

Exploring Decentralized Swarm Algorithms for Drones


A Technical Report submitted to the Department of Computer Science


Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Joanna Zhao
Spring, 2020

On my honor as a University Student, I have neither given nor received
unauthorized aid on this assignment as defined by the Honor Guidelines
for Thesis-Related Assignments

Signature  Date 2/25/2021
Joanna Zhao

Approved  Date 2/26/21
Sebastian Elbaum, Department of Computer Science

Exploring Decentralized Swarm Algorithms for Drones

Joanna Zhao (Advisor Sebastian Elbaum)

Computer Science, University of Virginia, Charlottesville, Virginia, United States, jz5er@virginia.edu

ABSTRACT

During this research project, I explored decentralized drone swarm algorithms and selected Craig Reynolds' boids model as the basis for implementation. When simulating the implementation, with correct parameters, the drone swarm can consistently navigate to all set destinations while staying apart from each other and not colliding. The swarm also avoids obstacles in general, but can get too close to obstacles on around 4 instances. All work is done in the context of simulators due to Covid-19 and loss of access to the lab.

1 Introduction

A drone swarm is multiple drones acting together to achieve some goal. A swarm of drones provides more capability and can cover larger areas compared to a single drone. They are especially shown in usefulness in the military and for search and rescue.

It is expensive for humans to operate vehicles to conduct search and rescue, and drones are “faster and more efficient in covering large areas” [1]. However, given the need to “monitor a large area of $30 \times 30 \text{ km}^2$ at a distance of 20km from the base (ship or land station)” for a common search and rescue operation, off-the-shelf drones would struggle to complete as it cannot cover an area of more than $10 \times 10 \text{ km}^2$ due to limited battery life of 2h and average speed of 50mph [1]. In addition, it is hard to establish connection between the drone and the base station after a few kilometers to video stream and detect endangered people [1]. Drone swarm in this case not only “enables multi-hop communication network” for long distance connection but also enables “coverage of a larger area in less time”, reducing the battery capacity limitation and increasing efficiency [1].

The first goal of this project is to explore drone swarms and understand the challenges associated with decentralized swarm management. Drone swarms implemented with a distributed system compared with centralized control are resilient and hard to fail, destroy, or manipulate. There are no single points of failure because all the drones are independently controlled instead of being controlled collectively by one computer or drone. Distributed systems also by definition move all the processing that computers do - avoiding obstacles and calculating all the destination and positional controls of drones - to individual drones, which enables the swarm to scale easily. The second goal of this project is to compare existing implementation and algorithms. The third goal of this project is to implement the algorithms in the context of Crazyflie drones in the lab.

However, due to the Covid-19 crisis, I cannot continue with the physical implementation of the swarm algorithms on Crazyflie drones as I do not have access to the research building anymore. The third goal is thus changed to implementing the swarm algorithms solely in the context of simulators. As the parameter space of the algorithm is huge, an additional goal is added to explore the parameter space.

2 Related Work

I explored three swarm algorithms: boids model, ant colony, and particle swarm. While the boids model is a more traditional approach using steering force vectors, ant colony and particle swarm are newer optimization algorithms used for search problems.

The boids model proposed by Craig Reynolds can be found on his website and through his papers “Steering Behaviors For Autonomous Characters” [3] and “Flocks, Herds, and Schools: A Distributed Behavioral Model” [4]. Boids are “generic simulated flocking creatures” designed by Reynolds for computer animation [2]. Each simulated boid is based on a point mass approximation, acts independently, and navigates only based on its changing local neighborhood environment [4]. Each individual boid in the basic flocking model

mainly combines three simple steering forces: separation, alignment, and cohesion [2]. Separation force is the combination of repel vectors from drones within the neighboring drone range [2]. Alignment force directs the drone towards the average direction of the neighboring drones [2]. Cohesion force pulls the drone towards the center of the neighboring drones [2].

In Reynolds' paper "Steering Behaviors For Autonomous Characters", he describes more steering forces such as seek, pursuit, flee, and obstacle avoidance. In "Flocks, Herds, and Schools: A Distributed Behavioral Model", Reynolds continues to describe the distributed model and includes a lot of intuitions and reasonings of the model, including obstacle avoidance reasonings that I later based my obstacle avoidance steering force on.

As described in the paper "Ant Colony Based Path Planning for Swarm Robots", each drone can use optimization algorithms - in this case ant colony but can apply to particle swarm- to plan the shortest route to the destination [5]. To ensure mutually exclusive paths between drones, the routes would be planned sequentially so that later drones when planning would avoid the paths planned by drones previously [5]. This assumes communication of planned paths between drones.

Ant Colony Optimization (ACO) Algorithm mimics the pattern of ants finding the shortest path to food in nature [5]. The algorithm mimics how ants use pheromone levels to communicate with other ants about food sources [5]. The shorter the path, the less time it will take to travel and thus the more likely it is that ants will pick that path again and leave more pheromone [5]. This continuously increases the pheromone level of the shortest path [5]. In addition, pheromone evaporates as time goes on, which discourages the ants to be trapped in suboptimal paths [5].

Particle swarm algorithm can be used for drone swarm in the context mentioned above if it is modified to identify the shortest path to the destination. Particle swarm algorithm "is a population-based stochastic algorithm for optimization" [6]. Particles are randomly initialized in a search space and are moved around to test new parameter values [6]. The next position of the particles is decided using its past position, the best position the particle has ever had, the best position found by the particles' neighbors, parameters, and uniform random numbers [6]. When the algorithm is run, each particle cycles around its previous best and neighbors' best to explore better positions until the update of the variables. After which, the particles switch to their respective new regions in the search space. In the end, the particles will cluster around the optima and the solution is found.

3 Approach

My approach is mainly based on the algorithm proposed by Craig Reynolds. I utilized and modified the seek, separation, cohesion, alignment, and obstacle avoidance steering forces described in the papers: "Steering Behaviors For Autonomous Characters" [3] and "Flocks, Herds, and Schools: A Distributed Behavioral Model" [4]. The parameters used can be found in Table 1 and the pseudocode for the algorithm can be found in Algorithm 1. Note how all the implemented steering forces are only calculated from neighboring drones to ensure the scalability of the algorithm.

In addition to summing up all the force vectors with weights proposed by Reynolds, I decided to normalize all the vectors before the summation and the weighting [3]. This is done for better control of steering forces using weight parameters, preventing extremely large or small steering forces from overwhelming or disappearing. This was inspired by Reynolds' suggestion for normalizing the separation, cohesion, and alignment steering forces before scaling and summing for the simple flocking behavior [3].

