Creating Shapes in the Sky: Distributed Formation Control of Drones

Andrew Stange and Allison Lo

Carnegie Mellon University

December 4, 2021

Abstract

We investigate distributed formation control of drones, specifically coordinating a swarm of drones to form the outline of an image or shape. We develop a controller that navigates each drone to its final position in the formation in three dimensional space. Given a set of randomly generated starting points for a fixed number of drones, we determine the final location of each drone using a point assignment algorithm for collision avoidance. An event-triggered controller verified in KeYmaera X then directs each drone to its final position. The controller was successfully proved for a triangle formation with three drones. A preprocessing step was used to discretize the outline of an image into the final set of points. Additional models were created for more complex shapes and increased numbers of drones. Due to challenges posed by the large number of branches in KeYmaera X as the number of drones increased, we were unable to complete the proofs for the more complex shapes.

1 Introduction

Utilizing multiple drones, or unmanned aerial vehicles (UAVs), for applications such as search and rescue and delivery of goods allows drones to perform more challenging tasks over larger areas than they would otherwise be unable to perform alone. One application of interest are light shows which rely on the formation control of drones, similar to the 2017 Super Bowl Halftime performance by Lady Gaga which used hundreds of drones to form an American flag, along with other shapes[1]. In this project, we develop a model for the coordination of a swarm of drones to form arbitrary shapes or the outline of images or objects. Each drone must safely move from an initial position to a designated ending position. We first start with arranging three drones in a triangle, and then attempt to increase the number of drones and complexity of the shapes used. Demonstrating the drones can safely form a predetermined shape is necessary for controlling drone swarms of arbitrary size and final shape.

We determine the final location for each drone by optimally assigning points based on a modified version of the Jonker-Volgenant Algorithm [1]. An event-triggered controller verified in KeYmaera X then directs each drone from its initial position to its prescribed final position. It is inefficient to direct one drone at a time to its final destination, so each drone runs its own instance of the controller and moves towards its respective destination simultaneously. Finally, we present an example walkthrough of the entire process, beginning with discretizing the outline of an image to obtain the set of final locations, assigning each drone to its final location, and moving each drone to its final position using the controller in KeYmaera X.

2 Related Work

Formation control has typically been grouped into leader-follower, behavioral, and virtual structure. The leader-follower approach designates drones as leaders or followers, while virtual structure views the formation as a rigid body[2]. An example of a behavioral approach is guiding the drones from initial points to ending points by arranging multiple UAVs in a shape such as a circle or polygon[3]. Each drone moves to the nearest ending position on the shape of interest. Control can also be split into centralized or decentralized control. In centralized control, a ground controller tells the drones how to navigate to their final positions[4]. In a distributed

[1] https://www.therobotreport.com/lady-gaga-300-intel-drones-and-the-super-bowl/

approach, each drone navigates to its assigned position via an individual controller[5]. In this paper, we use behavior-based distributed formation control.

An important factor to consider in formation control is collision avoidance, which ensures each drone can safely travel to its final position. In this paper, we utilize a modified version of the Jonker-Volgenant Algorithm (see Section 3). The assignment problem has been studied extensively in multi-robot task allocation problems[6] and also in more specific settings where communication between robots is limited[7].

We believe that this project is sufficiently novel compared to prior work [2][3] in the Logical Foundations of Cyber-Physical Systems course, due to the use of distributed control for a drone swarm, the development of an collision-avoidance algorithm for a drone swarm, and the novel application to which these technologies will be applied. In this paper, we present a novel approach to distributed formation control of drones where we implement the controller in differential dynamic logic, and formally verify it in KeYmaera X.

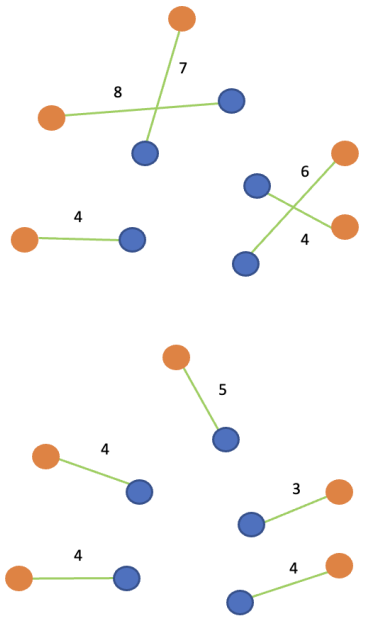

Figure 1: Top: Non-optimal point assignment, with total distance of 29. Bottom: Optimal point assignment, with minimized total distance of 20.

3 Point Assignment

Our distributed controller relies on a careful allocation of image points to drones as a collision avoidance strategy. When considering assignments of drones to image points, if the paths of two drones cross, we can swap the assigned points for these drones, resulting in an assignment of points in which each drone has to travel a shorter distance and there is no potential for drone collision. We consider this new assignment of points more optimal than the original assignment. Intuitively, the problem of assigning image points to drones so no paths intersect can be reformulated as minimizing the sum of path distances. The solution to this point assignment problem can be found by finding a minimum weight matching in a complete bipartite graph. This is a combinatorial optimization problem, specifically the balanced assignment problem. Given a number of agents at arbitrary starting positions and a number of ending positions, we want to minimize the total distance traveled by the agents. We assume the final points are known beforehand. See Figure 1.

The assignment problem can be solved in Python using the `scipy.optimize.linear_sum_assignment` function and is implemented with a modified Jonker-Volgenant algorithm with no initialization[8]. Given an $N_R \cdot N_C$ matrix of costs, where N_R is the number of rows and N_C is the number of columns, the algorithm solves the assignment problem by solving each subproblem when $N_R = 1, 2, ...$ and using the complementary slackness theorem to ensure the globally optimal solution is reached at each step[8].

