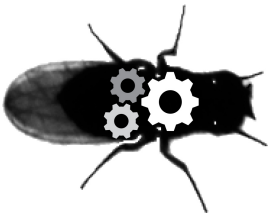
Master's thesis

Closed loop robotic system for stimulating tethered *Drosophila melanogaster*

Author : Nathan Müller
Supervised by : Prof. Pavan Ramdya, MSc. Victor Lobato Rios
EPFL Neuroengineering Laboratory

January 21, 2022



I would like to thank Prof. Pavan Ramdya for giving me the opportunity to conduct a multidisciplinary project through which I was able to implement a complete robotic solution. I am also grateful for his supervision throughout this project.
It was a pleasure to collaborate with MSc. Victor Lobato Rios. Thanks to his experience in flies experiments and mechatronics, the valuable insights he was able to provide for developing a meaningful robotic system have notably been a great help.
A special thanks also to Dr. Matthias Durrieu who always offered his help for flies manipulations needed to conduct the different experiments.

Lausanne, 21. January 2022 
Nathan Müller
Master of Robotics Candidate

Contents

1 Introduction **8**
1.1 *Drosophila melanogaster* . . . 8
1.2 Experimental principle . . . 9
1.3 State of the art . . . 10
1.3.1 Feeding . . . 10
1.3.2 Stimulating . . . 11
1.4 Robotic solution . . . 13
1.5 Main contributions . . . 14

2 Environment **15**
2.1 Seven cameras setup . . . 15
2.2 Experimental setup . . . 16

3 Requirements **16**
3.1 Experimental requirements . . . 16
3.2 Mechanical requirements . . . 17
3.3 Dynamical requirements . . . 17

4 Design **18**
4.1 Cartesian micromanipulator . . . 18
4.1.1 Sensapex uMp3 . . . 18
4.1.2 Siskiyou . . . 19
4.1.3 Trio . . . 20
4.1.4 Physik Intrumente . . . 20
4.1.5 Manual Stages . . . 21
4.1.6 Chosen solution: Sensapex uMp3 . . . 21
4.2 Updated design idea . . . 22
4.3 Translation stage . . . 23
4.4 Rotational stage . . . 23
4.5 Motors . . . 25
4.6 Motor controllers . . . 25
4.7 Motor drivers . . . 26
4.8 Power supplies . . . 26
4.9 Cameras . . . 26

5 Developed robotic system **26**
5.1 Mechanics . . . 26
5.2 Electronics . . . 29
5.3 Costs . . . 30

6 Low level controller **31**
6.1 Sensapex uMp3 controller . . . 31
6.2 Stepper motor controller . . . 31
6.2.1 PC communication . . . 32
6.2.2 Motor signals generation . . . 34
6.2.3 Example . . . 39

7 High level controller **41**
7.1 Computer vision . . . 41
7.2 Closed loop controller . . . 42
7.3 Open loop controller . . . 45
7.3.1 Forward kinematics model . . . 46
7.3.2 Inverse of the Jacobian . . . 46

8 Performances **47**
8.1 Mechanics . . . 47
8.2 Low level controller . . . 47
8.2.1 Communication . . . 47
8.2.2 Step signals generation . . . 48
8.2.3 Delays . . . 48
8.3 Dynamics . . . 48
8.3.1 Resolution . . . 48
8.3.2 Acceleration . . . 49
8.3.3 Speed . . . 49
8.4 High level controller . . . 49
8.4.1 Closed loop controller . . . 49
8.4.2 Open loop controller . . . 49

9 Applications **50**
9.1 Feeding . . . 50
9.1.1 Previous feeding method . . . 50
9.1.2 End effector design . . . 51
9.1.3 User defined goal position . . . 53
9.1.4 Automatic movement . . . 53
9.2 Trajectory . . . 58

10 Conclusion **60**

11 Further work **61**

References **62**

Appendices **66**

A Robotic platform assembly **66**

B Prototype board layout **77**

C Closed loop model **78**

D Needle preparation **80**

E Using the robotic system **83**

List of Figures

1 Female and male *Drosophila melanogaster* . . . 8
2 Seven cameras setup . . . 9
3 Preparation of *Drosophila melanogaster* for two-photon imaging in the seven cameras setup with microscope . . . 9
4 Spherical treadmill . . . 9
5 Workflow of releasing flies to feed them [15] . . . 10
6 Schematic of the fixed feeding element [16] . . . 10
7 Schematic of the motorized feeding element [9] . . . 11
8 Schematic of the VR screen setup [17] . . . 11
9 Schematic of the airflow setup [18] . . . 12
10 Schematic of the obstacle setup [19] . . . 12
11 Developed robotic system . . . 13
12 3D model of the most constraining seven cameras setup. The V shape is used as support for optic flow sensors which are not used in this work . . . 15
13 Side view of the seven cameras setup. Illustration of the dimensions . . . 15
14 Replicated setup used to develop the robotic system . . . 16
15 Forward velocity in freely behaving flies [26] . . . 17
16 Velocity profile of tethered flies [27] . . . 17
17 Illustration of a Cartesian robotic system [28] . . . 18
18 Sensapex uMp3 micromanipulator [22] . . . 18
19 Siskiyou MX7600 Series Manipulator [29] . . . 19
20 Trio-245 manipulator [30] . . . 20
21 PI L-402 Miniature Linear Stage [31] . . . 20
22 Manual solutions . . . 21
23 Positioning of the Sensapex uMp3 manipulator . . . 22
24 Illustration of the additional degrees of freedom . . . 22
25 Zaber LSM25B-TA4 linear stage [34] . . . 23
26 Rotational stage concepts . . . 24
27 Rotation table with a 50mm aperture [35] . . . 24
28 Rotational stage concepts . . . 24
29 STM32 Nucleo-64 development board with STM32F401RE MCU . . . 25
30 Bracket . . . 26
31 Electronics mount . . . 27
32 Rotational platform . . . 27
33 End effector . . . 27
34 Mount robotic system on base plate . . . 28
35 Robotic system . . . 28
36 MCU pins attribution . . . 29
37 Prototype board . . . 30
38 General overview of the low level controller . . . 31
39 Processes running on the MCU in idle state . . . 32
40 General overview of the reception of the commands . . . 34
41 Trapezoidal velocity profile . . . 35
42 Center aligned mode counter for timer 1 and 3 . . . 35
43 Acceleration profile with a constant decrease in the steps period . . . 36
44 Speed profile during acceleration. The orange dots correspond to the command and the blue dots to the applied frequency . . . 37
45 Pulse train generated during the acceleration . . . 37
46 Processes running on the MCU during the initialisation of the motors . . . 39
47 Processes running on the MCU during the homing of the linear stage . . . 39

48 Processes running on the MCU when commands are sent. It should be noted that the channel 4 "COM received" does not indicate the DMA received but that at least the command for one of the motors has been found during the buffer search. (1) get linear position received. (2) DMA sends linear position. (3) get move linear and get rotational position. (4) DMA sends rotational position. (5) get move rotational command . . . 40
49 Processes running on the MCU when both motors are moving . . . 41
50 DeepLabCut overview . . . 42
51 Sensapex velocity profile . . . 43
52 Pinhole camera model . . . 43
53 Closed loop control algorithm model. The projection in the horizontal plane does not correspond to those images for illustrative purposes . . . 44
54 Geometric model . . . 46
55 Working area. The dot at the origin represents the ball. The orange and the blue semi-circles correspond to the 12.5mm radius and the rail respectively . . . 47
56 uMp-RW3 rotary wheel unit . . . 50
57 Previous feeding method . . . 50
58 1mm hypodermic needle . . . 51
59 0.3mm hypodermic needle . . . 51
60 Side hole needle . . . 52
61 Developed side hole needle . . . 52
62 User defined goal and origin . . . 53
63 Cameras used to train the "front fly" neural network . . . 53
64 Points to detect for the front fly neural network . . . 54
65 Points to detect for the needle neural network . . . 54
66 Different end effectors to train the "needle" neural network . . . 55
67 Different initial positions from which the closed loop controller can successfully accomplish its task . . . 56
68 Contact with the ball . . . 58
69 Waypoints reached by the open loop algorithm . . . 59
70 Parts overview . . . 66
71 Sensapex: base installation . . . 67
72 Linear stage installation . . . 68
73 Rotational rotor and pinion installation overview . . . 69
74 Rotational motor mounting . . . 69
75 Shaft adapter: preparation . . . 70
76 Fixing screw inserted in pinion . . . 70
77 Pinion installation . . . 70
78 Installation of the rotational platform mounted on the rotational carriage . . . 71
79 Bracket installation . . . 71
80 Sensapex: mounting on the bracket . . . 72
81 Electronics installation . . . 73
82 Base of the end effector installation . . . 74
83 Installation of the needle . . . 75
84 Installation of the tube holder . . . 76
85 Fixing the rail to the base plate . . . 76
86 Prototype board layout . . . 77
87 Closed loop control algorithm model. The projection in the horizontal plane does not correspond to those images for illustrative purposes . . . 78
88 Needle grinding . . . 80
89 Grinded needle . . . 80
90 Plastic deposition at the needle tip . . . 80
91 Needle tip with plastic . . . 81
92 Inserting thread in the needle . . . 81
93 Finished needle . . . 82

List of Tables

1 Priorities of the different processes of the low level controller. From highest priority to lowest priority . . . 32
2 Communication protocol . . . 33
3 Minimal and maximal velocity along the different axes . . . 49
4 Communication protocol . . . 83

List of Videos (to be found in the folder of the report under Videos/)

1 Trapezoidal_velocity_profile/acceleration.mp4
2 Feeding/Previous/previous_feeding_method.mp4
3 Feeding/User_defined_goal/user_defined_goal.mp4
4 Feeding/Automatic/feeding_wild_fly_0.mp4
5 Feeding/Automatic/feeding_wild_fly_1.mp4
6 Feeding/Automatic/feeding_wild_fly_2.mp4
7 Feeding/Automatic/feeding_wild_fly_3.mp4
8 Trajectory/trajectory.mp4

1 Introduction

The *Drosophila melanogaster* – commonly known as the vinegar fly – is a versatile model organism that has been used in biomedical research for over a century to study a broad range of phenomena [1].

In this work, a visually guided robotic system for stimulating a tethered *Drosophila melanogaster* walking on a spherical treadmill has been developed.

This robotic platform will allow new experiments to be carried out. Two use cases are presented:

- Automated feeding of tethered *Drosophila melanogaster* enabling to conduct longer experiments;
- Trajectory following enabling to move an object in the fly's surroundings.

The tools developed during this project can be used to perform many other experiments.

1.1 *Drosophila melanogaster*

Although humans and vinegar flies (Figure 1) may not look very similar, it has become well established that most of the fundamental biological mechanisms and pathways that control development and survival are conserved across evolution between these two species [1]. In fact, the *Drosophila*'s genome is 60% homologous to that of humans, less redundant, and about 75% of the genes responsible for human diseases have homologs in flies [2]. Thus, the vinegar fly has become a valuable model system for unraveling basic mechanistic insights into the regulation of metabolism [3, 4, 5].

Drosophila melanogaster has been used in biomedical research to study a broad range of phenomena including genetics and inheritance, embryonic development, learning, behavior and aging. There are also many technical advantages of using *Drosophila* since they are easy and inexpensive to culture in laboratory conditions, they can be genetically modified in numerous ways and there are less ethical obstacles in the experiments which can be conducted than for humans studies [1].

Furthermore, the vinegar fly is an incredibly versatile organism capable of both innate and higher-order behaviors while having a less complex neural system than humans – six orders of magnitude less neurons – and hence a better comprehension of the brain dynamics is achievable.

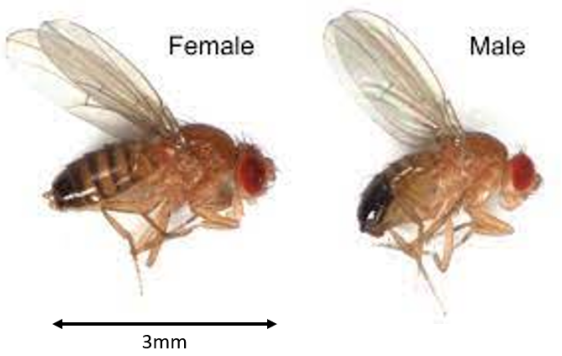
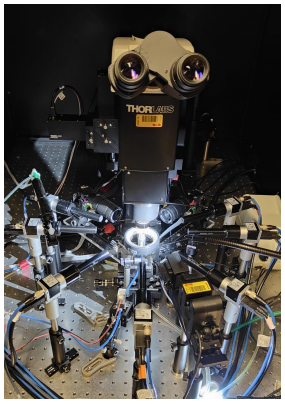


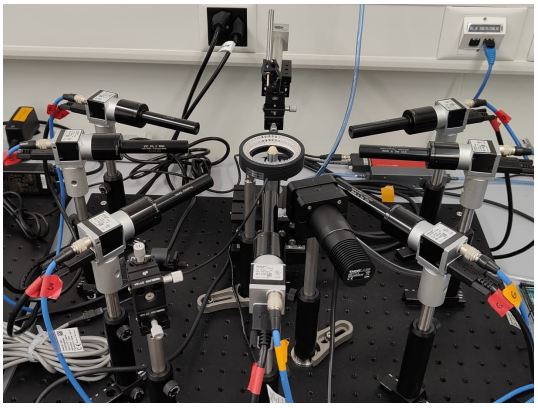
Figure 1: Female and male *Drosophila melanogaster*

1.2 Experimental principle

Many different experimental setups are used to perform experiments on *Drosophila* in the Neuroengineering Laboratory at EPFL. The robotic system here described has been developed to be compatible with the different seven cameras setups present in the laboratory. These setups are shown in Figure 2.



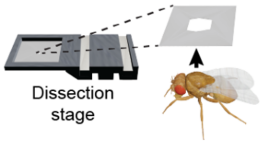
(a) Seven cameras setup with microscope



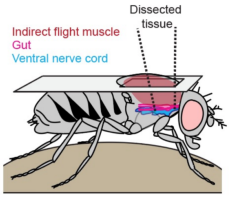
(b) Seven cameras setup without microscope

Figure 2: Seven cameras setup

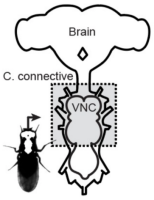
For the setup with the microscope, the fly is first tethered by gluing its thorax to a mechanical element. Then, it is dissected and placed under the microscope in order to record its ventral nerve cord (VNC). The VNC is imaged using a technique called two-photon imaging [6]. This process is illustrated in Figure 3. For the setup without the microscope, the fly is directly tethered by means of a metallic stick to which it is glued.



(a) Mechanical stage [7]



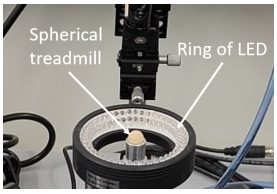
(b) Dissected fly [8]



(c) Ventral nerve cord (VNC) location [8]

Figure 3: Preparation of *Drosophila melanogaster* for two-photon imaging in the seven cameras setup with microscope

In these setups, the fly walks on a spherical treadmill. This spherical treadmill consists in a ball floating by mean of airflow coming from beneath (Figure 4). A ring of infrared LEDs is used to provide the illumination necessary for the camera recordings.



(a) Spherical treadmill and ring of LED

(b) Tethered vinegar fly on the spherical treadmill

Figure 4: Spherical treadmill

1.3 State of the art

In this Section, state of the art techniques for some experiments on *Drosophila melanogaster* are presented.

1.3.1 Feeding

The brain of *Drosophila* shows dynamics at multiple timescales, from the millisecond range of fast voltage or calcium transients to functional and structural changes occurring over multiple days. Therefore, monitoring neural circuits across these multiple timescales in behaving animals is required to relate such dynamics to behavior [9]. Slow changes occur for example due to sleep [10], circadian rhythms [11, 12] or memory consolidation [13, 14].

Thus, a feeding system is necessary in order to conduct experiments over multiple days. Different methods have been used for the feeding of tethered *Drosophila*. They are presented hereafter.

Releasing
Huang, C. et al. [15] release flies in their tube for them to feed in between measurements. A schematic of their experimental workflow is presented in Figure 5.

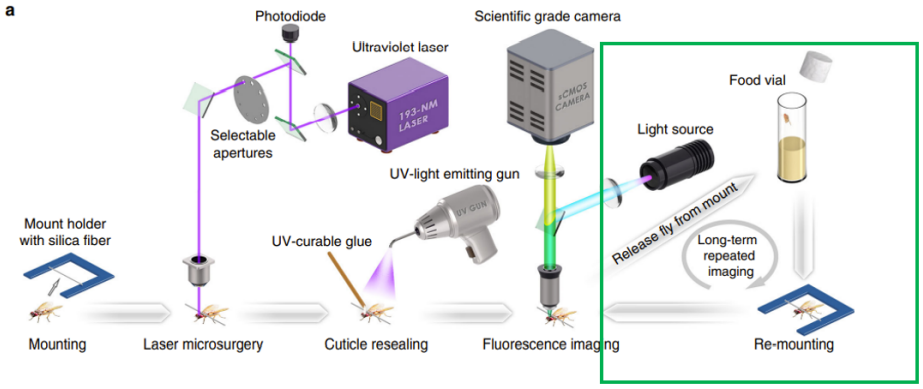


Figure 5: Workflow of releasing flies to feed them [15]

This method has the drawback of having to mount the flies multiple times. In order to do so, they have to be anesthetized each time which implies a recovery time before being able to perform new measurements.

Fixed
Another method, which has been used by Martin, J., L. et al. [16], consists in fixing a feeding element close to fly's proboscis – which is the mouth part of the *Drosophila*. This is illustrated in Figure 6.

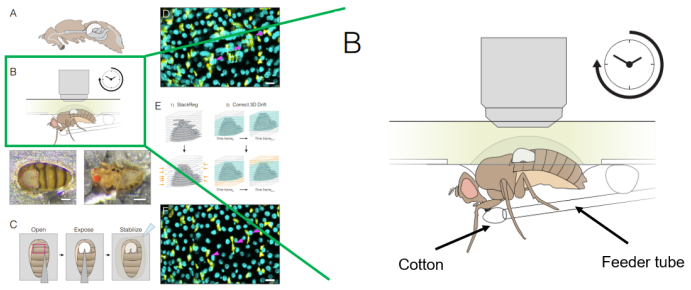


Figure 6: Schematic of the fixed feeding element [16]

This method would not be compatible with a fly walking on a spherical treadmill because it would not allow the fly to move freely on the spherical treadmill.

Motorized
An approach with an active robotic system has been developed by Valle, A., F. et al. [9]. Their system is depicted in Figure 7.

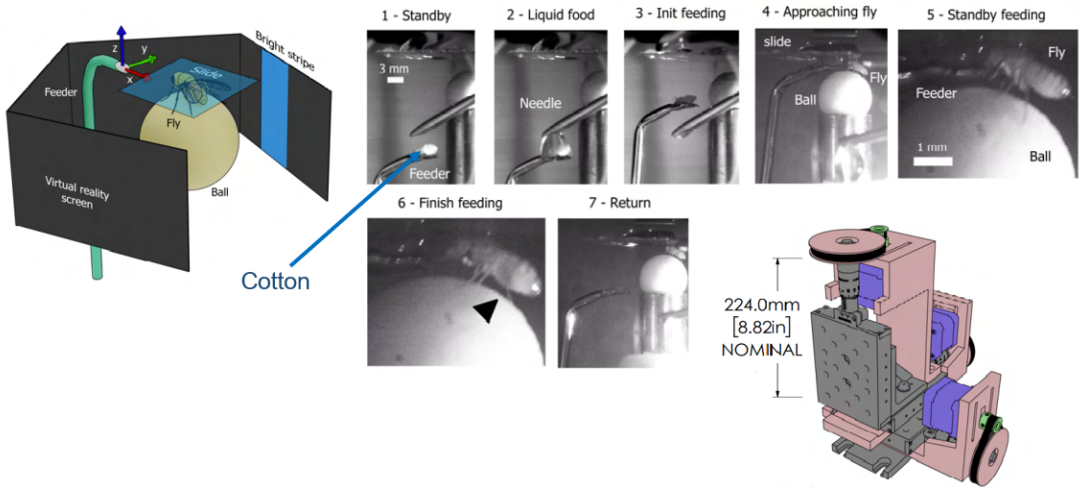


Figure 7: Schematic of the motorized feeding element [9]

Unfortunately, this solution has a too important mechanical bulk and could not fit in the seven cameras setup.

1.3.2 Stimulating

One can stimulate a tethered fly in many different ways. Some examples are presented below.

Moving objects
Hindmarsh Sten T. et al. [17] used a virtual reality (VR) screen with a moving black dot on it. When optogenetically stimulating a group neurons responsible for courtship, the male fly begins to track the dot as if it was a female. An illustration of their setup is shown in Figure 8.

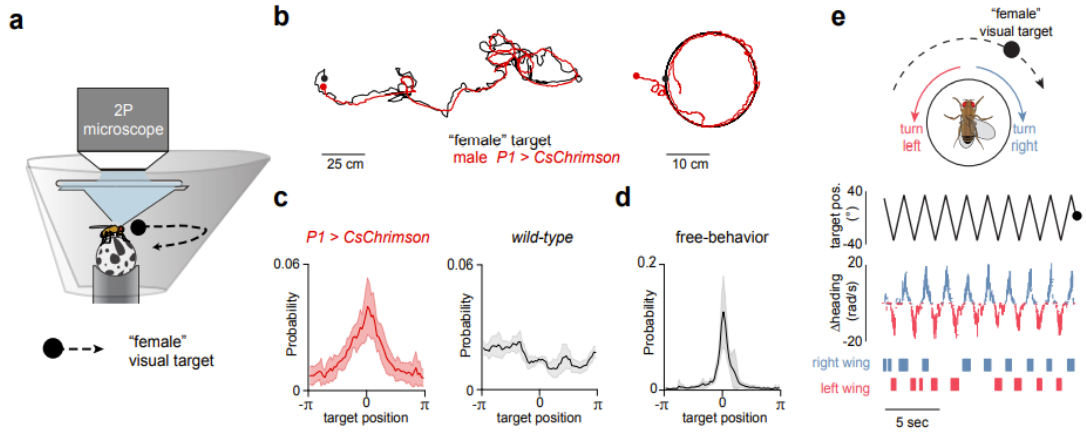


Figure 8: Schematic of the VR screen setup [17]

Moving airflow

A system to move airflow in the tethered *Drosophila melanogaster*'s environment has been developed by Zolin A. et al. [18]. A schematic of their experiment setup is shown in Figure 9

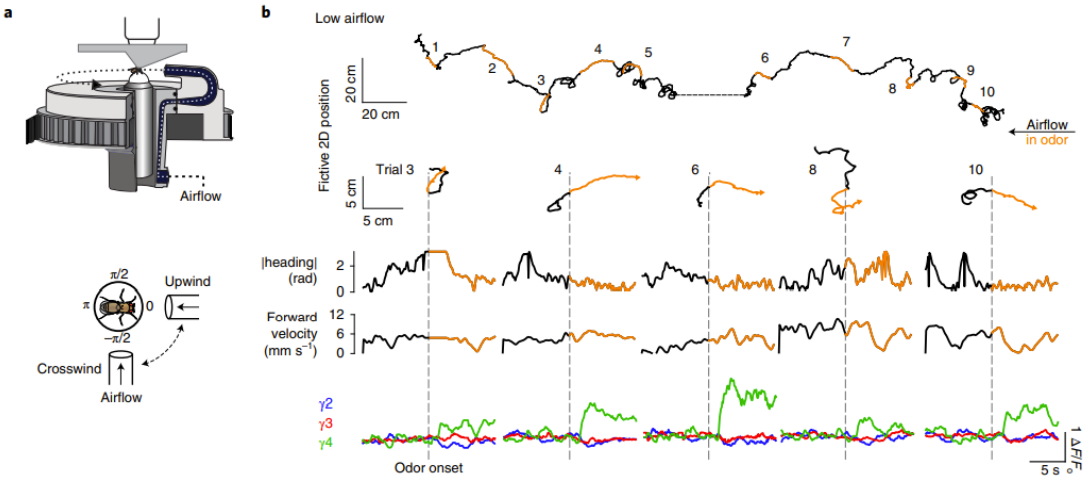


Figure 9: Schematic of the airflow setup [18]

Moving obstacles

Warren, R. et al [19] developed a system (Figure 10) to emulate a fixed obstacle for a mouse running on a wheel. In fact, the obstacles approaches the mouse at the same speed the mouse is running.

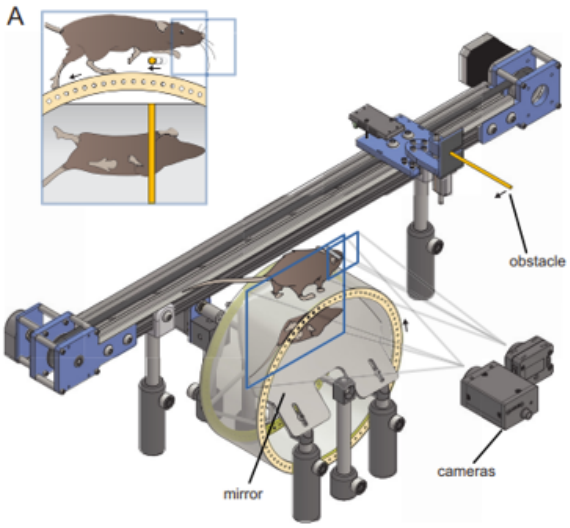


Figure 10: Schematic of the obstacle setup [19]

1.4 Robotic solution

In order to enable the laboratory to perform the above mentioned experiments, a versatile robotic solution has been developed. It is presented in Figure 11.

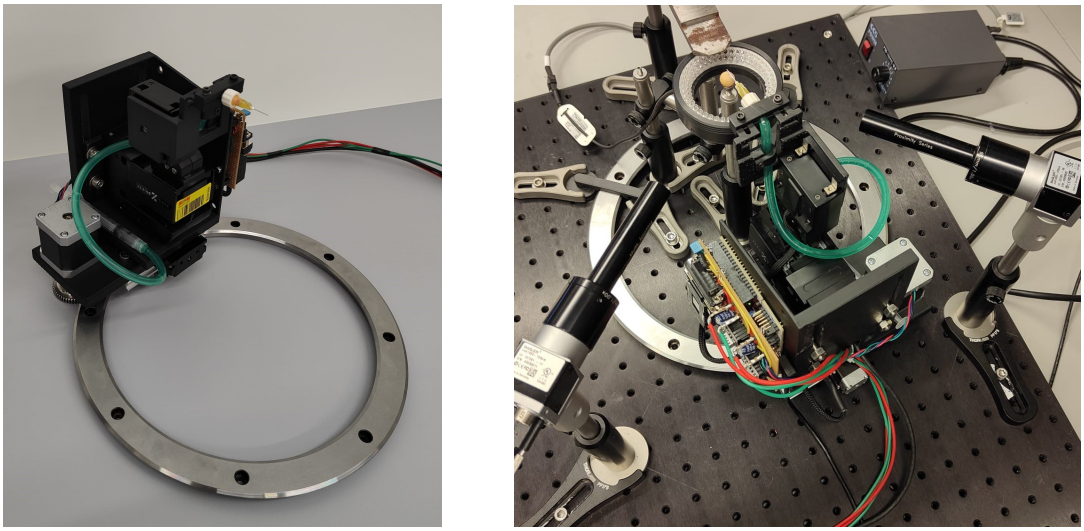


Figure 11: Developed robotic system

This robotic system allows to perform automated feeding of the fly and to move objects in the fly surroundings. The working principle of these two use-cases is presented in Section 9.

