ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

SEMESTER PROJECT

Building a Robotic Fly

Author:
Loic TIBERGHIEN

Teacher:
Pavan RAMDYA

Doctorant:
Victor LOBATO RIOS

January 4, 2019

Abstract

The aim of this project consists in designing a robotic fly, using as model the *drosophilia*. Two main steps are involved : obtaining the biological model using a CT scanner and then designing a robotic replicate of this model.

The first step was done by a student during the previous semester and allowed to extract a 3D mesh out of the scanner's data. Unfortunately, this 3D mesh presented undesired objects around the body of interest. The first task was to erase these artifacts semi-automatically during the processing of the images. The processing was done over the frames of the CT scan, before the application of the marching cube algorithm. Two major steps were included :

- Cleaning the frames where the bubble is not connected to the body using the *cv2.findContours* and the *cv2.drawContours* functions to perfectly erase the artifact.
- Cleaning the frames where the bubble is connected to the body by erasing a circle approximate of the bubble using the *cv2.HoughCircles* function to detect the artifact. The resulting image is smoothed using an opening operation.

This was an iterative process which had to be done for each artifact. The result had then to be merged using the simple command *cv2.bitwise_and* on every frame of each result.

The second part of the project concerns the design of the exoskeleton of the front legs of the fly in blender. This is done by following this routine :

1. Obtaining the 3D mesh corresponding to the front leg using Lucas Gautier's code. The *leg3* mesh was used.
2. Isolating each segment of the leg : **coxa, femur, tibia** and **tarsus.**
3. Simplifying the mesh using the **shrinkwrap** modifier in blender.
4. Hollowing the shape and giving it a thickness, using the **boolean** modifiers.
5. Adding technical components to the resulting shell : **bone links** and **magnet holders**.
6. Splitting the shell in two halves for the **coxa**, the **femur** and the **tibia**.

These designs were then printed using the *form2* 3D printer from *formslab*.

The last task consisted in building a *state of the art* for sensors that could be used to implement tactile sensory feedback to the robotic *drosophilia*. Several solutions were identified, mimicking either the *bristles* of the *campaniform sensilla* of the fly. The most relevant ones are capacitive contact sensors and "wrapable" pressure sensitive surfaces.

Contents

Part I

Image Processing

Introduction

The first part of the project concerned the processing of data sets obtained with a CT scanner. Indeed, using this machine, it is possible to obtain a *.tif* image stack (basically *"slices"* of the analyzed object), which can in turn be processed to obtain a 3D visualization of the body of interest.

A method to obtain this 3D visualization was developed by another student (Lucas Gautier) during the previous semester. This method was a python code and had great results. It was applied to two different types of data sets :

- The *"strasbourgfly"* stack which was a CT scan of an entire fly.
- Three different data sets presenting isolated legs.

The 3D visualizations presented a remaining problem : the presence of several artifacts all around the body. The objective of the first part of the project was to remove these artifacts to obtain a clean 3D view.

Some of this objects could be simply removed in blender and for some others the solution consisted in implementing an additional part to the python code written by the previous student which would remove these artifacts from the stack. The *Strasbourgfly.tif* data set was the main focus of this cleaning process.

Chapter 1

Treatment of the *"Strasbourgfly"* set

1.1 From stack to mesh treatment

This part was already handled by another student (Lucas) and his report details all of the process. The treatment handled each frame successively and the major steps were the following :

- A thresholding of the image to isolate the different parts : body, air and glue.
- The isolation of a mask of the bubble, since the air and the body were the same shade of gray, the outer layer of the frame had to be erased using a mask.
- Superposition of said mask with the image, removing the upper part.
- Opening of the resulting image to remove small artifacts.
- Dilation of final image to have a more coherent body.
- Execution of a marching cube algorithm which basically creates a 3D mesh out of the superposition of all the frames and results in an *.stl* file that can be opened in blender.

This routine allowed to obtain the result of figure 1.1 :

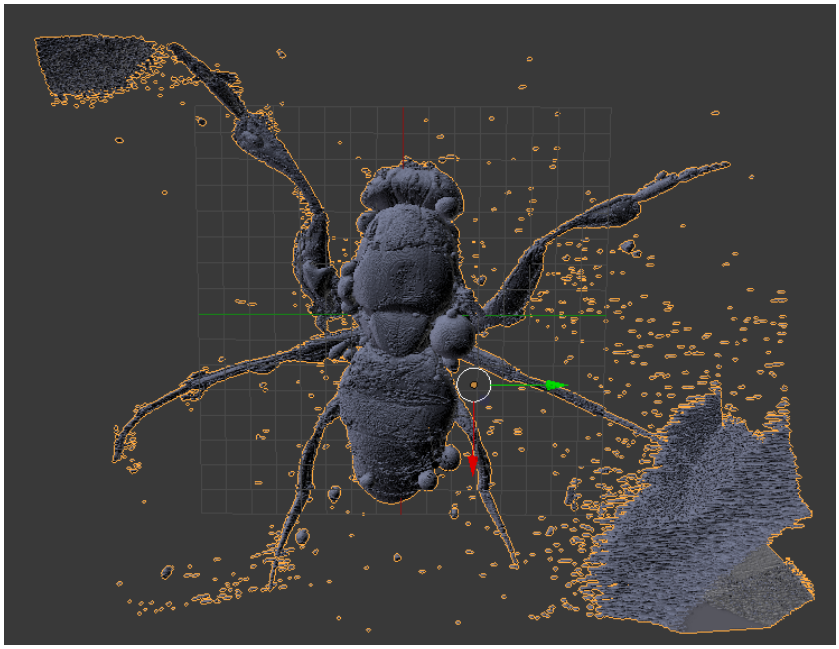


Figure 1.1: Unprocessed results obtained using Lucas' code.

1.2 Cleaning of the artifacts not connected to the body

The results of figure 1.1 present an important number of artifacts all around the body, either connected to it or not. From this point, the objective was to design a semi-automatic treatment allowing to remove these artifacts during the processing of the individual frames. Of all these artifacts, those that were not connected to the body could be erased very simply in blender following this routine :

Blender manipulation 0 : Isolation of the main body

1. **Selecting** a few vertices of the body.
2. Selecting all linked vertices : **CTRL + L**.
3. Inverting selection : **CTRL + I**.
4. Deleting selection **X** $\rightarrow$ **vertices**.

