Modelling Google Loon

Peter Gelsbo, s123397

Kongens Lyngby 2015

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

The goal of this project is to assess the feasibility of Google's Project Loon. With Project Loon, Google wants to provide remote areas of the globe with an internet connection using balloons. In this project, an algorithm to control the balloons' movements will be developed as well as a simulation tool to visualize the balloons' movements.

Summary (Danish)

Målet for dette projekt er at vurdere gennemførligheden af Googles Project Loon. Med Project Loon vil Google levere internetforbindelse til fjerne og øde områder af verden ved hjælp af balloner. I dette projekt vil der blive udviklet en algoritme til at kontrollere ballonernes bevægelser og et simulationsværktøj til at visualisere deres adfærd.

Preface

The project was carried out at DTU Compute.

The project deals with a feasibility assessment of Google's Project Loon. The products of this project is a simulation tool and a control algorithm.

This report documents the development process and the considerations done throughout the project.

Lyngby, 23-June-2015

Peter Gelsbo, s123397

Contents

Summary (English) i

Summary (Danish) iii

Preface v

1 Introduction 1
- 1.1 The Google Loon project . . . 2
- 1.2 Modelling Google Loon . . . 3
- 1.3 Project plan . . . 3
  - 1.3.1 Phases of the project . . . 3
  - 1.3.2 Initial project plan . . . 5

2 Problem analysis 7
- 2.1 Problem definition . . . 8
- 2.2 Requirements Specification . . . 9
  - 2.2.1 Functional requirements . . . 9
  - 2.2.2 Non-functional requirements . . . 10

3 Design 11
- 3.1 Models . . . 12
  - 3.1.1 Modelling the world . . . 12
    - 3.1.1.1 Geography and topography . . . 12
    - 3.1.1.2 Wind layers . . . 13
    - 3.1.1.3 Time . . . 14
  - 3.1.2 Modelling balloons . . . 15
    - 3.1.2.1 The physical balloon . . . 15
    - 3.1.2.2 Movement in the x/y-coordinate system . . . 16
    - 3.1.2.3 Changing altitude . . . 16

3.1.3 Summary of assumptions and conclusions 17
3.2 Algorithm design 17
3.2.1 Design Considerations 17
3.2.1.1 Centrally controlled algorithm 18
3.2.1.2 Distributed algorithm 18
3.2.1.3 Grouping balloons 19
3.2.2 Control Algorithm 20
3.2.2.1 Pseudoalgorithm 21
3.2.2.2 Situation 1 - Find neighbours 22
3.2.2.3 Situation 2 - Get closer 23
3.2.2.4 Situation 3 - Excessive overlap 31
3.2.2.5 Illustrations and examples 32
3.2.3 Summary of assumptions and conclusions 37
3.3 Designing the simulator 37
3.3.1 Visualizing the world and balloons 38
3.3.2 Functionality for testing 39

4 Implementation **41**
4.1 Implementation process 42
4.2 Implementing the model and simulation tool 42
4.2.1 Programming language 42
4.2.2 Design pattern 42
4.2.3 Model 43
4.2.3.1 World 44
4.2.3.2 WindLayer 44
4.2.3.3 Balloon 44
4.2.3.4 Direction 45
4.2.4 View 45
4.2.4.1 Window 46
4.2.5 Predefined configuration 48
4.3 Implementing the algorithm 49
4.3.1 Main Algorithm 49
4.3.1.1 Pseudoalgorithm 50
4.3.2 Secondary Algorithms 51
4.3.2.1 FindNeighbours 51
4.3.2.2 FindOverlapping 52
4.3.2.3 CalculateCenter 53
4.3.2.4 CalculateDirection 54
4.3.2.5 ChangeDirection 54
4.3.2.6 NextMove 55
4.3.2.7 CalculateRelativeDistance 55
4.3.2.8 CheckOverlap 56
4.3.3 Border conditions 57

5 Tests and calculations **59**
5.1 Tests and calculations 60
5.2 Testing the simulation tool 60
5.2.1 Drawing and movement 60
5.3 Testing the control algorithm 62
5.3.1 Detailed tests 62
5.3.1.1 Testing procedures 62
5.3.1.2 Test results 64
5.3.2 Long term / large scale tests 65
5.3.2.1 Efficiency - Number of direction changes 65
5.3.2.2 Stability - Balloons grouping, not splitting . . . 65
5.3.2.3 Scalability - A lot of balloons 65
5.3.2.4 Stability - Average distance between neighbours 66
5.3.3 Changing parameters 70

6 Discussion **73**
6.1 Discussion 74
6.1.1 Simulation tool 74
6.1.2 Control algorithm 74
6.1.3 Future improvements 75
6.2 Evaluating the project 76

7 Conclusion **79**
7.1 Conclusion 80

A Project Plan **81**

B User guide **83**

Bibliography **87**

Chapter 1

Introduction

1.1 The Google Loon project

Project Loon is a research project by Google which aims to provide internet access to remote parts of the world where cables and mobile networks are not existing, using a network of high altitude balloons.

The project started in 2011 as an unoficial project under Google X, a semi-secret facility which is behind many of Google's futuristic projects [GOO]. The project covers everything from creating the right fabric for the balloons to optimizing hardware, antennaes and devloping software systems to control the fleet of balloons.

Google's own description of the project is as follows.

> "Many of us think of the Internet as a global community. But two-thirds of the world's population does not yet have Internet access. Project Loon is a network of balloons traveling on the edge of space, designed to connect people in rural and remote areas, help fill coverage gaps, and bring people back online after disasters." ([LOOb])

The idea is to create a network of a large number of high altitude balloons flying around the globe. The balloons will follow steady wind layers in the stratosphere and will be able to change speed and direction by adjusting their altitude to follow a different wind layer. Google uses wind data from the american National Oceanic and Atmospheric Administration (NOAA), but plans to develop their own predictions based on wind data from the Project Loon balloons.

In a series of YouTube videos [LOOa], the Project Loon team explains how the balloons work and the technical solutions to the problems they have encountered during their work so far.

The balloons electronic equipment runs on solar power using a battery pack to operate at night. Google estimates that the balloons can stay in the stratosphere between 100 and 200 days.

An alternative solution to providing global internet coverage could be using existing technology such as sattelites. However, while this solution is well proven, launching satellites into orbit is much more expensive than launching helium filled balloons. Thus, with Project Loon, Google is trying to develop a cheaper alternative to satellites. They do, however, have competitors such as SpaceX [SPA] and Facebook [INT] who are trying to achieve the same goal of global internet connectivity, using satellites and drones respectively.

1.2 Modelling Google Loon

In this project, the objective is to assess the feasibility of Project Loon. The focus will be on the basic concept of flying balloons around the globe using wind-layers in the stratosphere and providing a good, stable coverage on the ground. Hence, considerations about design of the balloon's fabric, hardware, electronics and communication systems will largely be disregarded and any necessary information from these parts of the project will be based on assumptions.

The products of the project will be an algorithm for controlling the balloons and a simulation tool to visualize the balloons movements around the world. The algorithm will be based on a model of the world, wind layers and the balloonns themselves. This model and the simulation tool will allow different parameters to be changed to test the assumptions made during this project.

1.3 Project plan

The project is divided into four phases as a typical software project: Analysis, design, implementation and tests. Normally, a deployment phase would be included as well. However, as this project does not have an actual customer, the deployment phase is excluded.

Throughout the project, progress, assumptions and decisions will be documented and described in this report.

The initial project plan is found in Figure 1.2. A more detailed project plan can be found in Appendix A which covers the individual tasks, when they were expected to be finished and when they were actually finished.

1.3.1 Phases of the project

As in most software projects the development follows an iterative process where the four phases are repeated several times in order to gradually build the system and address new problems that are discovered during process. In Figure 1.1 the four phases are illustrated.

Initial planning
A project plan setting the time frame for the project is defined.

Analysis
Initially, the problem analyses defines the project by answering the following question: What is the goal of the project? Which products will be produced and what are the requirements for the products?

During the iterative process, new problems might appear which will need additional analysis to solve.

Design
Design is about assessing the problems and defining the solutions to those. For the simulation tool, the design deals with practical issues such as choosing a platform, programming language and design pattern, as well as the apperance of the program when running. For the algorithm, the design deals with the general algorithmic concepts. Here, the behaviour of the balloons will be described using words, equations and illustrations.

Implementation
The implmentation converts the design into pseudocode and source code. Pseudocode is simplified source code which is used to structure the work and identify any problems before implementing the real source code.

Testing
Testing includes both detailed testing of the individual components of the simulator and control algorithm, as well as long term and large scale tests of the algorithm.

Deployment
This project does not have a customer and is not expected to deliver a finished product. Hence, this project will never enter the final deployment phase.

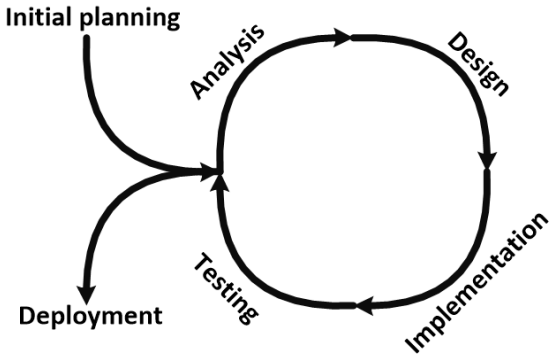


Figure 1.1: Illustration of the iterative process

1.3.2 Initial project plan

The initial project plan in Figure 1.2 shows the expected time consumption for each phase of the project, including start and end estimations.

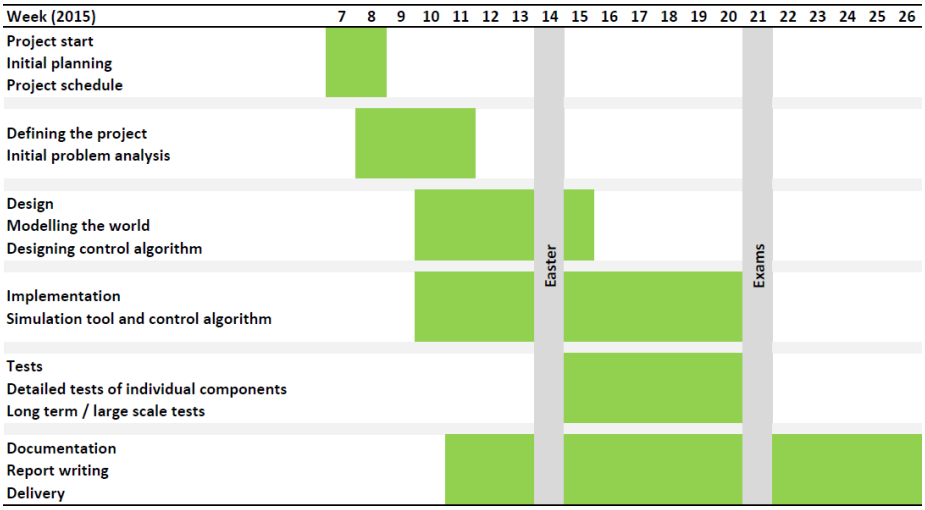


Figure 1.2: Initial project plan

Chapter 2

Problem analysis

2.1 Problem definition

To assess the feasability of Project Loon, this project will focus on creating an algorithm to control the balloons to make them fly together while providing a good, stable coverage on the ground. The algorithm should be be able to scale up from handling a few balloons to thousands of balloons. A simulation tool will be developed to visualize the algorithm running in different conditions and for various lengths of time.

Specifically, the following questions will be answered during this project.

- What is good, stable and continuous coverage and how can the balloons achieve this?
- What can be done to minimize human interaction?

2.2 Requirements Specification

This specification frames the project and defines which tasks needs to be done and states the expectations to the products. The requirements are divided in two categories: Functional and non-functional requirements.

2.2.1 Functional requirements

Describes the expected outcome of the project, the functionality and *look and feel* of the system.

General

1. Explore and assess the problems which emerges when looking into the concept of providing internet coverage using balloons.

Algorithm

1. Effective algorithm which controls the balloons so that they provide continous and stable coverage on the ground.
2. Based on parameters which can be changed dynamically to test different situations.

Simulation Tool

3. Provide a graphical view of the world and the balloons that move around.
4. Graphical indication of a balloon's coverage, range, direction and wind layer.
5. The possibility of manipulating the world by adjusting parameters through an interactive user interface such as buttons on the screen or keyboard commands.
6. Handle a variable number of balloons.
7. Allow the user to adjust simulation speed to see detailed behaviour as well as long term effects.