It could also be used to move airflow by attaching an appropriate end effector.

Similarly to the moving obstacle used with mice, this robotic system would allow to emulate moving objects with respect to the tethered fly's fixed referential. In fact, by monitoring the tethered *Drosophila*'s speed, an object could be perceived as fixed by the fly if it is moved in opposition to the fly's speed. FitTrac, a method to monitor the fly's velocity by measuring the movement of the spherical treadmill, has already been developed [20].

Another application could be to perform learning experiments:
One could, for example, move an object close to fly's right front limb and, if it touches the object with this limb, it will then receive a food reward which is again brought by the robot.
Another learning experiment could consist in presenting the fly with two food choices. One having an attractive taste and the other a repulsive one. The fly could distinguish between them by their shape or odor for example. By presenting the two choices and moving the objects in the fly's referential, one could assess whether it can learn which one is the most rewarding food source. This kind of experiments is usually performed in an arena in which two photon imaging cannot be performed. The robotic system would allow to perform it under the microscope.

1.5 Main contributions

The contributions of this project are the following:

- A wrapper around the Sensapex Python API [21] for the Sensapex uMp3 micromanipulator [22]; [23];
  - Enabling a new functionality: smooth transitions between commands.
- A stepper motor controller on a STM32F4 microcontroller [23];
  - Communicating with the PC application;
  - Allowing changing commands on the fly;
  - Controlling multiple motors;
  - Taking limit sensor as input.
- The mechanical design of a versatile robotic platform enabling new experiments for tethered *Drosophila melanogaster*;
  - Comparing and selecting existing solutions on the market in terms of costs and performances;
  - Designing mechanical parts to assemble them.
- A closed loop control algorithm;
  - Receiving cameras' frame as input;
  - Processing those frames for object recognition using DeepLabCut Python package [24].
- An open loop control algorithm for trajectory following.

2 Environment

In the previous Section, the experimental principle of the seven cameras setup has been exposed. This setup will be discussed here in more details in order to identify the space available for the robotic solution.

2.1 Seven cameras setup

A 3D model of the seven cameras setup has been developed in Fusion360 [25]. The 3D model is shown in Figure 12. The black circle around the spherical treadmill corresponds to the ring of LED used to illuminate the scene.

This model combines the most constraining elements of the different setups. The cameras' height is chosen as being the lowest one across these setups – the one from the setup without microscope – and the ball mount is the one from the setup with the microscope since it has a larger mechanical bulk than the post used in the other setup. This is done in order to ensure that the system can be used in those different setups. Some elements of the robotic platform are designed to be adjustable such that the solution can be adapted to these different setups.

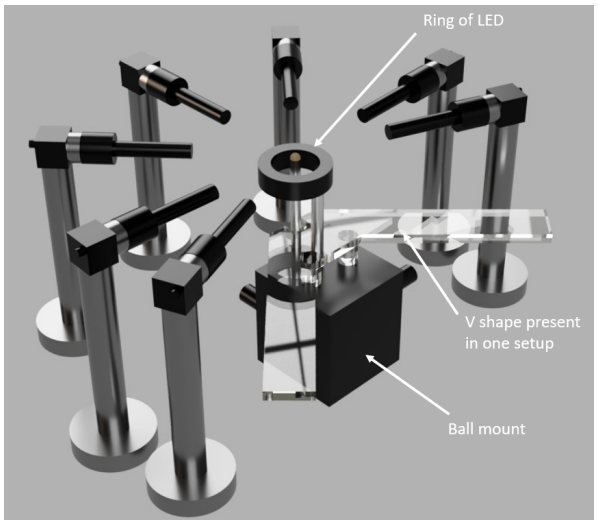


Figure 12: 3D model of the most constraining seven cameras setup. The V shape is used as support for optic flow sensors which are not used in this work

The dimensions of the setup from the side view are given in Figure 13.

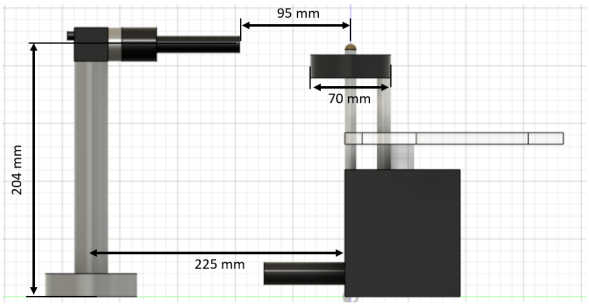


Figure 13: Side view of the seven cameras setup. Illustration of the dimensions

This shows how constrained the available space is.

2.2 Experimental setup

This setup has been replicated in order to develop and test the robotic platform safely. In this test setup, two cameras are used as presented in Figure 14.

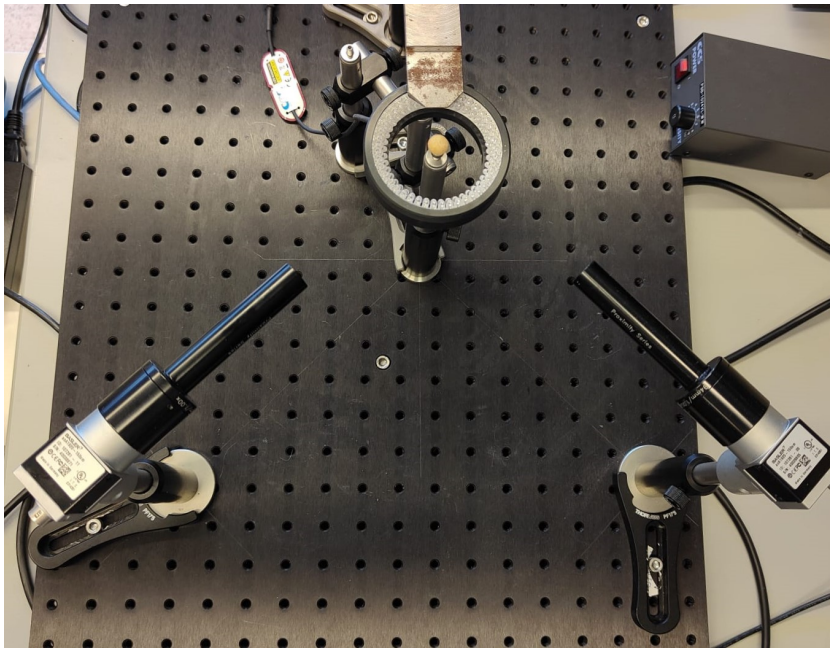


Figure 14: Replicated setup used to develop the robotic system

Contrary to the seven cameras setup, the test setup only allows receiving information from two cameras. Moreover, the fact that the system cannot hit the ball mount from the microscope setup can only be verified from the 3D model. The objective is that future work will integrate the developed robotic solution in the seven cameras setup.

3 Requirements

The environment being defined, the next step is to specify the requirements of the robotic platform. These requirements will be based on experimental needs, mechanical constraints imposed by the environment and the dynamical characteristics needed to interact with the fly.

3.1 Experimental requirements

It would be a great advantage to be able to feed the *Drosophila melanogaster* without having to remove the ball in order not to disturb the fly's behavior.

The resting position of the robotic system should be outside of the field of view of the cameras when no feeding/stimulation is occurring in order not to occlude them.

The robotic platform should be able to span the 2D space and hence two Degrees of Freedom (DOFs) in the horizontal plane are needed. In order to be able to go outside of the field of view of the cameras, a vertical degree of freedom is also required.

Moreover, in order to exit the cameras' field of view, it should have a travel range of at least 35mm along the radial direction to be able to go from the center of the ball to the outside edge of the LED ring before going down.

3.2 Mechanical requirements

The robotic solution should be designed in such a way that it cannot hit the cameras nor the ball mount in its workspace. This reduces the complexity of the controller which does not need to have a model of the environment and take care of avoiding those elements.

3.3 Dynamical requirements

The forward speed a freely behaving vinegar fly can reach is about 40 mm/s [26] as shown in Figure 15.

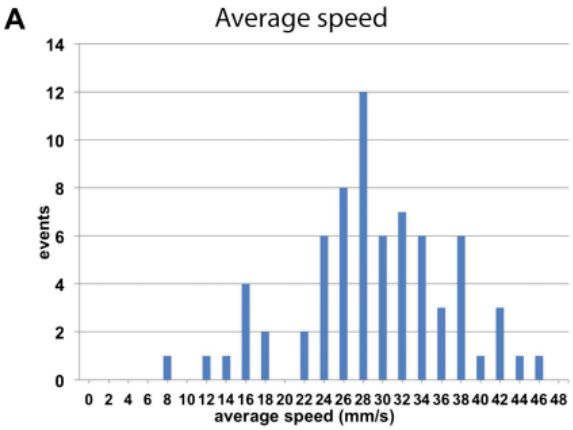


Figure 15: Forward velocity in freely behaving flies [26]

Creamer M. S. et al. [27] have performed experiments in tethered flies in which the maximal forward velocity reached by the flies is about 8 mm/s. The velocity of tethered flies is shown in Figure 16.

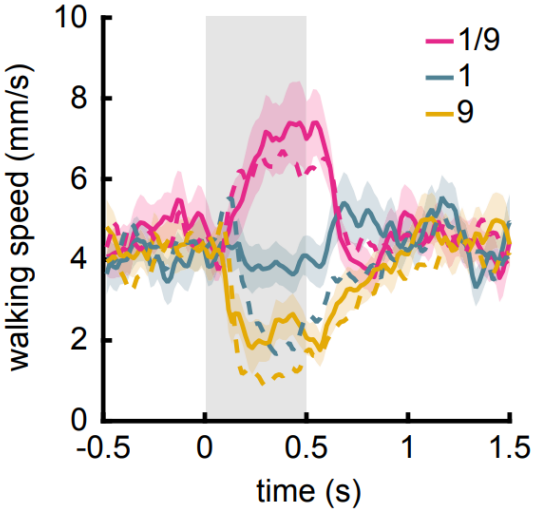


Figure 16: Velocity profile of tethered flies [27]

Therefore, the speed that the motors should be able to reach in order to move objects in the tethered fly referential should be at least of 10mm/s. The targeted speed is then defined to be 30mm/s.

Concerning the resolution of the robotic system, it is arbitrarily chosen as being at least of 10µm. This corresponds approximately to 1/100 of the fly width.

4 Design

According to the defined requirements, the design of the robotic system is presented.

4.1 Cartesian micromanipulator

In order to have the three DOFs required with the needed precision while conserving a compact form factor, a Cartesian micromanipulator seems to be the best suited solution. In fact, Cartesian robotic systems (Figure 17) have the three translational degrees of freedom and micromanipulators are known for their low mechanical footprint and their high precision.

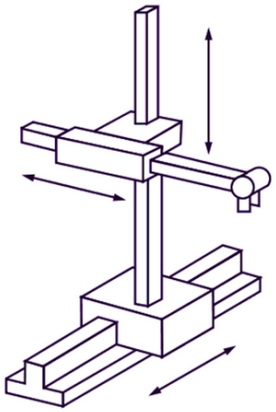


Figure 17: Illustration of a Cartesian robotic system [28]

An extensive research of the solutions available on the market has been made. The four solutions that best meet the defined requirements are presented and compared hereafter.

4.1.1 Sensapex uMp3

The Sensapex uMp3 micromanipulator [22] is already available in the laboratory. It is shown on Figure 18.

Figure 18: Sensapex uMp3 micromanipulator [22]

Its main characteristics are the following:

- Travel range along each axis: 20mm;
- Maximal speed along each axis: 5mm/s;
- Resolution: 5nm;
- Maximum encumbrance in resting position: 126.5mmx90mmx48.5mm;
- Python API available;
- Cost (micromanipulator and PC communication controller): 6870 €

This Cartesian manipulator meets the requirement for most aspects. However, one can note that the speed is not great enough and that its resolution is much higher than required. Therefore, alternatives are presented and compared.

No competitive solutions have been found on the market. They had either a too important mechanical footprint or their prices would not justify investing in another micromanipulator.

4.1.2 Siskiyou

A representative example of a solution having a too important mechanical foot print is the Siskiyou MX7600 Series Manipulator [29] (Figure 18). In fact, its total height is 206.5mm, which is more than the height of the cameras.

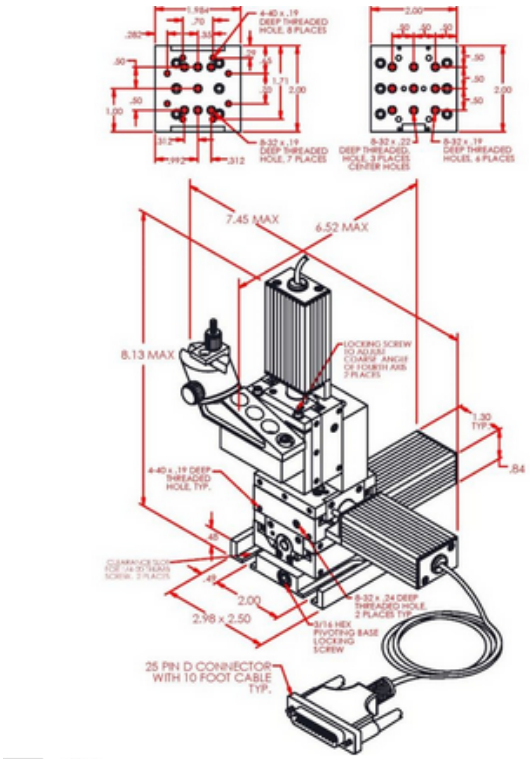


Figure 19: Siskiyou MX7600 Series Manipulator [29]

Many of the solutions on the market were designed with such external motors making their mechanical footprint too important.

4.1.3 Trio

More compact solutions were found such as the Trio-245 [30] which is shown in Figure 20.

Figure 20: Trio-245 manipulator [30]

Its main characteristics are the following:

- Travel range along each axis: 25mm;
- Maximal speed along each axis: 3mm/s (while homing);
- Resolution: 100 nm;
- Maximum encumbrance in resting position: 140mm x 190mm x 102mm;
- USB interface for computer control;
- Cost (micromanipulator and PC communication controller): 7095 €

This micromanipulator has the advantage of having a greater traveling range but its cost would not justify to invest in a new micromanipulator.

4.1.4 Physik Intrumente

The approach of combining linear stages has also been considered. However, this lead most of the time in a too big robotic system which could not fit inside the constrained environment of the seven cameras setup.

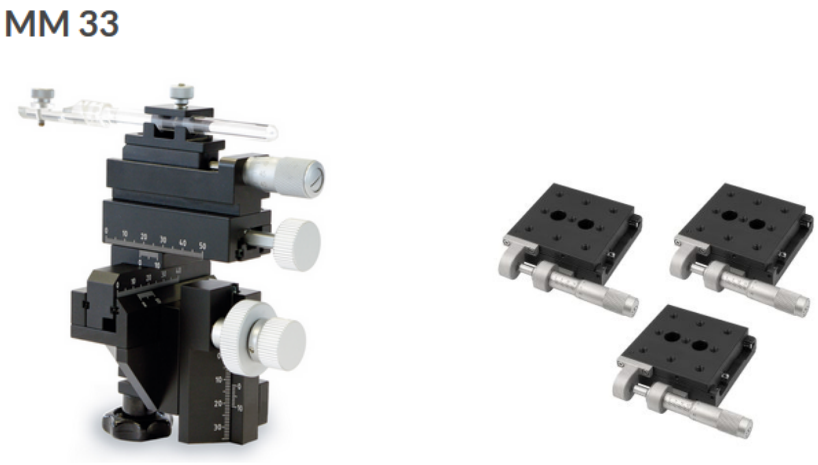
A linear stage with a small encumbrance could be the Physik Instrumente (PI) L-402 Miniature Linear Stage [31]:

Figure 21: PI L-402 Miniature Linear Stage [31]

However, the cost of the system combining three of these – including the controllers – would be about 10000 CHF.

4.1.5 Manual Stages

The option of motorizing manual micromanipulator such as the Marzhauser MM33 [32] (Figure 22a) or individual linear stages like the M-423-MIC [33] (Figure 22b) has also been studied.



(a) MM33 manual micromanipulator [32] (b) M-423-MIC linear stage [33]

Figure 22: Manual solutions

Motors would then need to be attached to the system making it complex to achieve a compact solution. Moreover, the number of cycles such systems could sustain at the high motorized speed is not clear.

4.1.6 Chosen solution: Sensapex uMp3

For the above mentioned reasons, the Sensapex uMp3 has been chosen. The design of the robotic platform is thought such that the Sensapex uMp 3 can be quickly removed if it needs to be used as a standalone for other experiments.

4.2 Updated design idea

The Sensapex manipulator is then chosen to be placed facing upward such that it fits between the cameras and the ball mount as illustrated in Figure 23.
None of the found solutions, including the uMp3, have a sufficient velocity in the horizontal plane nor a long enough travel range. Therefore, the micromanipulator will be responsible for fine movements and additional DOFs enabling faster movement over a longer travel range need to be added.

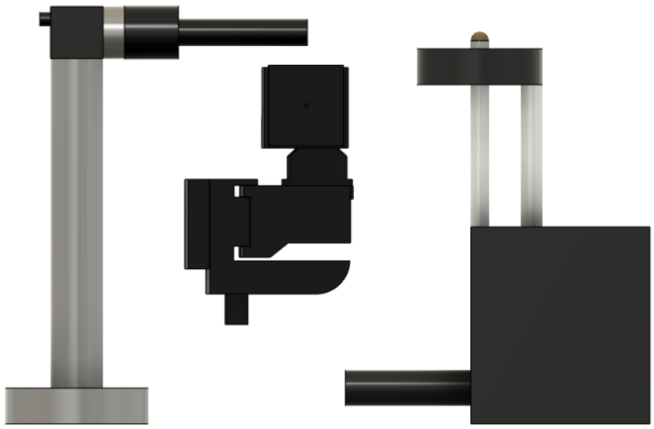


Figure 23: Positioning of the Sensapex uMp3 manipulator

And these additional DOFs will consist in a translation along the radial direction and a rotation about the post holding the ball, as illustrated in red and green on Figure 24 respectively. Their implementation is discussed below.

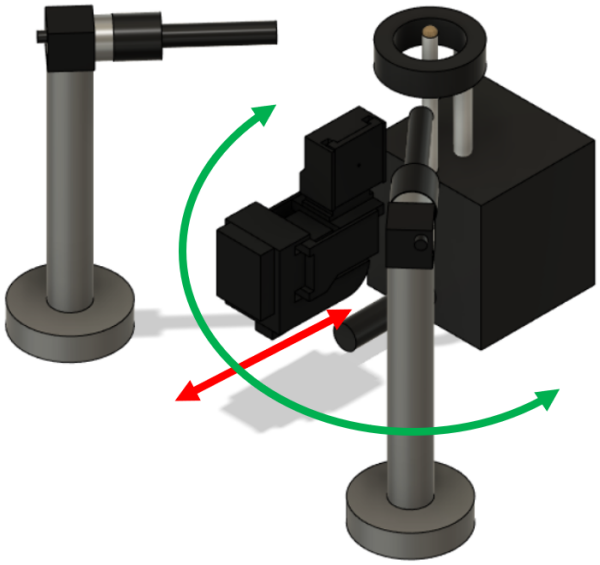


Figure 24: Illustration of the additional degrees of freedom

4.3 Translation stage

For the translation, more than fifteen solutions have been compared. The chosen solution is the Zaber LSM25B-TA4 linear stage [34] (Figure 25).

Figure 25: Zaber LSM25B-TA4 linear stage [34]

Its main characteristics are the following:

- Travel range: 25.4 mm;
- Maximal theoretical speed (cannot reasonably be reached on such a short travel range) : 104 mm/s;
- Resolution: 2.4384 mm per motor revolution;
- Total length: 128mm;
- Standard stepper motor connection;
- Limit switch for homing;
- Cost : 1370 CHF.

It offered the best compromise in terms of speed, travel range, resolution, compactness and price.
The uMp3 will placed on top of that stage and this linear stage will be placed onto the rotational stage presented below.

4.4 Rotational stage

Concerning the rotational DOF, it is important to note that it needs to have a big enough aperture to allow the ball mount to fit. Moreover, the solution cannot consist in a full circle because of the microscope which stands behind the ball mount.

Therefore, a solution using a rail as shown in Figure 26 seems the most appropriate. In fact, a rail provide an important aperture compared to most rotation tables (Figure 27). In addition, contrary to rotation tables, it can easily be cut in order to obtain just a portion which can fit in the setup.

(a) Rotational stage concept 1

(b) Rotational stage concept 2. A motor driving a pinion on the curved gear rack would be added

Figure 26: Rotational stage concepts

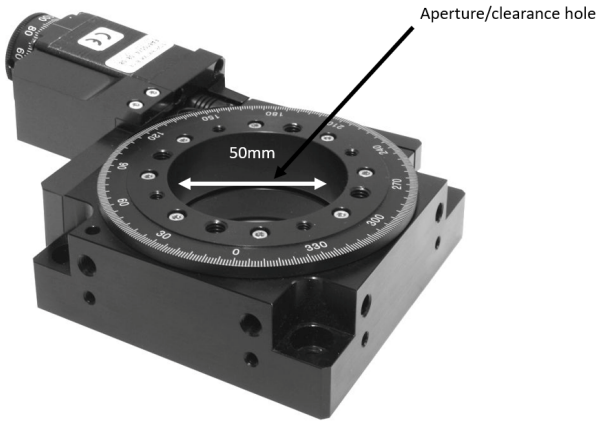
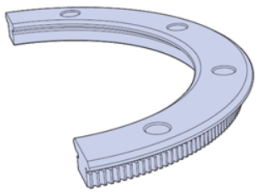
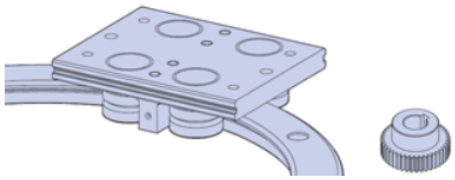


Figure 27: Rotation table with a 50mm aperture [35]

The selected solution is a curved rail with gear on the outside, a carriage and a pinion provided by Hepcomotion [36]. It has been chosen because it was the only manufacturer which could deliver in Europe and provide all the parts – except for the motor and a way to attach it. In fact, having parts from the same manufacturer ensures that all elements are compatible. The system is shown in Figure 28.



(a) Hepcomotion curved rial and gear



(b) Hepcomotion curved rail, carriage and pinion

Figure 28: Rotational stage concepts

A stepper motor will be attached to the carriage to drive the pinion.

4.5 Motors

Now that the mechanical elements are selected, the motors need to be chosen. The motors of the Sensapex uMp3 and the Zaber LSM linear stage are already included.

For the rotational stage, a 42BYGHW811 bipolar stepper motor is chosen. This is a standard stepper with 200 steps and a rated current of 2.5 A.

4.6 Motor controllers

Concerning the motor controllers, the Sensapex manipulator is controlled from the computer through Ethernet using their proprietary controller. For the stepper motor of the Zaber LSM, their proprietary controller – which is 48V – is proposed. It is however expensive (713 CHF) and would not allow to control the stepper motor needed for the rotational stage.

Therefore, another controller will be used. There are solutions which can directly communicate with the PC through USB such as the Phidgets [37]. However, no controller capable of taking the limit sensor input and having a rated voltage close to the one of the Zaber controller has been found.

A common solution would then be to use an Arduino [38], which offers many different libraries. Unfortunately, the Arduino stepper library [39] does not allow to change commands on the fly. This is however needed in order to move objects in the fly referential or to have a feedback controller which can change the command at each time step. Another library called AccelStepper [40] enables non-blocking commands. Unfortunately, it cannot reach frequencies above 4kHz according to their version historic.

Given the fact that the linear stage's stepper motor has 200 steps per revolution and that one revolution corresponds to 2.44mm, a step is equivalent to 12µm. By using microstepping [41], this resolution can be greatly increased. In fact, if one sets a relatively small microstepping of 1/4 of a step, one revolution then corresponds 800 steps, which makes a step equivalent to 3µm. With this microstepping setting, one needs a frequency of 9.836 kHz to reach the desired 30mm/s. For this reason, this library would not be suitable.

Based on the findings above, a STM32F4 microcontroller (MCU) [23] is chosen. The PC communication and the motor signal generation will be implemented using the Hardware Abstraction Layer (HAL) libraries. The selected model is the STM32 Nucleo-64 development board with STM32F401RE MCU [42] which provides 64 pins, a powerful CPU running at a maximum frequency of 84 Mhz and a floating point unit (FPU). All these characteristics make it a good candidate for developing an application without being too much constrained by the computational capabilities of the microcontroller, nor by the number of available pins, for a cost of 19 CHF. Furthermore, this microcontroller's timers output frequency can go up to 8Mhz in low speed output mode which is more than required. The STM32 Nucleo-64 development board with STM32F401RE MCU is presented in Figure 29.

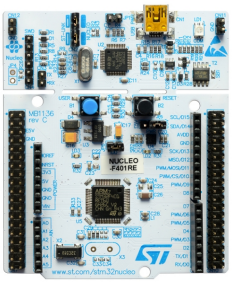


Figure 29: STM32 Nucleo-64 development board with STM32F401RE MCU

4.7 Motor drivers

The DRV8825 stepper driver [43] is chosen since it allows microstepping and can deliver up to 1.5A per phase without additional cooling – the Zaber and the 42BYGHW811 stepper motor being rated for 0.6A per phase and 2.5A respectively. Moreover it can accept up to 45V as power supply which makes it close to the voltage used by the Zaber proprietary controller.

4.8 Power supplies

The Sensapex uMp3 comes with its own power supply. For the Zaber stage, a LCE80PS42 power supply is chosen since it delivers 42V – in order to be close to the 48V of the proprietary controller – with a maximum current 1.9A (when the two phases of the Zaber stepper motor are 100% active, they need 1.2A).

For the rotational stepper motor, a LCE80PS24 is selected. This power supply delivers 24V with a maximal current of 3.3A. The maximal current of the stepper driver being set to 1.3A per phase – which is close to the current above it will start overheating – this power supply can provide enough current. The process of how to set this current limit is explained in [44].

4.9 Cameras

In order to have a closed loop controller, feedback about the end effector position is needed. This feedback will be provided by the Basler acA1920-150µm [45] cameras used in the seven cameras setup.

5 Developed robotic system

Following the above mentioned design choices, the developed robotic platform is discussed.

5.1 Mechanics

The main design features of the parts presented hereafter are discussed. These parts are 3D printed using the SLA formlabs Form 2 [46] with the standard black resin v4.

The bracket, shown in Figure 30, is used to mount the Sensapex uMp3 onto the linear stage. This bracket is designed with horizontal and vertical rails such that the position of the uMp3 can be adjusted to the different setups. It also includes pass through holes in order to attach cables with cable ties.

