Gliding in Space Experiment

Author: Jiangshan Luo

Student ID: U6488845

Course: COMP2310

Submission Date : 2017-09-28

Contents

1. Problem Statement..... 1

1.1 Globe Detection..... 1

1.2 Recharging Strategies..... 1

1.3 Performance Improvement..... 2

2. Design..... 2

2.1 Share the Information..... 2

2.2 Methods for Recharge..... 6

2.3 Abandoned Designs..... 9

3. Implementation..... 12

3.1 Message Passing..... 12

3.2 Recharge..... 14

3.3 Others..... 15

4. Analysis & Evaluation..... 15

1. Problem Statement

1.1 Globe Detection

The first problem is the globe detection. For a single vehicle, it needs to find the energy globes so that it could know where to get recharged. In terms of a swarm of vehicles, however, it's impossible for all the members to detect the globes successfully on their own, which points out another question about information sharing. On this question, I need to figure out how to inform others when one vehicle finds the globes, using shared memory or message passing. And a question follows that is how to implement that in high efficiency, because only by getting newest information in time could vehicles find actual globe positions and go to the right place to recharge when they need to. If the information sharing strategies are inefficient, vehicles will probably take the risk of always getting outdated message and could never find the current globe positions.

1.2 Recharging Strategies

In the second place, the strategies for recharging are also needed. As the Assignment Paper mentioned, "destinations might become unreachable, if multiple vehicles are bound for the same destination". As a result, control management and constraints on vehicles should be implanted to prevent a large number of objects from trying to reach a globe at approximately the

same time, even though the collision avoidance is provided.

1.3 Performance Improvement

Finally, to improve the performance, many other strategies could be deployed. For instance, some solutions that could lower the possibility for an individual or a group of vehicle(s) to lose contact with majority, and ensure the useful information could always be conveyed. Other solutions that are able to help individual to make estimation about the future position of the globes.

2. Design

2.1 Share the Information

My first attempt is shared memory based communication. Compared with message passing, this method has more efficiency on conveying information. Once a individual find the energy globes surrounding it, it could save the message to the shared memory and then everyone know that, which is easy to keep the information updated and avoid massive communications among vehicles (Figure 2.1.1). By using this method, all my vehicles were surviving successfully in Stage A. However, the crucial problem is that it seems to be impossible for the physical deployment of the distributed system in most cases.

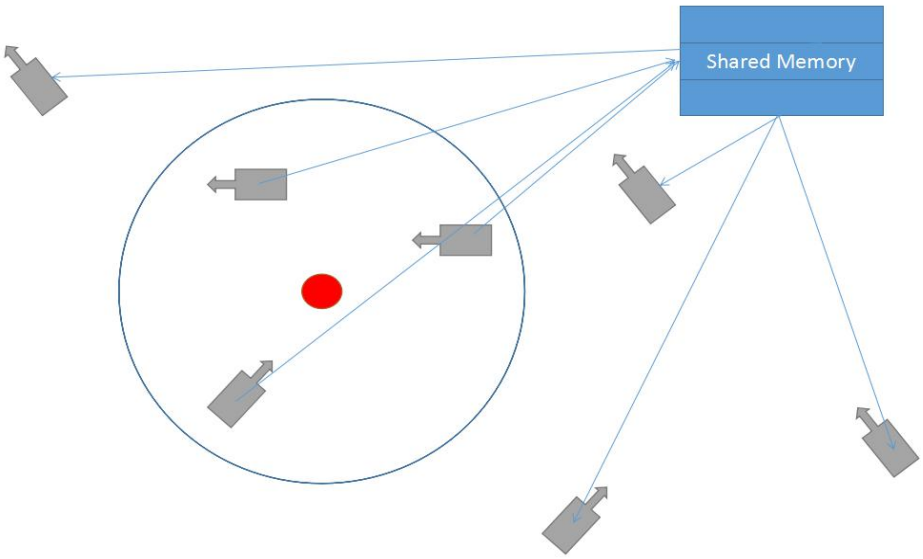

Figure 2.1.1 Shared Memory Based

Then I changed the memory based method to message passing. Although in terms of efficiency, it's not superior to shared memory, but it's more feasible in distributed system than that. Firstly, once find energy globes, the individual will broadcast a message including the number and position of globes so that the vehicles in outer scope could also received the globes information. Then, in order to pass the message as far as possible, I implanted message forwarding in the vehicle tasks as well as add a variable to record the send time of the original message (Figure 2.1.2).