2.2.2 Non-functional requirements

Describes the constraints of the system that needs to be met.

Algorithm

1. The algorithm must be developed in a way that allows it to be easily modified and extended to examine future problems.

Simulation Tool

2. Developed in a suitable programming language.
3. Can run on a common operating system such as Windows, Linux or OSX.
4. Cross platform compatibility is not required.

Chapter 3

Design

3.1 Models

Project Loon is a big and complex project involving a lot of technical knowledge from different areas of expertise. This section will narrow down the scope of the project to focus on the subjects that are important when developing an algorithm to control the balloons.

Modelling the real world in every little detail is close to impossible. Therefore, a number of assumptions must be made to simplify the world. These assumptions will sort out small or irrelevant details while keeping the modelled world realistic enough to be able to make useful conclusions.

In the following sections the considerations made in regards to the modelling of the world and balloons are described.

3.1.1 Modelling the world

The system needs a simplified model of the world in which the balloons can move around. Initially, the modelled world needs to be as simple as possible. This is done to keep focus on the basic concepts of the control algorithm. Later the model can be extended to test the algorithm in different scenarios.

3.1.1.1 Geography and topography

The shape of the world is essential to the behaviour of the balloons. Here we consider the geography and topography of the modelled world.

The geography and topography of the world describes the physical layout of the world. Should the world be spherical like the earth and how are seas, oceans, contients and other landmasses distributed?

A spherical world would be the obvious choice, as this would approximate the real world quite well. However, a spherical world is hard to visualise graphically without using advancd 3D tools, which goes beyond the scope of this project. Thus, a square world similar to the representaion of the world on a flat map is chosen to simplify the graphical simulation. The square world introduces some programatical issues when the balloons crosses the edge of the world. These issues are discussed in section 4.3.3.

A world with no topographical features is chosen: The world is flat, with no oceans, landmasses or mountains. A more detailed topograpgy could be interesting to introduce at a later stage, to see the impact of the landscapes could have on a balloon's behaviour. However, this strong simplification greatly simplifies the world and allows us to keep focus on the behaviour of the balloons and how control them relatively to each other.

3.1.1.2 Wind layers

The wind is what drives the balloons to fly around the world and hence a model of the wind should be made.

The balloons move in relatively stable wind layers which varies in speed and direction at different altitudes. It is these wind layers which allows the balloons to change direction and move towards each other.

It is possible to obtain real world wind data from the american NOAA[1] or similar instituions around the world, which can be processed to fit into the simplified world. Such a dataset is visualized on figure 3.1, showing the average wind speed and direction at approximately 10 km altitude during 2014.

As with the topography of the world, it would be interesting to see how the balloons would behave in a more detailed world where the winds are based on data that approximates the reald world to a greater extend. However, when developing and testing the algorithm, real world data is not beneficial and some simpler model needs to be introduced to ensure that the algorithm is working as expected.

A simplified model of the wind is chosen, where the world have four uniform wind layers moving north, south, east and west. It is possible to change these directions which makes it possible to test the algorithm in different but still simplified wind conditions. An example of this could be a situation where the wind moves east, north east and south east.

[1] National Oceanic and Atmospheric Administration

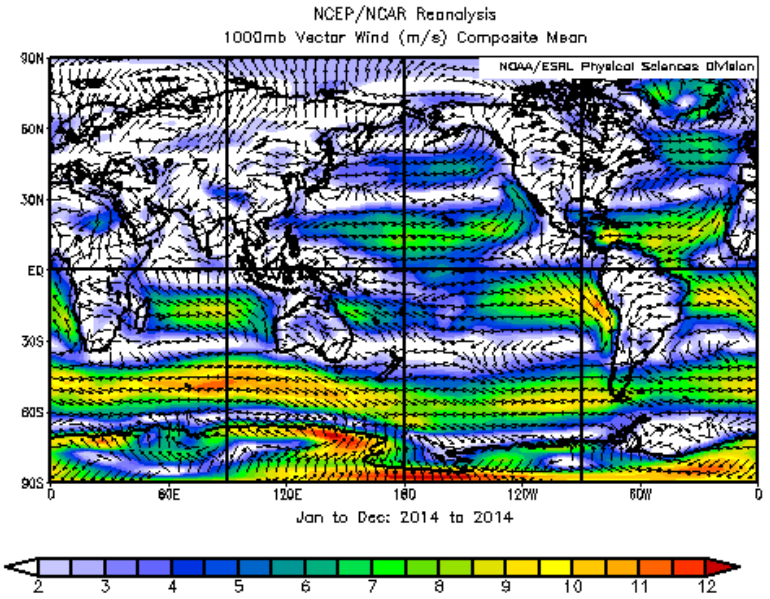


Figure 3.1: Average wind speed and direction in 2014 at 1000mb, approx. 10 km.

Source: NOAA [ESR]

3.1.1.3 Time

To simulate the the balloons' movement over time, the modelled world should support some sort of indication of time. Two options are considered here: "Real world" time and a simple time system.

Real world time
Representing time as seconds, hours and days is the first option. This is how time is usually represented. While this system is easy for humans to understand it adds unnecessary complexity to the model.

Simple time intervals
The second option is a simpler time system which is chosen to simplify the model. A timer ticks, and the balloons move to a new position on each tick, based on their speed and direction.

3.1.2 Modelling balloons

The system needs a simplified representation of the balloons. Here the focus will be on the physical balloon where the properties of the balloon is described as well as how the balloon moves.

3.1.2.1 The physical balloon

The Google Loon balloons are equipped with a lot of electronics to enable them to communicate with other balloons and provide coverage on the ground. Since the purpose of this project is to develop an algorithm to control the balloons and a graphical simulator to visualise the movements, we can disregard the physical design of the balloon and focus on its basic properties. These properties are arbitrary values that can be adjusted as needed.

The following information is avaliable to the balloon's neighbours that are within communication range.

Position
The current position of the balloon in an x/y-coordinate system.

Altitude
The altitude of the balloon. This determines which wind layer it is located in. As described in section 3.1.1.2, only four wind layers exists initially, each with a constand wind direction and speed.

Direction
The direction of the balloon, based on the wind layer in which it is located. This can be any direction between 0 and 360 degrees.

Speed
The Speed of the ballon, based on the wind layer in which it is located.

Coverage
Radius of the covered area on the ground.
This is a parameter which can be changed to test different conditions.

Range
: The communication range of the balloon. Neighbouring balloons within the communication range can *see* the balloon. However, two-way communication needs to be established to make a connection between two balloons. This is described in detail in section 3.2.2.
This is a parameter which can be changed to test different conditions.

3.1.2.2 Movement in the x/y-coordinate system

The balloon moves with the wind based on the direction and speed at its location in the wind layer in which it is currently moving in.

Disregarding the restrictions on wind direction the following example illustrates the calculation.

Direction: 22.5 deg (which translates to north northeast)
Speed: 3

$x = sin(22.5\,\text{deg}) * 3 = 2.77$
$y = cos(22.5\,\text{deg}) * 3 = 1.15$

The balloon will then move 2.77 in the x-direction and 1.15 in the y-direction. This movement is instant.

3.1.2.3 Changing altitude

In practice, changing the altitude of the balloons is a complex maneuver. However, the chosen altitude system of the modelled world is a strong simplification. A number of wind layers are avaliable to the balloon, and the balloon can change between those without considering the resource consumption this might require in the real world. Initially, changing altitude will be an instant process but this could be changed to be a gradual transition in a later stage of development.

3.1.3 Summary of assumptions and conclusions

The following assumptions and conclusions are made during the design of the model.

Square world
The modelled world is square and not spherical. The size of the world is arbitrary and can be adjusted as needed.

Flat world
The modelled world has no terrain. No land masses, seas, mountains or other topographical features.

Wind layers
The wind layers are simplified to four layers: North, south east and west. The four wind layers cover the entire world.

Time
The time ticks at intervals that allows for different simulation speeds.

Balloons
The hardware of the balloons is not modelled. Only the position, coverage and communication range is modelled. This information is based on arbitrary values that can be adjusted as needed.

Movement of balloons
Balloons moving across the edge of the world will appear instantly at the other side of the world. The balloons move at 100% the speed of the wind.

3.2 Algorithm design

In this section, we will discuss how to tackle the problem of controlling the balloons. Ideally, the balloons should be able to coordinate their movements without too much human interaction and do so in a manner that provides a stable coverage on the ground while minimizing their resource consumption.

3.2.1 Design Considerations

In order to provide the best coverage on the ground, the balloons must be able to communicate and coordinate their movements, either with each other or a central controller.

3.2.1.1 Centrally controlled algorithm

A centrally controlled system, needs an algorithm that runs on a system on the ground and sends commands to the balloons from a control center. Furthermore, the balloons will need to be able to communicate with each other to relay the internet connection, but will not be able to make decisions on their own.

Practically, this could be done with a sattelite connection to each balloon to send movement commands from the central system, as illustrated in Figure 3.2.

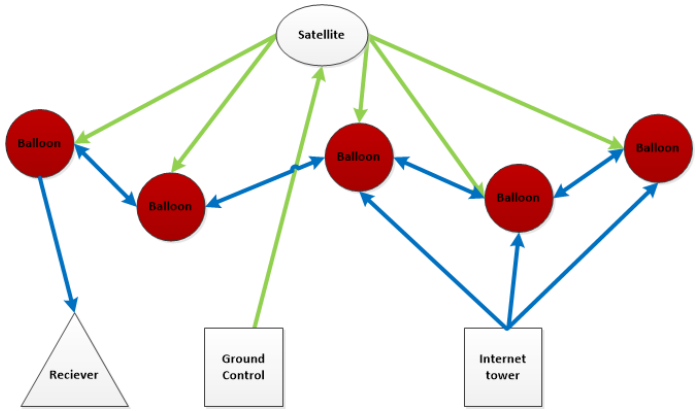


Figure 3.2: Illustration of a centrally controlled system.

3.2.1.2 Distributed algorithm

With a distributed algorithm, the balloons communicate with each other in order to decide their movements. The algorithm runs independently on each balloon and needs no central system. This is illustrated in Figure 3.3.

One disadvantage with this solution is that the balloons will need to be within communication range of each other. If they are moving around alone, they will not be able to decide how to get closer to each other.

In practice, this solution could be supplemented with a sattelite link to each balloon, which would allow manual control of the balloons in case human interaction is needed to solve a problem.

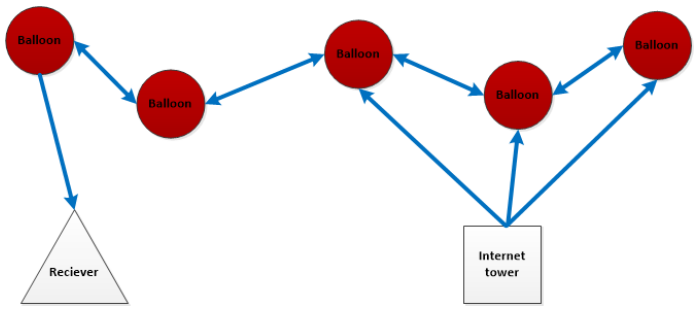


Figure 3.3: Illustration of a system based on a distributed algorithm.

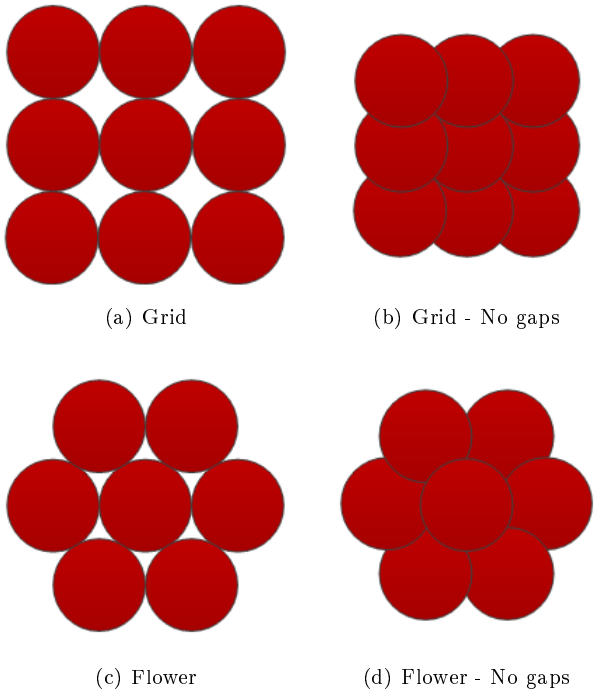
3.2.1.3 Grouping balloons

To provide consistent coverage, the balloons need to group together. Here, two possible formations are considered. A grid (Figure 3.4(a)) where the balloons align next to each other and a flower pattern (Figure 3.4(c)) where the balloons move as close to each other as possible.