It must be noted that a video is attached to this report explaining more clearly the different processes that were applied in blender throughout the project. Thanks to this process, the results depicted in figure 1.2 could be obtained :

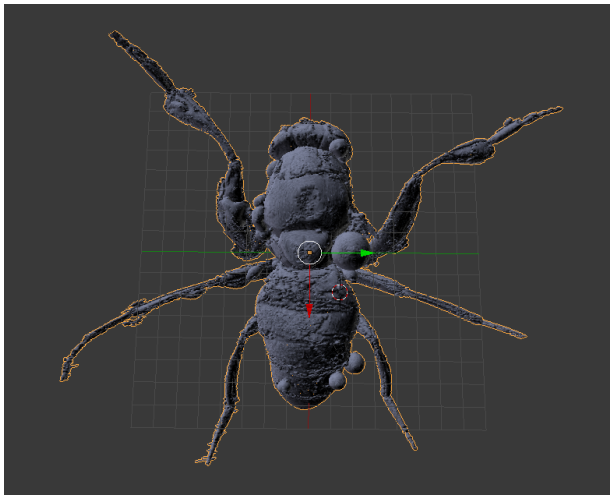
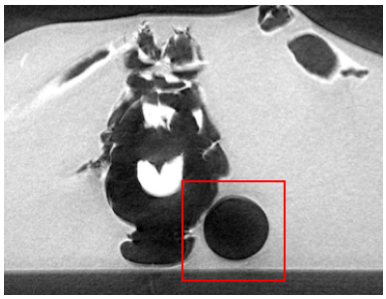


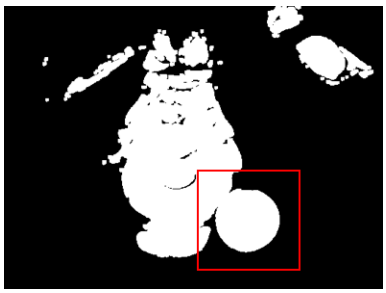
Figure 1.2: Result obtained with the initial code after isolation of the body

1.3 Cleaning of the artifacts connected to the body

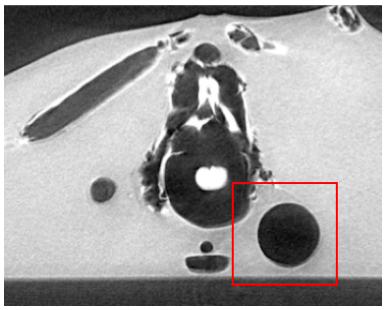
These artifacts were much more complex to treat since one had to have the software distinguish the artifact from the body. In order to do so, two successive treatments were applied : one on the frames were the bubble was not connected to the body and one on the frames were it was. The figure 1.3 presents more clearly the two situations.



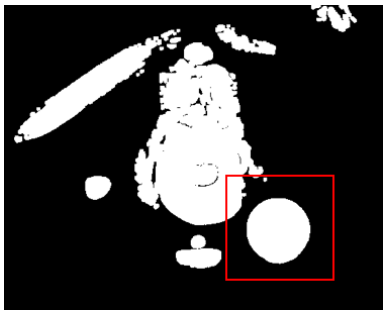
(a) Unprocessed connected bubble



(b) Processed binary connected bubble



(c) Unprocessed unconnected bubble



(d) Processed binary unconnected bubble

Figure 1.3: Figure presenting the unprocessed and binary visualization of the artifacts when they are connected and when they are not.

1.3.1 Removing the part of the artifact not connected to the body

It was much easier to remove the artifact while it was not connected to the body, and the result was cleaner too. Moreover, we chose to work on the resulting binary image for this part since the shapes were more clearly defined in this situation. The manual part of the process was to determine, using the software **imageJ**, the frames were each bubble was and was not connected to the body. When these were obtained, the *findContours_method.py* code could be used (it can be found attached to this report). Basically this code does the following :

- All the frames that are not concerned by the cleaning operation are processed normally with Lucas' code.
- For the frames that are concerned, a window appears on screen which presents the first frame (after treatment) in which the bubble is present. Then the user has to select the region of interest (ROI) (centered around the bubble and as small as possible). A tracker is initialized with this ROI as first object. Figure 1.4 presents this operation.

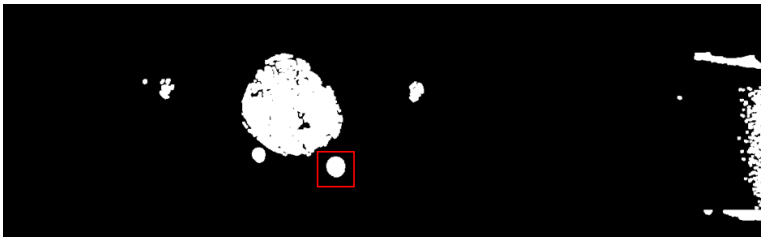


Figure 1.4: Selection of the Region of Interest

- Then the *cv2.findContours* fonction was used in this region of interest to delimit the boundary of the artifact.
- Using the *cv2.drawContours* function, the found contour was filled in (which basically erases it).
- The "cleaned" region of interest was replaced in the original frame, resulting in a frame perfectly cleaned from the artifact, as in figure 1.5 :

(a) Selection of the artifact *(b) Cleaned artifact*

Figure 1.5: Figure presenting the result of the first cleaning method

- When it comes to the next frame, the previously initialized tracker automatically locates the ROI and the process is executed once again.

This was the cleanest and easiest way of getting rid of this part of the artifacts. This program outputs a ".tif" file of each frame, numbered in order. All of these ".tif" files can easily be merged as one big ".tif" file using **ImageJ** . The resulting merged file contains the original processed body, without part of one bubble.

1.3.2 Removing the part of the artifact connected to the body

This task was the trickiest. Indeed, since the bubble is connected to the body, the challenge is to remove it without altering the surface of the body. Since this is hardly possible, the idea was to remove the bubble as cleanly as possible and then smoothing the surface of the body in order for it to be as close to natural as possible.

- Similarly as before, the manual part consisted in finding the frames relevant to this part of the cleaning. We use **imageJ** to find them.
- Then the *Houghs_method.py* code was used. All the frames not concerned with the cleaning opertion are processed normally.
- For the relevant frames, the user has once again to select the region of interest (ROI), but this time in the unprocessed frame (see figure 1.6. As previously, a tracker is initialized with this frame as reference.

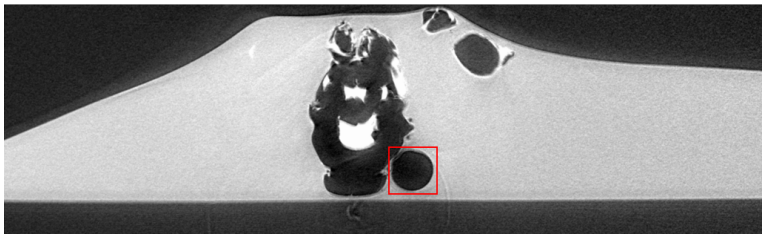


Figure 1.6: Selection of the Region of Interest in the unprocessed frame

- In this region of interest, the function *cv2.HoughCircles* was used in order to detect the shapes resembling circles in the ROI. Indeed, the bubbles were quite perfectly round and this characteristic could be used to differentiate them from the body. Figure 1.7 shows how the program can identify the artifact.