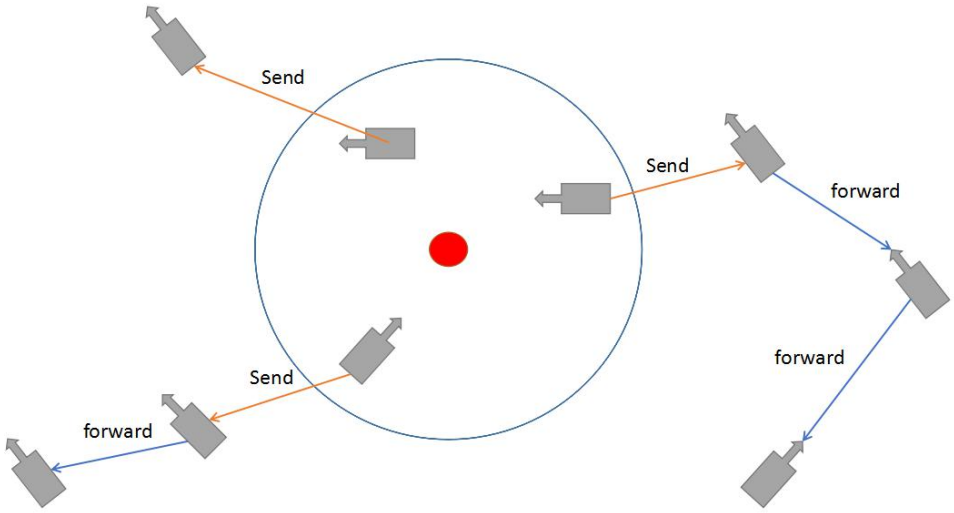

Figure 2.1.2 Message Based

Based on the design above, the vehicles will always choose to use the newest message based on the comparison of the message recorded time. Another action that one vehicle could have is to keep sending (forwarding) the message as long as the message is not outdated. Although it seems to lose precision (it will be solved later), it could expand the scope that message could reach since some uninformed vehicles pass by during that period could also get notifications.

The last strategy I used in this part is to keep the vehicles in close contact with the globes, in order to solve the problem mentioned in Section 1.3. Once a individual finds its message is outdated, after a period of time, it will keep

broadcasting a “Query Message” to ask others if they get some new information, and slowly move close to the globes recorded by the last message. As a vehicle nearby, if it receives a “Query Message”, it will broadcast a “Answer Message” as long as the message it has is newer than the query one. Back to the questioner, after sends the “Query Message”, it will start to listen the outside world for the answer. Once they get a respond, there are two options it could do. If the message is newer and could be defined as updated, then it gives up reaching the globe since it has already got what it wants. In the case that the message is still outdated, it will keep listening and trying to reach the globe, the message it receives could help it to adjust the destination to get closer to the actual globe. (Figure 2.1.3)

