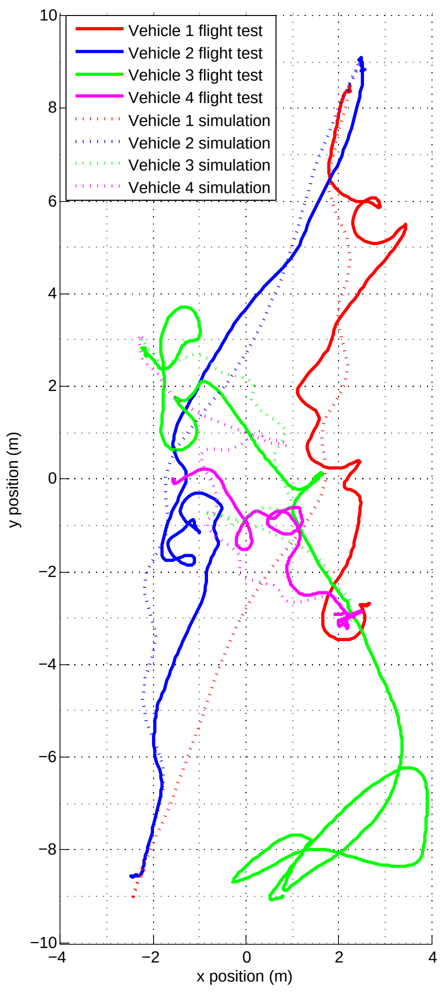




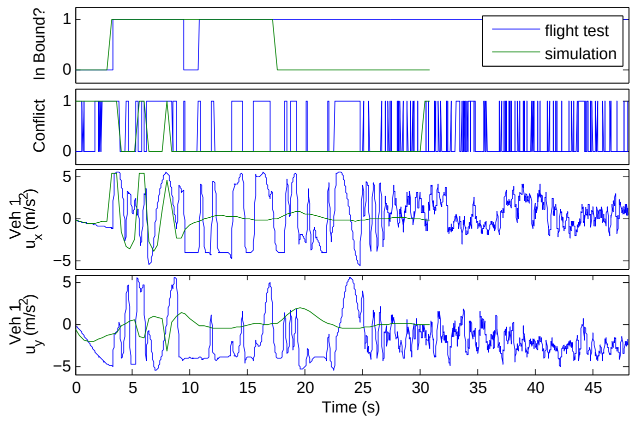
(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

(d) Vehicle speed

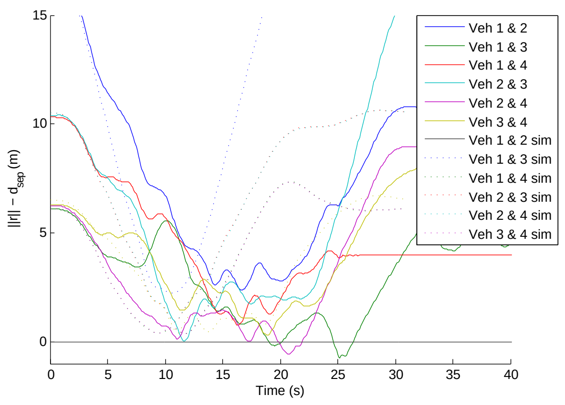
Figure 5.4: Flight test 2, scenario 3.



(a) Trajectories

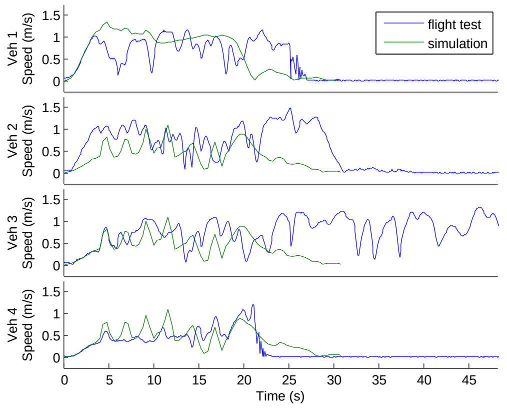


(b) Algorithm performance



(c) Vehicle separation minus minimum separation distance.

Note that a collision occurs if the curve goes below zero.



(d) Vehicle speed

Figure 5.5: Flight test 2, scenario 4.

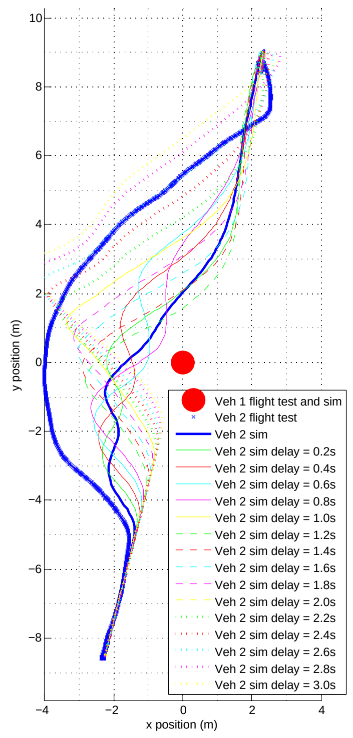


Figure 5.6: Vehicle trajectories after implementing different time delays in simulation.

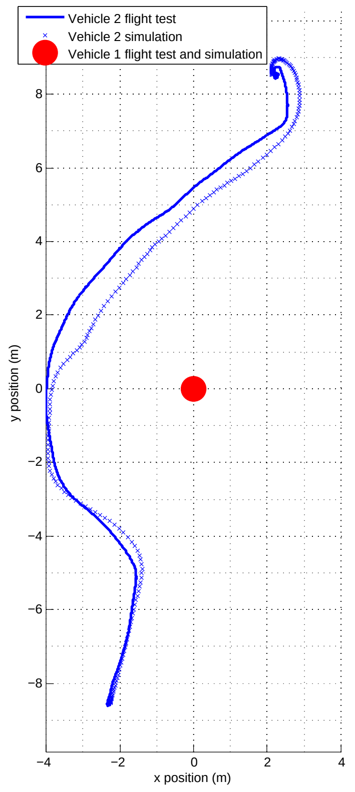
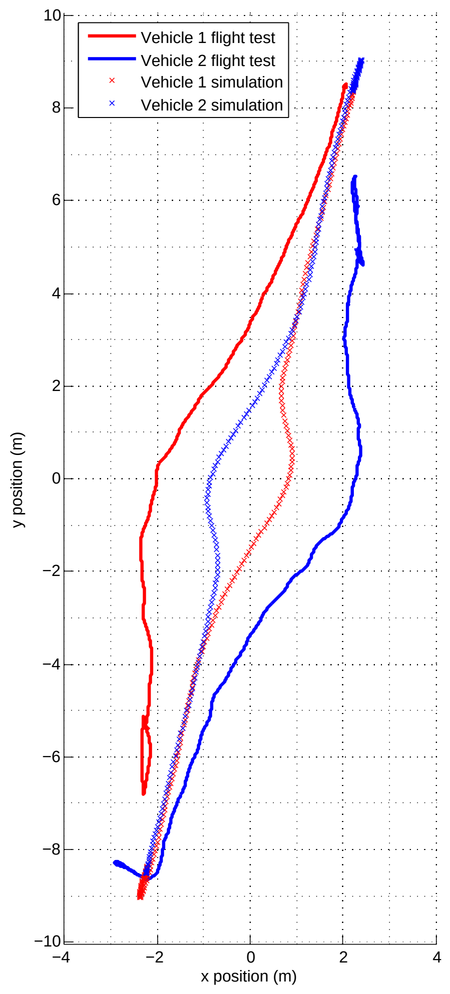
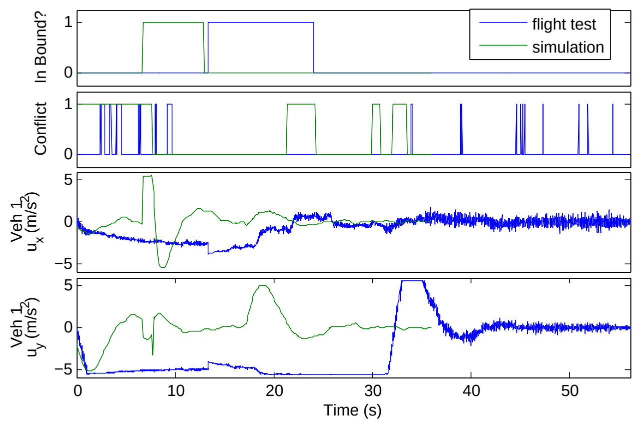


