Horizon 2020

Societies of Symbiotic Robot-Plant Bio-Hybrids as Social Architectural Artifacts

Deliverable D1.4

Evaluation of the robotic symbiont

Date of preparation: 2019/03/31	**Revision: 1**
Start date of project: 2015/04/01	**Duration: 48 months**
Project coordinator: UzL	**Classification: public**
Partners:	
lead: **ITU**	*contribution:* **CYB, CITA, UzL**
Project website:	http://florarobotica.eu/

H2020-FETPROACT-2014

DELIVERABLE SUMMARY SHEET

Grant agreement number:	640959
Project acronym:	*flora robotica*
Title:	Societies of Symbiotic Robot-Plant Bio-Hybrids as Social Architectural Artifacts

Deliverable N^o:	Deliverable D1.4
Due date:	M48
Delivery date:	2019/03/31

Name:	**Evaluation of the robotic symbiont**
Description:	This deliverable provides an overview of our achievements to complete the objectives of WP1: (1) the mechatronic basis for *flora robotica*; (2) Interaction mechanisms between the robotic and biological element of *flora robotica*; (3) Software abstraction that allows efficient programming and experimentation with *flora robotica*. To this end we cover braiding of robotic symbionts, shaping and control of the robotic symbionts, the Measurement Unit (MU) for obtaining sensor data from plants, and provide a software interface for obtaining and analyzing this data. Finally, we present our robotic nodes for controlling the growth of plants. Overall, we conclude that we have reached the objectives we set out to achieve at the beginning of the project.

Partners owning:	**ITU**
Partners contributed:	**CYB, CITA, UzL**
Made available to:	public

Contents

1 Introduction: Overview of robotic symbiont

This deliverable is the last in a series of three reporting on the robotic symbiont. In D1.1 we reported on our exploration work regarding a suitable technological basis for the robotic symbiont. At the start of the project it was anticipated that the robotic symbiont could be based on the existing LocoKit robot construction system. However, in D1.1 we established that LocoKit was too limited to work as a robotic symbiont and presented a rather large range of potential alternative technologies. In D1.2 we had focused our efforts and reported on the four technologies that have become the core technologies of the project. It was the braided robots together with the braiding machine, which provide the structural basis and actuation. It was the robotic node used to control plant shaping using light at different wave-lengths. Finally, it was the Measure Unit (MU) used to obtain sensor data from plants. Together these technologies provide a complete basis for a robotic symbiont including structure, robotic action, plant actuation, and plant sensing. In D1.3 we provided a progress report and in this final deliverable D1.4 we will provide a final report on these technologies.

In the previous deliverable D1.3 we presented the braiding machine and demonstrated that it was able to braid a variety of structures by controlling the braid module configuration and the braid pattern (e.g., cylinders, inter-braided double cylinders, division and merging of substructures, etc.). The braiding machine overall remains the same, but has seen small, but important improvements that make it more robust and able to braid with a wider range of materials. A specific example is that we used the braiding machine to braid columns from wooden strips for the demonstrator. A key problem of the braiding machine and the braid production work was that we did not have a way to come from a high-level design of a desired braid to a braid machine configuration and controller. This challenge has been the focus of the last period. We have attacked this problem from two directions.

In Sec. 2.2 we report on work where we develop a theoretical basis for the braiding machine that allows us to verify if a specific controller and braid configuration lead to collisions in the braiding process. We can view this as a theoretical bottom-up process to understand the limitations of the braiding machine.

In Sec. 2.3 we report on the other direction where a 3D model designed by an architect is compiled into a braiding machine configuration and controller. This work represents a key contribution of this period as we now have the pieces for a tool chain that goes from high-level 3D CAD model to the braided structure (akin to the tool chain in 3d printers where you go from CAD model to 3D printed model through slicer software to GCode commands to control the 3d printer). Furthermore, we present a simulator for simulating the resulting braid (before it is produced on the machine).

Once the braids are produced a key aim of the project was to understand how we can actuate them. In D1.3 we reported on how to do this using wires attached to servo motors. We also explored an alternative to servo motors in the form of NiTinol Nichrome composite wires. In Sec. 3.1 we present a third approach where distribution actuation modules are placed on the braid and control the angle between two strands of the braid and thereby deform the braided structure. We also report in Sec. 3.2 on preliminary work towards a new braid actuator based on twisted fibres. Finally, we continue in Sec. 3.3 to demonstrate how controllers for braid shaping can be evolved in simulation.

In Sec. 4, we report that the Measurement Unit (MU) developed by our SME partner Cybertronica (CYB) has matured further and is now certified and made available as a commercial product. The product is demonstrated and it is documented how it performs in an outdoor setting and how the MU as a standalone can be used to control a robot arm directly from sensor readings from the plant.

Finally, in Sec. 5 we report on the latest version of the robotic node for shaping plants which has seen a step-change in performance and drastic reduction in size (half compared to what was reported in D1.2). We further present a protocol for performing plant-robot experiments.

In summary, we cover all the mechatronic aspects of robot-plant interaction of this project. The production of braided structures and the underlying understanding of the process is a key contribution of the project. Another key contribution is the plant shaping work showing experiments with both plants and robots. Also, worth mentioning is that the Measuring Unit has reached a level of maturity uncommon for research projects. The actuation of braids is still work in progress as this was only the focus in the last part of the project.

2 Producing Braided Structures

In this section we introduce the braiding machine, a theoretical investigation of the limitations of the machine, and the steps required to go from a high-level 3D model to the robot configuration and controller that can produce this braid.

2.1 The braiding machine

This section is included to provide an overview of the braiding machine. This is provided for completeness, but is a summary of the corresponding section in D1.3. Hence, if the reader is aware of the general principles of the braiding machine the reader can skip to Section 2.1.4.

2.1.1. General functionality of modules

The functionality of the braiding machine is to transport carriers of filament in interweaving patterns to produce continuous reciprocal 2D or 3D braid structures. For the machine to be both cost-effective and versatile reconfiguration was essential as it allows for a variety of braid patterns from a minimum of hardware modules. The control software is designed for low level control of the microprocessor in the machine as well as for the high level control of reconfiguration, testing and simulation of carrier transportation.

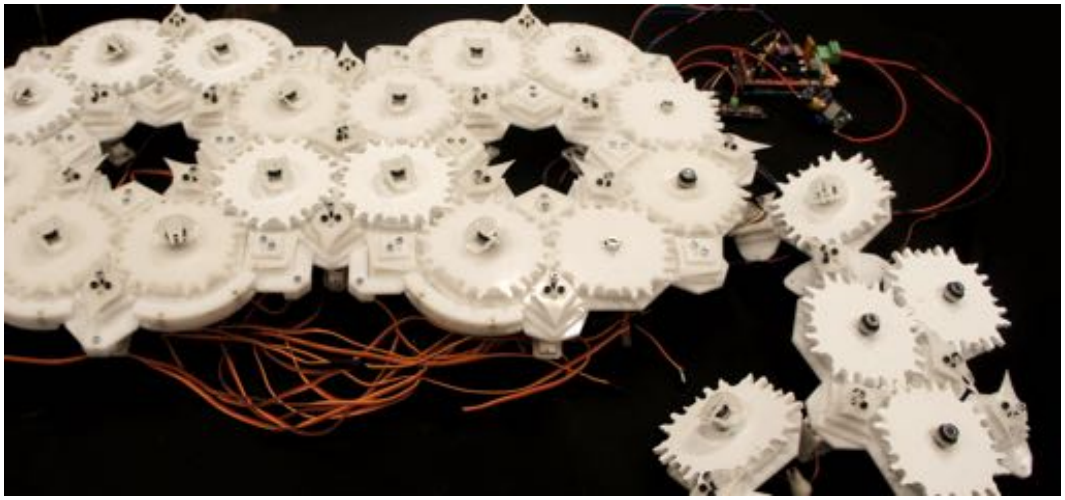


Figure 1: Braiding machine: reconfigurable hardware modules

2.1.2. Mechanics

To move carriers in various crossing transportation routes, four different types of reconfigurable modules were designed see Fig. 1. The (1) drivers of the braiding machine are designed to build tracks for the transportation routes while (2) junctions are routing the (3) carriers on their way through the machine. (4) Borders prevent the carriers from falling off the open edges of the drivers. Transportation routes can be designed by assembling octagonal driver modules into specific configurations. The number of possible configurations grow exponentially with the number of modules available. Drivers are equipped with a rotating sprocket on top transferring torsion

between drivers. Hence, they turn clockwise and counter clockwise respectively. This sprocket design is combined with eight slots as seats for carriers to reduce parts, cost and complexity. The driver baseplate is octagonal allowing it to connect with a straight edge to adjacent modules. On top of this baseplate two circular plates of different diameter are fixed. The cross-section of these plates make up one half of the concave rail while adjacent rim, junction and driver modules make up the other half. Junctions brace and fasten the driver modules, and the servo motor on its bottom actuates the top switching tooth to route carriers passing by. All servos in the junctions are controlled from a 16-channel 12-bit PWM/servo driver through I^2C interface to a micro controller using an Atmega2560 processor.

2.1.3. Electronics

The Atmega2560 micro controller board acts as low level timer and is connected to a number of integrated circuits. For a schematic overview see Fig. 2. (1) The aforementioned servo driver controlling all junctions. (2) An encoder with 6000 steps/revolution connected on one of the driver sprockets tracks position of the rotating sprockets constantly. We use a 12-bit encoder, the angle of rotation is represented by a number between 0 and 2048 (2^{11}). Therefore, the reading is accurate to about 0.09°. (3) A Rampsboard v.1.4 with up to 5 stepper motor drivers control up to 5 Nema 17 stepper motors which actuates the rotation of sprockets. Sprockets are directly mounted on the stepper motor axel to simplify design, ensure easy control of torque and manual overrule and to reduce damage at jamming by avoiding additional gears. A computer running the control software sends commands over the serial port to the microcontroller. The commands buffer to the internal memory of the microcontroller, which in turn executes the servo commands in correlation with sprocket position.

Figure 2: Direction of messages passed between software, middleware, and hardware.

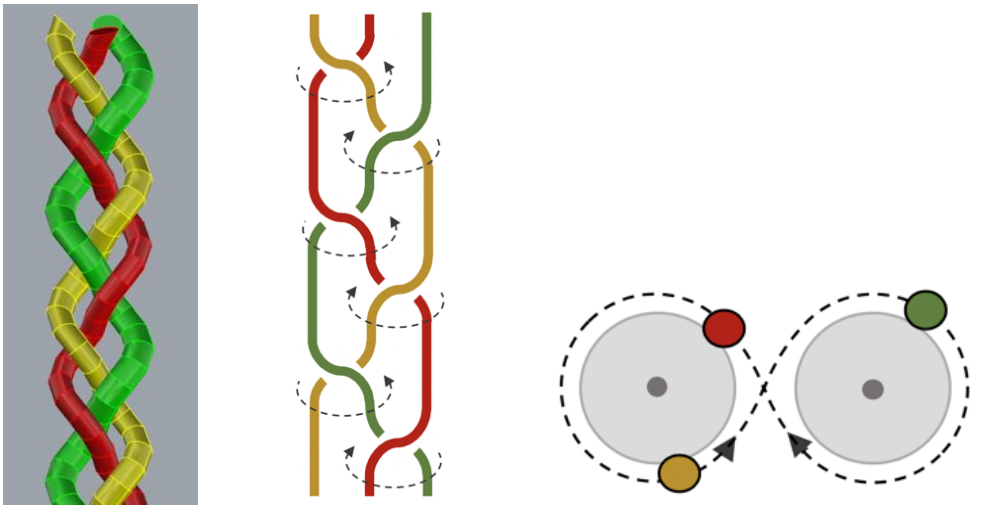
2.1.4. Developments

During this reporting period, we have made a number of small improvements to the braiding machine. The most general improvement was to deal with the misalignment and bending of the switches when braiding with stiffer materials. In order to mitigate it we changed the thickness of the switch headers and arms from 1mm to 2 mm. Furthermore, we increased their rotational range which allows us to set the range in firmware depending on the stiffness of the material, that replaces a tedious mechanical configuration step with every new braid with stiff material. In conclusion, the maturity of the braiding machine has increased in the last period and small modifications has been made to increase its robustness and ability to braid with a variety of materials.

2.2 Braiding with the machine: a theoretical view

A braid is a complex structure (pattern) formed by interlacing three or more strands of flexible material [3]. In principle, all strands of a braid are functionally equivalent with respect to interlacing pattern through the others. The minimum required number of strands for a braid is three that forms a flat and solid structure, for example, a rope or braided hair. However, to braid more complex structures, larger number of strands are needed. For instance, the minimum number of strands to form a tubular braid is four. A wide branch of research in mathematics, called braid theory, studies the topological concept of braids [1, 2] by defining braid groups and configuration spaces. In braid theory, the complex interlacing patterns of braided structures can be described as a sequence of single braiding operations on pairs of strands one at a time.

To produce a braid, each strand follows a certain pattern of interlacing interactions with other strands. The interactions are the acts of one strand passing under or over another strand. The interlacing pattern of interactions usually repeat for several rounds or for the whole braid. The patterns of the under/over passes determine some of the mechanical properties and the shape of the braid. A change in the pattern leads to a change in the shape or properties of the braided structure.



(a) 3-strands braid

(b) movement pattern of strands

(c) configuration consisting of 2 oppositely rotating discs to achieve the movements

Figure 3: (a) An example 3-strand braid, (b) the movements of the strands necessary for producing the braid, and (c) a mechanism for producing the movements of the strands to achieve the barid.

The interaction pattern can be achieved by using combinations of clockwise and counter clockwise rotational moves of the strands in relation to each other. Fig. 3 shows a simple 3-strands braid, the rotational moves of the strands, and a mechanism to achieve the pattern of the moves. As demonstrated in the figure, the movement pattern can be produced by using a configuration of two discs rotating in opposite directions. The discs guide the movement of the strands and exchange the strands between each other. The strands in this configuration are distanced (semi-)equally on the movement path and are exchanged between the discs at the crossings.

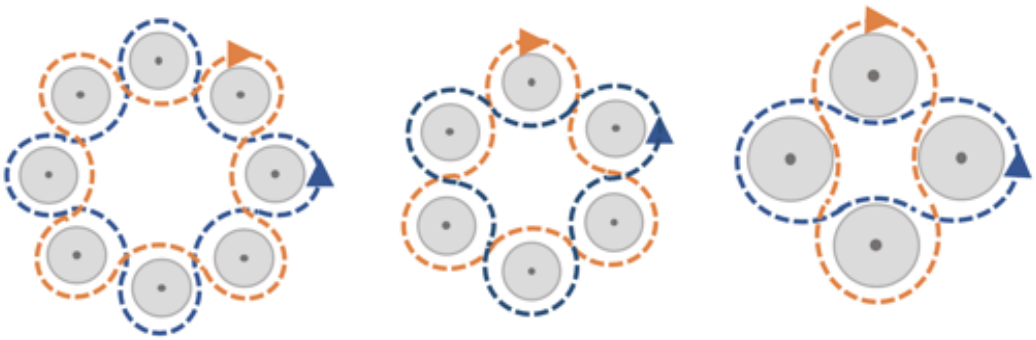


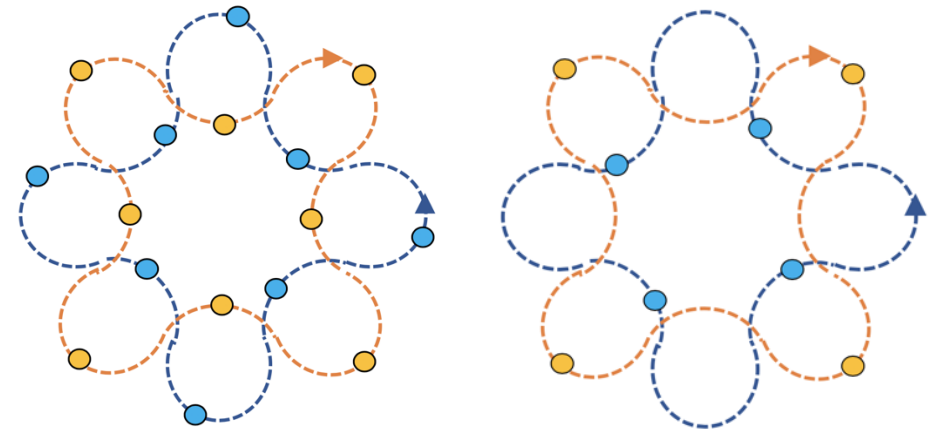
Figure 4: Examples pairs of opposite paths for producing tubular braids.

More complex braids can be constructed by using slightly different strategies. In most cases, several (e.g., two) groups of strands with different paths of movement are used to make the pattern. As an example, the patterns generated by a maypole dance movement of strands, can produce a tubular braid which in braiding terminology is called a tubular biaxial overbraid on a mandrel (i.e., the pole). In the maypole dance movement, two groups of strands move in opposite directions around a mandrel while alternating between the inner and outer circumferences. Similar to the 3-strands braids, the opposite directions and the alternation can be achieved by using oppositely rotating discs. The difference is that, unlike the 3-strands scenario where all the strands pass the whole path consisting of both clockwise and counter clockwise directions, here, every strand passes through one of the two paths which goes either clockwise or counter clockwise. Fig. 4 shows several example pairs of paths for two groups of strands.

The interlacing pattern of a braid is the result of the movement paths of the strands and the distance between the strands within their paths. Fig. 5 shows two different examples of strand configurations in the same pair of paths. The resulting braid in both examples is a tube while in one example (a, c), the braid is consisting of a *regular* pattern and in the other example (b, d), the braid is consisting of a *diamond* pattern. The movement paths for both examples can be realized by using 8 oppositely rotating discs (as in Fig. 4 left). The different patterns are achieved by using different number of strands and their positioning in relation to other strands on the discs along the different paths.

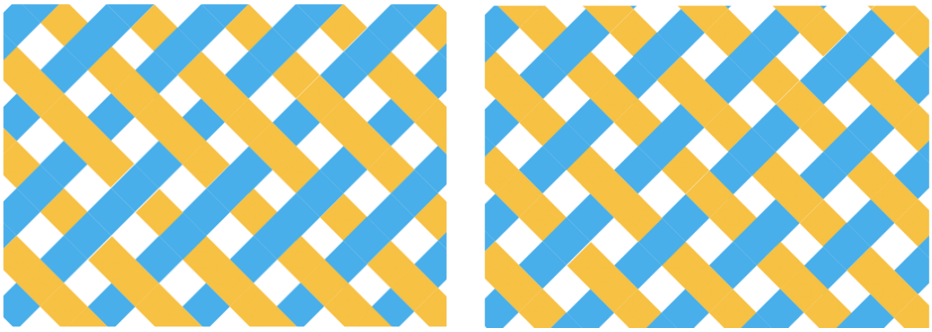
In a configuration with several groups of strands moving in different paths, every strand may or may not interact with the strands of other groups, but it never interacts with the strands of its own group. If we can assure that all the strands within each path are similar in terms of their interactions with the strands of other paths, the prediction or planning of interlacing patterns in a particular configuration of discs will be reduced to only a single strand for each path.

