UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

DEVELOPMENT OF ELEMENTARY BIM TOOLS FOR "DRONE-COMPATIBLE" CONSTRUCTION SYSTEMS

Thesis presented by
Milan RENIERS

for obtaining the master's degree in
Master ingénieur civil des constructions

Supervisor
Pierre LATTEUR

Readers
Sébastien GOESSENS & Fabian FRANZINI

Academic year 2015-2016

Acknowledgements

I would like to acknowledge the assistance of the following people:

- Prof P. Latteur and Mr S. Goessens, École Polytechnique de Louvain (EPL), Université Catholique de Louvain (UCL), for guidance and supervision of the research;

- All staff members involved with the EPL innovation classes supported by the International Lhoist Berghmans Innovation Chair, especially Quentin d'Aspremont for closely following up our work during the whole classes and his precious advice and feedback;

- The Lhoist Berghmans Innovation Chair for funding my travel to MIT;

- Ms A-L. Cadji for setting up the logistics for my travel to MIT;

- Mr M. Van Cauteren, Scientific Liaison Officer Wallonia-Brussels International, for advice, encouragements and guidance in Boston;

- Prof Mueller C., Department of Architecture, Massachusetts Institute of Technology (MIT), for receiving me and my co-workers at MIT;

- Mr J. Shearman, student at Massachusetts Institute of Technology (MIT), for technical explanation about their work on drones and for showing me around in their laboratory.

- Mr A. Moncourrier, student at EPL, UCL, for the creation of various numerical 3D models used throughout my thesis.

- Mr N. Nehri and Mr A. Paques, students at ECAM, Brussels Engineering School, for their technical advice and guidance regarding the drone communication protocols;

Table of contents

Definitions

Here are a few definitions providing the reader with the necessary knowledge to understand this thesis:

BIM

The National Building Information Model Standard Project Committee defines BIM as:

"Building Information Modeling (BIM) is a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition". [1]

Biomimicry

Biomimicry is theorized by Janine Benyus in 1997 and is defined by the Biomimicry Europa association as :
"An innovation process encouraging the transfer of ideas, concepts and strategies inspired from the living world, with the objective of designing human applications aiming at a sustainable development". [2]

CSS

The ISO definition of a CSS file is:

"Cascading Style Sheets (CSS) are used to apply style (formatting) to HTML and XML documents". [3]
As the html file defines the structure of the web page or interface, the CSS file defines the look and feel.

CSV

Bruce Dunwiddie, founder of CSVreader.com defines the csv file format as follows:

"CSV, comma separated values, files are commonly used to transport large amounts of tabular data between either companies or applications that are not directly connected. The files are easily editable using common spreadsheet applications like Microsoft Excel". [4]

DRICK

Drone-compatible brick.

HTML

The ISO definition of an HTML file is:

"Hypertext Markup Language is the main mark-up language for Web pages, consisting of elements which are used to add structural and semantic information to raw text". [5]
Thus it allows to format the contents of the pages. [6]

JavaScript

JavaScript is a programming language principally used in interactive web pages and servers. Its particularity is that the code is executed on the user's computer and not on the server. The execution of the code is done by the web browser.

ROS

The Open Source Robotics Foundation defines ROS as:

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms". [7]

STL

STL stands for Stereo Lithography. The STL file format is originally invented by 3D systems in the context of CAD software. It is mainly used for rapid prototyping, 3D printing and computer aided design. [8]
An STL file represents a 3D object tessellated into triangular facets. The STL file lists the [x, y, z] coordinates of the three vertices for each triangular facet and the [x, y, z] components of the vector normal to the facet. [9]
It does not contain any information about scale, units or colors.

SVG

SVG is a language describing 2D graphics. It describes images with characteristics such as gradients, transparency, effects, filters and animations. It allows three types of graphic object: vector shapes (lines, curves, ...) images and text. Like Flash and unlike Gif or Jpeg formats, an SVG image is not based on a set of pixels but on geometric shapes (circles, squares, triangles, etc.). [10]

WebGL

WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D and 2D graphics. It can be used without needing to install additional plug-ins and this in any compatible web browser. [11]

YAW

The National aeronautics and space administration (NASA Official) defines the yaw axis as:

"The yaw axis is defined to be perpendicular to the wings of the plane with its origin at the center of gravity and directed towards the bottom of the aircraft. A yaw motion is a movement of the nose of the aircraft from side to side." [12]

INTRODUCTION

Research justification

Economic impact

What is the common point between the following constructions?

Figure 1 The great wall of China, the "Gard" bridge in France, the pyramid of Cheops in Egypt [i1, i2, i3]

For sure, they all have been built a long time ago, but they also have been built in outrageous conditions: no respect for human rights and with very low even non-existent safety conditions. What if you were told this is still going on today in some regions of the world? Unfortunately, this will be the case until a less expensive solution than human labors exist.

What could be a cheaper solution? A drone-enabled construction.

Drones and drone-compatible bricks can not only replace workers in countries where the safety and general working conditions are weak, but they can also be sent by plane in devastated areas of the globe. At any time on this planet, someone is in need for rebuilding his home after a hurricane, a tsunami, a flood or even a war. And in this case only a small work force is needed to provide a new roof to the poorest who have often lost everything and who are too sick or too exhausted to work.

You might wonder right now: "Will all workers lose their jobs"? No, but they will most probably not exert the same job anymore. With these changes new jobs will be created that will eliminate the unsafe work conditions: drone assembling, special molds and brick creation, etc. This drone solution will require a huge amount of electrical energy, therefore, more jobs will also be needed in the energy sector and so on."

A revolution

In his conference "L'économie de la connaissance" the Professor and researcher Idris Aberkane states that "*Every revolution in the history of mankind that is scientific, cultural, political, moral, philosophical, technological or managerial goes inevitably through three steps:*

At first it is considered as ridiculous;
Then it is considered as dangerous;

And finally it is considered as obvious. " [13]

Then, he illustrates this by comparing it with the women's right to vote. Two hundred years ago it would have seemed ridiculous that women could have a political opinion. Nowadays this right seems totally obvious.

The drone revolution is just at its beginning and yet many applications have already been found for these flying robots. Nowadays, drones are used for videography, mapping, disaster response but also in the construction sector for dam and bridge inspections with the aim of identifying defects, leakages or cracks.

Designs are evolving fast and drones can lift gradually more weight with the passing days. Today the Université Catholique of Louvain (UCL) is the proud owner of a 2.5 meters wide drone.

The utility to have such a large drone is to be able to carry heavy loads. The UCL octocopter can carry at least 30 kilograms and weights itself 60 kg. The aim is using it to carry and lay bricks in position on a work site.

Biomimicry in the design and build process

Rethinking the world of construction, is an idea that animates Pierre Latteur, professor at the École Polytechnique de Louvain (EPL), attached to the Institute of Mechanics, Materials and Civil Engineering (iMMC) and the Civil and environmental engineering (GCE) division. His vision is to supplement or even to replace the construction cranes and manpower by drones. His inspiration took its origin in the observation of swallows that build nests by constantly flying in short trips to transport and assemble mud and twigs into a structure capable of supporting eggs and offspring. Other kinds of flying animals such as bees proceed in a similar way to build complex forms (see figure 2).

Figure 2 Complex structures built by animals. Left: swallows building their nest. Right: A honeycomb [i4, i5]

The feasibility of a construction approach based on biomimicry using unmanned aerial vehicles, commonly called drones has already been investigated and validated last year by Jean-Sebastien Breton and Justin Leplat in their thesis written under the supervision of Professor Pierre Latteur at UCL [14].

The design and build process suggested at that time was:

1. Designing of the building by the architect and the engineer;
2. Modeling of the building with a CAD tool;
3. Translation of the CAD model into drone compatible remote control instructions;
4. Assembling of the structure with the drones.

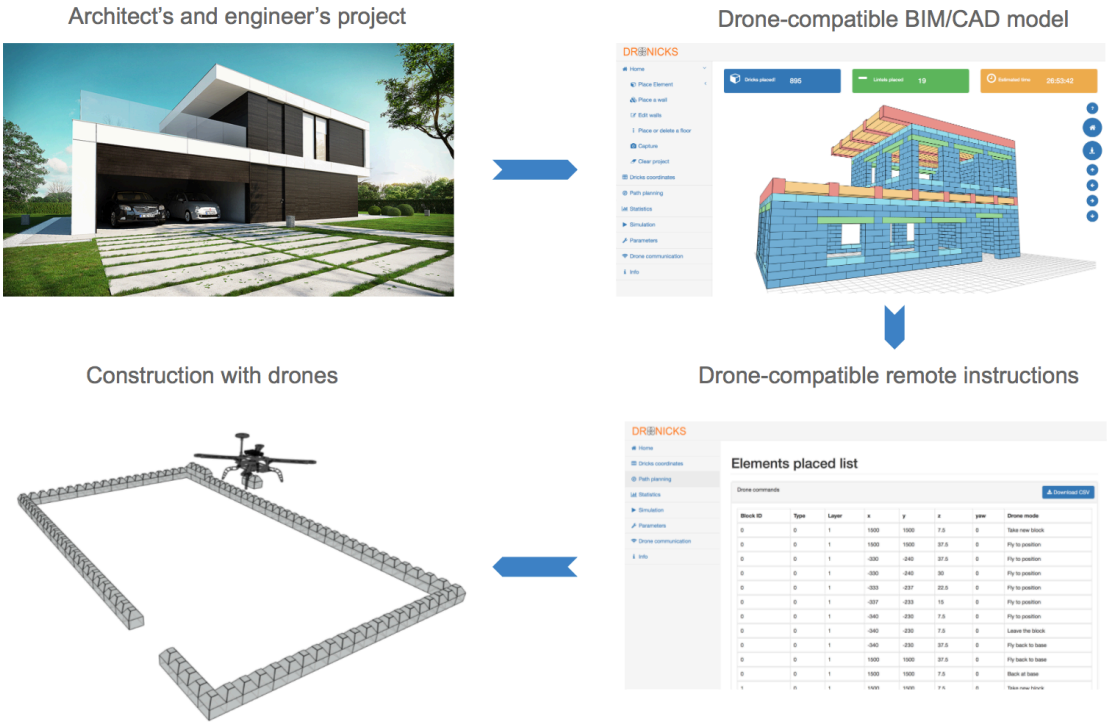


Figure 3 "Drone-compatible" design & build process

Similar works around the world

In 2011, researchers led by R. D'Andrea at ETH Zurich in association with the Swiss architecture firm Gramazio Kohler Architects, already demonstrated the feasibility of a construction mode based on rigid rectangular bricks placed above each other. They accomplished the construction of a six meters tall tower composed of 1500 foam modules assembled by 4 quadcopters (see figure 4) [15].

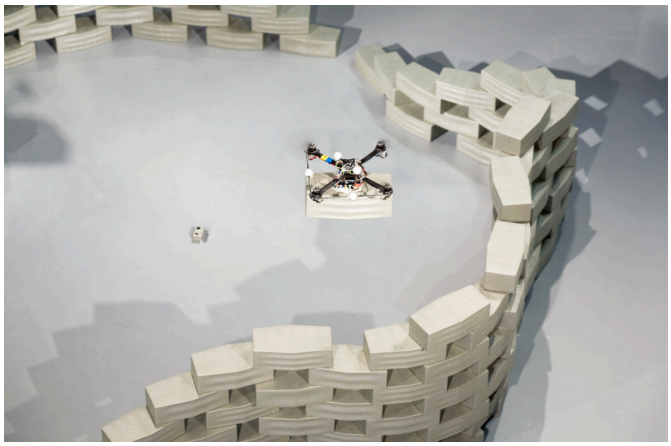


Figure 4 Construction with bricks placed by drones (ETH Zurich/Prof R. D'Andrea) [i6]

Professor R. D'Andrea's research group also investigated on how to "weave" simple tensile structures in the air with cable dispensers (see figure 5) [16].

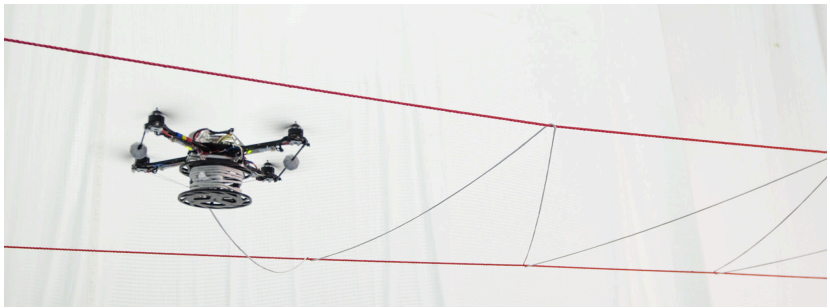


Figure 5 Tensile structure made of cable (ETH Zurich) [i7]

Finally, Quentin Lindsey, Daniel Mellinger, and Vijay Kumar from the University of Pennsylvania used teams of quadrotors to build tower-like cubic structures from modular parts assembled with magnets [17] (see figure 6).

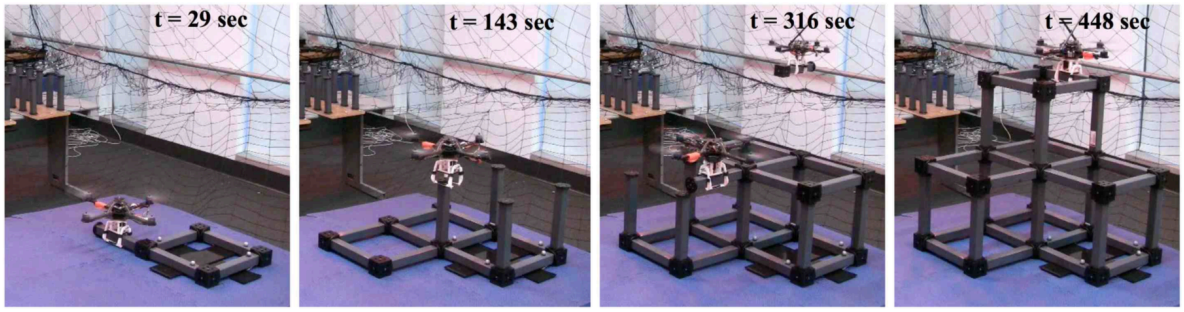


Figure 6 Intermediate snapshots of a pyramid-like Special Cubic Structure being built by three quadrotors (University of Pennsylvania) [i8]

Towards safer drones

As the drone revolution passed the phase of "ridiculous", it is now trying to solve one of its major issues: safety. Many new kind of safer drones have already been designed. To achieve the safety goal, the propeller(s) is (are) concealed inside an outher shell. One of them is called "Fleye". It has the shape and weight of a soccer ball and it can be held in hands safely as its single propeller is concealed within a sphere (see figure 7(a)) [18].

Another new kind of drone developed by Professor R. D'Andrea's research group, called the "Flying platform" uses electric ducted fans as control and propulsion systems (see figure 7(b)) [19].

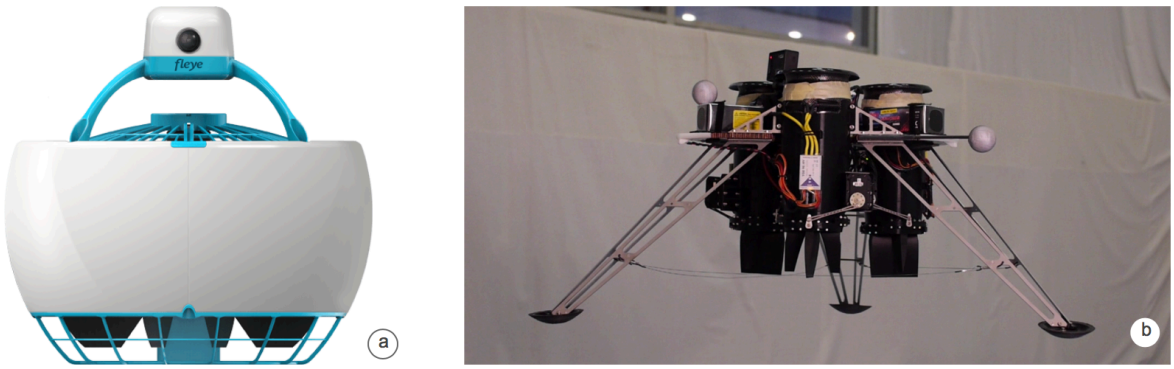


Figure 7 New safer drone types [i9, i10]

New designing mentality and brick types

While these developments are promising, there is a need for further investigation into structural units that can be used additively in drone-based construction, where less precision is available compared to manual or other robotic methods.

Sebastien Goessens, explains during his interview in the tv show of the RTBF « Quel temps!» broadcasted on may 23rd 2016 that the droxels (see figure 8) developed during his PhD at UCL might not necessarily be the final form that will be used in drone-compatible construction but that these forms will evolve with time and as the studies goes on. New blocks will be created with more complex forms that will allow new kind of constructions that will be realized in less time and in an easier and safer way by drones.

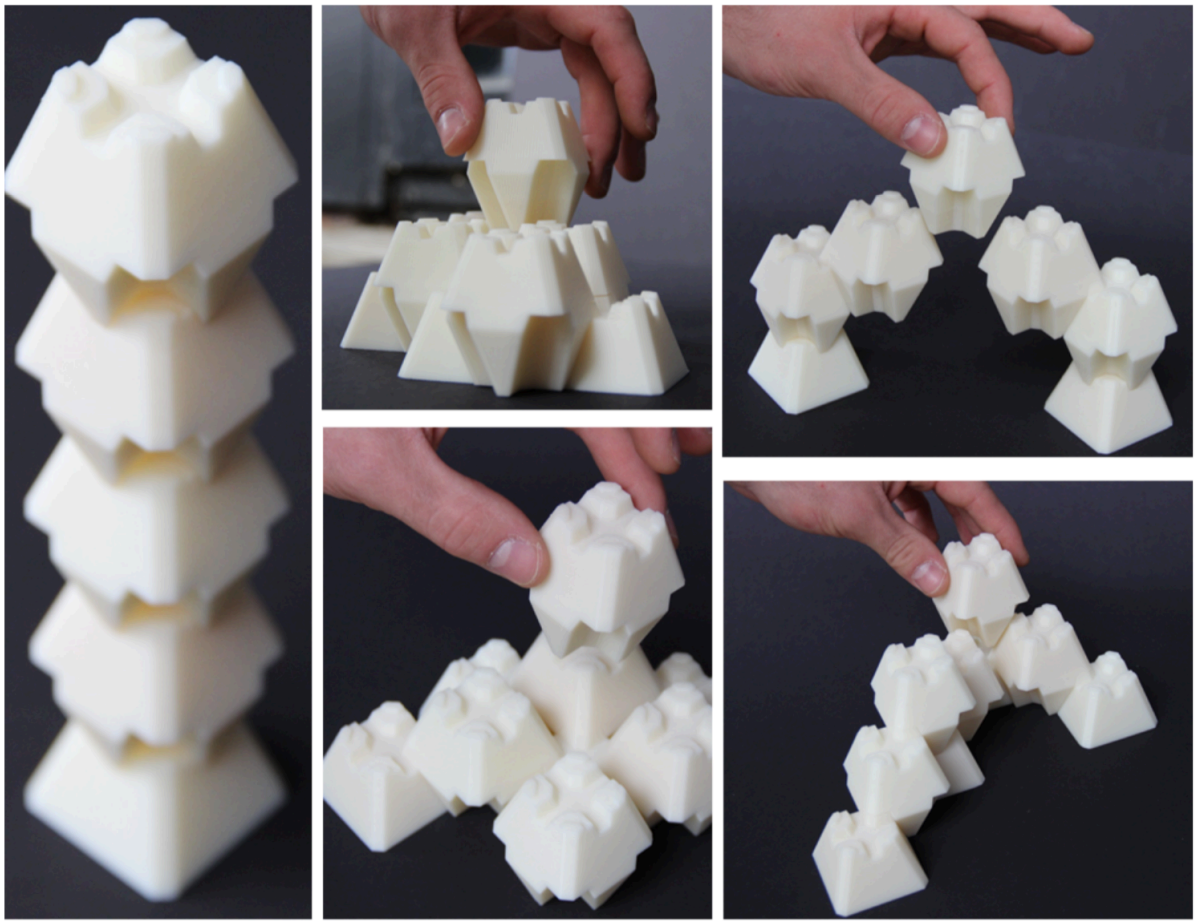


Figure 8 Structural configurations assembled with 3D printed Droxels

Considering the usage of drones in the construction sector will also require a significant mentality shift in the way buildings are conceived. According to him, buildings will not just be created with other elements and techniques but most probably the building architecture will move towards new shapes and building types that will be linked to the new construction elements used.

Drones provide interesting features when it comes to the development of a new construction method. Indeed, drones are able to move in space and have access to places in a way men or a cranes can't. Furthermore, the construction process could be safer, more flexible, faster and better coordinated with the usage of drones. This applied research project in collaboration with Prof. Caitlin Mueller of Massachusetts Institute of Technology aims to reduce the costs of defects caused by human or external factors, which can represent up to 10% of the total amount of a structure. While promoting the training of qualified personnel in a leading sector, it will also increase safety on site and will probably allow a higher speed of construction.

Thesis purpose

The purpose of this thesis is to develop elementary BIM tools for modeling a small size structure and to translate the 3D model into drone compatible remote control instructions.

In parallel, Amaury Moncourrier and Adrien Naveau, fellow students at the École Polytechnique de Louvain (EPL), UCL, are developing new shapes of drone-compatible bricks (dricks) and a lifting system.

Nabil Nehri and Alexis Paques, students at ECAM., Brussels Engineering School, are developing an accurate positioning system for the drone so it can define its coordinates in a well-defined reference frame. They are giving a first layer of artificial intelligence to the drone.

Finally, Charles Coppieters de Gibson, fellow student at the EPL, UCL, is studying the economic aspect of the subject.

Thesis objectives

This thesis aims to make drone-based constructions possible by following three main objectives:

Development of a user friendly elementary BIM interface for drone-compatible bricks (Chapter 1)

The objective is to develop an interface containing only the elementary structure units like bricks, beams, girders and slabs that will be used in the assembling of a simple structure such as a house.

Definition of drone-compatible building processes for basic structures (Chapter 2)

The objective includes defining the approach that best suits to design a structure, and with the intention to build it according to a drone-based construction method and finally, defining how the model will be translated into drone compatible instructions.

Development of algorithms controlling the construction process (Chapter 3)

The objective aims to create a second layer of artificial intelligence for the drone by predefining its journeys in the bricks laying process. Also, the most cost-effective stockpile position will be defined. Finally, a visual numerical simulation of the drone's behavior will be implemented in order to predict and avoid collision with already placed parts of the construction.

Each of these objectives is covered in a specific chapter.

CHAPTER 1

DEVELOPMENT OF A USER FRIENDLY ELEMENTARY BIM INTERFACE FOR DRONE-COMPATIBLE BRICKS

What is a BIM program?

In a Building Information Modeling (or also called Building Information Management) program, every elementary object such as a wall, a window or even a door has an appearance that is rendered in a 2D or 3D visualization frame but has also many other information linked to it. For instance, a beam will hold information such as its coordinates and orientation in the model, the fabrication cost and manufacturer etc. A BIM project allows many stakeholders to use the same file from the design until the conception and follow up. Knowing that the purpose of the program is to design a structure and to follow up its construction, the program should be usable and understandable by anybody. Principal stakeholders will be the designer and the foreman who use the same program and model it at different stages of the project.

Why coding a new BIM program?

User-friendliness

One way to solve the problem could have been to use the "*Dynamo*" add-on in the *Revit* program[1]. This free and open source program is supported by the Autodesk community. It allows visual programming by using functions and variables as boxes. Wires connect those functions and variables together creating a visual program.

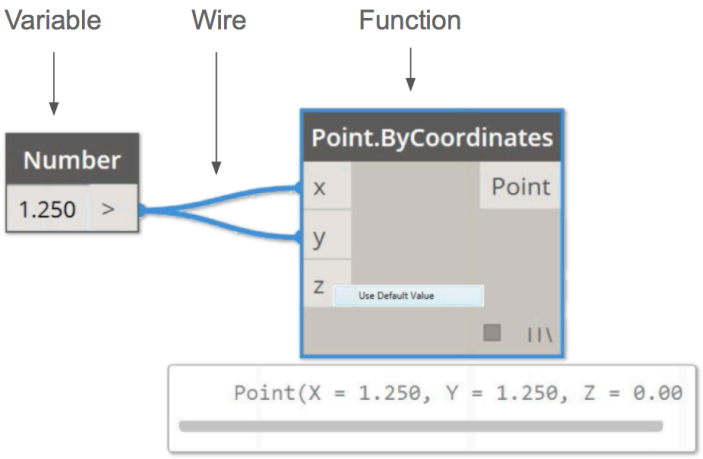


Figure 9 Dynamo elementary programming tools

[1] Other visual programming tools such as « Grass Hopper » are also convenient.

Dynamo is a very effective tool for creating small algorithms that avoid repetitive tasks in *Revit*.

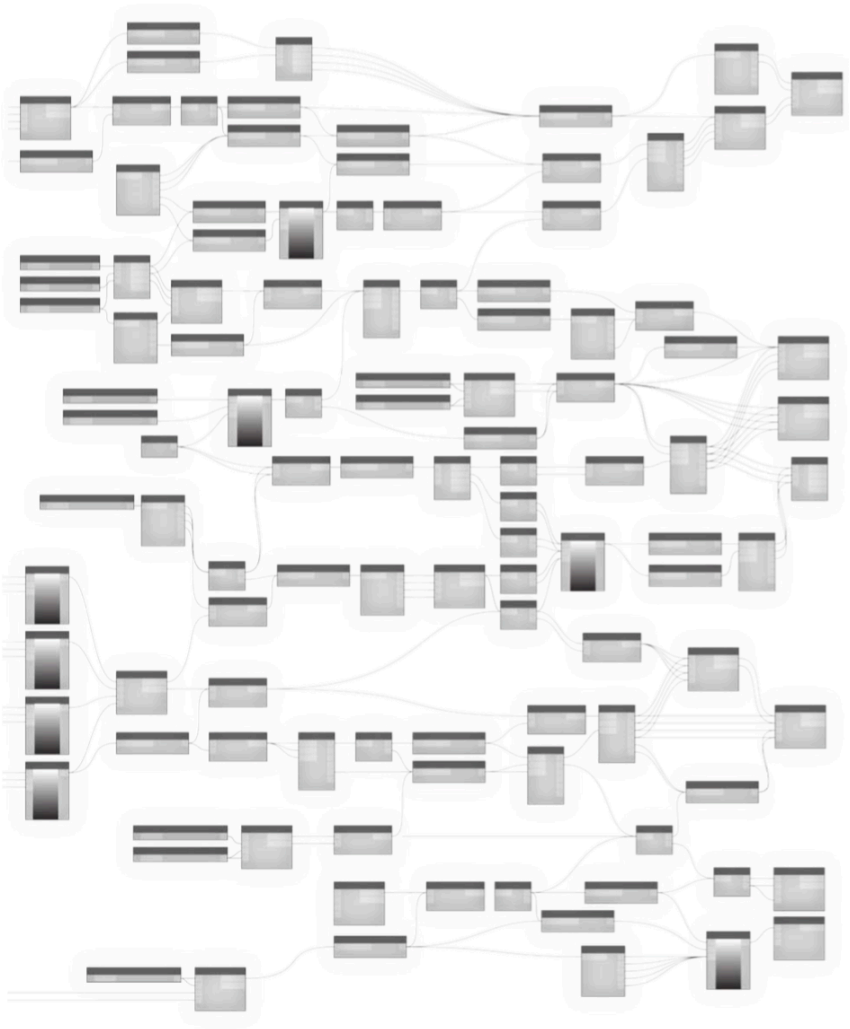


Figure 10 Small Dynamo program [i11]

Unfortunately, those wires can get confusing for the user and the pattern can get messy as the program grows. Moreover, so far there is no easy way to create a graphical user interface. This means that whoever wants to use the program needs to know how to visual-program and how it works. Furthermore, if a sub-routine does not exist in the predefined library, the user has to program it in the Python programming language. Therefore, it is not a user-friendly program.