However, normalization removes the distance factors in the steering force vectors so I proposed rings of ranges to compensate as shown in Figure 1. Neighboring drones within ring1 will contribute to the repel force vector for the current drone while the drones in between ring2 and ring3 will contribute to the coherence force vector for the current drone. The neighboring drones between ring1 and ring2 along with drones outside of ring3 will have no effect on the current drone. This also enables more control for the entire drone swarm's size as the stable distance between drones are distances between ring1 and ring2. Neighboring drones in that range does not pull or repel the current drone so the whole swarm stays in that state.

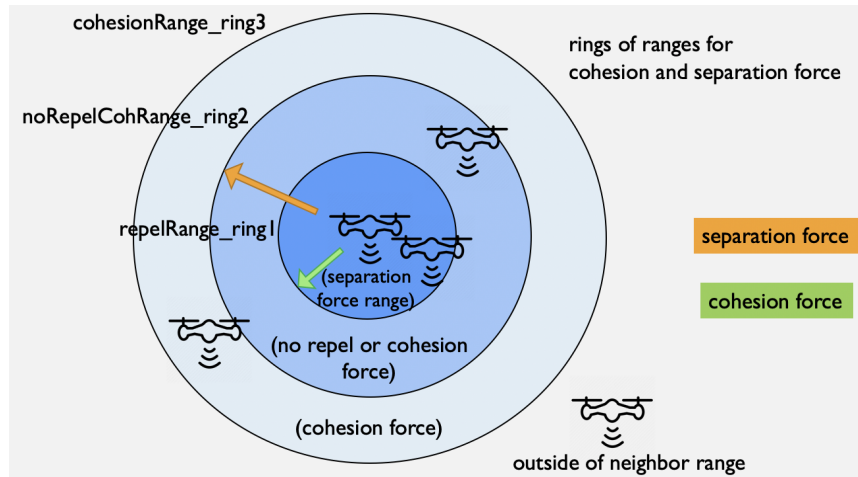


Figure 1: Rings of ranges

In addition, the obstacle avoidance steering force algorithm is significantly modified. Reynolds proposed two ways for obstacle avoidance: force field model and steer-to-avoid [4]. My approach includes some aspects of both ways proposed. Reynolds pointed out that in the force field obstacle avoidance model, boid heading towards the obstacle does not have any side thrusts and that the force tends to be too strong close up and too weak far away [4]. In my approach however, the second problem does not apply as the forces are normalized without rings of ranges for obstacle avoidance. The obstacle avoidance force is thus constantly applied once the drone is in range of the obstacle range. In addition, the obstacle avoidance force is a combination of 2 forces in my approach, one in the repel direction and one in a perpendicular direction, which avoids the drone not having any side thrust when approaching the obstacle directly.

3.1 Parameters

Table 1: All the parameters used in implementation of algorithm

Parameters	Definition
repelRange_ring1	neighbor drones within this range adds a repel force on current drone
noRepelCohRange_ring2	neighbor drones within ring1 to ring2 has no effect on current drone
cohesionRange_ring3	neighbor drones within ring2 to ring3 adds a cohesion/ attraction force on current drone
goalRange	all drones within this range from the goal point has reached that goal
goal_innerCircle_noPull	drones within this range will not have the goal attraction force. This is to help stabilize and decrease collision when all drones are hovering at one goal point
ObstacleRange_perpendicularForce	obstacles within this range from drone will adds a force perpendicular to the repel force of obstacle to drone
ObstacleRange_innerCircle_repel	obstacles within this range from drone will add a repel force from obstacle on drone
minDistLimit	drones within this distance from current drone are considered too close/ collision
obstMinDistLimit	obstacles within this distance from current drone are considered too close/ collision
freq_pub	publish rate per second
waitTimeUntilCountCollision	(in seconds) time that collision's not counted as drone swarm expand from one point
goalWaitTime	once drone reaches new goal, it will keep that goal for this amount of time to wait for other drones to regroup (prevent dragging drones that have not arrived at this goal to skip this goal)
setDroneNum	number of total drones in the swarm
weight_separation	weight for separation force vector when combining all steering forces
weight_cohesion	weight for cohesion force vector when combining all steering forces
weight_goal	weight for goal force vector when combining all steering forces
weight_alignn	weight for alignn force vector when combining all steering forces
weight_obstacles_perpendicular	weight for obstacle perpendicular force vector when combining all steering forces
weight_obstacles_repel	weight for obstacle repel force vector when combining all steering forces
weight_overall	weight that scales the normalized final steering force vector combination

3.2 Algorithm Pseudocode

ALGORITHM 1

```
get all the parameters from launch file
ROS subscribe to all drone positions and velocities
dest: list of destination
obst: list of obstacles
while dest is not empty:
    nextDest = next destination from dest
    while rospy still running:
        if reached goal, keep last goal as previous goal for goalWaitTime temporarily (wait for
        all drones to reach goal before move on)
        initialize vectors: separation_vect, cumOtherDronePos_vect, numtotNeighborDrones,
        cohesion_vect, alignn_vect
        initialize counters: numCohesionRangeDrones, numRepelDrones,
        numtotNeighborDrones
        for each otherDrone beside drone itself:
            tempDist = distance between otherDrone and this drone
            if tempDist > cohesionRange_ring3:
                continue
            elif tempDist > noRepelCohRange_ring2:
                numCohesionRangeDrones +=1
                updateCumOtherDronePosVec(otherDrone,
                cumOtherDronePos_vect)
            elif tempDist > repelRange_ring1:
                continue
            else:
                numRepelDrones += 1
                updateSeparationVec(otherDrone, separation_vect)
            check tempDist to update collision
            if tempDist <= cohesionRange_ring3:
                numtotNeighborDrones +=1
                updateCumOtherDroneVelVec(otherDrone,
                cumOtherDroneVel_vect)
        if numCohesionRangeDrones != 0:
            cohesion_vect = calcCohesionVec(cumOtherDronePos_vect,
            numCohesionRangeDrones )
        if numtotNeighborDrones !=0:
            alignn_vect = calcAlignnVec(cumOtherDroneVel_vect,
            numtotNeighborDrones )
        goal_vect = calcGoalVec(nextDest)
        calcFinalVelVec(separation_vect, cohesion_vect, goal_vect, alignn_vect)
        initialize vectors: obstaclesPerpendicularVel_vect, obstaclesRepelVel_vect
        for each obstacle in obst:
            tempDist = distance between this drone and obstacle
            if tempDist > ObstacleRange_perpendicularForce:
                continue
            updateObstaclesVelVec(obst, obstaclesPerpendicularVel_vect,
            obstaclesRepelVel_vect)
        calcNextPos(obstaclesPerpendicularVel_vect, obstaclesRepelVel_vect)
        publish to drone position input dest_pos that was modified by calcNextPos
    rospy.Rate(freq_pub).sleep
```

The methods used are explained in detail in Appendix B while the complete implementation can be found on GitHub linked in Appendix A.