Since the number of drones and the number of final points is always the same by assumption, the number of points in this assignment problem is always even. Since the number of points is even, the only situation in which this point assignment algorithm will fail is in the case where we have two drones that lie on the same line as both possible destination points. For example, if R_1 is the starting position for drone 1, R_2 is the starting position for drone 2, and points B_1 and B_2 are the possible final positions. Since these points all lie on a line, every assignment of starting points to final positions results in an intersection of paths. [4] This case is avoided in the context of image formation since all final points are on the same plane and no drones start on the image formation plane. Therefore, it cannot be the case that two drones are on the same line

[2]https://lfcps.org/course/lfcps18/projects/jbdurham.pdf
[3]https://lfcps.org/course/lfcps20/projects/aabedon.pdf
[4]https://stackoverflow.com/questions/38094704/connect-an-even-number-of-nodes-without-intersection

as two destination points.

As a result of using this algorithm, it is essential that each drone travels in a straight line from its starting position to its ending position. This condition is necessary to preserve safety. A Python script, `point_assn.py`, implements the point assignment using `scipy.optimize.linear_sum_assignment` and is included as a deliverable. An original goal was to implement the point assignment in KeYmaera X, however, the Jonker-Volgenant algorithm is $O(n^3)$ and KeYmaera X does not support matrix operations or array-style data structures so the algorithm would likely have to be reimplemented for every drone swarm size.

4 Modelling

This section introduces the event-triggered [5] models used to formally verify the distributed drone formation controller. This section will justify modelling decisions after an introduction to the variables used in the triangle model for three drones. Later subsections will motivate and discuss a semantic proof for reducing the complexity of the model and then walk through a proof of the validity of this three drone model.

4.1 Distributed Control

The models included in the deliverables simulate the distributed control of a drone formation using a single model. This counter-intuitive approach is required since if each drone were given its own model, each drone would have to simulate the entire swarm locally and would not be able to exchange position and velocity information with the other drones in the formation in real time. The provided models implement distributed control in a single model by keeping the discrete control of each drone separate from that of the others and by allowing all the drones to share a continuous evolution. With each control loop iteration, all drones make their decisions and then the model allows the entire system to evolve in unison. Through experimentation, it was found that this combined drone evolution is prohibitively complex even for small drone formations of only two drones. As a result, a semantic proof is presented in Section 4.9 that shows that proving safety conditions with a model that separates each drone's continuous evolution implies the safety of the more complex combined evolution model. This semantic proof allows the use of simpler models which scale more efficiently as the size of the drone formations increases. [6]

4.2 Assumptions

We make the following simplifying assumptions on the drones themselves and their physical environment for ease of modelling:

- Independence of control in all dimensions. This is a necessary assumption since the drones must be able to freely move in all dimensions; we cannot restrict physics.
- Knowledge of the exact position and velocity of other drones in the system (perfect sensing and communication). This is a useful assumption for formation control.
- No latency in communication between drones. This is a useful assumption to ensure that formation control is not affected by unstable communication.
- Control of drones is stable (changes in direction will not destabilize the drones). This is a necessary assumption to ensure unstable control will not cause drones to inadvertently collide or prevent them from reaching their final location.
- Assume drones have infinite battery life. This assumption is necessary to simplify modelling. Without this assumption a drone will need to periodically return to the ground.
- No environmental factors, including wind and air resistance. This is a useful assumption since the drones will be operating at slow speeds.
- No downdraft from neighboring drones. This is a necessary assumption because downdraft may cause the drone to destabilize.
- All drones start on the ground and the x-coordinate for the 2D plane where the image will be formed is greater than the initial x-coordinates for all of the drones in the swarm. Each drone's initial position differs from its final position in all dimensions.
- All drones are initially stationary and have an initial velocity of 0.

[5] An event-triggered model was used due to the complexity of the model and the difficulty of proving a time triggered controller that is able to stop a drone within a predetermined range of its final position. An event-triggered controller also leaves the control logic of the controller more apparent, but more difficult to implement in the real world.

[6] See the provided deliverables which provide examples of combined and separate evolution for a model for 2 drones in 2 dimensions.

4.3 Constants and Variables

The constants used in the model are:

- `imagePlaneX` - The x-coordinate of the 2D plane where the image will be formed. The final position for all drones will have their x-coordinate equal to `imagePlaneX`.
- `MAXACCEL` - The maximum acceleration of a drone in a single dimension.
- `MAXBRAKE` - The maximum braking of a drone in a single dimension.

The variables used in the model are listed below, using the variables for drone 1 as an example. The same variables are also used for drones 2 and 3. For each drone, the number in the variable name indicates which drone the variables belong to. For example, `xpos1` and `imageDest1Y` is for drone 1 while `xpos2` and `imageDest2Y` is for drone 2.

- `xpos1, ypos1, zpos1` - The position of the drone in the x-, y-, and z-dimensions, respectively.
- `xvel1, yvel1, zvel1` - The velocity of the drone in the x-, y-, and z-dimensions, respectively.
- `xaccel1, yaccel1, zaccel1` - The acceleration of the drone in the x-, y-, and z-dimensions, respectively.
- `xfinal1, yfinal1, zfinal1` - The final position of the drone in the x-, y-, and z-dimensions, respectively.
- `imageDest1Y, imageDest1Z` - The y- and z-coordinate, respectively, of the destination point.
- `xmult1, ymult1, zmult1` - The acceleration scaling factors in the x-, y-, and z-dimensions, respectively. These are used to ensure the drones travel in straight lines. This follows from the point assignment algorithm.

4.4 Preconditions

The following conditions must be met:

4.4.1 Constant Conditions

- `MAXACCEL > 0`, `MAXBRAKING > 0`
- `imagePlaneX = 0` - The image plane lies at x=0.

