

Simulating Metamorphism in a Swarm Robotics System

Final Report for CS39440 Major Project

Author: Mr. Rhydian Jenkins (rlj10@aber.ac.uk)

Supervisor: Dr. Elio Tuci (elt7@aber.ac.uk)

4th May 2016

Version 1.2 (First Draft – unrevised)

This report is submitted as partial fulfilment of a BSc degree in
Computer Science (G400)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

In signing below, I confirm that:


- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name 

Date: 04/05/2016

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name 

Date: 04/05/2016

Acknowledgements

I am grateful to my MMP tutor, Elio Tuci, for providing support and guidance throughout my project as well as ensuring that the project stayed on path. Elio was there to answer questions and provide alternative ideas to the structure of my project.

The Computer Science Department and the facilities of Aberystwyth University have been essential in allowing me to work on my project. Examples from previous students were available at reception, emails reminding me of approaching deadlines, and helpful forums and content on the Blackboard were available to me at all times.

Abstract

Swarm Robotics is a new, scalable, and rugged solution to the co-ordination of multi-robot systems. It allows each bot in a potentially large system to act purely out of self-calculated judgement based on its immediate environment. An overall manager could be used to individually feed instructions to each bot and micro-manage the entire task however this method falls flat as more and more bots are added or as the task becomes more and more complex. No single machine can keep up with thousands of agents all needing detailed instructions at once.

Swarm robotics takes this problem and moves the responsibility of control over to every bot. With each bot thinking for itself, the system has no limit to the amount of agents that can be added. Not only does the system become more scalable, but it also is less prone to error. If any of the bots malfunction or if any unexpected conditions arise, the bots will have the ability to adapt and decide for themselves based on their observable environment what to do.

My project aims at a specific problem within Swarm Robotics – getting multiple bot to metamorphose. This document will tackle all the problems faced with simulating S-Bots as they organise themselves into a specific shape. They will be doing this through the use of LEDs situated on the bot itself. The overall goal is to get multiple bots to organise themselves and connect with each other in order to form a desired shape.

This is a simulated project, and will not be tackling any of the hardware challenges such as image interpolation and hardware interfacing with the S-Bots. Although I will talk about implementing my software on read world S-Bots, we are simply trying to get our system working on simulated conditions.

Contents

1.	Project Report Layout.....	8
1.1.	Section 2: Background, Analysis, and Process	8
1.2.	Section 3: Design.....	8
1.3.	Section 4: Implementation.....	8
1.4.	Section 5: Testing.....	8
1.5.	Section 6: Critical Evaluation.....	9
2.	Background, Analysis, and Process.....	10
2.1.	Background	10
2.2.	Analysis	12
2.3.	Process	13
3.	Design	15
3.1.	Proposed Specification.....	15
3.2.	Overall Architecture	16
3.3.	Detailed Design	17
3.3.1.	Original Design UML.....	17
3.3.2.	Justification for Original UML.....	18
3.3.3.	Final Design UML.....	19
3.3.4.	Justification for Final UML.....	19
3.4.	User Interface Design.....	20
3.5.	Choice of Language and Libraries	22
4.	Implementation.....	23
4.1.	The Classes.....	23
4.1.1.	Simulator	23
4.1.2.	GUIManager	23
4.1.3.	BotCanvas.....	24
4.1.4.	CustomListener	24
4.1.5.	Bot	24
4.1.6.	Sensors	24
4.1.7.	LEDManager	25
4.1.8.	LED.....	25
4.2.	The Functions.....	25
4.2.1.	Virtual Force Field (VFF)	25

4.2.2.	Directions	28
4.2.2.1.	The Global Direction System	28
4.2.2.2.	Direction to a Bot	28
4.2.2.3.	Average Direction Finding	28
4.2.2.4.	Turning to Desired Direction	29
4.2.2.5.	Moving in Direction Facing	29
4.2.2.6.	Looking at a [X, Y] Position	30
4.2.3.	LEDs	31
4.2.3.1.	Meaning of the LEDs	31
4.2.3.2.	Updating the LEDs	32
4.2.3.3.	Closing the Open Connections	32
4.2.4.	Rendering	32
4.2.4.1.	Bots	32
4.3.	The Behaviour of the Bots	33
4.4.	Comparing with the Specification	34
5.	Testing	35
5.1.	Overall Approach to Testing	35
5.2.	Stress Testing	35
5.3.	VFF Testing	38
5.4.	Acceptance Testing	39
5.5.	Performance Testing	40
6.	Critical Evaluation	41
6.1.	Research Evaluation	41
6.2.	Specification Evaluation	41
6.3.	Process Evaluation	42
6.4.	Product Evaluation	43
6.5.	Conclusion	43
I.	Appendices	45
A.	Third-Party Code and Libraries	45
B.	Ethics Submission	45
C.	Code Samples	45
	Bibliography	52

1. Project Report Layout

This is a section by section breakdown on what was discussed in each part of the report.

1.1. Section 2: Background, Analysis, and Process

In the first section we will discuss the background research I conducted, where the topic of my dissertation came from, and how it evolved. We will also be analysing current systems and looking at the state of Swarm Robotics in the modern industry, as well as what challenges the current systems are currently facing. Finally, we will conclude the section by talking about the process I will follow in the development of my system.

1.2. Section 3: Design

We will begin this section by discussing the proposed specification. This is a list of all of the features my finished system will hopefully do. This includes a minimum list and a further list, which will be completed if time is available.

We will also discuss the architecture of the project, involving where the files are and why they are there, as well as a detailed structure of the code involving a before and after UML diagram, before briefly talking about my choices in the GUI design.

Finally, I will justify the choice of the language and details of specific libraries I have implemented as well as talk about my choice to use a custom simulator.

1.3. Section 4: Implementation

This section will cover how I implemented each class in the program. There will be a lot of mathematical detail, as well as discussion on the values used in all of the equations and why I used them. We will also discuss the technical choices in algorithms and what else I could have used in their places, as well as the overall roll of each class in the system.

1.4. Section 5: Testing

Here we will look at my methods of testing, what I tested, and what I found whilst doing so, and evaluate how successful testing was to the project. The major points of the testing have been split up into subsections where they will be discussed in further detail.

I will not discuss each and every single test case I conducted; instead I will simply highlight the important ones.

1.5. Section 6: Critical Evaluation

Section 6 will evaluate the overall success of the project. It will go into detail on both how the project could have been improved on as a whole, as well as what I think went well. I will also talk about what I would have done differently if I were to tackle a dissertation of this size again. The evaluation will be split up into subsections, each analysing specific parts of the project.

The section will then conclude with an evaluation of the project as a whole.

2. Background, Analysis, and Process

2.1. Background

Before even starting the research I knew that I wanted a project that seemed fun to do, interesting, and was something I have never done before. Ideally, something that was graphically heavy would have been ideal as that seems to be what I enjoy the most.

Swarm Robotics stood out to me as a potential project as it seemed like a new and exciting future technology [1] that has only begun to be implemented in the laboratory. Many papers have been published explaining the advantages of such a system, and how

I found that this kind of methodology is closer to the real world examples of nature where sometimes vast numbers of agents (ants, bees, birds) work together [2]. There is no leading project manager in an ants nest instructing each worker any on how to do their job, neither is there any king starling that instructs every other starling on where to fly to form the amazing flocks that we see at sunset [25].



Images: <http://www.lovethepics.com/2012/11/sensational-starling-murmuration-far-out-flocking-phenomenon-37-pics-13-vids/> (right), <http://www.alexanderwild.com/keyword/ant%20colony/> (left)

After finding out what the essence of what Swarm Robotics is, I began to read papers on the current state of robotics and discovered that the subject has a long way to go before it is truly relevant in every-day life. In other words, the theory of “a thousand robots are better than one” is a solid idea however it seems that the implementation of such systems have yet to be fully achieved. Systems such as AutoStore™ [3] could potentially adopt a swarm-like approach to their machines; however, existing swarm robotics systems are by and large still limited to displaying simple proof-of-concept behaviours under laboratory conditions [27].

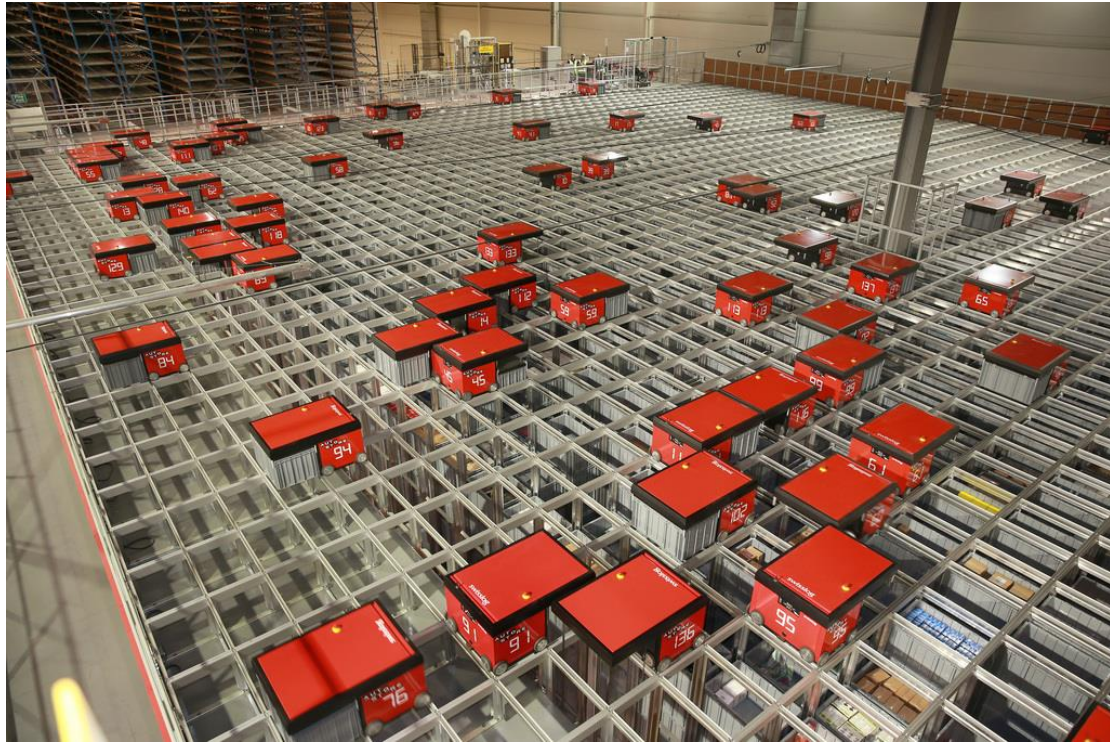


Image: AutoStore™ Warehouse

It did not take a lot of research to understand that the potential for such systems was what was driving the push of research and innovation forward [26], and formed the main reason Swarm Robotics was chosen as my research topic - I see it as a technology that holds a large place in multi-robot systems of the future.

With more research the lack of implementation became more apparent. Getting a machine to decide on what to do in unknown conditions is always hard, and these bots will need to do exactly that. Given a high level task instruction – “Build this”, “Search for survivors”, “Create this structure” etc – the bots will have to decide how they are going to help complete this task using only the data on their environment that the sensors they are equipped with gather. In other words, labour allocation is definitely going to be one of the major challenges.

Finally, I found a topic I would later invest the focus of my dissertation into – Metamorphism [20]. While looking through various articles and videos I found a paper discussing the need for bots to connect themselves together and form certain shapes in order to complete tasks that are too physically difficult for any single bot – such as situations where a gap in the terrain needs to be crossed without the use of a bridge, or where an object that is too heavy to move by a single bot is blocking the way.

Even though there has already been some development on this topic, it sounded like a fun and interesting topic that will allow me to learn and understand the fundamental principles of Swarm Robotics.

2.2. Analysis

Swarm Robotics currently has plenty of areas that need improving or developing before it is a viable option in the real world [4]. Some of the most challenging questions are “How can the cooperative schemes inspired from the nature swarms integrate with the limited sensing and computing abilities for a desired swarm level behaviour?”, and “How to design a swarm of robots with low cost and limited abilities which has the potential to show great swarm level intelligence through carefully designed cooperation?” [5]. Some of these problems are in need of cheaper and more powerful hardware to become available, however others need innovation in algorithms.

I found plenty of articles about ideas of Robots automatically building structures [6], exploring Mars [7], mapping unknown environments (e.g. buildings) [23], or scavenging for earthquake survivors [8] however most of these ideas are in very early stages of development and still have a long way to go before they're realised [24]. For now, the study of Swarm Robotics is limited to a relatively small group of robots performing relatively simple and mundane tasks [9].

I took particular interest in sending swarms of robots to explore Mars. For such a task, the rovers will need to be able to traverse through rough terrain and overcome obstacles too treacherous for any single bot [10]. For example, a steep incline will mean that the bot could tip over. One of the solutions to this is to have the bots attach themselves to each other in such a situation and tackle the obstacle as a group. For this to happen, the bots will need to independently arrange themselves to form a desired shape, which will involve the bots attaching themselves physically to another bot.

And so, I had my project. The problem was simple enough to understand - get the bots to form a particular shape. This task would have been a lot easier if the system did not comply with the traditional rules and had an over-all controller class telling each bot what to do. However, this would have defeated the point of Swarm Robotics. As we have already discussed, this system would not be scalable in the real world and will not be able to have a go at any and every environment it was placed in. Also, the bots will probably be small, cheap, and relatively simple mechanically [21]. This means that assuming every single bot in the system (potentially thousands) will need to work correctly. We cannot have 999 bots waiting for the 1000th to make a move before they all continue.

So, a system that has no leader is needed. Every bot will need to be exactly the same, and every bot will need to be observant to its immediate environment and nothing else. If done correctly, an external input that has not been foreseen or pre-calculated could move the bots in a random direction (For example, wind blowing bots off course [11]) and it will not make any difference. Each stage of the simulation the bots will act on what it immediately sees and that alone, calculating what to do on data that it has just gathered about its environment.

