ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Space Technologies Minor Project

Collision-free positioning of optical fibers: from MOONS to Sloan

Author : Clément Camus

Supervisors : Denis Gillet & Jean-Paul Kneib

January 12, 2018

Contents

1 Introduction

The purpose of this project is the transfer of knowledge and technology from the MOONS trajectory computation and collision avoidance software, made by Dr. Laleh Makarem (1; 2; 3) & Dominique Tao, to the Sloan project, which role will be similar: compute trajectories of hundreds of actuators on a plane, with an hexagonal arrangement, to place fibers under targets, avoiding collision and deadlocks. Therefore, the two main goals are the update of the program for the Sloan case, and the improvement of the convergence rate (that is, the number of actuator able to reach their assigned target without any collision or deadlocks).

During this project, I mainly worked on the first goal. This report aim at explaining the work I have done during this semester, as well as the work still ongoing. To do so, a reminder explaining the differences between the two projects will be given. Then, Sections 3 and 4 will explain the change in the architecture of the software, especially the introduction of a program generating the actuators structures for the Sloan project as well as the centralisation of every parameters in a single text file. Section 5 shows the update of the target setting function and the fiber attribution, regarding to the major changes on the Sloan project, being the adding of new fibers on each ferules. In Section 6, a small script enabling the generation of target files to test the software will be presented. Section 7 shows the improvements implemented on the animation available at the end of the computation. Finally, Sections 8 and 9 will describe ongoing jobs as well as the rest of the tasks to be done to obtain a working software for the Sloan project. Changes in the code of the software can be seen in the Appendices.

2 From MOONS to Sloan : Reminder

This section aims at explaining the differences between the MOONS project and the Sloan one, to understand the path taken in the rest of the project.

MOONS (Multi-Object Optical and Near-infrared Spectrograph) is a design for the Very Large Telescope, the European Southern Observatory telescope facility. The fiber system is made of 1'000 actuators whose role is to position fibers on the surface of the telescope. Each actuator has two degrees of freedom as shown on Figure 1. At the end of the second arm, each ferule contained only one near-IR fiber. The actuator structure had an hexagonal shape, with several holes for camera and fiducial tools.

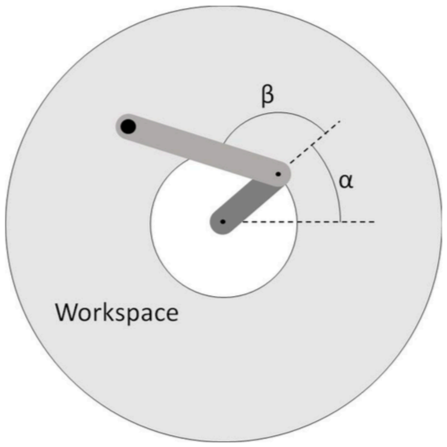


Figure 1: Actuator's design, with the two degrees of freedom being α and β angles. The arm displaced on the α angle is the α-arm, while the arm displaced on the β angle is the β-arm. The black dot at the end of the β-arm is the ferule, containing two to three fibers.

For the Sloan telescope, the arrangement, as seen on Figure 2, is also hexagonal. However, the system is made of 500 actuators, with a rectangle in the middle of the telescope. Some actuator slots are taken by fiducial tools, in red on Figure 2.

The main update from the Sloan system is the presence of two to three fibers on each ferule. As shown on Figure 2, each ferule contains both a visible and a calibration fiber, while 300 of the 500 motors contain an additional IR fiber. This structure allows a spatial coverage by IR fibers of 99.5% of the surface reachable by actuators, even if IR fibers are present only on three fifth of the actuators. A scheme of the new ferule is shown on Figure 3.

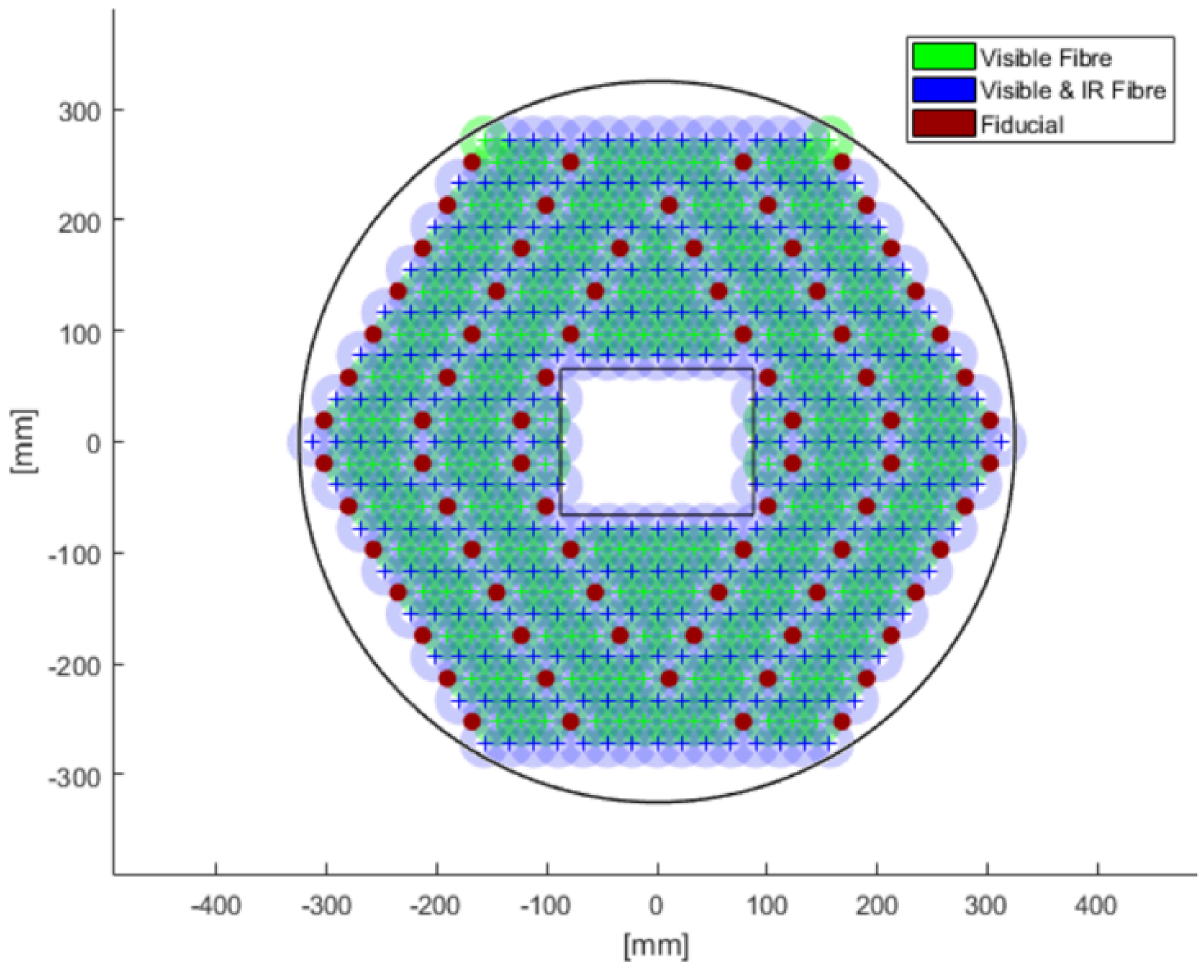


Figure 2: Actuators structure for the Sloan project.