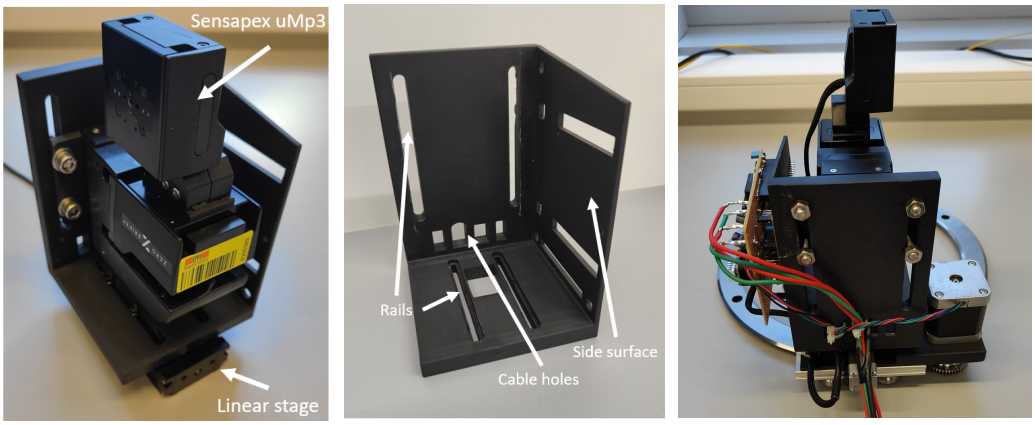


Figure 30: Bracket

The side surface confers a greater mechanical rigidity to the structure. This surface is then used to mount the electronics on the side as shown in Figure 31.

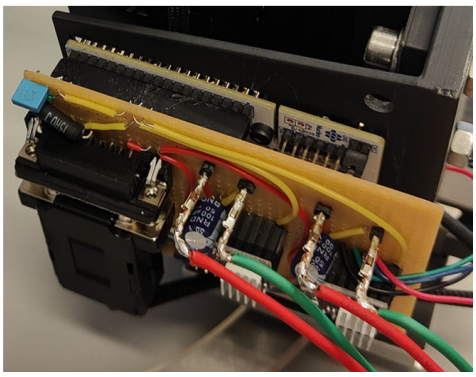
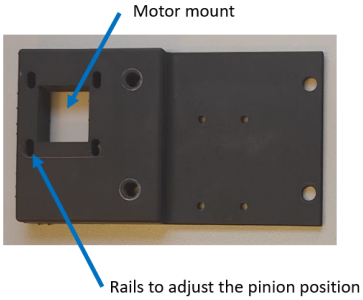
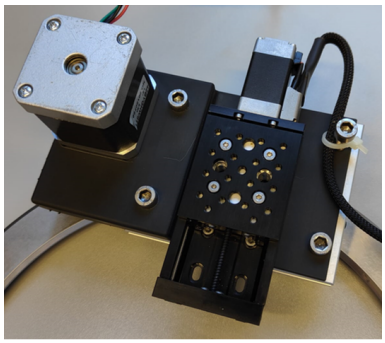


Figure 31: Electronics mount

The linear stage is mounted onto the rotational carriage by means of a platform. This platform also serves to fix the rotational motor, to which the pinion is attached. The position of the motor can be adjusted, by means of rails, in order to place the pinion correctly with respect to the gear rack. This element is illustrated in Figure 32.



(a) Rotational platform



(b) Rotational platform mounted

Figure 32: Rotational platform

The end effector – which is primarily designed to hold a needle for the feeding use case but can used for holding other objects – is fixed onto the Sensapex 33. It also has rails to adjust the position of the needle without having to move the entire bracket.

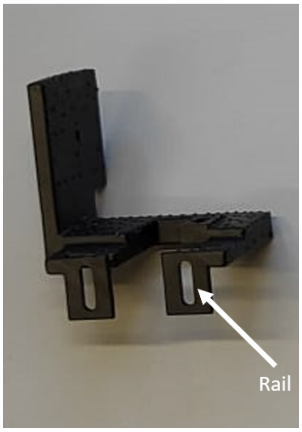


Figure 33: End effector

The rail of the rotational stage is fixed onto the optical breadboard (Figure 34). The fixation elements also serve as mechanical stops for the rotational stage.

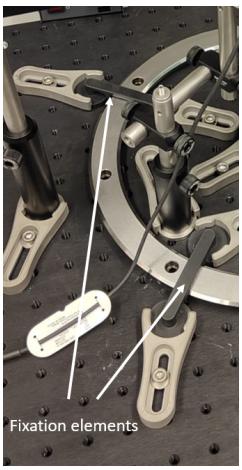


Figure 34: Mount robotic system on base plate

The complete robotic system with the end effector used for feeding – which consists in a needle, a tube and a syringe – is shown in Figure 35.

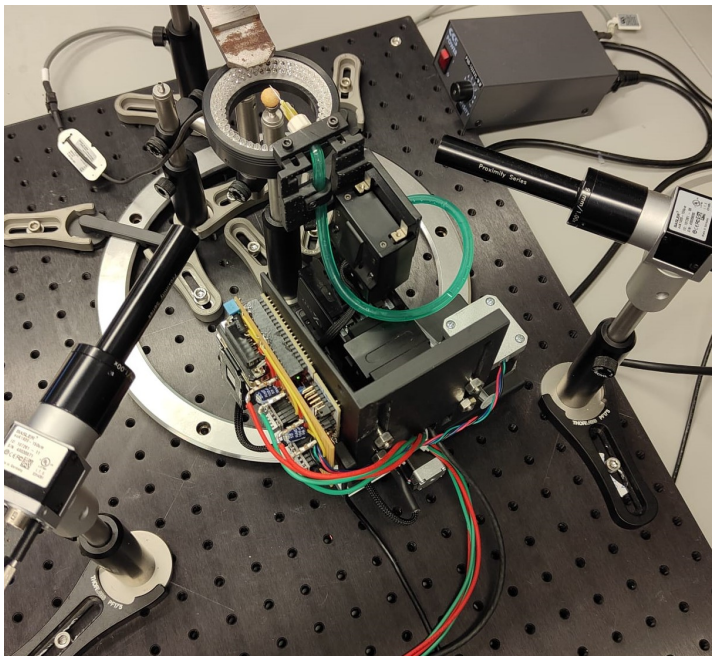


Figure 35: Robotic system

Using the 3D model of the seven cameras setup and testing on the test setup ensures that the robotic system cannot hit the ball mount nor the cameras over its whole working space.

Note that a portion of the rail could be cut in order to make it fit in the seven cameras setup.

The assembly protocol is presented in Appendix A.

5.2 Electronics

The electronics is mounted on a FR2 prototyping board since it enables fast prototyping.

The MCU pins attribution is shown in Figure 36.

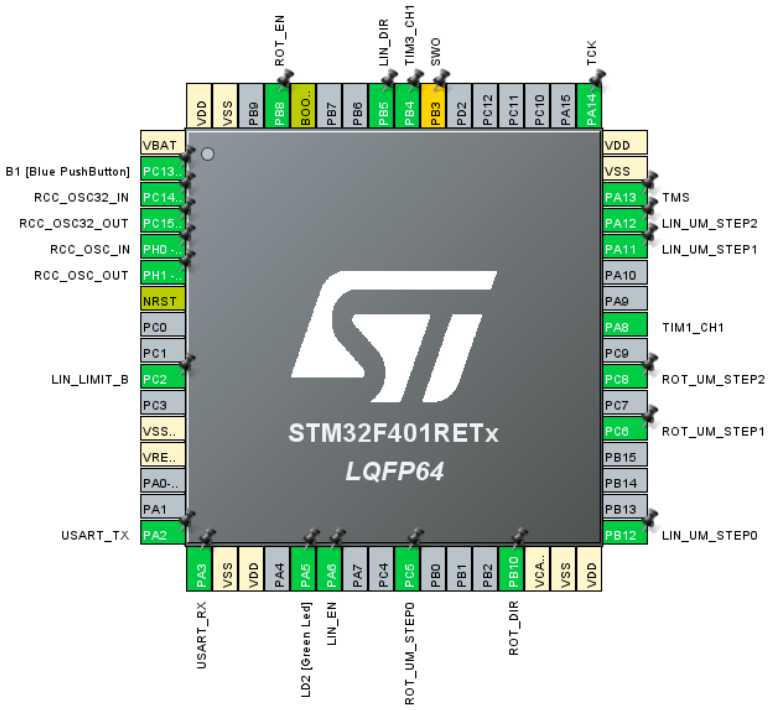


Figure 36: MCU pins attribution

They are connected to the corresponding pins of the stepper drivers. The power supplies are connected to the inputs of the stepper drivers and the outputs of the drivers are connected to their respective motor. Bulk capacitors of 100µF are used close to the stepper drivers power input in order to deal with the strong current variations.

The limit sensor of the Zaber linear stage is connected through the DB-15 connector. The output of this sensor being very noisy, it is filtered through a RC filter (15Ohm, 0.1µF).

The prototype board, with the above mentioned elements, is shown in Figure 37.

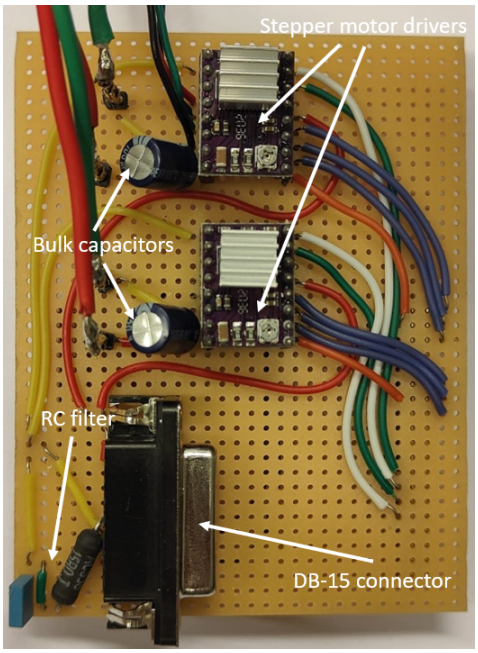


Figure 37: Prototype board

The connection scheme is presented in Appendix B.

5.3 Costs

The costs of the robotic solution are the following:

- Sensapex uMp3 and its controller: 6870 €
- Zaber linear stage: 1380 CHF;
- Rotational stage: 644 €;
- 42BYGHW811 stepper motor: 15 USD;
- Microcontroller: 19 CHF;
- 2x Stepper motor drivers: 10 CHF;
- Power supply LCE80PS42: 60 CHF;
- Power supply LCE80PS24: 60 CHF;
- Prototype board and small electronics components: 20 CHF
- 3D printer resin: 150 CHF.

The total cost of the solution is thus about 9300 CHF.

6 Low level controller

The low level controller is responsible for controlling the motors and for the communication with the Python PC application.

6.1 Sensapex uMp3 controller

The Sensapex low level control is managed by their proprietary controller which communicates with the PC through an Ethernet cable. The commands are sent by the PC application using the developed wrapper around their Python API [21].

6.2 Stepper motor controller

The software of the STM32 MCU is developed using STM32CubeIDE [47]. This IDE has the STM32CubeMX package [48] which allows to initialize the different microcontroller peripherals through a graphical user interface (GUI). The code is programmed in C using the HAL functions.

It is important to note that after the code generation from CubeMX, the initialisation of the Direct Memory Access (DMA) has to be placed before the one of the Universal Asynchronous Receiver Transmitter (UART). This is an issue with the automatic code generation which has been discussed on forums [49]. This has been corrected in the new version of CubeMx (6.4) and the way to solve the issue is described in [50]. This correction has not been made yet.

A general overview of the controller functioning is shown in Figure 38. The PC application sends a command to the MCU which needs to store it in memory using DMA, decode it and send the appropriate step signals to the motors. The limit switch of the linear stage stepper motor is taken into account when homing the motor. The MCU also sends messages to the PC when it is asked to communicate its position or to acknowledge the homing of the motor. This communication also takes place using DMA.

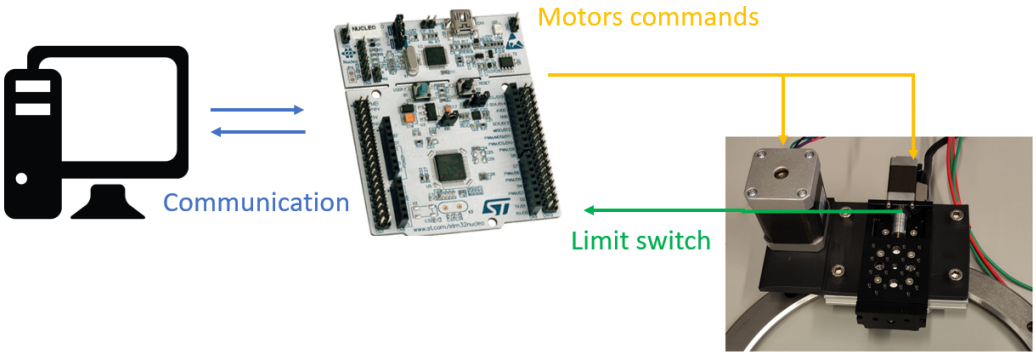


Figure 38: General overview of the low level controller

Each process has given priority and higher priority tasks can preempt on lower ones. Those priorities are organized from highest to lowest priority as follows:

Priority	Subpriority	Process
0	0	Falling edge interrupt: limit sensor
1	0	Timer 1: rotational stage motor
1	0	Timer 3: translational stage motor
2	0	Stream interrupt: DMA receive
2	0	Stream interrupt: DMA send
3	0	Timer 2: PC communication (decoding)

Table 1: Priorities of the different processes of the low level controller. From highest priority to lowest priority

The different processes running when the motor is idling are shown in Figure 39. In this figure, the preemption of the higher priority processes is illustrated.

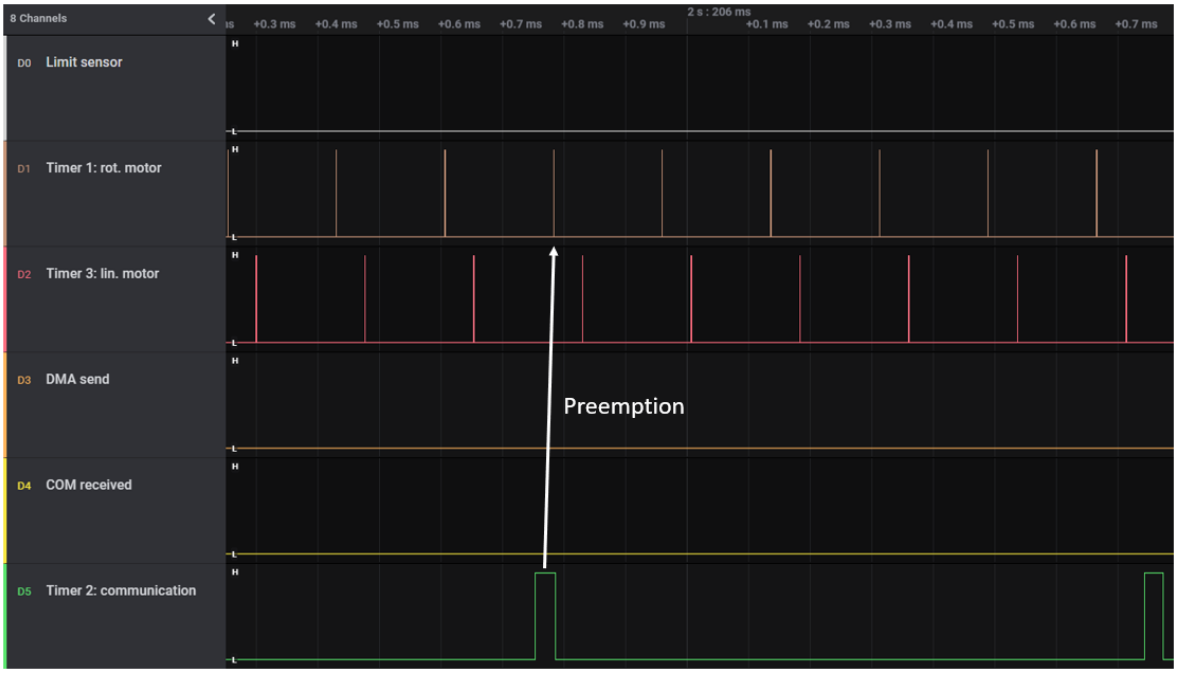


Figure 39: Processes running on the MCU in idle state

6.2.1 PC communication

The first functionality which needs to be implemented is the ability for the MCU to communicate with the PC.

The developed communication allows for variable length messages which has the benefit of enabling to send different command types without having to pad the end of the message with a given character. This also allows to quickly implement new commands without having to take care of the specific length the message should have (it should just not have a greater length than the buffer size). The communication is also made such that the messages can arrive at unknown timings without having the MCU program waiting.

This allows the PC and the MCU programs to be as independent as possible and limits the possibility of errors when one of the two is changed.

Protocol
The communication protocol is defined as follows:
The buffer is 20 bytes long (this can be changed if needed) and the first character is the identity character (ID) of the motor ('Z' for the Zaber linear stage or 'R' for the rotational stage), then comes the command mode ('R' for relative movement in numbers of steps or 'V' for velocity control, 'G' for getting the motor position, 'A' for activating/deactivating the power stage of the stepper drivers, 'N' for reinitializing the MCU). Then, for the 'R' and 'V' modes, comes the direction of the movement ('+' or '-') and finally the command content. The command content for the 'R' mode is the '<number of steps>,<speed>' (speed can be omitted and the default maximal speed is used). For the 'V' mode, the command content is simply the '<speed>'. The end of the command is indicated by the '\' character followed by the 'r' character. These termination characters allow for variable length commands.

This protocol is summarized in the table below:

byte	19	18	17	16 ... 0
content	ID	mode	direction	message content
input	'Z': Zaber linear stage / 'R': rotational stage	'R': relative movement in steps / 'V': for velocity control / 'G': for getting the motor position / 'A': for activating/deactivating the power stage / 'N': for reinitializing the MCU	direction: '+' / '-'	message content: in mode 'R': <number of steps>,<speed> (speed can be omitted) / in mode 'V': <speed>. The speed is the period between steps in clock count. The end of the command is indicated by '\r'

Table 2: Communication protocol

Note that the <speed> command content is in reality the minimum period in clock count of the motor timer between two steps of the stepper motor which relates to the speed. The design choice here is to keep the MCU software in the unit of steps and period/frequency such that it can be used regardless of the robotic platform. On the other side, the PC application is aware of the robotic platform and is responsible for converting the desired speed in those units. For the same reason, the position returned by the MCU is a number of steps.

Buffer management
The communication takes place over the USB port of the MCU using the UART module. The information is received and transmitted by DMA.

The commands received from the PC are processed as follows:
The messages received through the UART module are put in a circular buffer by DMA. At 1000 Hz, in the interrupt routine of timer 2, the circular buffer is read from the last received command backward. If a new command is found, it is put in one of the three tampon buffers of the corresponding motor. This buffer is selected as not being the one containing the newest command nor the one being eventually read from another process/interrupt routine. This buffer is then indicated as being the one with the newest command and the new command flag of the corresponding motor is raised. This ensures that processes with lower priority or higher priority, regardless of their frequency, can access the information without it being changed while they are processing it and that the newest command is always available.

If during the backward search another command for the same motor is found, it is disregarded. The termination character of the commands read or disregarded is changed from 'r' to 'z' in order to distinguish them from new commands. Therefore, if the PC sends commands at a higher rate than the reading frequency of the MCU, only the newest commands are applied. Note that the command frequency of the PC should ensure that the circular buffer is not completely filled in between two MCU

readings, otherwise the backward search would failed. Currently, the circular buffer is 100 bytes long. Hence, the PC communication should be less than 5 times faster with 20 bytes commands each time.

This process is illustrated in Figure 40.

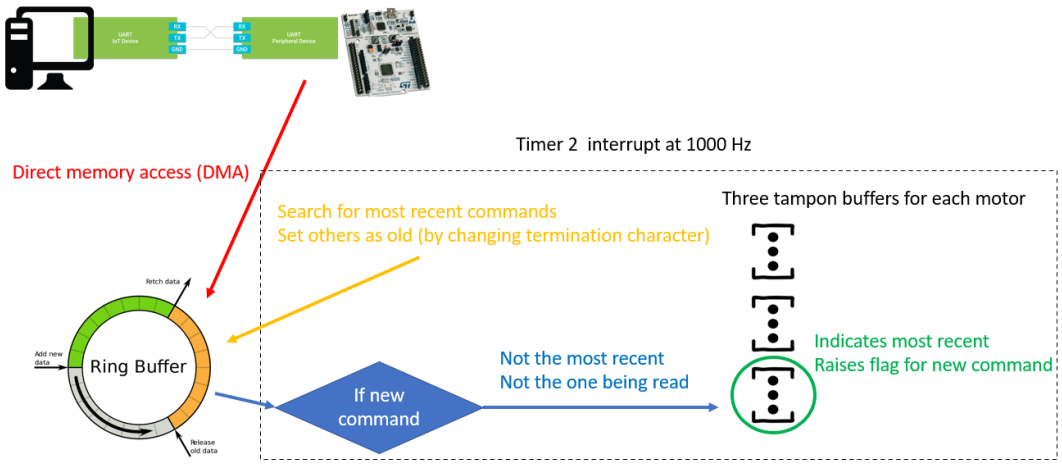


Figure 40: General overview of the reception of the commands

In the current implementation, the commands are processed in the same interrupt routine as the one used for reading the buffer. Therefore, the robust and flexible features of the communication are not required. Nonetheless, it was kept as is since this implementation allows other processes to access the information and hence easy design changes if needed.

After the backward search, the commands are then processed: the content of the command is extracted and put in the stepper_motor_decode structure of the corresponding motor. This structure also has the three buffers principle used above such that the newest information is always accessible. If the MCU is asked to communicate the position of the stepper motor or to acknowledge the homing of the motor, this will also take place in this process. This processing is not done in the motors interrupt routine so that they can be fast and reach high frequencies.

Finally, when the command are processed, the "command ready" flag of the corresponding motor is raised. This command will then be updated in the stepper motor interrupt routine presented in the next Section. It is important to note that the commands are not cumulative, the newest one is always applied and the old ones are disregarded.

6.2.2 Motor signals generation

The stepper motor controller needs to be able to change the command on the fly (non-blocking command). In fact, this is needed if one wants to move an object with respect to the *Drosophila*'s movements.

The step signals need to be generated according to the command. This is done by generating a trapezoidal velocity profile in the interrupt of the timer 1 for the rotational motor and timer 3 for the Zaber motor. This velocity profile is presented in Figure 41.

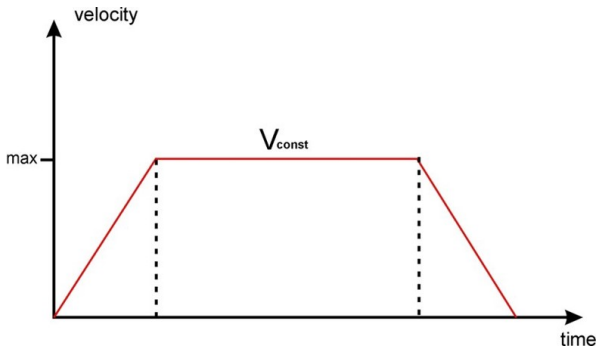


Figure 41: Trapezoidal velocity profile

Steps signal generation
In those timers, the frequency of the steps is controlled. This steps signal generation is done using the timers in center aligned mode because this mode is less prone to noise and consumes less current [51]. The timer 1 is configured (using the RCR register and the counter polarity) such that the interrupt, in which the frequency is updated, happens at each counter underflow. Preload registers are used such that the interrupt has a longer time to calculate the required frequency.

Since the timer 3 has no RCR register, it is emulated with a boolean value. This makes the two timers behave in the same way and the logic can be the same for both motors.

The center aligned counter mode is illustrated in Figure 42. The flash symbol represents the underflow interrupt in which the frequency of the next step is computed and the orange arrow symbolizes the moment this command is effective – the value in the preload registers is applied.

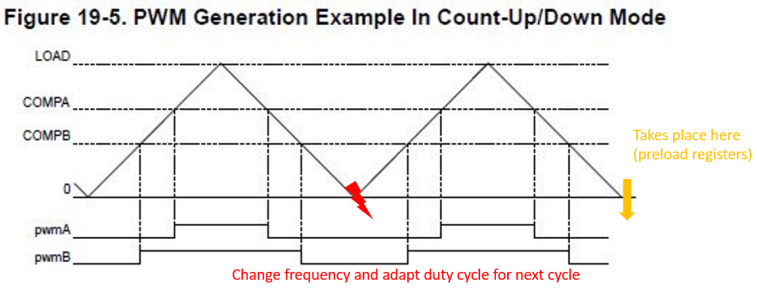


Figure 42: Center aligned mode counter for timer 1 and 3

In the interrupt occurring after each step, the period to the next step is calculated.

For the ‘R’ (relative movement in number of steps) mode:
the number of steps to the goal is calculated. If it is greater than the number of steps the motor has been accelerating for and the commanded speed is not exceeded, the motor continues to accelerate. Then if the maximal speed is reached and the number of accelerating steps is lower than the number of steps to the goal, the motor will stay at that speed. As soon as the number of steps to the goal or to the end of the travel range is smaller or equal to the number of accelerating steps, the motor will decelerate and then stop. This logic generates a trapezoidal acceleration profile and allows command changes on the fly.

For the ‘V’ (velocity) mode :
the motor accelerates/decelerates to the commanded velocity. As soon as the number of steps to the end of the travel range is smaller or equal to the number of accelerating steps, the motor will decelerate and then stop. This implementation also allows for command changes on the fly.

Trapezoidal velocity profile

Because the period between the steps changes according to the calculated frequency, a constant increase in frequency at each step would not result in a constant acceleration. In fact, the speed would have an exponential trend when accelerating. This is not desirable since the motor has to accelerate faster at higher speed when less torque is available. The speed profile generated by this constant increase of frequency at each step is shown in Figure 43.