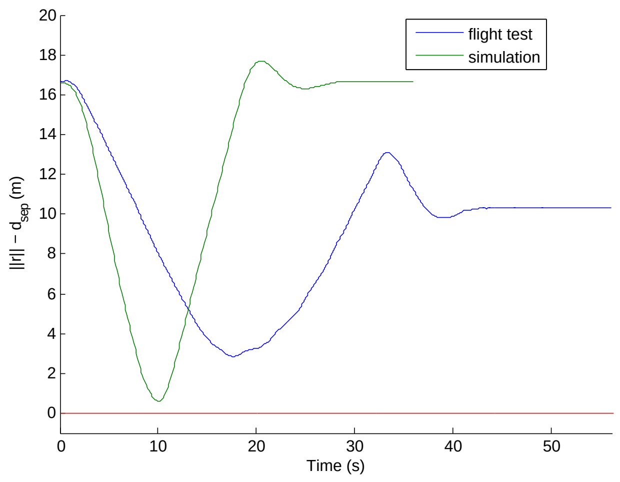
Figure 5.7: Comparison of vehicle trajectories from simulation (using parameters in Table 5.1) and flight test (using parameters from Table 4.1). The simulation parameters are chosen so that the vehicle emulates the flight test trajectory.



(a) Trajectories

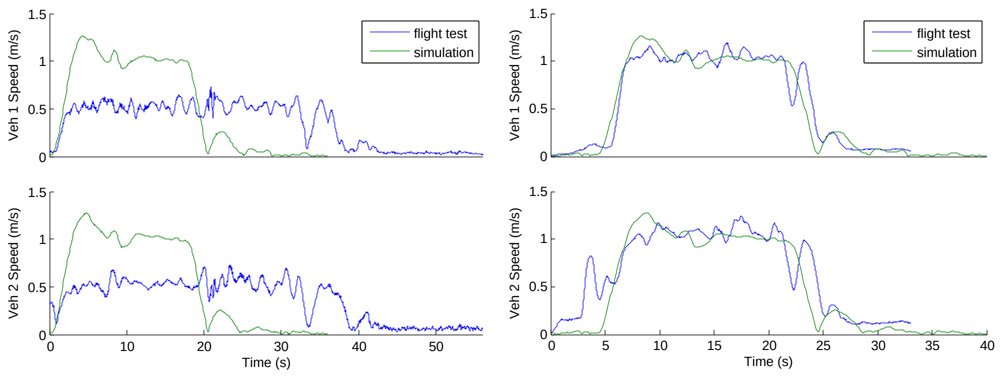


(b) Algorithm performance



(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

Figure 5.8: Flight test 3, scenario 1.



(a) Vehicle speed

(b) Vehicle speed that would result from halving the flight test time vector.

Figure 5.9: Flight test 3, scenario 1 speed comparison.

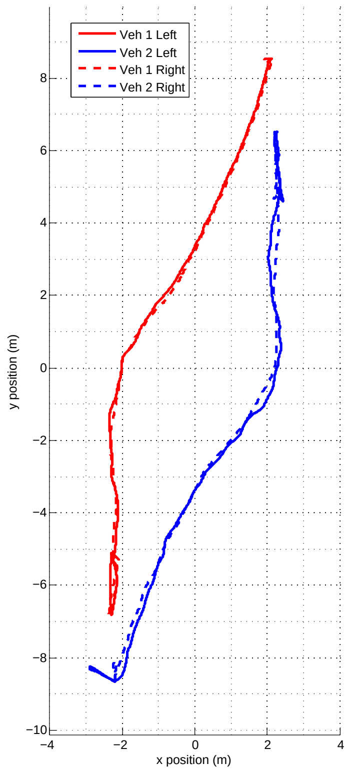
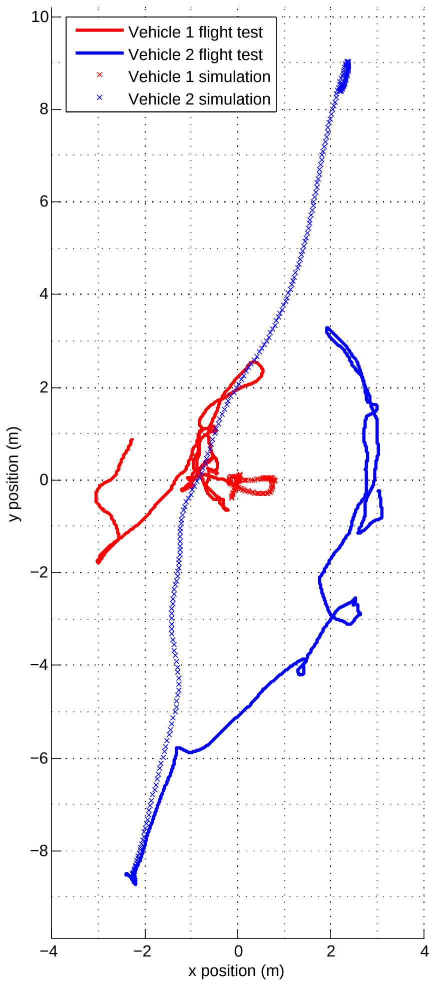
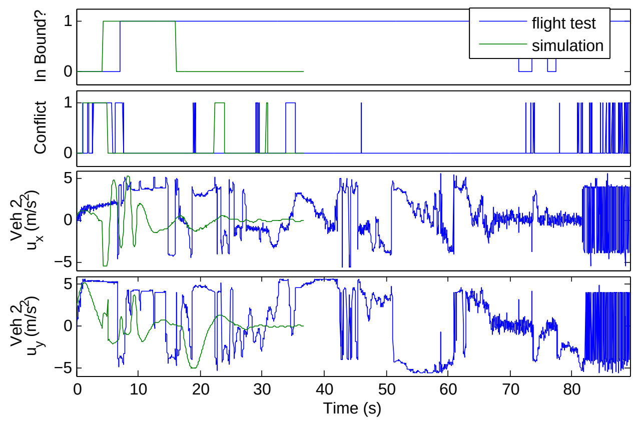