4.4.2 Drone Initial Conditions

- `imageDest1Y = 236, imageDest1Z = 201, imageDest2Y = 122, imageDest2Z = 2, imageDest3Y =6, imageDest3Z = 201` - The coordinates of the destination points for each drone determined by the `point_assn.py`. See Section 5.1. These final points for the drones in the `triangle_3_Proof.kyx` model are used as an example.
- `xpos1 = -914.72643558, ypos1 = -34.23173635, zpos1 = 0, xpos2 = -671.81134155, ypos2 = -274.26203647, zpos2 = 0, xpos3 = -488.59935271, ypos3 = -373.29266142, zpos3 = 0` - The coordinates of the starting positions for each drone, randomly generated by `point_assn.py`. See Section 5.1. These initial points for the drones in the `triangle_3_Proof.kyx` model are used as an example.
- `xfinal1 = imagePlaneX, xfinal2 = imagePlaneX, xfinal3 = imagePlaneX` - The final position of each drone in the x-dimension lies on `imagePlaneX`.
- `xvel1 = 0, yvel1 = 0, zvel1 = 0, xvel2 = 0, yvel2 = 0, zvel2 = 0, xvel3 = 0, yvel3 = 0, zvel3 = 0` - The initial velocity of each drone is 0.
- `xpos1 < xfinal1, xpos2 < xfinal2, xpos3 < xfinal3` - The starting position of each drone in the x-dimension is less than the final position of each drone in the x-dimension.
- `yfinal1 = imageDest1Y, yfinal2 = imageDest2Y, yfinal3 = imageDest3Y` - The final position of each drone in the y-dimension is `imageDest*Y`, where `*` represents the number assigned to each drone.
- `zfinal1 = imageDest1Z, zfinal2 = imageDest2Z, zfinal3 = imageDest3Z` - The final position of each drone in the z-dimension is `imageDest*Z`, where `*` represents the number assigned to each drone.
- `ypos1 != yfinal1, ypos2 != yfinal2, ypos3 != yfinal3` - The starting position of each drone in the y-dimension differs from the final position of each drone in the y-dimension.

- `zpos1 < zfinal1, zpos2 < zfinal2, zpos3 < zfinal3` - The starting position of each drone in the z-dimension is less than the final position of each drone in the z-dimension since each drone starts on the ground (`zpos1 = 0, zpos2 = 0, zpos3 = 0`) and a drone's final position cannot be on the ground.

- For each drone (using drone 1 as an example), `ymult1 = (yfinal1 - ypos1) / max(abs(xfinal1 - xpos1), max(abs(yfinal1 - ypos1), zfinal1 - zpos1)), xmult1 = (xfinal1 - xpos1) / max(abs(xfinal1 - xpos1), max(abs(yfinal1 - ypos1), zfinal1 - zpos1)), zmult1 = (zfinal1 - zpos1) / max(abs(xfinal1 - xpos1), max(abs(yfinal1 - ypos1), zfinal1 - zpos1)) - ymult1, xmult1, and zmult1` are found by taking the difference between the drone's final and starting positions in a given dimension, and dividing by the maximum absolute value of the difference between the drone's final and starting positions over each dimension. This is done so the dimension that is furthest from its final point accelerates at a rate of `MAXACCEL` while in all other dimensions the drone accelerates at a fraction of this acceleration. This allows each drone to travel in a straight line while not exceeding its maximum acceleration in any direction.

4.4.3 Drone Safety Conditions

The following safety conditions are presented for drone 1. The same safety conditions are used for drones 2 and 3, using different variables as necessary. See `triangle_3_Proof.kyx`.

- `((ypos1 != ypos2) | (xpos1 != xpos2) | (zpos1 != zpos2))` - Drone 1 has a different initial point than drone 2. This ensures these two drones do not collide initially.

- `((ypos1 != ypos3) | (xpos1 != xpos3) | (zpos1 != zpos3))` - Drone 1 has a different initial point than drone 3. This ensures these two drones do not collide initially.

- `∀xpos1 ∀ypos1 ∀zpos1 ∀x1 ∀x2 ((x1<=xfinal1 & x1>=xpos1 & x2<=xfinal2 & x2>=xpos2) -> (x1=x2 -> ypos1 !=ypos2 | zpos1!=zpos2))` - Drone 1 is never at the same position as drone 2 during its evolution from `xpos1` to `xfinal1`. This safety condition follows directly from the point assignment algorithm.

- `∀xpos1 ∀ypos1 ∀zpos1 ∀x1 ∀x2 ((x1<=xfinal1 & x1>=xpos1 & x2<=xfinal3 & x2>=xpos3) -> (x1=x2 -> ypos1 !=ypos3 | zpos1!=zpos3))` - Drone 1 is never at the same position as drone 3 during its evolution from `xpos1` to `xfinal1`. This safety condition follows directly from the point assignment algorithm.

4.5 Discrete Control

Due to the distributed nature of the drone formation control, each drone has the same discrete controller but with the set of variables specific to that drone. Discrete control for drone 1:

```
if( xvel1^2 / (2*MAXBRAKE) <
                    xfinal1-xpos1 ) {
    xaccel1 := xmult1*MAXACCEL;
}
else {
    xaccel1 := -1*MAXBRAKE;
}
if( yvel1^2 / (2*MAXBRAKE) <
                abs(yfinal1-ypos1) ) {
    yaccel1 := ymult1*MAXACCEL;
}
else {
    if(ypos1 <= yfinal1) {
        yaccel1 := -1*MAXBRAKE;
    }
    else {
        yaccel1 := MAXBRAKE;
    }
}
if( zvel1^2 / (2*MAXBRAKE) <
                    zfinal1-zpos1) {
    zaccel1 := zmult1*MAXACCEL;
}
else {
    zaccel1 := -1*MAXBRAKE;
}
```

The controller controls the evolution of the drone by altering the acceleration of the drone in each dimension. Due to the independence of control in each dimension, the controller sets the acceleration for each dimension independent of the state of the drone in the other dimensions. If the drone is able to brake to a complete stop before reaching its final destination, it will accelerate in that dimension. Else, it will brake. The conditionals used to check this are derived from the kinematic equation $v_f^2 = v_i^2 + 2 * a * d$.

Absolute values are needed in the control for the y-dimension since the initial y position may be greater than or less than the final y position. They are not needed in the x-dimension since all drones start at a negative x position and move to the image formation plane at x=0 so the initial x position is always less than the final x position. Similarly for the z-dimension, drones always start on the ground, z=0, and have a final z position strictly greater than zero.

Note that when assigning a positive acceleration, the discrete controller scales `MAXACCEL` by a fraction. For each drone, acceleration in each direction is scaled so that the dimension with the furthest distance between its initial and final points accelerates at `MAXACCEL` while the other dimensions accelerate by a fraction of this acceleration. This ensures that the drone will reach its final destination simultaneously in all dimensions while also not accelerating at a rate greater than `MAXACCEL`. These fractions force the drone to travel in a straight line between its initial and final points, a requirement for the safety guaranteed by the point assignment algorithm.