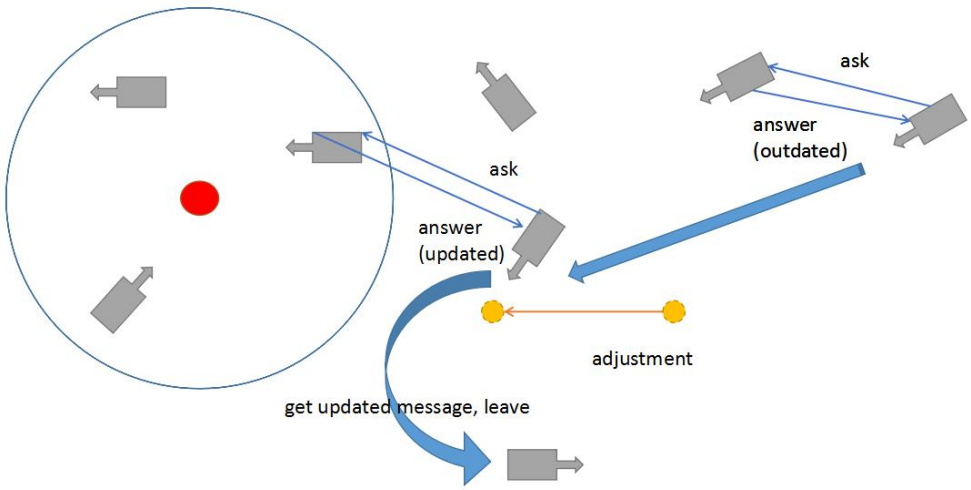

Figure 2.1.3 Ask and Answer Message

The reason why I let them go to the globes is that, presume one vehicle’s

message is outdated and it gets lost, it's more likely to find the updated message by doing this than just set it free as long as the globes will not mysteriously disappear and the time criteria for "outdated" is not too long. It probably simpler to broadcast their message all the time, regardless of whether it's outdated or not, and always get the newest message. I chose the Ask and Answer message instead, however, because it is more efficient, which could be more useful to solve the problem in Section 1.1. The former method will cause high pressure on the message network since lots of useless messages are passed. By contrast, Ask and Answer strategy will force the vehicles to send messages only if they could be valuable.

2.2 Methods for Recharge

Based on the information sharing methods above, the next issue is to get recharged. I set up a threshold to remind vehicles if they are in dangerous energy stage. And the simplest strategy for individual is going to find the globes once the energy is below the threshold (Figure 2.2.1). However, as I mentioned before, the vehicles may face the problem of cannot get to the globes since numbers of them are trying to do the same thing.

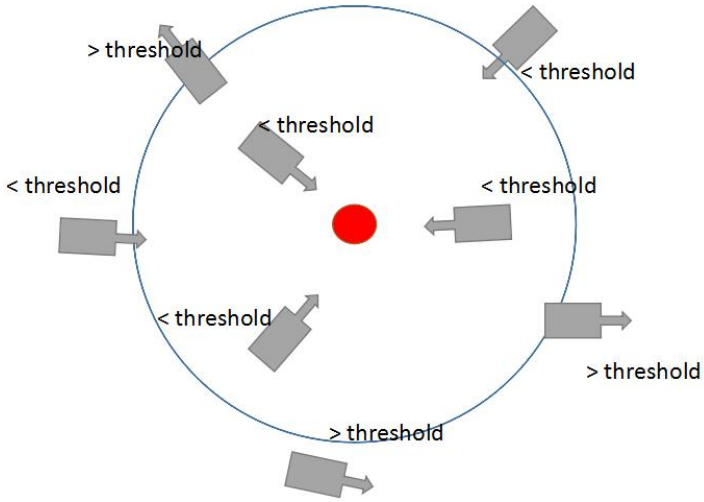

Figure 2.2.1 Threshold Based Recharging

I tried several attempts to fix this problem, and these methods could help to ease the stuck problem but still are not able to entirely avoid it. The first thing is to let the vehicles move as fast as they can to pass the globe and leave, so that the possibility of getting stuck could be lower than the slower speed one. That is mainly because the time for each individual to stay around the globes is shortened.

In the second place, for each vehicle, it will go to the closest globe to get recharged if receives information of several globes (Figure 2.2.2), because it could save energy cost to reach the globe. And with the distance constraints (it will be mentioned later), vehicles will be separated into numbers of groups, depending on the number of globes, so that the chance of getting stuck will

be reduced by the smaller population in each group.