Today the most used BIM program is *Revit* from the *Autodesk* suite. In this program one can draw different types of walls. Unfortunately, every drawn wall is one single object. What one needs for drone-enabled constructions is to know the coordinates of every brick composing this very wall. This is why there is a need for coding a new program that will allow drawing walls with the same ease as in *Revit* but with the generation of different information (the coordinates of every drick composing the wall).

A final user and mobile oriented solution

The main idea of this thesis is to build a program that can be modified and improved in the future and that will evolve with time. Another aspect to keep in mind is that the final user will be at the same time the designer and the foreman. This implies that the application needs to be easy to use and to understand in an office as well as on site, by the designer and by the foreman. Therefore, an intuitive graphical user interface is needed.

The financial, energy and commercial sector are already far ahead in the mobile revolution. One has just to think about mobile banking, mobile energy follow-up or even mobile food ordering to notice that this has already a place in our daily lives. What about mobile solutions on work sites? This is getting increasingly common and this is why a mobile solution is utterly convenient.

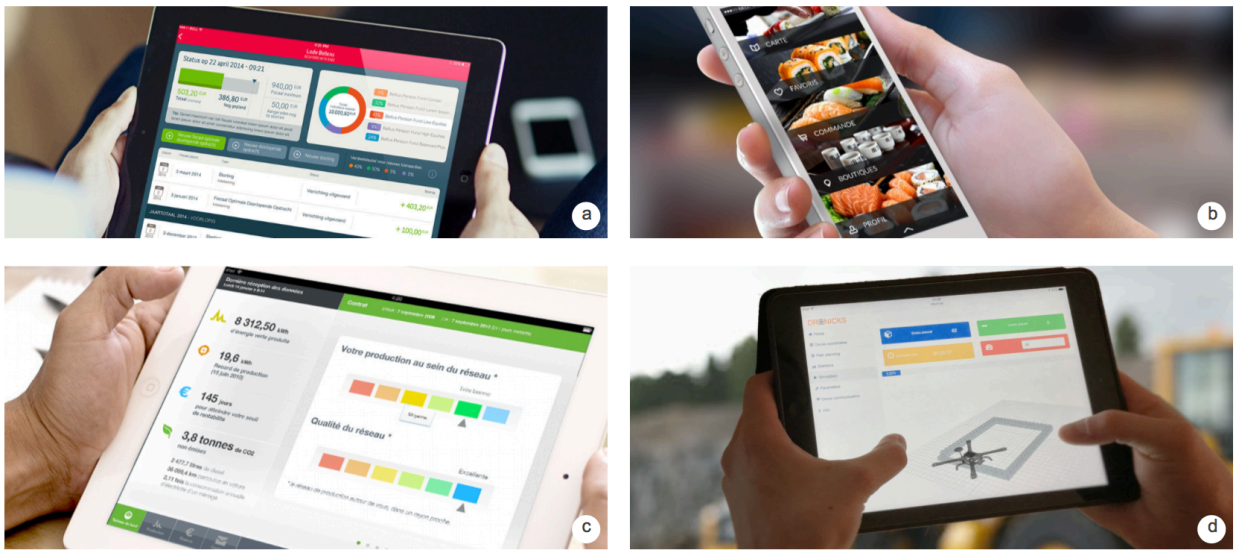


Figure 11Mobile banking (a) Mobile food ordering (b) Mobile energy consumption/production follow-up (c) Mobile work site monitoring (d) [i12]

A versatile solution

For a program to be versatile it should be cross platform. The easiest way to deal with this issue is to create a website application. It's accessible from about any tablet or computer and accessible by everybody on the globe. The solution implemented in this thesis uses WebGL for rendering 3D graphics. So for the device to be compatible, its web browser should be able to use or allow the use of WebGL and the device must also have hardware that supports WebGL.

Program overview

Dronicks

Figure 12 Dronicks program logo

The program that was designed is called *'Dronicks'*. The "portmanteau" *'Dronicks'* is the blend of the words drone and dricks.

Dricks is itself the "portmanteau" of the words drone and bricks meaning they are drone-compatible bricks.

Dronicks is an elementary BIM tool for designing drone-compatible houses or structures. It also allows along many other features, to simulate the erecting process of the designed model in a 3D frame.

The program's graphical user interface features

Home page

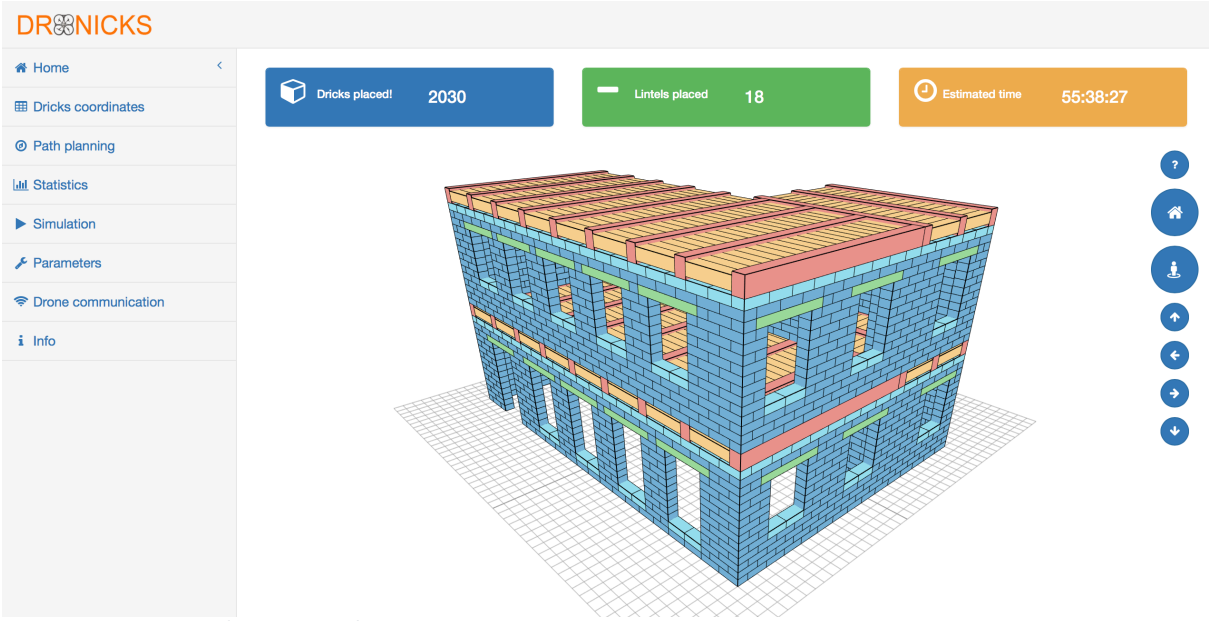


Figure 13 Capture of the 'Dronicks' home page

Placing and removing elements one by one

The user can place and remove single elementary construction units. From left to right these "blocks" are named a double drick, a mono drick, a topper (because it goes on top of a wall to have a horizontal finish), a half topper, a lintel (6 times the width of a mono drick), a girder, a slab.

Figure 14 Different block types in home page

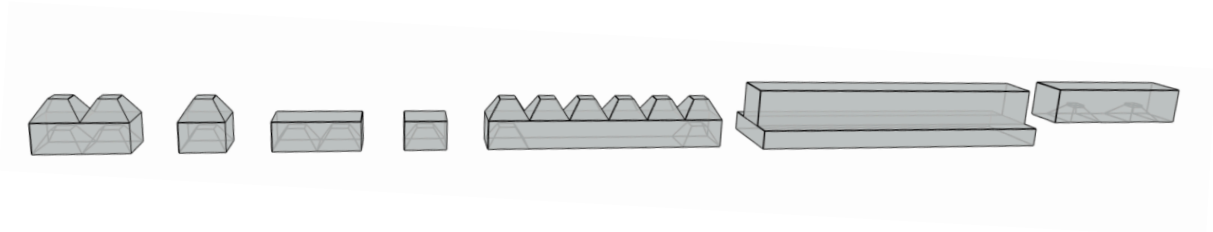


Figure 15 Different block types in the simulation page (STL models)

Screen captures

The user can also take a capture in PNG format of the scene.

Total process duration

The total process duration, showed in the yellow "Estimated time" label is here computed by summing up the duration of each drick placement. For each drick the placement duration is the sum of:

Time to grab a drick[2] in the stockpile;

Time for a vertical flight[3] up between the altitude of the drick in the stockpile and the altitude of the final position of a drick + a safety margin of 2 times the drick height;

Time for a horizontal flight[4] between the [x, y] coordinates of the stockpile and the [x, y] coordinates of the drick's final position;

Time for a vertical flight down of 2 dricks height (the drick is now in final position);

Time for placing and releasing[5] the drick in its final position;

Time for a vertical flight up of 2 dricks height (to have a higher altitude than the structure so that the drone can fly over it);

Time for a horizontal flight[6] between the [x, y] coordinates of the drick's final position and the [x, y] coordinates of the stockpile;

Time for a vertical flight[7] down between the altitude of the final position of a drick + a safety margin (2 times the drick height) and the altitude of the drick in the stockpile;

[2] Value entered by the user in the parameters page.
[3] Based on the vertical velocity the user entered in the parameters page.
[4] Based on the horizontal velocity the user entered in the parameters page.
[5] Value entered by the user in the parameters page.
[6] Based on the horizontal velocity the user entered in the parameters page.
[7] Based on the vertical velocity the user entered in the parameters page.

Placing a wall

In this page (see figure 15) the user can place a wall by clicking on the first corner of the wall and then by moving the mouse vertically or horizontally, then by clicking a second time placing the second corner of the wall. If the user moves the mouse vertically or horizontally, and clicks a third time it will place the third corner of a sequence of walls and so on. The wall that is being drawn is represented in dark grey. A dark blue line represents an already placed wall on the actual level and a light blue line represents a wall on the level under the actual level. Diagonal walls are not allowed for now.

The user can also choose on which level to draw. When the user places a wall that is not on the ground floor, it automatically (except if the user unchecks the option) will place the lowest layer of the new wall 2 layers above the last layer of the wall underneath it. This allows the user to place a floor between two levels.

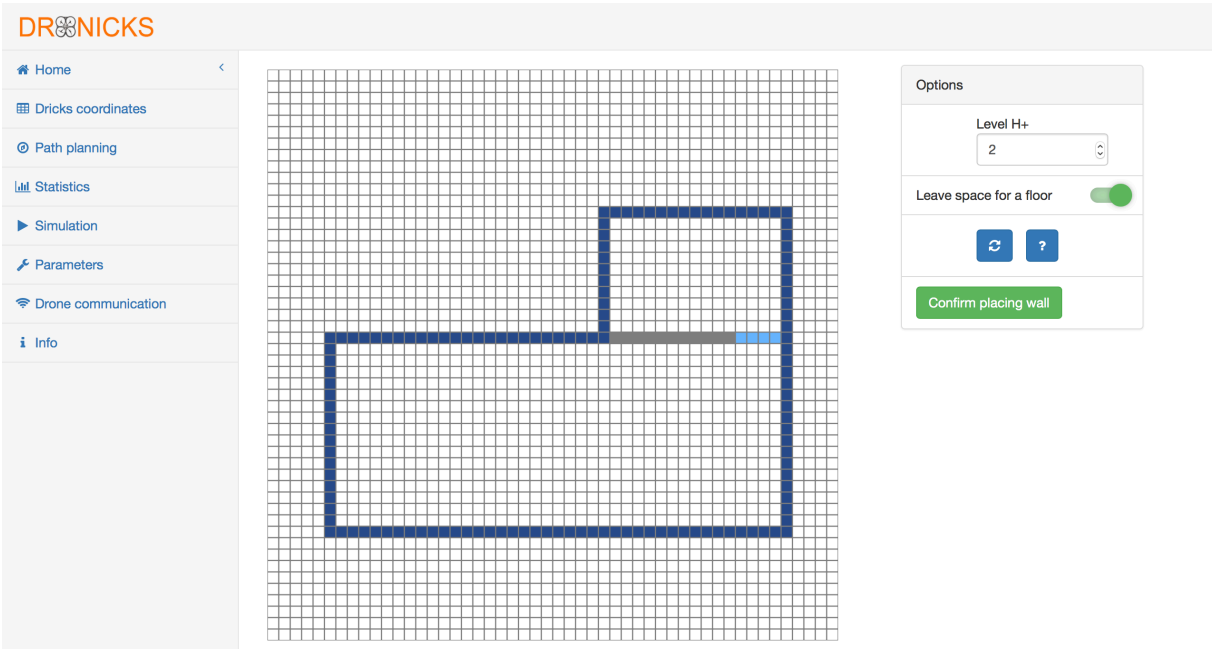


Figure 16 Capture of the 'Placing a wall' option

If the last corner chosen is the same as the first one, a closed loop wall will be created.

Editing a wall

When a wall is hovered with the mouse, it appears red. Clicking this wall will lead to the "Placing an opening" feature. Although if one of the two options "Connection mode" or "Delete wall mode" are checked, the outcome will be different. The delete wall mode is explicit and will delete the selected wall. The "Connection mode" on the other hand will merge two walls or wall sequences together forming a continuous blended wall.

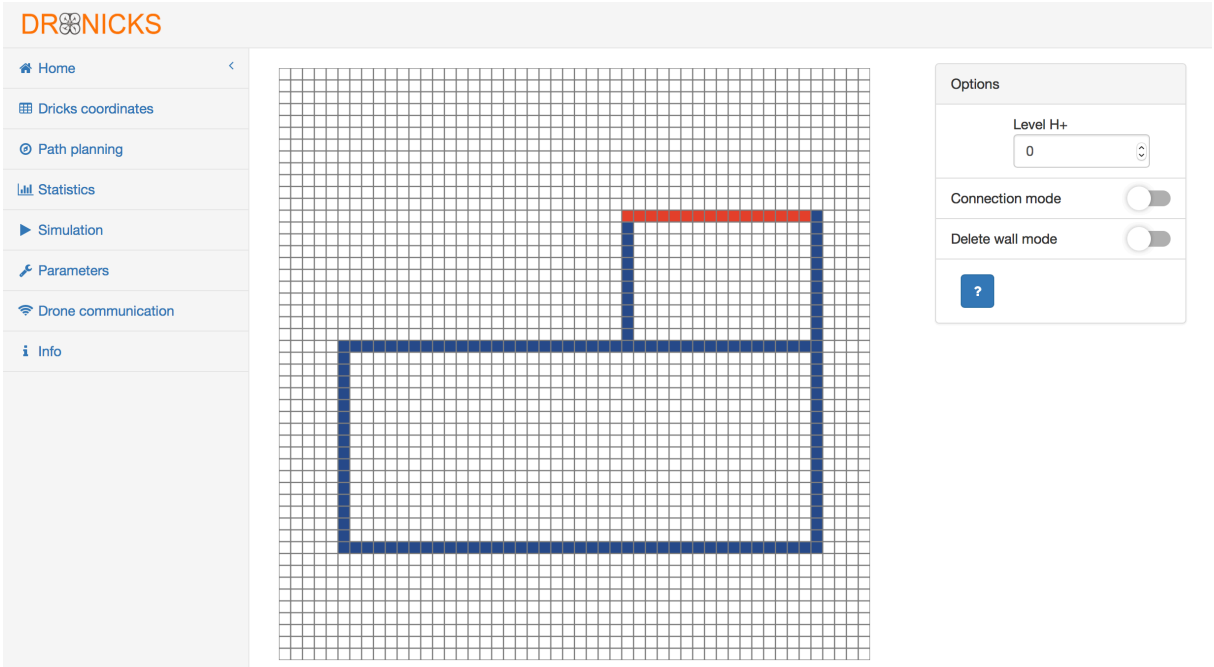


Figure 17 Capture of the 'Editing a wall' option

Placing an opening

The green zone is where the upper left corner of the window might be. When the user clicks on one of the green rectangles he can drag the mouse towards the lower right corner of the window. For sake of simplicity, a window is chosen to be always 4 rectangles wide. So far only dricks of 20 cm by 40 cm have been made. Each rectangle on the following figure represents a length of 20 cm. So a window will have a width of 80 cm. Blue rectangles are already placed openings.

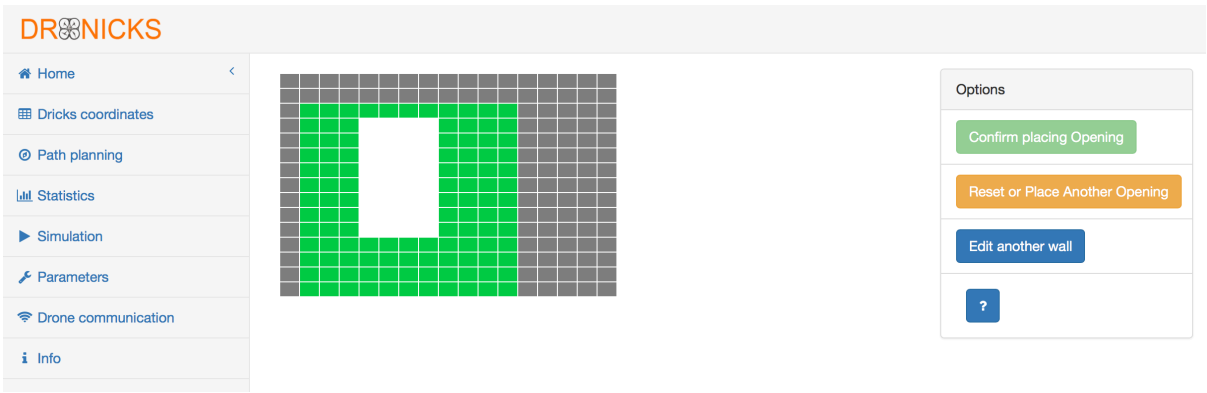


Figure 18 Captures of the 'Placing an opening' feature (a)

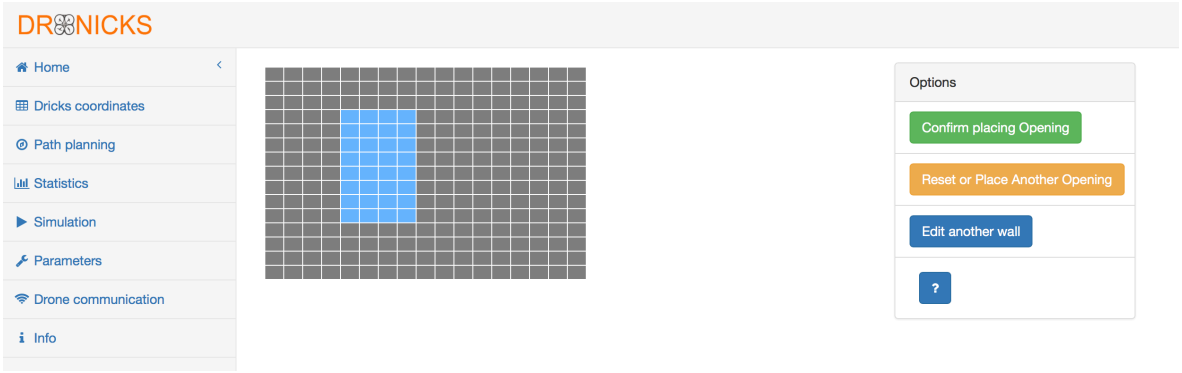


Figure 19 Captures of the 'Placing an opening' feature (b)

Placing a floor

The user can choose the girder length from 1.6 m to 7.6 m with intervals of 0.2 m. The user can also choose the orientation of the floor system. (This topic will be detailed in Chapter 2).

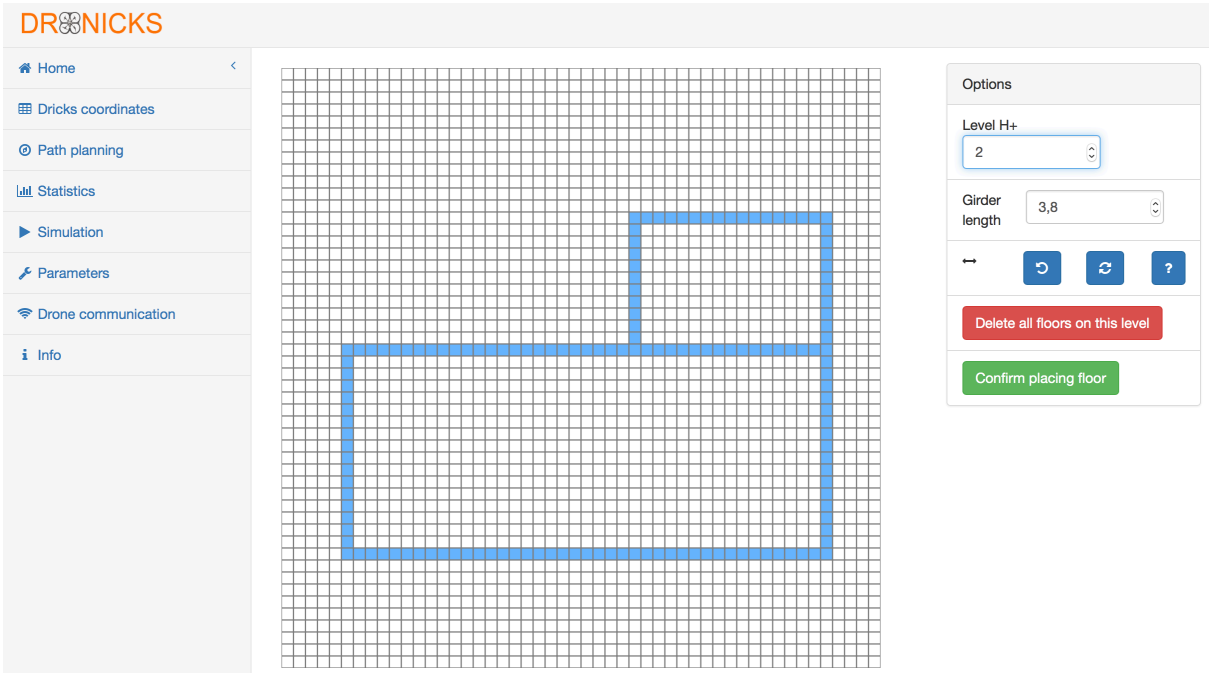


Figure 20 Capture of the 'Placing a floor' feature (a)

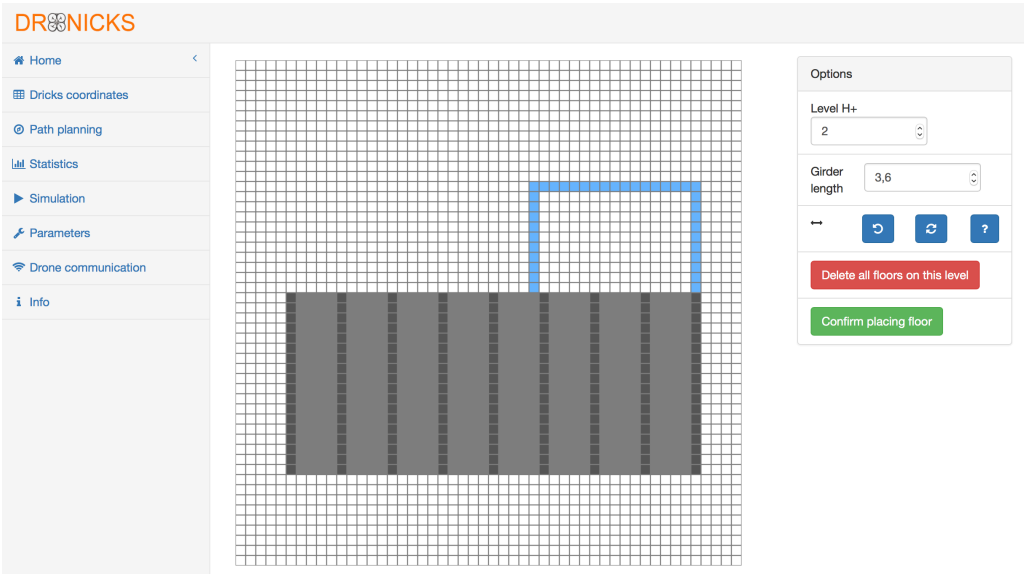


Figure 21 Capture of the 'Placing a floor' feature (b)

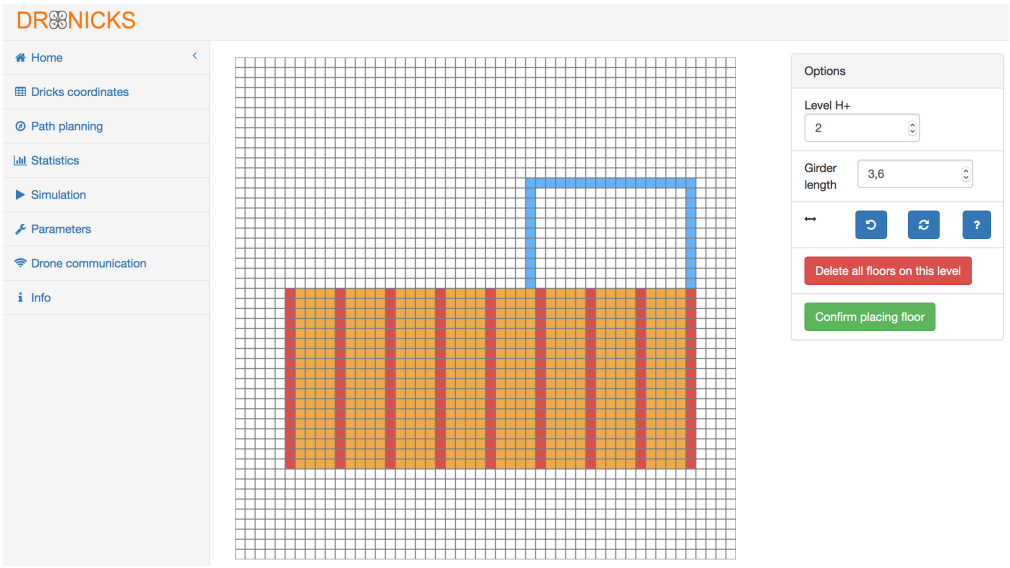


Figure 22 Capture of the 'Placing a floor' feature (c)

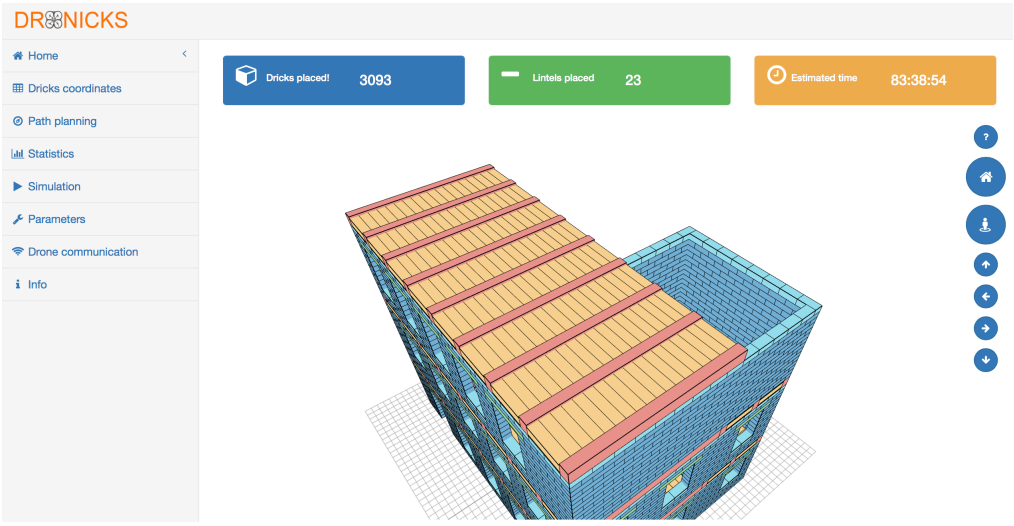
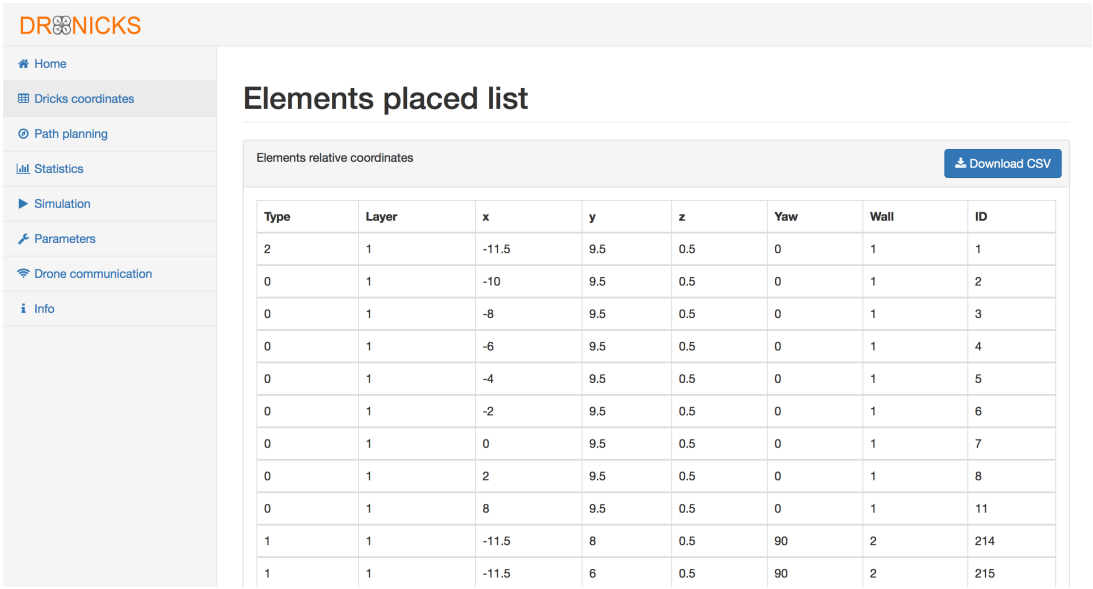


Figure 23 Capture of the result of the 'Placing a floor' feature

Drick coordinates

The *coordinates list* contains all the information needed to build the 3D model on screen. The units are as follows: 1 unit is 1 square width of the grid on the screen.