4.6 Dynamics

The semantic proof in Section 4.9 allows each drone to have a separate continuous evolution due to the nature of the differential equations and domain constraints used in the model. As was discussed in Section 4.1, these separate continuous evolutions for each drone decreases the model complexity significantly and allows the number of branches needed to prove the model to scale linearly with the number of drones in the model rather than combinatorially.

Within this drone-specific continuous evolution, there are eight differential equations joined by choice operators. These differential equations are needed to maintain fidelity to the physical world by allowing the model to evolve even when it has violated the domain constraint of the initial differential equation. The domain constraint for the first differential equation states that

```
& xvel1^2 / (2*MAXBRAKE) <=
                    xfinal1-xpos1
& yvel1^2 / (2*MAXBRAKE) <=
                    abs(yfinal1-ypos1)
& zvel1^2 / (2*MAXBRAKE) <=
                    zfinal1 - zpos1
```

Intuitively, these three conditions state that the drone is able to come to a complete stop before reaching its final destination in each dimension. This property follows from the kinematic equation, $v_f^2 = v_i^2 + 2 * a * d$. These conditions are needed to stop the continuous evolution of the drone and allow it to begin braking so the controller can guarantee that the drone will not overshoot its final location.

Since there are three domain constraints, we must have 8 differential equations to cover all their combinations. We cannot use a constraint of the form

```
& (xvel1^2 / (2*MAXBRAKE) >=
                    xfinal1-xpos1
| yvel1^2 / (2*MAXBRAKE) >=
                    abs(yfinal1-ypos1)
| zvel1^2 / (2*MAXBRAKE) >=
                    zfinal1 - zpos1)
```

to maintain this physical fidelity since this will fail to detect multiple events. In other words, if $xvel1^2/(2{*}MAXBRAKE) >= xfinal1{-}xpos1$ is already true, this domain constraint will fail to stop the continuous evolution when $yvel1^2/(2 * MAXBRAKE) >= yfinal1 - ypos1$ becomes true and the drone controller will miss this event and overshoot in the y-dimension.

Note the relation between the domain constraints of these differential equations and the conditionals used in the discrete control, discussed in Section 4.5. When the domain constraint $xvel1^2/(2 * MAXBRAKE) <= xfinal1{-}xpos1$ stops the evolution of the drone, the conditional that guards acceleration in the discrete control for that dimension, $xvel1^2/(2 * MAXBRAKE) < xfinal1 - xpos1$ becomes false, causing the drone to brake in that dimension. In this way, the domain constraints and the discrete controller work together to prevent the drone from overshooting its destination in each dimension.

Also note that all differential equations have overlapping domain constraints in order to allow the model to transition between them as needed. All differential equations also have the domain constraint that the drone's z-coordinate is greater than or equal to 0. This constraint is a result of the fact that drones cannot go underground.

The differential equations used in this continuous evolution are very straightforward and model linear motion in each dimension. We are able to model each of these dimensions separately for each drone due to the assumed independence of control in all dimensions for the drones.

Within the continuous evolution for each drone, the differential equations themselves are the same and allow the position and velocity of the drone to evolve in each dimension according to the accelerations selected by the discrete control for each drone.

4.7 Loop Invariant

The following components of the loop invariant are presented for drone 1. A similar invariant is used for drones 2 and 3, using different variables as necessary. See `triangle_3_Proof.kyx`.

- `xvel1`^{2} `/ (2*MAXBRAKE) <= xfinal1 - xpos1` - At velocity `xvel1` and position `xpos1`, the drone can brake and will not pass its final position on the image plane `xfinal1`.
- `yvel1`^{2} `/ (2*MAXBRAKE) <= abs(yfinal1 - ypos1)` - At velocity `yvel1` and position `ypos1`, the drone can brake and not overshoot its final position on the image plane. Since there is no constraint in the y-dimension that `ypos1 < yfinal1`, an absolute value must be used.
- `zvel1`^{2} `/ (2*MAXBRAKE) <= zfinal1 - zpos1)` - At velocity `zvel1` and position `zpos1`, the drone can brake and will not pass its final position `zfinal1`.
- `zpos1 >= 0` - In the z-dimension, the drone is either at or above the ground.
- `∀xpos1 ∀ypos1 ∀zpos1 ∀x1 ∀x2 ((x1<=xfinal1 & x1>=xpos1 & x2<=xfinal2 & x2>=xpos2) -> (x1=x2 -> ypos1 !=ypos2 | zpos1!=zpos2))` - Drone 1 is never at the same position as drone 2 during its evolution from `xpos1` to `xfinal1`. This safety condition follows directly from the point assignment algorithm. algorithm.
- `∀xpos1 ∀ypos1 ∀zpos1 ∀x1 ∀x2 ((x1<=xfinal1 & x1>=xpos1 & x2<=xfinal3 & x2>=xpos3) -> (x1=x2 -> ypos1 !=ypos3 | zpos1!=zpos3))` - Drone 1 is never at the same position as drone 3 during its evolution from `xpos1` to `xfinal1`. This safety condition follows directly from the point assignment algorithm. algorithm.

4.8 Postconditions

4.8.1 Image Formation Condition

The following conditions for image formation are presented for drone 1. Similar conditions are used for drones 2 and 3, using different variables as necessary. See `triangle_3_Proof.kyx`.

- `(xfinal1 = xpos1 -> xvel1 = 0) & (yfinal1 = ypos1 -> yvel1 = 0) & (zfinal1 = zpos1 -> zvel1 = 0)` - When the drone is at its final position on the image plane, its velocity is 0.
- `zpos1 >= 0` - In the z-dimension, the drone is either at or above the ground.

4.8.2 Safety Condition

The following safety conditions are presented for drone 1. Similar conditions are used for drones 2 and 3, using different variables as necessary. See `triangle_3_Proof.kyx`.

- `((ypos1 != ypos2) | (xpos1 != xpos2) | (zpos1 != zpos2))` - For drone 1 and drone 2, the position differs in either the x, y, or z dimension. This ensures a collision does not occur between drones 1 and 2.
- `((ypos1 != ypos3) | (xpos1 != xpos3) | (zpos1 != zpos3))` - For drone 1 and drone 3, the position differs in either the x, y, or z dimension. This ensures a collision does not occur between drones 1 and 3.