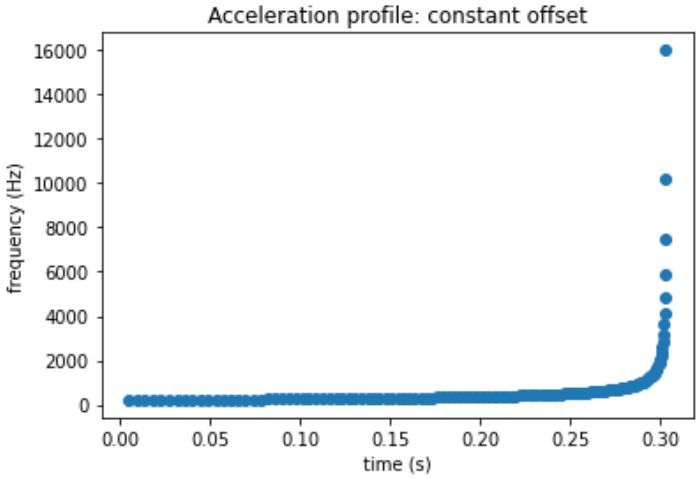


Figure 43: Acceleration profile with a constant decrease in the steps period

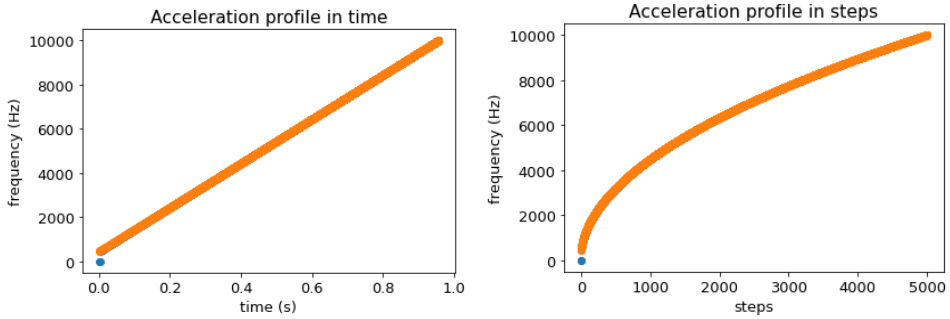
Therefore, the frequency increase has to take into account the period between the steps. Currently, the following formulas are used:

$$frequency = frequency + previous\_period\_seconds \times acceleration\_coeff \tag{1}$$

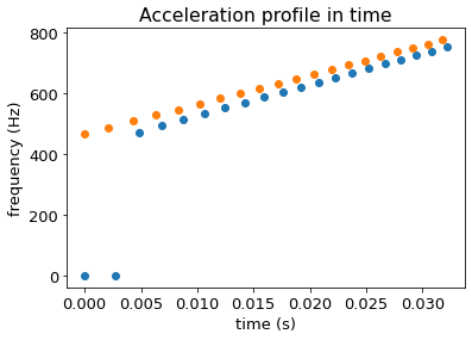
$$period\_seconds = \frac{1}{frequency} \tag{2}$$

$$period\_clock\_counts = \frac{period\_seconds}{clock\_frequency} \tag{3}$$

These formulas generate a constant acceleration as shown in Figure 44. On figure 44c, one can observe the starting frequency and the delay due to the preload registers.



(a) Speed profile during acceleration in time (b) Speed profile during acceleration in steps



(c) Zoom on the beginning of the acceleration profile

Figure 44: Speed profile during acceleration. The orange dots correspond to the command and the blue dots to the applied frequency

And the pulse train is shown in Figure 45. Those frames are extracted from Video 1.

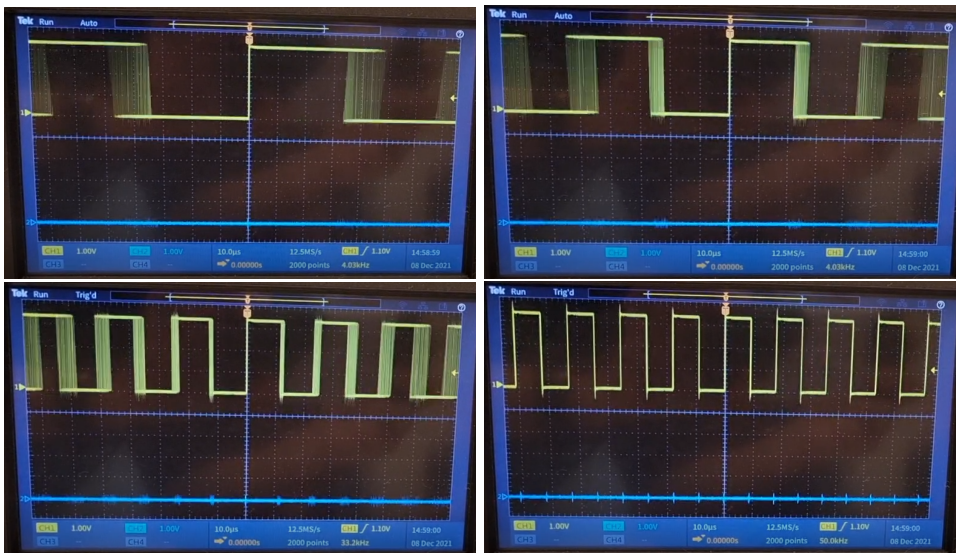


Figure 45: Pulse train generated during the acceleration

This implementation is however computationally heavy since it implies divisions of float numbers. Therefore, with the interrupt of the two motors running at the same time and at the same frequency, the maximal step frequency for each motor is 93 kHz. At higher frequencies, the two interrupts cannot be served in between steps and the steps will not be generated correctly.

Despite this drawback, it was chosen because it allows to change the acceleration easily during the design phase and for the fact that this maximal frequency is more than required for this robotic platform.

If it ever needed to be increased, other approaches could be used:
A common approach consists in simply having a lookup table containing the speed profile. This has the advantage of having an extremely low computational load, at the cost of a greater memory footprint. Moreover, if the desired acceleration changes, this lookup table needs to recomputed.
Other approaches allow real time computation of the speed profile such as the less computationally intense formula presented in [52] or the of use another timer responsible for the acceleration as described in [53] . This second timer takes the computation of the frequency away from the interrupt routine running the step generation and can run at smaller frequency.

Limit sensor
The limit sensor is used for the motors' homing. At the initialisation of the robotic platform, the linear stage motor will move toward the limit switch at minimal speed. This location will then serve as origin for the movement. The other end of the travel range is managed by counting the absolute number of steps and stopping before the allowed maximum number of steps. The motors' acceleration and maximal speeds are then properly tuned such that the motors do not miss steps. In fact, this would cause the controller to assume a wrong motor position and the stage could then hit the mechanical stops at the end of its travel range.

Note that the rotational stage does not have a limit sensor yet. It needs to be manually put to the left end of its travel range (when looking in the cameras direction) and it will assume it as the zero position. Note that moving the robotic system manually should only be done when the power stage of the stepper driver is deactivated. Otherwise the back electromagnetic field (EMF) of the motor may damage the drivers.

6.2.3 Example

In order to provide a general understanding of the processes running on the MCU, an example is presented.

In this example, the motors are first initialized. This is shown in Figure 46.

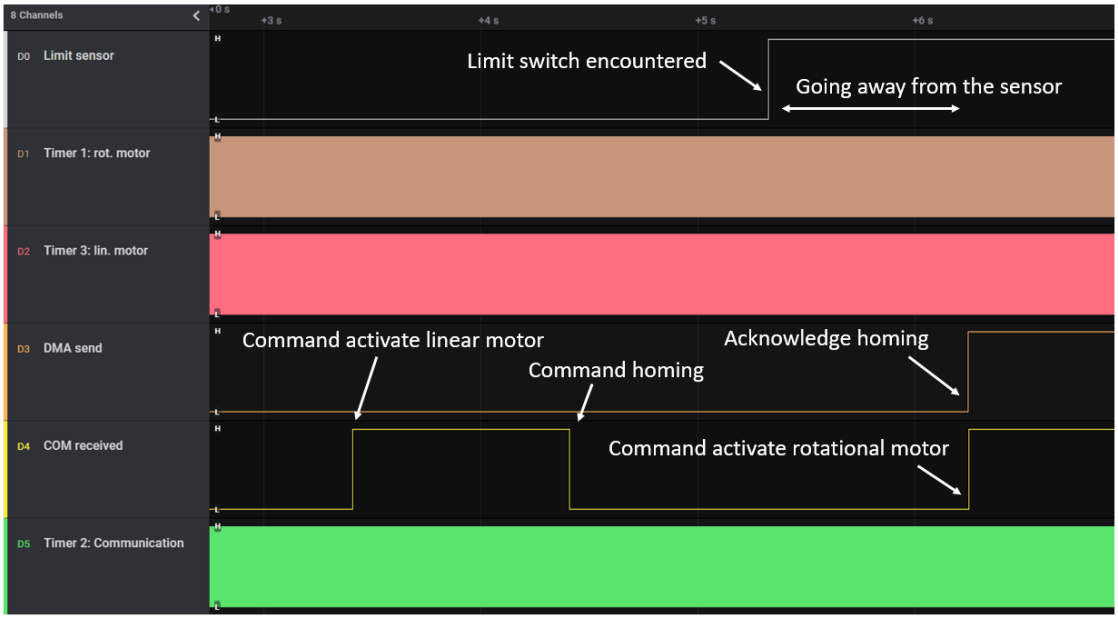


Figure 46: Processes running on the MCU during the initialisation of the motors

Zooming on the homing phase, one can observe that the timer 3 interrupt runs at a slow frequency defined for homing (Figure 47).

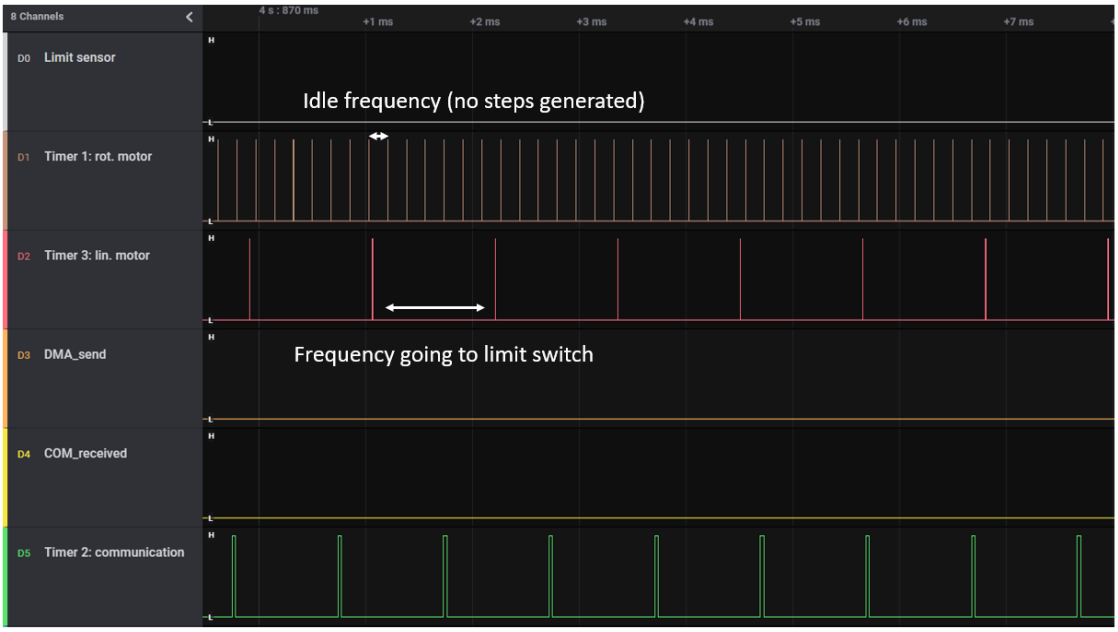


Figure 47: Processes running on the MCU during the homing of the linear stage

The following commands are then sent:

- Get linear position (waiting to get linear position);
- Move relative linear motor;
- Get rotational position (waiting to get rotational position);
- Move relative rotational motor;

And the MCU reacts as shown in Figure 48.

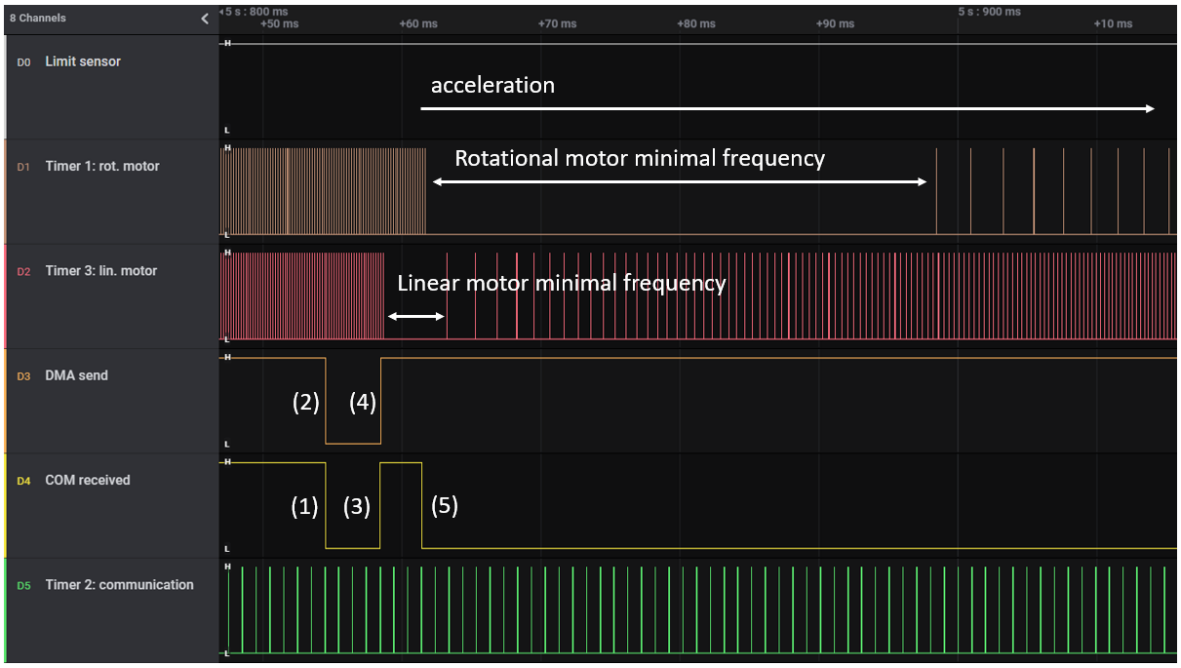


Figure 48: Processes running on the MCU when commands are sent. It should be noted that the channel 4 "COM received" does not indicate the DMA received but that at least the command for one of the motors has been found during the buffer search. (1) get linear position received. (2) DMA sends linear position. (3) get move linear and get rotational position. (4) DMA sends rotational position. (5) get move rotational command

By zooming on a time point at which both motors are running, one can observe that all interrupts can be served (Figure 49). This ensures the good functioning of the MCU program.

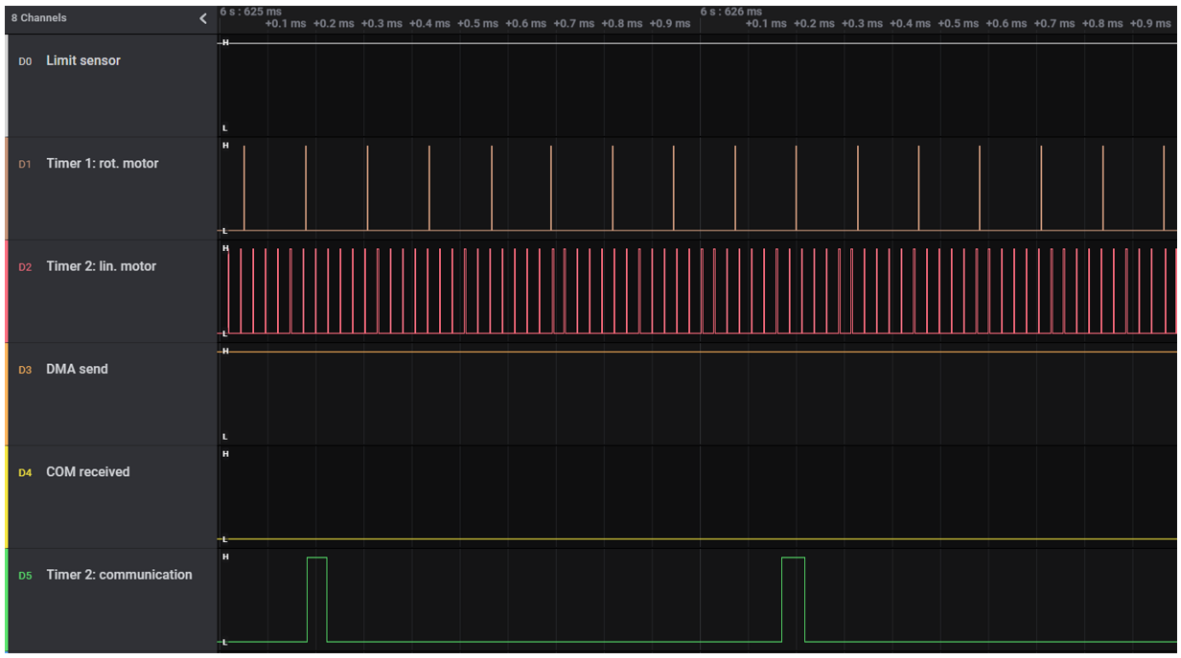


Figure 49: Processes running on the MCU when both motors are moving

7 High level controller

The high level controller implemented in Python on the PC consists in sending commands to the low level controller. For the feeding use case, it processes the images of the cameras to bring the end effector close to the fly's proboscis. For the trajectory following, it uses the forward kinematics model and the inverse of the Jacobian of the robotic platform to bring the end effector along a straight line to the next waypoint.

7.1 Computer vision

The computer vision algorithm used for closed loop control consists in training neural networks to detect objects in the cameras' frames using DeepLabCut (DLC) framework [24]. DLC is a versatile markerless pose estimation method based on transfer learning with deep neural networks that achieves excellent results with minimal training data. Even when only a small number of labeled frames ($\sim$ 200), the algorithm achieves excellent tracking performance on test frames that is comparable to human accuracy [54].

It uses the feature detector of DeeperCut [55] which are variations of deep residual neural networks (ResNet) [56] with readout layers that predict the location of an object. The network topology is presented in Figure 50.

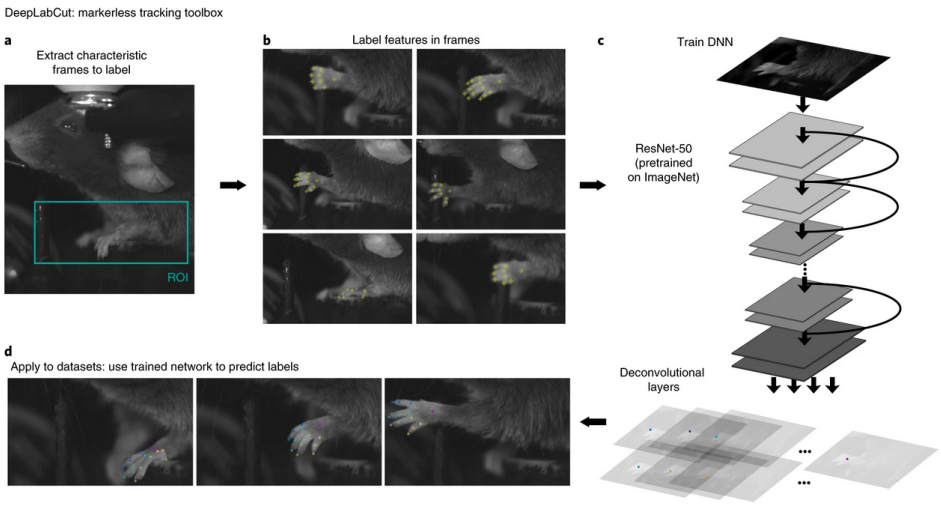


Figure 50: DeepLabCut overview

DLC is a deep convolutional network combining two key ingredients from algorithms for object recognition and semantic segmentation: pretrained ResNets on a popular, large-scale object recognition benchmark called ImageNet, on which it achieves excellent performance [56] and deconvolutional layers which are used to up-sample the visual information and produce spatial probability densities.

For each body part, its probability density represents the 'evidence' that an object is in a particular location. To fine-tune the network for a particular task, its weights are trained on labeled data, which consist of frames and the accompanying annotated body part locations (or other objects of interest in the frame) [54]. As a result of the initialization with the ResNet pretrained on ImageNet, this rewiring is robust and data-efficient.

In this work, two networks are used. One to identify the goal and another one to detect the end effector. Separating them allows to change one or the other independently in case the goal changes for one application but not the end effector or vice-versa. The detail of the training of the networks is presented in Section 9.1.4 in which the feeding use case is discussed.

The processing of the goal depends on the task at hand and will be discussed for the feeding use case which uses the closed loop control algorithm.

For the end effector, it is more general:
The corresponding network will predict the position of the end effector with a given probability. If this probability is higher than a threshold and is not in the first 10 pixels on the image boundary (this may happen in case of wrong predictions), it is taken as the position of the end effector. If it is not the case, the position is taken as the previous end effector position to which the last command is added. This enables to combine both the a priori knowledge of where the end effector should be and the measurement of the position given by the network.

Other estimation algorithm such as Kalman filter [57] or particle filter [58] could be used to combine these information in a more robust way.

7.2 Closed loop controller

The closed loop controller controls the Sensapex uMp3 in order to feed the fly by using the estimation of the position of the end effector and the goal in order to decide where to drive the end effector. At this stage of the project, sending a new command makes the Sensapex stop before serving it. Therefore,

the uMp3 is commanded to go to a constant offset in constant velocity, which produces the following velocity profile (if the constant offset is high enough to reach the commanded velocity):

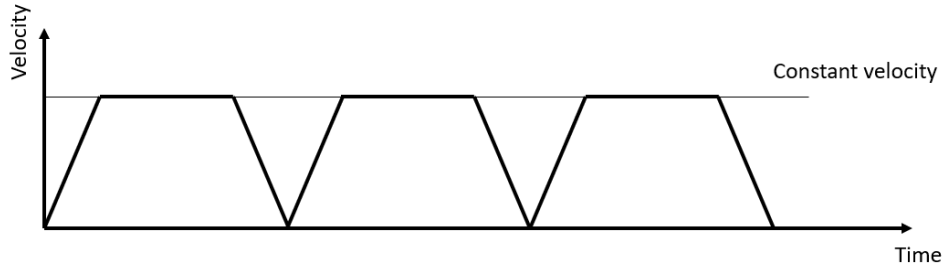


Figure 51: Sensapex velocity profile

Note that, later in the project, the functionality – called "take_step" – enabling to have smooth transitions between the commands has been implemented, based on recommendations from the Sensapex support team. "take_step" could be used in the closed loop algorithm in order to generate a smooth velocity profile.

In order to drive the end effector to the goal, the control algorithm has to use a model to transfer the information received from the networks in the pixel space to the 3D space of the robotic system.

There are multiple camera models which could be used to reconstruct the 3D position of a point from cameras' images. One of the most used model is the pinhole camera model [59] (Figure 52).

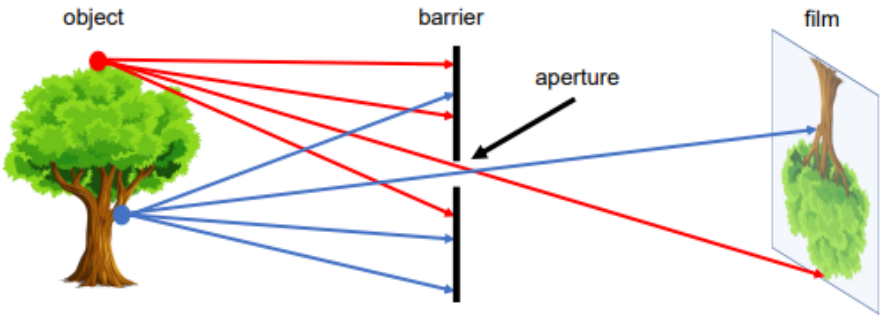


Figure 52: Pinhole camera model

These models require the cameras to be calibrated, which means that their intrinsic parameters (focal length and distortion parameters) and extrinsic parameters (position and orientation in the 3D space) have to be derived. A well known method to perform this calibration consists in showing a pattern under different angles and at different positions to the cameras and then run an optimisation algorithm [60].

However, in the present case, designing a small enough pattern would not be practical. Another alternative would be to use the uMp3 to draw those patterns, moving the end effector in space. This solution has not be retained since the cameras in the seven cameras setup are already calibrated, by means of a technique developed in the Neuroengineering laboratory. This method uses key points of the *Drosophila melanogaster* as a calibration pattern [61]. Therefore, developing a new technique would not be useful since the robotic platform is intended to be used in this setup.

Unfortunately, the method using the fly's keypoints would not lead to good results in the test setup. In fact, it needs the redundant information of more than two cameras to perform an accurate calibration.

Therefore, a simplified model is developed under the following assumption:
The cameras have no pinhole effect, the points are just projected onto the cameras array (the opposite orientation of the image is handled by the cameras' software which returns the image with the correct orientation). This assumption is valid as long as the considered point is close to the camera axis and stays at a similar distance from the camera (in this case the focus plane).

This model is illustrated in Figure 53.

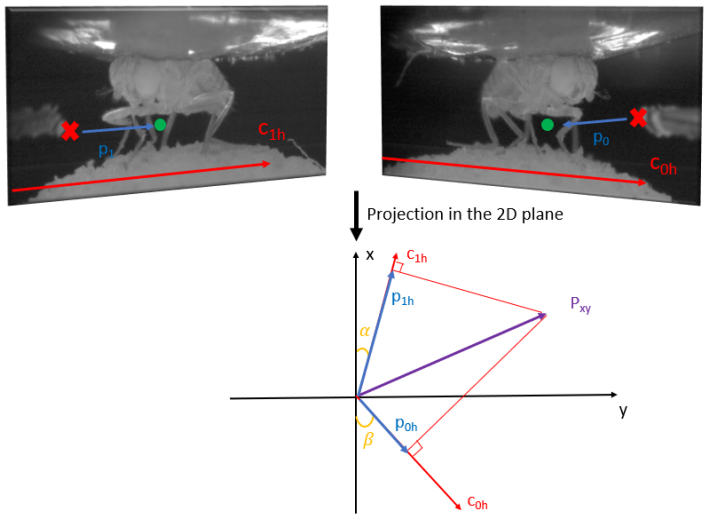


Figure 53: Closed loop control algorithm model. The projection in the horizontal plane does not correspond to those images for illustrative purposes

By using basic geometric considerations, the following formula is derived:

$$x = \frac{-p_{0h}sin(\alpha) + p_{1h}sin(\beta)}{sin(\alpha + \beta)} \tag{4}$$

$$y = \frac{p_{0h}cos(\alpha) + p_{1h}cos(\beta)}{sin(\alpha + \beta)} \tag{5}$$

For the vertical dimension, the average of the vertical component of p_0 and p_1 is taken. The $[x, y, z]$ vector is then normalized and the Sensapex uMp3 is commanded to go along the vector:
$[x_{normalized}, y_{normalized}, 2z_{normalized}]constant$
The commanded vector has a greater vertical amplitude such that the end effector reaches the correct height more rapidly. This reduces the risk of touching the ball or the fly.
The derivation of the formulas is presented in Appendix C.

The errors of the model will then be compensated for by the closed loop control algorithm. In fact, at each time step, the controller decides where to bring the end effector based on the position of the end effector and the position of the goal. At the next time step, it makes the decision again and is therefore able to correct the trajectory if it was heading slightly off the correct direction. If the decisions the algorithm makes are not too far off and the frequency of these decisions is sufficiently high with respect to movement's velocity, it is able to correct its trajectory until it reaches the goal.

As shown in the Section 9.1.4, the closed algorithm is able to perform well with this simple model and it will only benefit from a more accurate model using the seven calibrated cameras.

7.3 Open loop controller

The open loop controller controls the linear stage and the rotational stage in order to follow a predefined trajectory. For now, it commands the robotic platform to go at constant speed to a waypoint, stop close to this waypoint, wait for a given amount of time and then to go to the next one.