Ideally, the bots will do this without fail and be ready to act on literally any environment they are placed in without the need of calibration. Such a system will not be prone to any faults in the environment (i.e. a blockage in the planned path) or in any other bots (i.e. if a bot breaks down). Such a system will also be easily scalable. If the system has too many bots in, some can be taken away without worrying on their effect with the rest of the system. If there are not enough, bots can be added without the need for the others to even acknowledge the change in the amount of bots in the overall system.

In a perfect world, the bots will know the exact location of every other bot, as well as the exact location and properties of every rough patch of terrain and obstacle. As the environment of Mars is far from perfect, the bots will only be able to know the location and properties of their immediate

environment. A lot of the time, the bots will not be able to see any other bots and will appear to be completely on their own. This imposes another question, how do you attempt to spread out when you do not know the location of your fellow bots?

With all this taken into account, a feature list can now be constructed for the bots. They will need the ability to connect to other bots, as well as signal for bots to connect with themselves. They will also need the ability to sense their immediate environment, as well as a method for communicating with additional bots without breaking the rules of Swarm Robotics.

These requirements immediately drew me to S-Bots [12]. They are small, cheap, and equipped with hooks and LEDs that can be used as a means for connection and communication. Conveniently, my university had several S-Bots in the lab in case I needed to test my system in the real world with real physical bots. After further investigation, I was able to find an article that used S-Bots that attached themselves to other bots in order to lift them or move them.

2.3. Process

I knew that the stage of planning was going to be especially important as I have never tackled anything of this size or complexity before. It was worth taking extra time at this early stage to plan everything out as it would help me later on in the project. Before planning any of the structure or any code, or producing any UI designs, a full feature list was written. This feature list was a comprehensive outline containing all of the *stuff* the finished project will be able to do. After the list was completed, each feature was broken down into individual tasks. For example, one of the simulators features could be that the bots could be dragged and dropped by the mouse in order to move the bots around. This was considered quite an important feature and was to be completed as soon as possible. For this to be completed, tasks such as “produce a working GUI” and “Allow *something* to be drawn on the canvas” needed to be created first.

After careful planning the tasks for each feature, the system could be chronologically planned out. More important features that needed to be implemented in order to start work on others were completed first. With a full list of features, versions were planned. A version was to be a milestone for testing and backups, meaning if something during the development was to go wrong, an earlier version could be picked so and started from that point.

```
0.2
Plan has been formatted and expanded.

0.3
An unreviewed plan. This plan will be taken and revised with my tutor.

SIMULATOR
1.0 (finished 12/04/14)
Multi-target capability has been added. Bots can now form various shapes depending on what seed is selected.
The bots are now slightly attracted to the center of the canvas, which removes the need to look at the center when leaving it.
Bug fixes and refactoring to finalise the project.

1.2 (finished 05/04/14)
Bots now gather all data of which bots they're connected to through the LEDs.
Instead of the target system for selecting seed bots, the user will select and pick a bot to become a seed.
The bottom panel has now been removed, as well as any button listeners.
Added instructional text to top left of screen.

1.1 (finished 28/03/14)
When a bot connects to the bot, its LED will switch off and the other bot's LED will come on (for now, manually).
Also, the connecting system has been improved so that the bots will now connect in a straighter line.

1.0 (finished 14/03/14)
LEDs are now controlled by an LEDManager class. This means that the Bot can give high level commands to the LEDs
and not have to micromanage them.
Bots will now connect themselves to other bots properly, and display LED lights confirming so.
A bug has been fixed where the bot can be seen but the LED cannot as far away.

0.9 (finished 15/03/14)
Bots now have multiple coloured LEDs, only 1 of which has a charge (attraction). The others are used to confirm the bots have connected to another bot.
The user can now place an object in the canvas which will serve as a point/target where a seed robot will alert itself.
This target has an interface and a single unimplemented target type (line).
The positioning system bug has been corrected to make the center of all objects its x, y co-ord.

0.8 (finished 03/03/14)
VFP has now been implemented and works properly. The bots can now actively seek out LEDs and move towards them whilst avoiding other bots.
The values for the system has been tweaked.

0.7 (finished 01/03/14)
Version code plane has been refactored to make the system easier to understand and work more efficiently.
The color scheme has been changed to make the simulator more aesthetically appealing.
Work has begun on Virtual Force Field (VFP) to allow the bot to get to a desired position while avoiding other bots.
Although the system is technically implemented, version 'tweaking' is needed to get it to work properly.

0.6 (finished 25/02/14)
LEDs have been increased to 7 per Bot, and the functionality has been improved slightly to make them easier to use (switchable). Bots can now also move and avoid each other randomly. This is done by simply going in the opposite
direction to the average vector of close bots.
```

Image: Versions document during the report state of my project

Each version of the simulator was saved into one directory (regularly backed up!) with a text document describing what was done and what was completed with each version, when it was started, and how long it took. Any bugs found and fixed were also listed here, as well as any changes to the system that was made.

At this point the system was split up into versions, with each version having a list of features, and each feature having a list of tasks. Coding the project could now start at the beginning of the list. New tasks could be added as more content was thought of by doing it this way, it was easy to see progress and you never felt that you were lost with what to do next.

Therefor the system was built on a mixture between version control and feature driven methodologies. I chose this method of development for a number of reasons; Firstly, it allowed me to clearly see progress and talk about the next steps with my tutor the whole way. Secondly, the regular intervals between versions allowed easy milestones for backups, acceptance tests, and refactoring. Finally, when it came to reporting on what I did, it meant I had a full comprehensive list of what I did and when I did it.

A lot of the testing was visual: "Did the bot move to where it was supposed to?", "Do the bots look like they're avoiding each other?", "Does the bot move towards a target?". These sorts of tests would have been very difficult to write code that checks if they're done correctly and therefore made a TDD approach unfavourable.

Another approach briefly considered was that of the Waterfall Model (planned). Because I have never attempted to complete a project of this size, and of this level of research, I decided that a more agile approach with a more forgiving cost of change would have been more appropriate.

3. Design

For projects as large as this one it is very important that the design is done correctly. Everything must be planned out at least roughly in order to give an idea of how all the pieces of the project will fit together before any of the code can start. I found this beginning stage of the dissertation to be the most difficult and require the most time as I knew changes to this plan would be more costly the later I left them.

With this mentality I tried to have all of the classes and functions planned out exactly before even starting, which would mean I could have simply written the code with very little worry about the overall architecture and how to fit it all together and focus entirely on the functionality of the small section of code I am writing.

This proved to be extremely difficult. With little experience of this method of design, and even littler experience of systems this large and complex, the original UML (2.2.1.) ended up changing a lot as I wrote the project which was mainly due to unforeseen issues with the design and better solutions to problems that were discovered only after I started the code.

The idea to plan the entire project before starting it was ambitious as I have not attempted a project this large or complex. The plan was not followed it would be easy to consider the design stage as failed, however I feel that having that basic idea of how things will fit together and building on that was far easier and more effective than simply 'making it up as you go along'.

3.1. Proposed Specification

I have to remember that this is a research project and that each feature will require more time reading about solutions and backgrounds to those solutions than actually implementing the code. With this in mind I feel the feature list produced was not only achievable but also challenging, which allowed opportunity for me to learn as much as I can on this project.

The final project will have a minimum ability to form a simple shape – the simplest of which will be a line. For this, the simulator will need to be complete with functionality for bots to move about freely and be able to sense their environment. These bots will need to have LEDs on them that they are able to change and the ability to sense the position and colour of the other bots' LEDs. These bots will also need to be able to position themselves so that they can connect with a desired bot, all whilst avoiding the edges of the canvas and other bots. If two bots do end up too close to each other, they need to move away from those bots in a direction that will not move in the way of another bot. This process can prove to be extremely computational and needs to be efficient and effective to work properly in a simulation.

The system will also come with an intuitive GUI, which will allow me to add new bots , remove existing ones, move the bots that are on the screen (perhaps through a click-and-drag method), and elect a particular bot as a specified seed bot. The GUI will be intuitive and easy to use, as well as show clearly where the bots are, what direction they are facing, and what LEDs they currently have on.

Once the bots have the ability to roam, find a bot, and connect, I will now consider the system to be at its minimum specification, however I will plan and hope that the final system will have more features than this.

One of the further features will be the introduction of noise. In a real world simulation the bots' sensors will not be able to pin point the exact position and orientation of the LEDs that it sees. The camera that the S-Bots will be using in the 'real world' will also be prone problems such as dirt or smudges on the lenses which will further increase the probability of slight differences in what the camera sees and what is actually there. This problem gets more serious the further away the objects it is seeing are. A simulator with this kind of distortion included would give a more accurate behaviour to the bots' behaviour, even though it will probably hinder the ability of the bots to make a shape accurately, quickly, or reliably.

A problem arises when getting a machine to look at an image and interpolate data from it. S-Bots are able to see their own LEDs with their camera and can sometimes interpolate it as a separate bot. Given time at the end, this is one feature that I will work on. I will hopefully give my bots the ability to detect themselves and remove their own LEDs from any further calculations.

Finally, a system that only makes a line seems like a boring and easy system to work on. I would hope that the final solution will be able to produce multiple shapes upon request. This would be useful in the real world as different shapes are useful for different things. For example; A hill too steep for the bots to traverse will need a line of bots to push each other up, however pushing a heavy obstacle out of the way of the path will require a shape that closer resembles a 'block' of bots. Looking to the real world, different shapes are also seen in different situations. For example, an arrow shape is most effective shape when it comes to reducing air resistance in migrating birds.

3.2. Overall Architecture

As mentioned, the early UML design I had originally planned to follow ended up looking very different to the final architecture of the project as later in the project I found better ways to do things. One thing that didn't change was the tick/render system (taken from Markus Perssons "*MiniCraft*" [15]). The system works in a very typical game engine style; it implements the Runnable class and allocates each entity in the world with its thinking time, much like a CPU does to threads in a computer. This allocation of ticks is what gets the components in my simulator to think.

This kind of ticking system could be considered step based, where each bot will perform one action at each step, and no bot will perform another step until all other bots have completed theirs. However it could just as easily be changed to a system where each bot preforms their step independently from every other by splitting said bots into their own threads and therefor mimicking the real world more closely. As the bots are stateless and are not built on the assumption that every bot has completed a step before they complete theirs I see no problem in implanting a step based system in my simulator, as it can easily converted if real bots ever need to be used.

Moving on to the GUI components, I decided to run them as JFrames simply because I utilised other parts of the java.awt.* and mixing libraries is not conventional in the computing world . The same cannot be said for the BotCanvas class, which originally extended a JPanel component however this was later changed to a Canvas component as it had better rendering functionality. With this new component I was able to create a triple buffer strategy which meant that the bots appeared to move at a much smoother motion and at less processor cost.

A custom listener was used to listen for user actions. I found it more functional and simpler to have one class that extended all of the input listeners needed in the system (ActionListener,

MouseListener, and ButtonListener) than have separate classes for each of these. This CustomListener class has a pointer to a selected bot. If this bot is != null, it means that the user has selected bot (by clicking and dragging it).

Overall, I feel that the final architecture is more logical and easier to implement than the originally proposed one. No system is perfect, so it is unfair to say that mine cannot be improved. For example, the Bot class is a very large class. Given more time, I would have somehow split this class up into further 'helper' classes like I did with the Sensors class or the LEDManager class. In the original UML, an Operation class was used to help the bot decide what to do next. This would have been a feasible addition to the Bot package what would have helped split the program up further.

3.3. Detailed Design

Note: These UML diagrams do not contain all of the methods in each class, but only the important ones. Basic methods such as getters and setters, or methods used to preform basic arithmetic such as turnAround() are not shown.

All UML diagrams were created using Creately.com, a free online diagram creation service.

3.3.1. Original Design UML

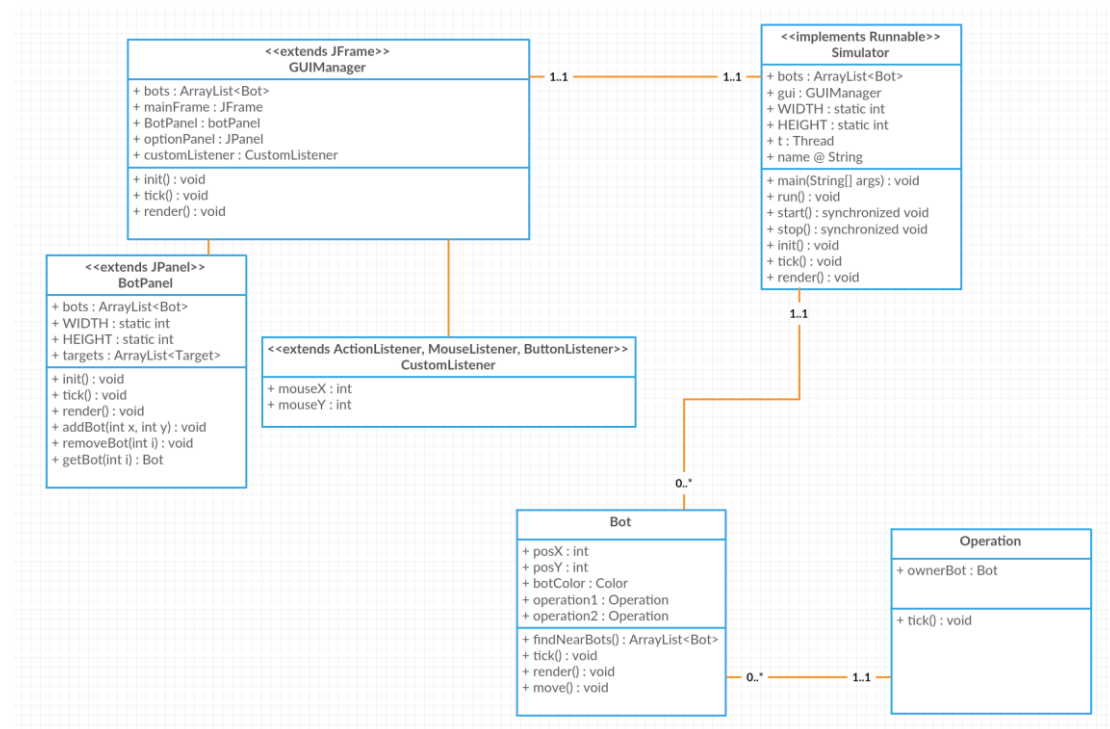


Image: Original UML design, created at Creately.com

3.3.2. Justification for Original UML

The entire architecture is ran on a 'tick / render' styled system. For every tick the class will 'think what to do next', and for every render the class will draw itself on the canvas. I have chosen this style of process because I have used it before in designing various games in Java, and it seems to work well.