4.9 Semantic Proof

4.9.1 Proof Motivation

As was discussed in Section 4.1, this semantic proof is used to demonstrate that a safety proof for a model which separates the continuous evolutions of each drone implies the safety of a model which allows all drones to evolve together. The separation of this continuous evolution allows the monotonicity proof rule to be applied to separate the drones and allow the model to scale more efficiently as the number of drones increases. This proof demonstrates that the differential equations for any pair of drones in the model can be separated, given the set of assumptions below.

4.9.2 Proof Overview

Show $[\{dc1; dc2; plant\}^*]P \leftarrow [\{dc1; plant1; dc2; plant2\}^*]P$ under the following assumptions:

1. $dc1$ and $dc2$ are hybrid programs which perform discrete control decisions. $V(dc1) \cap V(dc2) = \emptyset$.[7] Let $z = V(dc1)$ and $y = V(dc2)$.
2. *plant* is a hybrid program consisting of a differential equation, $[x' = f(x) \& Q_1 \wedge Q_2]P$, in which the set of variables $x = z \cup y$.[8] $V(Q1) \subseteq z$ and $V(Q2) \subseteq y$. Further, let $x' = f(x)$ be a simple differential equation that can be decomposed into functions of z

[7] Define $V(x) = FV(x) \cup BV(x)$.

[8] $\cup$ is a disjoint union.

and y using restrictions on the domain of f.

In other words, $f(k) = \begin{cases} f_{|_z} & k \in z \\ f_{|_y} & k \in y \\ f_{|_{x^C}} & k \notin x \end{cases}$

3. Let $plant1$ be a hybrid program consisting of a differential equation, $V(plant1) \subseteq z$. Let $plant2$ be a hybrid program consisting of a differential equation, $V(plant2) \subseteq y$.

Proof:

1) $[dc1; dc2; plant]P \leftarrow [dc1; dc2; plant1; plant2]P$ (by Section 4.9.3: Plant Separation Proof)

2) $[dc1; dc2; plant]P \leftarrow [dc1; plant1; dc2; plant2]P$ (by Section 4.9.4: Sequential Composition Reorder)

3) $[\{dc1; dc2; plant\}^*]P \leftarrow [\{dc1; dc2; plant1; plant2\}^*]P$ (by the transition semantics of loops in hybrid programs, $[\![\alpha]\!]^* = [\![\alpha^*]\!]$)

4.9.3 Plant Separation Proof

Show $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = [\![z' = g(z)\&Q_1; y' = h(y)\&Q_2]\!]$ assuming:

1. the set of variables $x = z \uplus y$
2. $V(Q1) \subseteq z$ and $V(Q2) \subseteq y$
3. Since x can be decomposed into $z \uplus y$ by the first assumption, the function $f : x \to \mathcal{R}$, can be expressed as $f_{|_z} \uplus f_{|_y} = g \uplus h$.[9] It follows that $V(\varphi_g) \subseteq z$ and $V(\varphi_h) \subseteq y$, where φ_g is the solution for g and φ_h is the solution for h. The solution for f, φ, can be expressed as: $\varphi(k) = \begin{cases} \varphi_g(k) & k \in z \\ \varphi_h(k) & k \in y \\ \varphi_{x^C}(k) & k \notin x \end{cases}$

Proof:

1) $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = \{(w, v) : \varphi(0) =$ w except at x' and $\varphi(r) = v$ for a solution $\varphi : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi| = x' = f(x)\&Q_1 \wedge Q_2\}$

(By the semantics of continuous evolution)

2) $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = \{(w, v) : \varphi(0) = (w_y \uplus w_z)$ except at x' and $\varphi(r) = (v_z \uplus v_y)$ for a solution $\varphi : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi| = x' = f(x)\&Q_1 \wedge Q_2\}$

(Recall that a state w is a mapping from variables to real numbers, $w : x \to \mathcal{R}$. By assumption 1, $w : (z \uplus y) \to \mathcal{R}$. By restricting the domain of w, define the states $w_z = w_{|_z}$ and $w_y = w_{|_y}$ such that $w = w_z \uplus w_y$.[10] Similarly define $v = v_z \uplus v_y$)

3) $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = \{(w, v) : (\varphi_g \uplus \varphi_h \uplus \varphi_{x^C})(0) = w_y \uplus w_z$ except at $(y \uplus z)'$ and $(\varphi_g \uplus \varphi_h \uplus \varphi_{x^C})(r) = (v_z \uplus v_y)$ for a solution $(\varphi_g \uplus \varphi_h \uplus \varphi_{x^C}) : [0, r] \to \mathcal{S}$ of any duration r satisfying $(\varphi_g \uplus \varphi_h \uplus \varphi_{x^C})| = (y \uplus z)' = (g(z) \uplus h(y))\&Q_1 \wedge Q_2\}$

(By assumption 3)

4) $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = \{(w, v) : (\varphi_g \uplus \varphi_h)(0) = w_y \uplus w_z$ except at $(y \uplus z)'$ and $(\varphi_g \uplus \varphi_h)(r) = (v_z \uplus v_y)$ for a solution $(\varphi_g \uplus \varphi_h) : [0, r] \to \mathcal{S}$ of any duration r satisfying $(\varphi_g \uplus \varphi_h)| = (y \uplus z)' = (g(z) \uplus h(y))\&Q_1 \wedge Q_2\}$

(φ_{x^C} is undefined over the domain x by assumption 3. For $k \in x$, $(\varphi_g \uplus \varphi_h \uplus \varphi_{x^C})(k) = (\varphi_g \uplus \varphi_h)(k)$)

5) $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = \{(w, v) : (\varphi_g(0) = w_z$ except at z' and $\varphi_g(r) = v_z$ for a solution $\varphi_g : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi_g| = z' = g(z)\&Q_1 \wedge Q_2) \uplus (\varphi_h(0) = w_y$ except at y' and $\varphi_h(r) = v_y$ for a solution $\varphi_h : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi_h| = y' = h(y)\&Q_1 \wedge Q_2)\}$

(By the definition of set union, assumption 1, and assumption 3)

6) $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = \{(w, v) : (\varphi_g(0) = w_z$ except at z' and $\varphi_g(r) = v_z$ for a solution $\varphi_g : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi_g| = z' = g(z)\&Q_1) \uplus (\varphi_h(0) = w_y$ except at y' and $\varphi_h(r) = v_y$ for a solution $\varphi_h : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi_h| = y' = h(y)\&Q_2)\}$

(By assumption 2, since $V(Q_2) \subseteq y$ and $V(Q_2) \cap z = \emptyset$, the truth value of Q_2 is not altered during the evolution of z' and the domain constraint Q_2 does not restrict the evolution of z' [11]. Similar reasoning for Q_1 and y'.)