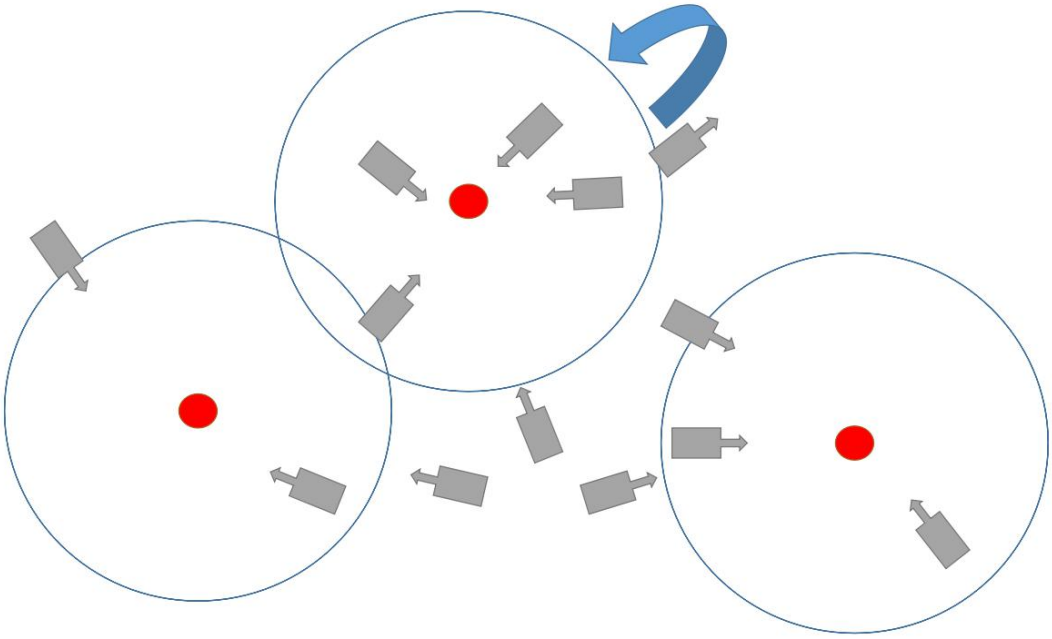

Figure 2.2.2 Closest Distance Strategy

Finally I planed to add distance constraints for each individual, which prevents vehicles from moving far away from the globes and couldn't get recharged in time. This design is to ask the vehicles back once they reach the distance boundary. (Figure 2.2.3)

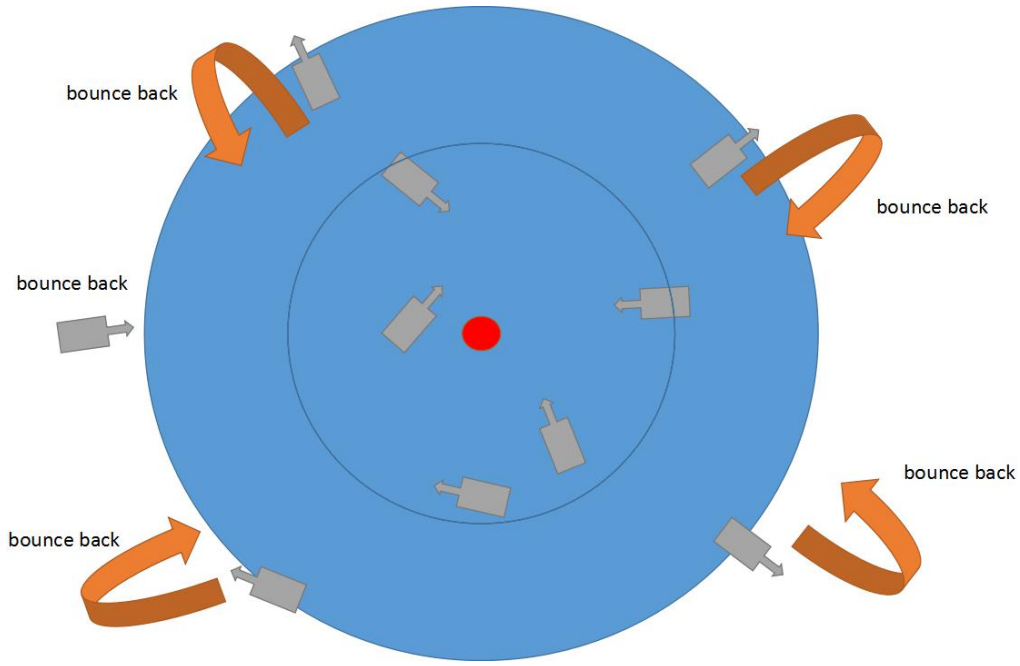

Figure 2.2.3 Distance Constraint

2.3 Abandoned Designs

This section is to describe some designs that were discarded during the development process. Some of them could be useful but hard to implement, while others are interesting but seems unfeasible.