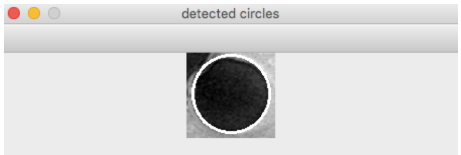


Figure 1.7: Result of the cv2.HoughCircles function

- Then, knowing the location of the bubble as a circle equation in the ROI, it was removed by drawing a full black circle in this location in the processed ROI, as depicted in figure 1.8.

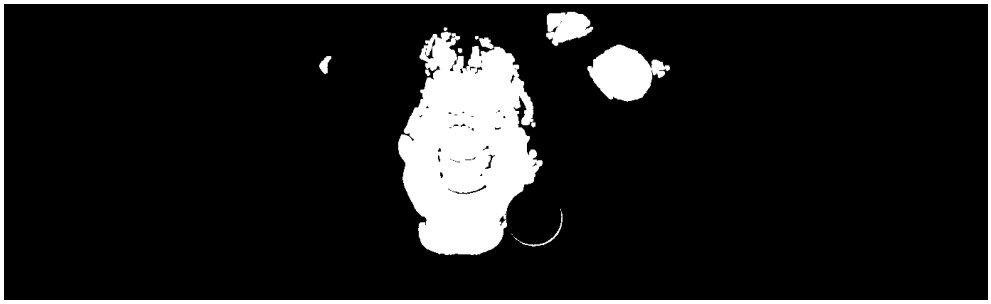


Figure 1.8: Processed frame with the erased circle.

- Since this cleaning wasn't perfect (the found circle didn't correspond perfectly to the bubble, which isn't surprising), parts of the bubble sometimes remained visible but were very small/thin. In order to get rid of these, the successive execution of a erosion and a dilation (which corresponds to an opening operation) were enough to get rid of the remaining artifacts in the Region of Interest. This also allowed to smooth the surface of the body.
- The cleaned ROI was replaced in the processed frame in order to obtain the final "cleaned" frame, as in figure 1.9 :

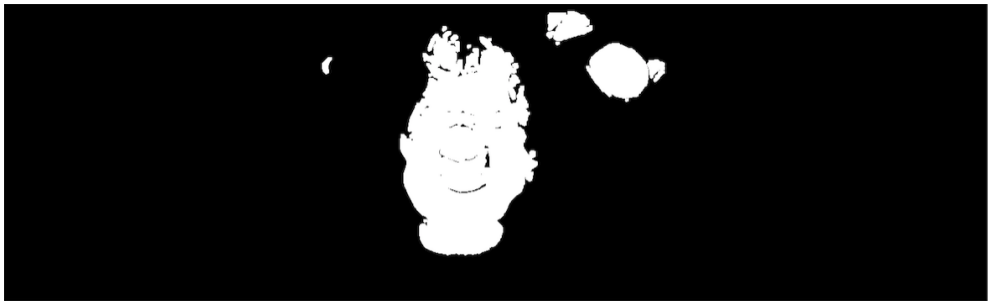


Figure 1.9: Final binary frame after removal

- For the next iterations, the object tracker automatically recognizes the ROI and the process is executed as before.

This was the cleanest way I found to remove the bubble while preserving the surface of the body. This program outputs a ".tif" file of each frame, numbered in order. All of these ".tif" files can easily be merged as one big ".tif" file using **ImageJ** . The resulting merged file contains the original processed body, without part of one bubble.

1.3.3 Merging all of the cleaned file together to build the cleaned mesh

Until now, several ".tif" files with the original body without parts of some bubbles were generated but none of them present the complete body without any of the artifacts. These have to be merged to actually obtain an overall cleaned model.

A simple program *merging_results.py* was written which very simply go through all of those files and, for each frame number, superposes all of the results using the function *cv2.bitwise_and*.

The process goes as follows : for each frame in every result file, if a pixel is white in every file, it will be outputted as white in the final merged results file. If it is black in any of the result files, it will be black in the final merged results file. Since the cleaned artifacts have been rendered black in order to erase them, this simple operation is sufficient to combine the results.

Since over all the files the entirety of each bubble was erased, the output mesh is cleaned of all the artifacts. And finally, the marching cube algorithm is ran over this final stack of frames. This program outputs an "*.stl*" file containing the 3D mesh of the cleaned fly.

Finally, the mesh of figure 1.10 was obtained :

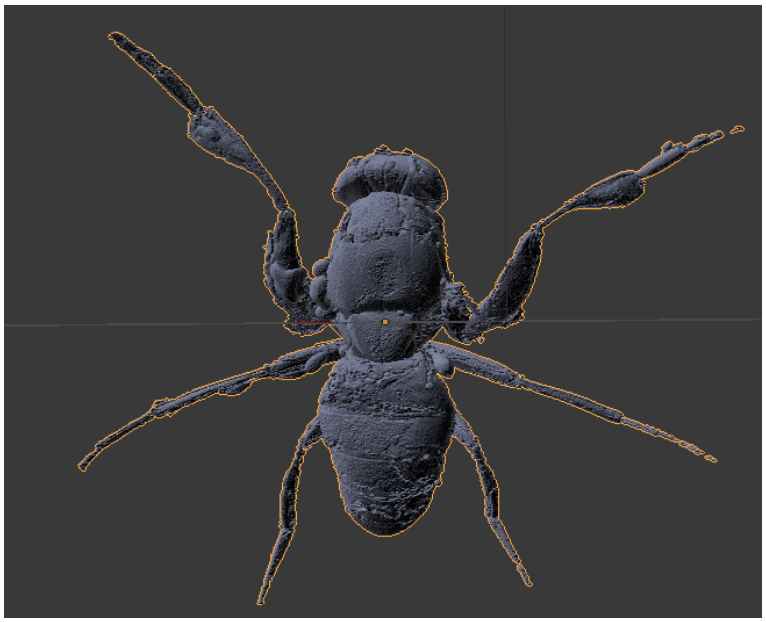


Figure 1.10: Final result for the strasbourgfly data set

Chapter 2

Treatment of the *"Legs"* set

2.1 Analysis of the set using imageJ

First, for each set, three thresholds were distinguished :

- The threshold between the air and the glue.
- the threshold between the glue and the body.
- the threshold between the body and the exoskeleton.

Those could be easily identified using the **threshold** tool of **imageJ**. Afterwards, one only needs to input the identified values into the previously written python code to process the data.