To achieve that, we restrict ourselves to a limited space of setups that meet this requirement by making a number of assumptions and rules for positioning the strands. In all the configurations discussed in the following, all discs rotate with the same speed and every two discs that are next to each other rotate in opposite directions. All the paths of the strands are closed loops realized by the discs. We assume that the strands are planned for producing rather dense patterns where the only relevant under/over interactions occur when two strands are at the same disc. Note that, in a general case and with a very sparse setup of strands, one can achieve interactions between strands at far apart discs, but that is not the case here. We also assume that the interlacing patterns are repeated and identical for all the strands in a path. In the following, first we discuss the procedure of determining the sequence of interactions and thus the interlacing pattern of



a) pair of paths with 16 strands

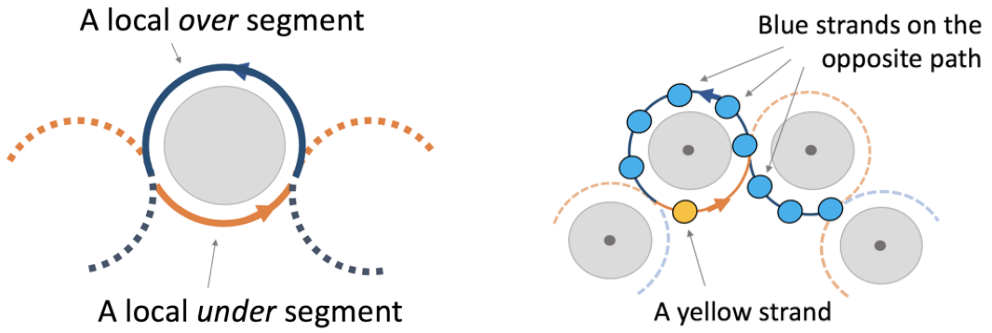
b) pair of paths with 8 strands



c) regular pattern (setup shown in a)

d) diamond pattern (setup shown in b)

Figure 5: Example strand setups (a, b) in a pair of paths to braid a tubular structure. The paths are realizable by 8 discs in a circular configuration. The braid pattern of the tubes are different for the different strand setups (c, d).



(a) (b)

Figure 6: a) An example pair of paths (a yellow and a blue path) crossing and splitting each other in a single disc making a pair of local segments. b) The strands of a blue path that interact with a yellow strand on an *under* segment of the yellow path over time. The yellow circle represents the yellow strand. The blue circles represent the strands that eventually interact with the yellow strand during the period of time its passing the yellow segment. Since the strands on the two paths move with the same speed in opposite directions, the number of the blue strands interacting with the yellow strand correlates with the total length of the yellow segment and its opposite blue segment.

a braid. Then we introduce a set of rules to follow for positioning the strands such that the assumptions for determining the interlacing pattern are realized.

2.2.1. Determining the sequence of interactions

The interlacing pattern can be predicted from the disc configuration and positioning of the strands on different paths. For that, the sequence of under/over interactions for a strand of every path is determined. To determine the interaction sequence in a dense setup, *local segments* are defined. Every pair of paths may cross each other at some points and split each other into several segments. The paths may also overlap with each other. A *local segment* is defined as a segment of a path that extends only within a single disc and is not overlapping into the other path of the pair (see Fig. 6(a) for an example). In every example of Fig. 4, the paths split each other into several segments and all the segments are local (extending only in one disc). For example, in the 8-disc configuration of Fig. 4 (left) each path contains eight local segments.

In a sequence of discs, the local segments can be grouped into *under* segments and *over* segments. For instance, in the examples of Fig. 4, the segments belonging to the outer circumferences can be called *over* segments while the inner segments are called *under* segments. The naming is rather arbitrary because the under and over are relative to the perspective of the observer. However, as long as the naming is used consistently for a sequence of discs, it can be helpful for distinguishing between the contribution of different segments to the braid.

To determine the interlacing pattern, the number of interactions between the strands of different paths are computed for every pair of local segments locating on the same disc. For a particular strand in a segment of a pair, it takes a period of time until it travels the segment. The number of interactions of the strand is the number of strands on the other path passing the opposite segment during this period. Since the strands in the two paths move with the same speed, but in opposite directions, the number of strands passing by the opposite segment during the period correlates with the total length of the two segments of the pair. Fig. 6(b) illustrates

how to count interactions. The number of interactions depends on the total length of the two segments and the density of the strands on the opposite path:

$$I = (O + U) \cdot \delta$$

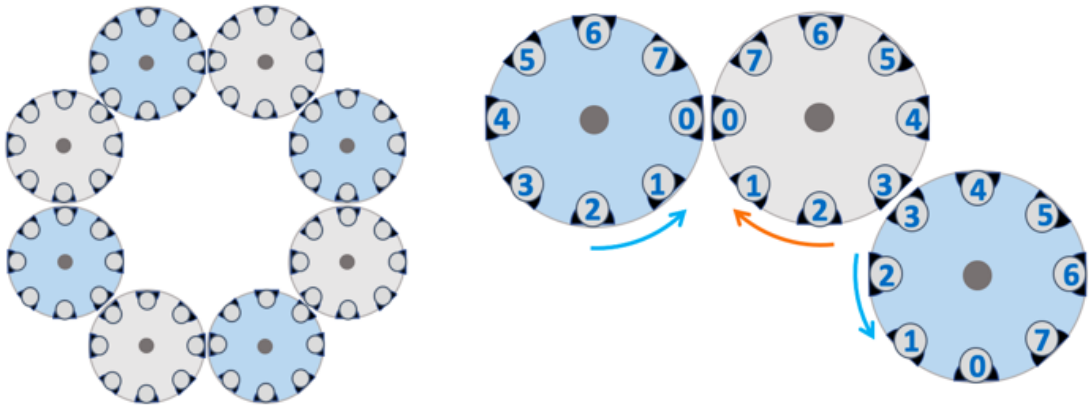
where I is the number of interactions of the strand, O and U are the lengths of the two segments in a pair, and δ is the density of strands on the opposite path. Whether the interactions of the strands in a segment are under or over interactions depends on the type of the segment (*under* segment or *over* segment). The interlacing pattern for the strands of each path is the sequence of interactions computed for all the local segments along the path.

Example – analyzing interlacing pattern in an 8-disc configuration The interlacing pattern of setups in Figs. 5a and b can be determined as follows. Both setups share the circular 8-disc configuration. Such a configuration demands that the relation between the length of the *under* segments to the *over* segments is 3 to 5. For clarity of the presentation, the length of the disc circumferences is represented by $C = 8l$ $(8 = 3 + 5)$ for a given value l. In both setups of the figure, four groups of similar local segments can be detected and therefore, four interaction numbers can be computed: interactions of *under* segments of the clockwise path ($I_{cw,u}$), *over* segments of the clockwise path ($I_{cw,o}$), *under* segments of the counter clockwise ($I_{ccw,u}$), and *over* segments of the counter clockwise path ($I_{ccw,o}$). By computing the interactions of a strand in one segment of each group, we can reconstruct the interlacing pattern of the resulting tubular braid. With $C = 8l$, the distances between the strands of the clockwise and counter clockwise paths for the first setup (Fig. 5a) are correspondingly $A = 4l$ and $B = 4l$ and for the second setup (Fig. 5b) are correspondingly $A = 8l$ and $B = 8l$. In both examples, the lengths of the *under* and *over* segments are $U = 3l$ and $O = 5l$ correspondingly. The interactions are computed as $I = (O + U) \cdot \frac{1}{D}$, where D is the distance between the strands leading to a density of $\frac{1}{D}$. Therefore, in the setup of Fig. 5a, the interactions are $I_{cw,u} = I_{cw,o} = I_{ccw,u} = l_{ccw,o} = 2$, meaning that regardless of the path, every strand passes two times in a row over and then two times in a row under the strands of the opposite path. Since there are 8 strands on both paths, the strands of both paths repeat the pattern twice until they meet the same opposite strand again. In the example of Fig. 5b, there is one interaction for all $I_{cw,u} = I_{cw,o} = I_{ccw,u} = I_{ccw,o} = 1$, meaning that every strand passes once under and once over the strands of the opposite path and the pattern repeats.

With the same logic, we can plan a set of other interlacing patterns. For example, in the same disc configuration as above, one can plan to produce a pattern with the clockwise strands that passes once over and once under while the strands of the counterclockwise path pass twice over and twice under (i.e., $I_{cw,u} = I_{cw,o} = 1$, $I_{ccw,u} = I_{ccw,o} = 2$). The required densities for such interlacing pattern are $\delta_{cw} = 1/4$ and $\delta_{ccw} = 1/8$, meaning that the strands should be distanced with $A = 4l$ and $B = 8l$.

2.2.2. Avoiding collisions between strands

In this section, we introduce a number of rules to ensure that the positioning of strands on their paths produce repeated and identical interlacing patterns as described before, and that the strands of different paths do not collide with each other. The first requirement for a repeated and identical sequence of interactions is a constant density of strands along each path. In other words, the strands need to be equally spaced within their path (**rule_1**). In addition, to keep the interactions identical for all the strands of a path, we need to make sure that every strand interacts with at least one strand of the opposite path during its traversal on every local segment along its path. To achieve that, we set a maximum allowed value for the distance between



(a) 8-disc configuration (b) Numbering of strand holders

Figure 7: Example discs with defined equally spaced positions to hold 8 strands on each disc. The holders are numbered for both the clockwise and counter clockwise rotating discs. The discs are organized such that the holders with the same number match each other

consecutive strands in a path. The maximum allowed distance is the minimum length of all the sums of the neighboring (an under and an over) segments along the path (**rule_2**).

Example – strand setups in an 8-disc configuration. In the 8-disc configurations of Fig. 4 (left), with the disc circumference of $C = 8l$ (as discussed in the previous section), the size of each of the two paths is $T = 32l$. Let's assume that for technical reasons, there are 8 positions on the circumference of every disc that can hold a strand. The distance between the holding positions is l. The discs are arranged such that the holding positions of neighboring discs match each other and thus the strands can exchange between the holding positions of different discs (see Fig. 7 for an illustration). **rule_2** requires that the distance between every two consecutive strands of a path is not more than the minimum of total length of all pairs of neighboring segments along the path. Since the total length of every two neighboring segments along each of the two paths in this example is $8l$ (for $3l$ and $5l$ being the lengths of the *under* and the *over* segments), the strands can be apart not more than $8l$ units ($D \leq 8l$). Considering that the length of each path is $T = 32l$, every path holds at least 4 strands ($N \geq 4$). **rule_1** requires equal spacing of the strands. Thus, to meet **rule_1** (while still fulfilling **rule_2**), each path holds $N \in \{32, 16, 8, 4\}$ strands with the spacing of $D \in \{1l, 2l, 4l, 8l\}$ correspondingly.

Complex configurations with overlapping paths. An example of a more complex disc configuration is shown in Fig. 8. The configuration can be used for braiding a double tube with a shared wall between the tubes. To produce one of the tubes, the circular 8-discs configuration of the previous example can be used. For the second tube, six extra discs can be used along with four discs that are shared with the 8-disc configuration. The four discs contribute to the shared wall between the tubes. Fig. 8b demonstrates four paths that are used for producing the braid. Two of the paths, that is, orange and black, move clockwise and the other two paths, blue and green, move counter clockwise. Fig. 8 shows the combinations of the four paths. The top row shows two pairs of paths that produce the tubes, that is, the pair of blue and orange

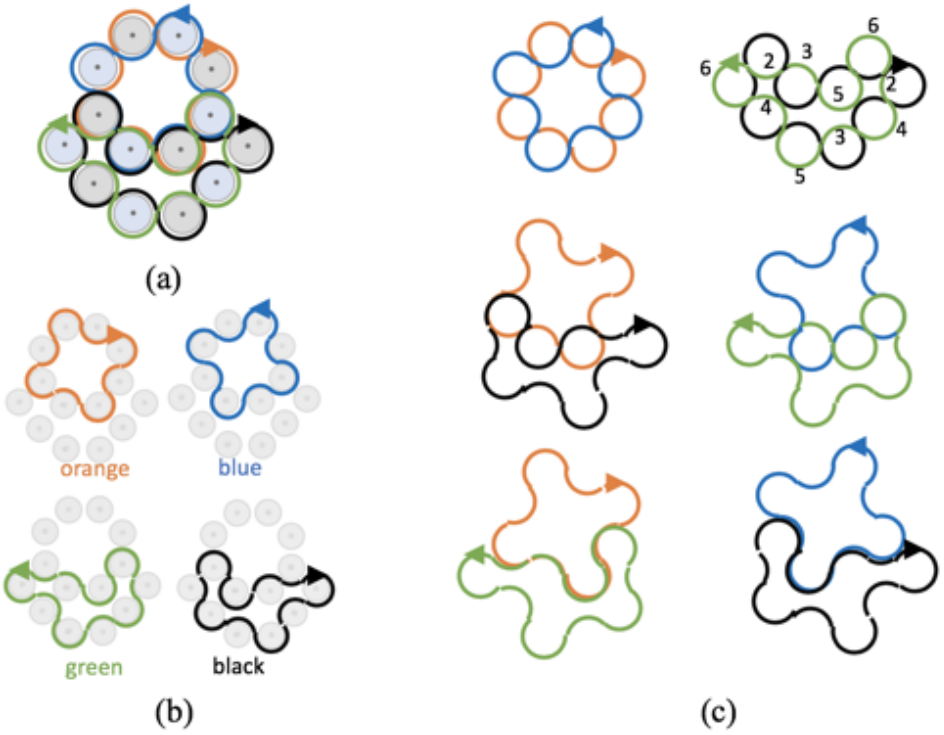


Figure 8: An example configuration for producing a double tube with a shared wall. The disc configuration consisting of 14 discs (a). Four paths that produce the two tubes (b). the pairs of paths for braiding each tube (c, top row), the pairs of paths for braiding the two tubes together (c, middle row), and the pairs of paths that do not contributing to the braiding (c, bottom row).

paths passing 8 discs and the pair of green and black paths passing 10 discs. The middle row shows how parts of the paths involved in producing different tubes cross each other at the four shared discs to braid the tubes together, that is, the orange crosses the black path and the blue crosses the green path. Note that at the crossing part, the paths move in opposite directions. The bottom row shows the paths overlapping with each other. At the overlapping part, the both paths move in the same direction and thus do not braid their strands. The numbers at the top-right combination of paths in Fig. 8b represent the size of the segments for the counter clockwise (green) path. The same sizes apply to the clockwise path.

To set up the strands such that the assumptions for determining the repeated and identical interlacing pattern are realized, we follow the rules stated before. The first rule (**rule_1**) requires an equidistance positioning of the strands on the paths. Thus, the distance for the paths of the 10-disc tube is $D_{10\text{discs}} \in \{1l, 2l, 4l, 5l, 8l, 10l, 20l, 40l\}$. **rule_2** requires that the upper bound for the distance of neighboring strands is the minimum of the sums of all pairs of neighboring segments. From the lengths of segments indicated in Fig. 8b, the rule requires that the distances are at most $D \leq 2l + 3l = 5l$. The combination of the two rules leads to the allowed strand distance of $D_{10\text{discs}} \in \{1l, 2l, 4l, 5l\}$ and strands number of $N_{10\text{discs}} \in \{40, 20, 10, 8\}$. For the paths of the 8-discs tube, the rules require $D_{8\text{discs}} \in \{1l, 2l, 4l\}$ and $N_{8\text{discs}} \in \{32, 16, 8\}$.

The next consideration is the fact that the paths of the two tubes are overlapping as shown in the bottom row of Fig. 8b. It means that the discs at the overlapping parts of the paths need

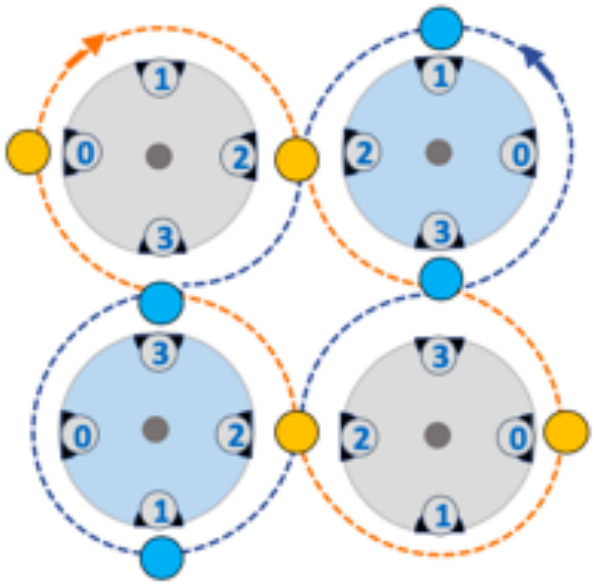


Figure 9: An example setup with two paths each holding four strands in a 4-disc configuration and numbered holding positions. The possible collisions at the crossings is avoided by locating the strands of different paths in holding positions with different numbers

to carry the strands of two paths at the same time but the strands should not collide with each other. Following the previous examples, we assume that there are defined positions for holding the strands on the circumference of the disc with equal distances of l (Fig. 7). With overlapping paths, in order to be able to fit the strands of both paths into the holding positions, the greatest common divisor (gcd) of the distances between the strands need to be more than 1 (**rule_3**). Otherwise, at some point, the strands of the different paths fall into the same holder. To meet this requirement in the example of Fig. 8, the possible distances for the strands on the two paths are combinations of $D_{10\text{discs}} \in \{2l, 4l\}$ and $D_{8\text{discs}} \in \{2l, 4l\}$ leading to $N_{10\text{discs}} \in \{20, 10\}$ and $N_{8\text{discs}} \in \{16, 8\}$.

With the defined holding positions and since all the discs are assumed to rotate with the same speed, another type of collision is also possible. The other collision to be avoided can happen at the crossings of the paths where the strands are exchanged between the discs. To solve the problem, we number the strand holders such that always the matching numbers of the neighboring (oppositely rotating) discs meet (see Fig. 8b). Since the strands are exchanged between discs at their meeting point, every strand always locates at the holders with identical numbers. A simple rule to avoid collisions at the crossings is to only allow the strands of pairs of paths that cross each other (opposite directions) to occupy holders with different numbers (**rule_4**). To be able to make such arrangement of strands on the paths, the same conditions for avoiding collisions due to overlap need to be met, that is, the gcd of the strand distances of the paths are more than 1. Fig. 9 represents an example setup that avoids collisions at the crossings by locating the strands of different paths at the holding positions with different numbers.

Summarizing the set of rules for positioning strands. The set of rules described above ensure that the positioning of strands doesn't violate the assumptions for determining the repeated interlacing patterns of dense setups. The first three rules put constraints on the distances between the strands in their paths. The last rule specifies where the strands can be positioned in terms of the possible locations around the discs. The rules are summarized as follows:

rule_1 The neighboring strands in a path are equally spaced. Such a setup is inline with braiding a repeated pattern.

rule_2 The upper bound for the distance between neighboring strands in a path is the minimum length of the combinations of sums of the neighboring segments along the path. The rule ensures that all the local segments of the same type (under or over) along a path participate in the interactions with the opposite path. It is inline with the assumption of identical number of interaction within the paths in a dense setup. **rule_1** and **rule_2** together ensure that the interactions are repeated and identical for all the strands of each path.

rule_3 For every two paths that overlap or cross each other, the gcd of the distances between the strands of the two paths need to be more than 1. The rule ensures that at least one equidistant positioning of strands within the two paths exists that doesn't lead to a collision of strands from different paths.

rule_4 For every two paths that cross each other (opposite directions), strands of different paths are only allowed in holding positions with different numbers (numbering of holding positions is described before). The rule ensures that the strands do not collide at the crossings along their way.