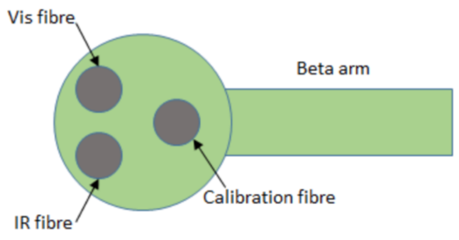


Figure 3: Ferule design for the Sloan project.

3 Sloan actuator structure

As said in Section 2, the Sloan telescope arrangement is made of 500 actuators, in an hexagonal shape. In the MOONS project, actuators types and position were listed in a cfg file. For time and practical reasons, this has been changed. A program called "circles.py" has been created to contain all actuators position and properties. The program is launched at each software's run, ensuring that, if a structure's parameter is changed (space between two parameters, fiber properties of several actuators etc.), the program will be automatically updated.

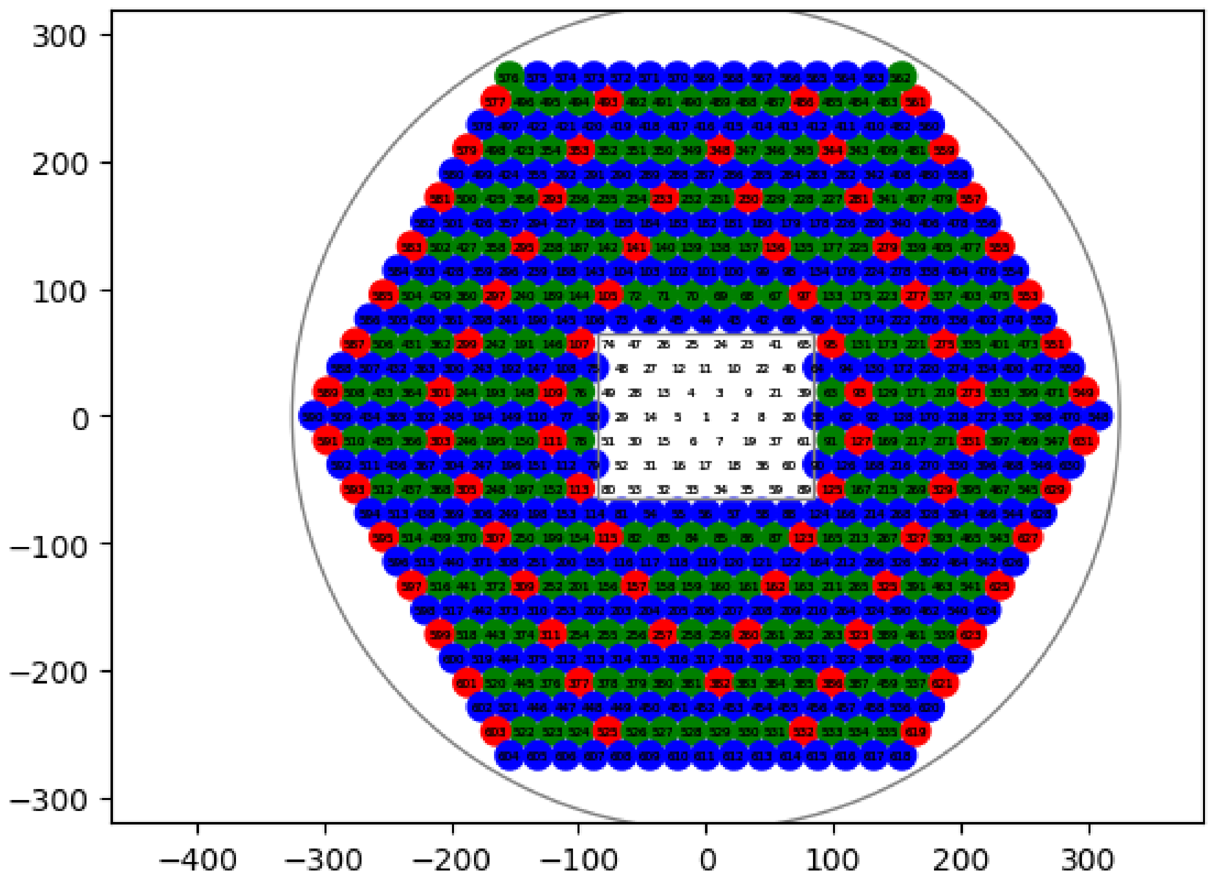


Figure 4: Actuators structure for the Sloan project.

The script creates the structure shown on Figure 4. One can compare it with the structure desired shown on Figure 2. The creation and indexation of actuators work as follow :

1. A first actuator is created on position (0, 0).
2. Then, using several functions available in the Appendix 1, a layer made of 6 actuators is created, beginning by the one on the right of the first actuator (actuator 2). The second

function creates an actuator (actuator 3) on the top left of the preceding actuator, the third function creates an actuator (actuator 4) on the left of actuator 3 and so on. As the first layer is made of 6 actuators, each function is applied once.

3. The second layer, made of 12 actuators, is made by applying the first function to create actuator 8. Then the second function is applied two times, creating actuators 9 and 10, and so on until the layer is finished.

4. The rest of the structure is made by applying Step 3 until a certain quantity of actuator has been created.

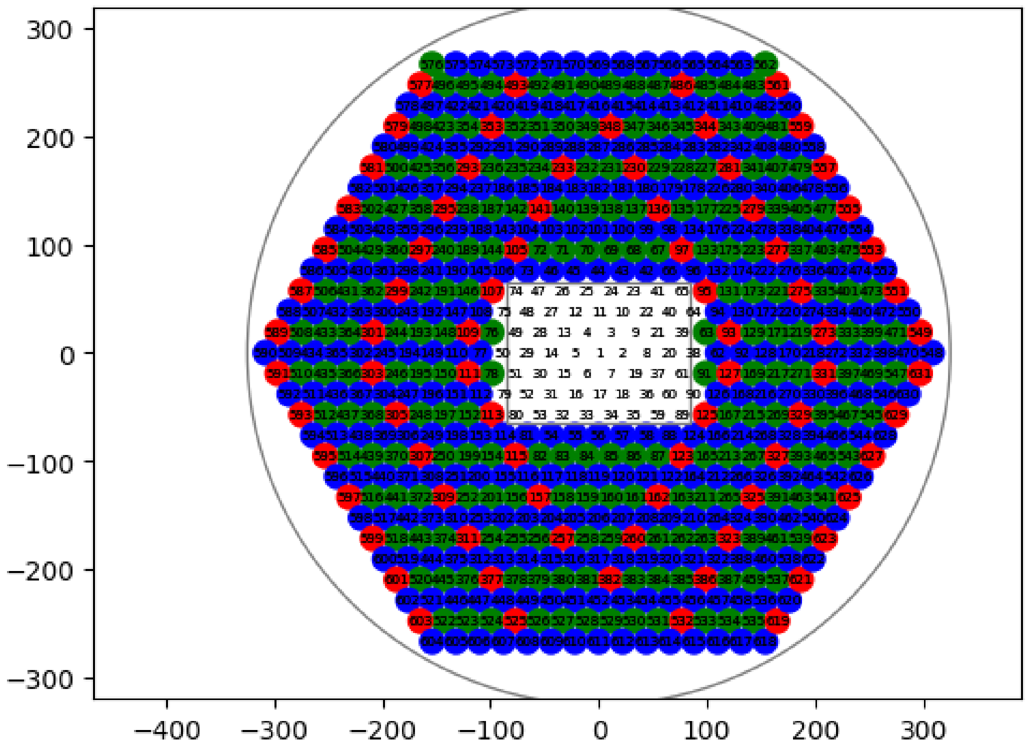
The indexation and construction of the structure is illustrated in the Appendix 1.