4 Implementation

The algorithm is implemented using Carl Hildebrandt's point simulator labsim [8] for drones, which uses ROS and is coded in Python. Working with Hildebrandt, we added support for running multiple drones into the simulator. After which, I added the ability to launch any number of drones by passing in an optional parameter when executing the launch file, `fly_path_recursion_base.launch`. `Fly_path_recursion_base.launch` not only takes in the number of drones in the swarm as optional parameter but also optionally takes in any number of parameters mentioned in Table 1 above. `Fly_path_recursion_base.launch` then passes all the parameters to `fly_path_recursion.launch`, which recursively starts drones numbered from `setDroneNum` to 1 and passes all the parameters into each drone's `path_planner` node as seen in Figure 2. Each drone is under their name space `/drone#` and has their own `path_planner` node running the same algorithm individually. `Path_planner` node handles all the control logic for the drone and executes according to Algorithm 1 described in the section above. The only node and topic shared between drones are `/view_node` which visualizes the drones together and `/tf`. To incorporate obstacles, both the path planner node and viewer node would need to get the obstacle information, and path planner node would accordingly use the information to calculate the obstacle force vectors while the view node would display the obstacles.

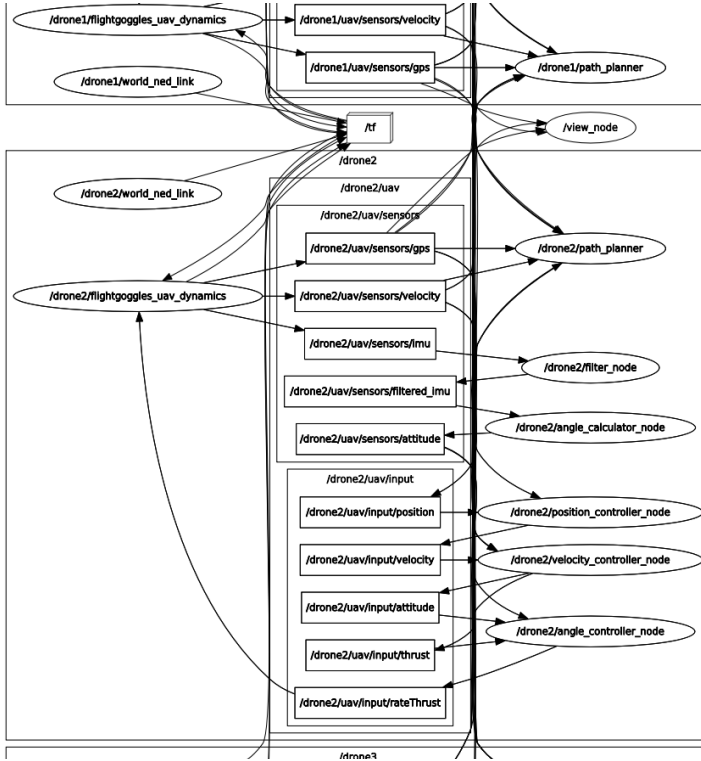


Figure 2: ROS rqt_graph

The simulator can support around 20 drones to execute a preset path. Above 20 drones, the simulator starts to become extremely slow and behaves erratically. The simulator can handle, as an upper limit, around 15 drones when the algorithm for drone swarms described above is added. This limit is not caused by the algorithm or the software, but it is limited by the hardware. This is because the supposed parallel system and algorithm is not actually run on individual drones but is all run on this one machine that simulates by alternating between the processing of all the drones. Thus, the processing becomes increasingly intensive as the number of drones increase and slows down the simulator a lot.

Before Carl Hildebrandt's point simulator labsim [8], I tried KTH Royal Institute of Technology's simulator dd2419 [9], but was having a lot of trouble modifying the simulator to support multiple drones, partly due to the lack of documentation outside of functionalities for running a single drone. After which, I also tried to

modify some other simulators including CrazyS [10], based on RotorS [11], with no success. Sim_cf [12], another simulator that uses RotorS [11] motor model, directly supports multiple drones. However, sim_cf uses Gazebo, and after adding new logic controls, the simulator has trouble supporting the amount of processing required for multiple drones.

4.1 Limitations of Current Implementation

The current obstacle steering force vectors, perpendicular and parallel, are not thoroughly tested and might not perform as well as other force vectors. It is especially affected when size or shape of obstacles are different. In addition, current obstacles are represented as collections of particles (generated through makeObst.py in src/simple_control/src), which is very computationally expensive during simulation for big obstacles.

Weight parameters also do not scale with corresponding range parameters, which can affect performance of the different force vectors because weight for obstacles, for example, might need to increase with decrease of obstacle range to minimize collisions.

Furthermore, to modify obstacles and destination points lists, one has to directly edit the lists in v3.0_mult_drone_preset_path.py in src/simple_control/src and viewer.py in visualizer/src.

5 Study

5.1 Goal of the Study

The purpose of this study is to start exploring the huge parameter space in the algorithm and gain some insights into their effects on the drone swarm.

5.2 Variables

Table 2 lists the independent variables/ parameters and the values explored for each. The values explored were chosen based off of parameter values that worked well in simulation when developing with 10 drones.

Table 2: Independent Variables/ Parameters

Parameters	Values
weight_separation	[40, 60, 100]
weight_obstacles_perpendicular	[50, 120, 200]
weight_obstacles_repel	[0, 40, 120]
weight_goal	[20, 45, 60]
weight_cohesion	[10, 35, 50]

Table 3 lists the parameters that stayed constant.

Table 3: Constant Parameters

Parameters	Value	Parameters	Value
weight_align:	10	weight_overall:	5
repelRange_ring1:	40	goalWaitTime	15
noRepelCohRange_ring2:	60	minDistLimit	2
cohesionRange_ring3:	80	obstMinDistLimit	2
goalRange:	45	freq_pub	20
goal_innerCircle_noPull:	7	setDroneNum	10
ObstacleRange_perpendicularForce:	45	waitTimeUntilCountCollision	7
ObstacleRange_innerCircle_repel:	35		

Table 4 lists the dependent variables

Table 4: Dependent Variables

Dependent Variable	Definition
CloseToOtherDrone	Number of times each drone recorded that another drone's distance is < minDistLimit (note most closeness is double counted because both drones record closeness)
CloseToObstacle	Number of times each drone recorded that another obstacle's distance is < obstMinDistLimit
AllReachDest	(Boolean value) whether all drones have reached all goals
TimeTookToReachAllDest	(in seconds) Time took for all drones to reach all destination (7 minute cutoff time as some successful runs take 2.5 min)