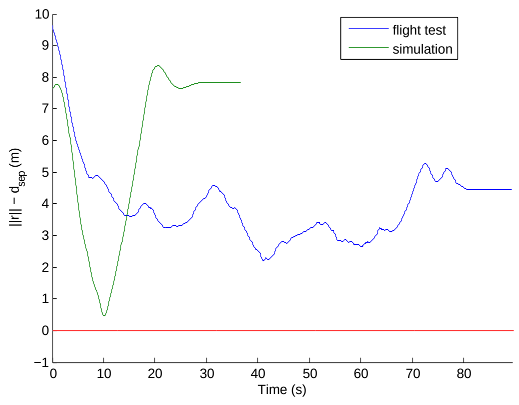
Figure 5.10: Flight test 3, scenario 1 - Comparison of all-turn-left deconfliction maneuver with all-turn right. The similar trajectories indicate a serious problem with the controller in calling the DRCA algorithm.



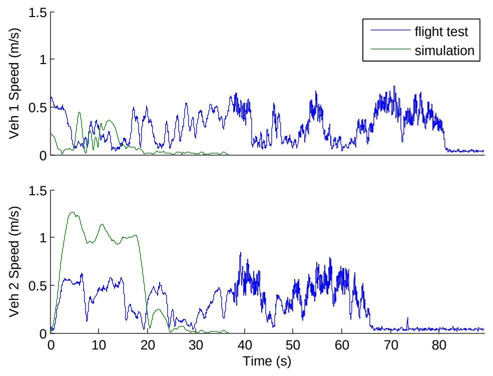
(a) Trajectories



(b) Algorithm performance

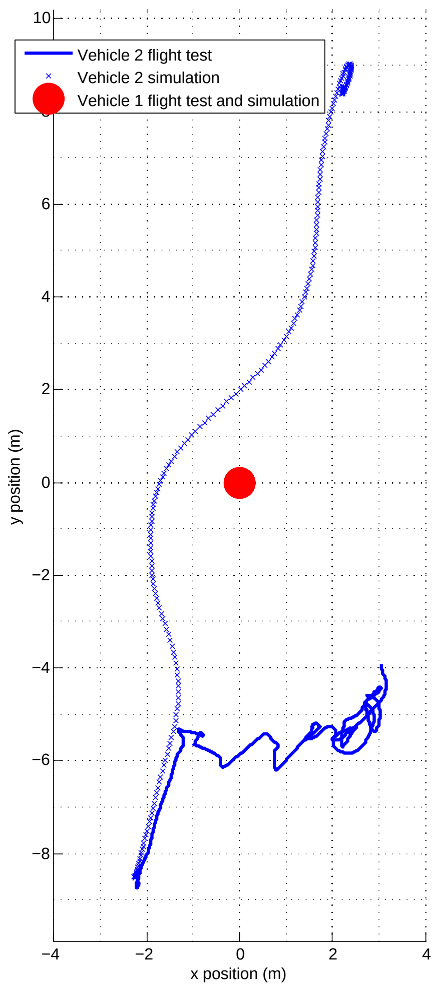


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

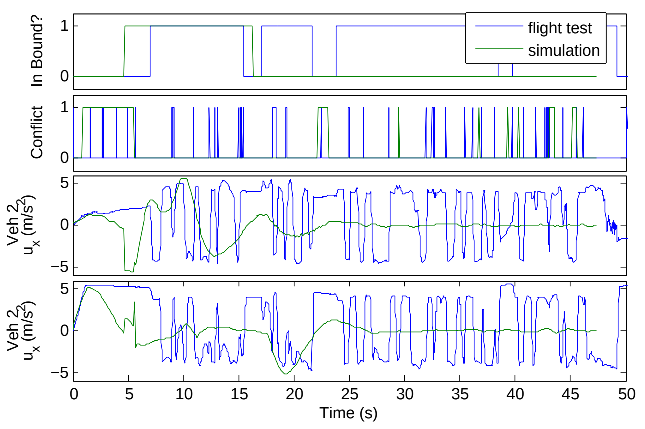


(d) Vehicle speed

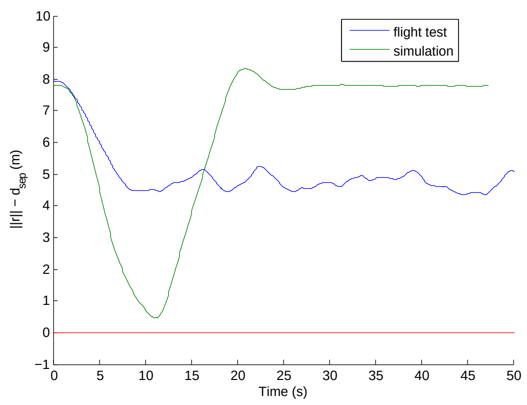
Figure 5.11: Flight test 3, scenario 2.



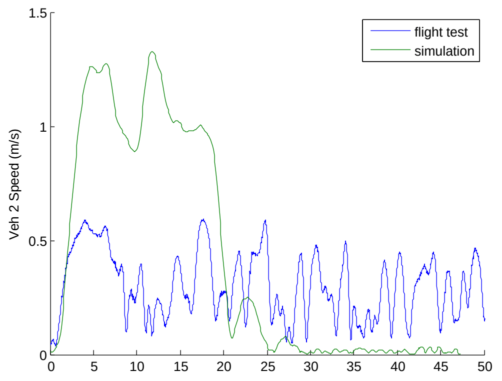
(a) Trajectories



(b) Algorithm performance



(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.



(d) Vehicle speed

Figure 5.12: Flight test 3, scenario 3.