Then, the second part of the program classifies each actuator in one of the four following categories: empty, fiducial, IR or visible. Each actuator positioned in the rectangle in the middle of the telescope is set in the Empty category. Each actuator positioned on a fiducial slot as shown on Figure 4 is set in the Fiducial category. As shown on Figure 4, every actuator containing an IR fibre have similar y-coordinates, which is a certain real number times any discrete (positive, negative or null) number; any actuator which is not an Empty or a Fiducial having the aforementioned y-coordinate property is set as IR. Any other actuator, plus the two actuators on the top left and top right of the arrangement (see Figure 4) are set as Visible.

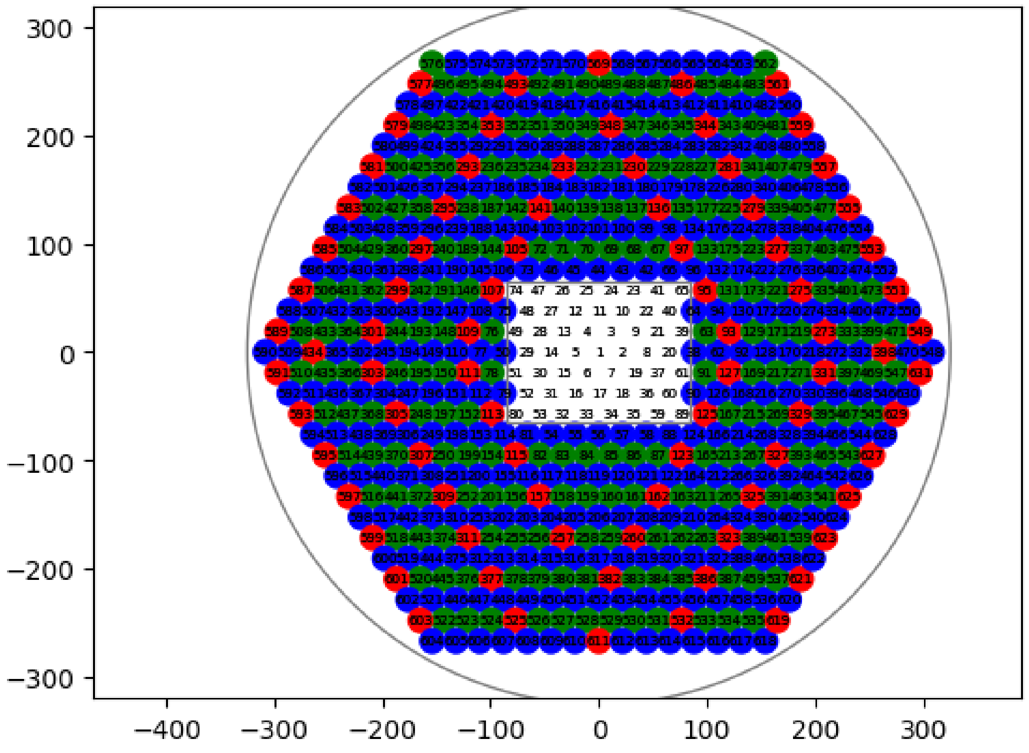
The script works for the structure set on December 2017. It is still possible to modify it easily; for instance :

- If the middle rectangle size is increased (or decreased), the only update to make is the addition (or subtraction) of actuator's indexes in the text file "empty.txt".
- If some actuators will be replaced by fiducial (or some fiducial removed), the only update is the addition (or subtraction) of actuator's indexes in the text file "fiducial.txt".
- If the dimensions of the structure are changed (space between actuators), the program updates automatically the list of actuators set as IR.
- If new actuators have to be added on the limits of the existing ones to approach the circular shape, only two steps are required: adding new actuator layers, and adding a new requirement such as the radial coordinate of the actuator has to be inferior to the radius of the telescope (or manually classifying the actuators created outside of the focal plane as Empty).

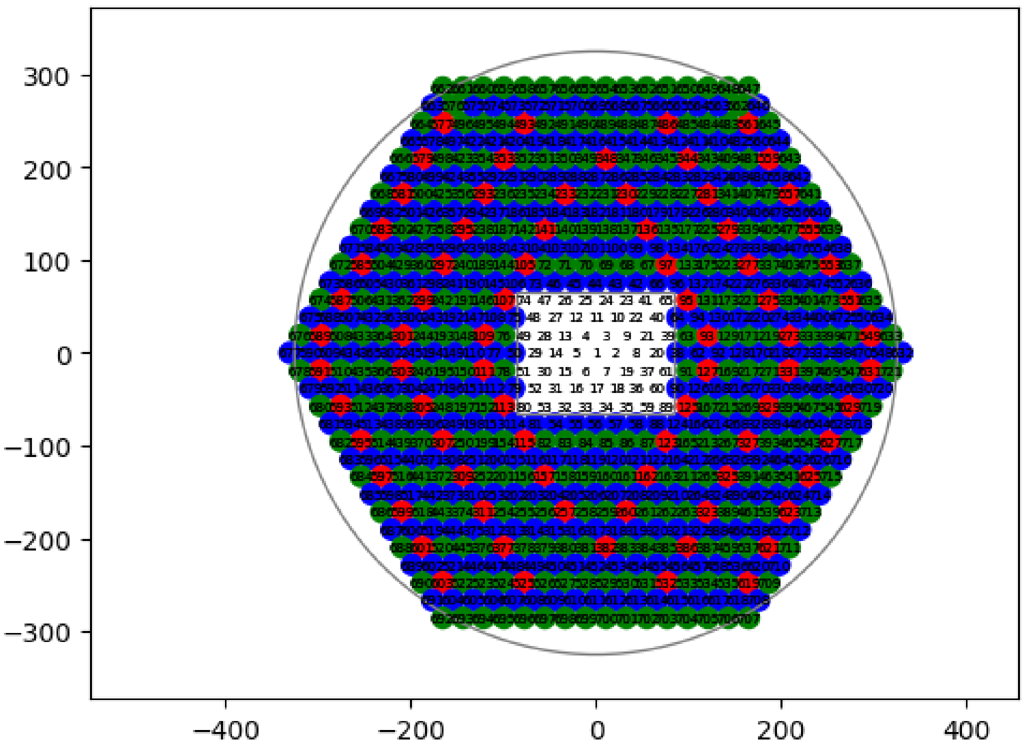
An example of each one of those adjustment is shown on Figure 4.