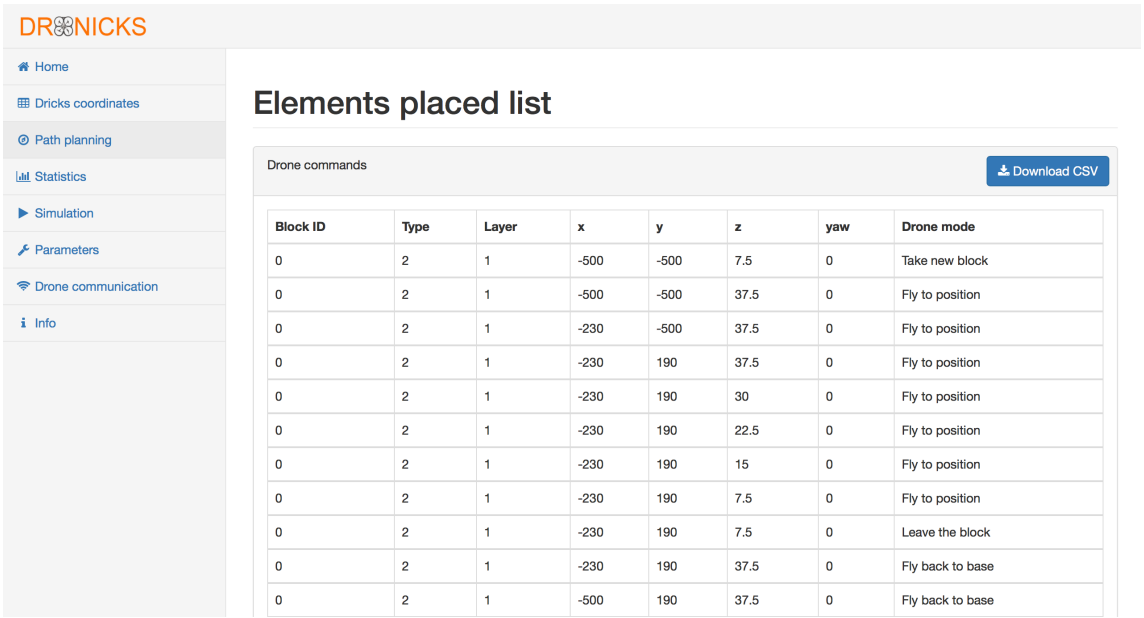


Type	Layer	x	y	z	Yaw	Wall	ID
2	1	-11.5	9.5	0.5	0	1	1
0	1	-10	9.5	0.5	0	1	2
0	1	-8	9.5	0.5	0	1	3
0	1	-6	9.5	0.5	0	1	4
0	1	-4	9.5	0.5	0	1	5
0	1	-2	9.5	0.5	0	1	6
0	1	0	9.5	0.5	0	1	7
0	1	2	9.5	0.5	0	1	8
0	1	8	9.5	0.5	0	1	11
1	1	-11.5	8	0.5	90	2	214
1	1	-11.5	6	0.5	90	2	215

Figure 24 Capture of the 'Dricks coordinates' page

Path planning

The path planning is the list that will be sent after few modifications to the drone. It gives the major set points (here in cm) to which the drone should go and the action that it should undertake.



Block ID	Type	Layer	x	y	z	yaw	Drone mode
0	2	1	-500	-500	7.5	0	Take new block
0	2	1	-500	-500	37.5	0	Fly to position
0	2	1	-230	-500	37.5	0	Fly to position
0	2	1	-230	190	37.5	0	Fly to position
0	2	1	-230	190	30	0	Fly to position
0	2	1	-230	190	22.5	0	Fly to position
0	2	1	-230	190	15	0	Fly to position
0	2	1	-230	190	7.5	0	Fly to position
0	2	1	-230	190	7.5	0	Leave the block
0	2	1	-230	190	37.5	0	Fly back to base
0	2	1	-500	190	37.5	0	Fly back to base

Figure 25 Capture of the 'Path planning' page

Simulation

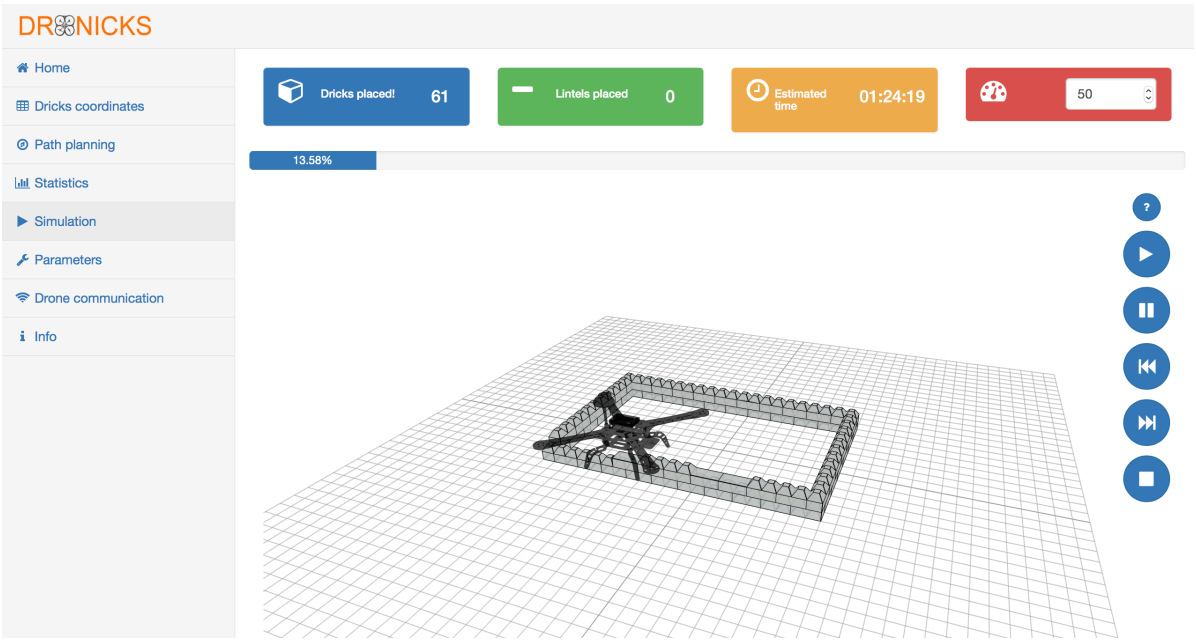


Figure 26 Capture of the 'Simulation' page

On this page the user can visualize the construction process of the designed model.

Parameters

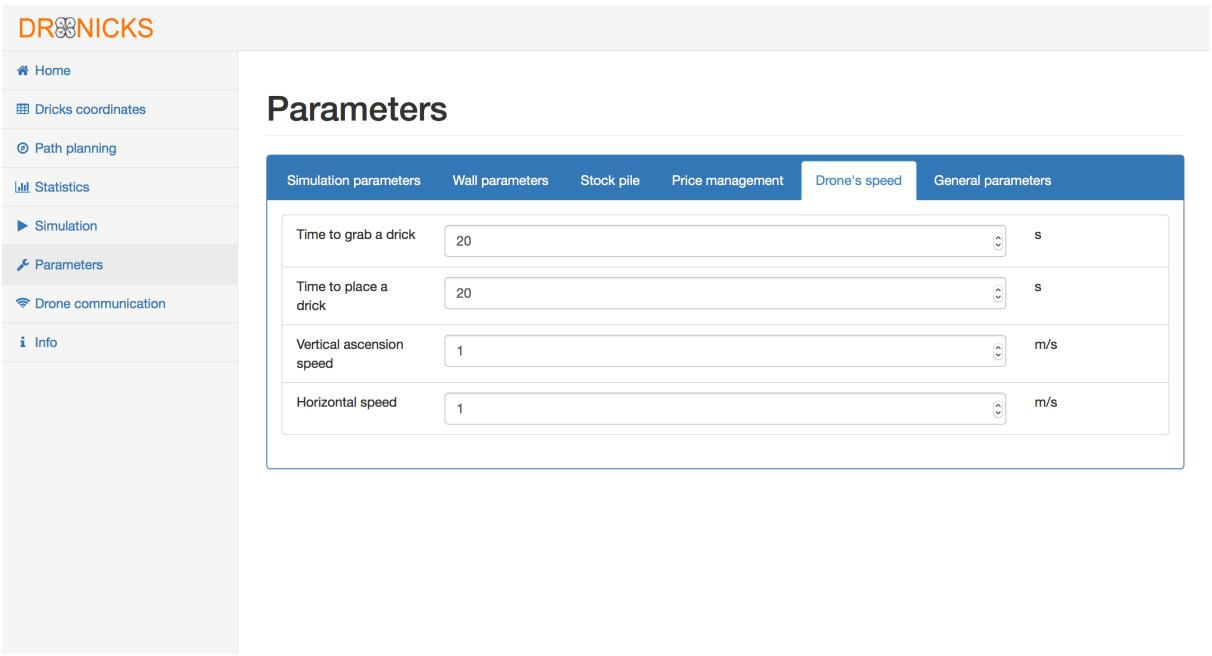


Figure 27 Capture of the 'Parameters' page

On this page, the user can define among other parameters the wall height, the stockpile coordinates [x, y] or even the drone's promptness:

The time needed to grab a drick. [s];
The time needed to place and release a drick. [s];

The drone's vertical speed (upwards and downwards). $\left[\frac{m}{s}\right]$;

The drone's horizontal speed $\left[\frac{m}{s}\right]$;

But also, the cost parameters of the project used in the statistics page such as:

The number of vertical square meters of dricks placed by a drone in one hour. $\left[\frac{m^2}{h}\right]$

The number of vertical square meters of dricks placed by a mason in one hour. $\left[\frac{m^2}{h}\right]$

The price per hour for a drone. [€/h]

The price per hour for a mason. [€/h]

Statistics

On this page, the user can follow up the *order list* of elements needed to build the model in real life. The user can compare rough estimations of block placement time and labor cost for the project. He can also follow up the square meters placed since the start date and time of the project. These parameters can be entered in the parameters page. This model is only valid for a site working on the clock as no break times are introduced into it.

Time for placement = $Total_{m^2} \; / \; \frac{m^2_{mason}}{h}$

or $Total_{m^2} \; / \; \frac{m^2_{drone}}{h}$

where

$Total_{m^2}$ is the total number of vertical square meters of dricks in the project;

$Total_{m^2} = Total\ number\ of\ dricks * drick\ lateral\ surface;$

$\frac{m^2_{mason}}{h}$ is the number of vertical square meters placed in one hour by a mason;

$\frac{m^2_{drone}}{h}$ is the number of vertical square meters placed in one hour by a drone.

Placement labor price = $Time\ for\ placement_{mason} * \frac{Price_{mason}}{h}$

or $Time\ for\ placement_{drone} * \frac{Price_{drone}}{h}$

The blue bar represents the placement labor time (respectively the placement cost) for a mason and the grey bar the placement labor time (respectively the placement cost) for a drone. The green bar (see figure 28) also represents the placement labor time (respectively the placement cost) for a drone but is computed in the same way as the estimated project time presented on the home page.

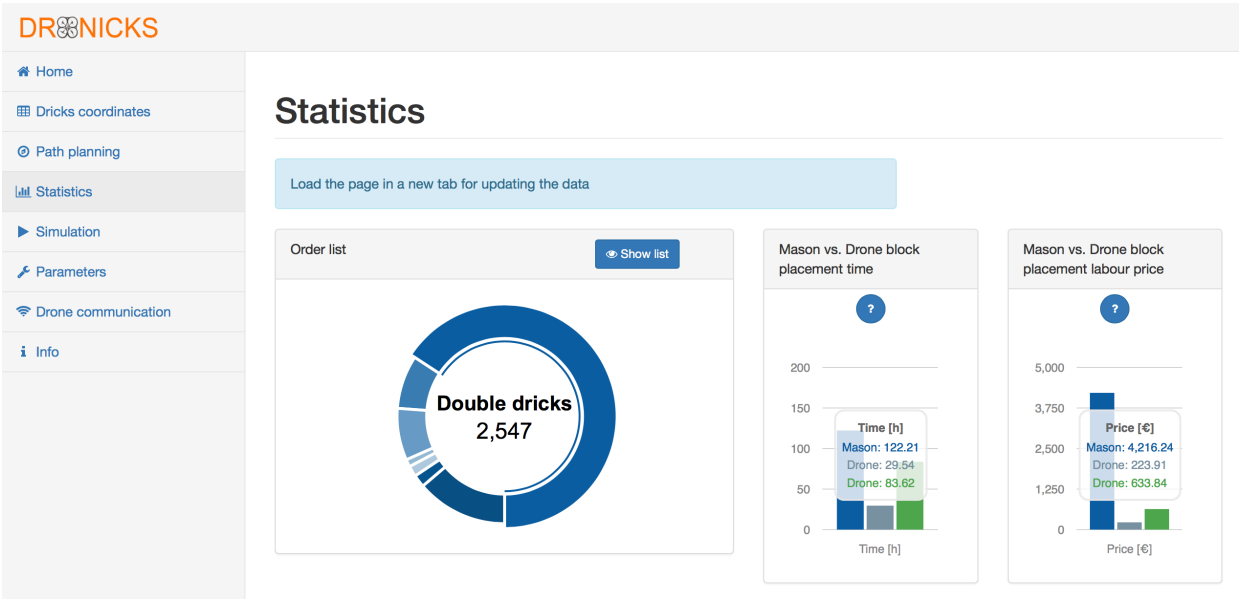
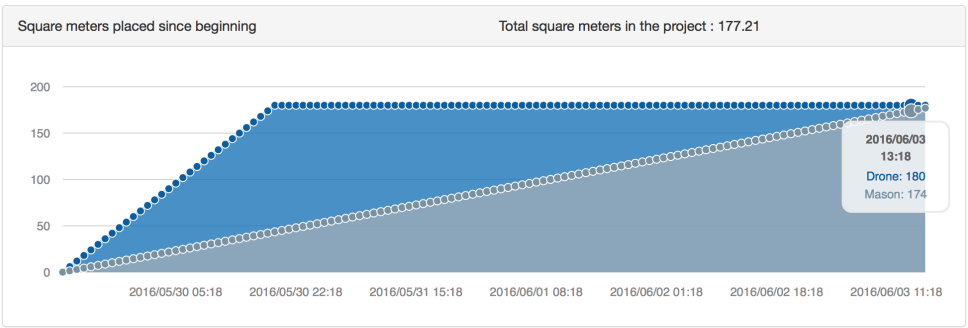


Figure 28 Capture of the 'Statistics' page (a)



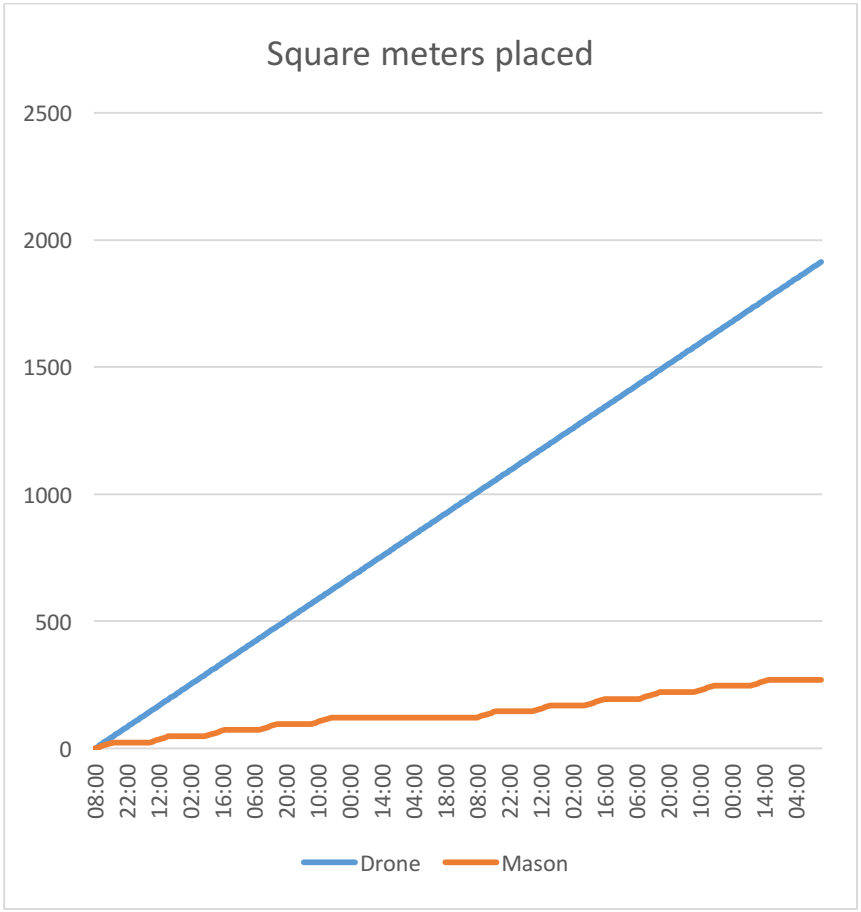
Graph 1 Capture of the 'Statistics' page (b)

In the "Square meters placed since beginning" panel, the total vertical square meters of the project are shown as well as a graph showing the evolution of the square meters placed by a drone and mason according to the elapsed time since the beginning of the project.

In this graph, masons are supposed to take turns so that the site works on the clock. If this wasn't true, the pace of the graph would be stepwise showing that a mason, a human being, only works 8 hours a day and 5 days a week on average. The following graph presents such a scenario and the assumptions made are:

Two drones work together continuously taking turns for recharging. They have a placement rate of 6 vertical square meters per hour.

Two masons work together but have days of 8 hours, 5 days per week. They have each a placement rate of 1.45 vertical square meters per hour.



Graph 2 Squares meters placed according to time (drone vs. mason)

Drone communication

This web page allows the user to create a communication channel between the web application and the drone, hence to send it its mission. The communication is done by connecting the '*Dronicks*' program and the drone on the same IP address within the same network.

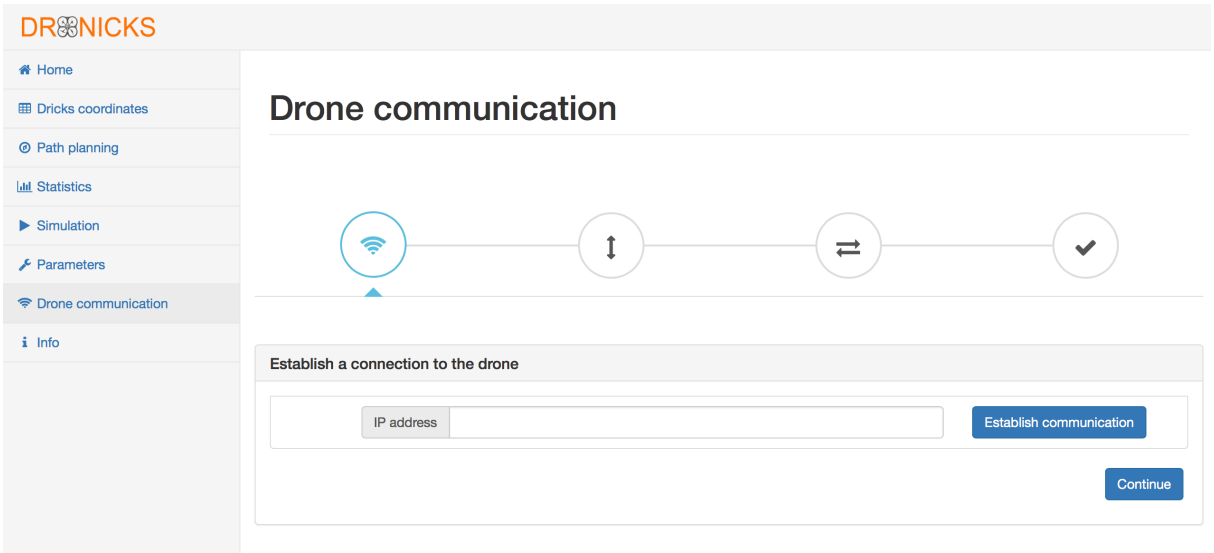


Figure 29 Capture of the 'Drone communication' page

After the communication is established, a ROS object is created with the url of the communication channel.

ROS is a system that allows the control of robotic components from a computer. A ROS system is composed of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model. For example, a drone might be implemented as a node, which publishes sensor data in a stream of messages. These messages can be read by any other node. A computer can for example read the data from a drone and send it feedback. [20]

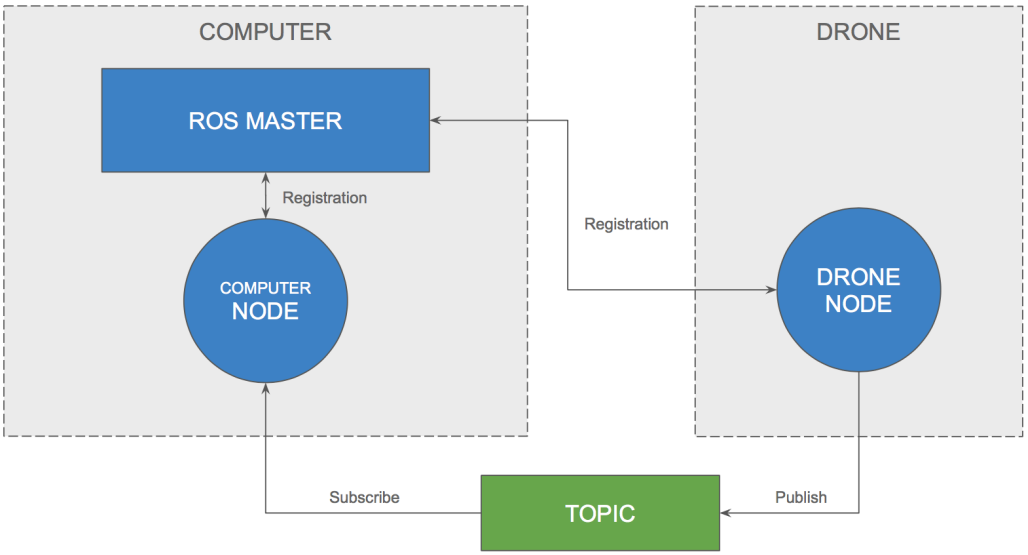


Figure 30 ROS node to node communication illustration

The computer is now a ROS node and can communicate with the drone which is itself a ROS node. The computer then starts listening to a specific TOPIC: it subscribes to it. Afterwards, the mission is prepared for the drone. A mission is a list of tasks that the drone will execute. It contains Cartesian coordinates and an action flag. The task is then read by the drone. For example: Take off and go to position (0,0,1). Finally, once all those tasks are assembled, the mission is published on the TOPIC. The drone can now read and execute its mission.

Info

This web page provides info about the web browser compatibilities and the used libraries and authors of those libraries.