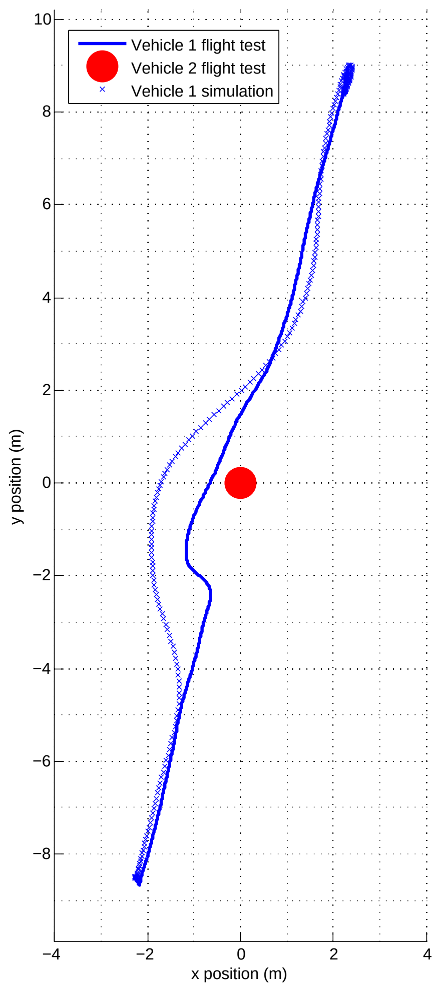
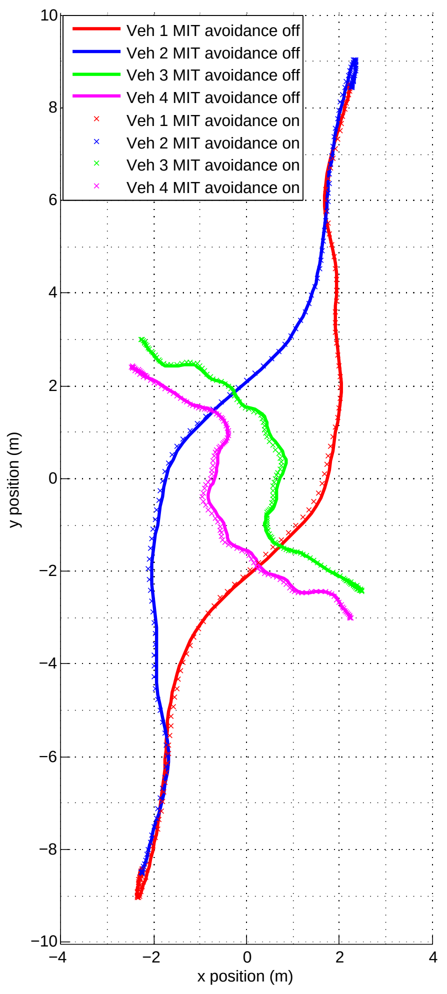
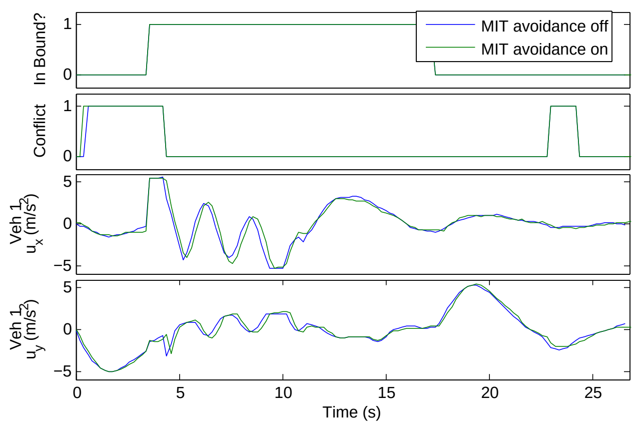


Figure 5.13: Flight test 3, scenario 3 - New trajectory of vehicle after changing parameters to those in Table 5.2 using the inversion method. If the theory is correct, the resulting trajectory should be identical to the simulation trajectory for this scenario, which is clearly not the case.



(a) Trajectories



(b) Algorithm performance

Figure 5.14: Comparison of simulation results with MIT avoidance turned on and off.

Chapter 6

FINAL FLIGHT TEST

To correct some of the oscillations evident in scenarios two and four, ϵ was increased from 0.8 to 1.0. The parameters used for the final round of flight tests are included in Table 6.1. They are, of course, compared to simulation results that use the same parameters.

Table 6.1: Parameters used in DRCA simulation.

Parameter	d_{sep}	s_{max}	u_{max}	ϵ	dt	$bound$
Value	1.0	1.25	5.6638	1.0	$0.4 \times \frac{s_{max}}{u_{max}}$	6.0

6.1 Scenario 1

The two-vehicle head-on flight test performs very close to simulation results (Figures 6.1(a)-6.1(d)). Figure 6.1(a) shows that the paths are similar, but that they do not line up perfectly. In flight test, the vehicles begin their deconfliction maneuver earlier and seem to have a little bit of a lag in the controls when compared to simulation. Figure 6.1(b) shows that controls in flight test are delayed compared to simulation, which results in different trajectories. Other than this delay, the controls are quite similar. Figure 6.1(b) also shows that the vehicles deconflict a bit slower in flight test compared to simulation. Figure 6.1(d) shows the speed of the vehicles, which further confirms the delay in the system. This delay is discussed further in the Analysis section. Figure 6.1(c) shows that the vehicles have very similar separation with respect to time in simulation and flight test. Since the curve does not go below zero, the vehicles succeed in avoiding collision. The DRCA algorithm behaves as designed in scenario one.

6.2 Scenario 2

The two-vehicle stationary case (with DRCA activated on both vehicles) yields another successful flight test (Figures 6.2(a)-6.2(d)). Figure 6.2(a) shows the flight paths of the vehicles are virtually identical. Likewise, Figure 6.2(b) shows that the controls and conflict resolution are extremely close. The speed (Figure 6.2(d)) has some minor variations, but is very similar overall. As expected, the speed of vehicle two has a maximum at 1.25 m/s. Figure 6.2(c) shows that the vehicles maintain similar separation over time in simulation and flight test. It also shows that the vehicles do not collide. Overall, the DRCA algorithm performs as designed.

6.3 Scenario 3

Flight testing the static obstacle case (with no collision avoidance running on the stationary vehicle) gives results similar to those in simulation (Figures 6.3(a)-6.3(d)). Like the first scenario, there seems to be a bit of delay in the flight test that shows up in the flight path (Figure 6.3(a)) and the controls (Figure 6.3(b)). Other than this delay, the flight path and controls look similar in flight test and simulation, as does the algorithm performance. Observing Figure 6.3(c) shows that the vehicles in flight test and simulation have similar separation distances. Upon closer inspection, the vehicles come about 0.2 m closer in simulation than in flight test. There is no apparent reason why the vehicles would come closer together in simulation, since all the parameters are identical to those in flight test. Figure 6.3(d) shows the speed of the vehicle and verifies the delay in the hardware environment. It also shows that the maximum speed in simulation is greater than the 1.25 m/s specified. This difference in speed probably causes the vehicles to come closer together in simulation (notice the time of maximum speed corresponds to the time of minimum separation distance).

6.4 Scenario 4

The four-vehicle head-on flight test was also successful, matching very closely to the simulation (Figures 6.4(a)-6.4(b)). Figure 6.4(a) shows the trajectories of all four vehicles in flight test and compares them with simulation. All flight paths match up fairly well. Figures 6.5(a)-6.5(d) show that the controls are similar between flight test and simulation, although they have a control lag similar to the lag in scenario one. Figure 6.4(b) portrays the similar separation distances in flight test and simulation, and confirms that there are no collisions between any vehicles. Figure 6.4(c) shows similar speeds for simulation and flight test, although the flight test spikes a bit higher at the time when the vehicles are closest together.