300
200
100
0
−100
−200
−300
−400
−300
−200
−100
0
100
200
300

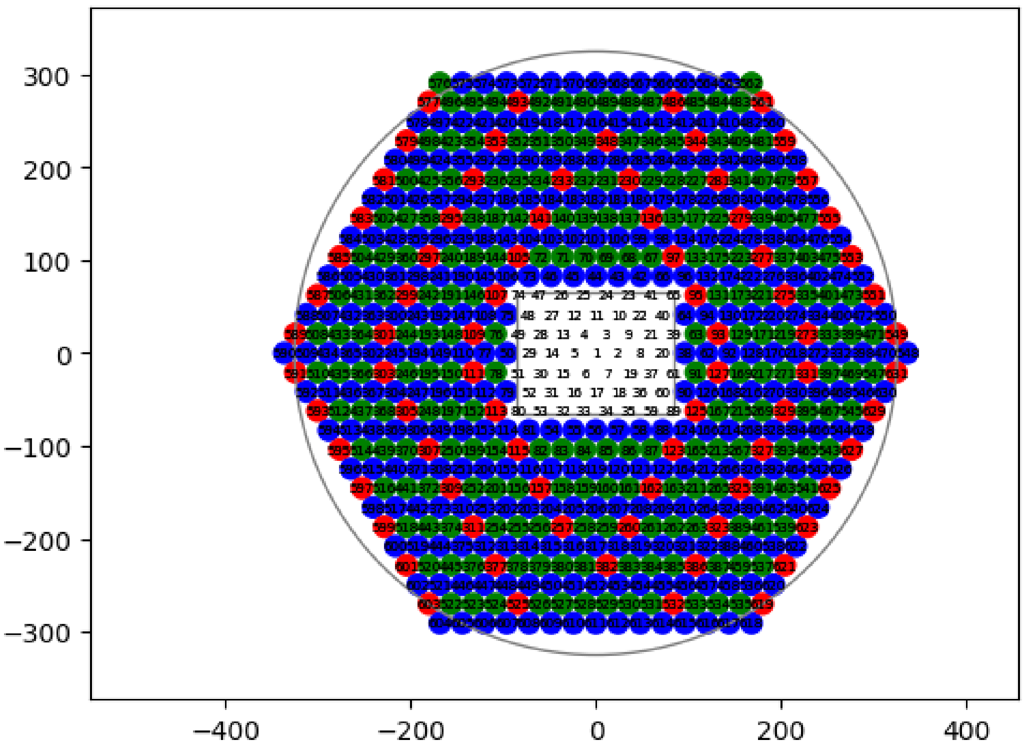
(a)


300
200
100
0
−100
−200
−300
−400
−300
−200
−100
0
100
200
300

(b)



(a)



(b)

Figure 4: Examples of adjustments made on the actuator's structure only by modifying text files. (a) Addition of six indexes (38, 50, 64, 75, 79, 90) in the file empty.txt. (b) Addition of four indexes (398, 434, 569, 611) in the file fiducial.txt. (c) Addition of a layer by modifying an entry in the file parameters.txt. (d) Increase of the separation length between actuators by modifying an entry in the file parameters.txt.

4 Parameters

Previously, the constants needed for the program to be launched (telescope's and actuators dimensions, DNF parameters etc.) were scattered all over the software. Thus, any attempt to modify one of those parameter was preceded by a research for the location of the variable definition in one of the programs. For instance, the previous software contained, on three different programs, the codes in Figures 9, 10 and 11 in Appendix 2. Moreover, a large part of those parameters were not at the very beginning of each program, but sometimes after the 100th code line, making the research for a variable tedious.

To simplify the change of the variables, each one of them have been moved in a text file called "parameters.txt". The program "parameters.py" then looks inside this text file the value of the parameters, and gives those values to the rest of the software. This allows an efficient modification of any parameter and minimises the hardcoding problem.

5 Target assigning function update

The target assigning function's purpose (set_target in fps_shared.py) is to propose a target for each actuator. If the target fulfills certain criteria, especially on its position (the actuator must be able to reach the target), then the actuator is assigned to the target. It means the actuator's ferule's center will be moved to the target position.

Two updates had to be made: first, the center of the ferule is not anymore the position of the fiber; as shown in Section 2 on Figure 3, the actuator has to be placed under the desired fiber. Furthermore, we have to be able to ask in advance for each target to be seen by a visible fiber, a calibration fiber or an IR fiber. The first problem will be treated in Section 5.1, while the second one will be covered in Section 5.2.

5.1 set_target

As said earlier, the function set_target is used to assign a target to a certain actuator. If the target was it the range of action of the actuator, then the actuator was assigned to this target. Otherwise, the assignment stoped. In addition to assigning the target to the actuator, the function gave the actuator's ferule's center its destination, which was set at the position of the target. As said in the Reminder, the position of the fiber is not anymore at the center of the ferule, but at one of three points near from the ferule, as shown on Figure 3. This situation imposed two tasks:

1. Knowing and setting in advance which fiber the target has to be surveyed with.
2. Changing the actuator desired position depending on the fiber used.

The first task had been solved the following way: in addition to the position, parity and priority of each target, the target file now requires a new column to be filled with a number (as for the priority) between 1, 0 and -1 depending on the chosen fiber. The correspondance is shown in Table 1. Former and new target file structures, with two example lines, are shown in Tables 2 and 3.

The second tasks required more reasoning. The following ideas were studied:

1. Drifting the position of the target depending on the choice of the fiber.
2. Moving each motors to the target (under the center), and then ask a small displacement to move the fiber under the target.
3. Changing the coordinates of the wished position for each motor, depending on the wished fiber.

The first proposition was removed, as it adds confusion in the software between the real position of the target and the virtual one, besides visualisation problems.

The second one was also removed, as it doesn't take into account deadlocked actuators, and can lead to collisions during the part where each actuator move the ferule to the good position. Moreover, the change in the position depends on the actual position of the actuator before the correction.

The third proposition has been chosen for its simplicity: after setting the destination for the actuator's ferule's center, a correction is added, in the referential of the actuator, depending on the fiber used. The code added can be seen in the Appendix 3. Notwithstanding, it appears this technique does not work in our case: if the two degrees of freedom available would have been the radius and the angle, the modification of the target's position in the center's ferule's referential would have been unique. However, as we have two angles as degrees of freedom, there exists an infinite number of combination (α, β) respecting the correction aforementioned: first, there is two set of combination depending on the parity (for a fixed α, both β and $\frac{\pi}{2} - \beta$ give the same solution), but furthermore, targets randomly generated are generally positioned on spots where the solution couple (α, β) is not unique, but continuous. Hence, the correction to the actuator's position will depends on the final (α, β), coming down to solution 2.

Indicator	Fiber used
1	IR
0	Visible
-1	Calibration

Table 1: Correspondance between the target file filling and the fiber to be used.

R actuator	θ actuator	R target	θ target	Arg 1 parity	Arg 2 parity	Priority
25.2587	30	32.6135	358.578	0	1	1
25.2587	90	25.4647	67.2179	0	1	1

Table 2: Former target file structure with two lines as example.

R actuator	θ actuator	R target	θ target	Arg 1 parity	Arg 2 parity	Priority	Fiber
25.2587	30	32.6135	358.578	0	1	1	1
25.2587	90	25.4647	67.2179	0	1	1	-1

Table 3: New target file structure with two lines as example.

5.2 Fiber attribution

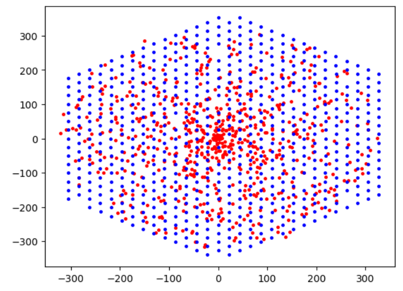
The next task will be to use the information on each actuator from the Sloan structure from Section 3. More precisely, the goal will be to give the autorisation to actuators to be set on targets only if they are able to. Thus, the following modifications have been made :

- If an actuator is an Empty one (empty rectangle for Sloan) or a Fiducial one, target is never set (Set_target returns False).
- If the target needs an IR fiber, but the proposed actuator only has visible and calibration, target is not set (Set_target returns False).
- If the target needs an IR fiber, and the proposed actuator owns an IR fiber, nothing changes.
- If the target needs a visible or a calibration fiber, nothing changes.

