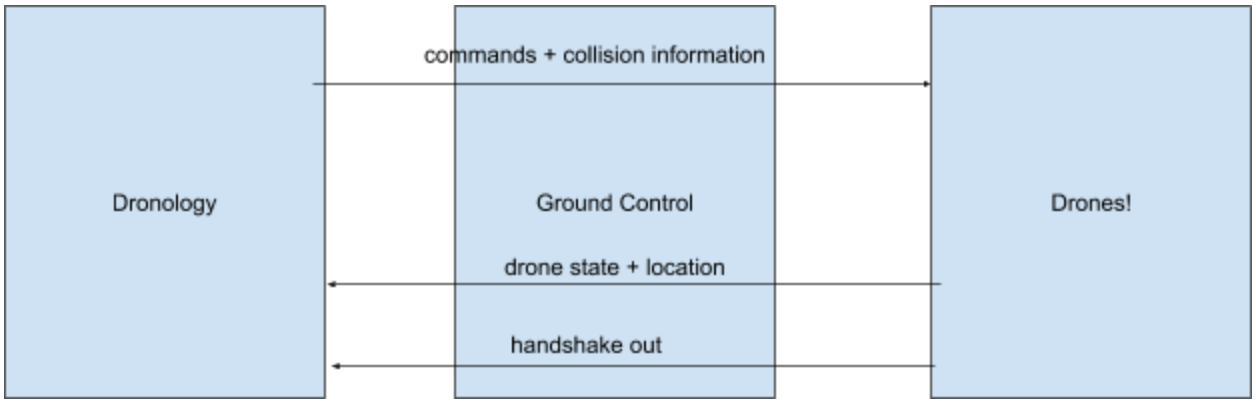
Collision Avoidance: Part 1

Katie Schermerhorn, Emily Obaditch, Luke Brennan

1. **An architectural/design diagram (if you use UML then this would be either a class diagram or a component and connector diagram). However, you can use a box and line diagram if you wish (but minimally show the major elements of your design (classes or modules, key public methods, key data structures).**



2. **A description of your approach. Explain the algorithm you are going to use – either textually, using pseudocode, as a sequence diagram – or however you feel best communicates what you plan to do to achieve collision avoidance.**

The approach chosen involves changing altitudes when drones become in danger of a collision. If a drone senses another drone at the same altitude within 5 meters, then all drones in danger of collision will stop, hover, and one (or more if necessary) will change altitude. The new altitude will be determined by the current altitude of the drone, if it reaches the maximum height threshold then the drone will start to go down in altitude, until it reaches the lowest height threshold. This will ensure that no drone goes up too high or down too low.

For example, if two drones are at the highest altitude and about to collide they will both hover and one will move down 5 meters, to a new zone. This not only allows for collision avoidance but also can scale well because with multiple planes of altitude the drones will have more airspace to fly in. This is a centralized plan due to the fact that each drone is sending its location to home to be checked for collision danger. In order to determine a potential collision

the program will repeatedly check the locations of each drone and calculate the distance between the drones. If two (or more) drones are found to be closer than 5 meters apart the method for changing altitude will be called. By constantly checking the locations of each drone, the program will be able to successfully avoid collisions.

3. **An architectural spike (i.e., code showing that you have started implementation of a "risky" part of your design).**

Our architectural spike for this part of the assignment was setting up a function to run every second that checks the locations of all of the drones. We implemented this by creating a Timer class that takes an interval (in this case 1) and a function, and preforms that function at every interval. The function we have it preform takes all the vehicle information and iterates through the vehicles to get each one's location. We will use this information for our collision avoidance algorithm. We looked at the dronology code to help us with the RepeatedTimer class.

```
72  def get_vehicle_locations(vehicles):
73      for vehicle in vehicles:
74          location = vehicle.location.globalrelativeframe
75          LOCATIONS.append(location)
76
77  class RepeatedTimer(object):
78      def __init__(self, interval, function, *args, **kwargs):
79          self.timer = None
80          self.interval = interval
81          self.function = function
82          self.args = args
83          self.kwargs = kwargs
84
85          self.active = False
86
87          self.start_timer()
88
89      def start_timer(self):
90          if not self.active:
91              self.next = self.interval
92              self.timer = threading.Timer(self.next - time.time(), self.run)
93              self.timer.start_timer()
94              self.active = True
95
96      def run(self):
97          self.active = False
98          self.start_timer()
99          self.function(*self.args, **self.kwargs)
100
101     def stop(self):
102         self.timer.cancel()
103         self.active = False
```