It does not control the Sensapex because smooth command transition were not implemented at that time. With the new "take_step" functionality mentioned above, the algorithm could be extended to command the Sensapex as well and distribute the commanded speed between the rotational stage, the linear stage and the micromanipulator. Another improvement would be to implement a velocity profile and smooth transitions between waypoints.

In order to follow a trajectory, the open loop controller uses the forward kinematics model and the inverse of the Jacobian of the robotic system to move towards the next waypoint along a straight line:
First the position of the motors is requested and the forward kinematics model is used to infer the position of the end effector in the horizontal plane.
If the distance between the end effector and the targeted waypoint is smaller than a threshold, the motors are stopped – they decelerate and then stop. The next waypoint is defined as the targeted waypoint.
Then, the inverse of the Jacobian is used to compute the velocity of the motors and this speed command is sent to the motors.

This trajectory is performed in open loop (without the feedback of the cameras) because the cameras are not calibrated and hence the simple model of the world would not be useful to accurately transpose coordinates from the 3D space to the pixel space. In fact, the model used by the closed loop algorithm predicts an inaccurate position in the 3D space from the pixels location, computes a command and corrects the trajectory at the next step. Therefore, using the simple model to predict an inaccurate position in the pixel space from a location in the 3D space and going to that pixel location would not ensure that the trajectory is followed accurately.

A possible workaround could be to perform the trajectory manually, record the positions in the pixel space and then use the closed loop controller to go to these pixels positions. This approach could be implemented in the future if needed.

For now, it is performed in open loop and, when the robot will be placed in the seven cameras setup, it will be able to benefit from the calibrated cameras and hence predict an accurate pixel position from a 3D space location and then go to that pixel location using the closed algorithm developed in this project.

7.3.1 Forward kinematics model

The geometric model is illustrated below:

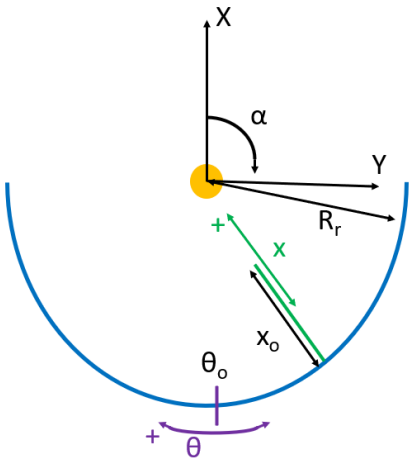


Figure 54: Geometric model

The following elements are indicated:

- X, Y, α: the coordinate system;
- R_r: the radius of the rail;
- x_o: the origin position of the linear stage (the middle of its travel range);
- x: the DOF controlled by the linear stage;
- θ_o: the origin position of the rotational stage;
- θ: the DOF controlled by the rotational stage.

The forward kinematics model is presented hereafter:

$$X = (R_r - x_o - x)cos(\theta + \theta_o) \tag{6}$$
$$Y = (R_r - x_o - x)sin(\theta + \theta_o) \tag{7}$$

7.3.2 Inverse of the Jacobian

The Jacobian is obtained by computing the derivative of the forward kinematics model. The Jacobian is then inverted using the SymPy Python package [62].

The inverse of the Jacobian is used to compute the speed needed in the robot joint space to match a velocity in the Cartesian space.

The inverse Jacobian is the following:

$$\dot{x} = -\frac{\dot{X}cos(\theta + \theta_o) + \dot{Y}sin(\theta + \theta_o)}{sin(\theta + \theta_o)^2 + cos(\theta + \theta_o)^2)} \tag{8}$$

$$\dot{\theta} = \frac{\dot{X}sin(\theta + \theta_o)}{(-R_r + x + x_o)sin(\theta + \theta_o)^2 + (-R_r + x + x_o)cos(\theta + \theta_o)^2} - \frac{\dot{Y}cos(\theta + \theta_o)}{(-R_r + x + x_o)sin(\theta + \theta_o)^2 + (-R_r + x + x_o)cos(\theta + \theta_o)^2} \tag{9}$$

8 Performances

The requirements defined in Section 3 are met. The performances of the robotic solution are presented hereafter.

8.1 Mechanics

Using the test setup and the 3D model of the seven cameras setup, the fact that the robot does not hit the cameras nor the ball mount is ensured.

The travel range of the end effector is large enough such that it can span a large region of the horizontal plane and exit the ring of LED in order to have a resting position outside the field of view of the cameras.

The workspace of the robot – without taking the V shape into account – when setting the Zaber linear stage and the Sensapex uMp3 at the middle of their travel range and placing the end effector 12.5mm away from the center of the ball (such that it has just enough travel range to exit the LED ring) is shown in Figure 55. For the setup in which the V shape is present, the angle range of the rotational is limited to about [45deg, -60deg] because of the electronics mounted on the side of the bracket.

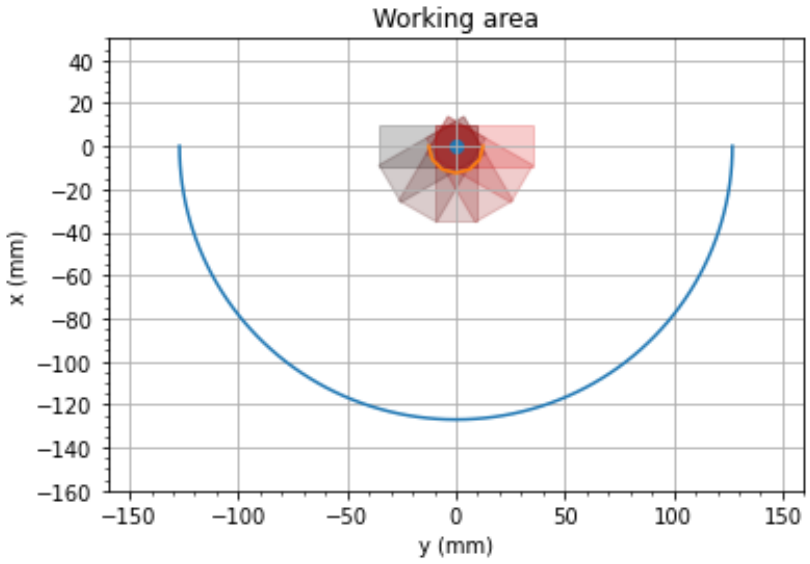


Figure 55: Working area. The dot at the origin represents the ball. The orange and the blue semi-circles correspond to the 12.5mm radius and the rail respectively

8.2 Low level controller

The low level controller allows variable lengths commands arriving at unknown timings as well as command changes on the fly. The performance of the MCU program when it is compiled with the O-fast optimisation is discussed in the next Subsections.

8.2.1 Communication

The command decoding for the stepper motors occurs at 1 kHz. From sending the command 'G' (get position) from the PC application to receiving it, it takes 2.95ms on average. For the Sensapex, it takes 0.1ms. The time for sending the movements commands is negligible.

Commanding the three motors and requesting their position at each time step allows to command the system every 6ms. Hence the high level controller frequency can be up to 163 Hz. If needed, this frequency could be greatly increased by doing the following combination of changes:

- One could double the frequency of the timer 2 interrupt. This would reduce the time of getting the position by 0.5 ms on average and allow a high level controller frequency up to 196 Hz;
- One could combine the previous change with having one command for getting both motor positions, this would allow a frequency up to 384 Hz;
- Finally, one could implement a less computationally intensive speed calculation and greatly increase the timer 2 interrupt frequency. The main limitation would then become the communication itself and the receiving of the command in the Python code.

8.2.2 Step signals generation

With the interrupt of the two motors running at the same time and at the same frequency, the maximal step frequency for each motor is 93 kHz. At higher frequencies, the two interrupts cannot be served in between steps and the steps will not be generated correctly. This is more than required by the two motors. In fact, the maximal frequency needed by the motors to reach the maximal speed indicated in Section 8.3.3 is 55kHz and 11.6 kHz for the Zaber and rotational stage respectively.

8.2.3 Delays

The frequency of the timer is changed at each counter underflow. Therefore, a newly received command can only be actualized during these update-events (UEV). The delay for actualising these commands is maximal when the motors run at their lowest frequency.

When the linear motor runs at its lowest frequency, the timer period is 4.6 ms and the maximal delay is 2.3ms on average.

Concerning the rotational stage, its maximal timer period is 37ms and hence a maximal delay of 18.5ms on average.

With a properly tuned high level controller, those delays can be compensated for by giving more responsibility to the Sensapex for the fast changes at slow speeds.

If these delays need to be reduced, the following modifications could be made:

- Using stepper driver capable of higher microstepping, at the cost of reducing the available torque, and increasing the frequency of the timers;
- Implementing a mechanical reduction between the rotational motor and the pinion and increasing the frequency of its timer.

8.3 Dynamics

The dynamical requirements are also met. They are presented hereafter.

8.3.1 Resolution

The resolution of the system is defined by the most precise element, hence the Sensapex uMp3: 5nm.

For reference, the resolution of the rotational stage is calculated as follows:
The gear rack has 336 teeth over its whole circumference, hence 1.016 degree per tooth. The pinion has 48 teeth which corresponds to 51 degrees. The motors has 200 steps and microstepping of 1/32 of a step is used which makes 6400 steps per revolution. Therefore, 0.008 degrees per steps.

The resolution of the Zaber linear is 0.76 µm. In fact, one revolution corresponds to 2.44mm and, with 1/16 microstepping, to 3200 steps. Therefore, one step is equivalent to 0.76 µm.

8.3.2 Acceleration

The acceleration of the linear stage and the rotational stage are 75mm/s^2 and 2.619rad/s^2 respectively. For the rotational stage this corresponds to a tangential acceleration of 351mm/s^2.

8.3.3 Speed

The speed of the different elements composing the robotic system is presented in the table below:

Element	Minimal radial speed	Maximal radial speed	Minimal tangential speed	Maximal tangential speed	Minimal vertical speed	Maximal vertical speed
Sensapex	100µm/s	5mm/s	100µm/s	5mm/s	100µm/s	5mm/s
Linear stage	160µm/s	42mm/s	0mm/s	0mm/s	0/mm/s	0mm/s
Rotational stage	0mm/s	0mm/s	0.49mm/s	217mm/s	0/mm/s	0mm/s
Robotic system	100µm/s	47mm/s	100µm/s	222mm/s	100µm/s	5mm/s

Table 3: Minimal and maximal velocity along the different axes

Those speeds are more than required to move objects in the fly's referential. Note that the maximal speed is the maximal instantaneous speed and that lower minimal speeds can be reached by driving the Sensapex and the linear/rotational stage in opposite direction.

For reference, the linear stage and the rotational stage can move at 21.35mm/s over 70% of its travel range and 65mm/s over 90% of its travel range respectively.

8.4 High level controller

Concerning the high level controller, the use cases presented in the next Section show that it can achieve these tasks successfully.

8.4.1 Closed loop controller

Currently, the closed loop controller used for feeding can run at 2Hz when the two network inferences are done on a quite modest GPU (Nvidia Quadro P200). This is sufficient to perform the task at a reasonable speed as shown in the next Section.

The frequency of the controller could be greatly increased by using a more powerful GPU and/or cropping the frames and retraining a network on those smaller frames. Benchmarks for DLC inference speed on different hardware is presented in [63].

8.4.2 Open loop controller

The open loop controller, which uses the forward kinematics model and the inverse of the Jacobian, can run at a frequency of 50 Hz. This frequency allows to follow trajectories as it will be shown in the corresponding use case.

9 Applications

As stated before, two use cases are presented. The first one shows the working principle of the automatic feeding of the fly and the second one shows how to generate open loop trajectories.

9.1 Feeding

Two solutions are presented for the automatic feeding. The first one consists in the user defining a goal, an origin and the time frequency at which the robot has to go to goal and how long it should stay before returning to the origin.

The other approach consists in using the closed loop algorithm discussed above to move the end effector close to the fly's proboscis.

9.1.1 Previous feeding method

The feeding method used until now in the Neuroengineering laboratory consisted in removing the ball and approaching a needle to the fly using the Sensapex uMp3 commanded manually with the uMp-RW3 rotary wheel unit (Figure 56).

Figure 56: uMp-RW3 rotary wheel unit

Some images illustrating this feeding method are shown in Figure 57. The corresponding video is Video 2.

Figure 57: Previous feeding method

9.1.2 End effector design

For the feeding application using the developed robotic solution, the objective is to let the spherical treadmill in place. This enables not disturbing the fly's behavior that much when it needs to be fed. Moreover, the neural activity happening during feeding could be measured in more realistic conditions.

For this reason, a new end effector providing liquid food to the fly has to be designed. In fact, the one shown in Figure 57 would present too much risks in the liquid touching and damaging the spherical treadmill.

A hypodermic needle seemed like an appropriate solution since it has a support at the bottom, which limits the risk of the droplet touching the ball. Then, taking inspiration from the state of the art feeding methods, which use some cotton to contain the liquid, a first test was made with a 1mm diameter needle with a kimwipe – which has a material with a good liquid absorbance – made into a small thread inserted in the needle and showing at the tip. The result is shown in Figure 58.

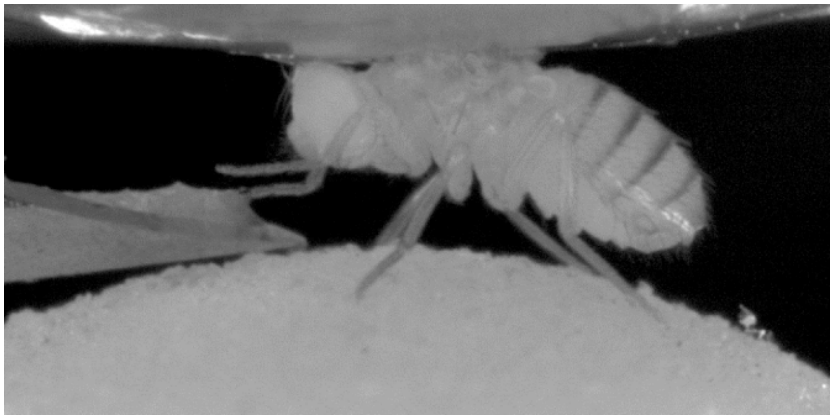


Figure 58: 1mm hypodermic needle

This gives an idea of the the dimensions of the fruit fly and one can conclude that this needle is definitely too big. A second test was performed with a 0.3mm diameter needle (Figure 59).

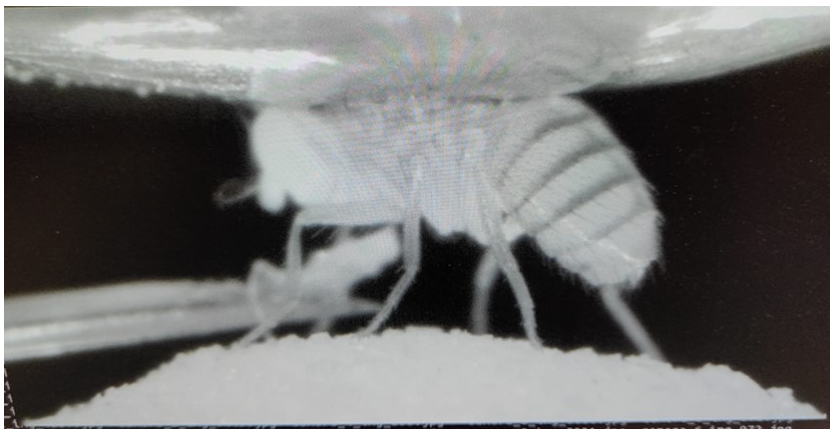


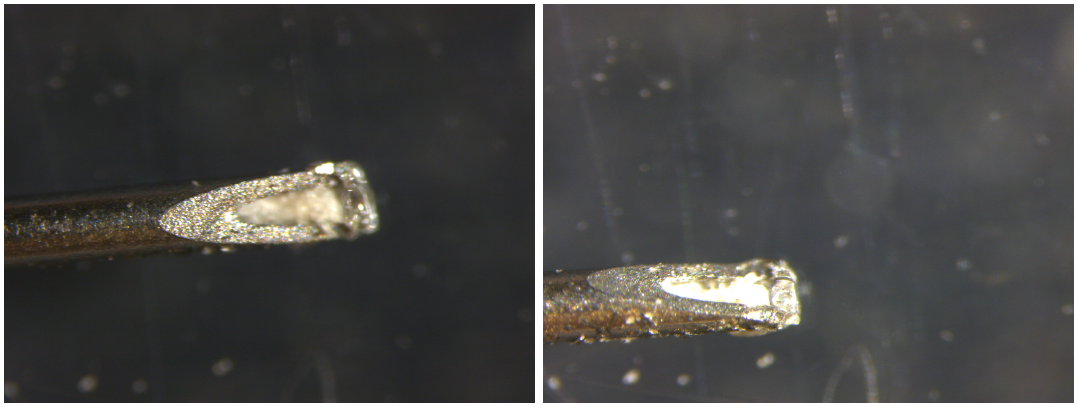
Figure 59: 0.3mm hypodermic needle

The dimensions are more reasonable. However, the contact surface between the thread and the needle being much smaller, the adhesion is too small and the clean wipe thread tends to slip on the side, which puts the ball at risk of being damaged by the liquid solution.

Therefore, the choice of using a side hole needle (Figure 60) is made. This will indeed prevent the thread from slipping to the side of the needle. Unfortunately, no side hole needle having the hole close enough to the end of the needle was found on the market. This is of particular importance because the tip of the needle should not touch the fly's body and potentially harm it when the hole is positioned under the proboscis.

Figure 60: Side hole needle

A procedure to create a side hole needle from a standard needle is then designed. It consists in grinding the tip of the needle, closing the tip surface with plastic and inserting a thread in the hole. The procedure is presented in Appendix D. The end result is presented in figure 61.



(a) Top view of the side hole needle (b) Side view of the side hole needle

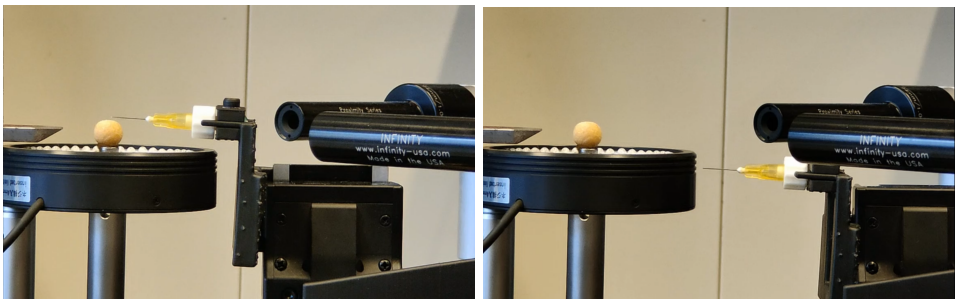
Figure 61: Developed side hole needle

The needle is then connected to a syringe by means of a tube. Currently, the syringe is actuated manually.

9.1.3 User defined goal position

For this first approach, the user should bring the end effector to the goal position (position for feeding the fly), then bring it to an origin position (outside the field of view of the cameras) and finally define time intervals at which the end effector will go the goal and how long it will stay.

This approach is shown in Video 3 and the list of commands to define the goal and the origin, as well as general information on how to use the platform, are described in Appendix E. The goal and the origin defined for this video are shown in Figure 62. As it can be observed on the video, the end effector goes away from the goal at a high speed. This speed can easily be adapted if needed.



(a) User defined goal (b) User defined origin

Figure 62: User defined goal and origin

9.1.4 Automatic movement

The second approach consists in approaching the fly automatically using the closed loop algorithm. The training of the neural networks is described hereafter.

"Front fly" neural network
The front fly neural network has been trained in the seven cameras setup (the test setup was not built at that point). 390 frames from the two cameras placed at 45 degrees and five different flies have been labelled. These cameras are the one used in the test setup (circled in green on Figure 63).

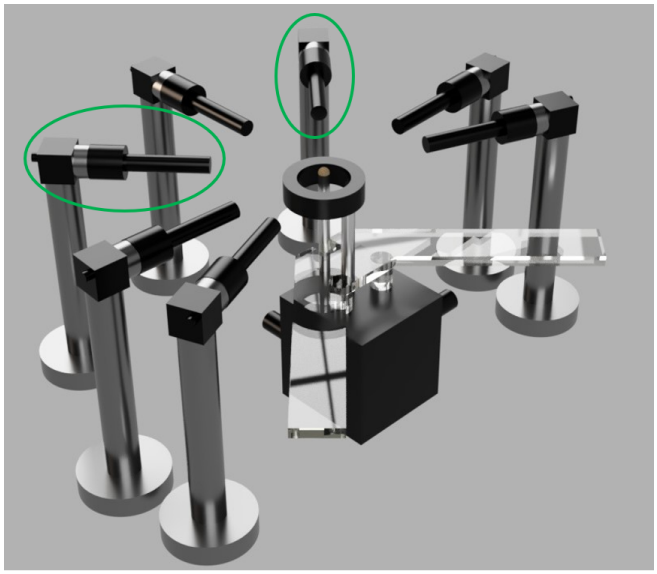


Figure 63: Cameras used to train the "front fly" neural network

The objective is for the network to detect the following points (Figure 64):

- the top of the eye (in violet);
- the bottom of the eye (in blue);
- the front of the head (in green)
- the origin of the proboscis (in orange);
- the point between the thorax coxa joints (in light blue);
- the proboscis (in red).

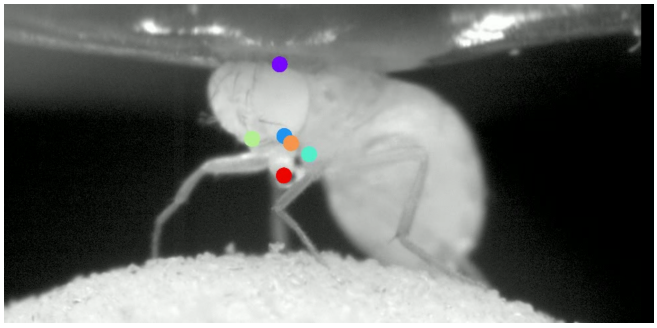


Figure 64: Points to detect for the front fly neural network

Performance
The network has been trained for 60000 iterations with a training/test split ratio of 0.95. With a probability threshold set to 0.6, it achieves a mean Euclidean error (MAE) of 2.77 pixels on the training dataset and performs similarly on the test set with an error of 2.57 pixels .
For reference, one pixel represents 4.8µm.

"Needle" neural network
The second network, used to detect the needle, has been trained on 1100 frames coming from the two same cameras with 13 different end effectors and a different *Drosophila* for each end effector. The keypoints are the tip of the needle and the angle at the top of the needle as shown on Figure 65.

Figure 65: Points to detect for the needle neural network

As stated before, 13 different needles have been used to ensure generalization and robustness of the network. Some of these needles are presented on Figure 66. One can observe different needle tip lengths and different amounts of plastic at the tip of the needle as well as frames with liquid and without.

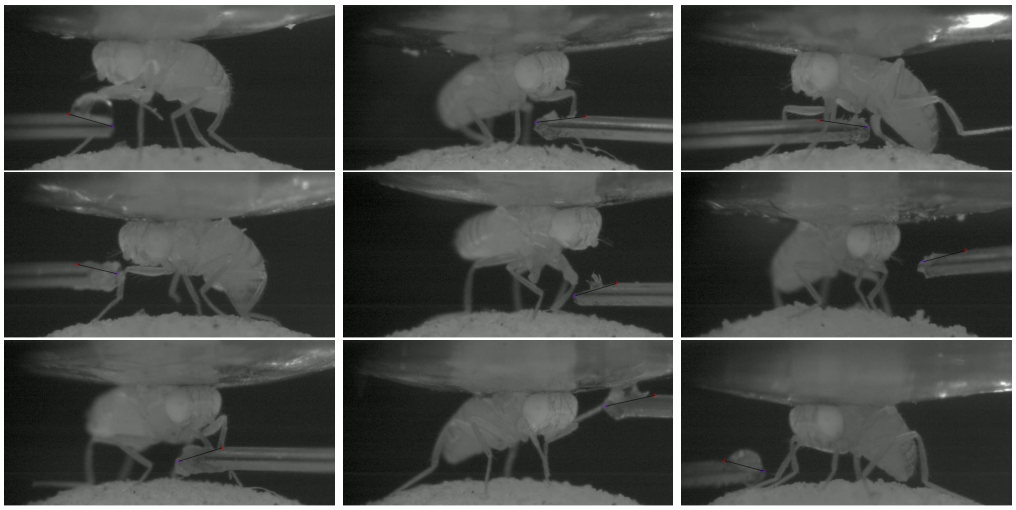


Figure 66: Different end effectors to train the "needle" neural network

Performance
This task is harder to learn because of the diversity of the needles which were presented and the fact that the depth of field of the cameras is small such that needle moving in space are often blurred. Therefore, the network has been trained on more frames than the previous for significantly more iterations: 300'000. With a train/test split ratio of 0.95 and a probability threshold set to 0.6, it achieves a MAE training error of 2.45 pixels and a test error of 7.98 pixels when the probability threshold is set to 0.6.

Training the network for more iterations showed overfitting since the training error was getting smaller and the test error increased.

Feeding
In order to demonstrate the working principle of the automatic feeding solution, four wild type *Drosophila melanogaster*, aged between 2 to 5 days, have been starved for 24 hours. The liquid food provided by the end effector consists in 1ml of deionized water with 15mg of sucrose. The algorithm brings the needle tip close to the fly's proboscis by using the bottom of the eye and the tip of the needle networks' inferences, as those were the most stable points. The goal is defined as an offset to the bottom of the eye – 70 pixels below and 60 pixels towards the fly's abdomen.

For each fly, the algorithm is able to reach the goal starting from different initial positions. This is demonstrated, with a total of 25 different initial positions, in Videos 4-7. For those videos, a third camera has been placed in order to provide a side view of the scene. It is currently not used by the algorithm.

On Figure 67, one can observe some of the initial positions from which the closed controller can successfully reach the goal.