The Simulator class (main) would be responsible for running the entire program. It would instruct all relevant classes to tick at set intervals which would mean that the system would run at the same speed on all systems. The FPS (frames per second), aka the number of renders the system can perform every second would be determined on the specs of the system running it. This number had no limit and would always run at the highest rate possible.

Bots are contained in an ArrayList owned by the GUIManager class. This class hands the ArrayList to the BotCanvas class which in turn will allow the user to move the bots for testing purposes.

This design had the bots' behaviour (what they should do next) determined by a separate class that was owned by the bot. A factory style method of loading in the class depending on what it is trying to do (i.e. roam randomly, connect to a bot, decide what LEDs to put on once its connected etc...) was going to be used. This would mean that for every 'tick', the bot will simply ask the current operation class to generate its own instructions and advise the bot on what to do next.

3.3.3. Final Design UML

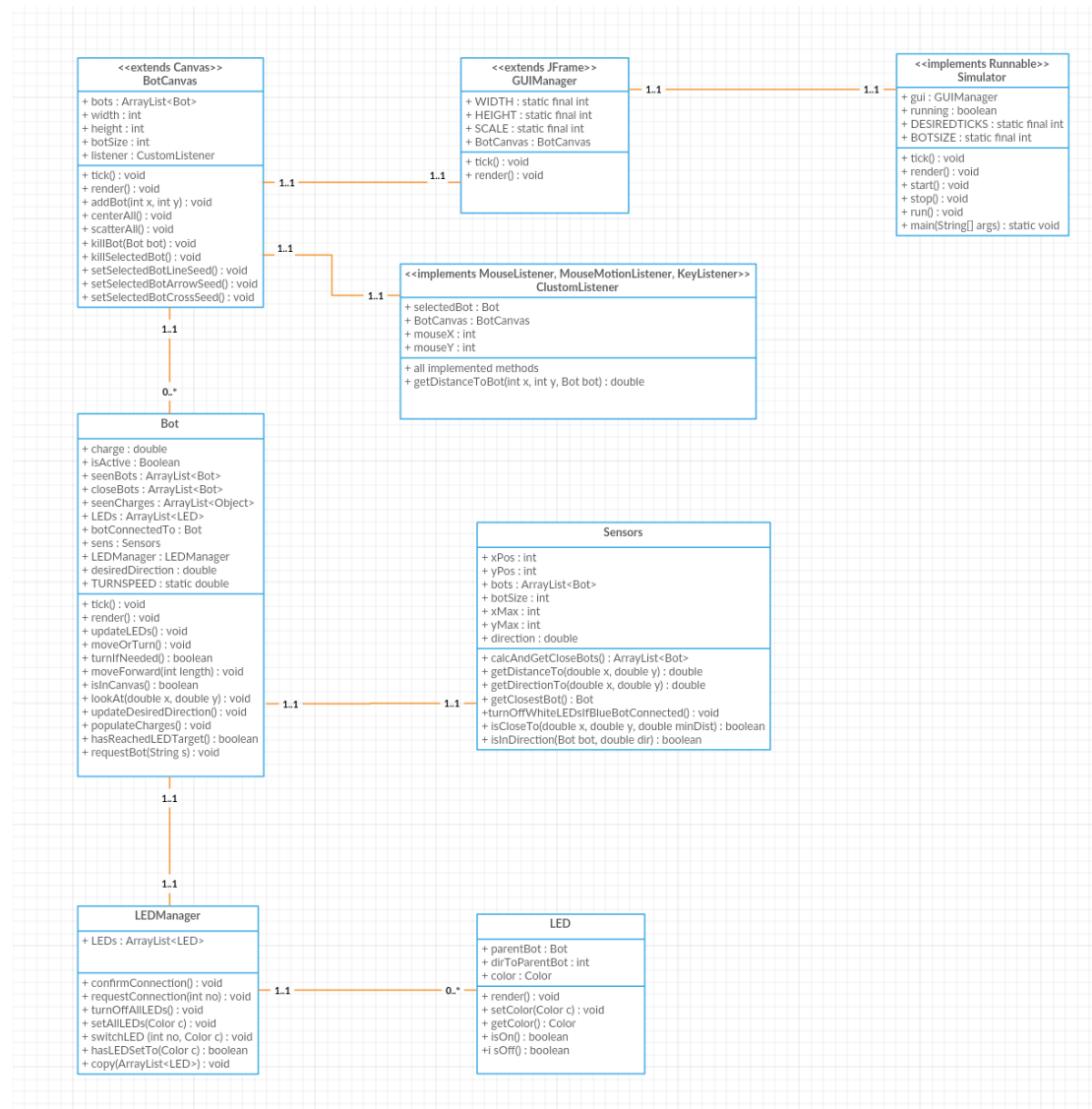


Image: Final UML design, created at Creately.com

3.3.4. Justification for Final UML

The final UML brings quite a large change when compared with the original design. Firstly, the Operation classes have been removed and their burden of decision has been moved to the Bot class. Secondly, each Bot class is equipped with a Sensors class which is able to keep the Bot informed about its immediate environment. Methods such as `getCloseBots()` and `getDistanceTo(int x, int y)` are available however the class cannot tell the bot about entities outside of its range. Noise can also be added to the Sensors class to make the values that it returns have a small random offset.

The sensors class was moved to its own class. As this is a simulation, in order to 'see' what bots are close to you the sensors class will have to go through each and every bot on the canvas and determine which ones are close. This means that knowledge of every single bot on the map is needed,

which would go against the rules of Swarm Robotics. In order to get around this, the knowledge of the entire system is encased in the Sensors class, which only tells the Bot what it would be able to see in the real world. For example, the Bot class would ask the Sensors class to get all of the bots that it sees, and it will return a list of all of the close bots only.

This way of splitting the Bot class and the Sensors class is a very easy way to simulate noise simply by getting the Sensors class to return an offset in the values that it provides. I feel this is a far more encapsulated way of allowing the bots to sense their environment.

Another major change is that the LEDs have been moved to their own class, and controlled through an LEDManager. This gives high level command functionality to the bot over its LEDs, meaning it doesn't have to micro manager each LED and instead simply request that a bot connect to it.

3.4. User Interface Design

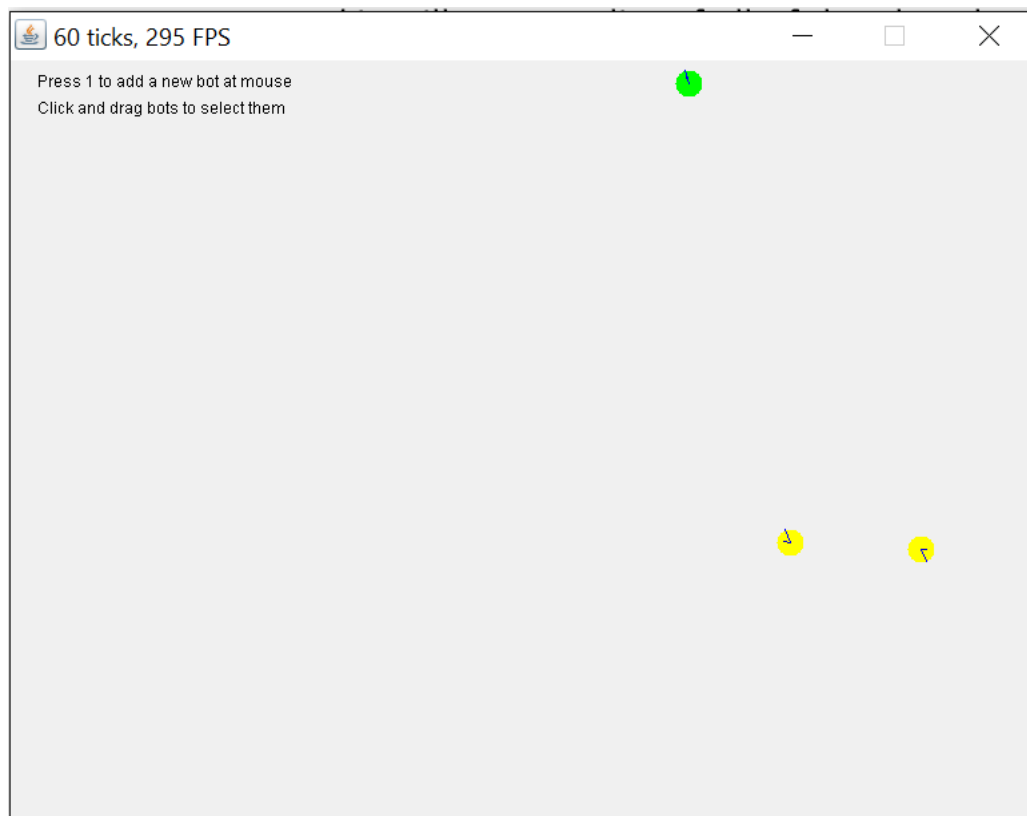


Image: Screen shot from version 1.3 (final) of my project, containing three unconnected bots

As the system was never intended for use by many people but more as a research tool, the user interface was not as important as a system such as a phone app. However, I am still happy with the way that my system looks and feel that if it were to be put into the hands of someone who does not know how to use it they will be able to work it out.

The instructional text seen in the top left corner of the window changes depending on whether or not the user has selected a bot as it only shows actions that the user can perform. When a bot has been

selected (clicking and holding on it), text appears instructing the player on how to remove that bot or set it as a seed.

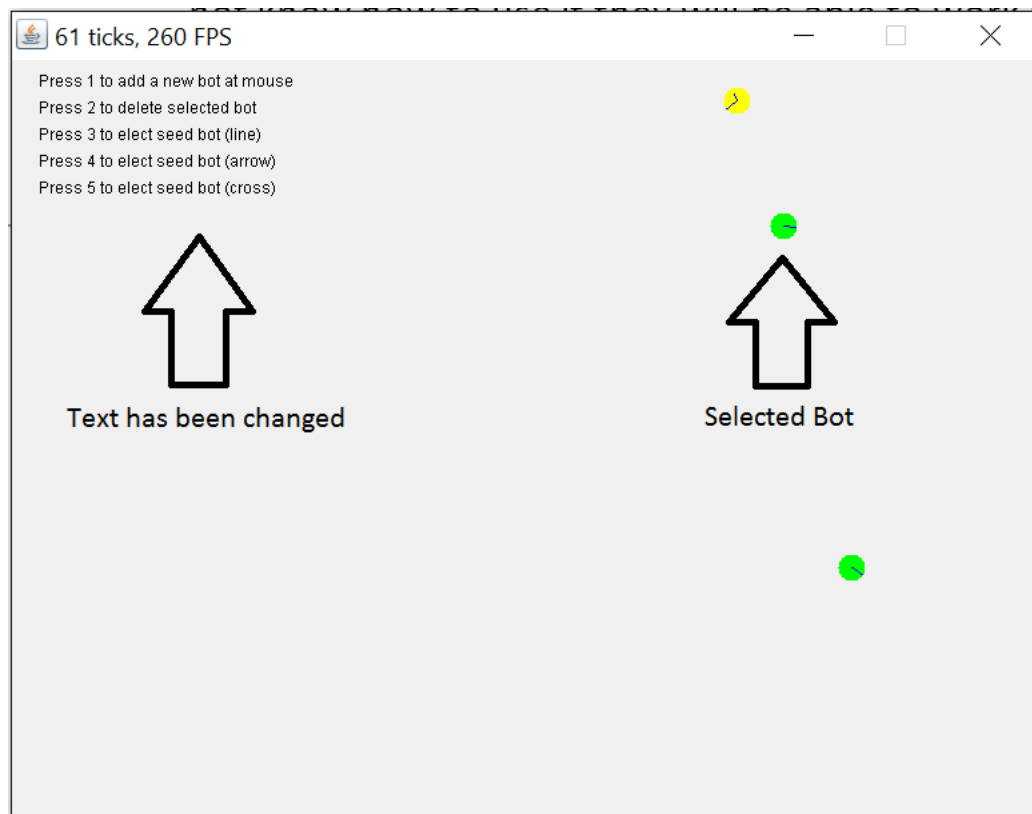


Image: Text change when a bot has been selected.

The graphical user interface is simple enough. During development there was a panel at the bottom containing buttons for testing. These buttons included functionality to move the bots (centre all and scatter all...), spawn in new bots, and remove bots already on the canvas.

It is essential that the window will not be resizable as the edges of the canvas act as the boundaries of the bots. To resize, a scale variable was implemented that was multiplied with the aspect ratio when the program was compiled. This means changing the size of the window is left to the developers and the functionality will not be available to the users, which was necessary as the bots will treat the edges of the canvas as a boundary to the world.

After most of the program was implemented, the button panel was removed and most of the functionality was transferred to the mouse. Text was superimposed to the top left of the canvas instructing the user on how to use the system which seemed a lot cleaner.

The bots will not be able to see the actual colour of the body of the other bots in the real world, however for development purposes I used this to show if the bots were able to sense any other bots around them. When the bot is rendered green, it means that it sees no other bot. It has no information of anything in its environment, and therefore is roaming randomly. As soon as the bot discovers another bot (meaning it sees one or more bots), the body colour is changed so that it renders yellow. Finally, when the bot is rendered red, it means that there is one or more bots close enough for a connection.