Self-crossing paths. For completeness of the discussion, we also consider special type of paths that make another possibility of collision of carriers. Such paths are called self-crossing where the path crosses itself at some points. Fig. 10 shows two examples of such paths. Self-crossing paths are not used often because variations of the same structures can be resulted from replacing the self-crossing path by sets of shorter paths that make local loops and cross each other.

A self-crossing path puts an additional restriction to the positioning of the carriers to avoid collisions between the carriers of the same path. In fact, we need to consider the length of the parts of the path that start and end at the same crossing point. These lengths make a set of forbidden values, meaning that if the distance between any pair of carriers (not necessarily consecutive carriers) equals these values, a collision will happen at some point along the process. A solution is to avoid the spacing of carriers to be the divisors of any of the forbidden values. In some cases, this can overly limit the possible setups, even to the extent that there is no allowable setups for the configuration. Another solution that is more practical but not always possible due to the requirements of the braiding patterns, is to use the previously mentioned rules (**rule_1** to **rule_4**) to compute the ideal distances for equal spacing of the carriers. The next step is to make modifications to the spacing by moving the carriers a few positions backward or forward such that the forbidden values do not occur in any place. Depending on the disc configurations, the ideal desired spacing of carriers, and the forbidden values, it can be a straight forward or a complicated or even an impossible task. However, even if the modification is possible, the **rule_1** requiring equidistance spacing of carriers is violated. Note that the **rule_1** was established to guarantee the equal density of strands of opposing paths in relation to each other such that the braiding pattern stays identical everywhere. That equal density can be still met with the slightly out of order positioning, in case the opposing paths are not too dense. This has to be checked by the designer (user). In addition, one needs to make sure that the other rules of positioning, for

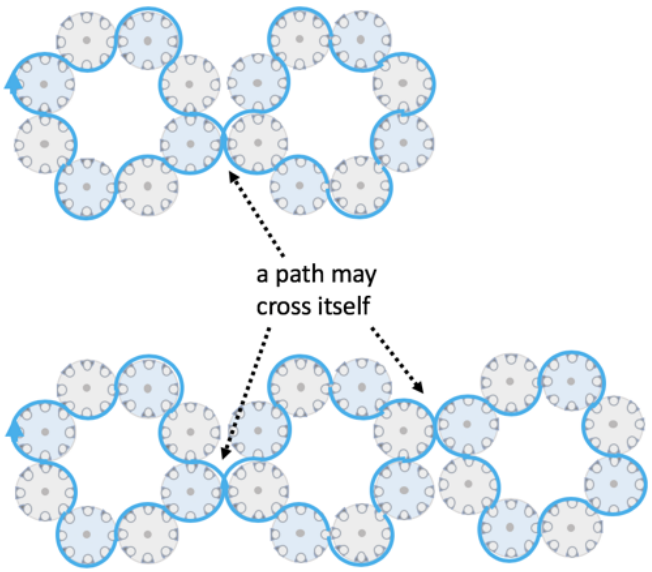


Figure 10: A path may cross itself

example, **rule_4** for using different slots around the discs in different paths, are still met after the modification of spacing.

As an example, consider the configurations of Fig. 10 with two rings of discs (Fig. 10 top) and three rings of discs (Fig. 10 bottom). If the length of the circumference of the discs is 8 units, the distance between any pair of carriers (not necessarily consecutive) should be unequal to 32 in the two-ring configuration and unequal to both 32 and 64 in the three-ring configuration.

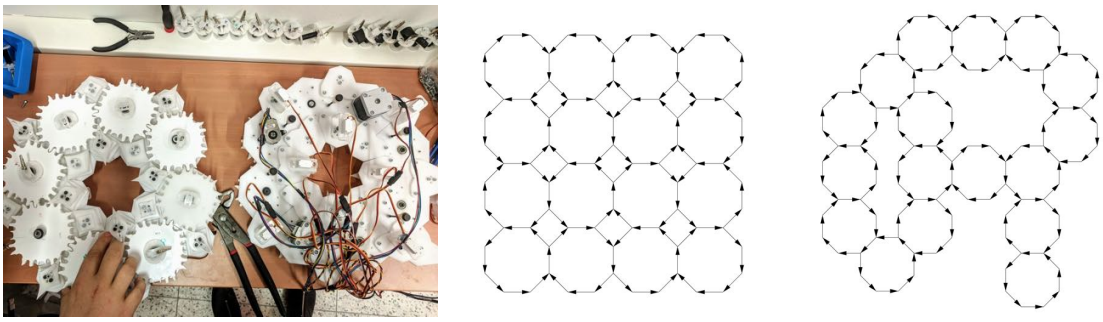
In the two-ring case with the path of length 64, there is no solution that satisfies all the rules and avoids distances that are not divisors of 32. Therefore, the only solution is to use the modification process. We now consider the case of having 8 as the ideal spacing between the carriers. With equally spacing of carriers (i.e., $8, 8, 8, 8, 8, 8, 8, 8$), the sum of every 4th consecutive distances hits the forbidden value of 32. A solution is to make two groups of 4 consecutive carrier distances and modify the last values in each group. The resulted spacing sequence is as follows: $8, 8, 8, 7, 8, 8, 8, 9$.

Another example is shown in Fig. 10 bottom, where the length of the path is 96 and distances of 32 and 64 are forbidden. The only equal spacings that are not the divisors of the forbidden values are $D \in \{3, 48, 96\}$ meaning $N \in \{32, 2, 1\}$ carriers in the path. Assuming that another path exists on the same discs but opposite direction, and following **rule_1** to **rule_4**, the only allowed spacing is $D \in \{3\}$ and $N \in \{32\}$ carriers.

Another solution would be to use the modification of the spacing. Assuming that the desired carrier spacing is 8. To avoid the collision of carriers at the self-crossings, the sequence of carrier spacing can be modified as follows so the forbidden values do not occur: $8, 8, 8, 7, 8, 8, 8, 7, 8, 8, 8, 10$.

2.2.3. Conclusion – theoretical view

Despite the braiding machine's conceptual simplicity, its control is non-trivial as discussed above. From the very start of this work we were challenged by finding collision-free braiding controllers. We have developed the above rules that allow us to develop controllers that are guaranteed not to cause collisions. With this concept, we can verify that a provided configuration and and a controller are effective and do not break the machine.



(a) example braiding machine (b) grid layout (c) complex layout

Figure 11: Example braiding machine configurations: (a) physical examples, (b) grid layout, and (c) complex layout

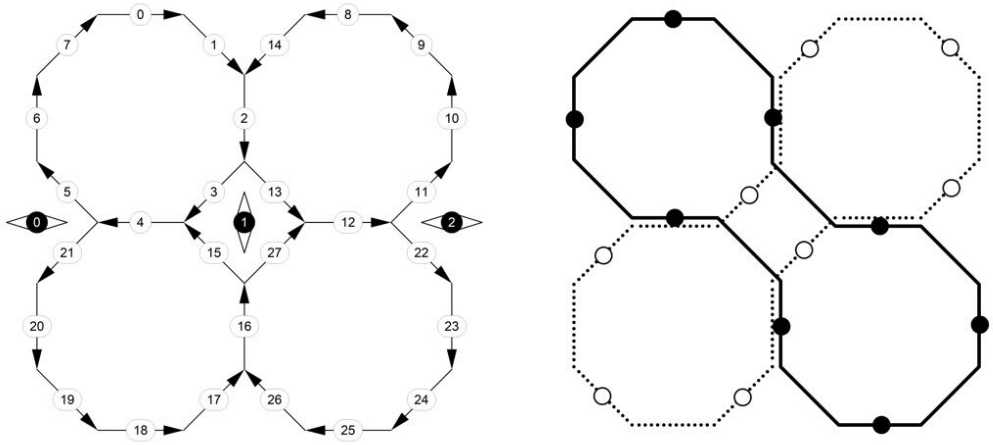
2.3 From high-level model to braiding instructions

The production of a braid using the described braiding machine requires three preparatory steps: 1) arranging the horngears into a viable configuration; 2) placing carriers within the horngears in a configuration that will not result in runtime collisions; and 3) specifying when, during runtime, to switch carrier directions. In combination with the previously reported simulation environment (D1.3, Sec. 2), the described braiding machine can be used in a bottom-up manner to explore braided outcomes. However, to achieve the wider architectural objectives within the flora robotica project, it is crucial to understand how to translate specific braid designs generated using the methods previously reported (D3.1, Sec. 4), into machine fabrication instructions. This section reports on our approach to achieve this goal.

2.3.1. Braid machine constraints revisited

The braid machine is based on a modular principle employing eight-sided modular horngears which can be connected to each other on any edge. The octagonal geometry allows for regular planar tessellated configurations, as well as more free-form planar configurations (Fig. 11).

Each horngear has eight steps (eight possible carrier positions), and connected horngears share one step position (Fig. 12a, white labelled positions: 2, 4, 16, 12). All connected horngears rotate in alternating directions. Between two horngears there is a switch that can alter the path of a carrier (Fig. 12a, indicated with black labels: 0, 1, 2). The switches are independent and are the basis for changing the sectional characteristics of the braid over time. A list of switch states for each switching mechanism (where 0s or 1s indicate carriers would move left or right) can be used as an input for the machine at each time step (where one time step is one eigths of a full rotation of the horngears). However, the list of switch states needs to be carefully sequenced and checked in order to ensure there are no runtime collisions. Therefore, it is crucial to understand the dynamics between machine layout, carrier starting positions and the switching logic over time. The movement of the material carriers (Fig. 12b) through the machine during the simulated fabrication process, can be represented as a sequence of machine positions (Fig. 12a, white labelled positions). The sequences (the position in xy-plane) and the adding of height for each time step (z-axis) make it possible to represent the paths of the filaments as three dimensional curves in space (Fig. 12c). In addition to providing a check on the runtime dynamics, the simulated curves provide the basis for a further braid relaxation simulation (Fig. 12d).



(a) Machine

(b) Carriers

(c) 3D Paths

(d) Braid

Figure 12: Basic layout principle of the octagon-horngear assembly

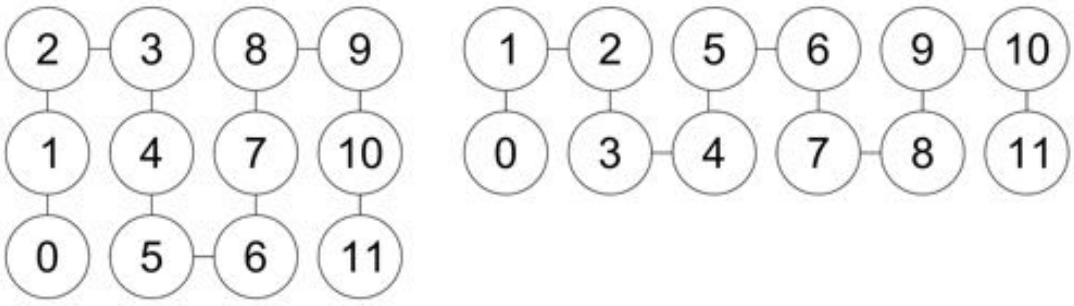


Figure 13: Same horngear topology, different layout geometry

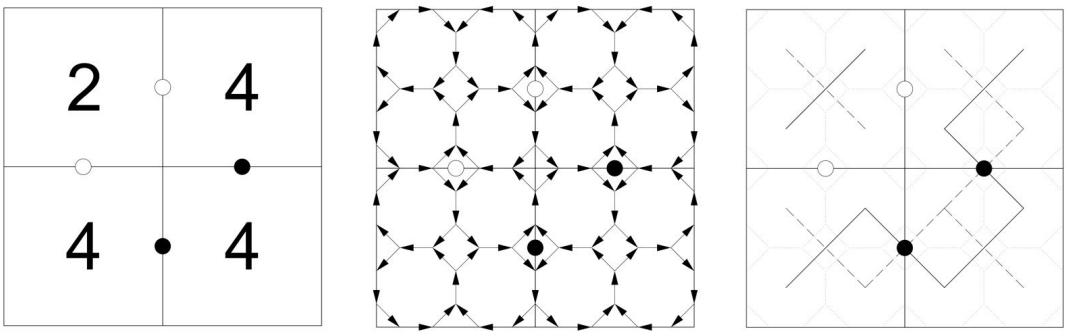
2.3.2. Braids - from matrix

The braid topology can be informed precisely through specifying the interaction (interlacing) of individual horngear-groups (subsequently horngear neighborhoods) with each other for every time step (matrix mode) or more intuitively derived from a 3D graph (line mode). The second method is especially suitable in a spatial design context where quick and iterative design explorations are necessary and a relation between graph geometry and braid topology is desired. For the matrix input method, switch states are informed by a numbering system of 'horngear neighborhoods' (HN) comprising four horngears. Two neighboring HNs with the same number indicate that the adjacent horngears of the two neighborhoods will interact with each other (interlace) at the next time step. The HN numbers are stored in 1-dimensional lists for each time step - with the HNs ordered in the form of a meander. The ordered meander fixes the machine topology, but the same topology can have geometric variants (Fig. 13). Having the HNs numbered in the form of a meander allows changes to the machine layout without changing the topology of the HNs.

The method for determining the interlacing tracks and the carrier positions for a machine layout from the horngear neighborhood connectivity (HNC) is described in the following example for the machine topology and the HNC of Fig. 14:

The white and black circles in Fig. 14a represent if horngear neighborhoods - for this specific time step - interlace with each other or not. Black means they interlace and white means they do not. This is also reflected in the equal or unequal numbering of the neighborhoods. A machine with four horngear neighborhoods has 16 horngears (Fig. 14b). 16 horngears have 128 carrier positions, which, if organised in the matrix shown, share 24 sides. This results in 104 possible carrier positions in the machine. The rotation direction of the horngears alternate throughout the system in a checkerboard-like manner (Fig. 14b, arrows). To get the opposing braiding tracks for the same time step, the pair of horngears rotating in the same direction in each horngear neighborhood are identified to build the basis for the left- and right-track(s) (Fig. 14c, dashed lines for right-track(s), continuous lines for left-track(s) - here the four line-pairs in each horngear neighborhood).

If neighborhoods should interlace with each other the four horngears adjacent to the border of the two neighborhoods (Fig. 14a, lines with black and white dots) have to be part of the braiding-tracks which is represented as another set of dashed and continuous lines, placed on the border of the two neighborhoods (Fig. 14c, here bottom and right neighborhood-border) - rotated by 90 degree to connect the same rotating directions of the horngears as in the neighborhoods.



(a) (b) (c)

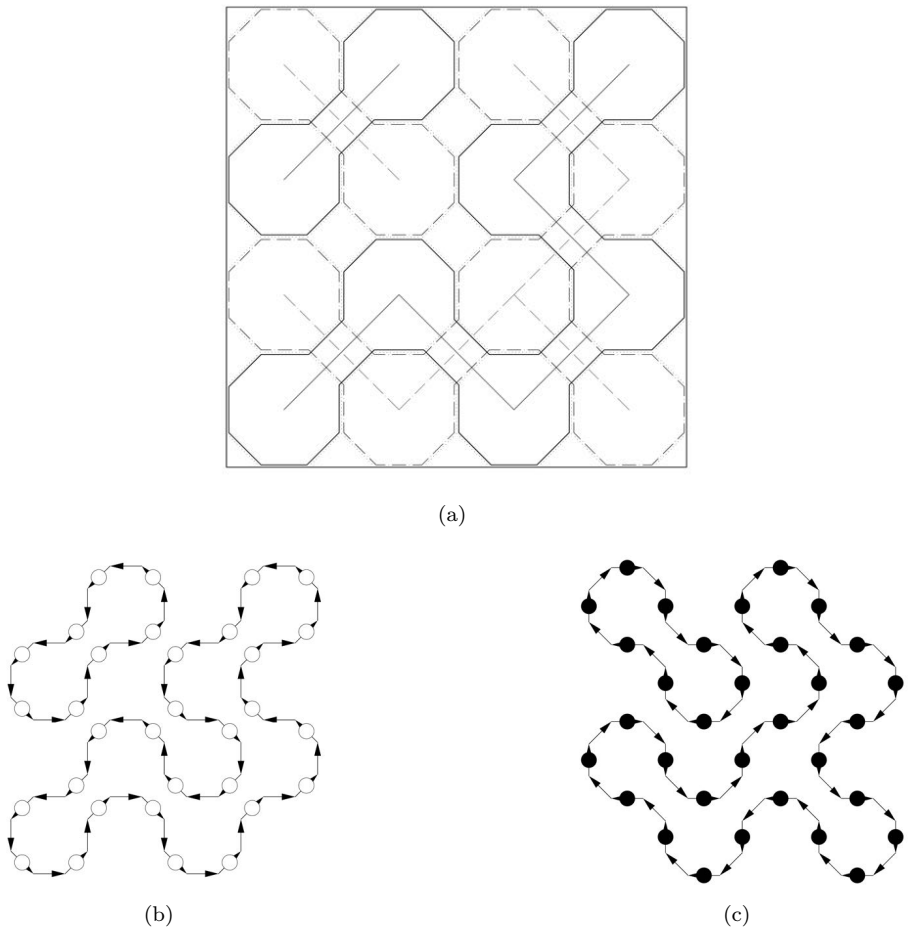
Figure 14: Determining interlacing tracks and carrier positions from HNC: (a) Numbering system horngear neighborhood (HN) and connectivity indication (black circles: connected, white circles: not connected), (b) Horngears and carrier directions, (c) Horngear connection

The result is a zigzag-network of interlacing dashed and continuous lines which reflects which horngears build the basis for the braiding-tracks. The tracks are found by getting all the closest edges of the octagons (horngears) around the individual zigzag lines (Fig. 15a, here four: two dashed, two continuous). The tracks around the continuous lines become left-tracks (Fig. 15) and the tracks around the dashed lines become right-tracks (Fig. 15c). Together all left- and all right-tracks have 64 carrier positions each (128 together). The numeric relationship is as follows: 4 horngear neighborhoods × 4 horngears × 8 positions = 128 carrier positions = 64 track positions for the left- and 64 track positions for the right-tracks. Here the opposing small tracks (Fig. 15b and Fig. 15c, top-left) have 16 positions and the long tracks (Fig. 15b and Fig. 15c, bottom-right) have 48.

Each track gets populated with carriers for half the amount of its track positions. 32 left-carriers (8 + 24) and 32 right carriers (8 + 24). The left carriers are offset by 1 - which makes it possible for the carriers from the left-tracks to pass the carriers from the right-tracks during runtime. Using this logic, every second position in the machine is filled with a carrier and the positions are always the same for all possible tracks. The carriers all initially start in caridinal positions in the horngears, that is, top-, bottom-, left- and right positions of the octagonal modules.

After every second time step all the carriers are in a possible transfer position, making it possible to change to another horngear neighborhood. Every time the carriers are passing the top-, bottom-, left- and right positions of the eight-sided horngears, it is possible to transfer to a new horngear-neighbourhood if there is one connected. This is not always the case as in the matrix configuration there are edge conditions, and in complex configurations there may be limited connectivity. Fig. 16a and b show the carrier movements 1 time step apart. Fig. 16c shows the carrier movements through the machine plotted in time (z-axis). The resulting 3D paths are used as the basis for a braid simulation/relaxation. In the case shown (Fig. 16c) the horngear neighborhood connectivity did not change over time. Fig. 17 shows a path plot that embodies changes of the horngear neighborhood connectivity over time, with the start at the left, a change occurring at the half-way point and the braid finishing at the right.