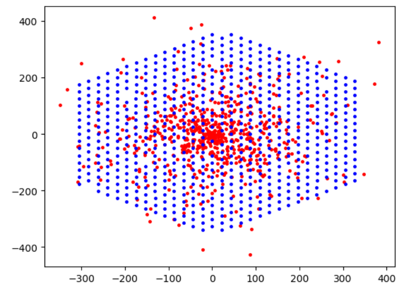
The code added to ensure it can be seen in the Appendix 3.

6 Target File Generator

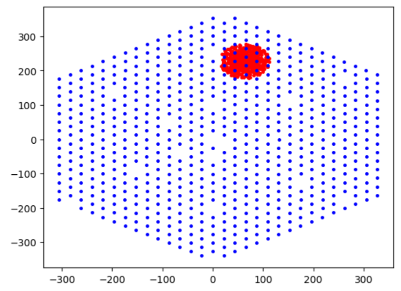
Once the software is ready, it must be tested on several target files, first to be sure it's working properly, and then to work on the convergence rate's improvement. For this purpose, a program called "target_generator" has been created. It is able to create five types of target distribution : Random, Random with a normal distribution, Focused, Focused with a normal distribution, and Homogeneous. An example of each one of those is available on Figure 5.



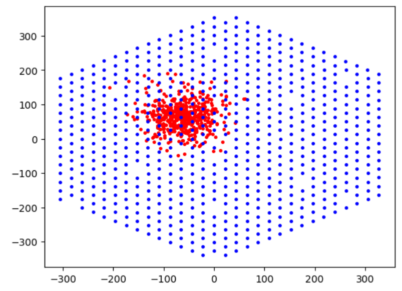
(a) Example of target file with a random distribution.



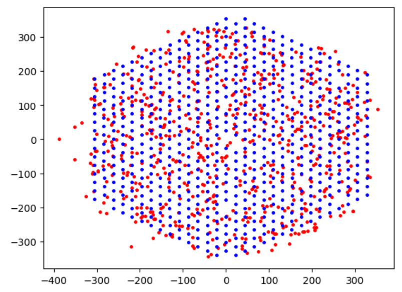
(b) Example of target file with a random and normal distribution.



(c) Example of target file with a distribution focused on a random point of the telescope.



(d) Example of target file with a normal distribution focused on a random point of the telescope.



(e) Example of target file with an homogeneous distribution.

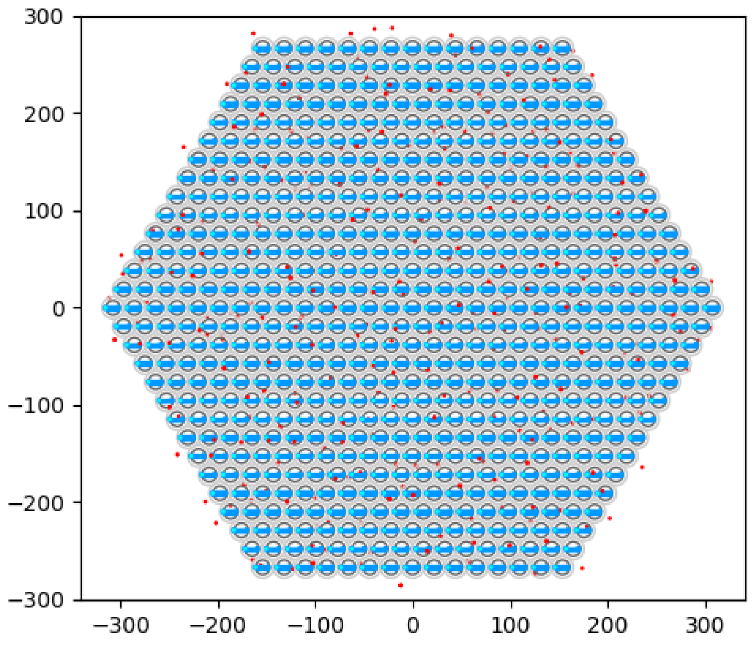
Figure 5: Examples of target files generated with the target_generator.py program. Targets are shown in red, while actuators are shown in blue. Actuator positions come from the former software (used for MOONS), and therefore contains its specificity, that is to say some holes where fiducial tools were positioned.

The homogeneous is the most similar to the MOONS's target files, as it places a target close to each actuators. Therefore, it will also be the one with the highest convergence rate.

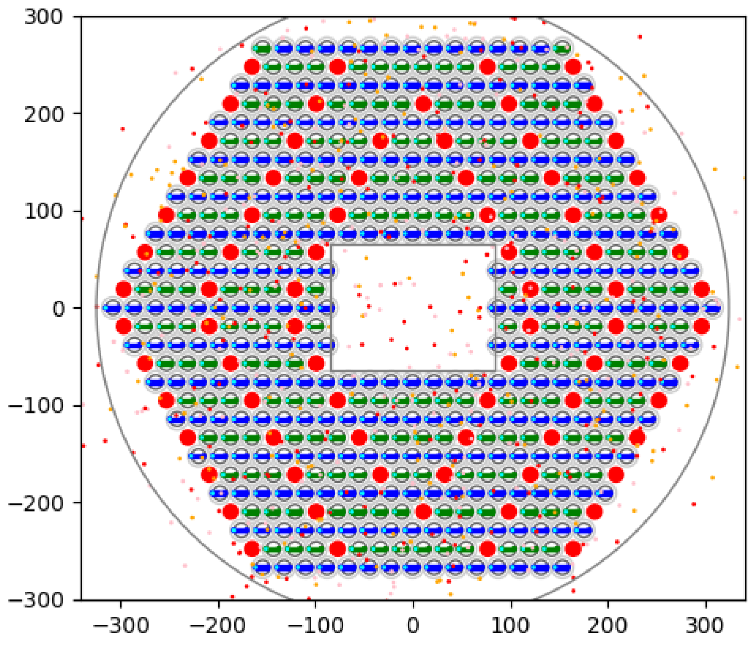
7 Visualisation

Figure 6 shows two pictures of the the animation available at the end of the computation, for the former and the new software. The improvements are the following:

- The former program only showed the targets surveyed by an actuator. Now, every targets are shown.
- As mentioned in Section 5.2, each target are set on one of the three fibers available. IR targets are shown in orange, Visible ones in red and Calibration ones in pink. Any target's color change requires hardcoding, however the operation is straightforward; the code to change is given in Appendix 5.
- Actuators have also been colorised depending on their functions, respecting the color code on Figures 2 and 4 : empty one are transparent, fiducial ones are red, those with an IR fiber are in blue while the others are in green.



(a) Former animation picture.



(b) New animation picture.

Figure 6: Former and new pictures of the animation available at the end of the computation. Explanations are given in Section 7.

8 To-do list

This Section enumerates the remaining main tasks to be done to obtain a working software.