3.5. Choice of Language and Libraries

I could have used an already built simulator such as Bullet [13], however the bots I needed to simulate need to have very specific hardware with very specific methods attached to them. For example, the bots need to have a series of LEDs attached to them with the ability to emit specific lights in specific positions on the bot in order for my system to work – in other words, S-Bots were needed. Although these pre-build simulators could have been adapted to my needs, the time and effort it would have taken to learn the simulators and adapt them would not be much different to building my own. Also, by going through the entire process and coding everything myself, I was able to get a much fuller understanding of my system and was able to learn some of the graphical algorithms that would have otherwise be given to me.

Getting into the specific libraries I used, using Javas Swing library seemed the obvious choice as they were taught to us during the second year of my degree. This (in theory) means that I have a head start and did not need to spend as long learning the libraries.

C++ was a considered candidate for the language I could have used. By using C++ I would have had further control on the pointers and memory management of the system and could possibly have made the simulator run smoother and more efficient. However, after further consideration, I decided that the system could have been run smoothly enough on a Java built platform. By choosing Java, it means that the project could have been compiled to a .jar file and run on any machine with Java – arguably making it more interoperable.

4. Implementation

As I implemented the system, it was clear that the original UML design I produced (see 2.2.1.) was in need of improvement. As the systems development went on, the design slowly changed until it became what it is now (see 2.2.3.).

Some aspects of the implementation could still be improved. For example, in order to keep all of the bots in the canvas, I simply have them face the centre when they reach the edge. As a consequence of this the bots struggle to make shapes close to the edges of the canvas. Another flaw of this would be that this affects the validity of my swarm robotics system. In the 'real' world, the bots will not know if they have left the edges of the area and will not know where the centre is even if they did.

Another bug I noticed while implementing the LEDs is that the bots would not align properly to form a straight line. This issue was fixed by making the bots aim for the position that was 1 radius away from LED in the opposite direction to that LEDs bot. This meant that the bots would be in the correct position and simply had to face the bot before connecting.

My simulated bots are limited to only turning at a fixed rate or move forward at a fixed speed at any one time. This makes the bots perform manoeuvres more slowly however it leaves less opportunity for error.

I explored the possibility of implementing a kinematics model [14] into my bots so that they could turn and move forward at the same time, however I found that this did work well with the VFF algorithm as the bots often need to turn completely around while there other bots extremely close by.

4.1. The Classes

Here I will briefly cover the responsibility of each class in the system, and describe how it fits into the overall architecture of the project.

4.1.1. Simulator

This is the main class, located in the Main package. It is responsible for running the entire simulator. The bulk of the work was taken from Notches' Ludum Dares entry "MiniCraft" [15], which allows the entire project to run at a given tick, while maintaining the highest frame rate possible. It is worth mentioning that MiniCraft is open source.

4.1.2. GUIManager

The GUIManager is a relatively simple class located in the Gfx package. It is effectively the JFrame – or the 'window' – that contains the BotCanvas. It sets the size, scale, and properties of the window (i.e. if it's resizable or not).

4.1.3. BotCanvas

This is where the magic happens. It is a canvas filling the GUI JFrame that allows everything to be drawn at a triple buffered strategy. Located in the Gfx package, the BotCanvas has functionality to edit some of the bots' positions. When the CustomListener class reports a mouse action, the BotCanvas will set the appropriate bots' position to the cursor's position.

Other responsibilities include adding and removing bots from the canvas, electing certain bots as seeds, and rendering the instructional text in the top left of the window.

4.1.4. CustomListener

This class is responsible for listening and dealing with all user inputs, both mouse actions and keyboard presses by implementing the MouseListener, MouseMotionListener, and KeyListener classes. When a key is typed, it will instruct the BotCanvas on what to do respectful to what key was typed.

When the user clicks on the canvas, the closest bot within a range is selected and set to follow the mouse. This is how the 'click-and-drag' feature is implemented. When the mouse is released, the selected bot is 'placed' back onto the canvas.

This class is found in the Listeners package.

4.1.5. Bot

This is the main Bot class. It contains various other classes such as Sensors, LEDManager, and LEDs. All these classes work together through the Bot class to decide what this instance of the bot should do next.

This class is found in the Bot package.

4.1.6. Sensors

Each and every bot owns a sensor object. The object is responsible for simulating the hardware mounted on the bot that senses the environment. The bot class can request certain information from the sensor class such as directions and distances to bots, details on any close bots that it can see, and the position and orientation of the current bot.

This class is found in the Bot package.

4.1.7. LEDManager

LEDManager is another class owned by the bot. This class provides high level command functionality to the bot, and removes the requirement for the bot to micro-manage each of its LEDs. By splitting off some of the code from the bot and putting it in these *'helper classes'* the bot class becomes smaller, simpler, and easier to read.

High level commands include 'confirm connection', 'request bot on x LED', and 'turn off all LEDs'. When these commands are issued it is up to the LEDManager class to decide what LEDs should be set to what colour.

This class is found in the Bot package.

4.1.8. LED

Located in the Bot package, this class represents a single LED light that will be attached to the bot. Each bot will have several of these objects. Although it is the responsibility of the LEDManager to tell the LED what to do, it is the LED class' responsibility to render itself and remember what colour it currently is.

4.2. The Functions

4.2.1. Virtual Force Field (VFF)

I feel that the VFF algorithm deserves its own section because it has been a massive part to the implementation of my project; this algorithm proved to be extremely useful and solved a lot of the challenges with the functionality of the bots. With it, the bots can now avoid other bots while ultimately moving towards a desired point of contact. It also deals with the angle of approach beautifully, and even allows me to tweak the values making the bots distance of comfort to other bots increase or decrease [22].

For example, if I feel the bots are getting too close to each other I can simply make the bots have a lower negative charge. If I wanted the bots to be more attracted to an LED I could simply make that LED have a higher positive charge. The challenge is getting a good balance of them both, making bots move towards white LEDs whilst avoiding other bots at the same time.

The algorithm works by having objects in the world either 'push' or 'pull' you towards them. Objects to be avoided have a negative charge and therefore will 'push' you away, and objects with positive charges 'pull' you towards them. The farther from 0 the charge is the more the object pushes or pulls you.

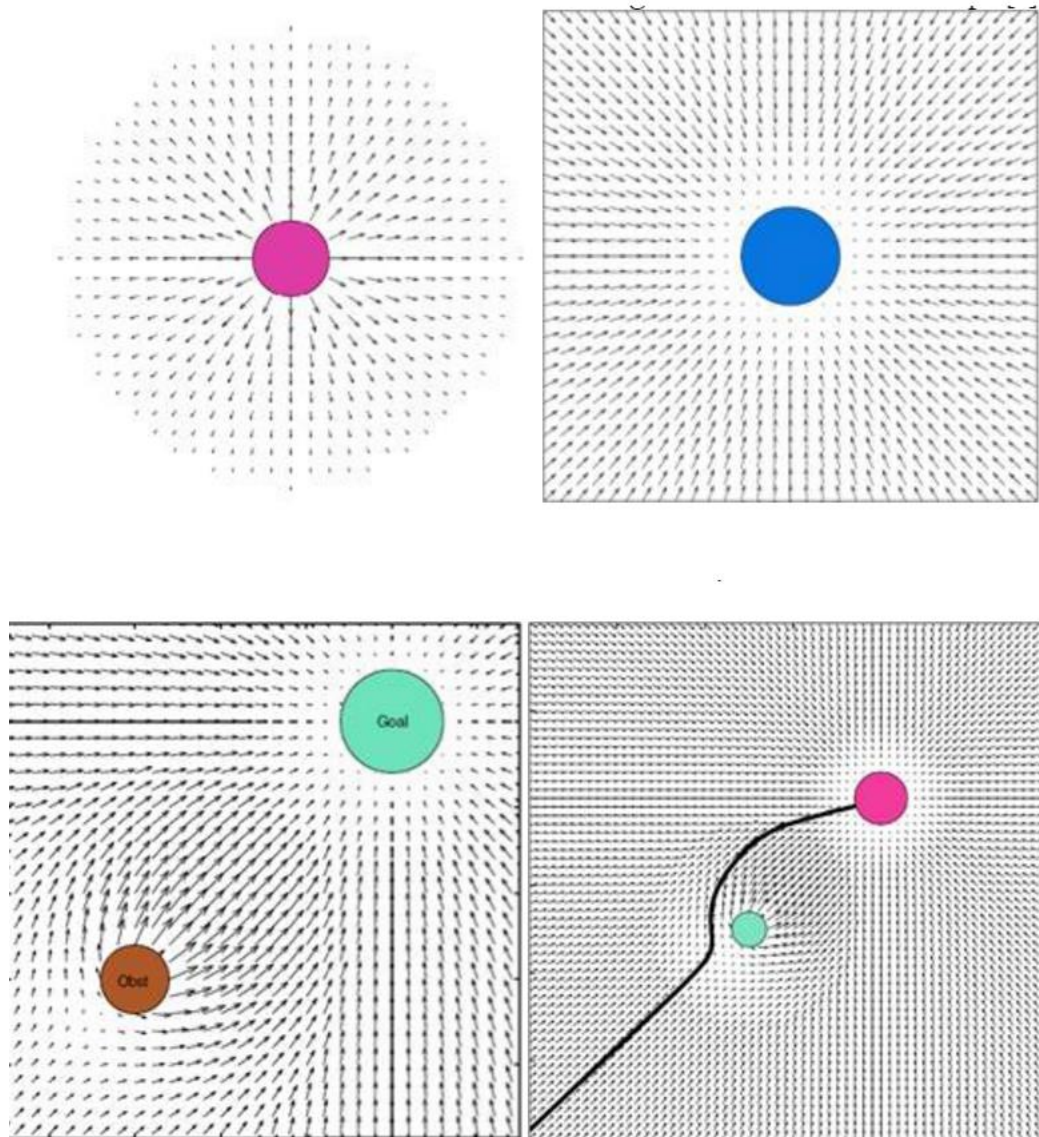


Image: VFF forces visualised, see [22]

Effectively, our bots are being pushed and pulled around the map. This can be visualised as a 3D gravity field representation (seen below) where repulsive (negatively charged) obstacles are represented by peaks in the field and attractive (positively charged) targets are represented as pits. You can imagine the bots as spheres rolling around this field unable to make it up the slopes whilst eventually falling into the target pits (note: the diagram has nothing to do with VFFs, and used purely to illustrate a point). The values of the negative and positive charges are scaled the closer the bots gets – meaning the as the bot approaches the object its charge is amplified, becoming more repulsive or attractive.

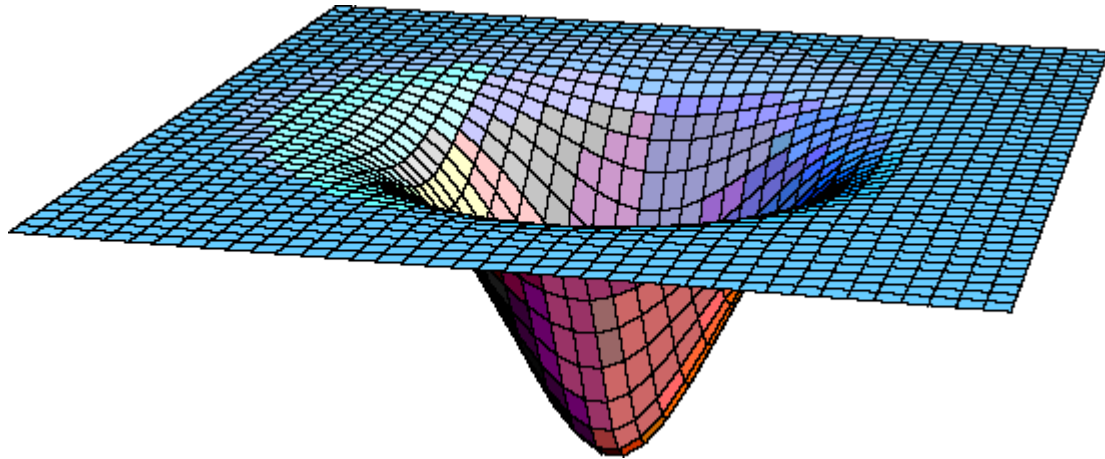


Image: Embedding diagram, <http://www.upscale.utoronto.ca/PVB/Harrison/GenRel/GenRel.html>

See Appendices C, Figure 9 for code snippet.

Our system is not perfectly suited for a VFF algorithm [28] and there are issues to be addressed. For example, bots that have an infinite direction of movement (ones that can move any direction at any time and not just forwards) make use of the algorithm more effectively. Also, the algorithm tells our bots what speed they should be going. The higher the attraction to a target the faster the bot should be travelling. As our bots are fixed to a single speed and direction at any one time, it is easy to conclude that the bots do not take advantage to the full use of the algorithm.

Another flaw is that our bots can both be considered a target and an obstacle. We want our bots to avoid other bots at all times, but also we want them to *sometimes* be attracted to a specific side of one, whilst avoiding the other side completely. This comes down to specific tweaking of values until a happy medium is met.

Finally, small gaps that are only just wide enough are not attempted by bots following this algorithm. In a tight space with many agents trying to get around, bots tend to move slower and more cautiously than needed.

So, we want bots to be repulsive to other bots. We also want specific LEDs to be attractive enough to draw a bot in without getting it to smash into anything it shouldn't. This can be a problem as sometimes large shapes of obstacles can form pushing our bot away too firmly for a single LED to pull back.

The answer is to make bots that are a part of a structure less repulsive than bots that aren't, making other bots less reluctant to getting close to them. I implemented this by making the green LEDs used to signify a completed connection have a small positive charge. There will be 7 green LEDs on each bot that is connected to a structure meaning the positive charge for each need to be $1/7^{\text{th}}$ of what it needs to be. As the target is to half the negative charge of the bot when it is connected, we have our values:

$$LEDCharge^{green} = \frac{Bot^{charge} * 0.5}{2}$$

Where:

$LEDCharge^{green}$ = the required charge of the green LED

$$Bot^{charge} = \text{the absolute value charge of the bot}$$

4.2.2. Directions

4.2.2.1. The Global Direction System

A universal direction system was used in the system in order to allow the sensors class to calculate directions relative to the bot. This global direction system can be thought of as a compass, but instead of facing 'north', you face '1'. The direction then rotates clockwise all the way around to 360 where it will then loop back around to 1. This means that instead of 'east' it's '90', and instead of 'south' it's '180', and instead of 'west' it's '270'. The direction is saved on the bot as a double value, and is initialised as a random number when the bot is created.