2.2 Application of the python routine

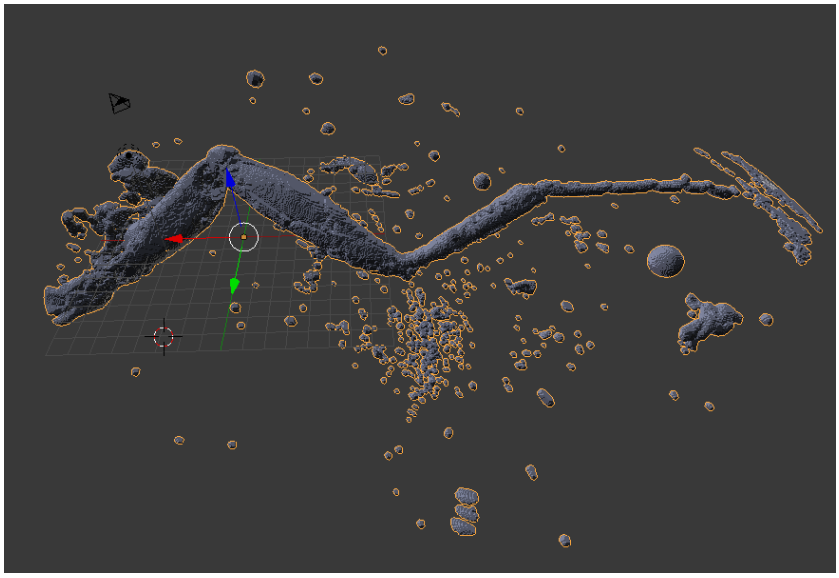
Two different programs were tested for this part. The first one was Lucas' code and the second one a code that I wrote which was similar to Lucas' but which isolated only the exoskeleton, thus the most external part of the body. Both results were considered and are quite alike (unsurprisingly).

It should be noted that the results of my code were less exploitable since they only consider the outer shell, which is less distinguishable on the CT scan and thus the obtained 3D model presents holes. I chose to ignore these results and focus on the data sets obtained with Lucas' code for the rest of the project.

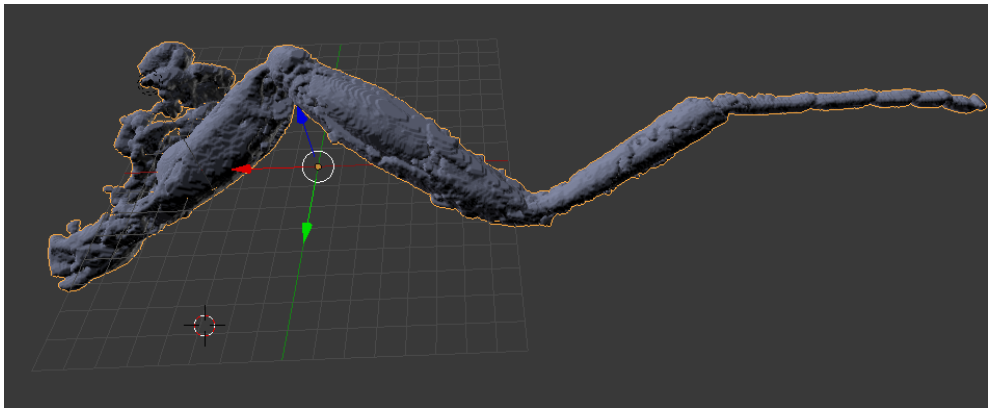
Indeed, the results obtained with Lucas' code were better (no holes) but we should keep in mind that it represents the soft tissue inside of the exoskeleton (even though they are hollow in the 3D form).

2.3 Blender processing

In this case there were no artifacts connected to the body so there was no need for a cleaning using a python routine. After loading the ".stl" file in blender, one only needed to select the body and remove all of the other elements (following the method of section 1.2, ***BM 0***), as depicted in figure 2.1:



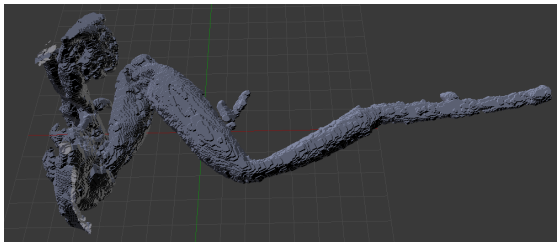
(a) Unprocessed 3D visualization of the leg



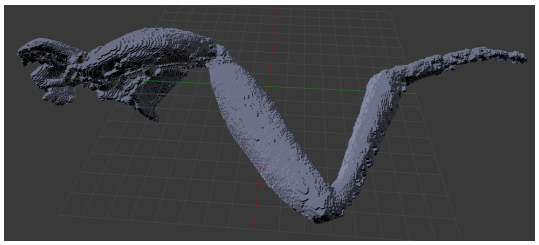
(b) Processed 3D visualization of the leg

Figure 2.1: Figure presenting the processed and unprocessed versions of one of the legs CT scans

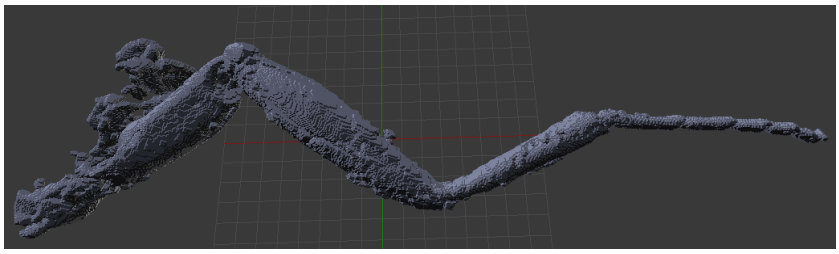
Finally, the three meshes of figure 2.2 were obtained :



(a) 3D visualization of the leg1



(b) 3D visualization of the leg2



(c) 3D visualization of the leg3

Figure 2.2: Figure presenting the visualizations of the three leg meshes

Conclusion for the Image Processing part

In a nutshell, the *image processing* part of this project allowed to pursue the work of the previous student which created a 3D mesh out of the image stack obtained from the CT scanner.

This could be done by manually identifying the frames presenting artifacts and then implementing a section to the python code which enables the user to select a relevant region, erase the artifact in it and track this region in the following frames, repeating the cleaning process along the way.

Taking a step back on this process, I think that it can still be perfected and more automated. For problems with many more frames, the manual part would be too fastidious. However, we still managed to achieve the desired cleaning on the *strasbourgfly* data set, which could in turn be 3D printed.

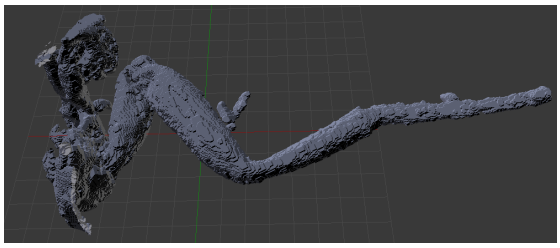
Part II

Exoskeleton Design

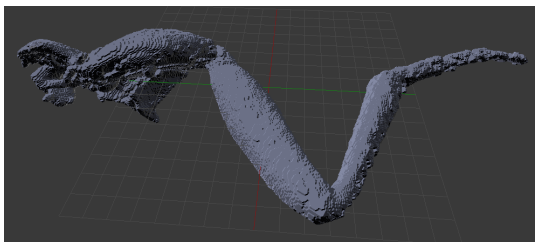
Introduction

The following task consisted in designing an exoskeleton for the front legs of a robotic fly prototype. Another student, Quentin Vingerhoets, worked on the inner part of this robotic arm : the actuation system, motors and overall structure. The details of his work should be presented in his report. My job here was to design an outer shell that would fit around this structure whilst resembling as much as possible to the actual biological data extracted from the CT scans. This exoskeleton was 3D printed using the *Form2* printer from *formslab*.