The first task to accomplish is, as described in Section 5.1, the correction on the target's position for each actuator depending on the fiber used. This was the main task of this project, however I personally searched in the wrong direction and discovered the mistake too late.

Another point to set is on the collision-free update for the Sloan project, especially on the behaviour of fiducial tools on the DNF for its actuator's neighbour. Moreover, the change in the DNF parameters have to be tested so no collision occurs during any process, while verifying that the number of deadlocks stays low.

Some improvements can me made but are not essential for the operation of the software. Indeed, the convergence rate was still improvable as on the last version. In the same way, the prioritisation can be still improved, to be sure a priority target will be surveyed.

The last category of tasks are the improvement of the code in the program: indeed, there is still some feature only used for the MOONS project remaining on the Sloan version. This can slow down the program, lead to mistakes and above all will decrease the understanding of the code.

9 Conclusion

The transition from the MOONS software to the Sloan needed two main modifications: the first one was on the structure of the actuators moving on the focal plane, which was different for two projects; the second one concerned the fibers used for the survey, as Sloan's ferules will contain to to three fibers.

The first adjustment has been completed so the actuator's structure respects the one discussed on the Preliminary Study Report(4), as shown in Sections 3 and 7. Furthermore, the program has been constructed so that some modifications where possible without hard-coding, and so that the program generating the structure also contained informations on the fibers available.

The second point has not been completed yet: a solution could not be found yet to correct the target's destination of actuators, so that the desired fiber is positioned under the target surveyed (see Section 5). At least, some possibilities, for instance relocating target's position, could be excluded. The main alternative to work on is asking each actuator to move at the end of the computation to have the fiber on target's position, but it will constitute two challenges, one on the trajectory computation, and another one on the collision-free part of the software.

Several components of the software have been improved: the structure has been modified to be able to adjust parameters easily (Section 4), and a new architecture has been designed (see Appendix 4), a program has been created to generate targets file to test the software (Section 6), and to finish, the visualisation has been modify to take into account the features specific to the Sloan project.

I personally enjoyed working on this project, as it was the first time I was working on the development of a software. The software is quite long to understand in depth, so some minor adjustments took some time, but then I could work on python's features I never worked on before (animations, user interface etc.). I would like to thank Prof. Kneib who gave me the opportunity to work on this project, Prof. Gillet for the freedom I had during the semester.

References

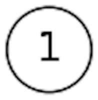
[1] Laleh Makarem, *Decentralized multi-robot coordination in crowded workspaces*, 2015.

[2] Laleh Makarem, Jean-Paul Kneib, and Denis Gillet, *Collision-Free Coordination of Fiber Positioners in Multi-object Spectrographs*, 2015.

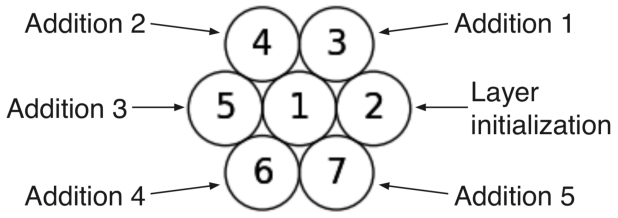
[3] Laleh Makarem, Jean-Paul Kneib, Denis Gillet, Hannes Bleuler, Mohamed Bouri, Laurent Jenni, Francisco Prada, and Justo Sanchez, *Collision avoidance in next-generation fiber positioner robotic systems for large survey spectrographs*, Astronomy & Astrophysics, 2014.

[4] Jean-Paul Kneib, Mohamed Bouri, Denis Gillet, Philipp Horler, Luzius Kronig, *SDSS-5 Fibre Positioner Preliminary Study Report, Second Version*, 2017.

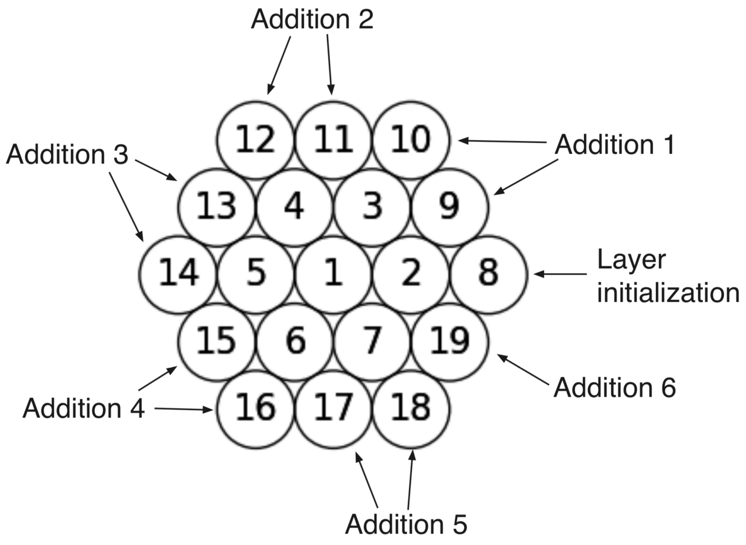
10 Appendix 1: Sloan actuators structure



(a) Initialisation.



(b) First actuator's layer.



(c) Second actuator's layer.

Figure 7: Actuators structure explanation: the initialisation creates an actuator at position (0, 0), layer initialisation then creates an actuator on the right, and each addition function creates an actuator close to the last actuator created. i^{th} layer is built by initialising the layer, repeating addition functions 1-5 i times, and finishing by repeating addition function 6 $(i-1)$ times.

```
position.append((0, 0))

def initialisation(num_circle):
    position.append((2*R*num_circle, 0))
    return

def addition_1():
    position.append(tuple((map(operator.add, position[(len(position)1)], (R
        ↪ , R * 3 ** 0.5)))))
    return

def addition_2():
    position.append(tuple((map(operator.add, position[(len(position)1)],
        ↪ (2*R, 0)))))
    return

def addition_3():
    position.append(tuple((map(operator.add, position[(len(position)1)], (R
        ↪ , R * 3 ** 0.5)))))
    return

def addition_4():
    position.append(tuple((map(operator.add, position[(len(position)1)], (R
        ↪ ,  R * 3 ** 0.5)))))
    return

def addition_5():
    position.append(tuple((map(operator.add, position[(len(position)1)],
        ↪ (2*R, 0)))))
    return

def addition_6():
    position.append(tuple((map(operator.add, position[(len(position)1)], (R
        ↪ , R * 3 ** 0.5)))))
    return

def repeat_function(num_circle, function):
    for _ in range(num_circle):function()
    return

def layer_prod(layer):
    initialisation(layer)
    repeat_function(layer, addition_1)
    repeat_function(layer, addition_2)
    repeat_function(layer, addition_3)
    repeat_function(layer, addition_4)
    repeat_function(layer, addition_5)
    repeat_function((layer  1), addition_6)
    return
```

Figure 8: Code generating the hexagonal structure shown Figure 4. The first line creates the actuator at position (0, 0) (see Figure 7a). Effects of functions initialisation and addition 1 to 6 are shown on Figure 7b. Upper layers (see Figure 7c) are built using functions repeat_function and layer_prod.