In both patterns, the balloons does not cover the entire area between then but leaves out gaps. These gaps will result in unstable coverage which is not desireable. In Figure 3.4(b) and 3.4(d) these gaps can be eliminated by letting the balloons overlap.

However, this overlap should be as small as possible utilize the potential covered area as good as possible.

The distance between the balloons in grid formation in is $\sqrt{2} * r$ when overlapping, compared to $\sqrt{3} * r$ in the flower formation. The smaller distance the more overlap, thus the best solution is the flower formation.



(a) Grid (b) Grid - No gaps

(c) Flower (d) Flower - No gaps

Figure 3.4: Illustrations of group patterns

3.2.2 Control Algorithm

Ideally, the balloons should require as little as possible manual maintenance and be able to decide their own movements. Thus, the chosen solution is a distributed algorithm where each balloon acts independently, based on its own situation and data from its neighbours.

The main algorithm runs on each balloon and considers the following four situations. The basic concepts of the algorithm is explained in detail in the following sections, while detailed descriptions of the parts that make up the algorithm can be found in section 4.3.

A number of examples are provided in section 3.2.2.5 where the four situations are explained and illustrated. The illustrations are based on the graphical view from the simulator, which is explained in section 4.2.4.1.

1. **No neighbours - Attempt to find nearby neighbours.**
   - Continues without change in wind layer.
2. **Get closer - The balloon has neighbours (two-way communication) but needs to get closer to provide optimal coverage.**
   - Center of neighbours is calculated.
   - Direction towards center is calculated.
   - Wind Layer is changed to approximate direction.
3. **Overlap - The balloon has neighbours but is overlapping too much with some of them (inefficient use of coverage).**
   - Center of overlapping neighbours is calculated.
   - Direction towards center is calculated.
   - Direction is reversed.
   - Wind Layer is changed to approximate direction.

3.2.2.1 Pseudoalgorithm

```
find neighbours
if no neighbours found                          // SIT 1
   move
else
   check overlap
   if no overlap found                          // SIT 2
      find center of neighbours
      calculate direction towards center
      change altitude based on direction
      move
   else                                         // SIT 3
      find center of overlapping neighbours
      calculate direction towards center
      reverse direction
      change altitude based on direction to center
      move
```

Algorithm 1: Pseudoalgorithm describing the main algorithm.

3.2.2.2 Situation 1 - Find neighbours

The balloon will look for neighbours within communication range. If no neighbours are found, the balloon will continue in the current wind layer.

Finding neighbours A neighbouring balloon is a balloon where two-way communication is possible, eg. where the neighbour is within communication range of the given balloon and the given balloon is within communicaition range of the neighbour.

Variable communication range The algorithm should support variable communication range for each balloon, to allow testing different situations. For example the simulation could support balloons running on reduced power because of a dead battery or similar.

Figure 3.5 and 3.6 illustrates balloons in and out of communication range. Figure 3.7 illustrates a situation with two balloons with different communication range, where two-way communication is not possible, because the low-power balloon does not cover the center of the high-power balloon.

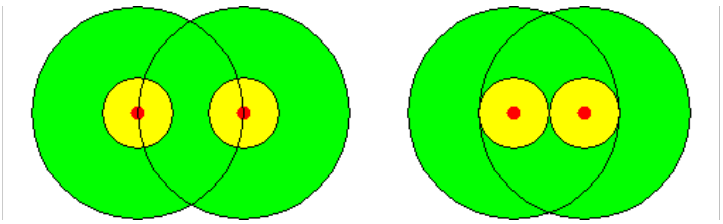


Figure 3.5: In communication range. Two-way communication established.

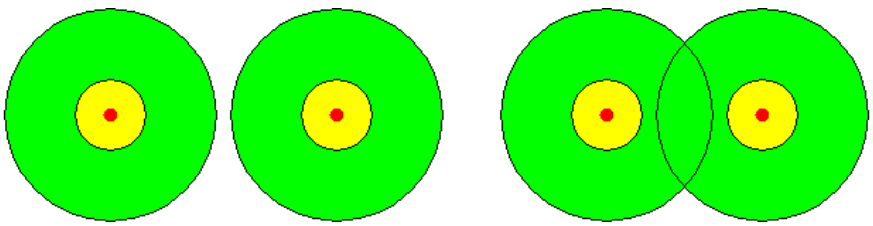


Figure 3.6: Out of communication range. No communication.

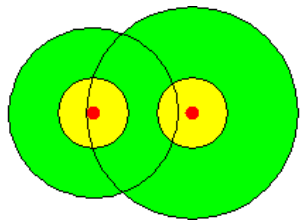


Figure 3.7: Different communication range. Two-way communication not possible.

3.2.2.3 Situation 2 - Get closer

If the balloon have found one or more neighbours, it will try to get closer to those.

Finding center of neighbours This is an essential part of the main algorithm. The center calculation is what makes the balloon move closer or away from each other. The chosen method is relatively simple but can be extended to support more advanced calculations based on speed, direction and perhaps even predicted paths from neighbouring balloons. However, to keep the algorithm as simple as possible, this calculation is only based on the relative distances of the balloon and its neighbours, and does not consider the options mentioned above.

Calculating the average relative distance and moving towards this point makes the balloon move towards each other. Aditionally, a balloon with one neighbour in one direction and two in another direction, will be drawn towards the higher density area, which is the two balloons. This ensures that a small group of balloons moving towards a larger group of balloons, will merge with the larger group and not stick together and continue to move on its own.

The examples below illustrates the different situations that a balloon can be in and explains how the algorithm ensures that the balloons will move correctly. The illustrations are based on the legend in figure 3.8. Red indicates the balloon which the calculation is based on. Blue is the neighbours of the balloon and green marks the calculated center.

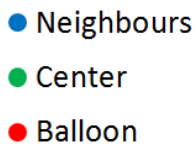
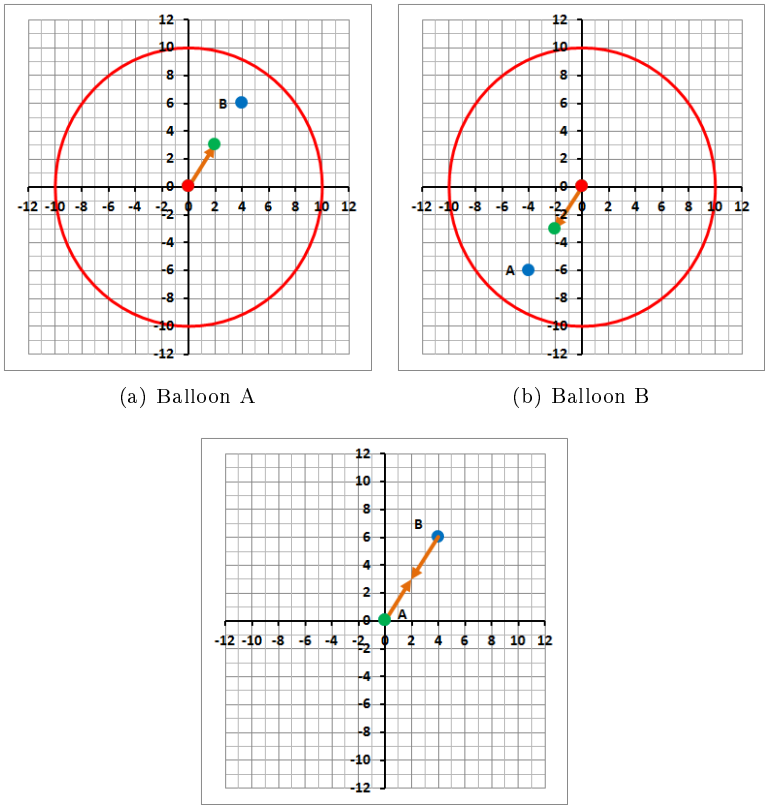


Figure 3.8: Legend for the center calculation illustrations.

The examples below illustrates the center calculation in different situations.

Example 1: Two balloons meeting
In this example, we will look at two balloons: A and B. Figure 3.9 illustrates the center calculation for each balloon. When the calculation has been completed on all balloons, the balloons will move in the directions shown on Figure 3.9(c).

A and B are within range of each other with no other neighbours. Thus, the center calculation for both balloons will result in a center point halfways between them.



(a) Balloon A (b) Balloon B

(c) All directions

Figure 3.9: Example 1
Illustrations of the center calculation on balloon A and B.
Both balloons having the same center point.

Table 3.1: Example 1 - Center calculations for balloon A and B.

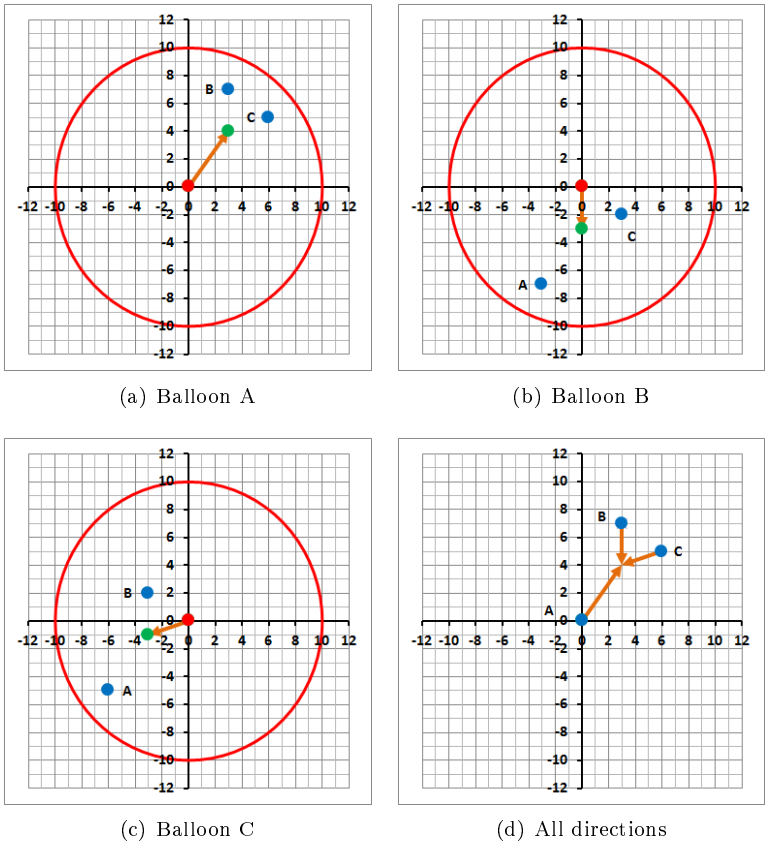
Balloon	Neighbours	Relative distances	Summarized distances	Relative center (avg. distance)
A	B	(4,6)	x = 0+4 = 4 y = 0+6 = 6	(2,3)
B	A	(-4,-6)	x = 0+(-4) = -4 y = 0+(-6) = -6	(-2,-3)

Example 2: Three balloons meeting
In this example, we will look at three balloons: A, B and C. Figure 3.10 illustrates the center calculation for each balloon. When the calculation has been completed on all balloons, the balloons will move in the directions shown on Figure 3.10(d).

The three balloons are all within range of each other and have no other neighbours. Hence, as in example 1, the resulting center point for each balloon should be the same.

Table 3.2: Example 2 - Center calculations for balloon A, B and C.

Balloon	**Neighbours**	**Relative distances**	**Summarized distances**	**Relative center (avg. distance)**
A	B C	(3,7) (6,5)	x = 0+3+6 = 9 y = 0+7+5 = 12	(3,4)
B	A C	(-3,-7) (3,-2)	x = 0+(-3)+3 = 0 y = 0+(-7)+(-2) = -9	(0,-3)
C	A B	(-6,-5) (-3,2)	x = 0+(-6)+(-3) = -9 y = 0+(-5)+2 = -3	(-3,-1)



(a) Balloon A (b) Balloon B

(c) Balloon C (d) All directions

Figure 3.10: Example 2
Illustrations of the center calculation on balloon A, B and C. All three balloons having the same center point.

Example 3: Move towards higher density area
In this example, we will look at four balloons: A, B, C and D. Figure 3.11 illustrates the center calculation for each balloon. When the calculation has been completed on all balloons, the balloons will move in the directions shown on Figure 3.11(e).

Not all balloons can see each other, and thus, the balloons will move towards the higher density area, which is around B and C.