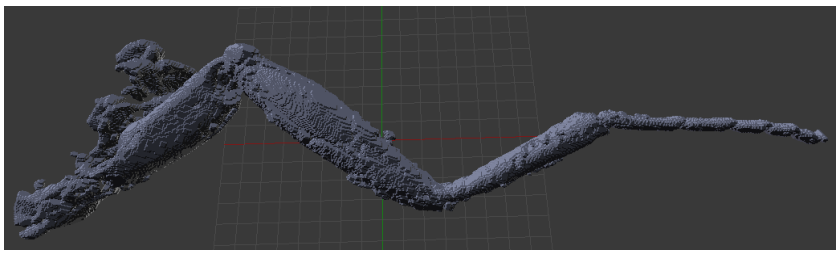
The main challenge here is to build a rigorous mechanical design, responding to precise constraints regarding dimensions and weight out of a biological body which is much more complex. Thanks to the previous image processing work, three mesh files were obtained, which were named, surprisingly, *leg1, leg2* and *leg3*. These are depicted in figure 2.3 :



(a) 3D visualization of the leg1



(b) 3D visualization of the leg2



(c) 3D visualization of the leg3

Figure 2.3: Figure presenting the visualizations of the three leg meshes

After a short analysis of these three meshes, the last one was chosen as model because of its precision (taking a look at the last segment, the tarsus, one can easily distinguish all the segments) and its stretched position (which makes the distinction between all the segments easier).

This work will consist in designing the 4 segments of this leg's exoskeleton : the **coxa**, the **femur**, the **tibia** and the **tarsus**. In order to do so, the segments must be separated and their meshes must be simplified to get a simple yet accurate outer shape. Furthermore, a more rigorous technical work must be done in order for the exoskeleton to fit around the inner mechanisms and for it to be printable. **A video is attached to this report presenting all of the tools that were used to obtain the exoskeleton.**

Chapter 3

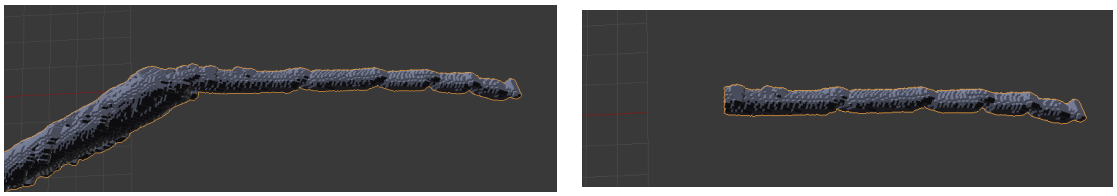
Design of the outer shape

3.1 Separation of the segments

As said previously, the *drosophilia's* leg is composed of 4 segments, from left to right on figure 2.3 :

- The coxa : the link with the main body.
- The femur.
- The tibia.
- The tarsus : last and most precise segment, itself composed of five different segments and a claw.

The first task consists in separating all of these segments. In blender this could be done by following the ***Blender manipulation 1*** of the appendix. It must be noted that the delimitation between one segment to the other was not precise and had to be estimated, as presented in figure 3.1 in the case of the tarsus :



(a) Tarsus linked to the leg *(b) Separated tarsus*

Figure 3.1: Figure presenting the result of the separation process

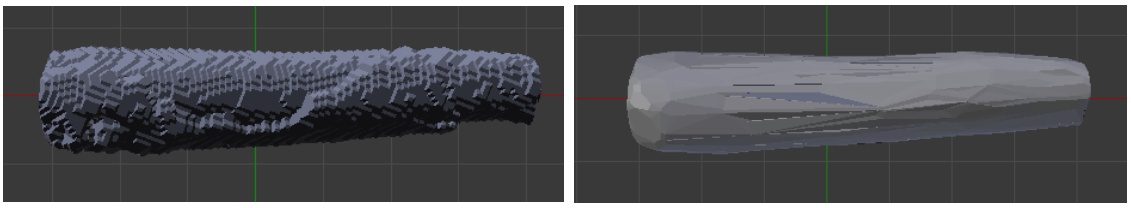
Thanks to this process, each segment of the leg could be isolated.

3.2 Simplification of the outer shape

The exoskeleton is designed in order to be 3D printed and as it is so far, it is too complex. The goal is now to simplify this outer shape in order for it to be printable. The ***Blender Manipulation 2*** depicts the process that must be followed in order to obtain this simplified shape.

This routine uses the **shrinkwrap** tool, which basically wraps a desired shape around a target, while preserving the number of vertices that composes this desired shape. This is equivalent to wrapping a sheet of paper around the concerned segment.

In order to obtain more precise extremities for the simplified shape, the **bisect** tool was used in order for the extreme faces to be vertical and simple (***Blender manipulation 3***). This process is presented in figure 3.2 :



(a) Original separated tibia *(b) Simplified tibia shell*

Figure 3.2: Figure presenting the result of the separation process

Thanks to this process, the outer form of our skeleton is now defined for the biggest pieces : coxa, femur, tibia.

3.3 Case of the tarsus

The tarsus is a more complex part of the leg : it is itself composed of five different segments and we want this property to appear in the simplified exoskeleton. Also, it can be seen that the tarsus on the CT scan is not perfectly straight, as figure 3.3 shows :

Figure 3.3: Original mesh of the tarsus

In order to obtain the outer shape of the tarsus the following procedure was used :

1. Separate the five segments of the tarsus using ***Blender Manipulation 1, (BM1)*** and ***BM 3*** combined (the bisect tool could be used to get precise delimitations of each segment)
2. Obtain the simplified shape of each segment using ***BM 2***
3. Readjust the position of each segment, by **rotating them** and translating them correctly.
4. Use the boolean modifiers following ***BM 4*** in order to unite all of the segments as one body

This method allowed to obtain a straight, simplified tarsus shape which still presented all five segments, as in figure 3.4 :