11 Appendix 2: Parameters

```
INS_POS_LENGTH1 = 8.0
INS_POS_WIDTH1 = 8.0
ALPHA_LIMITS = [180.0, 180.0]
ALPHA_DEFAULT = 0.0
ALPHA_RPM_LIMITS = [0.73, 2.78]
```

Figure 9: Former fps_share.py program (code lines 124-128).

```
# zone where the positioner is affected by the repulsive force of other positioners
STATUS_OFF_ZONE_INFLUENCE = 3.5
STATUS_ON_ZONE_INFLUENCE = 7.5

LIMITE_FOR_NOISE = 0.08

CSTE_FORCE=6.2
```

Figure 10: Former pa_dnf.py program (code lines 138-144).

```
sim_length = 210
dt = 0.25
max_speed_alpha = 2*math.pi*2.78/60
min_speed_alpha = 2*math.pi*0.73/60
max_speed_beta = 2*math.pi*3.75/60
min_speed_beta = 2*math.pi*0.98/60
```

Figure 11: Former path_generator.py program (code lines 39-44).

```
# PARAMETERS FILE

# Simulation constants
sim_length = 250
dt = 0.25
max_speed_alpha = 2*3.141592653*2.78/60
min_speed_alpha = 2*3.141592653*0.73/60
max_speed_beta = 2*3.141592653*3.75/60
min_speed_beta = 2*3.141592653*0.98/60

# Dimensions
INS_POS_LENGTH1 = 7.4
INS_POS_WIDTH1 = 8.0
```

Figure 12: parameters.txt file.

```python
import numpy as np

parameters = open('parameters.txt', 'r')
parameters = parameters.read().split()

def look_for(parameter):
    for i in range(0, len(parameters)):
        if parameters[i] == parameter:
            parameter = eval(parameters[i+2])
            break
        else:
            continue

    return parameter

sim_length = look_for('sim_length')
dt = look_for('dt')

max_speed_alpha = look_for('max_speed_alpha')
min_speed_alpha = look_for('min_speed_alpha')
max_speed_beta = look_for('max_speed_beta')
min_speed_beta = look_for('min_speed_beta')
```

Figure 13: parameters.py program. This program opens the parameters.txt file, and look for parameters by their names using the function look_for (the name inside the function must be exactly the same). When the parameter is found, the function returns the value after the equal sign.

```
INS_POS_LENGTH1 = param.INS_POS_LENGTH1
INS_POS_WIDTH1 = param.INS_POS_WIDTH1
ALPHA_LIMITS = [param.ALPHA_TRAVEL_MIN, param.ALPHA_TRAVEL_MAX]
ALPHA_DEFAULT = param.ALPHA_DEFAULT
ALPHA_RPM_LIMITS = [param.ALPHA_RPM_MIN, param.ALPHA_RPM_MAX]
```

Figure 14: New fps_share.py program.

```
# zone where the positioner is affected by the repulsive force of other positioners
STATUS_OFF_ZONE_INFLUENCE = param.STATUS_OFF_ZONE_INFLUENCE
STATUS_ON_ZONE_INFLUENCE = param.STATUS_ON_ZONE_INFLUENCE
LIMITE_FOR_NOISE = param.LIMITE_FOR_NOISE
CSTE_FORCE = param.CSTE_FORCE
```

Figure 15: New pa_dnf.py program.

```
sim_length = param.sim_length
dt = param.dt
max_speed_alpha = param.max_speed_alpha
min_speed_alpha = param.min_speed_alpha
max_speed_beta = param.max_speed_beta
min_speed_beta = param.min_speed_beta
```

Figure 16: New path_generator.py program.

12 Appendix 3: Set_target update

```
# Determine the local (wrt positioner centre) coordinates for the
# target.
self.x_fibre_local = self.x_fibre_focal  self.x_centre_focal
self.y_fibre_local = self.y_fibre_focal  self.y_centre_focal
(self.r_fibre_local, self.theta_fibre_local) = util.cartesian_to_polar(
   ↪ self.x_fibre_local, self.y_fibre_local)

if fiber == 1:
       (self.r_fibre_local, self.theta_fibre_local) = (self.r_fibre_local
          ↪ + param.R_CORRECTION_IR, self.theta_fibre_local + param.
          ↪ THETA_CORRECTION_IR)
elif fiber == 0:
       (self.r_fibre_local, self.theta_fibre_local) = (self.r_fibre_local
          ↪ + param.R_CORRECTION_VISIBLE, self.theta_fibre_local + param.
          ↪ THETA_CORRECTION_VISIBLE)
elif fiber ==  1:
       (self.r_fibre_local, self.theta_fibre_local) = (self.r_fibre_local
          ↪ + param.R_CORRECTION_CALIBRATION, self.theta_fibre_local +
          ↪ param.THETA_CORRECTION_CALIBRATION)
```

Figure 17: Code modifying the position targeted by the actuator, as function of the fiber used.

```
positioner_IR = False
positioner_fiducial = False
positioner_empty = False

for item in circles_test.IR:
        if positioner == item:
                positioner_IR = True

for item in circles_test.fiducial:
        if positioner == item:
                positioner_fiducial = True

for item in circles_test.empty:
        if positioner == item:
                positioner_empty = True

if fiber == 1 and positioner_IR == False:
        return False

if positioner_fiducial == True:
        return False

if positioner_empty == True:
        return False
```

Figure 18: Code ensuring that Empty and Fiducial can not be set on any target, and that only IR actuator can be set on target desired to be surveyed with an IR fiber.