Some values calculated will be outside the range of 0 – 360. In these cases, Java's remainder operator (%) can be applied to normalise the value so that it once again lands in the desired range of 0 – 360.

4.2.2.2. Direction to a Bot

Directions are calculated using the following formula:

$$dir_{bot} = \frac{atan2(xPos_{relative}, yPos_{relative})}{\pi} * 180$$

Where:

$$dir_{bot} = \text{direction to the bot}$$

$$xPos_{relative} = \text{relative x position to the bot}$$

$$yPos_{relative} = \text{relative y position to the bot}$$

This will return a value between 0 – 360. It is important to account that direction 0 will face right on the screen. That is, 'north' = facing right, 'east' = facing down, and 'south' = facing left.

Finding the direction to a specific bot was needed in the program for several of the algorithms used, including Virtual Force Field (VFF) calculations and confirming that a bot has connected to one of the white LEDs.

See Appendices C, Figure 1 for code snippet.

4.2.2.3. Average Direction Finding

Calculating the average direction was originally used to avoid large groups of bots by going in the opposite direction to the calculated which direction was the most crowded ($\text{desired_direction} = \text{averageDirection} + 180$). Although later replaced by the VFF algorithms, the calculation of the mean

direction of bots was used throughout the development of the system. Essentially, I took the dot product of every vector of the seen bots.

The following formula was adapted from one found on a public forum [16].

$$BT_X = \cos \sum_{n=0}^n \frac{BX_n * 2\pi}{360}$$

$$BT_Y = \sin \sum_{n=0}^n \frac{BY_n * 2\pi}{360}$$

$$angle_{avg} = \frac{\left(\frac{atan2(BT_Y, BT_X) * 360}{2} \right)}{\pi}$$

Where:

BT_X = Total relative X positions of the seen bots

BT_Y = Total relative Y positions of the seen bots

$angle_{avg}$ = Average vector of seen bots

n = number of seen bots

See Appendices C, figure 2 for code snippet.

4.2.2.4. Turning to Desired Direction

Turning to a desired direction is not instantaneous. Instead, the bot can only turn towards its desired direction at a fixed rate per tick. The easy option would be to turn clockwise or anti-clockwise until the desired direction is reached, however to improve efficiency the bots decide which way is the shortest way to turn before turning.

See Appendices C, figure 3 for code snippet.

4.2.2.5. Moving in Direction Facing

The issue here is calculating the new position that is one unit of travel in front of where the bot is facing. In the real world these values do not have to be pre-calculated and can simply be

executed, however we are in a simulated and not a real world – and so, this is done by following the following formula:

Note: Math.toRadians() is built in Java method and is used instead of calculating the radians manually.

$$\begin{aligned} \text{radians} &= \frac{\pi}{180} * \text{Bot}^{dir} \\ xPos^{next} &= xPos^{current} + (\cos \text{radians} * \text{length}) \\ yPos^{next} &= yPos^{current} + (\sin \text{radians} * \text{length}) \end{aligned}$$

Where:

$$\begin{aligned} xPos^{current} &= \text{the current xPosition of the bot} \\ yPos^{current} &= \text{the current yPosition of the bot} \\ xPos^{next} &= \text{the xPosition of the bot after the move} \\ yPos^{next} &= \text{the yPosition of the bot after the move} \\ bot^{dir} &= \text{the direction that the bot is currently facing} \\ \text{length} &= \text{distance of travel by the bot} \end{aligned}$$

See Appendices C, figure 4 for code snippet.

4.2.2.6. Looking at a [X, Y] Position

This function will be used when we want our bot to face a certain position on the canvas. This will be used to face the direction calculated for us in the VFF algorithm, where it will give us the direction that it is ‘pushing’ us.

This is a relatively simple equation [17]:

$$dir^{target} = \text{atan2}(yPos^{rel}, xPos^{rel}) * \frac{180}{\pi}$$

Where:

$$\begin{aligned} dir^{target} &= \text{the direction the bot is aiming to face} \\ yPos^{rel} &= \text{the relative yPosition from the bot to the target} \\ xPos^{rel} &= \text{the relative xPosition from the bot to the target} \end{aligned}$$

See Appendices C, figure 5 for code snippet.

4.2.2.7. Calculating Distances

It is important for our bot to be able to tell how far away something is to it in order to calculate the VFF force. In the real world, this would be done by a clever image interpolation algorithm however for our simulated system we will stick to relatively basic vector trigonometry. Pythagoras said it himself:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Where:

$$d = \text{distance between } (x_1, y_1) \text{ and } (x_2, y_2)$$

See Appendices C, figure 8 for code snippet.

4.2.3. LEDs

4.2.3.1. Meaning of the LEDs

Different LED colours have different meanings and are used to signal different things. Each LED has 5 states; red, green, blue, white, and off.

RED means that the bot has at least one connection still open (that is, at least one of its LEDs is still white). This is for bots that have already connected to it. A newly connected bot will turn blue confirming its connection until the bot it has connected to no longer have any red LEDs on, meaning it has confirmed all of its connections and is no longer waiting for any other bot to connect.

GREEN LEDs appear when the bot is already a part of a shape. It has already successfully connected to a bot and has a bot successfully connected to it. Green LEDs have a small positive VFF charge (explained in section 5.4.) meaning other bots are less reluctant to get close to them.

BLUE means that the bot has connected to another bot and is signalling confirmation to the other bot that it has done so. When a bot is waiting for another bot to connect, it is looking for a blue LED in the general area that the bot is expecting.

WHITE is a signal for a bot to connect. When roaming, the bots are effectively looking for a white LED to connect to as it signals an open connection. When a bot wants another bot to connect, a white LED will illuminate in the direction that is pending a connection.

OFF simply means none of the above. These LEDs will be completely ignored by other bots.

4.2.3.2. Updating the LEDs

The LEDs are calculated independently with each tick. That is, the previous state of each LED isn't taken into account when updating them. Below is the logic that is followed when deciding what colour the LEDs should be. This is decided in the Bot class.

Firstly, if there are no close bots that are waiting for a bot to connect to it (i.e. has a white LED on), the bot will turn all of its LEDs off as it will need to continue roaming around.

Otherwise, if the bot has connected to another bot and that bot has not yet turned green (i.e. the bot is still waiting for bots to connect to it), then turn blue, signalling you have connected to it.

Otherwise, if we are waiting for a bot to connect to us and still has at least one white LED on, then turn off any white LEDs remaining that has a bot signalling blue (i.e. that they have connected to us) in the direction that white LED is facing.

Otherwise, if our bot is in position to connect to another bot, save that other bot as the bot we are connected to and set our desired direction to face it.

Otherwise, if the bot we have connected to has its LEDs set to green (i.e. it has confirmed the connection), and we have our LEDs to blue (i.e. we are confirming a connection to someone), request a connection from behind by setting all LEDs to green and the rear LED to white.

See Appendices C, figure 6 for code snippet.

4.2.3.3. Closing the Open Connections

When a bot has at least one white LED on, it means that it is still waiting for bots to connect to it. The bot checks each of its open connections by determining if there is a blue LED that is both close enough for a connection and in the direction of one of its white LEDs. If this is found to be true, the bot assumes another bot as connected to that particular white LED and turns it off. This is repeated until all white LEDs are turned off before it sets all of its LEDs to green.

See Appendices C, figure 7 for code snippet.

4.2.4. Rendering

4.2.4.1. Bots

Bots are rendered as simple circles on the canvas, and are drawn in such a way so that the centre of the circles is where the bots position is taken. By doing it this way (as opposed to taking the position as the top left of the circle as Java seems to prefer) it makes it easier to determine if the bots have made contact with another bot. It is easy to determine if contact has been made between two bots by simply checking that the distance between the two bots' positions is less than or equal to the diameter of the bot.

Bots rendered green are telling us that they cannot see any other bot, and therefore are currently roaming randomly. Bots rendered yellow are bots that can see at least one other bot, and finally bots rendered red are bots that are close enough for another bot to connect.

Two lines can be seen inside of the bots; the longer of which is the direction that the bots are currently facing. The second, shorter line is the desired direction of the bots. This cannot be determined by other near bots and is used only to help visualise what the bots are doing.

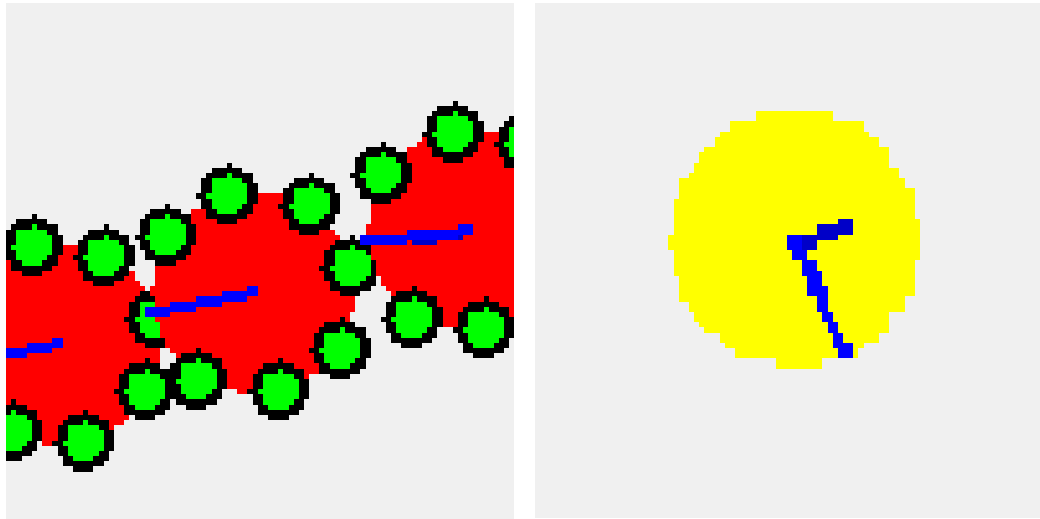


Image: Close up of the visual representation of some bots (as of version 1.3)

LEDs are rendered as a black ring with its current light colour filling the outer circle. If the LED is off, nothing is rendered. Like bot bots, the LEDs position is taken from the centre of the circle.

4.3. The Behaviour of the Bots

The bots are stateless – that is, there is no need to save the bots previous position, orientation, or LED setup in order to determine what to do next. A system like this makes the bots far more rugged to unknown environments and events.

The bots begin by gathering data on their surrounding by requesting all bots that are close to it from its respective Sensors class, which will return an array list of pointers to all close bot. After that, if any of the bots that our bot has spotted has a white LED on (colours explained in section 4.2.3.1.), it will save the closest white LED for later use.

The bot then updates its LEDs - this process is further explained in section 4.2.3.2. After our LEDs are set to the appropriate colours, it is time for our bot to consider moving. If any of the LEDs are set on it means that we are in a position that we want to be and do not need to move, and only turn towards the desired direction if we are not already facing it. If all of the LEDs are switched off, then we will either move forward if we are facing the right direction OR we will turn towards that desired direction.

This process is repeated every tick, under the condition that the bot is still set to 'active' (i.e. its 'isActive' variable is set to true). The only time the bots will be inactive is when I am moving them with my mouse. This is a feature purely for testing purposes and will not be needed in a simulation involving real bots.

4.4. Comparing with the Specification

Originally, the specification aimed for a system with a minimum functionality to form the most basic shape possible; a simple line. It left plenty of room for improvement such as getting the bots to form multiple shapes, simulating noise, and a more advanced algorithm for self-detection. Although most of these specifications were ended up being realised, there is further room for improvement in the system I have produced. All of these specifications have been met with the exception of noise.

The simulator produced also provides a clear GUI that allows me to clearly see the properties of the bots – position, direction, LED state... - and easily allows me to elect seed bots, move the bots around the canvas, add new bots, and remove existing ones.

Because of this, I am happy with how the final system compares with the original specification. I was able to keep with the time schedule I set myself at the beginning and managed to complete *most* of the specifications. However, given more time on the project, noise would have been implemented by adding a slight random offset to the values the sensor class returns. The size of this offset will increase the further away the bot is.

By implementing noise I could have performed tests that determine how well my system can cope under different levels of distortion. The level of noise would have slowly increased to see how high it can get before the bots can no longer form a shape correctly. Such a test would have provided a minimum performance requirement for the hardware on the bots, as we would have known how accurate the data has to be in order to allow the bots to successfully connect.

5. Testing

5.1. Overall Approach to Testing

Testing for these kinds of projects is mainly visual; most of the time if the bots appear to be working then they usually are. Some projects require the comparison of numbers or testing an output for acceptability, I would assume my robots were working correctly simply if they appeared to. That's not to say that I did not use a numeric value to test my project. As you will read in section 5.3., I measured the time it took for the bots to form given shapes under different starting conditions.

There were times that the bots appeared to work correctly however were not. For example, when I implemented the functionality to detect the relative direction a seen bot is to a bot (section 4.2.2.2.), it appeared to work correctly. It was only later on when I discovered a bug which meant I had to backtrack and fix it.

This system of testing, it seems, is not flawless. As my system is in no way safety critical (a bot crashes, so what! Start the simulation again!) I do not regret my testing methodology as I feel the time saved with only testing visually paid for the time spent fixing bugs that would have been picked up with further testing. Because of this, I would not change my approach to testing on similar projects in the future.

5.2. Stress Testing

It is important to remember that my system is simply a research tool and does not need to be able to cope under stress. If it were to ever be properly implemented using real S-Bots, the system will not have to process every bot and drawing them, but instead the bots will be thinking individually and independently. That being said, after completing a version, many bots were added to ensure that the frame rate would remain smooth at higher tick rates. I would add 300 bots ticking at 120 ticks per second and visually judge if the bots were rendering too slow.