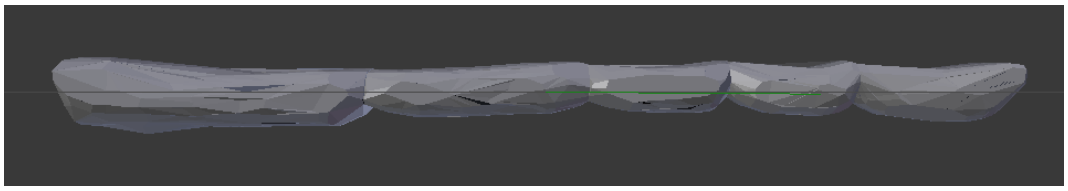
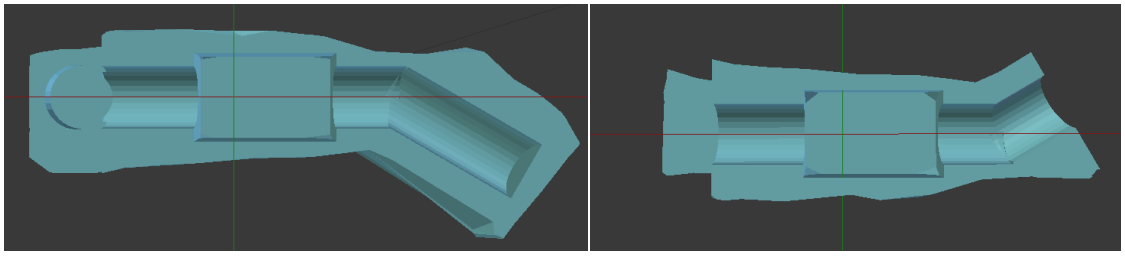


Figure 3.4: Final outer shape of the tarsus

We now have the simplified outer shape of all the segments. They must now be modified in order to be printable and to correctly fit around the inner mechanisms.

3.4 First trials for the tibia

It must be noted that I tried a first tibia design, which was inconclusive. Indeed, as figure 3.5 shows, the first trial consisted in a bent piece in which holes were applied.



(a) First trial for the tibia, lower part *(b) First trial for the tibia, upper part*

Figure 3.5: Figure presenting the first design of the tibia

This method for hollowing the piece was not satisfying since it resulted in a piece that was too massive and restrictive (regarding the number of elements that it could fit). Therefore, the following method was used.

Chapter 4

Technical components

4.1 Creation of the shell shape

With the previous section, we were able to get a simple shape for the outer surface of each segment of our exoskeleton. Since it must be printed, it is now important to give a thickness to this shape. The dimensions of this outer surface must also be readjusted in order for all the mechanisms to fit inside of it.

4.1.1 Readjusting the dimensions

In order for the exoskeleton to correspond to the inner mechanisms, a close collaboration with my project partner Quentin began. He gave me the files corresponding to the 3D modelisation of its inner structure.

To get a better numerical visualization of the complete system, I recreated the inner structure in blender. This was doable by creating new shapes (cubes and cylinders), resizing them, placing them accurately and applying **parent/child relations** between them. The ***Blender Manipulation 6*** relates this process.

This part was very fastidious but finally allowed to obtain precise structures such as the one of figure 4.1 :

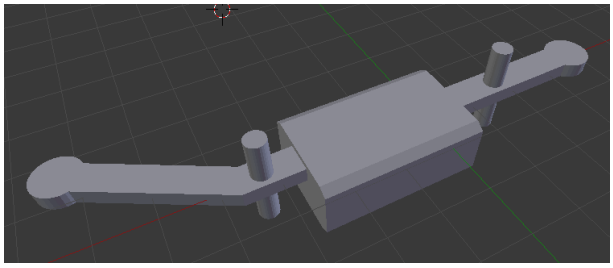


Figure 4.1: Inner structure of the tibia segment

The following step consisted in manually resizing the simplified shape over its 3 dimensions in order for it to fit around this structure. Here, the accuracy regarding the biological model is lost but it is necessary in order to respect these technical constraints. The scale of the exoskeleton is deemed acceptable when it looks like figure 4.2:

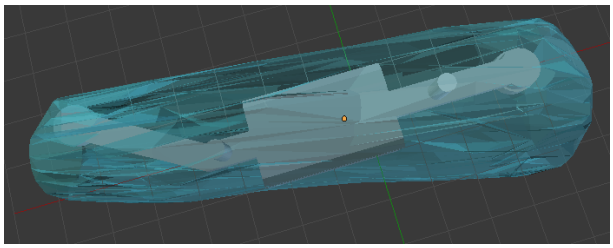


Figure 4.2: Resizing the outer surface for it to fit around the inner structure

This was done for all the segments and gave the dimensions of the inner surface of our exoskeleton.

4.1.2 Giving the shape a thickness

It is now important to give a thickness to the built shape. Since the inner shape is adjusted to the inner structure, the thickness must come "out" of it. In order to do so, the shape is **duplicated**, the duplicate is resized in order to be 4 mm larger regarding its height and width but not its length. This way, the duplicated shape is now similar to the original one but, assuming it is a cylinder, its radius is now 2mm bigger than that of the original shape. We call this new surface the *outer surface*.

In order to obtain a volume which is basically a tube with a thickness of 2mm, we simply subtract the first shape to the *outer surface*, using ***BM 5***.

This operation subtracted the first volume to the second one, thus hollowing it precisely as desired. It must be noted that blender is not perfect and, since these are complicated shapes the operation does not always happen correctly and some unwanted faces remain. In order to correct these mistakes, one simply has to go into **edit mode**, **select** and **delete** the unwanted faces. We obtain results such as the one depicted in figure 4.3 :

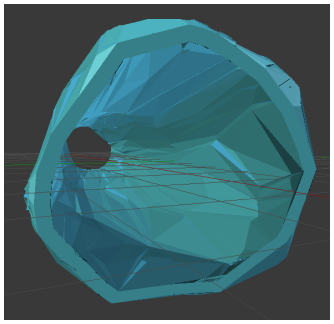


Figure 4.3: Result of the difference operation to obtain the hollow shell

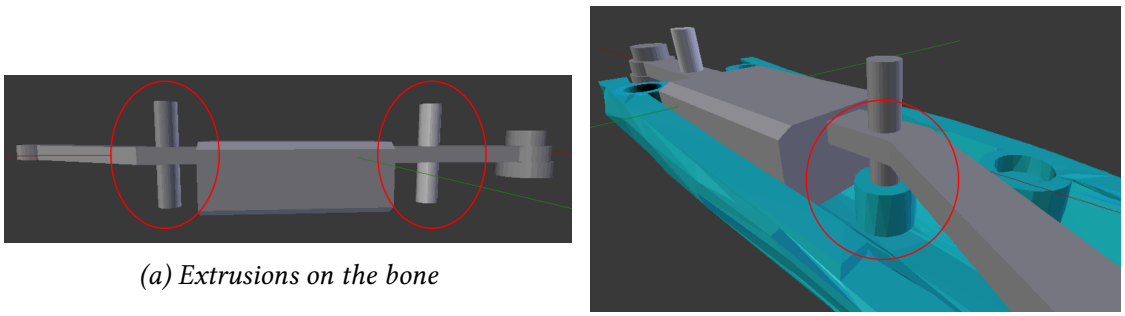
4.2 Adding the technical components

Finally, the desired shell shape could be obtained. It is thick enough to be resistant while being light enough for the motors to carry them. The remaining problem concerns the fixation of this shell on the inner structure.