7) $[\![x' = f(x)\&Q_1 \wedge Q_2]\!] = \{(w, \mu) : (\varphi_g(0) = w$ except at z' and $\varphi_g(r) = \mu$ for a solution $\varphi_g : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi_g| = z' = g(z)\&Q_1), (\mu, v) : (\varphi_h(0) = \mu$ except at y' and $\varphi_h(r) = v$ for a solution $\varphi_h : [0, r] \to \mathcal{S}$ of any duration r satisfying $\varphi_h| = y' = h(y)\&Q_2)\}$

(Define state $\mu = w_z^{\varphi_g(z)}$, μ has the same values for all variables except for those in set z.

[9] Let the disjoint union of functions be defined as a function that maps from the disjoint union of their domains to the union of their codomains. This disjoint union function maps its inputs to outputs according to the domain the input is in. Since the union is disjoint, only a single function in the union will be defined over that domain.

[10] Since states are functions, the definition of a disjoint unions of states is the same as the definition of a disjoint union of functions.

[11] Definition 2.6, page 51 [9]

This defines a piecewise evolution from w to v in which the variables in y are held constant in the evolution from w to μ and then the variables in z are held constant in the evolution from μ to v. Formally, $V(\varphi_g) \cap y = \emptyset$ and $V(\varphi_h) \cap z = \emptyset$.)

8) $[\![x' = f(x) \& Q_1 \wedge Q_2]\!] = [\![z' = g(z) \& Q_1; y' = h(y) \& Q_2]\!]$
(By the semantics of sequential composition)

4.9.4 Sequential Composition Reorder

Proof of Lemma 1 in Appendix C of "Tactical Contract Composition for Hybrid System Component Verification"[10].

This proof shows: $[\alpha;\beta]\phi \leftrightarrow [\beta;\alpha]\phi$ under the assumption that $BV(\alpha) \cap V(\beta) = \emptyset$ and $BV(\beta) \cap V(\alpha) = \emptyset$. Hybrid programs $dc2$ and $plant1$ meet the requirements for α and β in this proof since $BV(dc2) \cap V(plant1) = \emptyset$ and $BV(plant1) \cap V(dc2) = \emptyset$ since the sets of variables y and z are disjoint.

4.9.5 Proof Applicability Justification

For each of the following assumptions made in this semantic proof, we will demonstrate that any two drones in a given model meet these assumptions. Due to each drone running an independent instance of the discrete control outlined in the model, demonstrating that these assumptions hold for a single pair of drones implies that these assumptions hold for each pair of drones in a more complex model. As a result, we can safely apply this semantic proof to those more complex models in order to simplify their proofs.

1. Assumption 1 holds since the discrete control for each drone relies only on the current position and velocity of the drone and updates the acceleration of only that drone.

2. For assumption 2, consider *plant*, from a two drone, two dimensional model with combined domain constraints for the two drones:

   ```
   xpos1' = xvel1, xvel1' = xaccel1,
   ypos1' = yvel1, yvel1' = yaccel1,
   xpos2' = xvel2, xvel2' = xaccel2,
   ypos2' = yvel2, yvel2' = yaccel2
   xvel1^2 / (2*MAXBRAKE) <=
   xfinal1-xpos1
   yvel1^2 / (2*MAXBRAKE) <=
   abs(yfinal1-ypos1)
   xvel2^2 / (2*MAXBRAKE) <=
   xfinal2-xpos2
   yvel2^2 / (2*MAXBRAKE) <=
   abs(yfinal2-ypos2).
   ```

   All other differential equations in this model are included to maintain fidelity to the physical world. The differential equation in this code segment can be decomposed into two differential equations, one for each drone, such that there are no interdependencies. In other words, restricting the equations relating to drone 1 to variables related to drone 1 and similarly restricting the equations relating to drone 2 to variables related to drone 2 does not alter the solution for this differential equation. The evolution for drone 1 in the x and y dimensions is independent of the evolution of drone 2.

   A similar property holds for the domain constraints. Since the domain constraints for the x and y dimensions for the first drone do not include any variables for the second drone and vice versa, they meet the assumption that $V(Q1) \cap V(Q2) = \emptyset$.

3. For assumption 3, consider the differential equations below. Each only contains variables pertaining to their respective drones and the intersection of their sets of variables is empty.

   *plant*1:

   ```
   {xpos1' = xvel1,
   xvel1' = xaccel1,
   ypos1' = yvel1,
   yvel1' = yaccel1
   & xvel1^2 / (2*MAXBRAKE) <= xfinal1-xpos1
   & yvel1^2 / (2*MAXBRAKE) <= abs(yfinal1-ypos1)
   ```

   *plant*2:

   ```
   {xpos2' = xvel2,
   xvel2' = xaccel2,
   ypos2' = yvel2,
   yvel2' = yaccel2
   & xvel2^2 / (2*MAXBRAKE) <= xfinal2-xpos2
   & yvel2^2 / (2*MAXBRAKE) <= abs(yfinal2-ypos2)
   ```

4.10 Model Proof

Given the model preconditions outlined in Section 4.4, the proof relies heavily on the use of monotonicity to separate drones and on the safety guarantees from the point assignment algorithm.