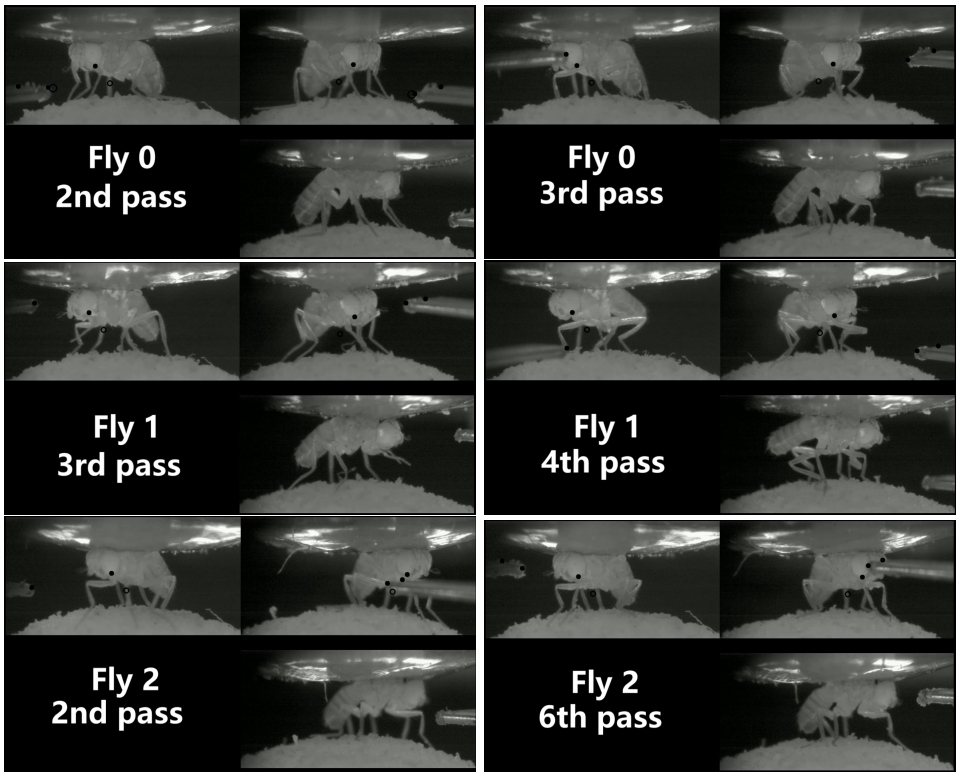
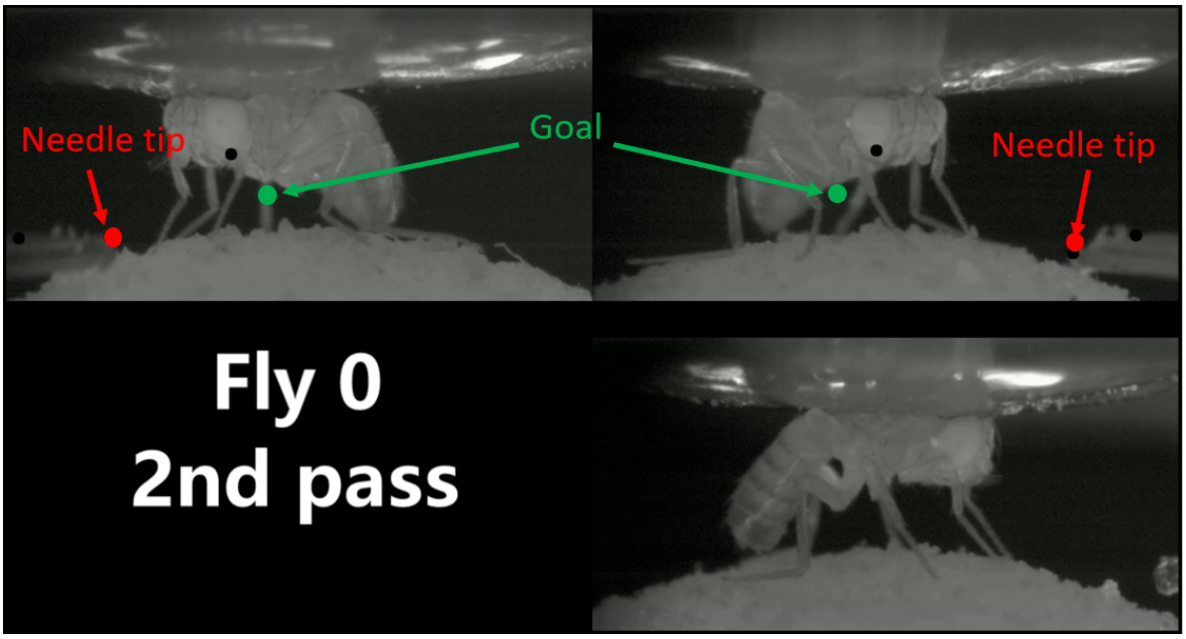
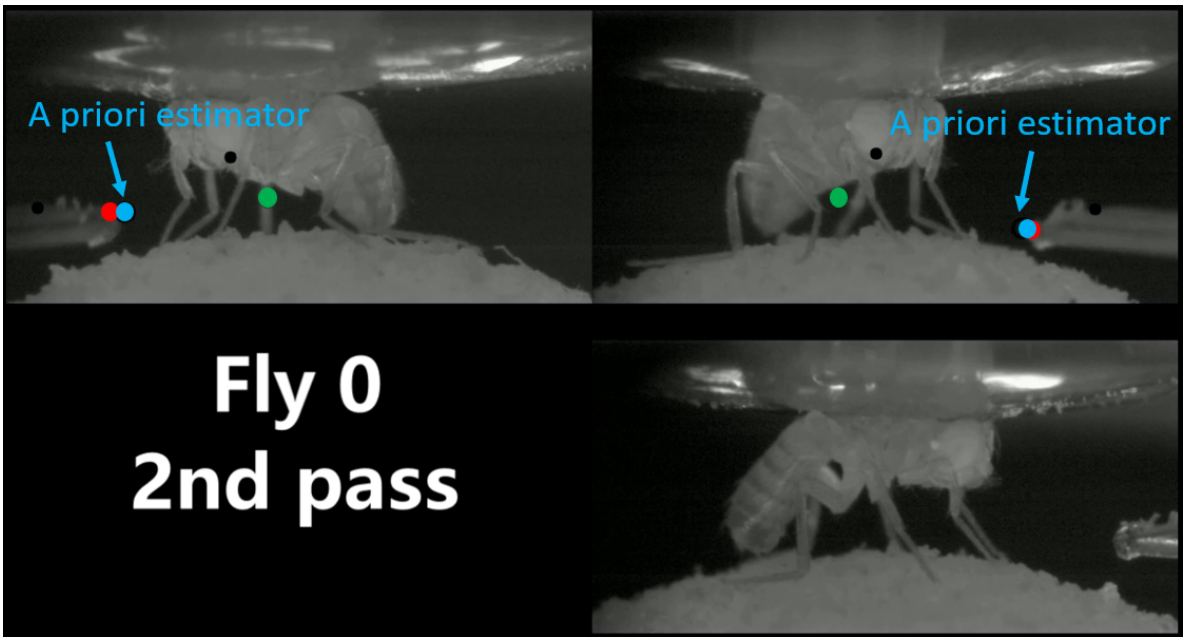


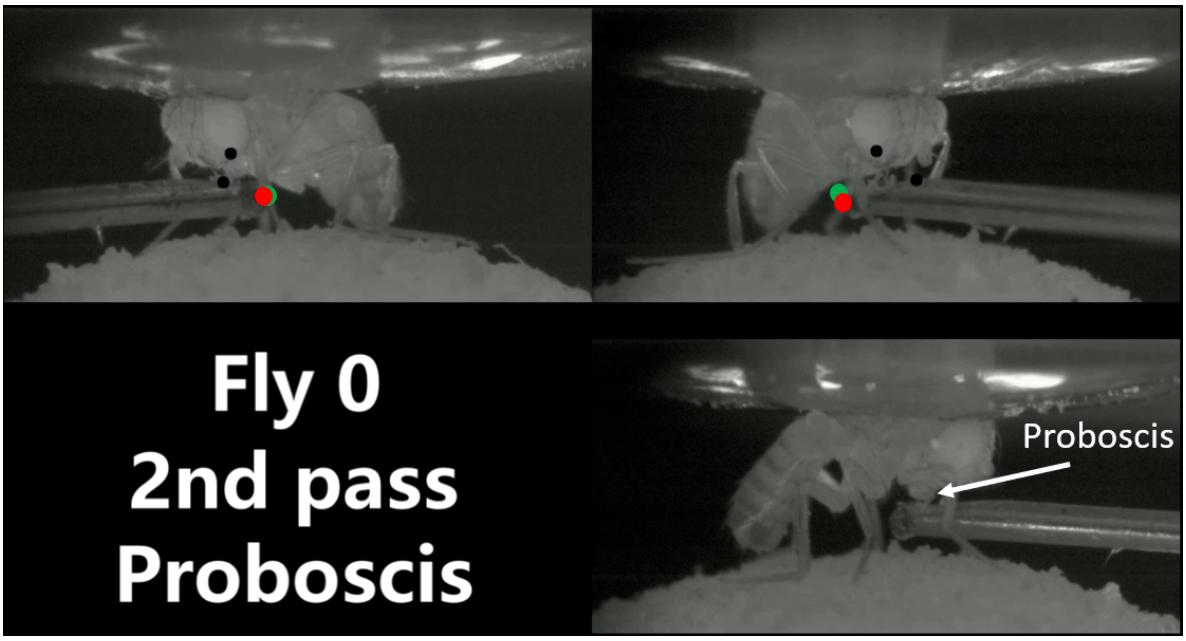
Figure 67: Different initial positions from which the closed loop controller can successfully accomplish its task

One can also observe that for each trial, the flies are positioned on the spherical treadmill in a slightly different way. This enforces the demonstration that the algorithm is capable of reaching the goal in different contexts.

Some frames from the video of fly 0 (Video 4), during the second pass, are shown below:







First, the initial position is shown. Then, the algorithms starts and the a priori estimator of the needle position appears. The end effector approaches to the goal and stops there (the a priori estimator is not indicated anymore). Finally, the fly touches the food with it proboscis.

In Video 4, during the pass 0, the algorithm is started at an initial position which is too close to the ball and touches it (Figure 68). In order to avoid these situations, a repulsive virtual boundary on the ball surface could be added to the algorithm or more conservative initial positions should be enforced.

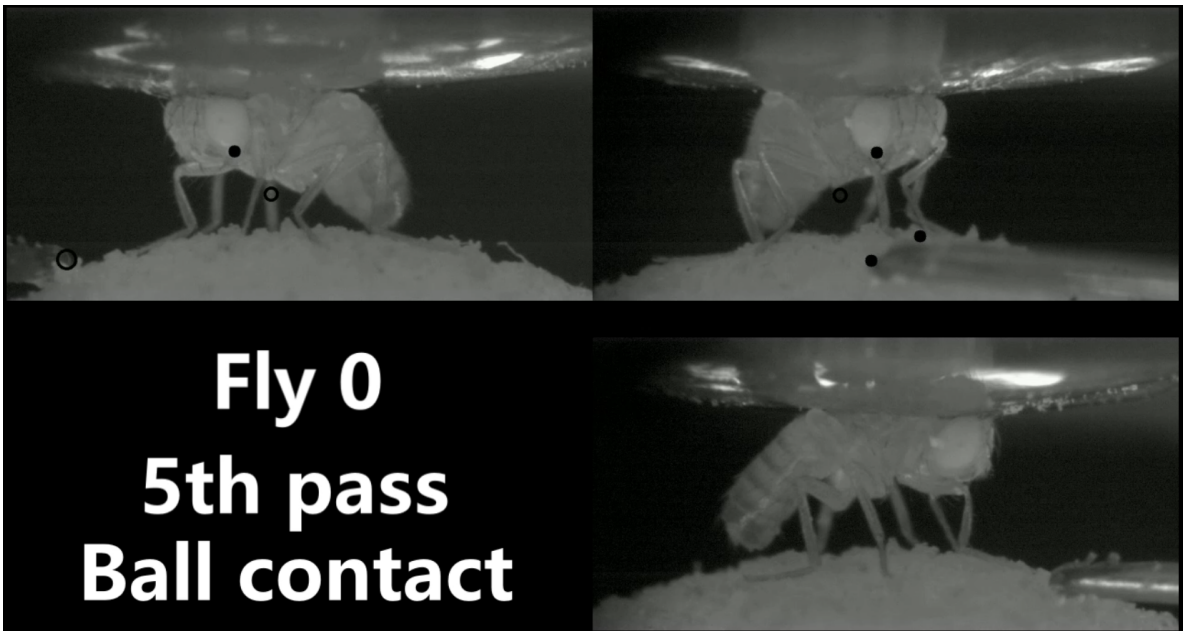


Figure 68: Contact with the ball

Note that the liquid is currently brought by manually actuating a syringe. A device to automatically bring the liquid to the end effector tip would allow to make the task fully automatic.

9.2 Trajectory

The robotic solution following a trajectory consisting in a square at 4mm/s is shown in Video 8. First the robotic system goes to its origin position. Then it moves to the first waypoint and, from there, moves from waypoint to waypoints. It does so by controlling both the rotational stage and the linear stage in order to move along a straight line towards the next waypoint. As one can observe on the video, it is able to the motors simultaneously and reach the waypoints.

However, because the trajectory is performed in open loop and the model parameters have not been very precisely measured, the trajectory does not exactly represents a square. Moreover, since the algorithm commands the robotic platform in constant speed and simply stops when close to one waypoint, positions are slightly overshot. This contributes to the trajectory not being a perfect square.

Having a smoother velocity profile and measuring the parameters more precisely would allow to follow trajectories more accurately and at much greater speeds.

The four waypoints as reached by the controller are shown in Figure 69.

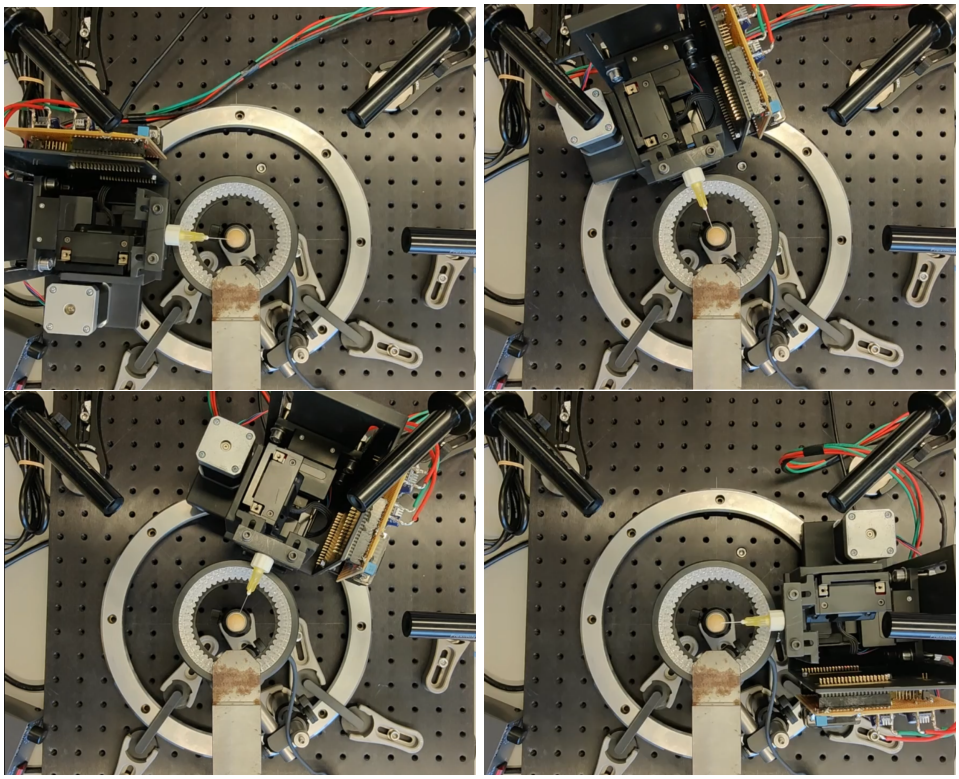


Figure 69: Waypoints reached by the open loop algorithm

10 Conclusion

In this work, a robotic system enabling new experiments for tethered *Drosophila melanogaster* has been developed. These new experiments include longer monitoring of the flies thanks to automatic feeding, which enables the study of diseases or aging and stimulating the flies by moving objects in their environment.

In order to do so, the following elements have been developed:

- A wrapper around the Sensapex Python API [21] for the Sensapex uMp3 micromanipulator [22]; [23];
  - Enabling a new functionality: smooth transitions between commands.
- A stepper motor controller on a STM32F4 microcontroller [23];
  - Communicating with the PC application;
  - Allowing changing commands on the fly;
  - Controlling multiple motors;
  - Taking limit sensor as input.
- The mechanical design of a versatile robotic platform enabling new experiments for tethered *Drosophila melanogaster*;
  - Comparing and selecting existing solutions on the market in terms of costs and performances;
  - Designing mechanical parts to assemble them.
- A closed loop control algorithm;
  - Receiving cameras' frame as input;
  - Processing those frames for object recognition using DeepLabCut Python package [24].
- An open loop control algorithm for trajectory following.

The working principle of two use cases was presented:

- Automatic feeding of the fly;
- Trajectory following using the forward kinematics model and the inverse of the Jacobian of the robotic platform.

The robotic solution could successfully accomplish the automatic feeding using the closed loop algorithm and the trajectory following in open loop.

The robot performances also allow moving objects in the horizontal plane at speeds corresponding to the fly moving on the spherical treadmill. This will enable moving objects in the fly's referential by using the movement of the spherical treadmill.

The robotic system presents two main limitations:
The maximal delay to command the rotational motor – occurring when runs at its minimal speed – is 18.5ms on average. This may be too important for certain applications and some ways to reduce it have been presented in Section 8.2.3.
With only two uncalibrated cameras, the robotic system cannot perform closed loop trajectories. This would be needed in order to accurately follow trajectories, in particular when close to the fly.

11 Further work

For the experiments using feedback from the cameras to move the needle containing food, the network used to detect the needle position could be made more robust by training it with a luminosity which allows to better detect the needle tip. Moreover, making more similar end effectors during the training would improve its performance at the cost of generalization.

Concerning the feeding experiments, using the seven cameras would allow to monitor the quantity of ingested food by measuring the amount of time the fly's proboscis is in contact with the liquid solution for example. Since the *Drosophila* needs to be made aware that the liquid solution contains food. The algorithm could be adapted such that a contact is made with the fly's proboscis. Moreover, in order to have a completely automatic feeding, a device allowing the bring the liquid to the tip of the needle should be installed.

Finally, for the experiments consisting in moving objects in the fly's referential, an additional degree of freedom could be added for rotating the end effector in order to keep the orientation of the objects when moved in the fly's referential. The implementation of the algorithm taking into account the movements of the ball would also be needed. Moreover, since the cameras are calibrated in the seven cameras setup, the generation of closed loop trajectories can be performed by combining the developed trajectory controller and adapting the closed loop algorithm such that it takes into account the parameters of the calibrated cameras.

References

[1] Barbara H. Jennings. "Drosophila – a versatile model in biology & medicine". In: *Materials Today* 14.5 (May 2011), pp. 190–195. ISSN: 1369-7021. DOI: 10.1016/S1369-7021(11)70113-4.

[2] Berrak Ugur, Kuchuan Chen, and Hugo J. Bellen. "Drosophila tools and assays for the study of human diseases". In: *Disease models & mechanisms* 9.3 (Mar. 2016), pp. 235–244. ISSN: 1754-8411. DOI: 10.1242/DMM.023762. URL: https://pubmed.ncbi.nlm.nih.gov/26935102/.

[3] Kamal N. Bharucha. "The Epicurean Fly: Using Drosophila Melanogaster to Study Metabolism". In: *Pediatric Research 2009 65:2* 65.2 (Feb. 2009), pp. 132–137. ISSN: 1530-0447. DOI: 10.1203/pdr.0b013e318191fc68. URL: https://www.nature.com/articles/pr200927.

[4] Wanli W. Smith et al. "From fat fruitfly to human obesity". In: *Physiology & behavior* 0 (Sept. 2014), p. 15. ISSN: 1873507X. DOI: 10.1016/J.PHYSBEH.2014.01.017. URL: /pmc/articles/PMC4125553/%20/pmc/articles/PMC4125553/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4125553/.

[5] Akhila Rajan and Norbert Perrimon. "Of flies and men: Insights on organismal metabolism from fruit flies". In: *BMC Biology* 11.1 (Apr. 2013), pp. 1–6. ISSN: 17417007. DOI: 10.1186/1741-7007-11-38/FIGURES/1. URL: https://bmcbiol.biomedcentral.com/articles/10.1186/1741-7007-11-38.

[6] Karel Svoboda and Ryohei Yasuda. "Principles of Two-Photon Excitation Microscopy and Its Applications to Neuroscience". In: *Neuron* 50.6 (June 2006), pp. 823–839. ISSN: 0896-6273. DOI: 10.1016/J.NEURON.2006.05.019. URL: http://www.cell.com/article/S0896627306004119/fulltext%20http://www.cell.com/article/S0896627306004119/abstract%20https://www.cell.com/neuron/abstract/S0896-6273(06)00411-9.

[7] Laura Hermans et al. "Long-term imaging of the ventral nerve cord in behaving adult Drosophila". In: *bioRxiv* (Oct. 2021), p. 2021.10.15.463778. DOI: 10.1101/2021.10.15.463778. URL: https://www.biorxiv.org/content/10.1101/2021.10.15.463778v1%20https://www.biorxiv.org/content/10.1101/2021.10.15.463778v1.abstract.

[8] Chin Lin Chen et al. "Imaging neural activity in the ventral nerve cord of behaving adult Drosophila". In: *Nature Communications 2018 9:1* 9.1 (Oct. 2018), pp. 1–10. ISSN: 2041-1723. DOI: 10.1038/s41467-018-06857-z. URL: https://www.nature.com/articles/s41467-018-06857-z.

[9] Andres Flores Valle, Rolf Honnef, and Johannes D. Seelig. "Automated long-term two-photon imaging in head-fixed walking Drosophila". In: *Journal of Neuroscience Methods* (Nov. 2021), p. 109432. ISSN: 0165-0270. DOI: 10.1016/J.JNEUMETH.2021.109432.

[10] Leonie Kirszenblat and Bruno van Swinderen. "Sleep in Drosophila". In: *Handbook of Behavioral Neuroscience* 30 (Jan. 2019), pp. 333–347. ISSN: 15697339. DOI: 10.1016/B978-0-12-813743-7.00022-0. URL: /record/2019-65209-021.

[11] Xitong Liang et al. "Morning and Evening Circadian Pacemakers Independently Drive Premotor Centers via a Specific Dopamine Relay". In: *Neuron* 102.4 (May 2019), pp. 843–857. ISSN: 0896-6273. DOI: 10.1016/J.NEURON.2019.03.028.

[12] Xitong Liang, Timothy E. Holy, and Paul H. Taghert. "Synchronous Drosophila circadian pacemakers display nonsynchronous Ca^2 rhythms in vivo". In: *Science (New York, N.Y.)* 351.6276 (Feb. 2016), pp. 976–981. ISSN: 1095-9203. DOI: 10.1126/SCIENCE.AAD3997. URL: https://pubmed.ncbi.nlm.nih.gov/26917772/.

[13] Chun Chao Chen et al. "Visualizing long-term memory formation in two neurons of the Drosophila brain". In: *Science (New York, N.Y.)* 335.6069 (Feb. 2012), pp. 678–685. ISSN: 1095-9203. DOI: 10.1126/SCIENCE.1212735. URL: https://pubmed.ncbi.nlm.nih.gov/22323813/.

[14] Ugur Dag et al. "Neuronal reactivation during post- learning sleep consolidates long-term memory in drosophila". In: *eLife* 8 (Feb. 2019). ISSN: 2050084X. DOI: 10.7554/ELIFE.42786.

[15] Cheng Huang et al. "Long-term optical brain imaging in live adult fruit flies". In: *Nature Communications 2018 9:1* 9.1 (Feb. 2018), pp. 1–10. ISSN: 2041-1723. DOI: 10.1038/s41467-018-02873-1. URL: https://www.nature.com/articles/s41467-018-02873-1.

[16] Judy Lisette Martin et al. "Long-term live imaging of the drosophila adult midgut reveals real-time dynamics of division, differentiation and loss". In: *eLife* 7 (Nov. 2018). ISSN: 2050084X. DOI: 10.7554/ELIFE.36248.

[17] Tom Hindmarsh Sten et al. "Sexual arousal gates visual processing during Drosophila courtship". In: *Nature* 595.7868 (July 2021), pp. 549–553. ISSN: 14764687. DOI: 10.1038/S41586-021-03714-W.

[18] Aryeh Zolin et al. "Context-dependent representations of movement in Drosophila dopaminergic reinforcement pathways". In: *Nature Neuroscience 2021 24:11* 24.11 (Oct. 2021), pp. 1555–1566. ISSN: 1546-1726. DOI: 10.1038/s41593-021-00929-y. URL: https://www.nature.com/articles/s41593-021-00929-y.

[19] Richard A. Warren et al. "A rapid whisker-based decision underlying skilled locomotion in mice". In: *eLife* 10 (2021), pp. 1–27. ISSN: 2050084X. DOI: 10.7554/ELIFE.63596.

[20] Richard J.D. Moore et al. "FicTrac: A visual method for tracking spherical motion and generating fictive animal paths". In: *Journal of Neuroscience Methods* 225 (Mar. 2014), pp. 106–119. ISSN: 0165-0270. DOI: 10.1016/J.JNEUMETH.2014.01.010.

[21] *sensapex/sensapex-py: Python wrapper for the Sensapex SDK*. URL: https://github.com/sensapex/sensapex-py.

[22] *Single micromanipulator system for patch-clamp – Sensapex Shop*. URL: https://shop.sensapex.com/product/system-of-one-in-vitro-micromanipulator-for-patch-clamp/.

[23] *STM32F4 - ARM Cortex-M4 High-Performance MCUs - STMicroelectronics*. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f4-series.html.

[24] *DeepLabCut — The Mathis Lab of Adaptive Motor Control*. URL: http://www.mackenziemathislab.org/deeplabcut.

[25] *Fusion 360 — 3D CAD, CAM, CAE & PCB Cloud-Based Software — Autodesk*. URL: https://www.autodesk.com/products/fusion-360/overview.

[26] César S. Mendes et al. "Quantification of gait parameters in freely walking wild type and sensory deprived Drosophila melanogaster". In: *eLife* 2013.2 (Jan. 2013). ISSN: 2050084X. DOI: 10.7554/ELIFE.00231.

[27] Matthew S. Creamer, Omer Mano, and Damon A. Clark. "Visual Control of Walking Speed in Drosophila". In: *Neuron* 100.6 (Dec. 2018), pp. 1460–1473. ISSN: 0896-6273. DOI: 10.1016/J.NEURON.2018.10.028.

[28] *What is Cartesian Robot? Explanation, Applications & Advantages - ElectricalWorkbook*. URL: https://electricalworkbook.com/cartesian-robot/.

[29] *MX7600 Series Manipulator*. URL: https://www.siskiyou.com/mx7600-series-manipulator.html.

[30] *World Precision Instruments — Micromanipulators — Motorised Manipulators — Sutter Instrument Manipulators — TRIO-245*. URL: https://www.wpi-europe.com/products/micromanipulators/motorised-manipulators/trio-245.aspx.

[31] *L-402 Miniature Linear Stage*. URL: https://www.physikinstrumente.com/en/products/linear-stages/stages-with-stepper-dc-brushless-dc-bldc-motors/l-402-miniature-linear-stage-1201902/.

[32] *Manual Micromanipulator MM33 by Märzhäuser. ergonomic, three, 3, axes, several, devices, cross, roller, bearings - Märzhäuser Wetzlar*. URL: https://www.marzhauser.com/en/products/micromanipulators/manual-micromanipulators/mm-33.html.

[33] *M-423-MIC Linear Stage Bundle*. URL: https://www.newport.com/p/M-423-MIC.

[34] *LSM Series - Miniature, Motorized Linear Stages - Zaber*. URL: https://www.zaber.com/products/linear-stages/LSM.

[35] *Motorized Rotation Stage - Motorized Positioners & Controllers - Catalog - Opto-Mechanical Products - Standa*. URL: https://www.standa.lt/products/catalog/motorised_positioners?item=244.

[36] *Curved Rails and Conveyor Systems — HepcoMotion*. URL: https://www.hepcomotion.com/products/curved-rails-and-conveyor-systems/.