Fig. 18 shows a changing machine layout while maintaining the same horngear neighborhood connectivity.



(a) (b) (c)

Figure 15: (a) Tracks from horngear connection, (b) left-tracks and carrier positions, and (c) right-tracks and carrier positions.

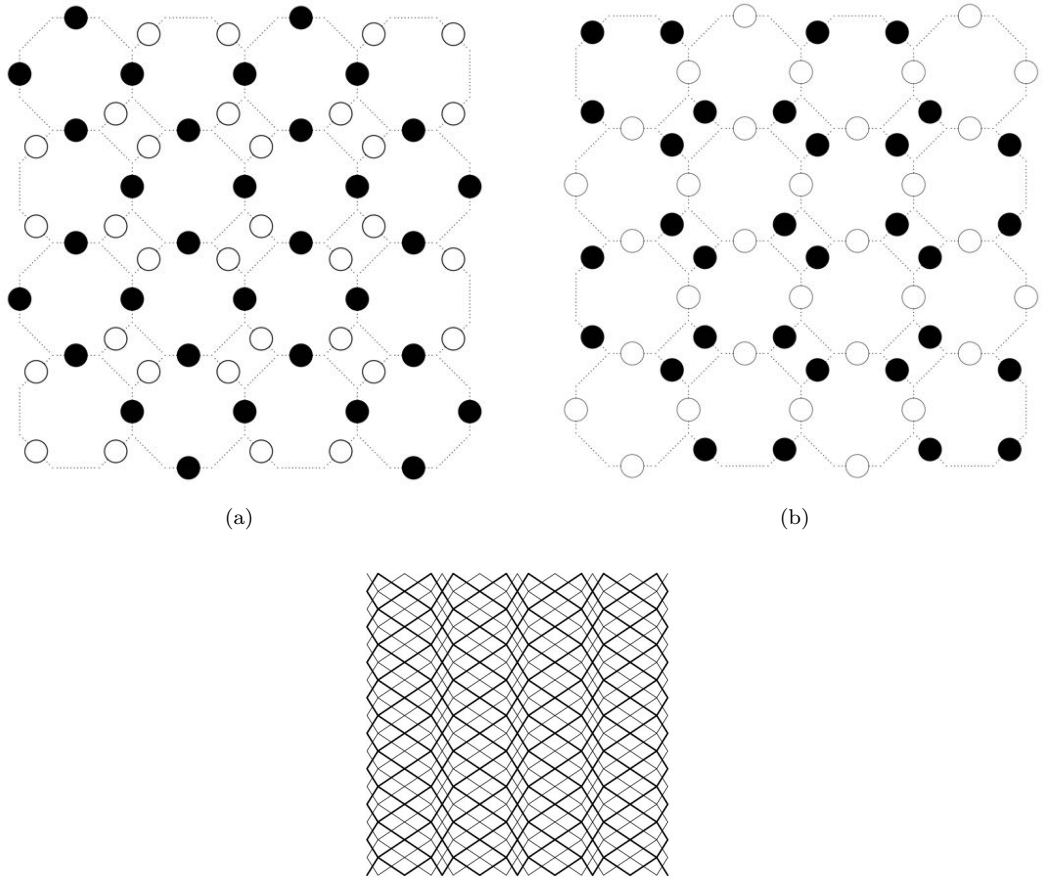


Figure 16: (a) Carrier positions at time step t, (b) carrier positions at time step t+1, and (c) carrier paths.

Figure 17: Path plot with changes of the horngear neighborhood connectivity over time, start at the left, and a change at half-way point.

Figure 18: Same horngear neighborhood connectivity for three geometric machine variants.

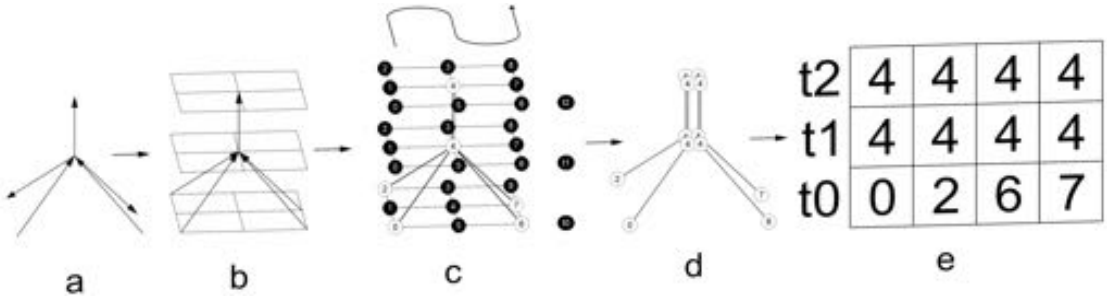


Figure 19: Graph to matrix.

2.3.3. Braids - from graphs

The underlying logic of the braids described using graphs is the same as the matrix-input except for an additional pre-process that translates the graph into a matrix. Graph edges used as input are exploded and aligned in the z-direction (direction of time steps of the braid fabrication simulation) to obtain a directed graph (Fig. 19a). The x- and y-positions of the start- and end-points of all edges in the graph inform a 2D-grid which gets repeated for all z-positions of the points (Fig. 19b). The grid points get numbered and sorted in a meander shape (1-dimensional list), repeated for each z-position (Fig. 19c). The directed graph is then analysed using the NetworkX library for Python to determine 'all simplest paths' from all source nodes to all sink nodes (Fig. 19c and Fig. 19d, white numbers). Nodes from these simplest paths can be written into 1-dimensional lists for each time step t (Fig. 19e). As explained earlier, the resulting matrix can be used as the input for the horngear neighborhood connectivity (Fig. 20).

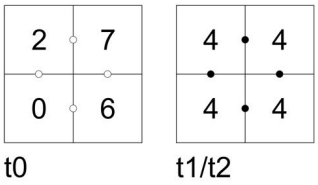


Figure 20: Horngear neighborhood connectivity for the Graph in Fig. 19a

2.3.4. Braid simulation

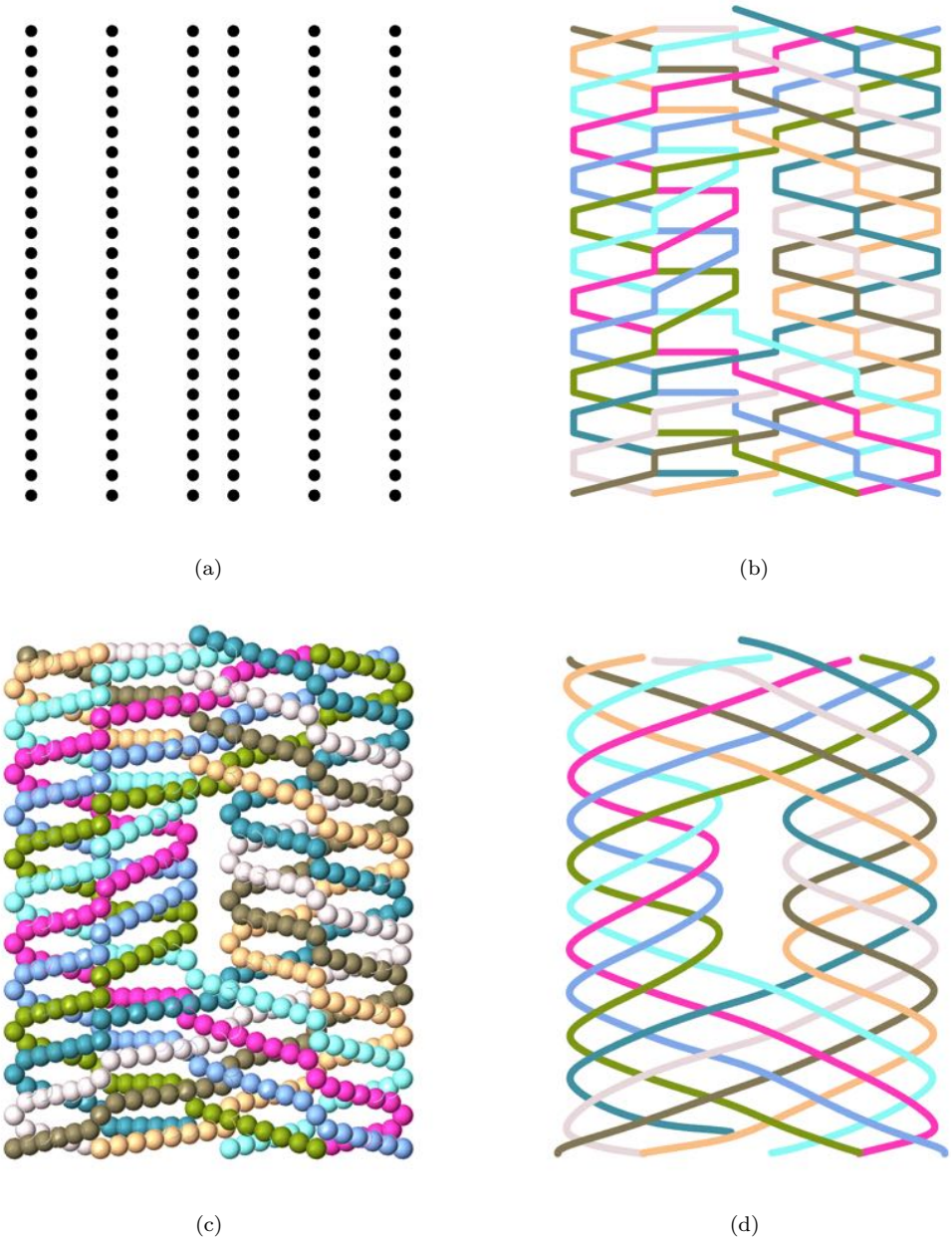
All carrier positions are plotted as points during the simulated fabrication of a braid. For each point, and each time step, the x- and y-coordinates are derived from the position in the machine, and the z-coordinates are incremented (Fig. 21a). The resulting points are connected with individual curves, one for each carrier starting position (Fig. 21b). The curves are divided by equal length, with the length of the curve division being based on the desired thickness of the simulated filament. Each division point is used as the center for a sphere with a diameter of the curve division length (Fig. 21c). The spheres are used as one of the input constraints (sphere collide) for Kangaroo 2, a physics engine for the CAD software Rhinoceros 3D. During the physics-based relaxation, the curves try to become straight lines, while the spheres are not allowed to enter each other. This results in a coarse approximating simulation of individual filament interaction (Fig. 21d).

2.3.5. Conclusion – braiding instructions

We have shown how specific braid machine instructions (machine configuration and runtime switching of carriers) can be derived from high-level braid descriptions. We have also resolved the carrier collision issue by means of having a structured start sequence for the carriers. The implication of using this method is that we are not able to produce all possible generated braid designs using the approach previously reported in D3.1. The benefit of the carrier constraint method is that it greatly simplifies the analytical approach to producing fabrication instruction. As a further iterative development step, these constraints should be incorporated into the braid representation method to guarantee that produced designs can be fabricated.

2.4 Conclusion

This concludes our reporting on the braiding machine. We have improved the robustness of the braiding machine and it is now able to braid with a wide range of materials. We have finally gotten to the bottom of the theory of the braiding machine and understand how to create braiding patterns that do not cause self-collisions in the machine. Finally, we are able to compile high-level 3D models into configurations and controllers that can be run on the braid machine. While there are still practical details that needs to be worked out, the development of the braiding machine has come full circle. From the initial idea in the beginning of the project to the theoretical understanding and automatic generation of controllers in the last period of the project.



(a) (b)

(c) (d)

Figure 21: Time-based carrier positions to braid simulation.

3 Shaping of Braided Structures

Once the braided structure has been produced we are interested in controlling its shape through internal actuation. There are two questions related to this question. The first is what is the mechatronic basis for this shape change and once the basis has been created how do we develop controllers for them. We will address these questions here.

3.1 Distributed actuation

In order to actuate the braided structures we have made an actuator node, which can be mounted on the intersection of two strands. The final version of the actuator node (see Fig. 22) is a sandwich of two semi ellipse PCBs with a combined height of approximately 35 mm, 82 mm width and a weight of 26.2 grams (see Appendix for details on the hardware iterations). The node is controlled by the ESP32-Wroom with build-in Wifi that is used for communication with a host.

The actuator is installed on the structure by sandwiching the node onto an intersection where the braided strands cross (see Fig. 23). The actuator node obtains power from two wires that runs in parallel to one of the strands. An IDC feed-through connector is used, hence, is possible to power several actuator nodes in series.

The actuation is performed by a servo, which has a stall torque of 600 grams at 4.8 V[1] and is able to rotate two strands with respect to each other in a scissors-like movement (see Fig. 24). The actuator node is complete and tested and we are in the process of installing 12 nodes in a braided structures to actuate it.

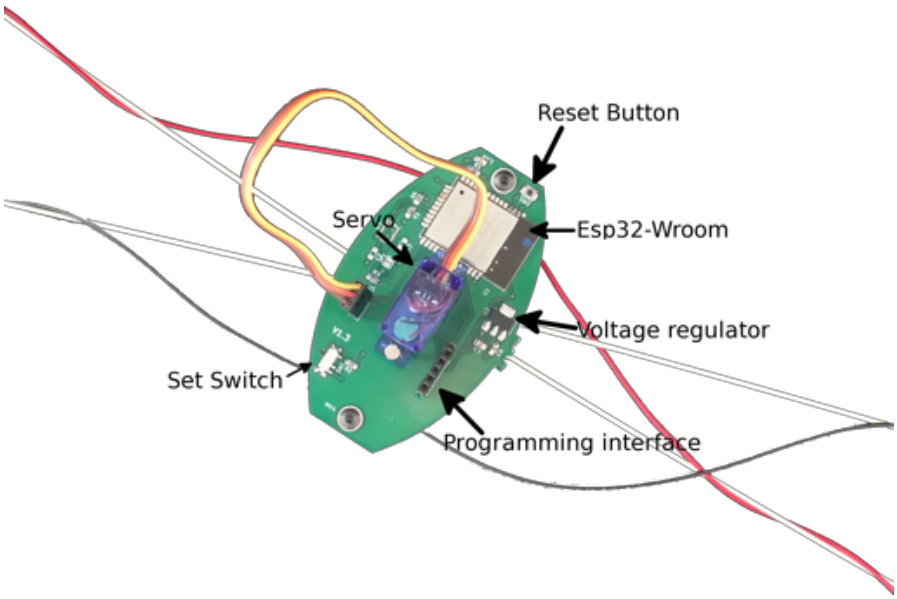


Figure 22: Overview of the final version of the actuator node.

[1] https://www.adafruit.com/product/2201

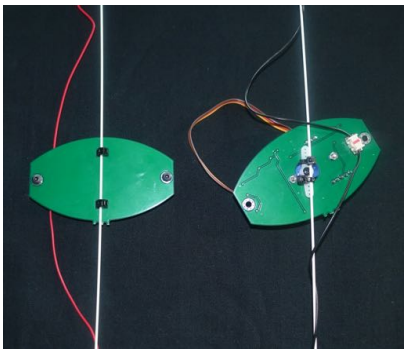


Figure 23: The separated actuator node sandwich is connected to two strands (in this case made from glass-fibre reinforced polymer) and powered through an IDC pass-through connection. The left photo shows one strand connected to one PCB whereas the right shows the other strand connected to the servo motor of the other PCB.

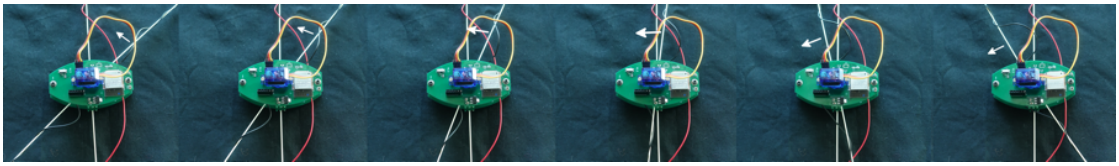


Figure 24: The node is able to rotate two strands with respect to each other.

Name	Functionality
ESP32-Wroom	Micro-controller with wifi
NCP1117ST33T3G	Voltage Regulators 3.3V
SG51R	Servo
IDC Feed through	Power connector

Table 1: A list of key components of the actuator node

3.2 Expansion and contraction with twisted fibers

In the previous section we investigated how rotation can be used to actuate a braid. In this section we consider the idea of having actuation in the form of contraction and expansion: either as separate actuators in parallel to the braided structure or the braided structure itself. We have made artifical muscle fibers from polyamide 6. Polyamide 6 (Nylon 6) contracts when heated. This effect can be increased by inserting twist into the fiber. Eventually the twist insertion will lead to coiling of the fiber. All muscles produced were made from 80 cm of 'raw' polyamide 6 of diameters 0.3 mm, 0.5 mm, and 0.8 mm. In general the produced fibers can be classified into coiled and large-diameter-coil muscles.

Coiled muscle fibers. Next, the fiber is twisted until the fiber is fully coiled. During twisting an insertion load is attached to the fiber (250 g for 0.5 mm fiber). This results in a substantially shorter muscle in comparison to the uncoiled fiber. Coiled fibers achieve lift of high loads by contraction that is limited by the distance between adjacent coils. To twist the fiber, it is fixed to

Figure 25: Coiled muscle

Figure 26: Large diameter coil muscle

a motor (19.6 Watt, more power would be required for diameters $\geq$ 0.8 mm) on one end and to the load on the other end. The load is fixed to prevent it from spinning while vertical movement is possible.

Large diameter coils. Large diameter coils allow greater stroke distance at cost of a reduced lift weight. Twist is inserted until the fiber is about to start coiling. The setup is similar to the coiled muscles, however, to ease the production of non-coiled muscle fibers the fiber was fixed at both ends. To allow the shortening of the fiber during twist-insertion (no-coiling), the fiber was installed with about 12.5 cm overlap. During the twist insertion it was held tight by hand to prevent formation of coiling nuclei.

The twisted fiber is then wrapped around a metal rod and fixated. Then the muscle is

Figure 27: Muscle prepared for annealing

annealed to stabilize its current form. This allows us to produce muscles with greater distances between adjacent coils and a larger coil diameter.

The annealing process is performed in an oven at 149°C, that is the annealing temperature for polyamide 6. Temperature is ramped up by 50°C/h. The annealing temperature is held for about an hour before the cooling ramp down of 25°C/h.

Homo- and Heterochiral muscle fibers. Homochiral muscle fibers are muscles, that have matching twist and wrap direction. These muscles do contract when stimulated. Heterochiral muscles, that is, opposing twist and wrap direction, expand when stimulated.

Heat stimulation. The contraction or expansion of the artificial muscle fibers is achieved by homogeneously wrapping the muscle precursor with NiCr80 heating wire (96.4 Ω/m) to allow direct electronic control with 2.92 to 7.13 Watt, depending on the raw fiber diameter and therefore the length of the used heating wire.

Figure 28: Closeup of heating wire

Findings about coils of large diameters.