Following the application of the loop proof rule, the "Init" branch largely proves by id and the braking condition of the loop invariant proves since the initial velocity of the drone is 0 in all dimensions. The "Post" branch

proves by auto due to the aspect of the loop invariant which guarantees that the drone is able to come to a complete stop without passing the image plane and the guarantee from the point assignment algorithm that the paths of two drones never intersects. Proving the "Step" branch requires multiple applications of monotonicity to separate each drone into its own branch. Each of these branches are then unfolded and proved. In each of these single drone branches, the postconditions to be proven can be broken down into safety invariants, image formation invariants, and invariants for other drones. The safety invariants are the conditions of the form: `∀ xpos1 ∀ ypos1 ∀ zpos1 ∀ x1 ∀ x2 (x1<=xfinal1 & x1>=xpos1 & x2<=xfinal2 & x2>=xpos2 -> (x1=x2 -> ypos1!=ypos2|zpos1!=zpos2))` For each drone branch, there is one of these conditions for each of the other drones in the formation. These safety invariants prove by Gödel vacuous since their correctness follows directly from the point assignment algorithm. The image formation invariant is of the form:

```
xvel1^2/(2*MAXBRAKE())<=xfinal1-xpos1
```

This invariant ensures that the drone is able to stop without overshooting its final destination.

The proof for differential equations whose domain constraints are of this same form are straightforward to prove by differential weakening and can be proven by auto. The more difficult branches are those for differential equations whose domain constraints are of the form:

```
xvel1^2/(2*MAXBRAKE())>=xfinal1-xpos1
```

for one or more dimensions.

While auto is generally able to prove these branches, the manual approach is to cut $(((xvel1^2/(2 * MAXBRAKE()) = xfinal1 - xpos1$ into the sequent and then apply a differential cut to make this same formula a domain constraint. This formula is guaranteed to hold since whenever a differential equation of this form occurs, the discrete control for the drone will be braking and the drone never reaches the $(((xvel1^2/(2 * MAXBRAKE()) > xfinal1 - xpos1$ part of the domain constraint.

Finally, the invariants for other drones prove by Gödel vacuous since they either do not mention variables for this drone, or they leave these variables free.

The *auto* tactic is relied on heavily through the proof for these models due to the large number of branches involved in even simple models. While these branches can be proven manually, relying on the *auto* tactic allowed a more time efficient approach to proving these models.

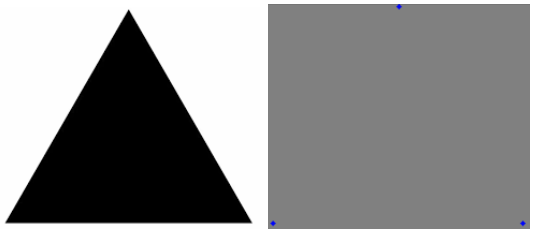

Figure 2: Left: Original image. Right: Discretized image, blue points with grey background.

Due to the semantic proof provided in Section 4.9, the guarantees of the point assignment algorithm, and the ability to reduce the proof of the controller into a number of single drone branches, the proof for two drones is functionally the same as the proof for a model with any larger number of drones. The only difference between the two drone proof and the larger swarm proof is the number of branches and the number of safety invariants to prove for each drone. Similarly, due to the use of the point assignment algorithm, the proof tactics remain the same regardless of the shape being formed by the drones.

5 Example Walkthrough

This section will provide a walkthrough of the pipeline developed in order to discretize images, run the point assignment algorithm, and formally verify that the resulting controller can be executed safely. Specifically, this example will have three drones safely arrange themselves in a triangle.

5.1 Image Discretization and Point Assignment Implementation

Given an image, we first run a preprocessing step which discretizes the outline of the image into points. These points determine the possible final locations for the drones. A Python script, `discretize_image.py`, implements this functionality using `OpenCV`. We approximate the boundary polygon of a shape using `cv2.approxPolyDP`, which uses the Douglas-Peucker algorithm[11].

See Figure 2 for the original image, as well as the discretized image output by `discretize_image.py`. The points output by `discretize_image.py` are then input to the point assignment script, `point_assn.py`, in order to assign each drone its final location.

`point_assn.py` first randomly generates starting points for each drone and then calcu-

lates the optimal point assignment for the drones using the distance between each drone's starting position and each final point. These initial points and final points are then input into the KeYmaera X model and the proof is run. As was discussed in Section 4.10, the proof for any larger number of drones relies on the same tactics used for proving the two drone case so the reader is directed to Section 4.10 for an explanation of the proof tactics used.

The verified triangle model `triangle_3_Proof.kyx` is included as a deliverable. `discretize_image.py` and `point_assn.py` are also included as a deliverable.

5.2 Challenges with Model Complexity

Despite only proving the model for the triangle formation, KeYmaera X models were created for more complex shapes including a pentagon with 5 drones, a circle with 8 drones, and Mickey Mouse ears with 12 drones. These models, as well as the initial images and discretized images for each shape, are included as deliverables. See Figure 3 for the original image of the Mickey Mouse ears[12], as well as the discretized image output by `discretize_image.py`. These models were developed but never proved due to unexpected difficulties with scaling the size of models in KeYmaera X.

Even when using the simplified models resulting from the semantic proof, the size of these larger models pushed the abilities of KeYmaera X even without expanding them into a large number of branches. For example, the Mickey Mouse model is over 2000 lines long. Simply opening these large models, switching between branches in the KeYmaera X user interface, and performing elementary proof tactics such as the loop invariant or *composeb* rule took a minute or more to perform. With large models like the Mickey Mouse model that require proving a thousand or more branches, this performance is prohibitive. It is likely that the safety invariants of the models could be better structured to be less redundant but this does not remove the core of the issue. Even models like the pentagon with five drones would have likely taken significantly longer to prove than the three drone triangle model, since while the number of branches scales roughly linearly with the number of drones, the performance of the KeYmaera X application quickly degrades around 5 drones due to the size of the model.

[12]https://commons.wikimedia.org/wiki/File:Mickey_Mouse_head_and_ears.svg

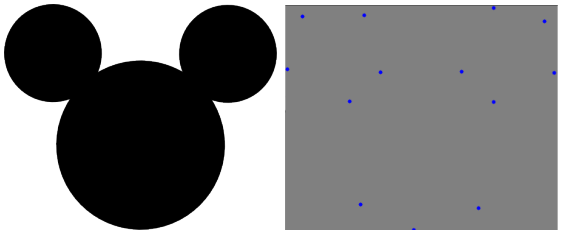

Figure 3: Left: Original image. Right: Discretized image, blue points with grey background.