Note the number of times counting closeness in distance happens maximum once every 2 seconds to give time for separation before counting closeness again

5.3 Results

Each set of results from different combinations of parameters is the average from 3 runs and can be found in Appendix D. The correlation values and scatter plots can be found below. (Note: I realized align_vect, whose weight is a constant in the study, was not normalized before combining all steering forces for the runs, which deviates from the described algorithm above and might cause slight differences in parameters to generate the same results)

	Close2OtherDrone	Close2Obst	AllReachDest	timeRchAllDestSec
wt_obst_repel	0.09249	0.207458	-0.003211	0.04173
wt_sep	-0.239658	0.06471	-0.384332	0.552406
wt_obst_perp	0.053448	-0.1367	0.074155	0.024496
wt_goal	-0.078656	-0.372993	0.772699	-0.733573
wt_cohes	0.156481	0.169992	-0.027111	-0.076421
Close2OtherDrone	1	0.250928	0.022979	-0.052778
Close2Obst	0.250928	1	-0.476487	0.460067
AllReachDest	0.022979	-0.476487	1	-0.871938
timeRchAllDestSec	-0.052778	0.460067	-0.871938	1

Figure 3: Correlation Values

The stronger correlation values were highlighted in Figure 3. In particular, `wt_goal` and `wt_sep` seems to have stronger correlation with the dependent variables. It makes sense that the stronger the pull towards the goal, the more likely it is that all drones will reach all goals. The stronger pull also makes the drones reach all destinations faster. It is also interesting to note the inverse relationship between `wt_goal` and `Close2Obst`. This is possibly because stronger pull towards goal means less time hovering around the obstacles, which might result in less collisions. Weight separation also seems to directly relate with time to reach all goals and inversely related to number of times close to other drones. The only correlation that was not expected is the direct relation between `wt_obst_repel` and `Close2Obst`. The more weight for obstacle repel, the less times there should be where drones get close to obstacles. This further supports the need to investigate obstacle forces, particularly the obstacle repel force.

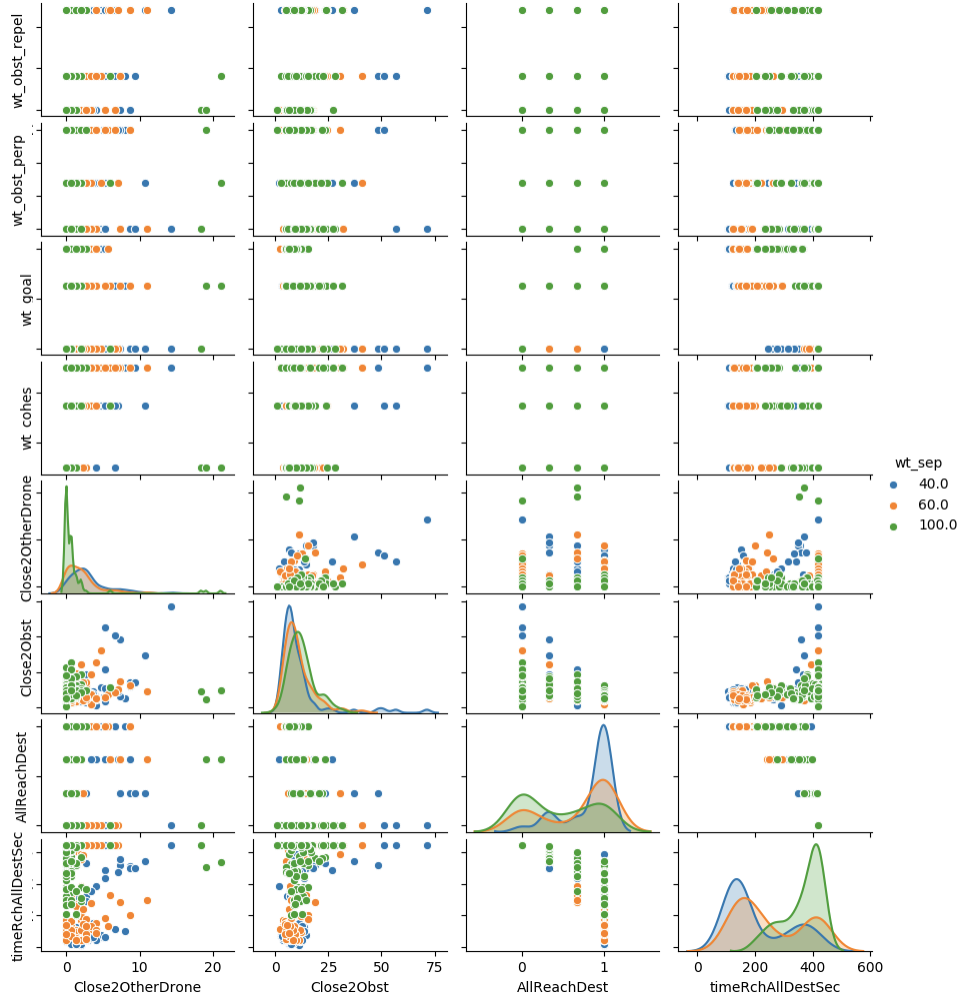


Figure 4: Scatter plot with weight separation as hue

The rest of the scatter plots with different hues can be found in Appendix C.

Figure 4 shows how weight separation combined with another independent variable relates to dependent variables/ results. Through the scatter plot, one can see patterns like how low weight_cohesion combined with high weight_separation can cause outliers and increase in risk of neighbors colliding.

The scatter plot also reinforces the correlations with weight separation. Weight separation is inversely related to number of closeness between drones and directly related to time that it takes to reach all goals, regardless of whether it is combined with another independent variable. The increase in weight to separate the drones was intended to cause less closeness of drones, however, at the same time it can cause the drones to spend more time separating from each other. This might have caused the drones to not directly head towards the goals, resulting in an increase in time to reach all goals.

Key Questions

As the five parameters are very intertwined with each other and due to the difficulty in analyzing a five-way relationship through 5D or 6D graphs, a list of key questions was generated to make more sense of parameter combinations. The key questions can display sets of parameters that produce good or bad results in different aspects.

