UAV Lab 7
Jack Fox, Yeon Kim and Lauren Bakke

EARS Requirement:
When two or more drones come within the minimum separation distance of one another, the drones will be given a vector that is in the complete opposite direction of its original vector (but same magnitude). Once the drones reach a minimum safe distance, their original vectors will be restored.

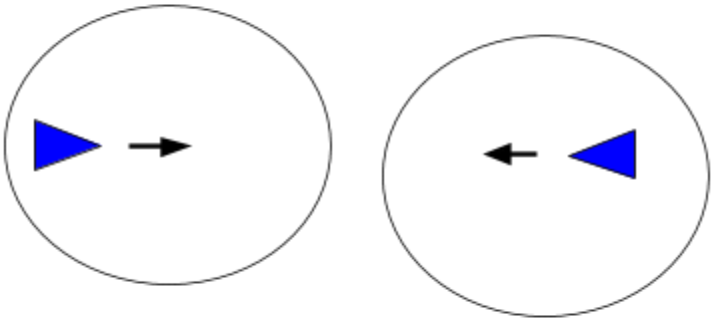
Algorithm Explained:
For this assignment, our algorithm was based on using the minimum-separation distance to push drones away from each other if they were within the minimum distance, and allow drones to continue on their course if they are not within the minimum distance.

We created a Drone class with many methods to accomplish this. Some methods like get_magnitude, get_ned_magnitude, or within_range do simple calculations, and other methods like set_and_send_ned work with the ned_controller to control the drone's speed or direction using NED.

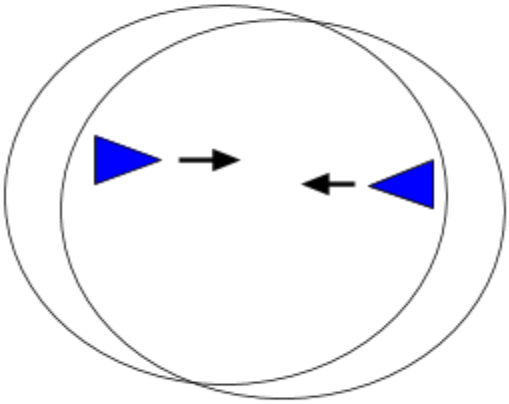
Our algorithm loops for thirty seconds after the drones begin their flight towards each other, continuously calling the avoid_collision method on each drone. This method first checks to see if the current drone is within range of the second drone. If it is in range, the algorithm calls the drone_vector method of the second drone to get its NED. It takes the NED from the second drone, and calculates a new NED that is in the direction of the opposing drone's NED with the magnitude of the current drone. It then adds the current drone's NED with the new combination NED it created, pushing it out of harm's way. If the drone is not in range, it will proceed as usual.

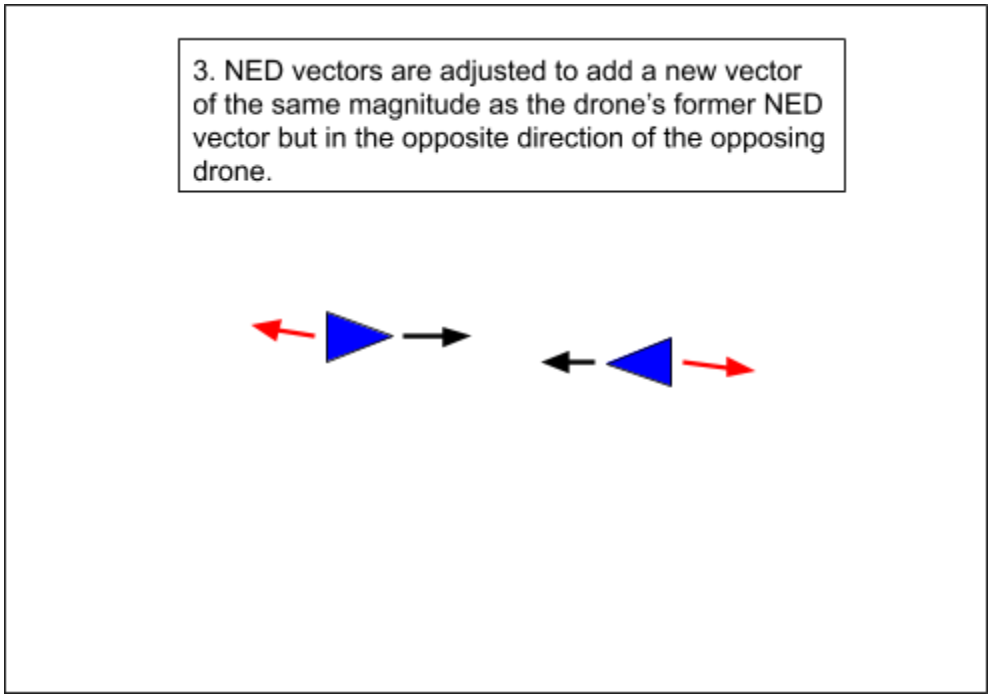
Since this avoid_collision method is being called every loop (i.e., every 0.005 seconds since we use time.sleep to delay them slightly), each drone will either remain on its course if there is no incoming drone, or be assigned an altered NED that is a combination of its original NED and the incoming drone's NED. Since we also store the original NED as an attribute of the drone class, once the incoming drone has reached our minimum separation distance, we set the drone's NED back to what it was before, so that it can continue in the direction of its destination. See the diagrams below for a summarized explanation.