Balloon A Has three neighbours within range (B,C,D). B and C are located north-east while D is located south west. Thus, B and C makes up a higher density area than D which will result in A moving in a north-eastern direction.

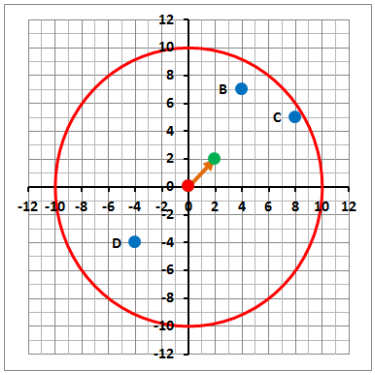
Balloon B Has two neighbours within range (A,C) as D is out of communication range. B will move towards a center point between A,B and C, similar to the situation in example 2.

Balloon C Has two neighbours within range (A,B) as D is out of communication range. C will move towards a center point between A,B and C, similar to the situation in example 2.

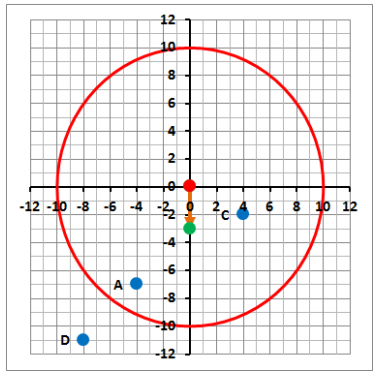
Balloon D Has one neighbour within range (A). D will move in direction of Balloon A, similar to the situation in example 1.

Table 3.3: Example 3 - Center calculations for balloon A, B, C and D.

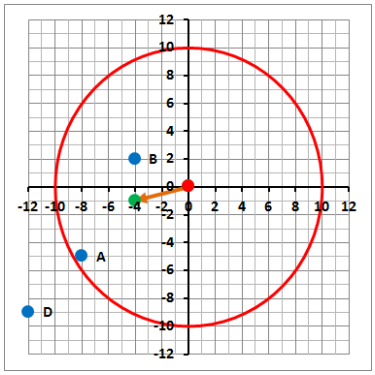
Balloon	Neighbours	Relative distances	Summarized distances	Relative center (avg. distance)
A	B C D	(4,7) (8,5) (-4,-4)	x = 0+4+8+(-4) = 8 y = 0+7+5+(-4) = 8	(2,2)
B	A C	(-4,-7) (4,-2)	x = 0+(-4)+4 = 0 y = 0+(-7)+(-2) = -9	(0,-3)
C	A B	(-8,-5) (-4,2)	x = 0+(-8)+(-4) = -12 y = 0+(-5)+2 = -3	(-4,-1)
D	A	(4,4)	x = 0+4 = 4 y = 0+4 = 4	(2,2)



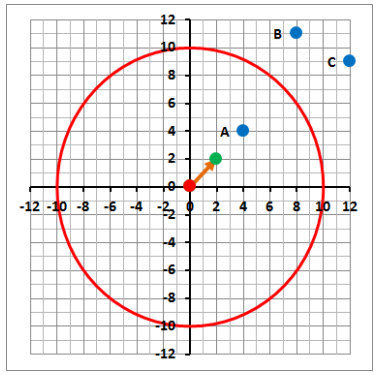
(a) Balloon A



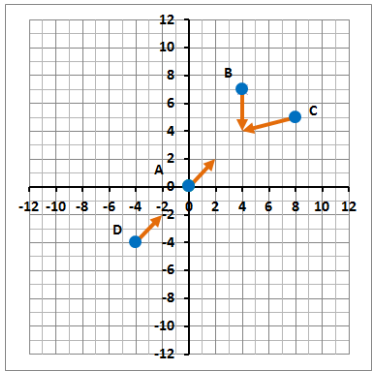
(b) Balloon B



(c) Balloon C



(d) Balloon D



(e) All directions

Figure 3.11: Example 3
Illustrations of the center calculation on balloon A, B, C and D. Balloons moving towards a higher density area.

Calculating direction Both the balloon's location the target point is a pair of x/y coordinates. Calculating the direction towards a point is then a simple trigonometric calculation on the relative distances between the two points.

$direction = atan(\frac{\delta y}{\delta x})$

Figure 3.12 illustrates such a calculation.

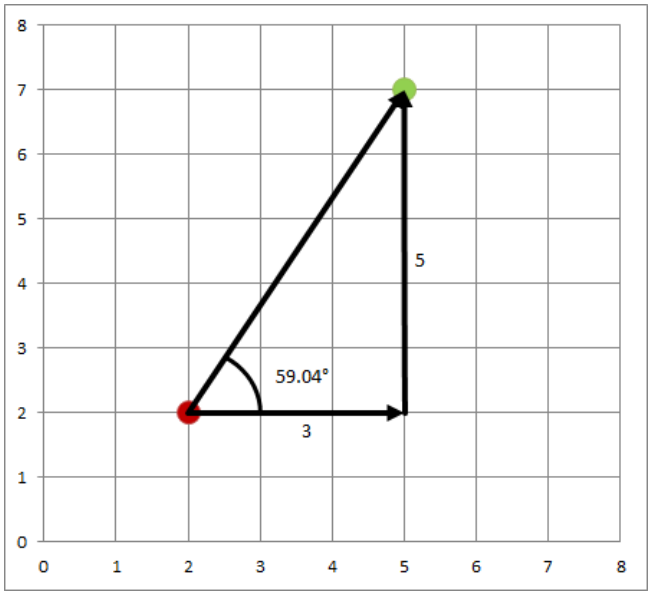


Figure 3.12: Calculating direction

Chainging altitude Once the direction is calculated, the balloon will change attempt to change altitude to a wind layer that matches this direction.

The balloon will iterate through the wind layers and check their directions at its location. The wind layer with the least difference in direction from the preferred direction is chosen. The change in wind layer is instant.

3.2.2.4 Situation 3 - Excessive overlap

If two or more balloons are too close to each other, they will need to spread out in order to maximize the covered area beneath them.

Finding overlapping neighbours To ensure continous coverage on the ground, the balloons needs to be placed with a certain distance between them to avoid holes in the coverage.

The optimal distance between two balloons is $\sqrt{3}*r$, based on the considerations made in the following four cases. The four cases are illustrated on Figure 3.13.

r is the coverage radius on the ground and it is assumed that r is constant for all balloons in any altitude. The distance $\sqrt{3}*r$ can be used to find neighbours which are too close and move away from those.

The chosen solution is option 4 (below), where a parameter p is introduced to allow adjusting the range.

1. **Incomplete coverage**
   - Balloons are placed with distance $2*r$.
   - Results in a hole in the coverage between the balloons.
2. **Overcompensation**
   - Balloons are placed with distance r.
   - Ensures solid coverage on the ground but is inefficient considering the potentially covered area.
3. **Optimal coverage**
   - Balloons are placed with distance $\sqrt{3}*r$.
   - Most efficient use of potential coverage. However, the center between the balloons can experience problems with all three balloons being on the edge of coverage.
4. **Optimal coverage with overlap**
   - Balloons are placed with distance $\sqrt{3}*r*p$.
   - The parameter p can be adjusted to test different situations. If the control algorithm can not provide a gap-free coverage using distance $\sqrt{3}*r$, the distance can be reduced to $\sqrt{3}*r*0.9$ or similar to ensure that any gaps are covered.

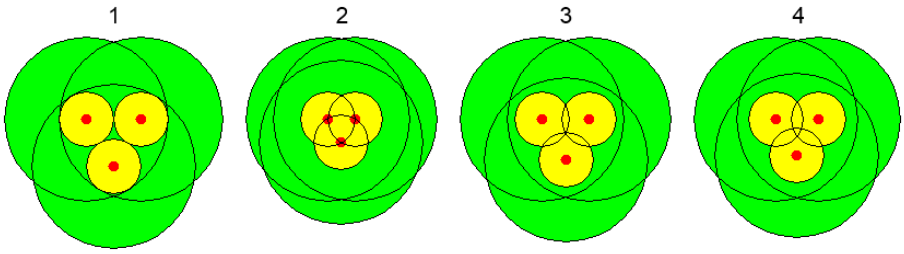


Figure 3.13: Illustrations of the four cases.

Finding center of overlapping neighbours Finding the center of overlapping neighbours follows the same procedure as in situation 2 (section 3.2.2.3), where the center of all neighbours are calculated.

Reversing direction This is a simple matter of calculating the opposite direction. If the direction is west, the reversed direction is east.

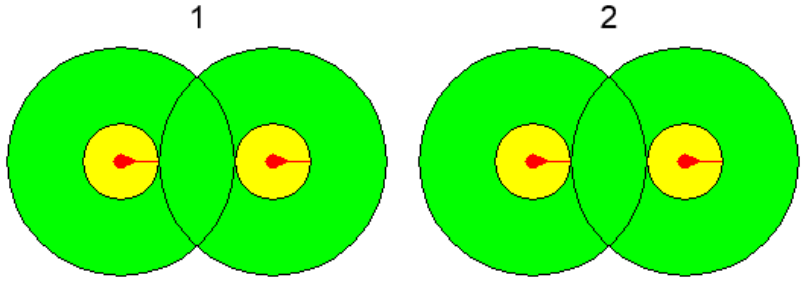
In practice, the direction is a number in the range $0 - 2\pi$. The reverse direction can be found as follows: $reversed = direction + \pi \ modulo \ 2 * \pi$

3.2.2.5 Illustrations and examples

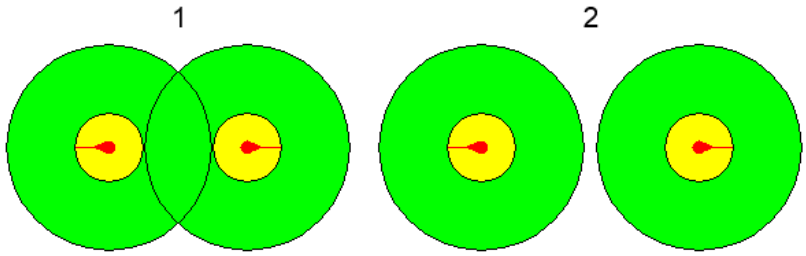
The following examples explains and illustrates the different situations that can occur when the algorithm is running on multiple balloons.

Example 1 - Two balloons moving in parallel
Two balloons moving in parallel will not approach each other. This example is illustrated on Figure 3.14. This is intended behaviour according to the algorithm but is not optimal. This problem could be solved by introducing random movements of the balloons.

1. **No neighbours**
   The balloons can not see each other and will not attept to get closer.
2. **Moving away**
   If the two balloons are moving in the same direction, they will follow each other as in Figure 3.14(a). If the they are moving in opposite directions, they will move away from each other as on Figure 3.14(b).



(a) Parallel, same direction



(b) Parallel, opposite directions

Figure 3.14: Example 1
Two balloons moving in parallel

Example 2 - Two balloons approaching each other
This is the most simple situation where two balloons meet each other. They will behave as follows. This example is illustrated in Figure 3.15.

1. **No neighbours**
   Initially the two balloons will not be able to see each other.
2. **Get closer**
   When the balloons get into communication range with each other, they will begin to communicate. They are not close enough and will continue towards each other.
3. **Excessive overlap**
   The balloons will continue towards each other until one of them detects an excessive overlap. This balloon wil begin to move away from the other.
4. **Stability**
   The other balloon detects no overlap, and continues towards the first balloon, and thus the balloons will move together.

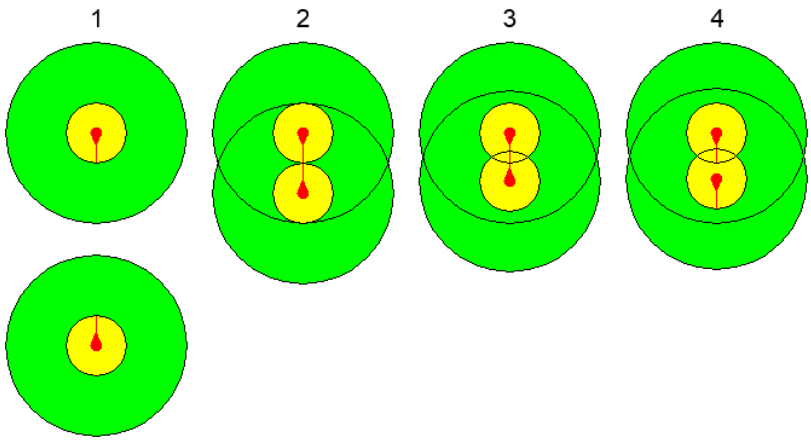


Figure 3.15: Example 2
Two balloons approaching each other

Example 3 - One balloon approaching two connected balloons
This situation is similar to example 2, but includes an additional balloon. The outcome should be that the approaching balloon merges with the two connected balloons forming a group of three.