The 4-vehicle scenario basically performs as expected from the theoretical results of the DRCA algorthim. One questionable behavior is the way vehicles 3 and 4 navigate in reaching their terminal points. Both of these vehicles make a hard right turn at the end of their flights to reach their terminal points. Upon closer inspection, u_x causes a hard right turn immediately when the vehicles come out of the bound, γ. Thus, deconfliction maintenance must be keeping the vehicles from navigating directly to their waypoints. Since the vehicles are moving slowly, the nearest conflict will be formed if they both stop (since $\tilde{\mathbf{v}}$ will be on the tip of the collision cone). The desired control directs the vehicles to fly to their waypoints, stop, and land. The natural equilbirium in deconfliction maintenance (zero acceleration, nonzero speed) tends to form when $\tilde{\mathbf{v}}$ is at $\frac{\epsilon}{2}$ (halfway inside the buffer). Thus, the algorithm prevents the vehicles from flying to the waypoint and stopping until they are outside the bound, γ.

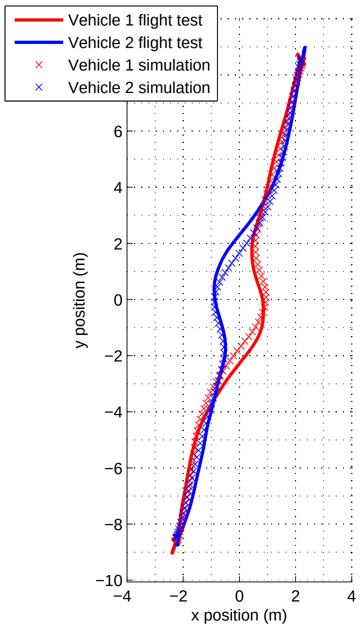
The paths of vehicles one and two seem to be fairly optimal. They resemble the two-vehicle head-on case from scenario one. Vehicles three and four, however, seem to navigate suboptimally. They do not go directly to their waypoints after deconflicting, but instead continue in a right turn that seems to overshoot the waypoint. Interestingly, it is not the DRCA algorithm that causes this behavior. A closer look at the waypoint tracker reveals that this suboptimal maneuver is actually a result of the desired control. The waypoint tracker is designed so that the vehicle follows a “carrot” that moves in a straight line from the initial position to the waypoint at the commanded waypoint speed. Vehicles three and four are commanded a waypoint speed of 0.3 m/s so that they reach the origin at approximately the same time as vehicles one and two. In the course of their deconfliction maneuvers, vehicles three and four fly a bit faster than 0.3 m/s. Thus, they must navigate back to the carrot, resulting in a longer right turn before flying toward the waypoint at 0.3 m/s. Figure 6.6 shows this result graphically.

6.5 Analysis of Final Flight Test

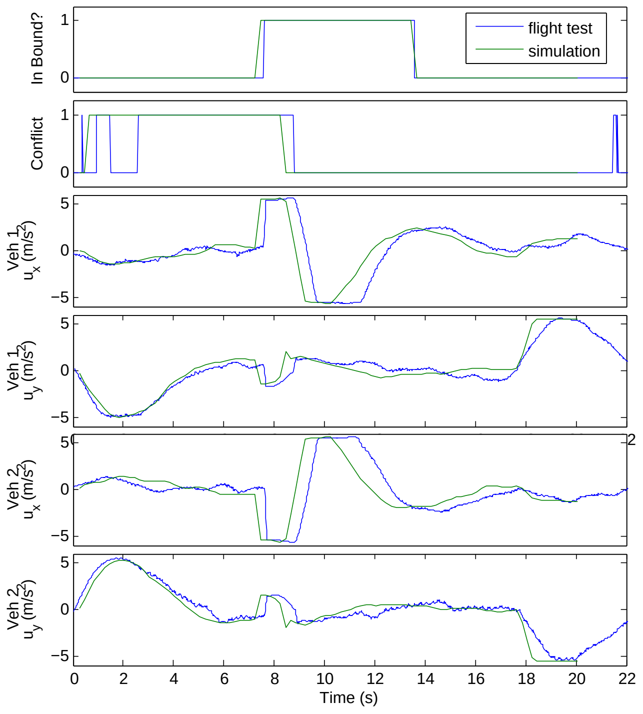
The system delay given in [13] is 10 milliseconds. However, up to 0.35 seconds of delay is experienced between simulation and flight test in scenario one and scenario four (vehicles one and two). Contrastingly, there is not as much delay in scenarios two and three, which both address avoidance of a stationary object. Thus, the delay may be algorithmic, when one computes the accelerations in simulation. In flight test, the accelerations are formed directly from VICON measurements.

The DRCA algorithm's saturation function works superbly for acceleration and fairly well for speed. Observing all of the above scenarios, the highest speed is 1.4 m/s in simulation (scenario three) and 1.6 m/s in flight test (scenario four). It makes sense that the acceleration (control) saturation works better than the speed saturation because the DRCA algorithm operates on the control level, not the velocity level. Besides, in most cases the saturation functions in the DRCA algorithm are redundant. They should already be defined in the vehicle controllers.

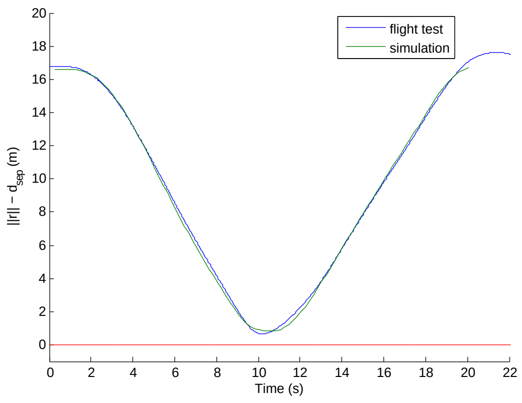
The results from these flight tests and simulations verify the functionality of the DRCA algorithm. Note that in all scenarios, when the vehicles come in bound (and if they are in conflict), they immediately execute their deconfliction maneuver (in this case, all-turn-left). Once out of conflict, deconfliction maintenance is engaged to keep the vehicles out of conflict. Deconfliction maintenance continues until the vehicles exit the bound, γ, and can once again follow their desired controls. Most importantly, there are no collisions in any of the above scenarios.