1. Best configurations to get all drones to the destination in the least amount of time

weight_obst_repel	weight_separation	weight_obst_perpendicular	weight_goal	weight_cohesion
40	40	50	60	35
Results:				
CloseToOtherDrone	CloseToObstacle	AllReachDest	TimeTookToReachAllDestSec	
2	11.33	1	107.6	
weight_obst_repel	weight_separation	weight_obst_perpendicular	weight_goal	weight_cohesion
0	40	50	60	10
Results:				
CloseToOtherDrone	CloseToObstacle	AllReachDest	TimeTookToReachAllDestSec	
2	6.67	1	109.67	

Figure 5: Configurations to get all drones to the destination in the least amount of time

The fastest times for all drones to reach all goals were 108 seconds and 110 seconds, which is less than two minutes. I also displayed the second fastest TimeTookToReachAllDestSec because with just a two second increase in time, number of drones that gets close to the obstacles decreases by almost half. This is likely because weight_obst_repel was not factored in, which was shown to not perform well in the correlation and scatter plot section above. Both sets of parameters have the same weights for separation, obstacle perpendicular, and goal, which seems to be a good combination to generate fast goal seeking times along with pretty low closeness between drones count.

2. Configurations for all drones to be at least minDistLimit apart at all times (CloseToOtherDrone = 0) and reach all goals

weight_obst_repel	weight_separation	weight_obst_perpendicular	weight_goal	weight_cohesion
0	60	50	45	10
40	40	120	45	10
0	60	120	60	10
0	60	50	45	35
40	60	200	60	50
120	100	50	60	10
0	60	120	45	50
0	100	200	60	10
0	100	200	60	50
40	40	50	45	10
40	100	50	60	50
120	100	200	60	35
40	60	50	45	35
40	60	50	60	35
120	100	50	60	35
0	40	50	20	10
120	100	120	60	10
120	100	120	60	50
40	40	50	20	10
0	100	200	60	35
0	100	120	45	10

Figure 6: Configurations that resulted in number of CloseToOtherDrone = 0 and reached all goals

All the above parameters resulted in no collisions and the reaching of all goals in less than seven minutes in all three trials.

- Best configuration for drones to be at least obstMinDistLimit away from obstacles and reach all goals in < 7 min in all 3 runs

weight_obst_repel	weight_separation	weight_obst_perpendicular	weight_goal	weight_cohesion
0	100	200	20	35
Results:				
CloseToOtherDrone	CloseToObstacle	AllReachDest	TimeTookToReachAllDestSec	
3.33	2.33	1	158.9	

Figure 7: Configurations that resulted in least number of drones getting close to obstacle

There were no parameter combinations that resulted in no drones getting close to obstacles. The best parameter combination resulted in a little over 2 collisions on average among 10 drones in the three trials. Note how weight obstacle repel is 0, consistent with previous observations.

- Configurations where drones do not reach destination within seven minutes for all 3 trials

Table 5: Configuration where goals are not reached by all drones in all 3 trials

weight_obst_repel	weight_separation	weight_obst_perpendicular	weight_goal	weight_cohesion
0	100	200	20	35
40	100	120	20	50
0	60	50	20	35
0	100	50	20	50
0	100	120	20	50
40	100	200	20	50
0	100	200	20	50
120	100	50	20	50
120	100	200	20	35
0	60	200	20	35
0	100	200	20	10
40	100	200	20	35
120	100	120	20	35
120	60	200	20	10
40	60	120	20	35
120	60	50	20	35
0	60	200	20	50
0	60	50	20	10
0	100	120	20	35
120	100	200	20	50
40	100	50	20	35
0	100	50	20	10
120	100	50	20	35
120	100	200	20	10
40	100	200	20	10
40	100	120	20	10
120	60	120	20	50
0	60	200	20	10

40	100	120	20	35
40	100	50	45	10
120	100	50	20	10
0	100	50	20	35
40	60	50	20	35
120	100	120	20	50
120	60	120	20	10
40	100	50	45	35
0	100	120	20	10
0	60	50	20	50
120	60	50	20	10
40	60	120	20	10
40	60	50	20	10
40	100	120	45	50
40	100	50	20	50
40	60	200	20	10
40	60	50	20	50
120	100	50	45	35
120	100	50	45	50
120	100	120	20	10
40	60	200	20	35
0	100	50	45	50
40	100	50	20	10
120	100	120	45	50
120	60	50	20	50
40	60	120	20	50
40	40	200	20	35
40	40	50	20	35
120	40	50	20	50

In the configurations of parameters above, all drones did not reach all destinations within seven minutes for all three trials. Those are parameter combinations that should be avoided. There are an overwhelming amount

of weight of goals being 20 in table above, showing that goal of 20 is probably not a good parameter combination with most of the other parameter values used.

Threats to Validity of Findings

Any answers/trends provided above are not guaranteed to be right as they are based on limited amounts of data collected. To be more confident in findings, not only should many more trials be conducted, but a variety of different parameters should also be tested. Although not a perfect depiction, it may also be beneficial to analyze ratios of parameters to provide further insights and boost confidence in results when more trials are performed.

Data Collection

The scripts used can be found in `src/flightcontroller/launch`: `scriptRunLaunchFiles.py`, `killScript.py`, `parserScript.py`, `analysisParserScript.py`. The code can be found in GitHub https://github.com/hildebrandt-carl/Autosoft_Lab/tree/master/Papers/Joanna

`ScriptRunLaunchFiles.py` will run simulations with all the different combinations of weight parameters. Each simulation will write all important information, such as location and time for when goal is reached and when drones get too close to other drones or obstacles, into a text file in sequential order. It will also launch a python subprocess `killScript.py` for each simulation run that kills the simulation when all drones have reached their destinations or when seven minutes have passed.

After the runs, `parserScript.py` and `analysisParserScript.py` can be used to generate tables and graphs from all the text files containing all the important information for each run. `AnalysisParserScript.py` parses and saves all the data into pandas dataframe for easy visualization and manipulation of data. However, note that pip installing Pandas breaks `view.py`'s `draw_artist` function. Virtual environment should be able to solve this issue, though it was not tested.

6 Conclusion

Through this project, I have learned a lot about drone swarms and swarm algorithms. I have learned about the challenges in managing a decentralized swarm system and realized the importance of setting constraints to define problem scope so the project does not get infinitely broad. I realized the importance to first map out the problem as clearly as I can, setting constraints, and then setting stages so I do not tackle and consider all the details of the problem at once.

When researching the swarm algorithms, I was actually also able to use newly learned concepts in my artificial intelligence class like search problems, optimization problems, and local search algorithms. This not only helped my understanding of research papers, but also solidified my knowledge on those concepts and connected them to real examples. Through this research project, I also learned how to find research papers and read them faster and more effectively as I am more familiar with the structure and system.

In addition, I also learned ROS, going from never hearing about ROS to being able to understand and modify several ROS gazebo simulators. Then with all that was learned, I was able to utilize ROS components and functionalities to implement and debug the swarm algorithm. In the process I also realized again the importance to modularize code, even though I thought it was unnecessary in the beginning due to the shortness in length. However, as the code got longer, the modularity provided much more clarity and readability.