The first design for vehicles is to find and go to the globe which contains less population. It could provide more opportunities for individual to recharge its energy than current design since there will be less competitors to reach the globe. (Figure 2.3.1)

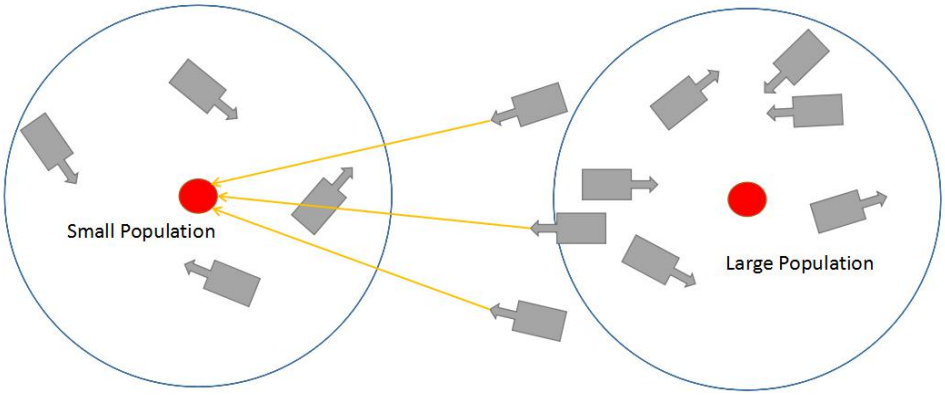

Figure 2.3.1 Follow the Less Population Globe

However, this is hard to be implemented based on Massage Passing Strategy. For individual vehicles, it's hard to count the number of vehicles which are approaching to a globe since it's not able to contact with all of the vehicles at the same time. Even though it can, there is another problem of processing it because everyone is sending multiple messages to notify others, which makes it time-consuming to count every individual in based on their identifications.

Secondly, I tried to build a stuck-free environment. I estimated the approximate energy cost and time cost to reach a position at several Throttle Rate. Based on this estimation, I planed to use Priority Queue to save the recharging sequence. In this case, an individual will calculate the time and energy waste to reach the globe and get recharged, then they put the information into the Priority Queue sorted by energy remain. A specific time slot has one Priority Queue, and a vehicle will go to the globe once it finds

that it's on the top. By doing this, all the vehicles will visit globes sequentially and it's stuck-free. (Figure 2.3.2)

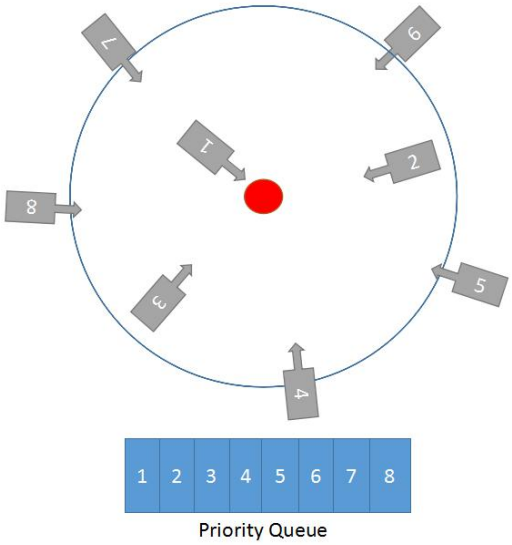

Figure 2.3.2 Stuck-Free Environment Based on Priority Queue

The same problem takes place that it's hard to get all vehicles, who wants to reach the globes, at the same time, which will cause confusions. And also, it seems impossible to notify all the vehicles that one has finished recharging, and that will unnecessary force many of them waiting for a long time. Most importantly, the program will lose concurrency, since they get recharged sequentially and parallel accessing with stuck-free is too hard (probably impossible) to calculate.