In order to address some of these performance issues, these larger models were tested on an Amazon Web Services instance [13] as well as on the author's local machine but the performance improvement was not sufficient to allow the proving of these larger models in a reasonable time frame.

6 Discussion

In this project, we learned about collision avoidance algorithms as well as how to construct efficient models and proofs that allow the verification of large and complex models. Throughout this project, we also learned about the limitations of KeYmaera X as the number of drones in the model increased. Despite running into these limitations, we achieved three of the four milestones originally outlined in the project proposal: 1) setting up the environment for one drone, 2) getting two drones to safely arrange themselves in an arbitrary line, and 3) scaling to multiple drones on simple shapes. Due to challenges with model complexity, we were unable to complete the last milestone: scaling to more complex shapes (see Section 5.2).

Another goal outlined in the proposal was to utilize a simulation of the drone swarm to demonstrate the efficacy of our controller in forming shapes and to aid in the development and design of the controller as the swarm size increases. Two simulation programs were considered, Gazebo and AirSim, which are both recommended for Ubuntu. Gazebo is primarily used with additional drone simulator packages, most of which contained instructions for older versions of Ubuntu, before 18.04 LTS. Thus, AirSim was selected. Building Unreal Engine, which is needed for AirSim, took half a day before the virtual hard disk dedicated to the program ran out of space. Due to time constraints, the simulation component was removed from the project.

[13]Tested on an AWS machine with 36 CPUs, 72 GiB RAM, and a 25 Gbps network bandwidth

This project leaves a number of topics open to future work, primarily improvements on model efficiency and on the collision avoidance algorithm. Interesting future work could include proving efficiency proprieties in addition to the safety conditions proved in this project, proving a time-triggered controller rather than the event-triggered one presented in this paper in order to make the ideas in this paper more concrete and realistic to implement, or utilizing simulation to demonstrate the efficacy of the drone formation control. More open-ended future work includes: implementing strong safety conditions that allow non-linear paths for drones between their initial and final points or implementing safety conditions that allow drone paths to intersect without collisions due to the time at which those paths intersect; or working on approaches to reduce the branching factor as the number of drones in the model increases in order to make complex models tractable to prove. A final question for future work would be to implement the point assignment algorithm, a form of combinatorial optimization, in KeYmaera X in order to allow the assignment of drones to final locations to occur at runtime rather than as a preprocessing step.

7 Deliverables

The deliverables for this project include:

7.1 KeYmaera X Models, Initial Images, Discretized Images

7.1.1 Two Drones, Two Dimensions (2_drone_2_dimensions folder)

- `two_drone_combined.kyx`, KeYmaera X model for the 2 drone case in 2D (model not proved). See Section 4.9 for more details.
- `two_drone_independent.kyx`, KeYmaera X model for the 2 drone case in 2D (model not proved). See Section 4.9 for more details.

7.1.2 Triangle (triangle_3 folder)

- `triangle.png`, original image of a triangle
- `discretized_triangle.png`, discretized image of a triangle.
- `triangle_3_Proof.kyx`, KeYmaera X model for the 3 drone, triangle case along with the proof.

7.1.3 Pentagon (pentagon_5 folder)

- `images.png`, original image of a pentagon
- `pentagon_5.kyx`, KeYmaera X model for the 5 drone, pentagon case (model not proved).

7.1.4 Circle (circle_8 folder)

- `circle.png`, original image of a circle.
- `circle_8.kyx`, KeYmaera X model for the 8 drone, circle case (model not proved).

7.1.5 Mickey Mouse(mickey_mouse_12 folder)

- `Mickey_Mouse_head_and_ears.png`, original image of Mickey Mouse ears.
- `Discretized Mickey Mouse.png`, discretized image of Mickey Mouse ears.
- `mickey_mouse.kyx`, KeYmaera X model for the 12 drone, Mickey Mouse case (model not proved).

7.2 Python Scripts

- `point_assn.py` for point assignment.
- `discretize_image.py` to discretize an image. See comments in the script for the different settings and filenames used for each image.

8 Contribution

Andrew Stange

- Model development and proofs
- Semantic proof and model complexity reduction

Allison Lo

- Image discretization
- Point assignment algorithm
- Simulation research

Both authors contributed equally to the white paper, proposal, and final paper.

References

[1] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38:325–340, 1987.

[2] Yi Liu, Junyao Gao, Cunqiu Liu, Fangzhou Zhao, and Jingchao Zhao. Reconfigurable formation control of multi-agents using virtual linkage approach. *Applied Sciences*, 8(7):1109, 2018.

[3] Sunan Huang, Wenbing Cui, Jiawei Cao, and Rodney Swee Teo. Self-organizing formation control of multiple unmanned aerial vehicles. *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, pages 5287 – 5291, 2019.

[4] Godwin Asaamoning, Paulo Mendes, Denis Rosario, and Eduardo Cerqueira. Drone swarms as networked control systems by integration of networking and computing. *Sensors*, 21(8):2642, 2021.

[5] Javier Alonso-Mora, Eduardo Montijano, Tobias Nageli, Otmar Hilliges, Mac Schwager, and Daniela Rus. Distributed multi-robot formation control in dynamic environments. *Autonomous Robots*, 43(5):1079 – 1100, 2019.

[6] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495 – 1512, 2013.

[7] Jingjin Yu, Soon-Jo Chung, and Petros Voulgaris. Target assignment in robotic networks: Distance optimality guarantees and hierarchical strategies. *IEEE Transactions on Automatic Control*, 60(2):327 – 341, 2015.

[8] David F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679 – 1696, 2016.

[9] Andre Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, 2019.

[10] Andreas Müller, Stefan Mitsch, Werner Retschitzegger, Wieland Schwinger, and Andre Platzer. Tactical contract composition for hybrid system component verification. *International Journal on Software Tools for Technology Transfer*, 20:615 – 643, 2018.

[11] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112 – 122, 1973.