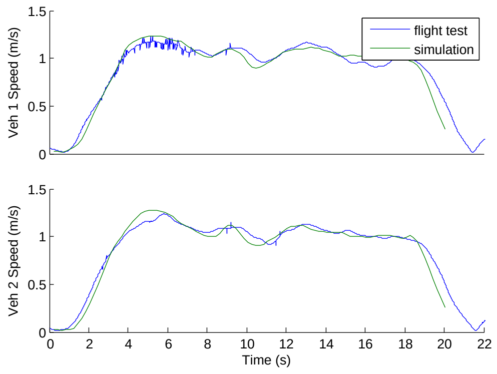
(a) Trajectories



(b) Algorithm performance

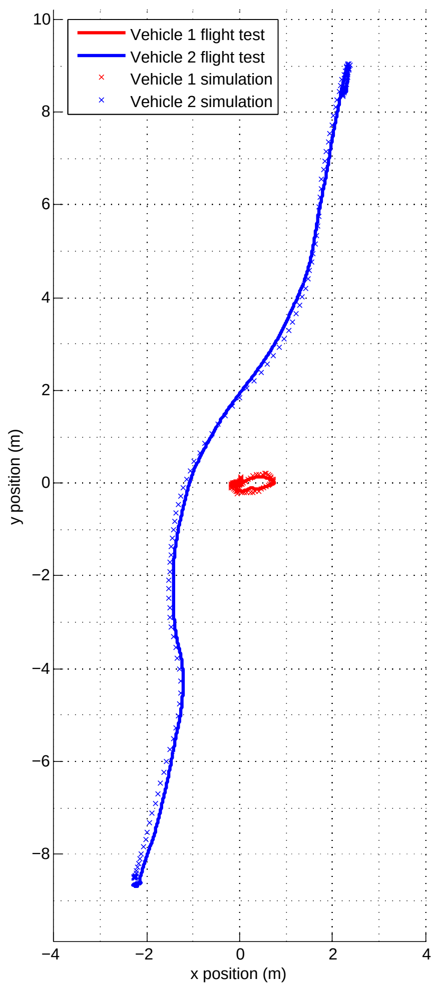


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

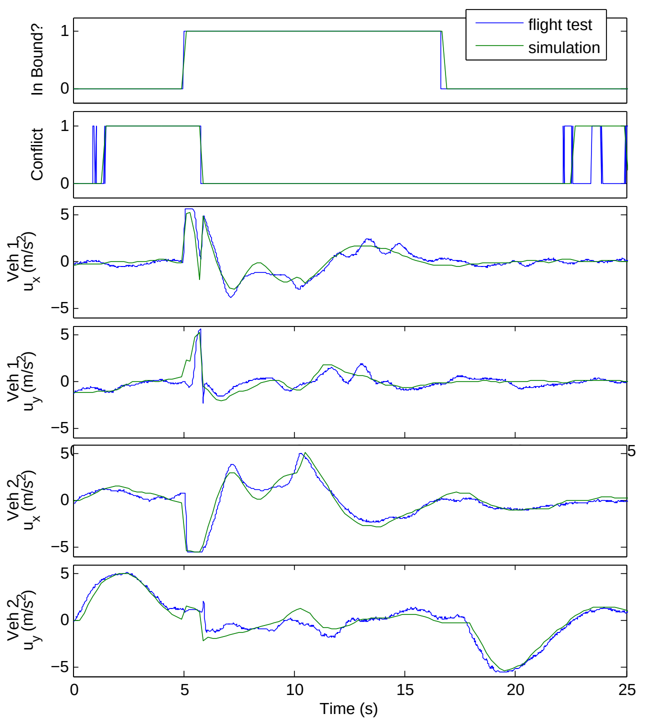


(d) Vehicle speed

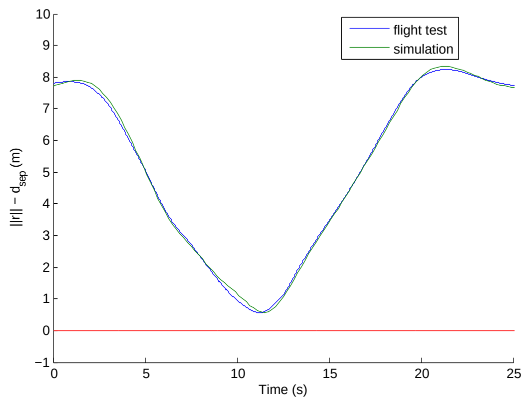
Figure 6.1: Flight test 4, scenario 1.



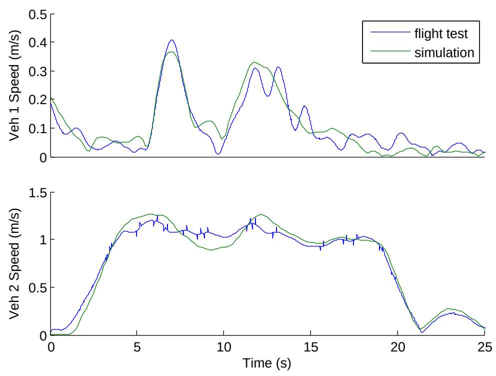
(a) Trajectories



(b) Algorithm performance

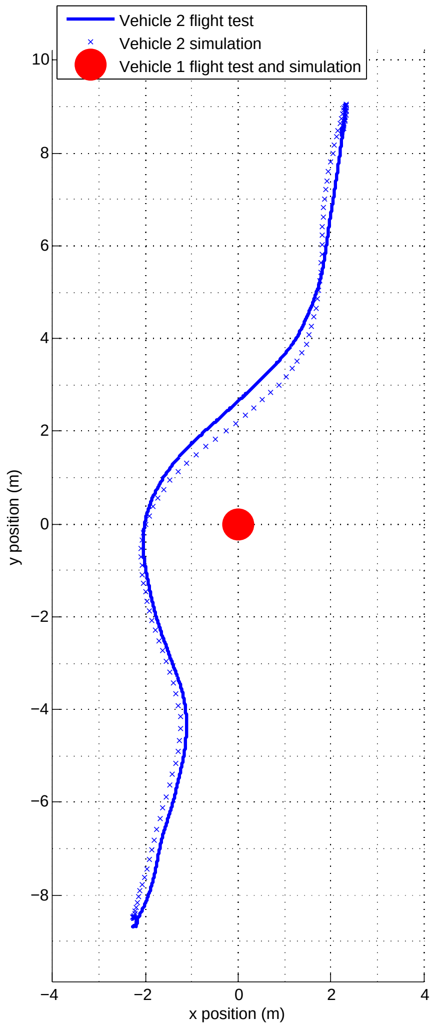


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

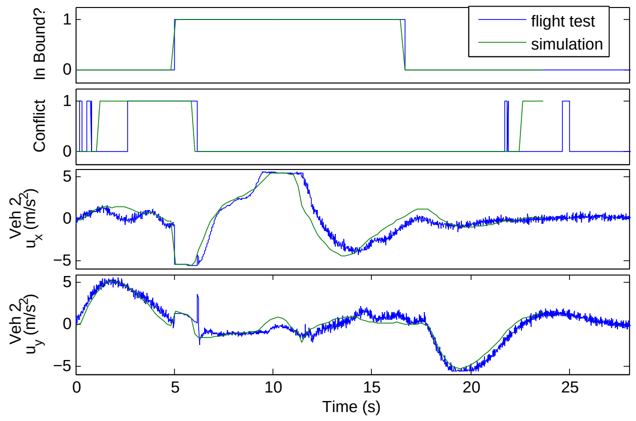


(d) Vehicle speed

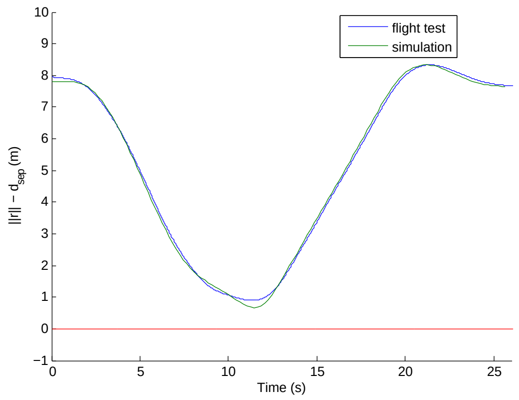
Figure 6.2: Flight test 4, scenario 2.