Finally, I got an inspiration from assignment page and considered a strategy that let all vehicles move to the same position, which far from the globes,

after they get recharged. And as a result they will form a circle and get strong connection among them, which could help them get updated messages all the time. However, it still has problems when the population swarm is too small (have trouble in contacting others if the circle is too big) or large (vehicles may get stuck if the circle is too small). And more importantly, if there are two groups of vehicles lose connection with the system, the message passing process will get serious problem (Figure 2.3.3), which makes the entire system undependable.

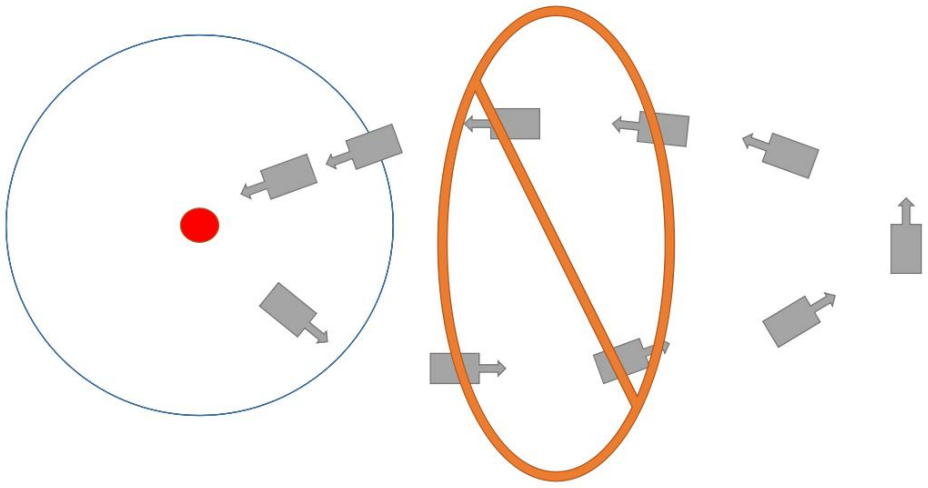

Figure 2.3.3 Connection is Broke

3. Implementation

I created a new package, named “vehicleFunction”, to save all the procedures, functions and variables that I use in this experiment, so that it could be convenient to program and review. And the code is for Stage D.

3.1 Message Passing

There are three types of message, defined by three Boolean variables, “is_find_globe”, “is_ask_info” and “is_answer_info”. And also, a Duration type variable “message_outdated” is used to judge whether the message is outdated or not. All the time comparisons are based on package “Ada.Real_Time”.

The first thing for each vehicle is to use “Energy_Globes_Around” to detect the surrounding globes. Once they find the globes, they will save the messages in their local memory, and then broadcast them as “is_find_globe” type.

In the second place, if an individual finds the message it has is outdated, it will broadcast a “is_ask_info” to ask for help.

Finally, in terms of receiving message, there are three options that one could choose:

a. If the message type is “is_find_globe” and it’s newer than current one, then replace the local one with the new message. And also, if the message one holds is still updated, it could keep forwarding that information in order to let more vehicles get notified.

b. If the message type is “is_ask_info”, which means someone nearby needs help, then it well send a “is_answer_info” message back as long as the information it has is newer. The reason of doing this is to make good use of any message that is valuable. As I mentioned in Design Section 2.1, even though the response message might be outdated, it could help the questioner to adjust its direction to find more useful information.

c. The last option is for questioners, if they receive any new message or response message, and it’s newer than what they currently hold, then they will save the message. And if the message is still outdated, they will keep listening to the outside world.

3.2 Recharge

Firstly, I set up a variable named “dangerous_charge_level” as a threshold. The vehicle will try to reach the globe once it finds itself in a dangerous energy level. Variable “closest_globe_index” always contains the index of the closest globe, which an individual will choose to reach.

And also, as mentioned in Design Section 2.2, there is a variable named “globe_distance_limit” to keep the vehicles in close connection with globes.