A more important type of stress testing would be observing the bots under cramped conditions – that is, when there isn't much space on the canvas.

The first test was of the bots' ability to avoid collision without any seed bots on the canvas. After that, I added 300 Bots to the canvas and timed how long they took (at 60 ticks per second) to all get out of each other's way. As expected, it took longer for all bots to be collision free as the average space per bot on the screen decreased. The important thing I was looking out for was that no bots were getting stuck in each other, and at all times the bots were at least attempting to get into free space. On the whole, I was extremely happy with the bots performance with collision avoidance. The results of the test can be seen below:

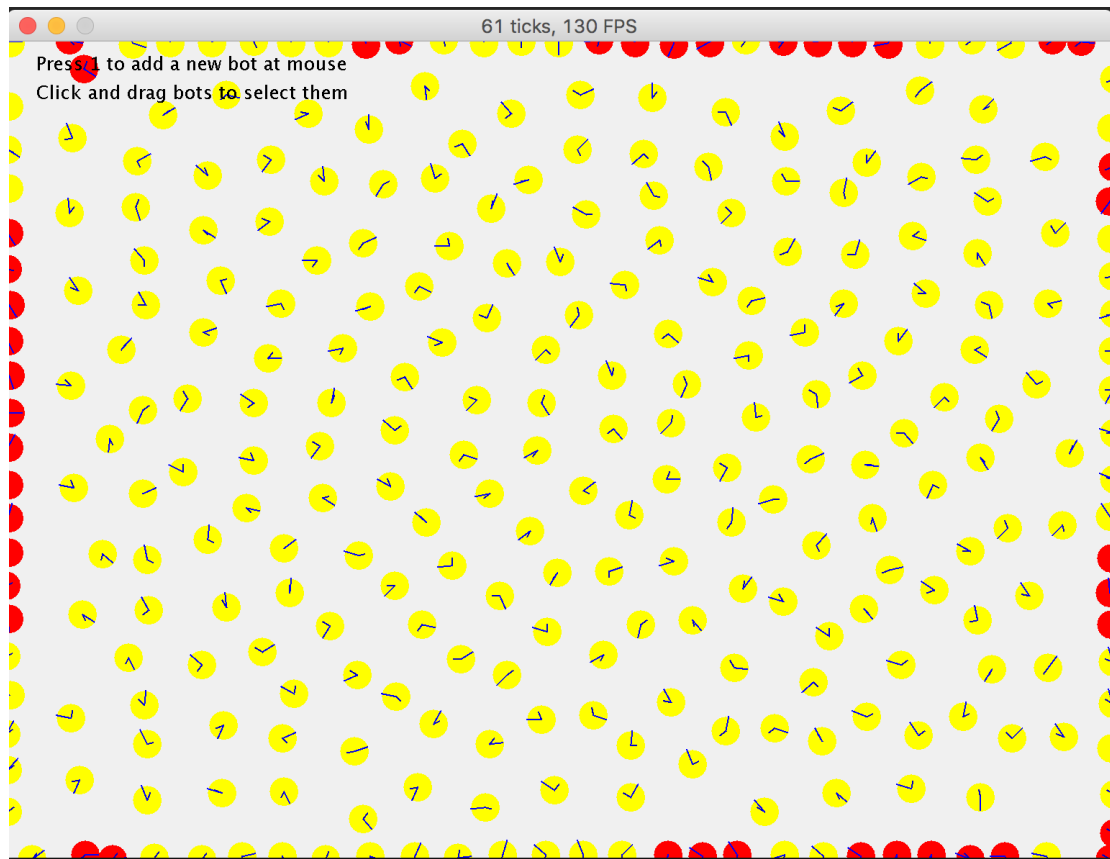


Image: 300 bots on the canvas with no LEDs on

Bots that come closest to colliding are those close to the edge. As the system stands, when a bot leaves the canvas it is told explicitly above everything else to turn and face the centre of window. This is just a feature of the simulator as an attempt to keep the bots in the bounds of the frame so that they remain in sight.

The same test was repeated with a larger canvas and 500 bots. The tick rate was lowered to 10 ticks per second to help the simulator keep up with the massive load.

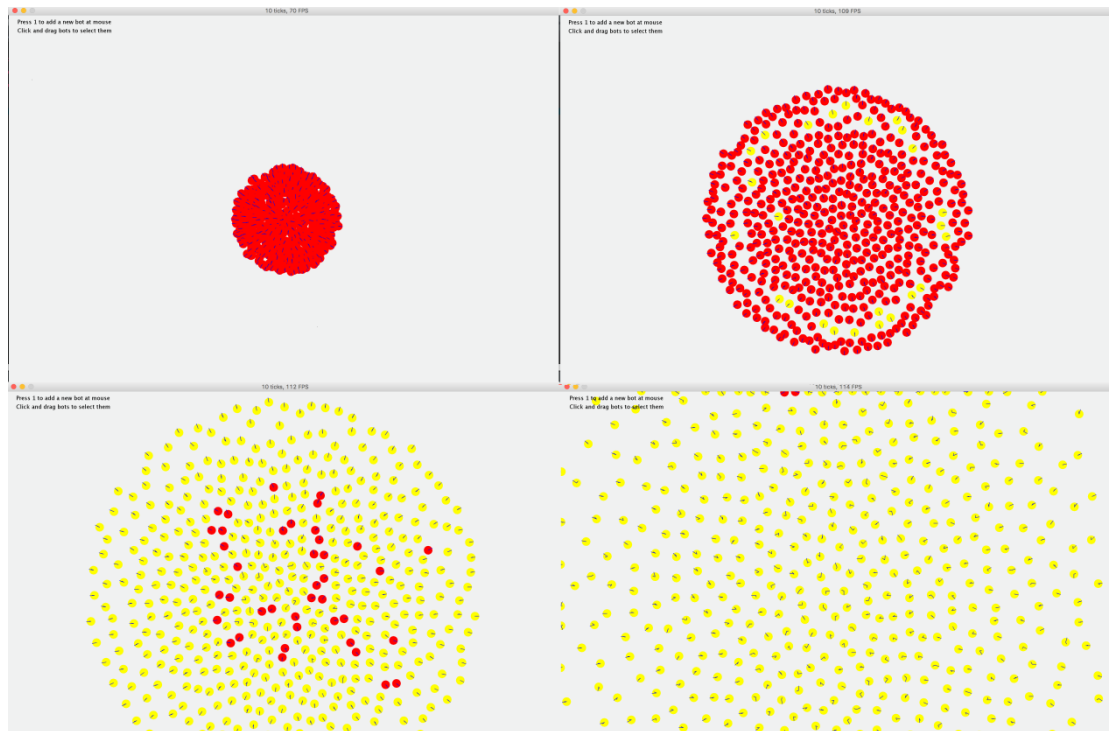


Image: 500 bots dispersing around the canvas at $t = 1s$, $t = 5s$, $t = 10s$, $t = 15s$.

The second test was of the bots' ability to adequately form a shape under cramped conditions. Again, 300 bots were added to the canvas that was running at 60 ticks per second and I timed how long it took for bots to connect with each other. Surprisingly, when there are more bots on the canvas they connect at a higher rate. However, unfortunately the VFF algorithm seemed to struggle with collision detection when two or more bots were already close to an LED. I found that if the bots are very close to their target they would go for it even if another bot is in the way. Ignoring the collisions, bots that are extremely close to each other when a connection opens tend to connect less straight than ones that have to travel, resulting in a skewed line. Fortunately these problems only occur when the canvas is extremely cramped.

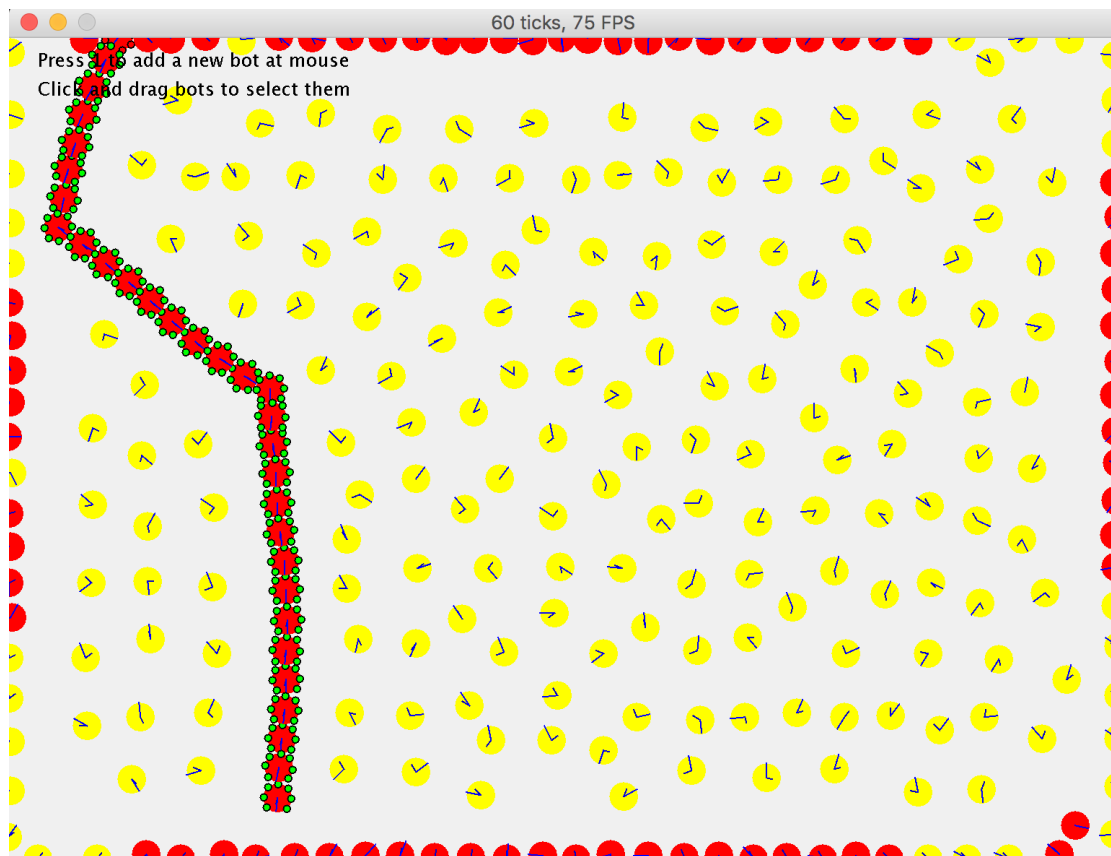


Image: 300 bots attempting to form a straight line under high population density

As it stands, the simulator forces bots to face the centre of the canvas when they leave its edges. While this was a good way to make sure bots stay in view and would not be as big as a problem in the real world, it does cause for some problems when the canvas is full. As seen in the diagram below, it would cause bots to accumulate at the edges of the canvas making connections near the perimeter nearly impossible.

5.3. VFF Testing

The VFF algorithm has several values that need tweaking in order to suit the system; the first of which is the charges of each bot and their respective LEDs. I needed to make the bots charge negative enough to keep other bots from colliding but not too negative to stop any bots wanting to connect to it, and believe me this was easier said than done.

Through trial and error, I found that this can be achieved best by setting the push force of the bots' negative charge to less than half as strong as the pull force of the LEDs' positive charge. The values used are -1.5 and 4.0. Another problem arises when bots are already in a shape. There becomes more bots to push but still only one LED to pull the bot to a connection. To tackle this problem the bots need to be less reluctant to get close to other bots that are already in a connected structure.

When the bots are fully connected in part of a structure, green LEDs are displayed confirm the connection. By treating each of these LEDs as holding a small positive charge, the combined pull force will draw bots closer and allow connections to happen. In testing, this worked beautifully. The bots do

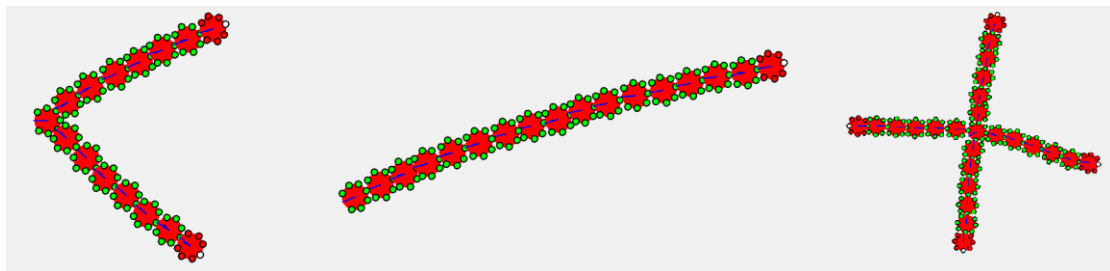
not move and are therefore still repulsive enough to void collision; however other bots are not scared to get close to them anymore.

5.4. Acceptance Testing

After each version was completed the project was compiled into a .jar and tested to see if all of the bots behaviour is working correctly. After I have confirmed so I would deem the version 'complete', make a backup, and begin work on the next version. This does not include performance; I would only test if the version *worked* and not if it worked *efficiently*.

When testing the systems performance I did find one or two problems. Firstly, on rare occasion, when there are two bots are roughly equal distance to a target they will 'argue' and take a long time to decide which bot would go in for the connection. If those bots are already close to the target (less than 1 bots length) and roughly equal distance to it they will sometimes both attempt a connection, which causes a collision.

These are rare situations to be in if the canvas is not over crowded, and therefore I will accept these problems in the final release. If I were to ever need to fix it (perhaps for implementing a safety critical system), I would look to the VFF algorithm – specifically where it scales the push and pulling forces as the bot gets closer.



Images: Bots forming different shapes successfully (Arrow, Line, Cross)

5.5. Performance Testing

Ignoring the systems functional performance, it is important to test not only if the bots do the job, but also how well they do it. To do this, I would fill the canvas with different amounts of bots and test to see how long certain shapes would take to form. I would consider groups of agents that were able to form a shape faster to be more successful in their performance. Here I am not considering *how straight* any structures produced to be, only recording the time they take to form.