For simplicity's sake, it was decided to print each segment's exoskeleton in two halves which we would stick to one another using magnets. This was done for every part except for the tarsus which was printed as one whole piece with a central hole, because of its small dimensions. Separating the shells in two was fairly easy using the **bisect** tool (***BM 3***). It was also essential to connect these two halves with the bone.

4.2.1 Links with the bone

In order to link the shell with the bone, a cylindrical extrusion was designed on the bone that would be fitted in the shell, as depicted in figure 4.4 :



(a) Extrusions on the bone

(b) Reception of the links in the shell

Figure 4.4: Figure presenting the connection structure between the bone and the exoskeleton

Building these *"receptors"* was complicated and implied a combination of several tools. The main idea was to :

Blender manipulation 7 : Creation of the link receptors

1. create a cylinder
2. **resize** it and **place** it very accurately respectively to the bone's extrusion
3. **subtract** the bone's extrusion to it (***BM 5***)
4. **unite** the hollowed cylinder with the corresponding shell's half (***BM 4***)

This had to be done several times for each shell's half.

4.2.2 Magnets for the fixation of the parts

As said previously, it was decided that the two halves of each segment would be fixed to one another using magnets. Thus, extrusions (that i called *"magnet holders"*) had to be designed in each half for the magnets to be placed in. This process resembles ***BM 7*** a lot and is resumed in ***BM 8*** :

Blender manipulation 8 : Creation of the magnet holders

1. create a cylinder which will be the magnet
2. create a wider cylinder which will be the magnet holder
3. **resize** them and **place** them very accurately respectively to one another
4. **subtract** the magnet to the magnet holder (***BM 5***)
5. **unite** the magnet holder with the corresponding shell's half (***BM 4***)

Because of the shells geometry it was sometimes necessary to crop the cylinders of the magnet holder using the **bisect** tool (***BM 3***). The magnets were placed at convenient locations, leaving space for the inner structure, as presented in figure 4.5 for instance :

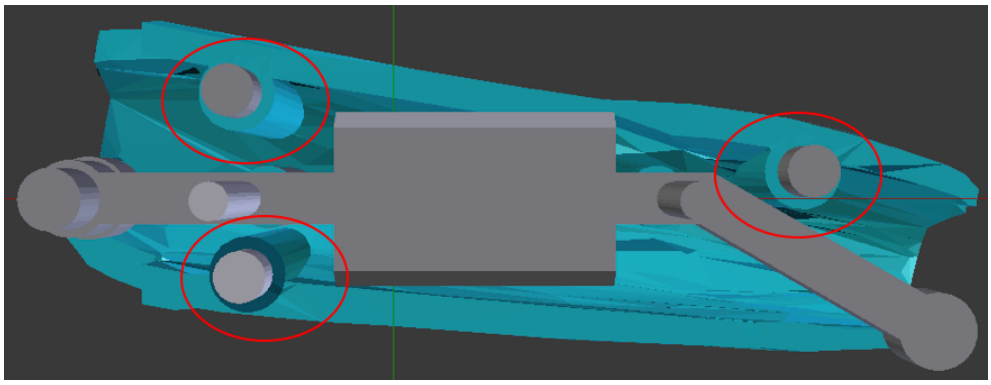


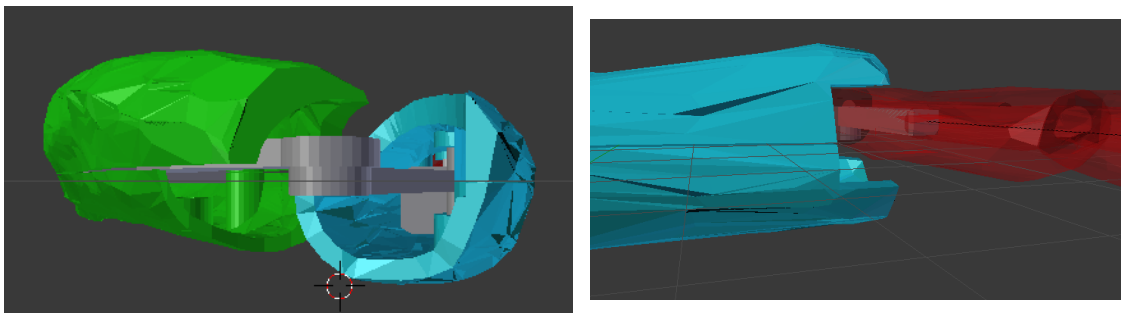
Figure 4.5: Figure presenting the position of the magnets in the lower half of the tibia shell

4.2.3 Holes for the joint overlaps

In the case of the **femur/tibia** joint, the exoskeletons have to overlap. To make this possible, we chose to have the femur be the "upper" part and the tibia the "lower" part of the joint. Thus, two cavities were drilled using cylinders and the tool ***BM 5*** : one in the upper part of the tibia and one in the lower part of the femur.

A similar problem was present for the **tibia/tarsus** joint. This time we chose to drill the cavity in the tibia and have the tarsus fit in it.

Figure 4.6 represents these situations more clearly.



(a) Overlap of the Femur/Tibia joint

(b) Overlap of the tibia/tarsus joint

Figure 4.6: Figure presenting the connection structure between the bone and the exoskeleton

It must be noted that the parts 4.1 and 4.2 only concern the Coxa, Femur and Tibia. Because of its dimensions, the Tarsus was treated differently and the corresponding process is described in section 4.3

4.3 Design of the Tarsus

In the case of the tarsus, since the "diameter" of the part is smaller, it is not convenient to cut it in halves. It seems more appropriate to print it as one piece with an internal cavity which would be slid on the corresponding bone. Figure 4.7 presents this attachment strategy :

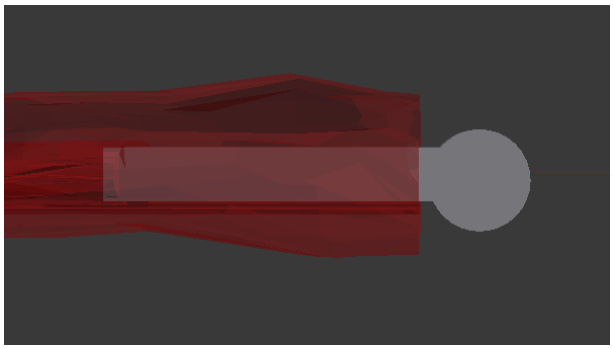


Figure 4.7: Attachment of the tarsus to its bone

In order to create the hole in which the tarsus' bone would fit (the *"attachment hole"*), the bone was subtracted (***BM 5***) to the tarsus. Then, in order to hollow the rest of part, the same chain of thought as previously was used :

Blender manipulation 9 : Hollowing the tarsus

1. **duplicate** the tarsus
2. reduce the dimensions of the duplicate by 6mm in width and height (in order to reduce the *"radius"* of the piece of 3 mm)
3. **crop** the piece on both ends using the **bisect** tool (***BM3***), so that the internal cavity joins the attachment hole
4. **substract** the internal cavity to the tarsus (***BM 5***)