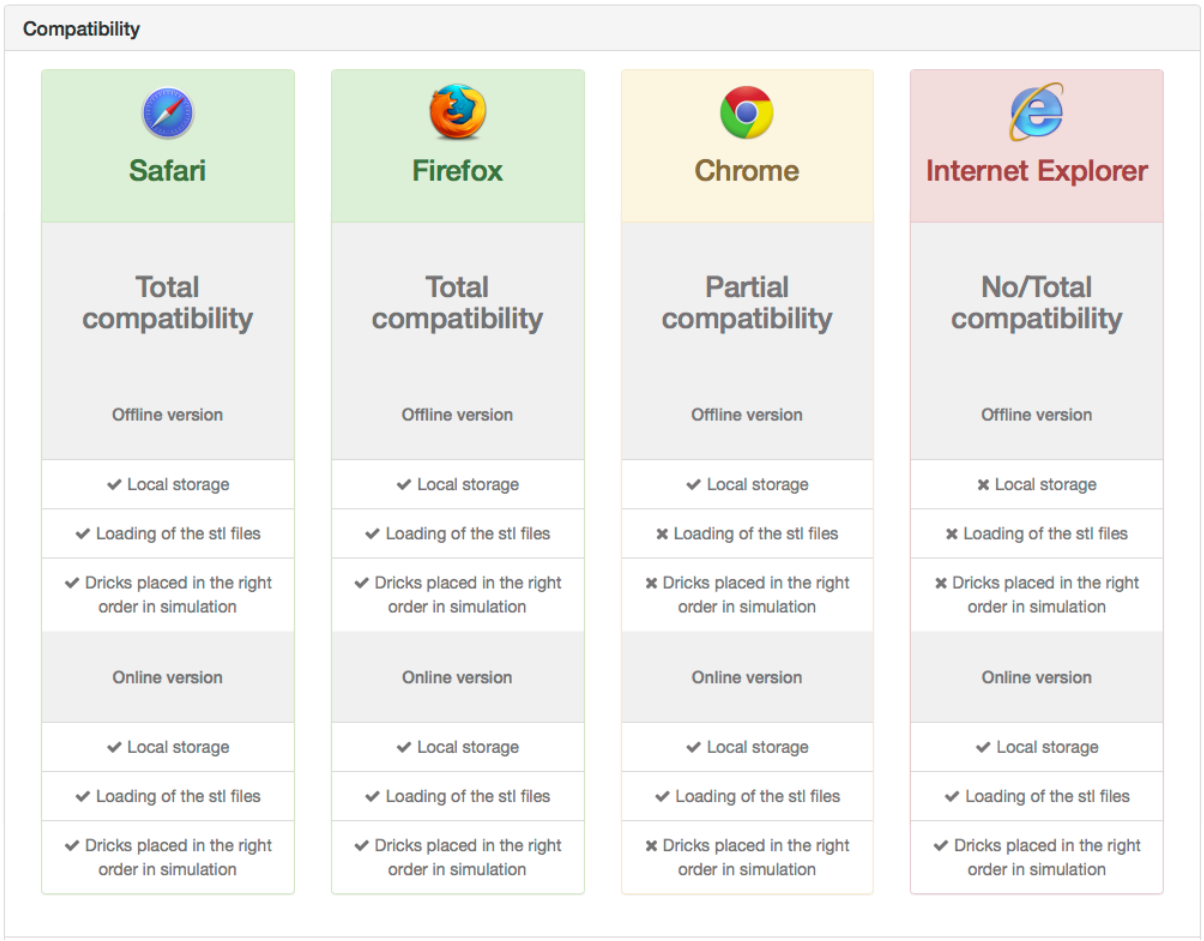


Figure 31 Browsers compatibility

The program's structure

The program uses four type of files:

- JavaScript (.js)
- Cascading Style Sheet (.css)
- Hypertext Markup Language (.html)
- Stereolythography (.stl)

The main dependencies between the files are depicted on the following figure:

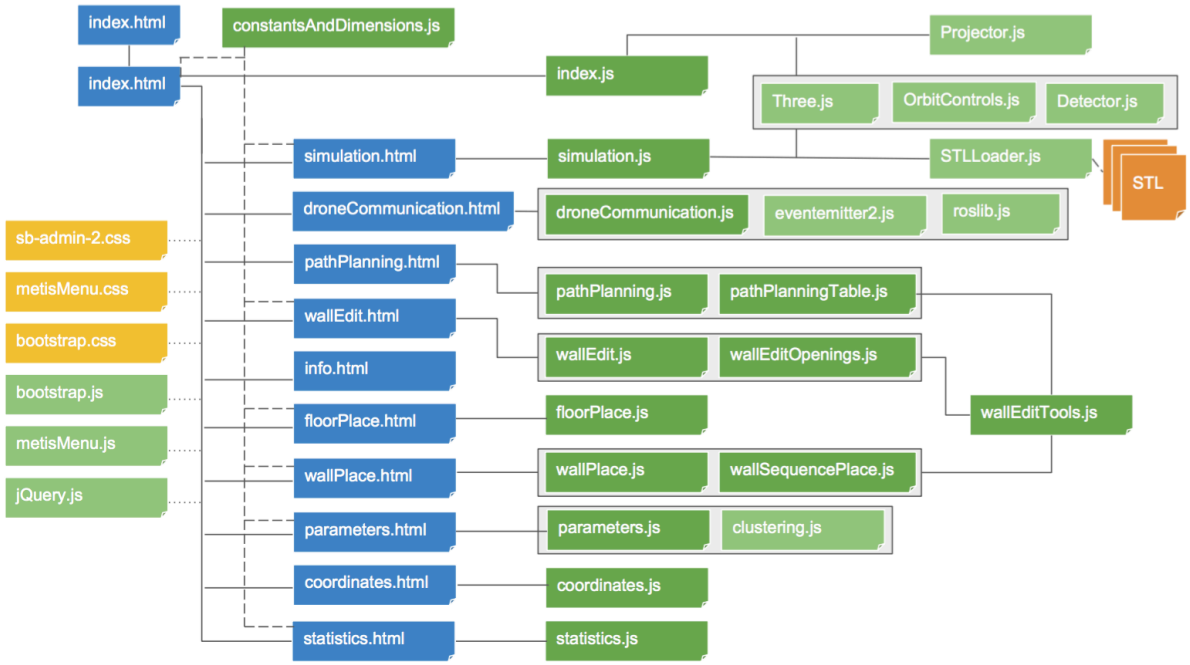


Figure 32 Dronicks main dependencies

As the html file defines the structure of the web page or interface, the CSS file defines the look and feel. The JS files are the core files of the program and the STL files are the 3D models used for the simulation.

The blue and dark green files on the above figure are made for the purpose of this thesis. The other files are existing open-source APIs.

Using the cache as memory

Every time a block is placed or removed, the list containing all the blocks, further called the *dricks list*, is updated and saved into the browser's cache memory. As long as the browser's cache is not erased, the user can exit the page and open it later on in the same state. The browser's cache is erased at the same time as the browser's history is cleared. Storing the information into the cache also allows to save data when navigating between different web pages inside the web application. Indeed, every web page using different code files needs to retrieve the data previously stored in the cache to act as if the program is one unique web page.

Conclusion

The proposed solution is simple in its file structure and above all the resulting '*Dronicks*' program is very intuitive to use. The graphical user interface applies the latest new tendencies of the web and is therefore more attractive to the user. What made Apple machines so popular is the idea of Steve Jobs wanting the user interface to inspire people a "Hey! Don't be afraid it's easy" rather than a "This is for nerds" [13]. This is exactly what this program is about, being simple and intuitive. Above all it falls under the era of mobile revolution as it is possible to use it on a tablet and in a wide range of web browsers.

From scratch to drone instructions

After every modification, the list containing all the placed dricks (the *dricks list*) is saved in the cache memory of the user's browser. A modification can be the placement of a wall, the removal of a single drick, the placement of an opening, the placement of a floor etc. When the user decides that his model is acceptable, he can visualize a simulation that mimics every movement of the drone and from there appreciate the erecting structure step by step. A *path planning list* is used to store the information relative to the drone's movements and actions. This list is directly based upon the *dricks list* but contains eleven instruction lines per drick instead of one. Whereas the *dricks list* contains only the final dricks positions, the *path planning list* contains all the intermediate positions for the drone from the stockpile where it first grabs a drick until it comes back at the stockpile for grabbing a new drick. When the user estimates the project is valid, he can establish communication with the drone and send the *path planning list* to the drone via a ROS topic. The drone can now read the mission and proceed.

Other approach of the problem

The approach used throughout this thesis is based on a list of dricks that is created at the beginning of the design process and updated after every alteration. Another way to approach the problem was suggested in the thesis written by Jean-Sebastien Breton and Justin Leplat under the supervision of Professor Latteur. [14]
It consists of designing the whole project in a classical CAD program and, once the project is finished, to voxelize[8] the forms and translate these voxelized 3D meshes into dricks. As it has already been done, this approach has been left aside but could been reconsidered though as the forms of the dricks have considerably evolved. Although, it would be way more complex than what was done before as the dricks now have a right-angled parallelepiped shape instead of a cubical-like one.

[8] Voxilization is the action of dividing a form into 3 dimensionnal pixels.

CHAPTER 2

DEFINITION OF DRONE-COMPATIBLE BUILDING PROCESSES FOR BASIC STRUCTURES

Algorithms for brick laying processes

"*Designing frequently involves establishing visual, systemic, or geometric relationships between the parts of a design. More times than not, these relationships are developed by workflows that gets us from concept to result by way of rules. Perhaps without knowing it, we are working algorithmically - defining a step-by-step set of actions that follow a basic logic of input, processing, and output. Programming allows us to continue to work this way but by formalizing our algorithms*." (The Dynamo Primer) [21]

Placing a simple wall

Placing a simple wall, in other words a straight wall with two extremities is just a particular case of the *"sequenced walls algorithm"* where only two corners are provided as input. This algorithm is described in the next sub-chapter.

Sequenced walls

A sequence of walls can be as simple as the walls (a) in the following figure or as complex as the walls (b). In either case, the algorithm for placing those walls is the same.

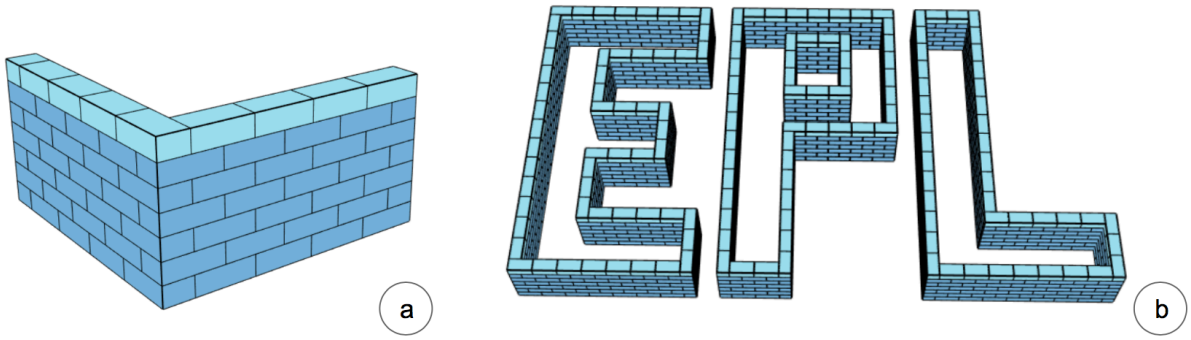


Figure 33 Simple and complex sequenced walls

First the user enters the coordinates of the corners of the sequence of walls by pointing those out on a 2D grid. Those [x, y] coordinates are pushed two by two in an array "*corners*".

Pre-processing of the coordinates

These corners are then preprocessed in a routine that attributes a "*priority*" to each wall extremity. Each wall extremity will either be "*principal*" or "*not principal*". Being "*principal*" means either to begin or to end the wall with a double drick. To determine this characteristic, the first layer composing the sequence is analyzed. Let's take the following simple sequence as example.

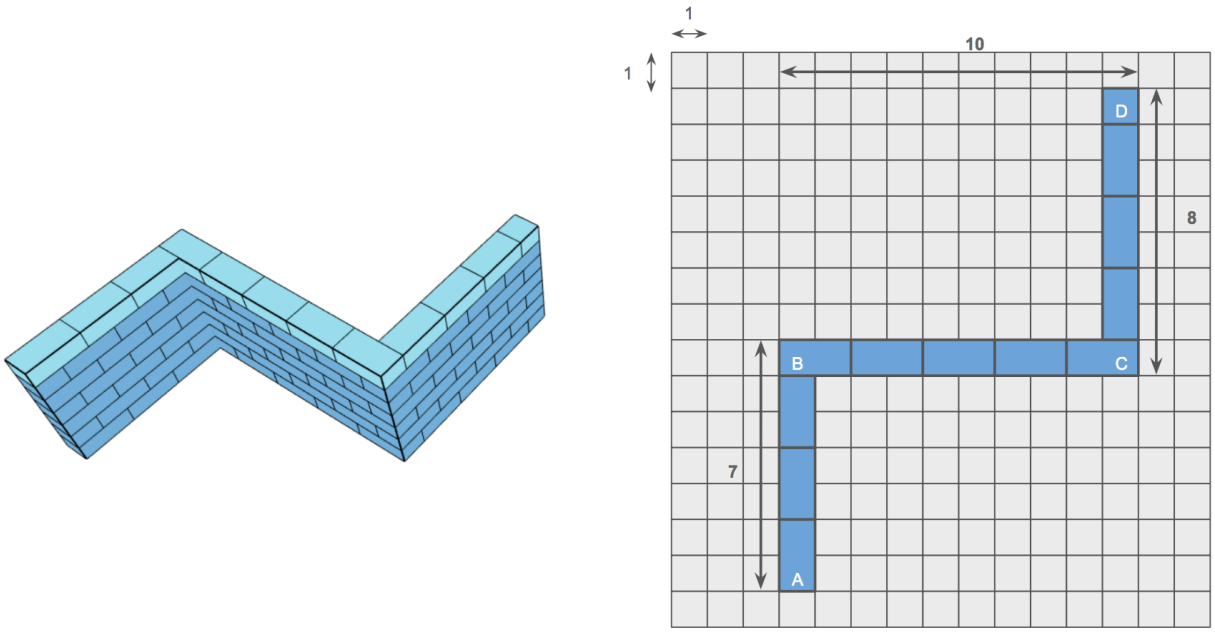


Figure 34 Wall sequence taken as example for illustrating the sequenced walls algorithm (3D left and 2D right)

STEP 0:
Assumption is made that the sequence's start point is A and the endpoint is D and that the first drick placed is always a double one. Indeed, if the sequence can be drawn with only double dricks then it would be a waste of time to use mono dricks because it would induce more placement journeys and a higher production cost. The first wall extremity is always set to be a principal one, meaning that it starts with a double drick.

STEP 1:
At the second extremity of the first wall (B) the following question should be asked to know if it is a principal one. "Can the wall A-B be filled with only double dricks?" or equivalently "Will the second extremity of the first layer of this wall contain the corner drick?". In this case the wall A-B is 7 units long which is uneven so the extremity B is not principal.

STEP 2:
The following analyzed extremity is the first extremity of the second wall B-C (also at B). For a new wall, the first extremity has the opposite priority (principal/not principal) of the second extremity of the previous wall. The first extremity of wall B-C is thus a principal one.

Then the cycle starts over.

STEP 1 (bis):
Then at the second extremity of the second wall (at C) STEP 1 is repeated: there is enough space to place exact 5 double dricks, hence it is a principal one.

STEP 2 (bis):
The second extremity of the second wall being a principal one, the first extremity of the third wall is not a principal one.

Repeat for as much walls the sequence includes.

The sequenced walls algorithm

Determine whether the sequence forms a closed loop. (The case of a non-closed loop will be explained first). Note: 1 unit represents 1 square width on the previous figure.

For each corner except the last but one:

Get the wall formed by this corner and the next corner.

For each layer:

Determine coordinates of both extremities
Determine the wall's yaw

If it's the first corner and the layer is uneven **then** place a mono drick.

Determine the potential offsets to give to the dricks of the current layer based on the wall's extremities priorities and layer:

If the layer is uneven **then**

If the wall's first extremity is not principal, **then**

A start offset will shift every drick of one unit towards the second extremity. (see following figure (a))

else

No start offset. (see following figure (b))

If the wall's second extremity is not principal, **then**

An ending offset will stop the placement of the double dricks one unit before the second extremity. (see following figure (c))

else

No ending offset. (see following figure (d))

else (the layer is even):

If the wall's first extremity is principal, **then**

A start offset will shift every drick of one unit towards the second extremity. (see following figure (e))

else

No start offset. (see following figure (f))

If the wall's second extremity is principal, **then**

An ending offset will stop the placement of the double dricks one unit before the second extremity. (see following figure (g))

else

No ending offset. (see following figure (h))

Now the needed offsets are determined, place double dricks from first extremity (+ possible offset) until the placement of a double drick would exceed the second extremity (- possible offset).

If it's the last corner, **then**

If the corner is not principal, **then**

If the layer is uneven **then**

Place a mono drick.

else

If the layer is even, **then**

Place a mono drick.

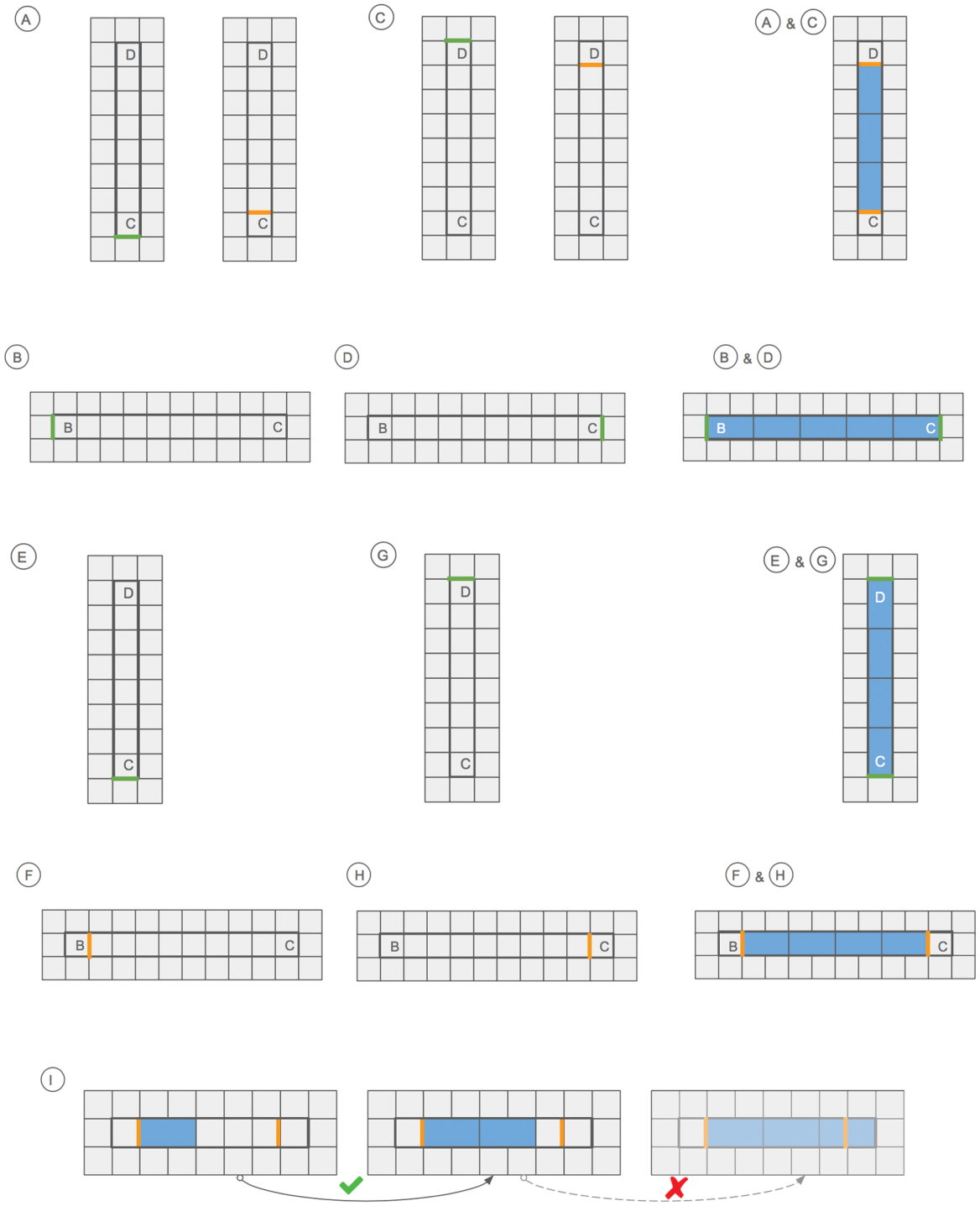


Figure 35 Sequenced walls algorithm illustrations

The algorithm illustrated for the example below is shown in the following figures from another perspective.

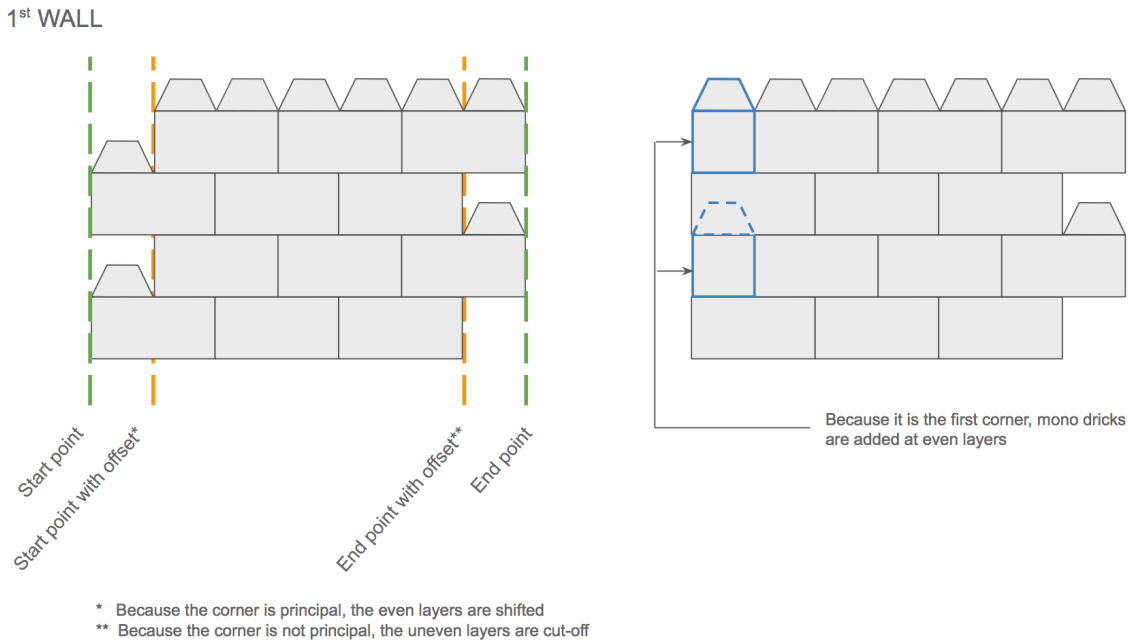


Figure 36 1st wall construction by the sequenced walls algorithm

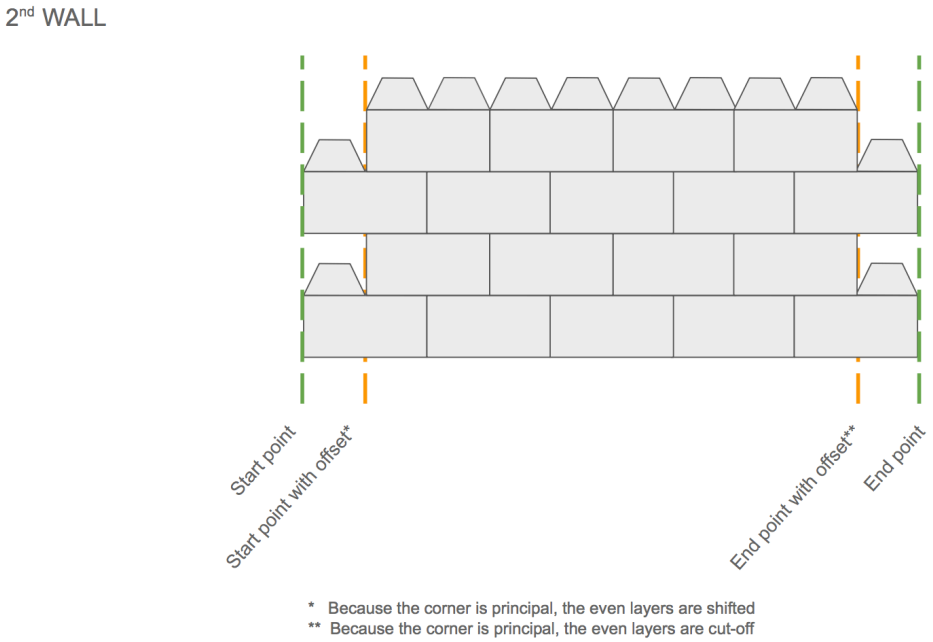


Figure 37 2nd wall construction by the sequenced walls algorithm

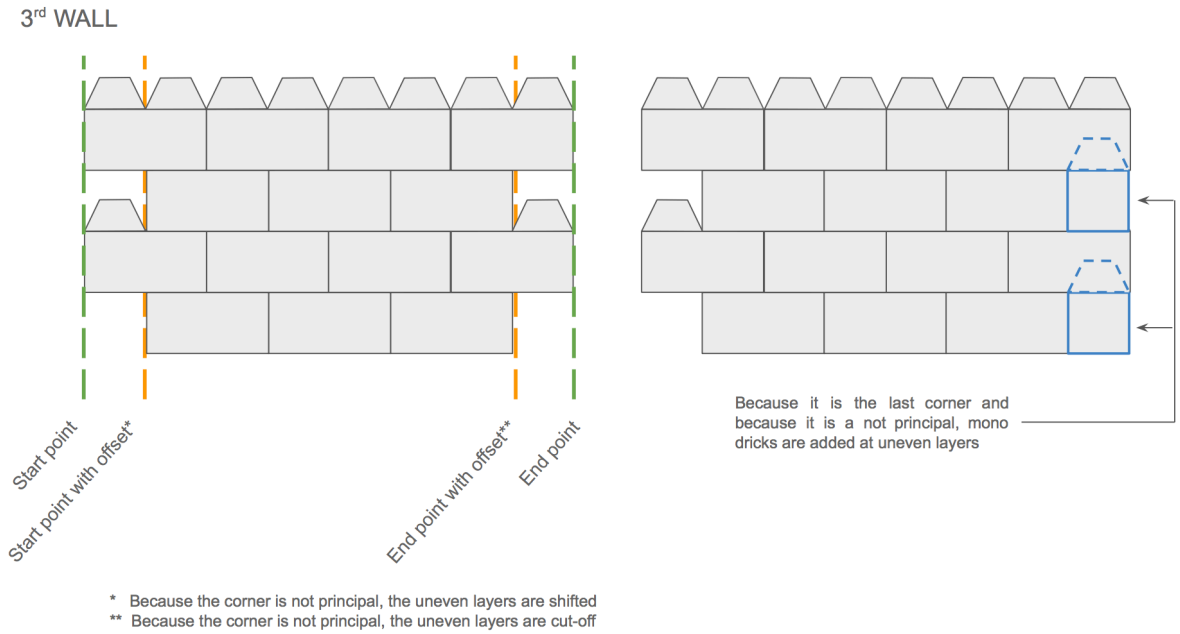


Figure 38 3rd wall construction by the sequenced walls algorithm

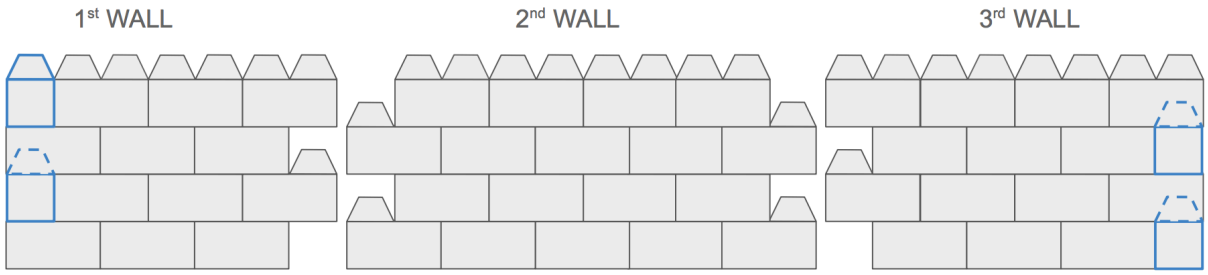


Figure 39 Straighten pattern of walls built by the sequenced walls algorithm

The algorithm for the case where the walls create a closed loop is the same except that every first drick of an uneven layer is skipped and becomes the last drick placed for that layer. The reason for this skipping is explained in the *"Best drick placement method"* sub-chapter of the next chapter.