Below is a table (screenshot from MS Excel) of the time it took for my system to produce certain shapes.

Number Of Bots	Shape Being Formed	Time (seconds)			Mean Time	Range
		Test 1	Test 2	Test 3		
10	Line	44	101	92	79	57
20	Line	50	89	160	99.666667	110
50	Line	105	120	90	105	30
100	Line	78	97	155	110	77
10	Arrow	136	78	138	117.333333	60
20	Arrow	125	122	62	103	63
50	Arrow	79	104	121	101.333333	42
100	Arrow	71	147	67	95	80
10	Cross	77	124	130	110.333333	53
20	Cross	41	75	119	78.333333	78
50	Cross	88	62	41	63.666667	47
100	Cross	104	97	113	104.666667	16

No demonstrable time pattern emerges when changing the number of bots in the system or the shape that the group is trying to form. The more bots on the canvas, the closer they must spread in order to stay away from each other which meant I that the bots would tend to connect at a higher rate when there are more roaming on the canvas – which seems to be due to a faster rate of discovery. The last bots to connect always take the longest to discover, which may indicate work on the random roaming may need work if I ever wanted to increase the speed of the system.

6. Critical Evaluation

6.1. Research Evaluation

I was able to find a lot of extremely helpful and relevant research and projects on swarm robotics during the early stages of design, and managed to learn a lot about Swarm Robotics as a whole as well as current systems and challenges still being developed. This made it extremely easy to decide on what the focus was going to be in my project.

Originally I had planned to implement a system that would simulate drones orbit a specified point in space while maintaining an equal distance to its neighbouring bots, however researching the problem and only managing to find a single paper [18] addressing it, and they cheated by using an overall controlling computer micromanaging each bot, suggested that the level of difficulty required more attention than a bachelor degree student could give it. My project was therefore simplified to having bots connect themselves to the other bots to form a shape.

The amount of research seemed to peak early on in the project. A lot of papers were being read when I was still forming ideas on what my project would specifically be about. As the project went on, less and less papers were being read on the current state Swarm Robotics as the focus shifted to more mathematical material explaining certain algorithms such as real time obstacle avoidance [19].

Despite the desire to simply get stuck into the code of the project as the features were already decided, I aimed to read at least one research paper on Swarm Robotics per week. The ultimate goal of this project was to expand my knowledge of swarm robotics, and this is done both in practice and in theory.

6.2. Specification Evaluation

The requirements I gave myself were, I feel, both challenging and achievable. To complete most of them, a lot of reading and research had to be done and a large proportion of the maths used was completely new to me meaning not only did my skills in programming and research, and my knowledge in Swarm Robotics improve but also my general understanding of vectors, shapes, and mathematical algorithms.

Not only did I write a full list of features that needed to be in my project, I also tried to plan out the entire structure of the project before even starting any of the code. This gave me a comprehensive list with all of the features and tasks in, as well as where they go and how they should be implemented. Although only a rough list that was susceptible to change, it meant that I was never lost on what to do. In larger projects I sometimes find that I lose motivation and inspiration to think about what needs to be done next, and having a 'to-do' list helps avoid this.

The initial list of features was surprisingly complete considering they were written so early on in the project. Although some were tweaked, such as the bots being elected as a seed bot rather than finding a spot on the canvas that elects them a seed, the structure seemed to stay consistent as time went on. I feel that I managed to correctly identify the features that my project needed to do before starting the code, which in the seemed to help make the development smoother.

As already mentioned prior in the report, two specifications were written; the first of which was considered a 'minimum' set of specifications. If these features were not implemented the project would have deemed incomplete for failing to meet its initial intended purposes. The second list of specifications was only to be implemented once the first list had been completed, and was never intended to be finished. These were the further features that were only to be implemented in the presence of spare time.

As the system stands, the bots can only make shapes involving straight lines (line, arrow, and cross). At the beginning, I would have hoped that my final system could have produced more interesting shapes than the ones it can currently manage. Ideally, I would have liked the bots to be able to organise themselves into a block or even a circular shape.

I am very happy with the specifications I initially wrote and therefore if I were to start the project again and have to re-write my specifications, not a lot would change. Perhaps I would have made noise a higher priority on the further features list that I wrote. Also, as it was initially planned, each version of the project should have taken roughly the same time. This turned out to be slightly untrue, and some versions ended up taking longer than others. This resulted in longer time frames between acceptances testing, which held risk to larger portions of work becoming unusable.

For example, one of the versions had a task to have the bots be able to calculate the average vector to other bots. After hours of working on this, the next version of the code had me implementing a system to go towards bots with a white LED on. This is where I discovered the Virtual Force Field system, which allowed the bots to avoid other bots while being attracted to other bots at the same time, therefor rendering the previous section of code obsolete. If I were to redo the specifications, I would put more time into thinking about the size of each version, and grouping the tasks in those versions together.

6.3. Process Evaluation

Although it doesn't tick all of the boxes, the project was planned out using a methodology similar to feature driven. Versions were planned out, and with each version a list of features or tasks were written. As I developed the system, any changes and bug fixes were logged in the text file containing all of the versions. This process has been described in further detail in section 6.3.

An agile process was definitely the process that I needed to follow as the specification had a list of features in descending order of importance that I would work through until a point was reached where the report was needed to be written.

No sprints or version deadlines were used. My management of time rested entirely on seeing how much time I had left and comparing it with how much I have managed to get done. If I felt I was behind, more hours were thrown at the project, and luckily the problem always seemed to get fixed in an appropriate time frame. Although time management was not an issue with this particular project, my future work will have a more structured time schedule making it easier to estimate when I will finish.

Admittedly, I was less organised with the report and simply decided to tackle the entire document after I had completely finished writing the program. I found that talking about the beginning of my project was difficult as I had to remember something I did and how I did it that happened months prior.

Going through the process of creating a piece of software as large as this has definitely taught me the importance of organisation. Future project reports will be written section by section as the content is completed, as well as having a well revised structure plan of the code.

6.4. Product Evaluation

While the VFF system worked well when one LED was on and managed a lot of the problems that needed tackling, it was not perfect. For example, there are instances where bots would appear to 'argue' with each other over which one of them will take the plunge and connect to a white LED. This would occur most commonly when there are two bots at equal distance to the target.

Tweaking VFF values helped avoid this however the system still occasionally argues with itself, slowing the speed that the group can connect (discussed further in section 4.2.1.). Because of how rare this occurrence is, and that the bots eventually settle their differences and get on with it, I accept this as a known issue and I am willing to accept it in the final release of my system.

As the system is more of a research tool and never intended for use by more than a handful of people, not much needs to be said about the GUI. I personally find it easy to use and clear to see what is currently happening in the simulation, which is more than enough to pass its acceptance test. Improvement could be made but it would simply be a waste of time.

The program is extremely interoperable as it can be used on any machine with access to a JVM. Also the file size is small enough to send in an email which makes it convenient to store.

All of the further features proposed in the specification were implemented, with the exception of noise. Though I am extremely happy with what I have managed to achieve in the time frame, I still regret the absence of correctly implemented noise in my system as such a feature would have been essential if I wanted to simulate my work on real S-Bots. By implementing this feature, I would have also been able to test my system under different levels of noise in order to determine the minimum resolution of accuracy that the hardware would need for my software to work on real bot.

One last issue that I feel needs to be addressed with the product is that as they stand there is no that the bots will be able to differentiate an LED that belongs to them and an LED that belongs to another bot. As it stands, the simulation automatically rules out themselves from the list of seen bots that is return from the sensors. To implement the software in real S-Bots, they will have to develop the ability to disregard themselves from any decision making.

6.5. Conclusion

A lot has been said on potential ideas involving building, search and rescue, and planetary exploration; however, most of it has yet to be implemented due to a lack of innovation in the technology. These ideas remain as proof of concept in labs and simulators only.

My system only tackled a small and specific part of Swarm Robotics. The research I conducted and the system I produced has taught me a great amount including the importance of organisation when building software. Proper planning has made time management and system design a lot smoother and has proven to be an essential part of the development of any project.

I feel my project was successful as both a research and software engineering point of view. I managed to learn a lot about the growing industry of Swarm Robotics during the design, research, and implementation, and I feel my skills in mathematical algorithms and programming have been strengthened. The system was completed with only a few of the further features not implemented.

For future projects, I will be sure to allow as much time for planning the report as I do writing it, and be sure to document what I have done as I do it as opposed to leaving it all to the end. As someone who struggles with getting their thoughts down on paper in a fluid way, I would say that the report was definitely the most challenging part of the project for me.

I have extremely enjoyed this project and hope to have the opportunity to work on systems such as this one in the future.

Appendices

A. Third-Party Code and Libraries

Eclipse was used as an IDE for the development of my system.

Code from Markus Perssons "MiniCraft" [15] was used in the allocation of ticks and renders.

Websites such as StackOverflow.com (for syntax checking) and Creately.com (for diagram creation) help throughout the development.

B. Ethics Submission

AU Status Undergraduate or PG Taught

Your aber.ac.uk email address rlj10@aber.ac.uk

Full Name Rhydian Jenkins

Please enter the name of the person responsible for reviewing your assessment. Elio Tuci

Please enter the aber.ac.uk email address of the person responsible for reviewing your application
elt7@aber.ac.uk

Supervisor or Institute Director of Research Department cs

Module code (Only enter if you have been asked to do so)

Proposed Study Title Swarm Robotics

Proposed Start Date 02/2016

Proposed Completion Date 04/05/2016

Are you conducting a quantitative or qualitative research project? Mixed Methods

Does your research require external ethical approval under the Health Research Authority? No

Does your research involve animals? No

Are you completing this form for your own research? Yes

Does your research involve human participants? No

Institute IMPACS

Please provide a brief summary of your project (150 word max) Metamorphism in a Swarm Robotics system. I will simulate Robots organising themselves with no external commands to make shapes on a computer screen.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material? Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data? Yes

Does the research pose more than minimal and predictable risk to the researcher? Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to? No

Please include any further relevant information for this section here: *null*

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one. Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change. Yes

Please include any further relevant information for this section here: *null*

C. Code Samples

```
/* GET DIRECTION TO A BOT */  
protected double getDirectionTo(Bot bot) {  
    double dir = Math.atan2(bot.getY()-getYPos(), bot.getX()-getXPos()) / Math.PI * 180;  
    return fixDirection(dir);  
}
```

Figure 1. Finding the direction to a bot.

```

protected double getAvgCloseBotDirToBots() {
    // found at http://www.pcreview.co.uk/threads/mean-direction-through-360-degrees.1746638/
    if (closeBots.size() == 0) {
        return sens.getDirection();
    }

    double cosDir = 0.0;
    double sinDir = 0.0;
    ArrayList<Double> cosDirs = new ArrayList<Double>();
    ArrayList<Double> sinDirs = new ArrayList<Double>();
    double sumCosDirs = 0.0;
    double sumSinDirs = 0.0;

    // calculate cosDirs and sinDirs
    for (int i = 0; i < closeBots.size(); i++) {
        cosDir = Math.cos(sens.getDirectionTo(closeBots.get(i)) * 2 * Math.PI / 360);
        sinDir = Math.sin(sens.getDirectionTo(closeBots.get(i)) * 2 * Math.PI / 360);

        this.fixDirections();

        cosDirs.add(cosDir);
        sinDirs.add(sinDir);
    }

    // Calc sums of cosDirs, sinDirs
    for (int i = 0; i < cosDirs.size(); i++) {
        sumCosDirs += cosDirs.get(i);
    }
    for (int i = 0; i < sinDirs.size(); i++) {
        sumSinDirs += sinDirs.get(i);
    }

    // Return avg dir
    // double returnVal = Math.atan(sumSinDirs/sumCosDirs)*360/2/Math.PI;
    double returnVal = Math.atan2(sumSinDirs, sumCosDirs) * 360 / 2 / Math.PI;
    return returnVal;
}

```

Figure 2. Calculating the average vector (dot product) of all close bots.

```

/* TURNS BOT TOWARDS DESIRED DIRECTION BY AMOUNT GIVEN */
private void turnToDesiredDirection() {

    double dir = sens.getDirection();

    if (dir < desiredDirection) {
        if (Math.abs(dir - desiredDirection) < 180) {
            dir += TURNSPEED;
        } else {
            dir -= TURNSPEED;
        }
    } else {
        if (Math.abs(dir - desiredDirection) < 180) {
            dir -= TURNSPEED;
        } else {
            dir += TURNSPEED;
        }
    }

    sens.setDirection(dir);
}

```

Figure 3. Calculating the best way to turn and turning the bot towards its desired direction.

```

/* MOVES THE BOT FORWARDS IN DIR FACING BY GIVEN AMOUNT */
private void moveForward(double length) {
    sens.setXPos(sens.getXPos() + (Math.cos(Math.toRadians(sens.getDirection())) * length));
    sens.setYPos(sens.getYPos() + (Math.sin(Math.toRadians(sens.getDirection())) * length));
}

```

Figure 4. For the simulator only; calculating a bots' new position after it has moved forward a given distance in the direction it is facing. Updates the bots' new position.

```

/* SETS THE DIRECTION OF THE BOT TO FACE THE GIVEN X, Y POINT */
private void lookAt(double x, double y) {
    // http://www.gamefromscratch.com/post/2012/11/18/GameDev-math-recipes-Rotating-to-face-a-point.aspx
    desiredDirection = Math.atan2(y - sens.getYPos(), x - sens.getXPos()) * (180 / Math.PI);
    fixDirections();
}