Some work still to be done on the simulator includes finding out how to scale parameters with change in number of drones, size and shape of obstacles, and range parameters. Better obstacle representation is also ideal as obstacles are currently represented as particles, which is very computationally intensive for big obstacles. In addition, the obstacle steering force vectors need to be examined for effectiveness especially when dealing with different shaped and sized obstacles.

Furthermore, the algorithm should be implemented on physical drones along with the continued exploration and implementation of other drone swarm algorithms. Data collection and analysis of parameters should also be continued to further understand the effects of the parameters.

REFERENCES

- [1] V. Lomonaco, A. Trotta, M. Ziosi, J. d. D. Y. Ávila, and N. Díaz-Rodríguez, “Intelligent drone swarm for search and rescue operations at sea,” *arXiv preprint arXiv:1811.05291*, 2018. <https://arxiv.org/pdf/1811.05291.pdf>
- [2] C. Reynolds, “Boids.” [Online]. Available: <http://www.red3d.com/cwr/boids/>. [Accessed: 13-May-2020].
- [3] C. W. Reynolds, "Steering Behaviors for Autonomous Characters", *Proc. Game Developers Conf.*, 1999.
- [4] C. W. Reynolds, "Flocks Herds and Schools: A Distributed Behavioral Model", *Computer Graphics*, vol. 21, no. 4, pp. 25-34, 1987.
- [5] A. Agrawal, S. A. P., and A. S., *Ant colony based path planning for swarm robots*: 2015 Conference on Advances In Robotics, July 2015, Article No.: 61, Pages 1–5, <https://doi.org/10.1145/2783449.2783511>
- [6] J. Kennedy, Particle swarm optimization. In *Encyclopedia of Machine Learning*; Springer: Berlin, Germany, 2011; pp. 760–766.
- [7] K.-B. Lee, Y.-J. Kim, and Y.-D. Hong, “Real-Time Swarm Search Method for Real-World Quadcopter Drones,” *Applied Sciences*, vol. 8, no. 7, p. 1169, Jul. 2018.
- [8] C. Hildebrandt, labsim, (2020), GitHub repository, <https://github.com/hildebrandt-carl/labsim> [Accessed: March-2020].
- [9] “Flight camp part 1,” *DD2419 VT19-1 Project Course in Robotics and Autonomous Systems*. [Online]. Available: https://kth.instructure.com/courses/8291/pages/flight-camp-part-1?module_item_id=118606. [Accessed: February-2020].
- [10] G. Silano, CrazyS, (2020), GitHub repository, <https://github.com/gsilano/CrazyS> [Accessed: February-2020].
- [11] Furrer F., Burri M., Achtelik M., Siegwart R. (2016) RotorS—A Modular Gazebo MAV Simulator Framework. In: Koubaa A. (eds) *Robot Operating System (ROS)*. Studies in Computational Intelligence, vol 625. Springer, Cham. Available: http://dx.doi.org/10.1007/978-3-319-26054-9_23 [Accessed: February-2020].
- [12] wuwushrek, sim_cf, (2019), GitHub repository, https://github.com/wuwushrek/sim_cf [Accessed: February-2020].

APPENDIX A

Github: https://github.com/hildebrandt-carl/Autosoft_Lab/tree/master/Papers/Joanna

APPENDIX B

All the methods pseudocode mentioned in the Approach section

note all operations are in terms of x,y,z dimension

note most vectors are part of the class the algorithm is in, so methods modify vectors and do not need to return them

```
updateCumOtherDronePosVec(otherDrone, cumOtherDronePos_vect)
    cumOtherDronePos_vect += position of otherDrone
```

```
updateSeparationVec(otherDrone, separation_vect)
    sep = drone current position - position of other drone
    if sep != 0:
        sep = 1/ sep^3 #further drones will be weighted less in separation direction
        separation_vect += sep #cumulate
```

```
updateCumOtherDroneVelVec(otherDrone, cumOtherDroneVel_vect)
    cumOtherDroneVel_vect += normalized other Drone velocity
```

```
calcCohesionVec(cumOtherDronePos_vect, numCohesionRangeDrones)
    initialize cohesion_vect
    cohesion_vect = cumOtherDronePos_vect/ numCohesionRangeDrones - current drone position
```

```

    return cohesion_vect

calcAlignnVec(cumOtherDroneVel_vect, numtotNeighborDrones)
    initialize alignn_vect
    alignn_vect = cumOtherDroneVel_vect/ numtotNeighborDrones
    return alignn_vect

calcGoalVec(nextDest)
    initialize goal_vect
    if distance between current drone and nextDest < goal_innerCircle_noPull:
        return goal_vect
    goal_vect = nextDest - current drone position
    return goal_vect

calcFinalVelVect(separation_vect, cohesion_vect, goal_vect, alignn_vect)
    normalize separation_vect, cohesion_vect, goal_vect, alignn_vect
    return sum(weight_sep*separation_vect, weight_coh*cohesion_vect, weight_goal* goal_vect,
    weight_alignn*alignn_vect)

updateObstaclesVelVec(obst, obstaclesPerpendicularVel_vect, obstaclesRepelVel_vect)
    if moving away from obstacle, return immediately, do not follow obstacle avoidance path (angle
    between the vector from calcFinalVelVect and the vector pointing from obst to drone < 50)
    tempObst = current drone position - obst
    if tempObst !=0:
        tempObst = 1/tempObst^3 #obstacle further away have less weight in obstacle avoidance force
        obstaclesPerpendicularVel_vect += One of the Perpendicular Directions of Obst vect
        if distance between drone and obstacle is < ObstacleRange_innerCircle_repel:
            obstaclesRepelVel_vect += -tempObst

calcNextPos(obstaclesPerpendicularVel_vect, obstaclesRepelVel_vect)
    normalize obstaclesPerpendicularVel_vect and obstaclesRepelVel_vect
    finalVel_vect += weightObstPerpendicular*obstaclesPerpendicularVel_vect + weightObstRepel+
    obstaclesRepelVel_vect
    normalize finalVel_vect
    dest_pos = current position of drone + weight_overall * finalVel_vect

```

APPENDIX C

Scatter plot with weight cohesion as hue



Scatter plot with weight goal as hue



Scatter plot with weight obstacle repel as hue



APPENDIX D

Averaged Results from Study

weight_obstacle_repel	weight_separation	weight_obstacles_perpendicular	weight_goal	weight_cohesion	CloseToOtherDrone	CloseToObstacle	AllReachDest	TimeToReachAllDestSec
0	40	120	20	35	4	1.67	0.67	291.77
0	100	120	20	50	0	5.33	0	420
40	40	200	60	35	5	6.67	1	132.73