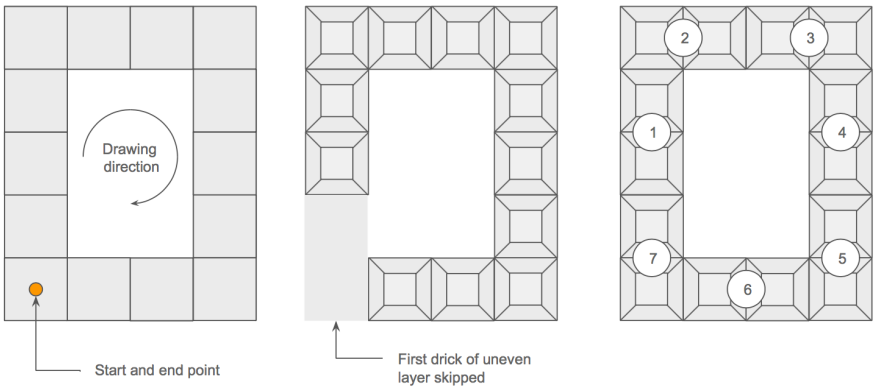


Figure 40 Closed loop case of the sequenced walls algorithm

Keeping the sequences in memory

For every placed wall or sequence of walls, 3 arrays are updated.

The first one contains the parameters of each wall. They are stored one after the other in a list without being grouped by sequence.

The second one contains arrays of wall ids representing the sequences:

> [[1,2,3],
> [4,5]]
> represents the situation where two sequences exist in the scene. One sequence being composed of the wall with id = 1, the wall with id = 2 and the wall with id =3. The other sequence is composed of the walls with ids 4 and 5.

The third one is an array containing the corner coordinates of every sequence:

> $[\ [\ [x_0, y_0]\ ,\ [x_1, y_1]\ ,\ [x_2, y_2]\ ,\ [x_3, y_3]\ ]\ ,$
> $[\ [x_0, y_0]\ ,\ [x_1, y_1]\ ,\ [x_2, y_2]\ \ \ \ \ \ \ \ ]\ ]$
>
> represents the coordinates of the corners of the first and second sequence. The first sequence being composed of 3 walls, will have 4 corners. The second one being composed of 2 walls, will have 3 corners.

If a single wall is placed, it will create a new entry in each of the 3 previously cited arrays. For the two last ones, it will seem as if the wall is a sequence on its own.

Merging two straight walls

When placing walls, the user can choose between placing just one straight wall or a sequence of merged walls. In the event that the user would have drawn two separate walls and that each wall is having one of its extremities "touching" an extremity of the other wall, then it might be interesting to request afterwards that those walls are merged. Walls in such a configuration will be called "neighbor walls" in the rest of this chapter.

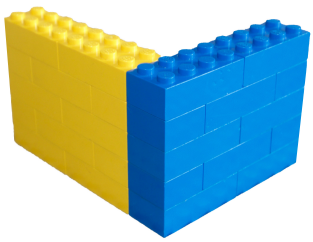


Figure 41 Non merged neighbor walls

How to proceed to merge such walls? There are two solutions.

The first solution is to replace one corner drick per wall every two layers in order to have common dricks for the two walls. This technique is efficient and fast.

The second solution is to redraw the walls in a sequenced fashion by using the corner coordinates of the corners. This technique requires to recompute every wall.

Corner drick replacement solution

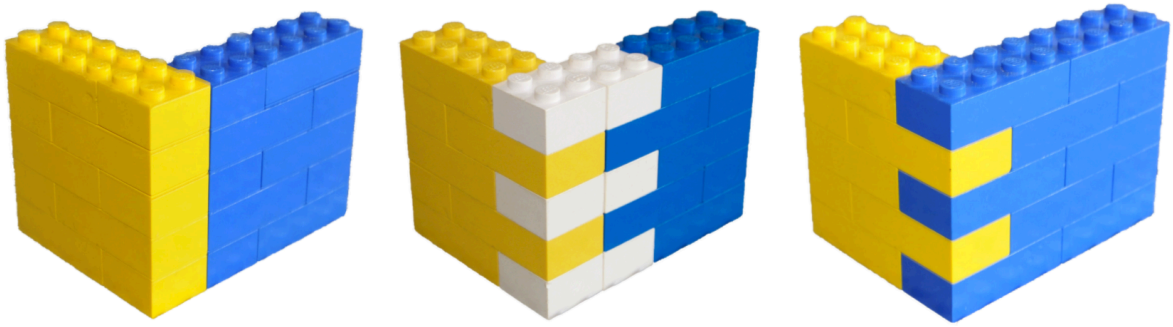


Figure 42 Replacement of corner dricks to merge two walls

2 possible cases, 1 impossible case

Although this technique is simple, it has its limitations. Three possible scenarios could occur. Unfortunately, one of them is impossible to solve with this technique. Let's have a look at the first layer of each scenario.

Scenario 1: Two mono dricks are touching each other

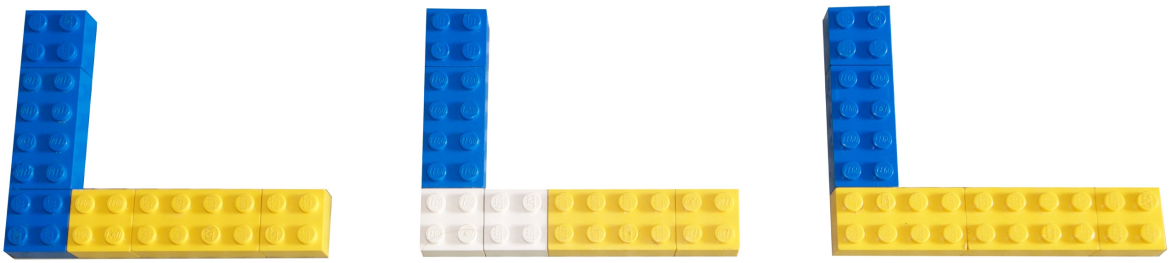


Figure 43 Corner replacement scenario 1

This is the simplest case. The two mono dricks (in white) are deleted and a double drick is added to the wall having the same yaw as the double drick that would result of pasting the two mono dricks together.
Drick replacement will only occur on the uneven layers as shown on the figure 42.

Scenario 2: Two double dricks are touching each other

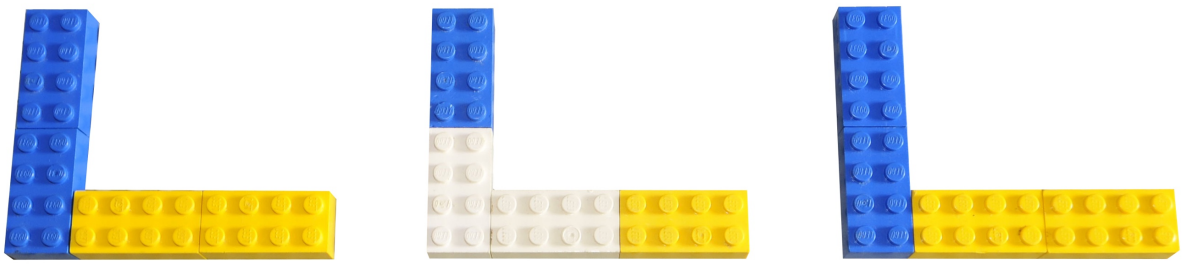


Figure 44 Corner replacement scenario 2

This scenario can be seen as the alternative to the first scenario. The first layer of scenario 2 is the same as the second layer of the first scenario and vice versa. Thus instead of changing the first layer, the second layer and all other even layers are changed and the first layer stays unchanged.

Scenario 3: One double drick is touching a mono drick

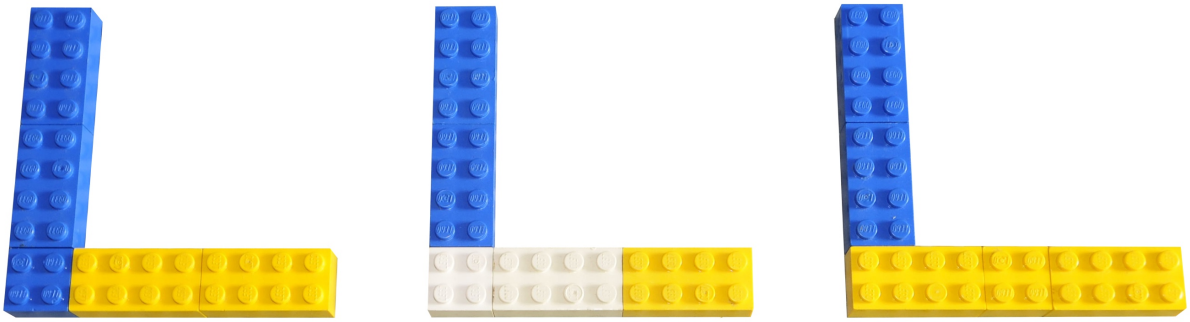


Figure 45 Corner replacement scenario 3

In this case, applying the transformation as on the above figure, will result in an impossible outcome as the dricks placed on the second layer will create a continuous vertical joint. The yellow wall will thus be composed of two unlinked sub parts.

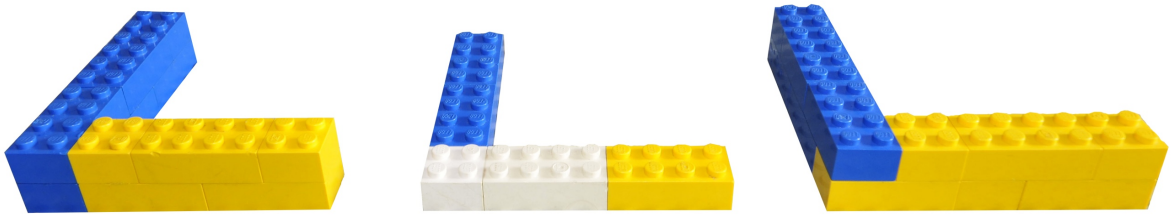


Figure 46 Corner replacement scenario 3 (2 layers)

This joint will be continuous from the first layer until the last layer.

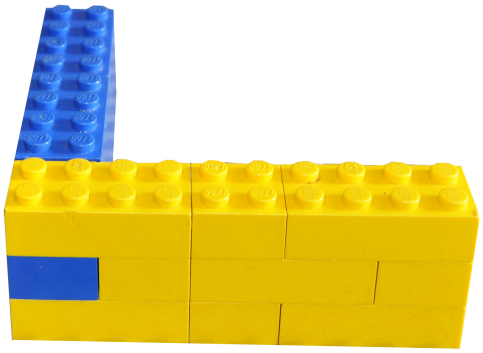


Figure 47 Corner replacement scenario 3 (3 layers)

Sequenced walls solution

The previous method being not 100% reliable, the second method consisting of redrawing the walls in a sequenced fashion by using the corner coordinates is used in the *'Dronicks'* program. This technique requires to recompute every wall but has the benefit of being achievable in every of the 3 previous analyzed scenarios. Although every previously placed opening has to be redrawn after two sequences are merges. This is not yet implemented in the *'Dronicks'* program.

Merging two walls

As every placed wall is either a sequence on its own or belongs to a sequence of multiple walls, merging two walls is just a particular case of the *"merging two sequences of walls algorithm"* and is explained in the next sub-chapter.

Merging two sequences of walls algorithm

The first step for merging two walls is to define their common corner coordinate in the 2D grid.

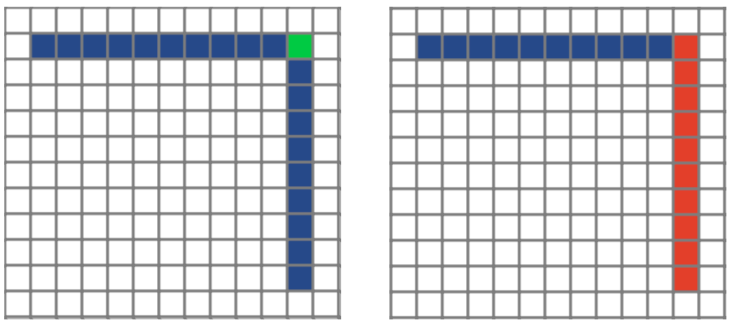


Figure 48 Merging two walls in "Edit mode" of the 'Dronicks' program

Afterwards, for the two walls touching (or containing) the corner tile[9] (in green in the above figure), the sequences associated with each wall are retrieved. Let's call "WALL1" the wall on which the second wall will be attached, in other words, the wall containing the corner coordinate (in red in the

[9] A tile is one square of the grid. One square of the grid has its side lengths equal to the mono drick side lengths.

above figure), and let's call the other wall "WALL2". Let's call the sequence WALL1 belongs to "S1" and the sequence WALL2 belongs to "S2". Let's call "C1" and "C2" the coordinates of the corners representing respectively the first and second sequence of walls.

If the two sequences are the same **then**

the wall forms an open loop and the coordinates of the beginning of the sequence are added to the end of the sequence. Doing so, the *"sequenced walls"* algorithm will form a closed loop pattern of walls.

else

4 cases are distinguished:

if CASE A: WALL1 is the first wall of S1 **then**

CASE A.1: WALL2 is the first wall of S2 **then**

All coordinates except the first corner of C2 are added at the front of C1 in reverse order.

CASE A.2: WALL2 is the last wall of S2 **then**

All coordinates except the last corner of C2 are added at the front of C1 in same order.

else CASE B: WALL1 is the last wall of S1 **then**

CASE B.1: WALL2 is the first wall of S2 **then**

All coordinates except the first corner of C2 are added at the end of C1 in same order.

CASE B.2: WALL2 is the last wall of S2 **then**

All coordinates except the last corner of C2 are added at the end of C1 in reverse order.

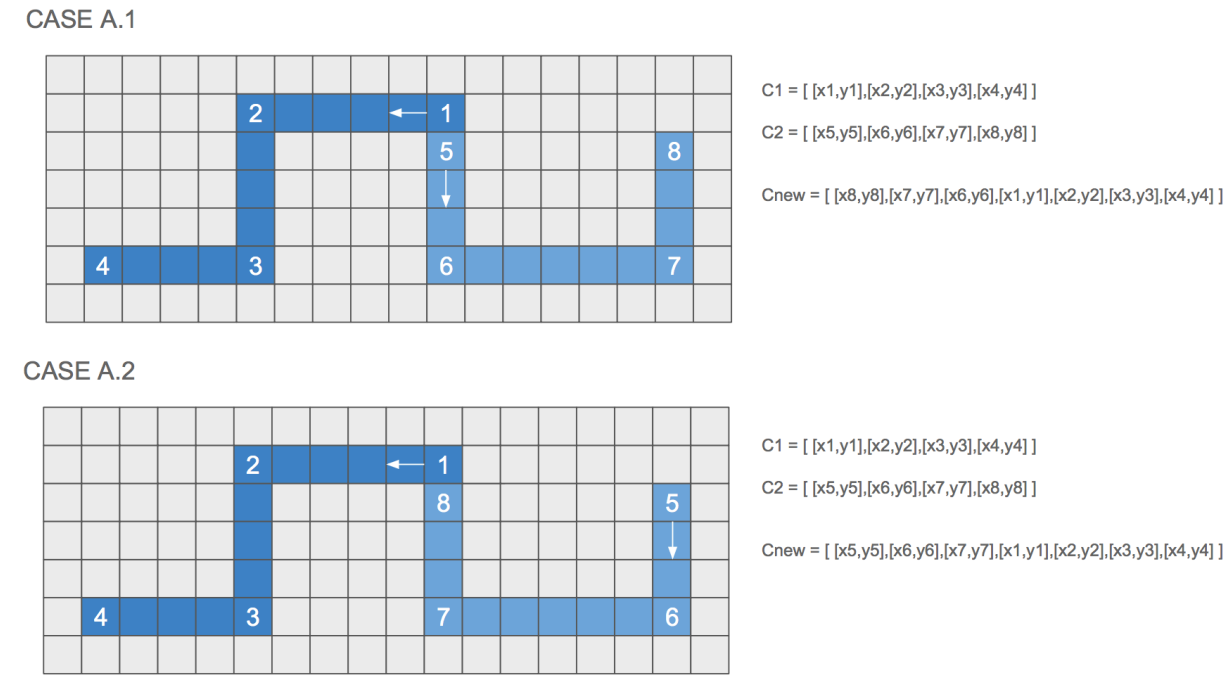


Figure 49 Illustration of the case A in the merging wall algorithm

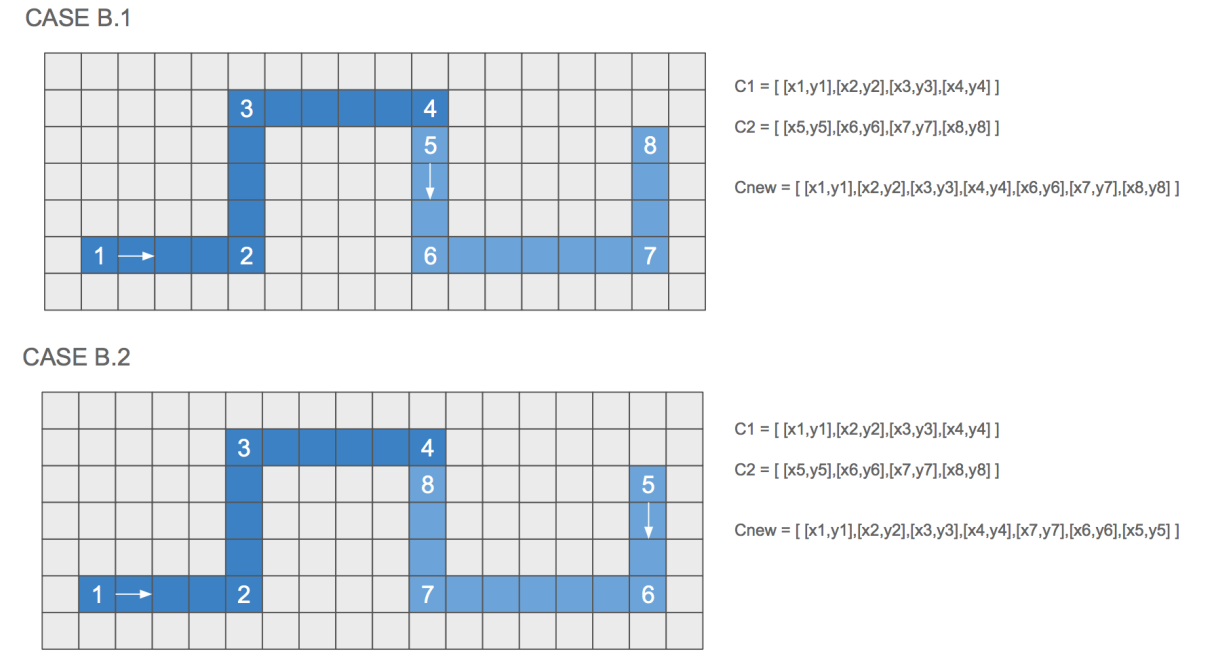


Figure 50 Illustration of the case B in the merging wall algorithm

Perpendicular non-load-bearing walls merging

Traditionally a non-load-bearing wall perpendicular to a load-bearing wall, is juxtaposed without interlocking blocks. This concept is translated in the program and no "T" intersections are merged. Besides, it is not possible to merge a T intersection with the same pattern used for the sequenced

walls. Indeed, as depicted on the following figures, a continuous joint of three blocks high would then appear in alternation on the left and right side of the intersection. (circled in white on the figure 51). This situation is to be avoided with the aim of eluding vertical crack transmission in the wall.

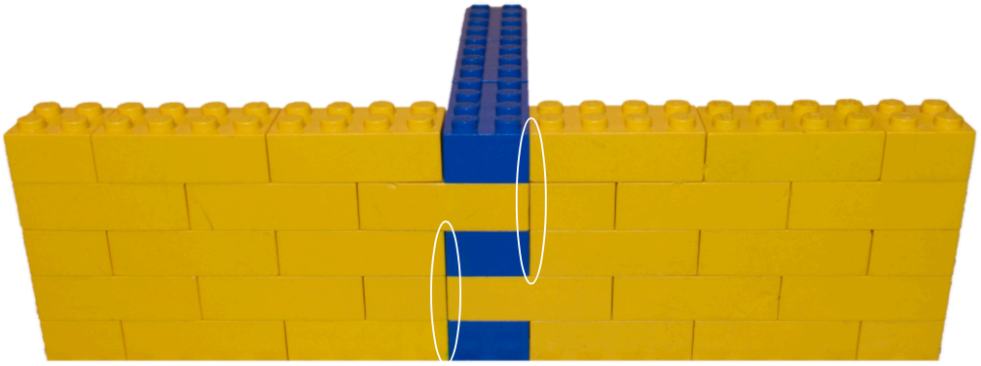


Figure 51 T intersection between walls

A suggested solution would introduce a 3 units wide drick as depicted on the following figure allowing to keep the alternation pattern.

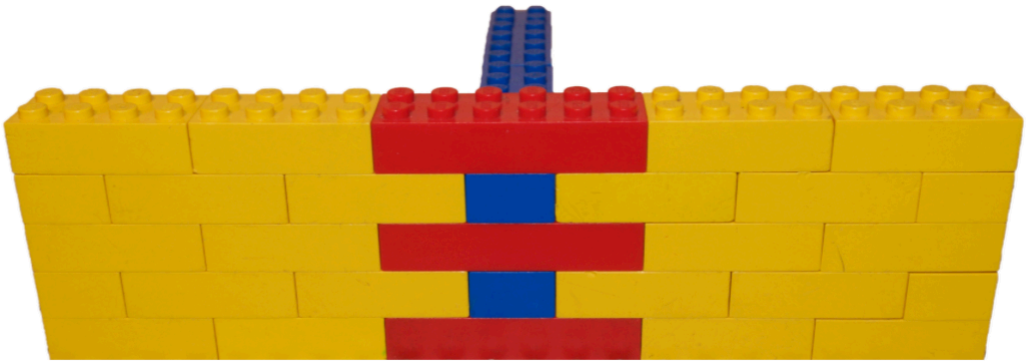


Figure 52 Suggested T section placement method

Placing an opening

When a wall is edited for placing openings, it is represented by a side view composed of rectangles. Each rectangle is one layer high and one mono drick width wide. Thus each rectangle represents either a mono drick or a half double drick. The green rectangles in the following figure represent every possible upper left corner for an opening that can be placed in the wall. When an opening is being placed, the opening appears white and when the user confirms his desire to place the opening, it appears blue.

The green zone of possible upper left corner for other windows is then updated. An upper left corner cannot be closer than one rectangle from the left side of the wall as the lintel is one rectangle wider than the window itself. The window is arbitrarily chosen to be 4 rectangles wide. Thus applying the same reasoning for the right side, gives a tape of 5 rectangles wide where no upper left corner can be placed. The two last layers of the level are not fitted for an upper left corner of an opening as there needs to be one layer for the lintel and then an ending layer of top flat dricks.

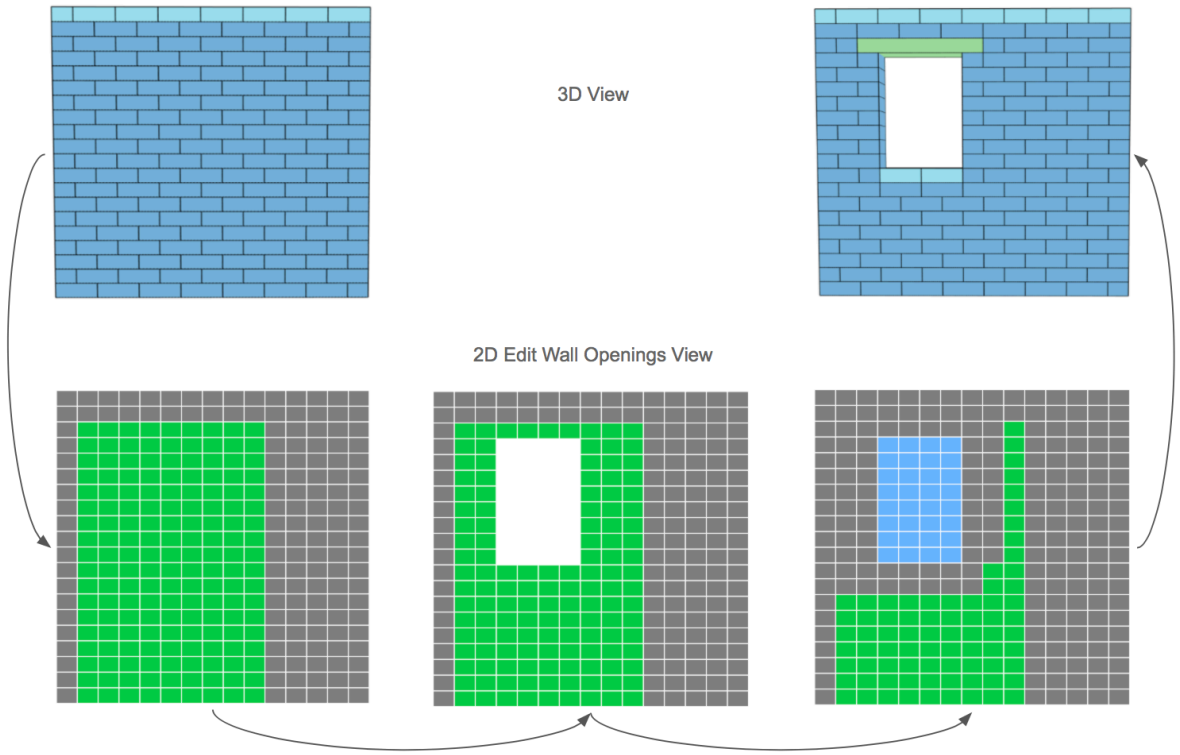


Figure 53 Placing an opening in a wall process

When an opening is placed, other openings cannot be placed in a way that would interfere with the previously placed lintel or one of the two flat dricks creating a balustrade for the window. So there is an extra impossible placement zone for the upper left corner to take into account:

As a window is chosen to be at least 3 layers high and there is always a layer of flat dricks under it, it means that a new window should have its upper left corner at least 4 layers above the lintel of another window or equivalently at least 5 layers above the upper left corner of a possible window under it.

A space of two layers is unused under a window because of the flat dricks under it and the lintel of the possible new window that could be placed under it.

A space of at least two rectangles is unused between two openings.

The pattern of possible upper left corners for a new window is shown in the figure 54.

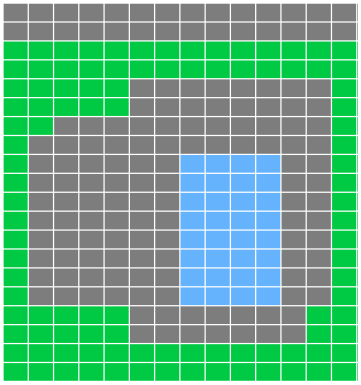


Figure 54 Possible placement zones for the upper left corner of an opening

The placement algorithm for an opening is explained underneath for a wall with yaw = 0°. The reasoning is exactly the same for the wall with yaw = 90° except that every applied action on the x coordinates is done on the y coordinates. Before starting, let's define a tile as a rectangle of the wall's side view representation.

STEP 0: Retrieve the coordinate of the tile representing the upper left and lower right corner of the window.

First, retrieve the coordinates of the center of the first and last drick of the wall.