We now have a completely hollowed piece. The last operation that is applied to it is the drilling of a half cylindrical hole on the side of the attachment hole. Since the internal cavity is expected to be used to store wires connected to sensors located on the surface of the exoskeleton, this cylindrical hole would be an exit for the wires. This is presented in figure 4.8 :

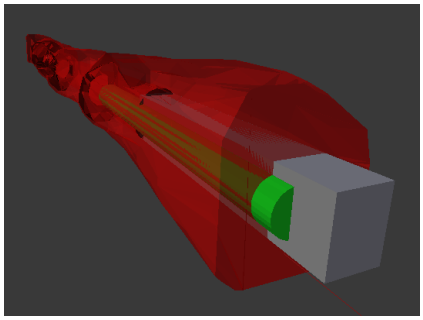


Figure 4.8: Cylindrical hole for the wires (in green)

Conclusion for the exoskeleton design part

This section of the report only presented the overall method that was used in order to design the 4 segments of out robotic leg. The actual designs are attached to this report, as well as a video that actually presents how to use the different tools in a more explicit manner.

Mechanical design in blender is actually quite hard since one has to work in absolute coordinates in order to place the objects respectively to one another. However, since the number of pieces was limited in our situation it was still a correct platform. Some additional notes can be added : for most holes, a small margin of 0.1mm can be added since the 3D printer generally prints them narrower than they actually are.

This part of the work allowed to show the major constraints of the problem and the tools that can be used to solve them. The resulting designs should be later on adapted to the sensor technology that will be used for the sensory feedback.

Part III

Sensors : State of the Art

Motivation

Another part of the project consisted in a research work regarding the tactile sensors that could be implemented on the exoskeleton in order for our robotic fly to perceive a sensory feedback. This feedback would be then implemented in the controller in order to mimic at best the different reflexes of the actual *drosophilia*.

It must be noted that the *drosophilia* presents 2 major types of tactile sensors on its legs which we want to mimic :

- The bristles : hair-like organs on the surface of the exoskeleton which, when bent because of the contact with another body, send electrical signals to the brain to relate this contact.
- The *campaniform sensilla* : zones of the outer exoskeleton which detect tension and compression in the surrounding cuticle.

State of the Art

Various technologies were thought of, which would mimic different components of the *drosophilia* tactile sensory system :

- **Deformation sensitive resistive material designed to be bristle-like :** use of bend sensors, can be cheaply bought or handmade quite easily (but depend on desired quality and could hardly be miniaturized)
  →*http://www.flexpoint.com/product/bend-sensor/*

- **Capacitive tactile sensor (analog to touchscreen technology), which would mimic the *campaniform sensilla* :** Could be very adapted since it could cover almost all of the exoskeleton's area
  →*https://pressureprofile.com/general-sensor-catalog.*

- **Position sensors, still analog to the *campaniform sensilla*, still capacitive based sensors, but less precise and less expensive :** These sensors are strip like and correspond fairly well to our need (we would choose the 3-pin sensor in order to have the switch-like response)
  →*https://www.tekscan.com/product-group/embedded-sensing/position-sensors*
  →*https://www.interlinkelectronics.com/fsr-408*

- **Quantum-tunneling based sensors :** extremely precise as well but very expensive and probably overkill in our situation
  →*https://www.peratech.com/m2436-100148-zf.html*

- **Binary matrix of electrical switches (palm sized crawling robots) :** the problem ressembles ours and the solution is interesting
  →*https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-40.pdf*

- **Electrical Impedance Tomography :** an interesting research over wrapable sensor sheets
  →*https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5134587/*

Appendix

Blender manipulation 1 : Separation of a segment

1. go into **Edit mode**
2. use **wireframe visualization pressing Z**
3. deselect all of the vertices by **pressing A**
4. select all of the vertices of the segment of interest : press **B or C** to enable rectangular or circular selection
5. separate these vertices from the original body by **pressing P**
6. go back to **object mode**
7. change the origin of the separated body to its center of gravity by **pressing SHIFT + CTRL + ALT + C**.
8. replace the desired segment and/or deleting the rest of the original body. Useful displacement tools : **R + X/Y/Z** rotates the body along the chosen axis, **G + X/Y/Z** translates it along that axis.

Blender manipulation 2 : Construction of a simplified shape

1. center the original segment at the origin : **press N** and set **location** at **(0,0,0)**
2. create a 3D shape **Add** $\rightarrow$ **Mesh** , you can choose depending on the width, length and shape of the original segment. Generally you pick the one that gives out the best result, in our case generally it is the **icosphere**.
3. **select** the created shape (and eventually **rename** it), and **center it at (0,0,0)**
4. go into **add modifier** and select the **subdivision surface** modifier in order to increase the number of faces/vertices of the created body. Increase the number of vertices by 3 to 6 **subdivisions** (the precise number changes depending on the target body, adjust it at will) and **apply** the modifier
5. add the **shrinkwrap** modifier and select the original segment as target, **apply** the modifier
6. **replace** the obtained segment using either **Z or G**

Blender manipulation 3 : The bisect tool

1. Go into **edit mode**
2. **Select** all of the vertices by **pressing A**
3. place the piece conveniently
4. click on the **bisect tool** and draw the desired **bisection line**
5. in the bottom left menu, precise the **plane normal** in order to obtain a rigorous section direction. You can also select **filled** to fill the resulting face and **clear inner/outer** to delete the unwanted part of the body.

Blender manipulation 4 : The boolean modifiers, union

WARNING : the position of the parts respectively to one another is very important while using the boolean modifiers

1. **Select** the part of interest and position it accurately in respect to the part with which we want to link it
2. add a **boolean modifier** and select **"union"** in the **operation menu**
3. select the body with which we want to apply the union as **target**
4. **apply** the modifier

Blender manipulation 5 : The boolean modifiers, difference

WARNING : the position of the parts respectively to one another is very important while using the boolean modifiers

1. **Select** the part of interest and position it accurately in respect to the part that we want subtracted to it
2. add a **boolean modifier** and select **"difference"** in the **operation menu**
3. select the body we want to subtract from our piece as **target**
4. **apply** the modifier

Blender manipulation 6 : Creation of the inner structure

1. create each new element : **Add** → **Mesh** and select the desired shape
2. resize it using the menu that appears on the right when **pressing N**
3. place it very precisely respectively to the other elements, in absolute coordinates
4. after all the elements are created, chose a **main component** for each segment (generally there is a main *bone* for each segment)
5. select the element that must **depend on** this main bone
6. **hold SHIFT** and select (**right click**) the main bone
7. press **CTRL + P** and select **set parent to object** to have the first element become a child to the main bone
8. to remove this relation, select the child and press **ALT +P, clear parent**
9. once all the structural elements of a segment are child to the main bone, they will translate/rotate with him if we displace it