1. **No neighbours**
   The single balloon approaches the connected balloons but is out of communication range. All three balloons will continue in the direction they are currently moving in.

2. **Getting closer**
   The balloons are getting closer but can not see each other yet.

3. **Communication established**
   Once the balloons are within communication range, they will begin to move towards each other. The two connected balloons will calculate a center point south of them, as the new balloon in the group changes their average position.

4. **Overlap, then stability**
   Once they are within the correct range, they will behave as in example 2(4). One or more of the balloons will detect an excessive overlap and will move away from the rest of the group, the others will follow. In Figure 3.16(4) two of the balloons detects overlap and moves away from the group - both of them in a eastern direction. One of them slightly to the north, one of them slightly to the south. However, as the winds only move north, south, east, west, the result is movement to the east.

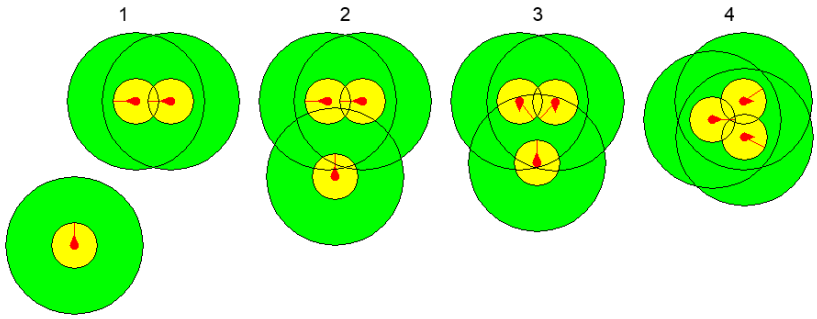


Figure 3.16: Example 3
One balloon approaching two connected balloons

Example 4 - Large groups A group of balloons stay together but will be in constant movement in different directions. The balloons in the center of the group will need to move with the wind. This inevitably causes the balloons to overlap which will make them move back and forth. The outermost balloons in the group will have an average center towards the center of the group, causing the balloons to continously overlap and thus moving back and forth as well.

When a balloon approaches a large group of balloons, it will merge with this group. When it gets within range of the group, the calculated center will be towards the center of the group and it will assume the same situation as the other outermost balloons.

A balloon approaching a larger group of balloons is illustrated in Figure 3.17.

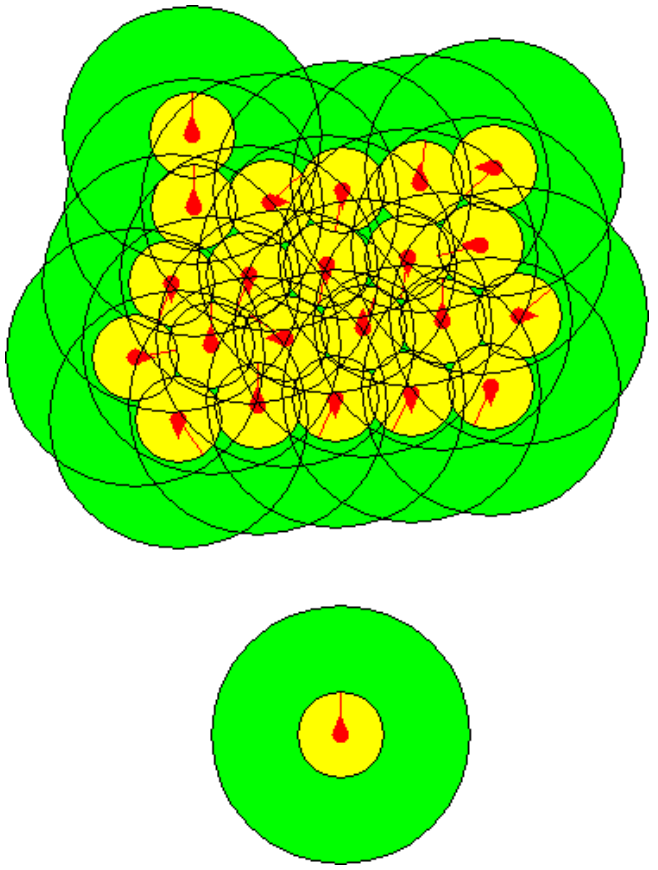


Figure 3.17: One balloon approaching a larg group of balloons

3.2.3 Summary of assumptions and conclusions

The following assumptions and conclusions are made during the design of the control algorithm.

Distributed algorithm
A distributed algorithm is more useful than a centrally controlled algorithm. The distributed algorithm eliminates a layer of complexity where all balloons needs to be in constant contact with a control center.

Pattern
Of the considered patterns, the flower pattern is the most efficient pattern.

Two-way communication
Balloons can not communicate with each other when out of range. Two-way communication needs to be established in order to exchange data.

Main algorithm
The main algorithm is based on the most simple terms. The balloons needs to be in the correct distance of each other. If they are too far away, they need to move closer. If they are too close, they need to move apart.

Center calculation
The center calculation, which is the function that calculates how the ballons can move towards or away from each other, is kept as simple as possible. It does not try to predict the future path of a balloon, even though this might be possible.

3.3 Designing the simulator

The simulation tool should be relatively simple. A lot of simplification and assumptions have been made during the design and moddeling of the world and algorithm. This simplicity will be reflected in this section. For testing purposes, the simulation tool should enable the user to test different preconfigured situations as well as taking manual control of a balloon.

Figure 3.18 sketches the graphical representation of the balloons and a mockup of the window is illustrated in Figure 3.19.

3.3.1 Visualizing the world and balloons

The chosen world is very simple. Hence, the graphical representation of the world does not require many considerations.

The world is a flat, square world of arbitrary size. This will be represented as a blank canvas where balloons can be drawn. The values indicating distance is arbitrary as well, thus it is chosen that one pixel will represent one distance unit.

The balloons should be represented by their position, coverage and communication range. Furthermore, their current direction and preferred direction should be visible as well. The chosen solution is to draw each balloon as a dot with circles around it indicating its coverage and range. Two arrows (or similar) going out of the center of the balloon will indicate its direction and preferred direction.

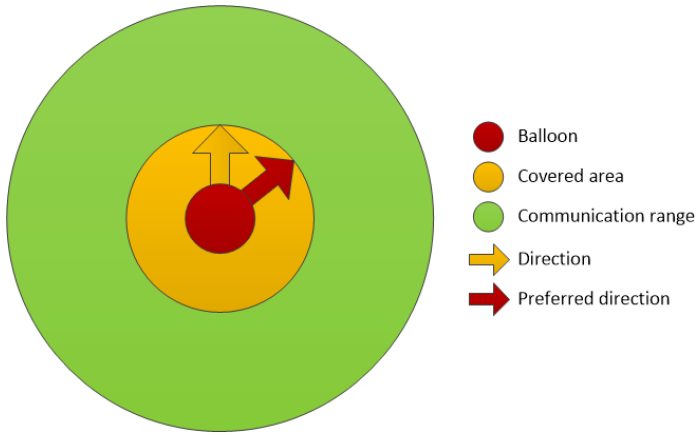


Figure 3.18: Graphical representation of a balloon based on the design.

As the wind layers are strongly simplified, these will not be represented graphically, other than through the directions of the balloons. If a more advanced model of wind layers was introduced, an overay indicating wind speed and direction using arrows could be considered similar to the representation in Figure 3.1.

3.3.2 Functionality for testing

The user should be able to interact with the simulator in order to test different configurations and situations. The chosen solution is to add a row of buttons to the window, to load pre-defined configurations as well as restarting and pausing the simulation.

Preferably, these configurations can be changed by the user using the tool. However, this is not a prioritized option and the configurations can be created in the source code.

Figure 3.19: Mock-up of the simulator window

Chapter 4

Implementation

4.1 Implementation process

The implementation of the simulation tool and the algorithm have been done through an iterative process where the different sub problems have been solved one by one.

4.2 Implementing the model and simulation tool

The simulation tool is a program that runs the control algorithm and represents the balloons and their movements graphically. This will allow the user to simulate the movements of the balloons over time and assess their behaviour.

A user guide is found in Appendix B.

4.2.1 Programming language

Initially, a programming language for the development of the tool must be chosen. It is assessed that the choice of programming language is not critical to the development process as the designed simulation tool does not require advanced drawing functionality but only basic graphics. Thus, Java is chosen as the programming language based on the developers preferations and lack of experience with other languages such as Python. The built in Abstract Window Toolkit (AWT) will be used for the graphical user interface and no external libraries are required.

4.2.2 Design pattern

The development process should comply to the model-view-control design pattern. This ensures structure in the source code and allows for an easy overview of the solution. Keeping the strict structure of a design pattern has not been prioritized during the development process. This has led to a situation where the control-elements have been implemented in the model and view code. The implemented structure is as follows.

- Window - View (GUI) / control (user input, time tick, requesting updates)
- World - Model / control (pre-defined configurations, logging functionality)
- WindLayer - Model
- Balloon - Model / control (control algorithm)
- Direction - Model

4.2.3 Model

The model follows the designed structure: A simple world containing four wind layers and a list of balloons. The number of wind layers is variable but during this project the wind layers are limited to four directions (north, south, east and west).

The diagram in Figure 4.1 is a simplified class diagram. The actual implementation contains several references between the classes used for logging and GUI.

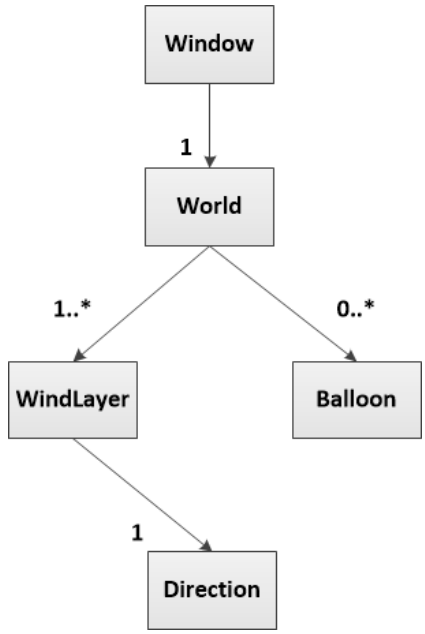


Figure 4.1: Simplified Class Diagram

4.2.3.1 World

The world is defined by a width and height which are arbitrary numbers. It contains a number of wind layers and a number of balloons.

Additionally the world object contains a set of predefined configurations and logging functionality for testing. This is not part of the model but belongs in the control category.

Finally, the model runs the control algorithm on each balloon when an update is requested from the window class. The update process iterates through a list of all balloons in the world and executes the algorithm on each of them.

A solution using threads is the most elegant and resembles the real world best. However, a lot of care needs to be taken when using threads to avoid conflicting access to an objects properties. It is assessed that this is not important and thus the simpler solution is chosen.

4.2.3.2 WindLayer

The WindLayer object describes a wind layer by its altitude, direction and speed. The designed wind layers move in a uniform direction constant speed.

If a more detailed model of the wind layers were required, the wind layer could be divided into a grid of wind fields containing the direction and speed values.

4.2.3.3 Balloon

The balloon object describes the individual ballon using the chosen parameters: Position, range, coverage and altitude.

Furthermore, the balloon object has an update method which is where the control algorithm is run.

Position
A point in the x/y coordinate system.

Range
An integer value indicating the communication range of the balloon.

Coverage
An integer value indicating the radius of the covered area below the balloon.

Altitude
The altitude of the balloon which indicates which wind layer it is located in. Based on he altitude, it is possible to derive the direction and speed.

4.2.3.4 Direction

The direction object is an utility object. It describes a floating point number in the range $0 - 2\pi$ and provides getter and setter methods as well as a reversing method to ensure that this is always respected.

The direction follows the normal x/y coordinate system which is turned 90 degrees compared to that of a compass:

0 = east
$\pi/2$ = north
π = west
$3 * \pi/2$ = south

4.2.4 View

The graphical user interface is relatively simple. Rather than using buttons to interact with the tool, a number of keyboard shortcuts are made. Visually, this keeps focus on the balloons and their movements.

4.2.4.1 Window

The simulator window is a blank canvas on which the balloons will be drawn.