0	60	120	20	10	1.33	6	0.33	405.43
40	60	200	60	10	1.33	4	1	152.87
40	100	120	45	10	21	11.67	0.67	369.27
0	60	50	45	10	0	3.67	1	184.33
0	60	120	20	35	0	5	0.67	371.53
120	60	50	60	50	1.33	8	1	137.33
40	40	50	60	35	2	11.33	1	107.6
120	60	50	45	10	1.33	10	1	235.5
120	100	50	20	10	0	14.67	0	420
0	100	200	45	50	0.67	10.33	0.67	338.6
0	60	200	45	35	2	5.67	1	186.27
40	60	50	45	50	7.33	18.67	0.67	241.93
0	40	120	20	50	2.67	5.67	1	261
120	40	50	60	10	2	11	1	126.13
120	60	120	20	50	7	12.67	0	420
0	100	120	20	35	0	11.33	0	420
0	60	200	45	10	2	8.33	1	291.77
40	100	200	20	10	0	12.67	0	420
0	40	50	45	10	2.67	3.67	1	122.97
0	60	200	60	10	0.67	3.67	1	166.73
0	40	120	45	50	2	4.33	1	137
120	100	200	45	50	2	23.33	0.67	382.2
40	60	120	20	50	4.67	40.67	0	420
0	40	200	60	50	0.67	5.33	1	136.9

40	100	200	45	10	0	6.33	0.67	393.3
120	100	120	60	10	0	10.67	1	290.6
40	100	50	45	35	0.67	17	0	420
0	60	50	45	50	1.33	10.67	1	153.2
120	40	200	20	50	8.67	17.67	0.67	355.57
0	60	120	60	35	2.67	5	1	139.37
0	40	200	20	10	0	10.67	0.33	409.8
40	40	120	20	10	0	8.67	0.33	383.9
120	60	120	60	35	1.33	12.33	1	148.67
40	100	120	20	10	0.67	12.67	0	420
120	100	120	20	50	2	16.33	0	420
0	100	120	60	35	0.67	9	1	273.57
40	100	200	60	10	0.67	7.67	1	365.73
40	60	120	45	50	0.67	4.67	1	182.87
0	100	200	60	35	0	13.33	1	261.9
0	60	120	45	10	0	7	0.67	293.33
40	60	120	60	50	0.67	9	1	139.43
120	100	120	45	35	1.33	15.67	1	359.2
120	100	120	20	10	0.67	24.33	0	420
120	40	120	45	10	1.33	5	1	164.4
40	40	50	45	10	0	9	1	121.1
40	60	120	20	10	0	21.33	0	420
120	60	200	60	50	5.67	9	1	168.17
0	100	120	20	10	0	17.33	0	420

120	40	120	20	50	5.33	26.67	0.67	342.83
40	40	200	20	10	0.67	13.67	0.33	400.63
120	60	200	45	35	4	5.67	1	176.03
120	60	50	60	35	1.33	10	1	128.5
0	40	50	20	50	8.67	14.67	0.33	348.77
40	60	120	45	10	0.67	8.33	1	257.63
120	60	200	20	10	0	10.33	0	420
120	60	200	20	50	1.33	16.67	0.33	400.9
120	100	200	60	10	0.67	8.67	1	338.87
120	60	120	45	35	2	12	1	199.3
120	60	120	60	50	2.67	9	1	152.27
40	60	200	60	50	0	5	1	152.27
40	60	50	20	10	0	21.67	0	420
0	40	50	45	35	3	5.33	1	121.6
40	40	120	60	10	2.33	7.33	1	127.03
0	60	50	20	35	0	5	0	420
120	40	50	45	50	2.67	13.67	1	164.03
40	40	200	60	10	4	8	1	134.8
120	40	120	45	50	2	6.33	1	152.73
120	40	200	20	10	0.67	7	0.67	381.4
120	100	50	20	50	0	6	0	420
0	40	120	45	10	2.67	4.67	1	135.4
40	60	50	45	10	1.33	8.33	1	230.1
0	60	200	60	35	2.67	5.67	1	158.1

0	40	120	20	10	0.67	9.33	0.33	392.53
120	60	50	20	50	4	32	0	420
120	100	200	20	35	0	6.33	0	420
120	40	120	45	35	3.67	11.33	1	177.6
0	60	120	60	50	0.67	6.33	1	143.23
40	60	200	45	10	0.67	9.33	1	261.3
120	60	50	20	35	1.33	10.33	0	420
120	60	50	60	10	2	11.67	1	154.73
120	60	120	20	35	0.67	13.67	0.33	403
40	40	200	60	50	2.67	3.67	1	137.6
120	40	50	20	10	0	10	0.33	395.93
40	40	200	45	50	8	6.33	1	151.97
120	40	200	45	50	2.67	12.67	1	165.67
0	40	200	45	10	2.67	4.67	1	145.9
0	40	200	60	35	3.67	6	1	143.1
40	40	200	20	50	7.33	48.33	0.33	358.7
40	40	50	20	10	0	12.67	1	340.2
0	40	50	60	10	2	6.67	1	109.67
0	100	120	45	35	0.67	11.33	0.33	385.93
120	60	120	45	50	6	14.33	0.67	263.07
40	60	200	20	35	1.33	24.33	0	420
120	40	50	60	35	4.67	14.33	1	139
120	40	200	45	35	3.33	6.67	1	160.37
120	60	200	45	50	8.67	15.33	1	200.57

40	60	50	20	50	3.33	23.33	0	420
40	40	50	45	50	4	6.67	1	123.6
0	100	50	60	10	0	7.67	0.67	362.87
0	60	200	45	50	5.33	7.33	1	190.97
0	100	50	20	10	18.33	11.33	0	420
40	100	50	60	35	0.67	13.67	0.67	322.97
40	100	120	45	35	0	18.67	0.67	379.8
40	100	50	20	10	0	28.33	0	420
0	100	50	45	35	0.67	16.67	0.33	403.87
40	60	50	60	10	2	6.67	1	124.67
120	60	200	45	10	2	7.33	1	248.8
120	100	200	45	35	1.33	13	0.67	364.4
0	60	50	60	50	2	11	1	122.2
0	100	200	60	50	0	8.67	1	248.77
0	40	120	60	35	2.67	4.33	1	127.83
120	40	50	45	10	2	8	1	139.03
40	40	50	20	35	5.33	56.67	0	420
0	100	120	45	10	0	15.67	1	372.1
120	40	120	20	10	0	18.67	0.67	358.63
120	40	200	60	50	5.33	4	1	131.27
120	100	50	45	35	0.67	24	0	420
40	40	50	45	35	1.33	11.67	1	120.73
0	60	200	60	50	3.33	2.33	1	158.9
40	60	200	20	10	0.67	22.67	0	420