Once an individual finds itself far away from globes, it will turn back. Moreover, the vehicles who have outdated messages will try to go back to the original globes' positions, and then be set free once they find the updated messages.

3.3 Others

In order to make the testing and programming easier, there are several parameters in vehicle task's local memory.

4. Analysis & Evaluation

The functionality of the strategies have been explained in the Design Section. In this part I will focus on the analysis of the test problems and the feasible solutions.

After implemented all strategies above, vehicles could survive in a relatively wide population range, from 40 to 150, based on a set of configurations (Figure 4.1). However, they still have troubles outside the range. For the tests below, I set the initial number to the population that I need, instead of adding them during the process. The reason is to prove all the vehicles could survive even though they don't know the globes at first as well as they need to recharge approximately during the same time slot in the first round.

Energy_Globes_Velocity	dangerous_charge_level	message_outdated	find_message_speed	globe_distance_limit	Number of tests	Time	Initial Population (Average)	Final Population (Average)
(x => 1.15, y => 1.0, z => 1.0)	0.5	0.5	0.5	0.1	3	10 min	20	18
							40	40
							70	70
							100	100
							150	150
							200	194
							300	216

Figure 4.1 General Test

In terms of low population, the first reason why vehicles cannot survive is that, compared with the high population swarm, they have relatively low possibility to find the globes in the beginning. It seems inevitable and I have nothing to do for that. However, another reason is that connection between them is not as powerful as the high population one, since there is lower number of vehicles per space unit. To solve this problem, I could lower the message outdated parameter so that the vehicle will be able to go back and find the latest information in time. Or I could increase the dangerous charge level, so that vehicles will have enough energy to find the globes. (Figure 4.2)

(x => 1.15, y => 1.0, z => 1.0)	0.5	0.2	0.5	0.1	3	10 min	20	19
		0.3					20	19
		0.4					20	18
	0.6	0.5					20	18
	0.7						20	19
	0.8						20	19
Oringinal Configuration							20	18

Figure 4.2 Test for Small Population Swarm

For the high density swarm, the inevitable reason is my computer's configuration, since my laptop was struggling during the tests, and the program was not working as fluently as the low population swarm. And when I execute the program in multiple times (so that it wouldn't be too long for the testing process), it's possible for the individual vehicle to be not able to react to the environment and run out of its energy. But there are other reasons that may cause this problem.

Firstly, the message outdated time. With the advantage of relatively strong connection, it's unnecessary for individuals to refresh their information as frequently as the smaller population swarm. By expanding the message outdated time, there will be less stress near the globes and vehicles should be easier to reach them. (Figure 4.3)

Secondly, the distance constraint. The original configuration is not suitable for large number of vehicles, because the possibility of getting stuck will become higher if the average of vehicles per space unit increase. As a result, it could be an option to expand the distance constraint scope. (Figure 4.3) However, the test data seems to be against this point. The performance was decline. Based on my assumption, changing the distance constraint parameter doesn't influence the density of vehicle near the globes, and even worse, it may probably increase the possibility for individuals to get lost since they could go further than before. In this case, this attempt is not useful. And some other strategies might be needed in order to decrease the vehicle density around the globes.

Energy_Globes_Velocity	dangerous_charge_level	message_outdated	find_message_speed	globe_distance_limit	Number of tests	Time	Initial Population (Average)	Final Population (Average)
(x => 1.15, y => 1.0, z => 1.0)	0.5	0.7	0.5	0.1	3	10 min	200	197
		0.9					200	200
		1					200	200
		0.5		0.15			200	191
				0.2			200	189
				0.25			200	190
Oringinal Configuration							200	194

Figure 4.3 Test for Large Population Swarm

In summary, there isn't a set of configuration which is suitable for all environment. That is mainly because the requirements of different density swarms are different. Large population swarms get a good capability of communication, since everyone stays relatively close to others. What should be done is to prevent individuals from getting stuck. On the contrary, the small population swarms don't need to be concerned about getting stuck because of the low density. Problem for them is to keep close contact with others. If not, it's possible for individuals to lose connection with the outside world and run out of charge.