```

Figure 5. Setting the desired direction of a bot to look at a specified position on the Canvas.


```

/* SETS ALL LEDS TO THE APPROPRIATE COLOR */
private void updateLEDs() {
    /*
     * LOGIC
     * if no close bots and not waiting for bot to connect, set all LEDs off as you need to be roaming around
     * if connected to a bot and that bot is not yet GREEN, turn BLUE
     * if there are WHITE LEDs on, turn off any WHITE LEDs where there are blue bots
     * if it has reached a bots white LED, connect to it and look at it
     * if the botConnectedTo has turned its LEDs GREEN and our LEDs are BLUE, request connection for other bot behind
     */

    // if no close bots and not waiting for bot to connect, set all LEDs off as you need to be roaming around
    if (closeBots.size() < 1 && !LEDManager.hasLEDSetTo(Color.WHITE)) {
        botConnectedTo = null;
        this.LEDManager.turnOffAllLEDs();
    }

    // if connected to a bot and that bot is not yet GREEN, turn BLUE
    if (botConnectedTo != null && !botConnectedTo.LEDManager.hasLEDSetTo(Color.GREEN)) {
        this.LEDManager.setAllLEDs(Color.BLUE);
    }

    // if there are WHITE LEDs on, turn off any WHITE LEDs where there are blue bots
    if (LEDManager.hasLEDSetTo(Color.WHITE)) {
        sens.turnOffWhiteLEDsIfBlueBotConnected();

        // if no WHITE LEDs remain, turn GREEN
        if (!LEDManager.hasLEDSetTo(Color.WHITE)) {
            LEDManager.setAllLEDs(Color.GREEN);
        }
    }

    // if it has reached a bots white LED, connect to it and look at it
    if (closestWhiteLED != null && sens.isCloseTo(closestWhiteLED.getTargetX(), closestWhiteLED.getTargetY(), 2)) {
        botConnectedTo = closestWhiteLED.getOwner();
        this.lookAt(botConnectedTo);
    }

    // if the botConnectedTo has turned its LEDs GREEN and our LEDs are BLUE, request connection for other bot behind
    if (botConnectedTo != null && botConnectedTo.LEDManager.hasLEDSetTo(Color.GREEN) && this.LEDManager.hasLEDSetTo(Color.BLUE)) {
        LEDManager.requestConnection("b");
    }
}

```

Figure 6. Deciding what the colours of each LED needs to be and setting them.

```

/* TURNS OFF ANY WHITE LEDS THAT HAVE A BLUE BOT NEXT TO THEM (MEANING THEY HAVE BEEN CONNECTED) */
protected void turnOffWhiteLEDsIfBlueBotConnected() {
    // if no WHITE LEDs are on, return
    if (!this.currentBot.LEDManager.hasLEDSetTo(Color.WHITE)) { return; }

    // if no BLUE LEDs found, return
    ArrayList<LED> closestBlueLEDs = getClosestBlueLEDs();
    if (closestBlueLEDs.size() < 1) { return; }

    /* WE HAVE A POTENTIAL CONNECTION NOW */

    // go through each white LED on our bot and set off if looking at closestBlueLED
    for (int i = 0; i < currentBot.LEDs.size(); i++) {
        // for each white LED that we have on...
        if (currentBot.LEDs.get(i).getColorString() == "white") {
            // go through all close blue LEDs seen
            for (int j = 0; j < closestBlueLEDs.size(); j++) {
                // if that white LED is in the direction of any of the closestBlueLEDs' owners...
                if (isInDirection(closestBlueLEDs.get(j).getOwner(), currentBot.LEDs.get(i).getDirToParentBot())) {
                    // turn that white LED red, as there is a bot there thats blue
                    currentBot.LEDs.get(i).setColor(Color.RED);
                }
            }
        }
    }
}

```

Figure 7. Deciding whether a bot has connected to any of the LEDs. The method also turns the LEDs off when a bot was found to be connected.

```
/* GETS DIST FROM BOT TO OTHER BOT */  
protected double getDistanceTo(Bot bot) {  
    double xDiff = this.getXPos() - bot.getX();  
    double yDiff = this.getYPos() - bot.getY();  
    double dist = Math.sqrt(xDiff*xDiff + yDiff*yDiff);  
    return dist;  
}
```

Figure 8. Calculating the distance to another bot.

```

/* UPDATES THE DESIRED DIRECTION DEPENDING ON VFF RESULTS */
private void updateDesiredDirection() {
    /* http://www.cs.mcgill.ca/~hsafad/robotics/, accessed 28/02/16 */

    // First, set all seen bots' charge to -VE, all on LEDs to +VE, all off LEDs to 0.0D
    populateCharges();

    // VARIABLES
    double dirX = 0.0D; // relative dirX and dirY that the bot will need to face
    double dirY = 0.0D;
    double dx = 0.0D; // used to calculate each push/pull force, added to dirX, dirY
    double dy = 0.0D;
    double targetCharge = 0.0D; // charge of current target bot / led
    double minS = 50;
    double distSq = 0.0D; // distance^2 to bot/led
    double safety = 0.0D;
    double norm = 0.0D;

    for (int i = 0; i < seenCharges.size(); i++) {
        try {
            // now, update direction depending on charges
            if (seenCharges.get(i) instanceof Bot) {
                targetCharge = ((Bot) seenCharges.get(i)).getCharge();
                distSq = sens.getDistanceTo((Bot) seenCharges.get(i));
                distSq = distSq * distSq;

                // calculated forces
                dx = targetCharge * (((Bot) seenCharges.get(i)).getX() - this.getX()) / distSq;
                dy = targetCharge * (((Bot) seenCharges.get(i)).getY() - this.getY()) / distSq;

                // add calculated forces to overall direction so far
                dirX += dx;
                dirY += dy;

                safety = distSq / ((dx*dirX + dy*dirY));
                if ((safety > 0) && (safety < minS)) { minS = safety; }
            } else if ((seenCharges.get(i) instanceof LED)) {
                targetCharge = ((LED) seenCharges.get(i)).getCharge();

                if (targetCharge != 0.0D) {
                    distSq = sens.getDistanceTo((LED) seenCharges.get(i));
                    distSq = distSq * distSq;

                    // calculated forces
                    dx = targetCharge * (((LED) seenCharges.get(i)).getTargetX() - this.getX()) / distSq;
                    dy = targetCharge * (((LED) seenCharges.get(i)).getTargetY() - this.getY()) / distSq;

                    // add calculated forces to overall direction so far
                    dirX += dx;
                    dirY += dy;

                    safety = distSq / ((dx*dirX + dy*dirY));
                    if ((safety > 0) && (safety < minS)) { minS = safety; }

                    if (minS < 5) {
                        targetCharge *= minS/5;
                    }

                    if (minS > 50) {
                        targetCharge *= minS/50;
                    }
                }
            } else {
                System.out.println("ERROR: unknown seenCharges item "+i);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // calculate vector normal, apply to dirX, dirY
    norm = Math.sqrt(dirX*dirX + dirY*dirY);
    dirX = dirX / norm;
    dirY = dirY / norm;

    // set desired direction if it calculates a number
    if (dirX == (double) dirX && dirY == (double) dirY) {
        this.setDesiredDirection(sens.getDirectionTo(dirX + sens.getXPos(), dirY + sens.getYPos()));
    }
}

```

Figure 9. Calculating the resultant push/pull force from the VFF algorithm.

Bibliography

- [1] Parker, L. (2003). *Current research in multirobot systems*. Artif Life Robotics, 7(1-2), pp.1-5.
- [2] Penders, J. (2007). *Robot Swarming Applications*. Materials and Engineering Research Institute, 1-8.
- [3] AutoStore. (2016). *AutoStore / Automated Warehouse Robots*. [online] Available at: <http://autostoresystem.com/> [Accessed 9 Feb. 2016].
- [4] Şahin, E. and Winfield, A. (2008). *Special issue on swarm robotics*. Swarm Intell, 2(2-4), pp.69-72.
- [5] Tan, Y. and Zheng, Z. (2013). *Research Advance in Swarm Robotics*. Defence Technology, 9(1), pp.18-39.
- [6] Augugliaro, F., Lupashin, S., Hamer, M., Male, C., Hehn, M., Mueller, M., Willmann, J., Gramazio, F., Kohler, M. and D'Andrea, R. (2014). *The Flight Assembled Architecture installation: Cooperative construction with flying machines*. IEEE Control Systems, 34(4), pp.46-64.
- [7] *NASA's tumbleweed-inspired rovers for large-scale planetary exploration*. (2008). Industrial Robot, 35(2).
- [8] M. Gnatowski, (2006). *Search-and-rescue using team of robots*. Industrial Robot: An International Journal, 9(28).
- [9] Penders, J. (2007). *Robot Swarming Applications*. Materials and Engineering Research Institute, 1-8.
- [10] O'Grady, R., Groß, R., Christensen, A. and Dorigo, M. (2010). *Self-assembly strategies in a group of autonomous mobile robots*. Autonomous Robots, 28(4), pp.439-455.

- [11] D.P. Barnes, P. Summers, A. Shaw, *An investigation into aerobot technologies for planetary exploration*, 6th ESA Workshop on Advanced Space Technologies for Robotics and Automation, ASTRA 2000, December 2000, pp. 3.6–5
- [12] Lis.epfl.ch. (2016). *Laboratory of Intelligent Systems / EPFL*. [online] Available at: <http://lis.epfl.ch/> [Accessed 03 Feb. 2016].
- [13] Bulletphysics.org. (2016). *Real-Time Physics Simulation*. [online] Available at: <http://bulletphysics.org/wordpress/> [Accessed 30 Jan. 2016].
- [14] Tuci, E. and Rabérin, A. (2015). *On the design of generalist strategies for swarms of simulated robots engaged in a task-allocation scenario*. *Swarm Intell*, 9(4), pp.267-290.
- [15] Persson, M (aka. Notch). (2016). *Ludum Dare 22 / Ludum Dare*. [online] Available at: <http://ludumdare.com/compo/ludum-dare-22/?action=preview&uid=398> [Accessed 13 Feb. 2016].
- [16] Mann, S., Mann, S., Nanavati, V. and Silverton, J. (2016). *Mean direction through 360 degrees?* [online] PC Review. Available at: <http://www.pcreview.co.uk/threads/mean-direction-through-360-degrees.1746638/> [Accessed 20 Feb. 2016].
- [17] GameFromScratch.com. (2012). *GameDev math recipes: Rotating to face a point*. [online] Gamefromscratch.com. Available at: <http://www.gamefromscratch.com/post/2012/11/18/GameDev-math-recipes-Rotating-to-face-a-point.aspx> [Accessed 28 Feb. 2016].
- [18] Phys.org. (2016). *Airborne robot swarms are making complex moves (w/ video)*. [online] Available at: <http://phys.org/news/2012-02-airborne-robot-swarms-complex-video.html> [Accessed 3 Mar. 2016].
- [19] Borenstein, J. and Koren, Y. (1989). *Real-time obstacle avoidance for fast mobile robots*. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5), pp.1179-1187.
- [20] O'Grady, R., Christensen, A. and Dorigo, M. (2009). SWARMORPH: Multirobot Morphogenesis Using Directional Self-Assembly. *IEEE Trans. Robot.*, 25(3), pp.738-743.

- [21] Mulgaonkar, Y., Cross, G., Kumar, V. (2015). *Design of Small, Safe, and Robust Quadrotor Swarms*. IEEE International Conference on Robotics and Automation (ICRA), pp. 2208-2215.
- [22] Cs.mcgill.ca. (2016). *Local Path Planning Using Potential Field*. [online] Available at: <http://www.cs.mcgill.ca/~hsafad/robotics/> [Accessed 28 Feb. 2016].
- [23] G. R. S. Bhattacharya and V. Kumar. (2015). *Persistent Homology for Path Planning in Uncertain Environments*, IEEE Transactions on Robotics (T-RO), vol. 31, iss. 3, pp. 578-590.
- [24] Brambilla, M., Ferrante, E., Birattari, M. and Dorigo, M. (2013). *Swarm robotics: a review from the swarm engineering perspective*. Swarm Intell, 7(1), pp.1-41.
- [25] Bonabeau, E., Dorigo, M. and Theraulaz, G. (1999). *Swarm intelligence: From Natural to Artificial Systems*. New York: Oxford University Press.
- [26] Dudek, Jenkin, Milios. and Wilkes. (1993). *A taxonomy for Swarm Robots*. Intelligent Robots and Systems '93, IROS '93.Proceedings of the 1993 IEEE/RSJ International Conference, 1, pp.441 - 447
- [27] Dorigo, M., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Forster, A., Gonzales, J., Guzzi, J., Longchamp, V., Magnenat, S., Mathews, N., Montes de Oca, M., O'Grady, R., Pinciroli, C., Pini, G., Retornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stutzle, T., Trianni, V., Tuci, E., Turgut, A. and Vaussard, F. (2013). *Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms*. IEEE Robotics & Automation Magazine, 20(4), pp.60-71.
- [28] Koren, Y., and Borenstein, J. (1991). *Potential Field Methods and their Inherent Limitations for Mobile Robot Navigation*. IEEE Conference on Robotics and Automation, 2, pp.1398-1401.

UNUSED

Main Chars, Main tasks, experimental results, future promising applications + problems to overcome

Navarro, I. and Matía, F. (2013). *An Introduction to Swarm Robotics*. ISRN Robotics, 2013, pp.1-10.

Local directional instructions from bots attached to the roof

Ducatelle, F., Di Caro, G., Pinciroli, C. and Gambardella, L. (2011). *Self-organized cooperation between robotic swarms*. *Swarm Intell*, 5(2), pp.73-96.

Metamorphism in a swarm robotics system

Christensen, A., O'Grady, R. and Dorigo, M. (2007). *Morphology Control in a Multirobot System*. *IEEE Robotics & Automation Magazine*, 14(99), pp.x5-x5.

External Forces (mentioning, mostly on joints)

Phong, L., Choi, J., Lee, W. and Kang, S. (2015). *A novel method for estimating external force: Simulation study with a 4-DOF robot manipulator*. *International Journal of Precision Engineering and Manufacturing*, 16(4), pp.755-766.

Positioning system using ultra sonic sensors

Figuerola, F. (1986). *An ultrasonic ranging system for robot position sensing*. *The Journal of the Acoustical Society of America*, 80(S1), p.S100.