- homochiral 0.8 mm muscles achieved a contraction of 54%
- heterochiral 0.8 mm muscles achieve expansion of at least 160% (250% without the unresponsive part of the muscle)
- locomotion of a fibers seems possible, however, there might have been impact from stresses in the electronic cables
- placing muscle fibers in a U-shape allows 'folding' of homo- and heterochiral muscles
- inserting additional twist into the muscle improves the folding movement
- combination of homo- and heterochiral muscles can be combined for movement in one direction.
- muscles also produce circular motion (torsional) forces around the muscle's axis when stimulated
- small diameter muscles react faster than those with higher diameter

In summary, the twisted muscle fibres made from polyamide 6 can be an interesting option for future research on actuated braids.

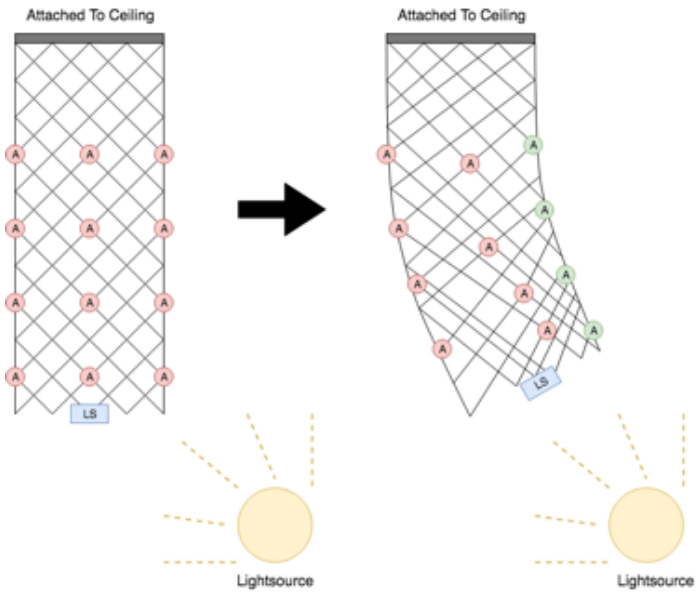


Figure 29: Braid actuation test setup

3.3 Distributed control of braided structures

In the previous sections we developed the technology to actuate and shape a braided structure. In the following we study how to control the movement of a braided structures in a distributed and automated way using machine learning. However, first we introduce the experiment and its motivation. Second, we report our method for applying distributed machine learning control to braided structures.

3.3.1. Motivation and assumptions

We have chosen a distributed approach to actuation because at the practical level we reduce the number of wires in the physical setup, that then allows us to rapidly mount or change the position of actuators on a braided structure.

The primary use case is a situation where distributed actuation of a braided structure can help increase the growth of a plant. We envisioned a situation where a braided structure with 12 actuators and one light sensor hangs from a ceiling with a strong light source nearby (see Fig. 29). The actuators should then control the braided structure to move it as close as possible to the light source, that – if a plant was embedded and growing in the structure – would allow the plant to get closer to the light source as well.

Instead of hand-coding a controller for this specific use case, we developed a machine learning framework that would allow a neural network to learn to control the movement of the actuators. For the specific experiment the controller learned to optimize the light. However, in general it could learn other tasks as long as a suitable feedback is available.

One learning strategy would be to do online learning where the controller is rewarded by the sensed level of light and thus the braids proximity to the light source. However, such an approach is likely to be meet some challenges. Mainly, a large training time of such a system should be expected as online training of a physical artifacts usually takes long compared to training a model in a simulated environment. To address this issue of long training time, we decided to

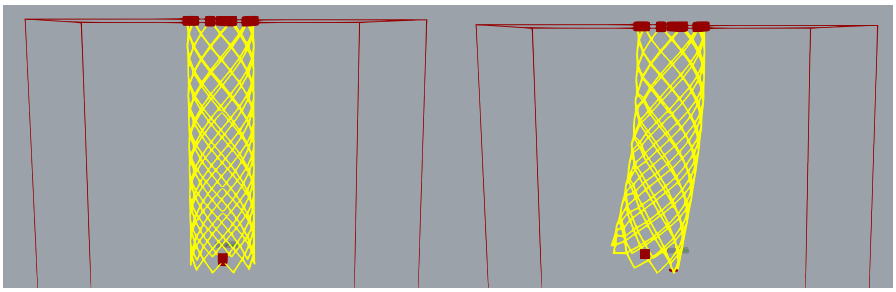


Figure 30: Braid simulation

work towards simulating the structure and actuators and train the controller in this simulator. In a second step we in the future would apply the virtually trained model to the physical braid.

3.3.2. Simulated environment

To train a model in a simulated environment that potentially could be applied to the physical braided structure, we would need a simulation of a braided structure with actuators that were close to the physical setup we envisioned. A simulation of the structure with actuators and light source was created using the software Rhino and Grasshopper. An OpenAI gym environment and a TensorFlow based neural network where created to facilitate the training. A script was then written to link the simulated environment with a neural network, allowing the network to issue commands and retrieve data from the simulation. Screen-shots of the simulated braided structure can be seen in Fig. 30.

We tested the environment with the reinforcement learning algorithm REINFORCE [5], to test the simulation in a learning tasks. We found that training of the model would be infeasibly long because the simulator's computational demands using cross platform communication. As improvement light readings for all possible positions were stored in a lookup table that was then used during trainging. As expected training time was significantly reduced.

As a preliminary test, we train the model with reward for turning on a specific actuator. To develop this behavior, the model was given a reward each time it performed a correct command. The model had two hidden layers of 100 neurons each with ReLU activation. The input is a vector of the position of the actuators and the output is a vector of the target actuator positions in the next control cycle. The training was done over 500 epochs with a batch size of 300 for each epoch, and an environment that reset after five steps. The model managed to find the optimal solution (turning on the indicated actuator and all other acutators off) after around 150 epochs. The results are shown in Fig. 31.

Next we trained the model to search a static light source with a limited command set of five different actuation commands. To develop this behavior, the model received the intensity of the simulated light source as a reward, that is, giving a bigger reward the closer the braided structure was to the simulated light source. We used the same training parameters as above. Here, the optimal solution was found after about 350 epochs, moving the braided structure as close to the simulated light source as possible. The results are shown in Fig. 32.

Based on our experience with these experiments, we decided to go for an approach with eight different actuation commands, since it allowed the model to approach the light source while maintaining a relative low training time. For further work, we will also experiment with a model that has full control of each actuator. As these results show, we are able to train a model to control a braided structure actuation in the simulated environment. However, if we changed

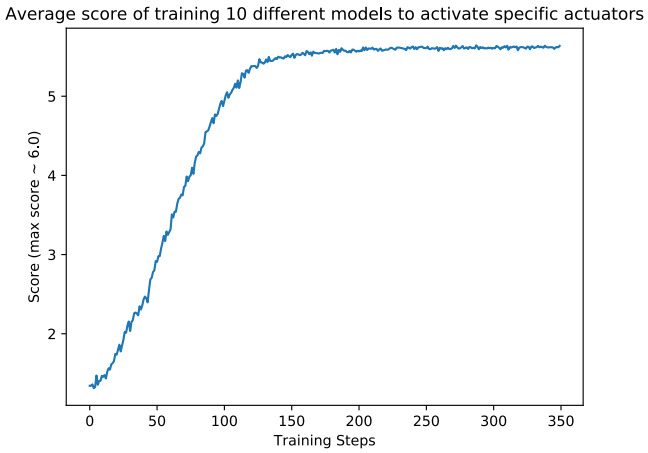


Figure 31: Preliminary tests: average reward over training steps.

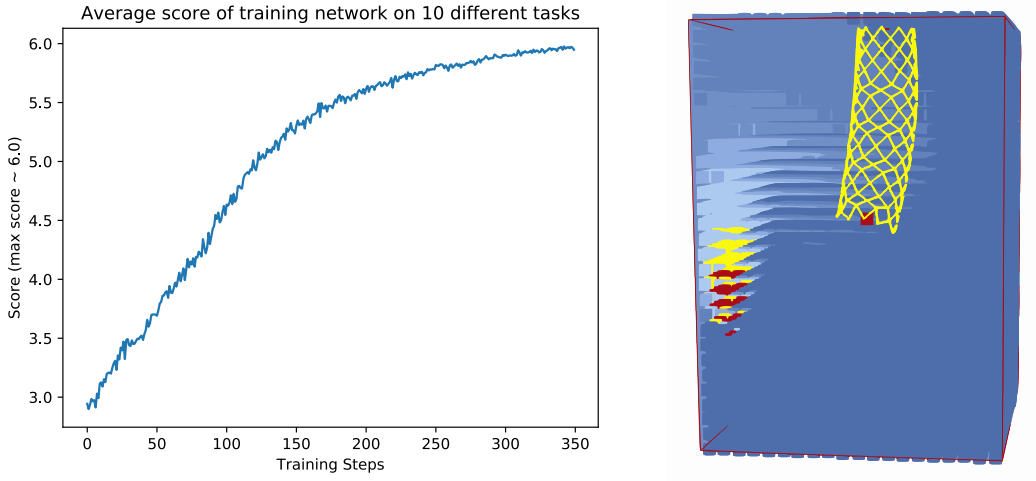


Figure 32: Left: average reward when training a model to search a simulated light source; right: example of a simulated braided structure searching the simulated light source.

the position of either the light source or the braided structure, the model performs similarly to an untrained model. This suggests that the model lacks adaptability, which is important, as a small change in the direction of a real world light source or the real world flexibility of the braid material is to be expected.

A solution could be to utilize an approach that allows for better generalization of learning tasks. One such approach is MAML (Model Agnostic Meta Learning), that is described to help agents perform better in environments where the task can change (such as a light source changing position, or the braid structures resistance changes). Furthermore, we still need to test the approach on a physical setup based on the work described in Sec. 3.1. While this study based on simulations was successful, we expect that complications, such as noisy light data or braid material deterioration, would introduce additionap challenges during the training process.

3.4 Conclusion

In this section we provided an overview of our efforts to actuate and control braided structures once produced. We have developed and tested 12 actuation node that can be mounted on braided structures. We have also investigated an alternative actuation technology based on twisted fibers which is suited for braided structures due to the compact size and easy integration into a braided structure. Finally, we reported a learning-based approach that allows us to control braided structures.

4 Sensing Plants

In multiple earlier deliverables (D1.1, D1.2, D1.3) we reported on the development of the *CYBRES phytosensor System* in order to create a way to interact with plants. In the current deliverable D1.4 in line with D2.4 we would like to present the possible applications of Phytosenors togehter with some experimental data and applications of Phytosensor for bio-hybrid systems. The current work, description of different functionalities, updates on the Phytosensor related firmware, software, additional features reflected in the **MU User Manual**[2].

4.1 Software overview

The software includes four levels, see Fig. 33, the lowest device-level runs on the MU device. It includes the real-time operating system – MU OS – developed by CYB for programmable-systems-on-chip (PSOC), impedance spectrometer, and parts of the firmware for low-level data handling. The MU OS includes different device drivers, the measurement module, data processing unit, and tasks scheduler. The development started in 2013 and is currently used on different CYB devices. The client-level software runs as the MU client program and performs all main tasks related to device and file management, handling time and excitation. The client operates with Gnuplot and DA (detector-actuator) scripts. Gnuplot scripts represent the third software level and is responsible for graphical utilities, 3D/4D plot and regression functionality. Finally, DA scripts handle statistical data processing, sensor-fusion functionality, perform actuator control and multi-device management. Gnuplot and DA scripts are open for users and can be customized for particular purposes.

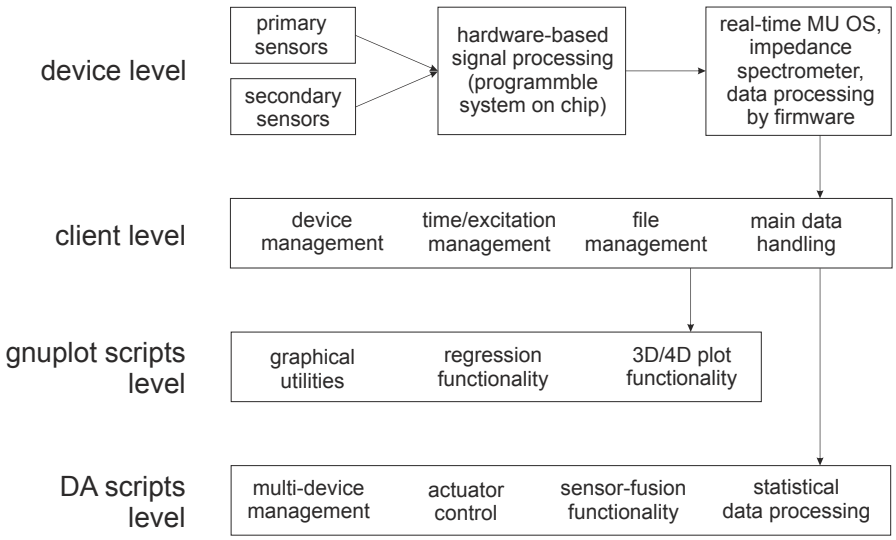


Figure 33: Software structure of the MU system.

The user interface is shown in Fig. 34. This software system includes a plotting engine, the USB interface part, the MU OS terminal, and different modules (in particular for phytosensing purposes). The client program has six sections: 'control' (system control), 'impedance' (setting

[2]CYBRES MU3 Phytosensor System User Manual
http://www.cybertronica.de.com/download/MU-EIS_Manual_en.pdf

for the EIS measurements), 'plot' (setting for the graphical output), 'system' (system setup), 'calibration' (calibration settings), and 'output' (the output window of the operating system and some interactive commands). The client part is fully compatible with MU OS and is also used in different CYBRES projects and devices.

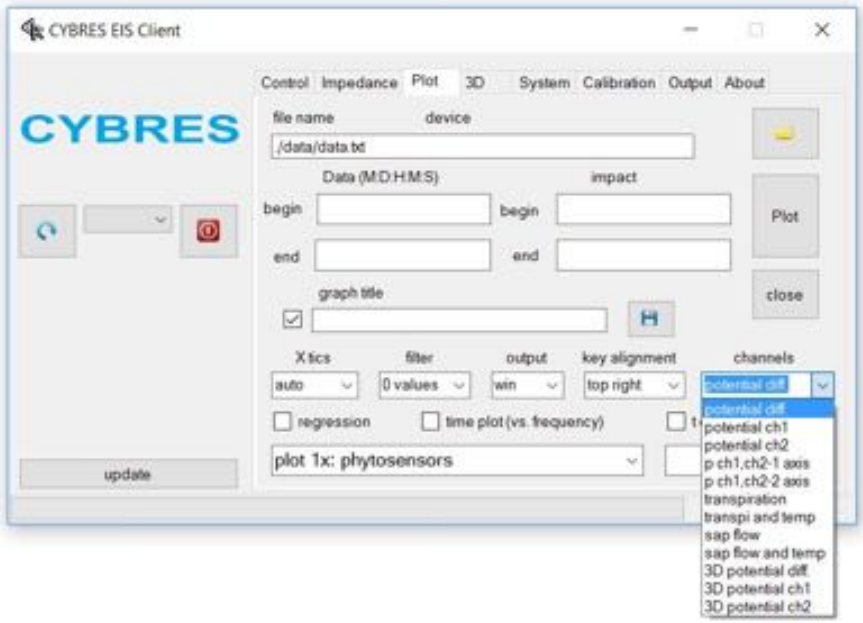


Figure 34: Client software with the option 'phytosensors' chosen and the list of available plots.

For phytosensing purposes a specific module was developed in the client program. It is integrated in the section 'plot' and maintains over ten different subsystems for data processing (also regression analysis for online data in the 'time mode'). Currently, the work is concentrated on developing algorithms and strategies for autonomous data processing, in particular electrophysiology.

4.2 Outdoor setup with the phytosensor

The phytosensor was further developed for use not only indoor, but also in outdoor environments. The connectors were correspondingly modified and water-proof packaging was used. There are two versions of the setup: with wired solution based on PoE (Power over Ethernet), see Fig. 35, and one based on solar cells and WiFi communication, see Fig. 36. Both solutions have their own applications areas: wired solution with Ethernet is intended for long-term installations in the area of 50 m^2, whereas wireless solution is mostly for short term (several days) experiments. Outdoor setup enables monitoring trees, bushes and similar biological objects; with multiple phytosensors and an urban or forest ecosystems can be monitored. Example of measurement is shown in Fig. 37.

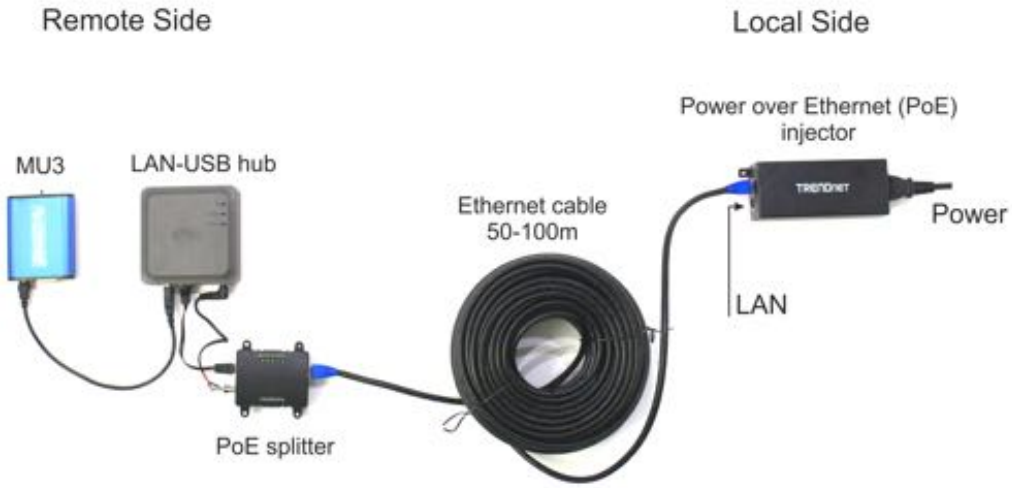


Figure 35: Outdoor setup with wired (Power over Ethernet – PoE) solution, powering and data communication via PoE.

4.3 Phytoactuation: robot arm setup

The list of available actuators for MU3 includes the following groups:

1. **wavOut/mpegOut device:** play sound .wav/.mp3 file from position 'x' to position 'y';
2. **sound device:** change volume, change right/left stereo balance;
3. **MIDI device:** generate musical MIDI tones;
4. **text-to-speech (TTS) device:** generate the voice message;
5. **logical/probabilistic device:** to compute different logical/probabilistic operations and expressions from detectors;
6. **adaptation mechanisms:** several instruments and methods to implement adaption in probabilistic networks;
7. **RGB LED device:** turn on/off R, G, B components of LED connected to the MU system;
8. **external physical devices connected to the MU system:** for example, turn on/off lamps/pumps connected to the MU system (by sensing the ASCI commands in COM-port);
9. **electrical stimulation device:** generate the electrical stimulation by the MU impedance measurement system;
10. **external physical devices connected for example to USB port:** generic control of external devices;
11. **'intelligent house' devices and systems:** on/off and parametric control of these devices;
12. **send message to file:** write text message to file;
13. **send message to IP port:** send text message to specific IP port in internet/intranet;
14. **send message to twitter account:** send text message to specific twitter account.