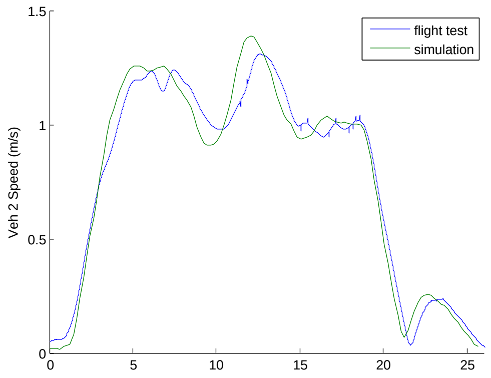
(a) Trajectories



(b) Algorithm performance

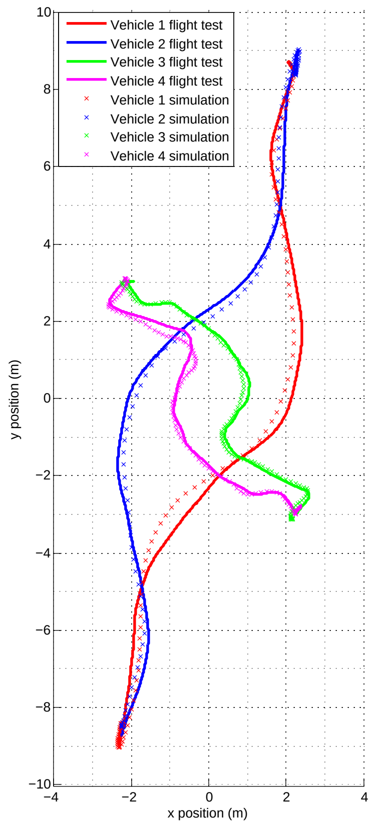


(c) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

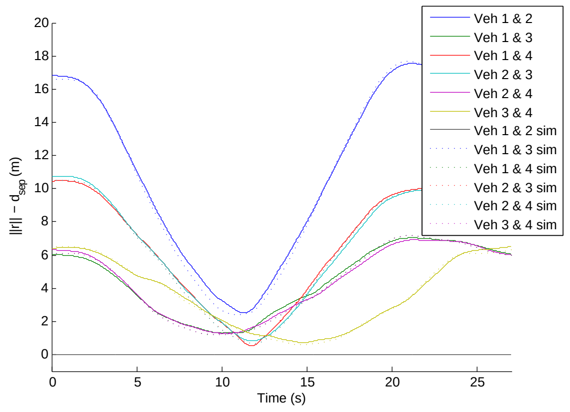


(d) Vehicle speed

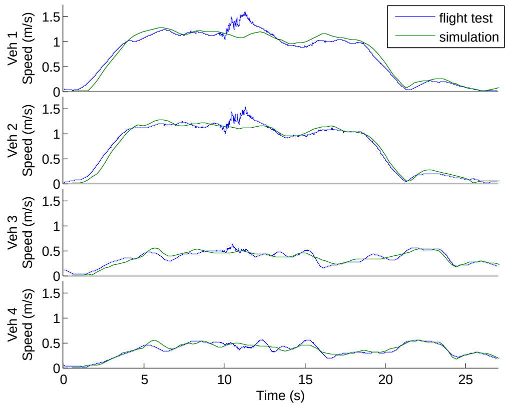
Figure 6.3: Flight test 4, scenario 3.



(a) Trajectories

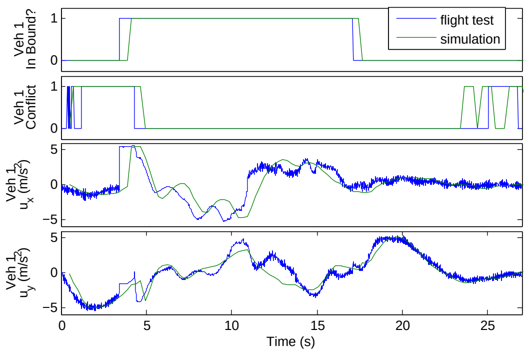


(b) Vehicle separation minus minimum separation distance. Note that a collision occurs if the curve goes below zero.

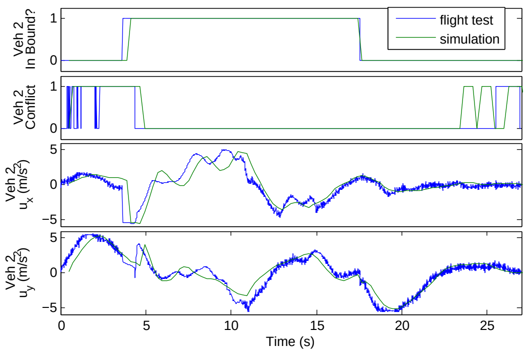


(c) Vehicle speed

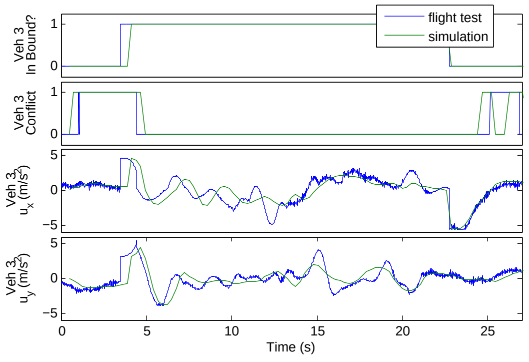
Figure 6.4: Flight test 4, scenario 4.



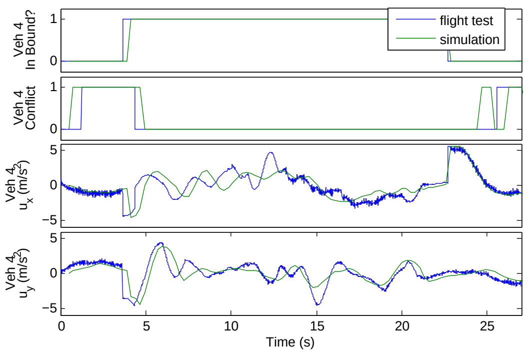
(a) Algorithm performance for vehicle 1



(b) Algorithm performance for vehicle 2



(c) Algorithm performance for vehicle 3



(d) Algorithm performance for vehicle 4

Figure 6.5: Flight test 4, scenario 4 algorithm performance.