If the wall is going right or down[10] **then** the x coordinate of the first tile (lower left rectangle) is the x coordinate of the first drick of the wall from which 0.5 is subtracted if the drick is a double drick **else** the x coordinate of the first tile is the x coordinate the last drick of the wall from which 0.5 is subtracted if the drick is a double drick.

A start offset is defined as the number of tiles between the side of the wall and the upper left tile of the opening. The stop offset is the start offset + 4 (because an opening is 4 tiles wide).

Then the coordinate of the tile representing the upper left and lower right corner of the window can be deduced by applying the previously found offsets to the coordinate of the first tile.

STEP 1: For every drick define a x_1 and a x_2 that are the relative coordinates of the two halves of the drick. Then for every drick in the wall that is between the "layer under"[11] and "layer top" (included) and that has its x_1 or x_2 is between or equal to the x_{start} and x_{stop} (see dark grey dricks on the following figure), then remove this drick from the *dricks list*. Then considering the previously deleted drick, if x_2 was x_{start} (respectively x_1 was x_{stop}) then place a mono drick in x_1 (respectively in x_2).

STEP 2: Is the same reasoning as STEP 1 except that it applies only on the "layer above" and between x_{start} – 1 and x_{stop} + 1. In the cleared space, a lintel can now be placed with its x coordinate being $x_{start} + 1.5$.

STEP 3: Place the two flat dricks on the "layer under".

[10] In other words : if the wall is drawn on the screen from left to rigth or from the top of the screen to the bottom of the screen.

[11] If the opening is not going until the lowest layer of the level. (else use the « layer under » + 1 as start layer)

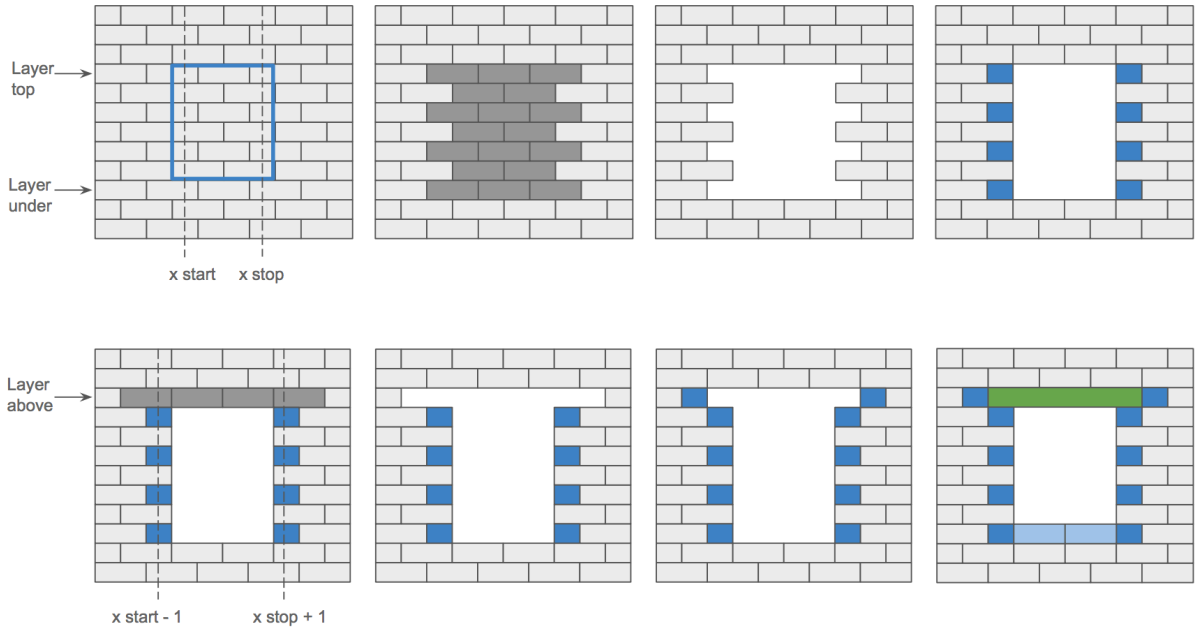


Figure 55 Illustrations of the steps of the "placing an opening" algorithm

Finally, proceed to a post processing algorithm to sort the *dricks list*: a "sequence verification". As the "extra" dricks placed for an opening (all the colored dricks in the above figure) are added at the end of the *dricks list*, they will be placed last. However, to assure that no drick will ever be placed between two already placed dricks (like the red dricks on the left picture of the following figure), a reorganization of the dricks in the *dricks list* is mandatory to obtain a sequential placing of the dricks as on the right picture of the following figure.

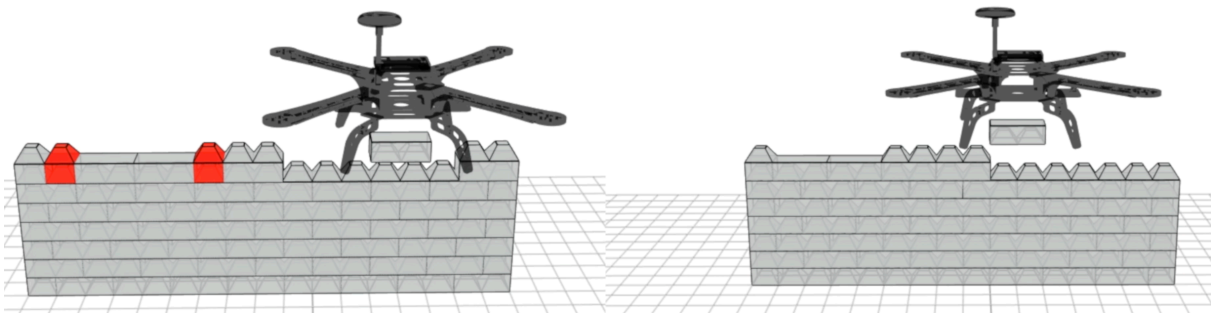


Figure 56 Non sequential placement vs. sequential placement of dricks around an opening

Sequence verification

The algorithm decomposes the *dricks list* in sub-sequences. There is one sequence for each layer of each wall. This means that one sub-sequence contains only dricks of the same layer and of the same wall, hence the same yaw.

The algorithm then sorts each drick composing the sub-sequence regarding the x or y coordinates depending on the yaw of the wall. It sorts the sub-sequence in an ascending or descending order depending on the wall direction. If the wall is drawn from left to right or top to bottom in the 2D grid on the screen (in the 'wall place' page) then a flag IS_GOING_RIGHT_OR_DOWN was set to true, else it was set to false.

Once all the subsequences are ordered, they are re-assembled to form the *dricks list* again. Nevertheless, if the algorithm would stop there, the first drick of uneven layers (in the case of a closed loop) carefully skipped in the "sequenced walls" algorithm would now be placed back at the beginning of their sub-sequence (and not at the end of the layer as it should). To rectify this situation, the *dricks list* is scanned again and all the "skipped" dricks are removed from it and placed back at its end in the same order they appeared in the *dricks list* before the scanning. Then sorting the *dricks list* per layer assures that the "skipped" dricks are back at their appropriate index in the list.

Modularity in drick sizes

Throughout the whole program, the proportion of the drick ½ is used. The program is made such as it is possible to change the width of the drick (hence the depth that is twice the width) but always with this assumption that the proportions are 2 to 1. The height of the drick can be changed unconditionally. To do this the user simply needs to change the "Wall parameters" in the parameters page and to add the corresponding STL files in the right folder. The STL files are stored under the folder "models". The sub folder "spacing_Δ" where Δ is the width of the drick contains all the models with width equal to Δ. Then likewise the sub-sub folder "height_H" contains all the models with height equal to H.

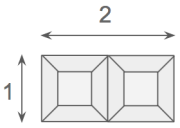


Figure 57 Drick standard proportions

Creating the wished folders and placing the STL models in the right subfolders guarantees a truthful simulation to the user.

Placing a floor system

When the user clicks a first time on the 2D grid in the "Place or delete a wall" page, it determines the upper left corner (on screen) of the floor system. It is represented on the following figure as the green dot. Let's give this green dot the coordinates $[x_1, y_1]$. Then when the user moves his mouse to the right side of the screen (or to the bottom if the system is turned 90°), a floor system appears in grey. The blue dot represents the coordinates pointed by the mouse. Let's give this blue dot the coordinates $[x_3, y_3]$.

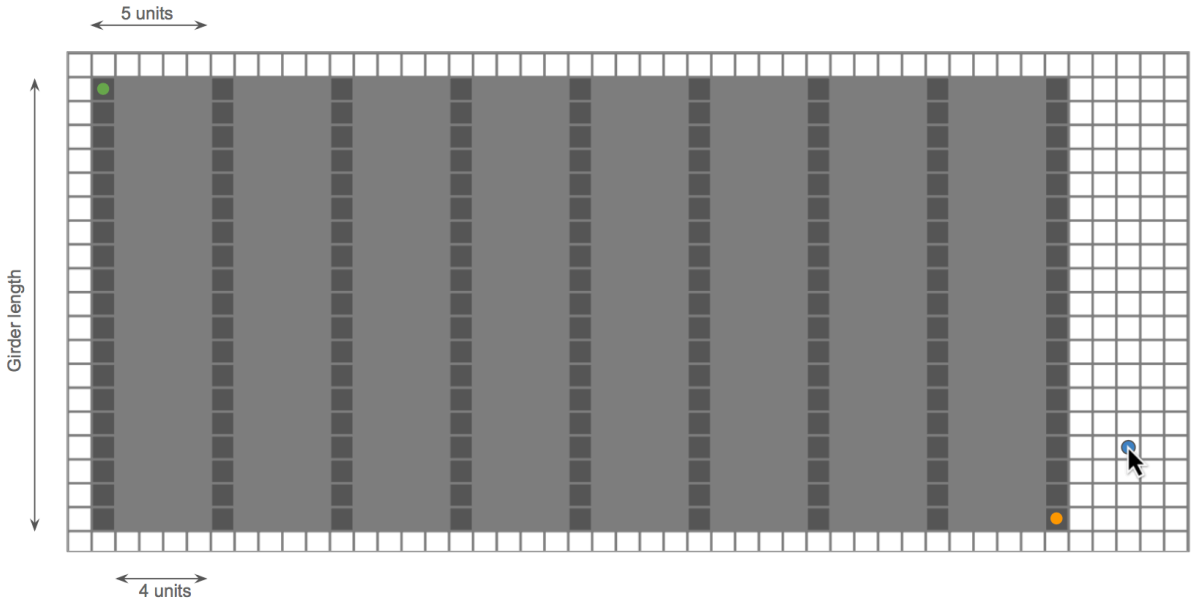


Figure 58 Illustration of the floor system placement process

In the above figure, the wall does not stretch out to the blue dot but to an orange dot that is computed based on two parameters: the length of the girder and x_3. Let's give this orange dot the coordinates $[x_2, y_2]$. Then y_2 is y_1 + the girder length and x_2 is an 'adapted' value of x_3.

This is because a floor system is always a succession of rows of girder-slab-girder-slab-girder etc. It starts and ends with a girder. Every girder (dark grey in the above figure) is placed 5 units to the right on the screen (or to the bottom if the yaw is changed) of the previous placed girder. The sequence of offsets from the start point for a girder is thus 0,5,10,15...

To find the maximum value that x_2 can take, a simple mathematical operation is executed on the x_3 value. This adapted value x_2 is equal to x_1 + floor($\frac{(x_3 - x_1)}{5}$) × 5.
The y value of the center of the girder is $\frac{(y_3 - y_1)}{2} + y_1$.

Possible further improvements

The main limitation in the actual configuration of the floor system is that the girders are always separated of a distance of 5 units (if 1 unit corresponds to the width of a mono drick) because the slabs (light grey in the above figure) are 4 units wide. A further improvement could be to add more different slab dimensions so that whatever the value of x_3 is, x_2 would always be equal to x_3. By introducing slabs of 2 and 3 units wide all combinations of floor systems are possible from 4 units wide until any desired size. The first ten combinations are illustrated in the following figure.

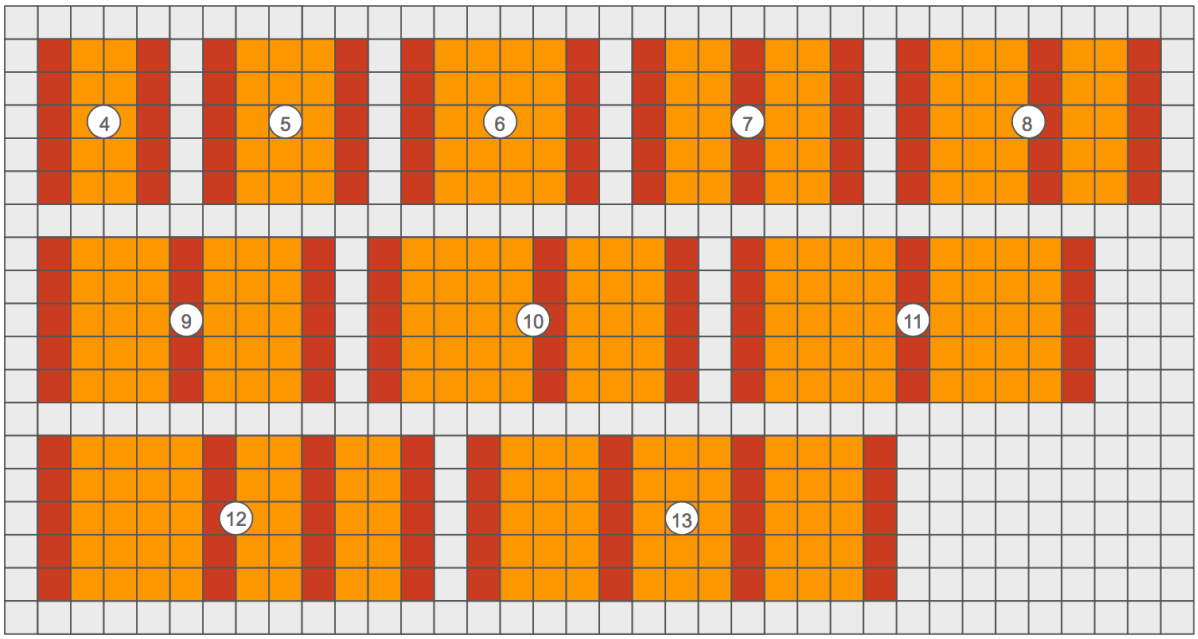


Figure 59 First 10 combinations for a floor system composed of of 2,3 and 4 units wide slabs

Conclusion

Even though some of the concepts used as a mason in the process of brick laying can seem obvious beforehand, translating those workflows into rules is not always a simple task. Contrary to the mason who evaluates the situations and the position of each block one after the other, the computer program predicts in a blink the exact and suitable position for every drick in a wall or sequence of walls. Although, determining these algorithms was not a straightforward process as every possible outcome of each algorithm had to be found by trial and error by testing them with Lego blocks. Afterwards, each outcome has been categorized. Each category could then be compared to others and some relations between the groups have been identified. Then, just as going up in a family tree, by grouping some sub-groups together as one goes up, a main relation could be identified as the start point of the algorithm. By creating formal rules, it was thus possible to establish repeatable patterns that benefit from geometric relationships between the parts of the design and took us from sometimes basic concepts to a deterministic result.

By placing and merging walls as sequences, it's possible to create complex closed loop sequences of walls that are perfectly intermingled. It is even possible to add openings by removing existing dricks and placing new ones such as lintels or flat dricks for a smooth finish like a balustrade. All these elementary construction elements can be transported by a single drone. If a second drone is added to the equation, it is even possible to place a floor composed of girders and slabs. Whereas the slabs can be placed by a single drone, the longest girders most certainly will require the use of two drones. On top of all this, the program offers some modularity by allowing a certain change in drick dimensions.

Dronicks is thus an elementary BIM program capable of designing multi-level houses that can be built by using at least one drone.

CHAPTER 3

DEVELOPMENT OF ALGORITHMS

CONTROLLING THE CONSTRUCTION

PROCESS

Main questionings

When dealing with drones and construction together, two main questions arises:
"How to avoid drone collision with the already built part of the structure?"
"How to minimize the project cost?"

Drone collision avoidance

The answer to the first question is actually quite simple and finds its source in the way 3D printers work. Printing layer by layer from bottom to top, the head of the printer never touches the already printed part. Applying this to the manufacturing of drone based structures, the drone places all the blocks of the 1st layer then the 2nd and so on.

A second technique is possible, but has its weaknesses: using a staircase pattern.

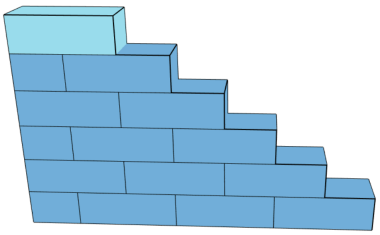


Figure 60 Stair case pattern

This obliges the drone to carry the drick in such a way that the propellers are always on the left and right side of the alignment of the wall (two above pictures in the figure 61). If the drone would modify its yaw, there is a high risk that its propellers touch the wall.

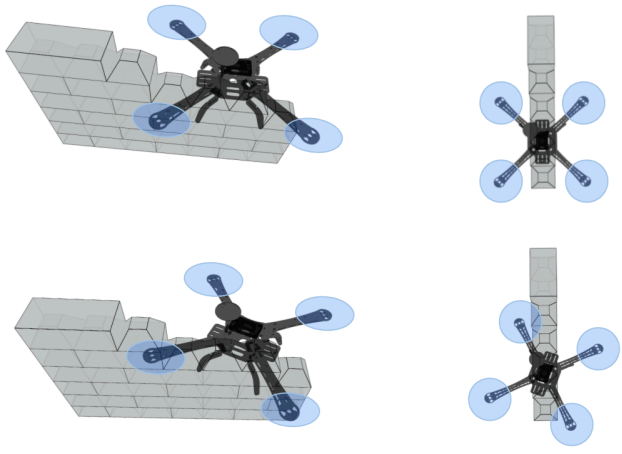


Figure 61 Collision scenario in staircase pattern (a)

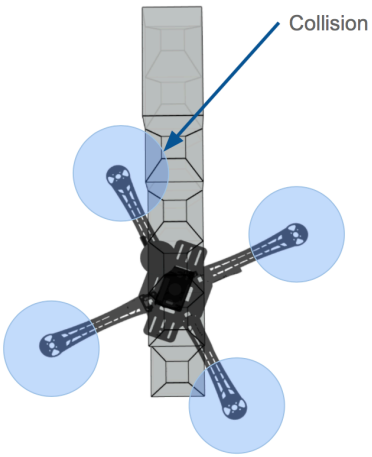


Figure 62 Collision scenario in staircase pattern (b)

For the sake of simplicity and to minimize the collision risk the first solution is used throughout this project.

Shortest Path 2D

The answer to the second question "How to minimize the project cost?" is not that direct. Among others such as finding the minimum drick cost, reducing to the minimum the travel time is one way of minimizing the total project cost. Reducing the travel time can be achieved by reducing the travel distance, using the shortest path between the stockpile and the final position of the drick.

Of course the best way to deal with the problem of the shortest path would be Dijkstra's algorithm which finds the shortest path between two nodes in a graph.

A graph is a data structure composed of nodes interconnected with edges that can be associated with a numeric value that can represent a cost, a length, … A graph can be imagined as a map with the cities as nodes and the roads as edges. Each edge is associated with a value: its length. Most GPS systems use the Dijkstra's algorithm to propose the shortest path from one point to another.

Dijkstra's algorithm

Let the node at which we are starting be called the START node. Let the target node be called TARGET. Let the current node be U. Let the neighbors of U be V. Let the distance of node V to U be d. Let the distance of V to START be D. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

Adapted to the problem of the drone going from a point A to a point B with obstacles on its way, the algorithm can be written in pseudo-code as:

STEP 0

The drone needs to go from A to B with an obstacle on its way.

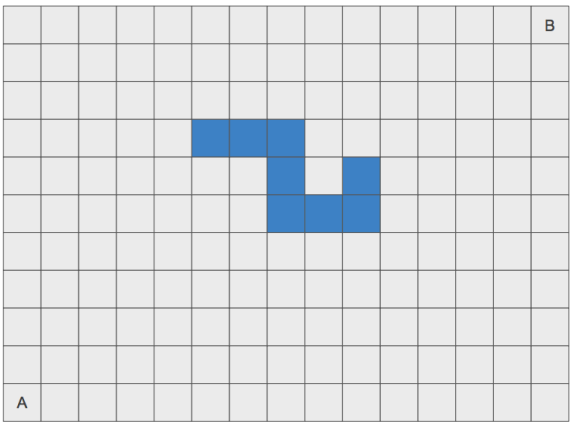


Figure 63 Dijkstra's algorithm illustration step 0

STEP 1

Create a grid with a defined resolution for example a decimeter. Every dot is a node that has *x,y* coordinates. With the orange ones being the nodes already explored (for step 1 only the START node). We create a matrix M of the size of the workspace and populate every element of the matrix as being an object with attributes
{X_{COORD}: x; Y_{COORD}: y; GO-ZONE: *true/false*; NEIGHBORS: *list of neighbors* ; D:-1; HOW_WE_GOT_HERE: [*i,j*]}

As computers cannot understand the concept of infinity, set *D* to -1.
Set START's HOW_WE_GOT_HERE to "start".
Set TARGET's HOW_WE_GOT_HERE to "target".

Set HOW_WE_GOT_HERE = [-1,-1] for the other nodes.[12]
Set NEIGHBORS as an empty list.

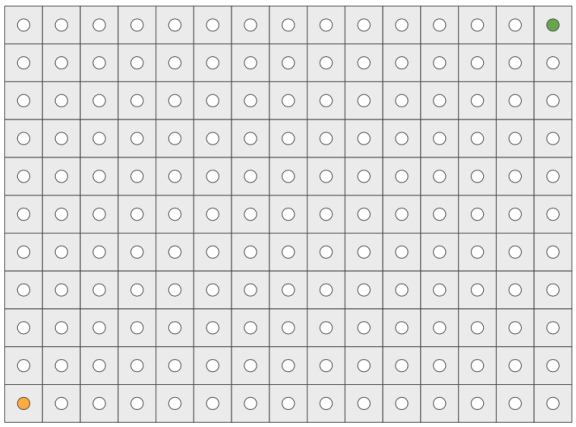


Figure 64 Dijkstra's algorithm illustration step 1

STEP 2

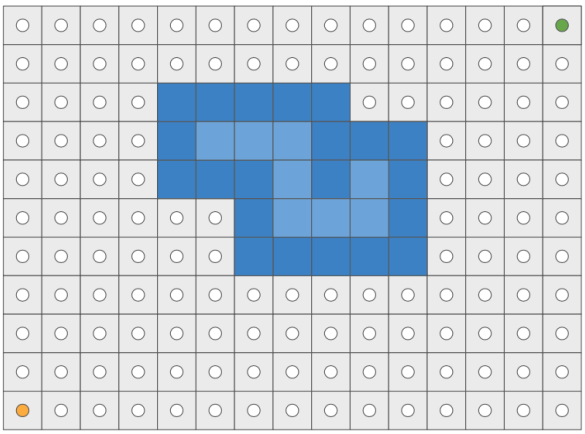


Figure 65 Dijkstra's algorithm illustration step 2

[12] -1 is an alternative to the concept of infinity that a computer cannot understand. In this case, when the distance is negative, it is equivalent to infinity.

Create a no-go zone of the size needed (for example two times the drone width) around the obstacle by setting for every node in the no-go zone[13] the NODE's GO-ZONE to false.
For every node compute its neighbors and add them to the *list of neighbors* of the NODE (only if they have their GO-ZONE flag to true). By doing so no node is linked in any way to a node in the no-go zone.

Now add every object of the matrix into a queue called Q.

STEP 3

Set node U to START.

While Q is not empty **do**

Remove U from Q;
NODE_WITH_MIN_DIST_TO_U = new Node; //define a node

For each neighbor V of U{

If V is explored, **then** skip it **else**
Get distance d from U to V; // $d = \sqrt{\left((x_{coord_v} - x_{coord_u})^2 + (y_{coord_v} - y_{coord_u})^2\right)}$

If V.D is -1 or V.D> d+U.D **then**
V.D = d+U.D;
HOW_WE_GOT_HERE = $[U_i, U_j]$;

If NODE_WITH_MIN_DIST_TO_U is -1 or NODE_WITH_MIN_DIST_TO_U.D > D **then**
NODE_WITH_MIN_DIST_TO_U = $[V_i, V_j]$;

If U.HOW_WE_GOT_HERE is equal to "target" **then** stop the iterations on the matrix.
U = NODE_WITH_MIN_DIST_TO_U; // Set U to the node V closest to the previous U and start a new iteration.

[13] The blue zone is the no-go zone and the light blue zone is the obstacle.