Figure 36: Outdoor setup with solar cells and WiFi data transfer

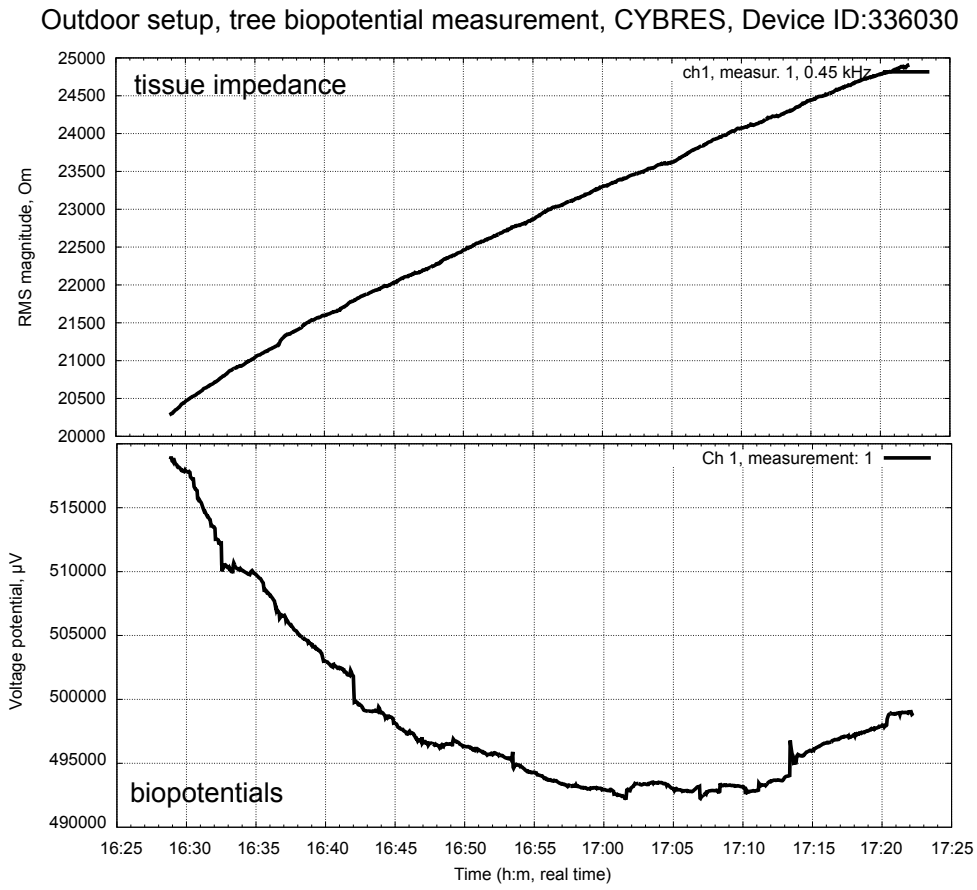


Figure 37: Example of measurements with outdoor setup on a tree.

Below are several example of technical tests we have done using the MU3:

USB relay with FTDI driver (e.g. type SainSmart 4/8 channel USB Relay Board): the board is controlled by 8-bit binary number, where each bit represents the relay state (0 = off, 1 = on), for example, '10001001' would turn on relays 8, 4, and 1, all others would turn off. The value 137 ('10001001') should be send via actuator A21-A40, for example, 'A21=COM6 9600 %H15' will turn on the relay 1, 2, 3, 4, 'A21=COM6 9600 %H0' will turn off all relay (COM6 is an example, it can be different in another system). Note that the control software for the USB Relay Board should be executed before starting the client.

Energenie EG-PMS2 programmable 6-Socket Power Outlet Strip (USB version): this device is controlled by own software 'pm.exe', use actuators A191-A200 for control. Note that 'pm.exe' should be executed before starting the client.

Pololu Mini Maestro 6/12/18/24-Channel USB Servo Controller enables USB based control over PWM motor drives, use it with A21-A40 actuators and '%Hx' marks for controlling the motors (see user manual of the Pololu devices).

using MU actuators with boards MU3.0, MU3.1, ...

In particular we tested Pololu Mini Maestro 12-Channel USB Servo Controller connected to a 6DoF robot arm, see Fig. 38. For controlling the behaviour of the robot the event-driven Petri nets are used, see Sec. 4.3.1.

4.3.1. Phytoactuation: event-driven Petri nets

The DA module provides the multi-token Petri-nets-like mechanism for implementing event-driven reactions. In fact, the implemented detector-actuator coupling, shown in Fig. 39(a), can be represented in the Petri net form, shown in Fig. 39(b), taking into account specific event-driven character of the DA module.

The Petri places are represented by Dx and Ax states, the Petri transitions are implicitly defined for all actuators and detectors, and Petri arcs are implemented in the condition mechanism. The important component of Petri nets is the token system that describes a concurrent behaviour of the network and the activated places. Since the concurrent behaviour of $Dx \to Ax$ is defined by detectors Dx, tokens are primarily used here for activation of states.

In the case of the DA module, tokens are z-variables of A171-A180 (and corresponding A181-A190) – 10 different tokens – that can cake take any integer values. The actuators A211-A220 analyze the value of corresponding z (each of A211-A220 corresponds to A171-A180) and can call an actuator. Following the concept of executing $Dx \to Ax$ at one step, all A211-A220 belong the 'replicator'-type of actuators, i.e. all related activities are executed at the same step.

For example, consider the shown in Fig. 39(a) fragment of Petri net. It defines four places and transitions, which however can be understood in two different ways. The transition can happen due to activation by different detectors (without considering internal tokens), or the transition is triggered by the same detector but the selection of activities is controlled by the token, Fig. 39(b). The first case represents in fact a reactive behaviour and can be covered by a normal mapping $Dx \to Ax$. More interesting case appears when the same detector should activate different actuators based on tokens. The fragment shown in Fig. 39(b) can be transformed in the form, suitable for the DA module, as shown in Fig. 40.

(a)

(b)

Figure 38: 6DoF robot arm with Pololu Mini Maestro 12-Channel USB Servo Controller connected to the phytosensor system. (a) Used setup; (b) Example of scenario with robot arm.

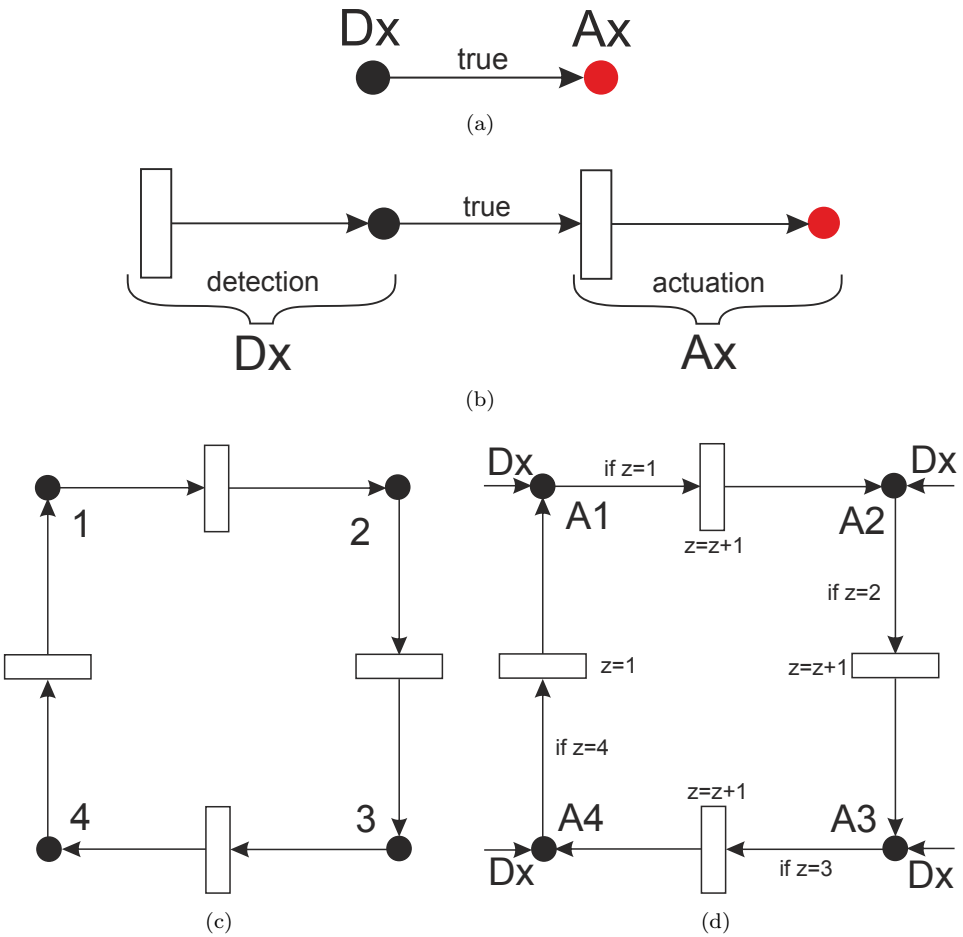


Figure 39: Petri net form of detector-actuator coupling: (a) the implemented detector-actuator coupling; (b) the corresponding Petri net; (c) fragment of Petri net with four places and transitions; (d) modified fragment taking into account activation by detectors and tokens.

The executable code is shown below:

```
-- define detector and connect to replicator
I1=4;
P1=1;
D1=160;     call replicator A160

-- define replicators
A160=171 211;  call A171 and A211
A161=181 4;    call A181 and A4

-- define token system
A171=1 0;   increase z0 by 1
A181=0 0;   set z0 to 0
A211=1 1 2 2 3 3 4 161;
```

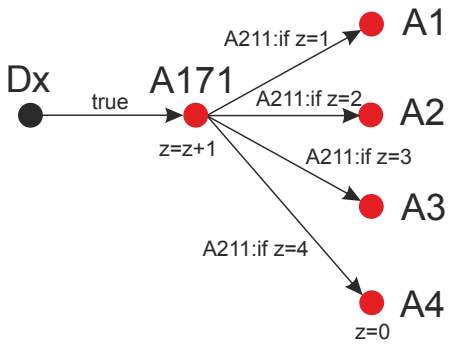


Figure 40: The fragment from Fig. 39(b) transformed in the form, suitable for the DA module.

Note that by using A171=0 x; or A181=0 x; (for all A171-A190) the values of z0...z9 can be set on the value x. Taking into account multiple tokens and detectors, the Petri-nets-like behaviour provides rich possibilities for actuation.

Probabilistic transition in Petri nets. There are two types of transitions in the fragment shown above: 1) single transition like the detector – replicator $D1 \rightarrow A160$; 2) multiple transitions like the replicator $A160 \rightarrow A171$ and $A160 \rightarrow A211$, the token system $A211$. Following the rule for probabilistic system, only the single transitions can be assigned with probabilistic value by 'B' keys. Thus, $D1 \rightarrow A160$ can be used as probabilistic transition.

4.4 Production of phytosensors for tests, evaluations, certification and demonstrations

For tests, evaluations, certification and demonstrations CYB produced about 25 phytosensors, see Fig. 41 and several replacement elements. These devices are distributed among partners in the consortium, used in exhibitions and demonstrations.

Figure 41: Produced phytosensors for tests, evaluations, certification, and demonstrations

4.5 Hardware exploitation: EMC tests and certification

As a part of exploitation, CYB performed development and tests towards EMC (electromagnetic compatibility) of the phytosensor system in order to comply with EMC requirements for Information Technology Equipment (ITE), namely standards CISPR 24 (Information technology

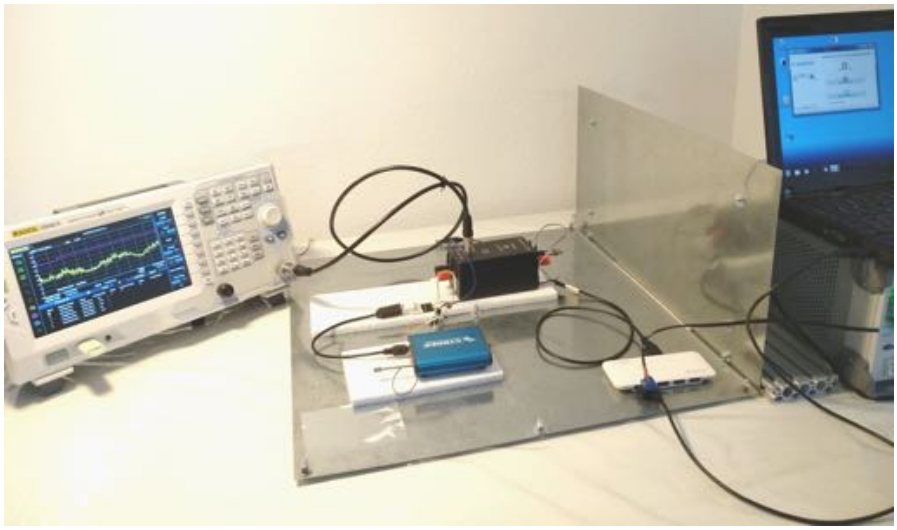
equipment - Immunity characteristics - Limits and methods of measurement) and CISPR 32 or 22 (Electromagnetic compatibility of multimedia equipment - Emission requirements). The pre-compliance tests and debugging of issues are performed locally at CYB facilities, see Fig. 42, and tests performed externally by involving the certified laboratories of SGS Germany GmbH (Munich), see Figs. 43 and 44. The phytosensor successfully passed tests specified by EN 61326-1:2013 (IEC 61326-1:2012) and FCC 47 CFR Part 15 §15.107, §15.109 (ICES -003 Issue 6) as well as conditions of the RoSH cetrification.

4.6 Operating phytosensor: interactions with users via color indication

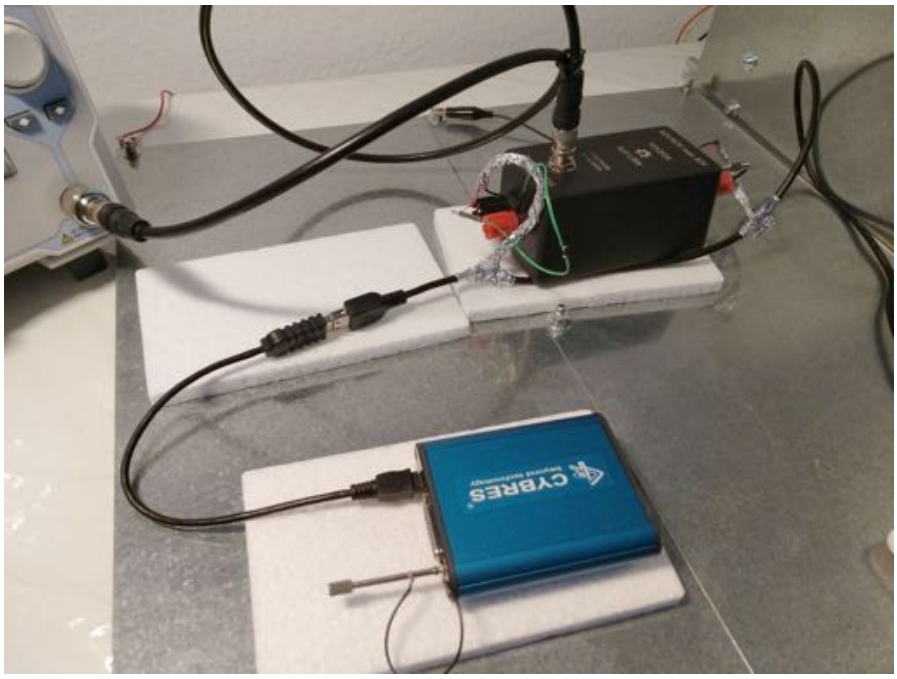
Interactions with users is possible via different means: light, sound, mechanical actuators, text/voice messages, and others. Light belongs to most important ways because of multitude of signals coded by colors, pulses and intensities. Low-intensity light does not disturb users and thus is preferable as means of interaction. We tested different possibilities for color indication: from DMX lighter up to light bases. Several of there solutions are shown in Figs. 45 and 46.

4.7 Conclusion

We have given an overview of the MU3 unit that has been developed throughout the *flora robotica* project and further use of the unit can be found in D2.4. The unit has extensive capabilities in terms of sensing and controlling external actuators based on this sensor input. Furthermore, it has a rich feature set in terms of data recording, analysis, and visualization. The MU3 unit has passed certification and is ready for commercialization.



(a)



(b)

Figure 42: Tests on electromagnetic compatibility of the phytosensor system in the local CYB facility.

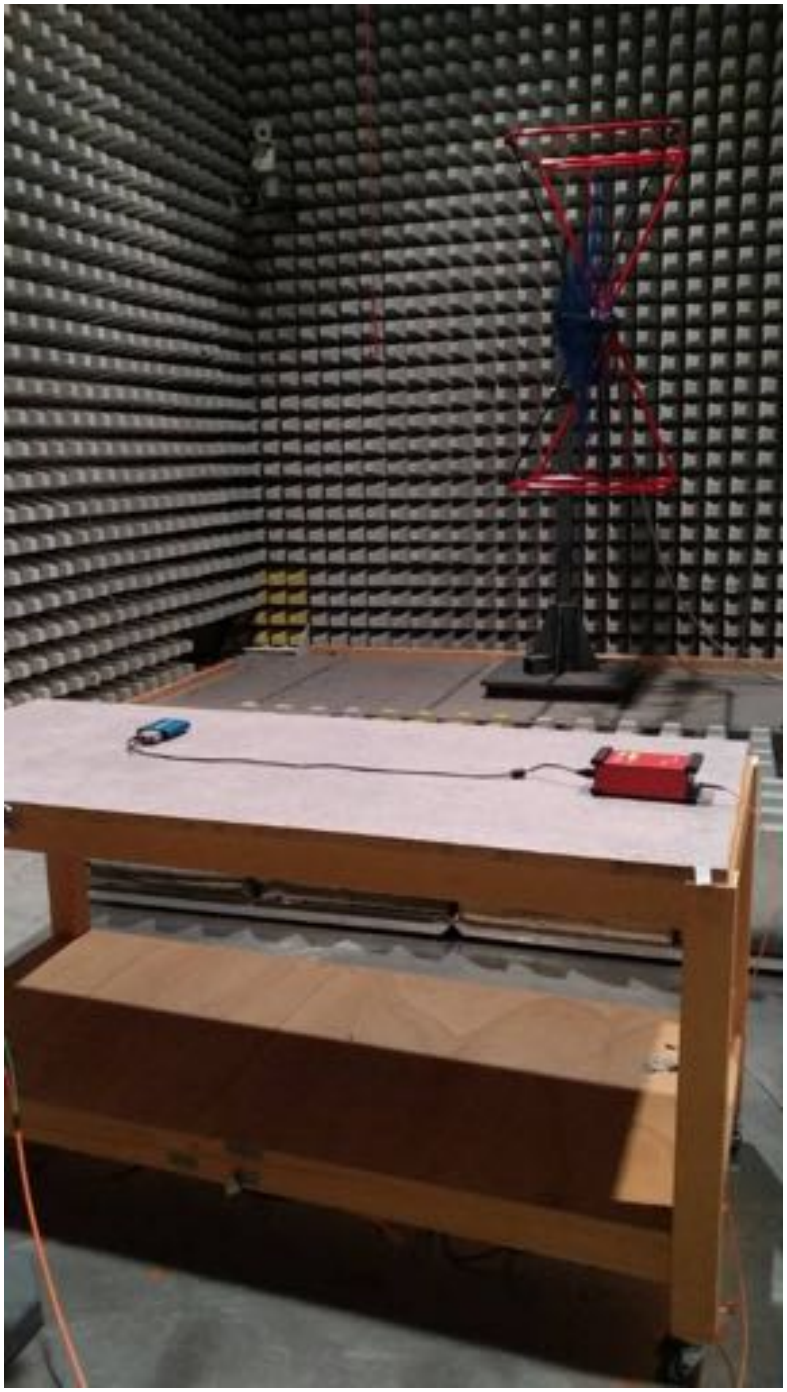


Figure 43: Tests on electromagnetic compatibility of the phytosensor system in the SGS Germany GmbH (Munich) certified laboratory: test on EM immunity.