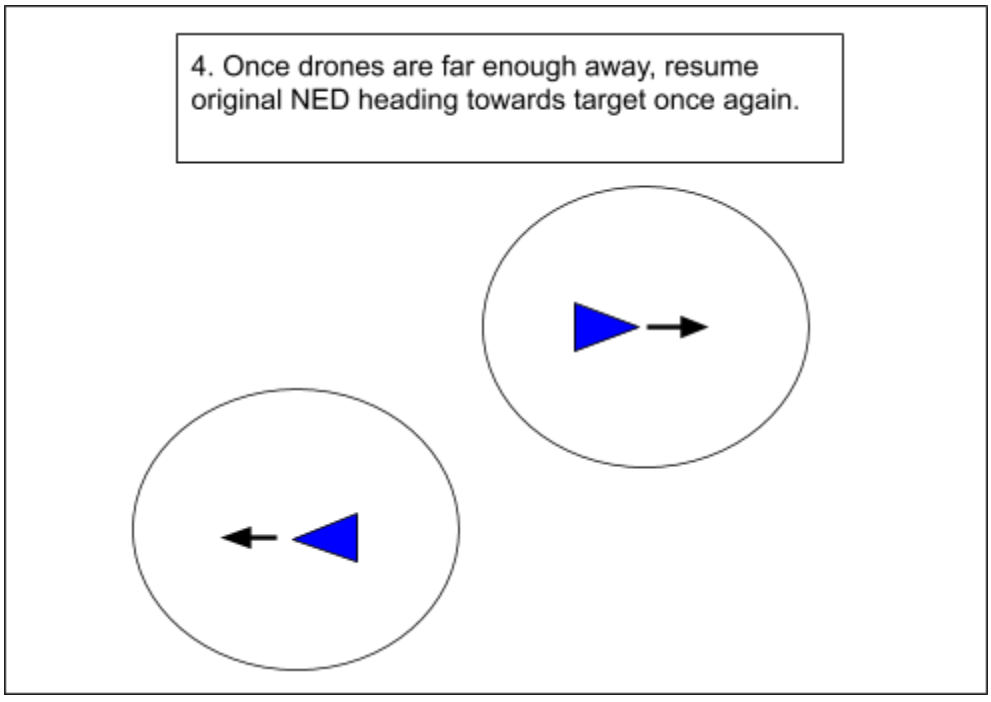
1. Drones are within a safe distance and algorithm determines their NED will go unchanged.



2. Drones are now within the minimum safe distance and the collision avoidance will engage.







Tests:
When conducting acceptance tests we would measure success by checking if the two drones collided. We made this check visually using our animated visualizations. If the dots representing our drones ever overlapped then the test was considered a failure. Tests were of three main types.

1. Two drones approach each other from 40 meters away

This test was conducted at a multitude of speeds. Tests for speeds of up to 3 m/s succeeded, the drones would alter their flight path once within 5 m of the other drone and then return to their original NED without collision. When tested with speeds of greater than 3 m/s it was found that the drones could not correct in time and would collide (tests failed).

2. Two drones fly parallel with a 5 meter distance between them

This test was conducted at a multitude of speeds up to 100 m/s. It was found that for all tests the drones would not collide and they would maintain their original NED (all tests passed).

3. Two drones fly parallel with a 1 meter distance between them

This test was conducted at a multitude of speeds up to 100 m/s. It was found that for all tests the drones would initially push away from each other (until at least a 5 m distance was between them) but they would not collide and they would eventually assume their original NED (all tests passed).

Visualization:
https://youtu.be/CdNpIAPga88

Report:

For this project, our original plan included using a “random nudge” that would nudge the drone slightly in a randomized direction, so that if the drones we approaching one another head on, they wouldn’t get stuck in a deadlock of flying towards each other, moving away and then again flying directly towards each other. However, we didn’t end up needing any kind of nudge, because the random variations of drone flight paths, even in simulation, remove this situation of the drones approaching directly head on (this was tested thoroughly in our unit tests). So, we were able to simplify our code and algorithm by removing that functionality and let the drones do the adjusting on their own.

For the plotting, we used matplotlib. We were stuck for awhile because our code was collecting so many data points while looping that the animation was moving so slowly that we thought the drone wasn’t moving at all, but once we added a short time.sleep(0.005), this reduced the number of data points significantly and made the animation much faster.