13 Appendix 4: Software architecture

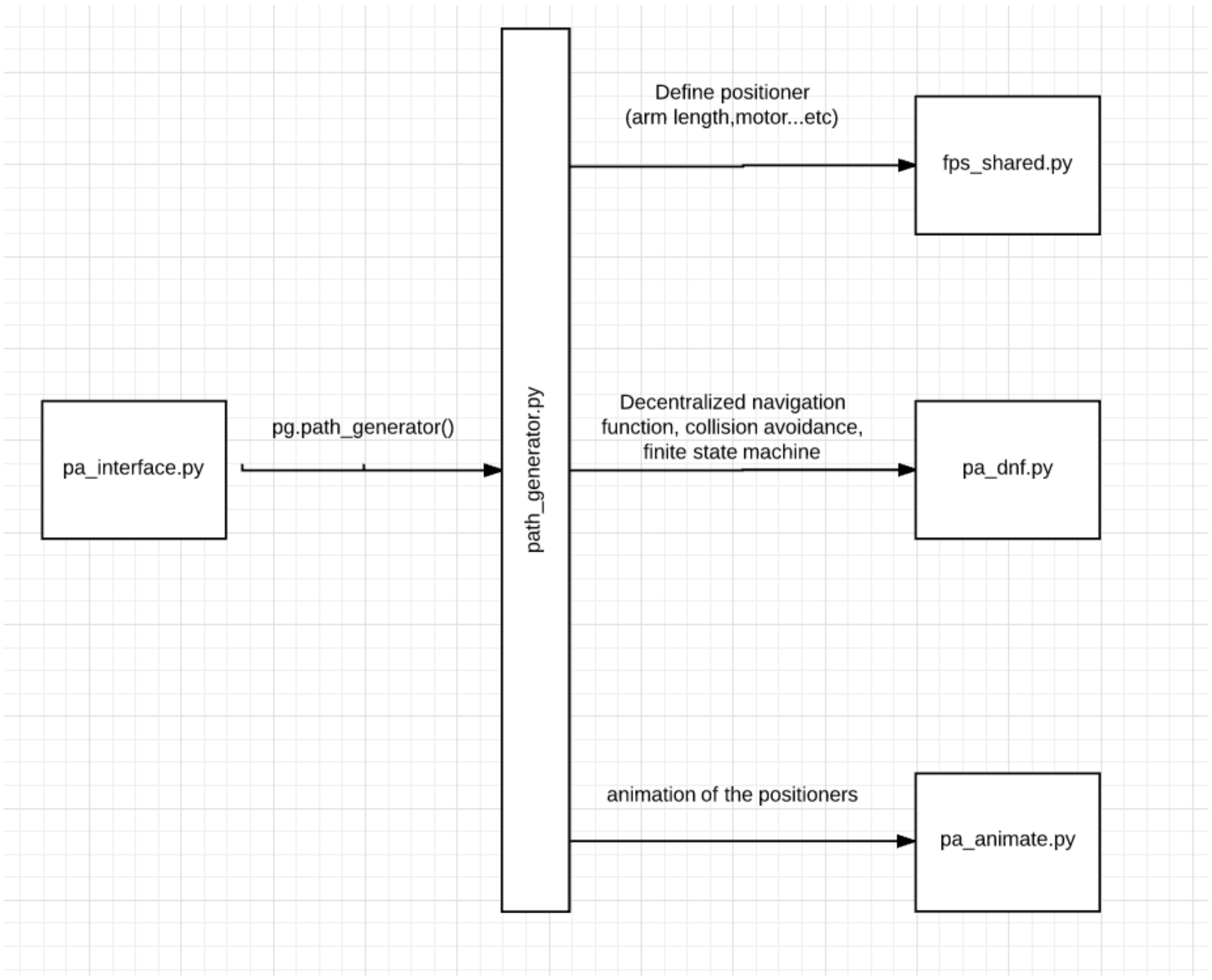


Figure 19: Former software's architecture.

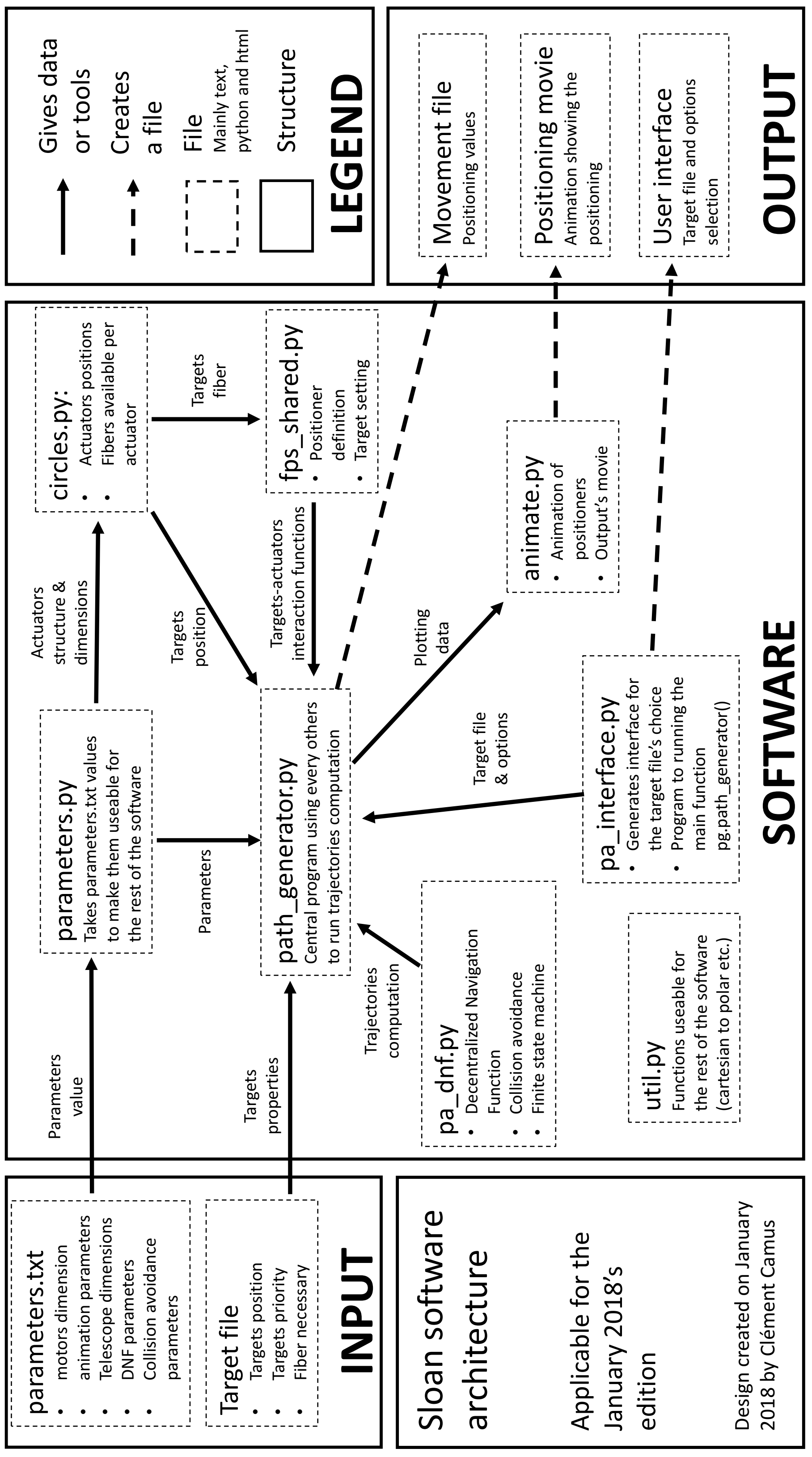

INPUT
parameters.txt
motors dimension
animation parameters
Telescope dimensions
DNF parameters
Collision avoidance parameters
Target file
Targets position
Targets priority
Fiber necessary
Sloan software architecture
Applicable for the January 2018's edition
Design created on January 2018 by Clément Camus
SOFTWARE
Parameters value
Targets properties
parameters.py
Takes parameters.txt values to make them useable for the rest of the software
Parameters
path_generator.py
Central program using every others to run trajectories computation
Trajectories computation
pa_dnf.py
Decentralized Navigation Function
Collision avoidance
Finite state machine
util.py
Functions useable for the rest of the software (cartesian to polar etc.)
Actuators structure & dimensions
Targets position
circles.py:
Actuators positions
Fibers available per actuator
Targets fiber
fps_shared.py
Positioner definition
Target setting
Targets-actuators interaction functions
Plotting data
animate.py
Animation of positioners
Output's movie
Target file & options
pa_interface.py
Generates interface for the target file's choice
Program to running the main function pg.path_generator()
LEGEND
Gives data or tools
Creates a file
File
Mainly text, python and html
Structure
OUTPUT
Movement file
Positioning values
Positioning movie
Animation showing the positioning
User interface
Target file and options selection

Figure 20: New software architecture.

14 Appendix 5: Visualisation

```
    for i, line in enumerate(fr2):
        x1, x2, x3, x4, x5, x6, x7, x8 = map(float, line.split())
        color_target = ''
        if x8 == 1: color_target = 'orange'
        elif x8 == 0: color_target = 'red'
        elif x8 == 1: color_target = 'pink'

        targetcircle = plt.Circle(util.polar_to_cartesian(x3,x4), radius=1,
            ↪  color=color_target, fill=True)
        subfig.add_patch(targetcircle)
```

Figure 21: Code assigning a color to each targets on the animation, depending on the fiber required for the survey.