Figure 44: Tests on electromagnetic compatibility of the phytosensor system in the SGS Germany GmbH (Munich) certified laboratory: test on electrostatic discharge immunity.

Figure 45: Interactions with users via color indication: low-intensity RGB light ball.

(a) (b)

Figure 46: Interactions with users via color indication: (a) low-intensity RGB light base 6 inch; (b) middle-intensity RGB base 8 inch.

5 Plant Shaping

In the previous deliverable *D1.2 Evaluation of mechatronics prototype of the robotic symbiont including supporting software*, we have reported on the first generation of decentralized hardware for the delivery of light stimuli to shape plants. Following rounds of experimentation and hardware development, we have established a more generalized protocol for this type of plant-robot experiment, which we are in the process of making publicly available as a resource. Here we report on the hardware section of our generalized open-source protocol. The other sections of our protocol are reported in the companion deliverable *D2.4 Report on the final algorithms and plant-affection of bio-hybrid organism*.

Step 1. Organize robot capabilities into decentralized nodes with single-board computers, integrated into modular mechanical supports. Ensure each identical robot node is able to control and execute its own behavior.

Step 2: Robotic provision of stimuli to plants. Provide blue light (400–500 nm) to plants at controllable intervals, at an intensity that will trigger their phototropic response, from the direction and orientation required for the respective portion of the experiment.

a) Select a red-green-blue (RGB) light-emitting diode (LED) or an isolated blue LED. In either case, the LED should include a blue diode with peak emission $\lambda_{\max} = 465$ nm.

b) Select an LED that when congregated in groups and set in the precise conditions of the utilized robot, can maintain the required light intensity level in each direction tested in the experiment setup. For each direction being tested, the blue diodes in the LEDs in a single robot should collectively be capable of maintaining a light intensity level of approximately 30 lumens without overheating, when situated in the utilized robot enclosure and any utilized heat dissipation strategies. (For example, in a robot utilizing three LEDs per direction, with microcontroller-enabled regulation of intensity, if the blue diodes emit with maximum light intensity $\Phi = 15$ lumens, then without overheating they should be able to maintain 65% of the maximum.) The selected LED should have a viewing angle of approximately 120°.

c) Interface the LEDs to the robot's single-board computer (either directly or indirectly via a microcontroller or other interfacing) such that individual control is enabled, either of each LED or at least of the LED groups serving each direction being tested in the setup.

Step 3: Sensing procedure for the proximity of plant growing tips. Use processed readings from infrared proximity (IR-proximity) sensors to reliably and autonomously detect the presence of plants approaching from each direction tested in the setup.

a) Select an IR-proximity sensor that regularly detects the growing tip of the selected plant species, when arranged perpendicularly to the central axis of the direction from which the plant approaches (when tested in an unobstructed environment). Successful detection should occur starting from a distance of 5 cm, as seen in Figure 47 starting at the timestamp labelled '07.04.16' on the horizontal axis.

b) Interface each IR-proximity sensor to the robot's single-board computer, and implement a weighted arithmetic mean approach to process the sensor readings into a determination of whether a plant is present within 5 cm. The sensor readings from the most recent 5 s should give 20% of the final average weight used in detection.

c) The selected IR-proximity sensor should not emit critical wavelengths that could interfere with the light-driven behaviors of the selected species, for instance, the shade-avoidance response. If the sensor partially emits at wavelengths below 800 nm, such wavelengths should not be present at distances greater than 5 mm from the sensor's IR source, as measured by spectrometer.

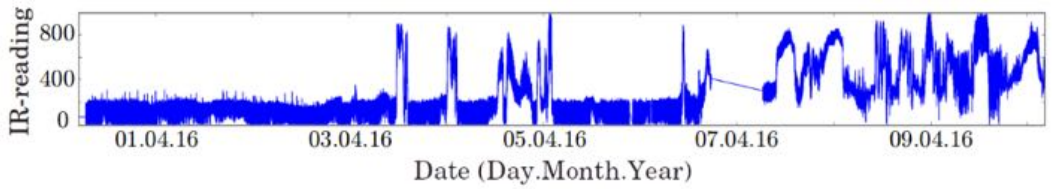


Figure 47: Sample IR-proximity sensor scaled voltage readings (vertical axis) during an experiment. Higher values indicate plant tip detection. In the case shown here, shortly after '03.04.16,' a plant tip climbs a support and arrives in the field of view of the robot.

Step 4: Overall hardware. Distribute the experiment functions over the set of robots, such that each robot can autonomously manage the portions that proceed in its own local area. Arrange the robots' provision of light stimuli and sensing capabilities according to the respective plant growth directions being tested.

a) Compose each robot around a single-board computer that interfaces with the sensors and actuators. The computer should be connected to a Wireless Local Area Network (WLAN) module. If using an off-the-shelf single-board computer (e.g., a Raspberry Pi), the interfacing to sensors and actuators can for instance be accomplished via a custom printed circuit board (PCB) attached to the computer's header. Each robot should be individually powered, with its own battery backup if possible.

b) Include one IR-proximity sensor per direction being tested for approaching plants, or in a continuous growth field enough sensors for their view angle to cover the possible growth area, according to the above requirements.

c) Include enough LEDs to deliver the above blue light requirements, per direction being tested for approaching plants. If using RGB LEDs rather than blue LEDs, optionally enable emittance from the red diode when the blue diode is not in use, to augment the red light delivery described below (for plant health via the support of photosynthesis). If red light is emitted from the robots at certain intervals, then the red diode of the RGB LED should have peak emission at approximately λ_{max} = 625–650 nm, with no critical wavelengths overlapping the green band (i.e., below 550 nm) or the far-red band (i.e., above 700 nm). The intensity level selected for red diodes in the robots should not produce heat levels higher than those of the blue diodes.

d) Include hardware that enables local cues between robots, for example a photoresistor (i.e., light-dependent resistor or LDR) facing each neighboring robot to monitor their light emittance status. Alternatively, the status of local neighbors can be communicated via WLAN.

e) Include hardware to dissipate heat, as required by the conditions of the selected blue diodes and the utilized robot enclosure. Heat dissipation can be executed by a combination of aluminum heatsinks, vents in the robot's case enclosure, and fans. (For example, If

aluminum heatsinks are mounted to the individual LEDs, but they still overheat in the robot enclosure, a fan can be added in a location with suitable airflow between the heatsinks and vents.) If there is any chance that the LEDs or other components could overheat during an experiment, then unconditionally include a fan activated by a digital temperature sensor on the single-board computer or supplemental PCB.

f) Organize the robot components such that the relevant directions are uniformly serviced. Ensure the blue diodes distribute an equivalent light intensity to each of the directions from which plants may approach, and that likewise the IR-proximity sensors are comparably positioned for their respective approaching growth directions. If including photoresistors for local communication, ensure they are equivalently positioned for each direction facing a neighboring robot in the setup.

Step 5: Mechanics. Integrate the robots into a set of modular mechanical supports that hold the robots in position (as needed) and serve as climbing scaffold for the plants, restricting the plants' likely average growth trajectories. The robots can optionally serve as supplementary mechanical joints between the supports, positioned such that they intersect the plant growth trajectories.

a) Minimize the size of the robot, such that it can be reliably surpassed by an unsupported growing tip of the selected plant species. Reducing robot size to the greatest extent possible will optimally increase experiment speed.

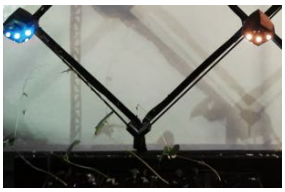
b) Shape the body of the robot to be as unobtrusive to plant growth as possible, in the event that a growing tip needs to incrementally navigate around the robot. Given the helical trajectory of circumnutation in twining plant species, recommended shapes will be rounded, faceted, domed, or otherwise absent of especially sharp protrusions or acute indentations.

c) Select a material and profile (i.e., shape of cross-section) for the mechanical supports, such that the selected plant species can effectively climb it. For *P. vulgaris*, select for instance a wooden rod with circular profile of a diameter roughly 8 mm. The mechanical supports also need to be structurally stiff enough to support the plants and robots within the setup. They can be augmented by supplemental structural support as needed (e.g., a transparent acrylic sheet behind the setup).

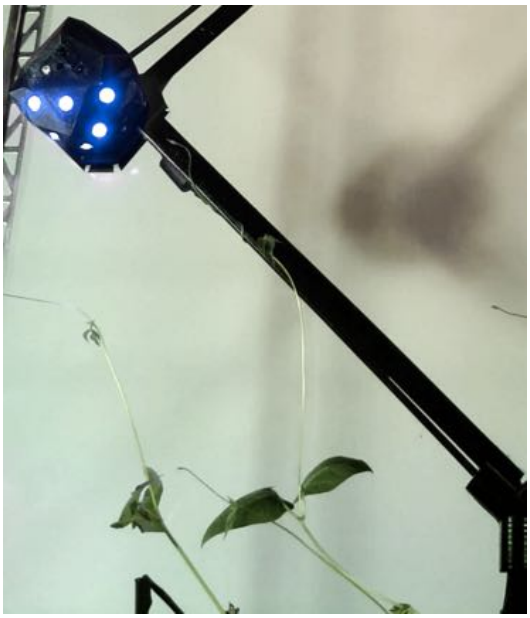
d) On each robot include attachment points to anchor the specified mechanical supports. Include enough to attach support for each direction by which a plant may approach or depart a robot.

e) Arrange the mechanical supports roughly in a regular grid pattern, uniformly diagonal with an angle of inclination at roughly 45°or steeper. The lengths of the supports and the distances between supports—which define the distances between robots and the size of a cell in the diagonal grid—should also be roughly uniform, depending on the objectives of the overall experiment setup. The minimum exposed distance of the scaffold between robots is 30 cm, to allow sufficient room for the climbing plants to attach after exploring the area in their unsupported condition, or for them to adequately explore a denser scaffold condition. The preferred exposed length is 40 cm or more, to allow some buffer for statistically extreme cases of plant attachment.

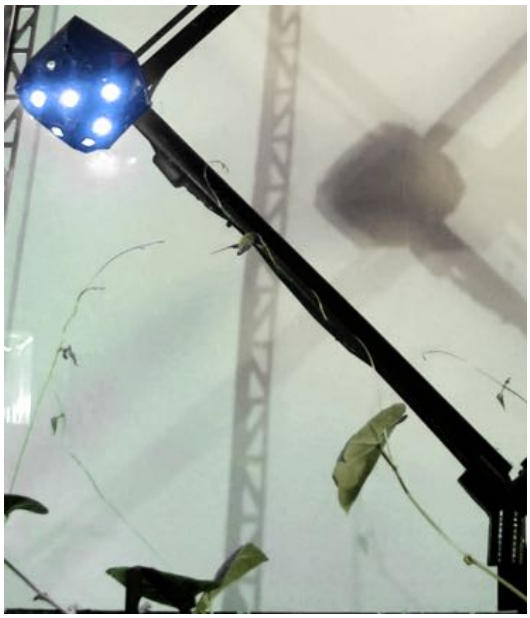
(a) Left *guiding* #1



(b) Left *guiding* #2



(c) Left *guiding* #1, closeup of (a). One week of growth; one plant attached.



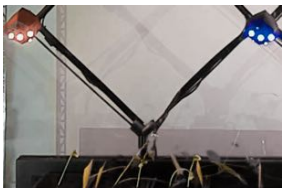
(d) Left *guiding* #2, closeup of (b). One week of growth; one plant attached.

Figure 48: Guiding plant growth to the **left-hand** robot. Frames of results from the verification experiments, which test the ability of the robotic nodes' *guiding* state to reliably steer the plant through a binary decision, to reach the correct rod and target.

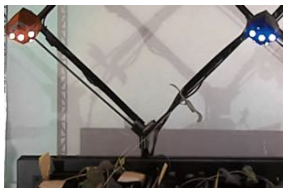
5.1 Verification experiments of single plant decisions

These experiments follow the hardware, software and overall experiment setup defined in *D1.2 Evaluation of mechatronics prototype of the robotic symbiont including supporting software*. There are two programmed states for the robot nodes: 1) the *guiding* state, in which a robot emits only blue light to attract the plants by triggering their phototropic response, and detects the proximity of approaching plant tips, and 2) the *feeding* state, in which a robot emits only red light, supporting the plants' photosynthesis without triggering any phototropic response.

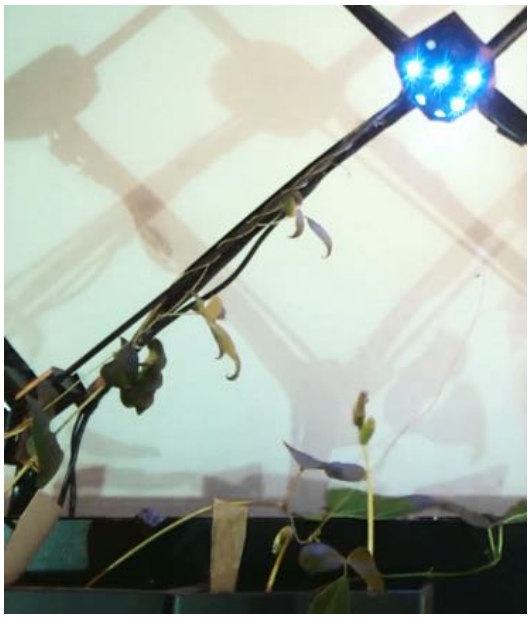
To further test the bio-hybrid system described in previous deliverables, we run a small set of verification experiments in addition to the original control experiments (where all robotic nodes were set constantly to the *feeding* state, to test the growth and motion behavior of the plants in conditions without triggered phototropism), and the original predefined pattern experiments (testing the ability of the robotic nodes and bio-hybrid setup to correctly shape plant growth in a full-length experiment, into a predefined pattern on a 180 cm diagrid). These new experiments are verification experiments, where we verify the ability of the robotic nodes' *guiding* state to guide



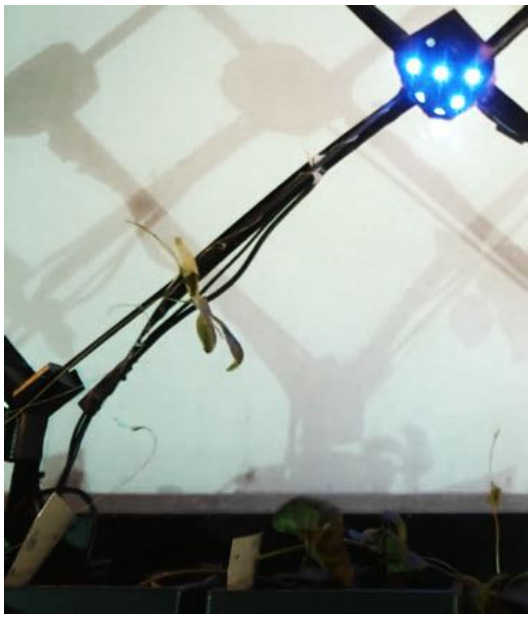
(a) Right *guiding* #1



(b) Right *guiding* #2



(c) Right *guiding* #1, closeup of (a). Two weeks of growth; two plants attached.



(d) Right *guiding* #2, closeup of (b). Three weeks of growth; one plant attached.

Figure 49: Guiding plant growth to the **right-hand** robot. Frames of results from the verification experiments, which test the ability of the robotic nodes' *guiding* state to reliably steer the plant through a binary decision, to reach the correct rod and target.

plant growth to the correct target—in a binary decision between left and right—by triggering the plants' phototropic response to blue light. The new verification experiments maintain the same experiment setup conditions used in the original control experiments and predefined pattern experiments, including the same minimum of four plants per experiment, and the same hardware and mechanical elements with uniform positions and orientations maintained.

The verification experiments are conducted over approximately seven weeks in total, and each contains a minimum of four plants. These experiments test the ability of the distributed mechatronic system to guide the plant toward a specified target, by steering the plant's decision-making at a given junction. This is tested in two experiments per target direction at the first diagrid junction. In each experiment, the targeted node—left or right—is set to the *guiding* state, and the opposing node is set to the *feeding* state. An experiment is considered successful if all attached plant tips find the correct rod, and only the *guiding* node detects an approaching plant. Another feature considered favorable is a high proportion of unsupported shoots growing with bias to the *guiding* node.

Verification experiment results:

1. **Figure 48: two left-decision experiments.** In these left *guiding* experiments, the node left of the junction is set to the *guiding* state, with the opposing set to *feeding* (a,b). Each frame shows the condition of growth just before the *guiding* node detects the approaching plant. In both experiments (see c,d), at least one plant attaches to the correct rod (i.e., the left-hand rod). None of the plants attaches to the incorrect rod. In further evidence of the nodes' effects, the unsupported plants also generally display growth biased in the correct direction, toward the left-hand node.

2. **Figure 49: two right-decision experiments.** In the right *guiding* experiments, the node right of the junction is set to the *guiding* state, with the opposing set to *feeding* (a,b). Each frame shows the condition of growth just before the *guiding* node detects the approaching plant. In both experiments (see c,d), at least one plant attaches to the correct rod (i.e., the right-hand rod). None of the plants attaches to the incorrect rod. In further evidence of the nodes' effects, the unsupported plants also generally display growth biased in the correct direction, toward the right-hand node.

There are two verification experiments per target (i.e., the node set to *guiding*), all occurring at the first diagrid junction. They each run continuously, for 13 days on average. In each of the four experiments, a plant successfully chooses and attaches to the correct rod, climbing it until reaching the target, see Figs. 48 and 49. In over 90% of the unsupported shoots, the typical circumnutation motion of winding is pronouncedly biased to the target. This results in consistent tilting of the upright stems toward the target, in areas where tissues have stiffened. In each experiment, the plant with stem angle and location most similar to that of the correct rod (i.e., the rod connected to the *guiding* node) is the first to attach. In each case, that first *leading* plant continues to climb the rod until it reaches the target. In one experiment, a second plant also attaches to the correct rod. The experiments are stopped once a plant reaches the target. Out of over 20 total plants in all verification experiments, none of the plants attaches to the incorrect rod (i.e., the rod leading to the *feeding* node). Taken together, these results suggest that the robotic nodes are capable of reliably steering the plant through a binary decision, until it reaches the specified target.

5.2 Final generation of decentralized hardware

Successful plant shaping experiments are reported in *D1.2 Evaluation of mechatronics prototype of the robotic symbiont including supporting software* and above. They are conducted on a wooden diagrid structure using a decentralized system of plant shaping robotic nodes [4]. However, according to the consortium's vision for the project, we follow braid as a methodology for scaffold construction, and open up to setups where plants may choose growth paths on a continuous field instead of on a few discrete rods. This motivates the development of a new and final generation, in which we also substantially improve efficiency and reliability.