[37] *Stepper Controllers Products at Phidgets*. URL: https://www.phidgets.com/?tier=2&catid=23&pcid=20.

[38] *Arduino - Home*. URL: https://www.arduino.cc/.

[39] *Arduino - Stepper*. URL: https://www.arduino.cc/en/reference/stepper.

[40] *AccelStepper: AccelStepper library for Arduino*. URL: http://www.airspayce.com/mikem/arduino/AccelStepper/.

[41] *Microstepping for Stepper Motors - Linear Motion Tips*. URL: https://www.linearmotiontips.com/microstepping-basics/.

[42] *NUCLEO-F401RE - STM32 Nucleo-64 development board with STM32F401RE MCU, supports Arduino and ST morpho connectivity - STMicroelectronics*. URL: https://www.st.com/en/evaluation-tools/nucleo-f401re.html.

[43] *Pololu - DRV8825 Stepper Motor Driver Carrier, High Current*. URL: https://www.pololu.com/product/2133.

[44] *Setting the Current Limit on Pololu Stepper Motor Driver Carriers - YouTube*. URL: https://www.youtube.com/watch?v=89BHS9hfSUk.

[45] *Basler ace acA1920-150um - Area Scan Camera*. URL: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca1920-150um/.

[46] *Form 2*. URL: https://formlabs.com/fr/3d-printers/form-2/.

[47] *stm32cubeide - Recherche Google*. URL: https://www.google.com/search?client=firefox-b-d&q=stm32cubeide.

[48] *STM32Cube Development Software - STM32 Open Development Environment - STMicroelectronics*. URL: https://www.st.com/en/ecosystems/stm32cube.html.

[49] *DMA not working in CubeMX generated code - order of initialization*. URL: https://community.st.com/s/question/0D50X0000Bmob3uSQA/dma-not-working-in-cubemx-generated-code-order-of-initialization.

[50] *MX_DMA_Init order in the main.c file generated by STM32CubeMX, How to fix?* URL: https://community.st.com/s/question/0D53W00001EzCmCSAV/mxdmainit-order-in-the-mainc-file-generated-by-stm32cubemx-how-to-fix.

[51] *Hands-On with STM32 Timers: Complementary Variable Frequency PWM - YouTube*. URL: https://www.youtube.com/watch?v=uv4VXhzi7Ys.

[52] *Generate stepper-motor speed profiles in real time - Embedded.com*. URL: https://www.embedded.com/generate-stepper-motor-speed-profiles-in-real-time/.

[53] *Applying acceleration and deceleration profiles to bipolar stepper*. URL: https://studylib.net/doc/8772662/applying-acceleration-and-deceleration-profiles-to-bipola....

[54] Alexander Mathis et al. "DeepLabCut: markerless pose estimation of user-defined body parts with deep learning". In: *Nature Neuroscience* 21.9 (Sept. 2018), pp. 1281–1289. ISSN: 15461726. DOI: 10.1038/S41593-018-0209-Y.

[55] Eldar Insafutdinov et al. "DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9910 LNCS (May 2016), pp. 34–50. ISSN: 16113349. DOI: 10.1007/978-3-319-46466-4{_}3. URL: https://arxiv.org/abs/1605.03170v3.

[56] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-December (Dec. 2016), pp. 770–778. ISSN: 10636919. DOI: 10.1109/CVPR.2016.90.

[57] Qiang Li et al. "Kalman filter and its application". In: *Proceedings - 8th International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2015* (Aug. 2016), pp. 74–77. DOI: 10.1109/ICINIS.2015.35.

[58] Hans R. Künsch. "Particle filters". In: *Bernoulli* 19.4 (Sept. 2013), pp. 1391–1403. ISSN: 13507265. DOI: 10.3150/12-BEJSP07.

[59] *CS231A Course Notes 1: Camera Models - Recherche Google*. URL: https://www.google.com/search?client=firefox-b-d&q=CS231A+Course+Notes+1%3A+Camera+Models.

[60] Zhengyou Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (Nov. 2000), pp. 1330–1334. ISSN: 01628828. DOI: 10.1109/34.888718.

[61] Semih Günel et al. "Deepfly3D, a deep learning-based approach for 3D limb and appendage tracking in tethered, adult Drosophila". In: *eLife* 8 (Oct. 2019). ISSN: 2050084X. DOI: 10.7554/ELIFE.48571.

[62] *SymPy*. URL: https://www.sympy.org/en/index.html.

[63] *Benchmarks — DLC Inference Speed Benchmark*. URL: https://deeplabcut.github.io/DLC-inferencespeed-benchmark/.

A Robotic platform assembly

The assembly protocol is presented hereafter. Note that a procedure similar to the one presented below can used if the rotational stage is first installed in the setup. In fact, this may not be possible to place it in the seven cameras setup if the other parts are already mounted.

An overview of the different parts is presented in Figure 70.

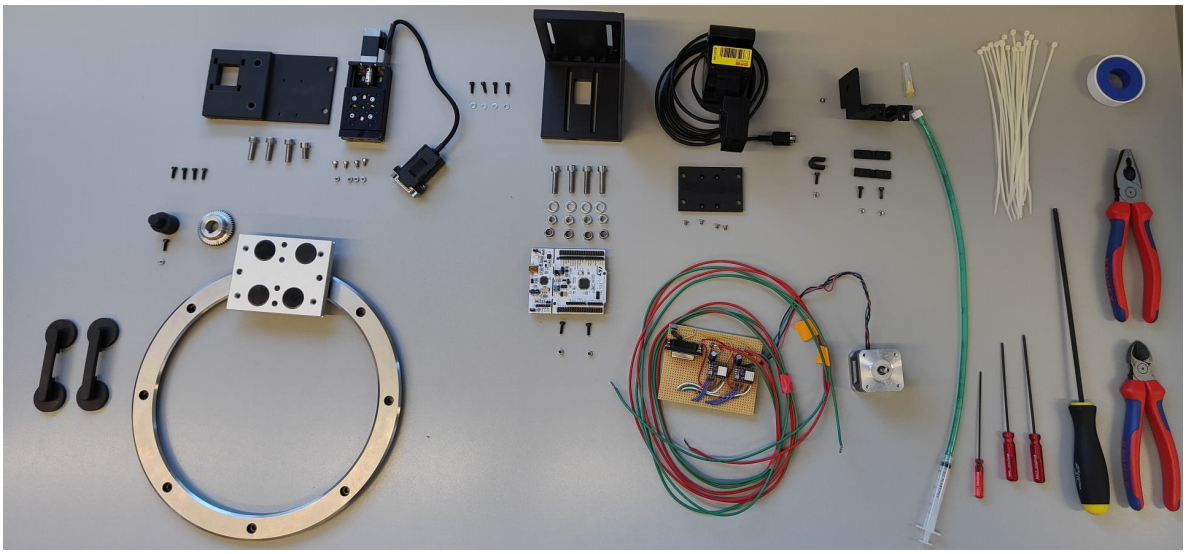
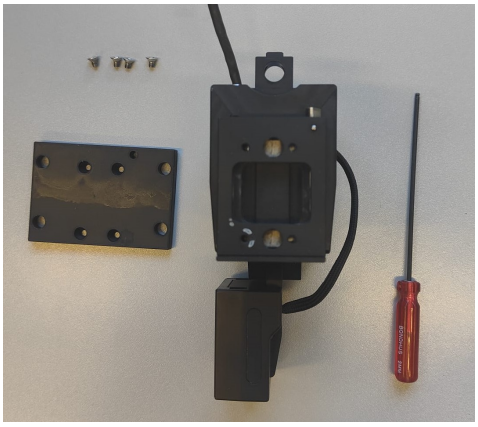


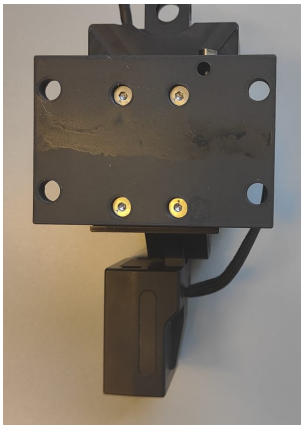
Figure 70: Parts overview

First, the base of the Sensapex should be mounted as indicated in Figure 71.

(a) Sensapex: base installation overview



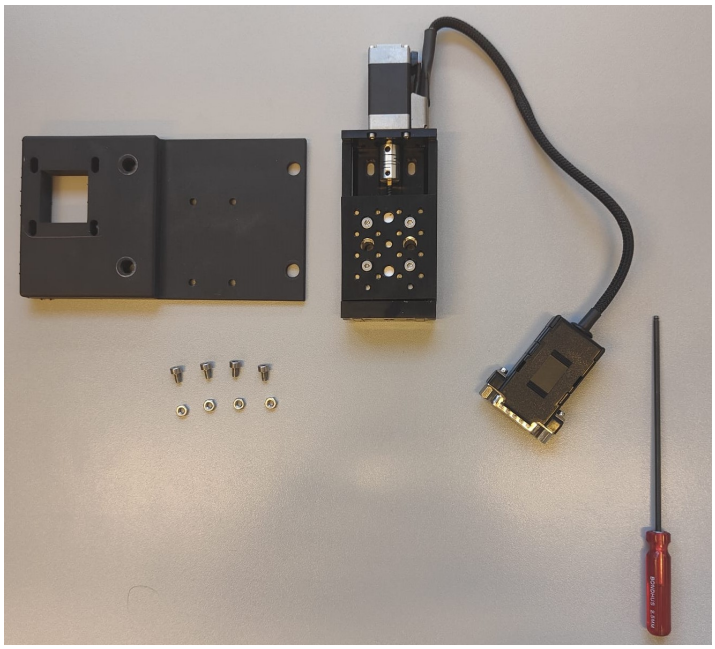
(b) Sensapex: base mounting holes



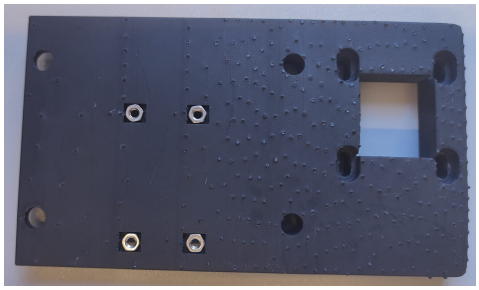
(c) Sensapex: base installed

Figure 71: Sensapex: base installation

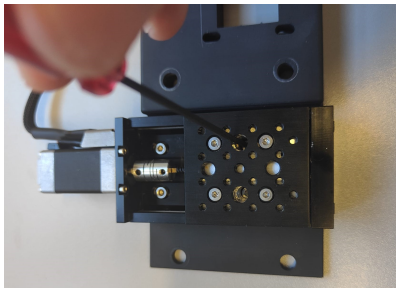
Next, the linear stage is installed onto the rotational platform:



(a) Linear stage installation overview



(b) Rotational platform nuts mounting holes



(c) Holes to mount the linear stage on top of the rotational platform

Figure 72: Linear stage installation

Then, the motor can installed onto the same platform (Figure 74).

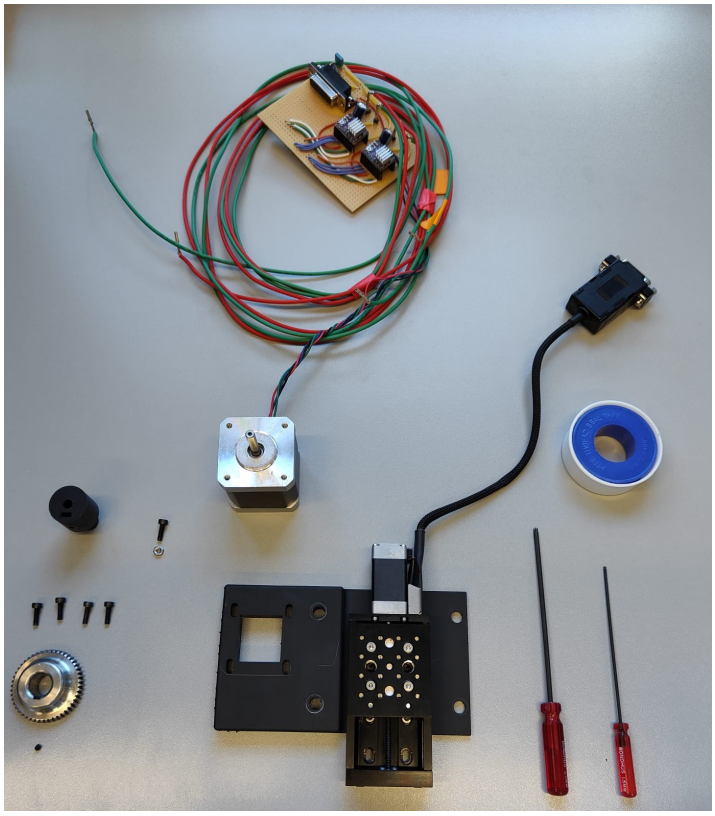
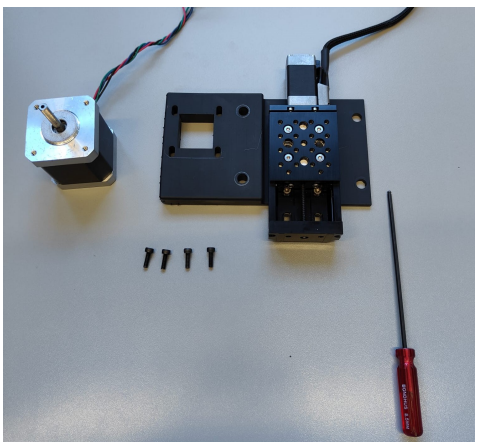
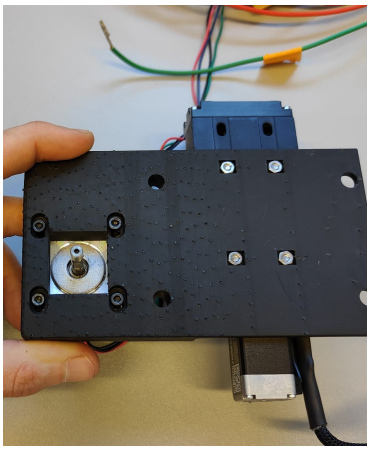


Figure 73: Rotational rotor and pinion installation overview



(a) Rotational motor installation overview



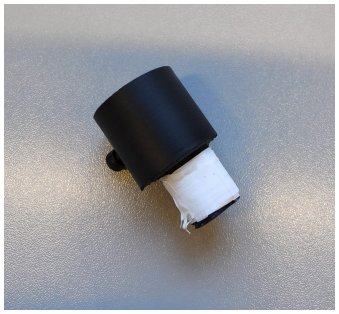
(b) Rotational motor installed

Figure 74: Rotational motor mounting

After that, the shaft adapter needs to be prepared as shown in Figure 75.

(a) Shaft adapter: nut insertion

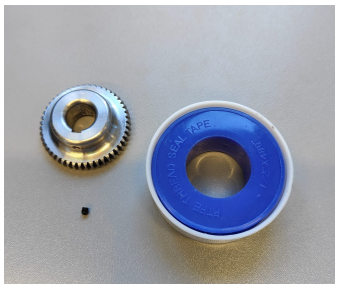
(b) Shaft adapter: fixing screw



(c) Shaft adapter: using Teflon to reduce mechanical game

Figure 75: Shaft adapter: preparation

The pinion is prepared as shown on Figure 76.



(a) Pinion preparation overview

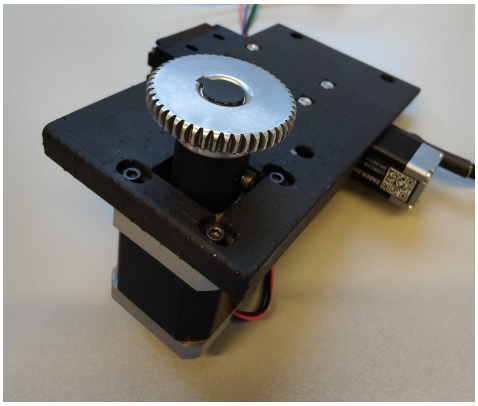
(b) Fixing screw with Teflon

(c) Shaft adapter: using Teflon to reduce mechanical game

Figure 76: Fixing screw inserted in pinion

And these parts are then mounted together:

(a) Shaft adapter mounted on the pinion



(b) Pinion installed on the motor

Figure 77: Pinion installation

Next, the rotational platform is mounted onto the carriage (Figure 78). Note that one of the screws is used to fix the linear stage cable by using a cable tie.

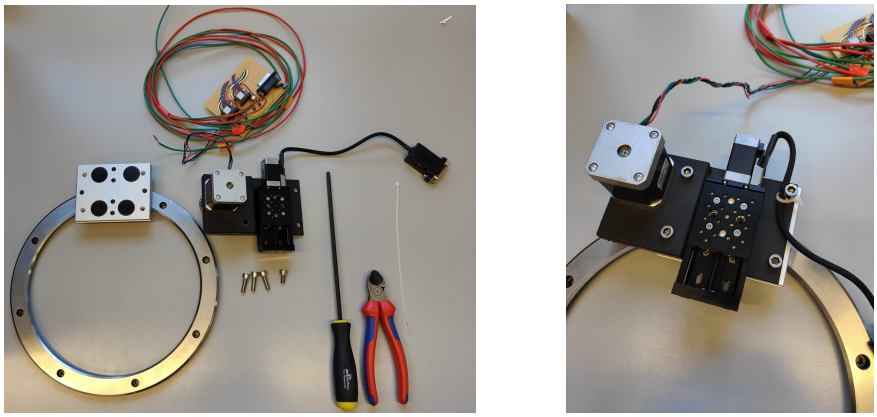
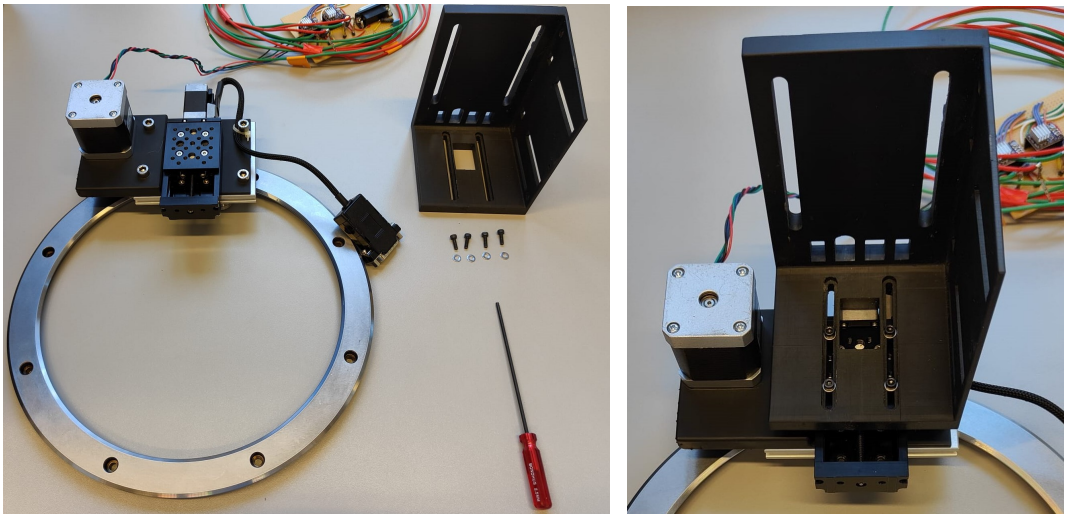


Figure 78: Installation of the rotational platform mounted on the rotational carriage

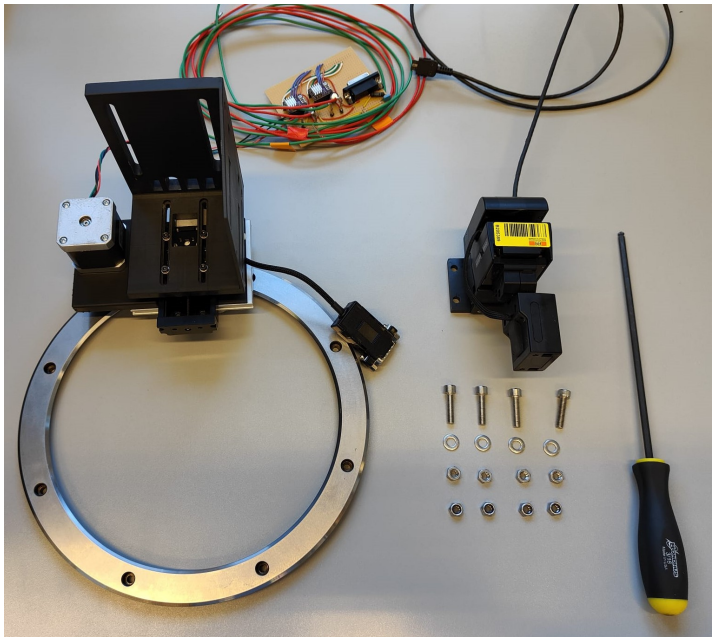
The bracket is installed onto the linear stage as shown in Figure 79.



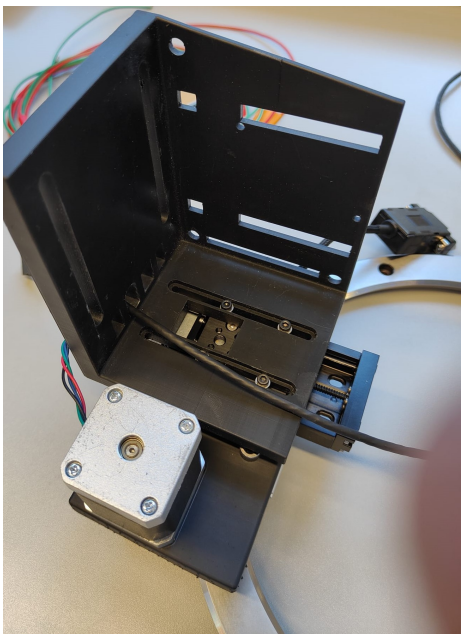
(a) Bracket installation overview (b) Bracket installed

Figure 79: Bracket installation

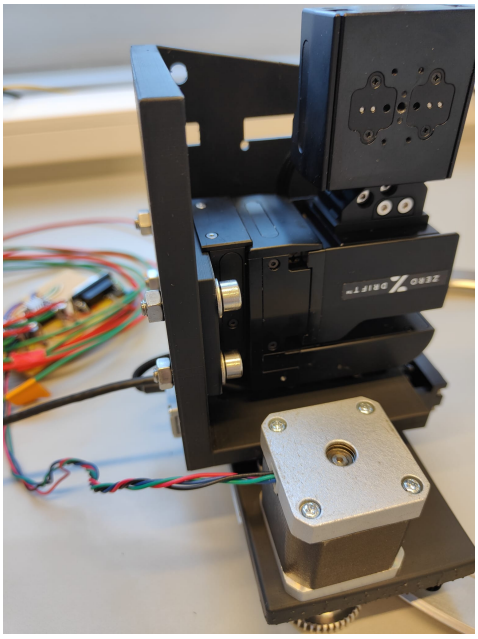
Then, the Sensapex is mounted on the bracket (Figure 79). Its cable exits the bracket through one of the holes and one should verify that the cable sits in the bottom rail on the corresponding side. This prevents the cable from being bent too much.



(a) Sensapex: mounting on the bracket overview



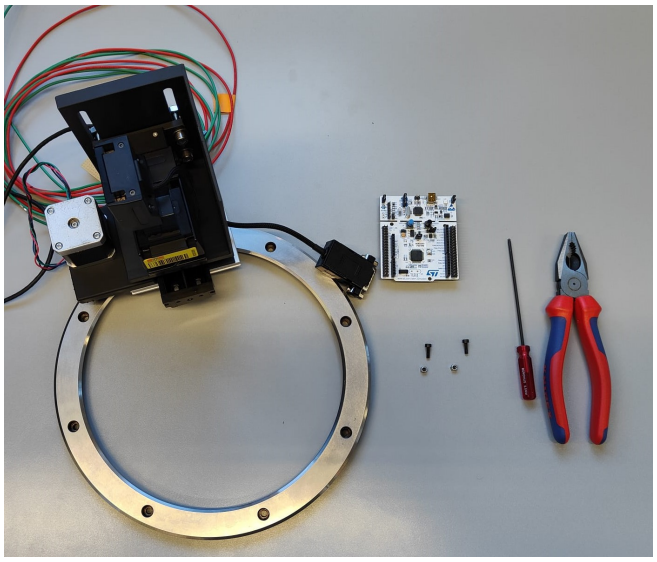
(b) Sensapex: cable pass through hole



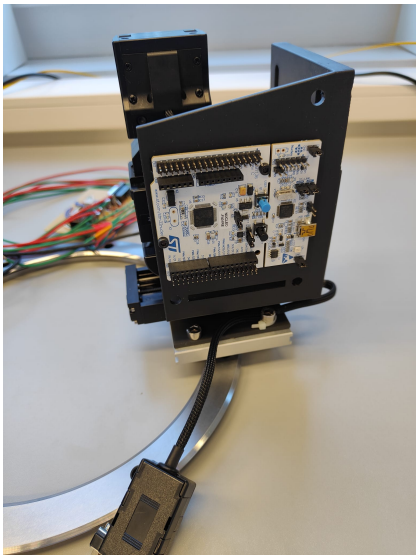
(c) Sensapex mounted on the bracket

Figure 80: Sensapex: mounting on the bracket

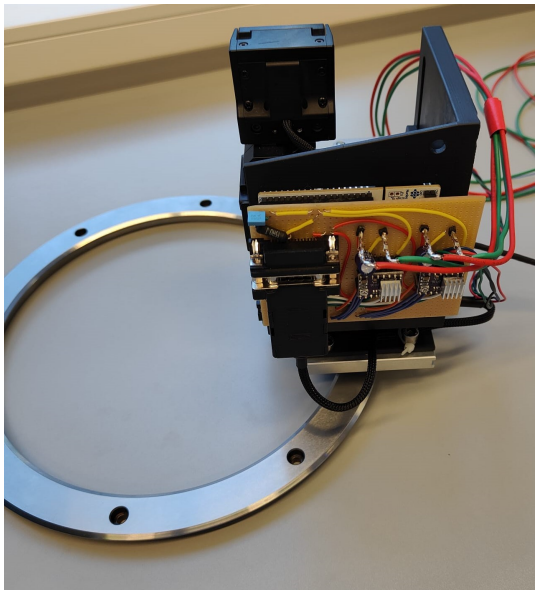
After that, the electronics is mounted on the side surface of the bracket.