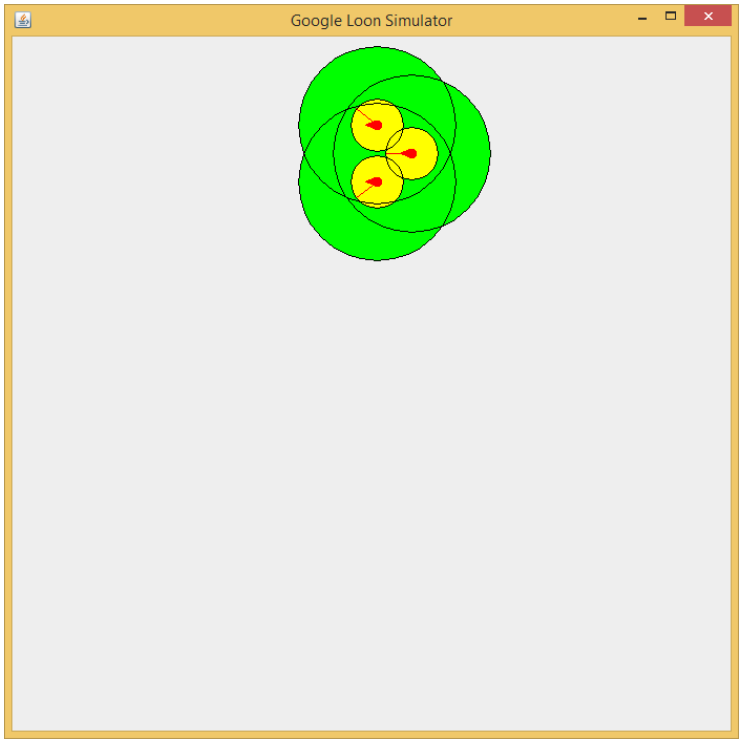


Figure 4.2: Screenshot of the simulator window

Visualizing the balloons
Each balloon is represented as in Figure 4.3.

The balloon is the red dot in the center. The tip of the dot indicates the current direction of the balloon and the red line indicates the preferred direction. The balloon is sorrounded by a green and yellow circle. Green indicates communication range and yellow is the covered area on the ground as described in section 3.1.2.1.

The communication range, covered area and red dot is drawn in this order to ensure that the most important (balloons and covered area) information is on top and visible.

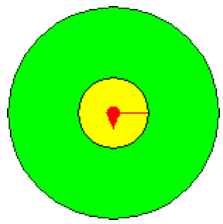


Figure 4.3: Red: The balloon and direction of the balloon
Yellow: The covered area
Green: Communication range.

User input
Instead of adding a row of buttons to the window as designed, the solution to user input is actions through keyboard shortcuts and using the mouse.

The keyboard shortcuts enable the user to load predefined configurations as well as pausing the simulator and increasing/decreasing simulation speed.

Using the mouse, the user can select a balloon for loggin. The log information will be written in the Java console.

Additional information about user input is found in the user guide in Appendix B.

Running the algorithm
In addition to handling the graphical user interface, the window class contains the main java method which initializes the program and runs the timer-thread where the updating of the model is handled.

The updating is done through the process illustrated in Figure 4.4.

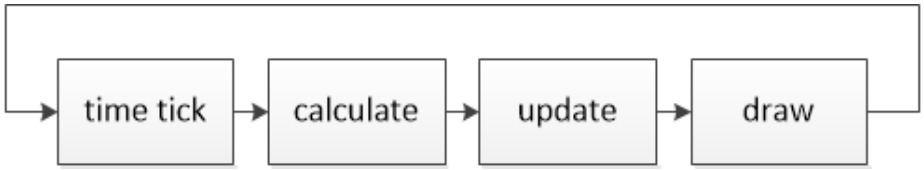


Figure 4.4: Update process

4.2.5 Predefined configuration

Ten predefined configurations are avaliable. Editing these configurations is only possible through source code changes, hence the user is not able to change a configurations using the simulation tool. The configurations can be run through its respective keyboard shortcut. Selecting a configuration resets the simulator and removes all existing balloons.

The avaliable configurations are:

1. 1 balloon flying alone.
2. 2 balloons moving towards each other.
3. 3 balloons moving towards each other
4. 4 balloons moving in parallel
5. 4 balloons starting at the exact same position.
6. 20 balloons starting at the exact same position.
7. 3 balloon starting at approximately the same position
8. 10 balloons in a row.
9. 100 balloons at random positions, moving in random directions, coverage and range reduced.
0. 50 balloons at random positions, moving in random directions.

4.3 Implementing the algorithm

In the following sections, the algorithm controlling the balloons is described in details using pseudoalgorithms. The pseudoalgorithms are simplified versions of the implemented code and does not consider the limitations of a square world where the balloons needs to be able to move seamlessly across the edges. This problem is discussed in section 4.3.3.

The development has been carried out through an iterative process where the different parts of the algorithm have been developed and tested individually to ensure the correct functionality and behaviour of the balloons. The solutions described in the following sections are the final result of this iterative process.

4.3.1 Main Algorithm

The main algorithm is based on the design from Section 3.2.2 where the balloon can be in three states.

- No neighbours
- Get closer to neighbours
- Move away from neighbours

The algorithm calls a number of secondary algorithms. The implementation of those algorithms are explained in section 4.3.2.

The algorithm takes a balloon object as input to update its position, thus returning void.

4.3.1.1 Pseudoalgorithm

```
INPUT     Balloon b
OUTPUT    void

neighbours = FindNeigbours(b)
overlapping = FindOverlapping(neighbours)

if neighbours
    if NOT overlapping
        center = CalculateCenter(b,neighbours)
        direction = CalculateDirection(b.position,center)
        ChangeDirection(direction)
    else
        center = CalculateCenter(b,overlapping)
        direction = CalculateDirection(b.position,center)
        ChangeDirection(direction.reverse)
NextMove(b)
```

Algorithm 2: Pseudoalgorithm describing implementation of the main algorithm.

4.3.2 Secondary Algorithms

The main algorithm uses a number of secondary algorithms for calculations and actions. In this section, these algorithms are explained in detail.

4.3.2.1 FindNeighbours

This algorithm finds the neighbours of a given balloon. In the pseudoalgorithm "AllBalloons" is used as a list of all balloons in the world, which is avaliable to all balloons.

Pseudoalgorithm

```
INPUT     Balloon
OUTPUT    List of balloons

List neighbours = List of balloons

for each balloon in AllBalloons
    if balloon = b OR neighbours contains balloon
        skip
    else
        dist = CalculateDistance(balloon)
        if dist < b.range AND dist < balloon.range
            add balloon to neighbours

return neighbours
```

Algorithm 3: Pseudoalgorithm for finding neighbouring balloons.

4.3.2.2 FindOverlapping

All overlapping balloons are neighbours if the coverage radius is less than the communication range. Hence, to find any overlapping balloons only the neighbours needs to be checked for overlap.

The balloons are overlapping if they are closer than a certain distance to each other. The *CalculateDistance* algorithm is described in section 4.3.2.7 and the *CheckOverlap* algorithm is described in section 4.3.2.8.

Pseudoalgorithm

```
INPUT    Balloon balloon, List of balloons
OUTPUT   List of balloons

List overlapping = List Balloon

for each b in balloons
   if CheckOverlap(balloon,b)
      overlapping.add(b)

return overlapping
```

Algorithm 4: Pseudoalgorithm for finding overlapping balloons.

4.3.2.3 CalculateCenter

This algorithm calculates the center of a group of balloons releative to a balloon in the group.

```
INPUT    Balloon b, List of balloons
OUTPUT   Point

x, y = 0
n = 1    // b is not in the list, but should
         // count for the average.

for each balloon in balloons
   Point p = CalculateRelativeDistance
                        (b.position, balloon.position)
   x += p.x
   y += p.y
   n += 1

x /= n       // Calculate average position
y /= n

center = Point
center.x = b.position.x+x
center.y = b.position.x+y

return center
```

Algorithm 5: Pseudoalgorithm for calculating a center point between a list of balloons.

4.3.2.4 CalculateDirection

This algorithm calculates the direction towards a given point.

```
INPUT     Point to, Point from
OUTPUT    Direction

Point p = CalculateRelativeDistance(to,from)
return atan(p.y/p.x)
```

Algorithm 6: Pseudoalgorithm for calculating the direction from one point to another.

4.3.2.5 ChangeDirection

This algorithm changes the wind layer of the balloon to allow it to move in the direction that is closest to the preferred direction. Here "winds" is a list of wind layers.

```
INPUT     Balloon b, Direction direction
Output    void

OptimalWindlayer = NULL

for each windlayer in winds
    ThisDiff = |b.direction-windlayer.direction|
    OptimalDiff = |b.direction-OptimalWindlayer.direction|

    if ThisDiff < OptimalDiff
        OptimalWindlayer = windlayer

if OptimalWindlayer != NULL
    b.windlayer = OptimalWindlayer
```

Algorithm 7: Pseudoalgorithm changing the direction of a balloon based on a preferred direction.

4.3.2.6 NextMove

This function calculates the next position of the balloon based on direction and speed, and moves the balloon to this position.

```
INPUT     Balloon b
OUTPUT    Point

x = cos(b.direction)*b.speed
y = sin(b.direction)*b.speed

b.position.translate(x,y)

return b.position
```

Algorithm 8: Pseudoalgorithm for calculating the next move of a balloon.

4.3.2.7 CalculateRelativeDistance

This function calculates the relative x/y distances between two points. Two functions are implemented, returning either a point with the x/y values or a double representing the absolute distance between the two points.

```
INPUT     Point, Point
OUTPUT    Point

x = p2.x - p1.x
y = p2.y - p1.y

return Point(x,y)
```

Algorithm 9: 1- Pseudoalgorithm for calculating the relative distance between two balloons.

```
INPUT     Point p1, Point p2
OUTPUT    double

Point p = CalculateRelativeDistance(p1,p2)

return sqrt(p.x^2+p.y^2)
```

Algorithm 10: 2 - Pseudoalgorithm for calculating the relative distance between two balloons.

4.3.2.8 CheckOverlap

This is a function to check whether two balloons are overlapping or not.

```
INPUT     Balloon b1, Balloon b2
OUTPUT    boolean

double DistanceParameter = 1.0

double abs = CalculateRelativeDistance
                    (b1.position, b2.position)

boolean b1OK =
   abs < b1.coverage*sqrt(3)*DistanceParameter

boolean b2OK =
   abs < b2.coverage*sqrt(3)*DistanceParameter;

return b1OK AND b2OK
```

Algorithm 11: Pseudoalgorithm checking overlap between two balloons.

4.3.3 Border conditions

Calculating positions, distances and direction across the borders of the window has provided some challenges during the implementation of the control algorithm. The above pseudoalgorithms does not contain the adjustments needed to tackle this problem, even though this has been one of the major challenges when identifying why the balloons did not behave as expected.

An example of one of these adjustments is in the CalculateRelativeDistance function. Here the algorithm needs to adjust the points properly. The source code snippet below ensures that the correct releative distance is calculated.

```
if(Math.abs(x)>this.world.width/2) {
    x = (this.world.width-Math.abs(x))*Math.signum(-x);
}
if(Math.abs(y)>this.world.height/2) {
    y = (this.world.height-Math.abs(y))*Math.signum(-y);
}
```

Algorithm 12: Source code snippet.

Chapter 5

Tests and calculations

5.1 Tests and calculations

In this section, the testing procedures and results for various tests will be described and explained. These tests will cover both the simulation tool and the control algorithm.

Some of the planned tests have unfortunately not been carried out due to lack of time towards the end of the project.

5.2 Testing the simulation tool

Testing the simulation tool covers the drawing of the balloon and how they move around. These tests are manual tests where the tool is used in different conditions to check if everything is displayed correctly.

5.2.1 Drawing and movement

Drawing the balloons properly is an essential part of the simulation tool. If the drawings are wrong it might confuse the user and cause wrong conclusions. This is ensured through manual tests where the following criteria are verified.

Figure 5.1 illustrates a balloon with direction $0.5 * \pi$ and preferred direction $0.7 * \pi$. Figure 5.2 illustrates a group of balloons.

Balloon, range and coverage
The balloon should be drawn at the correct position and the range and coverage should be indicated correctly.

Direction and preferred direction
The current direction of the balloon is indicated by a tip on the red balloon dot. The preferred direction is indicated by a red line from the center of the balloon to the edge of the coverage area. The two indicators must point in the correct directions in order not to confuse the user.

Balloons on top of each other
When a lot of balloons are grouped together, it is essential that the important details are visible. The information should be drawn in layers in the following order to ensure that all balloons are visible and that it is possible to identify gaps in the covered area visually.