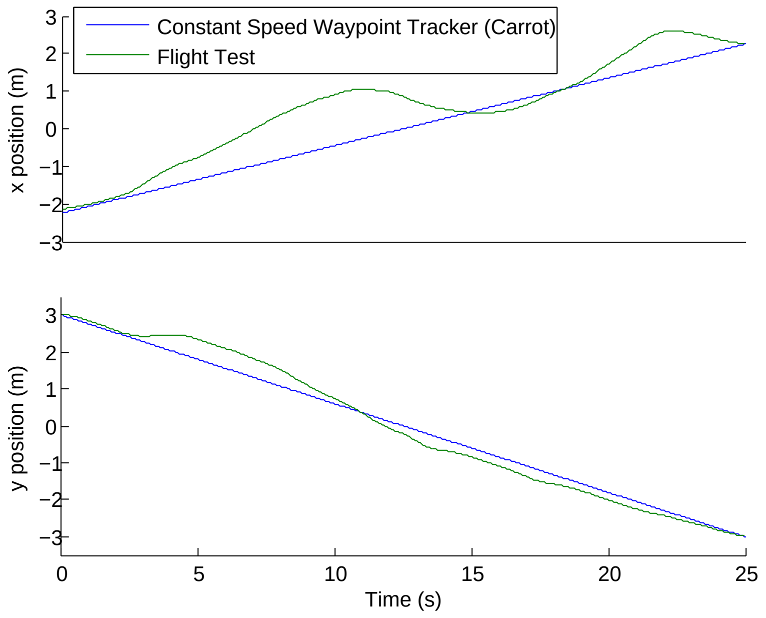


Figure 6.6: Flight test 4, scenario 4 - Waypoint tracker demonstration for vehicle 3. Note that the vehicle and the carrot arrive at the waypoint (2.25,-3) simultaneously. The vehicle modifies its path for collision avoidance by increasing its velocity in the x direction. It then must travel in the negative x-direction for a time in order to arrive at the waypoint at the same time as the carrot. This results in the odd path of the vehicle after deconfliction.

Chapter 7

CONCLUSION AND FUTURE WORK

The DRCA algorithm is successfully implemented on quadrotors at the Boeing VSTL. There are several differences between the DRCA algorithm implemented in the quadrotors and the algorithm developed in [9]. First, the prescribed deconfliction maneuver for the quadrotors is all-turn-left rather than the more complex maneuver described in [9]. Second, since the system at the Boeing VSTL is not set up to output accelerations (controls), a work-around is introduced to convert velocities into controls and vice versa using a new parameter, dt. Unfortunately, this new parameter introduces a delay into the system. Despite these changes, simulations and flight test results alike indicate that the DRCA algorithm is a viable system for collision avoidance.

Nevertheless, some limitations exist when implementing the DRCA algorithm on quadrotors. First, the enclosed environment and relatively small operating space for multiple vehicles nullify the guarantees for the DRCA algorithm outlined in [9]. Second, the capability of quadrotors to hover makes the DRCA algorithm difficult to implement. Deconfliction maintenance makes it impossible for the quadrotors to stop and hover anywhere within the bound because of the zero relative velocity. Although this implementation validates the theory behind the DRCA algorithm, the algorithm is certainly not the best algorithm to use for quadrotors because of these limitations. Perhaps the algorithm could be modified to accommodate the unique features of this particular environment.

There are a number of improvements that can be made upon future implementations of the DRCA algorithm in quadrotors. The actual deconfliction maneuver described in [9] should replace the all-turn-left maneuver. The all-turn-left maneuver is used here as a temporary means to validate other aspects of the DRCA algorithm, such as deconfliction maintenance, function as intended. To increase the effectiveness of the DRCA algorithm on quadrotor systems, the algorithm should be modified to allow the vehicles to hover near each other within the bound. Currently, deconfliction maintenance prevents this sort of behavior. Furthermore, the Boeing VSTL should change its current system so that the vehicles can output accelerations, which will eliminate the need for the dt

parameter and the delay introduced by the current work-around. If desired, the DRCA algorithm can be implemented in three dimensions. Extensions into the third dimension are made in [9].

The “Touch and Go” mission script, designed by the Boeing VSTL, should be used to further investigate the capability of the DRCA algorithm on quadrotors. This script is the current method employed by the Boeing VSTL in evaluating the capability of collision avoidance algorithms. The “Touch and Go” mission script causes an even number of vehicles to take off, trade places, and land. This sequence continues in an asymmetric way, so that the vehicles encounter one another at different points in the flight. For instance, one vehicle may be trying to land at its waypoint while the other vehicle is taking off from that same waypoint. The “Touch and Go” mission script is one tool that can be used to compare the DRCA algorithm with other collision avoidance algorithms.

Further work with the DRCA on quadrotors could include investigating vehicle swarms. Lalish outlines the theory for this sort of behavior and runs a simulation of swarming vehicles in [9]. Finally, the DRCA algorithm should be implemented on various other vehicles to observe behavior. Quadrotors are a unique type of vehicle, and investigating the performance of DRCA on additional platforms would be of great benefit to observing its usefulness. Vehicles of particular interest are fixed-wing aircraft and boats.

BIBLIOGRAPHY

[1] S. Bouabdallah, M. Becker, V. Perrot, and R. Siegwart. Toward obstacle avoidance on quadrotors. In *Proceedings of the XII International Symposium on Dynamic Problems of Mechanics (DINAME 2007)*, 2007.

[2] C. Carbone, U. Ciniglio, F. Corraro, and S. Luongo. A novel 3d geometric algorithm for aircraft autonomous collision avoidance. In *Proceedings of the 45th IEEE Conference on Decision & Control*, December 2006.

[3] A. Chakravarthy and D. Ghose. Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(5):562–574, September 1998.

[4] M. Endsley. Situation awareness, automation & free flight. In *FAA/Eurocontrol Air Traffic Management R&D Seminar.*, June 1997.

[5] W. Green and P. Oh. Optic-flow-based collision avoidance applications using a hybrid mav. *IEEE Robotics and Automation Magazine*, 15(1):96–103, 2008.

[6] G. Hoffman and C. Tomlin. Decentralized cooperative collision avoidance for acceleration constrained vehicles. In *Proceedings of the 47th IEEE Conference on Decision and Control*, December 2008.

[7] J. Kosecka, C. Tomlin, G. Pappas, and S. Sastry. Generation of conflict resolution maneuvers for air traffic management. In *in International Conference on Intelligent Robots and Systems (IROS*, pages 1598–1603, 1997.

[8] J. Kuchar and L. Yang. A review of conflict detection and resolution modeling methods. *IEEE Transactions on Intelligent Transportation Systems*, 1:179–189, 2000.

[9] E. Lalish. *Distributed Reactive Collision Avoidance for Multivehicle Systems*. PhD thesis, University of Washington, 2009.

[10] E. Lalish and K. Morgansen. Distributed reactive collision avoidance for multivehicle systems.

[11] E. Lalish, K. Morgansen, and T. Tsukamaki. Decentralized reactive collision avoidance for multiple unicycle-type vehicles. In *Proc. IEEE American Control Conference*, 2008.

[12] J. Roberts, T. Stirling, J. Zufferey, and D. Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *3rd US-European Competetion and Workshop on Micro Air Vehicle Systems (MAV07) & European Micro Air Vehicle Conference and Flight Competition (EMAV2007)*, September 2007.

[13] E. Saad, J. Vian, G. Clark, and S. Bieniawski. Vehicle swarm rapid prototyphing testbed. In *AIAA Infotech@Aerospace Conference and AIAA Unmanned...Unlimited Conference*, April 2009.

[14] C. Tomlin, G. Pappas, and S. Sastry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43:509–521, 1998.