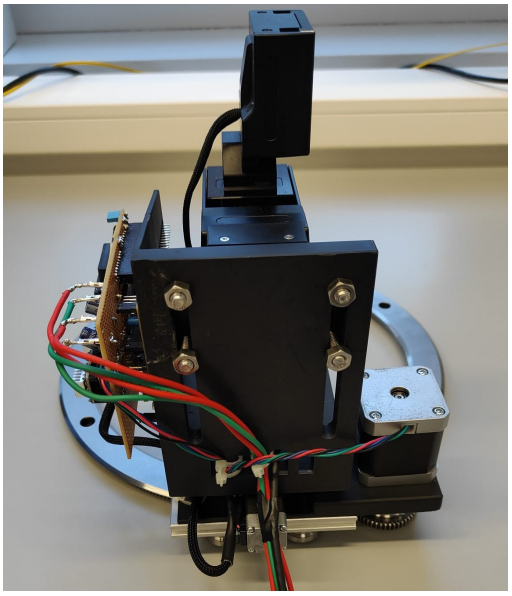
(a) Electronics mounting overview



(b) MCU mounted



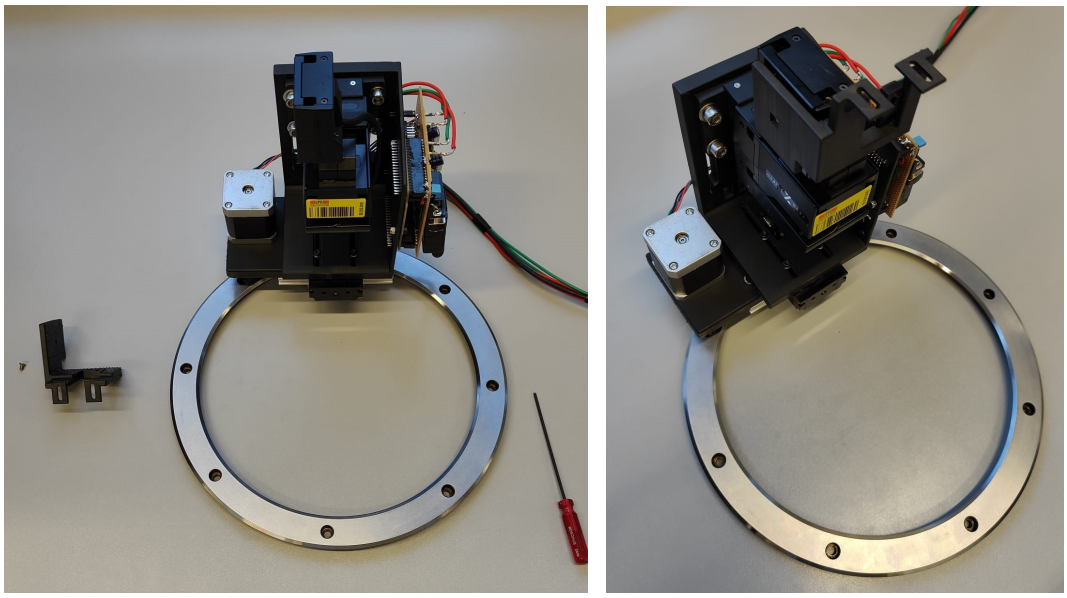
(c) Prototype board installed



(d) Cable management

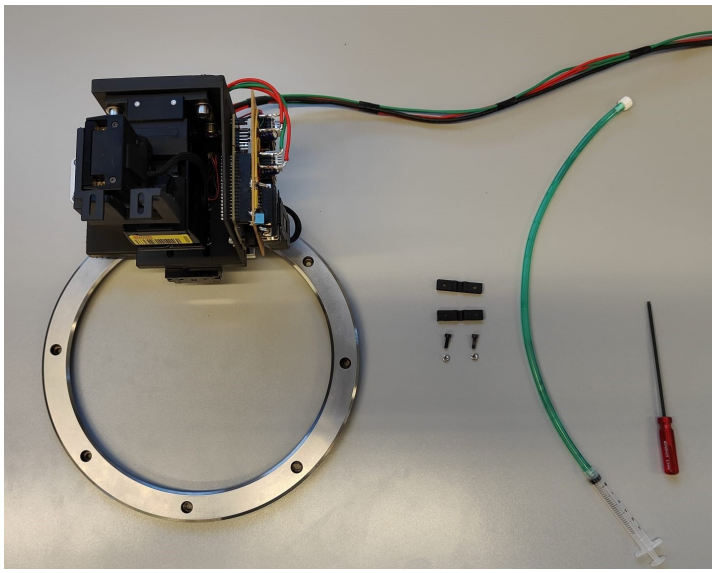
Figure 81: Electronics installation

The end effector is then installed as shown in Figures 82, 83 and 84.



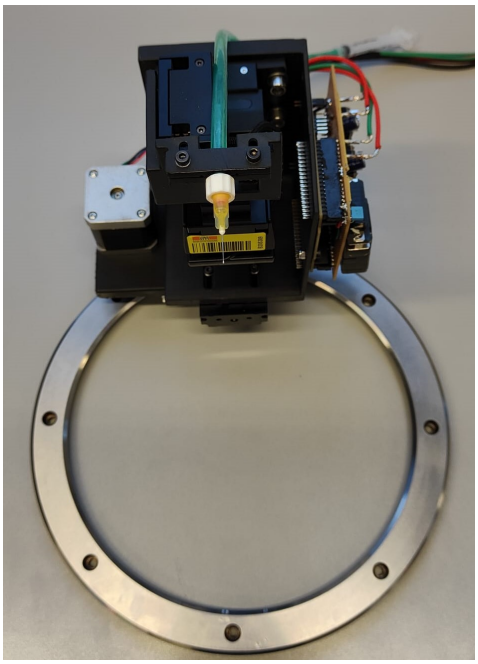
(a) Base of the end effector overview (b) Base of the end effector mounted

Figure 82: Base of the end effector installation



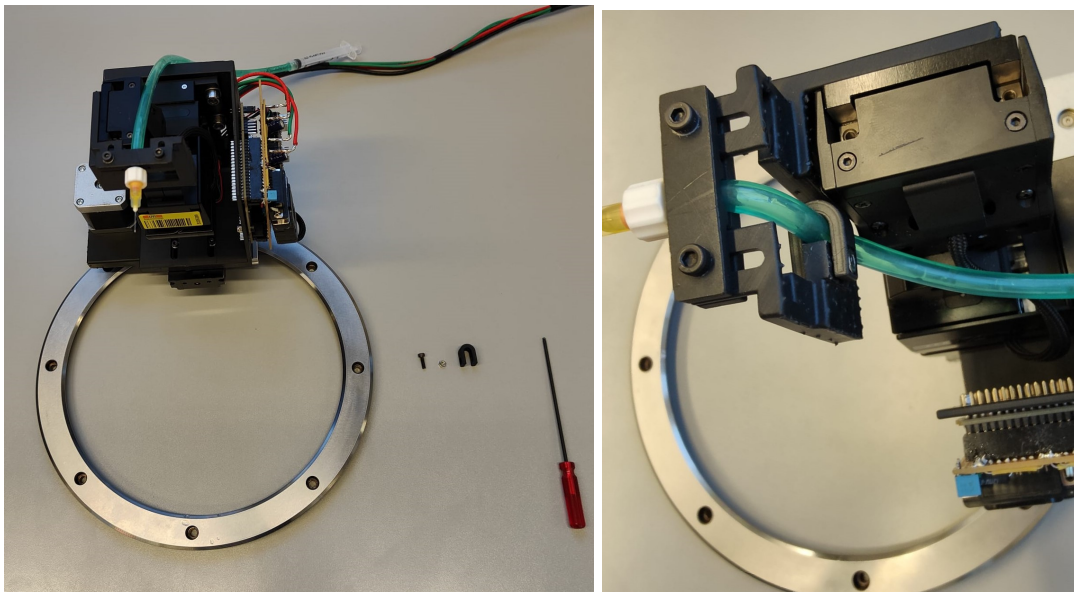
(a) Mounting of the needle overview

(b) Bottom holder nuts insertion



(c) Needle mounted

Figure 83: Installation of the needle



(a) Mounting of the tube holder overview (b) Tube holder mounted

Figure 84: Installation of the tube holder

Finally, the rail is fixed to the optical breadboard:

Figure 85: Fixing the rail to the base plate

B Prototype board layout

The connection scheme is presented in Figure 86.

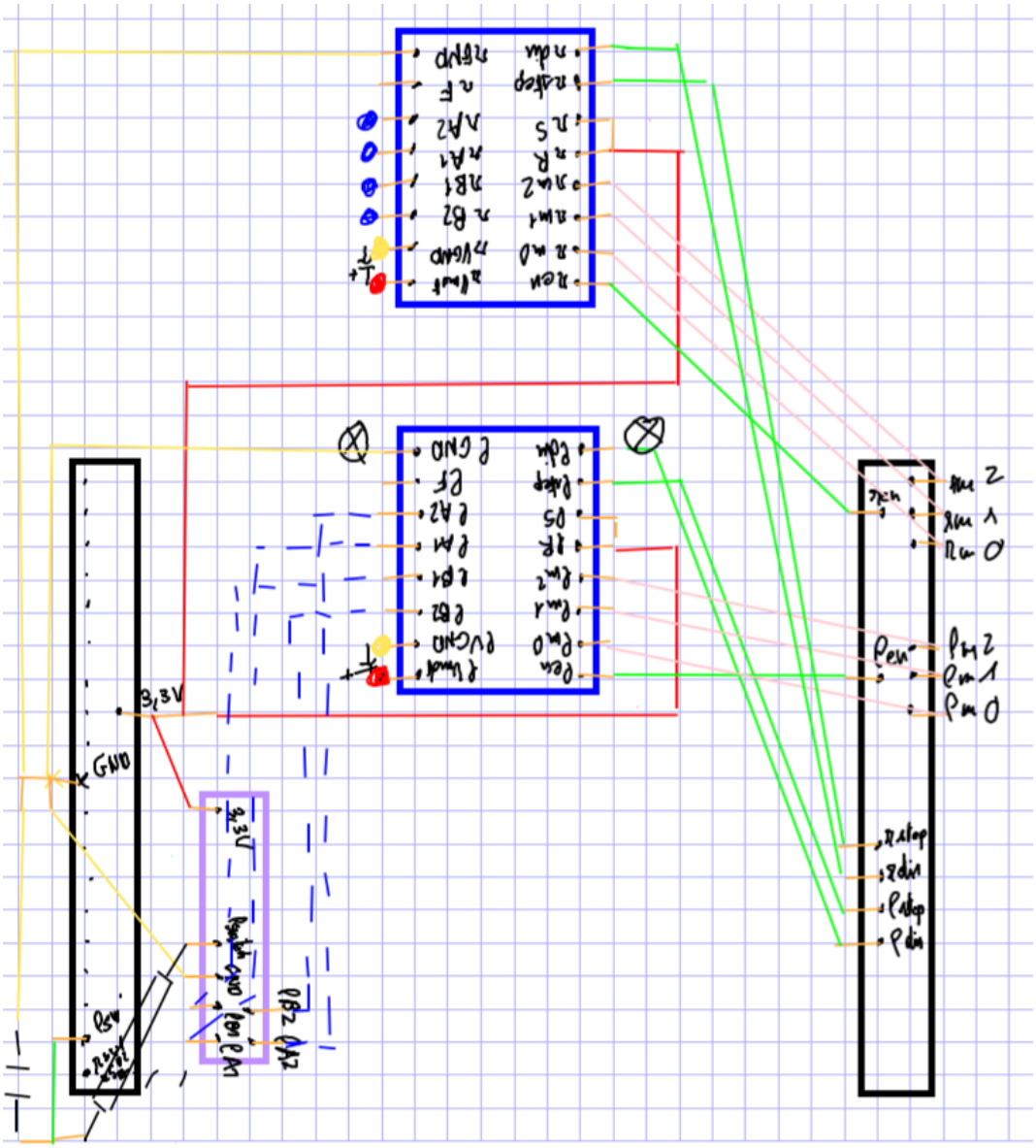


Figure 86: Prototype board layout

The blue, black and violet rectangles represent the stepper drivers, the MCU pins slots and the DB-15 connector respectively.

The cables passing underneath the board are depicted as dotted lines and the dots represent cables coming onto the board. The black dot with a cross inside represent the button on the MCU. They are show in order to ensure that the cables passing underneath the board do not push on these.

C Closed loop model

The derivation of the closed algorithm model (Figure 87), used to transfer pixel positions to the Cartesian 3D space, is presented hereafter.

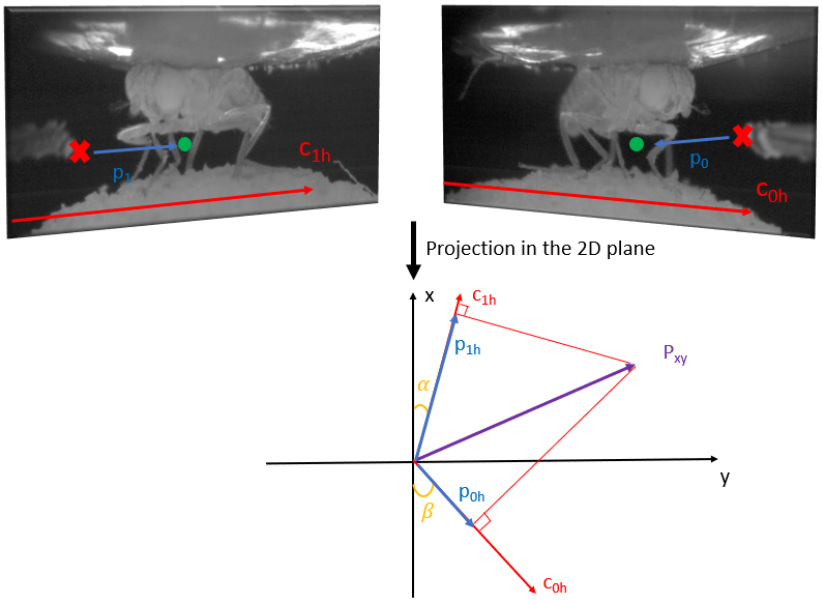


Figure 87: Closed loop control algorithm model. The projection in the horizontal plane does not correspond to those images for illustrative purposes

The coordinate of p_{0h} and p_{1h} in the x,y coordinate system are expressed as follows:

$$x_{p_{0h}} = -p_{0h}cos(\beta) \tag{10}$$
$$y_{p_{0h}} = p_{0h}sin(\beta) \tag{11}$$
$$x_{p_{1h}} = p_{1h}cos(\alpha) \tag{12}$$
$$y_{p_{1h}} = p_{1h}sin(\alpha) \tag{13}$$

The slopes of p_{0h} and p_{1h} in the x,y coordinate system are given below:

$$\delta_{p_{0h}} = -tan(\beta) \tag{14}$$
$$\delta_{p_{1h}} = tan(\alpha) \tag{15}$$

Knowing that the projection of P_{xy} is perpendicular to the cameras axis, the equation of the dotted lines from this point to the cameras axis are derived:

$$P_{p_{0h}} = \frac{1}{tan(\beta)}x + b_{p_{0h}} \tag{16}$$
$$P_{p_{1h}} = -\frac{1}{tan(\alpha)}x + b_{p_{1h}} \tag{17}$$

Using the fact that these dotted lines go through p_{0h} and p_{1h} respectively, the intercepts $b_{p_{0h}}$ and $b_{p_{1h}}$ can be calculated:

$$p_{0h}sin(\beta) = \frac{1}{tan(\beta)}(-p_{0h}cos(\beta)) + b_{p_{0h}} \tag{18}$$

$$b_{p_{0h}} = p_{0h}sin(\beta) + \frac{1}{tan(\beta)}p_{0h}cos(\beta) = p_{0h}\frac{sin(\beta)^2 + cos(\beta)^2}{sin(\beta)} = \frac{p_{0h}}{sin(\beta)} \tag{19}$$

$$p_{1h}sin(\beta) = -\frac{1}{tan(\beta)}(p_{1h}cos(\beta)) + b_{p_{1h}} \tag{20}$$

$$b_{p_{1h}} = p_{1h}sin(\beta) + \frac{1}{tan(\beta)}p_{1h}cos(\beta) = p_{1h}\frac{sin(\alpha)^2 + cos(\alpha)^2}{sin(alpha)} = \frac{p_{1h}}{sin(\alpha)} \tag{21}$$

Hence, the equations of the dotted lines is the following:

$$P_{p_{0h}} = \frac{1}{tan(\beta)}x + \frac{p_{0h}}{sin(\beta)} \tag{22}$$

$$P_{p_{1h}} = -\frac{1}{tan(\alpha)}x + \frac{p_{1h}}{sin(\alpha)} \tag{23}$$

One can then find the corresponding x, by finding the intersection of these lines:

$$\frac{1}{tan(\beta)}x + \frac{p_{0h}}{sin(\beta)} = -\frac{1}{tan(\alpha)}x + \frac{p_{1h}}{sin(\alpha)} \tag{24}$$

$$x(-\frac{1}{tan(\beta)} - \frac{1}{tan(\alpha)}) = \frac{p_{0h}}{sin(\beta)} - \frac{p_{1h}}{sin(\alpha)} \tag{25}$$

$$x = \frac{-\frac{p_{0h}}{sin(\beta)} + \frac{p_{1h}}{sin(\alpha)}}{\frac{1}{tan(\beta)} + \frac{1}{tan(\alpha)}} = \frac{\frac{-p_{0h}sin(\alpha)+p_{1h}sin(\beta)}{sin(\alpha)sin(\beta)}}{\frac{cos(\beta)sin(\alpha)+cos(\alpha)sin(\beta)}{sin(\alpha)sin(\beta)}} = \frac{-p_{0h}sin(\alpha) + p_{1h}sin(\beta)}{sin(\alpha+\beta)} \tag{26}$$

And y can be derived by substituting x and the intercept in equation 17:

$$y = \frac{1}{tan(\beta)}\frac{-p_{0h}sin(\alpha) + p_{1h}sin(\beta)}{sin(\alpha+\beta)} + \frac{p_{0h}}{sin(\beta)} = \frac{p_{0h}cos(\alpha) + p_{1h}cos(\beta)}{sin(\alpha+\beta)} \tag{27}$$

D Needle preparation

The preparation of the needle is discussed below.
The tip of the needle is first grinded by using the surface of a Dremel cutting tool:

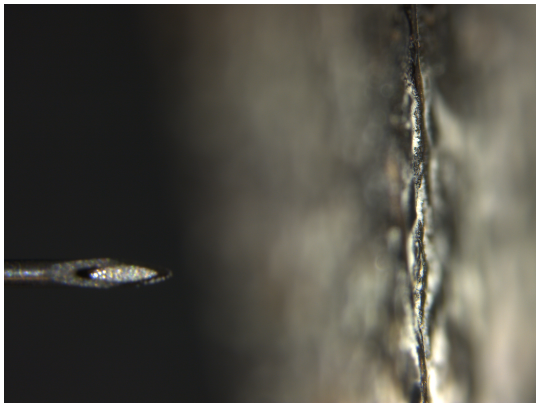
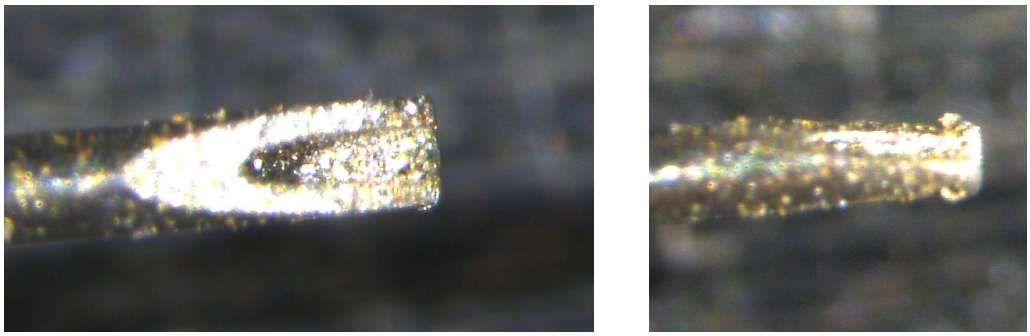


Figure 88: Needle grinding

It may be necessary to remove some material at the grinded tip. The desired result is show on Figure 89.



(a) Top view of the grinded needle

(b) Side view of the grinded needle

Figure 89: Grinded needle

Next, some acrylic is put at the needle tip by using a soldering iron:

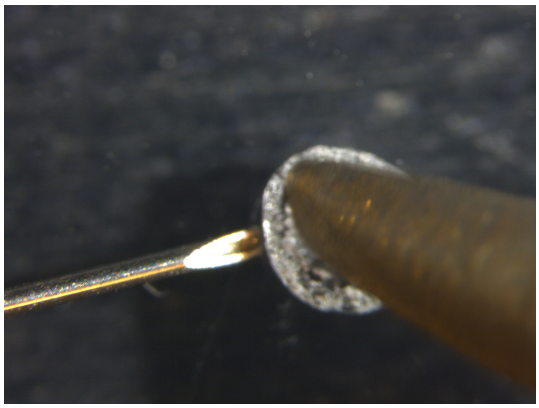
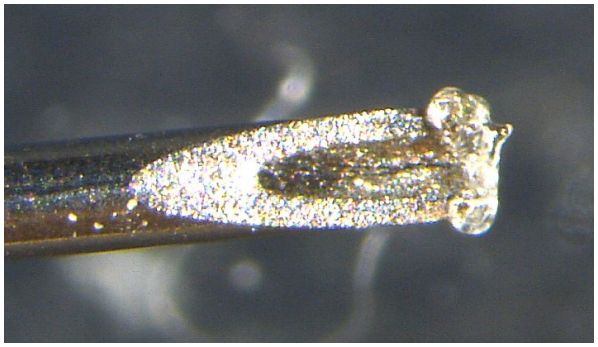
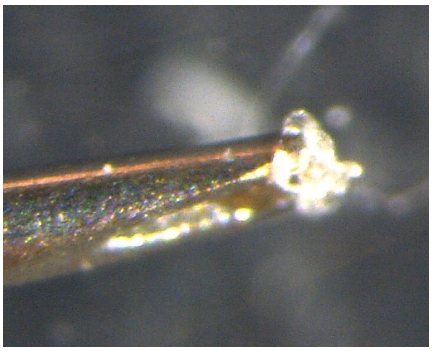


Figure 90: Plastic deposition at the needle tip

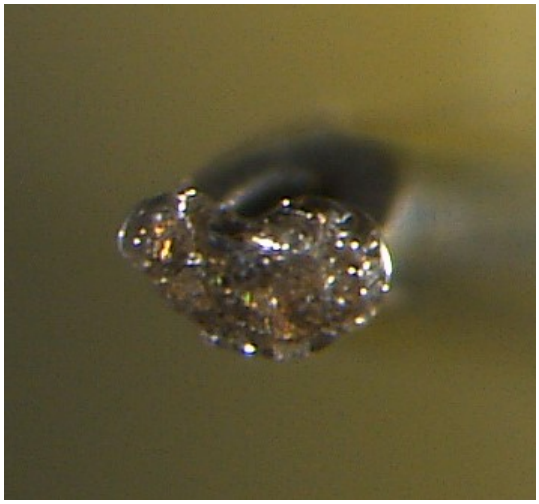
The plastic at the tip of the needle should be similar to the result presented in Figure 91.



(a) Top view of the needle with plastic at the tip



(b) Side view of the needle with plastic at the tip



(c) Front view of the needle with plastic at the tip

Figure 91: Needle tip with plastic

Finally, a small piece kimwipe – which has been rolled into a thread – is put into the needle (Figure 92).

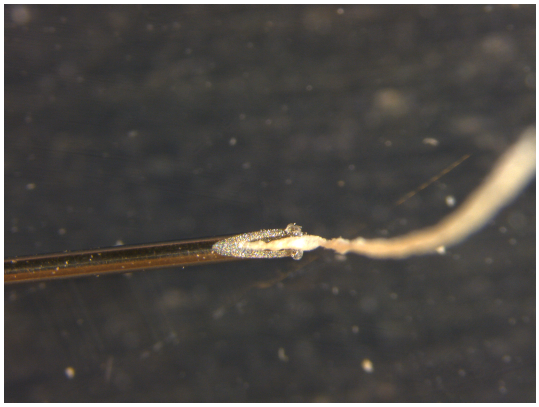
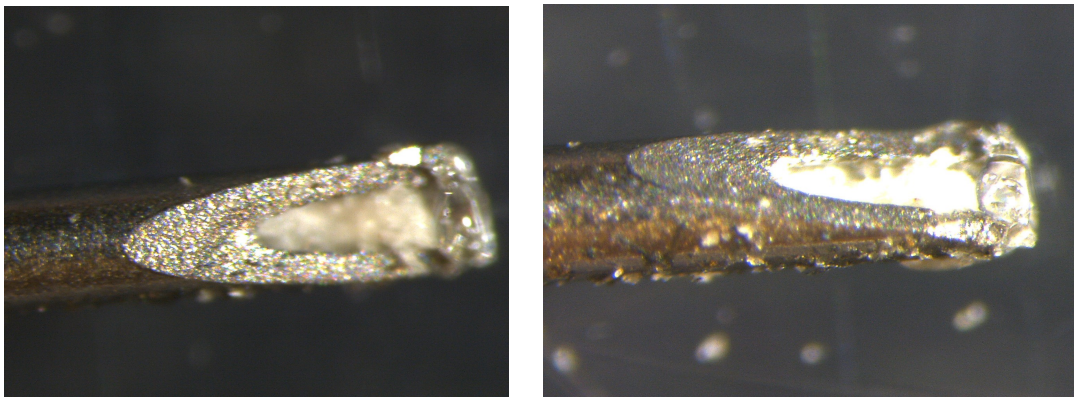


Figure 92: Inserting thread in the needle

The end result is shown in Figure 93.



(a) Top view of the finished needle

(b) Side view of the finished needle

Figure 93: Finished needle

E Using the robotic system

Before launching the different provided scripts, the rotational stage should be placed manually at its origin position (to the left of the spherical treadmill when looking in the cameras direction, against the mechanical stop). Note that manually moving the robotic system should only be done when the power stages of the stepper drivers are deactivated.

Next the robotic system can be commanded by typing commands in the terminal. The commands for the stepper motors are the same as indicated in the communication protocol:

byte	19	18	17	16 ... 0
content	ID	mode	direction	message content
input	'Z': Zaber linear stage / 'R': rotational stage	'R': relative movement in steps / 'V': for velocity control / 'G': for getting the motor position / 'A': for activating/deactivating the power stage / 'N': for reinitializing the MCU	direction: '+' / '-'	message content: in mode 'R': <number of steps>,<speed> (speed can be omitted) / in mode 'V': <speed>. The speed is the period between steps in clock count. The end of the command is indicated by '\r'

Table 4: Communication protocol

The Sensapex can be commanded using the wheel or through the terminal by typing the following command: 'U<axis><direction('+'/'-')><number of microns>'.

The axes are:

- X for the radial direction, positive when moving towards the spherical treadmill;
- Y for the tangential direction, positive when moving towards the right (when looking in the cameras direction);
- Z for vertical direction, positive when moving upward.

For the user defined goal/origin script, the goal and the origin are indicated by typing 'G' and 'O' in terminal respectively. When both positions have been defined and the Sensapex is at the origin position, the command 'GO' can be typed in the terminal and the user will be required to enter the time lapse during which the robotic system should stay at the goal and the time interval at which it should visit the goal.

For the automatic feeding script, the robot's end effector is first placed into the cameras field of view using the above mentioned commands (and possibly the wheel). Then, the command 'GO' can be typed in the terminal and the robotic system starts going to the goal.

Finally, for the trajectory script, the trajectory starts when the script is launched. Note that it is particularly important to place the rotational stage correctly before launching this script. Otherwise, the rotational stage may hit the mechanical stop at the end of its travel range.