1. Communication range
2. Covered area
3. The balloon dot and direction indications

Movement
The balloon should move as expected - in straight lines according to the direction of the balloon and seamlessly across the borders of the window.

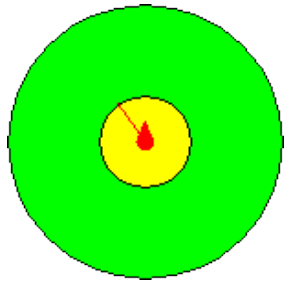


Figure 5.1: Single balloon

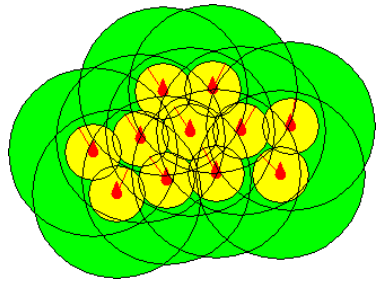


Figure 5.2: A group of balloons.

5.3 Testing the control algorithm

Testing of the control algorithm is divided into the following three categories.

- *Detailed tests* of the individual components of the algorithm.
- *Long term / large scale tests* for verifying correct behaviour during longer simulation periods and with a large number of balloons.
- *Paramter changes* where the effects changes to the parameters introduced in the design of the model and algorithm will be examined.

5.3.1 Detailed tests

Tests of the individual sub-components of the algorithm has been carried out throughout the project during the iterative development. The testing has primarily been a manual process of verifying correct behaviour and logging input and output of the functions.

5.3.1.1 Testing procedures

The manual tests are documented in the following testing procedures. These procedures can be used to verify the compoents when modifying the algorithm or extending the model or simulation tool.

For all tests it applies that it should work across the edges of the window of the simulator. It is important that the functions takes this into consideration and handles issue properly. An example is the relative distance calculation. Two balloons can be relatively close to each other even though they are drawn in each side of the window as if they were far apart.

Finding neighbours

This function should identify all balloons within two-way communication distance of a balloon, excluding the balloon itself.

This is verified by visual inspection and outputting the neighbours to the log.

The visual inspection is done using the debugging functionality built into the simulator. Clicking on a balloon marks it for debugging giving it

a black color. Neighbouring balloons will be marked blue. Pausing the simulation then gives a snapshot of the simulation, which makes it possible to verify that the neigbours are within two-way communication distance.

Outputting the neighbours of a balloon to the log makes it possible to verify the number of neighbours and that the list of neighbours does not include the balloon itself.

Finding overlapping neighbours

This can be verified by logging a list of all neighbours and a list of overlapping balloons, including the relative distance from the balloon to the neighbours. The balloons in the list of overlapping neighbours should be within the defined "overlap range".

The correct functionality can also be verified by visual inspection. If the overlap detection does not work properly a group of balloons will not be in the correct flower-pattern.

Calculating center

The center calculation can be verified by logging the positions of a balloon and its neighbours and the calculated center. The calculation can then be manually verified by calculating the average position using a spreadsheet or similar. An additional datapoint with position (0,0) should be added in this case, representing the balloon itself.

The center calcuation can be verified by visual inspection, by drawing lines from the center of each balloon to the calculated center. This helps verifying the calcuation for smaller groups of balloons where all balloons are each other's neighbours. In larger groups, this solution is not usable. This functionality is not enabled in the simulation tool but can be enabled by outcommenting a line of code.

Calculating direction

This is a relatively simple function that can be verified by feeding input and verifying output.

Changing direction

This function compares the a preferred direction to the avaliable directions in the wind layers, and returns the closest direction. The function can be verified by feeding it with various directions between 0 and 360 degrees and comparing the output to the directions of the avaliable wind layers. This should be tested with other wind layer configurations than just the north-south-east-west configuration which is used in this project.

This function is hard to verify visually since the preferred direction only indicates which direction the balloon wants to move in, and not which directions are actually possible to move in.

Move balloon
This is verified by logging the position, direction and speed of a balloon before and after a move. This information can be used to verify that the balloons has move correctly. Visually, the balloon should move in the direction that is indicated graphically.

Calculate relative distance The relative distance between balloons can be verified by logging the positions of two balloons and calculating the distance manually or using a spreadsheet. Here, it is essential that the relative distance handles the issue of balloons being located in opposite sides of the window, and thus seeming to be further away from each other than they actually are.

5.3.1.2 Test results

Table 5.1: Test results

Component name	Result	Comment
Finding neighbours	OK	Checked visually and by logging
Finding overlapping	OK	Checked visually and by logging
Calculating center	OK	Checked visually and by logging
Calculating direction	OK	Verified by manual running of the algorithm
Changing direction	OK	This function had an problem where the balloons had a tendency to move to the south. This was due to an error where the absolute difference between two locations were not calculated correctly. The problem was identified and corrected.
Move balloon	OK	Checked visually and by logging
Calculate relative distance	OK	Verified by logging and manual running of the algorithm

5.3.2 Long term / large scale tests

Long term / large scale tests will verify that the algorithm works as intended over longer periods of time and with a lot of balloons.

5.3.2.1 Efficiency - Number of direction changes

This test has not been carried out.

Purpose: Assess the resource consumption of the algorithm in terms of direction / altitude changes. This will give an indication of how effective the algorithm is in figuring out how to position the balloons. Especially if the algorithm is extended to predict the future path of a balloon, this test could be used to verify that the changes actually have an effect.

Procedure: Count the number of direction / altitude changes during each update and log the average count over time.

5.3.2.2 Stability - Balloons grouping, not splitting

This test has not been carried out.

Purpose: Verify that once the balloons have formed a group of balloons, the group does not split up. If the groups are broken up, it gives an unconsistent coverage. Preferably, all the balloons should eventually form one single group.

Procedure: Log the number of balloon groups for each update. This number should go towards one over a longer period of time.

5.3.2.3 Scalability - A lot of balloons

This test has not been carried out.

Purpose: The algorithm should be able to run on a large number of balloons. In practice, the balloon runs on a computer on each balloon which eliminates some problems, but the algorithm should be efficient enough to simulate a large number of balloons on a normal PC.

Procedure: Continously add a balloon to the world at a defined interval. Verify that the algorithm works with a large number of balloons.

Note: Drawing the balloons should be disabled to eliminate the resource consumption of the simulation tool, which is quite extensive.

5.3.2.4 Stability - Average distance between neighbours

One of the requirements is that the control algorithm should provide continuous and stable coverage on the ground. To test this, the average distance between neighbouring balloons is logged during an extended period of time.

The expected result is a situation where the balloons group together and stay in this group. The outcome should be a relatively stable average distance to neighbours. A drop in the average distance would indicate that the balloons are moving away from each other, thus not providing a stable, continuous coverage. An increase in the average distance would indicate that the balloons are excessively overlapping, thus not utilizing the potential covered area well enough.

The results from the tests can be seen in Table 5.2 and in Figure 5.4.

The preconditions for this test is as follows:

- 50 balloons at random positions
- Coverage parameter is 1.0
- Coverage is 25 for all balloons
- Optimal distance between balloons: $25 * \sqrt{3} * 1.0 = 43.30$
- Time period: 50,000 steps
- Logging interval: 25 steps

Test 1 (Figure 5.4(a))
The initial test shows a rather unstable situation where the average distance is far from the expected value.

Initially, the 50 balloons will be positioned at random around the world, which results in an excessive overlap. Quickly the balloons spread apart and enters a condition where their average distance fluctuates around 50.This is due to the

fact that balloons can see each other as neighbours, even though they are not next to each other. In Figure 5.3 this situation is illustrated. A balloon (black) is sorrounded by neighbours (blue). Two of the neighbours (purple) are not next to the black balloon, which disturbs the calculations in this test. This is solved in Test 2, where we only look at the nearest neighbours.

At around 12,000 steps, the balloons enter a completely stable condition. This is not a designed feature. According to the design of the algorithm, the balloons should not be able to enter a stable situation and align direction with each other. This is an implementation failure and will not be discussed further in this section. A potential solutoin is provided in the Discussion section 6.1.2.

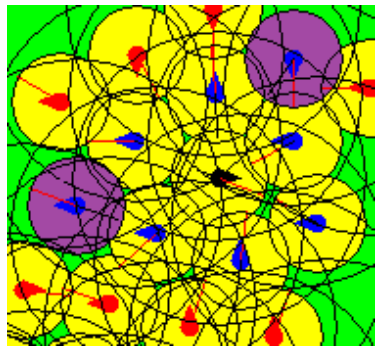


Figure 5.3: Neighbours too far away (Purple)

Test 2 (Figure 5.4(b))
In this test, only the neighbours within a radius of 2*coverage are contributing to the calculation.

This test shows a more stable coverage (smaller std. deviation and variance) with an average distance between neighbours averaging 1.71% higher than the optimal distance. While this is better than in Test 1, the average distance should be closer to the optimal distance.

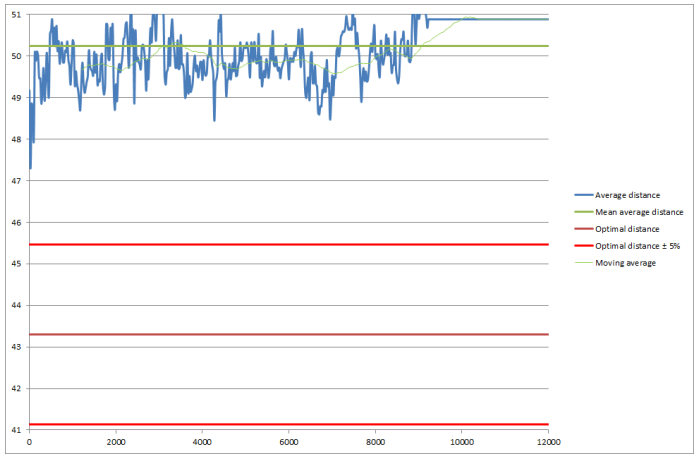
This difference might be caused by the fact, that the balloons does not enter a completely stable condition, but continuosly move around to adjust to their neighbours new positions. This can be eliminated by adjusting the coverage parameter introduced when designing the algorithm. In Test 3, the optimal distance is reduced by 1.71%, thus making the balloons move closer to each other.

Test 3 (Figure 5.4(c))
Now the average distance between neighbours is closer to the optimal distance. The difference is reduced to 0.59%, which is acceptable. This could be further optimized by running additional tests.

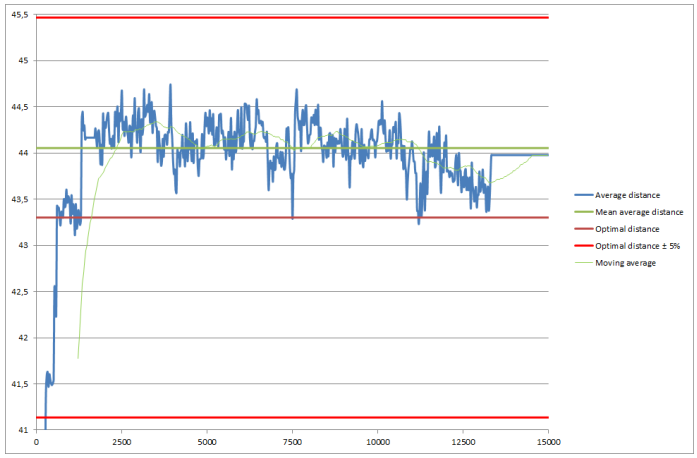
Results
The calculations of average distance, deviation and variance in Table 5.2 are based on the logged information from step 2500 and until the balloons align direction. Thus, the first 2500 steps does not contribute to the calculations. This is done to remove the the unwanted data from the first movements to keep focus on the situation where the balloons' movements are relatively stable.

Table 5.2: Results for stability tests.

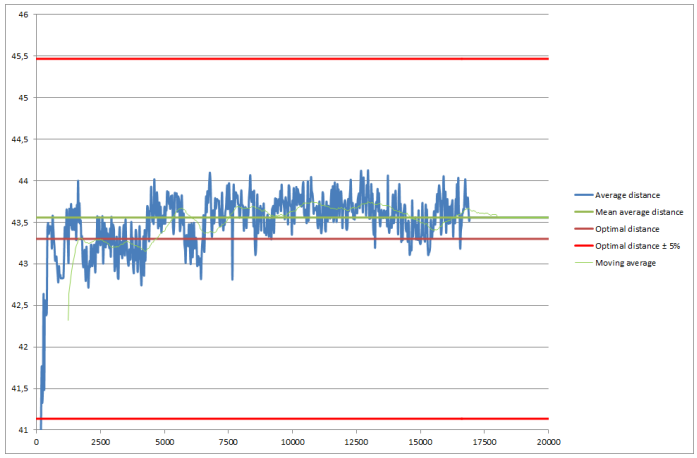
Test	**Test 1**	**Test 2**	**Test 3**
Optimal distance	43.30	43.30	43.30
Mean average distance	49.98	44.07	43.56
Difference from optimal	13.37%	1.73%	0.59%
Std.Deviation	0.651	0.258	0.235
Variance	0.424	0.067	0.055



(a) Test 1



(b) Test 2



(c) Test 3

Figure 5.4: Average distance between neighbours.