5.2.1. Primary robot nodes for growth attraction

The final version of the robotic node (see Fig. 50) is cylindrical with approximately 4.25 cm radius and 4.5 cm height, substantially smaller than the previous version (cf. [4]). (The context for the robotic nodes—the overall setup for plant interaction, including mechanics—is described in *D2.4 Report on the final algorithms and plant-affection of bio-hybrid organism.*)

For surrounding light intensity sensing, the robot is equipped with six GL55 photoresistors (cf. four in the previous version [4]), and a TCS34725 RGB color sensor for the first time. The task

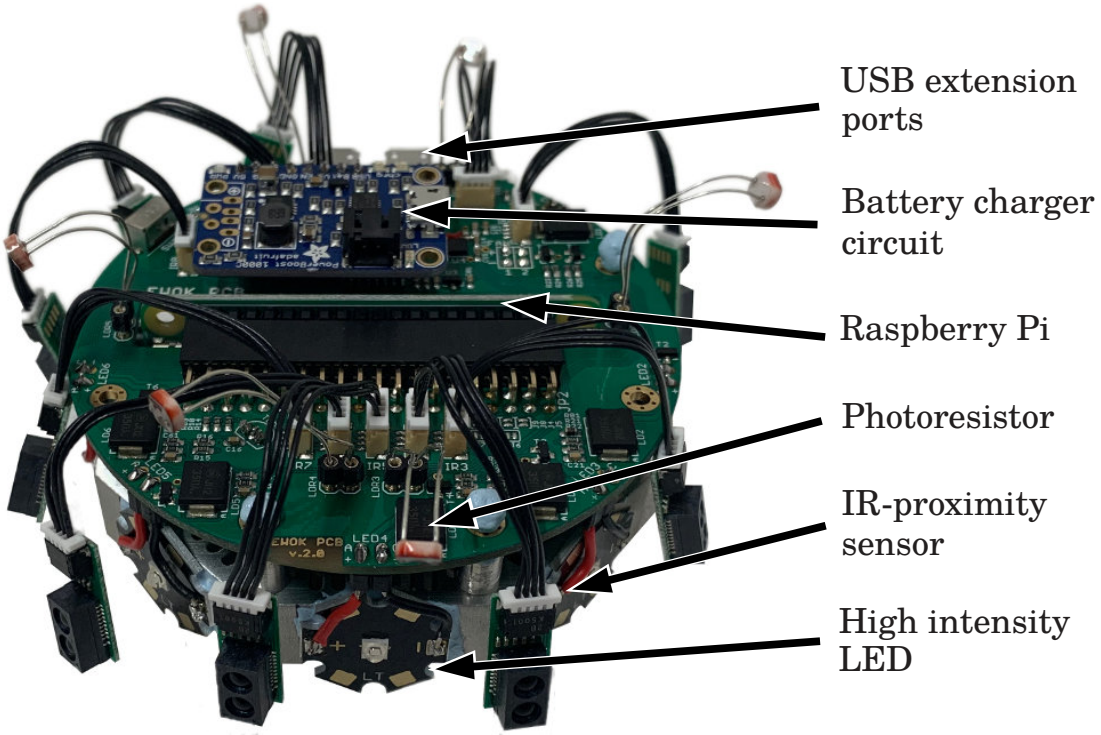


Figure 50: The final version of the plant shaping robotic node without 3D-printed casing, top view.

of the photoresistors is to sense the emmitance status of neighboring robotic nodes, enabling collective decision-making strategies based on this cue. The robotic nodes can then execute growth patterns based on locally sensed information. In this version there are six photoresistors to support a more circumferential view of light conditions, as the relative positions of neighboring nodes are unconstrained, in contrast to the previous version where they were on constrained grid positions. The RGB color sensor is included to enable the monitoring of photosynthetic light spectra, to generally support the maintenance of plant health.

For plant proximity detection, the robot is equipped with ten GP2Y0E02B IR-proximity sensors (cf. three in the previous version [4]), that provide a full circumferential view. In the previous version, plants could only approach the node from specific discrete locations (two in a 2D setup or three in a 3D setup). In this version, plants may approach from any direction in a continuous roughly 2D plane, so a more circumferential view is required. The field of view on an individual IR sensor is extremely narrow, so the IR sensors are placed as densely as possible around the node circumference, with ten sensors being the achievable maximum for this node size (see Fig. 52 for a view of mechanical positioning around the edge).

For stimuli-driven plant actuation, the robot is equipped with six 1 W blue LEDs to attract climbing plants by triggering their natural phototropism. The previous version used 3 W RGB LEDs with off-the-shelf microcontrollers for heat management. This solution turned out to be not ideal, getting very hot quickly and often blinking in the setup as it turned on and off to stop overheating. In this version of the robot we have therefore upgraded to a more efficient and reliable solution. The main light spectra triggering phototropism are UV and blue light, thus

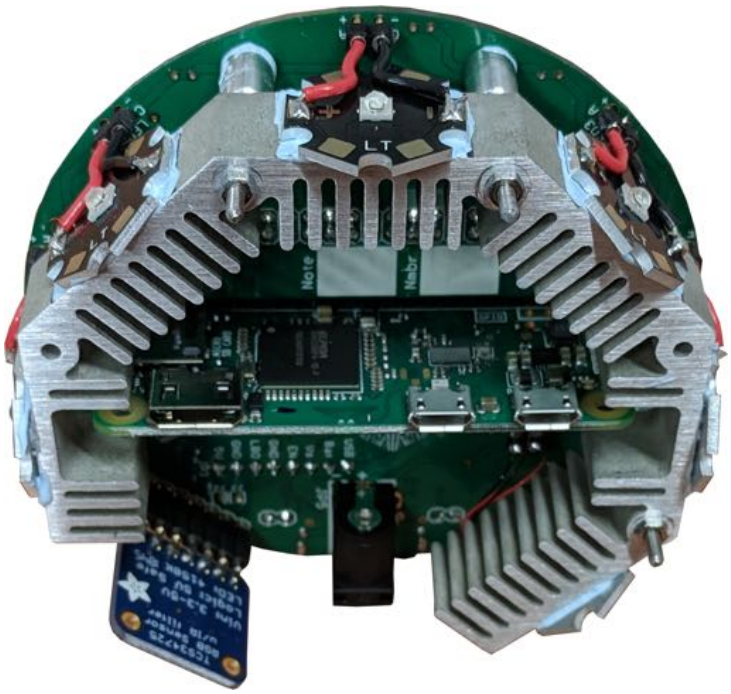


Figure 51: The final version of the plant shaping robotic node without 3D-printed casing, bottom view.

RGB LEDs are not required because only the blue diode is used for directional stimuli. Here we therefore use blue LEDs, moving the task of red light provision for photosynthesis to be outside of the robot, in the general lab setup instead. This is suitable because red light conditions do not shape plants, and therefore do not need control, ideally being uniform throughout the setup and over time. The blue LEDs in the robots are 1 W instead of 3 W because the 1 W version is significantly more efficient, providing nearly the same lumens as the 3 W. Temperature sensing and a fail-safe for overheating are located centrally on the PCB rather than at each LED.

Heat dissipation is managed by a custom waterjet cut aluminum heatsink (see Fig. 53) and two 5 V fans. The LEDs each come off-the-shelf with a small heatsink. These are affixed to the six faces of the custom heatsink using epoxy that is thermally conductive and electrically insulating. The heatsink is shaped as a donut, because the LEDs should be positioned around the circumference of the node, and the Raspberry Pi board is located in the center. The LEDs and heatsink do not make a full circle, as plants grow generally upwards and will not approach the node from above, so the upper edge of the node does not require an LED. The custom heatsink is flanged to maximize effectiveness within the available space (see Fig. 51). The width of the flanges, width of the gaps between them, and radii of the corners are all defined by the dimensional constraints of the water stream size and machine path flexibility of the waterjet cutting manufacturing process. As many flanges as possible are included within these constraints. Waterjet cutting is chosen as manufaturing technique because it does not involve heat or sintering, allowing material to be heat-treated before cutting, which is ideal for small-bath manufacturing of aluminum heatsinks. Because thick sheet material is used and LEDs (which should be perpendicular to the custom PCB) are mounted on the cut edge, high precision low-taper waterjet is used. Flanges are not included where they would conflict with the Raspberry Pi board,

and flange length is restricted to allow space for the 5 V fans, positioned on each side of the Raspberry Pi board. Screw holes are included in the heatsink, and bolts with aluminum spacers mechanically fix the heatsink to the custom PCB. As the PCB also generates quite some heat, aluminum spacers are included between, to transfer heat. The overall heat dissipation strategy enables continuous LED operation at 100% intensity (cf. 65% in the previous version [4]).

Figure 52: 3D-printed case cross-section showing the mechanical anchoring of an IR-proximity sensor around the circumference of the robot node. The ten IR sensors are the primary components requiring mechanical anchoring in the case, as other components are mechanically affixed to the PCB and heatsink.

Figure 53: Custom waterjet cut aluminum heatsink. Six LEDs are affixed to the six faces of the heatsink using thermal adhesive.

The robotic node has a power control system which can automatically switch from the operation on the external power supply to the internal battery. To decrease the time of development and debugging the integrated solution PowerBoost 1000C used. It is based at the two IC: TPS61090 - DC-DC Boost Converter for one-cell Li-Po battery which convert the 3.7 V battery voltage to 5 V DC required for electronics operation and MCP73871—stand-alone system load sharing and Li-Po battery charge management controller. With a PowerBoost 1000C and a 3.7 V

2000 mAh LiPo battery, in case of a power outage the robot can still operate for a time period of approximately six hours without plant actuation. This is an important fail-safe feature that protects the robot's hardware against sudden power cuts and allows for long-term plant-robot experiments without interruption. The three micro USB ports onboard can be used to enhance the robot capabilities by connecting it to external extension modules (e.g., growth repelling module, phytosensing system). The ports also support I^2C, GPIO serial communication with any Raspberry Pi based device (e.g., robot or sensor).

The Raspberry Pi Zero W and custom PCB are the kernels of the robotic node. The custom PCB (see Fig. 54) is a custom interface board between the Raspberry Pi and the sensors, LED actuators, power, etc, and provides the Raspberry Pi with control of all implemented functionalities. See Table 2, for a full components list.

Table 2: Final generation plant shaping robotic node component list.

Name	Functionality	Count
	Sensors	
GL55 Photoresistor	Ambient light intensity sensor	6
10 kΩ Thermistor	Temperature sensor at the PCB	2
TCS34725	RGB color and clear light intensity sensor	1
Sharp GP2Y0E02B	IR-proximity sensor	10
DS18B20	Digital temperature sensor	1
LSM9DS1	Accelerometer, magnetometer, and gyroscope	1
	Actuators	
CREE XP-E2	High intensity blue LED	6
	Others	
Raspberry Pi Zero WH	Central control and processing unit	1
Custom PCB Interface Board	An interface to the sensors and actuators	1
5 V fan	Internal cooling device	2
PowerBoost 1000C	Power management, battery charger circuit	1
3.7 V 2000mAh LiPo battery	Backup power source	1

5.2.2. Extension system for growth repelling

One extension module that can be connected to a main node via micro USB, and powered and controlled by that main node, is a spray module for growth repelling. The task of repelling growth can extend the flexibility of the overall setup by allowing the addition of negative feedback into what has so far been an exclusively positive feedback system. This repelling module (see Fig. 55) is tasked with spraying auxin inhibitor Toprex 375 SC (Syngenta), as described in *D2.4 Report on the final algorithms and plant-affection of bio-hybrid organism*. Toprex is stable in water solution, so can be kept in detachable refillable silicon bottles on the module. The main element of the module is a DC motor driven digital spray head extracted from commercial product automatic spray bottles. The spray head is mechanically fixed in a 3D printed case, which includes continuous screw threads for the attachment of the bottle. The 3D-printed surface of the threaded attachment point is coated in silicon for a watertight barrier. A high-precision rubber gasket is set into the 3D printed case where the tube passes through from the spray head to the bottle, creating a watertight seal.

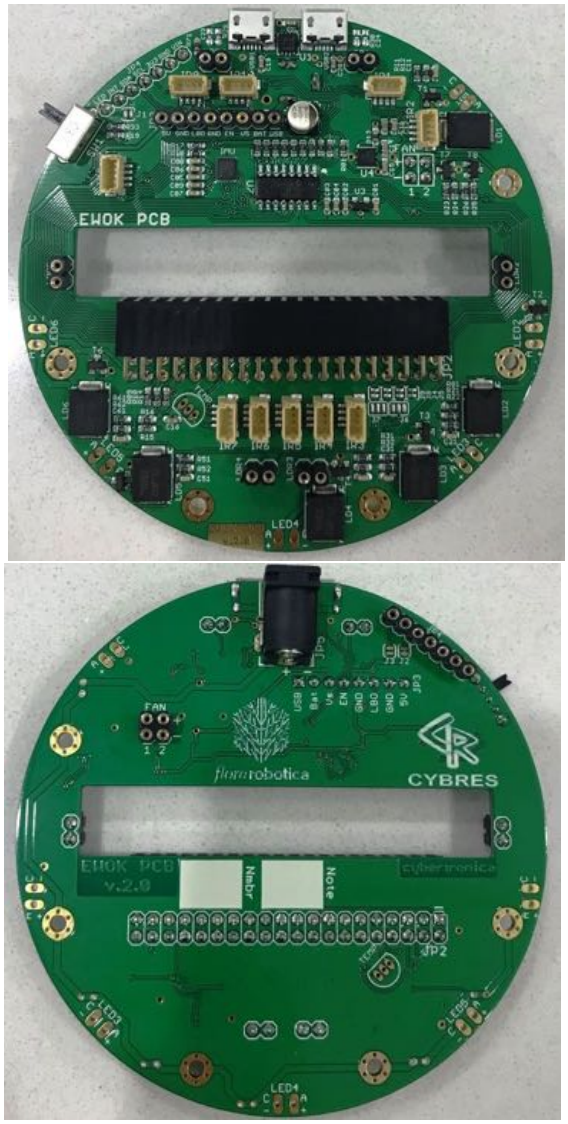


Figure 54: Custom PCB - the robotic node interface board (top and bottom view).

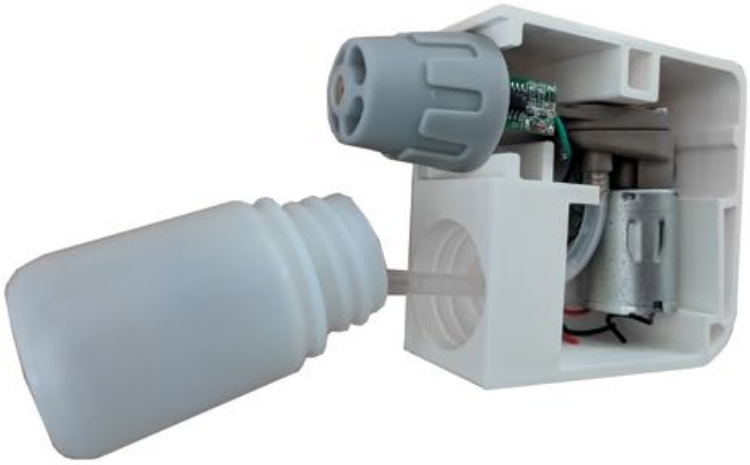


Figure 55: Extension module for growth repelling by controlled spraying of auxin inhibitor.

5.3 Conclusion

In this section we introduced an experimental protocol for plant shaping experiments as well as the robotics nodes for plant shaping. The nodes have been developed throughout the project and they and the experimental protocol are at this stage at a mature level.

6 Conclusion

This concludes deliverable *D2.4 Evaluation of the robotic symbiont*. The overall objectives of work package WP 1 as stated in the Description of Work are:

> Objectives: This work package has three objectives, which are the development of: 1) mechatronic basis for *flora robotica*; 2) interaction mechanisms between the robotic and biological element of *flora robotica*; 3) software abstraction that allows efficient programming and experimentation with *flora robotica*

In the beginning of the project we developed a forest worth of exploratory prototypes which have been crucial for our understanding of how robots and plants can work together. In particular, the completely unanticipated and carrying elements of braids was discovered in the first year of the project and could not have been achieved without this exploratory work. After this initial exploration we have focused and can document with this deliverable document that we have met the objectives we set out to achieve.

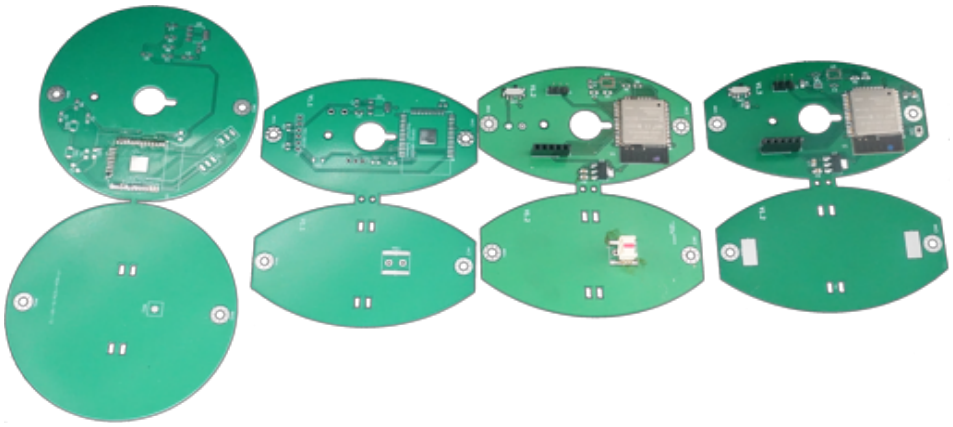


Figure 56: The iterations of the node actuator (from left: version 0 to 3)

Appendix: Detailed development iterations of the Actuator Node

Version 0: Version zero was made with a PCB (printed circuit board) thickness of 1.6 mm and general test layout to find bugs.

Version 1: In version one a couple of improvements was made, the board was redesigned to reduce weight and by using 0.8 mm instead of 1.6 mm thick PCB.

Version 2: In the second version everything looked to function we moved from using ESProg hardware design where GPIO0 is used, to shorting pin 25 to ground with a switch (see Fig. 22), the switch changes the esp from normal execution mode to serial bootloader. We also implemented the possibility for a 9 DOF (9 degrees of freedom) chip. A problem arose by switching from the 3.3 V supply from the programmer to a 5 V power-supply, when using the 5 V there was a problem with getting the esp's to boot, by switching the power-supply off and on we were able to boot them on one by one. The cause of the problem was that the enable pin was floating when multiple boards was put in series.

Version 3: The final version we solved the booting problem by adding a capacitor and implementing reset button for doing hard-reset.

References

[1] E. Artin. Theory of braids. *Annals of Mathematics*, 48(1):101–126, 1947.

[2] Christian Kassel. *Braid groups*. Graduate texts in mathematics ; 247. Springer, New York, NY, 2008. ISBN 9780387338415hbk.

[3] Y. Kyosev. *Braiding Technology for Textiles: Principles, Design and Processes*. Woodhead Publishing Series in Textiles. Elsevier Science, 2014. ISBN 9780857099211. URL https://books.google.dk/books?id=K2F7AwAAQBAJ.

[4] Mostafa Wahby, Mary Katherine Heinrich, Daniel Nicolas Hofstadler, Ewald Neufeld, Igor Kuksin, Payam Zahadat, Thomas Schmickl, Phil Ayres, and Heiko Hamann. Autonomously shaping natural climbing plants: a bio-hybrid approach. *Royal Society open science*, 5(10): 180296, 2018. doi: 10.1098/rsos.180296. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsos.180296.

[5] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.