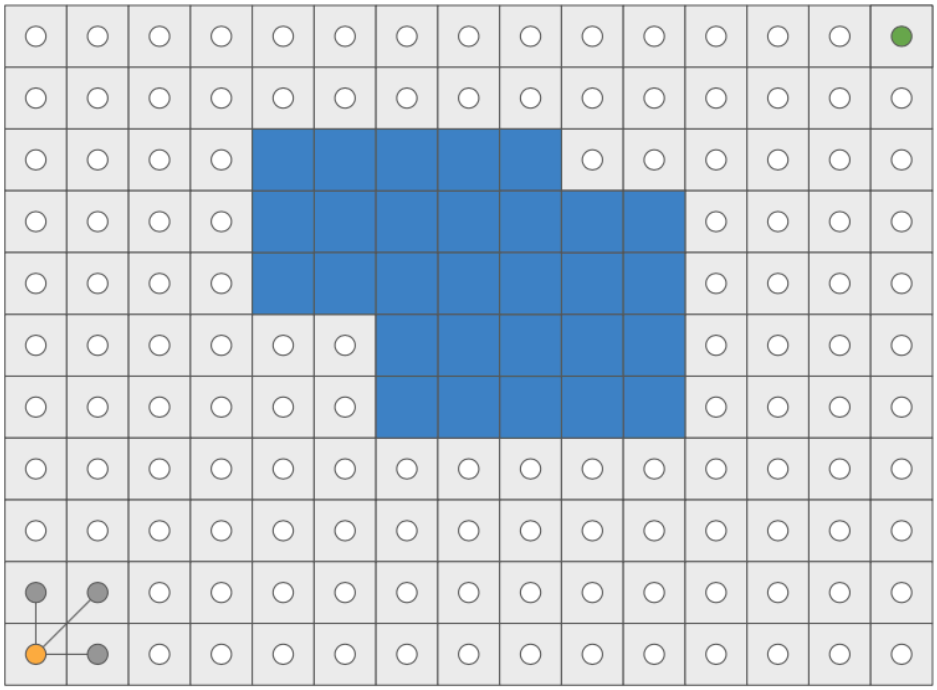


Figure 66 Dijkstra's algorithm illustration step 3 (a)

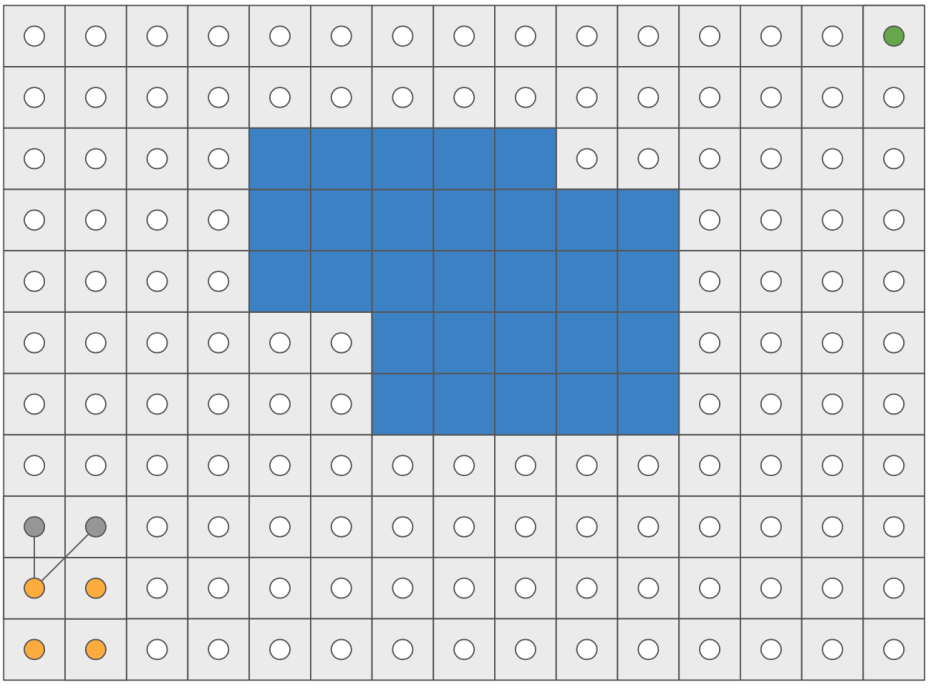


Figure 67 Dijkstra's algorithm illustration step 3 (b)

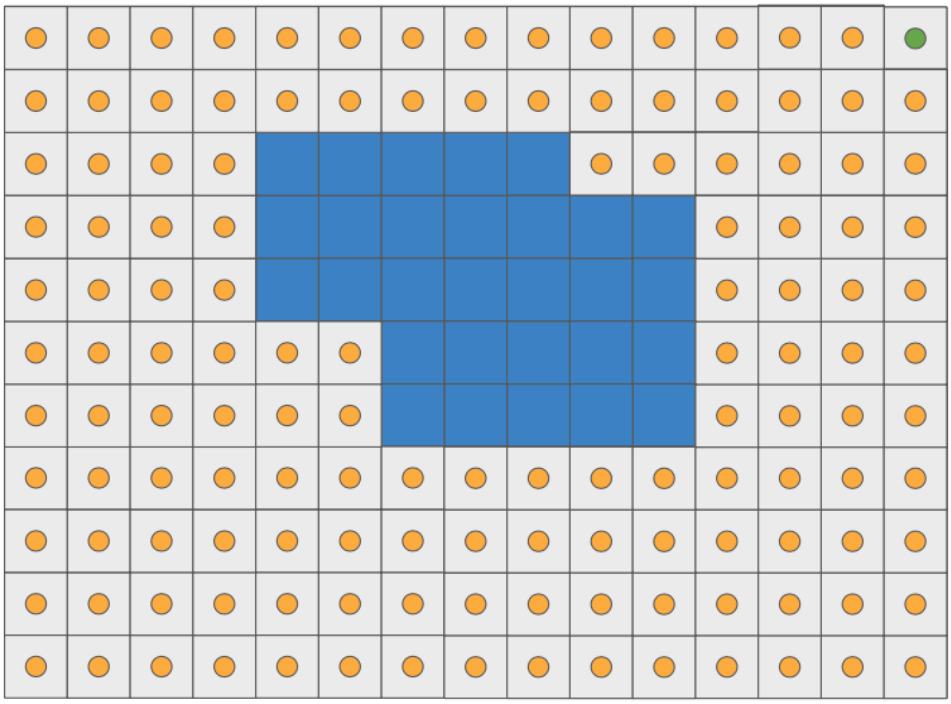


Figure 68 Dijkstra's algorithm illustration step 3 (c)

Once all the neighbors of the target have been searched, go to point TARGET (Its *I,J* indexes in the matrix are known in advance). Place the coordinates of TARGET at the beginning of a new stack S. Then go to the TARGET's HOW_WE_GOT_HERE and place its coordinates at the beginning of S. Repeat until the NODE's HOW_WE_GOT_HERE equals "start".

The stack S now contains all the coordinates of the shortest path between A and B taking into account an obstacle.

Shortest Path 3D

This method can easily be adapted to a 3D scheme. The matrix M has now 3 dimensions and every node has more neighbors than before. The rest of the process stays the same.

Straight line path

A more simplistic way of thinking and totally adapted solution to the needs of a little dimension construction model goes as follows from stockpile (x_s, y_s, z_s) to target position (x_t, y_t, z_t):

1. Take off and go to vertical target altitude + safety marge S_m[14]: $(x_s, y_s, z_t + S_m)$
2. Go to $(x_t, y_t, z_t + S_m)$
3. Go to (x_t, y_t, z_t)

To go back to the stockpile, do the same in reverse order.

[14] This safety marge should be at least the height of the drone and its drick lifting system plus the height of a drick plus an extra safety length to avoid random risks.

Attention should be paid to the fact that the drone lowers the drick in a vertical plumb line and can cause the drick that is being placed to collide with a neighbor drick during its descent due to the drone inaccurate positioning system. A more refined solution, that is implemented in the '*Dronicks*' program, is explained further in this chapter at the sub chapter *'Best drick placement methods'*.

Shortest Path solution used

So far, the displacement method of the drone to pick up a new block at a stockpile is done by using the straight line path. This algorithm is simple but efficient. The only inconvenience that could occur is the ground effect when flying over a portion of already built wall.

The ground effect arises under the altitude of 1 time the span width of an aircraft or 1 time the rotor diameter for a helicopter [21]. An interference occurs when the airflow pattern is disturbed as it encounters a flat surface. This effect is more pronounced nearer the ground surface. The effect is illustrated in the following figure.

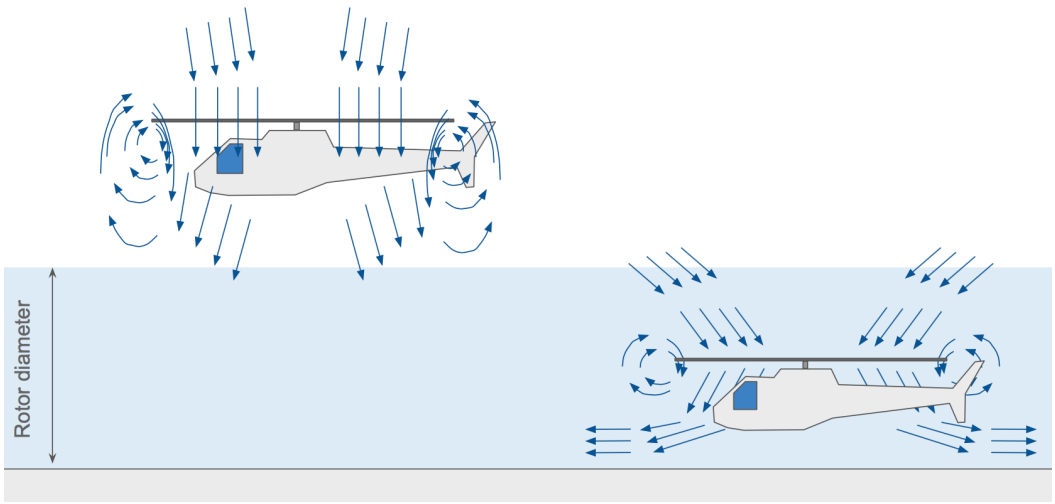


Figure 69 Helicopter not subjected (left) / subjected (right) to ground effect.

For a multi-copter drone the whole zone under the drone will undergo a turbulent flow as it has multiple parallel propellers. Indeed, the airflow of two neighboring propellers will interfere and thus produce a turbulent zone under the drone that will cause instabilities. This effect is illustrated in the following figures.

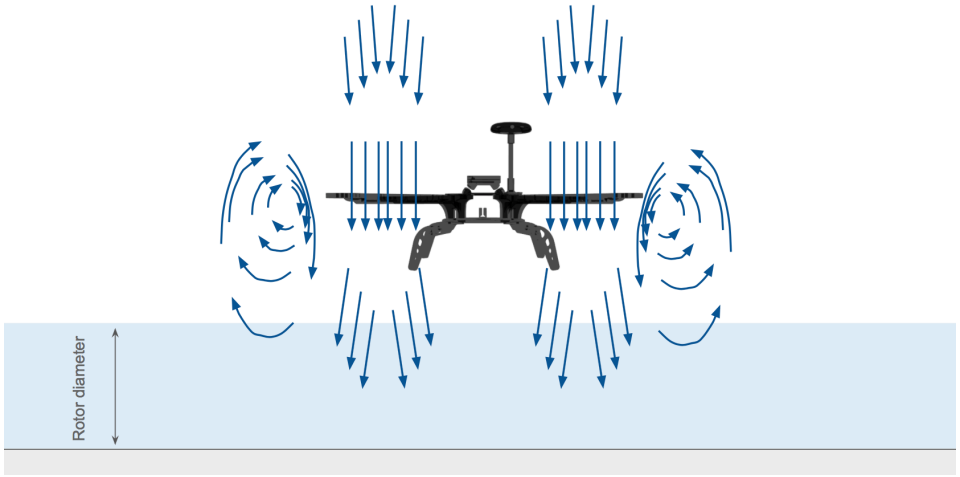


Figure 70 Drone not subjected to ground effect

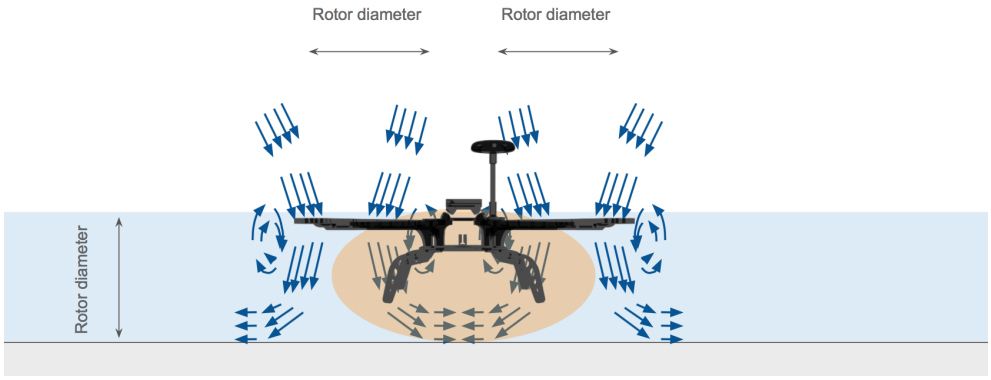


Figure 71 Drone subjected to ground effect

Although, those effects haven't been quantified yet in this research so the most simplistic solution (straight line path) is implemented and integrated into the '*Dronicks*' program. To minimize the ground effect, assuring that the drone flies at a height at least as great as one rotor diameter can be sufficient to avoid ground effects. Of course when placing a block, instabilities can occur but this would happen in any of the two path solutions.

Optimum stockpile position

An easy way to deal with the optimal stockpile position is by calculating all the possible stockpile positions that respect some predefined rules. Those rules can be for example:

- The stockpile should at least be x meters away from any wall, in other words, a no-go zone is defined;
- The stockpile position cannot be inside the convex hull[15];
- The stockpile should be placed in a certain zone (near the main entrance for accessibility);
- …

[15] An often used comparison to illustrate the convex hull is to define it as the region in a plane that would be limited by an elastic which contains all points and that is released until it is contracted maximally. Many algorithms exist to solve this problem but the simplest is the Gift Wrapping Algorithm.

First method

For each stock pile position, summing the travel time needed for placing each drick using the Dijkstra algorithm, gives the total project time associated with that stockpile position.[16]

Then, picking the lowest value among all assures the optimal stockpile position.

Second method

A more refined way to do this is by calculating the center of gravity of the structure. This ensures finding the very best position for the stockpile.

$$CR_x = \frac{\sum_{i=0}^{n} dricks[i].x}{n}$$

$$CR_y = \frac{\sum_{i=0}^{n} dricks[i].y}{n}$$

with

CR_x the x component of the coordinate of the center of gravity and CR_y the y component of the coordinate of the center of gravity.

$dricks[i].x$ the x component of the coordinate of the ith drick of the list containing all the dricks.
$dricks[i].y$ the y component of the coordinate of the ith drick of the list containing all the dricks.

However, there's great chance for this position of the *CR* to be inside the convex hull and thus a place where it should rather not be for safety reasons. Or, if a second level is placed the stockpile becomes inadequate.

To find the next best position that will be authorized, all possibilities of the go zone should be listed and sorted by distance from the center of gravity. Then, the closest value to the center of gravity is the optimum position for the stockpile.

The optimum stockpile position can be computed in *'Dronicks'* and entered as the stockpile to use.

On the figure below, the blue zone is the no-go zone and the light blue zone are the walls. The orange dot is the center of gravity and the green dot is the optimum stockpile position.

[16] Those type of algorithms are called brute force algorithms or exhaustive search algorithms because they list all potential solutions and check whether they meet the requirements.

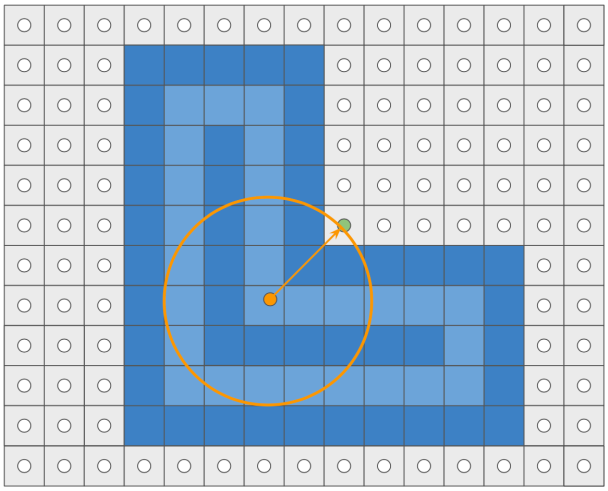


Figure 72 Optimum stockpile position

Multiple stockpiles

A reduction in the total project time can also be achieved by placing more than one stockpile. If the number of wanted stockpiles is defined, the dricks can be divided in clusters based on the density of dricks in a certain zone on the work site. In the following scenario illustrated on the following figure, two clusters can intuitively be distinguished. Hence the placement of one stockpile in the vicinity of each cluster instead of one at the center of gravity of all the dricks can lead to a significant reduction in the placement time of all the dricks.

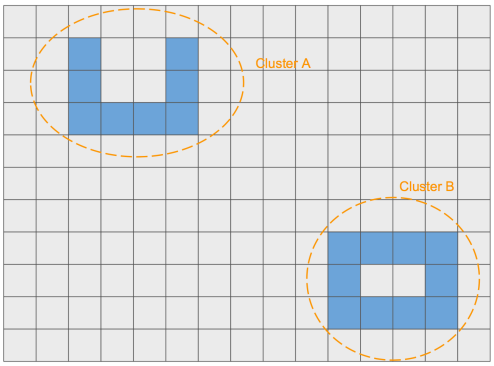


Figure 73 Clustering of the dricks

Cluster detection is proposed in the '*Dronicks*' program but not fully integrated. The program detects and indicates the best stockpile positions according to the number of stockpiles the user enters but does not use those positions in the simulation. The "k-means" algorithm [23] is therefore chosen among others for its simplicity of use. Indeed, only one simple parameter is needed: the number of required clusters. The algorithm creates arrays containing all the objects of one cluster. A post process algorithm computes the center of gravity of that cluster (or in other words the optimum stockpile position for that cluster). Even though this algorithm is not density based, it is often used for clustering. The "DBSCAN" algorithm or the "OPTICS" algorithm could have been used as well and are equally appropriate but require nevertheless more arguments: the minimum number of objects to form one cluster and the maximum radius of the cluster. Those algorithms are thus less suited as they cannot beforehand determine the number of clusters that will be computed.

"k-means" algorithm

In the light of this thesis the algorithm can be seen as:

STEP 0: Define the number "k" of clusters, hence stockpiles, you want.

STEP 1: Assign k random [x,y] coordinates, called centroids, in the definition domain of the problem. (for example the work site zone).

STEP 2: Associate each drick with the closest centroid. (K clusters now exist).

STEP 3: For each cluster, compute the centroids, hence the new stockpile positions.

STEP 4: Associate each drick with the closest centroid. (K clusters are redefined).

STEP 5: For each cluster, compute the centroids, hence the new stockpile positions. If those new stockpile positions are exactly the same as the previous ones (or they moved a distance Δ lower than a predefined threshold ε), then the optimum stockpile positions have been found, otherwise go back to STEP 4.

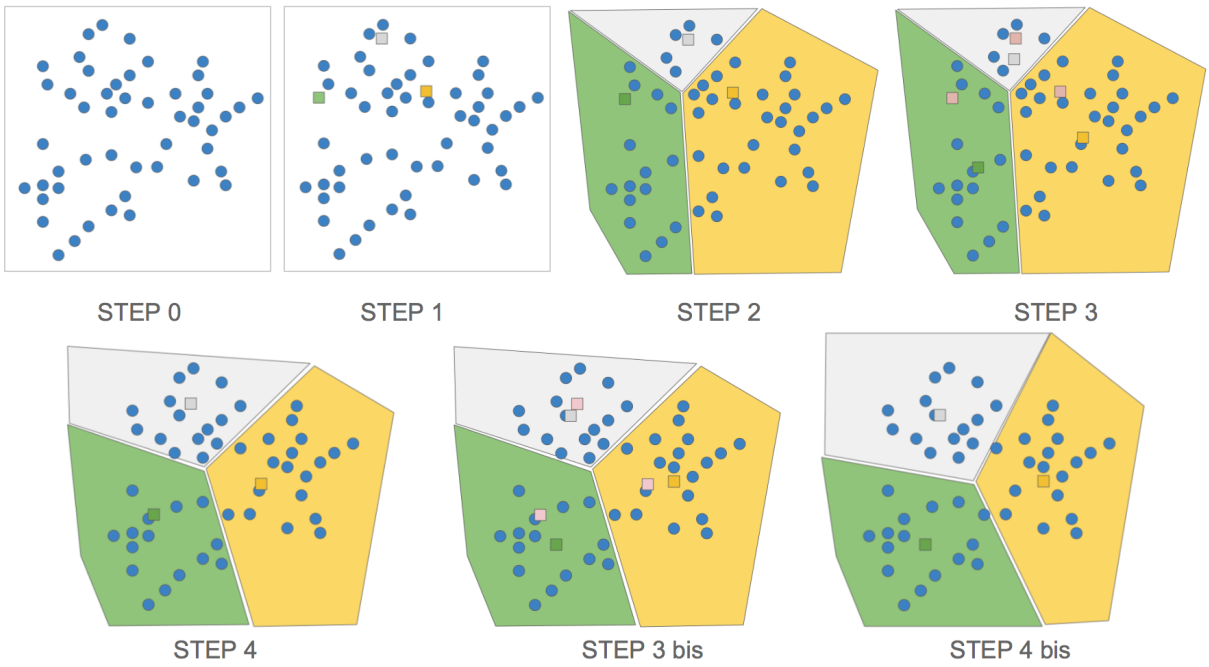


Figure 74 k-means steps illustrated for k = 3

How to avoid collision in a multi-drone environment?

Researchers led by Professor Raffaello D'Andrea at ETH Zurich, have built a program called the "foreman" that monitors the erection of a drone-built structure. The program is designed to enable multiple drones to work simultaneously on the construction. While some are recharging batteries on nesting boxes, the others fly around in a coordinated way. To avoid collision, the work site is also

divided in go and no-go zones. The parallelepiped with as base the convex hull of the structure is a no-go zone. This avoids collisions with the previously placed dricks. In order to avoid inter drone collisions, a zone, called a “freeway”, is reserved for each one of them. For all other drones this is a no-go zone. The attribution of a freeway to a drone is controlled by a space reservation system that assigns a space for one single drone before the drone’s trajectory is flown. If for another drone it is not possible to use another trajectory than one passing through this reserved space, it will have to wait until the drone who has reserved the space finished its task. [16]

This concept of “freeways” can be integrated in the ‘*Dronicks*’ program as from the moment more than one drone will build the structure. Once again the Dijkstra’s algorithm can be used in a 3D discretized work site.

The concept of freeways (black and red "tubes”) and space reservation system is illustrated in the figure below. This is a capture of the program “The foreman”.

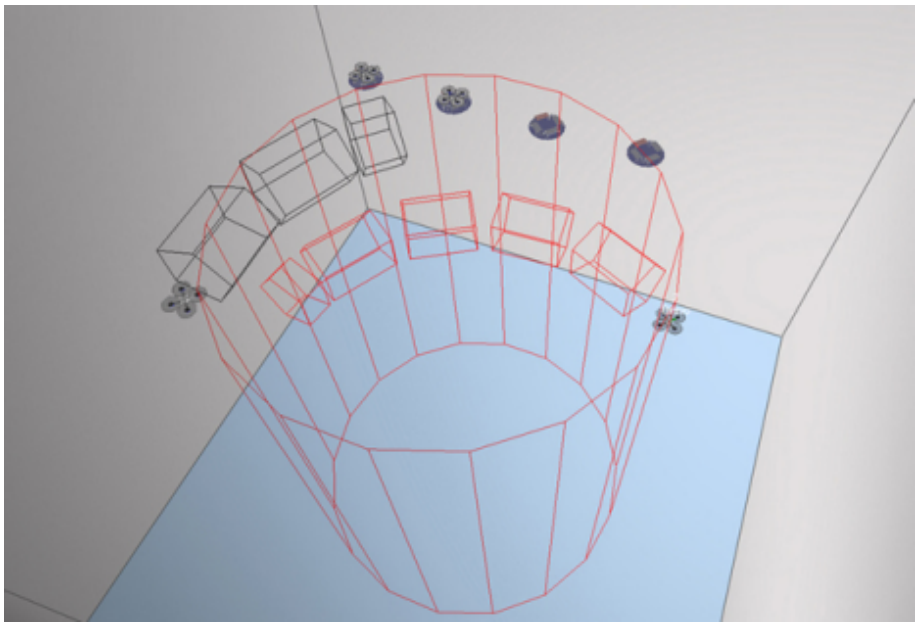


Figure 75 The 'foreman' program [i13]

Best drick placement methods

The final approach

Another conclusion drawn from D’Andrea’s studies is that the most accurate and reliable method of placing a drick, is to drop it quickly into place. This method avoids disturbances due to turbulences or ground effect occurring under the drone when it is close to a flat surface, hence a faster and more aggressive impact is recommended. Indeed, being able to accurately and reliably place a drick is essential to a flawless construction.

The closed loop pattern

In order for the final approach to be quick, the drone should be able to place the drick as fast as possible and without having to start over because the block is not perfectly aligned: it should be a one shot action. For this reason, dricks should be placed in a way that there are no other dricks blocking the placement process. Indeed, one of the dreaded scenarios is if a drick hits another block due to the drone's inaccuracy and falls in a leaning way as depicted on the following figure.

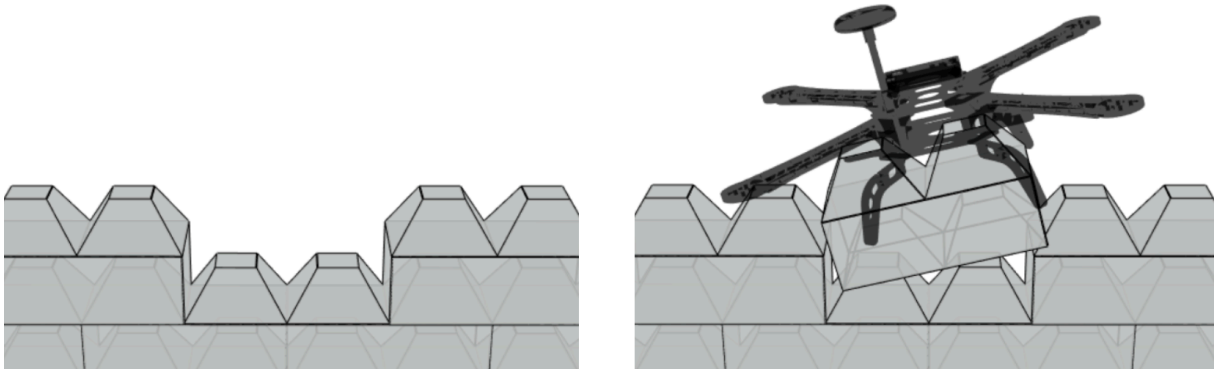


Figure 76 Faulty block placement

A solution to avoid this kind of situation, is to place the dricks in a way that there is always one of the two sides of the drick free.

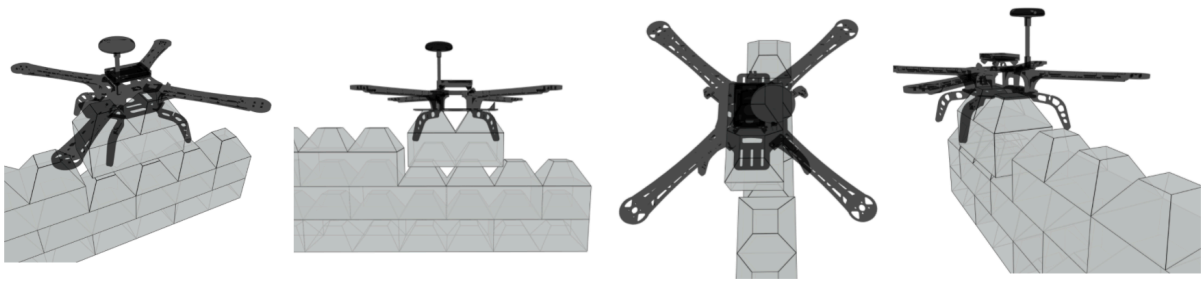


Figure 77 Drick placement simulation details

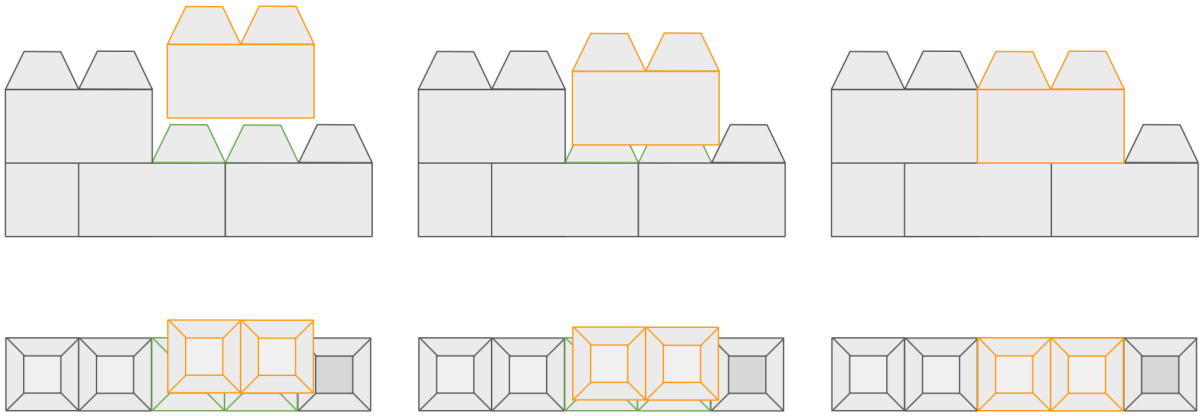


Figure 78 Side and top views of the last steps of a drick placement

An offset is given to the drick that is being placed in both directions in the plane. This is because when placing a corner drick[17], in addition to the drick on the side, there is another drick perpendicular to the one currently being placed.