40	60	120	45	35	3.33	8.67	1	165.47
120	100	200	20	50	0	11.33	0	420
120	100	120	60	35	2	13.33	1	290.43
0	100	200	45	35	0	17	0.33	399.73
40	60	50	60	50	2.67	9.33	1	127.63
0	40	200	45	50	2.67	7.33	1	145.4
40	100	200	45	50	0	22	0.33	381.77
0	40	200	20	35	1.33	7	1	259.7
120	60	120	45	10	2.33	6.33	1	220.33
120	40	120	60	50	3.33	9.67	1	129
0	100	50	20	50	0	5	0	420
0	100	120	60	10	0	9	0.67	364.67
120	100	120	45	10	0.67	7.67	0.33	405.43
0	40	200	45	35	2.67	3.33	1	159.97
120	100	50	45	50	0.67	24	0	420
0	60	200	20	10	0	14	0	420
120	100	120	20	35	0	10	0	420
40	100	50	20	35	0.67	11.33	0	420
40	40	50	60	10	0.67	7.33	1	110.8
0	60	120	20	50	2	14	0.33	376.87
40	60	50	20	35	2.67	16	0	420
40	100	120	20	50	0	3	0	420
120	60	200	60	35	4	6.33	1	153.03
0	40	50	45	50	1.33	7.33	1	124.47

40	60	50	45	35	0	9.33	1	141.93
40	60	50	60	35	0	9.67	1	122
0	100	200	45	10	19	5.33	0.67	352.6
40	100	120	20	35	6	14	0	420
0	100	50	45	10	0	16.33	0.33	409.33
0	100	50	20	35	0	15	0	420
40	40	120	60	35	1.33	6.33	1	123.2
0	40	120	45	35	1	3.33	1	133.33
40	60	120	60	35	2.67	9.33	1	140.63
120	40	120	20	35	10.67	37	0.33	371.1
120	100	200	45	10	0	13.33	0.67	399.03
0	40	50	20	35	3.33	7	0.67	275.57
0	40	50	60	50	1.33	8.33	1	112.27
40	40	120	20	35	2.67	10.67	1	245.27
120	100	120	60	50	0	12.67	1	203.37
0	100	200	20	50	0	5.67	0	420
0	60	50	60	10	2.67	5.33	1	127.63
120	100	120	45	50	0.67	31.67	0	420
0	60	50	45	35	0	5	1	151.43
0	40	200	60	10	2	4.67	1	141.53
40	100	200	60	50	2	11.67	1	244.1
120	60	50	20	10	0.67	18.33	0	420
40	100	120	45	50	1.33	21.67	0	420
120	40	200	60	35	3.33	2.67	1	134.5

40	40	120	20	50	2.67	23.67	0.33	365.8
0	60	50	20	10	0	10.67	0	420
120	100	50	45	10	1.33	12	0.33	414.13
40	40	50	20	50	9.33	17.67	0.33	348.5
0	60	200	20	50	6.67	10.33	0	420
0	40	50	20	10	0	9.67	1	393.33
120	60	50	45	35	2.67	15.33	1	188.03
40	60	200	45	35	3.33	9.33	1	237.77
40	100	200	20	50	0	5.33	0	420
120	60	120	60	10	0.67	8.33	1	154.47
0	100	200	20	35	0	0.67	0	420
40	40	200	45	35	0.67	7.67	1	153.2
120	60	200	60	10	2.33	4.67	1	171.63
40	100	50	45	10	0	14.33	0	420
40	100	200	20	35	0.67	9.67	0	420
120	40	120	60	10	1.33	7	1	131.37
40	100	120	60	50	1.33	7.33	1	201.83
40	100	200	45	35	0	8.33	0.33	416.87
0	40	120	60	10	2.67	3.67	1	130.73
120	100	50	60	35	0	9.67	1	283.67
120	40	200	45	10	6.67	10.67	1	157.17
40	40	120	45	50	2	10.67	1	140
40	40	200	45	10	2	4	1	145.47
120	60	50	45	50	11	11.33	0.67	249.17

0	40	50	60	35	1.33	7	1	110.77
40	100	200	60	35	0.67	14.33	1	247.73
40	60	120	60	10	0.67	6.67	1	136.4
40	40	120	45	35	0.67	3.33	1	136.63
0	40	120	60	50	1.33	6	1	123.3
0	60	50	60	35	1.33	8.33	1	122.07
120	60	120	20	10	0	16.67	0	420
0	100	200	20	10	0	7.67	0	420
0	40	200	20	50	7.33	7.67	0.33	375.83
120	100	200	20	10	0	12.33	0	420
40	60	120	20	35	0	10.33	0	420
40	100	50	45	50	0.67	20.67	0.33	370.43
0	60	50	20	50	2.67	17.67	0	420
40	40	50	60	50	1.33	7.33	1	110.93
40	100	120	60	35	0.67	15.67	1	290.27
120	40	120	60	35	3.67	9	1	132.53
40	60	200	45	50	1.33	8.33	1	206.07
120	100	200	60	50	2.67	12	1	282.67
120	40	200	20	35	7	11.33	0.67	336.5
0	100	50	60	50	1.33	7.67	0.67	277.1
120	100	50	60	10	0	5.33	1	336.93
120	40	200	60	10	2.67	7.33	1	148
0	60	120	45	35	2.67	7	1	167.63
0	60	120	45	50	0	5.67	1	169.33

120	100	50	60	50	2	10.33	1	255.67
120	100	50	20	35	0	12	0	420
40	60	200	20	50	2	30.67	0.33	393.2
40	40	120	45	10	0	4.33	1	154.5
120	100	200	60	35	0	9	1	311.93
0	100	50	45	50	0.67	27	0	420
40	100	50	60	10	0.67	6	1	334.1
0	60	120	60	10	0	4.67	1	141.27
40	40	120	60	50	1.33	7	1	124.1
40	100	50	60	50	0	9	1	233.7
40	100	50	20	50	0	22.67	0	420
120	60	200	20	35	2.33	17.33	0.33	388.83
120	40	50	45	35	2.67	13.67	1	151.77
120	40	50	20	35	5.33	13.67	0.67	316.1
0	100	50	60	35	0.67	9.33	1	235.03
40	100	120	60	10	0	10	0.67	324.07
0	100	120	60	50	1.33	8.67	1	205.57
120	40	50	20	50	14.33	71.33	0	420
0	100	120	45	50	0.67	11.67	0.33	370.37
40	60	200	60	35	4	6.33	1	145.47
0	60	200	20	35	0.67	6.33	0	420
40	40	200	20	35	6.67	51	0	420
120	40	50	60	50	2.67	12.67	1	122.6
0	100	200	60	10	0	6.33	1	332.5