5.3.3 Changing parameters

These tests are experiments which can uncover the limitations of the algorithm, which might not be observed within the assumptions made in the design.

For further research, the following experiements could be interesting to carry out.

Changing wind directions
What happens if the wind directions change from the defined north-south-east-west configuration?

An example could be the following configuration with 5 wind layers: 300, 330, 0, 30 and 60 degrees. Here the balloons will only be able to move in one general direction - north.

Delayed reactions, gradual altitude change
How does algorithm work if their movement between altitudes were delayed?

The number of altitude changes could be limited to one per 10 time steps or similar. Or the change in altitude could be a gradual process over a number of time steps where the balloon gradually changes direction instead of an instant change.

Limited communication
What would happen if the communication between the balloons were limited?

If the communcation is limited, the data from neighbours is limited. How would the algorith work in this situation?

Alternating communication range and coverage area
What would happen if the communication range and covered area were dynamic and changed?

In practice, this could be the result from a limited power supply on some of the balloons, or alternatinv terrain beneath them. How does the algorithm handle this?

Balloons dropping out
What happens if a balloon falls to the ground or simply shuts down because of lost power? How does the algorithm fill up the gap that this balloon leaves in the coverage?

Centrally controlled algorithm
Would a centrally controlled algorithm be better for solving the problem of having the balloons move correctly?

The control algorithm in this project is based on a distributed algorithm where a control center is not needed and the balloons communicate with each other in order to move around. The centrally controlled algorithm may elimate some of the limitations faced during this project, even though it was assessed that this solution would be more expensive and thus less attractive.

Chapter 6

Discussion

6.1 Discussion

The goal of this project was to assess the feasibility of Project Loon by creating an algorithm to control the balloons and a simulation tool to visualize their movements.

The chosen solution for the control algorithm is a distributed algorithm where each balloon controls its own movements based on data about the conditions at its location and neighbouring balloons. The developed simulation tool is releatively simple, but it provides the graphical view of the balloons needed to assess the algorithm visually.

6.1.1 Simulation tool

The implemented simulation tool varies on several points from the ideas drafted in the initial design phase. However, the tool provides the functionality needed for testing the algorithm and visualizes the balloons and their movements as defined.

The interactive functionality is usable but not configurable through the simulator. Changes to the pre-defined configurations are only possible through the source code, which is not optimal but neither critical to the goal of this project.

Thus, the simulation tool works as expected and satisfactorily, but will require some work if the model is extended to a more realistic world where distances are not arbitrary values.

6.1.2 Control algorithm

While the control algorithm largely works as defined, a few problems have arised during development and testing which have not been prioritezed to solve within the time frame of the project. Aside from these problems, the control algorithm works satisfactorily.

Occasionally, a large group of balloons aligns direction and then moves together in one direction. This situation is not defined in the design of the algorithm and is not intended. Why it occurs is uncertain, but it might have something to do with the order the balloons are updated in. The balloons are loaded into a list and the system iterates through this list in the same order on every update. If

the order was random or the implementation was based on a thread for each balloon, this situation might not occur. Furthermore, this situation is not entirely unbeneficial and should be anticipated in the design of the algorithm. A fourth situation in the main algorithm could be introduced, where the balloons align direction when they are within the correct distance from each other.

Another issue is that it is possible for one balloon to split up a larger group of balloons. This is a design failure in the center calculation where the balloons are figuring out how to move closer to each other. If one balloon (A) approaches a group of balloons, it might make contact with one of the outermost balloons (B) in the group before reaching the others. This can potentially change the average position of balloon B to be towards the approaching balloon (A) rather than towards the center of the group and thus it will break away from its group.

6.1.3 Future improvements

Looking forward, it would be interesting to extend the model and the simulation tool to support a more detailed description of the world. The simplyfied flat world could be made more realistic by introducing a more complex topography with terrain in different altitudes and removing the assumption that the altitude of a balloon does not influence the covered area below it. Differences in altitude would increase the communication distance between two balloons, as the vectorial distance between the balloons would be based on all three dimensions rather than two. Furthermore the wind layers could be extended to support more complex configurations and ultimately to use real world wind data. These changes could potentially help identify more complex issues with the concept of flying balloons around the world which have not been uncovered yet.

The control algorithm could be extended to take more information into consideration when chosing the direction it should move. As is, the algorithm is currently only based on the position of the balloons. This could be extended to look at the speed and direction of the balloon and its neighbours and predict an optimal path through several time intervals, rather than just looking at the current situation. This could improve the effectivity of the algorithm and make the balloons group together faster and perhaps even with less overlap and fewer gaps.

It could be interesting to further push the limitations of the algorithm as well. Currently, the balloons are free to move between wind layers at no cost. In practice, this is a complex maneuvre which consumes energy which is a limitation

in respect to the power supply from solar cells and batteries. Limiting the number of altitude changes and thus the energy consumption, and figuring out a solution to handle this limitation would be a nice addition to the algorithm.

6.2 Evaluating the project

A lot of challenges, especially related to timing and planning were experienced during the project. However, this process also caused a lot of learnings.

At the beginning of the project, an initial project plan was drafted. The plan was to start up the project by researching, sketching and developing small prototype programs to test ideas. This work should result in a problem analysis with defined goals for the project as well as narrowing the focus down to some specific problems. With a problem definition in hand, the iterative process of analysing, designing, implementing and testing could begin. Finally, the last three weeks of the project was devoted to documentation and report writing.

As seen in the final project plan in Appendix A the project did not go as expected, as some problems took longer to solve than anticipated. This was partly due to the vague project plan and partly due to not following the iterative process strictly.

The project plan should have been more extensive making use of project management theory. The problems and tasks could have been broken down into smaller parts with mile stones indicating when things should be finished in order not to get behind schedule. Based on these mile stones and activites, the critical path through the project should be identified. This would have made a clearer overview of the project and could have identified critical tasks which should not be delayed.

The iterative process was not followed in the beginining of the project. The actual problem definition and requirements took a while to work out and was defined after the work on designing and implementation started. This resulted in an unclear idea of which problems the project actually was going to solve and lost focus on the tasks that were important at the moment.

Later in the project, these issues were identified and more structure was added to the work on the iterative design and implementation. However, the implementation was not finished before the last three weeks as planned, giving less time for documentation and report writing.

To summearize, this project caused the following learnings. Correct planning of the project phases and identifying critical path is essential. Following the project plan, especially during the start of the project, is critical to avoid piling up requirements for extra resources and unexpected workload during the finishing phases of the project. The iterative approach to development helps keeping focus on the individual task and should be followed in order to finish activities and move on.

Chapter 7

Conclusion

7.1 Conclusion

Creating a network of balloons to provide internet across the world is a substantial task. However, based on the reflections done throughout this project, the feasibility of Project Loon is assessed to be realistic.

This project shows, that it is possible to create a distributed algorithm to control the balloons' flight to provide a stable coverage on the ground in a simplified world.

With additional time, a more thorough investigation using real world wind data could be carried out to uncover potential problems which have not been met in this project. Additionally it would be advised to test a centrally controled algorithm using existing communication networks (such as satellites) to provide a constant connection to all balloons, to feed them with data and manual control orders. While this would create another layer of complexity to the project and increase the operating costs, this could make a more efficient algorithm possible as well as providing a security that the balloons do not get lost.

Appendix A

Project Plan

x = forventet periode og tidsforbrug
o = faktisk periode og tidsforbrug

	Uge	Uge 7	Uge 8	Uge 9	Uge 10	Uge 11	Uge 12	Uge 13	Uge 14	Uge 15	Uge 16	Uge 17	Uge 18	Uge 19	Uge 20	Uge 21	Uge 22	Uge 23	Uge 24	Uge 25	Uge 26
	Dato (onsdag)	11-feb	18-feb	25-feb	04-mar	11-mar	18-mar	25-mar	01-apr	08-apr	15-apr	22-apr	29-apr	06-maj	13-maj	20-maj	27-maj	03-jun	10-jun	17-jun	24-jun
Generelt	Projekt start og tidsplan	x	x																		
			o	o																	
	Aflevering																				x
																					o
Afgrænsning	Informationssøgning og data		x	x	x																
			o	o	o																
	Afgrænsning af projekt			x	x	x															
						o	o	o		o											
Design	Modellering					x	x	x													
				o	o	o	o	o													
	- Geografi / Udformning					x	x	x													
				o	o	o	o	o													
	- Vind					x	x	x													
				o	o	o	o	o													
	- Ballon					x	x	x													
					o	o	o														
Implementering	Implementering af algortime				x	x	x	x		x	x	x	x	x	x						
								o		o	o	o	o	o	o		o	o	o		
	Implementering af simulator (verden)					x	x	x		x											
				o	o	o	o	o		o	o	o									
	Implementering af simulator (ballon)					x	x	x		x	x	x									
					o	o	o	o		o	o	o									
	Udvikling af simulator (beregninger og data til tests)							x		x				x	x						
																	o	o	o		
Test	Tests og verificering af del-komponenter				x	x	x	x		x	x	x	x	x	x						
								o		o	o	o	o	o	o		o	o	o		
	System test (large scale/long term/ænding af parametre)													x	x						
																			o		
Rapport-skrivning	Opsætning af rapport skabelon				x																
					o																
	Rapportskrivning og dokumentation					x		x		x		x		x			x	x	x	x	x
								o					o	o	o		o	o	o	o	o

påske

eksamener

Appendix B

User guide

Using the simulator

The simulation tool started by running the executable jar file. By default, the window will be empty.

To start the simulation, select one of the predefined configurations (keyboard shortcut 1-9) or load 50 random balloons (keyboard shortcut 0).

The simulation can be paused by pressing space. Use page up and page down to increase or decrease the simulation speed.

Keyboard shortcuts

To interact with the simulation tool, use the following keyboard shortcuts.

1-9
: Load different predefined configurations.

0
: Load 50 random balloons.

Space
: Pause simulator.

Page Up / Page Down
: Increase / decrease simulation speed.

Debugging

The simulation tool allows the user to log information about a balloon and its neighbours. Only one balloon can be selected for debugging at a time. The selected balloon will be marked with a black color and its neighbours will be marked blue as seen in Figure B.1.

It is possible to control the balloon marked for debugging by using the arrow keys on the keyboard.

To view the logged output, the user must run the executable jar file through a terminal. On windows, execute the command *java -jar SimulationTool.jar* in the command prompt.

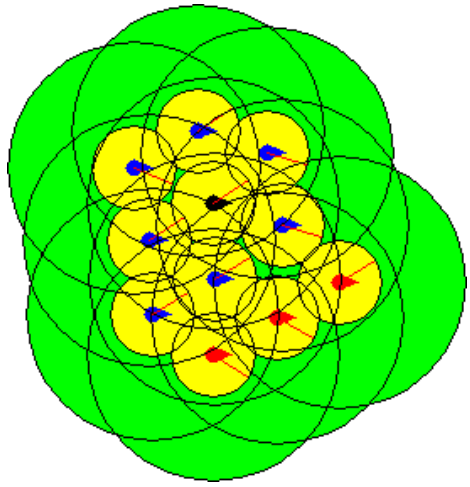


Figure B.1: Debugging a balloon.

Bibliography

[ESR] Earth System Research Laboratory data plot tool. http://www.esrl.noaa.gov/psd/cgi-bin/data/composites/printpage.pl.

[GOO] Google X on Wikipedia. http://en.wikipedia.org/wiki/Google_X.

[INT] Connecting the World from the Sky. https://fbcdn-dragon-a.akamaihd.net/hphotos-ak-ash3/t39.2365-6/851574_611544752265540_1262758947_n.pdf.

[LOOa] Project Loon on YouTube. https://www.youtube.com/user/ProjectLoon.

[LOOb] Project Loon website. https://www.google.com/loon/.

[SPA] QUESTION 7: PURPOSE OF EXPERIMENT. https://qzprod.files.wordpress.com/2015/06/spacex-satellites.pdf.