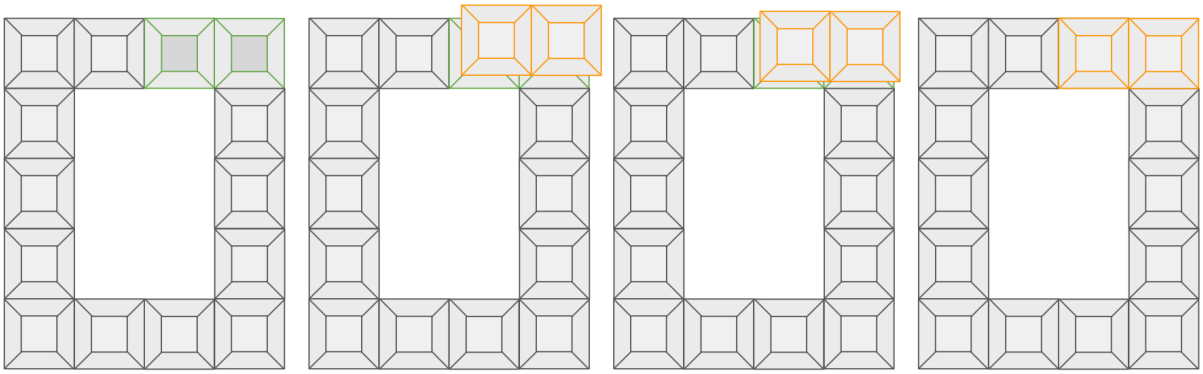


Figure 79 Top view of the last steps of a drick placement in a closed loop of dricks

It is necessary to note that this technique is only possible if the drick is lifted from the top. If the drick were to be placed with a grip on its side, this technique wouldn't be possible and the pattern should be adapted.

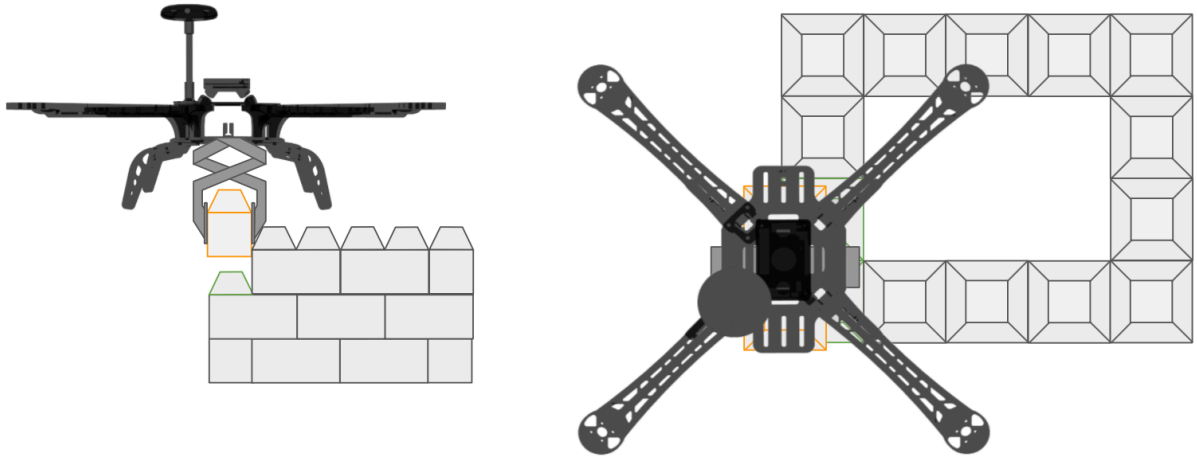


Figure 80 Collision scenario of the grip of the drone and the existing walls

[17] The last one of a closed loop of dricks.

An adapted pattern ensures that a layer will never finish with a corner drick such as on the above figure. Therefore, a wall starts with its first drick. Then dricks are placed until the last but one drick of the wall. The corner drick of the next wall is subsequently placed. Finally, the skipped drick of the previous wall is placed. The workflow for this technique is illustrated in the following figure. This technique though requires to place a drick between two previously placed dricks and is therefore to be avoided.

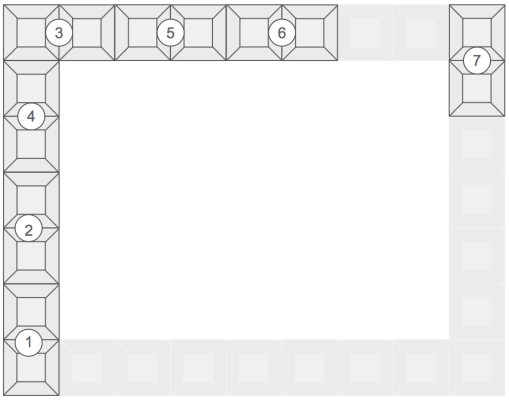


Figure 81 Workflow for alternative drick placement method

Control by simulation

In the simulation scene, four main options are possible. Playing the simulation, stopping it, going fast forward and fast backward.

Let's PLACED_LIST be the array of the already placed dricks in the scene and TO_BE_PLACED_LIST the array of all the loaded dricks but not yet placed in the scene. At first PLACED is empty and TO_BE_PLACED_LIST length is equal to the total number of dricks in the project.

The play function iterates through every line of the POSITIONLIST which is the same list as the PATH_PLANNING_LIST except it has more checkpoints for the drone to go to than the PATH_PLANNING_LIST. The number of extra steps is determined by a SIMULATION_FLUIDITY parameter entered in the parameter page. Every line indicates where the drone should move to and what action it should undertake. When the instruction in the list is "Take new block" then it removes the first drick of the TO_BE_PLACED_LIST and adds it at the end of the PLACED_LIST and finally renders it in the scene. With the next steps and until the instruction is "Leave the block" this drick will move to the same coordinates as the drone but with a certain offset (the drick is always a certain distance under the center of the drone). Then the block stays in final position and the drone continues going to the coordinates of the POSITIONLIST. This leads the drone back towards the stockpile where it will grab a new drick.

The fast forward function does pretty much the same except that it does only read one line of the POSITIONLIST: the next line containing the instruction “Leave the block”. It will thus remove the first drick of the TO_BE_PLACED_LIST and add it at the end of the PLACED_LIST and finally render it in the scene at its final position.

The fast backward does the contrary of the fast forward function. It takes the last drick of the PLACED_LIST and adds it at the beginning of the TO_BE_PLACED_LIST. Finally, it removes that drick from the scene.

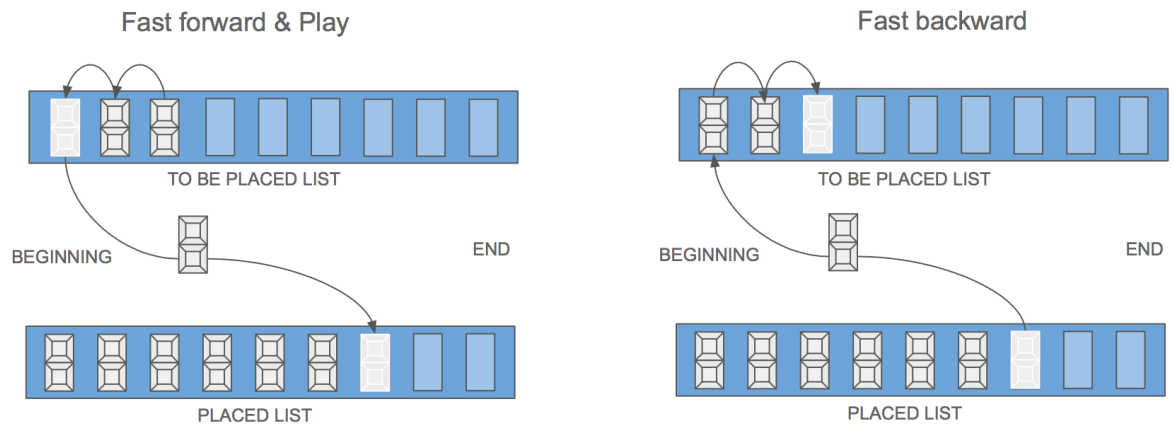


Figure 82 Illustration of fast forward, play & fast backward functions

Even though the dricks list is sorted in order to avoid wrong placed dricks, particular cases that are not envisaged can occur. This is why, if this would happen the dricks are colored red in the simulation and their id is shown to the user before the simulation starts.

This simulation can help eradicate most of the misconceptions, before starting the construction for real, by seeing on screen exactly what will happen beforehand.

Conclusion

The second layer of artificial intelligence for an autonomous building drone will be the planning of his route beforehand. The chosen technique for avoiding collision is to always fly higher than the last built layer of the edifice. Even if in this way it will avoid collision with the already built parts of the structure, it will though not prevent the drone from a collision with an unexpected obstacle such as a worker who moves around on the construction site. Therefore, another layer of artificial intelligence will have to be built in the drone core program (the first layer of intelligence).

The drone will possibly be prone to ground effects when hovering above an existing flat part of the structure i.e. a wall or a floor. Just like for the obstacle avoidance, the drone will need an extra layer of artificial intelligence to regulate its position when subject to ground effects.

One of the main objectives of the collective thesis developed at UCL under the supervision of Professor Latteur is to assure that drones can be more cost effective than masons. Therefore, it should be as autonomous as possible in its tasks, thus requiring less maintenance and monitoring and it should optimize the project time by minimizing the travel distances it carries out. To manage this, firstly the position of the stockpile is being optimized, so that it is as close as possible to the center of gravity of the construction. If not sufficient enough, another optimization could be done by clustering the edifice in sub parts of close dricks and to assign an optimally positioned stockpile for every cluster.

Another major objective is to ensure a flawless construction process. Thanks to a simulation it is possible to pre-visualize every step that the drone will carry out. This allows to eradicate complications before proceeding to the real mission. Another way to ensure flawless construction is by avoiding to place a drick between two other dricks. Therefore, re-writing the list containing all the dricks in a sequential order is done after adding an opening to a wall. Also for a closed loop sequence of walls, it is assured that the last drick of the sequence is a corner drick, avoiding thus the placement between two existing dricks. Moreover, the dricks are always placed with a certain offset compared to the already placed neighbor drick so that if there would be an error in the drone's position, the drick being placed would not touch its neighbor drick before being in final position. This workflow constrains however the dricks to be lifted from above. If the dricks were to be lifted by a grip grabbing it on its side, another workflow would need to be followed but then again dricks are placed between two existing dricks. The dilemma forces to choose one of the two techniques and to further develop either the placement process (between two existing dricks) or the lifting process (lifting from above with for example a magnet).

Further improvements

The existing '*Dronicks*' program can be improved at many levels. Here are the main enhancements that should be made according to the author:

The Dijkstra algorithm that can be found all over the internet in virtually every programing language should be integrated to optimize the total travel time of the drone. Moreover, this algorithm can also be used when introducing a second drone to the project. Then a drone-to-drone collision avoidance system will have to be set up. This system can benefit from the Dijkstra algorithm by creating reserved fly spaces for each drone and thus forcing the other drone to find its shortest path with this reserved space to circumvent.

Until now, the drone in the *'Dronicks'* program takes always the dricks from a single stockpile. An improvement that could spare minutes or even hours on the total project time would be to add several stockpiles. This integration is quite simple as the two only changes to deal with are:
Adding more stockpile position entries in the parameters page;
Adding a function in the path planning algorithm that for each drick compares the distance between it and every stockpile and assigns the closest one.

CONCLUSION

During this thesis an elementary BIM interface called 'Dronicks' was developed for modeling small size structures such as multi-level houses. The program was developed with aim of quickly designing concrete drone-compatible brick structures. Each wall being placed in the 3D model is instantly divided into drone-compatible elementary construction units which is unique in its kind. Furthermore, walls can be sequenced in an intermingled fashion so that they can form right-angled closed loop configurations. Openings can be added to walls and floors can be placed between consecutive levels. The program is modular as dimensions of the used bricks can be changed. The implemented program is intuitive, easy to use and requires no programming skills contrary to other CAD/BIM programs that would try to achieve the same resulsts.

If the first layer of artificial intelligence for a drone is its aptitude to react to the outer world by the means of sensors, the second layer of artificial intelligence for an autonomous building drone is the planning of its route beforehand. The program translates the 3D model into drone-compatible remote control instructions for every placed elementary construction unit. These instructions plan the main tasks the drone will have to execute upfront and the route it will take to do so. The chosen technique for avoiding collision with the already built parts of the structure is to always fly higher than the last built layer of the edifice.

One of the main objectives of the collective thesis developed at UCL under the supervision of Professor Latteur is to assure that drones can be more cost effective than masons. Therefore, it should be as autonomous as possible in its tasks, thus requiring less maintenance and monitoring and it should optimize the project time by minimizing the travel distances it carries out. To manage this, the position of the stockpile is being optimized, so that it is as close as possible to the center of gravity of the construction.

Another major objective is to ensure a flawless construction process. Thanks to a simulation tool in '*Dronicks*' it is possible to pre-visualize every step that the drone will carry out. This allows to eradicate complications before proceeding to the real mission. This simulation can also be used to test new specific placement approaches without damaging the drone by testing perilous actions.

Lastly, the communication between the *'Dronicks'* interface and the drone's communication system has already successfully been established. Although no real mission has yet been accomplished.

Figures list

Graphs list

REFERENCES

Images credits

[i1] "The great wall of China, China" courtesy of wallpaperstock.net, retrieved May 31st 2016.

[i2] "Pont du Gard, France" courtesy of campingmasderey.com, retrieved May 31st 2016.

[i3] "Pyramid of Cheops, Egypt" courtesy of sonyaandtravis.com, retrieved May 31st 2016.

[i4] "Complex structures built by animals: swallows building their nests" courtesy of sonyaandtravis.com, retrieved May 31st 2016.

[i5] "Complex structures built by animals: honeycomb" courtesy of thehoneybee.net, retrieved May 31st 2016.

[i6] "Construction with bricks placed by drones" courtesy of idsc.ethz.ch, retrieved May 31st 2016.

[i7] "*Tensile structure made of cable*" courtesy of flyingmachinearena.org, retrieved May 31st 2016.

[i8] "Construction of Cubic Structures with Quadrotor Teams" courtesy of roboticsproceedings.org, retrieved May 31st 2016.

[i9] "Fleye robot" courtesy of gofleye.com retrieved June 1st 2016.

[i10] "The platform" courtesy of idsc.ethz.ch retrieved June 1st 2016.

[i11] "Visual programming" courtesy of dynamoprimer.com retrieved May 19th 2016.

[i12] "Mobile applications" courtesy of movify.com, retrieved May 31st 2016.

[i13] "The foreman program" courtesy of idsc.ethz.ch, retrieved May 23rd 2016.

Literature credits

[1] "Frequently Asked Questions About the National BIM Standard-United States", *National BIM Standard - United States*, www.nationalbimstandard.org, retrieved April 24th 2016.

[2] "Biomimicry", *Biomimicry Europa association*, www.biomimicry.eu, 2015, retrieved June 1st 2016.

[3] ISO 13374-2:2007(en), § B.3.2.8, Condition monitoring and diagnostics of machines – Data processing, communication and presentation – Part 2: Data processing.

[4] DUNWIDDIE B., "The CSV file format", *CSVReader*, www.csvreader.com/csv_format.php, retrieved May 17th 2016.
[5] ISO/TR 14873:2013(en), §2.22, Information and documentation – Statistics and quality issues for web archiving.

[6] RAGGET D., "HTML 3.2 Reference Specification", *W3C*, www.w3.org/TR/REC-html32.html, retrieved may 17th 2016.

[7] "About ROS", *Open Source Robotics Foundation*, www.ros.org, retrieved April 24th 2016.

[8] "What Is An STL File?", *3D Systems Quickparts*, www.3dsystems.com/quickparts/learning-center/what-is-STL-file, retrieved May 17th 2016.

[9] INCLAN E. (*Florida International University*); Dr. PRASHANT J. (*Oak Ridge National Laboratory*), "Development of Pre-processing Software for Lattice Boltzmann Fluid Dynamics Solver", August 2012.

[10] ROTURIER, "SVG (Scalable Vector Graphic)", *Université Marne-La-Vallée*, retrieved May 18th 2016.

[11] SCHOLZ F., "WebGL", *Mozilla Developer Network*, www.developer.mozilla.org/en-US/docs/Web/API/WebGL_API, retrieved May 18th 2016.

[12] "Aircraft rotations", *NASA Official- National aeronautics and space administration*, www.grc.nasa.gov, retrieved April 24th 2016.

[13] ABERKANE I., "L'économie de la connaissance", www.youtube.com/watch?v=dM_JivN3HvI, retrieved May 19th 2016.

[14] BRETON J.-S.; LEPLAT J., "Feasibility Study For Drone-Based Manufacturing Of Architectural Structures" (Thesis), *Université Catholique de Louvain*, 2015.

[15] "Flying Machine Enabled Construction", *ETH Zurich,* www.idsc.ethz.ch/research-dandrea/research-projects/archive/flying-machine-enabled-construction.html, retrieved May 23rd 2016.

[16] "Building Tensile Structures with Flying Machines", *ETH Zurich*, http://www.idsc.ethz.ch/research-dandrea/research-projects/aerial-construction.html, retrieved May 23rd 2016.

[17] Q. LINDSEY , D. MELLINGER, and V. KUMAR , "Construction of Cubic Structures with Quadrotor Teams," *University of Pennsylvania*, Philadelphia, United States, 2011.

[18] "Fleye, your personal flying robot", *Go Fleye*, www.gofleye.com, retrieved June 1st 2016.

[19] "Flying Platform", *ETH Zurich,* www.idsc.ethz.ch/research-dandrea/research-projects/flying-platform.html, retrieved June 1st 2016.

[20] "ROS 101: INTRO TO THE ROBOT OPERATING SYSTEM", *Clear Path Robotics*, www.clearpathrobotics.com/2014/01/how-to-guide-ros-101/, retrieved may 18th 2016.

[21] "What is visual programming?", *The Dynamo Primer,* www.dynamoprimer.com/01_Introduction/1-1_what_is_visual_programming.html, retrieved may 19th 2016.

[22] HURT H., "Aerodynamics for naval aviators", University of Southern California, January 1965, www.faa.gov/regulations_policies/handbooks_manuals/aviation/media/00-80t-80.pdf, retrieved may 20th 2016.

[23] MACQUEEN J., "Some Methods for Classification and Analysis of Multivariate Observations", 5th Berkeley Symp. Math. Statist. Prob., Vol. 1, pp. 281-297.

APPENDICES

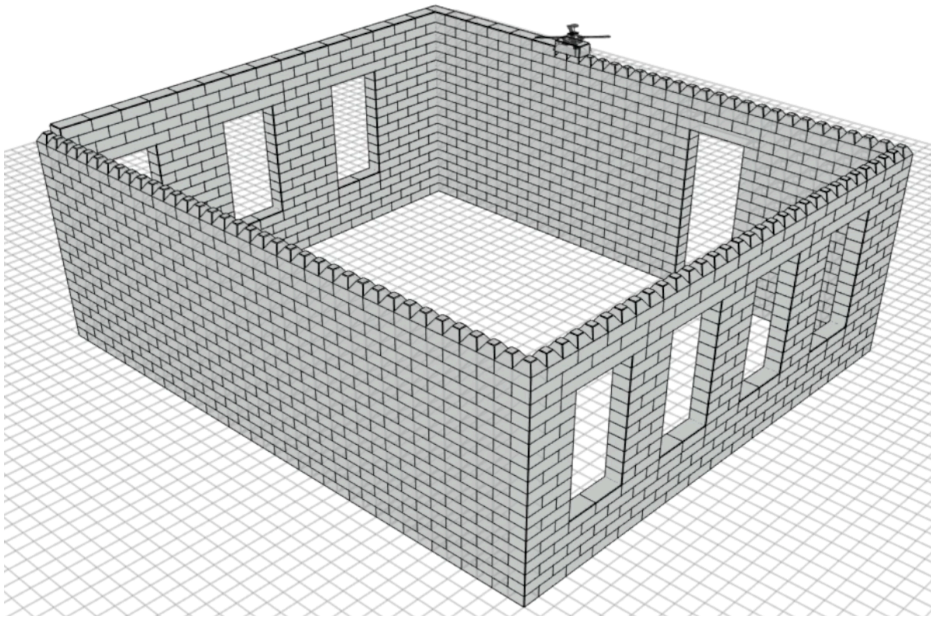


Figure 83 Extra: simulation page capture (a)

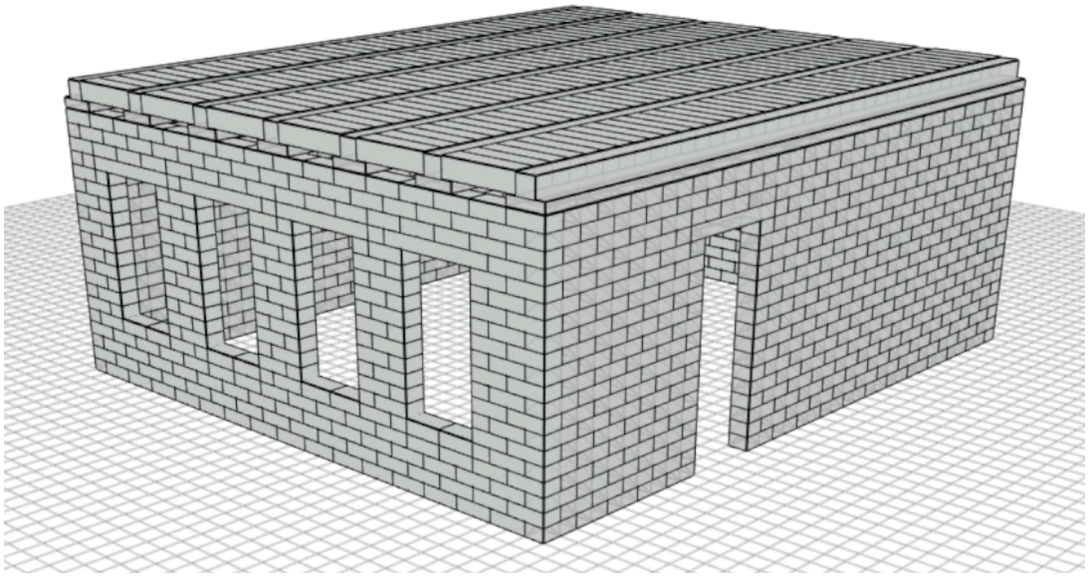


Figure 84 Extra: simulation page capture (b)

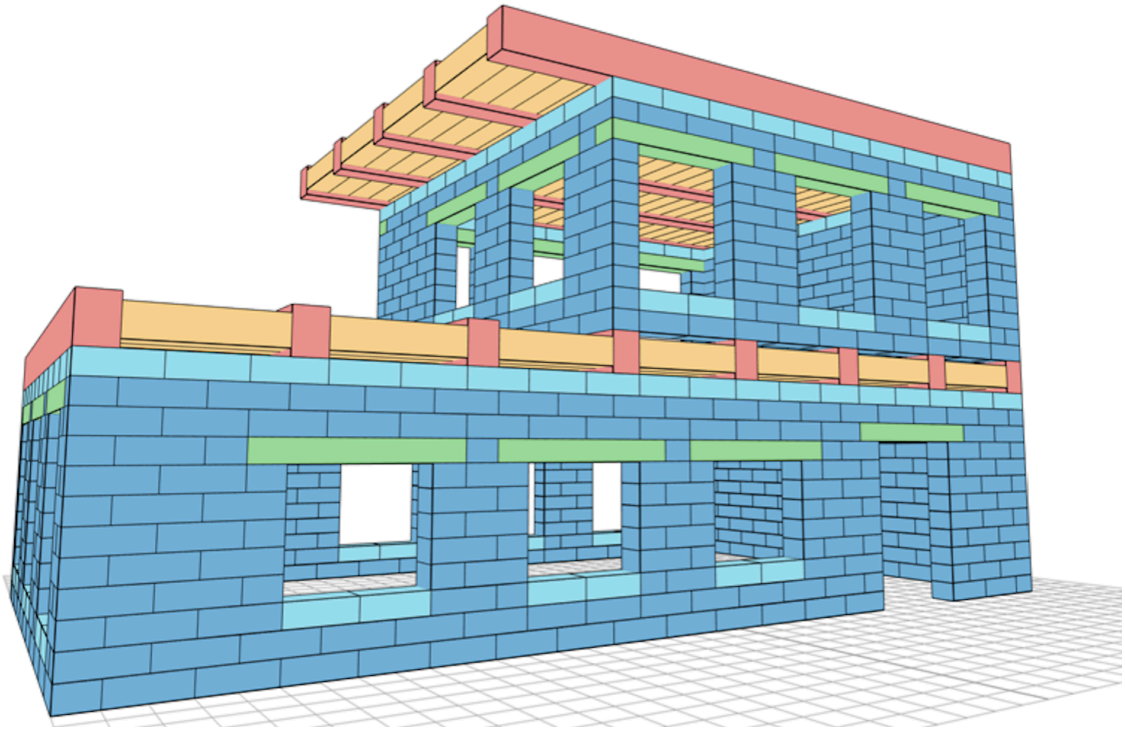


Figure 85 Extra: home page capture

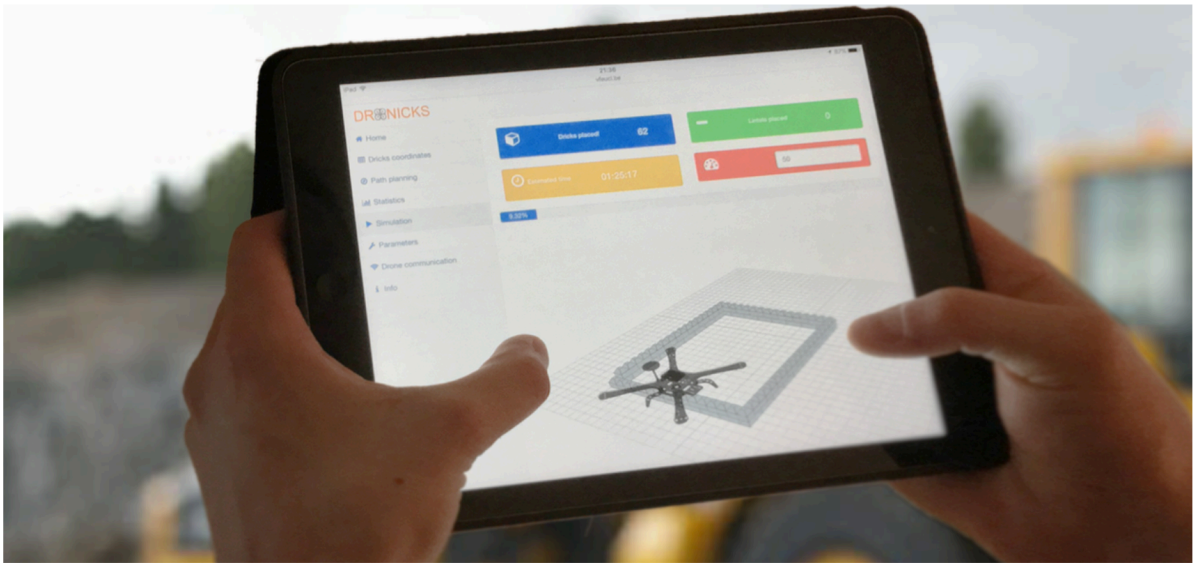


Figure 86 Extra: Dronicks on Ipad 2

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique **www.uclouvain.be/epl**